0-一：MYSQL

mysql主从复制原理：

在从节点上执行sart slave命令开启主从复制开关，开始进行主从复制。从节点上的I/O 进程连接主节点，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；

主节点接收到来自从节点的I/O请求后，通过负责复制的I/O进程（log dump 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，返回给从节点。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的bin-log file 的以及bin-log position（bin-log中的下一个指定更新位置）；

从节点的I/O进程接收到主节点发送过来的日志内容、日志文件及位置点后，将接收到的日志内容更新到本机的relay-log（中继日志）的文件（Mysql-relay-bin.xxx）的最末端，并将读取到的binary log（bin-log）文件名和位置保存到master-info 文件中，以便在下一次读取的时候能够清楚的告诉Master“我需要从某个bin-log 的哪个位置开始往后的日志内容，请发给我”；

Slave 的 SQL线程检测到relay-log 中新增加了内容后，会将relay-log的内容解析成在主节点上实际执行过SQL语句，然后在本数据库中按照解析出来的顺序执行，并在relay-log.info中记录当前应用中继日志的文件名和位置点。

如何监控mysql主从同步延迟：

show slave status\G的输出结果，需要监控下面三个参数：

1）Slave\_IO\_Running：该参数可作为io\_thread的监控项，Yes表示io\_thread的和主库连接正常并能实施复制工作，No则说明与主库通讯异常，多数情况是由主从间网络引起的问题；

2）Slave\_SQL\_Running：该参数代表sql\_thread是否正常，YES表示正常，NO表示执行失败，具体就是语句是否执行通过，常会遇到主键重复或是某个表不存在。

3）Seconds\_Behind\_Master：是通过比较sql\_thread执行的event的timestamp和io\_thread复制好的event的timestamp(简写为ts)进行比较，而得到的这么一个差值；

NULL—表示io\_thread或是sql\_thread有任何一个发生故障，也就是该线程的Running状态是No，而非Yes。

该值为零，是我们极为渴望看到的情况，表示主从复制良好，可以认为lag不存在。

正值— 表示主从已经出现延时，数字越大表示从库落后主库越多。

负值— 几乎很少见，我只是听一些资深的DBA说见过，其实，这是一个BUG值，该参数是不支持负值的，也就是不应该出现。

mysql搜索引擎区别：

InnoDB：支持事务处理，支持外键，支持崩溃修复能力和并发控制。如果需要对事务的完整性要求比较高(比如银行)，要求实现并发控制(比如售票，那选择InnoDB有很大的优势。如果需要频繁的更新、删除操作的数据库，也可以选择InnoDB，因为支持事务的提交(commit)和回滚(rollback)。

MyISAM：插入数据快，空间和内存使用比较低。如果表主要是用于插入新记录和读出记录，那么选择MyISAM能实现处理高效率。如果应用的完整性、并发性要求比 较低，也可以使用。

MEMORY：所有的数据都在内存中，数据的处理速度快，但是安全性不高。如果需要很快的读写速度，对数据的安全性要求较低，可以选择MEMOEY。它对表的大小有要求，不能建立太大的表。所以，这类数据库只使用在相对较小的数据库表。

注意，同一个数据库也可以使用多种存储引擎的表。如果一个表要求比较高的事务处理，可以选择InnoDB。这个数据库中可以将查询要求比较高的表选择MyISAM存储。如果该数据库需要一个用于查询的临时表，可以选择MEMORY存储引擎。

mysql事务：

1、事务定义：

1）.事务：事务是一个最小的不可在分的工作单元；通常一个事务对应一个完整的业务(例如银行账户转账业务，该业务是一个最小的工作单元)

2）.一个完整的业务需要批量的DML(insert、update、delete)语句共同联合完成。

3）.事务只和DML语句有关，或者说DML语句才有事务。这个和业务逻辑有关，业务逻辑不同，DML语句的个数不同。

2、事务特性：

原子性（Atomicity）：指事务是一个不可分割的最小工作单位，事务中的操作只有都发生和都不发生两种情况

一致性（Consistency）：事务必须使数据库从一个一致状态变换到另外一个一致状态，举一个栗子，李二给王五转账50元，其事务就是让李二账户上减去50元，王五账户上加上50元；一致性是指其他事务看到的情况是要么李二还没有给王五转账的状态，要么王五已经成功接收到李二的50元转账。而对于李二少了50元，王五还没加上50元这个中间状态是不可见的。

隔离性（Isolation）：一个事务的执行不能被其他事务干扰，即一个事务内部的操作及使用的数据对并发的其他事务是隔离的，并发执行的各个事务之间不能互相干扰。

持久性（Durability）：一个事务一旦提交成功，它对数据库中数据的改变将是永久性的，接下来的其他操作或故障不应对其有任何影响。

三、开启事务的步骤:start transaction;

四、事务并发时出现的问题

脏读：对于两个事务T1,T2，T1读取了已经被T2更新但还没有被提交的字段之后，若T2回滚，T1读取的内容就是临时且无效的

不可重复读 ：对于两个事务T1,T2，T1读取了一个字段，然后T2更新了该字段之后，T1在读取同一个字段，值就不同了

幻读：对于两个事务T1,T2，T1在A表中读取了一个字段，然后T2又在A表中插入了一些新的数据时，T1再读取该表时，就会发现神不知鬼不觉的多出几行了

五、事务的隔离级别

mysql中的四种事务隔离级别如下：

1. read uncommitted（读未提交数据）：允许事务读取未被其他事务提交的变更。（脏读、不可重复读和幻读的问题都会出现）。

2. read committed（读已提交数据）：只允许事务读取已经被其他事务提交的变更。（可以避免脏读，但不可重复读和幻读的问题仍然可能出现）

3.repeatable read（可重复读）：确保事务可以多次从一个字段中读取相同的值，在这个事务持续期间，禁止其他事务对这个字段进行更新(update)。（可以避免脏读和不可重复读，但幻读仍然存在）

4. serializable（串行化）：确保事务可以从一个表中读取相同的行，在这个事务持续期间，禁止其他事务对该表执行插入、更新和删除操作，所有并发问题都可避免，但性能十分低下（因为你不完成就都不可以弄，效率太低）

二：k8s及容器

K8S create和apply的区别：

create 顾名思义就是创建新资源。这里我们需要注意的是，如果再次运行相同的命令，就会抛出错误，因为资源名称在名称空间中应该是唯一的。

apply使配置在资源上生效。 这里分为两种情况，如下：

资源本不存在：创建新资源。

资源已存在：使配置在已存在的资源上生效（也就是更新资源的配置）。若配置没变化，则对应的资源也不会有什么改变。

k8s控制器：

1、Deployment：适合无状态的服务部署

2、StatefullSet：适合有状态的服务部署（redis）

3、DaemonSet：一次部署，所有的node节点都会部署，例如一些典型的应用场景：

运行集群存储 daemon，例如在每个Node上运行 glusterd、ceph

在每个Node上运行日志收集 daemon，例如 fluentd、 logstash

在每个Node上运行监控 daemon，例如 Prometheus Node Exporter

4、Job：一次性的执行任务

5、Cronjob：周期性的执行任务

k8s rs、rc、deploy区别：

Pod控制器有很多种，最初的是使用 ReplicationController，即副本控制器，用于控制pod数量。随着版本升级，出现了ReplicaSet，跟ReplicationController没有本质的不同，只是名字不一样，并且ReplicaSet支持集合式的selector。ReplicaSet的核心管理对象有三种：用户期望的副本数、标签选择器、pod模板。ReplicaSet一般不会直接使用，而是采用Deployment，Deployment是用来管理Replicaset，ReplicaSet来管理Pod。Deployment为ReplicaSet 提供了一个声明式定义(declarative)方法，用来替代以前的 ReplicationController 来方便的管理应用，比ReplicaSet的功能更加强大，且包含了ReplicaSet的功能。Deployment支持以下功能：

定义Deployment来创建Pod和ReplicaSet

滚动升级和回滚应用

扩容和缩容

暂停部署功能和手动部署

简述K8s中Pod的健康检查方式？

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

1. 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"。
2. ReadinessProbe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Endpoint。

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

K8s探针及使用：

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：

1. ExecAction：在容器内执行一个命令，若返回码为0，则表明容器健康。
2. 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。
3. 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。

Docker copy和add的区别、entrypoint和CMD的区别：

Docker网络模式：

默认网络当你安装 Docker 时，它会自动创建三个网络 bridge （创建容器默认连接到此网络）、none、host。你可以使用以下 docker network ls 或者 docker network list 命令列出这些网络

使用 docker run 创建 docker 容器时，可以用 \*\*--net \*\*或 \*\*--network \*\*选项指定容器的网络模式host 模式：使用 \*\*--net=host \*\*指定none 模式：使用 \*\*--net=none \*\*指定container 模式：使用 \*\*--net=container:${NAME/ID} \*\*指定bridge 模式：使用 \*\*--net=bridge \*\*指定，默认设置，可省略

1、Host 模式

host 模式：使用 --net=host 指定

相当于 VMware 中的桥接模式，与宿主机在同一个网络中，但是没有独立 IP 地址，直接使用宿主机的 IP 地址与外界进行通信，不再需要额外进行转换。

docker 启动时指定-network=host 或-net=host，此时通过-p 映射端口不会气到任何作用，端口号会以主机端口号为主，重复时则递增。

Docker 使用了 Linux 的 Namespace 技术来进行资源隔离，如 PID Namespace 隔离进程，Mount Namespace 隔离文件系统，Network Namespace 隔离网络等。一个 Network Namespace 提供了一份独立的网络环境，包括网卡，路由，iptable 规则等都与其他 Network Namespace 隔离。一个 Docker 容器一般会分配一个独立的 Network Namespace。

但是如果启动容器的时候使用 host 模式，那么这个容器将不会获得一个独立的 Network Namespace ，而是和宿主机共用一个 Network Namespace 。容器将不会虚拟出自己的网卡，配置自己的 IP 等，而是使用宿主机的 IP 和端口范围。此时容器不再拥有隔离的、独立的网络栈，不拥有端口资源。

容器的网络使用的是宿主机的网络，但是，容器的其他方面，如文件系统、进程列表等还是和宿主机隔离的。

2、 none 模式

使用 none 模式，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。

也就是说，这个 Docker 容器没有网卡、IP、路由等信息。这种网络模式下容器只有 lo 回环网络，没有其他网卡。这种类型的网络没有办法联网，封闭的网络能很好的保证容器的安全性。

3、 container 模式

这个模式指定新创建的容器和已经存在的一个容器共享一个 Network Namespace （网络命名空间），而不是和宿主机共享。

新创建的容器不会创建自己的网卡，配置自己的 IP，而是和一个指定的容器共享 IP、端口范围等。同样，两个容器除了网络方面，其他的如文件系统、进程列表等还是隔离的。两个容器的进程可以通过 lo 网卡设备通信。

4、 bridge 模式

bridge 模式是 docker 的默认网络模式，不用 --net 参数，就是 bridge 模式。

相当于 VMware 中的 nat 模式，容器使用独立 network Namespace ，并连接到 docker0 虚拟网卡。通过 docker0 网桥以及 iptables nat 表配置与宿主机通信，此模式会为每一个容器分配 Network Namespace、设置 IP 等，并将一个主机上的 Docker 容器连接到一个虚拟网桥上。

当 Docker 进程启动时，会在主机上创建一个名为 docker0 的虚拟网桥，此主机上启动的 Docker 容器会连接到这个虚拟网桥上。虚拟网桥的工作方式和物理交换机类似，这样主机上的所有容器就通过交换机连在了一个二层网络中。

从 docker0 子网中分配一个 IP 给容器使用，并设置 docker0 的 IP 地址为容器的默认网关。在主机上创建一对虚拟网卡 veth pair 设备。veth 设备总是成对出现的，它们组成了一个数据的通道，数据从一个设备进入，就会从另一个设备出来。因此，veth 设备常用来连接两个网络设备

容器和虚拟机的区别：

1、启动速度不同：docker 启动快速属于秒级别。虚拟机通常需要几分钟去启动。

2、性能损耗不同：docker 需要的资源更少，docker 在操作系统级别进行虚拟化，docker 容器和内核交互，几乎没有性能损耗，性能优于通过 Hypervisor 层与内核层的虚拟化。

3、系统利用率不同：docker 更轻量，docker 的架构可以共用一个内核与共享应用程序库，所占内存极小。同样的硬件环境，Docker 运行的镜像数远多于虚拟机数量，对系统的利用率非常高。

4、隔离性不同：与虚拟机相比，docker 隔离性更弱，docker 属于进程之间的隔离，虚拟机可实现系统级别隔离。

5、安全性不同：docker 的安全性也更弱。Docker 的租户 root 和宿主机 root 等同，一旦容器内的用户从普通用户权限提升为 root 权限，它就直接具备了宿主机的 root 权限，进而可进行无限制的操作。虚拟机租户 root 权限和宿主机的 root 虚拟机权限是分离的，并且虚拟机利用如 Intel 的 VT-d 和 VT-x 的 ring-1 硬件隔离技术，这种隔离技术可以防止虚拟机突破和彼此交互，而容器至今还没有任何形式的硬件隔离，这使得容器容易受到攻击。

6、可管理性不同：docker 的集中化管理工具还不算成熟。各种虚拟化技术都有成熟的管理工具，例如 VMware vCenter 提供完备的虚拟机管理能力。

7、可用和可恢复性不同：docker 对业务的高可用支持是通过快速重新部署实现的。虚拟化具备负载均衡，高可用，容错，迁移和数据保护等经过生产实践检验的成熟保障机制，VMware 可承诺虚拟机 99.999% 高可用，保证业务连续性。

8、创建、删除速度不同：虚拟化创建是分钟级别的，Docker容器创建是秒级别的，Docker 的快速迭代性，决定了无论是开发、测试、部署都可以节约大量时间。

9、交付、部署速度不同”虚拟机可以通过镜像实现环境交付的一致性，但镜像分发无法体系化；Docker 在 Dockerfile 中记录了容器构建过程，可在集群中实现快速分发和快速部署;

k8s master节点和node节点组件的区别：

master节点：

\*\*api server：\*\*负责对外提供restful的Kubernetes API服务，提供了资源操作的唯一入口，并提供认证、授权、访问控制、API 注册和发现等机制。我们操作kubectl 和其他Master组件都通过调用api server提供的rest接口实现各自的功能，如controller就是通过api server来实时监控各个资源的状态的。

\*\*etcd：\*\*是 Kubernetes 提供的一个高可用的键值数据库，用于保存集群所有的网络配置和资源对象的状态信息，也就是保存了整个集群的状态。数据变更都是通过api server进行的。整个kubernetes系统中一共有两个服务需要用到etcd用来协同和存储配置，分别是：

1）网络插件flannel，其它网络插件也需要用到etcd存储网络的配置信息；

2）kubernetes本身，包括各种资源对象的状态和元信息配置。

\*\*scheduler：\*\*监听新建pod副本信息，并通过调度算法为该pod选择一个最合适的Node节点。会检索到所有符合该pod要求的Node节点，执行pod调度逻辑。调度成功之后，会将pod信息绑定到目标节点上，同时将信息写入到etcd中。一旦绑定，就由Node上的kubelet接手pod的接下来的生命周期管理。Kubernetes目前提供了调度算法，但是同样也保留了接口，用户可以根据自己的需求定义自己的调度算法。

\*\*controller manager：\*\*负责维护集群的状态，比如故障检测、自动扩展、滚动更新等。每个资源一般都对应有一个控制器，这些controller通过api server实时监控各个资源的状态，controller manager就是负责管理这些控制器的。当有资源因为故障导致状态变化，controller就会尝试将系统由“现有状态”恢复到“期待状态”，保证其下每一个controller所对应的资源始终处于期望状态。比如我们通过api server创建一个pod，当这个pod创建成功后，api server的任务就算完成了。其中一个pod出现问题，controller会自动恢复创建新的pod。

node节点：

kubelet: 会监视已分配给节点的pod，负责pod的生命周期管理，同时与Master密切协作，维护和管理该Node上面的所有容器，实现集群管理的基本功能。即Node节点通过kubelet与master组件交互，可以理解为kubelet是Master在每个Node节点上面的agent。本质上，它负责使Pod的运行状态与期望的状态一致。

kube-proxy: 是实现service的通信与负载均衡机制的重要组件，将到service的请求转发到后端的pod上。

docker-engine(docker): 是负责容器的创建和管理工作。

Pod: 最小部署单元，一个Pod 有一个或多个容器组成，容器中共享存储和网络，在同一台docker主机上运行。

jenkins构建过程：

1) 在 GitLab 中创建对应的项目；

2) 配置 Jenkins 集成 Kubernetes 集群，后期 Jenkins 的 Slave 将为在 Kubernetes 中动态创

建的 Slave；

3) Jenkins 创建对应的任务（Job），集成该项目的 Git 地址和 Kubernetes 集群；

4) 开发者将代码提交到 GitLab；

5) 如有配置钩子，推送（Push）代码会自动触发 Jenkins 构建，如没有配置钩子，需要手

动构建；

6) Jenkins 控制 Kubernetes（使用的是 Kubernetes 插件）创建 Jenkins Slave（Pod 形式）；

7) Jenkins Slave 根据流水线（Pipeline）定义的步骤执行构建；

8) 通过 Dockerfile 生成镜像；

9) 将镜像提送（Push）到私有 Harbor（或者其它的镜像仓库）；

10) Jenkins 再次控制 Kubernetes 进行最新的镜像部署；

11) 流水线结束删除 Jenkins Slave。

三、NGINX

反向代理：

反向代理服务器位于用户与目标服务器之间，但是对于用户而言，反向代理服务器就相当于目标服务器，即用户直接访问反向代理服务器就可以获得目标服务器的资源。同时，用户不需要知道目标服务器的地址，也无须在用户端作任何设定。反向代理服务器通常可用来作为Web加速，即使用反向代理作为Web服务器的前置机来降低网络和服务器的负载，提高访问效率

特点：

1.反向代理服务器是介于用户和服务器之间的

2.用户直接访问代理服务器,以为代理服务器就是真实的服务器.

3.用户不清楚真实的服务器到底是谁.

4.反向代理是服务器端代理.保护了真实的服务器信息.

5.反向代理主要的作用 访问服务器获得服务器数据的.

NGINX有哪些模块：

nginx自带的针对后端节点健康检查的功能比较简单，通过默认自带的ngx\_http\_proxy\_module模块和ngx\_http\_upstream\_module，第三方健康检查模块：nginx\_upstream\_check\_module。

(1)Nginx核心功能模块

Nginx核心功能模块负责Nginx的全局应用，主要对应主配置文件的Main区块和Events区块区域，这里有很多Nginx必须的全局参数配置。

(2)标准的http功能模块集合

这些标准的http功能模块，虽然不是Nginx软件所必需的，但都是很常用的，因此绝大部分默认情况都会自动安装到Nginx软件中。不建议擅自改动，保留软件的默认配置就好，除非明确知道自身在做什么，有什么额外影响。在生产环境中，配置、调整及优化Nginx软件，主要就是根据这些模块的功能修改相应的参数来实现的。

企业场景常用的Nginx http功能模块有哪些？

ngx\_http\_core\_module包括一些核心的http参数配置，对应Nginx的配置为HTTP区块部分

ngx\_http\_access\_module访问控制模块，用来控制网站用户对Nginx的访问

ngx\_http\_gzip\_module压缩模块，对Nginx返回的数据压缩，属于性能优化模块

ngx\_http\_fastcgimodule FastCGI模块，和动态应用相关的模块，例如PHP

ngx\_http\_proxy\_module proxy代理模块

ngx\_http\_upstream\_module负载均衡模块，可以实现网站的负载均衡功能及节点的健康检查

ngx\_http\_rewrite\_module URL地址重写模块

ngx\_http\_limit\_conn\_module限制用户并发连接数及请求数模块

ngx\_http\_limit\_req\_module根据定义的key限制Nginx请求过程的速率

ngx\_http\_log\_module访问日志模块，以指定的格式记录Nginx客户访问日志等信息

ngx\_http\_auth\_basic\_module Web认证模块，设置Web用户通过账号、密码访问Nginx

ngx\_http\_ssl\_module ssl模块，用于加密的http连接，如https

ngx\_http\_stub\_status\_module记录Nginx基本访问状态信息等的模块

NGINX location优先级：

模式 意思

location =/url = 开头表示精确匹配,只有匹配上才生效

location ^~/url ^~ 开头表示前缀匹配

location ~pattern ~ 开头表示区分大小写的正则匹配

location ~\*pattern ~\*开头表示不区分大小写的正则匹配

location /url 不带任何开头的也表示前缀匹配，但是优先级在正则匹配之后

location / 通配匹配，任何未匹配到其他location的请求都会匹配到，相当于switch的default

nginx upstream的5种权重分配方式：

1、轮询（默认）：每个请求按时间顺序逐一分配到不同的后端服务器，后端服务器down掉，能自动剔除

2、weight:指定轮询几率，weight和访问比率成正比，用于后端服务器性能不均的情况。

3、ip\_hash:每个请求按访问ip的hash结果分配，这样每个访客固定访问一个后端服务器，可以解决session的问题。

4、fair（第三方）:按后端服务器的响应时间来分配请求，响应时间短的优先分配。

5、url\_hash（第三方）:按访问url的hash结果来分配请求，使每个url定向到同一个后端服务器，后端服务器为缓存时比较有效。例：在upstream中加入hash语句，server语句中不能写入weight等其他的参数，hash\_method是使用的hash算法

四、磁盘

Linux支持的常见文件系统：

Ext Linux 中最早的文件系统，由于在性能和兼容性上具有很多缺陷，现在已经很少使用

Ext2 是 Ext 文件系统的升级版本，Red Hat Linux 7.2 版本以前的系统默认都是 Ext2 文件系统。于 1993 年发布，支持最大 16TB 的分区和最大 2TB 的文件（1TB=1024GB=1024x1024KB)

Ext3 是 Ext2 文件系统的升级版本，最大的区别就是带日志功能，以便在系统突然停止时提高文件系统的可靠性。支持最大 16TB 的分区和最大 2TB 的文件

Ext4 是 Ext3 文件系统的升级版。Ext4 在性能、伸缩性和可靠性方面进行了大量改进。Ext4 的变化可以说是翻天覆地的，比如向下兼容 Ext3、最大 1EB 文件系统和 16TB 文件、无限数量子目录、Extents 连续数据块 概念、多块分配、延迟分配、持久预分配、快速 FSCK、日志校验、无日志模式、在线碎片整理、inode 增强、默认启用 barrier 等。它是 CentOS 6.3 的默认文件系统

xfs 被业界称为最先进、最具有可升级性的文件系统技术，由 SGI 公司设计，目前最新的 CentOS 7 版本默认使用的就是此文件系统。

swap swap 是 Linux 中用于交换分区的文件系统（类似于 Windows 中的虚拟内存)，当内存不够用时，使用交换分区暂时替代内存。一般大小为内存的 2 倍，但是不要超过 2GB。它是 Linux 的必需分区

NFS NFS 是网络文件系统（Network File System）的缩写，是用来实现不同主机之间文件共享的一种网络服务，本地主机可以通过挂载的方式使用远程共享的资源

iso9660 光盘的标准文件系统。Linux 要想使用光盘，必须支持 iso9660 文件系统

fat 就是 Windows 下的 fatl6 文件系统，在 Linux 中识别为 fat

vfat 就是 Windows 下的 fat32 文件系统，在 Linux 中识别为 vfat。支持最大 32GB 的分区和最大 4GB 的文件

NTFS 就是 Windows 下的 NTFS 文件系统，不过 Linux 默认是不能识别 NTFS 文件系统的，如果需要识别，则需要重新编译内核才能支持。它比 fat32 文件系统更加安全，速度更快，支持最大 2TB 的分区和最大 64GB 的文件

ufs Sun 公司的操作系统 Solaris 和 SunOS 所采用的文件系统

proc Linux 中基于内存的虚拟文件系统，用来管理内存存储目录 /proc

sysfs 和 proc —样，也是基于内存的虚拟文件系统，用来管理内存存储目录 /sysfs

tmpfs 也是一种基于内存的虚拟文件系统，不过也可以使用 swap 交换分区

RAID 0、RAID 1、RAID 5、RAID 10区别：

1、RAID 0：它是无数据冗余的存储空间条带化，具有成本低、读写性能高、存储空间利用率高等特点，适用于音、视频信号存储、临时文件的转储等对速度要求极其严格的特殊应用。但由于没有数据冗余，其安全性大大降低，构成阵列的任何一块硬盘的损坏都将带来灾难性的数据损失。这种方式其实没有冗余功能，没有安全保护，只是提高了磁盘读写性能和整个服务器的磁盘容量。

2、RAID 1：是两块硬盘数据完全镜像，安全性好、技术简单、管理方便、读写性能均好。因为它是一一对应的，所以它无法单块硬盘扩展，要扩展，必须同时对镜像的双方进行同容量的扩展。因为这种冗余方式为了安全起见，实际上只利用了一半的磁盘容量，数据空间浪费大。

3、RAID 0+1：也有写为RAID 10，综合了RAID 0和RAID 1的特点，独立磁盘配置成RAID 0，两套完整的RAID 0互相镜像。它的读写性能出色，安全性高，但构建阵列的成本投入大，数据空间利用率低。

4、RAID 5：是目前应用最广泛的RAID技术。各块独立硬盘进行条带化分割，相同的条带区进行奇偶校验，校验数据平均分布在每块硬盘上。以N块硬盘构建的RAID5阵列可以有N-1块硬盘的容量，存储空间利用率非常高。任何一块硬盘上的数据丢失，均可以通过校验数据推算出来，它和RAID 3最大的区别在于校验数据是否平均分布到各块硬盘上。RAID 5具有数据安全、读写速度快、空间利用率高等优点，应用非常广泛，但不足之处是如果1块硬盘出现故障以后，整个系统的性能将大大降低。

5、除了上面的4种常见的磁盘阵列外，还有其它几种磁盘阵列，比如：RAID 2、RAID 3、RAID 4、RAID 6、RAID 7等。

linux新磁盘如何处理：

磁盘分区（分区后磁盘才可用）：添加之后，可以使用fdisk -l来查看磁盘，我们添加的磁盘为/dev/sdb，此时磁盘还是无法使用的，需要先进行分区，接下来进行分区操作，将对新磁盘“/dev/sdb”做分区，使用fdisk /dev/sdb，大于2TB磁盘使用parted /dev/sdb

格式化：mkfs -t ext4 /dev/sdb1或者mkfs.ext4 -F /dev/sdb1

挂载：mount /dev/sdb1 /test

开机自动挂载:在挂载之后，就可以往磁盘里面写入数据了，但是使用mount命令挂载只是暂时的，在计算机重启之后就会消失，需要重新挂载，因此需要配置/etc/fstab文件，实现开机自动挂载

/etc/fstab文件详解：

/etc/fstab

# <file system> <dir> <type> <options> <dump> <pass>

tmpfs /tmp tmpfs nodev,nosuid 0 0

/dev/sda1 / ext4 defaults,noatime 0 1

/dev/sda2 none swap defaults 0 0

/dev/sda3 /home ext4 defaults,noatime 0 2

字段定义

/etc/fstab 文件包含了如下字段，通过空格或 Tab 分隔：

<file system> <dir> <type> <options> <dump> <pass>

第一列：<file systems> - 要挂载的分区或存储设备.

第二列：<dir> - <file systems>的挂载位置。

第三列：<type> - 要挂载设备或是分区的文件系统类型，支持许多种不同的文件系统：ext2, ext3, ext4, reiserfs, xfs, jfs, smbfs, iso9660, vfat, ntfs, swap 及 auto。 设置成auto类型，mount 命令会猜测使用的文件系统类型，对 CDROM 和 DVD 等移动设备是非常有用的。

第四列：<options> - 挂载时使用的参数，注意有些mount 参数是特定文件系统才有的。一些比较常用的参数有：

auto - 在启动时或键入了 mount -a 命令时自动挂载。

noauto - 只在你的命令下被挂载。

exec - 允许执行此分区的二进制文件。

noexec - 不允许执行此文件系统上的二进制文件。

ro - 以只读模式挂载文件系统。

rw - 以读写模式挂载文件系统。

user - 允许任意用户挂载此文件系统，若无显示定义，隐含启用 noexec, nosuid, nodev 参数。

users - 允许所有 users 组中的用户挂载文件系统.

nouser - 只能被 root 挂载。

owner - 允许设备所有者挂载.

sync - I/O 同步进行。

async - I/O 异步进行。

dev - 解析文件系统上的块特殊设备。

nodev - 不解析文件系统上的块特殊设备。

suid - 允许 suid 操作和设定 sgid 位。这一参数通常用于一些特殊任务，使一般用户运行程序时临时提升权限。

nosuid - 禁止 suid 操作和设定 sgid 位。

noatime - 不更新文件系统上 inode 访问记录，可以提升性能(参见 atime 参数)。

nodiratime - 不更新文件系统上的目录 inode 访问记录，可以提升性能(参见 atime 参数)。

relatime - 实时更新 inode access 记录。只有在记录中的访问时间早于当前访问才会被更新。（与 noatime 相似，但不会打断如 mutt 或其它程序探测文件在上次访问后是否被修改的进程。），可以提升性能(参见 atime 参数)。

flush - vfat 的选项，更频繁的刷新数据，复制对话框或进度条在全部数据都写入后才消失。

defaults - 使用文件系统的默认挂载参数，例如 ext4 的默认参数为:rw, suid, dev, exec, auto, nouser, async.

第五列：<dump> dump 工具通过它决定何时作备份. dump 会检查其内容，并用数字来决定是否对这个文件系统进行备份。 允许的数字是 0 和 1 。0 表示忽略， 1 则进行备份。大部分的用户是没有安装 dump 的 ，对他们而言 <dump> 应设为 0。

第六列：<pass> fsck 读取 <pass> 的数值来决定需要检查的文件系统的检查顺序。允许的数字是0, 1, 和2。 根目录应当获得最高的优先权 1, 其它所有需要被检查的设备设置为 2. 0 表示设备不会被 fsck 所检查。

五、zabbix监控

我们可以用zabbix监控哪些?

1. 硬件监控。 通过 SNMP 来进行路由器交换机的监控

2. 系统监控。 如 CPU 的负载，上下文切换、内存使用率、磁盘读写、磁盘使用率、磁盘 inode 节点。

3. 服务监控。 比如公司用 LNMP nginx 自带 Status 模块、 PHP 也有相关的 Status 、 MySQL 的话可以通过 percona 来进行监控。 4. 网络监控。 如果是云主机又不是跨机房，那么可以选择不监控网络。

5. 安全监控。 如果是云主机可以考虑使用自带的安全防护。当然也可以使用 iptables 。 如果是硬件，那么推荐使用硬件防火墙。

6.Web 监控。 web 监控的话题其实还是很多。比如可以使用自带的 web 监控来监控页面相关的延迟、 js 响应时间、下载时间、等等。

7. 日志监控。 如果是 web 的话可以使用监控 Nginx 的 500x 日志。 PHP 的 ERROR 日志。

8. 流量分析。 平时我们分析日志都是拿 awk sed xxx 一堆工具来实现。 这样对我们统计 ip 、 pv 、 uv 不是很方便。那么可以使用百度统计、 google 统计、商业，让开发嵌入代码即可

9. 可视化。 通过 screen 以及引入一 些第三方的库来美化界面，同时我们也需要知道、订单量突然增加、 突然减少。或者说突然来了一大波流量，这流量从哪儿来，是不是推广了，

还是被攻击了。可以结合监控平来来梳理各个系统之间的业务关系。

10. 自动化监控。 如上我们做了那么多的工作，当然不能是一台一台的来加 key 实现。 可以通过Zabbix 的主动模式以及被动模式来实现。当然最好还是通过 API 来实现。

zabbix的主动监控与被动监控

主动监控和被动监控都是相对于被监控端主机而言的

默认 zabbix 采用被动监控

当监控主机达到一定量级后 ,zabbix 服务器会越来越慢 , 此时可以考虑使用主动监控 , 释放服务器的压力 zabbix 也支持分布式监控

被动监控 :server 向 Agent 发起连接 , 发送监控 key,Agent 接受请求 , 响应监控数据

主动监控 :Agent 向 server 发起连接 ,Agent 请求需要检测监控项目列表 ,server 相应 Agent 发送一个 items 列表 ,Agent 确认收到监控列表 ,TCP 连接完成 , 会话关闭 ,Agent 开始周期性收集数据

区别 :

server 不用每次需要数据都连接 Agent,Agent 会自己收集数据并处理数据 ,Server 仅需要保存数据 即可

配置zabbix自定义监控流程

①被监控端修改 Agent 配置文件 ( 修改是否允许自定义 key, 加载配置文件目录 )

②被监控端创建存放自定义 key 文件 ( 在 zabbix\_agentd.conf 文件中定义 ), 书写自定义 key

③重起客户端 agentd

④测试自定义 key 是否生效

⑤在 zabbix 页面创建监控模板 , 创建应用 , 创建监控项 , 关联主机