k8s面试题

资料整理自网络，仅作免费交流分享，侵权删！

更多面试资料请关注微信公众号：程序员面试吧

1、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。

2、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。

3、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。

4、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。

5、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。

6、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

7、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。

8、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。

9、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。

10、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(Kubernetes 之 Pod 实现原理)。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(Kubernetes 之服务发现)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。

11、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。

12、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。

13、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。

14、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。

15、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。

16、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。

17、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。

18、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。

19、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。

20、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。

21、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。

22、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。

23、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。

24、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。

25、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。

26、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。

27、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。

28、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。

29、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。

30、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。

31、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。

32、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。

33、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。

34、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。

35、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。

36、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。

37、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。

38、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。

39、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。

40、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。

41、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。

42、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。

43、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。

44、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。

45、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。

46、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。

47、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。

48、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）。

49、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。

50、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。

51、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）。

52、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。

53、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。

54、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。

55、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。

56、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”。

57、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。

58、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。

59、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。

60、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。

61、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。

62、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。

63、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。

64、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。

65、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。

66、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。

67、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。

1-67道题来自：https://www.yuque.com/docs/share/d3dd1e8e-6828-4da7-9e30-6a4f45c6fa8e

68、 k8s是什么？请说出你的了解?

答：Kubenetes是一个针对容器应用，进行自动部署，弹性伸缩和管理的开源系统。主要功能是生产环境中的容器编排。

K8S是Google公司推出的，它来源于由Google公司内部使用了15年的Borg系统，集结了Borg的精华。

69、 K8s架构的组成是什么?

答：和大多数分布式系统一样，K8S集群至少需要一个主节点（Master）和多个计算节点（Node）。

- 主节点主要用于暴露API，调度部署和节点的管理；
- 计算节点运行一个容器运行环境，一般是docker环境（类似docker环境的还有rkt），同时运行一个K8s的代理（kubelet）用于和master通信。计算节点也会运行一些额外的组件，像记录日志，节点监控，服务发现等等。计算节点是k8s集群中真正工作的节点。

K8S架构细分：

1、Master节点（默认不参加实际工作）：

- Kubectl：客户端命令行工具，作为整个K8s集群的操作入口；
- Api Server：在K8s架构中承担的是“桥梁”的角色，作为资源操作的唯一入口，它提供了认证、授权、访问控制、API注册和发现等机制。客户端与k8s群集及K8s内部组件的通信，都要通过Api Server这个组件；
- Controller-manager：负责维护群集的状态，比如故障检测、自动扩展、滚动更新等；
- Scheduler：负责资源的调度，按照预定的调度策略将pod调度到相应的node节点上；
- Etcd：担任数据中心的角色，保存了整个群集的状态；

2、Node节点：

- Kubelet：负责维护容器的生命周期，同时也负责Volume和网络的管理，一般运行在所有的节点，是Node节点的代理，当Scheduler确定某个node上运行pod之后，会将pod的具体信息（image，volume）等发送给该节点的kubelet，kubelet根据这些信息创建和运行容器，并向master返回运行状态。（自动修复功能：如果某个节点中的容器宕机，它会尝试重启该容器，若重启无效，则会将该pod杀死，然后重新创建一个容器）；
- Kube-proxy：Service在逻辑上代表了后端的多个pod。负责为Service提供cluster内部的服务发现和负载均衡（外界通过Service访问pod提供的服务时，Service接收到的请求后就是通过kube-proxy来转发到pod上的）；
- container-runtime：是负责管理运行容器的软件，比如docker
- Pod：是k8s集群里面最小的单位。每个pod里边可以运行一个或多个container（容器），如果一个pod中有两个container，那么container的USR（用户）、MNT（挂载点）、PID（进程号）是相互隔离的，UTS（主机名和域名）、IPC（消息队列）、NET（网络栈）是相互共享的。我比较喜欢把pod来当做豌豆夹，而豌豆就是pod中的container；

69、 容器和主机部署应用的区别是什么?

答：容器的中心思想就是秒级启动；一次封装、到处运行；这是主机部署应用无法达到的效果，但同时也更应该注重容器的数据持久化问题。

另外，容器部署可以将各个服务进行隔离，互不影响，这也是容器的另一个核心概念。

70、请你说一下kubenetes针对pod资源对象的健康监测机制?

答：K8s中对于pod资源对象的健康状态检测，提供了三类probe（探针）来执行对pod的健康监测：

1） livenessProbe探针

可以根据用户自定义规则来判定pod是否健康，如果livenessProbe探针探测到容器不健康，则kubelet会根据其重启策略来决定是否重启，如果一个容器不包含livenessProbe探针，则kubelet会认为容器的livenessProbe探针的返回值永远成功。

2） ReadinessProbe探针

同样是可以根据用户自定义规则来判断pod是否健康，如果探测失败，控制器会将此pod从对应service的endpoint列表中移除，从此不再将任何请求调度到此Pod上，直到下次探测成功。

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，这个问题也可以换另一种方式解决，就是定义上面两类探针机制时，初始化时间定义的长一些即可。

每种探测方法能支持以下几个相同的检查参数，用于设置控制检查时间：

- initialDelaySeconds：初始第一次探测间隔，用于应用启动的时间，防止应用还没启动而健康检查失败
- periodSeconds：检查间隔，多久执行probe检查，默认为10s；
- timeoutSeconds：检查超时时长，探测应用timeout后为失败；
- successThreshold：成功探测阈值，表示探测多少次为健康正常，默认探测1次。

上面两种探针都支持以下三种探测方法：

1）Exec： 通过执行命令的方式来检查服务是否正常，比如使用cat命令查看pod中的某个重要配置文件是否存在，若存在，则表示pod健康。反之异常。

Exec探测方式的yaml文件语法如下：

```
spec:
  containers:
  - name: liveness
    image: k8s.gcr.io/busybox
    args:
    - /bin/sh
    - -c
    - touch /tmp/healthy; sleep 30; rm -rf /tmp/healthy; sleep 600
    livenessProbe:         #选择livenessProbe的探测机制
      exec:                       #执行以下命令
        command:
        - cat
        - /tmp/healthy
      initialDelaySeconds: 5           #在容器运行五秒后开始探测
      periodSeconds: 5                #每次探测的时间间隔为5秒
```

在上面的配置文件中，探测机制为在容器运行5秒后，每隔五秒探测一次，如果cat命令返回的值为“0”，则表示健康，如果为非0，则表示异常。

2）Httpget： 通过发送http/htps请求检查服务是否正常，返回的状态码为200-399则表示容器健康（注http get类似于命令curl -I）。

Httpget探测方式的yaml文件语法如下：

```
spec:
  containers:
  - name: liveness
    image: k8s.gcr.io/liveness
    livenessProbe:              #采用livenessProbe机制探测
      httpGet:                  #采用httpget的方式
    scheme:HTTP          #指定协议，也支持https
        path: /healthz          #检测是否可以访问到网页根目录下的healthz网页文件
        port: 8080              #监听端口是8080
      initialDelaySeconds: 3      #容器运行3秒后开始探测
      periodSeconds: 3                  #探测频率为3秒
```

上述配置文件中，探测方式为项容器发送HTTP GET请求，请求的是8080端口下的healthz文件，返回任何大于或等于200且小于400的状态码表示成功。任何其他代码表示异常。

3）tcpSocket： 通过容器的IP和Port执行TCP检查，如果能够建立TCP连接，则表明容器健康，这种方式与HTTPget的探测机制有些类似，tcpsocket健康检查适用于TCP业务。

tcpSocket探测方式的yaml文件语法如下：

```
spec:
  containers:
  - name: goproxy
    image: k8s.gcr.io/goproxy:0.1
    ports:
- containerPort: 8080
#这里两种探测机制都用上了，都是为了和容器的8080端口建立TCP连接
    readinessProbe:
      tcpSocket:
        port: 8080
      initialDelaySeconds: 5
      periodSeconds: 10
    livenessProbe:
      tcpSocket:
        port: 8080
      initialDelaySeconds: 15
      periodSeconds: 20
```

在上述的yaml配置文件中，两类探针都使用了，在容器启动5秒后，kubelet将发送第一个readinessProbe探针，这将连接容器的8080端口，如果探测成功，则该pod为健康，十秒后，kubelet将进行第二次连接。

除了readinessProbe探针外，在容器启动15秒后，kubelet将发送第一个livenessProbe探针，仍然尝试连接容器的8080端口，如果连接失败，则重启容器。

探针探测的结果无外乎以下三者之一：

- Success：Container通过了检查；
- Failure：Container没有通过检查；
- Unknown：没有执行检查，因此不采取任何措施（通常是我们没有定义探针检测，默认为成功）。

若觉得上面还不够透彻，可以移步其官网文档：

> https://kubernetes.io/docs/tasks/configure-pod-container/configure-liveness-readiness-startup-probes/

71、 如何控制滚动更新过程?

答：可以通过下面的命令查看到更新时可以控制的参数：

```
[root@master yaml]# kubectl explain deploy.spec.strategy.rollingUpdate
```

maxSurge： 此参数控制滚动更新过程，副本总数超过预期pod数量的上限。可以是百分比，也可以是具体的值。默认为1。

（上述参数的作用就是在更新过程中，值若为3，那么不管三七二一，先运行三个pod，用于替换旧的pod，以此类推）

maxUnavailable： 此参数控制滚动更新过程中，不可用的Pod的数量。

（这个值和上面的值没有任何关系，举个例子：我有十个pod，但是在更新的过程中，我允许这十个pod中最多有三个不可用，那么就将这个参数的值设置为3，在更新的过程中，只要不可用的pod数量小于或等于3，那么更新过程就不会停止）。

72、K8s中镜像的下载策略是什么?

答：可通过命令“kubectl explain pod.spec.containers”来查看imagePullPolicy这行的解释。

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。

73、 image的状态有哪些?

- Running：Pod所需的容器已经被成功调度到某个节点，且已经成功运行，
- Pending：APIserver创建了pod资源对象，并且已经存入etcd中，但它尚未被调度完成或者仍然处于仓库中下载镜像的过程
- Unknown：APIserver无法正常获取到pod对象的状态，通常是其无法与所在工作节点的kubelet通信所致。

74、 pod的重启策略是什么?

答：可以通过命令“kubectl explain pod.spec”查看pod的重启策略。（restartPolicy字段）

- Always：但凡pod对象终止就重启，此为默认策略。
- OnFailure：仅在pod对象出现错误时才重启

75、 Service这种资源对象的作用是什么?

答：用来给相同的多个pod对象提供一个固定的统一访问接口，常用于服务发现和服务访问。

76、版本回滚相关的命令?

```
[root@master httpd-web]# kubectl apply -f httpd2-deploy1.yaml  --record
#运行yaml文件，并记录版本信息；
[root@master httpd-web]# kubectl rollout history deployment httpd-devploy1
#查看该deployment的历史版本
[root@master httpd-web]# kubectl rollout undo deployment httpd-devploy1 --to-
revision=1
#执行回滚操作，指定回滚到版本1

#在yaml文件的spec字段中，可以写以下选项（用于限制最多记录多少个历史版本）：
spec:
  revisionHistoryLimit: 5
#这个字段通过 kubectl explain deploy.spec  命令找到revisionHistoryLimit   <integer>
行获得
```

77、 标签与标签选择器的作用是什么?

标签：是当相同类型的资源对象越来越多的时候，为了更好的管理，可以按照标签将其分为一个组，为的是提升资源对象的管理效率。

标签选择器：就是标签的查询过滤条件。目前API支持两种标签选择器：

- 基于等值关系的，如："="、""=="、"！="（注："=="也是等于的意思，yaml文件中的matchLabels字段）；
- 基于集合的，如：in、notin、exists（yaml文件中的matchExpressions字段）；

> 注：in:在这个集合中；notin：不在这个集合中；exists：要么全在（exists）这个集合中，要么都不在（notexists）；

使用标签选择器的操作逻辑：

- 在使用基于集合的标签选择器同时指定多个选择器之间的逻辑关系为"与"操作（比如：- {key: name,operator: In,values: [zhangsan,lisi]}，那么只要拥有这两个值的资源，都会被选中）；
- 使用空值的标签选择器，意味着每个资源对象都被选中（如：标签选择器的键是"A"，两个资源对象同时拥有A这个键，但是值不一样，这种情况下，如果使用空值的标签选择器，那么将同时选中这两个资源对象）
- 空的标签选择器（注意不是上面说的空值，而是空的，都没有定义键的名称），将无法选择出任何资源；

在基于集合的选择器中，使用"In"或者"Notin"操作时，其values可以为空，但是如果为空，这个标签选择器，就没有任何意义了。

两种标签选择器类型（基于等值、基于集合的书写方法）：

```
selector:
  matchLabels:           #基于等值
    app: nginx
  matchExpressions:          #基于集合
    - {key: name,operator: In,values: [zhangsan,lisi]}     #key、operator、values这三个字段是固定的
    - {key: age,operator: Exists,values:}   #如果指定为exists，那么values的值一定要为空
```

78、 常用的标签分类有哪些?

标签分类是可以自定义的，但是为了能使他人可以达到一目了然的效果，一般会使用以下一些分类：

- 版本类标签（release）：stable（稳定版）、canary（金丝雀版本，可以将其称之为测试版中的测试版）、beta（测试版）；
- 环境类标签（environment）：dev（开发）、qa（测试）、production（生产）、op（运维）；
- 应用类（app）：ui、as、pc、sc;
- 架构类（tier）：frontend（前端）、backend（后端）、cache（缓存）；
- 分区标签（partition）：customerA（客户A）、customerB（客户B）；
- 品控级别（Track）：daily（每天）、weekly（每周）。

79、 有几种查看标签的方式?

答：常用的有以下三种查看方式：

```
[root@master ~]# kubectl get pod --show-labels     #查看pod，并且显示标签内容
[root@master ~]# kubectl get pod -L env,tier       #显示资源对象标签的值
[root@master ~]# kubectl get pod -l env,tier       #只显示符合键值资源对象的pod，而"-L"是显示所有的pod
```

80、 添加、修改、删除标签的命令?

```
#对pod标签的操作
[root@master ~]# kubectl label pod label-pod abc=123     #给名为label-pod的pod添加标签
[root@master ~]# kubectl label pod label-pod abc=456 --overwrite       #修改名为label-pod的标签
[root@master ~]# kubectl label pod label-pod abc-              #删除名为label-pod的标签
[root@master ~]# kubectl get pod --show-labels

#对node节点的标签操作
[root@master ~]# kubectl label nodes node01 disk=ssd      #给节点node01添加disk标签

[root@master ~]# kubectl label nodes node01 disk=sss -overwrite    #修改节点node01的标签
[root@master ~]# kubectl label nodes node01 disk-          #删除节点node01的disk标签
```

81、 DaemonSet资源对象的特性?

DaemonSet这种资源对象会在每个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。所以，在其yaml文件中，不支持定义replicas，除此之外，与Deployment、RS等资源对象的写法相同。

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；

82、 说说你对Job这种资源对象的了解?

答：Job与其他服务类容器不同，Job是一种工作类容器（一般用于做一次性任务）。使用常见不多，可以忽略这个问题。

```
#提高Job执行效率的方法：
spec:
  parallelism: 2           #一次运行2个
  completions: 8           #最多运行8个
  template:
metadata:
```

83、描述一下pod的生命周期有哪些状态?

- Pending：表示pod已经被同意创建，正在等待kube-scheduler选择合适的节点创建，一般是在准备镜像；
- Running：表示pod中所有的容器已经被创建，并且至少有一个容器正在运行或者是正在启动或者是正在重启；
- Succeeded：表示所有容器已经成功终止，并且不会再启动；
- Failed：表示pod中所有容器都是非0（不正常）状态退出；
- Unknown：表示无法读取Pod状态，通常是kube-controller-manager无法与Pod通信。

84、 创建一个pod的流程是什么?

- 客户端提交Pod的配置信息（可以是yaml文件定义好的信息）到kube-apiserver；
- Apiserver收到指令后，通知给controller-manager创建一个资源对象；
- Controller-manager通过api-server将pod的配置信息存储到ETCD数据中心中；

- 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。
- 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。

85、 删除一个Pod会发生什么事情?

答：Kube-apiserver会接受到用户的删除指令，默认有30秒时间等待优雅退出，超过30秒会被标记为死亡状态，此时Pod的状态Terminating，kubelet看到pod标记为Terminating就开始了关闭Pod的工作；

关闭流程如下：

- pod从service的endpoint列表中被移除；
- 如果该pod定义了一个停止前的钩子，其会在pod内部被调用，停止钩子一般定义了如何优雅的结束进程；
- 进程被发送TERM信号（kill -14）
- 当超过优雅退出的时间后，Pod中的所有进程都会被发送SIGKILL信号（kill -9）。

86、 K8s的Service是什么?

答：Pod每次重启或者重新部署，其IP地址都会产生变化，这使得pod间通信和pod与外部通信变得困难，这时候，就需要Service为pod提供一个固定的入口。

Service的Endpoint列表通常绑定了一组相同配置的pod，通过负载均衡的方式把外界请求分配到多个pod上

87、 k8s是怎么进行服务注册的?

答：Pod启动后会加载当前环境所有Service信息，以便不同Pod根据Service名进行通信。

88、 k8s集群外流量怎么访问Pod?

答：可以通过Service的NodePort方式访问，会在所有节点监听同一个端口，比如：30000，访问节点的流量会被重定向到对应的Service上面。

89、 k8s数据持久化的方式有哪些?

答：

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。

主要使用场景：

- 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，使得第一个内容管理的容器可以将生成的数据存入其中，同时由同一个webserver容器对外提供这些页面。

emptyDir的特性：

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。如果仅仅是容器被销毁，pod还在，则不会影响volume中的数据。

总结来说：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。

2）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。

这种数据持久化方式，运用场景不多，因为它增加了pod与节点之间的耦合。

一般对于k8s集群本身的数据持久化和docker本身的数据持久化会使用这种方式，可以自行参考apiService的yaml文件，位于：/etc/kubernetes/main...目录下。

3）PersistentVolume（简称PV）：

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。

在一个PV的yaml文件中，可以对其配置PV的大小，指定PV的访问模式：

- ReadWriteOnce：只能以读写的方式挂载到单个节点；
- ReadOnlyMany：能以只读的方式挂载到多个节点；
- ReadWriteMany：能以读写的方式挂载到多个节点。以及指定pv的回收策略：
- recycle：清除PV的数据，然后自动回收；
- Retain：需要手动回收；
- delete：删除云存储资源，云存储专用；

> PS：这里的回收策略指的是在PV被删除后，在这个PV下所存储的源文件是否删除）。

若需使用PV，那么还有一个重要的概念：PVC，PVC是向PV申请应用所需的容量大小，K8s集群中可能会有多个PV，PVC和PV若要关联，其定义的访问模式必须一致。定义的storageClassName也必须一致，若群集中存在相同的（名字、访问模式都一致）两个PV，那么PVC会选择向它所需容量接近的PV去申请，或者随机申请。

68-89题来自：kubernetes面试题汇总_lvjianzhaoa的博客-CSDN博客

**
一、基本的Kubernetes面试问题**

这部分问题将包含您需要了解的与Kubernetes工作相关的所有基本问题。

90、Kubernetes与Docker Swarm的区别如何?

Features	Kubernetes	Docker Swarm
Installation & Cluster Configuration	Installation is complicated; but once setup, the cluster is very strong	Installation is very simple; but cluster is not very strong
GUI	GUI is the Kubernetes Dashboard	There is no GUI
Scalability	Highly scalable & scales fast	Highly scalable & scales 5x faster than Kubernetes
Auto-Scaling	Kubernetes can do auto-scaling	Docker Swarm cannot do auto-scaling
Load Balancing	Manual intervention needed for load balancing traffic between different containers in different Pods	Docker Swarm does auto load balancing of traffic between containers in the cluster
Rolling Updates & Rollbacks	Can deploy Rolling updates & does automatic Rollbacks	Can deploy Rolling updates, but not automatic Rollbacks
Data Volumes	Can share storage volumes only with other containers in same Pod	Can share storage volumes with any other container
Logging & Monitoring	In-built tools for logging & monitoring	3rd party tools like ELK should be used for logging & monitoring

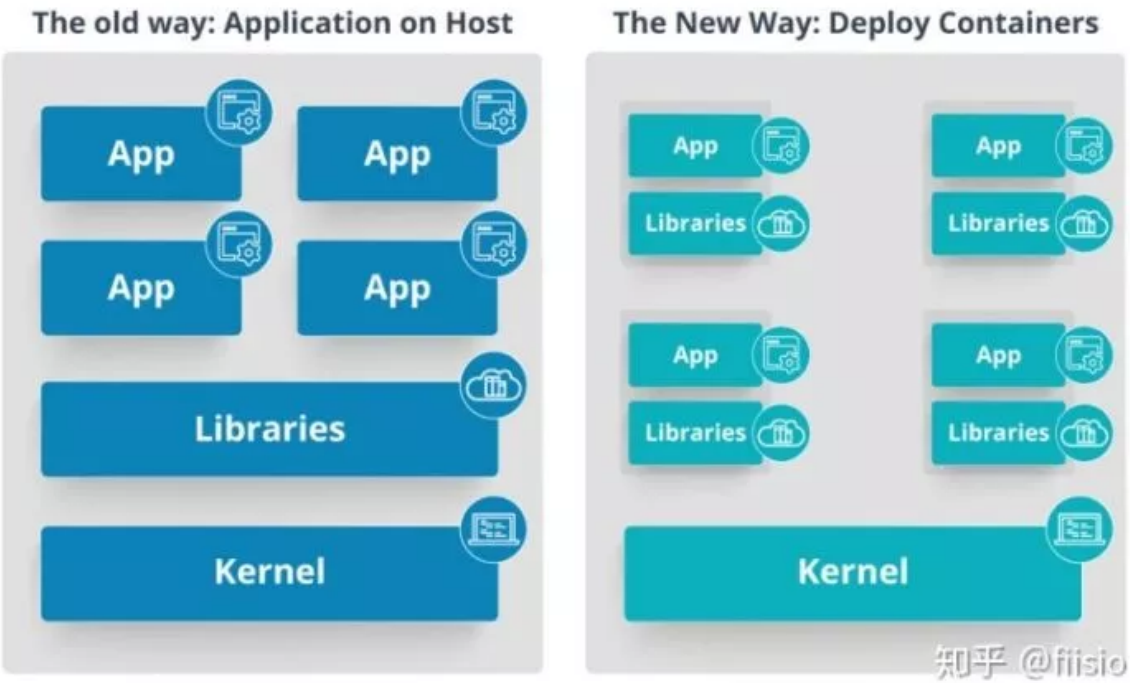
91、什么是Kubernetes?

Kubernetes是一个开源容器管理工具，负责容器部署，容器扩缩容以及负载平衡。作为Google的创意之作，它提供了出色的社区，并与所有云提供商合作。因此，我们可以说Kubernetes不是*一个容器化平台，而是一个多容器管理解决方案。*

92、Kubernetes与Docker有什么关系?

众所周知，Docker提供容器的生命周期管理，Docker镜像构建运行时容器。但是，由于这些单独的容器必须通信，因此使用Kubernetes。因此，我们说Docker构建容器，这些容器通过Kubernetes相互通信。因此，可以使用Kubernetes手动关联和编排在多个主机上运行的容器。

93、在主机和容器上部署应用程序有什么区别?



请参考上图。左侧架构表示在主机上部署应用程序。因此，这种架构将具有操作系统，然后操作系统将具有内核，该内核将在应用程序所需的操作系统上安装各种库。因此，在这种框架中，您可以拥有n个应用程序，并且所有应用程序将共享该操作系统中存在的库，而在容器中部署应用程序时，体系结构则略有不同。

这种架构将有一个内核，这是唯一一个在所有应用程序之间唯一共同的东西。因此，如果有一个需要Java的特定应用程序，那么我们将获得访问Java的特定应用程序，如果有另一个需要Python的应用程序，则只有该特定应用程序才能访问Python。

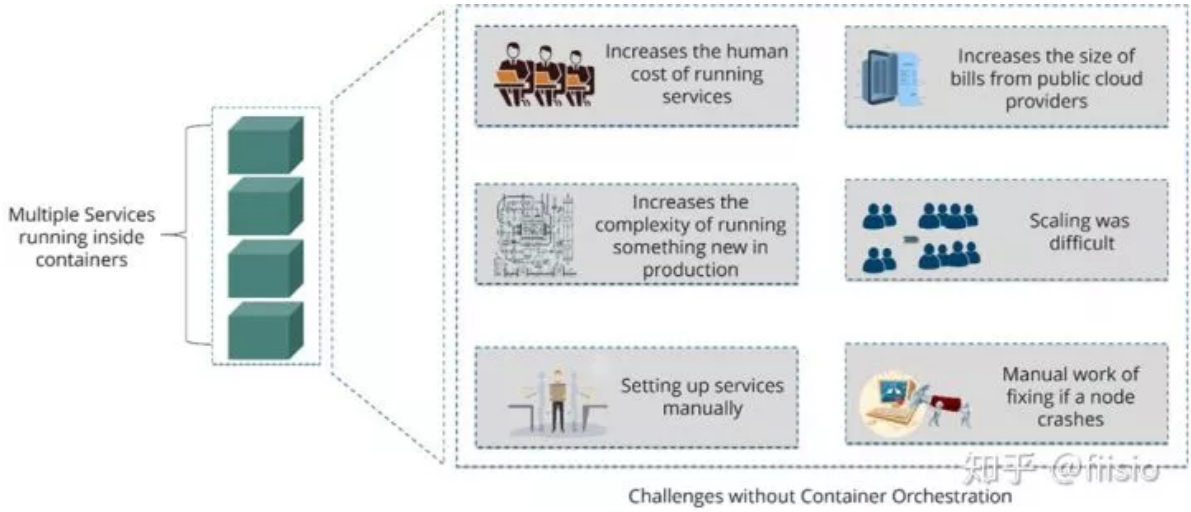
您可以在图表右侧看到的各个块基本上是容器化的，并且这些块与其他应用程序隔离。因此，应用程序具有与系统其余部分隔离的必要库和二进制文件，并且不能被任何其他应用程序侵占。

94、什么是Container Orchestration?

考虑一个应用程序有5-6个微服务的场景。现在，这些微服务被放在单独的容器中，但如果没有容器编排就无法进行通信。因此，由于编排意味着所有乐器在音乐中和谐共处，所以类似的容器编排意味着各个容器中的所有服务协同工作以满足单个服务器的需求。

95、Container Orchestration需要什么?

考虑到你有5-6个微服务用于执行各种任务的单个应用程序，所有这些微服务都放在容器中。现在，为了确保这些容器彼此通信，我们需要容器编排。



正如您在上图中所看到的，在没有使用容器编排的情况下，还存在许多挑战。因此，为了克服这些挑战，容器编排就到位了。

96、Kubernetes有什么特点?

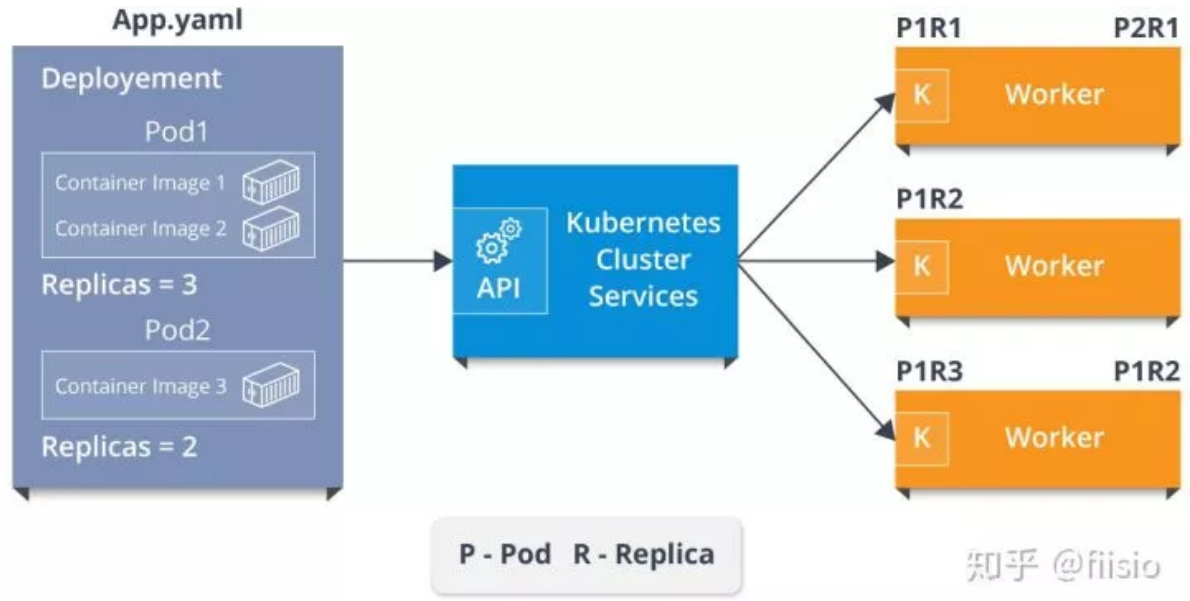
Kubernetes的功能如下：

97、Kubernetes如何简化容器化部署?

由于典型应用程序将具有跨多个主机运行的容器集群，因此所有这些容器都需要相互通信。因此，要做到这一点，你需要一些能够负载平衡，扩展和监控容器的东西。由于Kubernetes与云无关并且可以在任何公共/私有提供商上运行，因此必须是您简化容器化部署的选择。

98、对Kubernetes的集群了解多少?

Kubernetes背后的基础是我们可以实施所需的状态管理，我的意思是我们可以提供特定配置的集群服务，并且集群服务将在基础架构中运行并运行该配置。



因此，正如您在上图中所看到的，部署文件将具有提供给集群服务所需的所有配置。现在，部署文件将被提供给API，然后由集群服务决定如何在环境中安排这些pod，并确保正确运行的pod数量。

因此，位于服务前面的API，工作节点和节点运行的Kubelet进程，共同构成了Kubernetes集群。

99、什么是Google容器引擎?

Google Container Engine（GKE）是Docker容器和集群的开源管理平台。这个基于Kubernetes的引擎仅支持在Google的公共云服务中运行的群集。

100、什么是Heapster?

Heapster是由每个节点上运行的Kubelet提供的集群范围的数据聚合器。此容器管理工具在Kubernetes集群上本机支持，并作为pod运行，就像集群中的任何其他pod一样。因此，它基本上发现集群中的所有节点，并通过机上Kubernetes代理查询集群中Kubernetes节点的使用信息。

101、什么是Minikube?

Minikube是一种工具，可以在本地轻松运行Kubernetes。这将在虚拟机中运行单节点Kubernetes群集。

102、什么是Kubectl?

Kubectl是一个平台，您可以使用该平台将命令传递给集群。因此，它基本上为CLI提供了针对Kubernetes集群运行命令的方法，以及创建和管理Kubernetes组件的各种方法。

103、什么是Kubelet?

这是一个代理服务，它在每个节点上运行，并使从服务器与主服务器通信。因此，Kubelet处理PodSpec中提供给它的容器的描述，并确保PodSpec中描述的容器运行正常。

二、基于架构的Kubernetes访谈问题

这部分问题将涉及与Kubernetes架构相关的问题。

104、Kubernetes Architecture的不同组件有哪些?

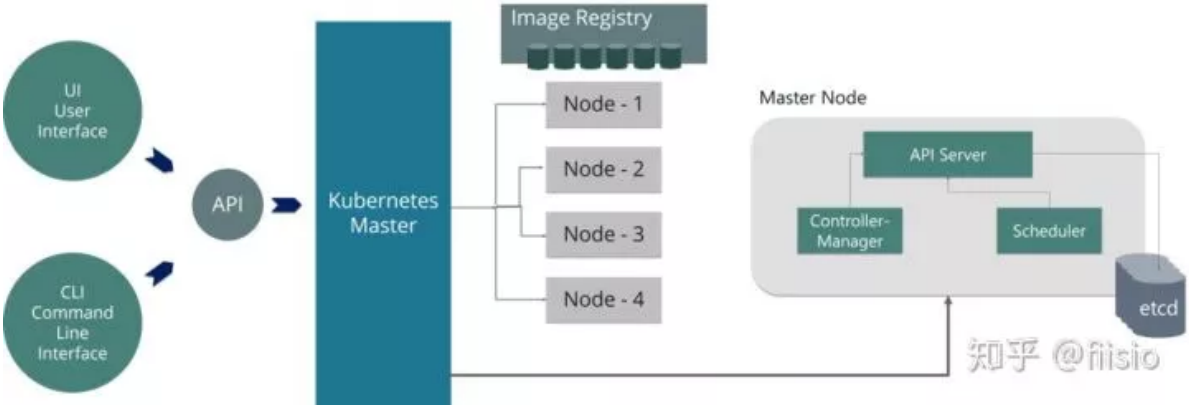
Kubernetes Architecture主要有两个组件 - 主节点和工作节点。如下图所示，master和worker节点中包含许多内置组件。主节点具有kube-controller-manager，kube-apiserver，kube-scheduler等。而工作节点具有在每个节点上运行的kubelet和kube-proxy。

105、你对Kube-proxy有什么了解?

Kube-proxy可以在每个节点上运行，并且可以跨后端网络服务进行简单的TCP / UDP数据包转发。基本上，它是一个网络代理，它反映了每个节点上Kubernetes API中配置的服务。因此，Docker可链接的兼容环境变量提供由代理打开的群集IP和端口。

106、能否介绍一下Kubernetes中主节点的工作情况?

Kubernetes master控制容器存在的节点和节点内部。现在，这些单独的容器包含在容器内部和每个容器内部，您可以根据配置和要求拥有不同数量的容器。因此，如果必须部署pod，则可以使用用户界面或命令行界面部署它们。然后，在节点上调度这些pod，并根据资源需求，将pod分配给这些节点。kube-apiserver确保在Kubernetes节点和主组件之间建立通信。



107、kube-apiserver和kube-scheduler的作用是什么?

kube -apiserver遵循横向扩展架构，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。

kube-scheduler负责工作节点上工作负载的分配和管理。因此，它根据资源需求选择最合适的节点来运行未调度的pod，并跟踪资源利用率。它确保不在已满的节点上调度工作负载。

108、你能简要介绍一下Kubernetes控制管理器吗?

多个控制器进程在主节点上运行，但是一起编译为单个进程运行，即Kubernetes控制器管理器。因此，Controller Manager是一个嵌入控制器并执行命名空间创建和垃圾收集的守护程序。它拥有责任并与API服务器通信以管理端点。

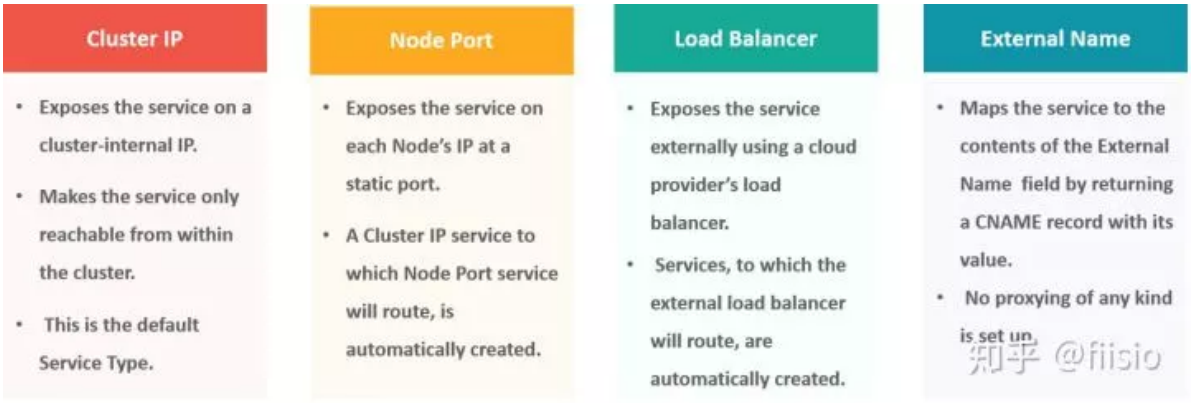
因此，主节点上运行的不同类型的控制器管理器是：

109、什么是ETCD?

Etcd是用Go编程语言编写的，是一个分布式键值存储，用于协调分布式工作。因此，Etcd存储Kubernetes集群的配置数据，表示在任何给定时间点的集群状态。

110、Kubernetes有哪些不同类型的服务?

以下是使用的不同类型的服务：



111、你对Kubernetes的负载均衡器有什么了解?

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。

112、什么是Ingress网络，它是如何工作的?

Ingress网络是一组规则，充当Kubernetes集群的入口点。这允许入站连接，可以将其配置为通过可访问的URL，负载平衡流量或通过提供基于名称的虚拟主机从外部提供服务。因此，Ingress是一个API对象，通常通过HTTP管理集群中服务的外部访问，是暴露服务的最有效方式。

现在，让我以一个例子向您解释Ingress网络的工作。

有2个节点具有带有Linux桥接器的pod和根网络命名空间。除此之外，还有一个名为flannel0（网络插件）的新虚拟以太网设备被添加到根网络中。

现在，假设我们希望数据包从pod1流向pod 4.请参阅下图。

- 因此，数据包将pod1的网络保留在eth0，并进入veth0的根网络。
- 然后它被传递给cbr0，这使得ARP请求找到目的地，并且发现该节点上没有人具有目的地IP地址。
- 因此，桥接器将数据包发送到flannel0，因为节点的路由表配置了flannel0。
- 现在，flannel守护程序与Kubernetes的API服务器通信，以了解所有pod IP及其各自的节点，以创建pods IP到节点IP的映射。
- 网络插件将此数据包封装在UDP数据包中，其中额外的标头将源和目标IP更改为各自的节点，并通过eth0发送此数据包。
- 现在，由于路由表已经知道如何在节点之间路由流量，因此它将数据包发送到目标节点2。
- 数据包到达node2的eth0并返回到flannel0以解封装并在根网络命名空间中将其发回。
- 同样，数据包被转发到Linux网桥以发出ARP请求以找出属于veth1的IP。
- 数据包最终穿过根网络并到达目标Pod4。

113、您对云控制器管理器有何了解?

Cloud Controller Manager负责持久存储，网络路由，从核心Kubernetes特定代码中抽象出特定于云的代码，以及管理与底层云服务的通信。它可能会分成几个不同的容器，具体取决于您运行的是哪个云平台，然后它可以使云供应商和Kubernetes代码在没有任何相互依赖的情况下开发。因此，云供应商开发他们的代码并在运行Kubernetes时与Kubernetes云控制器管理器连接。

各种类型的云控制器管理器如下：

114、什么是Container资源监控?

对于用户而言，了解应用程序的性能和所有不同抽象层的资源利用率非常重要，Kubernetes通过在容器，pod，服务和整个集群等不同级别创建抽象来考虑集群的管理。现在，可以监视每个级别，这只是容器资源监视。

各种容器资源监控工具如下：

115、Replica Set 和 Replication Controller之间有什么区别?

Replica Set 和 Replication Controller几乎完全相同。它们都确保在任何给定时间运行指定数量的pod副本。不同之处在于复制pod使用的选择器。Replica Set使用基于集合的选择器，而Replication Controller使用基于权限的选择器。

- Equity-Based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，在外行术语中，基于Equity的选择器将仅查找与标签具有完全相同短语的pod。 示例：假设您的标签键表示app = nginx，那么，使用此选择器，您只能查找标签应用程序等于nginx的那些pod。
- Selector-Based选择器：此类型的选择器允许根据一组值过滤键。因此，换句话说，基于Selector的选择器将查找已在集合中提及其标签的pod。 示例：假设您的标签键在（nginx，NPS，Apache）中显示应用程序。然后，使用此选择器，如果您的应用程序等于任何nginx，NPS或Apache，则选择器将其视为真实结果。

116、什么是Headless Service?

Headless Service类似于“普通”服务，但没有群集IP。此服务使您可以直接访问pod，而无需通过代理访问它。

117、使用Kubernetes时可以采取哪些最佳安全措施?

以下是使用Kubernetes时可以遵循的最佳安全措施：

118、什么是集群联邦?

在联邦集群的帮助下，可以将多个Kubernetes集群作为单个集群进行管理。因此，您可以在数据中心/云中创建多个Kubernetes集群，并使用联邦来在一个位置控制/管理它们。

联合集群可以通过执行以下两项操作来实现此目的。请参考下图。

三、基于场景的面试问题

这部分问题将包含您在面试中可能遇到的各种基于场景的问题。

119、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?

假设一家基于单一架构的公司处理众多产品。现在，随着公司在当今的扩展行业的扩展，他们的单一架构开始引发问题。

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?

解：

由于公司的目标是从单一应用程序转向微服务，它们最终可以逐个构建，并行构建，只需在后台切换配置。然后他们可以将这些内置微服务放在Kubernetes平台上。因此，他们可以从一次或两次迁移服务开始，并监控它们以确保一切运行稳定。一旦他们觉得一切顺利，他们就可以将其余的应用程序迁移到他们的Kubernetes集群中。

120、考虑一家拥有分布式系统的跨国公司，拥有大量数据中心，虚拟机和许多从事各种任务的员工。*您认为这样* 公司如何以与Kubernetes一致的方式管理所有任务?

考虑一家拥有分布式系统的跨国公司，拥有大量数据中心，虚拟机和许多从事各种任务的员工。

您认为这样 的公司如何以与Kubernetes一致的方式管理所有任务?

解：

正如我们所有人都知道IT部门推出了数千个容器，其任务在分布式系统中遍布全球众多节点。

在这种情况下，公司可以使用能够为基于云的应用程序提供敏捷性，横向扩展功能和DevOps实践的东西。

因此，该公司可以使用Kubernetes来定制他们的调度架构并支持多种容器格式。这使得容器任务之间的亲和性成为可能，从而提供更高的效率，并为各种容器网络解决方案和容器存储提供广泛支持。

121、考虑一种情况，即公司希望通过维持最低成本来提高其效率和技术运营速度。*您认为公司将如何实现这一目标?*

考虑一种情况，即公司希望通过维持最低成本来提高其效率和技术运营速度。

您认为公司将如何实现这一目标?

解：

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。

122、假设一家公司想要修改它的部署方法，并希望建立一个更具可扩展性和响应性的平台。*您如何看待这家公司能够实现这一目标以满足客户需求?*

假设一家公司想要修改它的部署方法，并希望建立一个更具可扩展性和响应性的平台。

您如何看待这家公司能够实现这一目标以满足客户需求?

解：

为了给数百万客户提供他们期望的数字体验，公司需要一个可扩展且响应迅速的平台，以便他们能够快速地将数据发送到客户网站。现在，要做到这一点，公司应该从他们的私有数据中心（如果他们使用任何）转移到任何云环境，如AWS。不仅如此，他们还应该实现微服务架构，以便他们可以开始使用Docker容器。一旦他们准备好基础框架，他们就可以开始使用最好的编排平台，即Kubernetes。这将使团队能够自主地构建应用程序并快速交付它们。

123、考虑一家拥有非常分散的系统的跨国公司，期待解决整体代码库问题。您认为公司如何解决他们的问题?

考虑一家拥有非常分散的系统的跨国公司，期待解决整体代码库问题。

您认为公司如何解决他们的问题?

解

那么，为了解决这个问题，我们可以将他们的单片代码库转移到微服务设计，然后每个微服务都可以被视为一个容器。因此，所有这些容器都可以在Kubernetes的帮助下进行部署和协调。

124、我们所有人都知道，从单片到微服务的转变解决了开发方面的问题，但却增加了部署方面的问题。公司如何解决部署方面的问题?

我们所有人都知道，从单片到微服务的转变解决了开发方面的问题，但却增加了部署方面的问题。

公司如何解决部署方面的问题?

解

团队可以试验容器编排平台，例如Kubernetes，并在数据中心运行。因此，通过这种方式，公司可以生成模板化应用程序，在五分钟内部署它，并在此时将实际实例集中在暂存环境中。这种Kubernetes项目将有数十个并行运行的微服务，以提高生产率，即使节点出现故障，也可以立即重新安排，而不会影响性能。

125、公司如何有效地实现这种资源分配?

假设一家公司希望通过采用新技术来优化其工作负载的分配。

公司如何有效地实现这种资源分配?

解

这个问题的解决方案就是Kubernetes。Kubernetes确保资源得到有效优化，并且只使用特定应用程序所需的那些资源。因此，通过使用最佳容器编排工具，公司可以有效地实现资源分配。

126、您认为公司如何处理服务器及其安装?

考虑一家拼车公司希望通过同时扩展其平台来增加服务器数量。

您认为公司如何处理服务器及其安装?

解

公司可以采用集装箱化的概念。一旦他们将所有应用程序部署到容器中，他们就可以使用Kubernetes进行编排，并使用像Prometheus这样的容器监视工具来监视容器中的操作。因此，利用容器的这种使用，在数据中心中为它们提供更好的容量规划，因为它们现在将受到更少的限制，因为服务和它们运行的硬件之间存在抽象。

127、考虑一种情况，公司希望向具有各种环境的客户提供所有必需的分发。您认为他们如何以动态的方式实现这一关键目标?

考虑一种情况，公司希望向具有各种环境的客户提供所有必需的分发。

您认为他们如何以动态的方式实现这一关键目标?

解

该公司可以使用Docker环境，组建一个横截面团队，使用Kubernetes构建Web应用程序。这种框架将帮助公司实现在最短的时间内将所需产品投入生产的目标。因此，在这样的机器运行的情况下，公司可以向所有具有各种环境的客户发放电子邮件。

128、假设公司希望在不同的云基础架构上运行各种工作负载，从裸机到公共云。公司将如何在不同界面的存在下实现这一目标?

假设公司希望在不同的云基础架构上运行各种工作负载，从裸机到公共云。

公司将如何在不同界面的存在下实现这一目标?

解

该公司可以将其基础设施分解为微服务，然后采用Kubernetes。这将使公司在不同的云基础架构上运行各种工作负载。

90-128题来自：https://zhuanlan.zhihu.com/p/74560934