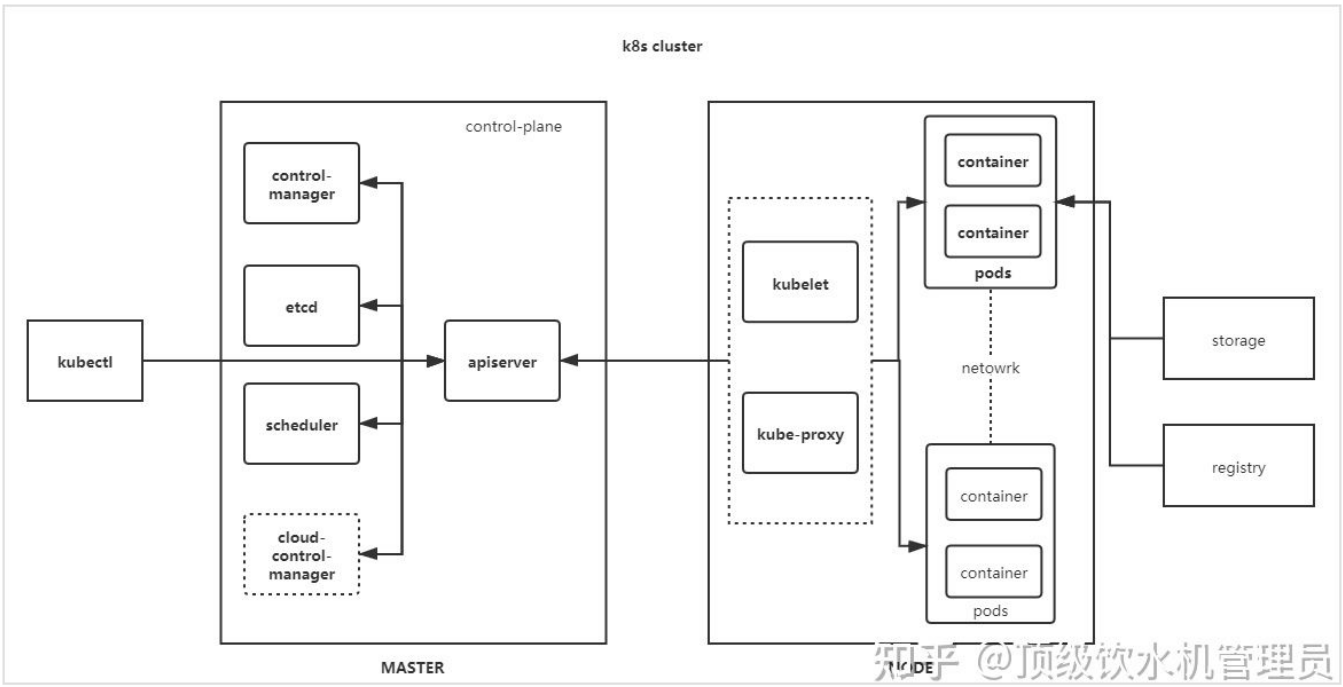
k8s架构与组件详解

一、K8s架构



k8s系统在设计是遵循c-s架构的，也就是我们图中apiserver与其余组件的交互。在生产中通常会有多个Master以实现K8s系统服务高可用。K8s集群至少有一个工作节点，节点上运行 K8s 所管理的容器化应用。

在Master通常上包括 kube-apiserver、etcd 存储、kube-controller-manager、cloud-controller-manager、kube-scheduler 和用于 K8s 服务的 DNS 服务器（插件）。这些对集群做出全局决策(比如调度)，以及检测和响应集群事件的组件集合也称为控制平面。

其实K8s官方并没有`Master`这一说，只是大多数安装工具（kubeadm）或者脚本为了架构更明了会把控制平面中的组件安装到一台机器上即Master机器，并且不会在此机器上运行用户容器。这不是强制性的，所以你也可以对将控制平面实行分布式部署，不过这样的话高可用会是一个不小的挑战。

在Node上组件包括 kubelet 、kube-porxy 以及服务于pod的容器运行时(runtime)。外部storage与registry用于为容器提供存储与镜像仓库服务。

从kubectl开始，我们来看一下K8s的基本工作流程：

1. kubectl 客户端首先将CLI命令转化为RESTful的API调用，然后发送到kube-apiserver。

2. kube-apiserver 在验证这些 API 调用后，将任务元信息并存储到etcd，接着调用 kube-scheduler 开始决策一个用于作业的Node节点。
3. 一旦 kube-scheduler 返回一个适合调度的目标节点后，kube-apiserver 就把任务的节点信息存入etcd，并创建任务。
4. 此时目标节点中的 kubelet正监听apiserver，当监听到有新任务需要调度到本节点后，kubelet通过本地runtime创建任务容器，执行作业。
5. 接着kubelet将任务状态等信息返回给apiserver存储到etcd。
6. 这样我们的任务已经在运行了，此时control-manager发挥作用保证任务一直是我们期望的状态。

二、K8s组件介绍

1、控制平面组件

kube-apiserver

API服务器为K8s集群资源操作提供唯一入口，并提供认证、授权、访问控制、API 注册和发现机制。

Kubernetes API 服务器的主要实现是 kube-apiserver。 kube-apiserver 设计上考虑了水平伸缩，也就是说，它可通过部署多个实例进行伸缩。 你可以运行 kube-apiserver 的多个实例，并在这些实例之间进行流量平衡。

etcd

etcd 是兼具一致性和高可用性的键值数据库，可以作为保存 Kubernetes 所有集群数据的后台数据库(例如 Pod 的数量、状态、命名空间等）、API 对象和服务发现细节。 在生产级k8s中etcd通常会以集群的方式存在，安全原因，它只能从 API 服务器访问。

etcd也是k8s生态的关键应用。关于 etcd 可参考 etcd 文档。

kube-scheduler

kube-scheduler 负责监视新创建、未指定运行Node的 Pods，决策出一个让pod运行的节点。

例如，如果应用程序需要 1GB 内存和 2 个 CPU 内核，那么该应用程序的 pod 将被安排在至少具有这些资源的节点上。每次需要调度 pod 时，调度程序都会运行。调度程序必须知道可用的总资源以及分配给每个节点上现有工作负载的资源。

调度决策考虑的因素包括单个 Pod 和 Pod 集合的资源需求、硬件/软件/策略约束、亲和性和反亲和性规范、数据位置、工作负载间的干扰和最后时限。

kube-controller-manager

k8s在后台运行许多不同的控制器进程，当服务配置发生更改时（例如，替换运行 pod 的镜像，或更改配置 yaml 文件中的参数），控制器会发现更改并开始朝着新的期望状态工作。

从逻辑上讲，每个控制器都是一个单独的进程， 但是为了降低复杂性，它们都被编译到同一个可执行文件，并在一个进程中运行。

控制器包括:

- 节点控制器（Node Controller）: 负责在节点出现故障时进行通知和响应
- 任务控制器（Job controller）: 监测代表一次性任务的 Job 对象，然后创建 Pods 来运行这些任务直至完成
- 端点控制器（Endpoints Controller）: 填充端点(Endpoints)对象(即加入 Service 与 Pod)
- 服务帐户和令牌控制器（Service Account & Token Controllers）: 为新的命名空间创建默认帐户和 API 访问令牌

cloud-controller-manager

云控制器管理器使得你可以将你的集群连接到云提供商的 API 之上， 同时可以将云平台交互组件与本地集群中组件分离。

`cloud-controller-manager` 仅运行特定于云平台的控制回路。 如果我们在自己的环境中运行 Kubernetes，大多数时候非混合云环境是用不到这个组件的。

与 `kube-controller-manager` 类似， `cloud-controller-manager` 将若干逻辑上独立的 控制回路组合到同一个可执行文件中，供你以同一进程的方式运行。 你可以对其执行水平扩容（运行不止一个副本）以提升性能或者增强容错能力。

下面的控制器都包含对云平台驱动的依赖：

- 节点控制器（Node Controller）: 用于在节点终止响应后检查云提供商以确定节点是否已被删除
- 路由控制器（Route Controller）: 用于在底层云基础架构中设置路由
- 服务控制器（Service Controller）: 用于创建、更新和删除云提供商负载均衡器

2.Node中组件

节点组件在每个节点上运行，维护运行的 Pod 并提供 Kubernetes 运行环境。

kubelet

一个在集群中每个node上运行的代理。 它保证容器都 运行在 Pod 中。kubelet 定期接收新的或修改过的 pod 规范 PodSpecs（主要通过 kube-apiserver）并确保 pod 及容器健康并以所需状态运行。该组件还向 kube-apiserver 报告运行它的主机的健康状况。

kubelet 不会管理不是由 Kubernetes 创建的容器。

kube-proxy

kube-proxy 是集群中每个节点上运行的网络代理， 实现 Kubernetes 服务（Service） 概念的一部分。用于处理单个主机子网划分并向外部世界公开服务。它跨集群中的各种隔离网络将请求转发到正确的 pod/容器。

kube-proxy 维护节点上的网络规则。这些网络规则允许从集群内部或外部的网络会话与 Pod 进行网络通信。

如果操作系统提供了数据包过滤层并可用的话，kube-proxy 会通过它来实现网络规则。否则，kube-proxy 仅转发流量本身。

容器运行时（Container Runtime）

容器运行时负责创建容器运行环境。

Kubernetes 支持多个容器运行时: Docker（即将被废弃）、containerd、CRI-O以及任何实现 Kubernetes CRI (容器运行环境接口)的runtime。

三、tips

7. K8s拥有一个完整的云原生生态，是一个缤纷多彩同时又把复杂度拉满的世界。
8. k8s基础是容器，虽然docker运行时已被k8s弃用，但是学习docker依然是上手容器化最佳方式。
9. Kubernetes 官方文档https://kubernetes.io/docs/home/