- Before
  - Terminology
  - 准备
- 简介
  - 基本用法
    - 配置说明
    - DefaultBackend
    - PathType
    - 主机名匹配
- 配置
  - 全局设置
  - 注解
    - 身份验证
    - 金丝雀
    - Rewrite
    - Configuration snippet
    - Server snippet
    - Client Body Buffer Size
    - Redirect from/to www¶
    - Custom timeouts
    - Custom max body size
  - 暴露TCP/UDP 服务
- 举个栗子
  - 一个简单的栗子
  - 加身份验证
    - 创建用户名和密码
    - 创建 k8s secret 来存储 用户/密码
    - 创建 ingress
    - 验证

Before

Terminology

- `节点(Node)`：Kubernetes 集群中其中一台工作机器，是集群的一部分。
- `集群(Cluster)`：一组运行由 Kubernetes 管理的容器化应用程序的节点。 在此示例和在大多数常见的 Kubernetes 部署环境中，集群中的节点都不在公共网络中。
- `边缘路由器(Edge router)`：在集群中强制执行防火墙策略的路由器（router）。 可以是由云提供商管理的网关，也可以是物理硬件。
- `集群网络(Cluster network)`：一组逻辑的或物理的连接，根据 Kubernetes 网络模型 在集群内实现通信。
- `服务(Service)`： Kubernetes 服务使用 标签选择算符（selectors）标识的一组 Pod。 除非另有说明，否则假定服务只具有在集群网络中可路由的虚拟 IP。
- `k8s`：指kubernetes, k根s之间正好是8个字母
- `k8s集群外部主机`：没有加入k8s集群的主机，不管是不是根k8s集群主机在同一个网段内，都叫k8s集群外部主机
- `Load Balancer` ：负载均衡器

准备

- `k8s` ：1.18+

简介

Kubernetes 提供的发布服务的方式：

- `ClusterIP`： 通过集群的内部 IP 暴露服务，选择该值时服务只能够在集群内部访问。 这也是默认的 ServiceType。

- `NodePort`： 通过每个节点上的 IP 和静态端口（NodePort）暴露服务。 NodePort 服务会路由到自动创建的 ClusterIP 服务。 通过请求 <节点 IP>:<节点端口 30000-32767>，你可以从集群的外部访问一个 NodePort 服务。
- `LoadBalancer`： 使用云提供商的负载均衡器向外部暴露服务。 外部负载均衡器可以将流量路由到自动创建的 NodePort 服务和 ClusterIP 服务上。
- `ExternalName`： 通过返回 CNAME 和对应值，可以将服务映射到 externalName 字段的内容（例如，foo.bar.example.com）。 无需创建任何类型代理。
- `externalIPs` ：如果外部的 IP 路由到集群中一个或多个 Node 上，Kubernetes Service 会被暴露给这些 externalIPs，`externalIPs` 不会被 Kubernetes 管理

```
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  type: NodePort
  selector:
    app: MyApp
  ports:
      # 默认情况下,`targetPort` 被设置为与 `port` 字段相同的值。
    - port: 80
      targetPort: 80
      # 可选字段
      # 默认情况下，Kubernetes 会从30000-32767范围内分配一个端口号
      nodePort: 30007
```

上面可以暴露给外面用的： `NodePort` 、 `LoadBalancer` 、 `externalIPs`

`Ingress` 是 k8s 为服务提供外部可访问的另一种方式。Ingress 不是一种服务类型，但它充当集群的入口点。 它可以将路由规则整合到一个资源中，因为它可以在同一IP地址下公开多个服务。

基本用法

下面通过一个最简单的示例来对配置做个简单的说明：

`test-ingress.yaml`

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: minimal-ingress
  annotations:
    # k8s 1.18 版本以后用 `ingressClassName` 替代
    # kubernetes.io/ingress.class: nginx
    nginx.ingress.kubernetes.io/rewrite-target: /
    # 最多只能有一个 IngressClass 被标记为默认
    ingressclass.kubernetes.io/is-default-class: true
spec:
  ingressClassName: nginx
  rules:
  - http:
      paths:
      - path: /testpath
        pathType: Prefix
        backend:
          service:
            name: test
            port:
              number: 80


kubectl apply -f test-ingress.yaml
```

与其他 k8s 资源一样， `ingress` 也是具备 `apiVersion` `kind` 和 `metadata` 等字段。

配置说明

- `host`：上面示例没有指定host,因此上述配置是适用于通过指定IP地址来访问服务的。
- `paths`：每个 `path` 都是由一个 service 来关联指定的服务的。
- `backend` ：是由 service 组成的。用来绑定 service

DefaultBackend

通常没有 `rules` 或配置的 `hosts` 、 `paths` 没有与 `ingress` 中的 http 请求匹配的，则会把流量发送到 `defaultBackend` 。

PathType

Ingress 中的每个 `path` 都要配置相应的 `Path Type` 。
目前支持的类型有三种：

- `ImplementationSpecific`
  这个类型匹配方式取决于 `IngressClass`
- `Exact`
  精确匹配 `URL` ,且区分大小写
- `Prefix`
  基于以 `/` 分隔的 URL 路径前缀匹配。区分大小写。
  `/foo/bar` 匹配 `/foo/bar/baz` , 但不匹配 `/foo/barbaz`

配置示例参考：

类型	路径	请求路径	匹配与否?
Prefix	`/`	（所有路径）	✓
Exact	`/foo`	`/foo`	✓
Exact	`/foo`	`/bar`	✕
Exact	`/foo`	`/foo/`	✕
Exact	`/foo/`	`/foo`	✕
Prefix	`/foo`	`/foo` , `/foo/`	✓
Prefix	`/foo/`	`/foo` , `/foo/`	✓
Prefix	`/aaa/bb`	`/aaa/bbb`	✕
Prefix	`/aaa/bbb`	`/aaa/bbb`	✓
Prefix	`/aaa/bbb/`	`/aaa/bbb`	✓，忽略尾部斜线
Prefix	`/aaa/bbb`	`/aaa/bbb/`	✓，匹配尾部斜线
Prefix	`/aaa/bbb`	`/aaa/bbb/ccc`	✓，匹配子路径
Prefix	`/aaa/bbb`	`/aaa/bbbxyz`	✕，字符串前缀不匹配
Prefix	`/` , `/aaa`	`/aaa/ccc`	✓，匹配 `/aaa` 前缀
Prefix	`/` , `/aaa` , `/aaa/bbb`	`/aaa/bbb`	✓，匹配 `/aaa/bbb` 前缀
Prefix	`/` , `/aaa` , `/aaa/bbb`	`/ccc`	✓，匹配 `/` 前缀
Prefix	`/aaa`	`/ccc`	✕，使用默认后端
混合	`/foo` （Prefix），`/foo` （Exact）	`/foo`	✓，优选 Exact 类型

主机名匹配

主机	host 头部	匹配与否?
`*.foo.com`	`bar.foo.com`	基于相同的后缀匹配
`*.foo.com`	`baz.bar.foo.com`	不匹配，通配符仅覆盖了一个 DNS 标签
`*.foo.com`	`foo.com`	不匹配，通配符仅覆盖了一个 DNS 标签

配置

在 `ingress` 中配置分为了三部分：

- `configmap` ：设置 ingress-nginx 的全局设置
- `annotations` ： ingress 特定的设置
- `custom template` ：

全局设置

全局配置一般是使用 configmap 。
下面是一个 ingress-controller 的配置

```yaml
kind: ConfigMap
apiVersion: v1
metadata:
  name: ingress-nginx-controller
  namespace: ingress-nginx
  labels:
    app.kubernetes.io/component: controller
    app.kubernetes.io/instance: ingress-nginx
    app.kubernetes.io/managed-by: Helm
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/version: 0.40.2
    helm.sh/chart: ingress-nginx-3.6.0
  annotations:
data:
  allow-backend-server-header: 'true'
  client-body-buffer-size: 20m
  enable-underscores-in-headers: 'true'
  generate-request-id: 'true'
  gzip-level: '6'
  gzip-types: >-
    application/atom+xml application/javascript application/x-javascript
    application/json application/rss+xml application/vnd.ms-fontobject
    application/x-font-ttf application/x-web-app-manifest+json
    application/xhtml+xml application/xml font/opentype image/svg+xml
    image/x-icon text/css text/javascript text/plain text/x-component
  ignore-invalid-headers: 'true'
  keep-alive: '75'
  large-client-header-buffers: 4 128k
  log-format-upstream: >-
    $remote_addr - [$remote_addr] - $remote_user [$time_local] "$request"
    $status $body_bytes_sent "$http_referer" "$http_user_agent" $request_length
    "$http_x_forwarded_for" $remote_addr $request_time [$proxy_upstream_name]
    $upstream_addr $upstream_response_length $upstream_response_time
    $upstream_status $req_id $host
  max-worker-connections: '65536'
  proxy-body-size: 20m
  proxy-buffer-size: 64k
  proxy-connect-timeout: '300'
  proxy-next-upstream-timeout: '10'
  proxy-read-timeout: '300'
  proxy-send-timeout: '300'
  reuse-port: 'true'
  server-tokens: 'false'
  upstream-keepalive-connections: '20000'
  upstream-keepalive-requests: '100000'
  upstream-keepalive-timeout: '3000'
  use-forwarded-headers: 'true'
  use-gzip: 'true'
  worker-cpu-affinity: auto
```

这个就相当于 `nginx` 的 `nginx.conf` 配置文件

```
user www-data;
worker_processes auto;
pid /run/nginx.pid;
include /etc/nginx/modules-enabled/*.conf;

events {
        worker_connections 768;
        multi_accept on;
}

http {

        ##
        # Basic Settings
        ##

        sendfile on;
        tcp_nopush on;
        tcp_nodelay on;
        keepalive_timeout 65;
        types_hash_max_size 2048;
        server_tokens off;

        # server_names_hash_bucket_size 64;
        # server_name_in_redirect off;

        include /etc/nginx/mime.types;
        default_type application/octet-stream;

        ##
        # SSL Settings
        ##

        ssl_protocols TLSv1 TLSv1.1 TLSv1.2; # Dropping SSLv3, ref: POODLE
        ssl_prefer_server_ciphers on;

        ##
        # Logging Settings
        ##

        access_log /var/log/nginx/access.log;
        error_log /var/log/nginx/error.log;

        ##
        # body size
        ##
        client_max_body_size 10m;
        proxy_read_timeout   300;

        ##
        # Gzip Settings
        ##

        gzip on;
        gzip_vary on;
        gzip_proxied any;
        gzip_comp_level 6;
        gzip_buffers 16 8k;
        gzip_http_version 1.1;
        gzip_types text/plain text/css application/json application/javascript text/xml application/xml application/xml+rss text/javascript;

        ##
        # Virtual Host Configs
        ##

        include /etc/nginx/conf.d/*.conf;
        include /etc/nginx/sites-enabled/*;
}
```

常用配置对照表：

名称	nginx 名称	说明	类型	默

名称	nginx 名称	说明	类型	默
worker-processes	worker_processes	设置 worker processes 数量, 默认值"auto"表示可用的CPU内核数	string	auto
max-worker-connections	worker_connections	设置 单个 worker 打开的最大同时连接数	int	16384
enable-multi-accept	multi_accept	如果禁用，worker进程将一次接受一个新连接。 否则， 工作进程将一次接受所有新连接	bool	true
keep-alive	keepalive_timeout	设置保持活动的客户端连接在服务器端保持打开状态的时间。 零值将禁用保持活动状态的客户端连接。	int	75
upstream-keepalive-timeout	keepalive_timeout	设置一个超时，在此超时期间， 与上游服务器的空闲 keepalive 连接将保持打开状态	int	60
server-tokens	server_tokens	在响应中发送nginx Server标头， 并在错误页面中显示NGINX版本	bool	true
server-name-hash-bucket-size	server_names_hash_bucket_size	int	``	
proxy-body-size	client_max_body_size	设置客户端请求正文的最大允许大小	string	1m
proxy-read-timeout	proxy-read-timeout	以秒为单位为从代理服务器读取相应设置超时	int	60
use-gzip	gzip	启用或禁用HTTP响应的压缩	bool	true
gzip-types	gzip_types	设置除"text/html"之外的MIME类型以进行压缩。 特殊值"*"与任何MIME类型匹配	string	"applicati applicatic applicatic javascript applicatic applicatic applicatic fontobjec applicatic applicatic app-mani applicatic applicatic font/open image/sv image/x-i text/javas text/plain text/x-cor
gzip-level	gzip_comp_level	设置将使用的gzip压缩级别	int	1

注解

注解都是以 nginx.ingress.kubernetes.io 作为前缀的,添加到特定的ingress实例上的。

身份验证

```
# http 身份验证的类型
nginx.ingress.kubernetes.io/auth-type: [basic|digest]

# Secret的名称，其中包含用户名和密码
nginx.ingress.kubernetes.io/auth-secret: secretName

# auth-secret 有两种格式:
# auth-file 默认情况下，密钥'auth'中的htpasswd文件在密钥内
# auth-map 密钥的密钥是用户名，值是哈希密码
nginx.ingress.kubernetes.io/auth-secret-type: [auth-file|auth-map]

nginx.ingress.kubernetes.io/auth-realm: "realm string"
```

金丝雀

这个之前有专门的介绍，这里就不废话了

```
nginx.ingress.kubernetes.io/canary-by-header

nginx.ingress.kubernetes.io/canary-by-header-value

nginx.ingress.kubernetes.io/canary-by-cookie

nginx.ingress.kubernetes.io/canary-by-cookie
```

Rewrite

在某些情况下，后端服务中公开的URL与Ingress规则中指定的路径不同.
没有重写，任何请求都将返回404.
设置 `nginx.ingress.kubernetes.io/rewrite-target` 注解到服务期望的路径.

如果应用程序根目录暴露在其他路径中，并且需要重定向，请设置注释
`nginx.ingress.kubernetes.io/app-root` 重定向请求到 `/` .

Configuration snippet

使用此注释，您可以将其他配置添加到NGINX位置。 例如:

```
nginx.ingress.kubernetes.io/configuration-snippet: |
  more_set_headers "Request-Id: $req_id";
```

Server snippet

使用注解 `nginx.ingress.kubernetes.io/server-snippet` 可以在服务器配置块中添加自定义配置.

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  annotations:
    nginx.ingress.kubernetes.io/server-snippet: |
        set $agentflag 0;

        if ($http_user_agent ~* "(Mobile)" ){
          set $agentflag 1;
        }

        if ( $agentflag = 1 ) {
          return 301 https://m.example.com;
        }
```

Client Body Buffer Size

设置缓冲区大小，以读取每个位置的客户端请求正文。 如果请求主体大于缓冲区， 整个身体或只将其一部分写入一个临时文件。 默认情况下，缓冲区大小等于两个内存页。 在x86，其他32位平台和x86-64上为8K。 在其他64位平台上，通常为16K。 该注解是 应用于入口规则中提供的每个位置

```
nginx.ingress.kubernetes.io/client-body-buffer-size: "1000" # 1000 bytes
nginx.ingress.kubernetes.io/client-body-buffer-size: 1k # 1 kilobyte
nginx.ingress.kubernetes.io/client-body-buffer-size: 1K # 1 kilobyte
nginx.ingress.kubernetes.io/client-body-buffer-size: 1m # 1 megabyte
nginx.ingress.kubernetes.io/client-body-buffer-size: 1M # 1 megabyte
```

Redirect from/to www¶

在某些情况下，需要从www.domain.com 重定向到 domain.com ， 反之亦然. 使用 `nginx.ingress.kubernetes.io/from-to-www-redirect` : "true"注解开启此功能

Custom timeouts

使用配置configmap可以为 `ingress-nginx` 设置默认的全局超时。 在某些情况下，要求具有不同的值。 为此，我们提供了允许进行此自定义的注释:

```
nginx.ingress.kubernetes.io/proxy-connect-timeout
nginx.ingress.kubernetes.io/proxy-send-timeout
nginx.ingress.kubernetes.io/proxy-read-timeout
nginx.ingress.kubernetes.io/proxy-next-upstream
nginx.ingress.kubernetes.io/proxy-next-upstream-timeout
nginx.ingress.kubernetes.io/proxy-next-upstream-tries
nginx.ingress.kubernetes.io/proxy-request-buffering
```

Custom max body size

当请求中的大小超过客户端请求正文的最大允许大小时，将向客户端返回413错误. 大小可以通过 `client_max_body_size` 参数配置.
与上面类似,想要对所有ingress规则进行全局设置, proxy-body-size 可以在 nginx ConfigMap中设置. 要在Ingress规则中使用自定义值，请定义以下注解:

```
nginx.ingress.kubernetes.io/proxy-body-size: 8m
```

暴露TCP/UDP 服务

ingress不支持TCP或UDP服务。
因此，需要在 Ingress Controller 使用标志 --tcp-services-configmap and --udp-services-configmap 指向现有的配置映射，其中的键是要使用的外部端口，并且该值指示使用以下格式公开的服务:

```
<namespace/service name>:<service port>:[PROXY]:[PROXY]
```

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: tcp-services
  namespace: ingress-nginx
data:
  9000: "default/example-go:8080"

---

kind: ConfigMap
apiVersion: v1
metadata:
  name: tcp-services
  namespace: ingress-nginx
  labels:
    app.kubernetes.io/component: controller
    app.kubernetes.io/instance: ingress-nginx
    app.kubernetes.io/managed-by: Helm
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/version: 0.33.0
    helm.sh/chart: ingress-nginx-2.10.0
data:
  '5200': 'anxincloud/et-upload:15200'
  '8888': 'smart-xxx/comfortablehome-datapush:18081'
  '19005': 'free-sun/broadcast-server:19003'
```

举个栗子

一个简单的栗子

```yaml
# deploy.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: http-svc
  namespace: dragon
spec:
  replicas: 1
  selector:
    matchLabels:
      app: http-svc
  template:
    metadata:
      labels:
        app: http-svc
    spec:
      volumes:
        - name: myvolume
          persistentVolumeClaim:
            claimName: myclaim
      containers:
      - name: http-svc
        image: repository.anxinyun.cn/devops/nginx-test:0.01
        ports:
        - containerPort: 80
---
apiVersion: v1
kind: Service
metadata:
  name: http-svc
  namespace: dragon
  labels:
    app: http-svc
spec:
  ports:
  - port: 8080
    targetPort: 80
    protocol: TCP
    name: http
    nodePort: 30000
  type: NodePort
  selector:
    app: http-svc
```

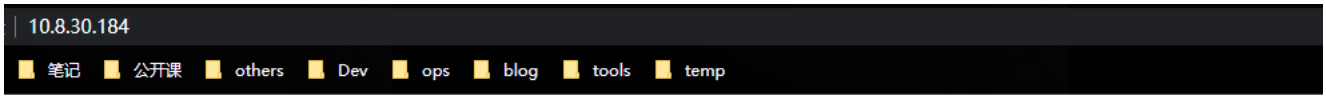
```yaml
# ingress-demo.yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ingress-demo
  namespace: dragon
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
    - http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: http-svc
                port:
                  number: 8080
```

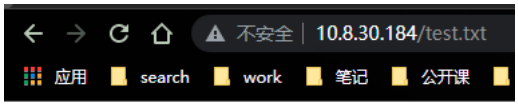
```
kubectl apply -f deploy.yaml
```

```
$ kubectl get po -n dragon
NAME                        READY   STATUS    RESTARTS   AGE
http-svc-7f994b6445-ff59j   1/1     Running   0          26h
```

```
kubectl apply -f ingress-demo.yaml
```

```
$ kubectl get ingress -n dragon
NAME                     CLASS    HOSTS   ADDRESS   PORTS   AGE
ingress-demo   public   *                 80      3d6h
```






```
Welcome to nginx!
hello ...
test
```

加身份验证

创建用户名和密码

需要通过 htpasswd 来生成一个auth 文件用来存取我们创建的用户和加密后的密码。
先要安装 htpasswd

```
$ sudo apt -y install apache2-utils
```

下面生成 auth 文件

```
$ htpasswd -c auth admin
New password:
Re-type new password:
Adding password for user admin
$ htpasswd auth test
New password:
Re-type new password:
Adding password for user test
```

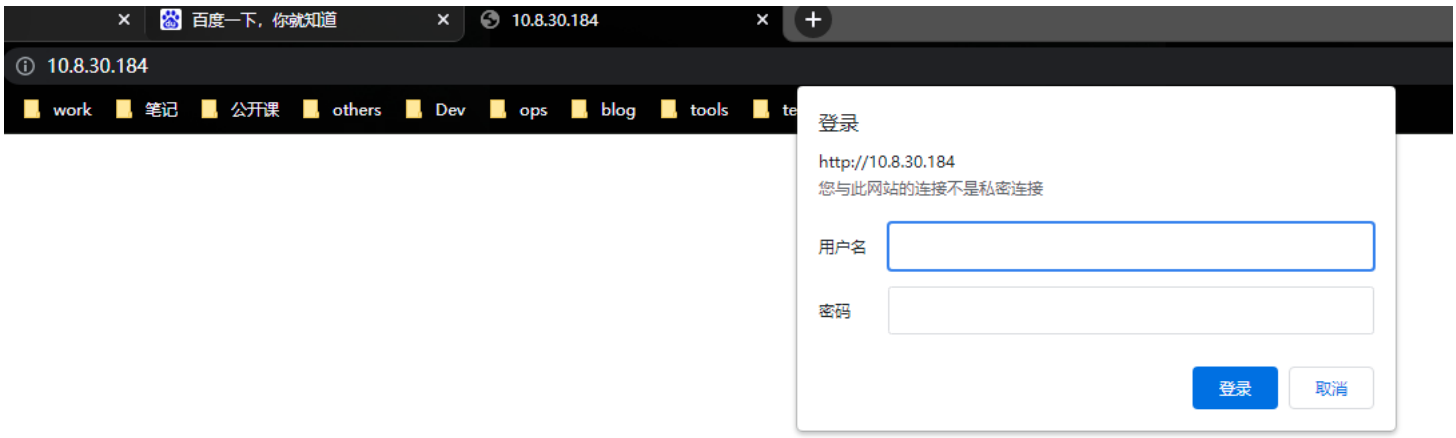
创建 k8s secret 来存储 用户/密码

```
$ kubectl -n dragon create secret generic basic-auth --from-file=auth
secret/basic-auth created

$ kubectl get secret basic-auth -n dragon -o yaml
apiVersion: v1
data:
  auth: YWRtaW46JGFwcjEkMmlzL21oMkskbXlNb0VmSzJoQVRsUmV2RDNGSmdLLgp0ZXN0OiRhcHIxJC9yYUllQjVYJGJROHcxOVhFMkU2ZTlFSTZLb1JScC8K
kind: Secret
metadata:
  creationTimestamp: "2021-11-18T09:34:13Z"
  name: basic-auth
  namespace: dragon
  resourceVersion: "826578"
  selfLink: /api/v1/namespaces/dragon/secrets/basic-auth
  uid: 05b06023-6cab-428f-819e-1bc152db5bbb
type: Opaque
```

创建 ingress

```
# ingress-demo.yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ingress-demo
  namespace: dragon
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
    nginx.ingress.kubernetes.io/auth-type: basic
    nginx.ingress.kubernetes.io/auth-secret: basic-auth
    nginx.ingress.kubernetes.io/auth-realm: "Authentication Required - admin"
spec:
  rules:
    - http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: http-svc
                port:
                  number: 8080


$ kubectl apply -f ingress-demo.yaml
ingress.networking.k8s.io/ingress-demo configured
```

验证



定制配置

```yaml
# ingress-demo.yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ingress-demo
  namespace: dragon
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
    nginx.ingress.kubernetes.io/auth-type: basic
    nginx.ingress.kubernetes.io/auth-secret: basic-auth
    nginx.ingress.kubernetes.io/auth-realm: "Authentication Required - admin"
    nginx.ingress.kubernetes.io/proxy-body-size: 8m
spec:
  rules:
    - http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: http-svc
                port:
                  number: 8080
```