ORACLE

Cloud Native Labs

Deploying to a Service Mesh

Using Istio to Simplify Kubernetes Deployments

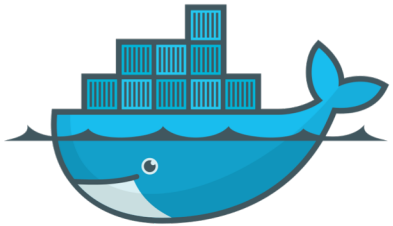
Jesse Butler Cloud Native Advocate, Oracle Cloud Infrastructure. @jlb13

#OracleCloudNative

cloudnative.oracle.com

Level Set

- Microservices
- Kubernetes
- Service Mesh



DevOps, Mother of Invention

- Continuous Integration
- Continuous Delivery
- Microservices
- Containers
- Orchestration

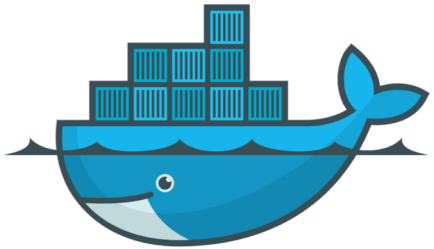
Let’s Talk About Istio

A service mesh that allows us to connect, secure, control and observe services at scale, often requiring no service code modification

Though other options exist, we’ll focus on Istio

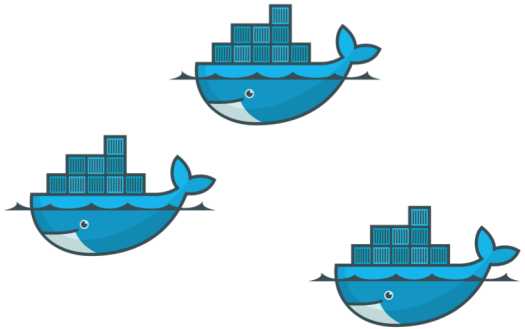
Docker

- Docker changed the way we build and ship software
- Application and host are decoupled, making application services portable
- Containers are an implementation detail, but a critical one



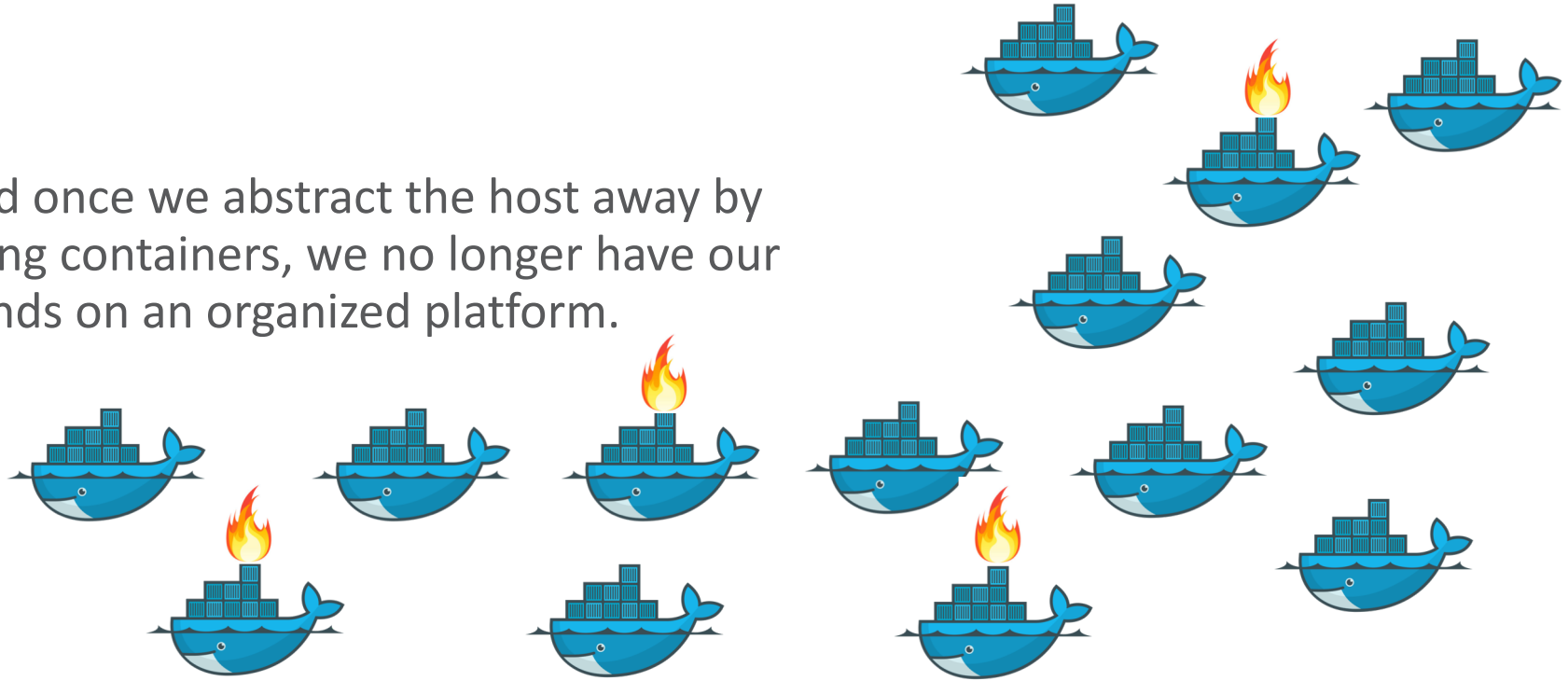
Docker Is a Start

But it’s not the best at running at scale.



Docker Is a Start

And once we abstract the host away by using containers, we no longer have our hands on an organized platform.



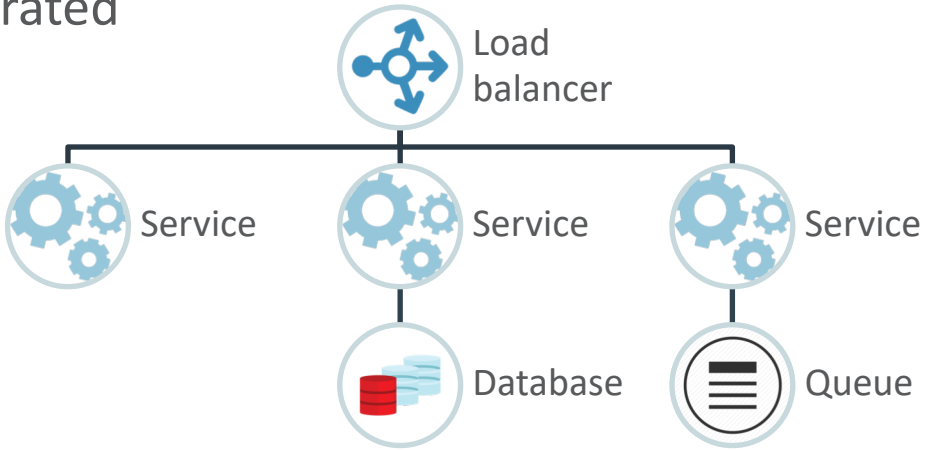
Kubernetes

- Kubernetes to the rescue.
- Scheduling and organization we need for deploying containers at scale
- Really great abstractions of our resources and workloads

Migration from the Old World…

…to Cloud Native Kubernetes Hotness

- Microservices running in orchestrated containers
- Everybody's happy
- What happens now?



Day Two

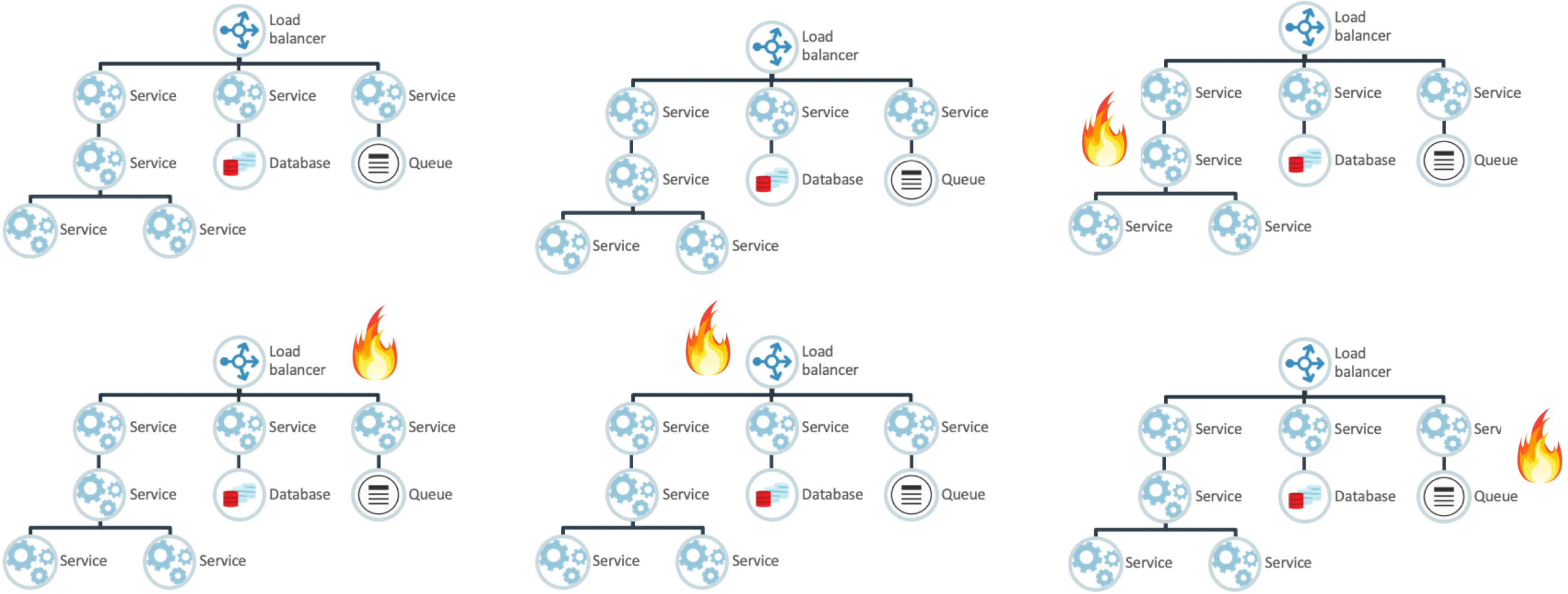


Table Stakes for Services at Cloud Scale

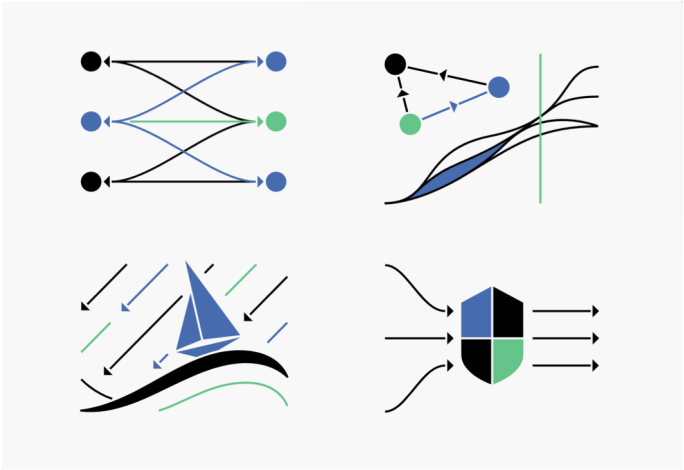
- We require a method to simply and repeatably deploy software, and simply and recoverably modify deployments
- We require telemetry, observability, and diagnosability for our software if we hope to run at cloud scale

Day 2 Solutions

- Ingress and Traffic Management
- Tracing and Observability
- Metrics and Analytics
- Identity and Security

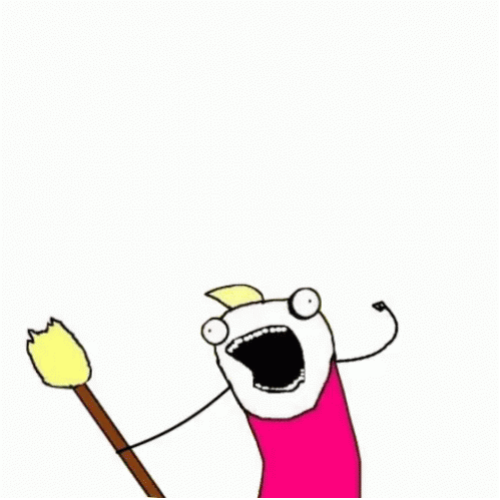
Abstract Requirements

- Traffic Management
- Observability
- Security
- Policy



Service Mesh for All the Things

- This is not a new solution which solves all the world's problems, but a different way to apply existing solutions
- Enables integration of existing (as well as future) best-in-class solutions for All The Things



What Is a Service Mesh?

- Infrastructure layer for controlling and monitoring service-to-service traffic
- A data plane deployed alongside application services, creating a mesh
- A control plane used to manage the mesh

Benefits of a Service Mesh

- Provides DevOps teams a stable and extensible platform to monitor and maintain deployed services
- Simplifies service implementation via service discovery, automated retries, circuit breaking, timeouts and more

Service Mesh is Not an API Gateway

API Gateways deal with north-south traffic, inbound to your cluster

Service Mesh is concerned with east-west traffic, between your services within your cluster

Ambassador

Back to Istio

- Envoy proxy instances in the data plane to own the traffic and create the mesh
- Leverages a sidecar pattern; each service added has a proxy injected into its pod
- This vantage is what gives a service mesh its power, it sees and understands all

Sidecar Proxy

Sidecar Proxy

Istio Features

- Traffic Management
  - Fine-grained control with rich routing rules, retries, failovers, and fault injection
- Observability
  - Automatic metrics, logs, and traces for all traffic within a cluster, including cluster ingress and egress

Istio Features

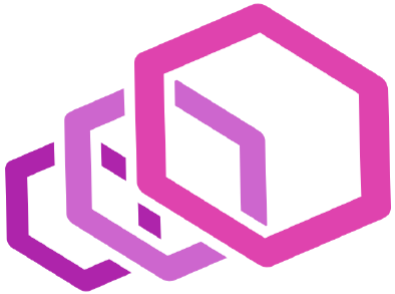
- Security
  - Strong identity-based AuthN and AuthZ layer, secure by default for ingress, egress and service-to-service traffic
- Policy
  - Extensible policy engine supporting access controls, rate limits and quotas

Istio Components

- Envoy
  - Sidecar proxy
- Pilot
  - Propagates rules to sidecars
- Mixer
  - Enforces access control, collects telemetry data
- Citadel
  - Service-to-service and end-user AuthN and AuthZ

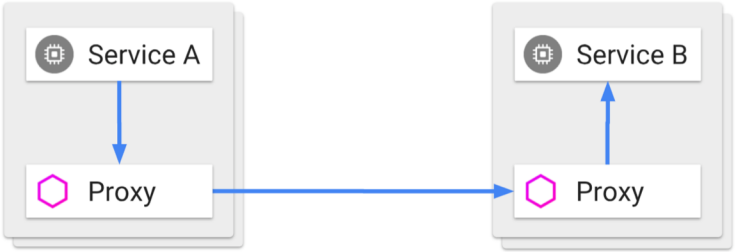
Envoy

High performance proxy which mediates inbound and outbound traffic.

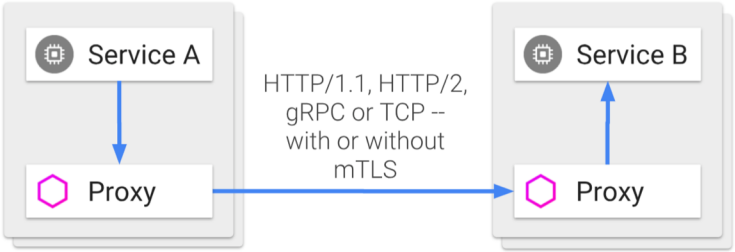


- HTTP/2 and gRPC proxies
- Dynamic service discovery
- Load balancing
- TLS termination
- Circuit breakers
- Health checks
- Split traffic
- Fault injection
- Rich metrics

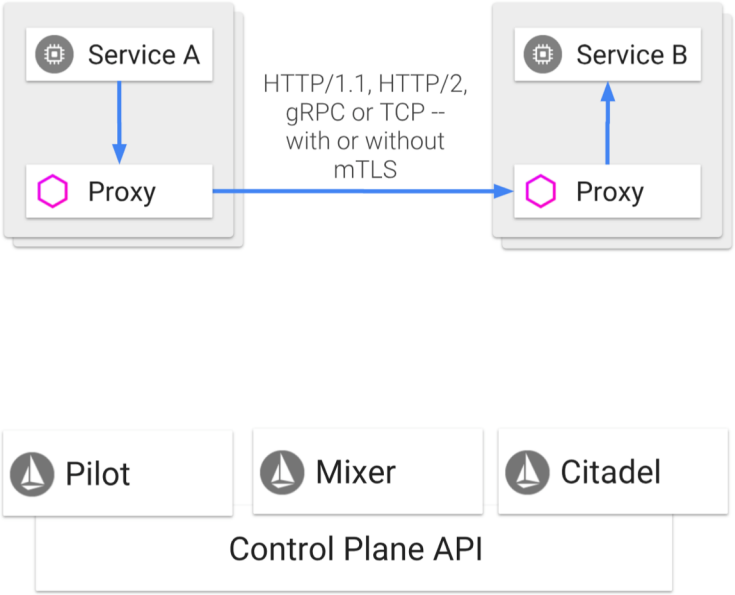
Istio Architecture



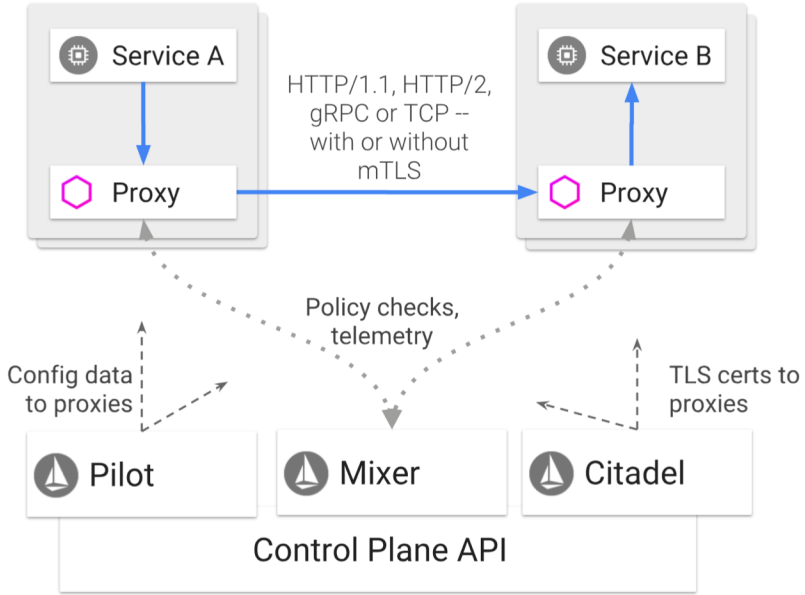
Istio Architecture



Istio Architecture



Istio Architecture



Using Istio

- istioctl, cli for mesh administration
- Kiali – BUI Dashboard and Control
- Configure services with typical Kubernetes workflows - CRDs
- Sidecar auto-injection is optional on a per-namespace basis

Demo

- Integrated observability
  - Kiali
  - Grafana
  - Jaeger

Kubernetes Objects

- Pods
  - Unit of deployment
- Deployments
  - What and how many to deploy
- Services
  - Abstraction of a set of pods and access to them

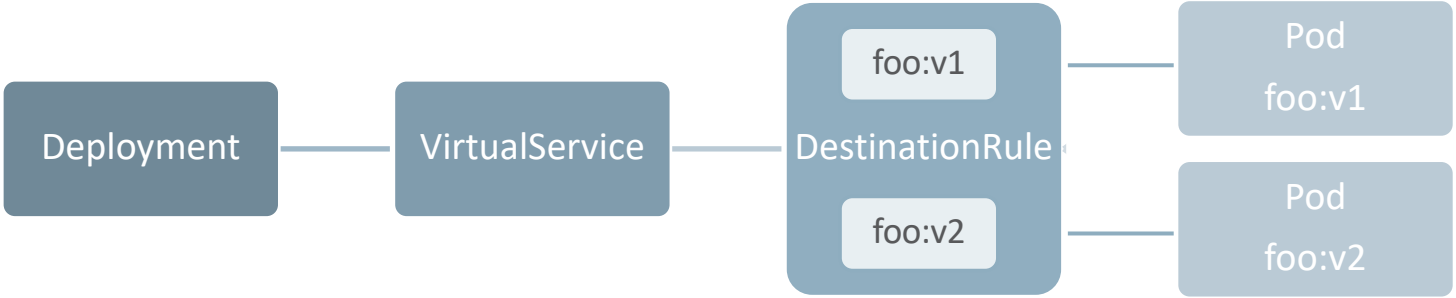
Istio Objects

- Virtual Service
  - Routes mesh traffic to a destination or subset
- Destination Rules
  - Sets policies on routed traffic e.g TLS or LB
- Gateways
  - Route ingress or egress traffic to and from mesh

Traffic Shifting Basics

- Identify subsets in Deployments by using labels (e.g. a subset per version)
- Configure Virtual Service to route traffic based upon the subsets
- Use Destination Rules to set policies related to the traffic, such as load balancing

Simple Traffic Shifting v1->v2



- ‘foo’ service routed through ‘foo’ VirtualService
- DestinationRules for ‘foo:v1’ and ‘foo:v2’ Pods

Updates with Kubernetes

- RollingUpdate rolls out new pods and kills off old pods, tunable with maxSurge and friends
- It does this by thumping them on the head with SIGTERM
- And continues to throw work at them
- And throws work at new pods before they are ready
- And the LoadBalancer configuration is updated concurrently or slightly later and yep this isn't so hot

Updates with Kubernetes The Right Way

- Implement Readiness Probes
- Ensure everyone can handle a SIGTERM
- Use a preStop lifecycle hook
- Automate all of this and don’t break it

Simplifying CD Life with Istio

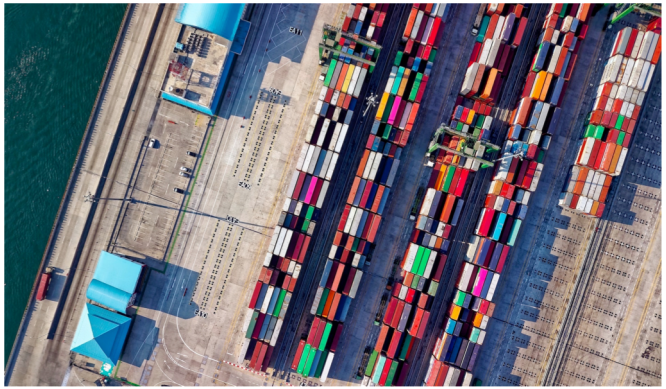
- Create a Service, and a Deployment for each version of software
- Use version labels to create subsets for each of the desired versions
- Set up DestinationRules to route traffic to the each of the versions
- Route the traffic based upon needs, using destination subset in VirtualService

Demo

- Simple Traffic Shifting
  - Migrate traffic from v1 to v2

Leveraging the Pattern

- We can manage traffic in an informed way
- We can take advantage of zero-downtime changes in routing between versions
- We can automate deployments of any kind
  - Canary deployments
  - Blue/Green deployments
  - Whatever we want



Demo

- Blue/Green Deployment
  - Move traffic from v1 to v2 gradually using weights

A Full Solution: Flagger

- Kubernetes operator that automates promotion of canary deployments
- Uses Istio for traffic shifting between service versions
- Uses Prometheus metrics for canary analysis
- Extension via webhooks for additional acceptance test, load test, etc

Demo

- Canary Deployments
  - Automated Canary Deployments with Flagger

Test in Prod with Dark Launches

- Istio offers Traffic Mirroring on a weighted basis as well, providing for Dark Launches
- An under-test version of a service can be deployed and production traffic routed to it
- Testing in production with production data

Demo

- Dark Launch
  - Use Traffic Mirroring for a Dark Launch

ORACLE

Cloud Native Labs

Thanks!

Twitter: @jlb13

cloudnative.oracle.com

cloud.oracle.com/tryit

Sign-up available for $500 trial
(for a start)