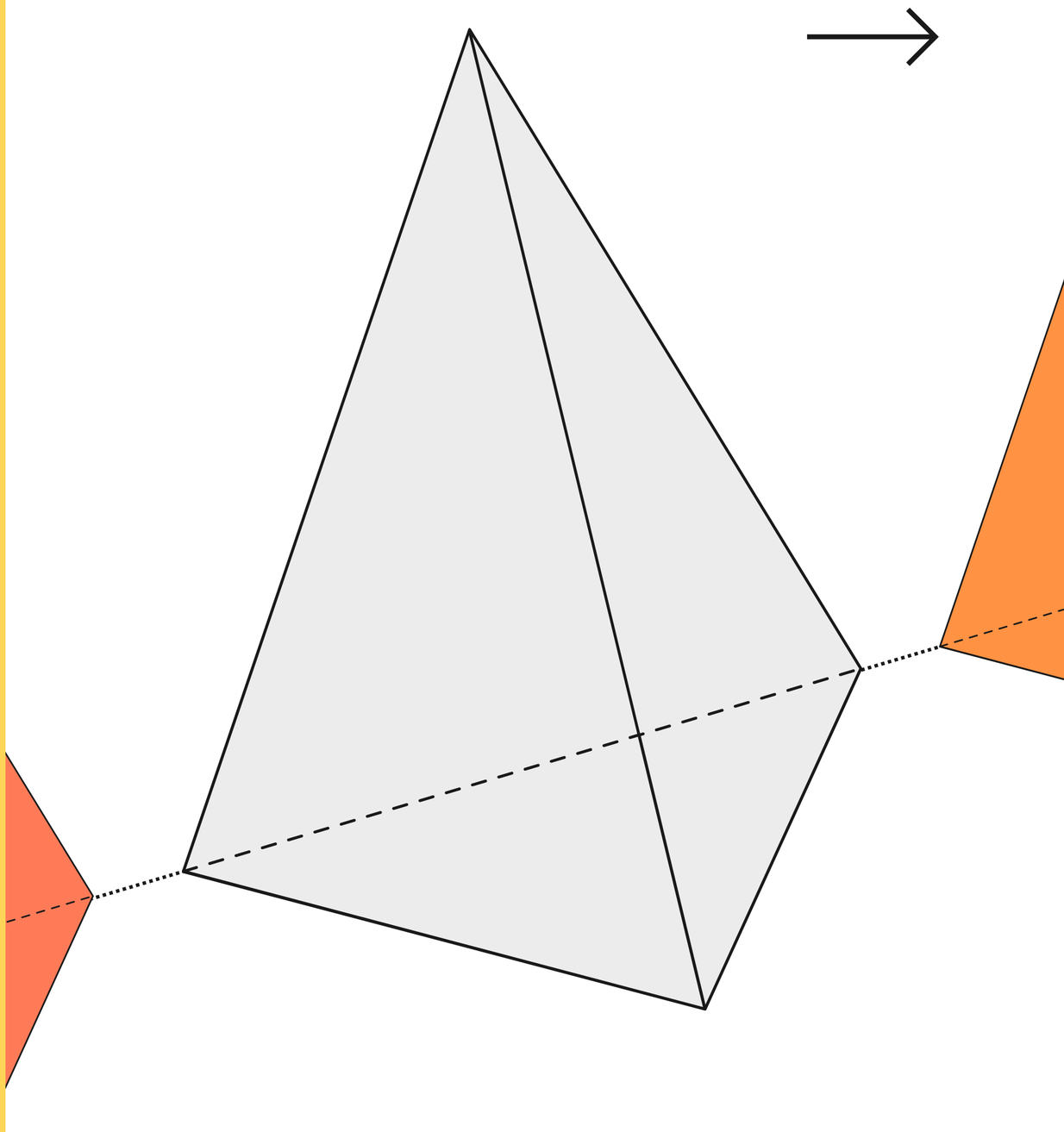
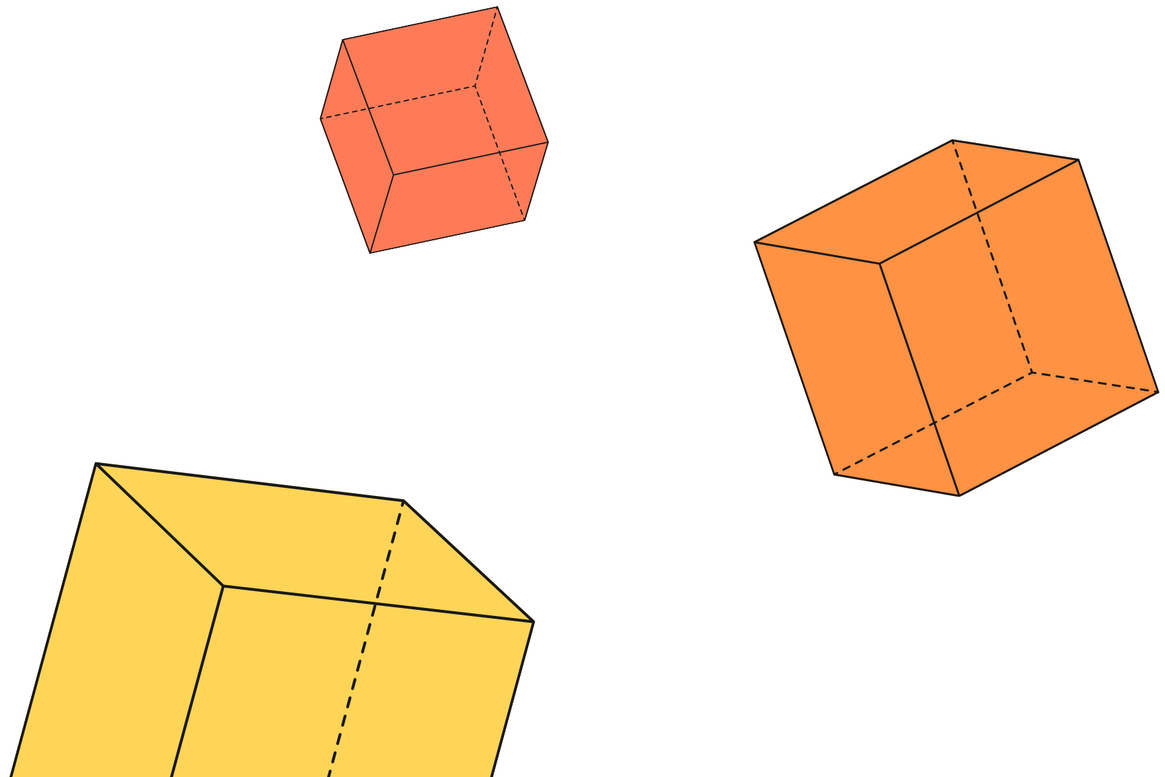
Istio-Powered Observability

NEERAJ PANDEY
Founder Vivid Climate & LyntCube

DEVOPSDAYS TW 2024



WHAT ARE WE LEARNING?



- Monoliths to Microservices
- Challenge of Orchestration
- Service Mesh
- Why and uses of Service Mesh
- Istio Service Mesh
- Istio and Observability Tools
- Implementation - Jaeger, Kiali, Grafana

INTRODUCTION →

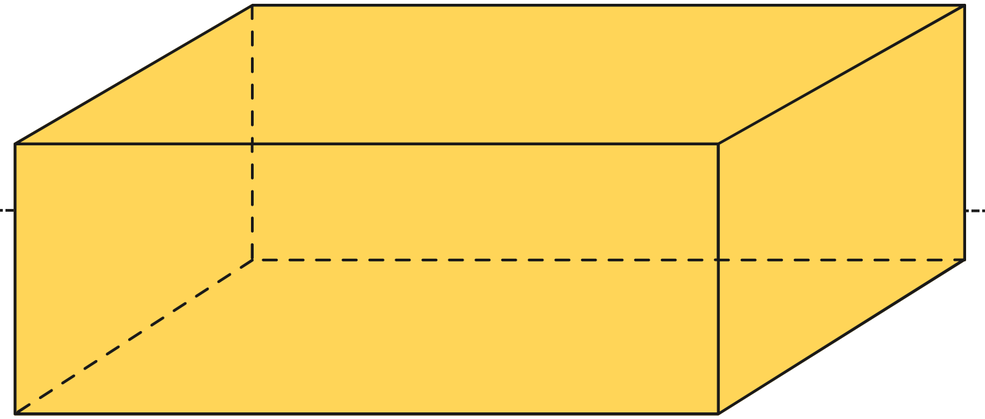
NEERAJ PANDEY

https://n4jp4y.com

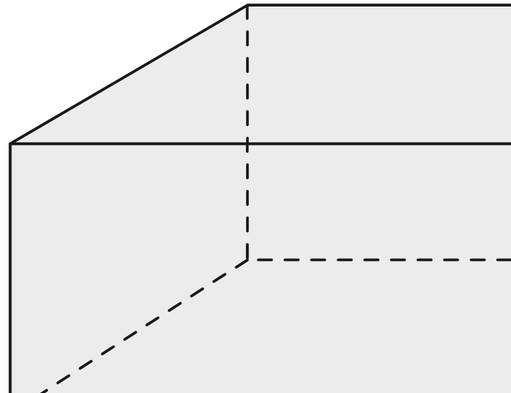
n4jpy4 @twittter @linkedin

Vivid Climate - LyntCube Studio - Thrax Algo

MONOLITHIC ERA



- Single codebase (Simplicity in Early Stages)
- Tight coupling of components
- Challenges with scalability and maintenance.



Challenges of Monolithics

SINGLE POINT OF FAILURE

Single issues can bring down the entire monolithic application, reducing resilience.

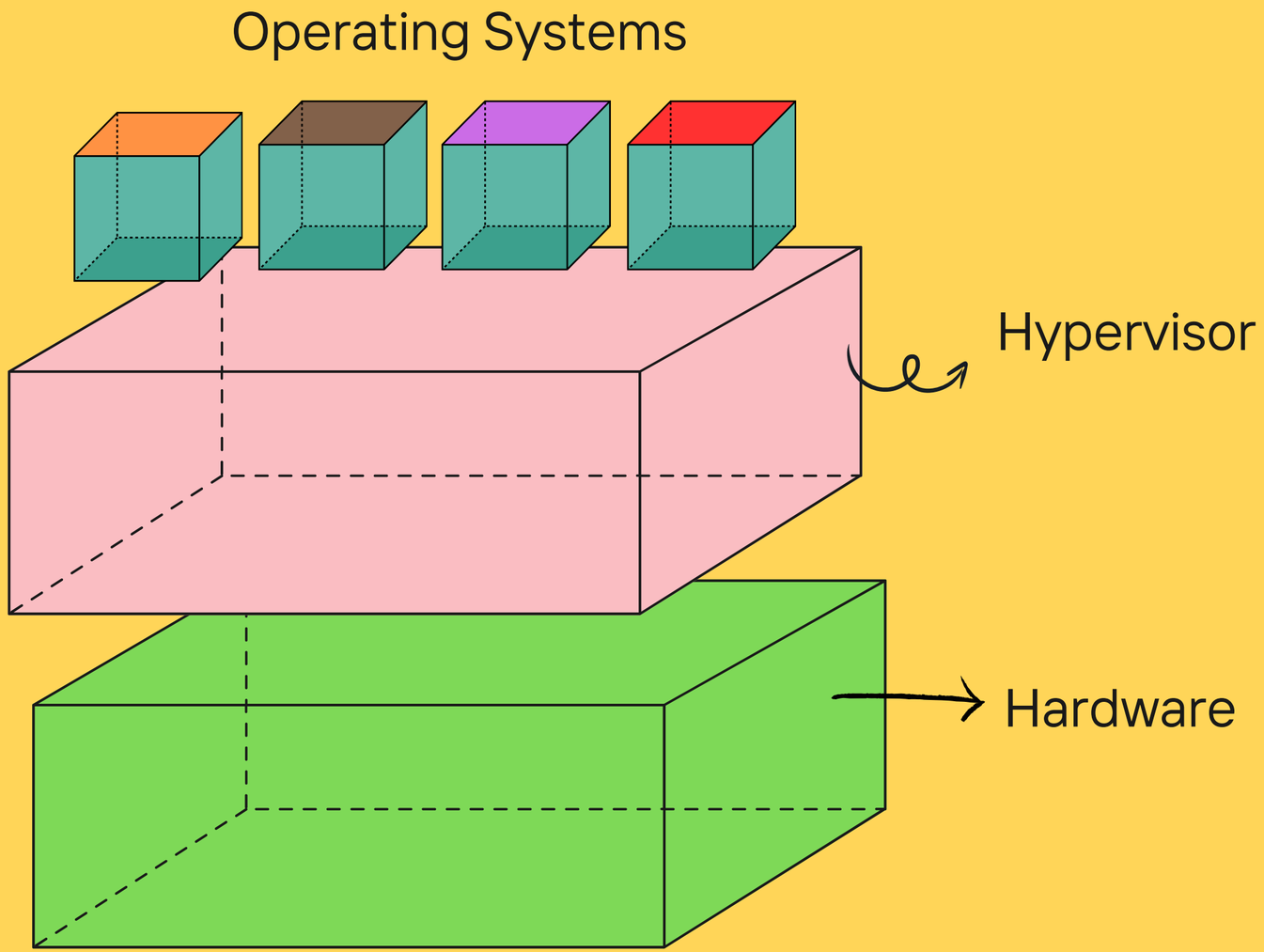
DEPLOYMENT DILEMMA

Updating monoliths requires system-wide downtime, affecting user experience.

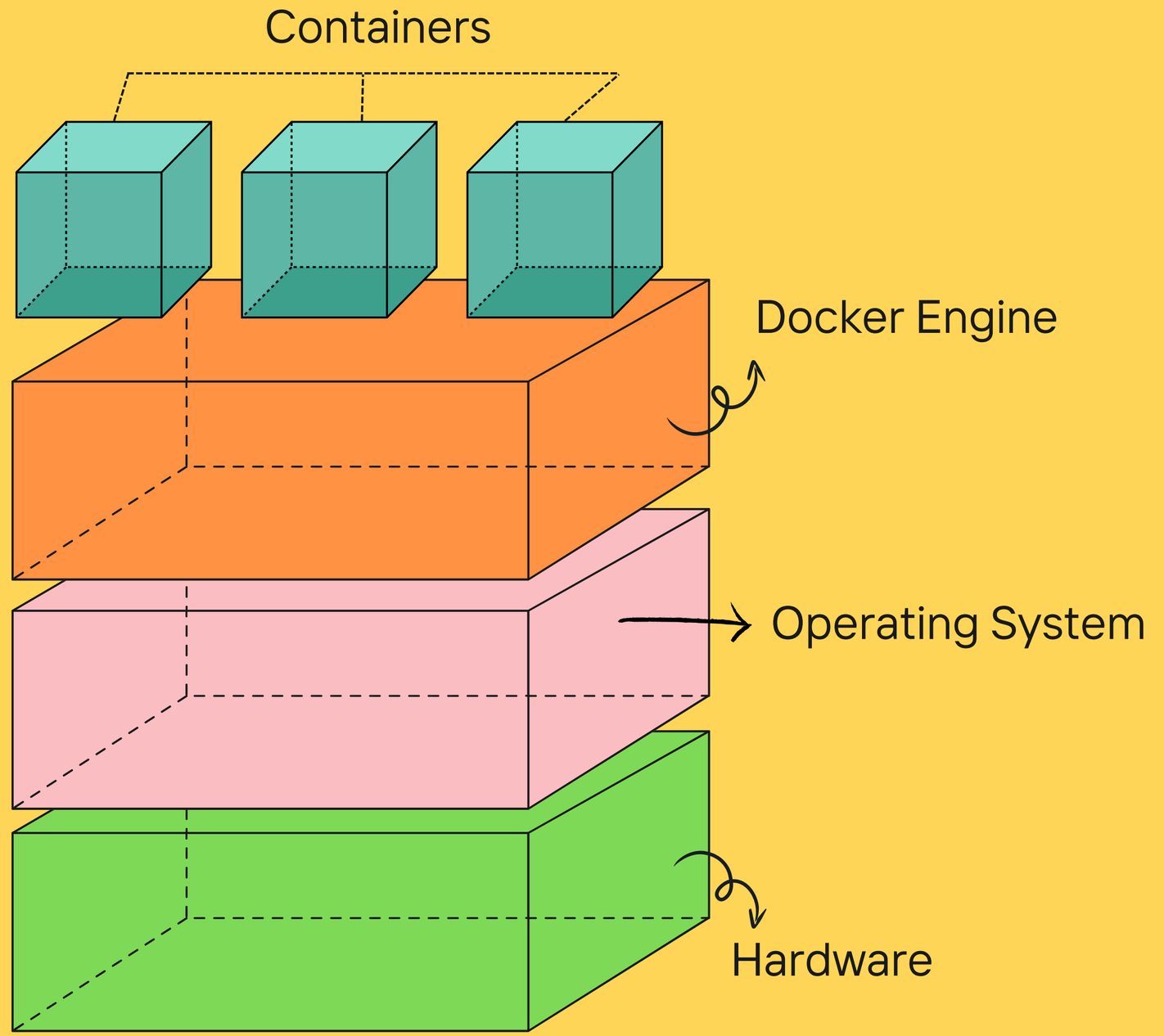
SCALING STRUGGLES

Scaling monoliths is resource-intensive; can't scale individual components.

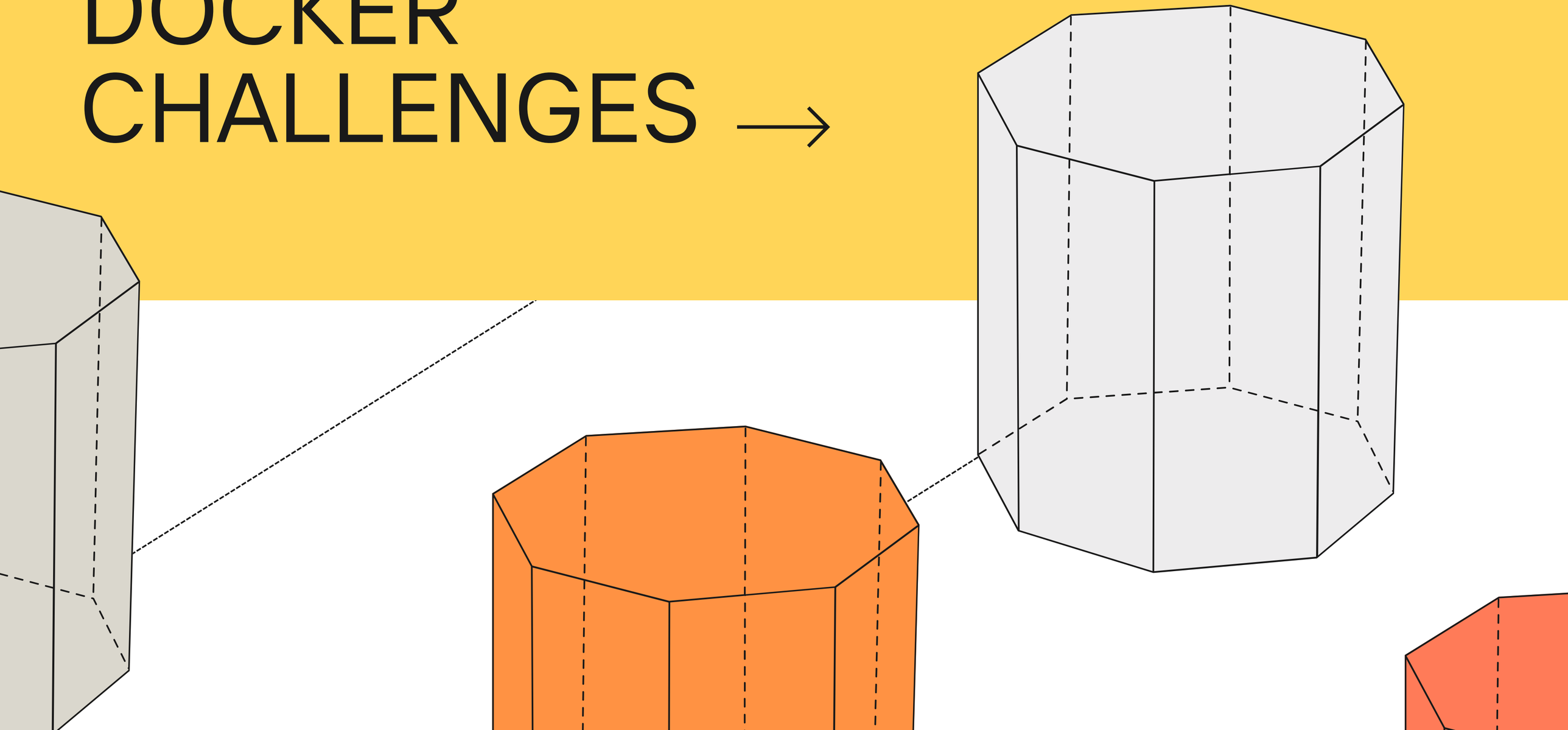
VIRTUAL MACHINES



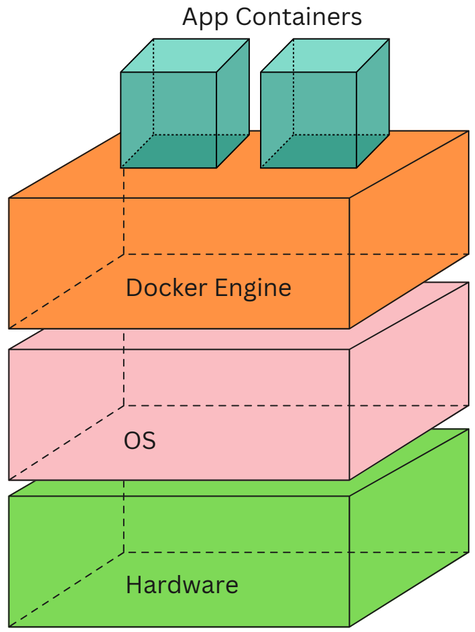
DOCKER



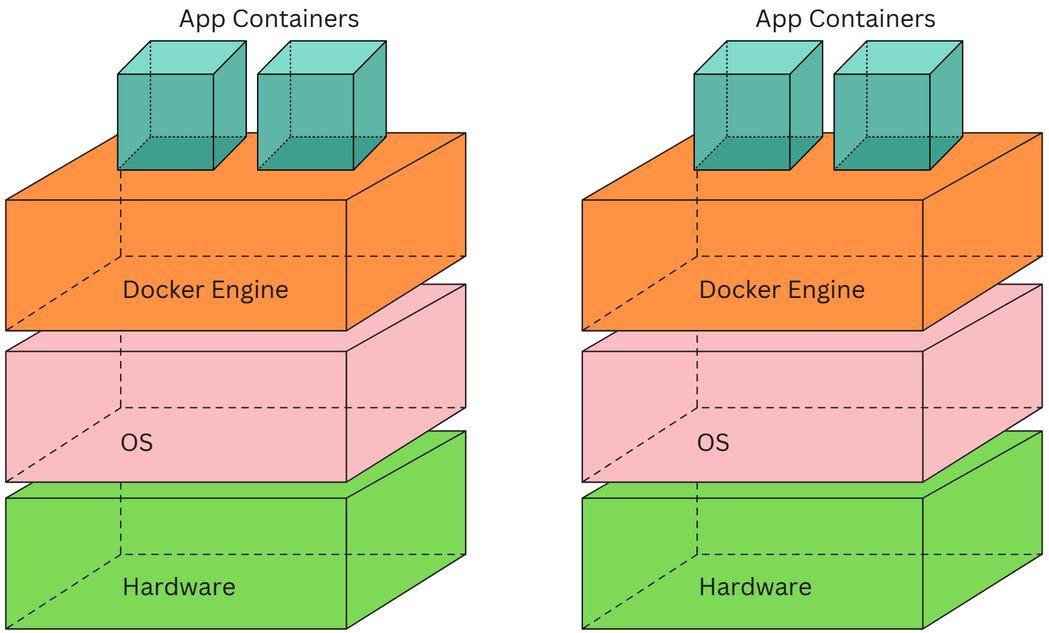
DOCKER CHALLENGES →



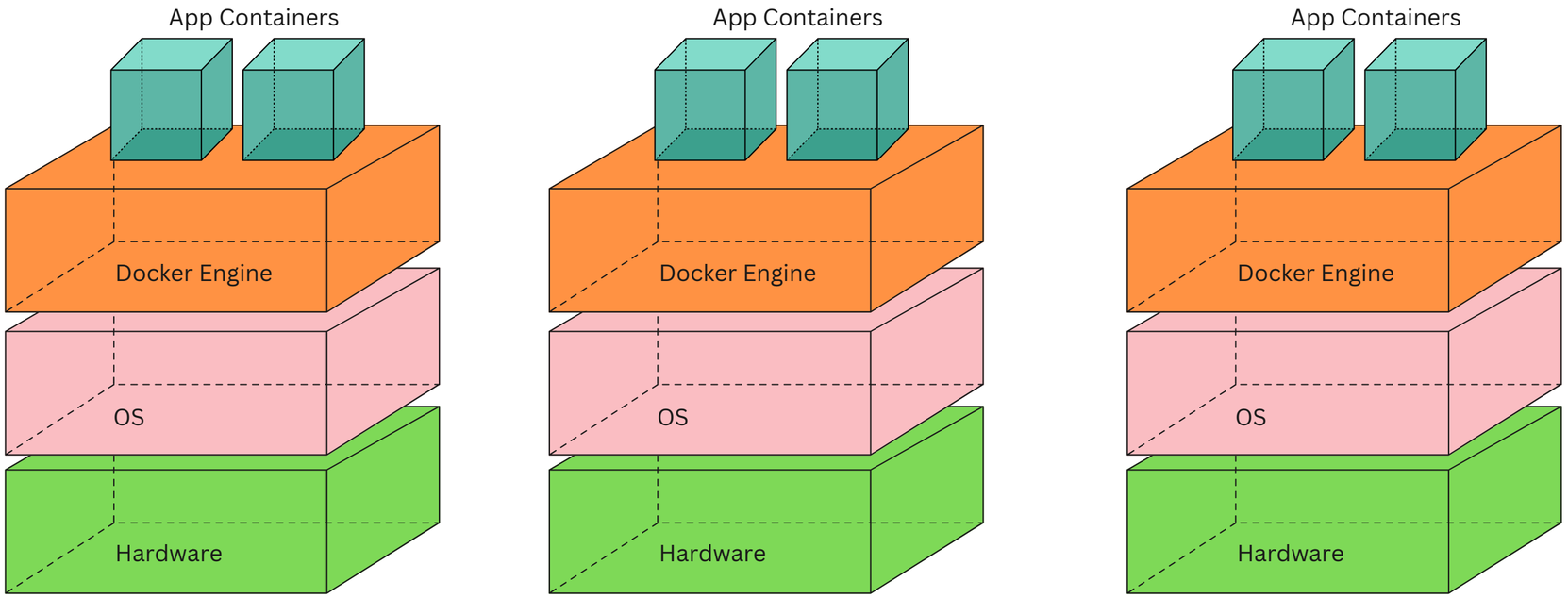
Docker Challanges →



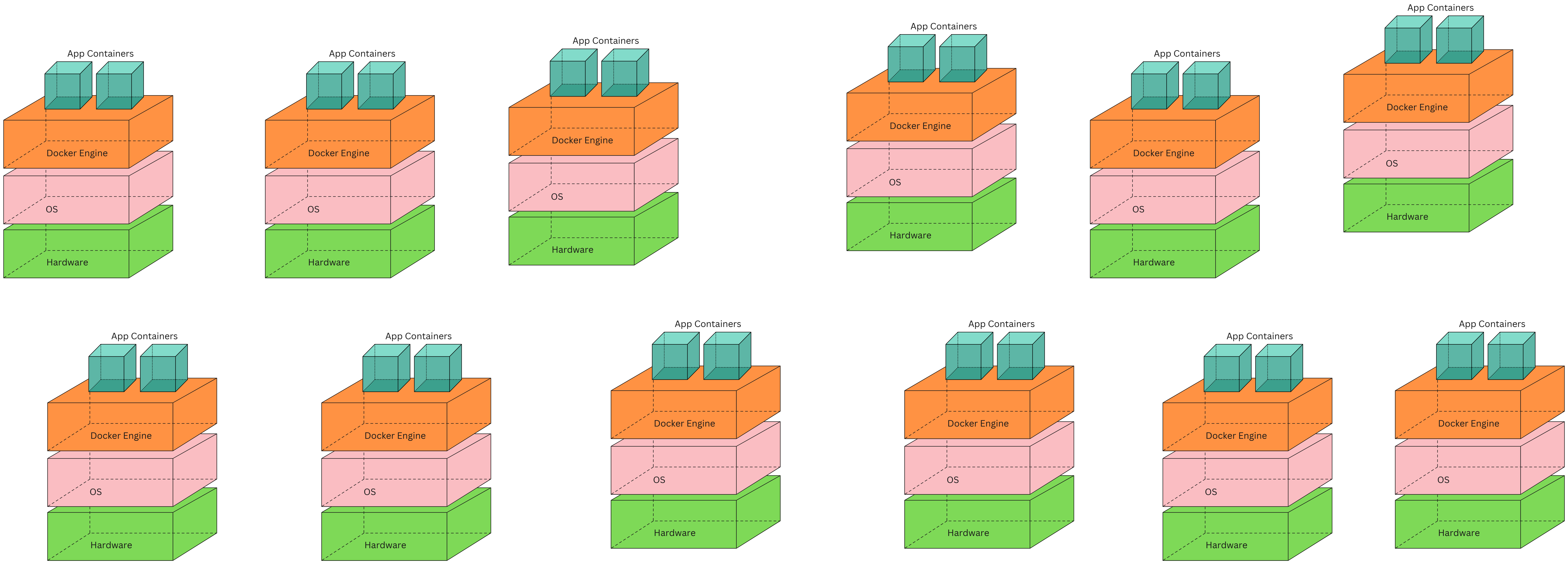
Docker Challanges →



Docker Challanges →



Docker Challanges →



Docker Challenges

As the number of containers and machines increases, managing Docker environments becomes increasingly complex.

Manual processes for scaling, updating, and configuration lead to inefficiencies and potential inconsistencies, highlighting the need for better orchestration solutions.

Managing Multiple Containers

Scaling and resource allocation become complex as the number of containers grows.

Manual Scaling

Adding new machines and containers manually is time-consuming and inefficient.

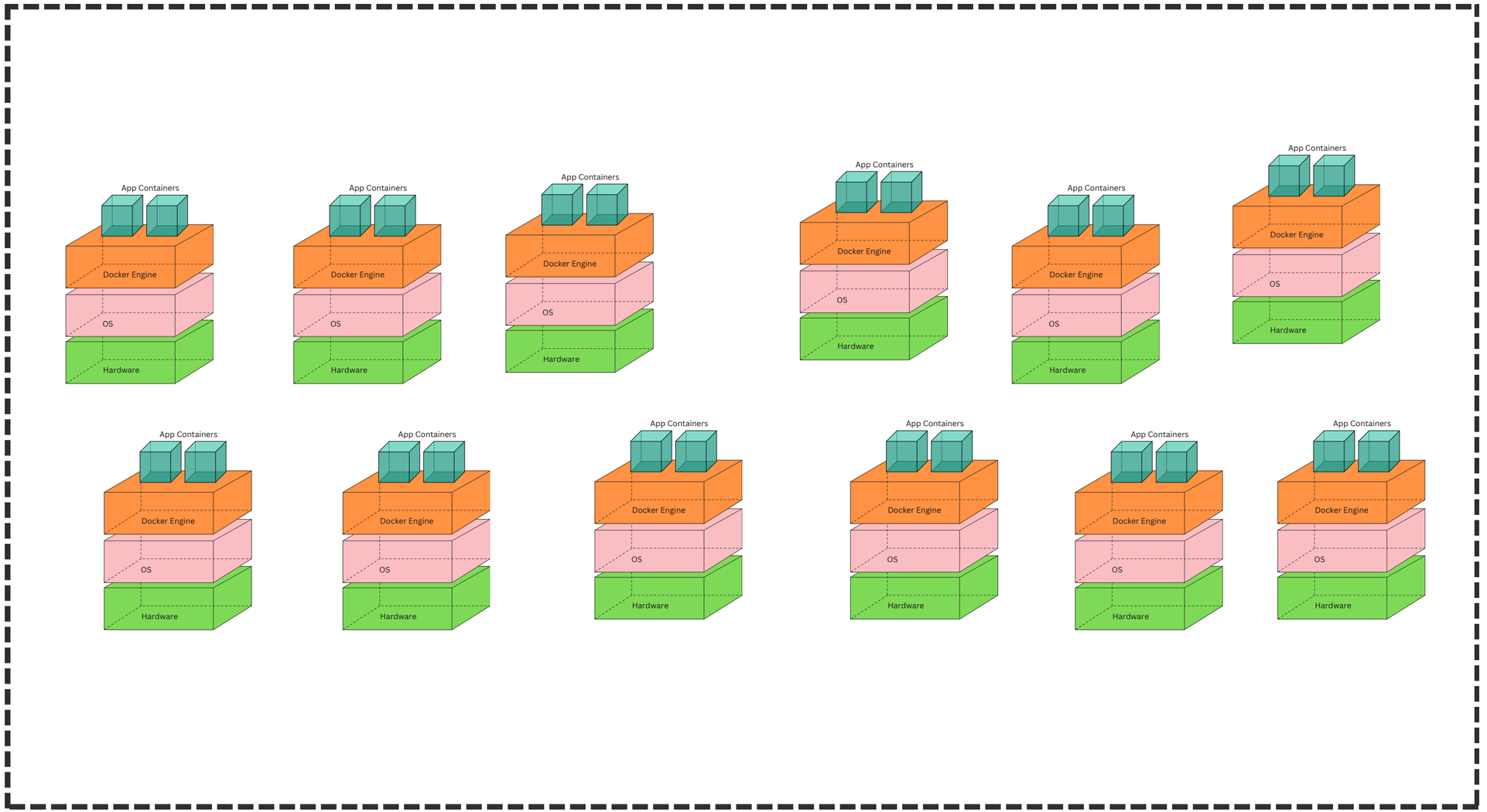
Update and Deployment Issues

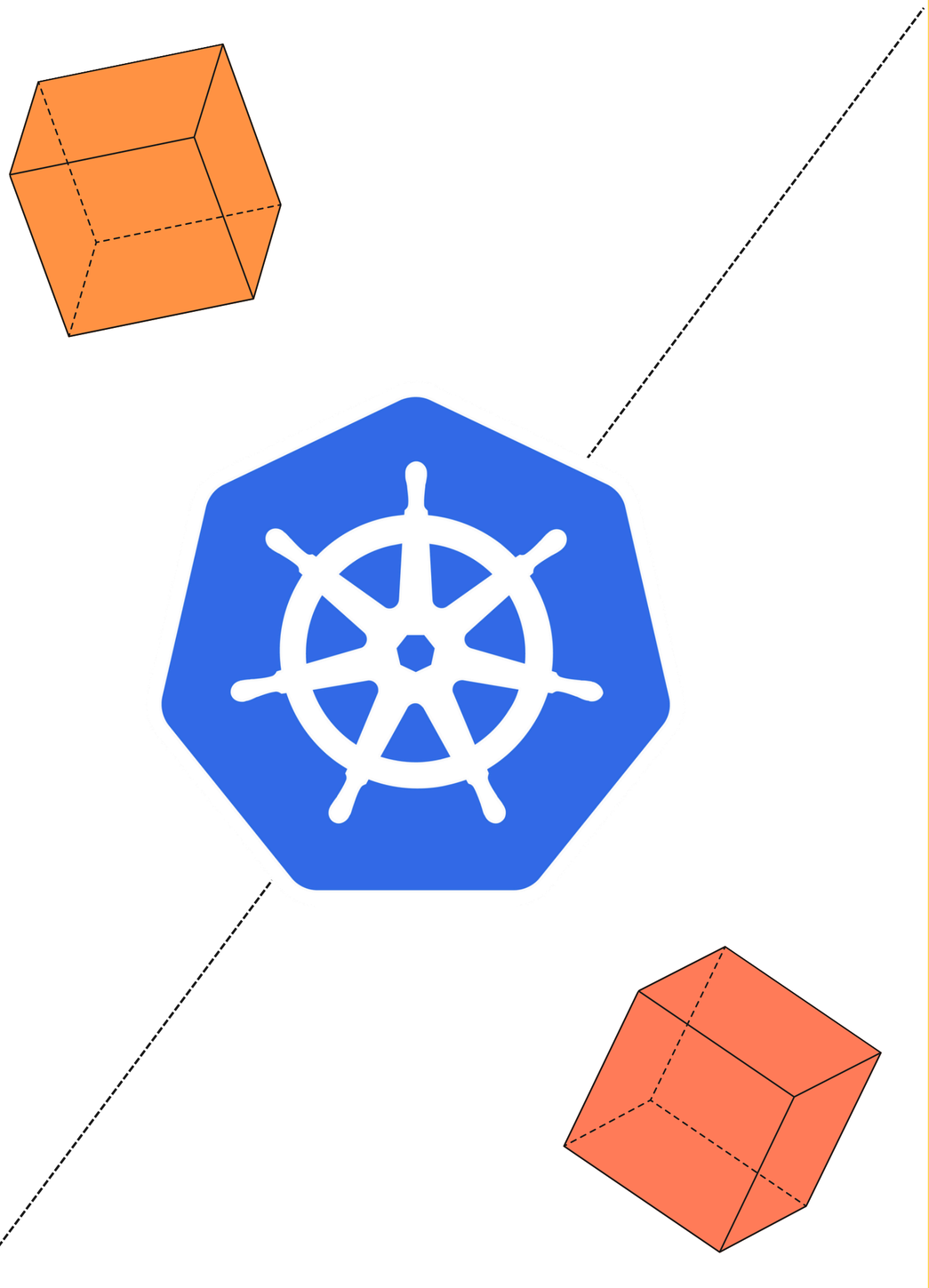
Updates require manual changes on each machine, leading to potential inconsistencies.

Configuration Management

Ensuring consistent configurations across all resources is error-prone and difficult.

Orchestrating the chaos →





Kubernetes

Container Orchestration

What is Kubernetes?

Automated Deployments

Kubernetes automates the process of deploying containerized applications across a cluster of machines.

Developers can define the desired state of their application, and Kubernetes takes care of the deployment process.

Self-Healing Capabilities

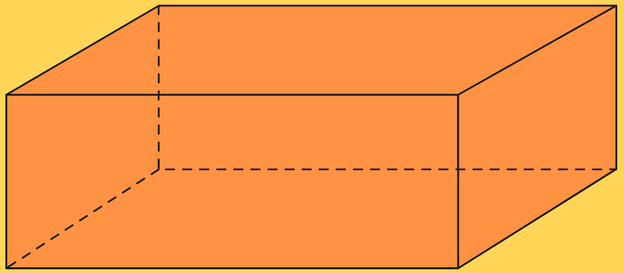
Kubernetes can automatically restart failed containers and reschedule them on healthy nodes (machines) within the cluster, ensuring high availability of your application.

Service Discovery

Kubernetes helps applications discover and communicate with each other within the cluster.

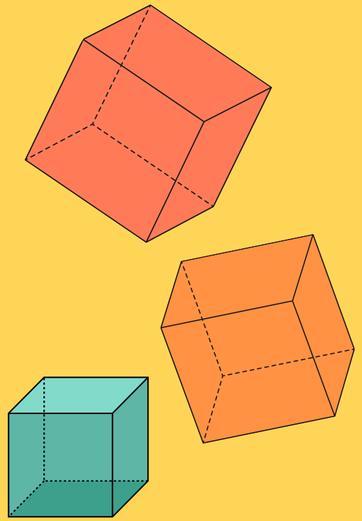
It also provides built-in load balancing to distribute traffic across multiple container instances of a microservice.

ARCHITECTURE



MASTER NODE

- Central control unit managing the cluster.
- Controls cluster , Manages resources



WORKER NODES

- Nodes that run application workloads.
- Runs applications , Manages Pods

Worker Node

Kubelet (Agent ensuring containers are running in a Pod)

Manages containerized applications on the node, ensuring they're running as expected.

Kube-Proxy (Handles network proxying and load balancing.)

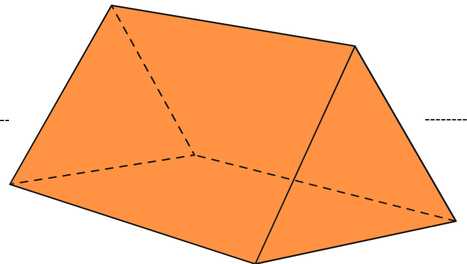
Manages network rules and traffic routing for services within the node.

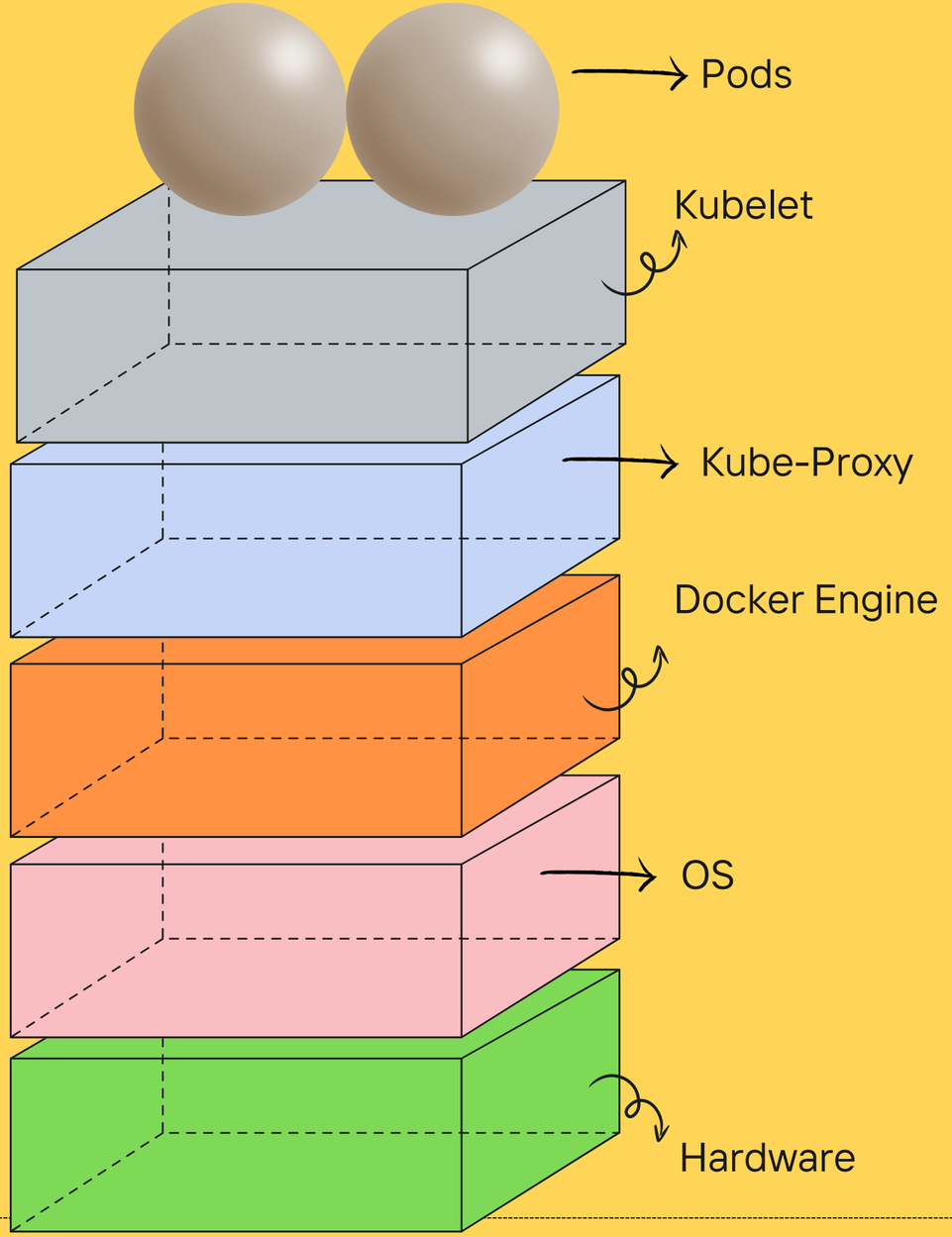
Container Runtime (Software for running containers.)

Executes and manages container life cycles, such as Docker or containerd.

Pods (Smallest deployable units in Kubernetes.)

Encapsulates one or more containers, sharing storage and network resources.

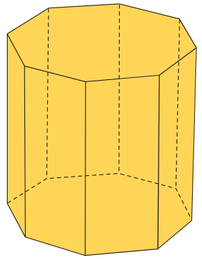




Master Node

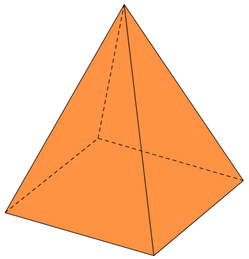
The master node is the central control unit of a Kubernetes cluster.

It manages the cluster's state, schedules workloads, and ensures the desired state matches the actual state through various components.



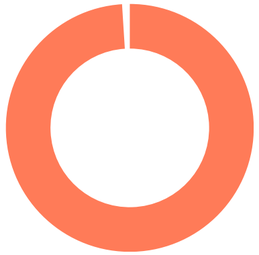
API Server (Frontend to the Kubernetes control plane)

Updates require manual changes on each machine, leading to potential inconsistencies.



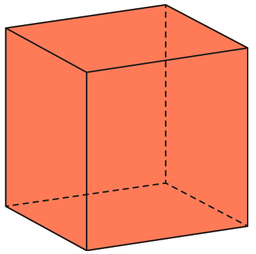
Scheduler (Assigns workloads to specific nodes)

Determines which nodes are suitable for scheduling new Pods based on resource availability.



Controller Manager (Ensures desired state of the cluster)

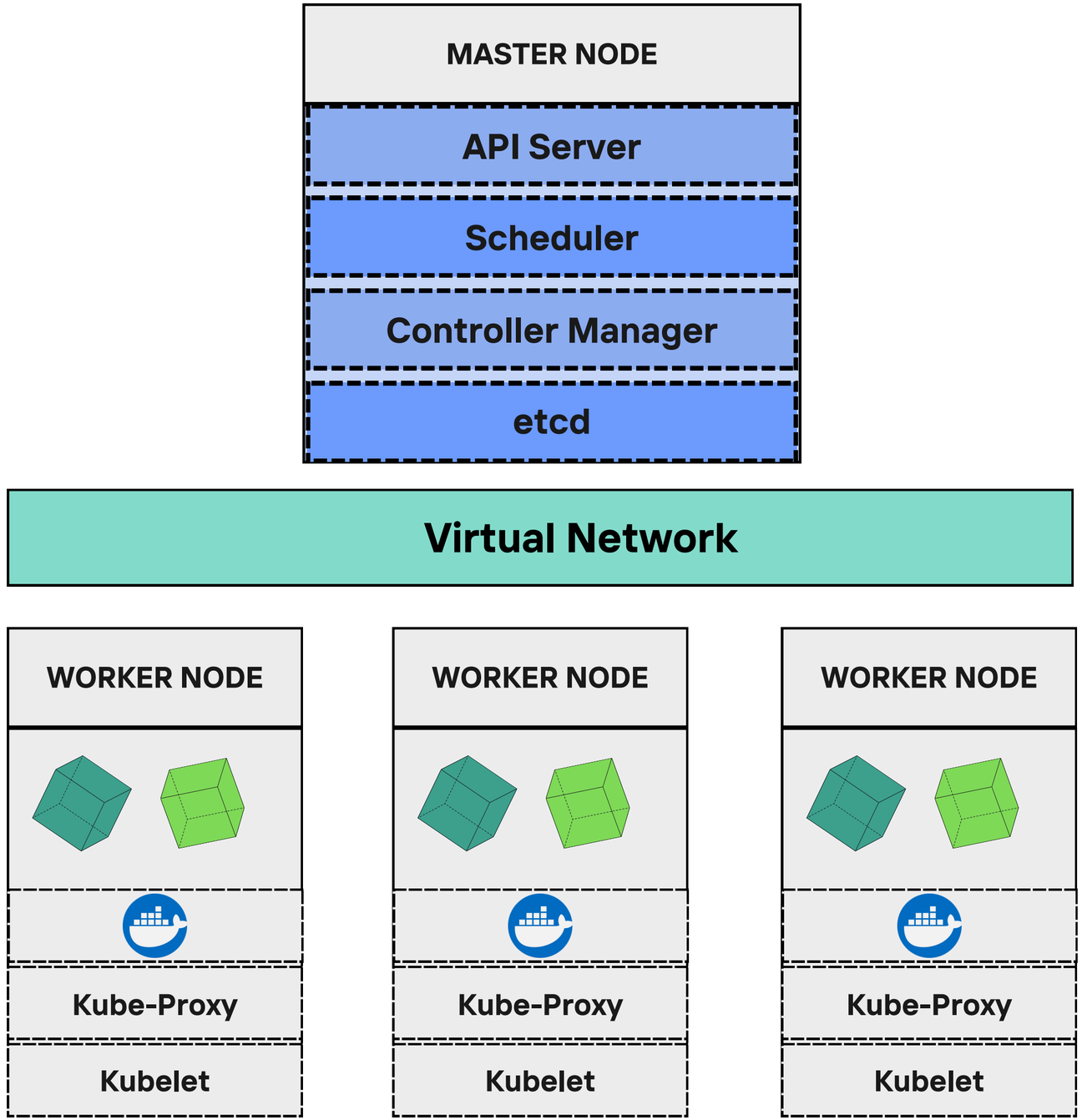
Runs various controllers that handle routine tasks, such as replication and endpoint management.



etcd: (Key-value store for all cluster data.)

Stores configuration data, state, and metadata for the Kubernetes cluster.

K8
ARCHITECTURTE



Kubernetes Benefits

Kubernetes offers numerous benefits that make it a powerful and essential tool for managing containerized applications in production environments.

Efficient Resource Utilization

Kubernetes schedules containers based on resource requirements and available capacity, ensuring efficient use of resources.

Automated Deployment and Scaling

Kubernetes automates the deployment and scaling of applications, allowing for continuous delivery and integration.

High Availability and Resiliency

Kubernetes provides self-healing mechanisms, such as auto-restarting failed containers and rescheduling them on healthy nodes.

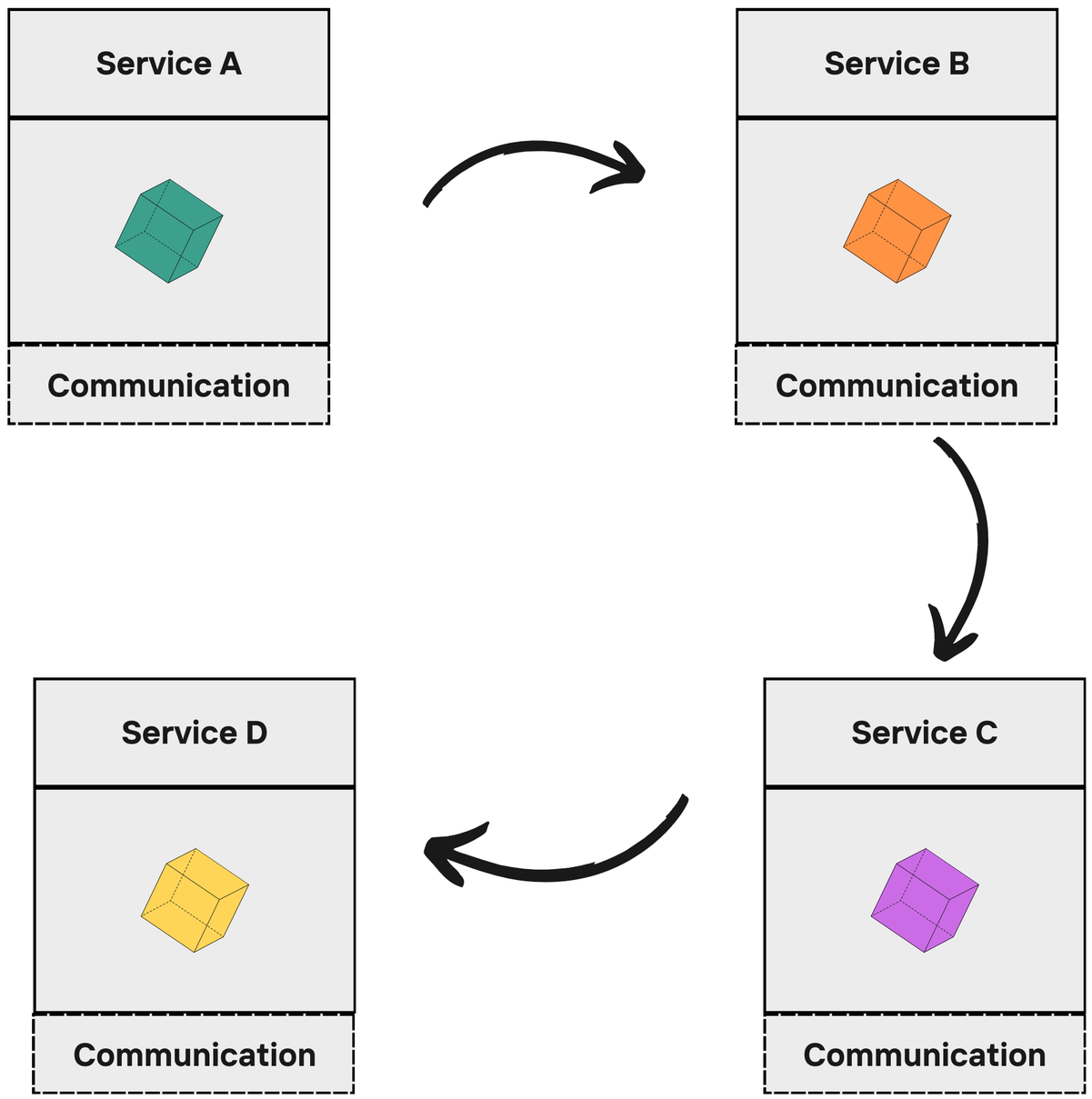
Portability and Flexibility

Kubernetes is platform-agnostic and can run on various environments, including on-premises, cloud, and hybrid setups.

Limitations of Kubernetes

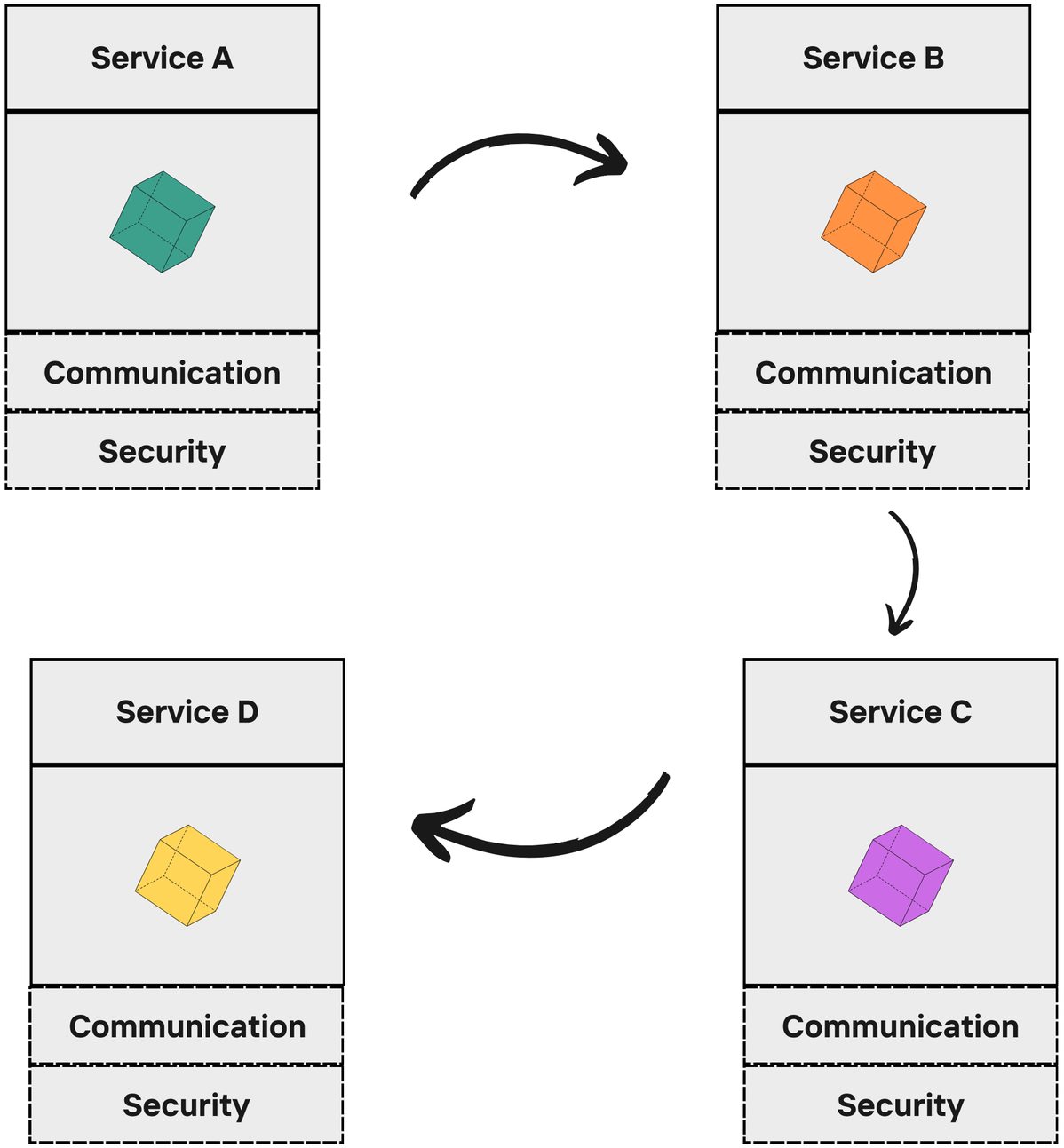
Communication Logic

- **Service Discovery**: Automatically detecting and connecting services.
- **Retry Logic**: Implementing retry mechanisms for failed requests.
- **Load Balancing**: Distributing network traffic across multiple services.



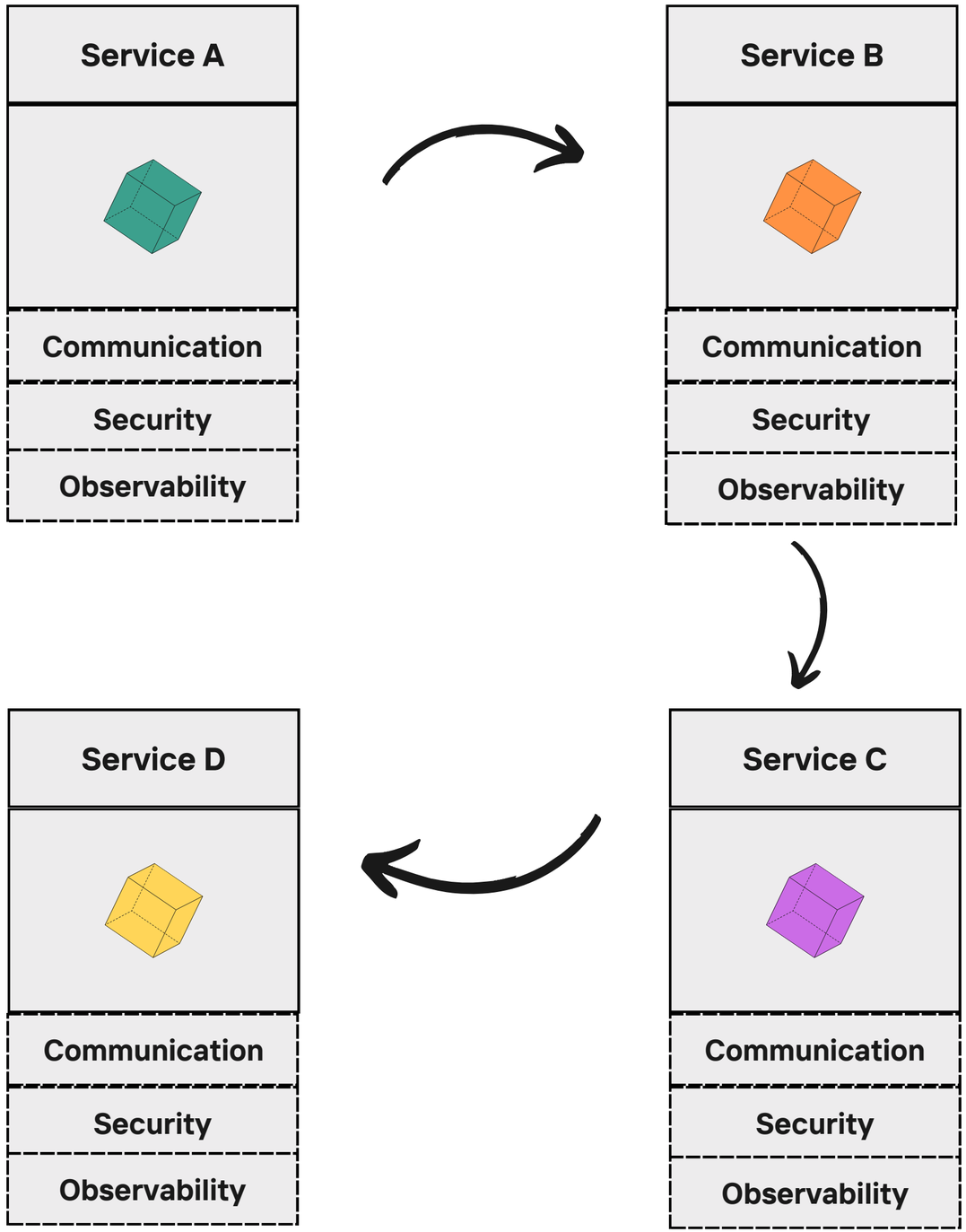
Security Concerns

- **Authorization**: Ensuring only authorized services can access specific resources.
- **Authentication**: Verifying the identity of services communicating with each other.
- **Encryption**: Securing data in transit between services to prevent interception.



Limited Observability

- **Health Monitoring:** Tracking the status and health of individual services.
- **Performance Metrics:** Measuring response times, resource usage, and throughput.
- **Communication Patterns:** Visualizing and tracing interactions between microservices.



DIY Service Management Challenges

Building your own solutions to manage inter-service communication, observability, and security can lead to increased complexity and maintenance overhead.

Each new service must incorporate logic for retries, load balancing, monitoring, and security, which can be time-consuming and error-prone.

Increased Complexity: Each service requires custom logic for critical functions.

Adding retry mechanisms, load balancing, health monitoring, and security policies to each service increases complexity and potential for errors.

Maintenance Overhead: Ongoing updates and troubleshooting are more challenging.

Keeping custom implementations up to date with best practices and security standards is labor-intensive.

Scalability Issues: Custom solutions can hinder scalability.

Managing these functions manually can limit the ability to scale services efficiently.

Introducing Service Mesh

A Communication Layer for Microservices

Introducing Service Mesh

A **service mesh** is a dedicated infrastructure layer that manages service-to-service communication within a microservices architecture, providing

Service Discovery and Routing

The service mesh helps microservices discover each other and efficiently route traffic to the appropriate service instances.

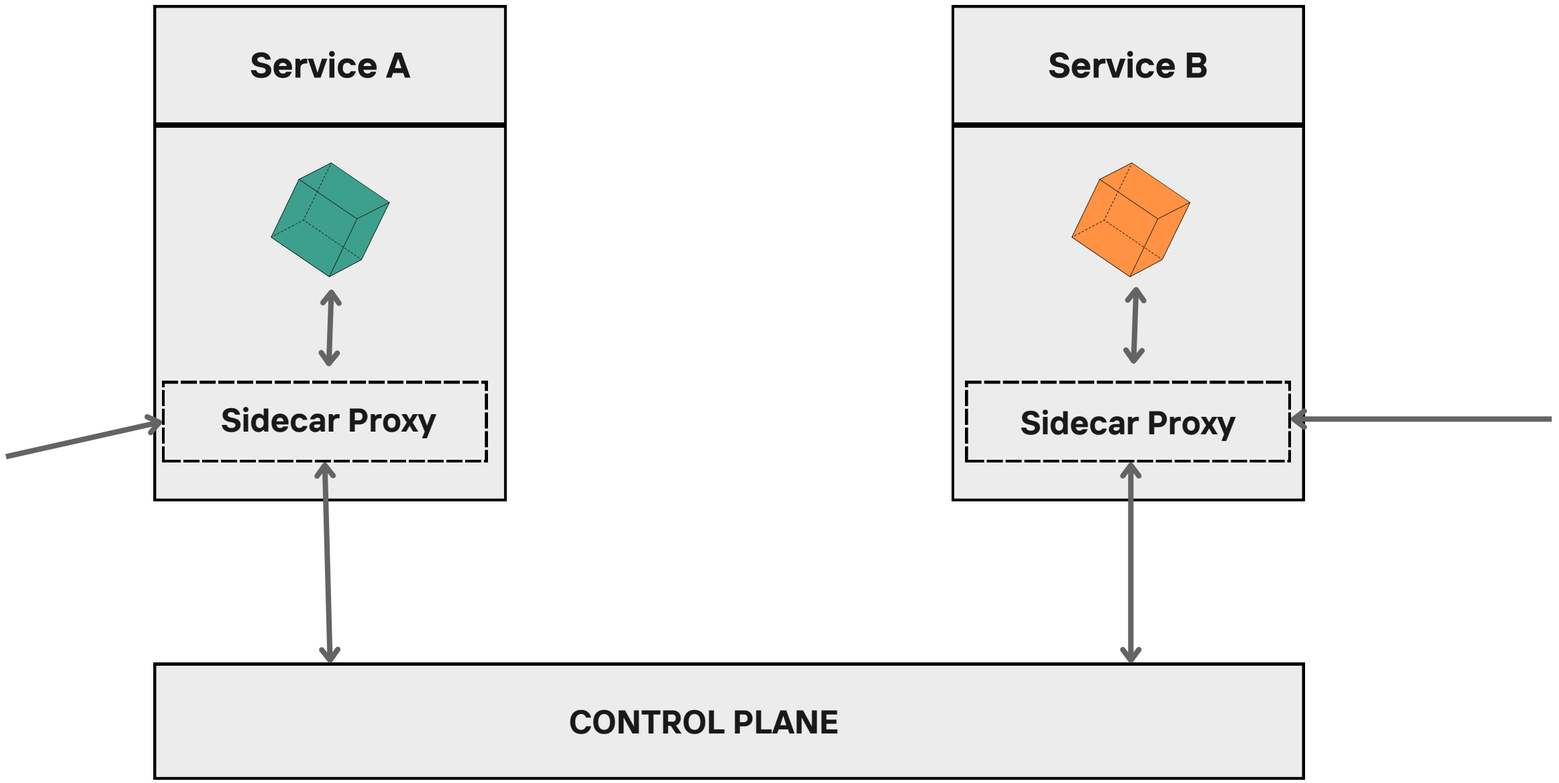
Security Enforcement

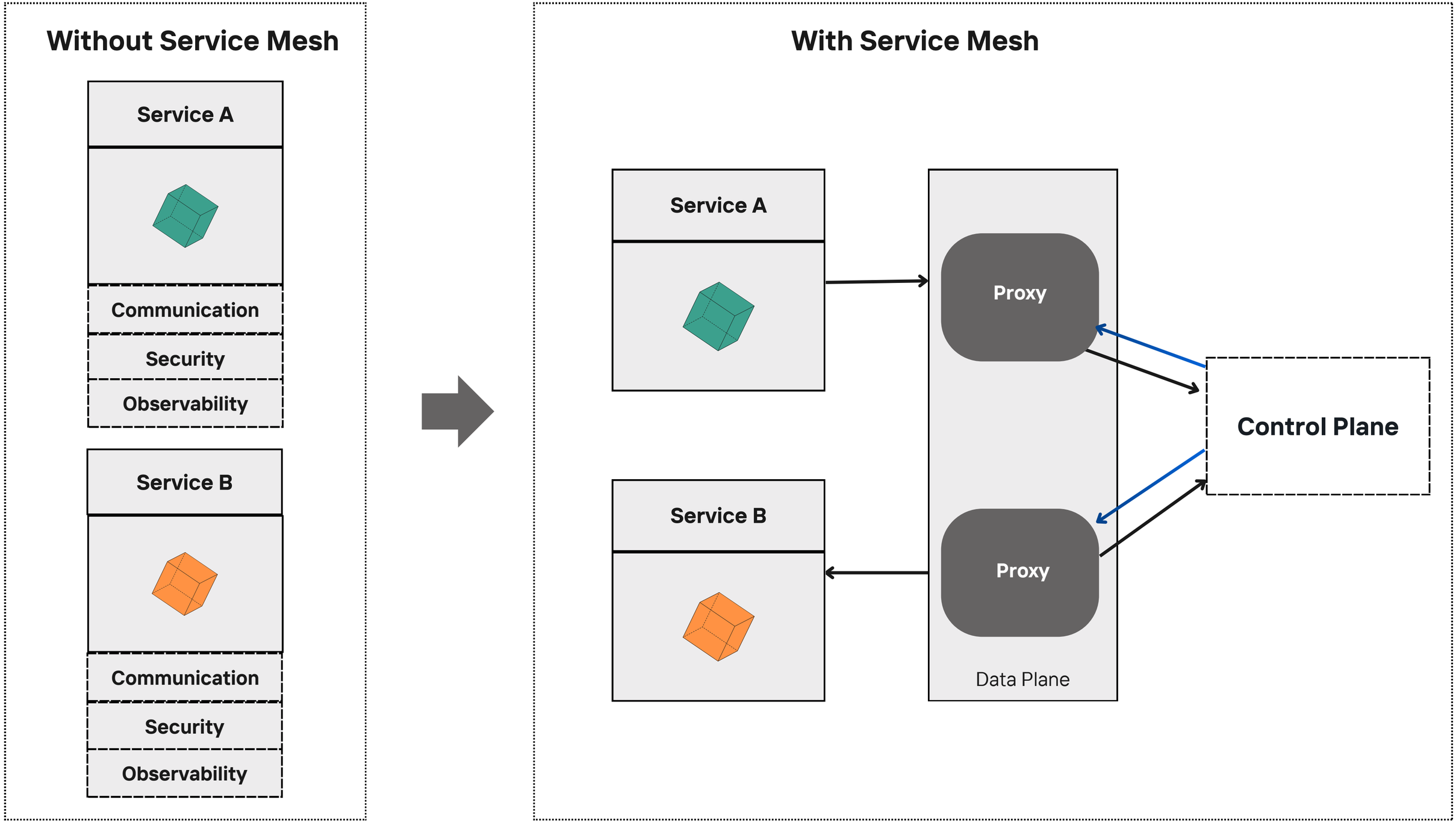
The service mesh can enforce security policies like authentication, authorization, and encryption for communication between microservices.

Enhanced Observability

The service mesh provides valuable insights into service-to-service communication patterns, aiding in troubleshooting and performance optimization.

Service Mesh Architecture




Without Service Mesh
Service A
Communication
Security
Observability
Service B
Communication
Security
Observability
With Service Mesh
Service A
Service B
Proxy
Proxy
Data Plane
Control Plane

Service Mesh Benefits

A service mesh provides a range of core features that enhance the management, observability, and security of microservices within a Kubernetes environment.

Service Discovery and Load Balancing

The service mesh automatically discovers and registers microservices within the cluster.

Manual Scaling

Adding new machines and containers manually is time-consuming and inefficient.

Update and Deployment Issues

Updates require manual changes on each machine, leading to potential inconsistencies.

Configuration Management

Ensuring consistent configurations across all resources is error-prone and difficult.

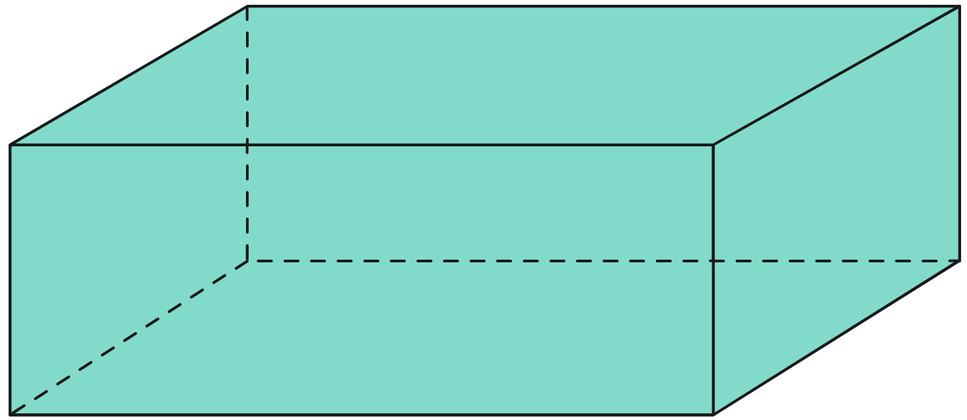
Istio

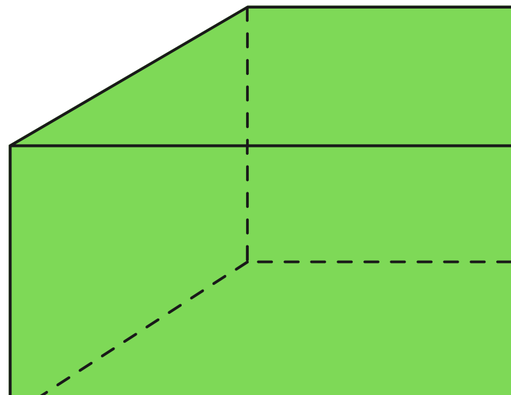
Service Mesh Implementation

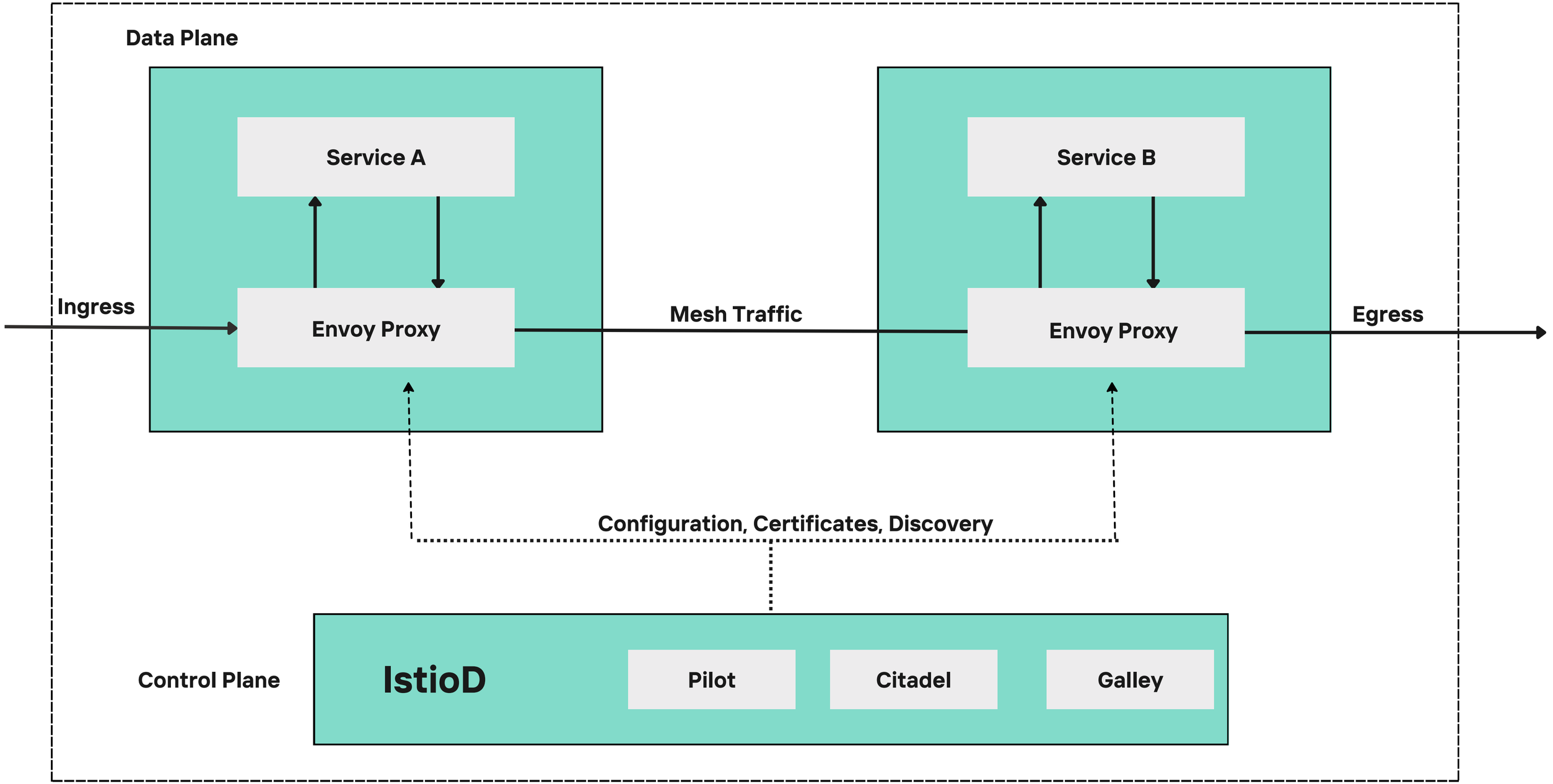
What is ISTIO?

Istio is an open-source service mesh that simplifies managing microservices by providing tools for traffic management, observability, and security.

- **Service Mesh**: A layer for managing service-to-service communication.
- **Open Source**: Community-driven and widely supported.

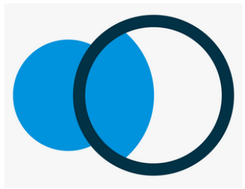





Data Plane
Service A
Service B
Ingress
Envoy Proxy
Mesh Traffic
Envoy Proxy
Egress
Configuration, Certificates, Discovery
Control Plane
IstioD
Pilot
Citadel
Galley

Observability in Istio

Istio integrates seamlessly with powerful observability tools like Jaeger, Kiali, and Grafana, enhancing the visibility and monitoring of your microservices.



Kiali

Provides a graphical interface to observe the service mesh, view metrics, and manage microservices

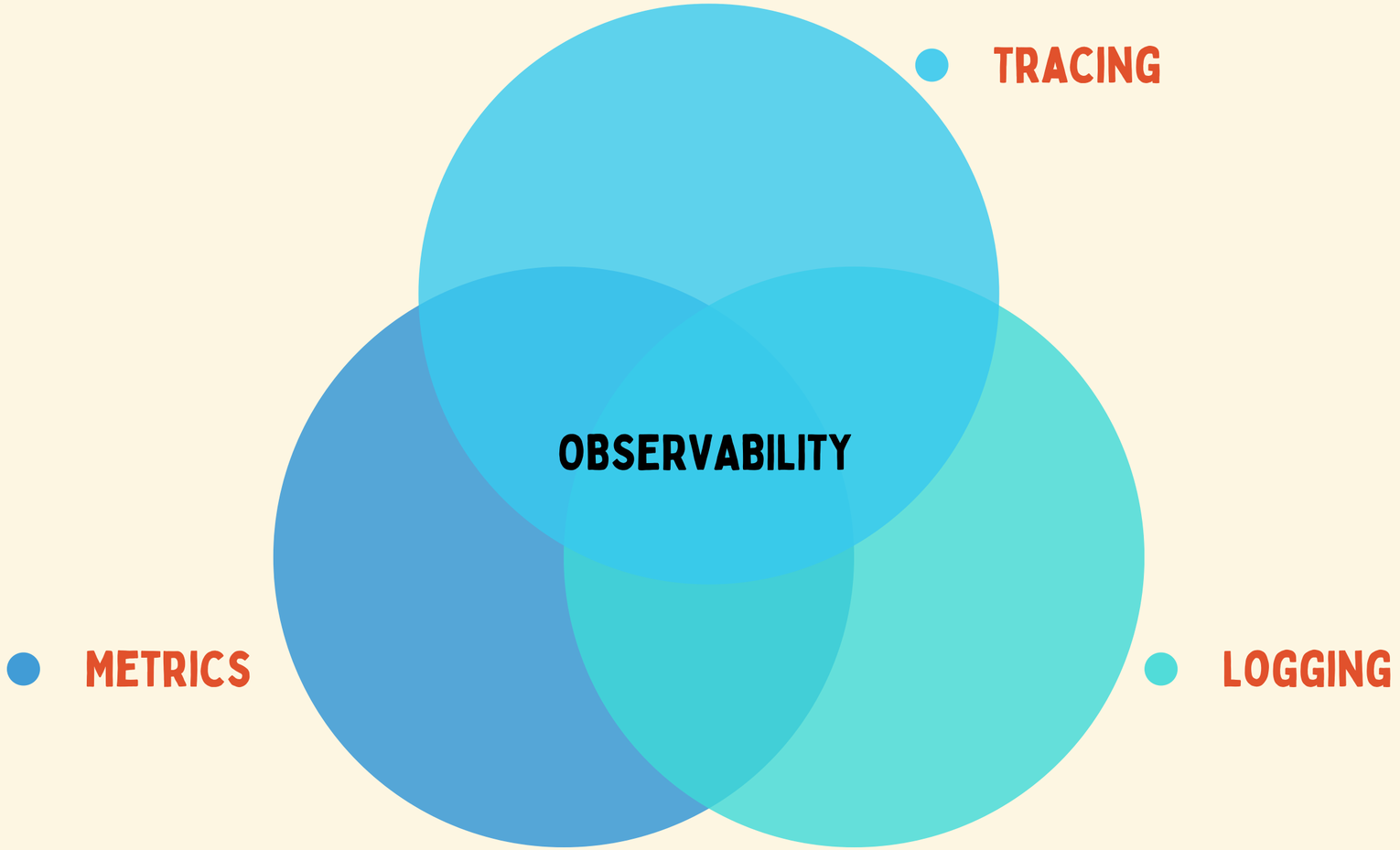
Jaeger

Tracks and visualizes the flow of requests through services, helping identify performance bottlenecks and failures

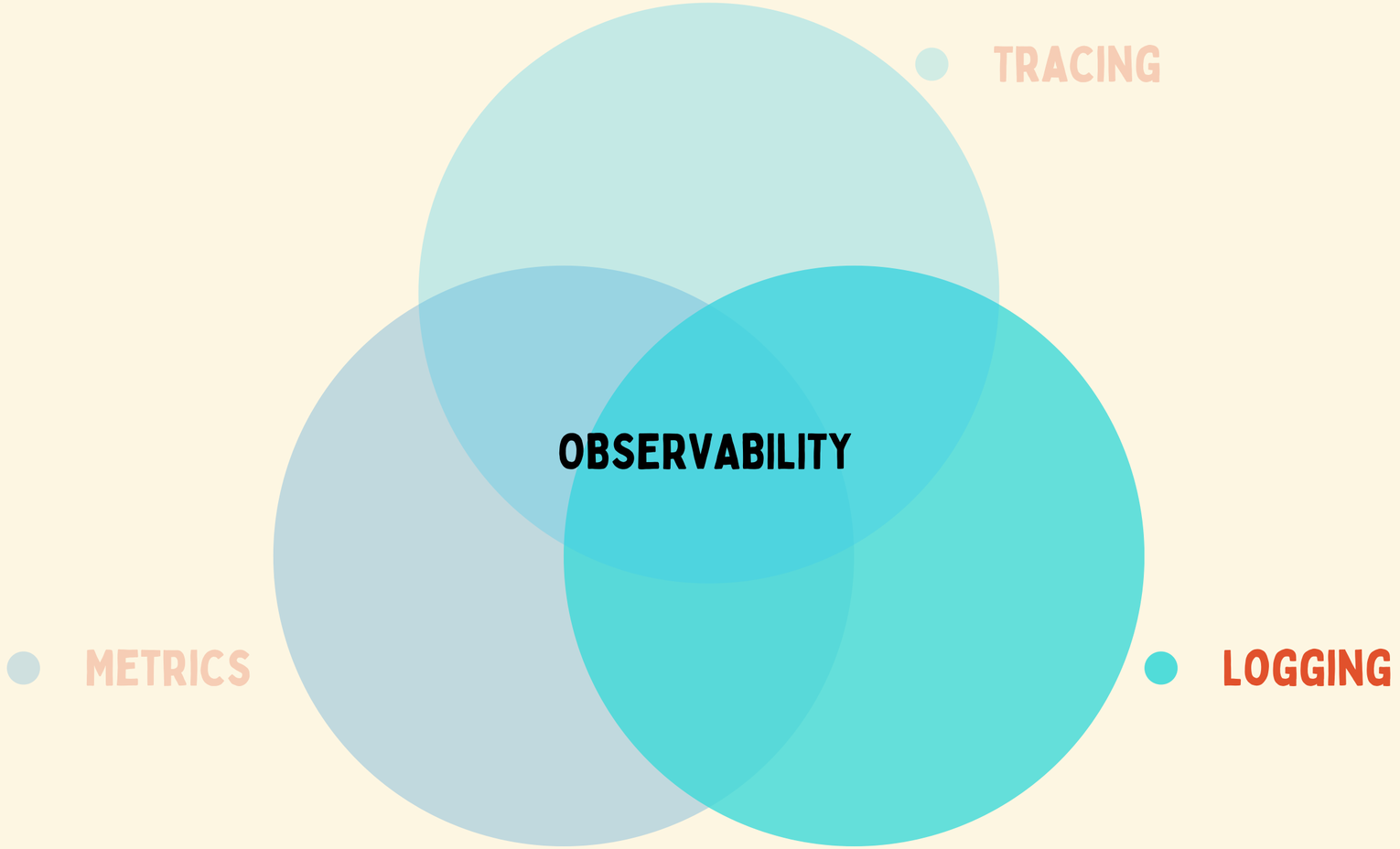
Kiali

Visualizes metrics collected from Prometheus, providing dashboards for real-time monitoring of system health and performance.

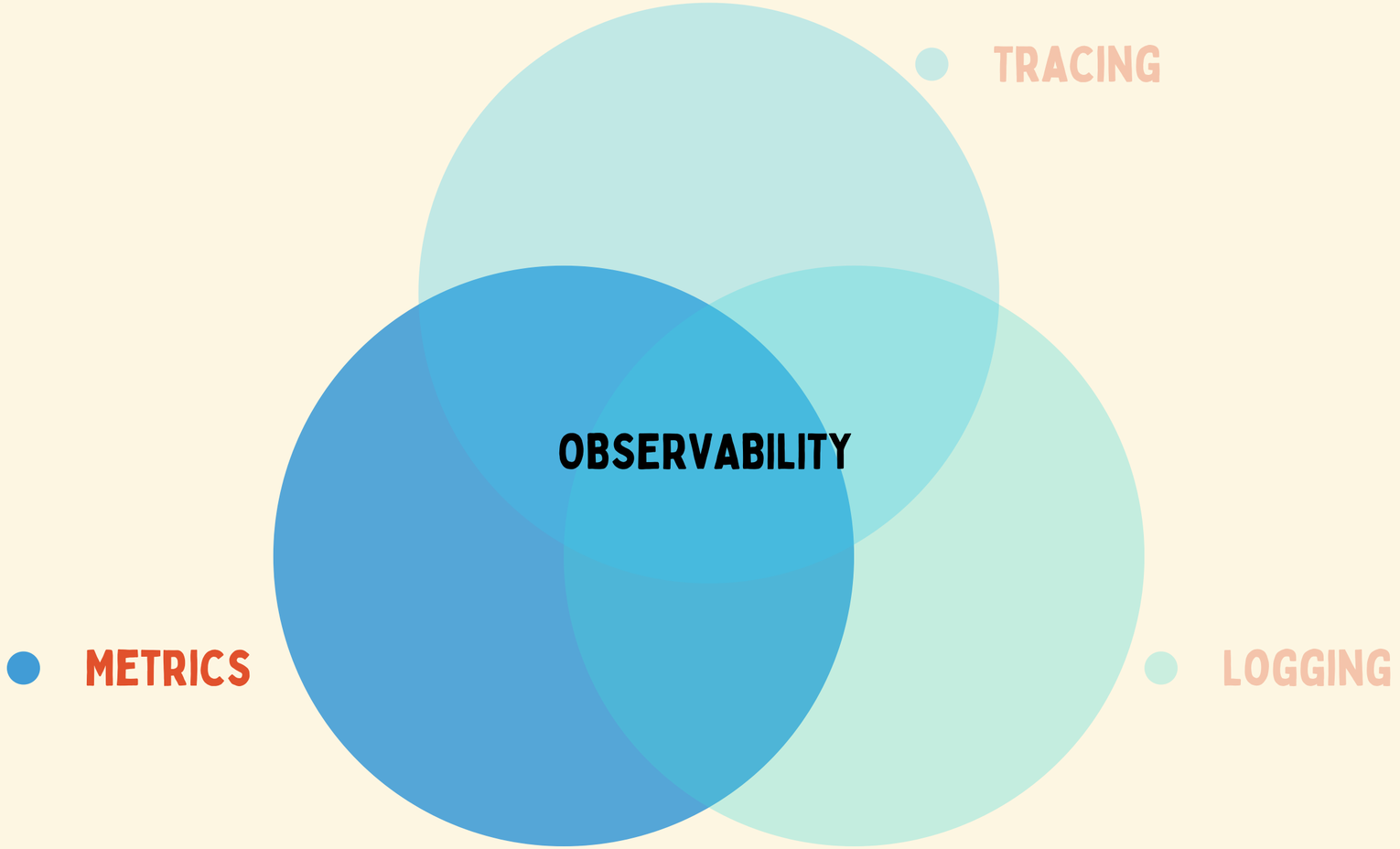
THREE PILLARS



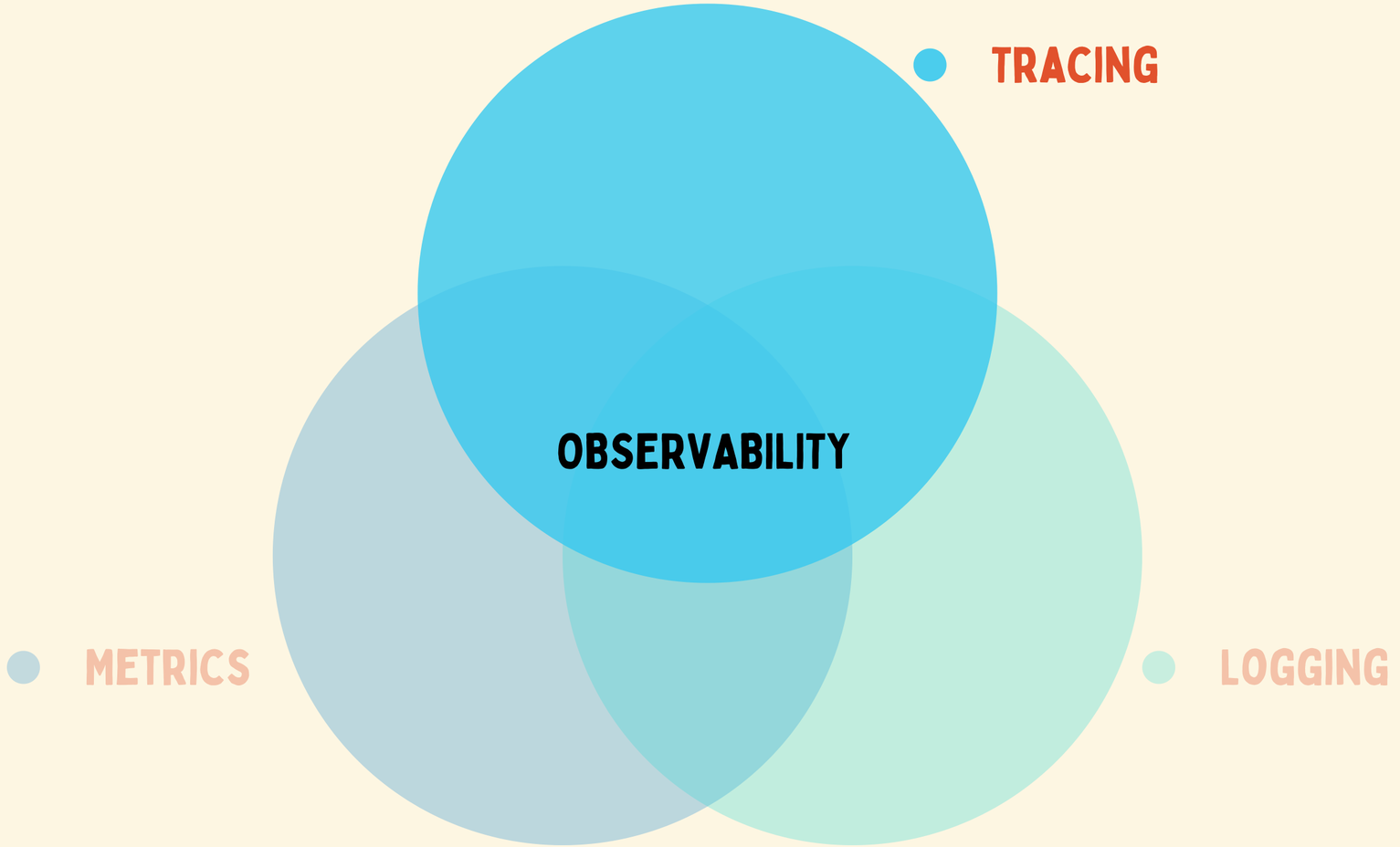
THREE PILLARS



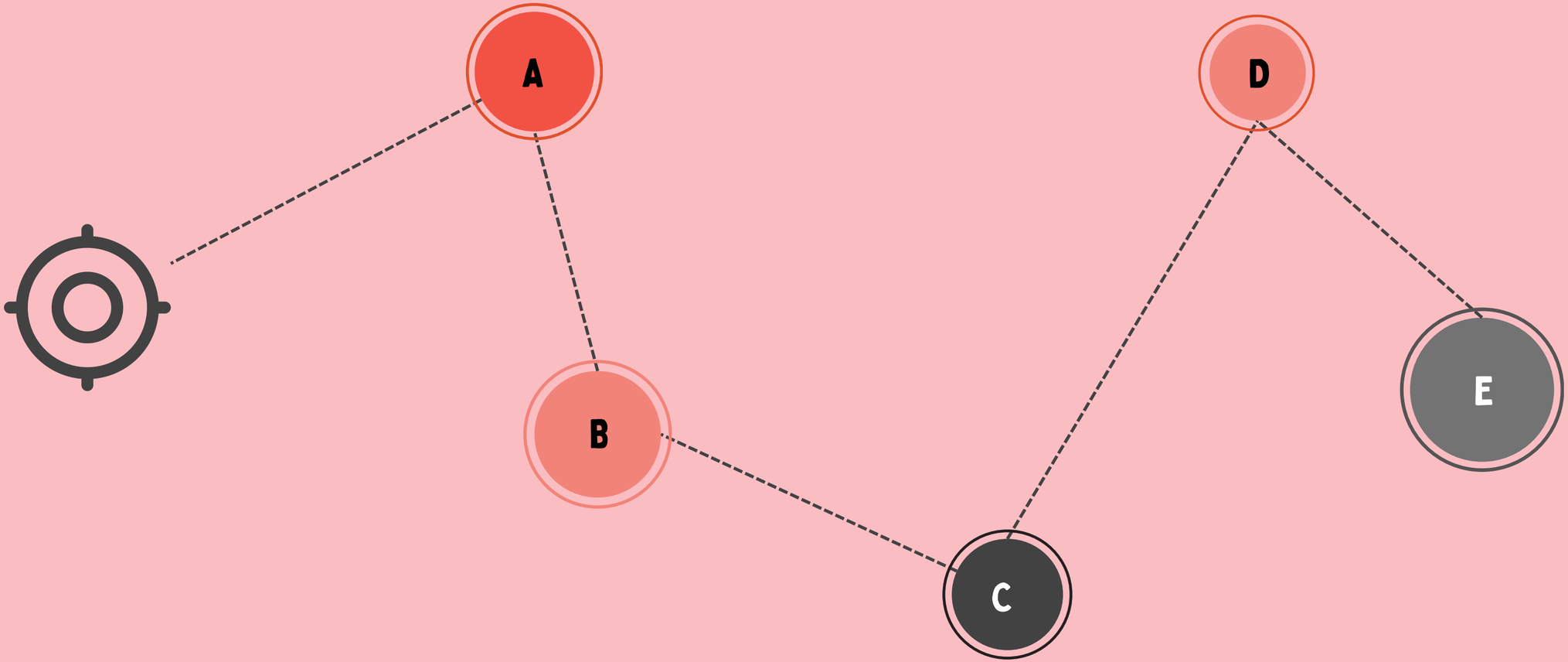
THREE PILLARS



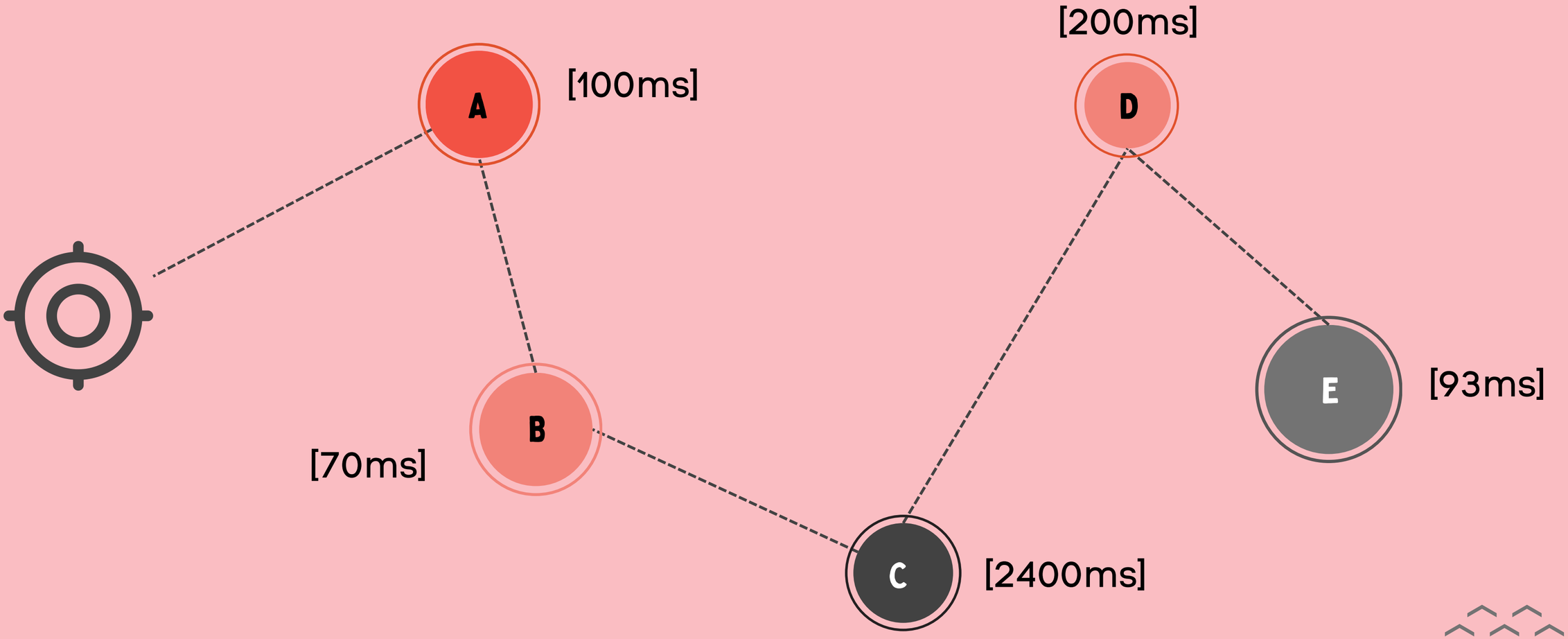
THREE PILLARS



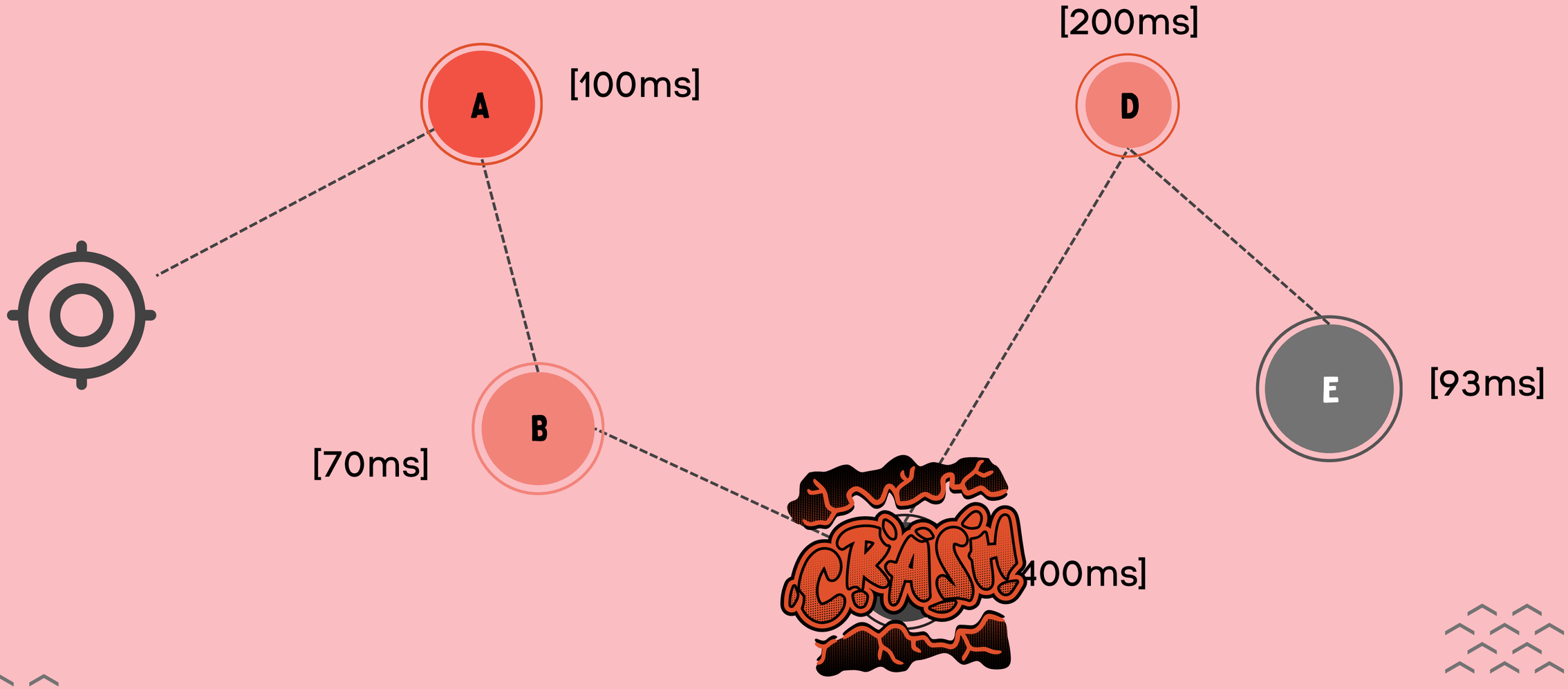
REQUEST



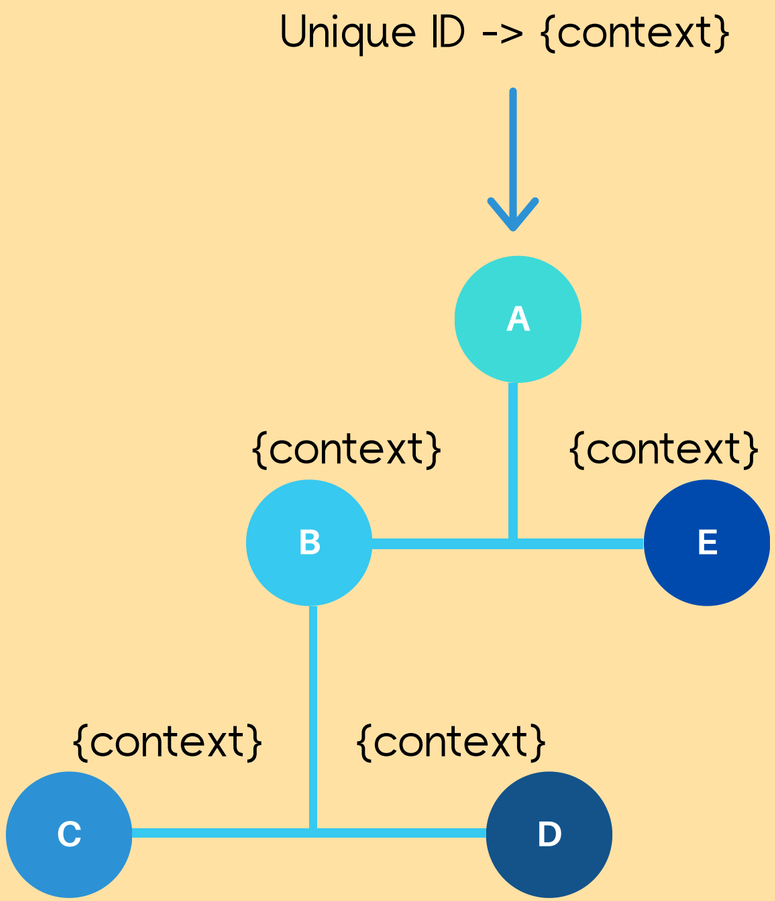
REQUEST

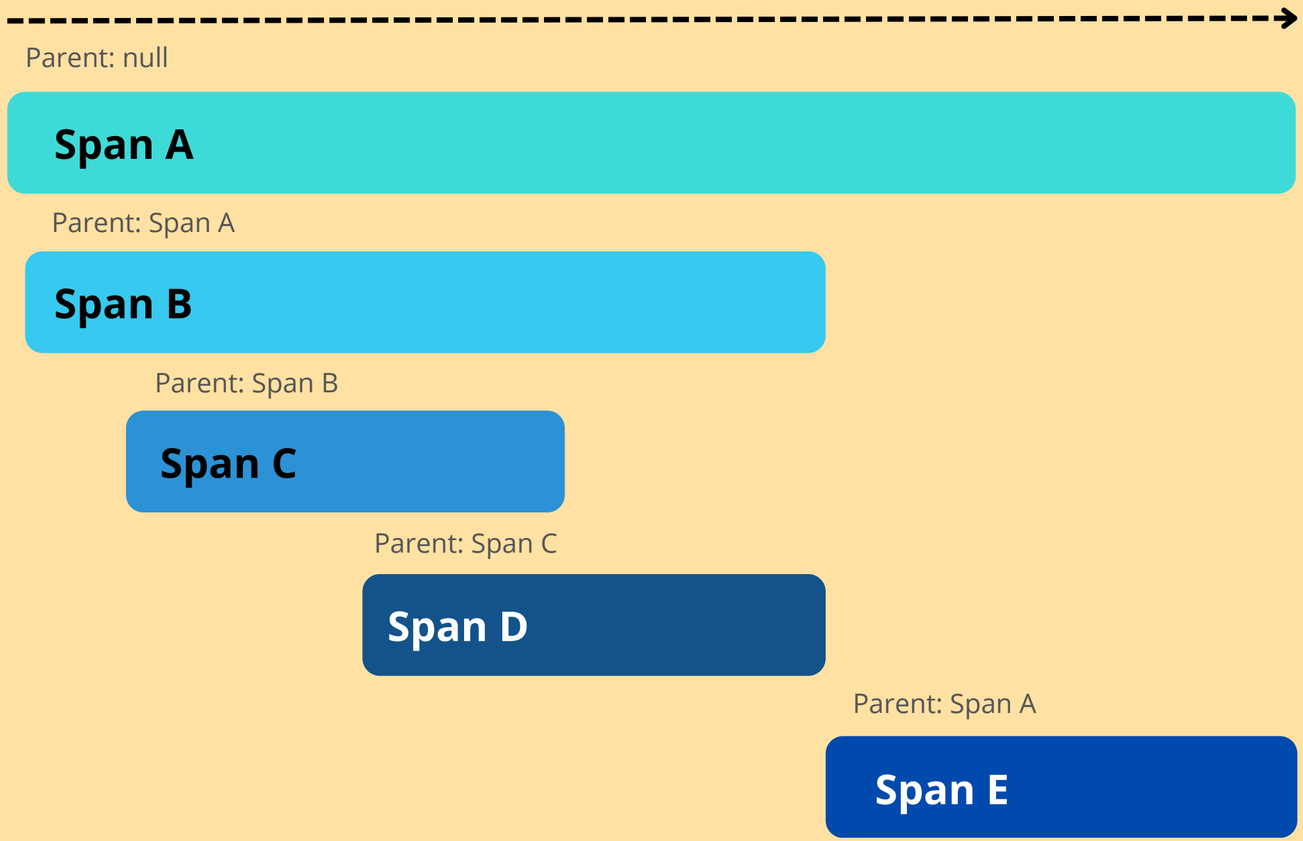


REQUEST

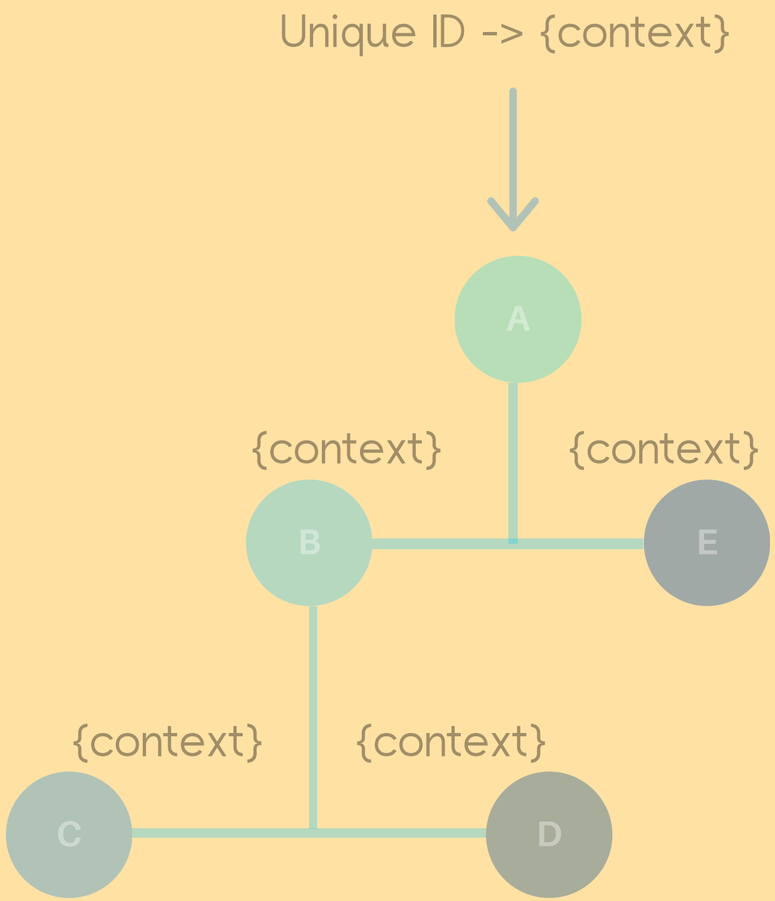


SPANS


Unique ID -> {context}
A
{context}
{context}
B
E
{context}
{context}
C
D


Parent: null
Span A
Parent: Span A
Span B
Parent: Span B
Span C
Parent: Span C
Span D
Parent: Span A
Span E

TRACE



TRACE

Parent: null

Span A Trace ID: X

Parent: Span A

Span B Trace ID: X

Parent: Span B

Span C Trace ID: X

Parent: Span C

Span D Trace ID: X

Parent: Span A

Span E Trace ID: X

How do we configure Istio?

CRD (Custom Resource Definition)

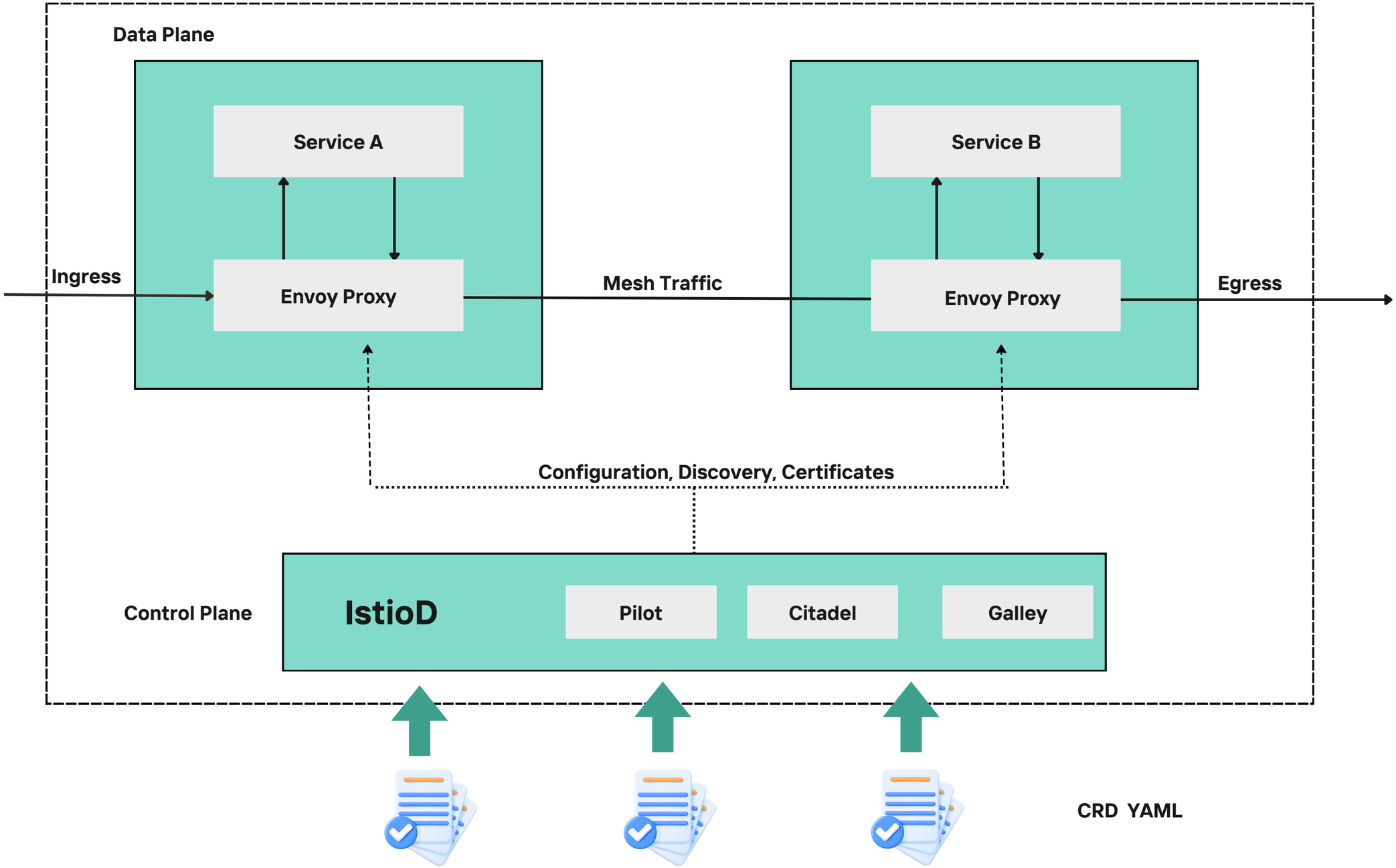
CRDs allow you to define and manage Istio-specific configurations, such as VirtualServices, DestinationRules, and Gateways, providing fine-grained control over traffic management and policies.

Traffic Management: Control the flow of traffic between services.

Istio provides powerful traffic management features, including routing, load balancing, retries, and circuit breaking. You can configure these settings using Istio CRDs to optimize service communication and performance.

Observability Setup: Enable monitoring, logging, and tracing.

Integrate Istio with observability tools like Prometheus, Grafana, Jaeger, and Kiali to gain insights into service behavior, monitor performance, and troubleshoot issues effectively.


Data Plane
Service A
Service B
Ingress
Envoy Proxy
Mesh Traffic
Envoy Proxy
Egress
Configuration, Discovery, Certificates
Control Plane
IstioD
Pilot
Citadel
Galley
CRD YAML

Kiali

Kiali provides observability, visualizations, and configuration capabilities for the Istio service mesh.

Service Graph: Visualizes the service mesh topology.

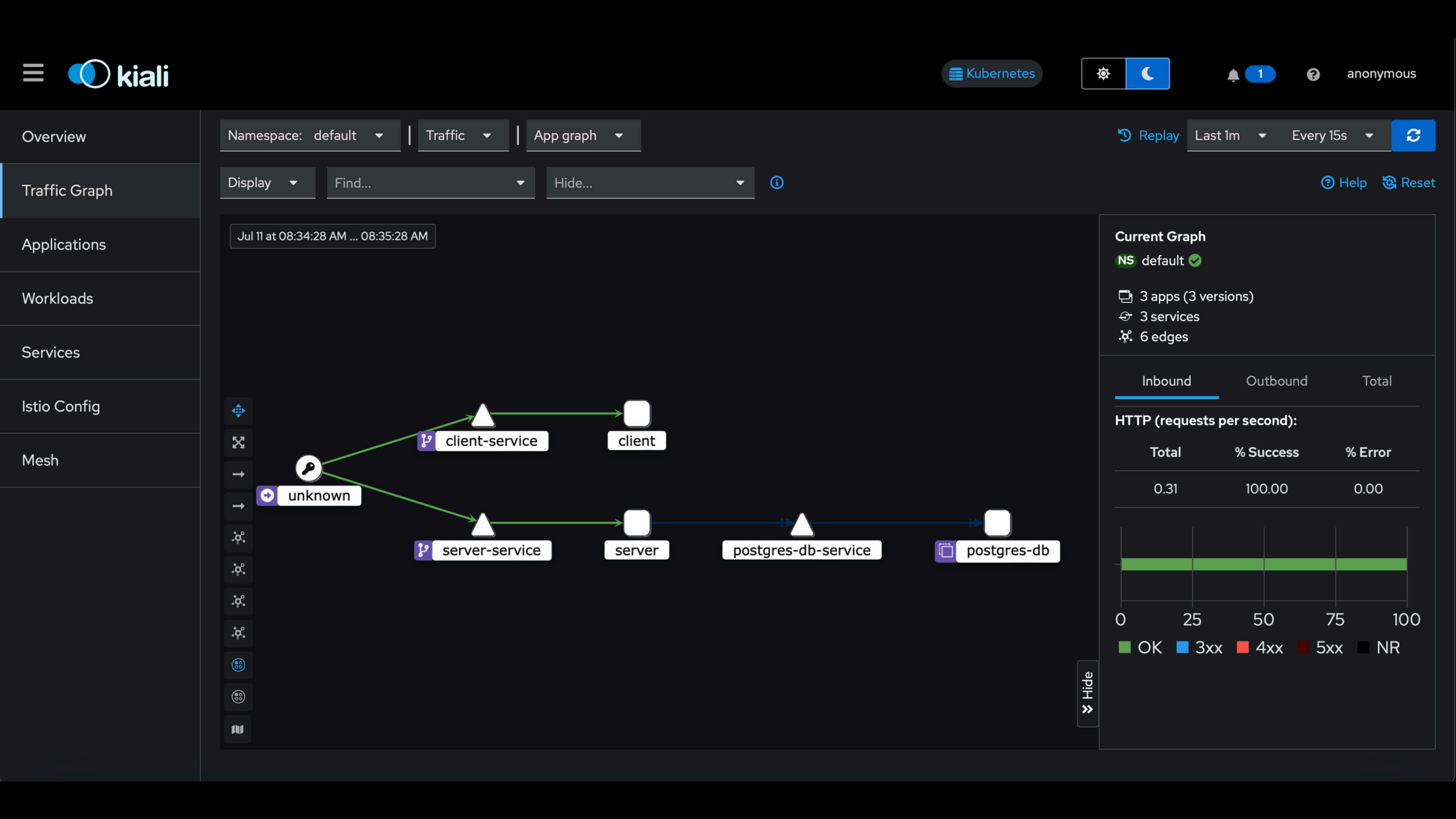
Shows how services are connected and their interactions.

Health Monitoring: Displays the health status of services.

Monitors service performance and detects anomalies.

It provides a visual representation of the service mesh topology, showing relationships between microservices.

Kiali displays real-time traffic flows, health metrics, and resource utilization within the mesh.


kiali
Kubernetes
1
anonymous
Overview
Traffic Graph
Applications
Workloads
Services
Istio Config
Mesh
Namespace: default
Traffic
App graph
Replay
Last 1m
Every 15s
Display
Find...
Hide...
Help
Reset
Jul 11 at 08:34:28 AM ... 08:35:28 AM
unknown
client-service
client
server-service
server
postgres-db-service
postgres-db
Current Graph
NS default
3 apps (3 versions)
3 services
6 edges
Inbound
Outbound
Total
HTTP (requests per second):
Total
% Success
% Error
0.31
100.00
0.00
0
25
50
75
100
OK
3xx
4xx
5xx
NR
Hide

Jaegar

Jaeger is a tool for monitoring and troubleshooting transactions in complex distributed systems.

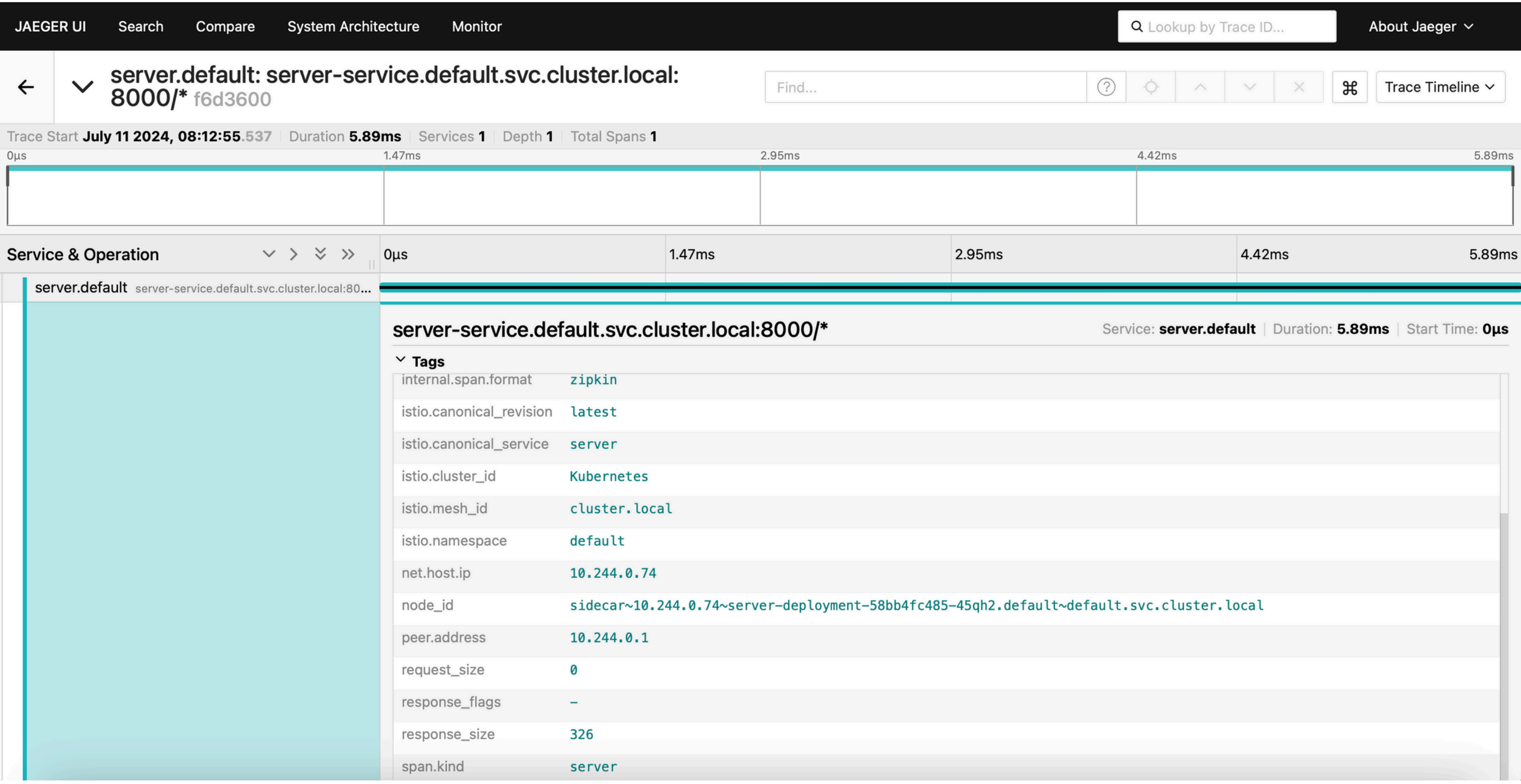
Trace Collection: Captures detailed tracing information.

Collects data about individual requests and their journey through the services.

Root Cause Analysis: Identifies performance issues and bottlenecks.

Helps pinpoint where latency and failures occur.

Istio automatically injects tracing information into service calls, allowing Jaeger to capture the complete request flow.


JAEGER UI
Search
Compare
System Architecture
Monitor
Lookup by Trace ID...
About Jaeger
server.default: server-service.default.svc.cluster.local: 8000/* f6d3600
Find...
Trace Timeline
Trace Start July 11 2024, 08:12:55.537 | Duration 5.89ms | Services 1 | Depth 1 | Total Spans 1
0µs
1.47ms
2.95ms
4.42ms
5.89ms
Service & Operation
server.default server-service.default.svc.cluster.local:80...
server-service.default.svc.cluster.local:8000/*
Service: server.default | Duration: 5.89ms | Start Time: 0µs
Tags
internal.span.format zipkin
istio.canonical_revision latest
istio.canonical_service server
istio.cluster_id Kubernetes
istio.mesh_id cluster.local
istio.namespace default
net.host.ip 10.244.0.74
node_id sidecar~10.244.0.74~server-deployment-58bb4fc485-45qh2.default~default.svc.cluster.local
peer.address 10.244.0.1
request_size 0
response_flags -
response_size 326
span.kind server

Grafana

Grafana is a powerful tool for creating dashboards and visualizing metrics collected from Prometheus.

Custom Dashboards: Build interactive and customizable dashboards.

Provides real-time insights into system performance and resource usage.

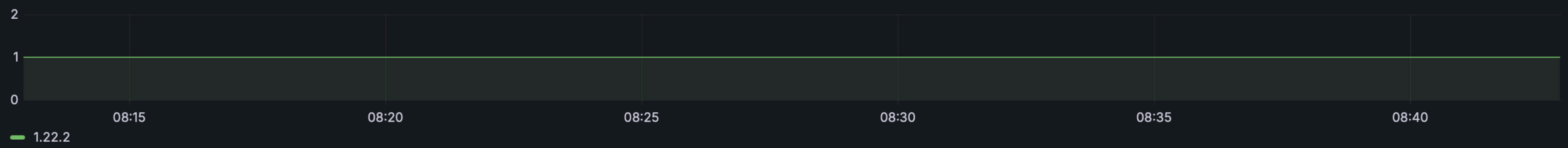
Alerting: Set up alerts based on specific metrics.

Notifies when certain thresholds are reached, enabling proactive management.

Grafana allows us to create custom dashboards that visualize these Istio metrics alongside tracing data from Jaeger.

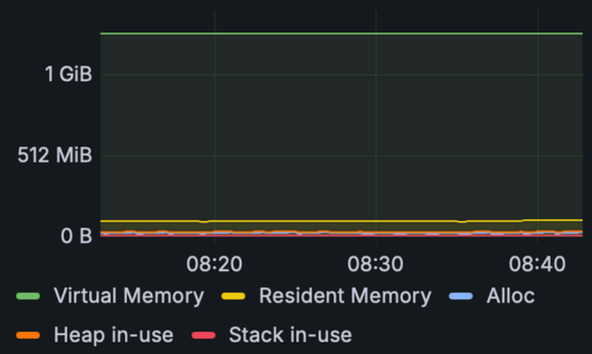
Deployed Versions

Pilot Versions

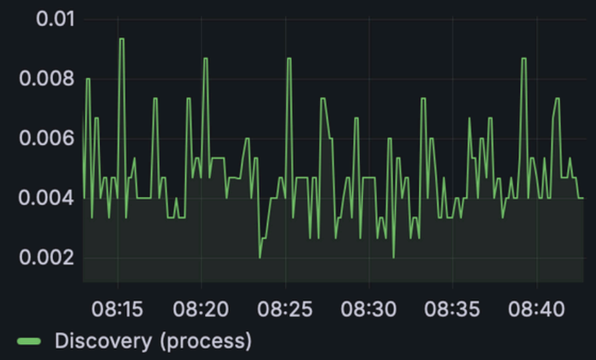


Resource Usage

Memory



CPU



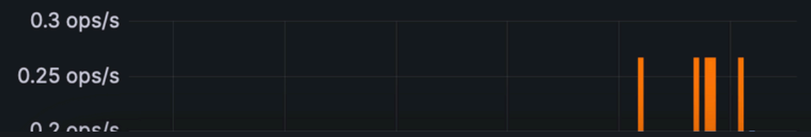
Disk

No data

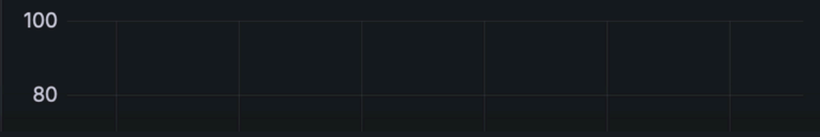
Goroutines

Pilot Push Information

Pilot Pushes



Pilot Errors

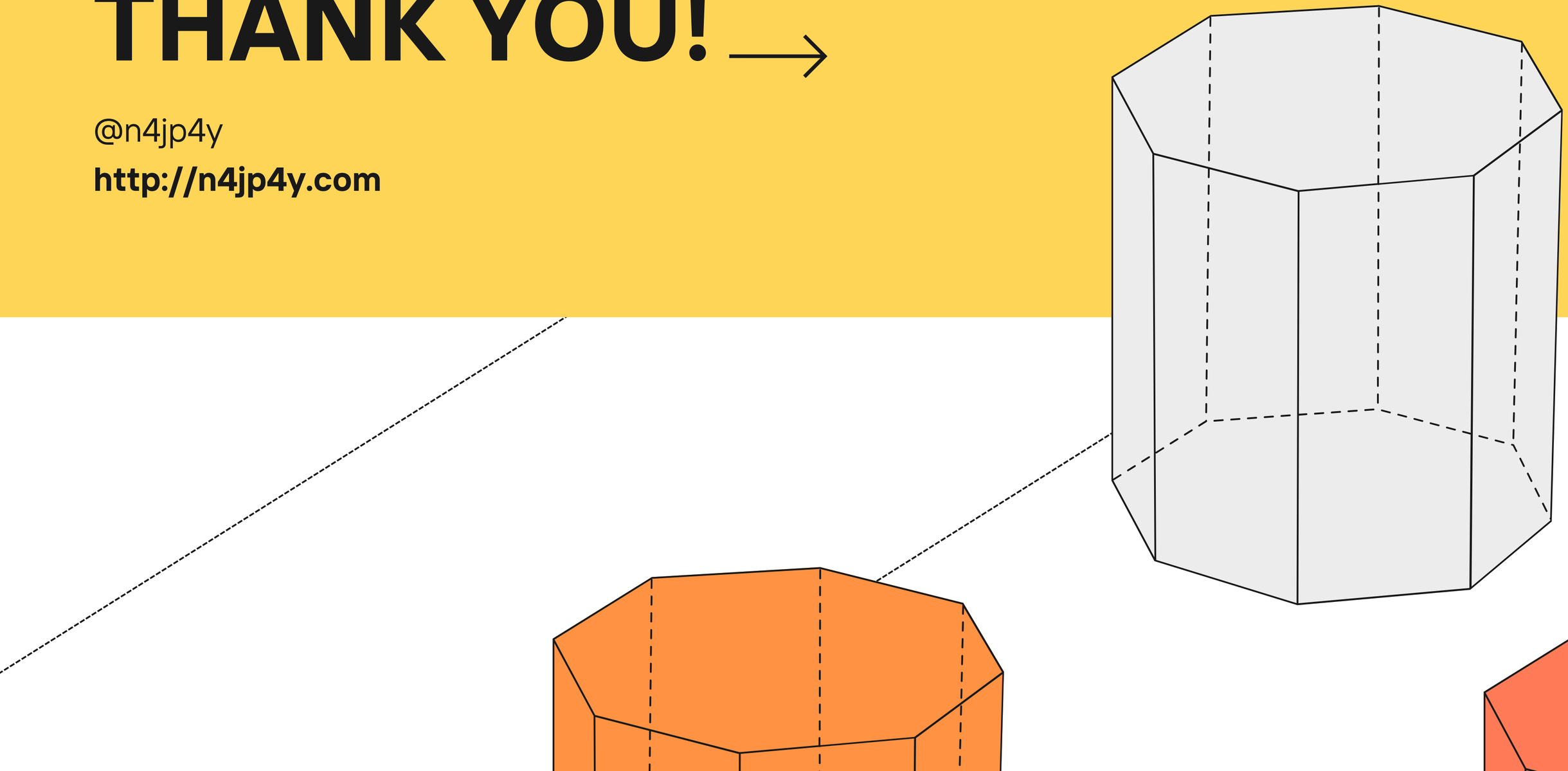


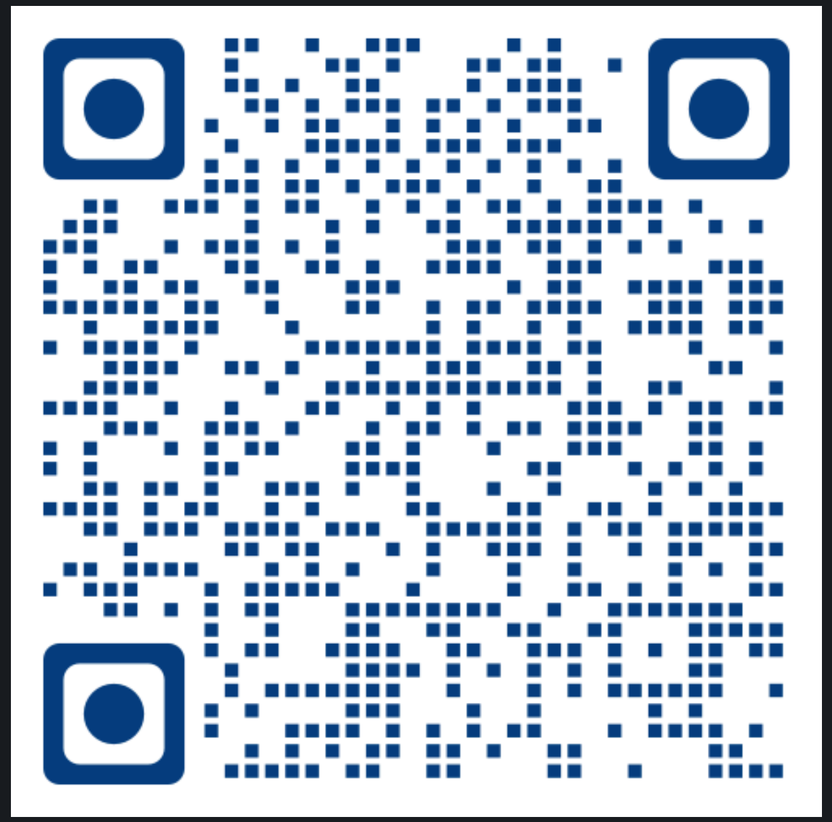
Proxy Push Time

THANK YOU! →

@n4jp4y

http://n4jp4y.com





For presentation
and Code Repository

http://n4jp4y.com