# 使用 Go 重新实现一套 Service Mesh

**张晋涛**

API7.ai
云原生技术专家

# 个人介绍

- 张晋涛
- Apache APISIX PMC
- Kubernetes Ingress-NGINX maintainer
- Microsoft MVP
- 『K8S 生态周报』的维护者
- 公众号：『MoeLove』
- GitHub: https://github.com/tao12345666333
- Blog: https://moelove.info/

# 目 录

第一部分

# Service Mesh 现状

# Solo 2022 调查报告

## 2022 Service Mesh Adoption Survey

This annual survey dives into how organizations are adopting microservices as well as their usage of Kubernetes and Istio.

**89%**

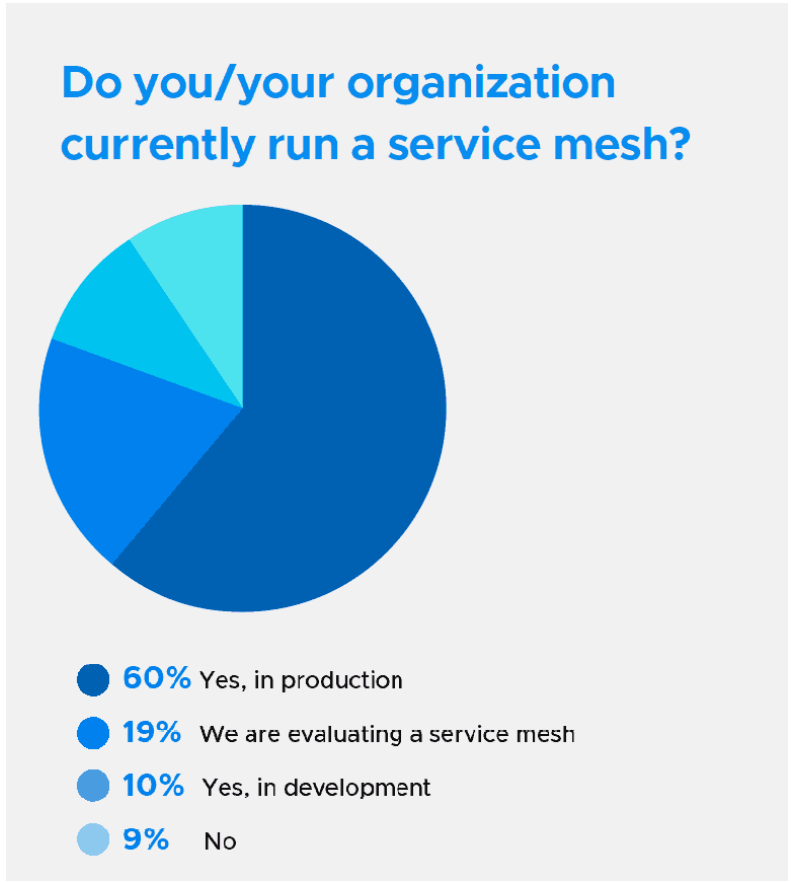
of organizations report very positive impact on app reliability as a result of using service mesh

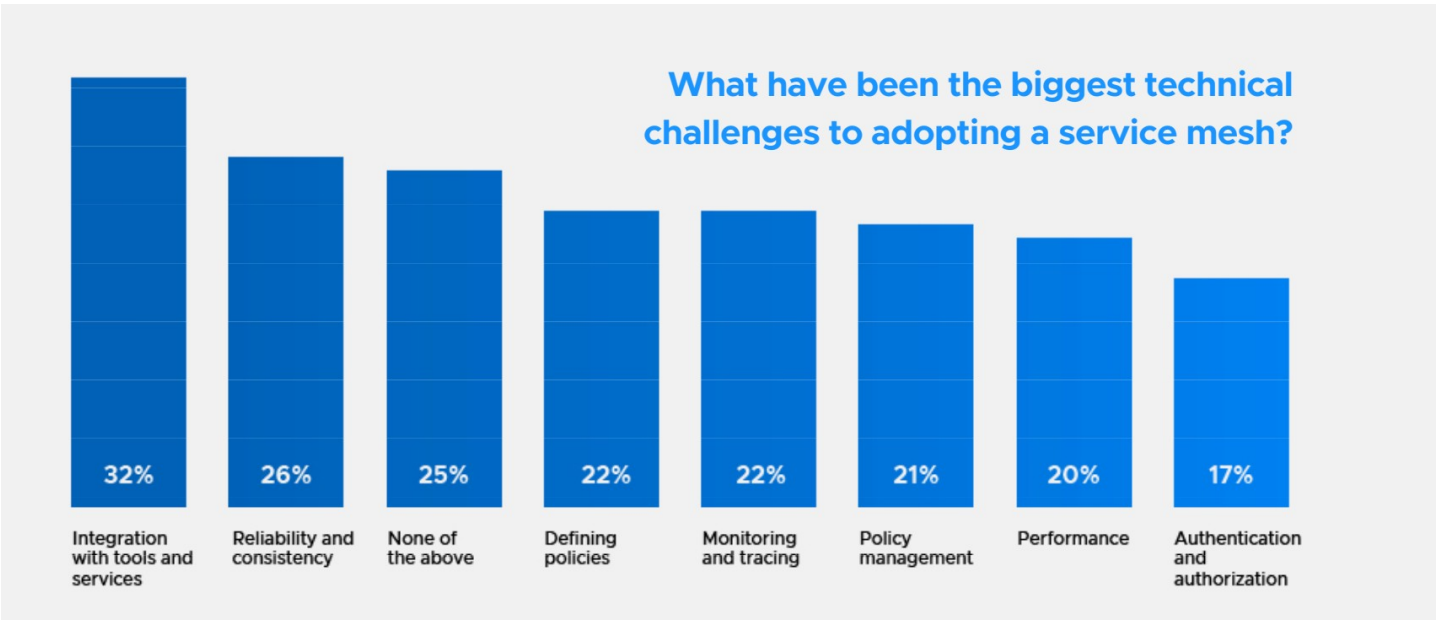
**Key Findings include:**

- Fully **85% of companies** say they are modernizing their apps to a **microservices architecture**
- A majority of companies **(53%) report at least half of their production workloads** running on **Kubernetes**
- Fully **93% of companies** are using or evaluating an **API gateway**
- Nearly half of all companies **(49%)** report using a service mesh at some level with a further **38% evaluating a service mesh for use**
- A majority of organizations **(56%) with at least half of their apps** on microservices architectures have release cycles that are daily or more frequent.

Ref: https://www.solo.io/resources/report/2022-service-mesh-adoption-survey/

# CNCF 2022 年报告

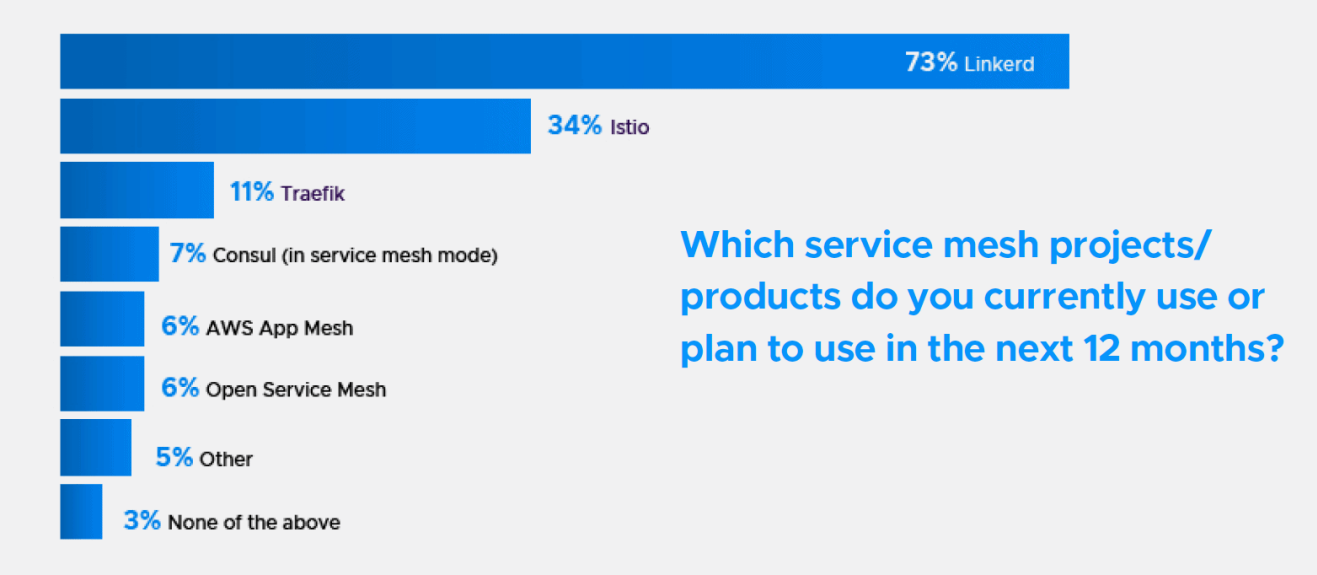



- Service Mesh 正在被越来越多的组织采纳或评估
- Service Mesh 方案采纳过程中存在多种挑战
  - 如何集成
  - 可观测性
  - 认证和授权
  - 策略管理
  - 性能
  - 可靠性和一致性

Ref: https://bit.ly/cncf-service-mesh

# CNCF 2022 报告



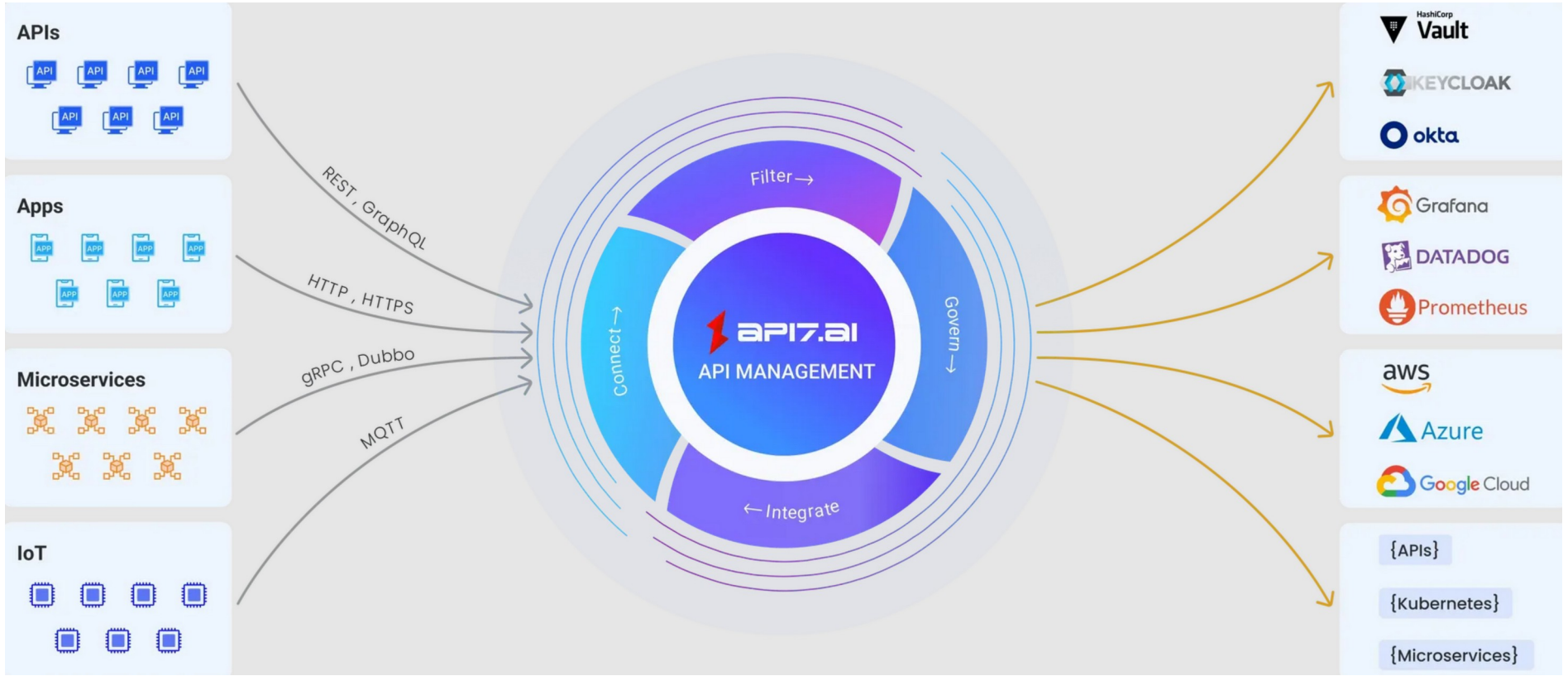

- Service mesh 方案百花齐放
- Linkerd 和 Istio 占据了主要市场

第二部分

# 为什么要重新实现 Service Mesh

# 背景

- API7.ai 是一家提供 API 全生命周期管理服务 / 产品的公司
- API7.ai 是 Apache APISIX 的创始团队
- Apache APISIX 是 Apache 基金会的顶级开源项目，是一款高性能易扩展的云原生 API 网关



# 重新实现 Service Mesh 的原因

- 简化复杂度
  - Envoy 进行二次开发的复杂度较高， Apache APISIX 可以使用任意语言进行插件开发
  - Apache APISIX 包含近 100 种开箱即用的插件
- 降低引入 Service Mesh 后的网络损耗
  - Apache APISIX 性能更优
- 构建基于 Apache APISIX 的 Service Mesh 生态
  - Apache APISIX 已有大量国内外用户，从 API 网关到 Service Mesh 是用户的诉求

第三部分

# 为什么用 Go 实现 Service Mesh

# Go 在云原生领域的优势

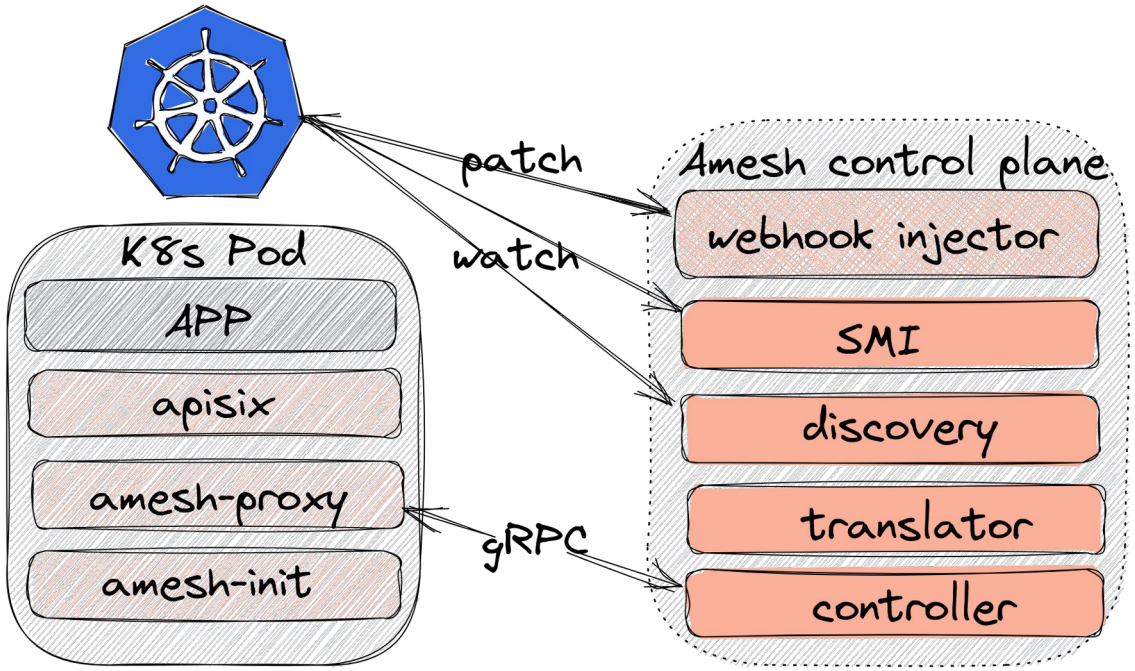
- 广泛的库和工具支持
- 易于部署
  - 我们希望此方案尽可能简单，资源占用少且完全通过容器化部署
- 性能优势
  - 在 Service Mesh 场景下会有大量的网络请求和数据传输
- 生态丰富
  - Docker， Kubernetes， Istio 等项目均通过 Go 实现，便于利用成熟的工具集
- 开发效率高

第四部分

# 如何用 Go 实现 Service Mesh

# 一套 Service Mesh 的通用架构

- Sidecar 模型应用最广
- 通过 webhook 实现 sidecar 自动注入
- Sidecar 中进行流量代理和拦截
- Controller 处理配置翻译和服务发现



# 配置规范 SMI

- SMI 是一套中立的规范
- 众多项目都进行了适配
- 核心使用者是 Open Service Mesh（计划归档）

## Service Mesh Interface Documents

The following documents are available:

| | Latest Release |
|---|---|
| **Core Specification:** | |
| SMI Specification | v0.6.0 |
| | |
| **Specification Components** | |
| Traffic Access Control | v1alpha3 |
| Traffic Metrics | v1alpha1 |
| Traffic Specs | v1alpha4 |
| Traffic Split | v1alpha4 |

## Ecosystem

- **Consul Connect*:** service segmentation (consul.io/docs/connect)
- **Flagger:** progressive delivery operator (flagger.app)
- **Gloo Mesh:** Multi-cluster service mesh management plane (solo.io/products/gloo-mesh)
- **Istio*:** connect, secure, control, observe (servicemeshinterface/smi-adapter-istio)
- **Linkerd:** ultralight service mesh (linkerd.io)
- **Traefik Mesh:** simpler service mesh (traefik.io/traefik-mesh)
- **Meshery:** the service mesh management plane (layer5.io/meshery)
- **Rio:** application deployment engine (rio.io)
- **Open Service Mesh:** lightweight and extensible cloud native service mesh (openservicemesh.io)
- **Argo Rollouts:** advanced deployment & progressive delivery controller (argoproj.io)

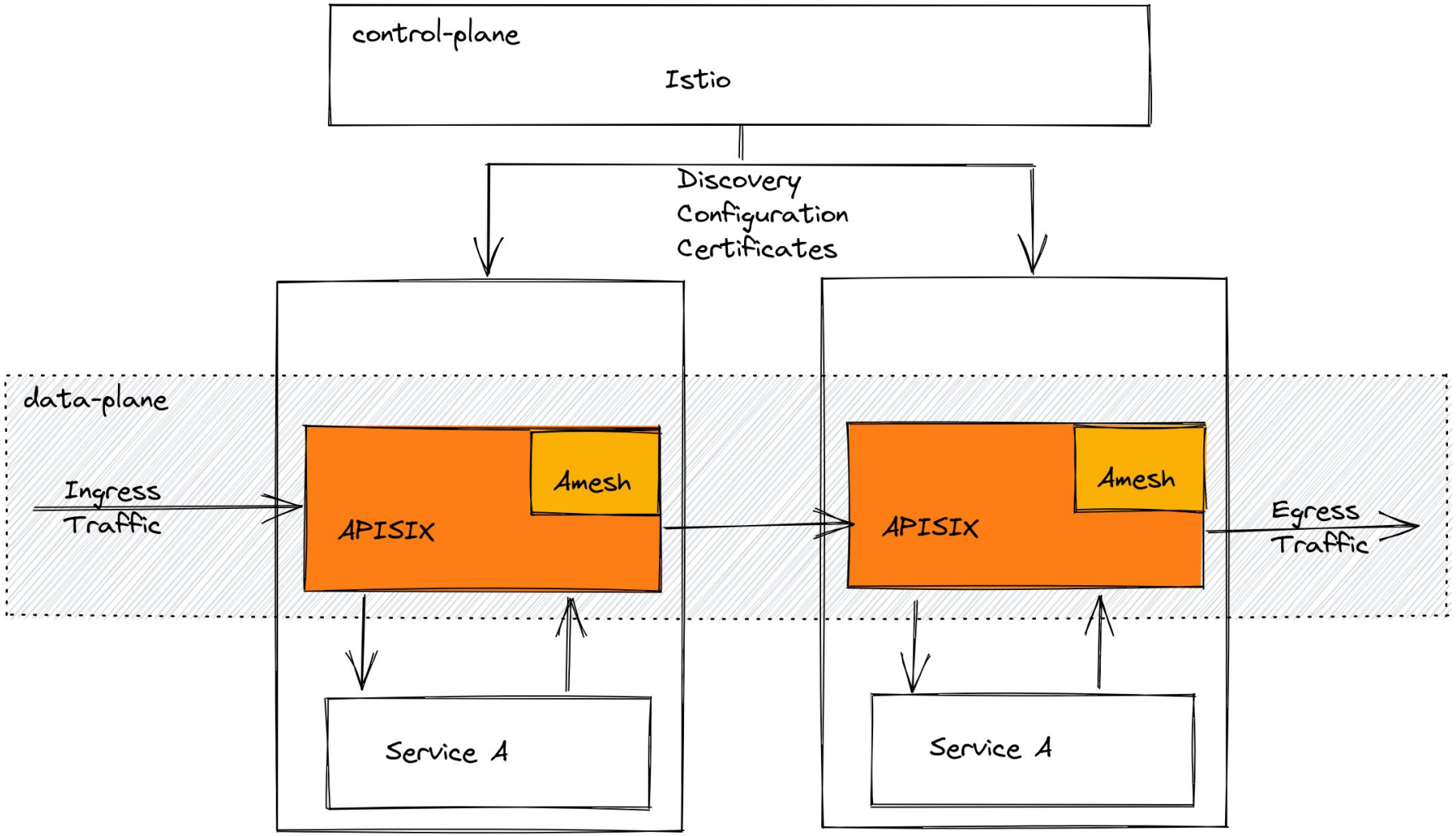
第五部分

# 我们有哪些收获

# 收获和方向调整

- 全新的采用 sidecar 模型的通用型 Service Mesh 很难短时间展露头角
  - 企业内自研的方案不在此列
- SMI 度过“甜蜜期”后，后续投入很少
  - Gateway API （GAMMA）成为主要方向
- 借力社区会更利于开源项目发展
  - Apache APISIX 的优势在于数据面
  - Istio 有广泛的社区和用户基础
  - 使用 Apache APISIX 替换 Envoy 成为 Istio 的数据面是一条相对有效的路径

# Amesh 架构




项目地址： https://github.com/api7/amesh

# Amesh 如何实现

- 支持 xDS API
  - EDS,RDS,SDS,LDS 等
- 将通过 xDS API 下发的数据转义为 APISIX 的配置
- 让 APISIX 中的配置生效
- 让 Istio 感知到 Amesh 的存在
  - 部署时注入 https://github.com/api7/Amesh/blob/main/manifest/injection-template.yaml

# Amesh 如何实现

```
// Resource types in xDS v3.
const (
    apiTypePrefix       = "type.googleapis.com/"
    EndpointType        = apiTypePrefix + "envoy.config.endpoint.v3.ClusterLoadAssignment"
    ClusterType         = apiTypePrefix + "envoy.config.cluster.v3.Cluster"
    RouteType           = apiTypePrefix + "envoy.config.route.v3.RouteConfiguration"
    ScopedRouteType     = apiTypePrefix + "envoy.config.route.v3.ScopedRouteConfiguration"
    ListenerType        = apiTypePrefix + "envoy.config.listener.v3.Listener"
    SecretType          = apiTypePrefix + "envoy.extensions.transport_sockets.tls.v3.Secret"
    ExtensionConfigType = apiTypePrefix + "envoy.config.core.v3.TypedExtensionConfig"
    RuntimeType         = apiTypePrefix + "envoy.service.runtime.v3.Runtime"

    // AnyType is used only by ADS
    AnyType = ""
)
```

```
switch resp.GetTypeUrl() {
case types.RouteConfigurationUrl:
    p.logger.Debugw(color.BlueString("recv RDS"),
        zap.Any("resp", resp),
    )
    for _, res := range resp.GetResources() {
        var route routev3.RouteConfiguration
        err := anypb.UnmarshalTo(res, &route, proto.UnmarshalOptions{
            DiscardUnknown: true,
        })

        if err != nil {
            p.logger.Errorw("found invalid RouteConfiguration resource",
                zap.Error(err),
                zap.Any("resource", res),
            )
            continue
        }

        p.logger.Debugw(color.GreenString("process route configurations"),
            zap.Any("route", &route),
        )

        partial, err := p.processRouteConfigurationV3(&route)
        if err != nil {
            p.logger.Errorw("failed to process RouteConfiguration",
                zap.Error(err),
                zap.Any("resource", res),
            )
            continue
        }

        resourceNames = append(resourceNames, route.Name)
        newManifest.Routes = append(newManifest.Routes, partial...)
    }
    if p.staticRouteConfigurations != nil {
        partial, err := p.processStaticRouteConfigurations(p.staticRouteConfigurations)
        if err != nil {
            p.logger.Errorw("failed to process StaticRouteConfiguration",
                zap.Error(err),
                zap.Any("resource", p.staticRouteConfigurations),
            )
        } else {
            newManifest.Routes = append(newManifest.Routes, partial...)
        }
    }
    p.routesLock.Lock()
    oldManifest.Routes = p.routes
    p.routes = newManifest.Routes
```

NORMAL +0 ~0 -0 main pkg/amesh/provisioner/xds.go

[tao] • Linux 6.2.9 0 zsh 1 zsh 2 zsh

项目地址： https://github.com/api7/amesh

# Amesh 如何让配置在 APISIX 生效

- CGO
  - 将 Amesh 构建为 .so 文件
  - 作为共享库挂载至 APISIX
  - 使用 NGINX lua_shared_dict 开辟内存空间
- NGINX FFI （ Foreign Function Interface ）
  - FFI 与 Amesh 通过 C 接口进行调用
  - 使用相同内存空间进行数据交换

```
55  func (s *SharedDictStorage) Store(key, value string) {
 1      if s.zone == nil {
 2          log.Warnw("zone is nil")
 3          return
 4      }
 5
 6      var keyCStr = C.CString(key)
 7      defer C.free(unsafe.Pointer(keyCStr))
 8      var keyLen = C.size_t(len(key))
 9
10      var valueCStr = C.CString(value)
11      defer C.free(unsafe.Pointer(valueCStr))
12      var valueLen = C.size_t(len(value))
13
14      errMsgBuf := make([]*C.char, 1)
15      var forcible = 0
16
17      C.ngx_lua_ffi_shdict_store(s.zone, 0x0004,
18          (*C.uchar)(unsafe.Pointer(keyCStr)), keyLen,
19          4,
20          (*C.uchar)(unsafe.Pointer(valueCStr)), valueLen,
21          0, 0, 0,
22          (**C.char)(unsafe.Pointer(&errMsgBuf[0])),
23          (*C.int)(unsafe.Pointer(&forcible)),
24      )
25  }
```

项目地址： https://github.com/api7/amesh

# Amesh 如何拦截流量

```
118 func (ic *iptablesConstructor) insertInboundRules() {
  1     if ic.cfg.InboundPortsInclude == "" {
  2         return
  3     }
  4     ic.iptables.AppendRuleV4(log.UndefinedCommand, PreRoutingChain, "nat", "-p", "tcp", "-j", InboundChain)
  5
  6     if ic.cfg.InboundPortsInclude == "*" {
  7         // Makes sure SSH is not redirected
  8         ic.iptables.AppendRuleV4(log.UndefinedCommand, InboundChain, "nat", "-p", "tcp", "--dport", "22", "-j", "RETURN")
  9         if ic.cfg.InboundPortsExclude != "" {
 10             for _, port := range split(ic.cfg.InboundPortsExclude) {
 11                 ic.iptables.AppendRuleV4(log.UndefinedCommand, InboundChain, "nat", "-p", "tcp", "--dport", port, "-j", "RETURN")
 12             }
 13         }
 14         ic.iptables.AppendRuleV4(log.UndefinedCommand, InboundChain, "nat", "-p", "tcp", "-j", InboundRedirectChain)
 15     } else {
 16         for _, port := range split(ic.cfg.InboundPortsInclude) {
 17             ic.iptables.AppendRuleV4(
 18                 log.UndefinedCommand, InboundChain, "nat", "-p", "tcp", "--dport", port, "-j", InboundRedirectChain,
 19             )
 20         }
 21     }
 22 }
 23
```

项目地址： https://github.com/api7/amesh

# Amesh 如何扩展 APISIX 原生能力

- 自定义资源 AmeshPluginConfig
  - 贴合 APISIX 原生语义
  - 配置时同步生效
  - 支持 APISIX 原生及扩展

```
70
1  func NewAmeshPluginConfigController(cli client.Client, scheme *runtime.Scheme,
2      clientset clientset.Interface, kubeClient kubernetes.Interface,
3      podInformer v1informer.PodInformer,
4      ameshPluginConfigInformer ameshv1alpha1informer.AmeshPluginConfigInformer) *AmeshPluginConfigReconciler {
5
6      eventBroadcaster := record.NewBroadcaster()
7      eventBroadcaster.StartRecordingToSink(&typedcorev1.EventSinkImpl{Interface: kubeClient.CoreV1().Events("")})
8
9      c := &AmeshPluginConfigReconciler{
10         Client:    cli,
11         clientset: clientset,
12         recorder:  eventBroadcaster.NewRecorder(scheme, v1.EventSource{Component: "AmeshPluginConfigReconciler"}),
13
14         Log:    ctrl.Log.WithName("controllers").WithName("AmeshPluginConfig"),
15         Scheme: scheme,
16
17         podInformer:   podInformer,
18         selectorCache: utils.NewSelectorCache(ameshPluginConfigInformer.Lister()),
19         pluginsCache:  map[string]*types.PodPluginConfig{},
20     }
21
```

项目地址： https://github.com/api7/amesh

# 感谢聆听！