Mesh Configuration Protocol (MCP)

Authors: jasonyoung@google.com
Work-Group: config
Short self link: https://goo.gl/EJu5hg
Contributors: @andraxylia, @ozevren, @geeknoid
Reviewers:

Status: WIP | In Review | **Approved** | Obsolete
Created: 04/23/2018
Release Version: N/A
Approvers: mtail@google.com [x], ozben@google.com [x]

Objective

We need a cohesive story for how components receive config from Galley, with as much shared code and technology as possible across the product. Istio components include Mixer, Pilot, Citadel, and Pilot-agent. See Istio Config Architecture for a high-level overview.

Problem

- Pilot and Mixer components are tightly coupled to the underlying platform for configuration. e.g. k8s api-server is required. It's difficult to test components standalone. Extending and integrating into different environments requires per-component code changes.

- Validation, ACL'ing, versioning, and schema upgrades is non-existent and/or not consistent across components. These concerns should largely be decoupled from runtime components and handled prior to config distribution.

- Config distribution is too coarse. The overall config input space is potentially quite large. It's not possible to distribute a subset of config space to subsets of component instances.

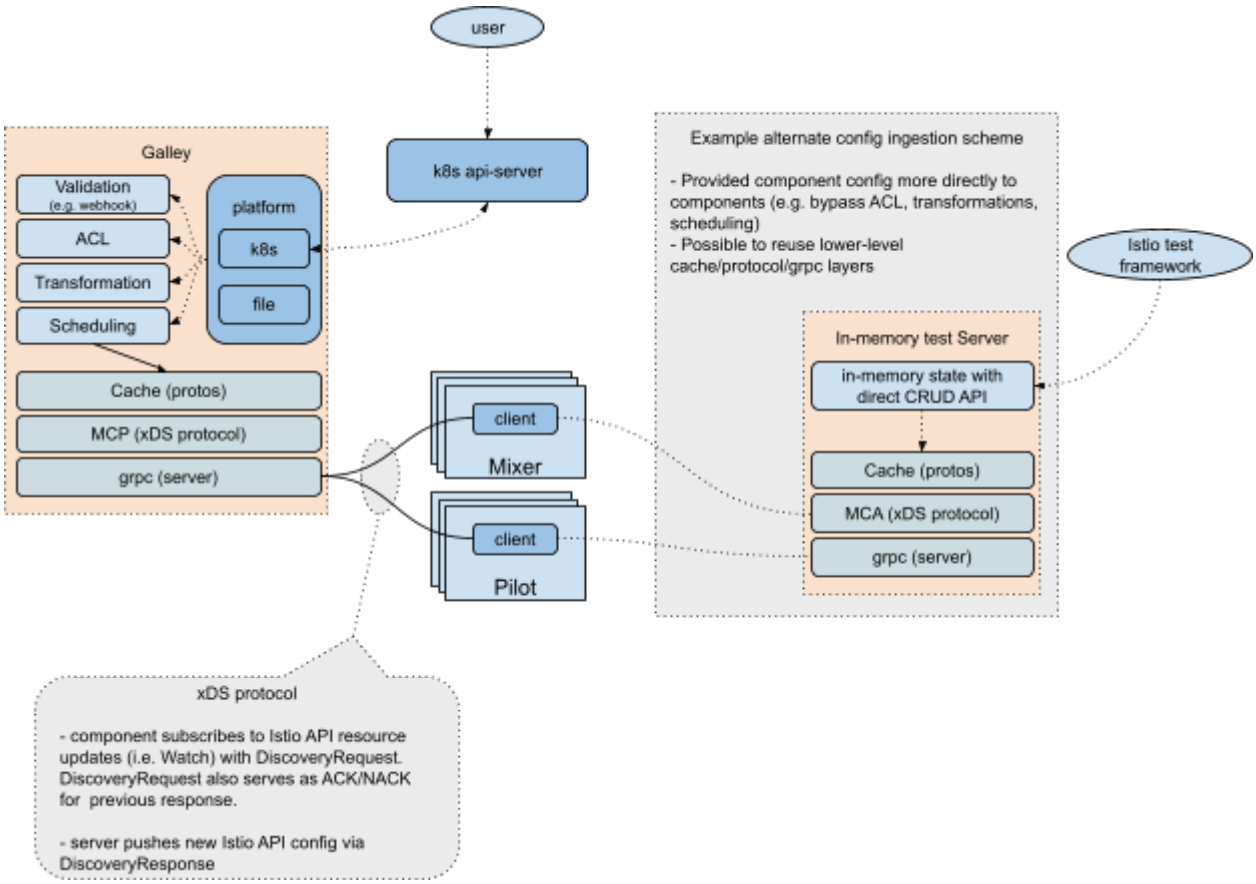
- Component config is spread across CRD, ConfigMaps, and binary flags. ConfigMaps and flag often require restarting components or pods to adopt new config. It's desirable to have one consistent approach to consume all istio runtime config.

- Component config consumption could be more efficient. K8s CRD are stored and transported as JSON. Components use protobuf at runtime. Each component incurs CPU/memory cost of YAML/JSON-to-Proto conversions. It's desirable to centralize conversions.

Non-Goals

- The user facing authoring model continues to depend on k8s api-server for configuration. This doc only addresses how components receive their configuration.

Proposal



- Use GRPC bidirectional streaming to push configuration updates to components. This client/server protocol is a subscription model based on Envoy's xDS gRPC protocol. We can give this a different name to avoid confusion, e.g. Mesh Configuration Protocol (MCP). Components subscribe to the relevant resource types.

- MCP carries existing Istio APIs defined in istio/api, e.g. istio/api/networking, istio/api/authentication, istio/api/mesh. Resources are transported as protobuf. Metadata (e.g. name) must be encoded in the resource directly as a submessage.

- Galley implements MCP server. Introduce adapter model whereby config ingestion is plugable. Support for CRD and file based watch are initially supported. Config ingested through adapter is mapped onto xDS watch subscription. Adapters can initially be in-process and alternatively moved out-of-process.

- Alternate MCP server implementations may be created to simplify testing and local development. Partners may also integrate with istio runtime components directly using MCP.

Migration path

A similar proposal has been written from the perspective of Pilot — see Decomposing Pilot. Mixer and other Istio components can gradually move to the new model. The eventual intent is for:

1. Istio components to be platform-agnostic, consuming Istio configuration through MCP.
2. Galley to provide the server implementation of MCP fully integrated with Kubernetes. Galley is responsible for config validation, enhancing ACL'ing, versioning, schema upgrades, etc.
3. Platform partners can integrate Pilot, Mixer, etc. using the same set of APIs (MCP)

It's desirable to move incrementally towards this new model. To that end, existing CRDs will be preserved and Galley will per-resource xDS subscription 1-to-1 with k8s CRDs. Galley can be optionally configured (via feature flag) to perform this function. Components can be optionally configured (via feature flag) to subscribe to configuration updates from Galley (vs. k8s directly).

Open Items

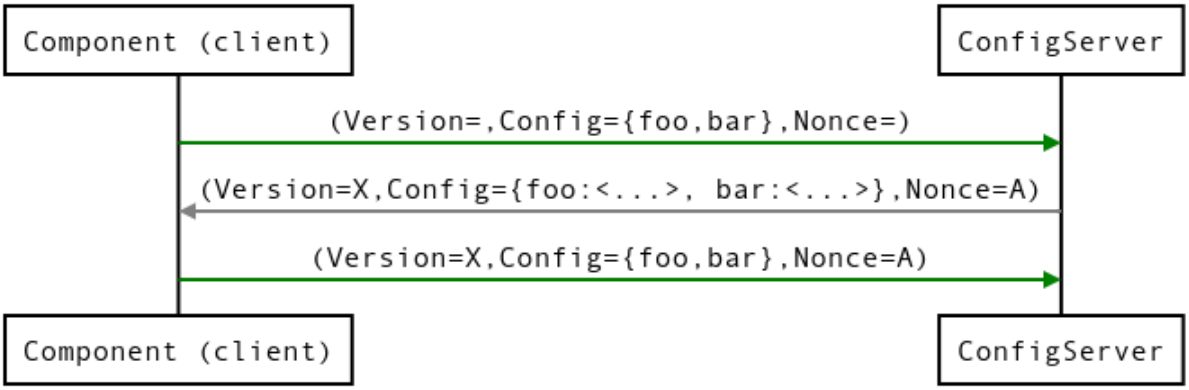
The following open items will be addressed incrementally.

- All Istio APIs need embedded metadata. Unit of xDS resource delivery is protobuf. Each protobuf must be fully self-describing. Current types lack metadata (e.g. name, namespace, labels).
- Declarative validation. Functional domains should be able to define validation schemas and have someone else (i.e. Galley) validate it. Istio API schemas should be annotated accordingly.
- Enhanced ACL model for Istio resources. e.g. networking API should be ACL'd by ownership of network name (host)
- Auth{n,z} between Galley and runtime components. Strawman: mTLS for authn. Separate policy discussion needed for authz

Appendix

(MCP) xDS protocol overview

Here's a brief summary of Envoy's xDS protocol applied to Istio config. Istio components are gRPC clients. Galley is the gRPC server. Components request config via subscriptions by initiating a gRPC stream and sending ConfigRequest protobuf. Config is delivered in ConfigResponse protobufs. "Push" semantics are achieved by clients always sending a ConfigRequest after receiving a ConfigResponse and Galley only sending ConfigResponse when new config is available.



Whereas Envoy has discovery services (EDS/CDS/RDS/LDS/etc) plus an aggregate service (ADS), the Istio config distribution service could be dimensioned by component type (Pilot, Mixer, Agent, etc.) or as a single aggregate service. See the xDS protocol docs for a more thorough explanation of the protocol and how ACK/NACKs and versioning are handled.

Alternatives considered

- Alternative design proposal - Galley Watch API design proposal
- Alternative prototype - https://github.com/ayj/istio/commit/45dc13f62cd1a

Use googleapis watcher API to stream declarative configuration data to components. Components watch a named entity that corresponds to their desired configuration state. The entity corresponds to a collection of elements to allow for composition and reuse across updates. In turn, components receive an ordered and reliable stream of configuration state.