IT运维面试问题总结

简述Heartbeat?
简述Keepalived及其工作原理?
简述Keepalived体系主要模块及其作用?
简述Keepalived如何通过健康检查来保证高可用?
简述LVS的概念及其作用?
简述LVS的工作模式及其工作过程?
简述LVS调度器常见算法（均衡策略）?
简述LVS、Nginx、HAProxy各自优缺点?
简述代理服务器的概念及其作用?
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?
简述什么是CAP理论?
简述什么是ACID理论?
简述什么是Kubernetes?
简述Kubernetes和Docker的关系?
简述Kubernetes中什么是Minikube、Kubectl、Kubelet?
简述Kubernetes常见的部署方式?
简述Kubernetes如何实现集群管理?
简述Kubernetes的优势、适应场景及其特点?
简述Kubernetes的缺点或当前的不足之处?
简述Kubernetes相关基础概念?
简述Kubernetes集群相关组件?
简述Kubernetes RC的机制?
简述Kubernetes Replica Set 和 Replication Controller 之间有什么区别?
简述kube-proxy作用?
简述kube-proxy iptables原理?
简述kube-proxy ipvs原理?
简述kube-proxy ipvs和iptables的异同?
简述Kubernetes中什么是静态Pod?
简述Kubernetes中Pod可能位于的状态?
简述Kubernetes创建一个Pod的主要流程?
简述Kubernetes中Pod的重启策略?
简述Kubernetes中Pod的健康检查方式?
简述Kubernetes Pod的LivenessProbe探针的常见方式?
简述Kubernetes Pod的常见调度方式?
简述Kubernetes初始化容器（init container）?
简述Kubernetes deployment升级过程?
简述Kubernetes deployment升级策略?
简述Kubernetes DaemonSet类型的资源特性?
简述Kubernetes自动扩容机制?
简述Kubernetes Service类型?
简述Kubernetes Service分发后端的策略?
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?

IT运维面试问题总结

Linux基础

简述Linux主流的发行版?

Redhat、CentOS、Fedora、SuSE、Debian、Ubuntu、FreeBSD等。

简述Linux启动过程?

- ⑴开机BIOS自检，加载硬盘。
- ⑵读取MBR，MBR引导。
- ⑶grub引导菜单(Boot Loader)。
- ⑷加载内核kernel。
- ⑸启动init进程，依据inittab文件设定运行级别。
- ⑹init进程，执行rc.sysinit文件。
- ⑺启动内核模块，执行不同级别的脚本程序。
- ⑻执行/etc/rc.d/rc.local。
- ⑼启动tty，进入系统登陆界面。

简述Linux删除文件的原理?

Linux系统是通过link的数量来控制文件删除的，只有当一个文件不存在任何link的时候，这个文件才会被删除。一般来说每个文件两个link计数器来控制：i_count和i_nlink。当一个文件被一个程序占用的时候i_count就加1。当文件的硬链接多一个的时候i_nlink也加1。删除一个文件，就是让这个文件，没有进程占用，同时i_link数量为0。

简述Linux运行级别?

- 0：关机模式
- 1：单用户模式<==破解root密码
- 2：无网络支持的多用户模式
- 3：有网络支持的多用户模式（文本模式，工作中最常用的模式）
- 4：保留，未使用
- 5：有网络支持的X-windows支持多用户模式（桌面）
- 6：重新引导系统，即重启

简述Linux常见目录及其作用?

- /（根目录）：Linux文件系统的起点；
- boot：存放Linux系统启动做必须的文件；
- var：存放经常变换的文件；
- home：普通用户的家目录
- root：Linux系统的root用户家目录；
- bin：存放系统基本的用户命令；
- sbin：存放系统基本的管理命令；
- use：存放Linux应用程序；
- etc：存放Linux系统和各种程序的配置文件。

简述Linux操作系统常见的文件系统有?

- EXT3
- EXT4
- XFS

简述Linux系统中的buffer和cache区别?

buffer和cache都是内存中的一块区域，当CPU需要写数据到磁盘时，由于磁盘速度比较慢，所以CPU先把数据存进buffer，然后CPU去执行其他任务，buffer中的数据会定期写入磁盘；当CPU需要从磁盘读入数据时，由于磁盘速度比较慢，可以把即将用到的数据提前存入cache，CPU直接从Cache中读取数据。

简述Linux中inode和block?

inode节点是一个64字节长的表，表中包含了文件的相关信息，如：字节数、属主UserID、属组GroupID、读写执行权限、时间戳等。在inode节点表中最重要的内容是：磁盘地址表。

文件名存放在目录当中，但Linux系统内部不使用文件名，而是使用inode号码识别文件。对于系统来说文件名只是inode号码便于识别的别称。即Linux文件系统通过把inode和文件名进行关联来查找文件。当需要读取该文件时，文件系统在当前目录表中查找该文件名对应的项，由此得到该文件相对应的inode节点号，通过该inode节点的磁盘地址表把分散存放的文件物理块连接成文件的逻辑结构。

文件是存储在硬盘上的，硬盘的最小存储单位叫做扇区sector，每个扇区存储512字节。操作系统读取硬盘的时候，不会一个个扇区地读取，这样效率太低，而是一次性连续读取多个扇区，即一次性读取一个块block。这种由多个扇区组成的块，是文件存取的最小单位。块的大小，最常见的是4KB，即连续八个sector组成一个block。

即512字节组成一个扇区（sector），多个扇区组成一个块（block），常见的块单位位4KB，即连续八个扇区组成一个block。

> 一个文件必须占用一个inode，但至少占用一个block。

简述Linux文件系统修复fsck过程?

成功修复文件系统的前提是要有两个以上的主文件系统（即两个系统），并保证在修复之前卸载将被修复的文件系统，然后使用命令fsck对受到破坏的文件系统进行修复。

fsck检查文件系统分为5步，每一步检查系统不同部分的连接特性并对上一步进行验证和修改。

检查从超级块开始、然后是分配的磁盘块、路径名、目录的连接性、链接数目以及空闲块链表、inode。

简述Linux中软链接和硬链接的区别?

- 软链接

软链接类似于Windows的快捷方式功能的文件，可以快速连接到目标文件或目录。即再创建一个独立的文件，而这个文件会让数据的读取指向它连接的那个文件的文件名。例如，文件A和文件B的inode号码虽然不一样，但是文件A的内容是文件B的路径。读取文件A时，系统会自动将访问者导向文件B。这时，文件A就称为文件B的软链接。

因此，文件A依赖于文件B而存在，如果删除了文件B，打开文件A就会报错。

- 硬链接

通过文件系统的inode链接来产生的新的文件名，而不是产生新的文件，称为硬链接。

一般情况下，每个inode号码对应一个文件名，但是Linux允许多个文件名指向同一个inode号码。意味着可以使用不同的文件名访问相同的内容。

创建硬链接，源文件与目标文件的inode号码相同，都指向同一个inode。inode信息中的链接数这时就会增加1。

    - 当一个文件拥有多个硬链接时，对文件内容修改，会影响到所有其他文件的内容；
    - 删除一个文件名，不影响另一个文件名的访问，删除一个文件名，只会使得inode中的链接数减1。

- 区别

软链接与硬链接最大的区别：软链接是文件A指向文件B的文件名，而不是文件B的inode号码，文件B的inode链接数不会因此发生变化。

> 不能对目录做硬链接，但是通过mkdir命令创建一个新目录，通常其硬链接数应该有2个，因为常见的目录本身为1个硬链接，而目录下面的隐藏目录.（点号）是该目录的又一个硬链接，也算是1个连接数。

简述TCP三次握手，四次断开，及其优点和缺点，同时相对于UDP的差别?

TCP与UDP概念：

- TCP：传输控制协议，即面向连接；
- UDP：用户数据报协议，无连接的，即发送数据之前不需要建立连接

TCP与UDP的优缺点上的区别：

- TCP的优点：

可靠，稳定。TCP的可靠体现在TCP在传递数据之前，会有三次握手来建立连接，而且在数据传递时，有确认、窗口、重传、拥塞控制机制，在数据传完后，还会断开连接用来节约系统资源。

- 三次握手：

1. 第一次握手，主机A向主机B发出一个含同步序列号的标志位的数据段给主机B ，向主机B请求建立连接。通过这个数据段，A向B声明通信请求，以及告知B可用某个序列号作为起始数据段进行响应；
2. 第二次握手，主机B收到主机A的请求后，用一带有确认应答(ACK)和同步序列号(SYN)标志位的数据段响应A。通过此数据段，B向A声明已收到A的请求，A可以传输数据了，同时告知A可用某个序列号作为起始数据段进行响应；
3. 第三次握手，主机A收到主机B的数据段后，再发送一个确认应答，确认已收到主机B 的数据段，之后开始正式实际传输数据。

> ACK：TCP报头的控制位之一，对数据进行确认。确认由目的端发出，来告知发送端这个序列号之前的数据段都收到了。比如，确认号为X，则表示前X-1个数据段都收到了。只有当ACK=1时，确认号才有效，当ACK=0时，确认号无效，此时会要求重传数据，保证数据的完整性。
>
> SYN：同步序列号，这个标志位只有在TCP建立连接时才会被置1，握手完成后SYN标志位被置0。

- 四次断开：

1. 当主机A完成数据传输后，将控制位FIN置1，提出停止TCP连接的请求；
2. 主机B收到FIN后对其作出响应，确认这一方向上的TCP连接将关闭，将ACK置1；
3. 主机B再提出反方向的关闭请求，将FIN置1；
4. 主机A对主机B的请求进行确认，将ACK置1，双方向的关闭结束。

- TCP的缺点：

慢、效率低、占用系统资源高、易被攻击：TCP在传递数据之前，要先建连接，需要消耗时间，而且在数据传递时，确认机制、重传机制、拥塞控制机制等都会消耗大量的时间，而且要在每台设备上维护所有的传输连接。同时，每个连接都会占用系统的CPU、内存等硬件资源。 而且，因为TCP有确认机制、三次握手机制，这些也导致TCP容易被人利用，实现DOS、DDOS、CC等攻击。

> DoS：拒绝服务（Denial of Servic），造成DoS的攻击行为被称为DoS攻击，其目的是使计算机或网络无法提供正常的服务。最常见的DoS攻击有计算机网络带宽攻击和连通性攻击。
>
> DDOS：分布式拒绝服务(DDoS:Distributed Denial of Service)，DDoS攻击指借助于客户/服务器技术，将多个计算机联合起来作为攻击平台，对一个或多个目标发动DDoS攻击，从而成倍地提高拒绝服务攻击的威力。

- UDP的优点：

快、比TCP稍安全、没有TCP的握手、确认、窗口、重传、拥塞控制等机制，UDP是一个无状态的传输协议，所以它在传递数据时非常快。没有TCP的这些机制，UDP被攻击者利用的漏洞就要少一些。但UDP也是无法避免攻击的，比如：UDP Flood攻击。

> UDP Flood攻击检测：短时间内向特定目标不断发送 UDP 报文，致使目标系统负担过重而不能处理合法的传输任务，就发生了 UDP Flood。启用 UDP Flood 攻击检测功能时，要求设置一个连接速率阈值，一旦发现保护主机响应的 UDP 连接速率超过该值，防火墙会输出发生 UDP Flood 攻击的告警日志，并且根据用户的配置可以阻止发往该主机的后续连接请求。

- UDP的缺点：

不可靠、不稳定。因为UDP没有那些可靠的机制，在数据传递时，如果网络质量不好，就会很容易丢包。

- TCP应用场景：

当对网络通讯质量有要求的时候，比如：整个数据要准确无误的传递给对方，要求可靠的应用，比如HTTP、HTTPS、FTP等传输文件的协议，POP、SMTP等邮件传输的协议。

- UDP应用场景：

当对网络通讯质量要求不高的时候，要求网络通讯速度能尽量的快。 比如QQ语音、QQ视频、TFTP。

简述TCP/IP及其主要协议?

TCP/IP协议是一个协议簇，其中包括很多协议的。

TCP/IP协议包括**应用层、传输层、网络层、网络访问层（网络接口层、网际层）**。

- 应用层：应用程序间沟通的层
    - 超文本传输协议(HTTP)：万维网的基本协议；
    - 文件传输(TFTP)：简单文件传输协议；
    - 远程登录(Telnet)：提供远程访问其它主机功能，它允许用户登录internet主机，并在这台主机上执行命令；
    - 网络管理(SNMP)：简单网络管理协议，该协议提供了监控网络设备的方法，以及配置管理、统计信息收集、性能管理及安全管理等；
    - 域名系统(DNS)：域名解析服务，该系统用于在internet中将域名及转换成IP地址；
- 传输层：提供了节点间的数据传送服务，给数据包加入传输数据并把它传输到下一层中，这一层负责传送数据，并且确定数据已被送达并接收。
    - 传输控制协议（TCP）
    - 用户数据报协议（UDP）
- 网络层：负责提供基本的数据封包传送功能，让每一个数据包都能够到达目的主机（但不检查是否被正确接收）。
    - Internet协议(IP)：根据网间报文IP地址，从一个网络通过路由器传到另一网络；
    - ICMP：Internet控制信息协议(ICMP)；
    - ARP：地址解析协议(ARP) ——"最不安全的协议"。
    - RARP：反向地址解析协议(RARP)：
- 网络访问层：又称作主机到网络层(host-to-network)，IP地址与物理地址硬件的映射及IP封装成帧，基于不同硬件类型的网络接口，网络访问层定义了与物理介质的连接。

简述OSI模型及其主要协议?

OSI模型是一个开放式系统互联参考模型，该模型人为的定义了七层结构。由下至上及其主要作用为：

1. 物理层：OSI的物理层规定了通信端点之间的机械特性、电气特性、功能特性以及过程特性，该层为上层协议提供了一个传输数据的物理媒体。该层数据的单位称为比特(bit)。其主要有：EIA/TIA、RS-232、EIA/TIA、RS-449、V.35、RJ-45、fddi令牌环网。
2. 数据链路层：定义了在单个链路上如何传输数据，其主要作用包括：作用包括物理地址寻址、数据的成帧、流量控制、数据的检错、重发等。该层数据的单位称为帧(frame)。其主要有：ARP、RARP、SDLC、HDLC、PPP、STP、帧中继。
3. 网络层：定义了端到端的包传输，定义了能够标识所有结点的逻辑地址，还定义了路由实现的方式和学习路由的方式。为了适应最大传输单元长度小于包长度的传输介质，网络层还定义了如何将一个包分解成更小的包的分段方法。主要负责寻找地址和路由选择，网络层还可以实现拥塞控制、网际互连等功能。该层数据的单位称为数据包(packet)。主要有：IP、IPX、RIP、OSPF。
4. 传输层：主要功能：
    - 为端到端连接提供传输服务；
    - 这种传输服务分为可靠和不可靠的，其中TCP是典型的可靠传输，而UDP则是不可靠传输；
    - 为端到端连接提供流量控制，差错控制，重新排序，服务质量等管理服务。

该层数据的单位称为数据段(segment)。主要有：TCP、UDP、SPX、DCCP、SCTP、RTP、RSVP、PPTP。

1. 会话层：他定义了如何开始、控制和结束一个会话，即负责建立和断开通信连接（数据流动的逻辑通路）。主要有：RPC、SQL、NetBIOS。
2. 表示层：定义数据格式及加密。主要负责数据格式的转换，确保一个系统的应用层信息可被另一个系统应用层读取。主要有：加密、ASII、TIFF、JPEG、HTML、PICT。
3. 应用层：与其他计算机进行通讯的一个应用，它是对应应用程序的通信服务的，为应用程序提供服务并规定应用程序中通信相关的细节。主要有：Telnet、HTTP、FTP、WWW、NFS、SMTP。

简述IP协议、IP地址?

IP协议(Internet Protocol)：又称互联网协议，是支持网间互连的数据报协议。它提供网间连接的完善功能，包括IP数据报规定互连网络范围内的IP地址格式。

为了实现连接到互联网上的结点之间的通信，必须为每个结点（入网的计算机）分配一个地址，并且应当保证这个地址是全网唯一的，这便是IP地址。

目前的IP地址（IPv4：IP第4版本）由32个二进制位表示，每8位二进制数为一个整数，中间由小数点间隔，整个IP地址空间有4组8位二进制数，由表示主机所在的网络的地址以及主机在该网络中的标识共同组成。 为了便于寻址和层次化的构造网络，IP地址被分为A、B、C、D、E五类，商业应用中只用到A、B、C三类。

- A类地址：网络标识由第一组8位二进制数表示，网络中的主机标识占3组8位二进制数，网络标识的第一位二进制数取值必须为"0"。A类地址允许有126个网段，每个网络大约允许有1670万台主机，通常分配给拥有大量主机的网络（如主干网）。 1.0.0.1 - 127.255.255.254
- B类地址：网络标识由前两组8位二进制数表示，网络中的主机标识占两组8位二进制数，网络标识的前两位二进制数取值必须为"10"。B类地址允许有16384个网段，每个网络允许有65533台主机，适用于结点比较多的网络（如区域网）。 128.1.0.1 - 191.255.255.254
- C类地址：网络标识由前3组8位二进制数表示，网络中主机标识占1组8位二进制数，网络标识的前3位二进制数取值必须为"110"。具有C类地址的网络允许有254台主机，适用于结点比较少的网络（如校园网）。 192.0.1.1 - 223.255.255.254

为了便于记忆，通常习惯采用4个十进制数来表示一个IP地址，十进制数之间采用句点"."予以分隔。这种IP地址的表示方法也被称为点分十进制法。

简述静态路由和动态路由及其特点?

静态路由：由系统管理员创建的路由，适用于网关数量有限的场合，且网络拓朴结构不经常变化的网络。其缺点是不能动态地适用网络状况的变化，当网络状况变化后需要网络管理员手动修改路由表。

动态路由：由路由选择协议动态构建的路由，路由协议之间通过交换各自所拥有的路由信息实时更新路由表的内容。动态路由可以自动学习网络的拓朴结构，并更新路由表。其缺点是路由广播更新信息将占据大量的网络带宽。

简述NAT的几种类型，及其原理?

常见的NAT主要有DNA和SNAT。

SNAT：指在数据包从网卡发送出去的时候，把数据包中的源地址部分替换为指定的IP。此时，接收方就认为数据包的来源是被替换的那个IP的主机。

DNAT：指数据包从网卡发送出去的时候，修改数据包中的目的IP。此时，若访问A，但因此DNAT的存在，所有访问A的数据包的目的IP全部修改为B，那么，实际上访问的是B。

简述包过滤防火墙和代理应用防火墙的区别?

包过滤防火墙：工作在网络层，根据包头中的源IP地址、目标IP地址、协议类型、端口号进行过滤；

代理应用防火墙：工作在应用层，使用代理服务器技术，将内网对外网的访问，变为防火墙对外网的访问，可以对包的内容进行分辨，从而过滤。

基础服务

简述Linux中常见的系统服务，其作用分别是?

常见的系统服务及其作用有：

- NTP/Chrony：用于时钟同步；
- DHCP：动态主机配置协议，用于自动分配主机地址，默认使用UDP 63端口；
- DNS：域名解析，运行在UDP协议之上，默认使用53端口；
- NFS：网络文件系统，依赖于RCP协议，其基本原则是"容许不同的客户端及服务端通过一组RPC分享相同的文件系统"，它是独立于操作系统，容许不同硬件及操作系统的系统共同进行文件的分享。
- Postfix：邮件服务；
- rsync：远程数据备份服务。
- VPN：虚拟专用网。

> 更多服务参考：**http://c.biancheng.net/view/1059.html**。

简述FTP主要的工作模式?

FTP工作模式是以服务端角度来区分，有主动模式和被动模式。

- 主动模式是指由FTP服务端主动向客户端发起连接，服务端端口号为20（用于传输）和21（用于控制），即20端口向客户端的一个大于1024的随机端口传输数据；
- 被动模式是指由FTP客户端向服务端发起连接，服务端采用随机端口等待客户端的随机端口来访问，从而传输数据。

简述FTP两种登录方式以及两种传输模式?

- FTP有两种登录方式：匿名登录和授权登录。

使用匿名登录时，用户名为：anonymous，密码为：任何合法email地址；

使用授权登录时，用户名为用户在远程FTP系统中的用户帐号，密码为用户在远程系统中的用户密码。

区别：使用匿名登录只能访问FTP目录下的资源，默认配置下只能下载；而授权登录访问的权限大于匿名登录，且上载、下载均可。

- FTP文件传输有两种文件传输模式：ASCII模式和binary模式。

ASCII模式用来传输文本文件；其他文件的传输使用binary模式。

简述DHCP的流程?

新节点通过DHCP获取地址信息的主要流程有如下四个过程：

1. 寻找DHCP Server
   客户机第一次登录网络的时，向网络上发出一个DHCPDISCOVER广播（包中包含客户机的MAC地址和计算机名等信息）。其源地址为0.0.0.0，目标地址为255.255.255.255。
2. 提供IP地址租用
   服务端监听到客户机发出的DHCP discover广播后，从剩余地址池中选择最前面的空置IP，连同其它TCP/IP设定，通过广播方式响应给客户端一个DHCP OFFER数据包（包中包含IP地址、子网掩

码、地址租期等信息）。源IP地址为DHCP Server的IP地址，目标地址为255.255.255.255。同时，DHCP Server为此客户保留它提供的IP地址，从而不会为其他DHCP客户分配此IP地址。

3. 接受IP租约
   客户机挑选最先响应的DHCP OFFER（一般是最先到达的那个），同时向网络广播DHCP REQUEST数据包（包中包含客户端的MAC地址、接受的租约中的IP地址、提供此租约的DHCP服务器地址等），声明将接受某一台服务器提供的IP地址。此时，由于还没有得到DHCP Server的最后确认，客户端仍然使用0.0.0.0为源IP地址，255.255.255.255为目标地址进行广播。
4. 租约确认
   服务端接收到客户端的DHCP REQUEST之后，会广播返回给客户机一个DHCP ACK消息包，表明已经接受客户机的选择，并将这一IP地址的合法租用以及其他的配置信息都放入该广播包发给客户机。
   客户机在接收到DHCP ACK广播后，会向网络发送三个针对此IP地址的ARP解析请求以执行冲突检测，查询网络上有没有其它机器使用该IP地址；如果发现该IP地址已经被使用，客户机会发出一个DHCP DECLINE数据包给DHCP Server，拒绝此IP地址租约，并重新发送DHCP discover信息。此时，在DHCP服务器管理控制台中，会显示此IP地址为BAD_ADDRESS。
   如果网络上没有其它主机使用此IP地址，则客户机的TCP/IP使用租约中提供的IP地址完成初始化，从而可以和其他网络中的主机进行通讯。

简述DNS查询可能需要哪些过程?

通常DNS查询有如下过程，任何一过程查询成功则返回查询结果，不再进行下一步查询：

1. 用户输入网址，优先调取本地hosts查询记录；
2. 使用本地dns缓存查询记录；
3. 使用网络设置的主dns查询记录；
4. 使用dns服务器中的缓存；
5. dns服务器转发查询，转发至上一级ISP DNS服务器，依次循环；
6. 若dns服务器未配置转发查询，则将查询需求发至13台根dns；
7. 返回查询IP结果给客户端。

简述DNS常见的服务器角色类型?

- 缓冲域名服务器
- 主域名服务器
- 从域名服务器

简述NFS文件系统及其作用?

网络文件系统是应用层的一种应用服务，它主要应用于Linux和Linux系统、Linux和Unix系统之间的文件或目录的共享。对于用户而言可以通过NFS方便的访问远地的文件系统，使之成为本地文件系统的一部分。采用NFS之后省去了登录的过程，方便了用户访问系统资源。

简述Samba作用及其使用场景?

Samba是在Linux上实现SMB协议的一个免费软件，由服务器及客户端程序构成。SMB（Server Messages Block，信息服务块）是一种在局域网上共享文件和打印机的一种通信协议，它为局域网内的不同计算机之间提供文件及打印机等资源的共享服务。SMB协议是客户机/服务器型协议，客户机通过该协议可以访问服务器上的共享文件系统、打印机及其他资源。主要用于windows与Linux之间的文件共享。

简述VPN概念以及常见的类型?

VPN是指在公共的网络上建立专用网络的技术，但是两个节点间并没有物理上的专用的端到端链路，而是通过广域网或者运营商提供的网络平台之上的逻辑网络，用户数据在逻辑链路中传输，同时VPN采用身份验证和加密技术，充分保证了安全性。常见的VPN有：IPSec VPN、PPTP VPN、L2TP VPN、SSL VPN。

简述YUM服务工作步骤?

客户端在通过yum安装软件时，会先访问repo仓库，下载仓库的元数据，根据元数据去查询所需要的rpm及其各种依赖关系。之后再在仓库进行相关下载，并自动解决rpm包的依赖关系。同时repo仓库应该是一个文件服务器，一般linux主机在下载过元数据的同时会将其保留在缓存中，以便后续使用。本质上是将底层的物理硬盘抽象的封装起来，然后以逻辑卷的方式呈现给上层应用。

磁盘管理

简述LVM概念及其特点?

LVM是对磁盘分区进行管理的一种机制，建立在硬盘和分区之上的一个逻辑层，用来提高磁盘管理的灵活性。通过LVM可将若干个磁盘分区连接为一个整块的卷组(Volume Group)，形成一个存储池。可以在卷组上随意创建逻辑卷(Logical Volumes)，并进一步在逻辑卷上创建文件系统，与直接使用物理存储在管理上相比，提供了更好灵活性。

- 设计概念
    - 物理存储介质（The physical media）：LVM存储介质可以是磁盘分区、整个磁盘、RAID阵列或SAN磁盘，设备必须初始化为LVM物理卷，才能与LVM结合使用；
    - 物理卷PV（physical volume）：物理卷就是LVM的基本存储逻辑块，但和基本的物理存储介质（如分区、磁盘等）比较，却包含有与LVM相关的管理参数，创建物理卷它可以用硬盘分区，也可以用硬盘本身；
    - 卷组VG（Volume Group）：一个LVM卷组由一个或多个物理卷组成；
    - 逻辑卷LV（logical volume）：LV建立在VG之上，可以在LV之上建立文件系统；
    - PE（physical extents）：PV物理卷中可以分配的最小存储单元，PE的大小是可以指定的，默认为4MB；
    - LE（logical extent）：LV逻辑卷中可以分配的最小存储单元，在同一个卷组中，LE的大小和PE是相同的，并且一一对应。
- 特点
    - 优点
        - 可以在系统运行的状态下动态的扩展文件系统的大小。
        - 文件系统可以跨多个磁盘，因此文件系统大小不会受物理磁盘的限制。
        - 可以增加新的磁盘到LVM的存储池中。
        - 可以以镜像的方式冗余重要的数据到多个物理磁盘。
        - 可以方便的导出整个卷组到另外一台机器。
    - 缺点
        - 在从卷组中移除一个磁盘的时候必须使用reducevg命令，有一定的限制：这个命令要求root权限，并且不允许在快照卷组中使用。
        - 当卷组中的一个磁盘损坏时，整个卷组都会受到影响。
        - 因为加入了额外的操作，存储性能受到影响。

简述RAID0、RAID1、RAID5原理及特点、使用场景?

RAID通常可以把硬盘整合成一个大磁盘，然后在大磁盘上再分区，提高数据量利用率、冗余性，根据其特点不同，常见的有RAID0、RADI1、RAID5。

RAID 0：指由多个盘组合成逻辑上的一个盘。

优点：读写快，容量利用率最高。

缺点：没有冗余，任何一块磁盘失效将影响到所有数据。

RAID 1：偶数盘，进行镜像。

优点：最高的冗余性。

缺点：浪费资源，成本高，数据利用率低。

RAID 5：奇数盘，至少3块磁盘。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。

优点：实现数据一定程度的冗余，同时也提升数据的读写性能。

缺点：构建此模式需要一定数量的磁盘。

冗余从好到坏：RAID 1 > RAID 10 > RAID 5 > RAID 0

性能从好到坏：RAID 0 > RAID 10 > RAID 5 > RAID 1

成本从低到高：RAID 0 > RAID 5 > RAID 1 > RAID 10

简述iSCSI存储及其优点?

iSCSI是Internet小型计算机系统接口，是一个基于TCP/IP的协议，用于通过IP网络仿真SCSI高性能本地存储总线，从而为远程存储设备提供数据传输和管理。iSCSI跨本地和广域网，通过分布式服务器和数组提供独立于位置的数据存储检索。

iSCSI优点

- 使用SAN摆脱了本地布线限制，促进了本地或远程数据中心的存储整合；
- iSCSI结构是逻辑性的，仅使用软件配置来进行新的存储分配，无需其他电缆和物理磁盘；
- iSCSI使用多个远程数据中心简化了数据复制、迁移和灾难恢复。

简述文件存储、块存储、对象存储?

文件存储：允许将数据组织为传统的文件系统。数据保存在一个文件中，该文件具有名称和一些相关的元数据，例如修改时间戳、所有者和访问权限。提供基于文件的存储使用目录和子目录的层次结构来组织文件的存储方式。

块存储：块存储提供了一个像硬盘驱动器一样工作的存储卷，组织成大小相同的块。通常，要么操作系统用文件系统格式化基于块的存储卷，要么应用程序(如数据库)直接访问它来存储数据。

对象存储：对象存储允许将任意数据和元数据存储为一个单元，并在平面存储池中标记为惟一标识符。使用API存储和检索数据，而不是将数据作为块或在文件系统层次结构中访问。

虚拟平台

简述什么是云计算及其基本特征?

云计算是一种采用按量付费的模式，基于虚拟化技术，将相应计算资源（如网络、存储等）池化后，提供便捷的、高可用的、高扩展性的、按需的服务（如计算、存储、应用程序和其他 IT 资源）。

云计算通常有如下基本特征：

- 自主服务：可按需的获取云端的相应资源（主要指公有云）；
- 网路访问：可随时随地使用任何联网终端设备接入云端从而使用相应资源。
- 资源池化：
- 快速弹性：可方便、快捷地按需获取和释放计算资源。
- 按量计费：

简述云计算常见部署模式?

- 私有云：云平台资源只给某个单位、或某部分用户内部使用。
- 公有云：云平台资源开放给社会公众服务。
- 社区云：云平台资源给几个固定的单位内使用。
- 混合云：两个或两个以上不同类型的云平台。

简述云计算三种服务模式?

- IaaS：基础设施即服务，云服务商将IT系统的基础设施（如计算资源、存储资源、网络资源）池化后作为服务进行售卖；
- PaaS：平台即服务，云服务商将IT系统的平台软件层（数据库、OS、中间件、运行库）作为服务进行售卖；
- SaaS：软件即服务，云服务商将IT系统的应用软件层作为服务进行售卖。

简述云计算和虚拟化的区别?

云计算：IT能力服务化，按需使用，按量计费，多租户隔离，是一个系统的轻量级管理控制面。

虚拟化：环境隔离，资源复用，降低隔离损耗，提升运行性能，提供高级虚拟化特性。

虚拟化是实现云计算的技术支撑之一，但并非云计算的核心关注点。

简述私有云相对公有云有哪些优势?

- 数据安全性更高；
- 可节省上云迁移过程中的大量成本；
- 业务快速部署，缩短业务周期；
- 降低企业成本，自主可控。

简述什么是KVM?

KVM指基于内核的虚拟机（Kernel-based Virtual Machine），它是一个Linux的一个内核模块，该内核模块使得Linux变成了一个 Hypervisor，从而实现虚拟化：

- 它由 Quramnet 开发，该公司于 2008年被 Red Hat 收购。
- 它支持 x86 (32 and 64 位)、s390、Powerpc 等 CPU。
- 它从 Linux 2.6.20 起就作为一模块被包含在 Linux 内核中。
- 它需要支持虚拟化扩展的 CPU。
- 它是完全开源的。

系统管理

简述Rsync及其特点?

Rsync是Linux系统中的数据镜像备份工具，通过rsync可以将本地系统数据通过网络备份到任何远程主机上。rysnc不仅仅能对不同位置的文件和目录进行同步，还可以差异计算，压缩传输文件来最小化数据传输，和cp命令相比，rysnc的优势在于高效的差异算法。并且，rysnc还支持网络数据传输，在复制文件的同时，会把源端与目的端的文件进行比较，只有当文件不一样的时候在进行复制。具有以下特性：

- 可以镜像保存整个目录树和文件系统。
- 可以同步增量数据，文件传输效率高，同步时间短。
- 可以保留原有文件的权限、时间等属性。
- 加密传输数据，保证了数据的安全性。

简述iptables规则工作过程?

iptables防火墙是一层层过滤的，实际是按照配置规则的顺序从上到下，从前到后进行过滤的。

如果匹配上了规则，即明确表明是阻止还是通过，此时数据包就不能向下匹配新规则了。

如果所有规则中没有明确表明是阻止还是通过这个数据包，也就是没有匹配上规则，向下进行匹配，直到匹配默认规则得到明确的阻止还是通过。

防火墙的默认规则是对应链的所有的规则执行完才会执行的，即最后执行的规则。

简述iptables有几个链、表及每个表的作用?

iptables有5个链：PREROUTING、INPUT、FORWARD、OUTPUT、POSTROUTING。

iptables有4个表：Filter、NAT、Mangle、RAW。

- Filter：主要和主机自身有关，真正负责主机防火墙功能（过滤流入流出主机的数据包）。filter表是iptables默认使用的表。filter定义了三个链（chains）：
    - INPUT：负责过滤所有目标地址是本机地址的数据包，通俗的讲，就是过滤进入主机的数据包
    - FORWARD：负责转发流经主机的数据包。起转发的作用，和nat关系很大，
    - OUTPUT：处理所有源地址是本机地址的数据包，通俗的讲，就是处理从主机发出去的数据包

对于filter表的控制是实现本机防火墙功能的重要手段，特别是对INPUT链的控制。

- nat：负责网络地址转换，即来源与目的ip地址的port的转换，一般用于局域网共享上网或特殊的端口转换服务相关，nat功能就相当于网络的acl控制。。nat定义了三个链（chains）：
    - OUTPUT：和主机发出去的数据包有关，改变主机发出数据包的目标地址。
    - PREROUTING：在数据包到达防火墙时进行路由判断之前执行的规则。作用时改变数据包的目的地址，目的端口等。
    - POSTROUTING：在数据包离开防火墙时进行路由判断之后执行的规则，作用改变数据包的源地址，源端口等。
- RAW：RAW表只使用在PREROUTING链和OUTPUT链上，因为优先级最高，从而可以对收到的数据包在连接跟踪前进行处理。一但用户使用了RAW表，在某个链上，RAW表处理完后，将跳过NAT表和 ip_conntrack处理，即不再做地址转换和数据包的链接跟踪处理了。RAW表可以应用在那些不需要做nat的情况下，以提高性能。如大量访问的web服务器，可以让80端口不再让iptables做数据包的链接跟踪处理，以提高用户的访问速度。
- Mangle：实现流量整形，主要用于修改数据包的ToS(Type of Service ，服务类型）、TTL(Time toLive，生存周期）以及为数据包设置 Mark 标记，以实现 QoS(Quality of Service，服务质量）调整以及策略路由等应用。

简述iptables各个表的优先级?

4个表的优先级由高到低的顺序为：raw-->mangle-->nat-->filter。

简述iptables处理经过的数据包的流程?

iptables利用表和链处理每个经过的数据包，具体流程（步骤）如下：

1. 数据包到达网络接口，比如 eth0。
2. 进入 raw 表的 PREROUTING 链，这个链的作用是在连接跟踪之前处理数据包。
3. 如果进行了连接跟踪，则进行处理。
4. 进入 mangle 表的 PREROUTING 链，在此可以修改数据包，比如 TOS 等。
5. 进入 nat 表的 PREROUTING 链，可以在此做DNAT，但不做过滤。
6. 决定路由，看是交给本地主机还是转发给其它主机，即决定是否继续往内还是往外。

到了这里需要分两种不同的情况进行讨论了。

- 若数据包决定要转发给其它主机，这时候它会依次经过：

1. 进入 mangle 表的 FORWARD 链，这里是在第一次路由（即步骤6）决定之后，在进行最后的路由决定之前，仍然可以对数据包进行某些修改。
2. 进入 filter 表的 FORWARD 链，这里可以对所有转发的数据包进行过滤。
3. 进入 mangle 表的 POSTROUTING 链，这里将完成了所有的路由决定，但数据包仍然在本地主机，还可以进行某些修改。
4. 进入 nat 表的 POSTROUTING 链，这里一般都是用来做 SNAT ，不在这里进行过滤。
5. 进入出去的网络接口，然后进行发送。

- 另一种情况是，数据包就是发给本地主机的，那么它会依次穿过：

1. 进入 mangle 表的 INPUT 链，这里是在第一次路由（即步骤6）决定之后，在进行最后的路由决定之前，仍然可以对数据包进行某些修改。
2. 进入 filter 表的 INPUT 链，这里可以对流入的所有数据包进行过滤，无论它来自哪个网络接口。
3. 交给本地主机的应用程序进行处理。
4. 处理完毕后进行路由决定，看该往那里发出。
5. 进入 raw 表的 OUTPUT 链，这里是在连接跟踪处理本地的数据包之前。
6. 连接跟踪对本地的数据包进行处理。
7. 进入 mangle 表的 OUTPUT 链，这里可以修改数据包，但不做过滤。
8. 进入 nat 表的 OUTPUT 链，可以对防火墙自己发出的数据做 NAT 。
9. 再次进行路由决定。
10. 进入 filter 表的 OUTPUT 链，可以对本地出去的数据包进行过滤。
11. 进入 mangle 表的 POSTROUTING 链，同上一种情况的第9步。
12. 进入 nat 表的 POSTROUTING 链，同上一种情况的第10步。
13. 进入出去的网络接口，然后进行发送。

运维工具

简述Ansible及其优势?

Ansible是一款极其简单的开源的自动化运维工具，基于Python开发，集合了众多运维工具(puppet, cfengine, 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。

同时Ansible是基于模块工作，其实现批量部署的是ansible所运行的模块。

Ansible其他重要的优势：

- 跨平台支持：Ansible在物理、虚拟、云和容器环境中为Linux、Windows、UNIX和网络设备提供无代理支持。
- 人类可读的自动化：Ansible playbook以YAML文本文件的形式编写，易于阅读，有助于确保每个人都理解他们将要做的事情。
- 对应用程序的完美描述：Ansible playbook可以进行任何更改，并且可以描述和记录应用程序环境的每个细节。
- 易于管理的版本控制：Ansible剧本和项目是纯文本。它们可以像源代码一样处理，并放在现有的版本控制系统中。
- 支持动态库存：Ansible管理的机器列表可以从外部资源动态更新，以便随时捕获所有受管服务器的正确的当前列表，无论基础设施或位置如何。
- 易于与其他系统集成的编排：HP SA、Puppet、Jenkins、Red Hat Satellite，以及存在于环境中的其他系统，都可以被利用并集成到Ansible工作流中。

简述Ansible工作机制及其特性?

Ansible是一款自动化运维工具，基于Python开发，具有批量系统配置, 批量程序部署, 批量运行命令等功能。

其工作机制如下：

1. 用户使用Ansible或Playbook，在服务器中断输入Ansible的Ad-Hoc命令集或Playbook；
2. Ansible遵循预先编排的规则将Playbooks逐条拆解为Play；
3. Play组织成Ansible可识别的任务（Task）；
4. Task会调用任务所涉及的所有模块（Module）和插件（Plugin）；
5. 读取Inventroy中定义的主机列表；
6. 通过SSH认证（默认）将任务集以临时文件或命令的形式传输到远程客户端执行并返回执行结果。

其特性如下：

1. no agents：不需要在被管控主机上安装任何客户端，只需SSH、Python即可，建议Python版本为2.6.6以上；
2. no server：无服务器端, 使用时直接运行命令即可；
3. modules in any languages：基于模块工作, 丰富的内置模块，可使用任意语言开发模块；
4. yaml, not code：使用yaml语言定制剧本playbook，易于管理，API简单明了；
5. ssh by default：基于SSH工作，整个过程简单、方便、安全，建议使用公钥方式认证；
6. strong multi-tier solution：可实现多级指挥。

简述Ansible中如何保存敏感数据?

在ansible内容中保留秘密数据并仍然公开共享，那么可以在playbooks中使用Vault。Ansible Vault，它包含在Ansible中，可以加密和解密Ansible使用的任何结构化数据文件。

简述Ansible适合的场景?

Ansible将编排与配置管理、供应和应用程序部署结合并统一在一个易于使用的平台上。Ansible的一些主要场景包括:

- 配置管理：集中配置文件管理和部署是Ansible的一个常见场景。

- 应用程序部署：当使用Ansible定义应用程序，并使用Ansible Tower管理部署时，团队可以有效地管理从开发到生产的整个应用程序生命周期。
- 部署：当在系统上部署或安装应用程序时，Ansible和Ansible Tower可以帮助简化供应系统的流程，无论是PXE启动的裸金属服务器或虚拟机，还是从模板创建虚拟机或云实例。
- 持续交付：创建CI/CD管道需要许多团队的协调和参与。如果没有一个简单的自动化平台，团队协作很难完成。而Ansible playbook在应用程序的整个生命周期中可以保持适当的部署(和管理)
- 安全性和审计：当安全策略在Ansible中定义时，可以将站点范围的安全策略的扫描和修复集成到其他自动化流程中。安全性是部署的所有内容中不可或缺的一部分。
- 编排：配置本身不能定义环境，需要定义多个配置如何交互，并确保可以将不同的部分作为一个整体来管理。

简述Ansible Inventory?

Ansible中受管主机列在主机清单（inventory）文本文件中，清单还将这些系统组织成group，以便更容易地进行批量管理。一个Inventory定义了Ansible将管理的主机集合。这些主机还可以分配至组，可以对组进行批量管理。组可以包含子组，主机可以是多个组的成员。Inventory根据类型可分为静态清单和动态清单：

- 静态主机Inventory可以由文本文件定义。
- 动态主机Inventory可以由脚本或其他程序根据需要使用外部信息提供者生成。

简述Ansible配置文件优先级?

Ansible 只使用最高优先级配置文件中的设置，其它配置文件中的设置将被忽略。即使存在其他优先级较低的文件，它们的设置也将被忽略，并且不会与所选配置文件中的设置相结合。

$ANSIBLE_CONFIG环境变量指定的任何文件都将覆盖所有其他配置文件。如果没有设置该变量，接下来将检查运行ansible命令的目录以查找ansible.cfg文件。如果该文件不存在，则检查用户的主目录以查找.ansible.cfg文件。如上配置文件都不存在时，才使用全局/etc/ansible/ansible.cfg文件。

简述Ansible ad-hoc命令?

Ad-Hoc命令是一种快速执行单个Ansible任务的方法，适合于不需要永久保存该任务，临时执行的场景。Ad-Hoc是简单的控制台操作，无需编写剧本就可以运行。它们对于快速测试和更改非常有用。

简述Ansible ad-hoc和playbook的区别?

- Ad-Hoc 命令可以作为一次性命令对一组目标主机运行单个、简单的任务。
- Ad-Hoc 不适合复杂配置管理或编配场景，Ad-Hoc 一次只能调用一个模块和一组参数。当需要多个操作时，必须使用多个 Ad-Hoc 来执行。
- playbook可以实现以一种简易重复的方式对一组目标主机运行多个复杂的任务。
- Playbook 是描述要在受管主机上实施的必要配置或程序性步骤的文件。
- Playbook 为配置管理和部署提供了强大而灵活的解决方案。
- Playbook 可以将冗长而复杂的管理任务转变为可轻松重复的历程，并且可预测成果然而。
- playbook 是一个文本文件，其中包含一个或多个按顺序运行的play的列表。
- playbook中，可以将playbook中的tasks保存为人类可读且可立即运行的形式。
- play 是一组有序的任务，应该对从目录中选择的主机运行。

简述Ansible变量?

Ansible 利用变量存储整个 Ansible 项目文件中可重复使用的值，从而可以简化项目的创建和维护，并减少错误的发生率。在定义Ansible变量时，通常有如下三种范围的变量：

- global范围：从命令行或Ansible配置中设置的变量；
- play范围：在 play 和相关结构中设置的变量；
- host范围：inventory、facts 或 register 的变量，在主机组和个别主机上设置的变量。

简述Ansible如何实现任务的循环?

简单循环：

- Ansible支持使用loop在一组item上迭代任务；
- loop可以使用列表中的每个项、列表中每个文件的内容、生成的数字序列或使用更复杂的结构来重复任务。
- 使用loop使管理员不必编写使用相同模块的多个任务。

复杂（嵌套）循环：

- with_nested键用于嵌套循环，循环在循环中运行。它需要一个包含两个或多个列表的列表。例如，将一个列表划分为两个列表，任务将迭代第一个列表中的每一项与第二个列表中的每一项。

简述Ansible hanlder?

Ansible模块被设计成幂等的，即在一个适当编写的剧本中，剧本及其任务可以在不更改受管主机的情况下多次运行，除非它们需要进行更改以使受管主机达到所需的状态。

然而，有时当一个任务对系统进行了更改后同时需要运行另一个任务。例如，对服务的配置文件的更改可能需要重新加载服务，以便更改后的配置生效。此时就需要使用hanlder程序。handler程序是响应由其他任务组成的通知的任务。每个handler程序都有一个全局惟一的名称，并在剧本中任务块的末尾触发。

如果没有任务通过名称调用handler程序，它将不会运行。

如果一个或多个任务都调用handler程序，它将在剧中的所有其他任务完成后仅运行一次。

因为handler程序是任务，所以可以在handler程序中使用与处理任何其他任务相同的模块。通常，handler程序用于重新启动主机和重新启动服务。

handler程序可以视为非活动任务，只有在使用notify语句显式调用时才会触发这些任务。

简述Ansible Block?

在 playbook 中， blocks 是囊括了任务的子句；

blocks 允许对任务进行逻辑分组，并可用于控制任务的执行方式，例如，管理员可以定义一组主要任务和一组附加任务，附加任务仅在第一组失败时执行。为此，可利用三个关键字在 playbook 中使用块：

- block：定义要运行的主要任务；
- rescue：定义将在 block 子句中定义的任务失败时运行的任务；
- always：定义始终都独立运行的任务，不论 block 和 rescue 子句中定义的任务是成功还是失败。

简述Ansible如何处理play错误的?

Ansible审查每个任务的返回代码，以确定任务是否成功或失败。默认情况下，当一个任务失败时，Ansible会立即中止该主机上的其他操作，并跳过所有后续任务。

实际生产中，若希望即使任务失败也能继续执行play，Ansible也包含了多种特性用于管理任务错误：

忽略任务失败：在任务中使用ignore_errors关键字忽略错误，即使任务失败，也继续在主机上执行playbook。

简述Ansible角色?

数据中心有各种不同类型的主机。如web服务器、数据库服务器，基于开发环境的服务器。随着时间的推移，具有处理所有这些情况的任务和人员的Ansible playbook将变得庞大而复杂。

- 角色允许将复杂的剧本组织成独立的、更小的剧本和文件。
- 角色提供了一种从外部文件加载任务、处理程序和变量的方法。

- 角色也可关联和引用静态的文件和模板。
- 角色可以编写成满足普通用途需求，并且能被重复利用。
- 定义角色的文件具有特定的名称，并以严格的目录结构进行组织。

简述Ansible Galaxy?

Ansible Galaxy是一个由各种Ansible管理员和用户编写的Ansible角色的公共库。它是一个包含数千个Ansible角色的归档文件，并且有一个可搜索的数据库，帮助Ansible用户识别可能帮助他们完成管理任务的角色。Ansible Galaxy包括指向新用户和角色开发人员的文档和视频的链接。

简述Ansible如何控制任务的并行执行?

通过在所有主机上并行运行任务，Ansible可以对剧本的执行进行更多的控制。默认情况下，Ansible默认最多并行5个，因此它将同时在5台不同的机器上运行一个特定的任务。Ansible可以通过配置forks来设置并行执行任务数量。

同时Ansible也可以通过serial来减少ork数量所指定的并行书，serial关键字主要用于控制滚动更新，避免一次性更新过多的节点。

简述Ansible故障后的排查思路?

- 日志判断：默认情况下，Ansible没有配置为将其输出，记录到任何日志文件中。可通过ansible.cfg配置文件default部分中的log_path参数或$ANSIBLE_LOG环境变量进行配置。然后通过日志进行定位。
- Debug模块：调试模块是Ansible可用的模块之一，它可以更好地了解控制节点上正在进行的操作。这个模块可以在playbook执行时为某个变量提供值。
- syntax-check：通过ansible-playbook 命令的 --syntax-check命选项检查剧本的YAML语法。
- diff：Ansible还提供了--diff选项。此选项报告对受管主机上的模板文件所做的更改。如果与--check选项一起使用，这些更改将显示出来，而不是实际执行。从而判断Ansible整个过程需要做何种更改。

开源应用

简述Ceph的优势及其特点?

Ceph是一个分布式的数据对象存储，Ceph相对其他存储系统具有如下优势：

- CRUSH算法： ceph摒弃了传统的集中式存储元数据寻址的方案，而使用CRUSH算法完成数据的寻址操作。能够实现各类负载的副本放置规则，例如跨机房、机架感知等。Crush算法有相当强大的扩展性，理论上支持数千个存储节点，从而增强了Ceph弹性扩展和高可用性。
- 高可用：通过CRUSH算法指定副本的物理存储位置以分隔故障域，支持数据强一致性，ceph可以忍受多种故障场景并自动尝试并行修复。
- 高扩展性：Ceph本身并没有主控节点，扩展起来比较容易，并且理论上，它的性能会随着磁盘数量的增加而线性增长。
- 特性丰富：Ceph支持三种调用接口：对象存储，块存储，文件系统挂载。三种方式可以一同使用。

Ceph主要特点如下：

- 统一存储；
- 无任何单点故障；
- 数据多份冗余；
- 存储容量可扩展；
- 自动容错及故障自愈。

简述Ceph存储体系架构?

Ceph体系架构主要由RADOS和RADOS GW和RBD以及CephFS构成。

- RADOS（Reliable, Autonomic Distributed Object Store）是Ceph的底层核心，RADOS本身也是分布式存储系统，CEPH所有的存储功能都是基于RADOS实现。RADOS由两个组件组成：OSD和Monitor。
    - OSD主要提供存储资源，每一个disk、SSD、RAID group或者一个分区都可以成为一个OSD，而每个OSD还将负责向该对象的复杂节点分发和恢复；
    - Monitor维护Ceph集群并监控Ceph集群的全局状态，提供一致性的决策。
- RADOS GW和RBD：RADOS GateWay、RBD其作用是在librados库的基础上提供抽象层次更高、更便于应用或客户端使用的上层接口。其中，RADOS GW是一个提供与Amazon S3和Swift兼容的RESTful API的gateway，以供相应的对象存储应用开发使用。 RBD则提供了一个标准的块设备接口，常用于在虚拟化的场景下为虚拟机创建volume。
- CEPHFS：CEPHFS则提供了POSIX接口，用户可直接通过客户端挂载使用。

简述Ceph Pool有几种类型?

Ceph存储池Pool是Ceph存储集群用于存储对象的逻辑分区。池类型确定池用于确保数据持久性的保护机制，Ceph有两种Pool类型：

replication类型：在集群中分布每个对象的多个副本。

erasure coding类型：将每个对象分割成块，并将它们与其他擦除编码块一起分发，以使用自动纠错机制保护对象。

简述Ceph Pool、PG、ODDs的关系?

Ceph存储池Pool是Ceph存储集群用于存储对象的逻辑分区。

Pool中存在一定的数量的PG，PG将对象存储在一组由CRUSH算法确定的osd中。

Ceph使用CRUSH算法将对象分配给池中的一个PG，根据池的配置和CRUSH算法，PG自动映射到一组OSDs。

一个PG里包含一堆对象，一个对象只能属于一个PG。

简述Ceph节点的角色?

所有Ceph存储集群的部署都始于部署一个个Ceph节点、网络和Ceph存储集群。Ceph存储集群至少需要一个Ceph Monitor和两个OSD守护进程。而运行Ceph文件系统客户端时，则必须要有元数据服务器（Metadata Server）。

- Ceph OSDs：Ceph OSD守护进程（ Ceph OSD ）的功能是存储数据，处理数据的复制、恢复、回填、再均衡，并通过检查其他OSD守护进程的心跳来向Ceph Monitors提供一些监控信息。当Ceph存储集群设定为有2个副本时，至少需要2个OSD守护进程，集群才能达到active+clean状态（Ceph默认有3个副本）。
- Monitors：Ceph Monitor维护着展示集群状态的各种图表，包括监视器图、OSD图、归置组（PG）图、和CRUSH 图。
- MDSs：Ceph元数据服务器（MDS）为Ceph文件系统存储元数据（也就是说，Ceph块设备和Ceph 对象存储不使用MDS）。元数据服务器使得POSIX文件系统的客户端，可以在不对Ceph存储集群造成负担的前提下，执行诸如ls、find等基本命令。

简述Ceph的适应场景?

Ceph的应用场景主要由它的架构确定，Ceph提供对象存储、块存储和文件存储。

- LIBRADOS应用

Librados提供了应用程序对RADOS的直接访问，目前Librados已经提供了对C、C++、Java、Python、Ruby和PHP的支持。

- RADOSGW应用

此类场景基于Librados之上，增加了HTTP协议，提供RESTful接口并且兼容S3、Swfit接口。RADOSGW将Ceph集群作为分布式对象存储，对外提供服务。

- RBD应用

此类场景也是基于Librados之上的，细分为下面两种应用场景。

第一种应用场景为虚拟机提供块设备。通过Librbd可以创建一个块设备（Container），然后通过QEMU/KVM附加到VM上。通过Container和VM的解耦，使得块设备可以被绑定到不同的VM上。

第二种应用场景为主机提供块设备。这种场景是传统意义上的理解的块存储。

以上两种方式都是将一个虚拟的块设备分片存储在RADOS中，都会利用数据条带化提高数据并行传输，都支持块设备的快照、COW（Copy-On-Write）克隆。最重要的是RBD还支持Live migration。

- CephFS（Ceph文件系统）

此类应用是基于RADOS实现的PB级分布式文件系统，其中引入MDS（Meta Date Server），它主要为兼容POSIX文件系统提供元数据，比如文件目录和文件元数据。

简述Docker的特性?

Docker主要有如下特性：

- 标准化
    - 保证一致的运行环境
    - 弹性伸缩，快速扩容
    - 方便迁移
    - 持续集成、持续交付与持续部署
- 高性能
    - 不需要进行硬件虚拟以及运行完整的操作系统
- 轻量级
    - 快速启动
    - 隔离性
    - 进程隔离

简述Docker容器的几种状态?

Docker容器可以有四种状态：

- 运行
- 已暂停
- 重新启动
- 已退出

简述Dockerfile、Docker镜像和Docker容器的区别?

Dockerfile 是软件的原材料，Docker 镜像是软件的交付品，而 Docker 容器则可以认为是软件的运行态。从应用软件的角度来看，Dockerfile、Docker 镜像与 Docker 容器分别代表软件的三个不同阶段，Dockerfile 面向开发，Docker 镜像成为交付标准，Docker 容器则涉及部署与运维。

简述Docker与KVM（虚拟机）的区别?

- 容器部署简单，虚拟机部署相对复杂。

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。

- 容器秒级启动，虚拟机通常分钟级启动。

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。

- 容器需要的资源（如磁盘、CPU、内存）相对更少。
- 容器比较轻便，虚拟机相对较重。

虚拟化系统一般都是指操作系统级概念，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。

简述Docker主要使用的技术?

容器主要使用如下技术：

- Cgroup：资源控制
- Namespace：访问隔离
- rootfs：文件系统隔离
- 容器引擎（用户态工具）：生命周期控制

简述Docker体系架构?

Docker体系相对简单，主要涉及如下5个组件：

- Docker客户端 – Docker

docker客户端则扮演着docker服务端的远程控制器，可以用来控制docker的服务端进程。

- Docker服务端 – Docker Daemon

docker服务端是一个服务进程，管理着所有的容器。**
**

- Docker镜像 – Image

docker镜像，一个能够运行在docker容器上的一组程序文件，是一个只读的模板，不包含任何动态数据。。

- Docker容器 – Docker Container

docker容器，就是运行程序的载体，容器是镜像运行时的实体。

- Docker镜像仓库 -- Registry

Docker仓库是集中存放镜像文件的场所，Docker仓库分为公开仓库（Public）和私有仓库（Private）两种形式。

简述Docker如何实现网络隔离?

Docker利用了网络的命名空间特性，实现了不同容器之间的网络隔离。命名空间可以支持网络协议栈的多个实例，独立的协议栈被隔离到不同的命名空间中。

因此处于不同命名空间中的网络栈是完全隔离的，彼此之间无法通信。每个独立的命名空间中可以有自己独立的路由表及独立的iptables设置来提供包转发、NAT及IP包过滤等功能。

简述Linux文件系统和Docker文件系统?

Linux文件系统：由bootfs和rootfs组成，bootfs主要包含bootloader和kernel，bootloader主要是引导加载kernel，当kernel被加载到内存之后bootfs就被卸载掉了。rootfs包含的就是典型linux系统中/dev,/proc,/bin,/etc等标准目录。

Docker文件系统：Docker容器是建立在Aufs分层文件系统基础上的，Aufs支持将不同的目录挂载到同一个虚拟文件系统下，并实现一种layer的概念。Aufs将挂载到同一虚拟文件系统下的多个目录分别设置成read-only，read-write以及whiteout-able权限。docker 镜像中每一层文件系统都是read-only。

简述Docker网络模式?

Docker使用Linux的Namespaces技术来进行资源隔离，其中Network Namespace实现隔离网络。

一个Network Namespace提供了一份独立隔离的网络环境，包括网卡、路由、Iptable规则等，Docker网络有如下四种模式：

- host模式：host模式下容器将不会获得独立的Network Namespace，该模式下与宿主机共用一个Network Namespace。容器将不会虚拟出自己的网卡，不会配置独有的IP等，而是使用宿主机的IP和端口。
- container模式：Container 网络模式是 Docker 中一种较为特别的网络的模式，处于container模式下的 Docker 容器会共享其他容器的网络环境，因此，两个或以上的容器之间不存在网络隔离，而配置container模式的容器又与宿主机以及除此之外其他的容器存在网络隔离。
- none模式：none模式下，Docker容器拥有自己的Network Namespace，但是，并不为Docker容器进行任何网络配置和构造任何网络环境。Docker 容器采用了none 网络模式，那么容器内部就只能使用loopback网络设备，不会再有其他的网络资源。Docker Container的none网络模式意味着不给该容器创建任何网络环境，容器只能使用127.0.0.1的本机网络。
- bridge模式：bridge模式是Docker默认的网络设置，此模式会为每一个容器分配Network Namespace、设置IP等，并将该宿主机上的Docker容器连接到一个虚拟网桥上。

简述Docker跨主机通信的网络实现方式?

docker跨主机通信按原理可通过以下三种方式实现：

- 直接路由方式：直接在不同宿主机之间添加静态路由；
- 桥接方式（如pipework）：通过静态指定容器IP为宿主机IP同一个网络的形式，即可实现。
- Overlay隧道方式：使用overlay网络实现，Overlay网络指在现有网络层之上叠加的虚拟化技术，实现应用在网络上的承载，并能与其他网络业务分离，并且以基于IP的网络技术为主，如flannel、ovs+gre。

简述flannel网络模型实现原理?

Flannel为每个host分配一个subnet，容器从subnet中分配IP，这些IP可以在host间路由，容器间无需使用nat和端口映射即可实现跨主机通信。每个subnet都是从一个更大的IP池中划分的，flannel会在每个主机上运flanneld的agent，负责从池子中分配subnet。

Flannel使用etcd存放网络配置、已分配的subnet、host的IP等信息，Flannel数据包在主机间转发是由backend实现的，目前已经支持UDP、VxLAN、host-gw、AWS VPC和GCE路由等多种backend。

简述什么是Apache服务器?

Apache服务器是一个非常流行、功能强大并且开源的Web服务器，基于HTTP超文本传输协议运行，这一协议提供了服务器和客户端Web浏览器通信的标准。它支持SSL、CGI文件、虚拟主机等许多功能特性。

简述Apache虚拟主机?

Apache虚拟主机相当于一个在同一台服务器中却相互独立的站点，从而实现一台主机对外提供多个web 服务，每个虚拟主机之间是独立的，互不影响的。Apache具有两种类型的虚拟主机：基于名称的虚拟主机和基于IP的虚拟主机。

简述Apache的Worker MPM和Prefork MPM之间的区别?

它们都是MPM，Worker和Prefork有它们各自在Apache上的运行机制，取决于哪种模式启动Apache。Worker MPM和Prefork MPM基本的区别在于它们产生子进程的处理过程。

1. Prefork MPM中，一个主httpd进行被启动，这个主进程会管理所有其它子进程为客户端请求提供服务。Worker MPM中一个httpd进程被激活，则会使用不同的线程来为客户端请求提供服务.
2. Prefork MPM使用多个子进程，每一个进程带有一个线程，Worker MPM使用多个子进程，每一个进程带有多个线程。
3. Prefork MPM中的连接处理，每一个进程一次处理一个连接而在Worker MPM中每一个线程一次处理一个连接。
4. 内存占用Prefork MPM占用庞大的内存，而Worker MPM占用更小的内存。

简述Nginx是什么及其主要特点?

Nginx是一款自由的、开源的、高性能的HTTP服务器和反向代理服务器。可以作为一个HTTP服务器进行网站的发布处理，同时也可以作为反向代理进行负载均衡的实现。其主要特点有：

- 占有内存少，并发能力强。
- Nginx使用基于事件驱动架构，使得其可以支持数以百万级别的TCP连接。
- 高度的模块化和自由软件许可证使得第三方模块非常丰富。
- Nginx是一个跨平台服务器，可以运行在Linux，Windows，FreeBSD，Solaris，AIX，Mac OS等操作系统上。

简述Nginx和Apache的差异?

- Nginx是一个基于事件的Web服务器，Apache是一个基于流程的服务器；
- Nginx所有请求都由一个线程处理，Apache单个线程处理单个请求；
- Nginx避免子进程的概念，Apache是基于子进程的；
- Nginx在内存消耗和连接方面更好，Apache在内存消耗和连接方面一般；
- Nginx的性能和可伸缩性不依赖于硬件，Apache依赖于CPU和内存等硬件；
- Nginx支持热部署，Apache不支持热部署；
- Nginx对于静态文件处理具有更高效率，Apache相对一般；
- Nginx在反向代理场景具有明显优势，Apache相对一般。

简述Nginx主要应用的场景?

基于Nginx的特性，Nginx的应用场景主要有：

- http服务器：Nginx是一个http服务可以独立提供http服务，可以做网页静态服务器。
- 虚拟主机：可以实现在一台服务器虚拟出多个网站。

- 正反代理：负载均衡或加速，当网站的访问量达到一定程度后，单台服务器不能满足用户的请求时，需要用多台服务器集群可以使用Nginx做反向代理，并且多台服务器可以平均分担负载。

简述Nginx HTTP连接和请求的关系?

HTTP是建立在TCP上，一次HTTP请求需要先建立TCP三次握手（称为TCP连接），在连接的基础上再进行HTTP请求。

HTTP请求建立在一次TCP连接基础上，对于HTTP会话，一次TCP连接可以建立多次HTTP请求。

简述Nginx支持哪些访问控制方式?

- 连接限制：Nginx自带的limit_conn_module模块（TCP连接频率限制模块）和limit_req_mudule模块（HTTP请求频率限制模块）支持对连接频率以及请求频率、来源进行限制，通常可可以用来防止DDOS攻击。
- IP限制：Nginx使用http_access_module模块可实现基于IP的访问控制，但通过代理可以绕过限制。
- 账号限制：Nginx使用http_auth_basic_module模块可实现用户密码的登录限制。
- 流量限制：Nginx使用http_core_moduleblock模块可实现客户端传送响应的速率限制。

简述Nginx Master进程和Worker节点?

master进程主要用来管理worker进程，包含：接收来自外界的信号，向各worker进程发送信号，监控worker进程的运行状态，当worker进程退出后(异常情况下)，会自动重新启动新的worker进程。

worker进程则是处理基本的网络事件。多个worker进程之间是对等的，他们同等竞争来自客户端的请求，各进程互相之间是独立的。一个请求，只可能在一个worker进程中处理，一个worker进程，不可能处理其它进程的请求。

简述Nginx如何处理HTTP请求?

首先，Nginx 在启动时，会解析配置文件，获取需要监听的端口与 IP 地址，然后在 Nginx 的 Master 进程里面先初始化好这个监控的Socket（创建 Socket，设置 addr、绑定ip和端口，然后listen 监听)。

然后，再 fork 出多个子进程出来。

之后，子进程会竞争 accept 新的连接。此时，客户端就可以向 nginx 发起连接了。当客户端与nginx完成TCP三次握手，与 nginx 建立好一个连接后。此时，某一个子进程会 accept 成功，得到这个建立好的连接的 Socket ，然后创建 nginx 对连接的封装。

接着，设置读写事件处理函数，并添加读写事件来与客户端进行数据的交换。

最后，Nginx 或客户端来主动关掉连接，完成整个HTTP请求的处理。

简述Nginx对于HTTP请求采用哪两种机制进行处理?

Nginx 是一个高性能的 Web 服务器，能够同时处理大量的并发请求。它结合多进程机制和异步非阻塞机制 。

- 多进程机制：服务器每当收到一个客户端请求时，就有服务器主进程 （master process）生成一个子进程（worker process）和客户端建立连接进行交互，直到连接断开，该子进程就结束了。
    - 使用进程的好处是各个进程之间相互独立，不需要加锁，减少了使用锁对性能造成影响。
    - 其次，采用独立的进程，可以让进程互相之间不会影响 ，如果一个进程发生异常退出时，其它进程正常工作，master进程则很快启动新的worker进程，确保服务不会中断，从而将风险降到最低。
    - 缺点是操作系统生成一个子进程需要进行 内存复制等操作，在资源和时间上会产生一定的开销。当有大量请求时，会导致系统性能下降 。

- 异步非阻塞机制：每个工作进程使用异步非阻塞方式，可以处理多个客户端请求 。
    - 当某个工作进程接收到客户端请求以后，调用 IO 进行处理，如果不能立即得到结果，就去处理其他请求（即为非阻塞 ）。而客户端在此期间也无需等待响应，可以进行其他任务（即为异步 ）。
    - 当IO返回时，就会通知此工作进程。该进程得到通知，暂时挂起当前处理的事务去响应客户端请求 。

简述Nginx支持哪些类型的虚拟主机?

对于Nginx而言，每一个虚拟主机相当于一个在同一台服务器中却相互独立的站点，从而实现一台主机对外提供多个 web 服务，每个虚拟主机之间是独立的，互不影响的。通过 Nginx 可以实现虚拟主机的配置，Nginx 支持三种类型的虚拟主机配置：

- 基于 IP 的虚拟主机（较少使用）
- 基于域名的虚拟主机
- 基于端口的虚拟主机

简述Nginx缓存及其作用?

缓存对于Web至关重要，尤其对于大型高负载Web站点。Nginx缓存可作为性能优化的一个重要手段，可以极大减轻后端服务器的负载。通常对于静态资源，即较少经常更新的资源，如图片，css或js等进行缓存，从而在每次刷新浏览器的时候，不用重新请求，而是从缓存里面读取，这样就可以减轻服务器的压力。

简述Nginx作为代理缓存后，客户端访问的过程?

使用Nginx作为代理缓存后，可加快客户端的访问，其过程大致如下：

1. 第一步：客户端第一次向Nginx请求数据A；
2. 第二步：当Nginx发现缓存中没有数据A时，会向服务端请求数据A；
3. 第三步：服务端接收到Nginx发来的请求，则返回数据A到Nginx，并且缓存在Nginx；
4. 第四步：Nginx返回数据A给客户端应用；
5. 第五步：客户端第二次向Nginx请求数据A；
6. 第六步：当Nginx发现缓存中存在数据A时，则不会请求服务端；
7. 第七步：Nginx把缓存中的数据A返回给客户端应用。

简述Nginx代理及其类型?

代理（forward）是一个位于客户端和原始服务器(origin server)之间的服务器，即代理服务器。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。

其通常有如下三种代理模式：

- 正向代理（forward proxy）：一个位于客户端和原始服务器(origin server)之间的服务器，为了从原始服务器取得内容，客户端向代理发送一个请求并指定目标(原始服务器)，然后代理向原始服务器转交请求并将获得的内容返回给客户端。
- 反向代理（reverse proxy）：指以代理服务器来接受 Internet上的连接请求，然后将请求，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，此时代理服务器对外就表现为一个反向代理服务器。
- 透明代理

简述Nginx盗链及如何防护?

盗链指的是在自己的界面展示非本服务器上的内容，通过技术手段获得其他服务器的资源。绕过他人资源展示页面，在自己页面向用户提供此内容，从而减轻自己服务器的负担，因为真实的空间和流量来自其他服务器。

因此，通常为了避免被盗链，通常Web服务器建议配置防盗链。Nginx防盗链其主要防盗链思路是能区别哪些请求是非正常用户请求，对于非正常用户的请求直接反馈403或重定向至其他页面。

简述Nginx负载均衡的意义?

负载均衡是将负载分摊到多个操作单元上执行，从而提高服务的可用性和响应速度，带给用户更好的体验。对于Web应用，通过负载均衡，可以将一台服务器的工作扩展到多台服务器中执行，提高整个网站的负载能力。其本质采用一个调度者，保证所有后端服务器都将性能充分发挥，从而保持服务器集群的整体性能最优，这就是负载均衡。

简述Nginx负载均衡的优势?

Nginx作为负载均衡器具有极大的优势，主要体现在：

- 高并发连接
- 内存消耗少
- 配置文件非常简单
- 成本低廉
- 支持Rewrite重写规则
- 内置的健康检查功能
- 节省带宽
- 稳定性高

简述Nginx负载均衡主要的均衡机制（策略）?

Nginx作为负载均衡器具有极大的优势，其负载均衡策略可以划分为两大类：内置策略和扩展策略，扩展策略为第三方提供。

- 内置策略
    - 轮询（默认）：Nginx根据请求次数，将每个请求均匀分配到每台服务器；
    - weight：加权轮询，加权轮询则是在第一种轮询的基础上对后台的每台服务赋予权重，服务器的权重比例越大，被分发到的概率也就越大。
    - least_conn：最少连接，将请求分配给连接数最少的服务器。Nginx会统计哪些服务器的连接数最少。
    - ip_hash：IP 哈希，绑定处理请求的服务器。第一次请求时，根据该客户端的IP算出一个HASH值，将请求分配到集群中的某一台服务器上。后面该客户端的所有请求，都将通过HASH算法，找到之前处理这台客户端请求的服务器，然后将请求交给它来处理。
- 扩展策略
    - fair：按后端服务器的响应时间来分配请求，响应时间短的优先分配。
    - url_hash：按访问url的hash结果来分配请求，使每个url定向到同一个后端服务器，后端服务器为缓存时比较有效。

简述Nginx负载均衡（反向代理）通过什么方式实现后端RS的健康检查?

nginx负载均衡（反向代理）包含内置的或第三方扩展来实现服务器健康检测的。如果后端某台服务器响应失败，nginx会标记该台服务器失效，在特定时间内，请求不分发到该台上。

- fail_timeout：该指令定义了多长时间服务器将被标记为失败。在fail_timeout后，服务器还是failed，nginx将检测该服务器是否存活，如果探测成功，将标记为活的。

- max_fails：该指令设置在fail_timeout期间内连续的失败尝试。默认情况下，max_fails为1。如果被设置为0，该服务器的健康检测将禁用。

简述Nginx动静分离?

为了提高网站的响应速度，减轻程序服务器（Tomcat，Jboss等）的负载，对于静态资源，如图片、js、css等文件，可以在反向代理服务器中进行缓存，这样浏览器在请求一个静态资源时，代理服务器就可以直接处理，而不用将请求转发给后端服务器。对于用户请求的动态文件，如servlet、jsp，则转发给Tomcat，Jboss服务器处理，这就是动静分离。即动态文件与静态文件的分离。

简述Nginx动静分离的原理?

动静分离可通过location对请求url进行匹配，将网站静态资源（HTML，JavaScript，CSS，img等文件）与后台应用分开部署，提高用户访问静态代码的速度，降低对后台应用访问。通常将静态资源放到nginx中，动态资源转发到tomcat服务器中。

简述Nginx同源策略?

同源策略是一个安全策略。同源，指的是协议，域名，端口相同。浏览器处于安全方面的考虑，只允许本域名下的接口交互，不同源的客户端脚本，在没有明确授权的情况下，不能读写对方的资源。

简述Nginx跨域及如何实现?

从一个域名的网页去请求另一个域名的资源，或任何协议、域名、端口有一处不同的请求，就被当作是跨域，即都被当成不同源。

通常基于安全考虑，Nginx启用了同源策略，即限制了从同一个源加载的文档或脚本如何与来自另一个源的资源进行交互。这是一个用于隔离潜在恶意文件的重要安全机制。

Nginx若要实现跨域访问，可通过JSONP和CORS进行实现。

简述Nginx重定向及其使用的场景?

重定向(Redirect)指通过各种方法将各种网络请求重新定个方向转到其它位置（如：网页重定向、域名的重定向、路由选择的变化也是对数据报文经由路径的一种重定向）。

URL重写是指通过配置conf文件，以让网站的URL中达到某种状态时则定向/跳转到某个规则，比如常见的伪静态、301重定向、浏览器定向等。当客户端浏览某个网址时，将其访问导向到另一个网址的技术。

其主要场景有如下两个：

- 将一串很长的网址，转成较短的网址，从而实现便于传播、易于记忆。
- 调整或更换Web服务器，网址（域名）又必须要变更（如访问目录、访问扩展名HTML变为PHP、访问域名），为了能使旧的访问依旧生效，从而实现自动重定向到新的网站。

简述Nginx地址重写、地址转发、反向代理?

地址重写：为了实现地址的标准化，如地址栏中中输入 www.baidu.com. 也可以输入 www.baidu.cn。最后都会被重写到 www.baidu.com 上。浏览器的地址栏也会显示www.baidu.com。即nginx把收到客户端请求的内容所对应的服务器地址发给客户端，让客户端自己去获取，nginx同时返回302正确信息。

地址转发：指在网络数据传输过程中数据分组到达路由器或桥接器后，该设备通过检查分组地址并将数据转发到最近的局域网的过程。

反向代理：当浏览器访问网站时，nginx反向代理服务器会代替客户端向后端服务器查找所需的内容，然后nginx反向代理服务器会把查找的内容返回给客户端。

简述Nginx地址重写和地址转发的差异?

地址重写和地址转发有以下不同点：

- 地址重写会改变浏览器中的地址，使之变成重写成浏览器最新的地址。而地址转发不会改变浏览器的地址的。
- 地址重写会产生两次请求，而地址转发只会有一次请求。
- 地址转发一般发生在同一站点项目内部，而地址重写且不受限制。
- 地址转发的速度比地址重定向快。

简述Nginx 301和302重定向及其区别?

301和302状态码都表示重定向，表示浏览器在拿到服务器返回的这个状态码后会自动跳转到一个新的URL地址，这个地址可以从响应的Location首部中获取（客户端输入的地址A瞬间变成了另一个地址B）。其主要差异为：

301：代表永久性转移(Permanently Moved)：旧地址A的资源已经被永久地移除了（这个资源不可访问了），搜索引擎在抓取新内容的同时也将旧的网址交换为重定向之后的网址；

302：代表暂时性转移(Temporarily Moved)：旧地址A的资源还在（仍然可以访问），这个重定向只是临时地从旧地址A跳转到地址B，搜索引擎会抓取新的内容而保存旧的网址。

简述Nginx高可用的常见方案?

Keepalived + Nginx 实现Nginx的高可用：通过Keepalived来实现同一个虚拟IP映射到多台Nginx代理服务器，从而实现Nginx的高可用性。

Heartbeat + Nginx 实现Nginx的高可用：通过Heartbeat的心跳检测和资源接管、集群中服务的监测、失效切换等功能，结合Nginx来实现高可用性。

简述SSL和HTTPS?

SSL（Secure Socket Layer）安全套接字层是一种数字证书，它使用ssl协议在浏览器和web server之间建立一条安全通道，数据信息在client与server之间的安全传输。

HTTPS（Hypertext Transfer Protocol Secure）是超文本传输协议和SSL/TLS的组合，用以提供加密通讯及对网络服务器身份的鉴定。

HTTPS也可以理解为HTTP over SSL，即HTTP连接建立在SSL安全连接之上。

简述NoSQL是什么?

NoSQL，指的是非关系型的数据库。NoSQL 有时也称作 Not Only SQL（意即"不仅仅是SQL"） 的缩写，其显著特点是不使用SQL作为查询语言，数据存储不需要特定的表格模式。

简述NoSQL（非关系型）数据库和SQL（关系型）数据库的区别?

NoSQL和SQL的主要区别有如下区别：

- 存储方式：
    - 关系型数据库是表格式的，因此存储在表的行和列中。他们之间很容易关联协作存储，提取数据很方便。
    - NoSQL数据库则与其相反，它是大块的组合在一起。通常存储在数据集中，就像文档、键值对或者图结构。
- 存储结构
    - 关系型数据库对应的是结构化数据，数据表都预先定义了结构（列的定义），结构描述了数据的形式和内容。预定义结构带来了可靠性和稳定性，但是修改这些数据比较困难。

    - NoSQL数据库基于动态结构，使用与非结构化数据。由于NoSQL数据库是动态结构，可以很容易适应数据类型和结构的变化。

- 存储规范
    - 关系型数据库的数据存储为了更高的规范性，把数据分割为最小的关系表以避免重复，获得精简的空间利用。
    - NoSQL数据存储在平面数据集中，数据经常可能会重复。单个数据库很少被分隔开，而是存储成了一个整体，这样整块数据更加便于读写 。

- 存储扩展
    - 关系型数据库数据存储在关系表中，操作的性能瓶颈可能涉及到多个表，需要通过提升计算机性能来克服，因此更多是采用纵向扩展
    - NoSQL数据库是横向扩展的，它的存储天然就是分布式的，可以通过给资源池添加更多的普通数据库服务器来分担负载。

- 查询方式
    - 关系型数据库通过结构化查询语言来操作数据库（即通常说的SQL）。SQL支持数据库CURD操作的功能非常强大，是业界的标准用法。
    - NoSQL查询以块为单元操作数据，使用的是非结构化查询语言（UnQl），它是没有标准的。
    - 关系型数据库表中主键的概念对应NoSQL中存储文档的ID。
    - 关系型数据库使用预定义优化方式（比如索引）来加快查询操作，而NoSQL更简单更精确的数据访问模式。

- 事务
    - 关系型数据库遵循ACID规则（原子性(Atomicity)、一致性(Consistency)、隔离性(Isolation)、持久性(Durability)）。
    - NoSQL数据库遵循BASE原则（基本可用（Basically Availble）、软/柔性事务（Soft-state）、最终一致性（Eventual Consistency））。
    - 由于关系型数据库的数据强一致性，所以对事务的支持很好。关系型数据库支持对事务原子性细粒度控制，并且易于回滚事务。
    - NoSQL数据库是在CAP（一致性、可用性、分区容忍度）中任选两项，因为基于节点的分布式系统中，不可能同时全部满足，所以对事务的支持不是很好。

简述NoSQL（非关系型）数据库和SQL（关系型）数据库的各自主要代表?

SQL：MariaDB、MySQL、SQLite、SQLServer、Oracle、PostgreSQL。

NoSQL代表：Redis、MongoDB、Memcache、HBASE。

简述MongoDB及其特点?

MongoDB是一个开源的、基于分布式的、面向文档存储的非关系型数据库。是非关系型数据库当中功能最丰富、最像关系数据库的。其主要特点如下：

查询丰富：MongoDB最大的特点是支持的查询语言非常强大，其语法有点类似于面向对象的查询语言，几乎可以实现类似关系数据库单表查询的绝大部分功能，而且还支持对数据建立索引。

面向文档：文档就是存储在MongoDB中的一条记录,是一个由键值对组成的数据结构。

模式自由：MongoDB每一个Document都包含了元数据信息，每个文档之间不强迫要求使用相同的格式，同时他们也支持各种索引。

高可用性：MongoDB支持在复制集(Replica Set)通过异步复制达到故障转移，自动恢复，集群中主服务器崩溃停止服务和丢失数据，备份服务器通过选举获得大多数投票成为主节点，以此来实现高可用。

水平拓展：MongoDB支持分片技术，它能够支持并行处理和水平扩展。

支持丰富：MongoDB另外还提供了丰富的BSON数据类型，还有MongoDB的官方不同语言的driver支持(C/C++、C#、Java、Node.js、Perl、PHP、Python、Ruby、Scala)。

简述MongoDB的优势有哪些?

- 面向文档的存储：以 JSON 格式的文档保存数据。
- 任何属性都可以建立索引。
- 复制以及高可扩展性。
- 自动分片。
- 丰富的查询功能。
- 快速的即时更新。

简述MongoDB适应的场景和不适用的场景?

MongoDB属于典型的非关系型数据库。

- 主要适应场景
    - 网站实时数据：MongoDB 非常适合实时的插入，更新与查询，并具备网站实时数据存储所需的复制及高度伸缩性。
    - 数据缓存：由于性能很高，MongoDB 也适合作为信息基础设施的缓存层。在系统重启之后，由 MongoDB 搭建的持久化缓存层可以避免下层的数据源过载。
    - 高伸缩性场景：MongoDB 非常适合由数十或数百台服务器组成的数据库。
    - 对象或 JSON 数据存储：MongoDB 的 BSON 数据格式非常适合文档化格式的存储及查询。
- 不适应场景
    - 高度事务性系统：例如银行或会计系统。传统的关系型数据库目前还是更适用于需要大量原子性复杂事务的应用程序。
    - 传统的商业智能应用：针对特定问题的 BI 数据库会对产生高度优化的查询方式。对于此类应用，数据仓库可能是更合适的选择。
    - 需要复杂 SQL 查询的场景。

简述MongoDB中的库、集合、文档?

- 库：MongoDB可以建立多个数据库，MongoDB默认数据库为"db"。MongoDB的单个实例可以容纳多个独立的数据库，每一个都有自己的集合和权限，不同的数据库也放置在不同的文件中。
- 集合：MongoDB集合就是 MongoDB 文档组，类似于 RDBMS （关系数据库中的表格）。集合存在于数据库中，集合没有固定的结构。
- 文档：MongoDB的Document是一组键值(key-value)对(即 BSON)，相当于关系型数据库的行。且不需要设置相同的字段，并且相同的字段不需要相同的数据类型。

简述MongoDB支持的常见数据类型?

MongoDB支持丰富的数据类型，常见的有：

- String：字符串。存储数据常用的数据类型。
- Integer：整型数值。用于存储数值。
- Boolean：布尔值。用于存储布尔值（真/假）。
- Array：用于将数组或列表或多个值存储为一个键。
- Date：日期时间。用 UNIX 时间格式来存储当前日期或时间。
- Binary Data：二进制数据。用于存储二进制数据。
- Code：代码类型。用于在文档中存储 JavaScript 代码。
- Regular expression：正则表达式类型。用于存储正则表达式。

简述MongoDB索引及其作用?

索引通常能够极大的提高查询的效率，如果没有索引，MongoDB在读取数据时必须扫描集合中的每个文件并选取那些符合查询条件的记录。

这种扫描全集合的查询效率是非常低的，特别在处理大量的数据时，查询可能要花费几十秒甚至几分钟，这对网站的性能是非常致命的。

索引是特殊的数据结构，索引存储在一个易于遍历读取的数据集合中，索引是对数据库表中一列或多列的值进行排序的一种结构。

简述MongoDB常见的索引有哪些?

MongoDB常见的索引有：

- 单字段索引（Single Field Indexes）
- 符合索引（Compound Indexes）
- 多键索引（Multikey Indexes）
- 全文索引（Text Indexes）
- Hash索引（Hash Indexes）
- 通配符索引（Wildcard Indexes）

简述MongoDB复制（本）集原理?

mongodb的复制至少需要两个节点。其中一个是主节点，负责处理客户端请求，其余的都是从节点，负责复制主节点上的数据。

mongodb各个节点常见的搭配方式为：一主一从、一主多从。

主节点记录在其上的所有操作oplog，从节点定期轮询主节点获取这些操作，然后对自己的数据副本执行这些操作，从而保证从节点的数据与主节点一致。

简述MongoDB的复制过程?

Primary节点写入数据，Secondary通过读取Primary的oplog（即Primary的oplog.rs表）得到复制信息，开始复制数据并且将复制信息写入到自己的oplog。如果某个操作失败，则备份节点停止从当前数据源复制数据。如果某个备份节点由于某些原因挂掉了，当重新启动后，就会自动从oplog的最后一个操作开始同步。同步完成后，将信息写入自己的oplog，由于复制操作是先复制数据，复制完成后再写入oplog，有可能相同的操作会同步两份，MongoDB设定将oplog的同一个操作执行多次，与执行一次的效果是一样的。

当Primary节点完成数据操作后，Secondary的数据同步过程如下：

1：检查自己local库的oplog.rs集合找出最近的时间戳。

2：检查Primary节点local库oplog.rs集合，找出大于此时间戳的记录。

3：将找到的记录插入到自己的oplog.rs集合中，并执行这些操作。

简述MongoDB副本集及其特点?

MongoDB副本集是一组Mongod维护相同数据集的实例，副本集可以包含多个数据承载点和多个仲裁点。在承载数据的节点中，仅有一个节点被视为主节点，其他节点称为次节点。

主要特点：

- N 个节点的集群，任何节点可作为主节点，由选举产生；
- 最小构成是：primary，secondary，arbiter，一般部署是：primary，2 secondary。
- 所有写入操作都在主节点上，同时具有自动故障转移，自动恢复；
- 成员数应该为奇数，如果为偶数的情况下添加arbiter，arbiter不保存数据，只投票。

简述MongoDB有哪些特殊成员?

MongoDB中Secondary角色存在一些特殊的成员类型：

- Priority 0（优先级0型）：不能升为主，可以用于多数据中心场景；
- Hidden（隐藏型）：对客户端来说是不可见的，一般用作备份或统计报告用；
- Delayed（延迟型）：数据比副集晚，一般用作 rolling backup 或历史快照。
- Vote（投票型）：仅参与投票。

简述MongoDB分片集群?

MongoDB分片集群（Sharded Cluster）：主要利用分片技术，使数据分散存储到多个分片（Shard）上，来实现高可扩展性。

分片是将数据水平切分到不同的物理节点。当数据量越来越大时，单台机器有可能无法存储数据或读取写入吞吐量有所降低，利用分片技术可以添加更多的机器来应对数据量增加以及读写操作的要求。

简述MongoDB分片集群相对副本集的优势?

MongoDB分片集群主要可以解决副本集如下的不足：

- 副本集所有的写入操作都位于主节点；
- 延迟的敏感数据会在主节点查询；
- 单个副本集限制在12个节点；
- 当请求量巨大时会出现内存不足；
- 本地磁盘不足；
- 垂直扩展价格昂贵。

简述MongoDB分片集群的优势?

MongoDB分片集群主要有如下优势：

- 使用分片减少了每个分片需要处理的请求数：通过水平扩展，群集可以提高自己的存储容量。比如，当插入一条数据时，应用只需要访问存储这条数据的分片。
- 使用分片减少了每个分片存储的数据：分片的优势在于提供类似线性增长的架构，提高数据可用性，提高大型数据库查询服务器的性能。当MongoDB单点数据库服务器存储成为瓶颈、单点数据库服务器的性能成为瓶颈或需要部署大型应用以充分利用内存时，可以使用分片技术。

简述MongoDB分片集群的架构组件?

MongoDB架构组件主要有：

- Shard：用于存储实际的数据块，实际生产环境中一个shard server角色可由几台机器组成一个replica set承担，防止主机单点故障。
- Config Server：mongod实例，存储了整个 ClusterMetadata，其中包括 chunk信息。
- Query Routers：前端路由，客户端由此接入，且让整个集群看上去像单一数据库，前端应用可以透明使用。

简述MongoDB分片集群和副本集群的区别?

副本集不是为了提高读性能存在的，在进行oplog的时候，读操作是被阻塞的；

提高读取性能应该使用分片和索引，它的存在更多是作为数据冗余，备份。

简述MongoDB的几种分片策略及其相互之间的差异?

MongoDB的数据划分是基于集合级别为标准，通过shard key来划分集合数据。主要分片策略有如下三种：

- 范围划分：通过shard key值将数据集划分到不同的范围就称为基于范围划分。对于数值型的shard key：可以虚构一条从负无穷到正无穷的直线（理解为x轴），每个shard key 值都落在这条直线的某个点上，然后MongoDB把这条线划分为许多更小的没有重复的范围成为块（chunks），一个chunk就是某些最小值到最大值的范围。
- 散列划分：MongoDB计算每个字段的hash值，然后用这些hash值建立chunks。基于散列值的数据分布有助于更均匀的数据分布，尤其是在shard key单调变化的数据集中。
- 自定义标签划分：MongoDB支持通过自定义标签标记分片的方式直接平衡数据分布策略，可以创建标签并且将它们与shard key值的范围进行关联，然后分配这些标签到各个分片上，最终平衡器转移带有标签标记的数据到对应的分片上，确保集群总是按标签描述的那样进行数据分布。标签是控制平衡器行为及集群中块分布的主要方法。

差异：

- 基于范围划分对于范围查询比较高效。假设在shard key上进行范围查询，查询路由很容易能够知道哪些块与这个范围重叠，然后把相关查询按照这个路线发送到仅仅包含这些chunks的分片。
- 基于范围划分很容易导致数据不均匀分布，这样会削弱分片集群的功能。
- 基于散列划分是以牺牲高效范围查询为代价，它能够均匀的分布数据，散列值能够保证数据随机分布到各个分片上。

简述MongoDB分片集群采取什么方式确保数据分布的平衡?

新加入的数据及服务器都会导致集群数据分布不平衡，MongoDB采用两种方式确保数据分布的平衡：

- 拆分

拆分是一个后台进程，防止块变得太大。当一个块增长到指定块大小的时候，拆分进程就会块一分为二，整个拆分过程是高效的。不会涉及到数据的迁移等操作。

- 平衡

平衡器是一个后台进程，管理块的迁移。平衡器能够运行在集群任何的mongd实例上。当集群中数据分布不均匀时，平衡器就会将某个分片中比较多的块迁移到拥有块较少的分片中，直到数据分片平衡为止。

分片采用后台操作的方式管理着源分片和目标分片之间块的迁移。在迁移的过程中，源分片中的块会将所有文档发送到目标分片中，然后目标分片会获取并应用这些变化。最后，更新配置服务器上关于块位置元数据。

简述MongoDB备份及恢复方式?

mongodb备份恢复方式通常有以下三种：

- 文件快照方式：此方式相对简单，需要系统文件支持快照和mongod必须启用journal。可以在任何时刻创建快照。恢复时，确保没有运行mongod，执行快照恢复操作命令，然后启动mongod进程，mongod将重放journal日志。
- 复制数据文件方式：直接拷贝数据目录下的一切文件，但是在拷贝过程中必须阻止数据文件发生更改。因此需要对数据库加锁，以防止数据写入。恢复时，确保mongod没有运行，清空数据目录，将备份的数据拷贝到数据目录下，然后启动mongod。
- 使用mongodump和mongorestore方式：在Mongodb中我们使用mongodump命令来备份MongoDB数据。该命令可以导出所有数据到指定目录中。恢复时，使用mongorestore命令来恢复MongoDB数据。该命令可以从指定目录恢复相应数据。

简述MongoDB的聚合操作?

聚合操作能够处理数据记录并返回计算结果。聚合操作能将多个文档中的值组合起来，对成组数据执行各种操作，返回单一的结果。它相当于 SQL 中的 count(*) 组合 group by。对于 MongoDB 中的聚合操作，应该使用aggregate()方法。

简述MongoDB中的GridFS机制?

GridFS是一种将大型文件存储在MongoDB中的文件规范。使用GridFS可以将大文件分隔成多个小文档存放，这样我们能够有效的保存大文档，而且解决了BSON对象有限制的问题。

简述MongoDB针对查询优化的措施?

MongoDB查询优化大致可能从如下步骤着手：

- 第一步：找出慢速查询

如下方式开启内置的查询分析器,记录读写操作效率：

db.setProfilingLevel(n,{m}),n的取值可选0,1,2；

  - 0：默认值，表示不记录；
  - 1：表示记录慢速操作，如果值为1，m必须赋值单位为ms，用于定义慢速查询时间的阈值；
  - 2：表示记录所有的读写操作。

查询监控结果：监控结果保存在一个特殊的集合system.profile里。

- 第二步：分析慢速查询

找出慢速查询的原因，通常可能的原因有：应用程序设计不合理、不正确的数据模型、硬件配置问题、缺少索引等

- 第三部：根据不同的分析结果进行优化，如建立索引。

简述MongoDB的更新操作是否会立刻fsync到磁盘?

不会，磁盘写操作默认是延时执行的，写操作可能在两三秒（默认在60秒内）后到达磁盘，可通过syncPeriodSecs参数进行配置。

简述MySQL索引及其作用?

是数据库管理系统中一个排序的数据结构，根据不同的存储引擎索引分为Hash索引、B+树索引等。常见的InnoDB存储引擎的默认索引实现为：B+树索引。

索引可以协助快速查询、更新数据库表中数据。

简述MySQL中什么是事务?

事务是一系列的操作，需要要符合ACID特性，即：事务中的操作要么全部成功，要么全部失败。

简述MySQL事务之间的隔离?

MySQL事务支持如下四种隔离：

未提交读(Read Uncommitted)：允许脏读，其他事务只要修改了数据，即使未提交，本事务也能看到修改后的数据值。也就是可能读取到其他会话中未提交事务修改的数据。

提交读(Read Committed)：只能读取到已经提交的数据。Oracle等多数数据库默认都是该级别 (不重复读)。

可重复读(Repeated Read)：可重复读。无论其他事务是否修改并提交了数据，在这个事务中看到的数据值始终不受其他事务影响。

串行读(Serializable)：完全串行化的读，每次读都需要获得表级共享锁，读写相互都会阻塞。

简述MySQL锁及其作用?

锁机制是为了避免，在数据库有并发事务的时候，可能会产生数据的不一致而诞生的的一个机制。锁从类别上分为：

共享锁：又叫做读锁，当用户要进行数据的读取时，对数据加上共享锁，共享锁可以同时加上多个。

排他锁：又叫做写锁，当用户要进行数据的写入时，对数据加上排他锁，排他锁只可以加一个，他和其他的排他锁,共享锁都相斥。

简述MySQL表中为什么建议添加主键?

主键是数据库确保数据行在整张表唯一性的保障，即使数据库中表没有主键，也建议添加一个自增长的ID列作为主键，设定了主键之后，在后续的删改查的时候可能更加快速以及确保操作数据范围安全。

简述MySQL所支持的存储引擎?

MySQL支持多种存储引擎，常见的有InnoDB、MyISAM、Memory、Archive等。通常使用InnoDB引擎都是最合适的，InnoDB也是MySQL的默认存储引擎。

简述MySQL InnoDB引擎和MyISAM引擎的差异?

- InnoDB支持事物，而MyISAM不支持事物。
- InnoDB支持行级锁，而MyISAM支持表级锁。
- InnoDB支持MVCC, 而MyISAM不支持。
- InnoDB支持外键，而MyISAM不支持。
- InnoDB不支持全文索引，而MyISAM支持。

简述MySQL主从复制过程?

1. Slave上面的IO线程连接上Master，并请求从指定日志文件的指定位置（或者从最开始的日志）之后的日志内容；
2. Master接收到来自Slave的IO线程的请求后，通过负责复制的IO线程根据请求信息读取指定日志指定位置之后的日志信息，返回给Slave端的IO线程。返回信息中除了日志所包含的信息之外，还包括本次返回的信息在Master端binary log文件的名称以及在Binary log中的位置；
3. Slave的IO线程收到信息后，将接收到的日志内容依次写入到Slave端的RelayLog文件（mysql-relay-lin.xxxxx）的最末端，并将读取到的Master端的bin-log的文件名和位置记录到master-info文件中，以便在下一次读取的时候能够明确知道从什么位置开始读取日志；
4. Slave的SQL线程检测到Relay Log中新增加了内容后，会马上解析该Log文件中的内容成为在Master端真实执行时候的那些可执行的查询或操作语句，并在自身执行那些查询或操作语句，这样，实际上就是在master端和Slave端执行了同样的查询或操作语句，所以两端的数据是完全一样的。

简述MySQL常见的读写分离方案?

MySQL+Amoeba读写分离方案：Amoeba(变形虫)项目，这个工具致力于MySQL的分布式数据库前端代理层，它主要在应用层访问MySQL的时候充当SQL路由功能。具有负载均衡、高可用性、SQL 过滤、读写分离、可路由相关的到目标数据库、可并发请求多台数据库合并结果。通过Amoeba你能够完成多数据源的高可用、负载均衡、数据切片、读写分离的功能。

MySQL+MMM读写分离方案：MMM即Multi-Master Replication Manager for MySQL，mysql多主复制管理器是关于mysql主主复制配置的监控、故障转移和管理的一套可伸缩的脚本套件(在任何时候只有一个节点可以被写入)。MMM也能对从服务器进行读负载均衡，通过MMM方案能实现服务器的故障转移，从而实现mysql的高可用。MMM不仅能提供浮动IP的功能，如果当前的主服务器挂掉后，会将你后端的从服务器自动转向新的主服务器进行同步复制，不用手工更改同步配置。

简述MySQL常见的高可用方案?

- MySQL主从复制：Mysql内建的复制功能是构建大型，高性能应用程序的基础。将Mysql的数据分布在多个节点（slaves）之上，复制过程中一个服务器充当主服务器，而一个或多个其它服务器充当从服务器。主服务器将更新写入二进制日志文件，并维护文件的一个索引以跟踪日志循环。这些日志可以记录发送到从服务器的更新。
- MySQL双主：参考MySQL主从复制。
- MySQL双主多从：参考MySQL主从复制。
- MySQL复制+Keepalived高可用：MySQL自身的复制，对外基于Keepalived技术，暴露一个VIP，从而实现高可用。
- Heartbeat + MySQL 实现MySQL的高可用：通过Heartbeat的心跳检测和资源接管、集群中服务的监测、失效切换等功能，结合MySQL来实现高可用性。

简述MySQL常见的优化措施?

MySQL可通过如下方式优化：

1. 开启查询缓存，优化查询。
2. 使用explain判断select查询，从而分析查询语句或是表结构的性能瓶颈，然后有针对性的进行优化。
3. 为搜索字段建索引
4. 对于有限定范围取值的字段，推荐使用 ENUM 而不是 VARCHAR。
5. 垂直分表。
6. 选择正确的存储引擎。

简述MySQL常见备份方式和工具?

- MySQL自带

mysqldump：mysqldump支持基于innodb的热备份，使用mysqldump完全备份+二进制日志可以实现基于时间点的恢复，通常适合备份数据比较小的场景 。

- 系统层面

tar备份：可以使用tar之类的系统命令对整个数据库目录进行打包备份。

lvm快照备份：可基于文件系统的LVM制作快照，进行对整个数据库目录所在的逻辑卷备份。

- 第三方备份工具

可使用其他第三方工具进行备份，如xtrabackup工具，该工具支持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。

简述常见的监控软件?

常见的监控软件有：

- Cacti：是一套基于PHP、MySQL、SNMP及RRDTool开发的网络流量监测图形分析工具。
- Zabbix：Zabbix是一个企业级的高度集成开源监控软件，提供分布式监控解决方案。可以用来监控设备、服务等可用性和性能。
- Open-falcon：open-falcon是一款用golang和python写的监控系统，由小米启动这个项目。

- Prometheus：Prometheus是由SoundCloud开发的开源监控报警系统和时序列数据库(TSDB)。Prometheus使用Go语言开发，是Google BorgMon监控系统的开源版本。

简述Prometheus及其主要特性?

Prometheus是一个已加入CNCF的开源监控报警系统和时序列数据库项目，通过不同的组件完成数据的采集，数据的存储和告警。

Prometheus主要特性：

- 多维数据模型
    - 时间序列数据通过 metric 名和键值对来区分。
    - 所有的 metrics 都可以设置任意的多维标签。
    - 数据模型更随意，不需要刻意设置为以点分隔的字符串。
    - 可以对数据模型进行聚合，切割和切片操作。
    - 支持双精度浮点类型，标签可以设为全 unicode。
- 灵活的查询语句（PromQL），可以利用多维数据完成复杂的查询
- Prometheus server 是一个单独的二进制文件，不依赖（任何分布式）存储，支持 local 和 remote 不同模型
- 采用 http 协议，使用 pull 模式，拉取数据，或者通过中间网关推送方式采集数据
- 监控目标，可以采用服务发现或静态配置的方式
- 支持多种统计数据模型，图形化友好
- 高效：一个 Prometheus server 可以处理数百万的 metrics
- 适用于以机器为中心的监控以及高度动态面向服务架构的监控

简述Prometheus主要组件及其功能?

Prometheus 的主要模块包含：prometheus server, exporters, push gateway, PromQL, Alertmanager, WebUI 等。

1. prometheus server：定期从静态配置的 targets 或者服务发现（主要是DNS、consul、k8s、mesos等）的 targets 拉取数据，用于收集和存储时间序列数据。
2. exporters：负责向prometheus server做数据汇报，暴露一个http服务的接口给Prometheus server定时抓取。而不同的数据汇报由不同的exporters实现，比如监控主机有node-exporters，mysql有MySQL server exporter。
3. push gateway：主要使用场景为，当Prometheus 采用 pull 模式，可能由于不在一个子网或者防火墙原因，导致 Prometheus 无法直接拉取各个 target 数据。此时需要push gateway接入，以便于在监控业务数据的时候，将不同数据汇总, 由 Prometheus 统一收集。实现机制类似于zabbix-proxy功能。
4. Alertmanager：从 Prometheus server 端接收到 alerts 后，会进行去除重复数据，分组，并路由到对收的接受方式，发出报警，即主要实现prometheus的告警功能。AlertManager的整体工作流程如下图所示:
5. webui：Prometheus内置一个简单的Web控制台，可以查询指标，查看配置信息或者Service Discovery等，实践中通常结合Grafana，Prometheus仅作为Grafana的数据源。

简述Prometheus的机制?

Prometheus简单机制如下：

- Prometheus以其Server为核心，用于收集和存储时间序列数据。Prometheus Server 从监控目标中拉取数据，或通过push gateway间接的把监控目标的监控数据存储到本地HDD/SSD中。
- 用户接口界面通过各种UI使用PromQL查询语言从Server获取数据。
- 一旦Server检测到异常，会推送告警到AlertManager，由告警管理负责去通知相关方。

简述Prometheus中什么是时序数据?

Prometheus 存储的是时序数据,，时序数据是指按照相同时序(相同的名字和标签)，以时间维度存储连续的数据的集合。时序(time series) 是由名字(Metric)，以及一组 key/value 标签定义的，具有相同的名字以及标签属于相同时序。

简述Prometheus时序数据有哪些类型?

Prometheus 时序数据分为 Counter, Gauge, Histogram, Summary 四种类型。

- Counter：计数器表示收集的数据是按照某个趋势（增加 / 减少）一直变化的，通常用它记录服务请求总量，错误总数等。
- Gauge：计量器表示搜集的数据是一个瞬时的，与时间没有关系，可以任意变高变低，往往可以用来记录内存使用率、磁盘使用率等。
- Histogram：直方图 Histogram 主要用于对一段时间范围内的数据进行采样，（通常是请求持续时间或响应大小），并能够对其指定区间以及总数进行统计，通常我们用它计算分位数的直方图。
- Summary：汇总Summary 和 直方图Histogram 类似，主要用于表示一段时间内数据采样结果，（通常是请求持续时间或响应大小），它直接存储了 quantile 数据，而不是根据统计区间计算出来的。

简述Zabbix及其优势?

Zabbix是一个企业级的高度集成开源监控软件，提供分布式监控解决方案。可以用来监控设备、服务等可用性和性能。其主要优势有：

- 自由开放源代码产品，可以对其进行任意修改和二次开发，采用GPL协议；
- 安装和配置简单；
- 搭建环境简单，基于开源软件构建平台；
- 完全支持Linux、Unix、Windows、AIX、BSD等平台，采用C语言编码，系统占用小，数据采集性能和速度非常快；
- 数据采集持久存储到数据库，便于对监控数据的二次分析；
- 非常丰富的扩展能力，轻松实现自定义监控项和实现数据采集。

简述Zabbix体系架构?

Zabbix体系相对清晰，其主要组件有：

- Zabbix Server：负责接收agent发送的报告信息的核心组件，所有配置、统计数据及操作数据均由其组织进行。
- Database Storage：专用于存储所有配置信息，以及有zabbix收集的数据。
- Web interface（frontend）：zabbix的GUI接口，通常与server运行在同一台机器上。
- Proxy：可选组件，常用于分布式监控环境中，代理Server收集部分被监控数据并统一发往Server端。
- Agent：部署在被监控主机上，负责收集本地数据并发往Server端或者Proxy端。

简述Zabbix所支持的监控方式?

目前由zabbix提供包括但不限于以下事项类型的支持：

- Zabbix agent checks：这些客户端来进行数据采集，又分为Zabbix agent（被动模式：客户端等着服务器端来要数据），Zabbix agent (active)（主动模式：客户端主动发送数据到服务器端）
- SNMP agent checks：SNMP方式，如果要监控打印机网络设备等支持SNMP设备的话，但是又不能安装agent的设备。
- SNMP traps：
- IPMI checks：IPMI即智能平台管理接口，现在是业界通过的标准。用户可以利用IPMI监视服务器的物理特性，如温度、电压、电扇工作状态、电源供应以及机箱入侵等。

简述Zabbix分布式及其适应场景?

zabbix proxy 可以代替 zabbix server 收集性能和可用性数据,然后把数据汇报给 zabbix server，并且在一定程度上分担了zabbix server 的压力。

此外，当所有agents和proxy报告给一个Zabbix server并且所有数据都集中收集时，使用proxy是实现集中式和分布式监控的最简单方法。

zabbix proxy 使用场景:

- 监控远程区域设备
- 监控本地网络不稳定区域
- 当 zabbix 监控上千设备时,使用它来减轻 server 的压力
- 简化分布式监控的维护

网络管理

简述什么是CDN?

CDN即内容分发网络，是在现有网络中增加一层新的网络架构，从而实现将源站内容发布和传送到最靠近用户的边缘地区，使用户可以就近访问想要的内容，提高用户访问的响应速度。

集群相关

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。

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。

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。

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。

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。

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。

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。

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。

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。

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。

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。

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。

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。

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。

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。

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。

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。

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。

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。

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。

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。

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。

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。

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。

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。

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。

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。

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。

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。

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。

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。

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。

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。

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。

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。

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。

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。

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。

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。

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。

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。

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。

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。

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。

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。

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。

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。

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。

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。

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。

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)。

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:

1. 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。

2. 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。
3. 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。

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。

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。

其他内容

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。