链滴

kubernetes 网络简介（下）-Ingress

作者：Leif160519
原文链接：https://ld246.com/article/1592117871540
来源网站：链滴

一、Ingress为弥补NodePort不足而生

NodePort存在的不足：

- 一个端口只能一个服务使用，端口需提前规划
- 只支持4层负载均衡（iptables和ipvs）

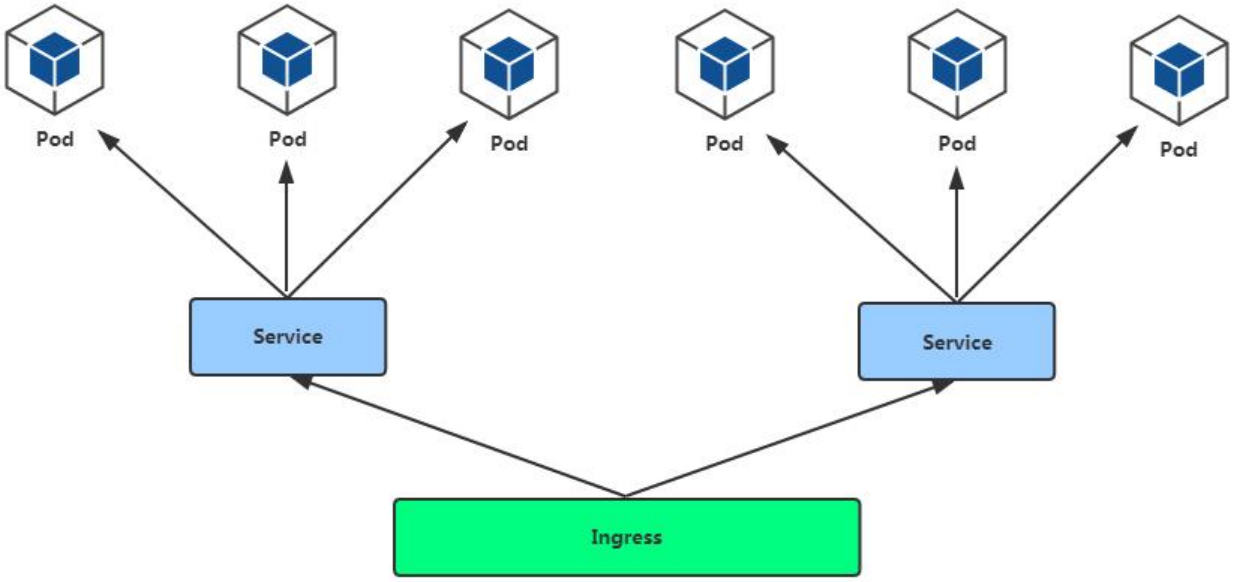
扩展：

- 四层：基于IP和端口转发
- 七层：HTTP协议，例如url、cookie

二、Pod与Ingress的关系

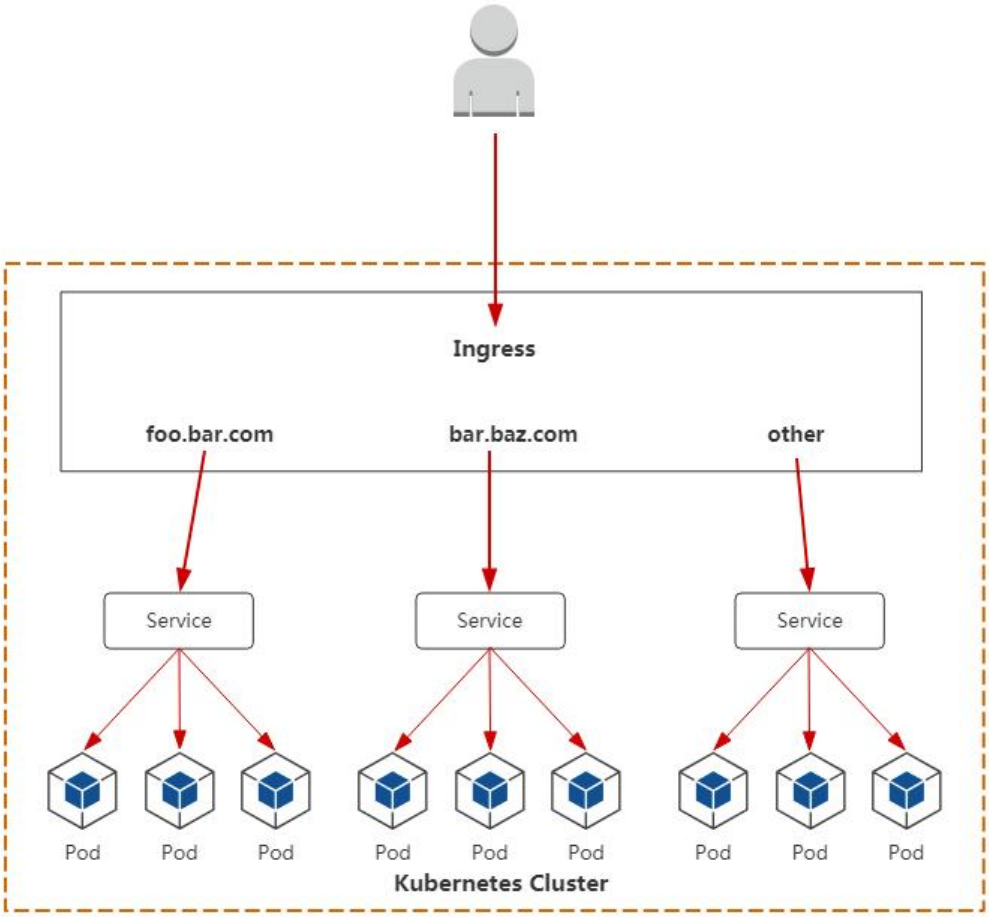
- 通过Service相关联
- 通过 Ingress Controller（类似iptables的机制）实现Pod的负载均衡

a.支持TCP/UDP 4层和HTTP 7层



Ingress与Nodeport同级。

三、Ingress Controller



实现：

原文链接：kubernetes 网络简介（下）-Ingress

- 部署Ingress Controller
- 创建Ingress规则

注意：由于kube-proxy借助了操作系统现有的机制实现了负载均衡，但是Ingress Controller技术在Linux内核中没有集成，所以需要单独部署；创建Ingress规则就好比创建一个Service规则

Ingress Controller有很多实现，我们这里采用官方维护的Nginx控制器。

Github：https://github.com/kubernetes/ingress-nginx

部署：

kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-nginx/nginx-0.30.0/deploy/static/mandatory.yaml

注意事项：

- 镜像地址修改成国内的： lizhenliang/nginx-ingress-controller:0.30.0
- 建议直接宿主机网络暴露： hostNetwork: true

注意:也可以下载本站资源ingress.zip

kubectl apply -f ingress-controller.yaml

```
    spec:
      hostNetwork: true
      # wait up to five minutes for the drain of connections
      terminationGracePeriodSeconds: 300
      serviceAccountName: nginx-ingress-serviceaccount
      nodeSelector:
        kubernetes.io/os: linux
      containers:
        - name: nginx-ingress-controller
          image: lizhenliang/nginx-ingress-controller:0.30.0
          args:
            - /nginx-ingress-controller
            - --configmap=$(POD_NAMESPACE)/nginx-configuration
            - --tcp-services-configmap=$(POD_NAMESPACE)/tcp-services
            - --udp-services-configmap=$(POD_NAMESPACE)/udp-services
            - --publish-service=$(POD_NAMESPACE)/ingress-nginx
            - --annotations-prefix=nginx.ingress.kubernetes.io
          securityContext:
```

参数解释：

- --configmap：配置文件路径
- --tcp-services-configmap：tcp转发服务的配置文件路径
- --udp-services-configmap：upd转发服务的配置文件路径
- --publish-service：Ingress-service配置文件路径
- --annotations-prefix：Ingress自身注解

其他主流控制器：

- Traefik：HTTP反向代理、负载均衡工具(类似nginx)
- Istio：服务治理，控制入口流量

```
[root@k8s-master ingress]# kubectl get deploy -n ingress-nginx
NAME                       READY   UP-TO-DATE   AVAILABLE   AGE
nginx-ingress-controller   1/1     1            1           5m18s

[root@k8s-master ingress]# kubectl get pod -n ingress-nginx
NAME                                        READY   STATUS    RESTARTS   AGE
nginx-ingress-controller-766fb9f77-cjmvm   1/1     Running   0          71s
```

接下来就是暴露Ingress-Controller的端口号到宿主机中，这时有人会想到用Service的NodePort去露，但是这样的话会多走一层，性能会有所下降，数据包会多一层转发，流程如下：

user -> nodeport -> iptables/ipvs -> ingress controller(pod) -> [service] -> pod

更好的办法就是通过hostnetwork，去声明Pod使用宿主机的网络，而Pod默认是使用容器自己的网的，这样的话除了容器的网络，其他都处于隔离状态，让容器网络与宿主机网络处于同一命名空间中

user -> lb -> ingress controller(pod) -> [service] -> pod

Ingress-Controller的端口号已经固定，即80和443，可以在分配的节点机器上查看

四、Ingress暴露应用

4.1 http

yaml示例ingress.yaml：

```yaml
# http
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: example-ingress
spec:
  rules:
  - host: blog.ctnrs.com
    http:
      paths:
      - path: /
        backend:
          serviceName: web
          servicePort: 80
```

参数解释：

- host：域名，类似nginx中的server_name
- path：类似nginx中的location
- serviceName：deployment中对应的Service名称，暴露的项目名称
- servicePort：监听端口，类似nginx中的listen，Service的端口（内部端口）

```
server {
```

```
        listen       80 ;
        server_name  blog.ctnrs.com;
        location / {
        }
    }
```

```
kubectl apply -f ingress.yaml

[root@k8s-master ingress]# kubectl get ing
NAME              CLASS    HOSTS              ADDRESS   PORTS     AGE
example-ingress       <none>   blog.ctnrs.com             80        45s
```

因为Ingress是基于域名进行分流的，所以需要绑定host去访问（Ingress-Controller在哪个节点上，ost就绑定哪个节点和域名）

原理解释：因为部署的Ingress-Controller就好比单独部署的Nginx，Nginx使用的宿主机的网络，所Pod在哪个节点上，才会监听哪个节点的端口。

对应关系：

```
[root@k8s-master ingress]# kubectl get pod
NAME                    READY   STATUS    RESTARTS   AGE
web-5b9bff6674-8wjlq    1/1     Running   1          28h
web-5b9bff6674-99ltd    1/1     Running   1          28h
web-5b9bff6674-skgsj    1/1     Running   1          28h
[root@k8s-master ingress]# kubectl get deploy
NAME    READY   UP-TO-DATE   AVAILABLE   AGE
web     3/3     3            3           28h
[root@k8s-master ingress]# kubectl get svc
NAME          TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)        AGE
kubernetes    ClusterIP   10.96.0.1        <none>        443/TCP        28h
web           NodePort    10.101.205.162   <none>        80:30000/TCP   28h
[root@k8s-master ingress]# kubectl get ing
NAME                  CLASS    HOSTS                  ADDRESS   PORTS     AGE
example-ingress       <none>   example.ctnrs.com                80        11m
tls-example-ingress   <none>   sslexample.ctnrs.com             80, 443   11m
```

4.2 https

提前准备自签域名证书,先安装cfssl工具cfssl.sh：

```
wget https://pkg.cfssl.org/R1.2/cfssl_linux-amd64
wget https://pkg.cfssl.org/R1.2/cfssljson_linux-amd64
wget https://pkg.cfssl.org/R1.2/cfssl-certinfo_linux-amd64
chmod +x cfssl*
mv cfssl_linux-amd64 /usr/bin/cfssl
mv cfssljson_linux-amd64 /usr/bin/cfssljson
mv cfssl-certinfo_linux-amd64 /usr/bin/cfssl-certinfo
```

使用cfssl工具签发certs.sh：

```
cat > ca-config.json <<EOF
{
  "signing": {
    "default": {
      "expiry": "87600h"
    },
    "profiles": {
```

```
      "kubernetes": {
        "expiry": "87600h",
        "usages": [
          "signing",
          "key encipherment",
          "server auth",
          "client auth"
        ]
      }
    }
  }
}
EOF

cat > ca-csr.json <<EOF
{
    "CN": "kubernetes",
    "key": {
        "algo": "rsa",
        "size": 2048
    },
    "names": [
        {
            "C": "CN",
            "L": "Beijing",
            "ST": "Beijing"
        }
    ]
}
EOF

cfssl gencert -initca ca-csr.json | cfssljson -bare ca -

cat > blog.ctnrs.com-csr.json <<EOF
{
  "CN": "blog.ctnrs.com",
  "hosts": [],
  "key": {
    "algo": "rsa",
    "size": 2048
  },
  "names": [
    {
      "C": "CN",
      "L": "BeiJing",
      "ST": "BeiJing"
    }
  ]
}
EOF

# 生成证书
cfssl gencert -ca=ca.pem -ca-key=ca-key.pem -config=ca-config.json -profile=kubernetes bl
g.ctnrs.com-csr.json | cfssljson -bare blog.ctnrs.com
```

```
# 将证书保存到k8s secret中
kubectl create secret tls blog-ctnrs-com --cert=blog.ctnrs.com.pem --key=blog.ctnrs.com-key
pem
```

查看证书：

```
[root@k8s-master ingress]# kubectl get secret
NAME                  TYPE                                  DATA   AGE
blog-ctnrs-com        kubernetes.io/tls                     2      2m8s
```

yaml示例：

```
# https
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: tls-example-ingress
spec:
  tls:
  - hosts:
    - blog.ctnrs.com
    secretName: blog-ctnrs-com
  rules:
    - host: blog.ctnrs.com
      http:
        paths:
        - path: /
          backend:
            serviceName: web
            servicePort: 80
```

参数解释：

- tls：tls证书信息
- host(s)：域名，类似nginx中的server_name
- secretName:：保存在k8s中的证书名称
- rules：http配置，使用https配置也会默认创建一个http，访问http会跳转到https
- path：类似nginx中的location
- serviceName：deployment中对应的Service名称，暴露的项目名称
- servicePort：监听端口，类似nginx中的listen，Service的端口（内部端口）

```
server {
    listen       443 ssl;
    crt xxx;
    key xxx;
    server_name  blog.ctnrs.com;
    root         /usr/share/nginx/html;
    location / {
    }
}
```

五、解决Ingress高可用

NodePort暴露端口之后，每个节点都可以访问，唯独Ingress只能是controller所在的节点上可以访问

方案：

1、扩容Ingress-Controller Pod副本数

（1）提高并发能力

（2）尽量让多个节点提供服务

2、把控制器固定到几台节点

（1）daemonset（与nodeport一样）

（2）nodeselector+污点

污点，即使加污点容忍也不会完全分配到专门几个节点