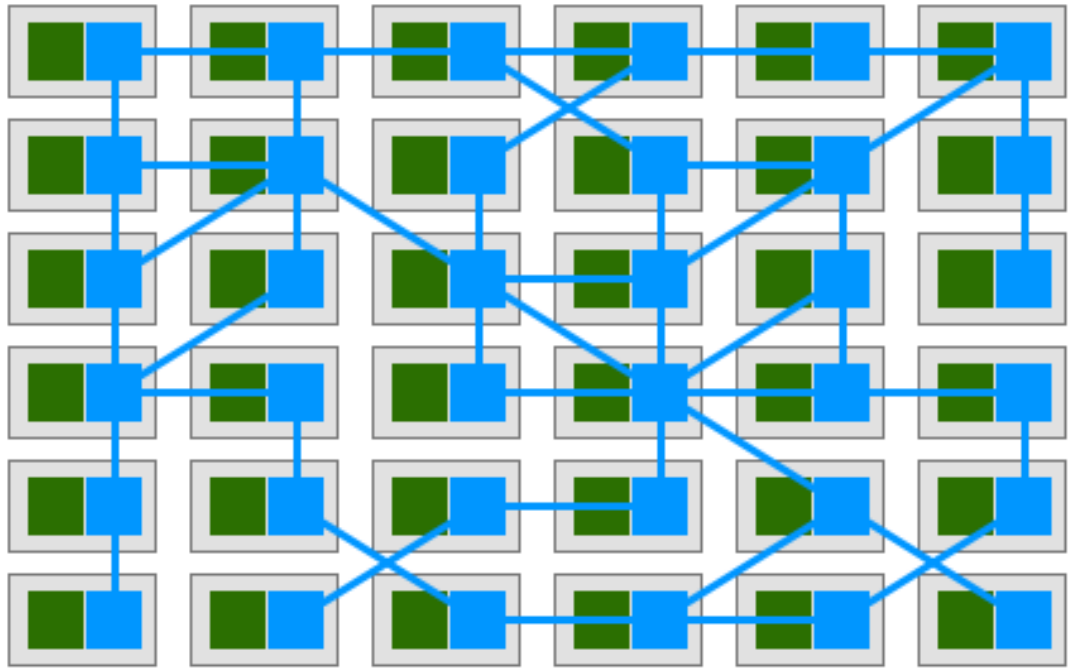
Basics of Service Mesh



Who Am I?

PhD in Telecommunications @ Budapest University of Technology

- Measurement and monitoring in Software Defined Networks
- Participating in 5G-PPP EU projects
- Graduated in the EIT Digital Doctoral School

Co-founder & CTO @ LeanNet Ltd.

- Evangelist of open networking solutions
- Currently focusing on cloud native transformation and the possible relation between SDN and cloud native

megyesi@leannet.eu

twitter.com/M3gy0

linkedin.com/in/M3gy0

Outline

Basics of service mesh

- The evolution of cloud computing
- Microservices leading to complexity
- The necessity of a new architectural component
- The sidecar proxy deployment strategy
- Data plane vs. control plane in a service mesh

Choices for realizing a service mesh

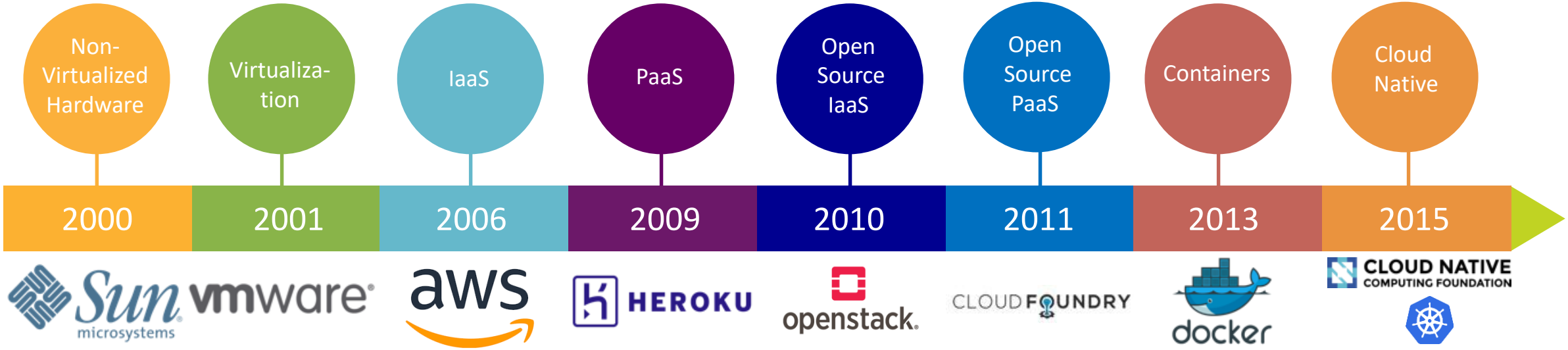
- Linkerd
- Linkerd2 (previously known as Conduit) → short demo as well!
- Envoy as the universal dataplane for
  - Istio
  - AWS App Mesh
  - Consul

From Servers Through Virtualization to Cloud Native

CLOUD NATIVE COMPUTING FOUNDATION

Cloud native computing uses an open source software stack to:

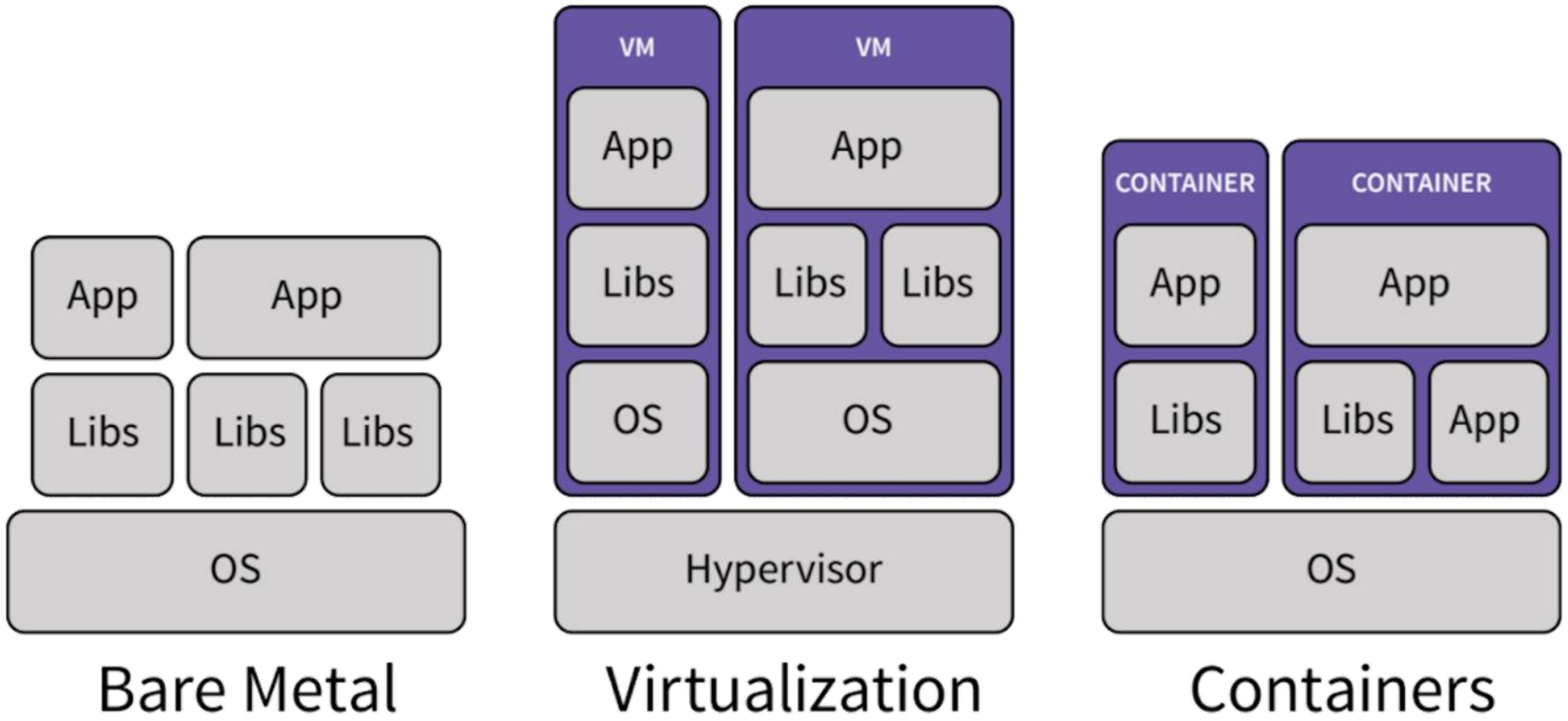
- segment applications into **microservices**,
- package each part into its own **container,**
- and dynamically **orchestrate** those containers to optimize resource utilization



What is a Container?

Containers are an application-centric way to deliver high-performing, scalable applications on the infrastructure of your choice

- A **bundle** of the **application** code along with its **runtime** and **dependencies**
- It creates an **isolated executable** environment, also known as container image
- It can be **deployed** on the platform of your choice, such as desktops, servers, VMs or in the cloud



docker

rkt

LXC LXD

kata containers

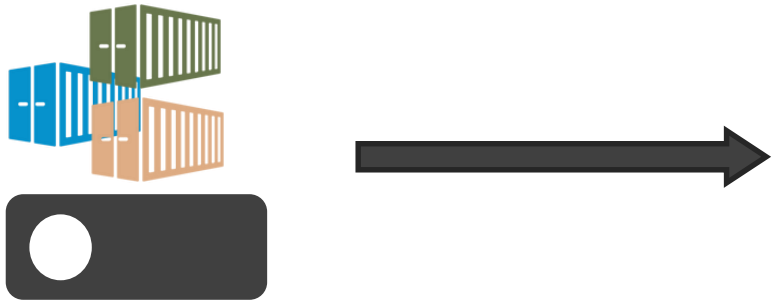
What is Container Orchestration?

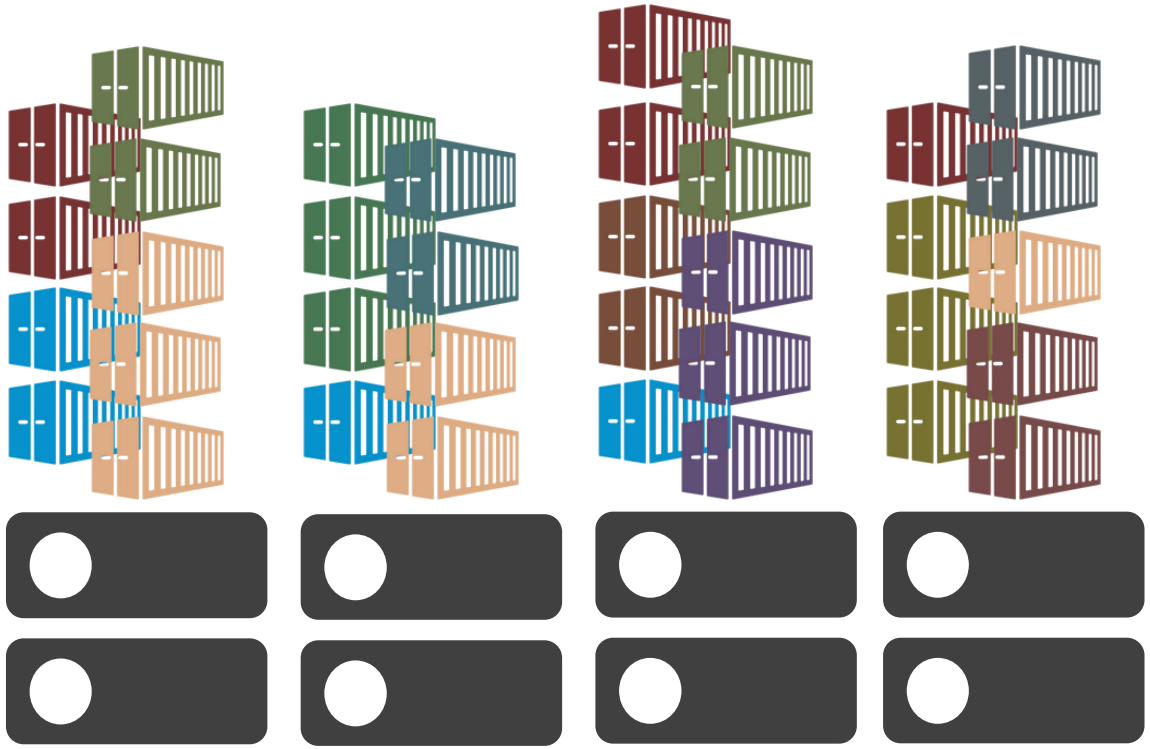
Running a few containers on a server is easy ☺

Running hundreds of containers on dozens of servers can be handled manually ☹

So container orchestrators:

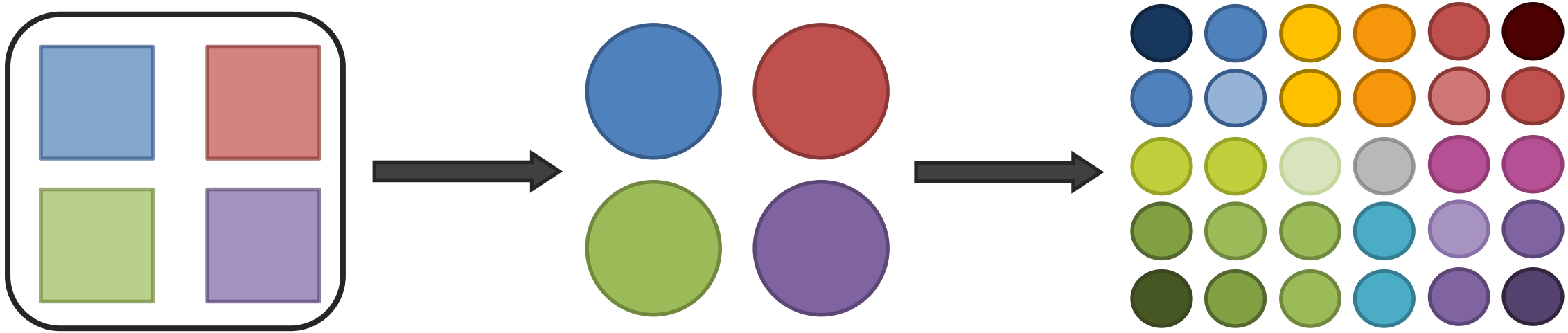
- group hosts together to form a cluster
- optimally schedule containers to run on different hosts
- guarantees that every container can talk to each other
- can scale out containers on-demand
- can do update/rollback without any downtime





What are Microservices?

Microservice architecture is software development form that structures an application as a collection of **loosely coupled services** having **bounded context**, which implement business capabilities. Microservices enable the **continuous delivery/deployment** of large, complex applications.



Monolithic software

- Vertically scalable
- Hard to maintain and evolve
- Very long build / test / release cycles
- Always fixing bugs
- Lack of innovation

Service Oriented Architecture

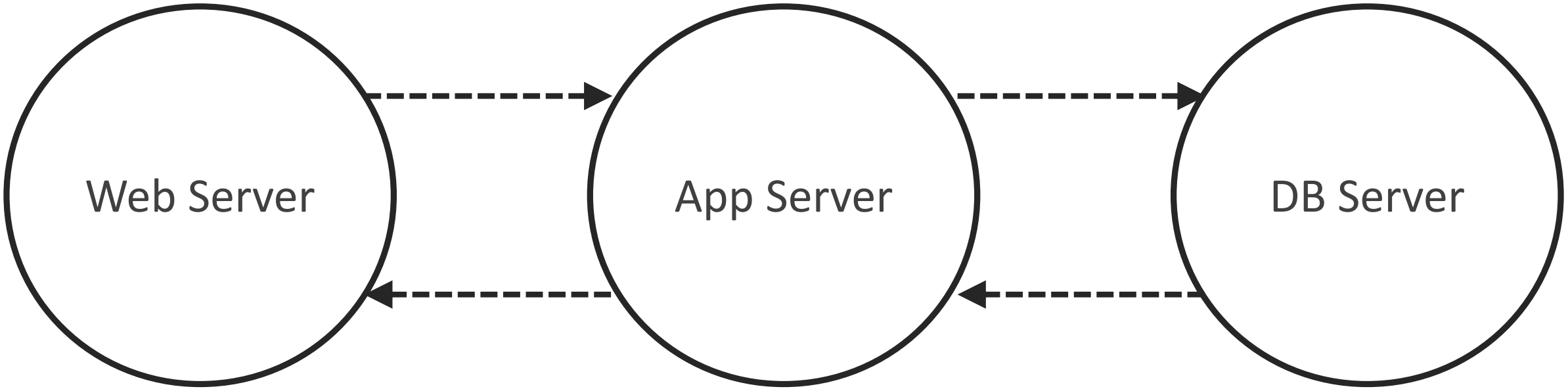
Microservices

- Horizontally scalable
- Services are easy to maintain
- Very short build / test / release cycles
- Easy to innovate

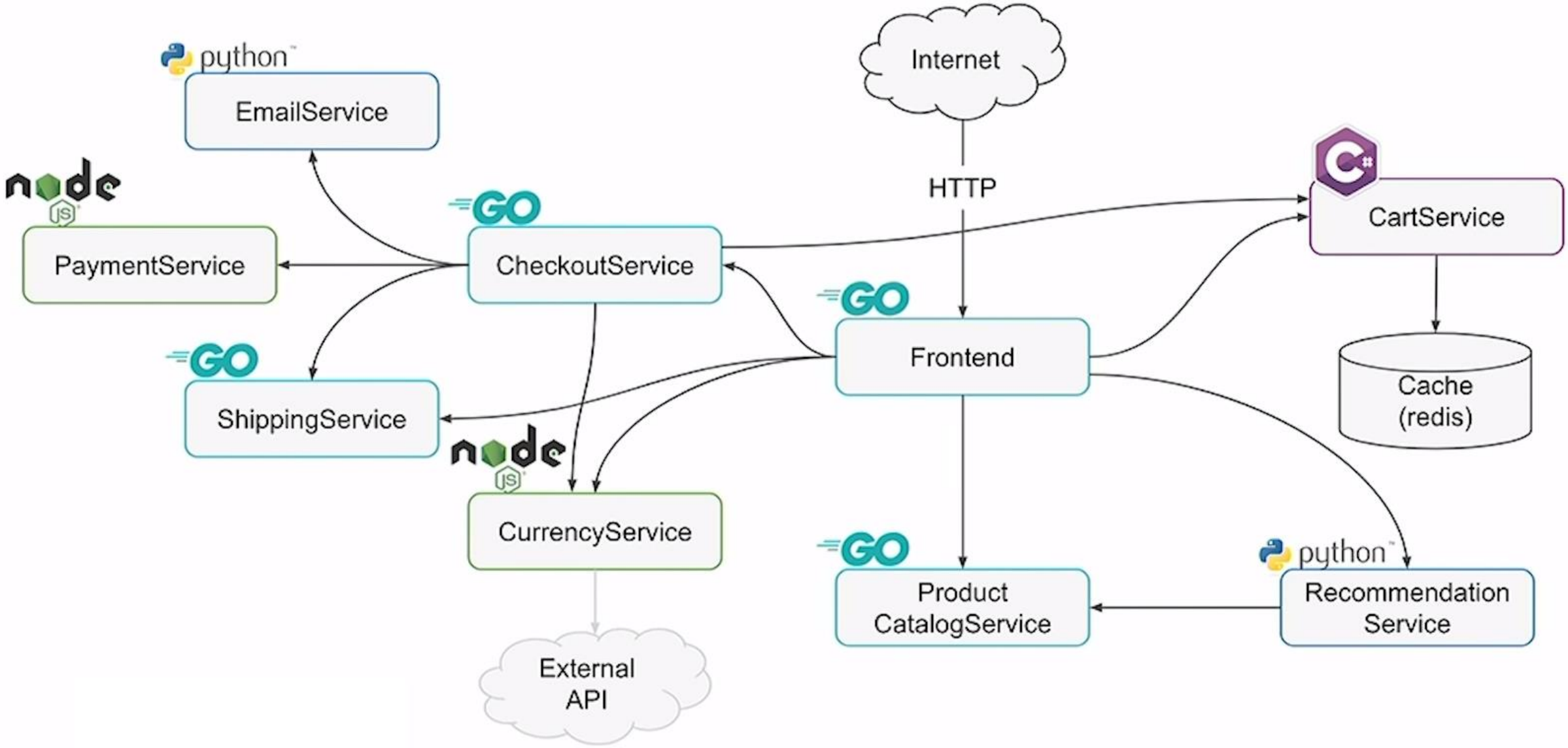
Cloud Native Changes the Fundamental Abstractions to Work With

Virtual machines	→	Containers
Data centers	→	Orchestrated environments
Hardware redundancy	→	Design for failure
Servers	→	Services
IP addresses, DNS	→	Service discovery
Server monitoring	→	Service monitoring
Monolithic applications	→	Microservices
TCP/IP	→	gRPC, REST

This is not a Microservice Architecture!

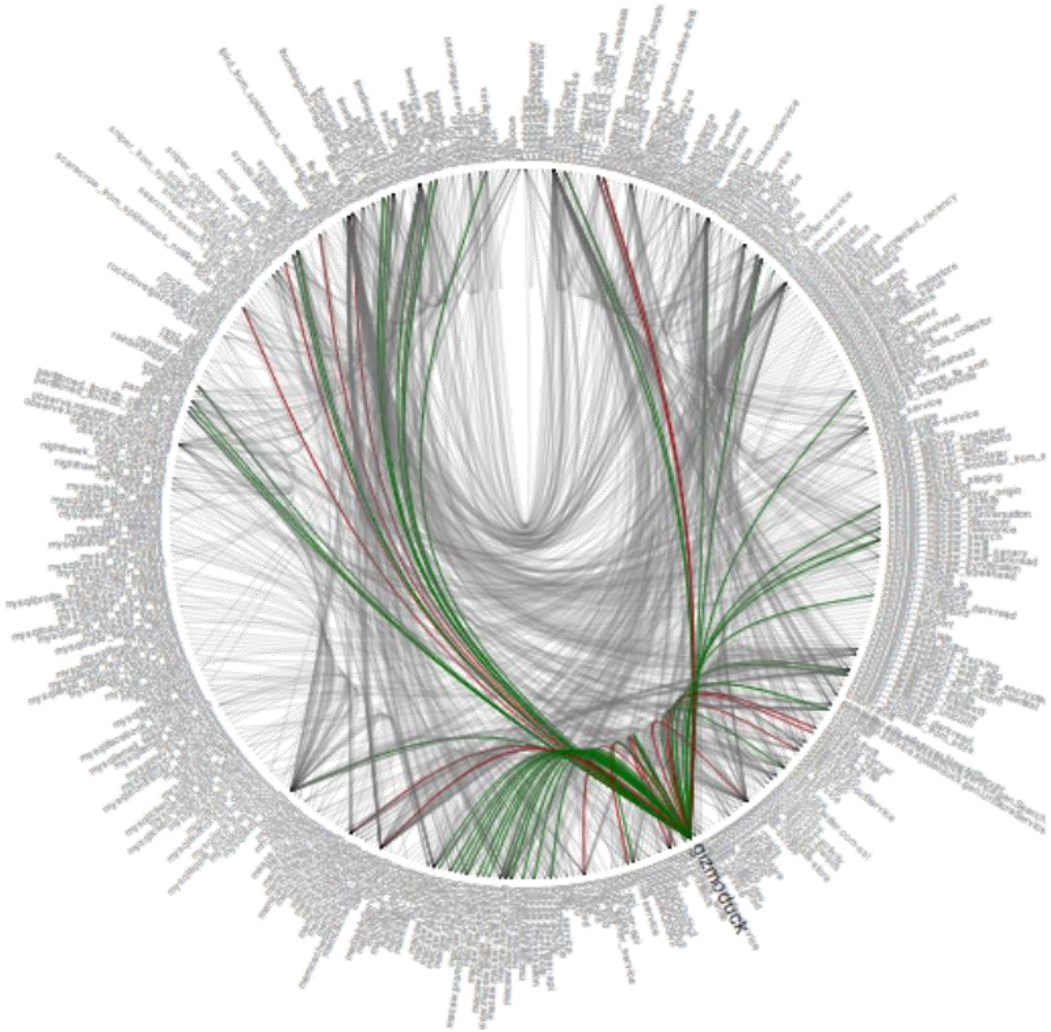


This is Getting There….

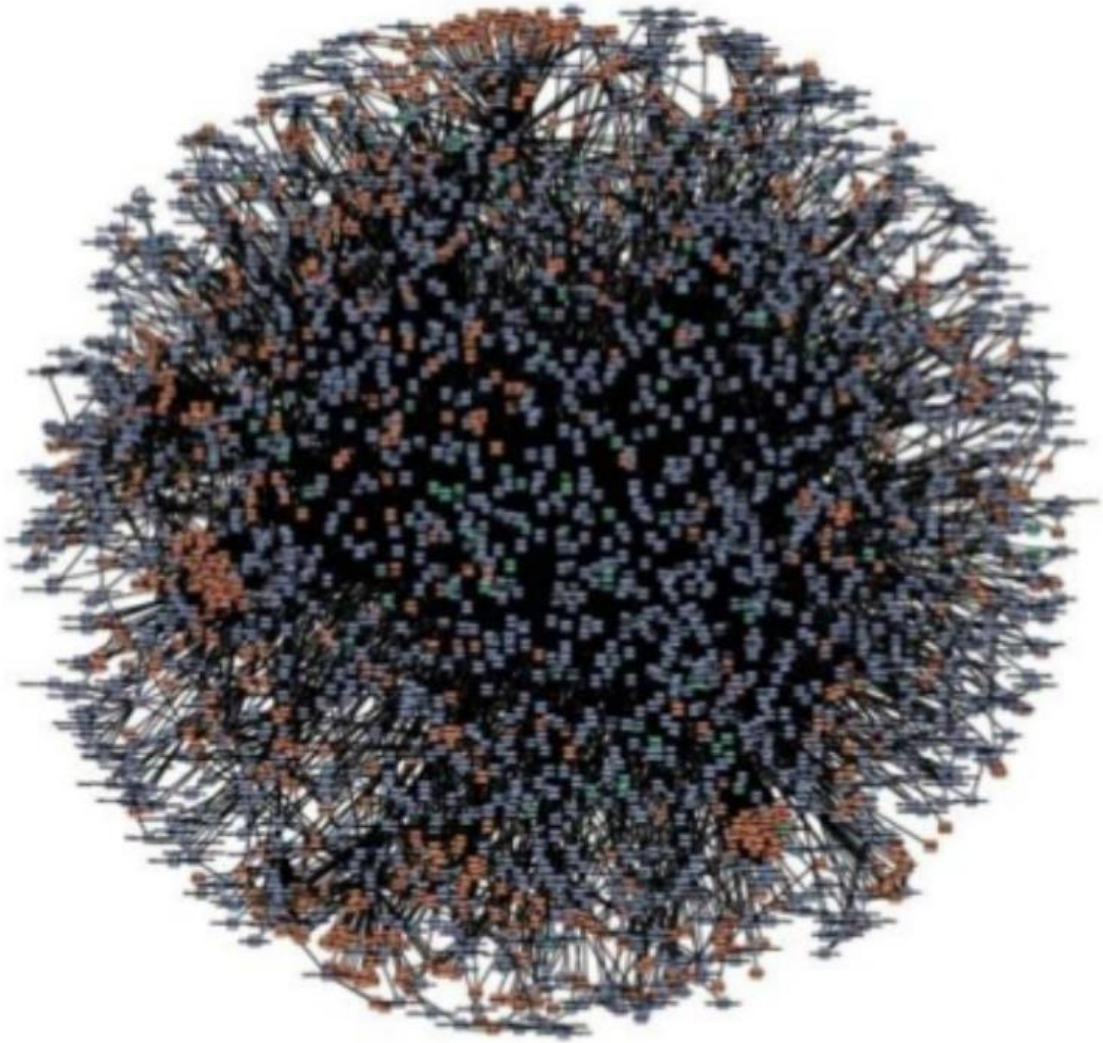


But These are True Microservice Architectures!

Twitter



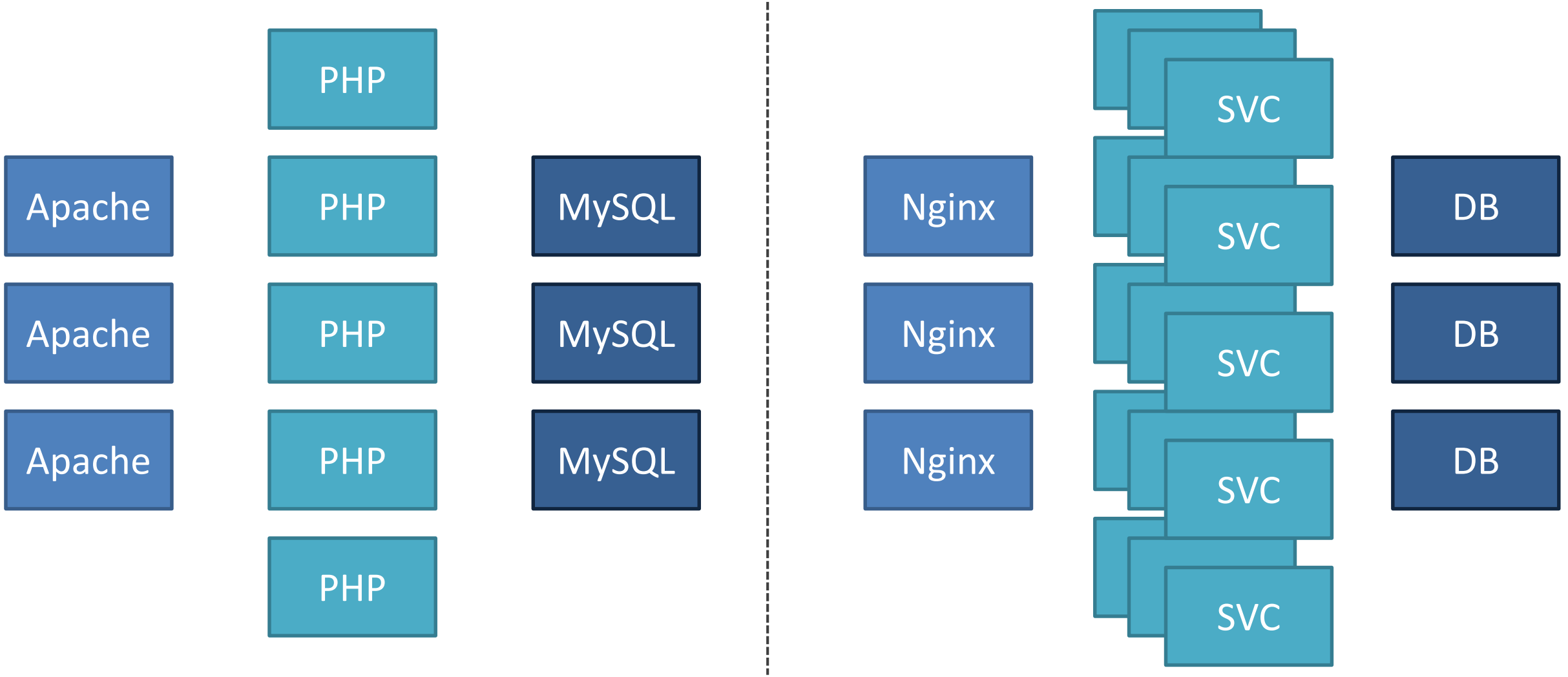
Amazon Web Service



What is a Service Mesh?

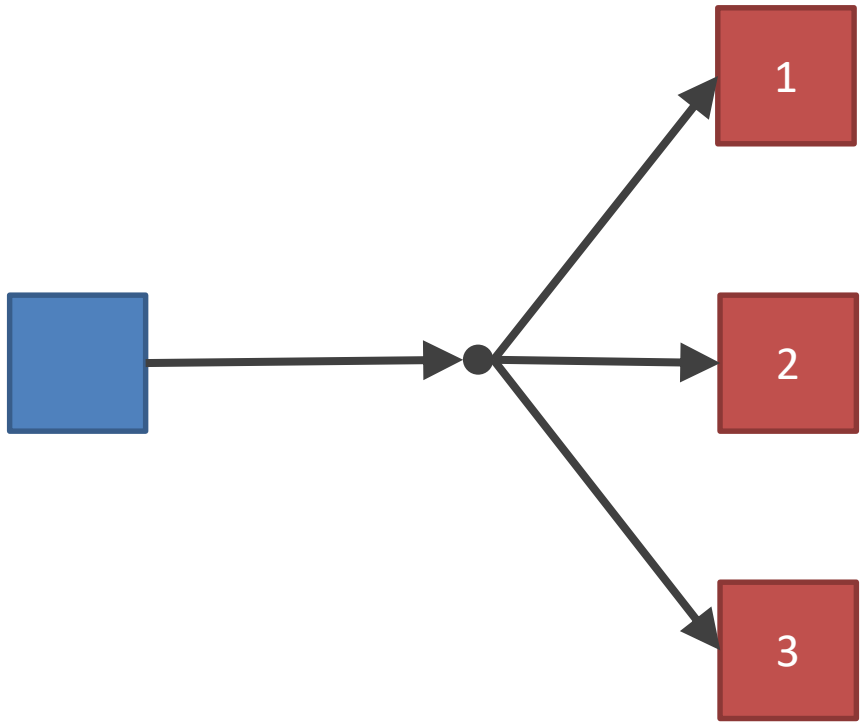
Service Mesh is **dedicated infrastructure layer**
in a **microservice environment**
to consistently **manage, monitor** and **control**
the **communication** between services across
the entire application

Evolution: LAMP to Web Scale



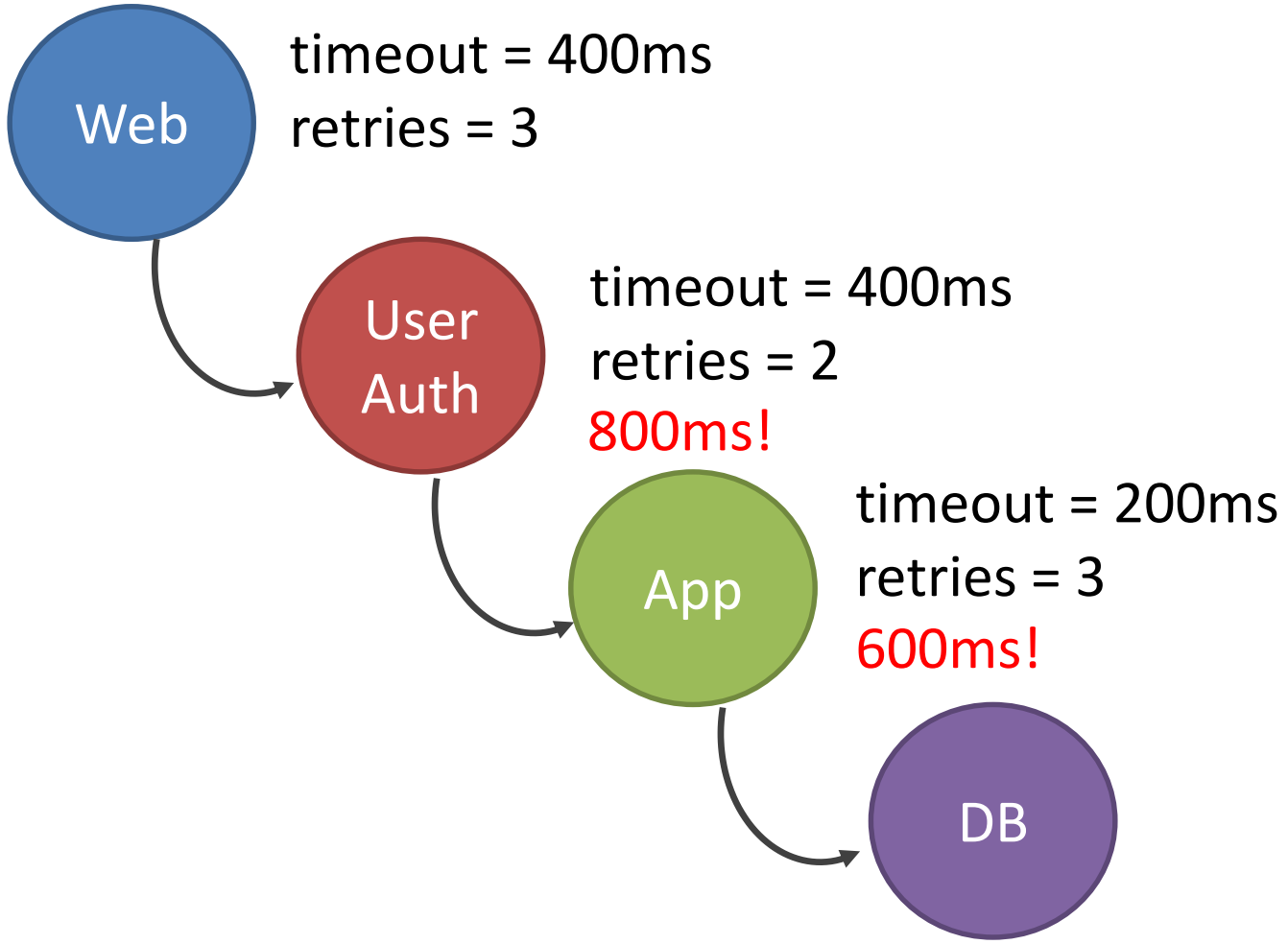
Evolution: The Need of Common Features in Service Components

- Dynamic service discovery
- Load balancing
- Health checks



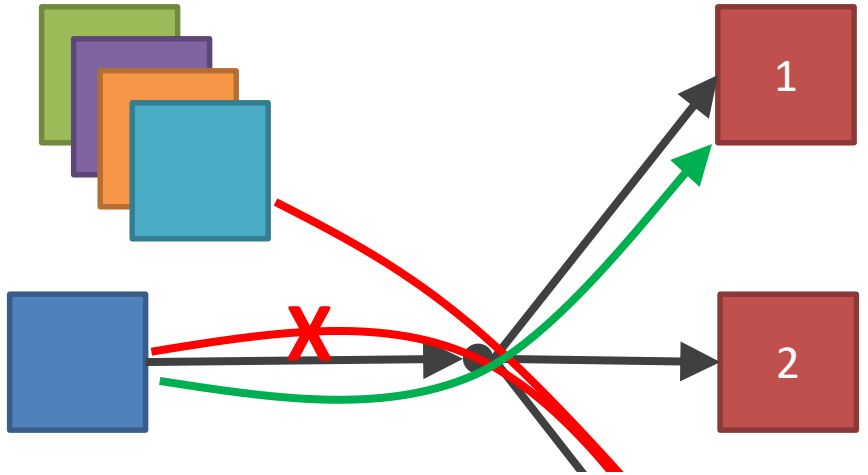
Evolution: The Need of Common Features in Service Components

- Dynamic service discovery
- Load balancing
- Health checks
- Timeouts
- Retries



Evolution: The Need of Common Features in Service Components

- Dynamic service discovery
- Load balancing
- Health checks
- Timeouts
- Retries
- Circuit breakers
- TLS termination
- HTTP/2 and gRPC proxies
- Monitoring and tracing via rich metrics
- Fault injection



Making the Netflix API More Resilient

Netflix Technology Blog Follow
Dec 8, 2011 · 7 min read

by Ben Schmaus

The API brokers catalog and subscriber metadata between internal services and Netflix applications on hundreds of device types. If any of these internal services fail there is a risk that the failure could propagate to the API and break the user experience for members.

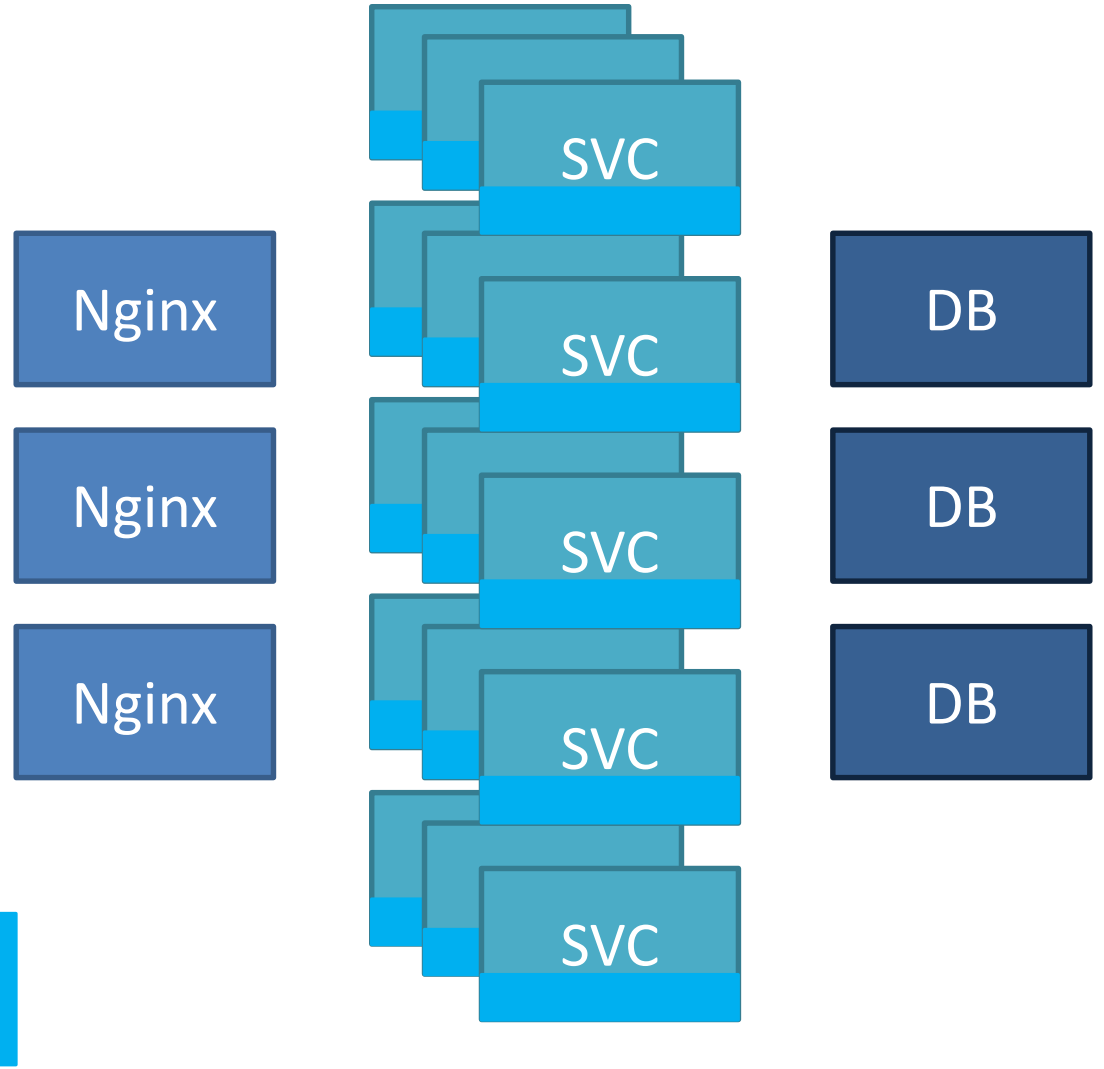
Evolution: Shared Libraries to Service Mesh

Examples for such fat libraries:

- Hystrix @ Netflix
- Stubby @ Google
- Finagle @ Twitter

Disadvantages of shared libraries:

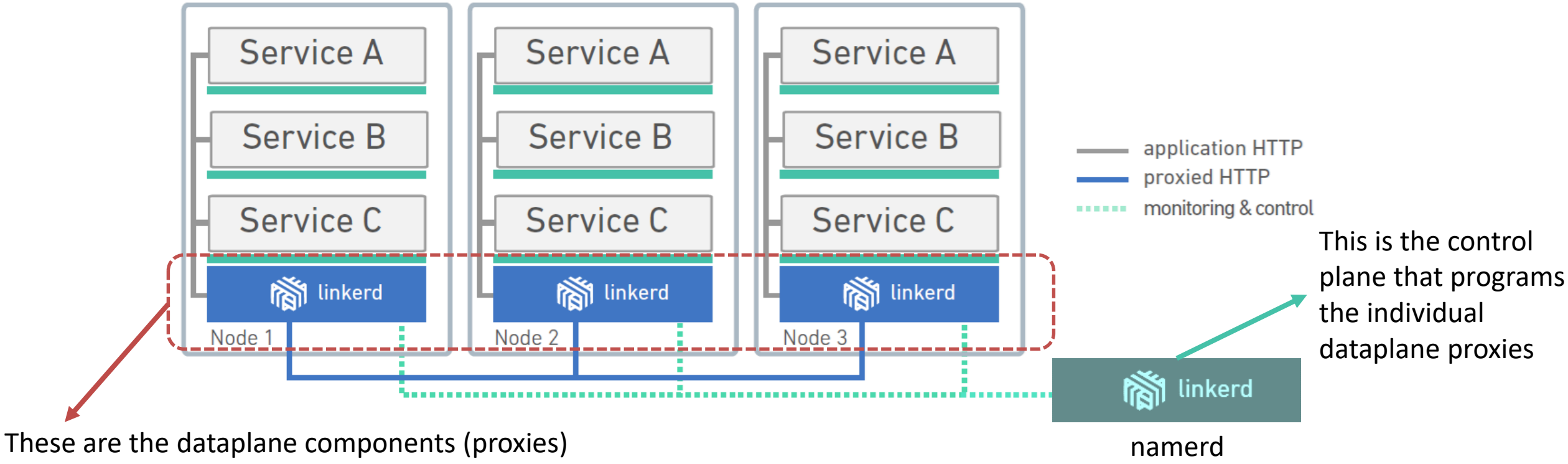
- Have to be implemented in multiple languages
- If the library changes the entire service has to be redeployed
- Too tight involvement of dev teams



Linkerd

A service mesh that adds reliability, security, and visibility to cloud native applications

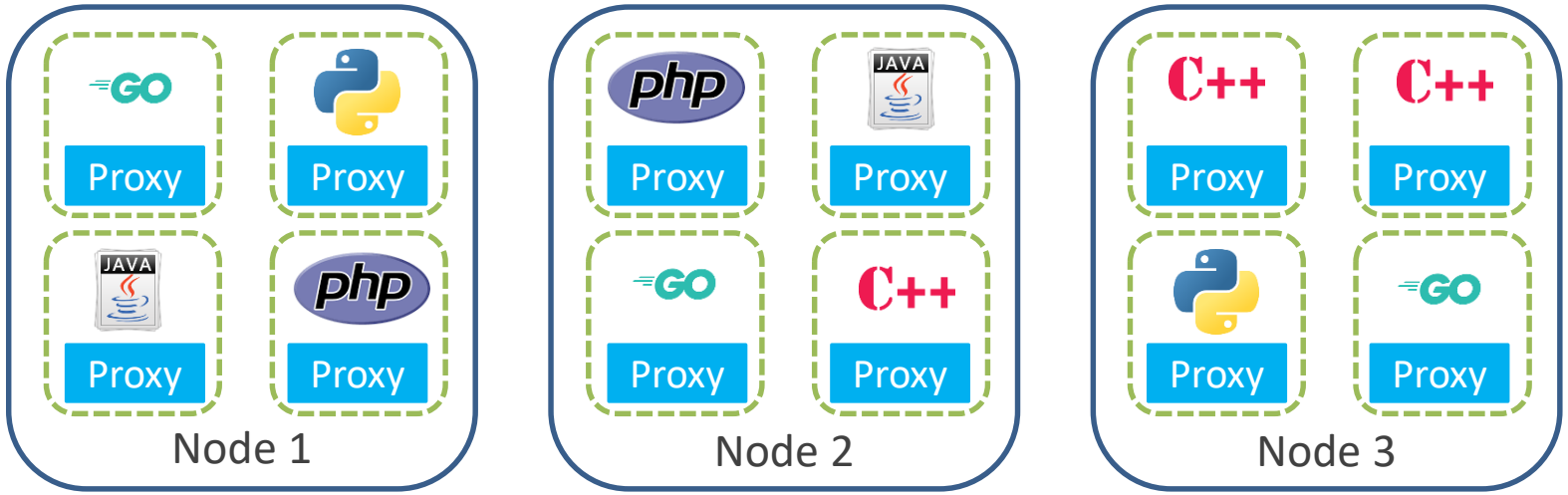
- Official CNCF Project
- Originally created by Buoyant Inc. based on Finagle
- Written in JAVA



Sidecar Model in Container Environments

Disadvantages of per-node model

- Raises security concerns in multi-tenant environments (shared TLS secrets, common authentication, etc.)
- Can only be scaled vertically, not horizontally (give it more memory and CPU and it will handle more connection)
- Not optimized for container workloads



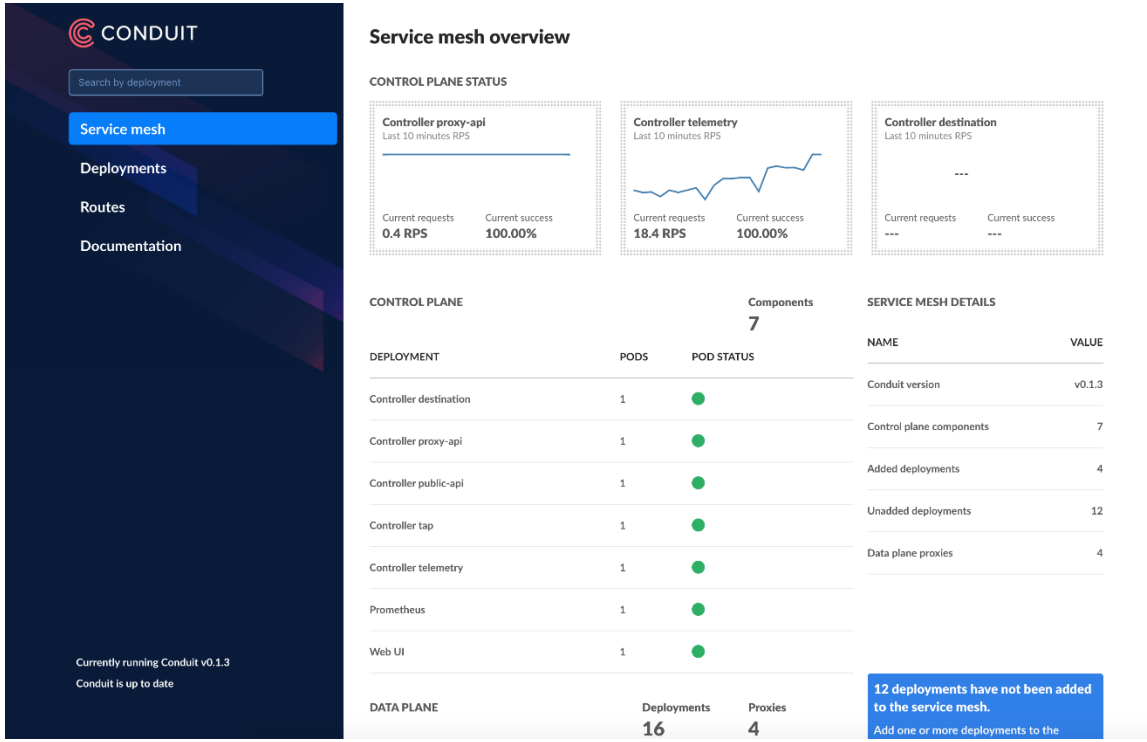
The sidecar model

- Put a proxy next to every container
- This is supported by the POD abstraction in Kubernetes
- Linkerd is considered to be too heavy for such environment!

Linkerd2 (previously know as Conduit)

A novel service mesh that was specifically designed to work in Kubernetes

- Created by Buoyant, the same team who created Linkerd
- Data plane is written in Rust to be very fast and lightweight to sidecar operations (~5MB container size)
- Control plane is written in GO to work well in Kubernetes
- Can be deployed service-by-service (it's not an all-or-nothing choice…)



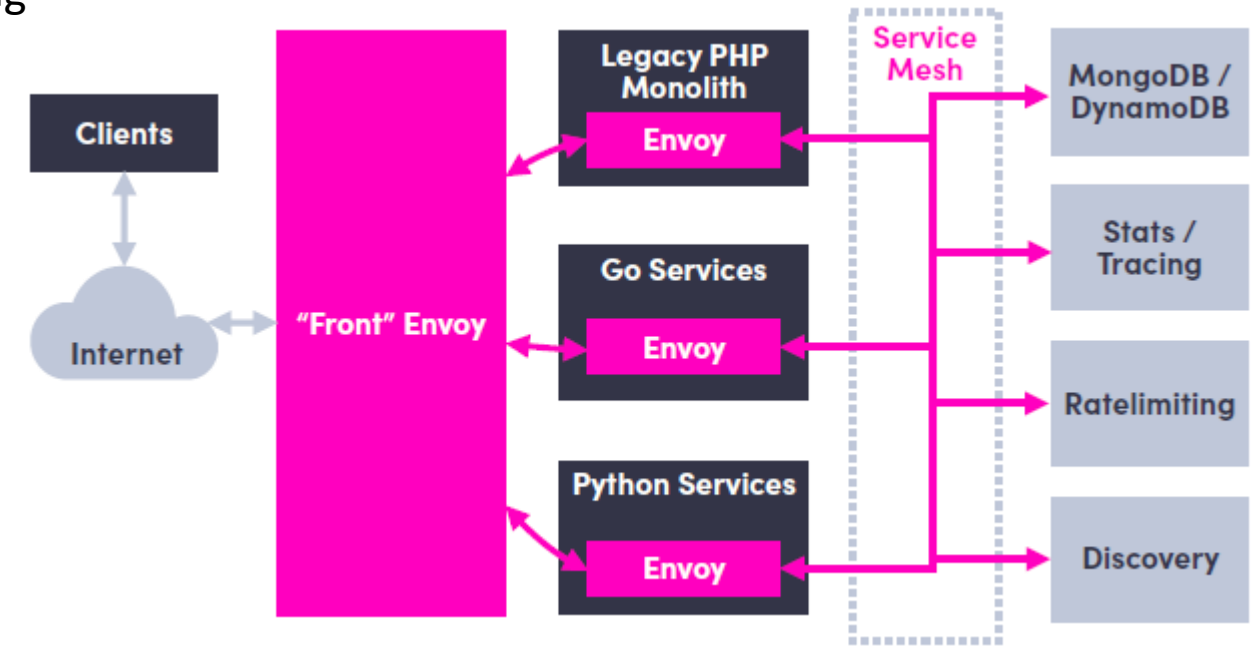
Envoy

Envoy is an open source edge and service proxy, designed for cloud-native applications

- Official CNCF Project
- Originally created by Lyft
- Written in C++
- Out of process architecture with advanced threading
- Best in class observability
- Rich APIs called xDS

Features include

- L3/L4 filter and routing architecture
- HTTP L7 filter architecture
- First class HTTP/2 support
- gRPC support
- MongoDB and DynamoDB L7 support



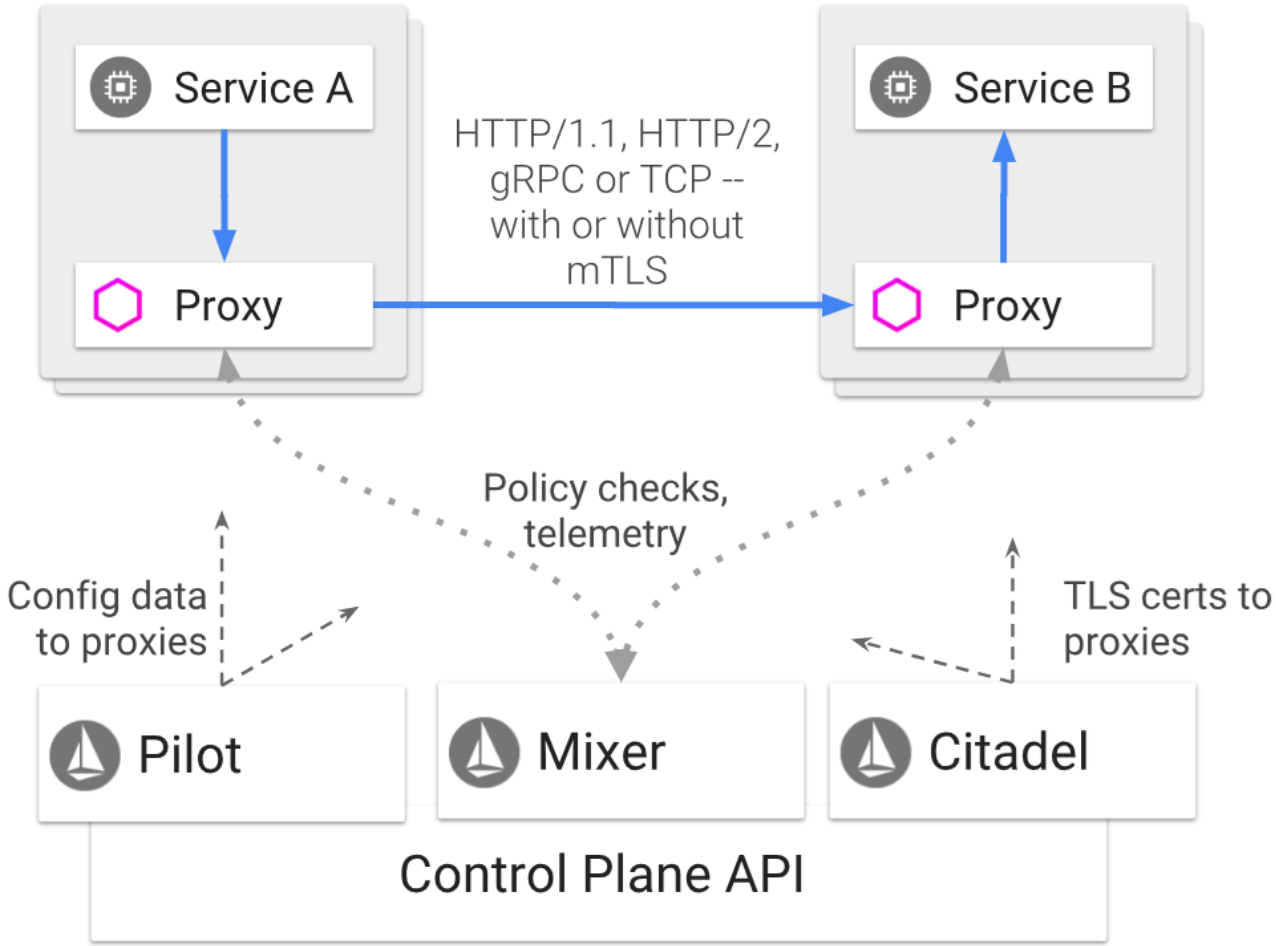
Istio

A service mesh control plane which uses Envoy as data plane

- Originally created by Google and IBM
- Written in GO
- Provides core features for:
  - Traffic management
  - Observability
  - Security

Has three main components

- Pilot for managing and configuring the Envoy proxies
- Mixer for providing policy controls and telemetry collection
- Citadel for service-to-service and end-user authentication



Other Examples

Consul by HashiCorp

- Provides a full featured control plane with service discovery, configuration, and segmentation functionality
- Has a built in proxy, but also compatible with **Envoy**, HAProxy and Nginx

App Mesh by Amazon

- Amazon's service mesh for both ECS and EKS
- Uses **Envoy** as dataplane

Vamp (startup @ Amsterdam)

- Use HAProxy

Synapse by AirBnB

- Can use HAProxy or Nginx

Finally, it’s demo time ☺