面试题总结

1.不能关机的情况下，如果挂载目录卸载不掉应该怎么办

2.如果你向文件系统中写入数据，发现无法写入，怎么处理

3.客户反映服务器卡顿，你会怎么排查

4.如果你在执行命令时，发现Linux系统命令执行速度很慢，你会怎么解决

5.了解七层模型吗，说一下；TCP协议和UDP协议的区别是什么；二层和三层的交换机有什么区别，说一下

6.说一下TCP三次握手和四次挥手

7.为什么四次挥手时要有一个TIME_WAIT状态

8.为什么连接时是三次握手，断开时需要四次挥手

9.软链接和硬链接有什么区别，怎么做链接

10.Raid知道吗，说一下软raid和硬raid的区别，说一下常见的Raid的原理

11.监管Linux系统资源使用哪个命令

12.CentOS操作系统的启动过程

13.你在工作中如何修改Apache的主页文件；在工作中会用nginx吗，说一下nginx和apache的区别

14.说一下代理和负载均衡的区别

15.lvs和nginx都能做负载均衡，那么二者有什么区别

16.说一下lvs的工作原理

17.介绍一下lvs常见的工作模式

18.lvs的常用算法

19.说一下Apache的三种工作模式

20.说一下Nginx的工作原理（Nginx为什么消耗资源少，并发能力强）

21.Nginx和Tomcat怎么进行数据传输

22.为什么在搭建web服务器时要做动静分离，什么是动静分离

23.介绍一下Nginx的常用模块

24.说一下Nginx做反向代理时常用的算法有哪些

25.说一下你知道的Nginx的特性

26.Nginx常用的参数优化

27.说一下location模块（路由功能）的匹配优先级

28.解释一下数据库的主从复制的原理（或者会问：怎么做的数据库的数据同步）

29.数据库主从不同步的原因

30.解释一下数据库的读写分离

31.如果数据库主库宕机了，怎么切换到从库，如果有多个从库，其余的从库如何处理

32.解释一下事务的四个特征，和什么是数据库的回滚

33.MySQL数据库的备份方式

34.MySQL的历史了解吗，它和Mariadb有什么关系
35.数据库常用的引擎有什么，说一下区别
36.了解非关系型数据库吗，和MySQL有什么区别
37.解释一下数据库的锁机制
38.说一下高可用负载均衡集群，原理是什么
39.什么是DNS，说一下DNS的解析过程，以及使用的端口，解释一下智能DNS（分离解析）
40.怎么添加DNS解析
41.说一下常用服务的端口号
42.路由器的两种工作模式，怎么获取动态路由（DHCP原理）
43.说一下常见的网络状态码
44.解释一下502是什么意思，说一下你的排错思路
45.解释一下http协议，和https协议有什么区别
46.SSH远程无法连接，说一下你的检查思路
47.说一下Linux系统中查看系统资源都能使用哪些命令
48.列出Linux中常用的文本处理命令
49.Linux系统磁盘的相关命令
50.说一下df和du的区别
51.Linux系统如何进行调优
52.Linux查看内核版本使用什么命令
53.Linux中你认为哪些目录比较重要
54.统计文件行数使用哪个命令
55.解释一下B/S和C/S结构
56.简单说一下rpm的用法，rpm安装和yum安装的区别是什么
57.说一下防火墙的四表五链
58.swap分区的作用
59.怎么查看系统中的进程和线程
60.buffer和cache的区别
61.如何查看系统的资源限制
62.说一下ulimit命令
63.怎么查看当前的系统环境变量，怎么查看所有的系统环境变量
64.程序自动启动有哪几种方式
65.用过抓包命令吗，说一下怎么指定网卡，怎么指定所有网卡，-nn和-vv什么含义
66.接触过redis集群吗，说一下redis集群的三种实现方式，原理是什么
67.服务器内存不够怎么处理
68.CPU资源不足怎么处理
69.遇到磁盘空间不足的报错，你应该如何处理

70.如何将新磁盘扩展到已有分区

71.Linux如何开机自启

72.说一下文件描述符对系统有什么影响

73.Linux怎么挂载新磁盘

74.找到修改30天以上的文件并删除

75.服务器系统日志放在哪里

76.如果你的一块磁盘中，只有几个小文件，使用df命令查看，发现磁盘是满的，什么原因

怎么解决

77.查看操作系统的版本信息

78.解释一下什么是HAC的脑裂，能说一下解决方案吗

79.Linux进程有哪些状态

80.Linux查看系统资源的命令

81.Linux查看服务运行端口的命令

82./etc/profile和.bash_profile有什么区别

83.数据库的锁是如何制定的

84.MySQL为什么会出现锁表的情况，怎么解锁，如何避免这种情况的出现

85.innodb引擎是插入快还是读取快

86.Docker和虚拟机的区别

87.Dockerfile文件

88.Linux中常用的网络命令

89.NetworkManager

90.简单介绍一下docker定义

91.KVM和docker有什么区别

92.为什么说docker是轻量级

93.Docker常见的网络模式

94.Docker中Namespaces和Cgroups的隔离类型

95.Docker如何缩容

96.kubernetes中组件和插件的功能

97.介绍一下kubernetes集群中的控制器

98.pod的分类

99.python删除文件

100.ansible任务执行模式

101.ansible 简介

102.ansible 执行命令过程

103.简述ETCD及其特点

104.简述ETCD适应的场景

105.简述什么是Kubernetes

107.简述Kubernetes和Docker的关系

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

109.简述Kubernetes常见的部署方式

110.简述Kubernetes如何实现集群管理？

111.简述Kubernetes的优势、适应场景及其特点？

113.简述Kubernetes相关基础概念？

113.简述Kubernetes集群相关组件？

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

115.简述kube-proxy作用？

116.简述kube-proxy iptables原理

117.简述kube-proxy ipvs原理

118.简述kube-proxy ipvs和iptables的异同？

119.简述Kubernetes中什么是静态Pod？

120.简述Kubernetes中Pod可能位于的状态？

120.简述Kubernetes创建一个Pod的主要流程？

120.简述Kubernetes中Pod的重启策略？

120.简述Kubernetes中Pod的健康检查方式？

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

120.简述Kubernetes Pod的常见调度方式？

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

120.简述Kubernetes deployment升级过程？

120.简述Kubernetes deployment升级策略

120.简述Kubernetes DaemonSet类型的资源特性？

120.简述Kubernetes自动扩容机制？

120.简述Kubernetes Service类型？

120.简述Kubernetes Service分发后端的策略？

120.简述Kubernetes Headless Service？

简述Kubernetes外部如何访问集群内的服务？

简述Kubernetes ingress？

简述Kubernetes镜像的下载策略？

简述Kubernetes的负载均衡器？

简述Kubernetes各模块如何与API Server通信？

简述Kubernetes Scheduler作用及实现原理？

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

简述Kubernetes kubelet的作用？

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

简述Kubernetes如何保证集群的安全性？

简述Kubernetes准入机制？

简述Kubernetes RBAC及其特点（优势）？
简述Kubernetes Secret作用？
简述Kubernetes Secret有哪些使用方式？
简述Kubernetes PodSecurityPolicy机制？
简述Kubernetes PodSecurityPolicy机制能实现哪些安全策略？
简述Kubernetes网络模型？
简述Kubernetes CNI模型？
简述Kubernetes网络策略？
简述Kubernetes网络策略原理？
简述Kubernetes中flannel的作用？
简述Kubernetes Calico网络组件实现原理？
简述Kubernetes共享存储的作用？
简述Kubernetes数据持久化的方式有哪些？
简述Kubernetes PV和PVC？
简述Kubernetes PV生命周期内的阶段？
简述Kubernetes所支持的存储供应模式？
简述Kubernetes CSI模型？
简述Kubernetes Worker节点加入集群的过程？
简述Kubernetes Pod如何实现对节点的资源控制？
简述Kubernetes Requests和Limits如何影响Pod的调度？
简述Kubernetes Metric Service？
简述Kubernetes中，如何使用EFK实现日志的统一管理？
简述Kubernetes如何进行优雅的节点关机维护？
简述Kubernetes集群联邦？
简述Helm及其优势？

port
nodePort
targetPort

面试题总结

1.不能关机的情况下，如果挂载目录卸载不掉应该怎么办

umount卸载不掉是因为正在被程序读取或者有数据正在写入

- 使用 `fuser` 或者 `lsof` 查看正在占用的磁盘的进程PID
- kill -9 PID强制结束进程
- umount卸载

- 如果kill -9无法杀死进程，可以使用**umount -f强制进行卸载**，但是这种方法是最不推荐的

2.如果你向文件系统中写入数据，发现无法写入，怎么处理

此时应该考虑两种情况：

- 磁盘空间被占满：此时可以删除一些垃圾数据；或者将一些历史数据导出，将磁盘中的数据进行删除；如果在安装文件系统时，底层使用了LVM，可以进行在线扩容，增加硬盘容量
- inode号被用完：建议导出一些历史数据，然后将硬盘中的数据删除

3.客户反映服务器卡顿，你会怎么排查

1. 查看服务器的内存和CPU，确定内存大小和CPU核数

```
cat /proc/meminfo | grep MemTotal

cat /proc/cpuinfo | greo "physical id" | sort | uniq | wc -l
cat /proc/cpuinfo | grep "cpu cores" | uniq
```

2. 查看CPU负载，确认负载高是因为CPU的使用、内存的使用还是io的使用引起的

   使用 `top` 命令进行查看

3. 查看数据库

```
show full processlist;  #查看数据库当前正在使用的线程
```

   通过这个命令可以查看当前登录到数据库的用户、用户的状态和执行的sql语句，通过这些能够分析操作和sql语句是否有误

4.如果你在执行命令时，发现Linux系统命令执行速度很慢，你会怎么解决

- lscpu：查看CPU的相关信息，看CPU的缓存是否过高
- free -mh：查看系统磁盘文件缓存是否过大
- top：查看系统负载是否过高
- 主要就是，查看系统中的CPU、负载等情况；检查系统是否在执行定时计划任务等；也可以选择杀死进程，但是杀死进程前需要根据占用过高的进程做进一步的分析

5.了解七层模型吗，说一下；TCP协议和UDP协议的区别是什么；二层和三层的交换机有什么区别，说一下

- 七层模型：OSI/ISO七层模型
  - 应用层：给用户一个操作界面，为用户提供服务
  - 表示层：为数据提供加密、压缩、表示
  - 会话层：确定数据是否需要进入网络进行传输
  - 传输层：
    - 对报文进行分组、组装
    - 提供传输协议的选择（TCP协议和UDP协议）
    - 差错校验
  - 网络层：IP地址编址
  - 数据链路层：MAC地址编址
  - 物理层：数据实际传输和电气特效定义
- TCP协议和UDP协议的区别
  - TCP协议：面向连接的、可靠的传输协议，主要用来传输字节流
  - UDP协议：面向无连接的、不可靠的传输协议，主要用来传输数据报文
  - TCP协议在连接过程中需要进行三次握手，会有延时性，连接过程中容易被攻击，安全性较差，但是可靠性高；UDP协议因为是无连接的，实时性好，安全性高，可靠性低
- 二、三层交换机的区别
  - 二层交换机：二层使用的是MAC地址寻址，采用存储转发的方式进行数据交换；
  - 三层交换机：三层交换机具有一定的路由转发功能，可以连接外网；还可以选择路由转发的线路，提高转发的效率

6.说一下TCP三次握手和四次挥手

- 三次握手：
  - 第一次握手：客户端将标志位SYN置为1，随机产生一个序列号Seq，将这个数据包发送给服务器，然后客户端进入SYN_Sent状态。
  - 第二次握手：服务器收到客户端发送的数据包后，会把SYN和ACK（确认序号有效）都置为1，将Ack（确认号）记为Seq+1，然后再随机产生一个序列号Seq，将数

据包返回给客户端，然后进入SYN_RCVD状态
    - 第三次握手：客户端收到数据包后，确认Ack为自己产生的序列号Seq+1、ACK=1，将SYN和ACK置为1，将Ack记为服务器端产生的序列号Seq+1，将数据包发送给服务器，服务器端进行确认，建立连接，此时客户端和服务器端进入ESTABLISHED状态。

- 四次挥手：

    - 第一次挥手：客户端发送一个标志位为FIN、随机产生一个序列号Seq，将数据包发送给服务器端，请求断开连接，进入FIN_WAIT_1状态
    - =第二次挥手：服务器端收到客户端的数据包，发送一个标志位为ACK、产生一个序列号Ack为Seq+1，将数据包发送给客户端，进入CLOSE_WAIT状态
    - 第三次挥手：服务器端发送一个标志位为FIN、Ack为上一次挥手的Seq+1、随机产生一个序列号Seq，将数据包发送给客户端，请求断开连接，进入LAST_ACK状态
    - 第四次挥手：收到FIN包后，客户端进入TIME_WAIT状态，发送一个标志位为ACK、Ack为服务器产生的随机序列号Seq+1数据包给服务器端，服务器端进入CLOSED状态

7.为什么四次挥手时要有一个TIME_WAIT状态

在理想状态下，四次报文发送完成后，就可以断开连接，但是网络是有丢包的可能性存在的，在数据包发送完成后，客户端会进入TIME_WAIT状态，这个时间是2MSL，一个MSL就是数据报文在网络中存活的最长时间，如果在2MSL时间内，客户端再次收到了服务器端的FIN数据包，那么这个时间就会被重置，如果在这个时间内客户端没有再次收到FIN数据包，那么我们就可以认为数据包被服务器端接收了，我们可以断开连接

8.为什么连接时是三次握手，断开时需要四次挥手

服务器端收到客户端发送的SYN报文后，可以同时发送SYN+ACK报文，所以是三次握手；在断开连接时，客户端发送完FIN报文后，只代表客户端不能发送数据了，但是还能接收数据。所以当服务器端收到FIN报文后，会先回复一个ACK报文，通知客户端我收到了，只有等服务器端所有的数据都发送完成，才会发送FIN报文断开连接

9.软链接和硬链接有什么区别，怎么做链接

- 我们可以使用 `ln` 制作硬链接，使用 `ln -s` 制作软链接
- 软链接：软链接只是一个路径，可以类似于Windows系统中的快捷方式，依赖于源文件存在，源文件如果被删除，那么链接也会失效；软链接可以跨区创建；可以对目录创建
- 硬链接：硬链接和源文件具有同一个inode号，是同一个文件，修改源文件硬链接也会被修改，删除源文件硬链接仍然可以使用；硬链接不能对目录创建（会引入循环，破坏

目录树结构）；不能跨区创建

10.Raid知道吗，说一下软raid和硬raid的区别，说一下常见的Raid的原理

- Raid：磁盘阵列，提供比单个磁盘更多的存储和冗余存储的技术
- 软raid：由操作系统模拟出的raid，硬盘损坏，raid就会失效
- 硬raid：由硬件raid卡制作的raid，硬盘损坏不会导致raid损坏，能够保证数据冗余
- Raid原理：
  - Raid0：是一种条带形式的存储，由两块或两块以上的磁盘组成，数据在写入时会以条带的形式写入所有的磁盘，存储速度最快，但是损坏几率最高，没有磁盘容错功能
  - Raid1：镜像存储，必须由2的倍数块硬盘组成，数据再写入磁盘时会同时复制到另一块硬盘中，因此写入速度最慢，磁盘利用率只有50%，具有磁盘容错
  - Raid10：硬盘先组成raid1再组成raid0，中和Raid1和Raid0的优缺点
  - Raid01：硬盘先组成raid0再组成raid1
  - Raid5：至少需要三块硬盘，每次写入数据时都会写入一个校验信息，每个校验信息都会被写入到不同的硬盘下，具有磁盘容错和冗余功能，磁盘利用率为n-1块盘

11.监管Linux系统资源使用哪个命令

dstat，默认未安装，需要安装。用来替换vmstat、iostat、netstat、nfsstat和ifstat这些命令的工具。

12.CentOS操作系统的启动过程

- CentOS6：开机，加电自检，加载分区表，加载grub菜单（`/boot/grub`），确定系统的第一个进程（`init`），确定系统的启动级别（`/etc/initab`），操作系统初始化（`/etc/rc.sysinit`），根据启动级别确定操作系统启动的服务项（`/etc/rcN.d`），加载环境变量（`/etc/bashrc,/etc/profile`）
- CentOS7：开机，加电自检，加载分区表，加载grub菜单（`/etc/grub2`），确定系统的第一个进程（`systemd`），确定系统的启动级别（`/etc/systemd/system/default.target`），操作系统初始化（`/usr/lib/systemd/system/default.target`），根据系统启动级别确定系统启动的服务项（`/usr/lib/systemd/system,/etc/systemd/system`），加载环境变量（`/etc/bashrc,/etc/profile`）

13.你在工作中如何修改Apache的主页文件；在工作中会用nginx吗，说一下nginx和apache的区别

- apache在安装完成后，有一个配置文件目录，yum安装在`/etc/httpd/conf/httpd.conf`，源码包安装会在`/usr/local/apache/conf/http.conf`，在配置文件中有`Directory Index`模块，找到后取消注释，在后面添加需要主页文件类型即可。
- 区别
  - Nginx是一种轻量级的web服务器，比Apache占用的资源更少
  - Nginx的工作模式是异步非阻塞模式，而Apache是同步阻塞型的工作模式，所以Nginx能抗高并发
  - Nginx有高度模块化的设计，编写模块比Apache简单

14.说一下代理和负载均衡的区别

- 代理分为正向代理和反向代理
  - 正向代理：为客户端处理请求，将客户端要请求的数据从后台服务器进行下载然后返回给客户端。客户端认为自己访问的是网站，实际上访问的是代理服务器的IP
  - 反向代理：通常和负载均衡搭配使用，接收所有客户端的请求，然后将请求交给后台的服务器进行处理，降低后台服务器的处理压力，后台一般会有多台服务器，通常会和负载均衡搭配使用
- 负载均衡：负载均衡只有在后台服务器大于等于两台时才会有意义

15.lvs和nginx都能做负载均衡，那么二者有什么区别

- lvs工作在四层，只处理请求，没有网络流量；nginx工作在七层，支持http的操作，因此lvs抗压能力比nginx强
- lvs的稳定性比nginx高，因为nginx没有现成的双机热备方案
- nginx的配置难度低，只要修改配置文件打开upstream模块；lvs需要搭建集群，还需要在iptables内写入路由转发规则
- lvs工作在四层，支持的软件比nginx多
- nginx访问失败会返回状态码；lvs会直接断开
- nginx自带健康检查功能，如果后台服务器宕机，那么nginx不会向这台服务器上分发请求
- lvs本身是不支持正则匹配的，不能做动静分离
- nginx可以跨平台运行，lvs只能在linux操作系统上有运行
- nginx对网络的依赖性比较低，理论上只要能ping通，就能进行访问；lvs对网络的依赖性比较高

16.说一下lvs的工作原理

lvs主要通过ipvs（钩子函数）实现功能，而ipvs主要工作在防火墙的input链上，用户可以使用ipvsadm对lvs进行管理

17.介绍一下lvs常见的工作模式

- DR（路由）

  负载均衡服务器和后台的真实服务器位于同一局域网内，客户端通过路由器访问负载均衡服务器，负载均衡服务器会进行MAC地址转换，因为负载均衡服务器和后台服务器位于同一网络内，可以直接将请求交给后台服务器进行处理，后台服务器会直接把数据返回给客户端

- NAT

  负载均衡服务器和真实服务器位于同一局域网内，客户端访问负载均衡服务器，负载均衡服务器会进行目标地址转换，将访问请求交给后台的服务器；当后台服务器返回数据时，负载均衡服务器会进行源地址转换，将数据交给客户端

- TUN（隧道）

  隧道模式的处理过程和DR模式相似，负载均衡服务器只会处理入站的报文，会将请求交给距离你最近的服务器进行处理，然后服务器直接返回数据，提高了访问速度

18.lvs的常用算法

- rr（Round Robbin）：按照顺序将请求轮流分配到后台服务器上
- wrr（Weight Round Robbin）：按照权重向后台服务器分配请求，权重越高，分配的请求越多
- 最小连接调度（Least Connetion）：lvs会优先把请求分配到连接数最少的后台服务器上
- 加权最小连接调度（Weight Least Connection）：具有较高权重的服务器会优先处理请求，同时负载调度器会根据后台服务器的负载情况，动态的调整后台服务器的权值
- 基于局部的最少连接（Locality-Based Least Connections）：根据请求的目标IP地址找出这个IP地址最近使用的一台服务器，如果这台服务器存在且未超载，将请求交给这台服务器去处理；如果服务器不存在，或者超载且有服务器处于一半的负载，那么就按照最小连接原则，从剩余的服务器中找到一台服务器来处理

- 带复制的基于局部的最少连接（Locality-Based Least Connections with Replication）：和基于局部的最少连接不同之处在于，它是一个IP对应一个服务器组；而基于局部的最少连接是一个IP对应一个服务器。根据请求的目标IP地址找出这个IP地址最近使用的一个服务器组，按最小连接原则在服务器组中找到一台服务器进行处理，如果这台服务器未超载，由这台服务器进行处理；如果超载，就从集群中选择一台服务器进行处理，并将这台服务器加入这个服务器组

- 目标地址散列（Destination Hashing）：先根据请求的IP地址，作为散列键从静态分配的散列表里找出对应的服务器，如果服务器未超载，将请求交给服务器进行处理，否则返回空

- 源地址散列调度（Source Hashing）：先根据请求的源IP地址，作为散列键从静态分配的散列表里找出对应的服务器，如果服务器未超载，将请求交给服务器进行处理，否则返回空

- 最短的期望的延迟（Shortest Expected Delay）：基于加权最小连接调度，但是还会进行一步运算：计算公式

  $$(1+n)/n, \text{其中} n \text{为权重}$$

  然后将请求交给数值最小的服务器进行处理

- 最少队列调度（Never Queue）：无需队列，如果有realserver的连接数等于0就直接分配过去，不需要在进行SED运算。

19.说一下Apache的三种工作模式

- prefork：单进程单线程模式，同一时间内只能处理一个请求。优点是运行稳定，不需要担心线程安全的问题；但是处理请求时单线程独占资源空间，会造成资源的浪费
- worker：单进程多线程模式，在同一时间内能够同时处理多个用户请求
- event：单进程多线程模式，event要比worker模式优秀，因为event工作模式下允许线程在处理完请求后将资源释放去继续处理另一个请求

20.说一下Nginx的工作原理（Nginx为什么消耗资源少，并发能力强）

异步非阻塞：即当系统内没有资源处理请求时，它会将请求挂起并注册一个事件，然后将这部分资源释放掉，当系统有多余的线程时再来处理挂起的请求，这样能够实现在资源固定的情况下处理更大的并发请求

21.Nginx和Tomcat怎么进行数据传输

- 如果是处理静态资源，Nginx直接处理返回给用户
- 如果处理动态资源，Nginx会将请求交给后台的Tomcat服务器进行处理，然后返回给用户

22.为什么在搭建web服务器时要做动静分离，什么是动静分离

- 动静分离：根据一定的规则将不变的资源和经常发生变化的资源进行分离，然后我们就可以将静态资源进行缓存。简单概括，就是把动态文件和静态文件进行分离
- 有一些请求需要后台进行处理，但是有一些请求是不需要后台进行处理的。如果不做动静分离，这些静态资源就会去向后台服务器发起请求，在并发较高时，会比较明显的导致响应时间增加；实现动静分离，能够明显提高用户访问静态资源的速度。

23.介绍一下Nginx的常用模块

- http_ssl_module：实现加密传输的模块
- http_proxy_module：实现反向代理
- http_upstream_module：负载均衡模块，通常和proxy模块搭配使用
- http_rewrite_module：重写模块，实现域名重写

24.说一下Nginx做反向代理时常用的算法有哪些

- rr：轮询算法，将用户的请求平均的分配到后台的服务器上，如果后台服务器宕机，会自动将宕机服务器清除
- wrr：加权轮询，根据权重分配请求，权重越大，处理的请求的越多
- ip_hash：根据请求的hash值进行分配，hash值相同的会被分配到同一台机器上
- fair：根据后台服务器的响应时间进行分配，响应时间短的优先分配，必须下载nginx的upstream_fair模块才能使用
- least_conn：哪台服务器的连接数最少，就分配到那台服务器上

25.说一下你知道的Nginx的特性

- 跨平台：可以在Linux和Windows上运行
- 配置简单
- 异步非阻塞的工作模式，能抗高并发
- 网络依赖性低，理论上只要能ping通，就能进行负载均衡，而且能够有效地区分内外网流量
- 自带健康检查，如果后台有一台服务器宕机，Nginx就不会向这台服务器上分配请求
- 支持gzip压缩，能够节省带宽
- 消耗内存少，稳定性高

26.Nginx常用的参数优化

- 隐藏版本信息
- 修改上传文件限制
- 控制客户端请求Nginx的速率
- 控制并发的连接数，防止ddos攻击
- 修改Nginx的默认用户
- 防盗链
- 开启ssl模块，使用https进行访问
- gzip压缩
- 动静分离，server端添加location模块

27.说一下location模块（路由功能）的匹配优先级

- 首先进行精确匹配 `=`
- 其次进行前缀匹配 `^~`
- 按文件中的顺序进行正则匹配
- 不带任何前缀的正则匹配
- 最后进行通用匹配
- 只要有一条匹配成功，就停止匹配

28.解释一下数据库的主从复制的原理（或者会问：怎么做的数据库的数据同步）

- 从服务器会向主服务器提交主机偏移位
- 从服务器的io线程会主动连接主服务器的io线程
- 主服务器会通过io线程将bin-log日志推送给从服务器，然后进入休眠状态
- 从服务器的io线程会唤醒sql线程，将bin-log日志转化为存储日志，进行数据同步，然后线程会进入休眠状态，等待下一次同步

29.数据库主从不同步的原因

- 网络延迟：数据库主从同步是基于binlog日志的异步复制，而binlog日志是通过网络进行传输的，网络延迟大时，可能会造成主从不同步
- 主从数据库负载不一致：主从同步时，主数据库需要一个io线程，而从服务器需要一个io线程和一个sql线程，当一台服务器负载过大时，会导致提供给线程的资源不足，主从同步就无法完成
- MySQL异常宕机
- 主从数据库设置的 `max_allowed_packet` 不一致，当主库执行一个较大的sql语句，从库无法成功执行，导致数据不同步
- 可能在数据同步过程中发生了bug

- MySQL主从数据库的版本不一致，如果主数据库的版本较高、从数据库版本较低，也会导致同步不成功

30.解释一下数据库的读写分离

如果被这么问到，不仅要解释什么是读写分离，还要说出数据库主从复制的原理，切忌只说读写分离，回答一定要完整

- 读写分离就是，主服务器负责写入数据，后台的一台或多台从服务器提供读操作
- 当主数据库写入数据时，从服务器会将数据同步过去
- **解释主从同步的原理**

31.如果数据库主库宕机了，怎么切换到从库，如果有多个从库，其余的从库如何处理

- 登录所有的从库，执行show slave status\G；查看Pos信息，选择Pos值最大的从库作为主库，执行stop slave命令
- 修改my.conf配置文件，开启bin-log日志，然后重启数据库，登录数据库执行restart master，修改对应服务器IP信息
- 登录其他从库，执行change master，查看状态

32.解释一下事务的四个特征，和什么是数据库的回滚

- 事务的四大特征：
  - 持久性：对数据库的所有操作被保存后将不能进行回滚
  - 原子性：对事务的操作要么全部被执行，要么全部都不执行
  - 一致性：事务的开启前后，数据库的完整性没有被破坏
  - 隔离性：同一时间内，同一个事务只允许一个请求；不同的事务不会受到干扰
- 数据库的回滚：对数据库进行操作后，如果操作过程中发生错误，那么事务将回到修改之前的状态

33.MySQL数据库的备份方式

- 增量备份
- 异地备份
- 重要的数据库会在设计架构时，搭建一个一摸一样的数据库做完整备份
- 通过命令行备份：mysqldump -uroot -p 数据库名 > *.sql文件
- 通过Rsync+Inotify进行冷备
- 设计架构时，会搭建数据库的集群，进行热备

34.MySQL的历史了解吗，它和Mariadb有什么关系

Mariadb是MySQL5.7版本之后的一种叫法，因为原来的MySQL在2009年被Oracle公司收购后，变成了收费版本，MySQL开发者重新开发了Mariadb作为开源项目继续进行更新和维护

35.数据库常用的引擎有什么，说一下区别

- MYISAM：性能优异，查询速度快；但是不支持事务、锁、外键约束等高级功能
- INNODB：支持事务、锁、外键约束等高级功能
- MEMORY：将数据储存在内存中，方便快速读写

36.了解非关系型数据库吗，和MySQL有什么区别

MySQL属于关系型数据库，和非关系型数据库的区别主要体现在以下几个方面：

- 存储方式：
  - 关系型数据库中的数据都是存储在表格中的，表与表之间能够通过能进行关联协作，在提取数据时很方便
  - 非关系型数据库是通过键值对进行存储的
- 存储结构：
  - 关系型数据库中，数据存储时的结构都是预定义好的，结构比较稳定，但是修改数据很困难
  - 非关系型数据库使用的是动态结构，很容易适应数据结构和内容的变化
- 查询方式：
  - 关系型数据库通过SQL语言进行数据查询，使用标准化的语言进行数据查看
  - 非关系型数据库的查询语言是非标准化的，使用的是UNQL
- 扩展
  - 关系型数据库的数据存储在多个表中，涉及到多表联合查询，需要提升服务器的硬件资源来进行扩展，是一种纵向扩展
  - 非关系型数据库的存储是分布式的，可以向地址池中增加服务器来扩展性能，是一种横向扩展

37.解释一下数据库的锁机制

数据库的锁机制是一种对数据的保护机制，因为数据库支持并发，所以可能存在同一时间内多个用户对同一个数据进行修改，这可能会破坏数据的一致性；当用户对数据进行修改时，系统会先对这个数据进行加锁，这就保证了事务对这个数据的控制，当这个数据被修改完成之前，其他的事务不能对这个数据进行修改，保证了数据的一致性。

38.说一下高可用负载均衡集群，原理是什么

如果被问到keepalived如何解决单节点故障，就回答原理

- 使用LVS做负载均衡集群，使用keepalived做高可用集群
- LVS负载均衡集群
  - LVS主要采用的是基于IP的负载均衡技术，它由两个核心组件构成：ipvs（钩子函数）、ipvsadm；其中，ipvs是LVS的核心组件，是一个框架，而ipvsadm是一个管理命令，用来定义路由转发规则，工作在用户空间中
- keepalived高可用集群
  - keepalived基于VRRP（虚拟路由冗余协议）来实现高可用功能，keepalived可以监控集群中各个节点的功能是否正常，如果节点异常或者服务故障，keepalived会将故障节点从集群中清除出去，在故障节点恢复正常后，keepalived会将节点重新拉入集群中。
  - VRRP（虚拟路由冗余协议）：是为了解决静态路由状态下出现的单节点路由故障问题。VRRP实际上是一个虚拟路由器集群，这个集群工作在实际的物理路由器之上，这些路由器中有一个master路由器，其余的路由器都是backup级别；只有master路由器不断的进行vrrp数据包的发送，而backup路由器只负责数据包的接收，同时负责监控master的运行状态，因此不会发生抢占的现象。如果master路由器出现故障，backup路由器无法接收到来自master服务器的数据包，那么就会认定master服务器故障，接着剩下的所有backup路由器会进行选举，优先级最高的backup会成为master，这个过程非常短，能够保证服务的持续性。

39.什么是DNS，说一下DNS的解析过程，以及使用的端口，解释一下智能DNS（分离解析）

- DNS：域名解析系统
- DNS解析过程：

    - 首先查看本地hosts文件，hosts文件无法解析则查看本地缓存，本地缓存无法解析则请求DNS服务器进行域名解析，如果是本地解析直接返回解析记录。这是DNS的递归
    - 首先请求根域服务器，返回一级域服务器的IP地址，然后请求一级域服务器返回二级域服务器的IP地址，这是DNS的迭代
    - 请求三级域服务器返回域名地址。DNS服务器将解析记录在本地记录后，将解析记录发送给客户端，完成访问。这是DNS的递归
    - 递归：直接返回域名地址；迭代：告诉你下一步怎么走，在这里就是获取DNS服务器地址

- 使用TCP/UDP53端口

- 智能DNS：DNS服务器会根据用户发起的查询自动判断用户所属的运营商，将请求交给相应的运营商进行解析，减少访问时间，提高访问速度

40.怎么添加DNS解析

```
vim /etc/resolv.conf
```

41.说一下常用服务的端口号

- DNS：TCP/UDP53端口
- DHCP：UDP67/68端口
- http：80
- https：443
- FTP：20/21
- ssh：22
- telnet远程连接：23
- SMTP邮件协议：25
- pop3：110
- IMAP：143
- Tomcat：8080
- php-fpm：9000
- mysql：3306

42.路由器的两种工作模式，怎么获取动态路由（DHCP原理）

- 静态路由static

- 动态路由DHCP（动态主机配置协议）

- DHCP租约过程

- 首先客户端在局域网内广播发送DHCP_Discover包，寻找DHCP服务器
- DHCP服务器响应，发送DHCP_Offer包
- 客户端选择IP，发送DHCP_Request包
- 服务器确定租约，发送DHCP_ACK包

43.说一下常见的网络状态码

http状态码主要分为5类：

- 1**：服务器收到请求，需要请求者继续进行后续操作
- 2**：操作被成功接收
- 3**：重定向，需要进一步操作完成请求
- 4**：客户端错误，主要指语法错误或者无法完成请求
- 5**：服务器端错误，服务器在处理请求的过程中出现错误

常见的错误状态码：

- 400：服务器不理解请求的语法
- 401：未授权，请求进行身份验证，登录需要授权的网站可能返回此提示
- 403：服务器拒绝请求，可能是服务器或主机拒绝了访问
- 404：服务器找不到页面，访问服务器上不存在的页面会返回这个提示
- 405：客户端请求中的方法被禁用
- 500：服务器内部错误，无法完成请求
- 502：服务器充当网关或者代理，从上游服务器收到了无效响应
- 503：服务器超载或暂停出现的状态码，通常只是暂时的
- 504：服务器充当网关或代理，从上游服务器收到的响应超时
- 505：服务器不支持http协议的请求，无法完成处理

44.解释一下502是什么意思，说一下你的排错思路

- 502：服务器充当网关或者代理，从上游服务器收到了无效响应。简单的说就是网关错误，web服务器通信失败

- 排错思路：

  - 因为是和web服务器通信失败，所以可以先检查网络，看网络是否畅通
  - 也可能是因为web服务器未启动，检查web服务器进程是否在运行状态，端口是否开放
  - web服务器处理的请求太多，无法响应这个请求，检查web服务器的日志来定位错误原因

45.解释一下http协议，和https协议有什么区别

- http协议：超文本传输协议，基于的是TCP/IP协议，是一种无连接（限制每次连接只处理一个请求，处理完成后立即断开）、无状态（服务器端对数据没有记忆能力）的协议，只能由客户端向服务器端请求数据，服务器端不会主动发送数据

- 区别

  - http是明文传输，容易被攻击，https是http协议+ssl加密组成，确保数据传输过程是加密进行的
  - http是无状态连接，https的连接过程是加密过的
  - https使用443端口连接，http使用80端口

46.SSH远程无法连接，说一下你的检查思路

主要从四个方面进行分析：网卡，端口，IP，防火墙

- 检查客户端网线连接是否正常，重新插一下网线，重新尝试远程连接
- 去机房检查网络连接、网卡是否正常，检查IP地址
- 检查防火墙是否开启，如果开启防火墙，查看防火墙规则中是否添加了远程连接的端口
- 查看IP安全策略中是否添加了需要远程连接的网段

47.说一下Linux系统中查看系统资源都能使用哪些命令

- netstat：能够查看进程和端口
- top、htop：查看系统负载；htop比top更加人性化，支持鼠标点击
- ps aux：查看所有进程，能够显示PID
- ps ef：查看使用的终端
- free：查看系统内存的使用情况
- w：查看系统负载，登录系统的用户和持续登录时间
- uptime：查看系统负载

48.列出Linux中常用的文本处理命令

- cat，less，more：查看文件内容；less和more支持翻页
- head：使用-n选项可以指定查看文件的前n行
- tail：动态查看文件内容
- grep：行匹配命令
- cut：列提取命令，可以指定分隔符，默认使用空格作为分隔符
- awk：基于行的处理命令
- sed：对文件中的内容进行删除、替换、增加的命令
- sort：排序命令

- uniq：取消重复行

49.Linux系统磁盘的相关命令

- df：查看文件系统的大小和使用情况
- du：查看目录及目录下文件的大小
- fdisk：centos6的磁盘分区命令
- gdisk：centos7的磁盘分区命令
- mount：挂载命令
- umount：卸载命令
- lsblk：列出系统中所有的块设备信息
- blkid：列出设备名称和UUID
- mke2fs：创建文件系统
- fsck：检测分区

50.说一下df和du的区别

- df主要是查看文件系统的大小
- du是用来查看文件大小的命令

51.Linux系统如何进行调优

- 配置时间服务器，进行时间同步
- 避免直接使用root用户，防止权限溢出
- 配置国内yum源
- 开启ssh服务，使用ssh密钥对登录，修改默认连接端口
- 备份系统中的重要文件，比
  如：/etc/passwd，/etc/shadow，/etc/group，/etc/gshadow
- 配置合理的SELinux上下文规则
- 配置合理的防火墙规则

52.Linux查看内核版本使用什么命令

uname -a

53.Linux中你认为哪些目录比较重要

- /etc：存放了系统中重要的配置文件
- /bin：存放了系统命令
- /usr/：系统安装时安装的软件默认安装在这个目录下
- /lib：存放系统程序需要调用的函数库

- /dev：设备文件保存位置
- /home：用户家目录

54.统计文件行数使用哪个命令

- wc -l：统计文件行数
- wc -c：统计文件字符数
- wc -w：统计文件字数

55.解释一下B/S和C/S结构

- B/S结构：浏览器/服务器架构，一般建立广域网上，通常面向的是不可知的用户群，对安全的控制能力相对较弱。有逻辑上相互分离的数据层、表示层和业务层构成。表示层向客户提供数据，业务层实施业务和数据规则， 数据层定义数据访问标准，三层体系结构中的核心是组件对象模型。
- C/S结构：客户端/服务器架构，一般建立在局域网上，通过专门的客户端、服务器进行数据传输和交换，安全性较好。是一个典型的两层架构，将任务合理的分配到客户端和服务端，降低了系统的通讯对资源的消耗，需要安装客户端才能进行操作。

56.简单说一下rpm的用法，rpm安装和yum安装的区别是什么

- rpm -ivh：安装rpm包，-i是安装，-v是显示详细信息，-h是以#号输出安装进度
- rpm -Uvh：升级安装，没有直接安装，有旧版本升级安装
- rpm -Fvh：升级安装，有旧版本升级安装，没有旧版本不安装
- rpm -e：卸载

rpm安装和yum安装最大的区别在于，yum安装会主动从yum仓库中下载所需要的依赖包；但是rpm安装不会主动解决依赖关系，需要提前将需要的依赖软件安装好

57.说一下防火墙的四表五链

- 四表，承载链
    - raw：数据报文跟踪
    - filter：数据报文过滤
    - mangle：数据报文修改
    - nat：路由转发，实现dnat（目标地址）和snat（源地址）转换
- 五链，承载规则
    - input：数据包入站应用此链中的规则
    - output：数据包出站应用此链中的规则
    - forward：处理数据包转发的报文请求

- prerouting：数据包做路由前选择，配合DNAT使用
- postrouting：数据包做路由后选择，配合SNAT使用

58.swap分区的作用

swap分区是一个交换分区，当物理内存不足时，swap分区会释放一部分内存给当前正在运行的程序使用。这部分空间一般来自那些很长时间没有运行过的程序，系统将这部分空间临时存放在swap空间中，给那些内存占有率比较高的程序使用。

59.怎么查看系统中的进程和线程

- 进程：`top，ps -aux，ps -ef`
- 线程：
  - `ps -aT`（查看系统中的所有线程）
  - `ps -T -p <PID>`：查看某个进程中的线程
  - `top -H -p <PID>`：查看进程中的线程

60.buffer和cache的区别

- buffer：缓冲，是为了提高内存和硬盘（或者其他io设备）之间的数据交换速度而设计的
- cache：缓存
  - 从CPU的角度来看，是为了提高内存和CPU之间的数据交换速度而设计的
  - 从内存读取和磁盘读取的角度来看，可以理解为系统为了更高的数据读取速率，将已经读取过的数据存放在内存中。

61.如何查看系统的资源限制

`ulimit -a`

62.说一下ulimit命令

- `ulimit`命令主要用来限制使用系统资源（内存、CPU等）的范围，是Linux系统的一个内置的bash命令
- `ulimit`设置项仅在当前shell才会生效，需要永久生效需要修改配置文件
- 将限制写入`~/.profile或者~/.bashrc`只对当前的用户生效
- 写入`/etc/security/limits.conf`下可针对性配置，在系统层面永久生效

- 调整相关硬限制值`(Hard Limit)`，设置一次后，下一次设置的值只能比上一次的值要小
- 如果不加S或者H修饰，默认同时修改硬限制和软限制

63.怎么查看当前的系统环境变量，怎么查看所有的系统环境变量

- 查看当前的系统变量：set
- 查看所有的系统环境变量：env
- set：显示当前shell定义的私有变量
- env：显示用户的环境变量
- export：显示当前导出成用户变量的shell变量，并显示变量属性

64.程序自动启动有哪几种方式

- 在CentOS7中，使用`systemctl enable 软件`，设置开机自启
- 在CentOS6中，使用`service 软件 enable`，设置开机自启
- 或者将命令写入`/etc/rc.d/rc.local`下，会在开机自动运行
- `/etc/rc.local是/etc/rc.d/rc.local的软链接`

65.用过抓包命令吗，说一下怎么指定网卡，怎么指定所有网卡，-nn和-vv什么含义

tcpdump可以将网络中传送的数据包的“头”完全截获下来提供分析，它支持针对网络层、协议、主机、网络或端口的过滤

- 包分析工具：tcpdump
- 指定网卡：-i
- 指定所有网卡：-i any
- -nn：禁止IP和端口的名称解析
- -vv：显示详细信息，v的个数越多，显示的信息越详细`(-v/-vv/-vvv)`

66.接触过redis集群吗，说一下redis集群的三种实现方式，原理是什么

- 主从模式：
  - slave端启动后，会主动向master端发送sync命令，master会拍摄快照保存到后台（RDB持久化），同时会缓存保存这段时间的命令，然后将快照和命令发送到slave端，slave端加载快照文件和缓存命令进行数据同步。
  - 如果master节点宕掉，那么redis集群就无法对外提供写服务。
- 哨兵模式：

    - 以主从模式作为基础，在集群加入监控服务器，监控服务器可以是一台也可以是一个集群。
    - 监控服务器以每秒一次的频率向集群中的master节点、slave节点和其他的监控节点发送`ping`命令。
    - 如果一个节点距离最后一次有效回复`ping`命令的时间超过了配置文件中规定的时间，那么这个节点就会被标记为主观下线。
    - 如果一个master节点被标记为主观下线，那么监视master节点的所有监控服务器会以每秒一次的频率确认master节点确实进入了主观下线状态。如果由足够多的监控服务器将master节点标记为了主观下线状态，那么这个master节点会被标记为客观下线状态。
    - 使用哨兵模式时，客户端连接的是哨兵的IP和端口，由哨兵来提供Redis集群的相关服务，如果master节点挂掉，那么哨兵就会从slave节点中重新选择一个作为master节点

- 高可用集群：

    - 高可用集群解决了当数据量过大时，单台服务器的存储容量不够的问题。高可用集群可以对数据进行分片，将数据储存在多个服务器中。
    - 在cluster集群中，每个节点都可以作为一个主从模式的集群（一主一从或者一主多从），其中slave只作为备用，不提供服务。客户端可以直接连接master节点进行读写。
    - 多个节点之间通过网络进行数据共享
    - cluster集群通过哈希函数进行数据分区，因为Redis数据库是通过`key-value`的方式进行数据存储，所以在构建cluster集群时，所有的键会根据哈希函数分配到`0-16383`的虚拟槽中。
    - 如果master节点被标记为客观下线状态，那么会从他的slave节点中选择一个替换它，保证集群的高可用。

67.服务器内存不够怎么处理

- 首先查看服务器的内存容量，然后查看服务器的运行程序和进程，分析是由于程序运行过多和线程占用造成的内存不足，还是由于服务器本身的硬件资源不足导致的。
- 也有可能是因为服务器中毒引起的内存不足

68.CPU资源不足怎么处理

- 使用`top`命令查看一下哪个进程占用的CPU资源过高
- 找到进程中消耗资源最多的线程id，使用的命令是`top -H -p 进程id`
- 将线程id转换为16进制， 使用命令`printf "%0x\n" 线程id`或者`echo 'obase=16,线程id' | bc`
- 查看线程的状态信息作进一步分析，使用命令`jstack 进程id | grep -A 10 线程id的16进制`

69.遇到磁盘空间不足的报错，你应该如何处理

- 可能一：
  - 使用`df -h`命令查看磁盘空间的使用情况，确定哪个目录占用的磁盘空间过高
  - 确定目录后，使用`du -h`命令进行逐级定位，确认到占用空间最大的大文件
  - 查看文件内容，确认一下是否需要保留，如果需要保留就通过压缩导出，不需要保留可以直接删除，或者对磁盘进行扩容
  - 或者可以直接使用`find`命令查找目录下大于400MB的文件名称，然后进行删除`find 目录 -size +400M`
- 可能二：
  - 使用`df -h`命令并不能发现大文件，那么可能是文件被删除，但是进程仍然在调用这个文件
  - 此时可以通过`lsof | grep delete`，找到占用的进程，把这个进程kill掉然后重启服务器即可

70.如何将新磁盘扩展到已有分区

- 首先将磁盘格式化，CentOS6使用`fdisk`命令，CentOS7使用`gdisk`命令
- 使用`pvcreate`将新分区创建为物理卷
- 使用`pvdispalay`列出原有的物理卷组
- 使用`vgextend`把卷加入到卷组中
- 使用`df -h`查出需要进行扩展的分区名称，使用`lvresize -L 要扩展的容量 分区名`进行扩容
- 使用`xfs_growfs 挂载点`让文件系统进行识别

71.Linux如何开机自启

```
echo '启动命令' > /etc/rc.local
chmod +x /etc/rc.local

#服务有自己的启动脚本文件
mv 脚本启动配置文件 /usr/local/bin
mv 脚本启动配置文件 /usr/local/sbin
systemctl enable 服务名称
```

72.说一下文件描述符对系统有什么影响

Linux服务器的内核文件描述符的个数存在限制（对于不同的Linux服务器类型，数值也不一样），默认限制在1024左右。这个数值对于不太繁忙的文件是足够的，但是对于web服务器和加压测试的服务器来说，这个数值是不够用的。因为web服务器和加压测试的服务器会打开大量的文件，导致文件描述符不够用，进而导致无法访问新的文件。

73.Linux怎么挂载新磁盘

- 对新磁盘进行格式化，写入文件系统
- 使用 `mount` 命令对磁盘进行挂载
- 将挂载写入 `/etc/fstab` 配置文件，执行mount -a实现永久挂载
- 重启系统

74.找到修改30天以上的文件并删除

```
find 目录 -ctime +30
#c：表示属性被修改过
#a：表示文件被访问过
#m：表示内容被修改过
#time：按天查找
#min：按分钟查找
```

75.服务器系统日志放在哪里

- `/var/log/message`：系统启动后的信息和错误日志
- `/var/log/security`：与系统安全相关的日志
- `/var/log/maillog`：与系统邮件相关的日志
- `/var/log/cron`：与定时计划任务相关的日志
- `/var/log/boot.log & /var/log/dmesg`：与系统启动相关的日志

76.如果你的一块磁盘中，只有几个小文件，使用df命令查看，发现磁盘是满的，什么原因

有僵尸进程存在，大文件虽然删除了，但是僵尸进程仍然在调用这个文件，所以会显示磁盘被占满

怎么解决

使用 `lsof | grep deleted` 找到删除的文件对应的进程号，使用 `kill -9` 关闭进程，重启系统

77.查看操作系统的版本信息

```
uname -a            #查看操作系统的版本信息
cat /proc/cpuinfo   #查看CPU版本信息
cat /proc/version   #查看内核版本信息
cat /etc/issue      #查看发行版本信息
```

78.解释一下什么是HAC的脑裂，能说一下解决方案吗

- 因为主机异常或宕机时，集群内的主机都认为是对方出现异常，而导致的抢占资源现象。

- 解决方案：

  - 添加冗余的心跳线，比如同时使用串行电缆和以太网电缆
  - 设置参考IP，当出现线路中断的情况时，让2个节点都去ping参考IP，不通就说明是本节点故障，放弃抢占，让正常的节点接管服务
  - 出现脑裂现象时，直接强制关闭一个节点，备节点没有收到心跳隐患，就会通过单独的线路关闭主节点电源

79.Linux进程有哪些状态

- `R(TASK_RUNNING)`：运行状态
- `S(TASK_INTERRUPTIBLE)`：可中断的睡眠状态
- `D(TASK_UNINTERRUPTIBLE)`：不可中断的睡眠状态
- `T(TASK_STOPPED)`：挂起状态，可以被唤醒
- `Z(TASK_ZAOMBLE)`：僵尸状态，程序已经中止，但是父程序没有调用 `wait` 函数获取子进程的相关信息，那么子进程的状态信息仍然保留在内存中，变成僵尸进程

80.Linux查看系统资源的命令

- CPU：vmstat，iostat，mpstat，top，dstat
- 内存：vmstat，free，dstat
- 磁盘io：vmstat，iostat，top，dstat

81.Linux查看服务运行端口的命令

netstat -anpt，lsof -i -P

82./etc/profile和.bash_profile有什么区别

- /etc/profile：存放的是系统环境变量
- .bash_profile：存放的是当前用户的环境变量

83.数据库的锁是如何制定的

数据库是一个多用户使用的共享资源，所以会存在多用户同时读取一个数据的情况，如果不加以控制，可能会破坏数据的一致性

锁机制就是为了加强对数据的并发控制

84.MySQL为什么会出现锁表的情况，怎么解锁，如何避免这种情况的出现

- 锁表
  - 高并发情况下造成的io拥堵
  - 数据主从同步失败
  - 使用命令进行锁表：lock
- 解锁：unlock
- 增加数据库的缓存可以避免锁表的情况出现

85.innodb引擎是插入快还是读取快

写入快，因为写操作不会锁定全表，所以在并发较高时，使用innodb引擎效率更高

86.Docker和虚拟机的区别

- Docker启动属于秒级启动，虚拟机启动需要几分钟去进行启动
- Docker属于操作系统级别的虚拟化，通过Docker守护进程直接和内核进行交互，几乎没有性能损耗；虚拟机是硬件级别的虚拟化，需要通过Hypevisor层，性能损耗比较大
- Docker更轻量，占用内存小，在同样的硬件环境下，Docker运行的镜像数量要远多于虚拟机数量。
- Docker通过Namespaces和Cgroups实现对应用程序的进程之间的隔离，虚拟机从操作系统层面实现隔离，所以虚拟机的隔离性更强、安全性更好

87.Dockerfile文件

- FROM：定义基础镜像，必须指定而且要在其他的dockerfile指令之前
- MAINTAINER：定义镜像制作人的基本信息
- RUN：声明要安装或执行的文件，可以运行任何被基础镜像支持的命令
- CMD：容器启动时需要进行的操作，可以是执行系统脚本或者系统命令，只能在文件中出现一次，如果写入多个默认只有最后一个生效
- ENTERPOINT：容器启动时需要进行的操作，指定容器启动时需要执行的命令，只能在文件中出现一次，如果写入多个默认只有最后一个生效
- USER：定义容器中的用户
- EXPOSE：将容器的端口信息映射到物理机的端口上
- ENV：在容器中添加一个环境变量
- ADD：把一个路径下的文件复制到另一个路径下，如果这个文件是压缩包，那么复制时会强制解压
- COPY：把一个路径下的文件复制到另一个路径下
- VOLUME：数据卷挂载，实现容器的数据持久化
- WORKDIR：指定工作目录，容器启动后，会直接切换到工作目录下
- ONBUILD：指定的命令在子镜像中执行

88.Linux中常用的网络命令

- netstat
- ifconfig
- ping
- traceroute
- host

89.NetworkManager

Centos7以后的网卡图形化管理界面，优先级高于Network

90.简单介绍一下docker定义

基于linux comtainer的内核技术，go语言开发 http2.0协议 基于namespace的名称空间隔离，基于cgroup做资源限制，基于chroot做伪根，基于网桥技术的网络通信

91.KVM和docker有什么区别

- Docker 容器的启动可以在秒级实现，这相比传统的虚拟机方式要快得多。 其次，Docker对系统资源的利用率很高，一台主机上可以同时运行数千个 Docker 容器。

- 容器除了运行其中应用外，基本不消耗额外的系统资源，使得应用的性能很高，同时系统的开销尽量小。传统虚拟机方式运行 10 个不同的应用就要起 10 个虚拟机，而 Docker 只需要启动 10 个隔离的应用即可。
- 虚拟化技术依赖物理CPU和内存，是硬件级别的；而docker构建在操作系统上，利用操作系统的containerization技术，所以docker甚至可以在虚拟机上运行。
- 虚拟化系统一般都是指操作系统镜像，比较复杂，称为“系统”；而docker开源而且轻量，称为“容器”，单个容器适合部署少量应用，比如部署一个redis、一个memcached。
- 传统的虚拟化技术使用快照来保存状态；而docker在保存状态上不仅更为轻便和低成本，而且引入了类似源代码管理机制，将容器的快照历史版本一一记录，切换成本很低。
- 传统的虚拟化技术在构建系统的时候较为复杂，需要大量的人力；而docker可以通过Dockfile来构建整个容器，重启和构建速度很快。更重要的是Dockfile可以手动编写，这样应用程序开发人员可以通过发布Dockfile来指导系统环境和依赖，这样对于持续交付十分有利。
- 当然KVM对比于容器也有一个比较大的优势就是可以使用不同的操作系统或内核。所以，举例说，你可以使用微软Azure，同时运行Windows Server2012的实例和SUSE Linux企业级服务器的实例。至于Docker，所有容器都必须使用同样的操作系统和内核。

92.为什么说docker是轻量级

- 不依赖于系统硬件
- 创建和销毁简单
- 扩容和缩容简单
- 秒级启动
- 基于linux内核技术

93.Docker常见的网络模式

- brige：docker创建容器时，默认使用此网络模式，会创建一个docker0网桥，和宿主机进行通信
- none：创建容器时，关闭容器的网络功能
- host：桥接。使用宿主机的IP和端口

94.Docker中Namespaces和Cgroups的隔离类型

- User：用户和用户组
- UTS：主机名和域名
- PID：进程编号
- Mount：挂载（文件系统）
- Network：网络栈，网络设备，端口

- IPC：消息队列，共享内存，信号量

95.Docker如何缩容

使用kubernetes中deployment控制器对docker进行缩容

具体的操作方式是修改deployment的yaml文件，改变replication的实例数

96.kubernetes中组件和插件的功能

- 组件
  - apiserver：本质是一个web服务器，通过resetful接口编写，是整个kubernetes集群的入口
  - scheduler：调度器，将pod负载到node节点上
  - controllermanager：维稳集群状态
  - kubelet：apiserver和CRI之间的接口
  - etcd：键值对数据库，存储集群状态信息
  - kube-proxy：通过ipvs、iptables实现网络分发、负载均衡
- 插件
  - coredns：kubernetes集群内部的DNS解析服务
  - flannel：基于CNI的扁平化网络
  - Ingress nginx：提供7层负载

97.介绍一下kubernetes集群中的控制器

- 批处理任务
  - job
  - cronjob
- 守护进程
  - 有状态：statefulset。持久化存储，基于pv、pvc实现；持久的网络标识，基于无头服务实现；有序部署回收或启停
  - 无状态
    - RC
    - RS：集合式的标签选择器
    - Deployment：官方支持滚动更新和回滚，通过创建RS来引导Pod

- DaemonSet：node节点上只能有一个运行结果

98.pod的分类

- 自主式pod
- 控制器管理的pod

pod是kubernetes集群中最小的单位。kubernetes集群在创建pod时，首先，kubctl会向kubeapi发送指令，etcd会存储kubeapi调度kubectl的过程，启动一个基础容器pause，如果这个pod没有运行节点，那么scheduler会将这个pod调度到node节点上

99.python删除文件

- os.remove() 或os.unlink(）删除文件
- os.removedirs(path)，删除文件夹，但是文件夹必须为空。

100.ansible任务执行模式

- ad-hoc模式(点对点模式)
  使用单个模块，支持批量执行单条命令。ad-hoc 命令是一种可以快速输入的命令，而且不需要保存起来的命令。**就相当于bash中的一句话shell。**
- playbook模式(剧本模式)
  是Ansible主要管理方式，也是Ansible功能强大的关键所在。**playbook通过多个task集合完成一类功能**，如Web服务的安装部署、数据库服务器的批量备份等。可以简单地把playbook理解为通过组合多条ad-hoc操作的配置文件。

101.ansible 简介

- ansible是新出现的自动化运维工具，基于Python开发，集合了众多运维工具（puppet、chef、func、fabric）的优点，实现了批量系统配置、批量程序部署、批量运行命令等功能。
  ansible是基于 paramiko 开发的,并且基于模块化工作，本身没有批量部署的能力。真正具有批量部署的是ansible所运行的模块，ansible只是提供一种框架。ansible不需要在远程主机上安装client/agents，因为它们是基于ssh来和远
  程主机通讯的。ansible目前已经已经被红帽官方收购，是自动化运维工具中大家认可度最高的，并且上手容易，学习简单。是每位运维工程师必须掌握的技能之一。

102.ansible 执行命令过程

1. 加载自己的配置文件，默认 `/etc/ansible/ansible.cfg`；
2. 查找对应的主机配置文件，找到要执行的主机或者组；
3. 加载自己对应的模块文件，如 command；
4. 通过ansible将模块或命令生成对应的临时py文件(python脚本)， 并将该文件传输至远程服务器；
5. 对应执行用户的家目录的 `.ansible/tmp/XXX/XXX.PY` 文件；
6. 给文件 +x 执行权限；
7. 执行并返回结果；
8. 删除临时py文件， `sleep 0` 退出；

103.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。

104.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。

105.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
-

107.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。

-

108.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

109.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。

110.简述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

111.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。

112.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。

-

113.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。

113.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

-

114.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。

115.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。
-

116.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
-

117.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

-

118.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。

119.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。

120.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。

120.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。

120.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。

120.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。

120.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。

120.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。

-

120.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。
-

120.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。

120.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。

120.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。

120.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

  Kubernetes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。

120.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。

120.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。

120.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。

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。

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。

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。

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。

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。

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。

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。

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。

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。

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。

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。

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。

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。

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。

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。

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。

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）。

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。

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。

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）。

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。

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。

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。

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。

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”。

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。

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。

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。

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：

- 1、在该Node上安装Docker、kubelet和kube-proxy服务；
- 2、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；
- 3、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；
- 4、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。

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。

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。

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。

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。

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。

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。

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。

port

即，这里的port表示：service暴露在cluster ip上的端口，**:port** 是提供给集群内部客户访问service的入口

nodePort

首先，nodePort是kubernetes提供给集群外部客户访问service入口的一种方式（另一种方式是LoadBalancer），所以，:nodePort 是提供给集群外部客户访问service的入口

targetPort

targetPort很好理解，targetPort是pod上的端口，从port和nodePort上到来的数据最终经过kube-proxy流入到后端pod的targetPort上进入容器。

#

Tomcat日常参数优化**

1、内存--设置JVM参数**

说明：

-Xms：设置JVM初始内存大小(默认是物理内存的1/64)

-Xmx：设置JVM可以使用的最大内存(默认是物理内存的1/4，建议：物理内存80%)

-XX:PermSize ：为JVM启动时Perm的内存大小

-XX:MaxPermSize ：为最大可占用的Perm内存大小(默认为32M)

2.禁用AJP协议

ajp协议是tomcat为了动静资源处理分离时，通过该协议可以将css、js等静态资源请求转发到Apache的http服务器

处理，提高并发量。但是在优化tomcat时，没有用到Apache服务器，则需要将其禁用。

3.将BIO通讯模式修改为NIO通讯模式

其中，原来的protocol="HTTP/1.1"表示遵循http1.1协议，同时，也是一个最原始的未经优化的通信协议，修改之后

的 protocol="org.apache.coyote.http11.Http11NioProtocol" ，表示以 NIO模式启动。

4、并发--启用外部连接池

maxThreads：tomcat起动的最大线程数，即同时处理的任务个数，默认值为150

5、配置缓存

说明：

compression 打开压缩功能

compressionMinSize 启用压缩的输出内容大小，这里面默认为2KB

compressableMimeType 压缩类型

connectionTimeout 定义建立客户连接超时的时间. 如果为 -1, 表示不限制建立客户连接的时间6、优化连接器--最终模板

acceptCount：允许的最大连接数，应大于等于 maxProcessors ，默认值为 100

enableLookups：是否反查域名，取值为： true 或 false 。为了提高处理能力，应设置为 false

connectionTimeout：网络连接超时，单位：毫秒。设置为 0 表示永不超时，这样设置有隐患的。通常可设置

为20000毫秒。

disableUploadTimeOut：允许Servlet容器，正在执行使用一个较长的连接超时值，以使Servlet有较长的时间

来完成它的执行，默认值为false

maxPostSize：指定POST方式请求的最大量，没有指定默认为2097152

Tomcat配置文件

一、context.xml 文件

Context.xml 是 Tomcat 公用的环境配置，tomcat 服务器会定时去扫描这个文件。一旦发现文件被修改（时间戳改变了），就会自动重新加载这个文件，而不需要重启服务器。推荐在 $CATALINA_BASEconf/context.xml 中进行独立的配置。因为 server.xml 是不可动态重加载的资源，服务器一旦启动了以后，要修改这个文件，就得重启服务器才能重新加载，而 context.xml 文件则不然。

二、web.xml文件

Web应用程序描述文件，都是关于是Web应用程序的配置文件。所有Web应用的 web.xml 文件的父文件。

三、server.xml文件

server.xml是对tomcat的设置，可以设置端口号，添加虚拟机这些的，是对服务器的设置

tomcat-users.xml

Tomcat Manager是Tomcat自带的、用于对Tomcat自身以及部署在Tomcat上的应用进行管理的web应用。Tomcat是Java领域使用最广泛的服务器之一，因此Tomcat Manager也成为了使用非常普遍的功能应用。

在默认情况下，Tomcat Manager是处于禁用状态的。准确地说，Tomcat Manager需要以用户角色进行登录并授权才能使用相应的功能，不过Tomcat并没有配置任何默认的用户，因此需要我们进行相应的用户配置之后才能使用Tomcat Manager。

开发者检入代码到源代码仓库。

2. CI系统会为每一个项目创建了一个单独的工作区。当预设或请求一次新的构建时，它将把源代码仓库的源码存放到对应的工作区。
3. CI系统会在对应的工作区内执行构建过程。
4. （配置如果存在）构建完成后，CI系统会在一个新的构件中执行定义的一套测试。完成后触发通知(Email,RSS等等)给相关的当事人。
5. （配置如果存在）如果构建成功，这个构件会被打包并转移到一个部署目标(如应用服务器)或存储为软件仓库中的一个新版本。软件仓库可以是CI系统的一部分，也可以是一个外部的仓库，诸如一个文件服务器或者像Java.net、 SourceForge之类的网站。
6. CI系统通常会根据请求发起相应的操作，诸如即时构建、生成报告，或者检索一些构建好的构件。