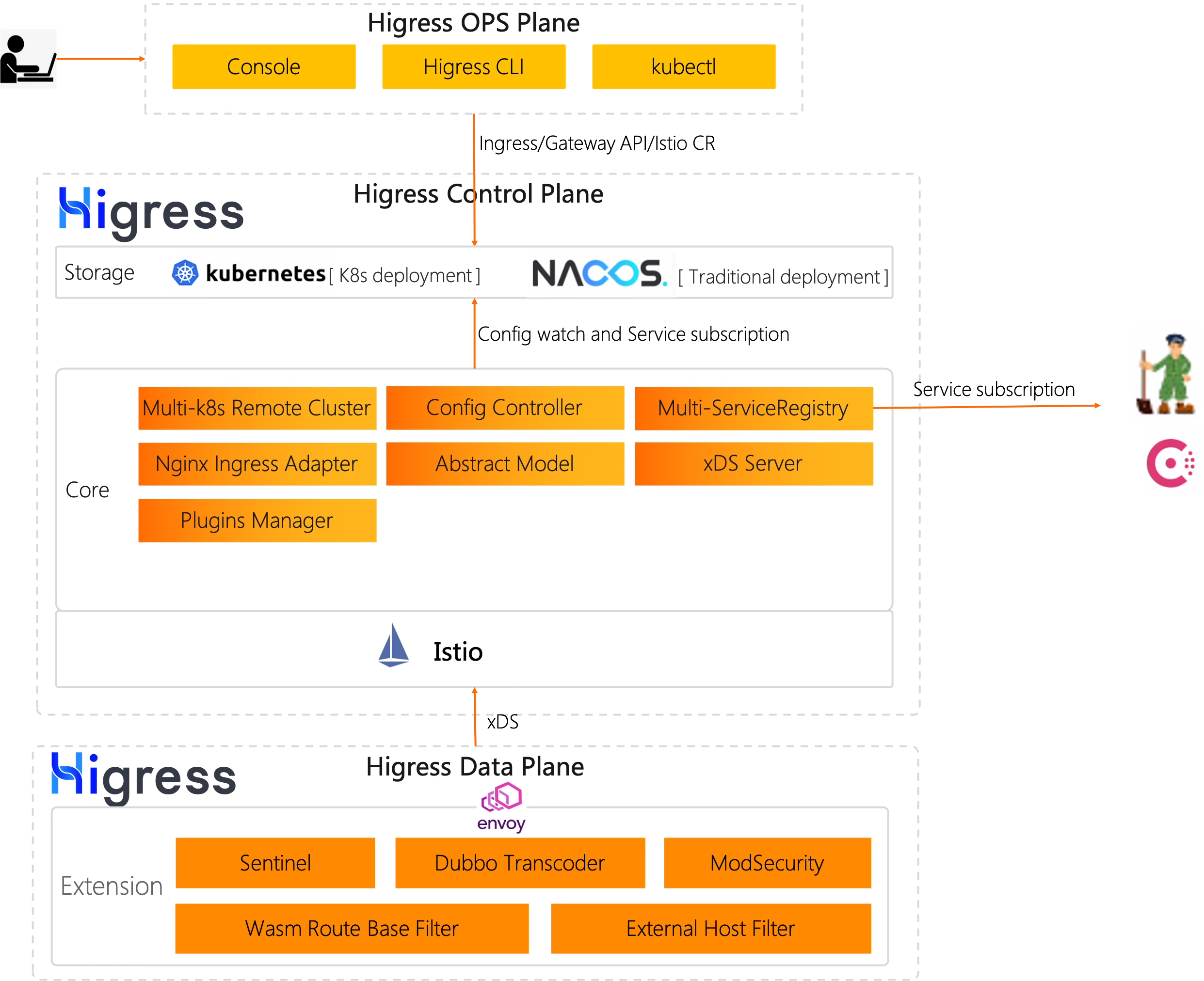
有道云笔记地址：<https://note.youdao.com/s/cAE1VMEN>

**1. 云原生网关Higress实战**

**1.1 Higress是什么**

[Higress](https://higress.io/zh-cn/docs/overview/what-is-higress)是基于阿里内部的Envoy Gateway实践沉淀、以开源Istio + Envoy为核心构建的下一代云原生网关，实现了流量网关 + 微服务网关 + 安全网关三合一的高集成能力，深度集成Dubbo、Nacos、Sentinel等微服务技术栈，能够帮助用户极大的降低网关的部署及运维成本且能力不打折；在标准上全面支持Ingress与Gateway API，积极拥抱云原生下的标准API规范；同时，Higress Controller也支持Nginx Ingress平滑迁移，帮助用户零成本快速迁移到Higress。



**1.2 Higress快速开始**

[基于docker compose安装Higress](https://higress.io/zh-cn/docs/user/quickstart/#%E7%8E%AF%E5%A2%83%E4%BA%8C%E8%84%B1%E7%A6%BB-k8s-%E5%9C%A8-docker-compose-%E4%B8%AD%E4%BD%BF%E7%94%A8)

**使用独立部署的 Nacos**

curl -fsSL https://higress.io/standalone/get-higress.sh | bash -s -- -c nacos://192.168.65.174:8848 -p admin

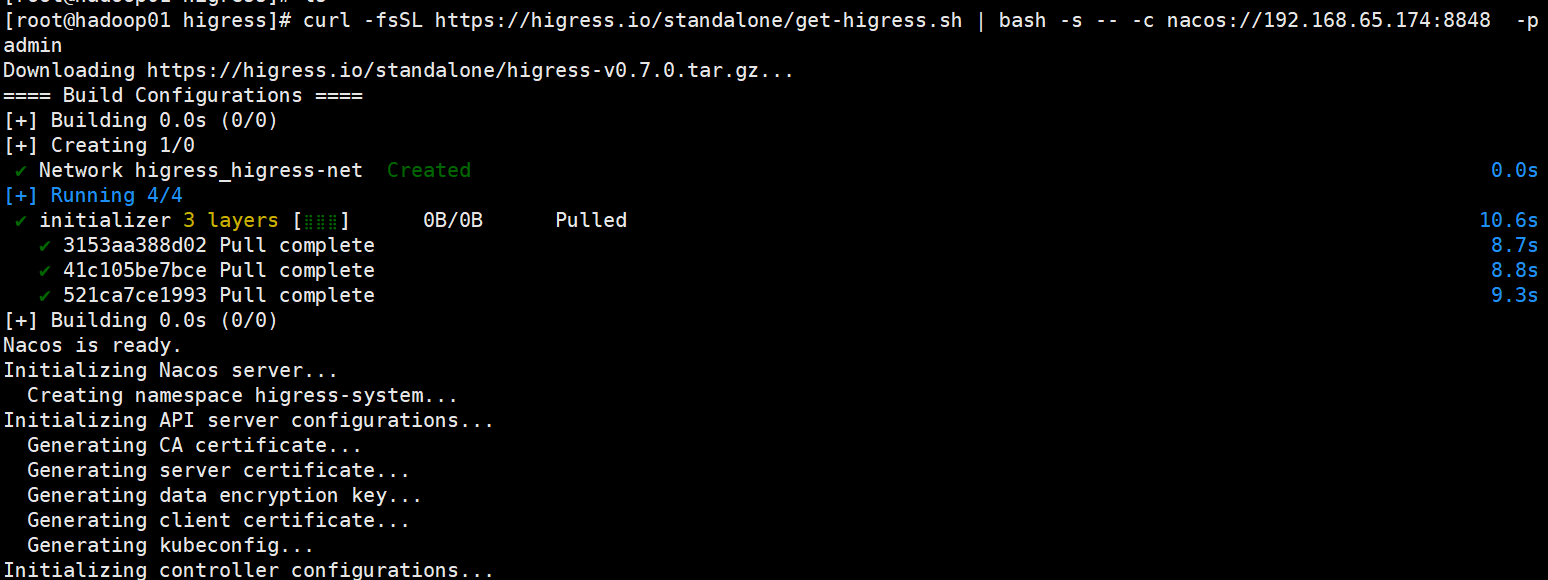
启动成功后，本机端口占用情况如下：

80端口：Higress 暴露，用于 HTTP 协议代理

443端口：Higress 暴露，用于 HTTPS 协议代理

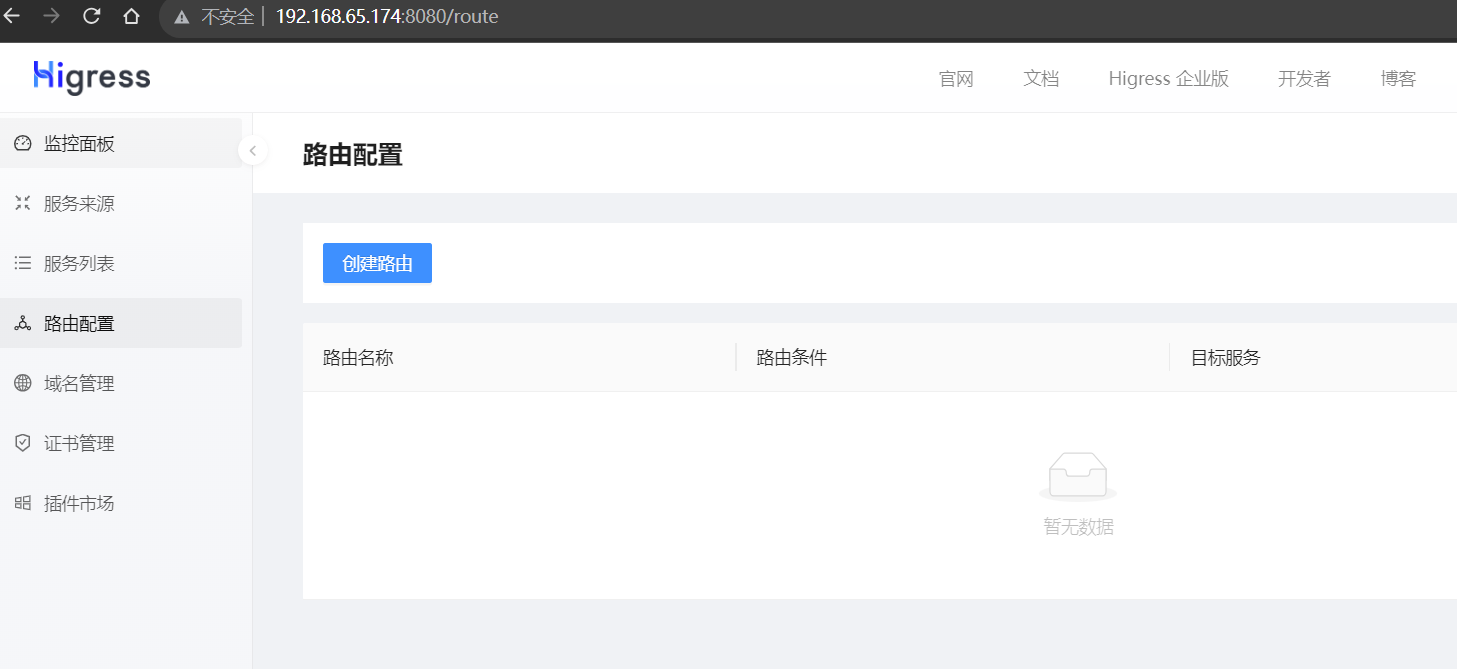
15020端口：Higress 暴露，用于暴露 Prometheus 指标

8080端口：Higress 控制台 暴露



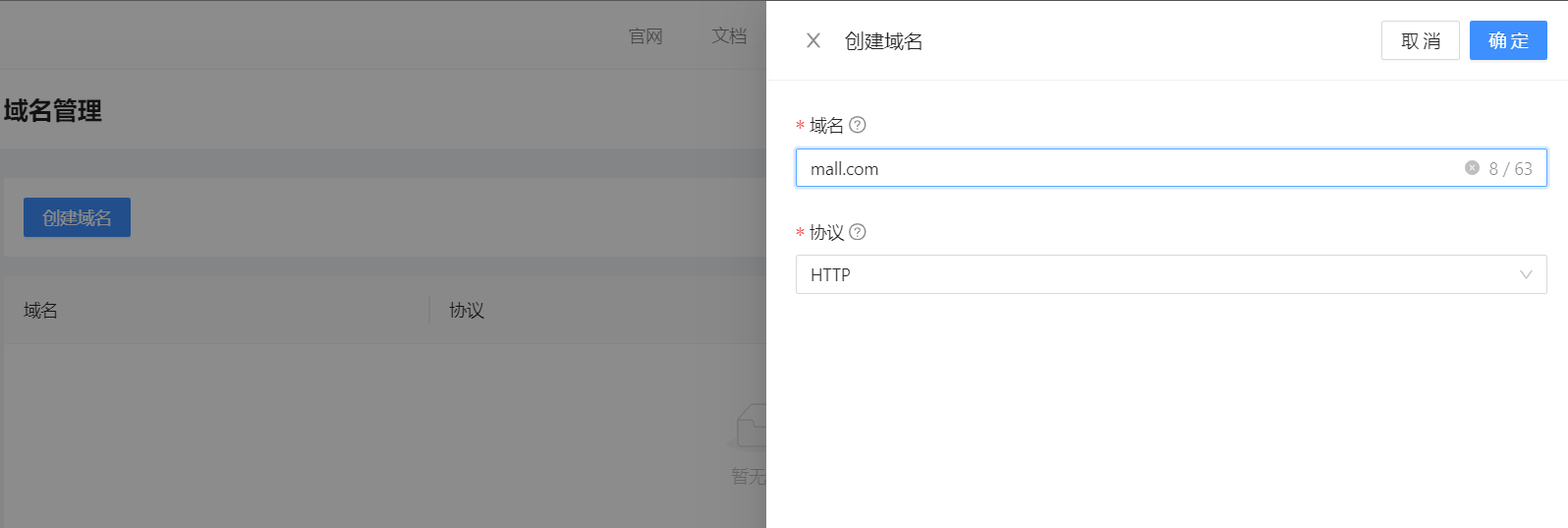
在浏览器中输入<http://192.168.65.174:8080/>，使用用户名 admin 和安装时设置的密码登录 Higress 控制台





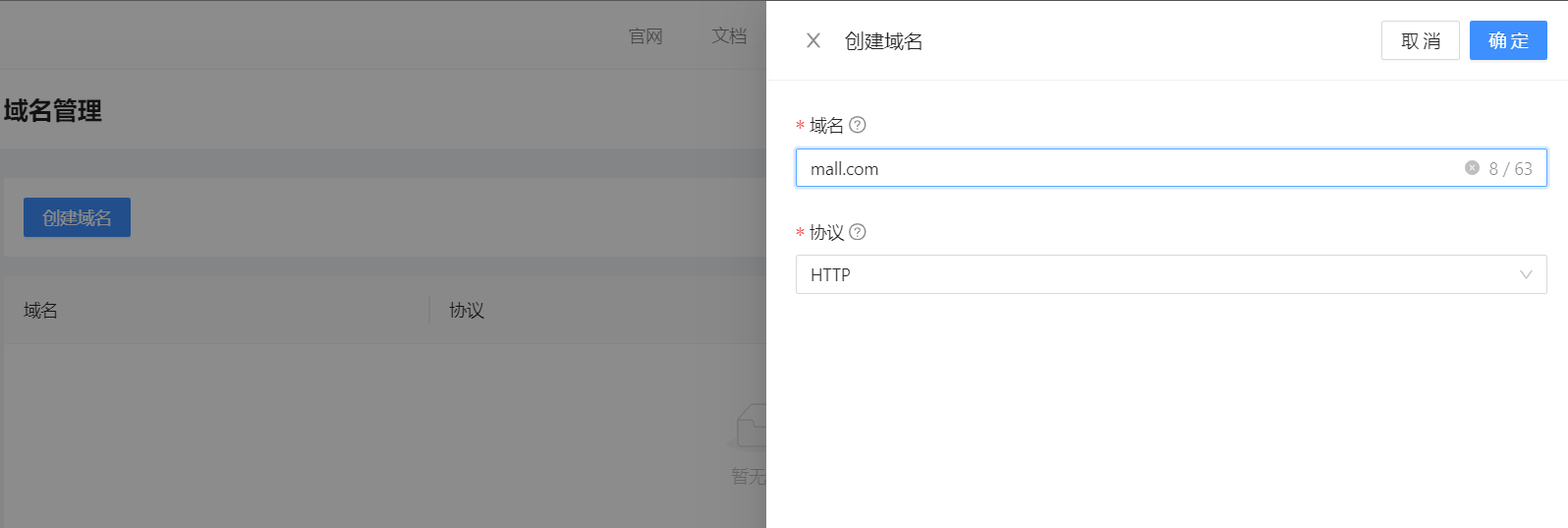
**配置 higress 域名**

配置域名解析至 higress 所在机器的 80/443 端口



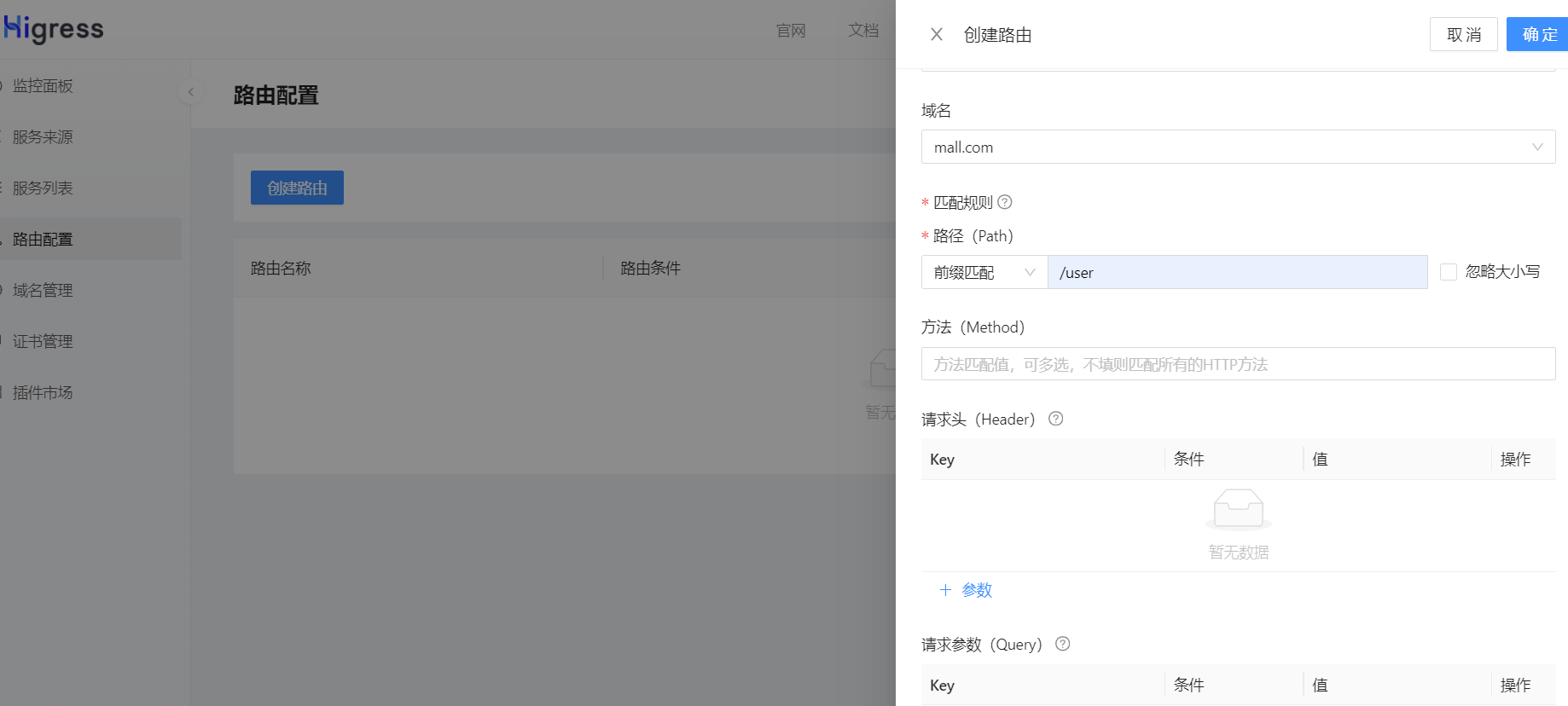
**服务列表**

服务列表自动同步指向的 nacos 中已注册服务列表



**配置路由转发**

在路由管理中创建两条新的路由，根据路由前缀将其转发至相应的微服务





**测试**

访问，验证测试路由可以正常工作

curl localhost/user/findOrderByUserId/1 -H 'host: mall.com'

[基于 K8s 集群安装Higress](https://higress.io/zh-cn/docs/user/quickstart/#%E5%9C%BA%E6%99%AF%E4%B8%80%E5%9C%A8%E6%A0%87%E5%87%86-k8s-%E9%9B%86%E7%BE%A4%E4%B8%AD%E4%BD%BF%E7%94%A8)

利用sealos快速安装kubernetes集群：<https://note.youdao.com/s/M2z4OzsL>

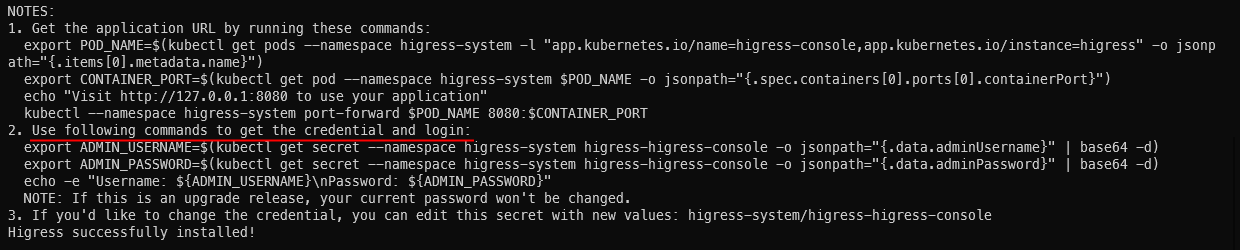
**安装Higress**

# 使用Helm 安装Higress

helm repo add higress.io https://higress.io/helm-charts

helm install higress -n higress-system higress.io/higress --create-namespace --render-subchart-notes --set higress-console.domain=console.higress.io

注意：安装完成后会输出一段文本，其中包含获取控制台登录信息的命令。请执行该命令并记录用户名和密码。

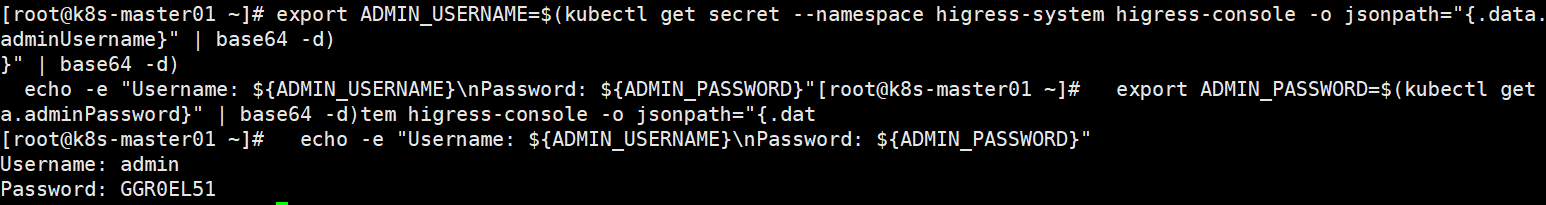


例如安装在 higress-system 命名空间下时，执行下面命令获取用户名密码：

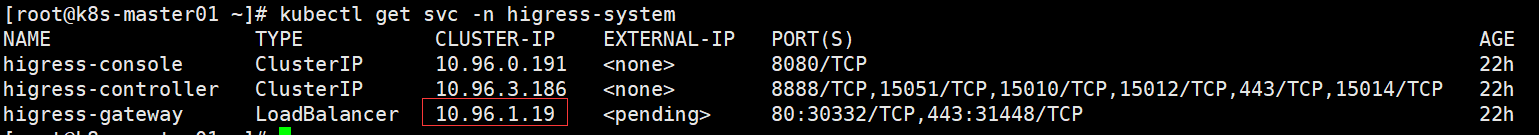
export ADMIN\_USERNAME=$(kubectl get secret --namespace higress-system higress-console -o jsonpath="{.data.adminUsername}" | base64 -d)

export ADMIN\_PASSWORD=$(kubectl get secret --namespace higress-system higress-console -o jsonpath="{.data.adminPassword}" | base64 -d)

echo -e "Username: ${ADMIN\_USERNAME}\nPassword: ${ADMIN\_PASSWORD}"



获取 Higress Gateway 的 LoadBalancer IP，并记录下来。后续可以通过该 IP 的 80 和 443 端口访问 Higress Gateway



**配置Higress**

假设在 default 命名空间下已经部署了一个名为 foo 的服务，而我们希望创建一个对应 http://foo.bar.com/foo 的路由指向该服务。

如果需要的话，各位可以使用下方 YAML 来创建对应的测试服务。

kind: Pod

apiVersion: v1

metadata:

name: foo-app

labels:

app: foo

spec:

containers:

- name: foo-app

image: higress-registry.cn-hangzhou.cr.aliyuncs.com/higress/http-echo:0.2.4-alpine

args:

- "-text=foo"

---

kind: Service

apiVersion: v1

metadata:

name: foo-service

spec:

selector:

app: foo

ports:

# Default port used by the image

- port: 5678

使用 Ingress CRD 进行路由配置，编写foo-ingress.yaml

apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

name: foo

spec:

ingressClassName: higress

rules:

- host: foo.bar.com

http:

paths:

- pathType: Prefix

path: "/foo"

backend:

service:

name: foo-service

port:

number: 5678

执行如下命令生效规则：

kubectl apply -f foo-ingress.yaml

**测试**

执行以下命令，验证测试路由可以正常工作：

curl http://10.96.1.19/foo -H 'host: foo.bar.com'

截图.png

**2. Higress流量治理实战**

在Higress上可以使用Ingress并借助Annotation实现高阶流量治理

**2.1 灰度发布**[​](https://higress.io/zh-cn/docs/user/annotation-use-case#%E7%81%B0%E5%BA%A6%E5%8F%91%E5%B8%83)

Higress提供复杂的路由处理能力，支持基于Header、Cookie以及权重的灰度发布功能。灰度发布功能可以通过设置注解来实现，为了启用灰度发布功能，需要设置注解higress.io/canary: "true"。通过不同注解可以实现不同的灰度发布功能。

说明：当多种方式同时配置时，灰度方式选择优先级为：基于Header > 基于Cookie > 基于权重（从高到低）。

**部署两个版本的服务**

1）在集群中部署第一个版本的 Deployment，本文以 nginx-v1 为例。YAML 示例如下：

apiVersion: apps/v1

kind: Deployment

metadata:

name: nginx-v1

spec:

replicas: 1

selector:

matchLabels:

app: nginx

version: v1

template:

metadata:

labels:

app: nginx

version: v1

spec:

containers:

- name: nginx

image: "openresty/openresty:centos"

ports:

- name: http

protocol: TCP

containerPort: 80

volumeMounts:

- mountPath: /usr/local/openresty/nginx/conf/nginx.conf

name: config

subPath: nginx.conf

volumes:

- name: config

configMap:

name: nginx-v1

---

apiVersion: v1

kind: ConfigMap

metadata:

labels:

app: nginx

version: v1

name: nginx-v1

data:

nginx.conf: |-

worker\_processes 1;

events {

accept\_mutex on;

multi\_accept on;

use epoll;

worker\_connections 1024;

}

http {

ignore\_invalid\_headers off;

server {

listen 80;

location / {

access\_by\_lua '

local header\_str = ngx.say("nginx-v1")

';

}

location /hello {

access\_by\_lua '

local header\_str = ngx.say("hello nginx-v1")

';

}

}

}

---

apiVersion: v1

kind: Service

metadata:

name: nginx-v1

spec:

type: ClusterIP

ports:

- port: 80

protocol: TCP

name: http

selector:

app: nginx

version: v1

2）再部署第二个版本的 Deployment，本文以 nginx-v2 为例。YAML 示例如下：

apiVersion: apps/v1

kind: Deployment

metadata:

name: nginx-v2

spec:

replicas: 1

selector:

matchLabels:

app: nginx

version: v2

template:

metadata:

labels:

app: nginx

version: v2

spec:

containers:

- name: nginx

image: "openresty/openresty:centos"

ports:

- name: http

protocol: TCP

containerPort: 80

volumeMounts:

- mountPath: /usr/local/openresty/nginx/conf/nginx.conf

name: config

subPath: nginx.conf

volumes:

- name: config

configMap:

name: nginx-v2

---

apiVersion: v1

kind: ConfigMap

metadata:

labels:

app: nginx

version: v2

name: nginx-v2

data:

nginx.conf: |-

worker\_processes 1;

events {

accept\_mutex on;

multi\_accept on;

use epoll;

worker\_connections 1024;

}

http {

ignore\_invalid\_headers off;

server {

listen 80;

location / {

access\_by\_lua '

local header\_str = ngx.say("nginx-v2")

';

}

location /hello {

access\_by\_lua '

local header\_str = ngx.say("hello nginx-v2")

';

}

}

}

---

apiVersion: v1

kind: Service

metadata:

name: nginx-v2

spec:

type: ClusterIP

ports:

- port: 80

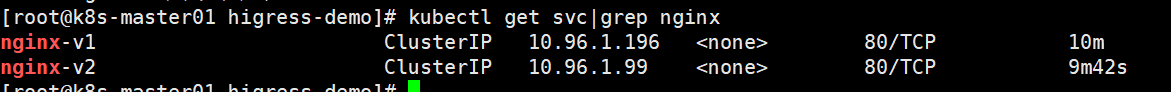
protocol: TCP

name: http

selector:

app: nginx

version: v2



**基于Header灰度发布**[​](https://higress.io/zh-cn/docs/user/annotation-use-case#%E5%9F%BA%E4%BA%8Eheader%E7%81%B0%E5%BA%A6%E5%8F%91%E5%B8%83)

只配置higress.io/canary-by-header：基于Request Header的名称进行流量切分。当请求包含该Header并其值为always时，请求流量会被分配到灰度服务入口；其他情况时，请求流量不会分配到灰度服务。

同时配置higress.io/canary-by-header和higress.io/canary-by-header-value：基于Request Header的名称和值进行流量切分。当请求中的header的名称和header的值与该配置匹配时，请求流量会被分配到灰度服务；其他情况时，请求流量不会分配到灰度服务。

请求Header为higress: always时将访问灰度服务nginx-v2；其他情况将访问正式服务nginx-v1，配置如下：

apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

annotations:

higress.io/canary: "true"

higress.io/canary-by-header: "higress"

name: higress-demo-canary

spec:

ingressClassName: higress

rules:

- http:

paths:

- backend:

service:

name: nginx-v2

port:

number: 80

path: /hello

pathType: Exact

---

apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

name: higress-demo

spec:

ingressClassName: higress

rules:

- http:

paths:

- backend:

service:

name: nginx-v1

port:

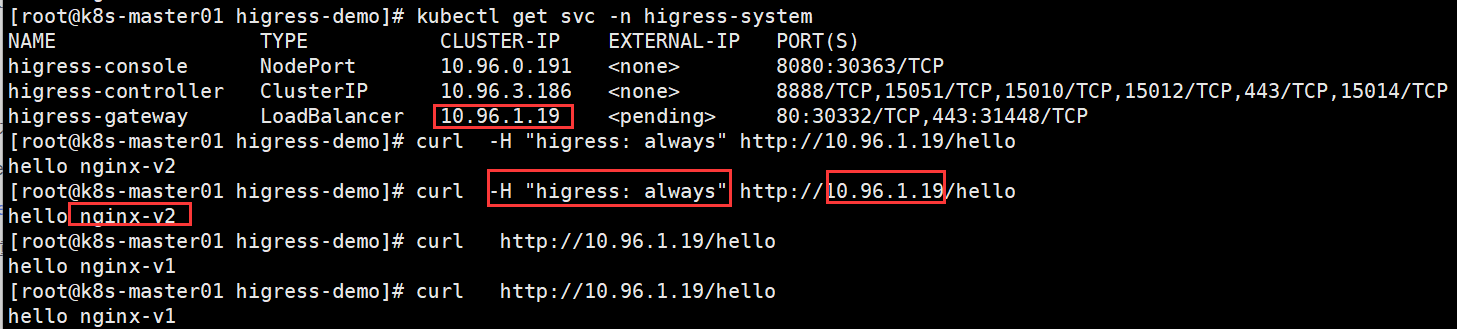
number: 80

path: /hello

pathType: Exact

部署后执行以下命令进行测试

curl -H "higress: always" http://10.96.1.19/hello



请求Header为higress: v2时将访问灰度服务nginx-v2；其他情况将访问正式服务nginx-v1，配置如下：

apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

annotations:

higress.io/canary: "true"

higress.io/canary-by-header: "higress"

higress.io/canary-by-header-value: "v1"

name: higress-demo-canary

spec:

ingressClassName: higress

rules:

- http:

paths:

- backend:

service:

name: nginx-v2

port:

number: 80

path: /hello

pathType: Exact

---

apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

name: higress-demo

spec:

ingressClassName: higress

rules:

- http:

paths:

- backend:

service:

name: nginx-v1

port:

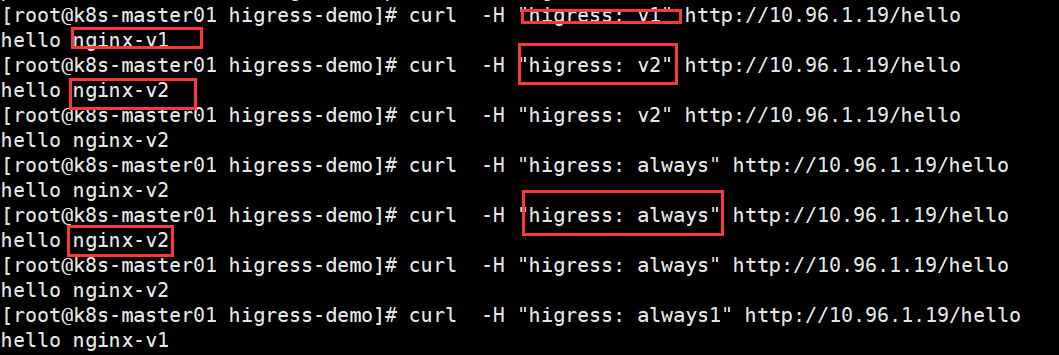
number: 80

path: /hello

pathType: Exact

部署后执行以下命令进行测试

curl -H "higress: v2" http://10.96.1.19/hello



**基于Cookie灰度发布**

higress.io/canary-by-cookie：基于Cookie的Key进行流量切分。当请求的Cookie中含有该Key且其值为always时，请求流量将被分配到灰度服务；其他情况时，请求流量将不会分配到灰度服务。

说明：基于Cookie的灰度发布不支持自定义设置Key对应的值，只能是always。

请求的Cookie为demo=always时将访问灰度服务nginx-v2；其他情况将访问正式服务nginx-v1。配置如下：

apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

annotations:

higress.io/canary: "true"

higress.io/canary-by-cookie: "demo"

name: higress-demo-canary

spec:

ingressClassName: higress

rules:

- http:

paths:

- backend:

service:

name: nginx-v2

port:

number: 80

path: /hello

pathType: Exact

---

apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

name: higress-demo

spec:

ingressClassName: higress

rules:

- http:

paths:

- backend:

service:

name: nginx-v1

port:

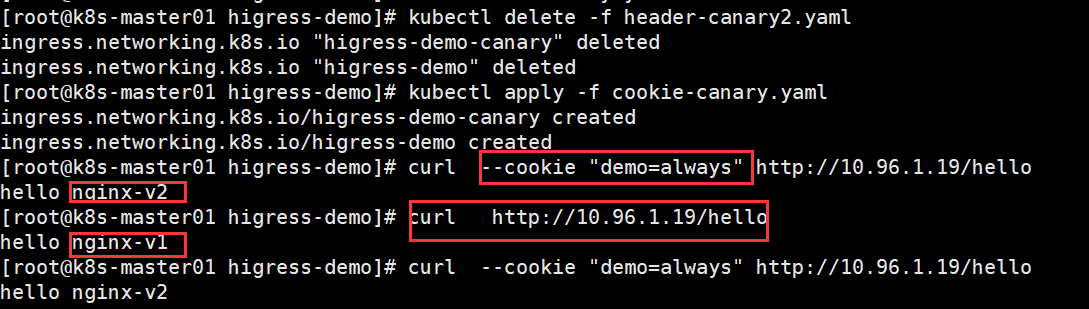
number: 80

path: /hello

pathType: Exact

部署后执行以下命令进行测试

curl --cookie "demo=always" http://10.96.1.19/hello



**基于权重灰度发布**[​](https://higress.io/zh-cn/docs/user/annotation-use-case#%E5%9F%BA%E4%BA%8E%E6%9D%83%E9%87%8D%E7%81%B0%E5%BA%A6%E5%8F%91%E5%B8%83)

higress.io/canary-weight：设置请求到指定服务的百分比（值为0~100的整数）

higress.io/canary-weight-totatl：设置权重总和，默认为100

配置灰度服务nginx-v2的权重为30%，配置正式服务nginx-v1的权重为70%。

apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

annotations:

higress.io/canary: "true"

higress.io/canary-weight: "30"

name: higress-demo-canary

spec:

ingressClassName: higress

rules:

- http:

paths:

- backend:

service:

name: nginx-v2

port:

number: 80

path: /hello

pathType: Exact

---

apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

name: higress-demo

spec:

ingressClassName: higress

rules:

- http:

paths:

- backend:

service:

name: nginx-v1

port:

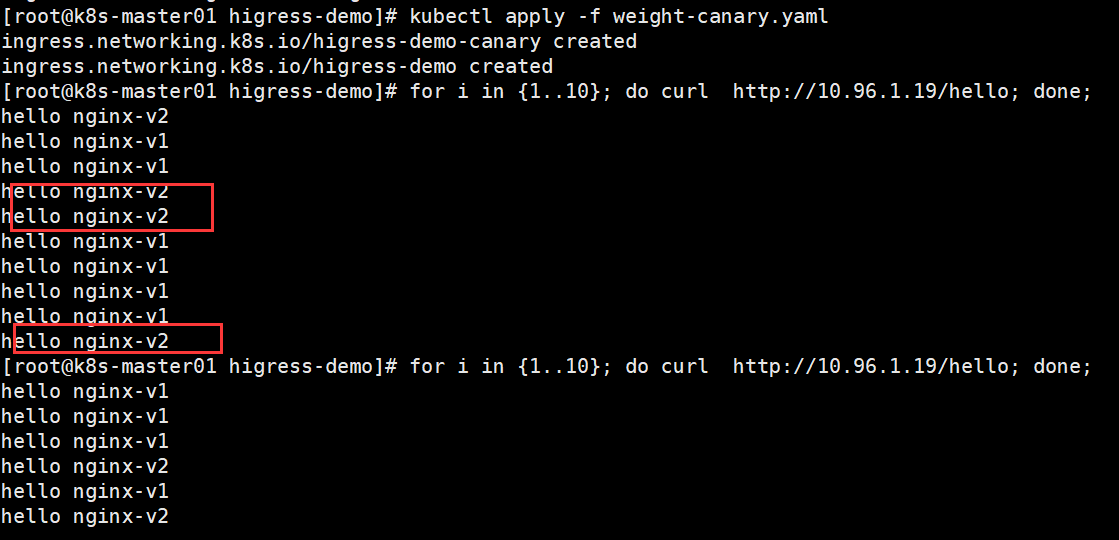
number: 80

path: /hello

pathType: Exact

部署后执行以下命令进行测试

for i in {1..10}; do curl -H "Host: canary.example.com" http://10.96.1.19/hello; done;



**2.2 跨域**

测试页面

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Title</title>

<script src="http://apps.bdimg.com/libs/jquery/1.9.1/jquery.min.js"></script>

</head>

<body>

<h3 id="demo"></h3>

</body>

<script>

$.get('http://192.168.65.130:30332/hello',function(data){

console.log(data)

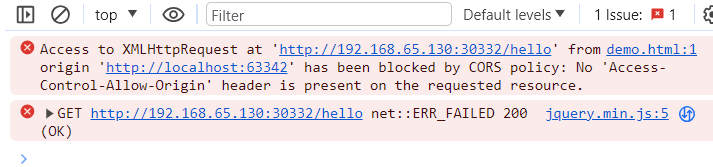
$('#demo').html(data);

});

</script>

</html>

存在跨域问题：



跨域资源共享CORS（Cross-Origin Resource Sharing）是指允许Web应用服务器进行跨域访问控制，从而实现跨域数据安全传输。

higress.io/enable-cors："true" or "false"。开启或关闭跨域。

higress.io/cors-allow-origin：允许的第三方站点，支持泛域名，逗号分隔；支持通配符**。**

higress.io/cors-allow-methods：允许的请求方法，如GET、POST，逗号分隔；支持通配符\*。默认值为GET, PUT, POST, DELETE, PATCH, OPTIONS。

higress.io/cors-allow-headers：允许的请求头部，逗号分隔；支持通配符\*。默认值为DNT,X-CustomHeader,Keep-Alive,User-Agent,X-Requested-With,If-Modified-Since,Cache-Control,Content-Type,Authorization。

higress.io/cors-expose-headers：允许的响应头部，逗号分隔。

higress.io/cors-allow-credentials："true" or "false"。是否允许携带凭证信息。默认允许。

higress.io/cors-max-age：预检结果的最大缓存时间，单位为秒；默认值为1728000。

apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

annotations:

higress.io/enable-cors: "true"

higress.io/cors-allow-origin: "\*"

higress.io/cors-allow-methods: "GET,POST"

higress.io/cors-allow-credentials: "false"

name: higress-demo

spec:

ingressClassName: higress

rules:

- http:

paths:

- backend:

service:

name: nginx-v1

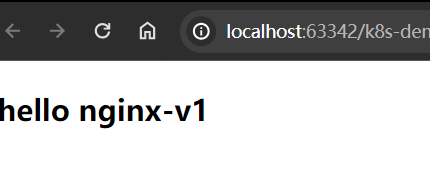
port:

number: 80

path: /hello

pathType: Exact

部署之后测试，可以正常访问



**2.3 Rewrite重写**

在请求转发给目标后端服务之前，重写可以修改原始请求的路径（Path）和主机域（Host)。

higress.io/rewrite-target：重写Path。

higress.io/upstream-vhost：重写Host。

**准备测试服务httpbin**

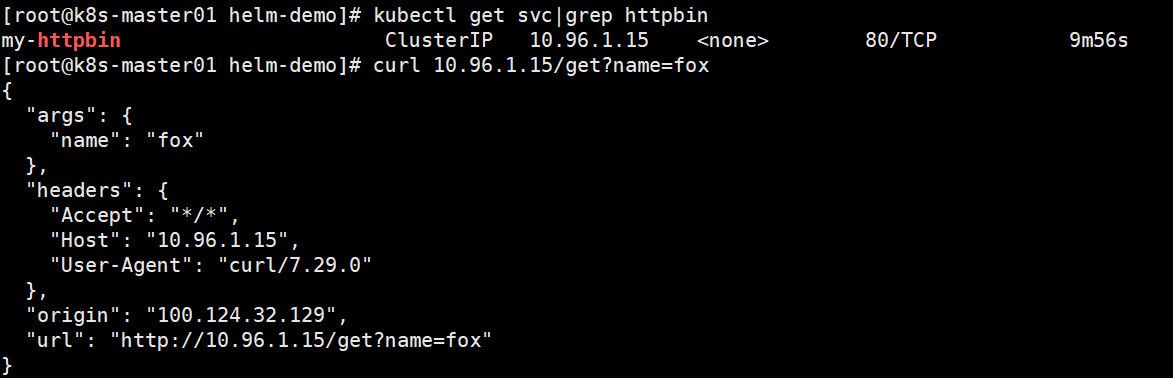
httpbin.org 一个简单的 HTTP 请求和响应服务，用 Python + Flask 编写。

官网地址：<https://httpbin.org/>

使用helm安装httpbin

helm repo add rgnu https://gitlab.com/mulesoft-int/helm-repository/-/raw/master/

helm install my-httpbin rgnu/httpbin --version 1.0.0



**Rewrite重写Path**[​](https://higress.io/zh-cn/docs/user/annotation-use-case#rewrite%E9%87%8D%E5%86%99path)

1. 将请求example.com/test在转发至后端服务之前，重写为example.com/get

apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

annotations:

higress.io/rewrite-target: "/get"

name: higress-demo

spec:

ingressClassName: higress

rules:

- host: example.com

http:

paths:

- backend:

service:

name: my-httpbin

port:

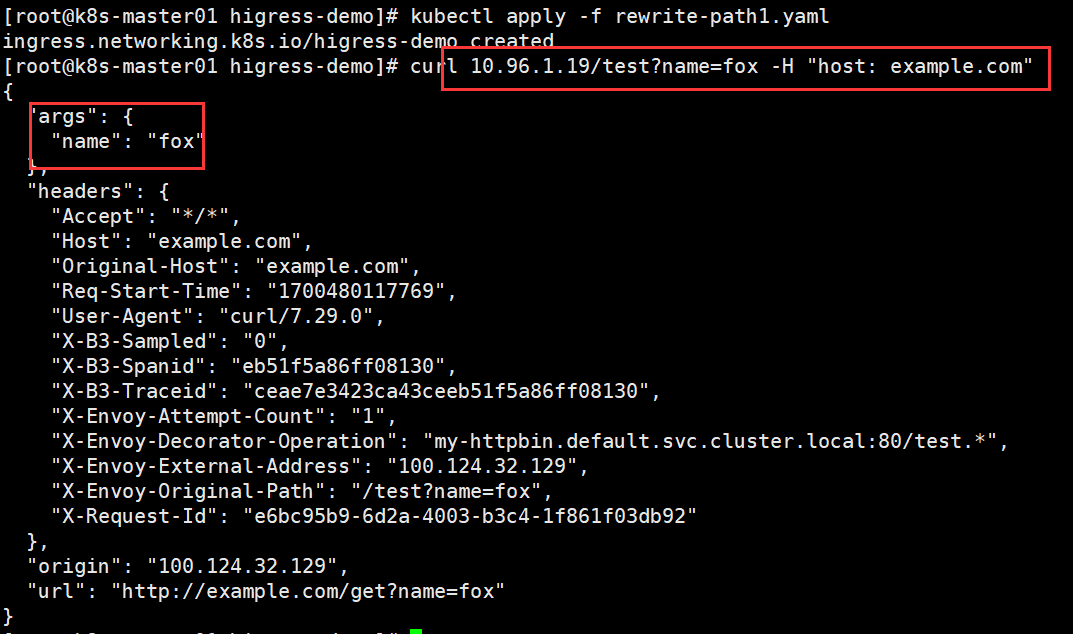
number: 80

path: /test

pathType: Exact

部署之后执行以下命令进行测试

curl 10.96.1.19/test?name=fox -H "host: example.com"



2.将请求example.com/v1/get在转发至后端服务之前，去掉Path前缀/v1

apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

annotations:

higress.io/rewrite-target: "/$2"

name: higress-demo

spec:

ingressClassName: higress

rules:

- host: example.com

http:

paths:

- backend:

service:

name: my-httpbin

port:

number: 80

path: /v1(/|$)(.\*)

pathType: ImplementationSpecific

$1表示路径中正则表达式匹配的第一个()的内容，$2为第二个，以此类推。

部署之后执行以下命令进行测试

curl 10.96.1.19/v1/get?name=fox -H "host: example.com"



**2.4 Header控制**[​](https://higress.io/zh-cn/docs/user/annotation-use-case#header%E6%8E%A7%E5%88%B6)

通过Header控制，您可以在转发请求到后端服务之前对请求Header进行增删改，在收到响应转发给客户端时对响应Header进行增删改。

**请求Header控制**[​](https://higress.io/zh-cn/docs/user/annotation-use-case#%E8%AF%B7%E6%B1%82header%E6%8E%A7%E5%88%B6)

higress.io/request-header-control-add：请求在转发给后端服务时，添加指定Header。若该Header存在，则其值拼接在原有值后面。语法如下：

单个Header：Key Value

多个Header：使用yaml特殊符号 |，每对Key Value单独处于一行

higress.io/request-header-control-update：请求在转发给后端服务时，修改指定Header。若该header存在，则其值覆盖原有值。语法如下：

单个Header：Key Value

多个Header：使用yaml特殊符号 |，每对Key Value单独处于一行

higress.io/request-header-control-remove：请求在转发给后端服务时，删除指定Header。语法如下：

单个Header：Key

多个Header：逗号分隔

1.对于请求example.com/headers添加两个Header，foo: bar和test: true。

apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

annotations:

higress.io/request-header-control-add: |

foo bar

test: true

name: demo

spec:

ingressClassName: higress

rules:

- host: example.com

http:

paths:

- backend:

service:

name: my-httpbin

port:

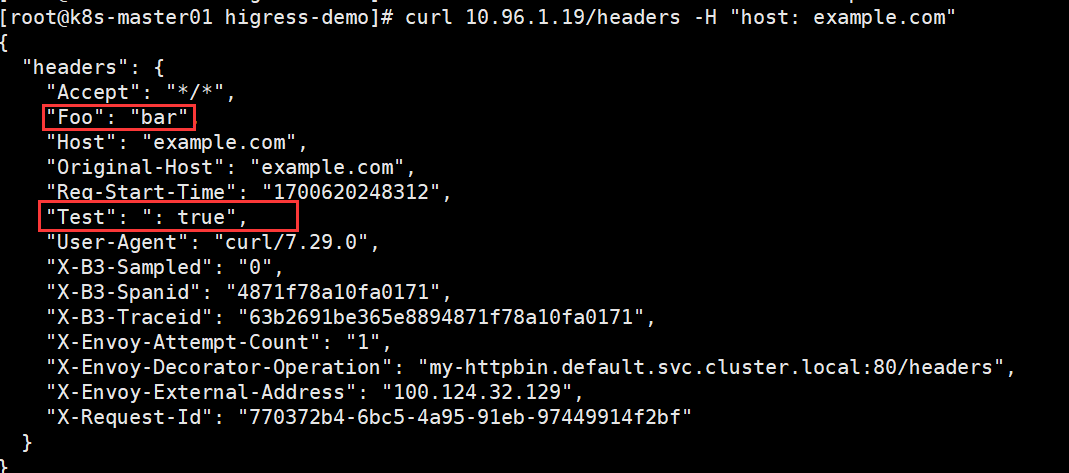
number: 80

path: /headers

pathType: Exact

部署之后执行以下命令进行测试

curl 10.96.1.19/headers -H "host: example.com"



2. Header控制可以结合灰度发布，对灰度流量进行染色。请求Header为higress：v2时将访问灰度服务nginx-v2，并添加Header，stage: gray；其他情况将访问正式服务nignx-v1，并添加Header，stage: prod。配置如下：

apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

annotations:

higress.io/canary: "true"

higress.io/canary-by-header: "higress"

higress.io/canary-by-header-value: "v2"

higress.io/request-header-control-add: "stage gray"

name: higress-demo-canary

spec:

ingressClassName: higress

rules:

- http:

paths:

- backend:

service:

name: my-httpbin-canary

port:

number: 80

path: /headers

pathType: Exact

---

apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

annotations:

higress.io/request-header-control-add: "stage prod"

name: higress-demo

spec:

ingressClassName: higress

rules:

- http:

paths:

- backend:

service:

name: my-httpbin

port:

number: 80

path: /headers

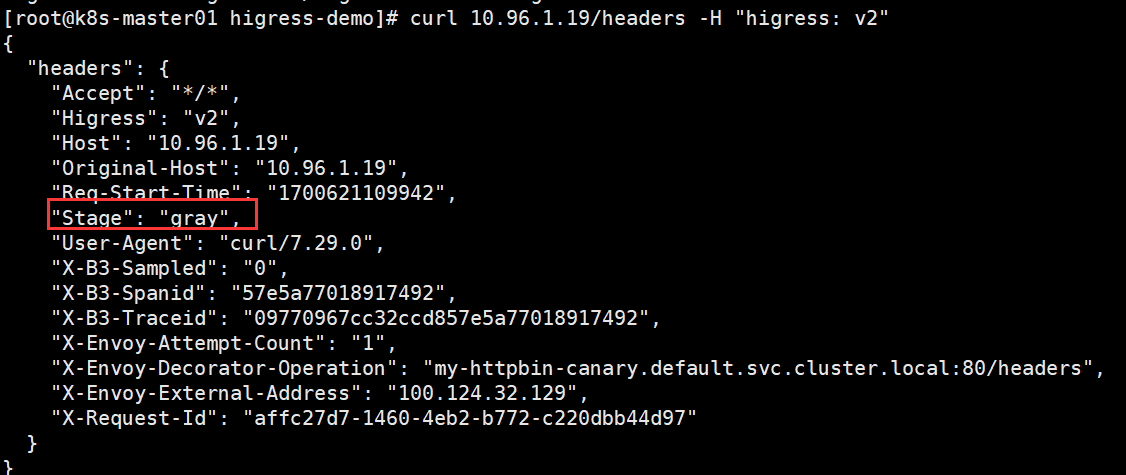
pathType: Exact

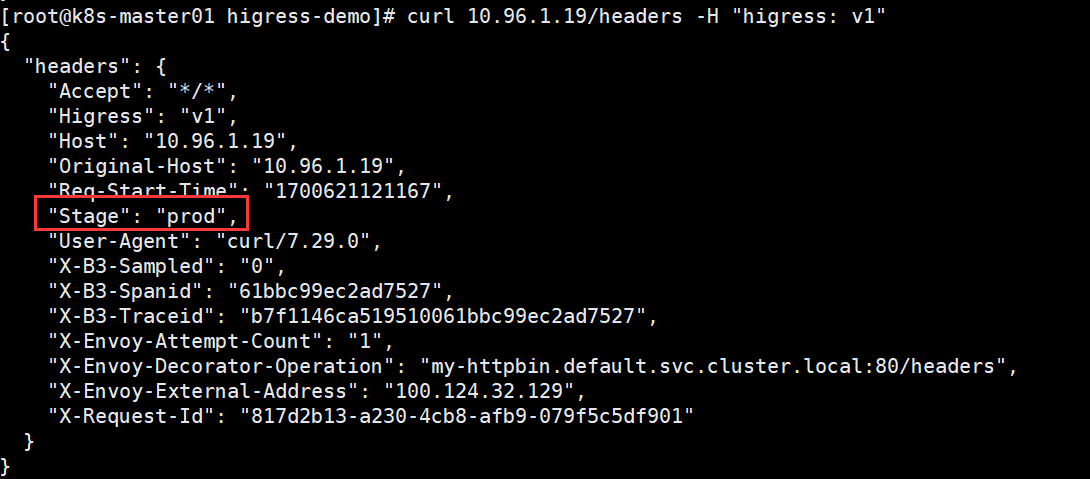
部署灰度版本的httpbin

helm install my-httpbin-canary rgnu/httpbin --version 1.0.0

部署之后执行以下命令进行测试

curl 10.96.1.19/headers -H "higress: v2"





**响应Header控制**[​](https://higress.io/zh-cn/docs/user/annotation-use-case#%E5%93%8D%E5%BA%94header%E6%8E%A7%E5%88%B6)

higress.io/response-header-control-add：请求在收到后端服务响应之后并且转发响应给客户端之前，添加指定Header。若该Header存在，则其值拼接在原有值后面。语法如下：

单个Header：Key Value

多个Header：使用yaml特殊符号 |，每对Key Value单独处于一行

higress.io/response-header-control-update：请求在收到后端服务响应之后并且转发响应给客户端之前，修改指定Header。若该header存在，则其值覆盖原有值。语法如下：

单个Header：Key Value

多个Header：使用yaml特殊符号 |，每对Key Value单独处于一行

higress.io/response-header-control-remove：请求在收到后端服务响应之后并且转发响应给客户端之前，删除指定Header。语法如下：

单个Header：Key

多个Header：逗号分隔

对于请求example.com/headers的响应删除Header：req-cost-time。

apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

annotations:

higress.io/response-header-control-remove: "req-cost-time"

name: higress-demo

spec:

ingressClassName: higress

rules:

- host: example.com

http:

paths:

- backend:

service:

name: my-httpbin

port:

number: 80

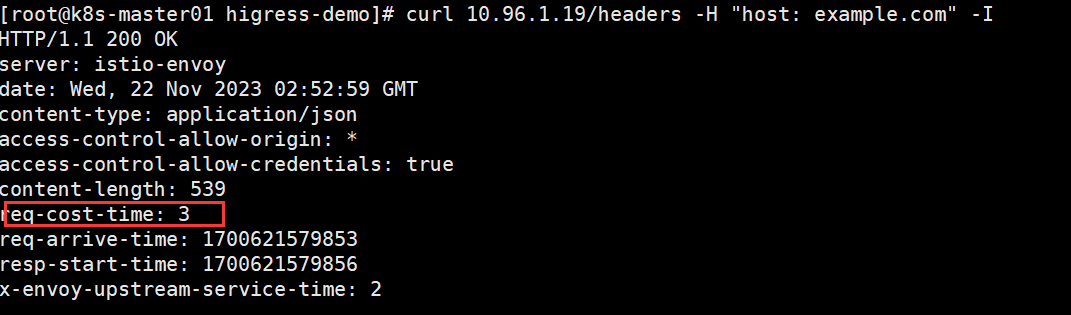
path: /headers

pathType: Exact

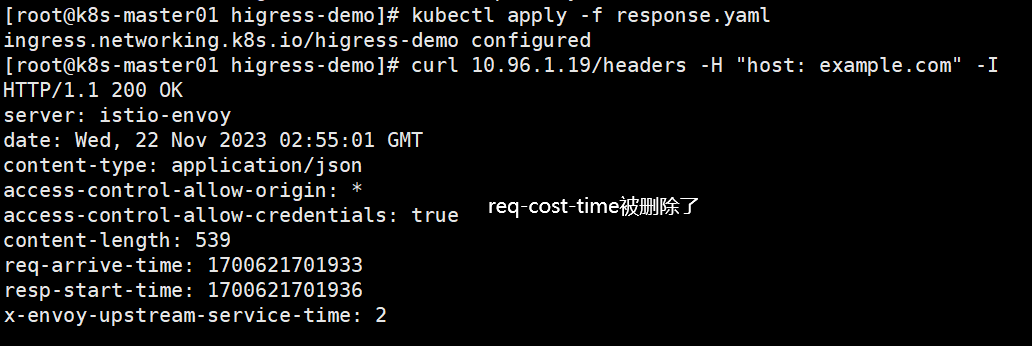
正常的响应结果

# -I 打印响应头

curl 10.96.1.19/headers -H "host: example.com" -I



部署后测试



**3. Higress插件实战**

<https://higress.io/zh-cn/docs/plugins/intro>

**3.1 通过 Higress 控制台进行配置**[​](https://higress.io/zh-cn/docs/plugins/intro#%E9%80%9A%E8%BF%87-higress-%E6%8E%A7%E5%88%B6%E5%8F%B0%E8%BF%9B%E8%A1%8C%E9%85%8D%E7%BD%AE)

Higress 控制台提供了 3 个入口进行插件配置:

全局配置：插件市场->选择插件进行配置

域名级配置：域名管理->选择域名->点击策略->选择插件进行配置

路由级配置: 路由配置->选择路由->点击策略->选择插件进行配置

这三个配置的生效优先级是: 路由级 > 域名级 > 全局

**3.2 通过 Higress WasmPlugin CRD 进行配置**[​](https://higress.io/zh-cn/docs/plugins/intro#%E9%80%9A%E8%BF%87-higress-wasmplugin-crd-%E8%BF%9B%E8%A1%8C%E9%85%8D%E7%BD%AE)

Higress WasmPlugin CRD 在 Istio [WasmPlugin](https://istio.io/latest/docs/reference/config/proxy_extensions/wasm-plugin/#WasmPlugin) CRD 的基础上进行了扩展，新增以下配置字段：

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名称 | 数据类型 | 填写要求 | 描述 |
| defaultConfig | object | 选填 | 插件默认配置，全局生效于没有匹配具体域名和路由配置的请求 |
| matchRules | array of object | 选填 | 匹配域名或路由生效的配置 |

matchRules中每一项的配置字段说明：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段名称 | 数据类型 | 填写要求 | 配置示例 | 描述 |
| ingress | array of string | ingress  和  domain  中必填一项 | ["default/foo","default/bar"] | 匹配 ingress 资源对象，匹配格式为:  命名空间/ingress名称 |
| domain | array of string | ingress  和  domain  中必填一项 | ["example.com","\*.test.com"] | 匹配域名，支持泛域名 |
| config | object | 选填 | - | 匹配后生效的插件配置 |

**请求屏蔽**[​](https://higress.io/zh-cn/docs/plugins/request-block#%E5%8A%9F%E8%83%BD%E8%AF%B4%E6%98%8E)

request-block插件实现了基于 URL、请求头等特征屏蔽 HTTP 请求，可以用于防护部分站点资源不对外部暴露

apiVersion: extensions.higress.io/v1alpha1

kind: WasmPlugin

metadata:

name: request-block

namespace: higress-system

spec:

defaultConfig:

block\_urls:

- swagger.html

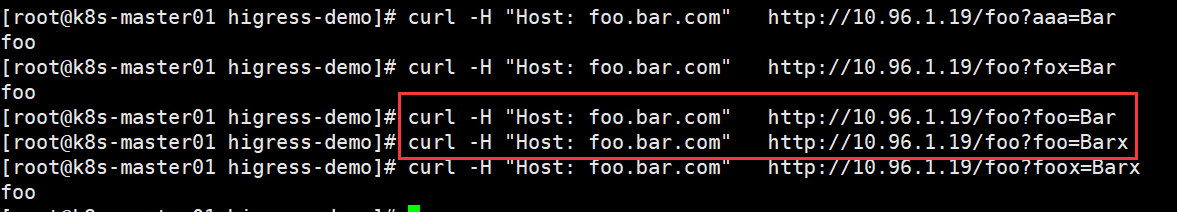
- foo=bar

case\_sensitive: false

url: oci://higress-registry.cn-hangzhou.cr.aliyuncs.com/plugins/request-block:1.0.0

部署后测试

curl -H "Host: foo.bar.com" http://10.96.1.19/foo?foo=Bar



**基于 Key 限流**

<https://higress.io/zh-cn/docs/plugins/key-rate-limit>[​](https://higress.io/zh-cn/docs/plugins/key-rate-limit#%E5%8A%9F%E8%83%BD%E8%AF%B4%E6%98%8E)

key-rate-limit插件实现了基于特定键值实现限流，键值来源可以是 URL 参数、HTTP 请求头

**识别请求头 x-api-key，进行区别限流**[​](https://higress.io/zh-cn/docs/plugins/key-rate-limit#%E8%AF%86%E5%88%AB%E8%AF%B7%E6%B1%82%E5%A4%B4-x-ca-key%E8%BF%9B%E8%A1%8C%E5%8C%BA%E5%88%AB%E9%99%90%E6%B5%81)

apiVersion: extensions.higress.io/v1alpha1

kind: WasmPlugin

metadata:

name: key-rate-limit-foo

spec:

matchRules:

- ingress:

- default/foo

config:

limit\_by\_header: "x-api-key"

limit\_keys:

- key: "example-key-a"

query\_per\_second: 1

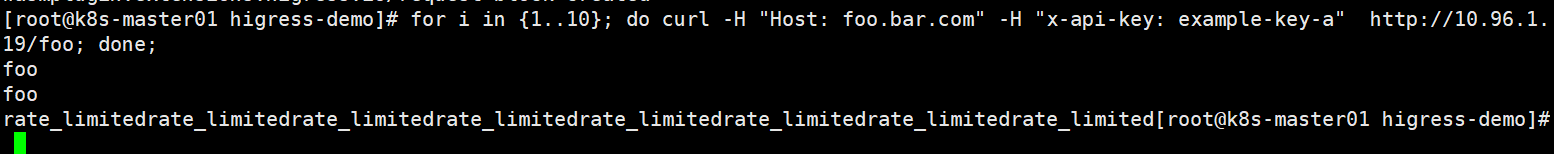
- key: "example-key-b"

query\_per\_minute: 3

url: oci://higress-registry.cn-hangzhou.cr.aliyuncs.com/plugins/key-rate-limit:1.0.0

部署后测试

for i in {1..10}; do curl -H "Host: foo.bar.com" -H "x-api-key: example-key-a" http://10.96.1.19/foo; done;

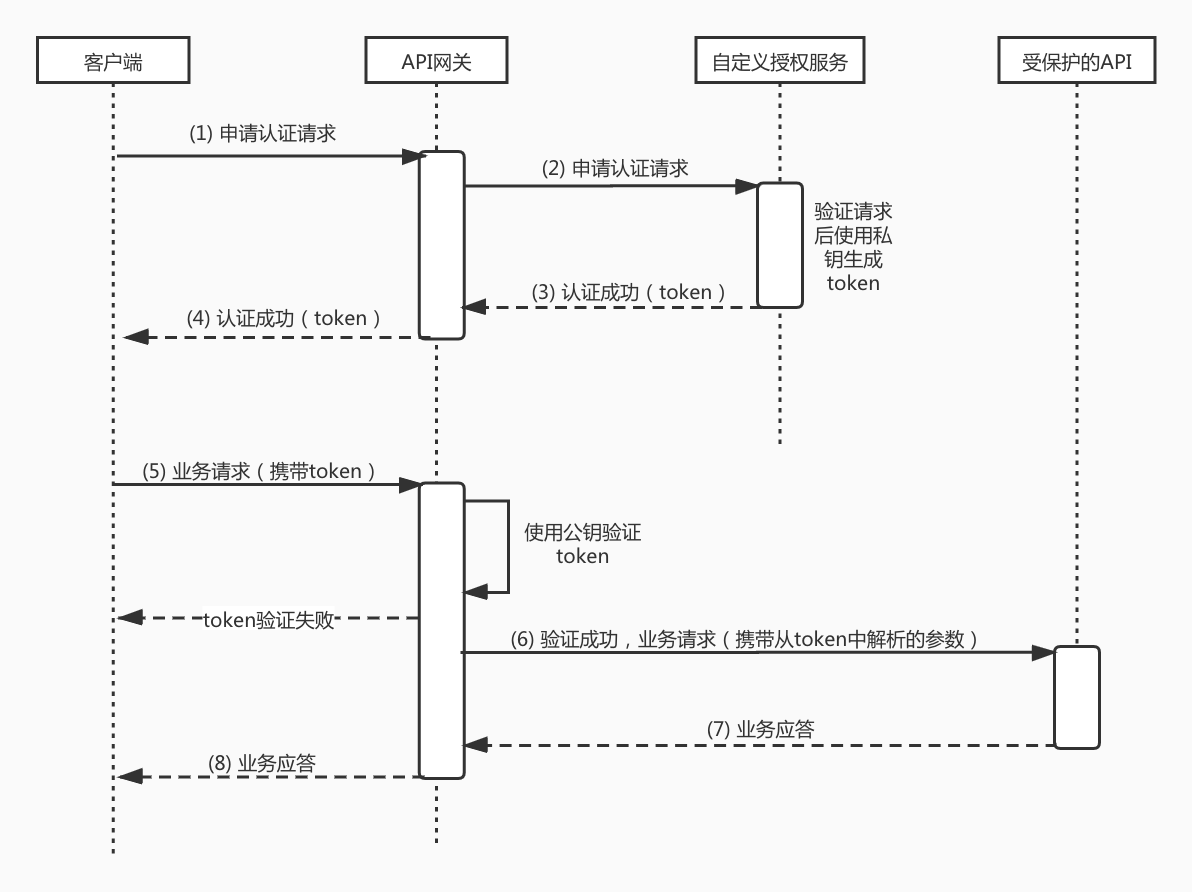


**JWT 认证**[​](https://higress.io/zh-cn/docs/plugins/jwt-auth#%E5%8A%9F%E8%83%BD%E8%AF%B4%E6%98%8E)

jwt-auth插件实现了基于JWT(JSON Web Tokens)进行认证鉴权的功能，支持从HTTP请求的URL参数、请求头、Cookie字段解析JWT，同时验证该Token是否有权限访问。

<https://higress.io/zh-cn/docs/plugins/jwt-auth>

**基于token的认证流程**



配置jwt插件

apiVersion: extensions.higress.io/v1alpha1

kind: WasmPlugin

metadata:

name: jwt-auth

namespace: higress-system

spec:

defaultConfig:

consumers:

- issuer: abcd

jwks: |

{

"keys": [

{

          "kty": "oct",

          "kid": "123",

          "k": "hM0k3AbXBPpKOGg\_\_Ql2Obcq7s60myWDpbHXzgKUQdYo7YCRp0gUqkCnbGSvZ2rGEl4YFkKqIqW7mTHdj-bcqXpNr-NOznEyMpVPOIlqG\_NWVC3dydBgcsIZIdD-MR2AQceEaxriPA\_VmiUCwfwL2Bhs6\_i7eolXoY11EapLQtutz0BV6ZxQQ4dYUmct--7PLNb4BWJyQeWu0QfbIthnvhYllyl2dgeLTEJT58wzFz5HeNMNz8ohY5K0XaKAe5cepryqoXLhA-V-O1OjSG8lCNdKS09OY6O0fkyweKEtuDfien5tHHSsHXoAxYEHPFcSRL4bFPLZ0orTt1\_4zpyfew",

          "alg": "HS256"

        }

]

}

name: consumer1

global\_auth: false

matchRules:

- config:

allow:

- consumer1

ingress:

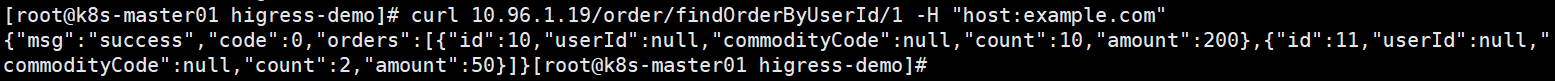
- default/higress-demo

url: oci://higress-registry.cn-hangzhou.cr.aliyuncs.com/plugins/jwt-auth:1.0.0

测试效果

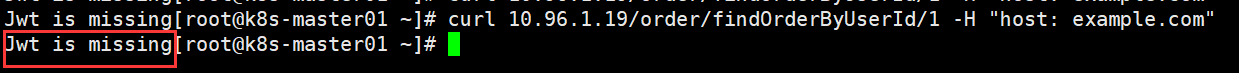
未部署jwt插件之前，执行如下命令

curl 10.96.1.19/order/findOrderByUserId/1 -H "host: example.com"



部署jwt插件之后，执行如下命令

curl 10.96.1.19/order/findOrderByUserId/1 -H "host: example.com"



将 JWT 设置在 http 请求头中

curl 10.96.1.19/order/findOrderByUserId/1 -H "host: example.com" -H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCIsImtpZCI6IjEyMyJ9.eyJpc3MiOiJhYmNkIiwic3ViIjoidGVzdCIsImlhdCI6MTY2NTY2MDUyNywiZXhwIjoxODY1NjczODE5fQ.-vBSV0bKeDwQcuS6eeSZN9dLTUnSnZVk8eVCXdooCQ4'

