《Istio 大咖说》第 8 期

小红书服务网格大规模落地经验分享

主持人：宋净超（Tetrate）

嘉宾：贾建云（小红书）

11 月 9 日（周二）

晚 8:00 - 9:00

联合主办方：tetrate 云原生社区 Cloud Native Community ServiceMesher

1、小红书服务网格介绍

2、小红书对服务网格的增强

3、关于落地服务网格的一些坑

小红书服务网格介绍 — 为什么引入ServiceMesh

1、中间件升级周期太长，且依赖业务更新基础库，无法做到业务无感知

2、内部语言众多，协议不统一。小语种甚至没有SDK，无法享受公司基础设施

小红书服务网格介绍 — 落地Mesh要解决的问题

1、流量拦截方式需要增强。解决性能和fallback问题

2、控制面性能问题。解决大规模集群频繁变更情况下推送效率问题

3、sidecar管理。解决热升级问题

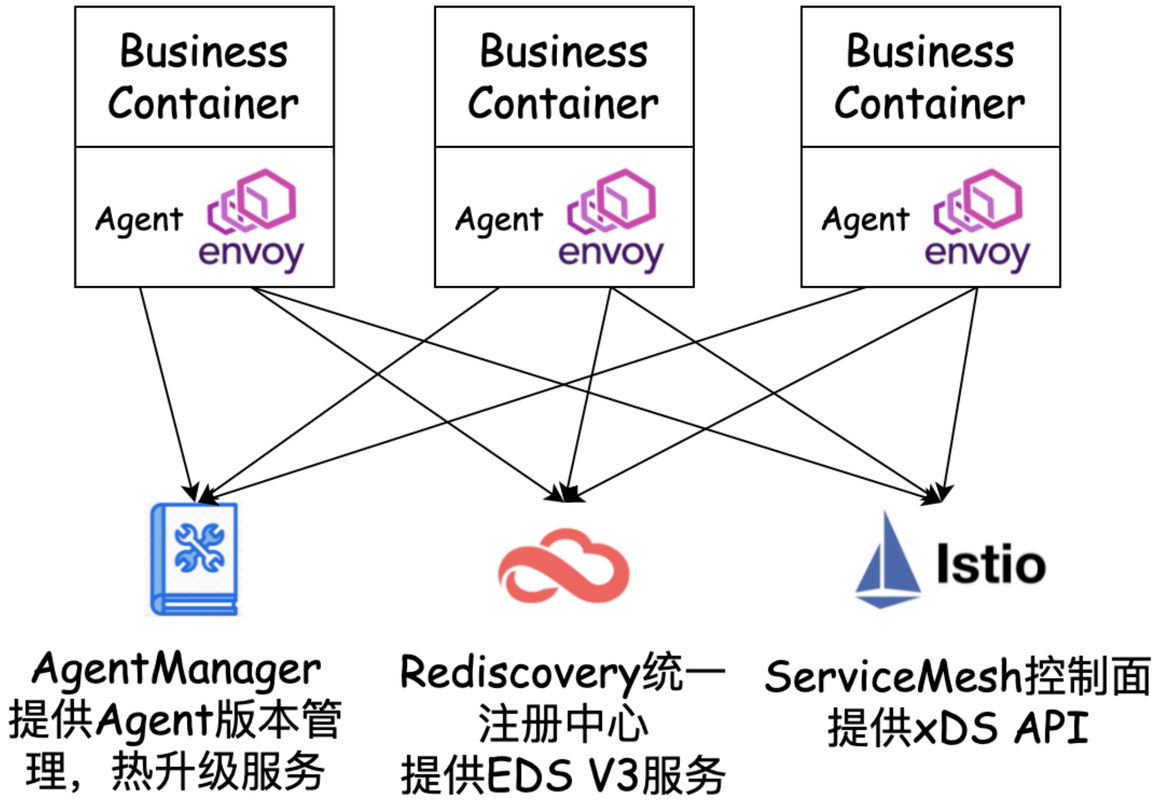
4、thrift扩展。如何适配内部thrift协议，增强thrift filter能力

5、envoy数据面性能增强。长尾、延迟敏感业务

6、产品化。提升用户体检

……

小红书服务网格介绍 — 整体架构和规模



线上1500+服务，5w+ pod

预计年底支撑2000+服务，10w+ pod

对接注册中心优势：

1、降低mesh控制面负载，提升eds推送效率

2、复用公司注册中心能力

AgentManager运维面：

1、实现envoy热升级

小红书对服务网格的增强

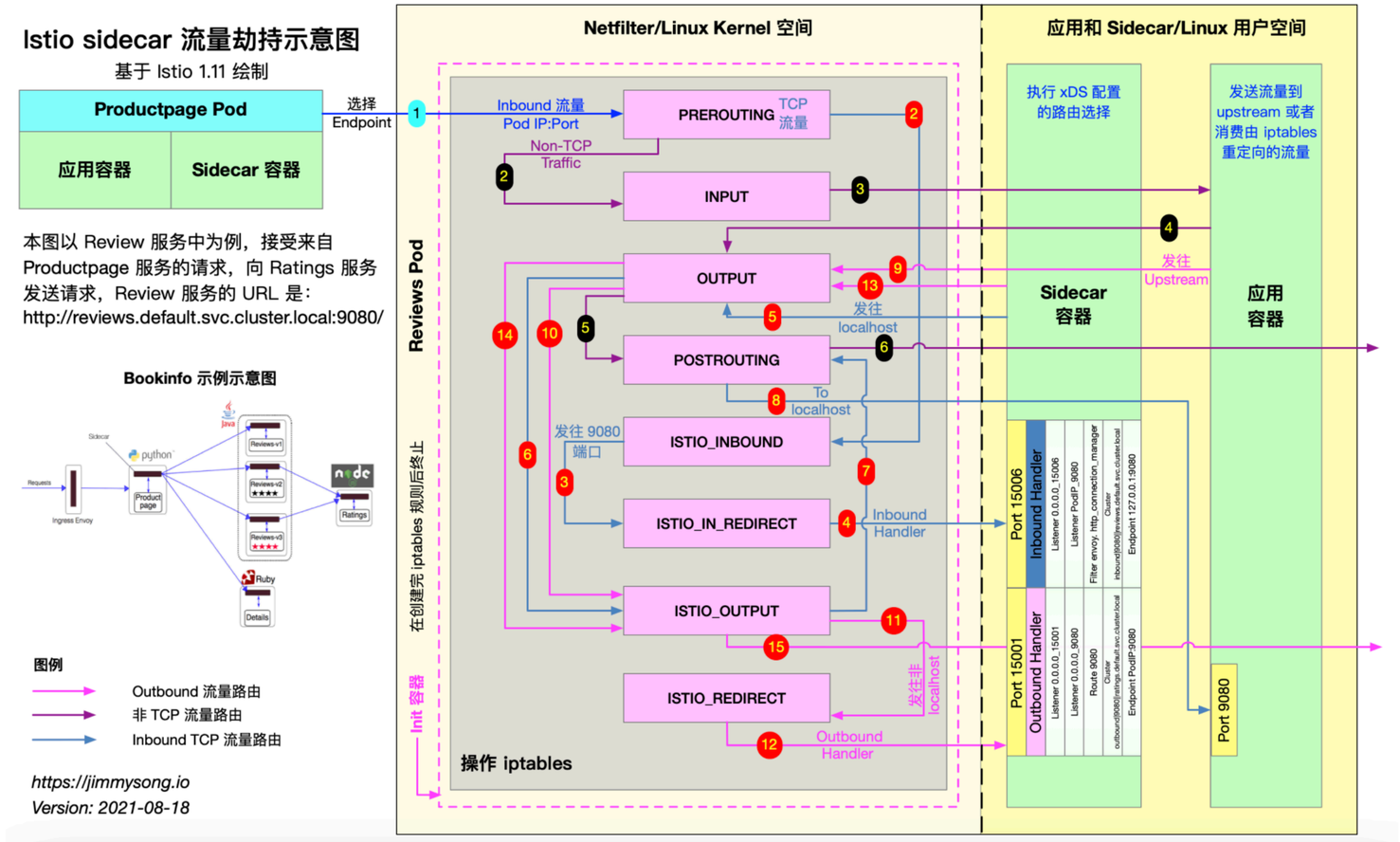
已经完成：

- 流量拦截
- 懒加载
- thrift扩展
- sidecar热升级
- 控制面优化
- 客户端健康检查(#35325)
- Sidecar与业务容器资源共享
- 支持pilot crd多集群读写

正在做：

- 对接SkyWalking
- 配置灰度下发
- 熔断/限流
- 故障注入
- ebpf 加速容器间网络传输
- 优雅退出
- 动态负载均衡
- 引入brpc，提升envoy性能

小红书对服务网格的增强 — 流量拦截(iptables)



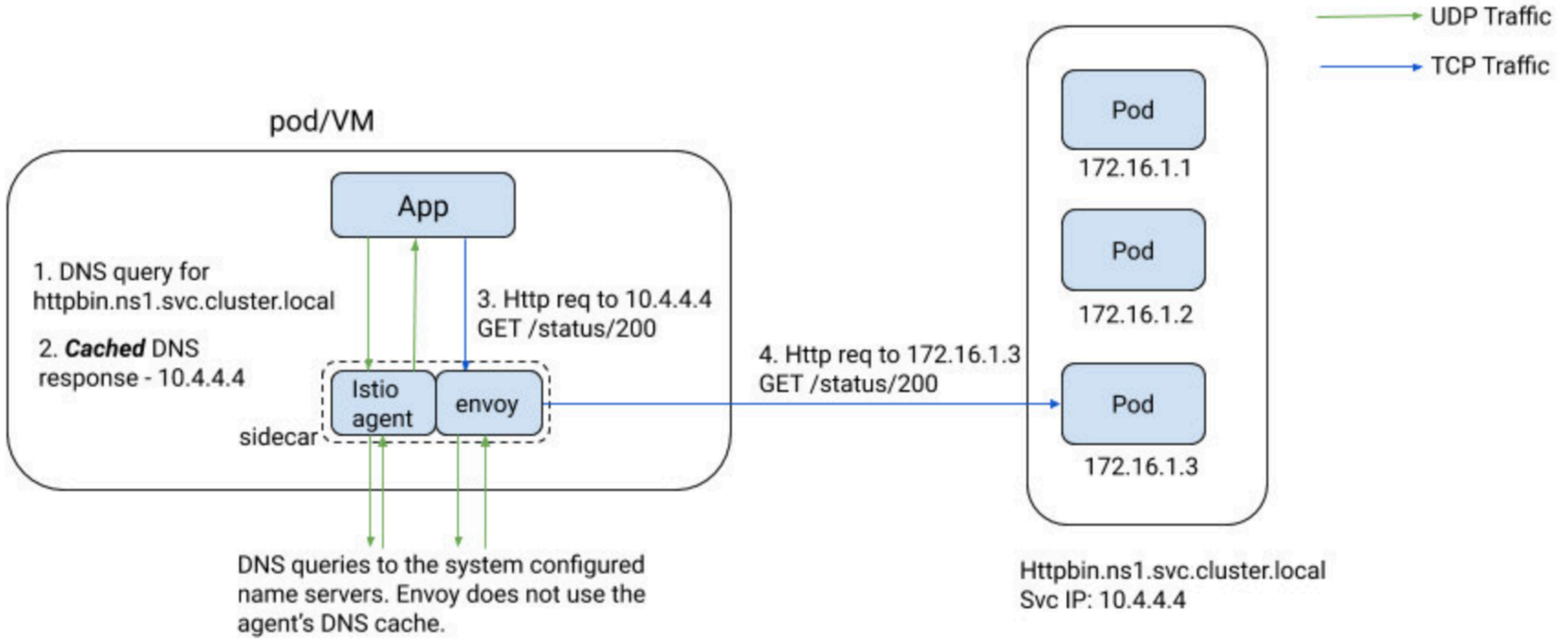
性能损耗

无法优雅fallback

必须引入smart dns

图片来源：https://jimmysong.io/blog/sidecar-injection-iptables-and-traffic-routing/

小红书对服务网格的增强 — 流量拦截(iptables)



例如：client 想要访问 server-a 服务，以http请求举例，具体的流程如下：

- 1）所有的服务（ServiceEntry）在网格控制面中已经分配了一个保留网段的VIP，该VIP 类似于服务的ID。
- 2）client 通过 http://server-a/getProduct url 发起访问
- 3）首先系统自动做dns解析。解析请求被拦截到sidecar dns ，sidecar dns 找到server-a对应的VIP（10.4.4.4）并返回。
- 4）然后client 通过 http://10.4.4.4/getProduct 访问
- 5）该访问被iptables（ tproxy） 模式拦截到envoy监听器
- 6）envoy 通过VIP 可以确定访问的是server-a 服务，然后通过设置好的负载均衡算法，均衡到server-a的具体某个实例。

小红书对服务网格的增强 — 流量拦截(懒加载)

服务名称	实例数量	依赖服务	操作
[redacted]	0(running) 0(error)	[redacted]countservice(20015) [redacted]merservice(20014) [redacted]nter(20016)	详情 编辑 监控 删除
[redacted]	2(running) 0(error)	[redacted]d-v1(8097)	详情 编辑 监控 删除
[redacted]	1(running) 0(error)	[redacted]merservice(20014)	详情 编辑 监控 删除
[redacted]	2(running) 0(error)	[redacted]ault(20058)	详情 编辑 监控 删除
test	0(running) 0(error)	nike(20006)	详情 编辑 监控 删除

小红书对服务网格的增强 — 流量拦截(懒加载)

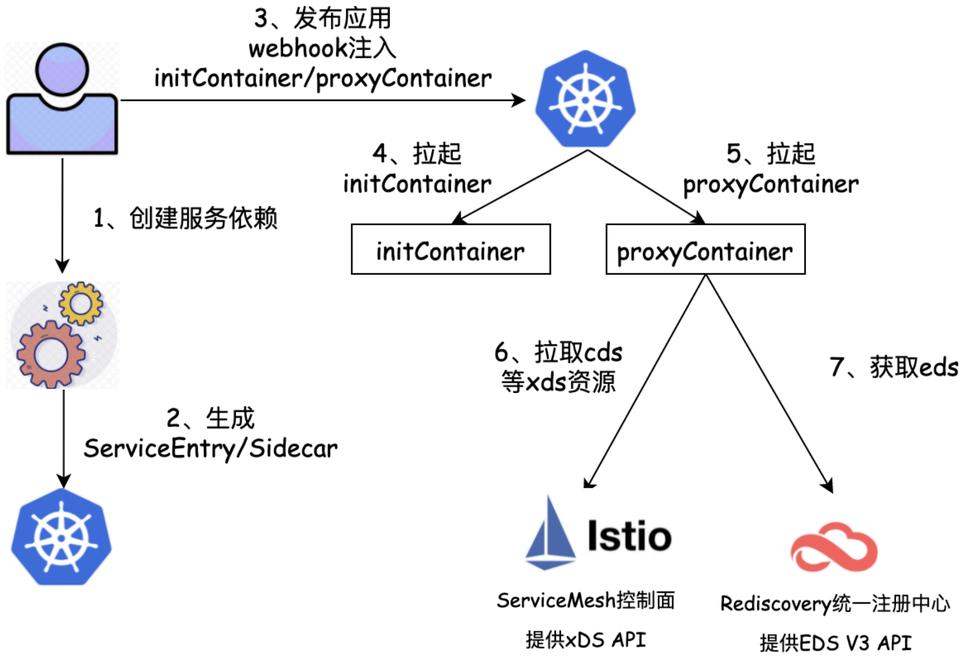
```
spec:
  egress:
  - bind: 0.0.0.0
    captureMode: NONE
    hosts:
    - istio-registry/[redacted]
    port:
      name: tcpport
      number: 20016
      protocol: TCP
  - bind: 0.0.0.0
    captureMode: NONE
    hosts:
    - istio-registry/[redacted]
    port:
      name: tcpport
      number: 20015
      protocol: TCP
  - bind: 0.0.0.0
    captureMode: NONE
    hosts:
    - istio-registry/[redacted]
    port:
      name: tcpport
      number: 20014
      protocol: TCP
  workloadSelector:
    labels:
      sd.xhs.com/service: [redacted]
```

只拦截出流量
不拦截入流量

优点：
减少路径，提升性能
方便做fallback
方便提供debug环境
简单支持虚机

缺点：
入流量指标缺失
限流不好做

小红书对服务网格的增强 — 流量拦截(懒加载)



webhook会注入一个volume：/etc/mesh
然后initContainer会此路径下面写入一个文件
mesh-config.json，内容如下：

CONTAINER: [redacted]

```
# pwd
/etc/mesh
# more mesh-config.json
{"currentService":"[redacted]","dependServices":{"[redacted]":"20058"}}
```

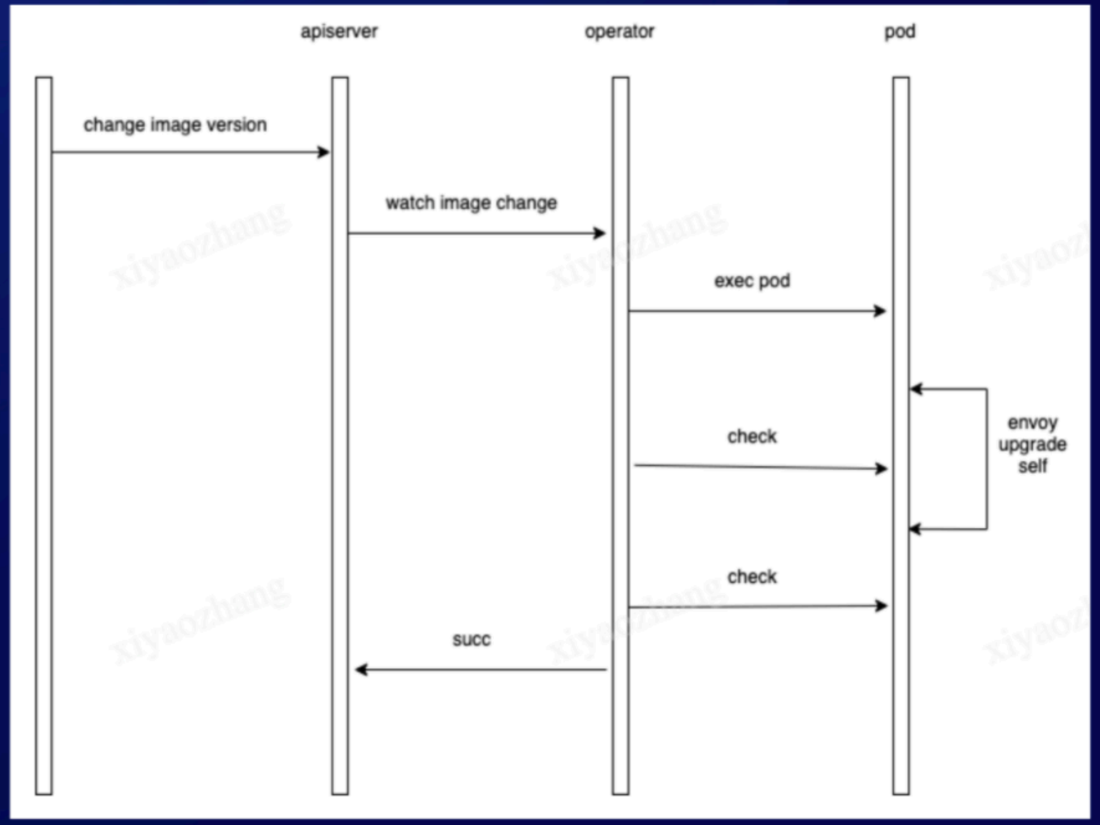
小红书对服务网格的增强 — 热升级

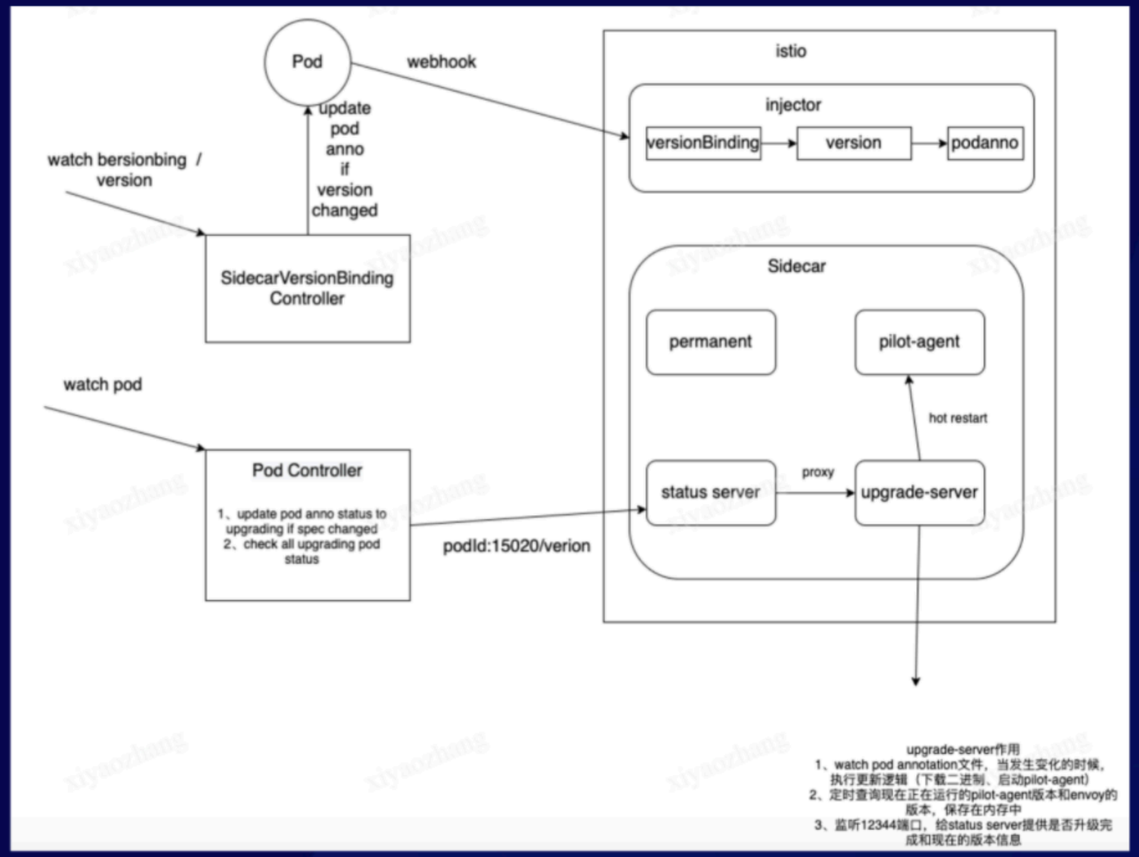
方案	厂商	缺点
依赖业务方重新发版本	华为	不灵活，大规模更新/bugfix阻力大
双容器	阿里/美团	方案复杂，资源分配不好处理
进程级别热升级	头条/美团早期版本/腾讯	需要额外开发运维面

数据面管理

- *Tcm-sidecar-manager**

感知sidecar版本与健康状态，支持sidecar热升级



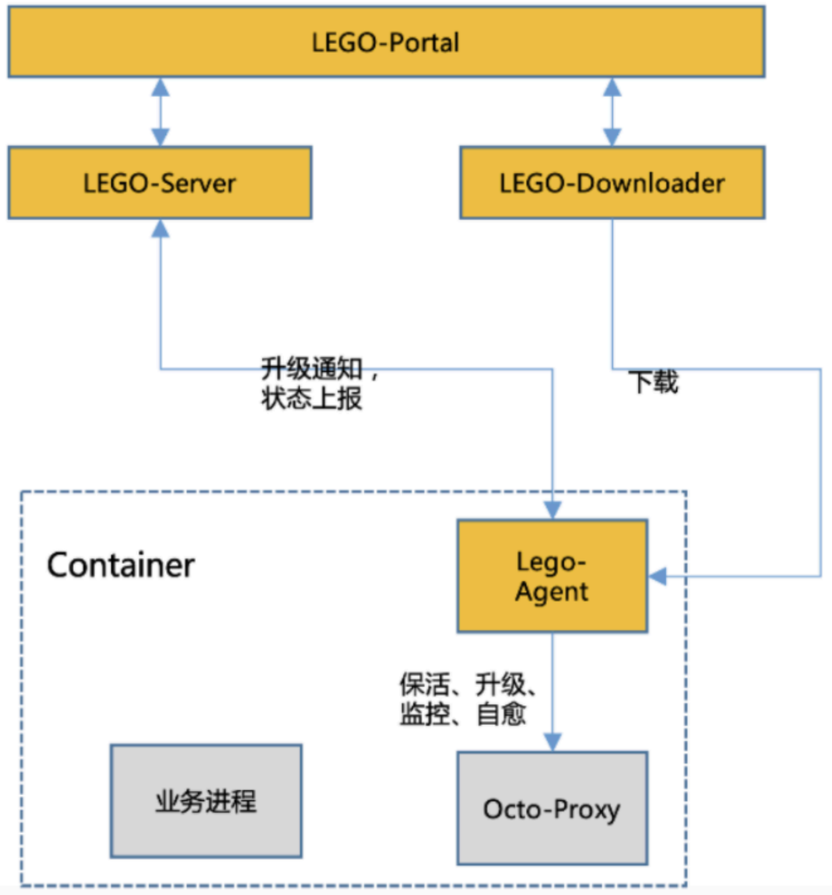


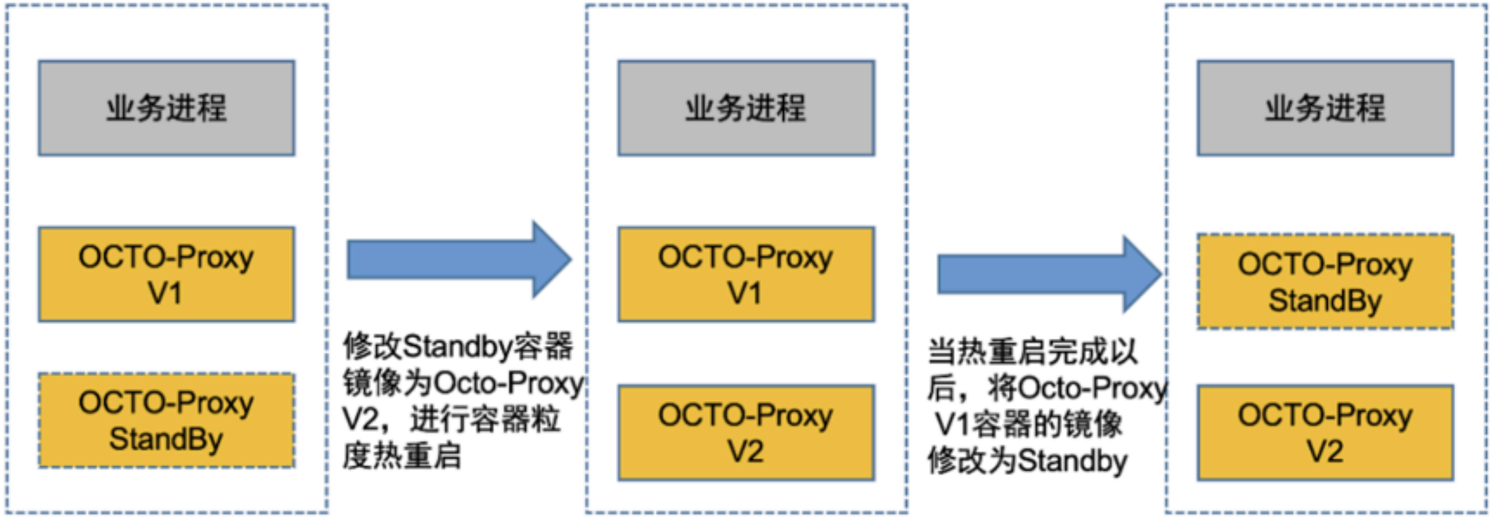
1. *修改envoy的二进制下载地址*
2. *Watch envoy版本变化，对pod执行exec操作，下载替换envoy二进制*
3. *pilot-agent 执行restart操作，执行envoy热升级*
4. *检查进程状态，更新pod状态*

1. *Sidecar version 定义envoy目的版本*
2. *SidecarVersionBinding定义应用的业务与升级的策略和周期*

云原生环境中，Envoy运行在标准的K8S Pod中，通常会独立出一个Sidecar容器。可以使用K8s提供的能力来完成对Envoy Sidecar容器的管理，例如容器注入、健康检查、滚动升级、资源限制等。

美团内部所使用的容器运行时模式为“单容器模式”，即一个Pod内只包含一个容器（不考虑Pause容器）。由于业务进程以及所有的基础组件都运行在一个容器中，所以只能采用进程粒度的管理措施，无法做到容器粒度的管理。我们通过自研的LEGO平台解决了OCTO-Proxy的运维问题。





1. 首先，我们在启动Pod时，给OCTO-Proxy不再只分配一个容器，而是分配两个容器，其中一个容器为正常运行状态，另一个容器为standby状态。
2. 当需要对OCTO-Proxy进行热重启升级时，我们修改standby容器的镜像为最新OCTO-Proxy的镜像，此时开始热重启流程。
3. 当热重启流程结束后，新容器进入正常工作状态，而旧容器进入等待状态，最终，我们将旧容器的镜像修改为standby镜像，结束整个热重启流程。

该方案利用一开始就驻留双容器在Pod内，避免了K8s对运行中Pod不能增删容器的限制，实现了在Pod中对OCTO-Proxy进行容器粒度的热重启。

小红书对服务网格的增强 — 热升级(sidecarset)

1、由于热升级依赖于epoch参数，双容器情况参数传递和维护需要解决

2、由于新老进程交替过程，老envoy进程不能退出，因此需要某种机制保障新envoy进程通知老envoy进程退出后，sidecarset再进行reset镜像的操作

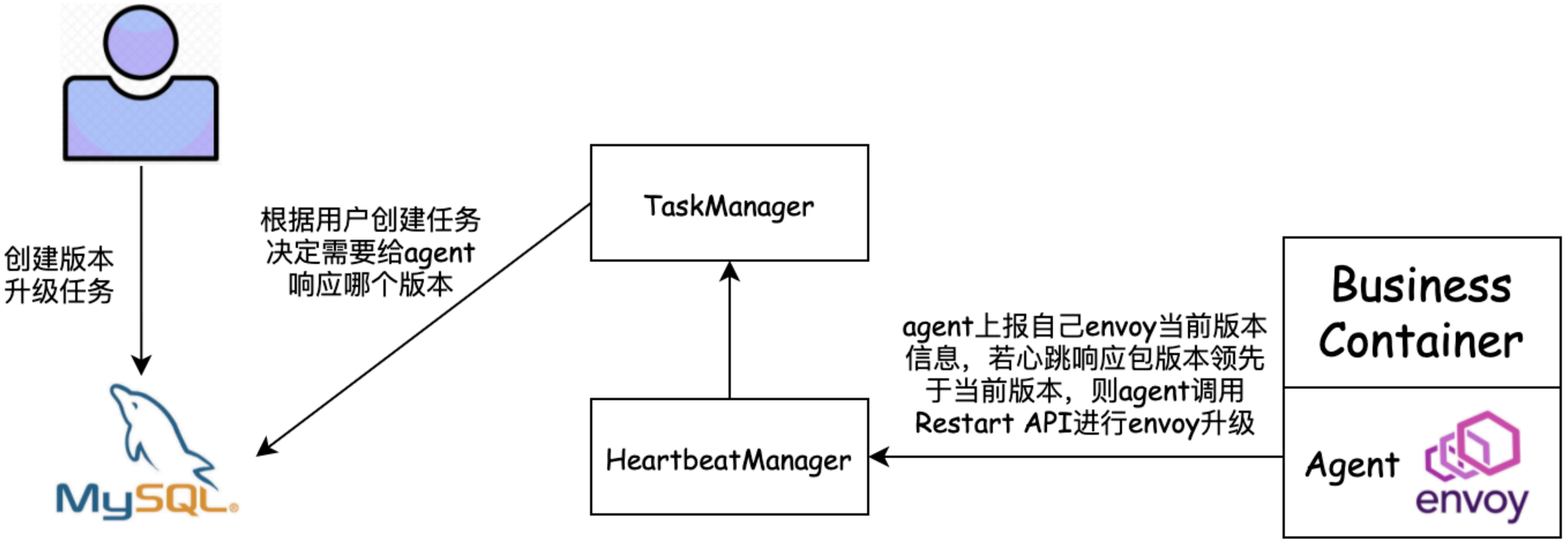
3、pilot-agent本身会有一些端口，升级过程中会出现端口冲突

4、empty镜像启动参数、hook与envoy容器一模一样，需要额外定制empty镜像

5、按照超卖实现的资源共享在双容器情况下面资源配比不好解决

参考文章：https://www.kubernetes.org.cn/9411.html

小红书对服务网格的增强 — sidecar热升级(进程级别)



小红书对服务网格的增强 — 资源共享

非SO服务

用户申请资源信息：

Request：8C 16G

Limit：8C 16G

Business Container	Request 4C 12G Limit 8C 16G
envoy	Request 4C 4G Limit 4C 4G

1、sidecar cpu request = 业务申请量 / 2
sidecar cpu limit = 业务申请量 / 2

2、sidecar mem request = min（业务容器 mem limit / 4， 固定值）
sidecar mem limit = min（业务容器 mem limit / 4， 固定值）

PS：内存是不可压缩资源，所以内存限制上做了4这个系数的处理

小红书对服务网格的增强 — 资源共享

S0服务

用户申请资源信息：

Request： 8C 16G

Limit： 8C 16G

Business Container	Request 4C 16G Limit 4C 16G
	Request 4C 4G Limit 4C 4G

1、通过Damon进程修改Container Cgroup cpu.shares为-1，从而让所有容器共享POD Crgoup节点配置实现共享CPU

2、envoy的资源分配作用：

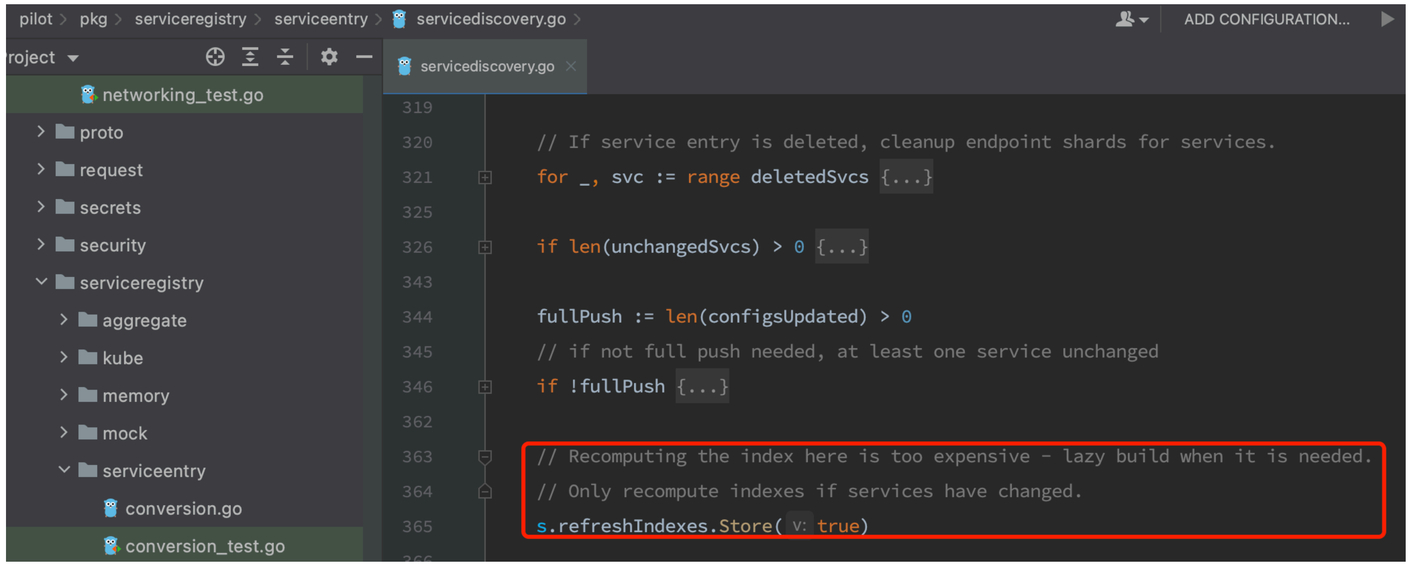
1、要保证POD为Guaranteed类型

2、保证CPU总和等于用户申请的资源。内存是不可压缩资源，超卖4G

关于落地服务网格的一些坑

pilot性能问题（#33485）
pilot回调bug（#33867）
pilot忽略远端集群POD实际端口（#33366）
Pilot-agent Restart API BUG（#35740）
主机网络导致端口冲突
envoy thrift filter更新route导致listener重建
envoy thrift filter不透传flag字段

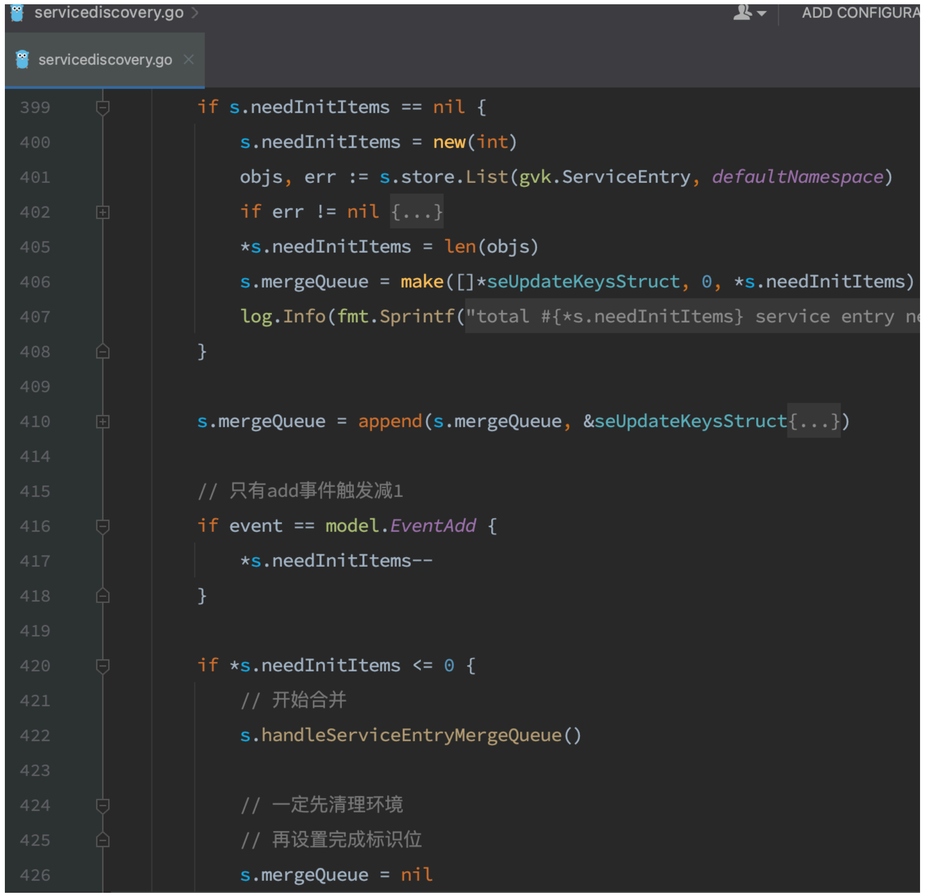
关于落地服务网格的一些坑 — pilot性能问题



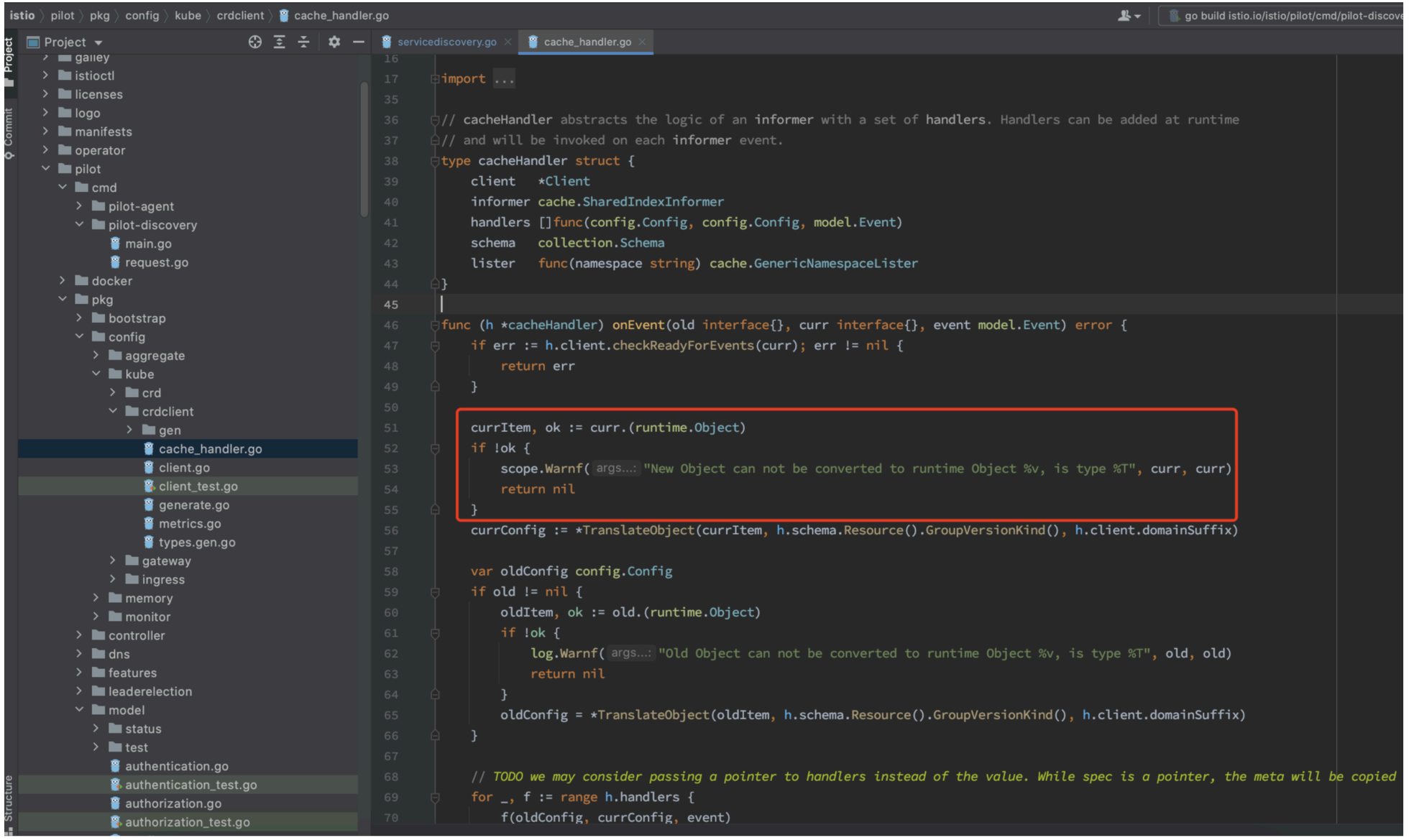

```go
// maybeRefreshIndexes will iterate all ServiceEntries, convert to ServiceInstance (expensive),
// and populate the 'by host' and 'by ip' maps, if needed.
func (s *ServiceEntryStore) maybeRefreshIndexes() {
    // Without this pilot becomes very unstable even with few 100 ServiceEntry objects
    // - the N_clusters * N_update generates too much garbage ( yaml to proto)
    // This is reset on any change in ServiceEntries that needs index recomputation.
    if !s.refreshIndexes.Load() {
    defer s.refreshIndexes.Store( v: false)

    s.doRefreshIndexes()
}
```

优化代码：



关于落地服务网格的一些坑 — pilot回调bug



关于落地服务网格的一些坑 — 远端集群POD实际端口

```go
func convertWorkloadInstanceToServiceInstance(workloadInstance *model.IstioEndpoint, serviceEntryServices []*model.Service,
    serviceEntry *networking.ServiceEntry) []*model.ServiceInstance {
    out := make([]*model.ServiceInstance, 0)
    for _, service := range serviceEntryServices {
        for _, serviceEntryPort := range serviceEntry.Ports {
            ep := *workloadInstance
            ep.ServicePortName = serviceEntryPort.Name
            // if target port is set, use the target port else fallback to the service port
            // TODO: we need a way to get the container port map from k8s
            if serviceEntryPort.TargetPort > 0 {
                ep.EndpointPort = serviceEntryPort.TargetPort
            } else {
                ep.EndpointPort = serviceEntryPort.Number
            }
            ep.EnvoyEndpoint = nil
            out = append(out, &model.ServiceInstance{
                Endpoint:    &ep,
                Service:     service,
                ServicePort: convertPort(serviceEntryPort),
            })
        }
    }
    return out
}
```

```go
func convertEndpoint(service *model.Service, servicePort *networking.Port,
    endpoint *networking.WorkloadEntry, ck *configKey) *model.ServiceInstance {

    var (
        instancePort uint32
    )

    addr := endpoint.GetAddress()
    if strings.HasPrefix(addr, model.UnixAddressPrefix) {
        instancePort = 0
        addr = strings.TrimPrefix(addr, model.UnixAddressPrefix)
    } else if len(endpoint.Ports) > 0 {
        instancePort = endpoint.Ports[servicePort.Name]
        if instancePort == 0 {
            instancePort = servicePort.Number
        }
    } else if servicePort.TargetPort > 0 {
        instancePort = servicePort.TargetPort
    } else {
        instancePort = servicePort.Number
    }
```

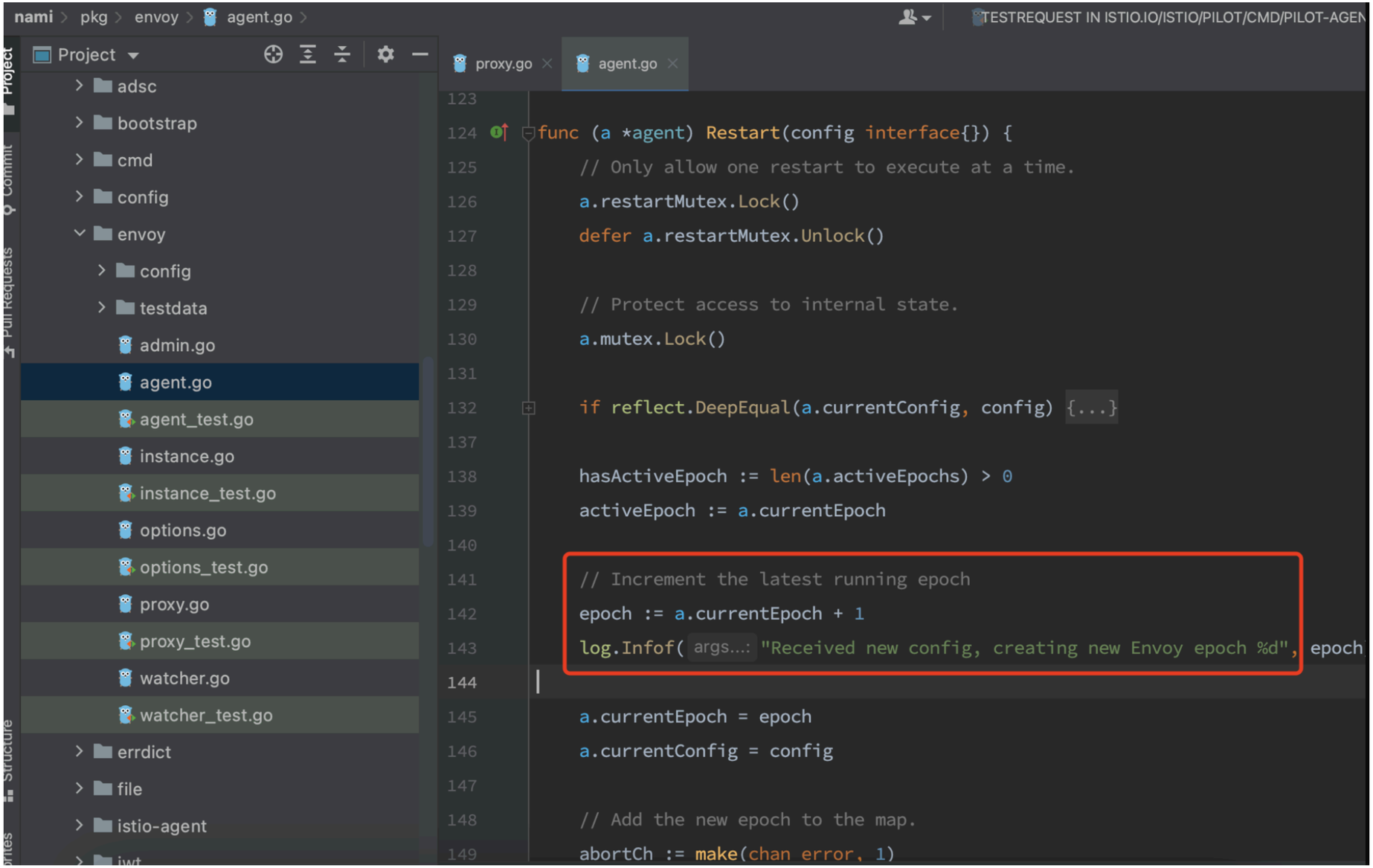
优化代码：

```go
func convertWorkloadInstanceToServiceInstance(workloadInstance *model.WorkloadInstance, serviceEntryServices
    serviceEntry *networking.ServiceEntry) []*model.ServiceInstance {
    out := make([]*model.ServiceInstance, 0)
    for _, service := range serviceEntryServices {
        for _, serviceEntryPort := range serviceEntry.Ports {
            ep := *workloadInstance.Endpoint
            ep.ServicePortName = serviceEntryPort.Name

            // 当打开 DiscoveryRemoteClusterWorkloadEntry Feature
            // 则按照主集群处理主集群 WorkloadEntry TargetPort 逻辑处理远端集群
            if features.DiscoveryRemoteClusterWorkloadEntry && len(workloadInstance.PortMap) > 0 {
                instancePort := workloadInstance.PortMap[serviceEntryPort.Name]
                if instancePort == 0 {
                    instancePort = serviceEntryPort.Number
                }
                ep.EndpointPort = instancePort
            } else if serviceEntryPort.TargetPort > 0 {...} else {...}

            ep.EnvoyEndpoint = nil
            out = append(out, &model.ServiceInstance{...})
        }
    }
    return out
```

关于落地服务网格的一些坑 — Restart API BUG



下载小红书

加入小红书