Kubernetes 2022 年最新常见面试题汇总

目录

常规题

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：

- 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?

答：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?

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。

4. 简述 Kubernetes 和 Docker 的关系?

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?

答：
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?

答：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。

7. 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。

8. 简述 Kubernetes 的优势、适应场景及其特点?

答：Kubernetes 作为一个完备的分布式系统支撑平台，其主要优势：

- 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?

答：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?

答：

- 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。
- Replica Set（副本集）： Pod 只是单个应用实例的抽象，要构建高可用应用，通常需要构建多个同样的副本，提供同一个服务。Kubernetes 为此抽象出副本集 Replica Set，其允许用户定义 Pod 的副本数，每一个 Pod 都会被当作一个无状态的成员进行管理，Kubernetes 保证总是有用户期望的数量的 Pod 正常运行。当某个副本宕机以后，控制器将会创建一个新的副本。当因业务负载发生变更而需要调整扩缩容时，可以方便地调整副本数量。
- 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。

11.简述 Kubernetes 集群相关组件？

答：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？

答：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。

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

答：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。

14.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。

15.简述 kube-proxy iptables 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。

16.简述 kube-proxy ipvs 原理?

答：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。

17.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。

18.简述 Kubernetes 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。

19.简述 Kubernetes 中 Pod 可能位于的状态?

答：

- 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。

20.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、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。

21.简述 Kubernetes 中 Pod 的重启策略?

答：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。

22.简述 Kubernetes 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。

- 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。

23.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：

- 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。

24.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：

- 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。

25.简述 Kubernetes 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，Kubernetes 才会初始化 Pod 的各种信息，并开始创建和运行应用容器。

26.简述 Kubernetes deployment 升级过程?

答：

- 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。

27.简述 Kubernetes deployment 升级策略?

答：

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。

28.简述 Kubernetes DaemonSet 类型的资源特性?

答：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。

29.简述 Kubernetes 自动扩容机制?

答：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。

30.简述 Kubernetes 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：

- 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。

31.简述 Kubernetes Service 分发后端的策略?

答：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。

32.简述 Kubernetes 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 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。

33.简述 Kubernetes 外部如何访问集群内的服务?

答：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。

34.简述 Kubernetes 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。

35.简述 Kubernetes 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。

36.简述 Kubernetes 的负载均衡器?

答：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。

37.简述 Kubernetes 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。

38.简述 Kubernetes 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（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。

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

答：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。

40.简述 Kubernetes 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。

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

答：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。

42.简述 Kubernetes 如何保证集群的安全性？

答：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。

43.简述 Kubernetes 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：

- 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。

44.简述 Kubernetes 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：

- 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。

45.简述 Kubernetes Secret 作用?

答：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。

46.简述 Kubernetes Secret 有哪些使用方式?

答：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。

47.简述 Kubernetes PodSecurityPolicy 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。

48.简述 Kubernetes PodSecurityPolicy 机制能实现哪些安全策略?

答:
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。

49.简述 Kubernetes 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）。

50.简述 Kubernetes 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。

- 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。

51.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。

52.简述 Kubernetes 网络策略原理?

答：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）。

53.简述 Kubernetes 中 flannel 的作用?

答：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。

54.简述 Kubernetes Calico 网络组件实现原理?

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。

55.简述 Kubernetes 共享存储的作用?

答：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。

56.简述 Kubernetes 数据持久化的方式有哪些?

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：

- 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。

57.简述 Kubernetes 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”。

58.简述 Kubernetes 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。

- 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。

59.简述 Kubernetes 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）。

- 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。

60.简述 Kubernetes CSI 模型?

答：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。

61.简述 Kubernetes 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、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。

62.简述 Kubernetes Pod 如何实现对节点的资源控制?

答：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。

63.简述 Kubernetes 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。

64.简述 Kubernetes Metric Service?

答：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。

65.简述 Kubernetes 中，如何使用 EFK 实现日志的统一管理?

答：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。

66.简述 Kubernetes 如何进行优雅的节点关机维护?

答：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。

67.简述 Kubernetes 集群联邦?

答：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。

68.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。
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。

69.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?

答：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。

70.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?

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。

71.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?

答：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。

72.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?

答：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。

73.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?

答：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。

74.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,公司如何有效地实现这种资源分配?

答：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。

75.您认为公司如何处理服务器及其安装?

答：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。

76.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?

答：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。

77.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?

答：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。

78.什么是 Google 容器引擎?

答：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。

79.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。

80.什么是 Headless Service?

答：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。

81.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），简述一下它们的工作原理和区别。

答：
1、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 节点上。

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 Reflector 来完成集中式的路由分发

通过将整个互联网的可扩展 IP 网络原则压缩到数据中心级别，Calico 在每一个计算节点利用 Linuxkernel 实现了一个高效的 vRouter 来负责数据转发，而每个 vRouter 通过 BGP 协议负责把自己上运行的 workload 的路由信息向整个 Calico 网络内传播，小规模部署可以直接互联，大规模下可通过指定的 BGProute reflector 来完成。这样保证最终所有的 workload 之间的数据流量都是通过 IP 包的方式完成互联的。

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 docker0 虚拟网卡转发到 flannel0 虚拟网卡（先可以不经过 docker0 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 docker0 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。

82.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。

83.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。

[root@master test]# vim nginx-deployment.yaml

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

```
[root@master test]# kubectl describe deploy nginx-deployment
[root@master test]# kubectl edit deploy nginx-deployment/
```

2、StatefullSet：适合有状态的服务部署

适合部署有状态应用

解决 Pod 的独立生命周期，保持 Pod 启动顺序和唯一性

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）

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

应用场景：例如数据库

无状态服务的特点：

deployment 认为所有的 pod 都是一样的

不用考虑顺序的要求

不用考虑在哪个 node 节点上运行

可以随意扩容和缩容

有状态服务的特点：

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

实例之间不对等的关系，以及依靠外部存储的应用

常规的 service 服务和无头服务的区别

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

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

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：

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

在每一个 Node 上运行一个 Pod

新加入的 Node 也同样会自动运行一个 Pod

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

一次性执行任务，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

诞生

通常情况下，service 和 pod 的 IP 仅可在集群内部访问

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、 3 ipvs、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.

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。

原理

Ingress 也是 Kubernetes API 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。

84.简述 k8s 的调度机制。

答：
1、Scheduler 工作原理：

请求及 Scheduler 调度步骤：

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。

节点优先级排序(Priority)：根据优先级选出最佳节点

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 的调用工作方式

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，Disk I/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。

85.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 Iptables 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 Cluster IP 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 netfilter 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，Ipvs 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。

86.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：

是 pod 里其他容器共享 Linux namespace 的基础

扮演 PID 1 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：。

87.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 是，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。

88.Pod 启动失败如何解决以及常见的原因有哪些

答：

1、一般查看系统资源是否满足，然后就是查看 pod 日志看看原因

2、describe 通过这个参数查看 pod 失败的原因

3、还有就是看组件日志，apiserver 等组件日志，有没有异常

4、访问不到看看网络插件状态和日志，还有就是 dns 状态和日志

89.简述 K8s 中 label 的几种应用场景。

答：
一些 Pod 有 Label（enter image description here）。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。

90.简述你知道的 Jenkins Pipeline 中脚本语法中的几个关键字。

答：
pipeline 是 jenkins2.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

pipiline 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 的自动扩展以及其插件集成的多个选项。

块(Blocks{})

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

章节(Sections)

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

指令(Directives)

- environment, options, parameters, triggers, stage, tools, when

步骤(steps)

- 执行脚本式 pipeline，如 script{}

91.Docker 的网络通信模式。

答：
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:NAMEorID 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。

92.你所用的到的日志分析工具有哪些以及它们如何与 K8s 集群通讯。

答：
背景：
k8s 上面会跑大量的 pod/容器。 特别是 集群相关的控制面 容器， 业务容器，有时候我们要对容器日志进行 监控与分析处理，这时一套好用的日志系统就格外的重要。

而成熟的日志解决方案有哪些呢？ 以前的 ELK ， k8s 主推的基于云原生的 EFK ，这里的 F 是 CNCF 认证的子项目 fluentd ，fluentd 是 ruby 的项目，由于个人体验习惯与 yaml 配置，以及项目特点， 选择了 filebeat ，所以 整套日志系统使用的技术包括（elastic search fidlebeat kibana）。

官方项目地址：
https://github.com/elastic/beats/tree/master/filebeat
https://www.elastic.co/guide/en/beats/filebeat/current/filebeat-getting-started.html
配置文件：
https://github.com/chenpfeisoo/EFK-logging-deployment
该配置文件中对于采集器有 fluentd ，也有 filebeat ，看个人喜好部署
注意事项：
对于 filebeat 我们会在 comfigmap 中配置 对采集做配置
比如 多行显示：
官方推荐

```
multiline.pattern: '^\['

multiline.negate: true

multiline.match: after
```

java 应用：

```
include_lines: ['Caused by']

multiline:

pattern: '^\['

negate:  true
```

match: after

93.Kubelet 与 kubeproxy 作用。Kubeproxy 的三种代理模式和各自的原理以及它们的区别。

答:
kubelet

kubelet 进程用于处理 master 下发的任务，管理 pod 中的容器，注册 自身所在的节点.

kube-proxy 运行机制解析

kube-proxy 本质上,类似一个反向代理. 我们可以把每个节点上运行的 kube-proxy 看作 service 的透明代理兼 LB.

Service 是 k8s 中资源的一种，也是 k8s 能够实现减少运维工作量，甚至免运维的关键点，我们公司的运维都要把服务搭在我们集群里，接触过的人应该都能体会到其方便之处。Service 能将 pod 的变化屏蔽在集群内部，同时提供负载均衡的能力，自动将请求流量分布到后端的 pod，这一功能的实现靠的就是 kube-proxy 的流量代理，一共有三种模式，userspace、iptables 以及 ipvs。

1、userspace

为每个 service 在 node 上打开一个随机端口（代理端口）

建立 iptables 规则，将 clusterip 的请求重定向到代理端口

到达代理端口（用户空间）的请求再由 kubeproxy 转发到后端 pod。

这里为什么需要建 iptables 规则，因为 kube-proxy 监听的端口在用户空间，所以需要一层 iptables 把访问服务的连接重定向给 kube-proxy 服务，这里就存在内核态到用户态的切换，代价很大，因此就有了 iptables。

2、iptables

kube-proxy 不再负责转发，数据包的走向完全由 iptables 规则决定，这样的过程不存在内核态到用户态的切换，效率明显会高很多。但是随着 service 的增加，iptables 规则会不断增加，导致内核十分繁忙（等于在读一张很大的没建索引的表）。

3、ipvs

用 ipset 存储 iptables 规则，这样规则的数量就能够得到有效控制，而在查找时就类似 hash 表的查找。

94.Iptables 四个表五个链。

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

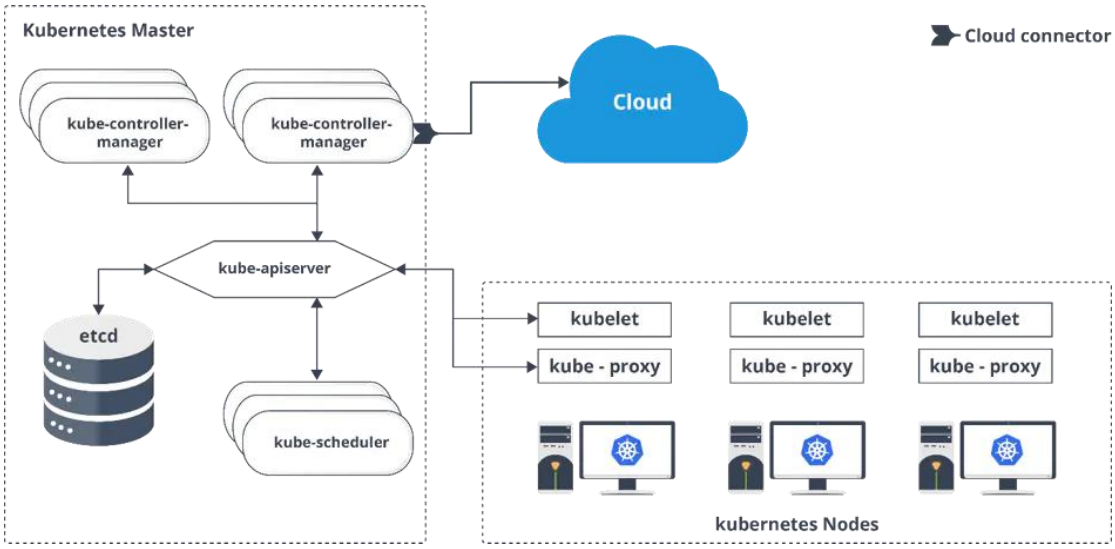
PREROUTING 链：在进行路由选择前处理数据包

95.Kubernetes 如何简化容器化部署？

答：由于典型的应用程序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，您需要一些可以平衡负载，缩放和监视容器的东西。由于 Kubernetes 与云无关，并且可以在任何公共/私有提供商上运行，因此必须选择简化容器化部署。

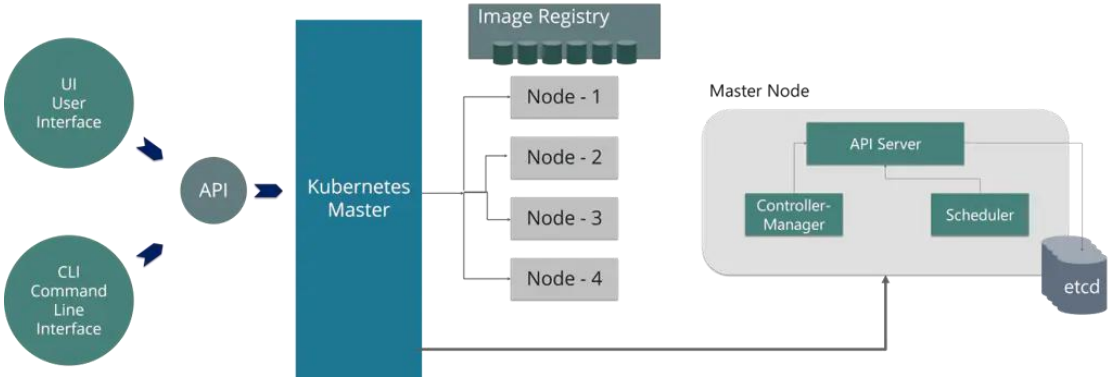
96.Kubernetes 体系结构有哪些不同的组成部分？

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。



97.您能否简要介绍一下 Kubernetes 中主节点的工作?

答：Kubernetes 主节点控制节点，并且在节点内部存在容器。现在，这些单独的容器包含在 Pod 内，每个 Pod 内，您可以根据配置和要求拥有各种数量的容器。因此，如果必须部署 Pod，则可以使用用户界面或命令行界面来部署它们。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了通信。



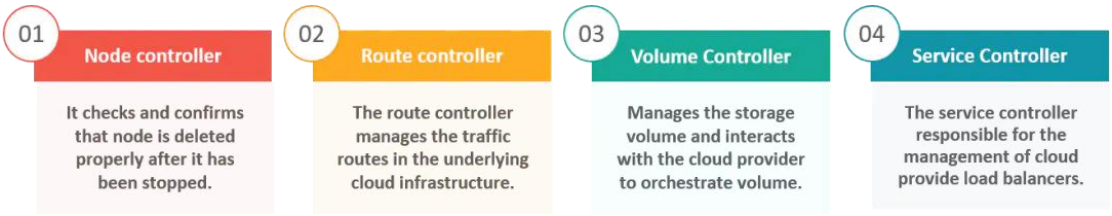
98.kube-apiserver 和 kube-scheduler 的作用是什么?

答：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。

kube 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。

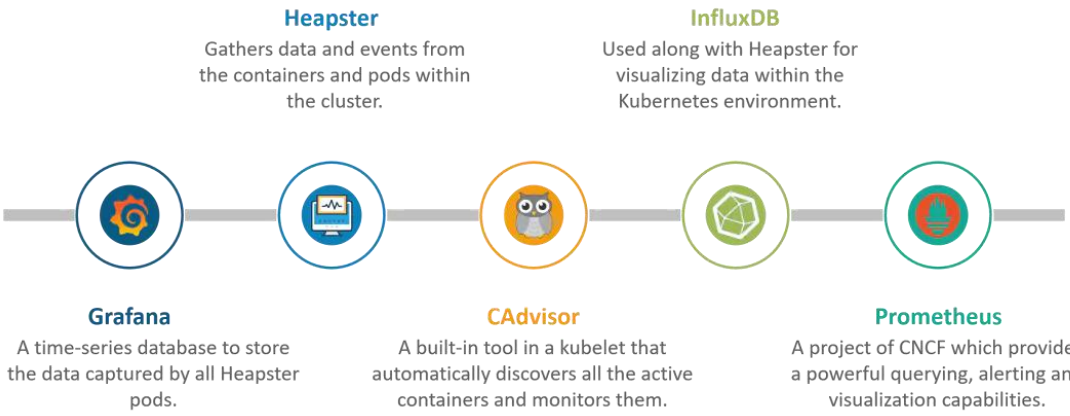
99.您对云控制器经理了解什么？

答：Cloud Controller Manager 负责持久性存储，网络路由，从核心 Kubernetes 特定代码中提取特定于云的代码以及管理与基础云服务的通信。根据您所运行的云平台，它可能会分成几个不同的容器，然后它使云供应商和 Kubernetes 代码得以开发而没有任何相互依赖关系。因此，云供应商可以在运行 Kubernetes 时开发他们的代码并与 Kubernetes 云控制器管理器连接。
各种类型的云控制器管理器如下：



100. 什么是容器资源监视？

答：对于用户而言，了解所有不同抽象层的应用程序性能和资源利用率非常重要，Kubernetes 通过在不同级别（例如容器，pod，服务和整个集群）创建抽象，从而对集群的管理进行了分解。现在，可以监视每个级别，这不过是容器资源监视而已。
各种容器资源监视工具如下：

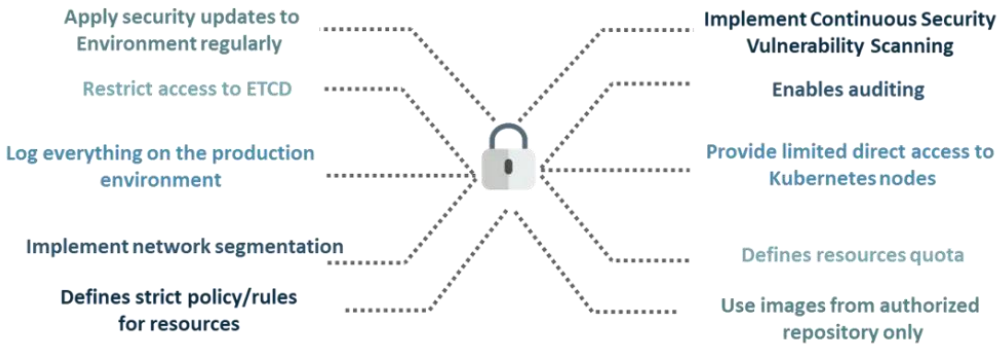


101. 副本集和复制控制器之间有什么区别？

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。
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。
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。

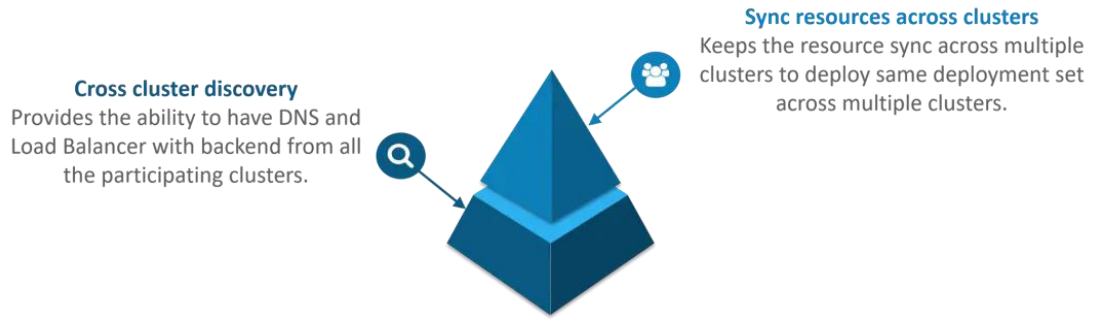
102. 使用 Kubernetes 时可以采取的最佳安全措施是什么？

答：
以下是使用 Kubernetes 时可以遵循的最佳安全措施：



103. 什么是联合集群？

答：借助联合集群，可以将多个 Kubernetes 集群作为一个集群进行管理。因此，您可以在一个数据中心/云中创建多个 Kubernetes 集群，并使用联合在一个地方控制/管理所有集群。
联合群集可以通过执行以下两项操作来实现此目的。请参考下图。



场景题

104. 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。

105. 考虑一家拥有非常分散的系统，拥有大量数据中心，虚拟机以及许多从事各种任务的员工的跨国公司。*您认为这样 的公司如何与 Kubernetes 一致地管理所有任务?*

答：众所周知，IT 部门启动了数千个容器，任务在分布式系统中的多个节点上运行。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 实践的功能。
因此，该公司可以使用 Kubernetes 定制其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力成为可能，它通过对各种容器联网解决方案和容器存储的广泛支持而提高了效率。

106. 考虑一种情况，公司希望通过保持最低成本来提高效率和技术运营速度。*您如何看待公司将如何实现这一目标?*

答：该公司可以通过构建 CI / CD 管道来实现 DevOps 方法，但是此处可能出现的一个问题是，配置可能需要花费一些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。一旦他们开始在云环境中工作，他们就可以在集群上调度容器，并可以在 Kubernetes 的帮助下进行编排。这种方法将帮助公司减少部署时间，并在各种环境中更快地完成部署。

107. 假设一家公司想要修改其部署方法，并希望构建一个可扩展性和响应性更高的平台。*您如何看待这家公司能够实现这一目标以满足他们的客户？*

答：为了给数百万客户提供他们期望的数字体验，该公司需要一个可扩展且响应迅速的平台，以便他们可以快速将数据获取到客户网站。现在，要做到这一点，公司应该从其私有数据中心（如果他们使用的是任何数据中心）迁移到任何云环境（例如 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。一旦他们准备好了基础框架，便可以开始使用可用的最佳编排平台，即 Kubernetes。这将使团队能够自主构建应用程序并非常快速地交付它们。

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

答：好了，要解决该问题，他们可以将其整体代码库转移到微服务设计中，然后将每个微服务都视为一个容器。因此，所有这些容器都可以在 Kubernetes 的帮助下进行部署和编排。

109. 我们所有人都知道从单服务到微服务的转变从开发方面解决了问题，但在部署方面却增加了问题。*公司如何解决部署方面的问题？*

答：该团队可以尝试使用容器编排平台（例如 Kubernetes）并在数据中心中运行它。因此，借助此工具，该公司可以生成模板化的应用程序，在五分钟内对其进行部署，并在此时将实际实例包含在登台环境中。这种 Kubernetes 项目将具有数十个并行运行的微服

务，以提高生产率，即使节点发生故障，也可以立即对其进行重新调度，而不会影响性能。

110. 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?*

答：解决这个问题的方法莫过于 Kubernetes。Kubernetes 确保有效地优化资源，并且仅使用特定应用程序所需的那些资源。因此，通过使用最佳的容器编排工具，公司可以有效地实现资源分配。

111. 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。*您认为公司将如何处理服务器及其安装?*

答：公司可以采用集装箱化的概念。一旦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 Prometheus 等容器监视工具来监视容器中的动作。因此，使用这样的容器，可以为它们提供更好的数据中心容量规划，因为由于服务和运行的硬件之间的这种抽象，它们现在将具有更少的约束。

112. 考虑一个公司要向具有各种环境的客户提供所有必需的分发产品的方案。*您如何看待他们如何动态地实现这一关键目标?*

答：该公司可以使用 Docker 环境，组成一个跨部门团队，以使用 Kubernetes 构建 Web 应用程序。这种框架将帮助公司实现在最短时间内将所需物品投入生产的目标。因此，通过运行这种机器，公司可以向所有具有各种环境的客户提供帮助。

113. 假设一家公司希望在从裸机到公共云的不同云基础架构上运行各种工作负载。*在存在不同接口的情况下，公司将如何实现这一目标?*

答：该公司可以将其基础架构分解为微服务，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。

其他

114. K8S 集群服务访问失败?

原因分析：证书不能被识别，其原因为：自定义证书，过期等。
解决方法：更新证书即可。

115. 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。

kubectl delete svc nginx-deployment

116. 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。

117. 外网无法访问 K8S 集群提供的服务?

原因分析：K8S 集群的 type 为 ClusterIP，未将服务暴露至外网。
解决方法：修改 K8S 集群的 type 为 NodePort 即可，于是可通过所有 K8S 集群节点访问服务。
kubectl edit svc nginx-deployment

118. pod 状态为 ErrImagePull?

readiness-httpget-pod 0/1 ErrImagePull 0 10s

```
[root@k8s-master01 ~]# kubectl get pod
NAME                    READY   STATUS         RESTARTS   AGE
myapp-pod               1/1     Running        5          21h
readiness-httpget-pod   0/1     ErrImagePull   0          10s
```

原因分析：image 无法拉取；

```
[root@k8s-master01 ~]# kubectl logs readiness-httpget-pod
Error from server (BadRequest): container "readiness-httpget-container" in pod "readiness-httpget-pod" is waiting to start: image can't be pulled
```

```
Events:
  Type     Reason     Age                From               Message
  ----     ------     ----               ----               -------
  Normal   Pulling    59m (x4 over 61m)  kubelet, k8s-node01  Pulling image "wagnyagnlinux/myapp:v1"
  Warning  Failed     59m (x4 over 61m)  kubelet, k8s-node01  Failed to pull image "wagnyagnlinux/myapp:v1": rpc error: code = Unknown desc = Error response from da
emon: pull access denied for wagnyagnlinux/myapp, repository does not exist or may require 'docker login'
  Warning  Failed     59m (x4 over 61m)  kubelet, k8s-node01  Error: ErrImagePull
  Normal   BackOff    59m (x6 over 61m)  kubelet, k8s-node01  Back-off pulling image "wagnyagnlinux/myapp:v1"
  Warning  Failed     56m (x19 over 61m)  kubelet, k8s-node01  Error: ImagePullBackOff
  Normal   Scheduled  5m25s              default-scheduler  Successfully assigned default/readiness-httpget-pod to k8s-node01
[root@k8s-master01 ~]#
```

解决方法：更换镜像即可。

119. 创建 init C 容器后，其状态不正常?

NAME READY STATUS RESTARTS AGE
myapp-pod 0/1 Init:0/2 0 20s
原因分析：查看日志发现，pod 一直出于初始化中；然后查看 pod 详细信息，定位 pod 创建失败的原因为：初始化容器未执行完毕。
Error from server (BadRequest): container "myapp-container" in pod "myapp-pod" is waiting to start: PodInitializing

```
[root@k8s-master01 ~]# kubectl describe pod myapp-pod
Name:         myapp-pod
Namespace:    default
Priority:     0
Node:         k8s-node01/192.168.66.20
Start Time:   Tue, 08 Jun 2021 13:13:08 +0800
Labels:       app=myapp
Annotations:  kubectl.kubernetes.io/last-applied-configuration:
                {"apiVersion":"v1","kind":"Pod","metadata":{"annotations":{},"la
bels":{"app":"myapp"},"name":"myapp-pod","namespace":"default"},"spec":{"c...
Status:       Pending
IP:           10.244.2.16
Init Containers:
  init-myservice:
    Container ID:  docker://2e53fe352546032bbe8840f40e4a8228bcae32c8165ad065566e
6183a8752824
    Image:         busybox
    Image ID:      docker-pullable://busybox@sha256:930490f97e5b921535c153e0e711
0d251134cc4b72bbb8133c6a5065cc68580d
    Port:          <none>
    Host Port:     <none>
    Command:
      sh
      -c
      until nslookup myservice;do echo waiting for myservice;sleep 2;done;
    State:          Running
      Started:      Tue, 08 Jun 2021 13:13:16 +0800
    Ready:          False
    Restart Count:  0
    Environment:    <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-dg4j6 (ro
)
```

```
    Command:
      sh
      -c
      echo The app is running! && sleep 3600
    State:          Waiting
      Reason:       PodInitializing
    Ready:          False
    Restart Count:  0
    Environment:    <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-dg4j6 (ro
)
Conditions:
  Type              Status
  Initialized       False
  Ready             False
  ContainersReady   False
  PodScheduled      True
Volumes:
  default-token-dg4j6:
    Type:        Secret (a volume populated by a Secret)
    SecretName:  default-token-dg4j6
    Optional:    false
QoS Class:       BestEffort
Node-Selectors:  <none>
Tolerations:     node.kubernetes.io/not-ready:NoExecute for 300s
                 node.kubernetes.io/unreachable:NoExecute for 300s
Events:
  Type    Reason     Age    From                Message
  ----    ------     ----   ----                -------
  Normal  Pulling    58m    kubelet, k8s-node01  Pulling image "busybox"
  Normal  Pulled     58m    kubelet, k8s-node01  Successfully pulled image "busy
box"
  Normal  Created    58m    kubelet, k8s-node01  Created container init-myservic
e
  Normal  Started    58m    kubelet, k8s-node01  Started container init-myservic
e
  Normal  Scheduled  2m37s  default-scheduler    Successfully assigned default/m
yapp-pod to k8s-node01
```

```
[root@k8s-master01 ~]# kubectl logs myapp-pod -c init-myservice
Server:         10.96.0.10
Address:        10.96.0.10:53

** server can't find myservice.default.svc.cluster.local: NXDOMAIN

*** Can't find myservice.svc.cluster.local: No answer
*** Can't find myservice.cluster.local: No answer
*** Can't find myservice.default.svc.cluster.local: No answer
*** Can't find myservice.svc.cluster.local: No answer
*** Can't find myservice.cluster.local: No answer
```

waiting for myservice

Server: 10.96.0.10
Address: 10.96.0.10:53

** server can't find myservice.default.svc.cluster.local: NXDOMAIN

*** Can't find myservice.svc.cluster.local: No answer
*** Can't find myservice.cluster.local: No answer
*** Can't find myservice.default.svc.cluster.local: No answer
*** Can't find myservice.svc.cluster.local: No answer
*** Can't find myservice.cluster.local: No answer
解决方法：创建相关 service，将 SVC 的 name 写入 K8S 集群的 coreDNS 服务器中，于是 coreDNS 就能对 POD 的 initC 容器执行过程中的域名解析了。
kubectl apply -f myservice.yaml

```
[root@k8s-master01 ~]# kubectl get svc
NAME         TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)   AGE
kubernetes   ClusterIP   10.96.0.1       <none>        443/TCP   6d3h
myservice    ClusterIP   10.101.138.80   <none>        80/TCP    28s
[root@k8s-master01 ~]#
```

```
[root@k8s-master01 ~]# kubectl get pod -w
NAME        READY   STATUS     RESTARTS   AGE
myapp-pod   0/1     Init:0/2   0          23m
myapp-pod   0/1     Init:1/2   0          23m
myapp-pod   0/1     Init:1/2   0          23m
```

```
[root@k8s-master01 ~]# kubectl apply -f mydb.yaml
service/mydb created
[root@k8s-master01 ~]#
[root@k8s-master01 ~]#
[root@k8s-master01 ~]# kubectl get pod -w
NAME        READY   STATUS     RESTARTS   AGE
myapp-pod   0/1     Init:1/2   0          27m
myapp-pod   0/1     PodInitializing   0          28m
myapp-pod   1/1     Running           0          28m
```

NAME READY STATUS RESTARTS AGE

myapp-pod 0/1 Init:1/2 0 27m
myapp-pod 0/1 PodInitializing 0 28m
myapp-pod 1/1 Running 0 28m

120. 探测存活 pod 状态为 CrashLoopBackOff?

原因分析：镜像问题，导致容器重启失败。
解决方法：更换镜像即可。

```yaml
apiVersion: v1
kind: Pod
metadata:
    name: readiness-httpget-pod
    namespace: default
spec:
    containers:
    - name: readiness-httpget-container
      image: hub.atguigu.com/library/myapp:v1
      imagePullPolicy: IfNotPresent    #若本地存在image则不需下载
      readinessProbe:
        httpGet:
            port: 80
            path: /index1.html
        initialDelaySeconds: 1
        periodSeconds: 3
```

```
[root@k8s-master01 ~]# kubectl get pod -w
NAME                    READY   STATUS              RESTARTS   AGE
myapp-pod               1/1     Running             0          56m
readiness-httpget-pod   0/1     ContainerCreating   0          9s
readiness-httpget-pod   0/1     Running             0          16s
```

121. POD 创建失败?

readiness-httpget-pod 0/1 Pending 0 0s
readiness-httpget-pod 0/1 Pending 0 0s
readiness-httpget-pod 0/1 ContainerCreating 0 0s
readiness-httpget-pod 0/1 Error 0 2s
readiness-httpget-pod 0/1 Error 1 3s
readiness-httpget-pod 0/1 CrashLoopBackOff 1 4s
readiness-httpget-pod 0/1 Error 2 15s
readiness-httpget-pod 0/1 CrashLoopBackOff 2 26s
readiness-httpget-pod 0/1 Error 3 37s
readiness-httpget-pod 0/1 CrashLoopBackOff 3 52s
readiness-httpget-pod 0/1 Error 4 82s
原因分析：镜像问题导致容器无法启动。

```
[root@k8s-master01 ~]# kubectl logs readiness-httpget-pod
url.js:106
    throw new errors.TypeError('ERR_INVALID_ARG_TYPE', 'url', 'string', url);
    ^

TypeError [ERR_INVALID_ARG_TYPE]: The "url" argument must be of type string. Received type undefined
    at Url.parse (url.js:106:11)
    at Object.urlParse [as parse] (url.js:100:13)
    at module.exports (/myapp/node_modules/mongodb/lib/url_parser.js:17:23)
    at connect (/myapp/node_modules/mongodb/lib/mongo_client.js:159:16)
    at Function.MongoClient.connect (/myapp/node_modules/mongodb/lib/mongo_client.js:110:3)
    at Object.<anonymous> (/myapp/app.js:12:13)
    at Module._compile (module.js:641:30)
    at Object.Module._extensions..js (module.js:652:10)
    at Module.load (module.js:560:32)
    at tryModuleLoad (module.js:503:12)
    at Function.Module._load (module.js:495:3)
    at Function.Module.runMain (module.js:682:10)
    at startup (bootstrap_node.js:191:16)
    at bootstrap_node.js:613:3
```

解决方法：更换镜像。

```yaml
apiVersion: v1
kind: Pod
metadata:
    name: readiness-httpget-pod
    namespace: default
spec:
    containers:
    - name: readiness-httpget-container
      image: hub.atguigu.com/library/nginx
      imagePullPolicy: IfNotPresent    #若本地存在image则不需下载
      readinessProbe:
        httpGet:
          port: 80
          path: /index1.html
        initialDelaySeconds: 1
        periodSeconds: 3
```

```yaml
apiVersion: v1
kind: Pod
metadata:
    name: readiness-httpget-pod
    namespace: default
spec:
    containers:
    - name: readiness-httpget-container
      image: hub.atguigu.com/library/nginx
      imagePullPolicy: IfNotPresent    #若本地存在image则不需下载
      readinessProbe:
        httpGet:
            port: 80
            path: /index1.html
        initialDelaySeconds: 1
        periodSeconds: 3
```

122. POD 的 ready 状态未进入?

readiness-httpget-pod 0/1 Running 0 116s

原因分析：POD 的执行命令失败，无法获取资源。

```
[root@k8s-master01 ~]# kubectl logs pod readiness-httpget-pod
Error from server (NotFound): pods "pod" not found
[root@k8s-master01 ~]# kubectl logs readiness-httpget-pod
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf
10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /etc/nginx/conf.d/default.conf
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
/docker-entrypoint.sh: Configuration complete; ready for start up
2021/06/11 07:10:12 [notice] 1#1: using the "epoll" event method
2021/06/11 07:10:12 [notice] 1#1: nginx/1.21.0
2021/06/11 07:10:12 [notice] 1#1: built by gcc 8.3.0 (Debian 8.3.0-6)
2021/06/11 07:10:12 [notice] 1#1: OS: Linux 5.4.123-1.el7.elrepo.x86_64
2021/06/11 07:10:12 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 52706963:52706963
2021/06/11 07:10:12 [notice] 1#1: start worker processes
2021/06/11 07:10:12 [notice] 1#1: start worker process 30
2021/06/11 07:10:12 [notice] 1#1: start worker process 31
2021/06/11 07:10:14 [error] 30#30: *1 open() "/usr/share/nginx/html/index1.html" failed (2: No such file or directory), client: 10.244.2.1, server: localhost, reques
t: "GET /index1.html HTTP/1.1", host: "10.244.2.25:80"
10.244.2.1 - - [11/Jun/2021:07:10:14 +0000] "GET /index1.html HTTP/1.1" 404 153 "-" "kube-probe/1.15" "-"
10.244.2.1 - - [11/Jun/2021:07:10:17 +0000] "GET /index1.html HTTP/1.1" 404 153 "-" "kube-probe/1.15" "-"
2021/06/11 07:10:17 [error] 30#30: *2 open() "/usr/share/nginx/html/index1.html" failed (2: No such file or directory), client: 10.244.2.1, server: localhost, reques
t: "GET /index1.html HTTP/1.1", host: "10.244.2.25:80"
```

解决方法：进入容器内部，创建 yaml 定义的资源

```
[root@k8s-master01 ~]# kubectl exec -it readiness-httpget-pod -- /bin/sh
# cd /usr/share/nginx/html
# ls
50x.html  index.html  index1.html
# echo 123 > index1.html
#
```

```
[root@k8s-master01 ~]# kubectl get pod -w |grep readiness-httpget-pod
readiness-httpget-pod   0/1     Pending   0          0s
readiness-httpget-pod   0/1     Pending   0          0s
readiness-httpget-pod   0/1     ContainerCreating   0          0s
readiness-httpget-pod   0/1     Running             0          1s
readiness-httpget-pod   1/1     Running             0          26m
```

123. pod 创建失败?

```
[root@k8s-master01 ~]# kubectl apply -f myregistry-secret.yml
error: error validating "myregistry-secret.yml": error validating data: ValidationError(Pod.spec.imagePullSecrets[0]): invalid type for io.k8s.api.
core.v1.LocalObjectReference: got "string", expected "map"; if you choose to ignore these errors, turn validation off with --validate=false
[root@k8s-master01 ~]#
[root@k8s-master01 ~]#
[root@k8s-master01 ~]# cat myregistry-secret.yml
apiVersion: v1
kind: Pod
metadata:
    name: foo
spec:
    containers:
        - image: hub.atguigu.com/library/nginx:latest
          name: foo
    imagePullSecrets:
        - name:  myregistrykey
```

原因分析：yml 文件内容出错—使用中文字符；

解决方法：修改 myregistrykey 内容即可。

```
[root@k8s-master01 ~]# vi myregistry-secret.yml
[root@k8s-master01 ~]#
[root@k8s-master01 ~]#
[root@k8s-master01 ~]# kubectl apply -f myregistry-secret.yml
pod/foo created
[root@k8s-master01 ~]# cat myregistry-secret.yml
apiVersion: v1
kind: Pod
metadata:
    name: foo
spec:
    containers:
    - image: hub.atguigu.com/library/nginx:latest
      name: foo
    imagePullSecrets:
    - name: myregistrykey
```

124. kube-flannel-ds-amd64-ndsf7 插件 pod 的 status 为 Init:0/1?

```
[root@k8s-master ~]# kubectl get pod -n kube-system -o wide
NAME                                   READY   STATUS             RESTARTS   AGE   IP            NODE         NOMINATED NODE   READINESS GATE
S
coredns-7f89b7bc75-7wbtx               1/1     Running            0          54m   10.244.0.3    k8s-master   <none>           <none>
coredns-7f89b7bc75-h2p7h               1/1     Running            0          54m   10.244.0.2    k8s-master   <none>           <none>
etcd-k8s-master                        1/1     Running            0          55m   10.0.0.81     k8s-master   <none>           <none>
kube-apiserver-k8s-master              1/1     Running            0          55m   10.0.0.81     k8s-master   <none>           <none>
kube-controller-manager-k8s-master     1/1     Running            0          55m   10.0.0.81     k8s-master   <none>           <none>
kube-flannel-ds-amd64-bdln2            1/1     Running            0          25m   10.0.0.83     k8s-slave2   <none>           <none>
kube-flannel-ds-amd64-d95vh            1/1     Running            0          25m   10.0.0.81     k8s-master   <none>           <none>
kube-flannel-ds-amd64-ndsf7            0/1     Init:0/1           0          25m   10.0.0.82     k8s-slave1   <none>           <none>
kube-proxy-67bjs                       1/1     Running            0          54m   10.0.0.83     k8s-slave2   <none>           <none>
kube-proxy-8bd5m                       0/1     ImagePullBackOff   0          54m   10.0.0.82     k8s-slave1   <none>           <none>
kube-proxy-jpwrf                       1/1     Running            0          54m   10.0.0.81     k8s-master   <none>           <none>
kube-scheduler-k8s-master              1/1     Running            0          55m   10.0.0.81     k8s-master   <none>           <none>
[root@k8s-master ~]#
```

排查思路：kubectl -n kube-system describe pod kube-flannel-ds-amd64-ndsf7 #查询 pod 描述信息；

```
                 node.kubernetes.io/unschedulable:NoSchedule op=Exists
Events:
  Type    Reason     Age   From               Message
  ----    ------     ----  ----               -------
  Normal  Scheduled  27m   default-scheduler  Successfully assigned kube-system/kube-flannel-ds-amd64-ndsf7 to k8s-slave1
  Normal  Pulling    27m   kubelet            Pulling image "quay.io/coreos/flannel:v0.11.0-amd64"
[root@k8s-master ~]#
```

原因分析：k8s-slave1 节点拉取镜像失败。

解决方法：登录 k8s-slave1，重启 docker 服务，手动拉取镜像。

```
[root@k8s-slave1 ~]# docker pull 10.0.0.81:5000/flannel:v0.11.0-amd64
v0.11.0-amd64: Pulling from flannel
cd784148e348: Pull complete
04ac94e9255c: Pull complete
e10b013543eb: Pull complete
005e31e443b1: Pull complete
74f794f05817: Pull complete
Digest: sha256:bd76b84c74ad70368a2341c2402841b75950df881388e43fc2aca000c546653a
Status: Downloaded newer image for 10.0.0.81:5000/flannel:v0.11.0-amd64
10.0.0.81:5000/flannel:v0.11.0-amd64
```

k8s-master 节点，重新安装插件即可。

kubectl create -f kube-flannel.yml;kubectl get nodes

```
[root@k8s-master ~]# kubectl get nodes
NAME         STATUS   ROLES                  AGE   VERSION
k8s-master   Ready    control-plane,master   83m   v1.20.4
k8s-slave1   Ready    <none>                 83m   v1.20.4
k8s-slave2   Ready    <none>                 83m   v1.20.4
```

125. K8S 创建服务 status 为 ErrImagePull?

```
[root@k8s-master ~]# kubectl get pod -o wide
NAME         READY   STATUS         RESTARTS   AGE   IP           NODE         NOMINATED NODE   READINESS GATES
test-nginx   0/1     ErrImagePull   0          96s   10.244.2.2   k8s-slave2   <none>           <none>
```

排查思路：

kubectl describe pod test-nginx

```
Events:
  Type     Reason     Age                From               Message
  ----     ------     ----               ----               -------
  Normal   Scheduled  5m33s              default-scheduler  Successfully assigned default/test-nginx to k8s-slave2
  Warning  Failed     69s (x3 over 4m7s)   kubelet            Failed to pull image "nginx:alpine": rpc error: code = Unknown desc = contex
t canceled
  Warning  Failed     69s (x3 over 4m7s)   kubelet            Error: ErrImagePull
  Normal   BackOff    32s (x5 over 4m7s)   kubelet            Back-off pulling image "nginx:alpine"
  Warning  Failed     32s (x5 over 4m7s)   kubelet            Error: ImagePullBackOff
  Normal   Pulling    20s (x4 over 5m32s)  kubelet            Pulling image "nginx:alpine"
```

原因分析：拉取镜像名称问题。

解决方法：删除错误 pod；重新拉取镜像；

kubectl delete pod test-nginx;kubectl run test-nginx --image=10.0.0.81:5000/nginx:alpine

126. 不能进入指定容器内部?

```
[root@k8s-master01 ~]# kubectl exec volume-test-pod -c volume-test-container -it -- /bin/sh
Error from server (BadRequest): container volume-test-container is not valid for pod volume-test-pod
```

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。

```
[root@k8s-master01 ~]# kubectl apply -f volume-emptyDir.yml
pod/volume-test-pod created
[root@k8s-master01 ~]# kubectl get pod
NAME              READY   STATUS    RESTARTS   AGE
volume-test-pod   2/2     Running   0          2s
```

```
[root@k8s-master01 ~]# kubectl exec volume-test-pod -c volume-test-container -it -- ls /cache
test.txt
[root@k8s-master01 ~]# kubectl exec volume-test-pod -c liveness-exec-container -it -- ls /test
test.txt
```

127. 创建 PV 失败?

原因分析：pv 的 name 字段重复。

解决方法：修改 pv 的 name 字段即可。

```
[root@k8s-master01 ~]# kubectl apply -f PV.yml
persistentvolume/nfspv1 unchanged
persistentvolume/nfspv01 unchanged
persistentvolume/nfspv02 created
persistentvolume/nfspv03 created
[root@k8s-master01 ~]# kubectl get pv
NAME      CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS      CLAIM   STORAGECLASS   REASON   AGE
nfspv01   5Gi        RWO            Retain           Available           nfs                     5m5s
nfspv02   5Gi        RWO            Retain           Available           nfs                     1s
nfspv03   5Gi        RWO            Retain           Available           nfs                     1s
nfspv1    10Gi       RWO            Retain           Available           nfs                     20m
```

128. pod 无法挂载 PVC?

```
[root@k8s-master01 ~]# kubectl get pv
NAME      CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS      CLAIM   STORAGECLASS   REASON   AGE
nfspv01   5Gi        RWX            Retain           Available           nfs                     9m23s
nfspv02   5Gi        RWX            Retain           Available           nfs                     4m19s
nfspv03   1Gi        RWX            Retain           Available           slow                    4m19s
nfspv1    10Gi       RWO            Retain           Available           nfs                     25m
```

```
[root@k8s-master01 ~]# kubectl get pvc
NAME        STATUS   VOLUME   CAPACITY   ACCESS MODES   STORAGECLASS   AGE
www-web-0   Bound    nfspv1   10Gi       RWO            nfs            1s
[root@k8s-master01 ~]# kubectl get pod
NAME    READY   STATUS    RESTARTS   AGE
web-0   1/1     Running   0          18s
web-1   0/1     Pending   0          16s
```

原因分析：pod 无法挂载 PVC。

```
[root@k8s-master01 ~]# kubectl get pv
NAME      CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS      CLAIM                STORAGECLASS   REASON   AGE
nfspv01   5Gi        RWX            Retain           Available                        nfs                     43m
nfspv02   5Gi        RWO            Retain           Bound       default/www-web-1    nfs                     38m
nfspv03   1Gi        RWO            Retain           Available                        slow                    38m
nfspv1    10Gi       RWO            Retain           Bound       default/www-web-0    nfs                     59m
[root@k8s-master01 ~]# kubectl get pvc
NAME        STATUS   VOLUME    CAPACITY   ACCESS MODES   STORAGECLASS   AGE
www-web-0   Bound    nfspv1    10Gi       RWO            nfs            29m
www-web-1   Bound    nfspv02   5Gi        RWO            nfs            29m
www-web-2   Pending                                      nfs            7m28s
```

```
                mountPath: /usr/share/ngi
  volumeClaimTemplates:
  - metadata:
      name: www
    spec:
      accessModes: ["ReadWriteOnce"]
      storageClassName: "nfs"
      resources:
        requests:
          storage: 1Gi
```

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。

```
[root@k8s-master01 ~]# kubectl get pv
NAME      CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS      CLAIM               STORAGECLASS   REASON   AGE
nfspv01   5Gi        RWX            Retain           Available                       slow                    67s
nfspv02   5Gi        RWO            Retain           Bound       default/www-web-1   nfs                     67s
nfspv03   1Gi        RWO            Retain           Bound       default/www-web-0   nfs                     67s
nfspv1    10Gi       RWO            Retain           Bound       default/www-web-2   nfs                     67s
[root@k8s-master01 ~]# kubectl get pvc
NAME        STATUS   VOLUME    CAPACITY   ACCESS MODES   STORAGECLASS   AGE
www-web-0   Bound    nfspv03   1Gi        RWO            nfs            13s
www-web-1   Bound    nfspv02   5Gi        RWO            nfs            10s
www-web-2   Bound    nfspv1    10Gi       RWO            nfs            7s
[root@k8s-master01 ~]# kubectl get pod
NAME    READY   STATUS    RESTARTS   AGE
web-0   1/1     Running   0          13s
web-1   1/1     Running   0          10s
web-2   1/1     Running   0          7s
[root@k8s-master01 ~]# kubectl get statefulset
NAME   READY   AGE
web    3/3     14s
```

129. 问题：pod 使用 PV 后，无法访问其内容?

```
[root@k8s-master01 ~]# kubectl get pod -o wide
NAME    READY   STATUS    RESTARTS   AGE   IP             NODE         NOMINATED NODE   READINESS GATES
web-0   1/1     Running   0          45m   10.244.2.157   k8s-node01   <none>           <none>
web-1   1/1     Running   0          45m   10.244.1.77    k8s-node02   <none>           <none>
web-2   1/1     Running   0          45m   10.244.2.158   k8s-node01   <none>           <none>
[root@k8s-master01 ~]# curl 10.244.2.157
<html>
<head><title>403 Forbidden</title></head>
<body>
<center><h1>403 Forbidden</h1></center>
<hr><center>nginx/1.21.0</center>
</body>
</html>
[root@k8s-master01 ~]# kubectl get svc
NAME         TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
kubernetes   ClusterIP   10.96.0.1    <none>        443/TCP   23d
nginx        ClusterIP   None         <none>        80/TCP    87m
[root@k8s-master01 ~]# kubectl get pv
NAME      CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS      CLAIM               STORAGECLASS   REASON   AGE
nfspv01   5Gi        RWX            Retain           Available                       slow                    47m
nfspv02   5Gi        RWO            Retain           Bound       default/www-web-1   nfs                     47m
nfspv03   1Gi        RWO            Retain           Bound       default/www-web-0   nfs                     47m
nfspv1    10Gi       RWO            Retain           Bound       default/www-web-2   nfs                     47m
```

原因分析：nfs 卷中没有文件或权限不对。

```
[root@k8s-master01 ~]# kubectl exec web-0 -it -- cat /usr/share/nginx/html/test.html
cat: /usr/share/nginx/html/test.html: No such file or directory
command terminated with exit code 1
```

```
[root@k8s-harbor nfs2]# cd ../nfs3
[root@k8s-harbor nfs3]# ls
[root@k8s-harbor nfs3]#
```

解决方法：在 nfs 卷中创建文件并授予权限。

```
[root@k8s-harbor nfs3]# echo "hello world" >test.html
[root@k8s-harbor nfs3]# cat test.html
hello world
[root@k8s-harbor nfs3]# ll test.html
-rw-r--r-- 1 root root 12 Jun 25 11:41 test.html
[root@k8s-harbor nfs3]# chmod 777 test.html
[root@k8s-harbor nfs3]# chown nfsnobody test.html
[root@k8s-harbor nfs3]# ll test.html
-rwxrwxrwx 1 nfsnobody root 12 Jun 25 11:41 test.html
[root@k8s-harbor nfs3]# mv test.html index.html
[root@k8s-harbor nfs3]# ll index.html
-rwxrwxrwx 1 nfsnobody root 12 Jun 25 11:41 index.html
```

```
[root@k8s-master01 ~]# curl 10.244.2.157
hello world
[root@k8s-master01 ~]#
```

130. 查看节点状态失败?

Error from server (NotFound): the server could not find the requested resource (get services http:heapster:)

原因分析：没有 heapster 服务。

解决方法：安装 promethus 监控组件即可。

```
[root@k8s-master01 manifests]# kubectl get pod -n monitoring
NAME                                   READY   STATUS    RESTARTS   AGE
alertmanager-main-0                    2/2     Running   0          5m50s
alertmanager-main-1                    2/2     Running   0          5m40s
alertmanager-main-2                    2/2     Running   0          5m30s
grafana-7dc5f8f9f6-9x682               1/1     Running   0          9m11s
kube-state-metrics-5cbd67455c-zzb7n    4/4     Running   0          5m45s
node-exporter-42qk9                    2/2     Running   0          9m10s
node-exporter-b45s8                    2/2     Running   0          9m10s
node-exporter-h4pwr                    2/2     Running   0          9m10s
prometheus-adapter-668748ddbd-p6mzb    1/1     Running   0          9m11s
prometheus-k8s-0                       3/3     Running   1          5m49s
prometheus-k8s-1                       3/3     Running   1          5m49s
prometheus-operator-7447bf4dcb-hc6nc   1/1     Running   0          9m11s
```

```
[root@k8s-master01 manifests]# kubectl top node
NAME           CPU(cores)   CPU%   MEMORY(bytes)   MEMORY%
k8s-master01   451m         22%    3155Mi          82%
k8s-node01     110m         5%     1065Mi          37%
k8s-node02     115m         5%     1174Mi          63%
```

131. pod 一直处于 pending'状态?

```
[root@k8s-master01 ~]# kubectl get pod -o wide
NAME                  READY   STATUS    RESTARTS   AGE   IP       NODE     NOMINATED NODE   READINESS GATES
nod-affinity-required   0/1     Pending   0          12m   <none>   <none>   <none>           <none>
```

原因分析：由于已使用同样镜像发布了 pod，导致无节点可调度。

```
[root@k8s-master01 ~]# kubectl describe pod nod-affinity-required
Name:         nod-affinity-required
Namespace:    default
Priority:     0
```

解决方法：删除所有 pod 后部署 pod 即可。

```
[root@k8s-master01 ~]# vi node_soft_policy.yaml
[root@k8s-master01 ~]#
[root@k8s-master01 ~]# kubectl get pod
NAME                    READY   STATUS    RESTARTS   AGE
nod-affinity-required   0/1     Pending   0          19m
[root@k8s-master01 ~]# kubectl delete pod --all
pod "nod-affinity-required" deleted
[root@k8s-master01 ~]# kubectl get pod
No resources found.
[root@k8s-master01 ~]# kubectl apply -f node_soft_policy.yaml
pod/nod-affinity-preferred created
[root@k8s-master01 ~]# kubectl get pod
NAME                    READY   STATUS    RESTARTS   AGE
nod-affinity-preferred  1/1     Running   0          2s
[root@k8s-master01 ~]# kubectl apply -f node_soft_policy.yaml
pod/nod-affinity-preferred unchanged
```

132. helm 安装组件失败?

[root@k8s-master01 hello-world]# helm install

Error: This command needs 1 argument: chart nam

[root@k8s-master01 hello-world]# helm install ./
Error: no Chart.yaml exists in directory "/root/hello-world"

原因分析：文件名格式不对。

解决方法：mv chart.yaml Chart.yaml

```
[root@k8s-master01 hello-world]# ls
Chart.yaml  templates
[root@k8s-master01 hello-world]# helm install ./
NAME:   joyous-wasp
LAST DEPLOYED: Wed Jul  7 21:31:46 2021
NAMESPACE: default
STATUS: DEPLOYED

RESOURCES:
==> v1/Pod(related)
NAME                          READY  STATUS             RESTARTS  AGE
hello-world-86d6cf5f4d-hbf4t  0/1    ContainerCreating  0         2s

==> v1/Service
NAME         TYPE       CLUSTER-IP      EXTERNAL-IP  PORT(S)       AGE
hello-world  NodePort   10.106.218.186  <none>       80:31658/TCP  2s

==> v1beta1/Deployment
NAME         READY  UP-TO-DATE  AVAILABLE  AGE
hello-world  0/1    0           0          2s
```