在 Kubernetes 上部署无状态应用

课程目标

- 理解 Kubernetes 的基本组件
- 理解 Pod 概念
- 理解 Service 概念和几种不同的使用方式
- 理解 Deployment 概念和如何进行滚动更新

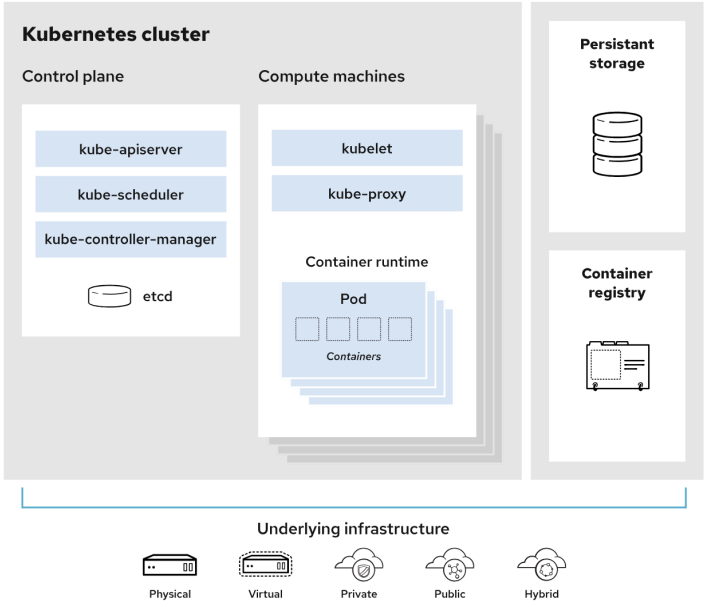
介绍

Kubernetes 的基本概念

- 集群（Cluster）：Kubernetes 集群是一组可以运行容器化应用的节点，这些节点可以是物理服务器或虚拟机。
- 节点（Node）：节点是一个工作机器，物理或虚拟的，每个节点都有一个 Kubelet 代理，负责管理节点并与 Kubernetes 主节点通信。
- Pod：Pod 是 Kubernetes 的最小部署单位，一个或多个共享存储和网络的容器的集合。
- 服务（Service）：服务是对运行在一组 Pods 中的应用服务的抽象，提供一种将应用作为网络服务对外访问的方式。
- 部署（Deployment）：Deployment 描述期望应用达到的状态，如运行的 Pod 数量，Deployment 控制器更改实际状态以匹配期望状态。
- 命名空间（Namespace）：命名空间是一种隔离机制，支持在同一集群中运行多个用户或应用程序，以隔离他们的资源。
- 卷（Volume）：卷是一种数据的存储方式，能够持久化存储数据，以便在 Pod 生命周期结束后依然存在。

介绍

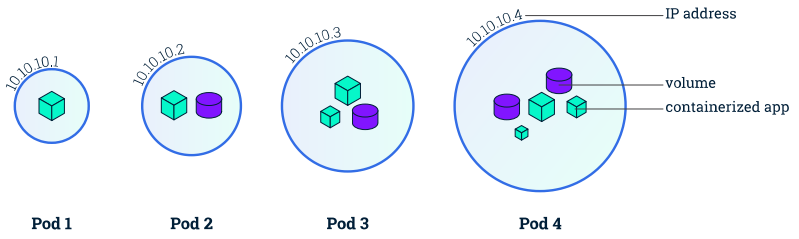
Kubernetes 组件



- 控制平面
- Kubernetes 节点
- 持久存储
- 容器镜像仓库

Pod

Pods 基础知识



Kubernetes 调度的最小单位

- 容器组合：Pod 包含一个或多个共享网络和存储资源的容器，适合紧密耦合的应用组件。
- 生命周期：Pod 有独特的生命周期，创建后会被调度到节点上执行，然后可能因状态改变或节点故障而终止或重启。
- 网络和存储：Pod 内的所有容器共享唯一的 IP 地址和网络端口，并可以定义共享存储卷进行数据共享。

Pod

Pod 例子

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  labels:
    zone: prod
    version: v1
spec:
  containers:
  - name: nginx
    image: nginx:alpine
    ports:
    - containerPort: 80
```

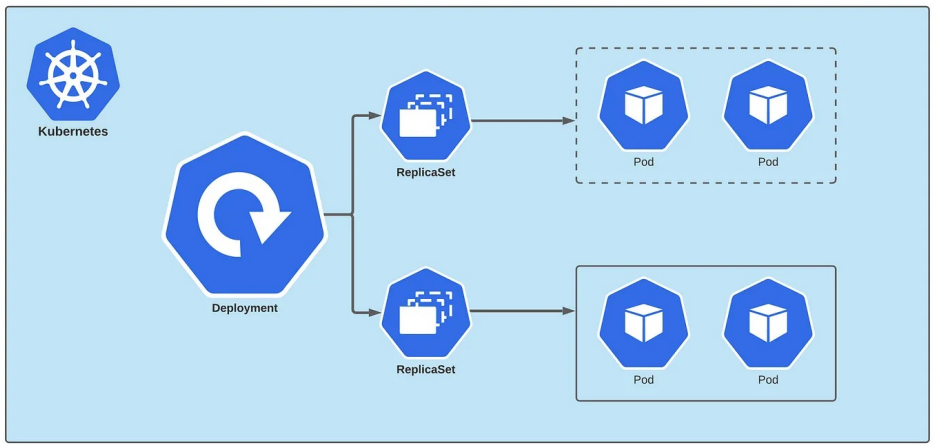
- apiVersion: Kubernetes 配置文件中用来指定配置资源对象所使用的 Kubernetes API 版本。
- Kind: Kubernetes 配置文件中用来指定配置资源对象的类型。(常见的包括 Service， Deployment， StatefulSet，PersistentVolume等)
- Metadata: Kubernetes 配置文件中用来指定配置资源对象元数据的字段。
- spec: 用于定义资源对象的规范，包括容器、网络、存储等方面的配置。

Pod Review

- Pod 基本概念
- 如何编写一个 Pod 配置文件

Deployment

介绍



Pod是Kubernetes中最小的可部署单元，而Deployment是用于管理Pod的对象。 Deployment可以创建和管理多个Pod副本，并提供滚动更新和回滚等功能。通过Deployment，我们可以方便地管理和扩展应用程序的副本。

Deployment

创建 deployment

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: hello-deploy
spec:
  replicas: 10
  selector:
    matchLabels:
      app: hello-world
  minReadySeconds: 10
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 1
      maxSurge: 1
  template:
    metadata:
      labels:
        app: hello-world
    spec:
      containers:
      - name: hello-pod
        image: nginx:1.14.0-alpine
        ports:
        - containerPort: 80
```

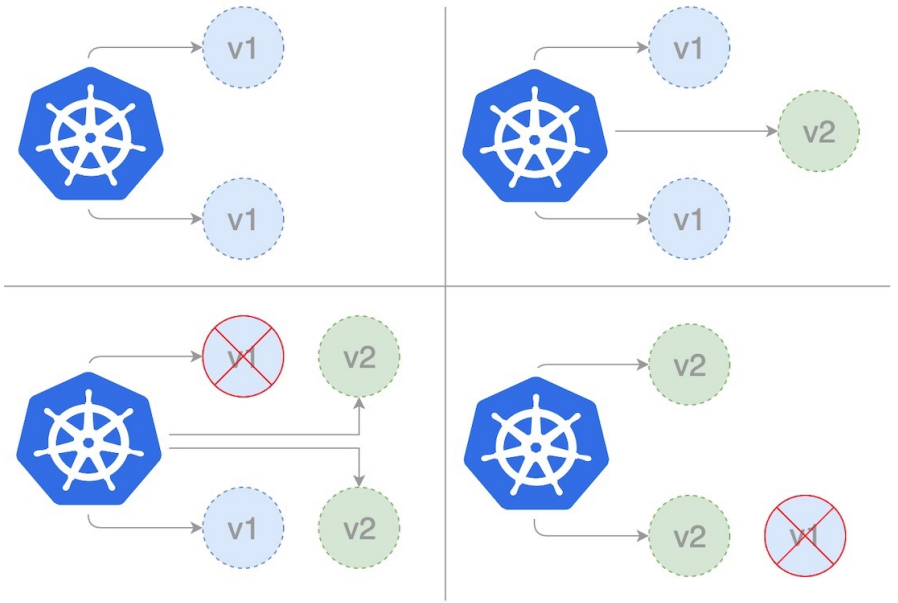
```
❯ kubectl get deploy hello-deploy
NAME           READY   UP-TO-DATE   AVAILABLE   AGE
hello-deploy   10/10   10           10          92m
```

- minReadySeconds: 10：指定了在 Pod 变为 Ready 状态之前的最小等待时间（以秒为单位）。
- maxUnavailable: 1：指定了在进行滚动更新期间允许的最大不可用Pod的数量。
- maxSurge: 1：指定了在进行滚动更新期间允许的最大超出Pod的数量。

Deployment

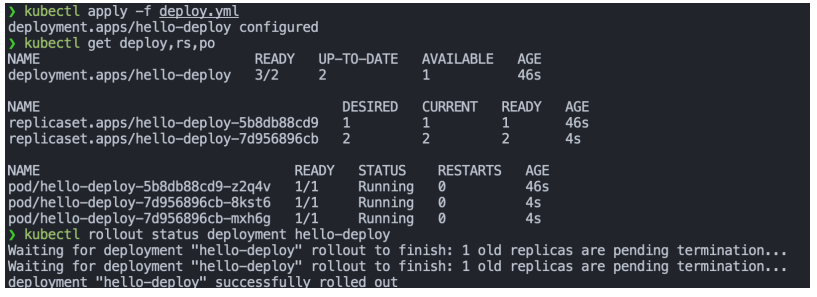
滚动更新

```
image: nginx:1.15.1-alpine
```



Deployment

滚动更新



- 无中断服务： 滚动更新允许你在不停止服务的情况下更新应用程序版本。当你在 Deployment 中更改容器的镜像版本时，Kubernetes 会逐个替换旧的 Pod，同时确保服务的可用性。
- 流量管理： 在滚动更新过程中，新的 Pod 启动并准备好接受流量后，Kubernetes 将开始将流量转向新的 Pod，然后关闭一个旧的 Pod。这样做可以确保任何时候系统都至少有所需数量的 Pod 在运行。

Deployment

回滚

```
❯ kubectl rollout history deployment hello-deploy
deployment.apps/hello-deploy
REVISION  CHANGE-CAUSE
1         <none>
2         <none>

❯ kubectl rollout undo deployment hello-deploy --to-revision=1
deployment.apps/hello-deploy rolled back
```

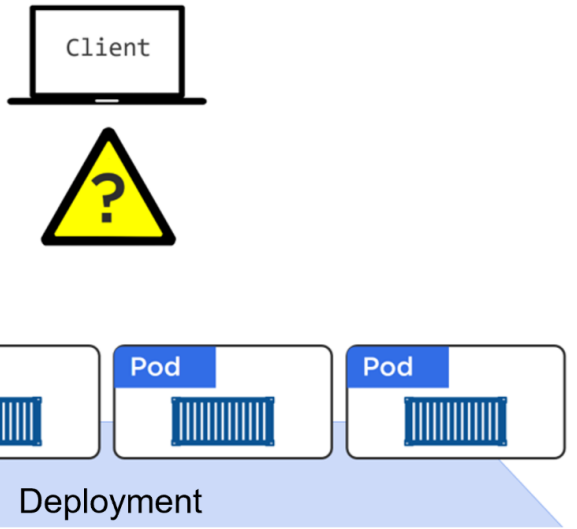
- 版本回滚： 如果新版本的应用程序出现问题，你可以使用 Kubernetes 的回滚功能恢复到 Deployment 的前一个版本。这个操作可以立即执行，而且通常不会对应用程序的可用性产生影响

Deployment Review

- Deployment 概念
- Deployment 的滚动更新和回滚
- 如何编写 Deployment 配置文件

Service

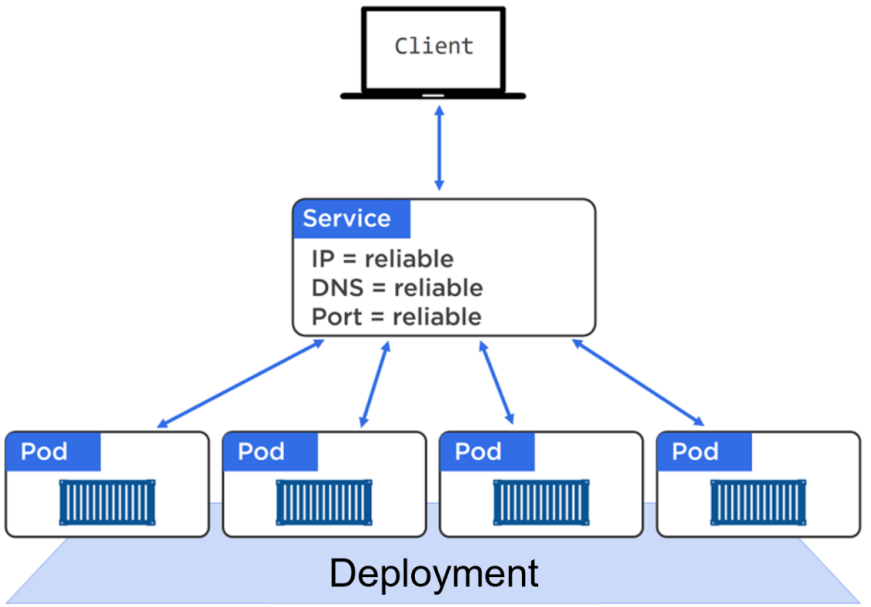
为什么要引入 Service



Pod IP是不可靠的。当Pod失败时，它们会被具有新IP的新Pod替换。扩展 Kubernetes 部署会引入具有新IP地址的新Pod。

Service

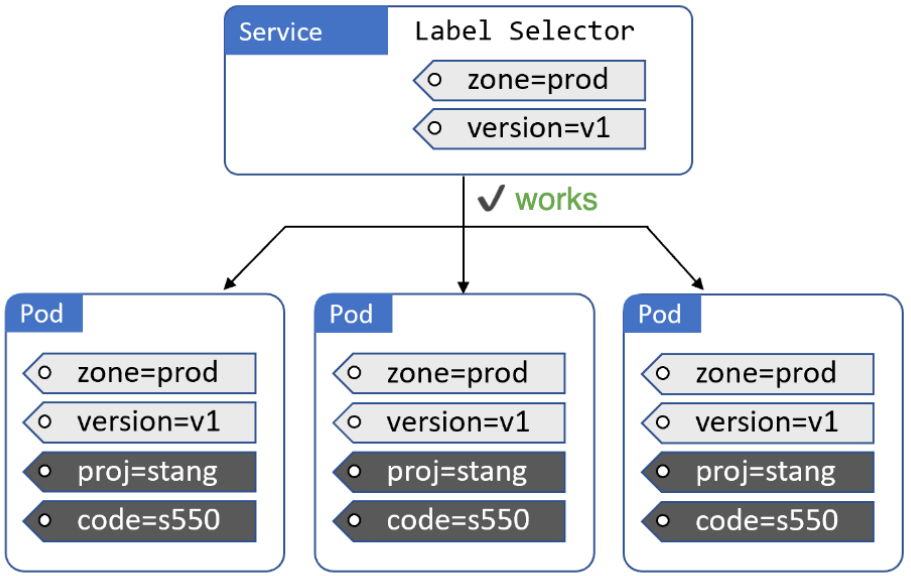
介绍



- 它充当了一个抽象层，将前端（客户端）与后端（Pods）解耦，并提供了一个访问Pod的单一入口点。
- 应用程序可以依赖于稳定的网络端点，而不管Pod的位置或状态发生的变化。
- 提供负载均衡的能力。它将传入的网络流量分布到多个Pod上，确保工作负载均匀分布。

Service

Lables

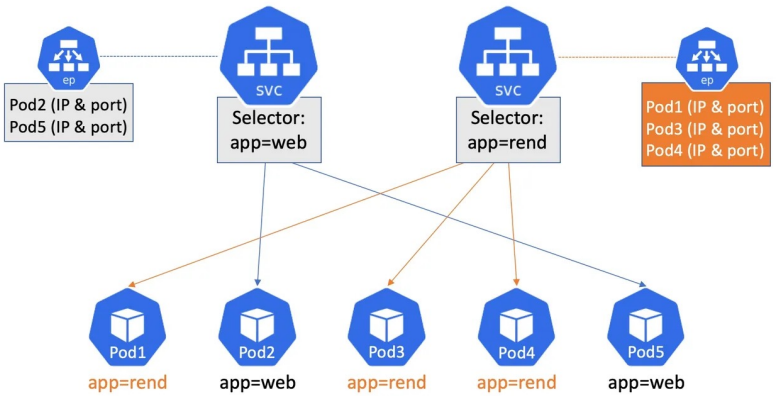


为了建立Service与应该路由流量的Pods之间的连接，Service使用标签选择器。标签选择器充当过滤器，指定应包括在Service的路由配置中的哪些Pods。

Service和Pods之间的松耦合是通过标签和标签选择器实现的。

Service

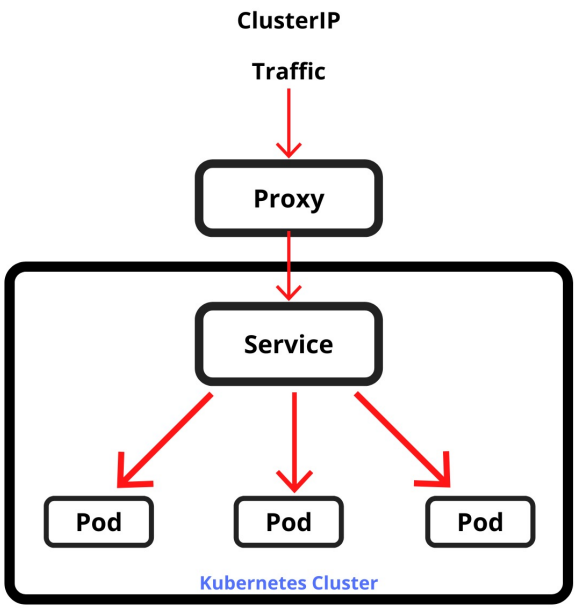
Endpoints



- 提供 Service 的稳定网络地址：Endpoint 为 Service 提供一个稳定的网络地址，使得其他服务或客户端可以通过该地址访问 Service。
- 动态管理后端 Pod 的网络连接：Endpoint 动态地管理与 Service 关联的后端 Pod 的网络连接。当 Pod 的状态发生变化（如创建、删除、更新）时，Endpoint 会相应地更新与之关联的网络连接。
- 负载均衡：Endpoint 可以根据 Service 的负载均衡策略，在多个后端 Pod 之间分发请求，以实现负载均衡。
- 服务发现：Endpoint 提供了一种服务发现的机制，使得其他服务或客户端能够自动发现并连接到 Service 的后端 Pod。

Service

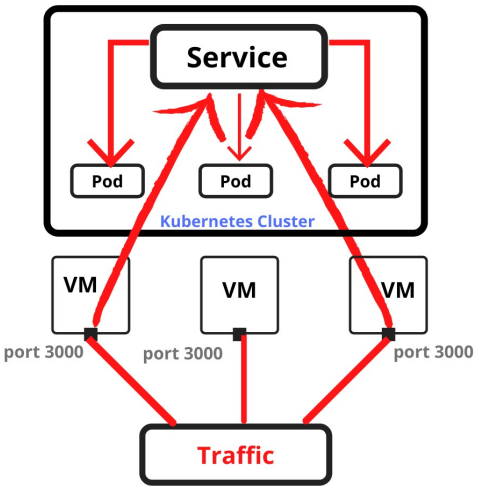
Service Type – ClusterIP



- 内部服务通信：在Kubernetes集群内部，可以使用ClusterIP来创建一个内部服务，用于实现不同的Pod之间的通信。这种服务通常不会被外部访问到，只能在集群内部使用。
- 数据库访问：ClusterIP可以用于连接到数据库服务，例如MySQL或PostgreSQL。通过将数据库服务暴露为ClusterIP，其他Pod可以通过该ClusterIP来访问数据库。
- 内部负载均衡：当有多个副本的Pod运行相同的应用程序时，可以使用ClusterIP来实现内部负载均衡。通过将服务暴露为ClusterIP，Kubernetes会自动将请求分发到不同的Pod副本上。

Service

Service Type – NodePort

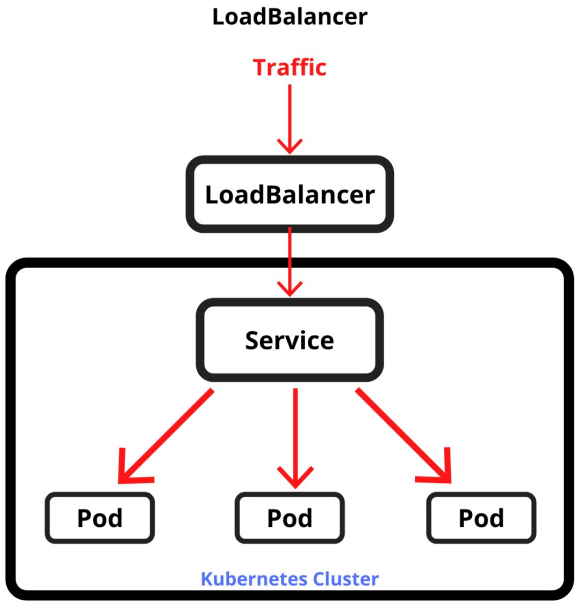


NodePort

- 外部访问服务：通过使用NodePort类型的服务，可以将服务公开给集群外部的网络。外部用户可以通过访问任意节点的IP地址和NodePort端口号来访问服务。
- 测试和开发环境：在测试和开发环境中，NodePort可以方便地将服务暴露给开发人员和测试人员进行访问和测试，而无需配置复杂的负载均衡器或域名解析。
- 高可用性和负载均衡：当多个节点上运行相同的服务时，NodePort类型的服务可以实现负载均衡和高可用性。请求会自动分发到不同的节点上运行的Pod副本上。

Service

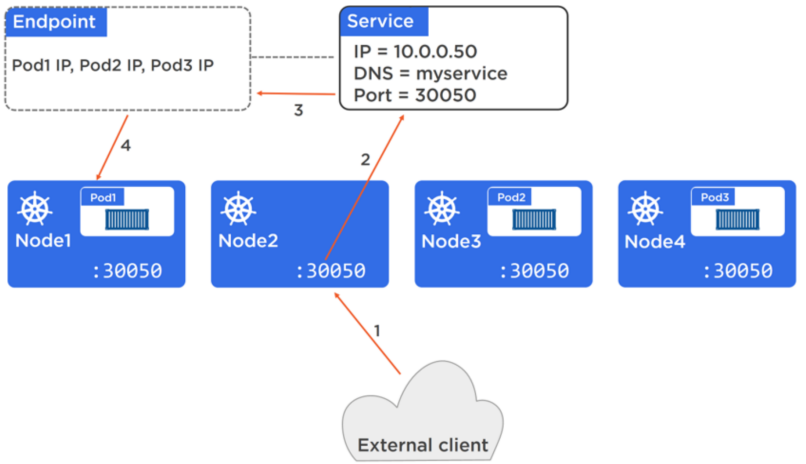
Service Type – LoadBalancer



- 外部访问服务：LoadBalancer类型的服务可以将服务公开给集群外部的网络。云服务提供商将自动为服务创建一个负载均衡器，并将其配置为将流量转发到服务的后端。
- 高可用性和负载均衡：当多个副本的Pod运行相同的服务时，LoadBalancer类型的服务可以实现负载均衡和高可用性。负载均衡器会自动将流量分发到不同的Pod副本上，以确保服务的可用性和性能。
- 自动扩展：当服务的负载增加时，LoadBalancer可以自动扩展服务的后端Pod副本数量，以应对流量的增加。这样可以确保服务的性能和可用性。

Service

NodePort 例子



1. 外部客户端通过端口30050访问Node2。
2. 请求被重定向到Service对象（即使Node2上没有运行与Service关联的Pod）
3. Service对象有一个关联的Endpoint对象，其中包含始终更新的与标签选择器匹配的Pod列表
4. 客户端被指向Node1上的pod1。

Service Review

- Service 概念
- 3种 Service 类型和使用场景
- 如何编写 Service 配置文件

Q & A