【Kubernetes】Ingress Nginx安装使用

原创 dingpwen 于 2022-05-29 15:08:48 发布 698
分类专栏: 云原生 文章标签: kubernetes nginx 运维

> Ingress Nginx官方地址:
> https://kubernetes.github.io/ingress-nginx/deploy/
> https://kubernetes.github.io/ingress-nginx/deploy/baremetal/

在Kubenetes集群中，要使得我们的微服务可以被集群外的机器访问，就得为每个微服务暴漏一个端口，一方面这样不安全，另一方面端口数量有限，所以需要用到类似ingress nginx类型工具来进行内部dns管理。

ingress nginx安装 很简单，可以通过helm或者kubectl进行安装，对于国内来说，在没有vpn的情况下，我们没法下载google的东西，所以一般不用helm安装，或者用helm安装需要提前下载好相关文件。相反用kubectl安装相对简单。

对于我们自建集群来说，都可以说是 Bare metal clusters，所以我们使用的是:

> kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-nginx/controller-v1.2.0/deploy/static/provider/baremetal/deploy.yaml

在裸机集群中使用Ingress Nginx，有多种模式，这里根据个人实践，描叙一下三种模式：LoadBalancer、NodePort、HostNetwork。

1. LoadBalancer 模式
   这种模式，需要把上面的deploy.yaml中ingress-nginx-controller的Service中的type改为Balancer:

```
type: LoadBalancer
```

我们知道自建集群是不支持负载均衡的，所以需要安装metallb插件，安装方法也很简单:
首先修改kube-proxy的配置，添加strictARP: true

```
apiVersion: kubeproxy.config.k8s.io/v1alpha1
kind: KubeProxyConfiguration
mode: "ipvs"
ipvs:
  strictARP: true
```

然后运行:

```
kubectl apply -f https://raw.githubusercontent.com/metallb/metallb/v0.12.1/manifests/namespace.yaml
kubectl apply -f https://raw.githubusercontent.com/metallb/metallb/v0.12.1/manifests/metallb.yaml
```

如果无法下载，可以先去github复制相关文件保存到本地。

然后config.yaml:

```
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: metallb-system
  name: config
data:
  config: |
    address-pools:
    - name: default
      protocol: layer2
      addresses:
      - 192.168.106.120-192.168.106.128
```

运行 kubectl apply -f onfig.yaml 后，运行命令kubectl -n ingress-nginx get svc，就会看到

```
$ kubectl -n ingress-nginx get svc
NAME                   TYPE          CLUSTER-IP      EXTERNAL-IP   PORT(S)
ingress-nginx          LoadBalancer  192.168.106.120   203.0.113.10  80:30100/TCP,443:30101/TCP
```

然后我们就可以直接在浏览器中输入 http://192.168.106.120，可以发现可以访问我们的ingress-nginx服务。
注意这里给负载均衡用的地址范围不能是master或node的实际地址。

2. NodePort模式
   这里是指不安装metallb的情况下。这里我得到的实践结果与官方有一定差别。
   官方推荐设置如下:
   添加externalTrafficPolicy: Local
   添加externalIPs（可选）

```yaml
apiVersion: v1
kind: Service
metadata:
  labels:
    app.kubernetes.io/component: controller
    app.kubernetes.io/instance: ingress-nginx
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/part-of: ingress-nginx
    app.kubernetes.io/version: 1.2.0
  name: ingress-nginx-controller
  namespace: ingress-nginx
spec:
  externalTrafficPolicy: Local
  ipFamilies:
  - IPv4
  ipFamilyPolicy: SingleStack
  ports:
  - appProtocol: http
    name: http
    port: 80
    protocol: TCP
    targetPort: http
  - appProtocol: https
    name: https
    port: 443
    protocol: TCP
    targetPort: https
  selector:
    app.kubernetes.io/component: controller
    app.kubernetes.io/instance: ingress-nginx
    app.kubernetes.io/name: ingress-nginx
  type: NodePort
  externalIPs:
  - 193.168.108.129
```

实验结果是如果externalIPs使用的是VIP，而不是master或node的实际ip，只有在master机器上可以curl 193.168.108.129，即直接访问80端口，在其它机器上不行，其它机器只能使用ip:nodeport 来访问。externalIPs使用的是运行ingress机器的实际ip，或者不干脆不设置，只能通过nodeip:nodeport来访问。

3. Hostnetwork模式
   如下配置：

```
template:
  spec:
    hostNetwork: true
```

这种情况下可以将ingress nginx的service删除，可以直接通过nodeip(即80端口）访问相关服务。通过kubectl get ing 可以获得相关地址

```
$ kubectl get ingress -o wide
NAME           HOSTS              ADDRESS                   PORTS
test-ingress   myapp.example.com   203.0.113.2,203.0.113.3   80
```

如果想自定义Ingress上报时的IP也很容易，给nginx-ingress-controller加上–publish-status-address启动参数就行。

k8s loadbalancer与ingress实践

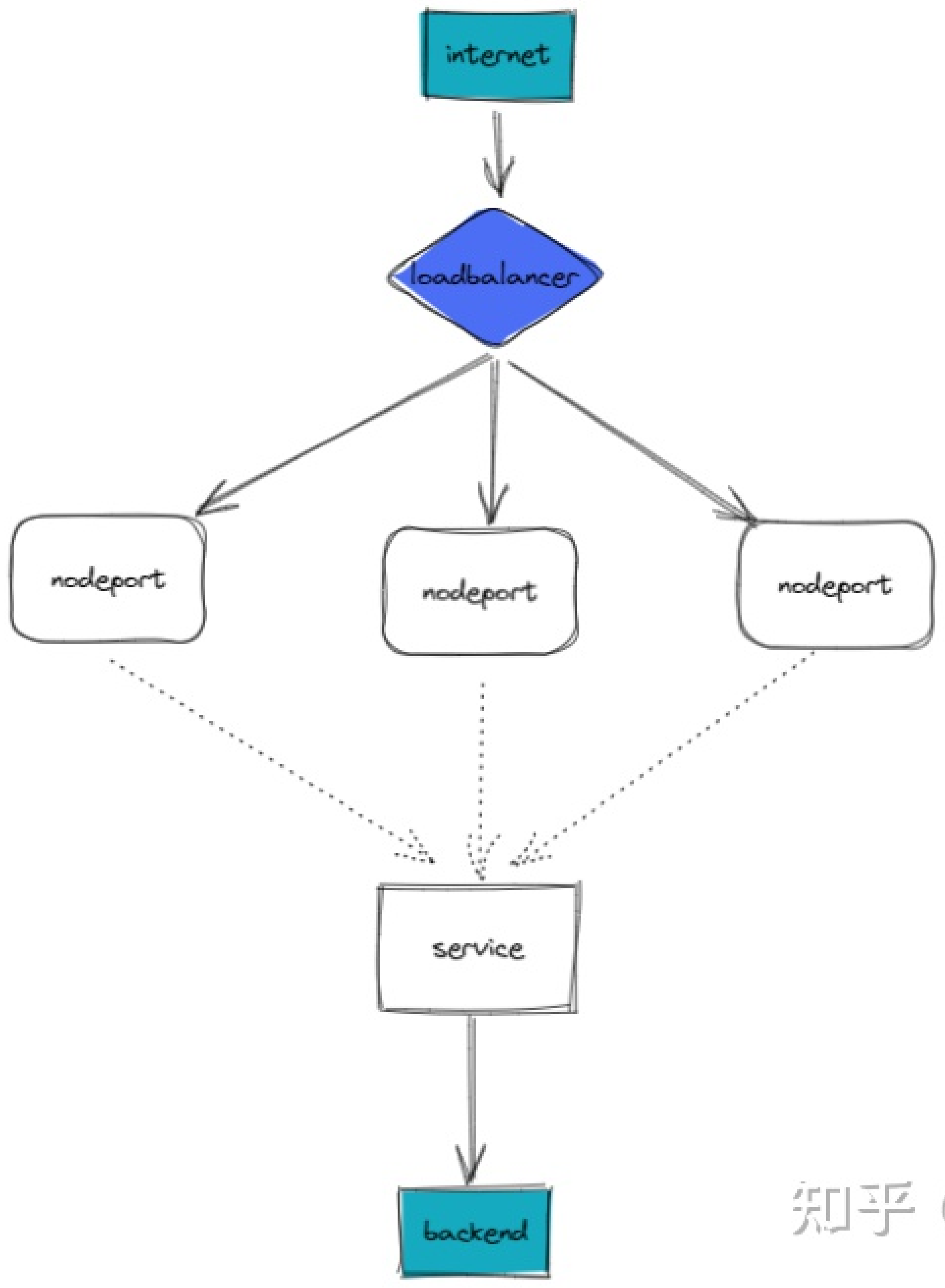
开放云
www.opencloud.world 让一部分人先看到世界

k8s可以通过三种方式将集群内服务暴露到外网，分别是NodePort、LoadBalancer、Ingress，其中NodePort作为基础通信形式我们在《k8s网络模型与集群通信》中进行了介绍，这里我们主要关注LoadBalancer和Ingress

LoadBalancer

loadbalancer是服务暴露到因特网的标准形式，和nodeport一样我们只需在创建service是指定type为loadbalancer即可，接着Service 的通过 `status.loadBalancer` 字段将需要创建的负载均衡器信息发布供负载均衡服务创建。不过loadbalancer是云服务商”专属“，像腾讯云CLB、阿里云SLB，这样在创建service时会自动帮我们创建一个负载均衡器。

大多数云上负载均衡也是基于nodeport，他们的结构如下：

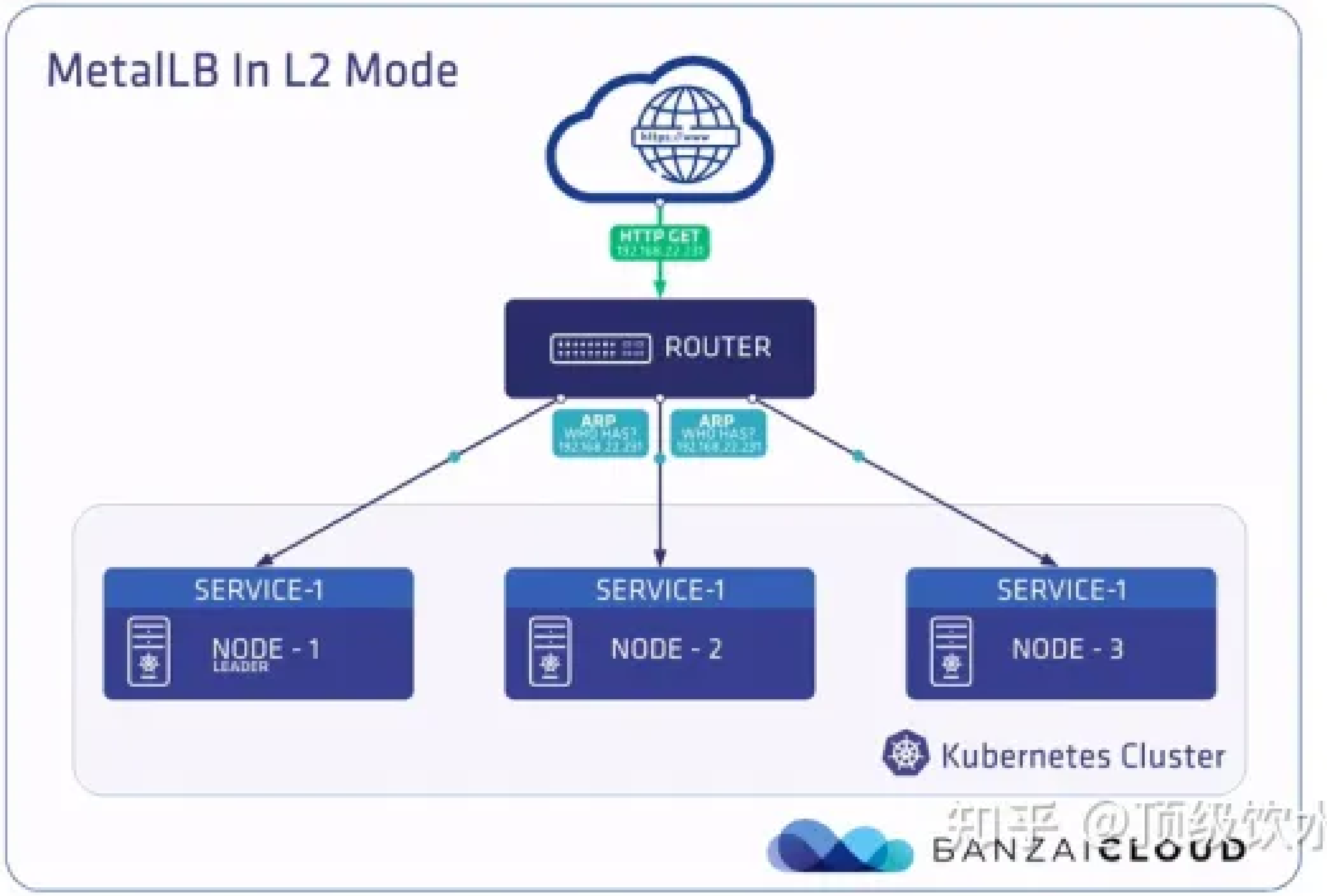

internet
loadbalancer
nodeport
nodeport
nodeport
service
backend

如果要在本地创建一个负载均衡器如何实现呢？

MetalLB，一个CNCF沙箱项目，使用标准路由协议(ARP/BGP)，实现裸机K8s集群的负载均衡器。

安装方式可参考官方文档：installation

L2（子网）模式的结构，图源



安装后我们获得如下两个组件：

- `metallb-system/controller` deployment。用于处理IP分配的控制器。
- `metallb-system/speaker` daemonset。集群中每个节点启动一个协议服务守护进程。

接着添加一个configmap配置metallb IP池。

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: metallb-system
  name: config
data:
  config: |
    address-pools:
    - name: default
      protocol: layer2
      addresses:
      - 192.168.1.240-192.168.1.250
```

这样当我们创建一个loadbalancer类型的service时,EXTERNAL-IP将会从地址池中获取一个用于外部访问的IP 192.168.1.243 当外部流量进入时，ARP将我们的请求地址广播获取所属的service，接着k8s内部 通过 `iptables` 规则和 `kube-proxy` ，将流量从服务端点引导到后端。

```yaml
#nginx_deployment_service.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
  namespace: metallb-system
spec:
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:latest
        ports:
        - name: http
          containerPort: 80

---
apiVersion: v1
kind: Service
metadata:
  namespace: metallb-system
  name: nginx
spec:
  ports:
  - name: http
    port: 80
    protocol: TCP
    targetPort: 80
  selector:
    app: nginx
  type: LoadBalancer
```

查看service `kubectl get svc`

```
NAME     TYPE           CLUSTER-IP     EXTERNAL-IP    PORT(S)        AGE
nginx    LoadBalancer   10.96.243.159  192.168.1.243  80:31052/TCP   40h
```

测试访问： `curl 192.168.1.243`

```
# curl 192.168.1.243
<!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>
```

负载均衡可以建立在 OSI 网络模型的不同级别上，主要是在 L4（传输层，例如 TCP/UDP）和 L7（应用层，例如 HTTP）上。在 Kubernetes 中， `Services` 是 L4 的抽象，LoadBalancer类型负载均衡依然有局限性，同时我们看到每创建一个service对应的负载均衡器都会消耗一个静态IP，这并不合理。当然k8s中的另一种资源对象ingress可工作在 L7 层实现应用程序协议（HTTP/HTTPS）的负载均衡。

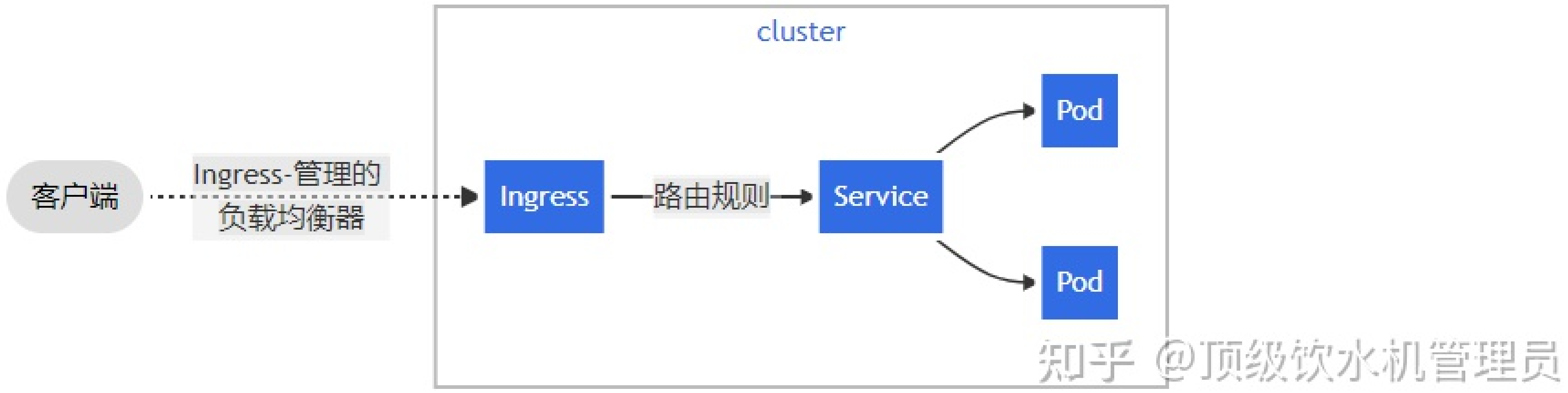
Ingress

Ingress 公开了从集群外部到集群内服务的 HTTP 和 HTTPS 路由。 流量路由由 Ingress 资源上定义的规则控制。我们可以将 Ingress 配置为服务提供外部可访问的 URL、负载均衡流量、终止 SSL/TLS，以及提供基于名称的虚拟主机等能力。

我们所说的Ingress包含两个部分：

- ingress k8s资源对象：流量路由规则的控制
- ingress-controller控制器：控制器的实现有非常多，可参考官方文档中列表Ingress 控制器，这里我们使用k8s官方维护的控制器NGINX Ingress Controller

外部流量进入集群时先经过ingress-controller，然后根据ingress配置的路由规则将请求转发到后端service。



ingress-controller

ingress-controller其实就是守护进程加一个反向代理的应用，守护进程不断监听集群中资源的变化，将ingress中的配置信息生成反向代理配置。在nginx-ingress controller中即生成 `nginx.conf` 的配置文件。

在本文中因为我们上面已经配置好了loadbalancer的服务，这样我们创建一个type为LoadBalancer的service关联这组pod，再把域名解析指向该地址，就实现了集群服务的对外暴露。当然你也可以使用 `NodePort` 、 `Hostnetwork` 的方式，感兴趣的小伙伴可以进行测试。

ingress-controller不是k8s内部组件，可以通过helm或资源清单方式安装,可查看ingress-nginx deploy

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-nginx/controller-v1.1.0/deploy/static/provider/cloud/deploy.y
```

然后我们编辑service

```
kubectl edit service/ingress-nginx-controller -n ingress-nginx
```

修改spec.type为LoadBalancer即可。

这样我们创建好了nginx-ingress controller，下一步就要配置ingress路由规则。

ingress规则

host：k8s.com

基于url的路由：

- /api/v1
- /api/v2

这两个url分别路由到不同的service中

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: test
  namespace: training
  annotations:
    ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
  - host: k8s.com
    http:
      paths:
      - path: /api/v1
        backend:
          serviceName: service-apiv1
          servicePort: 80
      - path: /api/v2
        backend:
          serviceName: service-apiv2
          servicePort: 80
```

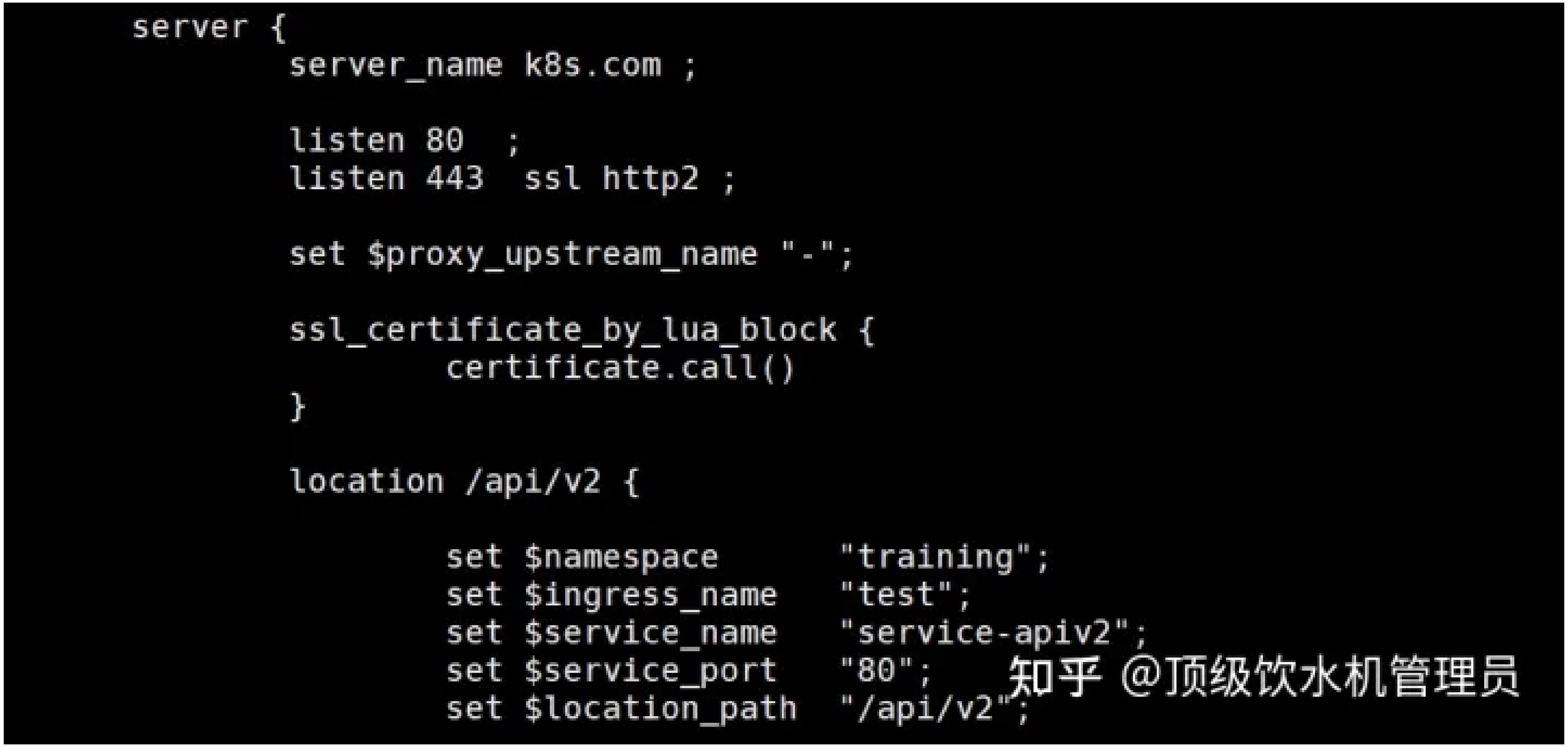
`ingress.kubernetes.io/rewrite-target` 是nginx-ingress controller的一个注解，当后端服务中暴露的 URL 与 Ingress 规则中指定的路径不同时可以通过此重定向。

查看svc可以看到此时控制器已经获得了一个EXTERNAL-IP

```
#kubectl get svc -n ingress-nginx
NAME                                 TYPE           CLUSTER-IP     EXTERNAL-IP     PORT(S)                      AGE
ingress-nginx-controller             LoadBalancer   10.96.87.23    192.168.1.245   80:32603/TCP,443:31906/TCP   621d
ingress-nginx-controller-admission   ClusterIP      10.96.109.70   <none>          443/TCP                      621d
```

现在nginx-ingress controller和ingress路由规则都有了。

我们可以进入到nginx-ingress controller pod中查看nginx.conf可以看到此时我们的ingress配置已经被生成为路由规则。



```
    server {
            server_name k8s.com ;

            listen 80  ;
            listen 443  ssl http2 ;

            set $proxy_upstream_name "-";

            ssl_certificate_by_lua_block {
                    certificate.call()
            }

            location /api/v2 {

                    set $namespace      "training";
                    set $ingress_name   "test";
                    set $service_name   "service-apiv2";
                    set $service_port   "80";
                    set $location_path  "/api/v2";
```

接下来就是指定我们的backend，即上面的server-apiv1/2

我们添加两个用于暴露的service和deployment，和loadbalancer中测试清单一样，我们稍稍修改一下名称即可。

```
apiVersion: apps/v1
kind: Deployment
```

```yaml
metadata:
  name: nginx-apiv1
  namespace: training
spec:
  selector:
    matchLabels:
      app: nginx-apiv1
  template:
    metadata:
      labels:
        app: nginx-apiv1
    spec:
      containers:
      - name: nginx-apiv1
        image: nginx:latest
        ports:
        - name: http
          containerPort: 80

---
apiVersion: v1
kind: Service
metadata:
  namespace: training
  name: service-apiv1
spec:
  ports:
  - name: http
    port: 80
    protocol: TCP
    targetPort: 80
  selector:
    app: nginx-apiv1
  type: NodePort
```

将nginx-apiv1换成nginx-apiv2创建出另一个service和deployment。

最后修改hosts解析k8s.com

```
192.168.1.245 k8s.com
```

使用curl命令测试url路由（记得在pod中添加测试文件，否则虽然url进行了路由但会出现404）。

```
# curl k8s.com/api/v1/index.html
api v1
# curl k8s.com/api/v2/index.html
api v2
```

这样我们对ingress有了初步了解，ingress的路由规则可自定项较多也比较繁杂，可通过官方文档进一步学习。

发布于 2022-01-04 22:02

Kubernetes　Ingress（kubernetes）　容器（虚拟化）

Installing NGINX Ingress Controller

This tutorial covers the installation of NGINX Ingress controller, which is an open source project made by the Kubernetes community. k0s doesn't come with an in-built Ingress controller, but it's easy to deploy NGINX Ingress as shown in this document. Other Ingress solutions can be used as well (see the links at the end of the page).

NodePort vs LoadBalancer vs Ingress controller

Kubernetes offers multiple options for exposing services to external networks. The main options are NodePort, LoadBalancer and Ingress controller.

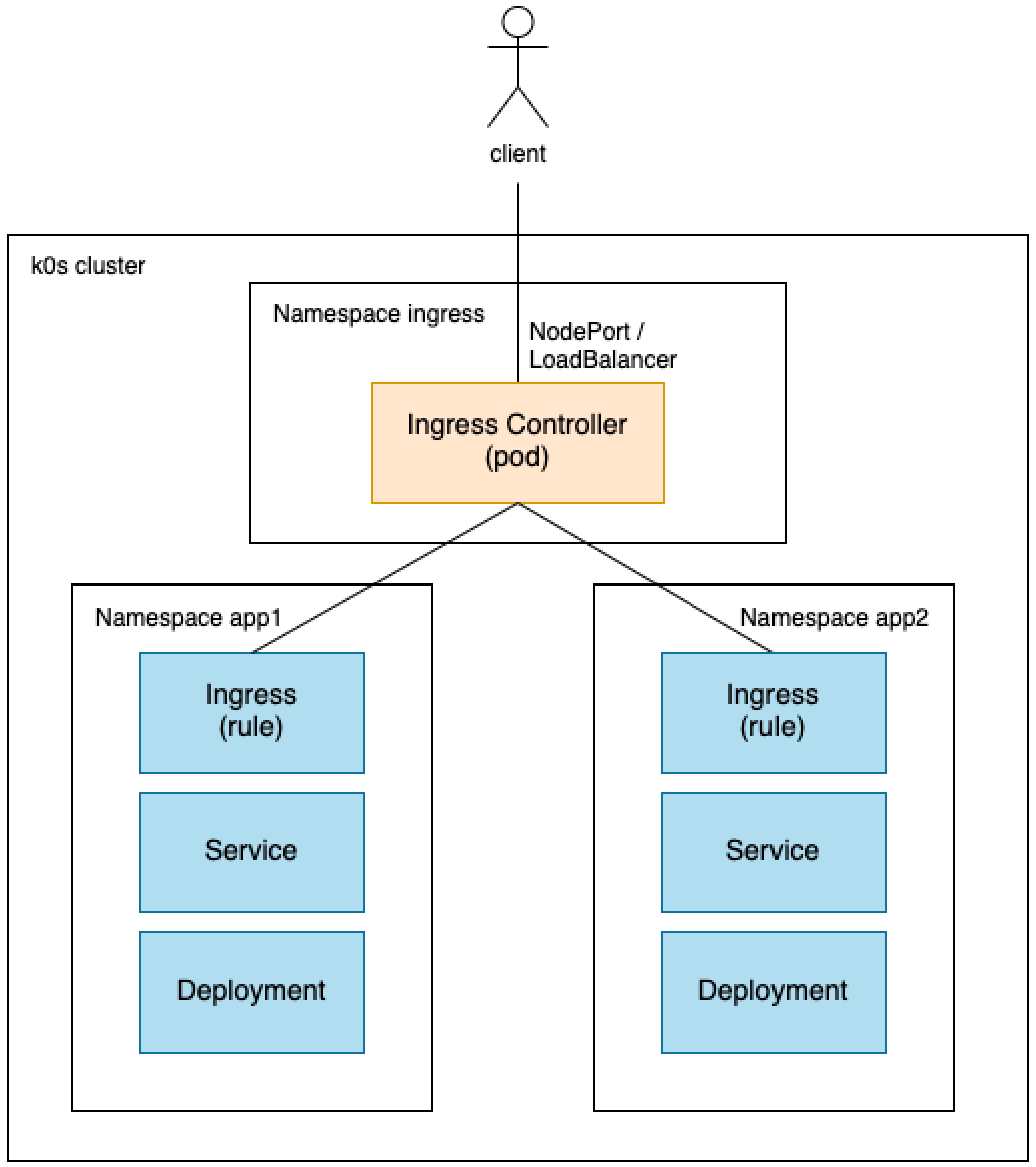
NodePort, as the name says, means that a port on a node is configured to route incoming requests to a certain service. The port range is limited to 30000-32767, so you cannot expose commonly used ports like 80 or 443 with NodePort.

LoadBalancer is a service, which is typically implemented by the cloud provider as an external service (with additional cost). Load balancers can also be installed internally in the Kubernetes cluster with MetalLB, which is typically used for bare-metal deployments. Load balancer provides a single IP address to access your services, which can run on multiple nodes.

Ingress controller helps to consolidate routing rules of multiple applications into one entity. Ingress controller is exposed to an external network with the help of NodePort or LoadBalancer. You can also use Ingress controller to terminate TLS for your domain in one place, instead of terminating TLS for each application separately.

NGINX Ingress Controller

NGINX Ingress Controller is a very popular Ingress for Kubernetes. In many cloud environments, it can be exposed to an external network by using the load balancer offered by the cloud provider. However, cloud load balancers are not necessary. Load balancer can also be implemented with MetalLB, which can be deployed in the same Kubernetes cluster. Another option to expose the Ingress controller to an external network is to use NodePort. Both of these alternatives are described in more detail on below, with separate examples.



Install NGINX using NodePort

Installing NGINX using NodePort is the most simple example for Ingress Controller as we can avoid the load balancer dependency. NodePort is used for exposing the NGINX Ingress to the external network.

1. Install NGINX Ingress Controller (using the official manifests of version `v1.0.0` by the ingress-nginx project, supports Kubernetes versions >= `v1.19`)

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-nginx/controller-v1.0.0/deploy/static/provider/baremetal/deploy.yaml
```

2. Check that the Ingress controller pods have started

   ```
   kubectl get pods -n ingress-nginx
   ```

3. Check that you can see the NodePort service

   ```
   kubectl get services -n ingress-nginx
   ```

4. From version `v1.0.0` of the Ingress-NGINX Controller, a ingressclass object is required.

   In the default installation, an ingressclass object named `nginx` has already been created.

   ```
   $ kubectl -n ingress-nginx get ingressclasses
   NAME    CONTROLLER             PARAMETERS   AGE
   nginx   k8s.io/ingress-nginx   <none>       162m
   ```

   If this is only instance of the Ingresss-NGINX controller, you should add the annotation `ingressclass.kubernetes.io/is-default-class` in your ingress class:

   ```
   kubectl -n ingress-nginx annotate ingressclasses nginx ingressclass.kubernetes.io/is-default-class="true"
   ```

5. Try connecting the Ingress controller using the NodePort from the previous step (in the range of 30000-32767)

   ```
   curl <worker-external-ip>:<node-port>
   ```

   If you don't yet have any backend service configured, you should see "404 Not Found" from nginx. This is ok for now. If you see a response from nginx, the Ingress Controller is running and you can reach it.

6. Deploy a small test application (httpd web server) to verify your Ingress controller.

   Create the following YAML file and name it "simple-web-server-with-ingress.yaml":

   ```
   apiVersion: v1
   kind: Namespace
   metadata:
     name: web
   ---
   apiVersion: apps/v1
   ```

```yaml
kind: Deployment
metadata:
  name: web-server
  namespace: web
spec:
  selector:
    matchLabels:
      app: web
  template:
    metadata:
      labels:
        app: web
    spec:
      containers:
      - name: httpd
        image: httpd:2.4.48-alpine3.14
        ports:
        - containerPort: 80
---
apiVersion: v1
kind: Service
metadata:
  name: web-server-service
  namespace: web
spec:
  selector:
    app: web
  ports:
    - protocol: TCP
      port: 5000
      targetPort: 80
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: web-server-ingress
  namespace: web
spec:
  ingressClassName: nginx
  rules:
  - host: web.example.com
    http:
      paths:
      - path: /
```

```
        pathType: Prefix
        backend:
          service:
            name: web-server-service
            port:
              number: 5000
```

Deploy the app:

```
kubectl apply -f simple-web-server-with-ingress.yaml
```

7. Verify that you can access your application using the NodePort from step 3.

```
curl <worker-external-ip>:<node-port> -H 'Host: web.example.com'
```

If you are successful, you should see `<html><body><h1>It works!</h1></body></html>`.

Install NGINX using LoadBalancer

In this example you'll install NGINX Ingress controller using LoadBalancer on k0s.

1. Install LoadBalancer

   There are two alternatives to install LoadBalancer on k0s. Follow the links in order to install LoadBalancer.

   - MetalLB as a pure SW solution running internally in the k0s cluster
   - Cloud provider's load balancer running outside of the k0s cluster

2. Verify LoadBalancer

   In order to proceed you need to have a load balancer available for the Kubernetes cluster. To verify that it's available, deploy a simple load balancer service.

```
apiVersion: v1
kind: Service
metadata:
  name: example-load-balancer
spec:
  selector:
    app: web
  ports:
    - protocol: TCP
```

```
      port: 80
      targetPort: 80
  type: LoadBalancer
```

```
kubectl apply -f example-load-balancer.yaml
```

Then run the following command to see your LoadBalancer with an external IP address.

```
kubectl get service example-load-balancer
```

If the LoadBalancer is not available, you won't get an IP address for EXTERNAL-IP. Instead, it's `<pending>`. In this case you should go back to the previous step and check your load balancer availability.

If you are successful, you'll see a real IP address and you can proceed further.

You can delete the example-load-balancer:

```
kubectl delete -f example-load-balancer.yaml
```

3. Install NGINX Ingress Controller by following the steps in the previous chapter (step 1 to step 4).
4. Edit the NGINX Ingress Controller to use LoadBalancer instead of NodePort

   ```
   kubectl edit service ingress-nginx-controller -n ingress-nginx
   ```

   Find the spec.type field and change it from "NodePort" to "LoadBalancer".
5. Check that you can see the ingress-nginx-controller with type LoadBalancer.

   ```
   kubectl get services -n ingress-nginx
   ```

6. Try connecting to the Ingress controller

   If you used private IP addresses for MetalLB in step 2, you should run the following command from the local network. Use the IP address from the previous step, column EXTERNAL-IP.

   ```
   curl <EXTERNAL-IP>
   ```

If you don't yet have any backend service configured, you should see "404 Not Found" from nginx. This is ok for now. If you see a response from nginx, the Ingress Controller is running and you can reach it using LoadBalancer.

7. Deploy a small test application (httpd web server) to verify your Ingress.

   Create the YAML file "simple-web-server-with-ingress.yaml" as described in the previous chapter (step 6) and deploy it.

   ```
   kubectl apply -f simple-web-server-with-ingress.yaml
   ```

8. Verify that you can access your application through the LoadBalancer and Ingress controller.

   ```
   curl <worker-external-ip> -H 'Host: web.example.com'
   ```

   If you are successful, you should see `<html><body><h1>It works!</h1></body></html>`.

Additional information

For more information about NGINX Ingress Controller installation, take a look at the official ingress-nginx documentation.

Alternative examples for Ingress Controllers on k0s

Traefik Ingress

Failed to install ingress controller

aracloud May '21

Hi

I'm new to k8s at all.

I added a new cluster to rancher server with 3 modes (one master, two worker). I disabled the default nginx-ingress-controller because I want to install my own one.

So, via another chart (**https://charts.bitnami.com/bitnami**) I tried to install nginx-ingress-ctrl but the log states:

```
helm install --namespace=ara-system --timeout=10m0s --values=/home/shell/helm/values-nginx-ingress-controller-7.6.6.yaml --
version=7.6.6 --wait=true nginx-ing-ctrl /home/shell/helm/nginx-ingress-controller-7.6.6.tgz
creating 10 resource(s)
beginning wait for 10 resources with timeout of 10m0s
Service does not have load balancer ingress IP address: ara-system/nginx-ing-ctrl-nginx-ingress-controller
Service does not have load balancer ingress IP address: ara-system/nginx-ing-ctrl-nginx-ingress-controller
Service does not have load balancer ingress IP address: ara-system/nginx-ing-ctrl-nginx-ingress-controlle
```

What does that mean "Service does not have load balancer ingress IP address".
What is required here to get it up and running?

Thx

vincent Retired May '21

Skip to main content

There is no generic implementation of LoadBalancer services, so they do nothing by default unless you have a Cloud Provider or another controller like MetakLB or kube-vip installed to provide them.

That is apparently not the case in your cluster, so it's waiting for a balancer to be provisioned that isn't coming.

This is also why the default ingress setup uses a HostPort instead of a LoadBalancer.

---

aracloud May '21

@vincent
Hi Vicent

How do I install an additional ingress controller or how do I define a load balancer in order to install additional ingress controller.

Thx

---

vincent Retired May '21

Depending on what and where you're creating, most kinds of clusters already come with either traefik or nginx configured as an ingress. To install another you follow their docs because it is no longer part of Rancher.

Again depending, most kinds of clusters *do not* come with an implementation of LoadBalancers. Some of the hosted kubernetes-as-a-service providers (e.g. GKE) do. Some infrastructure-as-a-service providers (e.g. EC2) have a Cloud Provider you can configure when creating the cluster to enable them. Our k3s distro comes with a very simple implementation so that they do something out of the box. Otherwise you deploy your own (e.g. kube-vip and MetalLB) after the cluster is setup by following their instructions.