Docker+K8S+DevOps 微服务架构师

学神 IT 教育：从零基础到实战，从入门到精通！

联系方式：

学神 IT 教育官方网站: http://www.xuegod.cn

学神 K8S 精英学习 11 群 QQ 群: 957231097

学习顾问：小语老师　　学习顾问：边边老师　　学神微信公众号

微信扫码添加学习顾问微信，同时扫码关注学神公众号了解最新行业动态，获取更多学习资料及答疑就业服务！

第 4 章：k8s 应用更新策略：灰度发布和蓝绿发布

本节所讲内容：

4.1 生产环境如何实现蓝绿部署

实战 1：通过 k8s 实现线上业务的蓝绿部署

4.2 通过 k8s 实现滚动更新：滚动更新流程和策略

4.3 通过 k8s 完成线上业务的金丝雀发布

实战 2：七层调度器 Ingress Controller 安装和配置

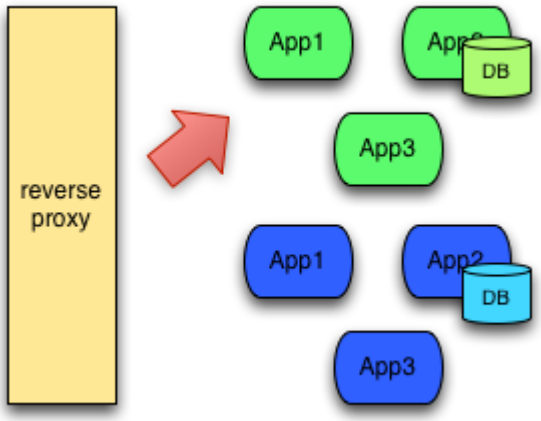
实战 3：通过 Ingress-nginx 实现灰度发布

4.1 生产环境如何实现蓝绿部署?

4.1.1 什么是蓝绿部署?

蓝绿部署中，一共有两套系统：一套是正在提供服务系统，标记为“绿色”；另一套是准备发布的系统，标记为“蓝色”。两套系统都是功能完善的、正在运行的系统，只是系统版本和对外服务情况不同。

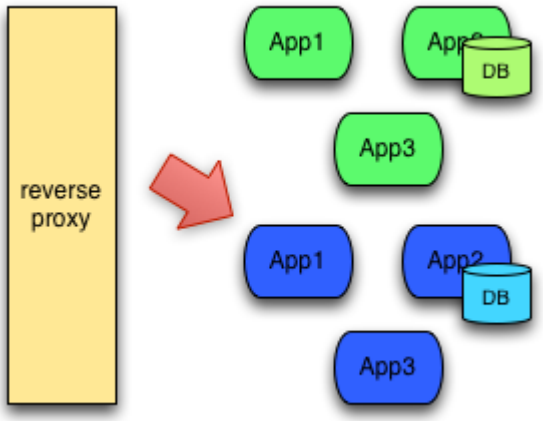
开发新版本，要用新版本替换线上的旧版本，在线上的系统之外，搭建了一个使用新版本代码的全新系统。 这时候，一共有两套系统在运行，正在对外提供服务的老系统是绿色系统，新部署的系统是蓝色系统。



蓝色系统不对外提供服务，用来做什么呢?

用来做发布前测试，测试过程中发现任何问题，可以直接在蓝色系统上修改，不干扰用户正在使用的系统。(注意，两套系统没有耦合的时候才能百分百保证不干扰)

蓝色系统经过反复的测试、修改、验证，确定达到上线标准之后，直接将用户切换到蓝色系统：



切换后的一段时间内，依旧是蓝绿两套系统并存，但是用户访问的已经是蓝色系统。这段时间内观察蓝色系统（新系统）工作状态，如果出现问题，直接切换回绿色系统。

当确信对外提供服务的蓝色系统工作正常，不对外提供服务的绿色系统已经不再需要的时候，蓝色系统正式成为对外提供服务系统，成为新的绿色系统。 原先的绿色系统可以销毁，将资源释放出来，用于部署下一个蓝色系统。

4.1.2 蓝绿部署的优势和缺点

优点：

1、更新过程无需停机，风险较少

2、回滚方便，只需要更改路由或者切换 DNS 服务器，效率较高

缺点：

1、成本较高，需要部署两套环境。如果新版本中基础服务出现问题，会瞬间影响全网用户；如果新版本有问题也会影响全网用户。

2、需要部署两套机器，费用开销大

3、在非隔离的机器（Docker、VM）上操作时，可能会导致蓝绿环境被摧毁风险

4、负载均衡器/反向代理/路由/DNS 处理不当，将导致流量没有切换过来情况出现

实战 1：通过 k8s 实现线上业务的蓝绿部署

下面实验需要的镜像包在课件，把镜像压缩包上传到 k8s 的各个工作节点，docker load -i 解压：

```
[root@xuegod62 ~]# docker load -i myapp-lan.tar.gz
[root@xuegod64 ~]# docker load -i myapp-lv.tar.gz
```

Kubernetes 不支持内置的蓝绿部署。目前最好的方式是创建新的 deployment，然后更新应用程序的 service 以指向新的 deployment 部署的应用

1.创建蓝色部署环境（新上线的环境，要替代绿色环境）

下面步骤在 k8s 的控制节点操作：

```
[root@xuegod63 ~]# kubectl create ns blue-green

[root@xuegod63 ~]# cat   lan.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp-v1
  namespace: blue-green
spec:
  replicas: 3
  selector:
   matchLabels:
    app: myapp
    version: v1
  template:
   metadata:
    labels:
     app: myapp
     version: v1
   spec:
    containers:
    - name: myapp
      image: janakiramm/myapp:v1
      imagePullPolicy: IfNotPresent
      ports:
      - containerPort: 80
```

然后可以使用 kubectl 命令创建部署。

```
[root@xuegod63 ~]# kubectl apply -f lan.yaml
```

验证部署是否成功:

```
[root@xuegod63 ~]# kubectl get pods -n blue-green
```

显示如下：

```
NAME                          READY    STATUS     RESTARTS    AGE
myapp-v1-67fd9fc9c8-tsl92     1/1      Running    0           53s
myapp-v1-67fd9fc9c8-24tbp     1/1      Running    0           53s
myapp-v1-67fd9fc9c8-cw59c     1/1      Running    0           53s
```

2.创建绿色部署环境（原来的部署环境）

```
[root@xuegod63 ~]# cat   lv.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp-v2
```

```
  namespace: blue-green
spec:
  replicas: 3
  selector:
   matchLabels:
    app: myapp
    version: v2
  template:
   metadata:
    labels:
     app: myapp
     version: v2
   spec:
    containers:
    - name: myapp
      image: janakiramm/myapp:v2
      imagePullPolicy: IfNotPresent
      ports:
      - containerPort: 80
```

可以使用 kubectl 命令创建部署。

```
[root@xuegod63 ~]# kubectl apply -f lv.yaml
```

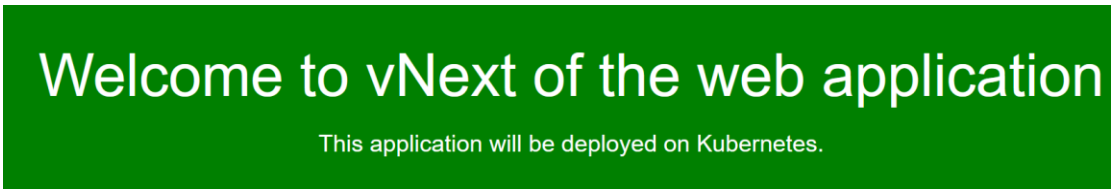
创建前端 service

```
[root@xuegod63 ~]# cat service_lanlv.yaml
apiVersion: v1
kind: Service
metadata:
  name: myapp-lan
  namespace: blue-green
  labels:
    app: myapp
    version: v2
spec:
  type: NodePort
  ports:
  - port: 80
    nodePort: 30062
    name: http
  selector:
    app: myapp
    version: v2
```

更新服务：

[root@xuegod63 ~]# kubectl apply -f service_lanlv.yaml

在浏览器访问 http://k8s-master 节点 ip:30062 显示如下：



修改 service_lanlv.yaml 配置文件，修改标签，让其匹配到蓝程序（升级之之后的程序）

```
[root@xuegod63 ~]# cat service_lanlv.yaml
apiVersion: v1
kind: Service
metadata:
  name: myapp-lan
  namespace: blue-green
  labels:
    app: myapp
    version: v1
spec:
  type: NodePort
  ports:
  - port: 80
    nodePort: 30062
    name: http
  selector:
    app: myapp
    version: v1
```

更新资源清单文件：

[root@xuegod63 ~]# kubectl apply -f service_lanlv.yaml

在浏览器访问 http://k8s-master 节点 ip:30062 显示如下：

实验完成之后，把资源先删除，以免影响后面实验：

```
[root@xuegod63 ~]# kubectl delete   -f lan.yaml
[root@xuegod63 ~]# kubectl delete   -f lv.yaml
[root@xuegod63 ~]# kubectl delete -f service_lanlv.yaml
```

4.2 通过 k8s 实现滚动更新-滚动更新流程和策略

4.2.1 滚动更新简介

滚动更新是一种自动化程度较高的发布方式，用户体验比较平滑，是目前成熟型技术组织所采用的主流发布方式，一次滚动发布一般由若干个发布批次组成，每批的数量一般是可以配置的（可以通过发布模板定义），例如第一批 1 台，第二批 10%，第三批 50%，第四批 100%。每个批次之间留观察间隔，通过手工验证或监控反馈确保没有问题再发下一批次，所以总体上滚动式发布过程是比较缓慢的

4.2.2 在 k8s 中实现金滚动更新

首先看下 Deployment 资源对象的组成：

```
[root@xuegod63 ~]# kubectl explain deployment
[root@xuegod63 ~]# kubectl explain deployment.spec
 KIND:     Deployment
 VERSION:  apps/v1
 RESOURCE: spec <Object>
 DESCRIPTION:
      Specification of the desired behavior of the Deployment.
      DeploymentSpec is the specification of the desired behavior of the
      Deployment.
 FIELDS:
    minReadySeconds    <integer>
      Minimum number of seconds for which a newly created pod should be ready
      without any of its container crashing, for it to be considered available.
      Defaults to 0 (pod will be considered available as soon as it is ready)
    paused      <boolean>
      Indicates that the deployment is paused.
#暂停，当我们更新的时候创建 pod 先暂停，不是立即更新
    progressDeadlineSeconds     <integer>
      The maximum time in seconds for a deployment to make progress before it is
      considered to be failed. The deployment controller will continue to process
      failed deployments and a condition with a ProgressDeadlineExceeded reason
      will be surfaced in the deployment status. Note that progress will not be
      estimated during the time a deployment is paused. Defaults to 600s.
    replicas    <integer>
      Number of desired pods. This is a pointer to distinguish between explicit
      zero and not specified. Defaults to 1.
    revisionHistoryLimit <integer>
#保留的历史版本数，默认是 10 个
      The number of old ReplicaSets to retain to allow rollback. This is a
      pointer to distinguish between explicit zero and not specified. Defaults to
      10.
    selector    <Object> -required-
      Label selector for pods. Existing ReplicaSets whose pods are selected by
      this will be the ones affected by this deployment. It must match the pod
```

```
        template's labels.
      strategy     <Object>
   #更新策略，支持的滚动更新策略
        The deployment strategy to use to replace existing pods with new ones.
      template     <Object> -required-
        Template describes the pods that will be created.

  kubectl explain deploy.spec.strategy
   KIND:       Deployment
   VERSION:    apps/v1
   RESOURCE: strategy <Object>
   DESCRIPTION:
        The deployment strategy to use to replace existing pods with new ones.
        DeploymentStrategy describes how to replace existing pods with new ones.
   FIELDS:
      rollingUpdate     <Object>
        Rolling update config params. Present only if DeploymentStrategyType =
        RollingUpdate.
      type     <string>
        Type of deployment. Can be "Recreate" or "RollingUpdate". Default is
        RollingUpdate.
   #支持两种更新，Recreate 和 RollingUpdate
   #Recreate 是重建式更新，删除一个更新一个
   #RollingUpdate 滚动更新，定义滚动更新的更新方式的，也就是 pod 能多几个，少几个，控制
更新力度的

  kubectl explain deploy.spec.strategy.rollingUpdate
   KIND:       Deployment
   VERSION:    apps/v1
   RESOURCE: rollingUpdate <Object>
   DESCRIPTION:
        Rolling update config params. Present only if DeploymentStrategyType =
        RollingUpdate.
        Spec to control the desired behavior of rolling update.
   FIELDS:
      maxSurge   <string>
        The maximum number of pods that can be scheduled above the desired
number
        of pods. Value can be an absolute number (ex: 5) or a percentage of desired
        pods (ex: 10%). This can not be 0 if MaxUnavailable is 0. Absolute number
        is calculated from percentage by rounding up. Defaults to 25%. Example:
        when this is set to 30%, the new ReplicaSet can be scaled up immediately
        when the rolling update starts, such that the total number of old and new
        pods do not exceed 130% of desired pods. Once old pods have been killed,
```

new ReplicaSet can be scaled up further, ensuring that total number of pods running at any time during the update is at most 130% of desired pods.

#我们更新的过程当中最多允许超出的指定的目标副本数有几个；

它有两种取值方式，第一种直接给定数量，第二种根据百分比，百分比表示原本是 5 个，最多可以超出 20%，那就允许多一个，最多可以超过 40%，那就允许多两个

maxUnavailable <string>

The maximum number of pods that can be unavailable during the update. Value can be an absolute number (ex: 5) or a percentage of desired pods (ex: 10%). Absolute number is calculated from percentage by rounding down. This can not be 0 if MaxSurge is 0. Defaults to 25%. Example: when this is set to 30%, the old ReplicaSet can be scaled down to 70% of desired pods immediately when the rolling update starts. Once new pods are ready, old ReplicaSet can be scaled down further, followed by scaling up the new ReplicaSet, ensuring that the total number of pods available at all times during the update is at least 70% of desired pods.

#最多允许几个不可用

假设有 5 个副本，最多一个不可用，就表示最少有 4 个可用

deployment 是一个三级结构，deployment 控制 replicaset，replicaset 控制 pod，

例子：用 deployment 创建一个 pod

[root@xuegod63 ~]# cat deploy-demo.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp-v1
  namespace: blue-green
spec:
  replicas: 2
  selector:
   matchLabels:
    app: myapp
    version: v1
  template:
   metadata:
    labels:
     app: myapp
     version: v1
   spec:
    containers:
    - name: myapp
      image: janakiramm/myapp:v1
      imagePullPolicy: IfNotPresent
```

```
        ports:
        - containerPort: 80
```

更新资源清单文件：

```
[root@xuegod63 ~]# kubectl apply -f deploy-demo.yaml
```

查看 deploy 状态：

```
[root@xuegod63 ~]# kubectl get deploy -n blue-green
```

显示如下：

```
NAME          READY    UP-TO-DATE    AVAILABLE    AGE
myapp-v1      2/2      2             2            60s
```

创建的控制器名字是 myapp-v1

```
[root@xuegod63 ~]# kubectl get rs -n blue-green
```

显示如下：

```
AME                      DESIRED    CURRENT    READY    AGE
myapp-v1-67fd9fc9c8      2          2          2        2m35s
```

创建 deploy 的时候也会创建一个 rs（replicaset），67fd9fc9c8 这个随机数字是我们引用 pod 的模板 template 的名字的 hash 值

```
[root@xuegod63 ~]# kubectl get pods -n blue-green
```

显示如下：

```
NAME                           READY    STATUS     RESTARTS    AGE
myapp-v1-67fd9fc9c8-tsl92      1/1      Running    0           3m23s
myapp-v1-67fd9fc9c8-np57d      1/1      Running    0           3m23s
```

通过 deployment 管理应用，在更新的时候，可以直接编辑配置文件实现，比方说想要修改副本数，把 2 个变成 3 个

```
[root@xuegod63 ~]# cat   deploy-demo.yaml
```

直接修改 replicas 数量，如下，变成 3

```
spec:
  replicas: 3
```

修改之后保存退出，执行

```
[root@xuegod63 ~]# kubectl apply -f deploy-demo.yaml
```

注意：apply 不同于 create，apply 可以执行多次；create 执行一次，再执行就会报错有重复。

```
[root@xuegod63 ~]# kubectl get pods   -n blue-green
```

显示如下：

```
NAME                           READY    STATUS     RESTARTS    AGE
myapp-v1-67fd9fc9c8-tsl92      1/1      Running    0           8m18s
```

```
myapp-v1-67fd9fc9c8-4bv5n   1/1     Running   0          8m18s
myapp-v1-67fd9fc9c8-cw59c   1/1     Running   0          18s
```

上面可以看到 pod 副本数变成了 3 个

```
#查看 myapp-v1 这个控制器的详细信息
[root@xuegod63 ~]# kubectl describe deploy myapp-v1 -n blue-green
#显示如下：
 Name:                   myapp-v1
 Namespace:              blue-green
 CreationTimestamp:      Sun, 21 Mar 2021 18:46:52 +0800
 Labels:                 <none>
 Annotations:            deployment.kubernetes.io/revision: 1
 Selector:               app=myapp,version=v1
 Replicas:               3 desired | 3 updated | 3 total | 3 available | 0 unavailable
 StrategyType:           RollingUpdate
 #默认的更新策略 rollingUpdate
 MinReadySeconds:        0
 RollingUpdateStrategy:  25% max unavailable, 25% max surge
 #最多允许多 25%个 pod，25%表示不足一个，可以补一个
 Pod Template:
   Labels:  app=myapp
            version=v1
   Containers:
    myapp:
     Image:        janakiramm/myapp:v1
     Port:         80/TCP
     Host Port:    0/TCP
     Environment:  <none>
     Mounts:       <none>
   Volumes:        <none>
 Conditions:
   Type           Status  Reason
   ----           ------  ------
   Progressing    True    NewReplicaSetAvailable
   Available      True    MinimumReplicasAvailable
 OldReplicaSets:  <none>
 NewReplicaSet:   myapp-v1-67fd9fc9c8 (3/3 replicas created)
 Events:
   Type    Reason             Age                  From                   Message
   ----    ------             ----                 ----                   -------
   Normal  ScalingReplicaSet  3m26s                deployment-controller  Scaled
down replica set myapp-v1-67fd9fc9c8 to 2
   Normal  ScalingReplicaSet  2m1s (x2 over 10m)   deployment-controller  Scaled
```

up replica set myapp-v1-67fd9fc9c8 to 3

例子：测试滚动更新
在终端执行如下：

[root@xuegod63 ~]# kubectl get pods -l app=myapp -n blue-green -w

打开一个新的终端窗口更改镜像版本，按如下操作：

[root@xuegod63 ~]# vim deploy-demo.yaml

把 image: janakiramm/myapp:v1 变成 image: janakiramm/myapp:v2

保存退出，执行

[root@xuegod63 ~]# kubectl apply -f deploy-demo.yaml

再回到刚才监测的那个窗口，可以看到信息如下：

```
NAME                          READY   STATUS              RESTARTS   AGE
myapp-v1-67fd9fc9c8-tsl92     1/1     Running             0          22m
myapp-v1-67fd9fc9c8-4bv5n     1/1     Running             0          22m
myapp-v1-67fd9fc9c8-cw59c     1/1     Running             0          14m
myapp-v1-75fb478d6c-24tbp     0/1     Pending             0          0s
myapp-v1-75fb478d6c-24tbp     0/1     Pending             0          0s
myapp-v1-75fb478d6c-24tbp     0/1     ContainerCreating   0          0s
myapp-v1-75fb478d6c-24tbp     1/1     Running             0          11s
myapp-v1-67fd9fc9c8-cw59c     1/1     Terminating         0          15m
myapp-v1-75fb478d6c-f52l6     0/1     Pending             0          0s
myapp-v1-75fb478d6c-f52l6     0/1     Pending             0          0s
myapp-v1-75fb478d6c-f52l6     0/1     ContainerCreating   0          0s
myapp-v1-67fd9fc9c8-cw59c     0/1     Terminating         0          15m
myapp-v1-75fb478d6c-f52l6     1/1     Running             0          11s
myapp-v1-67fd9fc9c8-4bv5n     1/1     Terminating         0          23m
myapp-v1-75fb478d6c-jlw28     0/1     Pending             0          0s
myapp-v1-75fb478d6c-jlw28     0/1     Pending             0          0s
myapp-v1-75fb478d6c-jlw28     0/1     ContainerCreating   0          0s
myapp-v1-75fb478d6c-jlw28     1/1     Running             0          1s
```

pending 表示正在进行调度，ContainerCreating 表示正在创建一个 pod，running 表示运 行一个 pod，running 起来一个 pod 之后再 Terminating（停掉）一个 pod，以此类推，直 到所有 pod 完成滚动升级

在另外一个窗口执行

[root@xuegod63 ~]# kubectl get rs -n blue-green

显示如下：

```
NAME                       DESIRED   CURRENT   READY   AGE
myapp-v1-75fb478d6c   3         3         3       2m7s
myapp-v1-67fd9fc9c8   0         0         0       25m
```

上面可以看到 rs 有两个，下面那个是升级之前的，已经被停掉，但是可以随时回滚

[root@xuegod63 ~]# kubectl rollout history deployment myapp-v1 -n blue-green

查看 myapp-v1 这个控制器的滚动历史，显示如下：

```
deployment.apps/myapp-v1
REVISION  CHANGE-CAUSE
1         <none>
2         <none>
```

回滚操作如下：

[root@xuegod63 ~]# kubectl rollout undo deployment/myapp-v1 --to-revision=2 -n blue-green

4.2.3 自定义滚动更新策略

maxSurge 和 maxUnavailable 用来控制滚动更新的更新策略

取值范围

数值

1. maxUnavailable: [0，副本数]
2. maxSurge: [0，副本数]

注意：两者不能同时为 0。

比例

1. maxUnavailable: [0%, 100%] 向下取整，比如 10 个副本，5%的话==0.5 个，但计算按照 0 个；
2. maxSurge: [0%, 100%] 向上取整，比如 10 个副本，5%的话==0.5 个，但计算按照 1 个；

注意：两者不能同时为 0。

建议配置

1. maxUnavailable == 0
2. maxSurge == 1

这是我们生产环境提供给用户的默认配置。即“一上一下，先上后下”最平滑原则：

1 个新版本 pod ready（结合 readiness）后，才销毁旧版本 pod。此配置适用场景是平滑更新、保证服务平稳，但也有缺点，就是“太慢”了。

总结：

maxUnavailable：和期望的副本数比，不可用副本数最大比例（或最大值），这个值越小，越能保证服务稳定，更新越平滑；

maxSurge：和期望的副本数比，超过期望副本数最大比例（或最大值），这个值调的越大，副本更新速度越快。

自定义策略：

修改更新策略：maxUnavailable=1，maxSurge=1

[root@xuegod63 ~]# kubectl patch deployment myapp-v1 -p '{"spec":{"strategy":{"rollingUpdate": {"maxSurge":1,"maxUnavailable":1}}}}' -n blue-green

查看 myapp-v1 这个控制器的详细信息

[root@xuegod63 ~]# kubectl describe deployment myapp-v1 -n blue-green

显示如下：

RollingUpdateStrategy: 1 max unavailable, 1 max surge

上面可以看到 RollingUpdateStrategy: 1 max unavailable, 1 max surge

这个 rollingUpdate 更新策略变成了刚才设定的，因为我们设定的 pod 副本数是 3，1 和 1 表示最少不能少于 2 个 pod，最多不能超过 4 个 pod

这个就是通过控制 RollingUpdateStrategy 这个字段来设置滚动更新策略的

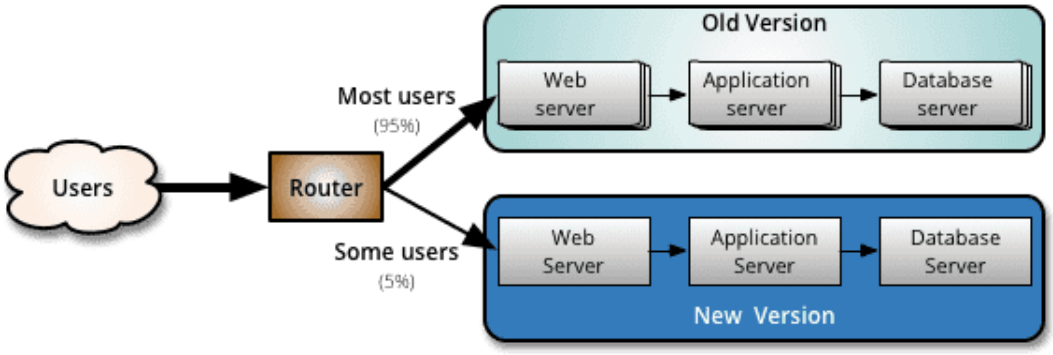
4.3 通过 k8s 完成线上业务的金丝雀发布

4.3.1 金丝雀发布简介

金丝雀发布的由来：17 世纪，英国矿井工人发现，金丝雀对瓦斯这种气体十分敏感。空气中哪怕有极其微量的瓦斯，金丝雀也会停止歌唱；当瓦斯含量超过一定限度时，虽然人类毫无察觉，金丝雀却早已毒发身亡。当时在采矿设备相对简陋的条件下，工人们每次下井都会带上一只金丝雀作为瓦斯检测指标，以便在危险状况下紧急撤离。

金丝雀发布（又称灰度发布、灰度更新）：金丝雀发布一般先发 1 台，或者一个小比例，例如 2%的服务器，主要做流量验证用，也称为金丝雀（Canary）测试 （国内常称灰度测试）。

简单的金丝雀测试一般通过手工测试验证，复杂的金丝雀测试需要比较完善的监控基础设施配合，通过监控指标反馈，观察金丝雀的健康状况，作为后续发布或回退的依据。 如果金丝测试通过，则把剩余的 V1 版本全部升级为 V2 版本。如果金丝雀测试失败，则直接回退金丝雀，发布失败。



优点：灵活，策略自定义，可以按照流量或具体的内容进行灰度(比如不同账号，不同参数)，出现问题不会影响全网用户

缺点：没有覆盖到所有的用户导致出现问题不好排查

4.3.2 在 k8s 中实现金丝雀发布

打开一个标签 1 监测更新过程

[root@xuegod63 ~]# kubectl get pods -l app=myapp -n blue-green -w

打开另一个标签 2 执行如下操作：

[root@xuegod63 ~]# kubectl set image deployment myapp-v1 myapp=janakiramm/myapp:v2 -n blue-green && kubectl rollout pause deployment myapp-v1 -n blue-green

回到标签 1 观察，显示如下：

```
NAME                          READY   STATUS              RESTARTS   AGE
myapp-v1-67fd9fc9c8-5fd2f     1/1     Running             0          86s
myapp-v1-67fd9fc9c8-92mdr     1/1     Running             0          86s
myapp-v1-75fb478d6c-wddds     0/1     Pending             0          0s
myapp-v1-75fb478d6c-wddds     0/1     Pending             0          0s
myapp-v1-75fb478d6c-wddds     0/1     ContainerCreating   0          0s
myapp-v1-75fb478d6c-wddds     0/1     ContainerCreating   0          1s
myapp-v1-75fb478d6c-wddds     1/1     Running             0          2s
```

注：上面的解释说明把 myapp 这个容器的镜像更新到 janakiramm/myapp:v2 版本 更新镜像之后，创建一个新的 pod 就立即暂停，这就是我们说的金丝雀发布；如果暂停几个小时之后没有问题，那么取消暂停，就会依次执行后面步骤，把所有 pod 都升级。

解除暂停：

回到标签 1 继续观察：

打开标签 2 执行如下：

[root@xuegod63 ~]# kubectl rollout resume deployment myapp-v1 -n blue-green

在标签 1 可以看到如下一些信息，下面过程是把余下的 pod 里的容器都更的版本：

```
NAME                          READY   STATUS              RESTARTS   AGE
myapp-v1-67fd9fc9c8-5fd2f     1/1     Running             0          86s
myapp-v1-67fd9fc9c8-92mdr     1/1     Running             0          86s
myapp-v1-75fb478d6c-wddds     0/1     Pending             0          0s
myapp-v1-75fb478d6c-wddds     0/1     Pending             0          0s
myapp-v1-75fb478d6c-wddds     0/1     ContainerCreating   0          0s
myapp-v1-75fb478d6c-wddds     0/1     ContainerCreating   0          1s
myapp-v1-75fb478d6c-wddds     1/1     Running             0          2s
myapp-v1-67fd9fc9c8-92mdr     1/1     Terminating         0          10m
myapp-v1-75fb478d6c-z6f5z     0/1     Pending             0          0s
myapp-v1-75fb478d6c-z6f5z     0/1     Pending             0          0s
myapp-v1-75fb478d6c-z6f5z     0/1     ContainerCreating   0          0s
myapp-v1-75fb478d6c-z6f5z     0/1     ContainerCreating   0          1s
myapp-v1-67fd9fc9c8-92mdr     0/1     Terminating         0          10m
```

```
myapp-v1-75fb478d6c-z6f5z    1/1     Running       0          2s
myapp-v1-67fd9fc9c8-5fd2f    1/1     Terminating   0          10m
myapp-v1-67fd9fc9c8-5fd2f    0/1     Terminating   0          10m
myapp-v1-67fd9fc9c8-5fd2f    0/1     Terminating   0          10m
myapp-v1-67fd9fc9c8-5fd2f    0/1     Terminating   0          10m
myapp-v1-67fd9fc9c8-92mdr    0/1     Terminating   0          10m
myapp-v1-67fd9fc9c8-92mdr    0/1     Terminating   0          10m
```

[root@xuegod63 ~]# kubectl get rs -n blue-green

可以看到 replicaset 控制器有 2 个了

```
NAME                    DESIRED   CURRENT   READY   AGE
myapp-v1-67fd9fc9c8     0         0         0       13m
myapp-v1-75fb478d6c     2         2         2       7m28s
```

回滚：

如果发现刚才升级的这个版本有问题可以回滚，查看当前有哪几个版本：

[root@xuegod63 ~]# kubectl rollout history deployment myapp-v1 -n blue-green

显示如下：

```
deployment.apps/myapp-v1
REVISION  CHANGE-CAUSE
1         <none>
2         <none>
```

上面说明一共有两个版本，回滚的话默认回滚到上一版本，可以指定参数回滚：

[root@xuegod63 ~]# kubectl rollout undo deployment myapp-v1 -n blue-green --to-revision=1

#回滚到的版本号是 1

[root@xuegod63 ~]# kubectl rollout history deployment myapp-v1 -n blue-green

显示如下：

```
deployment.apps/myapp-v1
REVISION  CHANGE-CAUSE
2         <none>
3         <none>
```

上面可以看到第一版没了，被还原成了第三版，第三版的前一版是第二版

[root@xuegod63 ~]# kubectl get rs -n blue-green -o wide

显示如下：

```
NAME                    DESIRED   CURRENT   READY   AGE   CONTAINERS
IMAGES                  SELECTOR
myapp-v1-67fd9fc9c8     2         2         2       18m   myapp
janakiramm/myapp:v1     app=myapp,pod-template-hash=67fd9fc9c8,version=v1
myapp-v1-75fb478d6c     0         0         0       12m   myapp
janakiramm/myapp:v2     app=myapp,pod-template-hash=75fb478d6c,version=v1
```

以看到上面的 rs 已经用第一个了，这个就是还原之后的 rs

实战 2：七层调度器 Ingress Controller 安装和配置

4.4 Ingress 和 Ingress Controller 深度解读

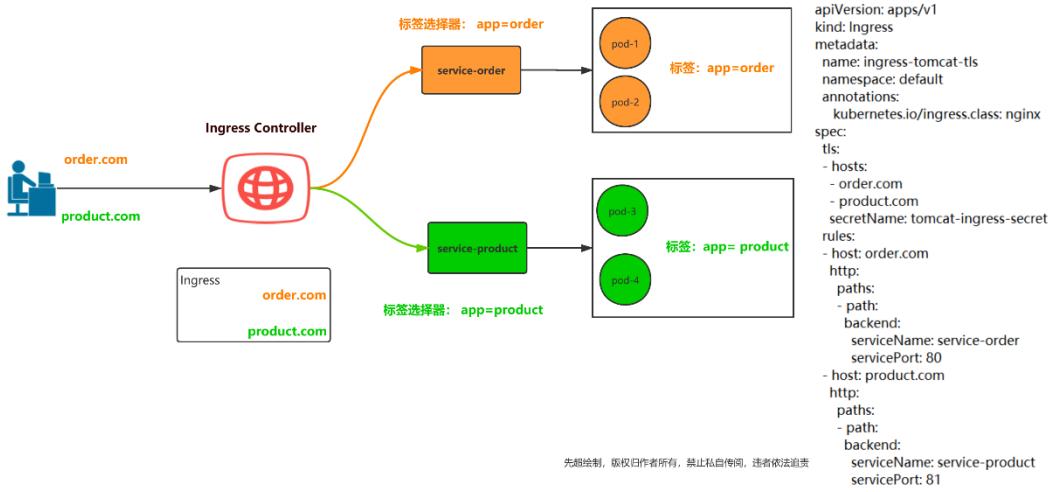
互动：为什么要使用 k8s 原生的 Ingress controller 做七层负载均衡?

4.4.1 Ingress 介绍

Ingress 官网定义：Ingress 可以把进入到集群内部的请求转发到集群中的一些服务上，从而可以把服务映射到集群外部。Ingress 能把集群内 Service 配置成外网能够访问的 URL，流量负载均衡，提供基于域名访问的虚拟主机等。

Ingress 简单的理解就是你原来需要改 Nginx 配置，然后配置各种域名对应哪个 Service，现在把这个动作抽象出来，变成一个 Ingress 对象，你可以用 yaml 创建，每次不要去改 Nginx 了，直接改 yaml 然后创建/更新就行了；那么问题来了：”Nginx 该怎么处理？”

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 Ingress Controller Nginx 里，最后 reload 一下，工作流程如下图：

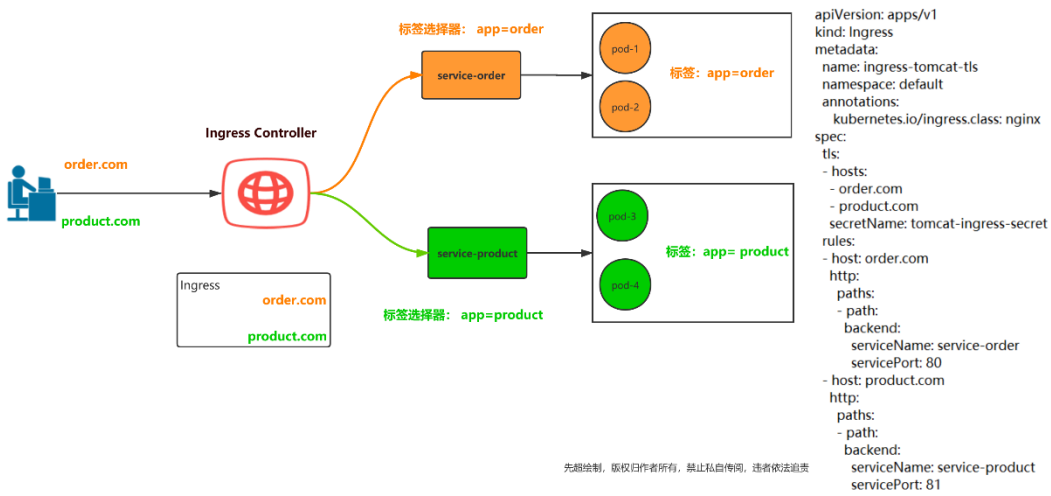


实际上 Ingress 也是 Kubernetes API 的标准资源类型之一，它其实就是一组基于 DNS 名称（host）或 URL 路径把请求转发到指定的 Service 资源的规则。用于将集群外部的请求流量转发到集群内部完成的服务发布。我们需要明白的是，Ingress 资源自身不能进行“流量穿透”，仅仅是一组规则的集合，这些集合规则还需要其他功能的辅助，比如监听某套接字，然后根据这些规则的匹配进行路由转发，这些能够为 Ingress 资源监听套接字并将流量转发的组件就是 Ingress Controller。

注：Ingress 控制器不同于 Deployment 控制器的是，Ingress 控制器不直接运行，它不由 kube-controller-manager 进行控制，它仅仅是 Kubernetes 集群的一个附件，类似于 CoreDNS，需要在集群上单独部署。

4.4.2 Ingress Controller 介绍

Ingress Controller 是一个七层负载均衡调度器，客户端的请求先到达这个七层负载均衡调度器，由七层负载均衡器在反向代理到后端 pod，常见的七层负载均衡器有 nginx、traefik，以我们熟悉的 nginx 为例，假如请求到达 nginx，会通过 upstream 反向代理到后端 pod 应用，但是后端 pod 的 ip 地址是一直在变化的，因此在后端 pod 前需要加一个 service，这个 service 只是起到分组的作用，那么我们 upstream 只需要填写 service 地址即可



4.4.3 Ingress 和 Ingress Controller 总结

Ingress Controller

Ingress Controller 可以理解为控制器，它通过不断的跟 Kubernetes API 交互，实时获取后端 Service、Pod 的变化，比如新增、删除等，结合 Ingress 定义的规则生成配置，然后动态更新上边的 Nginx 或者 trafik 负载均衡器，并刷新使配置生效，来达到服务自动发现的作用。

Ingress 则是定义规则，通过它定义某个域名的请求过来之后转发到集群中指定的 Service。它可以通过 Yaml 文件定义，可以给一个或多个 Service 定义一个或多个 Ingress 规则。

4.4.4 使用 Ingress Controller 代理 k8s 内部应用的流程

（1）部署 Ingress controller，我们 ingress controller 使用的是 nginx

（2）创建 Pod 应用，可以通过控制器创建 pod

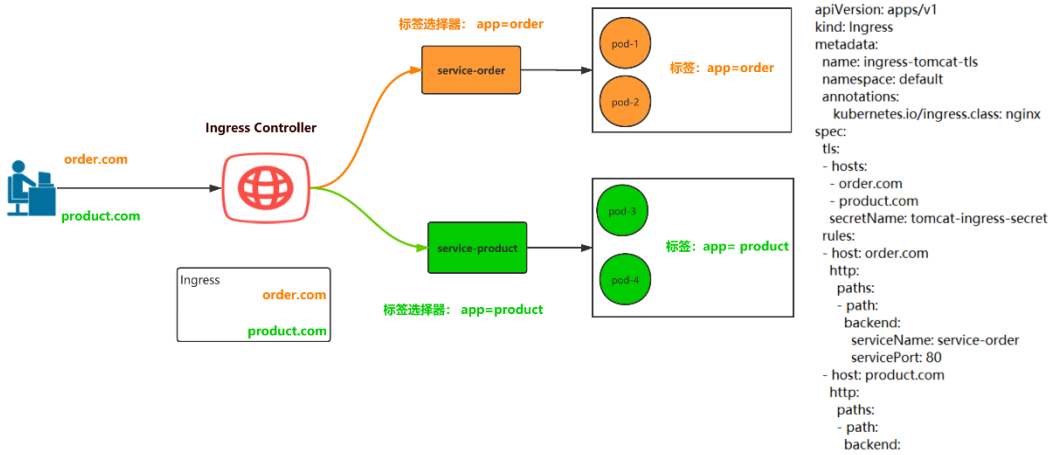
（3）创建 Service，用来分组 pod

（4）创建 Ingress http，测试通过 http 访问应用

（5）创建 Ingress https，测试通过 https 访问应用

客户端通过七层调度器访问后端 pod 的方式

使用七层负载均衡调度器 ingress controller 时，当客户端访问 kubernetes 集群内部的应用时，数据包走向如下图流程所示：



4.4.5 安装 Nginx Ingress Controller

#把 defaultbackend.tar.gz 和 nginx-ingress-controller.tar.gz 镜像上传到 xuegod62、xuegod64 节点，手动解压镜像：

```
[root@xuegod62 ~]# docker load -i defaultbackend.tar.gz
[root@xuegod62 ~]# docker load -i nginx-ingress-controller.tar.gz

[root@xuegod64 ~]# docker load -i defaultbackend.tar.gz
[root@xuegod64 ~]# docker load -i nginx-ingress-controller.tar.gz
```

#更新 yaml 文件，下面需要的 yaml 文件在课件，可上传到 xuegod63 机器上：

缺失 yaml 文件内容

安装 Ingress conrtroller 需要的 yaml 所在的 github 地址：https://github.com/kubernetes/ingress-nginx/

```
[root@xuegod63]# kubectl apply -f nginx-ingress-controller-rbac.yml
[root@xuegod63]# kubectl apply -f default-backend.yaml
deployment.apps/default-http-backend created
service/default-http-backend created
[root@xuegod63]# kubectl apply -f nginx-ingress-controller.yaml
deployment.apps/nginx-ingress-controller created
[root@xuegod63]# kubectl get pods -n kube-system | grep ingress
#显示如下，说明部署成功了：
nginx-ingress-controller-74cf657846-qrvdm    1/1       Running    0          30s
```

注意：

default-backend.yaml 和 nginx-ingress-controller.yaml 文件指定了 nodeName：xuegod62，表示 default 和 nginx-ingress-controller 部署在 xuegod62 节点，大家的 node 节点如果主机名不是 xuegod62，需要自行修改成自己的主机名，这样才会调度成功，一定要让 default-http-backend 和 nginx-ingress-controller 这两个 pod 在一个节点上。

default-backend.yaml：这是官方要求必须要给的默认后端，提供 404 页面的。它还提供了一个 http 检测功能，检测 nginx-ingress-controll 健康状态的，通过每隔一定时间访问 nginx-ingress-controll 的/healthz 页面，如是没有响应就返回 404 之类的错误码。

实战 3：通过 Ingress-nginx 实现灰度发布

Ingress-Nginx 是一个 K8S ingress 工具，支持配置 Ingress Annotations 来实现不同场景下的灰度发布和测试。 Nginx Annotations 支持以下几种 Canary 规则：

假设我们现在部署了两个版本的服务，老版本和 canary 版本

nginx.ingress.kubernetes.io/canary-by-header：基于 Request Header 的流量切分，适用于

灰度发布以及 A/B 测试。当 Request Header 设置为 always 时，请求将会被一直发送到 Canary 版本；当 Request Header 设置为 never 时，请求不会被发送到 Canary 入口。

nginx.ingress.kubernetes.io/canary-by-header-value：要匹配的 Request Header 的值，用于通知 Ingress 将请求路由到 Canary Ingress 中指定的服务。当 Request Header 设置为此值时，它将被路由到 Canary 入口。

nginx.ingress.kubernetes.io/canary-weight：基于服务权重的流量切分，适用于蓝绿部署，权重范围 0 - 100 按百分比将请求路由到 Canary Ingress 中指定的服务。权重为 0 意味着该金丝雀规则不会向 Canary 入口的服务发送任何请求。权重为 60 意味着 60%流量转到 canary。权重为 100 意味着所有请求都将被发送到 Canary 入口。

nginx.ingress.kubernetes.io/canary-by-cookie：基于 Cookie 的流量切分，适用于灰度发布与 A/B 测试。用于通知 Ingress 将请求路由到 Canary Ingress 中指定的服务的 cookie。当 cookie 值设置为 always 时，它将被路由到 Canary 入口；当 cookie 值设置为 never 时，请求不会被发送到 Canary 入口。

部署两个版本的服务
这里以简单的 nginx 为例，先部署一个 v1 版本:

```
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")
                ';
            }
        }
    }
---
apiVersion: v1
kind: Service
metadata:
  name: nginx-v1
spec:
  type: ClusterIP
  ports:
  - port: 80
```

```
    protocol: TCP
    name: http
  selector:
    app: nginx
    version: v1


再部署一个 v2 版本:
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")
                ';
            }
        }
    }
---
apiVersion: v1
kind: Service
metadata:
  name: nginx-v2
spec:
  type: ClusterIP
  ports:
  - port: 80
    protocol: TCP
    name: http
  selector:
    app: nginx
    version: v2
```

再创建一个 Ingress，对外暴露服务，指向 v1 版本的服务:

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: nginx
  annotations:
```

```
      kubernetes.io/ingress.class: nginx
  spec:
    rules:
    - host: canary.example.com
      http:
        paths:
        - backend:
            serviceName: nginx-v1
            servicePort: 80
          path: /
```

访问验证一下：

$ curl -H "Host: canary.example.com" http://EXTERNAL-IP # EXTERNAL-IP 替换为 Nginx Ingress 自身对外暴露的 IP

nginx-v1

基于 Header 的流量切分

创建 Canary Ingress，指定 v2 版本的后端服务，且加上一些 annotation，实现仅将带有名为 Region 且值为 cd 或 sz 的请求头的请求转发给当前 Canary Ingress，模拟灰度新版本给成都和深圳地域的用户：

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  annotations:
    kubernetes.io/ingress.class: nginx
    nginx.ingress.kubernetes.io/canary: "true"
    nginx.ingress.kubernetes.io/canary-by-header: "Region"
    nginx.ingress.kubernetes.io/canary-by-header-pattern: "cd|sz"
  name: nginx-canary
spec:
  rules:
  - host: canary.example.com
    http:
      paths:
      - backend:
          serviceName: nginx-v2
          servicePort: 80
        path: /
```

测试访问：

$ curl -H "Host: canary.example.com" -H "Region: cd" http://EXTERNAL-IP # EXTERNAL-IP 替换为 Nginx Ingress 自身对外暴露的 IP

```
nginx-v2
$ curl -H "Host: canary.example.com" -H "Region: bj" http://EXTERNAL-IP
nginx-v1
$ curl -H "Host: canary.example.com" -H "Region: cd" http://EXTERNAL-IP
nginx-v2
$ curl -H "Host: canary.example.com" http://EXTERNAL-IP
nginx-v1
```

可以看到，只有 header Region 为 cd 或 sz 的请求才由 v2 版本服务响应。

基于 Cookie 的流量切分

与前面 Header 类似，不过使用 Cookie 就无法自定义 value 了，这里以模拟灰度成都地域用户为例，仅将带有名为 user_from_cd 的 cookie 的请求转发给当前 Canary Ingress 。先删除前面基于 Header 的流量切分的 Canary Ingress，然后创建下面新的 Canary Ingress:

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  annotations:
    kubernetes.io/ingress.class: nginx
    nginx.ingress.kubernetes.io/canary: "true"
    nginx.ingress.kubernetes.io/canary-by-cookie: "user_from_cd"
  name: nginx-canary
spec:
  rules:
  - host: canary.example.com
    http:
      paths:
      - backend:
          serviceName: nginx-v2
          servicePort: 80
        path: /
```

测试访问:

```
$ curl -s -H "Host: canary.example.com" --cookie "user_from_cd=always" http://EXTERNAL-IP # EXTERNAL-IP 替换为 Nginx Ingress 自身对外暴露的 IP
nginx-v2
$ curl -s -H "Host: canary.example.com" --cookie "user_from_bj=always" http://EXTERNAL-IP
nginx-v1
$ curl -s -H "Host: canary.example.com" http://EXTERNAL-IP
nginx-v1
```

可以看到，只有 cookie user_from_cd 为 always 的请求才由 v2 版本的服务响应。

基于服务权重的流量切分

基于服务权重的 Canary Ingress 就简单了，直接定义需要导入的流量比例，这里以导入 10% 流量到 v2 版本为例（如果有，先删除之前的 Canary Ingress):

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  annotations:
    kubernetes.io/ingress.class: nginx
    nginx.ingress.kubernetes.io/canary: "true"
    nginx.ingress.kubernetes.io/canary-weight: "10"
  name: nginx-canary
spec:
  rules:
  - host: canary.example.com
    http:
      paths:
      - backend:
          serviceName: nginx-v2
          servicePort: 80
        path: /
```

测试访问:

```
$ for i in {1..10}; do curl -H "Host: canary.example.com" http://EXTERNAL-IP; done;
nginx-v1
nginx-v1
nginx-v1
nginx-v1
nginx-v1
nginx-v1
nginx-v2
nginx-v1
nginx-v1
nginx-v1
```

可以看到，大概只有十分之一的几率由 v2 版本的服务响应，符合 10% 服务权重的设置

总结:

4.1 生产环境如何实现蓝绿部署

实战 1：通过 k8s 实现线上业务的蓝绿部署

4.2 通过 k8s 实现滚动更新：滚动更新流程和策略

4.3 通过 k8s 完成线上业务的金丝雀发布

实战 2：七层调度器 Ingress Controller 安装和配置

实战 3：通过 Ingress-nginx 实现灰度发布