Istio Service Mesh
& Helm Repository Manager

Objectives

- Overview of Istio as an example of the service mesh pattern
- Overview of Helm as a Kubernetes package manager

Istio Service Mesh

What is Istio?

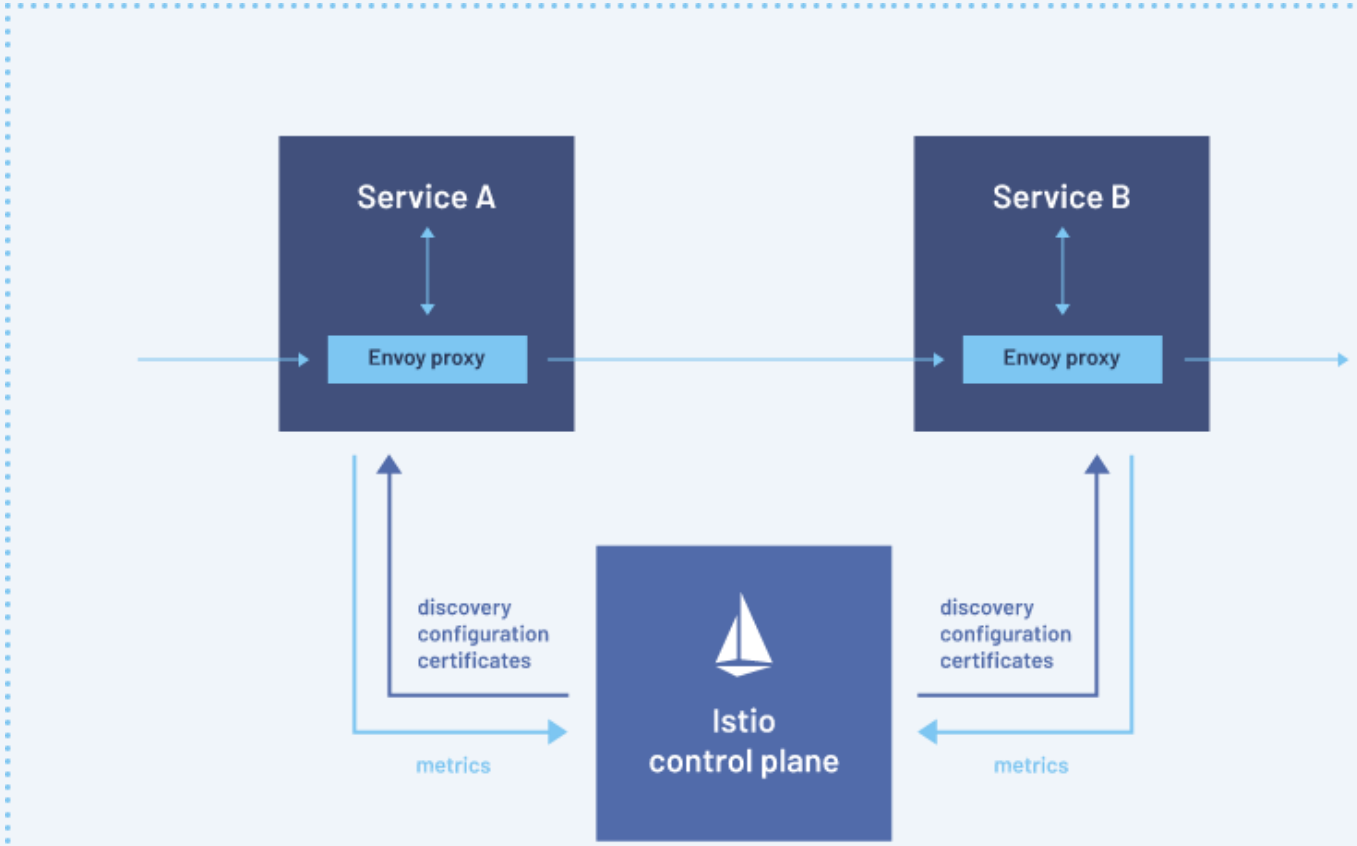
"Istio extends Kubernetes to establish a programmable, application-aware network using the powerful Envoy service proxy. Working with both Kubernetes and traditional workloads, Istio brings standard, universal traffic management, telemetry, and security to complex deployments."

https://istio.io

Istio is a Service Mesh

- Open Source

- Layers transparently onto existing distributed applications
  - Works by adding a “sidecar” along with every application deployed
  - This lets you program application-aware traffic management.



Features

- Secure service-to-service communication in a cluster
  - TLS encryption
  - Strong identity based authentication and authorization
- Automatic load balancing for HTTP, gRPC, WebSocket and TCP traffic
  - This can also be region and time zone aware.
- Fine-grained control of traffic behaviour
  - Routing rules, Circuit Breaking, Fault Injection
- Pluggable policy layer and configuration API supporting access controls, rate limits and quotas
- Automatic metrics, logs and traces for all traffic within a cluster.

How does Istio Work?

- It has a data plane and a control plane

- Data Plane:
  - The communication between services.
  - It uses an Envoy proxy to intercept all network traffic
  - Each service started has a proxy deployed alongside it.

- Control Plane
  - Takes your desired configuration and combines with its view of the services to dynamically program the proxy servers.

Lab

Do the Istio Install lab in Labdocs

The package manager for Kubernetes

Helm is the best way to find, share, and use software built for Kubernetes.

Helm – managing Kubernetes

"Helm helps you manage Kubernetes applications — Helm Charts help you define, install, and upgrade even the most complex Kubernetes application."

What is Helm?

- Helm is a tool for managing Kubernetes packages called *charts*

- Helm can
  - Create new charts from scratch
  - Package charts into an archive file
  - Interact with chart repositories
  - Install and uninstall charts into an existing Kubernetes cluster
  - Manage the release cycle of chart previously installed with Helm

Three key concepts

- The chart
  - A bundle of information to create an instance of a Kubernetes application
- The config
  - Configuration information that can be merged into a packaged chart to create a releasable chart
- The release
  - A running instance of a chart combined with a specific config.

Helm components

- The Helm Client
  - Command line for end users.
    - Local chart development
    - Managing repositories
    - Managing releases
    - Interfacing with the Helm library
- The Helm Library
  - Provides the logic for all Helm operations
  - Interfaces with the Kubernetes API server to enable
    - Combining a chart and a configuration to build a release
    - Installing charts into Kubernetes and providing a release object
    - Upgrading and uninstalling charts by interacting with Kubernetes

Helm Charts

- A chart is a collection of files inside a directory structure.
- The directory name is the (unversioned) chart name.
- A chart describing WordPress would look like this:

```
wordpress/
  Chart.yaml
  LICENSE
  README.md
  values.yaml
  values.schema.json
  charts/
  crds/
  Templates/
  templates/NOTES.txt
```

A YAML file containing information about the chart

```
wordpress/
  Chart.yaml
  LICENSE
  README.md
  values.yaml
  values.schema.json
  charts/
  crds/
  Templates/
  templates/NOTES.txt
```

OPTIONAL: A plain text file containing the license for the chart

```
wordpress/
  Chart.yaml
  LICENSE
  README.md
  values.yaml
  values.schema.json
  charts/
  crds/
  Templates/
  templates/NOTES.txt
```

OPTIONAL: A human-readable README file

```
wordpress/
  Chart.yaml
  LICENSE
  README.md
  values.yaml
  values.schema.json
  charts/
  crds/
  Templates/
  templates/NOTES.txt
```

The default configuration values for this chart

```
wordpress/
  Chart.yaml
  LICENSE
  README.md
  values.yaml
  values.schema.json
  charts/
  crds/
  Templates/
  templates/NOTES.txt
```

OPTIONAL: A JSON Schema for imposing a structure on the values.yaml file

```
wordpress/
  Chart.yaml
  LICENSE
  README.md
  values.yaml
  values.schema.json
  charts/
  crds/
  Templates/
  templates/NOTES.txt
```

A directory containing any charts upon which this chart depends.

```
wordpress/
  Chart.yaml
  LICENSE
  README.md
  values.yaml
  values.schema.json
  charts/
  crds/
  Templates/
  templates/NOTES.txt
```

Custom Resource Definitions

```
wordpress/
  Chart.yaml
  LICENSE
  README.md
  values.yaml
  values.schema.json
  charts/
  crds/
  Templates/
  templates/NOTES.txt
```

A directory of templates that, when combined with values, will generate valid Kubernetes manifest files.

```
wordpress/
  Chart.yaml
  LICENSE
  README.md
  values.yaml
  values.schema.json
  charts/
  crds/
  Templates/
  templates/NOTES.txt
```

OPTIONAL: A plain text file containing short usage notes

The Chart.yaml File

- This is the central requirement for a chart.
- It has the following structure:

```
apiVersion: The chart API version (required)
name: The name of the chart (required)
version: A SemVer 2 version (required)
kubeVersion: A SemVer range of compatible Kubernetes versions (optional)
description: A single-sentence description of this project (optional)
type: The type of the chart (optional)
keywords:
      - A list of keywords about this project (optional)
home: The URL of this projects home page (optional)
sources:
      - A list of URLs to source code for this project (optional)
                                                                  ...
```

...

```
dependencies: # A list of the chart requirements (optional)
      - name: The name of the chart (nginx)
        version: The version of the chart ("1.2.3")
        repository: (optional) The repository URL
("https://example.com/charts") or alias ("@repo-name")
        condition: (optional) A yaml path that resolves to a boolean, used for
enabling/disabling charts (e.g. subchart1.enabled )
        tags: # (optional)
            - Tags can be used to group charts for enabling/disabling together
        import-values: # (optional)
            - ImportValues holds the mapping of source values to parent key to
be imported. Each item can be a string or pair of child/parent sublist items.
        alias: (optional) Alias to be used for the chart. Useful when you have
to add the same chart multiple times
```

...

...

```
maintainers: # (optional)
      - name: The maintainers name (required for each maintainer)
        email: The maintainers email (optional for each maintainer)
        url: A URL for the maintainer (optional for each maintainer)
icon: A URL to an SVG or PNG image to be used as an icon (optional).
appVersion: The version of the app that this contains (optional). Needn't be
SemVer. Quotes recommended.
deprecated: Whether this chart is deprecated (optional, boolean)
annotations:
      example: A list of annotations keyed by name (optional).
```

Charts and Versioning

- **Every** chart must have a version number

- It must follow the SemVer2 standard (https://semver.org/)
  - This is the MAJOR.MINOR.PATCH convention

Chart Type

- A chart can be either an `application` or a `library`.

- `application`:
  - Default type
  - Standard chart that can be operated on fully

- `library`:
  - Provides utilities or functions for the chart builder.
  - A library is not installable and usually doesn't contain any resource objects.
  - You could use an application chart as a library chart (by just setting the type to `library`)
    - In this case, all resource objects of the chart will not be rendered.

More info

- https://helm.sh

Summary

- Overview of Istio as an example of the service mesh pattern
- Overview of Helm as a Kubernetes package manager

Questions or Comments?