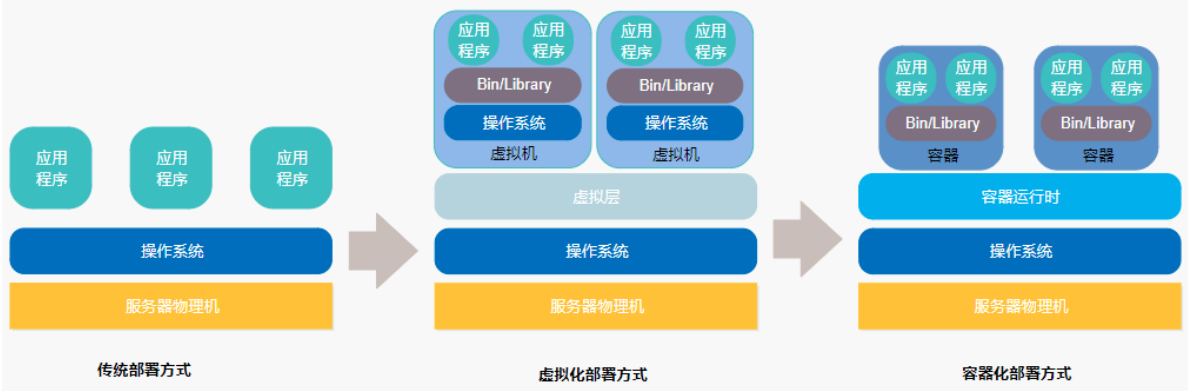
k8s 概述

author: Uncle Dragon
date: 2021-07-26

k8s 是什么

Kubernetes 是一个可移植的、可扩展的开源平台，用于管理容器化的工作负载和服务，可促进声明式配置和自动化。 Kubernetes 拥有一个庞大且快速增长的生态系统。Kubernetes 的服务、支持和工具广泛可用。

Kubernetes 这个名字源于希腊语，意为“舵手”或“飞行员”。k8s 这个缩写是因为 k 和 s 之间有八个字符的关系。 Google 在 2014 年开源了 Kubernetes 项目。Kubernetes 建立在 Google 在大规模运行生产工作负载方面拥有十几年的经验 的基础上，结合了社区中最好的想法和实践



回顾

传统部署

早期各个组织机构公司都是直接在物理机上运行自家的应用程序。无法为物理机中的应用定义资源边界，这导致资源分配问题。

举个栗子： 在物理服务器上运行了多个程序，就可能会出现其中一个程序占用了大部分物理资源，结果就可能会导致其他程序的性能下降甚至是无法正常运行。

所以早期为避免上述问题，只能是一个服务器上运行一个应用服务，但是由于资源利用不足，无法扩展，并且需要维护很多的物理服务器，成本提高了不少。

虚拟化部署

应用部署进化到这一时期，解决了资源边界的问题。利用虚拟化技术你可以在一台物理机上运行出多个虚拟机(VM)。虚拟化允许应用在虚拟机之间隔离。并提供了一定程度的安全性，(一个应用无法随意访问运行在另一个虚拟机的应用的资源)

虚拟化技术提升了物理机的资源利用率，并且可以轻松的添加或更新应用程序，一定程度上实现了较好的可伸缩性，降低了硬件成本。

每个虚拟机(VM) 都是一台完整的计算机，在虚拟化硬件之上运行所有的组件，包括自己的操作系统。

容器化部署

容器类似于VM,但是它具备一定的隔离属性，在应用程序之间是共享操作系统的。因此，容器是轻量级的。

容器也具有自己的文件系统、CPU、内存、进程空间等。由于它们与底层基础架构隔离，所以他们可以方便的在不同的云和操作系统间迁移。

容器因为具有诸多好处而变得流行起来，下面列出一些它的优势：

- 敏捷应用程序的创建和部署：与 使用 VM 相比，提高了应用的部署更加简单，提高了应用发布的效率。
- 持续开发、集成和部署：通过快速简单的回滚（由于镜像不可变性），支持可靠且频繁的 容器镜像构建和部署。
- 关注开发与运维的分离：在构建/发布时而不是在部署时创建应用程序容器镜像， 从而将应用程序与基础架构分离。
- 可观察性：不仅可以显示操作系统级别的信息和指标，还可以显示应用程序的运行状况和其他指标信号。
- 跨开发、测试和生产的环境一致性：在便携式计算机上与在云中相同地运行。
- 跨云和操作系统发行版本的可移植性：可在 Ubuntu、RHEL、CoreOS、本地、Google Kubernetes Engine 和其他任何地方运行。
- 以应用程序为中心的管理：提高抽象级别，从在虚拟硬件上运行 OS 到使用逻辑资源在 OS 上运行应用程序。
- 松散耦合、分布式、弹性、解放的微服务：应用程序被分解成较小的独立部分， 并且可以动态部署和管理 - 而不是在一台大型单机上整体运行。
- 资源隔离：可预测的应用程序性能。
- 资源利用：高效率和高密度

为甚么要用 k8s

容器是打包和运行应用程序的好方式。在生产环境中，你需要管理运行应用程序的容器，并确保不会停机。 例如，如果一个容器发生故障，则需要启动另一个容器。如果系统处理此行为，会不会更容易？

这就是 Kubernetes 来解决这些问题的方法！ Kubernetes 为你提供了一个可弹性运行分布式系统的框架。 Kubernetes 会满足你的扩展要求、故障转移、部署模式等。 例如，Kubernetes 可以轻松管理系统的 Canary 部署。

Kubernetes 为你提供：

- **服务发现和负载均衡**

  Kubernetes 可以使用 DNS 名称或自己的 IP 地址公开容器，如果进入容器的流量很大， Kubernetes 可以负载均衡并分配网络流量，从而使部署稳定。

- **存储编排**

  Kubernetes 允许你自动挂载你选择的存储系统，例如本地存储、公共云提供商等。

- **自动部署和回滚**

  你可以使用 Kubernetes 描述已部署容器的所需状态，它可以以受控的速率将实际状态 更改为期望状态。例如，你可以自动化 Kubernetes 来为你的部署创建新容器， 删除现有容器并将它们的所有资源用于新容器。

- **自动完成装箱计算**

  Kubernetes 允许你指定每个容器所需 CPU 和内存（RAM）。 当容器指定了资源请求时，Kubernetes 可以做出更好的决策来管理容器的资源。

- **自我修复**

  Kubernetes 重新启动失败的容器、替换容器、杀死不响应用户定义的 运行状况检查的容器，并且在准备好服务之前不将其通告给客户端。

- **密钥与配置管理**

  Kubernetes 允许你存储和管理敏感信息，例如密码、OAuth 令牌和 ssh 密钥。 你可以在不重建容器镜像的情况下部署和更新密钥和应用程序配置，也无需在堆栈配置中暴露密钥。

k8s 不是什么

Kubernetes 不是传统的、包罗万象的 PaaS（平台即服务）系统。 由于 Kubernetes 在容器级别而不是在硬件级别运行，它提供了 PaaS 产品共有的一些普遍适用的功能， 例如部署、扩展、负载均衡、日志记录和监视。 但是，Kubernetes 不是单体系统，默认解决方案都是可选和可插拔的。 Kubernetes 提供了构建开发人员平台的基础，但是在重要的地方保留了用户的选择和灵活性。

Kubernetes：

- 不限制支持的应用程序类型。 Kubernetes 旨在支持极其多种多样的工作负载，包括无状态、有状态和数据处理工作负载。 如果应用程序可以在容器中运行，那么它应该可以在 Kubernetes 上很好地运行。
- 不部署源代码，也不构建你的应用程序。 持续集成(CI)、交付和部署（CI/CD）工作流取决于组织的文化和偏好以及技术要求。
- 不提供应用程序级别的服务作为内置服务，例如中间件（例如，消息中间件）、 数据处理框架（例如，Spark）、数据库（例如，mysql）、缓存、集群存储系统 （例如，Ceph）。这样的组件可以在 Kubernetes 上运行，并且/或者可以由运行在 Kubernetes 上的应用程序通过可移植机制（例如， 开放服务代理来访问。
- 不要求日志记录、监视或警报解决方案。 它提供了一些集成作为概念证明，并提供了收集和导出指标的机制。
- 不提供或不要求配置语言/系统（例如 jsonnet），它提供了声明性 API， 该声明性 API 可以由任意形式的声明性规范所构成。
- 不提供也不采用任何全面的机器配置、维护、管理或自我修复系统。
- 此外，Kubernetes 不仅仅是一个编排系统，实际上它消除了编排的需要。 编排的技术定义是执行已定义的工作流程：首先执行 A，然后执行 B，再执行 C。 相比之下，Kubernetes 包含一组独立的、可组合的控制过程， 这些过程连续地将当前状态驱动到所提供的所需状态。 如何从 A 到 C 的方式无关紧要，也不需要集中控制，这使得系统更易于使用 且功能更强大、系统更健壮、更为弹性和可扩展。

k8s 组件

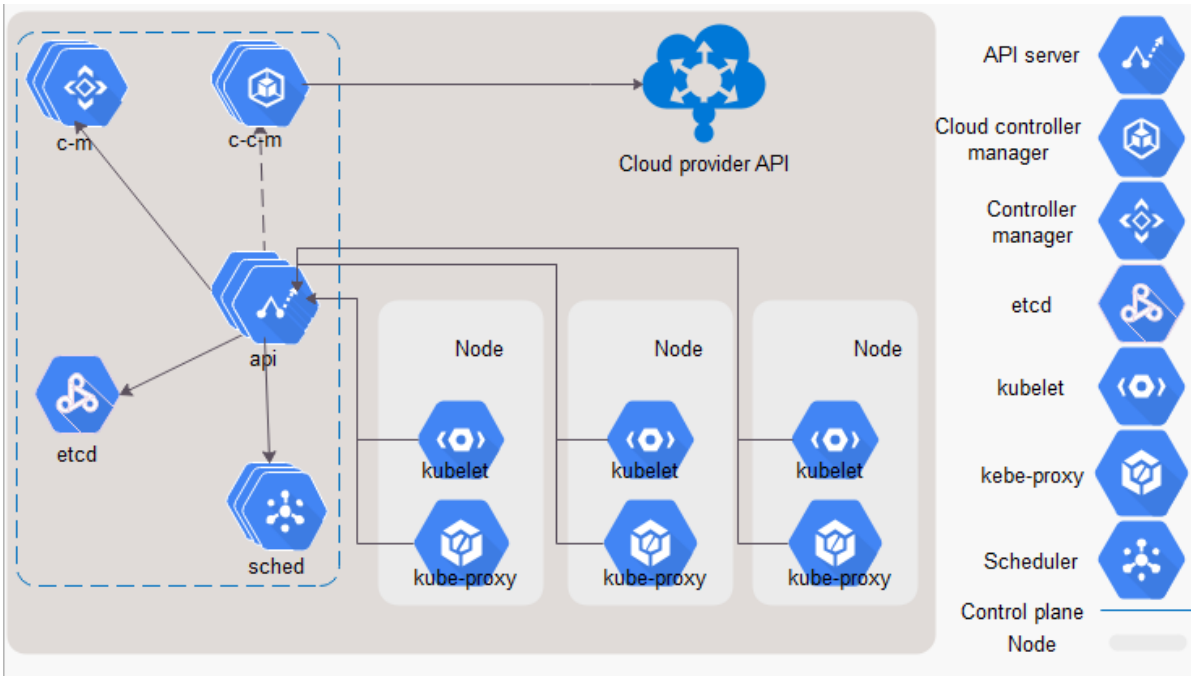
当你部署完 Kubernetes, 即拥有了一个完整的集群。

一个 Kubernetes 集群由一组被称作节点的机器组成。这些节点上运行 Kubernetes 所管理的容器化应用。集群具有至少一个工作节点。

工作节点托管作为应用负载的组件的 Pod 。控制平面管理集群中的工作节点和 Pod 。 为集群提供故障转移和高可用性，这些控制平面一般跨多主机运行，集群跨多个节点运行。

本小节概述了交付正常运行的 Kubernetes 集群所需的各种组件。

这张图表展示了包含所有相互关联组件的 Kubernetes 集群



Control Plane Components

`Control Plane Components`对集群做出全局决策(比如调度)，以及检测和响应集群事件（例如，当不满足部署的 `replicas` 字段时，启动新的 pod。

控制平面组件可以在集群中的任何节点上运行。 然而，为了简单起见，设置脚本通常会在同一个计算机上启动所有控制平面组件， 并且不会在此计算机上运行用户容器。

kube-apiserver

API 服务器是 Kubernetes `control-plane`的组件， 该组件公开了 Kubernetes API。 API 服务器是 Kubernetes 控制面的前端。

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，并在这些实例之间平衡流量。

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。

您的 Kubernetes 集群的 etcd 数据库通常需要有个备份计划。

kube-scheduler

控制平面组件，负责监视新创建的、未指定运行节点（node) 的 [Pods]，选择节点让 Pod 在上面运行。

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

运行控制器(controller)进程的控制平面组件。

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。

这些控制器包括:

- 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

云控制器管理器是指嵌入特定云的控制逻辑的 `control-plane` 组件。 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， 并将与该云平台交互的组件同与你的集群交互的组件分离开来。

`cloud-controller-manager` 仅运行特定于云平台的控制回路。 如果你在自己的环境中运行 Kubernetes，或者在本地计算机中运行学习环境， 所部署的环境中不需要云控制器管理器。

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

Node 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

一个在集群中每个节点（node）上运行的代理。 它保证容器（containers）都 运行在 Pod 中。

kubelet 接收一组通过各类机制提供给它的 PodSpecs，确保这些 PodSpecs 中描述的容器处于运行状态且健康。 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。

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）

容器运行环境是负责运行容器的软件。

Kubernetes 支持多个容器运行环境: Docker、containerd、CRI-O 以及任何实现 Kubernetes CRI (容器运行环境接口)。

插件（Addons）

插件使用 Kubernetes 资源（DaemonSet、Deployment)等实现集群功能。 因为这些插件提供集群级别的功能，插件中命名空间域的资源属于 `kube-system` 命名空间。

下面描述众多插件中的几种:

DNS

尽管其他插件都并非严格意义上的必需组件，但几乎所有 Kubernetes 集群都应该 有集群 DNS， 因为很多示例都需要 DNS 服务。

集群 DNS 是一个 DNS 服务器，和环境中的其他 DNS 服务器一起工作，它为 Kubernetes 服务提供 DNS 记录。

Kubernetes 启动的容器自动将此 DNS 服务器包含在其 DNS 搜索列表中。

Web 界面（仪表盘）

Dashboard 是 Kubernetes 集群的通用的、基于 Web 的用户界面。 它使用户可以管理集群中运行的应用程序以及集群本身并进行故障排除。

容器资源监控

容器资源监控 将关于容器的一些常见的时间序列度量值保存到一个集中的数据库中，并提供用于浏览这些数据的界面。

集群层面日志

集群层面日志机制负责将容器的日志数据 保存到一个集中的日志存储中，该存储能够提供搜索和浏览接口

k8s API

Kubernetes control-plane 的核心是 kube-apiserver。 kube-apiserver 负责提供 HTTP API，以供用户、集群中的不同部分和集群外部组件相互通信。

Kubernetes API 使你可以查询和操纵 Kubernetes API 中对象（例如：Pod、Namespace、ConfigMap 和 Event）的状态。

大部分操作都可以通过 kubectl 命令行接口或 类似 kubeadm 这类命令行工具来执行， 这些工具在背后也是调用 API。不过，你也可以使用 REST 调用来访问这些 API。

如果你正在编写程序来访问 Kubernetes API，可以考虑使用 客户端库之一。

k8s 对象

理解 Kubernetes 对象

在 Kubernetes 系统中，*Kubernetes* 对象 是持久化的实体。 Kubernetes 使用这些实体去表示整个集群的状态。特别地，它们描述了如下信息：

- 哪些容器化应用在运行（以及在哪些节点上）
- 可以被应用使用的资源
- 关于应用运行时表现的策略，比如重启策略、升级策略，以及容错策略

Kubernetes 对象是 “目标性记录” —— 一旦创建对象，Kubernetes 系统将持续工作以确保对象存在。 通过创建对象，本质上是在告知 Kubernetes 系统，所需要的集群工作负载看起来是什么样子的， 这就是 Kubernetes 集群的 **期望状态（Desired State）**。

操作 Kubernetes 对象 —— 无论是创建、修改，或者删除 —— 需要使用 Kubernetes API。 比如，当使用 `kubectl` 命令行接口时，CLI 会执行必要的 Kubernetes API 调用，也可以在程序中使用客户端库直接调用 Kubernetes API。

对象规约（Spec）与状态（Status）

几乎每个 Kubernetes 对象包含两个嵌套的对象字段，它们负责管理对象的配置： 对象 `spec`（规约） 和 对象 `status`（状态） 。 对于具有 `spec` 的对象，你必须在创建对象时设置其内容，描述你希望对象所具有的特征： 期望状态（*Desired State*） 。

`status` 描述了对象的 当前状态（*Current State*），它是由 Kubernetes 系统和组件 设置并更新的。在任何时刻，Kubernetes 控制平面 都一直积极地管理着对象的实际状态，以使之与期望状态相匹配。

例如，Kubernetes 中的 Deployment 对象能够表示运行在集群中的应用。 当创建 Deployment 时，可能需要设置 Deployment 的 `spec`，以指定该应用需要有 3 个副本运行。 Kubernetes 系统读取 Deployment 规约，并启动我们所期望的应用的 3 个实例 —— 更新状态以与规约相匹配。 如果这些实例中有的失败了（一种状态变更），Kubernetes 系统通过执行修正操作 来响应规约和状态间的不一致 —— 在这里意味着它会启动一个新的实例来替换。

关于对象 spec、status 和 metadata 的更多信息，可参阅 Kubernetes API 约定。

描述 Kubernetes 对象

创建 Kubernetes 对象时，必须提供对象的规约，用来描述该对象的期望状态， 以及关于对象的一些基本信息（例如名称）。 当使用 Kubernetes API 创建对象时（或者直接创建，或者基于`kubectl`）， API 请求必须在请求体中包含 JSON 格式的信息。 **大多数情况下，需要在 .yaml 文件中为 `kubectl` 提供这些信息。** `kubectl` 在发起 API 请求时，将这些信息转换成 JSON 格式。

这里有一个 `.yaml` 示例文件，展示了 Kubernetes Deployment 的必需字段和对象规约

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 2 # tells deployment to run 2 pods matching the template
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
```

```
19          - containerPort: 80
```

使用类似于上面的 `.yaml` 文件来创建 Deployment的一种方式是使用 `kubectl` 命令行接口（CLI）中的 `kubectl apply` 命令， 将 `.yaml` 文件作为参数。下面是一个示例：

```
$ kubectl apply -f
https://k8s.io/examples/application/deployment.yaml --record
deployment.apps/nginx-deployment created
```

必需字段

在想要创建的 Kubernetes 对象对应的 `.yaml` 文件中，需要配置如下的字段：

- `apiVersion` - 创建该对象所使用的 Kubernetes API 的版本
- `kind` - 想要创建的对象的类别
- `metadata` - 帮助唯一性标识对象的一些数据，包括一个 `name` 字符串、UID 和可选的 `namespace`

你也需要提供对象的 `spec` 字段。 对象 `spec` 的精确格式对每个 Kubernetes 对象来说是不同的，包含了特定于该对象的嵌套字段。 Kubernetes API 参考 能够帮助我们找到任何我们想创建的对象的 spec 格式。 例如，可以从 core/v1 PodSpec 查看 `Pod` 的 `spec` 格式， 并且可以从 apps/v1 DeploymentSpec 查看 `Deployment` 的 `spec` 格式。