HUAWEI

华为云学院

学以致用　云世界大有可为

Envoy原理介绍及线上问题踩坑

介绍人：张伟

华为云 +智能，见未来

www.huaweicloud.com

个人介绍

张伟

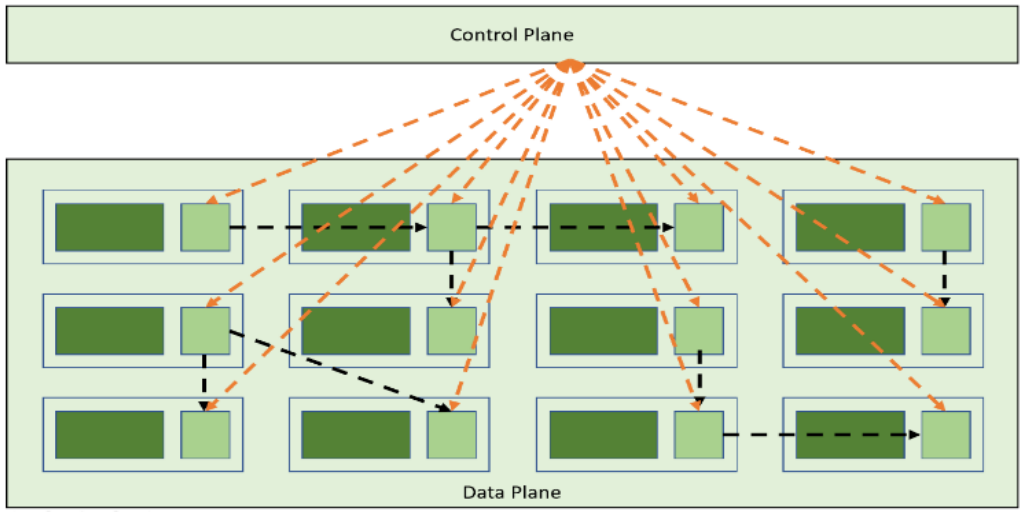
华为云容器网格数据面技术专家

拥有10年以上中间件及高性能系统开发经验，作为架构师及核心开发人员发布过传输网管系统、Tuxedo交易中间件、ts-server多媒体转码服务、GTS高性能事务云服务、SC高性能注册中心、ASM数据面等多个产品。先后就职于亿阳信通、北电、甲骨文、polycom、阿里巴巴等公司；目前在华为云云原生团队负责网格数据面的架构设计及开发工作。

目录

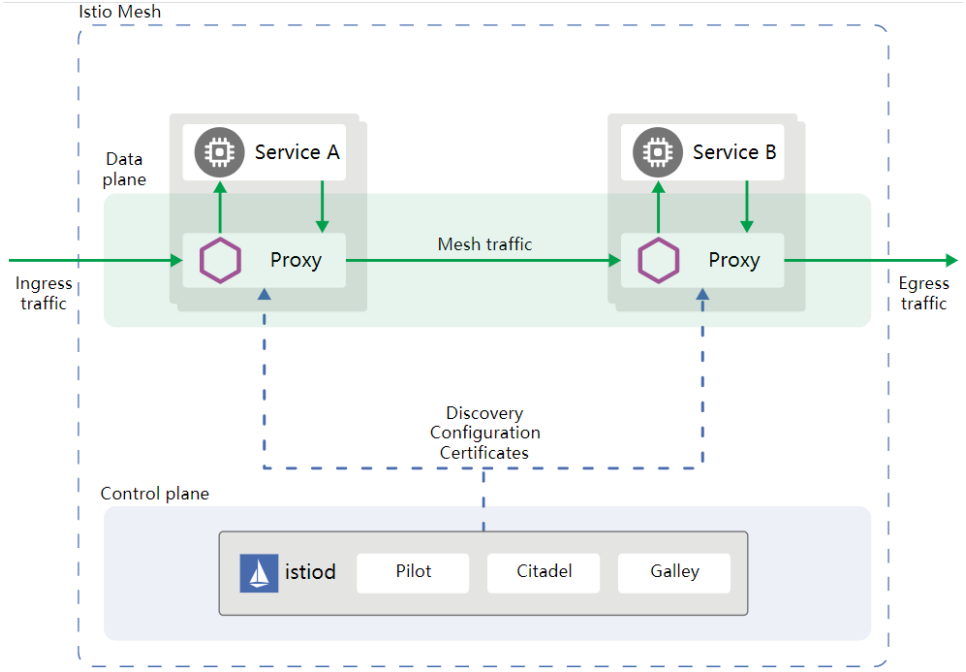
前言

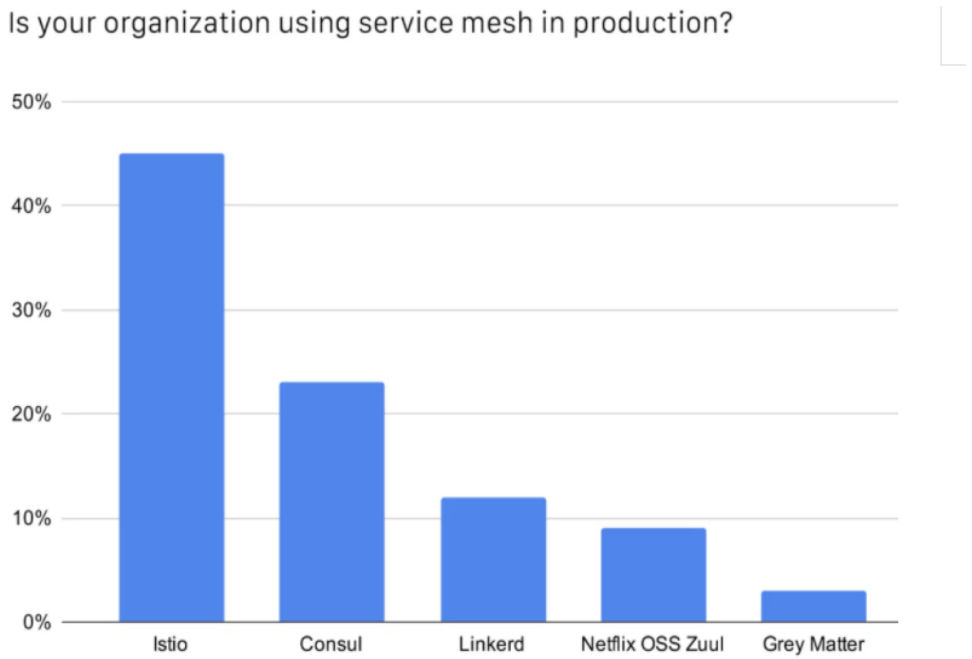
- 微服务架构最早由Fred George在2012年的一次技术大会上所提出，他讲到如何通过拆分SOA服务实现服务之间的解耦，这是微服务最早的雏形。
- 微服务架构通过细粒度的服务解耦拆分，带来缩短开发周期、独立部署、易扩展等好处的同时，同时带来对服务发现、负载均衡、熔断等能力前所未有的诉求。
- 第一阶段为dubbo、SpringCloud侵入式开发框架，语言强相关。
- 非侵入服务网格最早为2016年Linkerd。
- 2017年，Goole、IBM、Lyft发布Istio。Istio目前为服务网格的事实标准，并且是2019年Github增长最快的TOP 10开源项目之一。目前最新为1.10版本。



Envoy介绍

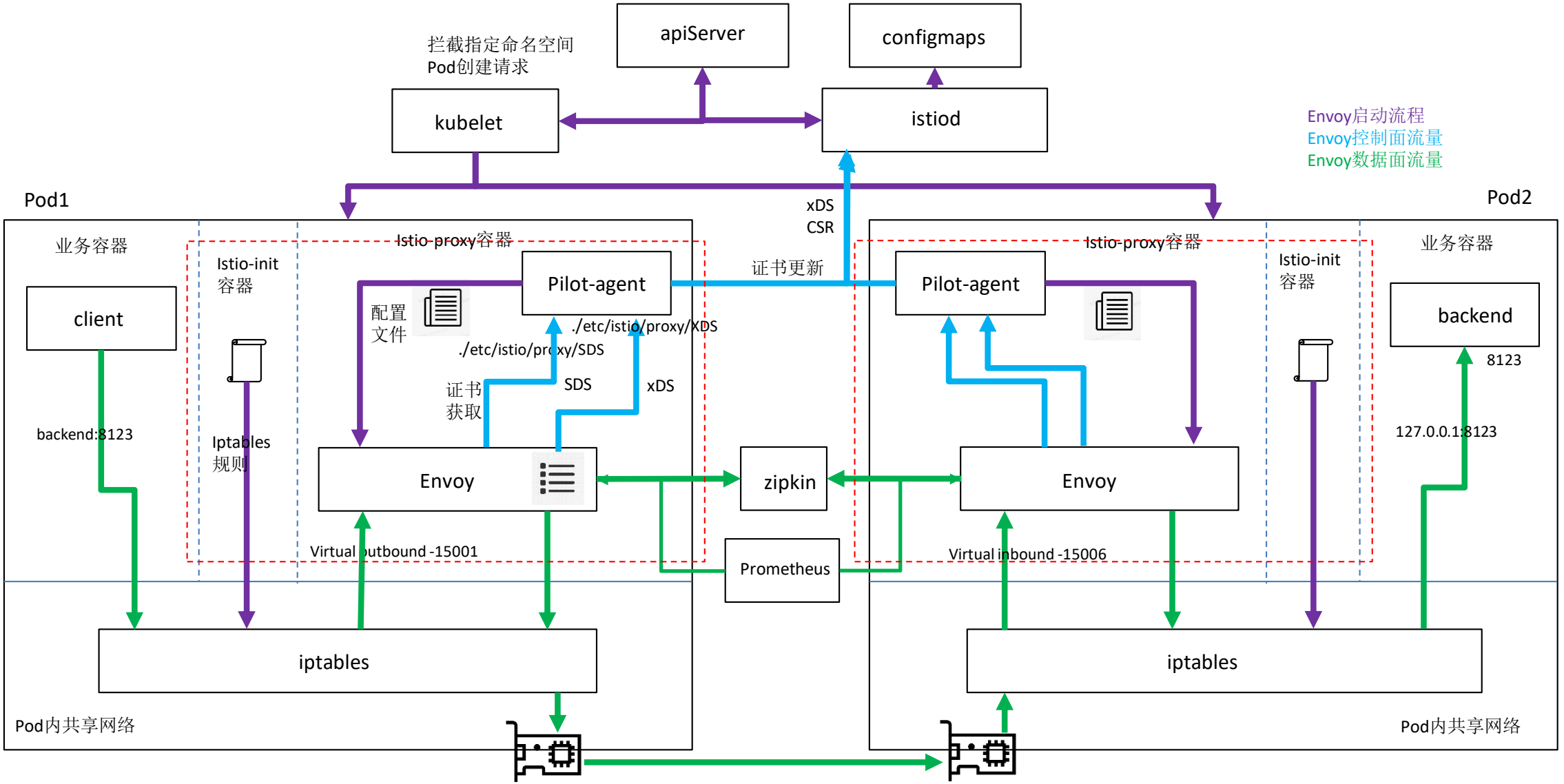
- Envoy采用C++实现，本身为四层及七层代理，可以根据用户应用请求内的数据进行高级服务治理能力，包括服务发现、路由、高级负载均衡、动态配置、链路安全及证书更新、目标健康检查、完整的可观测性等。
- 目前常见数据面主要有三种：Envoy、Linkerd、Traefic。Envoy由于高性能和扩展能力前在数据面遥遥领先。
- Iptables使Pod间出入应用的流量均由Envoy代理，对应用来说完全透明。支持主要常用网路协议Http1/Http2/Tls/gRPC/Tcp等。





Envoy原理及总体架构-启动

可以修改全局注入参数
作用于所有目标空间的
pod

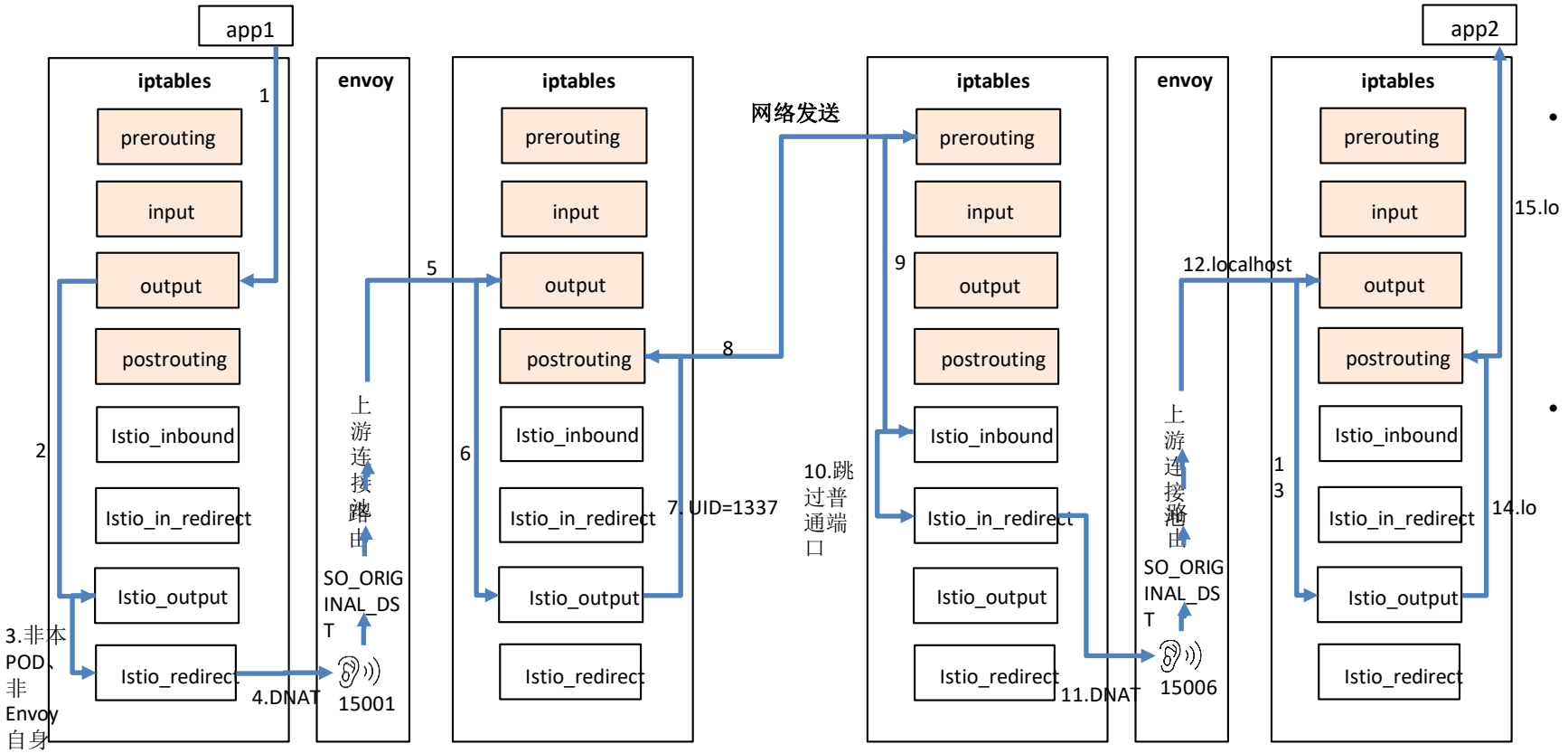


Envoy原理及总体架构-说明

- 1. 启动阶段
  - istiod拦截pod创建请求，识别为指定namepsace则根据configmap配置生成带有Envoy两个容器的创建POD请求，修改过的创建请求被kubelet接收，并在节点创建POD。
  - istio-init容器添加用于配置容器网络内iptables规则
  - istio-proxy容器启动pilot-agent进程，使用UID=1337 GID=1337创建Envoy启动命令行与配置文件
  - 可以通过自定义deployment内istio注解sidecar.istio.io/inject: “false”跳过自动注入过程，或修改部分启动参数。
- 2. 控制面通信
  - Pilot-agent进程本身创建UDS接收Envoy连接，用于证书更新下发。并且与istiod建立证书更新通道。
  - Envoy 通过pilot-agent转发机制与istiod建立长连接，通过xDS协议接收系统下发的监听器、路由、集群节点等更新信息。
- 3. 数据面通信
  - 客户端请求进入容器网络，并被iptables规则拦截，经过DNAT后进入Envoy virtualOutbound监听器
  - virtualOutbound经过监听过滤器恢复用于原始目标服务，并找到后端处理器处理新连接。
  - 后端处理器在配置中指定处理协议，根据协议相关的网络过滤器处理读取到的数据。
  - 如果为http协议，再经过请求过滤器处理http协议头部，如路由选择等功能并创建上游连接池
  - 将修改及编码后的http消息通过网络发送到对端Envoy的容器网络。
  - Iptables识别为入流量则进入virtualInbound端口。
  - ORIGINAL_DST恢复原始目标后，根据virtualInbound配置的监听过滤器找到对应的本地服务器地址。并发起localhost的请求。
  - 请求进入本地服务器内进行处理并返回响应。

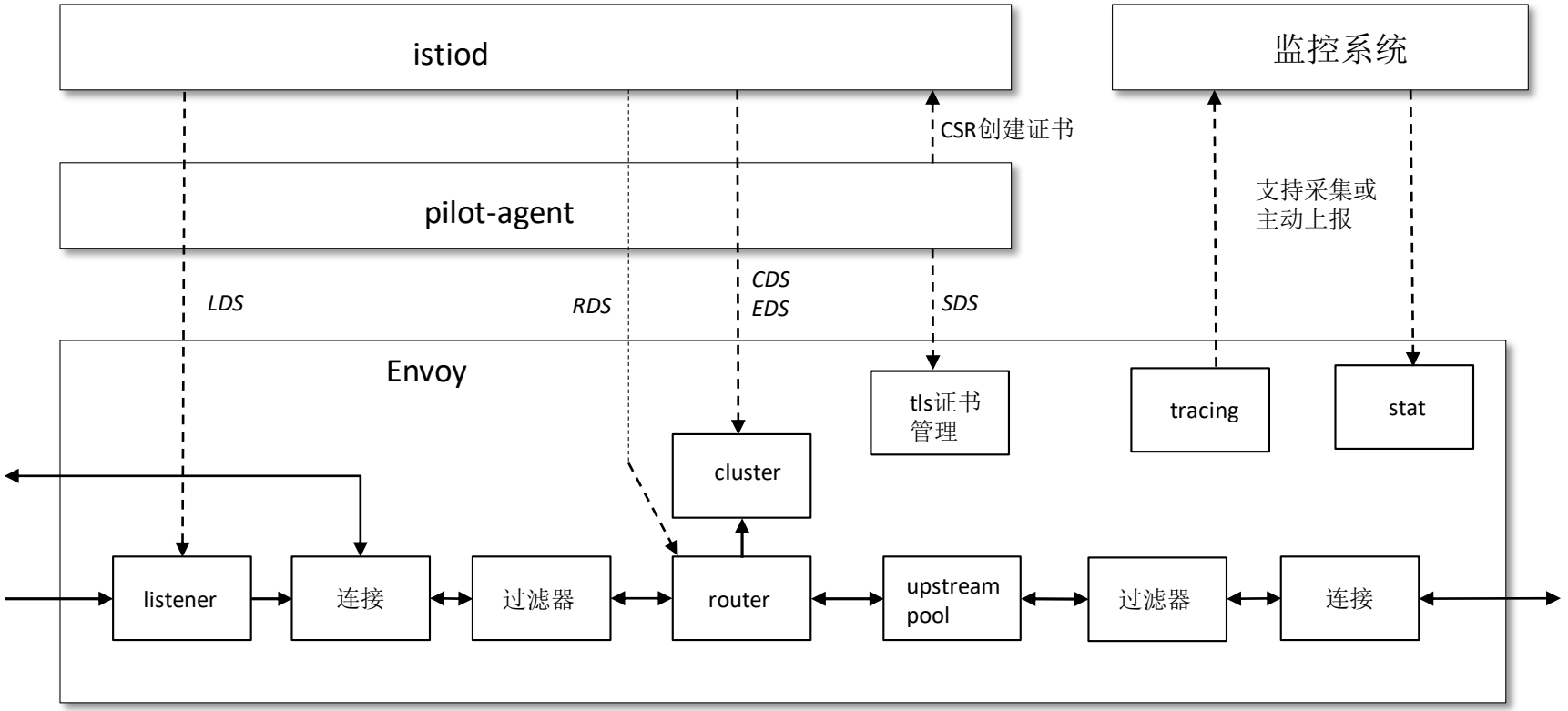
Envoy原理及总体架构-流量拦截

Envoy在每个POD网络空间内设置用于拦截流量的iptables规则



- outbound方向：本POD内发起对外调用流量
  - outbound方向增加ISTIO_OUTPUT、ISTIO_REDIRECT链。
  - 除目标为127.0.0.x及Envoy自身发出的流量外，其余都通过REDIRECT（DNAT）保存原始目标地址后，进入Envoy的15001端口。
- inbound方向：从二层网络设备进入POD内的流量
  - 增加ISTIO_INBOUND、ISTIO_IN_REDIRECT链。
  - 跳过15008、22、15090、15021、15020系统服务外，其余都通过REDIRECT（DNAT）保存原始目标地址后，进入Envoy的15006端口。

Envoy启动配置及xDS



xDS	描述	模式	请求路径
LDS	监听器配置	POST	/envoy.service.listener.v3.ListenerDiscoveryService/StreamListeners
RDS	路由配置	POST	/envoy.service.route.v3.RouteDiscoveryService/StreamRoutes
CDS	上游cluster配置	POST	/envoy.service.cluster.v3.ClusterDiscoveryService/StreamClusters
EDS	上游cluster endpoint配置	POST	/envoy.service.endpoint.v3.EndpointDiscoveryService/StreamEndpoints
SDS	安全及证书配置	POST	/envoy.service.secret.v3.SecretDiscoveryService/StreamSecrets

```
admin:
  access_log_path: /tmp/admin_access.log
  address:
    socket_address: { address: 127.0.0.1, port_value: 9901 }

static_resources:
  listeners:
  - name: listener_0
    address:
      socket_address: { address: 0.0.0.0, port_value: 10000 }
    filter_chains:
    - filters:
      - name: envoy.http_connection_manager
        config:
          stat_prefix: ingress_http
          codec_type: AUTO
          route_config:
            name: local_route
            virtual_hosts:
            - name: local_service
              domains: ["*"]
              routes:
              - match: { prefix: "/" }
                route: { cluster: some_service }
          http_filters:
          - name: envoy.router
  clusters:
  - name: some_service
    connect_timeout: 0.25s
    type: STATIC
    lb_policy: ROUND_ROBIN
    hosts: [{ socket_address: { address: 127.0.0.1, port_value: 8080 }}]
```

监听端口，连接入口

L4过滤，请求入口

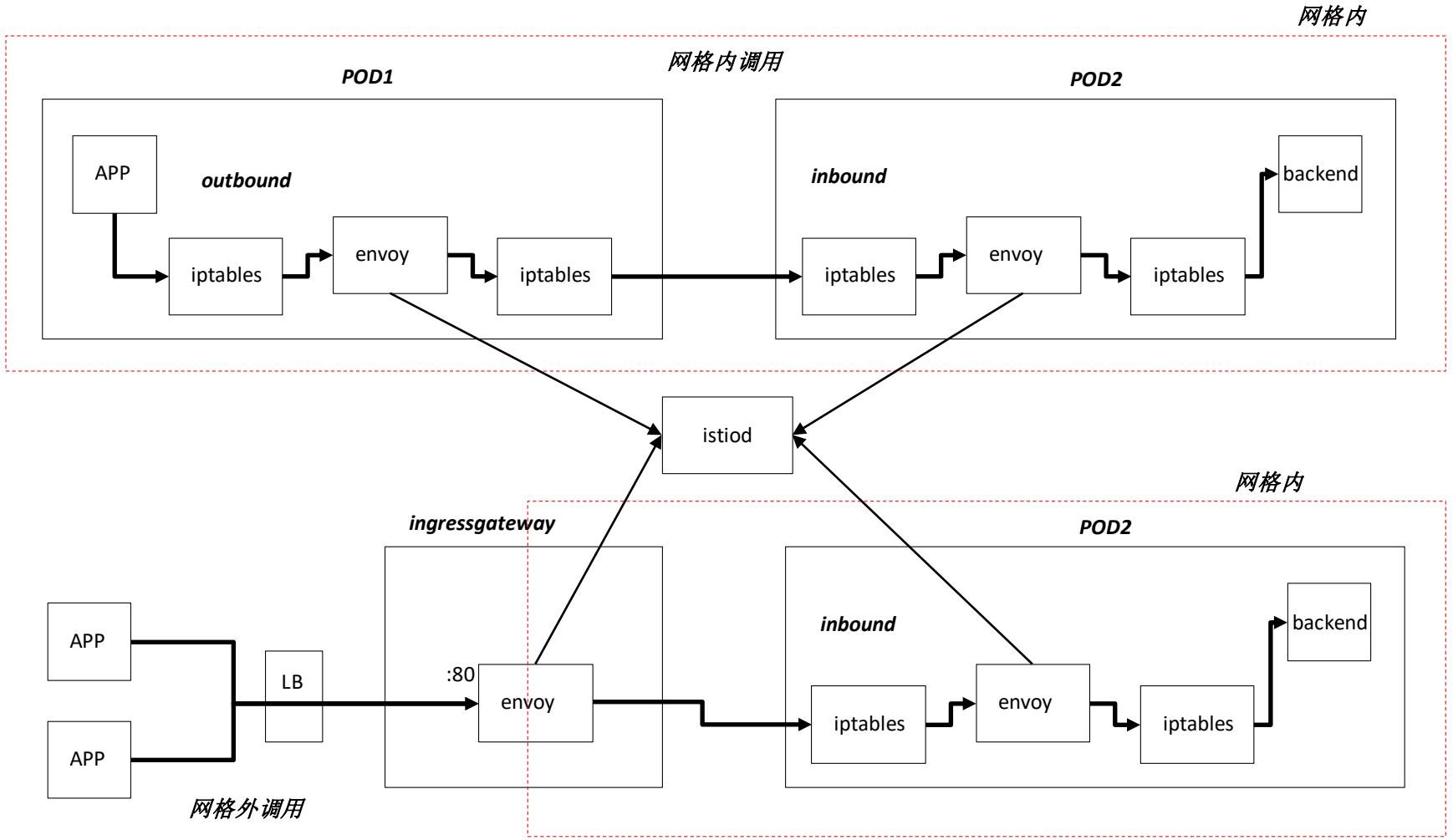
路由策略

L7过滤

上游集群

目标主机策略，请求出口

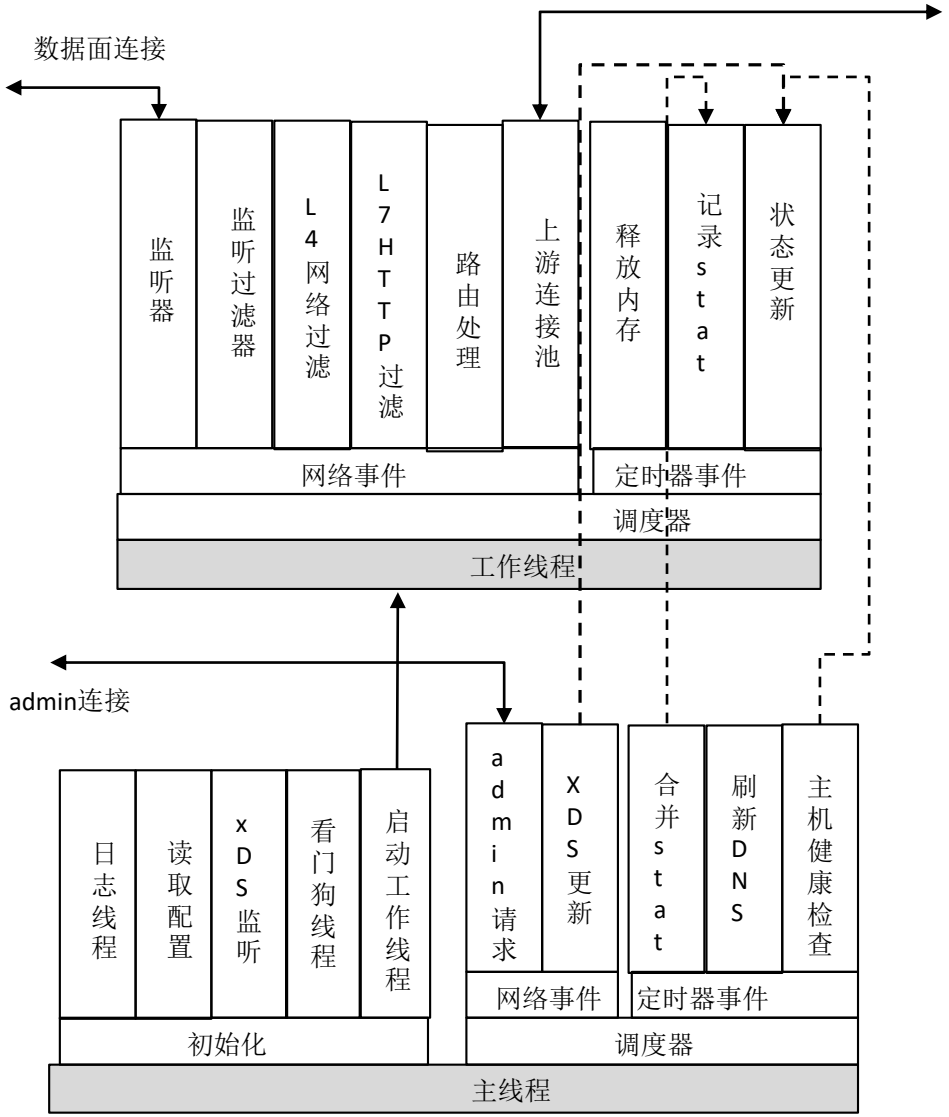
Envoy常用部署方式



- 网格内部调用，通过自动注入到网格内的iptables规则进行拦截
- 默认为2个工作线程
- 默认最大上游连接数1024，最大挂起等待请求数1024

- 外部请求通过直接访问ingressgateway网关端口进入网格
- ingressgateway为envoy相同二进制，不做iptables规则注入，作为外网客户端网格内代理。
- ingressgateway不设置工作线程数限制，并且最大上游连接及最大挂起等待请求数请求数默认值不做限制。

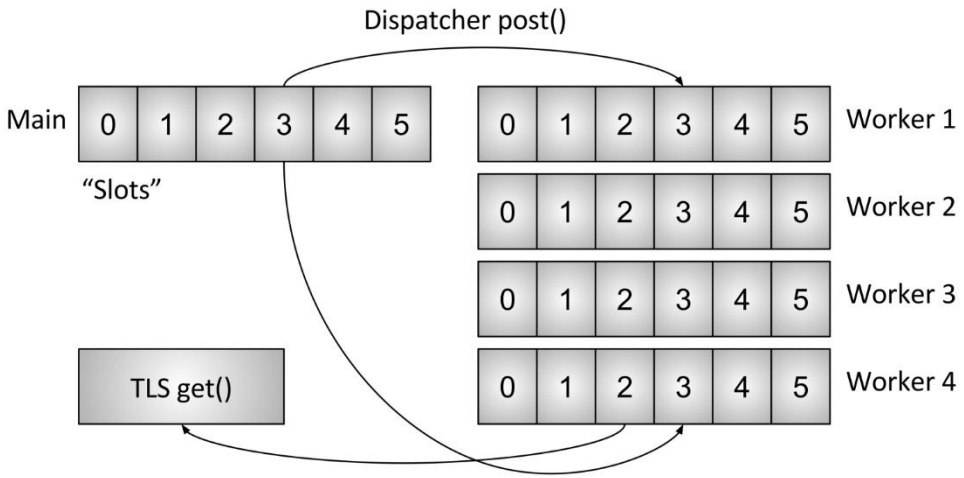
Envoy网络及线程模型



- 分为Envoy主线程及worker线程：
- 主线程：
  - 负责初始化Envoy并读取解析配置文件
  - 启动gRPC监听器，并启动xDS变化监听
  - 启动日志写入线程，每个目标日志文件有独立线程负责输出
  - 启动concurrency数目的工作线程
  - 启动看门狗线程监控各个工作线程是否定期touch，否则SIGABRT杀掉线程
  - 启动admin RESTful监听，处理运行状态输出，prometheus收集等请求
  - 定期将工作线程内监控数据stat进行合并
  - 定期刷新DNS信息，加速域名解析。
  - 目标cluster内主机列表健康状态判断。
- worker线程：
  - 通过启动配置参数concurrency指定，不支持动态调整。
  - 启动virtualoutbound/virtualinbound网络监听，每个工作线程都对此监听端口进行监听。由内核随机挑选监听线程处理新连接。
  - 进行连接负载均衡处理后，选择最终的业务监听器处理新连接。
  - 之后对于此连接的所有处理都在此线程进行，包括下游数据集解码，路由选择、上游数据编码发送等。
  - 同时此工作线程还要处理定期观测信息与主线程同步（通过异步加回调）、线程内配置及集群管理器状态更新等工作。
  - 请求完成后延迟释放内存，解决本次事件处理中回调所引用对象可以被安全访问，并在下次事件处理中安全删除。

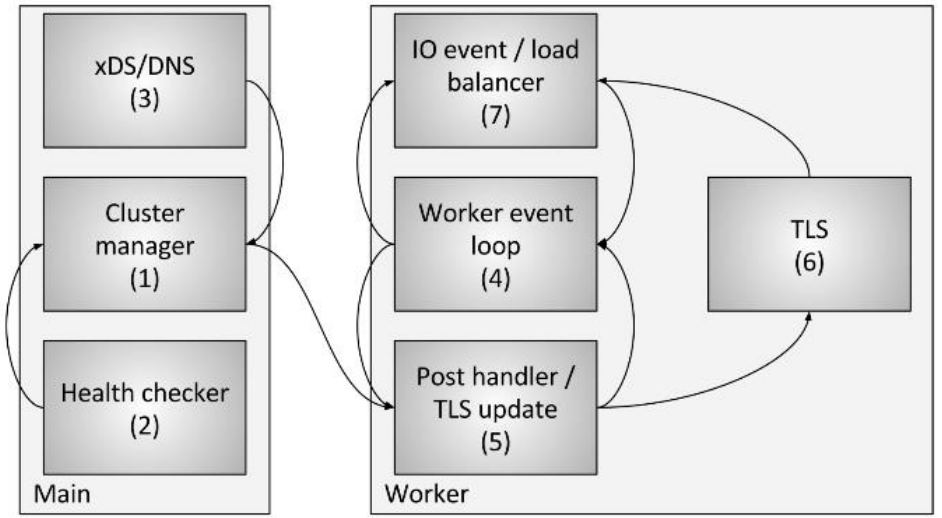
Envoy网络及线程模型-共享数据同步

- 1. 调度器通过epoll监控文件事件(网络）及定时器事件进行排队任务处理
- 2. 线程间通信通过post接口发送任务，此任务通过定时器事件激活
- 3. 线程间数据交换通过post更新TLS，这样每个线程内代码都不需要加锁处理
- 4. 每个线程的TLS对象本身只保存真实对象的共享指针进行读操作，减少内存消耗。
- 5. 全局对象更新只发生在主线程，并通过COW方式通知工作线程进行指针修改
- 每个TLS slot通过allocateSlot分配，在使用前通过set在每个线程中创建一个拷贝并保存。
- 在主线程中调用此slot的runOnAllThreads在所有线程中延迟执行回调，回调内更新每个线程内拷贝对象状态

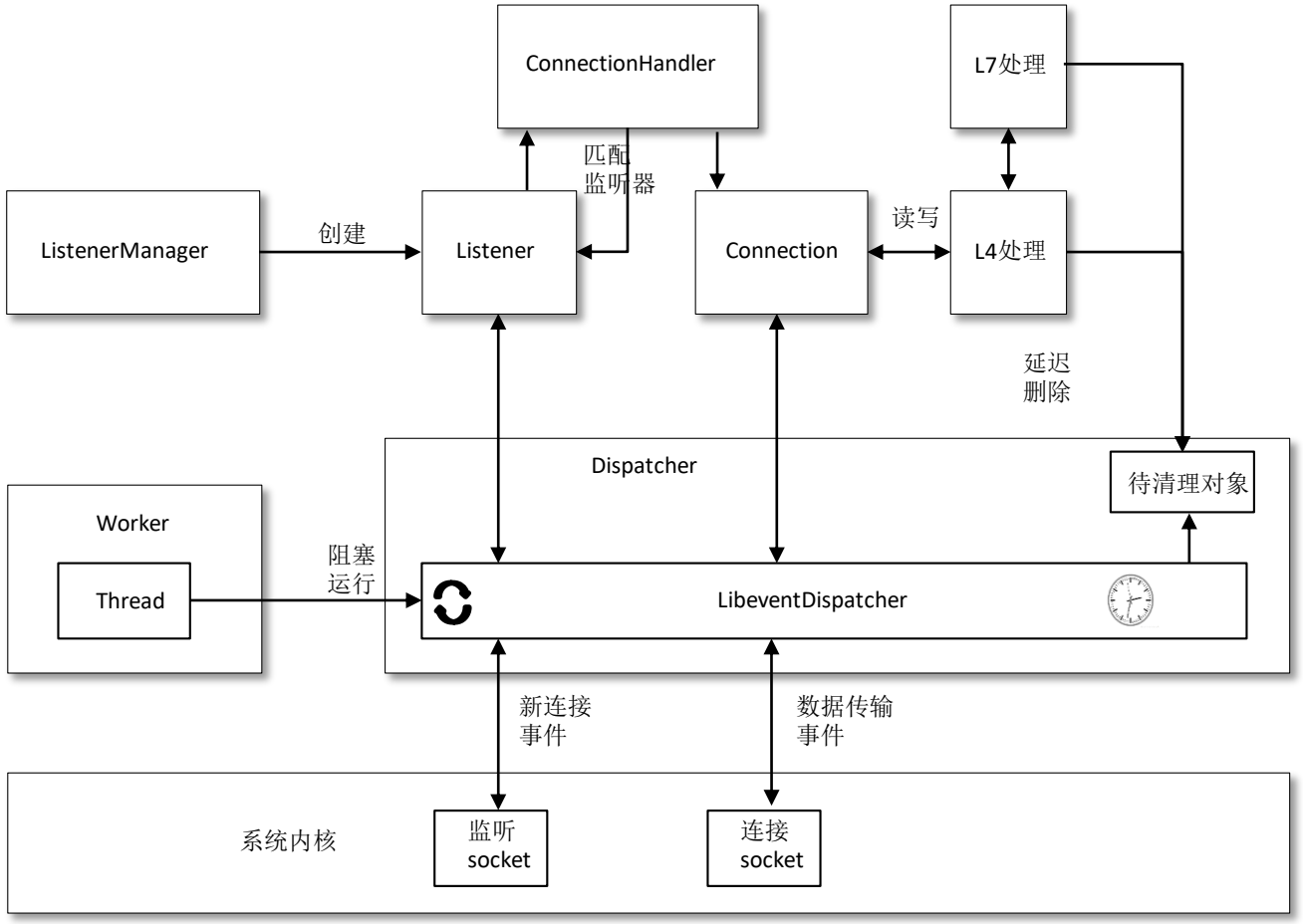


Envoy网络及线程模型-集群信息更新

- 1. Envoy需要在运行中支持添加集群并监控每个集群内目标主机列表变化
- 2. 监控到目标主机健康状态变化时，需要通知到工作线程内主机可用状态。
- 3. 当收到节点变化EDS消息时，需要通知到工作线程内新上线、下线主机。
- 4. Envoy使用前面提到的TLS方式实现集群状态更新，集群管理器保存一个TLS slot，类型为ThreadLocalClusterManagerImpl。当节点变化、DNS解析更新、健康状态变化时，将调用集群管理器的postThreadLocalClusterUpdate方法
- 5. 此方法将延迟调用所有线程内ThreadLocalClusterManagerImpl slot的回调函数
- 6. 此函数内将保存新clusterEntry对象的引用。
- 7. 下一轮请求解析时将从头TLS中获取到更新后的集群可用状态。

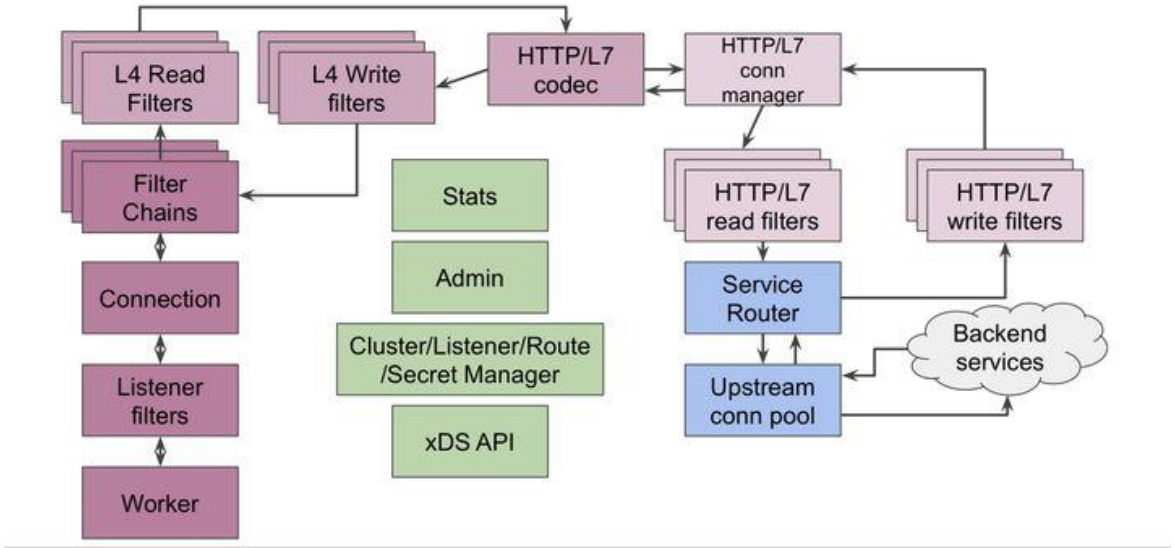


Envoy网络及线程模型-网络处理



- Envoy启动时创建的Listener与工作线程绑定并启动监听（向libevent注册Read回调onSocketEvent），并进入阻塞运行状态，直到进程退出。
- 当新连接到达时，内核网络协议栈调用回调并创建新连接的Socket。
- 通过ConnectionHandler调用监听过滤器获得真实访问目标地址
- 根据目标地址匹配得到业务监听器后创建Connection连接对象
- 之后Connection对象再次向libevent注册Read/Write回调onFileEvent，并作为L4层过滤管理器处理onNewConnection，onData数据接收。
- 对于HTTP协议，将继续经过L7层编解码处理后向上游发送请求。
- 当请求处理完毕后，将调用deferredDelete删除请求对象并记录统计观测数据。
- 使用异步I/O方式发送网络数据，降低对线程内其他操作的阻塞。

Envoy过滤器架构

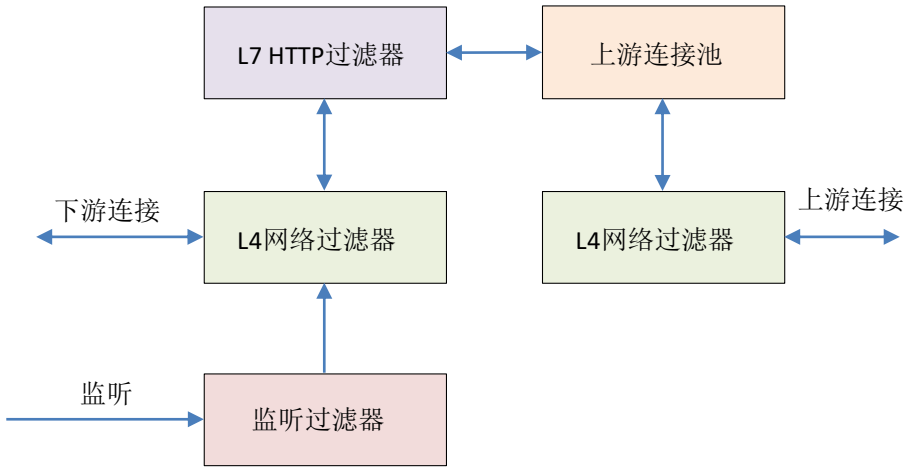


- 根据位置及作用类型，分为：
  - 监听过滤器（Network::ListenerFilter）
    - onAccept接收新连接，判断协议类型，TLS握手，HTTP协议自动识别、提取连接地址信息

  L4 网络过滤器
    - HTTP、Mysql、Dubbo协议处理、元数据交换，四层限流，开发调试支持等。
    - onNewConnection新连接建立，可以决定是否拒绝
    - onData处理连接数据到达
    - onWrite处理连接数据发送

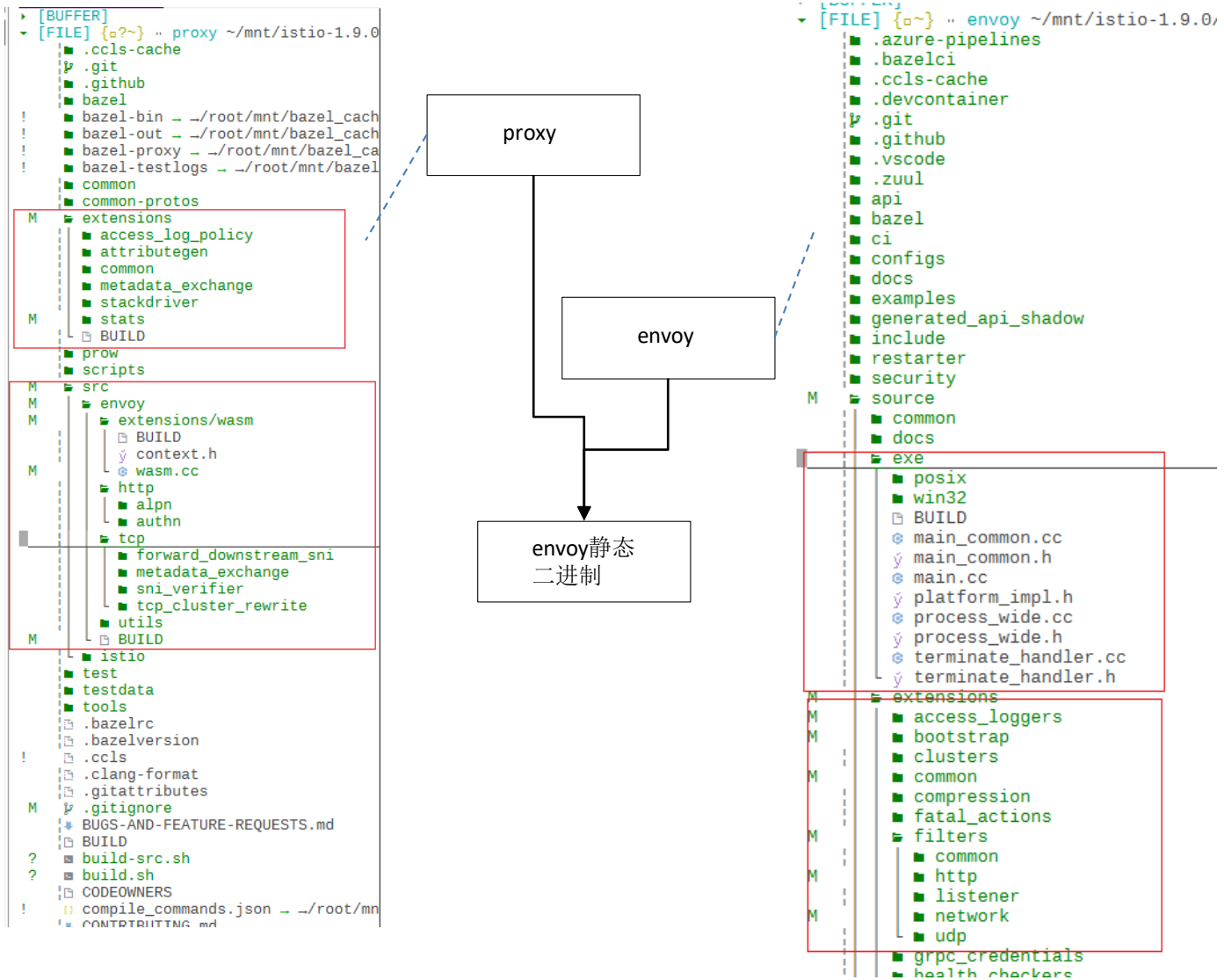
  L7 HTTP过滤器
    - 修改HTTP请求头，限流处理，Lua扩展、WASM扩展、开发调试支持、压缩、元数据交换、路由等。
    - decodeHeaders处理HTTP请求头部
    - decodeData处理HTTP请求数据
    - decodeTrailers处理HTTP请求结束位置
    - encodeHeaders发送请求前编码头部
    - encodeData发送请求前编码数据
    - encodeTrailers消息发送前编码处理
- 过滤器中可以获取连接对象并直接发送响应数据，同时可以返回StopIteration结束过滤器迭代。
- 实际使用过滤器根据Envoy静态及动态配置注册，并可以在运行中通过EnvoyFilter动态添加或删除。

Envoy过滤器架构-常用过滤器



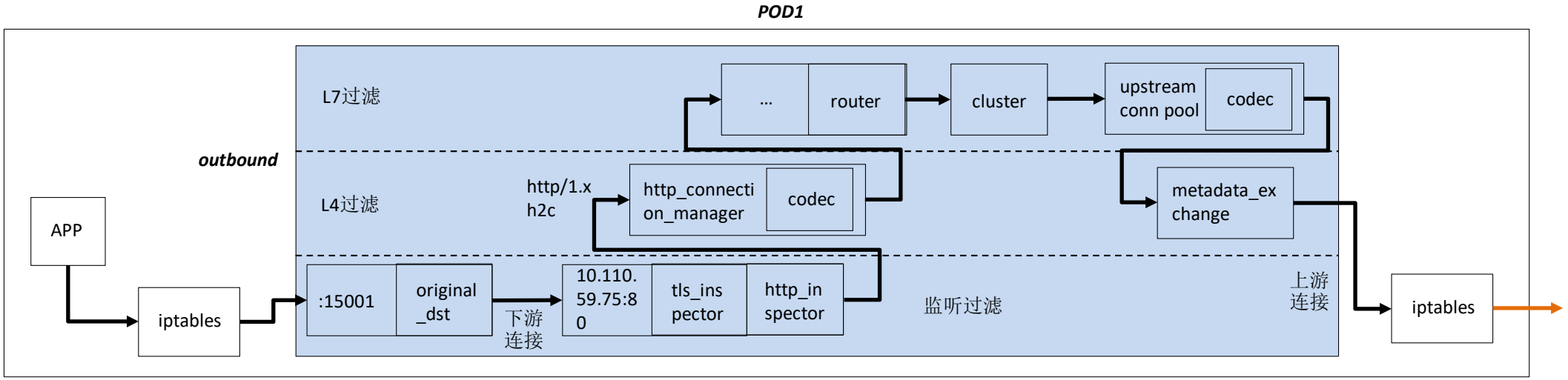
插件名称	工作模式	功能介绍
envoy.listener.tls_inspector	监听过滤器	检测下游连接是否为TLS加密，并且获取ALPN（应用层协商协议），用于网络层过滤器匹配判断。
envoy.listener.http_inspector	监听过滤器	检测应用层协议是否HTTP，并判断具体类型为HTTP/1.x或HTTP/2，用于网络过滤器匹配判断
envoy.listener.original_dst	监听过滤器	根据Socket上属性SO_ORIGINAL_DST获取iptables DNAT前的目标服务地址，作为后续负载均衡的输入。
envoy.filters.network.tcp_proxy	L4网络过滤器	基于L4层1对1上下游网络连接代理
envoy.filters.network.wasm	L4网络过滤器	基于WASM（WebAssembly）技术，支持沙箱、热升级、跨语言的扩展机制，处理L4层新连接、数据收发。
envoy.filters.network.dubbo_proxy	L4网络过滤器	解析dubbo RPC协议并提取请求中方法、接口、metadata等信息，并根据元数据进行路由选择。
envoy.filters.network.local_ratelimit	L4网络过滤器	基于L4层网络限流，通过令牌桶防止定期时间间隔内过多下游连接。
envoy.filters.network.http_connection_manager	L4网络过滤器	专门用于处理HTTP请求的网络过滤器，根据协议类型处理HTTP编解码并调用L7层HTTP过滤器。
envoy.filters.http.lua	L7 HTTP过滤器	基于Lua脚本语言，处理HTTP请求及相应，每个Lua运行时运行在工作线程中。
envoy.filters.http.local_ratelimit	L7 HTTP过滤器	基于L4层请求限流，通过令牌桶防止定期时间间隔内过多下游请求
envoy.filters.http.wasm	L7 HTTP过滤器	基于WASM（WebAssembly）技术，支持沙箱、热升级、跨语言的扩展机制，处理L7层HTTP请求编解码。
envoy.filters.http.router	L7 HTTP过滤器	HTTP路由及负载均衡的入口点，作为L7层过滤器链中最后一个过滤器。

Envoy过滤器架构-相关代码



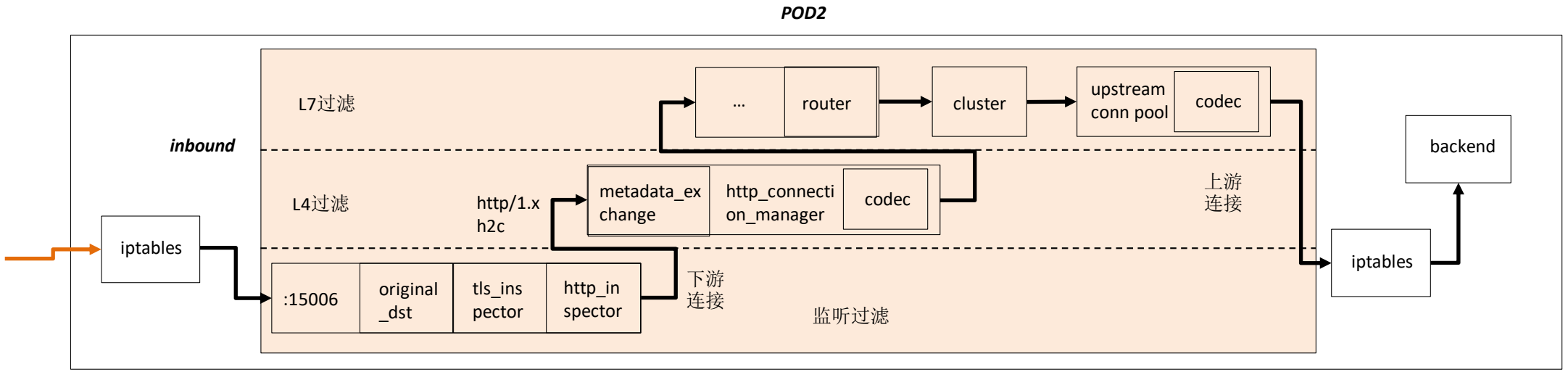
- Istio项目中Envoy代码分为两部分：
  - Envoy原始项目的clone，在 https://github.com/istio/envoy.git
  - Istio中适配所使用的的插件 https://github.com/istio/proxy.git
- 编译时由proxy项目作为入口，自动引用envoy项目
- 主要框架代码位于envoy项目，包含进程启动，线程及网络、主要过滤器框架，观测数据处理等。
- 启动入口点位于envoy项目 source/exe目录下
- proxy项目中主要提供metadata_exchange，stats等必要WASM扩展
- envoy项目中过滤器插件主要位于source/extensions/filters，listener目录包含监听过滤器，network目录包含L4层网络过滤器，http目录包含L7层HTTP过滤器

EnvoyHTTP请求流程



- APP发出的请求被iptables拦截，并根据源信息判断为outbound被DNAT后拦截进入Envoy 15001端口
- 15001上监听器通过ORIGINAL_DST获取原始目标地址（服务的clusterIp），匹配业务监听器（不真正监听网络）地址并传递新建下游连接。
- 下游连接过滤器判断TLS，ALPN（应用协议名），HTTP版本后匹配到L4层http_connection_manager网络过滤器。
- http_connection_manager使用http codec解码http协议header/body/tailer等并触发回调函数。
- http header/body处理回调中将调用L7层HTTP过滤器处理（可修改http原始请求等）最后调用Router过滤器。
- Router过滤器负责根据配置中路由部分及请求内url等进行匹配并找到目标cluster。
- 根据cluster的负载均衡策略及当前可用POD实例信息，选择最适合的目标POD地址，并创建此目标地址的连接池。
- 根据连接池配置的最大可用连接数及允许的最大等待连接数等信息将下游请求与上游连接进行关联。
- 当连接准备好后对下游请求使用codec进行HTTP编码，并发送到上游连接的L4层网络过滤器。
- 上游连接的L4层网络过滤器使用metadata_exchange传输本Envoy内保存的与调用APP相关的元数据信息，包括POD名称、用户空间、cluster_id等信息。之后将待发送请求通过Socket发送到网络，经过iptables时判断发送者为Envoy则不再拦截。

EnvoyHTTP请求流程



- 目标POD收到从网络进入的流量，通过iptables拦截后判断为inbound并DNAT处理后，进入Envoy的15006端口。
- 由于inbound方向流量的目标地址一定为本POD地址，无需再提取后转到不同业务监听器，因此virtualinbound负责TLS、ALPN及HTTP协议版本判断。
- L4层网络过滤器metadata_exchange首先解析请求中包含的源POD元数据信息，并同时发送回本Envoy代理POD的元数据信息。之后进入L4层http_connection_manager网络过滤器。
- L7层处理流程同outbound方向，区别为inbound通过Router匹配后的目标cluster所指向的上游地址为127.0.0.1。
- 之后创建与本POD内业务容器的服务端口的Socket连接并完成请求发送。
- 由于请求方向在建立时会保存下游与上游连接的双向对应关系，因此Response匹配的对上游进行L7层过滤器解码、通过Router关联关系找到下游并编码发送HTTP请求（POD1处理类似）。
- 以上所提到的Envoy L4层网络读取及数据发送为全异步读写模式，采用网络事件触发机制完成响应数据的接收和发送。
- 由于Router部分请求处理方向需要进行更多路由选择计算及负载均衡计算工作，因此通常outbound方向处理较复杂，CPU消耗比inbound更高。

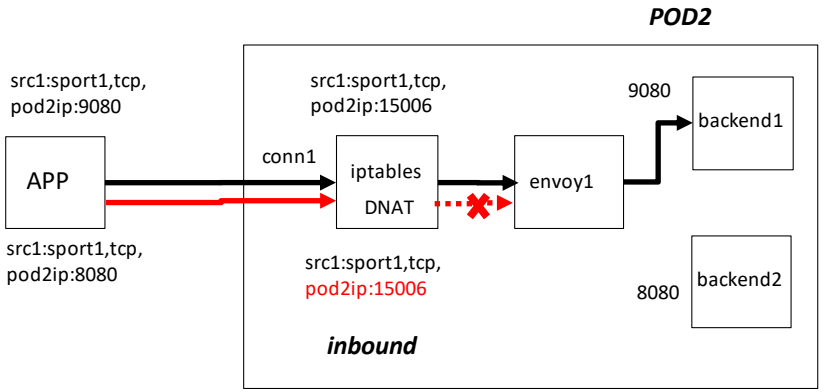
生产环境问题分析及解决方法（1）

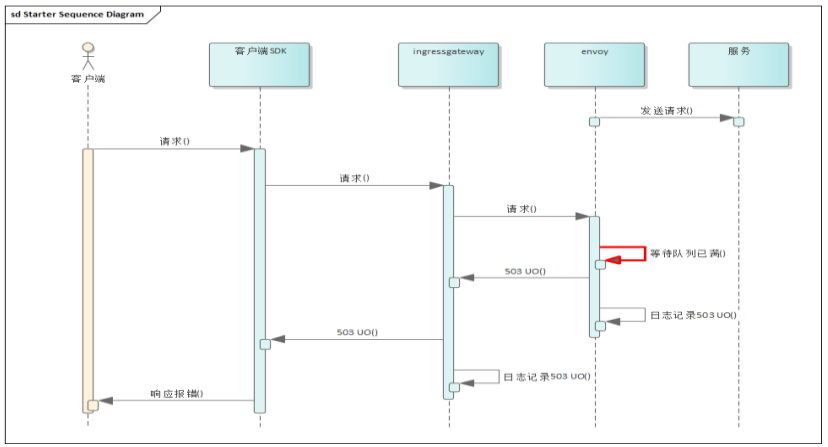
503 Service Unavailable

由于临时的服务器维护或者过载，服务器当前无法处理请求。这个状况是暂时的，并且将在一段时间以后恢复。[61]如果能够预计延迟时间，那么响应中可以包含一个Retry-After头用以标明这个延迟时间。如果没有给出这个Retry-After信息，那么客户端应当以处理500响应的方式处理它。

TCP五元组：(不能重复，否则conntrack无法区分)

srcip:srcport,prot,dstip:dstport





503 UF问题分析

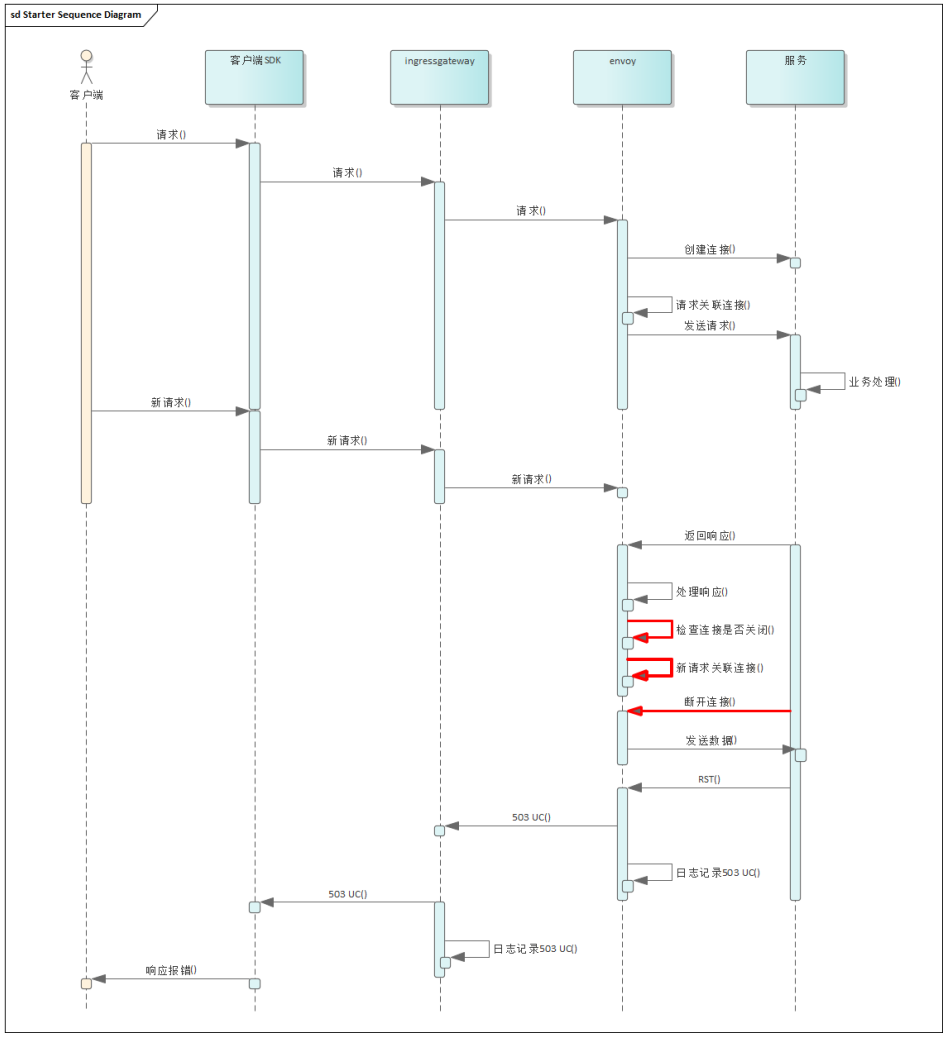
现象	日志报错503 UF，等待8S后建立连接失败。 日志如下： [2021-02-09T06:29:10.489Z] "GET /v1/xx/xx/xx/xx HTTP/1.1" 503 UF "-" "-" 0 91 288 - "100.95.165.3" "xx-xx" "513cca39-1ea7-47db-8c04-a5827464ce22" "100.85.225.193" "10.17.10.181:xx" outbound|xx|191130102|xx.xx.svc.cluster.local- 10.17.8.9:xx 100.95.165.3:28788 100.85.225.193 -
原因分析	1. 抓包看到出现短时间内大量Retransmission 2. 超过cluster内建立与目标主机连接的最大重试次数（3次） 3. 原因为客户POD内部署两个对外服务端口，当客户端同时发起对不同服务的访问时，路由结果可能会落到相同的POD内，第一个连接目标为9080，第二个连接目标为8080。由于目标服务端口不同，通过五元组规则可知，src-ip:src-port可以相同。而由于目标POD内流量被自动DNAT拦截入15006端口，此时目标dst-ip:dst-port被临时替换为envoy-ip:15006，此时将无法区分两个连接的流量。因此当第一个连接建立成功后，第二个连接的SYNC包将被当作重复包丢弃，导致第二个连接建立失败。
解决方案	1、与客户沟通拆分两个微服务到不同的POD（符合微服务拆分原则） 2、如果无法拆分微服务，则需要解决源端口重用的问题，目前没有采用此种方法。

503 UO问题分析

现象	日志报错503 UO 日志如下： [2021-03-31T11:16:55.538Z] "GET /aaabbbcccddd HTTP/1.1" 503 UO "-" "-" 0 81 5 - "-" "-" "3c2a392c-56fc-9d8c-9895-f657a4444679" "test-503-svc:8080" "-" - - 10.106.246.126:8080 10.244.92.179:48788 - default
原因分析	1. 上游服务正在处理中的请求为最大连接数maxConnetions 2. 未被处理请求进入等待队列大小为maxPendingRequests（默认1024） 3. 新请求超出等待队列最大数量报overflow，日志记录503 UO
解决方案	1、检查服务处理能力，缩短服务处理时间或水平扩展。 2、调整等待队列请求上限值。

生产环境问题分析及解决方法（2）

503 UC问题分析



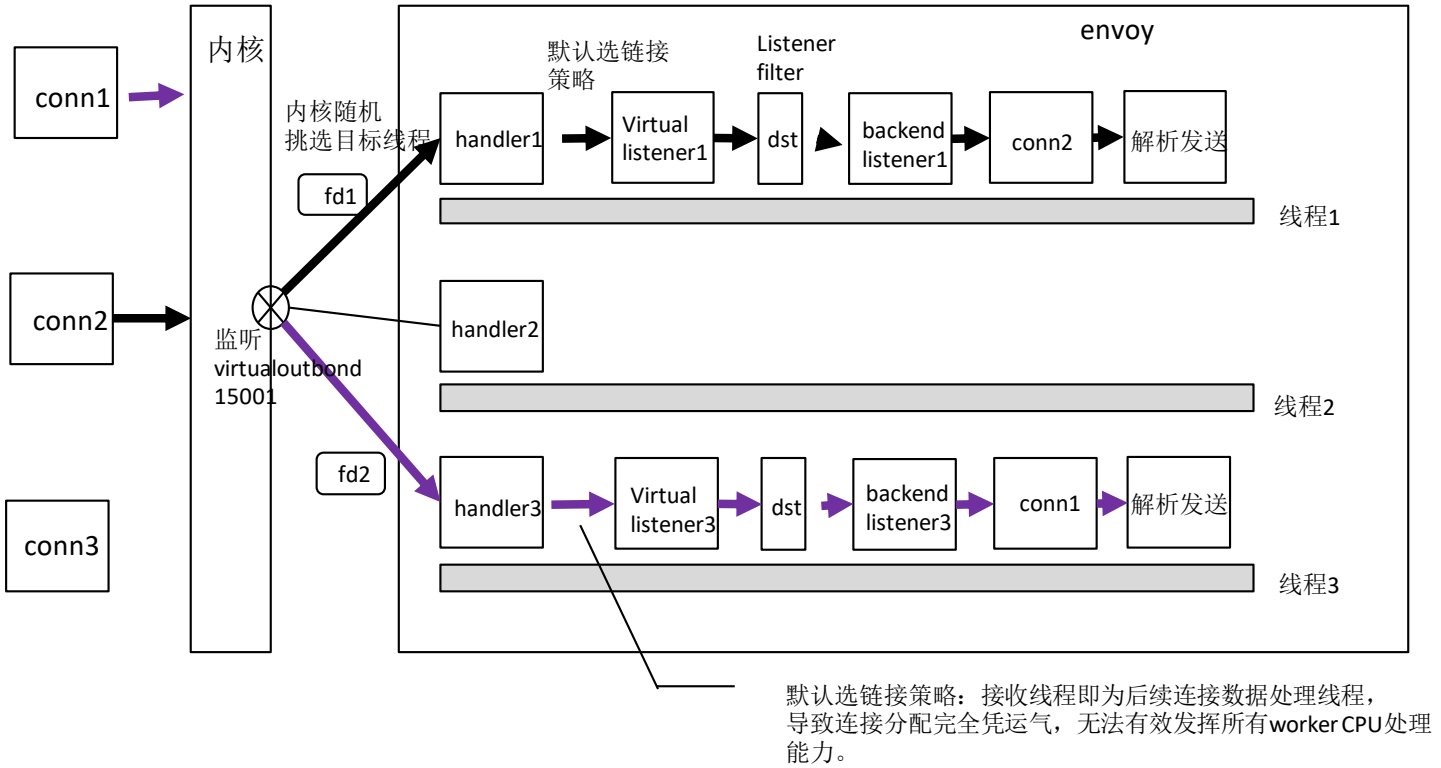
现象	一小时内会不规则的发生几次关于查询xx、进入xx遇到的503 UC错误，请求时延很短，但会断开连接。 日志如下： [2021-03-03T10:32:47.139Z] "POST /v1/xx/xx/xx/xx/xx/983980038/stopxx HTTP/1.1" 503 UC "-" "-" 0 95 1 - "10.13.22.7" "Apache-HttpClient/4.5.12 (Java/1.8.0_232)" "U4REJ819523DU961535U8316KUUG2G3X" "10.18.8.13:28443" "10.19.51.51:xx" outbound|xx|210201100|xx.xx.svc.cluster.local - 10.19.100.71:28443 10.13.22.7:34706 - -
原因分析	1. 对于http1.1协议envoy默认重用与服务端建立的连接 2. 当服务返回响应后，会主动关闭此连接发送FIN 3. Envoy收到响应后，将尝试检测下此连接是否关闭，如果已经关闭就从可用连接列表中移除 4. 存在时间窗口导致envoy检测时此FIN还没有收到，且请求处理为单线程，则认为连接依然可用 5. 将新请求绑定到连接上（可能已经关闭）进行数据发送 6. 系统对于已经关闭的连接返回RST，envoy产生503 UC上游断开报错 总结：Envoy在处理响应后尝试检测连接状态，存在时间窗口没有检测到断开，并重用此连接发送后续请求而遇到网络RST，并且Envoy连接池不会对当前连接发送失败后进行重试，导致Envoy返回503UC报错。
解决方案	1、禁止服务侧Envoy对上游重用连接，通过设置maxRequestsPerConnection=1可以动态生效，但会有一定性能下降。同时不影响上游连接数maxConnetions（默认1024）和最大等待请求数量maxPendingRequests（默认1024） 2、配置目标cluster重试策略，同时需要修复针对inbound配置时新参数与缺省参数冲突不生效的问题:

```
apiVersion: networking.istio.io/v1alpha3
kind: EnvoyFilter
metadata:
  name: inbound-max-pending
  namespace: default
spec:
  workloadSelector:
    labels:
      app: 503-server
  configPatches:
    - applyTo: CLUSTER
      match:
        context: SIDECAR_INBOUND
        cluster:
          name: "inbound|8080|http|
          portNumber:
      patch:
        operation: MERGE
        value:
          circuit_breakers:
            thresholds:
              - max_pending_requests:
                max_retries:
```

```
configPatches:
  - applyTo: HTTP_ROUTE
    match:
      context: SIDECAR_INBOUND
      routeConfiguration:
        name: "inbound|8080|http|
        portNumber:
        vhost:
          name: inbound|http|
    patch:
      operation: MERGE
      value:
        route:
          retryPolicy:
            retryOn: 5xx,gateway-error,connect-failure,refused-stream,retriable-4xx
            numRetries: 3
            perTryTimeout: 5s
            #retryHostPredicate:
            #- name: envoy.retry_host_predicates.previous_hosts
            hostSelectionRetryMaxAttempts: 5
```

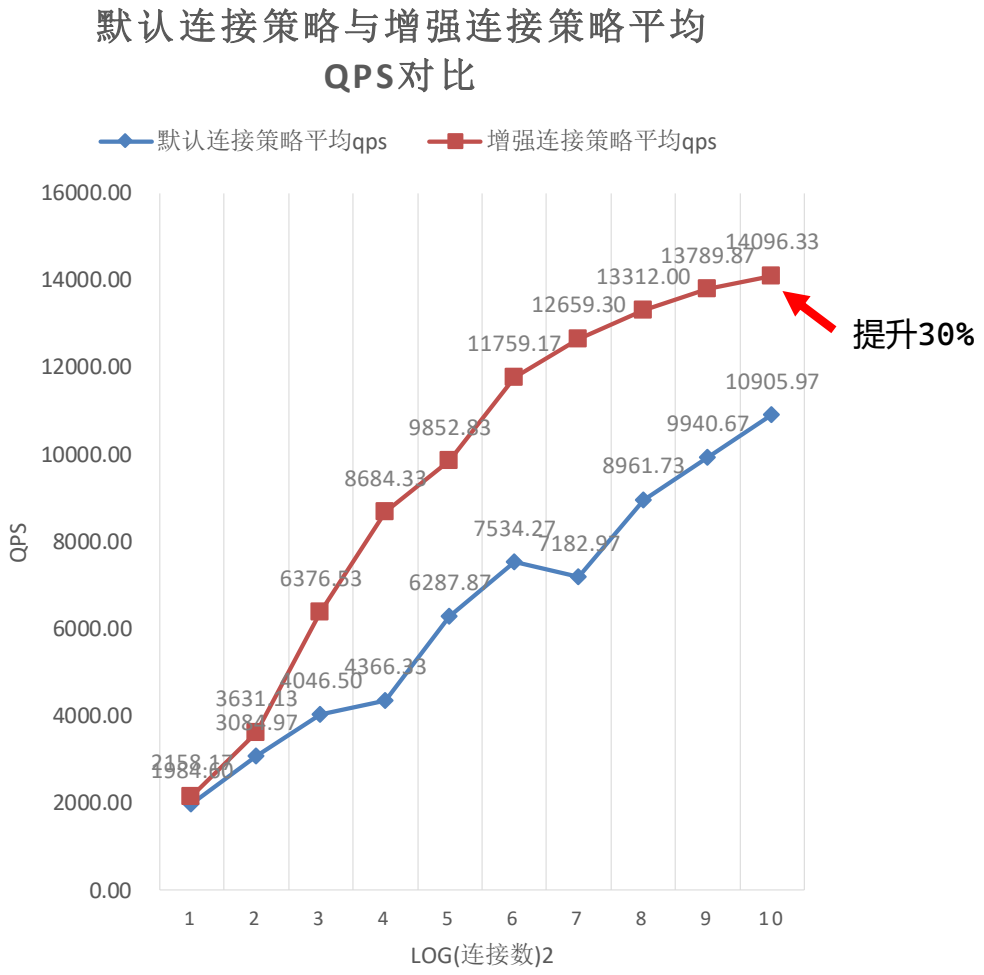
针对Envoy做的一些优化及效果

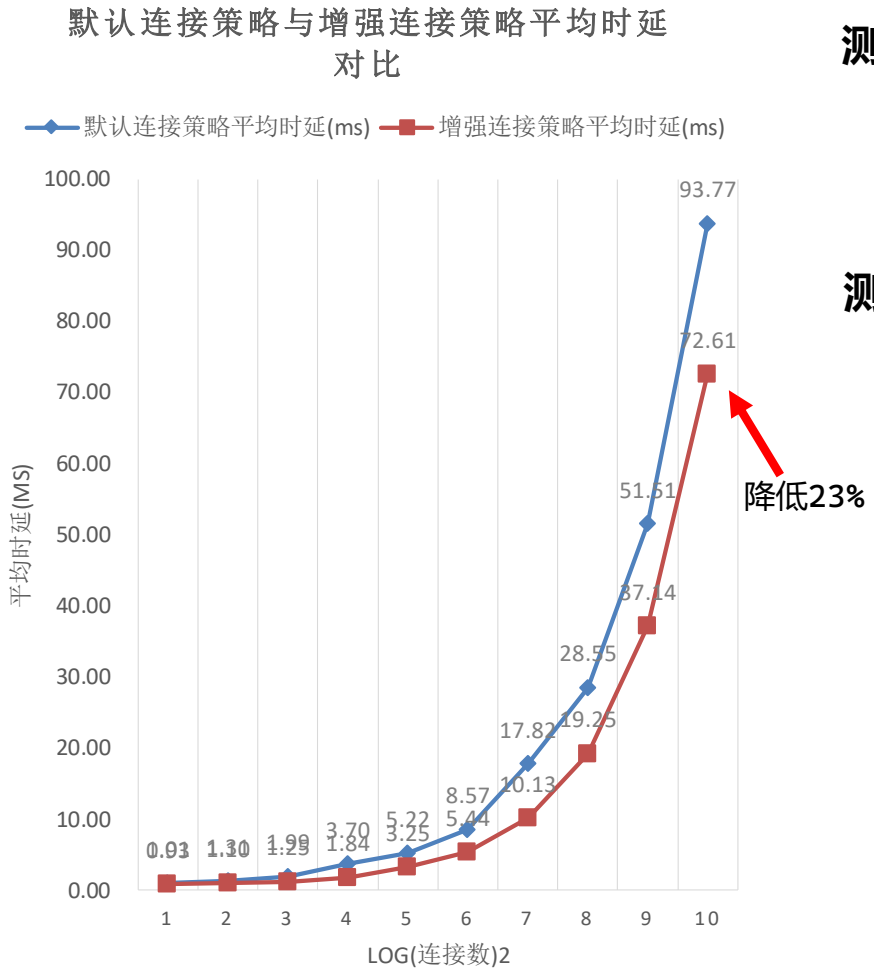
默认连接处理策略



- virtualoutbound监听器监听在相同的监听端口，由内核随机挑选一个线程用于处理新连接。
- 当Envoy配置的线程数比较多时（越多越明显），常观察到新连接被分配到某些线程。
- Envoy的线程模型是工作线程由libevent事件信号驱动串行的处理发生的新事件（网络，定时器）
- 请求堆积在某些线程将导致tp90上升，QPS处理能力下降。
- 对于长连接和规格较高的Envoy影响更明显。短链接可以一定程度上提升连接线程选择的随机性。
- 实际使用中客户更关注tp90的端到端时延，减少由于超时导致的请求失败。

针对Envoy做的一些优化及效果





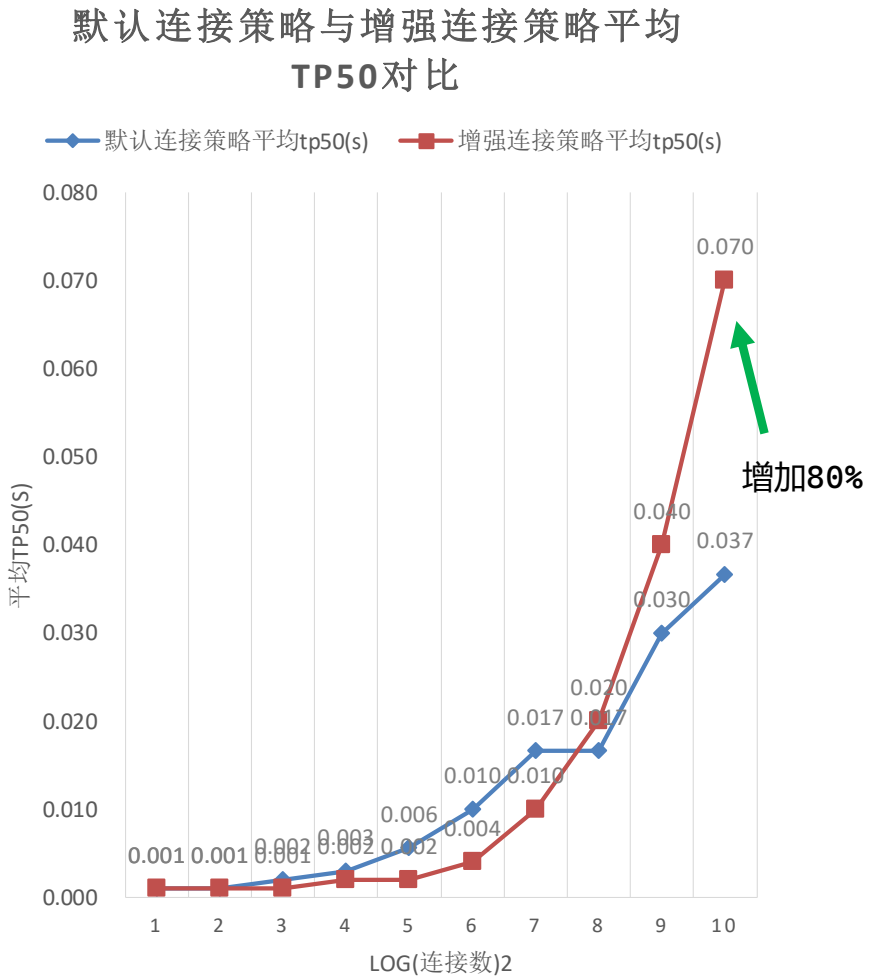
测试条件

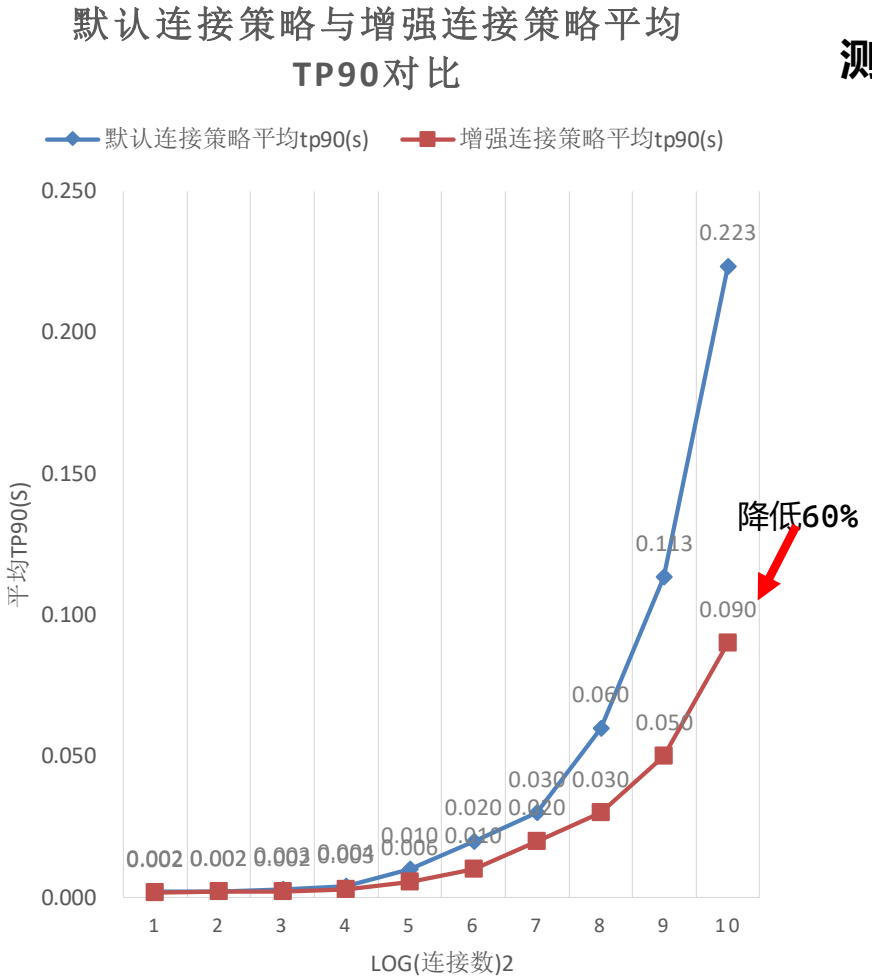
- Envoy: 4线程，4core，默认内存
- fortio –q 0 –c 2~1024连接，http1长连接模式，每组测试三次，每次30s

测试结果

默认连接策略	增强连接策略
QPS变化不均匀，跟内核随机选择的Envoy处理线程有关	QPS变化基本平滑，并可以方便的找到当前线程数下对应斜率最大位置为128连接
	QPS平均提升30%以上
QPS及平均时延在线程数确定时，随连接数增加快速恶化	
	端到端平均时延降低23%左右

针对Envoy做的一些优化及效果





测试结果

默认连接策略	增强连接策略
Tp50（最快的50%请求）增加不大，分析可能被分配到压力较小的线程因此处理较快。	Tp50较默认连接策略增加较大，分析由于平均后不存在压力显著较小的线程，同时qps的增加也会增加端到端tp50时延
	虽然tp50增加，但tp90显著降低60%，并且tp50更接近于tp90的时延，系统整体端到端时延的均衡性更好
Tp50及tp90在线程数确定时，随连接数增加快速恶化	

Envoy问题分析方法

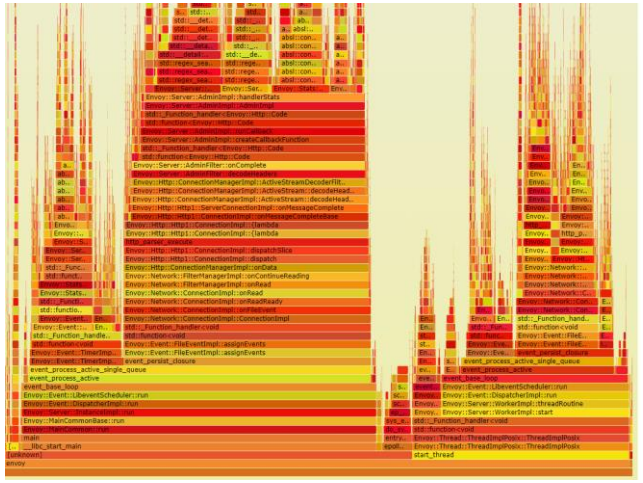
- **查看istio配置**
  - 通过pilot-agent：访问Envoy 15000端口，指定url获取：
    - kubectl exec -it $podname -c istio-proxy -- pilot-agent request GET /config_dump > config.json
  - 查看listener：istioctl pc listener backend-welink-649fdfd55d-2xhzw --port 8123 -o json
  - 查看endpoint：istioctl pc endpoint backend-welink-649fdfd55d-2xhzw
- **运行期日志**

```
[2021-07-07T00:54:08.074Z] [33] "GET / HTTP/1.1" 200 - via_upstream - "-" 0 612 13 6 "-" "curl/7.77.0-DEV" "e9fa5a4c-344e-9970-b235-d58d80f399e8" "nginx" "10.244.92.162:80" outbound|80||nginx.default.svc.cluster.local 10.244.92.173:33592 10.100.102.130:80 10.244.92.173:58270 - default
```

```
"log_format": {
  "text_format": "[%START_TIME%] [%THREAD_ID%] \"%REQ(:METHOD)% %REQ(X-ENVOY-ORIGINAL-PATH?:PATH)% %PROTOCOL%\" %RESPONSE_CODE% %RESPONSE_FLAGS% %RESPONSE_CODE_DETAILS% %CONNECTION_TERMINATION_DETAILS% \"%UPSTREAM_TRANSPORT_FAILURE_REASON%\" %BYTES_RECEIVED% %BYTES_SENT% %DURATION% %RESP(X-ENVOY-UPSTREAM-SERVICE-TIME)% \"%REQ(X-FORWARDED-FOR)%\" \"%REQ(USER-AGENT)%\" \"%REQ(X-REQUEST-ID)%\" \"%REQ(:AUTHORITY)%\" \"%UPSTREAM_HOST%\" %UPSTREAM_CLUSTER% %UPSTREAM_LOCAL_ADDRESS% %DOWNSTREAM_LOCAL_ADDRESS% %DOWNSTREAM_REMOTE_ADDRESS% %REQUESTED_SERVER_NAME% %ROUTE_NAME%\n"
```

  - Accesslog：格式 https://www.envoyproxy.io/docs/envoy/latest/configuration/observability/access_log/usage
  - 调试日志：pilot-agent request POST /logging?connection=trace #Cxxx
- **抓包**
  - 进入pod容器网络空间执行 tcpdump -i any 'port (15001 or 8080)' -w fortio.cap
- **压测工具**
  - fortio load -qps 3000 -c 128 -t 60s --keepalive=false http://backend-welink:8123 #http1
  - nighthawk #http2
  - perf record -F 2000 -g -p $pid; perf script -i perf.data > out.perf; stackcollapse-perf.pl out.perf > out.folded; flamegraph.pl out.folded > cpu.svg
- **镜像修改**
  - 编译pilot-agent, envoy二进制后替换现有envoy镜像并配置到自定义deployment的image中，
  - Dockerfile:
    - From istio/proxyv2:1.9.0

      COPY envoy /usr/local/bin/envoy
      COPY pilot-agent /usr/local/bin/pilot-agent
  - 可以通过自定义deployment内istio注解修改部分启动参数。
    - proxy.istio.io/config: “{concurrency: 6}“ #修改工作线程数，sidecar模式默认2
    - sidecar.istio.io/proxyImage: “istio/proxyv2new:1.9.0 “ #修改默认注入镜像



华为ASM产品介绍（1）

- 华为云CCE容器引擎已深度集成Istio

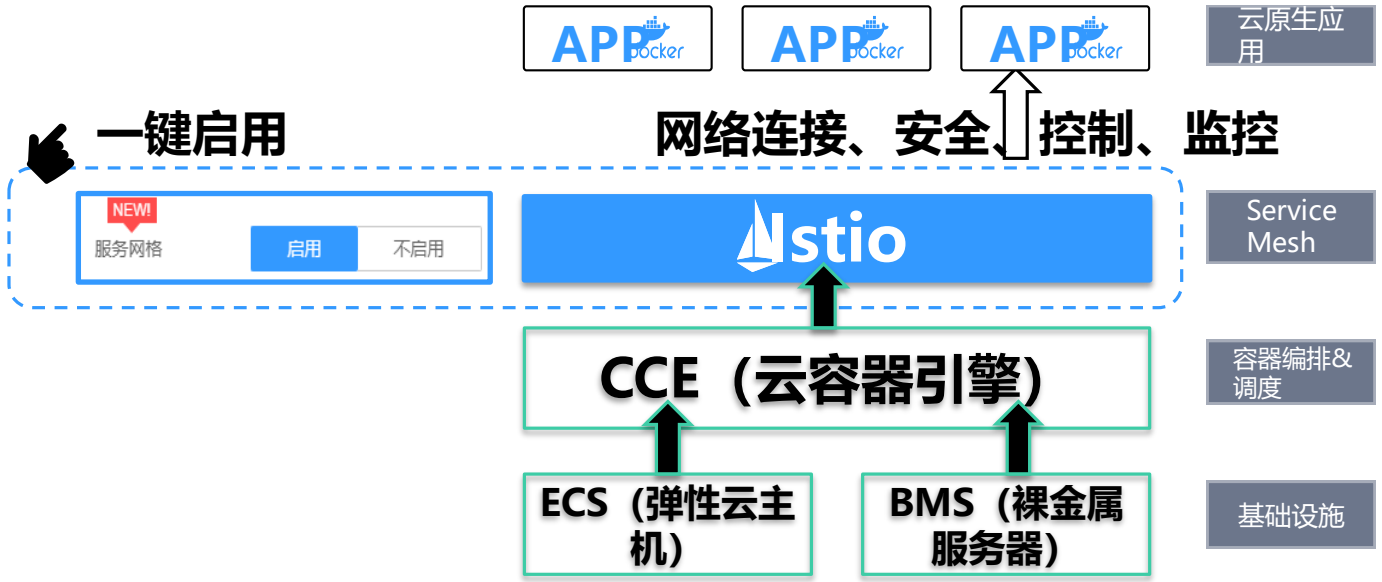
全面治理

智能路由与流量管理
全景应用拓扑，可视化治理

开箱即用

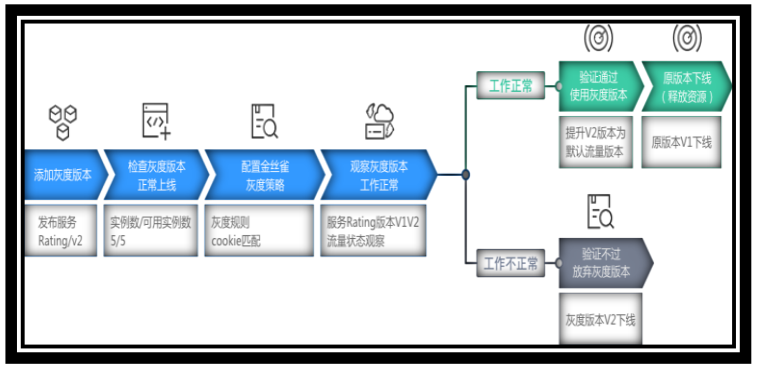
一键启用，与容器引擎无缝整合
内置金丝雀、A/B测试等灰度发布流程

- 无需安装，一键启用，无侵入实现应用治理



华为ASM产品介绍（2）

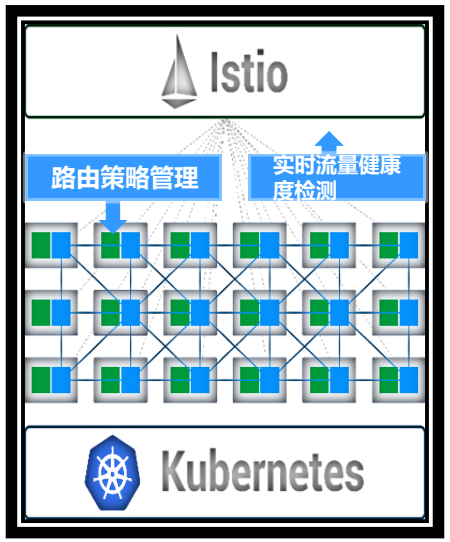
- 内置金丝雀、A/B Testing典型灰度发布流程



灰度发布全流程自动化管理：

- 灰度版本一键部署，流量切换一键生效
- 配置式灰度策略，支持流量比例、请求内容（Cookie、OS、浏览器等）、源IP
- 一站式健康、性能、流量监控，实现灰度发布过程量化、智能化、可视化

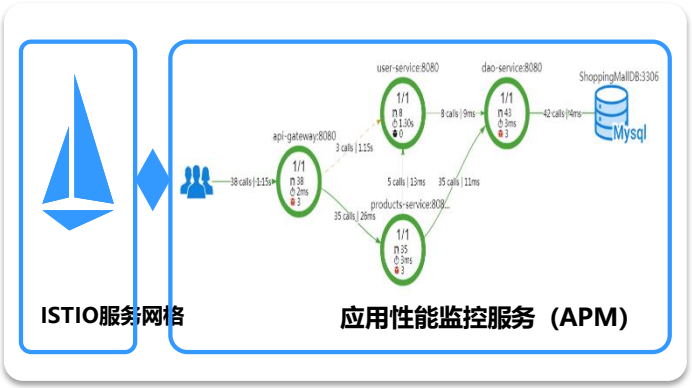
- 策略化的智能路由与弹性流量管理



无侵入智能流量管理：

- 权重、内容等路由规则，实现应用灵活灰度发布
- HTTP会话保持，满足业务处理持续性诉求
- 限流、熔断，实现服务间链路稳定、可靠
- 服务安全认证：认证、鉴权、审计等，提供服务安全保障基石

- 图形化应用全景拓扑，流量治理可视化



实时流量可视化

- 链路健康状态
- 响应时延
- 链路请求数
- 链路异常响应

流量治理可视化

- 路由管理
- 限流、熔断
- 故障注入

Istio数据面发展趋势

- 现状问题：
  - 引入透明Sidecar模式，导致端到端时延增加、并随着集群规模增加导致系统资源消耗上升，对大规模Istio集群的实际使用造成影响。
  - 运维难度增加，透明代理模式无法针对不同业务本身的特点定制监控能力，同时只能从业务容器外面收集应用的运行状态。

- 演进方式：
  - 从Envoy自身I/O及线程模型、容器网络协议栈优化提升端到端性能，降低tp90网络时延。
  - 通过运行时拉取集群依赖服务及POD实例配置信息，同时考虑配置信息共享的方式，降低每Envoy资源消耗。
  - 增加更多运维监控维度及探测点，收集更全面的观测信息。同时支持对出现问题的Envoy进行旁路处理。

本课总结

名称	简介
Envoy	基于C++11,14的高性能服务网格数据面代理
xDS	Envoy与上层控制面如istiod使用的基于gRPC的应用层协议，用于传输配置变更。
自动注入及流量拦截	POD创建时，由istiod进行自动修改deployment并将istio-init, istio-proxy容器注入到新创建POD内；当发生调用时，iptables规则将自动拦截出入流量进入Envoy代理。
线程模型	Envoy采用每个工作线程独立处理网络及定时器事件，线程间无数据共享，提升性能。
过滤器架构	Envoy采用可扩展插件架构实现监听过滤器、L4网络过滤器、L7 HTTP过滤器；同时支持基于L4/L7 WASM及L7 Lua过滤器的二次扩展。

参考链接

相关内容的华为云官网链接： https://support.huaweicloud.com/usermanual-cce/cce_01_0006.html

详细ASM官网资料：https://support.huaweicloud.com/istio/

Istio官方文档： https://istio.io/latest/docs/

envoy官方文档：https://www.envoyproxy.io/docs/envoy/latest/

HUAWEI

华为云学院

学以致用　云世界大有可为

感谢