41 使用 Service 访问一组特定的 Pod

更新时间：2020-10-30 09:52:37

> 成功的奥秘在于目标的坚定。——迪斯雷利

试想这么一种场景，我们的应用程序都通过 Deployment 来管理，Deployment 后端管理了一组 Pod，每个 Pod 都有自己的 IP 地址。而且对于 Deployment 这种模式，Pod 挂掉之后 Deployment 会重新启动一个新的 Pod。

这就引入了一个问题，如果其他应用想要访问这个 Deployment 提供的服务，直接去访问 Pod 肯定是不行的，那么有没有一种类似服务发现的机制帮助我们做这件事情呢？

1. Service 介绍

针对上面说的这个问题，Kubernetes 提供了一种 API 对象叫做 Service。Service 可以理解为一种访问一组特定 Pod 的策略。

举个例子，考虑一个图片处理应用程序，通过 Pod 运行了 3 个副本，并且是无状态的。前端访问该应用程序时，不需要关心实际是调用了那个 Pod 实例。后端的 Pod 发生重启时，前端不应该也不需要感知到。对于这种解耦关系，我们就可以通过 Service 来做。Service 与后端的多个 Pod 进行关联（通过 selector），前端只需要访问 Service 即可。

2. 创建 Service

在 Kubernetes 中，Service 对象也可以通过一个 yaml 文件来定义，下面就是一个简单 Service 定义。

```yaml
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
spec:
  selector:
    app: nginx
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
```

这个 Service 对象做的事情也比较简单，创建一个名称为 my-service 的 Service 对象，它会将对 80 端口的 TCP 请求转发到一组 Pod 上，这些 Pod 的特点是被打上标签 `app=nginx`，并且使用 TCP 端口 80。这些 Pod 我们暂时还没有创建，我们先把这个 Service 通过 `kubectl apply` 创建出来。

```
$ kubectl apply -f nginx-service.yaml -n imooc
service/nginx-service created
```

同样的，我们通过 `kubectl describe service` 来查看一下我们创建出来的 Service 对象。

```
$ kubectl describe service nginx-service -n imooc
Name:              nginx-service
Namespace:         imooc
Labels:            <none>
Annotations:       kubectl.kubernetes.io/last-applied-configuration:
                   {"apiVersion":"v1","kind":"Service","metadata":{"annotations":{},"name":"nginx-service","namespace":"imooc"},"spec":{"ports":[{"port":80,"...
Selector:          app=nginx
Type:              ClusterIP
IP:                10.0.213.149
Port:              <unset>  80/TCP
TargetPort:        80/TCP
Endpoints:         <none>
Session Affinity:  None
Events:            <none>
```

这里有几个关键的信息，包括：

- selector：Service 会根据 selector 条件去选择 label 满足条件的 Pod 进行请求转发；
- Type: Service 的类型，这里是 ClusterIP 类似，也是默认的类型。简单来说，ClusterIP 类型会分配一个固定的 IP，然后只能通过集群内部进行访问；
- IP：Service 对象分配的 IP，可以认为是一个 vip；
- Port/TargetPort：前者是 Service 对象监听的端口，后者是转发的目标 Pod 的端口；
- Endpoints：是一个列表，表示转发到后端的 Pod 的 IP 集合。

这其中比较重要的一个点就是 endpoints，因为现在集群内没有满足条件的 Pod 可以供转发，所以 endpoints 字段目前为空。

3. 请求转发

下面我们创建一组满足条件的 nginx 的 Pod：具有 label `app=nginx` 和使用端口 80。下面就是我们的 Deployment 的定义。

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.9.1
        ports:
        - containerPort: 80
        resources:
          limits:
            cpu: 100m
            memory: 200Mi
          requests:
            cpu: 100m
            memory: 200Mi
```

通过 `kubectl apply` 创建该 Deployment。

```
$ kubectl apply -f nginx-dm.yaml -n imooc
deployment.apps/nginx-deployment created
```

我们通过 `kubectl get pods` 看一下改 deployment 创建的 pod 情况，通过 `-o wide` 参数可以显示更多的字段，比如 IP，节点名称，我们这里主要是为了查看 IP，所以其他字段域暂时先隐藏掉。记住下面的几个 Pod 的 IP。

```
kubectl get pods -n imooc -o wide | grep nginx
nginx-deployment-c464767dd-6ts4x   1/1     Running     0          85s     10.1.1.154
nginx-deployment-c464767dd-d9mh7   1/1     Running     0          85s     10.1.2.159
nginx-deployment-c464767dd-qd22h   1/1     Running     0          85s     10.1.2.31
```

我们现在再回过头来查看一下之前创建的 Service 对象。如下所示，我们可以看到其中的 Endpoints 字段域不再为空了，而是上面的三个 Pod 的 IP:Port 集合。

```
$ kubectl describe service nginx-service -n imooc
Name:              nginx-service
Namespace:         imooc
Labels:            <none>
Annotations:       kubectl.kubernetes.io/last-applied-configuration:
                     {"apiVersion":"v1","kind":"Service","metadata":{"annotations":{},"name":"nginx-service","namespace":"imooc"},"spec":{"ports":[{"port":80,"...
Selector:          app=nginx
Type:              ClusterIP
IP:                10.0.213.149
Port:              <unset>  80/TCP
TargetPort:        80/TCP
Endpoints:         10.1.1.154:80,10.1.2.159:80,10.1.2.31:80
Session Affinity:  None
Events:            <none>
```

实际上，Service 对象会创建一个 endpoints 对象，我们可以通过 `kubectl get endpoints` 来查看。

```
$ kubectl get endpoints -n imooc
NAME            ENDPOINTS                                      AGE
nginx-service   10.1.1.154:80,10.1.2.159:80,10.1.2.31:80   113m
$ kubectl describe endpoints nginx-service -n imooc
Name:         nginx-service
Namespace:    imooc
Labels:       <none>
Annotations:  endpoints.kubernetes.io/last-change-trigger-time: 2020-04-19T12:55:17+08:00
Subsets:
  Addresses:         10.1.1.154,10.1.2.159,10.1.2.31
  NotReadyAddresses:  <none>
  Ports:
    Name     Port  Protocol
    ----     ----  --------
    <unset>  80    TCP

Events:  <none>
```

现在我们已经创建出来了后端应用，我们看一下请求是如何进行转发的。所有到 Service IP 的 80 端口的请求都会被转发到后端的三个 Pod 中的一个，转发到哪个 Pod 对应到不同的负载均衡策略。还有一点需要注意的是，ClusterIP 类型的 Service 只能在集群内部进行访问。如下所示，我们直接访问 Service IP 对应的端口 80，直接返回了 Nginx 的欢迎页面，也就是转发到了运行 nginx 的 Pod 中了。

```
$ curl 10.0.213.149:80
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    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>
```

4. 多端口 Service

有时候我们会为同一个应用分配多个端口，比如开放 http 端口 80，开放 https 端口 443，我们同样可以在 Service 对象中配置多个端口。但是需要注意的是，当使用多个端口时，必须提供所有端口名称，以他们无歧义。端口名称只能包含小写字母数字字符和中划线，并且必须以字母数字字符开头和结尾。如下是一个多端口 Service 的定义描述。

```
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  selector:
    app: MyApp
  ports:
    - name: http
      protocol: TCP
      port: 80
      targetPort: 9376
    - name: https
      protocol: TCP
      port: 443
      targetPort: 9377
```

5. 设置固定 IP

前面的 Service 都是分配了随机 IP，随机 IP 在 ApiServer 的启动参数 `service-cluster-ip-range` 的 CIDR 范围内。

如果我们想要对 IP 有更强的掌控力，那么我们可以在 Service 的定义中通过参数 `spec.clusterIP` 指定自己的 clusterIP，比如希望替换一个已存在的 DNS 条目，或者遗留系统中已经配置了一个固定 IP 并且修改起来比较麻烦。

6. 服务发现

Kubernetes 提供了两种服务发现模式：环境变量和 DNS。

环境变量

环境变量的方式指的是 Kubernetes 会将集群中的 Service 对象以环境变量的方式注入到 Pod 中，形如 `{SVCNAME}_SERVICE_HOST` 和 `{SVCNAME}_SERVICE_PORT` 。

举个例子，一个 redis 实例 `redis-master` 的 Service 暴露了 TCP 端口 6379，同时分配了 ClusterIP 地址 10.0.0.11，对应的环境变量如下。

```
REDIS_MASTER_SERVICE_HOST=10.0.0.11
REDIS_MASTER_SERVICE_PORT=6379
REDIS_MASTER_PORT=tcp://10.0.0.11:6379
REDIS_MASTER_PORT_6379_TCP=tcp://10.0.0.11:6379
REDIS_MASTER_PORT_6379_TCP_PROTO=tcp
REDIS_MASTER_PORT_6379_TCP_PORT=6379
REDIS_MASTER_PORT_6379_TCP_ADDR=10.0.0.11
```

环境变量这种方式有个比较明显的弊端：环境变量不会自动更新。如果 Service 在 Pod 启动之后才创建成功，那么这个 Service 在该 Pod 内的环境变量中是找不到的。

DNS

Kubernetes 集群的 DNS 服务器，比如 CoreDNS，会监控集群中的新服务，并为每个服务创建一组 DNS 记录。如果整个集群中都启用了 DNS，则所有的 Pod 都应该能够通过其 DNS 名称自动解析服务。

举个例子，如果在 Namespace `my-ns` 中有一个名称为 `my-svc` 的服务，则 DNS 服务器会为该服务创建一个 DNS 条目 `my-svc.my-ns` 。位于 Namespace `my-ns` 下的 Pod 则可以通过名称 `my-svc` 或者 `my-svc.my-ns` 来进行服务发现。其他 Namespace 下的 Pod 则可以通过 `my-svc.my-ns` 来进行服务发现。

7. Headless Service

对于拥有 ClusterIP 的 Service，当我们访问其 ClusterIP 时，其会自动为我们做负载均衡。但是有的时候我们想要嵌入我们自己的负载均衡策略，那么对于这种情况，可以通过指定 ClusterIP 的值为 `None` ，这个时候创建出来的 Service 则为 `Headless` Service。我们在做服务发现时，这个 Service 返回的为后端的 Pod 列表，这个时候我们就可以灵活发挥了。下面举个例子。

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
spec:
  clusterIP: None
  selector:
    app: nginx
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
```

这个是一个很简单的 Headless Service，后端代理了多个具有 label: app=nginx 的 Pod。我们将该 Service 进行部署，然后在 Kubernetes 集群中的某个 Pod 内部通过 nslookup 来查询该服务。

```
/ $ nslookup nginx-service

Name:      nginx-service
Address 1: 10.1.1.154 10-1-1-154.nginx-service.imooc.svc.cluster.local
Address 2: 10.1.2.31 10-1-2-31.nginx-service.imooc.svc.cluster.local
Address 3: 10.1.2.159 10-1-2-159.nginx-service.imooc.svc.cluster.local
```

我们可以看到服务发现时候，直接将后端的 Pod 列表返回了，每个 Pod 对应的 DNS 条目为 ip.<svc-name>.<namespace>.svc.cluster.local，这个时候我们就可以根据需求来做进一步操作了。

8. 总结

本篇文章介绍了 Kubernetes 中的 API 对象 Service 的基本情况，下一篇文章将会和大家介绍 Kubernetes 提供的多种 Service 类型。

}

40 Kubernetes 有状态应用管理 StatefulSet

42 Kubernetes Service 类型