Kubernetes

Kubernetes 是用于自动部署，扩展和管理容器化应用程序的开源系统。
它将组成应用程序的容器组合成逻辑单元，以便于管理和服务发现。Kubernetes 源自Google 15 年生产环境的运维经验，同时凝聚了社区的最佳创意和实践。

Kubernetes 特性

- 自动化上线和回滚
  Kubernetes 会分步骤地将针对应用或其配置的更改上线，同时监视应用程序运行状况以确保你不会同时终止所有实例。如果出现问题，Kubernetes 会为你回滚所作更改。你应该充分利用不断成长的部署方案生态系统。

- 服务发现与负载均衡
  无需修改你的应用程序即可使用陌生的服务发现机制。Kubernetes 为容器提供了自己的 IP 地址和一个 DNS 名称，并且可以在它们之间实现负载均衡。

- 存储编排
  自动挂载所选存储系统，包括本地存储、诸如 GCP 或 AWS 之类公有云提供商所提供的存储或者诸如 NFS、iSCSI、Gluster、Ceph、Cinder 或 Flocker 这类网络存储系统。

- Secret 和配置管理
  部署和更新 Secrets 和应用程序的配置而不必重新构建容器镜像，且 不必将软件堆栈配置中的秘密信息暴露出来。

- 自动装箱
  根据资源需求和其他约束自动放置容器，同时避免影响可用性。 将关键性的和尽力而为性质的工作负载进行混合放置，以提高资源利用率并节省更多资源。

- 批量执行
  除了服务之外，Kubernetes 还可以管理你的批处理和 CI 工作负载，在期望时替换掉失效的容器。

- IPv4/IPv6 双协议栈
  为 Pod 和 Service 分配 IPv4 和 IPv6 地址

- 水平扩缩
  使用一个简单的命令、一个 UI 或基于 CPU 使用情况自动对应用程序进行扩缩。

- 自我修复
  重新启动失败的容器，在节点死亡时替换并重新调度容器，杀死不响应用户定义的健康检查的容器，并且在它们准备好服务之前不会将它们公布给客户端。

- 为扩展性设计
  无需更改上游源码即可扩展你的 Kubernetes 集群。

Kubernetes Pods

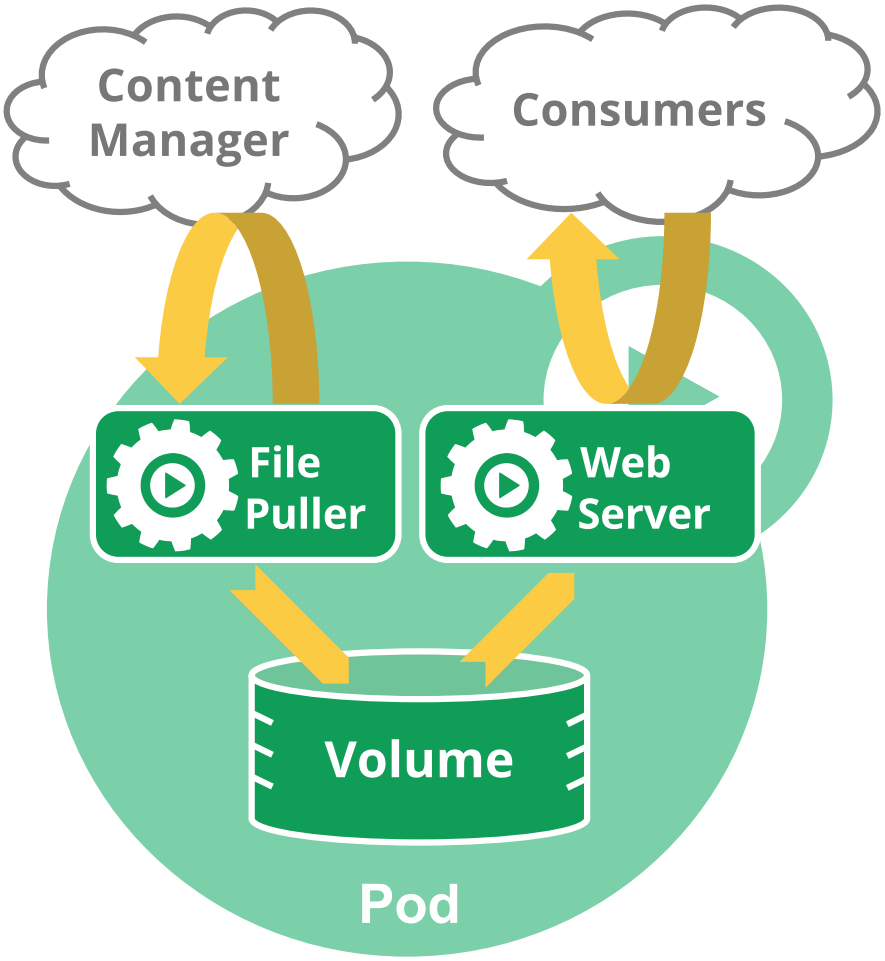
`Pod` 是可以在 Kubernetes 中创建和管理的、最小的可部署的计算单元。

`Pod` 是一组（一个或多个） 容器； 这些容器共享存储、网络、以及怎样运行这些容器的声明。 通常不需要直接创建 Pod，会使用 `Deployment` 或者 `Job` 这类工作负载资源 来创建 `Pod` 。如果 `Pod` 需要跟踪状态，可以考虑 `StatefulSet` 资源。

Kubernetes 集群中的 Pod 主要有两种用法：

- 运行单个容器的 `Pod` 。
  "每个 Pod 一个容器"模型是最常见的 Kubernetes 用例； 在这种情况下，可以将 Pod 看作单个容器的包装器，并且 Kubernetes 直接管理 Pod，而不是容器。

- 运行多个协同工作的容器的 `Pod` 。
  `Pod` 可能封装由多个紧密耦合且需要共享资源的共处容器组成的应用程序。这些位于同一位置的容器可能形成单个内聚的服务单元。例如：一个容器将文件从共享卷提供给公众， 而另一个单独的“边车”（sidecar）容器则刷新或更新这些文件。 Pod 将这些容器和存储资源打包为一个可管理的实体。



每个`Pod`运行给定应用程序的单个实例。如果希望横向扩展应用程序（例如，运行多个实例 以提供更多的资源），则应该使用多个 Pod，每个实例使用一个 Pod。 在 Kubernetes 中，这通常被称为 副本（Replication）。 通常使用一种工作负载资源及其控制器 来创建和管理一组 Pod 副本。

使用 Pod

你很少在 Kubernetes 中直接创建一个个的 Pod，甚至是单实例（Singleton）的 Pod。 这是因为 Pod 被设计成了相对临时性的、用后即抛的一次性实体。 当 Pod 由你或者间接地由 控制器 创建时，它被调度在集群中的节点上运行。 Pod 会保持在该节点上运行，直到 Pod 结束执行、Pod 对象被删除、Pod 因资源不足而被 驱逐 或者节点失效为止。

Pod 和控制器

你可以使用工作负载资源来创建和管理多个 Pod。 资源的控制器能够处理副本的管理、上线，并在 Pod 失效时提供自愈能力。 例如，如果一个节点失败，控制器注意到该节点上的 Pod 已经停止工作，就可以创建替换性的 Pod。调度器会将替身 Pod 调度到一个健康的节点执行。

下面是一些管理一个或者多个 Pod 的工作负载资源的示例：

- Deployment

- StatefulSet
- DaemonSet

Pod 模版

负载资源的控制器通常使用 Pod 模板（Pod Template） 来替你创建 Pod 并管理它们。
Pod 模板是包含在工作负载对象中的规范，用来创建 `Pod` 。这类负载资源包括 `Deployment` 、 `Job` 和 `DaemonSets` 等。
工作负载的控制器会使用负载对象中的 PodTemplate 来生成实际的 Pod。 PodTemplate 是你用来运行应用时指定的负载资源的目标状态的一部分。

```
apiVersion: batch/v1
kind: Job
metadata:
  name: hello
spec:
  template:
    # 这里是 Pod 模版
    spec:
      containers:
      - name: hello
        image: busybox
        command: ['sh', '-c', 'echo "Hello, Kubernetes!" && sleep 3600']
      restartPolicy: OnFailure
    # 以上为 Pod 模版
```

修改 Pod 模版或者切换到新的 Pod 模版都不会对已经存在的 Pod 起作用。 Pod 不会直接收到模版的更新。相反， 新的 Pod 会被创建出来，与更改后的 Pod 模版匹配。

例如，Deployment 控制器针对每个 Deployment 对象确保运行中的 Pod 与当前的 Pod 模版匹配。如果模版被更新，则 Deployment 必须删除现有的 Pod，基于更新后的模版 创建新的 Pod。每个工作负载资源都实现了自己的规则，用来处理对 Pod 模版的更新。

容器探针

由 kubelet 对容器执行的定期诊断。要执行诊断，kubelet 可以执行三种动作：

1. `ExecAction` （借助容器运行时执行）
2. `TCPSocketAction` （由 kubelet 直接检测）
3. `HTTPGetAction` （由 kubelet 直接检测）