KubeCon | CloudNativeCon

North America 2017

Istio's Mixer:

Policy Enforcement with Custom Adapters

Limin Wang, Software Engineer, *Google*
Torin Sandall, Software Engineer, *Styra*

Outline

- Istio and policy (how to enforce your custom policy in Istio)
- Integrate Open Policy Agent to Istio (demo)

What is Istio?

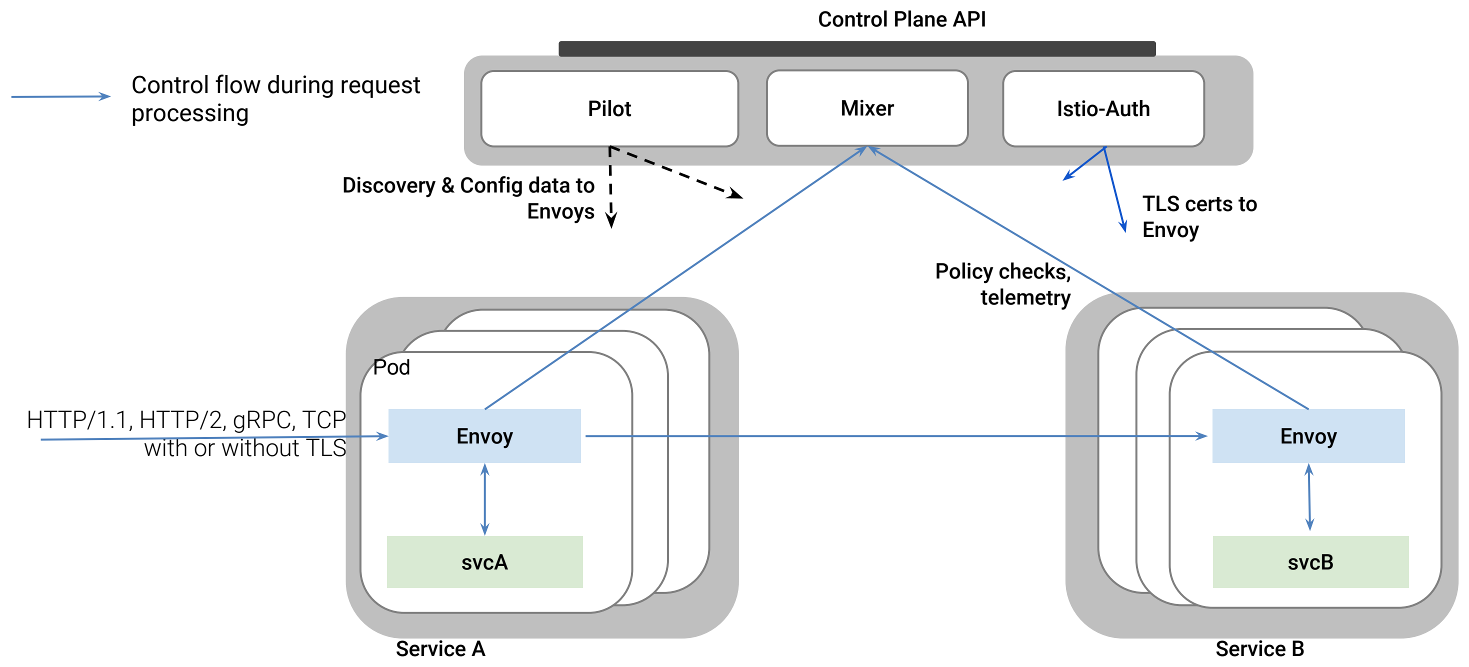
An open platform to connect, manage, secure microservices

- Istio provides:
  - Traffic management
  - Observation
  - Policy Enforcement
  - Service Identity and Security
  - And more …

istio.io

github.com/istio

Istio Architecture


Control Plane API
Control flow during request processing
Pilot
Mixer
Istio-Auth
Discovery & Config data to Envoys
TLS certs to Envoy
Policy checks, telemetry
Pod
HTTP/1.1, HTTP/2, gRPC, TCP with or without TLS
Envoy
svcA
Envoy
svcB
Service A
Service B

Policies in Istio

- Route rules
  - Load balancing, traffic splitting, request timeout, retry, fault injection
- Quota policies
- Monitoring policies
  - Metrics, logging, tracing
- Security policies
  - Service-to-service mTLS authentication
  - Simple authorization: denier, white/black list, expression language (ABAC)

Policies in Istio (cont.)

- Upcoming security policies
    - Authentication policy
        - Enable/disable mTLS per service
        - End user authentication
    - Authorization policy
        - Role Based Access Control (RBAC)
        - Open Policy Agent
        - Expression language with richer semantics
    - Audit policy

Example Policy (RBAC)

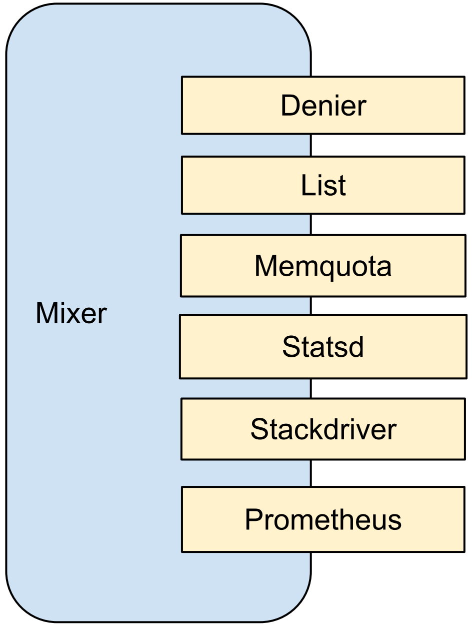
```
kind: ServiceRole
apiVersion: config.istio.io/v1alpha2
metadata:
  name: review-product-viewer
  namespace: default
spec:
   rules:
     - services: [“reviews”]
       methods: [“GET”, “HEAD”]
     - services: [“products”]
       paths: [“/books”, “/books/*”]
       methods: [“GET”, “HEAD”]
```

```
kind: ServiceRoleBinding
apiVersion: config.istio.io/v1alpha2
metadata:
  name: example-role-binding
  namespace: default
spec:
  subjects:
    - name: “istio-ingress-service-account”
  roleRef:
    kind: ServiceRole
    name: review-product-viewer
```

More information on Istio RBAC Design Doc.

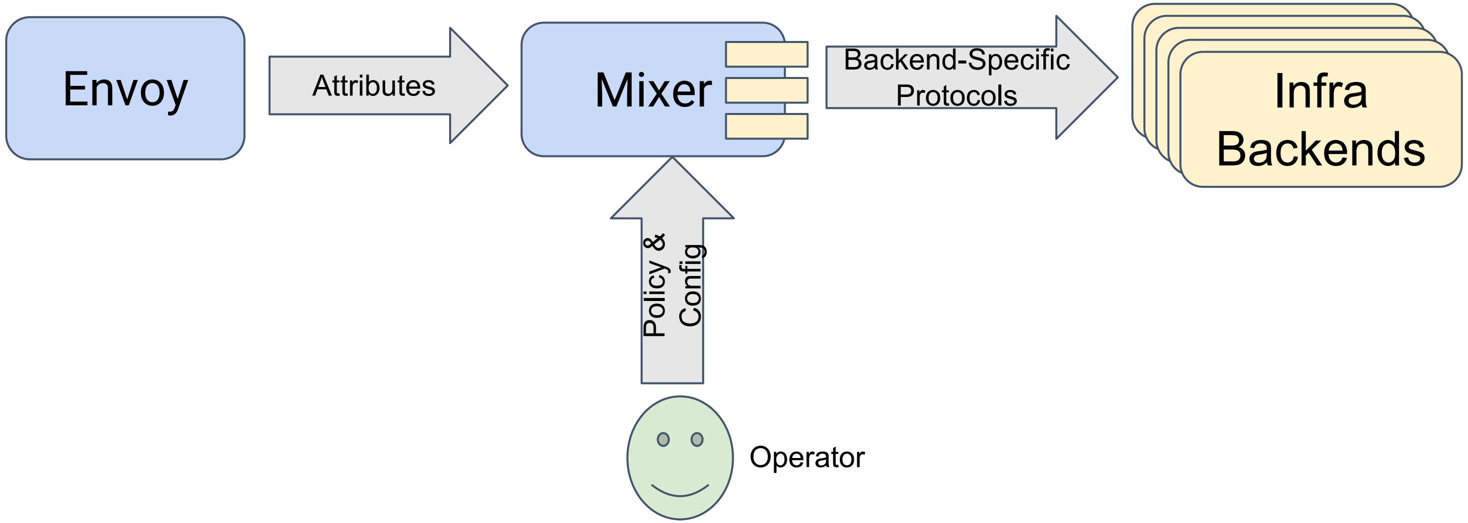
Extend Policy System through Mixer

- Mixer is the central point for policy evaluation and extensibility.
- Mixer provides the following core features:
    - Precondition and quota checking (Check)
    - Telemetry reporting (Report)
- Mixer achieves high extensibility by having a general purpose plug-in model - the plug-ins are known as ***Adapters***.



Mixer’s Adapters

- Mixer is an attribute-processing and routing machine.
  - Attributes => Instances => Adapters => (Backends)



How to Provide a Custom Adapter

- Determine your adapter type (check/quota/report)
- Determine the runtime input to your adapter
  - Template: adapter input schema
  - You can apply multiple templates
    - Built-in templates, or your custom templates
- Determine how to configure your adapter.
  - Handler: configured adapter
- Determine the business logic for your adapter to handle runtime input.

More information on https://github.com/istio/istio/blob/master/mixer/doc/adapters.md

Example: A Toy Adapter

Build an adapter to verif
built-in ListEntry adapte

- Adapter type: check
- Adapter input: built-in listEntry template
- Adapter configuration: a list of strings.
- How the adapter handles runtime input: looks up the value in a list of strings.

```
...
package listEntry;
option (istio.mixer.v1.template.template_variety) = TEMPLATE_VARIETY_CHECK;
message Template {
   // Specifies the entry to verify in the list.
   string value = 1;
}
```

Steps to Build a Custom Adapter

Step 1. Write basic adapter skeleton code (online tutorial or build-in adapters)

```
...
func GetInfo() adapter.Info {
  return adapter.Info{
     Name:        "listChecker",
     Description: "Checks whether a string is in the list",
     SupportedTemplates: []string{
        listentry.TemplateName,
     },
     NewBuilder:    func() adapter.HandlerBuilder { return &builder{} },
     DefaultConfig: &config.Params{},
  }
}
```

Steps to Build a Custom Adapter

Step 2. Write adapter configuration.

```
package adapter.listChecker.config;

message Params {

    repeated string list = 1;

}
```

Step 3. Validate adapter configuration.

```
func (b *builder) SetAdapterConfig(cfg adapter.Config) { b.conf = cfg.(*config.Params) }

func (b *builder) Validate() (ce *adapter.ConfigErrors) {
     // Check if the list is empty
     if b.conf.List == nil {

         ce = ce.Append(“list”, “list cannot be empty”)
     }
     return
}
```

Steps to Build a Custom Adapter

Step 4. Write business logic for your adapter.

```
func (b *builder) Build(context context.Context, env adapter.Env) (adapter.Handler, error)
{ return &handler{list: b.conf.List}, nil }

func (h *handler) HandleListEntry(ctx context.Context, inst *listentry.Instance) (adapter.CheckResult, error) {
   code := rpc.OK
   for _, str := range h.list {
      if inst.Value == str {
        code = rpc.NOT_FOUND
        break
      }
    }
   return adapter.CheckResult{
       Status: rpc.Status{Code: int32(code)},
  }, nil
}
```

Configure Policy Using Custom Adapter

1. Create an instance of listentry template.

```
apiVersion: “config.istio.io/v1alpha2”
kind: listentry
metadata:
  name: srcVersion
spec:
  value: source.labels[“version”]
```

2. Create a handler of listChecker adapter.

```
apiVersion: “config.istio.io/v1alpha2”
kind: listChecker
metadata:
  name: versionChecker
spec:
  list: [“v1”, “v2”]
```

3. Create a checkVersion policy

```
apiVersion: “config.istio.io/v1alpha2”
kind: rule
metadata:
  name: checkVersion
spec:
  match: destination.labels[“app”] == “ratings”
  actions:
  - handler: versionChecker.listChecker
    instances:
    - srcVersion.listentry
```

4. Apply the policy!

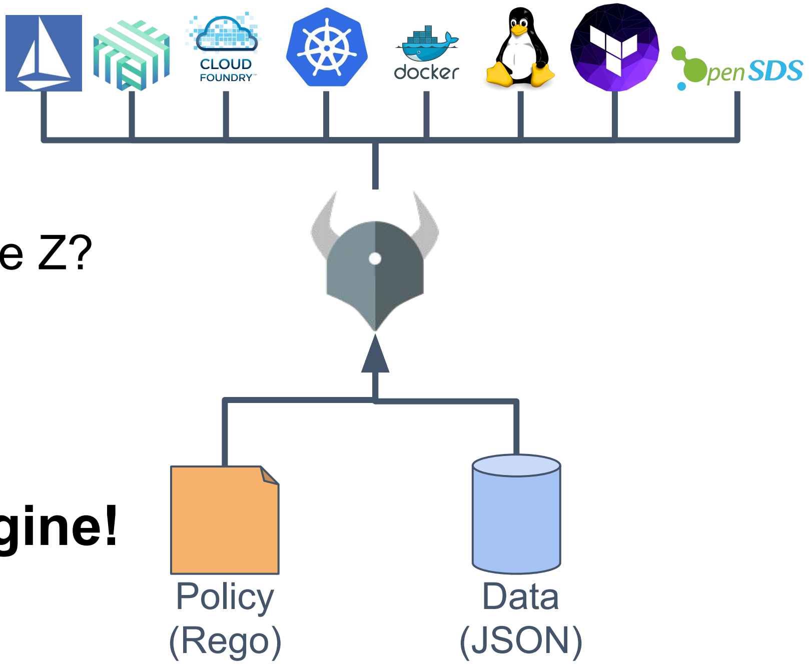
```
istioctl create -f *.yaml
```

- Overview: Open Policy Agent
- OPA Adapter
- Demo

+

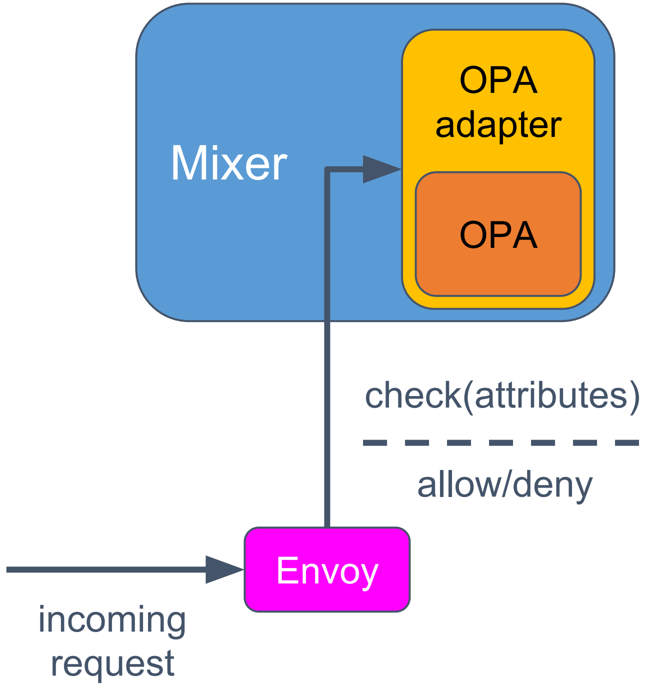
Open Policy Agent (OPA)

- **General-purpose policy engine**
  - Offload authorization decisions
- **Declarative Policy Language (Rego)**
  - Is X allowed to call operation Y on resource Z?
- **Library or Daemon**
  - In-memory policies and data
  - Zero runtime dependencies
  - Implemented in Go
- **Don’t roll your own authorization engine!**



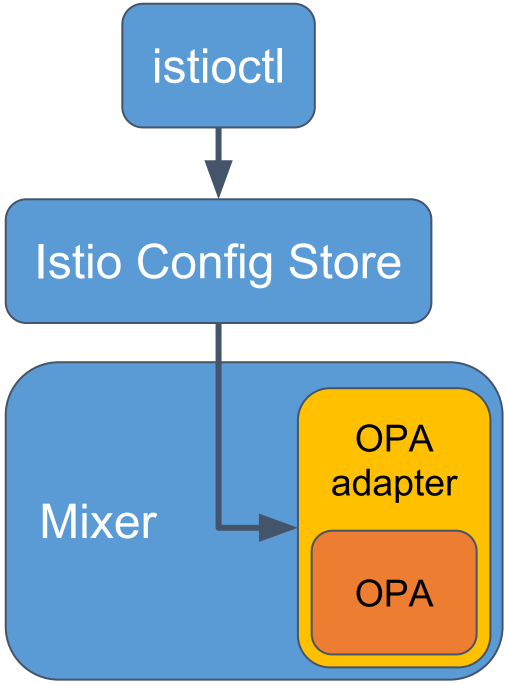
Mixer’s OPA Adapter

- Adapter type: **Check**
- Attributes: (authz template)
  - **Subject**: `map<string, value>`
  - **Action**: `map<string, value>`
- Standalone adapter
  - **No external dependencies**
- Fail closed (deny) in case of error(s)
  - To be configurable in future



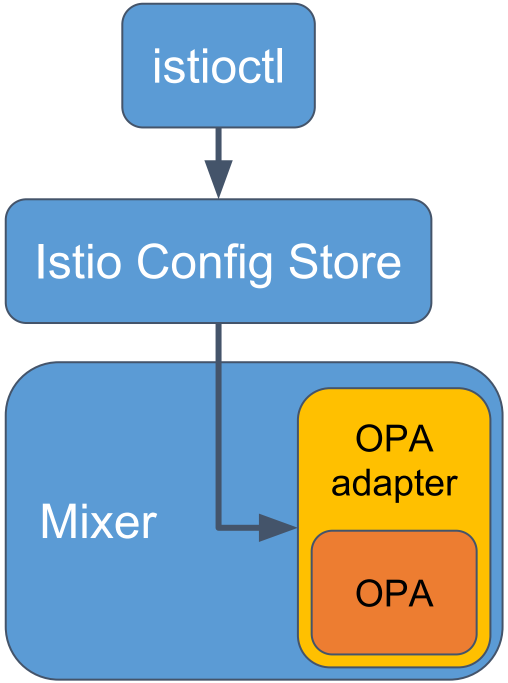
Mixer config (1/3): Rule

```
apiVersion: config.istio.io/v1alpha2
kind: rule
metadata:
  name: authz
spec:
  actions:
  - handler: opa-handler
    instances:
    - authz-instance
```



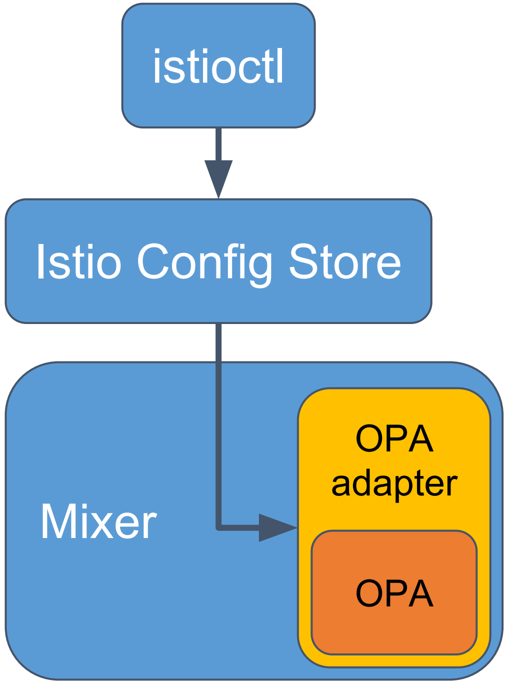
Mixer config (2/3): Instance

```
apiVersion: config.istio/v1alpha2
kind: authz
metadata:
  name: authz-instance
spec:
  subject:
    user: source.uid | “”
  action:
    namespace: target.namespace | “default”
    service: target.service | “”
    path: target.path | “”
    method: request.method | “”
```



Mixer config (3/3): Handler

```
apiVersion: config.istio.io/v1alpha2
kind: opa
metadata:
  name: opa-handler
spec:
  checkMethod: authz.allow
  policy: |
    package authz
    default allow = false
    allow { is_read }
    is_read { input.action.method = “GET” }
```



Demo

Conclusion

- **Use Istio to enforce wide range of policy across your microservices**

- **Plugin framework makes it easy to add adapters**
  - Authorization, quota, telemetry, …

- **Come join us!**
  - istio-users@googlegroups.com
  - Istio working groups (Security, Integrations, …)
  - More information: istio.io, github.com/istio

KubeCon | CloudNativeCon

North America 2017

Questions?