

39 | 谈谈Service与Ingress

39 | 谈谈Service与Ingress

张磊 2018-11-21



□

10:08

讲述：张磊 大小：4.65M

你好，我是张磊。今天我和你分享的主题是：谈谈 Service 与 Ingress。

在上一篇文章中，我为你详细讲解了将 Service 暴露给外界的三种方法。其中有一个叫作 LoadBalancer 类型的 Service，它会为你在 Cloud Provider（比如：Google Cloud 或者 OpenStack）里创建一个与该 Service 对应的负载均衡服务。

但是，相信你也应该能感受到，由于每个 Service 都要有一个负载均衡服务，所以这个做法实际上既浪费成本又高。作为用户，我其实更希望看到 Kubernetes 为我内置一个全局的负载均衡器。然后，通过我访问的 URL，把请求转发给不同的后端 Service。

这种全局的、为了代理不同后端 Service 而设置的负载均衡服务，就是 Kubernetes 里的 Ingress 服务。

所以，Ingress 的功能其实很容易理解：**所谓 Ingress，就是 Service 的 “Service”。**

举个例子，假如我现在有这样一一个站点：https://cafe.example.com。其中，https://cafe.example.com/coffee，对应的是“咖啡点餐系统”。而，https://cafe.example.com/tea，对应的则是“茶水点餐系统”。这两个系统，分别由名叫 coffee 和 tea 这样两个 Deployment 来提供服务。

那么现在，我如何能使用 Kubernetes 的 Ingress 来创建一个统一的负载均衡器，从而实现当用户访问不同的域名时，能够访问到不同的 Deployment 呢？

上述功能，在 Kubernetes 里就需要通过 Ingress 对象来描述，如下所示：

```
apiVersion: extensions/v1beta1
```

```
kind: Ingress
```

```
metadata:
```

```
name: cafe-ingress
```

```
spec:
```

```
tls:
```

```
- hosts:
```

```
- cafe.example.com
```

```
secretName: cafe-secret
```

```
rules:
```

```
- host: cafe.example.com
```

```
http:
```

```
paths:
```

```
- path: /tea
```

```
backend:
```

```
serviceName: tea-svc
```

```
servicePort: 80
```

```
- path: /coffee
```

```
backend:
```

```
serviceName: coffee-svc
```

servicePort: 80

□复制代码

在上面这个名叫 `cafe-ingress.yaml` 文件中，最值得我们关注的，是 `rules` 字段。在 Kubernetes 里，这个字段叫作：**IngressRule**。

IngressRule 的 Key，就叫做：`host`。它必须是一个标准的域名格式（Fully Qualified Domain Name）的字符串，而不能是 IP 地址。

备注：Fully Qualified Domain Name 的具体格式，可以参考[RFC 3986](#) 标准。

而 `host` 字段定义的值，就是这个 Ingress 的入口。这也就意味着，当用户访问 `cafe.example.com` 的时候，实际上访问到的是这个 Ingress 对象。这样，Kubernetes 就能使用 IngressRule 来对你的请求进行下一步转发。

而接下来 IngressRule 规则的定义，则依赖于 `path` 字段。你可以简单地理解为，这里的每一个 `path` 都对应一个后端 Service。所以在我们的例子里，我定义了两个 `path`，它们分别对应 `coffee` 和 `tea` 这两个 Deployment 的 Service（即：`coffee-svc` 和 `tea-svc`）。

通过上面的讲解，不难看到，所谓 Ingress 对象，其实就是 Kubernetes 项目对“反向代理”的一种抽象。

一个 Ingress 对象的主要内容，实际上就是一个“反向代理”服务（比如：Nginx）的配置文件的描述。而这个代理服务对应的转发规则，就是 IngressRule。

这就是为什么在每条 IngressRule 里，需要有一个 `host` 字段来作为这条 IngressRule 的入口，然后还需要有一系列 `path` 字段来声明具体的转发策略。这其实跟 Nginx、HAProxy 等项目的配置文件的写法是一致的。

而有了 Ingress 这样一个统一的抽象，Kubernetes 的用户就无需关心 Ingress 的具体细节了。

在实际的使用中，你只需要从社区里选择一个具体的 Ingress Controller，把它部署在 Kubernetes 集群里即可。

然后，这个 Ingress Controller 会根据你定义的 Ingress 对象，提供对应的代理能力。目前，业界常用的各种反向代理项目，比如 Nginx、HAProxy、Envoy、Traefik 等，都已经为 Kubernetes 专门维护了对应的 Ingress Controller。

接下来，我就以最常用的 Nginx Ingress Controller 为例，在我们前面用 `kubeadm` 部署的 Bare-metal 环境中，和你实践一下 Ingress 机制的使用过程。

部署 Nginx Ingress Controller 的方法非常简单，如下所示：

```
$ kubectl apply -f
https://raw.githubusercontent.com/kubernetes/ingress-
nginx/master/deploy/mandatory.yaml
```

□复制代码

其中，在[mandatory.yaml](#)这个文件里，正是 Nginx 官方为你维护的 Ingress Controller 的定义。我们来看一下它的内容：

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: nginx-configuration
  namespace: ingress-nginx
  labels:
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/part-of: ingress-nginx
---
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: nginx-ingress-controller
  namespace: ingress-nginx
  labels:
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/part-of: ingress-nginx
spec:
  replicas: 1
  selector:
    matchLabels:
      app.kubernetes.io/name: ingress-nginx
      app.kubernetes.io/part-of: ingress-nginx
  template:
    metadata:
```

labels:

app.kubernetes.io/name: ingress-nginx

app.kubernetes.io/part-of: ingress-nginx

annotations:

...

spec:

serviceAccountName: nginx-ingress-serviceaccount

containers:

- name: nginx-ingress-controller

image: quay.io/kubernetes-ingress-controller/nginx-ingress-controller:0.20.0

args:

- /nginx-ingress-controller

- --configmap=\$(POD_NAMESPACE)/nginx-configuration

- --publish-service=\$(POD_NAMESPACE)/ingress-nginx

- --annotations-prefix=nginx.ingress.kubernetes.io

securityContext:

capabilities:

drop:

- ALL

add:

- NET_BIND_SERVICE

www-data -> 33

runAsUser: 33

env:

- name: POD_NAME

valueFrom:

fieldRef:

fieldPath: metadata.name

- name: POD_NAMESPACE

- name: http

valueFrom:

fieldRef:

```
fieldPath: metadata.namespace
```

```
ports:
```

```
- name: http
```

```
containerPort: 80
```

```
- name: https
```

```
containerPort: 443
```

□复制代码

可以看到，在上述 YAML 文件中，我们定义了一个使用 nginx-ingress-controller 镜像的 Pod。需要注意的是，这个 Pod 的启动命令需要使用该 Pod 所在的 Namespace 作为参数。而这个信息，当然是通过 Downward API 拿到的，即：Pod 的 env 字段里的定义（env.valueFrom.fieldRef.fieldPath）。

而这个 Pod 本身，就是一个监听 Ingress 对象以及它所代理的后端 Service 变化的控制器。

当一个新的 Ingress 对象由用户创建后，nginx-ingress-controller 就会根据 Ingress 对象里定义的内容，生成一份对应的 Nginx 配置文件（/etc/nginx/nginx.conf），并使用这个配置文件启动一个 Nginx 服务。

而一旦 Ingress 对象被更新，nginx-ingress-controller 就会更新这个配置文件。需要注意的是，如果这里只是被代理的 Service 对象被更新，nginx-ingress-controller 所管理的 Nginx 服务是不需要重新加载（reload）的。这当然是因为 nginx-ingress-controller 通过[Nginx Lua](#)方案实现了 Nginx Upstream 的动态配置。

此外，nginx-ingress-controller 还允许你通过 Kubernetes 的 ConfigMap 对象来对上述 Nginx 配置文件进行定制。这个 ConfigMap 的名字，需要以参数的方式传递给 nginx-ingress-controller。而你在这个 ConfigMap 里添加的字段，将会被合并到最后生成的 Nginx 配置文件当中。

可以看到，一个 Nginx Ingress Controller 为你提供的服务，其实是一个可以根据 Ingress 对象和被代理后端 Service 的变化，来自动进行更新的 Nginx 负载均衡器。

当然，为了让用户能够用到这个 Nginx，我们就需要创建一个 Service 来把 Nginx Ingress Controller 管理的 Nginx 服务暴露出去，如下所示：

```
$ kubectl apply -f
https://raw.githubusercontent.com/kubernetes/ingress-
nginx/master/deploy/provider/baremetal/service-nodeport.yaml
```

□复制代码

由于我们使用的是 Bare-metal 环境，所以 service-nodeport.yaml 文件里的内容，就是一个 NodePort 类型的 Service，如下所示：

```
apiVersion: v1
kind: Service
metadata:
  name: ingress-nginx
  namespace: ingress-nginx
  labels:
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/part-of: ingress-nginx
spec:
  type: NodePort
  ports:
    - name: http
      port: 80
      targetPort: 80
      protocol: TCP
    - name: https
      port: 443
      targetPort: 443
      protocol: TCP
  selector:
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/part-of: ingress-nginx
```

□复制代码

可以看到，这个 Service 的唯一工作，就是将所有携带 ingress-nginx 标签的 Pod 的 80 和 443 端口暴露出去。

而如果你是公有云上的环境，你需要创建的就是 *LoadBalancer* 类型的 *Service* 了。

上述操作完成后，你一定要记录下这个 Service 的访问入口，即：宿主机的地址和 NodePort 的端口，如下所示：

```
$ kubectl get svc -n ingress-nginx
```

```
NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE
```

```
ingress-nginx NodePort 10.105.72.96 <none>
```

```
80:30044/TCP,443:31453/TCP 3h
```

❏复制代码

为了后面方便使用，我会把上述访问入口设置为环境变量：

```
$ IC_IP=10.168.0.2 # 任意一台宿主机的地址
```

```
$ IC_HTTPS_PORT=31453 # NodePort 端口
```

❏复制代码

在 Ingress Controller 和它所需要的 Service 部署完成后，我们就可以使用它了。

■ 备注：这个“咖啡厅” Ingress 的所有示例文件，都在[这里](#)。

首先，我们要在集群里部署我们的应用 Pod 和它们对应的 Service，如下所示：

```
$ kubectl create -f cafe.yaml
```

❏复制代码

然后，我们需要创建 Ingress 所需的 SSL 证书 (tls.crt) 和密钥 (tls.key)，这些信息都是通过 Secret 对象定义好的，如下所示：

```
$ kubectl create -f cafe-secret.yaml
```

❏复制代码

这一步完成后，我们就可以创建在本篇文章一开始定义的 Ingress 对象了，如下所示：

```
$ kubectl create -f cafe-ingress.yaml
```

❏复制代码

这时候，我们就可以查看一下这个 Ingress 对象的信息，如下所示：

```
$ kubectl get ingress
```

```
NAME HOSTS ADDRESS PORTS AGE
```

```
cafe-ingress cafe.example.com 80, 443 2h
```

```
$ kubectl describe ingress cafe-ingress
```

```
Name: cafe-ingress
```

```
Namespace: default
```

```
Address:
```

```
Default backend: default-http-backend:80 (<none>)
```


TLS:

cafe-secret terminates cafe.example.com

Rules:

Host Path Backends

cafe.example.com

/tea tea-svc:80 (<none>)

/coffee coffee-svc:80 (<none>)

Annotations:

Events:

Type Reason Age From Message

Normal CREATE 4m nginx-ingress-controller Ingress default/cafe-ingress

□复制代码

可以看到，这个 Ingress 对象最核心的部分，正是 Rules 字段。其中，我们定义的 Host 是cafe.example.com，它有两条转发规则（Path），分别转发给 tea-svc 和 coffee-svc。

当然，在 Ingress 的 YAML 文件里，你还可以定义多个 Host，比如 restaurant.example.com、movie.example.com等等，来为更多的域名提供负载均衡服务。

接下来，我们就可以通过访问这个 Ingress 的地址和端口，访问到我们前面部署的应用了，比如，当我们访问https://cafe.example.com:443/coffee时，应该是 coffee 这个 Deployment 负责响应我的请求。我们可以来尝试一下：

```
$ curl --resolve cafe.example.com:$IC_HTTPS_PORT:$IC_IP
https://cafe.example.com:$IC_HTTPS_PORT/coffee --insecureServer
address: 10.244.1.56:80
```

Server name: coffee-7dbb5795f6-vglbv

Date: 03/Nov/2018:03:55:32 +0000

URI: /coffee

Request ID: e487e672673195c573147134167cf898

□复制代码

我们可以看到，访问这个 URL 得到的返回信息是：Server name: coffee-7dbb5795f6-vglbv。这正是 coffee 这个 Deployment 的名字。

而当我访问`https://cafe.example.com:433/tea`的时候，则应该是 tea 这个 Deployment 负责响应我的请求（Server name: tea-7d57856c44-lwbnp），如下所示：

```
$ curl --resolve cafe.example.com:$IC_HTTPS_PORT:$IC_IP
https://cafe.example.com:$IC_HTTPS_PORT/tea --insecure

Server address: 10.244.1.58:80

Server name: tea-7d57856c44-lwbnp

Date: 03/Nov/2018:03:55:52 +0000

URI: /tea

Request ID: 32191f7ea07cb6bb44a1f43b8299415c
```

□复制代码

可以看到，Nginx Ingress Controller 为我们创建的 Nginx 负载均衡器，已经成功地将请求转发给了对应的后端 Service。

以上，就是 Kubernetes 里 Ingress 的设计思想和使用方法了。

不过，你可能会有一个疑问，**如果我的请求没有匹配到任何一条 IngressRule，那么会发生什么呢？**

首先，既然 Nginx Ingress Controller 是用 Nginx 实现的，那么它当然会为你返回一个 Nginx 的 404 页面。

不过，Ingress Controller 也允许你通过 Pod 启动命令里的 `-default-backend-service` 参数，设置一条默认规则，比如：`-default-backend-service=nginx-default-backend`。

这样，任何匹配失败的请求，就都会被转发到这个名叫 `nginx-default-backend` 的 Service。所以，你就可以通过部署一个专门的 Pod，来为用户返回自定义的 404 页面了。

总结

在这篇文章里，我为你详细讲解了 Ingress 这个概念在 Kubernetes 里到底是怎么回事儿。正如我在文章里所描述的，Ingress 实际上就是 Kubernetes 对“反向代理”的抽象。

目前，Ingress 只能工作在七层，而 Service 只能工作在四层。所以当你想要在 Kubernetes 里为应用进行 TLS 配置等 HTTP 相关的操作时，都必须通过 Ingress 来进行。

当然，正如同很多负载均衡项目可以同时提供七层和四层代理一样，将来 Ingress 的进化中，也会加入四层代理的能力。这样，一个比较完善的“反向代理”机制就比较成熟了。

而 Kubernetes 提出 Ingress 概念的原因其实也非常容易理解，有了 Ingress 这个抽象，用户就可以根据自己的需求来自由选择 Ingress Controller。比如，如果你的应用对代理服务的中断非常敏感，那么你就应该考虑选择类似于 Traefik 这样支持“热加载”的 Ingress Controller 实现。

更重要的是，一旦你对社区里现有的 Ingress 方案感到不满意，或者你已经有了自己的负载均衡方案时，你只需要做很少的编程工作，就可以实现一个自己的 Ingress Controller。

在实际的生产环境中，Ingress 带来的灵活度和自由度，对于使用容器的用户来说，其实是非常有意义的。要知道，当年在 Cloud Foundry 项目里，不知道有多少人为了给 Gorouter 组件配置一个 TLS 而伤透了脑筋。

思考题

如果我的需求是，当访问 `www.mysite.com` 和 `forums.mysite.com` 时，分别访问到不同的 Service（比如：`site-svc` 和 `forums-svc`）。那么，这个 Ingress 该如何定义呢？请你描述出 YAML 文件中的 `rules` 字段。