eBay边缘节点的云原生网络实践

1. 前言

边缘节点又被称为POP（point-of-presence）节点，通常边缘节点设备更接近用户终端设备其主要作用是加速终端用户访问数据中心网络资源。从节点构成上来说，通常POP是由缓存设备和本地负载均衡设备组成。
这里是基于POP节点的网络访问路径：

用户->Internet->边缘路由器->专线->POP节点->专线（骨干网）->数据中心

即用户请求通过互联网连接eBay的边缘路由器，边缘路由器通过专线连接POP节点，再通过专线接入eBay的骨干网络，最后进入数据中心。这里的边缘节点分布在全球各地，通常是租用互联网服务提供商(ISP)的机房的若干节点和专线，然后在节点部署软件实现的负载均衡，最终将这些节点变成用户访问eBay的接入点。
另外边缘节点也提供缓存（caching)功能，将用户访问的内容保存到缓存服务器，当用户请求命中缓存后可以直接返回，这样也能减少网络延迟以及改善用户体验。

1.1 云原生(Cloud-Native)网络

云原生网络是指通过软件的方式实现网络功能，而这些软件是运行在Linux容器，比如典型的就是Kubernetes集群。在云原生网络实现之前，这些功能都是运行在专门的物理设备，比如硬件负载均衡设备。
通过云原生网络，我们期望边缘节点能够具备以下特征：

1. 全球分布(Globally distributed)
边缘节点的地理分布在全球各地，更加接近我们的客户，也就意味着更低的网络延迟和更好的用户体验。
2.可扩展(Extensible)
从运维的角度看易扩展，便于流量迁移和管理，可以更加灵活的管理边缘节点内部的流量。
3.可伸缩(Elastic)
根据流量模式按需扩展以有效处理负载。通常硬件设备的带宽是固定的，但是基于云原生网络，我们能够实现总带宽的弹性伸缩。
4.高安全性(Secure)
抵御已知的网络安全威胁，比如DDOS攻击，同时也能够提供高安全性的端到端加密网络。

1.2 UFES

UFES(Unified Front End Services)是一个eBay进行了很多年的项目，2017年启动，旨在构建一个低延迟，高稳定性的基于云原生网络(cloud-native)的网络前端和边缘计算平台。也正是通过这个项目我们搭建了大量的软件实现的边缘节点，实现了基于软件的4层和L7负载均衡，其中4层的实现是基于IPVS，L7的实现基于Envoy。

UFES项目主要目标是eBay的公网流量，eBay公网流量主要来自下几个领域:

- 桌面端(Desktop Web): 是指通过桌面浏览器访问的流量，也就是我们常见的 www.ebay.com 主站，也称之为SEO/dweb域。
- 手机浏览器端(Mobile Web): 是指通过手机浏览器访问的流量，也就是通过手机浏览器访问m.ebay.com，也称之为mweb域。
- 原生应用端(Native APP): 是指通过安卓或者IOS的eBay应用访问eBay的流量，这里主要都是api接口的调用，主要访问的地址是api.ebay.com

2. 传统基于硬件负载均衡的架构

在启动UFES项目之前，eBay的公网流量都是经过边缘路由器(Edge router)接入边缘节点，边缘节点是由Netscaler的硬件负载均衡设备对组成，工作在Active-Standby模式。我们给不同的域名配置了一组专有的硬件负载均衡设备，比如在桌面端主站配置了十七组设备分布在全球，也就意味着全球用户都是通过这十七个边缘节点访问eBay主站。这些硬件负载均衡上面配置了大量的L7转发规则，比如在桌面端主站上面配置了以下规则：

- www.ebay.com/itm/ ->将请求转发到详情页
- www.ebay.com/sch ->将请求转发到搜索服务

所以在www.ebay.com后端其实有数百个应用在为它提供服务，这些应用虽然并没有暴露公网地址直接接收用户请求，但是通过边缘节点设备的转发规则，最终为全球用户提供服务。

图2-1是基于硬件负载均衡的网络架构，其中GTM是一个智能的DNS，能够对后端VIP进行健康检查，基于VIP的状态决定是否返回给用户，Akamai是CDN服务提供商。

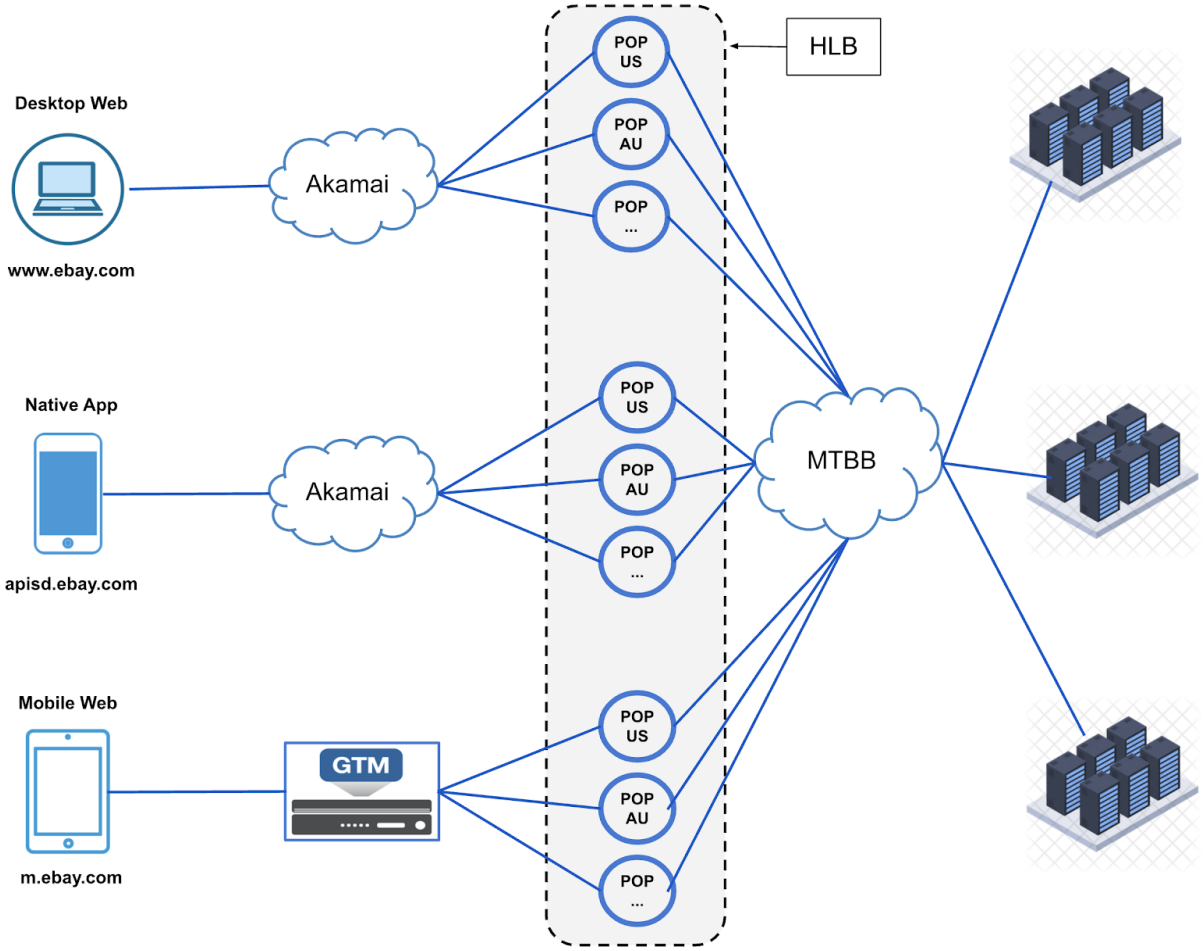

图2-1 基于硬件负载均衡的网络架构

2.1 边缘节点的特征

边缘节点的主要作用是将外网的请求接入数据中心，并且根据不同的L7转发规则将用户请求转发到不同的应用集群进行相应的处理。所以边缘节点上会配置公网的VIP地址，它主要有以下几个特征：

1. 外网流量经硬件负载均衡设备进入数据中心。
2. 接入点VIP数量少，比如在桌面应用端，我们只配置了3个VIP。
3. L7规则多，包括响应规则（比如重定向）以及转发规则（比如转发到后端不同应用），3个大的公网域总共配置了将近4000条L7规则。
4. 流量大，比如手机端应用日访问量达45亿次

2.2 硬件负载均衡遇到的挑战

由于eBay传统的边缘节点是基于硬件设备，而这种专门的物理设备除了价格很贵以外，在实际应用中还遇到以下一些挑战：

1. 硬件负载均衡器具有较高的运行和配置成本，不容易扩展，比如我们经常由于购物季流量过高需频繁迁移VIP，硬件负载均衡设备不支持弹性扩容。

2. 由于工作在active-standby模式，无法提供高可用性，其对故障域处理是的1 + 1而非N + 1，也就是在一个边缘节点不能接受1台以上宕机。
3. 依赖第三方供应商来提供核心前端流量管理功能，配置不够灵活。
4. eBay的应用基于微服务架构，这种微服务的体系结构导致负载均衡器数量的增加，eBay总用有超过3,000个的公网VIP。
5. 边缘节点全球分布数量有限，eBay用户在美国以外的地区也有很高的流量，但边缘节点少，而硬件负载均衡PoP配置需要消耗较多的人工，时间和成本。

3. 云原生网络的边缘节点架构

正是因为硬件负载均衡设备存在着种种问题，而且公司的方向是将所有的应用都迁移到Tess (eBay的kubernetes)，云原生网络也是其中的一部分，其中最重要的一个目标便是将边缘节点的硬件负载均衡设备替换成能够支持k8s原生部署的软件实现。

图3-1是云原生网络的边缘节点架构，其中边缘节点被分拆成两部分：前端代理（Front Proxy)及数据中心网关（DC gateway).

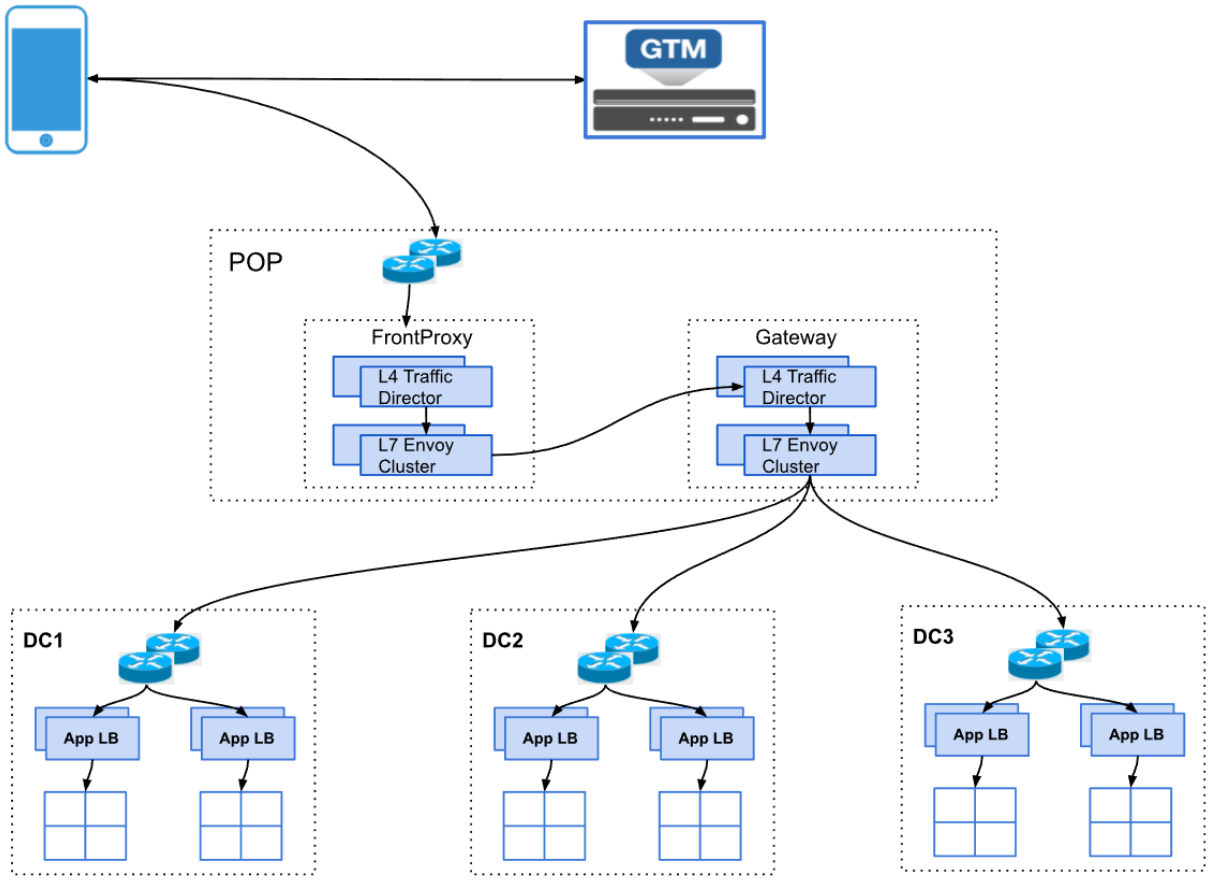

图3-1 云原生网络的边缘节点架构

3.1 前端代理

前端代理(Front Proxy) 是一层软件负载均衡，它连接客户和eBay，是所有进入eBay流量的前端代理。这一层部署在所有的边缘节点和数据中心，主要代理公网的流量，包括桌面端，手机浏览器端以及原生手机应用端。前端代理的主要功能如下：

1. 能够在离用户最近的地方做SSL卸载，减少TLS连接建立的时间。
2. 高度优化前端代理的TCP连接设置。
3. 为公网 VIP 启用任播。
4. 配置响应规则，比如响应301、302等。
5. 顶层轻量化，大大减少前端代理的上游集群和L7规则。

前端代理还有一个很重要的功能是容灾、故障恢复和流量管理，这一层的流量会跨数据中心，也意味着即使后端应用的某个数据中心出现故障，只需要将该数据中心的VIP停止，从而不会对用户造成数据面的影响，包括DNS缓存（TTL和客户端缓存）带来的影响。

3.2 数据中心网关

这一层在UFES项目早期并没有设计，它的主要作用是作为数据中心的接入层，将前端代理的流量转发到后端的应用集群。我们会在每一个可用区配置一个网关，所有的转发规则都配置在数据中心网关。这一层的流量也是跨数据中心的，权重比例是本数据中新承担99%流量，远端数据中心承担1%流量。

数据中心网关的主要作用包括以下几点:

1. 支持云原生的L7规则管理系统，核心功能不再依赖硬件设备提供商Netscaler。
2. 为公网和内部的4层和L7的流量提供更多的可见性。
3. 基于Envoy能够提供如授权/身份验证、速率限制等功能。
4. 替换服务公网流量的Web层的硬件负载均衡设备，有了数据中心网关后数据流路径如下：
   App Tier(HLB) -> Web Tier (DC gateway) -> PoP -> GTM

4. 基于IPVS和Contour的边缘网关

传统的硬件负载均衡设备能够同时管理L4和L7的流量，典型的例子是我们可以创建一个TCP协议的VIP, 用来管理只有4层流量的场景。另外也可以创建HTTP协议的VIP，可以进一步管理L7规则，支持TLS等等。然而在软件实现上，L4和L7需要分别采用不同的技术实现，这里我们用到的是IPVS和Contour/Envoy.

4.1 基于IPVS的L4负载均衡

L4这一层主要负责TCP包的转发，表4.1是我们在选择L3/L4软件负载的主要考虑因素，当时可选的主要有IPVS, Netfitler, OVS以及EBPF, 从技术成熟度以及社区接受度来看，都促使我们选择了IPVS。

表4.1 L3/L4软件负载的主要考虑因素

	IPVS	Netfilter	OVS	EBPF

社区采用（实战测试）	高	-	-	-
支持ECMP（任播）	高	-	-	-
支持一致性哈希	中	-	-	-
支持DSR和隧道模式	高	-	-	-
弹性和可操作性	高	-	-	-
规模和集群支持	高	-	-	-
状态管理和与 k8s 的集成	中	-	-	-
性能（延迟速度）	高	-	-	-
支持负载均衡	高	-	-	-
可扩展性和易于贡献开源	高	-	-	-

图4-1是L4 组件控制面，这里的L4 Director是fork了google的开源项目seesaw并进行了定制，L4 director会在用户空间生成IPVS service的哈希表，并基于Linux 内核通信的 Netlink 套接字发送到 kernel，更多细节可以参考作者之前一篇博客 [eBay的4层软件负载均衡实现](#)。

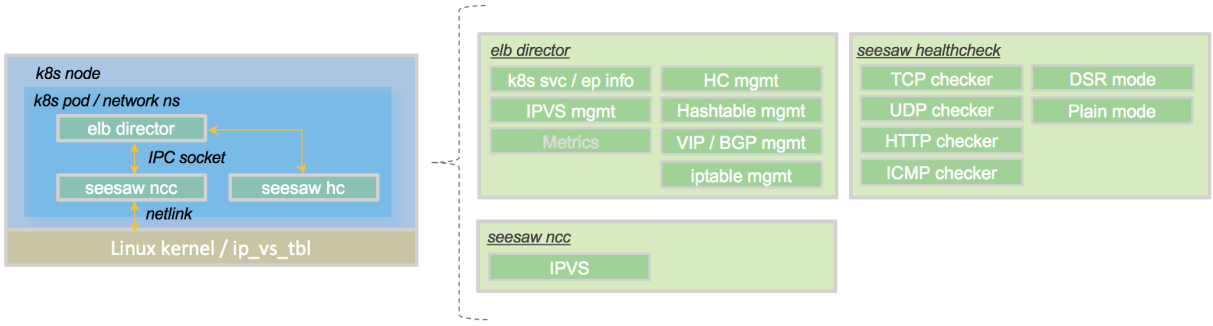

图4-1 L4 组件控制面板

图4-2是L4组件数据包封装的细节，这里采用的是IPVS IPIP 隧道模式，L4 director会将IPIP数据包发送到后端真实的服务器（也就是Envoy)解包再进一步处理，所以需要将VIP的地址绑定到后端服务器的tun0接口，后端Envoy Pod会从上游服务获取到响应之后直接返回（Direct Server Return）到客户端。

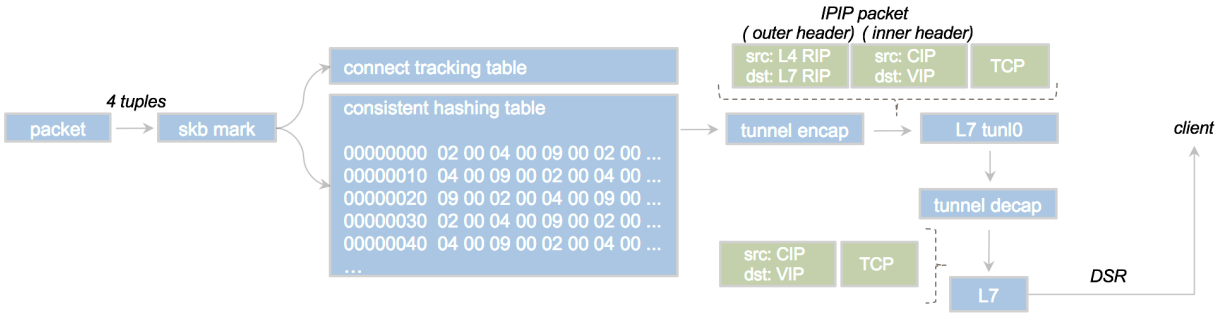

图4-2 L4数据包封装

4.2 基于Envoy/Contour的L7负载均衡

L7数据面-Envoy

L7主要负责L7规则的转发，TLS卸载，限速，以及安全相关的授权和认证等等，在选择L7的数据面的时候，主要的考虑因素如表4.2：

表4.2 L7软件负载的主要考虑因素

	Nginx	Envoy	HAProxy	ATS	ProxyGen	Fabio
社区采用(实战测试)	高	低	高	中	低	中
弹性和可操作性	高	中	高	高	高	中
规模和集群支持	高	中	高	高	高	中
状态管理和与 k8s 的集成	高	高	中	中	低	低
性能和延迟	-	中	-	-	高	-
对象缓存功能 (CDN)	中	低	中	高	低	低
支持负载均衡算法(连接、延迟、队列、随机	高	中	高	高	低	高
L7支持(重定向、过滤、修改、会话)	高	中	高	中	低	中
安全功能和 TLS 支持(速率限制、机器人检测)	中	低	中	中	低	中
支持一致性哈希	中	高	中	中	中	中
可扩展性和易于贡献开源	高	高	中	中	低	高

其中基于Envoy的微服务架构，服务网格是网络云原生的一个发展方向，而且本身已经在eBay生产环境上应用过，这些也促使我们选择Envoy作为数据面。

L7控制面-Contour

在早期的时候我们用的是自定义的IngresController和IngressRoute，因为当时社区的方案也不是很成熟，但是在前两年我们把IngressController换成了社区的Contour，一个主要的考虑因素是因为Contour本身代码相对Istio要简单很多，定制起来也比较容易，没有必要再去维护一个inhouse的Ingress Controller。

图4-3是基于Contour的架构图：

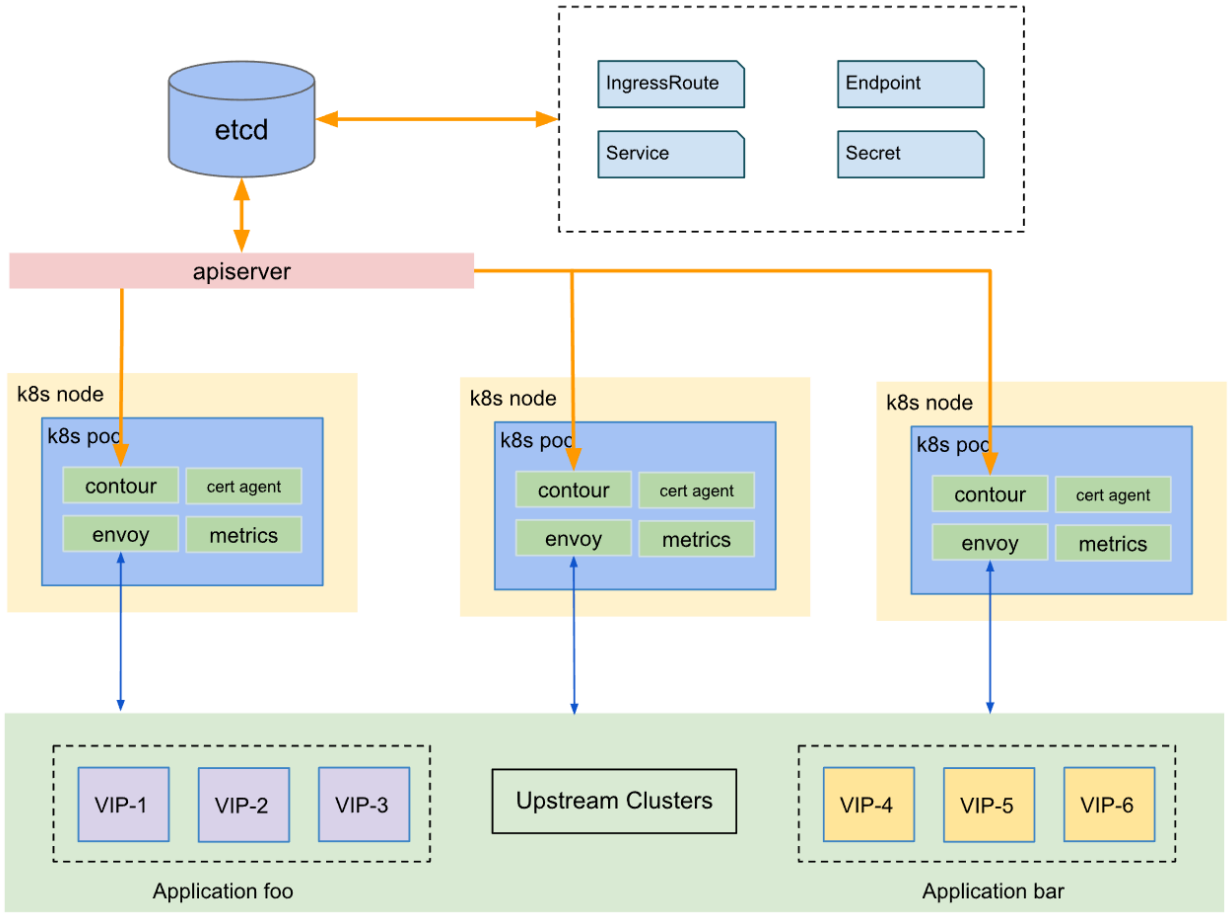

图4-3 L4数据包封装

这里Contour作为边缘网关L7控制面，主要功能包括以下几点：

1. 负责TLS终结
2. 实现Envoy xDS，将IngressRoute配置推送到Envoy
3. 集成eBay CA，证书生成自动化

有了L4和L7的软件实现之后，完整的边缘节点架构图如图4-4：

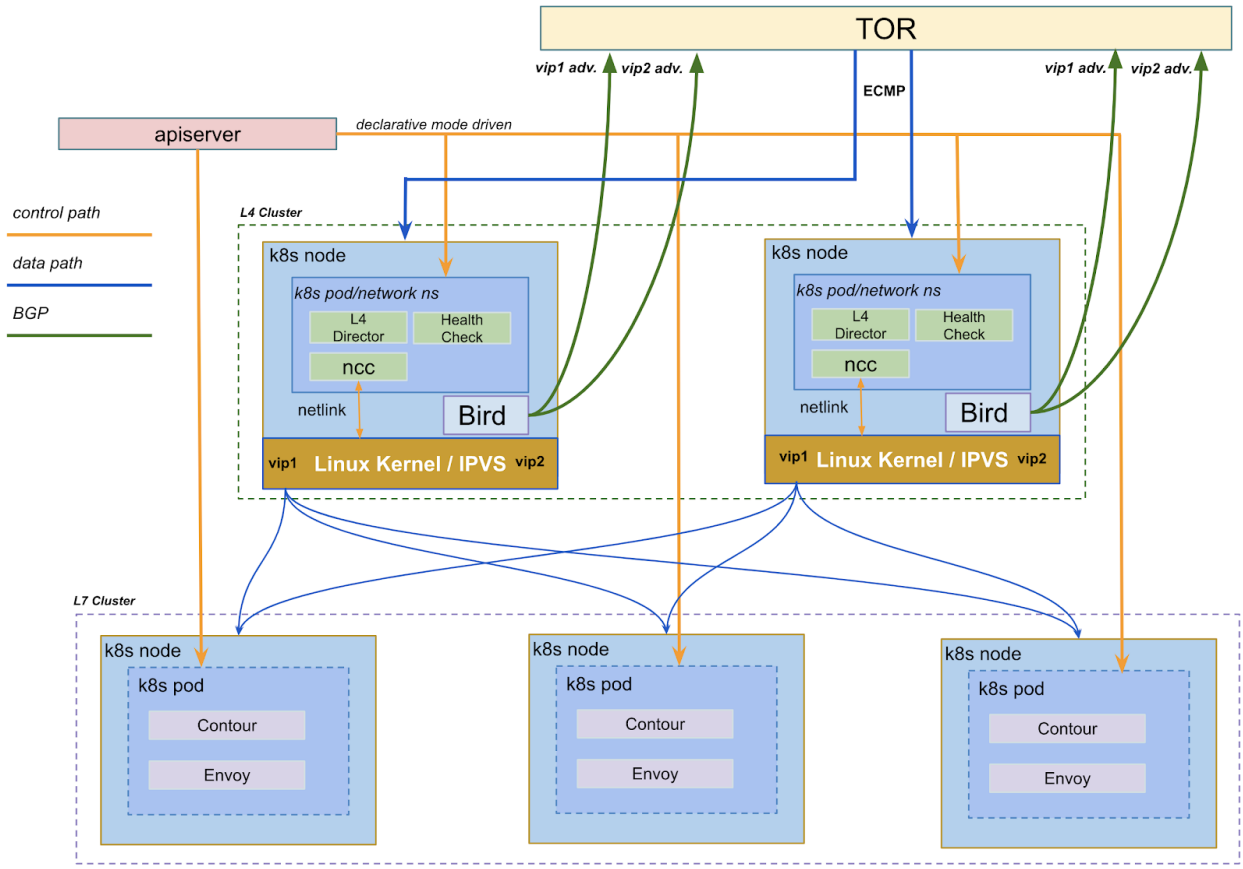

图4-4 边缘节点架构图

其中基于IPVS的L4负载均衡模块主要有以下几个特点：

1. 自定义的一致性哈希kernel模块作为IPVS调度器，保证同一个五元组会被转发到同一个后端服务器
2. 基于k8s的控制面，支持k8s原生部署。
3. 可水平扩展，可以通过扩容的方式增加集群数量，提高整体带宽。
4. 支持直接服务器返回(Direct Server Return)，用户请求会经过IPVS Pod到达后端服务器，服务端直接返回。
5. BGP宣告VIP网段到路由器，通过ECMP支持多活。

而基于Contour和Envoy的L7负载均衡模块主要有以下几个特点：

1. Contour管理边缘节点节点L7规则，负责将IngressRoute转换成Envoy配置。
2. 集成eBay CA，实现证书自动化。
3. L7规则从硬件负载均衡全量迁移到Envoy。
4. 不同domain配置专有L7集群，实现数据面和控制面的隔离。

虽然边缘节点被拆分成了前端代理和数据中心网关，但是它们控制面的架构其实是完全一样的，都是通过L4集群和L7集群组成。主要区别在于：

1. 前端代理分配公网IP作为服务的VIP，数据中心网关分配私有IP作为服务VIP。
2. L7规则配置不同，其中响应规则配置在前端代理，转发规则配置在数据中心网关。

图4-5是基于前端代理和数据中心网关的边缘节点部署架构，目前我们已经在全球各地部署了超过20个POP：

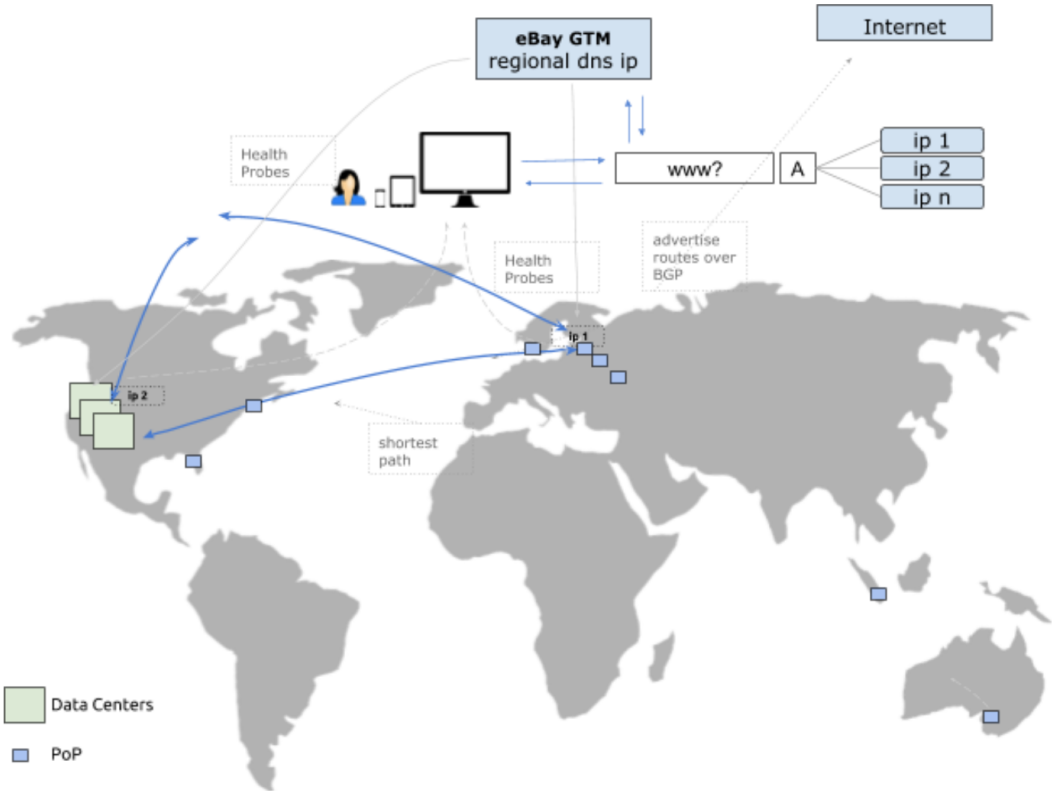

图4-5 边缘节点架构图

5. UFES动态内容缓存

缓存(caching)是边缘节点提供的另一个重要的功能，通过在边缘节点提供动态负载缓存功能来改善最终用户体验，它可以减少由底层网络和后端系统引起的延迟。通过缓存，主要能实现以下几个目标：

1. 提高访问速度，在 POP缓存动态页面的静态部分，测试显示减少延迟500-700 毫秒。
2. 适用于非个性化页面
3. 可能的候选缓存页面/功能：
    a. 今日易趣:未登录主页的用户
    b. 浏览页面
    c. 交易页面

图5-1是UFES边缘节点缓存实现的架构图:

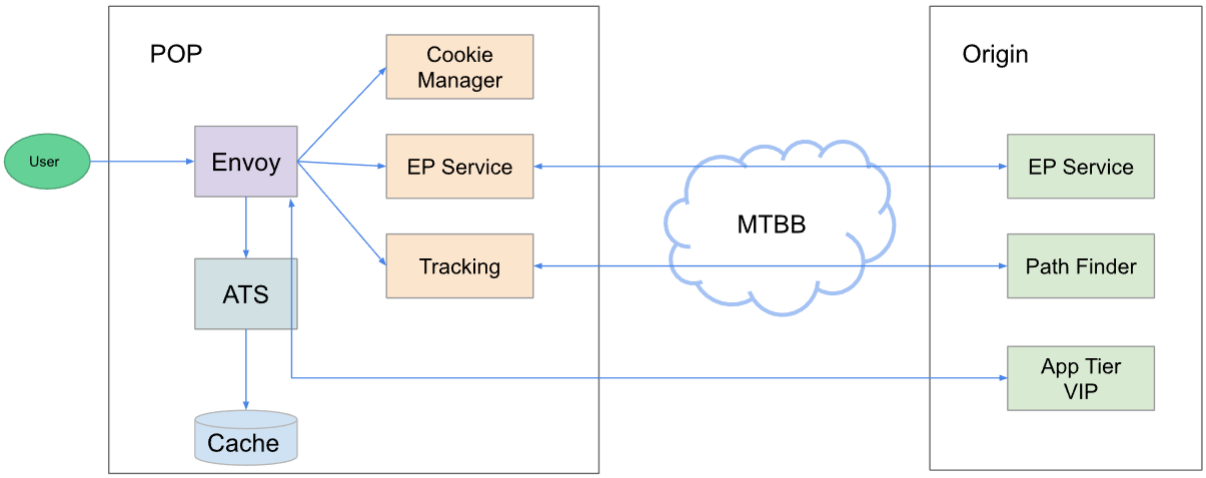

图5-1 UFES缓存架构图

其中MTBB(Multi-Tenant Backbone)是eBay的骨干网络。Tracking是一个sidecar，用来管理用户请求的多个调用的状态。EP(Experimentation Platform)服务用来给用户请求生成 一个ID，后续会基于这个ID判断页面是否缓存在ATS。

ATS(Apache Traffic Server)提供缓存功能。 ATS 管理完整的缓存对象生命周期，包括 Hit、Miss、Age、Refresh 和缓存对象操作，为了实现缓存的功能，需要定制Envoy过滤器支持以下功能：

1. 识别支持缓存的应用和URL，比如url-path 以“/b”开头被标记为支持缓存。
2. 调用 Orchestrator，根据请求的headers, url 生成 ATSKey。
3. 调用ATS，输入ATSkey，响应是是否命中缓存。
4. 修改响应，包括解析修改HTML内容，修改header和cookie。

6. PaaS自动化集成

前面介绍了很多UFES相关的背景，UFES是更接近于底层的组件，接下来我们要解决的问题是：

1. 如何将硬件负载均衡设备上的L7规则全量迁移到软件负载均衡？
2. 在迁移阶段如何同时对软件和硬件负载均衡上的L7规则进行管理？

这些主要就是PaaS这一层需要解决的问题，这里需要提供各种自动化的API和工作流来对软件负载均衡进行初始化和管理。举一个不太恰当的例子，UFES就像是提供的砖和瓦，而PaaS这一层就需要根据一栋老房子，造出一栋几乎一模一样的新房子。
这一层也像胶水层一样，需要把各个组件集成到一起，重新打包成一个新的面向用户的功能，并且各种组件的问题都会在这里放大，因为即使每个依赖的组件提供的稳定性是99%，如果有十个依赖，那0.99的10次方是大概是0.90，也就只能90%的稳定性。这是PaaS层遇到的问题，同时也是微服务带来的一个挑战。

6.1 Zebra

Zebra是eBay一套PaaS系统，主要作用是为应用创建和管理虚拟机/Pod集群，负载均衡以及DNS。这套系定义了一套工作流程，并且统利用spring quartz进行任务的调度。L7规则的各种自动化也是基于这套系统。

IngressTemplate

自动化的第一步是需要定义一种资源用来描述L7规则，这里我们定义了一套IngressTemplate，有点类似k8s里面的Ingress，但是这套IngressTemplate是定义在内部一个基于MongoDB的叫CMS的配置管理系统。

这里我们实现了一套基于antlr4分析语法器，能够将Netscaler DSL的L7规则转换成我们的IngressTemplate，举个例子它能将下面的Netscaler DSL的L7规则：

```
HTTP.REQ.HOSTNAME.SERVER.SET_TEXT_MODE(IGNORECASE).EQ(\"www.ebay.ca\") &&
(HTTP.REQ.URL.PATH.GET(1).SET_TEXT_MODE(IGNORECASE).EQ(\"itm\") ||
HTTP.REQ.URL.PATH.GET(2).SET_TEXT_MODE(IGNORECASE).EQ(\"like\"))
```

转换成Json格式的IngressTemplate:

```
{
  "rule": {
    "path": [{
      "get": "1",
      "equal": "\"itm\"",
      "ignorecase": "true"
    },
      {
        "get": "2",
        "equal": "\"like\"",
        "ignorecase": "true"
      }
    ],
    "server": {
      "equal": "\"www.ebay.ca\"",
      "ignorecase": "true"
    }
  }
}
```

这么做的主要目的是为了能够将L7规则与某一种特定的DSL解耦，能够将这套IngressTemplate很容易的转换到其它的L7实现。另外Netscaler的L7规则实际上是一个逻辑表达式，通过这种解析能够支持更灵活，比如A || B与B || A转换出来的IngressTemplate是一样的。

自动化API

我们在Zebra上提供了大量的API用来管理L7规则，主要包括:

1. 支持对硬件和软件负载均衡的L7规则创建和删除。
2. 支持初始化一个软件负载均衡，相当于把硬件上的L7规则全部拷贝到软件负载均衡。
3. 支持证书自动化。
4. 检测硬件和软件负载均衡设备的配置差异。

图6-1 是各个组件之间调用的架构图，其中WISB是what it should be，有点类似于k8s里面的资源定义，而WIRI是what it really is，是指实际的硬件配置。

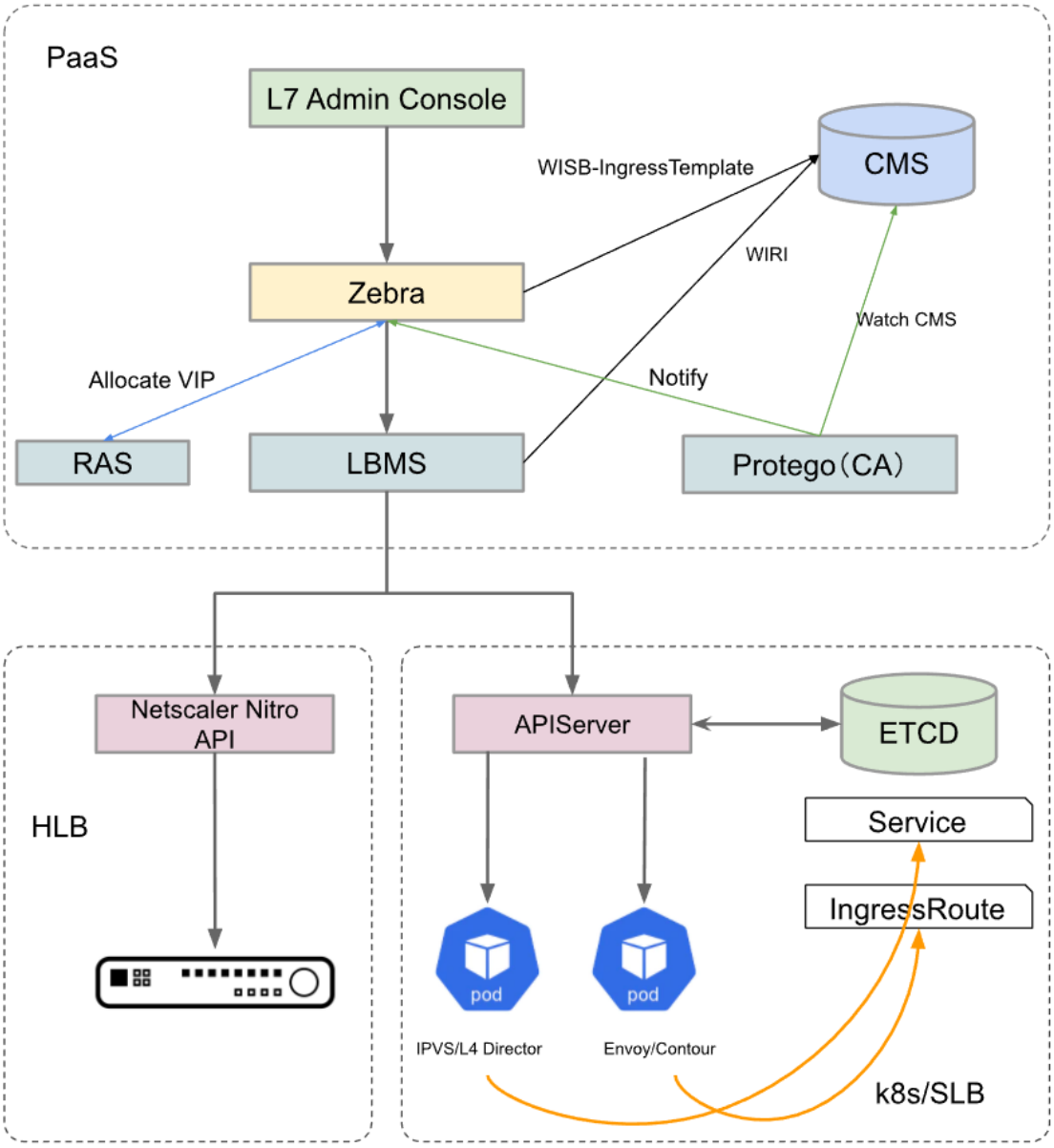

图6-1 PaaS/UFES集成架构图

流量验证

软件负载均衡设备通过PaaS自动化配置完成后，还需要做的一件事情是运行流量验证（traffic validator），通过自动化的CI运行。

它的工作原理是通过抓取生产环境下真实的用户访问记录并提取出URL访问的模式并且记录下相应的返回码，然后将同样的URL集合在新创建的软件负载均衡上运行一遍，如果返回码完全匹配则认为通过验证，否则可能是有某些L7规则出现不匹配，需要查找原因并重新配置。

6.2 L7统一管理终端

在迁移阶段，软件和硬件负载均衡会同时服务生产环境流量，在这一段时间如果有L7规则的变更，需要同时配置在两边。同时为了方便管理员能够更加方便的管理，我们开发了一个L7管理终端，支持管理员同时操作软件和硬件负载均衡，图7-1是管理终端的一个界面：

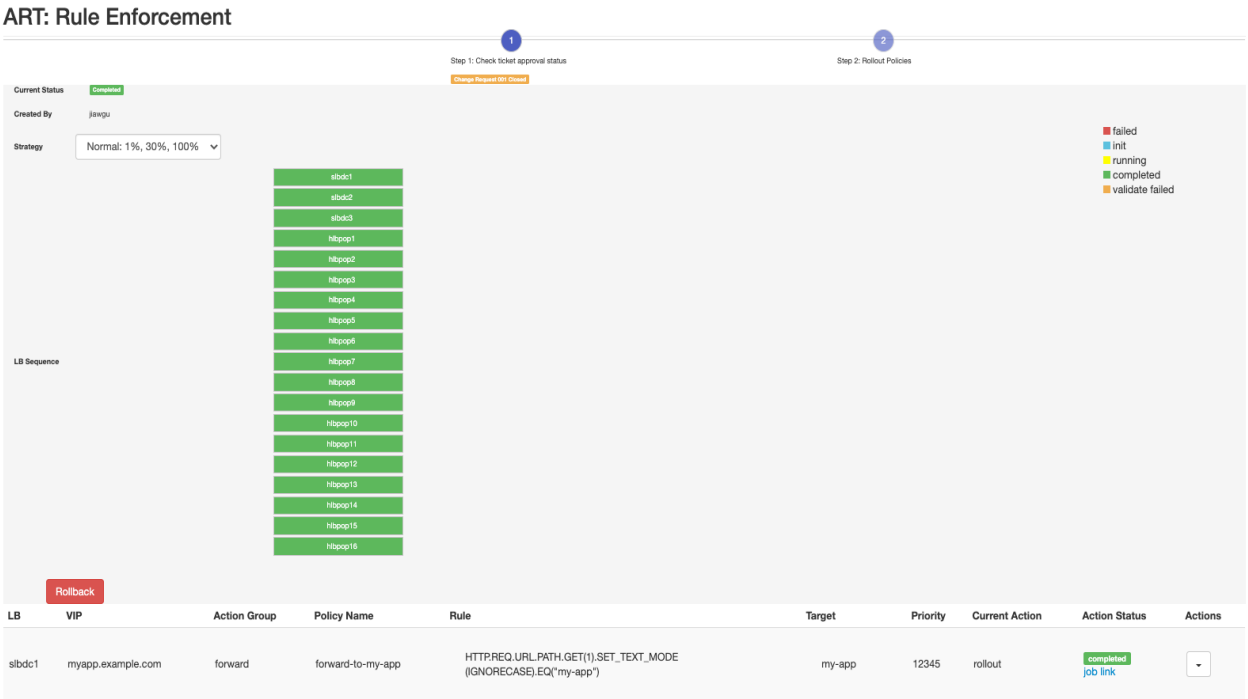

图7-1 L7统一管理终端

在L7管理终端上提供了多种功能，主要包括以下几点：

1. 统一管理软硬件负载均衡。这里通过LBMS对软件硬件负载均衡设备提供的统一的API适配，对管理终端来说，对软件和硬件负载均衡操作是完全兼容的。
2. 定义多种发布策略。在发布新的L7规则的时候，可以支持单个设备发布，也可以支持分组多设备发布。
3. 定义发布和回滚工作流程。

6.3 PaaS集成的反思

现在再回过头来看整个PaaS的集成，其实一开始并不符合cloud native的原则。这本身还是一个命令式的集中式系统，虽然我们定义了WISB，但还是一个命令式（Imperative Programming）而非声明式（Declarative Programming）的编程。这里将IngressTemplate保存在私有的MongoDB(CMS)，整个Zebra系统某种意义上编程了一套k8s里面IngressRoute的 spec builder，因为Zebra最终的输出其实是通过LBMS创建了很多k8s的service以及IngressRoute 。这样导致的结果是，我们需要为了适配不同的场景而往Zebra堆很多功能，这些功能虽然也带有流量管理，服务发现机制，但是最终它只是某种意义上的一个迁移的工具，很多功能当某一天迁移结束后也就失去作用了。

另外一方面，这里又涉及到业务驱动和技术驱动的差别，前者见效快但是长期的投入会比较多，后者见效慢，而且投入会比前者更多。但是它会去构建一些核心的和高频使用的组件，正如Jian总说的，越是被高频率使用的组件我们越会有更多的机会去迭代和增强，而且生命力也是最长的，这才是最值得长期投入的。但这里也是一个业务和技术的妥协，早期在技术并不很成熟的情况下先保证业务，后续再逐渐将控制面也迁移到k8s。其实在eBay的应用上k8s的时候也遇到了同样的问题，一开始我们的平台部门还没有适配原生部署，所以我们采取的方式是assimilation，也就是启动一个fat container，这个container和VM行为几乎完全兼容，Pod启动之后并没有部署代码，而是通过fat container里面的agent再去部署应用，目前我们正在面临的问题是如何将这一套assimilation的方式再迁移到原生部署。

所以如果能再有机会重新开始一次，我觉得有几点是我们可以重新考虑的：

1. 在Tess构建Federation的IngressRoute，如此一来我们只需要维护一份联邦的IngressRoute，再新增PoP的时候自动同步到新的k8s集群，相应的控制器会去配置IPVS和Contour，而不需要每次需要在创建新的POP的时候再去通过Zebra的自动化重新生成IngressRoute spec。
2. 构建k8s的控制器来自动发现并且生成IngressRoute，一方面它能初始化L7规则的spec，另外一个方面我们也可以利用这个控制器检测实际配置和spec不一致的地方，可以长期发挥作用。

7. Contour/Envoy的定制和扩展

为了满足eBay实际的业务需求，也就意味着原本硬件负载均衡设备上所有我们已经使用到的功能，在软件负载均衡上也需要有相应的实现。但是社区没有我们这么复杂的使用场景，这样也就意味着我们需要对L7的控制面Contour和数据面Envoy都需要做相应的定制，这里也整理了一下UFES在这方面的工作。

7.1 扩展Contour

IngressRoute是Contour引入的一个定制资源定义，因为k8s支持的Ingress资源能够支持的用户场景有限，为了适配eBay的用户场景，我们给IngressRoute做了大量的定制。

1. 定制apiVersion为contour.ufes.io/v1beta1，通常每一个VIP会有两个IngresRoute，一个配置了80（HTTP)的L7规则，另外一个是443(HTTPS)的：这里是一个IngressRoute的示例:

```yaml
apiVersion: contour.ufes.io/v1beta1
kind: IngressRoute
metadata:
  generation: 5
  name: www-ebay-com
  namespace: seons
spec:
  filters:
    - filterConfig:
        reset: true
        upstream_cluster: reset
      filterName: envoy.accesscontrol
    - filterConfig:
        drop: true
        upstream_cluster: drop
      filterName: envoy.accesscontrol
  http_connection_manager:
    idle_timeout: 600s
    stat_prefix: ingress_http
    use_remote_address: true
  listener:
    bind: 10.0.0.1
    port:
      number: 80
  route_configuration:
    default_cluster: homepage-web-1-80
    default_cluster_port: "80"
    drop_on_upstream_timeout: true
    reset_on_upstream_disconnect: true
  routes:
    - additionalProperties:
        NSName: homepage-to-globaldeals
        NSPriority: "10000"
      match:
        case_sensitive: false
        uri:
          prefix: /globaldeals
      name: homepage-to-globaldeals
      permitInsecure: false
      services:
        - name: globaldeals-web-1-80
          port: 80
```

从这个扩展资源中我们可以看到，我们给IngressRoute增加了很多属性，比如additionalProperties，这个是用来与硬件设备上的L7规则名字以及优先级的，另外还定制了安全相关的filter，用来重启或者丢弃用户的请求。在硬件负载均衡设备上，每条L7规则都会有一个名字以及执行的优先级，而IngressRoute上Route配置的顺序就决定了执行顺序，所以需要在生成Route时候根据优先级做好排序。

2. 引入HTTPMatchRequest
HTTPMatchRequest其实是Istio VirtualService里面的功能，Contour社区只支持简单的Prefix匹配，通过引入HTTPMatchRequest能够支持更加复杂的规则，比如以下的一些常见的规则：

```
{"uri":{"prefix":"/"},"headers":{":method":{"exact":"POST"}}}
{"uri":{"exact":"/backend"},"headers":{":authority":{"regex":"www.example.*"},":method":{"exact":"POST"}}}
{"uri":{"regex_path_and_query":"www.example.com?test=true"},"case_sensitive":true}
```

这里我们也可以看到IngressRoute同时还能支持IngnoreCase，社区目前还不支持。

3. IngressRouteSpec增加Listener
eBay的每一个公网域通常有多个VIP，比如桌面浏览器端分配了3个VIP，需要支持Envoy监听在不同的VIP，这里给IngressRouteSpec增加了Listener:

```
// IngressRouteSpec defines the spec of the CRD
type IngressRouteSpec struct {
    // Listener provides optional Listener config for Envoy
    // In the case of Delegation, only Listener from the root IR will be recognized
    Listener interface{} `json:"listener,omitempty"`
    // Virtualhost appears at most once. If it is present, the object is considered
    // to be a "root".
    VirtualHost *VirtualHost `json:"virtualhost,omitempty"`
    // Routes are the ingress routes. If TCPProxy is present, Routes is ignored.
    Routes []Route `json:"routes"`
    // TCPProxy holds TCP proxy information.
    TCPProxy *TCPProxy `json:"tcpproxy,omitempty"`
}
```

这样做的目的是为了让Envoy能够监听在独立的VIP，社区默认Envoy会监听在0.0.0.0，这样也就导致我们必须要使用SNI，并且无法对单个的VIP进行管理，这一点对于独立的流量管理非常重要。比如假如我们使用SNI，也就是多个VIP的L7规则合并到一个listener 0.0.0.0:443，如果其中某一个VIP出现了问题那我们将无法把这个VIP停止。

下面是使用了独立的listener之后我们可以看到Envoy监听在不同的VIP：

```
/ # netstat -ln
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 127.0.0.1:8001          0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:8002            0.0.0.0:*               LISTEN
tcp        0      0 10.0.0.1:80             0.0.0.0:*               LISTEN
tcp        0      0 10.0.0.1:443            0.0.0.0:*               LISTEN
```

4.支持Weighted Endpoint以及停止Endpoint接收流量

为了实现高可用，eBay的生产环境流量大部分都是跨三个数据中心，而且需要支持本数据中心99%流量，远端数据中心1%的流量。假设每个数据中心有一个Gateway VIP，需要支持不同的VIP权重不一样，并且能够支持停止某一个数据中心的流量。

UFES的做法是，给前端网关创建一个k8s Service，它的ExternalIP就是公网的IP ,而这个service的Endpoint是由PaaS层维护的，里面通常是三个数据中心网关的私有IP，并且在Endpoin标记如下annotation:

```
{
  "weights": [{
    "endpoint_address": "10.0.0.1",
    "endpoint_port": "80",
    "endpoint_name": "dweb-lvsaz-10.0.0.1",
    "endpoint_enabled": "true",
    "endpoint_weight": 1
  }, {
    "endpoint_address": "10.0.0.2",
    "endpoint_port": "80",
    "endpoint_name": "dweb-rnoa-10.0.0.2",
    "endpoint_enabled": "true",
    "endpoint_weight": 99
  }, {
    "endpoint_address": "10.0.0.3",
    "endpoint_port": "80",
    "endpoint_name": "dweb-slcaz-10.0.0.3",
    "endpoint_enabled": "true",
    "endpoint_weight": 1
  }
```

然后修改Contour的EDS，能够适配Endpoint的annotation并将值推送到Envoy:

```
endpointsMap := make(map[string]EndpointValues, len(spec.WeightedEndpoints))
    for _, weight := range spec.WeightedEndpoints {
        var epval EndpointValues
        epval.Weight = weight.Weight
        epval.Enabled = true
        if weight.Enabled == "false" {
            epval.Enabled = false
        }
        endpointsMap[weight.EndPointAddress+":"+weight.EndPointPort] = epval
    }
```

最终更新的是Envoy Clusters的upstream hosts状态，如表7.1：

表7.1 Envoy cluster upstream hosts属性

Name	Type	Description
healthy	String	The health status of the host. See below
weight	Integer	Load balancing weight (1-100)

5. 集成eBay的CA
社区的实现是将证书和key都保存在k8s的secret里面，但是这种方案到不到我们对于证书保密的要求: 证书和key不能落盘，哪怕是通过加密的方式保存，而实际上secret其实也是一种明文的方式保存，base64解码就能得到了。

这里我们的解决方案是将证书的一个引用地址保存到secret里面，然后会有一个cert agent负责 从不同的两个组件分别拿到证书和key，Contour通过GRPC从cert agent获取证书和key，再通过SDS推送给Envoy。下面是对IngressRoute的定制，TLS能够同时支持secretName和IdOrURL。

```
// TLS describes tls properties. The CNI names that will be matched on
// are described in fqdn, the tls.secretName secret must contain a
// matching certificate unless tls.passthrough is set to true.
type TLS struct {
    // required, the name of a secret in the current namespace
    SecretName string `json:"secretName,omitempty"`
    // Esams 2.0 Certificate ID / URL
    IdOrURL string `json:"idOrURL,omitempty"`
    // Minimum TLS version this vhost should negotiate
    MinimumProtocolVersion string `json:"minimumProtocolVersion,omitempty"`
    // If Passthrough is set to true, the SecretName will be ignored
    // and the encrypted handshake will be passed through to the
    // backing cluster.
    Passthrough bool `json:"passthrough,omitempty"`
}
```

7.2 扩展Envoy

1. Envoy重定向支持 prefix_rewrite/strip_query
这种应用场景是，L7规则需要先根据前缀重写URL，再做重定向。这里对Envoy的定制是能够根据收到的请求动态生成重定向响应，我们在重定向下添加了一个新的配置 prefix_rewrite。同时为了删除重定向的查询字符串，我们添加了一个新标志strip_query，然后在执行重定向。
这个PR已经合并到社区 :https://github.com/envoyproxy/envoy/pull/2661

2. 支持通配 wildcard domain作为默认路由
假设Envoy配置了以下两条路由，如果我们去访问http://nomatch.com/api/application_data, envoy会因为匹配不到路由返回404.

```
[{
  "name": "noroutematch",
  "domains": ["nomatch.com"],
  "routes": [{
    "path": "/find1",
    "cluster": "root_www2"
  }]
}, {
  "name": "default",
  "domains": ["*"],
  "routes": [{
    "prefix": "/api/application_data",
    "cluster": "ats"
  }]
}]
```

相应的改动在source/common/router/config_impl.cc:

```
RouteConstSharedPtr specific_route_entry =
    virtual_host->getRouteFromEntries(headers, random_value);
if (specific_route_entry) {
  return specific_route_entry;
} else {
  // We could not find a route match in the specific domain. Look for routes in domain '*' for
  // any match.
  if (default_virtual_host_.get()) {
    return (default_virtual_host_.get())->getRouteFromEntries(headers, random_value);
  }
  return nullptr;
}
```

3.创建Envoy HTTP filter根据 L7 匹配执行Drop和Reset
Drop和Reset是已经负载均衡设备上两种很常见的Responder行为，顾名思义它能够将用户的请求直接丢弃或者重置。这种一般是和安全有关，比如检测到攻击我们可以创建一个L7规则快速的将请求Drop掉，而不会对后端服务器造成压力。

这里主要创建了一个access_control_filter.cc的HTTP过滤器，其主要代码如下：

```
FilterHeadersStatus AccessControlFilter::decodeHeaders(HeaderMap&, bool) {

  if (!matchesTargetUpstreamCluster()) {
    return FilterHeadersStatus::Continue;
  }
  if (config_->isResetEnabled()) {
    resetConnection();
    return FilterHeadersStatus::StopIteration;
  } else if (config_->isDropEnabled()) {
    dropRequest();
    return FilterHeadersStatus::StopIteration;
  }
  return FilterHeadersStatus::Continue;
}

void AccessControlFilter::resetConnection() {
  callbacks_->resetStream(true);
}

void AccessControlFilter::dropRequest() {
  callbacks_->dropStream();
}
```

正是因为有了这个过滤器，我们可以在IngressRoute里面使用reset/drop的service，用来执行重置和直接丢弃请求的响应。

```
  routes:
    - match:
        case_sensitive: false
        uri:
          prefix: /admin
      name: seo-admin-reset
      services:
        - name: reset
```

UFES对于Contour和Envoy的定制和扩展其实不止我提到的这些，有些扩展已经回馈到社区，而有的可能并不适合社区，但是都是为了解决了我们生产环境的实际问题。

8. 未来展望

1. 公网启用Anycast。这个是目前我们正在进行中的一个项目，有个别的公网应用已经启用了anycast，通过anycast可以带来几个好处：
    a. 更好的故障恢复能力，通过在多个数据中心配置同一个anycast IP，即使一个数据中心宕了也不会造成数据面的影响，而且故障路由收敛速度更快。

   b. 通过应用共享公网anycast IP可以减少IP数量，能够将长尾应用合并。
   c. 更低的延迟。我们比较了anycast和现在的unicast，整体上用户的访问速度提升了10%以上。
2. 公网流量接入数据中心基于Istio的应用网关。在UFES项目中我们实现了边缘节点的云原生网络，但是数据中心的应用架构还是基于两层的硬件负载均衡设备，未来的目标是将数据中心的硬件负载均衡也迁移到云原生网络上。

9. 小结

到目前eBay三大公网域的流量已经全部迁移到软件负载均衡设备，每天都接收几十亿次以上的用户请求进入数据中心，而且我们已经停止购买边缘节点的硬件负载均衡设备。网络云原生最大的好处是它能够通过开源软件的方式实现黑盒硬件提供的功能，并且在性能上没有损失，还能够实现弹性扩容，它带来的好处是值得长期投入的。

UFES项目在这一方面是eBay的先驱，它本身有很多不完美的地方，比如各种显得有些随意的annotation，没有定义完整的资源模型，甚至VIP之前都是人工分配，随着VIP的数量增多以及配置量的增加，控制面也有性能的问题正在解决。一部分的原因是因为在项目启动初期，社区的方案都还没有成熟，比如k8s的Ingress controller至今还没成熟的方案，另一个原因是因为eBay有一些定制的需求也是社区没有考虑到的，但是这个项目的成功上生产环境以及稳定运行将近一两年而没有出大的事故，这在给我们积累信心的同时，也证明网络云原生这条路是有希望的，进而也促进我们决心实施的另外一个项目--旨在替换数据中心的硬件负载均衡设备的基于Istio/IPVS的应用网关。

另外一点是，UFES这个项目是经过很个阶段的迭代的，比如第一个阶段只是建立一些基于硬件负载均衡设备的POP，带来的收益是桌面应用端访问延迟减少了200+ 毫秒，手机浏览器端减少了160+ 毫秒。而第二个阶段是在边缘节点实现软件的负载均衡，花费了将近两年的时间，这一个阶段还经历了将边缘节点拆分成前端代理和数据中心网关。第三个阶段是动态内容缓存以及网页内容预取（Content Pre-fetching），第四个阶段是全面接管公网的VIP以及集成unicast这个也是我们目前正在做的。

10. 写在最后

最后，这个项目的PaaS集成是始于笔者，其中的艰难深有体会：一方面是要实现各种自动化和各种集成，开发任务很重。另一方面又要承担部分ops的工作，几十个SLB，每个SLB又有若干个VIP需要自动化的初始配置，而且一旦遇到UFES的问题还需要很多个来回的配置软件负载均衡设备。更重要的一点是还有site incident的达尔摩斯之剑，公网流量的管理没有小事，所以能走到现在这一步也很不容易。另外这一两年有好几个参与UFES/PaaS项目的同事离职了，包括Qiu Yu大牛: 基于IPVS的L4负载均衡的作者，同时实现了基于一致性哈希的IPVS调度器的kernel模块，也参与了Contour的定制。另外Daxiong，接力完成了UFES前端代理和数据中心网关的拆分相关的PaaS集成，并且对接生产环境流量。还有Ji Yuan，L7统一管理终端作者，以及最近还有Zewei参与第四个阶段的集成。

还是回到那句话，这个项目有很多不完美的地方，但是这些付出都是有意义的，值得在这里记录一下。同时我希望借这边文章再阐释一下我对PaaS End to End的理解，PaaS不仅仅是集成，它也提供了一个机会，如何从无到有的构建一个云原生网络的系统，从小的看能够看到每个数据包是如何从客户端经过物理网络，主机网络，Linux内核，容器网络到达数据中心的后端应用层，再返回给用户；往大的看是如何在有很多系统依赖的情况下构建一个高稳定性的系统，并且尽可能的通过自动化减少人为操作。