Если вы видите что-то необычное, просто сообщите мне.

2 способа направить трафик Ingress между пространствами Kubernetes

The tech industry of full of workarounds, you are probably using or relying on some workaround. And there is no problem with that per se. But most important is that when you do a workaround you should be aware of that and change it to the standard way if it's more intuitive.

Проблема

A couple of years ago I had a use case where a single domain had 2 sub-paths each of them having its own service in different namespaces. Let's see this example:

Пару лет назад я встретил такую ситуацию когда один домен имел 2 дополнительных маршрута к сервисам которые находились в соседних namespaces. Давайте рассмотрим пример:

example.com/app => сервис "backend" в пространстве "app"
example.com/blog => сервис "wordpress" в пространстве "blog".

Проблема была в том, что Ingress может отправлять траффик в сервис внутри одного пространства, ingress может быть только 1 для домен. Но дальше Nginx Ingress ввели так

называемый Mergeable Ingress Resources. Однако, в это же время я работал со старой версией которая не поддерживала подобного.

В след раз, я нашел общее решение которое выглядит как костыли. В общем говоря, это не плохие костыли. В зависимости о том, как вы управляете инфраструктурой, и вы можете смотреть на это как централизованное или нецентрализованное применение .

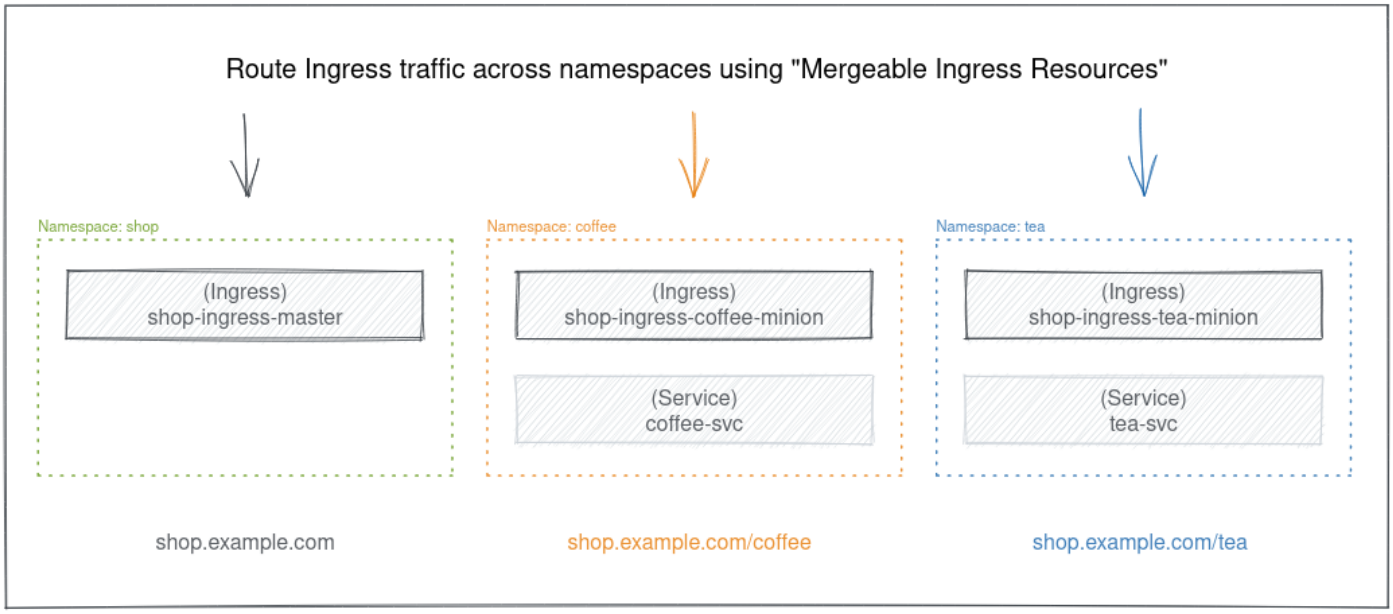
Возможно два решения.

The Solution So here are the 2 ways to route Ingress traffic across namespaces in Kubernetes. The 1st one you could call the standard way (which relies on the Ingress controller capabilities), and 2nd is the generic way that I used back in the days.

1. Mergeable Ingress Resources

If you took a look at the official Nginx docs you will find the Cross-namespace Configuration page suggests using Mergeable Ingress Resources. That approach relies on a simple idea, there is a single Ingress resource that has all configurations related to the host/domain and that resource is called "master", and any number of the Ingress resources handles the paths under that host/domain and each of these resources is called "minion".

Each one of the master or minion can or can not contain some of the Ingress annotations based on their role. Here I will use here the examples from the official documentation.



Config for shop.example.com like TLS and host-level annotations.

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: shop-ingress-master
  namespace: shop
  annotations:
    kubernetes.io/ingress.class: "nginx"
    nginx.org/mergeable-ingress-type: "master"
spec:
  tls:
  - hosts:
    - shop.example.com
    secretName: shop-secret
  rules:
  - host: shop.example.com
```

Config for shop.example.com/coffee which is in the coffee namespace and routes the traffic of the coffee-svc service.

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
```

```
  name: shop-ingress-coffee-minion
  namespace: coffee
  annotations:
    kubernetes.io/ingress.class: "nginx"
    nginx.org/mergeable-ingress-type: "minion"
spec:
  rules:
  - host: shop.example.com
    http:
      paths:
      - path: /coffee
        pathType: Prefix
        backend:
          service:
            name: coffee-svc
            port:
              number: 80
```

Config for shop.example.com/tea which is in the tea namespace and routes the traffic of the tea-svc service.

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: shop-ingress-tea-minion
  namespace: tea
  annotations:
    kubernetes.io/ingress.class: "nginx"
    nginx.org/mergeable-ingress-type: "minion"
spec:
  rules:
  - host: shop.example.com
    http:
      paths:
      - path: /tea
        pathType: Prefix
        backend:
          service:
            name: tea-svc
            port:
```

```
            number: 80
```

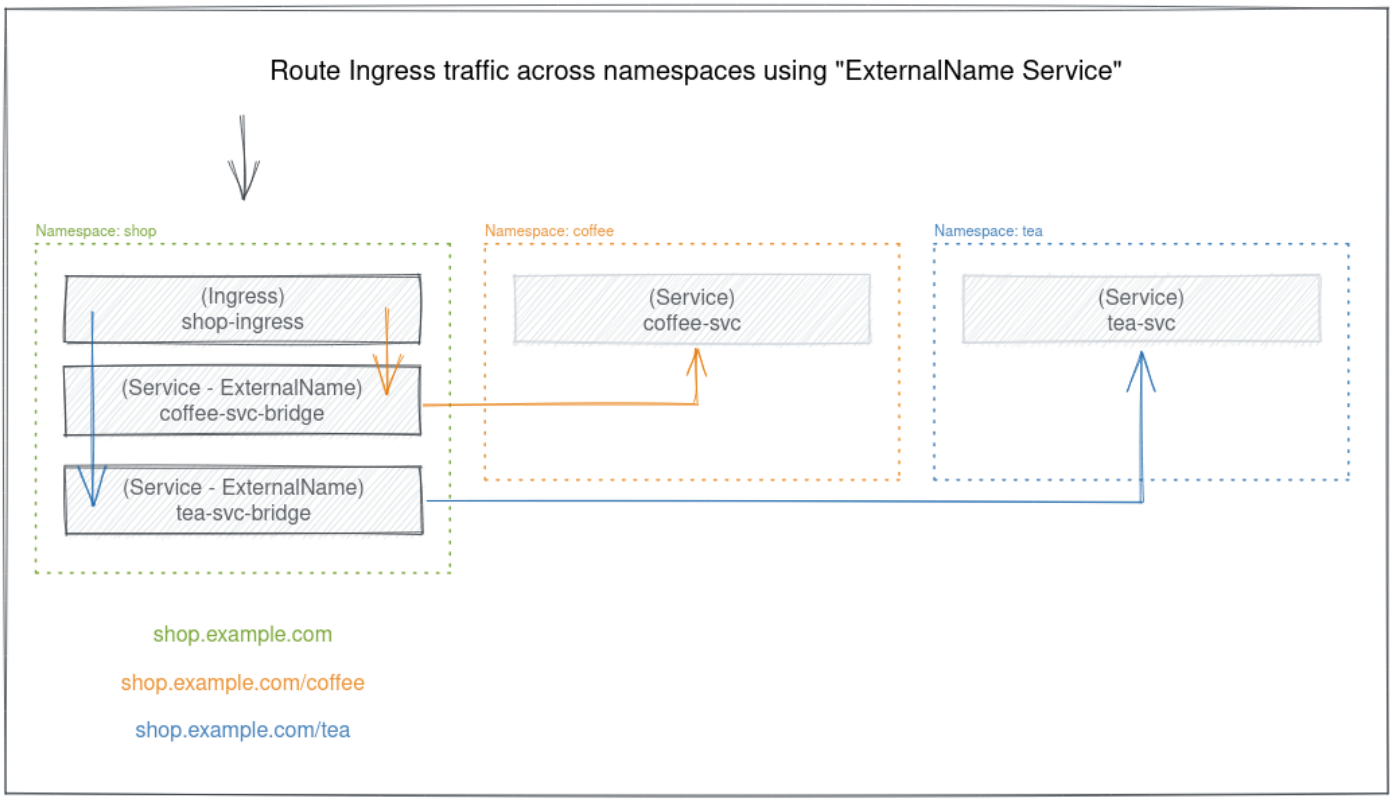
As you see, the Ingress config is split into 2 parts, the host/domain config, and the paths config. Each one of them could be in a different namespace and handles the services in that namespace.

2. ExternalName Service

For one reason or another, that non-centralized way of managing Ingress resources (where the Ingress object is split across namespaces) might not fit all workloads. So here is another way I used it before and I find it much simpler for many use cases.

This method relies on native Kubernetes ExternalName Service which is simply a DNS CNAME! This method is centralized where it uses the normal Ingress object in addition to ExternalName Service within the same namespace as a bridge to the services in any other namespace.

The following is an example of that setup with a single Ingress resource and 2 ExternalName services.



Config for shop.example.com including the 2 sub-paths /coffee and /tea.

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: shop-ingress
  namespace: shop
  annotations:
    kubernetes.io/ingress.class: "nginx"
spec:
  tls:
  - hosts:
    - shop.example.com
    secretName: shop-secret
  rules:
  - host: shop.example.com
    http:
      paths:
      - path: /coffee
        pathType: Prefix
        backend:
          service:
            name: coffee-svc-bridge
            port:
              number: 80
      - path: /tea
        pathType: Prefix
        backend:
          service:
            name: tea-svc-bridge
            port:
              number: 80
```

The coffee-svc-bridge service in the shop namespace is a CNAME for the coffee-svc service in coffee namespace:

```
apiVersion: v1
kind: Service
metadata:
  name: coffee-svc-bridge
  namespace: shop
```

```
spec:
  type: ExternalName
  externalName: coffee-svc.coffee
```

The tea-svc-bridge service in the shop namespace is a CNAME for the tea-svc service in tea namespace:

```
apiVersion: v1
kind: Service
metadata:
  name: tea-svc-bridge
  namespace: shop
spec:
  type: ExternalName
  externalName: tea-svc.tea
```

As you see, the Ingress config comes in 1 part and is normal. And use the ExternalName services as a bridge to access the services in the other namespaces.

Conclusion

Maybe the second approach looks like a workaround, but for some workloads could be better and easier to follow and digest. But in general, it's good to have different ways to use what's fit better.

Enjoy :-)

Revision #4
Created 2023-06-13 10:00:00 UTC by gasick
Updated 2023-06-13 20:26:57 UTC by gasick