Navigating the service mesh ecosystem

Lachlan Evenson, Microsoft

CNCF Ambassador

What is Service Mesh Interface (SMI)?

Announced in May 2019, Service Mesh Interface (SMI) is a specification for service meshes that run on Kubernetes. It defines a common standard that can be implemented by a variety of providers.

Service Mesh Interface provides:

- A standard interface for meshes on Kubernetes
- A basic feature set for the most common mesh use cases
- Flexibility to support new mesh capabilities over time
- Space for the ecosystem to innovate with mesh technology

Get started quickly

Simpler is better

Ecosystem friendly

Initial specifications for the top three service mesh features covering the most common service mesh capabilities:

- **Traffic policy** – apply policies like identity and transport encryption across services

- **Traffic telemetry** – capture key metrics like error rate and latency between services

- **Traffic management** – shift and weight traffic between different services

Why does the ecosystem need SMI?

Provider Agnostic

The goal of SMI is to provide a common, portable set of service mesh APIs which a Kubernetes user can use in a provider agnostic manner. In this way people can define applications that use service mesh technology without tightly binding to any specific implementation.

Kubernetes Native

SMI is specified as a collection of Kubernetes Custom Resource Definitions (CRD) and Extension API Servers. These APIs can be installed onto any Kubernetes cluster and manipulated using standard tools.

Extensible

With many exciting mesh capabilities in development, we fully expect to evolve SMI APIs over time, and look forward to extending the current specification with new capabilities.

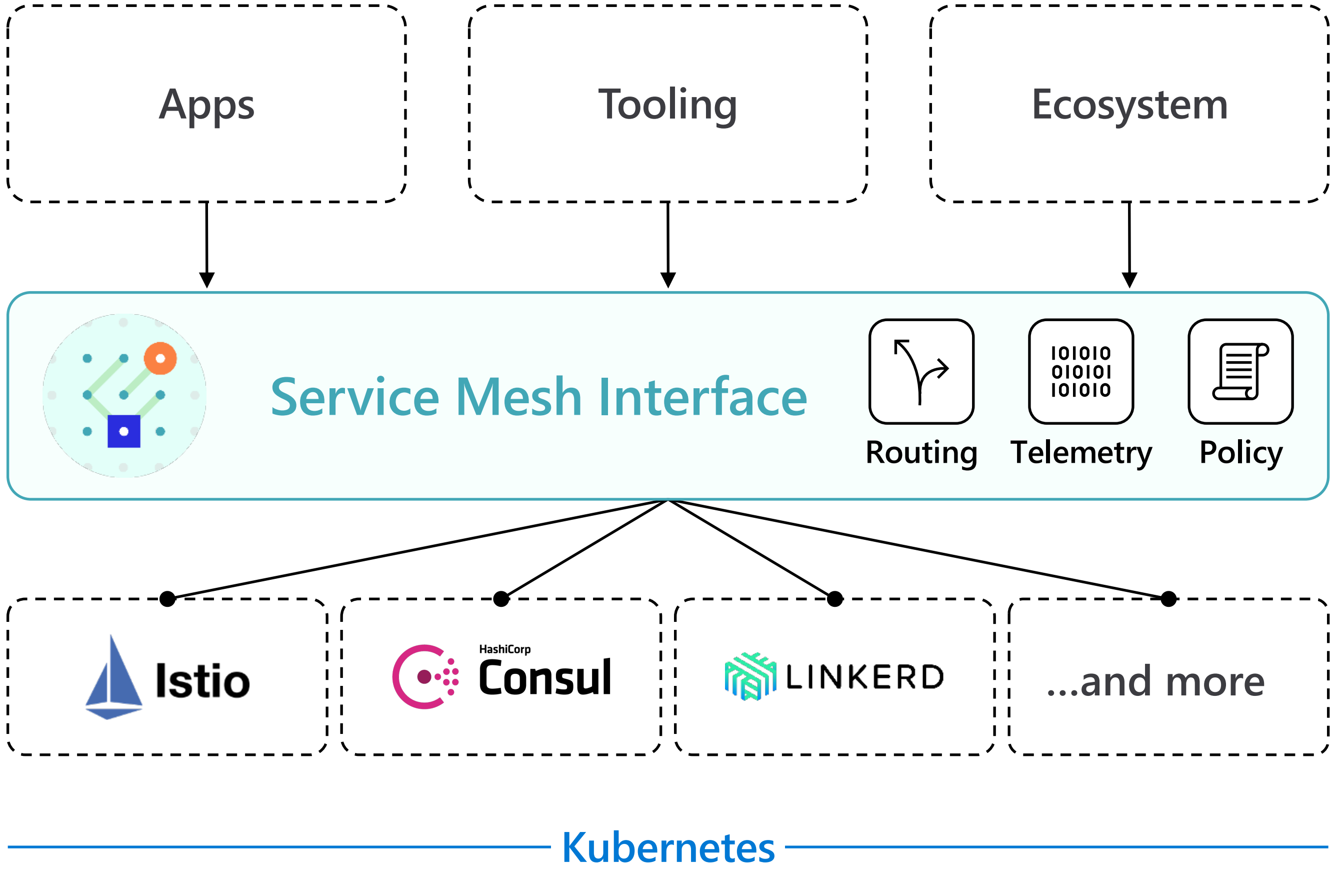
SMI in the service mesh ecosystem

Service Mesh Interface (SMI)

SMI defines a set of APIs that can be implemented by individual mesh providers. Service meshes and tools can either integrate directly with SMI or an adapter can consume SMI and drive native mesh APIs.

Service Mesh Interface aims to provide:

- **Standard interface** for service mesh on Kubernetes
- **Basic feature set** to address most common scenarios
- **Extensible** to support new features as they become widely available



What APIs are included?

- **Traffic Access Control** - configure access to specific pods and routes based on the identity of a client for locking down applications to only allowed users and services.

- **Traffic Specs** - define how traffic looks on a per-protocol basis. These resources work in concert with access control and other types of policy to manage traffic at a protocol level.

- **Traffic Split** - incrementally direct percentages of traffic between various services to assist in building out canary rollouts.

- **Traffic Metrics** - expose common traffic metrics for use by tools such as dashboards and autoscalers.

SMI Spec Status

Latest Release	Working Draft	
Core Specification:		
SMI Specification	v0.5.0	v0.6.0-WD
Specification Components		
Traffic Access Control	v1alpha2	v1alpha3-WD
Traffic Metrics	v1alpha1	v1alpha2-WD
Traffic Specs	v1alpha3	v1alpha4-WD
Traffic Split	v1alpha3	v1alpha4-WD

SMI Technical Overview

- Kubernetes Custom Resource Definitions (CRD)
    - Installable on any Kubernetes cluster
    - interact using standard Kubernetes tools like kubectl

- SMI provider runs in Kubernetes cluster to act on APIs
    - for configurable resources:
        - reflects back on their contents
        - configures the provider's components within a cluster
    - for extension APIs:
        - translates from internal types to return types
    - SMI SDK for Go for easy of implementation

- Common components
    - Extension API Servers: https://github.com/servicemeshinterface/smi-metrics
    - Init containers
    - Innovate on functionality instead of retreading the same patterns

In partnership with

Microsoft

LINKERD

HashiCorp

solo.io

kinvolk

Red Hat

RANCHER

docker

LAYER5

CANONICAL

weaveworks

ASPEN MESH

Pivotal

vmware

kubecost

SMI Community & Code Repositories

- Joined CNCF as Sandbox project, April 2020: smi-spec.io/blog/smi-joins-cncf/
- SMI site: smi-spec.io
- GitHub: https://github.com/servicemeshinterface/

- Meetings - every other Wednesday, 10am PT - https://github.com/servicemeshinterface/smi-spec#communications
- Slack – CNCF Slack #smi
- SMI SDK for Golang - https://github.com/servicemeshinterface/smi-sdk-go
- SMI adapter for Istio - https://github.com/servicemeshinterface/smi-adapter-istio
- Expose SMI Metrics - https://github.com/servicemeshinterface/smi-metrics
- SMI Website - https://github.com/servicemeshinterface/smi-spec.io

SMI Goals & Non-Goals

- **Goals**
  - define common, portable set of Service Mesh APIs
  - provider agnostic
  - enable ecosystem tools to innovate and offer higher-level services
  - iterate over time as the service mesh ecosystem evolves

- **Non-Goals**
  - implement a service mesh offered by the SMI project
  - require implementation of specific SMI APIs
  - restrict what it means to be a service mesh: providers are welcome to add provider-specific extensions and APIs beyond the SMI spec

Open Service Mesh: Adding to the SMI ecosystem

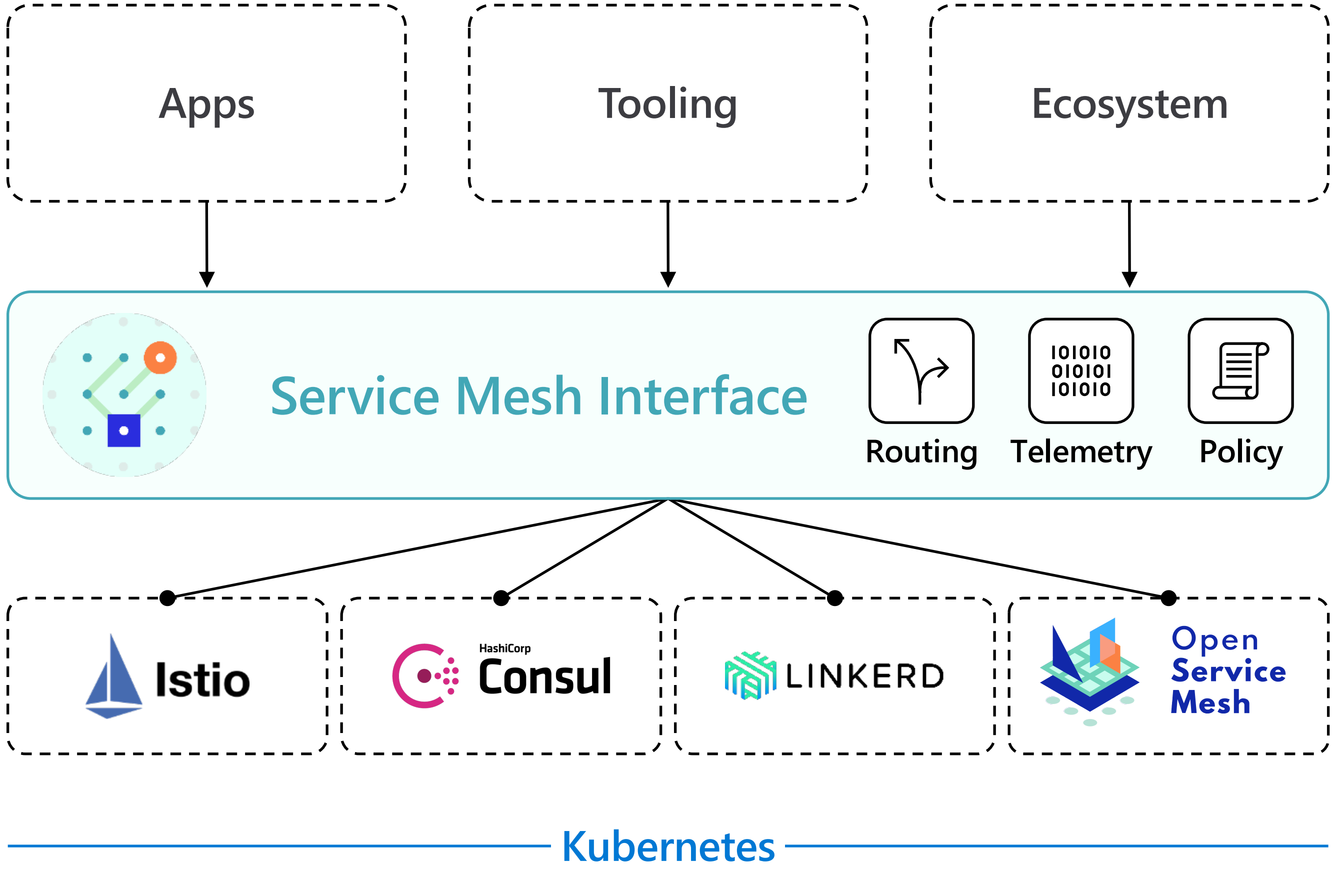
Open Service Mesh (OSM)

openservicemesh.io

Announced August 2020: openservicemesh.io/blog/introducing-open-service-mesh/

OSM is being proposed for CNCF donation: github.com/cncf/toc/pull/507

OSM welcomes you to the project: github.com/openservicemesh/osm/



Ecosystem

HashiCorp Consul

weave flagger

LINKERD

mæsh

MESHERY

Open Service Mesh

RIO

SERVICE MeshHub

Istio

Open Service Mesh (OSM)

Open Service Mesh (OSM) is a lightweight and extensible cloud native service mesh.

OSM takes a **simple approach** for users to uniformly manage, secure, and get out-of-the box observability features for highly dynamic microservice environments.

Using the CNCF Envoy project, OSM implements Service Mesh Interface (SMI) for securing and managing your microservice applications.

OSM principles

- **Simple** to understand and contribute to
- **Effortless** to install, maintain, and operate
- **Painless** to troubleshoot
- **Easy** to configure via Service Mesh Interface (SMI)

Standardized

Simplified

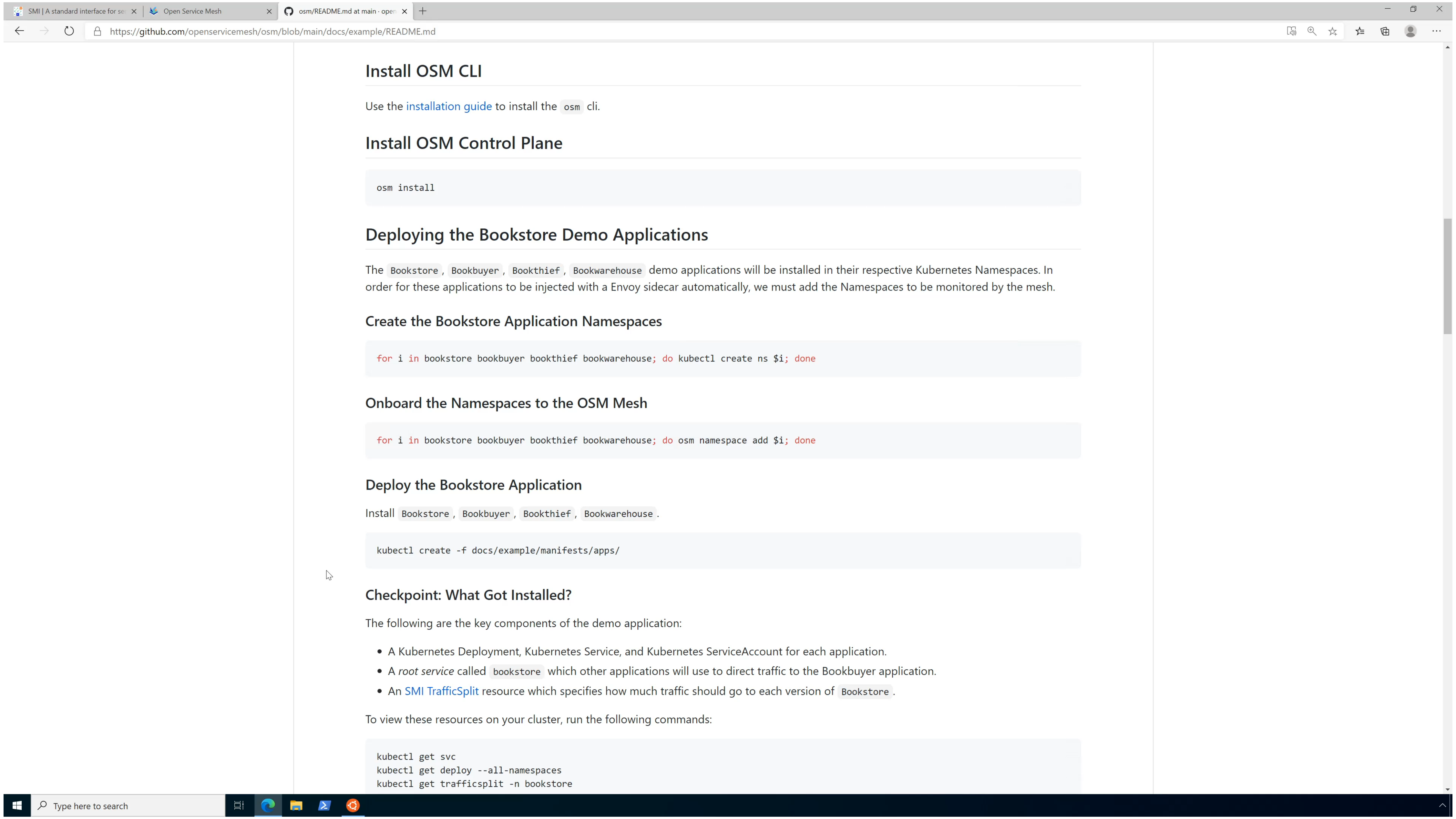
Extensible

OSM features

- Easily and transparently configure **traffic shifting** for deployments
- Secure end-to-end service to service communication by **enabling mTLS**
- Define and execute fine grained **access control** policies for services
- Observability and insights into application metrics for **debugging and monitoring** services
- Integrate with external **certificate management** services/solutions with a pluggable interface
- Onboard applications onto the mesh by enabling **automatic sidecar injection** of Envoy proxy
- **Flexible** enough to handle both simple and complex scenarios through SMI and Envoy XDS APIs

demo: open service mesh

OSM Demo


SMI | A standard interface for se
Open Service Mesh
osm/README.md at main · open
https://github.com/openservicemesh/osm/blob/main/docs/example/README.md
Type here to search

Install OSM CLI

Use the installation guide to install the `osm` cli.

Install OSM Control Plane

```
osm install
```

Deploying the Bookstore Demo Applications

The `Bookstore`, `Bookbuyer`, `Bookthief`, `Bookwarehouse` demo applications will be installed in their respective Kubernetes Namespaces. In order for these applications to be injected with a Envoy sidecar automatically, we must add the Namespaces to be monitored by the mesh.

Create the Bookstore Application Namespaces

```
for i in bookstore bookbuyer bookthief bookwarehouse; do kubectl create ns $i; done
```

Onboard the Namespaces to the OSM Mesh

```
for i in bookstore bookbuyer bookthief bookwarehouse; do osm namespace add $i; done
```

Deploy the Bookstore Application

Install `Bookstore`, `Bookbuyer`, `Bookthief`, `Bookwarehouse`.

```
kubectl create -f docs/example/manifests/apps/
```

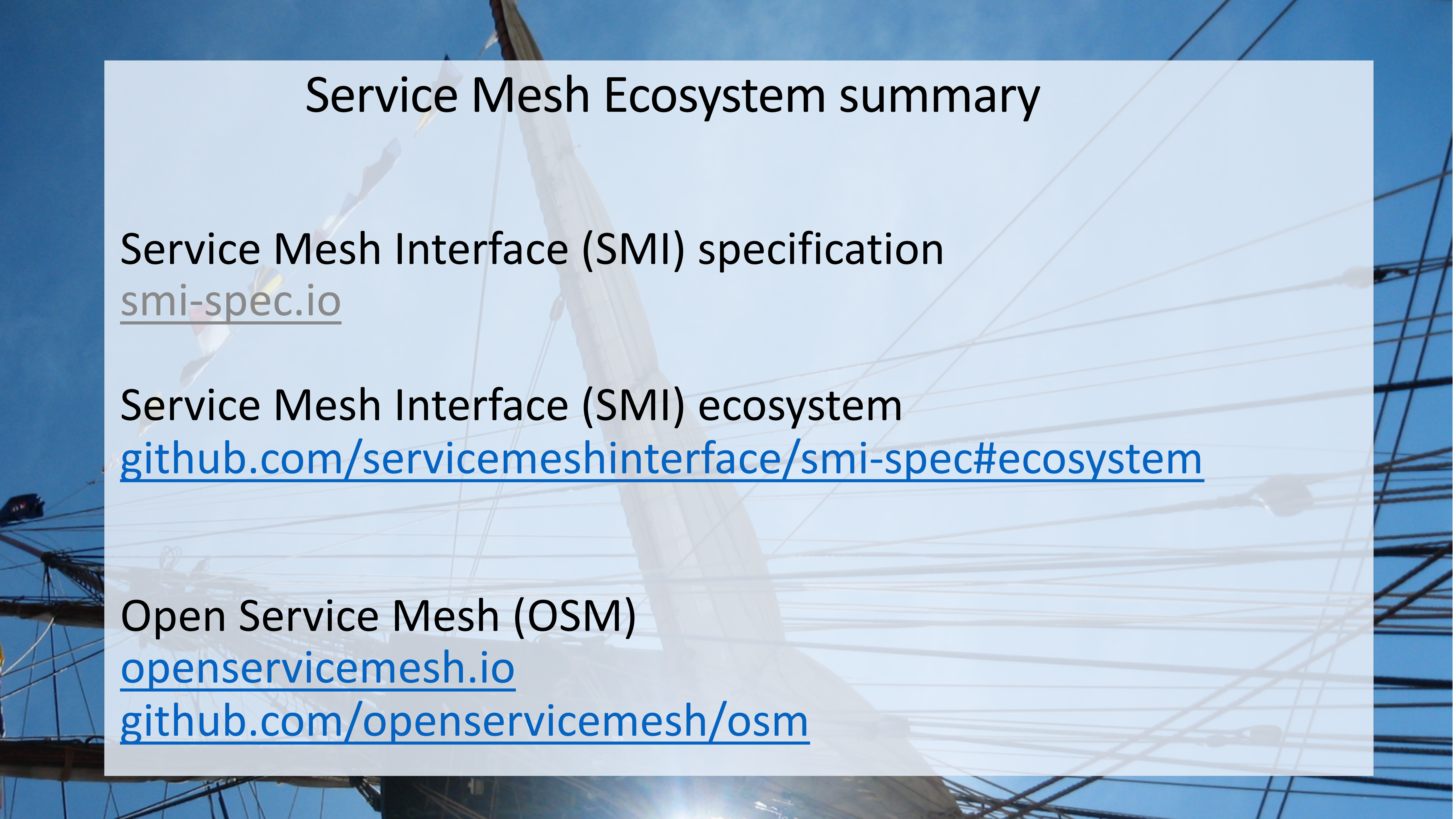
Checkpoint: What Got Installed?

The following are the key components of the demo application:

- A Kubernetes Deployment, Kubernetes Service, and Kubernetes ServiceAccount for each application.
- A *root service* called `bookstore` which other applications will use to direct traffic to the Bookbuyer application.
- An SMI TrafficSplit resource which specifies how much traffic should go to each version of `Bookstore`.

To view these resources on your cluster, run the following commands:

```
kubectl get svc
kubectl get deploy --all-namespaces
kubectl get trafficsplit -n bookstore
```



Service Mesh Ecosystem summary

Service Mesh Interface (SMI) specification
smi-spec.io

Service Mesh Interface (SMI) ecosystem
github.com/servicemeshinterface/smi-spec#ecosystem

Open Service Mesh (OSM)
openservicemesh.io
github.com/openservicemesh/osm