Kubernetes 七层负载均衡 Ingress

前言：

课程名称：Kubernetes 七层负载均衡 Ingress

实验环境：

本章节 Kubernetes 集群环境如下：

角色	IP	主机名	组件	硬件
控制节点	192.168.128.11	k8s-master01	apiserver controller-manager scheduler etcd containerd	CPU：4vCPU 硬盘：100G 内存：4GB 开启虚拟化
工作节点	192.168.128.21	k8s-node01	kubelet kube-proxy containerd calico coredns	CPU：6vCPU 硬盘：100G 内存：8GB 开启虚拟化
工作节点	192.168.128.22	k8s-node02	kubelet kube-proxy containerd calico coredns	CPU：6vCPU 硬盘：100G 内存：8GB 开启虚拟化

官网手册地址：https://kubernetes.github.io/ingress-nginx/

张岩峰老师微信，加我微信，邀请你加入 VIP 交流答疑群：

微信号：ZhangYanFeng0429

二维码：

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

1、初识 Ingress

1.1、四层负载均衡和七层负载均衡对比

1、四层的负载均衡就是基于（IP+端口）的负载均衡：在三层负载均衡的基础上，通过发布三层的 IP 地址（VIP），然后加四层的端口号，来决定哪些流量需要做负载均衡，对需要处理的流量进行 NAT 处理，转发至后台服务器，并记录下这个 TCP 或者 UDP 的流量是由哪台服务器处理的，后续这个连接的所有流量都同样转发到同一台服务器处理。

2、七层的负载均衡就是基于虚拟的 URL 或主机 IP 的负载均衡：在四层负载均衡的基础上（没有四层是绝对不可能有七层的），再考虑应用层的特征，比如同一个 Web 服务器的负载均衡，除了根据 VIP 加 80 端口辨别是否需要处理的流量，还可根据七层的 URL、浏览器类别、语言来决定是否要进行负载均衡。举个例子，如果你的 Web 服务器分成两组，一组是中文语言的，一组是英文语言的，那么七层负载均衡就可以当用户来访问你的域名时，自动辨别用户语言，然后选择对应的语言服务器组进行负载均衡处理。

1.2、在 Kubernetes 中为什么要做负载均衡？

- 1、Pod 漂移问题

Kubernetes 具有强大的副本控制能力，能保证在任意副本（Pod）挂掉时自动从其他机器启动一个新的，还可以动态扩容等，通俗地说，这个 Pod 可能在任何时刻出现在任何节点上，也可能在任何时刻死在任何节点上。那么自然随着

Pod 的创建和销毁，Pod IP 肯定会动态变化。

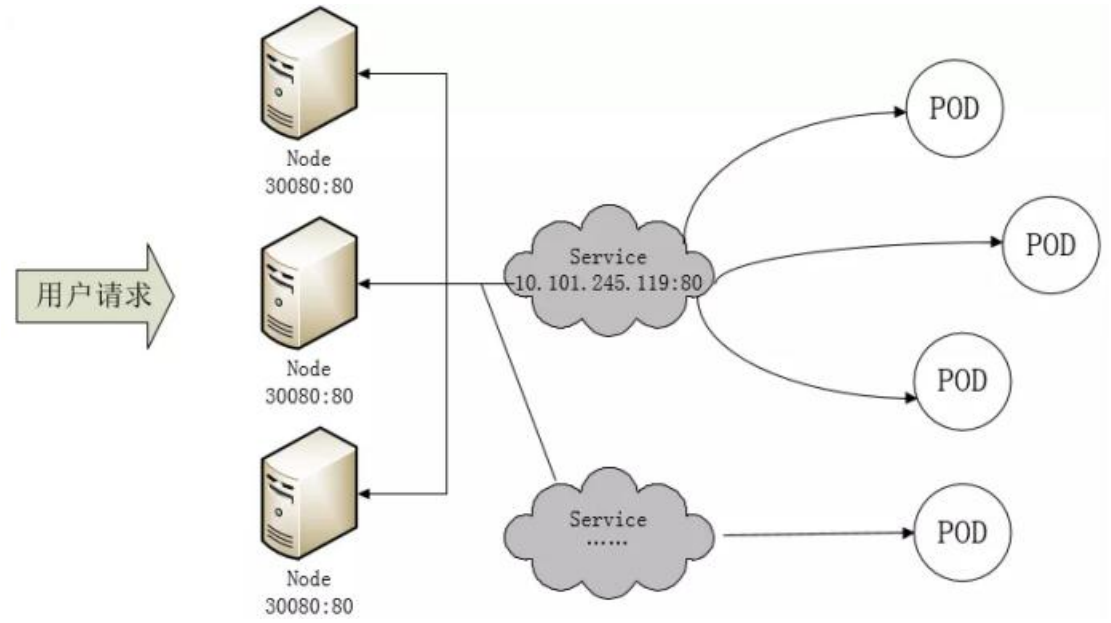
那么如何把这个动态的 Pod IP 暴露出去？

这里借助于 Kubernetes 的 Service 机制，Service 可以以标签的形式选定一组带有指定标签的 Pod，并监控和自动负载他们的 Pod IP，那么我们向外暴露只暴露 Service IP 就行了。

这就是 NodePort 模式：即在每个节点上开起一个端口，然后转发到内部 Pod IP 上，如下图所示：

此时的访问方式：http://nodeip:nodeport/，即数据包流向如下：

客户端请求 -> node 节点的 ip:端口 -> service 的 ip:端口 -> pod 的 ip:端口



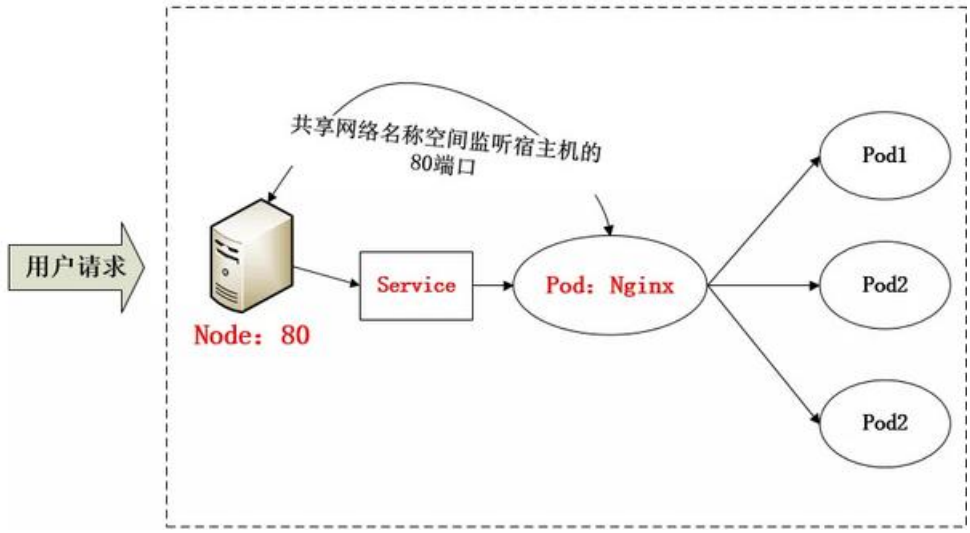
- 2、端口管理问题

采用 NodePort 方式暴露服务面临的问题是，服务一旦多起来，NodePort 在每个节点上开启的端口会及其庞大，而且难以维护。

这时，我们能否使用一个 Nginx 直接对内进行转发呢？众所周知的是，Pod 与 Pod 之间是可以互相通信的，而 Pod 是可以共享宿主机的网络名称空间的，也就是说当在共享网络名称空间时，Pod 上所监听的就是 Node 上的端口。那么这又该如何实现呢？

简单的实现就是使用 DaemonSet 在每个 Node 上监听 80，然后写好规则，因为 Nginx 外面绑定了宿主机 80 端口（就像 NodePort），本身又在集群内，那么向后直接转发到相应 Service IP 就行了，如下图所示：

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。



- 3、域名分配及动态更新问题

从上面的方法，采用 Nginx-Pod 似乎已经解决了问题，但是其实这里面有一个很大缺陷：当每次有新服务加入又该如何修改 Nginx 配置呢？

我们知道使用 Nginx 可以通过虚拟主机域名进行区分不同的服务，而每个服务通过 upstream 进行定义不同的负载均衡池，再加上 location 进行负载均衡的反向代理，在日常使用中只需要修改 nginx.conf 即可实现，那在 K8S 中又该如何实现这种方式的调度呢？

假设后端的服务初始服务只有 ecshop，后面增加了 bbs 和 member 服务，那么又该如何将这 2 个服务加入到 Nginx-Pod 进行调度呢？总不能每次手动改或者 Rolling Update 前端 Nginx Pod 吧。此时 Ingress 出现了，如果不算上面的 Nginx，Ingress 包含两大组件：Ingress Controller 和 Ingress。

1.3、Ingress 和 Ingress Controller

1.3.1、Ingress 介绍

在 Kubernetes 中，Ingress 是一种 API 对象，它充当了将外部网络流量路由到 Kubernetes 集群内部服务的入口。它是一个规范化的流量管理方式，可以方便地进行配置和管理。

Ingress 支持扩展的路由规则，包括基于主机、路径、HTTP 方法和其他 Web 请求流量的路由控制。这使得 Ingress 成为将 Kubernetes 作为 Web 应用程序托管解决方案时的一个理想选择。

在 Kubernetes 中，使用不同的 Ingress Controller 来实现 Ingress 规范。实际上，Ingress Controller 是一种 Kubernetes 部署，它通过 Ingress API 对象接收流量，实现请求路由和负载平衡等功能。Nginx Ingress Controller、Traefik Ingress Controller、HAProxy Ingress Controller 等都常用于

Kubernetes 中。

总之，Kubernetes Ingress 是一种定义配置和路由网络流量的规范，而 Ingress Controller 是用于实现 Ingress 规范的 Kubernetes 部署。

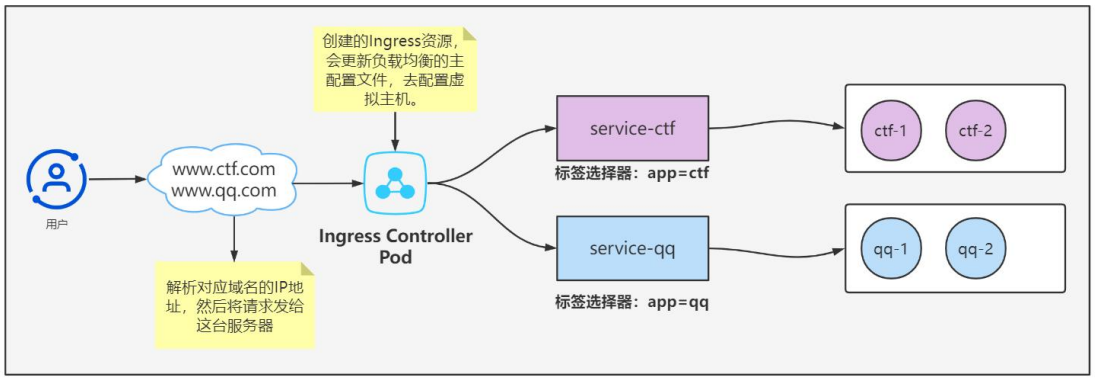
1.3.2、Ingress Controller 介绍

在 Kubernetes 中，Ingress Controller 是一种特殊的 Kubernetes 部署，它通过实现 Ingress 规范来将外部的网络流量路由到 Kubernetes 集群内部的服务。Ingress Controller 通常是作为 Pod 来运行的，它们使用 Ingress 规范中定义的路由规则和负载平衡算法来将请求路由到正确的服务实例中。

Kubernetes 本身并没有提供 Ingress Controller 的实现，而是通过官方提供的 API 规范来定义 Ingress 对象。实际上，Ingress 规范仅定义了路由规则，而不涉及任何负载平衡、SSL 终止等方面的细节。因此，为了实现 Ingress 规范，需要使用不同的 Ingress Controller 来构建完整的网络流量路由和负载平衡方案。

常用的 Ingress Controller 有 Nginx Ingress Controller、Traefik Ingress Controller、HAProxy Ingress Controller 等等。这些 Ingress Controller 除了实现 Ingress 规范，还提供了一些扩展功能，例如基于策略的负载均衡、SSL 卸载以及基于 HTTP 重定向的路由等功能。

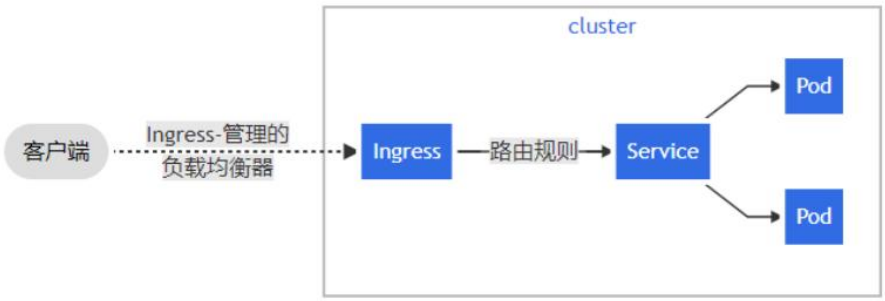
Ingress Controller 这东西就是解决“Nginx 的处理方式”的。Ingress Controller 通过与 Kubernetes API 交互，动态的去感知集群中 Ingress 规则变化，然后读取他，按照他自己模板生成一段 Nginx 配置，再写到 Nginx Pod 里，最后 reload 一下，工作流程如下图：



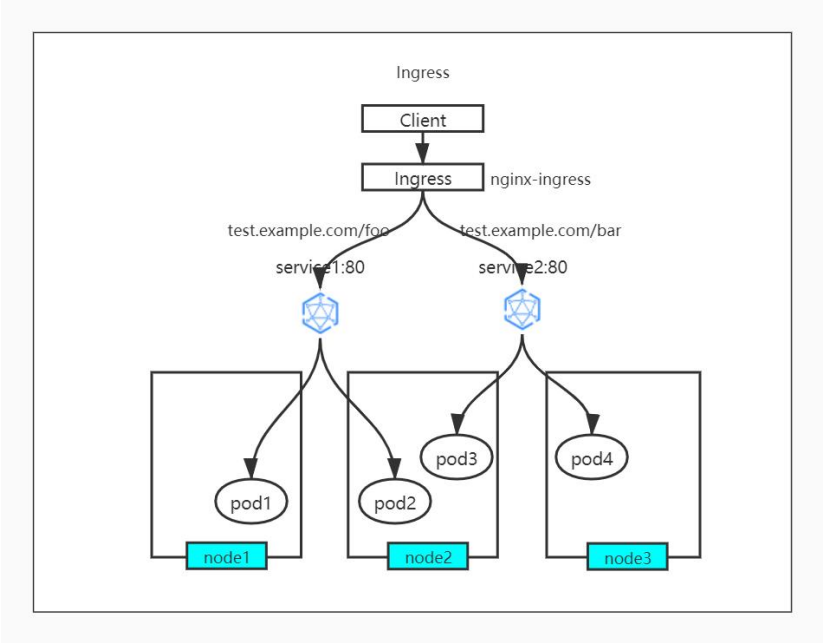
实际上 Ingress 也是 Kubernetes API 的标准资源类型之一，它其实就是一组基于 DNS 名称（host）或 URL 路径把请求转发到指定的 Service 资源的规则。用于将集群外部的请求流量转发到集群内部完成的服务发布。我们需要明白的是，Ingress 资源自身不能进行“流量穿透”，仅仅是一组规则的集合，这些集合规则还需要其他功能的辅助，比如监听某套接字，然后根据这些规则的匹配进行路

由转发，这些能够为 Ingress 资源监听套接字并将流量转发的组件就是 Ingress Controller。

图一：

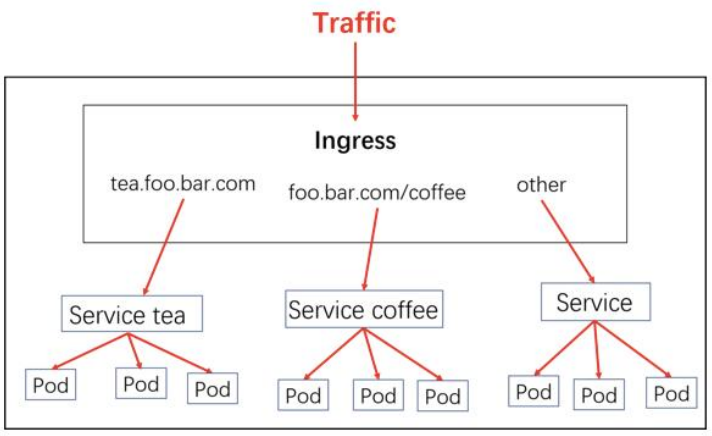


图二：



图三：

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。



总之，Kubernetes Ingress Controller 是为实现 Ingress 规范而设计的 Kubernetes 部署，它通过定义路由规则来将外部的网络流量路由到 Kubernetes 服务之中，并提供负载均衡、SSL 卸载等额外功能。

1.4、使用 Ingress Controller 代理 k8s 内部应用的流程

使用 Ingress Controller 代理 Kubernetes 内部应用的流程大致如下：

1、安装 Ingress Controller：为了将外部流量路由到内部 Kubernetes 应用程序，需要安装并配置一种 Ingress Controller，例如 Nginx Ingress Controller、Traefik Ingress Controller 或 HAProxy Ingress Controller。通常情况下，可以使用 Helm Charts 或者官方提供的 YAML 文件来安装 Ingress Controller。

2、创建一个新的 Kubernetes Service：Ingress Controller 需要知道要将流量路由到哪个 Service 上，因此首先需要创建一个新的 Service 对象，可以使用 Deployment 或 Pod 来创建后端服务，或使用 ServiceType: ClusterIP 来创建一个单独的 Service 对象。

3、创建 Ingress 对象并定义路由规则：创建一个新的 Ingress 对象，并定义流量路由规则。路由规则可以基于主机、路径、HTTP 方法和其他 Web 请求流量进行控制，并且可以在不同服务间进行负载均衡。

4、检查 Ingress 资源是否正常运行：使用 kubectl 命令检查新创建的 Ingress 资源的状态，确保它被成功地应用到 Kubernetes 集群中，并且可以正常处理流量。

在完成这些步骤以后，就可以使用 Ingress Controller 将外部流量路由到 Kubernetes 内部应用程序了。需要注意的是，Ingress Controller 代理内部应用程序的方式会根据不同的 Ingress Controller 而异，具体取决于实际的配置。

1.5、Ingress 应用场景

Kubernetes Ingress 是一种路由网络流量的规范，它提供了在 Kubernetes 中将外部流量路由到集群内部服务的标准化方式。下面是一些 Kubernetes Ingress 的应用场景：

1、Web 应用程序：对于基于 Web 的应用程序，可以使用 Kubernetes Ingress 将不同的路径和主机名路由到不同的服务，实现灵活的请求路由和负载均衡。

2、API Gateway：在微服务架构中，API Gateway 是将多个微服务后端聚合到一个入口点的一种常见模式。使用 Kubernetes Ingress 可以很容易地实现 API Gateway 模式，并提供灵活的请求路由、负载均衡和 API 版本控制等功能。

3、SSL/TLS 终止点：Kubernetes Ingress 可以在请求进入集群内部之前进行 SSL/TLS 协议的终止。这可以让应用程序仅需要处理普通 HTTP 请求，而由 Ingress Controller 处理加密和解密的过程。

4、应用程序流量控制：可以通过 Kubernetes Ingress 设置流量规则和限制，对外部请求进行限制，减少恶意攻击并保护应用程序免受 DDoS 攻击。

总之，Kubernetes Ingress 可以在不同的应用场景中提供灵活的请求路由、负载均衡和访问控制等功能，是一种强大的网络流量路由解决方案。

2、安装 Nginx Ingress Controller

小节目标：安装 k8s 所适应的 ingress，并实现 ingress 高可用。

- ingress-controller 官方网址
  https://github.com/kubernetes/ingress-nginx/

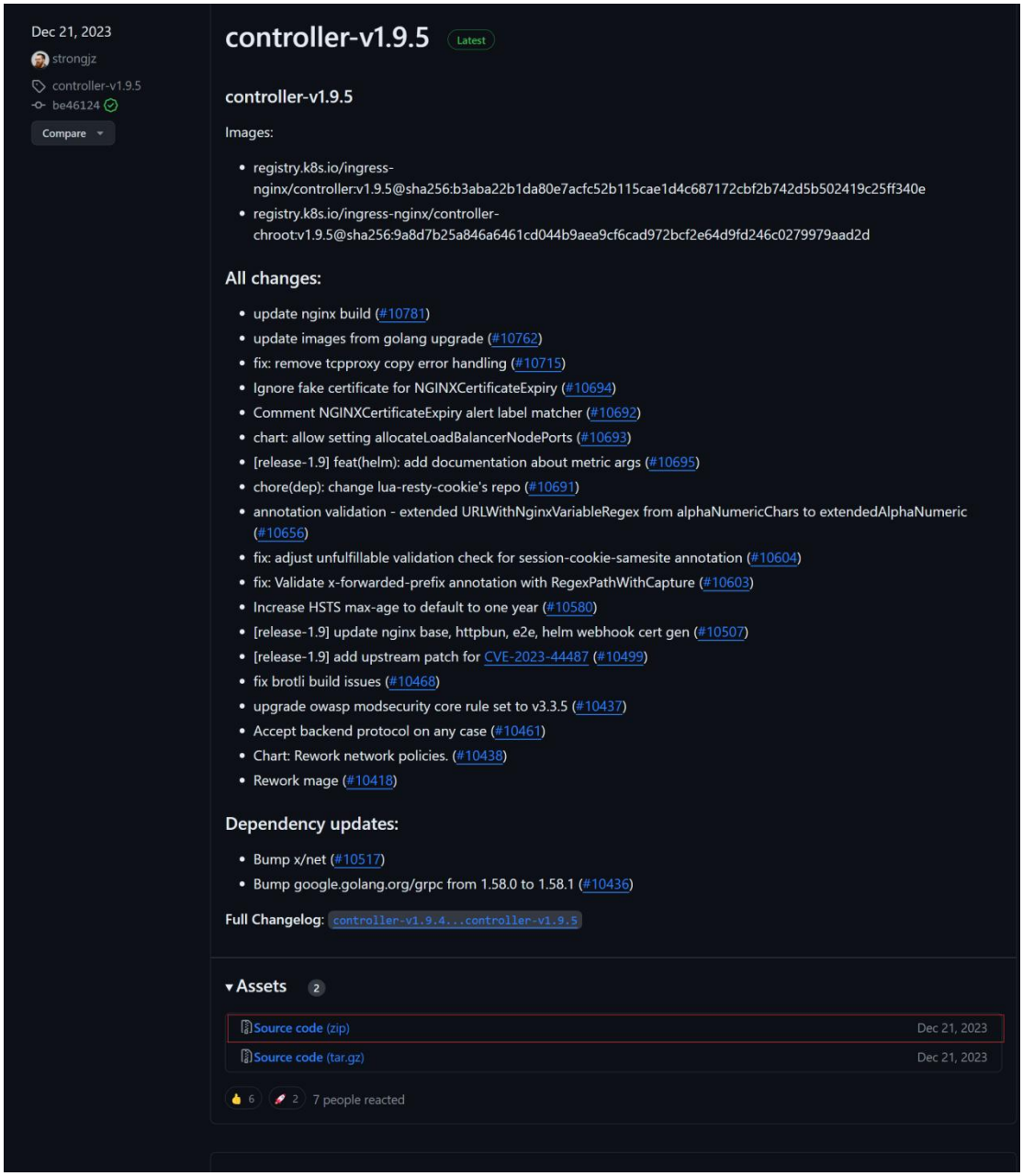
- Ingress-controller 和 k8s 版本对应关系

Ingress-controller 和 k8s 版本对照：“下图出自官网”

Supported	Ingress-NGINX version	k8s supported version	Alpine Version	Nginx Version	Helm Chart Version
END	v1.9.5	1.28, 1.27, 1.26, 1.25	3.18.4	1.21.6	4.9.0*
END	v1.9.4	1.28, 1.27, 1.26, 1.25	3.18.4	1.21.6	4.8.3
END	v1.9.3	1.28, 1.27, 1.26, 1.25	3.18.4	1.21.6	4.8.*
END	v1.9.1	1.28, 1.27, 1.26, 1.25	3.18.4	1.21.6	4.8.*
END	v1.9.0	1.28, 1.27, 1.26, 1.25	3.18.2	1.21.6	4.8.*
END	v1.8.4	1.27, 1.26, 1.25, 1.24	3.18.2	1.21.6	4.7.*
END	v1.8.2	1.27, 1.26, 1.25, 1.24	3.18.2	1.21.6	4.7.*
END	v1.8.1	1.27, 1.26, 1.25, 1.24	3.18.2	1.21.6	4.7.*
END	v1.8.0	1.27, 1.26, 1.25, 1.24	3.18.0	1.21.6	4.7.*
END	v1.7.1	1.27, 1.26, 1.25, 1.24	3.17.2	1.21.6	4.6.*

这里我们需要选择我们 k8s 集群支持的 ingress 版本。

● 下载 ingress 源码包，我们的 k8s 集群是 1.28.1 版本，这里我们安装 v1.9.5 版本的 ingress



● 安装 nginx ingress controller v1.7.1 版本

（1）上传、解压、拷贝软件包文件

1、将下载的压缩包文件上传至 k8s 任意 master 节点上

```
[root@k8s-master01 ~]# ll ingress-nginx-controller-v1.9.5.zip
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

```
[root@k8s-master01 ~]# ll ingress-nginx-controller-v1.9.5.zip
-rw-r--r-- 1 root root 4529729 Jan 21 19:01 ingress-nginx-controller-v1.9.5.zip
[root@k8s-master01 ~]#
```

2、解压

```
[root@k8s-master01 ~]# unzip ingress-nginx-controller-v1.9.5.zip
```

3、将部署 ingress-controller 需要的 deploy.yaml 文件拷贝出来

```
[root@k8s-master01 ~]# cp ingress-nginx-controller-v1.9.5/deploy/static/provider/cloud/deploy.yaml ingress-nginx-controller-v1.9.5.yaml
```

提示：这个 yaml 文件就是部署 ingress controller 需要的资源清单文件。

（2）修改 ingress-nginx-controller-v1.9.5.yaml 文件

```
[root@k8s-master01 ~]# vi ingress-nginx-controller-v1.9.5.yaml
# 1、将 Deployment 资源类型修改为 DaemonSet 资源
---
apiVersion: apps/v1
kind: DaemonSet
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: 1.9.5
  name: ingress-nginx-controller
  namespace: ingress-nginx
spec:
  minReadySeconds: 0
  revisionHistoryLimit: 10
  selector:

# 2、添加 hostNetwork: true 仅主机暴露配置
```

将deployment控制器换成daeomset控制器

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

```
---
apiVersion: apps/v1
kind: DaemonSet
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: 1.9.5
  name: ingress-nginx-controller
  namespace: ingress-nginx
spec:
  minReadySeconds: 0
  revisionHistoryLimit: 10
  selector:
    matchLabels:
      app.kubernetes.io/component: controller
      app.kubernetes.io/instance: ingress-nginx
      app.kubernetes.io/name: ingress-nginx
  strategy:
    rollingUpdate:
      maxUnavailable: 1
    type: RollingUpdate
  template:
    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: 1.9.5
    spec:
      hostNetwork: true
      containers:
      - args:
        - /nginx-ingress-controller
```

添加仅主机暴露端口

3、默认的镜像下载不下来，这里修改成阿里云的地址

[root@k8s-master01 ~]# cat ingress-nginx-controller-v1.9.5.yaml | grep image:

```
[root@k8s-master01 ~]# cat ingress-nginx-controller-v1.9.5.yaml | grep image:
        image: registry.k8s.io/ingress-nginx/controller:v1.9.5@sha256:b3aba22b1da80e7acfc52b115cae1d4c687172cbf2b742d5b502419c25ff340e
        image: registry.k8s.io/ingress-nginx/kube-webhook-certgen:v20231011-8b53cabe0@sha256:a7943503b45d552785aa3b5e457f169a5661fb94d82b8a3373bcd9ebaf9aac80
        image: registry.k8s.io/ingress-nginx/kube-webhook-certgen:v20231011-8b53cabe0@sha256:a7943503b45d552785aa3b5e457f169a5661fb94d82b8a3373bcd9ebaf9aac80
[root@k8s-master01 ~]#
```

[root@k8s-master01 ~]# vi ingress-nginx-controller-v1.9.5.yaml

将 controller 镜像替换为：

registry.cn-hangzhou.aliyuncs.com/google_containers/nginx-ingress-controller:v1.9.5

将 kube-webhook-certgen 镜像替换为：

registry.cn-hangzhou.aliyuncs.com/google_containers/kube-webhook-certgen:v20231011-8b53cabe0

[root@k8s-master01 ~]# cat ingress-nginx-controller-v1.9.5.yaml | grep image:

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

```
[root@k8s-master01 ~]# cat ingress-nginx-controller-v1.9.5.yaml | grep image:
        image: registry.cn-hangzhou.aliyuncs.com/google_containers/nginx-ingress-controller:v1.9.5
        image: registry.cn-hangzhou.aliyuncs.com/google_containers/kube-webhook-certgen:v20231011-8b53cabe0
        image: registry.cn-hangzhou.aliyuncs.com/google_containers/kube-webhook-certgen:v20231011-8b53cabe0
[root@k8s-master01 ~]#
```

4、修改 ingress-nginx-controller 添加 Node 硬亲和性

下面是默认的，做的是 node 标签选择器，将其禁用

```
        volumeMounts:
        - mountPath: /usr/local/certificates/
          name: webhook-cert
          readOnly: true
      dnsPolicy: ClusterFirst
#      nodeSelector:
#        kubernetes.io/os: linux
      serviceAccountName: ingress-nginx
      terminationGracePeriodSeconds: 300
      volumes:
      - name: webhook-cert
        secret:
          secretName: ingress-nginx-admission
---
apiVersion: batch/v1
kind: Job
metadata:
```

添加 node 硬亲和性：

```
      dnsPolicy: ClusterFirst
#      nodeSelector:
#        kubernetes.io/os: linux
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
            - matchExpressions:
              - key: kubernetes.io/ingress
                operator: In
                values:
                - nginx
      serviceAccountName: ingress-nginx
      terminationGracePeriodSeconds: 300
      volumes:
      - name: webhook-cert
        secret:
          secretName: ingress-nginx-admission
---
apiVersion: batch/v1
kind: Job
metadata:
```

```
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
            - matchExpressions:
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

```
                - key: kubernetes.io/ingress
                  operator: In
                  values:
                  - nginx

# 修改 daemonset 滚动更新策略
```

```
---
apiVersion: apps/v1
kind: DaemonSet
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: 1.9.5
  name: ingress-nginx-controller
  namespace: ingress-nginx
spec:
  minReadySeconds: 0
  revisionHistoryLimit: 10
  selector:
    matchLabels:
      app.kubernetes.io/component: controller
      app.kubernetes.io/instance: ingress-nginx
      app.kubernetes.io/name: ingress-nginx
  updateStrategy:
    rollingUpdate:
      maxUnavailable: 1
    type: RollingUpdate
  template:
    metadata:
      labels:
        app.kubernetes.io/component: controller
```

修改更新字段为daemonset支持的字段

至此修改完毕！

（3）给节点打标签，会将启动的 ingress controller pod 调度到打标签的节点上。

```
[root@k8s-master01 ~]# kubectl label node k8s-node01 kubernetes.io/ingress=nginx
node/k8s-node01 labeled
[root@k8s-master01 ~]# kubectl label node k8s-node02 kubernetes.io/ingress=nginx
node/k8s-node02 labeled
```

（4）创建 ingress-nginx-controller

```
[root@k8s-master01 ~]# kubectl apply -f ingress-nginx-controller-v1.7.1.yaml
```

（5）查看 pod

```
[root@k8s-master01 ~]# kubectl get pods -n ingress-nginx
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

```
[root@k8s-master01 ~]# kubectl get pods -n ingress-nginx
NAME                                   READY   STATUS      RESTARTS   AGE
ingress-nginx-admission-create-ll8rq   0/1     Completed   0          117s
ingress-nginx-admission-patch-xpp5l    0/1     Completed   2          117s
ingress-nginx-controller-9xfm7         1/1     Running     0          117s
ingress-nginx-controller-w4gkv         1/1     Running     0          30s
[root@k8s-master01 ~]#
```

6、用 curl 访问 Node 的 80 端口号，验证 nginx-ingress-controller 服务是否正常工作：

```
[root@k8s-master01 ~]# curl k8s-node01
<html>
<head><title>404 Not Found</title></head>
<body>
<center><h1>404 Not Found</h1></center>
<hr><center>nginx</center>
</body>
</html>

[root@k8s-master01 ~]# curl k8s-node02
<html>
<head><title>404 Not Found</title></head>
<body>
<center><h1>404 Not Found</h1></center>
<hr><center>nginx</center>
</body>
</html>
```

访问请求成功，则表示安装成功，已经正常工作！

3、Ingress 资源清单文件编写

3.1、Ingress 资源清单文件解读

Ingress 资源可以通过如下命令查看相关语法：

```
[root@k8s-master01 ~]# kubectl explain Ingress
```

- Ingress 资源说明

属性名称	取值类型	取值说明
apiVersion	string	Api 版本。通常为 networking.k8s.io/v1
kind	string	资源类型。通常为 Ingress
metadata	Object	元数据
metadata.name	String	控制器的名称

metadata.namespace	String	控制器所属的命名空间，默认值为 default
metadata.labels[]	List	自定义标签列表
metadata.annotation[]	List	自定义注解列表
spec	Object	规范 Ingress 所需行为的规范
spec.defaultBackend	Object	defaultBackend 是 Ingress 规范中的一个参数，用于指定没有匹配到任何规则的请求该发送到哪个后端服务。该参数是可选的，如果没有设置，则请求会被丢弃，或者返回一个指定的 HTTP 响应代码，例如 404。 这个后端服务可以是一个已经存在的 Service 或 Deployment，也可以是自定义的资源对象。只能指定一个。
spec.defaultBackend.resource	Object	指定后端服务为 Deployment 等，也可以是自定义的资源对象。
spec.defaultBackend.resource.apiGroup	string	指定对应资源的 apiGroup
spec.defaultBackend.resource.kind	string	指定资源的类型
spec.defaultBackend.resource..name	string	指定对应资源的名称
spec.defaultBackend.service	Object	指定后端服务为 Service。
spec.defaultBackend.service.name	string	指定后端服务 Service 的名称
spec.defaultBackend.service.port	Object	指定后端服务 Service 的端口
spec.ingressClassName	string	在 Kubernetes Ingress 规范中，ingressClassName 参数是一个可选项，用于指定 Ingress 控制器应该使用哪个类别或类型来处理该规范。该参数与 Kubernetes 平台定义的 Ingress 类型分离，它允许多个 Ingress 控制器在同一个集群中共存并彼此独立地处理网络流量。 如：我们使用的 Nginx Ingress Controller，就指定 value 为 nginx。
spec.rules	[]Object	spec.rules 参数是一个列表，用于定义请求的入口规则。每个规则对应一个 Ingress 请求的入口点，定义了该规则的转发行为。根据规则定义的主机名和 URL 路径，将请求转发到相应的后端服务或工作负载。
spec.rules.host	string	spec.rules.host 参数指定了一个主机名，该规范将应用于该主机名匹配的所有 Ingress 请求。具体而言，当该参数与请求中的主机名匹配（通常是客户端请求的主机名或 DNS 名称）时，请求将被转发到该规范所定义的后端服务或工作负载中。
spec.rules.http	Object	http 参数定义了使用 HTTP 协议的入口规则。每个 http

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

		对象定义一个或多个 URL 路径，以及将路径请求转发到的后端服务或工作负载。同时，可以在 http 对象中配置多个端口和主机名。
spec.rules.http.paths	[]Object	spec.rules.http.paths 参数定义了 URL 路径，并指定了这些路径应该如何转发到相应的后端服务或工作负载。
spec.rules.http.paths.backend	Object	在 Kubernetes Ingress 规范中，backend 参数用于定义将 Ingress 请求转发到的后端服务或工作负载。 这个后端服务可以是一个已经存在的 Service 或 Deployment，也可以是自定义的资源对象。只能指定一个。
spec.rules.http.paths.backend.resource	Object	指定后端服务为 Deployment 等，也可以是自定义的资源对象。
spec.rules.http.paths.backend.service	Object	指定后端服务为 Service。
spec.rules.http.paths.path	string	在 Kubernetes Ingress 规范中，path 参数定义了 Ingress 请求的 URL 路径匹配规则。这个参数的值是一个字符串，用于精确匹配或前缀匹配 URL 路径。需要注意的是，路径必须以/开头。
spec.rules.http.paths.pathType	string	在 Kubernetes Ingress 规范中，pathType 参数用于定义 Path 的匹配类型。它可以是 Prefix 或 Exact 两种类型之一。 ● Prefix：前缀匹配。当请求的路径以 Path 的值开头时，将进行匹配。例如，如果将 Path 定义为/example，则路径/example/foo 和/example/bar 都将匹配。 ● Exact：完全匹配。当请求的路径完全与 Path 的值相同时，才会进行匹配。例如，如果将 Path 定义为/example，则路径/example 才会匹配。
spec.tls		tls 参数用于配置 Ingress 中用于 TLS 通信的证书和私钥。 spec.tls 参数主要包含以下参数： ● hosts：一个字符串数组，列出将使用此证书的主机名。必填项。 ● secretName：包含 TLS 密钥的 Kubernetes Secret 的名称。Secret 必须包括 tls.crt 和 tls.key 字段。必须与证书匹配的 tLS 证书密钥。必填项。
spec.tls.hosts	[]string	
spec.tls.secretName	string	

3.2、Ingress PathType 匹配类型

ingress.spec.rules.http.paths.pathType 是 ingress 资源类型中很重要的概念，下面详细解读一下。

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

Ingress 中的每个路径都需要有对应的路径类型（Path Type）。未明确设置 pathType 的路径无法通过合法性检查。当前支持的路径类型有三种：

- Prefix：前缀匹配。当请求的路径以 Path 的值开头时，将进行匹配。例如，如果将 Path 定义为/example，则路径/example/foo 和/example/bar 都将匹配。
- Exact：完全匹配。当请求的路径完全与 Path 的值相同时，才会进行匹配。例如，如果将 Path 定义为/example，则路径/example 才会匹配。
- ImplementationSpecific：对于这种路径类型，匹配方法取决于 IngressClass。具体实现可以将其作为单独的 pathType 处理或者与 Prefix 或 Exact 类型作相同处理。

下图定义了 Prefix 和 Exact 路径和请求路径是否匹配：

类型	路径	请求路径	匹配与否?
Prefix	/	(所有路径)	是
Exact	/foo	/foo	是
Exact	/foo	/bar	否
Exact	/foo	/foo/	否
Exact	/foo/	/foo	否
Prefix	/foo	/foo , /foo/	是
Prefix	/foo/	/foo , /foo/	是
Prefix	/aaa/bb	/aaa/bbb	否
Prefix	/aaa/bbb	/aaa/bbb	是
Prefix	/aaa/bbb/	/aaa/bbb	是，忽略尾部斜线
Prefix	/aaa/bbb	/aaa/bbb/	是，匹配尾部斜线

3.3、Ingress 的策略配置技巧

为了实现灵活的负载分发策略，Ingress 策略可以按多种方式进行配置，下面对几种常见的 Ingress 转发策略进行说明。

1、转发到单个后端服务上

基于这种设置，客户端到 Ingress Controller 的访问请求都将被转发到后端的唯一 Service 上，在这种情况下 Ingress 无须定义任何 rule。

通过如下所示的设置，对 Ingress Controller 的访问请求都将被转发到“tomcat:8080”这个服务上。

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ingress-myapp
  namespace: default
spec:
  ingressClassName: nginx
  rules:
  - host: tomcat.zyf.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
           name: tomcat
           port:
            number: 8080
```

2、同一域名下，不同的 URL 路径被转发到不同的服务上

这种配置常用于一个网站通过不同的路径提供不同的服务的场景，例如/web 表示访问 Web 页面，/api 表示访问 API 接口，对应到后端的两个服务，通过 Ingress 的设置很容易就能将基于 URL 路径的转发规则定义出来。

通过如下所示的设置，对“tomcat.zyf.com/web”的访问请求将被转发到“web-service:80”服务上；对“tomcat.zyf.com/api”的访问请求将被转发到“api-service:80”服务上：

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ingress-myapp
  namespace: default
spec:
  ingressClassName: nginx
  rules:
  - host: tomcat.zyf.com
    http:
      paths:
      - path: /web
        pathType: Prefix
        backend:
          service:
           name: web-service
```

```
              port:
               number: 80
        - path: /api
          pathType: Prefix
          backend:
            service:
             name: api-service
             port:
              number: 8081
```

3、不同的域名（虚拟主机名）被转发到不同的服务上

这种配置常用于一个网站通过不同的域名或虚拟主机名提供不同服务的场景，例如 nginx.web1.com 域名由 service1 提供服务，nginx.web2.com 域名由 service2 提供服务。

通过如下所示的设置，对“nginx.web1.com”的访问请求将被转发到“service1:80”服务上，对“nginx.web2.com”的访问请求将被转发到“service2:80”服务上：

```
  apiVersion: networking.k8s.io/v1
  kind: Ingress
  metadata:
    name: ingress-myapp
    namespace: default
  spec:
    ingressClassName: nginx
    rules:
    - host: nginx.web1.com
      http:
        paths:
        - path: /
          pathType: Prefix
          backend:
            service:
             name: service1
             port:
              number: 80
    - host: nginx.web2.com
      http:
        paths:
        - path: /
          pathType: Prefix
          backend:
            service:
```

```
              name: service2
              port:
               number: 80
```

4、主机名通配符

主机名可以是精确匹配(例如“foo.bar.com”)或者使用通配符来匹配（例如“*.foo.com”）。 精确匹配要求 HTTP host 头部字段与 host 字段值完全匹配。 通配符匹配则要求 HTTP host 头部字段与通配符规则中的后缀部分相同。

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	www.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

yaml 示例：

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ingress-myapp
  namespace: default
spec:
  ingressClassName: nginx
  rules:
  - host: "nginx.web1.com"
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
           name: service1
           port:
            number: 80
  - host: "*.web2.com"
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
           name: service2
           port:
            number: 80
```

4、Ingress 实战

4.1、Ingress HTTP 代理 tomcat

实验目标：

对 tomcat.zyf.com 网站的访问设置 Ingress 策略，定义对其/路径的访问转发到后端 tomcat Service 的规则。

（1）部署后端 tomcat web 服务

1、创建资源清单文件

```
[root@k8s-master01 ~]# vi myweb-demo.yaml
apiVersion: v1
kind: Service
metadata:
  name: tomcat
  namespace: default
spec:
  selector:
    app: tomcat
  ports:
  - name: http
    targetPort: 8080
    port: 8080
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: tomcat-deploy
  namespace: default
spec:
  replicas: 2
  selector:
    matchLabels:
      app: tomcat
  template:
    metadata:
      labels:
        app: tomcat
    spec:
      containers:
      - name: tomcat
```

```
            image: tomcat:8.5.34-jre8-alpine
            ports:
            - name: http
              containerPort: 8080

2、更新资源清单文件
    [root@k8s-master01 ~]# kubectl apply -f myweb-demo.yaml
    service/tomcat created
    deployment.apps/tomcat-deploy created

3、查看创建的资源
    [root@k8s-master01 ~]# kubectl get pods
    NAME                             READY   STATUS    RESTARTS   AGE
    tomcat-deploy-ccd6bf4c6-jfvfm    1/1     Running   0          83s
    tomcat-deploy-ccd6bf4c6-rh687    1/1     Running   0          83s

    [root@k8s-master01 ~]# kubectl get svc tomcat
    NAME     TYPE        CLUSTER-IP    EXTERNAL-IP   PORT(S)    AGE
    tomcat   ClusterIP   10.10.85.13   <none>        8080/TCP   2m11s
```

（2）配置 ingress 策略

```
1、创建资源清单文件
    [root@k8s-master01 ~]# cat tomcat-ingress.yaml
    apiVersion: networking.k8s.io/v1
    kind: Ingress
    metadata:
      name: ingress-myapp
      namespace: default
    spec:
      ingressClassName: nginx
      rules:
      - host: tomcat.zyf.com
        http:
          paths:
          - path: /
            pathType: Prefix
            backend:
              service:
               name: tomcat
               port:
                number: 8080
对上面的 yaml 文件说明：
```

这是一个 Kubernetes Ingress 的配置文件，它定义了一个 Ingress 资源对象，用于将一个域名（tomcat.zyf.com）映射到一个 Kubernetes Service 的端口（8080），以将流量路由到 Tomcat 应用程序上。

具体来说，这个配置文件指定了以下内容：

- metadata.name：Ingress 资源对象的名称。
- metadata.namespace：Ingress 资源对象所在的 Kubernetes Namespace。
- spec.ingressClassName：这个 Ingress 资源对象使用的 Ingress Controller 的名称，这里是 nginx。
- spec.rules.host：要映射的域名，这里是 tomcat.zyf.com。
- spec.rules.http.paths.path：要匹配的 URI 路径，这里是根路径/。
- spec.rules.http.paths.pathType：路径类型，这里是 Prefix，表示匹配以指定路径为前缀的所有请求。
- spec.rules.http.paths.backend.service.name：要路由到的 Kubernetes Service 的名称，这里是 tomcat。
- spec.rules.http.paths.backend.service.port.number：要路由到的 Kubernetes Service 暴露的端口号，这里是 8080。

2、创建资源清单文件

```
[root@k8s-master01 ~]# kubectl apply -f tomcat-ingress.yaml
ingress.networking.k8s.io/ingress-myapp created
```

3、查看创建的 ingress 资源

```
[root@k8s-master01 ~]# kubectl get ingress
NAME            CLASS   HOSTS            ADDRESS   PORTS   AGE
ingress-myapp   nginx   tomcat.zyf.com             80      11m
```

（3）测试访问

需要说明的是，客户端只能通过域名 tomcat.zyf.com 访问服务，这时要求客户端或者 DNS 将 tomcat.zyf.com 域名解析到后端多个 Node 的真实 IP 地址上。

linux 访问测试：

通过 curl 访问 tomcat.zyf.com 提供的服务（可以用--resolve 参数模拟 DNS 解析，目标地址为域名。也可以用-H’Host:tomcat.zyf.com’参数设置在 HTTP 头中要访问的域名，目标地址为 IP 地址），可以得到 myweb 服务返回的网页内容。

```
[root@k8s-master01 ~]# curl --resolve tomcat.zyf.com:80:192.168.128.21 http://tomcat.zyf.com
或
[root@k8s-master01 ~]# curl --resolve tomcat.zyf.com:80:192.168.128.22 http://tomcat.zyf.com
```

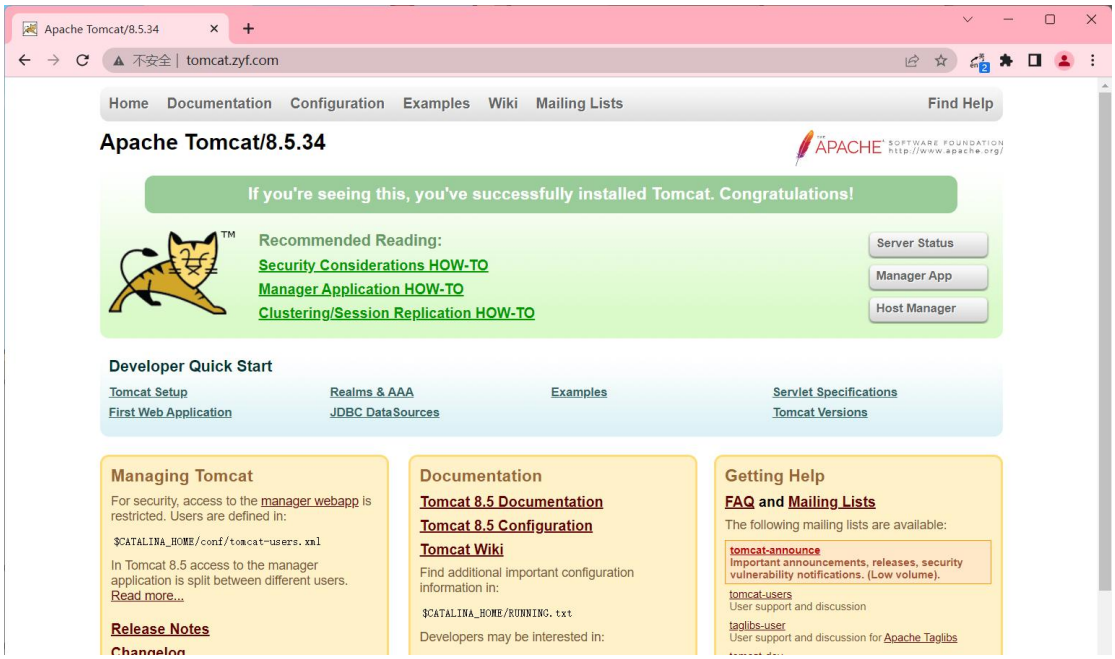
Windows 访问测试：

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

如果通过浏览器访问，那么需要先在本机上设置域名 tomcat.zyf.com 对应的 IP 地址，再到浏览器上进行访问。以 Windows 为例，修改 C:\Windows\System32\drivers\etc\hosts 文件，加入一行记录：

```
192.168.128.21 192.168.128.22 tomcat.zyf.com
```

浏览器输入“http://tomcat.zyf.com/”，访问结果如下：



4.2、Ingress HTTPS 代理 tomcat

4.2.1、构建 TLS 站点

创建 Secret 资源，Secret 资源中必须包括 tls.crt 和 tls.key 字段。

（1）准备证书，在 k8s-master01 节点操作

```
1、创建证书和私钥存放位置
    [root@k8s-master01 ~]# mkdir /test-ssl
    [root@k8s-master01 ~]# cd /test-ssl/

2、创建私钥
    [root@k8s-master01 test-ssl]# openssl genrsa -out tls.key 2048
    Generating RSA private key, 2048 bit long modulus
    ......................+++
    ......................+++
    e is 65537 (0x10001)

3、自签一张证书
    [root@k8s-master01 test-ssl]# openssl req -new -x509 -key tls.key -out tls.crt
```

```
-subj /C=CN/ST=Beijing/L=Beijing/O=DevOps/CN=tomcat.zyf.com
    [root@k8s-master01 test-ssl]# ll
    total 8
    -rw-r--r-- 1 root root 1285 Jun 13 23:47 tls.crt
    -rw-r--r-- 1 root root 1675 Jun 13 23:46 tls.key
```

（2）创建 secret，在 k8s-master01 节点操作

```
    [root@k8s-master01 test-ssl]# kubectl create secret tls tomcat-ingress-secret
--cert=tls.crt --key=tls.key
    secret/tomcat-ingress-secret created
```

（3）查看 secret

```
    [root@k8s-master01 ~]# kubectl get secret tomcat-ingress-secret
    NAME                    TYPE                DATA   AGE
    tomcat-ingress-secret   kubernetes.io/tls   2      26s

    [root@k8s-master01 ~]# kubectl describe secret tomcat-ingress-secret
    Name:         tomcat-ingress-secret
    Namespace:    default
    Labels:       <none>
    Annotations:  <none>

    Type:  kubernetes.io/tls

    Data
    ====
    tls.crt:  1285 bytes
    tls.key:  1675 bytes
```

4.2.2、创建 Ingress

在创建 ingress 之前，先清理 4.1 小节创建的 inress。
创建 ingress，指定刚才创建好的 secret。代理到后端的 tomcat svc。

（1）创建 ingress

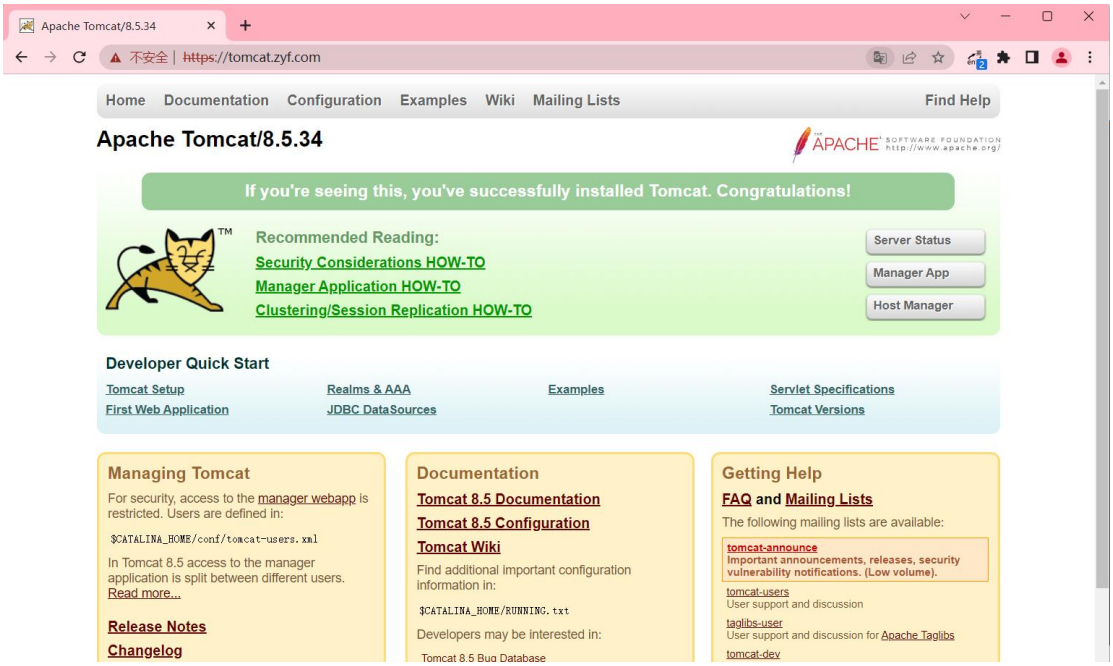
```
1、创建资源清单文件
    [root@k8s-master01 ~]# cat ingress-tomcat-tls.yaml
    apiVersion: networking.k8s.io/v1
    kind: Ingress
    metadata:
      name: ingress-tomcat-tls
      namespace: default
```

```
  spec:
    ingressClassName: nginx
    tls:
    - hosts:
      - tomcat.zyf.com
      secretName: tomcat-ingress-secret
    rules:
    - host: tomcat.zyf.com
      http:
        paths:
        - path: /
          pathType: Prefix
          backend:
            service:
             name: tomcat
             port:
              number: 8080

2、更新资源清单文件
  [root@k8s-master01 ~]# kubectl apply -f ingress-tomcat-tls.yaml
  ingress.networking.k8s.io/ingress-tomcat-tls created

3、查看 ingress
  [root@k8s-master01 ~]# kubectl get ingress ingress-tomcat-tls
  NAME                 CLASS   HOSTS             ADDRESS   PORTS     AGE
  ingress-tomcat-tls   nginx   tomcat.zyf.com              80, 443   43s
```

（2）客户端测试访问

浏览器访问 https://tomcat.zyf.com，选择接受风险并继续即可，出现如下：

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。



4.3、Ingress Nginx 添加 Basic Auth 安全认证

- 基本介绍

htpasswd 是 Apache 的 Web 服务器内置的工具，用于创建和更新储存用户名和用户基本认证的密码文件。

一般用于页面可以没有安全认证页面，来做简单安全验证保证服务安全。

- htpasswd 命令参数介绍

-c：创建一个新的密码文件
-b：在命令行中一并输入用户名和密码而不是根据提示输入密码
-D：删除指定的用户
-n：不更新密码文件,只将加密后的用户名密码输出到屏幕上
-p：不对密码进行加密,采用明文的方式
-m：采用 MD5 算法对密码进行加密（默认的加密方式）
-d：采用 CRYPT 算法对密码进行加密
-s：采用 SHA 算法对密码进行加密
-B：采用 bcrypt 算法对密码进行加密

- 配置实现

（1）部署后端 tomcat web 服务

```
[root@k8s-master01 ~]# vi myweb-demo.yaml
apiVersion: v1
kind: Service
metadata:
  name: tomcat
```

```
    namespace: default
  spec:
    selector:
      app: tomcat
    ports:
    - name: http
      targetPort: 8080
      port: 8080
  ---
  apiVersion: apps/v1
  kind: Deployment
  metadata:
    name: tomcat-deploy
    namespace: default
  spec:
    replicas: 2
    selector:
      matchLabels:
        app: tomcat
    template:
      metadata:
        labels:
          app: tomcat
      spec:
        containers:
        - name: tomcat
          image: tomcat:8.5.34-jre8-alpine
          ports:
          - name: http
            containerPort: 8080
```

（2）生成秘钥文件

使用 basic 有个小坑，在创建 secret 之前通过 htpasswd 工具生成的记录用户名密码的文件的文件名，必须叫 auth，不然最终访问的结果会是 503 错误。

```
1、安装 httpd-tools 服务
   [root@k8s-master01 ~]# yum -y install httpd-tools

2、生成密码文件，命令格式“命令 -c 密码文件 用户名”
   [root@k8s-master01 test]# htpasswd -c auth admin
   New password:
   Re-type new password:
   Adding password for user admin
```

```
3、查看生成好的秘钥
    [root@k8s-master01 test]# cat auth
    admin:$apr1$nUBvOqcj$qviQ/ryMxgfCqh4RyWOJY.
```

（3）创建 Secret 资源存储用户密码

语法：kubectl create secret generic <secretname> --from-file=<file>

```
    [root@k8s-master01 ~]# kubectl create secret generic auth --from-file=auth
    secret/auth created
```

（4）配置 ingress 认证

```
1、创建资源清单文件
    [root@k8s-master01 ~]# vi tomcat-ingress.yaml
    apiVersion: networking.k8s.io/v1
    kind: Ingress
    metadata:
      name: ingress-tomcat
      annotations:
        nginx.ingress.kubernetes.io/auth-type: basic
        nginx.ingress.kubernetes.io/auth-secret: auth
    spec:
      ingressClassName: nginx
      rules:
      - host: tomcat.zyf.com
        http:
          paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: tomcat
                port:
                  number: 8080
```

对上面的 yaml 文件说明：

这也是一个 Kubernetes Ingress 的配置文件，与先前的配置文件类似，但它具有额外的注释和注释配置。

具体来说，这个配置文件指定了以下新注释：

- nginx.ingress.kubernetes.io/auth-type：用于指定身份验证类型。这里设置为基本（basic）身份验证。
- nginx.ingress.kubernetes.io/auth-secret：用于指定包含用户名和密码信息的 Kubernetes Secret 的名称，以支持基本身份验证。

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

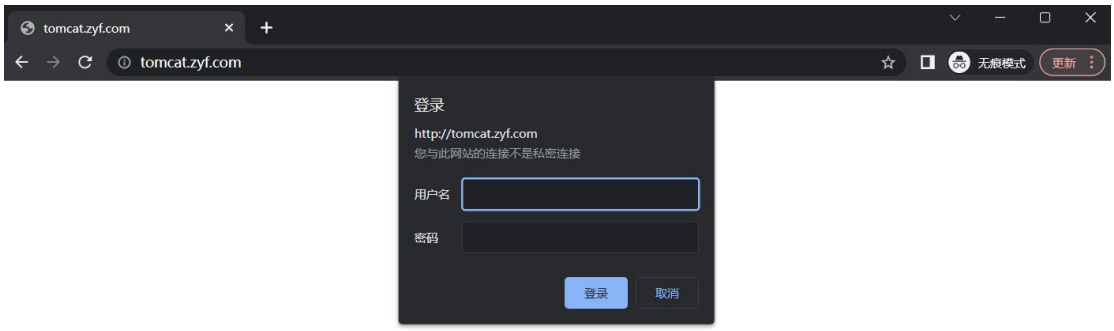
通过添加这些注释，Ingress Controller 将自动为主机名为 tomcat.zyf.com 上的所有请求提供基本身份验证。

2、更新资源清单文件

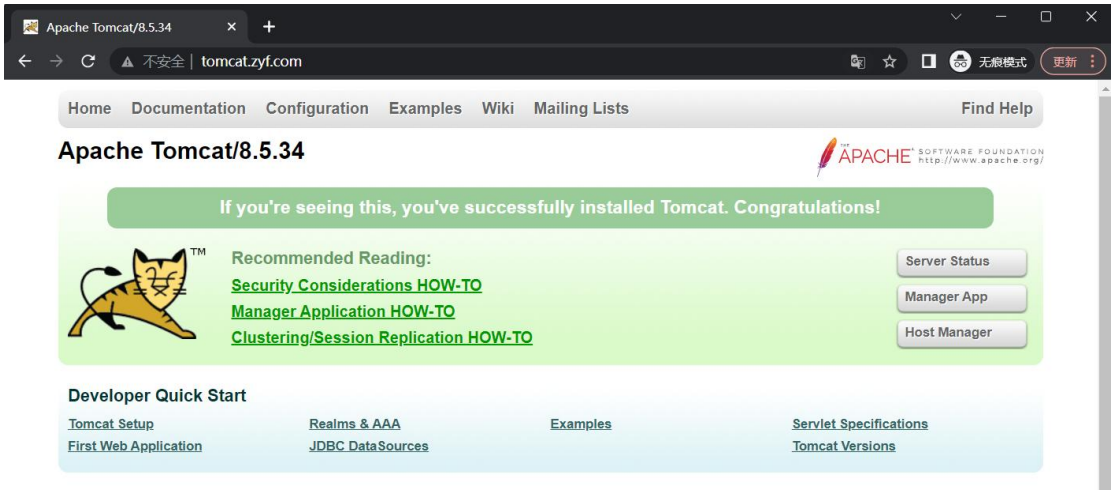
```
[root@k8s-master01 ~]# kubectl apply -f tomcat-ingress.yaml
ingress.networking.k8s.io/tomcat created
```

（5）浏览器访问测试

浏览器访问：tomcat.zyf.com



我的用户名密码都是 admin，输入用户名和密码，进行登录：



测试登录正常。

4.4、Ingress 灰度（金丝雀）发布

Nginx Annotations 的几种 Canary 规则：

- nginx.ingress.kubernetes.io/canary

  必须设置该 Annotation 值为 true，否则其它规则将不会生效。设置为 true 表示启用 canary 功能。设置为 false 表示不启用 canary 功能。

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

  nginx.ingress.kubernetes.io/canary-by-header 表示基于请求头的名称进行灰度发

布。请求头名称的特殊取值：always：无论什么情况下，流量均会进入灰度服务。never：无论什么情况下，流量均不会进入灰度服务。若没有指定请求头名称的值，则只要该头存在，都会进行流量转发。

- nginx.ingress.kubernetes.io/canary-by-header-value
  表示基于请求头的值进行灰度发布。需要与 canary-by-header 头配合使用。

- nginx.ingress.kubernetes.io/canary-weight
  表示基于权重进行灰度发布。取值范围：0~权重总值。若未设定总值，默认总值为 100。

- nginx.ingress.kubernetes.io/canary-weight-total
  表示设定的权重总值。若未设定总值，默认总值为 100。

4.4.1、基于服务权重的流量切分

- 基于客户端请求的流量切分场景

需求：

假设当前线上环境，您已经有一套服务 Service V1 对外提供 7 层服务，此时上线了一些新的特性，需要发布上线一个新的版本 Service V2。

希望将百分之 10 的客户端请求转发到 Service V2 服务中。

待运行一段时间稳定后，可将所有的流量从 Service V1 切换到 Service V2 服务中，再平滑地将 Service V1 服务下线。

图 1：现有一套服务 Service V1 正在对外提供服务。

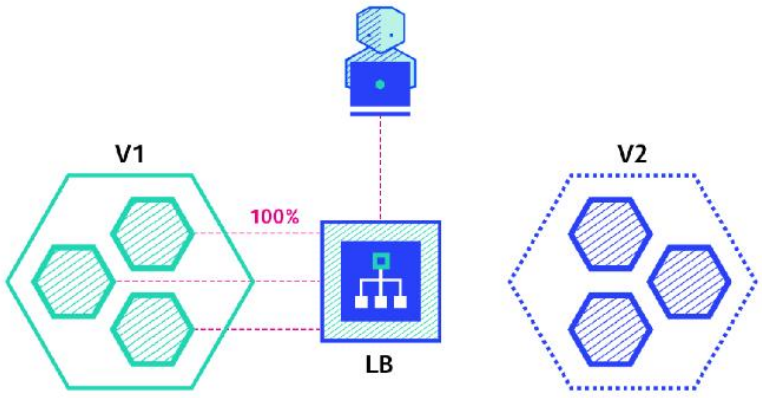


图 2：将百分之 10 的客户端请求转发到 Service V2 服务中。

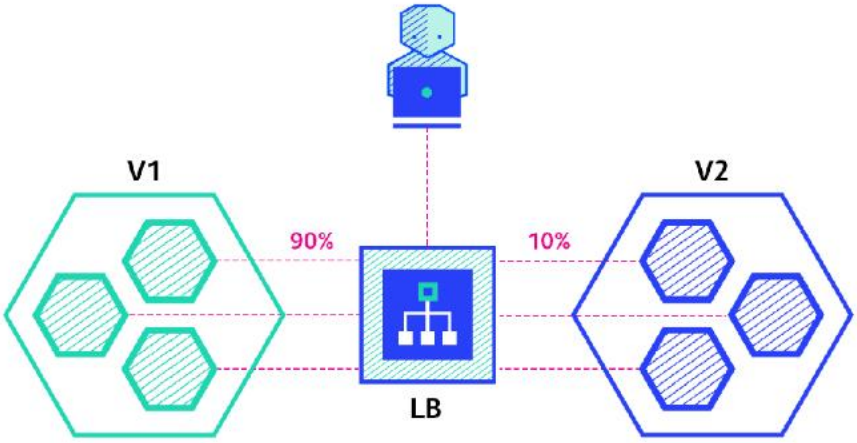
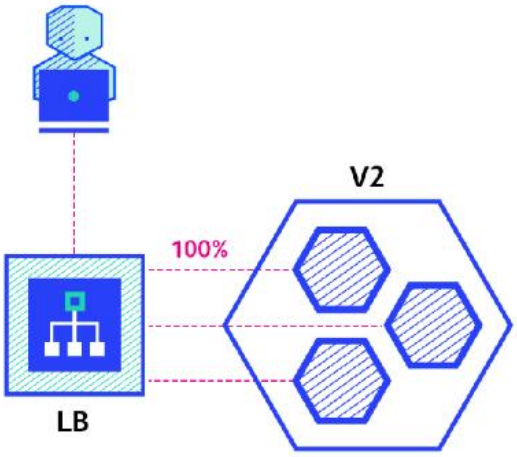


图 3：完成新旧版本迭代。



通过上面的 annotation 来实现灰度发布，其思路如下：

1、在集群中部署两套系统，一套是 v1 版本（old-nginx），一套是 v2 版本（new-nginx），两个版本都有自己的 service。

2、定义两个 ingress 配置，一个正常提供服务，一个增加 canary 的 annotation。

3、待 v2 版本无误后，将其切换成 v2 版本，并且将旧的版本下线，流量全部接入新的 v2 版本

（1）创建 old-nginx pod、svc、ingress

```
1、创建资源清单文件
    [root@k8s-master01 ~]# vi nginx-old.yaml
    ---
    apiVersion: networking.k8s.io/v1
    kind: Ingress
    metadata:
      name: ingress-old
      namespace: default
```

```
  spec:
    ingressClassName: nginx
    rules:
    - host: nginx.zyf.com
      http:
        paths:
        - path: /
          pathType: Prefix
          backend:
            service:
             name: old-nginx
             port:
              number: 80
  ---
  apiVersion: v1
  kind: Service
  metadata:
    name: old-nginx
    namespace: default
  spec:
    selector:
      app: old-nginx
    ports:
    - name: http
      targetPort: 80
      port: 80
  ---
  apiVersion: apps/v1
  kind: Deployment
  metadata:
    name: old-nginx
    namespace: default
  spec:
    replicas: 2
    selector:
      matchLabels:
        app: old-nginx
    template:
      metadata:
        labels:
          app: old-nginx
      spec:
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

```
        containers:
        - name: nginx
          image: registry.cn-hangzhou.aliyuncs.com/acs-sample/old-nginx
          ports:
          - name: http
            containerPort: 80

2、更新资源清单文件
    [root@k8s-master01 ~]# kubectl apply -f nginx-old.yaml
    ingress.networking.k8s.io/ingress-old created
    service/old-nginx created
    deployment.apps/old-nginx created

3、访问测试
    [root@k8s-master01  ~]#  curl  --resolve  nginx.zyf.com:80:192.168.128.21
http://nginx.zyf.com
    old
```

（2）灰度发布新版本服务

```
1、创建资源清单文件
    [root@k8s-master01 ~]# vi nginx-new-v1.yaml
    ---
    apiVersion: networking.k8s.io/v1
    kind: Ingress
    metadata:
      name: ingress-new
      namespace: default
      annotations:
        nginx.ingress.kubernetes.io/canary: "true"
        nginx.ingress.kubernetes.io/canary-weight: "10"
    spec:
      ingressClassName: nginx
      rules:
      - host: nginx.zyf.com
        http:
          paths:
          - path: /
            pathType: Prefix
            backend:
              service:
               name: new-nginx
               port:
```

```
              number: 80
  ---
  apiVersion: v1
  kind: Service
  metadata:
    name: new-nginx
    namespace: default
  spec:
    selector:
      app: new-nginx
    ports:
    - name: http
      targetPort: 80
      port: 80
  ---
  apiVersion: apps/v1
  kind: Deployment
  metadata:
    name: new-nginx
    namespace: default
  spec:
    replicas: 2
    selector:
      matchLabels:
        app: new-nginx
    template:
      metadata:
        labels:
          app: new-nginx
      spec:
        containers:
        - name: nginx
          image: registry.cn-hangzhou.aliyuncs.com/acs-sample/new-nginx
          ports:
          - name: http
            containerPort: 80
```

2、更新资源清单文件

```
[root@k8s-master01 ~]# kubectl apply -f nginx-new-v1.yaml
ingress.networking.k8s.io/ingress-new created
service/new-nginx created
deployment.apps/new-nginx created
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

3、测试

```
[root@k8s-master01 ~]# for i in `seq 10`
do
curl --resolve nginx.zyf.com:80:192.168.128.21 http://nginx.zyf.com
done
```

```
[root@k8s-master01 ~]# for i in `seq 10`
> do
> curl --resolve nginx.zyf.com:80:192.168.128.21 http://nginx.zyf.com
> done
old
old
old
old
old
old
new
old
old
old
[root@k8s-master01 ~]#
```

（3）系统运行一段时间后，当新版本服务已经稳定并且符合预期后，需要下线老版本的服务，仅保留新版本服务在线上运行。

1、将所有流量切到新版本服务上

```
[root@k8s-master01 ~]# vi nginx-new.yaml
  annotations:
    nginx.ingress.kubernetes.io/canary-weight: "100"
```

```
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ingress-new
  namespace: default
  annotations:
    nginx.ingress.kubernetes.io/canary: "true"
    nginx.ingress.kubernetes.io/canary-weight: "100"
spec:
  ingressClassName: nginx
  rules:
  - host: nginx.zyf.com
    http:
```

2、更新 ingress

```
[root@k8s-master01 ~]# kubectl apply -f nginx-new.yaml
ingress.networking.k8s.io/ingress-new configured
service/new-nginx unchanged
deployment.apps/new-nginx unchanged
```

3、测试是否所有流量都到新版本上

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

```
[root@k8s-master01 ~]# for i in `seq 10`; do curl --resolve nginx.zyf.com:80:192.168.128.21 http://nginx.zyf.com; done
```

```
[root@k8s-master01 ~]# for i in `seq 10`; do curl --resolve nginx.zyf.com:80:192.168.128.21 http://nginx.zyf.com; done
new
new
new
new
new
new
new
new
new
new
[root@k8s-master01 ~]#
```

4、删除旧版本

```
[root@k8s-master01 ~]# kubectl delete -f nginx-old.yaml
ingress.networking.k8s.io "ingress-old" deleted
service "old-nginx" deleted
deployment.apps "old-nginx" deleted
```

4.4.2、基于客户端请求头的流量切分

假设线上已运行了一套对外提供的七层 demo 应用，此时开发了一些新的功能，需要上线新版本 demo 应用，但是又不想直接替换成新版本 demo 应用，而是希望将请求头包含 user=kubesre 的客户端请求转发到新版本 demo 应用中，进行验证测试新版本 demo 应用，等测试验证通过并稳定后，可将所有流量从老版本 demo 应用切换到新版本 demo 应用中，再平滑地将老版本 demo 应用下线。

（1）部署旧版本

```
[root@k8s-master01 ~]# kubectl apply -f nginx-old.yaml
ingress.networking.k8s.io/ingress-old created
service/old-nginx created
deployment.apps/old-nginx created
```

（2）部署新版本

1、创建资源清单文件

```
[root@k8s-master01 ~]# cat nginx-new-v2.yaml
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ingress-new
  namespace: default
  annotations:
    nginx.ingress.kubernetes.io/canary: "true"
```

```
    nginx.ingress.kubernetes.io/canary-by-header: "user"
    nginx.ingress.kubernetes.io/canary-by-header-value: "kubesre"
spec:
  ingressClassName: nginx
  rules:
  - host: nginx.zyf.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
           name: new-nginx
           port:
            number: 80
---
apiVersion: v1
kind: Service
metadata:
  name: new-nginx
  namespace: default
spec:
  selector:
    app: new-nginx
  ports:
  - name: http
    targetPort: 80
    port: 80
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: new-nginx
  namespace: default
spec:
  replicas: 2
  selector:
    matchLabels:
      app: new-nginx
  template:
    metadata:
      labels:
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

```
            app: new-nginx
        spec:
          containers:
          - name: nginx
            image: registry.cn-hangzhou.aliyuncs.com/acs-sample/new-nginx
            ports:
            - name: http
              containerPort: 80

2、更新资源清单文件
    [root@k8s-master01 ~]# kubectl apply -f nginx-new-v2.yaml
    ingress.networking.k8s.io/ingress-new created
    service/new-nginx created
    deployment.apps/new-nginx created

3、测试
    [root@k8s-master01  ~]#  curl  --resolve  nginx.zyf.com:80:192.168.128.21
http://nginx.zyf.com
    old
    [root@k8s-master01 ~]# curl --resolve nginx.zyf.com:80:192.168.128.21 -H "user:
kubesre" http://nginx.zyf.com
    new
```

4.4.3、基于客户端来源 IP 的流量切分

假设线上已运行了一套对外提供的七层 demo 应用，此时开发了一些新的功能，需要上线新版本 demo 应用，又不想直接替换成新版本 demo 应用，而是只希望公司内部人员能访问到新版本 demo 应用中，进行测试验证新版本 demo 应用，非公司内部人员访问还是访问到老版本应用中。等公司内部人员测试验证通过并稳定后，可将所有流量从老版本 demo 应用切换到新版本 demo 应用中，再平滑地将老版本 demo 应用下线。

（1）部署旧版本

```
    [root@k8s-master01 ~]# kubectl apply -f nginx-old.yaml
    ingress.networking.k8s.io/ingress-old created
    service/old-nginx created
    deployment.apps/old-nginx created
```

（2）部署新版本

```
1、创建资源清单文件
    [root@k8s-master01 ~]# vi nginx-new-v3.yaml
    ---
```

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ingress-new
  namespace: default
  annotations:
    nginx.ingress.kubernetes.io/canary: "true"
    nginx.ingress.kubernetes.io/canary-by-header: "X-Forwarded-For"
    nginx.ingress.kubernetes.io/canary-by-header-value: "10.10.3.152"
spec:
  ingressClassName: nginx
  rules:
  - host: nginx.zyf.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
           name: new-nginx
           port:
            number: 80
---
apiVersion: v1
kind: Service
metadata:
  name: new-nginx
  namespace: default
spec:
  selector:
    app: new-nginx
  ports:
  - name: http
    targetPort: 80
    port: 80
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: new-nginx
  namespace: default
spec:
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

```
    replicas: 2
    selector:
      matchLabels:
        app: new-nginx
    template:
      metadata:
        labels:
          app: new-nginx
      spec:
        containers:
        - name: tomcat
          image: registry.cn-hangzhou.aliyuncs.com/acs-sample/new-nginx
          ports:
          - name: http
            containerPort: 80

2、更新资源清单文件
    [root@k8s-master01 ~]# kubectl apply -f nginx-new-v3.yaml
    ingress.networking.k8s.io/ingress-new created
    service/new-nginx created
    deployment.apps/new-nginx created

3、测试
    # 通过请求头模拟来源 IP，真实环境不需要
    [root@k8s-master01 ~]# curl -H "X-Forwarded-For:10.10.3.152" --resolve
nginx.zyf.com:80:192.168.128.21 http://nginx.zyf.com
    new

    # 其他来源则切到老版本
    [root@k8s-master01 ~]# curl --resolve nginx.zyf.com:80:192.168.128.21
http://nginx.zyf.com
    old
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。