扫码添加小助手，发送“CKA”加群

HUAWEI

CloudNative Lives

istio入门级实训

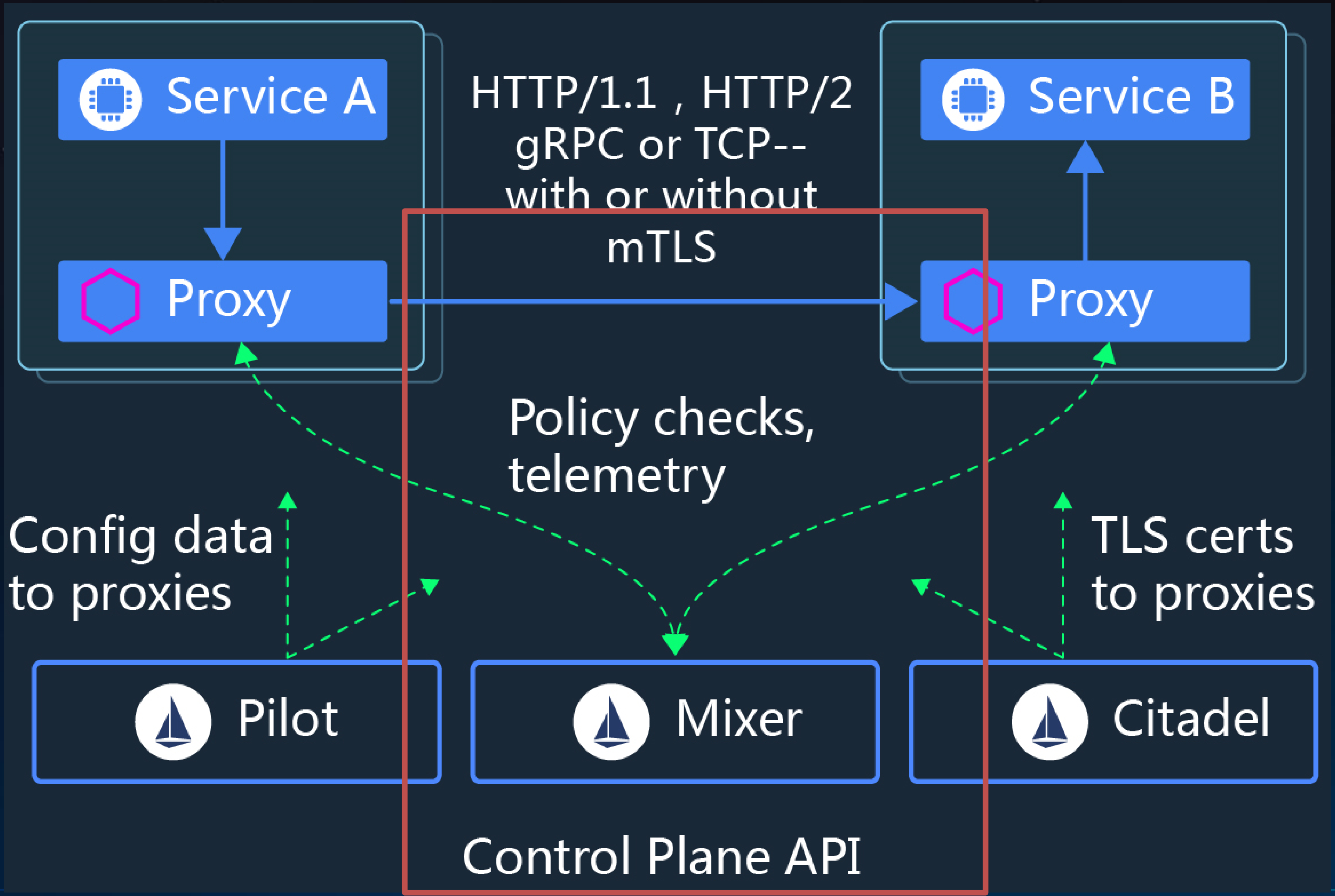
# xDS 协议解析

华为云容器团队核心架构师 & CNCF社区主要贡献者倾力打造

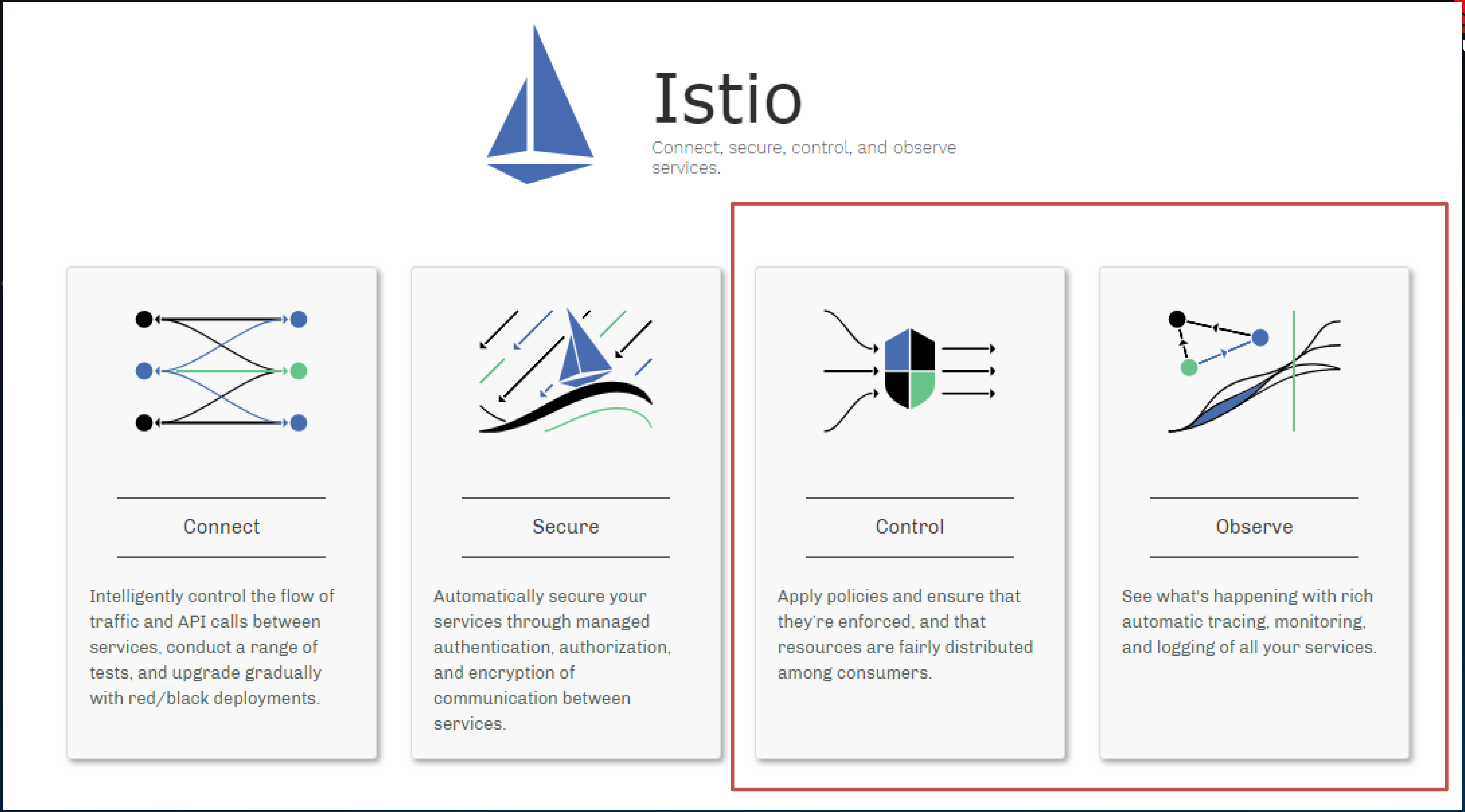
## Agenda

- Istio架构回顾&Mixer介绍
- Mixer的功能和设计
- Mixer的配置模型
- Mixer的典型应用
- Mixer实践1和2

# 回顾：Istio架构


Service A
Proxy
HTTP/1.1 , HTTP/2
gRPC or TCP--
with or without
mTLS
Service B
Proxy
Policy checks,
telemetry
Config data
to proxies
TLS certs
to proxies
Pilot
Mixer
Citadel
Control Plane API

# Istio 官方四大功能中两个基于Mixer实现



Istio

Connect, secure, control, and observe services.

**Connect**

Intelligently control the flow of traffic and API calls between services, conduct a range of tests, and upgrade gradually with red/black deployments.

**Secure**

Automatically secure your services through managed authentication, authorization, and encryption of communication between services.

**Control**

Apply policies and ensure that they're enforced, and that resources are fairly distributed among consumers.

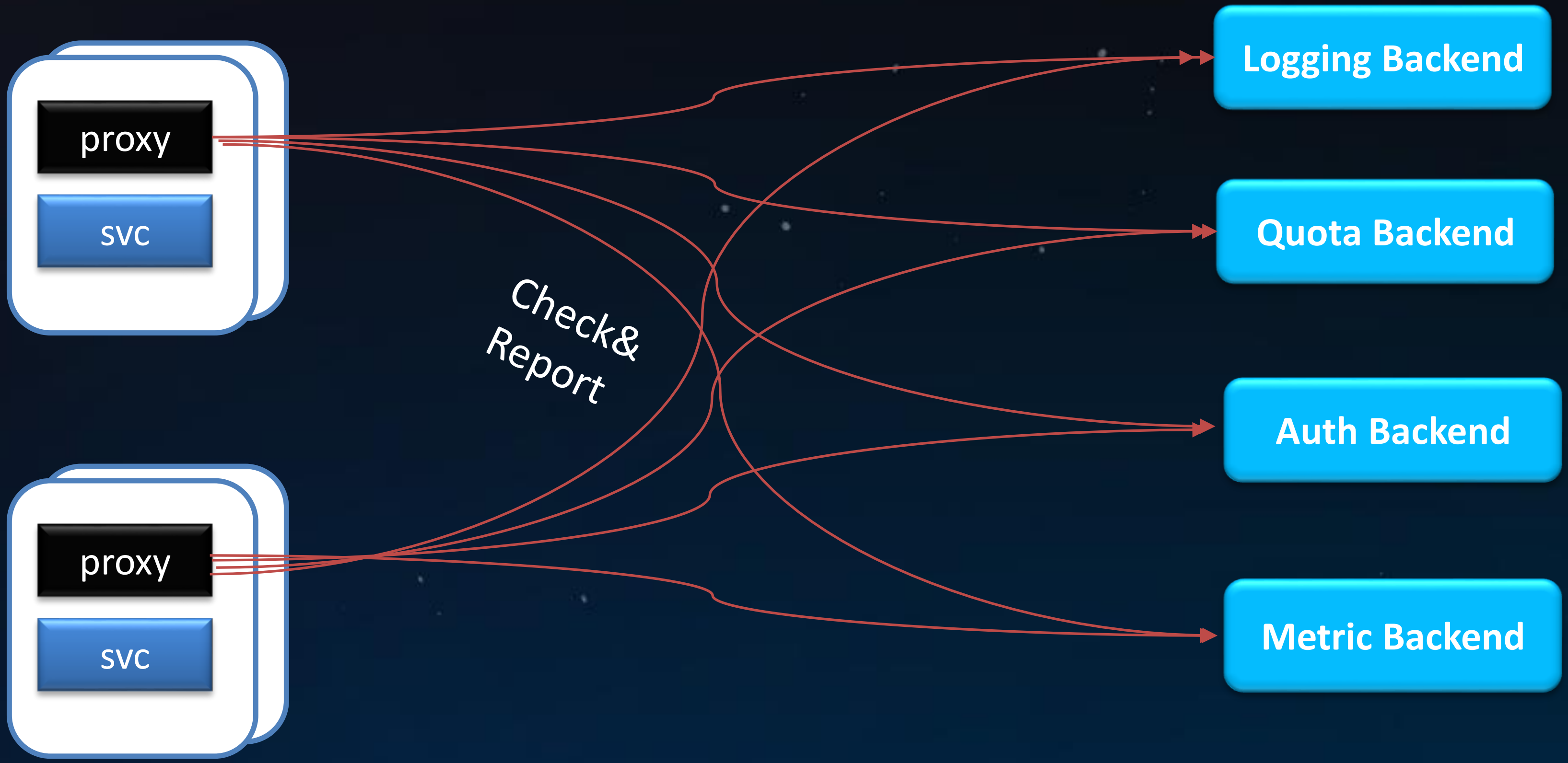
**Observe**

See what's happening with rich automatic tracing, monitoring, and logging of all your services.

# Mixer在Istio中角色

- 功能上：负责策略控制和遥测收集
- 架构上：提供插件模型，可以扩展和定制

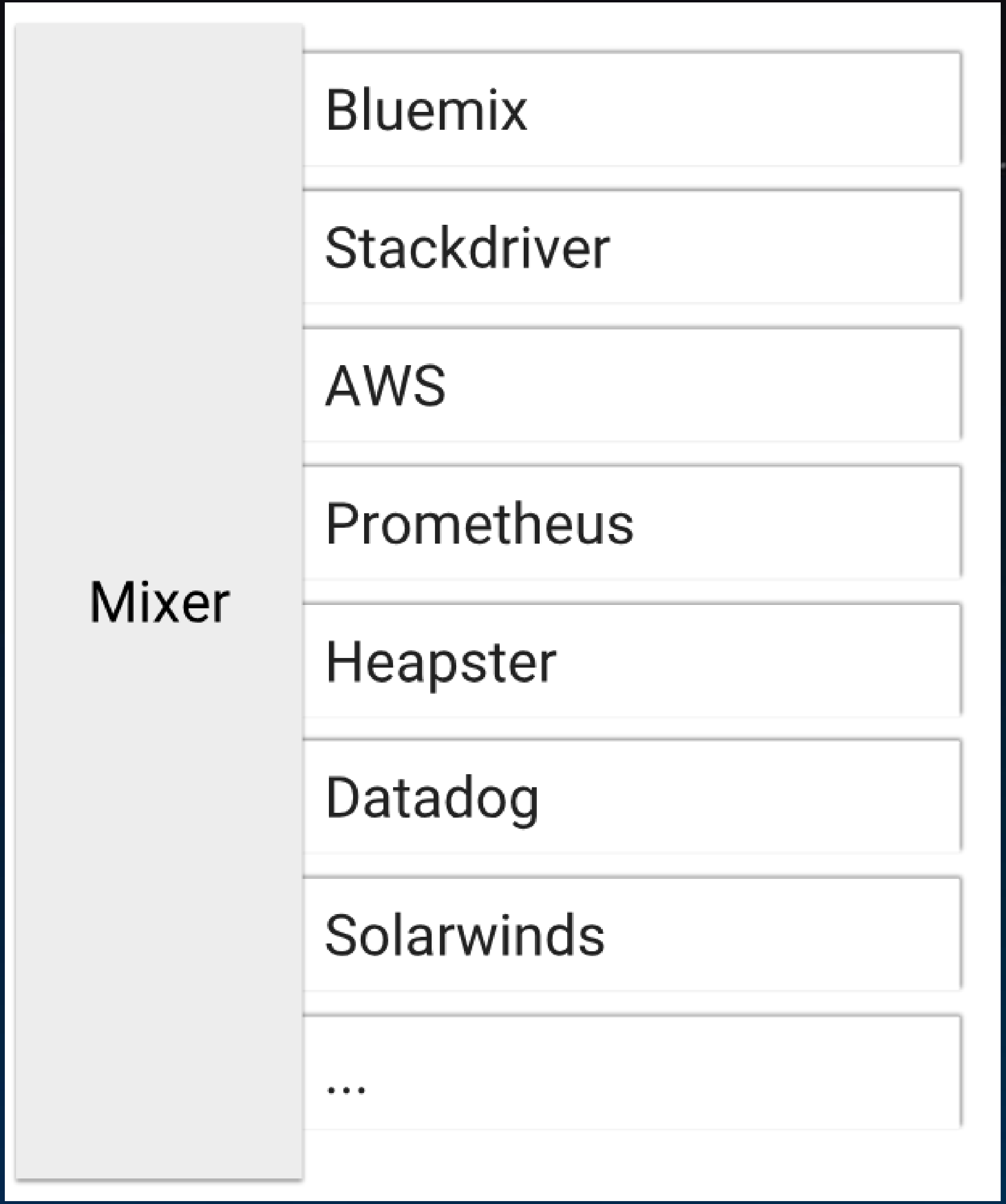
# 没有Mixer的时候


proxy
svc
proxy
svc
Check&
Report
Logging Backend
Quota Backend
Auth Backend
Metric Backend

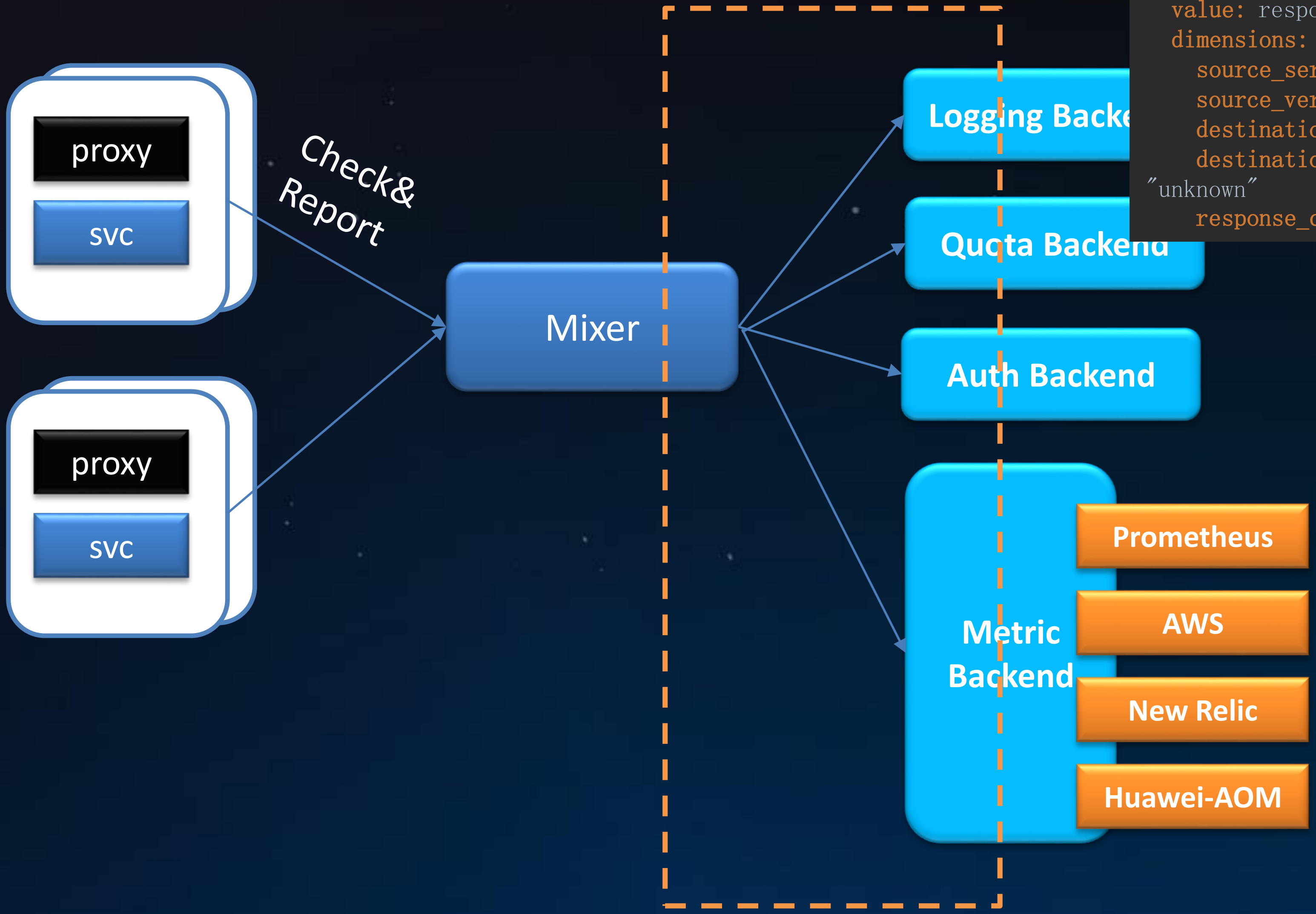
## Mixer的Adapter机制

Mixer 处理不同基础设施后端的灵活性是通过使用通用插件模型实现的，这种插件称为Adapter。

Mixer通过它们与不同的基础设施后端连接，这些后端可提供核心功能，提供日志、监控、配额、ACL 检查等



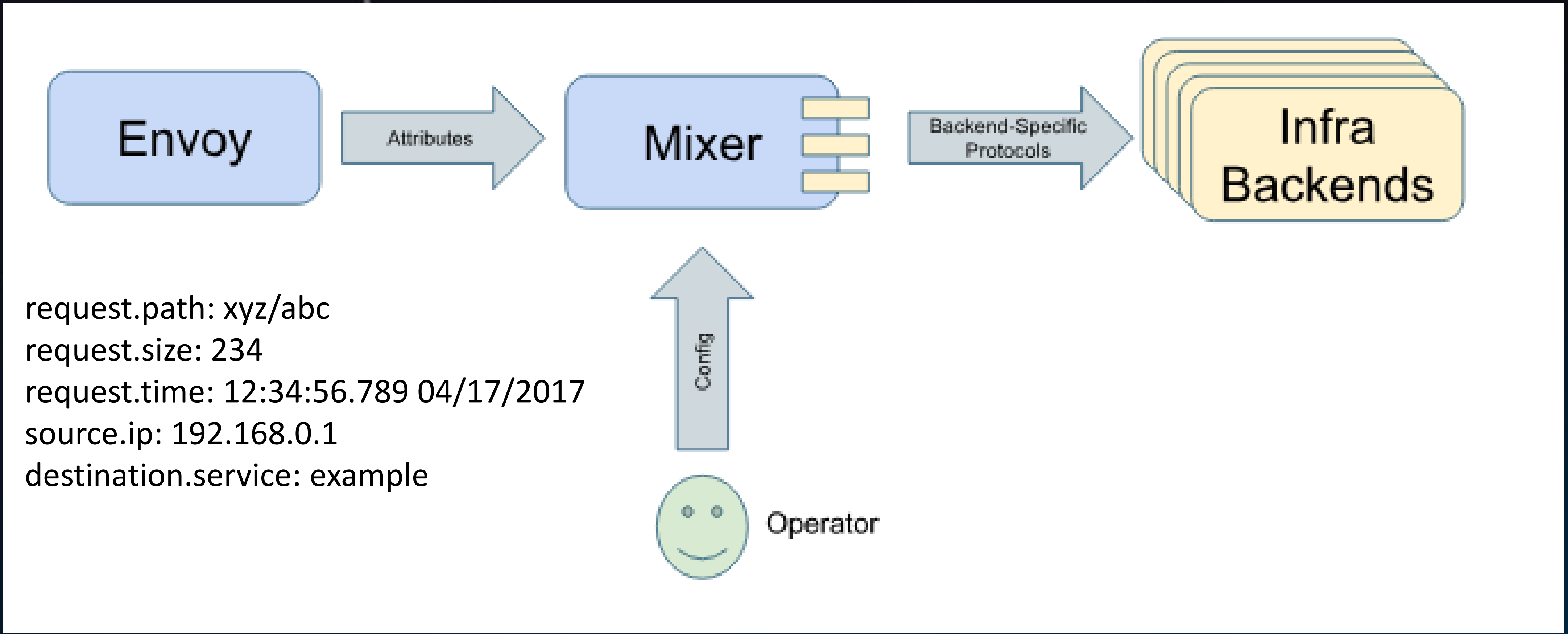
# Mixer完整视图



```yaml
apiVersion: "config.istio.io/v1alpha2"
kind: metric
metadata:
  name: requestduration
  namespace: istio-system
spec:
  value: response.duration | "0ms"
  dimensions:
    source_service: source.service | "unknown"
    source_version: source.labels["version"] | "unknown"
    destination_service: destination.service | "unknown"
    destination_version: destination.labels["version"] |
"unknown"
    response_code: response.code | 200
```

解耦、中介、运维时配置

# Mixer的处理流程



request.path: xyz/abc
request.size: 234
request.time: 12:34:56.789 04/17/2017
source.ip: 192.168.0.1
destination.service: example

1 Envoy生成属性上报Mixer
2. Mixer 调用对应后端处理属性

https://istio.io/docs/reference/config/policy-and-telemetry/attribute-vocabulary/

## Mixer 配置模型概述

- Handler: 创建 Handler,即配置Mixer适配器
- Instance: 从 Istio 属性中生成 instance。
- Rule: 配置一组规则，这些规则描述了何时调用特定适配器及哪些实例。

## Mixer 配置模型1： Handler

- 实例化一个Adapter，包括了Mixer和后端交互的接口。

```
apiVersion: ”config.istio.io/v1alpha2”
kind: stdio
metadata:
  name: handler
spec:
  outputAsJson: true
```

Stdio Adapter 定义参照：mixer/adapter/stdio/config/config.proto:94

# Mixer 配置模型2：实例（Instance）

```
apiVersion: "config.istio.io/v1alpha2"
kind: logentry
metadata:
  name: accesslog
  namespace: {{ .Release.Namespace }}
spec:
  severity: '"Info"'
  timestamp: request.time
  variables:
    sourceIp: source.ip | ip("0.0.0.0")
    sourceApp: source.labels["app"] | ""
    sourcePrincipal: source.principal | ""
    sourceName: source.name | ""
    destinationApp: destination.labels["app"] | ""
    destinationIp: destination.ip | ip("0.0.0.0")
    destinationServiceHost: destination.service.host | ""
    destinationWorkload: destination.workload.name | ""
    destinationName: destination.name | ""
    destinationNamespace: destination.namespace | ""
    protocol: request.scheme | context.protocol | "http"
    method: request.method | ""
    url: request.path | ""
    responseCode: response.code | 0
    responseSize: response.size | 0
    requestSize: request.size | 0
    requestId: request.headers["x-request-id"] | ""
    userAgent: request.useragent | ""
    responseTimestamp: response.time
    …
```

实例将请求中的属性映射成为适配器的输入，每次请求适配器消费的数据。

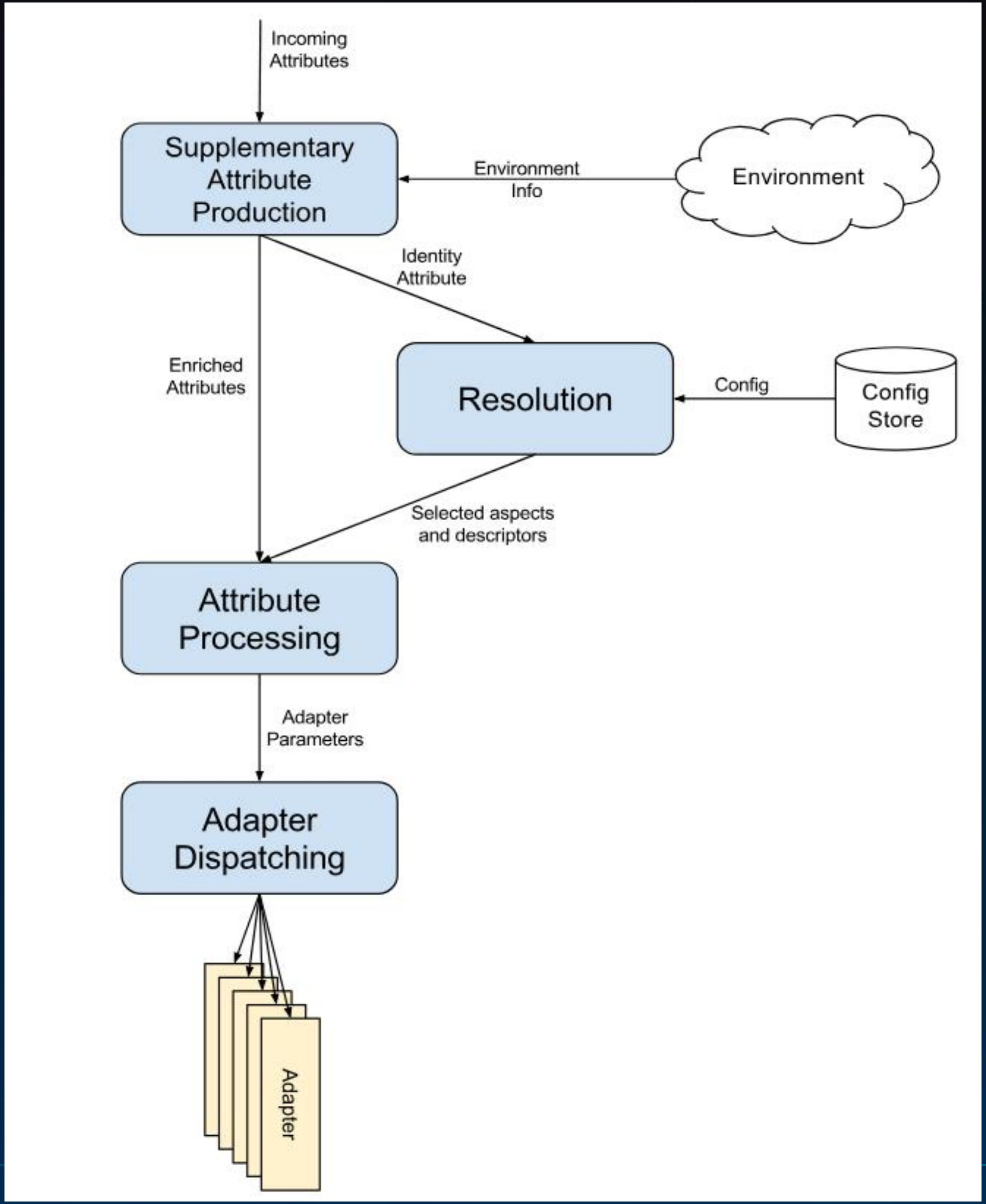
## Mixer 配置模型3：规则（Rule）

```
apiVersion: "config.istio.io/v1alpha2"
kind: rule
metadata:
  name: stdio
  namespace: {{ .Release.Namespace }}
spec:
  match: context.protocol == "http" || context.protocol == "grpc"
  actions:
  - handler: handler.stdio
    instances:
    - accesslog.logentry
```

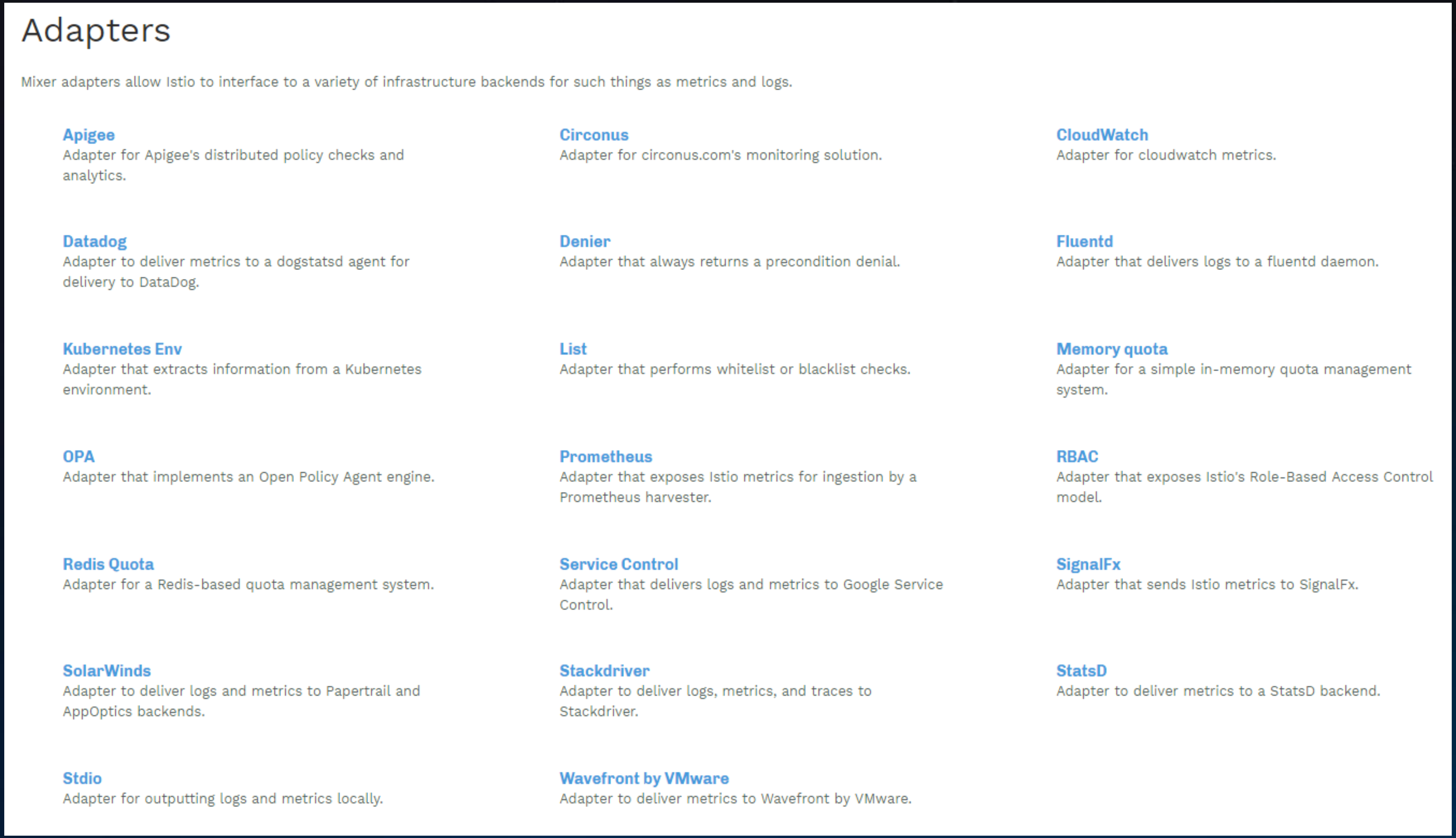
告诉 Mixer 哪个 instance 在什么时候发送给哪个 handler来处理

# Request的属性处理流程

接收属性
补充属性，
处理属性



## Mixer Adapters



### Adapters

Mixer adapters allow Istio to interface to a variety of infrastructure backends for such things as metrics and logs.

**Apigee**
Adapter for Apigee's distributed policy checks and analytics.

**Circonus**
Adapter for circonus.com's monitoring solution.

**CloudWatch**
Adapter for cloudwatch metrics.

**Datadog**
Adapter to deliver metrics to a dogstatsd agent for delivery to DataDog.

**Denier**
Adapter that always returns a precondition denial.

**Fluentd**
Adapter that delivers logs to a fluentd daemon.

**Kubernetes Env**
Adapter that extracts information from a Kubernetes environment.

**List**
Adapter that performs whitelist or blacklist checks.

**Memory quota**
Adapter for a simple in-memory quota management system.

**OPA**
Adapter that implements an Open Policy Agent engine.

**Prometheus**
Adapter that exposes Istio metrics for ingestion by a Prometheus harvester.

**RBAC**
Adapter that exposes Istio's Role-Based Access Control model.

**Redis Quota**
Adapter for a Redis-based quota management system.

**Service Control**
Adapter that delivers logs and metrics to Google Service Control.

**SignalFx**
Adapter that sends Istio metrics to SignalFx.

**SolarWinds**
Adapter to deliver logs and metrics to Papertrail and AppOptics backends.

**Stackdriver**
Adapter to deliver logs, metrics, and traces to Stackdriver.

**StatsD**
Adapter to deliver metrics to a StatsD backend.

**Stdio**
Adapter for outputting logs and metrics locally.

**Wavefront by VMware**
Adapter to deliver metrics to Wavefront by VMware.

https://istio.io/docs/reference/config/policy-and-telemetry/adapters/

## Mixer 的 Check Adapter

mixer/adapter/list/

### Adapter实现

```go
func (h *handler) HandleListEntry(_ context.Context, entry
*listentry.Instance) (adapter.CheckResult, error) {
   found, err := l.checkList(entry.Value)
   code := rpc.OK
   msg := ""

   if err != nil {
      code = rpc.INVALID_ARGUMENT

   } else if h.config.Blacklist {
      if found {
         code = rpc.PERMISSION_DENIED
         msg = fmt.Sprintf("%s is blacklisted", entry.Value)
      }
   } else if !found {
      code = rpc.NOT_FOUND
      msg = fmt.Sprintf("%s is not whitelisted", entry.Value)
   }

   return adapter.CheckResult{
      Status:        status.WithMessage(code, msg),
      ValidDuration: h.config.CachingInterval,
      ValidUseCount: h.config.CachingUseCount,
   }, nil
}
```

### Adapter配置定义

```
// Configuration format for the `list` adapter.
message Params {
   ...
    // Determines the type of list that the adapter is consulting.
    enum ListEntryType {
        // List entries are treated as plain strings.
        STRINGS = 0;
        // List entries are treated as case-insensitive strings.
        CASE_INSENSITIVE_STRINGS = 1;
        // List entries are treated as IP addresses and ranges.
        IP_ADDRESSES = 2;
        // List entries are treated as re2 regexp. See
[here](https://github.com/google/re2/wiki/Syntax) for the supported syntax.
        REGEX = 3;
    }

    // Determines the kind of list entry and overrides.
    ListEntryType entry_type = 7;

    // Whether the list operates as a blacklist or a whitelist.
    bool blacklist = 8;
}
```

istio.io\istio\mixer\adapter\list

# Mixer 的 Report Adapter

## Adapter实现

```go
func (h *handler) HandleLogEntry(_ context.Context, instances
[]*logentry.Instance) error {
   var errors *multierror.Error

   fields := make([]zapcore.Field, 0, 6)
   for _, instance := range instances {
      entry := zapcore.Entry{
         LoggerName: instance.Name,
         Level:      h.mapSeverityLevel(instance.Severity),
         Time:       instance.Timestamp,
      }

      for _, varName := range h.logEntryVars[instance.Name] {
         if value, ok := instance.Variables[varName]; ok {
            fields = append(fields, zap.Any(varName, value))
         }
      }

      if err := h.write(entry, fields); err != nil {
         errors = multierror.Append(errors, err)
      }
      fields = fields[:0]
   }

   return errors.ErrorOrNil()
}
```

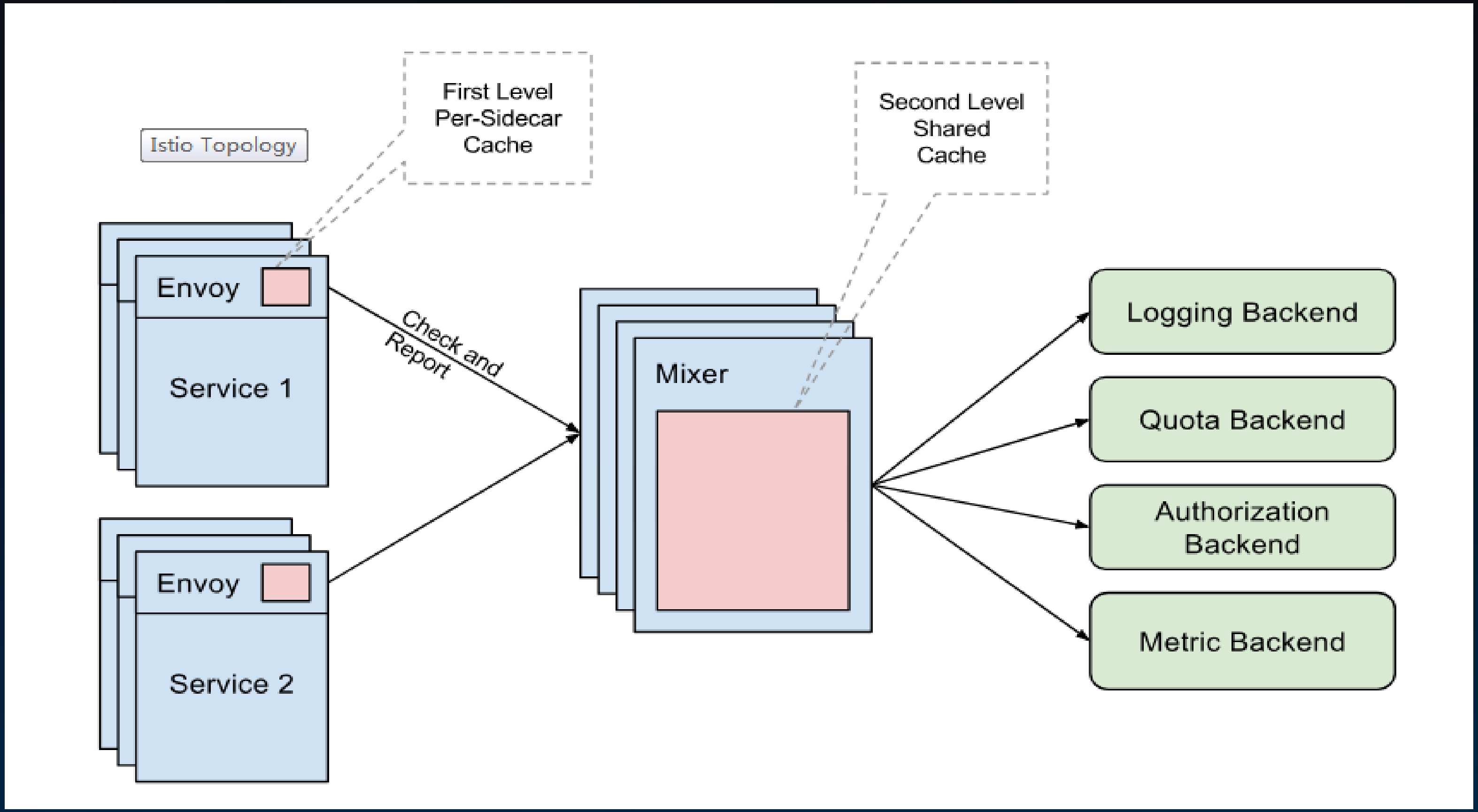
## Adapter配置定义

```
message Params {

..

    // Whether to output a console-friendly or json-friendly format.
Defaults to true.
    bool output_as_json = 4;

    // The minimum level to output, anything less than this level is
ignored. Defaults to INFO (everything).
    Level output_level = 5;

    // The file system path when outputting to a file or rotating file.
    string output_path = 6;

    // The maximum size in megabytes of a log file before it gets
    // rotated. It defaults to 100 megabytes.
    int32 max_megabytes_before_rotation = 7;
    int32 max_days_before_rotation = 8;
    int32 max_rotated_files = 9;
}
```

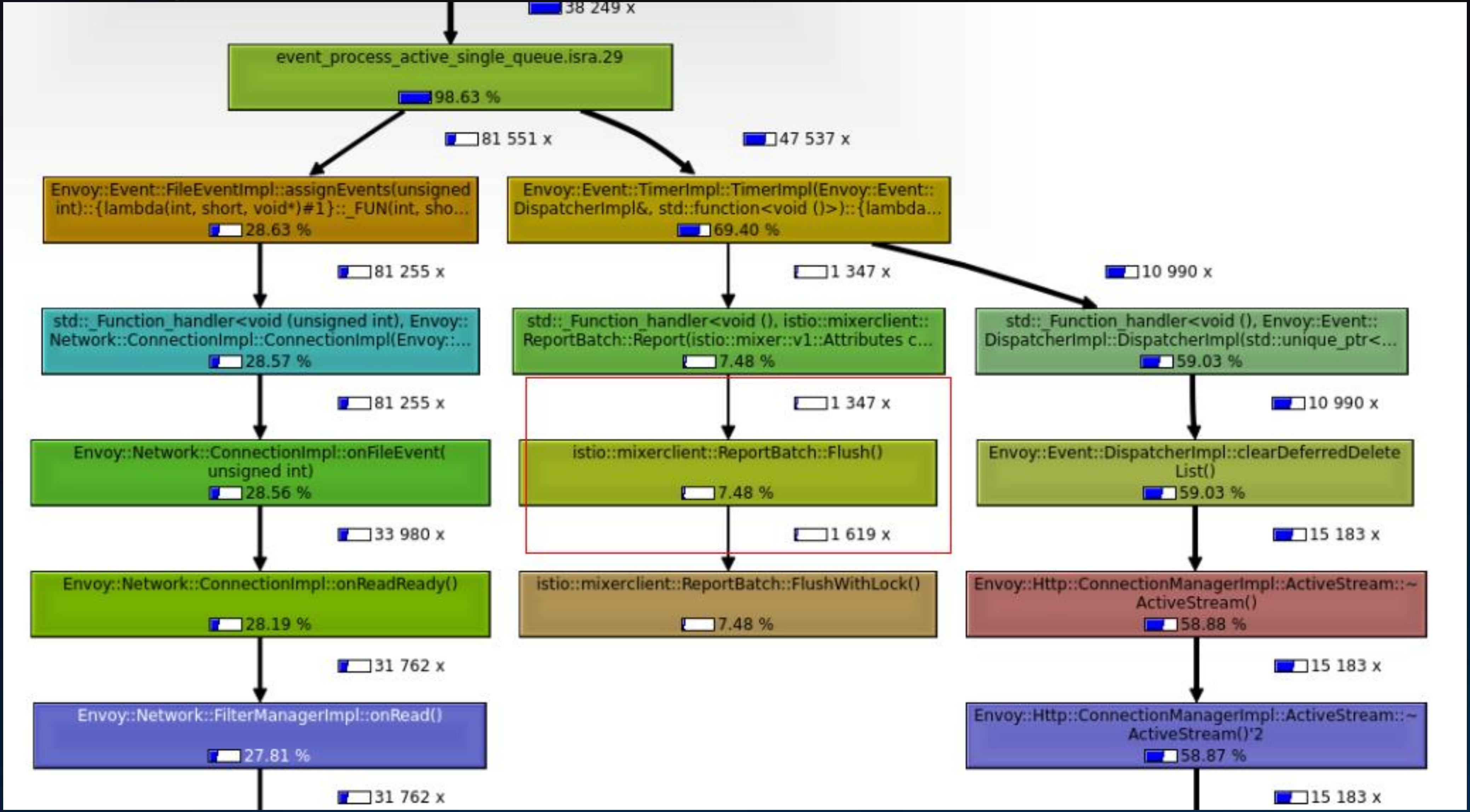
istio.io\istio\mixer\adapter\stdio

# Mixer的高可用设计

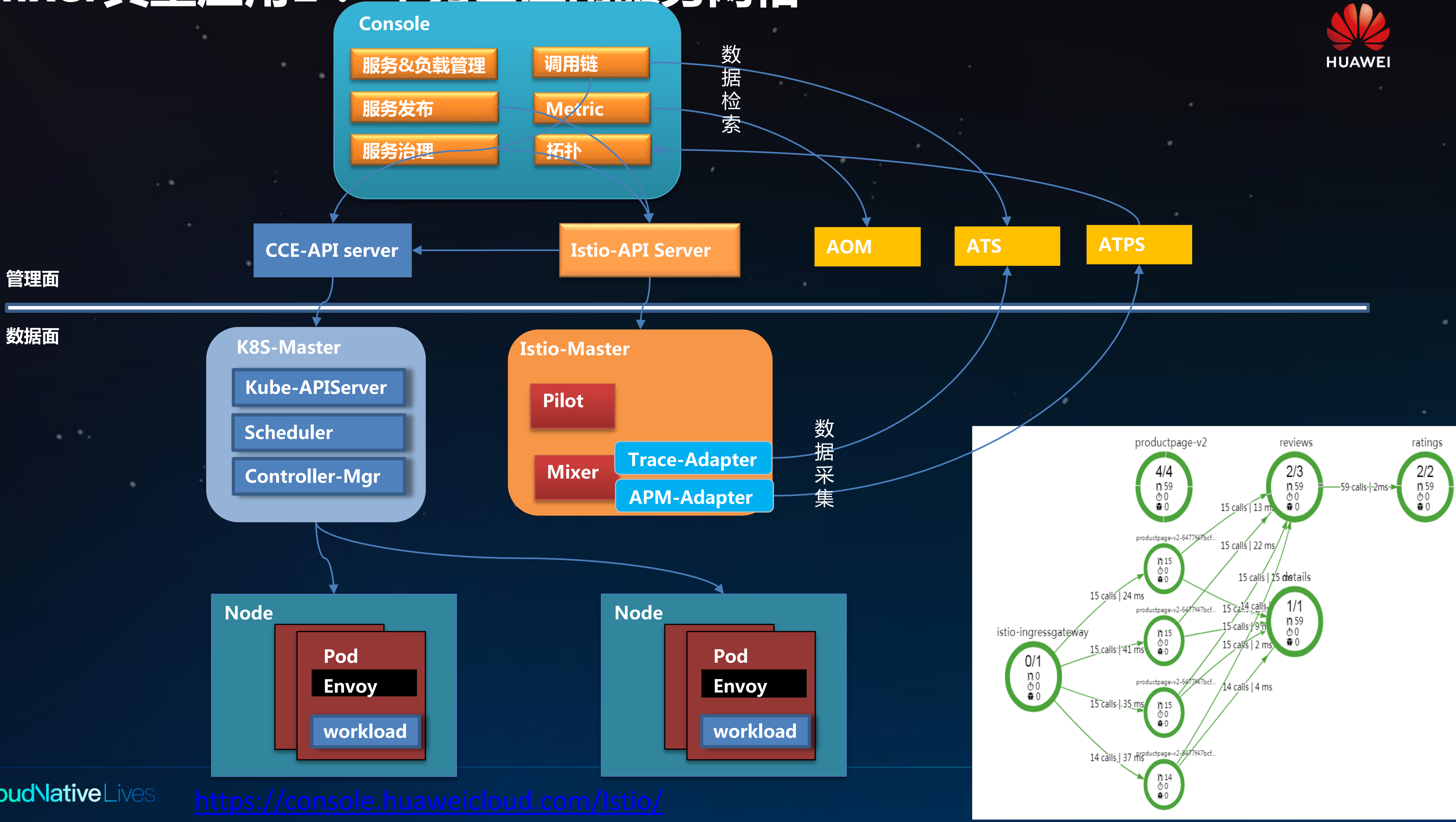
- 无状态
- 高可用以用
- 缓存和缓冲



# Mixer 的 Batch Report


38 249 x
event_process_active_single_queue.isra.29
98.63 %
81 551 x
47 537 x
Envoy::Event::FileEventImpl::assignEvents(unsigned int)::{lambda(int, short, void*)#1}::_FUN(int, sho...
28.63 %
Envoy::Event::TimerImpl::TimerImpl(Envoy::Event::DispatcherImpl&, std::function<void ()>)::{lambda...
69.40 %
81 255 x
1 347 x
10 990 x
std::_Function_handler<void (unsigned int), Envoy::Network::ConnectionImpl::ConnectionImpl(Envoy::...
28.57 %
std::_Function_handler<void (), istio::mixerclient::ReportBatch::Report(istio::mixer::v1::Attributes c...
7.48 %
std::_Function_handler<void (), Envoy::Event::DispatcherImpl::DispatcherImpl(std::unique_ptr<...
59.03 %
81 255 x
1 347 x
10 990 x
Envoy::Network::ConnectionImpl::onFileEvent(unsigned int)
28.56 %
istio::mixerclient::ReportBatch::Flush()
7.48 %
Envoy::Event::DispatcherImpl::clearDeferredDeleteList()
59.03 %
33 980 x
1 619 x
15 183 x
Envoy::Network::ConnectionImpl::onReadReady()
28.19 %
istio::mixerclient::ReportBatch::FlushWithLock()
7.48 %
Envoy::Http::ConnectionManagerImpl::ActiveStream::~ActiveStream()
58.88 %
31 762 x
15 183 x
Envoy::Network::FilterManagerImpl::onRead()
27.81 %
Envoy::Http::ConnectionManagerImpl::ActiveStream::~ActiveStream()'2
58.87 %
31 762 x
15 183 x

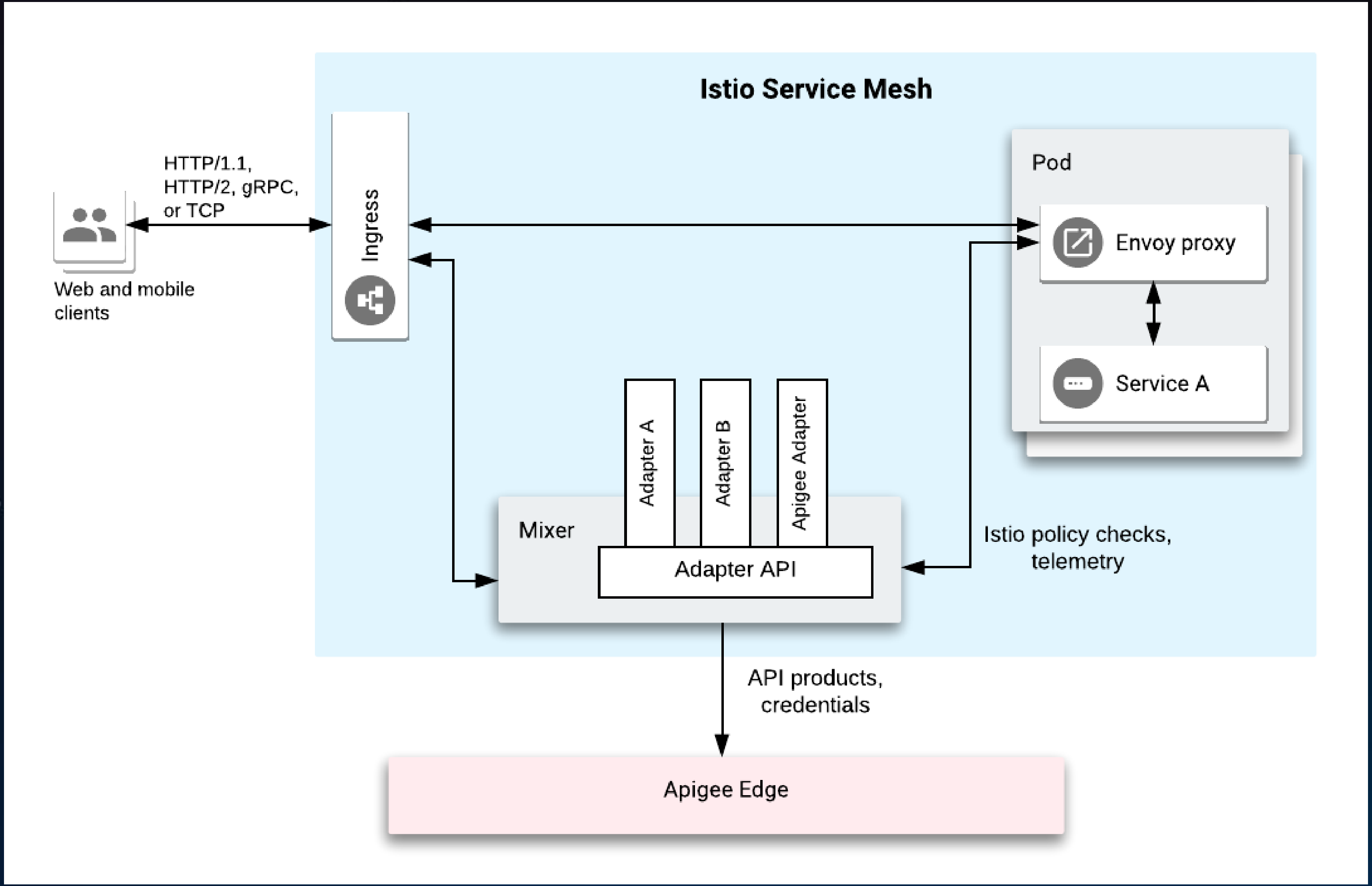
# Mixer典型应用1： 华为云应用服务网格



CloudNativeLives

https://console.huaweicloud.com/Istio/

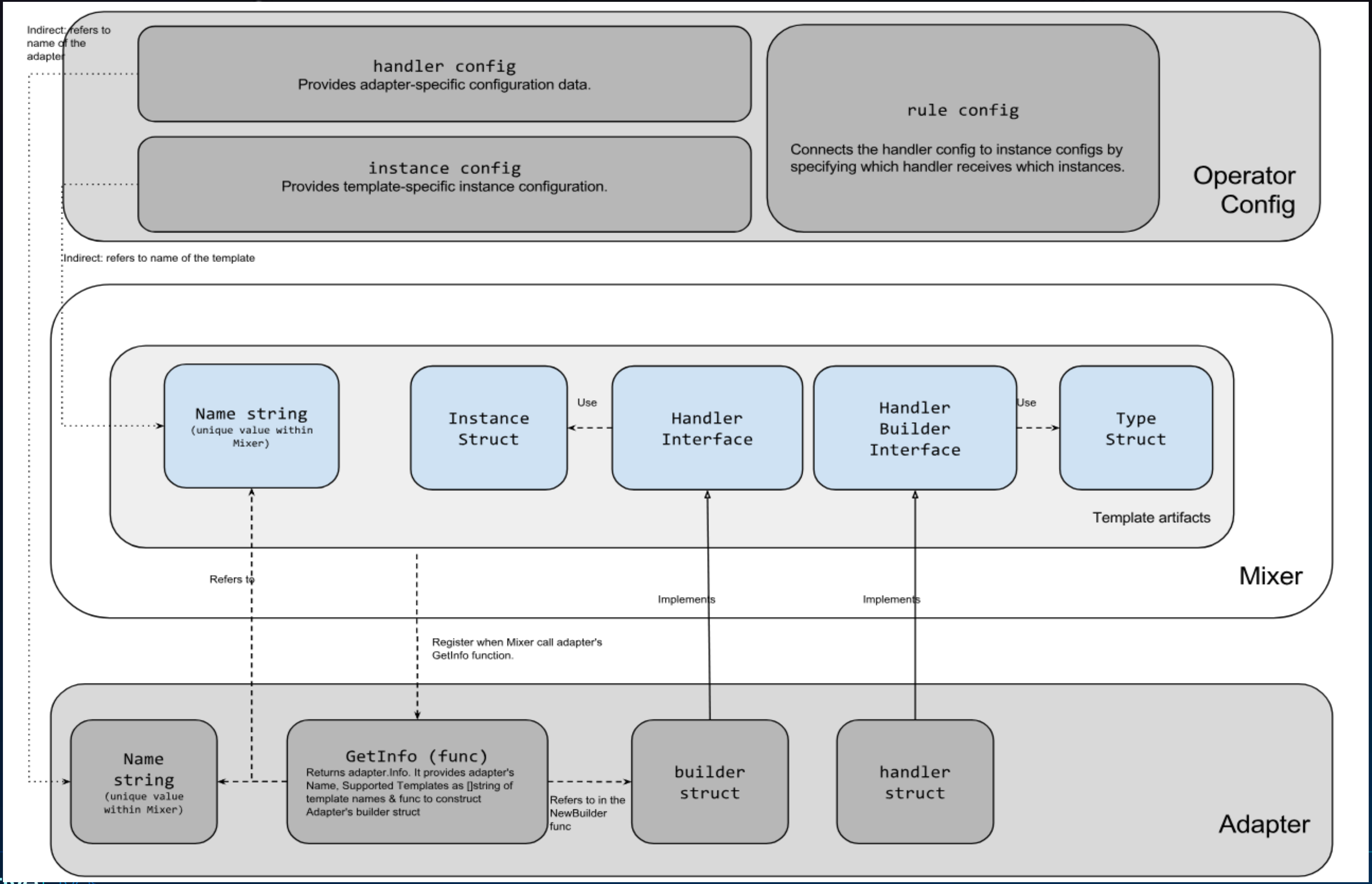
# Mixer典型应用2： Google Apigee



https://docs.apigee.com/api-platform/istio-adapter/concepts

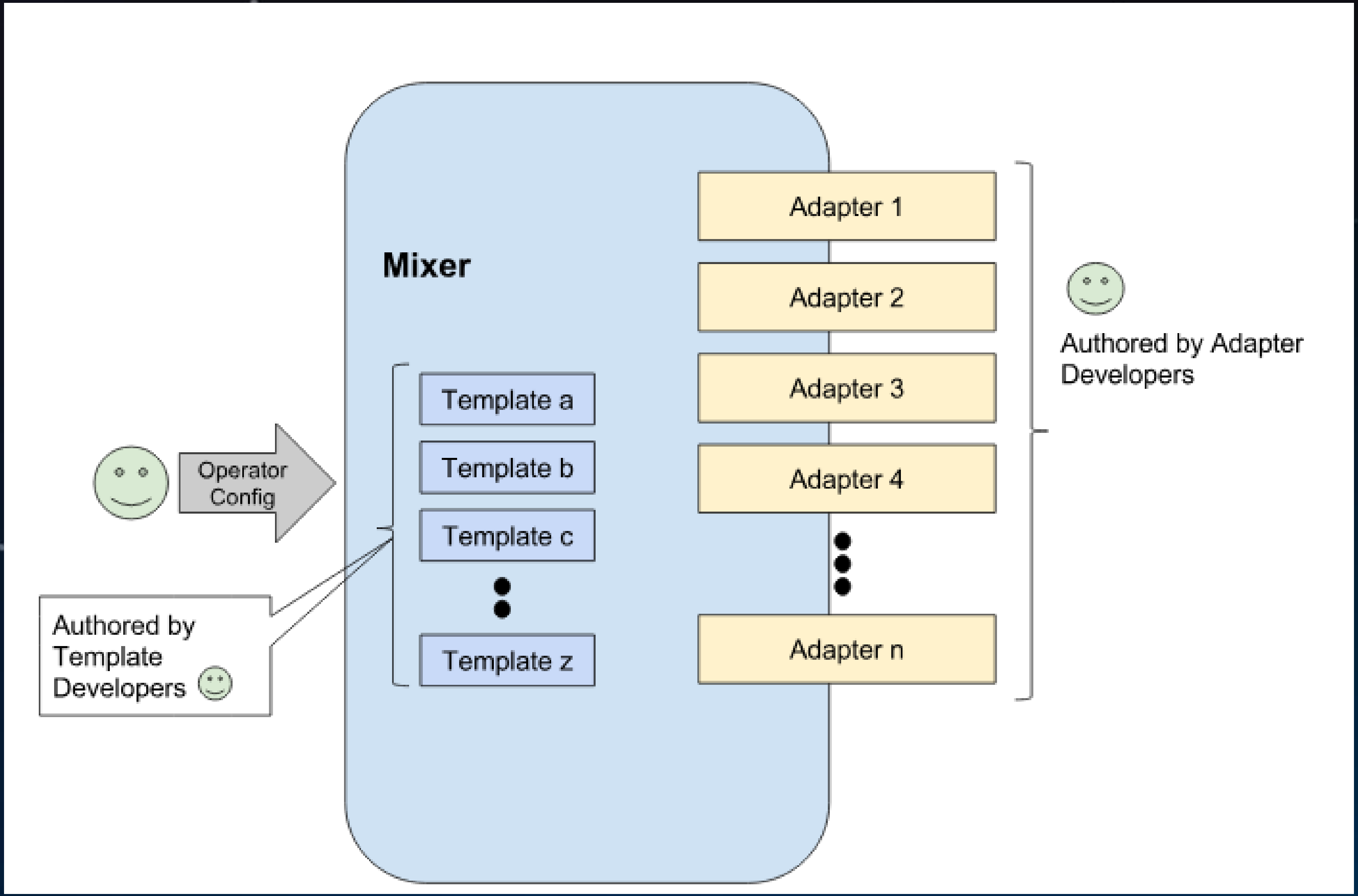
https://istio.io/docs/reference/config/policy-and-telemetry/templates/analytics/

# 实践1 从0开发并运行一个Mixer Adapter：原理


Indirect: refers to name of the adapter
handler config
Provides adapter-specific configuration data.
instance config
Provides template-specific instance configuration.
rule config
Connects the handler config to instance configs by specifying which handler receives which instances.
Operator Config
Indirect: refers to name of the template
Name string (unique value within Mixer)
Instance Struct
Use
Handler Interface
Handler Builder Interface
Use
Type Struct
Template artifacts
Mixer
Refers to
Implements
Implements
Register when Mixer call adapter's GetInfo function.
Name string (unique value within Mixer)
GetInfo (func)
Returns adapter.Info. It provides adapter's Name, Supported Templates as []string of template names & func to construct Adapter's builder struct
Refers to in the NewBuilder func
builder struct
handler struct
Adapter

# 实践1 从0开发并运行一个Mixer Adapter：两个角色

配置模板使用一个Adapter



开发代码定义模板开发一个Adapter

# 实践1 从0开发并运行一个Mixer Adapter：步骤

1.创建独立的Adapter目录，并开发Adapter的代码开发Adapter代码
cd $MIXER_REPO/adapter && mkdir mysampleadapter && cd mysampleadapter
#创建mysampleadapter .go文件定义处理逻辑

2. //配置config.proto，描述配置的定义。#创建config.proto文件，描述adapter的配置参数
mkdir config

3. 根据proto生成go的配置，并在adapter代码中使用
go generate ./...
go build ./...

4.在Mixer中注册这个新的Adapter。# 在inventory.yaml 中注册adapter，
**mysampleadapter: "istio.io/istio/mixer/adapter/mysampleadapter"**
go generate $MIXER_REPO/adapter/doc.go

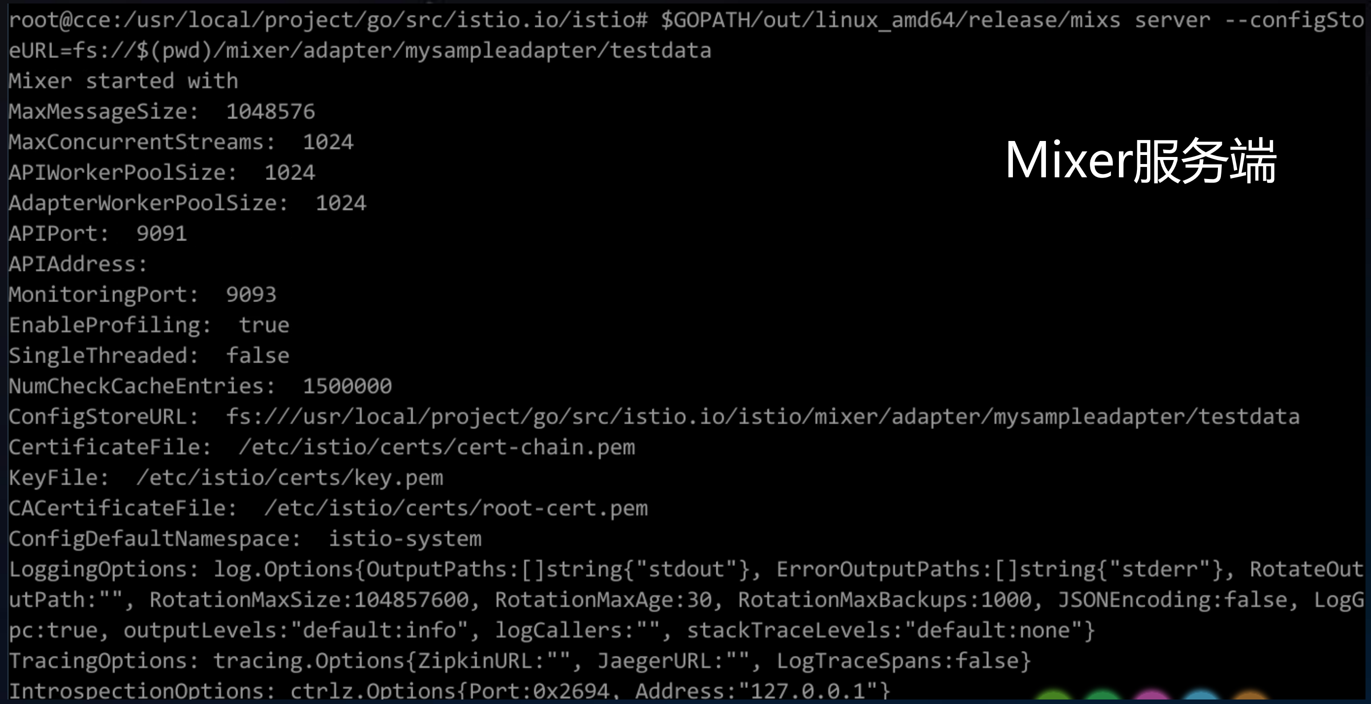
5. 配置并使用新创建的adapter。 #在testdata目录下创建使用该adapter的配置，即handler，instance，rulle。
mkdir $MIXER_REPO/adapter/mysampleadapter/testdata
#确认两个文件attributes.yaml和mysampleadapter.yaml

5. 启动mixer 服务端
pushd $ISTIO/istio && make mixs
$GOPATH/out/linux_amd64/release/mixs server --configStoreURL=fs://$(pwd)/mixer/adapter/mysampleadapter/testdata

6.启动一个客户端，模拟上报数据
pushd $ISTIO/istio && make mixc
$GOPATH/out/linux_amd64/release/mixc report -s destination.service="svc.cluster.local" -t request.time="2019-01-10T20:00:00Z"

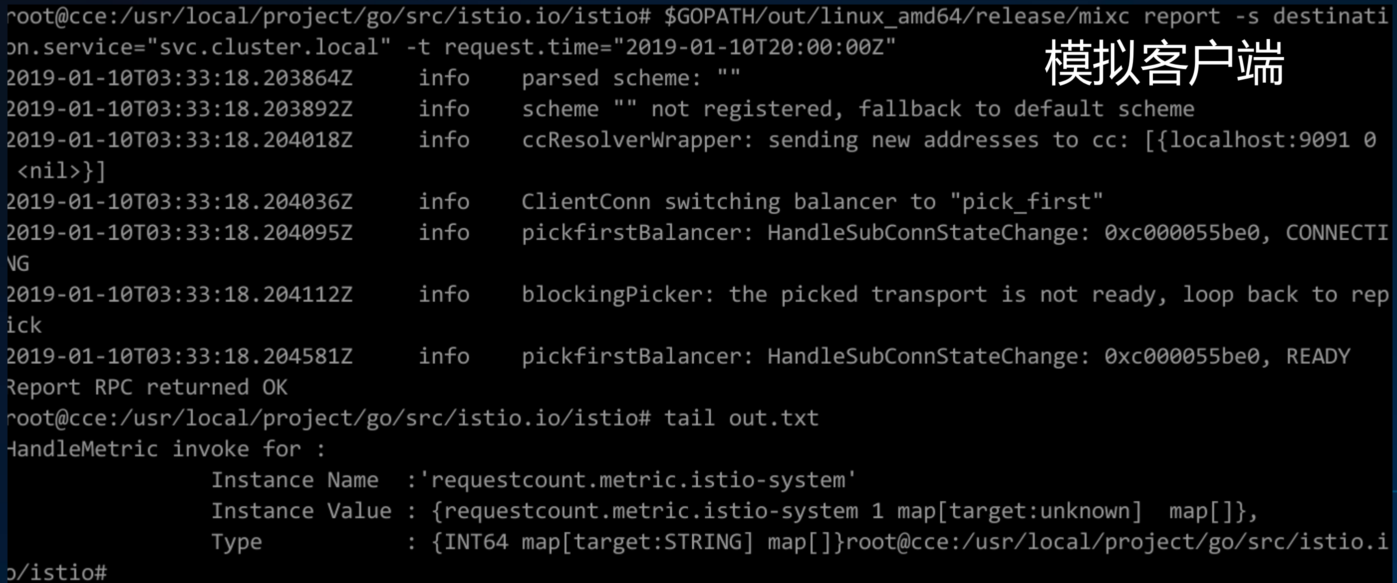
7.查看结果输出
tail $ISTIO/istio/out.txt

# 实践1 从0开发并运行一个Mixer Adapter：效果



Mixer服务端

```
root@cce:/usr/local/project/go/src/istio.io/istio# $GOPATH/out/linux_amd64/release/mixs server --configSto
eURL=fs://$(pwd)/mixer/adapter/mysampleadapter/testdata
Mixer started with
MaxMessageSize:  1048576
MaxConcurrentStreams:  1024
APIWorkerPoolSize:  1024
AdapterWorkerPoolSize:  1024
APIPort:  9091
APIAddress:
MonitoringPort:  9093
EnableProfiling:  true
SingleThreaded:  false
NumCheckCacheEntries:  1500000
ConfigStoreURL:  fs:///usr/local/project/go/src/istio.io/istio/mixer/adapter/mysampleadapter/testdata
CertificateFile:  /etc/istio/certs/cert-chain.pem
KeyFile:  /etc/istio/certs/key.pem
CACertificateFile:  /etc/istio/certs/root-cert.pem
ConfigDefaultNamespace:  istio-system
LoggingOptions: log.Options{OutputPaths:[]string{"stdout"}, ErrorOutputPaths:[]string{"stderr"}, RotateOut
utPath:"", RotationMaxSize:104857600, RotationMaxAge:30, RotationMaxBackups:1000, JSONEncoding:false, LogG
pc:true, outputLevels:"default:info", logCallers:"", stackTraceLevels:"default:none"}
TracingOptions: tracing.Options{ZipkinURL:"", JaegerURL:"", LogTraceSpans:false}
IntrospectionOptions: ctrlz.Options{Port:0x2694, Address:"127.0.0.1"}
```



模拟客户端

```
root@cce:/usr/local/project/go/src/istio.io/istio# $GOPATH/out/linux_amd64/release/mixc report -s destinati
on.service="svc.cluster.local" -t request.time="2019-01-10T20:00:00Z"
2019-01-10T03:33:18.203864Z     info    parsed scheme: ""
2019-01-10T03:33:18.203892Z     info    scheme "" not registered, fallback to default scheme
2019-01-10T03:33:18.204018Z     info    ccResolverWrapper: sending new addresses to cc: [{localhost:9091 0
 <nil>}]
2019-01-10T03:33:18.204036Z     info    ClientConn switching balancer to "pick_first"
2019-01-10T03:33:18.204095Z     info    pickfirstBalancer: HandleSubConnStateChange: 0xc000055be0, CONNECTI
NG
2019-01-10T03:33:18.204112Z     info    blockingPicker: the picked transport is not ready, loop back to rep
ick
2019-01-10T03:33:18.204581Z     info    pickfirstBalancer: HandleSubConnStateChange: 0xc000055be0, READY
Report RPC returned OK
root@cce:/usr/local/project/go/src/istio.io/istio# tail out.txt
HandleMetric invoke for :
                Instance Name  :'requestcount.metric.istio-system'
                Instance Value : {requestcount.metric.istio-system 1 map[target:unknown]  map[]},
                Type           : {INT64 map[target:STRING] map[]}root@cce:/usr/local/project/go/src/istio.i
o/istio#
```

详见演示...

# 实践2 通过Mixer收集自定义的遥测数据：目标

HUAWEI

- 编写自定义的Metric模板
- 在Istio中创建自定义Metric、Prometheus Handler和Rule
- 认识Prometheus Adapter
- 实践Prometheus 的主要能力

ISTIO + Prometheus

# 实践2 通过Mixer收集自定义的遥测数据：步骤

--1. 创建配置，包括prometheus的handler、metric和rule
kubectl apply -f double-request.yaml

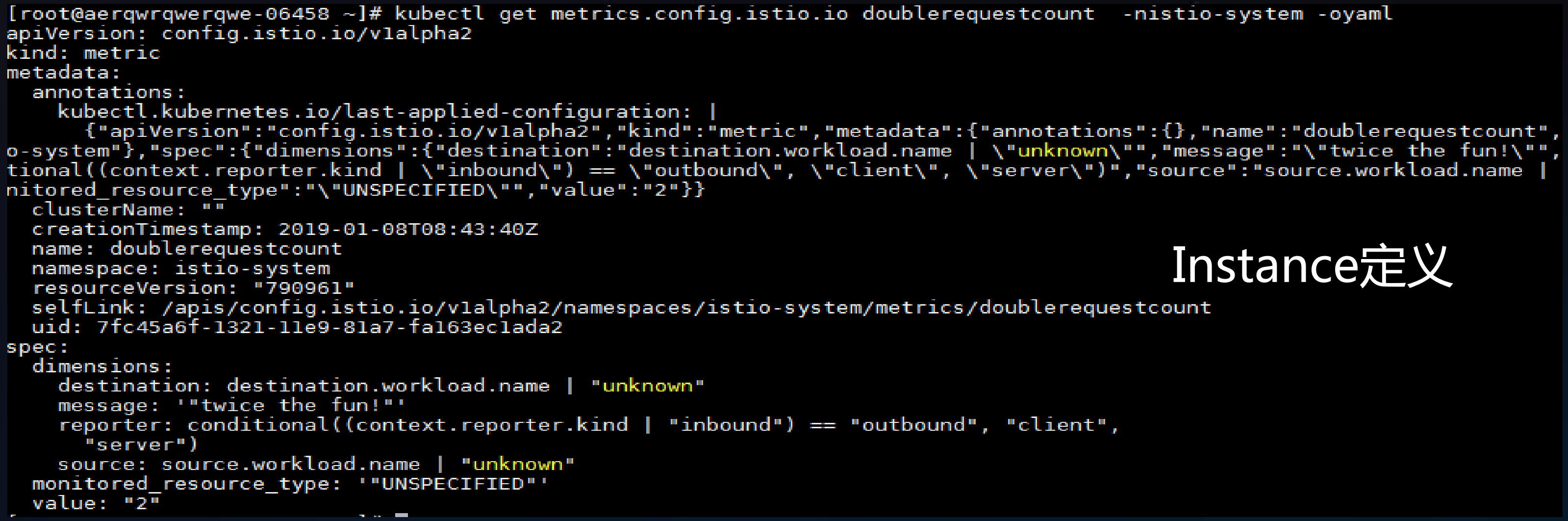
--2. 查看创建的对象
kubectl get metrics.config.istio.io  -nistio-system
kubectl get rules.config.istio.io  -nistio-system
kubectl get prometheus.config.istio.io  -nistio-system

-- 3. 发起对服务的访问，生成访问metric数据

--4.通过Prometheus查看metric数据
--4.1 查看doublereques的metric
http://49.4.84.29:9090/graph?g0.range_input=1h&g0.expr=istio_double_request_count&g0.tab=1

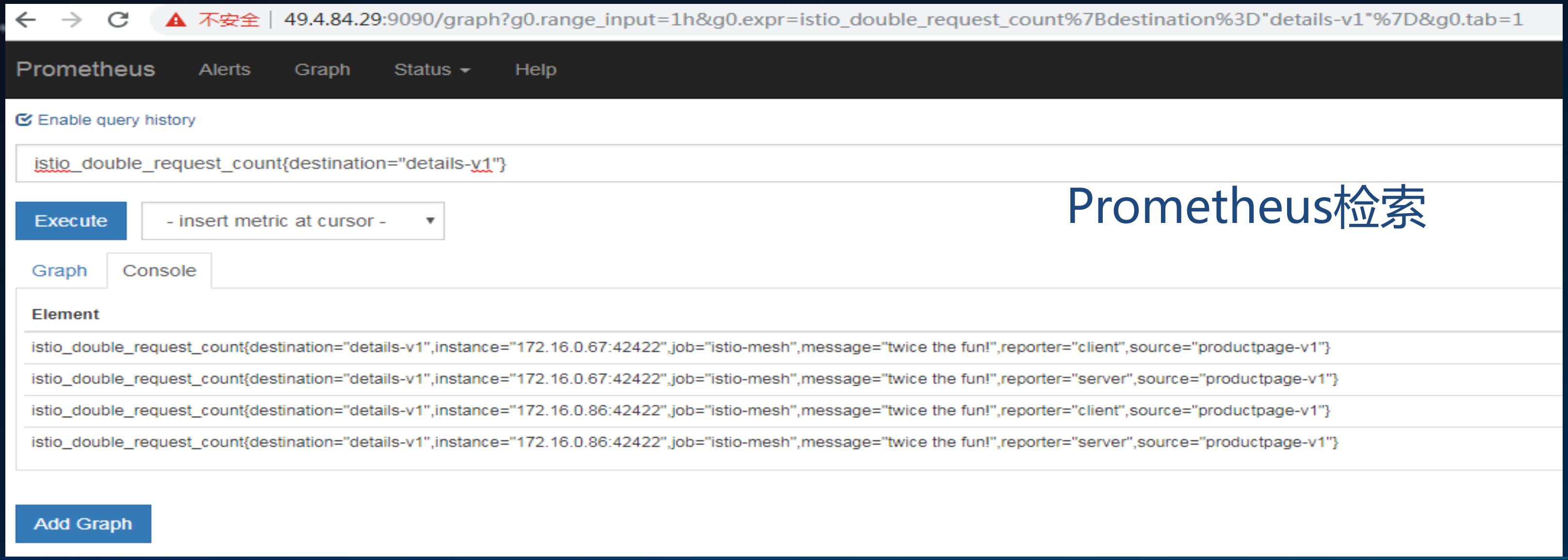
--4.2 通过prometheus检索特定目标的metric
istio_double_request_count{destination="details-v1"}

# 实践2 通过Mixer收集自定义的遥测数据：效果



```
[root@aerqwrqwerqwe-06458 ~]# kubectl get metrics.config.istio.io doublerequestcount  -nistio-system -oyaml
apiVersion: config.istio.io/v1alpha2
kind: metric
metadata:
  annotations:
    kubectl.kubernetes.io/last-applied-configuration: |
      {"apiVersion":"config.istio.io/v1alpha2","kind":"metric","metadata":{"annotations":{},"name":"doublerequestcount",
o-system"},"spec":{"dimensions":{"destination":"destination.workload.name | \"unknown\"","message":"\"twice the fun!\"",
tional((context.reporter.kind | \"inbound\") == \"outbound\", \"client\", \"server\")","source":"source.workload.name |
nitored_resource_type":"\"UNSPECIFIED\"","value":"2"}}
  clusterName: ""
  creationTimestamp: 2019-01-08T08:43:40Z
  name: doublerequestcount
  namespace: istio-system
  resourceVersion: "790961"
  selfLink: /apis/config.istio.io/v1alpha2/namespaces/istio-system/metrics/doublerequestcount
  uid: 7fc45a6f-1321-11e9-81a7-fa163ec1ada2
spec:
  dimensions:
    destination: destination.workload.name | "unknown"
    message: '"twice the fun!"'
    reporter: conditional((context.reporter.kind | "inbound") == "outbound", "client",
      "server")
    source: source.workload.name | "unknown"
  monitored_resource_type: '"UNSPECIFIED"'
  value: "2"
```

Instance定义



49.4.84.29:9090/graph?g0.range_input=1h&g0.expr=istio_double_request_count%7Bdestination%3D"details-v1"%7D&g0.tab=1

istio_double_request_count{destination="details-v1"}

Prometheus检索

```
Element
istio_double_request_count{destination="details-v1",instance="172.16.0.67:42422",job="istio-mesh",message="twice the fun!",reporter="client",source="productpage-v1"}
istio_double_request_count{destination="details-v1",instance="172.16.0.67:42422",job="istio-mesh",message="twice the fun!",reporter="server",source="productpage-v1"}
istio_double_request_count{destination="details-v1",instance="172.16.0.86:42422",job="istio-mesh",message="twice the fun!",reporter="client",source="productpage-v1"}
istio_double_request_count{destination="details-v1",instance="172.16.0.86:42422",job="istio-mesh",message="twice the fun!",reporter="server",source="productpage-v1"}
```

详见演示...

HUAWEI

# Thank You

直播 每周四 晚20:00