istio-simple-concepts

Destination Rules

Virtual Service

Destination Rule

Destination rules serve as the single source of truth about which service versions are available to receive traffic from virtual services. You can use these resources to define policies that apply to traffic that is intended for a service after routing has occurred. Destination rules are describing circuit breaking, load balancing, tls settings..

Simple Destination Rule

```yaml
apiVersion: networking.istio.io/v1beta1
kind: DestinationRule
metadata:
  name: stage-product-detail-api
spec:
  host: stage-product-detail-api
  subsets:
  - labels:
      app: product-detail-api
    name: stage
```

Least Connection Rule

```yaml
apiVersion: networking.istio.io/v1beta1
kind: DestinationRule
metadata:
  name: stage-product-detail-api
spec:
  host: stage-product-detail-api
  trafficPolicy:
    loadBalancer:
      simple: LEAST_CONN
  subsets:
  - labels:
      app: product-detail-api
    name: stage
```

Version Based Connection Rule

```yaml
apiVersion: networking.istio.io/v1beta1
kind: DestinationRule
```

```
metadata:
  name: stage-product-detail-api
spec:
  host: stage-product-detail-api
  trafficPolicy:
    loadBalancer:
      simple: LEAST_CONN
  subsets:
  - labels:
      app: product-detail-api
      version: v1
    name: stage
    trafficPolicy:
      loadBalancer:
        simple: ROUND_ROBIN
```

Circuit Breaker

```
apiVersion: networking.istio.io/v1beta1
kind: DestinationRule
metadata:
  name: stage-product-detail-api
spec:
  host: stage-product-detail-api-ds
  trafficPolicy:
    connectionPool:
      http:
        http1MaxPendingRequests: 1
        maxRequestsPerConnection: 1
      tcp:
        maxConnections: 1
    outlierDetection:
      baseEjectionTime: 180.000s
      consecutiveErrors: 1
      interval: 1.000s
      maxEjectionPercent: 100
```

- Maximum Connections: The maximum number of connections to a backend. Any excess connection will be pending in a queue. You can modify this number by changing the maxConnections field.
- Maximum Pending Requests: The maximum number of pending requests to a backend. Any excess pending requests will be denied. You can modify this number by changing the http1MaxPendingRequests field.
- Maximum Requests: The maximum number of requests in a cluster at any given time. You can modify this number by changing the maxRequestsPer-

Connection field.

https://www.envoyproxy.io/docs/envoy/latest/api-v2/api/v2/cluster/outlier_detection.proto

Any Host Match

```
apiVersion: networking.istio.io/v1beta1
kind: DestinationRule
metadata:
  name: stage-product-detail-api
spec:
  host: "*"
  trafficPolicy:
    loadBalancer:
      simple: LEAST_CONN
  subsets:
  - labels:
      app: product-detail-api
    name: stage
```

Apply any inbound host match.

Virtual Service

A VirtualService resource in much the same capacity as a traditional Kubernetes Ingress resource, in that a VirtualService resource matches traffic and directs it to a Service resource.

However, a VirtualService resource can be much more specific in the traffic it matches and where that traffic is sent, it will operate on internal as well as external traffic, and offers a lot of additional functionality to manipulate traffic along the way. It can be combine with destination rule to foward traffic specific pod.

Simple Virtual Service

```
kind: VirtualService
metadata:
  name: stage-product-detail-api
spec:
  hosts:
  - stageproductdetailapi.trendyol.com
  http:
    route:
    - destination:
        host: stage-product-detail-api
        subset: stage
```

HTTPMatchRequest Specific Uri Prefix Virtual Service

```yaml
kind: VirtualService
metadata:
  name: stage-product-detail-api
spec:
  hosts:
  - stageproductdetailapi.trendyol.com
  http:
  - match:
    - uri:
        prefix: /test
    route:
    - destination:
        host: stage-product-detail-api
        subset: stage
```

HTTPMatchRequest Specific Uri Regex Virtual Service

```yaml
kind: VirtualService
metadata:
  name: stage-product-detail-api
spec:
  hosts:
  - stageproductdetailapi.trendyol.com
  http:
  - match:
    - uri:
        regex: /test
    route:
    - destination:
        host: stage-product-detail-api
        subset: stage
```

HTTPMatchRequest Specific Header Regex Virtual Service

```yaml
kind: VirtualService
metadata:
  name: stage-product-detail-api
spec:
  hosts:
  - stageproductdetailapi.trendyol.com
  http:
  - match:
    - headers:
        user-agent:
          regex: '^.*(trendyol).*$'
```

```
    route:
    - destination:
        host: stage-product-detail-api
        subset: stage
```

Timeout Virtual Service

```
kind: VirtualService
metadata:
  name: stage-product-detail-api
spec:
  hosts:
  - stageproductdetailapi.trendyol.com
  http:
  - match:
    - headers:
        user-agent:
            regex: '^.*(trendyol).*$'
  - timeout: 0.200s
    route:
    - destination:
        host: stage-product-detail-api
        subset: stage
```

It work with if your service in mesh. To control routing for traffic bound to services outside the mesh you need to create service entry and custom virtual service.

Network Fail Virtual Service (Injecting faults)

```
kind: VirtualService
metadata:
  name: stage-product-detail-api
spec:
  hosts:
  - stageproductdetailapi.trendyol.com
  http:
  - match:
    - headers:
        user-agent:
            regex: '^.*(trendyol).*$'
  - timeout: 0.200s
    route:
    - destination:
        host: stage-product-detail-api
        subset: stage
    fault:
```

```yaml
abort:
  percentage:
    value: 50
  httpStatus: 5xx
```