运维工程师面试题

1、什么是运维？什么是游戏运维？

1）运维是指大型组织已经建立好的网络软硬件的维护，就是要保证业务的上线与运作的正常，
在他运转的过程中，对他进行维护，他集合了网络、系统、数据库、开发、安全、监控于一身的技术
运维又包括很多种，有DBA运维、网站运维、虚拟化运维、监控运维、游戏运维等等

2）游戏运维又有分工，分为开发运维、应用运维（业务运维）和系统运维
开发运维：是给应用运维开发运维工具和运维平台的
应用运维：是给业务上线、维护和做故障排除的，用开发运维开发出来的工具给业务上线、维护、做故障排查
系统运维：是给应用运维提供业务上的基础设施，比如：系统、网络、监控、硬件等等

总结：开发运维和系统运维给应用运维提供了“工具”和“基础设施”上的支撑
开发运维、应用运维和系统运维他们的工作是环环相扣的

2、在工作中，运维人员经常需要跟运营人员打交道，请问运营人员是做什么工作的？

游戏运营要做的一个事情除了协调工作以外
还需要与各平台沟通，做好开服的时间、开服数、用户导量、活动等计划

3、现在给你三百台服务器，你怎么对他们进行管理？

管理3百台服务器的方式：
1）设定跳板机，使用统一账号登录，便于安全与登录的考量。
2）使用salt、ansible、puppet进行系统的统一调度与配置的统一管理。
3）建立简单的服务器的系统、配置、应用的cmdb信息管理。便于查阅每台服务器上的各种信息记录。

4、简述raid0 raid1 raid5 三种工作模式的工作原理及特点

RAID，可以把硬盘整合成一个大磁盘，还可以在大磁盘上再分区，放数据
还有一个大功能，多块盘放在一起可以有冗余（备份）
RAID整合方式有很多，常用的：0 1 5 10

RAID 0，可以是一块盘和N个盘组合
其优点读写快，是RAID中最好的
缺点：没有冗余，一块坏了数据就全没有了

RAID 1，只能2块盘，盘的大小可以不一样，以小的为准
10G+10G只有10G，另一个做备份。它有100%的冗余，缺点：浪费资源，成本高

RAID 5 ，3块盘，容量计算10*（n-1）,损失一块盘
特点，读写性能一般，读还好一点，写不好

冗余从好到坏：RAID1 RAID10 RAID 5 RAID0
性能从好到坏：RAID0 RAID10 RAID5 RAID1
成本从低到高：RAID0 RAID5 RAID1 RAID10

单台服务器：很重要盘不多，系统盘，RAID1
数据库服务器：主库：RAID10 从库 RAID5\RAID0（为了维护成本，RAID10）
WEB服务器，如果没有太多的数据的话，RAID5,RAID0（单盘）
有多台，监控、应用服务器，RAID0 RAID5

我们会根据数据的存储和访问的需求，去匹配对应的RAID级别

5、LVS、Nginx、HAproxy有什么区别？工作中你怎么选择？

LVS：是基于四层的转发
HAproxy：是基于四层和七层的转发，是专业的代理服务器
Nginx：是WEB服务器，缓存服务器，又是反向代理服务器，可以做七层的转发

区别：LVS由于是基于四层的转发所以只能做端口的转发、而基于URL的、基于目录的这种转发LVS就做不了

工作选择：HAproxy和Nginx由于可以做七层的转发，所以URL和目录的转发都可以做，在很大并发量的时候我们就要选择LVS，像中小型公司的话并发量没那么大，选择HAproxy或者Nginx足已，由于HAproxy由是专业的代理服务器，配置简单，所以中小型企业推荐使用HAproxy

6、Squid、Varinsh和Nginx有什么区别，工作中你怎么选择？

Squid、Varinsh和Nginx都是代理服务器

什么是代理服务器：

能代替用户去访问公网，并且能把访问到的数据缓存到服务器本地，等用户下次再访问相同的资源的时候，代理服务器直接从本地回应给用户，当本地没有的时候，我代替你去访问公网，我接收你的请求，我先在我自已的本地缓存找，如果我本地缓存有，我直接从我本地的缓存里回复你如果我在我本地没有找到你要访问的缓存的数据，那么代理服务器就会代替你去访问公网

区别：

1）Nginx本来是反向代理/web服务器，用了插件可以做做这个副业
但是本身不支持特性挺多，只能缓存静态文件

2）从这些功能上。varnish和squid是专业的cache服务，而nginx这些是第三方模块完成

3）varnish本身的技术上优势要高于squid，它采用了可视化页面缓存技术
在内存的利用上，Varnish比Squid具有优势，性能要比Squid高。
还有强大的通过Varnish管理端口，可以使用正则表达式快速、批量地清除部分缓存
它是内存缓存，速度一流，但是内存缓存也限制了其容量，缓存页面和图片一般是挺好的

4）squid的优势在于完整的庞大的cache技术资料，和很多的应用生产环境

工作中选择：

要做cache服务的话，我们肯定是要选择专业的cache服务，优先选择squid或者varnish。

7、Tomcat和Resin有什么区别，工作中你怎么选择？

区别：Tomcat用户数多，可参考文档多，Resin用户数少，可考虑文档少，最主要区别则是Tomcat是标准的java容器，不过性能方面比resin的要差一些，但稳定性和java程序的兼容性，应该是比resin的要好

工作中选择：现在大公司都是用resin，追求性能；而中小型公司都是用Tomcat，追求稳定和程序的兼容

8、什么是中间件？什么是jdk?

中间件是一种独立的系统软件或服务程序，分布式应用软件借助这种软件在不同的技术之间共享资源

中间件位于客户机/ 服务器的操作系统之上，管理计算机资源和网络通讯是连接两个独立应用程序或独立系统的软件。相连接的系统，即使它们具有不同的接口

但通过中间件相互之间仍能交换信息。执行中间件的一个关键途径是信息传递通过中间件，应用程序可以工作于多平台或OS环境。

jdk：jdk是Java的开发工具包，它是一种用于构建在 Java 平台上发布的应用程序、applet 和组件的开发环境

9、讲述一下Tomcat8005、8009、8080三个端口的含义?

8005==》 关闭时使用
8009==》 为AJP端口，即容器使用，如Apache能通过AJP协议访问Tomcat的8009端口
8080==》 一般应用使用

10、什么叫CDN?

即内容分发网络，其目的是通过在现有的Internet中增加一层新的网络架构，将网站的内容发布到最接近用户的网络边缘，使用户可就近取得所需的内容，提高用户访问网站的速度。

11、什么叫网站灰度发布?

灰度发布是指在黑与白之间，能够平滑过渡的一种发布方式
AB test就是一种灰度发布方式，让一部用户继续用A，一部分用户开始用B
如果用户对B没有什么反对意见，那么逐步扩大范围，把所有用户都迁移到B上面 来
灰度发布可以保证整体系统的稳定，在初始灰度的时候就可以发现、调整问题，以保证其影响度

12、简述DNS进行域名解析的过程?

用户要访问http://www.baidu.com，会先找本机的host文件，再找本地设置的DNS服务器，如果也没有的话，就去网络中找根服务器，根服务器反馈结果，说只能提供一级域名服务器.cn，就去找一级域名服务器，一级域名服务器说只能提供二级域名服务器.com.cn,就去找二级域名服务器，二级域服务器只能提供三级域名服务器.http://baidu.com.cn，就去找三级域名服务器，三级域名服务器正好有这个网站http://www.baidu.com，然后发给请求的服务器，保存一份之后，再发给客户端

13、RabbitMQ是什么东西?

RabbitMQ也就是消息队列中间件，消息中间件是在消息的传息过程中保存消息的容器
消息中间件再将消息从它的源中到它的目标中标时充当中间人的作用
队列的主要目的是提供路由并保证消息的传递；如果发送消息时接收者不可用
消息队列不会保留消息，直到可以成功地传递为止，当然，消息队列保存消息也是有期限地

14、讲一下Keepalived的工作原理?

在一个虚拟路由器中，只有作为MASTER的VRRP路由器会一直发送VRRP通告信息,
BACKUP不会抢占MASTER，除非它的优先级更高。当MASTER不可用时(BACKUP收不到通告信息)
多台BACKUP中优先级最高的这台会被抢占为MASTER。这种抢占是非常快速的(<1s)，以保证服务的连续性
由于安全性考虑，VRRP包使用了加密协议进行加密。BACKUP不会发送通告信息，只会接收通告信息

15、讲述一下LVS三种模式的工作过程?

LVS 有三种负载均衡的模式，分别是VS/NAT（nat 模式） VS/DR(路由模式) VS/TUN（隧道模式）

一、NAT模式（VS-NAT）

原理：就是把客户端发来的数据包的IP头的目的地址，在负载均衡器上换成其中一台RS的IP地址，并发至此RS来处理,RS处理完后把数据交给负载均衡器,负载均衡器再把数据包原IP地址改为自己的IP，将目的地址改为客户端IP地址即可期间,无论是进来的流量,还是出去的流量,都必须经过负载均衡器

优点：集群中的物理服务器可以使用任何支持TCP/IP操作系统，只有负载均衡器需要一个合法的IP地址

缺点：扩展性有限。当服务器节点（普通PC服务器）增长过多时,负载均衡器将成为整个系统的瓶颈，因为所有的请求包和应答包的流向都经过负载均衡器。当服务器节点过多时，大量的数据包都交汇在负载均衡器那，速度就会变慢!

二、IP隧道模式（VS-TUN）

原理：首先要知道，互联网上的大多Internet服务的请求包很短小，而应答包通常很大，那么隧道模式就是，把客户端发来的数据包，封装一个新的IP头标记(仅目的IP)发给RS，RS收到后,先把数据包的头解开,还原数据包,处理后,直接返回给客户端,不需要再经过负载均衡器。注意,由于RS需要对负载均衡器发过来的数据包进行还原,所以说必须支持IPTUNNEL协议，所以,在RS的内核中,必须编译支持IPTUNNEL这个选项

优点：负载均衡器只负责将请求包分发给后端节点服务器，而RS将应答包直接发给用户，所以，减少了负载均衡器的大量数据流动，负载均衡器不再是系统的瓶颈，就能处理很巨大的请求量，这种方式，一台负载均衡器能够为很多RS进行分发。而且跑在公网上就能进行不同地域的分发。

缺点：隧道模式的RS节点需要合法IP，这种方式需要所有的服务器支持"IP Tunneling"(IP Encapsulation)协议，服务器可能只局限在部分Linux系统上

三、直接路由模式（VS-DR）

原理：负载均衡器和RS都使用同一个IP对外服务但只有DR对ARP请求进行响应，所有RS对本身这个IP的ARP请求保持静默也就是说,网关会把对这个服务IP的请求全部定向给DR，而DR收到数据包后根据调度算法,找出对应的RS,把目的MAC地址改为RS的MAC（因为IP一致），并将请求分发给这台RS这时RS收到这个数据包,处理完成之后，由于IP一致，可以直接将数据返给客户，则等于直接从客户端收到这个数据包无异,处理后直接返回给客户端，由于负载均衡器要对二层包头进行改换,所以负载均衡器和RS之间必须在一个广播域，也可以简单的理解为在同一台交换机上

优点：和TUN（隧道模式）一样，负载均衡器也只是分发请求，应答包通过单独的路由方法返回给客户端，与VS-TUN相比，VS-DR这种实现方式不需要隧道结构，因此可以使用大多数操作系统做为物理服务器。

缺点：（不能说缺点，只能说是不足）要求负载均衡器的网卡必须与物理网卡在一个物理段上。

16、mysql的innodb如何定位锁问题，mysql如何减少主从复制延迟?

mysql的innodb如何定位锁问题:

在使用 show engine innodb status检查引擎状态时，发现了死锁问题

在5.5中，information_schema 库中增加了三个关于锁的表（MEMORY引擎）

```
innodb_trx         ## 当前运行的所有事务

innodb_locks     ## 当前出现的锁

innodb_lock_waits  ## 锁等待的对应关系
```

mysql如何减少主从复制延迟:

如果延迟比较大，就先确认以下几个因素：

1.从库硬件比主库差，导致复制延迟

2.主从复制单线程，如果主库写并发太大，来不及传送到从库就会导致延迟。更高版本的mysql可以支持多线程复制

3.慢SQL语句过多

4.网络延迟

5.master负载：主库读写压力大，导致复制延迟，架构的前端要加buffer及缓存层

6.slave负载：一般的做法是，使用多台slave来分摊读请求，再从这些slave中取一台专用的服务器只作为备份用，不进行其他任何操作.另外， 2个可以减少延迟的参数:

```
-slave-net-timeout=seconds 单位为秒 默认设置为 3600秒

#参数含义：当slave从主数据库读取log数据失败后，等待多久重新建立连接并获取数据

-master-connect-retry=seconds 单位为秒 默认设置为 60秒

#参数含义：当重新建立主从连接时，如果连接建立失败，间隔多久后重试
```

通常配置以上2个参数可以减少网络问题导致的主从数据同步延迟

MySQL数据库主从同步延迟解决方案

最简单的减少slave同步延时的方案就是在架构上做优化，尽量让主库的DDL快速执行

还有就是主库是写，对数据安全性较高，比如sync_binlog=1，innodb_flush_log_at_trx_commit = 1 之类的设置，而slave则不需要这么高的数据安全，完全可以讲sync_binlog设置为0或者关闭binlog

innodb_flushlog也可以设置为0来提高sql的执行效率。另外就是使用比主库更好的硬件设备作为slave

17、如何重置mysql root密码?

一、 在已知MYSQL数据库的ROOT用户密码的情况下，修改密码的方法：

在SHELL环境下，使用mysqladmin命令设置：

```
mysqladmin -u root -p password “新密码”   回车后要求输入旧密码
```

在mysql>环境中,使用update命令，直接更新mysql库user表的数据：

```
Update  mysql.user  set  password=password(‘新密码’)  where  user=’root’;

flush   privileges;
```

注意：mysql语句要以分号”；”结束在mysql>环境中，使用grant命令，修改root用户的授权权限。

```
grant  all  on  *.*  to   root@'localhost'  identified  by  '新密码';
```

二、 如忘记了mysql数据库的ROOT用户的密码，又如何做呢？方法如下：

关闭当前运行的mysqld服务程序：service mysqld stop（要先将mysqld添加为系统服务）

使用mysqld_safe脚本以安全模式（不加载授权表）启动mysqld 服务

```
/usr/local/mysql/bin/mysqld_safe  --skip-grant-table  &
```

使用空密码的root用户登录数据库，重新设置ROOT用户的密码

```
#mysql  -u   root

Mysql> Update  mysql.user  set  password=password('新密码')  where  user='root';

Mysql> flush   privileges;
```

18、lvs/nginx/haproxy优缺点

Nginx的优点是：

1、工作在网络的7层之上，可以针对http应用做一些分流的策略，比如针对域名、目录结构

它的正则规则比HAProxy更为强大和灵活，这也是它目前广泛流行的主要原因之一

Nginx单凭这点可利用的场合就远多于LVS了。

2、Nginx对网络稳定性的依赖非常小，理论上能ping通就就能进行负载功能，这个也是它的优势之一

相反LVS对网络稳定性依赖比较大，这点本人深有体会；

3、Nginx安装和配置比较简单，测试起来比较方便，它基本能把错误用日志打印出来

LVS的配置、测试就要花比较长的时间了，LVS对网络依赖比较大。

4、可以承担高负载压力且稳定，在硬件不差的情况下一般能支撑几万次的并发量，负载度比LVS相对小些。

5、Nginx可以通过端口检测到服务器内部的故障，比如根据服务器处理网页返回的状态码、超时等等，并且会把返回错误的请求重新提交到另一个节点，不过其中缺点就是不支持url来检测。比如用户正在上传一个文件，而处理该上传的节点刚好在上传过程中出现故障，Nginx会把上传切到另一台服务器重新处理，而LVS就直接断掉了

如果是上传一个很大的文件或者很重要的文件的话，用户可能会因此而不满。

6、Nginx不仅仅是一款优秀的负载均衡器/反向代理软件，它同时也是功能强大的Web应用服务器

LNMP也是近几年非常流行的web架构，在高流量的环境中稳定性也很好。

7、Nginx现在作为Web反向加速缓存越来越成熟了，速度比传统的Squid服务器更快，可考虑用其作为反向代理加速器

8、Nginx可作为中层反向代理使用，这一层面Nginx基本上无对手，唯一可以对比Nginx的就只有lighttpd了

不过lighttpd目前还没有做到Nginx完全的功能，配置也不那么清晰易读，社区资料也远远没Nginx活跃

9、Nginx也可作为静态网页和图片服务器，这方面的性能也无对手。还有Nginx社区非常活跃，第三方模块也很多

Nginx的缺点是：

1、Nginx仅能支持http、https和Email协议，这样就在适用范围上面小些，这个是它的缺点

2、对后端服务器的健康检查，只支持通过端口来检测，不支持通过url来检测

不支持Session的直接保持，但能通过ip_hash来解决

LVS：使用Linux内核集群实现一个高性能、高可用的负载均衡服务器

它具有很好的可伸缩性（Scalability)、可靠性（Reliability)和可管理性（Manageability)

LVS的优点是：

1、抗负载能力强、是工作在网络4层之上仅作分发之用，没有流量的产生

这个特点也决定了它在负载均衡软件里的性能最强的，对内存和cpu资源消耗比较低

2、配置性比较低，这是一个缺点也是一个优点，因为没有可太多配置的东西

所以并不需要太多接触，大大减少了人为出错的几率

3、工作稳定，因为其本身抗负载能力很强，自身有完整的双机热备方案

如LVS+Keepalived，不过我们在项目实施中用得最多的还是LVS/DR+Keepalived

4、无流量，LVS只分发请求，而流量并不从它本身出去，这点保证了均衡器IO的性能不会收到大流量的影响。

5、应用范围较广，因为LVS工作在4层，所以它几乎可对所有应用做负载均衡，包括http、数据库、在线聊天室等

LVS的缺点是：

1、软件本身不支持正则表达式处理，不能做动静分离

而现在许多网站在这方面都有较强的需求，这个是Nginx/HAProxy+Keepalived的优势所在

2、如果是网站应用比较庞大的话，LVS/DR+Keepalived实施起来就比较复杂了

特别后面有Windows Server的机器的话，如果实施及配置还有维护过程就比较复杂了

相对而言，Nginx/HAProxy+Keepalived就简单多了。

HAProxy的特点是：

1、HAProxy也是支持虚拟主机的。

2、HAProxy的优点能够补充Nginx的一些缺点，比如支持Session的保持，Cookie的引导

同时支持通过获取指定的url来检测后端服务器的状态

3、HAProxy跟LVS类似，本身就只是一款负载均衡软件

单纯从效率上来讲HAProxy会比Nginx有更出色的负载均衡速度，在并发处理上也是优于Nginx的

4、HAProxy支持TCP协议的负载均衡转发，可以对MySQL读进行负载均衡

对后端的MySQL节点进行检测和负载均衡，大家可以用LVS+Keepalived对MySQL主从做负载均衡

5、HAProxy负载均衡策略非常多，HAProxy的负载均衡算法现在具体有如下8种：

- roundrobin，表示简单的轮询，这个不多说，这个是负载均衡基本都具备的；
- static-rr，表示根据权重，建议关注；

- leastconn，表示最少连接者先处理，建议关注；
- source，表示根据请求源IP，这个跟Nginx的IP_hash机制类似,我们用其作为解决session问题的一种方法，建议关注；
- ri，表示根据请求的URI；
- rl_param，表示根据请求的URI参数'balance url_param' requires an URL parameter name；
- hdr(name)，表示根据HTTP请求头来锁定每一次HTTP请求；
- rdp-cookie(name)，表示根据据cookie(name)来锁定并哈希每一次TCP请求。

19、mysql数据备份工具

mysqldump工具

mysqldump是mysql自带的备份工具，目录在bin目录下面：/usr/local/mysql/bin/mysqldump。支持基于innodb的热备份，但是由于是逻辑备份，所以速度不是很快，适合备份数据比较小的场景，Mysqldump完全备份+二进制日志可以实现基于时间点的恢复。

基于LVM快照备份

在物理备份中，有基于文件系统的物理备份（LVM的快照），也可以直接用tar之类的命令对整个数据库目录，进行打包备份，但是这些只能进行冷备份，不同的存储引擎备份的也不一样，myisam自动备份到表级别，而innodb不开启独立表空间的话只能备份整个数据库。

tar包备份

percona提供的xtrabackup工具，支持innodb的物理热备份，支持完全备份，增量备份，而且速度非常快，支持innodb存储引起的数据在不同，数据库之间迁移，支持复制模式下的从机备份恢复备份恢复，为了让xtrabackup支持更多的功能扩展，可以设立独立表空间，打开 innodb_file_per_table功能，启用之后可以支持单独的表备份

20、keepalive的工作原理和如何做到健康检查

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

Keepalived健康检查方式配置

```
HTTP_GET|SSL_GET
HTTP_GET | SSL_GET
{
url {
path /# HTTP/SSL 检查的url可以是多个
digest <STRING> # HTTP/SSL 检查后的摘要信息用工具genhash生成
status_code 200# HTTP/SSL 检查返回的状态码
}
connect_port 80 # 连接端口
bindto<IPADD>
connect_timeout 3 # 连接超时时间
```

```
nb_get_retry 3 # 重连次数
delay_before_retry 2 #连接间隔时间
}
```

21、统计ip访问情况，要求分析nginx访问日志，找出访问页面数量在前十位的ip

```
cat access.log | awk '{print $1}' | uniq -c | sort -rn | head -10
```

22、使用tcpdump监听主机为192.168.1.1，tcp端口为80的数据，同时将输出结果保存输出到tcpdump.log

```
tcpdump 'host 192.168.1.1 and port 80' > tcpdump.log
```

23、如何将本地80 端口的请求转发到8080 端口，当前主机IP 为192.168.2.1

```
iptables -A PREROUTING -d 192.168.2.1 -p tcp -m tcp -dport 80 -j DNAT-to-
destination 192.168.2.1:8080
```

24、简述raid0 raid1 raid5 三种工作模式的工作原理及特点

RAID 0：带区卷，连续以位或字节为单位分割数据，并行读/写于多个磁盘上，因此具有很高的数据传输率，但它没有数据冗余，RAID 0 只是单纯地提高性能，并没有为数据的可靠性提供保证，而且其中的一个磁盘失效将影响到所有数据。因此，RAID 0 不能应用于数据安全性要求高的场合

RAID 1：镜像卷，它是通过磁盘数据镜像实现数据冗余，在成对的独立磁盘上产生互为备份的数据，不能提升写数据效率。当原始数据繁忙时，可直接从镜像拷贝中读取数据，因此RAID1 可以提高读取性能，RAID 1 是磁盘阵列中单位成本最高的，镜像卷可用容量为总容量的1/2，但提供了很高的数据安全性和可用性，当一个磁盘失效时，系统可以自动切换到镜像磁盘上读写，而不需要重组失效的数据

RAID5：至少由3块硬盘组成，分布式奇偶校验的独立磁盘结构，它的奇偶校验码存在于所有磁盘上，任何一个硬盘损坏，都可以根据其它硬盘上的校验位来重建损坏的数据（最多允许1块硬盘损坏），所以raid5可以实现数据冗余，确保数据的安全性，同时raid5也可以提升数据的读写性能

25、你对现在运维工程师的理解和以及对其工作的认识

运维工程师在公司当中责任重大，需要保证时刻为公司及客户提供最高、最快、最稳定、最安全的服务

运维工程师的一个小小的失误，很有可能会对公司及客户造成重大损失

因此运维工程师的工作需要严谨及富有创新精神

26、实时抓取并显示当前系统中tcp 80端口的网络数据信息，请写出完整操作命令

```
tcpdump -nn tcp port 80
```

27、服务器开不了机怎么解决一步步的排查

A、造成服务器故障的原因可能有以下几点：

B、如何排查服务器故障的处理步骤如下：

1：1、先看服务器的电源指示灯是否亮，如果电源灯不亮，先检查并确认电源没问题时，试着按开机键是否能点亮服务器．如果不能点亮，和数据确认后先更换备用服务器以便快速恢复业务．

2：　2 如果服务器电源灯亮，接上显示器和键盘，如果服务器系统有异常（比如蓝屏…）不能正常登录系统，先和数据确认，是否执行能重启服务器或是更换备用服务器，以便快速恢复业务．

3：　3 如果正确输入用户名和密码情况下能登录系统，查看网卡指示灯是否正常，并用 ifconfig 命令查看网卡接口状态。用 ping 对 端 ip 测试网络是否连通，

4：4、如果 ping 不通，先和数据人员确认并检查网卡配置文件参数是否配置正确，是否正确配置网关（不正确则修正后）用“ifdown；ifup　网卡名“命令重启单个网卡，网卡接口（灯）状态正常后，再用 ping 命令测试，

5：5、还 ping 不通，及时排查并确保本地尾纤，模块等物理设备接入正常，收发光在规定范围内，和数据人员确认是否可以重启服务器，并确认数据方面没有网络配置和数据方面的变化。

6：6、能 ping 通则告知数据人员，并让数据人员帮忙确认链路是否正常，有没有丢包现象等，没有丢包就 OK，有丢包就继续排查尾纤，模块等，直到链路正常没有丢包，数据人员能及时的从远程登录服务器做数据配置能快速恢复业务为 OK。

7 、7、如果不能接入服务器，与数据确认是否可以重启，重启后登陆正常，继续 3.4.5.6 步骤，如果还是不行，权衡利弊，有没有必要更换新的服务器上去，恢复业务要紧。

28、Linux系统中病毒怎么解决

1）最简单有效的方法就是重装系统

2）要查的话就是找到病毒文件然后删除，中毒之后一般机器cpu、内存使用率会比较高，机器向外发包等异常情况，排查方法简单介绍下

- top 命令找到cpu使用率最高的进程
- 一般病毒文件命名都比较乱，可以用 ps aux 找到病毒文件位置

- rm -f 命令删除病毒文件
- 检查计划任务、开机启动项和病毒文件目录有无其他可以文件等

3）由于即使删除病毒文件不排除有潜伏病毒，所以最好是把机器备份数据之后重装一下

29、发现一个病毒文件你删了他又自动创建怎么解决

公司的内网某台linux服务器流量莫名其妙的剧增,用iftop查看有连接外网的情况，针对这种情况一般重点查看netstat连接的外网ip和端口。

用lsof -p pid可以查看到具体是那些进程，哪些文件经查勘发现/root下有相关的配置conf.n hhe两个可疑文件，rm -rf后不到一分钟就自动生成了，由此推断是某个母进程产生的这些文件。所以找到母进程就是找到罪魁祸首

查杀病毒最好断掉外网访问，还好是内网服务器，可以通过内网访问，断了内网，病毒就失去外联的能力，杀掉它就容易的多，怎么找到呢，找了半天也没有看到蛛丝马迹，没办法只有ps axu一个个排查，方法是查看可以的用户和和系统相似而又不是的冒牌货，果然，看到了如下进程可疑

看不到图片就是/usr/bin/.sshd，于是我杀掉所有.sshd相关的进程，然后直接删掉.sshd这个可执行文件，然后才删掉了文章开头提到的自动复活的文件

总结一下，遇到这种问题，如果不是太严重，尽量不要重装系统

一般就是先断外网，然后利用iftop,ps,netstat,chattr,lsof,pstree这些工具顺藤摸瓜

一般都能找到元凶。但是如果遇到诸如此类的问题

/boot/efi/EFI/redhat/grub.efi: Heuristics.Broken.Executable FOUND，个人觉得就要重装系统了

30、说说TCP/IP的七层模型

应用层 (Application)：
网络服务与最终用户的一个接口。
协议有：HTTP FTP TFTP SMTP SNMP DNS TELNET HTTPS POP3 DHCP

表示层（Presentation Layer）：
数据的表示、安全、压缩。（在五层模型里面已经合并到了应用层）
格式有，JPEG、ASCll、DECOIC、加密格式等

会话层（Session Layer）：
建立、管理、终止会话。（在五层模型里面已经合并到了应用层）
对应主机进程，指本地主机与远程主机正在进行的会话

传输层 (Transport)：
定义传输数据的协议端口号，以及流控和差错校验。
协议有：TCP UDP，数据包一旦离开网卡即进入网络传输层

网络层 (Network)：
进行逻辑地址寻址，实现不同网络之间的路径选择。
协议有：ICMP IGMP IP（IPV4 IPV6） ARP RARP

数据链路层 (Link)：
建立逻辑连接、进行硬件地址寻址、差错校验等功能。（由底层网络定义协议）
将比特组合成字节进而组合成帧，用MAC地址访问介质，错误发现但不能纠正

物理层（Physical Layer）：
是计算机网络OSI模型中最低的一层

物理层规定:为传输数据所需要的物理链路创建、维持、拆除而提供具有机械的，电子的，功能的和规范的特性

简单的说，物理层确保原始的数据可在各种物理媒体上传输。局域网与广域网皆属第1、2层

物理层是OSI的第一层，它虽然处于最底层，却是整个开放系统的基础

物理层为设备之间的数据通信提供传输媒体及互连设备，为数据传输提供可靠的环境

如果您想要用尽量少的词来记住这个第一层，那就是“信号和介质”

31、你常用的Nginx模块，用来做什么

rewrite模块，实现重写功能
access模块：来源控制
ssl模块：安全加密
ngx_http_gzip_module：网络传输压缩模块
ngx_http_proxy_module 模块实现代理
ngx_http_upstream_module模块实现定义后端服务器列表
ngx_cache_purge实现缓存清除功能

32、请列出你了解的web服务器负载架构

Nginx
Haproxy
Keepalived
LVS

33、查看http的并发请求数与其TCP连接状态

```
netstat -n | awk '/^tcp/ {++S[$NF]} END {for(a in S) print a, S[a]}'


还有ulimit -n 查看linux系统打开最大的文件描述符，这里默认1024

不修改这里web服务器修改再大也没用，若要用就修改很几个办法，这里说其中一个：

修改/etc/security/limits.conf
* soft nofile 10240
* hard nofile 10240
重启后生效
```

34、用tcpdump嗅探80端口的访问看看谁最高

```
tcpdump -i eth0 -tnn dst port 80 -c 1000 | awk -F"." '{print $1"."$2"."$3"."$4}'|
sort | uniq -c | sort -nr |head -20
```

35、写一个脚本，实现判断192.168.1.0/24网络里，当前在线的IP有哪些，能ping通则认为在线

```
#!/bin/bash
for ip in `seq 1 255`
do

{

ping -c 1 192.168.1.$ip > /dev/null 2>&1
```

```
if [ $? -eq 0 ]; then
echo 192.168.1.$ip UP
else
echo 192.168.1.$ip DOWN
fi
}&
done
wait
```

36、已知 apache 服务的访问日志按天记录在服务器本地目录/app/logs 下，由于磁盘空间紧张现在要求只能保留最近 7 天的访问日志！请问如何解决？请给出解决办法或配置或处理命令

创建文件脚本：

```
#!/bin/bash

for n in `seq 14`

do

date -s "11/0$n/14"

touch access_www_`(date +%F)`.log

done
```

解决方法：

```
# pwd/application/logs

# ll

-rw-r--r--. 1 root root 0 Jan  1 00:00 access_www_2015-01-01.log
-rw-r--r--. 1 root root 0 Jan  2 00:00 access_www_2015-01-02.log
-rw-r--r--. 1 root root 0 Jan  3 00:00 access_www_2015-01-03.log
-rw-r--r--. 1 root root 0 Jan  4 00:00 access_www_2015-01-04.log
-rw-r--r--. 1 root root 0 Jan  5 00:00 access_www_2015-01-05.log
-rw-r--r--. 1 root root 0 Jan  6 00:00 access_www_2015-01-06.log
-rw-r--r--. 1 root root 0 Jan  7 00:00 access_www_2015-01-07.log
-rw-r--r--. 1 root root 0 Jan  8 00:00 access_www_2015-01-08.log
-rw-r--r--. 1 root root 0 Jan  9 00:00 access_www_2015-01-09.log
-rw-r--r--. 1 root root 0 Jan 10 00:00 access_www_2015-01-10.log
-rw-r--r--. 1 root root 0 Jan 11 00:00 access_www_2015-01-11.log
-rw-r--r--. 1 root root 0 Jan 12 00:00 access_www_2015-01-12.log
-rw-r--r--. 1 root root 0 Jan 13 00:00 access_www_2015-01-13.log

-rw-r--r--. 1 root root 0 Jan 14 00:00 access_www_2015-01-14.log

# find /application/logs/ -type f -mtime +7 -name "*.log"|xargs rm -f

##也可以使用-exec rm -f {} \;进行删除

# ll
```

```
-rw-r--r--. 1 root root 0 Jan  7 00:00 access_www_2015-01-07.log
-rw-r--r--. 1 root root 0 Jan  8 00:00 access_www_2015-01-08.log
-rw-r--r--. 1 root root 0 Jan  9 00:00 access_www_2015-01-09.log
-rw-r--r--. 1 root root 0 Jan 10 00:00 access_www_2015-01-10.log
-rw-r--r--. 1 root root 0 Jan 11 00:00 access_www_2015-01-11.log
-rw-r--r--. 1 root root 0 Jan 12 00:00 access_www_2015-01-12.log
-rw-r--r--. 1 root root 0 Jan 13 00:00 access_www_2015-01-13.log

-rw-r--r--. 1 root root 0 Jan 14 00:00 access_www_2015-01-14.log
```

37、如何优化 Linux系统（可以不说太具体）？

- 不用root，添加普通用户，通过sudo授权管理
- 更改默认的远程连接SSH服务端口及禁止root用户远程连接
- 定时自动更新服务器时间
- 配置国内yum源
- 关闭selinux及iptables（iptables工作场景如果有外网IP一定要打开，高并发除外）
- 调整文件描述符的数量
- 精简开机启动服务（crond rsyslog network sshd）
- 内核参数优化（/etc/sysctl.conf）
- 更改字符集，支持中文，但建议还是用英文字符集，防止乱码
- 锁定关键系统文件
- 清空/etc/issue，去除系统及内核版本登录前的屏幕显示

38、请执行命令取出 linux 中 eth0 的 IP 地址(请用 cut，有能力者也可分别用 awk,sed 命令答)

```
cut方法1：

# ifconfig eth0|sed -n '2p'|cut -d ":" -f2|cut -d " " -f1
192.168.20.130

awk方法2：

# ifconfig eth0|awk 'NR==2'|awk -F ":" '{print $2}'|awk '{print $1}'

192.168.20.130

awk多分隔符方法3：

# ifconfig eth0|awk 'NR==2'|awk -F "[: ]+" '{print $4}'

192.168.20.130

sed方法4：

# ifconfig eth0|sed -n '/inet addr/p'|sed -r 's#^.*ddr:(.*)Bc.*$#\1#g'

192.168.20.130
```

39、请写出下面 linux SecureCRT 命令行快捷键命令的功能?

Ctrl + a：光标移动到行首
Ctrl + c：终止当前程序
Ctrl + d：如果光标前有字符则删除，没有则退出当前中断
Ctrl + e：光标移动到行尾
Ctrl + l：清屏
Ctrl + u：剪切光标以前的字符
Ctrl + k：剪切光标以后的字符
Ctrl + y：复制u/k的内容
Ctrl + r：查找最近用过的命令
tab：命令或路径补全
Ctrl+shift+c：复制
Ctrl+shift+v：粘贴

40、每天晚上 12 点，打包站点目录/var/www/html 备份到/data 目录下（最好每次备份按时间生成不同的备份包）

```
# cat a.sh

#/bin/bash

cd /var/www/ && /bin/tar zcf /data/html-`date +%m-%d%H`.tar.gz html/

# crontab –e

00 00 * * * /bin/sh /root/a.sh
```

1-40题来自微信公众号：杰哥的*IT*之旅

41.linux如何挂在windows下的共享目录

```
mount.cifs //192.168.1.3/server /mnt/server -o user=administrator,pass=123456
```

> linux 下的server需要自己手动建一个 后面的user与pass 是windows主机的账号和密码 注意空格和逗号

42.查看http的并发请求数与其TCP连接状态

```
netstat –n | awk '/^tcp/ {++b[$NF]} END {for(a in b) print a, b[a]}'
```

> 还有ulimit -n 查看linux系统打开最大的文件描述符，这里默认1024，不修改这里web服务器修改再大也没用。

> 修改/etc/security/limits.conf

```
* soft nofile 10240
* hard nofile 10240
```

> 重启后生效

43.用tcpdump嗅探80端口的访问看看谁最高

```
tcpdump -i eth0 -tnn dst port 80 -c 1000 | awk -F"." '{print $1"."$2"."$3"."$4}'
| sort | uniq -c | sort -nr |head -5 
```

44.查看/var/log目录下文件数

```
ls /var/log/ -lR| grep "^-" |wc -l
```

45.查看当前系统每个IP的连接数

```
netstat -n | awk '/^tcp/ {print $5}'| awk -F: '{print $1}' | sort | uniq -c |
sort -rn
```

46.shell下32位随机密码生成

```
cat /dev/urandom | head -1 | md5sum | head -c 32 >> /pass
```

> 将生成的32位随机数 保存到/pass文件里了

47.统计出apache的access.log中访问量最多的5个IP

```
 cat access_log | awk  '{print $1}' | sort | uniq -c | sort -n -r | head -5
```

48.如何查看二进制文件的内容

> 我们一般通过hexdump命令 来查看二进制文件的内容。

> hexdump -C XXX(文件名) -C是参数 不同的参数有不同的意义

- -C 是比较规范的 十六进制和ASCII码显示
- -c 是单字节字符显示
- -b 单字节八进制显示
- -o 是双字节八进制显示
- -d 是双字节十进制显示
- -x 是双字节十六进制显示

49.ps aux 中的VSZ代表什么意思，RSS代表什么意思

> VSZ:虚拟内存集,进程占用的虚拟内存空间

> RSS:物理内存集,进程占用实际物理内存空间

50.检测并修复/dev/hda5

> fsck用来检查和维护不一致的文件系统。若系统掉电或磁盘发生问题，可利用fsck命令对文件系统进行检查,用法:

51.Linux系统的开机启动顺序

> 加载BIOS–>读取MBR–>Boot Loader–>加载内核–>用户层init一句inittab文件来设定系统运行的等级(一般3或者5，3是多用户命令行，5是界面)–>init进程执行rc.syninit–>启动内核模块–>执行不同级别运行的脚本程序–>执行/etc/rc.d/rc.local(本地运行服务)–>执行/bin/login,就可以登录了。

52.符号链接与硬链接的区别

> 我们可以把符号链接，也就是软连接 当做是 windows系统里的 快捷方式。

> 硬链接 就好像是 又复制了一份.

> ln 3.txt 4.txt 这是硬链接，相当于复制，不可以跨分区，但修改3,4会跟着变，若删除3,4不受任何影响。

> ln -s 3.txt 4.txt 这是软连接，相当于快捷方式。修改4,3也会跟着变，若删除3,4就坏掉了。不可以用了。

53.保存当前磁盘分区的分区表

> dd 命令是一个强大的命令，在复制的同时进行转换

```
dd if=/dev/sda of=./mbr.txt bs=1 count=512
```

54.如何在文本里面进行复制、粘贴，删除行，删除全部，按行查找和按字母查找。

> 以下操作全部在`vi/vim`命令行状态操作，不要在编辑状态操作。

- 在文本里 移动到想要复制的行按`yy`想复制到哪就移动到哪，然后按`P`就黏贴了
- 删除行 移动到改行 按`dd`
- 删除全部`dG`这里注意`G`一定要大写
- 按行查找 :90 这样就是找到第90行
- 按字母查找 `/path` 这样就是找到`path`这个单词所在的位置，文本里可能存在多个,多次查找会显示在不同的位置。

55.手动安装grub

```
grub-install /dev/sda
```

41-55来自原文:https://zhang.ge/1986.html

56.取出文件aaa.txt的第4到7行

```
[root@localhost ~]# cat aaa.txt
1.aaa
2.bbbbbbb
3.ccccccccccccc
4.ddddddddddddddddddddddd
5.eeeeeeeeeeeeeeeee
6.fffffffffffffffffffffffffffffffff
7.ggggggggggggggggggggg
8.hhhhhhhhhhhhhhhhhhhhhhhhhhhhh
9.iiiiiiiiiiiiiiiiii
10.jjjjjjjjjjjjjjjjjjjjjjjjj
11.kkk
12.lllllllllll
```

```
[root@localhost ~]# sed -n '4,7p' aaa.txt
4.ddddddddddddddddddddddd
5.eeeeeeeeeeeeeeeee
6.fffffffffffffffffffffffffffffffff
7.ggggggggggggggggggggg
```

57.当前目录下txt结尾的文件

```
[root@localhost ~]# ls
1.txt  2.txt  3.pdf  aaa.txt  anaconda-ks.cfg
[root@localhost ~]# find ./ -name "*.txt"
./aaa.txt
./1.txt
./2.txt
```

58.查找/usr目录下超过1M的文件

```
[root@localhost ~]# find /usr -type f -size +10240k
/usr/lib/locale/locale-archive
/usr/lib64/libicudata.so.50.1.2
```

59.写一个定时任务5点到8点执行

```
* 5-8 * * * /usr/bin/backup
```

60.mysql主从复制原理

主库db的更新事件(update、insert、delete)被写到binlog。
主库创建一个binlog dump thread，把binlog的内容发送到从库。
从库启动并发起连接，连接到主库。
从库启动之后，创建一个I/O线程，读取主库传过来的binlog内容并写入到relay log。

从库启动之后，创建一个SQL线程，从relay log里面读取内容，从Exec_Master_Log_Pos位置开始执行读取到的更新事件，将更新内容写入到slave的db。

61.vim有几种工作模式

命令模式。行末模式，编辑模式

62.简述dns解析流程？访问www.baidu.com的解析流程

优先查找本地dns缓存,查找本地/etc/hosts文件，是否有强制解析,如果没有去/etc/resolv.conf指定的dns服务器中查找记录(需联网,在dns服务器中找到解析记录后，在本地dns中添加缓存,完成一次dns解析

63.讲解一下DNS查询的两种模式

递归查询

递归查询是一种DNS 服务器的查询模式，在该模式下DNS 服务器接收到客户机请求，必须使用一个准确的查询结果回复客户机。如果DNS 服务器本地没有存储查询DNS 信息，那么该服务器会询问其他服务器，并将返回的查询结果提交给客户机。

迭代查询

DNS 服务器另外一种查询方式为迭代查询，DNS 服务器会向客户机提供其他能够解析查询请求的DNS 服务器地址，当客户机发送查询请求时，DNS 服务器并不直接回复查询结果，而是告诉客户机另一台DNS 服务器地址，客户机再向这台DNS 服务器提交请求，依次循环直到返回查询的结果为止。

64.描述一下正向代理和反向代理

正向代理

比如我们国内访问国外网站，直接访问访问不到，我们可以通过一个正向代理服务器，请求发到代理服，代理服务器能够访问国外网站，这样由代理去国外网站取到返回数据，再返回给我们，这样我们就能访问了。

反向代理

反向代理实际运行方式是指以代理服务器来接受internet上的连接请求，然后将请求转发给内部网络上的服务器，并将从服务器上得到的结果返回给internet上请求连接的客户端，此时客户只是访问代理服务器却不知道后面有多少服务器。

56-64来自：https://mp.weixin.qq.com/s/SelnWzfxUqzwbSaRxck_aA

65.简述 ETCD 及其特点?

etcd 是 CoreOS 团队发起的开源项目，是一个管理配置信息和服务发现（service discovery）的项目，它的目标是构建一个高可用的分布式键值（key-value）数据库，基于 Go 语言实现。

特点：

- 简单：支持 REST 风格的 HTTP+JSON API
- 安全：支持 HTTPS 方式的访问
- 快速：支持并发 1k/s 的写操作
- 可靠：支持分布式结构，基于 Raft 的一致性算法，Raft 是一套通过选举主节点来实现分布式系统一致性的算法。

66.简述 ETCD 适应的场景?

etcd 基于其优秀的特点，可广泛的应用于以下场景：

`服务发现 (Service Discovery)`：服务发现主要解决在同一个分布式集群中的进程或服务，要如何才能找到对方并建立连接。本质上来说，服务发现就是想要了解集群中是否有进程在监听 udp 或 tcp 端口，并且通过名字就可以查找和连接。`消息发布与订阅`：在分布式系统中，最适用的一种组件间通信方式就是消息发布与订阅。即构建一个配置共享中心，数据提供者在这个配置中心发布消息，而消息使用者则订阅他们关心的主题，一旦主题有消息发布，就会实时通知订阅者。通过这种方式可以做到分布式系统配置的集中式管理与动态更新。应用中用到的一些配置信息放到 etcd 上进行集中管理。`负载均衡`：在分布式系统中，为了保证服务的高可用以及数据的一致性，通常都会把数据和服务部署多份，以此达到对等服务，即使其中的某一个服务失效了，也不影响使用。etcd 本身分布式架构存储的信息访问支持负载均衡。etcd 集群化以后，每个 etcd 的核心节点都可以处理用户的请求。所以，把数据量小但是访问频繁的消息数据直接存储到 etcd 中也可以实现负载均衡的效果。`分布式通知与协调`：与消息发布和订阅类似，都用到了 etcd 中的 Watcher 机制，通过注册与异步通知机制，实现分布式环境下不同系统之间的通知与协调，从而对数据变更做到实时处理。`分布式锁`：因为 etcd 使用 Raft 算法保持了数据的强一致性，某次操作存储到集群中的值必然是全局一致的，所以很容易实现分布式锁。锁服务有两种使用方式，一是保持独占，二是控制时序。集群监控与 Leader 竞选：通过 etcd 来进行监控实现起来非常简单并且实时性强。

67.简述 HAProxy 及其特性?

HAProxy 是可提供高可用性、负载均衡以及基于 TCP 和 HTTP 应用的代理，是免费、快速并且可靠的一种解决方案。HAProxy 非常适用于并发大（并发达 1w 以上）web 站点，这些站点通常又需要会话保持或七层处理。HAProxy 的运行模式使得它可以很简单安全的整合至当前的架构中，同时可以保护 web 服务器不被暴露到网络上。

HAProxy 的主要特性有：

- 可靠性和稳定性非常好，可以与硬件级的 F5 负载均衡设备相媲美；
- 最高可以同时维护 40000-50000 个并发连接，单位时间内处理的最大请求数为 20000 个，最大处理能力可达 10Git/s；
- 支持多达 8 种负载均衡算法，同时也支持会话保持；
- 支持虚机主机功能，从而实现 web 负载均衡更加灵活；
- 支持连接拒绝、全透明代理等独特的功能；
- 拥有强大的 ACL 支持，用于访问控制；
- 其独特的弹性二叉树数据结构，使数据结构的复杂性上升到了 0(1)，即数据的查寻速度不会随着数据条目的增加而速度有所下降；
- 支持客户端的 keepalive 功能，减少客户端与 haproxy 的多次三次握手导致资源浪费，让多个请求在一个 tcp 连接中完成；
- 支持 TCP 加速，零复制功能，类似于 mmap 机制；
- 支持响应池（response buffering）；

- 支持 RDP 协议；
- 基于源的粘性，类似 nginx 的 ip_hash 功能，把来自同一客户端的请求在一定时间内始终调度到上游的同一服务器；
- 更好统计数据接口，其 web 接口显示后端集群中各个服务器的接收、发送、拒绝、错误等数据的统计信息；
- 详细的健康状态检测，web 接口中有关于对上游服务器的健康检测状态，并提供了一定的管理功能；
- 基于流量的健康评估机制；
- 基于 http 认证；
- 基于命令行的管理接口；
- 日志分析器，可对日志进行分析。

68.简述 HAProxy 常见的负载均衡策略?

HAProxy 负载均衡策略非常多，常见的有如下 8 种：

- roundrobin：表示简单的轮询。
- static-rr：表示根据权重。
- leastconn：表示最少连接者先处理。
- source：表示根据请求的源 IP，类似 Nginx 的 IP_hash 机制。
- ri：表示根据请求的 URI。
- rl_param：表示根据 HTTP 请求头来锁定每一次 HTTP 请求。
- rdp-cookie(name)：表示根据据 cookie(name)来锁定并哈希每一次 TCP 请求。

69.简述负载均衡四层和七层的区别?

`四层负载均衡器` 也称为 4 层交换机，主要通过分析 IP 层及 TCP/UDP 层的流量实现基于 IP 加端口的负载均衡，如常见的 LVS、F5 等；

`七层负载均衡器` 也称为 7 层交换机，位于 OSI 的最高层，即应用层，此负载均衡器支持多种协议，如 HTTP、FTP、SMTP 等。7 层负载均衡器可根据报文内容，配合一定的负载均衡算法来选择后端服务器，即“内容交换器”。如常见的 HAProxy、Nginx。

70.简述 LVS、Nginx、HAproxy 的什么异同?

- 相同：三者都是软件负载均衡产品。
- 区别：
- LVS 基于 Linux 操作系统实现软负载均衡，而 HAProxy 和 Nginx 是基于第三方应用实现的软负载均衡；
- LVS 是可实现 4 层的 IP 负载均衡技术，无法实现基于目录、URL 的转发。而 HAProxy 和 Nginx 都可以实现 4 层和 7 层技术，HAProxy 可提供 TCP 和 HTTP 应用的负载均衡综合解决方案；
- LVS 因为工作在 ISO 模型的第四层，其状态监测功能单一，而 HAProxy 在状监测方面功能更丰富、强大，可支持端口、URL、脚本等多种状态检测方式；
- HAProxy 功能强大，但整体性能低于 4 层模式的 LVS 负载均衡。
- Nginx 主要用于 Web 服务器或缓存服务器。

71.简述 Heartbeat?

Heartbeat 是 Linux-HA 项目中的一个组件，它提供了心跳检测和资源接管、集群中服务的监测、失效切换等功能。heartbeat 最核心的功能包括两个部分，心跳监测和资源接管。心跳监测可以通过网络链路和串口进行，而且支持冗余链路，它们之间相互发送报文来告诉对方自己当前的状态，如果在指定的时间内未收到对方发送的报文，那么就认为对方失效，这时需启动资源接管模块来接管运行在对方主机上的资源或者服务。

72.简述 Keepalived 及其工作原理?

Keepalived 是一个基于 VRRP 协议来实现的 LVS 服务高可用方案，可以解决静态路由出现的单点故障问题。

在一个 LVS 服务集群中通常有主服务器（MASTER）和备份服务器（BACKUP）两种角色的服务器，但是对外表现为一个虚拟 IP，主服务器会发送 VRRP 通告信息给备份服务器，当备份服务器收不到 VRRP 消息的时候，即主服务器异常的时候，备份服务器就会接管虚拟 IP，继续提供服务，从而保证了高可用性。

73.简述 Keepalived 体系主要模块及其作用?

keepalived 体系架构中主要有三个模块，分别是 core、check 和 vrrp。

- `core 模块`为 keepalived 的核心，负责主进程的启动、维护及全局配置文件的加载和解析。
- `vrrp 模块`是来实现 VRRP 协议的。
- `check`负责健康检查，常见的方式有端口检查及 URL 检查。

74.简述 Keepalived 如何通过健康检查来保证高可用?

Keepalived 工作在 TCP/IP 模型的第三、四和五层，即网络层、传输层和应用层。

- `网络层`，Keepalived 采用 ICMP 协议向服务器集群中的每个节点发送一个 ICMP 的数据包，如果某个节点没有返回响应数据包，则认为此节点发生了故障，Keepalived 将报告次节点失效，并从服务器集群中剔除故障节点。
- `传输层`，Keepalived 利用 TCP 的端口连接和扫描技术来判断集群节点是否正常。如常见的 web 服务默认端口 80，ssh 默认端口 22 等。Keepalived 一旦在传输层探测到相应端口没用响应数据返回，则认为此端口发生异常，从而将此端口对应的节点从服务器集群中剔除。
- `应用层`，可以运行 FTP、telnet、smtp、dns 等各种不同类型的高层协议，Keepalived 的运行方式也更加全面化和复杂化，用户可以通过自定义 Keepalived 的工作方式，来设定监测各种程序或服务是否正常，若监测结果与设定的正常结果不一致，将此服务对应的节点从服务器集群中剔除。

Keepalived 通过完整的健康检查机制，保证集群中的所有节点均有效从而实现高可用。

75.简述 LVS 的概念及其作用?

LVS 是 linux virtual server 的简写 linux 虚拟服务器，是一个虚拟的服务器集群系统，可以在 unix/linux 平台下实现负载均衡集群功能。

LVS 的主要作用是：通过 LVS 提供的负载均衡技术实现一个高性能、高可用的服务器群集。因此 LVS 主要可以实现：

- 把单台计算机无法承受的大规模的并发访问或数据流量分担到多台节点设备上分别处理，减少用户等待响应的时间，提升用户体验。
- 单个重负载的运算分担到多台节点设备上做并行处理，每个节点设备处理结束后，将结果汇总，返回给用户，系统处理能力得到大幅度提高。
- 7*24 小时的服务保证，任意一个或多个设备节点设备宕机，不能影响到业务。在负载均衡集群中，所有计算机节点都应该提供相同的服务，集群负载均衡获取所有对该服务的如站请求。

76.简述 LVS 的工作模式及其工作过程?

LVS 有三种负载均衡的模式，分别是 VS/NAT（nat 模式）、VS/DR（路由模式）、VS/TUN（隧道模式）。

- NAT 模式（VS-NAT）

- 原理：首先负载均衡器接收到客户的请求数据包时，根据调度算法决定将请求发送给哪个后端的真实服务器（RS）。然后负载均衡器就把客户端发送的请求数据包的目标 IP 地址及端口改成后端真实服务器的 IP 地址（RIP）。真实服务器响应完请求后，查看默认路由，把响应后的数据包发送给负载均衡器，负载均衡器在接收到响应包后，把包的源地址改成虚拟地址（VIP）然后发送回给客户端。
- 优点：集群中的服务器可以使用任何支持 TCP/IP 的操作系统，只要负载均衡器有一个合法的 IP 地址。
- 缺点：扩展性有限，当服务器节点增长过多时，由于所有的请求和应答都需要经过负载均衡器，因此负载均衡器将成为整个系统的瓶颈。
- IP 隧道模式（VS-TUN）
- 原理：首先负载均衡器接收到客户的请求数据包时，根据调度算法决定将请求发送给哪个后端的真实服务器（RS）。然后负载均衡器就把客户端发送的请求报文封装一层 IP 隧道（T-IP）转发到真实服务器（RS）。真实服务器响应完请求后，查看默认路由，把响应后的数据包直接发送给客户端，不需要经过负载均衡器。
- 优点：负载均衡器只负责将请求包分发给后端节点服务器，而 RS 将应答包直接发给用户。所以，减少了负载均衡器的大量数据流动，负载均衡器不再是系统的瓶颈，也能处理很巨大的请求量。
- 缺点：隧道模式的 RS 节点需要合法 IP，这种方式需要所有的服务器支持“IP Tunneling”。
- 直接路由模式（VS-DR）
- 原理：首先负载均衡器接收到客户的请求数据包时，根据调度算法决定将请求发送给哪个后端的真实服务器（RS）。然后负载均衡器就把客户端发送的请求数据包的目标 MAC 地址改成后端真实服务器的 MAC 地址（R-MAC）。真实服务器响应完请求后，查看默认路由，把响应后的数据包直接发送给客户端，不需要经过负载均衡器。
- 优点：负载均衡器只负责将请求包分发给后端节点服务器，而 RS 将应答包直接发给用户。所以，减少了负载均衡器的大量数据流动，负载均衡器不再是系统的瓶颈，也能处理很巨大的请求量。
- 缺点：需要负载均衡器与真实服务器 RS 都有一块网卡连接到同一物理网段上，必须在同一个局域网环境。

77.简述 LVS 调度器常见算法（均衡策略）？

LVS 调度器用的调度方法基本分为两类：

- 固定调度算法：rr，wrr，dh，sh
- rr：轮询算法，将请求依次分配给不同的 rs 节点，即 RS 节点中均摊分配。适合于 RS 所有节点处理性能接近的情况。
- wrr：加权轮训调度，依据不同 RS 的权值分配任务。权值较高的 RS 将优先获得任务，并且分配到的连接数将比权值低的 RS 更多。相同权值的 RS 得到相同数目的连接数。
- dh：目的地址哈希调度（destination hashing）以目的地址为关键字查找一个静态 hash 表来获得所需 RS。
- sh：源地址哈希调度（source hashing）以源地址为关键字查找一个静态 hash 表来获得需要的 RS。
- 动态调度算法：wlc，lc，lblc，lblcr
- wlc：加权最小连接数调度，假设各台 RS 的权值依次为 Wi，当前 tcp 连接数依次为 Ti，依次去 Ti/Wi 为最小的 RS 作为下一个分配的 RS。
- lc：最小连接数调度（least-connection），IPVS 表存储了所有活动的连接。LB 会比较将连接请求发送到当前连接最少的 RS。
- lblc：基于地址的最小连接数调度（locality-based least-connection）：将来自同一个目的地址的请求分配给同一台 RS，此时这台服务器是尚未满负荷的。否则就将这个请求分配给连接数最小的 RS，并以它作为下一次分配的首先考虑。

78.简述 LVS、Nginx、HAProxy 各自优缺点?

- Nginx 的优点：
- 工作在网络的 7 层之上，可以针对 http 应用做一些分流的策略，比如针对域名、目录结构。Nginx 正则规则比 HAProxy 更为强大和灵活。
- Nginx 对网络稳定性的依赖非常小，理论上能 ping 通就就能进行负载功能，LVS 对网络稳定性依赖比较大，稳定要求相对更高。
- Nginx 安装和配置、测试比较简单、方便，有清晰的日志用于排查和管理，LVS 的配置、测试就要花比较长的时间了。
- 可以承担高负载压力且稳定，一般能支撑几万次的并发量，负载度比 LVS 相对小些。
- Nginx 可以通过端口检测到服务器内部的故障，比如根据服务器处理网页返回的状态码、超时等等。
- Nginx 不仅仅是一款优秀的负载均衡器/反向代理软件，它同时也是功能强大的 Web 应用服务器。
- Nginx 作为 Web 反向加速缓存越来越成熟了，速度比传统的 Squid 服务器更快，很多场景下都将其作为反向代理加速器。
- Nginx 作为静态网页和图片服务器，这方面的性能非常优秀，同时第三方模块也很多。
- Nginx 的缺点：
- Nginx 仅能支持 http、https 和 Email 协议，这样就在适用范围上面小些。
- 对后端服务器的健康检查，只支持通过端口来检测，不支持通过 url 来检测。
- 不支持 Session 的直接保持，需要通过 ip_hash 来解决。
- LVS 的优点：
- 抗负载能力强、是工作在网络 4 层之上仅作分发之用，没有流量的产生。因此负载均衡软件里的性能最强的，对内存和 cpu 资源消耗比较低。
- LVS 工作稳定，因为其本身抗负载能力很强，自身有完整的双机热备方案。
- 无流量，LVS 只分发请求，而流量并不从它本身出去，这点保证了均衡器 IO 的性能不会收到大流量的影响。
- 应用范围较广，因为 LVS 工作在 4 层，所以它几乎可对所有应用做负载均衡，包括 http、数据库等。
- LVS 的缺点是：
- 软件本身不支持正则表达式处理，不能做动静分离。相对来说，Nginx/HAProxy+Keepalived 则具有明显的优势。
- 如果是网站应用比较庞大的话，LVS/DR+Keepalived 实施起来就比较复杂了。相对来说 Nginx/HAProxy+Keepalived 就简单多了。
- HAProxy 的优点：
- HAProxy 也是支持虚拟主机的。
- HAProxy 的优点能够补充 Nginx 的一些缺点，比如支持 Session 的保持，Cookie 的引导，同时支持通过获取指定的 url 来检测后端服务器的状态。
- HAProxy 跟 LVS 类似，本身就只是一款负载均衡软件，单纯从效率上来讲 HAProxy 会比 Nginx 有更出色的负载均衡速度，在并发处理上也是优于 Nginx 的。
- HAProxy 支持 TCP 协议的负载均衡转发。

79.简述代理服务器的概念及其作用?

代理服务器是一个位于客户端和原始（资源）服务器之间的服务器，为了从原始服务器取得内容，客户端向代理服务器发送一个请求并指定目标原始服务器，然后代理服务器向原始服务器转交请求并将获得的内容返回给客户端。

其主要作用有：

- 资源获取：代替客户端实现从原始服务器的资源获取；
- 加速访问：代理服务器可能离原始服务器更近，从而起到一定的加速作用；
- 缓存作用：代理服务器保存从原始服务器所获取的资源，从而实现客户端快速的获取；
- 隐藏真实地址：代理服务器代替客户端去获取原始服务器资源，从而隐藏客户端真实信息。

80.简述高可用集群可通过哪两个维度衡量高可用性，各自含义是什么?

- RTO（Recovery Time Objective）：RTO 指服务恢复的时间，最佳的情况是 0，即服务立即恢复；最坏是无穷大，即服务永远无法恢复；
- RPO（Recovery Point Objective）：RPO 指指当灾难发生时允许丢失的数据量，0 意味着使用同步的数据，大于 0 意味着有数据丢失，如“RPO=1 d”指恢复时使用一天前的数据，那么一天之内的数据就丢失了。因此，恢复的最佳情况是 RTO = RPO = 0，几乎无法实现。

81.简述什么是 CAP 理论?

CAP 理论指出了在分布式系统中需要满足的三个条件，主要包括：

- Consistency（一致性）：所有节点在同一时间具有相同的数据；
- Availability（可用性）：保证每个请求不管成功或者失败都有响应；
- Partition tolerance（分区容错性）：系统中任意信息的丢失或失败不影响系统的继续运行。

CAP 理论的核心是：一个分布式系统不可能同时很好的满足一致性，可用性和分区容错性这三个需求，最多只能同时较好的满足两个。

82.简述什么是 ACID 理论?

- 原子性(Atomicity)：整体不可分割性，要么全做要不全不做；
- 一致性(Consistency)：事务执行前、后数据库状态均一致；
- 隔离性(Isolation)：在事务未提交前，它操作的数据，对其它用户不可见；
- 持久性 (Durable)：一旦事务成功，将进行永久的变更，记录与 redo 日志。

83.简述什么是 Kubernetes?

Kubernetes 是一个全新的基于容器技术的分布式系统支撑平台。是 Google 开源的容器集群管理系统（谷歌内部:Borg）。在 Docker 技术的基础上，为容器化的应用提供部署运行、资源调度、服务发现和动态伸缩等一系列完整功能，提高了大规模容器集群管理的便捷性。并且具有完备的集群管理能力，多层次的安全防护和准入机制、多租户应用支撑能力、透明的服务注册和发现机制、内建智能负载均衡器、强大的故障发现和自我修复能力、服务滚动升级和在线扩容能力、可扩展的资源自动调度机制以及多粒度的资源配额管理能力。

84.简述 Kubernetes 和 Docker 的关系?

- Docker 提供容器的生命周期管理和，Docker 镜像构建运行时容器。它的主要优点是将将软件/应用程序运行所需的设置和依赖项打包到一个容器中，从而实现了可移植性等优点。
- Kubernetes 用于关联和编排在多个主机上运行的容器。

85.简述 Kubernetes 中什么是 Minikube、Kubectl、Kubelet?

- `Minikube`是一种可以在本地轻松运行一个单节点 Kubernetes 群集的工具。
- `Kubectl`是一个命令行工具，可以使用该工具控制 Kubernetes 集群管理器，如检查群集资源，创建、删除和更新组件，查看应用程序。
- `Kubelet` 是一个代理服务，它在每个节点上运行，并使从服务器与主服务器通信。

86.简述 Kubernetes 常见的部署方式?

常见的 Kubernetes 部署方式有：

- kubeadm：也是推荐的一种部署方式；
- 二进制：
- minikube：在本地轻松运行一个单节点 Kubernetes 群集的工具。

87.78述 Kubernetes 如何实现集群管理?

在集群管理方面，Kubernetes 将集群中的机器划分为一个 Master 节点和一群工作节点 Node。其中，在 Master 节点运行着集群管理相关的一组进程 kube-apiserver、kube-controller-manager 和 kube-scheduler，这些进程实现了整个集群的资源管理、Pod 调度、弹性伸缩、安全控制、系统监控和纠错等管理能力，并且都是全自动完成的。

88.简述 Kubernetes 的优势、适应场景及其特点?

Kubernetes 作为一个完备的分布式系统支撑平台，其主要优势：

- 容器编排
- 轻量级
- 开源
- 弹性伸缩
- 负载均衡

Kubernetes 常见场景：

- 快速部署应用
- 快速扩展应用
- 无缝对接新的应用功能
- 节省资源，优化硬件资源的使用

Kubernetes 相关特点：

- 可移植: 支持公有云、私有云、混合云、多重云（multi-cloud）。
- 可扩展: 模块化,、插件化、可挂载、可组合。
- 自动化: 自动部署、自动重启、自动复制、自动伸缩/扩展。

89.简述 Kubernetes 的缺点或当前的不足之处?

Kubernetes 当前存在的缺点（不足）如下：

- 安装过程和配置相对困难复杂。
- 管理服务相对繁琐。
- 运行和编译需要很多时间。
- 它比其他替代品更昂贵。
- 对于简单的应用程序来说，可能不需要涉及 Kubernetes 即可满足。

90.简述 Kubernetes 相关基础概念?

- `master`：k8s 集群的管理节点，负责管理集群，提供集群的资源数据访问入口。拥有 Etcd 存储服务（可选），运行 Api Server 进程，Controller Manager 服务进程及 Scheduler 服务进程。
- `node`（worker）：Node（worker）是 Kubernetes 集群架构中运行 Pod 的服务节点，是 Kubernetes 集群操作的单元，用来承载被分配 Pod 的运行，是 Pod 运行的宿主机。运行 docker eninge 服务，守护进程 kunelet 及负载均衡器 kube-proxy。

- `pod`：运行于 Node 节点上，若干相关容器的组合。Pod 内包含的容器运行在同一宿主机上，使用相同的网络命名空间、IP 地址和端口，能够通过 localhost 进行通信。Pod 是 Kurbernetes 进行创建、调度和管理的最小单位，它提供了比容器更高层次的抽象，使得部署和管理更加灵活。一个 Pod 可以包含一个容器或者多个相关容器。
- `label`：Kubernetes 中的 Label 实质是一系列的 Key/Value 键值对，其中 key 与 value 可自定义。Label 可以附加到各种资源对象上，如 Node、Pod、Service、RC 等。一个资源对象可以定义任意数量的 Label，同一个 -Label 也可以被添加到任意数量的资源对象上去。Kubernetes 通过 Label Selector（标签选择器）查询和筛选资源对象。
- `Replication Controller`：Replication Controller 用来管理 Pod 的副本，保证集群中存在指定数量的 Pod 副本。集群中副本的数量大于指定数量，则会停止指定数量之外的多余容器数量。反之，则会启动少于指定数量个数的容器，保证数量不变。Replication Controller 是实现弹性伸缩、动态扩容和滚动升级的核心。
- `Deployment`：Deployment 在内部使用了 RS 来实现目的，Deployment 相当于 RC 的一次升级，其最大的特色为可以随时获知当前 Pod 的部署进度。
- `HPA`（Horizontal Pod Autoscaler）：Pod 的横向自动扩容，也是 Kubernetes 的一种资源，通过追踪分析 RC 控制的所有 Pod 目标的负载变化情况，来确定是否需要针对性的调整 Pod 副本数量。
- `Service`：Service 定义了 Pod 的逻辑集合和访问该集合的策略，是真实服务的抽象。Service 提供了一个统一的服务访问入口以及服务代理和发现机制，关联多个相同 Label 的 Pod，用户不需要了解后台 Pod 是如何运行。
- `Volume`：Volume 是 Pod 中能够被多个容器访问的共享目录，Kubernetes 中的 Volume 是定义在 Pod 上，可以被一个或多个 Pod 中的容器挂载到某个目录下。
- `Namespace`：Namespace 用于实现多租户的资源隔离，可将集群内部的资源对象分配到不同的 Namespace 中，形成逻辑上的不同项目、小组或用户组，便于不同的 Namespace 在共享使用整个集群的资源的同时还能被分别管理。

91.简述 Kubernetes 集群相关组件?

Kubernetes Master 控制组件，调度管理整个系统（集群），包含如下组件:

- `Kubernetes API Server`：作为 Kubernetes 系统的入口，其封装了核心对象的增删改查操作，以 RESTful API 接口方式提供给外部客户和内部组件调用，集群内各个功能模块之间数据交互和通信的中心枢纽。
- `Kubernetes Scheduler`：为新建立的 Pod 进行节点(node)选择(即分配机器)，负责集群的资源调度。
- `Kubernetes Controller`：负责执行各种控制器，目前已经提供了很多控制器来保证 Kubernetes 的正常运行。
- `Replication Controller`：管理维护 Replication Controller，关联 Replication Controller 和 -Pod，保证 Replication Controller 定义的副本数量与实际运行 Pod 数量一致。
- `Node Controller`：管理维护 Node，定期检查 Node 的健康状态，标识出(失效|未失效)的 Node 节点。
- `Namespace Controller`：管理维护 Namespace，定期清理无效的 Namespace，包括 Namesapce 下的 API 对象，比如 Pod、Service 等。
- `Service Controller`：管理维护 Service，提供负载以及服务代理。
- `EndPoints Controller`：管理维护 Endpoints，关联 Service 和 Pod，创建 Endpoints 为 Service 的后端，当 Pod 发生变化时，实时更新 Endpoints。
- `Service Account Controller`：管理维护 Service Account，为每个 Namespace 创建默认的 Service Account，同时为 Service Account 创建 Service Account Secret。
- `Persistent Volume Controller`：管理维护 Persistent Volume 和 Persistent Volume Claim，为新的 Persistent Volume Claim 分配 Persistent Volume 进行绑定，为释放的 Persistent Volume 执行清理回收。

- `Daemon Set Controller`：管理维护 Daemon Set，负责创建 Daemon Pod，保证指定的 Node 上正常的运行 Daemon Pod。
- `Deployment Controller`：管理维护 Deployment，关联 Deployment 和 Replication Controller，保证运行指定数量的 Pod。当 Deployment 更新时，控制实现 Replication Controller 和 Pod 的更新。
- `Job Controller`：管理维护 Job，为 Jod 创建一次性任务 Pod，保证完成 Job 指定完成的任务数目
- `Pod Autoscaler Controller`：实现 Pod 的自动伸缩，定时获取监控数据，进行策略匹配，当满足条件时执行 Pod 的伸缩动作。

92.简述 Kubernetes RC 的机制?

Replication Controller 用来管理 Pod 的副本，保证集群中存在指定数量的 Pod 副本。当定义了 RC 并提交至 Kubernetes 集群中之后，Master 节点上的 Controller Manager 组件获悉，并同时巡检系统中当前存活的目标 Pod，并确保目标 Pod 实例的数量刚好等于此 RC 的期望值，若存在过多的 Pod 副本在运行，系统会停止一些 Pod，反之则自动创建一些 Pod。

93.简述 Kubernetes Replica Set 和 Replication Controller 之间有什么区别?

Replica Set 和 Replication Controller 类似，都是确保在任何给定时间运行指定数量的 Pod 副本。不同之处在于 RS 使用基于集合的选择器，而 Replication Controller 使用基于权限的选择器。

94.简述 kube-proxy 作用?

kube-proxy 运行在所有节点上，它监听 apiserver 中 service 和 endpoint 的变化情况，创建路由规则以提供服务 IP 和负载均衡功能。简单理解此进程是 Service 的透明代理兼负载均衡器，其核心功能是将到某个 Service 的访问请求转发到后端的多个 Pod 实例上。

95.简述 kube-proxy iptables 原理?

Kubernetes 从 1.2 版本开始，将 iptables 作为 kube-proxy 的默认模式。iptables 模式下的 kube-proxy 不再起到 Proxy 的作用，其核心功能：通过 API Server 的 Watch 接口实时跟踪 Service 与 Endpoint 的变更信息，并更新对应的 iptables 规则，Client 的请求流量则通过 iptables 的 NAT 机制"直接路由"到目标 Pod。

96.简述 kube-proxy ipvs 原理?

IPVS 在 Kubernetes1.11 中升级为 GA 稳定版。IPVS 则专门用于高性能负载均衡，并使用更高效的数据结构（Hash 表），允许几乎无限的规模扩张，因此被 kube-proxy 采纳为最新模式。

在 IPVS 模式下，使用 iptables 的扩展 ipset，而不是直接调用 iptables 来生成规则链。iptables 规则链是一个线性的数据结构，ipset 则引入了带索引的数据结构，因此当规则很多时，也可以很高效地查找和匹配。

可以将 ipset 简单理解为一个 IP（段）的集合，这个集合的内容可以是 IP 地址、IP 网段、端口等，iptables 可以直接添加规则对这个"可变的集合"进行操作，这样做的好处在于可以大大减少 iptables 规则的数量，从而减少性能损耗。

97.简述 kube-proxy ipvs 和 iptables 的异同?

iptables 与 IPVS 都是基于 Netfilter 实现的，但因为定位不同，二者有着本质的差别：iptables 是为防火墙而设计的；IPVS 则专门用于高性能负载均衡，并使用更高效的数据结构（Hash 表），允许几乎无限的规模扩张。

与 iptables 相比，IPVS 拥有以下明显优势：

1. 为大型集群提供了更好的可扩展性和性能；
2. 支持比 iptables 更复杂的复制均衡算法（最小负载、最少连接、加权等）；
3. 支持服务器健康检查和连接重试等功能；
4. 可以动态修改 ipset 的集合，即使 iptables 的规则正在使用这个集合。

98.简述 Kubernetes 中什么是静态 Pod?

静态 pod 是由 kubelet 进行管理的仅存在于特定 Node 的 Pod 上，他们不能通过 API Server 进行管理，无法与 ReplicationController、Deployment 或者 DaemonSet 进行关联，并且 kubelet 无法对他们进行健康检查。静态 Pod 总是由 kubelet 进行创建，并且总是在 kubelet 所在的 Node 上运行。

99.简述 Kubernetes 中 Pod 可能位于的状态?

- `Pending`：API Server 已经创建该 Pod，且 Pod 内还有一个或多个容器的镜像没有创建，包括正在下载镜像的过程。
- `Running`：Pod 内所有容器均已创建，且至少有一个容器处于运行状态、正在启动状态或正在重启状态。
- `Succeeded`：Pod 内所有容器均成功执行退出，且不会重启。
- `Failed`：Pod 内所有容器均已退出，但至少有一个容器退出为失败状态。
- `Unknown`：由于某种原因无法获取该 Pod 状态，可能由于网络通信不畅导致。

100.简述 Kubernetes 创建一个 Pod 的主要流程?

Kubernetes 中创建一个 Pod 涉及多个组件之间联动，主要流程如下：

1. 客户端提交 Pod 的配置信息（可以是 yaml 文件定义的信息）到 kube-apiserver。
2. Apiserver 收到指令后，通知给 controller-manager 创建一个资源对象。
3. Controller-manager 通过 api-server 将 pod 的配置信息存储到 ETCD 数据中心中。
4. Kube-scheduler 检测到 pod 信息会开始调度预选，会先过滤掉不符合 Pod 资源配置要求的节点，然后开始调度调优，主要是挑选出更适合运行 pod 的节点，然后将 pod 的资源配置单发送到 node 节点上的 kubelet 组件上。
5. Kubelet 根据 scheduler 发来的资源配置单运行 pod，运行成功后，将 pod 的运行信息返回给 scheduler，scheduler 将返回的 pod 运行状况的信息存储到 etcd 数据中心。

101.简述 Kubernetes 中 Pod 的重启策略?

Pod 重启策略（RestartPolicy）应用于 Pod 内的所有容器，并且仅在 Pod 所处的 Node 上由 kubelet 进行判断和重启操作。当某个容器异常退出或者健康检查失败时，kubelet 将根据 RestartPolicy 的设置来进行相应操作。

Pod 的重启策略包括 Always、OnFailure 和 Never，默认值为 Always。

- `Always`：当容器失效时，由 kubelet 自动重启该容器；
- `OnFailure`：当容器终止运行且退出码不为 0 时，由 kubelet 自动重启该容器；
- `Never`：不论容器运行状态如何，kubelet 都不会重启该容器。

同时 Pod 的重启策略与控制方式关联，当前可用于管理 Pod 的控制器包括 ReplicationController、Job、DaemonSet 及直接管理 kubelet 管理（静态 Pod）。

不同控制器的重启策略限制如下：

- RC 和 DaemonSet：必须设置为 Always，需要保证该容器持续运行；
- Job：OnFailure 或 Never，确保容器执行完成后不再重启；
- kubelet：在 Pod 失效时重启，不论将 RestartPolicy 设置为何值，也不会对 Pod 进行健康检查。

102.简述 Kubernetes 中 Pod 的健康检查方式?

对 Pod 的健康检查可以通过两类探针来检查：LivenessProbe 和 ReadinessProbe。

- `LivenessProbe 探针`：用于判断容器是否存活（running 状态），如果 LivenessProbe 探针探测到容器不健康，则 kubelet 将杀掉该容器，并根据容器的重启策略做相应处理。若一个容器不包含 LivenessProbe 探针，kubelet 认为该容器的 LivenessProbe 探针返回值用于是“Success”。
- `ReadineeProbe 探针`：用于判断容器是否启动完成（ready 状态）。如果 ReadinessProbe 探针探测到失败，则 Pod 的状态将被修改。Endpoint Controller 将从 Service 的 Endpoint 中删除包含该容器所在 Pod 的 Eenpoint。
- `startupProbe 探针`：启动检查机制，应用一些启动缓慢的业务，避免业务长时间启动而被上面两类探针 kill 掉。

103.简述 Kubernetes Pod 的 LivenessProbe 探针的常见方式?

kubelet 定期执行 LivenessProbe 探针来诊断容器的健康状态，通常有以下三种方式：

- `ExecAction`：在容器内执行一个命令，若返回码为 0，则表明容器健康。
- `TCPSocketAction`：通过容器的 IP 地址和端口号执行 TCP 检查，若能建立 TCP 连接，则表明容器健康。
- `HTTPGetAction`：通过容器的 IP 地址、端口号及路径调用 HTTP Get 方法，若响应的状态码大于等于 200 且小于 400，则表明容器健康。

104.简述 Kubernetes Pod 的常见调度方式?

Kubernetes 中，Pod 通常是容器的载体，主要有如下常见调度方式：

- Deployment 或 RC：该调度策略主要功能就是自动部署一个容器应用的多份副本，以及持续监控副本的数量，在集群内始终维持用户指定的副本数量。
- NodeSelector：定向调度，当需要手动指定将 Pod 调度到特定 Node 上，可以通过 Node 的标签（Label）和 Pod 的 nodeSelector 属性相匹配。
- NodeAffinity 亲和性调度：亲和性调度机制极大的扩展了 Pod 的调度能力，目前有两种节点亲和力表达：
- requiredDuringSchedulingIgnoredDuringExecution：硬规则，必须满足指定的规则，调度器才可以调度 Pod 至 Node 上（类似 nodeSelector，语法不同）。
- preferredDuringSchedulingIgnoredDuringExecution：软规则，优先调度至满足的 Node 的节点，但不强求，多个优先级规则还可以设置权重值。
- Taints 和 Tolerations（污点和容忍）：
- Taint：使 Node 拒绝特定 Pod 运行；
- Toleration：为 Pod 的属性，表示 Pod 能容忍（运行）标注了 Taint 的 Node。

105.简述 Kubernetes 初始化容器（init container）?

`init container` 的运行方式与应用容器不同，它们必须先于应用容器执行完成，当设置了多个 init container 时，将按顺序逐个运行，并且只有前一个 init container 运行成功后才能运行后一个 init container。当所有 init container 都成功运行后，Kubernetes 才会初始化 Pod 的各种信息，并开始创建和运行应用容器。

106.简述 Kubernetes deployment 升级过程?

- 初始创建 Deployment 时，系统创建了一个 ReplicaSet，并按用户的需求创建了对应数量的 Pod 副本。
- 当更新 Deployment 时，系统创建了一个新的 ReplicaSet，并将其副本数量扩展到 1，然后将旧 ReplicaSet 缩减为 2。
- 之后，系统继续按照相同的更新策略对新旧两个 ReplicaSet 进行逐个调整。
- 最后，新的 ReplicaSet 运行了对应个新版本 Pod 副本，旧的 ReplicaSet 副本数量则缩减为 0。

107.简述 Kubernetes deployment 升级策略?

在 Deployment 的定义中，可以通过 spec.strategy 指定 Pod 更新的策略，目前支持两种策略：Recreate（重建）和 RollingUpdate（滚动更新），默认值为 RollingUpdate。

- `Recreate`：设置 spec.strategy.type=Recreate，表示 Deployment 在更新 Pod 时，会先杀掉所有正在运行的 Pod，然后创建新的 Pod。
- `RollingUpdate`：设置 spec.strategy.type=RollingUpdate，表示 Deployment 会以滚动更新的方式来逐个更新 Pod。同时，可以通过设置 spec.strategy.rollingUpdate 下的两个参数（maxUnavailable 和 maxSurge）来控制滚动更新的过程。

108.简述 Kubernetes DaemonSet 类型的资源特性?

DaemonSet 资源对象会在每个 Kubernetes 集群中的节点上运行，并且每个节点只能运行一个 pod，这是它和 deployment 资源对象的最大也是唯一的区别。

因此，在定义 yaml 文件中，不支持定义 replicas。

它的一般使用场景如下：

- 在去做每个节点的日志收集工作。
- 监控每个节点的的运行状态。

109.简述 Kubernetes 自动扩容机制?

Kubernetes 使用 Horizontal Pod Autoscaler（HPA）的控制器实现基于 CPU 使用率进行自动 Pod 扩缩容的功能。

HPA 控制器周期性地监测目标 Pod 的资源性能指标，并与 HPA 资源对象中的扩缩容条件进行对比，在满足条件时对 Pod 副本数量进行调整。

- **HPA 原理**

ubernetes 中的某个 Metrics Server（Heapster 或自定义 Metrics Server）持续采集所有 Pod 副本的指标数据。

HPA 控制器通过 Metrics Server 的 API（Heapster 的 API 或聚合 API）获取这些数据，基于用户定义的扩缩容规则进行计算，得到目标 Pod 副本数量。

当目标 Pod 副本数量与当前副本数量不同时，HPA 控制器就向 Pod 的副本控制器（Deployment、RC 或 ReplicaSet）发起 scale 操作，调整 Pod 的副本数量，完成扩缩容操作。

110.简述 Kubernetes Service 类型?

通过创建 Service，可以为一组具有相同功能的容器应用提供一个统一的入口地址，并且将请求负载分发到后端的各个容器应用上。其主要类型有：

- `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，通常用于公有云。

111.简述 Kubernetes Service 分发后端的策略?

Service 负载分发的策略有：RoundRobin 和 SessionAffinity

- RoundRobin：默认为轮询模式，即轮询将请求转发到后端的各个 Pod 上。
- SessionAffinity：基于客户端 IP 地址进行会话保持的模式，即第 1 次将某个客户端发起的请求转发到后端的某个 Pod 上，之后从相同的客户端发起的请求都将被转发到后端相同的 Pod 上。

112.简述 Kubernetes Headless Service?

在某些应用场景中，若需要人为指定负载均衡器，不使用 Service 提供的默认负载均衡的功能，或者应用程序希望知道属于同组服务的其他实例。

Kubernetes 提供了 Headless Service 来实现这种功能，即不为 Service 设置 ClusterIP（入口 IP 地址），仅通过 Label Selector 将后端的 Pod 列表返回给调用的客户端。

113.简述 Kubernetes 外部如何访问集群内的服务?

对于 Kubernetes，集群外的客户端默认情况，无法通过 Pod 的 IP 地址或者 Service 的虚拟 IP 地址:虚拟端口号进行访问。

通常可以通过以下方式进行访问 Kubernetes 集群内的服务：

- 映射 Pod 到物理机：将 Pod 端口号映射到宿主机，即在 Pod 中采用 hostPort 方式，以使客户端应用能够通过物理机访问容器应用。
- 映射 Service 到物理机：将 Service 端口号映射到宿主机，即在 Service 中采用 nodePort 方式，以使客户端应用能够通过物理机访问容器应用。
- 映射 Sercie 到 LoadBalancer：通过设置 LoadBalancer 映射到云服务商提供的 LoadBalancer 地址。这种用法仅用于在公有云服务提供商的云平台上设置 Service 的场景。

114.简述 Kubernetes ingress?

Kubernetes 的 Ingress 资源对象，用于将不同 URL 的访问请求转发到后端不同的 Service，以实现 HTTP 层的业务路由机制。

Kubernetes 使用了 Ingress 策略和 Ingress Controller，两者结合并实现了一个完整的 Ingress 负载均衡器。

使用 Ingress 进行负载分发时，Ingress Controller 基于 Ingress 规则将客户端请求直接转发到 Service 对应的后端 Endpoint（Pod）上，从而跳过 kube-proxy 的转发功能，kube-proxy 不再起作用，全过程为：ingress controller + ingress 规则 —-> services。

同时当 Ingress Controller 提供的是对外服务，则实际上实现的是边缘路由器的功能。

115.简述 Kubernetes 镜像的下载策略?

K8s 的镜像下载策略有三种：Always、Never、IFNotPresent。

- `Always`：镜像标签为 latest 时，总是从指定的仓库中获取镜像。
- `Never`：禁止从仓库中下载镜像，也就是说只能使用本地镜像。
- `IfNotPresent`：仅当本地没有对应镜像时，才从目标仓库中下载。默认的镜像下载策略是：当镜像标签是 latest 时，默认策略是 Always；当镜像标签是自定义时（也就是标签不是 latest），那

么默认策略是 IfNotPresent。

116.简述 Kubernetes 的负载均衡器?

负载均衡器是暴露服务的最常见和标准方式之一。

根据工作环境使用两种类型的负载均衡器，即内部负载均衡器或外部负载均衡器。

内部负载均衡器自动平衡负载并使用所需配置分配容器，而外部负载均衡器将流量从外部负载引导至后端容器。

117.简述 Kubernetes 各模块如何与 API Server 通信?

Kubernetes API Server 作为集群的核心，负责集群各功能模块之间的通信。

集群内的各个功能模块通过 API Server 将信息存入 etcd，当需要获取和操作这些数据时，则通过 API Server 提供的 REST 接口（用 GET、LIST 或 WATCH 方法）来实现，从而实现各模块之间的信息交互。

如 kubelet 进程与 API Server 的交互：每个 Node 上的 kubelet 每隔一个时间周期，就会调用一次 API Server 的 REST 接口报告自身状态，API Server 在接收到这些信息后，会将节点状态信息更新到 etcd 中。

如 kube-controller-manager 进程与 API Server 的交互：kube-controller-manager 中的 Node Controller 模块通过 API Server 提供的 Watch 接口实时监控 Node 的信息，并做相应处理。

如 kube-scheduler 进程与 API Server 的交互：Scheduler 通过 API Server 的 Watch 接口监听到新建 Pod 副本的信息后，会检索所有符合该 Pod 要求的 Node 列表，开始执行 Pod 调度逻辑，在调度成功后将 Pod 绑定到目标节点上。

118.简述 Kubernetes Scheduler 作用及实现原理?

Kubernetes Scheduler 是负责 Pod 调度的重要功能模块，Kubernetes Scheduler 在整个系统中承担了“承上启下”的重要功能，“承上”是指它负责接收 Controller Manager 创建的新 Pod，为其调度至目标 Node；“启下”是指调度完成后，目标 Node 上的 kubelet 服务进程接管后继工作，负责 Pod 接下来生命周期。

Kubernetes Scheduler 的作用是将待调度的 Pod（API 新创建的 Pod、Controller Manager 为补足副本而创建的 Pod 等）按照特定的调度算法和调度策略绑定（Binding）到集群中某个合适的 Node 上，并将绑定信息写入 etcd 中。

在整个调度过程中涉及**三个对象**：

分别是待调度 Pod 列表、可用 Node 列表，以及调度算法和策略。

Kubernetes Scheduler 通过调度算法调度为待调度 Pod 列表中的每个 Pod 从 Node 列表中选择一个最适合的 Node 来实现 Pod 的调度。

随后，目标节点上的 kubelet 通过 API Server 监听到 Kubernetes Scheduler 产生的 Pod 绑定事件，然后获取对应的 Pod 清单，下载 Image 镜像并启动容器。

119.简述 Kubernetes Scheduler 使用哪两种算法将 Pod 绑定到 worker 节点?

Kubernetes Scheduler 根据如下两种调度算法将 Pod 绑定到最合适的工作节点：

- `预选（Predicates）`：输入是所有节点，输出是满足预选条件的节点。kube-scheduler 根据预选策略过滤掉不满足策略的 Nodes。如果某节点的资源不足或者不满足预选策略的条件则无法通过预选。如“Node 的 label 必须与 Pod 的 Selector 一致”。

- `优选（Priorities）`：输入是预选阶段筛选出的节点，优选会根据优先策略为通过预选的 Nodes 进行打分排名，选择得分最高的 Node。例如，资源越富裕、负载越小的 Node 可能具有越高的排名。

120.简述 Kubernetes kubelet 的作用?

在 Kubernetes 集群中，在每个 Node（又称 Worker）上都会启动一个 kubelet 服务进程。

该进程用于处理 Master 下发到本节点的任务，管理 Pod 及 Pod 中的容器。

每个 kubelet 进程都会在 API Server 上注册节点自身的信息，定期向 Master 汇报节点资源的使用情况，并通过 cAdvisor 监控容器和节点资源。

121.简述 Kubernetes kubelet 监控 Worker 节点资源是使用什么组件来实现的?

kubelet 使用 cAdvisor 对 worker 节点资源进行监控。

在 Kubernetes 系统中，cAdvisor 已被默认集成到 kubelet 组件内，当 kubelet 服务启动时，它会自动启动 cAdvisor 服务，然后 cAdvisor 会实时采集所在节点的性能指标及在节点上运行的容器的性能指标。

122.简述 Kubernetes 如何保证集群的安全性?

Kubernetes 通过一系列机制来实现集群的安全控制，

主要有如下不同的维度：

- 基础设施方面：保证容器与其所在宿主机的隔离；
- 权限方面：
- 最小权限原则：合理限制所有组件的权限，确保组件只执行它被授权的行为，通过限制单个组件的能力来限制它的权限范围。
- 用户权限：划分普通用户和管理员的角色。
- 集群方面：
- API Server 的认证授权：Kubernetes 集群中所有资源的访问和变更都是通过 Kubernetes API Server 来实现的，因此需要建议采用更安全的 HTTPS 或 Token 来识别和认证客户端身份（Authentication），以及随后访问权限的授权（Authorization）环节。
- API Server 的授权管理：通过授权策略来决定一个 API 调用是否合法。对合法用户进行授权并且随后在用户访问时进行鉴权，建议采用更安全的 RBAC 方式来提升集群安全授权。
- 敏感数据引入 Secret 机制：对于集群敏感数据建议使用 Secret 方式进行保护。
- AdmissionControl（准入机制）：对 kubernetes api 的请求过程中，顺序为：先经过认证 & 授权，然后执行准入操作，最后对目标对象进行操作。

123.简述 Kubernetes 准入机制?

在对集群进行请求时，每个准入控制代码都按照一定顺序执行。

如果有一个准入控制拒绝了此次请求，那么整个请求的结果将会立即返回，并提示用户相应的 error 信息。

`准入控制`（AdmissionControl）准入控制本质上为一段准入代码，在对 kubernetes api 的请求过程中，顺序为：先经过认证 & 授权，然后执行准入操作，最后对目标对象进行操作。常用组件（控制代码）如下：

- `AlwaysAdmit`：允许所有请求
- `AlwaysDeny`：禁止所有请求，多用于测试环境。

- `ServiceAccount`：它将 serviceAccounts 实现了自动化，它会辅助 serviceAccount 做一些事情，比如如果 pod 没有 serviceAccount 属性，它会自动添加一个 default，并确保 pod 的 serviceAccount 始终存在。
- `LimitRanger`：观察所有的请求，确保没有违反已经定义好的约束条件，这些条件定义在 namespace 中 LimitRange 对象中。NamespaceExists：观察所有的请求，如果请求尝试创建一个不存在的 namespace，则这个请求被拒绝。

124.简述 Kubernetes RBAC 及其特点（优势）？

RBAC 是基于角色的访问控制，是一种基于个人用户的角色来管理对计算机或网络资源的访问的方法。

相对于其他授权模式，RBAC 具有如下优势：

- 对集群中的资源和非资源权限均有完整的覆盖。
- 整个 RBAC 完全由几个 API 对象完成， 同其他 API 对象一样， 可以用 kubectl 或 API 进行操作。
- 可以在运行时进行调整，无须重新启动 API Server。

125.简述 Kubernetes Secret 作用?

Secret 对象，主要作用是保管私密数据，比如密码、OAuth Tokens、SSH Keys 等信息。

将这些私密信息放在 Secret 对象中比直接放在 Pod 或 Docker Image 中更安全，也更便于使用和分发。

126.简述 Kubernetes Secret 有哪些使用方式?

创建完 secret 之后，可通过如下三种方式使用：

- 在创建 Pod 时，通过为 Pod 指定 Service Account 来自动使用该 Secret。
- 通过挂载该 Secret 到 Pod 来使用它。
- 在 Docker 镜像下载时使用，通过指定 Pod 的 spc.ImagePullSecrets 来引用它。

127.简述 Kubernetes PodSecurityPolicy 机制?

Kubernetes PodSecurityPolicy 是为了更精细地控制 Pod 对资源的使用方式以及提升安全策略。

在开启 PodSecurityPolicy 准入控制器后，Kubernetes 默认不允许创建任何 Pod，需要创建 PodSecurityPolicy 策略和相应的 RBAC 授权策略（Authorizing Policies），Pod 才能创建成功。

128.简述 Kubernetes PodSecurityPolicy 机制能实现哪些安全策略?

在 PodSecurityPolicy 对象中可以设置不同字段来控制 Pod 运行时的各种安全策略，常见的有：

- 特权模式：privileged 是否允许 Pod 以特权模式运行。
- 宿主机资源：控制 Pod 对宿主机资源的控制，如 hostPID：是否允许 Pod 共享宿主机的进程空间。
- 用户和组：设置运行容器的用户 ID（范围）或组（范围）。
- 提升权限：AllowPrivilegeEscalation：设置容器内的子进程是否可以提升权限，通常在设置非 root 用户（MustRunAsNonRoot）时进行设置。
- SELinux：进行 SELinux 的相关配置。

129.简述 Kubernetes 网络模型?

Kubernetes 网络模型中每个 Pod 都拥有一个独立的 IP 地址，并假定所有 Pod 都在一个可以直接连通的、扁平的网络空间中。

所以不管它们是否运行在同一个 Node（宿主机）中，都要求它们可以直接通过对方的 IP 进行访问。

设计这个原则的原因是，用户不需要额外考虑如何建立 Pod 之间的连接，也不需要考虑如何将容器端口映射到主机端口等问题。

同时为每个 Pod 都设置一个 IP 地址的模型使得同一个 Pod 内的不同容器会共享同一个网络命名空间，也就是同一个 Linux 网络协议栈。这就意味着同一个 Pod 内的容器可以通过 localhost 来连接对方的端口。

在 Kubernetes 的集群里，IP 是以 Pod 为单位进行分配的。

一个 Pod 内部的所有容器共享一个网络堆栈（相当于一个网络命名空间，它们的 IP 地址、网络设备、配置等都是共享的）。

130.简述 Kubernetes CNI 模型?

CNI 提供了一种应用容器的插件化网络解决方案，定义对容器网络进行操作和配置的规范，通过插件的形式对 CNI 接口进行实现。

CNI 仅关注在创建容器时分配网络资源，和在销毁容器时删除网络资源。在 CNI 模型中只涉及两个概念：容器和网络。

- `容器`（Container）：是拥有独立 Linux 网络命名空间的环境，例如使用 Docker 或 rkt 创建的容器。容器需要拥有自己的 Linux 网络命名空间，这是加入网络的必要条件。
- `网络`（Network）：表示可以互连的一组实体，这些实体拥有各自独立、唯一的 IP 地址，可以是容器、物理机或者其他网络设备（比如路由器）等。

对容器网络的设置和操作都通过插件（Plugin）进行具体实现，CNI 插件包括两种类型：

CNI Plugin 和 IPAM（IP Address Management）Plugin。

CNI Plugin 负责为容器配置网络资源，IPAM Plugin 负责对容器的 IP 地址进行分配和管理。

IPAM Plugin 作为 CNI Plugin 的一部分，与 CNI Plugin 协同工作。

130.简述 Kubernetes 网络策略?

为实现细粒度的容器间网络访问隔离策略，Kubernetes 引入 Network Policy。

Network Policy 的主要功能是对 Pod 间的网络通信进行限制和准入控制，设置允许访问或禁止访问的客户端 Pod 列表。

Network Policy 定义网络策略，配合策略控制器（Policy Controller）进行策略的实现。

131.简述 Kubernetes 网络策略原理?

Network Policy 的工作原理主要为：policy controller 需要实现一个 API Listener，监听用户设置的 Network Policy 定义，并将网络访问规则通过各 Node 的 Agent 进行实际设置（Agent 则需要通过 CNI 网络插件实现）。

132.简述 Kubernetes 中 flannel 的作用?

Flannel 可以用于 Kubernetes 底层网络的实现，主要作用有：

- 它能协助 Kubernetes，给每一个 Node 上的 Docker 容器都分配互相不冲突的 IP 地址。
- 它能在这些 IP 地址之间建立一个覆盖网络（Overlay Network），通过这个覆盖网络，将数据包原封不动地传递到目标容器内。

133.简述 Kubernetes Calico 网络组件实现原理?

Calico 是一个基于 BGP 的纯三层的网络方案，与 OpenStack、Kubernetes、AWS、GCE 等云平台都能够良好地集成。

Calico 在每个计算节点都利用 Linux Kernel 实现了一个高效的 vRouter 来负责数据转发。每个 vRouter 都通过 BGP 协议把在本节点上运行的容器的路由信息向整个 Calico 网络广播，并自动设置到达其他节点的路由转发规则。

Calico 保证所有容器之间的数据流量都是通过 **IP 路由的方式完成互联互通的。**

Calico 节点组网时可以直接利用数据中心的网络结构（L2 或者 L3），不需要额外的 NAT、隧道或者 Overlay Network，没有额外的封包解包，能够节约 CPU 运算，提高网络效率。

134.简述 Kubernetes 共享存储的作用?

Kubernetes 对于有状态的容器应用或者对数据需要持久化的应用，因此需要更加可靠的存储来保存应用产生的重要数据，以便容器应用在重建之后仍然可以使用之前的数据。因此需要使用共享存储。

135.简述 Kubernetes 数据持久化的方式有哪些?

Kubernetes 通过数据持久化来持久化保存重要数据，常见的方式有：

- EmptyDir（空目录）：没有指定要挂载宿主机上的某个目录，直接由 Pod 内保部映射到宿主机上。类似于 docker 中的 manager volume。
- 场景：
- 只需要临时将数据保存在磁盘上，比如在合并/排序算法中；
- 作为两个容器的共享存储。
- 特性：
- 同个 pod 里面的不同容器，共享同一个持久化目录，当 pod 节点删除时，volume 的数据也会被删除。
- emptyDir 的数据持久化的生命周期和使用的 pod 一致，一般是作为临时存储使用。
- Hostpath：将宿主机上已存在的目录或文件挂载到容器内部。类似于 docker 中的 bind mount 挂载方式。
- 特性：增加了 pod 与节点之间的耦合。
- PersistentVolume（简称 PV）：如基于 NFS 服务的 PV，也可以基于 GFS 的 PV。它的作用是统一数据持久化目录，方便管理。

136.简述 Kubernetes PV 和 PVC?

PV 是对底层网络共享存储的抽象，将共享存储定义为一种“资源”。

PVC 则是用户对存储资源的一个“申请”。

137.简述 Kubernetes PV 生命周期内的阶段?

某个 PV 在生命周期中可能处于以下 4 个阶段（Phaes）之一。

- Available：可用状态，还未与某个 PVC 绑定。
- Bound：已与某个 PVC 绑定。
- Released：绑定的 PVC 已经删除，资源已释放，但没有被集群回收。
- Failed：自动资源回收失败。

138.简述 Kubernetes 所支持的存储供应模式?

Kubernetes 支持两种资源的存储供应模式：静态模式（Static）和动态模式（Dynamic）。

- `静态模式`**：**集群管理员手工创建许多 PV，在定义 PV 时需要将后端存储的特性进行设置。
- `动态模式`**：**集群管理员无须手工创建 PV，而是通过 StorageClass 的设置对后端存储进行描述，标记为某种类型。此时要求 PVC 对存储的类型进行声明，系统将自动完成 PV 的创建及与 PVC 的绑定。

139.简述 Kubernetes CSI 模型?

Kubernetes CSI 是 Kubernetes 推出与容器对接的存储接口标准，存储提供方只需要基于标准接口进行存储插件的实现，就能使用 Kubernetes 的原生存储机制为容器提供存储服务。

CSI 使得存储提供方的代码能和 Kubernetes 代码彻底解耦，部署也与 Kubernetes 核心组件分离，显然，存储插件的开发由提供方自行维护，就能为 Kubernetes 用户提供更多的存储功能，也更加安全可靠。

CSI 包括 CSI Controller 和 CSI Node：

- CSI Controller 的主要功能是提供存储服务视角对存储资源和存储卷进行管理和操作。
- CSI Node 的主要功能是对主机（Node）上的 Volume 进行管理和操作。

140.简述 Kubernetes Worker 节点加入集群的过程?

通常需要对 Worker 节点进行扩容，从而将应用系统进行水平扩展。

主要过程如下：

- 在该 Node 上安装 Docker、kubelet 和 kube-proxy 服务；
- 然后配置 kubelet 和 kubeproxy 的启动参数，将 Master URL 指定为当前 Kubernetes 集群 Master 的地址，最后启动这些服务；
- 通过 kubelet 默认的自动注册机制，新的 Worker 将会自动加入现有的 Kubernetes 集群中；
- Kubernetes Master 在接受了新 Worker 的注册之后，会自动将其纳入当前集群的调度范围。

141.简述 Kubernetes Pod 如何实现对节点的资源控制?

Kubernetes 集群里的节点提供的资源主要是计算资源，计算资源是可计量的能被申请、分配和使用的基础资源。

当前 Kubernetes 集群中的计算资源主要包括 CPU、GPU 及 Memory。CPU 与 Memory 是被 Pod 使用的，因此在配置 Pod 时可以通过参数 CPU Request 及 Memory Request 为其中的每个容器指定所需使用的 CPU 与 Memory 量，Kubernetes 会根据 Request 的值去查找有足够资源的 Node 来调度此 Pod。

通常，一个程序所使用的 CPU 与 Memory 是一个动态的量，确切地说，是一个范围，跟它的负载密切相关：负载增加时，CPU 和 Memory 的使用量也会增加。

142.简述 Kubernetes Requests 和 Limits 如何影响 Pod 的调度?

当一个 Pod 创建成功时，Kubernetes 调度器（Scheduler）会为该 Pod 选择一个节点来执行。

对于每种计算资源（CPU 和 Memory）而言，每个节点都有一个能用于运行 Pod 的最大容量值。调度器在调度时，首先要确保调度后该节点上所有 Pod 的 CPU 和内存的 Requests 总和，不超过该节点能提供给 Pod 使用的 CPU 和 Memory 的最大容量值。

143.简述 Kubernetes Metric Service?

在 Kubernetes 从 1.10 版本后采用 Metrics Server 作为默认的性能数据采集和监控，主要用于提供核心指标（Core Metrics），包括 Node、Pod 的 CPU 和内存使用指标。

对其他自定义指标（Custom Metrics）的监控则由 Prometheus 等组件来完成。

144.简述 Kubernetes 中，如何使用 EFK 实现日志的统一管理?

在 Kubernetes 集群环境中，通常一个完整的应用或服务涉及组件过多，建议对日志系统进行集中化管理，通常采用 EFK 实现。

EFK 是 Elasticsearch、Fluentd 和 Kibana 的组合，其各组件功能如下：

- Elasticsearch：是一个搜索引擎，负责存储日志并提供查询接口；
- Fluentd：负责从 Kubernetes 搜集日志，每个 node 节点上面的 fluentd 监控并收集该节点上面的系统日志，并将处理过后的日志信息发送给 Elasticsearch；
- Kibana：提供了一个 Web GUI，用户可以浏览和搜索存储在 Elasticsearch 中的日志。

通过在每台 node 上部署一个以 DaemonSet 方式运行的 fluentd 来收集每台 node 上的日志。

Fluentd 将 docker 日志目录/var/lib/docker/containers 和/var/log 目录挂载到 Pod 中，然后 Pod 会在 node 节点的/var/log/pods 目录中创建新的目录，可以区别不同的容器日志输出，该目录下有一个日志文件链接到/var/lib/docker/contianers 目录下的容器日志输出。

145.简述 Kubernetes 如何进行优雅的节点关机维护?

由于 Kubernetes 节点运行大量 Pod，因此在进行关机维护之前，建议先使用 kubectl drain 将该节点的 Pod 进行驱逐，然后进行关机维护。

146.简述 Kubernetes 集群联邦?

Kubernetes 集群联邦可以将多个 Kubernetes 集群作为一个集群进行管理。

因此，可以在一个数据中心/云中创建多个 Kubernetes 集群，并使用集群联邦在一个地方控制/管理所有集群。

147.简述 Helm 及其优势?

`Helm`是 Kubernetes 的软件包管理工具。类似 Ubuntu 中使用的 apt、Centos 中使用的 yum 或者 Python 中的 pip 一样。

Helm 能够将一组 K8S 资源打包统一管理, 是查找、共享和使用为 Kubernetes 构建的软件的最佳方式。

Helm 中通常每个包称为一个 Chart，一个 Chart 是一个目录（一般情况下会将目录进行打包压缩，形成 name-version.tgz 格式的单一文件，方便传输和存储）。

- Helm 优势

在 Kubernetes 中部署一个可以使用的应用，需要涉及到很多的 Kubernetes 资源的共同协作。

使用 helm 则具有如下优势：

- 统一管理、配置和更新这些分散的 k8s 的应用资源文件；
- 分发和复用一套应用模板；
- 将应用的一系列资源当做一个软件包管理。
- 对于应用发布者而言，可以通过 Helm 打包应用、管理应用依赖关系、管理应用版本并发布应用到软件仓库。
- 对于使用者而言，使用 Helm 后不用需要编写复杂的应用部署文件，可以以简单的方式在 Kubernetes 上查找、安装、升级、回滚、卸载应用程序。

148简述 OpenShift 及其特性?

OpenShift 是一个容器应用程序平台，用于在安全的、可伸缩的资源上部署新应用程序，而配置和管理开销最小。

OpenShift 构建于 Red Hat Enterprise Linux、Docker 和 Kubernetes 之上，为企业级应用程序提供了一个安全且可伸缩的多租户操作系统，同时还提供了集成的应用程序运行时和库。

其主要特性：

- 自助服务平台：OpenShift 允许开发人员使用 Source-to-Image(S2I)从模板或自己的源代码管理存储库创建应用程序。系统管理员可以为用户和项目定义资源配额和限制，以控制系统资源的使用。
- 多语言支持：OpenShift 支持 Java、Node.js、PHP、Perl 以及直接来自 Red Hat 的 Ruby。OpenShift 还支持中间件产品，如 Apache httpd、Apache Tomcat、JBoss EAP、ActiveMQ 和 Fuse。
- 自动化：OpenShift 提供应用程序生命周期管理功能，当上游源或容器映像发生更改时，可以自动重新构建和重新部署容器。根据调度和策略扩展或故障转移应用程序。
- 用户界面：OpenShift 提供用于部署和监视应用程序的 web UI，以及用于远程管理应用程序和资源的 CLi。
- 协作：OpenShift 允许在组织内或与更大的社区共享项目。
- 可伸缩性和高可用性：OpenShift 提供了容器多租户和一个分布式应用程序平台，其中包括弹性，高可用性，以便应用程序能够在物理机器宕机等事件中存活下来。OpenShift 提供了对容器健康状况的自动发现和自动重新部署。
- 容器可移植性：在 OpenShift 中，应用程序和服务使用标准容器映像进行打包，组合应用程序使用 Kubernetes 进行管理。这些映像可以部署到基于这些基础技术的其他平台上。
- 开源：没有厂商锁定。
- 安全性：OpenShift 使用 SELinux 提供多层安全性、基于角色的访问控制以及与外部身份验证系统(如 LDAP 和 OAuth)集成的能力。
- 动态存储管理：OpenShift 使用 Kubernetes 持久卷和持久卷声明的方式为容器数据提供静态和动态存储管理
- 基于云(或不基于云)：可以在裸机服务器、活来自多个供应商的 hypervisor 和大多数 IaaS 云提供商上部署 OpenShift 容器平台。
- 企业级：Red Hat 支持 OpenShift、选定的容器映像和应用程序运行时。可信的第三方容器映像、运行时和应用程序由 Red Hat 认证。可以在 OpenShift 提供的高可用性的强化安全环境中运行内部或第三方应用程序。
- 日志聚合和 metrics：可以在中心节点收集、聚合和分析部署在 OpenShift 上的应用程序的日志信息。OpenShift 能够实时收集关于应用程序的度量和运行时信息，并帮助不断优化性能。
- 其他特性：OpenShift 支持微服务体系结构，OpenShift 的本地特性足以支持 DevOps 流程，很容易与标准和定制的持续集成/持续部署工具集成。

149.简述 OpenShift projects 及其作用?

OpenShift 管理 projects 和 users。

一个 projects 对 Kubernetes 资源进行分组，以便用户可以使用访问权限。还可以为 projects 分配配额，从而限制了已定义的 pod、volumes、services 和其他资源。

project 允许一组用户独立于其他组组织和管理其内容，必须允许用户访问项目。如果允许创建项目，用户将自动访问自己的项目。

150.简述 OpenShift 高可用的实现?

OpenShift 平台集群的高可用性(HA)有两个不同的方面：

OpenShift 基础设施本身的 HA(即主机)；以及在 OpenShift 集群中运行的应用程序的 HA。

默认情况下，OpenShift 为 master 节点提供了完全支持的本机 HA 机制。

对于应用程序或“pods”，如果 pod 因任何原因丢失，Kubernetes 将调度另一个副本，将其连接到服务层和持久存储。

如果整个节点丢失，Kubernetes 会为它所有的 pod 安排替换节点，最终所有的应用程序都会重新可用。pod 中的应用程序负责它们自己的状态，因此它们需要自己维护应用程序状态(如 HTTP 会话复制或数据库复制)。

151.简述 OpenShift 的 SDN 网络实现?

默认情况下，Docker 网络使用仅使用主机虚机网桥 bridge，主机内的所有容器都连接至该网桥。

连接到此桥的所有容器都可以彼此通信，但不能与不同主机上的容器通信。

为了支持跨集群的容器之间的通信，OpenShift 容器平台使用了软件定义的网络(SDN)方法。

软件定义的网络是一种网络模型，它通过**几个网络层的抽象来管理网络服务。**

SDN 将处理流量的软件(称为控制平面)和路由流量的底层机制(称为数据平面)解耦。SDN 支持控制平面和数据平面之间的通信。

在 OpenShift 中，可以为 pod 网络配置三个 SDN 插件:

- ovs-subnet：默认插件，子网提供了一个 flat pod 网络，其中每个 pod 可以与其他 pod 和 service 通信。
- ovs-multitenant：该为 pod 和服务提供了额外的隔离层。当使用此插件时，每个 project 接收一个惟一的虚拟网络 ID (VNID)，该 ID 标识来自属于该 project 的 pod 的流量。通过使用 VNID，来自不同 project 的 pod 不能与其他 project 的 pod 和 service 通信。
- ovs-network policy：此插件允许管理员使用 NetworkPolicy 对象定义自己的隔离策略。

cluster network 由 OpenShift SDN 建立和维护，它使用 Open vSwitch 创建 overlay 网络，master 节点不能通过集群网络访问容器，除非 master 同时也为 node 节点。

152.简述 OpenShift 角色及其作用?

OpenShift 的角色具有不同级别的访问和策略，包括集群和本地策略。

user 和 group 可以同时与多个 role 关联。

153.简述 OpenShift 支持哪些身份验证?

OpenShift 容器平台支持的其他认证类型包括:

- Basic Authentication (Remote)：一种通用的后端集成机制，允许用户使用针对远程标识提供者验证的凭据登录到 OpenShift 容器平台。用户将他们的用户名和密码发送到 OpenShift 容器平台，OpenShift 平台通过到服务器的请求验证这些凭据，并将凭据作为基本的 Auth 头传递。这要求用户在登录过程中向 OpenShift 容器平台输入他们的凭据。
- Request Header Authentication：用户使用请求头值(如 X-RemoteUser)登录到 OpenShift 容器平台。它通常与身份验证代理结合使用，身份验证代理对用户进行身份验证，然后通过请求头值为 OpenShift 容器平台提供用户标识。
- Keystone Authentication：Keystone 是一个 OpenStack 项目，提供标识、令牌、目录和策略服务。OpenShift 容器平台与 Keystone 集成，通过配置 OpenStack Keystone v3 服务器将用户存储在内部数据库中，从而支持共享身份验证。这种配置允许用户使用 Keystone 凭证登录 OpenShift 容器平台。
- LDAP Authentication：用户使用他们的 LDAP 凭证登录到 OpenShift 容器平台。在身份验证期间，LDAP 目录将搜索与提供的用户名匹配的条目。如果找到匹配项，则尝试使用条目的专有名称(DN)和提供的密码进行简单绑定。
- GitHub Authentication：GitHub 使用 OAuth，它允许与 OpenShift 容器平台集成使用 OAuth 身份验证来促进令牌交换流。这允许用户使用他们的 GitHub 凭证登录到 OpenShift 容器平台。为了防止使用 GitHub 用户 id 的未授权用户登录到 OpenShift 容器平台集群，可以将访问权限限制在特定的 GitHub 组织中。

154.简述什么是中间件?

`中间件`是一种独立的系统软件或服务程序，分布式应用软件借助这种软件在不同的技术之间共享资源。

通常位于客户机/服务器的操作系统中间，是**连接两个独立应用程序或独立系统的软件。**

通过中间件实现两个不同系统之间的信息交换。

155. 磁盘使用率检测（用shell脚本）

```
root@ecs-c13b ~]# cat fdisk.sh
#!/bin/bash
# 截取IP
IP=`ifconfig eth0 |awk -F " " 'NR==2{print $2}'`
# 定义使用率,并转换为数字
SPACE=`df -Ph |awk '{print int($5)}'`

for i in $SPACE
do
if [ $i -ge 90 ]
then
       echo "$IP的磁盘使用率已经超过了90%，请及时处理"
fi
done
```

156. LVS 负载均衡有哪些策略?

LVS一共有三种工作模式：DR，Tunnel,NAT

157. 谈谈你对LVS的理解？**

LVS是一个虚拟的服务器集群系统，在unix系统下实现负载均衡的功能；采用IP负载均衡技术和机遇内容请求分发技术来实现。

LVS采用三层结构，分别是：

第一层：负载调度器

第二层：服务池

第三层：共享存储

负载调度器（load balancer/ Director），是整个集群的总代理，它有两个网卡，一个网卡面对访问网站的客户端，一个网卡面对整个集群的内部。负责将客户端的请求发送到一组服务器上执行，而客户也认为服务是来自这台主的。举个生动的例子，集群是个公司，负载调度器就是在外接揽生意，将接揽到的生意分发给后台的真正干活的真正的主机们。当然需要将活按照一定的算法分发下去，让大家都公平的干活。

服务器池（server pool/ Realserver），是一组真正执行客户请求的服务器，可以当做WEB服务器。就是上面例子中的小员工。

共享存储（shared storage），它为服务器池提供一个共享的存储区，这样很容易使得服务器池拥有相同的内容，提供相同的服务。一个公司得有一个后台账目吧，这才能协调。不然客户把钱付给了A，而换B接待客户，因为没有相同的账目。B说客户没付钱，那这样就不是客户体验度的问题了。

158. 负载均衡的原理是什么?

当客户端发起请求时，请求直接发给Director Server（调度器），这时会根据设定的调度算法，将请求按照算法的规定智能的分发到真正的后台服务器。以达到将压力均摊。

但是我们知道，http的连接时无状态的，假设这样一个场景，我登录某宝买东西，当我看上某款商品时，我将它加入购物车，但是我刷新了一下页面，这时由于负载均衡的原因，调度器又选了新的一台服务器为我提供服务，我刚才的购物车内容全都不见了，这样就会有十分差的用户体验。

所以就还需要一个存储共享，这样就保证了用户请求的数据是一样的

159. LVS由哪两部分组成的?

LVS 由2部分程序组成，包括 ipvs 和 ipvsadm。

1.ipvs(ip virtual server)：一段代码工作在内核空间，叫ipvs，是真正生效实现调度的代码。

\2. ipvsadm：另外一段是工作在用户空间，叫ipvsadm，负责为ipvs内核框架编写规则，定义谁是集 群服务，而谁是后端真实的服务器(Real Server)

160. 与lvs相关的术语有哪些?

DS：Director Server。指的是前端负载均衡器节点。

RS：Real Server。后端真实的工作服务器。

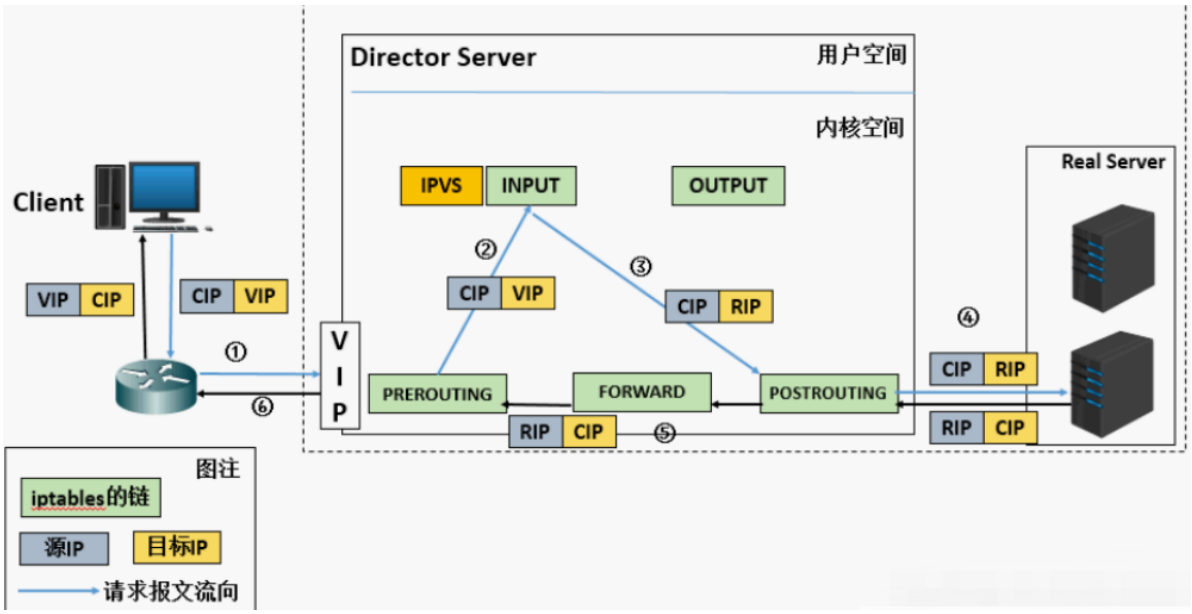
VIP：Virtual IP 向外部直接面向用户请求，作为用户请求的目标的IP地址。

DIP：Director Server IP，主要用于和内部主机通讯的IP地址。

RIP：Real Server IP，后端服务器的IP地址。

CIP：Client IP，访问客户端的IP地址。

161. LVS-NAT模式的原理**



(a). 当用户请求到达Director Server，此时请求的数据报文会先到内核空间的PREROUTING链。此时报文的源IP为CIP，目标IP为VIP

(b). PREROUTING检查发现数据包的目标IP是本机，将数据包送至INPUT链

(c). IPVS比对数据包请求的服务是否为集群服务，若是，修改数据包的目标IP地址为后端服务器 IP， 然后将数据包发至POSTROUTING链。此时报文的源IP为CIP，目标IP为RIP

(d). POSTROUTING链通过选路，将数据包发送给Real Server

(e). Real Server比对发现目标为自己的IP，开始构建响应报文发回给Director Server。此时报文 的源IP为RIP，目标IP为CIP

(f). Director Server在响应客户端前，此时会将源IP地址修改为自己的VIP地址，然后响应给客户 端。此时报文的源IP为VIP，目标IP为CIP

162. LVS-NAT模型的特性**

RS应该使用私有地址，RS的网关必须指向DIP

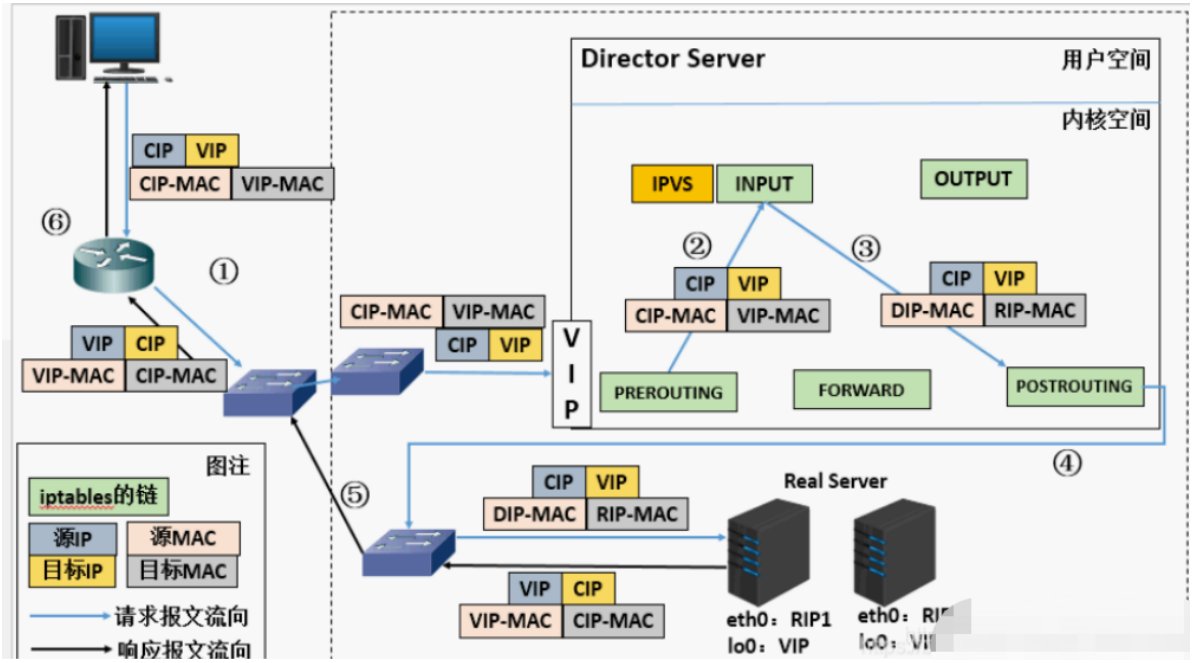
DIP和RIP必须在同一个网段内

请求和响应报文都需要经过Director Server，高负载场景中，Director Server易成为性能瓶颈

支持端口映射

RS可以使用任意操作系统 缺陷：对Director Server压力会比较大，请求和响应都需经过director server

163. LVS-DR模式原理**



(a) 当用户请求到达Director Server，此时请求的数据报文会先到内核空间的PREROUTING链。此 时报文的源IP为CIP，目标IP为VIP

(b) PREROUTING检查发现数据包的目标IP是本机，将数据包送至INPUT链

(c) IPVS比对数据包请求的服务是否为集群服务，若是，将请求报文中的源MAC地址修改为DIP的 MAC地址，将目标MAC地址修改RIP的MAC地址，然后将数据包发至POSTROUTING链。此时的 源IP和目的IP均未修改，仅修改了源MAC地址为DIP的MAC地址，目标MAC地址为RIP的MAC地址

(d) 由于DS和RS在同一个网络中，所以是通过二层来传输。POSTROUTING链检查目标MAC地址为 RIP的MAC地址，那么此时数据包将会发至Real Server。

(e) RS发现请求报文的MAC地址是自己的MAC地址，就接收此报文。处理完成之后，将响应报文通 过lo接口传送给eth0网卡然后向外发出。此时的源IP地址为VIP，目标IP为CIP

(f) 响应报文最终送达至客户端

164. LVS-DR模型的特性**

特点1：保证前端路由将目标地址为VIP报文统统发给Director Server，而不是RS

RS可以使用私有地址；也可以是公网地址，如果使用公网地址，此时可以通过互联网对RIP进行直接访问

RS跟Director Server必须在同一个物理网络中

所有的请求报文经由Director Server，但响应报文必须不能进过Director Server

不支持地址转换，也不支持端口映射

RS可以是大多数常见的操作系统

RS的网关绝不允许指向DIP(因为我们不允许他经过director)

RS上的lo接口配置VIP的IP地址

缺陷：RS和DS必须在同一机房中

165. LVS三种负载均衡模式的比较**

三种负载均衡：nat，tunneling，dr

|类目|NAT|TUN|DR|

|--|--|--|--|

操作系统|任意|支持隧道|多数（支持non-arp）

|服务器网络|私有网络|局域网/广域网|局域网

|服务器数目|10-20|100|大于100

|服务器网关|负载均衡器|自己的路由|自己的路由|

效率|一般|高|最高

166. LVS的负载调度算法**

- 轮叫调度
- 加权轮叫调度
- 最小连接调度
- 加权最小连接调度
- 基于局部性能的最少连接
- 带复制的基于局部性能最小连接
- 目标地址散列调度
- 源地址散列调度

167. LVS与nginx的区别**

vs的优势（互联网老辛）：

抗负载能力强，因为lvs工作方式的逻辑是非常简单的，而且工作在网络的第4层，仅作请求分 发用，没有流量，所以在效率上基本不需要太过考虑。lvs一般很少出现故障，即使出现故障 一般也是其他地方（如内存、CPU等）出现问题导致lvs出现问题。

配置性低，这通常是一大劣势同时也是一大优势，因为没有太多的可配置的选项，所以除了增减 服务器，并不需要经常去触碰它，大大减少了人为出错的几率。

工作稳定，因为其本身抗负载能力很强，所以稳定性高也是顺理成章的事，另外各种lvs都有完整 的双机热备方案，所以一点不用担心均衡器本身会出什么问题，节点出现故障的话，lvs会自动判 别，所以系统整体是非常稳定的。

无流量，lvs仅仅分发请求，而流量并不从它本身出去，所以可以利用它这点来做一些线路分流之 用。没有流量同时也保住了均衡器的IO性能不会受到大流量的影响。

lvs基本上能支持所有应用，因为lvs工作在第4层，所以它可以对几乎所有应用做负载均衡，包括 http、数据库、聊天室等。

nginx与LVS的对比：

nginx工作在网络的第7层，所以它可以针对http应用本身来做分流策略，比如针对域名、目录结构 等，相比之下lvs并不具备这样的功能，所以nginx单凭这点可以利用的场合就远多于lvs了；但 nginx有用的这些功能使其可调整度要高于lvs，所以经常要去触碰，由lvs的第2条优点来看，触碰 多了，人为出现问题的几率也就会大。

nginx对网络的依赖较小，理论上只要ping得通，网页访问正常，nginx就能连得通，nginx同时还 能区分内外网，如果是同时拥有内外网的节点，就相当于单机拥有了备份线路；lvs就比较依赖于网 络环境，目前来看服务器在同一网段内并且lvs使用direct方式分流，效果较能得到保证。另外注 意，lvs需要向托管商至少申请多于一个ip来做visual ip。

nginx安装和配置比较简单，测试起来也很方便，因为它基本能把错误用日志打印出来。lvs的安装 和配置、测试就要花比较长的时间，因为同上所述，lvs对网络依赖性比较大，很多时候不能配置成 功都是因为网络问题而不是配置问题，出了问题要解决也相应的会麻烦的多。

nginx也同样能承受很高负载且稳定，但负载度和稳定度差lvs还有几个等级：nginx处理所有流量 所以受限于机器IO和配置；本身的bug也还是难以避免的；nginx没有现成的双机热备方案，所以 跑在单机上还是风险比较大，单机上的事情全都很难说。

nginx可以检测到服务器内部的故障，比如根据服务器处理网页返回的状态码、超时等等，并且会 把返回错误的请求重新提交到另一个节点。目前lvs中ldirectd也能支持针对服务器内部的情况来监 控，但lvs的原理使其不能重发请求。比如用户正在上传一个文件，而处理该上传的节点刚好在上传 过程中出现故障，nginx会把上传切到另一台服务器重新处理，而lvs就直接断掉了。

两者配合使用：

nginx用来做http的反向代理，能够upsteam实现http请求的多种方式的均衡转发。由于采用的是异步转发可以做到如果一个服务器请求失败，立即切换到其他服务器，直到请求成功或者最后一台服务器失败为止。这可以最大程度的提高系统的请求成功率。

lvs采用的是同步请求转发的策略。这里说一下同步转发和异步转发的区别。同步转发是在lvs服务器接收到请求之后，立即redirect到一个后端服务器，由客户端直接和后端服务器建立连接。异步转发是nginx在保持客户端连接的同时，发起一个相同内容的新请求到后端，等后端返回结果后，由nginx返回给客户端。

进一步来说：当做为负载均衡服务器的nginx和lvs处理相同的请求时，所有的请求和响应流量都会经过nginx；但是使用lvs时，仅请求流量经过lvs的网络，响应流量由后端服务器的网络返回。

也就是，当作为后端的服务器规模庞大时，nginx的网络带宽就成了一个巨大的瓶颈。

但是仅仅使用lvs作为负载均衡的话，一旦后端接受到请求的服务器出了问题，那么这次请求就失败了。但是如果在lvs的后端在添加一层nginx（多个），每个nginx后端再有几台应用服务器，那么结合两者的优势，既能避免单nginx的流量集中瓶颈，又能避免单lvs时一锤子买卖的问题。

168. 负载均衡的作用有哪些？**

1、转发功能

按照一定的算法【权重、轮询】，将客户端请求转发到不同应用服务器上，减轻单个服务器压力，提高系统并发量。

2、故障移除

通过心跳检测的方式，判断应用服务器当前是否可以正常工作，如果服务器期宕掉，自动将请求发送到其他应用服务器。

3、恢复添加

如检测到发生故障的应用服务器恢复工作，自动将其添加到处理用户请求队伍中。

169. nginx实现负载均衡的分发策略**

Nginx 的 upstream目前支持的分配算法：

1)、轮询 ——1：1 轮流处理请求（默认）

每个请求按时间顺序逐一分配到不同的应用服务器，如果应用服务器down掉，自动剔除，剩下的继续轮询。

2)、权重 ——you can you up 通过配置权重，指定轮询几率，权重和访问比率成正比，用于应用服务器性能不均的情况。

3)、ip_哈希算法 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应用服务器，可以解决session共享 的问题。

170. keepalived 是什么?

广义上讲是高可用，狭义上讲是主机的冗余和管理

Keepalived起初是为LVS设计的，专门用来监控集群系统中各个服务节点的状态，它根据TCP/IP参考模型的第三、第四层、第五层交换机制检测每个服务节点的状态，如果某个服务器节点出现异常，或者工作出现故障，Keepalived将检测到，并将出现的故障的服务器节点从集群系统中剔除，这些工作全部是自动完成的，不需要人工干涉，需要人工完成的只是修复出现故障的服务节点。

后来Keepalived又加入了VRRP（VritrualRouterRedundancyProtocol，虚拟路由冗余协议）的功能，VRRP出现的目的是解决静态路由出现的单点故障问题，通过VRRP可以实现网络不间断稳定运行，因此Keepalvied一方面具有服务器状态检测和故障隔离功能，另外一方面也有HAcluster功能。

所以，keepalived的核心功能就是健康检查和失败且换。所谓的健康检查，就是采用tcp三次握手，icmp请求，http请求，udp echo请求等方式对负载均衡器后 面的实际的服务器(通常是承载真实业务的服务器)进行保活；

而失败切换主要是应用于配置了主备模式的负载均衡器，利用VRRP维持主备负载均衡器的心跳，当主负载均衡器出现问题时，由备负载均衡器承载对应的业务，从而在最大限度上减少流量损失，并提供服务的稳定性

171. 你是如何理解VRRP协议的

为什么使用VRRP?

主机之间的通信都是通过配置静态路由或者(默认网关)来完成的，而主机之间的路由器一旦发生故障，通信就会失效，因此这种通信模式当中，路由器就成了一个单点瓶颈，为了解决这个问题，就引入了VRRP协议。

VRRP协议是一种容错的主备模式的协议，保证当主机的下一跳路由出现故障时，由另一台路由器来代替出现故障的路由器进行工作，通过VRRP可以在网络发生故障时透明的进行设备切换而不影响主机之间的数据通信。

VRRP的三种状态：

VRRP路由器在运行过程中有三种状态：

Initialize状态：系统启动后就进入Initialize，此状态下路由器不对VRRP报文做任何处理；

Master状态；

Backup状态；一般主路由器处于Master状态，备份路由器处于Backup状态。

172. keepalived的工作原理?

keepalived采用是模块化设计，不同模块实现不同的功能。

keepalived主要有三个模块，分别是core、check和vrrp。

core：是keepalived的核心，负责主进程的启动和维护，全局配置文件的加载解析等

check：负责healthchecker(健康检查)，包括了各种健康检查方式，以及对应的配置的解析包括LVS的配置解析；可基于脚本检查对IPVS后端服务器健康状况进行检查

vrrp：VRRPD子进程，VRRPD子进程就是来实现VRRP协议的

Keepalived高可用对之间是通过 VRRP进行通信的， VRRP是通过竞选机制来确定主备的，主的优先级高于备，因此，工作时主会优先获得所有的资源，备节点处于等待状态，当主宕机的时候，备节点就会接管主节点的资源，然后顶替主节点对外提供服务

在Keepalived服务对之间，只有作为主的服务器会一直发送 VRRP广播包,告诉备它还活着，此时备不会抢占主，当主不可用时，即备监听不到主发送的广播包时，就会启动相关服务接管资源，保证业务的连续性.接管速度最快

173. 出现脑裂的原因

什么是脑裂?

在高可用（HA）系统中，当联系2个节点的“心跳线”断开时，本来为一整体、动作协调的HA系统， 就分裂成为2个独立的个体。

由于相互失去了联系，都以为是对方出了故障。两个节点上的HA软件像“裂脑人”一样，争抢“共享 资源”、争起“应用服务”，就会发生严重后果。共享资源被瓜分、两边“服务”都起不来了；或者两边 “服务”都起来了，但同时读写“共享存储”，导致数据损坏。

都有哪些原因导致脑裂?

高可用服务器对之间心跳线链路发生故障，导致无法正常通信。

因心跳线坏了（包括断了，老化）。

因网卡及相关驱动坏了，ip配置及冲突问题（网卡直连）

因心跳线间连接的设备故障（网卡及交换机）

因仲裁的机器出问题（采用仲裁的方案）

高可用服务器上开启了 iptables防火墙阻挡了心跳消息传输。

高可用服务器上心跳网卡地址等信息配置不正确，导致发送心跳失败

其他服务配置不当等原因，如心跳方式不同，心跳广插冲突、软件Bug等。

174. 如何解决keepalived脑裂问题?

在实际生产环境中，我们从以下方面防止脑裂：

同时使用串行电缆和以太网电缆连接、同时使用两条心跳线路，这样一条线路断了，另外一条还是好的，依然能传送心跳消息。

当检查脑裂时强行关闭一个心跳节点（这个功能需要特殊设备支持，如stonith、fence）相当于备 节点接收不到心跳消息，通过单独的线路发送关机命令关闭主节点的电源，做好对脑裂的监控报警。

解决常见方案：

如果开启防火墙，一定要让心跳消息通过，一般通过允许IP段的形式解决。可以拉一条以太网网线或者串口线作为主被节点心跳线路的冗余，开发检测程序通过监控软件检测脑裂。