服务网格

常见问题

腾讯云

文档目录

常见问题

常见问题汇总

最近更新时间：2024-03-25 10:10:56

权限相关
为集群添加网格权限

最近更新时间：2023-05-22 11:10:05

问题描述

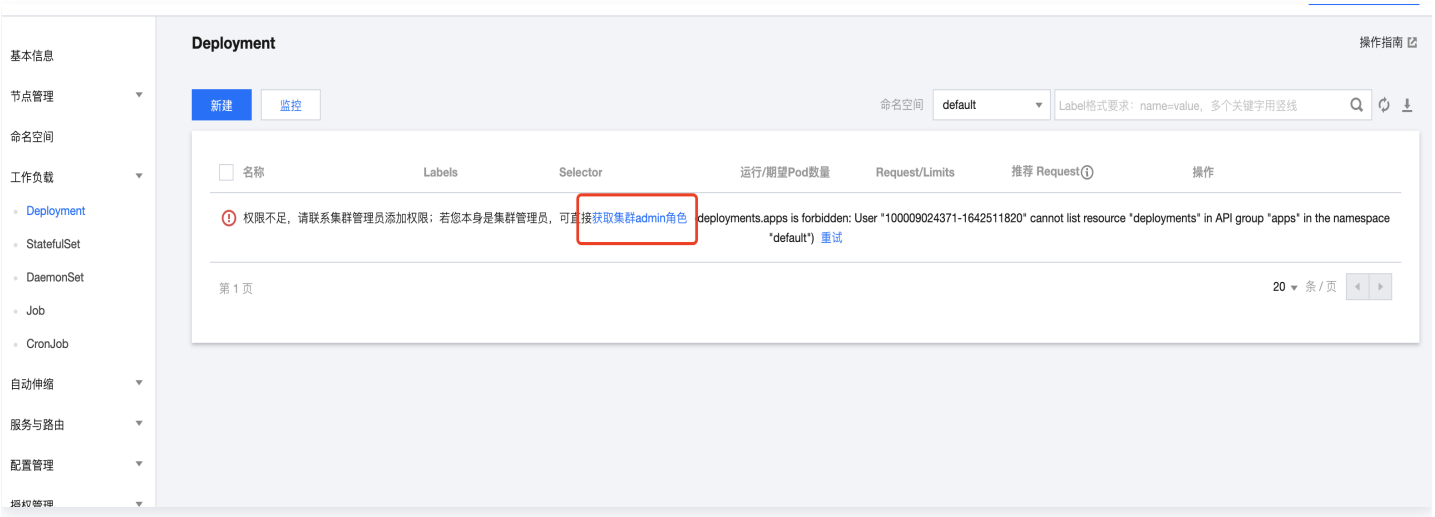
使用子账号身份登录腾讯云控制台操作服务网格，提示无集群权限：

解决方案

当前登录的子账号无该集群 admin 权限，可参考以下集群授权方法。

管理员云账号自授权

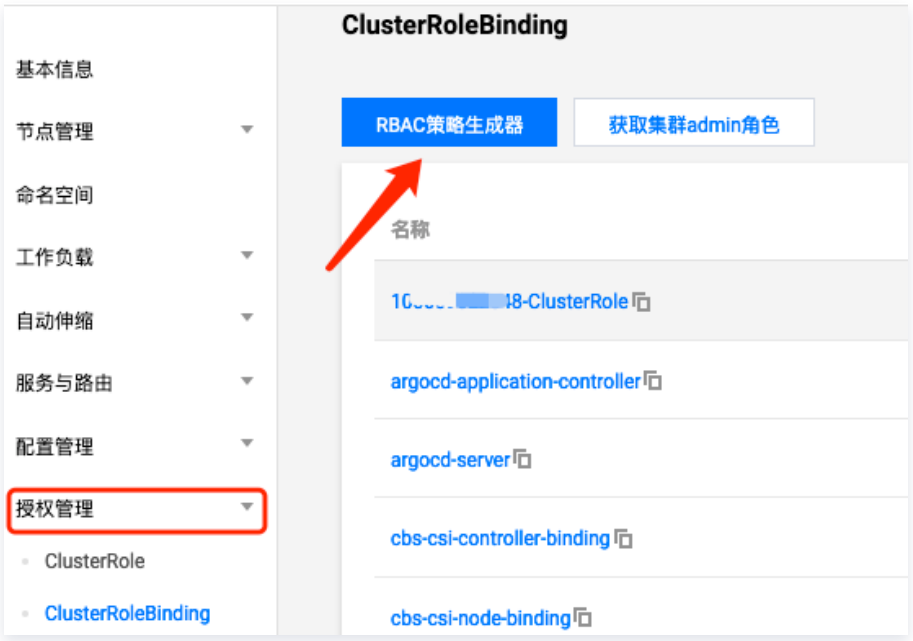
如果您的主账号具备管理员权限，可直接在 TKE 控制台，使用**获取集群admin角色**功能，快速为自己的子账号授予该集群的 admin 权限，如下图所示：



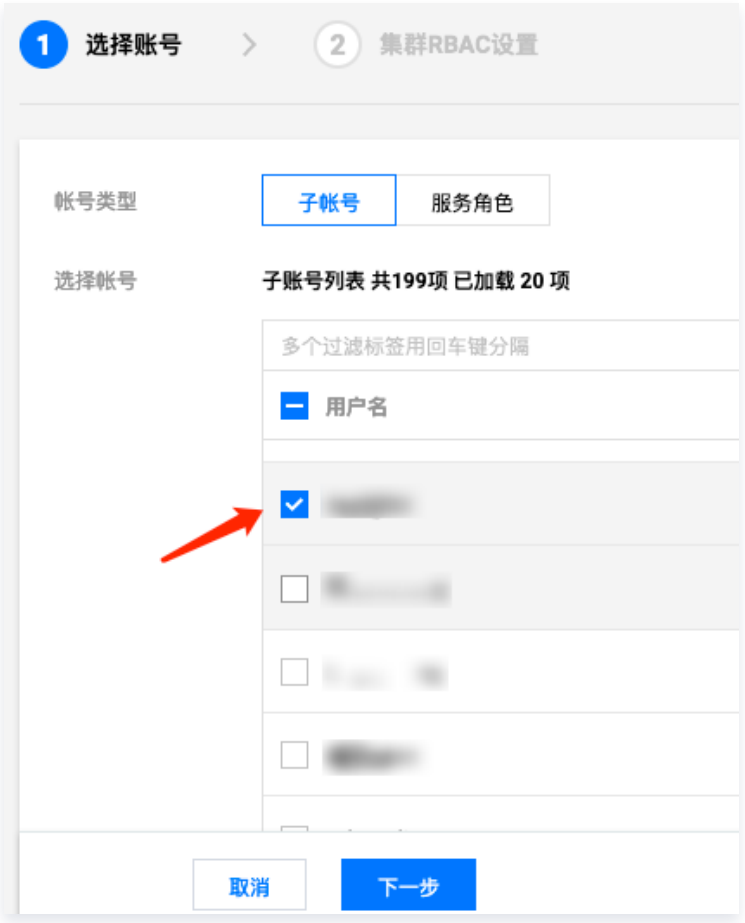
为其他子账号授权

1. 登录集群创建者的腾讯云账号（用该账号为子账号授权）。
2. 在 TKE 控制台 中选择集群。

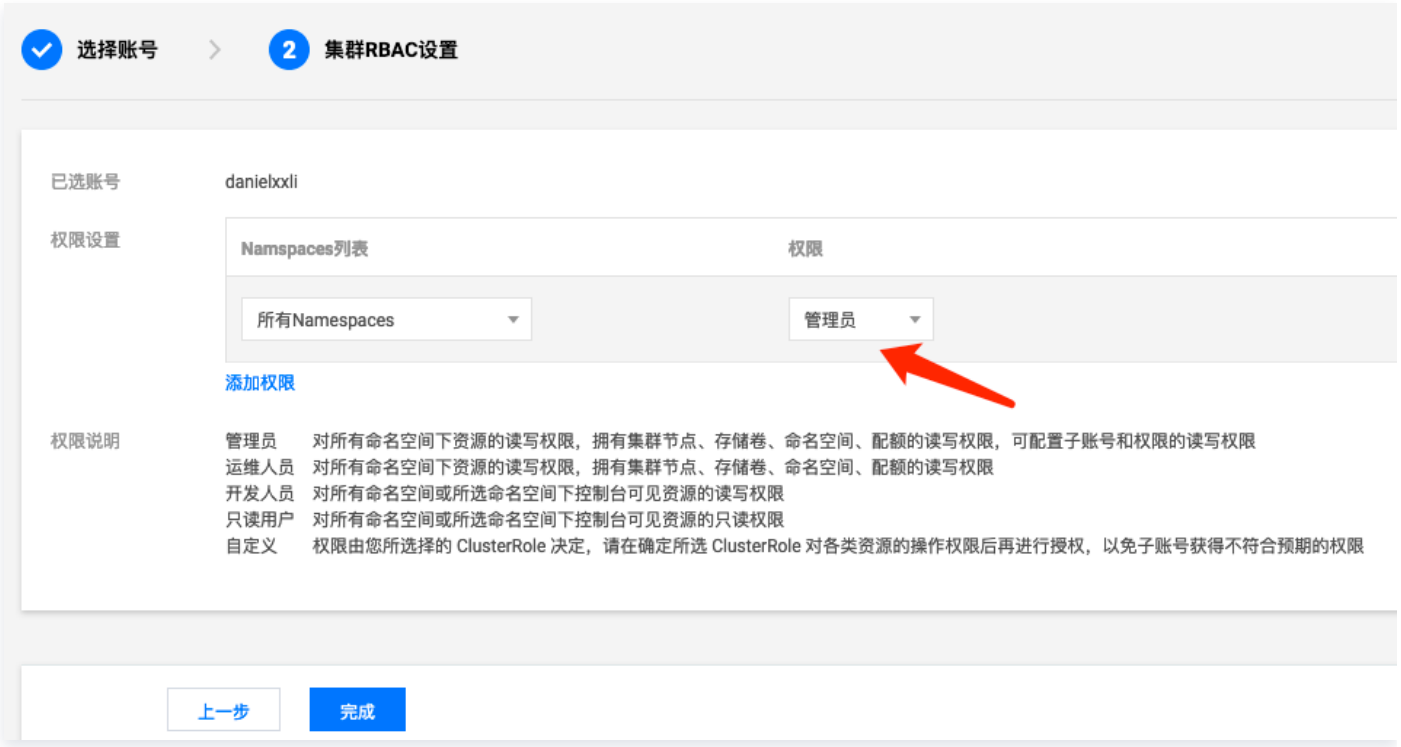
3. 在集群详情页，单击**授权管理** > **RBAC策略生成器**。如下图所示：



4. 在弹窗中勾选要授权的子账号。如下图所示：



5. 单击**完成**，授权管理员权限。如下图所示：



子账号集群管理失败

最近更新时间：2023-02-28 17:41:25

使用子账号身份登录腾讯云控制台操作服务网格，有时可能会遇到权限问题：

- **关联集群失败，提示权限问题：**

- **查看服务时提示权限不足：**

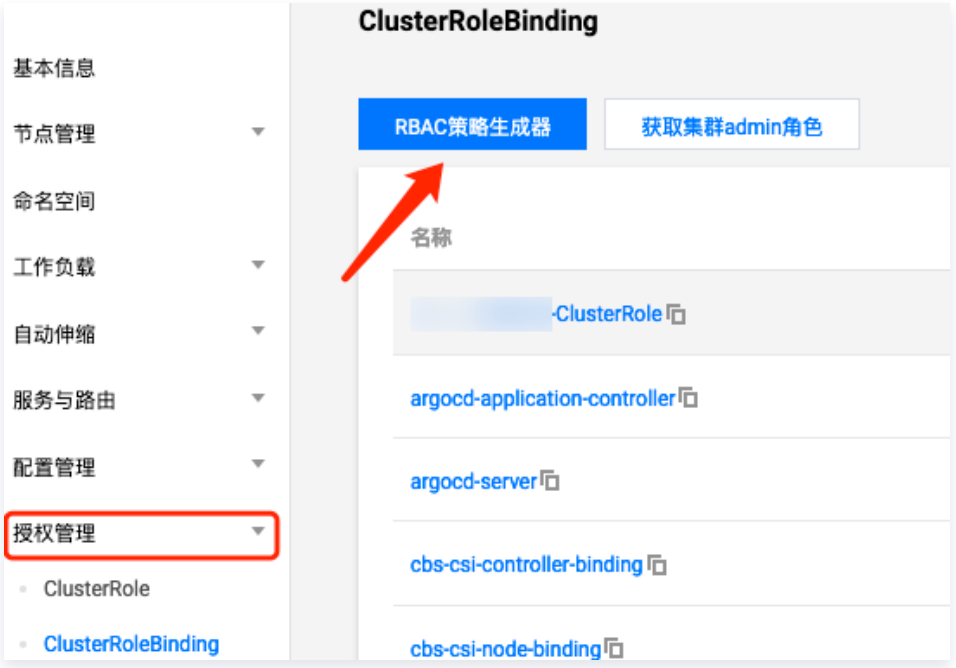
- **升级网格时提示 403 Forbidden：**

原因

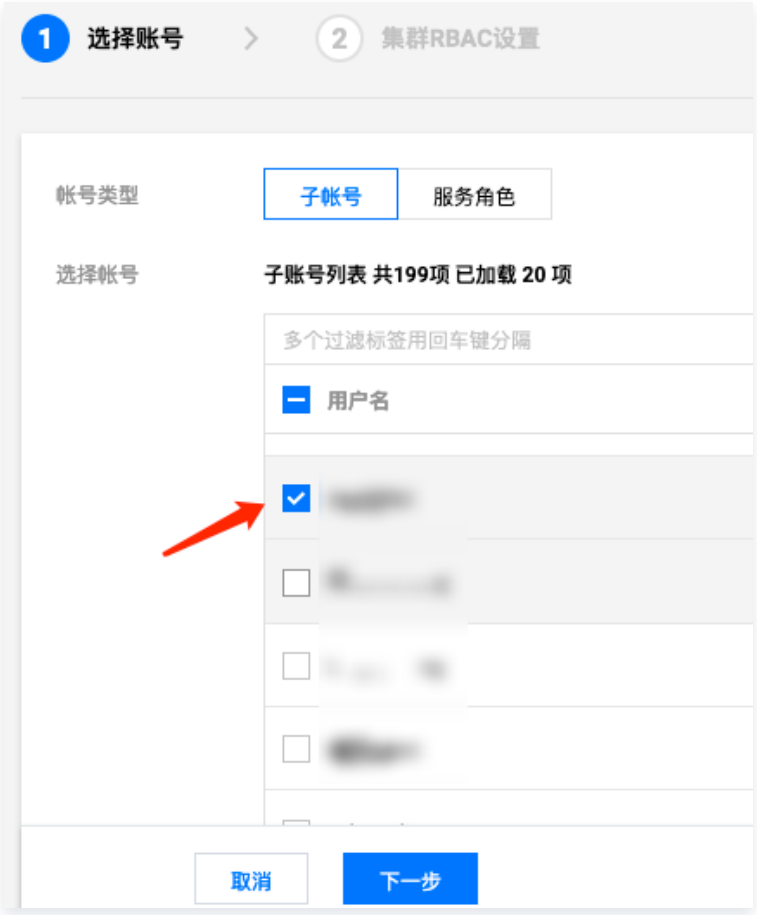
核心点在于当前登录的子账号需要有操作网格所关联集群的权限，如果没有或者部分集群没有，就需要给子账号授权（每个集群都需要），可参考以下授权方法。

集群授权方法

1. 登录集群创建者的腾讯云账号（用该账号为子账号授权）。
2. 进入 容器服务控制台，在集群列表中单击集群 ID，进入集群详情页。
3. 选择**授权管理 > ClusterRoleBinding**，单击 **RBAC 策略生成器**。如下图所示：



4. 勾选要授权的子账号，并单击**下一步**。如下图所示：



5. 在**集群 RBAC 设置**中，为已选账号设置管理员权限。如下图所示：

6. 单击**完成**。

域名配置相关问题

最近更新时间：2024-03-25 14:54:33

是否支持泛域名?

支持。这是 istio 原生支持的能力，配置 VirtualService 时，hosts 字段写成泛域名的形式：

```
apiVersion: networking.istio.io/v1beta1
kind: VirtualService
metadata:
  name: example
spec:
  hosts:
  - "*.example.com"
  gateways:
  - "istio-system/example-gw"
  http:
    route:
    - destination:
        host: web.prod.svc.cluster.local
```

关联的 Gateway 中 hosts 也需要能覆盖到该泛域名：

```
apiVersion: networking.istio.io/v1beta1
kind: Gateway
metadata:
  name: example-gw
  namespace: istio-system
spec:
  selector:
    app: istio-ingressgateway
    istio: ingressgateway
  servers:
  - port:
      number: 80
      name: http
      protocol: HTTP
    hosts:
    - "*.example.com"
```

若您使用了 https，也确保使用的证书是签发的泛域名证书。

异常状态码汇总

最近更新时间：2024-03-25 15:13:21

431 状态码：Request Header Fields Too Large

现象

istio 中 http 请求，envoy 返回 431 异常状态码：

```
HTTP/1.1 431 Request Header Fields Too Large
```

原因分析

此状态码说明 http 请求 header 大小超限了，默认限制为60KiB，由 `HttpConnectionManager` 配置的 `max_request_headers_kb` 字段决定，最大可调整到96KiB：

max_request_headers_kb

(UInt32Value) The maximum request headers size for incoming connections. If unconfigured, the default max request headers allowed is 60 KiB. Requests that exceed this limit will receive a 431 response. The max configurable limit is 96 KiB, based on current implementation constraints.

解决方案

可以通过 EnvoyFilter 调整 `max_request_headers_kb` 字段来提升 header 大小限制。
EnvoyFilter 示例（istio 1.6 验证通过）：

```
apiVersion: networking.istio.io/v1alpha3
kind: EnvoyFilter
metadata:
  name: max-header
  namespace: istio-system
spec:
  configPatches:
  - applyTo: NETWORK_FILTER
    match:
      context: ANY
      listener:
        filterChain:
          filter:
            name: "envoy.http_connection_manager"
    patch:
      operation: MERGE
      value:
        typed_config:
          "@type":
"type.googleapis.com/envoy.config.filter.network.http_connection_manager.v2.HttpConnec
```

```
tionManager"
          max_request_headers_kb: 96
```

高版本兼容上面的 v2 配置，但建议用 v3 的 配置 (istio 1.8 验证通过)：

```
apiVersion: networking.istio.io/v1alpha3
kind: EnvoyFilter
metadata:
  name: max-header
  namespace: istio-system
spec:
  configPatches:
  - applyTo: NETWORK_FILTER
    match:
      context: ANY
      listener:
        filterChain:
          filter:
            name: "envoy.http_connection_manager"
    patch:
      operation: MERGE
      value:
        typed_config:
          "@type":
"type.googleapis.com/envoy.extensions.filters.network.http_connection_manager.v3.HttpC
onnectionManager"
          max_request_headers_kb: 96
```

若 header 大小超过 96 KiB，这种情况本身也很不正常，建议将这部分数据放到 body。

426 状态码：Upgrade Required

现象

Istio 使用 Envoy 作为数据面转发 HTTP 请求，而 Envoy 默认要求使用 HTTP/1.1 或 HTTP/2，当客户端使用 HTTP/1.0 时就会返回 426 Upgrade Required。

常见的 nginx 场景

如果用 nginx 进行 proxy_pass 反向代理，默认会用 HTTP/1.0，您可以显示指定 proxy_http_version 为 1.1：

```
upstream http_backend {
    server 127.0.0.1:8080;

    keepalive 16;
}

server {
    ...
```

```
    location /http/ {
        proxy_pass http://http_backend;
        proxy_http_version 1.1;
        proxy_set_header Connection "";
        ...
    }
}
```

压测场景

ab 压测时会发送 HTTP/1.0 的请求，Envoy 固定返回 426 Upgrade Required，根本不会进行转发，所以压测的结果也不会准确。可以换成其它压测工具，如 wrk 。

解决方案：让 istio 支持 HTTP/1.0

有些 SDK 或框架可能会使用 HTTP/1.0 协议，例如使用 HTTP/1.0 去资源中心/配置中心拉取配置信息，在不想改动代码的情况下让服务跑在 istio 上，也可以修改 istiod 配置，加上 `PILOT_HTTP10: 1` 的环境变量来启用 HTTP/1.0，这个在 TCM 中已产品化，可以在网格基本信息页面的高级设置中开启：

访问 StatefulSet Pod IP 返回 404

现象

在 istio 中业务容器访问同集群一 Pod IP 返回 404，在 istio-proxy 中访问却正常。

原因

Pod 属于 StatefulSet，使用 headless service，在 istio 中对 headless service 的支持跟普通 service 不太一样，如果 pod 用的普通 service，对应的 listener 有兜底的 passthrough，即转发到报文对应的真实目的IP+Port，但 headless service 的就没有，我们理解是因为 headless service 没有 vip，它的路由是确定的，只指向后端固定的 pod，如果路由匹配不上就肯定出了问题，如果也用 passthrough 兜底路由，只会掩盖问题，所以就没有为 headless service 创建 passthrough 兜底路由。同样的业务，上了 istio 才会有这个问题，也算是 istio 的设计或实现问题。

示例场景

使用了自己的服务发现，业务直接使用 Pod IP 调用 StatefulSet 的 Pod IP。

解决方案

使用网格一般不要直接访问 Pod IP，应该访问 service。如果实在有场景需要，同集群访问 statefulset pod ip 时带上 host，可以匹配上 headless service 路由，避免匹配不到发生 404。

Envoy Sidecar 未就绪导致 Pod 启动失败

最近更新时间：2023-08-29 10:10:56

一些服务在往 istio 上迁移过渡的过程中，有时可能会遇到 Pod 启动失败，然后一直重启，排查原因是业务启动时需要调用其它服务（例如从配置中心拉取配置），如果失败就退出，没有重试逻辑。调用失败的原因是 envoy 未就绪（envoy 需要从控制面拉取配置，需要一点时间），导致业务发出的流量无法被处理（参考 k8s issue #65502 ）。

最佳实践

目前这类问题的最佳实践是让应用更加健壮一点，增加重试逻辑，调用失败后不要立刻退出。或者在启动命令前加下 sleep，等待几秒。
如果不想改动应用，您可以参考以下规避方案。

规避方案

规避方案为调整 sidecar 注入顺序。
在 istio 1.7，社区通过给 istio-injector 注入逻辑增加一个 `HoldApplicationUntilProxyStarts` 开关解决了该问题，开关打开后，proxy 将会注入到第一个 container。
示例：

```
// Boolean flag for enabling/disabling the holdApplicationUntilProxyStarts behavior.
// This feature adds hooks to delay application startup until the pod proxy
// is ready to accept traffic, mitigating some startup race conditions.
// Default value is 'false'.
HoldApplicationUntilProxyStarts *types.BoolValue `protobuf:"bytes,33,opt,name=hold_application_unt
```

```
// nolint: staticcheck
holdPod := mc.DefaultConfig.HoldApplicationUntilProxyStarts.GetValue() ||
    valuesStruct.GetGlobal().GetProxy().GetHoldApplicationUntilProxyStarts().GetValue()

proxyLocation := MoveLast
// If HoldApplicationUntilProxyStarts is set, reorder the proxy location
if holdPod {
    proxyLocation = MoveFirst
}
```

查看 istio-injector 自动注入使用的 template，可以知道如果打开了 `HoldApplicationUntilProxyStarts` 就会为 sidecar 添加一个 postStart hook：

```
{{- if .Values.global.proxy.lifecycle }}
  lifecycle:
    {{ toYaml .Values.global.proxy.lifecycle | indent 4 }}
{{- else if $holdProxy }}
  lifecycle:
    postStart:
      exec:
        command:
        - pilot-agent
        - wait
{{- end }}
```

它的目的是为了阻塞后面的业务容器启动，sidecar 完全启动后才开始启动后面的业务容器。
开关配置分为全局和局部两种，下面介绍两种启用方法。

全局配置

修改 istio 的 configmap 全局配置：

```
kubectl -n istio-system edit cm istio
```

在 `defaultConfig` 下加入 `holdApplicationUntilProxyStarts: true`

```
apiVersion: v1
data:
  mesh: |-
    defaultConfig:
      holdApplicationUntilProxyStarts: true
  meshNetworks: 'networks: {}'
kind: ConfigMap
```

若使用 IstioOperator，defaultConfig 修改 CR 字段 `meshConfig`：

```
apiVersion: install.istio.io/v1alpha1
kind: IstioOperator
metadata:
  namespace: istio-system
  name: example-istiocontrolplane
spec:
  meshConfig:
    defaultConfig:
      holdApplicationUntilProxyStarts: true
```

局部配置

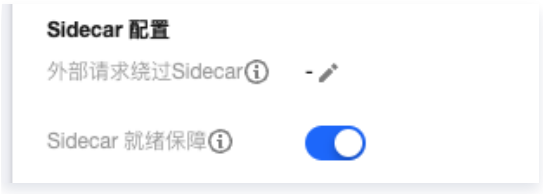
如果使用 istio 1.8 及其以上的版本，可以为需要打开此开关的 Pod 加上 `proxy.istio.io/config` 注解，将 `holdApplicationUntilProxyStarts` 置为 `true`，示例：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
```

```
  template:
    metadata:
      annotations:
        proxy.istio.io/config: |
          holdApplicationUntilProxyStarts: true
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: "nginx"
```

需要注意的是，打开开关后，意味着业务容器需要等 sidecar 完全 ready 后才能启动，会让 Pod 启动速度变慢一些。在需要快速扩容应对突发流量场景可能会显得吃力，所以建议是自行评估业务场景，利用局部配置的方法，只给需要的业务打开此开关。

在 TCM 中对这个进行了产品化，您可以在网格基本信息页的高级设置中，将 Sidecar 就绪保障开关打开，就相当于对所有 Sidecar 默认开启 `holdApplicationUntilProxyStarts: true` 。



Envoy 默认会将 Header 转换为小写

最近更新时间：2023-07-21 10:06:11

Envoy 缺省会把 http header 的 key 转换为小写，例如有一个 http header `Test-Upper-Case-Header: some-value`，经过 envoy 代理后会变成 `test-upper-case-header: some-value`。这个在正常情况下没问题，RFC 2616 规范也说明了处理 HTTP Header 是大小写不敏感的。

依赖大小写的场景

通常 header 转换为小写不存在规范问题，但有些情况下对 header 大小写敏感会存在以下问题，例如：

- 业务解析 header 依赖大小写。
- 使用的 SDK 对 Header 大小写敏感，如读取 `Content-Length` 来判断 response 长度时依赖首字母大写。

Envoy 支持的规则

Envoy 只支持两种规则：

- 全小写（默认使用的规则）。例如：`test-upper-case-header: some-value`
- 首字母大写（默认没有启用）。例如：`Test-Upper-Case-Header: some-value`

如果应用的 http header 的大小写完全没有规律，则无法兼容。例如：`Test-UPPER-CASE-Header: some-value`

规避方案

规避方案为强制指定为 TCP 协议。将服务声明为 TCP 协议，不让 istio 进行七层处理，也就不会更改 http header 大小写了，但需要注意的是同时也会丧失 istio 的七层能力。
如果服务在集群内，可以在 Service 的 port 名称中带上 “tcp” 前缀，如下所示：

```
kind: Service
metadata:
  name: myservice
spec:
  ports:
  - number: 80
    name: tcp-web # 指定该端口协议为 tcp
```

如果服务在集群外，可以通过一个类似如下 ServiceEntry 将服务强制指定为 TCP Service，以避免 envoy 对其进行七层的处理：

```
apiVersion: networking.istio.io/v1alpha3
kind: ServiceEntry
metadata:
  name: qcloud-cos
spec:
  hosts:
  - "private-1251349835.cos.ap-guangzhou.myqcloud.com"
  location: MESH_INTERNAL
  addresses:
  - 169.254.0.47
  ports:
  - number: 80
```

```
    name: tcp
    protocol: TCP
  resolution: DNS
```

最佳实践

如果希望 Envoy 对某些请求开启 Header 首字母大写的规则，可以使用 EnvoyFilter 指定 Header 规则为首字母大写。示例如下：

```
apiVersion: networking.istio.io/v1alpha3
kind: EnvoyFilter
metadata:
  name: http-header-proper-case-words
  namespace: istio-system
spec:
  configPatches:
  - applyTo: NETWORK_FILTER # http connection manager is a filter in Envoy
    match:
      # context omitted so that this applies to both sidecars and gateways
      listener:
        name: XXX # 指定 cos使用的listener name，可以从config_dump中查询到
        filterChain:
          filter:
            name: "envoy.http_connection_manager"
    patch:
      operation: MERGE
      value:
        name: "envoy.http_connection_manager"
        typed_config:
          "@type":
"type.googleapis.com/envoy.config.filter.network.http_connection_manager.v2.HttpConnec
tionManager"
          http_protocol_options:
            header_key_format:
              proper_case_words: {}
```

> **注意**
>
> 需替换 `listener name` 。

建议

应用程序应遵循 RFC 2616 规范，对 Http Header 的处理采用大小写不敏感的原则。

Headless Service 相关问题

最近更新时间：2024-09-03 17:11:11

服务间通过注册中心调用响应 404

现象

传统服务（例如 Spring Cloud）迁移到 istio 后，服务间调用返回 404。

原因

网格没有使用 Kubernetes 的服务发现，而是通过注册中心获取服务 IP 地址，服务间调用不经过域名解析，直接向获取到的目的 IP 发起调用。由于 istio 的 LDS 会拦截 headless service 中包含的 PodIP+Port 的请求，然后匹配请求 hosts，如果没有 hosts 或者 hosts 中没有这个 PodIP+Port 的 service 域名（例如直接是 Pod IP），就会匹配失败，最后返回 404。

解决方案

1. 注册中心不直接注册 Pod IP 地址，注册 service 域名。
2. 客户端请求时带上 hosts（需要更新代码）。

负载均衡策略不生效

由于 istio 默认对 headless service 进行 passthrough，使用 `ORIGINAL_DST` 转发，即直接转发到原始的目的 IP，不做任何的负载均衡，所以 `Destinationrule` 中配置的 `trafficPolicy.loadBalancer` 都不会生效，影响的功能包括：

- 会话保持（consistentHash）
- 地域感知（localityLbSetting）

解决方案

解决方案为单独再创建一个 service（非 headless）。

访问不带 sidecar 的 headless service 失败

现象

client（有sidecar）通过 headless service 访问 server (无sidecar)，访问失败，access log 中可以看到 response_flags 为 `UF,URX`。

原因

istio 1.5/1.6 对 headless service 支持有个 bug，不管 endpoint 有没有 sidecar，都固定启用 mtls，导致没有 sidecar 的 headless 服务(如 redis) 访问被拒绝 (详见 #21964)，更多细节可参考 Istio 运维实战系列（2）：让人头大的『无头服务』-上。

解决方案

方案1： 配置 `DestinationRule` 禁用 mtls

```
kind: DestinationRule
metadata:
  name: redis-disable-mtls
spec:
```

```
  host: redis.default.svc.cluster.local
  trafficPolicy:
    tls:
      mode: DISABLE
```

方案2：升级 istio 到1.7及其以上的版本

Pod 重建后访问失败

现象

client 通过 headless service 访问 server，当 server 的 pod 发生重建后，client 访问 server 失败，access log 中可以看到 response_flags 为 `UF,URX` 。

原因

istio 1.5对 headless service 支持的 bug。

- client 通过 dns 解析 headless service，返回其中一个 Pod IP，然后发起请求。
- envoy 检测到是 headless service，使用 `ORIGINAL_DST` 转发，即不做负载均衡，直接转发到原始的目的 IP。
- 当 headless service 的 pod 发生重建，由于 client 与它的 sidecar (envoy) 是长连接，所以 client 侧的连接并没有断开。
- 又由于是长连接，client 继续发请求并不会重新解析 dns，而是仍然发请求给之前解析到的旧 Pod IP。
- 由于旧 Pod 已经销毁，Envoy 会返回错误 (503)。
- 客户端并不会因为服务端返回错误而断开连接，后续请求继续发给旧的 Pod IP，如此循环，一直失败。
- 更多详情参考 Istio 运维实战系列（3）：让人头大的『无头服务』-下 。

解决方案

升级 istio 到1.6及其以上的版本，Envoy 在 Upstream 链接断开后会主动断开和 Downstream 的长链接。

Istio-init crash

最近更新时间：2023-08-29 10:10:56

在 istio 环境下有 pod 处于 `Init:CrashLoopBackOff` 状态：

```
wk-sys-acl-v1-0-5-7cf7f79d6c-d9qcr                              0/2
Init:CrashLoopBackOff   283        64d     172.16.9.229    10.1.128.6      <none>
<none>
```

查询 istio-init 的日志：

```
Environment:
------------
ENVOY_PORT=
INBOUND_CAPTURE_PORT=
ISTIO_INBOUND_INTERCEPTION_MODE=
ISTIO_INBOUND_TPROXY_MARK=
ISTIO_INBOUND_TPROXY_ROUTE_TABLE=
ISTIO_INBOUND_PORTS=
ISTIO_LOCAL_EXCLUDE_PORTS=
ISTIO_SERVICE_CIDR=
ISTIO_SERVICE_EXCLUDE_CIDR=

Variables:
----------
PROXY_PORT=15001
PROXY_INBOUND_CAPTURE_PORT=15006
PROXY_UID=1337
PROXY_GID=1337
INBOUND_INTERCEPTION_MODE=REDIRECT
INBOUND_TPROXY_MARK=1337
INBOUND_TPROXY_ROUTE_TABLE=133
INBOUND_PORTS_INCLUDE=*
INBOUND_PORTS_EXCLUDE=15090,15021,15020
OUTBOUND_IP_RANGES_INCLUDE=*
OUTBOUND_IP_RANGES_EXCLUDE=
OUTBOUND_PORTS_EXCLUDE=
KUBEVIRT_INTERFACES=
ENABLE_INBOUND_IPV6=false

Writing following contents to rules file:  /tmp/iptables-rules-
1618279687646418248.txt617375845
* nat
-N ISTIO_REDIRECT
-N ISTIO_IN_REDIRECT
-N ISTIO_INBOUND
-N ISTIO_OUTPUT
-A ISTIO_REDIRECT -p tcp -j REDIRECT --to-ports 15001
```

```
-A ISTIO_IN_REDIRECT -p tcp -j REDIRECT --to-ports 15006
-A PREROUTING -p tcp -j ISTIO_INBOUND
-A ISTIO_INBOUND -p tcp --dport 22 -j RETURN
-A ISTIO_INBOUND -p tcp --dport 15090 -j RETURN
-A ISTIO_INBOUND -p tcp --dport 15021 -j RETURN
-A ISTIO_INBOUND -p tcp --dport 15020 -j RETURN
-A ISTIO_INBOUND -p tcp -j ISTIO_IN_REDIRECT
-A OUTPUT -p tcp -j ISTIO_OUTPUT
-A ISTIO_OUTPUT -o lo -s 127.0.0.6/32 -j RETURN
-A ISTIO_OUTPUT -o lo ! -d 127.0.0.1/32 -m owner --uid-owner 1337 -j ISTIO_IN_REDIRECT
-A ISTIO_OUTPUT -o lo -m owner ! --uid-owner 1337 -j RETURN
-A ISTIO_OUTPUT -m owner --uid-owner 1337 -j RETURN
-A ISTIO_OUTPUT -o lo ! -d 127.0.0.1/32 -m owner --gid-owner 1337 -j ISTIO_IN_REDIRECT
-A ISTIO_OUTPUT -o lo -m owner ! --gid-owner 1337 -j RETURN
-A ISTIO_OUTPUT -m owner --gid-owner 1337 -j RETURN
-A ISTIO_OUTPUT -d 127.0.0.1/32 -j RETURN
-A ISTIO_OUTPUT -j ISTIO_REDIRECT
COMMIT

iptables-restore --noflush /tmp/iptables-rules-1618279687646418248.txt617375845
iptables-restore: line 2 failed
iptables-save
# Generated by iptables-save v1.6.1 on Tue Apr 13 02:08:07 2021
*nat
:PREROUTING ACCEPT [5214353:312861180]
:INPUT ACCEPT [5214353:312861180]
:OUTPUT ACCEPT [6203044:504329953]
:POSTROUTING ACCEPT [6203087:504332485]
:ISTIO_INBOUND - [0:0]
:ISTIO_IN_REDIRECT - [0:0]
:ISTIO_OUTPUT - [0:0]
:ISTIO_REDIRECT - [0:0]
-A PREROUTING -p tcp -j ISTIO_INBOUND
-A OUTPUT -p tcp -j ISTIO_OUTPUT
-A ISTIO_INBOUND -p tcp -m tcp --dport 22 -j RETURN
-A ISTIO_INBOUND -p tcp -m tcp --dport 15090 -j RETURN
-A ISTIO_INBOUND -p tcp -m tcp --dport 15021 -j RETURN
-A ISTIO_INBOUND -p tcp -m tcp --dport 15020 -j RETURN
-A ISTIO_INBOUND -p tcp -j ISTIO_IN_REDIRECT
-A ISTIO_IN_REDIRECT -p tcp -j REDIRECT --to-ports 15006
-A ISTIO_OUTPUT -s 127.0.0.6/32 -o lo -j RETURN
-A ISTIO_OUTPUT ! -d 127.0.0.1/32 -o lo -m owner --uid-owner 1337 -j ISTIO_IN_REDIRECT
-A ISTIO_OUTPUT -o lo -m owner ! --uid-owner 1337 -j RETURN
-A ISTIO_OUTPUT -m owner --uid-owner 1337 -j RETURN
-A ISTIO_OUTPUT ! -d 127.0.0.1/32 -o lo -m owner --gid-owner 1337 -j ISTIO_IN_REDIRECT
-A ISTIO_OUTPUT -o lo -m owner ! --gid-owner 1337 -j RETURN
-A ISTIO_OUTPUT -m owner --gid-owner 1337 -j RETURN
-A ISTIO_OUTPUT -d 127.0.0.1/32 -j RETURN
-A ISTIO_OUTPUT -j ISTIO_REDIRECT
-A ISTIO_REDIRECT -p tcp -j REDIRECT --to-ports 15001
```

```
COMMIT
# Completed on Tue Apr 13 02:08:07 2021
panic: exit status 1

goroutine 1 [running]:
istio.io/istio/tools/istio-iptables/pkg/dependencies.
(*RealDependencies).RunOrFail(0x3bb0090, 0x22cfd22, 0x10, 0xc0006849c0, 0x2, 0x2)
    istio.io/istio/tools/istio-iptables/pkg/dependencies/implementation.go:44 +0x96
istio.io/istio/tools/istio-iptables/pkg/cmd.
(*IptablesConfigurator).executeIptablesRestoreCommand(0xc0009dfd68, 0x22c5a01, 0x0,
0x0)
    istio.io/istio/tools/istio-iptables/pkg/cmd/run.go:493 +0x387
istio.io/istio/tools/istio-iptables/pkg/cmd.
(*IptablesConfigurator).executeCommands(0xc0009dfd68)
    istio.io/istio/tools/istio-iptables/pkg/cmd/run.go:500 +0x45
istio.io/istio/tools/istio-iptables/pkg/cmd.(*IptablesConfigurator).run(0xc0009dfd68)
    istio.io/istio/tools/istio-iptables/pkg/cmd/run.go:447 +0x2625
istio.io/istio/tools/istio-iptables/pkg/cmd.glob..func1(0x3b5d680, 0xc0004cce00, 0x0,
0x10)
    istio.io/istio/tools/istio-iptables/pkg/cmd/root.go:64 +0x148
github.com/spf13/cobra.(*Command).execute(0x3b5d680, 0xc0004ccd00, 0x10, 0x10,
0x3b5d680, 0xc0004ccd00)
    github.com/spf13/cobra@v1.0.0/command.go:846 +0x29d
github.com/spf13/cobra.(*Command).ExecuteC(0x3b5d920, 0x0, 0x0, 0x0)
    github.com/spf13/cobra@v1.0.0/command.go:950 +0x349
github.com/spf13/cobra.(*Command).Execute(...)
    github.com/spf13/cobra@v1.0.0/command.go:887
main.main()
    istio.io/istio/pilot/cmd/pilot-agent/main.go:505 +0x2d
```

原因与解决方案

详情请参见 issue 。

直接原因

这种情况应该通常是清理了已退出的 istio-init 容器，导致 k8s 检测到 pod 关联的容器不在了，然后会重新拉起被删除的容器，而 istio-init 的执行不可重入，因为之前已创建了 iptables 规则，导致后拉起的 istio-init 执行 iptables 失败而 crash。

根因与解决方案

清理的动作通常是执行了 `docker container rm` 或 `docker container prune` 或 `docker system prune` 。一般是 crontab 定时脚本里定时清理了容器导致，需要停止清理。

Virtual Service 不生效

最近更新时间：2023-08-29 10:10:56

使用 istio，在集群中定义了 `VirtualService`，但测试发现定义的规则似乎没有生效，通常是一些配置问题，本文列举常见的可能原因。

集群内访问：gateway 字段没有显式指定 “mesh”

如果 `VirtualService` 没有指定 `gateways` 字段，实际隐含了一层意思，istio 会默认加上一个叫 “mesh” 的保留 Gateway，表示集群内部所有 Sidecar，也就表示此 `VirtualService` 规则针对集群内的访问生效。
但如果指定了 `gateways` 字段，istio 不会默认加上 “mesh”，如：

```
apiVersion: networking.istio.io/v1beta1
kind: VirtualService
metadata:
  name: productpage
spec:
  gateways:
  - istio-test/test-gateway
  hosts:
  - bookinfo.example.com
  http:
  - route:
    - destination:
        host: productpage
        port:
          number: 9080
```

这个表示此 `VirtualService` 规则仅对 `istio-test/test-gateway` 这个 Gateway 生效，如果是在集群内访问，流量不会经过这个 Gateway，所以此规则也就不会生效。
那如果要同时在集群内也生效该怎么做呢？答案是给 `gateways` 显式指定上 “mesh”：

```
  gateways:
  - istio-test/test-gateway
  - mesh
```

表示此 `VirtualService` 不仅对 `istio-test/test-gateway` 这个 Gateway 生效，也对集群内部访问生效。
参考 istio 官方文档 对此字段的解释：

gateways	string[]	The names of gateways and sidecars that should apply these routes. Gateways in other namespaces may be referred to by <gateway namespace>/<gateway name>; specifying a gateway with no namespace qualifier is the same as specifying the VirtualService's namespace. A single VirtualService is used for sidecars inside the mesh as well as for one or more gateways. The selection condition imposed by this field can be overridden using the source field in the match conditions of protocol-specific routes. The reserved word mesh is used to imply all the sidecars in the mesh. When this field is omitted, the default gateway (mesh) will be used, which would apply the rule to all sidecars in the mesh. If a list of gateway names is provided, the rules will apply only to the gateways. To apply the rules to both gateways and sidecars, specify mesh as one of the gateway names.

值得注意的是，从集群内访问一般是直接访问 service 名称，这里 `hosts` 就需要加上访问集群内的 service 名称：

```
hosts:
- bookinfo.example.com
```

```
- productpage
```

通过 ingressgateway 访问: hosts 定义错误

若从 ingressgateway 访问，需要确保 `Gateway` 和 `VirtualService` 中的 hosts 均包含实际访问用到的 `Host` 或使用通配符能匹配得上，通常是外部域名。
只要有一方 hosts 没定义正确，都可能导致 `404 Not Found`，正确示例：

```
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: test-gateway
  namespace: istio-test
spec:
  selector:
    app: istio-ingressgateway
    istio: ingressgateway
  servers:
  - port:
      number: 80
      name: HTTP-80-www
      protocol: HTTP
    hosts:
    - bookinfo.example.com # 这里定义外部访问域名

---

apiVersion: networking.istio.io/v1beta1
kind: VirtualService
metadata:
  name: productpage
spec:
  gateways:
  - istio-test/test-gateway
  hosts:
  - bookinfo.example.com  # 这里也要定义外部访问域名
  http:
  - route:
    - destination:
        host: productpage
        port:
          number: 9080
```

Virtual Service 路由匹配顺序问题

最近更新时间：2023-08-29 10:10:56

在写 VirtualService 路由规则时，通常会 match 各种不同路径转发到不同的后端服务，如果不小心命名冲突了，导致始终只匹配到前面的服务，例如：

```
apiVersion: networking.istio.io/v1beta1
kind: VirtualService
metadata:
  name: test
spec:
  gateways:
  - default/example-gw
  hosts:
  - 'test.example.com'
  http:
  - match:
    - uri:
        prefix: /usrv
    rewrite:
      uri: /
    route:
    - destination:
        host: usrv.default.svc.cluster.local
        port:
          number: 80
  - match:
    - uri:
        prefix: /usrv-expand
    rewrite:
      uri: /
    route:
    - destination:
        host: usrv-expand.default.svc.cluster.local
        port:
          number: 80
```

istio 匹配是按顺序匹配，不像 nginx 那样使用最长前缀匹配。这里使用 prefix 进行匹配，第一个是 `/usrv` ，表示只要访问路径前缀含 `/usrv` 就会转发到第一个服务，由于第二个匹配路径 `/usrv-expand` 本身也属于带 `/usrv` 的前缀，所以永远不会转发到第二个匹配路径的服务。

解决方案

这种情况可以调整下匹配顺序，如果前缀有包含的冲突关系，越长的放在越前面：

```
apiVersion: networking.istio.io/v1beta1
kind: VirtualService
metadata:
```

```
  name: test
spec:
  gateways:
  - default/example-gw
  hosts:
  - 'test.example.com'
  http:
  - match:
    - uri:
        prefix: /usrv-expand
    rewrite:
      uri: /
    route:
    - destination:
        host: usrv-expand.default.svc.cluster.local
        port:
          number: 80
  - match:
    - uri:
        prefix: /usrv
    rewrite:
      uri: /
    route:
    - destination:
        host: usrv.default.svc.cluster.local
        port:
          number: 80
```

也可以用正则匹配：

```
apiVersion: networking.istio.io/v1beta1
kind: VirtualService
metadata:
  name: test
spec:
  gateways:
  - default/gateway
  hosts:
  - 'test.example.com'
  http:
  - match:
    - uri:
        regex: "/usrv(/.*)?"
    rewrite:
      uri: /
    route:
    - destination:
        host: nginx.default.svc.cluster.local
        port:
          number: 80
```

```
        subset: v1
  - match:
    - uri:
        regex: "/usrv-expand(/.*)?"
    rewrite:
      uri: /
    route:
    - destination:
        host: nginx.default.svc.cluster.local
        port:
          number: 80
        subset: v2
```

公网 Ingress Gateway 不通

最近更新时间：2024-03-20 10:27:31

从公网访问 ingressgateway 不通。

常见原因

安全组没放通 NodePort

TCM 安装的 ingressgateway，暴露方式默认使用 CLB 绑定节点 NodePort，所以流量链路是：client > CLB > NodePort > ingressgateway。
关键链路在于 CLB > NodePort，CLB 转发的数据包不会做 SNAT，所以报文到达节点时源 IP 就是 client 的公网 IP，如果节点安全组入站规则没有放通 client > NodePort 链路的话，是访问不通的。
解决方案1：放通节点安全组入站规则对公网访问 NodePort 区间端口（30000-32768）。

解决方案2：若担心直接放开整个 NodePort 区间所有端口有安全风险，可以只暴露 ingressgateway service 所用到的 Nodeport。
解决方案3：若只允许固定 IP 段的 client 访问 ingressgateway，可以只对这个 IP 段放开整个 NodePort 区间所有端口。
解决方案4：为 ingressgateway 启用 CLB 直通 Pod，这样流量就不经过 NodePort，所以就没有此安全组问题。启用 CLB 直通 Pod 需要集群网络支持 VPC-CNI，详细请参见 如何启用 CLB 直通 Pod 。

启用 Smart DNS 后解析失败

最近更新时间：2023-08-29 10:10:56

在启用了 istio 的 Smart DNS（智能 DNS）后，我们发现有些情况下 DNS 解析失败，例如：

- 基于 alpine 镜像的容器内解析 dns 失败。
- grpc 服务解析 dns 失败。

原因

Smart DNS 初期实现存在一些问题，响应的 DNS 数据包格式跟普通 DNS 有些差别，走底层库 glibc 解析没问题，但使用其它 dns 客户端可能就会失败：

- alpine 镜像底层库使用 musl libc，解析行为跟 glibc 有些不一样，musl libc 在这种这种数据包格式异常的情况会导致解析失败，而大多应用走底层库解析，导致大部分应用解析失败。
- 基于 c/c++ 的 grpc 框架的服务，dns 解析默认使用 c-ares 库，没有走系统调用让底层库解析，c-ares 在这种数据包异常情况，部分场景会解析失败。

修复

在 istio 1.9.2 的时候修复了这个问题，参考关键 PR #31251 以及其中一个 issue 。

规避

如果暂时无法升级 istio 到 1.9.2 以上，可以通过以下方式来规避：

- 基础镜像从 alpine 镜像到其它镜像 (其它基础镜像底层库基本都是 glibc)。
- c/c++ 的 grpc 服务，指定 `GRPC_DNS_RESOLVER` 环境变量为 `native`，表示走底层库解析，不走默认的 c-ares 库。环境变量解释请参见 GRPC 官方文档 。

地域感知不生效

最近更新时间：2023-08-29 10:10:57

使用 istio 地域感知能力时，测试发现没生效，本文介绍几个常见原因。

DestinationRule 未配置 outlierDetection

地域感知默认开启，但还需要配置 DestinationRule，且指定 `outlierDetection` 才可以生效，指定这个配置的作用主要是让 istio 感知 endpoints 是否异常，当前 locality 的 endpoints 发生异常时会 failover 到其它地方的 endpoints。
配置示例：

```
apiVersion: networking.istio.io/v1beta1
kind: DestinationRule
metadata:
  name: nginx
spec:
  host: nginx
  trafficPolicy:
    outlierDetection:
      consecutive5xxErrors: 3
      interval: 30s
      baseEjectionTime: 30s
```

client 未配置 service

istio 控制面会为每个数据面单独下发 EDS，不同数据面实例(Envoy)的locality可能不一样，生成的 EDS 也就可能不一样。istio 会获取数据面的locality信息，获取方式主要是找到数据面对应的 endpoint 上保存的 region、zone 等信息，如果 client 没有任何 service，也就不会有 endpoint，控制面也就无法获取 client 的 locality 信息，也就无法实现地域感知。

解决方案

为 client 配置 service，selector 选中 client 的 label。如果 client 本身不对外提供服务，service 的 ports 也可以随便定义一个。

使用了 headless service

如果是访问 headless service，本身是不支持地域感知的，因为 istio 会对 headless service 请求直接 passthrough，不做负载均衡，客户端会直接访问到 dns 解析出来的 pod ip。

解决方案

单独再创建一个 service（非 headless）。

没有自动注入Sidecar

最近更新时间：2023-08-29 10:10:57

通过 `kubectl label namespace xxx istio-injection=enabled` 为某个 namespace 开启 sidecar 自动注入，发现不生效。

TCM 自动注入的 label

TCM 1.6 及其以上的版本并不是用 `istio-injection=enabled` 这个 label 来开启 namespace 的自动注入，而是用类似 `istio.io/rev=1-8-1` 这样的 label，根据不同的版本不一样：

- 1.6 是 `istio.io/rev=1-6-9`
- 1.8 是 `istio.io/rev=1-8-1`

所以如果用社区常见的 `istio-injection=enabled` 这样的 label，在 TCM 上是不会生效的，以下为不使用社区常见的 label 的原因。

网格灰度升级需使用不同 label

为了支持服务网格的灰度升级，就需要新旧两个控制面共存一段时间，让旧的数据面逐渐滚动更新升级并连到新的控制面，如果两个控制面都使用同一个 label 来识别是否需要自动注入，在滚动更新的过程中，两个控制面都去注入 sidecar，就会造成冲突。所以 istio 官方给出了 灰度升级方案 ，正是基于不同控制面使用不同 label 的方法来实现多控制面共存，这也是 TCM 所采用的方案。

TCM 如何开启自动注入

在服务网格控制台，选择**服务>sidecar自动注入**。如下图所示：

然后勾选需要开启自动注入的命名空间即可。
也可以通过使用 kubectl 给 namespace 打 label 来开启:

```
kubectl label namespace xxx istio.io/rev=1-8-1
```

> **注意**
> 注意替换 namespace 以及根据版本替换对应的 label。

熔断不生效

最近更新时间：2023-08-29 10:10:57

未定义 http1MaxPendingRequests 导致熔断不生效

我们给 DR 配置了 `maxConnections`：

```
apiVersion: networking.istio.io/v1beta1
kind: DestinationRule
metadata:
  name: nginx
spec:
  host: nginx
  trafficPolicy:
    connectionPool:
      tcp:
        maxConnections: 1
```

但测试当并发超过这里定义的最大连接数时，并没有触发熔断，只是 QPS 很低。通常是因为没有配置 `http1MaxPendingRequests`，不配置默认为 `2^32-1`，非常大，表示如果超过最大连接数，请求就先等待（不直接返回503），当连接数低于最大值时再继续转发。
如果希望连接达到上限或超过上限一定量后或直接熔断（响应503），就需要显式指定 `http1MaxPendingRequests`：

```
apiVersion: networking.istio.io/v1beta1
kind: DestinationRule
metadata:
  name: nginx
spec:
  host: nginx
  trafficPolicy:
    connectionPool:
      tcp:
        maxConnections: 1
      http:
        http1MaxPendingRequests: 1
```

重试策略导致服务异常

最近更新时间：2023-08-29 10:10:57

Istio 为 Envoy 设置了缺省的重试策略，会在 connect-failure, refused-stream, unavailable, cancelled, retriable-status-codes 等情况下缺省重试两次。出现错误时，可能已经触发了服务器逻辑，在操作不是幂等（任意多次执行所产生的影响均与一次执行的影响相同）的情况下，可能会导致错误。

解决方案

可以通过配置 VS 关闭重试：

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: ratings
spec:
  hosts:
  - ratings
  http:
  - retries:
      attempts: 0
```

链路追踪信息不完整

最近更新时间：2023-08-29 10:10:57

通过 UI 展示的链路追踪显示不完整，缺失前面或后面的调用链路。

原因

绝大多数情况下都是因为没在业务层面将 tracing 所需要的 http header 正确传递或根本没有传递。
要在 istio 中使用链路追踪，并不是说业务无侵入，有个基本要求是：业务收到 tracing 相关的 header 要将其传递给被调服务。这个步骤是无法让 istio 帮您完成的，因为 istio 无法感知您的业务逻辑，不知道业务中调用其它服务的请求到底是该对应前面哪个请求，所以需要业务来传递 header，最终才能将链路完整串起来。