Kubernetes 应用更新策略

前言：

课程名称：Kubernetes 应用更新策略

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：6GB 开启虚拟化

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

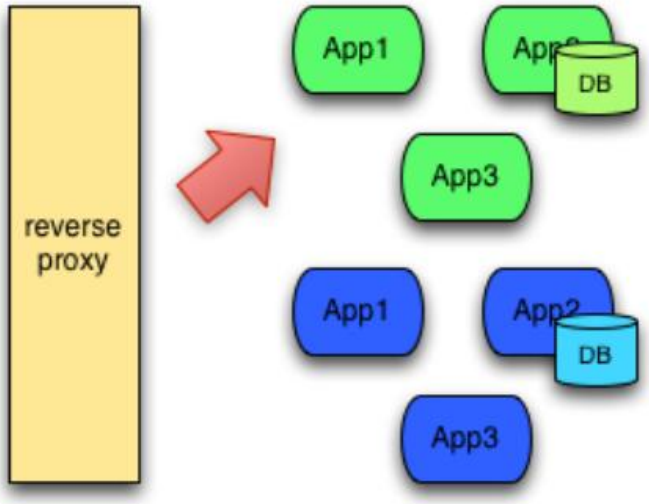
二维码：

1、生产环境如何实现蓝绿部署？

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。

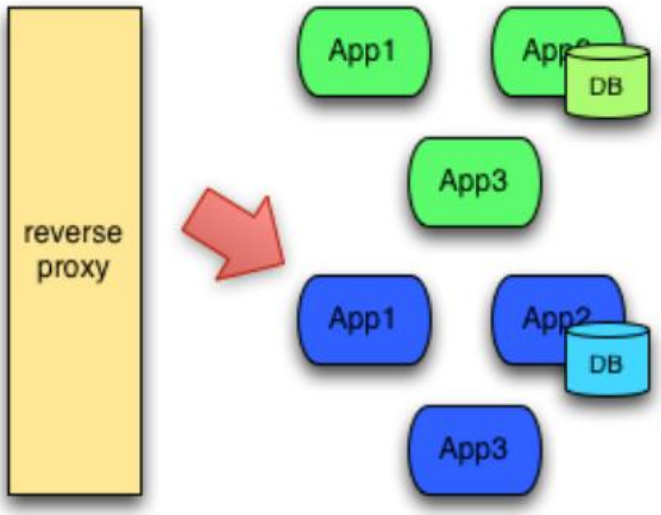
开发新版本，要用新版本替换线上的旧版本，在线上的系统之外，搭建了一个使用新版本代码的全新系统。这时候，一共有两套系统在运行，正在对外提供服务的老系统是绿色系统，新部署的系统是蓝色系统。



蓝色系统不对外提供服务，用来做什么呢？

蓝色系统用来做发布前测试，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。

1.2、蓝绿部署的优势和缺点

优势：

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。

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。

2、通过 k8s 实现线上业务蓝绿部署

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

配置思路：

1、创建一套绿色系统，使用 deployment 进行部署，标签为 app: myapp 和 version: v2。

2、创建 svc，对标签进行精确匹配，匹配 app: myapp 和 version: v2。这样流量走向是绿色系统。

3、部署一套蓝色系统，使用 deployment 进行部署，标签为 app: myapp 和 version: v1。

4、修改 svc，对标签进行精确匹配，匹配 app: myapp 和 version: v1。这样流量走向就是蓝色系统。

操作如下：

（1）创建命名空间

```
[root@k8s-master01 ~]# kubectl create ns blue-green
namespace/blue-green created
```

（2）创建绿色系统

```
[root@k8s-master01 ~]# vi green.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: green
  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

[root@k8s-master01 ~]# kubectl apply -f green.yaml
deployment.apps/green created
```

```
[root@k8s-master01 ~]# kubectl get pods -n blue-green
NAME                     READY   STATUS    RESTARTS   AGE
green-7686d8b4d6-4tpbt   1/1     Running   0          40s
green-7686d8b4d6-ghx9k   1/1     Running   0          40s
green-7686d8b4d6-xvrk9   1/1     Running   0          40s
```

（3）创建前端 service

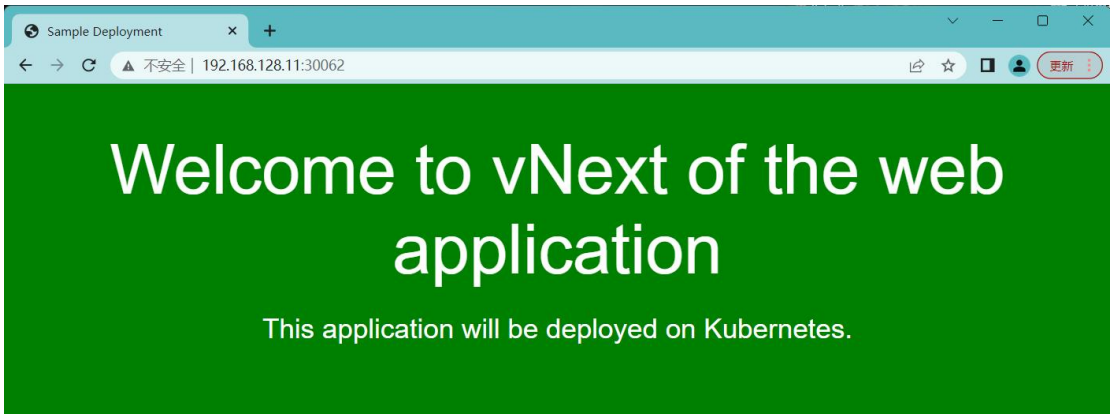
```
[root@k8s-master01 ~]# vi myapp-svc.yaml
apiVersion: v1
kind: Service
metadata:
  name: myapp-green
  namespace: blue-green
spec:
  type: NodePort
  ports:
  - port: 80
    nodePort: 30062
    name: http
  selector:
    app: myapp
    version: v2

[root@k8s-master01 ~]# kubectl apply -f myapp-svc.yaml
service/myapp-green created

[root@k8s-master01 ~]# kubectl get svc -n blue-green
```

```
[root@k8s-master01 ~]# kubectl get svc -n blue-green
NAME          TYPE       CLUSTER-IP     EXTERNAL-IP   PORT(S)        AGE
myapp-green   NodePort   10.10.74.163   <none>        80:30062/TCP   27s
```

（4）访问业务系统

浏览器访问：http://192.168.128.11:30062/

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。



（5）另开一个终端，一直访问我们的业务，测试切换系统的时候，会不会出现业务中断的情况

```
    [root@k8s-master01 ~]# while true; do curl -sI -w "%{http_code}\n" -o /dev/null http://192.168.128.11:30062; sleep 2 ;done
```

```
[root@k8s-master01 ~]# while true; do curl -sI -w "%{http_code}\n" -o /dev/null http://192.168.128.11:30062; sleep 2 ;done
200
200
200
```

可以看到现在状态码一直是 200，说明访问是正常的。

（6）部署蓝色系统

```
    # 绿色系统的标签是 app: myapp 和 version: v2，这里蓝色系统标签为：app: myapp 和 version: v1
    [root@k8s-master01 ~]# vi blue.yaml
    apiVersion: apps/v1
    kind: Deployment
    metadata:
      name: blue
      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

    [root@k8s-master01 ~]# kubectl apply -f blue.yaml
    deployment.apps/blue created

    # 查看蓝色系统是否启动
    [root@k8s-master01 ~]# kubectl get pods -n blue-green -o wide -l
version=v1
```

```
[root@k8s-master01 ~]# kubectl get pods -n blue-green -o wide -l version=v1
NAME                    READY   STATUS    RESTARTS   AGE     IP              NODE         NOMINATED NODE   READINESS GATES
blue-5557789c4d-9782g   1/1     Running   0          3m38s   10.244.58.196   k8s-node02   <none>           <none>
blue-5557789c4d-wn9s4   1/1     Running   0          3m38s   10.244.58.195   k8s-node02   <none>           <none>
blue-5557789c4d-wxhq4   1/1     Running   0          3m38s   10.244.85.197   k8s-node01   <none>           <none>
```

```
    # 检查蓝色系统访问是否正常
    [root@k8s-master01 ~]# curl 10.244.58.196
```

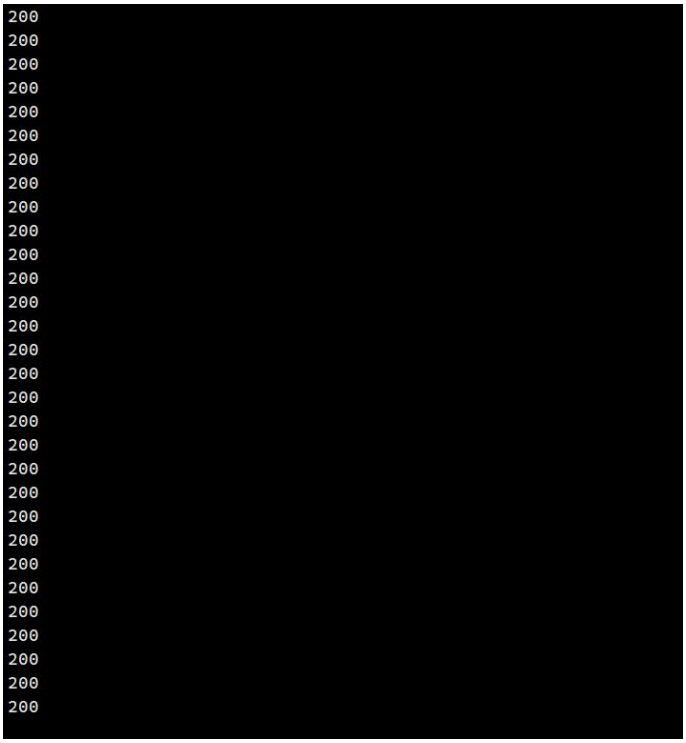
```
[root@k8s-master01 ~]# curl 10.244.58.196
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>Sample Deployment</title>
  <style>
    body {
      color: #ffffff;
      background-color: blue;
      font-family: Arial, sans-serif;
      font-size: 14px;
    }
```

（6）修改 myapp-svc.yaml 配置文件，修改标签，让其匹配到蓝程序（升级之后的程序）

```
    [root@k8s-master01 ~]# vi myapp-svc.yaml
    apiVersion: v1
    kind: Service
    metadata:
      name: myapp-green
      namespace: blue-green
    spec:
      type: NodePort
      ports:
      - port: 80
```

```
    nodePort: 30062
    name: http
  selector:
    app: myapp
    version: v1

# 更新 svc
[root@k8s-master01 ~]# kubectl apply -f myapp-svc.yaml
service/myapp-green configured
```

（7）立马检查运行 while 循环访问业务的终端，查看是否有业务中断现象



可以看到现在状态码一直是 200，我们在做业务迭代的时候，可以做到用户无感知。

（8）访问业务

浏览器访问：http://192.168.128.11:30062/

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。

可以看到现在已经是蓝色系统了。

3、通过 k8s 实现线上业务滚动更新

3.1、滚动更新简介

当 kubernetes 集群中的某个服务需要升级时，传统的做法是，先将要更新的服务下线，业务停止后再更新版本和配置，然后重新启动并提供服务。如果业务集群规模较大时，这个工作就变成了一个挑战，而且先全部了停止，再逐步升级的方式会导致服务较长时间不可用。这个过程面临一个问题，就是在某段时间内，服务是不可用的，对于用户来说是非常不友好的。

kubernetes 提供了滚动更新（rolling-update）的方式来解决上述问题。

简单来说，滚动更新就是针对多实例服务的一种不中断服务的更新升级方式。一般情况下，对于多实例服务，滚动更新采用对各个实例逐个进行单独更新而非同一时刻对所有实例进行全部更新的方式。

对于 k8s 集群部署的 service 来说，rolling update 就是指一次仅更新一个 pod，并逐个进行更新，而不是在同一时刻将该 service 下面的所有 pod shutdown，避免业务中断，从而达到平滑升级的效果。

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。

3.2、滚动更新参数说明

- 配置参数

```
spec:
```

```
    strategy:
      type: RollingUpdate
      rollingUpdate:
        maxSurge: 1
        maxUnavailable: 1
```

字段含义：

deployment.spec.strategy：更新策略

deployment.spec.strategy.type：设置更新策略。有两个可选项：recreate 和 RollingUpdate（默认）。Recreate 是重建式更新，全部删除，然后重新创建。RollingUpdate 表示滚动更新。

deployment.spec.strategy.rollingUpdate：配置滚动更新规则

deployment.spec.strategy.rollingUpdate.maxSurge：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。

deployment.spec.strategy.rollingUpdate.maxUnavailable：升级过程中最多允许有多少个 POD 处于不可用状态。maxUnavailable =1 表示升级过程中最多会有一个 pod 可以处于无法服务的状态,在这里就是至少要有 5-1 个 pod 正常。

扩展：

deployment.spec.revisionHistoryLimit：保留的历史版本数，默认是 10 个。

deployment.spec.minReadySeconds：表示 pod 启动后经过多少秒 pod 才开始对外提供服务。

3.3、在 k8s 中实现滚动更新

（1）环境准备

```
1、创建 deployment yaml 文件
    [root@k8s-master01 ~]# vi test.yaml
    apiVersion: apps/v1
    kind: Deployment
    metadata:
      name: test
    spec:
      replicas: 2
      selector:
        matchLabels:
          app: myapp
      template:
        metadata:
          labels:
```

```
        app: myapp
    spec:
      containers:
      - name: myapp
        image: janakiramm/myapp:v1
        imagePullPolicy: IfNotPresent
        ports:
        - containerPort: 80
```

2、创建 deployment

```
[root@k8s-master01 ~]# kubectl apply -f test.yaml
deployment.apps/test created
```

2、查看 deployment

```
[root@k8s-master01 ~]# kubectl get deploy
NAME   READY   UP-TO-DATE   AVAILABLE   AGE
test   2/2     2            2           114s
```

3、查看 ReplicaSet

```
[root@k8s-master01 ~]# kubectl get rs
NAME              DESIRED   CURRENT   READY   AGE
test-6976cfb7b7   2         2         2       118s
```

创建 deploy 的时候也会创建一个 rs（replicaset），6976cfb7b7 这个随机数字是我们引用 pod 的模板 template 的名字的 hash 值

（2）滚动更新

1、修改 yaml 文件

```
[root@k8s-master01 ~]# vi test.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: test
spec:
  replicas: 2
  revisionHistoryLimit: 10
  strategy:
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 0
  selector:
    matchLabels:
      app: myapp
```

```
      template:
        metadata:
          labels:
            app: myapp
        spec:
          containers:
          - name: myapp
            image: janakiramm/myapp:v2
            imagePullPolicy: IfNotPresent
            ports:
            - containerPort: 80
```

2、另开一个终端执行如下命令

```
[root@k8s-master01 ~]# kubectl get pods -l app=myapp -w
```

3、滚动升级，更新 yaml 文件

```
[root@k8s-master01 ~]# kubectl apply -f test.yaml
deployment.apps/test configured
```

4、在另一个终端查看更新过程

```
^C[root@k8s-master01 ~]# kubectl get pods -l app=myapp -w
NAME                    READY   STATUS    RESTARTS   AGE
test-6976cfb7b7-dxl7b   1/1     Running   0          6s
test-6976cfb7b7-f9vkj   1/1     Running   0          6s
test-77b98748c4-m96s8   0/1     Pending   0          0s
test-77b98748c4-m96s8   0/1     Pending   0          0s
test-77b98748c4-m96s8   0/1     ContainerCreating   0          0s
test-77b98748c4-m96s8   0/1     ContainerCreating   0          1s
test-77b98748c4-m96s8   1/1     Running             0          2s
test-6976cfb7b7-dxl7b   1/1     Terminating         0          40s
test-77b98748c4-ctwfw   0/1     Pending             0          0s
test-77b98748c4-ctwfw   0/1     Pending             0          0s
test-77b98748c4-ctwfw   0/1     ContainerCreating   0          0s
test-6976cfb7b7-dxl7b   1/1     Terminating         0          40s
test-77b98748c4-ctwfw   0/1     ContainerCreating   0          0s
test-6976cfb7b7-dxl7b   0/1     Terminating         0          41s
test-6976cfb7b7-dxl7b   0/1     Terminating         0          41s
test-6976cfb7b7-dxl7b   0/1     Terminating         0          41s
test-77b98748c4-ctwfw   1/1     Running             0          1s
test-6976cfb7b7-f9vkj   1/1     Terminating         0          41s
test-6976cfb7b7-f9vkj   1/1     Terminating         0          41s
test-6976cfb7b7-f9vkj   0/1     Terminating         0          42s
test-6976cfb7b7-f9vkj   0/1     Terminating         0          42s
test-6976cfb7b7-f9vkj   0/1     Terminating         0          42s
```


5、查看 rs

下图可以看出现在业务是 77b98748c4，旧版本是 6976cfb7b7，旧版本的 rs 没有去做删除，这个是用于回滚使用的。

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 rs
NAME              DESIRED   CURRENT   READY   AGE
test-6976cfb7b7   0         0         0       8m43s
test-77b98748c4   2         2         2       8m5s
[root@k8s-master01 ~]# kubectl get pods
NAME                    READY   STATUS    RESTARTS   AGE
test-77b98748c4-ctwfw   1/1     Running   0          8m14s
test-77b98748c4-m96s8   1/1     Running   0          8m16s
```

（3）版本回退

两种方式：

1、再次编辑 yaml 文件，将镜像改成旧版本的进行，更新 yaml。

2、使用 rollout 进行版本回退

这里讲解一下 rollout 进行版本回退，编辑 yaml 回退很简单，这里不做演示。

1、查看历史版本

```
[root@k8s-master01 ~]# kubectl rollout history deployment/test
deployment.apps/test
REVISION  CHANGE-CAUSE
1         <none>
2         <none>
```

提示：VERSION=1 表示最初始的版本。VERSION=2 表示一次升级的版本，依次类推。

2、查看某个版本历史详情

```
[root@k8s-master01 ~]# kubectl rollout history deployment/test --revision=1
```

```
[root@k8s-master01 ~]# kubectl rollout history deployment/test --revision=1
deployment.apps/test with revision #1
Pod Template:
  Labels:       app=myapp
        pod-template-hash=6976cfb7b7
  Containers:
   myapp:
    Image:      janakiramm/myapp:v1
    Port:       80/TCP
    Host Port:  0/TCP
    Environment:        <none>
    Mounts:     <none>
  Volumes:      <none>
```

可以看出使用的镜像是 v1 版本，那这个就是我们需要回退的版本。

3、回退版本

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。

```
    # 回退到上次的版本
    [root@k8s-master01 ~]# kubectl rollout undo deployment/test

    # 回退到指定版本
    [root@k8s-master01 ~]# kubectl rollout undo deployment/test
--to-revision=1
    deployment.apps/test rolled back
```

（4）扩展知识

我们查询历史的时候 CHANGE-CAUSE 内容是空的，我们可以指定 CHANGE-CAUSE 内容，记录执行命令。

```
    [root@k8s-master01 ~]# kubectl rollout history deployment/test
    deployment.apps/test
    REVISION  CHANGE-CAUSE
    2         <none>
    3         <none>

    [root@k8s-master01 ~]# kubectl set image deployment/test
myapp=janakiramm/myapp:v2 --record=true
    Flag --record has been deprecated, --record will be removed in the
future
    deployment.apps/test image updated
    # 这里提示--record 已弃用，将来会被删除。不过没关系，在当前版本还是可以正常使用的。

    [root@k8s-master01 ~]# kubectl rollout history deployment/test
[root@k8s-master01 ~]# kubectl rollout history deployment/test
deployment.apps/test
REVISION  CHANGE-CAUSE
3         <none>
4         kubectl set image deployment/test myapp=janakiramm/myapp:v2 --record=true
```

注意事项

1、更新策略

我们这里使用的是滚动更新 RollingUpdate。如何使用 Recreate，会和传统的升级方法一样，先停掉所有的 pod，然后重建 pod。

2、参数设置

注意点 1：

当 maxSurge 设置为 0 的时候，maxUnavailable 不能设置为 0。

当有 5 个 pod 时，maxSurge 设置为 0，那么先删除后启动 pod。那么这个时候 maxUnavailable 设置为 0，至少要有 5 个 pod 正常，那么就无法进行滚动更

新。

一定要注意两者不能同时为 0。

注意点 2：

当有 10 个副本的话，maxSurge 设置为 5%，5%的话==0.5 个，但计算按照是 1 个。

当有 10 个副本的话，maxUnavailable 设置为 5%，5%的话==0.5 个，但计算是按照 0 个。

当有 10 个副本的话，maxSurge 和 maxUnavailable 都为 1%时，如下图：

```
[root@k8s-master01 ~]# kubectl describe deployment test
Name:                   test
Namespace:              default
CreationTimestamp:      Thu, 12 Jan 2023 01:08:51 -0500
Labels:                 <none>
Annotations:            deployment.kubernetes.io/revision: 2
Selector:               app=myapp
Replicas:               10 desired | 10 updated | 10 total | 10 available | 0 unavailable
StrategyType:           RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy:  1% max unavailable, 1% max surge
Pod Template:
  Labels:  app=myapp
  Containers:
```

maxUnavailable 最大不可用 1%，maxSurge 最大激增 1%。简单的说，maxUnavailable 不足 1，计为 0。maxSurge 不足 1，计为 1。

进行滚动更新，验证如下图所示：

```
[root@k8s-master01 ~]# kubectl get pods -w
NAME                    READY   STATUS    RESTARTS   AGE
test-77b98748c4-5sz57   1/1     Running   0          7s
test-77b98748c4-7st48   1/1     Running   0          7s
test-77b98748c4-8zmk2   1/1     Running   0          7s
test-77b98748c4-hj94v   1/1     Running   0          7s
test-77b98748c4-hmbpc   1/1     Running   0          7s
test-77b98748c4-m629z   1/1     Running   0          7s
test-77b98748c4-mnrxg   1/1     Running   0          7s
test-77b98748c4-qvvnw   1/1     Running   0          7s
test-77b98748c4-ttw7g   1/1     Running   0          7s
test-77b98748c4-zgkv2   1/1     Running   0          7s
test-6976cfb7b7-c62fj   0/1     Pending   0          0s
test-6976cfb7b7-c62fj   0/1     Pending   0          0s
test-6976cfb7b7-c62fj   0/1     ContainerCreating   0          0s
test-6976cfb7b7-c62fj   0/1     ContainerCreating   0          1s
test-6976cfb7b7-c62fj   1/1     Running             0          2s
test-77b98748c4-qvvnw   1/1     Terminating         0          18s
```

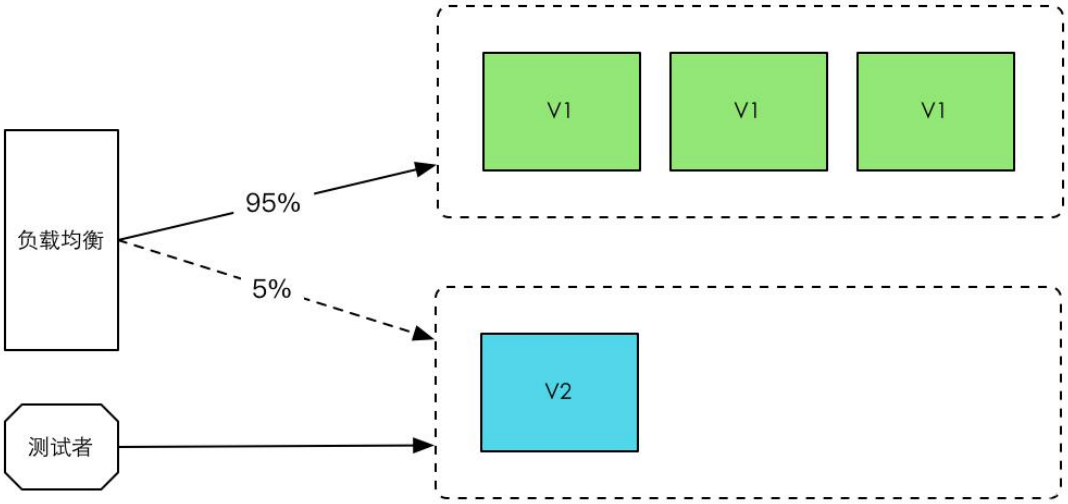
可以看出是先创建 1，再删除 1，一直出现到整个业务更新完毕。

4、通过 k8s 完成线上业务金丝雀发布

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。

4.1、什么是金丝雀发布？

金丝雀发布也叫灰度发布，起源是，矿井工人发现，金丝雀对瓦斯气体很敏感，矿工会在下井之前，先放一只金丝雀到井中，如果金丝雀不叫了，就代表瓦斯浓度高。



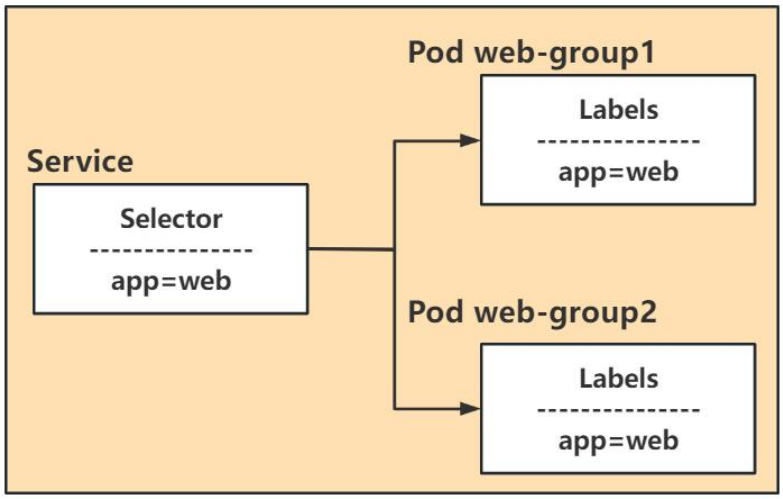
在灰度发布开始后，先启动一个新版本应用，但是并不直接将流量切过来，而是测试人员对新版本进行线上测试，启动的这个新版本应用，就是我们的金丝雀。如果没有问题，那么可以将少量的用户流量导入到新版本上，然后再对新版本做运行状态观察，收集各种运行时数据，如果此时对新旧版本做各种数据对比，就是所谓的 A/B 测试。

当确认新版本运行良好后，再逐步将更多的流量导入到新版本上，在此期间，还可以不断地调整新旧两个版本的运行的服务器副本数量，以使得新版本能够承受越来越大的流量压力。直到将 100%的流量都切换到新版本上，最后关闭剩下的老版本服务，完成灰度发布。

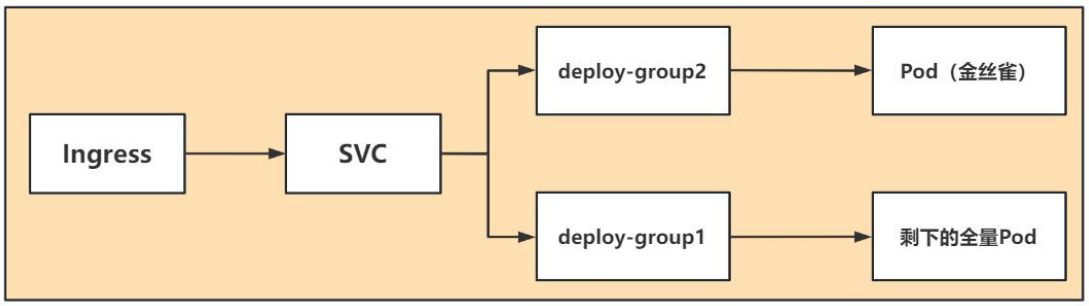
如果在灰度发布过程中（灰度期）发现了新版本有问题，就应该立即将流量切回老版本上，这样，就会将负面影响控制在最小范围内。

4.2、在 k8s 中实现金丝雀发布

service 会通过 selector 里面的值去匹配 pod 里的 lables 值。如下图：



我们存在两组 deployment，一组为正在运行的业务容器，一组为金丝雀 pod，都通过 service Selector 进行标签绑定。具体项目架构结构如下图所示：



其实实现也很简单，下面我们开始操作

（1）创建业务 deploy Pod

```
[root@k8s-master01 ~]# vi deploy-group1.yaml
kind: ConfigMap
apiVersion: v1
metadata:
  name: deploy-group1
data:
  index.html: |-
    Welcome to the deploy-group1 test page
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: deploy-group1
spec:
  replicas: 10
  selector:
```

```
      matchLabels:
       app: web
     template:
      metadata:
       labels:
        app: web
      spec:
       containers:
       - name: web
         image: nginx:latest
         imagePullPolicy: IfNotPresent
         volumeMounts:
         - name: index-html
           mountPath: /usr/share/nginx/html/index.html
           subPath: index.html
       volumes:
         - name: index-html
           configMap:
             name: deploy-group1
             items:
             - key: index.html
               path: index.html

[root@k8s-master01 ~]# kubectl apply -f deploy-group1.yaml
configmap/deploy-group1 created
deployment.apps/deploy-group1 created
```

（2）创建业务 svc

```
[root@k8s-master01 ~]# cat deploy-svc.yaml
apiVersion: v1
kind: Service
metadata:
  name: deploy-svc
spec:
  type: NodePort
  ports:
  - port: 80
    nodePort: 30808
    name: http
  selector:
    app: web
```

```
[root@k8s-master01 ~]# kubectl apply -f deploy-svc.yaml
service/deploy-svc created
```

（3）浏览器访问测试

浏览器访问：http://192.168.128.11:30808/

（4）部署金丝雀

```
[root@k8s-master01 ~]# cat deploy-group2.yaml
kind: ConfigMap
apiVersion: v1
metadata:
  name: deploy-group2
data:
  index.html: |-
    Welcome to the deploy-group2 test page
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: deploy-group2
spec:
  replicas: 1
  selector:
   matchLabels:
    app: web
  template:
   metadata:
    labels:
     app: web
   spec:
    containers:
    - name: web
      image: nginx:latest
      imagePullPolicy: IfNotPresent
      volumeMounts:
      - name: index-html
```

```
            mountPath: /usr/share/nginx/html/index.html
            subPath: index.html
        volumes:
          - name: index-html
            configMap:
              name: deploy-group2
              items:
              - key: index.html
                path: index.html

[root@k8s-master01 ~]# kubectl apply -f deploy-group2.yaml
configmap/deploy-group2 created
deployment.apps/deploy-group2 created
```

这里金丝雀的副本数为 1，业务副本数为 10，比率为 1:10，10 个用户访问，可能会有 1 个用户能访问到金丝雀。

（5）测试

[root@k8s-master01 ~]# for i in `seq 10`; do curl http://192.168.128.11:30808/; echo " "; sleep 2; done

```
[root@k8s-master01 ~]# for i in `seq 10`; do curl http://192.168.128.11:30808/; echo " "; sleep 2; done
Welcome to the deploy-group1 test page
Welcome to the deploy-group2 test page
Welcome to the deploy-group1 test page
Welcome to the deploy-group1 test page
Welcome to the deploy-group1 test page
Welcome to the deploy-group1 test page
Welcome to the deploy-group1 test page
Welcome to the deploy-group1 test page
Welcome to the deploy-group1 test page
Welcome to the deploy-group1 test page
```

上图可以看出访问了 10 次，只有 1 次请求到了金丝雀。

总结：

上面我们是使用 kubernetes 原生的技术实现的金丝雀发布。在生产环境中，要实现金丝雀发布，可以使用 ingress nginx 或 istio 技术来实现。这里只是为了让大家知道什么是金丝雀发布。

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。