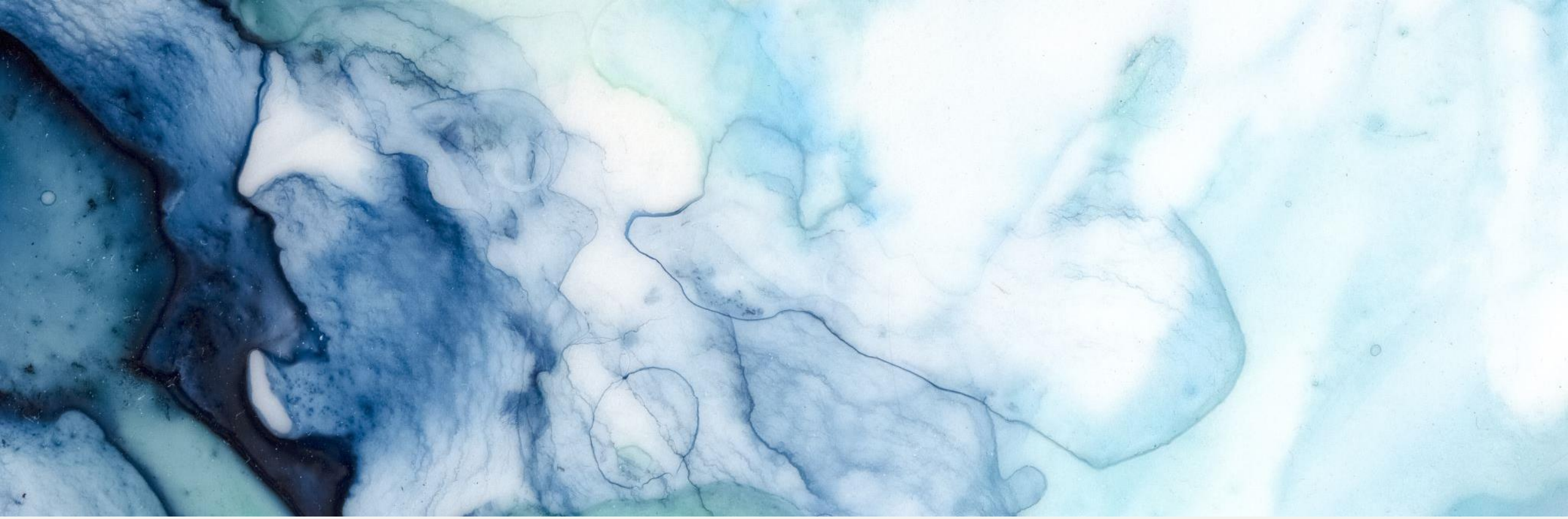


Microservices Deployment in Cloud Service Mesh

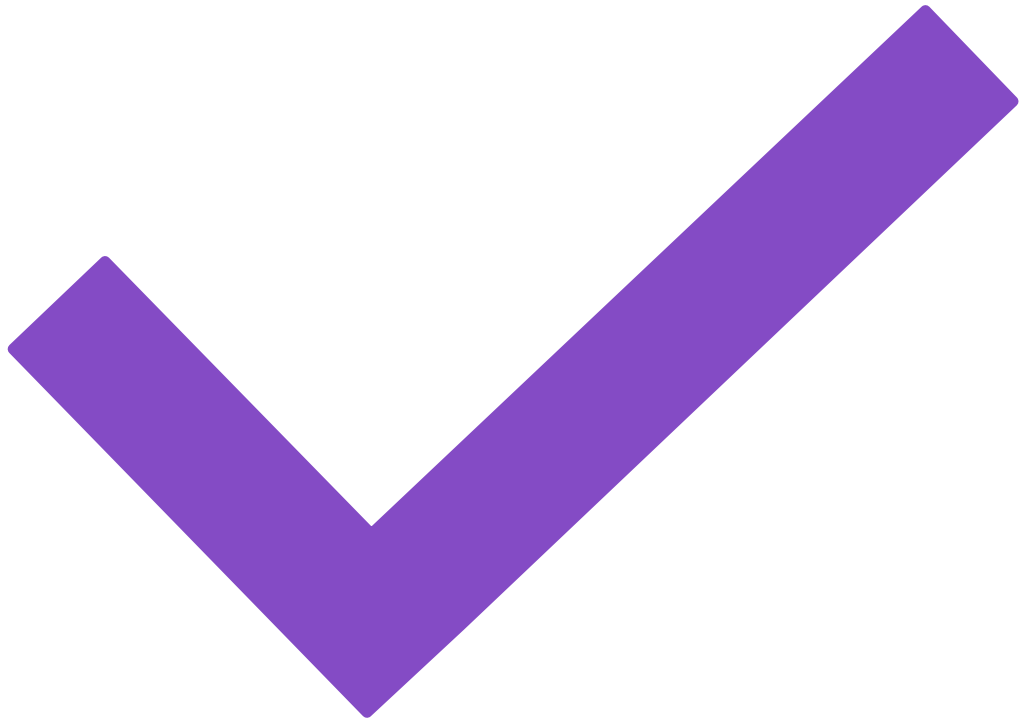
ECE 4150

Spring 2024

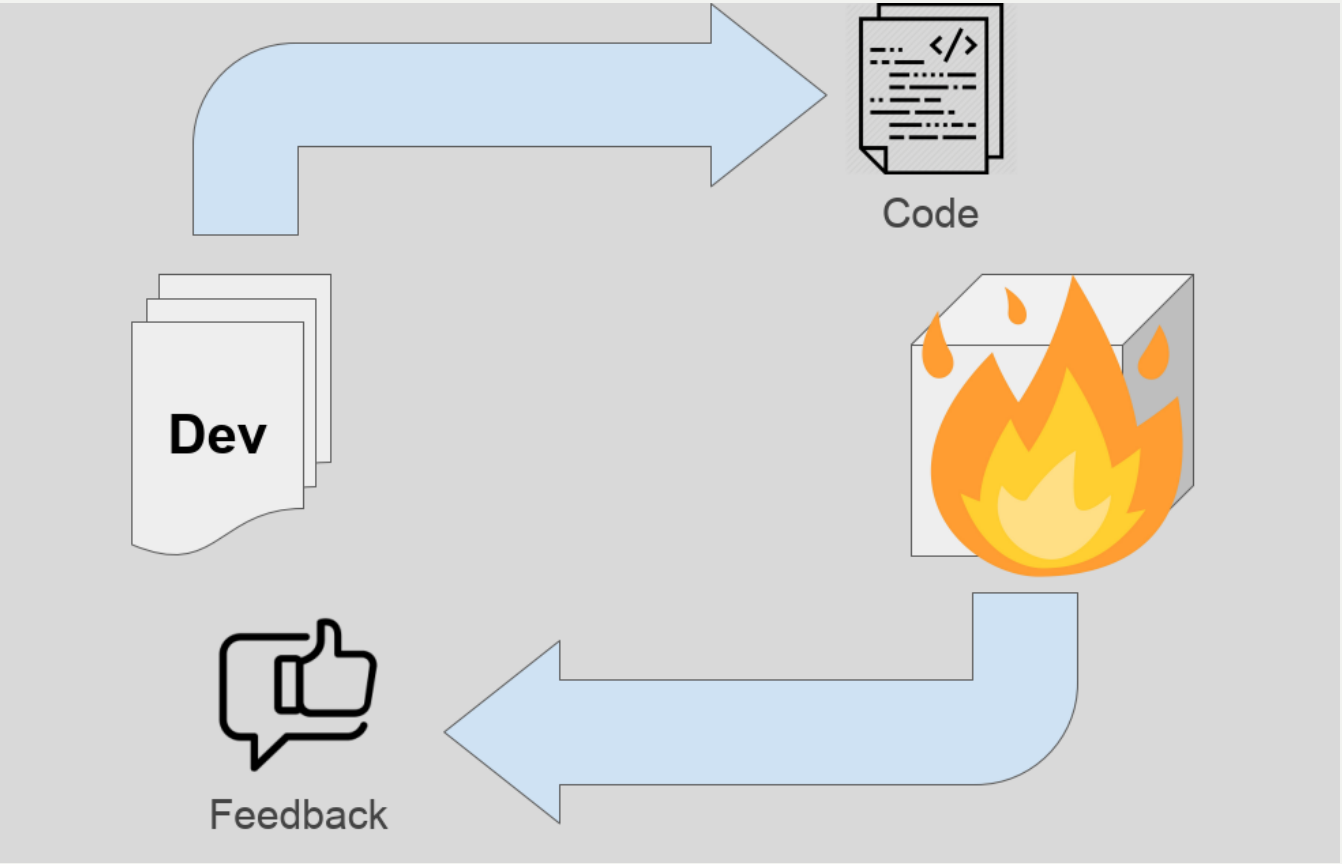
Madisetti

Outline

- Problems with Microservices
- Kubernetes Model
- Istio Service Mesh
- Features that Istio Provides
- Examples



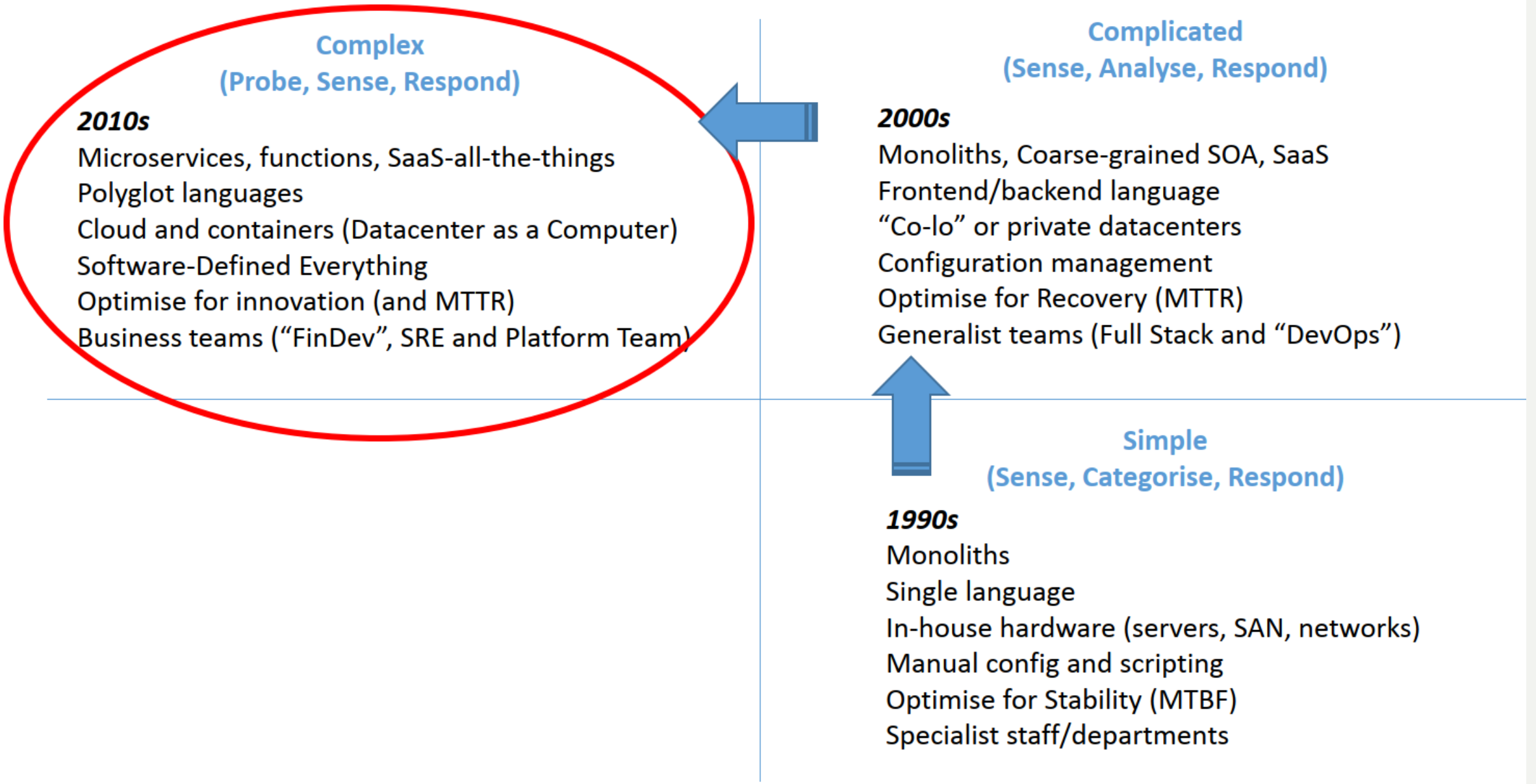
Ideal world vs practice



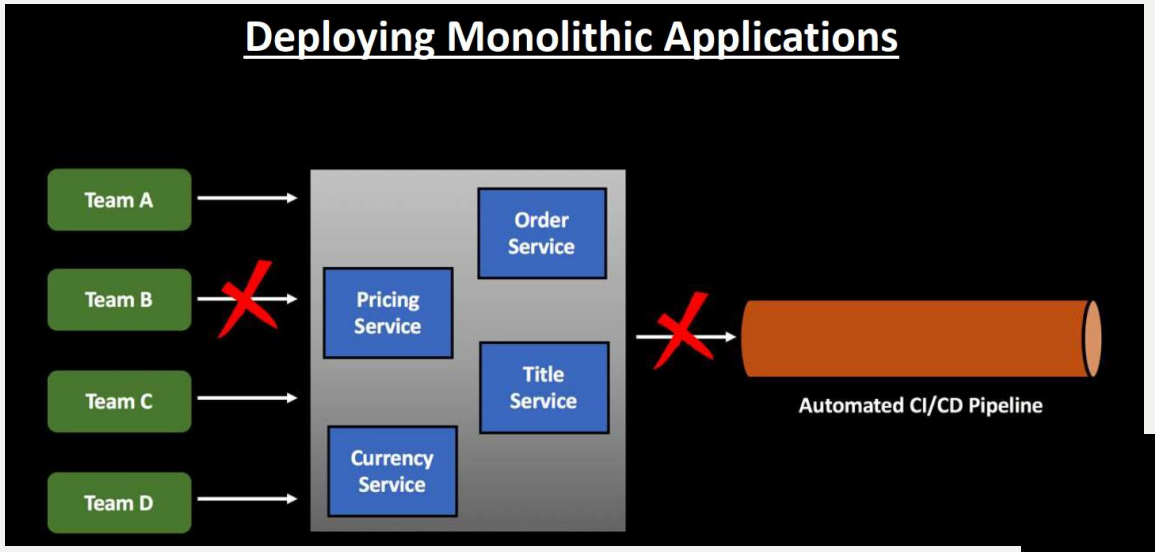
Fallacies of Distributed Computing

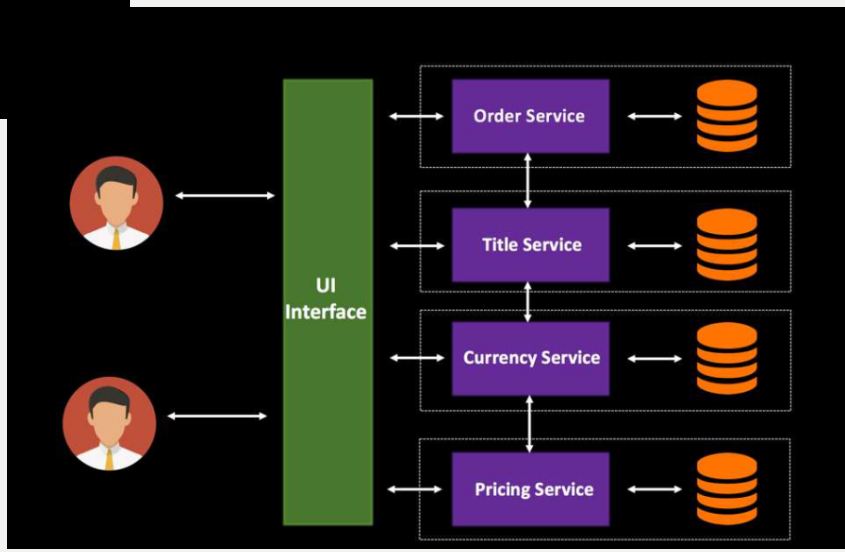
1. The network is reliable.
2. Latency is zero.
3. Bandwidth is infinite.
4. The network is secure.
5. Topology doesn't change.
6. There is one administrator.
7. Transport cost is zero.
8. The network is homogeneous.

https://www.somethingsimilar.com/2013/01/14/notes-on-distributed-systems-for-young-bloods/


Complex
(Probe, Sense, Respond)
2010s
Microservices, functions, SaaS-all-the-things
Polyglot languages
Cloud and containers (Datacenter as a Computer)
Software-Defined Everything
Optimise for innovation (and MTTR)
Business teams ("FinDev", SRE and Platform Team)
Complicated
(Sense, Analyse, Respond)
2000s
Monoliths, Coarse-grained SOA, SaaS
Frontend/backend language
"Co-lo" or private datacenters
Configuration management
Optimise for Recovery (MTTR)
Generalist teams (Full Stack and "DevOps")
Simple
(Sense, Categorise, Respond)
1990s
Monoliths
Single language
In-house hardware (servers, SAN, networks)
Manual config and scripting
Optimise for Stability (MTBF)
Specialist staff/departments

Microservices





Shorter Development Cycle

Faster Deployments

Highly Scalable

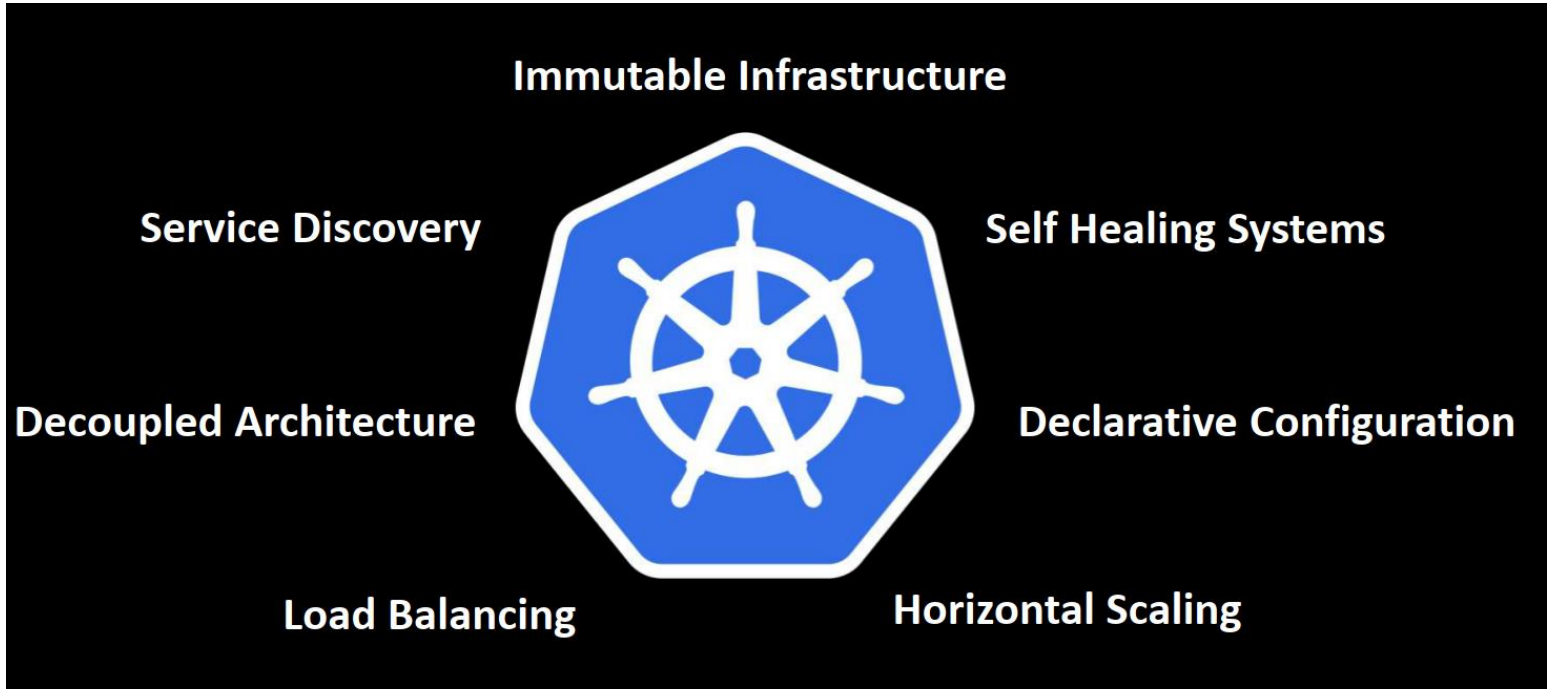
Right Technology Stack

Business Domain Driven

Resiliency & Observability

High Cohesion & Loose Coupling

Kubernetes (K8S) provides the Infrastructure to Microservices



What is a *Tech Stack?*

A ***tech stack*** is a combination of tools, software, and frameworks with which you build your microservices.

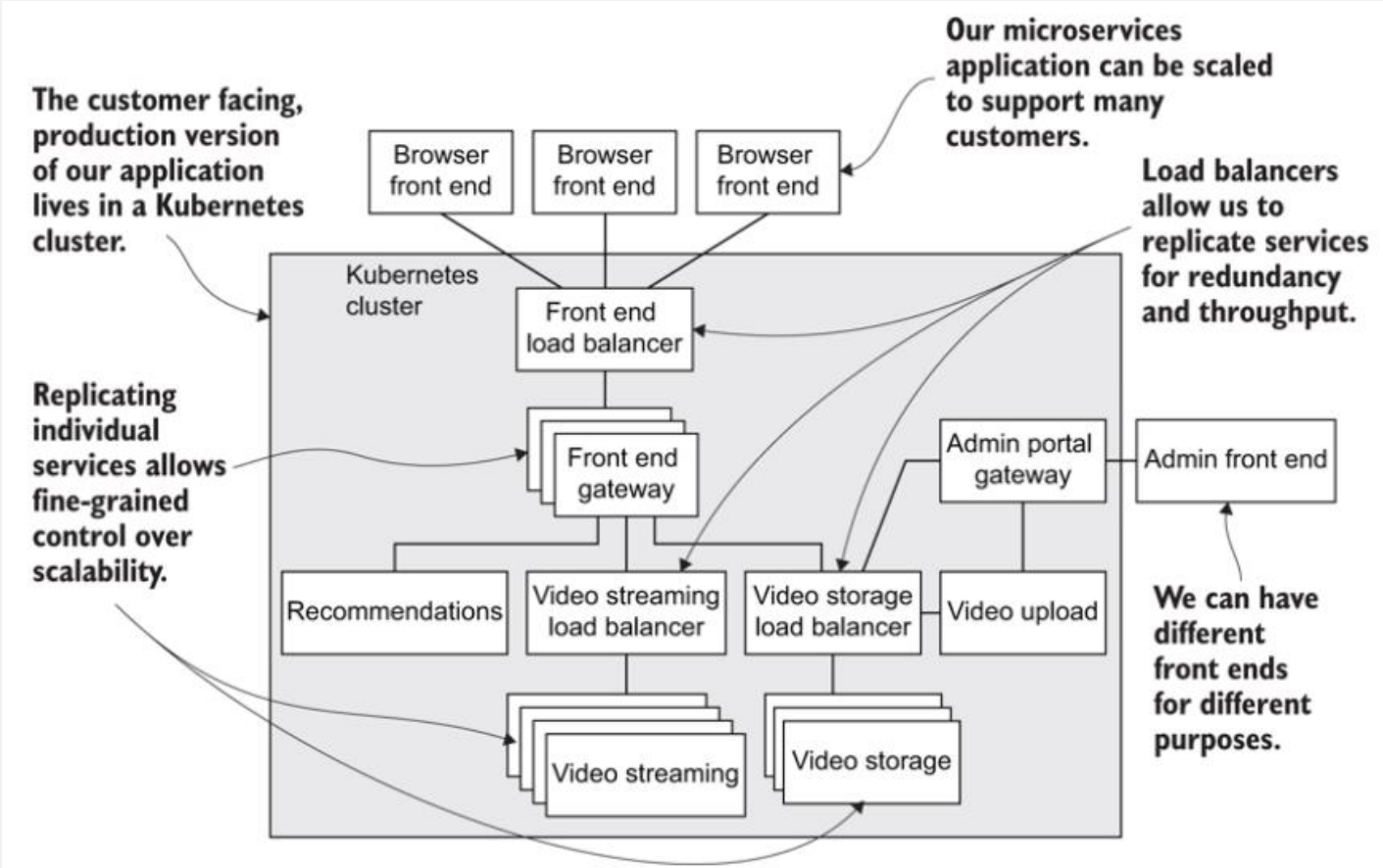
Examples of tech stacks: **MEAN** (MongoDB, Express, Angular, Node.js), **LAMP** (Linux, Apache, MySQL, PHP)… You can choose a different tech stack for each microservice in a microservices application.

Tools that are useful in Microservices

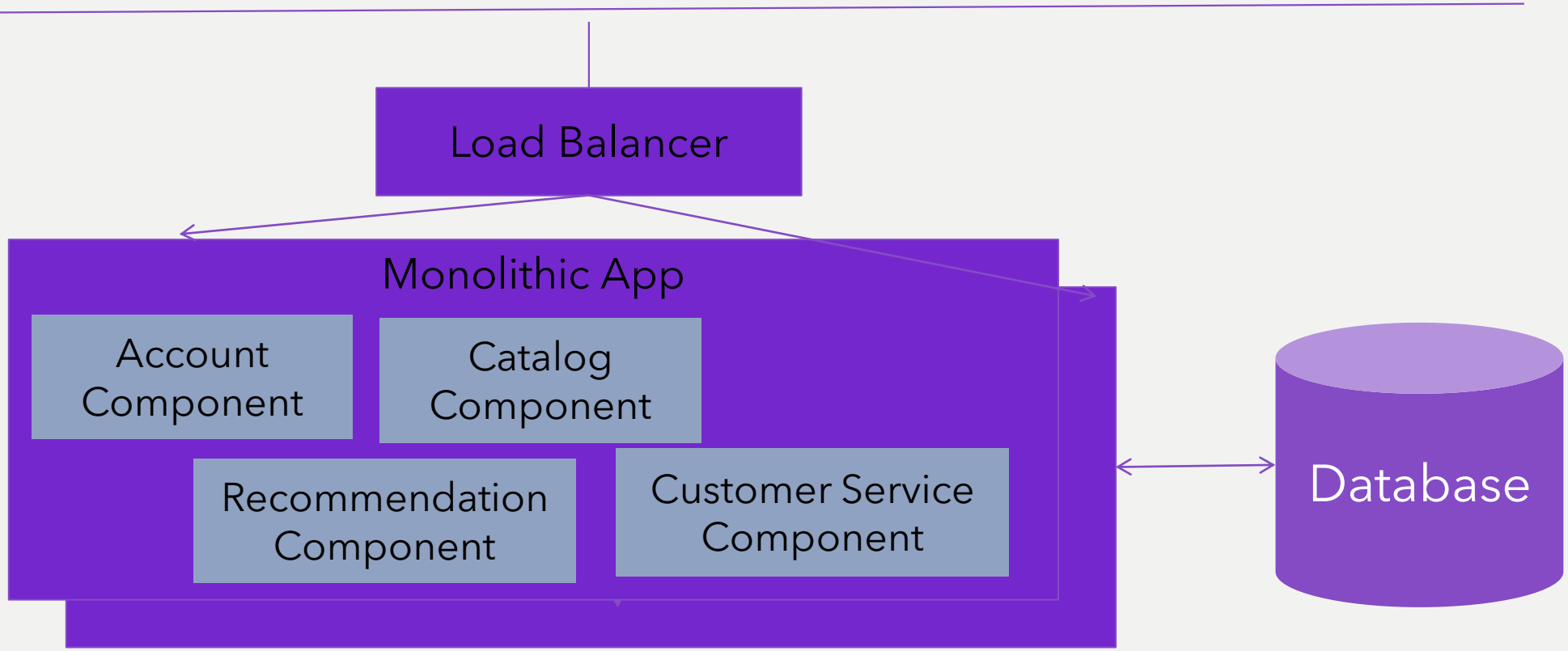
- Docker – To package and deploy microservices
- Docker Compose – To test microservices
- Kubernetes (K8S) – To host application in the cloud
- Terraform – To build our software factory production infrastructure in the cloud using code (***IaC*** approach – a declarative approach).

Packaging for Production & Deployment

- Docker – To package and deploy microservices
- Docker Compose – To test microservices
- Kubernetes (K8S) – To host application in the cloud
- Terraform – To build our software factory production infrastructure in the cloud using code (*IaC* approach – a declarative approach).



Monolithic Architecture Netflix (prior 2010)



Microservices Architecture (2010+) on AWS

Load Balancer

API Gateway

Account Service

Catalog Service

Catalog Data Service

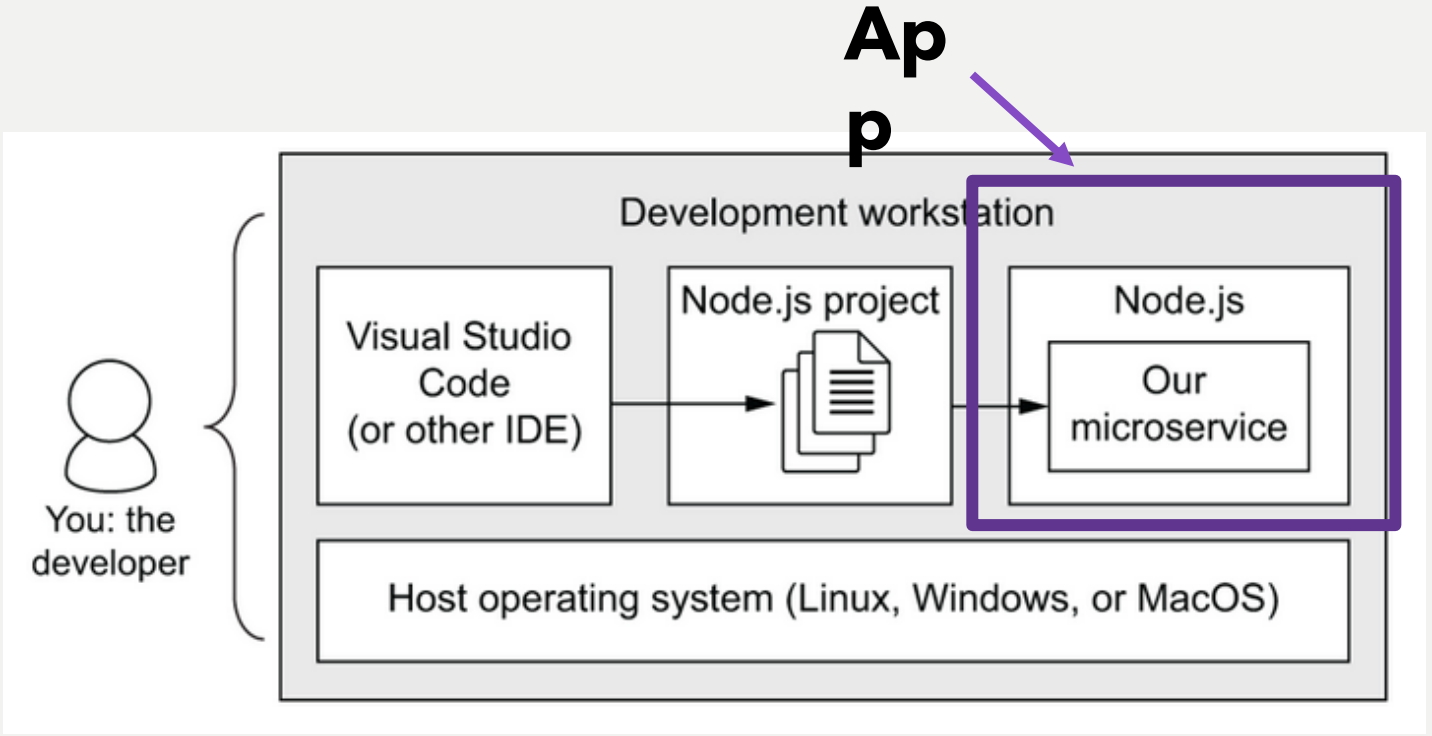
Catalog DB

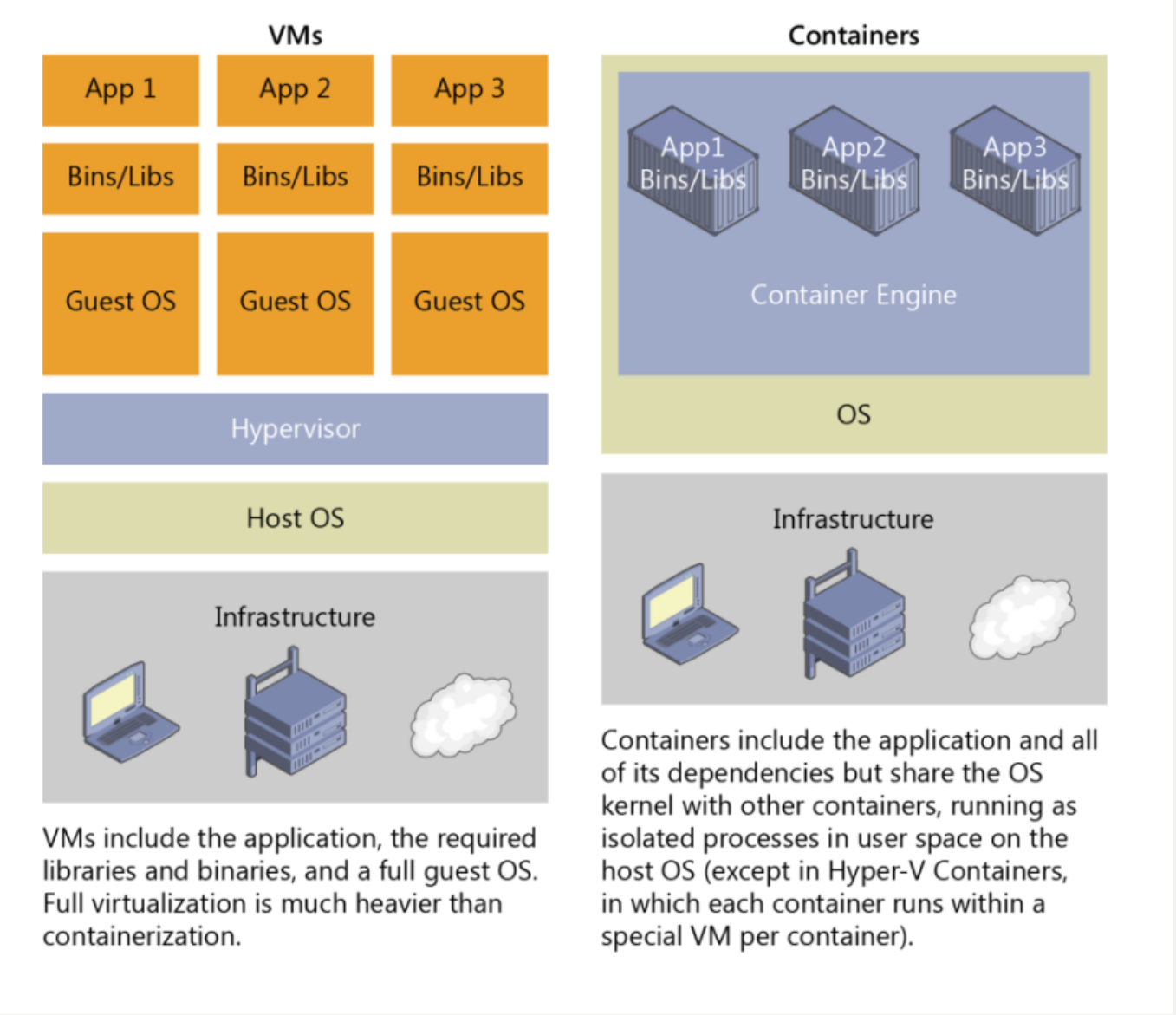
Recommendation Service

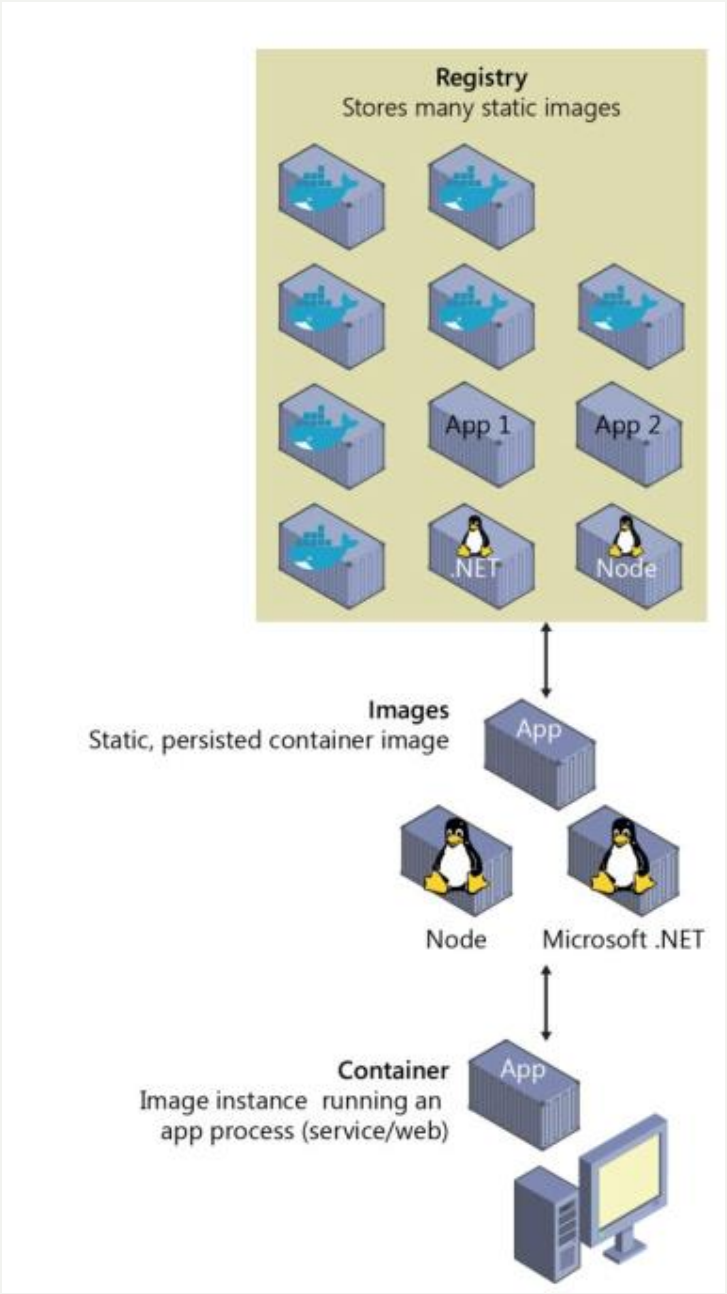
Customer Service Service

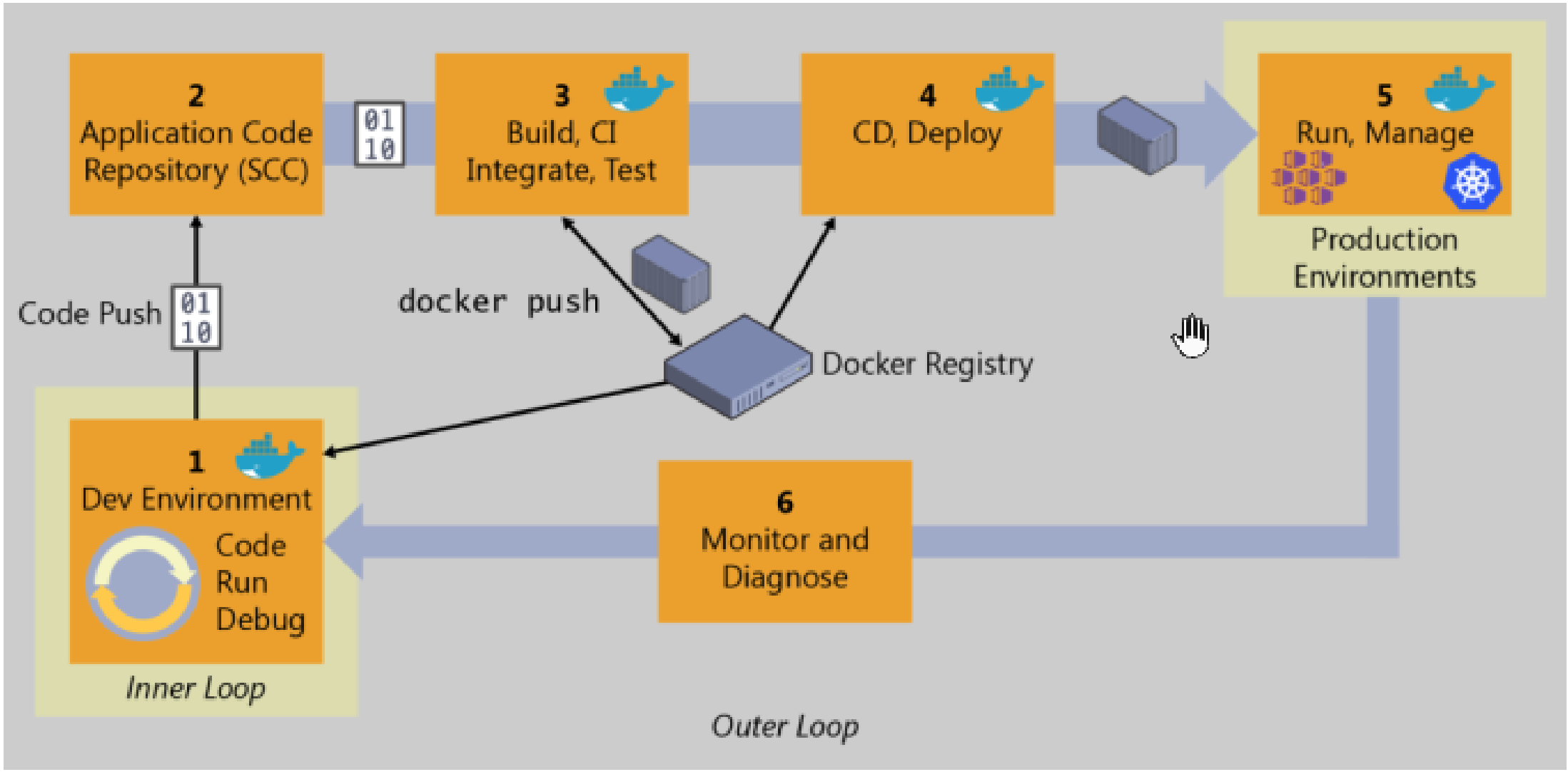
Customer DB

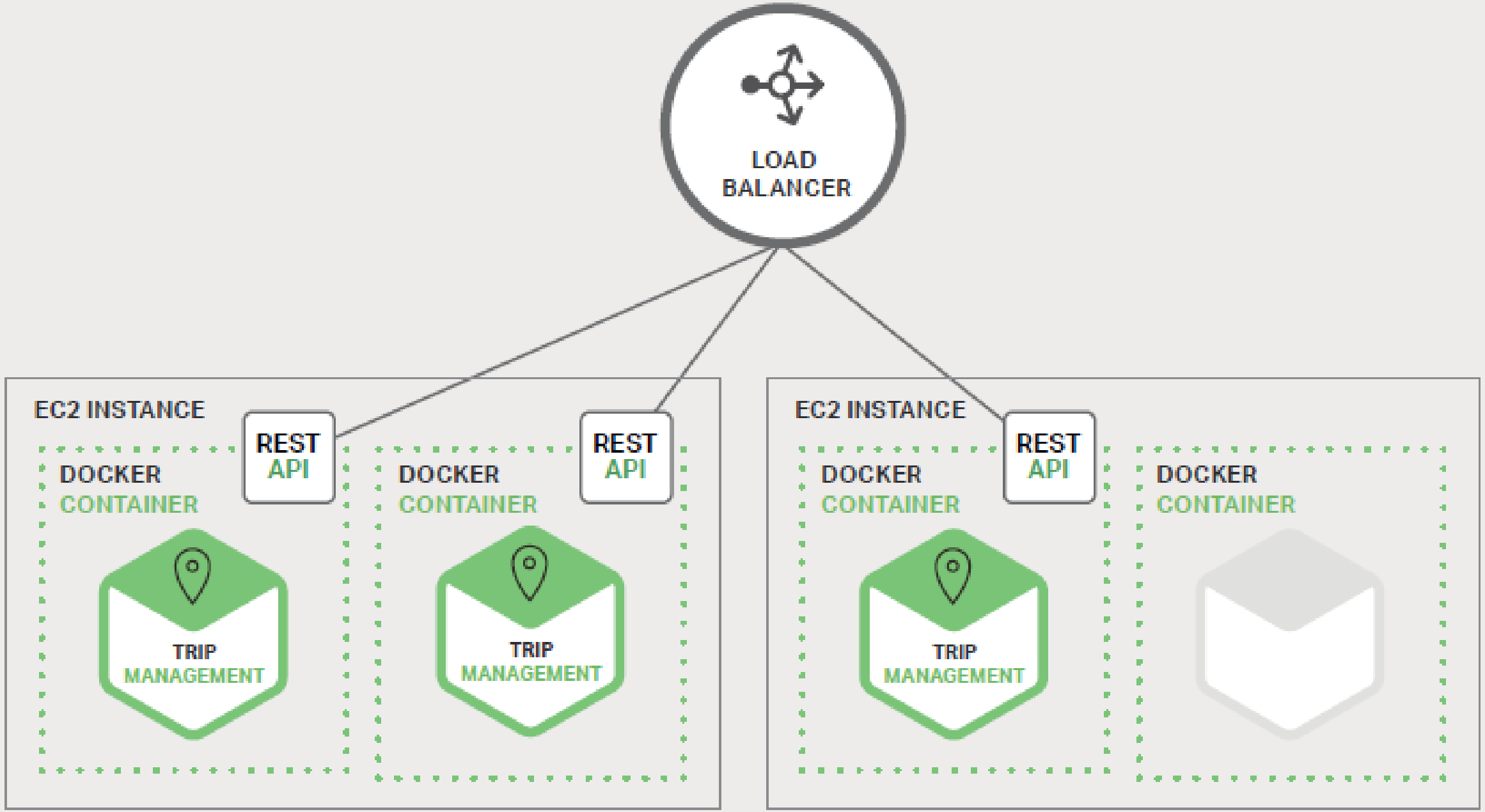
Example of Deploying a Single Service




VMs
App 1
App 2
App 3
Bins/Libs
Bins/Libs
Bins/Libs
Guest OS
Guest OS
Guest OS
Hypervisor
Host OS
Infrastructure
VMs include the application, the required libraries and binaries, and a full guest OS. Full virtualization is much heavier than containerization.
Containers
App1 Bins/Libs
App2 Bins/Libs
App3 Bins/Libs
Container Engine
OS
Infrastructure
Containers include the application and all of its dependencies but share the OS kernel with other containers, running as isolated processes in user space on the host OS (except in Hyper-V Containers, in which each container runs within a special VM per container).

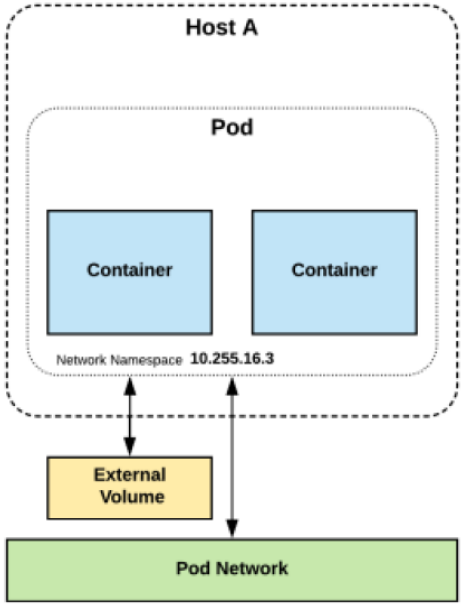

Registry
Stores many static images
App 1
App 2
.NET
Node
Images
Static, persisted container image
App
Node
Microsoft .NET
Container
Image instance running an app process (service/web)
App

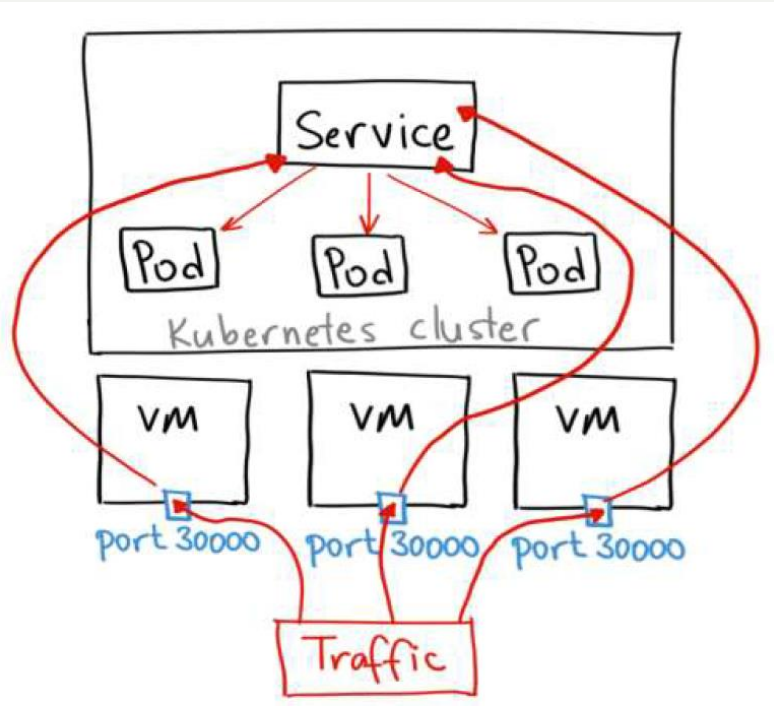

2
Application Code
Repository (SCC)
01
10
3
Build, CI
Integrate, Test
4
CD, Deploy
5
Run, Manage
Production
Environments
Code Push
01
10
docker push
Docker Registry
1
Dev Environment
Code
Run
Debug
Inner Loop
6
Monitor and
Diagnose
Outer Loop


LOAD
BALANCER
EC2 INSTANCE
REST
API
DOCKER
CONTAINER
TRIP
MANAGEMENT
REST
API
DOCKER
CONTAINER
TRIP
MANAGEMENT
EC2 INSTANCE
REST
API
DOCKER
CONTAINER
TRIP
MANAGEMENT
DOCKER
CONTAINER

What are Pods?

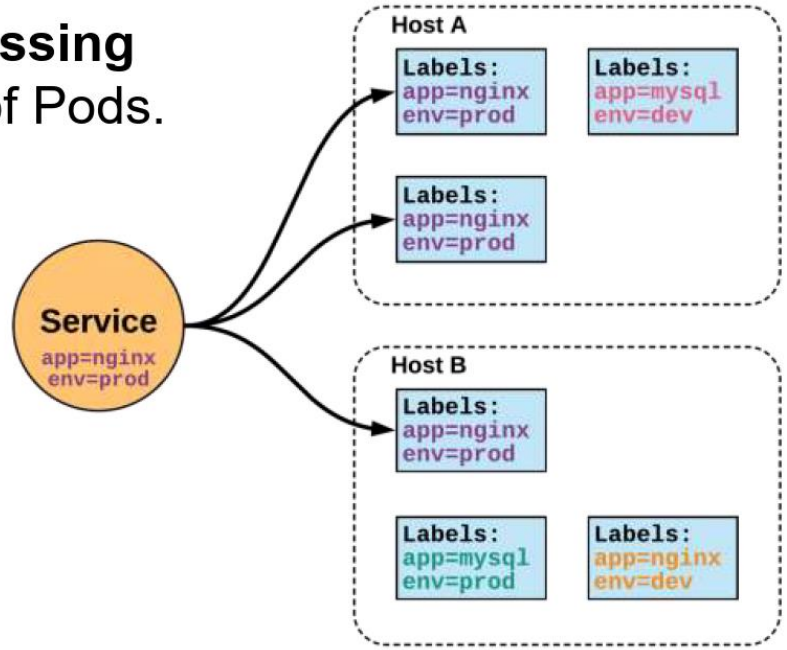
- **Atomic unit** or smallest "*unit of work*"of Kubernetes.
- Pods are **one or MORE containers** that share volumes and namespace.
- **They are also ephemeral!**

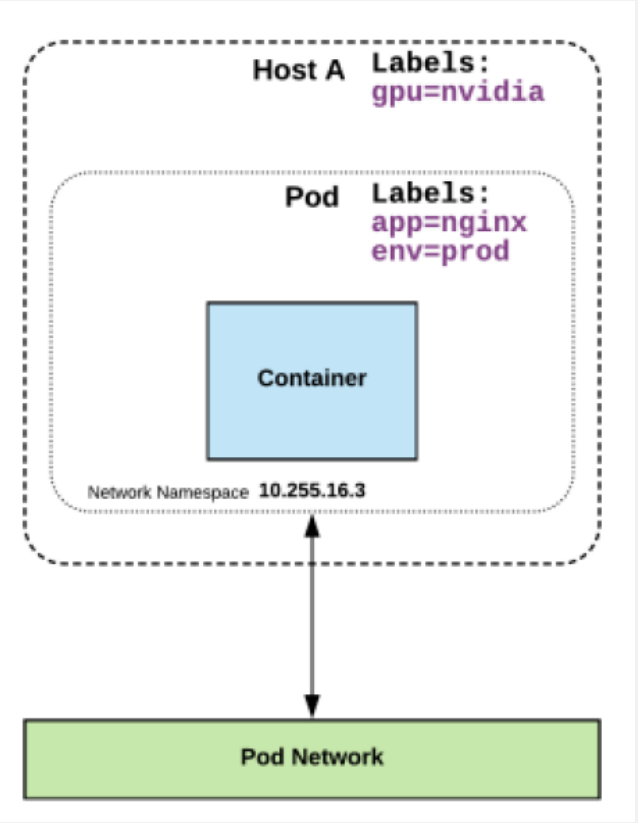




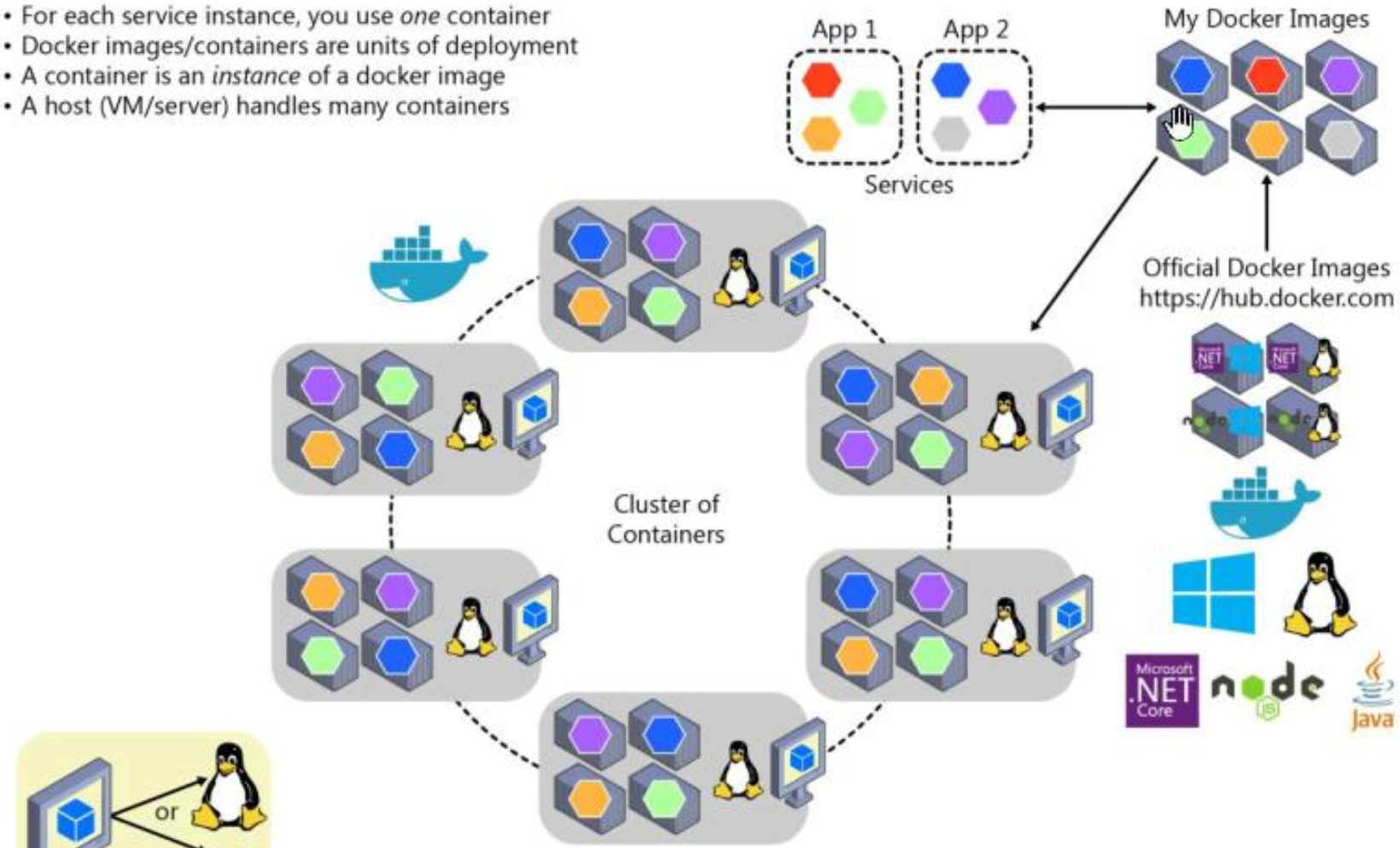
Services

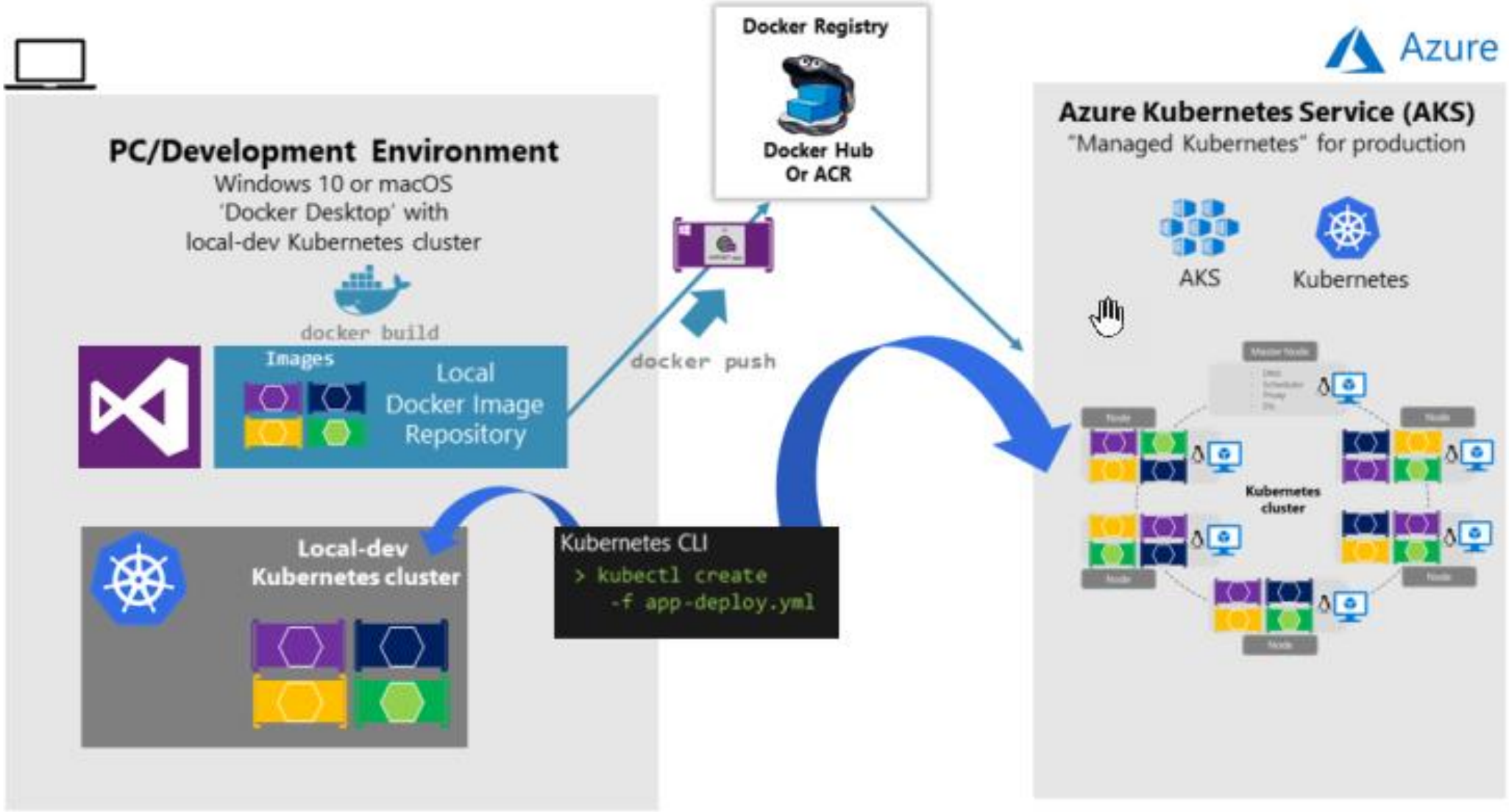
- **Unified method of accessing** the exposed workloads of Pods.
- **Durable resource**
  - static cluster IP
  - static namespaced DNS name



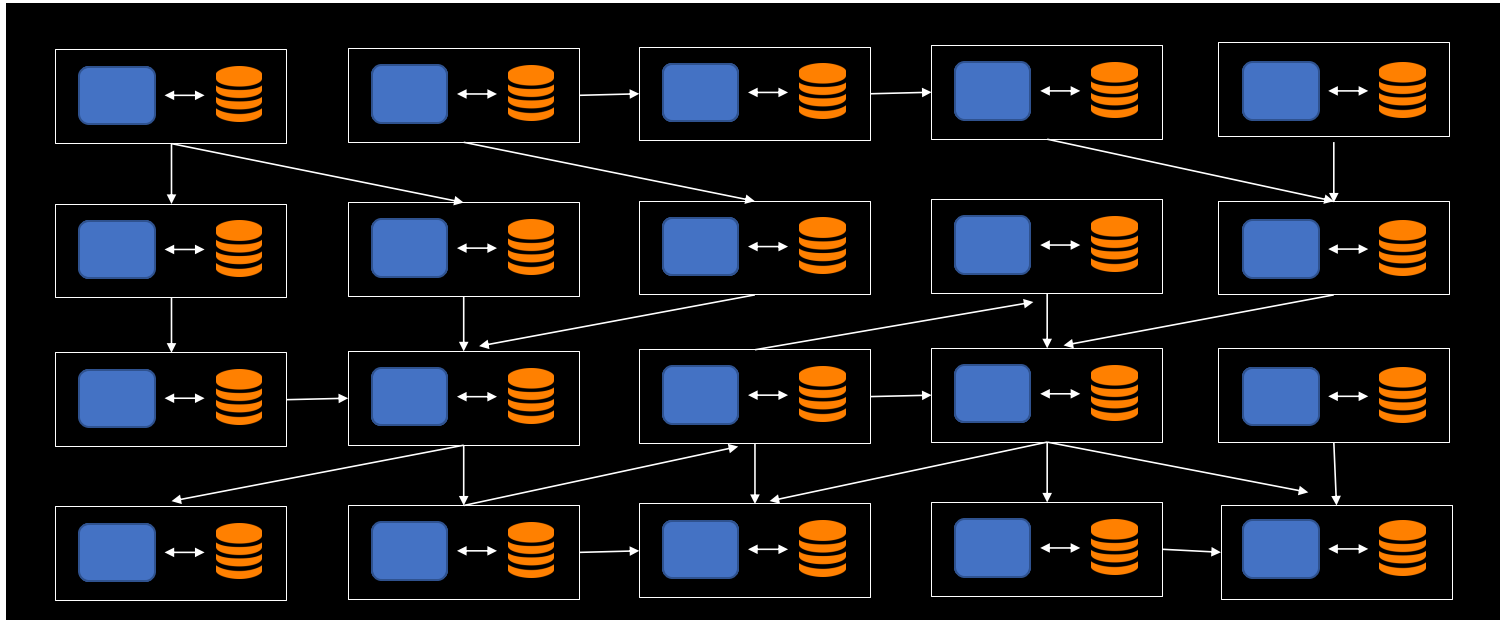


- For each service instance, you use *one* container
- Docker images/containers are units of deployment
- A container is an *instance* of a docker image
- A host (VM/server) handles many containers

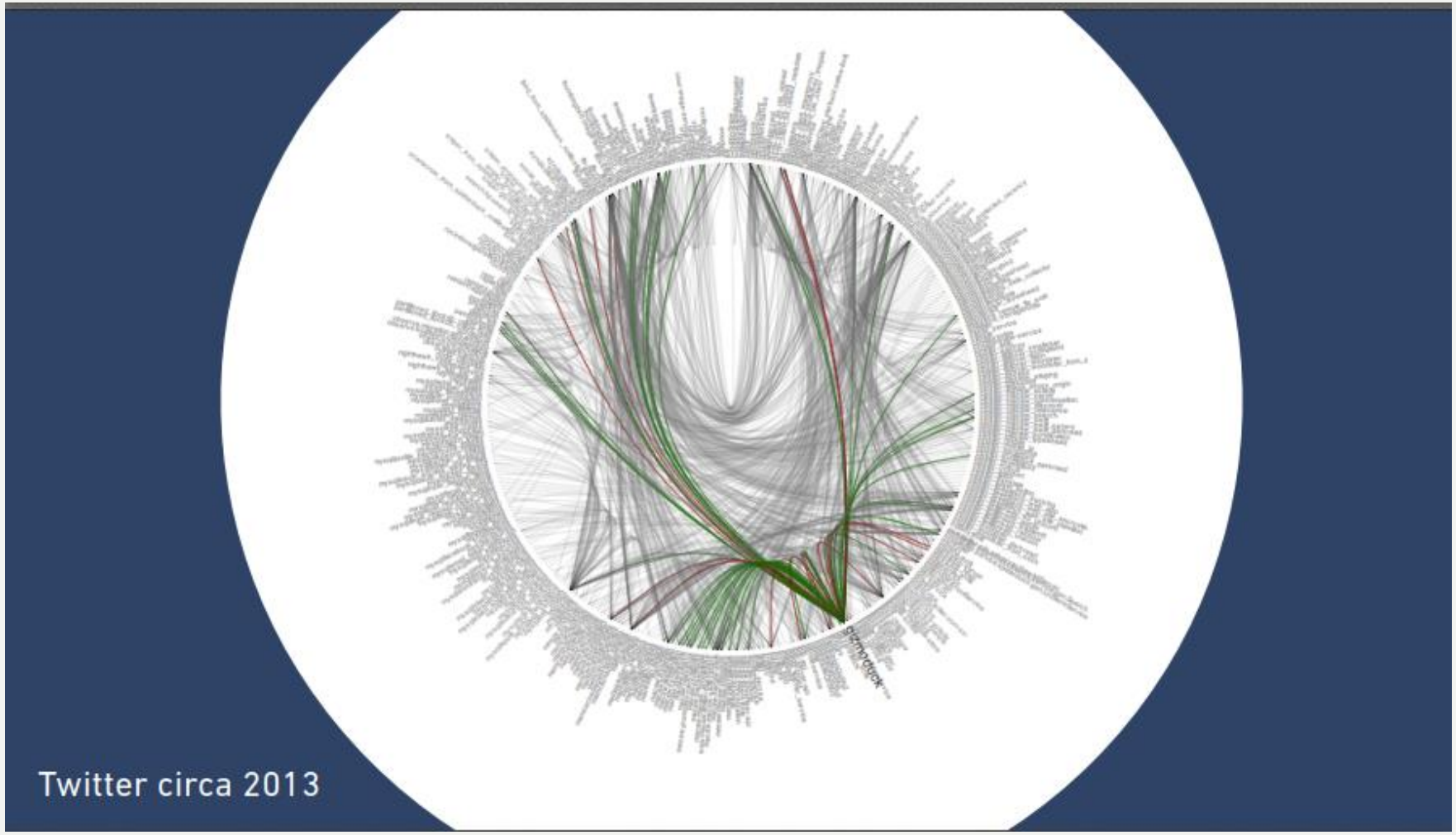



Docker Registry
Docker Hub
Or ACR
Azure
PC/Development Environment
Windows 10 or macOS
'Docker Desktop' with
local-dev Kubernetes cluster
docker build
Images
Local
Docker Image
Repository
docker push
Azure Kubernetes Service (AKS)
"Managed Kubernetes" for production
AKS
Kubernetes
Kubernetes
cluster
Local-dev
Kubernetes cluster
Kubernetes CLI
> kubectl create
-f app-deploy.yml

Microservices can quickly multiply in complexity

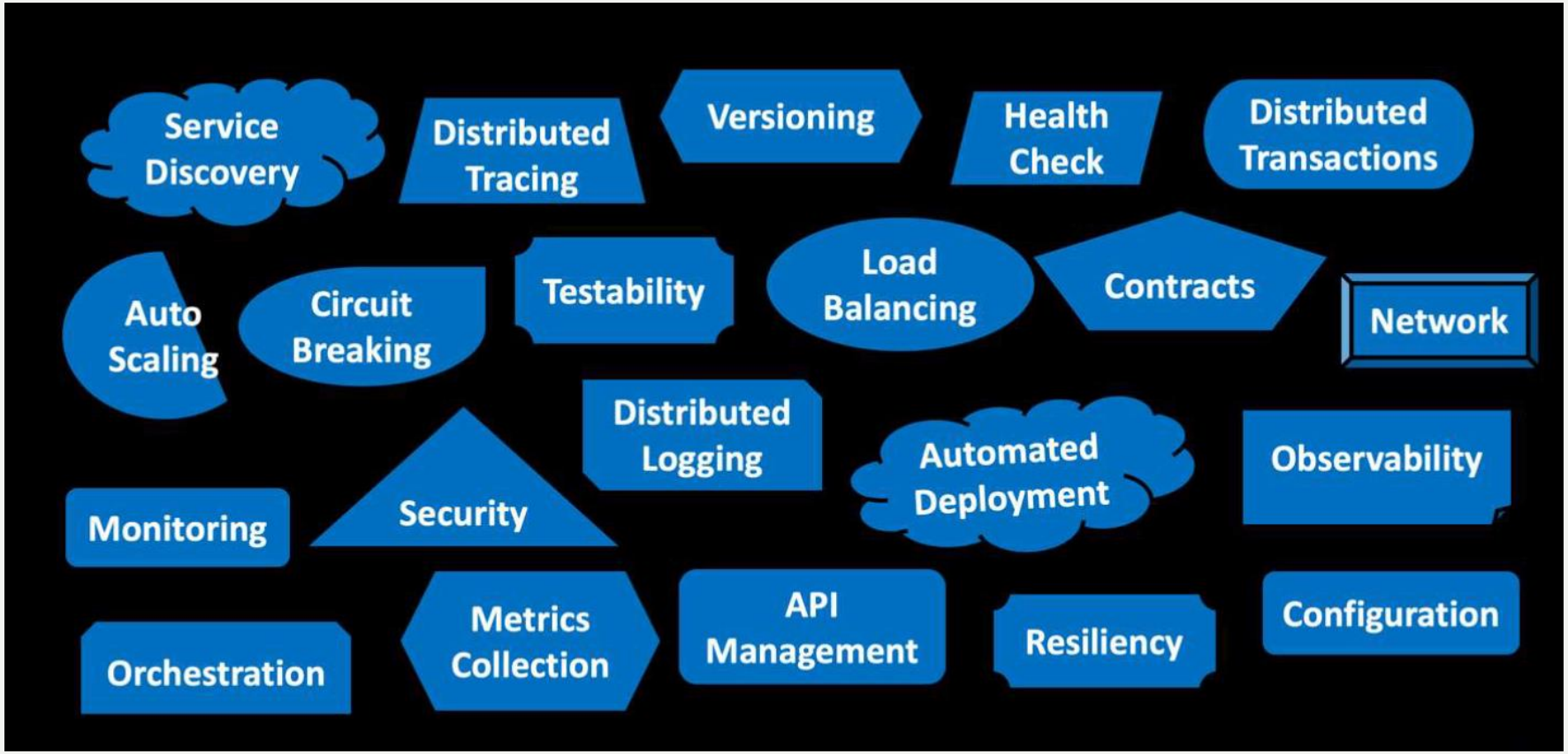


Microservices can lead to lot of complexity – 100k+ microservices in a large enterprise system



Open Problems to Solve in Microservices Architecture & Containers

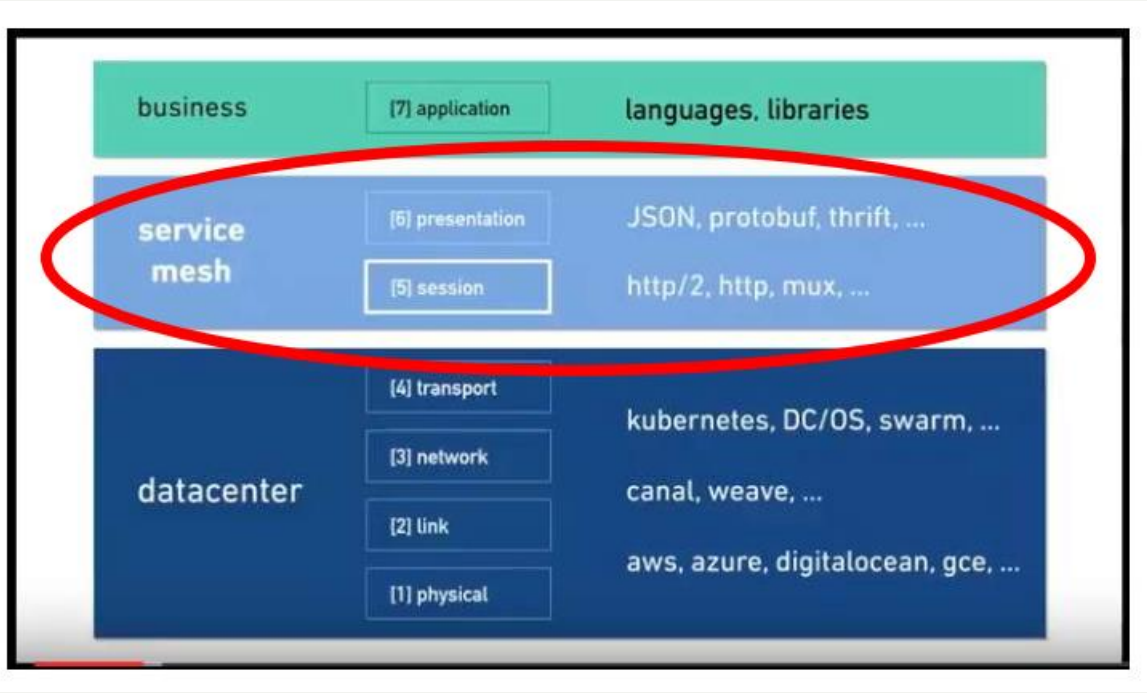
(Amazon, Google, Microsoft are actively working on this)



Service Mesh solved a lot of home-grown solutions

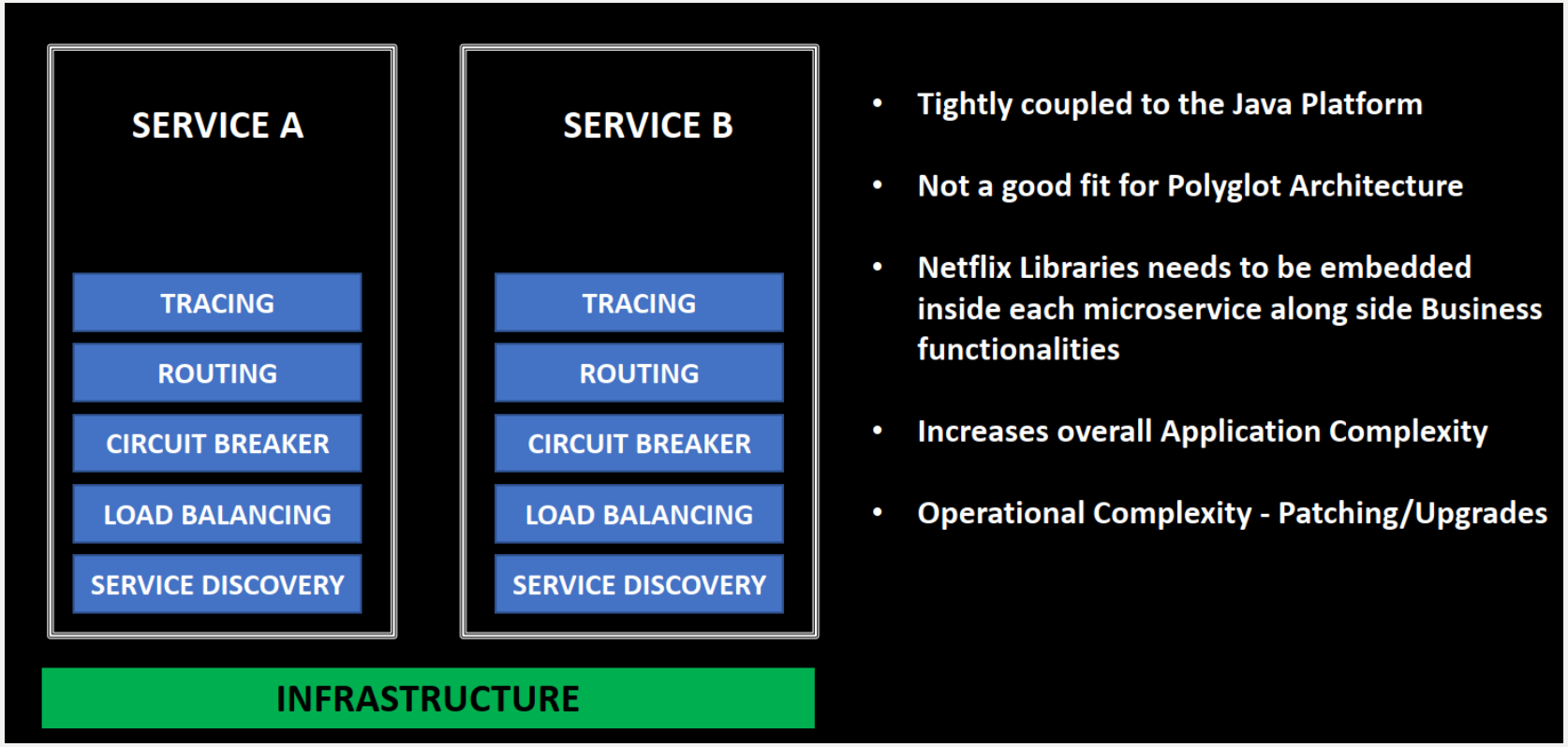
- Netflix
  - Karyon + HTTP/JSON or RxNetty RPC + Eureka + Hystrix + …
- Twitter
  - Finagle + Thrift + ZooKeeper + Zipkin
- AirBnB
  - HTTP/JSON + SmartStack + ZooKeeper + Charon/Dyno
- Google
  - Stubby + GSLB + GFE + Dapper

Before Service Mesh

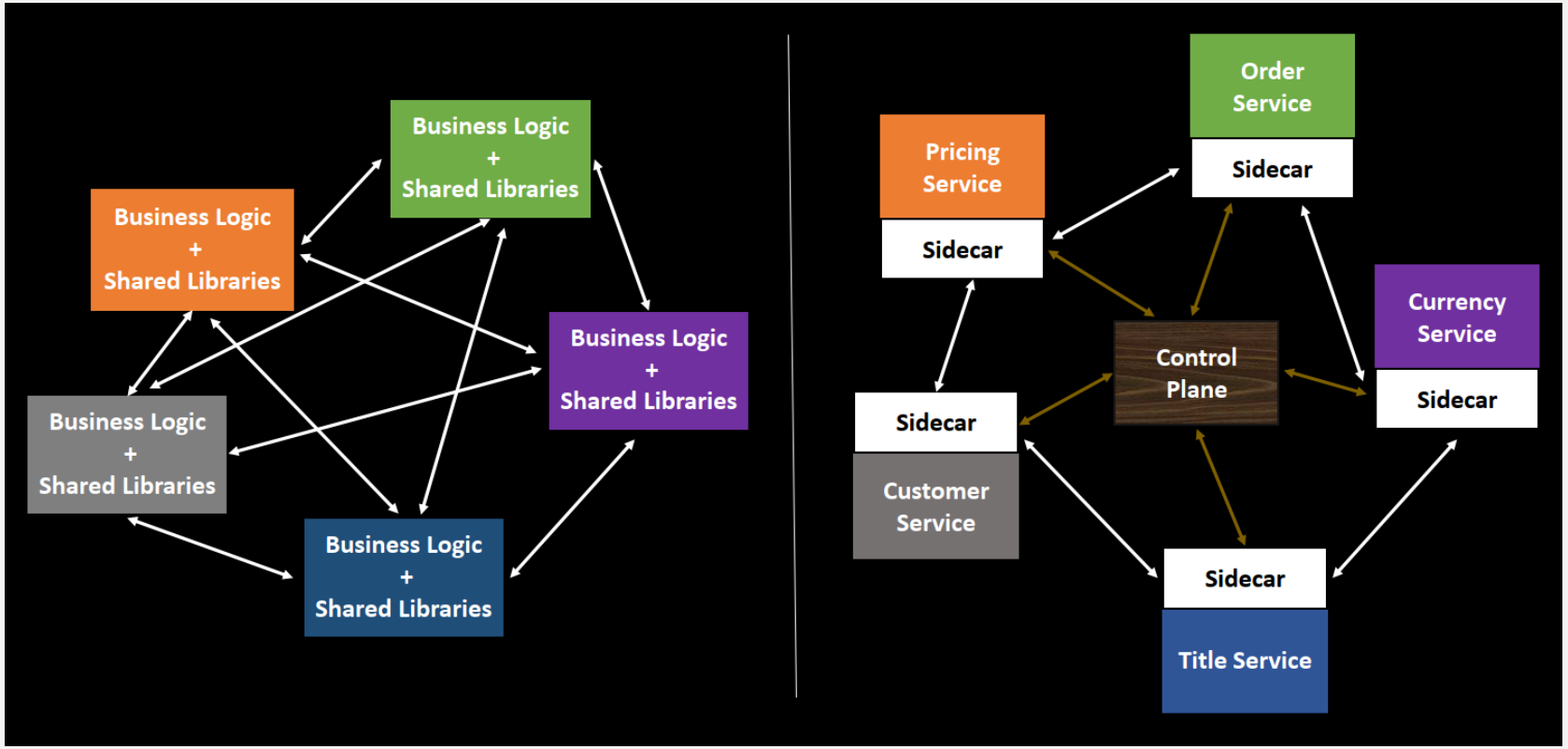


After Service Mesh

Why didn't "home-grown" Netflix approach work in general?



Service Mesh



Service Mesh - Benefits

- Normalises naming and adds logical routing
  - user-service -> AWS-us-east-1a/prod/users/v4
- Adds traffic shaping and traffic shifting
  - Load balancing
  - Deploy control
  - Per-request routing (shadowing, fault injection, debug)
- Adds baseline reliability
  - Health checks, timeouts/deadlines, circuit breaking, and retry (budgets)

- Increased security
  - Transparent mutual TLS
  - Policies (service Access Control Lists - ACL)
- Observability
  - Top-line metrics like request volume, success rates and latencies
  - Distributed tracing
- Sane defaults (to protect the system)
  - Options to tune

What is a Service Mesh – Generic Definition

- A Service Mesh is the connective tissue between your *microservices* that adds the capabilities of traffic control, service discovery, load balancing, resilience, observability, security, ……

Origins of ISTIO

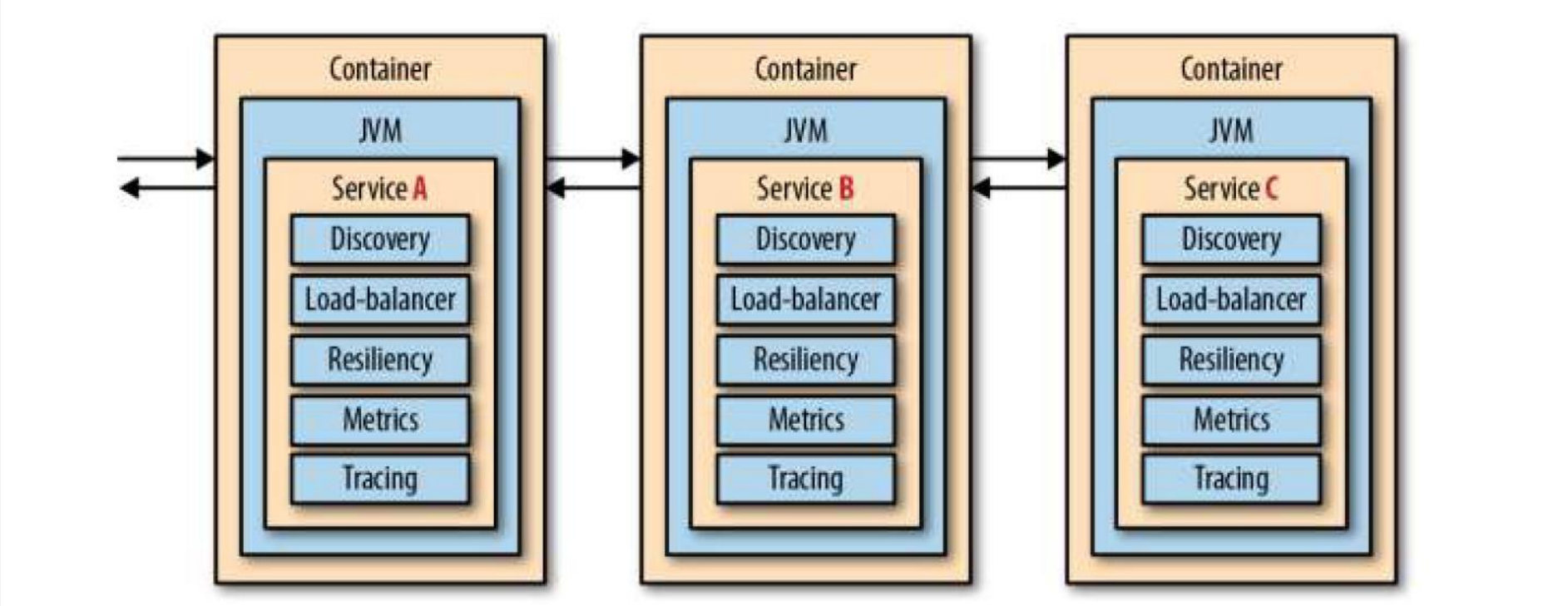
Collaboration between Google IBM and Lyft

Open-source service mesh for microservices

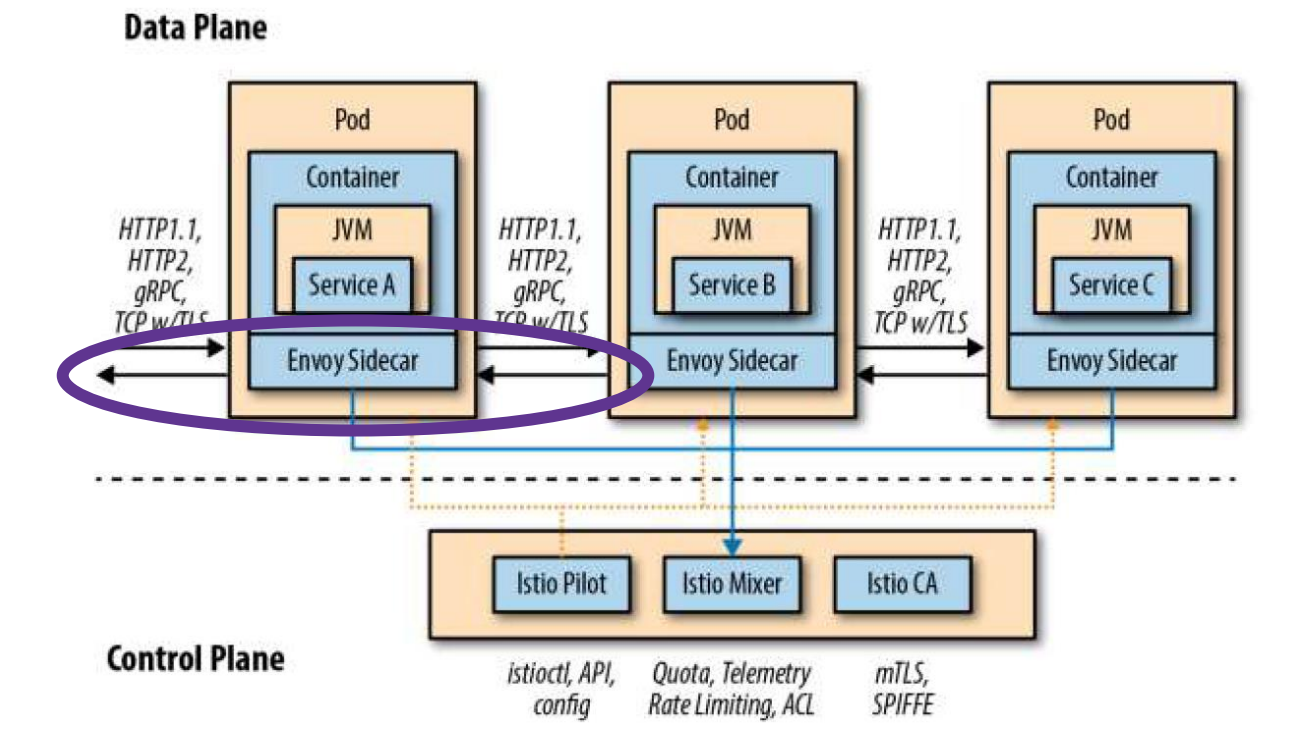
How did Service Mesh Help Solve these Open Problems?
Real World v. Practical World

Fallacy	Solutions
The network is reliable	Automatic Retries, Message Queues
Latency is zero	Caching Strategy, Bulk Requests, Deploy in AZs near client
Bandwidth is infinite	Throttling Policy, Small payloads with Microservices
The network is secure	Network Firewalls, Encryption, Certificates, Authentication
Topology does not change	No hardcoding IP, Service Discovery Tools
There is one administrator	DevOps Culture eliminates Bus Factor
Transport cost is zero	Standardized protocols like JSON, Cost Calculation
The network is homogenous	Circuit Breaker, Retry and Timeout Design Pattern

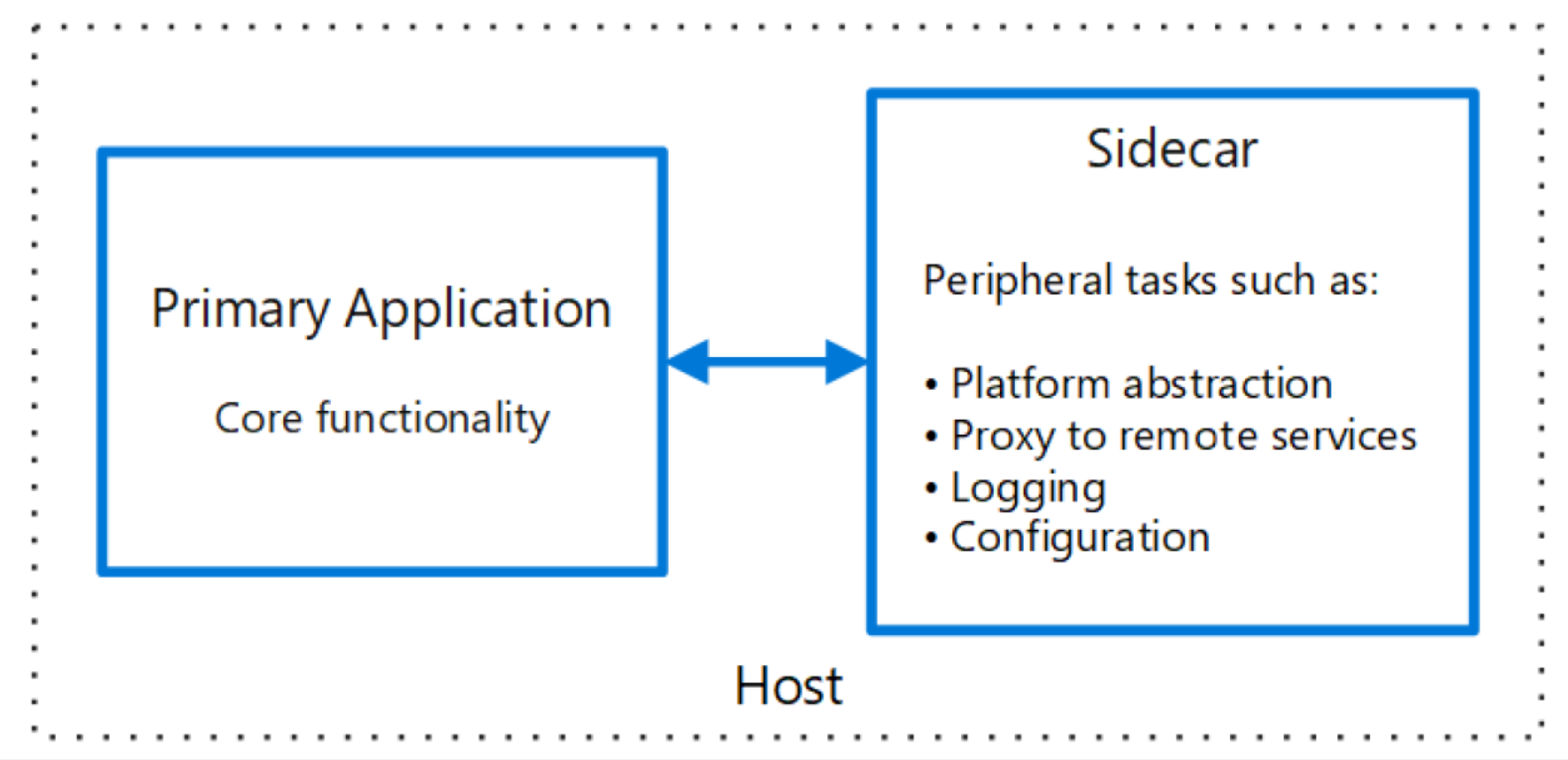
Microservices before ISTIO



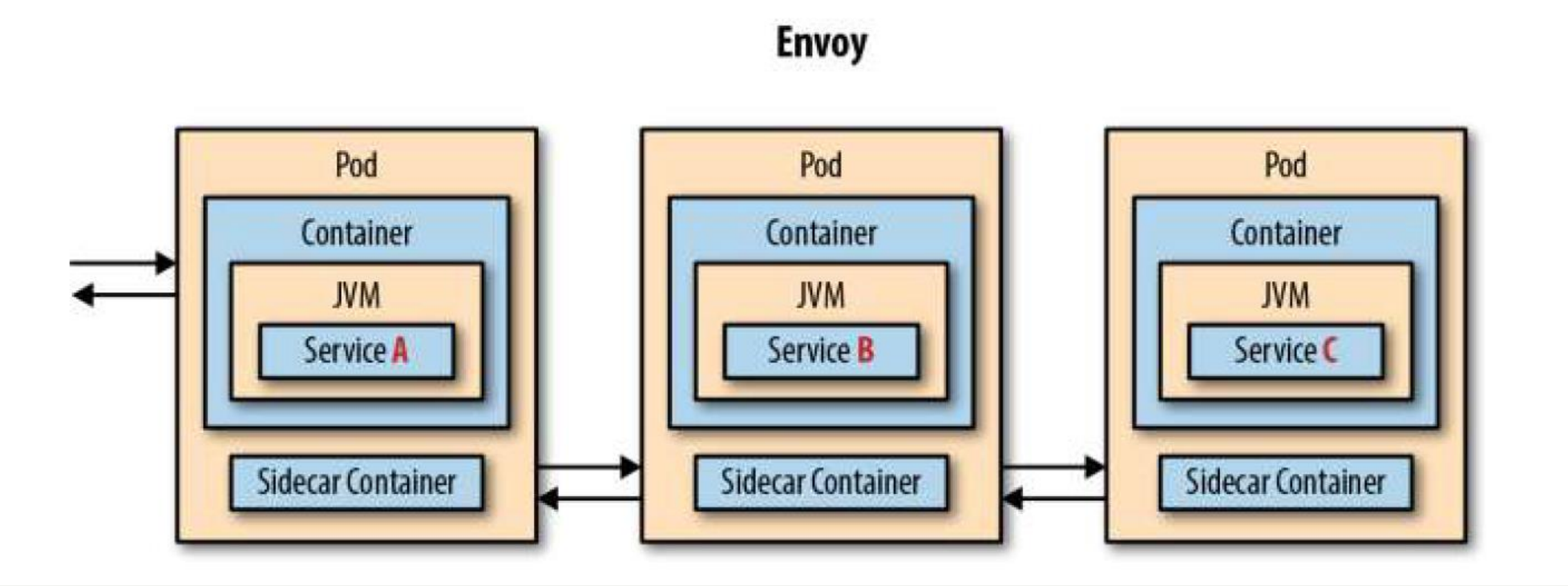
Microservices after ISTIO



What is the Sidecar Pattern?



Envoy is a Sidecar

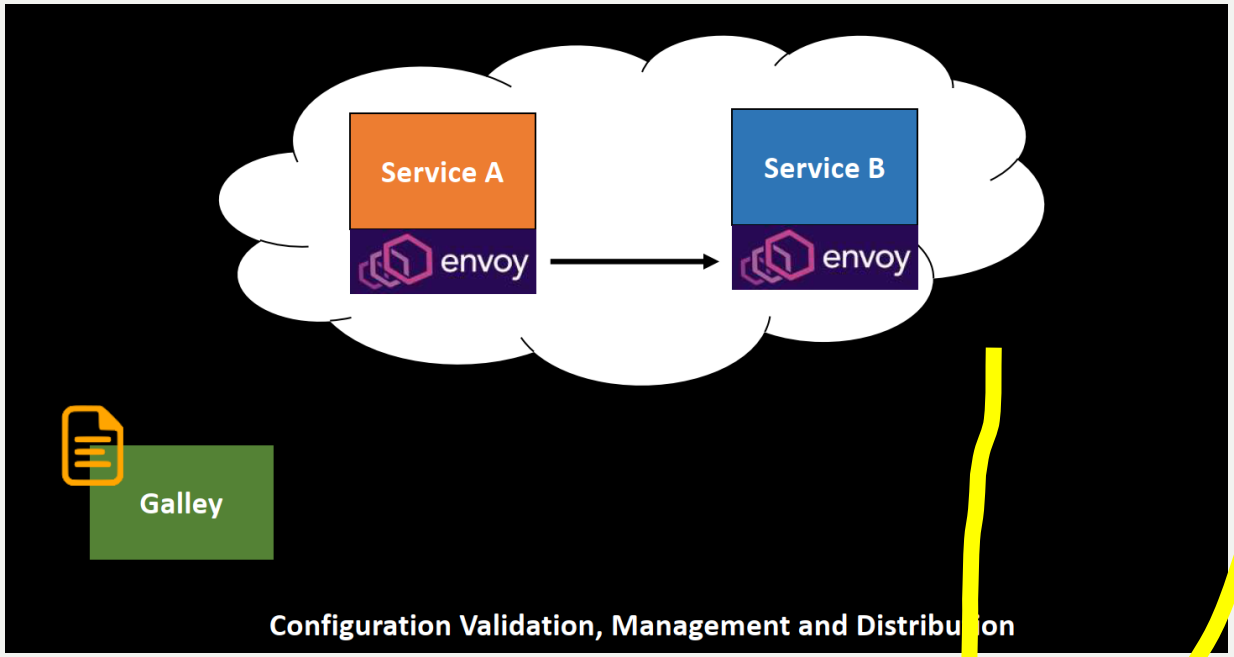


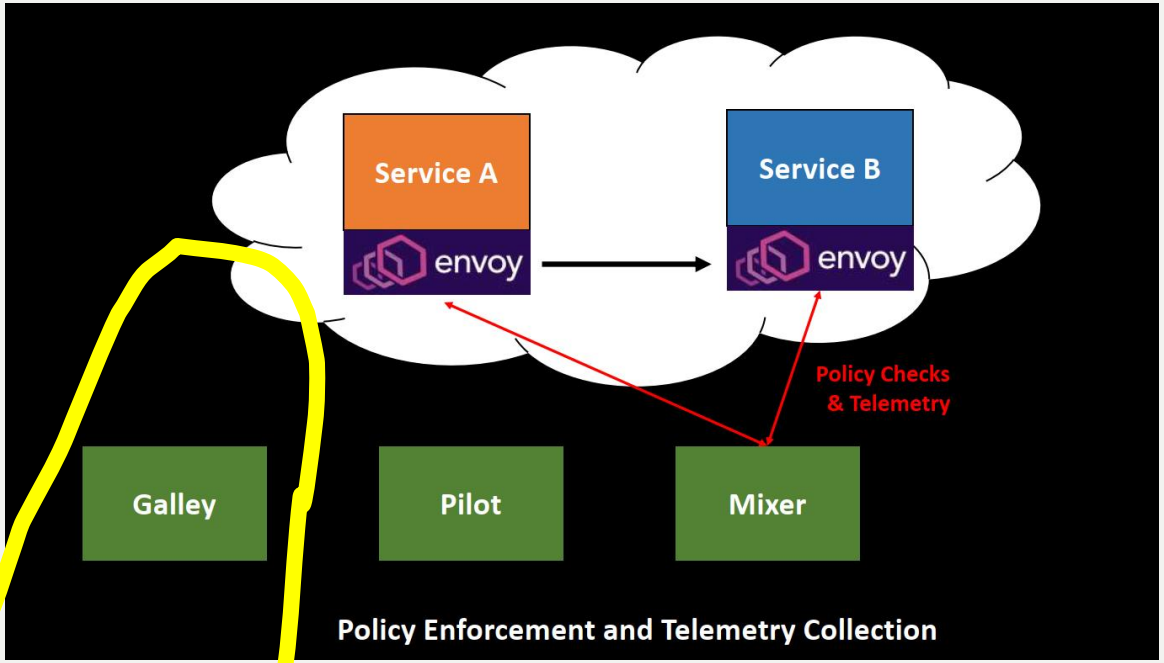
What are the three primary Istio services?

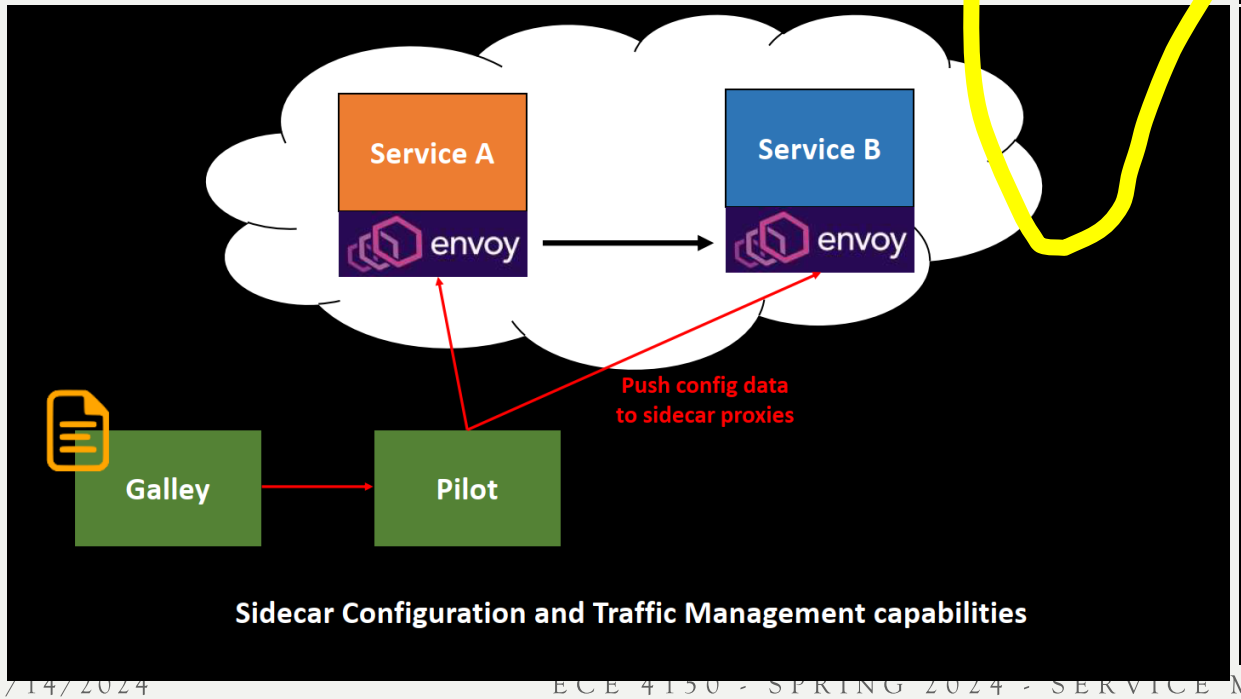
Pilot

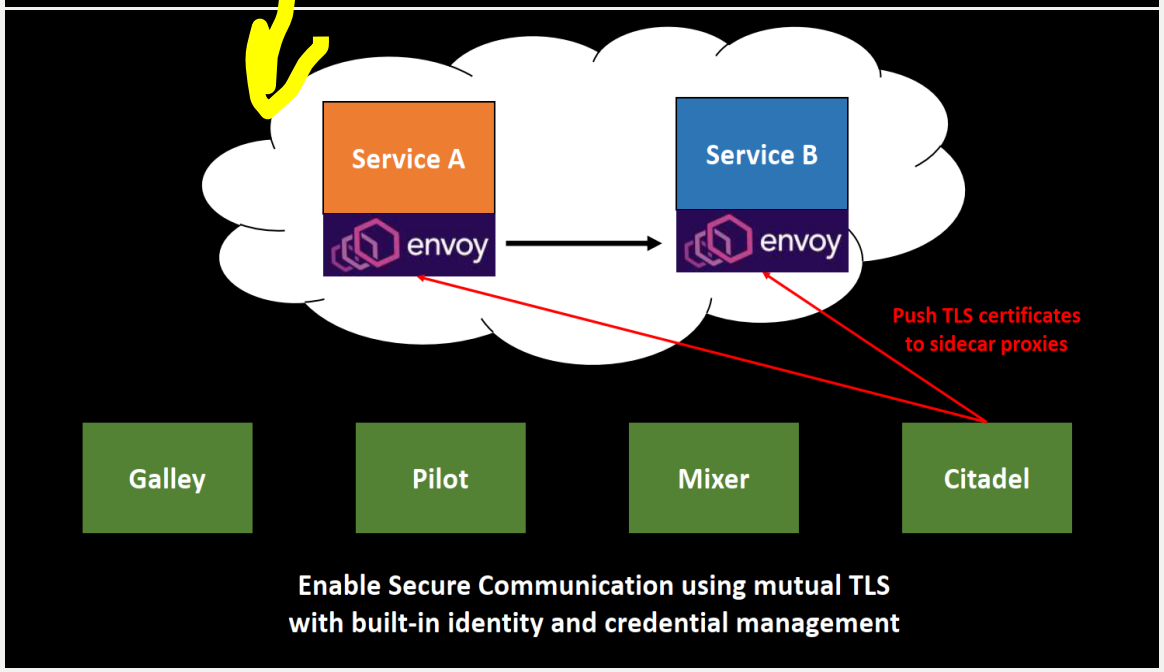
Mixer

Citadel

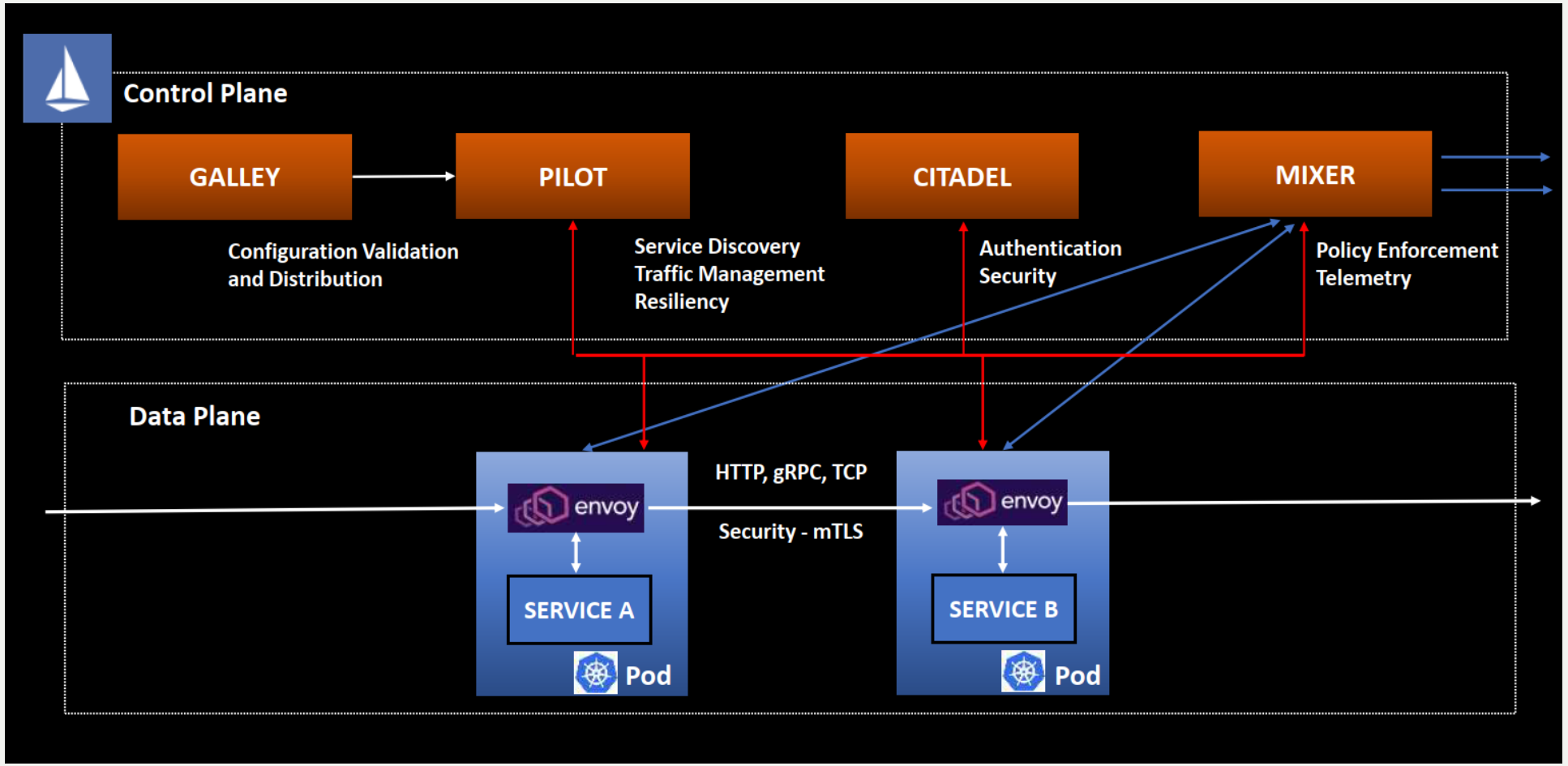

Service A
envoy
Service B
envoy
Galley
Configuration Validation, Management and Distribution


Service A
envoy
Service B
envoy
Policy Checks
& Telemetry
Galley
Pilot
Mixer
Policy Enforcement and Telemetry Collection

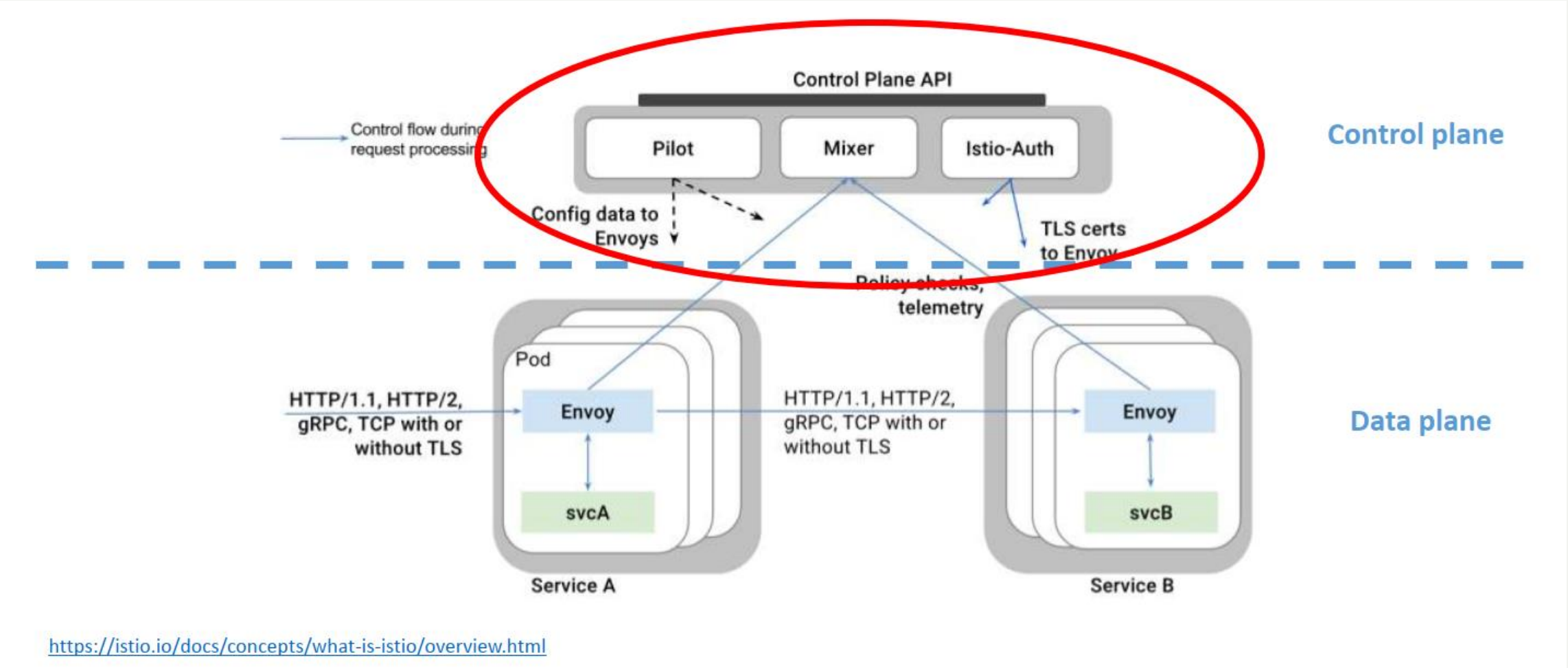

Service A
envoy
Service B
envoy
Push config data
to sidecar proxies
Galley
Pilot
Sidecar Configuration and Traffic Management capabilities


Service A
envoy
Service B
envoy
Push TLS certificates
to sidecar proxies
Galley
Pilot
Mixer
Citadel
Enable Secure Communication using mutual TLS
with built-in identity and credential management

Istio Components



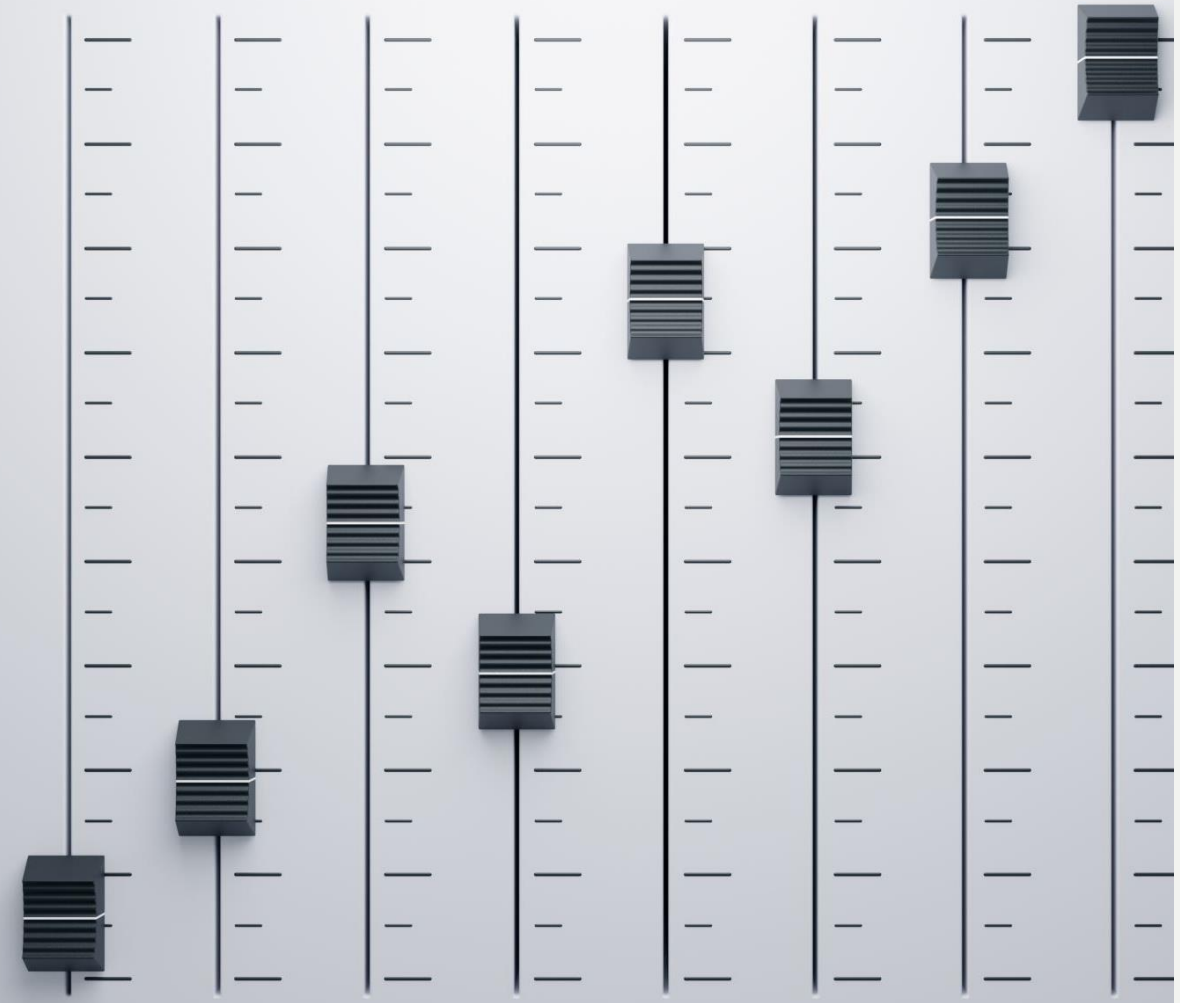
Control – How they fit the control plane



https://istio.io/docs/concepts/what-is-istio/overview.html

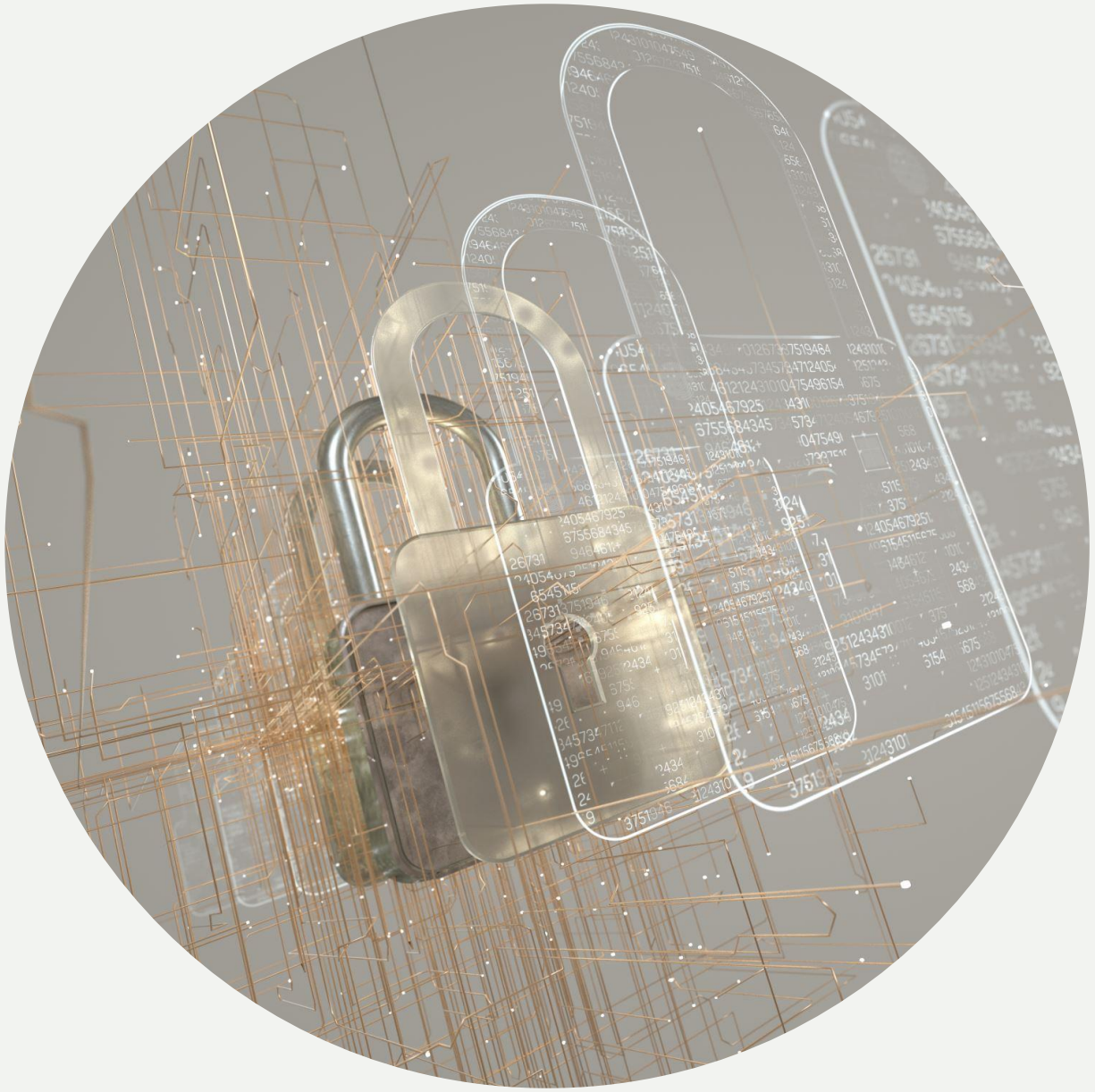
Pilot

- Pilot provides
  - Service Discovery
  - Fine-grained request distribution, retries, timeouts.



Mixer

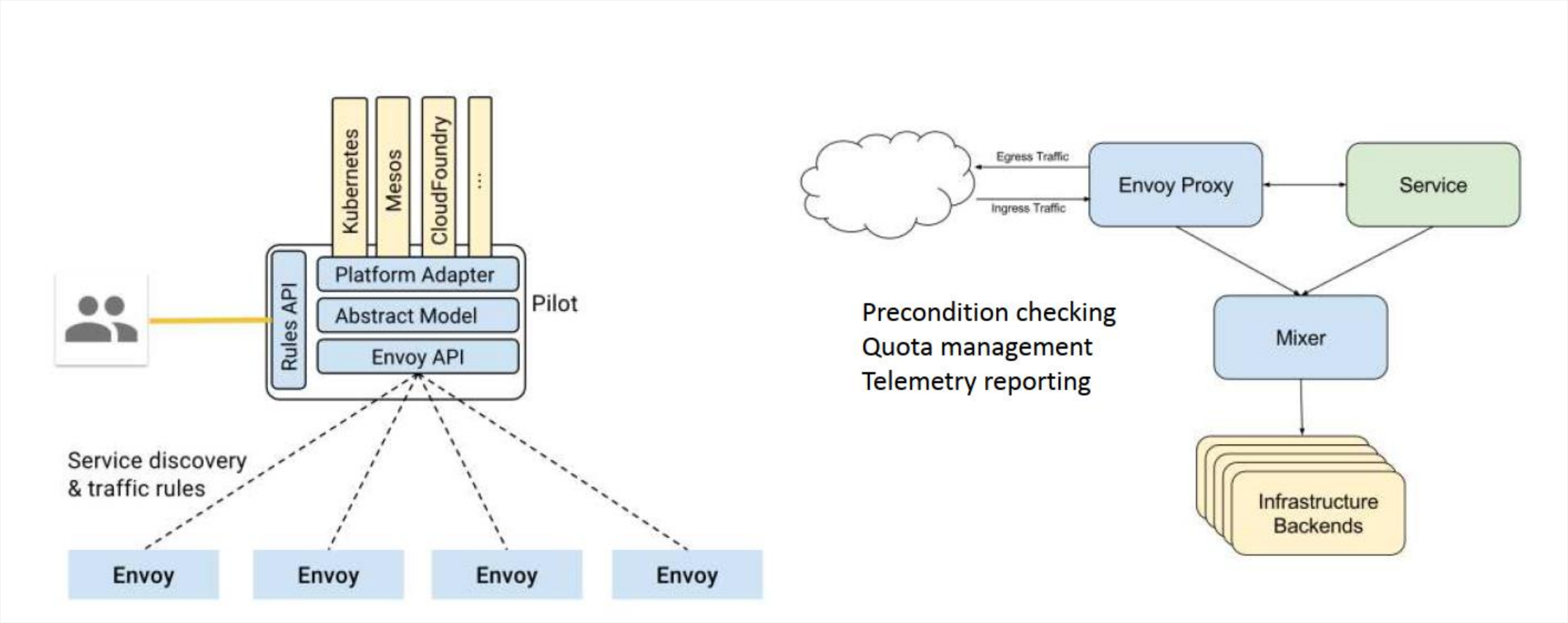
- Mixer allows you to:
  - Create policies and rules
  - Rate-Limiting rules
  - Custom metrics
  - Pluggable architecture



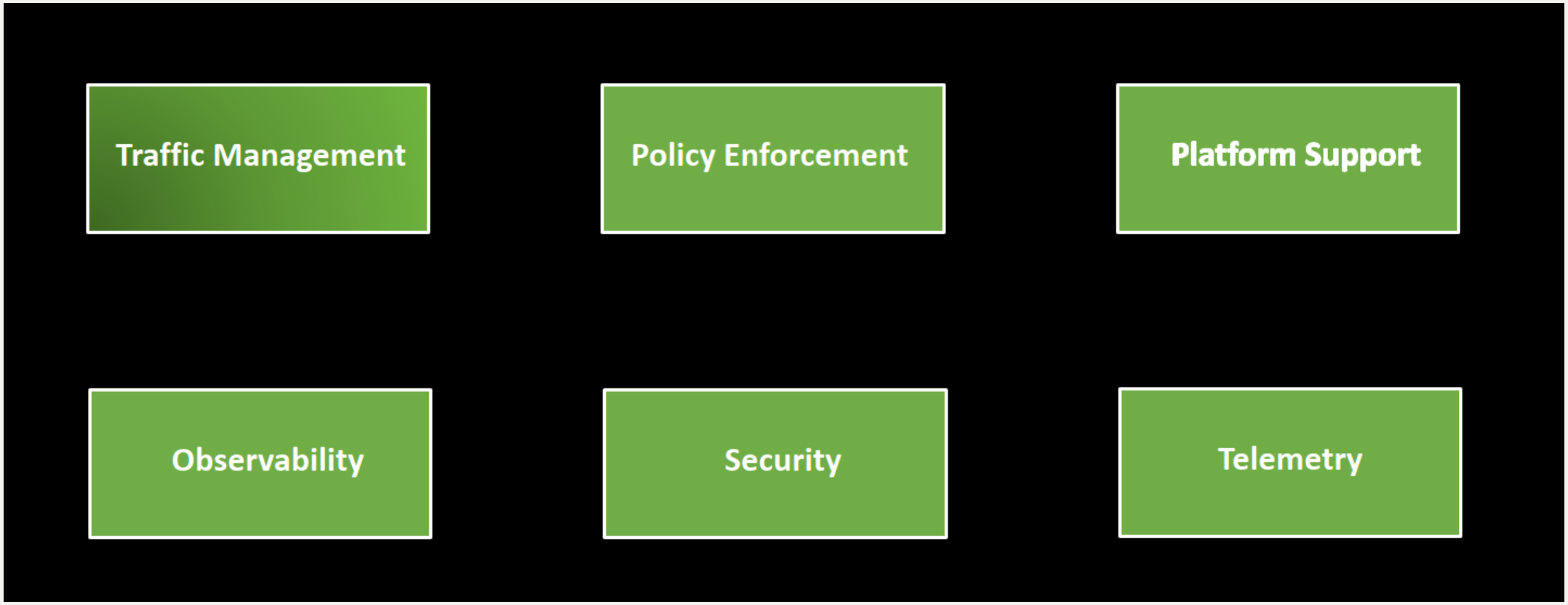
Citadel

- Responsible for secure operations
  - Certificate Signing
  - Certificate Issuance
  - mTLS security between services through transparent encryption

Pilot, Mixer and Citadel



What features does Service Mesh provide?



Traffic Control with Istio – Canary and Dark Launches

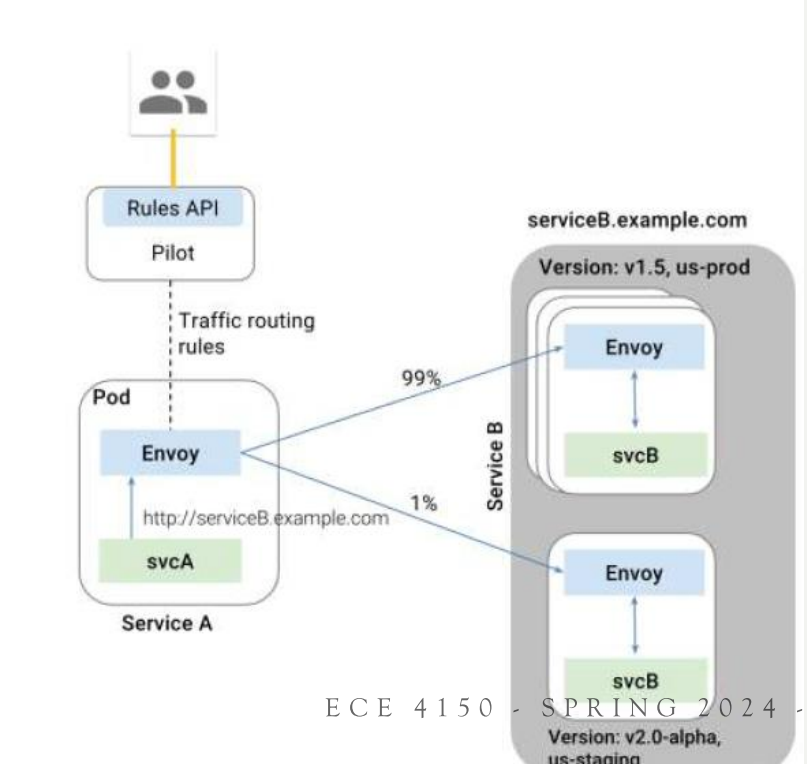
```yaml
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: recommendation
  namespace: tutorial
spec:
  host: recommendation
  subsets:
  - labels:
      version: v1
    name: version-v1
  - labels:
      version: v2
    name: version-v2
```

```yaml
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: recommendation
  namespace: tutorial
spec:
  hosts:
  - recommendation
  http:
  - route:
    - destination:
        host: recommendation
        subset: version-v1
      weight: 100
```

Original Launch

Traffic Control: Canary Release for API v2

- Allows for phased releases with smaller amounts of load



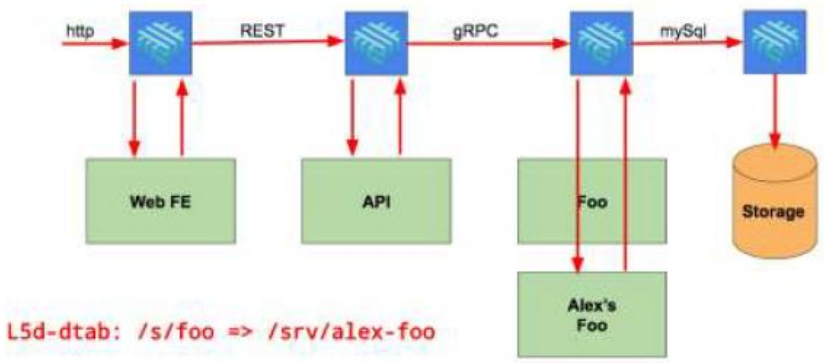

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: recommendation
  namespace: tutorial
spec:
  hosts:
  - recommendation
  http:
  - route:
    - destination:
        host: recommendation
        subset: version-v1
      weight: 90
    - destination:
        host: recommendation
        subset: version-v2
      weight: 10
```

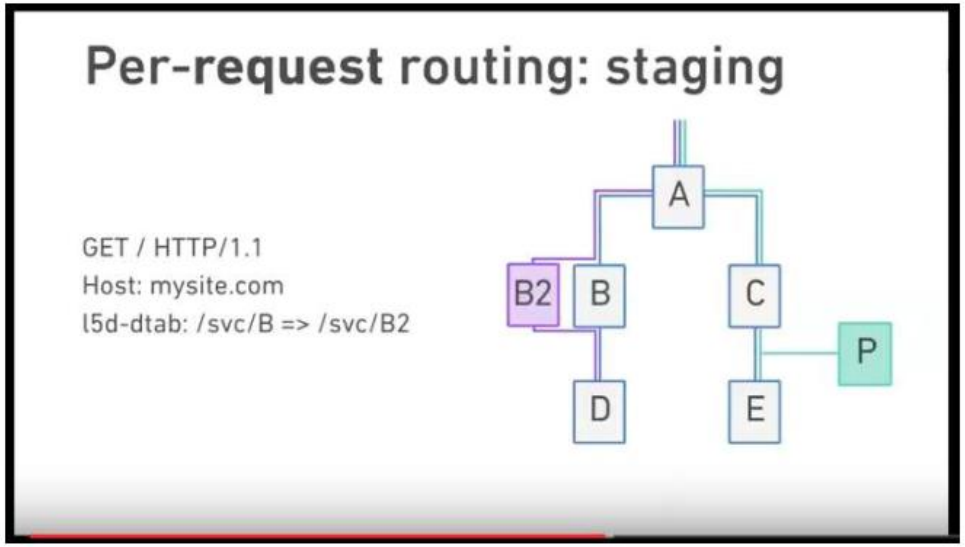
Traffic Control: Dark Launch

- Create duplicate or mirror traffic that duplicates traffic to a new version of your microservice to see how it behaves compared to the live application pod

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: recommendation
  namespace: tutorial
spec:
  hosts:
  - recommendation
  http:
  - route:
    - destination:
        host: recommendation
        subset: version-v1
    mirror:
      host: recommendation
      subset: version-v2
```

Traffic Routing:
Shadow, Fault Inject, Debug

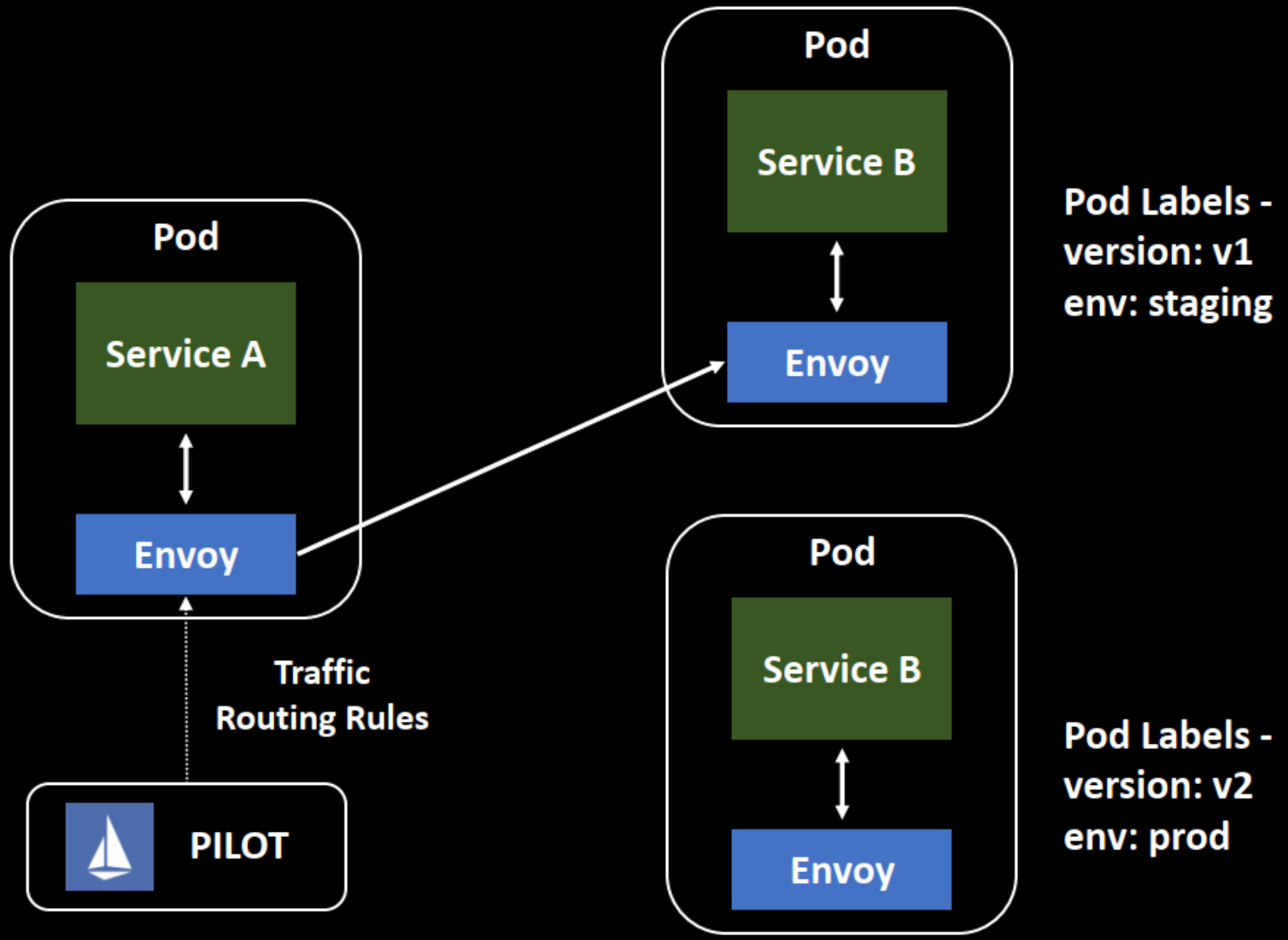




Traffic Routing

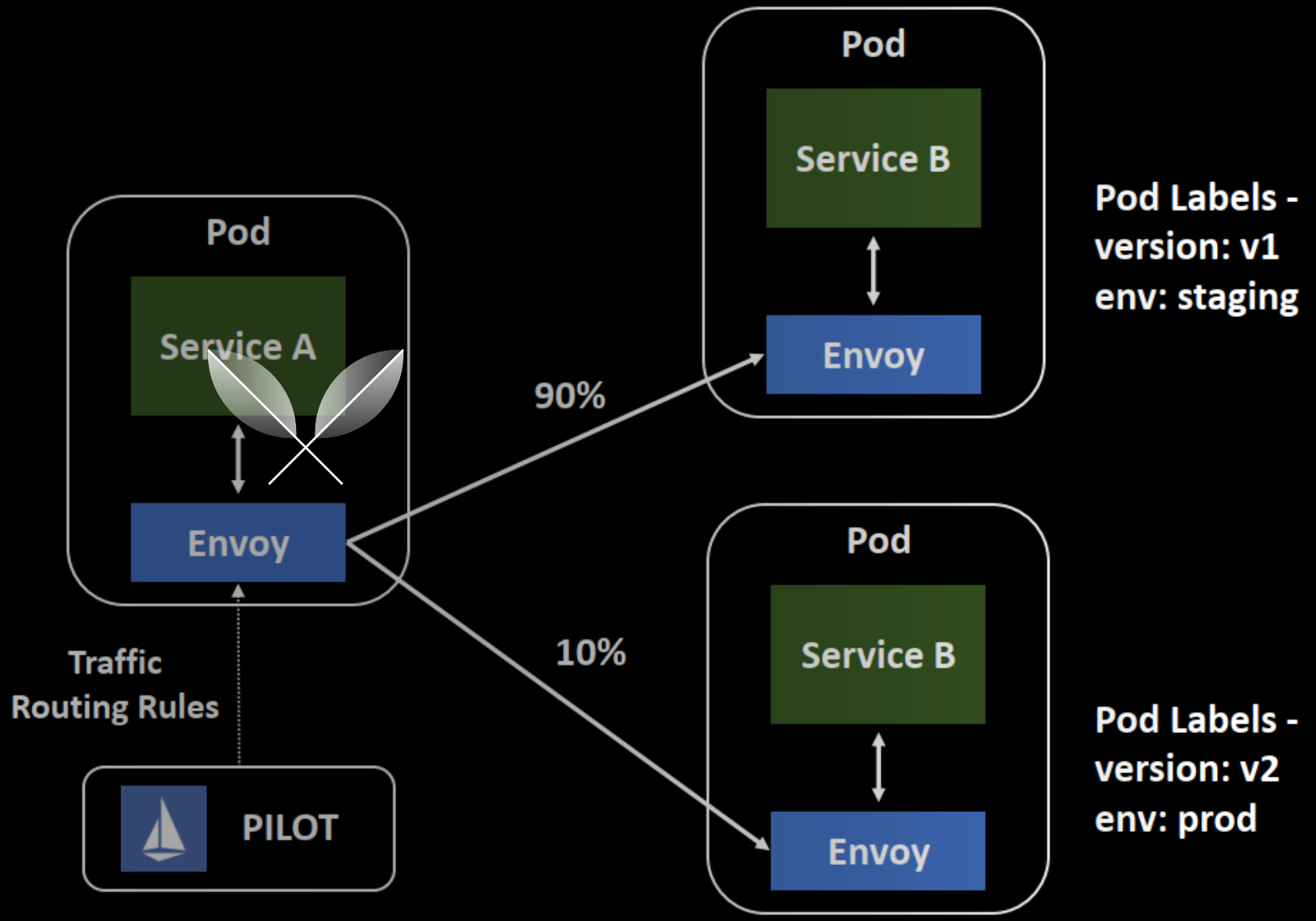
Route all traffic to v1 of ServiceB

```
ind: VirtualService
etadata:
 name: serviceB
pec:
 hosts:
 - serviceB
 http:
 - route:
   - destination:
       host: serviceB
       subset: v1
```



Canary Deployment

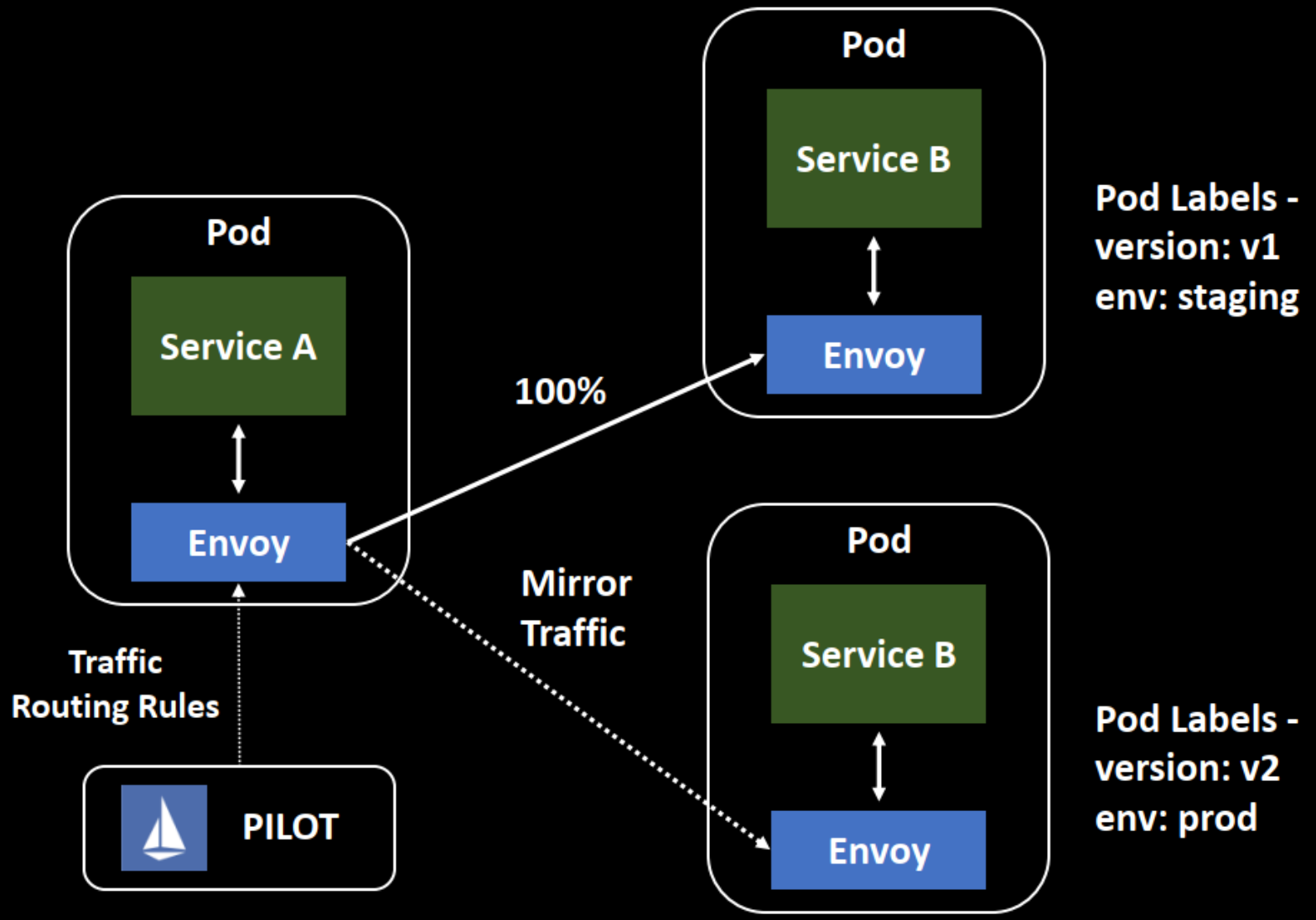
```
# Percentage based Traffic Split

kind: VirtualService
metadata:
  name: serviceB
spec:
  hosts:
   - serviceB
  http:
  - route:
    - destination:
        host: serviceB
        subset: v1
      weight: 90
    - destination:
        host: serviceB
        subset: v2
      weight: 10
```



Dark Launches

 # Traffic Mirroring

```
kind: VirtualService
metadata:
  name: serviceB
spec:
  hosts:
   - serviceB
  http:
  - route:
    - destination:
        host: serviceB
        subset: v1
      weight: 100
    mirror:
        host: serviceB
        subset: v2
```



Service Resiliency with Istio: Load balancing Rules

- **Options**

  Round Robin – Even distributes load across endpoints

  Random – Evenly distributes load without any order

  Least_Conn – Picks to two random hosts and determines which of the two has fewer outstanding requests to that endpoint

```
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: recommendation
  namespace: tutorial
spec:
  host: recommendation
  trafficPolicy:
    loadBalancer:
      simple: RANDOM
```

Service Resiliency: Timeout

- Calls to reduce unpredictable behavior.
- Did the service fail?

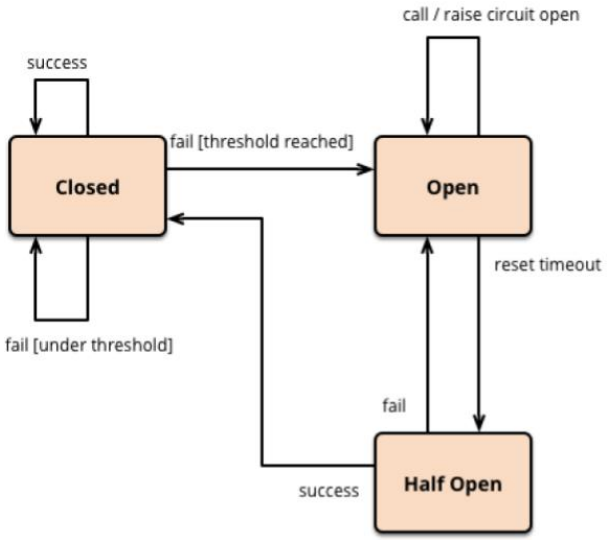
```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  creationTimestamp: null
  name: recommendation
  namespace: tutorial
spec:
  hosts:
  - recommendation
  http:
  - route:
    - destination:
        host: recommendation
    timeout: 1.000s
```

Service Resiliency: Retry

Retry capabilities allow more resilient services

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: recommendation
  namespace: tutorial
spec:
  hosts:
  - recommendation
  http:
  - route:
    - destination:
        host: recommendation
    retries:
      attempts: 3
      perTryTimeout: 2s
```

Circuit Breaker Rule – Service Resiliency




```
destination: serviceB.example.cluster.local policy:
- tags:
    version: v1
  circuitBreaker:
    simpleCb:
      maxConnections: 100
      httpMaxRequests: 1000
      httpMaxRequestsPerConnection: 10
      httpConsecutiveErrors: 7
      sleepWindow: 15m
      httpDetectionInterval: 5m
```

```
- protocol: http
  client:
    failureAccrual:
      kind: io.l5d.successRate
      successRate: 0.9
      requests: 20
      backoff:
        kind: constant
        ms: 10000
```

Service Resiliency: Multiple Concurrent Requests

- Circuit Breaker Rule

```
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  creationTimestamp: null
  name: recommendation
  namespace: tutorial
spec:
  host: recommendation
  subsets:
    - name: version-v1
      labels:
        version: v1
    - name: version-v2
      labels:
        version: v2
      trafficPolicy:
        connectionPool:
          http:
            http1MaxPendingRequests: 1
            maxRequestsPerConnection: 1
          tcp:
            maxConnections: 1
        outlierDetection:
          baseEjectionTime: 120.000s
          consecutiveErrors: 1
          interval: 1.000s
          maxEjectionPercent: 100
```

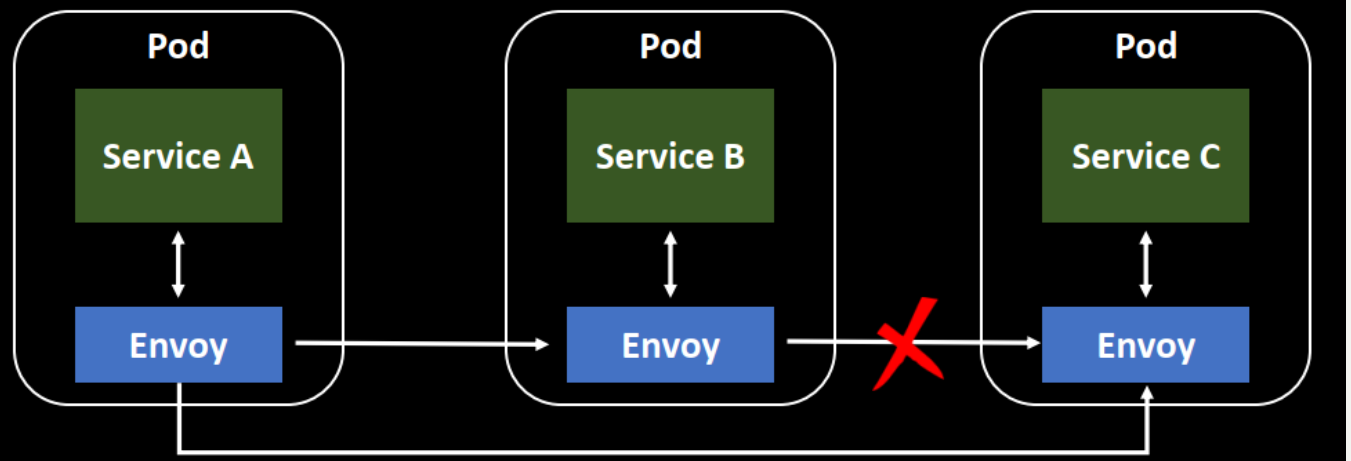
Service Resiliency: Introduce delays to stress test

- Introduces 7 seconds of delay into 50% of responses from the recommendation

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  creationTimestamp: null
  name: recommendation
  namespace: tutorial
spec:
  hosts:
  - recommendation
  http:
  - fault:
      delay:
        fixedDelay: 7.000s
        percent: 50
    route:
    - destination:
        host: recommendation
        subset: app-recommendation
```

Circuit Breaker

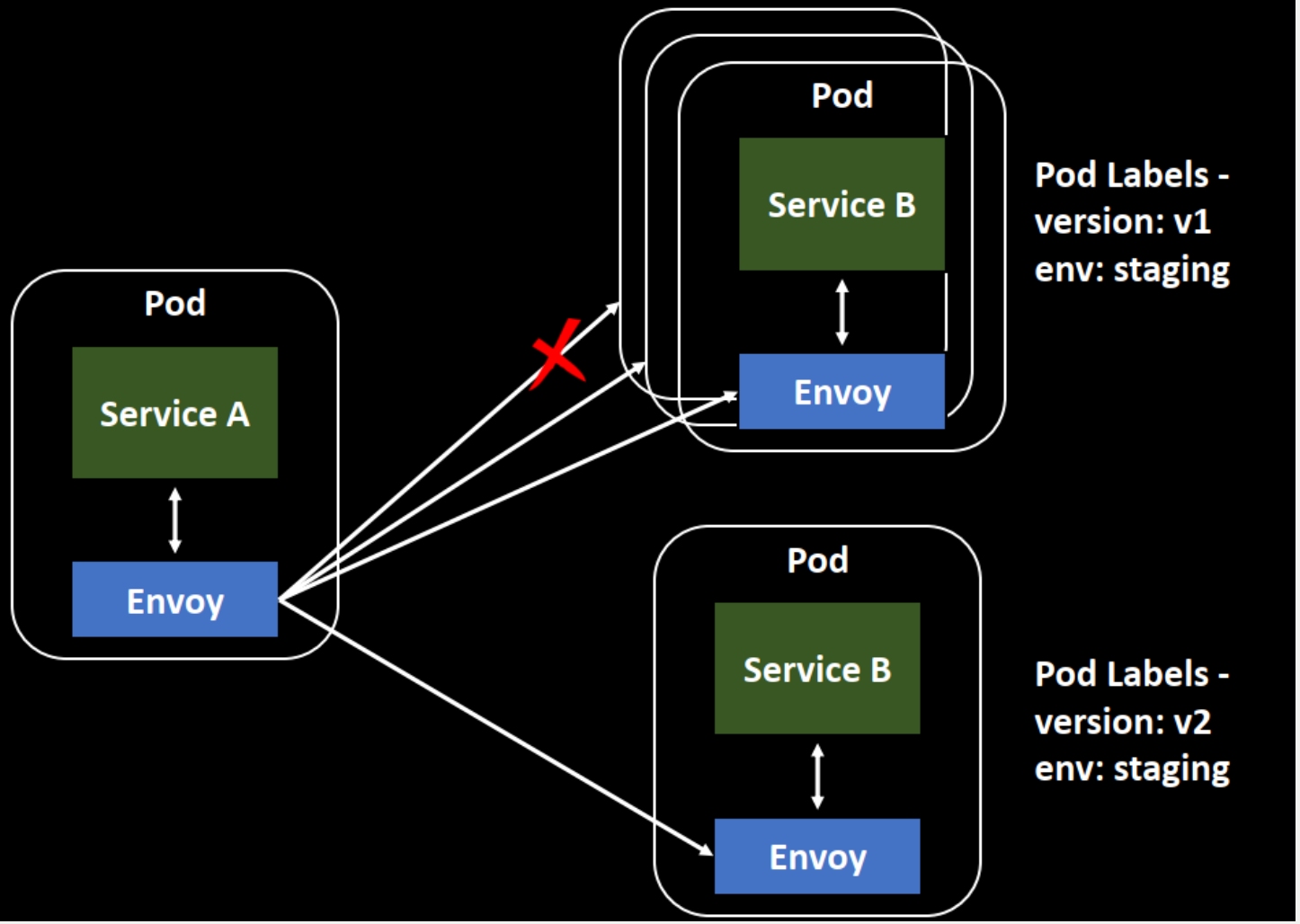
```
# Limits the number of concurrent
connections and requests

kind: DestinationRule
metadata:
  name: serviceC
spec:
  hosts:
  - serviceC
  trafficPolicy:
    connectionPool:
     http:
       http1MaxPendingRequests: 10
       maxRequestsPerConnection: 1
     tcp:
       maxConnections: 1
```



Outlier Detection

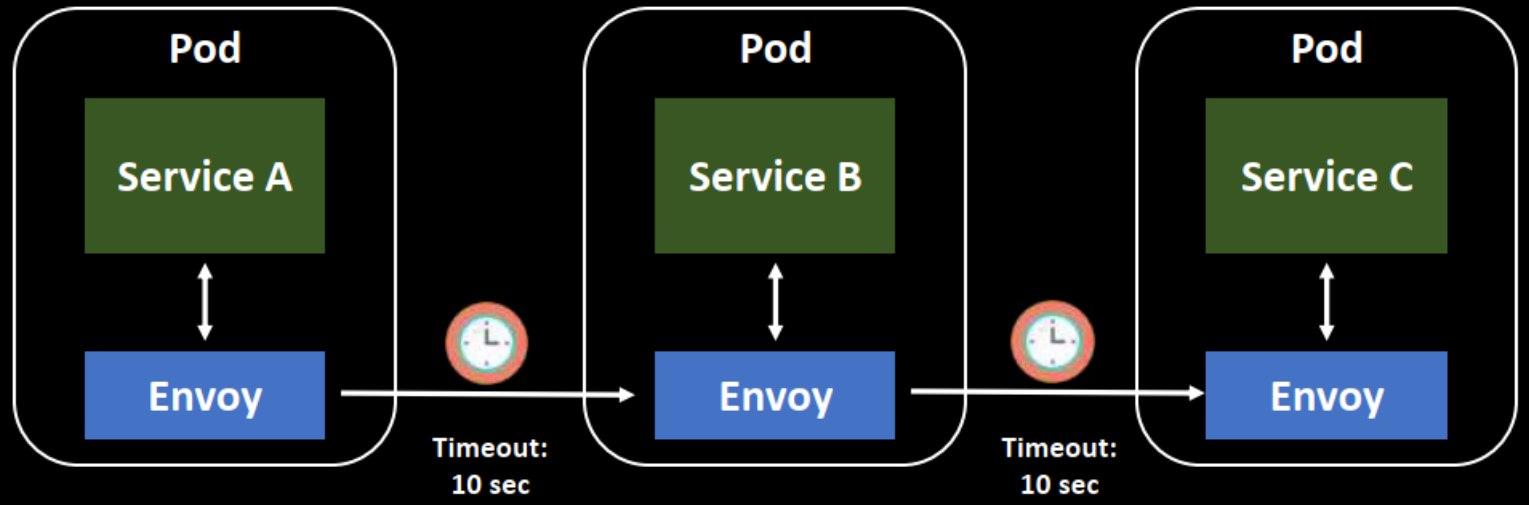
 # Detect faulty instances in the pool & remove from traffic routing

```yaml
kind: DestinationRule
metadata:
  name: serviceB
spec:
  hosts:
  - serviceB
  trafficPolicy:
    outlierDetection:
      baseEjectionTime: 20s
      consecutiveErrors: 3
      interval: 10s
      maxEjectionPercent: 100
```



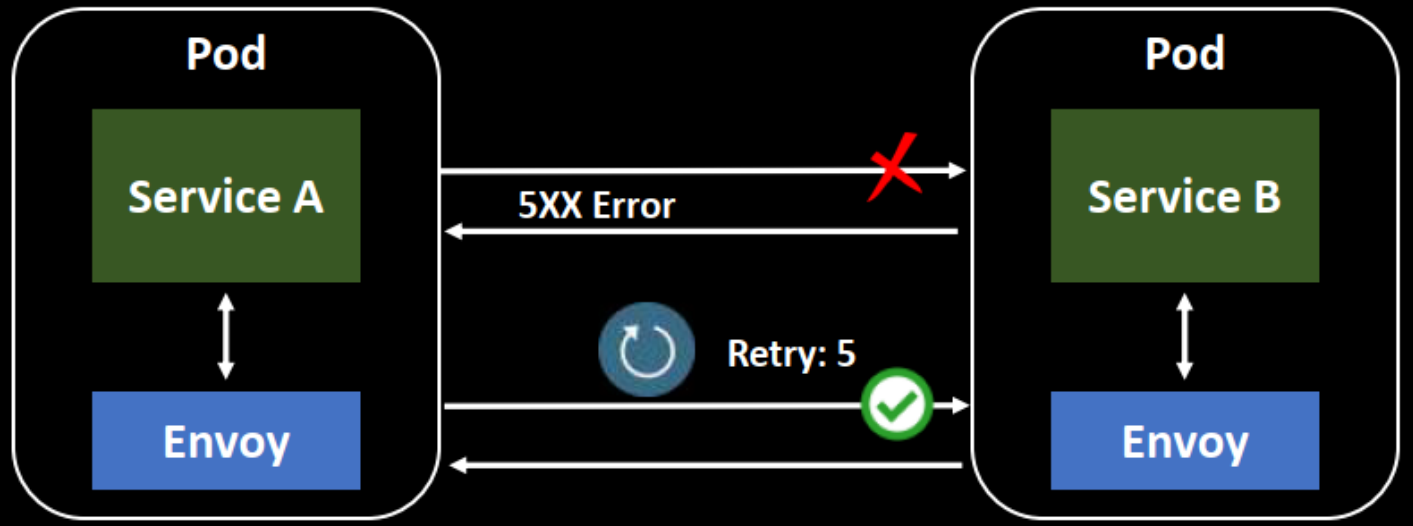
Timeout

```
# Timeout strategy for service
communication over network

kind: VirtualService
metadata:
  name: serviceB
spec:
  hosts:
  - serviceB
  http:
  - route:
    - destination:
        host: serviceB
    timeout: 10s
```



Istio Retry Policy

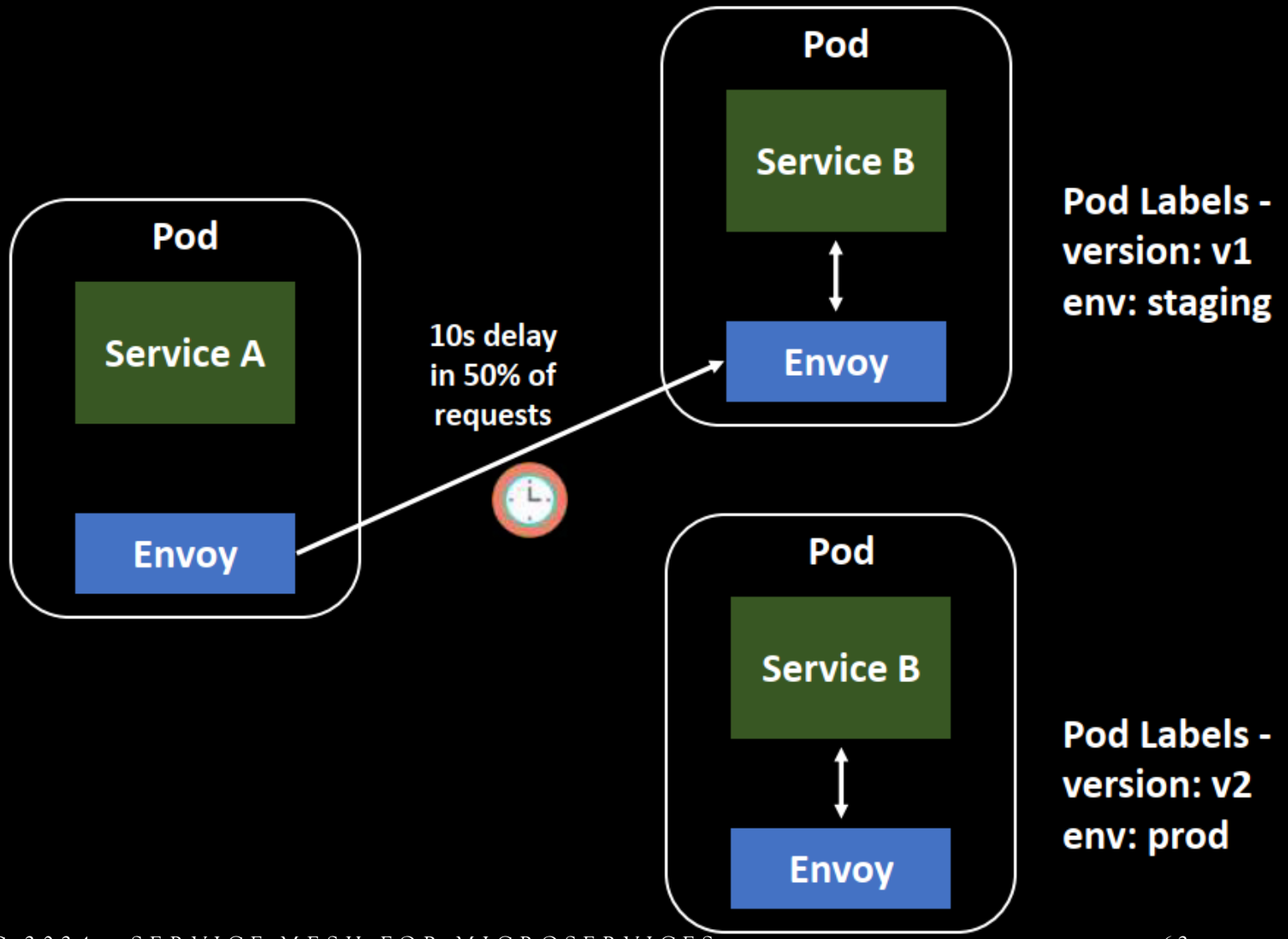
```
# Retry strategy for service
communication over network

kind: VirtualService
metadata:
  name: serviceB
spec:
  hosts:
  - serviceB
  http:
  - route:
    - destination:
        host: serviceB
    retries:
      attempts: 3
      perTryTimeout: 2s
```



Chaos Testing – Inject Delays

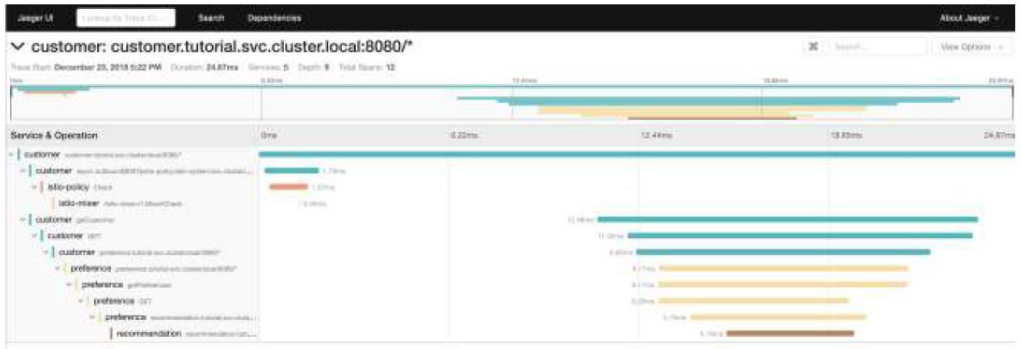
 # Create rule to delay traffic to ServiceB v1

```
kind: VirtualService
metadata:
  name: serviceB
spec:
  hosts:
  - serviceB
  http:
  - fault:
      delay:
        fixedDelay: 10s
        percent: 50
    route:
    - destination:
        host: serviceB
        subset: v1
```



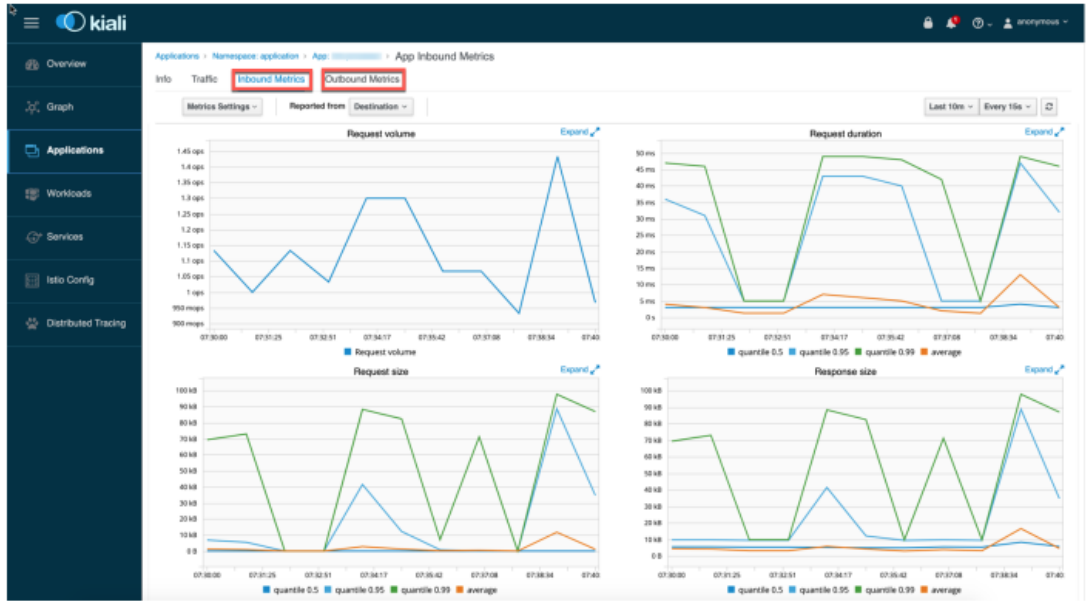
Observability – Tracing and Metrics

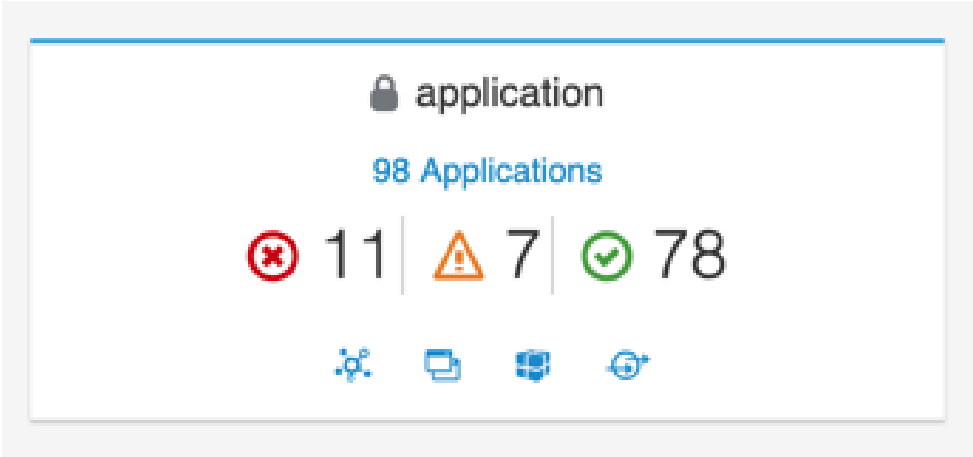
- Jaeger (Outsourced by Uber)



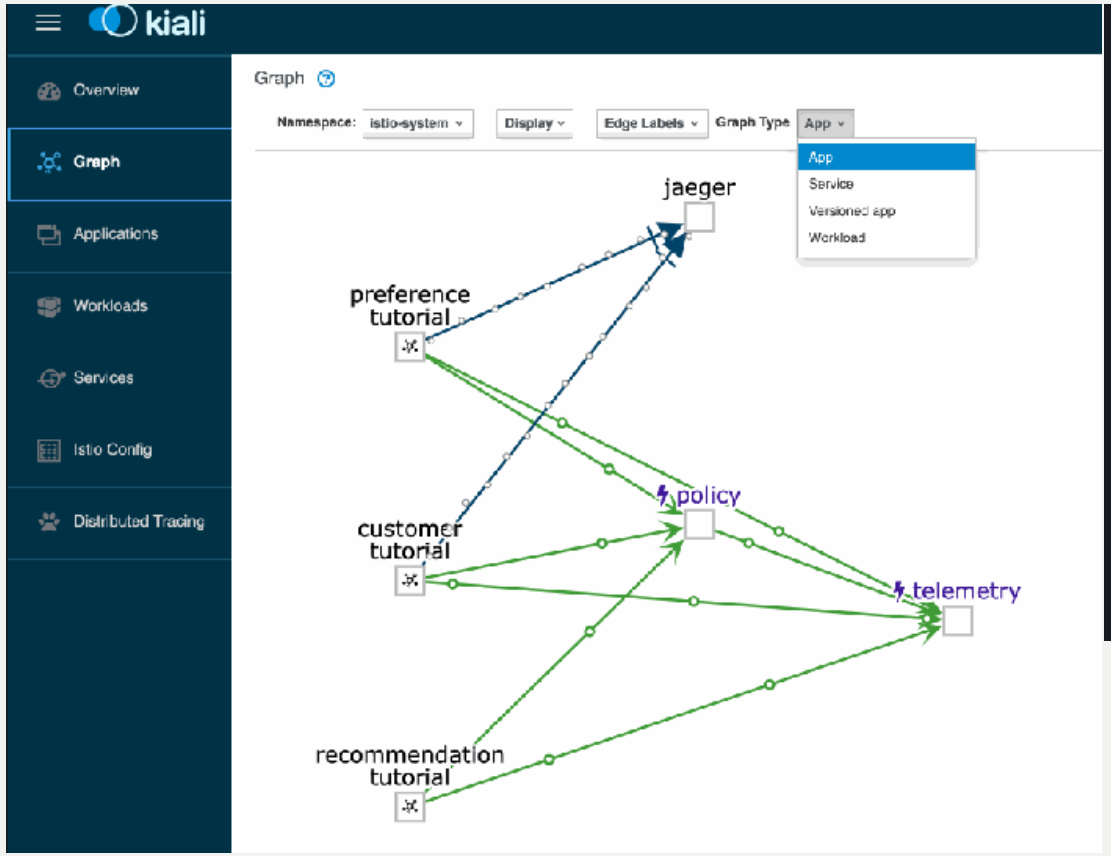
Observability: Service Graphs

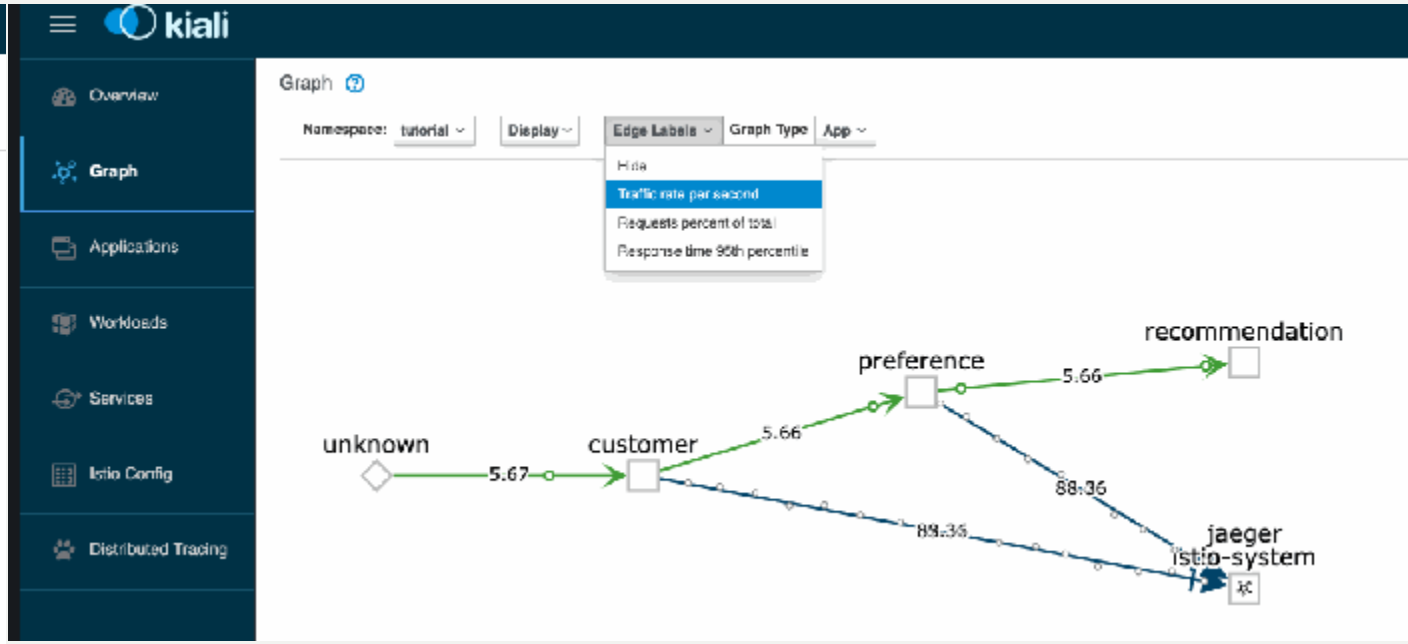
Visualization



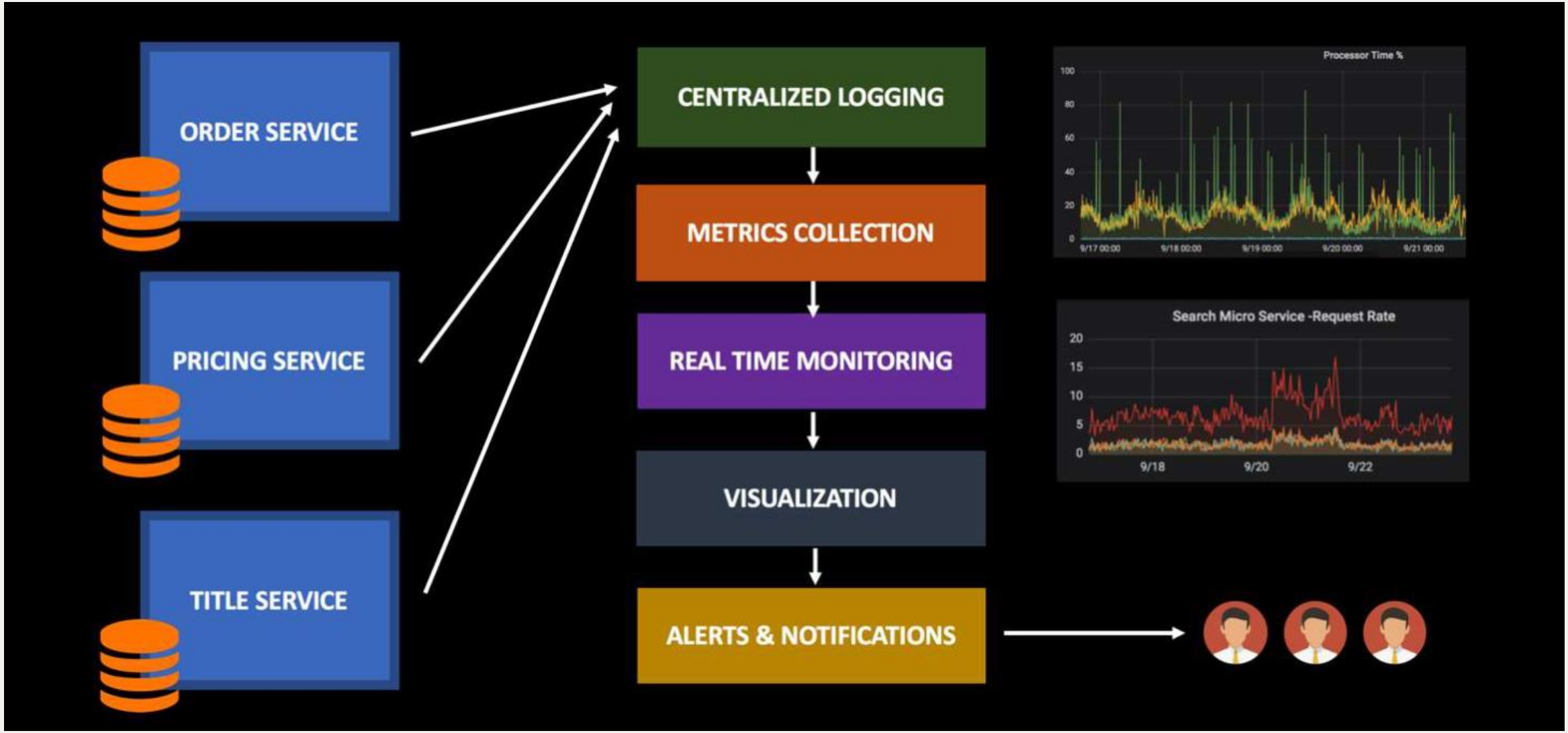


Visualization – Service Graphs

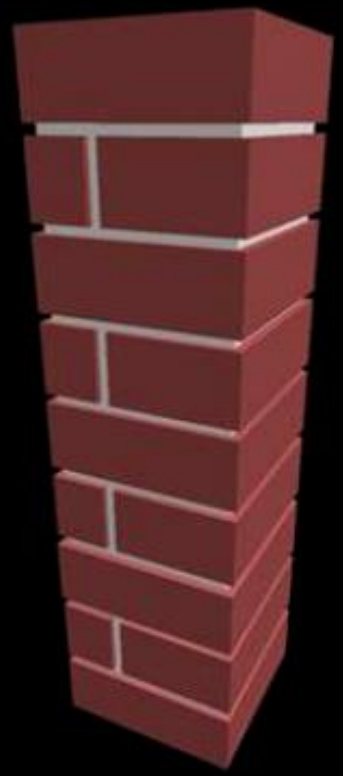




https://dzone.com/articles/visualizing-the-istio-service-mesh-using-kiali


ORDER SERVICE
PRICING SERVICE
TITLE SERVICE
CENTRALIZED LOGGING
METRICS COLLECTION
REAL TIME MONITORING
VISUALIZATION
ALERTS & NOTIFICATIONS
Processor Time %
100
80
60
40
20
0
9/17 00:00
9/18 00:00
9/19 00:00
9/20 00:00
9/21 00:00
Search Micro Service -Request Rate
20
15
10
5
0
9/18
9/20
9/22

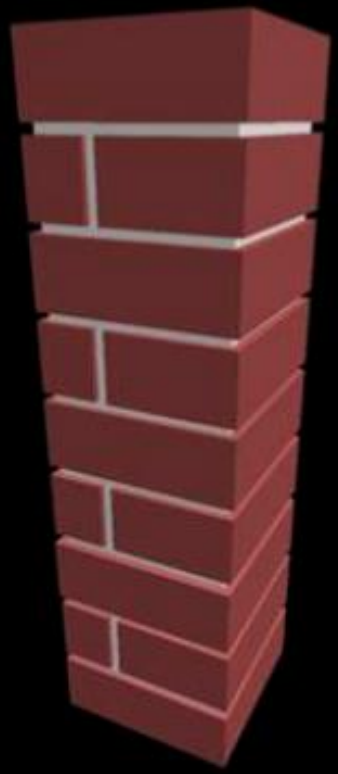
LOGGING



sumologic

fluentd

METRICS



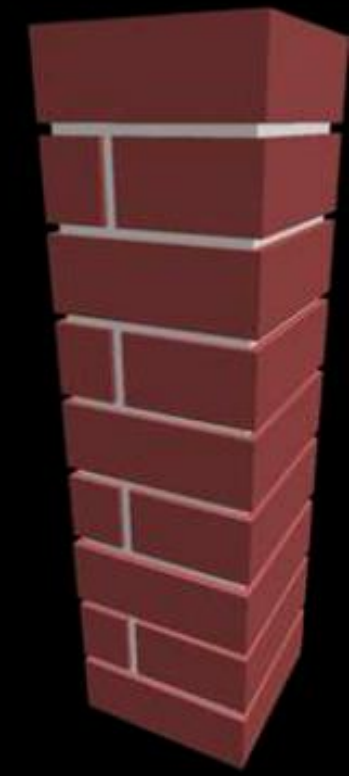
DATADOG

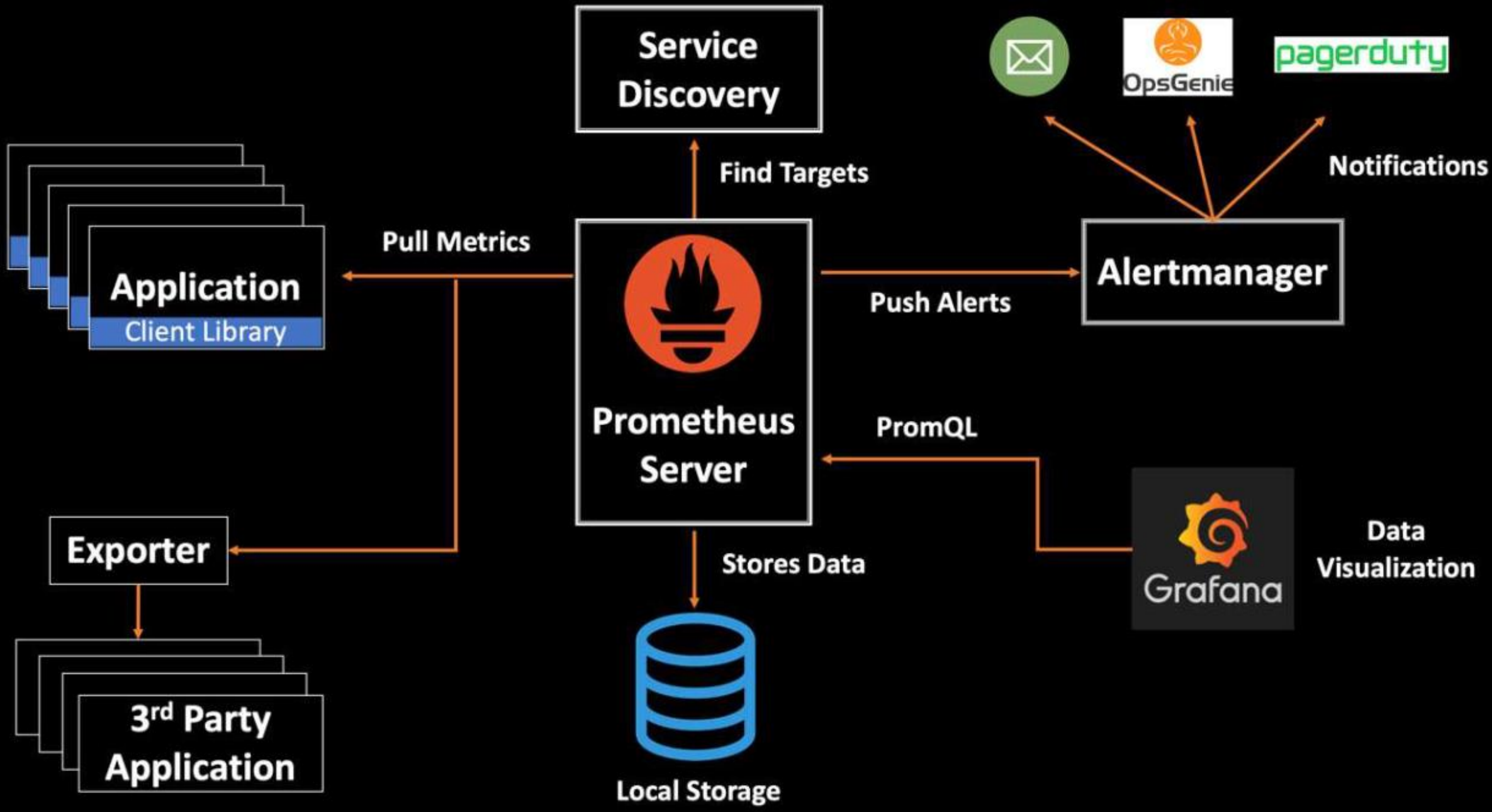
OpsGenie

kiali

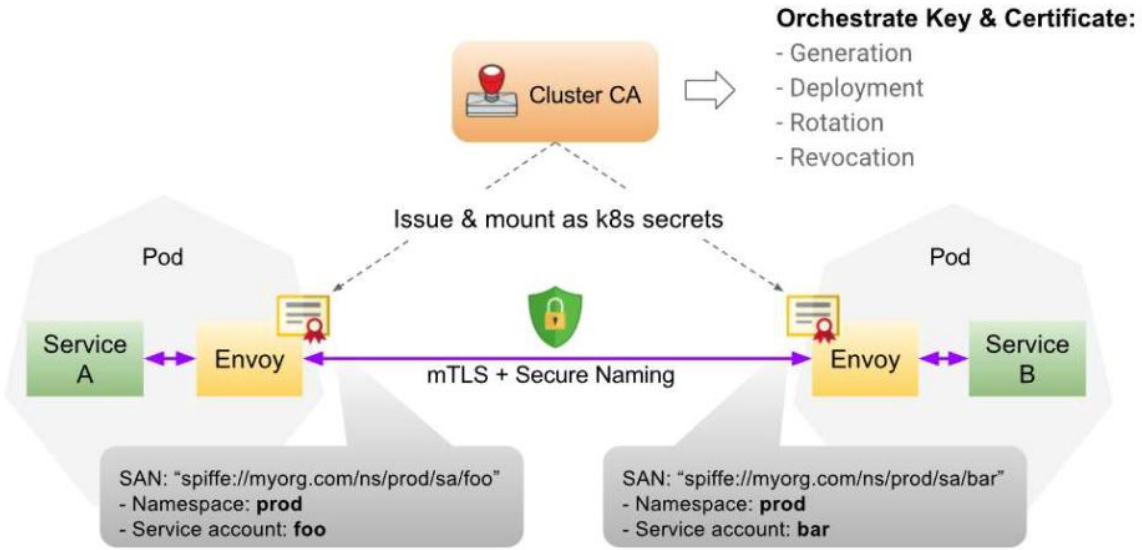
Grafana

TRACING




Service Discovery
Find Targets
Notifications
OpsGenie
pagerduty
Application
Client Library
Pull Metrics
Prometheus Server
Alertmanager
Push Alerts
PromQL
Exporter
Stores Data
Grafana
Data Visualization
3rd Party Application
Local Storage

Security – Enforce mTLS



Security – Enforce mTLS

```
apiVersion: "authentication.istio.io/v1alpha1"
kind: "Policy"
metadata:
  name: "default"
  namespace: "tutorial"
spec:
  peers:
  - mtls: {}
```

```
apiVersion: "networking.istio.io/v1alpha3"
kind: "DestinationRule"
metadata:
  name: "default"
  namespace: "tutorial"
spec:
  host: "*.tutorial.svc.cluster.local"
  trafficPolicy:
    tls:
      mode: ISTIO_MUTUAL
```

Security – Access Control with Mixer Policies

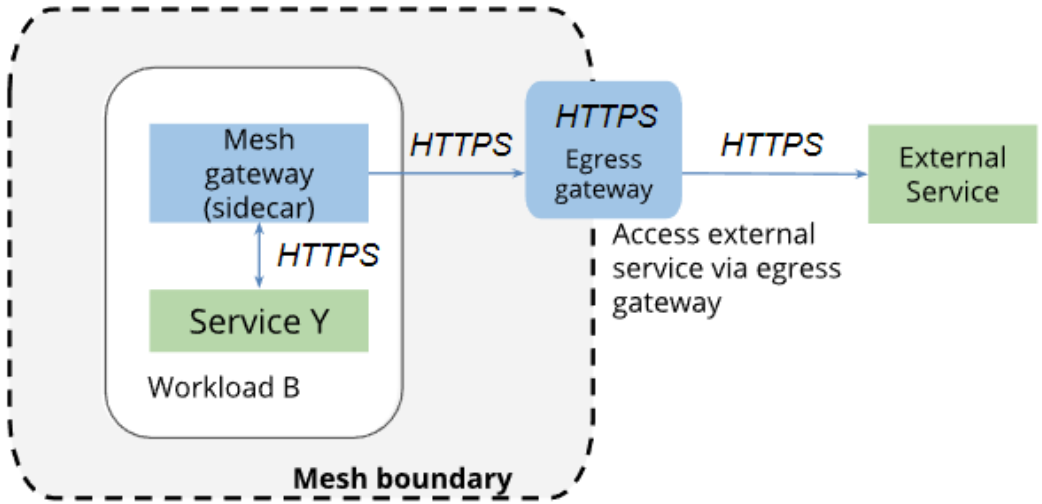
https://istio.io/latest/blog/2018/egress-monitoring-access-control/

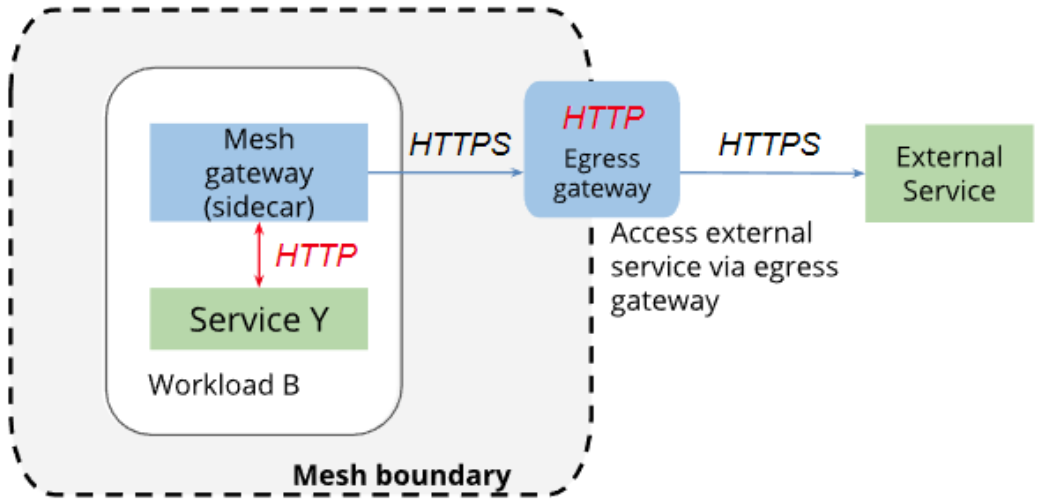
```
metadata:
  name: direct-cnn-through-egress-gateway
spec:
  hosts:
  - edition.cnn.com
  gateways:
  - istio-egressgateway
  - mesh
  http:
  - match:
    - gateways:
      - mesh
      port: 80
    route:
    - destination:
        host: istio-egressgateway.istio-system.svc.cluster.local
        subset: cnn
```

```
  - match:
    - gateways:
      - istio-egressgateway
      port: 443
      uri:
        regex: "/health|/sport"
    route:
    - destination:
        host: edition.cnn.com
        port:
          number: 443
      weight: 100
EOF
```

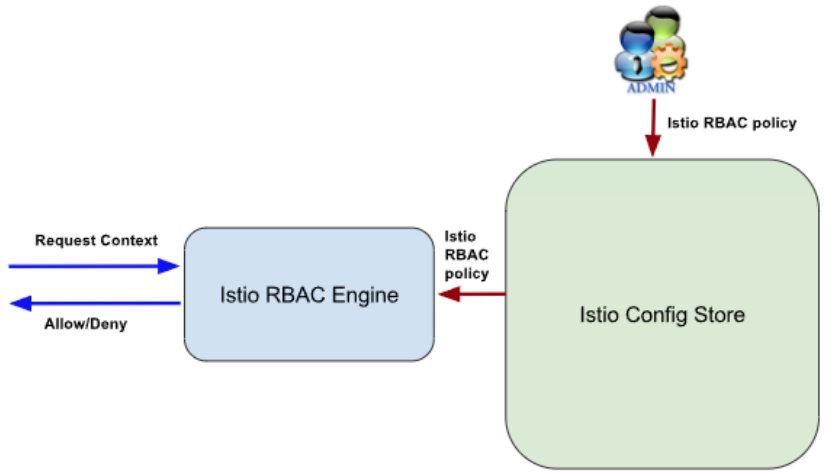
Different ways to implement Egress security with Istio

https://istio.io/latest/blog/2018/egress-monitoring-access-control/





Security – Role-Based Access Control (RBAC) with ISTIO



Security – RBAC

Istio RBAC policy

Istio RBAC introduces `ServiceRole` and `ServiceRoleBinding`, both of which are defined as Kubernetes CustomResourceDefinition (CRD) objects.

- `ServiceRole` defines a role for access to services in the mesh.
- `ServiceRoleBinding` grants a role to subjects (e.g., a user, a group, a service).

`ServiceRole`

A `ServiceRole` specification includes a list of rules. Each rule has the following standard fields:

- **services**: A list of service names, which are matched against the `action.service` field of the "requestcontext".
- **methods**: A list of method names which are matched against the `action.method` field of the "requestcontext". In the above "requestcontext", this is the HTTP or gRPC method. Note that gRPC methods are formatted in the form of "packageName.serviceName/methodName" (case sensitive).
- **paths**: A list of HTTP paths which are matched against the `action.path` field of the "requestcontext". It is ignored in gRPC case.

A `ServiceRole` specification only applies to the **namespace** specified in `"metadata"` section. The "services" and "methods" are required fields in a rule. "paths" is optional. If not specified or set to "*", it applies to "any" instance.

Concepts

- What is Istio?
- Traffic Management
- Security
  - Authentication Policy
  - Mutual TLS Authentication
  - Istio Role-Based Access Control (RBAC)
- Policies and Telemetry

Setup

Tasks

Guides

Performance and Scalability

Reference

`ServiceRoleBinding`

https://istio.io/v0.8/docs/concepts/security/rbac/

A `ServiceRoleBinding` specification includes two parts:

- **roleRef** refers to a `ServiceRole` resource **in the same namespace**.
- A list of **subjects** that are assigned the role.

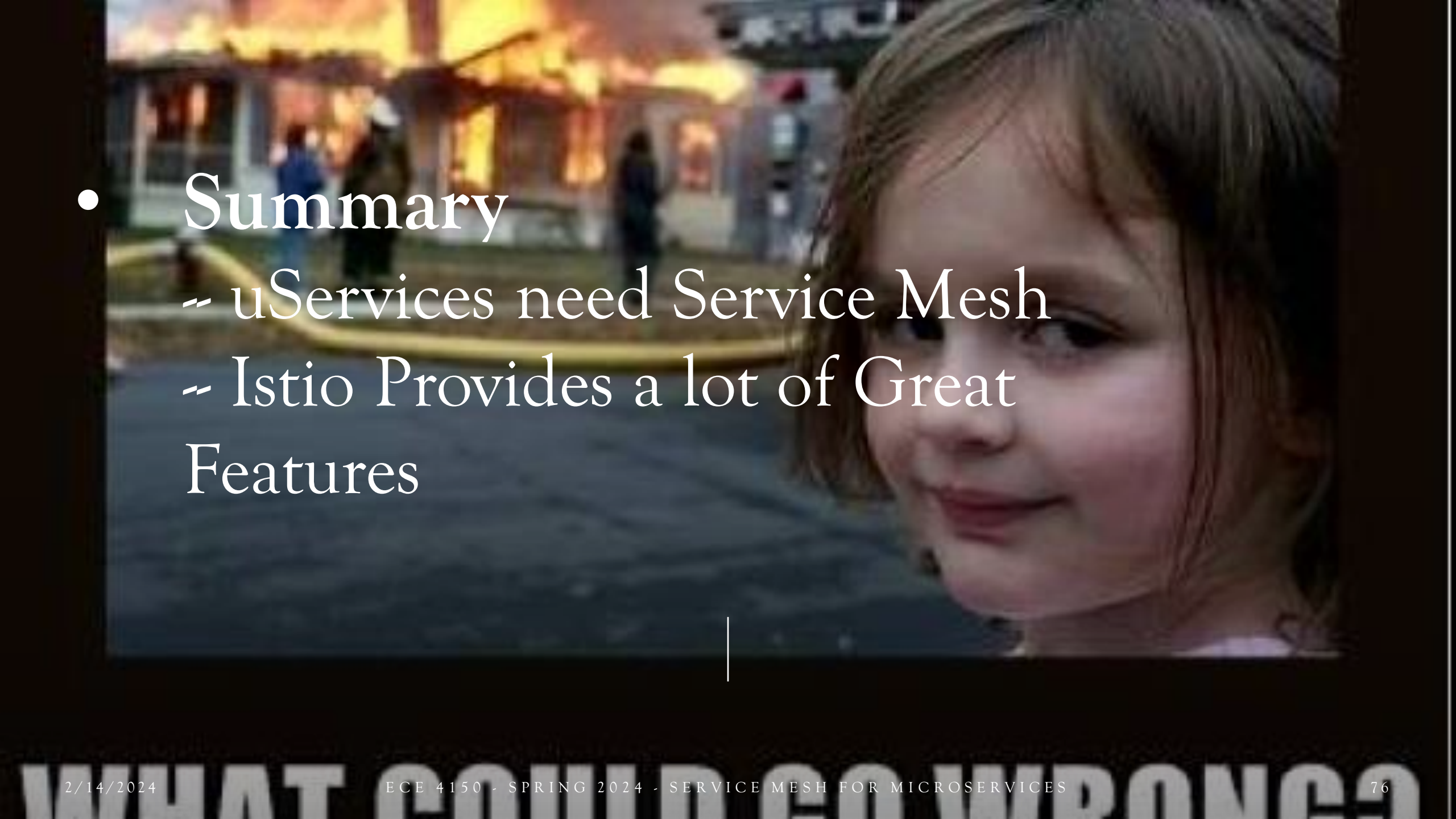
A subject can either be a “user”, or a “group”, or is represented with a set of “properties”. Each entry (“user” or “group” or an entry in “properties”) must match one of fields (“user” or “groups” or an entry in “properties”) in the “subject” part of the “requestcontext” instance.

Here is an example of `ServiceRoleBinding` resource “test-binding-products”, which binds two subjects to ServiceRole “product-viewer”:

- user “alice@yahoo.com”.
- “reviews.abc.svc.cluster.local” service in “abc” namespace.

```
apiVersion: "config.istio.io/v1alpha2"
kind: ServiceRoleBinding
metadata:
  name: test-binding-products
  namespace: default
spec:
  subjects:
  - user: "alice@yahoo.com"
  - properties:
      service: "reviews.abc.svc.cluster.local"
      namespace: "abc"
  roleRef:
    kind: ServiceRole
    name: "products-viewer"
```

In the case that you want to make a service(s) publicly accessible, you can use set the subject to `user: "*"`. This will assign a `ServiceRole` to all users/services.



- **Summary**
  - -- uServices need Service Mesh
  - -- Istio Provides a lot of Great Features