virtusa

HIL Tech Council presents

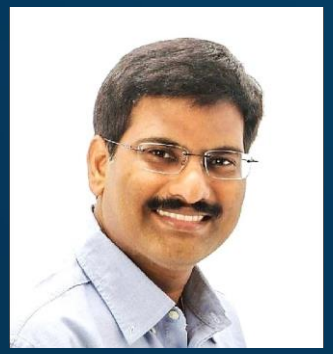
A Live Coding Session

BOOTING Microservices

A complete E2E implementation

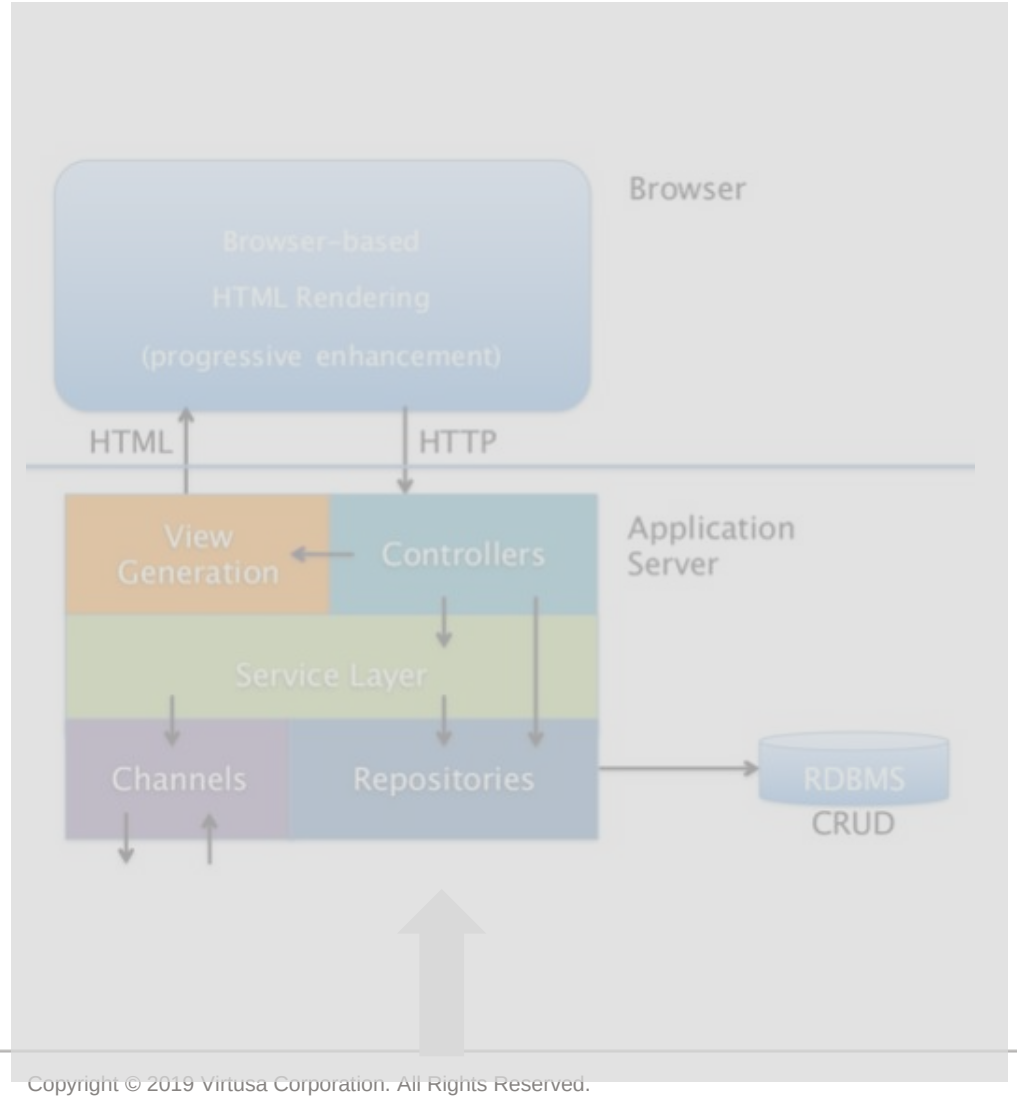
Presenter's Name, ***Narasimhaiah Narahari (NN) & Rasool***

Sept 11th 2019, 3.000 PM to 5.30 PM

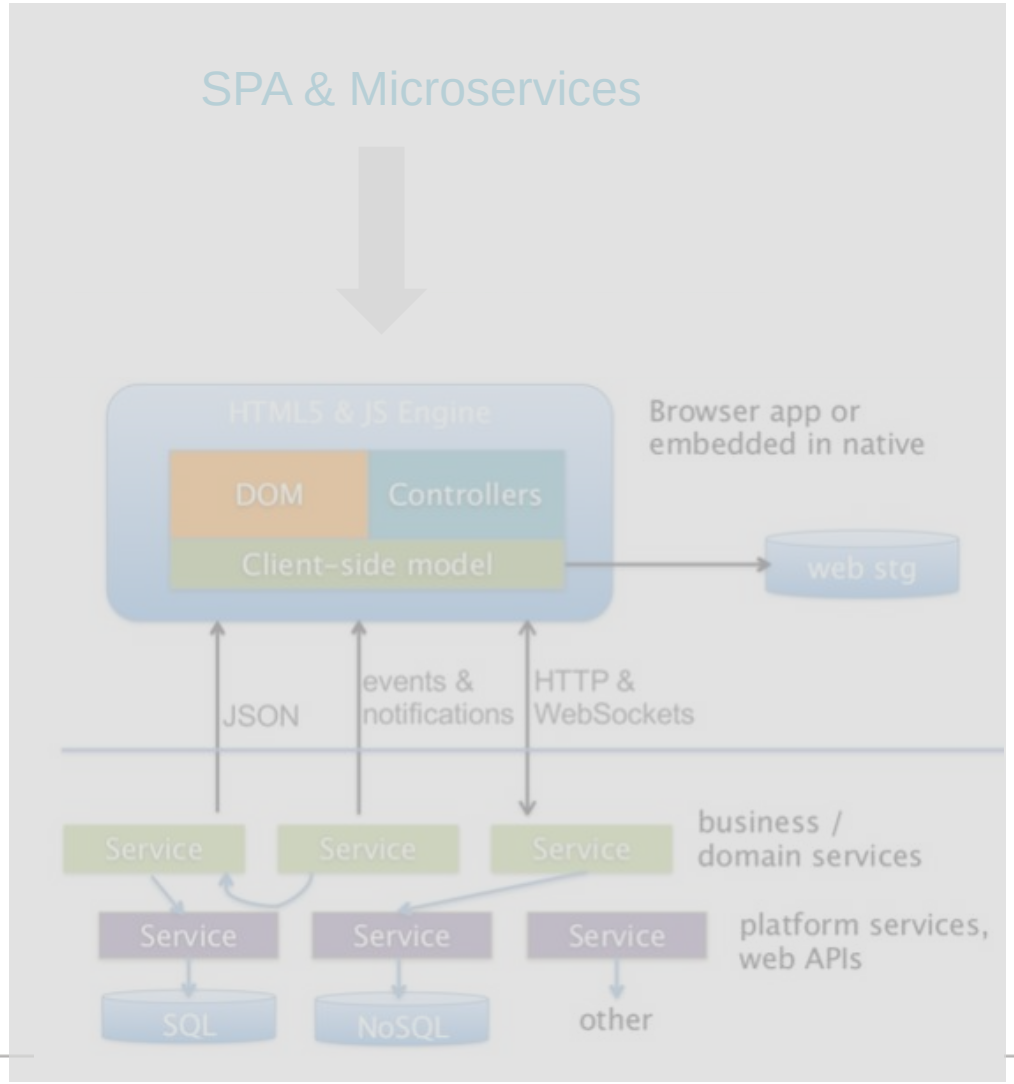


Traditional vs Modern Technical Architecture

virtusa



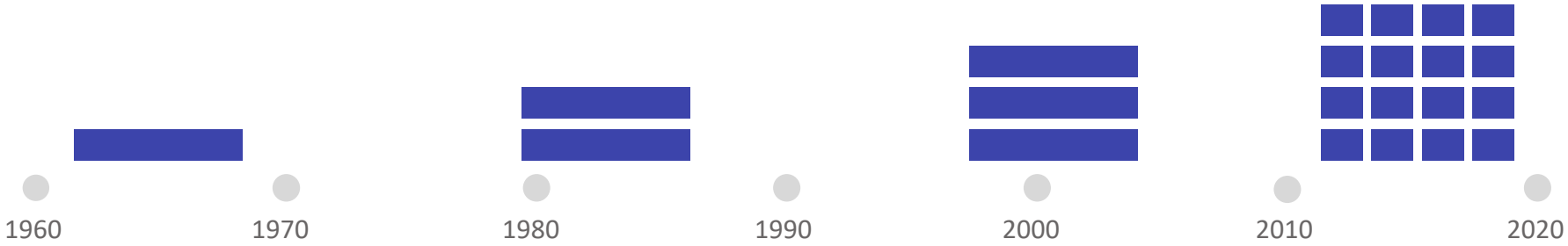
Traditional & Popular



Evolution of Services Orientation

virtusa

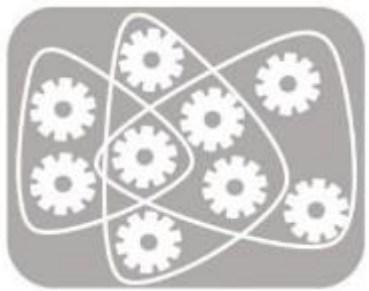
Abstraction in the Enterprise



Era of Mainframe | Era of Client Server | Era of the Web | Era of the Cloud

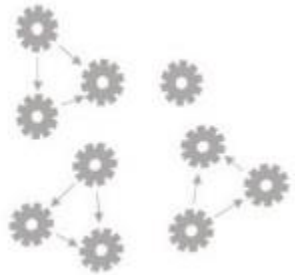
Evolution of Services Orientation

Pre-SOA (Monolithic)
Tight Coupling



Traditional SOA
Looser Coupling

Microservices
Loosely Coupled/Decoupled



Source: Sequoia; Capgemini

What are Microservices?

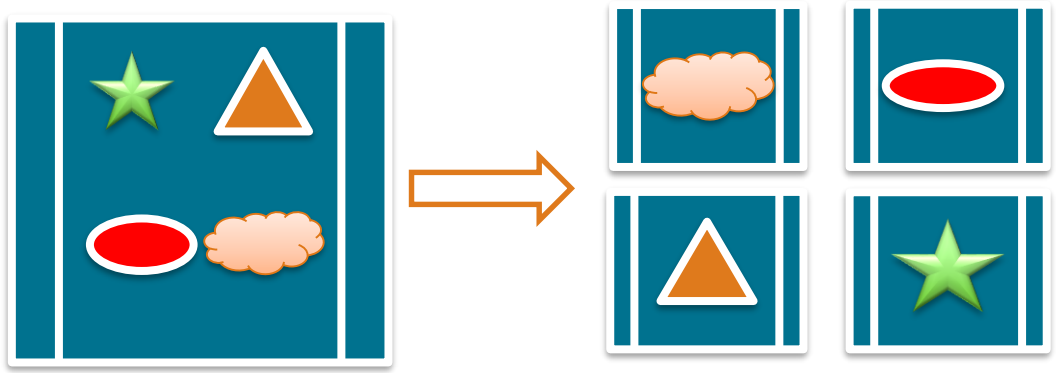
*In short, the microservice **architectural style** is an approach to developing a single application as a suite of **small services**, each running in its **own process** and communicating with **lightweight** mechanisms, often an HTTP resource API.*

*These services are built around **business capabilities** and **independently deployable** by fully **automated deployment** machinery. There is a bare **minimum of centralized management** of these services, which may be **written in different programming languages** and use **different data storage** technologies.*

-- James Lewis and Martin Fowler

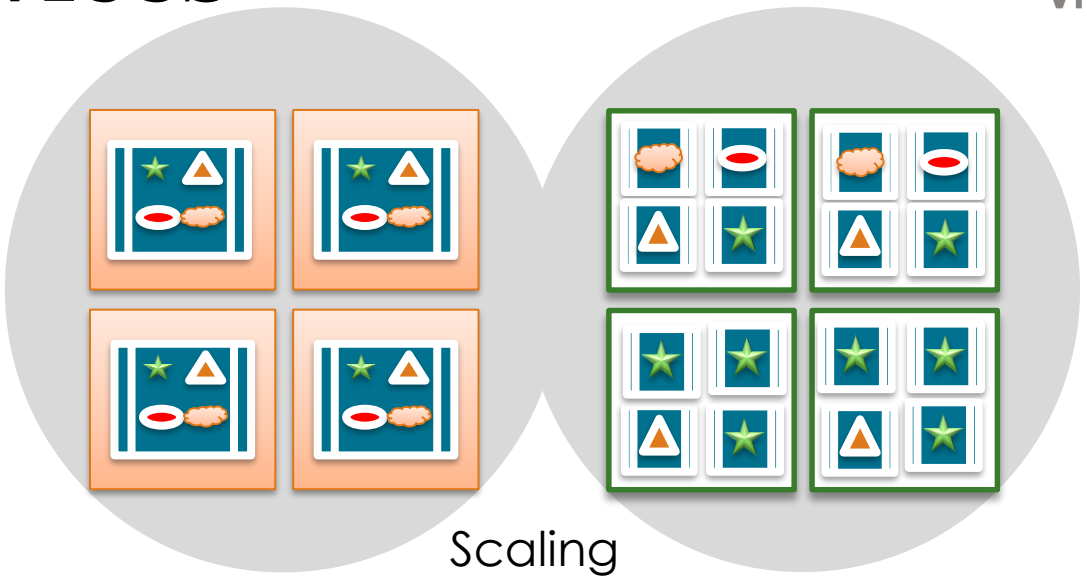
Monolithic vs Microservices

virtusa

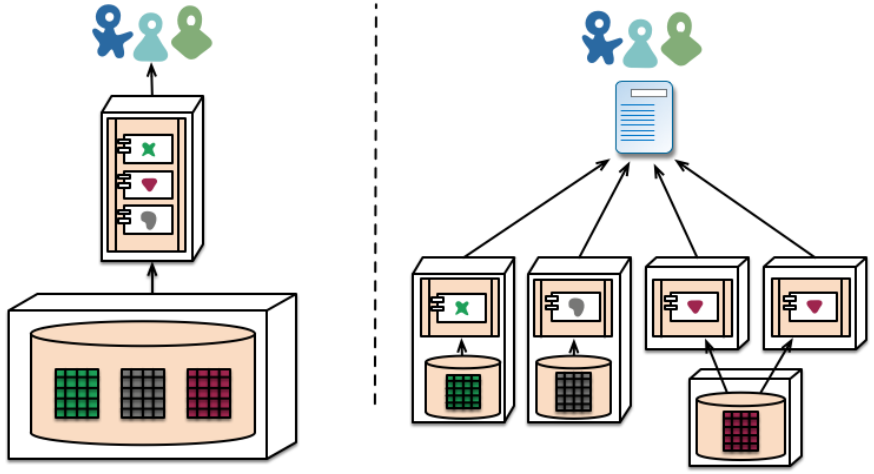


Jumbo

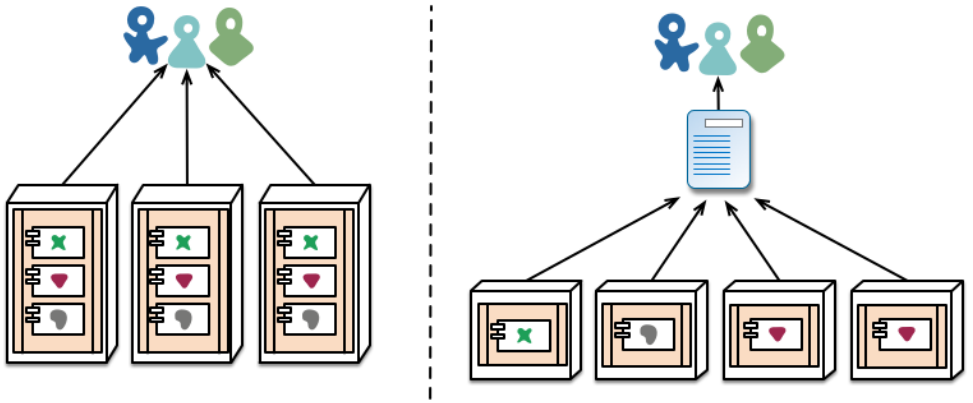
Atomic



Scaling



Decentralized Data Management

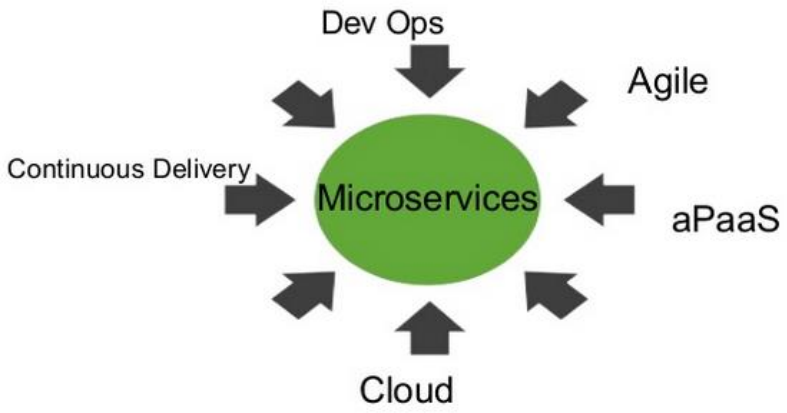


Deployment

Microservices - Eco System, Characteristics

virtusa

Eco System



Characteristics

12-Factor Methodology

Codebase	Dependencies	Config
Backing Services	Build, release, run	Processes
Port binding	Concurrency	Disposability
Dev/prod parity	Logs	Admin processes

CQRS stands for Command Query Responsibility Segregation
DDD stands for Domain-Driven Design

Best practices for microservices architectures

Clip slide

I. Codebase
One codebase tracked in revision control, many deploys

II. Dependencies
Explicitly declare and isolate dependencies

III. Config
Store config in the environment

IV. Backing Services
Treat backing services as attached resources

V. Build, release, run
Strictly separate build and run stages

VI. Processes
Execute the app as one or more stateless processes

VII. Port binding
Export services via port binding

VIII. Concurrency
Scale out via the process model

IX. Disposability
Maximize robustness with fast startup and graceful shutdown

X. Dev/prod parity
Keep development, staging, and production as similar as possible

XI. Logs
Treat logs as event streams

XII. Admin processes
Run admin/management tasks as one-off processes

12factor.net

The 12 factor app is a methodology for building apps that:

- Use **declarative** formats for setup automation, to minimize time and cost for new developers joining the project;
- Have a **clean contract** with the underlying operating system, offering **maximum portability** between execution environments;
- Are suitable for **deployment** on modern **cloud platforms**, obviating the need for servers and systems administration;
- **Minimize divergence** between development and production, enabling continuous deployment for maximum agility;
- And can **scale up** without significant changes to tooling, architecture, or development practices.

The 12 factor methodology can be applied to apps written in any programming language, and which use any combination of backing services (database, queue, memory cache, etc).

Guiding Principles For Microservices

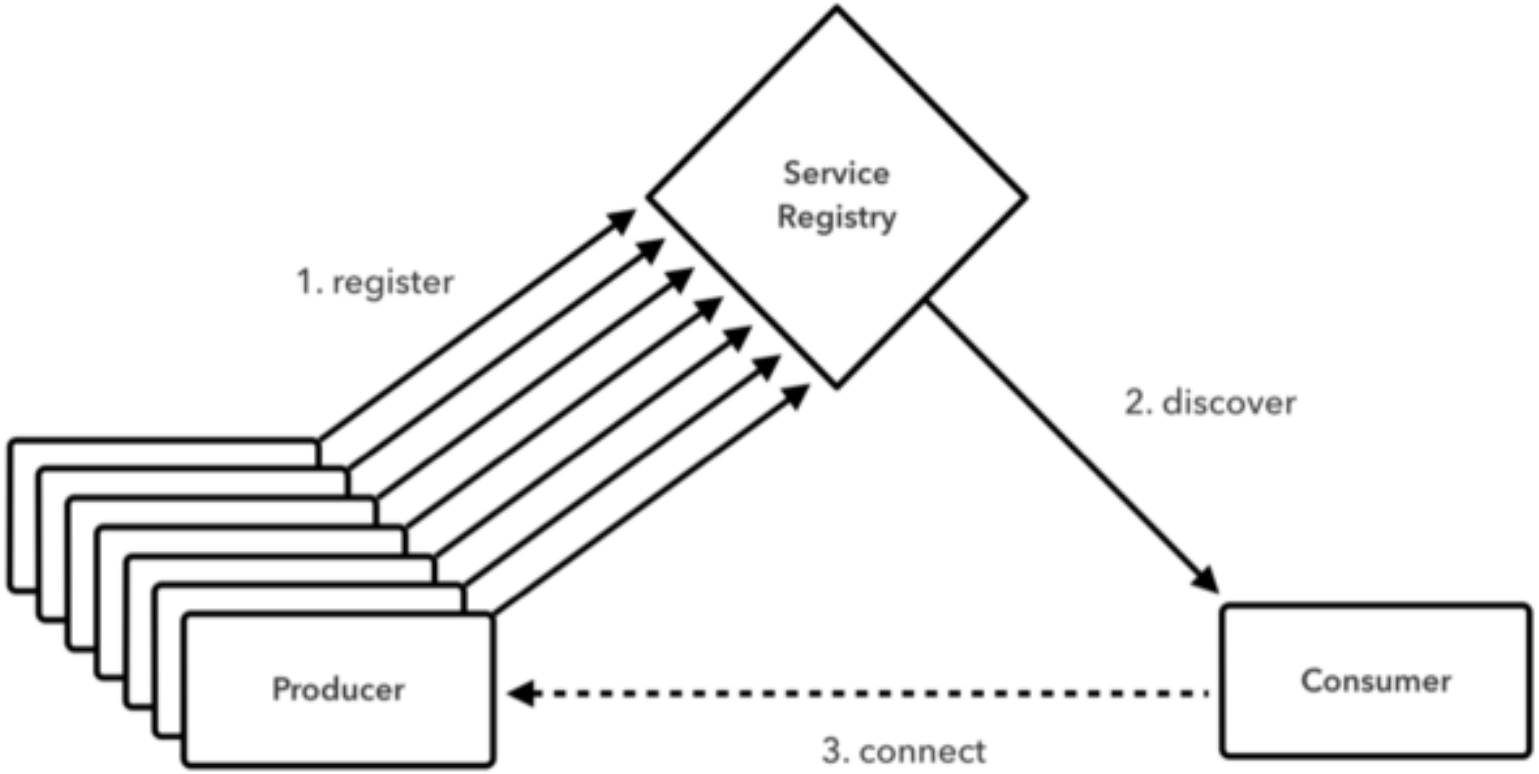
virtusa

- ✓ Modelled Around Business Domain
  - Microservices provide the ability to separate system capability into different domains using Domain Driven Design principles.
- ✓ Culture Of Automation
  - In a Microservice architecture, the number of deployment units increase and an automated solution is needed. This can be achieved through automating the build and deploy process via DevOps.
- ✓ Deploy Independently
  - Agility and scalability is maximized when each Microservice is versioned and deployed independently leveraging native cloud services and DevOps.
- ✓ Consumer First
  - Microservices focus on delivering easy to consume interactions that are device friendly and scalable. They align to the single responsibility principle.
- ✓ Isolate Failure
  - Service Isolation addresses both scalability and resilience consistent with native cloud principles.
- ✓ Highly Observable
  - To offset the independence of each microservice, observable services with correlation ids provide traceability and coordination of services required in an evolving product ecosystem.

Distributed Patterns

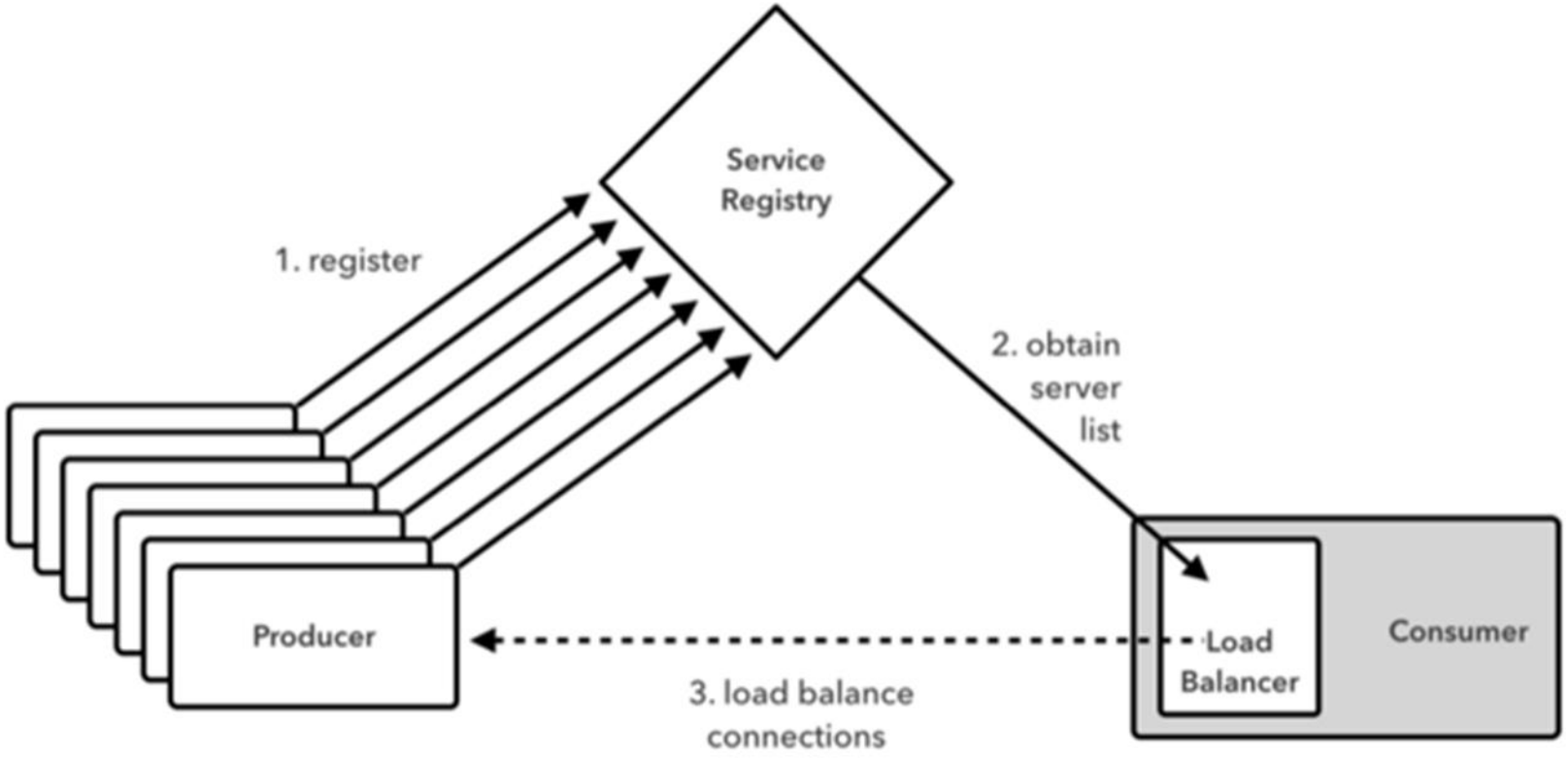
Service Registry and Discovery

virtusa



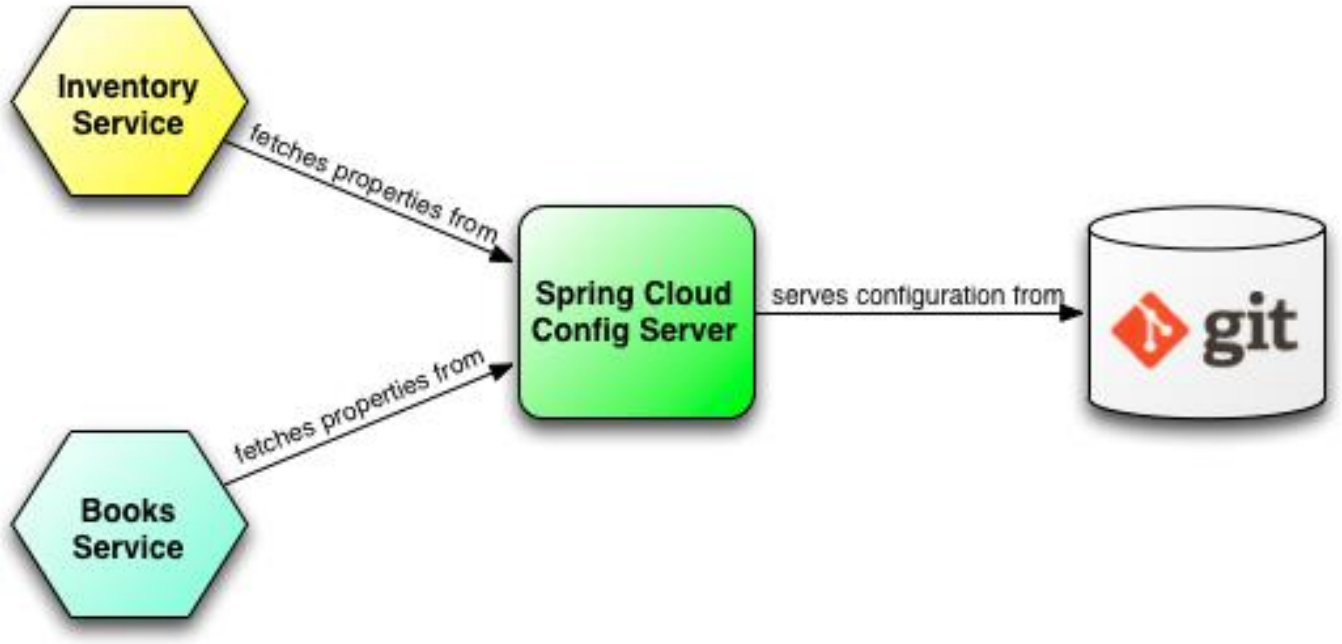
Client side load balancing

virtusa



Centralized/Distributed configuration

virtusa

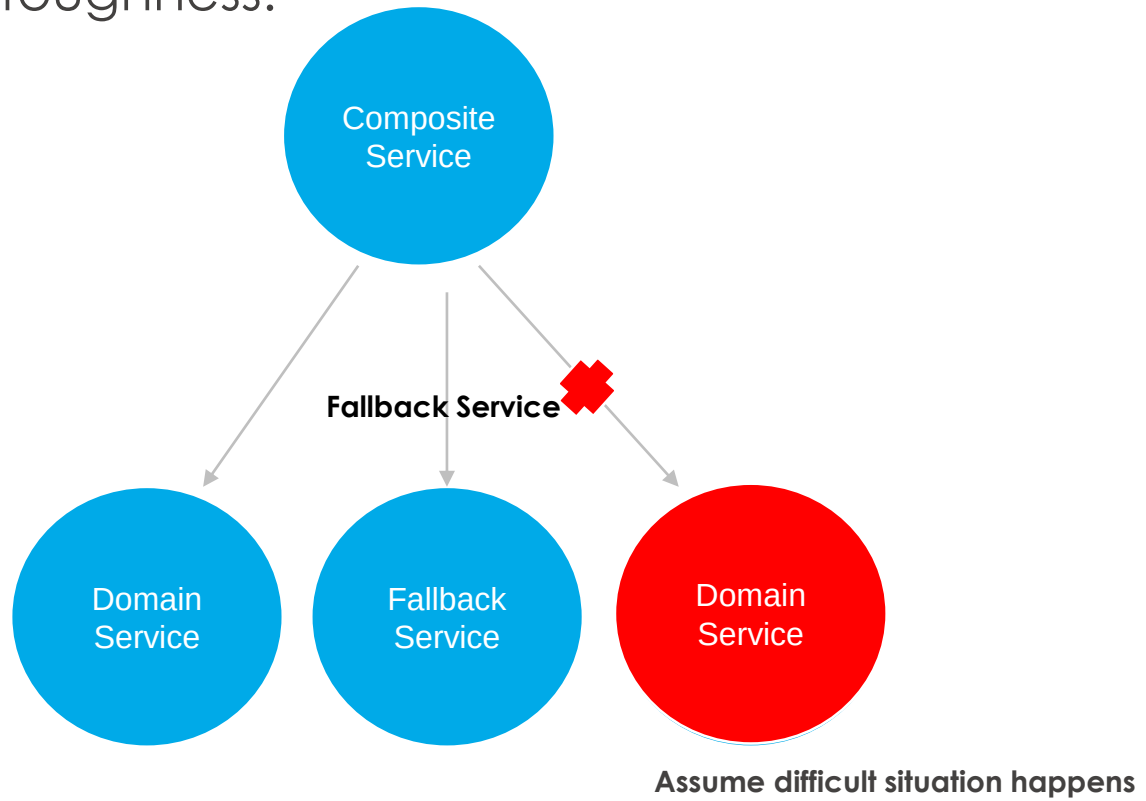


Resiliency - Fallback and Circuit Breaker Mechanism

virtusa

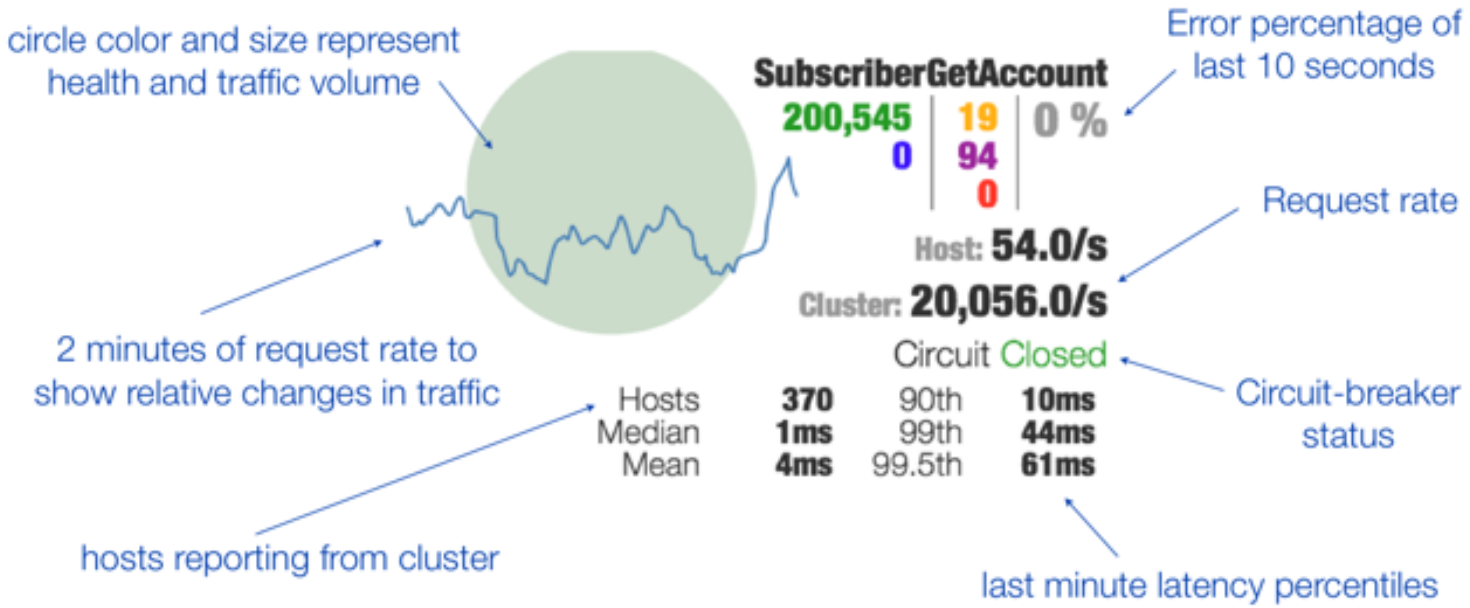
The capacity to recover quickly from difficulties; toughness.

- Timeout
- Throttle
- Circuit Break



Hystrix Dashboards & Monitoring

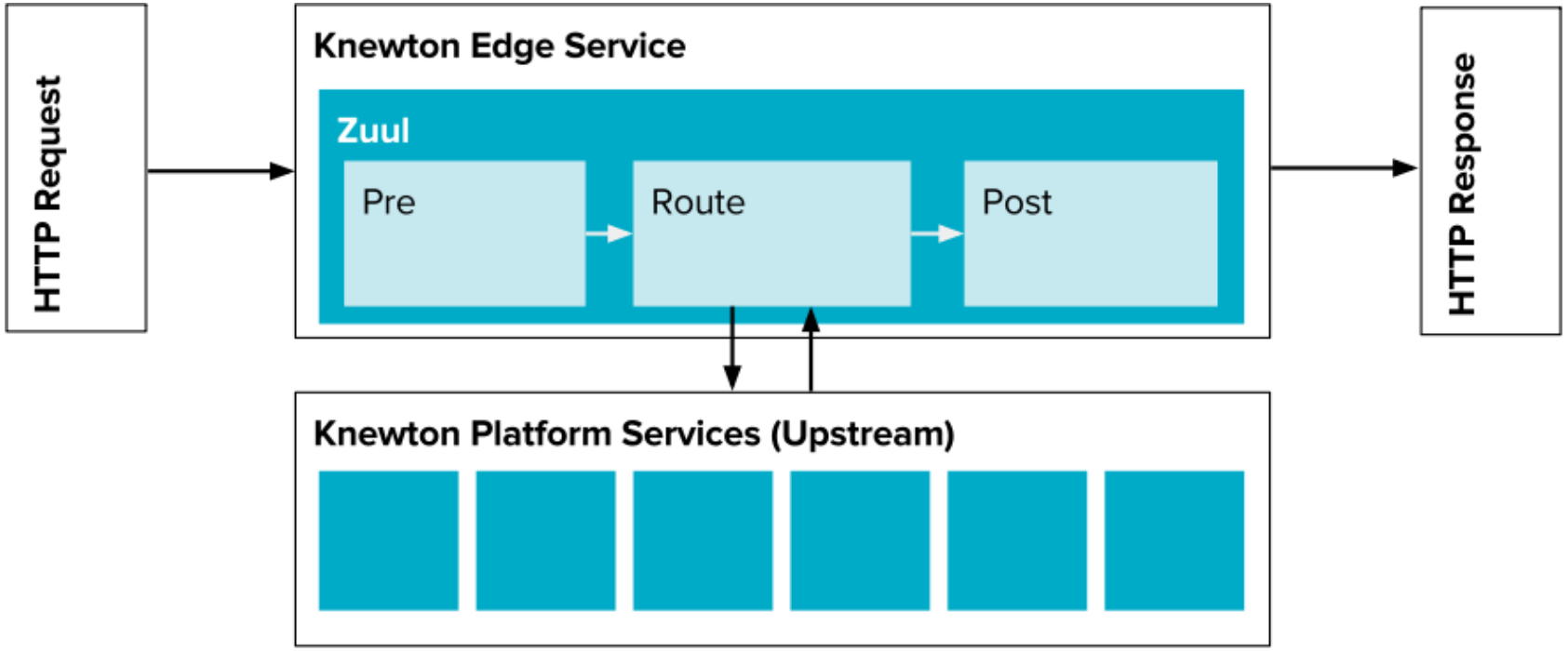
virtusa



The Hystrix Dashboard allows you to monitor a single server

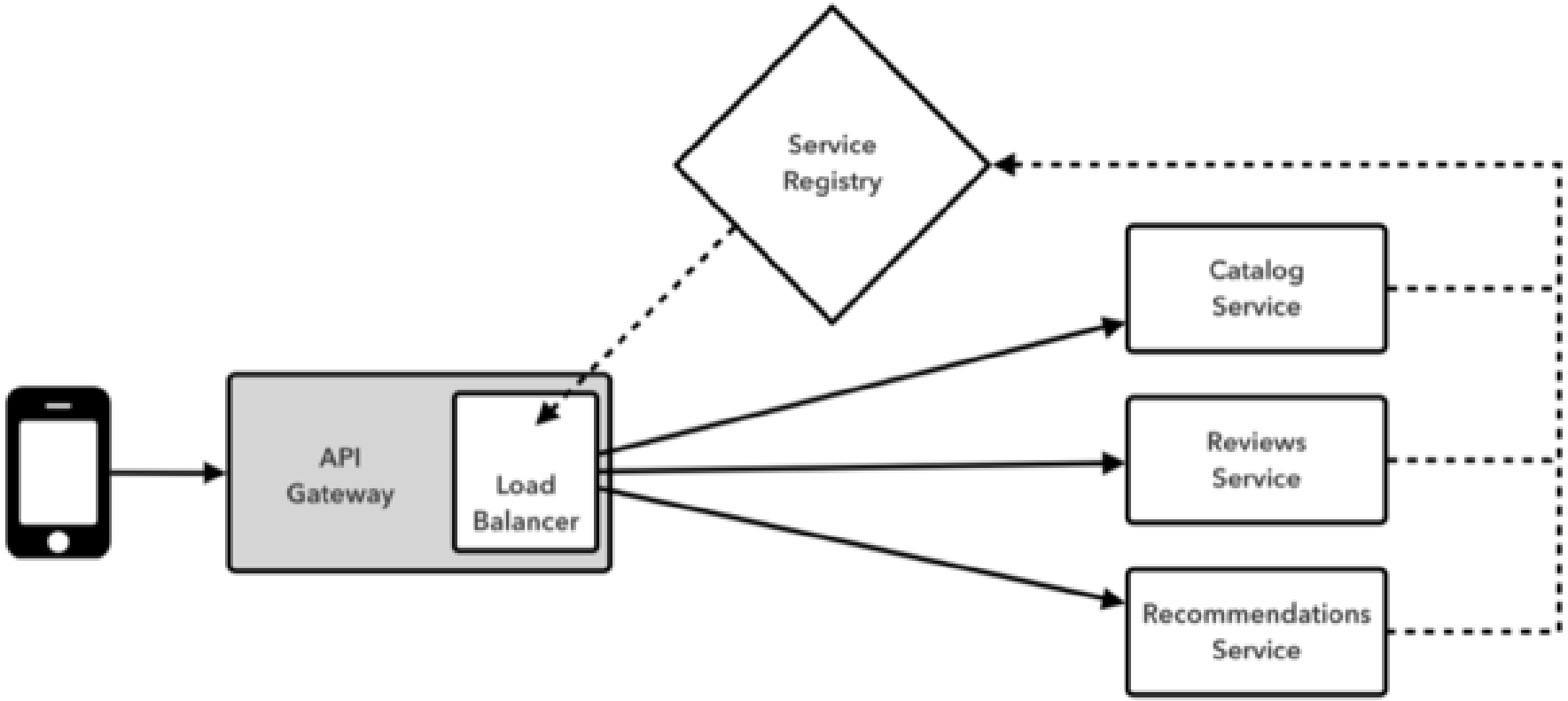
Routing and Filtering

virtusa



API Gateway Pattern

virtusa



Spring Cloud, Netflix Frameworks, ELK

virtusa

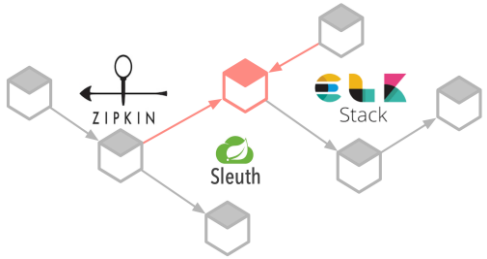
Operations Component	Netflix, Spring, ELK
Service Discovery server	Netflix Eureka
Dynamic Routing and Load Balancer	Netflix Ribbon
Circuit Breaker	Netflix Hystrix
Monitoring	Netflix Hystrix dashboard and Turbine
Edge Server	Netflix Zuul
Central Configuration server	Spring Cloud Config Server
OAuth 2.0 protected API's	Spring Cloud + Spring Security OAuth2
Centralised log analyses	Logstash, Elasticsearch, Kibana (ELK)

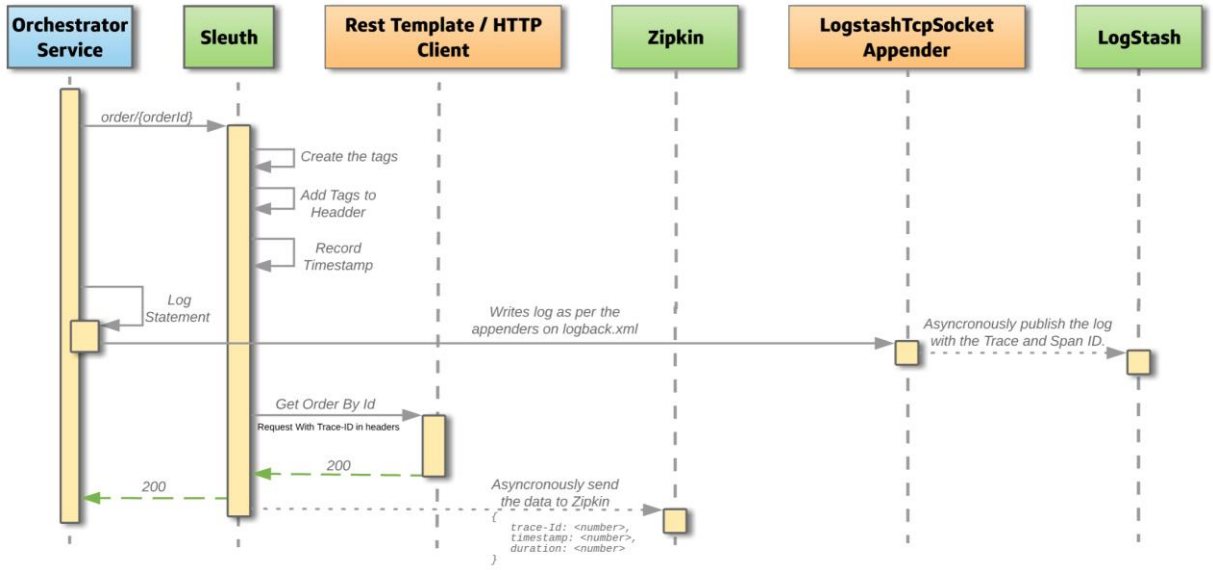
Service Time Tracing

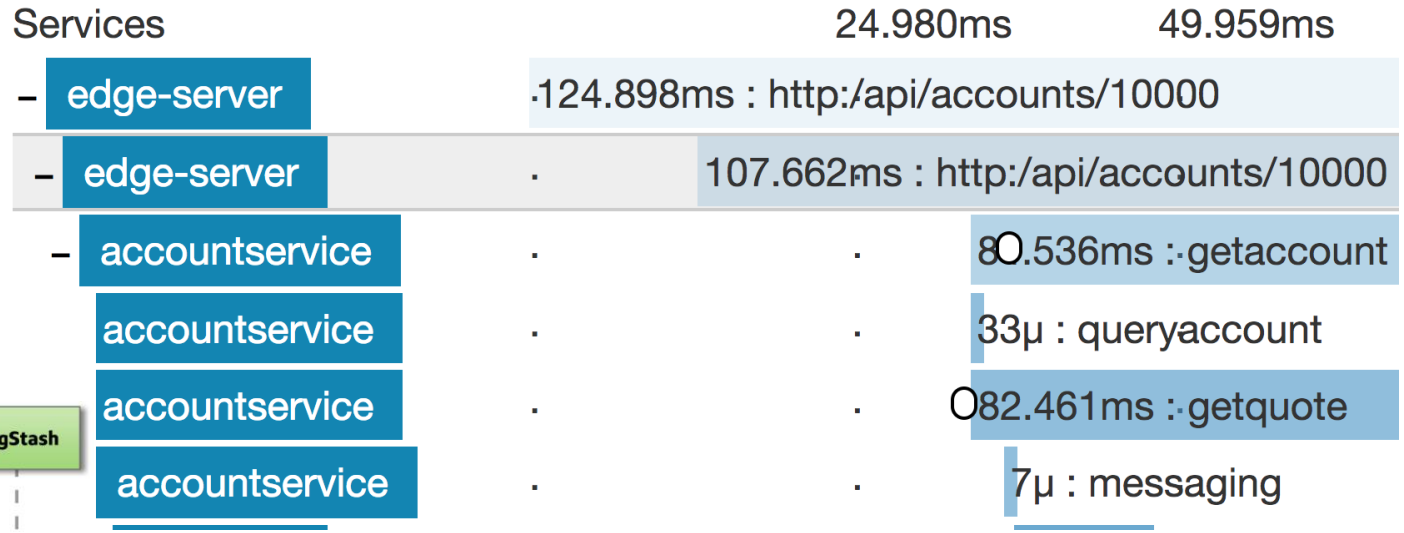
virtusa

Zipkin: A Java-based distributed tracing application that helps gather timing data for every request propagated between independent services

Spring Cloud Sleuth: lets you track the progress of subsequent microservices by adding trace and span id's on the appropriate HTTP request headers. The library is based on the MDC (Mapped Diagnostic Context) concept, where you can easily extract values put to context and display them in the logs.







How do they work together ?

Based on the below diagram (Image A), when the Orchestrator Service makes a HTTP call on the service `*/order/{orderId}*`, the call is intercepted by Sleuth and it adds the necessary tags to the request headers. After the Orchestrator Service receives the HTTP response, the data is sent asynchronously to Zipkin to prevent delays or failures relating to the tracing system from delaying or breaking the flow.

Microservices reference architecture

virtusa

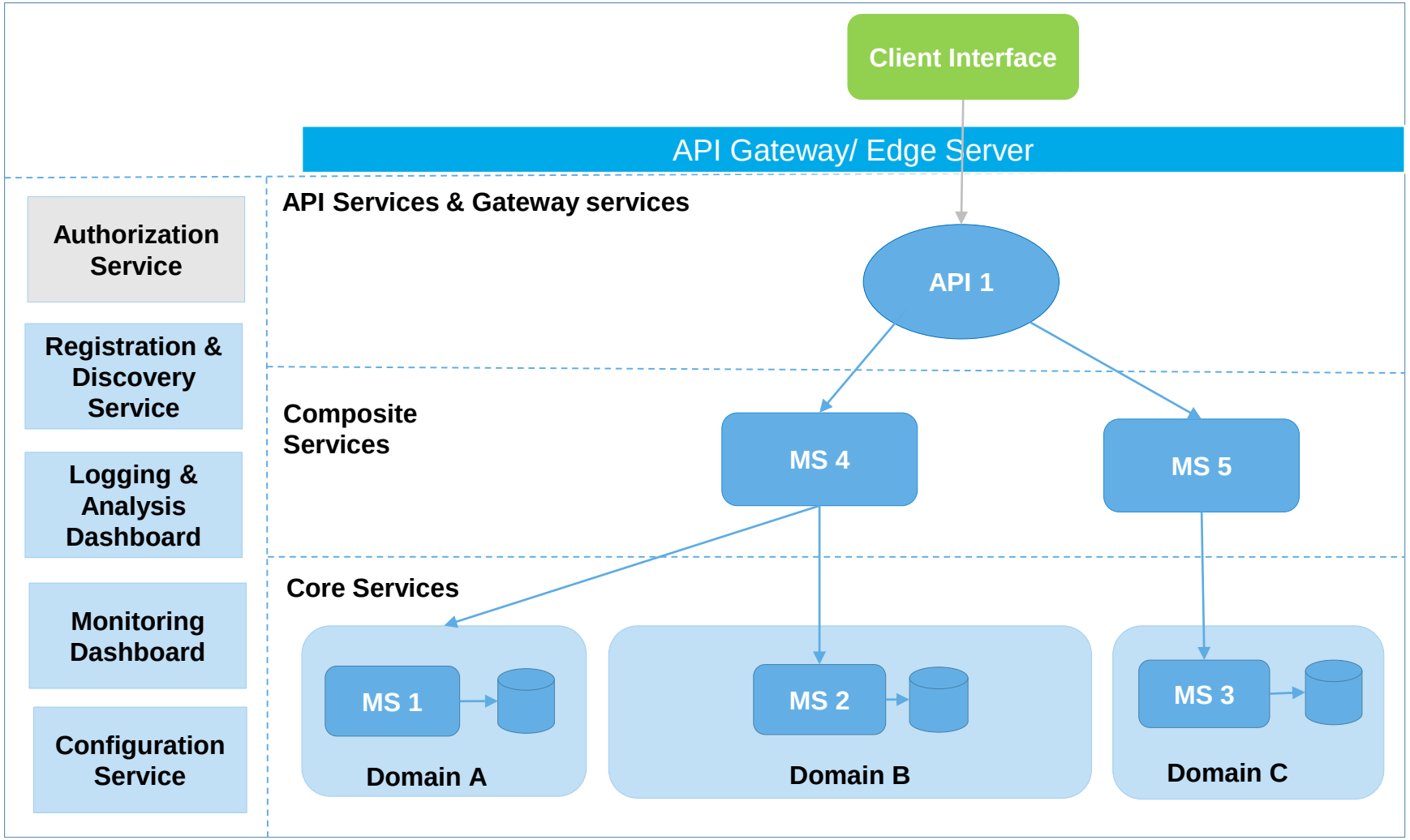
Microservices implementation with:

Vertical Services:
- Core Services
- Composite Services
- API Services

Horizontal Services:
- With domain

Supporting Services:
- Authorization Server
- Service Discovery
- Logging Analysis Dashboard
- Monitoring
- Configuration Server

Technologies:
- Java, Maven
- Spring Cloud
- Spring BOOT
- Netflix Frameworks
- ZipKin and more



A Basic Microservice Implementation

virtusa

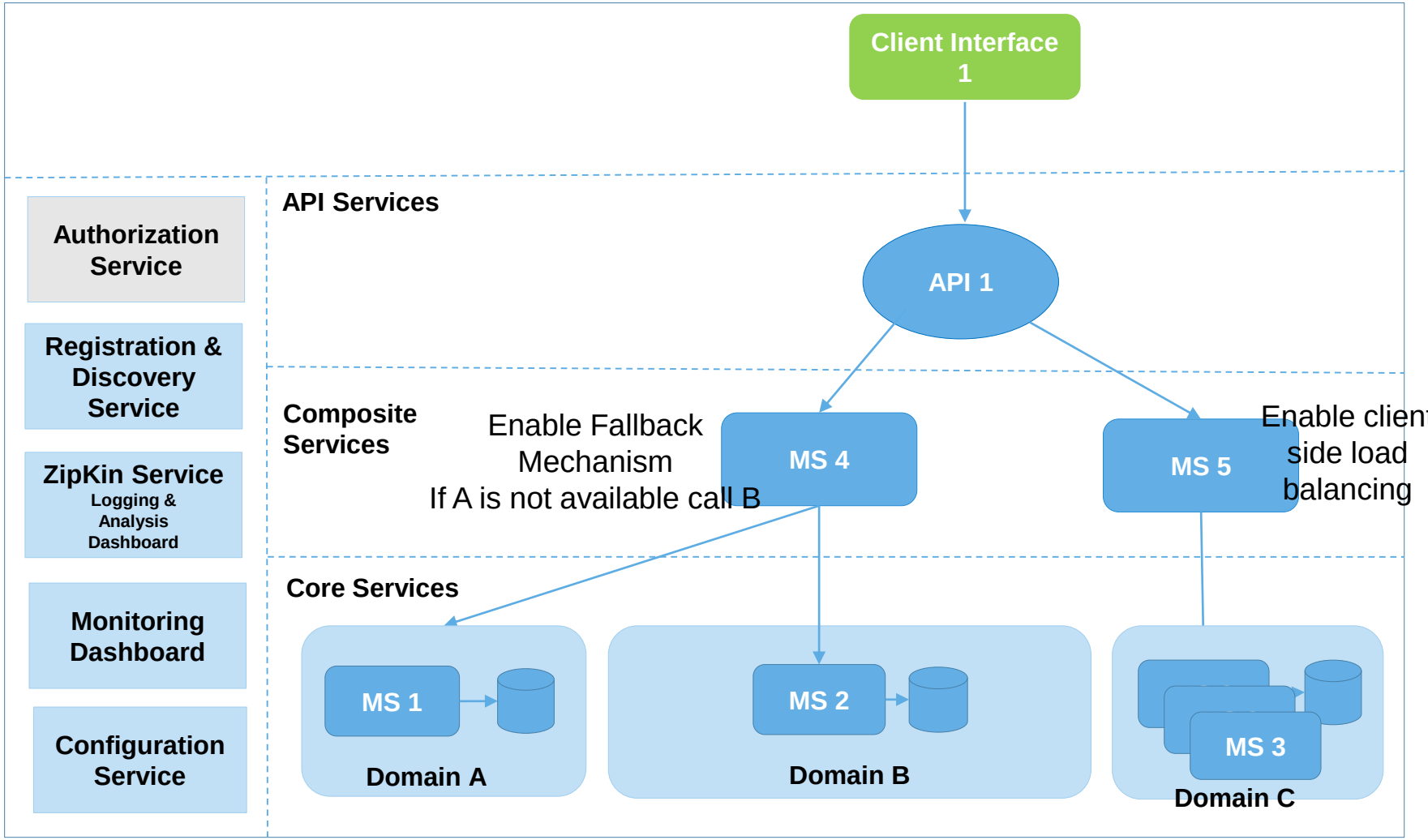
Microservices implementation with:

Vertical Services:
- Core Services
- Composite Services
- API Services

Horizontal Services:
- With domain

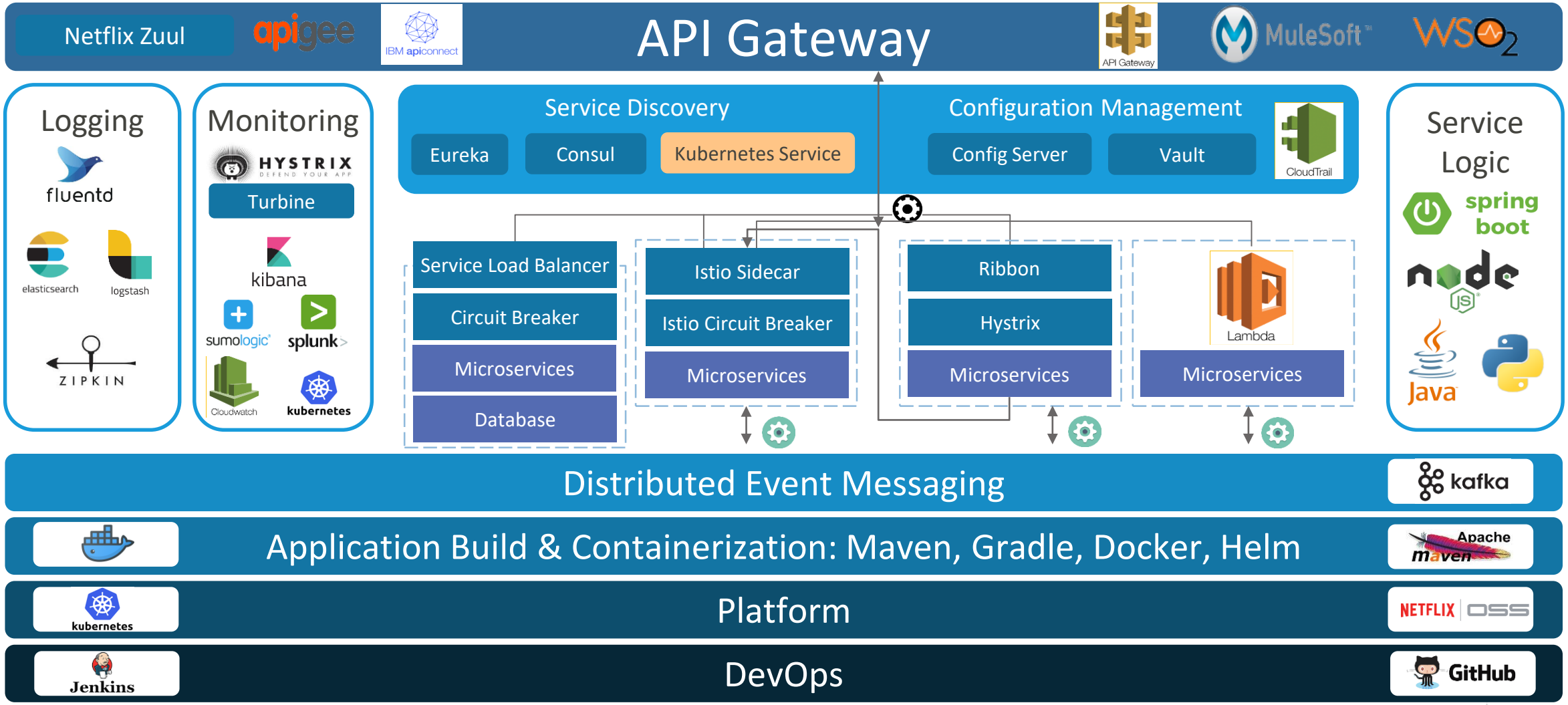
Supporting Services:
- Authorization Server
- Service Discovery
- Logging Analysis Dashboard
- Monitoring
- Configuration Server

Technologies:
- Java, Maven
- Spring Cloud
- Spring BOOT
- Netflix Frameworks
- ZipKin and more



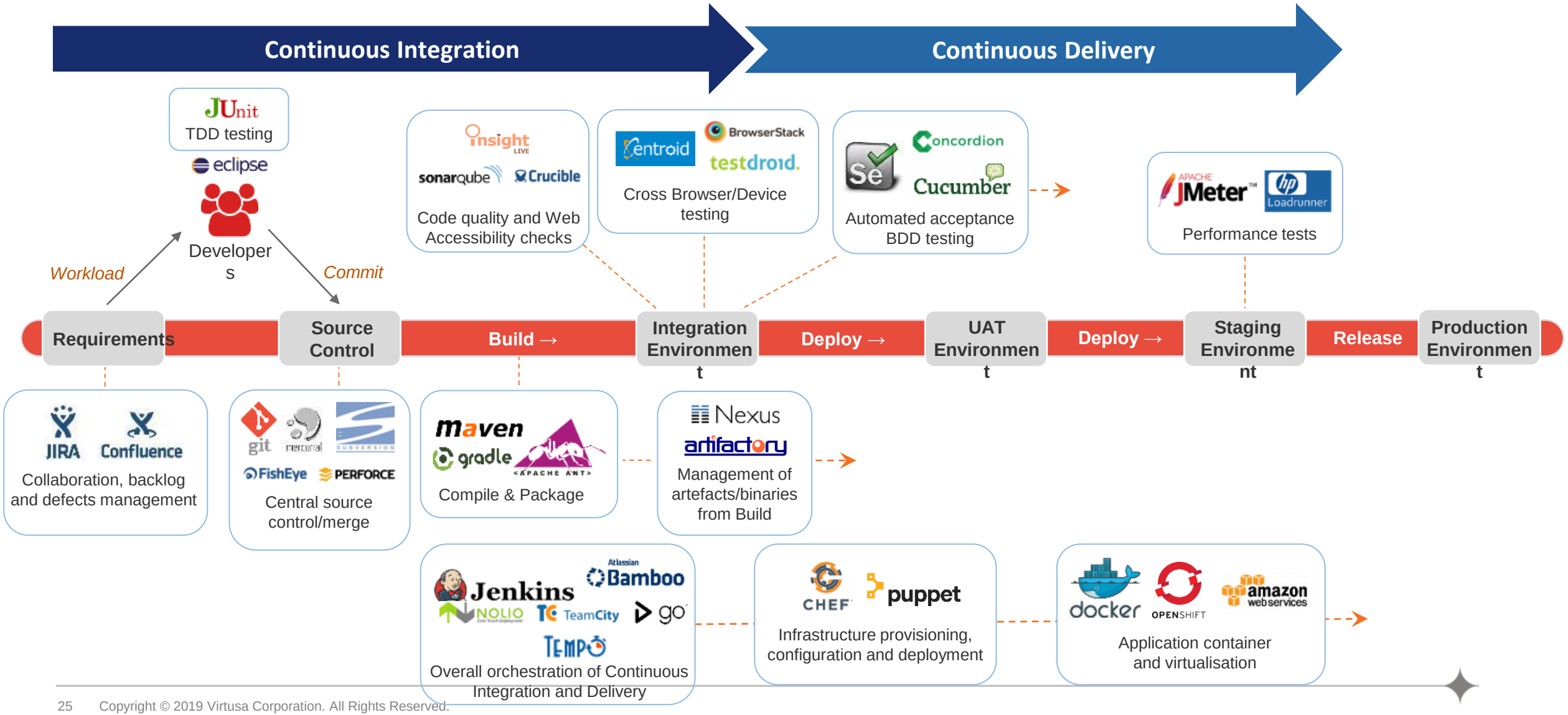
Microservices reference architecture

virtusa



Microservices depend on DevOps automation tools for configuration and deployment

virtusa



virtusa

Thank You !!!