k8s 面试题

1、什么是 k8s?

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。

k8s 是一个 docker 集群的管理工具
k8s 是容器的编排工具

2、k8s 的核心功能

1.自愈

自愈： 重新启动失败的容器，在节点不可用时，替换和重新调度节点上的容器，对用户定义的健康检查不响应的容器会被中止，并且在容器准备好服务之前不会把其向客户端广播。

2.弹性伸缩

通过监控容器的 cpu 的负载值,如果这个平均高于 80%,增加容器的数量,如果这个平均低于 10%,减少容器的数量

3.服务的自动发现和负载均衡

不需要修改您的应用程序来使用不熟悉的服务发现机制，Kubernetes 为容器提供了自己的 IP 地址和一组容器的单个 DNS 名称，并可以在它们之间进行负载均衡。

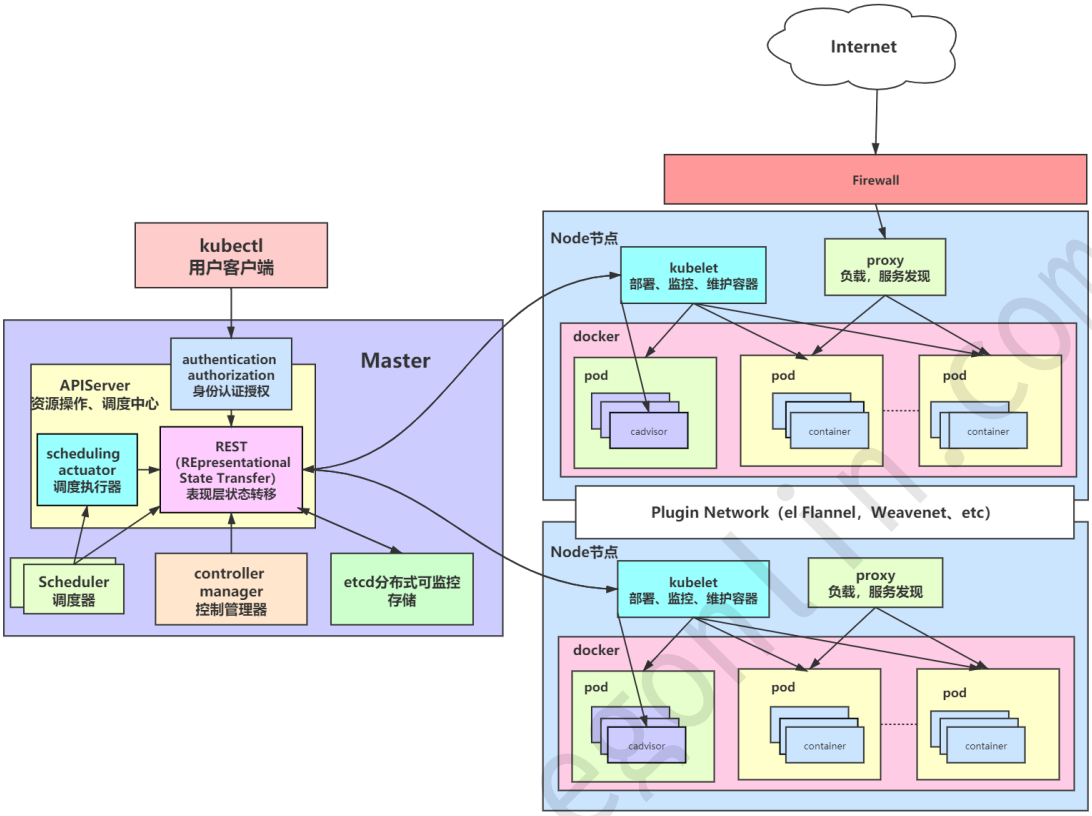
4.滚动升级和一键回滚

Kubernetes 逐渐部署对应用程序或其配置的更改，同时监视应用程序运行状况，以确保它不会同时终止所有实例。 如果出现问题，Kubernetes 会为您恢复更改，利用日益增长的部署解决方案的生态系统。

5.私密配置文件管理

web 容器里面,数据库的账户密码(测试库密码)

3、k8s 的组成？



1.Master 节点（默认不参加工作）

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(可以不在 master 节点)：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）；

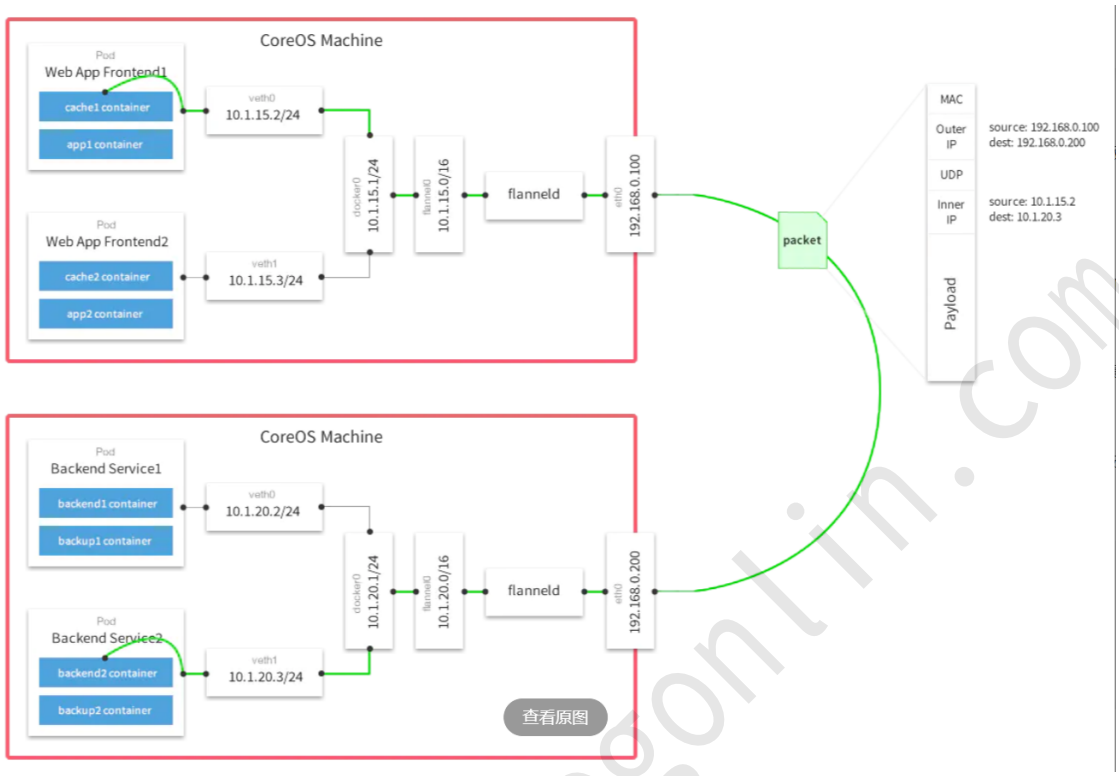
container-runtime：

是负责管理运行容器的软件，比如 docker

3.扩展组件

组件名称	说明
kube-dns	负责为整个集群提供 DNS 服务
Ingress Controller	为服务提供外网入口
Heapster	提供资源监控
Dashboard	提供 GUI
Federation	提供跨可用区的集群
Fluentd-elasticsearch	提供集群日志采集、存储与查询
flannel	提供集群间的网络

4、说出对 fannel 的了解



1.功能

让集群中的不同节点主机创建的 Docker 容器都具有全集群唯一的虚拟 IP 地址。但在默认的 Docker 配置中，每个 Node 的 Docker 服务会分别负责所在节点容器的 IP 分配。Node 内部得容器之间可以相互访问,但是跨主机(Node)网络相互间是不能通信。Flannel 设计目的就是为集群中所有节点重新规划 IP 地址的使用规则，从而使得不同节点上的容器能够获得"同属一个内网"且"不重复的"IP 地址，并让属于不同节点上的容器能够直接通过内网 IP 通信。

2.原理

Flannel 使用 etcd 存储配置数据和子网分配信息。flannel 启动之后，后台进程首先检索配置和正在使用的子网列表，然后选择一个可用的子网，然后尝试去注册它。etcd 也存储这个每个主机对应的 ip。flannel 使用 etcd 的 watch 机制监视/coreos.com/network/subnets 下面所有元素的变化信息，并且根据它来维护一个路由表。为了提高性能，flannel 优化了 Universal TAP/TUN 设备，对 TUN 和 UDP 之间的 ip 分片做了代理。

3.工作流程

1、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，这是个 P2P 的虚拟网卡，flanneld 服务监听在网卡的另外一端。

2、Flannel 通过 Etcd 服务维护了一张节点间的路由表，该张表里保存了各个节点主机的子网网段信息。

3、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样的由 docker0 路由到达目标容器。

5、说一下你对 fannel 和 calico 了解及区别？

Flannel

由 CoreOS 开发的项目 Flannel，可能是最直接和最受欢迎的 CNI 插件。它是容器编排系统中最成熟的网络结构示例之一，旨在实现更好的容器间和主机间网络。随着 CNI 概念的兴起，Flannel CNI 插件算是早期的入门。

1）安装简单，不需要专门的数据存储

Flannel 相对容易安装和配置。它被打包为单个二进制文件 flanneld，许多常见的 Kubernetes 集群部署工具和许多 Kubernetes 发行版都可以默认安装 Flannel。Flannel 可以使用 Kubernetes 集群的现有 etcd 集群来使用 API 存储其状态信息，因此不需要专用的数据存储。

Flannel 配置第 3 层 IPv4 overlay 网络。它会创建一个大型内部网络，跨越集群中每个节点。在此 overlay 网络中，每个节点都有一个子网，用于在内部分配 IP 地址。在配置 pod 时，每个节点上的 Docker 桥接口都会为每个新容器分配一个地址。同一主机中的 Pod 可以使用 Docker 桥接进行通信，而不同主机上的 pod 会使用 flanneld 将其流量封装在 UDP 数据包中，以便路由到适当的目标。

Flannel 有几种不同类型的后端可用于封装和路由。默认和推荐的方法是使用 VXLAN，因为 VXLAN 性能更良好并且需要的手动干预更少。

总的来说，Flannel 是大多数用户的不错选择。从管理角度来看，它提供了一个简单的网络模型，用户只需要一些基础知识，就可以设置适合大多数用例的环境。一般来说，在初期使用 Flannel 是一个稳妥安全的选择，直到你开始需要一些它无法提供的东西。

Calico

1）功能全面、灵活性高。

Calico 是 Kubernetes 生态系统中另一种流行的网络选择。虽然 Flannel 被公认为是最简单的选择，但 Calico 以其性能、灵活性而闻名。Calico 的功能更为全面，不仅提供主机和 pod 之间的网络连接，还涉及网络安全和管理。Calico CNI 插件在 CNI 框架内封装了 Calico 的功能。

2）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。

它创建的网络环境具有简单和复杂的属性。与 Flannel 不同，Calico 不使用 overlay 网络。相反，Calico 配置第 3 层网络，该网络使用 BGP 路由协议在主机之间路由数据包。这意味着在主机之间移动时，不需要将数据包包装在额外的封装层中。BGP 路由机制可以本地引导数据包，而无需额外在流量层中打包流量。

3）有标准的调试工具，方便排错

除了性能优势之外，在出现网络问题时，用户还可以用更常规的方法进行故障排除。虽然使用 VXLAN 等技术进行封装也是一个不错的解决方案，但该过程处理数据包的方式同场难以追踪。使用 Calico，标准调试工具可以访问与简单环境中相同的信息，从而使更多开发人员和管理员更容易理解行为。

总结

两者都属于 k8s 网络插件。我们公司使用的是 flannel，calico 性能更好也更灵活，但是 flannel 更加简单好用，当公司由二次开发 kubernetes 的场景时，建议使用 calico，当有 IPV6 使用场景时，必须使用 calico。

6、kubectl 这个命令执行的过程？(以部署 nginx 服务为例)

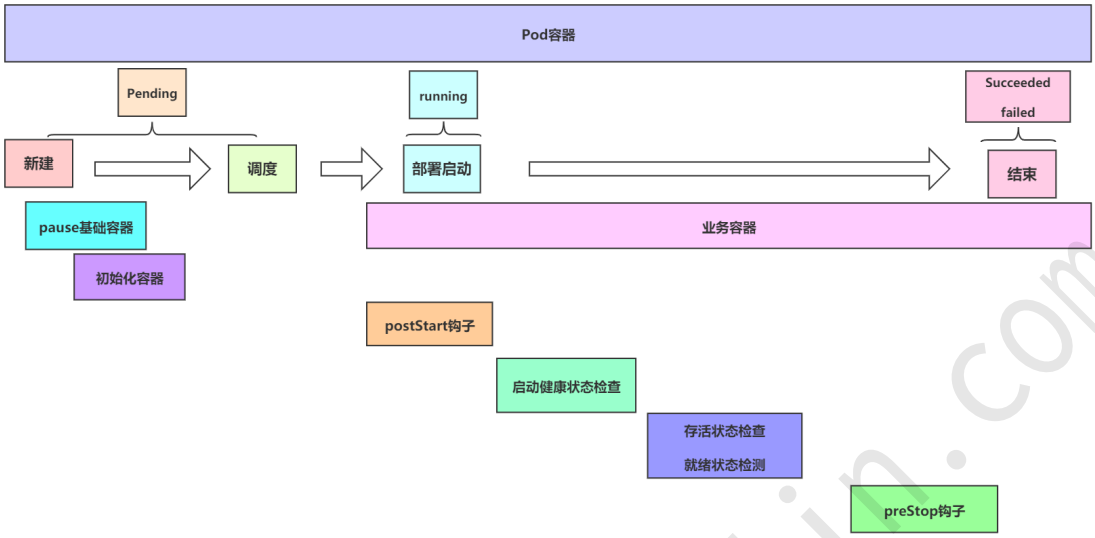
1、kubectl 发送了一个部署 nginx 的任务
2、进入 Master 节点，进行安全认证，
3、认证通过后，APIserver 接受指令
4、将部署的命令数据记录到 etcd 中
5、APIserver 再读取 etcd 中的命令数据
6、APIserver 找到 scheduler（调度器），说要部署 nginx
7、scheduler（调度器）找 APIserver 调取工作节点数据。
8、APIserver 调取 etcd 中存储的数据，并将数据发给 scheduler。
9、scheduler 通过计算，比较找到适合部署 nginx 的最佳节点是 node1，发送给 APIserver。
10、APIserver 将要部署在 node1 的计划存储到 etcd 中。
11、APIserver 读取 etcd 中的部署计划，通知 node1 节点的 kubelet 部署容器
12、kubelet 根据指令部署 nginx 容器，kube-proxy 为 nginx 容器创建网桥
13、容器网桥部署完成后，kubelet 通知 APIserver 部署工作完成。
14、APIserver 将部署状态存储到 etcd 当中，同时通知 controller manager(控制器）有新活了
15、controller manager 向 APIserver 要需要监控容器的数据
16、APIserver 找 etcd 读取相应数据，同时通知 kubelet 要源源不断发送监控的数据
17、APIserver 将 kubelet 发送来的数据存储到 etcd 当中
18、APIserver 将 etcd 的数据返回给 controller manager
19、controller manager 根据数据计算判断容器是否存在或健康

7、容器和主机部署应用的区别是什么？

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。

8、说一下 pod 的生命周期？



1、启动包括初始化容器的任何容器之前先创建 pause 基础容器，它初始化 Pod 环境并为后续加入的容器提供共享的名称空间。
2、按顺序以串行的方式运行用户定义的各个初始化容器进行 Pod 环境初始化；任何一个初始化容器运行失败都将导致 Pod 创建失败，并按其 restartPolicy 的策略进行处理，默认为重启。
3、等待所有容器初始化成功完成后，启动业务容器，多容器 Pod 环境中，此步骤会并行启动所有业务容器。他们各自按其自定义展开其生命周期；容器启动的那一刻会同时运行业务容器上定义的 PostStart 钩子事件，该步骤失败将导致相关容器被重启。
4、运行容器启动健康状态监测（startupProbe），判断容器是否启动成功；该步骤失败，同样参照 restartPolicy 定义的策略进行处理；未定义时，默认状态为 Success。
5、容器启动成功后，定期进行存活状态监测（liveness）和就绪状态监测（readiness）；存活监测状态失败将导致容器重启，而就绪状态监测失败会是的该容器从其所属的 Service 对象的可用端点列表中移除。
6、终止 Pod 对象时，会想运行 preStop 钩子事件，并在宽限期（termiunationGracePeriodSeconds）结束后终止主容器，宽限期默认为 30 秒。

#简述
1、创建 pod，并调度到合适节点
2、创建 pause 基础容器，提供共享名称空间
3、串行业务容器容器初始化
4、启动业务容器，启动那一刻会同时运行主容器上定义的 Poststart 钩子事件
5、健康状态监测，判断容器是否启动成功
6、持续存活状态监测、就绪状态监测
7、结束时，执行 prestop 钩子事件
8、终止容器

9、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。

10、说一下 PostStart、PreStop 钩子？

PostStart ：在容器创建后立即执行。但是，并不能保证钩子将在容器 ENTRYPOINT 之前运行，因为没有参数传递给处理程序。 主要用于资源部署、环境准备等。不过需要注意的是如果钩子花费时间过长以及于不能运行或者挂起，容器将不能达到 Running 状态。

容器启动后执行，注意由于是异步执行，它无法保证一定在 ENTRYPOINT 之后运行。如果失败，容器会被杀死，并根据 RestartPolicy 决定是否重启

PreStop ：在容器终止前立即被调用。它是阻塞的，意味着它是同步的，所以它必须在删除容器的调用出发之前完成。主要用于优雅关闭应用程序、通知其他系统等。如果钩子在执行期间挂起，Pod 阶段将停留在 Running 状态并且不会达到 failed 状态

容器停止前执行，常用于资源清理。如果失败，容器同样也会被杀死

11、请你说一下 kubenetes 针对 pod 资源对象的健康监测机制，以及三种检查方式？

1.livenessProbe 探针(检查是否存活)

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（检查是否就绪）

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（检查是否启动成功）

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。

4.检查方式?

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。

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）。

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。

12、K8s 中镜像的下载策略是什么？

1、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。

13、pod 的重启策略是什么？

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 设置来进行相应的操作。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 自动启动该容器

OnFailure：当容器终止运行且退出代码不为 0 时，有 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

kubelet 重启失效容器的时间间隔以 sync-frequency 乘以 2n 来计算；例如 1、2、4、8 倍等，最长延时 5min，并且在成功重启后的 10min 后重置该时间。

Pod 的重启策略与控制方式息息相关，当前可用于管理 Pod 的控制器包括 Replicat

ionController、Job、DaemonSet 及直接通过 kubelet 管理（静态 Pod）。每种控制器对 Pod 的重启策略要求如下：

1. RC 和 DaemonSet：必须设置为 Always，需要保证该容器持续运行
2. Job 和 CronJob：OnFailure 或 Never，确保容器执行完成后不再重启。
3. kubelet：在 Pod 失效时自动重启它，不论将 RestartPolicy 设置为什么值，也不会对 Pod 进行健康检查

14、标签和标签选择器的作用是什么？

1.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。

2.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：

1）基于等值关系

如：“=”、“==”、“！=”（注：“==”也是等于的意思，yaml 文件中的 matchLabels 字段）；

2）基于集合的

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）；

15、查看标签的方式

```
kubectl get node --show-labels

kubectl get pod --show-labels
```

16、添加、修改、删除标签的命令？

```
kubectl label node node-1 app=nginx

kubectl label node node-1 app=mycat --overwrite

kubectl label node node-1 app-
```

17、删除一个 pod 集群内部会发生什么？

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 的状态是 Terminating，Kubelet 看到 Pod 标记为 Terminating 开始了关闭 Pod 的工作

1、Pod 从 service 的列表中被删除

2、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结束进程

3、进程被发送 TERM 信号（kill -14）

4、当超过优雅退出时间时，Pod 中的所有进程都很被发送 SIGKILL 信号（kill -9）

18、持久化的方式有哪些？

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。

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。

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。

19、kubernetes 认证

kubernetes 提供了多种认证方式，比如客户端证书,静态 token,静态密码文件,ServiceAccountTokens 等等。你可以同时使用一种或多种认证方式。只要通过任何一个都被认作是认证通过。

20、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。

21、kube-scheduler 工作原理，多少节点对外提供服务？

根据各种调度算法将 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。例如，如果某节点的资源不足或者不满足预选策略的条件如“Node 的 label 必须与 Pod 的 Selector 一致”时则无法通过预选。

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。

22、集群使用的网络方案，pod 如何和 node 网络通信的？

Flannel：使用 vxlan 技术为各节点创建一个可以互通的 Pod 网络，使用的端口为 UDP 8472（需要开放该端口，如公有云 AWS 等）。
flanneld 第一次启动时，从 etcd 获取配置的 Pod 网段信息，为本节点分配一个未使用的地址段，然后创建 flannedl.1 网络接口（也可能是其它名称，如 flannel1 等）。
flannel 将分配给自己的 Pod 网段信息写入 /run/flannel/subnet.env 文件，docker 后续使用这个文件中的环境变量设置 docker0 网桥，从而从这个地址段为本节点的所有 Pod 容器分配 IP。

23、k8s 集群节点需要关机维护，需要怎么操作？

```
# 驱逐 node 节点上 pod
$ kubectl drain k8s-node-01 --force --ignore-daemonsets
# 关机$ init 0
```

24、生产中碰到过什么问题，故障排查思路，如何解决的？

1.故障归类

Pod 状态 一直处于 Pending

Pod 状态 一直处于 Waiting

Pod 状态 一直处于 ContainerCreating

Pod 状态 处于 ImagePullBackOff

Pod 状态 处于 CrashLoopBackOff

Pod 状态 处于 Error

Pod 状态 一直处于 Terminating

Pod 状态 处于 Unknown

2.排查故障的命令

1）查看 Pod 配置是否正确

```
kubectl get pod <pod-name> -o yaml
```

2）查看 Pod 详细信息

```
kubectl describe pod <pod-name>
```

3）查看容器日志

```
kubectl logs <pod-name> [-c <container-name>]
```

3.故障问题与排查方法

1）Pod 一直处于 Pending 状态

Pending 状态

这个状态意味着，Pod 的 YAML 文件已经提交给 Kubernetes，API 对象已经被创建并保存在 Etcd 当中。但是，这个 Pod 里有些容器因为某种原因而不能被顺利创建。比如，调度不成功（可以通过 kubectl describe pod 命令查看到当前 Pod 的事件，进而判断为什么没有调度）。

可能原因

资源不足（集群内所有的 Node 都不满足该 Pod 请求的 CPU、内存、GPU 等资源）；HostPort 已被占用（通常推荐使用 Service 对外开放服务端口）。

2）Pod 一直处于 Waiting 或 ContainerCreating 状态

通过 kubectl describe pod 命令查看到当前 Pod 的事件。

可能原因

1、镜像拉取失败比如，镜像地址配置错误、拉取不了国外镜像源（gcr.io）、私有镜像密钥配置错误、镜像太大导致拉取超时（可以适当调整 kubelet 的 --image-pull-progress-deadline 和 --runtime-request-timeout 选项）等。

2、CNI 网络错误，一般需要检查 CNI 网络插件的配置，比如：无法配置 Pod 网络、无法分配 IP 地址。

3、容器无法启动，需要检查是否打包了正确的镜像或者是否配置了正确的容器参数。

4、Failed create pod sandbox，查看 kubelet 日志，原因可能是磁盘坏道（input/output error）。

3）Pod 一直处于 ImagePullBackOff 状态

通常是镜像名称配置错误或者私有镜像的密钥配置错误导致。这种情况可以使用 docker pull 来验证镜像是否可以正常拉取。

如果私有镜像密钥配置错误或者没有配置，按下面检查：

①查询 docker-registry 类型的 Secret

```
# 查看 docker-registry Secret
$ kubectl  get secrets my-secret -o yaml | grep 'dockerconfigjson:' | awk '{print $NF}' | base64 -d
```

②创建 docker-registry 类型的 Secret

```
# 首先创建一个 docker-registry 类型的 Secret
$ kubectl create secret docker-registry my-secret --docker-server=DOCKER_REGISTRY_SERVER --docker-username=DOCKER_USER --docker-password=DOCKER_PASSWORD --docker-email=DOCKER_EMAIL
```

```
# 然后在 Deployment 中引用这个 Secret
spec:
  containers:
  - name: private-reg-container
    image: <your-private-image>
  imagePullSecrets:
  - name: my-secret
```

4）Pod 一直处于 CrashLoopBackOff 状态

CrashLoopBackOff 状态说明容器曾经启动了，但又异常退出。此时可以先查看一下容器的日志。

通过命令 kubectl logs 和 kubectl logs --previous 可以发现一些容器退出的原因，比如：容器进程退出、健康检查失败退出、此时如果还未发现线索，还可以到容器内执行命令来进一步查看退出原因（kubectl exec cassandra – cat /var/log/cassandra/system.log），如果还是没有线索，那就需要 exec 登录该 Pod 所在的 Node 上，查看 Kubelet 或者 Docker 的日志进一步排查。

5）Pod 处于 Error 状态

通常处于 Error 状态说明 Pod 启动过程中发生了错误

常见原因

依赖的 ConfigMap、Secret 或者 PV 等不存在；请求的资源超过了管理员设置的限制，比如超过了 LimitRange 等；违反集群的安全策略，比如违反了 PodSecurityPolicy 等；容器无权操作集群内的资源，比如开启 RBAC 后，需要为 ServiceAccount 配置角色绑定；

6）Pod 处于 Terminating 或 Unknown 状态

从 v1.5 开始，Kubernetes 不会因为 Node 失联而删除其上正在运行的 Pod，而是将其标记为 Terminating 或 Unknown 状态。想要删除这些状态的 Pod 有三种方法：

1、从集群中删除该 Node。使用公有云时，kube-controller-manager 会在 VM 删除后自动删除对应的 Node。而在物理机部署的集群中，需要管理员手动删除 Node（如 kubectl delete node ）。

2、Node 恢复正常。Kubelet 会重新跟 kube-apiserver 通信确认这些 Pod 的期待状态，进而再决定删除或者继续运行这些 Pod。用户强制删除。用户可以执行 kubectl delete pods pod-name --grace-period=0 --force 强制删除 Pod。除非明确知道 Pod 的确处于停止状态（比如 Node 所在 VM 或物理机已经关机），否则不建议使用该方法。特别是 StatefulSet 管理的 Pod，强制删除容易导致脑裂或者数据丢失等问题。

3、Pod 行为异常，这里所说的行为异常是指 Pod 没有按预期的行为执行，比如没有运行 podSpec 里面设置的命令行参数。这一般是 podSpec yaml 文件内容有误，可以尝试使用 --validate 参数重建容器，比如：

kubectl delete pod mypod 和 kubectl create --validate -f mypod.yaml，也可以查看创建后的 podSpec 是否是对的，比如：kubectl get pod mypod -o yaml，修改静态 Pod 的 Manifest 后未自动重建，Kubelet 使用 inotify 机制检测 /etc/kubernetes/manifests 目录（可通过 Kubelet 的 --pod-manifest-path 选项指定）中静态 Pod 的变化，并在文件发生变化后重新创建相应的 Pod。但有时也会发生修改静态 Pod 的 Manifest 后未自动创建新 Pod 的情景，此时一个简单的修复方法是重启 Kubelet。

Unknown 这是一个异常状态，意味着 Pod 的状态不能持续地被 kubelet 汇报给 kube-apiserver，这很有可能是主从节点（Master 和 Kubelet）间的通信出现了问题。

25、 K8S 中 request 和 limit 区别？

requst 资源请求量

含义

容器运行时，向 k8s 节点申请的最少保障资源

cpu 的 request

cpu 的 request、limit 会反映在容器的 cgroup 参数上

内存的 request

内存的 request 不会反映在容器的 cgroup 参数上，但 limit 会。

所以容器内存即使有 request，但是在容器的 cgroup 不被采用作为限制，那么其他没有 limit 或 limit 比 request 大的容器，就会来抢占这里的内存，导致这里的内存不足，结果是 k8s 节点并未保障容器的内存 request，request - current 的内存被其他容器占用

调度规则

k8s 节点的 request 剩余总资源大于容器请求的 request 资源，容器才能被分配到该节点，否则不予调度

limit 资源上限

含义：容器在 k8s 节点上消耗的资源上限

26、你对 K8S 控制器了解过哪些？

1.Deployment

在 Deployment 对象中描述所需的状态，然后 Deployment 控制器将实际状态以受控的速率更改为所需的状态。

部署无状态应用

特点：集群之中，随机部署

2.DaemonSet

每一个节点上部署一个 Pod，删除节点自动删除对应的 POD（zabbix-agent）

特点：每一台上有且只有一台

使用场景：

运行集群的存储 daemon，

每个 Node 的日志收集，fluentd ,logtash

每个 Node 的监控程序.

3.Job

负责批量任务，仅执行一次任务， 保证批处理任务一个或多个 Pod 成功结束

4.CronJob

定时调度某个时间执行一次或循环多次执行

5.StatefulSet

作为 Controller 为 Pod 提供唯一的标识 ， 证部署和 scale 的顺序。StatefulSet 用来解决有状态服务的问题，

场景：

1、稳定的持久化存储，即 Pod 重新调度后能访问持久化数据，基于 PVC 实现，

2、稳定的网络标识，即 Pod 重新调度后 PodName 和 HostName 不变，基于 Headless Services(即没有 Cluster IP 的 Service) 来实现.

3、有序部署和有序扩展， 0-> N 下一个执行基于前一个已经 Running 或 Ready，基于 InitC 实现

4、有序删除 N->0

27、你们 K8S 监控怎么做的？

使用 prometheus

1.Kubernetes 监控什么？

1）node 节点

①Node 资源利用率

一般生产环境几十个 node，几百个 node 去监控

②Node 数量

一般能监控到 node，就能监控到它的数量了，因为它是一个实例，一个 node 能跑多少个项目，也是需要去评估的，整体资源率在一个什么样的状态，什么样的值，所以需要根据项目，跑的资源利用率，还有值做一个评估的，比如再跑一个项目，需要多少资源。

③Pods 数量（Node）

其实也是一样的，每个 node 上都跑多少 pod,不过默认一个 node 上能跑 110 个 pod，但大多数情况下不可能跑这么多，比如一个 128G 的内存，32 核 cpu,一个 java 的项目，一个分配 2G,也就是能跑 50-60 个，一般机器，pod 也就跑几十个，很少很少超过 100 个。

④资源对象状态

比如 pod，service,deployment,job 这些资源状态，做一个统计。

2）Pod

①Pod 数量（项目）

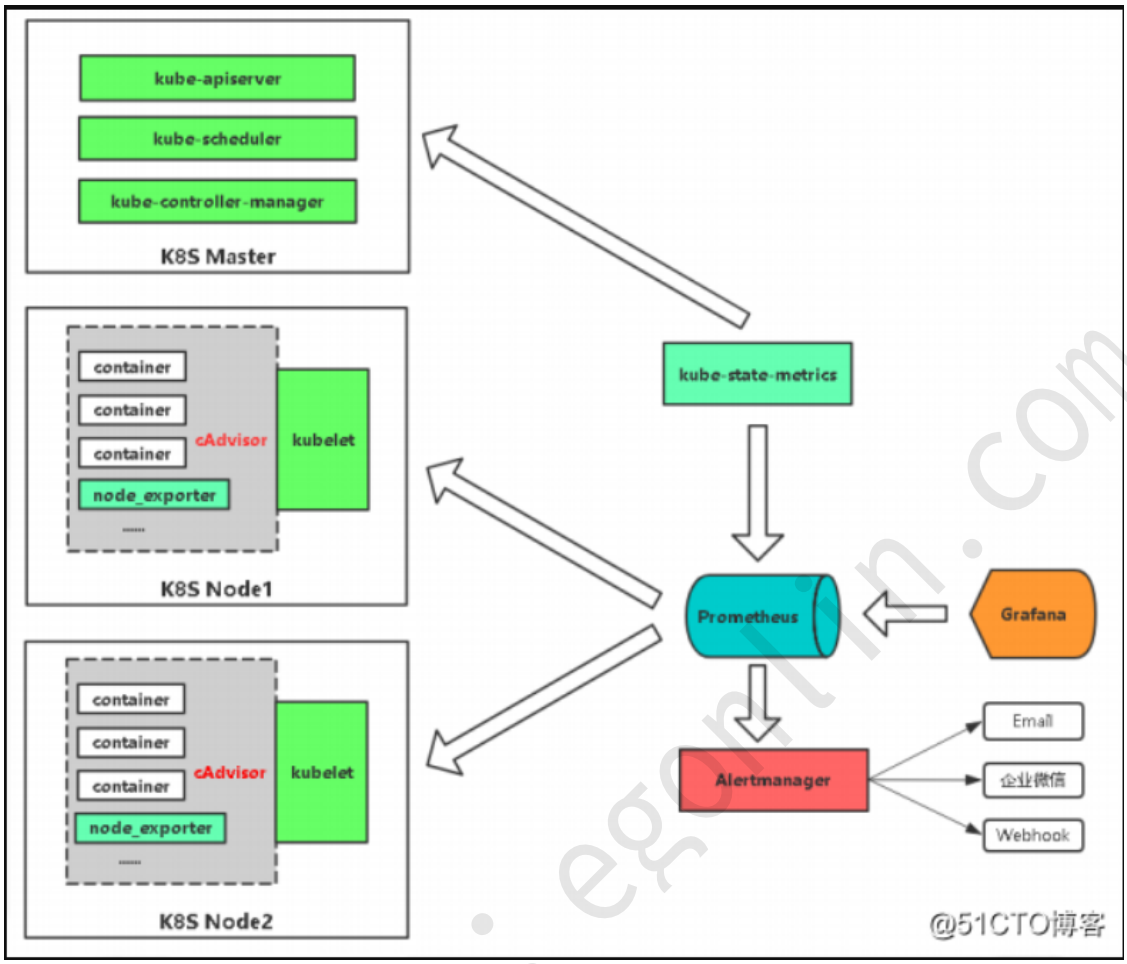
你的项目跑了多少个 pod 的数量，大概的利益率是多少，好评估一下这个项目跑了多少个资源占有多少资源，每个 pod 占了多少资源。

②容器资源使用率

每个容器消耗了多少资源，用了多少 CPU，用了多少内存

③应用程序

这个就是偏应用程序本身的指标了，这个一般在我们运维很难拿到的，所以在监控之前呢，需要开发去给你暴露出来，这里有很多客户端的集成，客户端库就是支持很多语言的，需要让开发做一些开发量将它集成进去，暴露这个应用程序的想知道的指标，然后纳入监控，如果开发部配合，基本运维很难做到这一块，除非自己写一个客户端程序，通过 shell/python 能不能从外部获取内部的工作情况，如果这个程序提供 API 的话，这个很容易做到。



2.怎么监控？

1）node 节点

如果想监控 node 的资源，就可以放一个 node_exporter,这是监控 node 资源的，node_exporter 是 Linux 上的采集器，你放上去你就能采集到当前节点的 CPU、内存、网络 IO，等待都可以采集的。

2）Pod 容器

如果想监控容器，k8s 内部提供 cAdvisor 采集器，pod 呀，容器都可以采集到这些指标，都是内置的，不需要单独部署，只知道怎么去访问这个 Cadvisor 就可以了。

3）k8s 其他资源对象

如果想监控 k8s 资源对象，会部署一个 kube-state-metrics 这个服务，它会定时的 API 中获取到这些指标，帮你存取到 Prometheus 里，要是告警的话，通过 Alertmanager 发送给一些接收方，通过 Grafana 可视化展示。

监控指标	具体实现	举例
Pod 性能	cAdvisor	容器 CPU，内存利用率
Node 性能	node-exporter	节点 CPU，内存利用率

监控指标	具体实现	举例
K8S 资源对象	kube-state-metrics	Pod/Deployment/Service

服务发现：
https://prometheus.io/docs/prometheus/latest/configuration/configuration/#kubernetes_sd_config

4）应用程序

分为四种情况

1、容器化普罗米修斯携带 metrics
2、容器化普罗米修斯无 metrics
3、物理机普罗米修斯携带 metrics
4、物理机普罗米修斯无 metrics

容器化普罗米修斯

①携带 metrics
1、创建一个 EndProints（将外部的应用的 metrics 接口的地址接入集群）
2、创建跟上条的 EndPrints 同名称的 Service
    1、只有创建 service 跟 endproints 同名称才能够关联上
    2、将 endPrints 接入的地址，通过 service 交给集群使用
3、部署 ServiceMonitor
    1、将上述接入集群的 service 中的需要被监控的 metrics 接口注入到普罗米修斯

②无 metrics
1、创建一个 EndProints（将外部的应用的 metrics 接口的地址接入集群）
2、创建跟上条的 EndPrints 同名称的 Service
    1、只有创建 service 跟 endproints 同名称才能够关联上
    2、将 endPrints 接入的地址，通过 service 交给集群使用
3、部署 exporter,构建出一个 metrics 接口（1 和 2 步合并进来）
4、部署 ServiceMonitor
    1、将上述接入集群的 service 中的需要被监控的 metrics 接口注入到普罗米修斯

物理机普罗米修斯

①携带 metrics
1、将 metrics 接口地址写入普罗米修斯配置文件
2、重启普罗米修斯

②无 metrics
1、部署 exporter,构建出一个 metrics 接口
2、将 metrics 接口地址写入普罗米修斯配置文件
3、重启普罗米修斯

28、k8s 的日志收集怎么做的？

1.日志收集级别

应用(Pod)级别
节点级别
集群级别

1）应用（Pod 级别）

Pod 级别的日志 ， 默认是输出到标准输出和标志输入，实际上跟 docker 容器的一致。
使用 kubectl logs pod-name -n namespace 查看

https://kubernetes.io/docs/reference/generated/kubectl/kubectl-commands#logs

2）节点级别

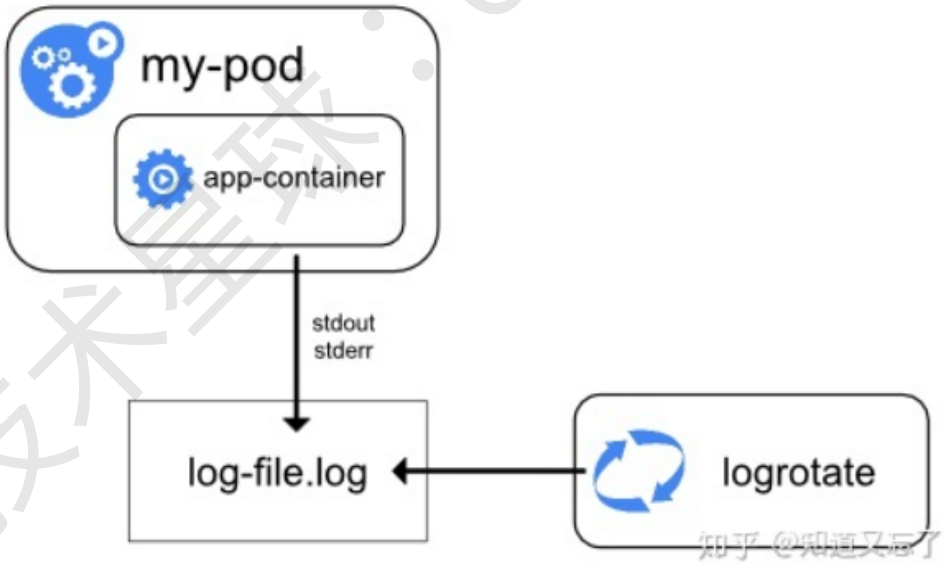
Node 级别的日志 ， 通过配置容器的 log-driver 来进行管理 ， 这种需要配合 log rotare 来进行 ， 日志超过最大限制 ， 自动进行 logrotate 操作。

log-driver:

https://docs.docker.com/config/containers/logging/configure/

logrotate

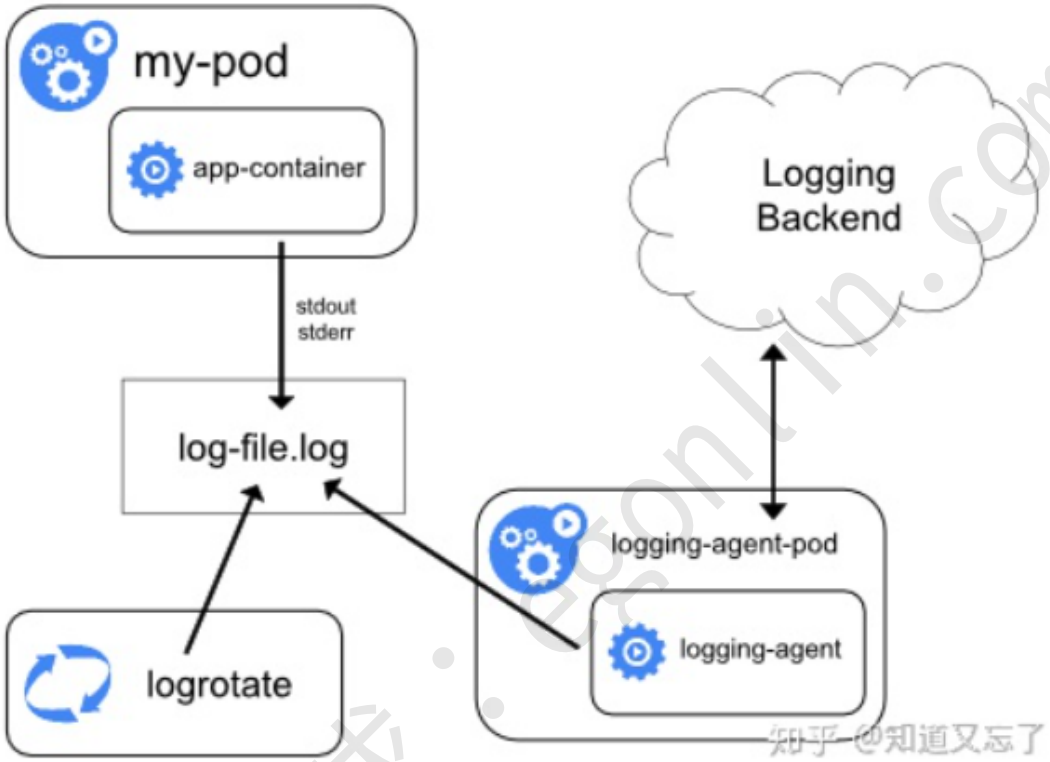
https://linux.die.net/man/8/logrotate



3）集群级别（三种）

①节点代理

在 node 级别进行日志收集。一般使用 DaemonSet 部署在每个 node 中。这种方式优点是耗费资源少，因为只需部署在节点，且对应用无侵入。缺点是只适合容器内应用日志必须都是标准输出。

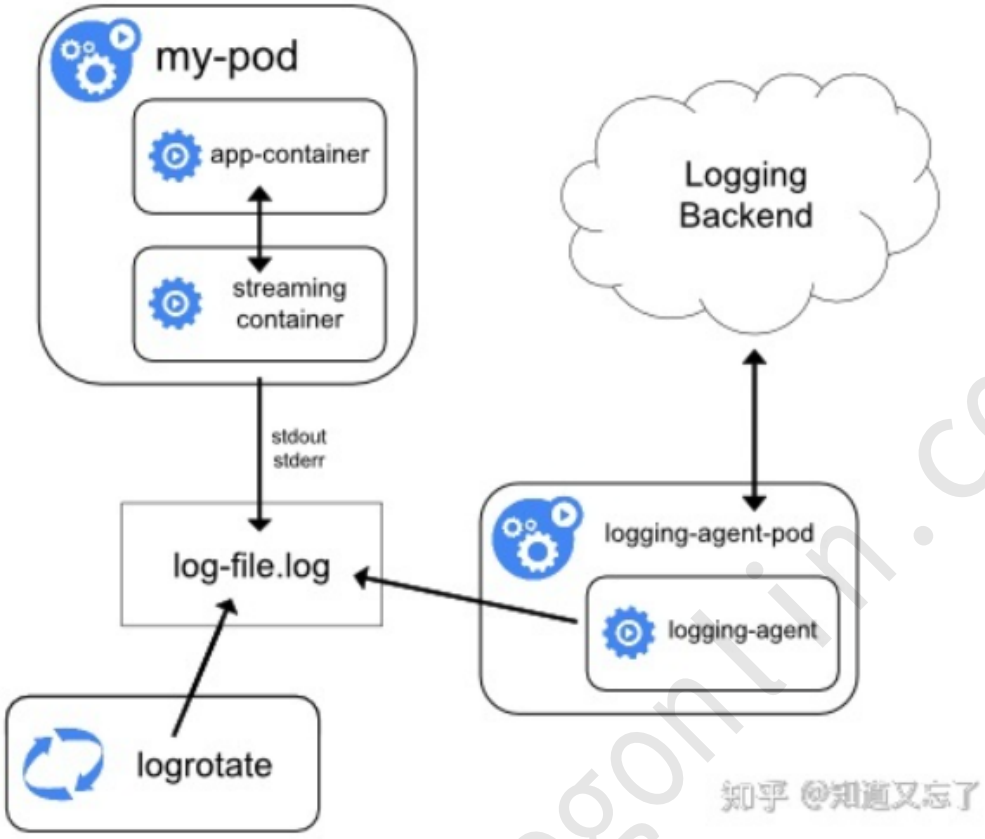


②使用 sidecar container 作为容器日志代理

也就是在 pod 中跟随应用容器起一个日志处理容器，有两种形式：

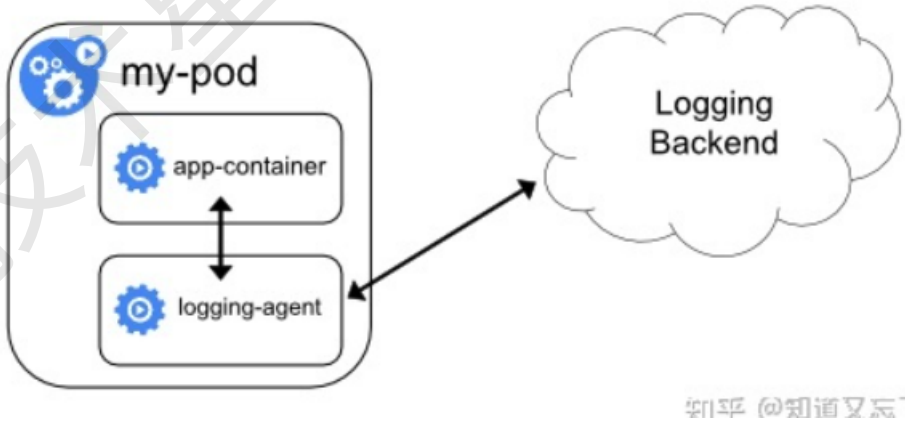
1）

直接将应用容器的日志收集并输出到标准输出（叫做 Streaming sidecar container），但需要注意的是，这时候，宿主机上实际上会存在两份相同的日志文件：一份是应用自己写入的；另一份则是 sidecar 的 stdout 和 stderr 对应的 JSON 文件。这对磁盘是很大的浪费 ，所以说，除非万不得已或者应用容器完全不可能被修改。



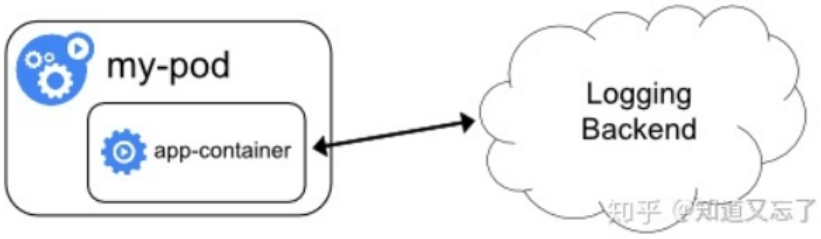
2）

每一个 pod 中都起一个日志收集 agent（比如 logstash 或 fluebtd）也就是相当于把方案一里的 logging agent 放在了 pod 里。但是这种方案资源消耗(cpu，内存)较大，并且日志不会输出到标准输出，kubectl logs 会看不到日志内容。



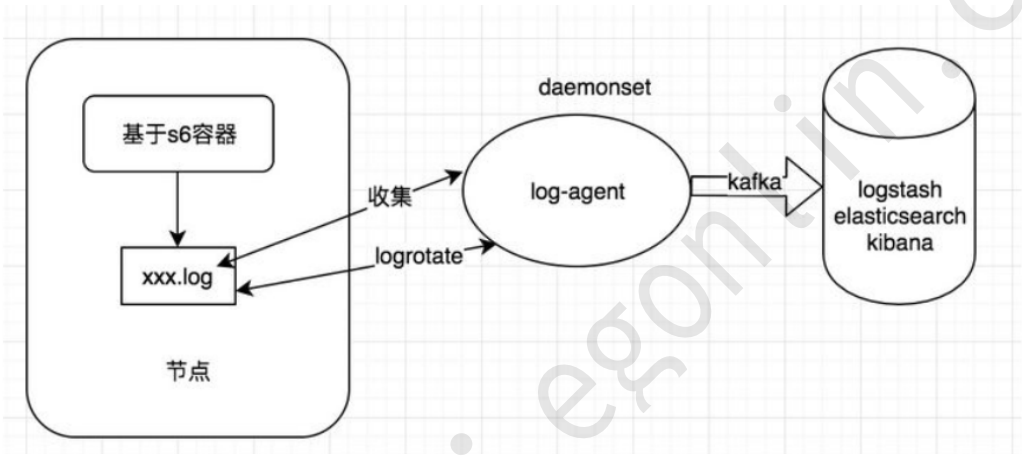
③应用容器中直接将日志推到存储后端

这种方式就比较简单了，直接在应用里面将日志内容发送到日志收集服务后端。



2、日志架构

通过上文对 k8s 日志收集方案的介绍，要想设计一个统一的日志收集系统，可以采用节点代理方式收集每个节点上容器的日志，日志的整体架构如图所示。



关于 ELK：

E:elasticsearch(数据库)

L:logstash（筛选器）

K:kibana（展示器）

elasticsearch 的运行原理：将热数据存放在内存，

架构解释

所有应用容器都是基于 s6 基底镜像的，容器应用日志都会重定向到宿主机的某个目录文件下比如/data/logs/namespace/appname/podname/log/xxxx.log

log-agent 内部 包含 filebeat ，logrotate 等工具，其中 filebeat 是作为日志文件收集的 agent

通过 filebeat 将收集的日志发送到 kafka

kafka 在讲日志发送的 es 日志存储/kibana 检索层

logstash 作为中间工具主要用来在 es 中创建 index 和消费 kafka 的消息

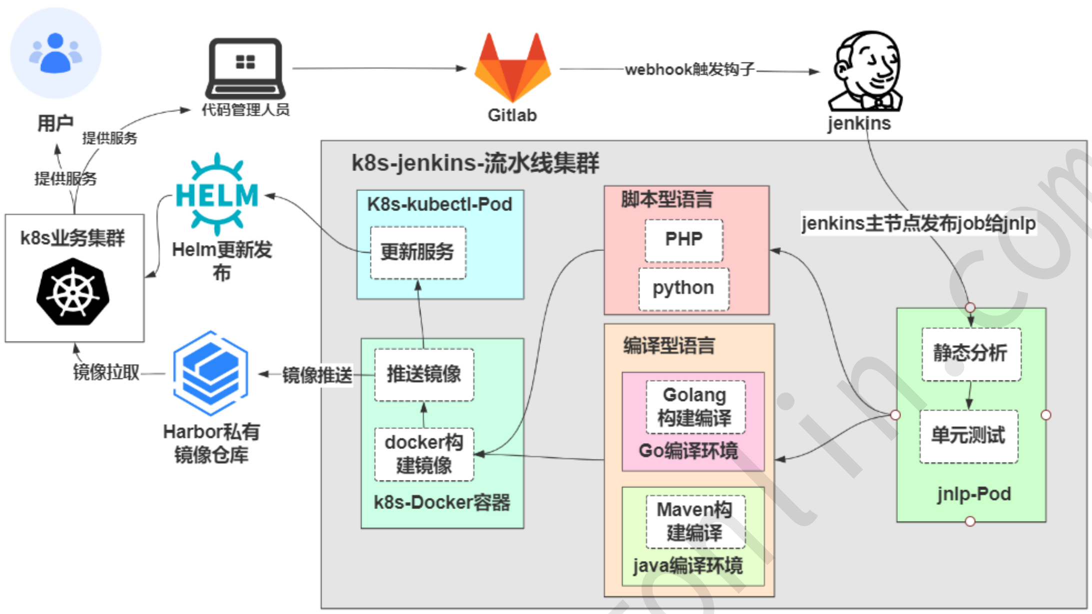
解决的问题

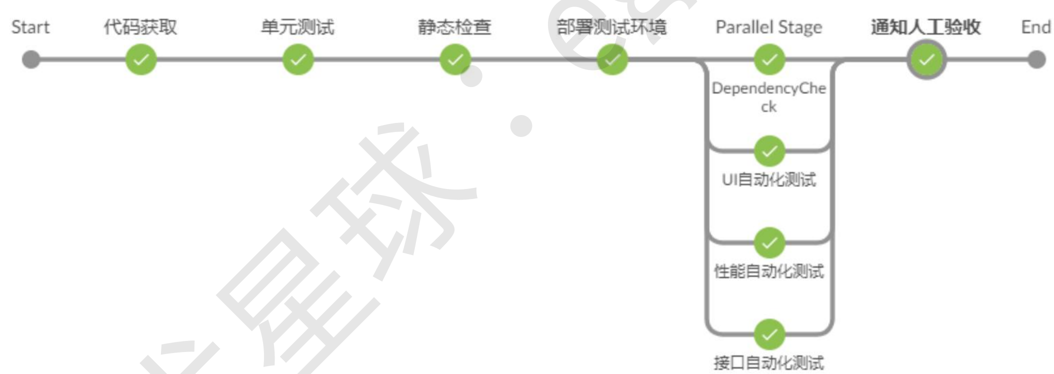
1、用户部署的新应用，如何动态更新 filebeat 配置，

2、如何保证每个日志文件都被正常的 rotate，

3、如果需要更多的功能则需要二次开发 filebeat，使 filebeat 支持更多的自定义配置。

29、k8s 环境下，怎么代码上线？





30、kubernetes 的 service 有哪些类型？

1.ClusterIP 内网

kubernetes 默认就是这种方式，是集群内部访问的方式，外部是无法访问的。其主要用于为集群内 Pod 访 问时,提供的固定访问地址,默认是自动分配地址,可使用 ClusterIP 关键字指定固定 IP。

2.nodeport 外网

NodePort 是将主机 IP 和端口跟 kubernetes 集群所需要暴露的 IP 和端口进行关联，方便其对外提供服务。内部可以通过 ClusterIP 进行访问，外部用户可以通过 NodeIP:NodePort 的方式单独访问每个 Node 上的实例。

3.LoadBalancer 弹性公网

LoadBalancer 类型的 service 是可以实现集群外部访问服务的另外一种解决方案。不过并不是所有的 k8s 集群都会支持，大多是在公有云托管集群中会支持该类型。负载均衡器是异步创建的，关于被提供的负载均衡器的信息将会通过 Service 的 status.loadBalancer 字段被发布出去。

4.ExternalName：将其他链接设置一个集群内部的别名。

ExternalName Service 是 Service 的一个特例，它没有选择器，也没有定义任何端口或 Endpoints。它的作用是返回集群外 Service 的外部别名。它将外部地址经过集群内部的再一次封装(实际上就是集群 DNS 服务器将 CNAME 解析到了外部地址上)，实现了集群内部访问即可。例如你们公司的镜像仓库，最开始是用 ip 访问，等到后面域名下来了再使用域名访问。你不可能去修改每处的引用。但是可以创建一个 ExternalName，首先指向到 ip，等后面再指向到域名。

5.headless service 域名(属于 ClusterIP)

kubernates 中还有一种 service 类型：headless serivces 功能，字面意思无 service 其实就是改 service 对外无提供 IP。一般用于对外提供域名服务的时候。

Service 与 Pod 之间的关系
service -> endprints -> pod

补充：Ingress（配合 headless 使用）

Ingress 为 Kubernetes 集群中的服务提供了入口，可以提供负载均衡、SSL 终止和基于名称的虚拟主机，在生产环境中常用的 Ingress 有 Treafik、Nginx、HAProxy、Istio 等。在 Kubernetesv 1.1 版中添加的 Ingress 用于从集群外部到集群内部 Service 的 HTTP 和 HTTPS 路由，流量从 Internet 到 Ingress 再到 Services 最后到 Pod 上，通常情况下，Ingress 部署在所有的 Node 节点上。Ingress 可以配置提供服务外部访问的 URL、负载均衡、终止 SSL，并提供基于域名的虚拟主机。但 Ingress 不会暴露任意端口或协议。

HeadLessService 实际上是属于 ClusterIP

nginx ingress : 性能强
traefik : 原生支持 k8s
istio : 服务网格，服务流量的治理

service ---> endpoints ---> pod
ingress ---> endpoints ---> pod

31、Pod 的调度算法

1.deployment 全自动调度

运行在哪个节点上完全由 master 的 scheduler 经过一系列的算法计算得出，用户无法进行干预

2.nodeselector 定向调度

1、指定 pod 调度到一些 node 上，通过设置 node 的标签和 deployment 的 nodeSelctor 属性相匹配

2、多个 node 有相同标签，scheduler 会选择一个可用的 node 进行调度

3、nodeselector 定义的标签匹配不上任何 node 上的标签， 这个 pod 是无法调度

4、k8s 给 node 预定义了一些标签， 通过 kubectl describe node xxxx 进行查看

5、用户可以使用 k8s 给 node 预定义的标签

3.NodeAffinity： node 节点亲和性

硬限制 ： 必须满足指定的规则才可以调度 pod 到 node 上

软限制 ： 优先满足指定规则，调度器会尝试调度 pod 到 node 上， 但不强求

4.PodAffinity：pod 亲和与互斥调度

根据在节点上正在运行的 pod 标签而不是节点的标签进行判断和调度

亲和： 匹配标签两个 pod 调度到同一个 node 上

互斥： 匹配标签两个 pod 不能运行在同一个 node 上

32、如何升级 k8s 新版本

1、配置 k8s 国内 yum 源

```
cat <<EOF > /etc/yum.repos.d/kubernetes.repo
[kubernetes]
name=Kubernetes
baseurl=https://mirrors.aliyun.com/kubernetes/yum/repos/kubernetes-el7-x86_64/
enabled=1
gpgcheck=0
repo_gpgcheck=0
gpgkey=https://mirrors.aliyun.com/kubernetes/yum/doc/yum-key.gpg https://mirrors.aliyun.com/kubernetes/yum/doc/rpm-package-key.gpg
EOF
```

2、查找可更新版本

```
$ yum list updates | grep 'kubeadm'
```

```
## kubeadm 中包含 kuberctl，所以 kubectl 不用升级
```

3、升级 kubeadm 版本

```
yum install -y kubeadm*
```

4、查看新版本的容器镜像版本

```
$ kubeadm config images list
```

5、更换镜像下载地址

```
MY_REGISTRY=registry.aliyuncs.com/google_containers
```

6、拉取镜像

```
## 拉取镜像
docker pull ${MY_REGISTRY}/k8s-gcr-io-kube-apiserver:v1.20.5
docker pull ${MY_REGISTRY}/k8s-gcr-io-kube-controller-manager:v1.20.5
docker pull ${MY_REGISTRY}/k8s-gcr-io-kube-scheduler:v1.20.5
docker pull ${MY_REGISTRY}/k8s-gcr-io-kube-proxy:v1.20.5
docker pull ${MY_REGISTRY}/k8s-gcr-io-etcd:3.3.10
docker pull ${MY_REGISTRY}/k8s-gcr-io-pause:3.1
docker pull ${MY_REGISTRY}/k8s-gcr-io-coredns:1.3.6
```

7、打标签

```
## 添加 Tag
docker tag ${MY_REGISTRY}/k8s-gcr-io-kube-apiserver:v1.13.4 k8s.gcr.io/
kube-apiserver:v1.13.4
docker tag ${MY_REGISTRY}/k8s-gcr-io-kube-scheduler:v1.13.4 k8s.gcr.io/
kube-scheduler:v1.13.4
docker tag ${MY_REGISTRY}/k8s-gcr-io-kube-controller-manager:v1.13.4 k8
s.gcr.io/kube-controller-manager:v1.13.4
docker tag ${MY_REGISTRY}/k8s-gcr-io-kube-proxy:v1.13.4 k8s.gcr.io/kube
-proxy:v1.13.4
docker tag ${MY_REGISTRY}/k8s-gcr-io-etcd:3.2.24 k8s.gcr.io/etcd:3.2.24
docker tag ${MY_REGISTRY}/k8s-gcr-io-pause:3.1 k8s.gcr.io/pause:3.1
docker tag ${MY_REGISTRY}/k8s-gcr-io-coredns:1.2.6 k8s.gcr.io/coredns:1.
2.6
```

8、检测 k8s 新版本

```
kubeadm upgrade plan
```

显示

```
[preflight] Running pre-flight checks.
[upgrade] Making sure the cluster is healthy:
[upgrade/config] Making sure the configuration is correct:
[upgrade/config] Reading configuration from the cluster...
[upgrade/config] FYI: You can look at this config file with 'kubectl -n
 kube-system get cm kubeadm-config -oyaml'
[upgrade] Fetching available versions to upgrade to
[upgrade/versions] Cluster version: 版本号
[upgrade/versions] kubeadm version: 版本号
I0316 13:06:25.721868   10166 version.go:94] could not fetch a Kubernet
es version from the internet: unable to get URL "https://dl.k8s.io/rele
ase/stable.txt": Get https://storage.googleapis.com/kubernetes-release/
release/stable.txt: net/http: request canceled (Client.Timeout exceeded
 while awaiting headers)
I0316 13:06:25.721890   10166 version.go:95] falling back to the local
client version: 版本号
```

```
[upgrade/versions] Latest stable version: 版本号
[upgrade/versions] Latest version in the 版本号 series: 版本号

Components that must be upgraded manually after you have upgraded the c
ontrol plane with 'kubeadm upgrade apply':
COMPONENT   CURRENT       AVAILABLE
Kubelet     5 x 版本号    版本号

Upgrade to the latest version in the v1.13 series:

COMPONENT            CURRENT   AVAILABLE
API Server           版本号     版本号
Controller Manager   版本号     版本号
Scheduler            版本号     版本号
Kube Proxy           版本号     版本号
CoreDNS              版本号     版本号
Etcd                 版本号     版本号

You can now apply the upgrade by executing the following command:

    kubeadm upgrade apply 版本号
```

9、查看各个节点状态

```
$ kubectl get nodes -o wide
```

10、升级 k8s

```
kubeadm upgrade apply 新版本号
```

成功显示

```
[upgrade/successful] SUCCESS! Your cluster was upgraded to "v1.13.4". E
njoy!

[upgrade/kubelet] Now that your control plane is upgraded, please proce
ed with upgrading your kubelets if you haven't already done so.
```

11、升级节点逐个升级

1）设置节点不可调度

```
$ kubectl drain 节点名称 --ignore-daemonsets
```

2）查找可用的 kubelet 升级包

```
$ yum list updates | grep 'kubelet'
```

3）升级 kubelet

```
$ yum install -y kubelet-新版本号
```

master 节点运行时会报错

在 master 节点上执行这个命令时，预计会出现下面这个错误，该错误是可以安全忽略的（因为 master 节点上有 static pod 运行）：

```
ode "master" already cordoned
error: pods not managed by ReplicationController, ReplicaSet, Job, Daem
onSet or StatefulSet (use --force to override): etcd-kubeadm, kube-apis
erver-kubeadm, kube-controller-manager-kubeadm, kube-scheduler-kubeadm
```

4）重启

```
# 重新加载系统配置
$ systemctl daemon-reload

# 重启 kubelet
$ systemctl restart kubelet

# 查看 kubelet 状态
$ systemctl status kubelet
```

5）设置可调度

```
$ kubectl uncordon 节点名称
```

6）查看

在对集群中所有节点的 kubelet 进行升级之后，请执行以下命令，以确认所有节点又重新变为可用状态

```
$ kubectl get nodes -o wide
```

12、故障恢复

如果 `kubeadm upgrade` 因某些原因失败并且不能回退（例如：执行过程中意外的关闭了节点实例），您可以再次运行 `kubeadm upgrade`，因为其具有幂等性，所以最终应该能够保证集群的实际状态就是您声明的所需状态。

您可以在使用 `kubeadm upgrade` 命令时带上 `–force` 来忽略某些启动时候的错误 参集群，使其从故障状态恢复。

```
kubeadm upgrade --force
```

33、K8S 哪些组件用到证书？

1.Etcd

1、Etcd 对外提供服务，要有一套 etcd server 证书

2、Etcd 各节点之间进行通信，要有一套 etcd peer 证书

3、Kube-APIserver 访问 Etcd，要有一套 etcd client 证书

2.kubernetes

4、Kube-APIserver 对外提供服务，要有一套 kube-apiserver server 证书

5、kube-scheduler、kube-controller-manager、kube-proxy、kubelet 和其他可能用到的组件，需要访问 kube-APIserver，要有一套 kube-APIserver client 证书

6、kube-controller-manager 要生成服务的 service account，要有一对用来签署 service account 的证书(CA 证书)

7、kubelet 对外提供服务，要有一套 kubelet server 证书

8、kube-APIserver 需要访问 kubelet，要有一套 kubelet client 证书

34、ingress 控制器有哪些?

Kubernetes Ingress
Nginx Ingress
Kong Ingress
Traefik Ingress
HAProxy Ingress
Istio Ingress
APISIX Ingress

Kubernetes Ingress

github.com/kubernetes/ingress-nginx

Kubernetes Ingress 的官方推荐的 Ingress 控制器，它基于 nginx Web 服务器，并补充了一组用于实现额外功能的 Lua 插件。

由于 Nginx 的普及使用，在将应用迁移到 K8S 后，该 Ingress 控制器是最容易上手的控制器，而且学习成本相对较低，如果你对控制器的能力要求不高，建议使用。

不过当配置文件太多的时候，Reload 是很慢的，而且虽然可用插件很多，但插件扩展能力非常弱。

Nginx Ingress

github.com/nginxinc/kubernetes-ingress

Nginx Ingress 是 NGINX 开发的官方版本，它基于 NGINX Plus 商业版本，NGINX 控制器具有很高的稳定性，持续的向后兼容性，没有任何第三方模块，并且由于消除了 Lua 代码而保证了较高的速度（与官方控制器相比）。

相比官方控制器，它支持 TCP/UDP 的流量转发，付费版有很广泛的附加功能，主要缺点就是缺失了鉴权方式、流量调度等其他功能。

Kong Ingress

github.com/Kong/kubernetes-ingress-controller

Kong Ingress 建立在 NGINX 之上，并增加了扩展其功能的 Lua 模块。

kong 在之前是专注于 API 网关，现在已经成为了成熟的 Ingress 控制器，相较于官方控制器，在路由匹配规则、upstream 探针、鉴权上做了提升，并且支持大量的模块插件，并且便与配置。

它提供了一些 API、服务的定义，可以抽象成 Kubernetes 的 CRD，通过 Kubernetes Ingress 配置便可完成同步状态至 Kong 集群。

Traefik Ingress

github.com/containous/traefik

traefik Ingress 是一个功能很全面的 Ingress，官方称其为：Traefik is an Edge Router that makes publishing your services a fun and easy experience.

它具有许多有用的功能：连续更新配置（不重新启动），支持多种负载平衡算法，Web UI，指标导出，支持各种协议，REST API，Canary 版本等。开箱即用的“Let's Encrypt”支持是另一个不错的功能。而且在 2.0 版本已经支持了 TCP / SSL，金丝雀部署和流量镜像/阴影等功能，社区非常活跃。

Istio Ingress

istio.io/docs/tasks/traffic-management/ingress

Istio 是 IBM，Google 和 Lyft（Envoy 的原始作者）的联合项目，它是一个全面的服务网格解决方案。它不仅可以管理所有传入的外部流量（作为 Ingress 控制器），还可以控制集群内部的所有流量。在幕后，Istio 将 Envoy 用作每种服务的辅助代理。从本质上讲，它是一个可以执行几乎所有操作的大型处理器。其中心思想是最大程度的控制，可扩展性，安全性和透明性。

借助 Istio Ingress，您可以微调流量路由，服务之间的访问授权，平衡，监控，金丝雀发布等.

不过社区现在更推荐使用 Ingress Gateways。

HAProxy Ingress

github.com/jcmoraisjr/haproxy-ingress

HAProxy 作为王牌的负载均衡器，在众多控制器中最大的优势还在负载均衡上。

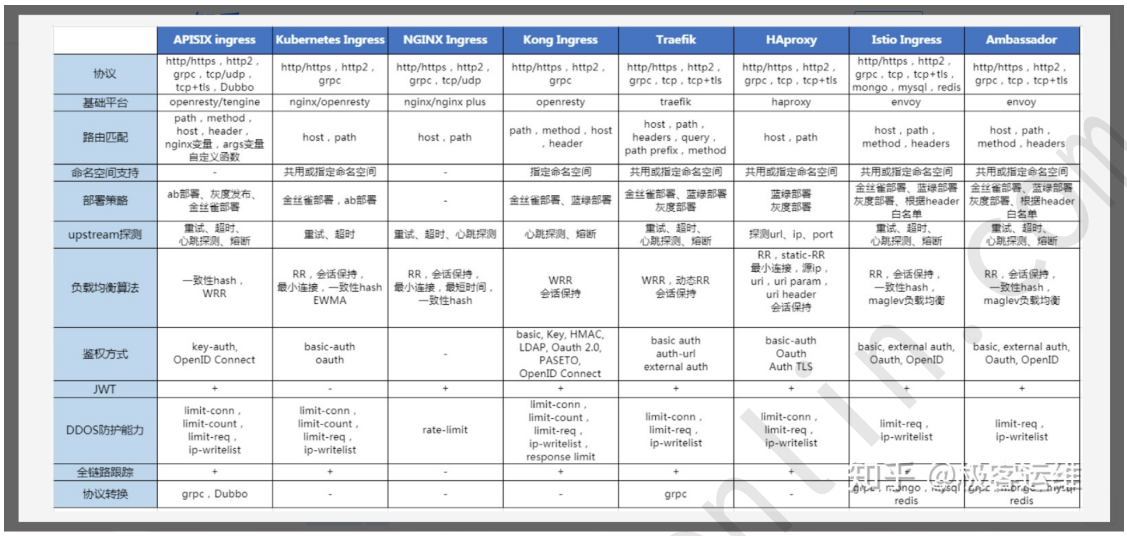
它提供了“软”配置更新（无流量丢失），基于 DNS 的服务发现，通过 API 的动态配置。HAProxy 还支持完全自定义配置文件模板（通过替换 ConfigMap）以及在其中使用 Spring Boot 函数。

APISIX Ingress

github.com/api7/ingress-controller

ApiSix Ingress 是一个新兴的 Ingress Controller，它主要对标 Kong Ingress。

它具有非常强大的路由能力、灵活的插件拓展能力，在性能上表现也非常优秀。同时，它的缺点也非常明显，尽管 APISIX 开源后有非常多的功能，但是缺少落地案例，没有相关的文档指引大家如何使用这些功能。



	APISIX ingress	Kubernetes Ingress	NGINX Ingress	Kong Ingress	Traefik	HAproxy	Istio Ingress	Ambassador
协议	http/https，http2，grpc，tcp/udp，tcp+tls，Dubbo	http/https，http2，grpc	http/https，http2，grpc，tcp/udp	http/https，http2，grpc	http/https，http2，grpc，tcp，tcp+tls	http/https，http2，grpc，tcp，tcp+tls	http/https，http2，grpc，tcp，tcp+tls，mongo，mysql，redis	http/https，http2，grpc，tcp，tcp+tls
基础平台	openresty/tengine	nginx/openresty	nginx/nginx plus	openresty	traefik	haproxy	envoy	envoy
路由匹配	path，method，host，header，nginx变量，args变量 自定义函数	host，path	host，path	path，method，host，header	host，path，headers，query，path prefix，method	host，path	host，path，method，headers	host，path，method，headers
命名空间支持	-	共用或指定命名空间	-	指定命名空间	共用或指定命名空间	共用或指定命名空间	共用或指定命名空间	共用或指定命名空间
部署策略	ab部署、灰度发布、金丝雀部署	金丝雀部署，ab部署	-	金丝雀部署、蓝绿部署	金丝雀部署、蓝绿部署 灰度部署	蓝绿部署 灰度部署	金丝雀部署、蓝绿部署 灰度部署、根据header白名单	金丝雀部署、蓝绿部署 灰度部署、根据header白名单
upstream探测	重试、超时、心跳探测、熔断	重试、超时	重试、超时、心跳探测	心跳探测、熔断	重试、超时、心跳探测、熔断	探测url、ip、port	重试、超时、心跳探测、熔断	重试、超时、心跳探测、熔断
负载均衡算法	一致性hash，WRR	RR，会话保持，最小连接，一致性hash EWMA	RR，会话保持，最小连接，最短时间，一致性hash	WRR 会话保持	WRR，动态RR 会话保持	RR，static-RR 最小连接，源ip，uri，uri param，uri header 会话保持	RR，会话保持，一致性hash，maglev负载均衡	RR，会话保持，一致性hash，maglev负载均衡
鉴权方式	key-auth, OpenID Connect	basic-auth oauth	-	basic, Key, HMAC, LDAP, Oauth 2.0, PASETO, OpenID Connect	basic auth auth-url external auth	basic-auth Oauth Auth TLS	basic, external auth, Oauth, OpenID	basic, external auth, Oauth, OpenID
JWT	+	-	+	+	+	+	+	+
DDOS防护能力	limit-conn，limit-count，limit-req，ip-writelist	limit-conn，limit-count，limit-req，ip-writelist	rate-limit	limit-conn，limit-count，limit-req，ip-writelist，response limit	limit-conn，limit-req，ip-writelist	limit-conn，limit-req，ip-writelist	limit-req，ip-writelist	limit-req，ip-writelist
全链路跟踪	+	+	-	+	+	+	envoy，mongo，mysql，grpc，thrift，redis	envoy，mongo，mysql，grpc，thrift，redis
协议转换	grpc，Dubbo	-	-	-	grpc	-		

35、上家公司 K8S 用什么方式安装的？

kubeadm，

二进制只是为了方便了解架构，一般公司不会使用

36、K8S 运维过程中遇到了哪些坑？

问这个问题主要是想试试你到底有没有工作经验

1、nfs 数据挂载时 NoRouteToHost

原因

防火墙开启

```
# 查看防火墙的状态
systemctl status firewalld.service
# 执行后，出现 绿色字体 active（running），证明防火墙开启中

# 关闭防火墙（开机的时候防火墙还是会重启）
systemctl stop firewalld.service

# 永久关闭
systemctl disable firewalld.service
```

```
# 开启
systemctl enable firewalld.service
```

2、某 node 节点 master 无法部署 job

原因：磁盘空间满了

解决办法：1、node 清理磁盘空间
2、master 重新下发任务

3、某个 node 节点一直 NotReady

原因：

宕机重启之后相关服务未启（未添加自启动），手动启动

```
systemctl restart flanneld && systemctl status flanneld
systemctl restart kube-proxy && systemctl status kube-proxy
systemctl restart kubelet && systemctl status kubelet
```

4、kubelet 服务无法启动 Failed to start Kubernetes API Server

原因：是由于系统交换内存导致，关闭即可：`swapoff -a`，重新启动就好了。

5、高版本 docker 与老版本 linux 内核不兼容，导致内容泄露

1.问题表现

```
Jun  8 03:03:28 [node hostname] kernel: [88820.247535] SLUB: Unable to
allocate memory on node -1 (gfp=0x8020)
Jun  8 03:03:28 [node hostname] kernel: [88820.247541]   cache: kmalloc
-192(33:b40c3884668600191e02b3c32efaf12bdd1e6d0a4d3869662045e4193af6c26
c), object size: 192, buffer size: 192, default order: 0, min order: 0
Jun  8 03:03:28 [node hostname] kernel: [88820.247543]   node 0: slabs:
 202, objs: 4242, free: 0
Jun  8 03:03:28 [node hostname] kernel: [88820.254869] SLUB: Unable to
allocate memory on node -1 (gfp=0x8020)
Jun  8 03:03:28 [node hostname] kernel: [88820.254873]   cache: kmalloc
-192(33:b40c3884668600191e02b3c32efaf12bdd1e6d0a4d3869662045e4193af6c26
c), object size: 192, buffer size: 192, default order: 0, min order: 0
Jun  8 03:03:28 [node hostname] kernel: [88820.254875]   node 0: slabs:
 202, objs: 4242, free: 0
```

重启后，机器仍不断打出该 log。但 `free` 查看内存，发现内存有空闲。

2.查阅资料发现内核过低

3.升级内核为较新稳定版本