

Service

Service

Service 的概念

核心迭代

I、userspace 代理模式

II、iptables 代理模式

III、ipvs 代理模式

限制

Service 的类型

ClusterIP

创建 myapp-deploy.yaml 文件

创建 Service 信息

Headless Service

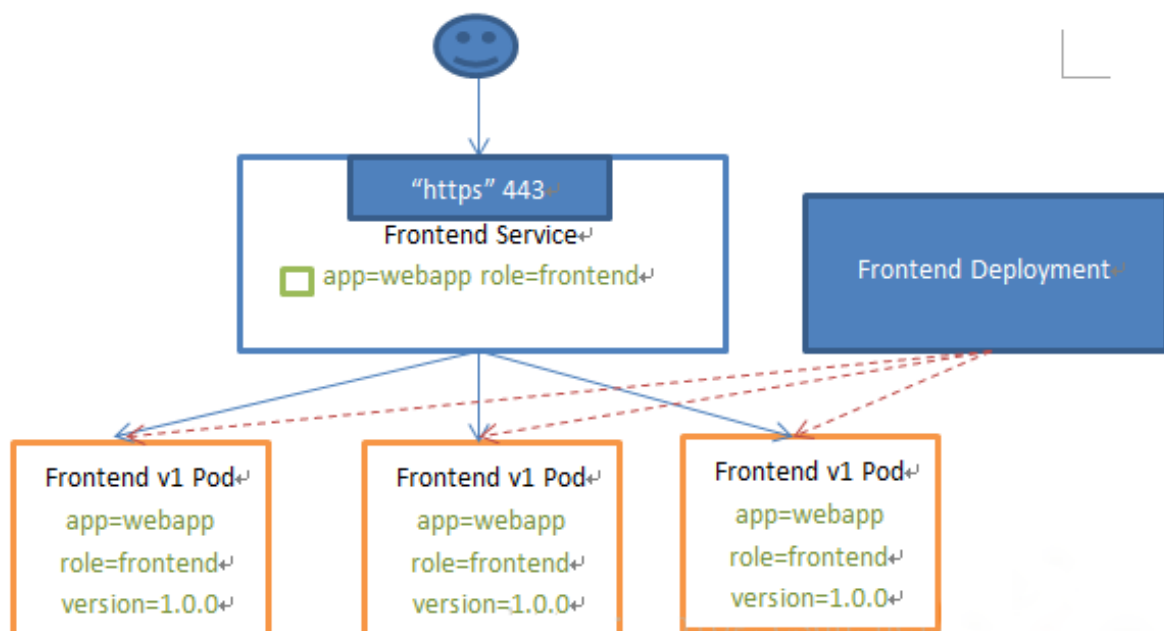
NodePort

LoadBalancer

ExternalName

Service 的概念

Kubernetes `Service` 定义了这样一种抽象：一个 `Pod` 的逻辑分组，一种可以访问它们的策略 —— 通常称为微服务。这一组 `Pod` 能够被 `Service` 访问到，通常是通过 `Label Selector`



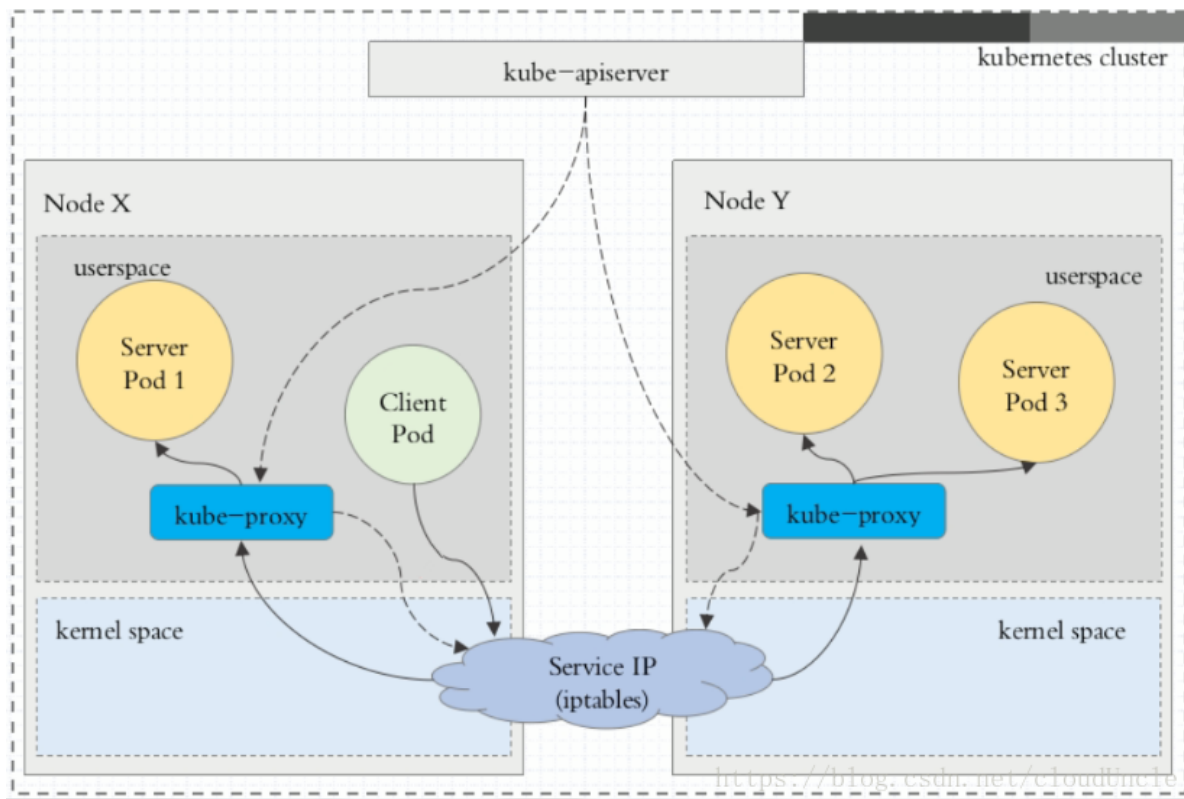
核心迭代

在 Kubernetes 集群中，每个 Node 运行一个 kube-proxy 进程。kube-proxy 负责为 Service 实现了一种 VIP（虚拟 IP）的形式，而不是 ExternalName 的形式。在 Kubernetes v1.0 版本，代理完全在 userspace。在 Kubernetes v1.1 版本，新增了 iptables 代理，但并不是默认的运行模式。从 Kubernetes v1.2 起，默认就是 iptables 代理。在 Kubernetes v1.8.0-beta.0 中，添加了 ipvs 代理

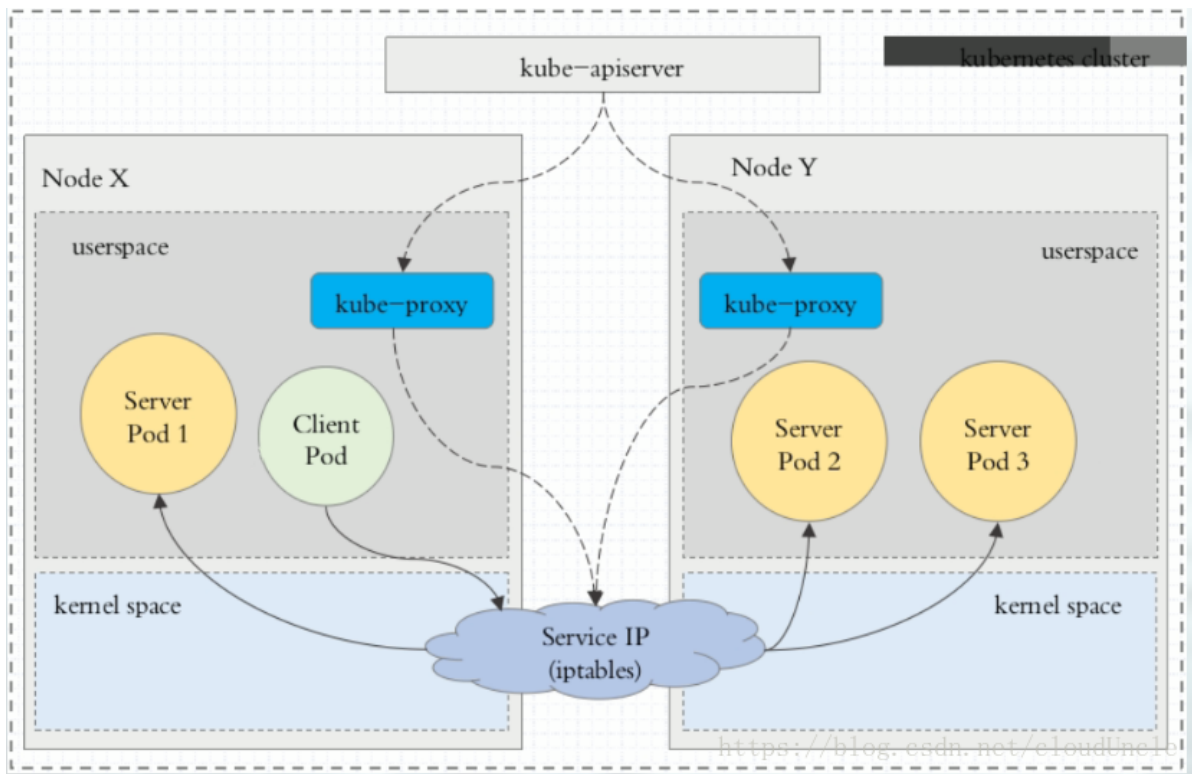
在 Kubernetes 1.14 版本开始默认使用 ipvs 代理

在 Kubernetes v1.0 版本，Service 是“4层”（TCP/UDP over IP）概念。在 Kubernetes v1.1 版本，新增了 Ingress API（beta 版），用来表示“7层”（HTTP）服务

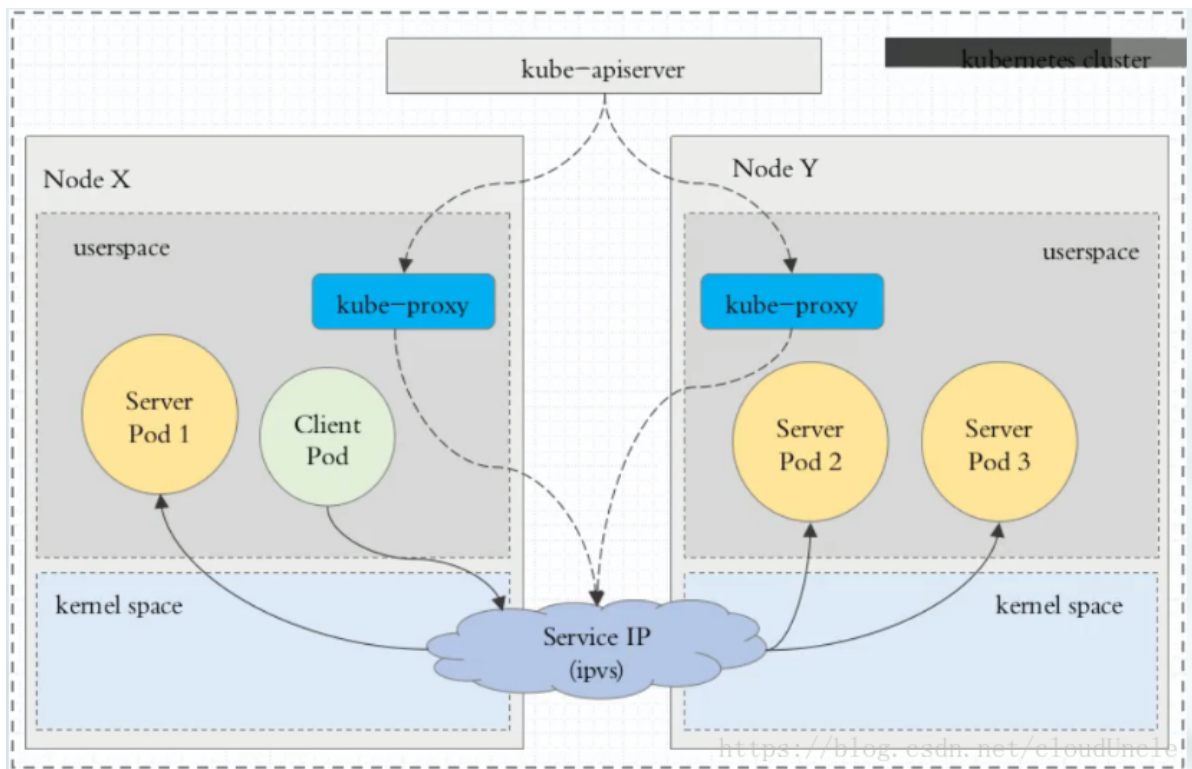
I、userspace 代理模式



II、iptables 代理模式



III、ipvs 代理模式



限制

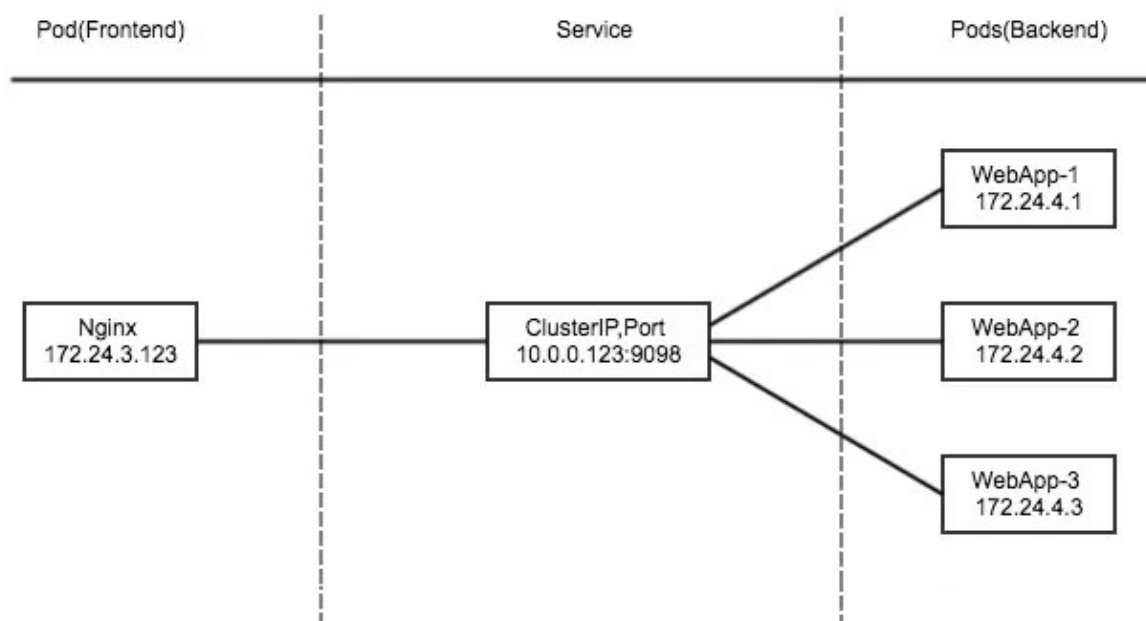
Service能够提供负载均衡的能力，但是在使用上有以下限制：

- 只提供 4 层负载均衡能力，而没有 7 层功能，但有时我们可能需要更多的匹配规则来转发请求，这点上 4 层负载均衡是不支持的

Service 的类型

- ClusterIp: 默认类型，自动分配一个仅 Cluster 内部可以访问的虚拟 IP
- NodePort: 在 ClusterIP 基础上为 Service 在每台机器上绑定一个端口，这样就可以通过 NodePort 来访问该服务
- LoadBalancer: 在 NodePort 的基础上，借助 cloud provider 创建一个外部负载均衡器，并将请求转发到: NodePort
- ExternalName: 把集群外部的服务引入到集群内部来，在集群内部直接使用。没有任何类型代理被创建，这只有 kubernetes 1.7 或更高版本的 kube-dns 才支持

ClusterIP



为了实现图上的功能，主要需要以下几个组件的协同工作：

- apiserver: 用户通过 kubectl 命令向 apiserver 发送创建 service 的命令，apiserver 接收到请求后将数据存储到 etcd 中
- kube-proxy: kubernetes 的每个节点中都有一个叫做 kube-proxy 的进程，这个进程负责感知 service, pod 的变化，并将变化的信息写入本地的 ipvs 规则中
- ipvs: 基于内核的钩子函数机制实现负载均衡

创建 myapp-deploy.yaml 文件

```
[root@master manifests]# vim myapp-deploy.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp-deploy
  namespace: default
spec:
  replicas: 3
  selector:
    matchLabels:
      app: myapp
      release: stable
  template:
    metadata:
      labels:
        app: myapp
        release: stable
        env: test
    spec:
      containers:
        - name: myapp
          image: wangyanglinux/myapp:v2
          imagePullPolicy: IfNotPresent
          ports:
            - name: http
              containerPort: 80
```

创建 Service 信息

```
[root@master manifests]# vim myapp-service.yaml
apiVersion: v1
kind: Service
metadata:
  name: myapp
  namespace: default
spec:
  type: ClusterIP
  selector:
    app: myapp
    release: stable
  ports:
    - name: http
      port: 80
      targetPort: 80
```

Headless Service

有时不需要或不想要负载均衡，以及单独的 Service IP。遇到这种情况，可以通过指定 Cluster IP (spec.clusterIP) 的值为 “ None ” 来创建 Headless Service。这类 Service 并不会分配 Cluster IP, kube-proxy 不会处理它们，而且平台也不会为它们进行负载均衡和路由

```
# yum -y install bind-utils
[root@k8s-master manifests]# vim myapp-svc-headless.yaml
apiVersion: v1
kind: Service
metadata:
  name: myapp-headless
  namespace: default
spec:
  selector:
    app: myapp
  clusterIP: "None"
  ports:
  - port: 80
    targetPort: 80

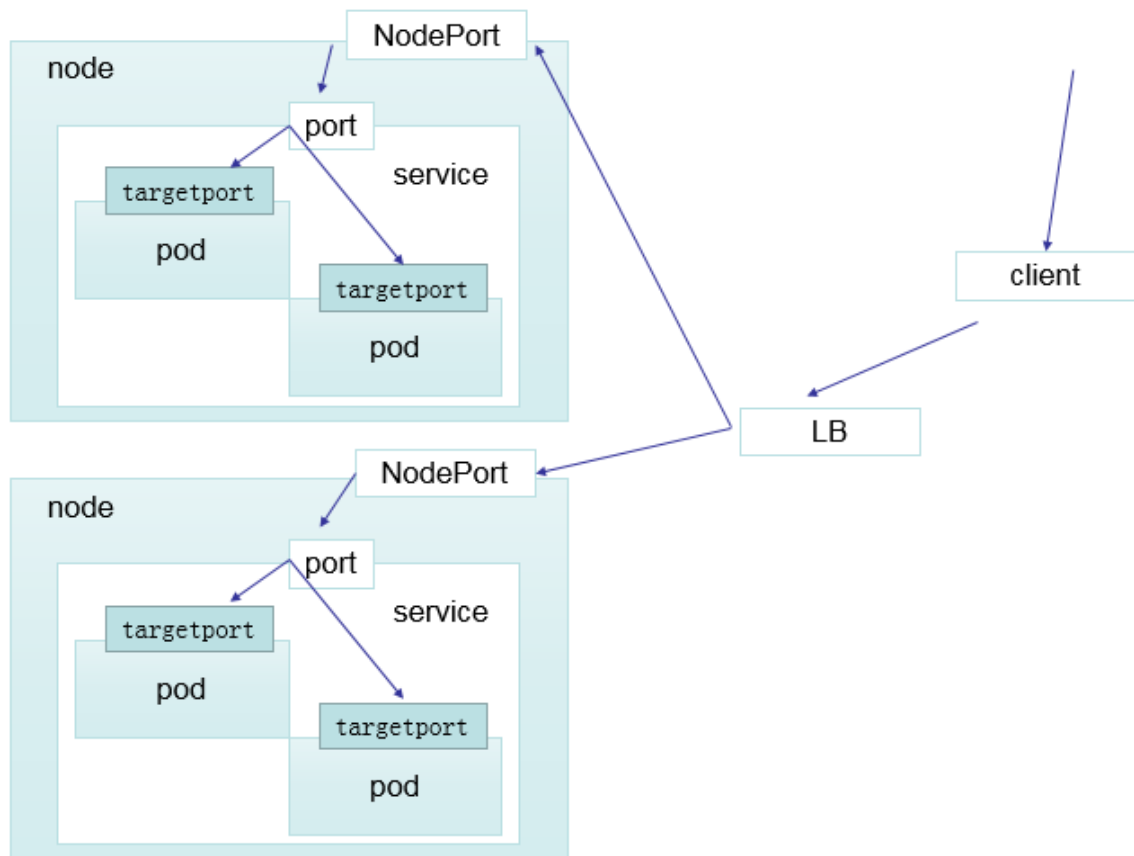
[root@k8s-master manifests]# dig -t A myapp-headless.default.svc.cluster.local.
@10.96.0.10
```

NodePort

```
[root@master manifests]# vim myapp-service.yaml
apiVersion: v1
kind: Service
metadata:
  name: myapp
  namespace: default
spec:
  type: NodePort
  selector:
    app: myapp
    release: stable
  ports:
  - name: http
    port: 80
    targetPort: 80
```

LoadBalancer

loadBalancer 和 nodePort 其实是同一种方式。区别在于 loadBalancer 比 nodePort 多了一步，就是可以调用 cloud provider 去创建 LB 来向节点导流



ExternalName

这种类型的 Service 通过返回 CNAME 和它的值，可以将服务映射到 externalName 字段的内容(例如: hub.hongfu.com)。ExternalName Service 是 Service 的特例，它没有 selector，也没有定义任何的端口和 Endpoint。相反的，对于运行在集群外部的服务，它通过返回该外部服务的别名这种方式来提供服务

```

kind: Service
apiVersion: v1
metadata:
  name: my-service-1
  namespace: default
spec:
  type: ExternalName
  externalName: hub.hongfu.com
  
```

当查询主机 my-service.default.svc.cluster.local (SVC_NAME.NAMESPACE.svc.cluster.local) 时，集群的 DNS 服务将返回一个值 my.database.example.com 的 CNAME 记录。访问这个服务的工作方式和其他的相同，唯一不同的是重定向发生在 DNS 层，而且不会进行代理或转发