

## 《Istio大咖说》第3期

# 如何让 Istio 变得更为高效和智能



主持人：宋净超（Tetrate）



嘉宾：杨笛航（网易数帆）



6月9日

晚8:00 – 9:00

联合主办方：



扫码观看直播

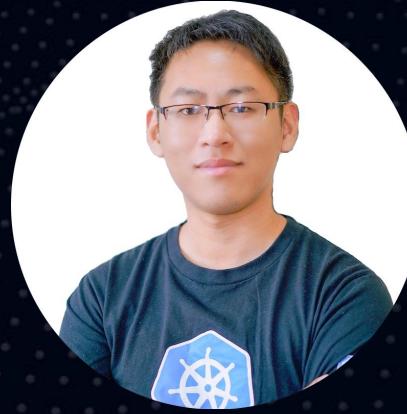


# tetrate



THE ENTERPRISE SERVICE MESH COMPANY

# Istio 大咖说第 3 期



宋净超 ( Jimmy Song )

Tetrate 布道师、云原生社区创始人  
<https://jimmysong.io>

主持人



杨笛航

Istio 社区成员、网易数帆架构师

嘉宾

# Agenda



## 1. 背景

Istio的优势与不足

## 2. 智能网格管理器

Slime架构介绍

## 3. Slime的解决方案

Slime功能模块介绍

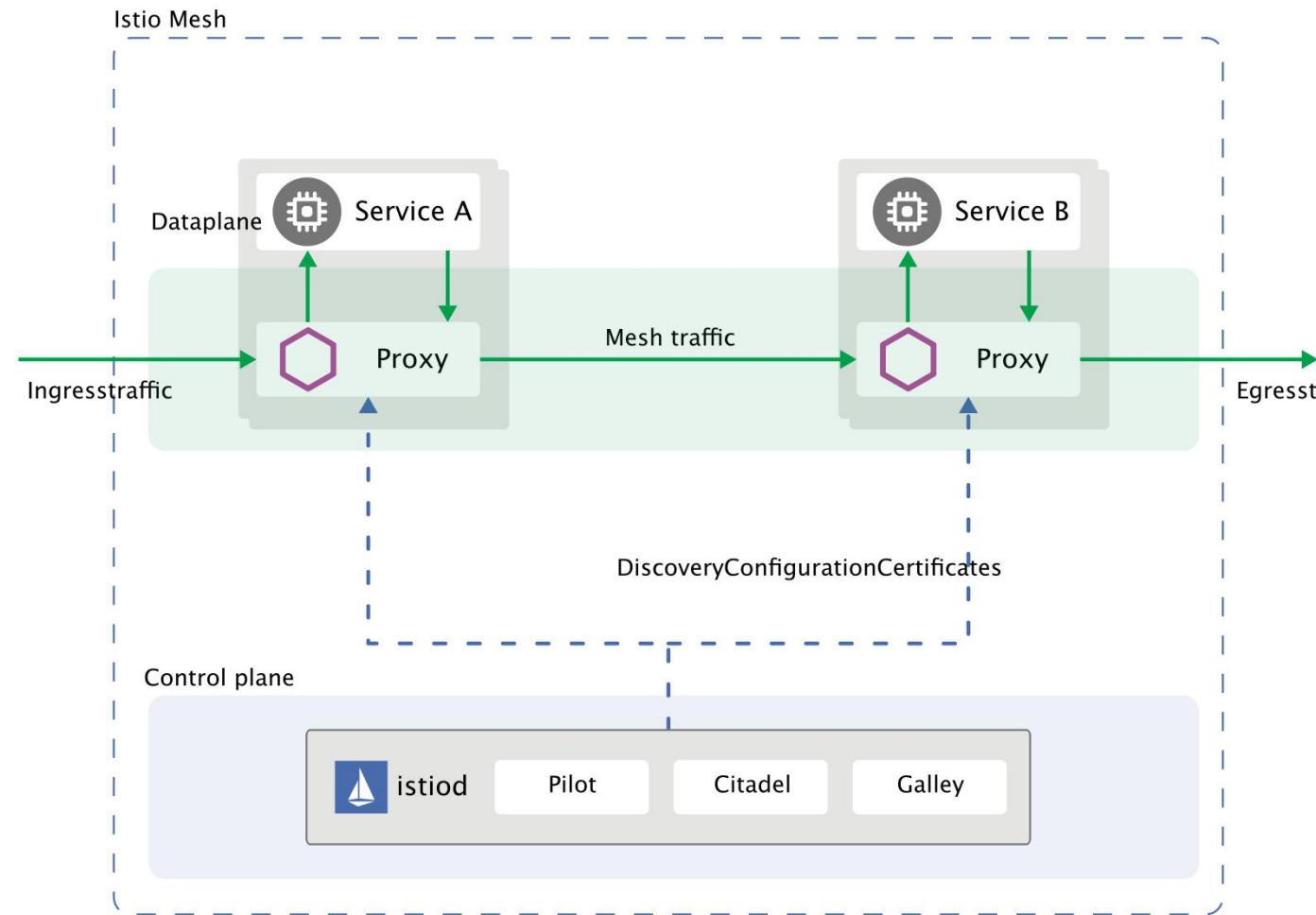
## 4. 高效流量管理之路

Slime的后续规划



# 背景

# 背景 – Istio简介



服务治理逻辑下沉到proxy容器中，实现了业务逻辑和微服务治理逻辑的物理解耦，降低微服务框架的开发与运维成本。Istio是Service Mesh架构使用最为广泛的实现，其接口（CRD）已隐隐具备了成为业界标准的趋势。

# 背景 – 生产过程中遇到的问题

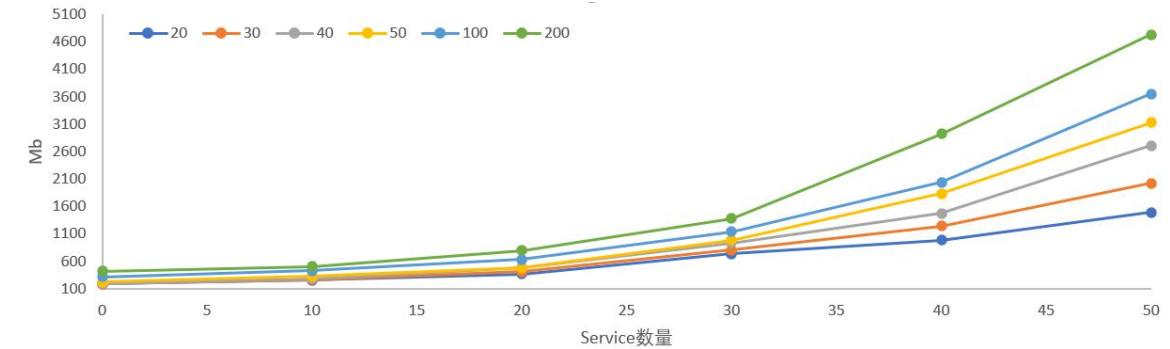
## 配置全量下发导致的性能问题

- Envoy收到大量冗余配置

- a) Envoy启动时间变长
- b) Envoy内存开销增加
- c) 占据Envoy主线程，阻塞pilot事件推送

- 增加Pilot侧处理推送事件复杂度

- a) pilot推送时内存增加，易引发OOM
- b) 配置下发时延增加



存量配置对pilot动态内存的影响

### 50xds, 10svc

Goroutine Total		Execution	Network wait	Sync block	Blocking syscall	Scheduler wait	GC sweeping	GC pause
<a href="#">208571</a>	30s		364ms	0ns	23s	0ns	409ms	83ms (0.3%) 840ms (2.8%)
<a href="#">208582</a>	30s		250ms	0ns	22s	0ns	353ms	77ms (0.3%) 840ms (2.8%)
<a href="#">208204</a>	30s		290ms	0ns	22s	0ns	346ms	127ms (0.4%) 840ms (2.8%)
<a href="#">207802</a>	30s		345ms	0ns	22s	0ns	346ms	57ms (0.2%) 840ms (2.8%)
<a href="#">207677</a>	30s		296ms	0ns	23s	0ns	339ms	44ms (0.1%) 840ms (2.8%)

### 50xds, 50svc

Goroutine Total		Execution	Network wait	Sync block	Blocking syscall	Scheduler wait	GC sweeping	GC pause
<a href="#">208237</a>	30s		3885ms	0ns	868ms	0ns	16s	1280ms (4.3%) 11s (38.6%)
<a href="#">207877</a>	30s		3894ms	0ns	399ms	0ns	16s	1275ms (4.2%) 11s (38.6%)
<a href="#">208041</a>	30s		3788ms	0ns	906ms	0ns	17s	1157ms (3.9%) 11s (38.6%)
<a href="#">207677</a>	30s		3696ms	0ns	4615ms	0ns	17s	1128ms (3.8%) 11s (38.6%)
<a href="#">207827</a>	30s		3909ms	0ns	663ms	52μs	18s	1182ms (3.9%) 11s (38.6%)

## 社区的方案：

需要用户根据服务依赖关系，手动配置SidecarScope，控制面根据SidecarScope为服务下发其所关心的服务发现及配置信息。

# 背景 – 生产过程中遇到的问题

## 扩展性问题

Mixer集中式的架构以及随之带来的性能问题，导致Istio在新版本中将其弃用。而新版本带来的MixerV2，虽然解决了数据面功能扩展的问题，但是仍然需借助EnvoyFilter才能下发插件配置，EnvoyFilter作为通用接口，其易用性和可维护性不是十分理想。

```
apiVersion: config.istio.io/v1alpha2
kind: QuotaSpec
metadata:
  name: request-count
  namespace: istio-system
spec:
  rules:
    - quotas:
        - charge: 1
        quota: requestcountquota
```

**Mixer限流配置 | EnvoyFilter限流配置**



```
apiVersion: networking.istio.io/v1alpha3
kind: EnvoyFilter
metadata:
  name: filter-local-ratelimit-svc
  namespace: istio-system
spec:
  workloadSelector:
    labels:
      app: clt
  configPatches:
    - applyTo: HTTP_FILTER
      listener:
        filterChain:
          filter:
            name: "envoy.http_connection_manager"
  patch:
    operation: INSERT_BEFORE
    value:
      name: envoy.filters.http.local_ratelimit
      typed_config:
        "@type": type.googleapis.com/udpa.type.v1.TypedStruct
        type_url: type.googleapis.com/envoy.extensions.filters.http.local_ratelimit.v3.LocalRateLimit
        value:
          stat_prefix: http_local_rate_limiter
          token_bucket:
            max_tokens: 1
            tokens_per_fill: 1
            fill_interval: 600s
          filter_enabled:
            runtime_key: local_rate_limit_enabled
            default_value:
              numerator: 100
              denominator: HUNDRED
          filter_enforced:
            runtime_key: local_rate_limit_enforced
            default_value:
              numerator: 100
```

## 背景 – 生产过程中遇到的问题

配置全量下发  
↓  
手动配**SidecarScope**

扩展功能  
↓  
运维**EnvoyFilter**

更智能的方法?

更高效的方法?



智能网格管理器

# Agenda



1.背景

2.智能网格管理器

3.Slime的解决方案

4.高效流量管理之路



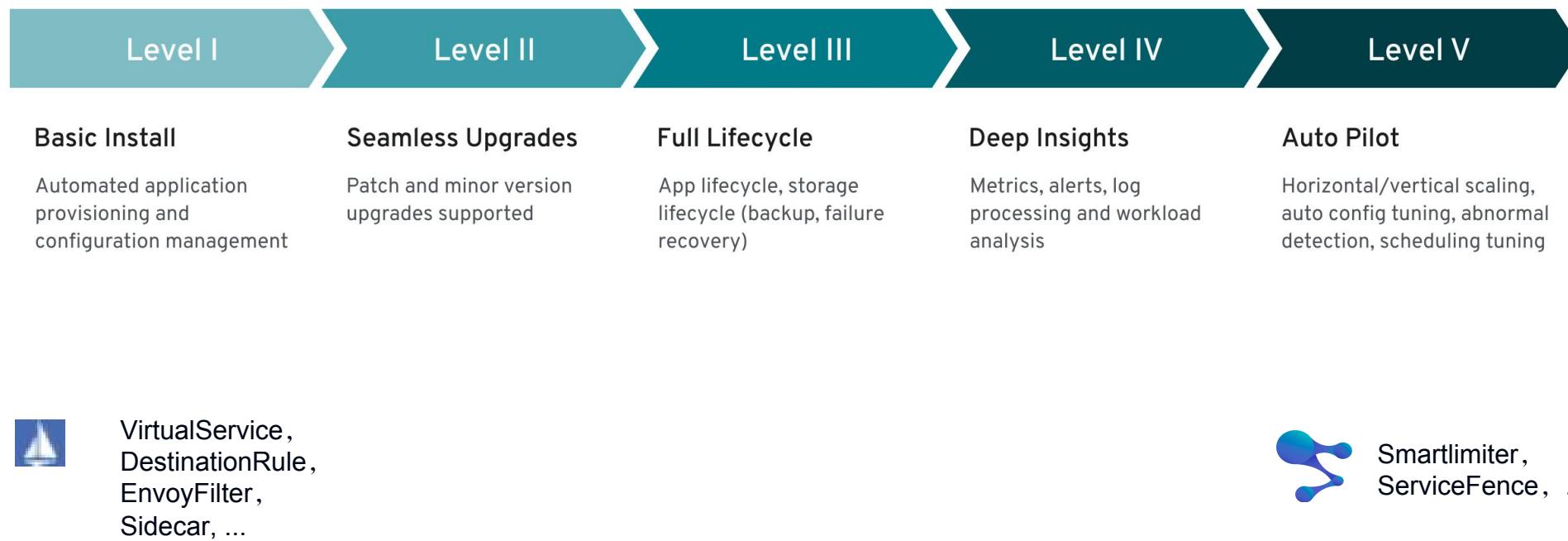
# 智能网格管理器

# 智能网格管理器

为了解决冗余配置的问题，用户需要手动配置SidecarScope，有什么方法能自动配置SidecarScope么？为了解决扩展问题，用户需要运维EnvoyFilter，有什么方法能避免运维EnvoyFilter么？

其实答案已经呼之欲出了，要解决上述问题本质上需要在Istio之上构建一层智能管理器，智能管理器一方面的作用是可以将一些复杂的IstioCR转换为便于管理的新接口，使运维管理更为高效，另一方面在结合监控信息之后，可以自动调整IstioCR，使流量管理更为智能。我们可以类比服务运维将对IstioCR的运维也分为3个阶段。如果把istio的CRD看作BasicConfig，智能管理器就是针对IstioCR的AutoPilot。Slime组件的提出，正是对这一角色的一次尝试。

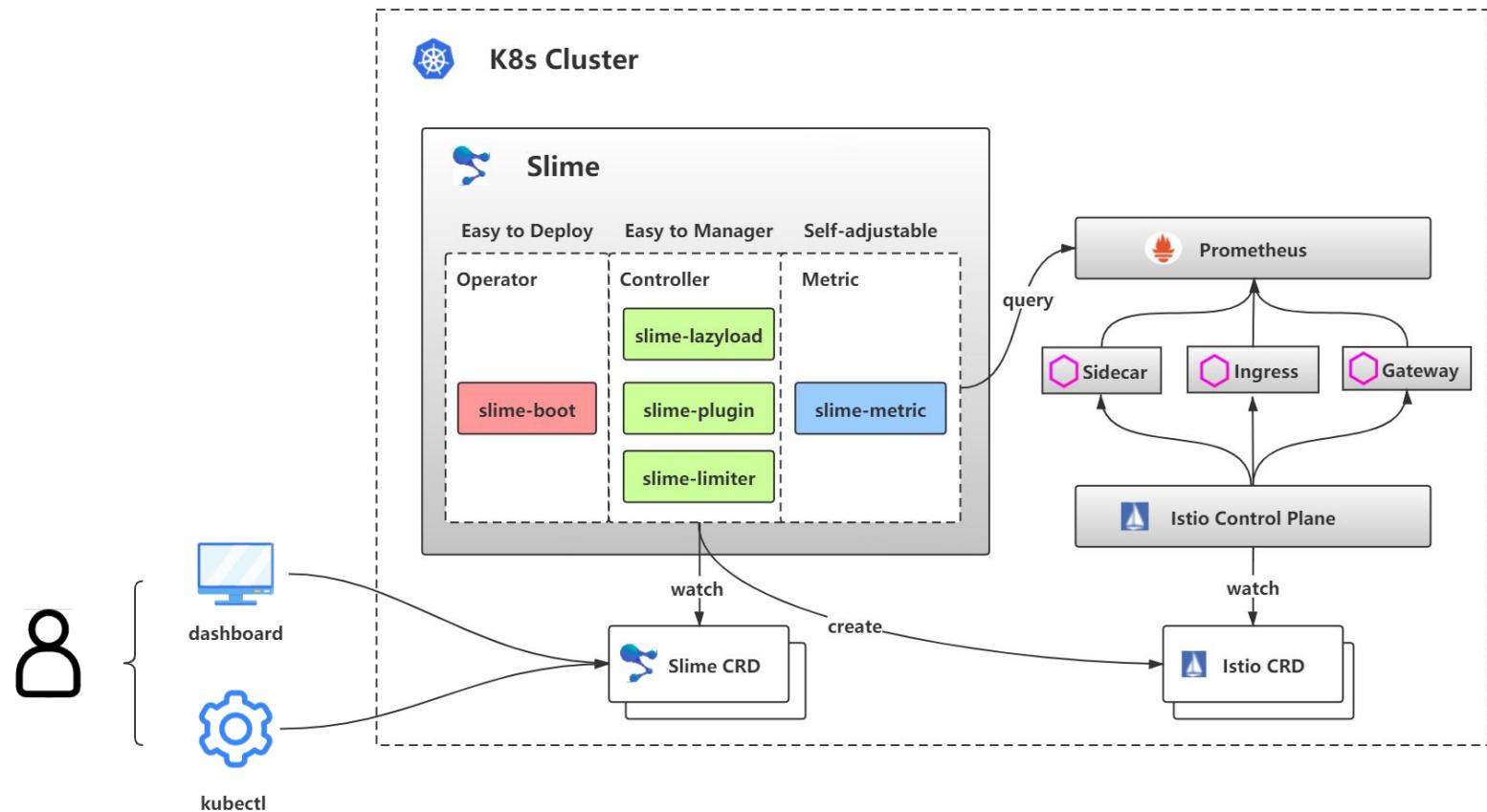
服务运维的5个阶段：



# 架构简介

Slime架构分为四大块：

- 1.slime-boot，部署slime-module的operator组件
- 
- 2.slime-controller，slime-module的核心线程，负责将SlimeCRD转换为IstioCRD。
- 3.slime-metric，slime-module的监控获取线程，用于感知服务状态，slime-controller会根据服务状态动态调整服务治理规则。
- 4.slime-builder，通过slime-builder可以快速构建slime-module（TODO）。



# 架构简介

基于slime框架，我们开放出了三个slime模块，分别解决了在生产落地过程中出现的一些问题：

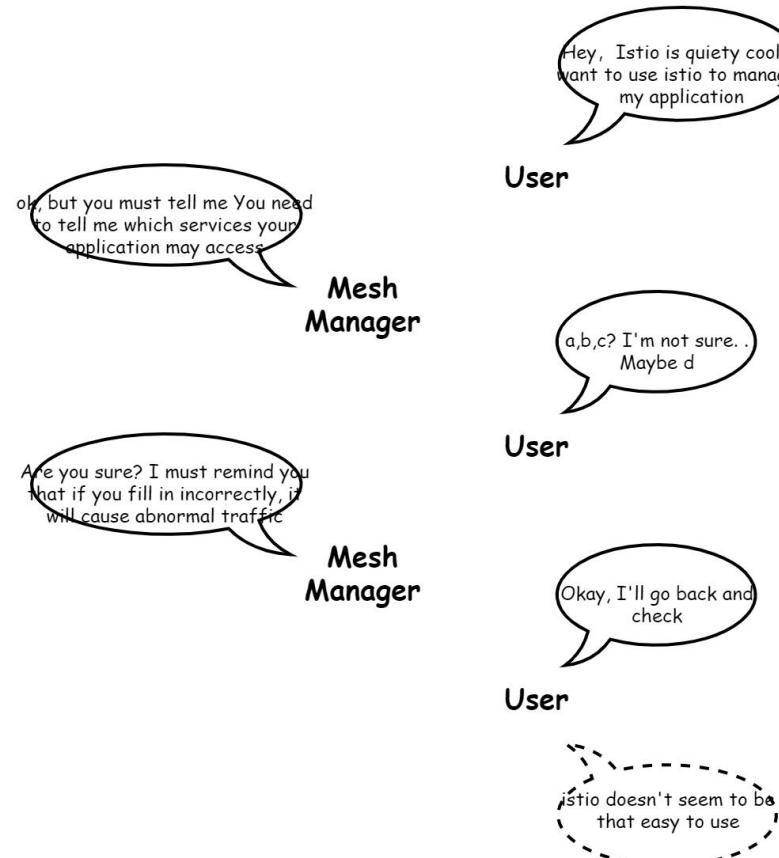
1. 配置懒加载，解决配置全量下发带来的性能问题
2. HTTP插件管理，解决插件扩展以及插件配置维护的问题
3. 自适应限流，解决限流功能配置复杂，服务级限流缺失的问题，同时结合监控数据实现自适应限流



# Slime的解决方案

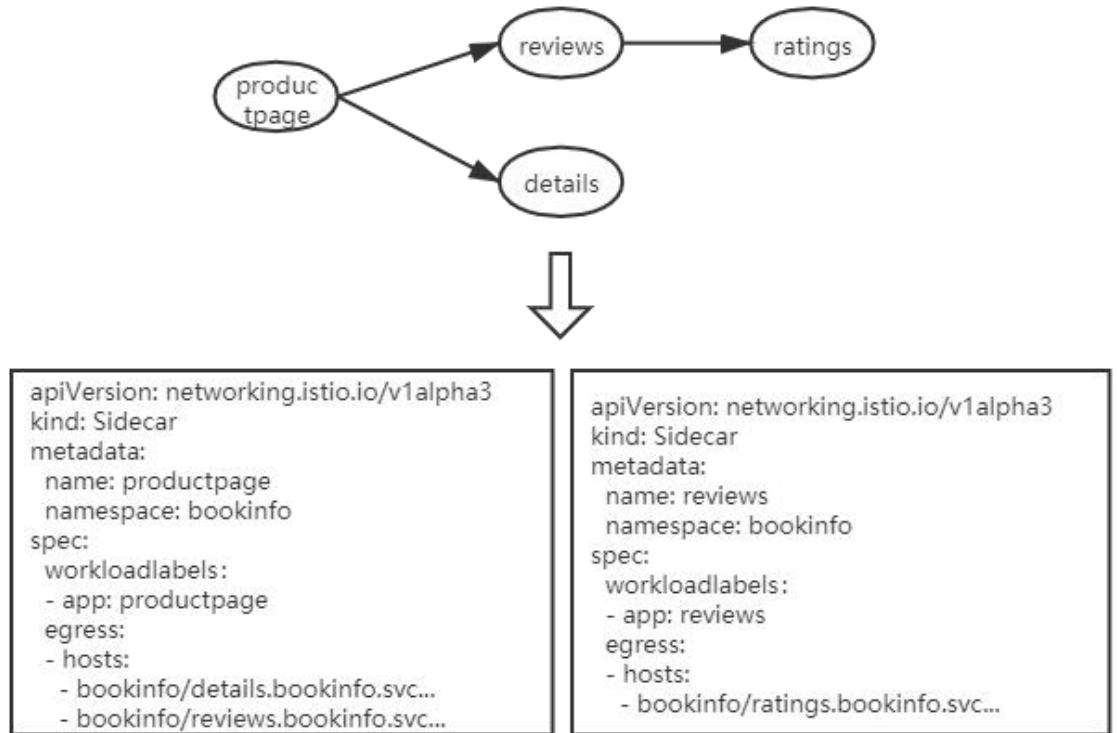
# 配置懒加载

解决网格性能问题的社区方案：用户提前根据服务依赖关系配置 Sidecar Scope。



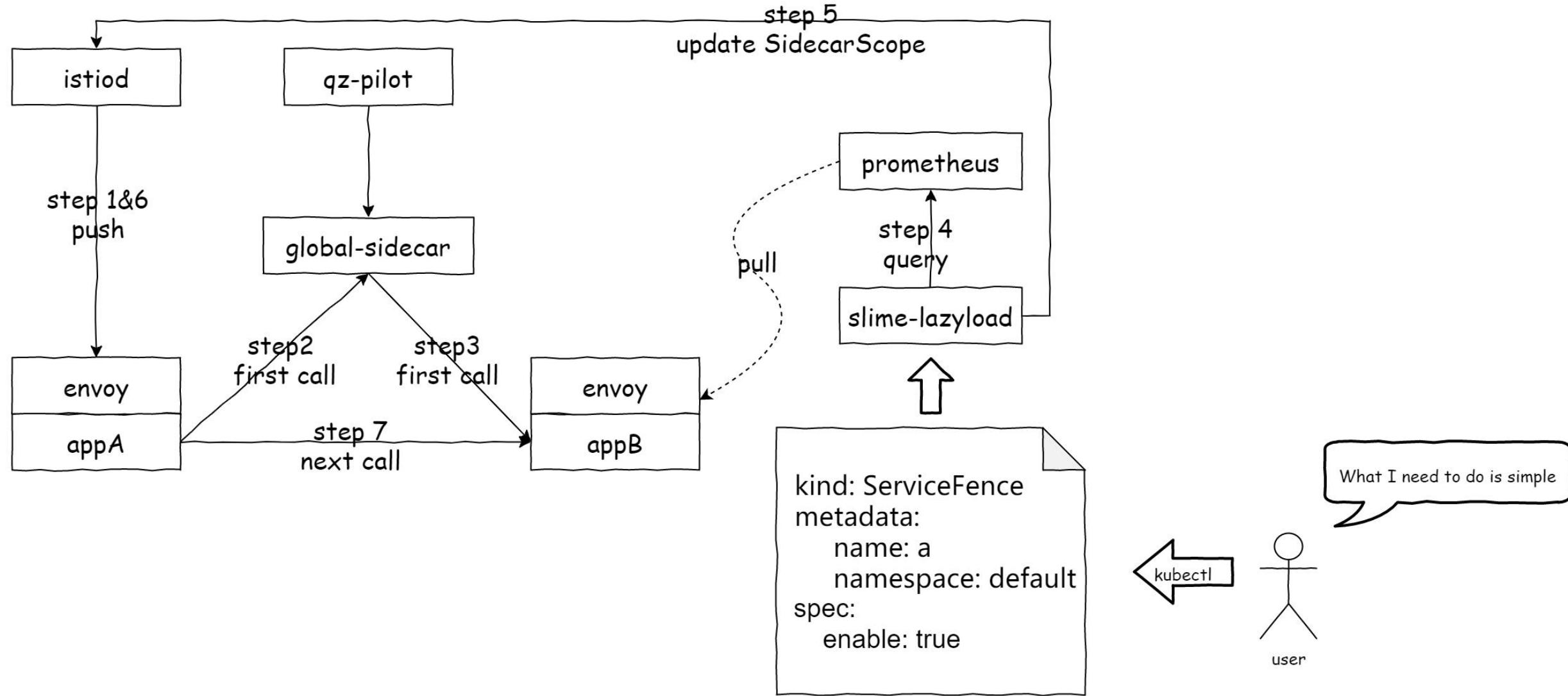
易用性和性能不可兼得

破局思路：根据拓扑图自动生成配置依赖信息



但是拓扑的生成依赖一次成功调用，如何在缺乏服务发现信息时依然能调用成功时实现配置懒加载的关键

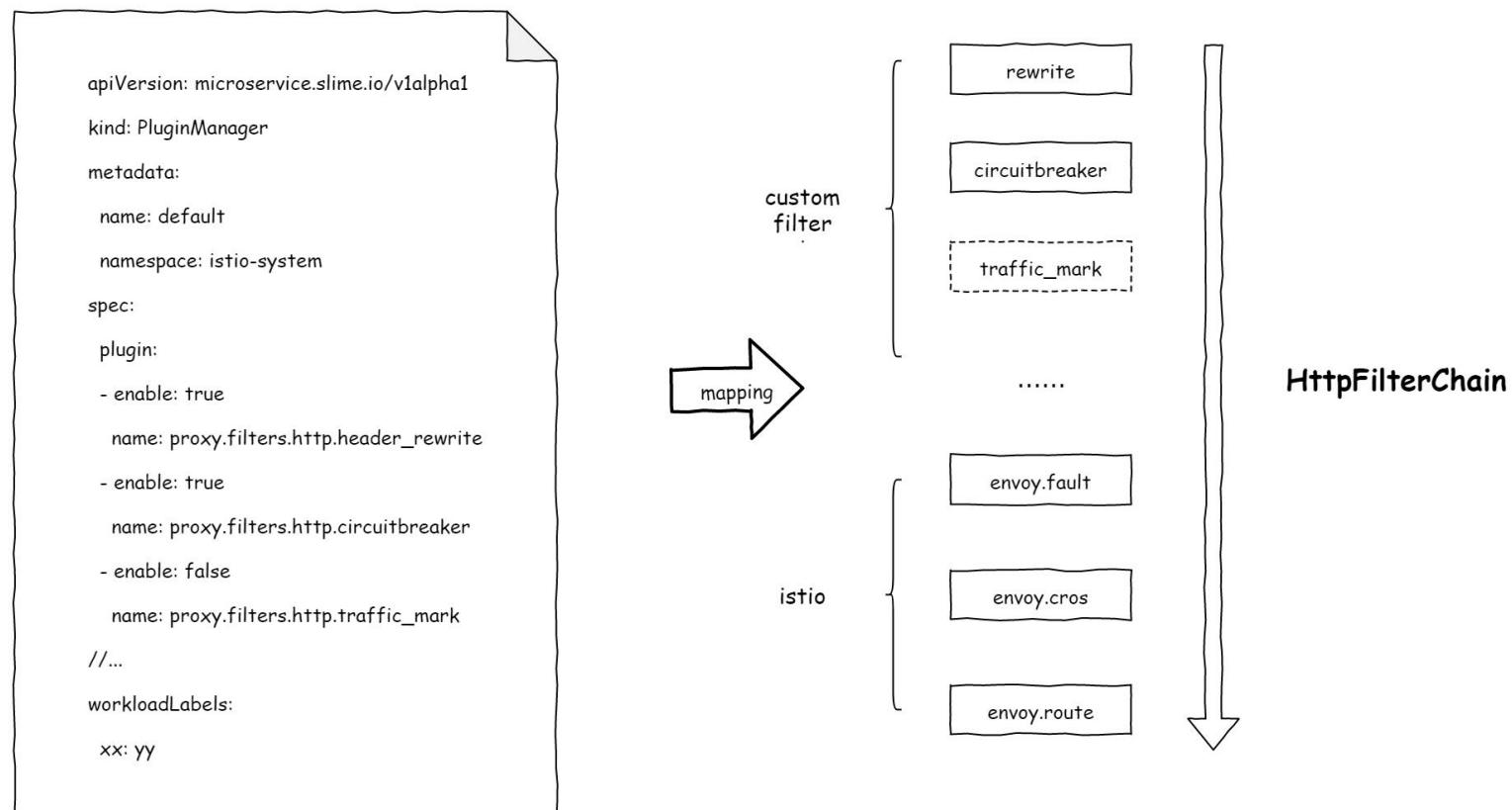
# 配置懒加载



# HTTP插件管理

在网关场景下，流量管理比较复杂，需要使用定制化插件来处理流量，在开发Slime的插件模块之前，插件扩展只能通过 `envoyfilter` 来实现，`envoyfilter` 是 xds 层面的配置，管理和维护这样的配置有一定的门槛，且非常容易出错。

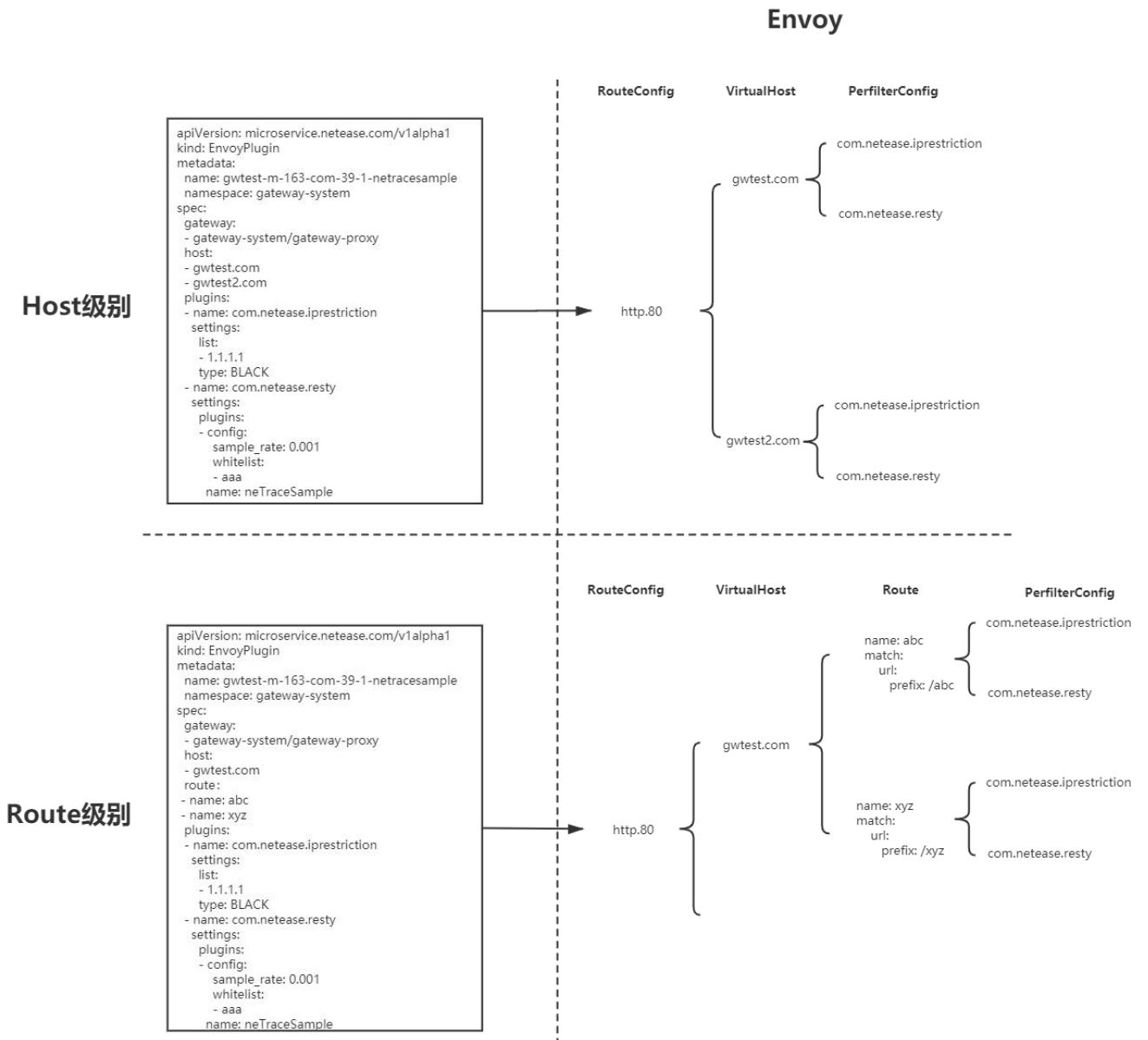
于是我们决定利在 Slime 的接口扩展特性在 `envoyfilter` 上层做一层面向插件管理的抽象。在 `Http` 插件管理这一特定场景下，我们可以预先确定 `EnvoyFilter` 中哪部分需要被 Patch。xDS 中关于 HTTP 插件的配置有两段，一部分在 LDS 中，作为 `HttpConnectionManager` 的 `SubFilter`，决定了 Envoy 将会启用哪些插件以及插件执行顺序，这一部分被我们抽象为新的 CRD—`Pluginmanager`。



# HTTP插件管理

另一部分在RDS中并且有两个粒度，分别是 virtualHost粒度的perFilterConfig以及route粒度的 perFilterConfig，决定了具体某个Host或者是Route的插件行为。部分被抽象为EnvoyPlugin，通过 EnvoyPlugin的Host/Route字段可以设置插件配置的生效范围。在网关的控制台上，后端服务往往被映射为某个Host下的某几个API接口， EnvoyPlugin更加贴合网关的配置模型。

虽然在Envoy层面这些插件配置还是会被展开为树状结构，但是在配置管理层面只需处理一个序聚合的数组。



# 自适应限流

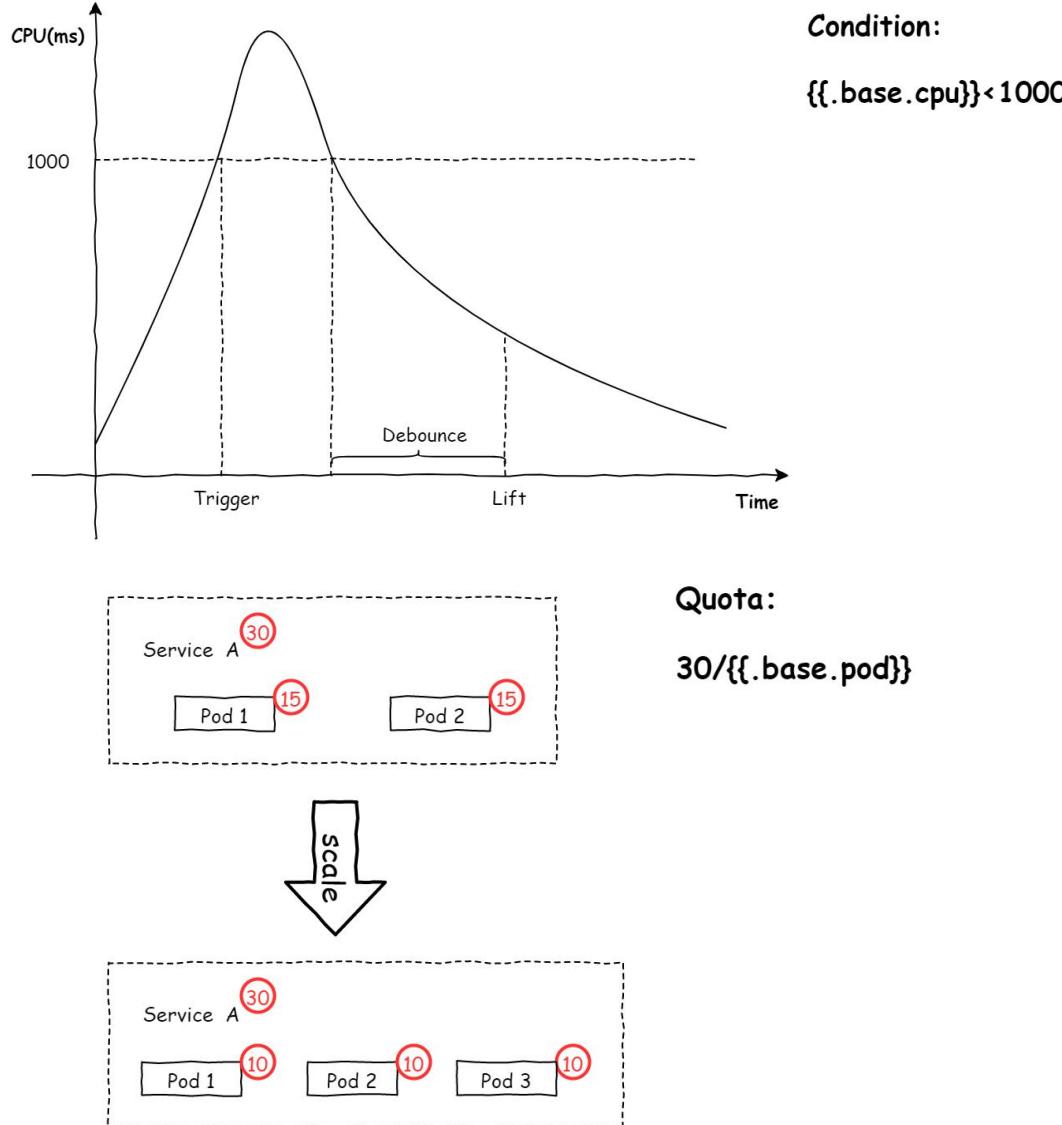
## Slime限流模块的优势：

1. 配置简单
2. 基于自定义监控项进行自适应限流
3. Subset维度的限流
4. 服务级本地限流

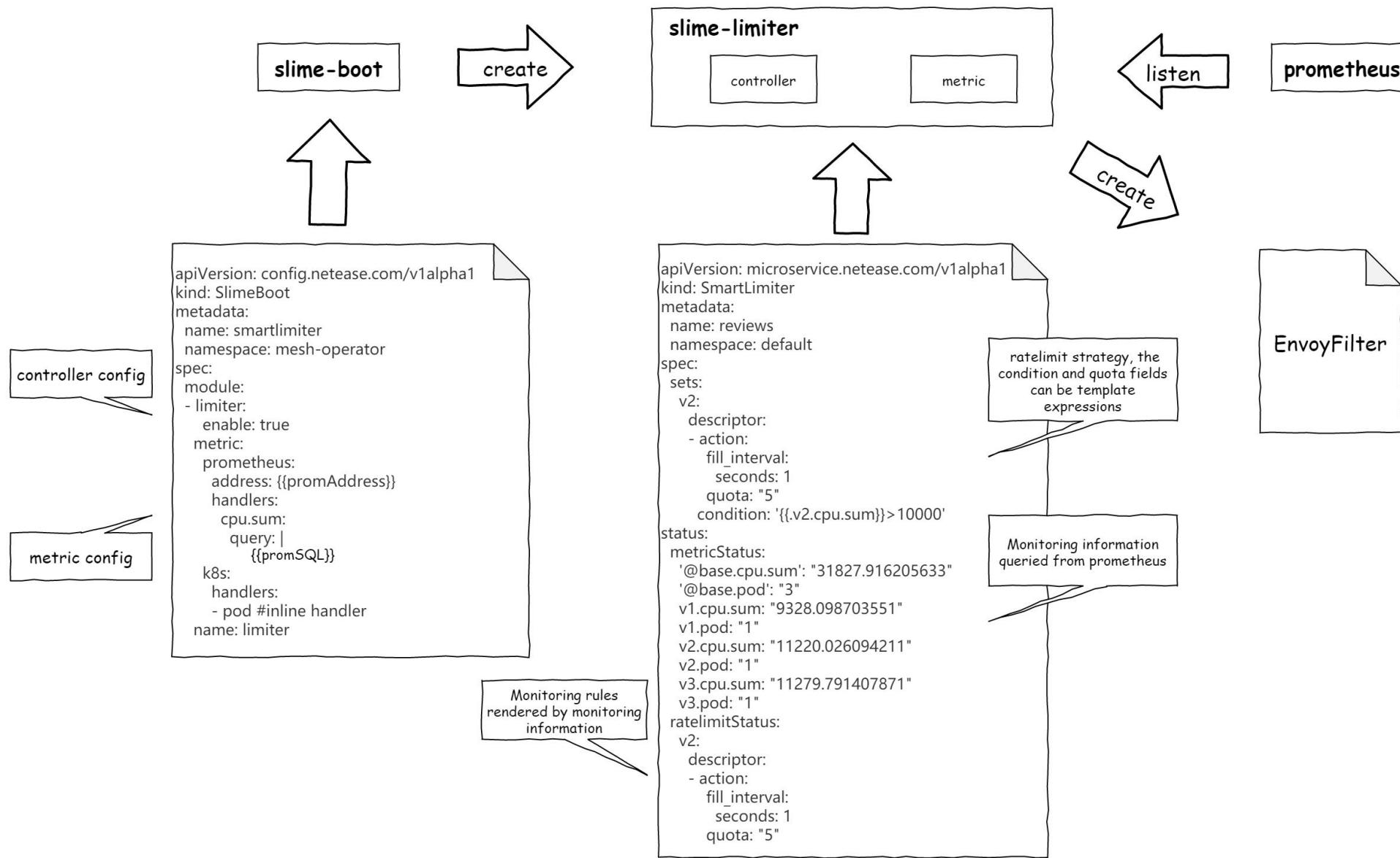
例如希望在CPU超过1000ms时触发服务A的访问限制，限额为30qps，对应的SmartLimiter定义如下：

```

kind: SmartLimiter
metadata:
  name: a
  namespace: default
spec:
  set:
    _base:
      descriptors:
        - action:
            fill_interval:
              seconds: 1
      quota: "30/{{.base.pod}}" # 30为该服务的额度，将其均分给每个pod
      condition: "{{.base.cpu}}>1000" # 根据监控项{cpu}的值自动填充该模板
  
```



# 自适应限流





# 高效网格管理之路

# 治理策略和治理规则

Istio的CRD本质是一套定义流量行为的规则。通常情况下一个稳定系统的流量行为是需要基于服务状态而改变的。监控是判断服务状态的有效手段，基于监控制定治理策略，动态调整服务规则是我们认为的一条更高效的流量管理之路。通过下表，可以比较治理规则和治理策略的异同：

	治理规则	治理策略	基于的监控项
服务发现	填写依赖服务	按需发现	服务拓扑
限流规则	触发/不触发，固定配额	条件触发，动态配额	时延，CPU，内存等
路由规则	路由+分组	灰度发布	错误率，时间
		版本分流	自定义监控项

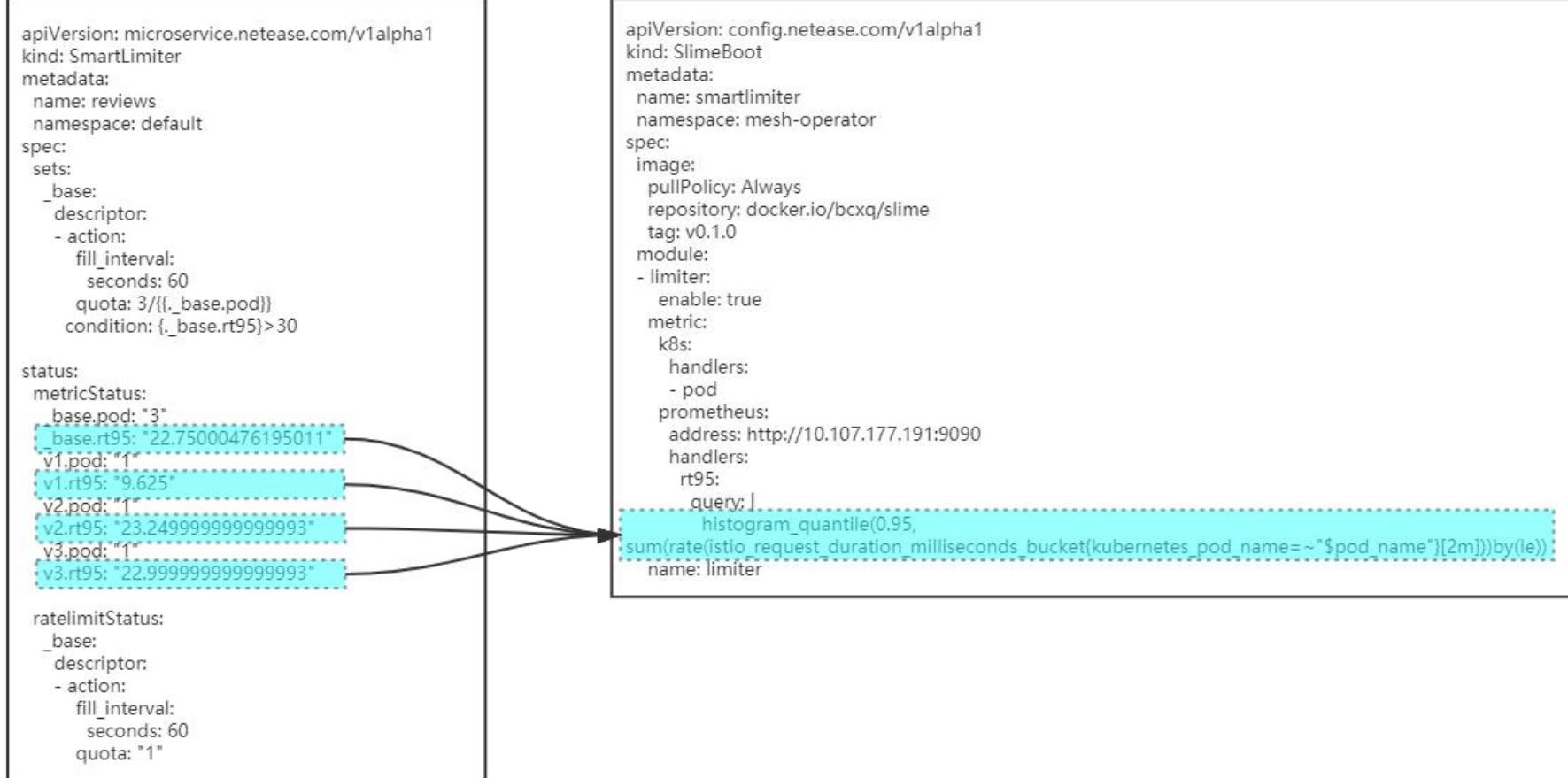
通过治理策略的抽象可以降低人为感知服务状态的运维负担，减少人为判断服务状态的错误率，服务异常发生时的治理规则响应也能更为及时。

# Slime的扩展性

## 监控模块扩展性

我们支持了使用prometheus作为监控源，用户只需填写promSQL语句即可为slime模块配置其关心的监控数据。

以自适应限流模块为例，我们可以通过更改限流模块的metric配置添加或更改其关心的监控指标，从而实现基于该监控指标的自适应限流。

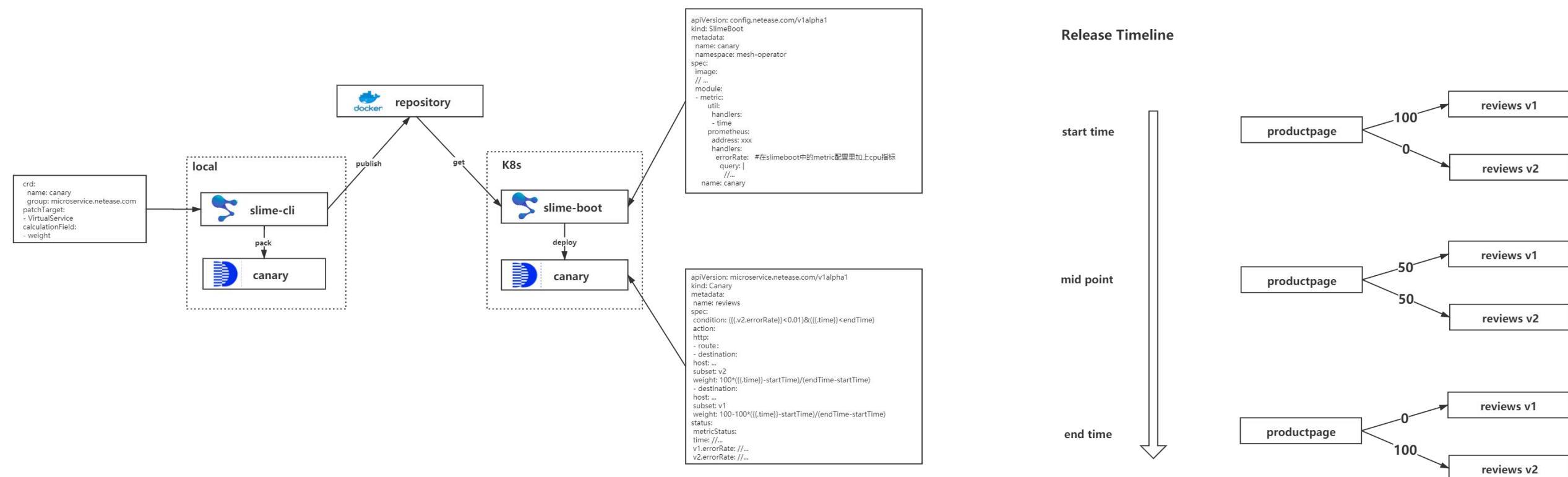


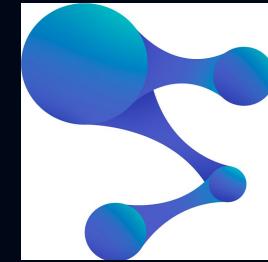
# Slime的扩展性

## 治理模块扩展性

治理模块的扩展性是slime后续发展的主要方向，slime将在v0.2.0版本中推出slime-module的代码生成工具slime-scaffolds，通过slime-scaffolds可以快速编写出一个具有auto-pilot能力的slime-module，这个过程我们希望做到0编码，用户提供一份配置文件即可实现。

下图是通过slime架构开发灰度发布模块的一个流程图（假想）：





# slime-io

Smart Service Mesh Manager

<https://github.com/slime-io>



网易轻舟

# Thank You!



Tetrate 中国



云原生社区