ORACLE

Cloud Native Labs

Microservices at Scale

Next Steps in Kubernetes with Service Mesh

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

#OracleCloudNative

cloudnative.oracle.com

The Old World

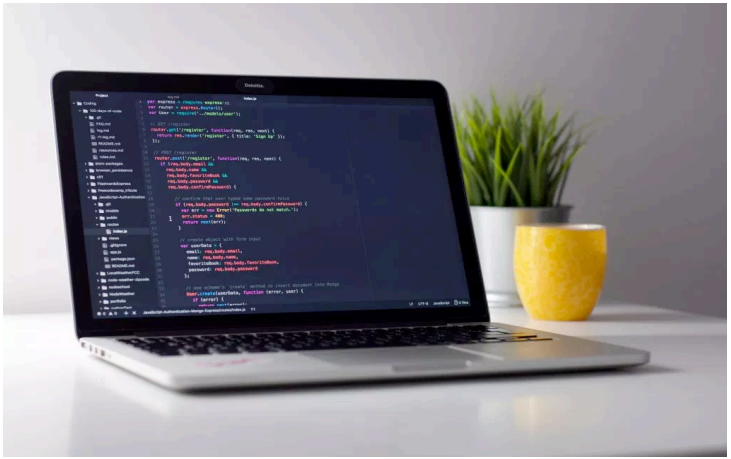
- Proprietary systems and software were bundled and sold atomically
- Independent silos arose per vendor, each with ecosystems and vendors
- Systems analysts surfaced system data and implemented improvements

More Recent History

- There were a lot of moving parts in the typical Old World IT organization
- The advent of web applications made time to market a keystone metric
- DevOps arose as a means of reducing friction between where software is created and where it is deployed

Advent of DevOps

- DevOps brings the concerns of development and ops together
- Goal is to create a system which delivers customer satisfaction with as little friction as possible
- DevOps is as much a cultural shift as it is technical

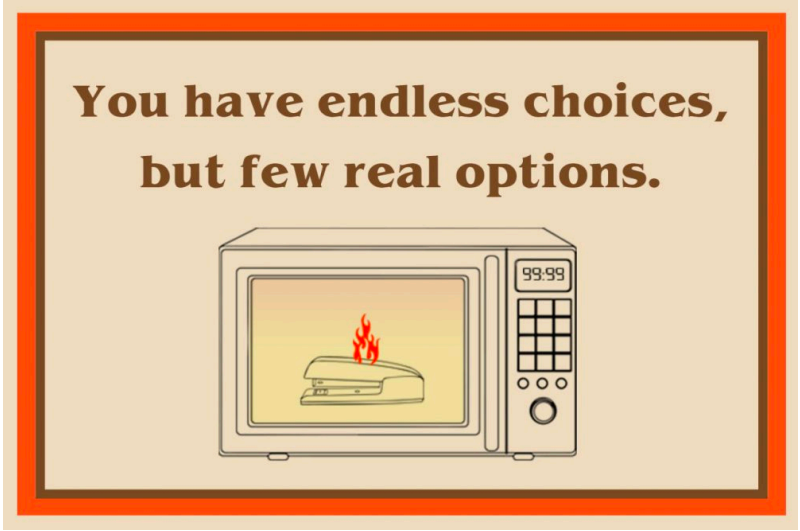


DevOps, Mother of Invention

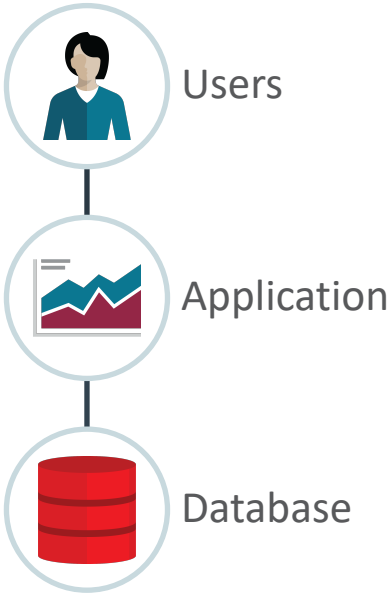
- Microservices
- Continuous Integration
- Continuous Delivery
- Containers
- Cloud Adoption

Cloud Native

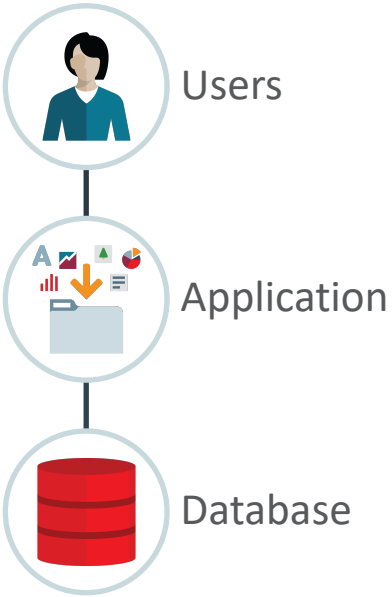
- Migrating to the cloud is more than renting someone else's computers
- Massive migration offers an opportunity for change
- Cloud Native practices align with DevOps practices
- This is proven ground, thankfully



Monolithic Applications



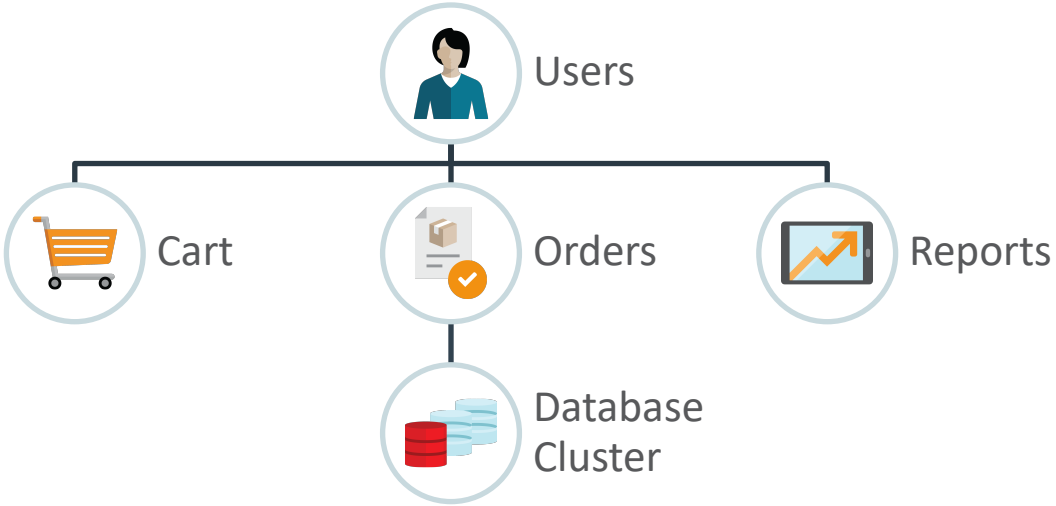
Monolithic Applications



Microservices

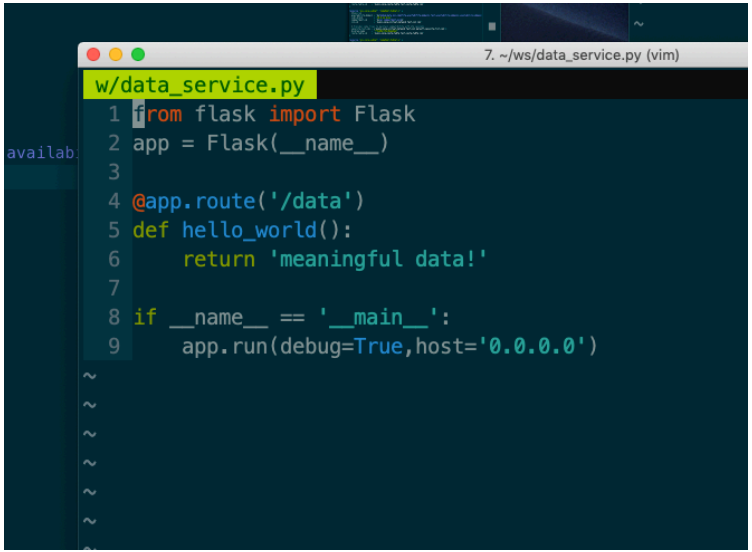
- Microservices are the de facto standard for cloud native software
- Microservices allow development teams to deploy portable and scalable applications
- Microservices can be difficult to manage and monitor, putting burden on Ops and DevOps alike

Microservices



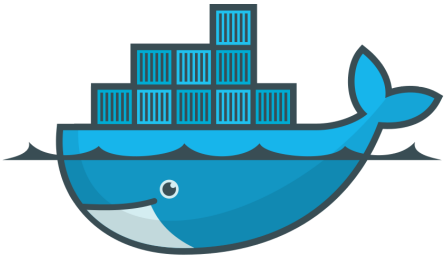
Adopting Microservices

- Microservices do one thing as simply as possibly
- Promotion of single responsibility principle (or the UNIX Philosophy)
- Microservices should be idempotent and stateless
- Applications can and do have state, services should be stateless



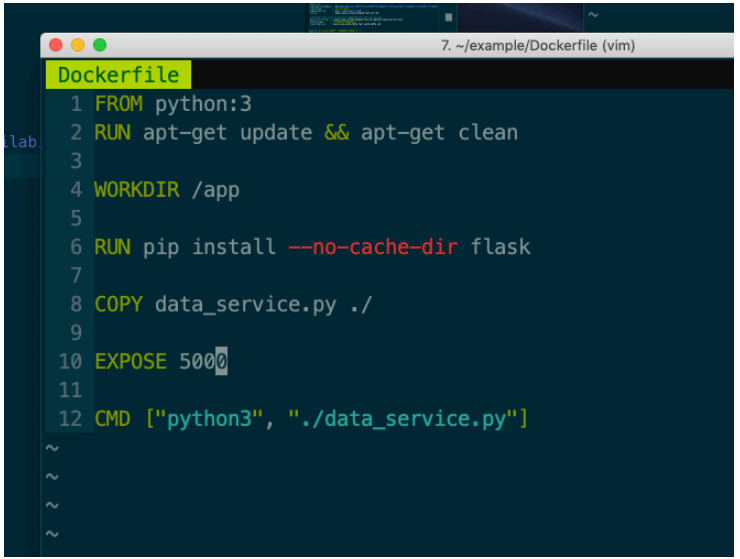
Docker

- 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



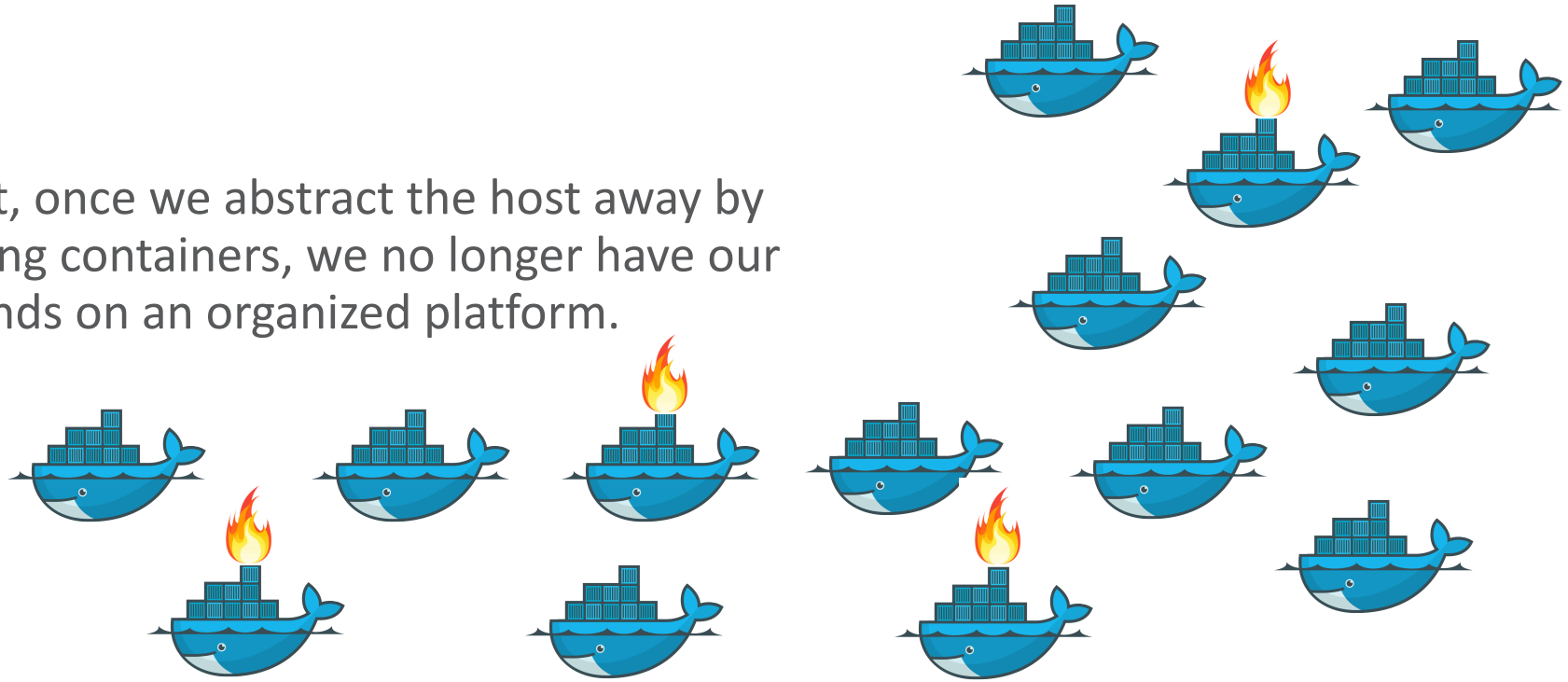
Using Docker

- Docker is used in production at massive scale every day
- Interactively, a development utility for creating containers and container images
- Dockerfile defines content of a container and its runtime configuration
- ‘`docker build. –tag data_service:1.0`’



Docker Is a Start

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



Kubernetes

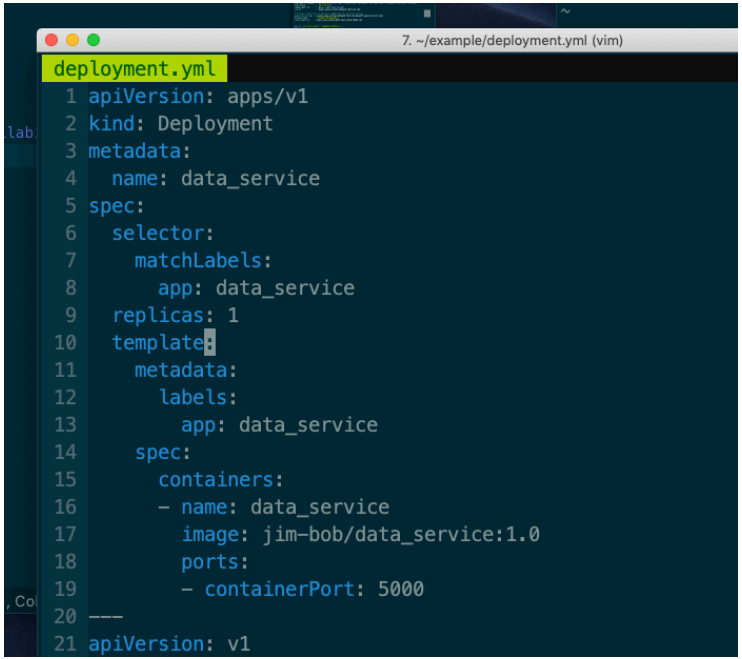
Kubernetes provides abstractions for deploying software in containers at scale

Kubernetes as a Platform

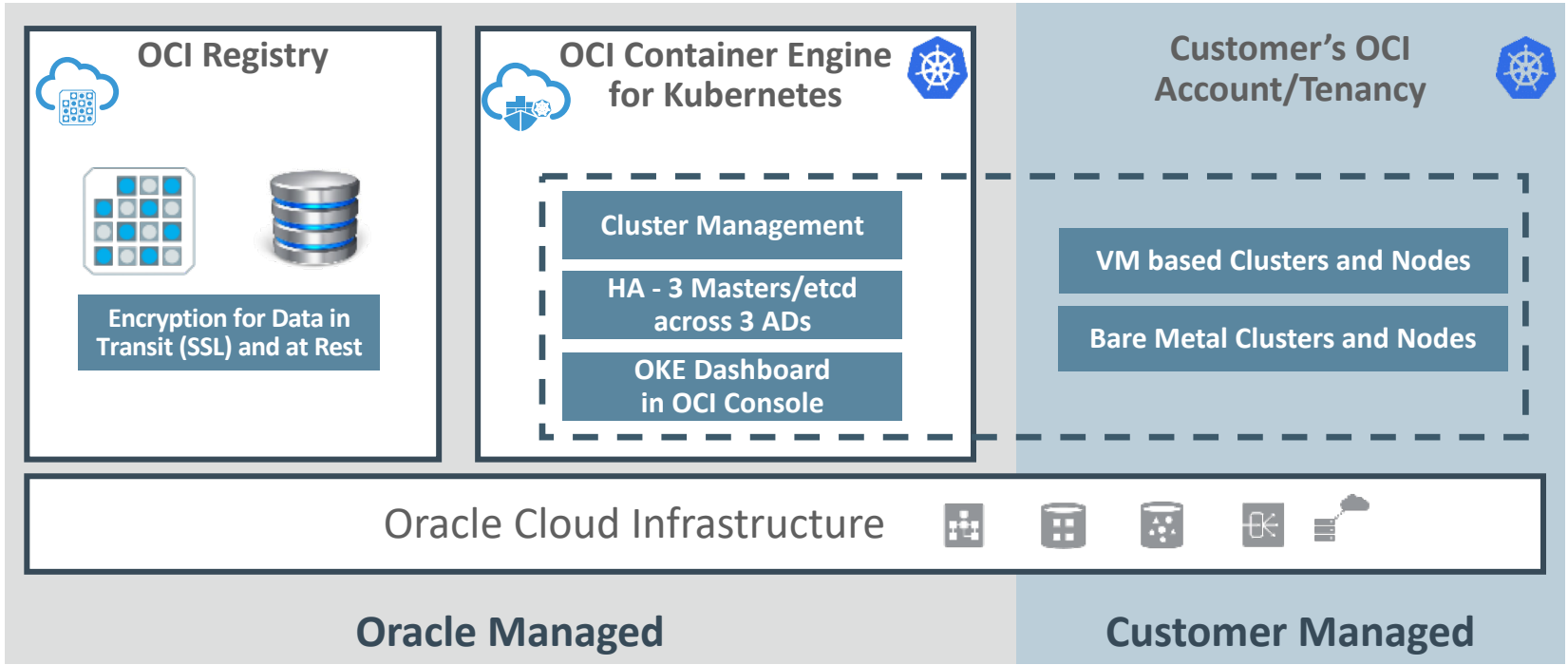
- Infrastructure resource abstraction
- Cluster software where one or more masters control worker nodes
- Scheduler deploys work to the nodes
- Work is deployed in groups of containers

Using Kubernetes

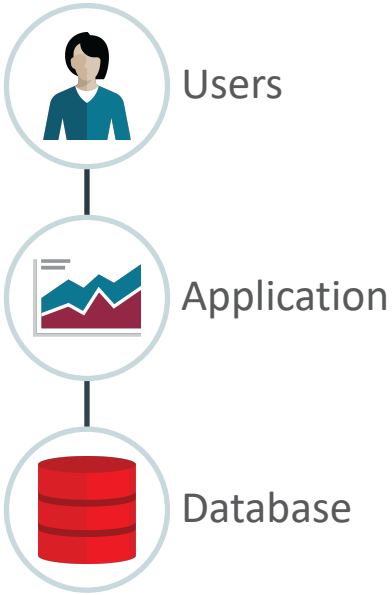
- Deployments are defined in YAML
- We define what images to use to create our containers, configuration elements, how many instances to run
- Kubernetes makes it happen, and keeps it all running as defined
- ‘`kubectl create -f`’ and glory awaits



Working with OKE and OCIR on OCI

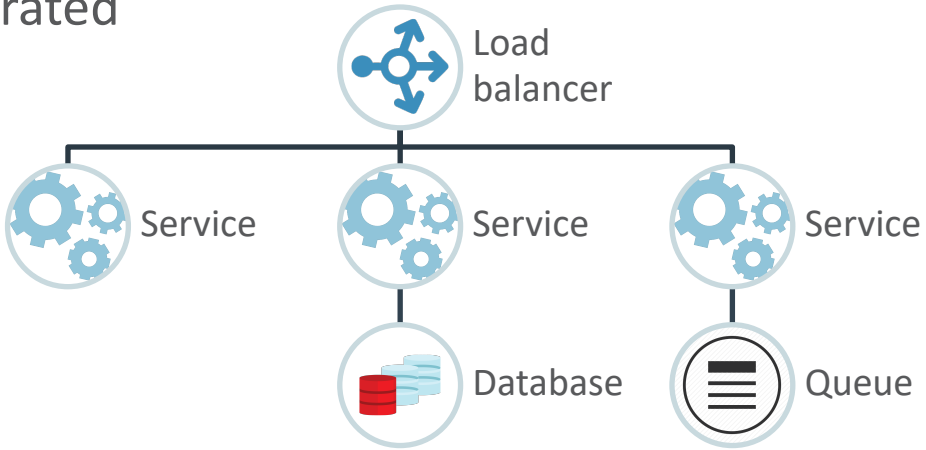


Migration from the Old World…



…to Cloud Native Kubernetes Hotness

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



Day Two

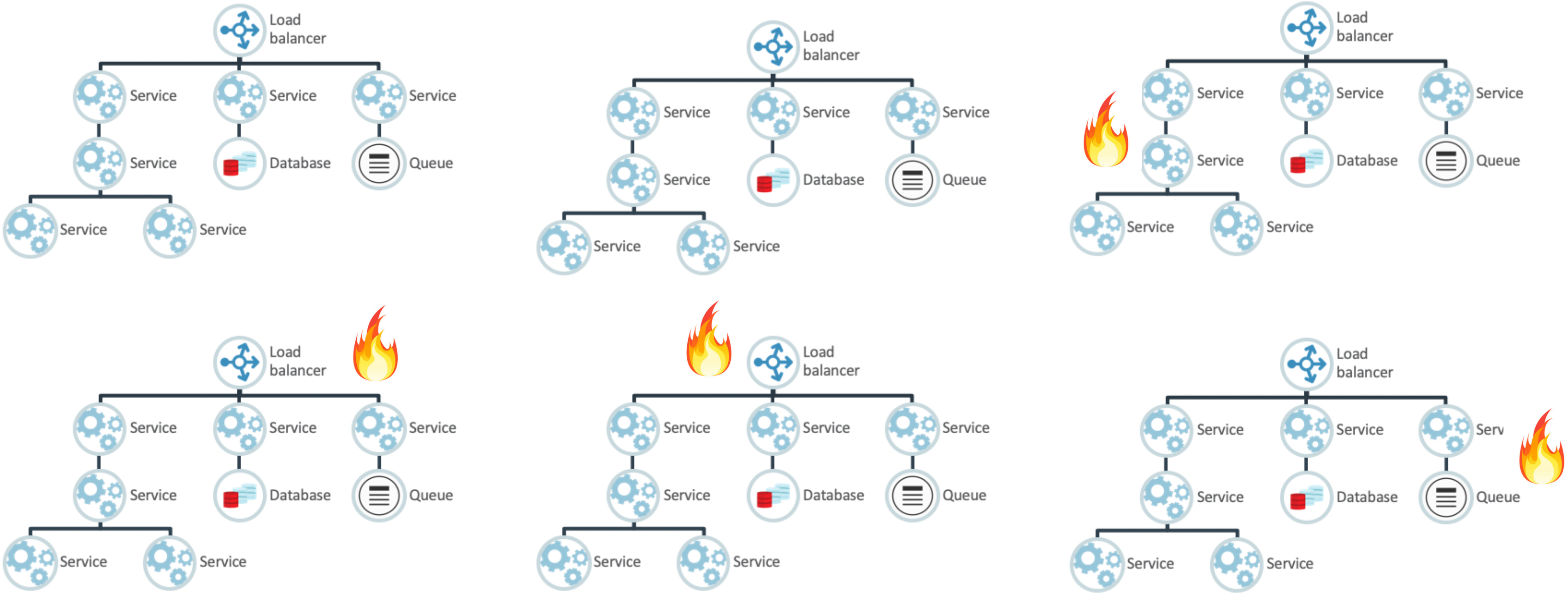


Table Stakes for Services at Cloud Scale

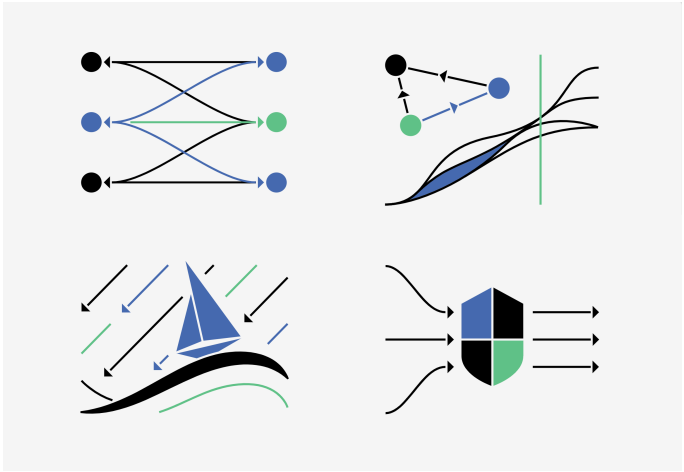
- We require a method to simply and repeatably deploy software and reliably modify those deployments
- We require telemetry, observability, and diagnosability for our software if we hope to run at cloud scale

Day 2 Solutions

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

Abstract Requirements

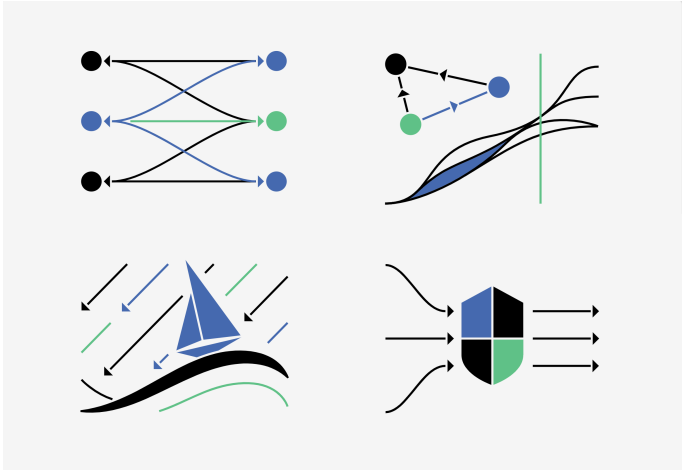
- Traffic Management
- Observability
- Security
- Identity & Policy



Hard Things are Hard

These are Hard Problems, and some software may address one of them well.

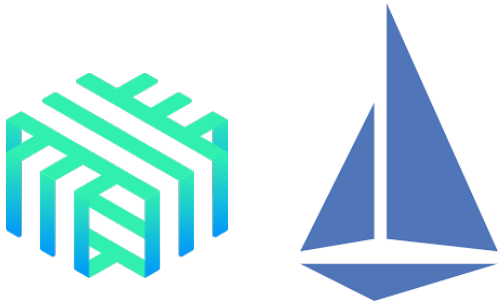
Service mesh has an opportunity to address them all.



Let’s Talk About Service Mesh

Connect, secure, control and observe services at scale, often requiring no service code modification

Though many options exist, Linkerd and Istio are the two main projects



Service Mesh

- Infrastructure layer for controlling and monitoring service-to-service traffic
- Data plane deployed alongside application services, control plane used to manage the mesh
- Greatly simplifies service implementation offering transparent service discovery, automated retries, 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

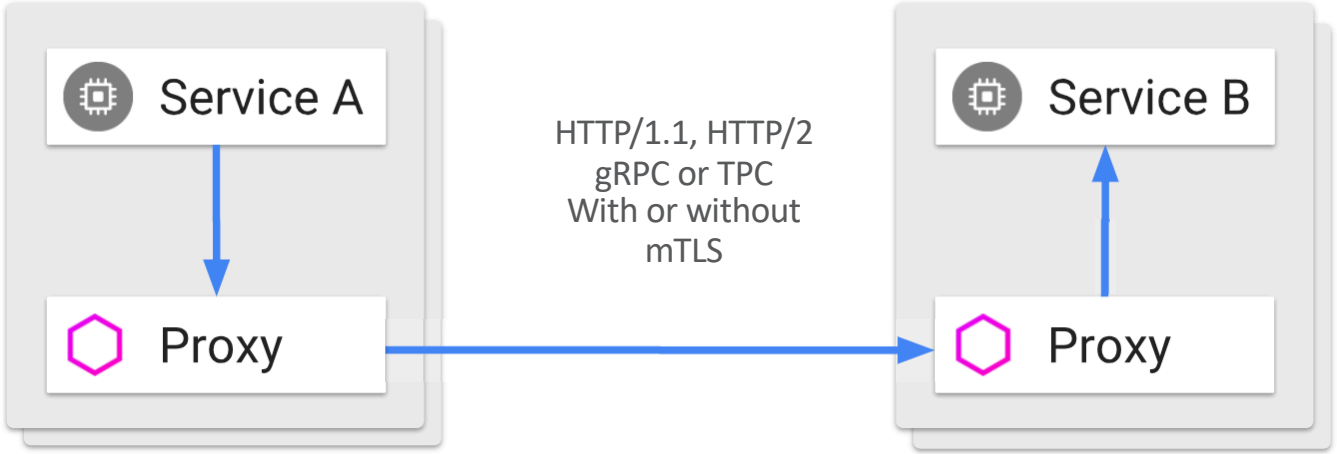
Service Mesh Architecture

- Both Istio and Linkerd use a sidecar pattern, adding a proxy container for each pod added to the mesh
- Each proxy instance manages traffic for its pod, and is fully configurable
- This vantagepoint is what gives a service mesh its power – it sees and knows all

Sidecar Proxy

Sidecar Proxy

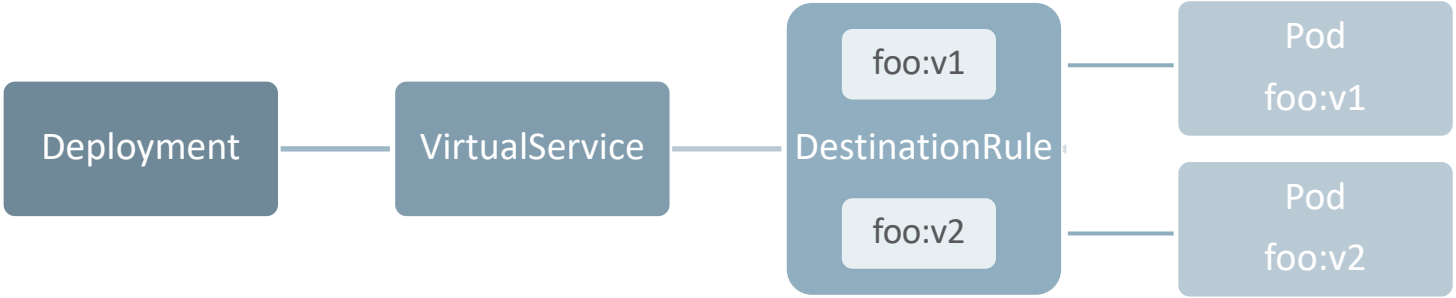
Sidecar Proxy



Traffic Management

- Each service deployed within the mesh has a proxy instance
- Each proxy can be fully configured based upon our needs
- Effectively, we can move and manipulate traffic as needed

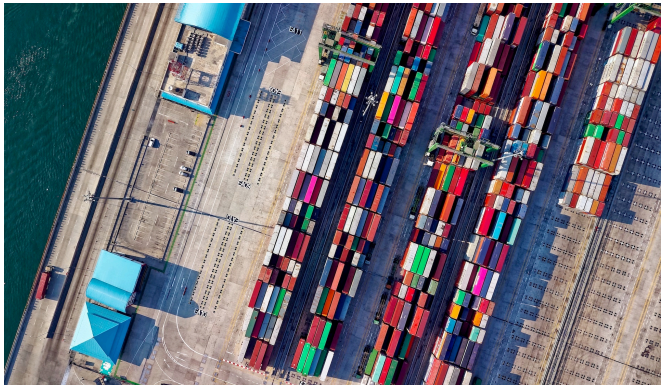
Traffic Management Details with Istio



- ‘foo’ service routed through ‘foo’ VirtualService
- DestinationRules for ‘foo:v1’ and ‘foo:v2’ pods, with weights

Leveraging Traffic Shifting

- Manage traffic in an informed way
- 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



Observability

- Metrics
  - Aggregate data regarding the behavior of a thing over time
- Tracing
  - Instrumentation which provides an instance of an action, traversing the entire stack
- Logging
  - Developer breadcrumbs we leave to give context for a certain code path

Triaging Issues

- Metrics must be implemented and scraped for analytic use
- Tracing are implemented on a per-span basis
- Logs are provided by the developer, a gift they give their future selves

Service Mesh Brings Observability Gifts

- All traffic in the mesh is routed through the proxies
- Metrics and traces can be taken “for free”, with no modifications to code
- Specific traces and metrics must be implemented of course
- A lot of issues can be triaged with boundary tracing

Security

- Deploying services in containers requires careful provisioning, build and deployment practices
- There are options to leverage in both CI/CD and registry scanning
- Once services are deployed in the wild, they are on their own

Security

- Istio and Linkerd are capable of creating a zero-touch, zero-trust network
- Services within your cluster authenticate via the mesh
- Leveraging mTLS, the cluster is transparently hardened and protected from many types of attacks

Let’s Look at Istio

Istio a service mesh for Kubernetes that allows us to connect, secure, control and observe services at scale, often requiring no service code modification.

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

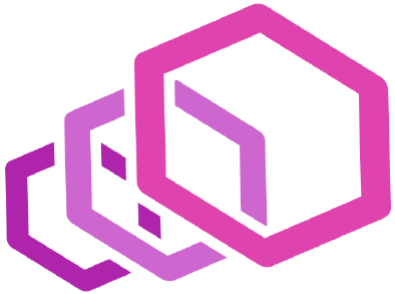
- Security
  - Strong identity-based AuthN and AuthZ layer, secure by default for ingress, egress and service-to-service traffic
- Policy
  - Extensible policy engine supporting access controls, rate limits and quotas

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

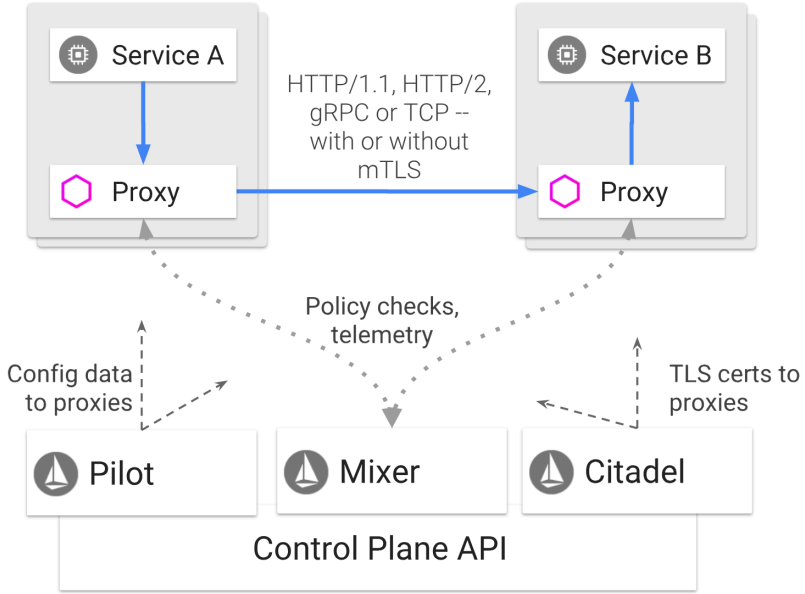
Envoy

High performance proxy which mediates inbound and outbound traffic.



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

Istio Architecture



Using Istio

- istioctl, cli for mesh admin
- Kiali – dashboard BUI
- Configure services with typical Kubernetes workflows - CRDs
- Sidecar auto-injection is optional

Let’s Look at Linkerd

Linkerd is an ultralight service mesh for Kubernetes and other orchestration platforms

Linkerd2 has a wholly reimplemented proxy and is built for low latency and massive scaling

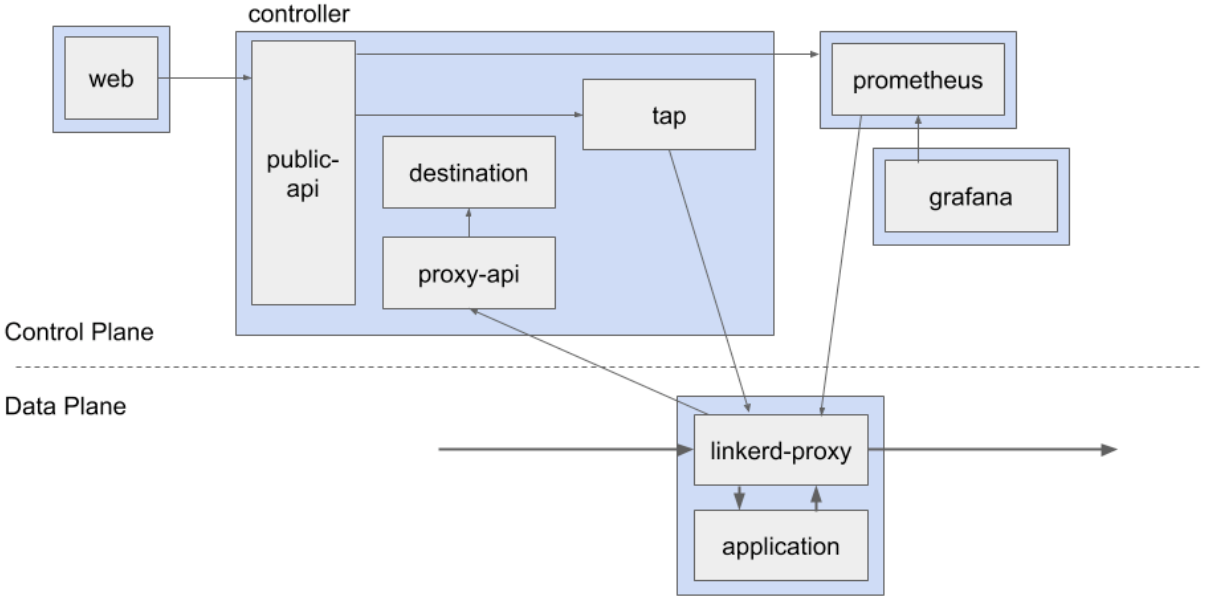
Linkerd Features

- Deep runtime diagnostics
  - Comprehensive suite of diagnostic tools, including automatic service dependency maps and live traffic samples
- Actionable service metrics
  - Allows you to monitor *golden metrics*—success rate, request volume, and latency—for every service and define response

Linkerd Features

- Simple, minimalist design
  - No complex APIs or configuration. For most applications, Linkerd will "just work" out of the box
- Ultralight and ultra fast
  - Built in Rust, Linkerd's data plane proxies are incredibly small (<10 mb) and blazing fast (p99 < 1ms)

Linkerd Components

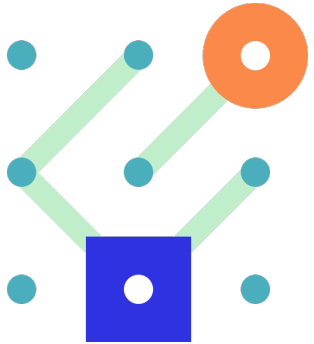


Using Linkerd

- Linkerd CLI utilities
  - Routes, stats, tap, profiles
- Unified dashboard
- Configure services with typical Kubernetes workflows - CRDs
- Automated sidecar injection optional

Linkerd or Istio? Or Aspen Mesh or Consul or…

- Superficially speaking…
  - Istio for depth and features
  - Linkerd for simplicity and ease-of-use
  - Others might be interesting as well
- Service Mesh Interface Specification may help lessen the burden
- Any choice is better than no choice!



ORACLE

Cloud Native Labs

Thanks!

cloudnative.oracle.com

@jlb13