前言

在前一章，我们学会如何在 `Kubernetes` 内部署自己的第一个应用。但是在实际应用中，我们还会遇到一些特定场景：

> A 用户是VIP，我怎么才能让VIP用户看到内测版本呢？ 我不想停机，怎么发布新版本呢？ 如何让新版本服务只开放小流量访问呢？ ……

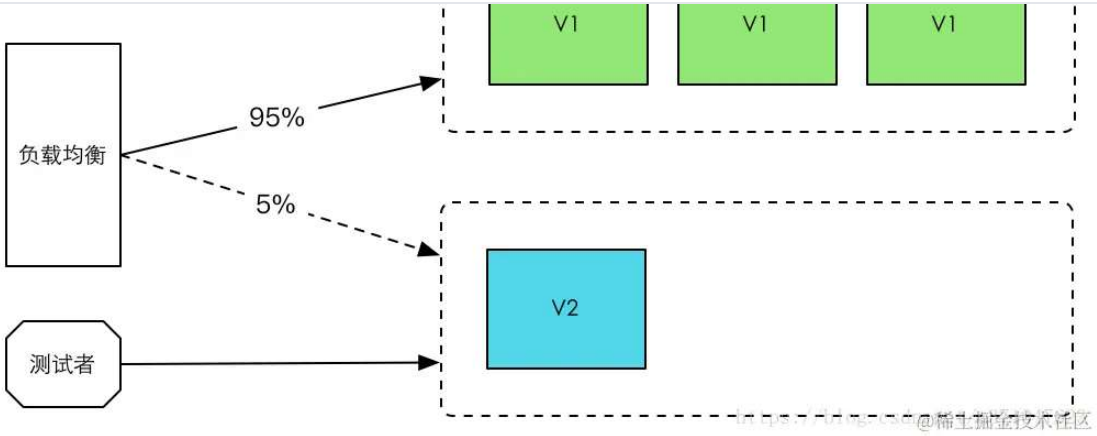
显然，这些场景对于我们单纯的访问来看是无法做到的。那么有什么好办法呢?

什么是灰度发布

首先我们来看**灰度发布**。灰度发布是一种发布方式，也叫 `金丝雀发布` 。**起源是矿工在下井之前会先放一只金丝雀到井里，如果金丝雀不叫了，就代表瓦斯浓度高。原因是金丝雀对瓦斯气体很敏感。**这就是金丝雀发布的又来，非常形象地描述了我们的发布行为。

灰度发布的做法是：会在现存旧应用的基础上，启动一个新版应用。但是新版应用并不会直接让用户访问。而是先让测试同学去进行测试。如果没有问题，则可以将真正的用户流量慢慢导入到新版上。在这中间，持续对新版本运行状态做观察，直到慢慢切换过去，**这就是所谓的A/B测试。** 当然，你也可以招募一些 **灰度用户，** 给他们设置独有的灰度标示（Cookie，Header），来让他们可以访问到新版应用。

当然，如果中间切换出现问题，也应该将流量迅速地切换到老应用上。



实现方案

在上一章节，我们使用 `k8s` 部署了 `ingress` 。这里我们就利用 `ingress annotations` 中的 `canary` 配置项来实现灰度发布逻辑。

1. 准备新版本的 Service

在开始准备灰度之前，需要准备一套新环境以备流量切分。切换到 `deployment` 目录，我们新启动一套 `v2` 的环境配置，在这里可以将原先 `v1` 的配置文件，拷贝一份替换为 `v2` 的镜像：

shell 复制代码

```shell
cd deployment && cp v1.yaml v2.yaml
```

修改 `v2.yaml` ，将 `Deployment Name` ， `Service Name` 和匹配规则都替换为 `v2` ，并将镜像版本替换为 `v2`

yaml 复制代码

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: front-v2
spec:
  selector:
    matchLabels:
      app: nginx-v2
  replicas: 3
  template:
    metadata:
      labels:
```

```
      - name: nginx
        image: registry.cn-hangzhou.aliyuncs.com/janlay/k8s_test:v2
        ports:
        - containerPort: 80
---
apiVersion: v1
kind: Service
metadata:
  name: front-service-v2
spec:
  selector:
    app: nginx-v2
  ports:
  - protocol: TCP
    port: 80
    targetPort: 80
  type: NodePort
```

接着使用 `kubectl apply` 命令使其配置生效：

shell 复制代码

```
kubectl apply -f ./v2.yaml
```

2. 根据不同方案进行切分

根据 Cookie 切分流量

基于 `Cookie` 切分流量。这种实现原理主要根据**用户请求中的 Cookie 是否存在灰度标示 Cookie**去判断是否为灰度用户，再决定是否返回灰度版本服务。

我们新建一个全新的 ingress 配置文件，名称叫 `gary` ：

shell 复制代码

```
cd ./ingress && vim ./gray.yaml
```

输入以下配置：

yaml 复制代码

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: nginx-demo-canary
```

```
    nginx.ingress.kubernetes.io/canary: "true"
    nginx.ingress.kubernetes.io/canary-by-cookie: "users_from_Beijing"
spec:
  rules:
  - http:
      paths:
       - backend:
          serviceName: front-service-v2
          servicePort: 80
  backend:
     serviceName: front-service-v2
     servicePort: 80
```

我们可以看到，在 `annotations` 这里，有两个关于灰度的配置项。分别是：

- nginx.ingress.kubernetes.io/canary：可选值为 `true / false` 。代表是否开启灰度功能
- nginx.ingress.kubernetes.io/canary-by-cookie：灰度发布 `cookie` 的 `key` 。当 `key` 值等于 `always` 时，灰度触发生效。等于其他值时，则不会走灰度环境。

保存后，使用 `kubectl apply` 生效配置文件查看效果：

shell 复制代码

```
kubectl apply -f ./gray.yaml
```

执行成功后，可以使用 `kubectl get svc` 命令来获取 `ingress` 的外部端口：

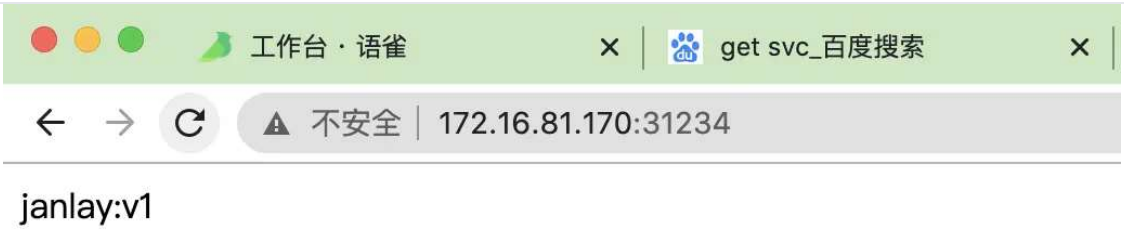
shell 复制代码

```
kubectl -n ingress-nginx get svc
```

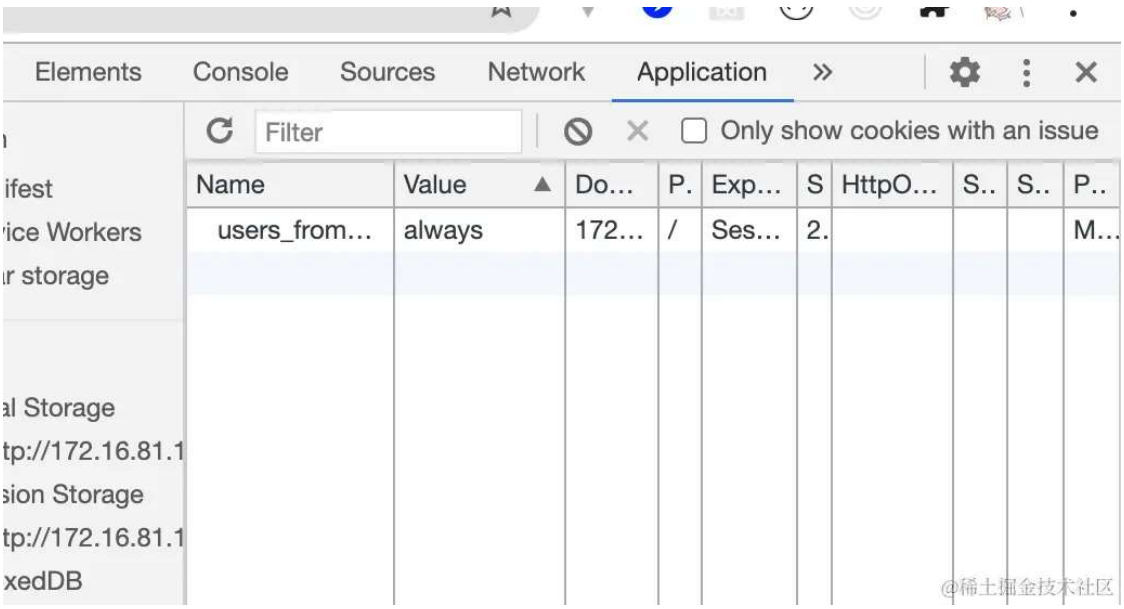
> -n: 根据资源名称进行模糊查询

其中， `PORT` 字段是我们可以访问的外部端口。80为 `ingress` 内部端口，31234 为外部端口。

```
[root@master ingress]# kubectl -n ingress-nginx get svc
NAME                                 TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)                      AGE
ingress-nginx-controller             NodePort    10.97.150.187   <none>        80:31234/TCP,443:31235/TCP   38h
ingress-nginx-controller-admission   ClusterIP   10.101.226.10   <none>        443/TCP                      38h
[root@master ingress]# kubectl get svc
```



接下来，我们在chrome控制台中手动设置一个 cookie。key为 `users_from_Beijing` ，值为 `always`



刷新页面，发现我们访问的是v2。灰度发布环境搭建成功

基于 `Header` 切分流量，这种实现原理主要根据**用户请求中的 header 是否存在灰度标示 header**去判断是否为灰度用户，再决定是否返回灰度版本服务。

当然配置也很简单，只需要修改 `annotations` 配置项即可：

- nginx.ingress.kubernetes.io/canary-by-header：要灰度 `header` 的 `key` 值
- nginx.ingress.kubernetes.io/canary-by-header-value: 要灰度 `header` 的 `value` 值

保存后，使用 `kubectl apply` 生效配置文件：

shell 复制代码

```shell
kubectl apply -f ./gray.yaml
```

如何查看效果呢？我们可以使用 `curl` 命令来加header头去请求访问调用：

yaml 复制代码

```yaml
curl --header 'header的key:header的value' 127.0.0.1:端口值
```

由于我这里配置的灰度 `header` 为 `janlay` ， `value` 为 `isme` ，所以如以下结果：

通过对比发现，当 `janlay` 不是 `isme` 时，灰度失效。验证成功

```
[root@master ingress]# curl --header 'janlay:isme' 127.0.0.1:31234
janlay:v422323232
[root@master ingress]# curl --header 'janlay:isme22' 127.0.0.1:31234
janlay:v1
[root@master ingress]#
```


基于权重切分流量

这种实现原理主要是根据用户请求，通过根据灰度百分比决定是否转发到灰度服务环境中

在这里，我们修改 `annotations` 配置项即可：

值 = `100` 时，代表全走灰度。

保存后，使用 `kubectl apply` 生效配置文件：

shell 复制代码

```shell
kubectl apply -f ./gray.yaml
```

我们用shell脚本语言写个轮询，循环10次调用服务，看灰度命中概率：

shell 复制代码

```shell
for ((i=1; i<=10; i++)); do curl 127.0.0.1:端口值;echo; done
```

```
[root@master ~]# for ((i=1; i<=10; i++)); do curl 127.0.0.1:31234;echo; done
janlay:v422323232

janlay:v1

janlay:v1

janlay:v1

janlay:v1

janlay:v422323232

janlay:v1

janlay:v422323232

janlay:v422323232

janlay:v422323232

[root@master ~]#
```

@稀土掘金技术社区

通过轮询10次发现，其命中概率大概在 `4-6` 次左右。这个命中概率只是相对于单词请求而言，拥有50%的概率。所以批量执行存在误差是正常的。

注意事项：优先级

上面的三种灰度方案都了解完后，如果同时配置三种方案，那么他们在 `ingress` 中的优先级是怎样的？ 在官方文档的后面有一个 `Note` 提示，上面明确有写：

`k8s` 会优先去匹配 `header` ，如果未匹配则去匹配 `cookie` ，最后是 `weight`

总结

这里我们学习了灰度发布的配置方式和其3种模式：基于 `cookie` 切分，基于 `header` 切分，基于权重概率切分。

接下来我们会学习基于 `deployment` 的滚动发布模式，讲解如何不宕机平滑发布服务

参考链接

- kubernetes ingress 官方文档：kubernetes.github.io/ingress-ngi...

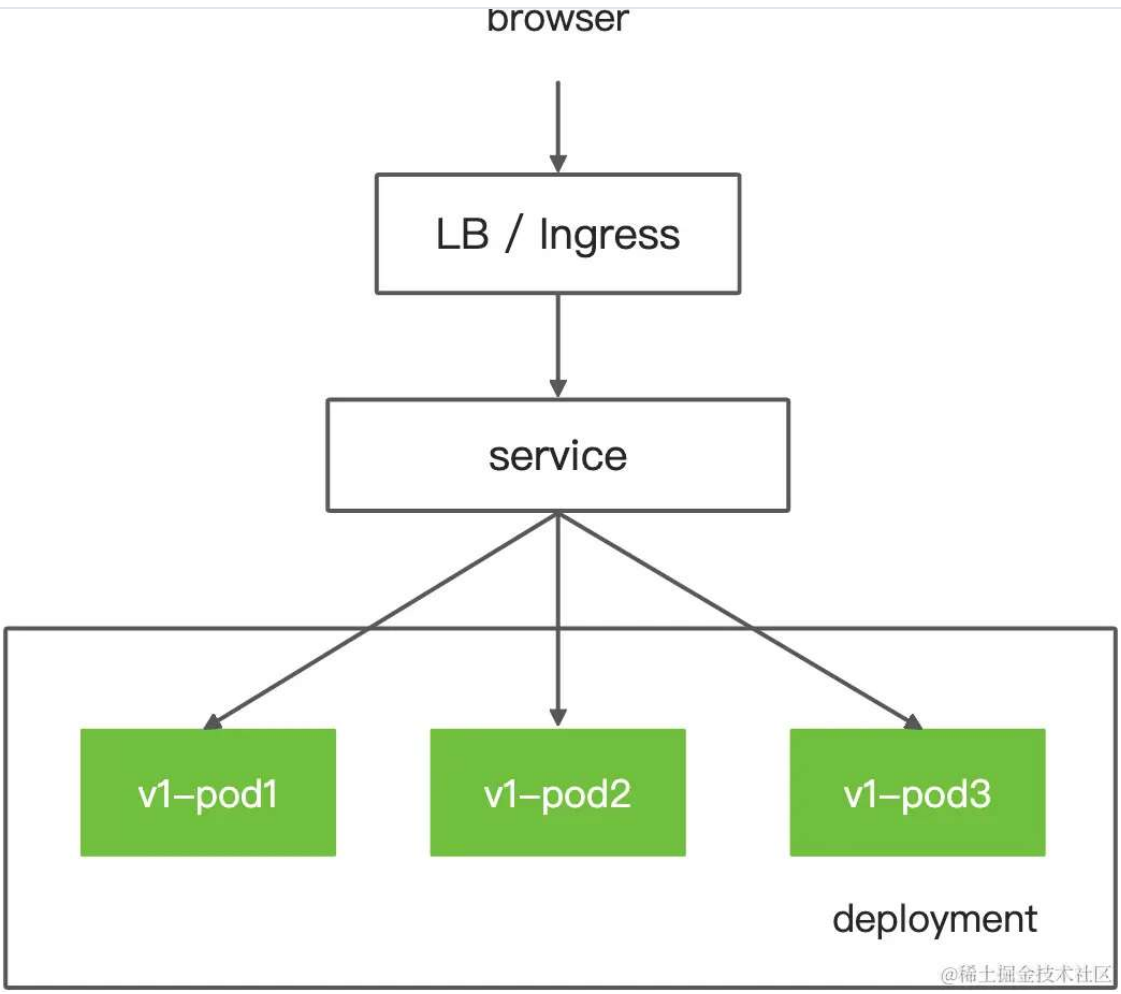
什么是滚动发布

滚动发布，则是我们一般所说的无宕机发布。其发布方式如同名称一样，一次取出一台/多台服务器（看策略配置）进行新版本更新。当取出的服务器新版确保无问题后，接着采用同等方式更新后面的服务器。

发布流程和策略

就绪状态

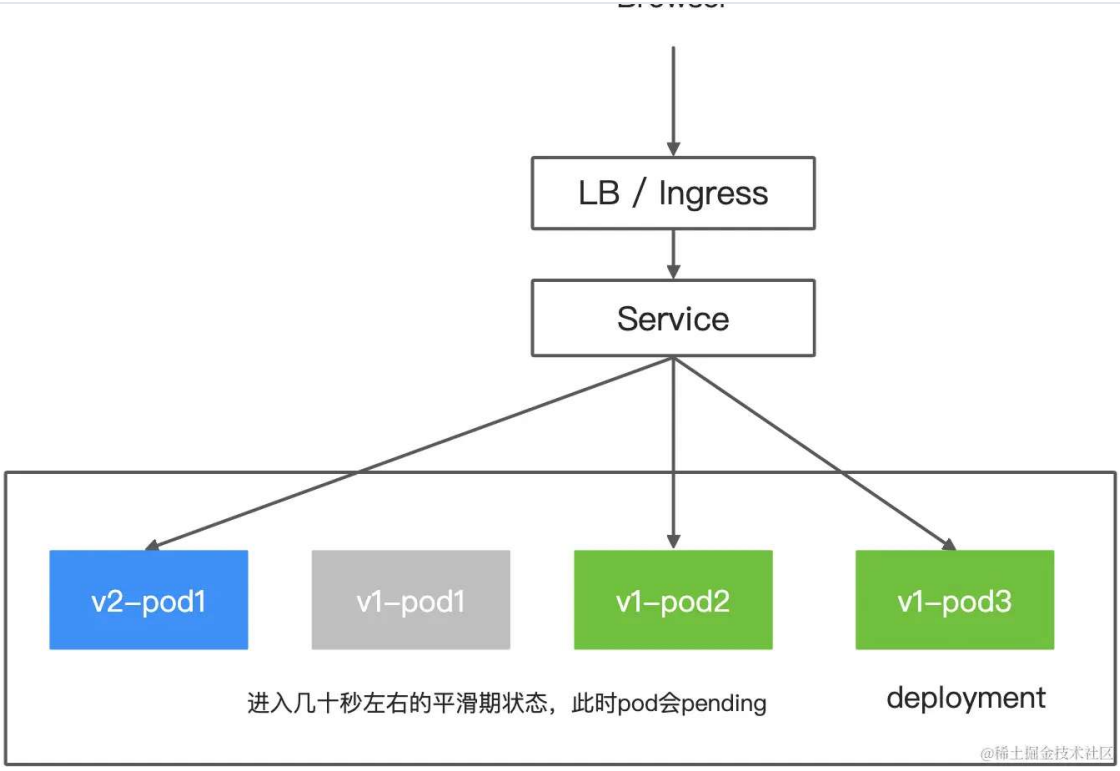
第一步，我们准备一组服务器。这组服务器当前服务的版本是 V1。

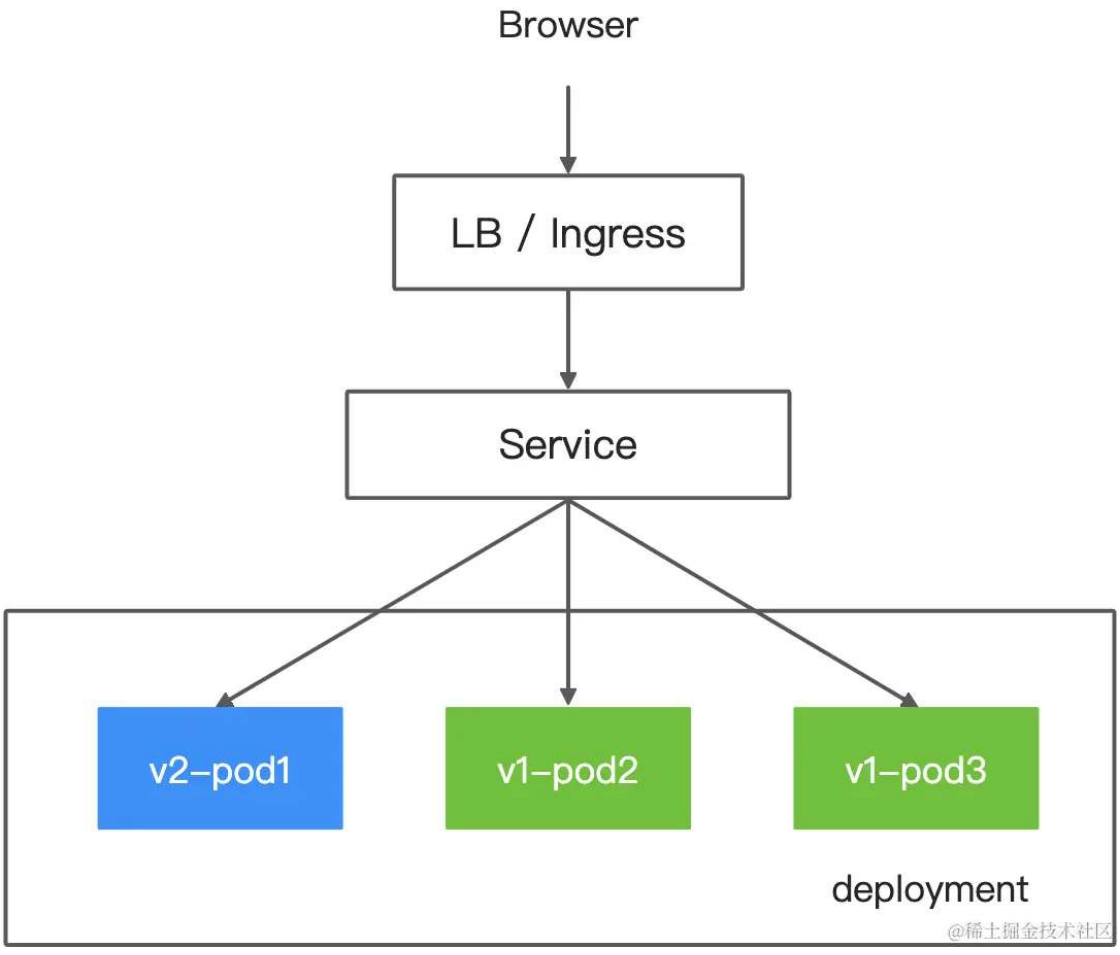


升级第一个 Pod

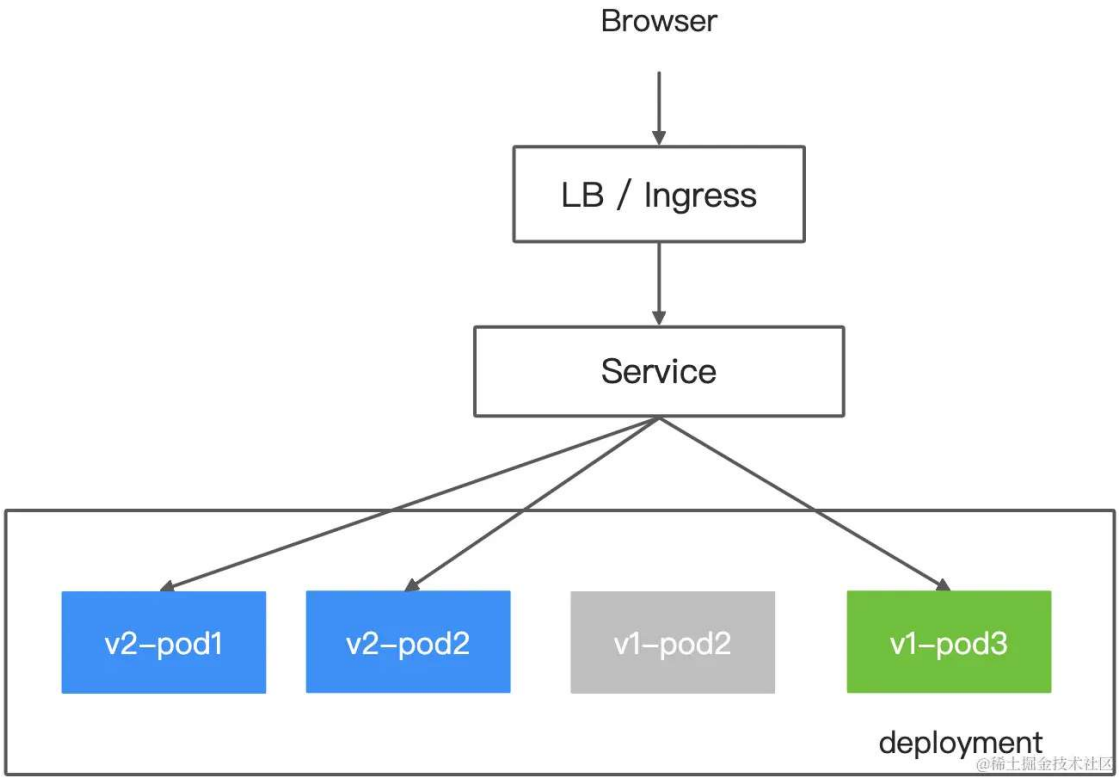
第二步开始升级。

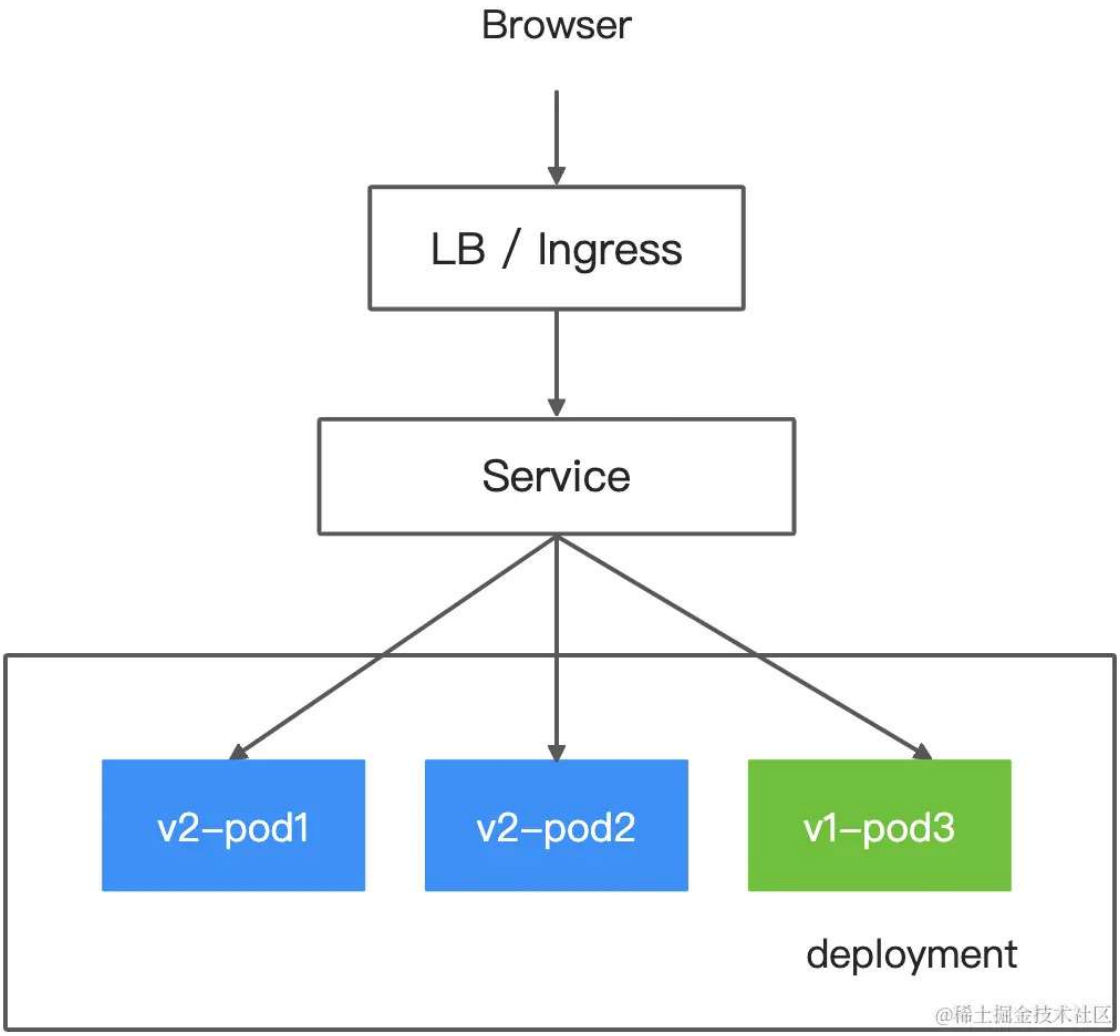
首先，会增加一个 V2 版本的 Pod1 上来，**将 V1 版本的 Pod1 下线但不移除。此时，V1版本的 Pod1 将不会接受流量进来，而是进入一个平滑期等待状态（大约几十秒）后才会被杀掉。**

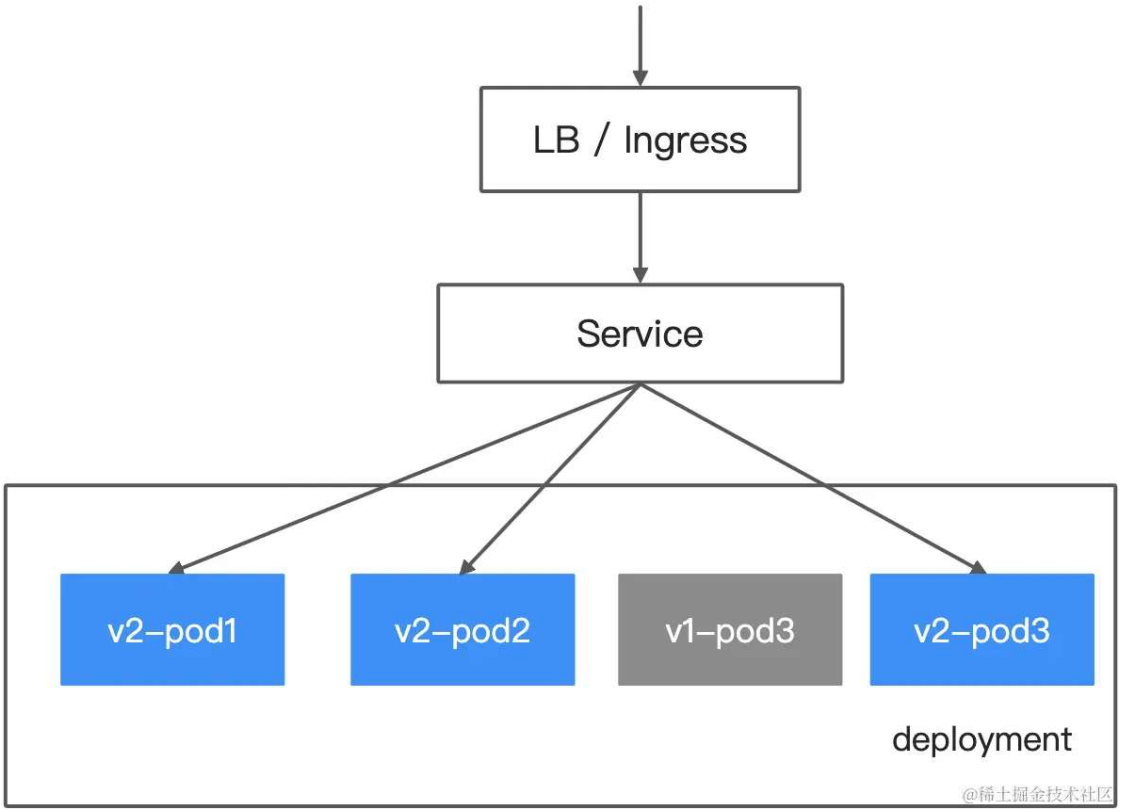


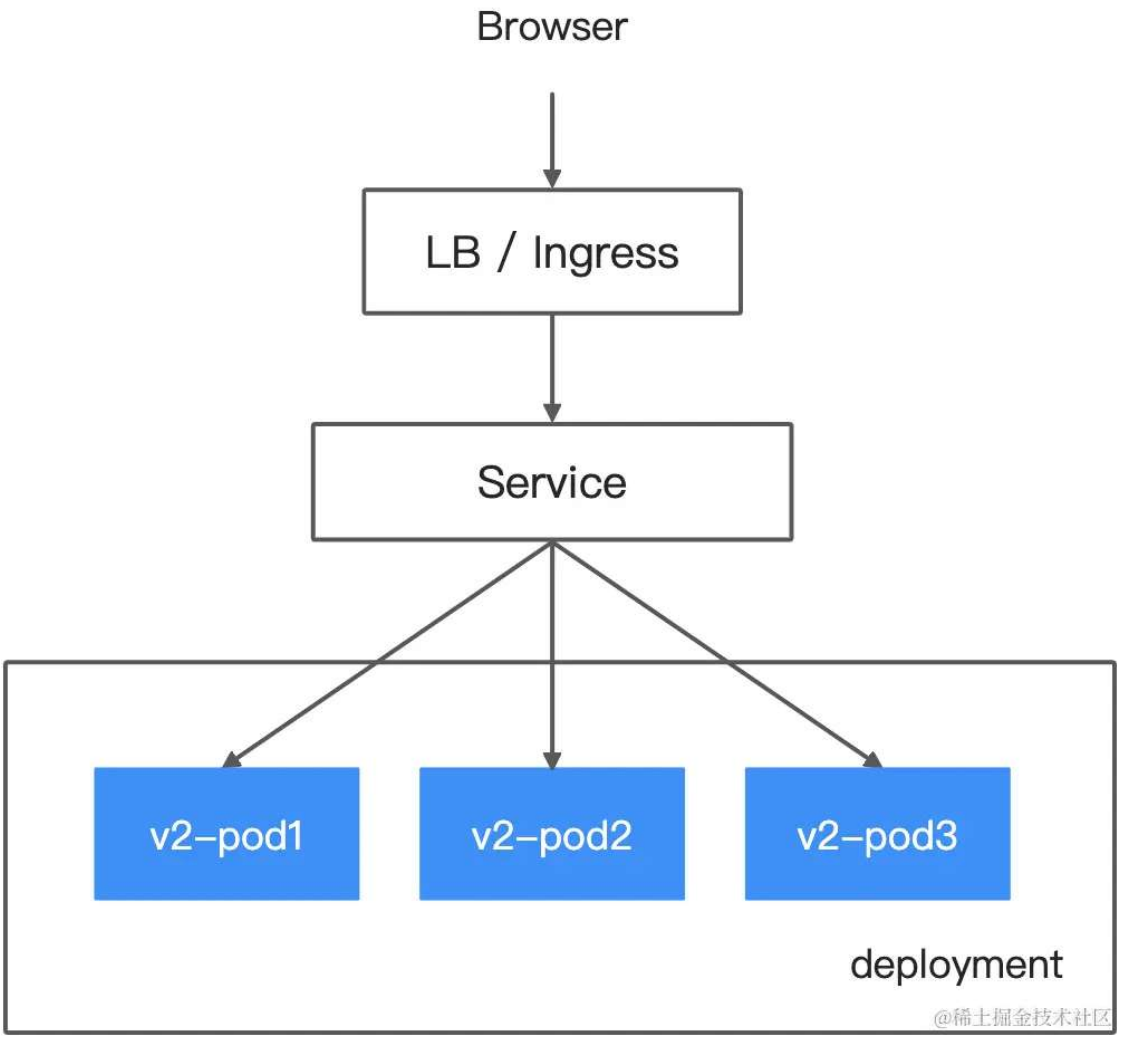


升级剩下的 Pod









滚动发布作为众多发布类型的一种，必然也存在着一些优点和缺点：

优点

1. 不需要停机更新，无感知平滑更新。
2. 版本更新成本小。不需要新旧版本共存

缺点

1. 更新时间长：每次只更新一个/多个镜像，需要频繁连续等待服务启动缓冲（详见下方平滑期介绍）
2. 旧版本环境无法得到备份：始终只有一个环境存在
3. 回滚版本异常痛苦：如果滚动发布到一半出了问题，回滚时需要使用同样的滚动策略回滚旧版本。

Kubernetes 中的滚动发布

在 `Kubernetes` 的 `ReplicaSet` 中，默认就是滚动发布镜像。我们只需要通过简单的配置即可调整滚动发布策略

编辑 `deployment` 文件：

```shell
1  vim ./v2.yaml
```

```yaml
  name: front-v2
spec:
  minReadySeconds: 1
  strategy:
  # indicate which strategy we want for rolling update
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 0
  selector:
    matchLabels:
      app: nginx-v2
  replicas: 10
  template:
    metadata:
      labels:
```


yaml 复制代码

```yaml
minReadySeconds: 1
  replicas: 10
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 0
```

字段的含义分别为：

名称	含义
minReadySeconds	容器接受流量延缓时间：单位为秒，默认为0。如果没有设置的话，k8s会认为容器启动成功后就可以用了。设置该值可以延缓容器流量切分
strategy.type = RollingUpdate	ReplicaSet 发布类型，声明为滚动发布，默认也为滚动发布
strategy.rollingUpdate.maxSurge	最多Pod数量：为数字类型/百分比。如果 maxSurge 设置为1，replicas 设置为10，则在发布过程中pod数量最多为10 + 1个（多出来的为旧版本pod，平滑期不可用状态）。maxUnavailable 为 0 时，该值也不能设置为0
strategy.rollingUpdate.maxUnavailable	升级中最多不可用pod的数量：为数字类型/百分比。当 maxSurge 为 0 时，该值也不能设置为0。

接着使 Kubernetes 生效配置。配置生效后立即继续发布动作，随后监听查看发布状态更改：

```shell
kubectl apply -f ./v2.yaml && kubectl rollout status deployment/front-v2
```

```
[root@master deployment]# kubectl rollout status deployment/front-v2
Waiting for deployment "front-v2" rollout to finish: 1 out of 3 new replicas have been updated...
Waiting for deployment "front-v2" rollout to finish: 1 out of 3 new replicas have been updated...
Waiting for deployment "front-v2" rollout to finish: 2 out of 3 new replicas have been updated...
Waiting for deployment "front-v2" rollout to finish: 2 out of 3 new replicas have been updated...
Waiting for deployment "front-v2" rollout to finish: 2 out of 3 new replicas have been updated...
Waiting for deployment "front-v2" rollout to finish: 2 out of 3 new replicas have been updated...
Waiting for deployment "front-v2" rollout to finish: 1 old replicas are pending termination...
Waiting for deployment "front-v2" rollout to finish: 1 old replicas are pending termination...
Waiting for deployment "front-v2" rollout to finish: 1 old replicas are pending termination...
deployment "front-v2" successfully rolled out
```


我们通过日志可以看到，3个 replicas 的更新逻辑逻辑为：**单个逐个地去进行更新 Pod。而不是一次性将旧的Pod 全部杀死后，再启动新的 Pod。**

通过简单的配置，我们即可在k8s中实现滚动发布。

另一种发布模式

既然 k8s 的默认发布方式就是滚动发布，那么有没有其他的更新方式?

在 Kubernetes 中，有一种发布方式为 Recreate 。这种发布方式比较暴力，它会直接把所有旧的 Pod 全部杀死。杀死后再批量创建新的 Pod 。

我们只需要将 strategy.type 改为 Recreate 即可：

```shell
vim ./v2.yaml
# type: Recreate
```

接着更新 deployment 并查看发布状态：

```shell
kubectl apply -f ./v2.yaml && kubectl rollout status deployment/front-v2
```

```
Waiting for deployment "front-v2" rollout to finish: 0 out of 3 new replicas have been updated...
Waiting for deployment "front-v2" rollout to finish: 0 of 3 updated replicas are available...
Waiting for deployment "front-v2" rollout to finish: 0 of 3 updated replicas are available...
Waiting for deployment "front-v2" rollout to finish: 0 of 3 updated replicas are available...
Waiting for deployment "front-v2" rollout to finish: 0 of 3 updated replicas are available...
deployment "front-v2" successfully rolled out
```


我们看到， `k8s` 会将所有旧的 `Pod` 杀死，随后再批量启动新的 `Pod` 。

这种发布方式相对滚动发布偏暴力。**且在发布空窗期（杀死旧Pod，新Pod还没创建成功的情况下）服务会不可用。**

接着我们再来介绍下上面提到的 `kubectl rollout` 命令：

kubectl rollout

`kubectl rollout` 命令可以用来操纵 `deployment` 的资源进行管理。包括对版本的快速回退，暂停/恢复版本更新，根据更新历史回退版本等功能。

例如暂停一个 `deployment` 的发布：

shell 复制代码

```shell
kubectl rollout pause deployment/名称
```

继续一个 `deployment` 的发布：

shell 复制代码

```shell
kubectl rollout resume deployment/名称
```

查看一个 `deployment` 的发布状态：

shell 复制代码

```shell
kubectl rollout status deployment/名称
```

结束语

在这一章，我们通过 `Kuberentes` 实现了灰度发布/滚动发布环境，基本上可以满足需求。在下一章，我们会给大家带来新东西 —— 健康度检查。可以让你更好地管理服务状态，控制服务的运行行为