Multi Source Config Model For Galley

Owner: mitchconnors@
Work-Group: usability, config

Status: WIP | **I**n Review | **Approved** | Obsolete
Created: 05/08/2019
Release Version: N/A
Approvers:ConfigWG, TOC

TL;DR

Proposes a config file format for automatically pulling Command Line Args from config files for all Istio Command Line components. Current implementation PR for Galley can be found here: https://github.com/istio/istio/pull/13594.

Provenance

This document represents an iteration and expansion of Config File Model for Galley, by @ozben. This document draws heavily from, and, in places, shamelessly copies its predecessor. It seeks to take lessons learned from executing on that design, and expand them to apply to Istio as a whole. Many thanks to Oz for getting this conversation started, and providing an excellent first iteration of Multi Source Config for Galley.

Overview

Istio is incredibly complicated to configure.[1] Some portion of this complexity arises from the long lists of command line arguments which are passed into each Istio binary at startup. While helm and kubernetes allow users with simple use cases to ignore most of these values, applying Istio in non-standard ways requires manually editing these lists, or maintaining them in systemd and other scripts. The table below lists occurences of 'Flags().' in the istio code base[2], which usually (though not always) represents a Command Line Flag definition.

Top-Level Folder	Flag Definitions
galley	46
istioctl	64
mixer	69

[1] See Everyone who has ever used Istio
[2] As of commit 4a3a68ea3b966e16

pilot	78
pkg	48[3]
security	33

While it is important to allow users to configure Istio in the way they are most comfortable,[4] it is also critical that alternate sources of command line config do not introduce variances in behavior, validation, or naming. With over 300 command line flags, any potential solution must require as little recoding as possible of these flags to maintain integrity (and to be achievable from a development perspective).

Additionally, some config formats (such as config files) are more amenable to structured config, which groups related flags under a common heading. Structuring config is a great way to make it more readable (especially when there are a multitude of flags available), but it is not easily compatible with flat config formats such as environment variables or command line flags.

Allowing structured config will enable next generation istio projects such as the operator and installer to exercise greater control over the configuration of Istio components. For instance, the installer could create one file per component, with the file contents checked into source control, leveraging a git ops strategy for controlling the Istio installation. This would separate component configuration from Kubernetes orchestration parameters, giving each source file better cohesion. Integration and End to End tests could also benefit from moving static configuration out of the process initialization and into source controlled config files, making the tests less dependant on the operating system environment. The overarching goal is to provide the flexibility for each operator to configure Istio in a manner that best fits their objectives.

The goal of this document is to:

- Publicize the approach, avoid surprising people
- Get feedback on the approach and details
- Request approval for standardizing across Istio

Mesh Config v.s. Multi Source Config

Alternate Config Sources are philosophically different than the mesh config file, as mesh config contains settings that mostly affect Istio mesh related settings. Most of its contents affects the final Envoy configurations that are generated.

In contrast, the config affected by Flags targets the operational behavior of a single running Istio process, such as Galley.

[3] Many of these are duplicated across all istio command line utilities
[4] The 12 Factor App principles, for instance, require Environment Variables as the avenue of config

Design

Istio leverages Cobra for Command Line Flag definition and processing, and the author of Cobra has built Viper as a library for processing config from multiple sources, with tight integration with Cobra. Viper can process config from flags, code, files, environment variables, and key value stores, and has clearly defined precedence allowing for config to pull from multiple sources simultaneously. At this time, we are proposing to allow config from the existing flags as well as from Config Files, but other sources can be added with minimal overhead.

Viper also allows structured configuration using delimited flag names to define config groups. This proposal uses Viper aliases to allow one flag to be specified in a flat or structure manner, depending on the preference of the user for the source being used. This improves user experience, making the config more native to the source being used, while avoiding redefining flags altogether, which would risk divergence. The aliases are not displayed in command usage or documentation, and custom work will be necessary to achieve automated documentation for users. The settings file itself may be formatted as YAML, TOML, HCL, JSON, or Java Config, and values may be specified using structured names, or as a flat map with flag name.

Here is a rough outline of the Galley Server Config file using structured config in YAML with explanatory comments, and some of the fields specified:

```
# General settings that apply to the whole process or to multiple sub-components.
#
general:
 # The path to the Mesh config file. If the file is not found, default Mesh config
values are used.
 meshConfigFile: "/etc/mesh-config/mesh"

 # Path to the kube config file to use. If not specified, then in-cluster
configuration will be used.
 kubeConfig: "/home/.kube/config"

# Config processing and distribution related settings.
processing:
 #...

# Validation section contains settings for the Kubernetes Admission controller that
Galley uses to integrate and validate configuration.
validation:
  enabled: false
```

Here is the same file as JSON:

```
{
“General”: {
  “meshConfigFile”: "/etc/mesh-config/mesh",
  “kubeConfig”: "/home/.kube/config"
},
“Processing”: {},
“validation”:{
  “enabled”: false
}
}
```

Here is a YAML file specifying the same values, but using a flat map of flag names:

```
meshConfigFile: "/etc/mesh-config/mesh"
kubeConfig: "/home/.kube/config"
enable-validation: false
```

And finally, here is the exact same command specified as Flags from the command line (this is already supported):

```
./galley server --meshConfigFile /etc/mesh-config/mesh --kubeConfig
/home/.kube/config --enable-validation false
```

In each of these examples, the specified config undergoes the exact same validation, processing, and default rules, meaning that each of the examples are equivalent in terms of execution.

Implementation

Migrating to Viper on top of Cobra requires minimal changes to the existing code, with no per-flag code needed. First, a flag must be added to the root command to allow specifying a config file. Next, the FlagSet for each command needs to be passed to the BindFlags function on viper. Finally, each flag will need to have its value retrieved from Viper, rather than Cobra (this can be accomplished with a for loop, avoiding per-flag code). If we choose to support environment variables with names identical to our flags, that is an additional like of code for each supporting command. Additional work would be required to support key value stores (like etcd) or processing config changes during the lifecycle of the process.

Risk

Viper is an under-maintained project, with no significant contributions since December 2018, and many open issues and pull requests that have gone ignored. This appears to be true of the Cobra and Pflags libraries as well, which are used throughout Istio and Kubernetes. In order to patch blocking bugs, Istio has forked their own version of Viper, which carries its own set of

risks. Adopting Viper does not substantially increase our risk profile, since Cobra and Pflags are already being used and are in similar states, but increased dependence on these utilities may create additional burden on Istio to maintain these libraries, which are not central to the Istio mission.