



微服务结合容器云平台的思考和实践

2018.06.25 徐运元

2008年毕业于浙江大学，曾在思科和浙大网新有超过9年的工作经验和5年的云计算领域工作经验，带领团队完成公司第一代基于Kubernetes的云平台开发和第二代基于Kubernetes的DevOps云平台开发

来自于浙江大学SEL实验室



目录

CONTENTS



Kubernetes平台下的微服务演进



Pilot核心功能解读



Pilot-Agent核心流程解读

Kubernetes平台下的微服务演进

当我们在讨论微服务的时候我们在讨论什么？

• 解决如何微服务的问题

- 微服务拆分原则
- 业务API设计
- 数据一致性保证
- 可扩展性考虑
- ...

• 解决微服务化后带来的问题

温饱问题

- 计算资源的快速分配
- 基本的监控
- 快速部署
- 易于分配的存储
- 易于访问的外围（负载均衡）
- 服务注册和发现

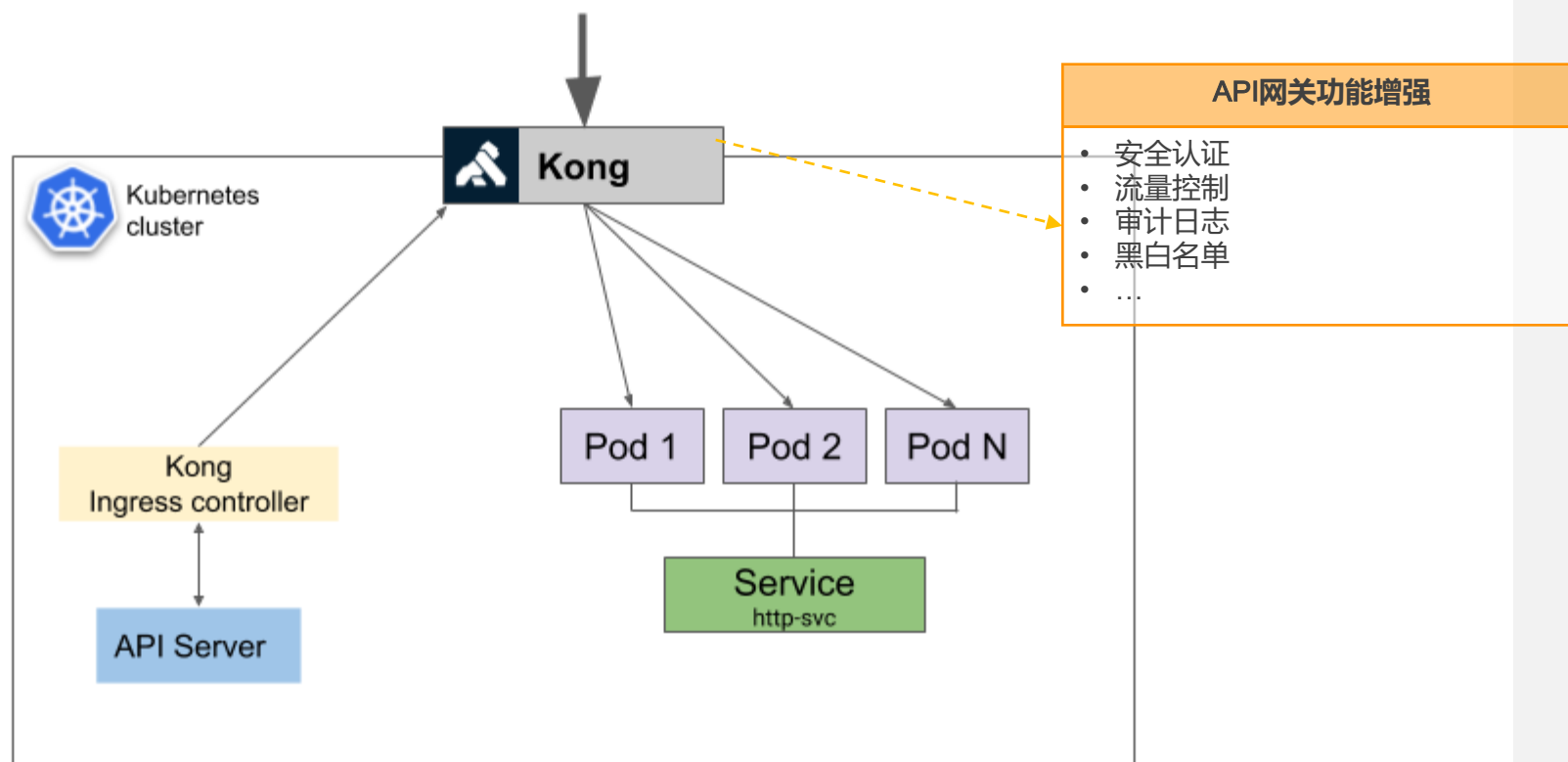
致富问题

- 认证和授权
- 智能路由
- 流量管理
- 服务降级
- ...

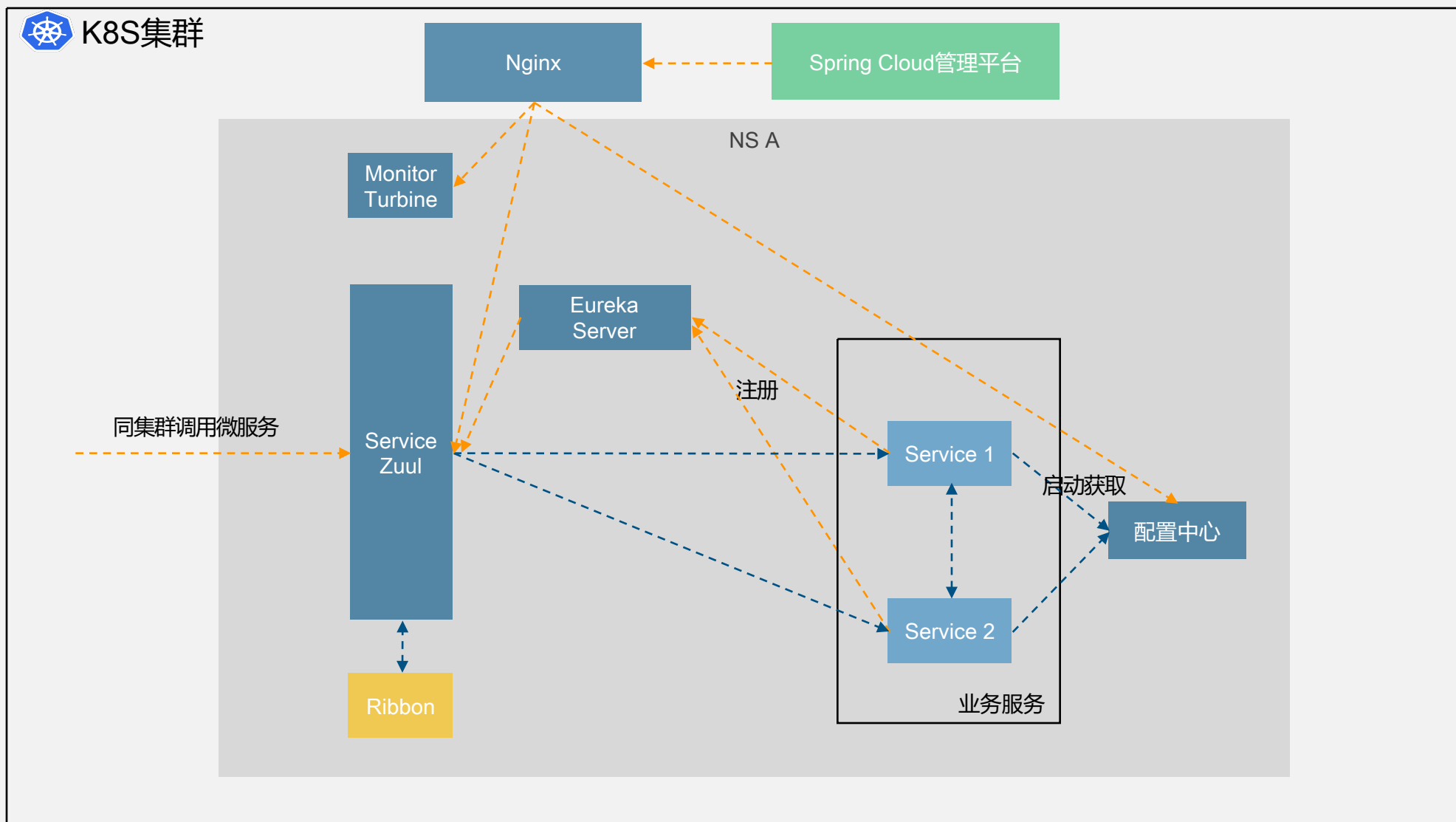
Kubernetes对于微服务的支撑

功能列表	详情
快速资源分配	容器编排和调度
服务部署&弹性伸缩	Deployment
服务注册&服务发现	Service概念和分布式DNS
API网关	简单路由功能
统一日志中心	Fluentd & ES
统一监控中心	Prometheus
统一配置管理	Configmap、Secret
负载均衡	简单负载均衡，基于Iptables Roundrobin
流量控制	简单根据服务实例进行控制

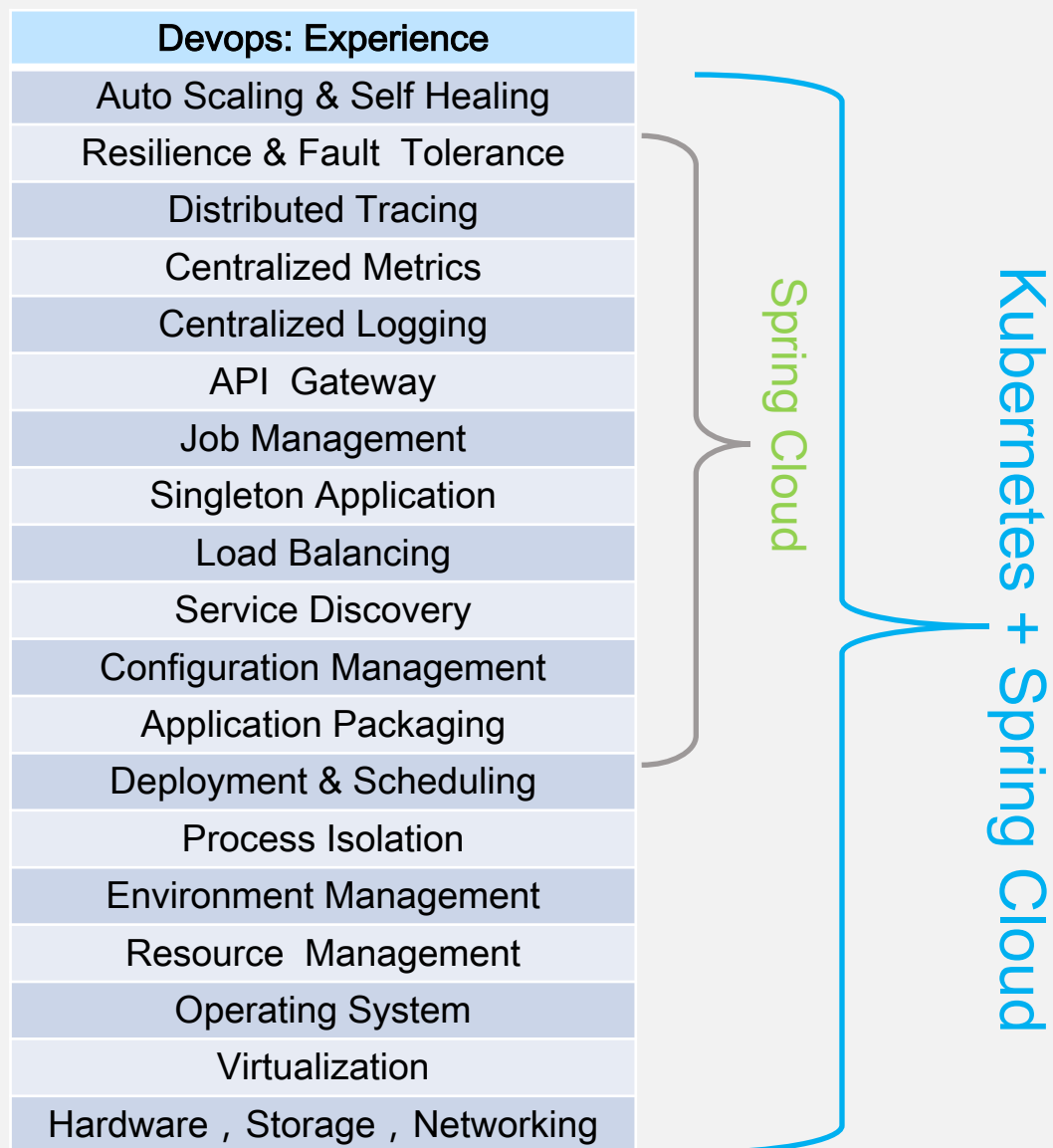
云平台微服务演进之基于API网关的微服务方案



云平台微服务演进之基于Spring Cloud的微服务方案

SC流量 

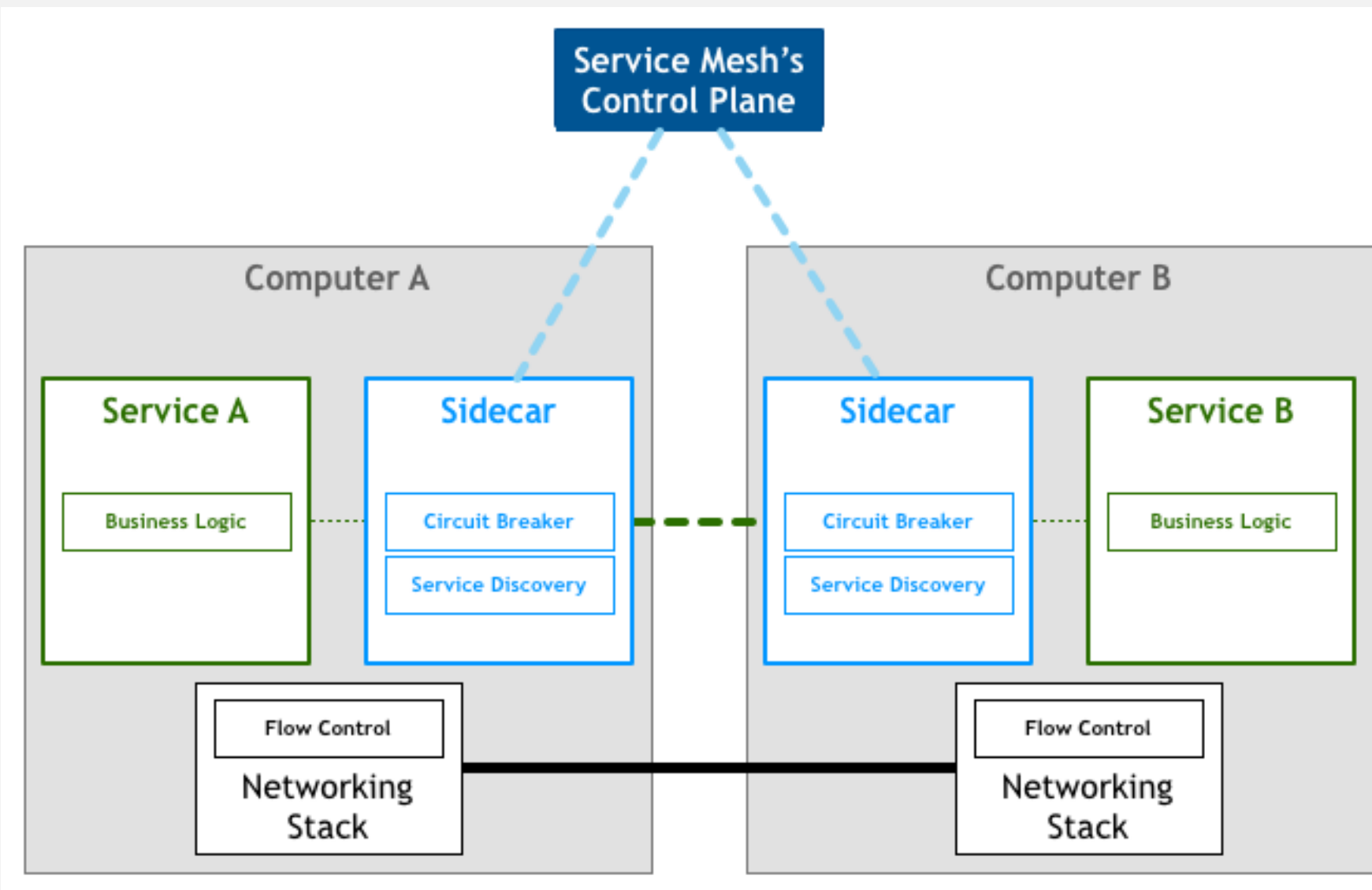
基于Kubernetes的Spring Cloud实现



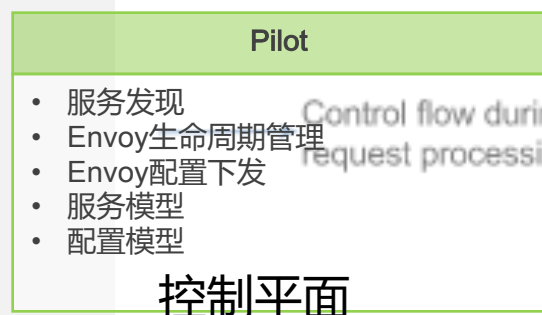
看起来很美背后

- 功能上的重叠
- 服务降级
- 细粒度的鉴权（服务间的调用）
- RPC支持
- 跨语言的问题
- ...

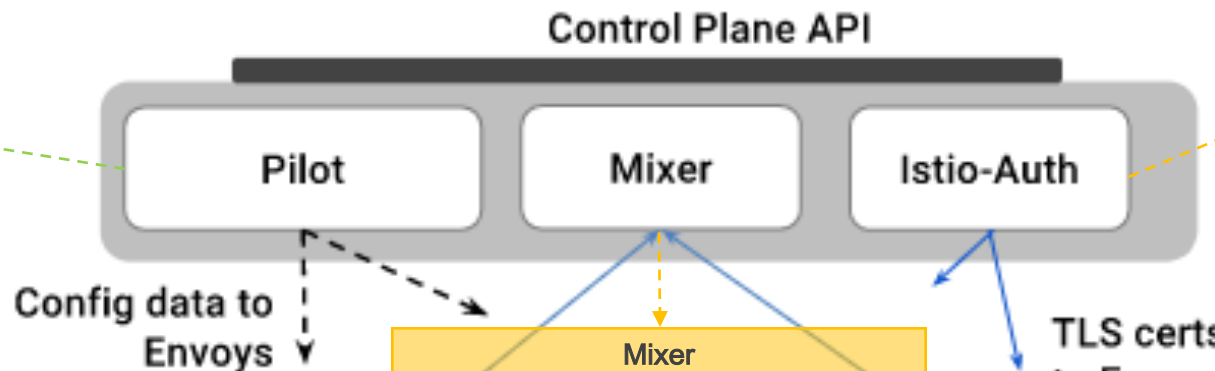
云平台微服务演进之Service Mesh



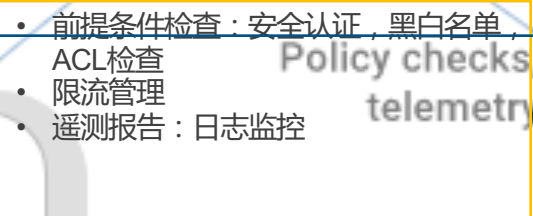
云平台微服务演进之Service Mesh



Control flow during request processing



TLS certs to Envoy

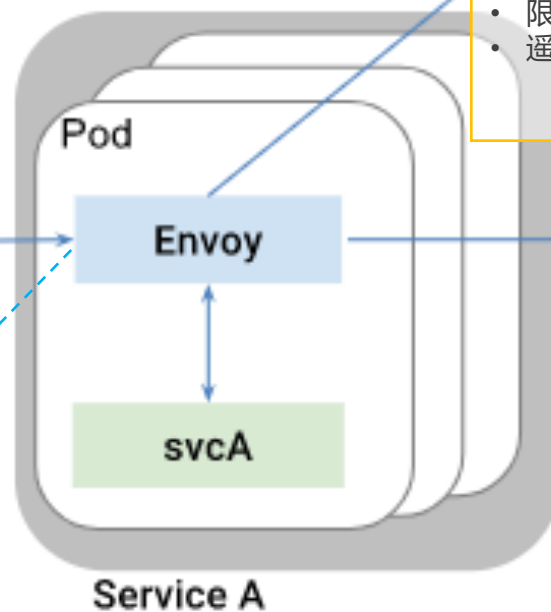


数据平面

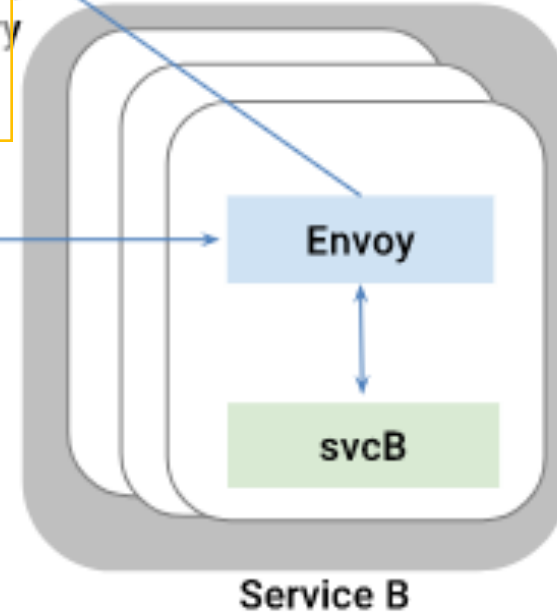
HTTP/1.1, HTTP/2, gRPC, TCP with or without TLS

Envoy without TLS

- 负载均衡
- 智能路由（灰度、蓝绿）
- 流量管理（超时、重试、熔断）
- 故障处理
- 故障注入
- ...



HTTP/1.1, HTTP/2, gRPC, TCP with or without TLS



Istio的核心组件

- Envoy

是一个高性能轻量级代理，它掌控了service的入口流量和出口流量，它提供了很多内置功能，如动态负载均衡服务发现、负载均衡、TLS终止、HTTP/2 & gRPC流量代理、熔断、健康检查等功能。

- Mixer

翻译过来是混音器，Mixer负责在整个Service Mesh中实施访问控制和使用策略。Mixer是一个可扩展组件，内部提供了多个模块化的适配器([adapter](#))。

- Pilot

翻译过来是领航员，Pilot对Envoy的生命周期进行管理，同时提供了智能路由（如A/B测试、金丝雀部署）、流量管理（超时、重试、熔断）功能。Pilot接收用户指定的高级路由规则配置，转换成Envoy的配置，使这些规则生效。

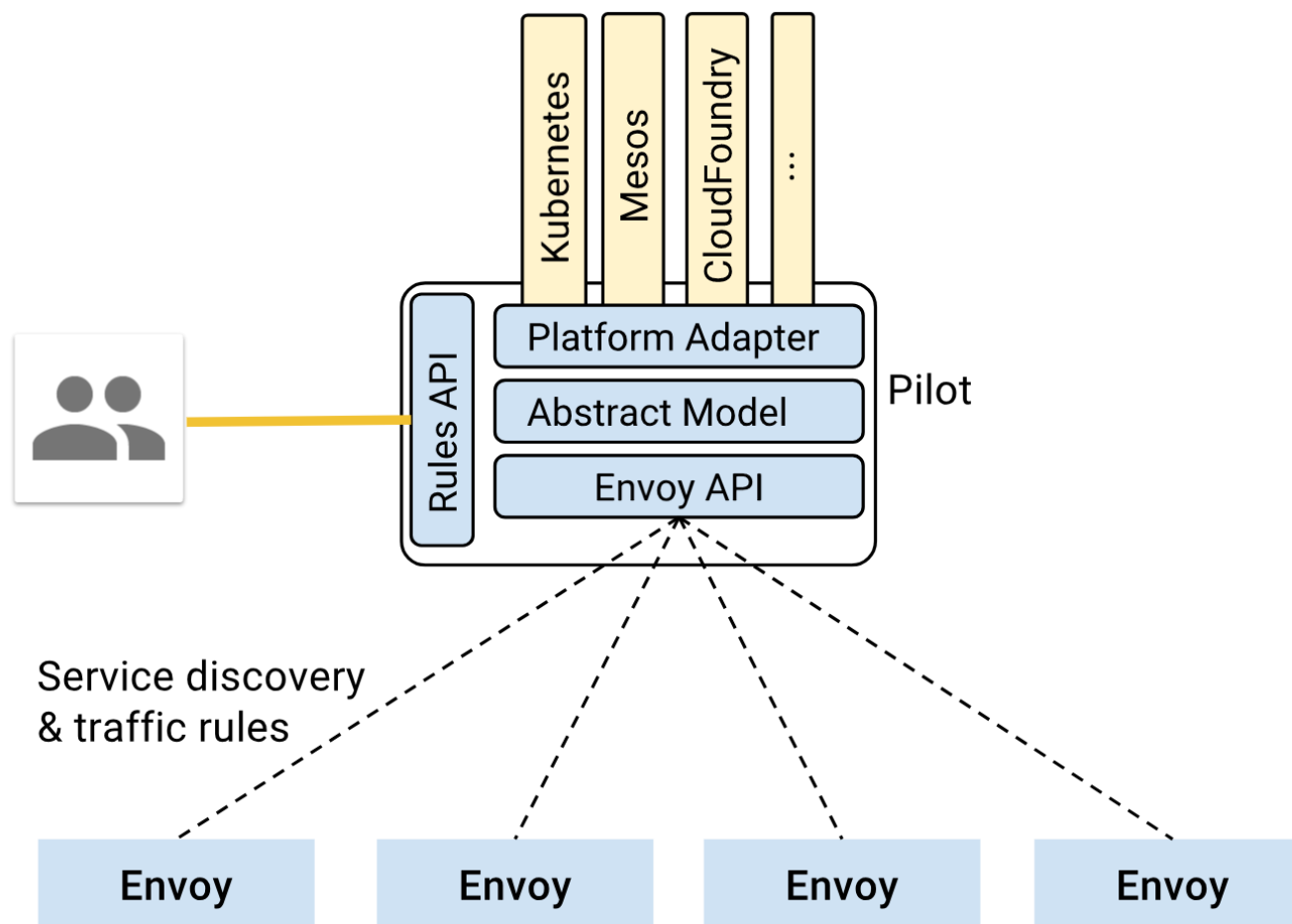
- Istio-Auth

服务间认证和终端用户认证功能

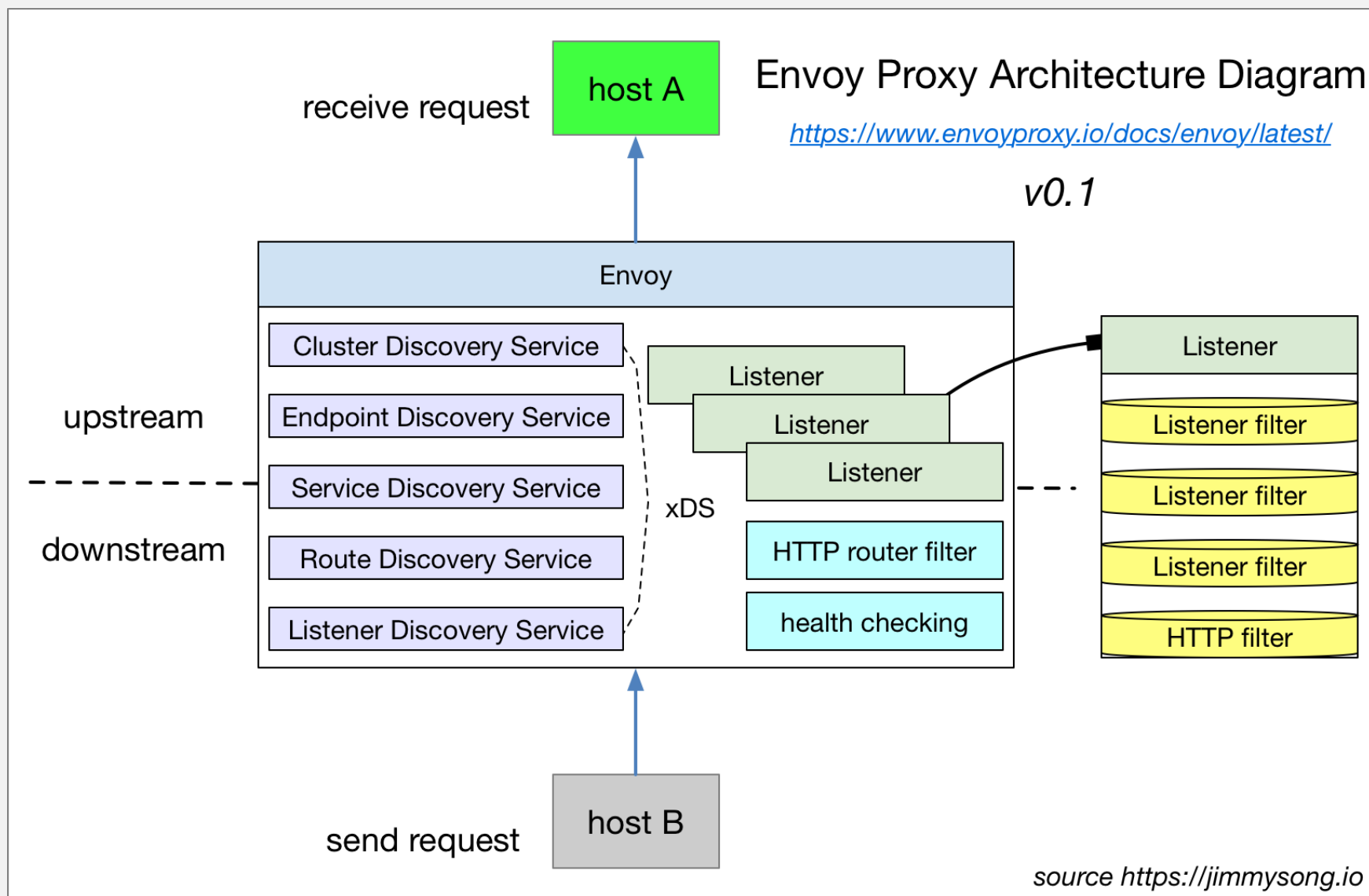
Istio的Pilot功能解析



Pilot官方架构



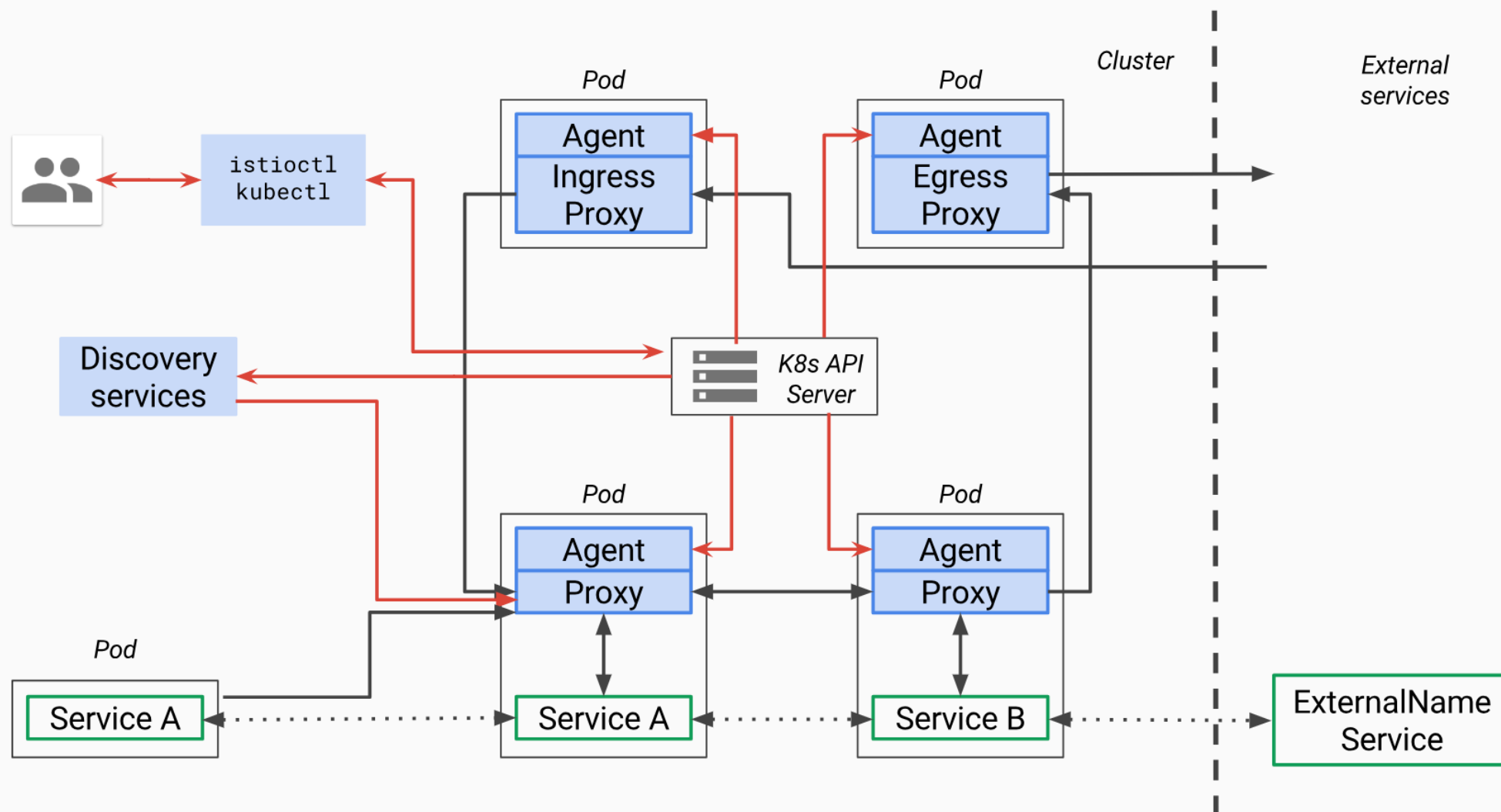
Pilot Architecture



Pilot-Agent的核心流程解读



Pilot工作流程



Pilot-Agent的部署形式

```
▼ folder pilot ~/go/src/istio.io/istio/pilot
  ▼ folder cmd
    ▼ folder pilot-agent
      🐙 main.go
      🐙 request.go
      ▶ folder pilot-discovery
      ▶ folder sidecar-injector
        🐙 cmd.go
      ▶ folder docker
      ▶ folder pkg
      ▶ folder test
      ▶ folder tools
      📄 codecov.requirement
      📄 OWNERS
  ▶ External Libraries
  📄 Scratches and Consoles
```

pilot-agent在pilot/cmd包下面，是个单独的二进制。

pilot-agent跟envoy打包在同一个docker镜像里，镜像由Dockerfile.proxy定义，Makefile（include了tools/istio-docker.mk）把这个dockerfile build成了\${HUB}/proxy:\${TAG}镜像，也就是Kubernetes里跟应用放在同一个pod下的sidecar。非Kubernetes情况下需要把pilot-agent、envoy跟应用部署在一起，这就有点“污染”应用的意思了。

Pilot-Agent的功能介绍

在proxy镜像中，pilot-agent负责的工作包括：

- 生成envoy的配置。
- 启动envoy。
- 监控并管理envoy的运行状况，比如envoy出错时pilot-agent负责重启envoy，或者envoy配置变更后 reload envoy。

而envoy负责接受所有发往该pod的网络流量，分发所有从pod中发出的网络流量。

根据代码中的sidecar-injector-configmap.yaml（用来配置如何自动化地inject istio sidecar），inject过程中，除了proxy镜像作为sidecar之外，每个pod还会带上initcontainer（Kubernetes中的概念），具体镜像为proxy_init。proxy_init通过注入iptables规则改写流入流出pod的网络流量规则，使得流入流出pod的网络流量重定向到proxy的监听端口，而应用对此无感。

Init Container的工作原理

istio/tools/deb/istio-iptables.sh

#将所有流量送入Envoy

```
iptables -t nat -N ISTIO_REDIRECT
```

```
iptables -t nat -A ISTIO_REDIRECT -p tcp -j REDIRECT --to-port ${PROXY_PORT}
```

#所有进入容器的流量送入Envoy

```
iptables -t ${table} -A PREROUTING -p tcp -j ISTIO_INBOUND
```

```
iptables -t nat -A ISTIO_INBOUND -p tcp -j ISTIO_REDIRECT
```

#所有出去的流量送入Envoy

```
iptables -t nat -A OUTPUT -p tcp -j ISTIO_OUTPUT
```

```
iptables -t nat -A ISTIO_OUTPUT -o lo ! -d 127.0.0.1/32 -j ISTIO_REDIRECT
```

#防止回环

```
iptables -t nat -A ISTIO_OUTPUT -m owner --uid-owner ${uid} -j RETURN
```

Pilot-Agent主要功能分析-生产Envoy配置

envoy的配置主要在pilot-agent的init方法与proxy命令处理流程的前半部分生成。其中init方法为pilot-agent二进制的命令行配置大量的flag与flag默认值，而proxy命令处理流程的前半部分负责将这些flag组装成为envoy的配置ProxyConfig对象。下面分析几个相对重要的配置。

role

pilot-agent的role类型为model包下的Proxy，决定了pilot-agent的“角色”，role包括以下属性：

1. Type

pilot-agent有三种运行模式。根据role.Type变量定义，类型为model.Proxy，定义在context.go文件中，允许的3个取值范围为：

i. "sidecar"

默认值，可以在启动pilot-agent，调用proxy命令时覆盖。Sidecar type is used for sidecar proxies in the application containers.

ii. "ingress"

Ingress type is used for cluster ingress proxies.

iii. "router"

Router type is used for standalone proxies acting as L7/L4 routers.

2. IPAddress, ID, Domain

它们都可以通过pilot-agent的proxy命令的对应flag来提供用户自定义值。如果用户不提供，则会在proxy命令执行时，根据istio连接的服务注册中心（service registry）类型的不同，会采用不同的配置方式。agent当前使用的具体service registry类型保存在pilot-agent的registry变量里，在init函数中初始化为默认值Kubernetes。当前只处理以下三种情况：

- Kubernetes
- Consul
- Other

Pilot-Agent主要功能分析-生产Envoy配置

registry值	role. IPAddress	role. ID	role. Domain
Kubernetes	环境变量INSTANCE_IP	环境变量POD_NAME. 环境变量POD_NAMESPACE	环境变量POD_NAMESPACE. svc. cluster. local
Consul	private IP, 默认127. 0. 0. 1	IPAddress. service. consul	service. consul
Other	private IP, 默认127. 0. 0. 1	IPAddress	“ ”

其中的private ip通过WaitForPrivateNetwork函数获得。

istio需要从服务注册中心（service registry）获取微服务注册的情况。当前版本中istio可以对接的服务注册中心类型包括：

1. "Mock"

MockRegistry is a service registry that contains 2 hard-coded test services.

2. "Config"

ConfigRegistry is a service registry that listens for service entries in a backing ConfigStore.

3. "Kubernetes"

KubernetesRegistry is a service registry backed by K8s API server.

4. "Consul"

ConsulRegistry is a service registry backed by Consul.

5. "Eureka"

EurekaRegistry is a service registry backed by Eureka.

6. "CloudFoundry"

CloudFoundryRegistry is a service registry backed by Cloud Foundry.

Pilot-Agent主要功能分析-生产Envoy配置

agent.waitForExit会调用envoy.Run方法启动envoy进程，为此需要获取envoy二进制所在文件系统路径和命令行参数两部分信息：

1. envoy二进制所在文件系统路径：envoy.Run通过proxy.config.BinaryPath变量得知envoy二进制所在的文件系统位置，proxy就是envoy对象，config就是pilot-agent的main方法在一开始初始化的proxyConfig对象。里面的BinaryPath在pilot-agent的init方法中被初始化，初始值来自pilot/pkg/model/context.go的DefaultProxyConfig函数，值是/usr/local/bin/envoy。
2. envoy的启动参数形式为下面的startupArgs，包含一个-c指定的配置文件，还有一些命令行参数。除了下面代码片段中展示的这些参数，还可以根据agent启动参数，再加上--concurrency，--service-zone等参数。

而上面的-c指定的envoy配置文件有几种生成的方式：

1. 运行pilot-agent时，用户不指定customConfigFile参数（agent init时默认为空），但是制定了templateFile参数（agent init时默认为空），这时agent的main方法会根据templateFile帮用户生成一个customConfigFile，后面就视作用户制定了customConfigFile。这个流程在agent的main方法里。
2. 如果用户制定了customConfigFile，那么就用customConfigFile。
3. 如果用户customConfigFile和templateFile都没指定，则调用pilot/pkg包下的bootstrap_config.go中的WriteBootstrap自动生成一个配置文件，默认将生成的配置文件放在/etc/istio/proxy/envoy-rev%d.json，这里的%d会用epoch序列号代替。WriteBootstrap在envoy.Run方法中被调用。

Pilot-Agent主要功能分析-Envoy监控与管理

为envoy生成好配置文件之后，pilot-agent还要负责envoy进程的监控与管理的工作，包括：

1. 创建envoy对象，结构体包含proxyConfig（前面步骤中为envoy生成的配置信息），role.serviceNode（似乎是agent唯一标识符），loglevel和pilotsan（service account name）。
2. 创建agent对象，包含前面创建的envoy结构体，一个epochs的map，3个channel：configCh，statusCh和abortCh。
3. 创建watcher并启动协程执行watcher.Run。watcher.Run首先启动协程执行agent.Run（agent的主循环），然后调用watcher.Reload(kickstart the proxy with partial state (in case there are no notifications coming))，Reload会调用agent.ScheduleConfigUpdate，并最终导致第一个envoy进程启动，见后面分析。然后监控各种证书，如果证书文件发生变化，则调用ScheduleConfigUpdate来reload envoy，然后watcher.retrieveAZ(TODO)。
4. 调用cmd.WaitSignal，等待进程接收到SIGINT，SIGTERM信号，接受到信号之后会kill所有envoy进程，并退出agent进程。

Pilot-Agent主要功能分析-Envoy启动流程

1. 前面pilot-agent proxy命令处理流程中，watcher.Run会调用agent.ScheduleConfigUpdate，这个方法只是简单地往configCh里写一个新的配置，所谓的配置是所有certificate算出的sha256哈希值。
2. configCh的这个事件会被agent.Run监控到，然后调用agent.reconcile。
3. reconcile方法会启动协程执行agent.waitForExit从而启动envoy看reconcile方法名就知道是用来保证desired config和current config保持一致的。reconcile首先会检查desired config和current config是否一致，如果是的话，就不用启动新的envoy进程。否则就启动新的envoy。在启动过程中，agent维护两个map来管理一堆envoy进程，在调用waitForExit之前会将desiredConfig赋值给currentConfig，表示reconcile工作完成
4. waitForExit会调用agent.proxy.Run，也就是envoy的Run方法，这里会启动envoy。

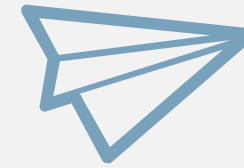
每次配置发生变化，都会调用agent.reconcile，也就会启动新的envoy，这样envoy越来越多，老的envoy进程怎么办？agent代码的注释里已经解释了这问题，原来agent不用关闭老的envoy，同一台机器上的多个envoy进程会通过unix domain socket互相通讯，即使不同envoy进程运行在不同容器里，也一样能够通讯。而借助这种通讯机制，可以实现新envoy进程替换之前的老进程，也就是所谓的envoy hot restart。

对于Istio和云平台集成的一些思考

- 可视化的统一管理平台
- 多租户的资源隔离
- Mixer的性能问题

参考资料

- Service Mesh深度学习系列|istio源码分析之pilot-agent组件分析
- Patten: Service Mesh
- Envoy基本架构和配置解析



感谢聆听

Thanks !