# Service Mesh

Unpacking Istio

Jeff Barnes
GC Micro-mission

# Unscripted post-demo deck

Provide background for the unscripted Service Mesh walkthrough

- How we got here and why
- Where to next

Service Mesh aligns to a number of current deliverables and overall guidance/standards in the GC, SSC and EA:

- Provides a mechanism to implement GC guidance such as and Directive on Management of Information Technology, with an updated Appendix C and D https://www.tbs-sct.gc.ca/pol/doc-eng.aspx?id=15249 and the Digital Operations Strategic Plan: 2018-2022 https://www.canada.ca/en/government/system/digital-government/digital-operations-strategic-plan-2018-2022.html
- Use case for software defined overlays, micro segmentation and even zero-trust networking (related to the E2E and CBSA zoning, SCED and the CSE guidance updates)
- Supports partner application migration, especially monitoring
- Components support the Government of Canada Standards on APIs https://www.canada.ca/en/government/publicservice/modernizing/government-canada-standards-apis.html, API gateway and framework (including distributed tracing/open tracing)
- Open source, globally adopted
- Numerous other cloud-native benefits (container orchestration, reduces hardware footprint, declarative, application support including circuit breaking, fault tolerance,....)

# Origin Story

In an effort to better understand security for containers, orchestration, CI/CD; created a simple POC including Container Security Providers (Neuvector, AquaSec), CI/CD (Jenkins), Docker, minikube, Cloud Foundry, OpenShift; on bare metal and in various public clouds

Quickly discovered introductory tutorials often did not include security requirements and challenges concerning containers, especially in enterprise environments that must meet NIST SP800-53

Started with Go, Node and other runtimes; created and then secured Docker images, including the adoption of multi-stage and non-root builds, vulnerability scanning, code analysis, and other security recommendations*

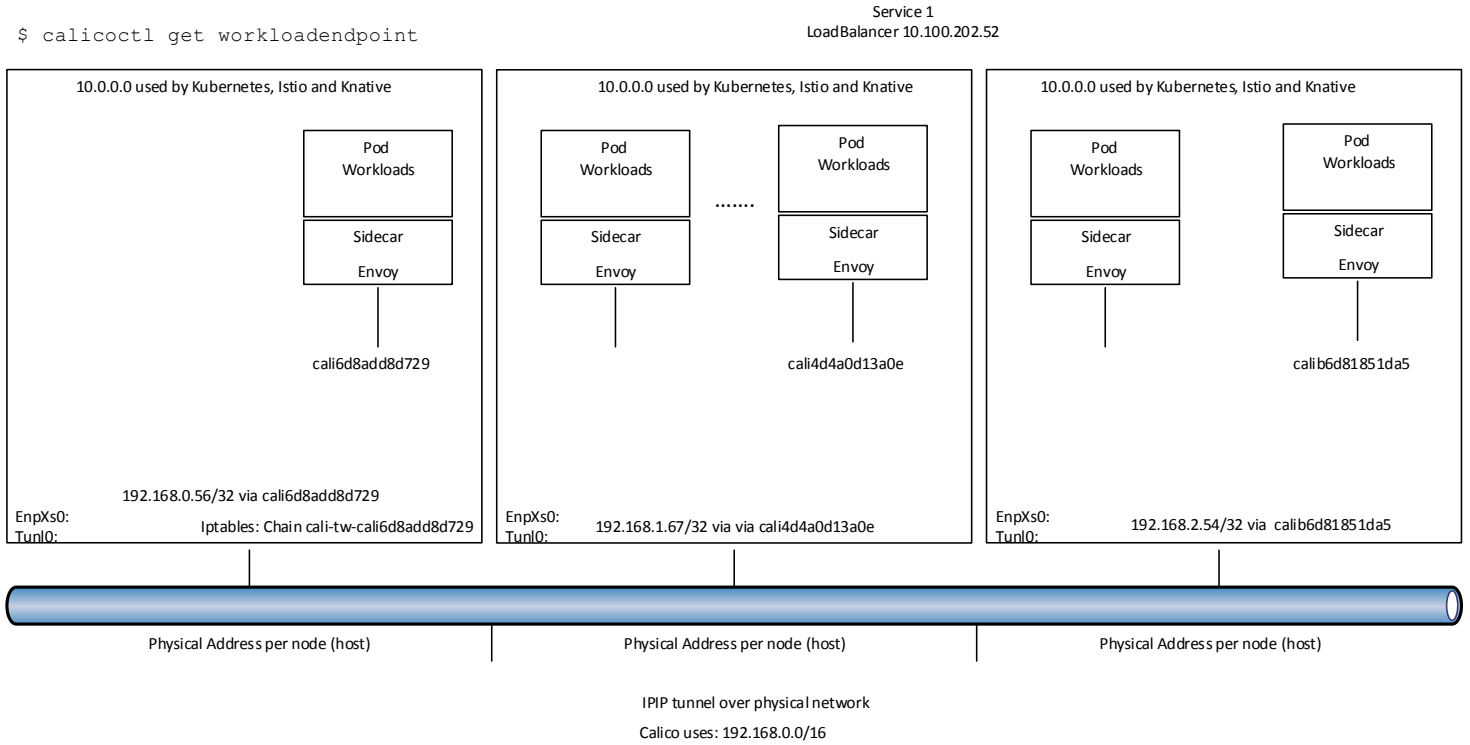
Subsequently led to a POC for securing platforms, especially Kubernetes and OpenShift OKD; which in turn included Helm for application deployment; and now where we are now – investigating Service Mesh using Istio

# Container Orchestrator Networking

Kubernetes 1.13 deployment, Calico IPIP + Kubeproxy (iptables) for CNI

High-level overview of Kubernetes networking.

- Physical Addresses assigned to hosts (nodes) that comprise the cluster
- Calico uses 192.168.0.0/16 for a pseudo-overlay on top of the nodes
- 10.0.0.0 used by Kubernetes (and Istio)

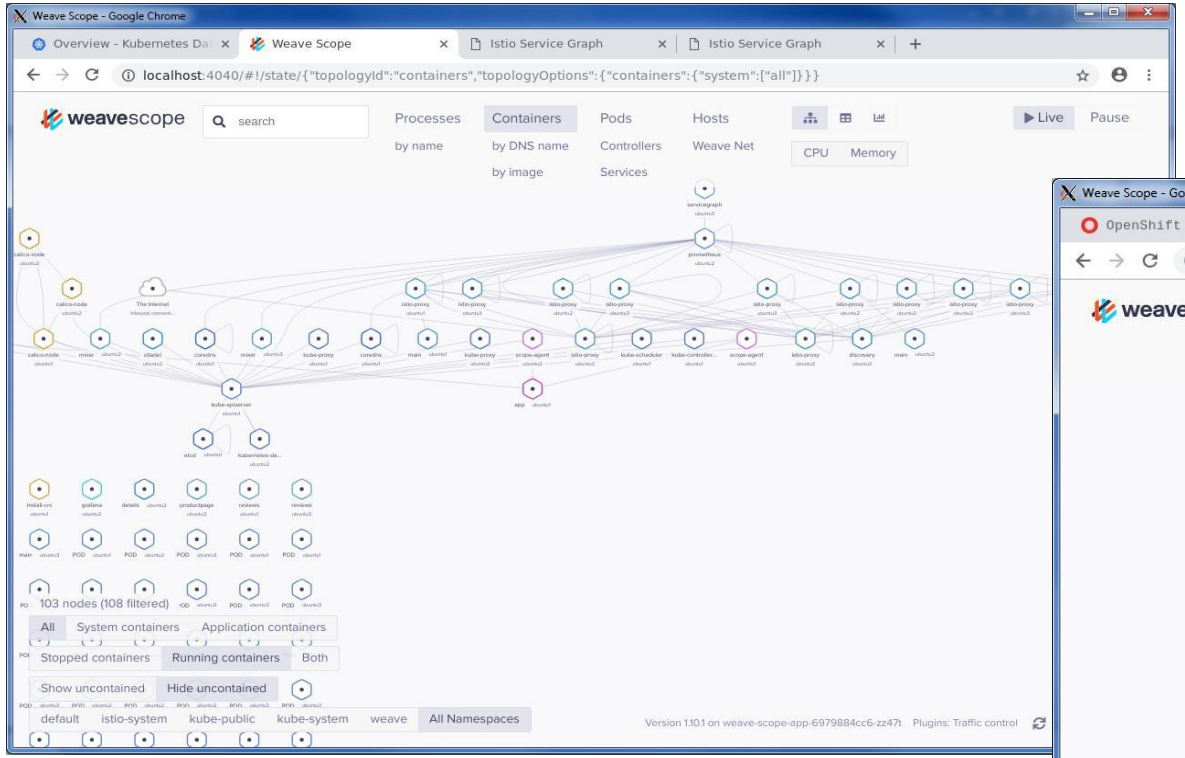



```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: my-network-policy
  namespace: default
spec:
  podSelector:
    matchLabels:
      role: db
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          project: myproject
    - podSelector:
        matchLabels:
          role: frontend
    ports:
    - protocol: TCP
      port: 6379
```

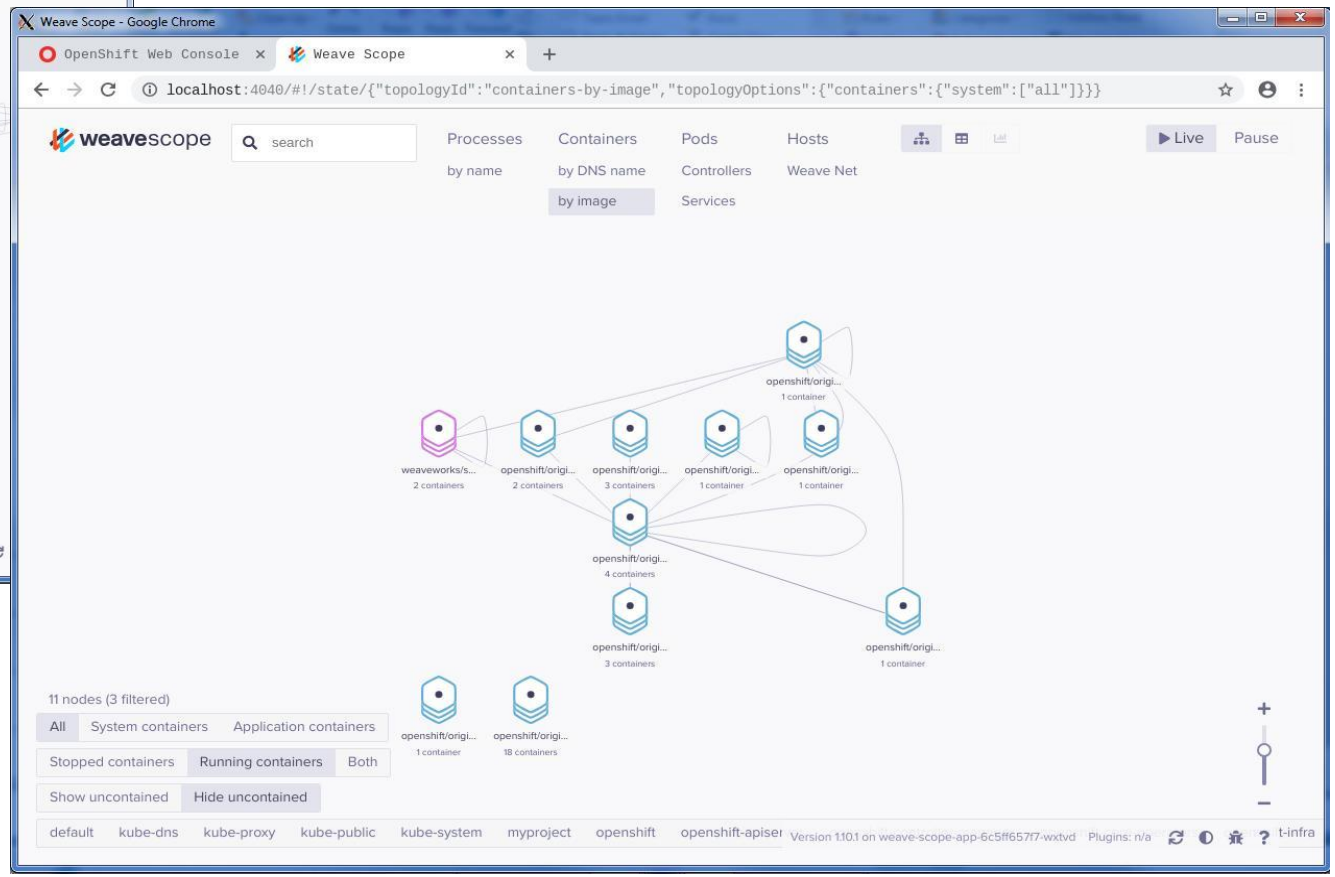
Example Network Policy

# Infrastructure Visualization

Using Weavescope to visual Kubernetes and OpenShift clusters




OpenShift cluster



# Why a Service Mesh

Decided to install a Service Mesh to better understand deployment, security and support of microservices and to examine use case for zoning, micro-segmentation, zero-trust networking, and cloud-native deployment

*"Much as Kubernetes abstracts the underlying compute capability from the hardware and operating system, so too do service meshes abstract routing and connectivity from the underlying networking. Once the routing and connectivity is separated from the infrastructure and can be treated as code, it becomes practical to have a truly application tailored, least privilege approach to microsegmentation."*

Out of the box, Istio 1.0.5 comes with a number of features and components; including Prometheus and Grafana (logging/telemetry), Visualization, Zipken and Jaeger for distributed tracing; plus all the benefits of sidecar proxies: Request Routing (for Canary or A/B deployments), Fault Injection, Traffic Shifting/control, Security/segmentation, mTLS, Health checking and on and on (all without changing any application code)

Envoy can stand alone and is included in many service meshes, such as Istio, AWS App Mesh….


Consul is a common proxy used for building service meshes in heterogeneous technical environments

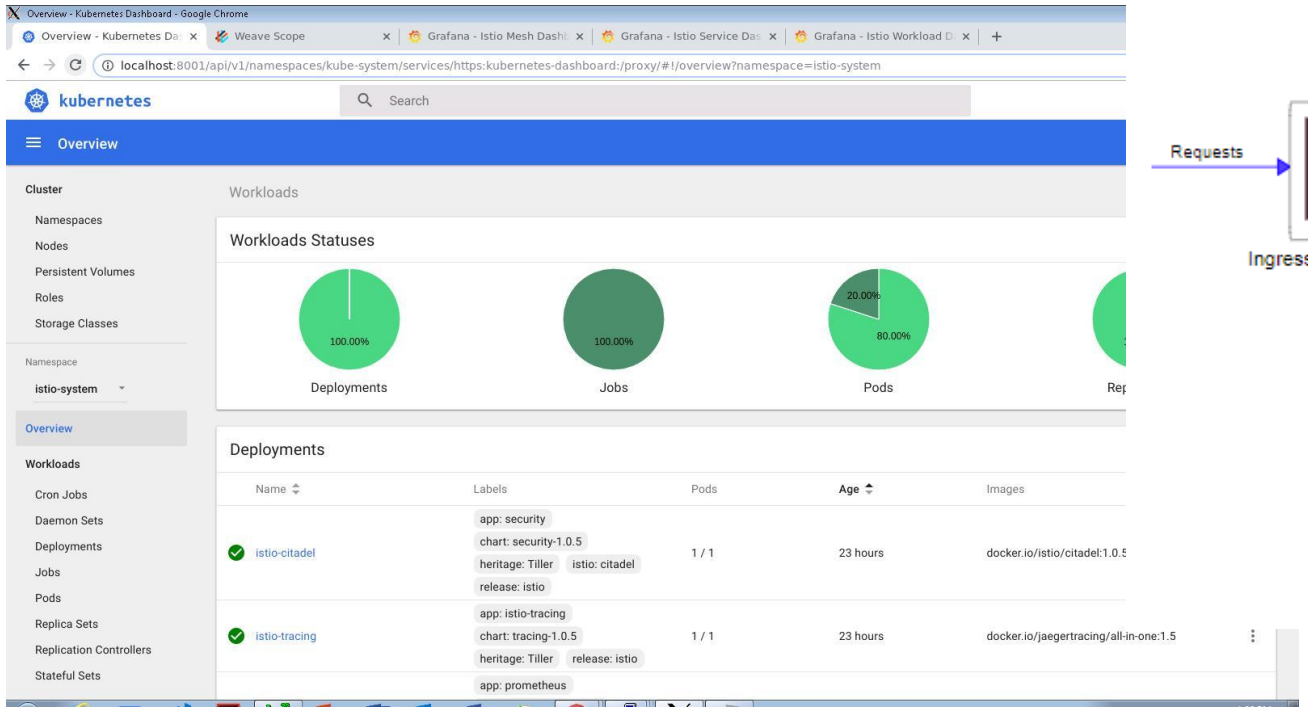
# Istio

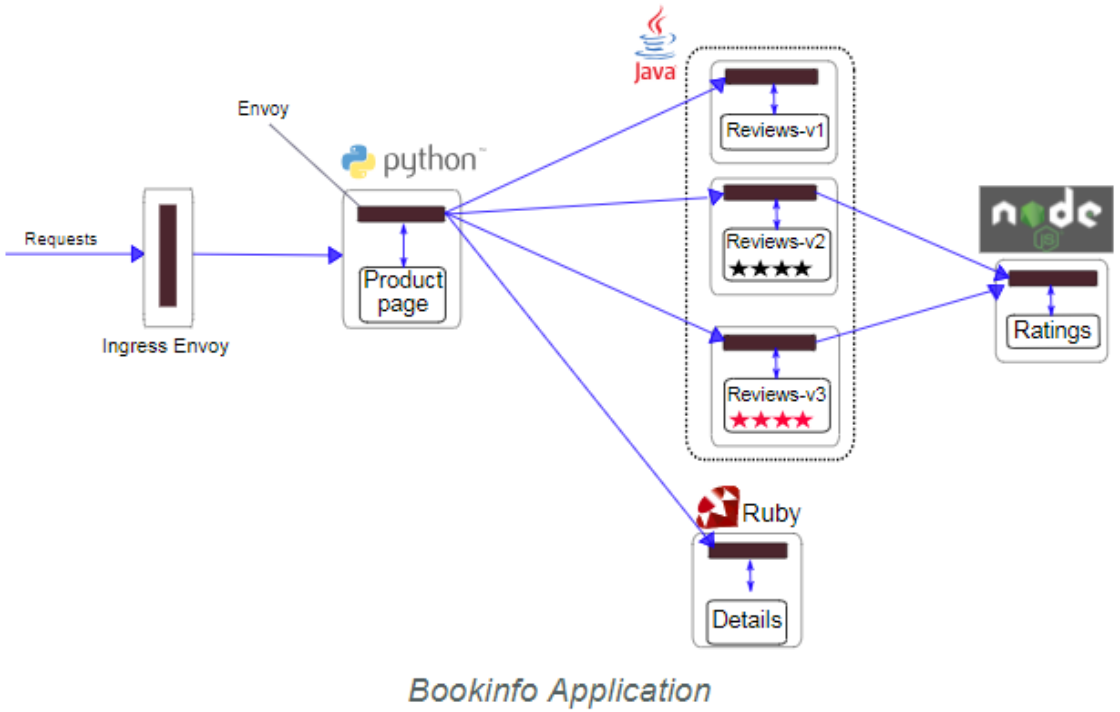
Install in Istio-system namespace of a current kubernetes cluster.  Main components include:

- Envoy - an open-source edge and service proxy designed for cloud-native applications
- Mixer - a platform-independent component responsible for enforcing access control and usage policies
- Pilot - provides service discovery for the Envoy sidecars and traffic management capabilities

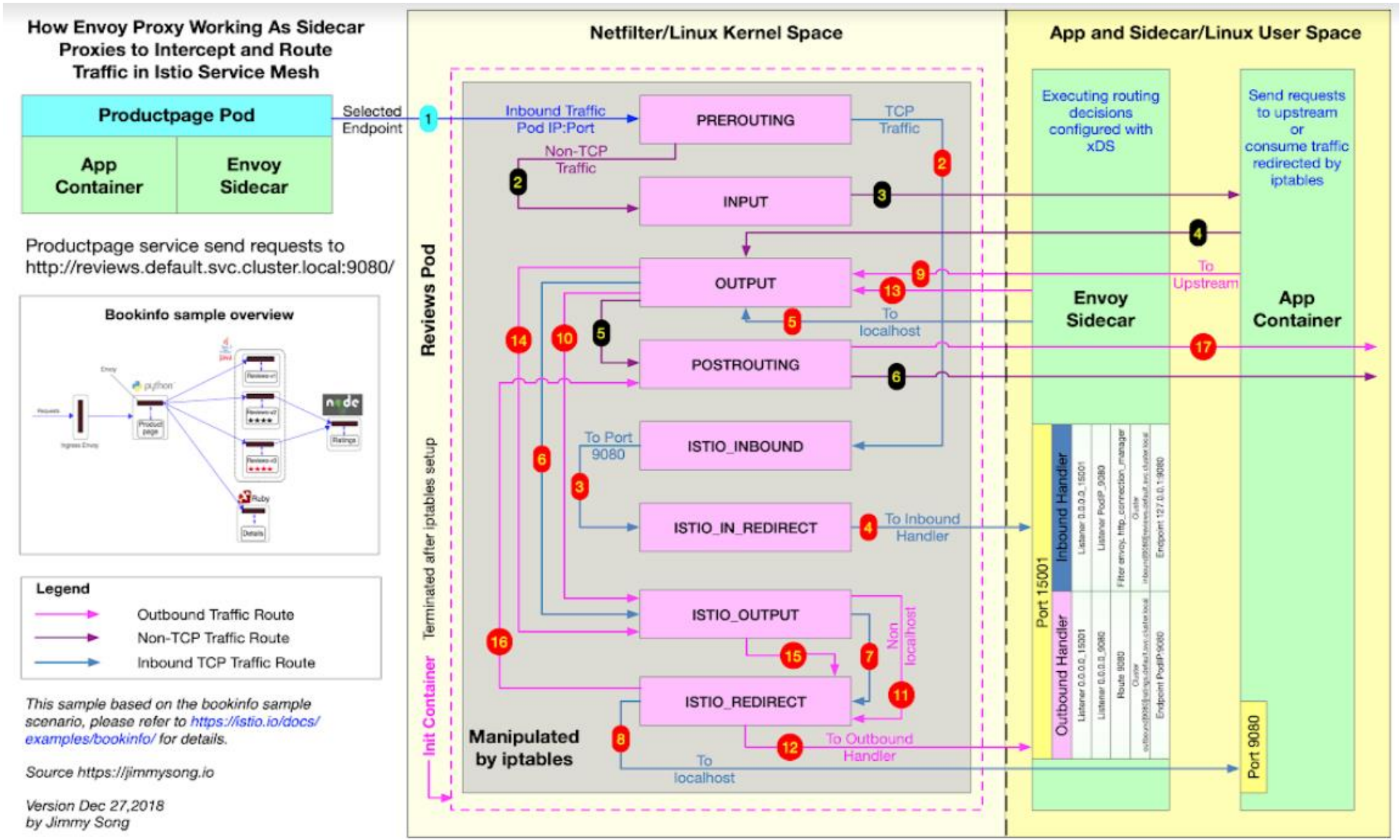
We install two services in the default namespace

- ***Main*** – a 3 pod Simple web server
- ***Bookinfo*** – Example microservices for testing



# Underneath the hood



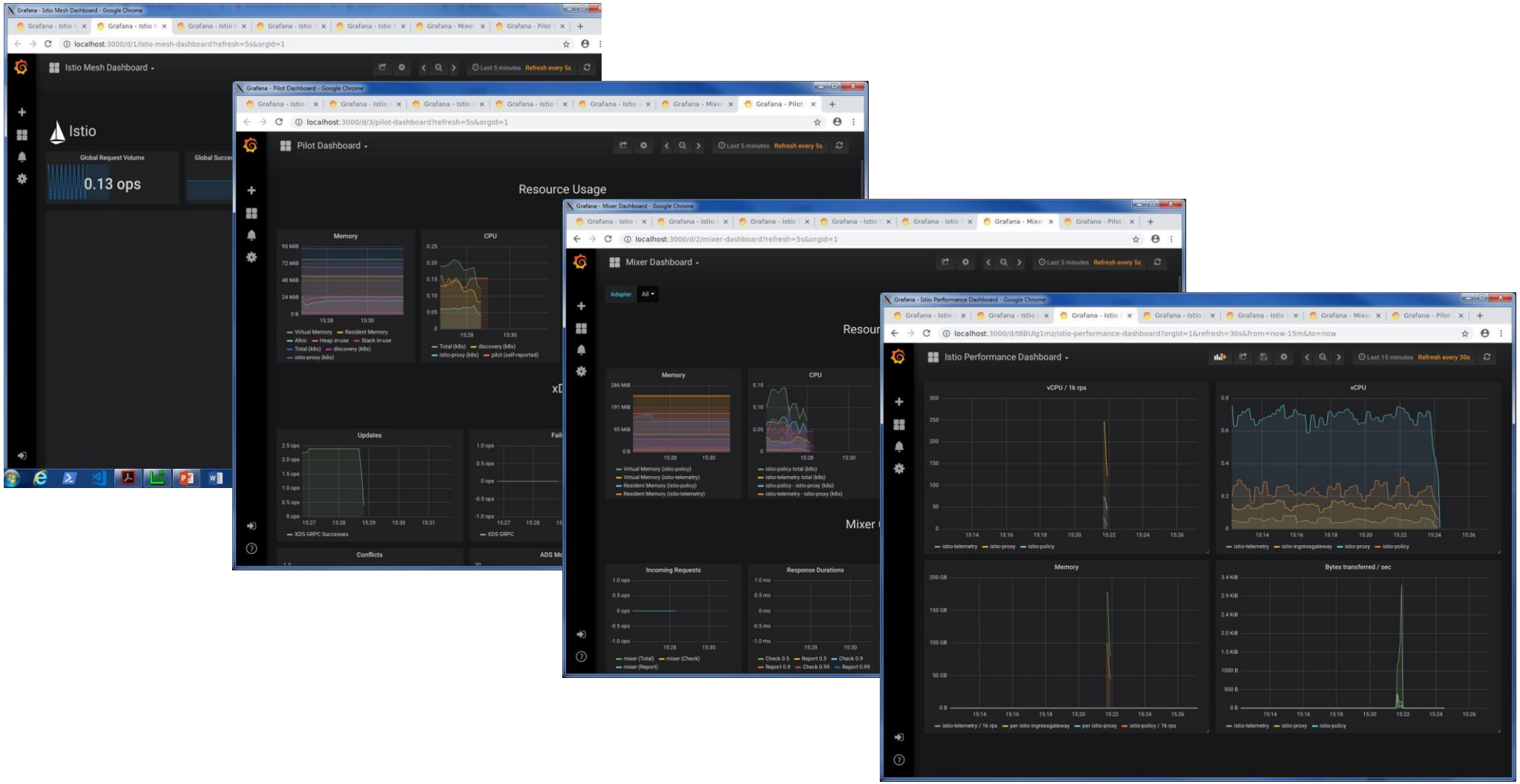
# Traffic Shaping

Performed the following on ***Bookinfo***, works as intended

Request routing This task will first direct all incoming traffic for the Bookinfo application to the v1 version of the reviews service. It will then send traffic only from a specific test user to version v2, leaving all other users unaffected.
Fault injection We will now use Istio to test the resiliency of the Bookinfo application by injecting an artificial delay in requests between the reviews:v2 and ratings services. Observing the resulting behavior as the test user, we will notice that the v2 version of the reviews service has a bug. Note that all other users are unaware of this testing against the live system.
Traffic Shifting Finally, we will use Istio to gradually migrate traffic for all users from to a v3 version of the reviews service, one which includes the fix for the bug discovered in v2.
Collecting metrics This task will configure Mixer to collect a uniform set of metrics across all services in the Bookinfo application.
Querying metrics This task installs the Prometheus add-on for metrics collection and demonstrates querying a configured Prometheus server for Istio metrics.
Distributed tracing We will now use Istio to trace how requests are flowing across services in the application. Distributed tracing speeds up troubleshooting by allowing developers to quickly understand how different services contribute to the overall end-user perceived latency. In addition, it can be a valuable tool to diagnosis and troubleshooting in distributed applications.
Using the Istio Dashboard This task installs the Grafana add-on with a preconfigured dashboard for monitoring mesh traffic.

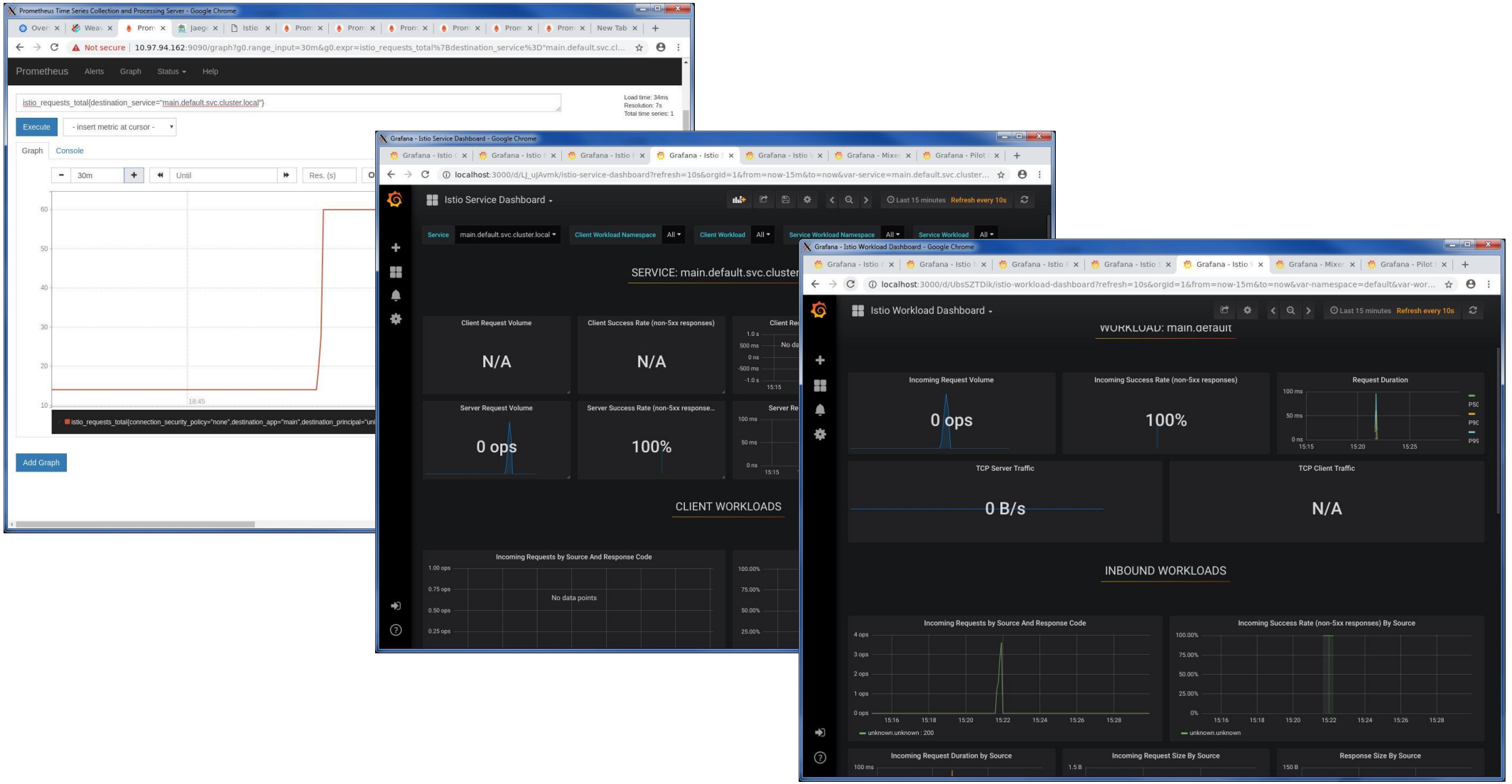
# Prometheus and Grafana (logging/telemetry)

A number of Grafana dashboards are installed with Istio, which track various metrics found in Istio components (like Envoy)



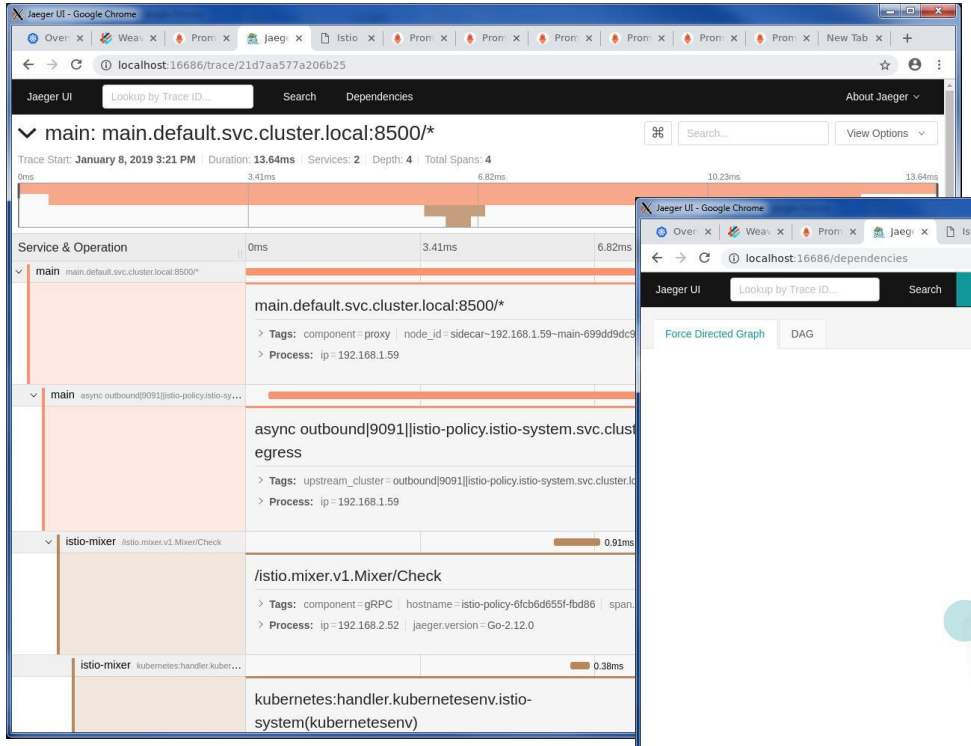
# Prometheus and Grafana (logging/telemetry)

***Main*** and ***Bookinfo*** can be tracked in Prometheus and the Grafana dashboards

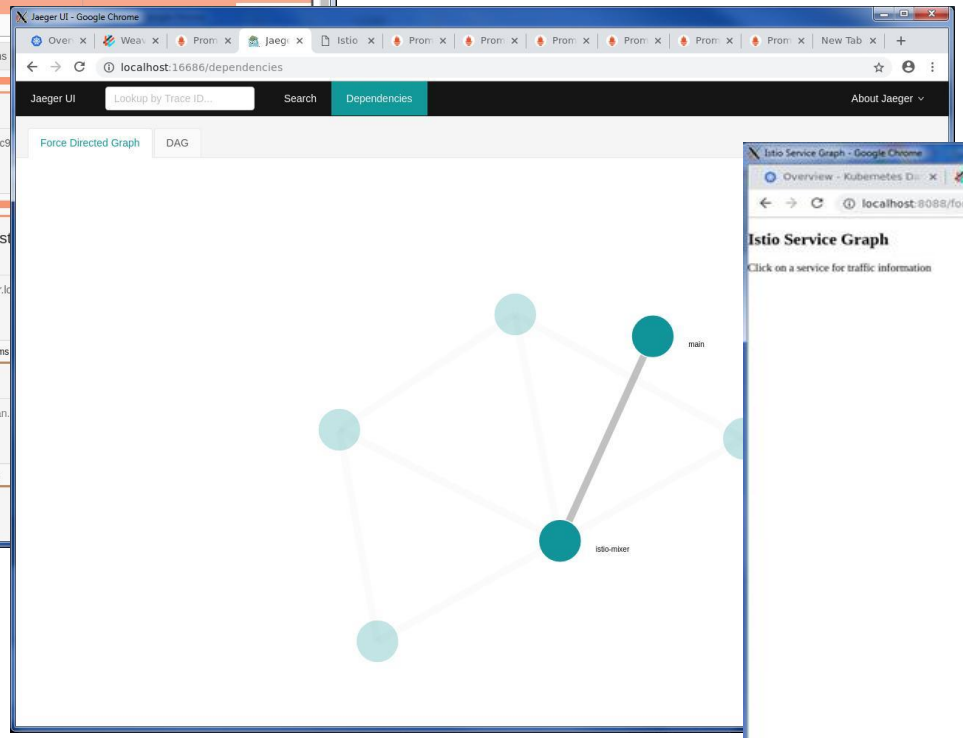


# Distributed/open-tracing (Jaeger) and Service Graph

Accessing Service ***Main***

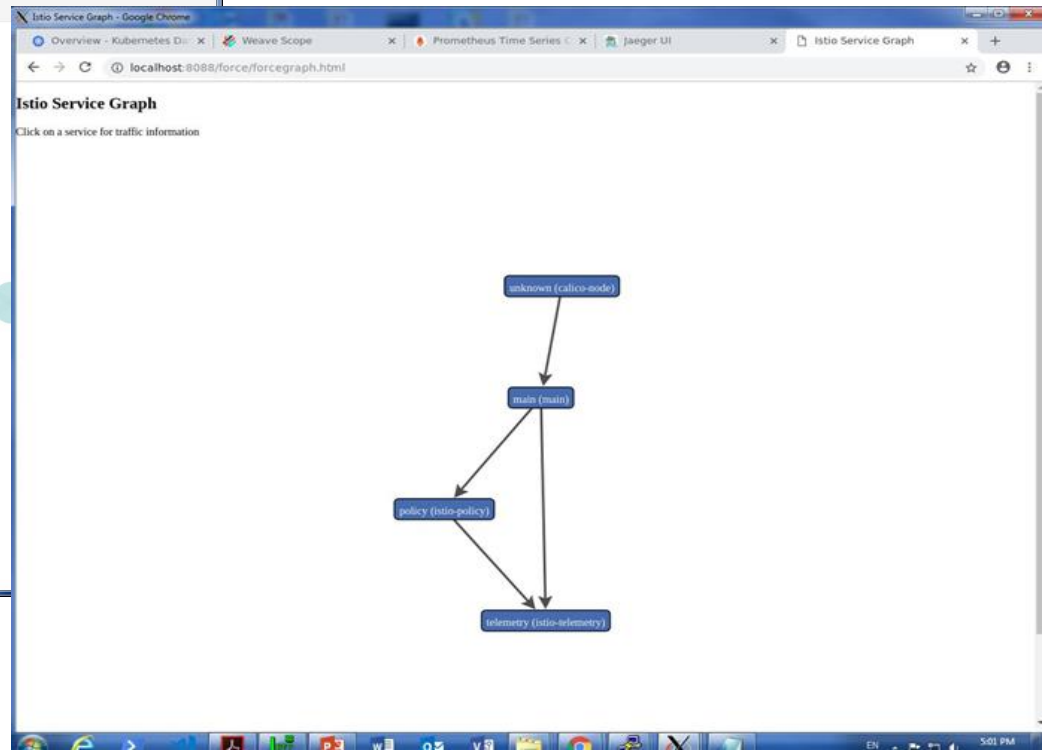



Tracking Spans, events….
in Jaeger



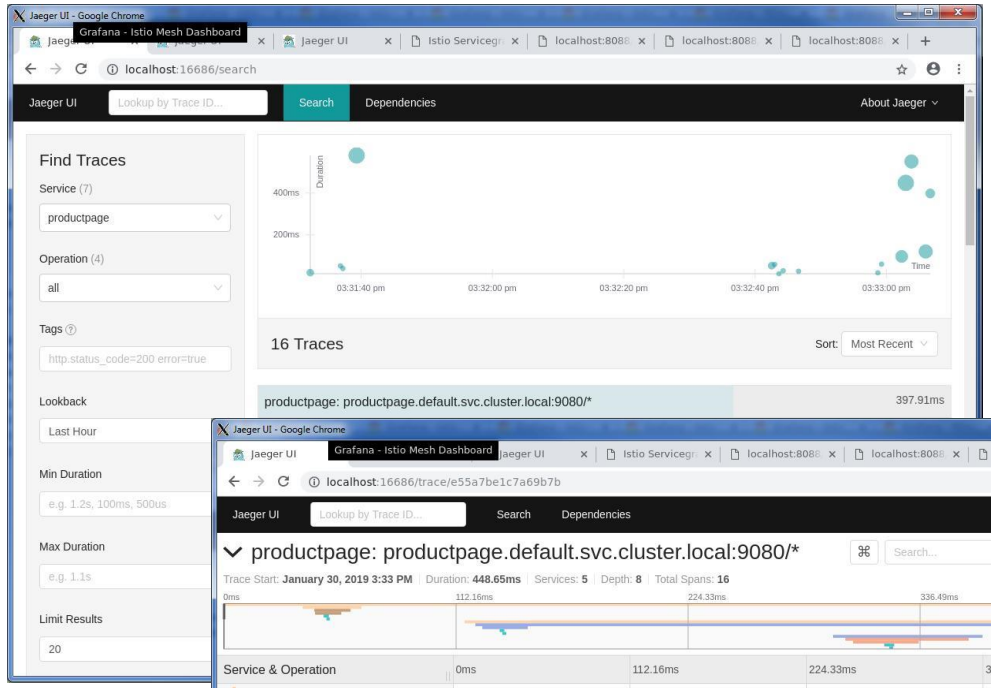

Force Directed Graph

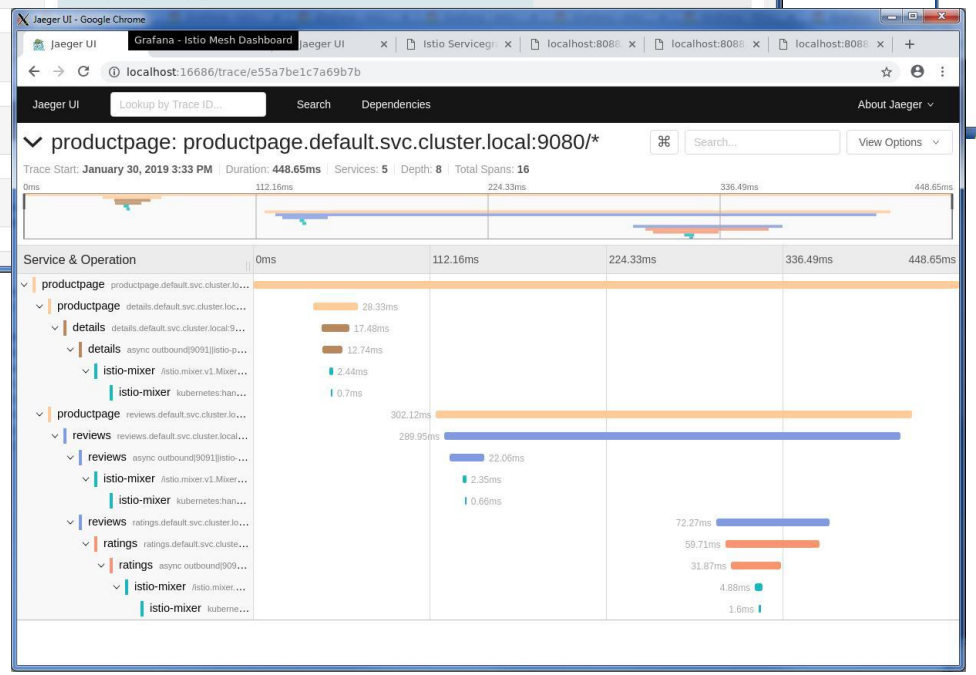
Istio Service Graph



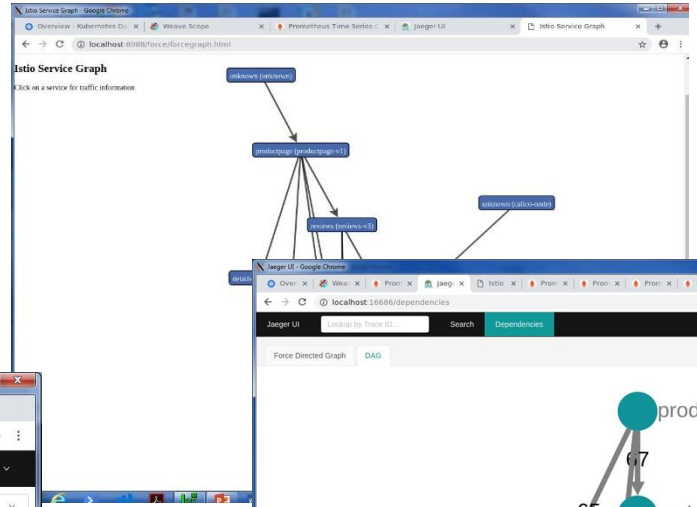
# Distributed/open-tracing (Jaeger) and Service Graph

***Bookinfo*** Application – Tracing/graphing comes out of the box as components of Istio

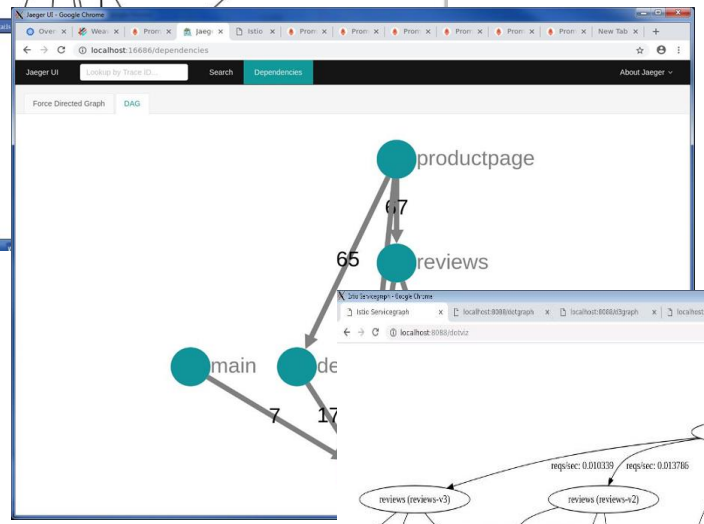


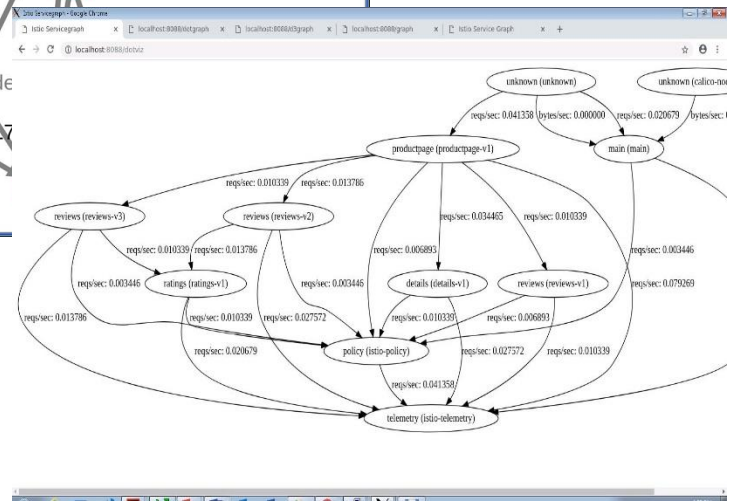

Tracking Spans, events…. in Jaeger




Istio Service Graph




Jaeger directed acyclic graph

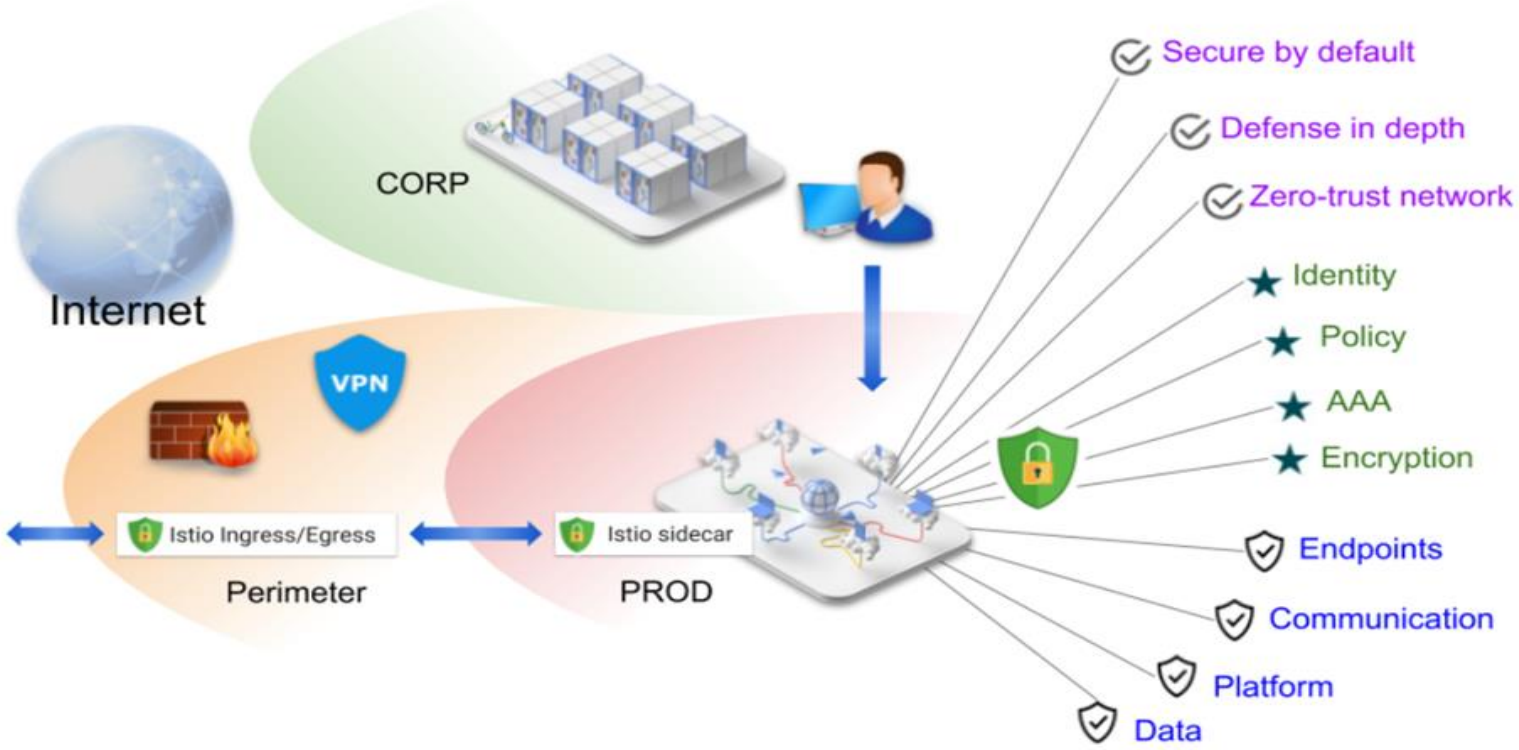



dotviz

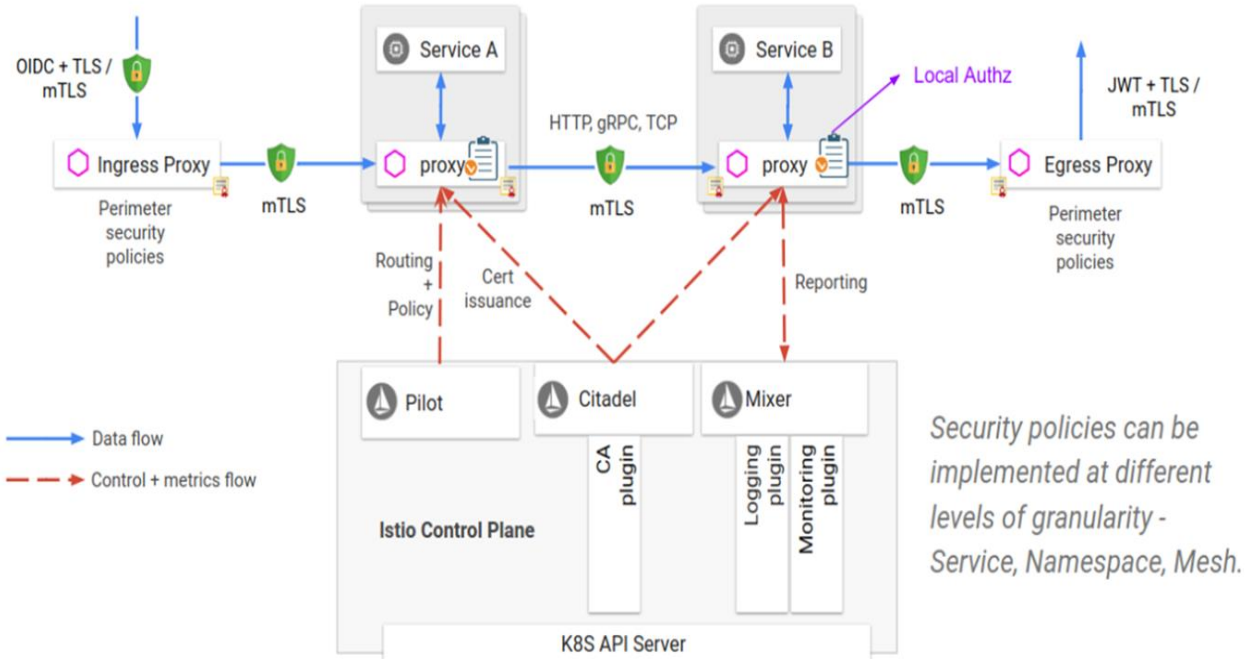
# Security

The Istio security features provide strong identity, powerful policy, transparent TLS encryption, and authentication, authorization and audit (AAA) tools to protect your services and data. The goals of Istio security are:

- Security by default: no changes needed for application code and infrastructure
- Defense in depth: integrate with existing security systems to provide multiple layers of defense
- Zero-trust network: build security solutions on untrusted networks

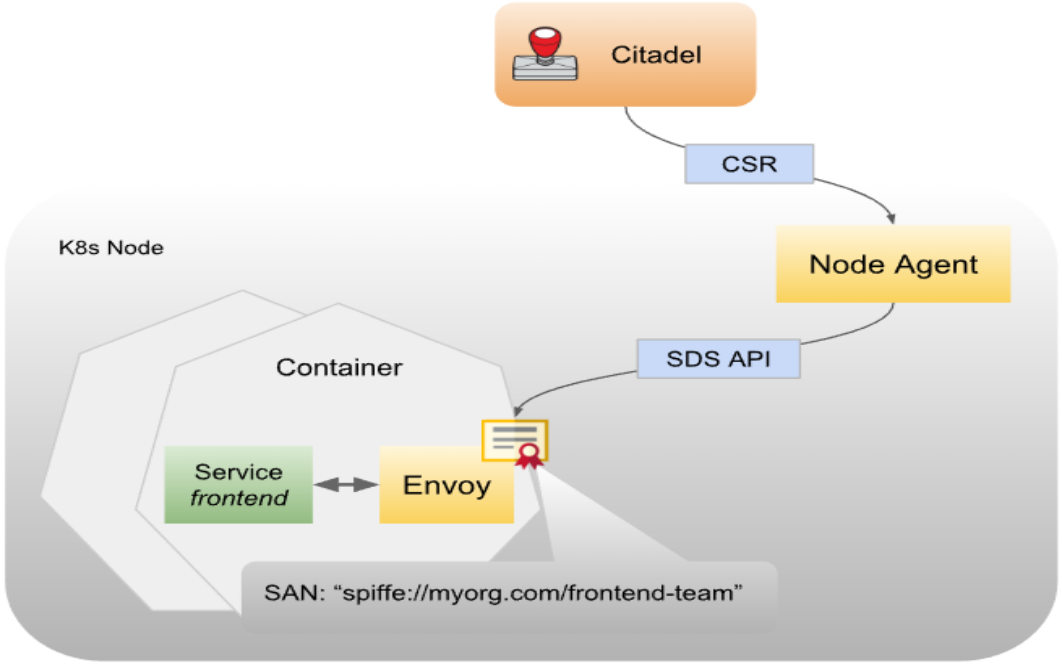


# Security




- Citadel for key and certificate management
- Sidecar and perimeter proxies to implement secure communication between clients and servers
- Pilot to distribute authentication policies and secure naming information to the proxies
- Mixer to manage authorization and auditing

The SPIFFE standard provides a specification for a framework capable of bootstrapping and issuing identities to services across heterogeneous environments.



# Conclusion