**Docker常规题**

1、Docker和虚拟机有啥不同？

答：Docker是轻量级的沙盒，在其中运行的只是应用，虚拟机里面还有额外的系统。

Docker利用了Linux内核中很多安全特性来保证不同容器之间的隔离，并且通过签名机制来对镜像进行验证。大量生产环境的部署证明，Docker虽然隔离型无法与虚拟机相比，但仍具有极高的安全性。

2、如何清理后台停止的容器？

答：可以使用 sudo docker rm $sudo(docker ps -a -q)

查看本地镜像：docker images

查看本地容器：docker ps -a

3、如何查看镜像支持的环境变量

答：可以使用docker run IMAGE env命令

4、当启动容器的时候提示：exec format error？该如何解决问题

答：检查启动命令是否有可执行权限，进入容器手工运行脚本进行排查。

5、本地的镜像文件都存放在哪里？

答：与Docker相关的本地资源都存放在/var/lib/docker/目录下，其中container目录存放容器信息，graph目录存放镜像信息，aufs目录下存放具体的内容文件。 如果希望将Docker的本地文件存储到其他分区，可以使用Linux软链接的方式来做。

docker的配置文件放在哪里？

Ubuntu系统下Docker的配置文件是/etc/default/docker，CentOS系统配置文件存放在/etc/sysconig/docker。

6、退出容器时候自动删除？

答：使用--rm选项，例如sudo docker run --rm -it ubuntu

7、如何停止所有正在运行的容器？

答：使用docker kill $(sudo docker ps -q)

8、如何清理批量后台停止的容器？

答：使用docker rm $(sudo docker ps -a -q)

9、如何临时退出一个正在交互的容器的终端，而不终止它？

答：按Ctrl+p，后按Ctrl+q，如果按Ctrl+c会使容器内的应用进程终止，进而会使容器终止。

10、如何批量清理临时镜像文件？

答：可以使用sudo docker rmi $(sudo docker images -q -f danging=true)

在我们构建镜像的过程中，常常需要使用build命令根据Dockerfile进行构建镜像，并且会build很多次，镜像名字也是相同的，那么就会出现很多虚悬镜像（临时镜像文件）

-f :显示满足条件的镜像；

-q :只显示镜像ID。

11、可以在一个容器中同时运行多个应用进程吗？

答：一般不推荐在同一个容器内运行多个应用进程，如果有类似需求，可以通过额外的进程管理机制，比如supervisord来管理所运行的进程。

12、如何控制容器占用系统资源（CPU，内存）的份额？

答：在使用docker create命令创建容器或docker run创建并运行容器的时候，可以使用-c 或 --cpu-shares[=0]参数来调整同期使用CPU的权重，使用-m 或 --memory参数来调整容器使用内存的大小。

13、什么是容器编排？

答：考虑一个应用程序有5-6个微服务的情况。现在，这些微服务放置在单独的容器中，但是如果没有容器编排，将无法进行通信。因此，由于编排意味着将所有乐器在音乐中和谐地融合在一起，因此类似的容器编排意味着单个容器中的所有服务可以一起工作以满足单个服务器的需求。

**K8s简答题**

**1、简述etcd及其特点？**

答：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：

1. 简单：支持REST风格的HTTP+JSON API
2. 安全：支持HTTPS方式的访问
3. 快速：支持并发 1k/s的写操作
4. 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。

**2、简述etcd适应的场景？**

答：etcd基于其优秀的特点，可广泛的应用于以下场景：

1. 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。
2. 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。
3. 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分，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。
4. 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。
5. 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。
6. 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。

**3、简述什么是Kubernetes？**

k8s可以说是云服务的“操作系统”，高效部署管理云服务；

k8s是轮船、舵手、船长，docker是上面的集装箱货物；

答：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。

答：答：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。

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

**4、简述Kubernetes如何实现集群管理？**

答：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。

**5、简述Kubernetes的优势、适用场景及其特点？**

答：Kubernetes作为一个完备的分布式系统支撑平台，其主要优势：

1. 容器编排
2. 轻量级
3. 开源
4. 弹性伸缩
5. 负载均衡

Kubernetes常见场景：

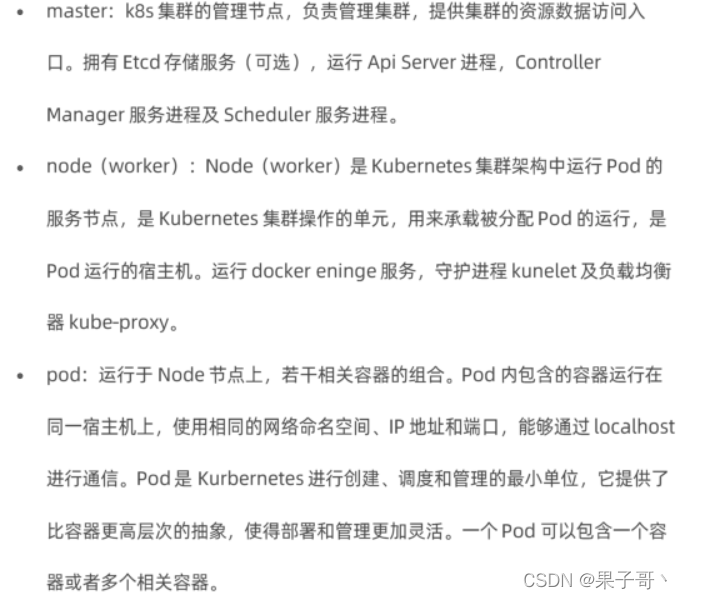
1. 快速部署应用
2. 快速扩展应用
3. 无缝对接新的应用功能
4. 节省资源，优化硬件资源的使用

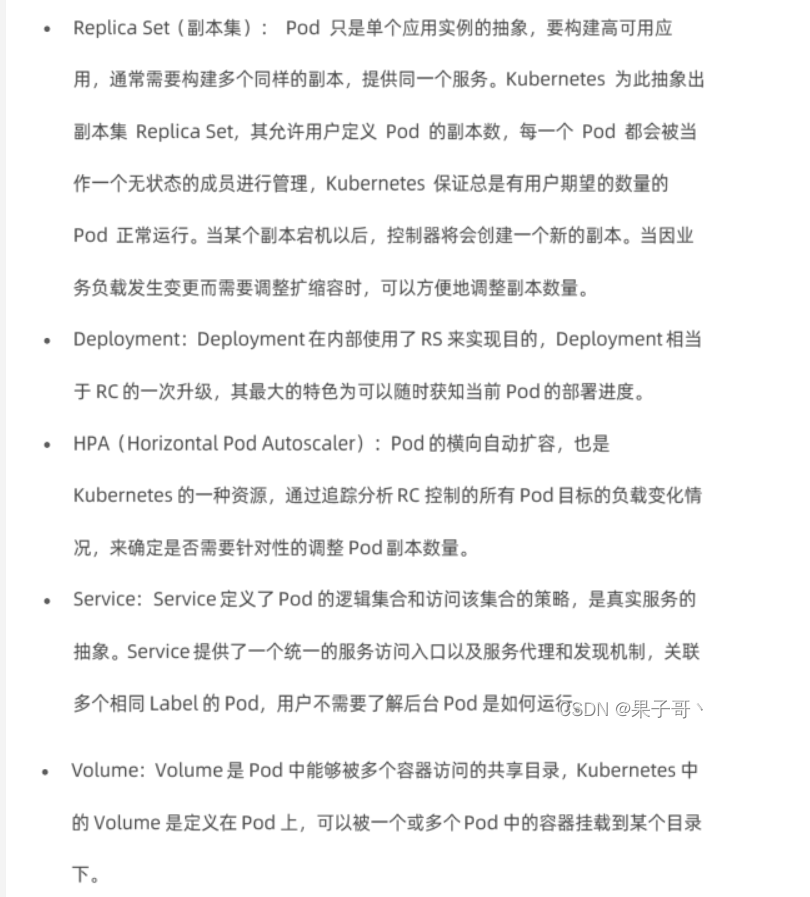
Kubernetes相关特点：

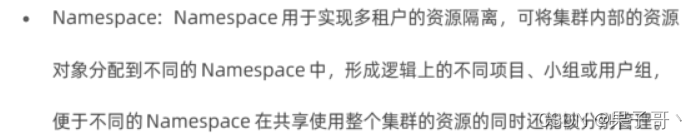
1. 可移植性：支持公有云、私有云、混合云、多重云。
2. 可扩展：模块化、插件化、可挂载、可组合
3. 自动化：自动部署、自欧东重启、自动复制、自动伸缩/扩展

**6、简述K8s相关基础概念**

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







**7、简述K8s集群相关组件**

答：K8s Master控制组件，调度管理整个系统（集群），包含如下组件：







**8、简述kube-proxy作用？**

答：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。

**9、简述kube-proxy iptables、ipvs原理？**

答：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。

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。

**10、简述kube-proxy ipvs和iptables的异同？**

答：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。

**11、简述K8s中什么是静态Pod？**

答：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。

**12、简述K8s中Pod可能处于的状态？**

答：

1. 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。
2. 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。
3. Succeeded：Pod内所有容器均成功执行退出，且不会重启。
4. 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。
5. 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。

**❤13、简述Kubernetes创建一个Pod的主要流程？**

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

1、用户通过kubectl命名发起请求

2、apiserver通过对应的kubeconfig进行认证，认证通过后将yaml中的Pod信息存储到etcd。

3、Controller-Manager通过apiserver的watch接口发现了Pod信息的更新，执行该资源所依赖的拓扑结构整合，整合后将对应的信息交给apiserver，apiserver写到etcd，此时Pod已经可以被调度了。

4、Scheduler同样通过apiserver的watch接口更新到Pod可以被调度，通过算法给Pod分配节点，并将Pod和对应节点绑定的信息交给apiserver，apiserver写到etcd，然后将Pod交给kubelet

5、kubelet收到Pod后，调用CNI接口给Pod创建Pod网络，调用CRI接口去启动容器，调用CSI进行存储卷的挂载。

6、网络，容器，存储创建完成后Pod创建完成，等业务进程启动后，Pod运行成功。



**14、简述K8s中的Pod的重启策略？**

答：Pod重启策略应用于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。

1. Always：当容器失效时，由kubelet自动重启该容器；
2. 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；
3. Never：不论容器运行状态如何，kubelet都不会重启
4. 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：

0 RC和DaemonSet：必须设置为Always，需要保证该容器持续运行；

1. Job:OnFailure 或Never，确保容器执行完成后不再重启；
2. 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。

**15、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。
3. 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。

**16、简述Kubernetes Pod的LivenessProbe探针的常见方式？**

答：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。

**17、简述Kubernetes Pod的常见调度方式？**

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

1. 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。

1. 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。

1. NodeAffinity：亲和性调度，扩展了Pod的调度能力，目前有两种节点亲和力表达

* 1. 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）
  2. 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。

1. Taints和Tolerations（污点和容忍）

* 1. Taint：使Node拒绝特定Pod运行
  2. Toleration：为Pod的属性，表示Pod能容忍（运行）标注了Taint的Node

**18、简述K8s初始化容器（init container）？**

答：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，K8s才会初始化Pod的各种信息，并开始创建和运行应用容器。

**19、简述K8s DaemonSet类型的资源特性？**

1. 答：DaemonSet资源对象会在每个K8s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：

1. 在去做每个节点的日志收集工作。
2. 监控每个节点的运行状态

**20、简述K8s自动扩容机制？**

答：K8s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。

1. HPA原理

K8s中的某个Metrics Server持续采集所有pod副本的指标数据。HPA控制器通过Metric Server的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控制器就会向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。

**21、简述K8s Service类型？**

答：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：

1. 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；
2. 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；
3. 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。

**22、简述K8s Service分发后端的策略？**

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

1. RoundRobin：默认为轮询模式，即轮询将请求转发到后端的各个Pod上。
2. 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。

**23、简述K8s Headless Service？**

答：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ClusterlP（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。

**24、简述K8s ingress？**

答：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。

**25、简述K8s 镜像的下载策略？**

答：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。

**26、简述K8s各模块如何与API Server通信？**

答：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。

**27、简述K8s Scheduler作用及实现原理？**

答：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（APl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监听到的K8s 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。

**28、简述K8s Scheduler使用哪两种算法将Pod绑定到worker节点？**

预选调度：筛选出可以被调度的节点。像是节点有故障或者资源不够Pod的申请量等情况肯定是不能作为被调度的节点的。优选调度：从刚才预选调度中筛选出来的可以被调度的节点中再一次进行筛选。这次要挑出来一个最适合被调度的节点。

答：K8s Scheduler根据如下两种调度算法将Pod绑定到最合适的工作节点：

1. 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”。

1. 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。

**29、简述K8s kubelet的作用？**

答：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。

**30、简述K8s kubelet监控Worker节点资源是使用什么组件实现？**

答：kubelet使用cAdvisor对worker节点资源进行监控。在K8s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节点上运行的容器的性能指标。

**31、简述K8s如何保证集群的安全性？**

答：主要有如下不同的维度：

1. 基础设施方面：保证容器与其所在宿主机的隔离；
2. 权限方面：
   1. 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。
   2. 用户权限：划分普通用户和管理员的角色
3. 集群方面：
   1. 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KubernetesAPI 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。
   2. 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。

* 1. 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。
  2. 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。

**32、简述K8s准入机制？**

答：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：

1. AlwaysAdmint：允许所有请求
2. AlwaysDeny：禁止所有请求，多用于测试环境。
3. 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。
4. 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。
5. 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。

**33、简述K8s RBAC及其特点（优势）？**

答：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：

1. 对集群中的资源和非资源权限均有完整的覆盖。
2. 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。
3. 可以在运行时进行调整，无须重新启动API Server。

**34、简述K8s Secret作用？**

答：Secret对象，主要作用是保管私密数据，比如密码、OAuth Tokens、SSHKeys等信息。将这些私密信息放在Secret对象中比直接放在Pod或Docker lmage中更安全，也更便于使用和分发。

创建完secret之后，可通过如下三种方式使用：

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

**35、简述K8s PodSecutiryPolicy机制？**

答：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。

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：

1. 特权模式：privileged是否允许Pod以特权模式运行。
2. 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。
3. 用户和组：设置运行容器的用户ID（范围）或组（范围）。
4. 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。
5. SELinux：进行SELinux的相关配置。

**36、简述K8s网络模型**

答：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）。

**37、简述K8s CNI模型？**

答：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。

1. 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。
2. 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（IPAddress 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CNIPlugin协同工作。

**38、简述Kubernetes网络策略？**

答：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。

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）。

**39、简述K8s中flannel的作用？**

答：Flannel可以用于Kubernetes底层网络的实现，主要作用有：

1. 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。
2. 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。

覆盖网络：简单说来覆盖网络就是应用层网络，它是面向应用层的，不考虑或很少考虑网络层，物理层的问题。详细说来，覆盖网络是指建立在另一个网络上的网络。该网络中的结点可以看作通过虚拟或逻辑链路而连接起来的

**40、简述K8s Calico网络组建实现原理**

答：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。

<https://zhuanlan.zhihu.com/p/378861682>

<https://blog.csdn.net/qq_24433609/article/details/120490220>

**❤（重要）简述你知道的几种CNI网络插件，并详述其工作原理。K8s常用的CNI网络插件（calico && flannel），简述它们的工作原理和区别。**

答：

**1、Calico工作原理**

calico根据iptables规则进行路由转发，并没有进行封包，解包的过程，这和flannel比起来效率就会快多

calico 包括如下重要组件：Felix，etcd，BGP Client，BGP Route Reflector。下面分别说明一下这些组件。

Felix：主要负责路由配置以及ACLS规则的配置以及下发，它存在在每个node节点

etcd：分布式键值存储，主要负责网络元数据一致性，确保Calico网络状态的准确性，可以与kubernetes共用；

BGPClient（BIRD），主要负责把Felix写入kernel的路由信息分发到当前 Calico网络，确保 workload间的通信的有效性；

BGPRoute Reflector（BIRD），大规模部署时使用，摒弃所有节点互联的mesh模式，通过一个或者多个BGPRoute Reftector来完成集中式的路由分发

**2、Flannel的工作原理**

Flannel实质上是一种“覆盖网络（overlay network）"，也就是将TCP数据包装在另一种网络包里面进行路由转发和通信，目前已经支持UDP、VXLAN、AWS VPC和GCE路由等数据转发方式。

默认的节点间数据通信方式是UDP转发。

工作原理：

数据从源容器中发出后，经由所在主机的 dockero虚拟网卡转发到ftannel0虚拟网卡（先可以不经过dockero网卡，使用cni模式），这是个P2P的虚拟网卡，flanneld服务监听在网卡的另外一端。

Flannel通过Etcd服务维护了一张节点间的路由表，详细记录了各节点子网网段。

源主机的flanneld服务将原本的数据内容UDP封装后根据自己的路由表投递给目的节点的flanneld服务，数据到达以后被解包，然后直接进入目的节点的flannel0虚拟网卡，然后被转发到目的主机的dockero虚拟网卡，最后就像本机容器通信一下的有docker0路由到达目标容器。

flannel在进行路由转发的基础上进行了封包解包的操作，这样浪费了CPU的计算资源。

**41、简述K8s 数据持久化的方式？**

答：Kubernetes通过数据持久化来持久化保存重要数据，常见的方式有：

1. 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.
2. 场景：
   1. 只需要临时将数据保存在磁盘上，比如在合井/排序算法中；
   2. 作为两个容器的共享存储。
3. 特性：
   1. 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。
   2. 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。
4. 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。
   1. 特性：增加了pod与节点之间的糊合。

PersistentVolume（简称PV）：如基于NF5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。

**42、简述K8s PV和PVC？**

答：

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”。

**43、简述K8s PV生命周期内的阶段？**

答：某个PV在生命周期中可能处于以下4个阶段（Phaes）之一。

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

**44、简述K8s所支持的存储供应模式？**

答：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）。

1. 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。
2. 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。

**45、简述K8s的CSI模型？**

答：KubernetesCSl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。CSl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。

CSl包括CSI Controller和CSI Node：

1. CSl 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。
2. CSl Node的主要功能是对主机（Node）上的Volume进行管理和操作。

**46、简述K8s Worker节点加入集群的过程？**

答：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. 1、在该Node上安装Docker、kubelet和kube-proxy服务；
2. 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. 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. 4、K8s 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。

**47、简述K8s Requests和Limits如何影响Pod的调度？**

答：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。

**48、简述K8s Metric Service？**

答：在K8s从1.10版本后采用Met日长石 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。

**49、如何使用EFK实现日志的统一管理？**

答：在K8s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：

1. Elasticsearch：是一个搜索引擎，负责存储日志并提供查询接口。
2. Fluentd（或使用filebeat）：负责从K8s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。
3. 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。

**50、简述K8s如何进行优雅的节点关机维护？**

1. 答：由于K8s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。

**51、简述Helm及其优势？**

答：

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。

1. 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）。
2. 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：

1. 统一管理、配置和更新这些分散的k8s的应用资源文件；
2. 分发和复用一套应用模板；
3. 将应用的一系列资源当做一个软件包管理。
   1. 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。
   2. 对于使用者而言，使用Helm后不需要编写复杂的应用部署文件，可以以简单的方式在K8s上查找、安装、升级、回滚、卸载应用程序。

**52、希望通过维持最低成本来提高其效率和技术运营速度。认为公司将如何实现这一目标？**

答：公司可以通过构建CI/CD管道来实现DevOps方法，但是这里可能出现的一个问题是配置可能需要一段时间才能启动并运行。因此，在实施CI/CD管道之后，公司的下一步应该是在云环境中工作。一旦他们开始处理云环境，他们就可以在集群上安排容器，并可以在Kubernetes的帮助下进行协调。这种方法将有助于公司缩短部署时间，并在各种环境中加快速度。

**❤53、Worker节点宕机，简述Pods驱逐流程。**

答：

1、概述：

在Kubernetes集群中，当节点由于某些原因（网络、宕机等）不能正常工作时会被认定为不可用状态（Unknown或者False状态），当时间超过了pod-eviction-

timeout值时，那么节点上的所有Pod都会被节点控制器计划删除。

2、Kubernetes 集群中有一个节点生命周期控制器：node\_lifecycle\_controller.go。

它会与每一个节点上的kubelet进行通信，以收集各个节点已经节点上容器的相关状态信息。当超出一定时间后不能与kubelet通信，那么就会标记该节点为Unknown状态。并且节点生命周期控制器会自动创建代表状况的污点，用于防止调度器调度pod到该节点。

3、那么Unknown状态的节点上已经运行的pod会怎么处理呢？节点上的所有Pod 都会被污点管理器（taint\_manager.go）计划删除。而在节点被认定为不可用状态到删除节点上的Pod之间是有一段时间的，这段时间被称为容忍度。你可以通过下面的方式来配置容忍度的时间长短：

tolerations:

- key: node.kubernetes.io/not-ready

operator: Exists

effect: NoExecute

tolerationSeconds: 180

- key: node.kubernetes.io/unreachable

operator: Exists

effect: NoExecute

tolerationSeconds: 180

如果在不配置的情况下，Kubernetes会自动给Pod添加一个key为node.kubernetes.io/not-ready的容忍度并配置 tolerationSeconds=300，同样，Kubernetes 会给Pod 添加一个key为node.kubernetes.io/unreachable的容忍度并配置 tolerationSeconds=300。

4、当到了删除Pod时，污点管理器会创建污点标记事件，然后驱逐pod。这里需要注意的是由于已经不能与kubelet通信，所以该节点上的Pod在管理后台看到的是处于灰色标记，但是此时如果去获取pod的状态其实还是处于Running状态。每种类型的资源都有相应的资源控制器（Controller），例如：

deployment\_controller.go、stateful\_set\_control.go。每种控制器都在监听资源变化，从而做出相应的动作执行。deployment 控制器在监听到Pod 被驱逐后会创建一个新的Pod出来，但是Statefulset 控制器并不会创建出新的Pod，原因是因为它可能会违反StatefulSet固有的至多一个的语义，可能出现具有相同身份的多个成员，这将可能是灾难性的，并且可能导致数据丢失。

**54、简述你知道的K8s中几种Controller控制器，并详述其工作原理。简述ingress-controller的工作机制。**

答：

**1、deployment：适合无状态的服务部署**

适合部署无状态的应用服务，用来管理pod和replicaset，具有上线部署、副本设定、滚动更新、回滚等功能，还可提供声明式更新，例如只更新一个新的Image

编写yaml文件，并创建nginx服务pod资源。

apiVersion: apps/v1

kind: Deployment

metadata:

name: nginx-deployment

labels:

app: nginx

spec:

replicas: 3

selector:

matchLabels:

app: nginx

template:

metadata:

labels:

app: nginx

spec:

containers:

- name: nginx1

image: nginx:1.15.4

ports:

- containerPort: 80

查看控制器参数：可以使用describe或者edit两种方式

kubectl describe deploy nginx-deployment

kubectl edit deploy nginx-deployment

1. 无状态服务的特点：
2. deployment认为所有的pod都是一样的
3. 不用考虑顺序的要求
4. 不用考虑在哪个node节点上运行
5. 可以随意扩容和缩容

**2、StatefulSet：适合有状态的服务部署**

1. 解决Pod的独立生命周期，保持Pod启动顺序和唯一性
2. 稳定，唯一的网络标识符，持久存储（例如：etcd配置文件，节点地址发生变化，将无法使用）
3. 有序，优雅的部署和扩展、删除和终止（例如：mysql主从关系，先启动主，再启动从）有序，滚动更新
4. 应用场景：例如数据库
5. 有状态服务的特点：
6. 实例之间有差别，每个实例都有自己的独特性，元数据不同，例如etcd，zookeeper
7. 实例之间不对等的关系，以及依靠外部存储的应用
8. 常规的service服务和无头服务的区别
9. service：一组Pod访问策略，提供cluster-IP群集之间通讯，还提供负载均衡和服务发现
10. Headless service无头服务，不需要cluster-IP，直接绑定具体的Pod的IP，无头服务经常用于statefulset的有状态部署
11. 创建无头服务的service资源和dns资源

由于有状态服务的IP地址是动态的，所以使用无头服务的时候要绑定dns服务

**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：

1. 运行集群存储daemon，例如在每个Node上运行glusterd、ceph
2. 在每个Node上运行日志收集 daemon，例如fluentd、logstas
3. 在每个Node 上运行监控daemon，例如Prometheus Node Exporter
4. 在每一个Node上运行一个Pod
5. 新加入的Node也同样会自动运行一个Pod
6. 应用场景：监控，分布式存储，日志收集等

**4、job：一次性的执行任务**

类似Linux中的job

应用场景：如离线数据处理，视频解码等业务。

**5、Cronjob：周期性的执行任务**

周期性任务，像Linux的Crontab一样

应用场景：如通知，备份等

使用cronjob要慎重，用完之后要删掉，不然会占用很多资源

**6、ingress-controller的工作机制**

通常情况下，service和pod的IP仅可以在集群内部访问。

k8s提供了service方式：NodePort来提供对外的服务，外部的服务可以通过访问Node 节点ip+NodePort端口来访问集群内部的资源，外部的请求先到达service所选中的节点上，然后负载均衡到每一个节点上。

NodePort虽然提供了对外的方式但也有很大弊端：

由于service的实现方式：user\_space、iptebles、3ipvs、方式这三种方式只支持在4层协议通信，不支持7层协议，因此NodePort不能代理https服务。

1. NodePort 需要暴露service所属每个node节点上端口，当需求越来越多，端口数量过多，导致维护成本过高，并且集群不好管理。
2. 原理：

Ingress也是KubernetesAPl的标准资源类型之一，它其实就是一组基于DNS名称（host）或URL路径把请求转发到指定的Service资源的规则。用于将集群外部的请求流量转发到集群内部完成的服务发布。我们需要明白的是，Ingress资源自身不能进行“流量穿透”，仅仅是一组规则的集合，这些集合规则还需要其他功能的辅助，比如监听某套接字，然后根据这些规则的匹配进行路由转发，这些能够为Ingress资源监听套接字并将流量转发的组件就是Ingress Controller。

Ingress 控制器不同于Deployment等pod控制器的是，Ingress控制器不直接运行为kube-controller-manager的一部分，它仅仅是Kubernetes集群的一个附件，类似于CoreDNS，需要在集群上单独部署。

ingress controller通过监视api server获取相关ingress、service、endpoint、secret、node、configmap 对象，并在程序内部不断循环监视相关service是否有新的endpoints变化，一旦发生变化则自动更新nginx.conf模板配置并产生新的配置文件进行reload。

54、简述K8s的调度机制。

答：

1、Scheduler工作原理

请求及Scheduler调度步骤：

1. 节点预选（Predicate）：排除完全不满足条件的节点，如内存大小，端口等条件不满足。
2. 节点优先级排序（Priority）：根据优先级选出最佳节点
3. 节点择优（Select）：根据优先级选定节点

2、具体步骤

首先用户通过Kubernetes客户端Kubectl提交创建Pod的Yaml的文件，向Kubernetes系统发起资源请求，该资源请求被提交到Kubernetes系统中，用户通过命令行工具Kubectl向Kubernetes 集群即APIServer用的方式发送“POST"请求，即创建Pod的请求。

APIServer 接收到请求后把创建Pod的信息存储到Etcd中，从集群运行那一刻起，资源调度系统 Scheduler 就会定时去监控APIServer通过APIServer 得到创建Pod的信息，Scheduler 采用watch机制，一旦Etcd存储Pod信息成功便会立即通知APIServer，APIServer会立即把Pod创建的消息通知Scheduler，Scheduler发现Pod的属性中Dest Node为空时（Dest Node=""）便会立即触发调度流程进行调度。

而这一个创建Pod对象，在调度的过程当中有3个阶段：节点预选、节点优选、节点选定，从而筛选出最佳的节点

节点预选：基于一系列的预选规则对每个节点进行检查，将那些不符合条件的节点过滤，从而完成节点的预选

节点优选：对预选出的节点进行优先级排序，以便选出最合适运行Pod对象的节点

节点选定：从优先级排序结果中挑选出优先级最高的节点运行Pod，当这类节点多于1个时，则进行随机选择

3、k8s的调度工作方式

Kubernetes调度器作为集群的大脑，在如何提高集群的资源利用率、保证集群中服务的稳定运行中也会变得越来越重要Kubernetes的资源分为两种属性。

可压缩资源（例如CPU循环，DiskI/O带宽）都是可以被限制和被回收的，对于一个Pod来说可以降低这些资源的使用量而不去杀掉Pod。

不可压缩资源（例如内存、硬盘空间）一般来说不杀掉Pod就没法回收。未来Kubernetes会加入更多资源，如网络带宽，存储IOPS的支持。

**54、简述kube-proxy的三种工作模式和原理**

答：

1、userspace模式

该模式下kube-proxy会为每一个Service创建一个监听端口。发向Cluster IP的请求被lptables规则重定向到Kube-proxy监听的端口上，Kube-proxy根据LB算法选择一个提供服务的Pod并和其建立链接，以将请求转发到Pod上。

该模式下，Kube-proxy 充当了一个四层Load balancer的角色。由于kube-proxy运行在userspace中，在进行转发处理时会增加两次内核和用户空间之间的数据拷贝，效率较另外两种模式低一些；好处是当后端的Pod不可用时，kube-proxy可以重试其他Pod。

2、iptables模式

为了避免增加内核和用户空间的数据拷贝操作，提高转发效率，Kube-proxy提供了iptables模式。在该模式下，Kube-proxy为service后端的每个Pod创建对应的iptables规则，直接将发向ClusterIP的请求重定向到一个Pod IP。

该模式下Kube-proxy不承担四层代理的角色，只负责创建iptables规则。该模式的优点是较userspace模式效率更高，但不能提供灵活的LB策略，当后端Pod不可用时也无法进行重试。

3、该模式和iptables类似，kube-proxy监控Pod的变化并创建相应的ipvs rules。

ipvs也是在kernel模式下通过netfitter实现的，但采用了hash table来存储规则，因此在规则较多的情况下，lpvs相对iptables转发效率更高。除此以外，ipvs支持更多的LB算法。如果要设置kube-proxy为ipvs模式，必须在操作系统中安装IPVS内核模块。

**55、K8s每个Pod中有一个特殊的Pause容器，能否去除，简述原因。**

答：

pause container作为pod 里其他所有container的 parent container，主要有两个职责：

1. 是pod里其他容器共享Linux namespace的基础
2. 扮演PID1的角色，负责处理僵尸进程

在Linux里，当父进程fork一个新进程时，子进程会从父进程继承 namespace。目前Linux实现了六种类型的namespace，每一个namespace是包装了一些全局系统资源的抽象集合，这一抽象集合使得在进程的命名空间中可以看到全局系统资源。命名空间的一个总体目标是支持轻量级虚拟化工具container的实现，container机制本身对外提供一组进程，这组进程自己会认为它们就是系统唯一存在的进程。

在Linux里，父进程fork的子进程会继承父进程的命名空间。与这种行为相反的一个系统命令就是unshare。

**56、简述pod中readness和liveness的区别和各自应用场景。**

答：

存活性探针（liveness probes）和就绪性探针（readiness probes）

用户通过Liveness 探测可以告诉Kubernetes什么时候通过重启容器实现自愈；

Readiness 探测则是告诉Kubernetes什么时候可以将容器加入到Service负载均衡池中，对外提供服务。语法是一样的。

主要的探测方式支持http探测，执行命令探测，以及tcp探测。

1、执行命令探测

kubelet是根据执行命令的退出码来决定是否探测成功。

当执行命令的退出码为0时，认为执行成功，否则为执行失败。如果执行超时，则状态为Unknown。

2、http探测

http探测是通过kubelet请求容器的指定url，并根据response来进行判断。

当返回的状态码在200到400（不含400）之间时，也就是状态码为2xx和3xx时，认为探测成功，否则认为失败。

3、tcp探测

tcp探测是通过探测指定的端口。如果可以连接，则认为探测成功，否则认为失败。

探测失败的可能原因：

执行命令探测失败的原因主要可能是容器未成功启动，或者执行命令失败。当然也可能docker或者docker-shim存在故障。

由于http和tcp都是从kubelet自node节点上发起的，向容器的ip进行探测。所以探测失败的原因除了应用容器的问题外，还可能是从node到容器ip的网络不通。

**57、简述K8s中label的几种应用场景。**

答：

一些Pod有Label。一个Label是attach到Pod的一对键值对，用来传递用户定义的属性。

比如，你可能创建了一个“tier”和“app”标签，通过Label（tier=frontend,app=myapp）来标记前端Pod容器

使用Label(tier=backend,app=myapp)标记后台Pod。

然后可以使用Selectors选择带有特定Label的Pod，并且将Service或者Replication Controller应用到上面。

**58、简述你知道的Jenkins Pipeline中脚本语法中的几个关键字。**

答：

pipeline是jenkins 2.X最核心的特性，帮助jenkins实现从CI到CD与DevOps的转变。

pipeline提供一组可扩展的工具，通过pipeline domain specific language syntax可以到达pipeline as code目的。

pipeline as code：jenkinsfile存储在项目的源代码库

为什么要使用pipeline？

1、代码：pipeline以代码的形式实现，通过被捡入源代码控制，使团队能够编译，审查和迭代其cd流程

2、可连续性：jenkins 重启或者中断后都不会影响 pipeline job

3、停顿：pipeline可以选择停止并等待人工输入或者批准，然后在继续 pipeline运行

4、多功能：pipeline支持现实世界的复杂CD要求，包括fork、join子进程，循环和并行执行工作的能力。

5、可扩展：pipeline插件支持其DSL的自动扩展以及插件集成的多个选项。

块（Blocks{}）

1. 由大括号括起来的语句：如Pipeline{}，Sections{}，parameters{}，script{}

章节（Sections）

1. 通常包括一个或者多个指令或步骤，如agent，post，stages，steps

指令（Directives）

1. environment，options，parameters，trigger，stage，tools，when

步骤（Steps）

1. 执行脚本式pipeline，如script{}

**59、Docker的网络通信模式**

1、host 模式，使用–net=host 指定。

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。

2、container模式，使用–net=container:NAMEorlD指定。指定新创建的容器和已经存在的一个容器共享一个Network Namespace，而不是和宿主机共享。

3、none模式，使用–net=none指定。

告诉docker 将新容器放到自己的网络堆栈中，但是不要配置它的网络。

4、bridge 模式，使用–net=bridge 指定，默认设置。

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，并将一个主机上的Docker容器连接到一个虚拟网桥上。

**60、Iptables四表五链**

答：

raw表：确定是否对该数据包进行状态跟踪

mangle表：为数据包设置标记

nat表：修改数据包中的源、目标IP地址或端口

filter表：确定是否放行该数据包（过滤）

INPUT：处理入站数据包

OUTPUT：处理出站数据包

FORWARD：处理转发数据包

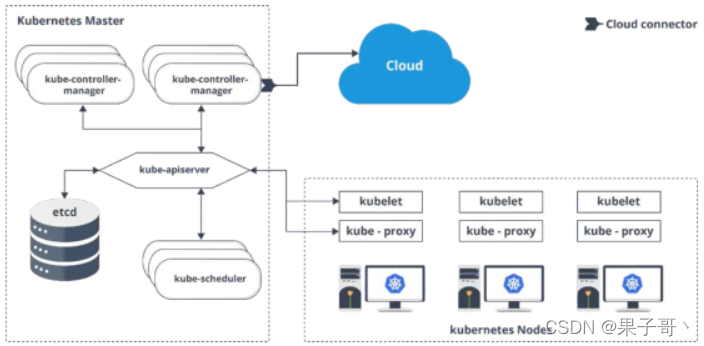
POSTROUTING链：在进行路由选择后处理数据包

PREROUTING链：在进行路由选择前处理数据包

**61、K8s体系结构有哪些不同的组成部分？**

答：Kubernetes架构主要包含两个组件-主节点和工作节点。如下图所示，主节点和工作节点中有许多内置组件。主节点具有kube-controller-manager，kube-

apiserver，kube-scheduler等。而工作程序节点在每个节点上都运行kubelet和kube-proxy。



kube-apiserver和kube-scheduler的作用是什么？

答：

1. kube-apiserver遵循横向扩展架构，并且是主节点控制面板的前端。这将公开Kubernetes主节点组件的所有API，并负责在Kubernetes节点和Kubernetes主组件之间建立通信。
2. kube-scheduler调度程序负责在工作节点上分配和管理工作负载。因此，它根据资源需求选择最合适的节点来运行计划外的Pod，并跟踪资源利用率。它可以确保未在已满的节点上调度工作负载。

**62、副本集和复制控制器之间又什么区别？**

答：副本集和复制控制器执行几乎相同的操作。它们两者都确保在任何给定时间都运行指定数量的Pod副本。不同之处在于使用选择器来复制容器。副本集使用基于集合的选择器，而复制控制器使用基于权益的选择器。

1. 基于股权的选择器：这种类型的选择器允许按标签键和值进行过滤。因此，以通俗易懂的术语来说，基于权益的选择器将仅查找具有与标签词组完全相同的词组的豆荚。

示例：假设您的标签键为app=nginx，那么使用此选择器，您只能查找标签为app等于nginx的吊舱。

1. 基于选择器的选择器：这种类型的选择器允许根据一组值过滤键。因此，换句话说，基于选择器的选择器将查找其标签已在集合中提及的Pod。

示例：说您的标签密钥说（nginx，NPS，Apache）中为app。然后，使用此选择器，如果您的应用等于nginx，NPS或Apache中的任何一个，则选择器会将其视为真实结果。

**场景题**

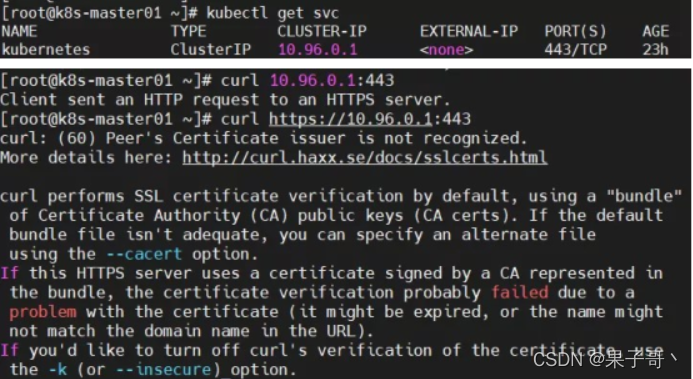
1、假设一家基于整体架构的公司处理许多产品。现在，随着公司在当今规模化行业中的发展，其整体架构开始引起问题。您如何看待公司从单一服务转向微服务并部署其服务容器？

答：由于该公司的目标是从单一应用程序转变为微服务，因此它们最终可以一步一步地并行构建，而只需在后台切换配置即可。然后，他们可以将每个内置微服务放在Kubernetes平台上。因此，他们可以从迁移服务一次或两次并监视它们以确保一切运行稳定开始。一旦他们感觉一切顺利，就可以将应用程序的其余部分迁移到其Kubernetes集群中。

2、K8s集群服务访问失败？

原因分析：证书不能被识别，其原因：自定义证书，过期等。

解决办法：更新证书



3、K8s集群服务访问失败

curl:（7）Failed connect to 10.103.22.158：3000；Connection refused

原因分析：端口映射错误，服务正常工作，但不能提供服务。

解决方法：删除svc，重新映射端口即可。

kubectt delete svc nginx-deployment

4、K8s集群服务暴露失败？

Error from server（AlreadyExists）：services"nginx-deployment"already exists

原因分析：该容器已暴露服务了。

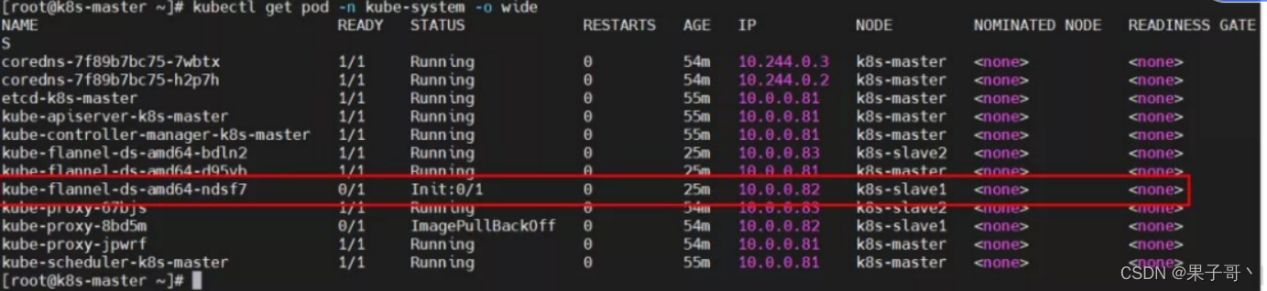
解决方法：删除svc，重新映射端口即可。

5、外网无法访问K8s集群提供的服务？

1. 原因分析：K8S集群的type为ClusterlP，未将服务暴露至外网。
2. 解决方法：修改K85集群的type为NodePort即可，于是可通过所有K8S集群节点访问服务。

kubectt edit svc nginx-deployment

6、kube-flannel-ds-amd64-ndsf7插件pod的status为init:0/1？



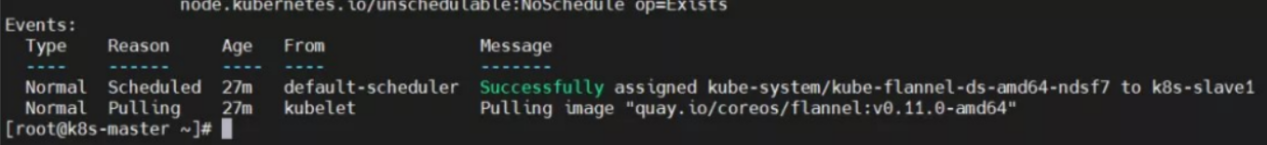
排查思路：kubectl -n kube-system describe pod kube-flannel-ds-amd64-ndfs7

查询pod描述信息

原因分析：k8s-slave1节点拉取镜像失败。

解决办法：登录k8s-slave1，重启docker服务，手动拉取镜像

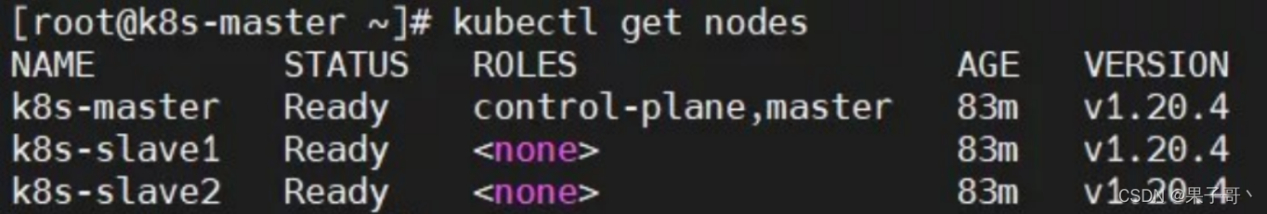




k8s-master节点，重新安装插件即可。

kubectl create -f kube-flannel.yml

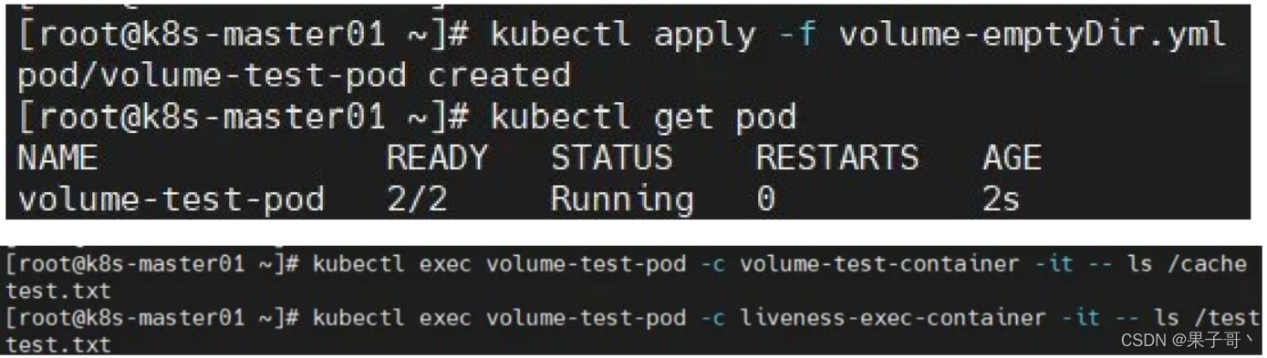
kubectl get nodes



7、不能进入指定容器内部？

原因分析：yml文件comtainers字段重复，导致该pod 没有该容器。

解决方法：去掉yml文件中多余的containers字段，重新生成pod。



stickPicture.png

8、Pod无法挂载PVC？

accessModes与可使用的PV不一致，导致无法挂载PVC，由于只能挂载大于1G且accessModes为RWO的PV，故只能成功创建1个pod，第2个pod一致pending，按序创建时则第3个pod一直未被创建；解决方法：修改yml文件中accessModes或PV的accessModes即可。

