

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

# Kubernetes 四层负载均衡 Service

前言：  
课程名称：Kubernetes 四层负载均衡 Service

实验环境：  
本章节 Kubernetes 集群环境如下：

角色	IP	主机名	组件	硬件
控制节点	192.168.128.11	k8s-master01	apiserver controller-manager scheduler etcd containerd	CPU：4vCPU 硬盘：100G 内存：4GB 开启虚拟化
工作节点	192.168.128.21	k8s-node01	kubelet kube-proxy containerd calico coredns	CPU：6vCPU 硬盘：100G 内存：8GB 开启虚拟化
工作节点	192.168.128.22	k8s-node02	kubelet kube-proxy containerd calico coredns	CPU：6vCPU 硬盘：100G 内存：8GB 开启虚拟化

张岩峰老师微信，加我微信，邀请你加入 VIP 交流答疑群：  
微信号：ZhangYanFeng0429  
二维码：

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**



## 1、初识四层负载均衡 Service

### 1.1、为什么要有 Service?

在 Kubernetes 中，Service 是一种可以暴露 Pod 给其他容器或外部用户访问的抽象概念。虽然 Kubernetes 内部网络中的 Pod 使用 IP 地址可以直接访问，但它们的 IP 地址通常只是临时的，会随着 Pod 的重启和重新调度而变化。因此，直接使用 Pod IP 访问有时是不可取的，而 Service 在解决这个问题上起到了重要的作用。

Service 的作用主要有以下几点：

1、提供固定的 IP 和 DNS 名称。当你创建一个 Service 后，它会分配一个稳定的虚拟 IP 地址，可以通过这个地址和 Service 名称访问服务，无需关心后台 Pod 的 IP。

2、负载均衡。Service 可以让请求分布到多个 Pod 上，从而实现负载均衡。例如，当有多个后台 Pod 时，Service 可以基于预定义的负载均衡算法将请求分发到这些 Pod 中。

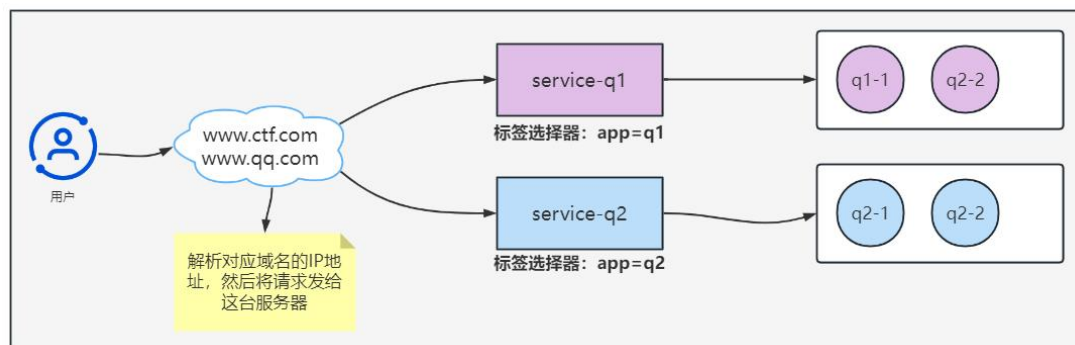
3、服务发现。Service 通过匹配 Label Selector，可以将一组后台 Pod 当成一个 Service 进行管理，这样我们就可以方便地对这些 Pod 进行管理、监控及网络访问控制等操作。

通过使用 Service，我们可以轻松地在 Kubernetes 集群内部和外部访问应用服务。在实际的应用场景中，往往会结合 Ingress 以实现更复杂的代理和负载均衡需求。

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

Service 原理图：



## 1.2、Service 概述

service 是一个固定接入层，客户端可以通过访问 service 的 ip 和端口访问到 service 关联的后端 pod，这个 service 工作依赖于在 kubernetes 集群之上部署的一个附件，就是 kubernetes 的 dns 服务（不同 kubernetes 版本的 dns 默认使用的也是不一样的，1.11 之前的版本使用的是 kubeDNs，较新的版本使用的是 coredns），service 的名称解析是依赖于 dns 附件的，因此在部署完 k8s 之后需要再部署 dns 附件，kubernetes 要想给客户端提供网络功能，需要依赖第三方的网络插件（flannel、calico、kube-ovn 等）。

每个 k8s 节点上都有一个组件叫做 kube-proxy，kube-proxy 这个组件将始终监视着 apiserver 中有关 service 资源的变动信息，需要跟 master 之上的 apiserver 交互，随时连接到 apiserver 上获取任何一个与 service 资源相关的资源变动状态，这种是通过 kubernetes 中固有的一种请求方法 watch（监视）来实现的，一旦有 service 资源的内容发生变动（如创建，删除），kube-proxy 都会将它转化成当前节点之上的能够实现 service 资源调度，把我们请求调度到后端特定的 pod 资源之上的规则，这个规则可能是 iptables，也可能是 ipvs，取决于 service 的实现方式。

## 1.3、Service 工作原理

k8s 在创建 Service 时，会根据标签选择器 selector（lable selector）来查找 Pod，据此创建与 Service 同名的 endpoint 对象，当 Pod 地址发生变化时，endpoint 也会随之发生变化，service 接收前端 client 请求的时候，就会通过 endpoint，找到转发到哪个 Pod 进行访问的地址。（至于转发到哪个节点的 Pod，由负载均衡 kube-proxy 决定）

## 1.4、kubernetes 集群中有三类 IP 地址

### ● 1、Node Network（节点网络）

例如：物理节点或者虚拟节点的网络，如 ens33 接口上的网络地址。

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

```
[root@k8s-master01 ~]# ip addr
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP
group default qlen 1000
    link/ether 00:0c:29:d1:b8:8d brd ff:ff:ff:ff:ff:ff
    inet 192.168.128.11/24 brd 192.168.128.255 scope global noprefixroute ens33
```

● 2、Pod network (pod 网络)  
创建的 Pod 具有的 IP 地址

```
[root@k8s-master01 ~]# kubectl get pods -o wide
NAME          READY   STATUS    RESTARTS   AGE      IP
mysql-5926j    1/1     Running   0          2d1h     172.16.58.222
```

Node Network 和 Pod network 这两种网络地址是我们实实在在配置的，其中节点网络地址是配置在节点接口之上，而 pod 网络地址是配置在 pod 资源之上的，因此这些地址都是配置在某些设备之上的，这些设备可能是硬件，也可能是软件模拟的。

● 3、Cluster Network (集群地址，也称为 service network)

这个地址是虚拟的地址 (virtual ip)，没有配置在某个接口上，只是出现在 service 的规则当中。

```
[root@k8s-master01 ~]# kubectl get svc kubernetes
NAME          TYPE          CLUSTER-IP   EXTERNAL-IP   PORT(S)    AGE
kubernetes    ClusterIP     10.255.0.1   <none>        443/TCP    6d6h
```

2、Service 资源清单文件编写

Service 资源可以通过如下命令查看相关语法：

```
[root@k8s-master01 ~]# kubectl explain Service
```

● Service 资源说明

属性名称	取值类型	取值说明
apiVersion	<string>	Api 版本
kind	<string>	资源类型
metadata	<Object>	元数据
metadata.name	String	控制器的名称
metadata.namespace	String	控制器所属的命名空间，默认值为 default
metadata.labels[]	List	自定义标签列表
metadata.annotation[]	List	自定义注解列表
spec	<Object>	规范 Service 所需行为的规范
spec.clusterIP	<string>	表示分配给该 Service 的集群内部的 IP 地址。这个 IP 地址可以用来访问该 Service 提供的服务。 clusterIP 可以被配置为一个特定的 IP 地址，也可以让

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

		<p>Kubernetes 自动分配一个可用的 IP 地址。如果配置为一个特定的 IP 地址，需要保证该 IP 地址未被其他对象使用，并且在同一子网（subnet）内。</p> <p>例如，如果设置 clusterIP 为 10.0.0.100，则所有访问该 Service 的请求都将被路由到该 IP 地址。</p>
spec.ports	<[]Object>	用于服务监听的端口号
spec.ports.appProtocol	<string>	<p>用于指定服务端口上使用的应用层协议。例如 HTTP 或 TLS。</p> <p>示例：</p> <p>http 服务监听了 80 端口，targetPort 为 8080，protocol 为 TCP，此外还指定了 appProtocol 为 http。在使用这种配置时，负载均衡器将根据应用层协议名 http 来路由到后端上。</p> <p>spec：</p> <pre>ports: - name: http   port: 80   protocol: TCP   targetPort: 8080   appProtocol: http</pre>
spec.ports.name	<string>	这个名称并不是必须的，但为了方便管理和使用，通常会给服务的每个端口指定一个名称。
spec.ports.nodePort	<integer>	<p>用于指定 NodePort 模式下服务的节点端口号，在 NodePort 模式下，Kubernetes 会为每个节点分配一个端口号，将访问此端口的流量转发到 Service 的端口上。使用 nodePort 属性可以指定节点端口号。</p> <p>示例：</p> <p>Service 对象监听了 80 端口，NodePort 模式下使用的节点端口号为 30080。当 Kubernetes 集群中的任何节点上的流量通过该端口 30080 访问该 Service 时，流量将被转发到 Service 的 80 端口上，然后 service 会转发到 targetPort 上（也就是 pod 端口上）。</p> <p>spec：</p> <pre>type: NodePort ports: - name: http   port: 80   targetPort: 80   nodePort: 30080</pre>

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

spec.ports.port	<integer>	<p>用于指定服务监听的端口号。</p> <p>示例：</p> <p>Service 对象监听了 80 端口，protocol 为 TCP，targetPort 为 8080。所有发送到 80 端口的流量都将被转发到该 targetPort 上。</p> <p>spec:</p> <pre>ports: - name: http   port: 80   protocol: TCP   targetPort: 8080</pre>
spec.ports.protocol	<string>	<p>标识服务端口监听的协议类型。</p> <p>Kubernetes 中支持 TCP、UDP、SCTP 等多种协议类型。在 Service 对象中，使用 protocol 属性指定服务端口监听的协议类型。默认协议类型为 TCP。</p>
spec.ports.targetPort	<string>	<p>用于指定服务的后端容器监听的端口号。</p> <p>在 Kubernetes 中，Service 对象通常用于代理到后端的 Pod。targetPort 属性用于指定后端 Pod 监听的端口号。当流量进入该端口时，Service 会将其转发到相应的 Pod 上。</p>
spec.selector	<map[string]string>	<p>指定当前服务关联的 Pod Selector。</p> <p>Pod Selector 用于将服务和后端 Pod 进行关联。当创建服务时，可以使用 selector 属性指定服务需要关联哪些 Pod。Kubernetes 将会自动为匹配 Selector 的 Pods，提供网络地址和 DNS 记录。</p>
spec.type	<string>	<p>指定本地或集群内服务的访问方式。</p> <p>Service 对象可以以三种不同的类型来暴露服务：ClusterIP、NodePort 和 LoadBalancer。在创建 Service 对象时，需要通过指定 type 属性，来选择合适的访问方式。</p> <ul style="list-style-type: none"><li>● ClusterIP：默认类型。当指定该类型时，Kubernetes 会为该服务创建一个虚拟的 IP 地址，仅能从集群内部访问该服务。</li><li>● NodePort：当指定该类型时，Kubernetes 会为该服务在每个节点上暴露一个端口号。通过该端口号，可以从外部访问该服务。同时也可以从集群内部访问。</li><li>● LoadBalancer：该类型可用于将容器服务暴露到云平台的负载均衡器中。在使用该类型时，可以通过云平台的负载均衡器，将服务暴露到集群外部并允许外部 IP 地址的访问。</li></ul>
spec.sessionAffinity	<string>	<p>指定服务负载均衡算法。</p>

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

		<p>当创建 Service 对象时,可以通过指定 sessionAffinity 属性,来选择合适保持客户端对同一后端 Pod 之间会话的策略,以实现一些业务需求。Kubernetes 支持以下两种负载均衡策略。</p> <ul style="list-style-type: none"><li>● None: 默认值。这意味着在多个 Pod 之间使用的是基于 IP 的负载均衡。您可以使用此策略将请求随机分发到匹配选择器的 Pod 上,且不受请求者的来源 IP 影响。</li><li>● ClientIP: 这是另外一种 Affinity 策略,它根据请求的源 IP 地址将流量转发到具有相同源 IP 地址的后端 Pod。</li></ul> <p>示例:</p> <p>Service 对象负责将流量转发到与 app=myapp 的 Pod,服务没有 IP 地址,因为它被设置为一个 clusterIP=None 集群 IPService。服务负载均衡使用的是 ClientIP 的 Affinity 策略。</p> <pre>spec:   clusterIP: None   port:     - name: web       port: 8080       targetPort: 80   selector:     app: myapp   sessionAffinity: ClientIP</pre>
spec.sessionAffinityConfig	<Object>	<p>用于指定高级会话亲和性配置。</p> <p>默认情况下, Kubernetes 通过源 IP 地址来实现会话亲和性。但是,在某些情况下,源 IP 地址可能会发生变化,从而导致业务系统的不兼容性。为了解决这个问题, Kubernetes 提供了 sessionAffinityConfig 属性。使用该属性可以让用户配置其他的、更复杂的会话亲和性算法。</p>
spec.sessionAffinityConfig.clientIP.timeoutSeconds	<integer>	<p>设置会话超时时间。默认 10800 秒 (三小时)</p>

### 3、创建 Service: type 类型是 ClusterIP

目标:使用 Deployment 控制器创建 2 个副本,创建 service 类型为 ClusterIP 来代理 Deployment 控制器所创建的副本。

#### (1) 创建测试 Pod

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。



版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

```
[root@k8s-master01 ~]# vi pod_test.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-nginx
spec:
  selector:
    matchLabels:
      run: my-nginx
  replicas: 2
  template:
    metadata:
      labels:
        run: my-nginx
    spec:
      containers:
        - name: my-nginx
          image: nginx
          imagePullPolicy: IfNotPresent
          ports:
            - containerPort: 80
```

提示：

```
- containerPort: 80 #pod 中的容器需要暴露的端口
```

# 更新资源清单文件

```
[root@k8s-master01 ~]# kubectl apply -f pod_test.yaml
deployment.apps/my-nginx created
```

# 查看 pod IP 地址

```
[root@k8s-master01 ~]# kubectl get pods -o wide
```

```
[root@k8s-master01 ~]# kubectl get pods -o wide
NAME                                READY   STATUS    RESTARTS   AGE   IP              NODE     NOMINATED NODE   READINESS GATES
my-nginx-5f7668c4b-72hj9            1/1     Running   0           48s   10.244.85.197   k8s-node01    <none>            <none>
my-nginx-5f7668c4b-cnfb2b          1/1     Running   0           48s   10.244.58.199   k8s-node02    <none>            <none>
```

# 请求 pod ip 地址，查看结果

```
[root@k8s-master01 ~]# curl 10.244.85.197:80
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**



版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

```
[root@k8s-master01 ~]# curl 10.244.85.197:80
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
[root@k8s-master01 ~]#
```

```
[root@k8s-master01 ~]# curl 10.244.58.199:80
```

```
[root@k8s-master01 ~]# curl 10.244.58.199:80
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
[root@k8s-master01 ~]#
```

需要注意的是，pod 虽然定义了容器端口，但是不会使用调度到该节点上的 80 端口，也不会使用任何特定的 NAT 规则去路由流量到 Pod 上。这意味着可以在同一个节点上运行多个 Pod，使用相同的容器端口，并且可以从集群中任何其他 Pod 或节点上使用 IP 的方式访问到它们。

## (2) 误删除其中一个 Pod，查看 pod 标签

```
[root@k8s-master01 ~]# kubectl delete pods my-nginx-5f7668c4b-72hj9
pod "my-nginx-5f7668c4b-72hj9" deleted
```

```
[root@k8s-master01 ~]# kubectl get pods -o wide
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

```
[root@k8s-master01 ~]# kubectl get pods -o wide
NAME                READY   STATUS    RESTARTS   AGE   IP            NODE       NOMINATED NODE   READINESS GATES
my-nginx-5f7668c4b-cnf2b  1/1     Running   0           4m45s  10.244.58.199  k8s-node02  <none>           <none>
my-nginx-5f7668c4b-k2rnq  1/1     Running   0           26s    10.244.85.198  k8s-node01  <none>           <none>
```

通过上面可以看到重新生成了一个 pod: my-nginx-5f7668c4b-k2rnq, ip 是 10.244.85.198, 在 k8s 中创建 pod, 如果 pod 被删除了, 重新生成的 pod ip 地址会发生变化, 所以需要在 pod 前端加一个固定接入层。接下来创建 service:

查看 pod 标签:

```
[root@k8s-master01 ~]# kubectl get pods --show-labels
NAME                READY   STATUS    RESTARTS   AGE   LABELS
my-nginx-5f7668c4b-cnf2b  1/1     Running   0           5m54s  pod-template-hash=5f7668c4b,run=my-nginx
my-nginx-5f7668c4b-k2rnq  1/1     Running   0           95s    pod-template-hash=5f7668c4b,run=my-nginx
```

### (3) 创建 service 资源清单文件

```
[root@k8s-master01 ~]# vi service_ClusterIP.yaml
apiVersion: v1
kind: Service
metadata:
  name: my-nginx
  labels:
    run: my-nginx
spec:
  type: ClusterIP
  ports:
  - name: tcp-80
    port: 80
    protocol: TCP
    targetPort: 80
    appProtocol: http
  selector:
    run: my-nginx
```

对上面的 yaml 文件说明:

用于创建名为 my-nginx 的 Service, 该 Service 用于将外部流量引入到名为 my-nginx 的 Deployment 中。

在配置文件中, 我们定义了 Service 的 metadata 和 spec 部分。metadata 部分定义了 Service 的名称和 label “run: my-nginx”。

在 spec 部分, 我们定义了 Service 的类型为 ClusterIP, 表示只能从 Kubernetes 集群内部访问该 Service。我们还定义了端口, 将 Service 的 80 端口绑定到目标容器的 80 端口, 同时指定 appProtocol 为 http。

最后, 我们定义了 selector, 来指定该 Service 映射到哪些 Pod 上。在这个例子中,

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

我们将 Service 映射到 label 为 “run: my-nginx” 的 Pod 上。

#### (4) 创建 service 资源

```
[root@k8s-master01 ~]# kubectl apply -f service_ClusterIP.yaml
service/my-nginx created
```

#### (5) 查看创建的 service 资源

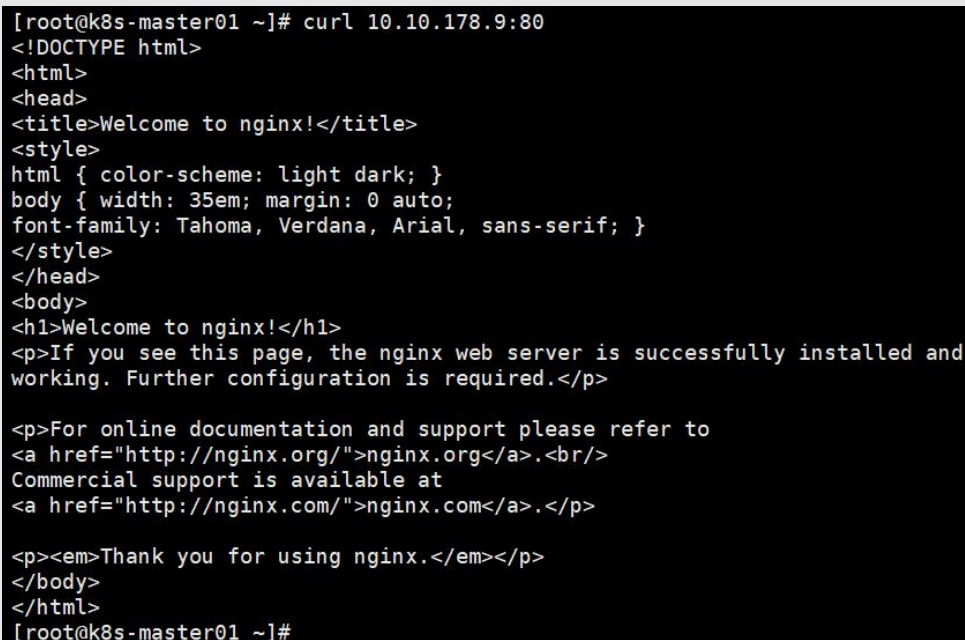
```
[root@k8s-master01 ~]# kubectl get svc my-nginx
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
my-nginx	ClusterIP	10.10.178.9	<none>	80/TCP	26s

#### (6) 测试访问

在 k8s 任意节点访问 service 的 ip:端口就可以把请求代理到后端 pod

```
[root@k8s-master01 ~]# curl 10.10.178.9:80
```



```
[root@k8s-master01 ~]# curl 10.10.178.9:80
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
[root@k8s-master01 ~]#
```

通过上面可以看到请求 service IP:port 跟直接访问 pod ip:port 看到的结果一样，这就说明 service 可以把请求代理到它所关联的后端 pod。

注意：上面的 10.10.178.9:80 地址只能是在 k8s 集群内部可以访问，在外部无法访问，比方说我们想要通过浏览器访问，那么是访问不通的，如果想要在 k8s 集群之外访问，就需要使用 service 的 nodePort 类型。

#### (7) 扩展：访问 10.10.178.9:80 是如何将流量转发给后端 pod 的？

```
[root@k8s-master01 ~]# kubectl describe svc my-nginx
```

Name:	my-nginx
Namespace:	default
Labels:	run=my-nginx
Annotations:	<none>

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

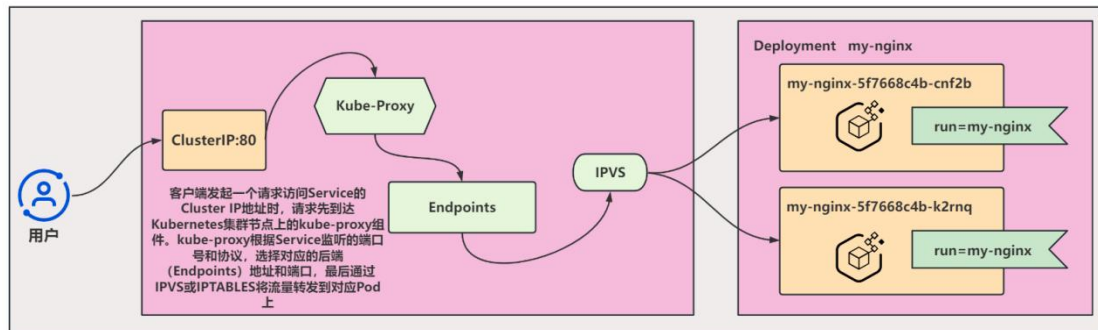
```
Selector:          run=my-nginx
Type:              ClusterIP
IP Family Policy:  SingleStack
IP Families:       IPv4
IP:               10.10.178.9
IPs:              10.10.178.9
Port:             tcp-80 80/TCP
TargetPort:       80/TCP
Endpoints:        10.244.58.199:80,10.244.85.198:80
Session Affinity: None
Events:           <none>

[root@k8s-master01 ~]# kubectl get endpoints my-nginx
NAME            ENDPOINTS                                AGE
my-nginx        10.244.58.199:80,10.244.85.198:80      3m46s

[root@k8s-master01 ~]# ipvsadm -ln | grep 80
TCP  10.10.178.9:80 rr
    -> 10.244.58.199:80 Masq 1      0      0
    -> 10.244.85.198:80 Masq 1      0      0
```

当我们创建一个 Kubernetes Service 后，Service 会自动创建一个虚拟 IP 地址，这个虚拟 IP 地址被称为 Service Cluster IP。在 Service 的 spec 中，我们使用 selector 对象来指定哪些 Pod 是这个 Service 的“后端”，Kubernetes 就会为 Service 自动创建一个相关的 Endpoints 对象，用来记录由这个 Service 管理的 Pod 的 IP 地址和端口号信息。

当客户端发起一个请求访问 Service 的 Cluster IP 地址时，请求先到达 Kubernetes 集群节点上的 kube-proxy 组件。kube-proxy 会根据 Service 监听的端口号和协议，选择对应的后端（Endpoints）地址和端口，然后转发请求到这些后端 Pod 上。kube-proxy 还会通过 iptables 规则或者 IPVS 负载均衡算法等方式来做请求的负载均衡。如下图：



当 Service 的后端 Pod 数量发生变化时，比如新增或者删除 Pod，Endpoints 对象也将会自动更新，使得 kube-proxy 可以将流量转发到新的 Pod 或停止流量

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

转发到已删除的 Pod。这种方式可以让 Service 和 Pod 可以独立地运作，从而让我们更方便、更灵活地管理微服务架构中的不同部分。

#### (8) 扩展：集群的 FQDN (非常重要)

说明：

service 只要创建完成，我们就可以直接解析它的服务名，每一个服务创建完成后都会在集群 dns 中动态添加一个资源记录，添加完成后我们就可以解析了，资源记录格式是：

SVC\_NAME.NS\_NAME.DOMAIN.LTD.

SVC 名称.命名空间名称.域名后缀，集群默认的域名后缀是 svc.cluster.local。

就像我们上面创建的 my-nginx 这个服务，它的完整名称解析就是：  
my-nginx.default.svc.cluster.local

演示查询：

```
[root@k8s-master01 ~]# kubectl exec -it my-nginx-5f7668c4b-cnf2b -- bash
root@my-nginx-5f7668c4b-cnf2b:/# curl my-nginx.default.svc.cluster.local
```

```
root@my-nginx-5f7668c4b-cnf2b:/# curl my-nginx.default.svc.cluster.local
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
root@my-nginx-5f7668c4b-cnf2b:/#
```

```
root@my-nginx-5f7668c4b-cnf2b:/# exit
```

## 4、创建 Service：type 类型是 NodePort

目标：使用 Deployment 控制器创建 2 个副本，创建 service 类型为 NodePort 来代理 Deployment 控制器所创建的副本。

### (1) 创建测试 Pod

```
# 创建 yaml 文件
[root@k8s-master01 ~]# vi pod_test.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-nginx
spec:
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

```
selector:
  matchLabels:
    run: my-nginx
replicas: 2
template:
  metadata:
    labels:
      run: my-nginx
  spec:
    containers:
      - name: my-nginx
        image: nginx
        imagePullPolicy: IfNotPresent
        ports:
          - containerPort: 80

# 更新资源清单文件
[root@k8s-master01 ~]# kubectl apply -f pod_test.yaml
deployment.apps/my-nginx created

# 查看 pod 是否创建成功
[root@k8s-master01 ~]# kubectl get pods -o wide

[root@k8s-master01 ~]# kubectl get pods -o wide
NAME                                READY   STATUS    RESTARTS   AGE   IP              NODE           NOMINATED NODE   READINESS GATES
my-nginx-5f7668c4b-4jb74            1/1     Running   0           14s   10.244.85.199   k8s-node01     <none>           <none>
my-nginx-5f7668c4b-8rj67            1/1     Running   0           14s   10.244.58.200   k8s-node02     <none>           <none>
[root@k8s-master01 ~]#
```

## (2) 创建并更新 service 资源清单文件:

```
[root@k8s-master01 ~]# vi service_NodePort.yaml
apiVersion: v1
kind: Service
metadata:
  name: my-nginx
  labels:
    run: my-nginx
spec:
  type: NodePort
  ports:
    - port: 80
      protocol: TCP
      targetPort: 80
      nodePort: 30380
  selector:
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**



版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

```
run: my-nginx
```

```
[root@k8s-master01 ~]# kubectl apply -f service_NodePort.yaml
service/my-nginx created
```

### (3) 查看 service

```
[root@k8s-master01 ~]# kubectl get svc my-nginx
```

```
[root@k8s-master01 ~]# kubectl get svc my-nginx
NAME         TYPE        CLUSTER-IP      EXTERNAL-IP  PORT(S)          AGE
my-nginx     NodePort    10.10.228.115   <none>       80:30380/TCP     23s
[root@k8s-master01 ~]#
```

### (4) 测试

# 集群节点访问测试:

```
[root@k8s-master01 ~]# curl 10.10.228.115
```

```
[root@k8s-master01 ~]# curl 10.10.228.115
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
[root@k8s-master01 ~]#
```

注意：10.10.228.115 是 k8s 集群内部的 service cluster ip 地址，只能在 k8s 集群内部访问，在集群外无法访问。

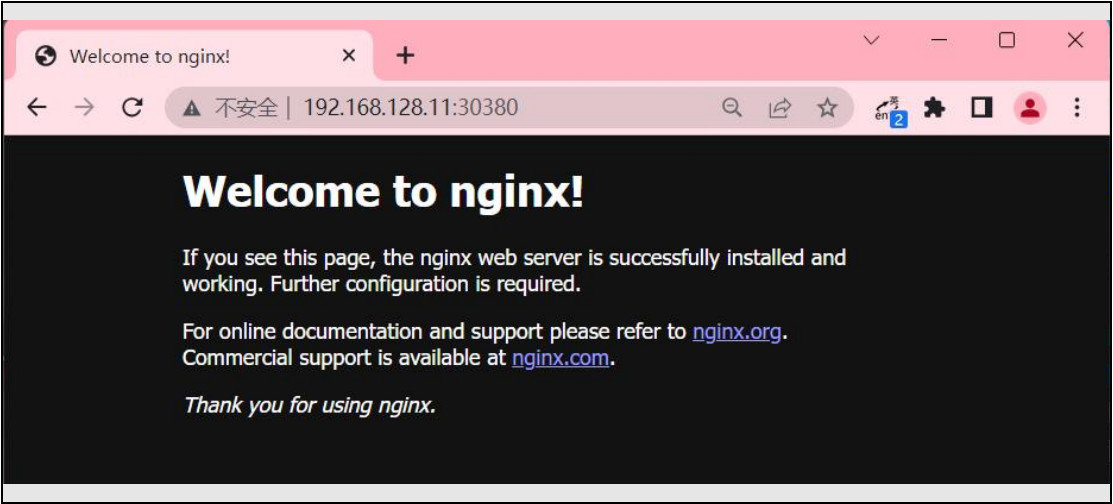
# 在集群外访问 service

提示：在集群外部访问 node 节点 IP+service 代理的端口，即可代理到 Pod 浏览器访问 192.168.128.11:30380，结果如下图：

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**



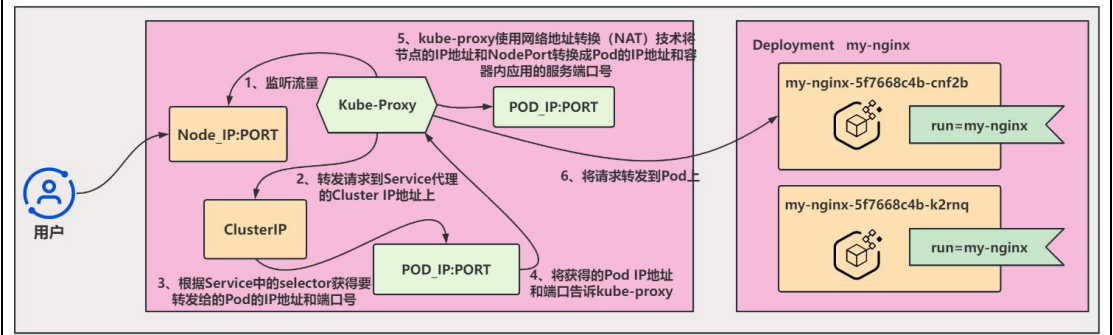
版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**



(5) 扩展：访问 192.168.128.11:30380 是如何将流量转发给后端 pod 的？

NodePort 类型的 Service 除了会创建一个 Cluster IP 地址外，还会分配一个静态端口号，供集群外部访问 Service。集群外部的请求将会先访问 Kubernetes 集群节点的该端口，然后被重定向到 Service 相应的 Pod 上。

具体来说，当 Kubernetes 集群中的某个节点的 kube-proxy 组件监听到 NodePort 端口的请求，它会转发请求到 Service 代理的 Cluster IP 地址，并根据 Service 中的 selector 获得要转发给的 Pod 的 IP 地址和端口号。之后，kube-proxy 会使用网络地址转换（NAT）技术将节点的 IP 地址和 NodePort 转换成 Pod 的 IP 地址和容器内应用的服务端口号，最后将请求转发到 Pod 上。如下图：



需要注意的是，如果有多个节点，流量到达每个节点上的 NodePort 端口时，都会转发到相同的 Pod 上。当 Pod 的 IP 地址发生变化时，节点上的 kube-proxy 组件会自动刷新其 iptables 规则，确保新的 Pod IP 地址可以正常被转发。

## 5、创建 Service：type 类型是 ExternalName

ExternalName 类型的 Service 是 Kubernetes 的一种特殊类型，它用于将 Service 映射到集群外的其他 DNS 名称。当 ExternalName Service 被创建时，它不会创建任何代理 Pod，而是仅仅创建一个 service 类型为 ExternalName 的

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

服务，并通过 `spec.externalName` 指定它映射的外部服务的 DNS 名称。

`ExternalName` 类型的 `Service` 是一种比较特殊的用法，它没有在 Kubernetes 集群内部代理 Pod，而是将服务映射到集群外部的服务。它主要用于一些与 Kubernetes 无关、独立于 Kubernetes 系统的服务，比如某些外部数据库、缓存等，这些服务可能并不是以 Pod 的形式运行在集群中，但是它们又需要被集群内部的其他组件调用。使用 `ExternalName Service` 就可以让 Kubernetes 内的其他组件可以直接通过 `Service` 名称（DNS 名称）访问这些外部服务，而不需要了解这些外部服务的详细信息。

举个例子，假设我们的集群中有一个数据库服务，这个数据库服务并不运行在 Kubernetes 中，而是在集群外某个主机上运行，并暴露了一个 DNS 域名。此时我们可以通过创建一个 `ExternalName` 类型的 `Service` 来将这个 DNS 域名映射到 `Service` 名称上，然后在 Kubernetes 内部的其他组件中直接通过 `Service` 名称访问这个数据库服务，而无需了解它的详细信息。

总之，`ExternalName` 类型的 `Service` 主要应用于需要将 Kubernetes 系统与外部服务连接起来的场合，通过简单的 DNS 名称映射，让 Kubernetes 内部可以统一访问和调用外部服务。

目标：

通过创建一个 `ExternalName` 类型的 `Service` 来将这个 `www.baidu.com` 域名映射到 `Service` 名称上，然后在 Kubernetes 内部的其他组件中直接通过 `Service` 名称访问 `www.baidu.com`。

### （1）创建 svc

```
[root@k8s-master01 ~]# vi baidu-svc.yaml
apiVersion: v1
kind: Service
metadata:
  name: baidu
spec:
  type: ExternalName
  externalName: www.baidu.com

[root@k8s-master01 ~]# kubectl apply -f baidu-svc.yaml
service/baidu created

[root@k8s-master01 ~]# kubectl get svc baidu
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
baidu	ExternalName	<none>	www.baidu.com	<none>	5s

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

## (2) 创建 pods, 进行测试

```
[root@k8s-master01 ~]# kubectl run busybox --image busybox:latest
--restart=Never --rm -it busybox -- sh
/ # ping baidu.default.svc.cluster.local
PING baidu.default.svc.cluster.local (39.156.66.18): 56 data bytes
64 bytes from 39.156.66.18: seq=0 ttl=127 time=17.997 ms
64 bytes from 39.156.66.18: seq=1 ttl=127 time=18.061 ms

/ # wget -q -O - https://baidu.default.svc.cluster.local
--no-check-certificate
wget: server returned error: HTTP/1.1 403 Forbidden
```

访问的时候被 403 拒绝了，那是因为百度服务器不允许我们这样做代理。

## 6、无标签选择器的 Service

创建一个 service, 不使用标签选择器, 通过创建 endpoints 指定 IP 和 PORT, 然后和 SVC 进行绑定。

实战：k8s 集群引用外部的 mysql 数据库

### (1) 在 k8s-node01 节点上安装 mysql 数据库：

```
[root@k8s-node01 ~]# yum install mariadb-server -y
[root@k8s-node01 ~]# systemctl restart mariadb
[root@k8s-node01 ~]# netstat -tln | grep 3306
tcp        0      0 0.0.0.0:3306          0.0.0.0:*            LISTEN

[root@k8s-node01 ~]# mysql
MariaDB [(none)]> create user root@'192.168.%.%' identified by
'123456';
Query OK, 0 rows affected (0.001 sec)

MariaDB [(none)]> grant all on *.* to root@'192.168.%.%' identified
by '123456';
Query OK, 0 rows affected (0.000 sec)
```

### (2) 在 master 节点创建 yaml 文件：

```
[root@k8s-master01 ~]# vi mysql_service.yaml
apiVersion: v1
kind: Service
metadata:
  name: mysql
spec:
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

```
type: ClusterIP
ports:
- port: 3306
```

### (3) 更新资源清单文件

```
[root@k8s-master01 ~]# kubectl apply -f mysql_service.yaml
service/mysql created
```

### (4) 查看创建的 mysql SVC

```
[root@k8s-master01 ~]# kubectl get svc | grep mysql
mysql      ClusterIP   10.102.14.71   <none>      3306/TCP    19s

[root@k8s-master01 ~]# kubectl describe svc mysql
Name:                mysql
Namespace:           default
Labels:              <none>
Annotations:         <none>
Selector:            <none>
Type:                ClusterIP
IP Families:         <none>
IP:                  10.102.14.71
IPs:                 10.102.14.71
Port:                <unset> 3306/TCP
TargetPort:          3306/TCP
Endpoints:           <none> # 还没有 endpoint
Session Affinity:    None
Events:              <none>
```

### (5) 创建 Endpoints 的 yaml 文件：

注意 endpoints 的名称必须和 service 的名称对应起来。

```
[root@k8s-master01 ~]# vi mysql_endpoint.yaml
apiVersion: v1
kind: Endpoints
metadata:
  name: mysql
subsets:
- addresses:
  - ip: 192.168.128.11
  ports:
  - port: 3306
```

对上面的 yaml 文件说明：

创建一个名为 mysql 的 Endpoints 对象的示例，它将一个或多个 IP 地址和端口映射到

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**

了一个命名的 mysql 服务上。这个 mysql 服务可以在集群内部被其它 Pod 的容器使用，进而访问这些 IP 地址上的服务。

该清单表明了 this Endpoints 对象的名称是 mysql，它只有一个子集 (subset)，包含了一个 IP 地址 192.168.128.11 和一个端口 3306。

其中，metadata 字段包含了 Endpoints 对象的元数据信息，例如名称、标签、注释等等；subsets 字段是 Endpoints 对象的核心，它包含了 Endpoints 对象映射的 IP 地址和端口信息，可以有多个子集，每个子集可以映射多个 IP 地址和端口。

#### (6) 更新资源清单文件

```
[root@k8s-master01 ~]# kubectl apply -f mysql_endpoint.yaml
endpoints/mysql created
```

#### (7) 查看创建的 mysql SVC:

```
[root@k8s-master01 ~]# kubectl describe svc mysql
Name:                mysql
Namespace:            default
Labels:               <none>
Annotations:          <none>
Selector:             <none>
Type:                 ClusterIP
IP Families:          <none>
IP:                   10.102.14.71
IPs:                  10.102.14.71
Port:                 <unset> 3306/TCP
TargetPort:           3306/TCP
Endpoints:            192.168.128.11:3306 #这个就是定义的外部数据库
Session Affinity:     None
Events:               <none>
```

上面配置就是将外部 IP 地址和服务引入到 k8s 集群内部，由 service 作为一个代理来达到能够访问外部服务的目的。

#### (8) 在集群节点上进行测试:

```
[root@k8s-node01 ~]# mysql -h 10.102.14.71
MariaDB [(none)]>
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，**禁止私自传阅，违者依法追责。**