

# k8s实战之Service

## 一、概述

为了适应快速的业务需求，微服务架构已经逐渐成为主流，微服务架构的应用需要有非常好的服务编排支持，k8s中的核心要素Service便提供了一套简化的服务代理和发现机制，天然适应微服务架构，任何应用都可以非常轻易地运行在k8s中而无须对架构进行改动；

k8s分配给Service一个固定IP，这是一个虚拟IP(也称为ClusterIP)，并不是一个真实存在的IP，而是由k8s虚拟出来的。虚拟IP的范围通过k8s API Server的启动参数 `--service-cluster-ip-range=19.254.0.0/16`配置；

虚拟IP属于k8s内部的虚拟网络，外部是寻址不到的。在k8s系统中，实际上是由k8s Proxy组件负责实现虚拟IP路由和转发的，所以k8s Node中都必须运行了k8s Proxy，从而在容器覆盖网络之上又实现了k8s层级的虚拟转发网络。

服务代理：

在逻辑层面上，Service被认为是真实应用的抽象，每一个Service关联着一系列的Pod。在物理层面上，Service有事真实应用的代理服务器，对外表现为一个单一访问入口，通过k8s Proxy转发请求到Service关联的Pod。

Service同样是根据Label Selector来刷选Pod进行关联的，实际上k8s在Service和Pod之间通过Endpoint衔接，Endpoints同Service关联的Pod；相对应，可以认为是Service的服务代理后端，k8s会根据Service关联到Pod的PodIP信息组合成一个Endpoints。

```
1 #kubectl get service my-nginx
2 #kubectl get pod --selector app=nginx
3 k8s创建Service的同时，会自动创建跟Service同名的Endpoints:
4 #kubectl get endpoints my-nginx -o yaml
5 #kubectl describe service my-nginx
```

Service不仅可以代理Pod，还可以代理任意其他后端，比如运行在k8s外部的服务。加速现在要使用一个Service代理外部MySQL服务，不用设置Service的Label Selector。

Service的定义文件：mysql-service.yaml：



```
1 apiVersion: v1
2 kind: Service
3 metadata:
```

```
4     name: mysql
5 spec:
6     ports:
7     - port: 3306
8       targetPort: 3306
9       protocol: TCP
```



同时定义跟Service同名的Endpoints，Endpoints中设置了MySQL的IP：192.168.3.180；  
Endpoints的定义文件mysql-endpoints.yaml：

```
1 apiVersion: v1
2 kind: Endpoints
3 metadata:
4     name: mysql
5 subsets:
6 - addresses:
7     - ip: 192.168.39.175
8   ports:
9     - port: 3306
10 protocol: TCP
```



```
#kubectl create -f mysql-service.yaml -f mysql-endpoints.yaml
```

微服务化应用的每一个组件都以Service进行抽象，组件与组件之间只需要访问Service即可以互相通信，而无须感知组件的集群变化。  
这就是服务发现；

```
#kubectl exec my-pod -- nslookup my-service.my-ns --namespace=default
```

```
#kubectl exec my-pod -- nslookup my-service --namespace=my-ns
```

## 二、Service发布

---

k8s提供了NodePort Service、LoadBalancer Service和Ingress可以发布Service;

### NodePort Service

NodePort Service是类型为NodePort的Service, k8s除了会分配给NodePort Service一个内部的虚拟IP, 另外会在每一个Node上暴露端口NodePort, 外部网络可以通过[NodeIP]:[NodePort]访问到Service。

### LoadBalancer Service (需要底层云平台支持创建负载均衡器,比如GCE)

LoadBalancer Service是类型为LoadBalancer的Service, 它是建立在NodePort Service集群基础上的, k8s会分配给LoadBalancer Service一个内部的虚拟IP, 并且暴露NodePort。除此之外, k8s请求底层云平台创建一个负载均衡器, 将每个Node作为后端, 负载均衡器将转发请求到[NodeIP]:[NodePort]。



```
1 apiVersion: v1
2 kind: Service
3 metadata:
4   name: my-nginx
5 spec:
6   selector:
7     app: nginx
8   ports:
9     - name: http
10     port: 80
11     targetPort: 80
12     protocol: TCP
13   type: LoadBalancer
```



负载均衡器由底层云平台创建提供, 会包含一个LoadBalancerIP, 可以认为是LoadBalancer Service的外部IP, 查询LoadBalancer Service:

```
#kubectl get svc my-nginx
```

### Ingress

k8s提供了一种HTTP方式的路由转发机制, 称为Ingress。Ingress的实现需要两个组件支持, Ingress Controller和HTTP代理服务器。HTTP代

理服务器将会转发外部的HTTP请求到Service，而Ingress Controller则需要监控k8s API，实时更新HTTP代理服务器的转发规则；



```
1 apiVersion: extensions/v1beta1
2 kind: Ingress
3 metadata:
4   name: my-ingress
5 spec:
6   rules:
7   - host: my.example.com
8     http:
9       paths:
10      - path: /app
11 backend:
12   serviceName: my-app
13   servicePort: 80
```



Ingress 定义中的.spec.rules 设置了转发规则，其中配置了一条规则，当HTTP请求的host为my.example.com且path为/app时，转发到Service my-app的80端口；

```
#kubectl create -f my-ingress.yaml; kubectl get ingress my-ingress
```

NAME	RULE	BACKEND	ADDRESS
my-ingress	-		
		my.example.com	
	/app	my-app:80	

当Ingress创建成功后，需要Ingress Controller根据Ingress的配置，设置HTTP代理服务器的转发策略，外部通过HTTP代理服务就可以访问到Service；