# 灰度发布¶

在日常工作中我们经常需要对服务进行版本更新升级，所以我们经常会使用到滚动升级、蓝绿发布、灰度发布等不同的发布操作。而 `ingress-nginx` 支持通过 Annotations 配置来实现不同场景下的灰度发布和测试，可以满足金丝雀发布、蓝绿部署与 A/B 测试等业务场景。

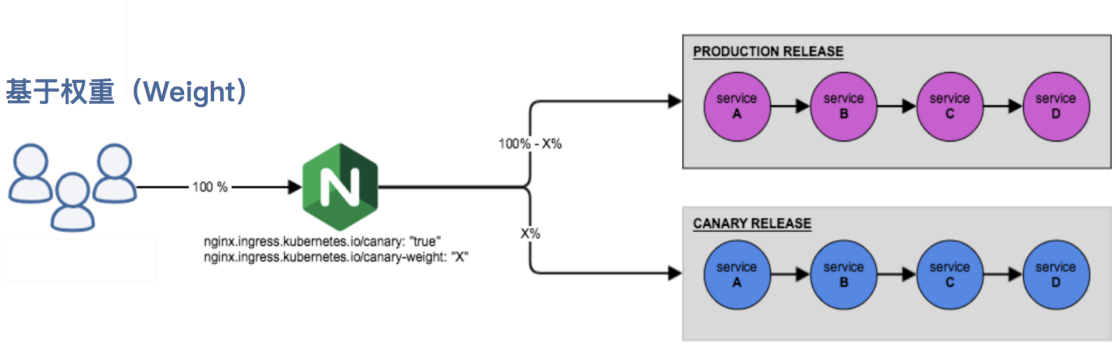
ingress-nginx 的 Annotations 支持以下 4 种 Canary 规则：

- `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；对于任何其他 Header 值，将忽略 Header，并通过优先级将请求与其他金丝雀规则进行优先级的比较。
- `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。该规则允许用户自定义 Request Header 的值，必须与上一个 annotation (即：canary-by-header) 一起使用。
- `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 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；对于任何其他值，将忽略 cookie 并将请求与其他金丝雀规则进行优先级的比较。

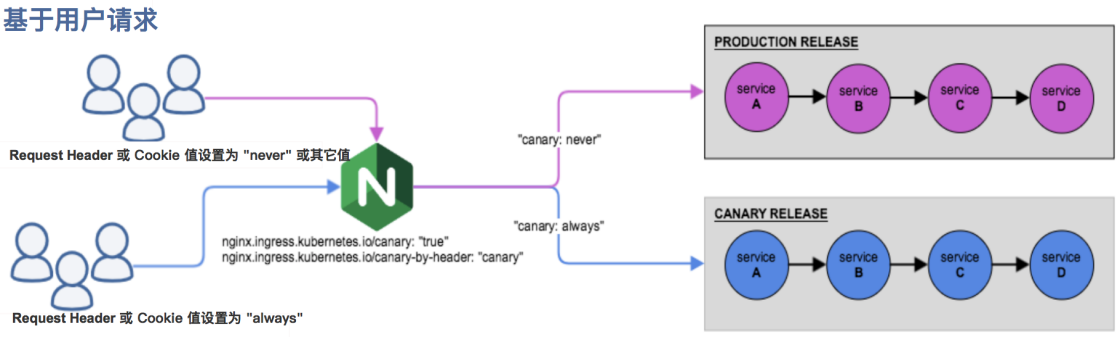
> 需要注意的是金丝雀规则按优先顺序进行排序：`canary-by-header - > canary-by-cookie - > canary-weight`

总的来说可以把以上的四个 annotation 规则划分为以下两类：

- 基于权重的 Canary 规则




- 基于用户请求的 Canary 规则



下面我们通过一个示例应用来对灰度发布功能进行说明。

**第一步. 部署 Production 应用**

首先创建一个 production 环境的应用资源清单：

```
# production.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: production
  labels:
    app: production
spec:
  selector:
    matchLabels:
      app: production
  template:
    metadata:
      labels:
        app: production
    spec:
      containers:
      - name: production
        image: cnych/echoserver
        ports:
        - containerPort: 8080
        env:
          - name: NODE_NAME
            valueFrom:
              fieldRef:
                fieldPath: spec.nodeName
          - name: POD_NAME
            valueFrom:
              fieldRef:
                fieldPath: metadata.name
          - name: POD_NAMESPACE
            valueFrom:
              fieldRef:
                fieldPath: metadata.namespace
          - name: POD_IP
            valueFrom:
              fieldRef:
                fieldPath: status.podIP
---
apiVersion: v1
kind: Service
metadata:
  name: production
  labels:
    app: production
spec:
  ports:
  - port: 80
    targetPort: 8080
    name: http
  selector:
    app: production
```

然后创建一个用于 production 环境访问的 Ingress 资源对象：

```
# production-ingress.yaml
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: production
  annotations:
    kubernetes.io/ingress.class: nginx
spec:
  rules:
  - host: echo.qikqiak.com
    http:
      paths:
      - backend:
          serviceName: production
          servicePort: 80
```

直接创建上面的几个资源对象：

```
➜ kubectl apply -f production.yaml
➜ kubectl apply -f production-ingress.yaml
➜ kubectl get pods -l app=production
NAME                          READY   STATUS    RESTARTS   AGE
production-856d5fb99-d6bds   1/1     Running   0          2m50s
➜ kubectl get ingress
NAME         CLASS    HOSTS                ADDRESS        PORTS   AGE
production   <none>   echo.qikqiak.com     10.151.30.11   80      90s
```

应用部署成功后，将域名 `echo.qikqiak.com` 映射到 master1 节点（ingress-nginx 所在的节点）的外网 IP，然后即可正常访问应用了：

```
➜ curl http://echo.qikqiak.com


Hostname: production-856d5fb99-d6bds

Pod Information:
    node name:  node1
    pod name:   production-856d5fb99-d6bds
    pod namespace:  default
    pod IP: 10.244.1.111

Server values:
    server_version=nginx: 1.13.3 - lua: 10008

Request Information:
    client_address=10.244.0.0
    method=GET
    real path=/
```

```
    query=
    request_version=1.1
    request_scheme=http
    request_uri=http://echo.qikqiak.com:8080/

Request Headers:
    accept=*/*
    host=echo.qikqiak.com
    user-agent=curl/7.64.1
    x-forwarded-for=171.223.99.184
    x-forwarded-host=echo.qikqiak.com
    x-forwarded-port=80
    x-forwarded-proto=http
    x-real-ip=171.223.99.184
    x-request-id=e680453640169a7ea21afba8eba9e116
    x-scheme=http

Request Body:
    -no body in request-
```

**第二步. 创建 Canary 版本** 参考将上述 Production 版本的 `production.yaml` 文件，再创建一个 Canary 版本的应用。

```
# canary.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: canary
  labels:
    app: canary
spec:
  selector:
    matchLabels:
      app: canary
  template:
    metadata:
      labels:
        app: canary
    spec:
      containers:
      - name: canary
        image: cnych/echoserver
        ports:
        - containerPort: 8080
        env:
          - name: NODE_NAME
            valueFrom:
              fieldRef:
                fieldPath: spec.nodeName
          - name: POD_NAME
            valueFrom:
              fieldRef:
                fieldPath: metadata.name
          - name: POD_NAMESPACE
            valueFrom:
```

```
              fieldRef:
                fieldPath: metadata.namespace
          - name: POD_IP
            valueFrom:
              fieldRef:
                fieldPath: status.podIP
---
apiVersion: v1
kind: Service
metadata:
  name: canary
  labels:
    app: canary
spec:
  ports:
  - port: 80
    targetPort: 8080
    name: http
  selector:
    app: canary
```

接下来就可以通过配置 Annotation 规则进行流量切分了。

**第三步. Annotation 规则配置**

**1. 基于权重**：基于权重的流量切分的典型应用场景就是蓝绿部署，可通过将权重设置为 0 或 100 来实现。例如，可将 Green 版本设置为主要部分，并将 Blue 版本的入口配置为 Canary。最初，将权重设置为 0，因此不会将流量代理到 Blue 版本。一旦新版本测试和验证都成功后，即可将 Blue 版本的权重设置为 100，即所有流量从 Green 版本转向 Blue。

创建一个基于权重的 Canary 版本的应用路由 Ingress 对象。

```
# canary-ingress.yaml
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: canary
  annotations:
    kubernetes.io/ingress.class: nginx
    nginx.ingress.kubernetes.io/canary: "true"    # 要开启灰度发布机制，首先需要启用
Canary
    nginx.ingress.kubernetes.io/canary-weight: "30"  # 分配30%流量到当前Canary版本
spec:
  rules:
  - host: echo.qikqiak.com
    http:
      paths:
      - backend:
          serviceName: canary
          servicePort: 80
```

直接创建上面的资源对象即可：

```
➜ kubectl apply -f canary.yaml
```

```
➜ kubectl apply -f canary-ingress.yaml
➜ kubectl get pods
NAME                          READY   STATUS    RESTARTS   AGE
canary-66cb497b7f-48zx4       1/1     Running   0          7m48s
production-856d5fb99-d6bds    1/1     Running   0          21m
......
➜ kubectl get svc
NAME                         TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)
                  AGE
canary                       ClusterIP   10.106.91.106    <none>        80/TCP
                  8m23s
production                   ClusterIP   10.105.182.15    <none>        80/TCP
                  22m
......
➜ kubectl get ingress
NAME         CLASS    HOSTS                ADDRESS         PORTS   AGE
canary       <none>   echo.qikqiak.com     10.151.30.11    80      108s
production   <none>   echo.qikqiak.com     10.151.30.11    80      22m
```

Canary 版本应用创建成功后，接下来我们在命令行终端中来不断访问这个应用，观察 Hostname 变化：

```
➜ for i in $(seq 1 10); do curl -s echo.qikqiak.com | grep "Hostname"; done
Hostname: production-856d5fb99-d6bds
Hostname: canary-66cb497b7f-48zx4
Hostname: production-856d5fb99-d6bds
Hostname: production-856d5fb99-d6bds
Hostname: production-856d5fb99-d6bds
Hostname: production-856d5fb99-d6bds
Hostname: production-856d5fb99-d6bds
Hostname: canary-66cb497b7f-48zx4
Hostname: canary-66cb497b7f-48zx4
Hostname: production-856d5fb99-d6bds
```

由于我们给 Canary 版本应用分配了 30% 左右权重的流量，所以上面我们访问10次有3次访问到了 Canary 版本的应用，符合我们的预期。

**2. 基于 Request Header**: 基于 Request Header 进行流量切分的典型应用场景即灰度发布或 A/B 测试场景。

在上面的 Canary 版本的 Ingress 对象中新增一条 annotation 配置 `nginx.ingress.kubernetes.io/canary-by-header: canary`（这里的 value 可以是任意值），使当前的 Ingress 实现基于 Request Header 进行流量切分，由于 `canary-by-header` 的优先级大于 `canary-weight`，所以会忽略原有的 `canary-weight` 的规则。

```
annotations:
  kubernetes.io/ingress.class: nginx
  nginx.ingress.kubernetes.io/canary: "true"    # 要开启灰度发布机制，首先需要启用
Canary
  nginx.ingress.kubernetes.io/canary-by-header: canary  # 基于header的流量切分
  nginx.ingress.kubernetes.io/canary-weight: "30"  # 会被忽略，因为配置了 canary-
by-headerCanary版本
```

更新上面的 Ingress 资源对象后，我们在请求中加入不同的 Header 值，再次访问应用的域名。

> 注意：当 Request Header 设置为 never 或 always 时，请求将不会或一直被发送到 Canary 版本，对于任何其他 Header 值，将忽略 Header，并通过优先级将请求与其他 Canary 规则进行优先级的比较。

```
➜ for i in $(seq 1 10); do curl -s -H "canary: never" echo.qikqiak.com | grep
"Hostname"; done
Hostname: production-856d5fb99-d6bds
Hostname: production-856d5fb99-d6bds
Hostname: production-856d5fb99-d6bds
Hostname: production-856d5fb99-d6bds
Hostname: production-856d5fb99-d6bds
Hostname: production-856d5fb99-d6bds
Hostname: production-856d5fb99-d6bds
Hostname: production-856d5fb99-d6bds
Hostname: production-856d5fb99-d6bds
Hostname: production-856d5fb99-d6bds
```

这里我们在请求的时候设置了 `canary: never` 这个 Header 值，所以请求没有发送到 Canary 应用中去。如果设置为其他值呢：

```
➜ for i in $(seq 1 10); do curl -s -H "canary: other-value" echo.qikqiak.com |
grep "Hostname"; done
Hostname: production-856d5fb99-d6bds
Hostname: production-856d5fb99-d6bds
Hostname: canary-66cb497b7f-48zx4
Hostname: production-856d5fb99-d6bds
Hostname: production-856d5fb99-d6bds
Hostname: production-856d5fb99-d6bds
Hostname: production-856d5fb99-d6bds
Hostname: canary-66cb497b7f-48zx4
Hostname: production-856d5fb99-d6bds
Hostname: canary-66cb497b7f-48zx4
```

由于我们请求设置的 Header 值为 `canary: other-value`，所以 ingress-nginx 会通过优先级将请求与其他 Canary 规则进行优先级的比较，我们这里也就会进入 `canary-weight: "30"` 这个规则去。

这个时候我们可以在上一个 annotation (即 `canary-by-header`）的基础上添加一条 `nginx.ingress.kubernetes.io/canary-by-header-value: user-value` 这样的规则，就可以将请求路由到 Canary Ingress 中指定的服务了。

```
annotations:
  kubernetes.io/ingress.class: nginx
  nginx.ingress.kubernetes.io/canary: "true"    # 要开启灰度发布机制，首先需要启用
Canary
  nginx.ingress.kubernetes.io/canary-by-header-value: user-value
  nginx.ingress.kubernetes.io/canary-by-header: canary  # 基于header的流量切分
  nginx.ingress.kubernetes.io/canary-weight: "30"  # 分配30%流量到当前Canary版本
```

同样更新 Ingress 对象后，重新访问应用，当 Request Header 满足 `canary: user-value` 时，所有请求就会被路由到 Canary 版本：

```
➜ for i in $(seq 1 10); do curl -s -H "canary: user-value" echo.qikqiak.com |
grep "Hostname"; done
Hostname: canary-66cb497b7f-48zx4
Hostname: canary-66cb497b7f-48zx4
Hostname: canary-66cb497b7f-48zx4
Hostname: canary-66cb497b7f-48zx4
Hostname: canary-66cb497b7f-48zx4
Hostname: canary-66cb497b7f-48zx4
Hostname: canary-66cb497b7f-48zx4
Hostname: canary-66cb497b7f-48zx4
Hostname: canary-66cb497b7f-48zx4
Hostname: canary-66cb497b7f-48zx4
```

**3. 基于 Cookie**：与基于 Request Header 的 annotation 用法规则类似。例如在 A/B 测试场景下，需要让地域为北京的用户访问 Canary 版本。那么当 cookie 的 annotation 设置为 `nginx.ingress.kubernetes.io/canary-by-cookie: "users_from_Beijing"`，此时后台可对登录的用户请求进行检查，如果该用户访问源来自北京则设置 cookie `users_from_Beijing` 的值为 `always`，这样就可以确保北京的用户仅访问 Canary 版本。

同样我们更新 Canary 版本的 Ingress 资源对象，采用基于 Cookie 来进行流量切分，

```
annotations:
  kubernetes.io/ingress.class: nginx
  nginx.ingress.kubernetes.io/canary: "true"    # 要开启灰度发布机制，首先需要启用
Canary
  nginx.ingress.kubernetes.io/canary-by-cookie: "users_from_Beijing"  # 基于
cookie
  nginx.ingress.kubernetes.io/canary-weight: "30"  # 会被忽略，因为配置了 canary-
by-cookie
```

更新上面的 Ingress 资源对象后，我们在请求中设置一个 `users_from_Beijing=always` 的 Cookie 值，再次访问应用的域名。

```
➜ for i in $(seq 1 10); do curl -s -b "users_from_Beijing=always"
echo.qikqiak.com | grep "Hostname"; done
Hostname: canary-66cb497b7f-48zx4
Hostname: canary-66cb497b7f-48zx4
Hostname: canary-66cb497b7f-48zx4
Hostname: canary-66cb497b7f-48zx4
Hostname: canary-66cb497b7f-48zx4
Hostname: canary-66cb497b7f-48zx4
Hostname: canary-66cb497b7f-48zx4
Hostname: canary-66cb497b7f-48zx4
Hostname: canary-66cb497b7f-48zx4
Hostname: canary-66cb497b7f-48zx4
```

我们可以看到应用都被路由到了 Canary 版本的应用中去了，如果我们将这个 Cookie 值设置为 never，则不会路由到 Canary 应用中