

# Application Delivery Fundamentals 2.0 B: Java

Micro Service



High performance. Delivered.



# Micro Service

---

## Goals

- Independent deployment of components
- Independent scaling of components
- Independent implementation stacks for each component
- Easy self-serve deployments of components
- Repeatable deployments of components (external configuration management)
- Deployments without service interruptions



# Micro Service

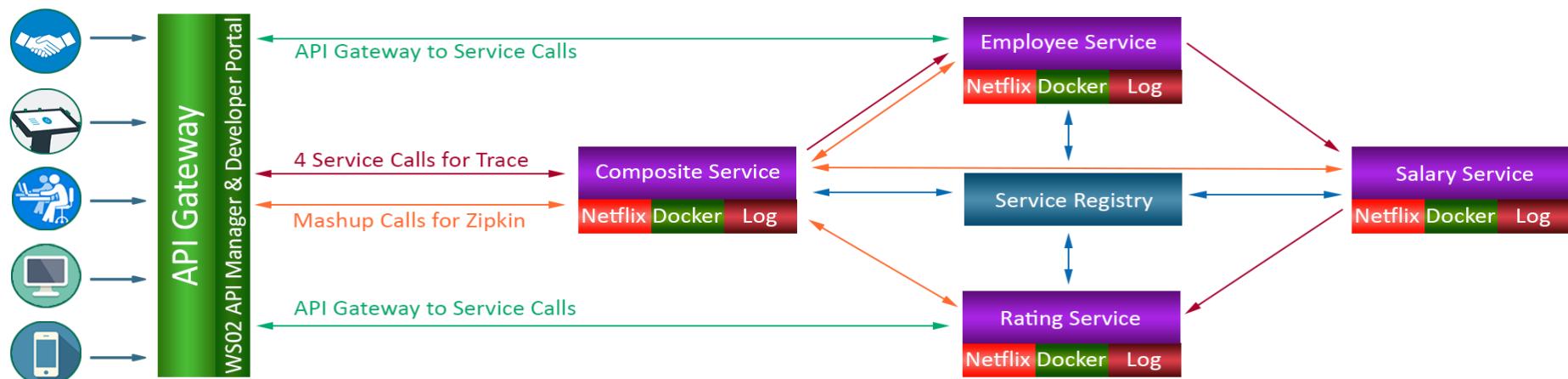
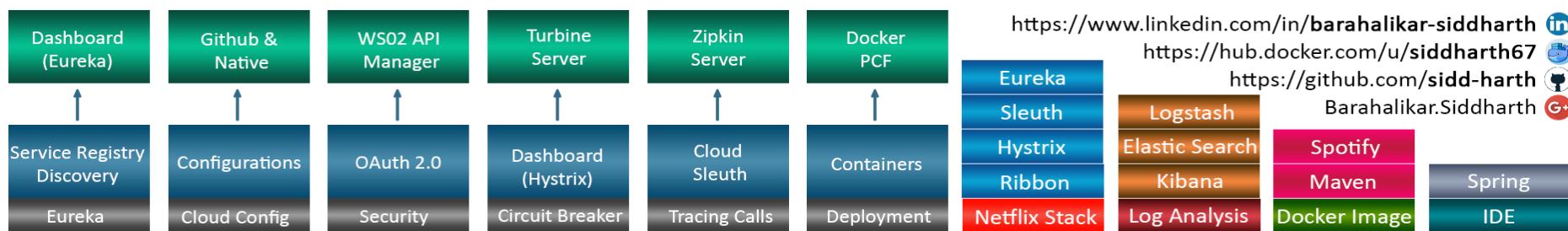
---

## Goals

- Protection of system availability from individual Instance failure
- Automatic replacement of component instances when they fail (self-healing)
- Easy scaling of components by adjusting a simple parameter value
- Canary testing (A **canary deployment / canary test** allows you to gradually release new features to a subset of your users while still serving your current branch to the rest of your users)
- "Red/black" or "blue/green" deployments(Instant reversal of new revision deployments)



## Spring Boot Microservices Managed via Spring Cloud, Netflix OSS, ELK Stack, Docker & WSO2 APIM



Salary - Lists Salary Details (ID)
Ratings - Lists Rating Details (ID)
Employee - Lists Employee Details (ID)
Composite - Aggregates all 3 Service Details
Basic Information on Services

WSO2 - 9443	Employee - 1111	Zipkin - 9411
Logstash - 5000	Salary - 2222	Config - 8888
Elastic - 9200	Rating - 3333	Eureka - 8761
Kibana - 5601	Composite - 1234	Hystrix - 8989
Initial Port Numbers (will change on Docker deployment)		

ELK 5.1.1   Maven 4.0.0   Spotify 0.4.10
WSO2 2.0.0   Docker 17.07.0-ce 17.09.0-ce
Spring Cloud Camden.SR6   3.9.0.RELEASE
Logback-encoder 4.6   Zipkin 1.18.0 runtime
Hystrix 1.3.0   Actuator 1.5.6   1.5.7.RELEASE

# Monolithic Architecture



- Monolith means composed all in one piece.
- The Monolithic application describes a single-tiered software application in which different components combined into a single program from a single platform.



# Monolithic Architecture

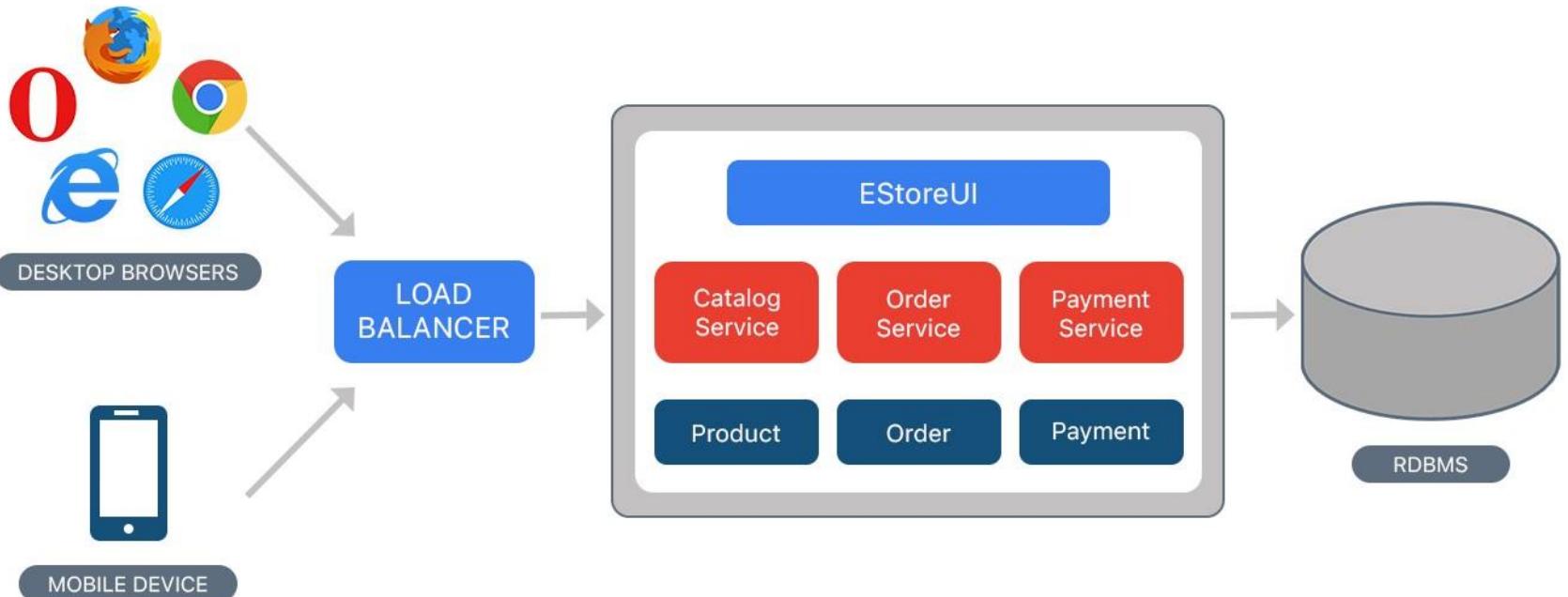
---

Components can be:

- Authorization — responsible for authorizing a user
- Presentation — responsible for handling HTTP requests and responding with either HTML or JSON/XML (for web services APIs).
- Business logic — the application's business logic.
- Database layer — data access objects responsible for accessing the database.
- Application integration — integration with other services (e.g. via messaging or REST API). Or integration with any other Data sources.
- Notification module — responsible for sending email notifications whenever needed.



# Example of Monolithic Approach





## Drawbacks

---

- Maintenance — If Application is too large and complex to understand entirely, it is challenging to make changes fast and correctly.
- The size of the application can slow down the start-up time.
- You must redeploy the entire application on each update.
- Monolithic applications can also be challenging to scale when different modules have conflicting resource requirements.
- Reliability — Bug in any module (e.g. memory leak) can potentially bring down the entire process.
- Moreover, since all instances of the application are identical, that bug impact the availability of the entire application



## Drawbacks

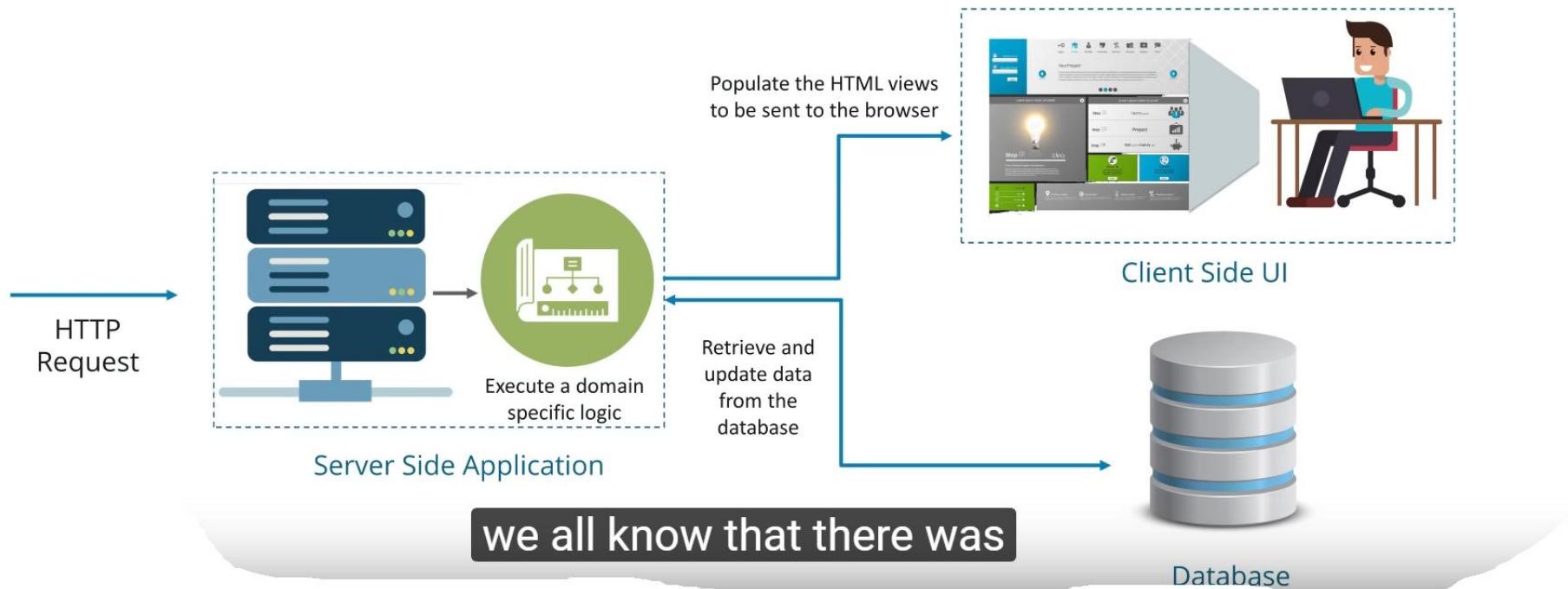
---

- Regardless of how easy the initial stages may seem, Monolithic applications have difficulty to adopting new and advance technologies.
- Since changes in languages or frameworks affect an entire application.
- It requires efforts to thoroughly work with the app details.
- Hence it is costly considering both time and efforts.



# Monolithic Architecture

Monolithic Architecture is like a big container wherein all the software components of an application are assembled together and tightly packaged





# Monolithic Architecture

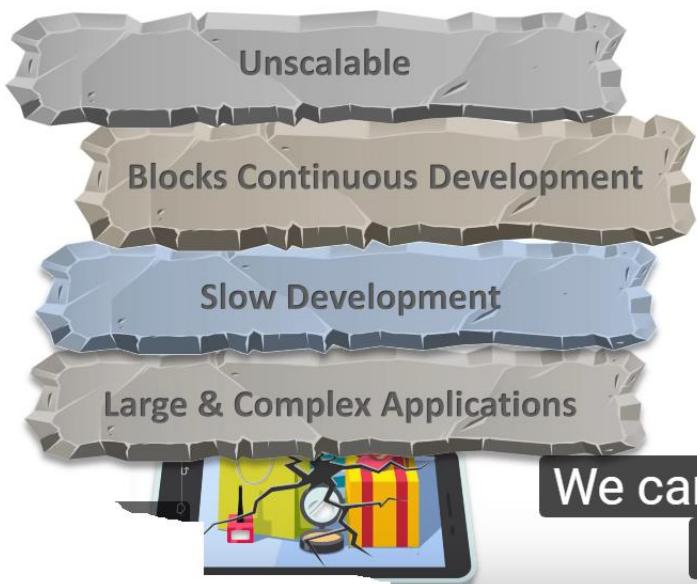


Slow Development

Large & Complex Applications



# Monolithic Architecture

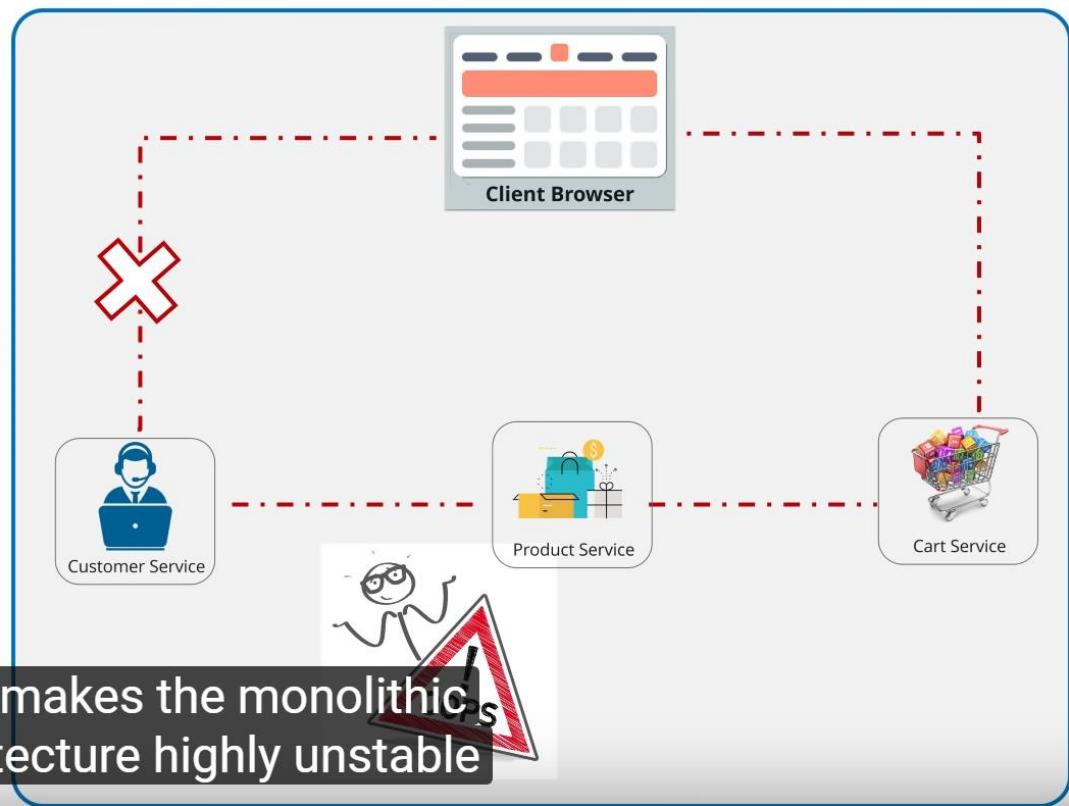


We cannot scale each component  
Independence, right?



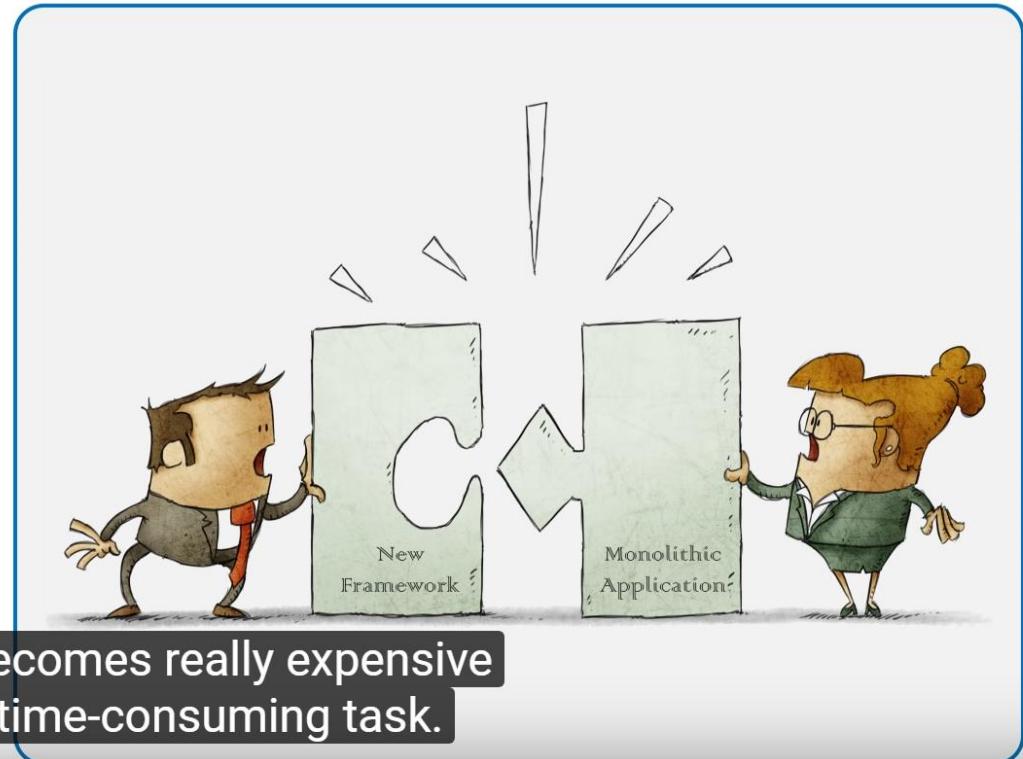


# Monolithic Architecture





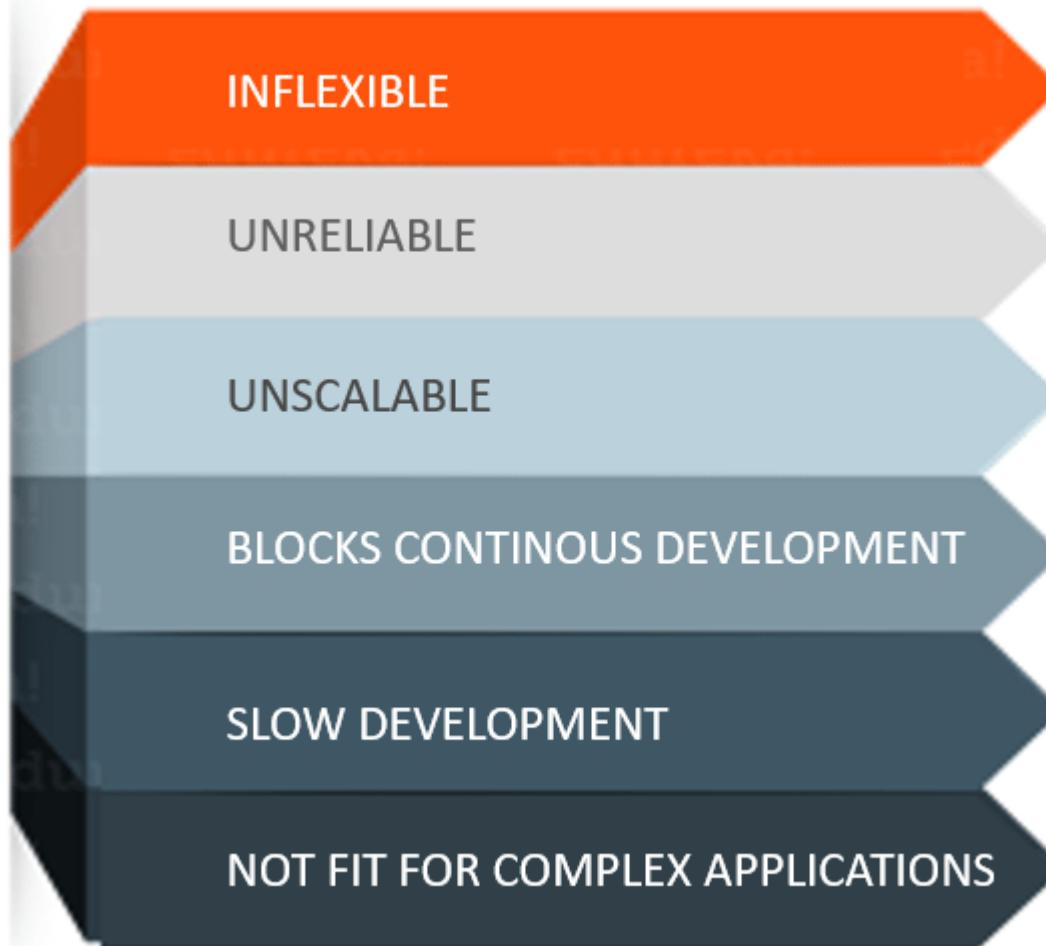
# Monolithic Architecture



So it becomes really expensive  
and time-consuming task.



# Monolithic Architecture



# Micro Service

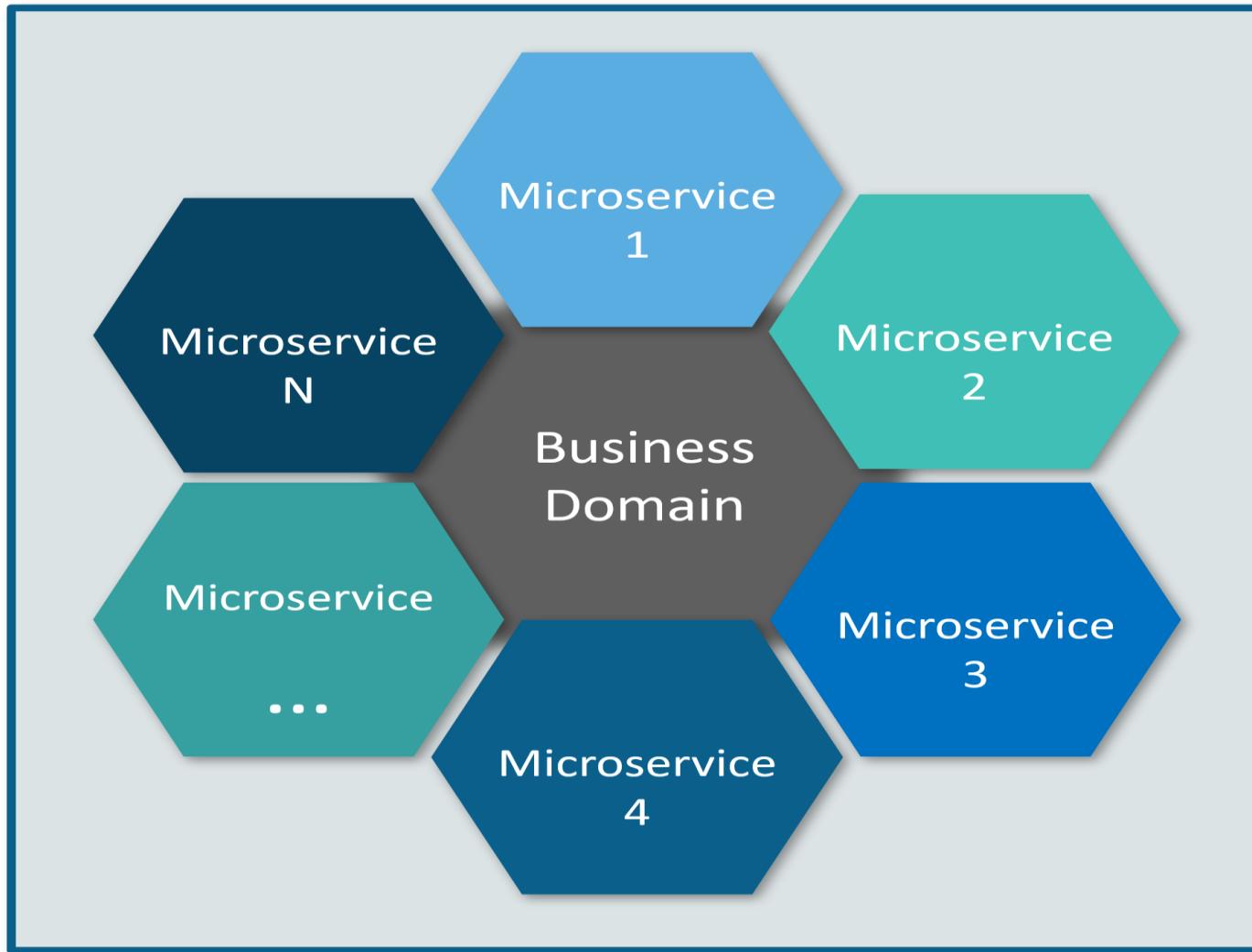


*Small autonomous services that work together - Sam Newman*

*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*



# Microservice



# Micro Service



*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....contd*



## Micro Service

*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....contd*

# Micro Service

---



*These services are built around  
business capabilities and  
independently deployable by fully  
automated deployment  
machinery...contd*



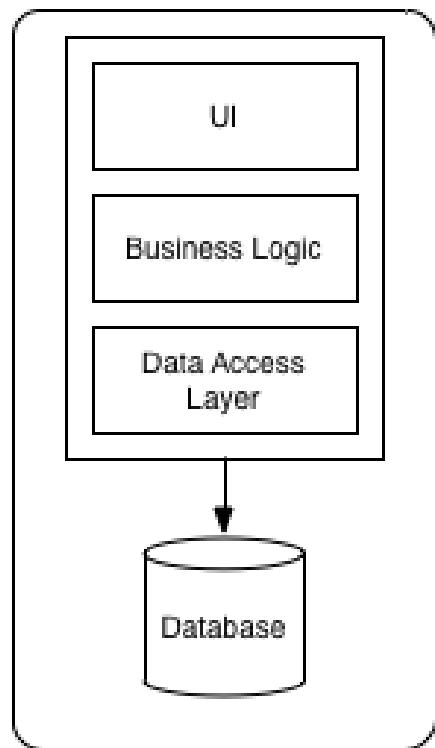
# Micro Service



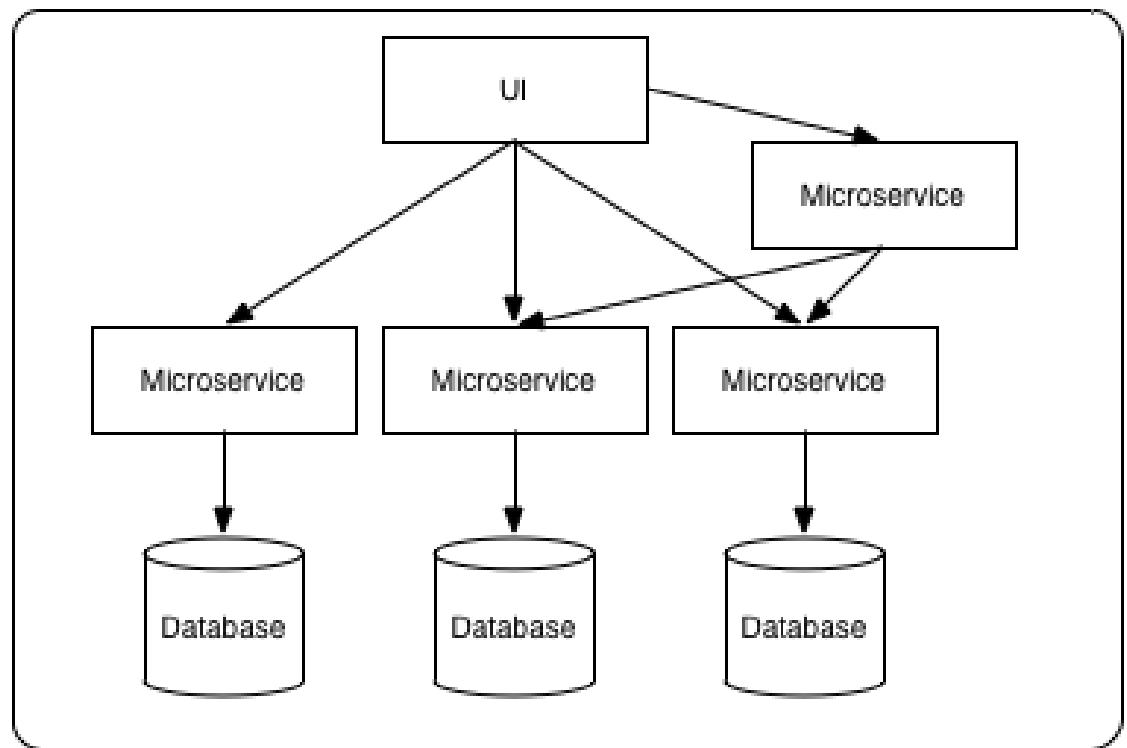
*There is a bare minimum of centralized management of these services, which may be written in different programming languages and use different data storage technologies - James Lewis and Martin Fowler*



# Microservice



Monolithic Architecture

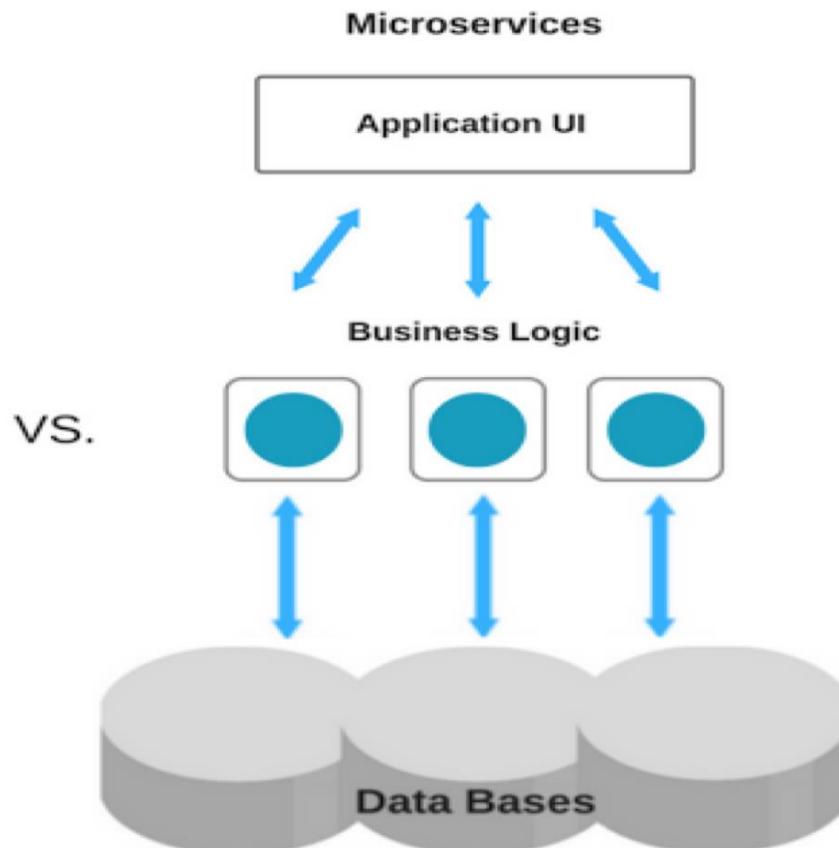
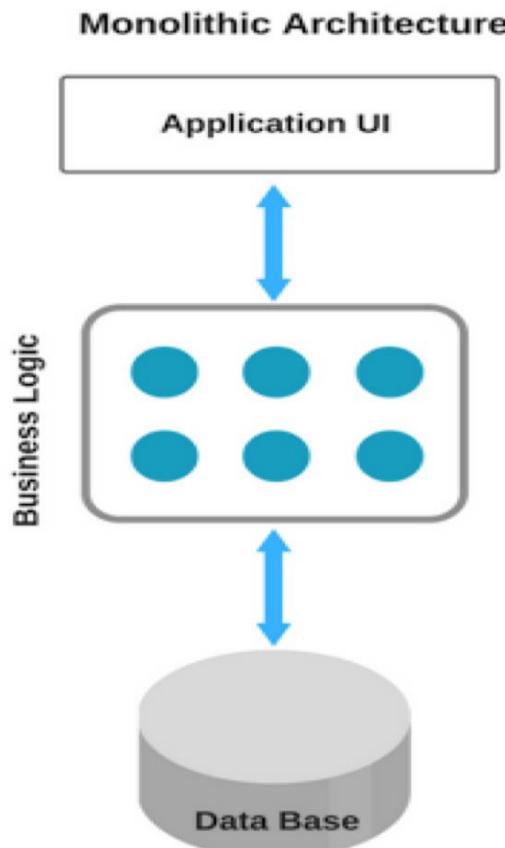


Microservices Architecture

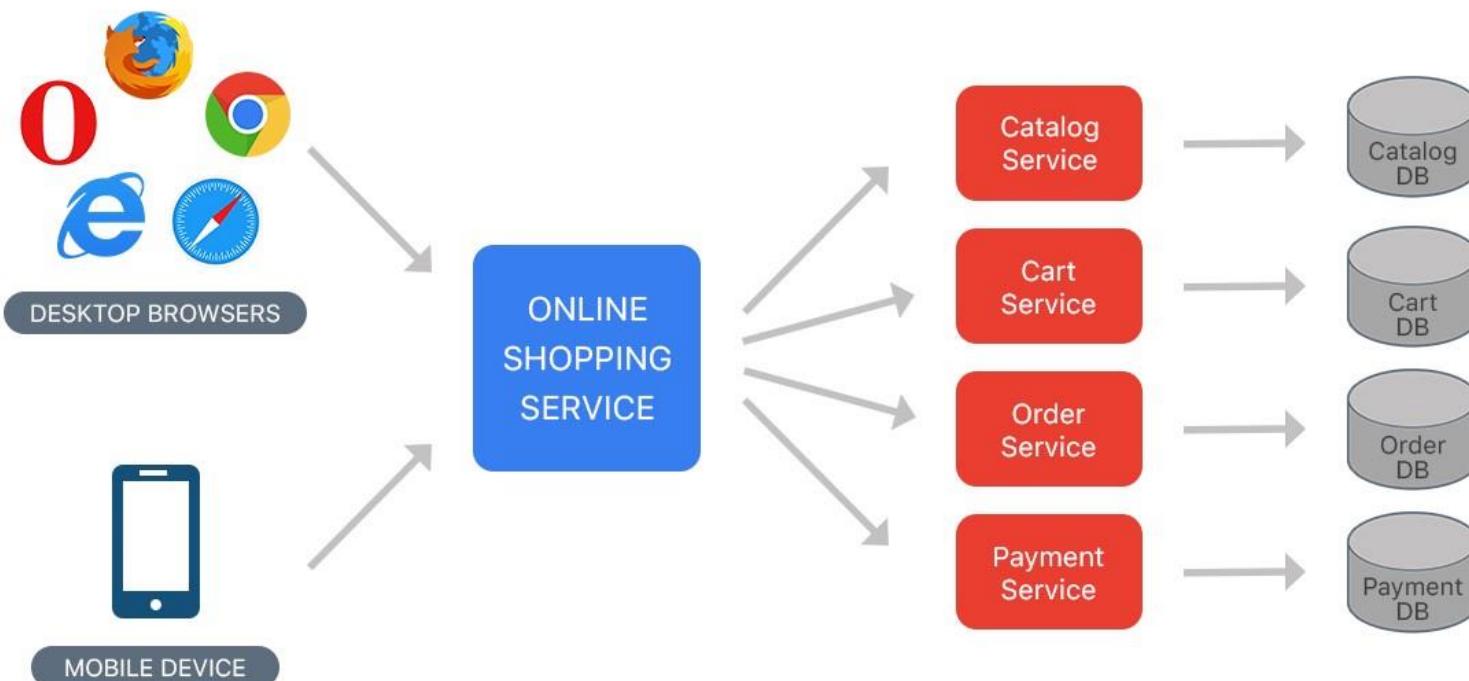
# MicroService



The difference between the monolithic and microservices architecture

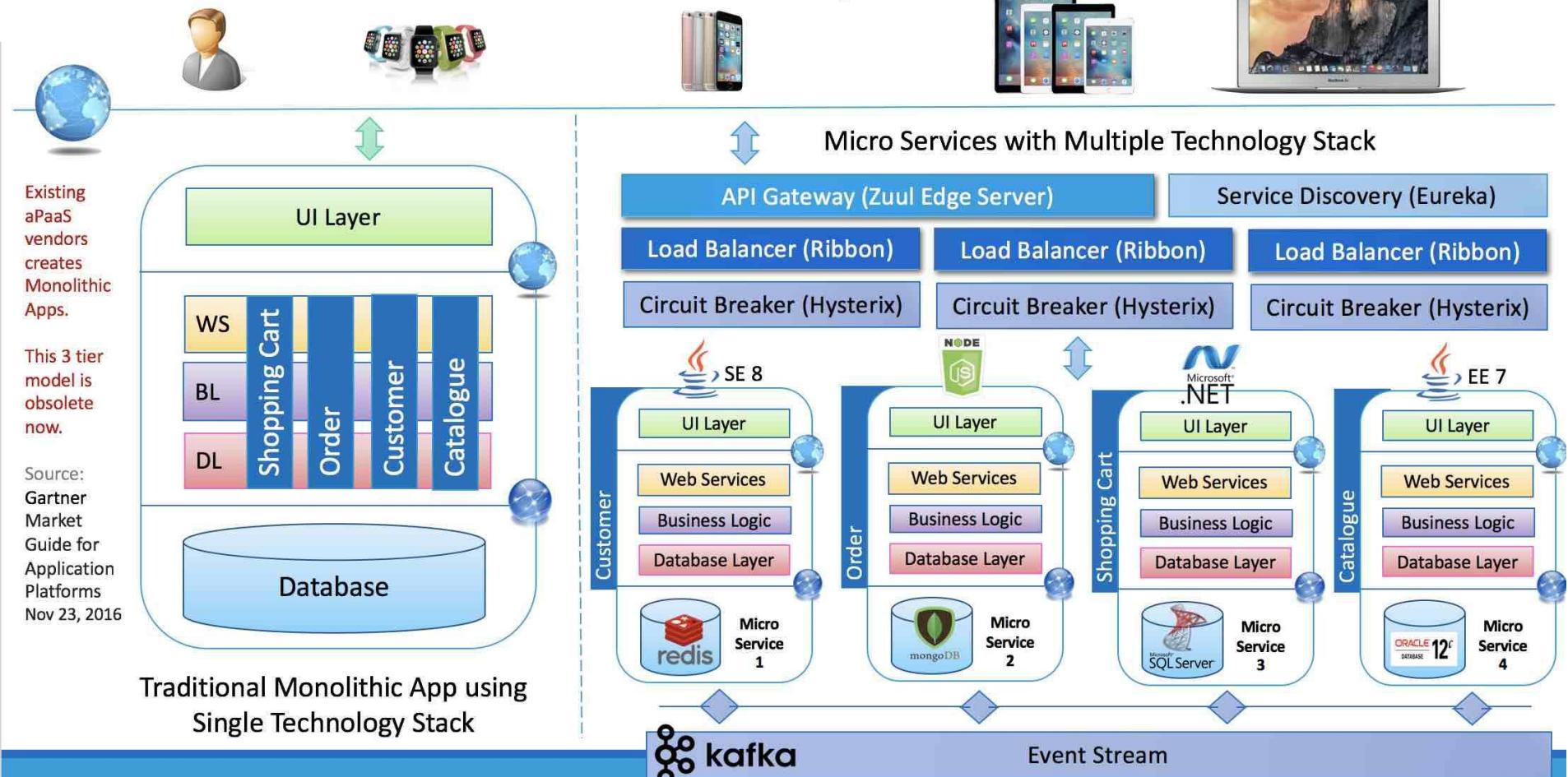


# Micro Service





## Monolithic vs. Micro Services Example

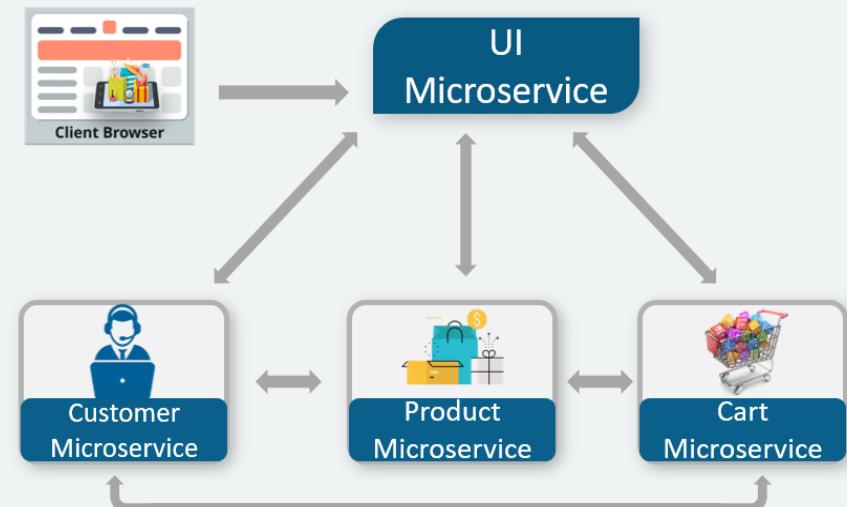




## Monolithic Architecture

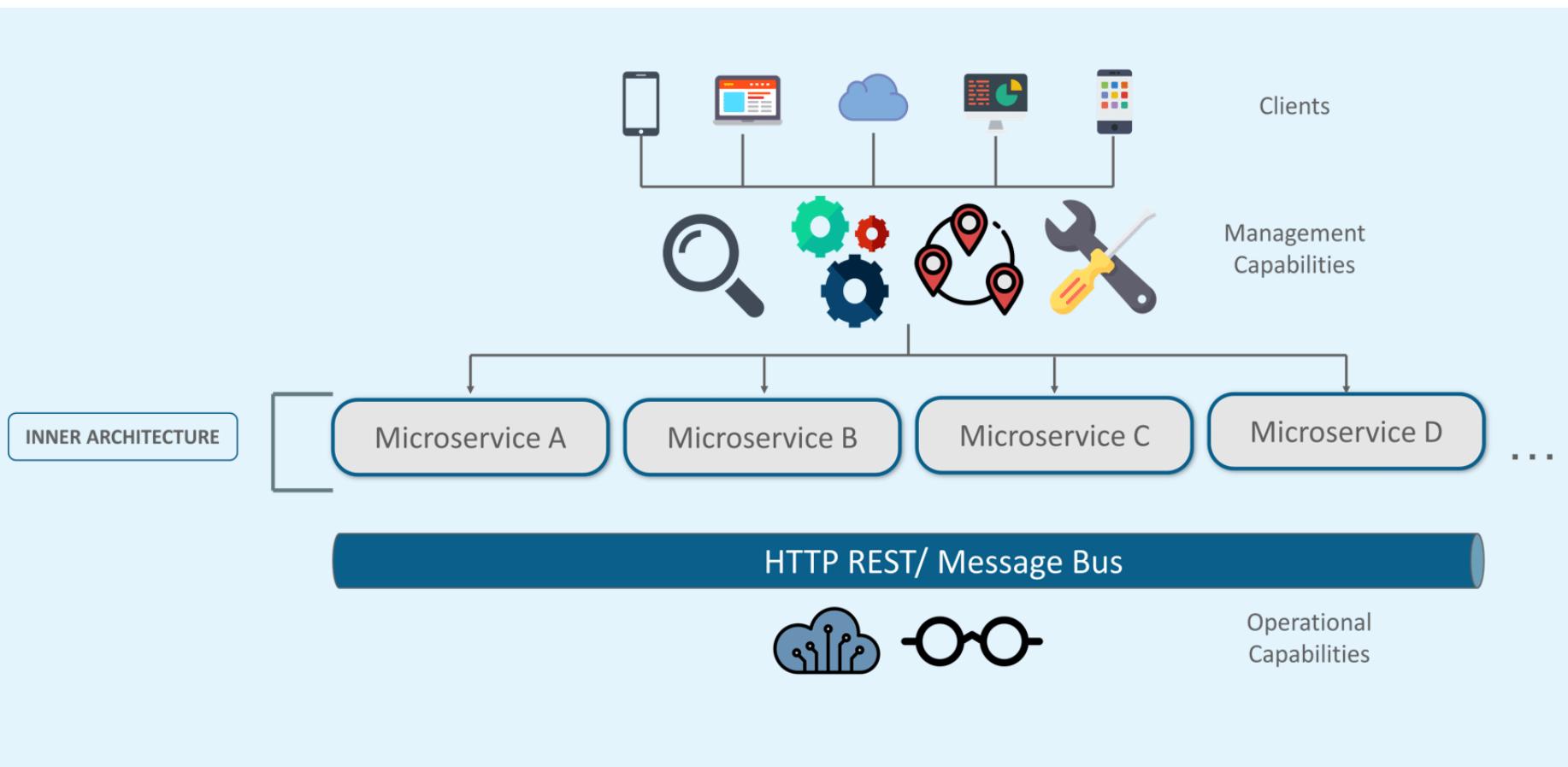


## Microservice Architecture





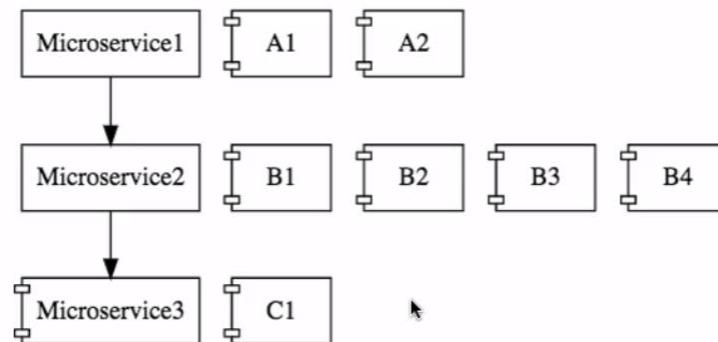
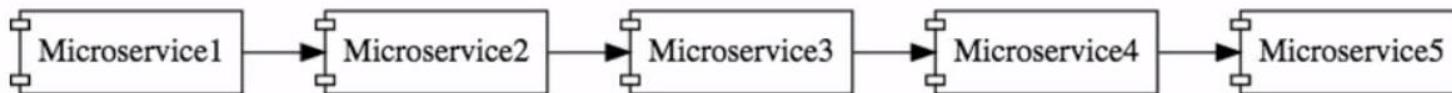
# Microservice Architecture



# Micro Service



- RESTful Web Services
- & Small Well Chosen Deployable Units
- & Cloud Enabled





# What is Microservice

---

- **Microservices** is a variant of the service-oriented architecture (SOA) architectural style that structures an application as a collection of loosely coupled services.



# What is Microservice

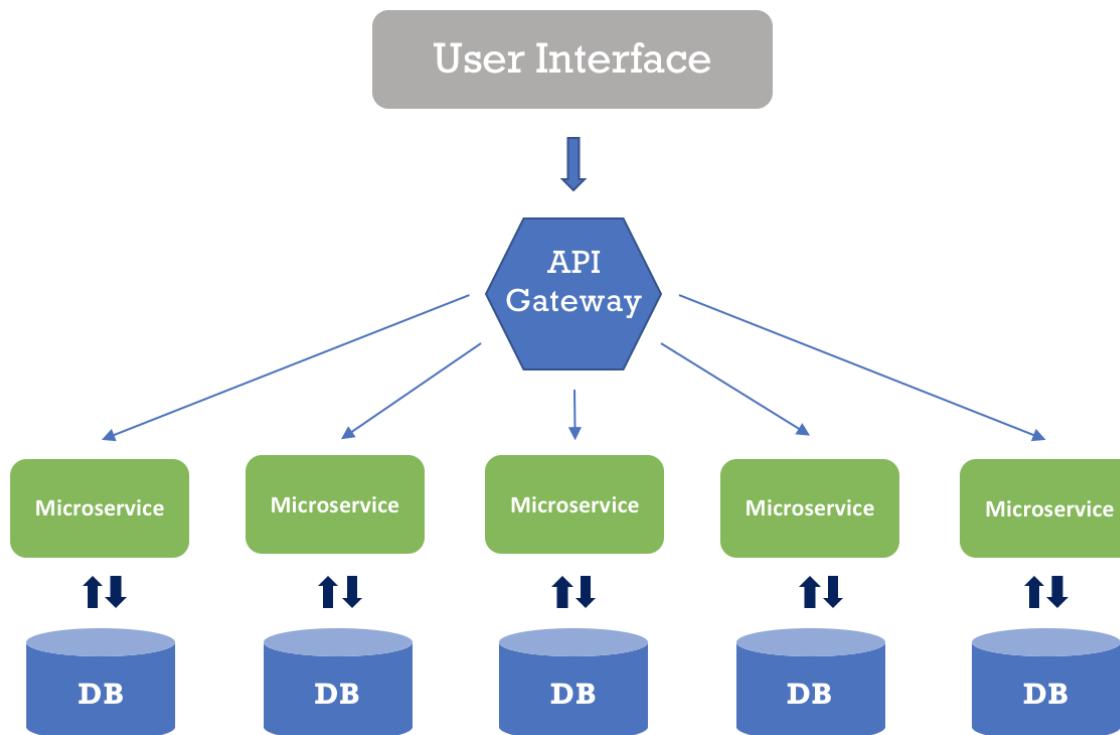
---

- In a microservices architecture, services should be fine-grained and the protocols should be lightweight.
- The benefit of decomposing an application into different smaller services is that it improves modularity and makes the application easier to understand, develop and test.

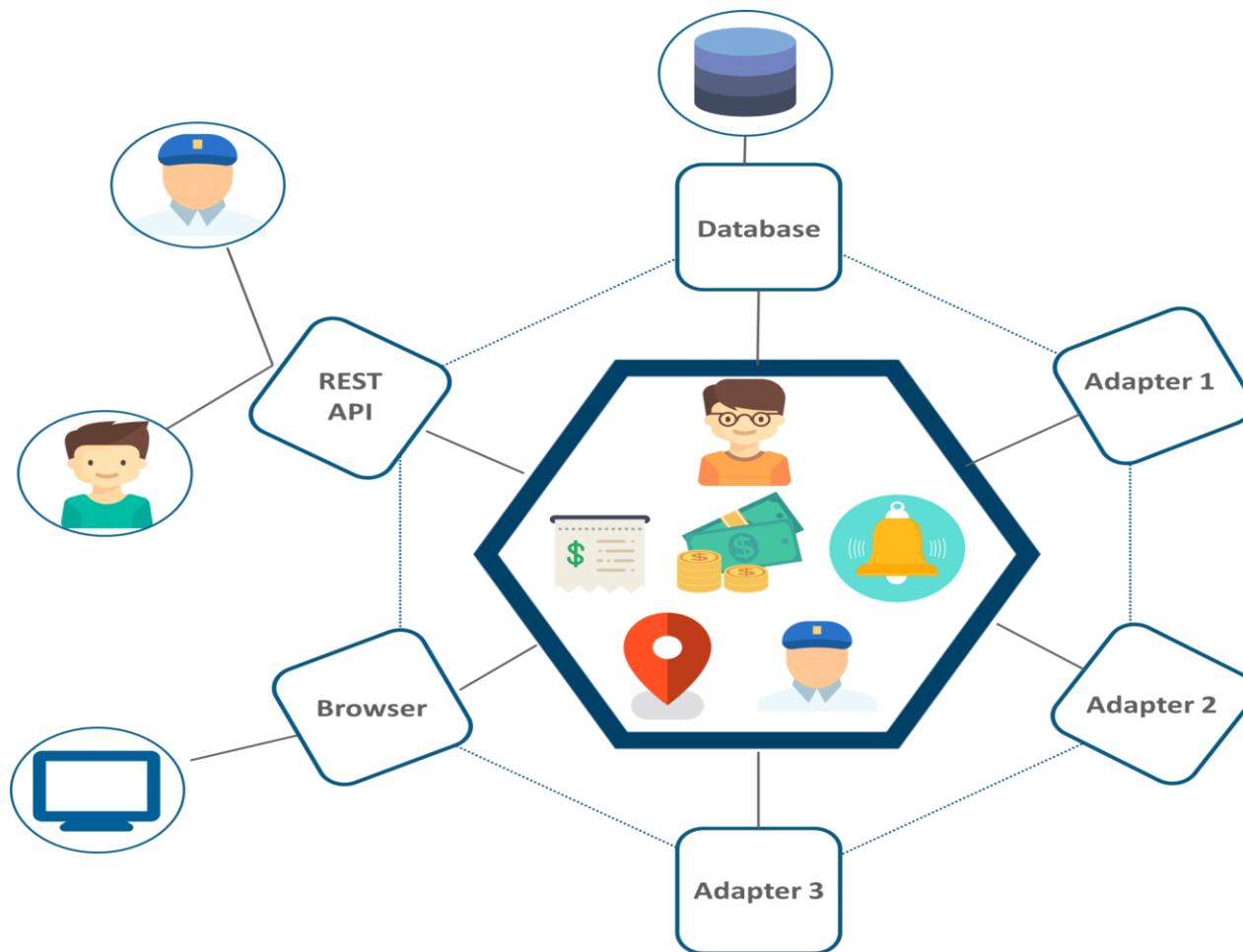


# What is Microservices

- It also parallelizes development by enabling small autonomous teams to develop, deploy and scale their respective services independently.



## UBER's Previous Architecture





## Problem Statement

---

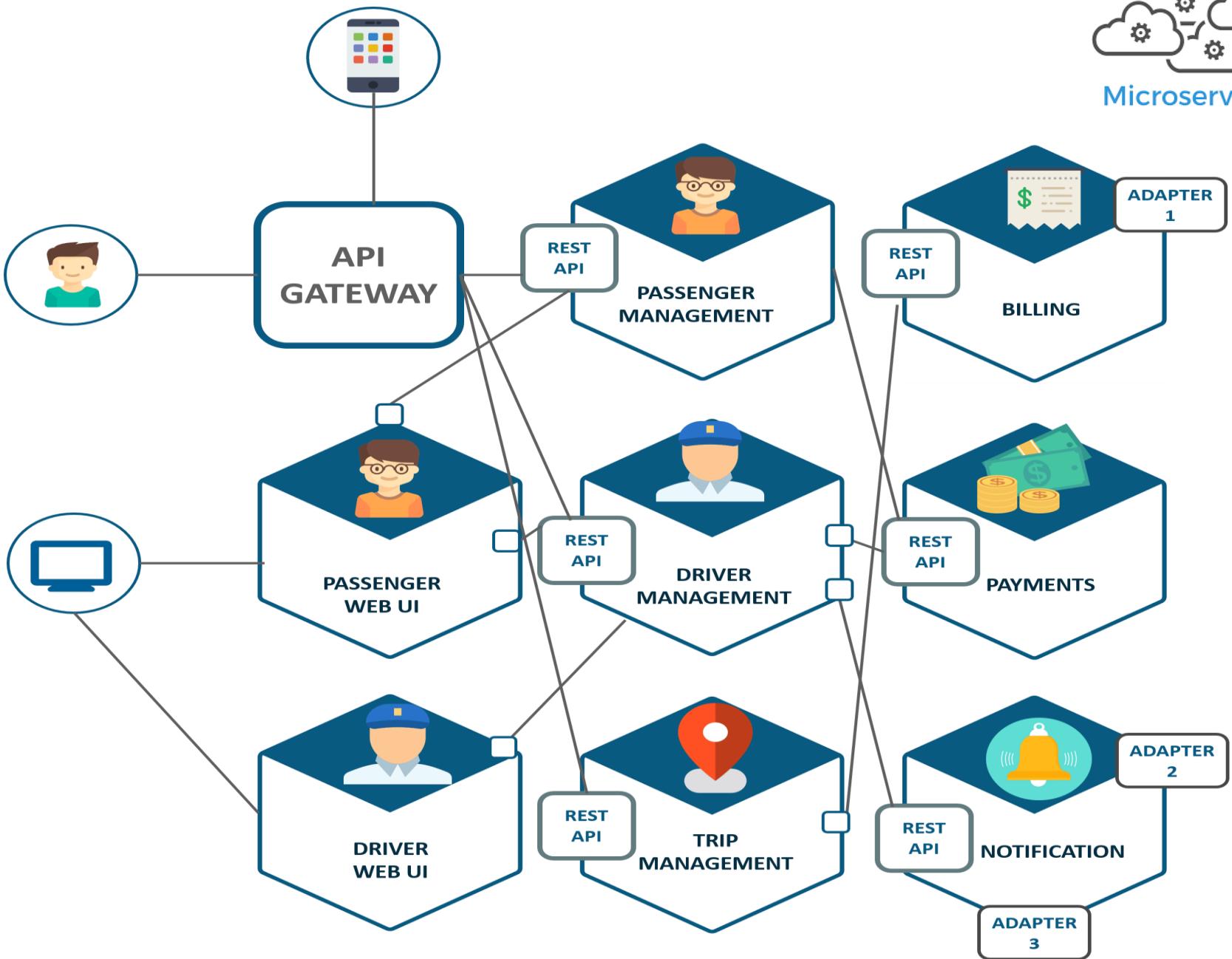
- While UBER started expanding worldwide this kind of framework introduced various challenges. The following are some of the prominent challenges
- All the features had to be re-built, deployed and tested again and again to update a single feature.
- Fixing bugs became extremely difficult in a single repository as developers had to change the code again and again.
- Scaling the features simultaneously with the introduction of new features worldwide was quite tough to be handled together.



## Solution

---

- To avoid such problems UBER decided to change its architecture and follow the other hyper-growth companies like Amazon, Netflix, Twitter and many others.
- Thus, UBER decided to break its monolithic architecture into multiple codebases to form a microservice architecture.



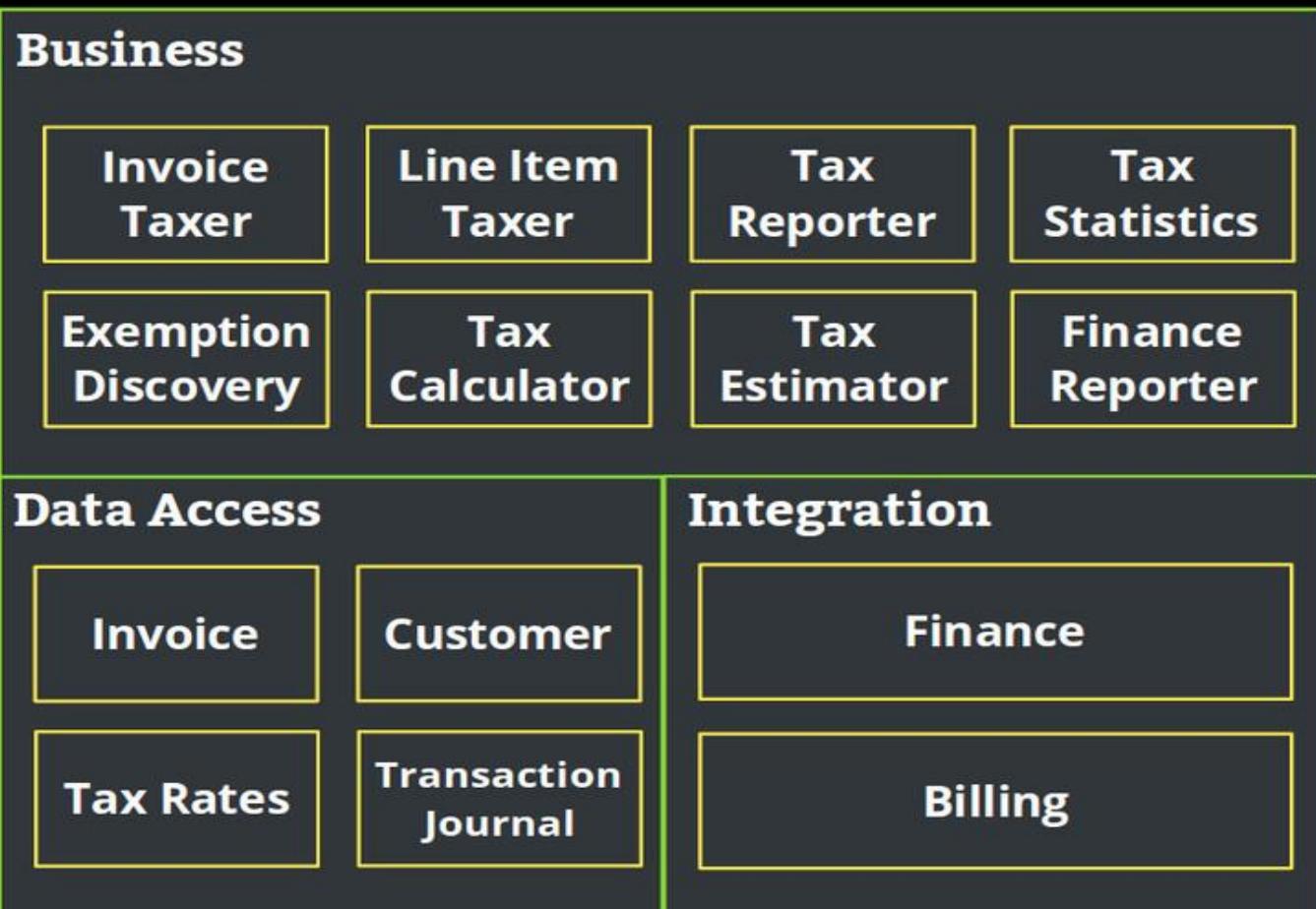


# Existing Scenarios

---

- Netflix has a widespread architecture that has evolved from monolithic to SOA.
- It receives more than *one billion* calls every day, from more than 800 different types of devices, to its streaming-video API.
- Each API call then prompts around five additional calls to the backend service.
- Amazon has also migrated to microservices.
- They get countless calls from a variety of applications—including applications that manage the web service API as well as the website itself—which would have been simply impossible for their old, two-tiered architecture to handle.
- The auction site eBay is yet another example that has gone through the same transition.
- Their core application comprises several autonomous applications, with each one executing the business logic for different function areas.

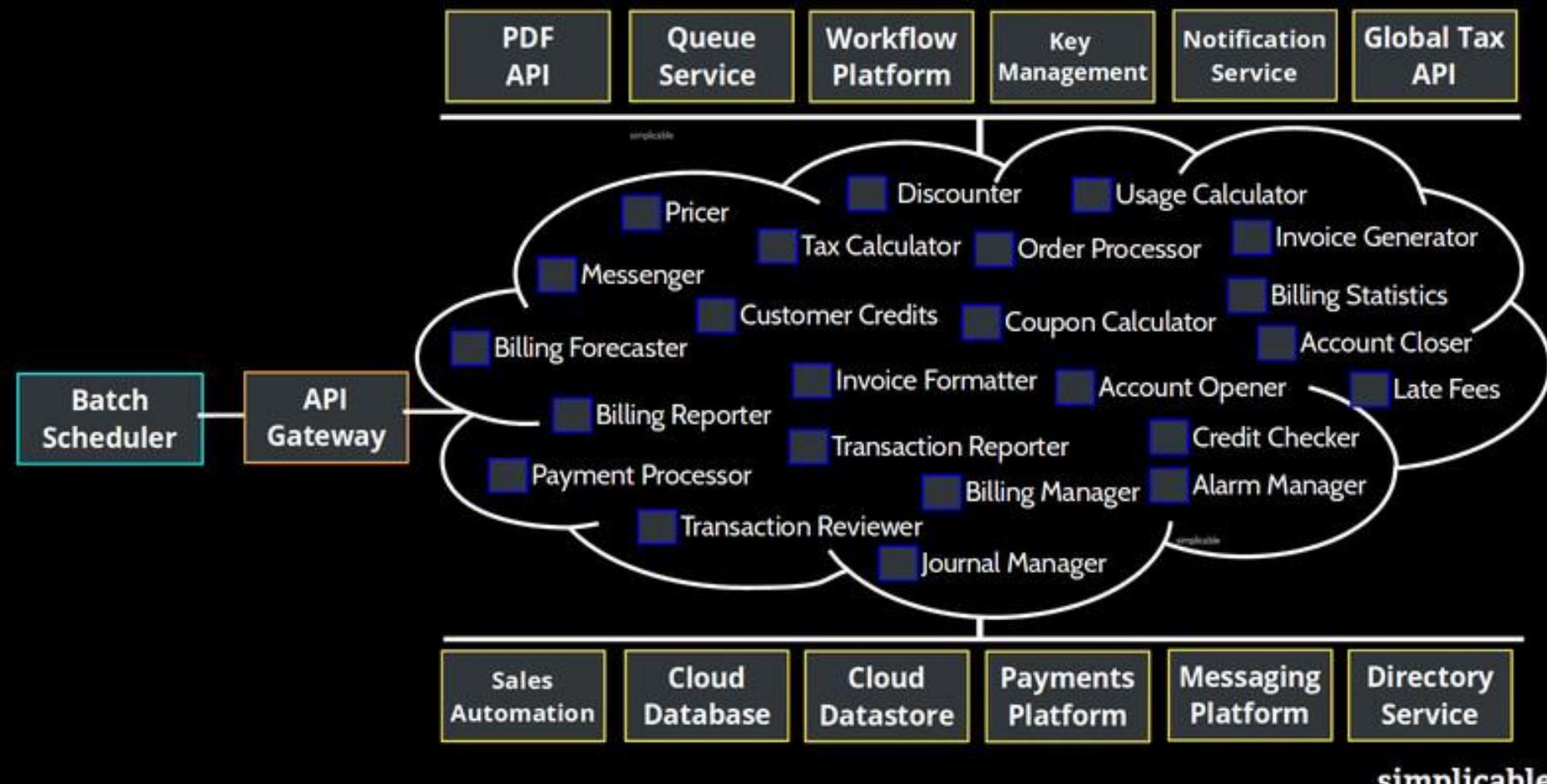
# Tax Calculator



simplicable

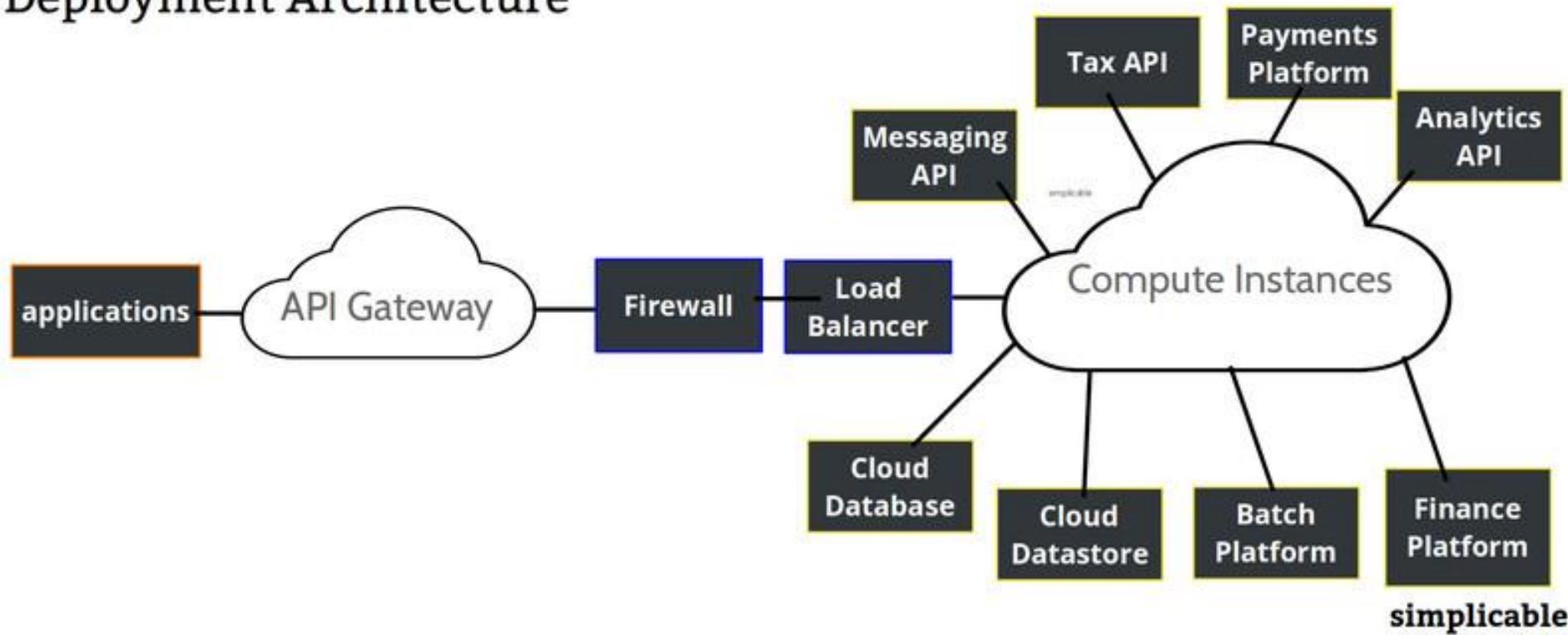


# Services Architecture





## Deployment Architecture





# Principles of Micro service

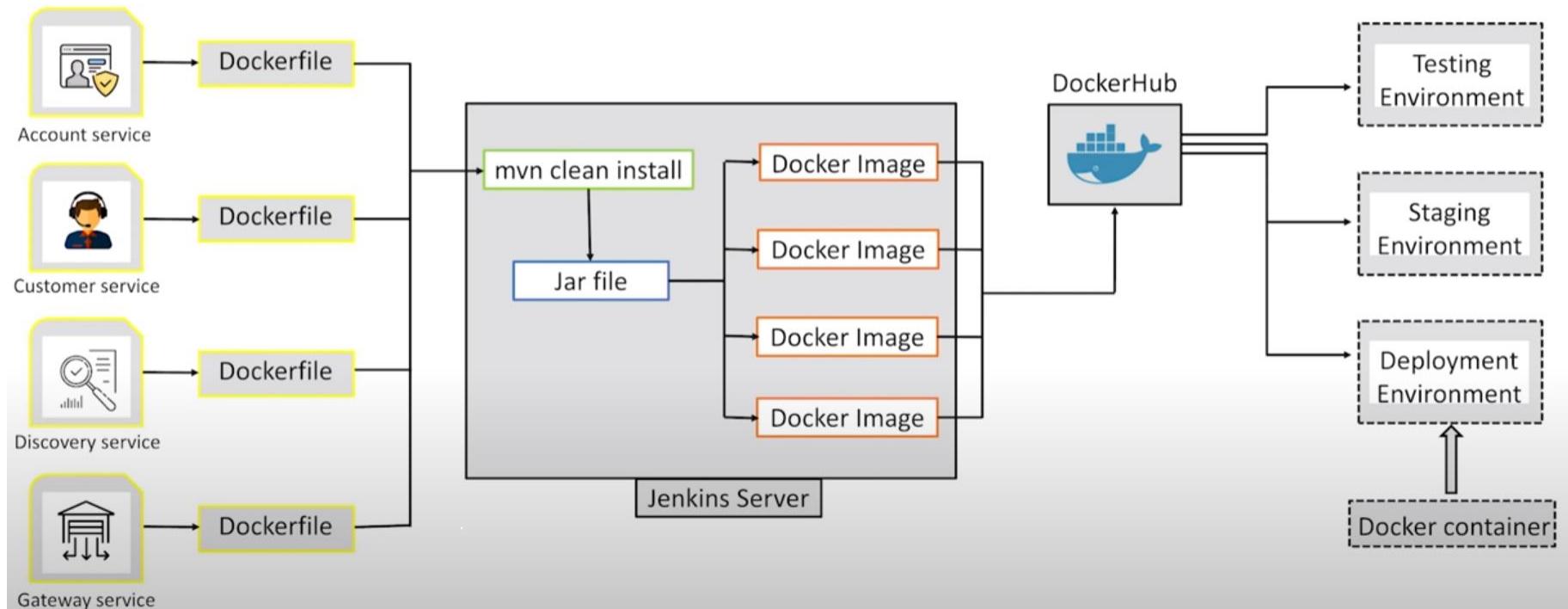
---

- Independent & Autonomous Services
- Scalability
- Decentralization
- Resilient Services
- Real-Time Load Balancing
- Availability
- Continuous delivery through DevOps Integration
- Seamless API Integration and Continuous Monitoring
- Isolation from Failures
- Auto -Provisioning

# Continuous delivery through DevOps Integration



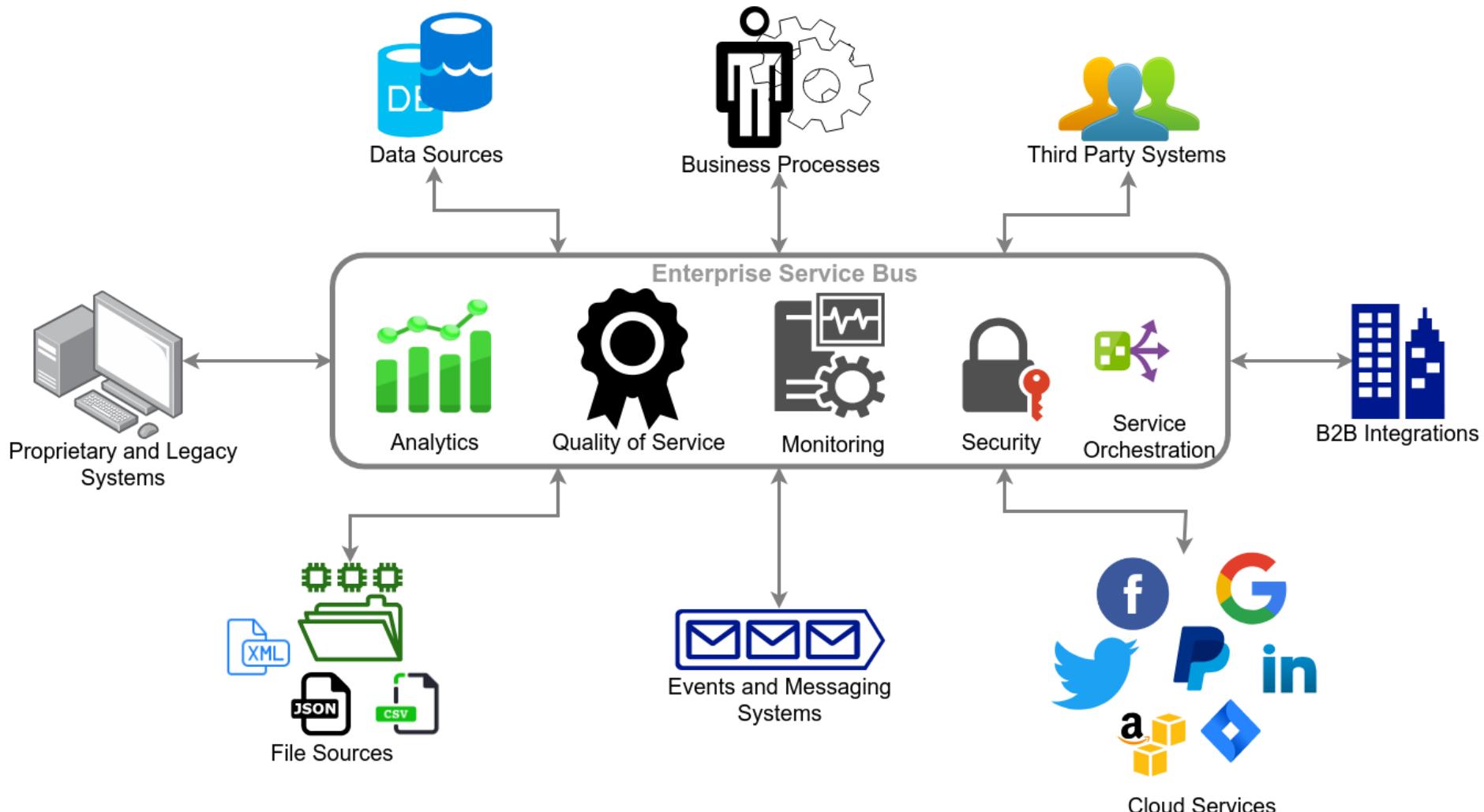
## Solution



# API Integration ESB or Microservices



# ESB for API Integration



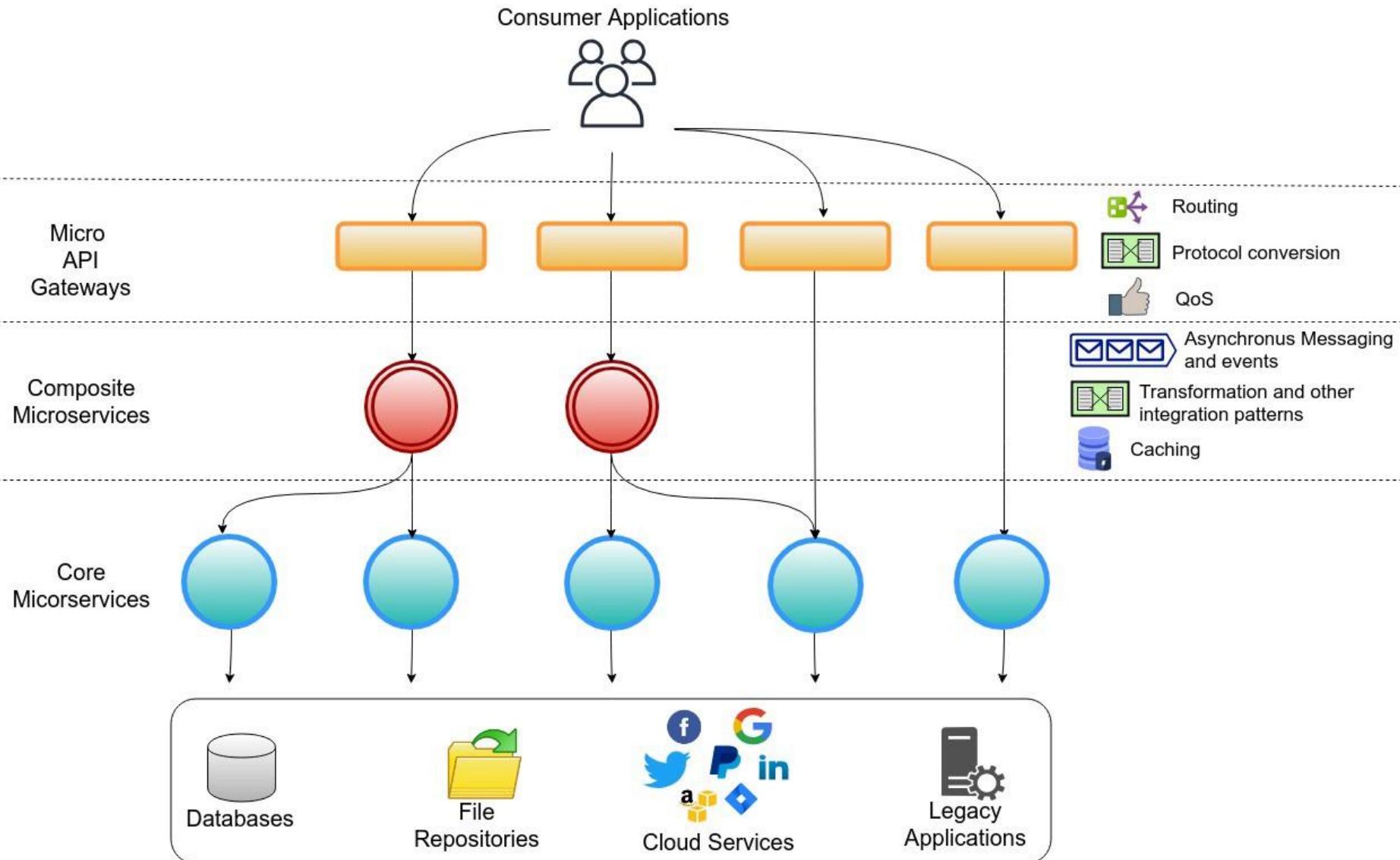
# Disadvantages of ESB



- **Added delay in round-trip**
- **Connectivity requirements(All calls through ESB)**
- **Single point of failure**
- **Complexity in scalability**



# Microservice for API Integration





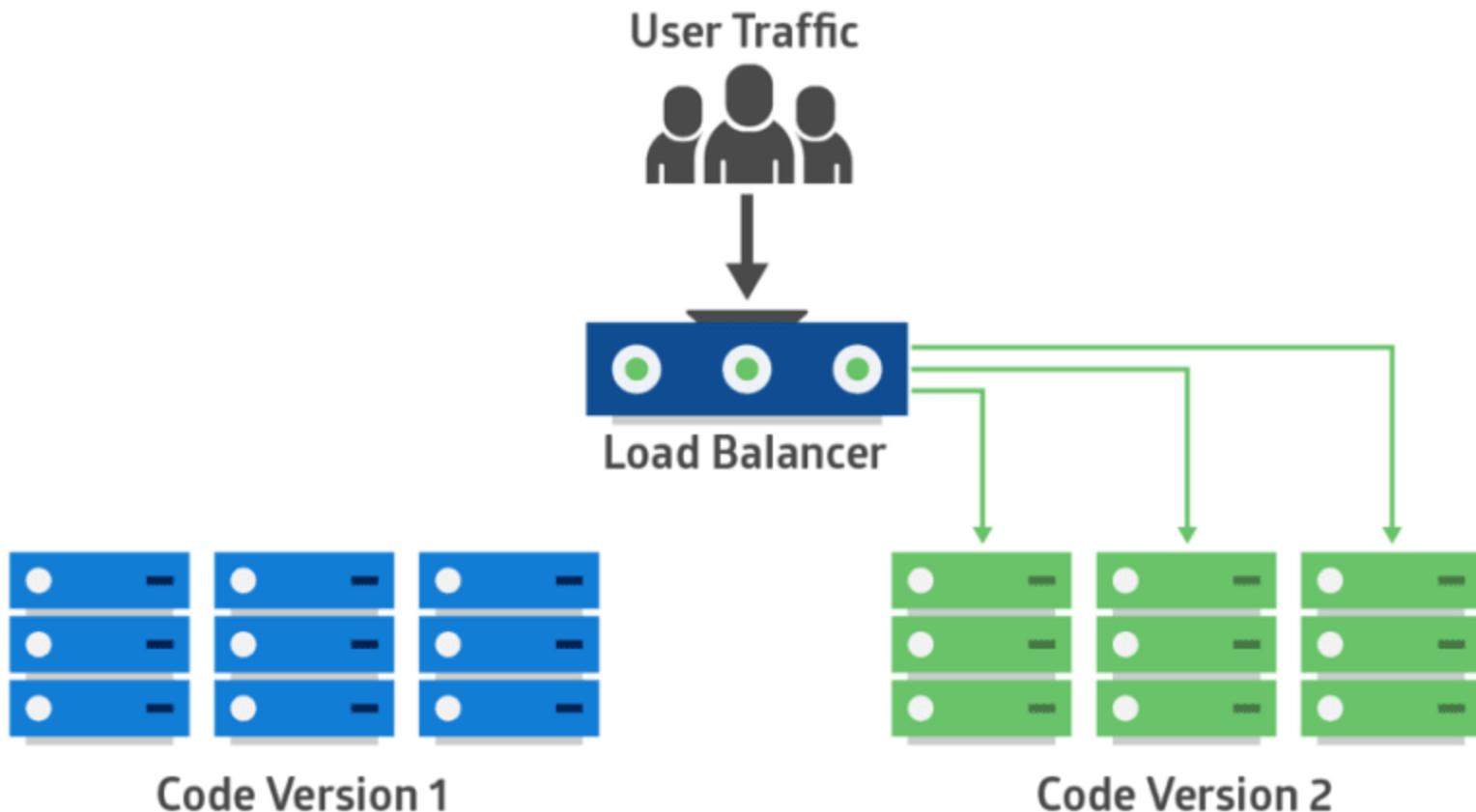
# ESB or Microservice

---

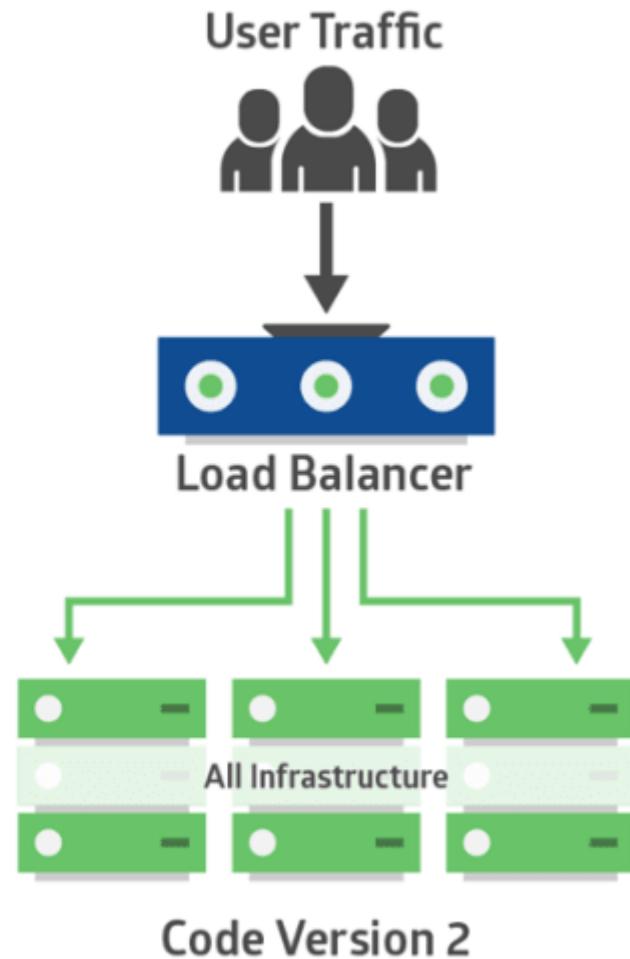
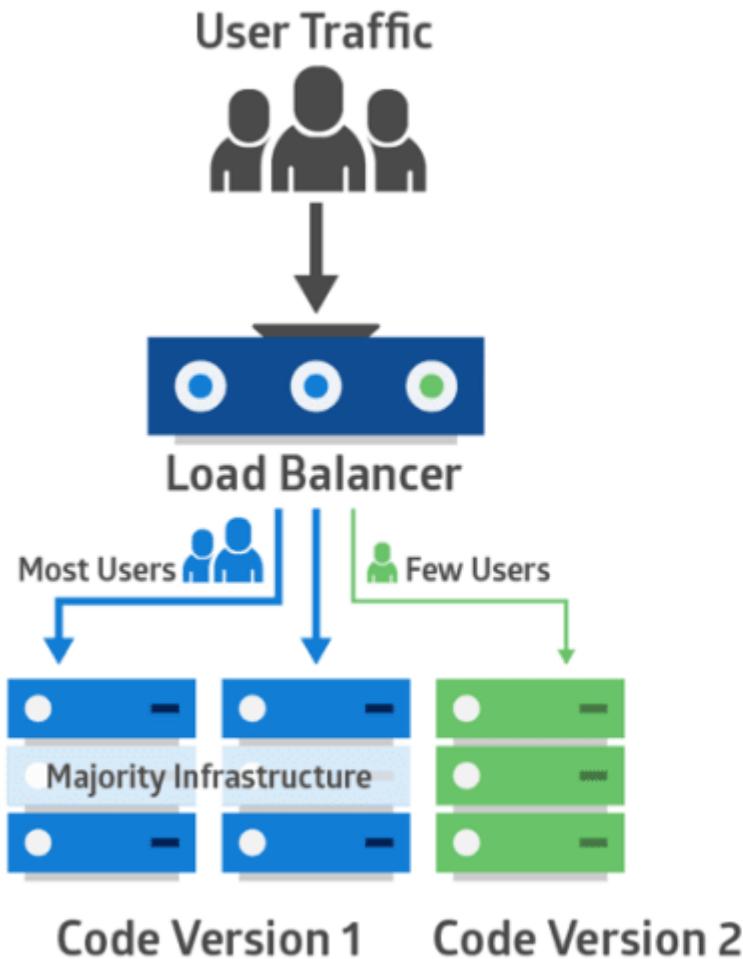
- When you select the suitable solution for the organization, following factors need to be assessed.
- Types of services to be integrated
- Communication protocols required
- Performance numbers
- Transaction requirements
- Security restrictions
- Inter-dependencies of services
- Business timelines
- Target expenses for the infrastructure
- Skill set of the team



# Blue Green Deployment



# Canary Testing



# TWELVE-FACTOR APPS AND MICROSERVICES



- Modern architecture aims to develop large and complex applications in software as service (SaaS) models.
- The twelve-factor methodology provides guidelines for developing SaaS-based applications.
- Microservices, containers and their ecosystem fit well into the twelve-factor methodology.

# TWELVE-FACTOR APPS AND MICROSERVICES



**TABLE 1 SOLUTION COMPONENTS FOR 12 FACTOR APPS**

FACTORS	BRIEF DETAILS <i>(Ref: <a href="https://www.12factor.net/">https://www.12factor.net/</a>)</i>	MICROSERVICE ECOSYSTEM COMPONENT
Codebase	One codebase tracked in revision control, many deploys	Source control systems such as gitlab or bitbucket can be leveraged for this. Source control systems provide in-built support for code revisions and version controls. Deployment can be done through build pipeline and CI/CD pipeline.
Dependencies	Explicitly declare and isolate dependencies	Build and packaging libraries such as Maven, Gradle and npm allow us to declare the dependent libraries along with the library version.
Config	Store config in the environment	Environment specific configurations (such as URLs, connection strings etc.) can be injected to the configuration files (such as application.properties in Spring application) in the build pipeline.
Backing services	Treat backing services as attached resources	Backing services such as storage, database or an external service should be accessible and managed through configuration files to ensure portability.
Build, release, run	Strictly separate build and run stages	Source code branches and automated CI/CD pipelines can be leveraged to manage environment specific releases.
Processes	Execute the app as one or more stateless processes	Stateless is the core tenets of microservices. Implementing token-based security helps us implement stateless authentication and authorization.

# TWELVE-FACTOR APPS AND MICROSERVICES



**TABLE 1 SOLUTION COMPONENTS FOR 12 FACTOR APPS**

FACTORS	BRIEF DETAILS (Ref: <a href="https://www.12factor.net/">https://www.12factor.net/</a> )	MICROSERVICE ECOSYSTEM COMPONENT
Port binding	Export services via port binding	The services can be made visible through exposed ports. Container infrastructure provides configuration files (such as service.yaml in Docker) to bind the ports for services.
Concurrency	Scale out via the process model	By leveraging independent deployment feature of microservices, we can individually scale the most needed microservice by using on-demand scaling feature of containers.
Disposability	Maximize robustness with fast startup and graceful shutdown	Individual containers/pods can be started quickly. Container orchestrator can handle container shutdown gracefully.
Dev/prod parity	Keep development, staging, and production as similar as possible	We can achieve parity in environment dependencies, server dependencies, configurations through a container model.
Logs	Treat logs as event streams	Each microservice can log to standard output, which can be picked up by tools such as Kibana or Splunk to manage and visualize the logs centrally.
Admin processes	Run admin/management tasks as one-off processes	Container orchestration is managed by tools such as Kubernetes, while log management is carried out by tools such as Kibana or Splunk. Other application-specific administration can be deployed as a separate microservice.



# Microservice Reference Architecture

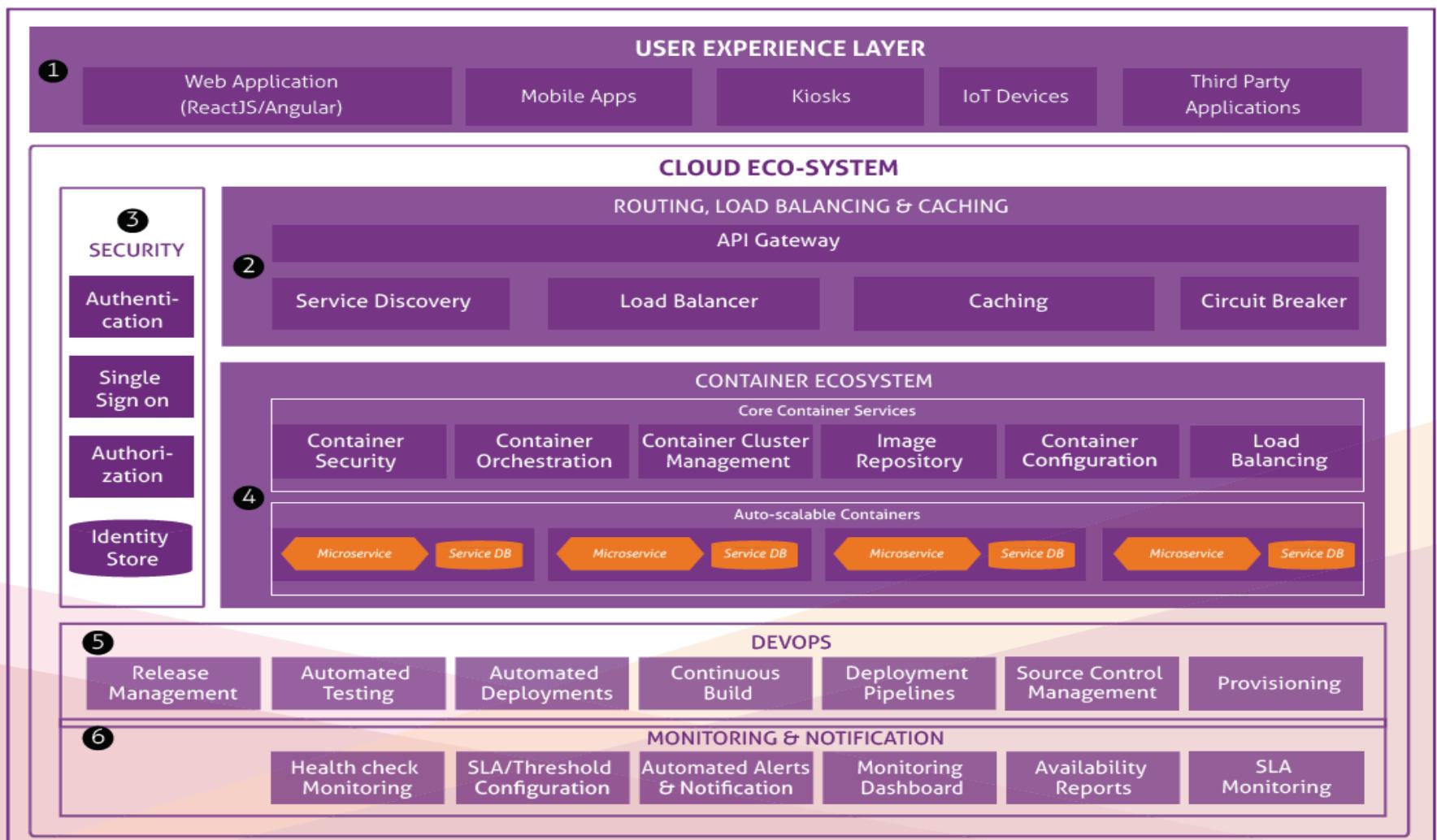


Figure 1: Microservices Reference Architecture



# Sample Microservice Interaction

## CLOUD NATIVE MICROSERVICE FRAMEWORK – SOLUTION COMPONENTS

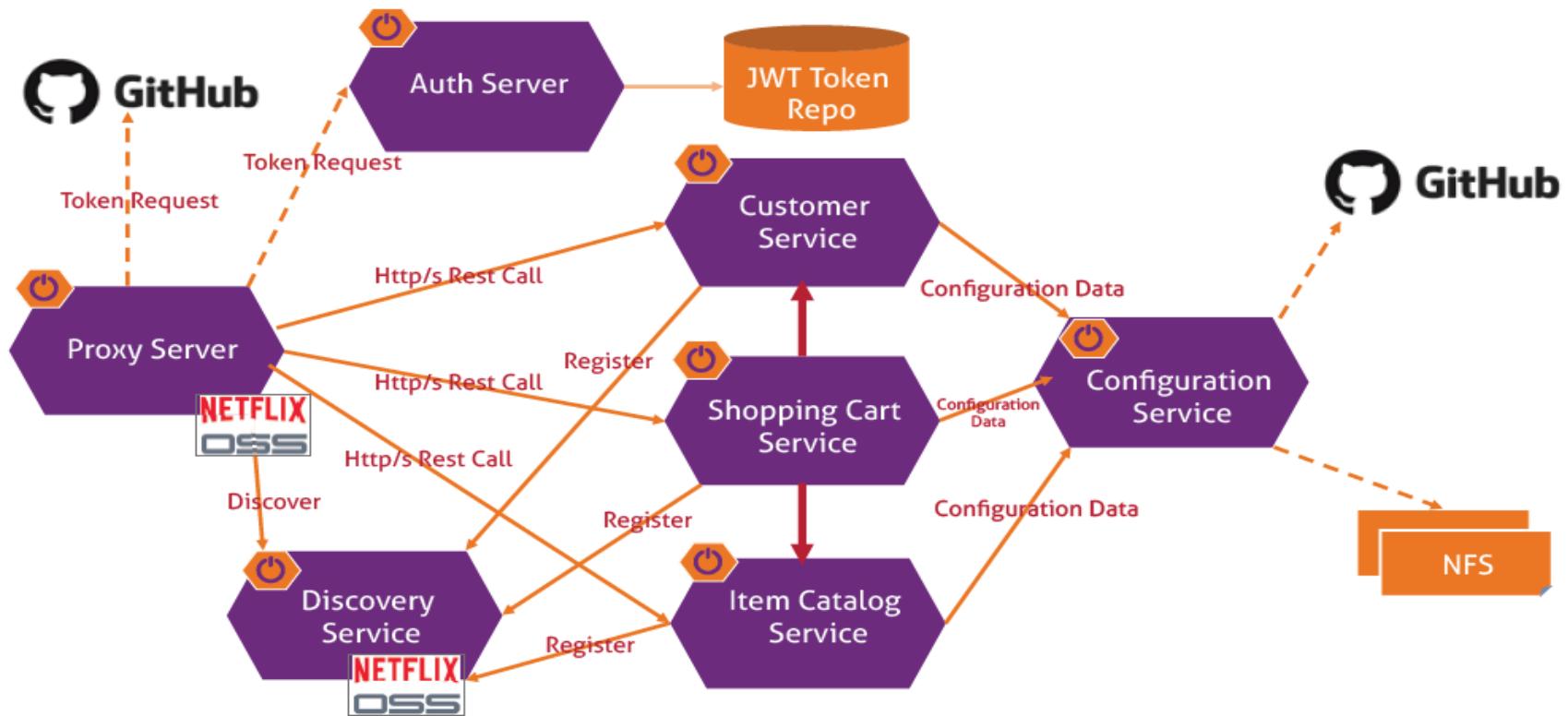


Figure 2: Sample Microservice Interaction Diagram



# Sample Microservice Interaction

INFRA SERVICE	DETAILS
Auth Server	Custom Authorization server implementation which provides a configurable (in-memory, JWT) identity token. The token will be used to verify a user's authenticity every time the client tries to access the above business service
Proxy Server	We can leverage the Netflix OSS- Zuul service.
Service Discovery	We can leverage the Netflix OSS- Eureka service
Configuration Service	Custom cloud configuration service implementation. It provides configurable in-file or Git storage for service configuration.



# Sample Microservice Interaction

ARCHITECTURE COMPONENT	SAMPLE PRODUCT STACK
Api Gateway	Netflix OSS- Zuul
Authentication Service	Spring – Security OAuth2, OpenID Connect
Service Discovery	Netflix OSS- Eureka, Apache Zookeeper
Configuration Service	Spring – Cloud Config Server
Microservice	Spring – Boot, Vert.x, Dropwizard
Monitoring	Netflix OSS- Turbine, Prometheus, Splunk, ELK (Elasticsearch, Logstash, Kibana), CAdvisor Visualization – Grafana, Kibana
Circuit Breaker	Netflix OSS- Hystrix
Microservices Testing	Wiremock
Container Ecosystem	Docker – Container technology Docker Swarm, Kubernetes – Container orchestrator

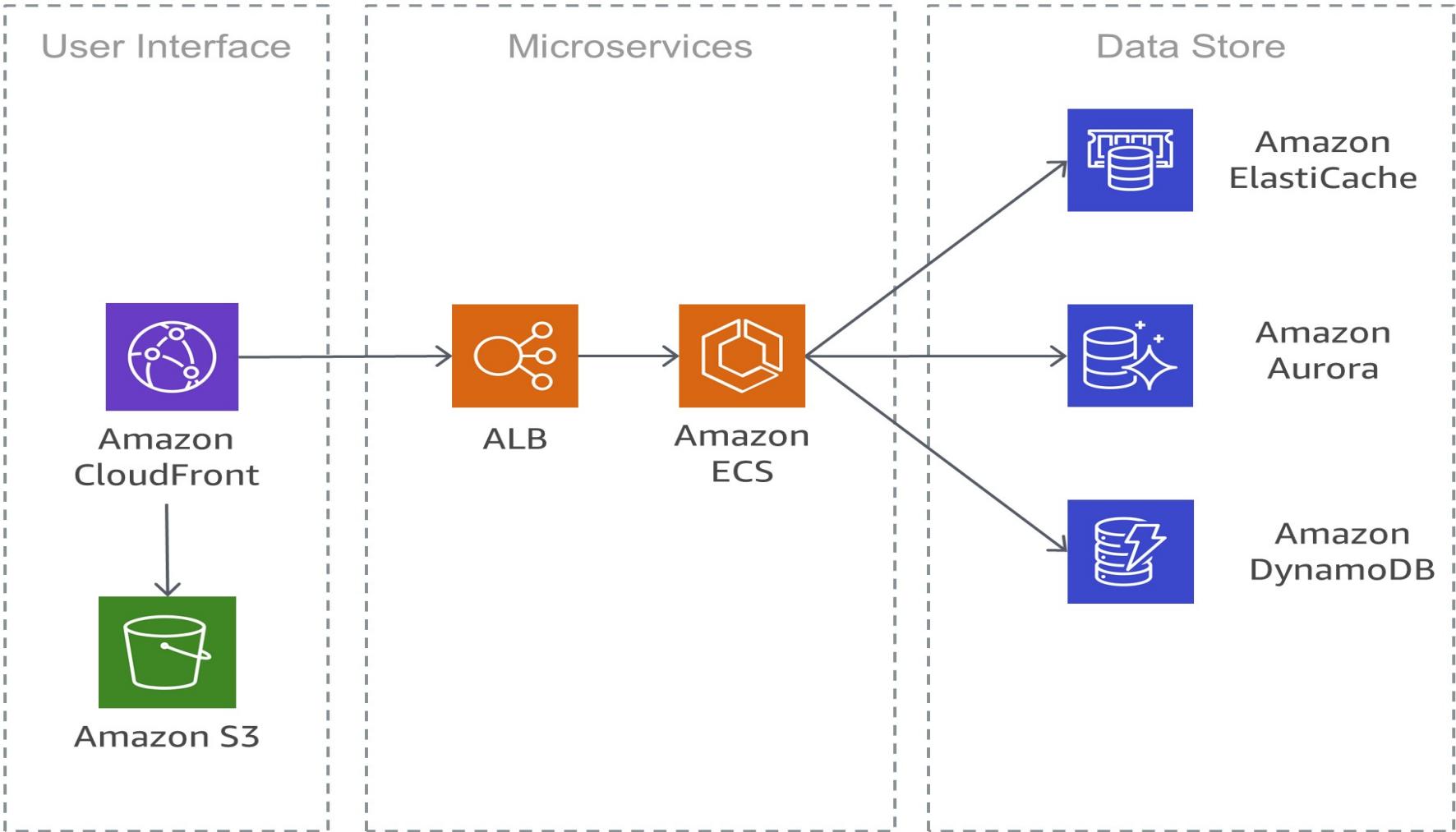


# Cloud Deployment Architecture - AWS

**Cloudfront CDN:** Cloudfront native CDN allows to effectively deliver the content for the consumer from different geographies.

- **S3 Bucket:** ReactJS/Angular-based web applications will be deployed on S3 bucket & it is integrated with AWS Cloud front. The frontend is secured with WAF (Web Application Firewall). Deploying frontend static content on S3 makes it highly scalable & cost effective.
- **Custom Services Layer:** Backend microservices are developed using NodeJS/ Spring Boot and are deployed on containers using AWS Cloud-native container orchestration services ECS. These services are accessed through AWS API Gateway.

# Simple Microservices Architecture on AWS





# Cloud Deployment Architecture - AWS

## AWS-BASED MICROSERVICES DEPLOYMENT ARCHITECTURE

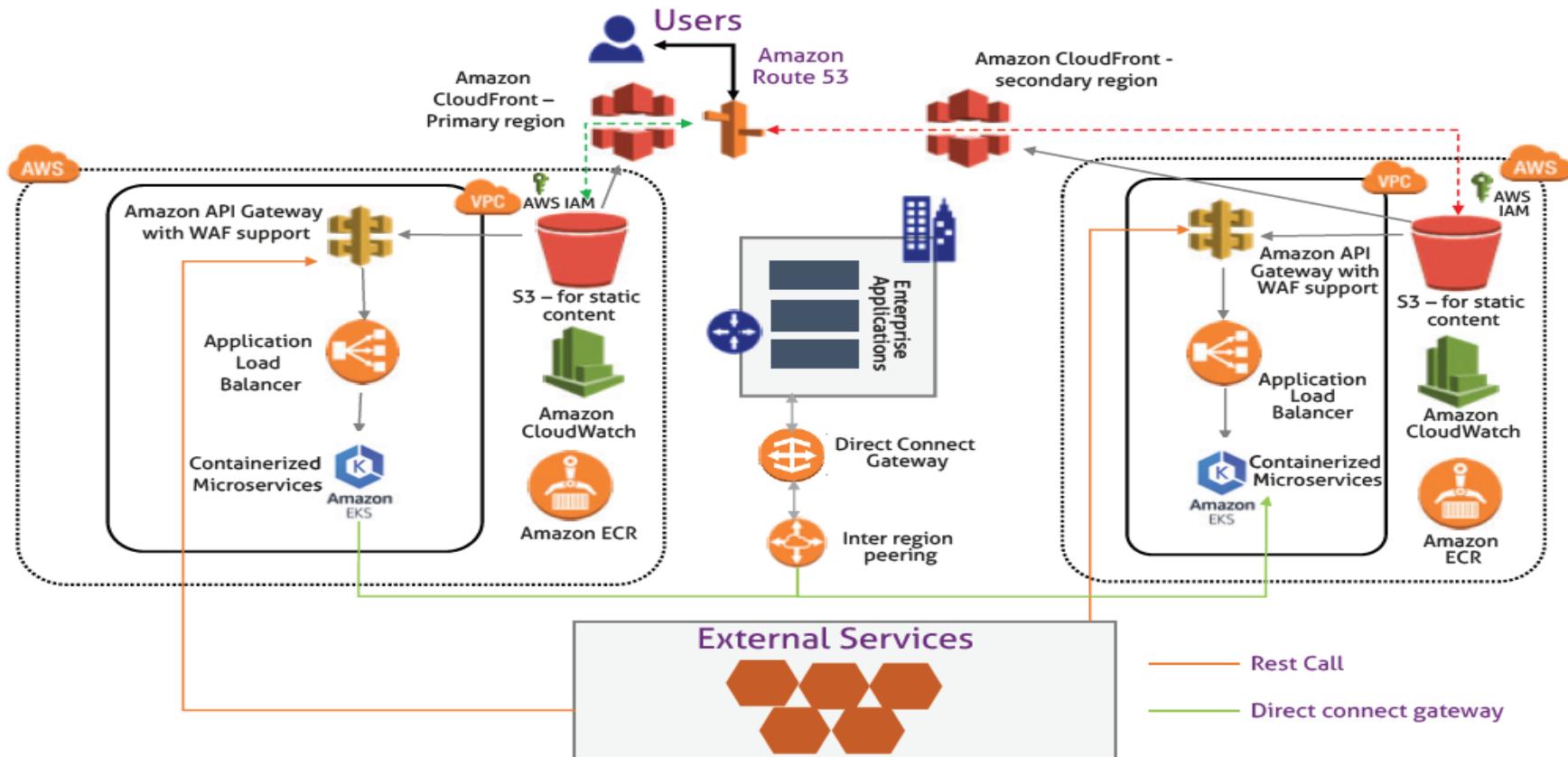


Figure 3: Sample AWS-based Microservices Deployment Architecture

# Cloud Deployment Architecture - Azure



## AZURE-BASED MICROSERVICES DEPLOYMENT ARCHITECTURE

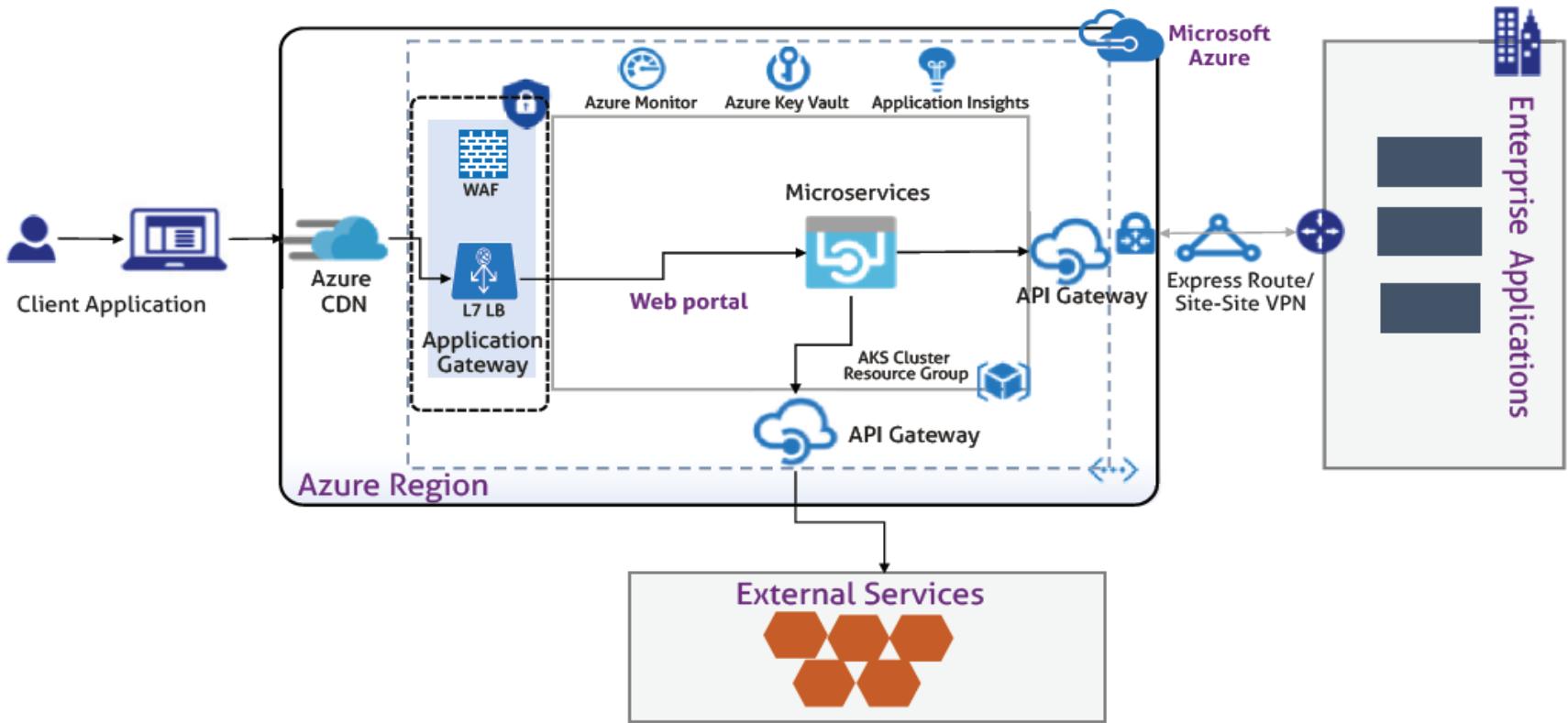


Figure 4: Sample Azure-based Microservices Deployment Architecture



# Microservices Features

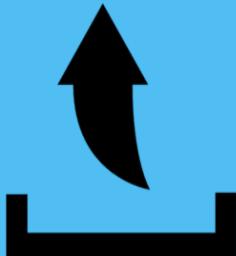




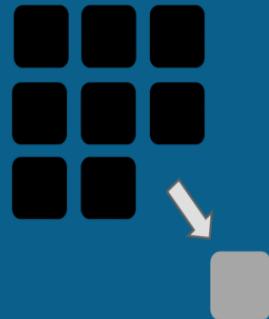
# Advantages Of Microservices



Independent  
Development



Independent  
Development



Independent  
Deployment

Fault  
Isolation

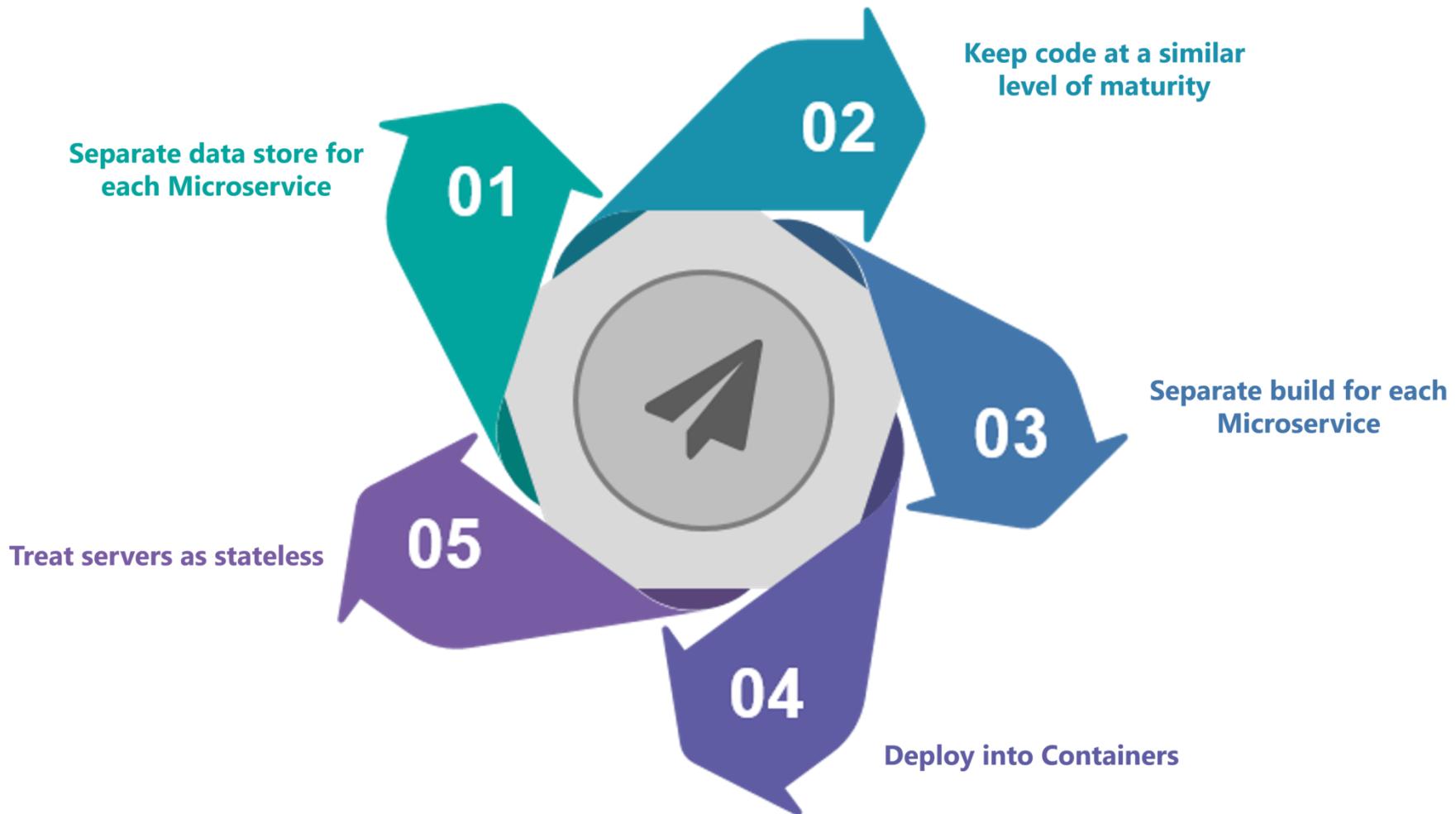
Mixed  
Technology  
Stack



Granular  
Scaling

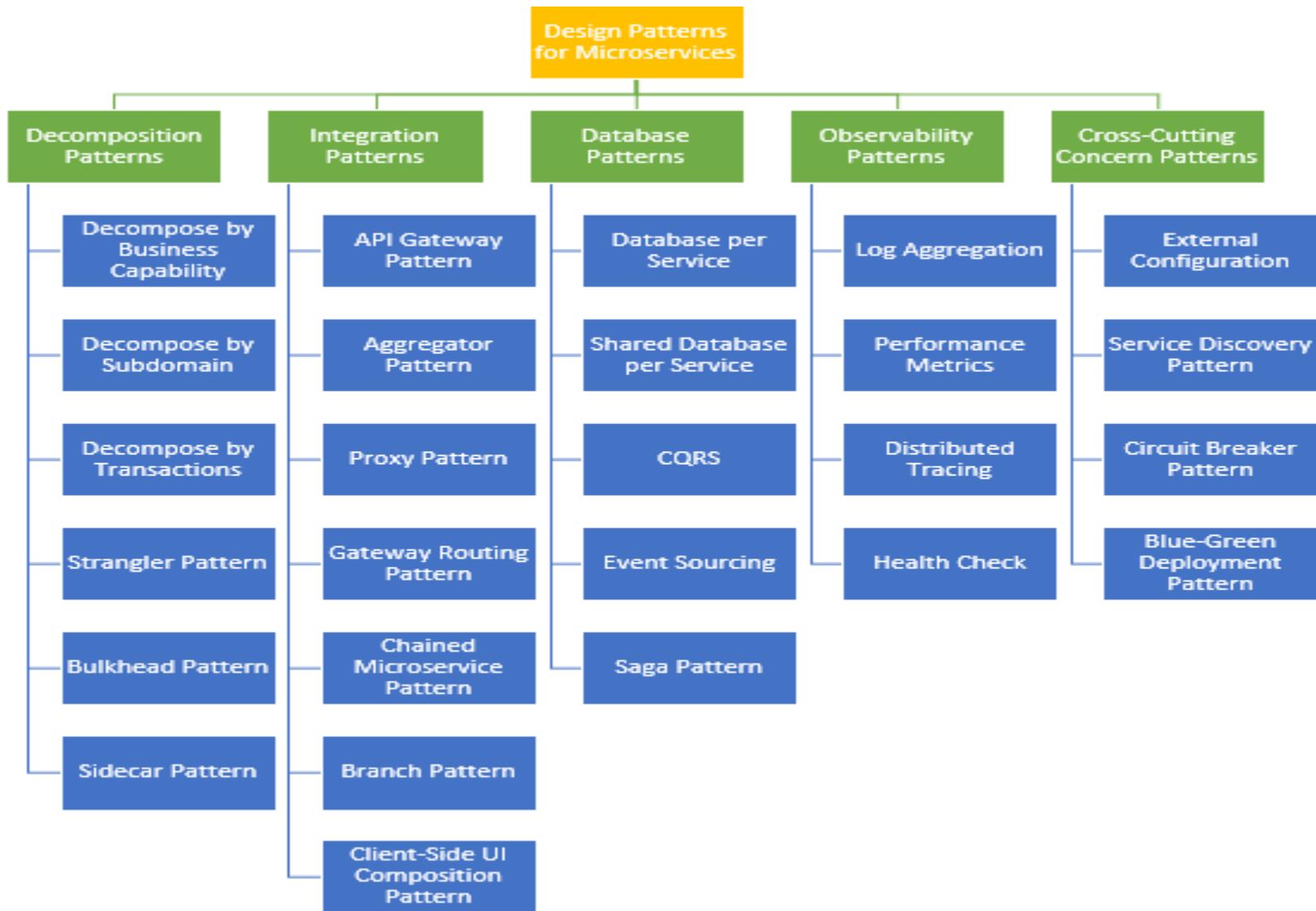


# Best Practices To Design Microservices





# Design Patterns of Micro services



# Decomposition Patterns

# Decompose by Business Capability



- Problem
- Microservices is all about making services loosely coupled, applying the single responsibility principle.
- However, breaking an application into smaller pieces has to be done logically.
- How do we decompose an application into small services?

# Decompose by Business Capability



- **Solution**
- One strategy is to decompose by business capability.
- A business capability is something that a business does in order to generate value.
- The set of capabilities for a given business depend on the type of business.
- For example, the capabilities of an insurance company typically include sales, marketing, underwriting, claims processing, billing, compliance, etc.
- Each business capability can be thought of as a service, except it's business-oriented rather than technical.



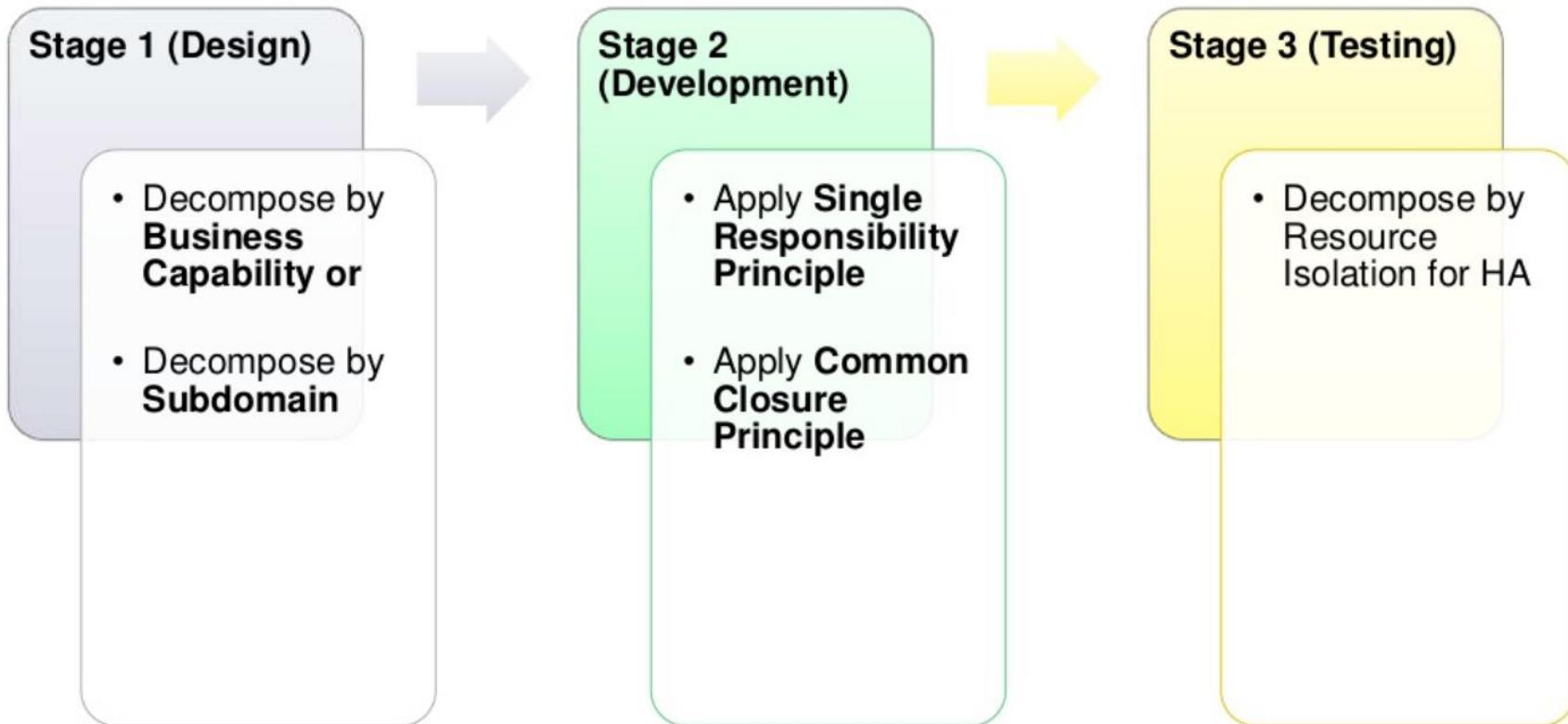
# Decompose by Business Capability

- Microservice is all about making services loosely coupled, applying the single responsibility principle.
- It decomposes by business capability.
- Define services corresponding to business capabilities.
- A business capability is a concept from business architecture modeling.
- A business capability often corresponds to a business object, e.g.
  - Order Management is responsible for orders
  - Customer Management is responsible for customers.
- For instance, in an e-commerce solution, the main business capabilities are order management, product promotions, service management and others. We can create microservices based on these.



# Microservice Decomposition Strategy

The process to define Microservice decomposition.





# Decompose by Subdomain

---

- Decomposing an application using business capabilities might be a good start, but there will be so-called “God Classes” which will not be easy to decompose.
- These classes will be common among multiple services.
- A domain consists of multiple subdomains.
- Each subdomain corresponds to a different part of the business.



## Decompose by Subdomain

---

- Subdomains can be classified as follows:
- Core — key differentiator for the business and the most valuable part of the application
- Supporting — related to what the business does but not a differentiator. These can be implemented in-house or outsourced
- Generic — not specific to the business and are ideally implemented using off the shelf software



## Decompose by Subdomain

---

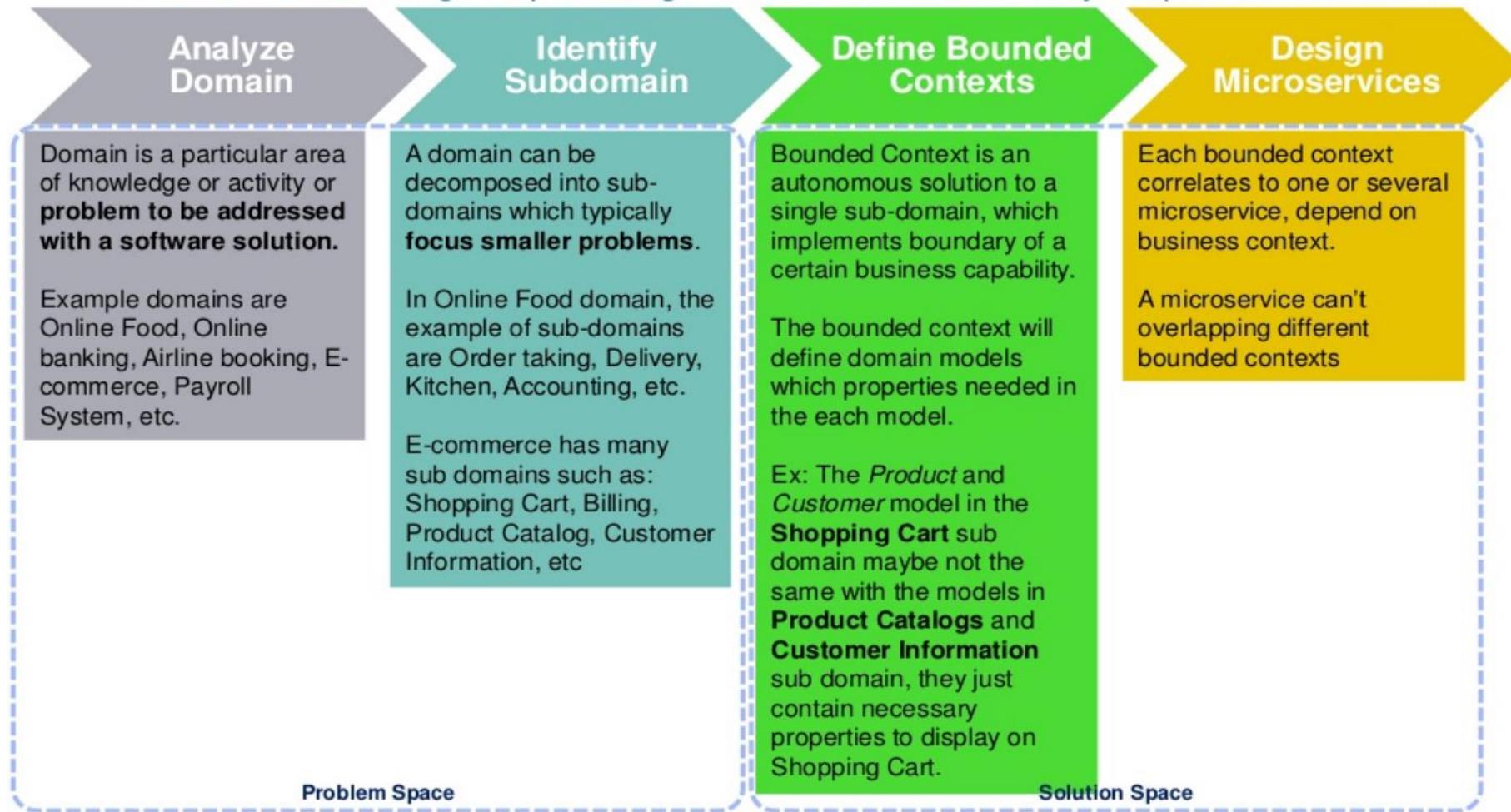
- The subdomains of an Order management include:
  - Product catalog service
  - Inventory management services
  - Order management services
  - Delivery management services

# Decompose by Subdomain



## Decompose by Subdomain Pattern

Domain-driven design help to design Microservices that loosely coupled and cohesive.

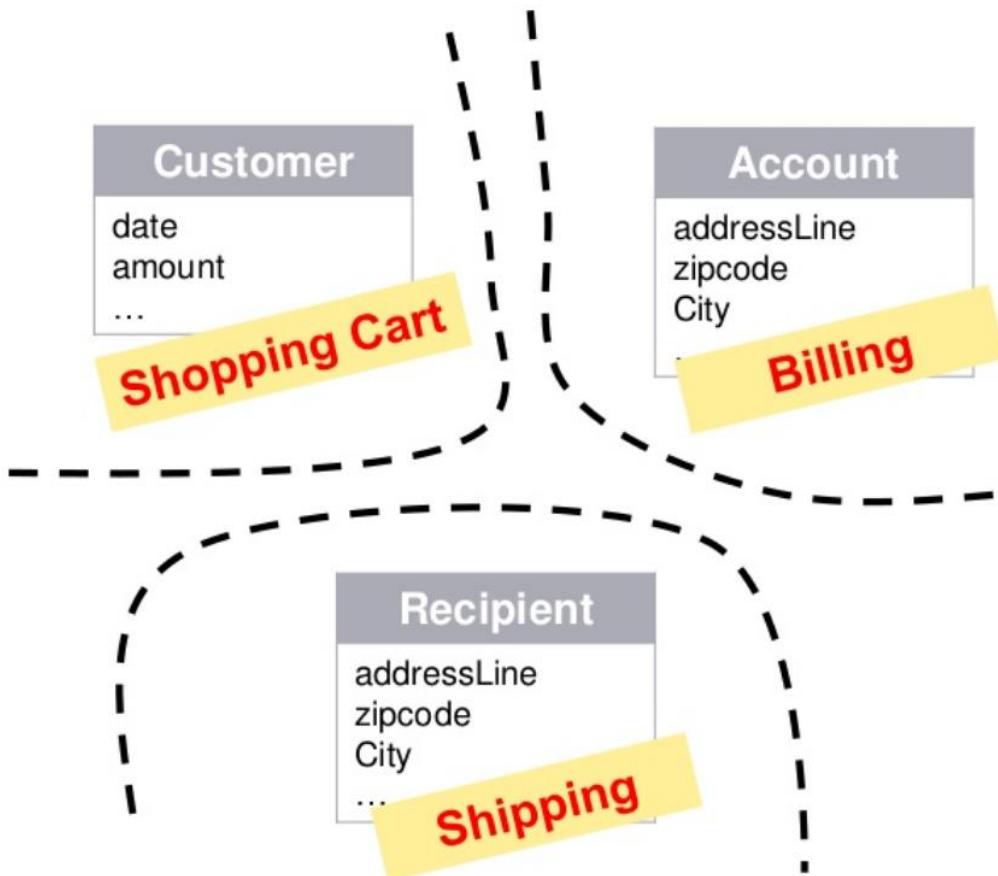


# Decompose by Subdomain



## If We Using Bounded Context

Split by business domains or business capabilities



- Achieve Autonomous Team
- Avoid distributed monolith application
- Avoid cascading failure
- Simplify the complexity of the business logic
- Achieve Business and IT alignment

But

- Create partial duplication of data
- How to correlate or integrate the data?
- How to create data consistency and data integrity?
- How to create effective and efficient query?

# Decompose by Subdomain



2

## DDD: Bounded Context – Strategic Design



An App User's Journey can run across multiple Bounded Context / Micro Services.



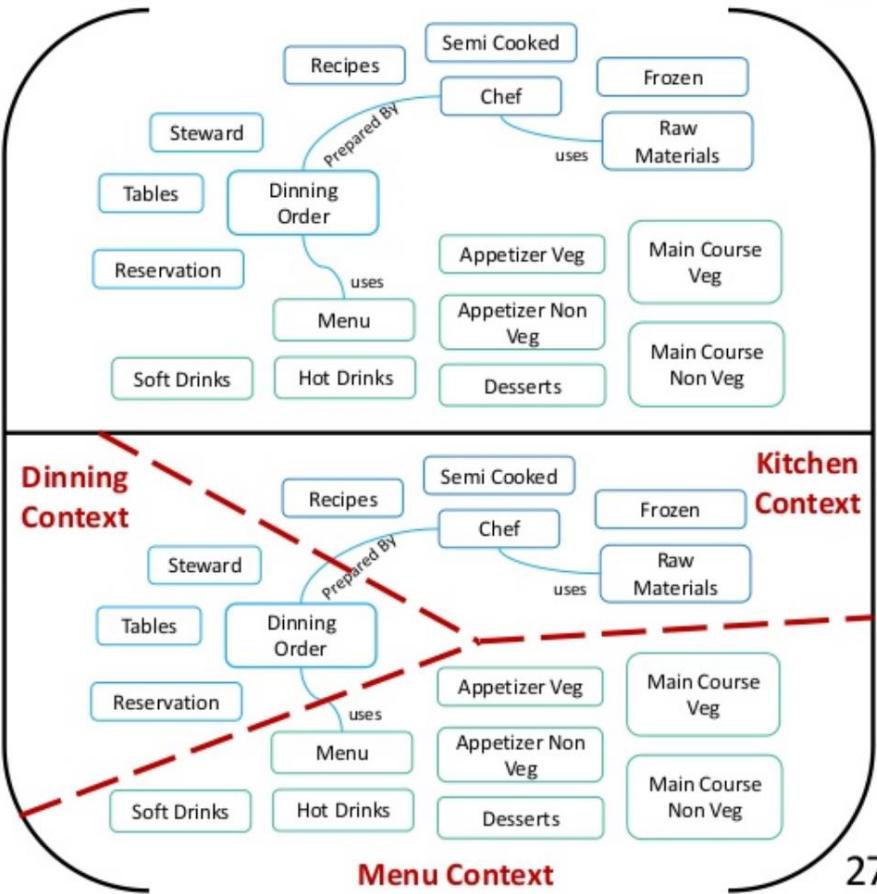
Areas of the domain treated independently

Discovered as you assess requirements and build language

Source: Domain-Driven Design  
Reference by Eric Evans



Understanding Bounded Context (DDD) of a Restaurant App



# Decompose by Transactions / Two-phase commit (2pc) pattern

---



- Identify the main transactions of the application and develop microservices for them.
- For instance, the main transactions of an e-commerce application are login, checkout, search and such; We can create microservices for these transactions.

# Decompose by Transactions / Two-phase commit (2pc) pattern



- You can decompose services over the transactions.
- Then there will be multiple transactions in the system.
- The distributed transaction consists of two steps:
- Prepare phase — during this phase, all participants of the transaction prepare for commit and notify the coordinator that they are ready to complete the transaction
- Commit or Rollback phase — during this phase, either a commit or a rollback command is issued by the transaction coordinator to all participants

# Decompose by Transactions / Two-phase commit (2pc) pattern



- The problem with 2PC is that it is quite slow compared to the time for operation of a single microservice.
- Coordinating the transaction between microservices, even if they are on the same network, can really slow the system down, so this approach isn't usually used in a high load scenario.



## Decomposition based on resources

---

- We can create microservices based on nouns or resources and define the operations.
- For instance, in an e-commerce solution, ‘products’ is a resource and we can define the list all products
- (GET /products), query particular product (GET /product/{1}), insert product (PUT /product/{}).



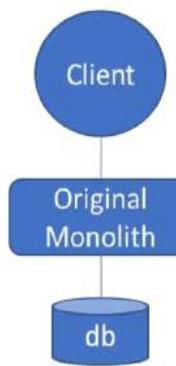
# Strangler Pattern

- This creates two separate applications that live side by side in the same URI space.
- **Over time, the newly refactored application “strangles” or replaces the original application and shuts off monolithic application.**
- Application steps are transform, coexist, and eliminate:
- Transform — Create a parallel new site with modern approaches.
- Coexist — Leave the existing site where it is for a time. Redirect from the existing site to the new one so the functionality is implemented incrementally.
- Eliminate — Remove the old functionality from the existing site.

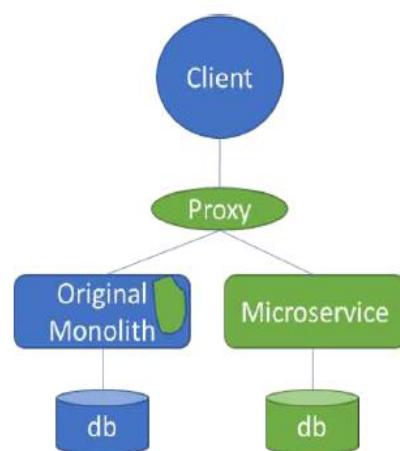
# Strangler Pattern



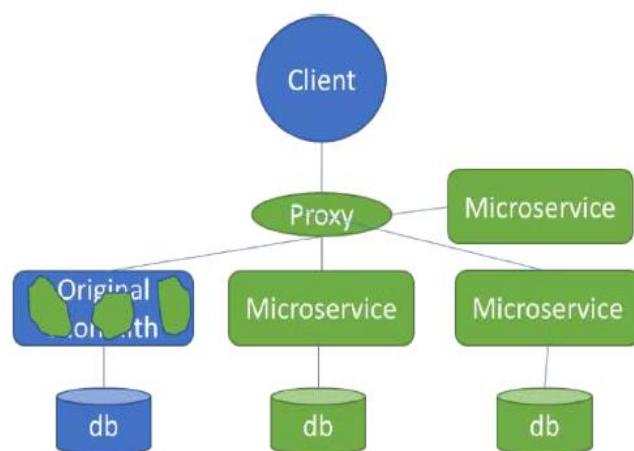
**Transform** – Create a parallel microservice



**Co-exist** – Incrementally redirect the traffic from the legacy to microservice



**Eliminate** – eliminate the legacy module





# Bulkhead Pattern

---

- Isolate elements of an application into pools so that if one fails, the others will continue to function.
- This pattern is named Bulkhead because it resembles the sectioned partitions of a ship's hull.
- Partition service instances into different groups, based on consumer load and availability requirements.
- This design helps to isolate failures, and gives sustain service functionality for some consumers, even during a failure.



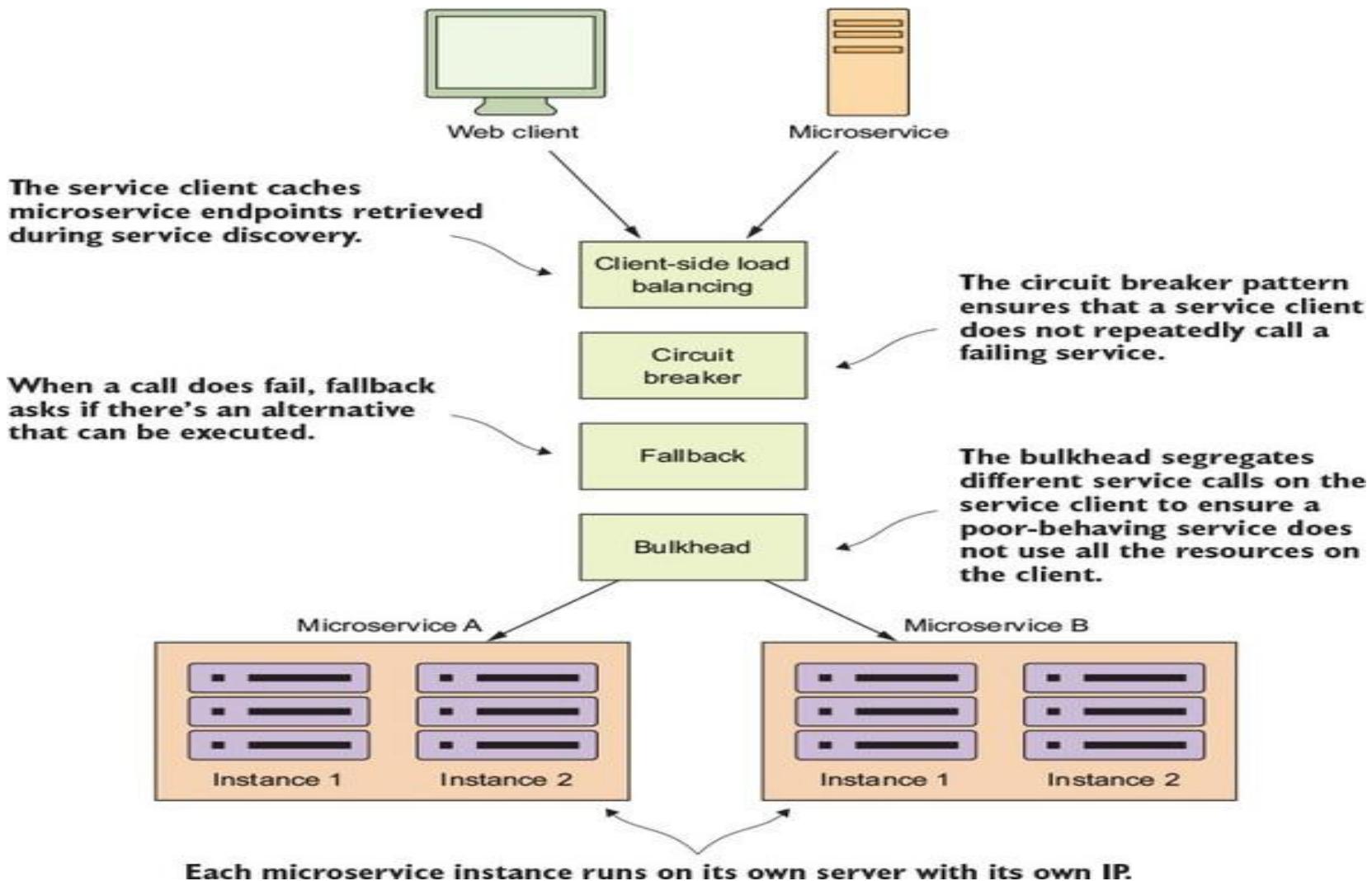
# Bulkhead Pattern

---

- Bulkhead is a term borrowed from cargo ships.
- In a cargo ship, the bulkhead is a wall built between different cargo sections, which makes sure that a fire or flood in one section is restricted to that section and other sections are not impacted.
- Failure in one service or a group of services should not bring down the whole application.
- To implement the bulkhead pattern, we need to make sure that all our services work independently of each other and failure in one will not create a failure in another service.
- Techniques such as maintaining a single-responsibility pattern, an asynchronous-communication pattern, or fail-fast and failure-handling patterns help us to achieve ...



# Bulkhead Pattern





# Sidecar Pattern

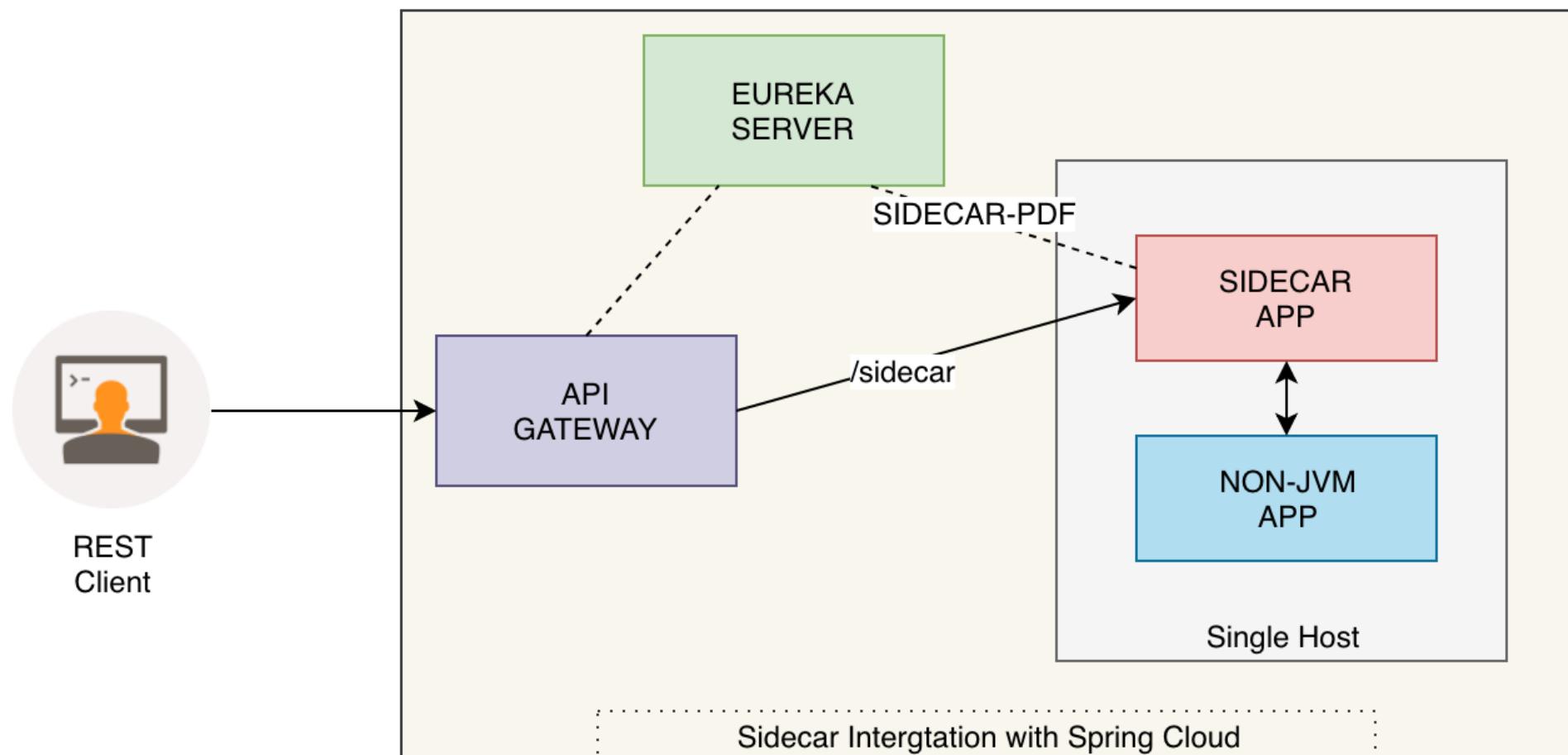
- Deploy components of an application into a separate processor container to provide isolation and encapsulation.
- This pattern can also enable applications to be composed of heterogeneous components and technologies.
- This pattern is named Sidecar because it resembles a sidecar attached to a motorcycle.
- In the pattern, the sidecar is attached to a parent application and provides supporting features for the application.
- The sidecar also shares the same lifecycle as the parent application, is created and retired alongside the parent.
- The sidecar pattern is sometimes referred to as the sidekick pattern and is the last decomposition pattern.



# Sidecar Pattern

- The problem statement
- There are multiple problems related to inter service communication when polyglot comes into picture. few of them are:
- How does non-JVM apps discover JVM microservices in Spring Cloud environment and vice-versa.
- In any cloud native application, we would never want to hard code host and port of apps for discovery purpose.
- How does non-JVM app communicates (i.e. calls REST endpoints) with JVM apps and vice versa.
- How do we utilize the client side load balancing features for non-JVM apps.
- How do we utilize the config server for non-JVM apps?

# Sidecar for the Rescue



# Integration Patterns



# API Gateway Pattern

---

- An API gateway provides a centralized access point for invoking a microservice.
- The API gateway handles security (such as authentication, authorization), governance (such as logging service, monitoring service), request routing, protocol transformation, data transformation and the aggregation of responses from multiple services.

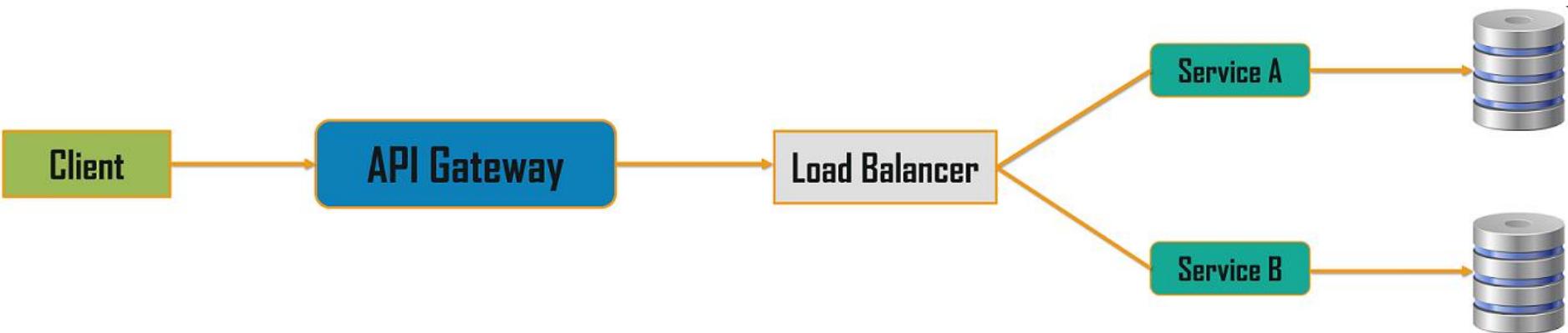


# API Gateway Pattern

- An API Gateway helps to address many concerns raised by the microservice implementation.
- An API Gateway is the single point of entry for any microservice call.
- It can work as a proxy service to route a request to the concerned microservice.
- It can aggregate the results to send back to the consumer.
- This solution can create a fine-grained API for each specific type of client.
- It can also convert the protocol request and respond.
- It can also offload the authentication/authorization responsibility of the microservice.



## API Gateway Design Pattern





# Aggregator Pattern

---

- When breaking the business functionality into several smaller logical pieces of code, it becomes necessary to think about how to collaborate the data returned by each service.
- This responsibility cannot be left with the consumer.
- The Aggregator pattern helps to address this.
- It talks about how we can aggregate the data from different services and then send the final response to the consumer.



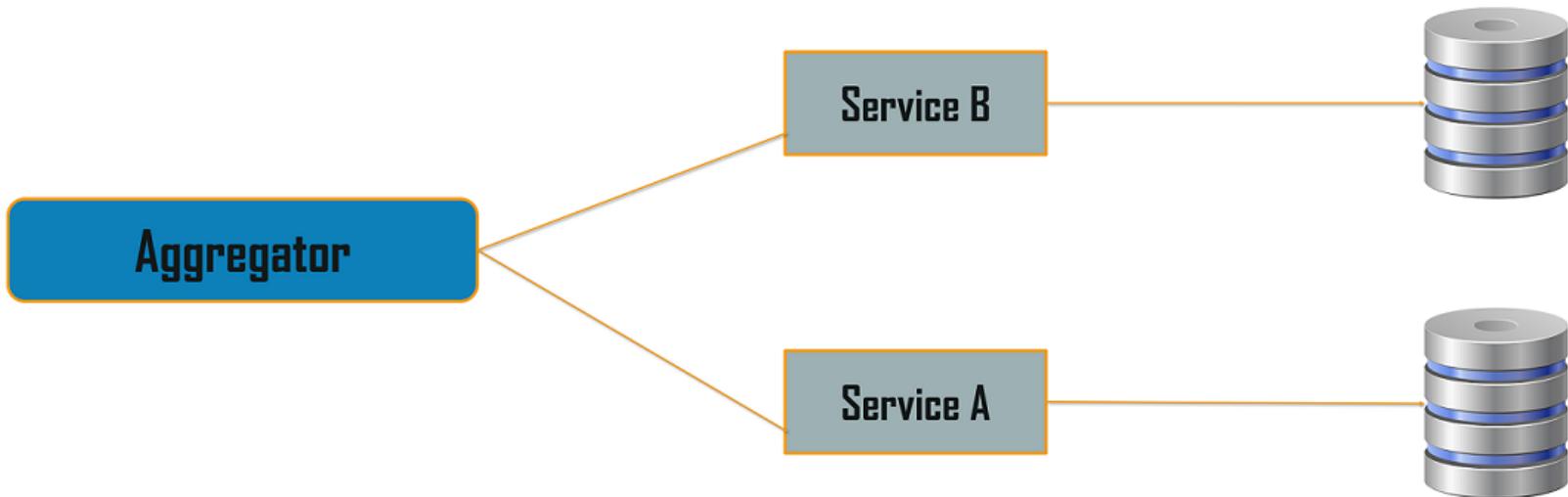
# Aggregator Pattern

---

- A composite microservice will make calls to all the required microservices, consolidate the data, and transform the data before sending back.
- An API Gateway can also partition the request to multiple microservices and aggregate the data before sending it to the consumer.
- When a single microservice needs responses from multiple microservices, a composite service can take the responsibility of aggregating the response.

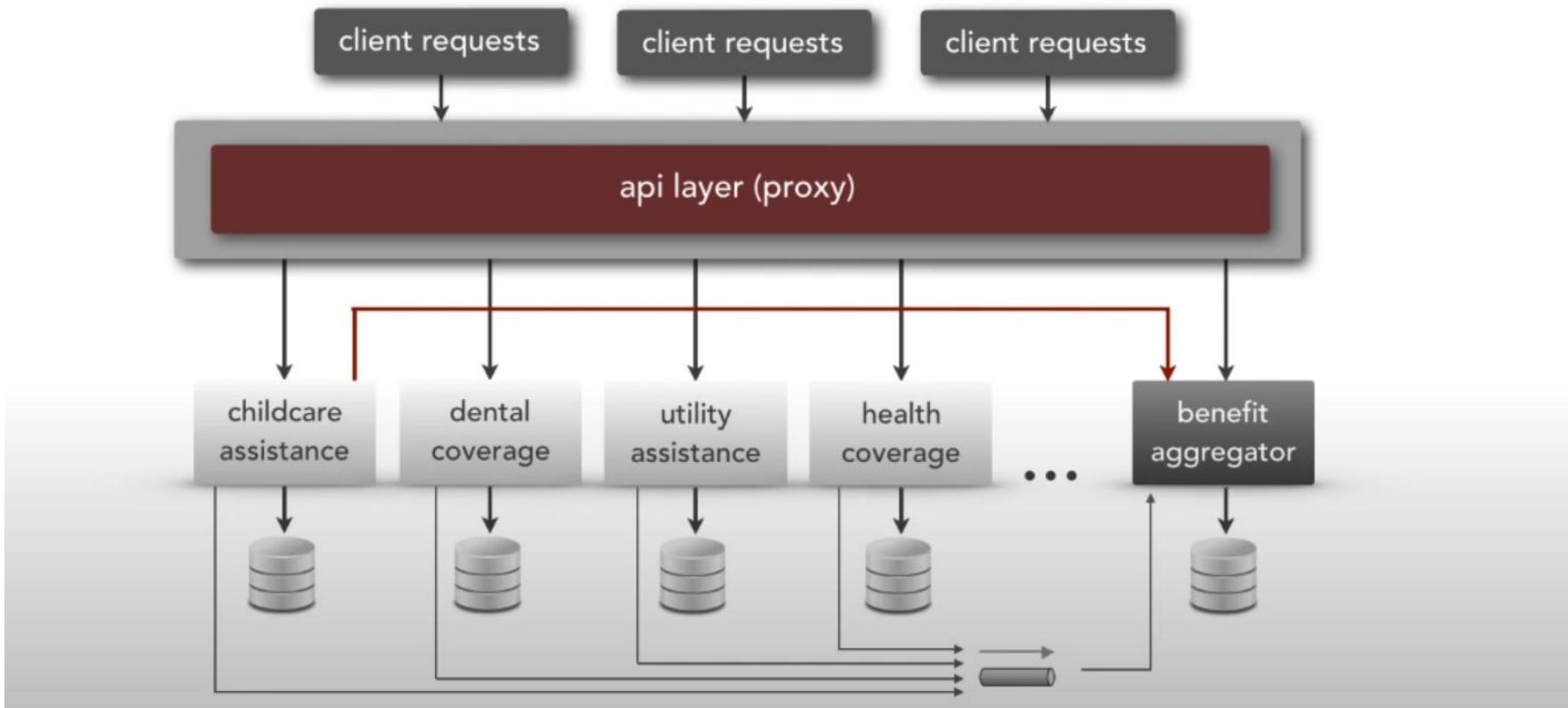


## Aggregator Pattern





## microservice aggregator



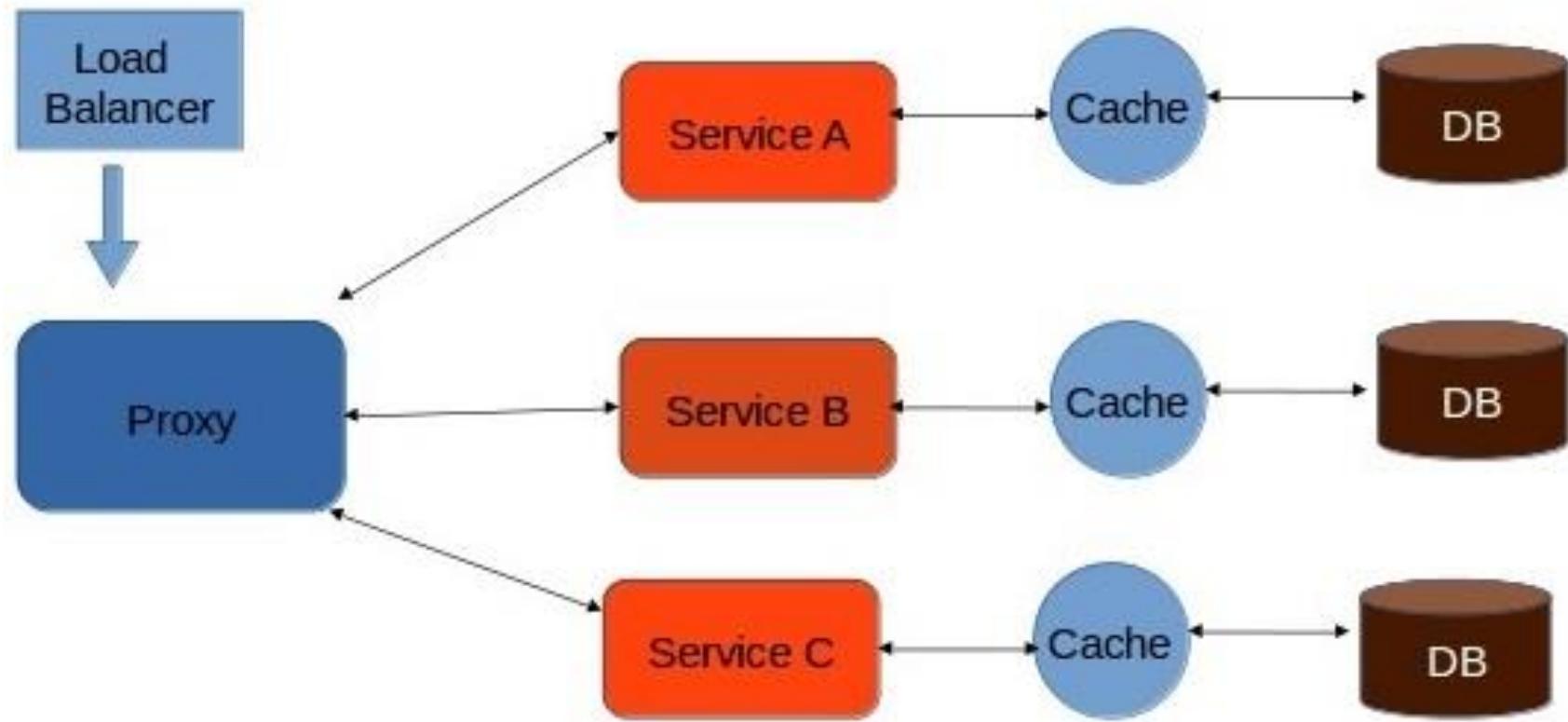


# Proxy Pattern

- Proxy microservice design pattern is a variation of Aggregator.
- Proxy means ‘in place of’, representing’ or ‘in place of’ or ‘on behalf of’ are literal meanings of proxy and that directly explains Proxy Design Pattern.
- Proxies are also called surrogates, handles, and wrappers.
- They are closely related in structure, but not purpose, to Adapters and Decorators.
- A real world example can be a cheque or credit card is a proxy for what is in our bank account. It can be used in place of cash, and provides a means of accessing that cash when required. And that's exactly what the Proxy pattern does – “Controls and manage access to the object they are protecting”.



# Proxy Pattern





## UI composition pattern

---

- The end user interface layer is laid out into various sections, which individually invokes the corresponding microservice asynchronously.
- Modern user interfaces use single page application (SPA) built by Angular or ReactJS frameworks.



## Backend for frontend

---

- Instead of creating a general-purpose microservice, we can design a microservice and its response specifically for the client agents (such as desktop browsers, mobile devices etc.).
- This tight coupling of client agents with the corresponding backend service helps us to efficiently create response data.



# Gateway Routing Pattern

- The API gateway is responsible for request routing.
- An API gateway implements some API operations by routing requests to the corresponding service.
- When it receives a request, the API gateway consults a routing map that specifies which service to route the request to.
- A routing map might, for example, map an HTTP method and path to the HTTP URL of service.
- This function is identical to the reverse proxying features provided by web servers such as NGINX.

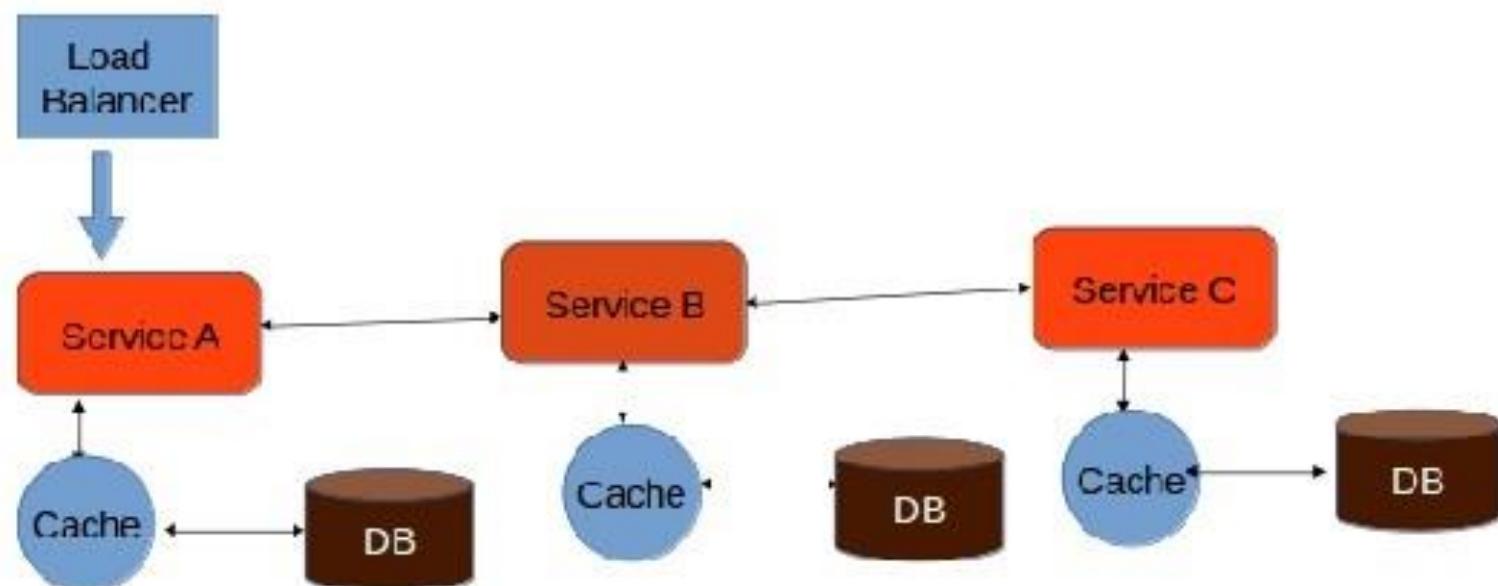


# Chained Micro Service Pattern

- There will be multiple dependencies of for single services or microservice eg: Sale microservice has dependency products microservice and order microservice.
- Chained microservice design pattern will help you to provide the consolidated outcome to your request.
- The request received by a microservice: 1, which is then communicating with microservice-2 and it may be communicating with microservice-3.
- All these services are synchronous calls.

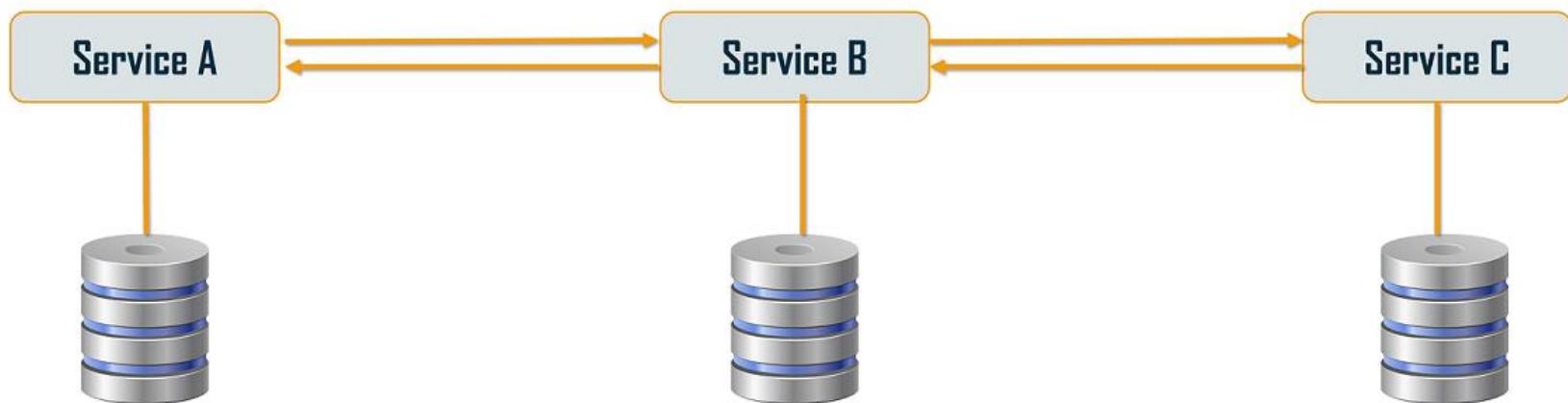


# Chained Micro Service Pattern





## Chained or Chain of Responsibility Pattern

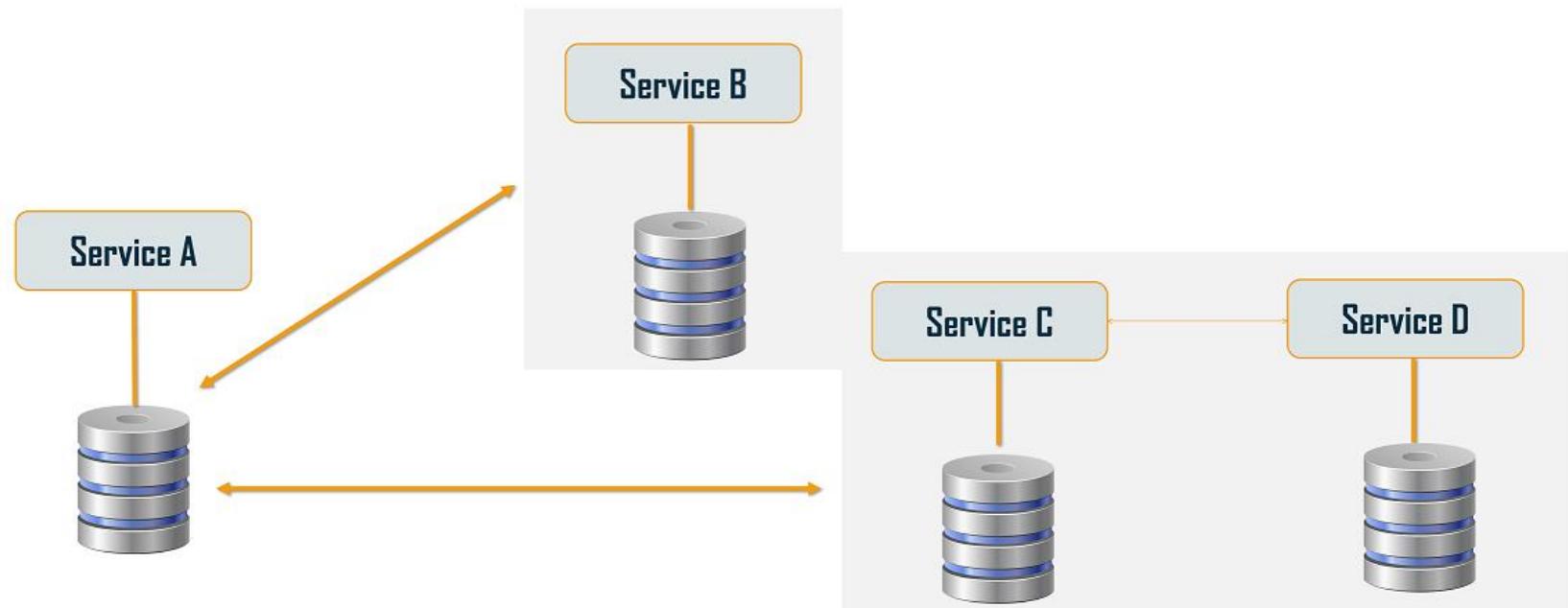




# Branch Pattern

- Branch microservice is a design pattern in which you can simultaneously process the requests and responses from two or more independent microservices.
- So, unlike the chained design pattern, the request is not passed in a sequence, but the request is passed to two or more mutually exclusive microservices chains.
- This design pattern extends the Aggregator design pattern and provides the flexibility to produce responses from multiple chains or single chain.
- For example, if you consider an e-commerce application, then you may need to retrieve data from multiple sources and this data could be a collaborated output of data from various services.
- So, you can use the branch pattern, to retrieve data from multiple sources.

# Branch Pattern



# Database Patterns



# DATA-RELATED PATTERNS

---

- As microservices are self-contained and designed for independent scalability, we end up having service-specific databases (such as database server per service, database schema per service and service-specific tables).
- Due to this design, we face challenges such as:
  - A single service that reads or updates data from multiple databases
  - A single business transaction spanning multiple services and databases
  - Replication of data in the databases



## Database per Service

---

- To solve the above concerns, one database per microservice must be designed; it must be private to that service only.
- It should be accessed by the microservice API only.
- It cannot be accessed by other services directly.
- For example, for relational databases, we can use private-tables-per-service, schema-per-service, or database-server-per-service.

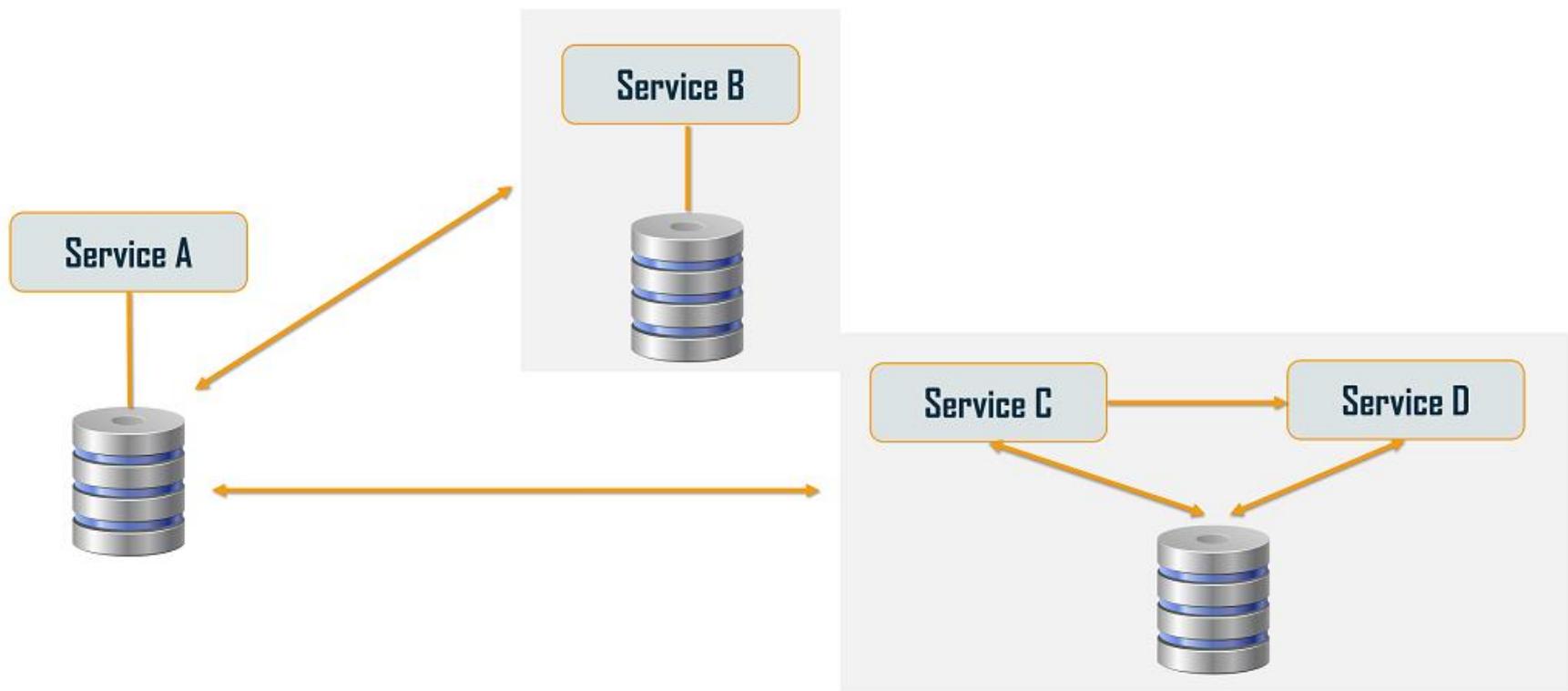


## Shared Database per service

---

- It is an anti-pattern for microservices.
- But if the application is a monolith and trying to break into microservices, denormalization is not that easy.
- In the later phase, we can move to DB per service pattern.
- A shared database per service is not ideal, but that is the working solution for the above scenario.
- Most people consider this an anti-pattern for microservices, but for brownfield applications, this is a good start to break the application into smaller logical pieces.
- This should not be applied for greenfield applications.

## Database or Shared Data Pattern





# Event Sourcing

---

- The event sourcing pattern is generally used along with it to create events for any data change.
- Materialized views are kept updated by subscribing to the stream of events.
- Event Sourcing Most applications work with data, and the typical approach is for the application to maintain the current state.
- For example, in the traditional create, read, update, and delete (CRUD) model a typical data process is to read data from the store.
- It contains limitations of locking the data, often using transactions.

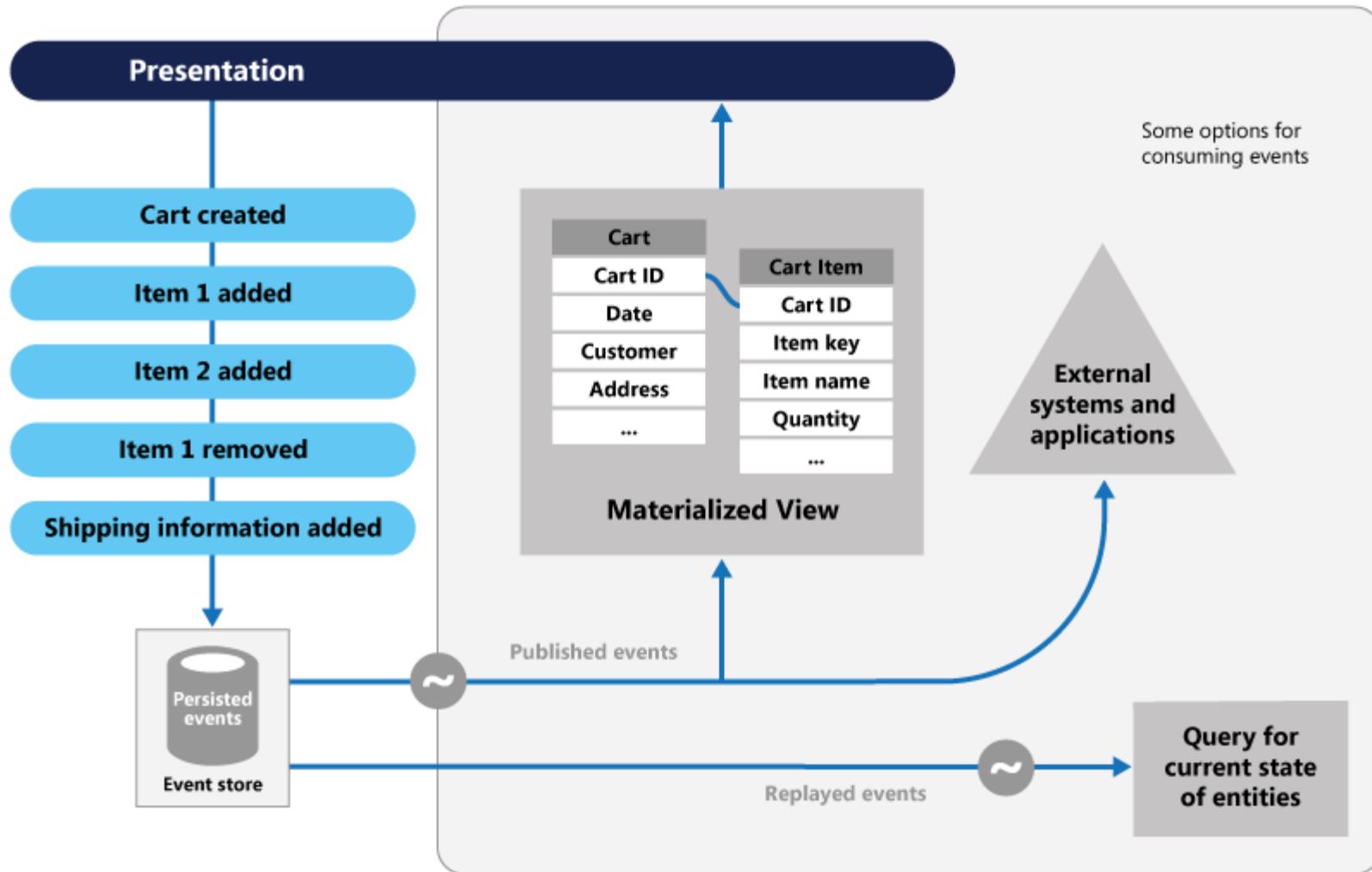


# Event Sourcing

- The Event Sourcing pattern defines an approach to handle operations on data that's driven by a sequence of events, each of which is recorded in an append-only store.
- Application code sends a series of events that imperatively describe each action that has occurred on the data to the event store, where they're persisted.
- Each event represents a set of changes to the data (such as `AddedItemToOrder`).
- The events are persisted in an event store that acts as the system of record.
- Typical uses of the events published by the event store are to maintain materialized views of entities as actions in the application change them, and for integration with external systems.
- For example, a system can maintain a materialized view of all customer orders that are used to populate parts of the UI.
- As the application adds new orders, adds or removes items on the order, and adds shipping information, the events that describe these changes can be handled and used to update the materialized view.

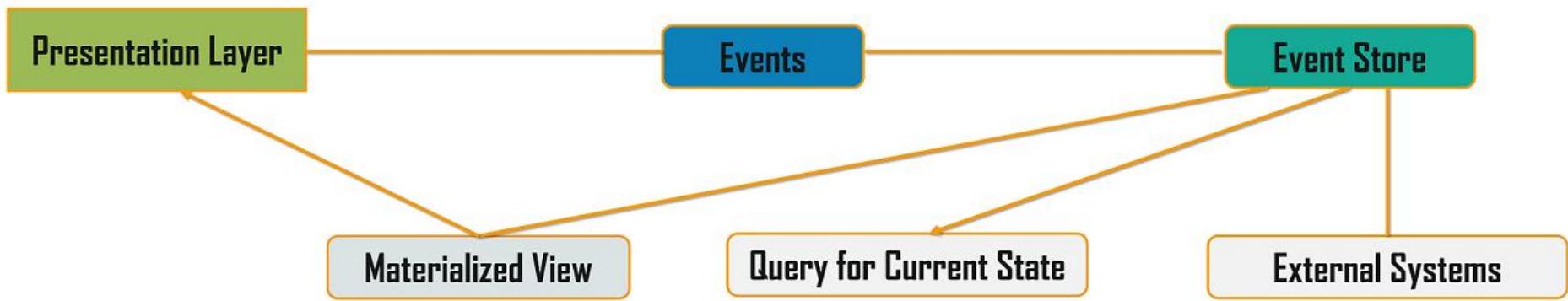


# Event Sourcing





## Event Sourcing Design Pattern



# Command Query Responsibility Segregation (CQRS)

---



- Database handling is split into two categories, the **command part** for handling data creation, update, deletion and the **query part** that uses materialized views to retrieve data.
- The materialized view is updated by subscribing to data change events.
- Event sourcing pattern is used along with CQRS to create immutable events.

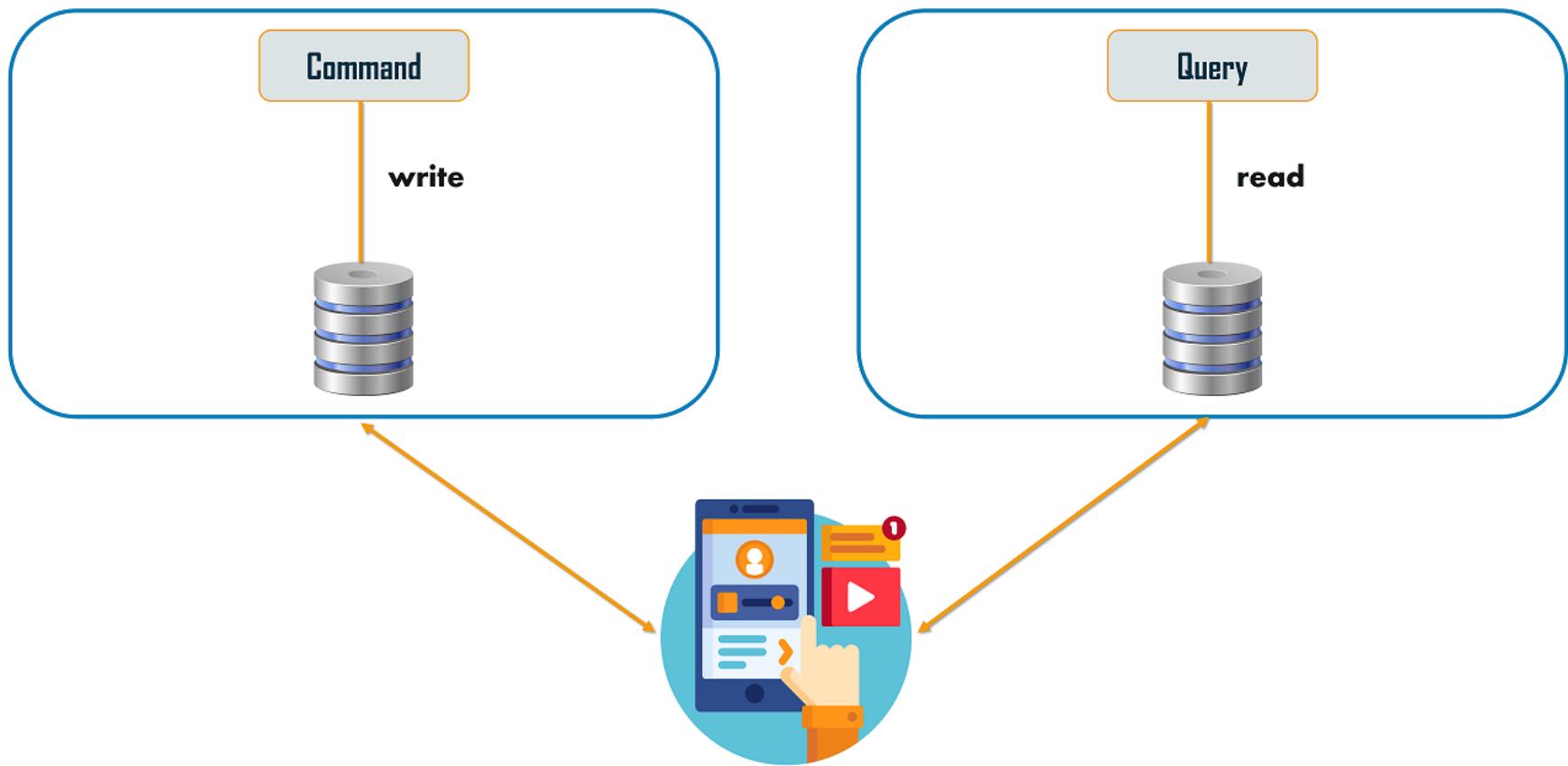
# Command Query Responsibility Segregation (CQRS)

---



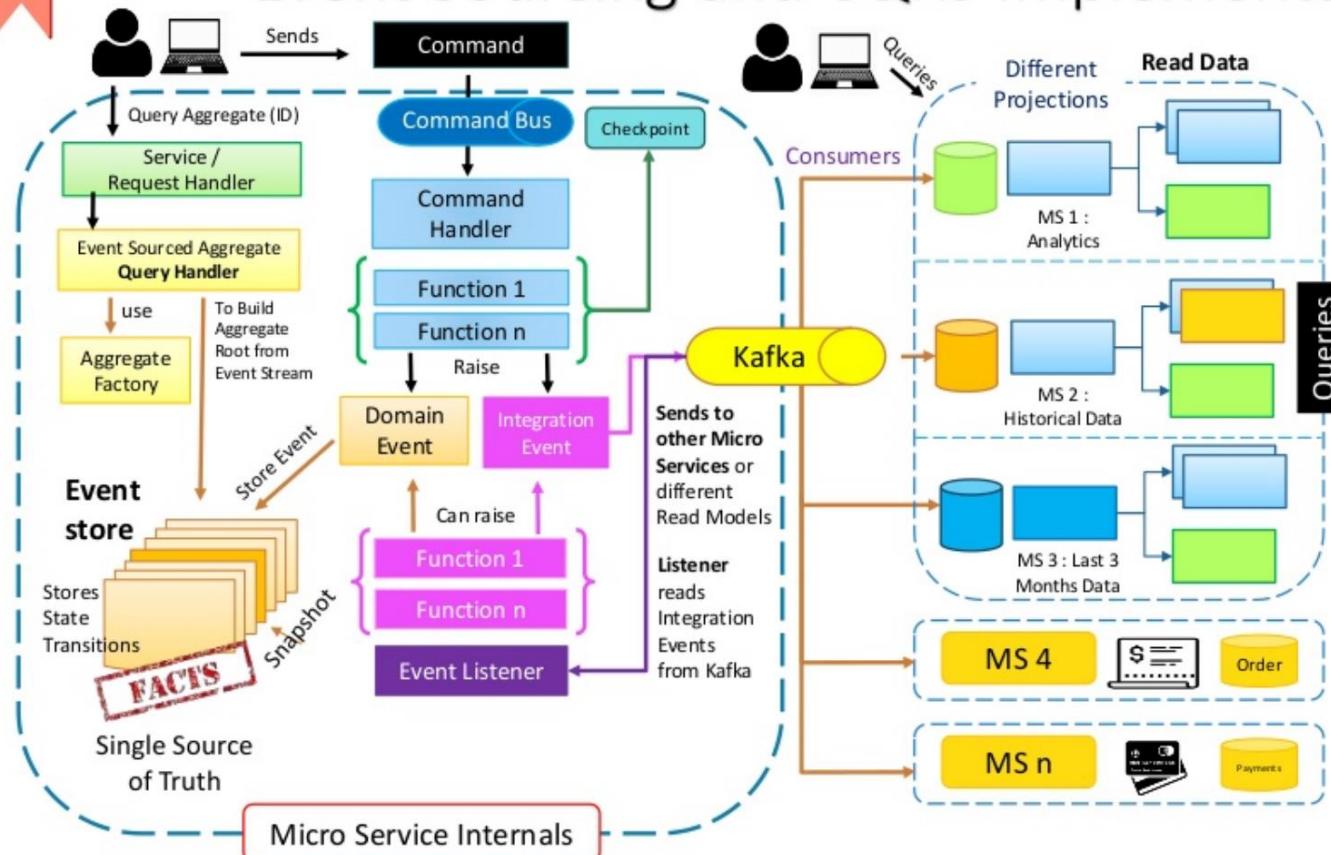
- Once we implement database-per-service, there is a requirement to query, which requires joint data from multiple services.
- CQRS suggests splitting the application into two parts—the command side and the query side.
- The command side handles the Create, Update, and Delete requests.
- The query side handles the query part by using the materialized views.

## Command Query Responsibility Segregator (CQRS) Design Pattern



2.3

## Event Sourcing and CQRS Implementation

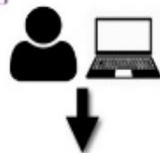
[Clip slide](#)


### Advantages of ES/CQRS

1. Trusted Audit Trail
2. Debugging
3. Historical State
4. Variant Schemas
5. Distribution Support
6. In-Memory Processing
7. Alternative History

**5.5**

## Use Case : Shopping Site – Event Sourcing / CQRS

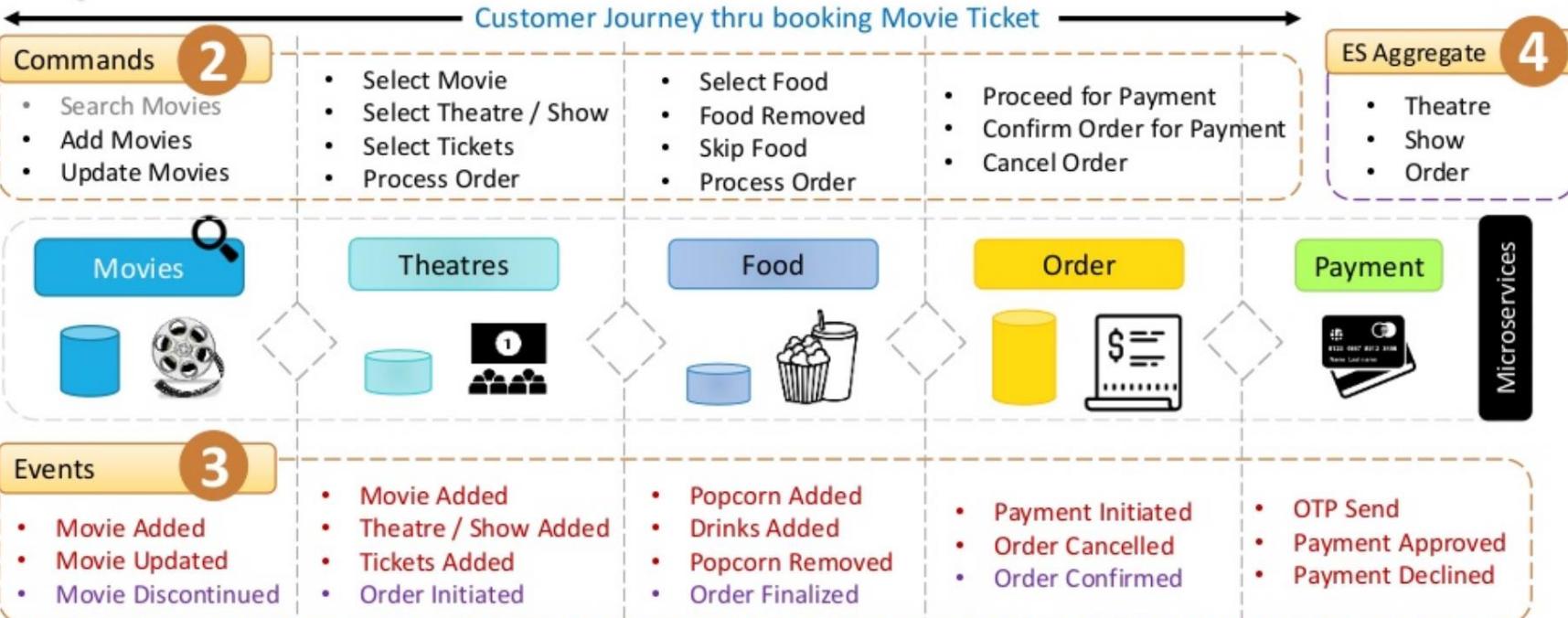
[Clip slide](#)


Commands are End-User interaction with the App and based on the commands (Actions) Events are created. These Events includes both **Domain Events** and **Integration Events**. **Event Sourced Aggregates** will be derived using Domain Events. Each Micro Service will have its own separate Database. Depends on the scalability requirement each of the Micro Service can be scaled separately. For Example. Catalogue can be on a 50 node cluster compared to Customer Micro Service.



**5.6**
**1**


# Use Case : Movie Booking – Event Sourcing / CQRS

Clip slide


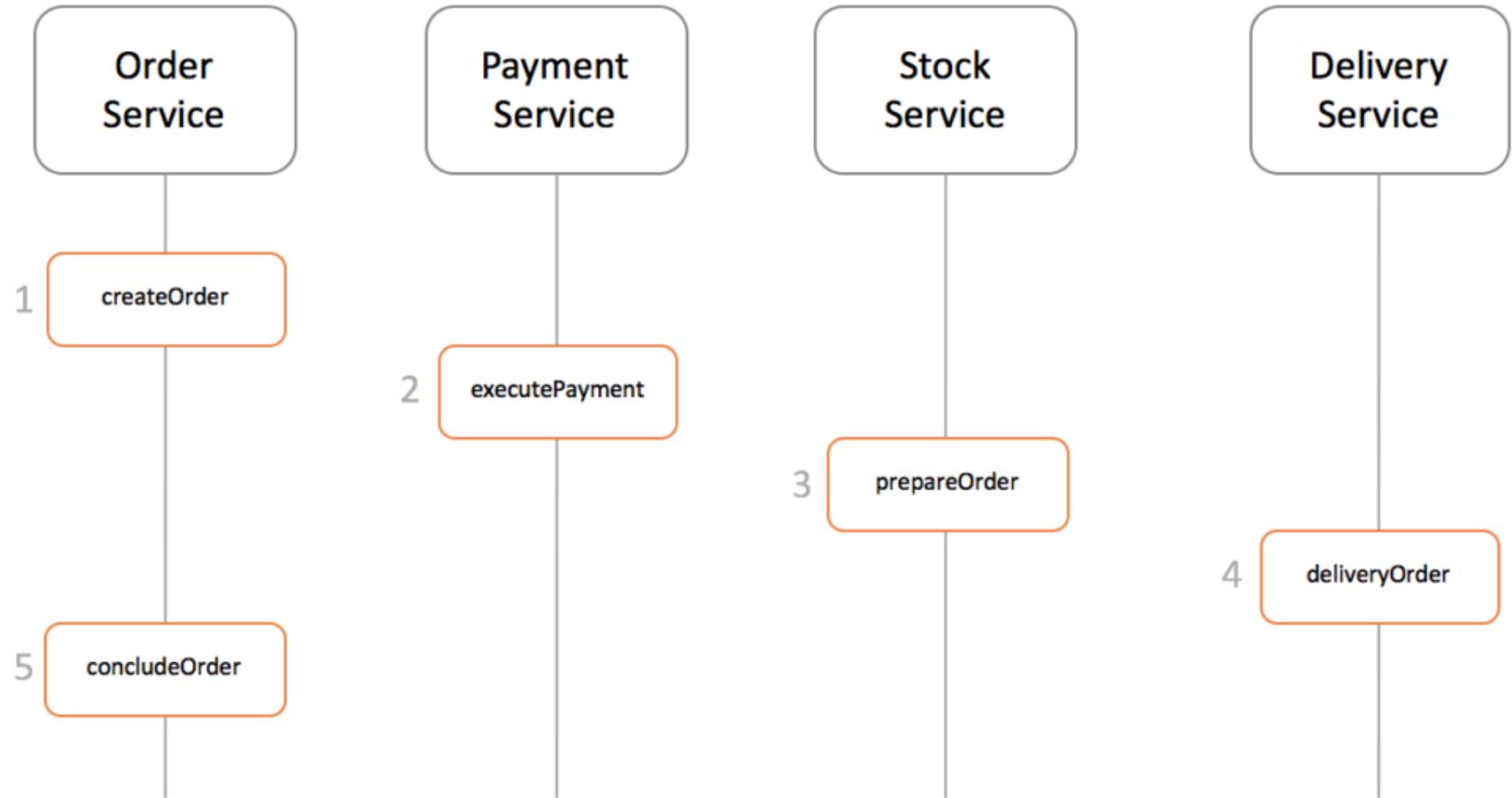


# Saga Pattern

- When a business transaction needs to manage data consistency that is spread across multiple databases, we could use Saga pattern.
- As a part of the Saga pattern, each transaction is orchestrated locally or centrally to execute it entirely and handle the failure/rollback scenario.
- For instance, if a business transaction needs to handle data related to order and customer service, each of these services produces and listens to each other to handle the transaction.



# Saga Design Pattern



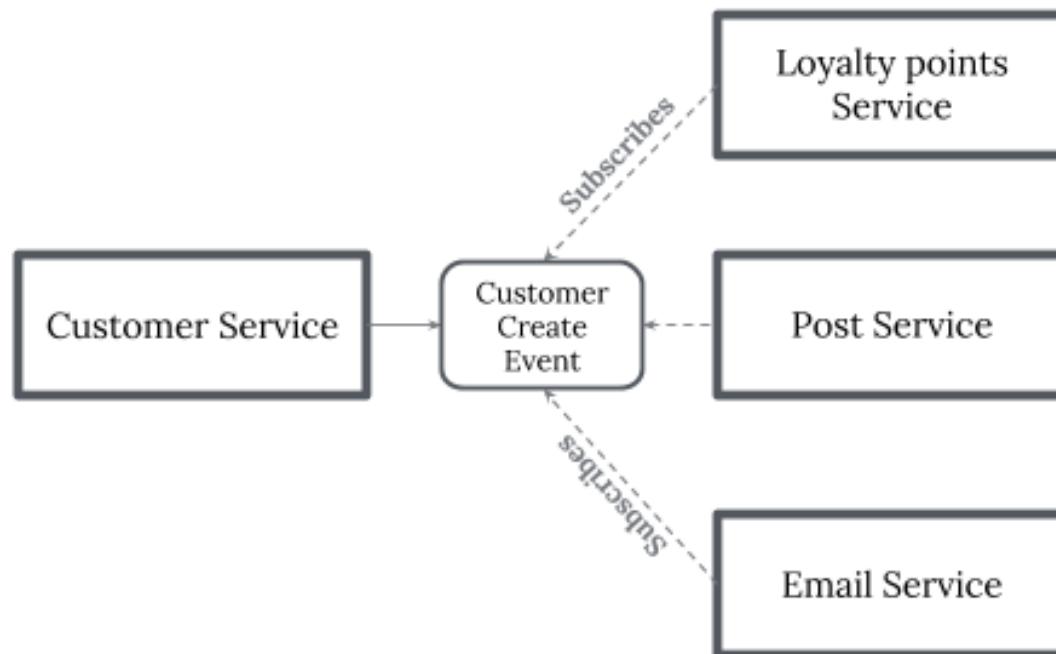


# Saga Pattern

- When each service has its own database and a business transaction spans multiple services, how do we ensure data consistency across services?
- Each request has a compensating request that is executed when the request fails.
- It can be implemented in two ways:
- Choreography—When there is no central coordination, each service produces and listens to another service's events and decides if an action should be taken or not.
- Choreography is a way of specifying how two or more parties; none of which have any control over the other parties' processes, or perhaps any visibility of those processes—can coordinate their activities and processes to share information and value.

# Saga Pattern

- Use choreography when coordination across domains of control/visibility is required.
- It dictates acceptable patterns of requests and responses between parties.





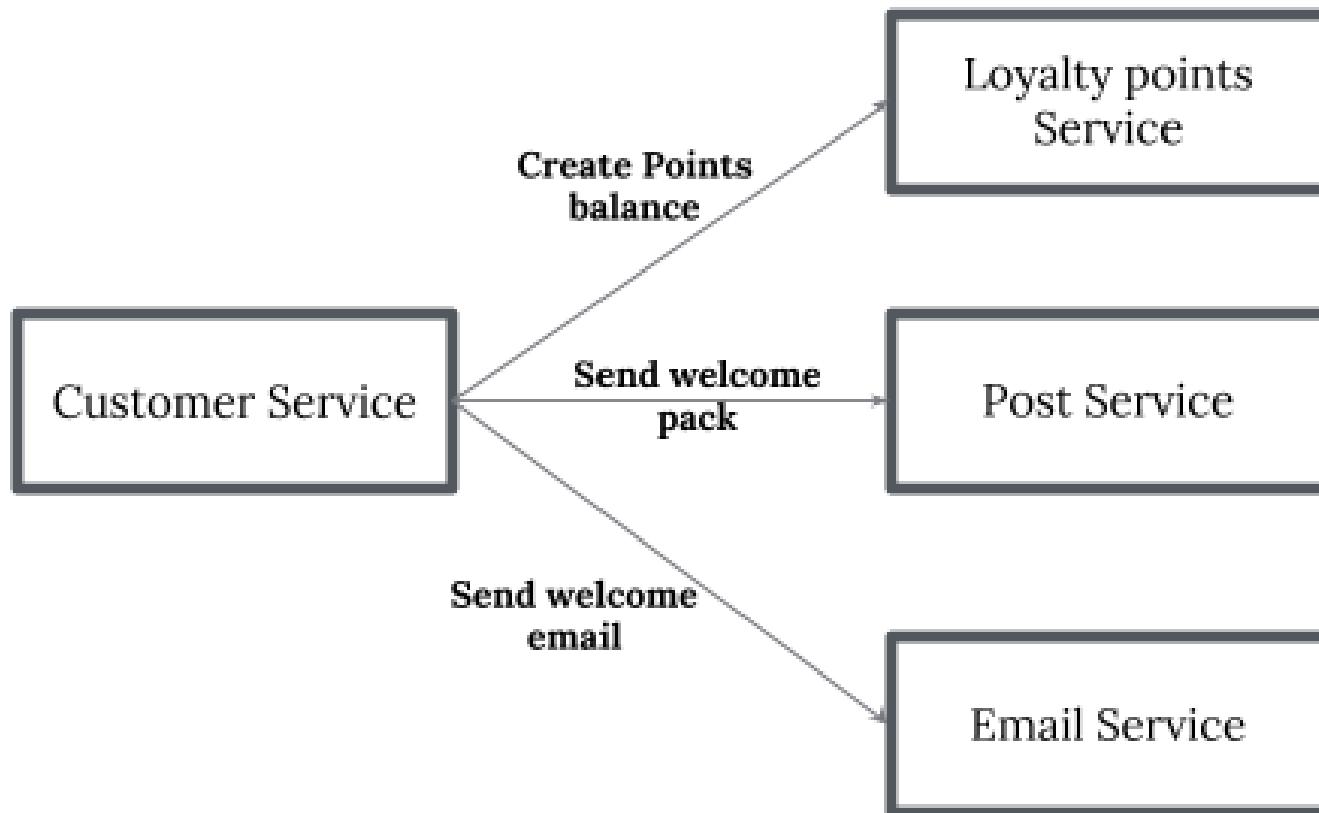
# Saga Pattern

---

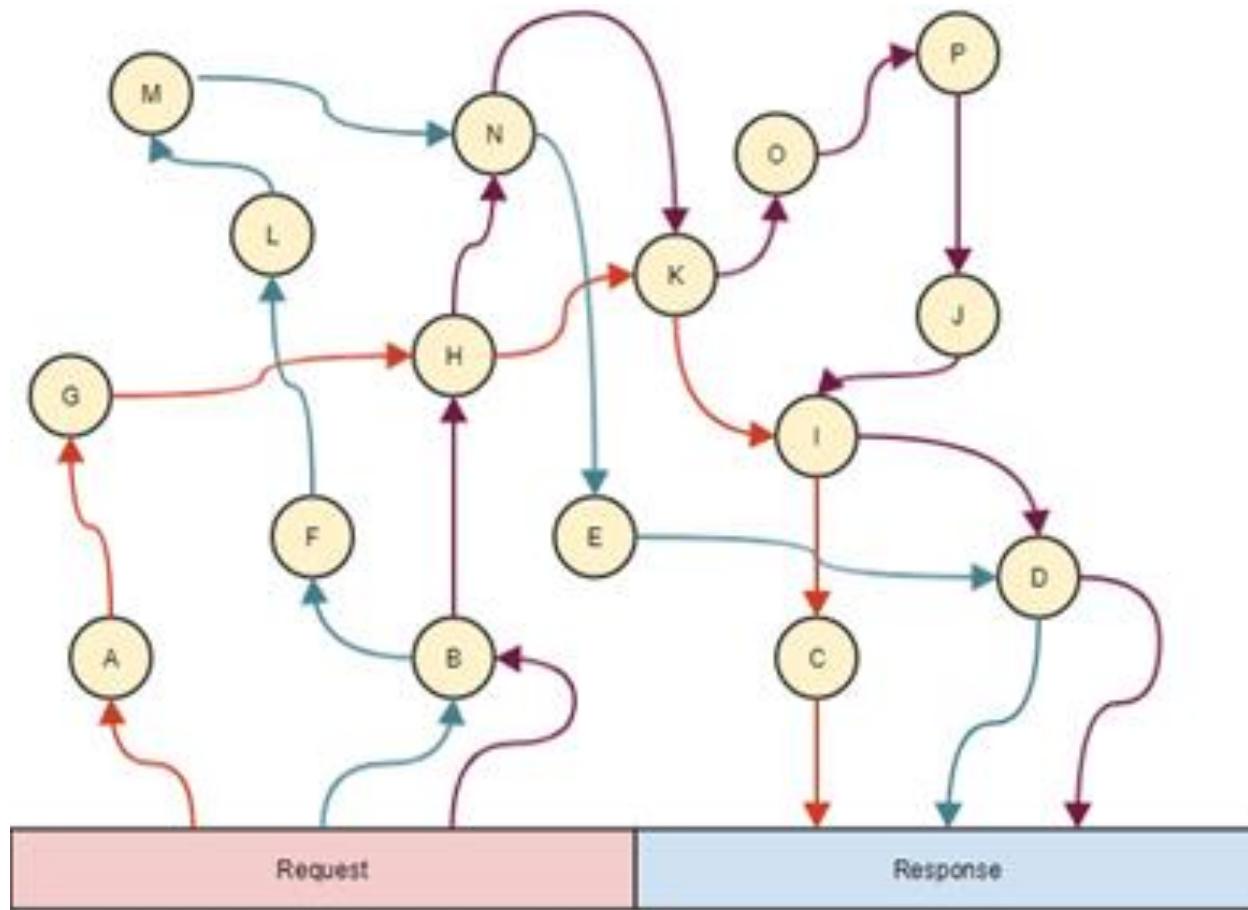
- Orchestration—An orchestrator (object) takes responsibility for a saga's decision making and sequencing business logic.
- when you have control over all the actors in a process.  
when they're all in one domain of control and you can control the flow of activities.
- This is, of course, most often when you're specifying a business process that will be enacted inside one organization that you have control over.



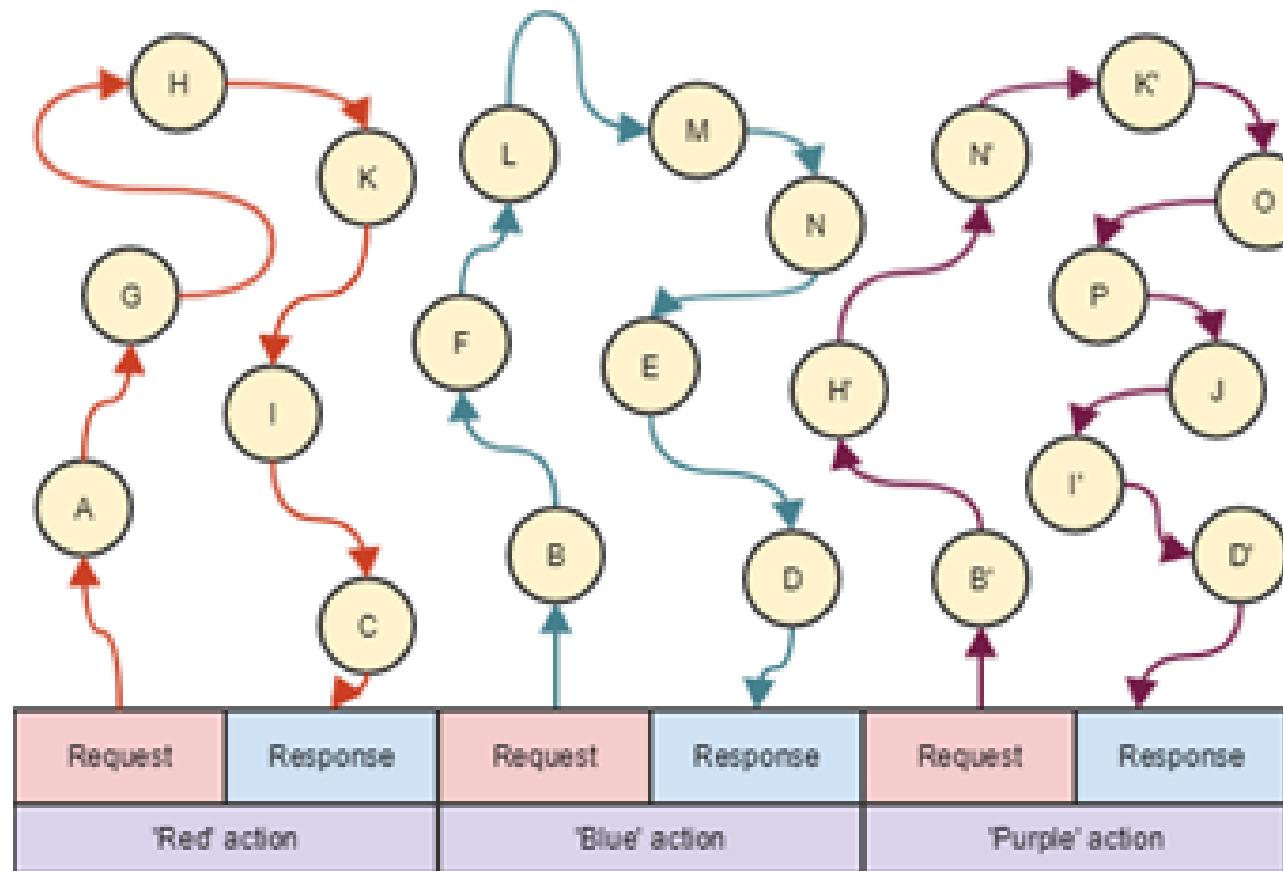
# Saga Pattern



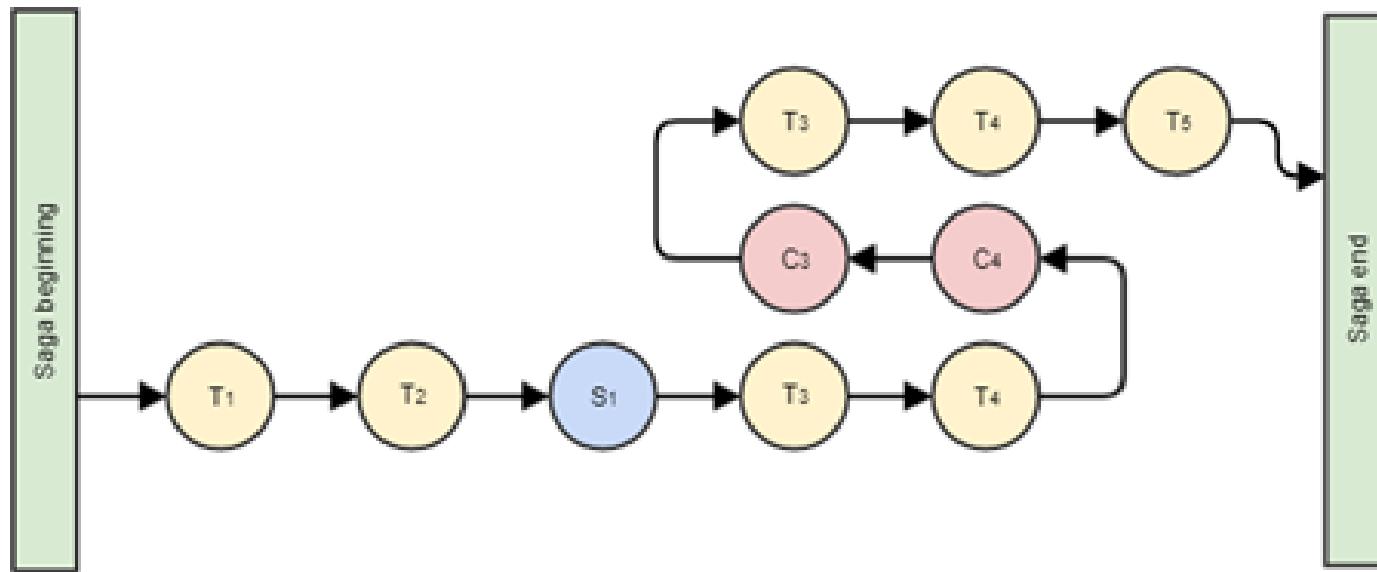
# Saga Pattern



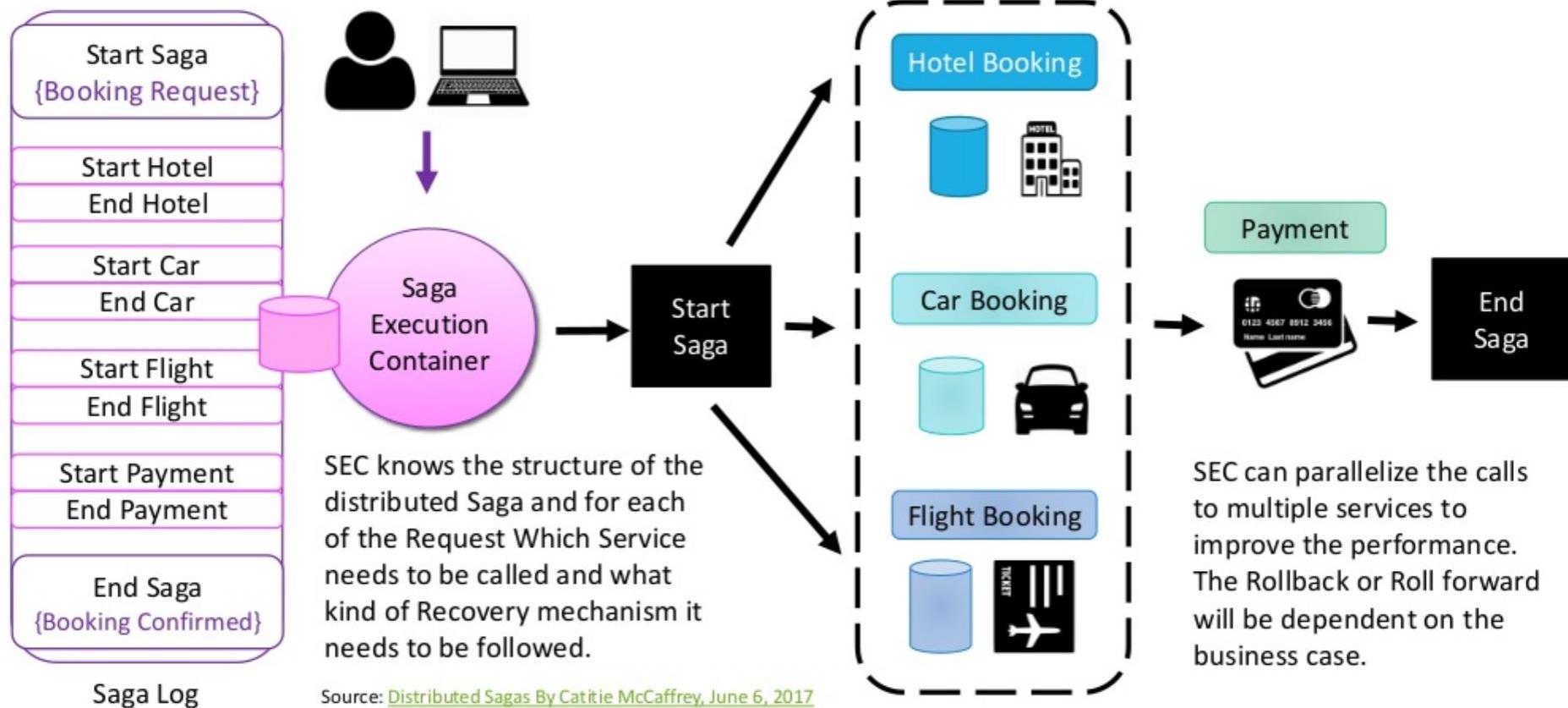
# Saga Pattern



# Saga Pattern



## Use Case : Travel Booking – Distributed Saga (SEC)



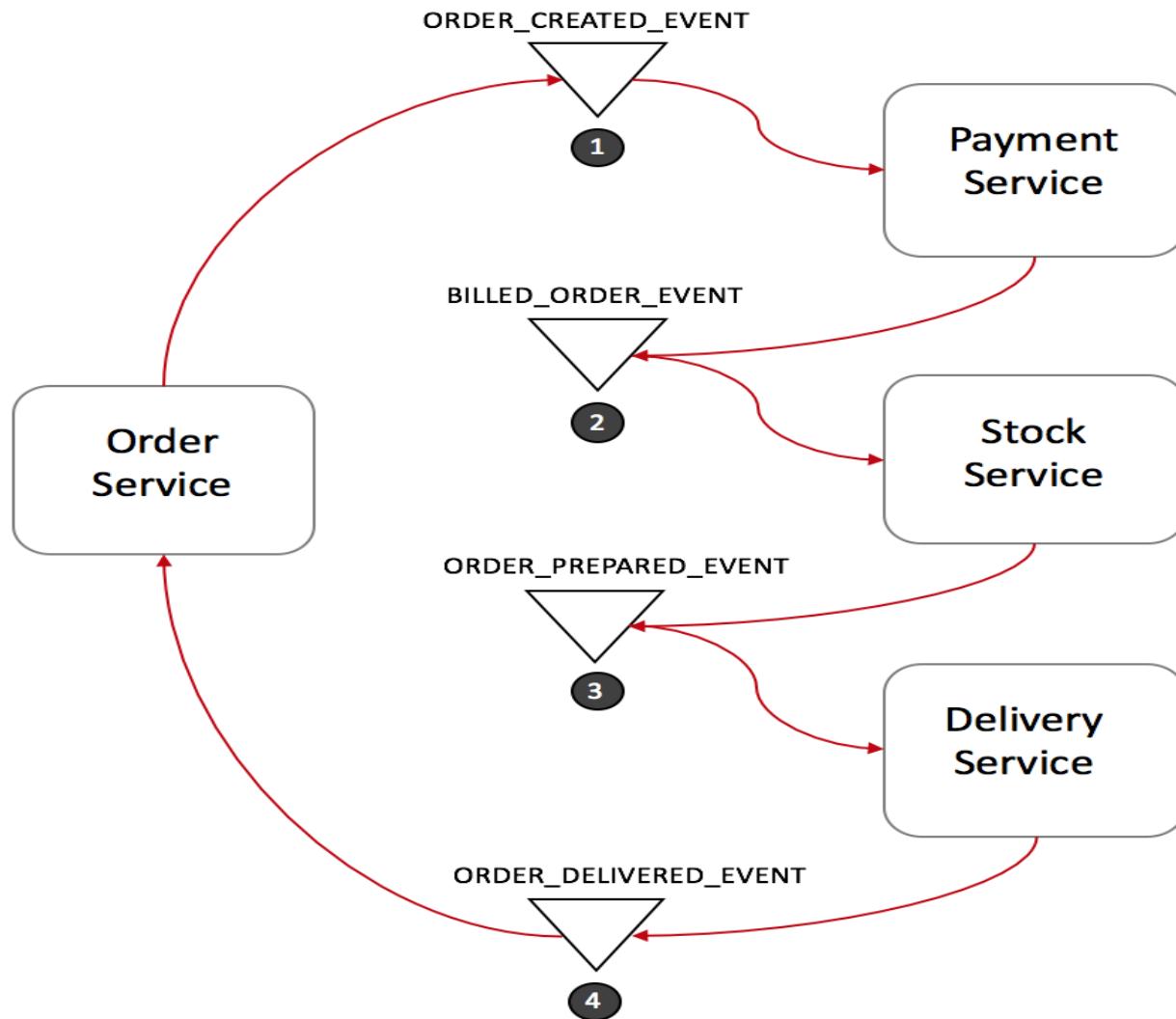


# Saga Design Pattern

- Events/Choreography: When there is no central coordination, each service produces and listen to other service's events and decides if an action should be taken or not.
- Command/Orchestration: when a coordinator

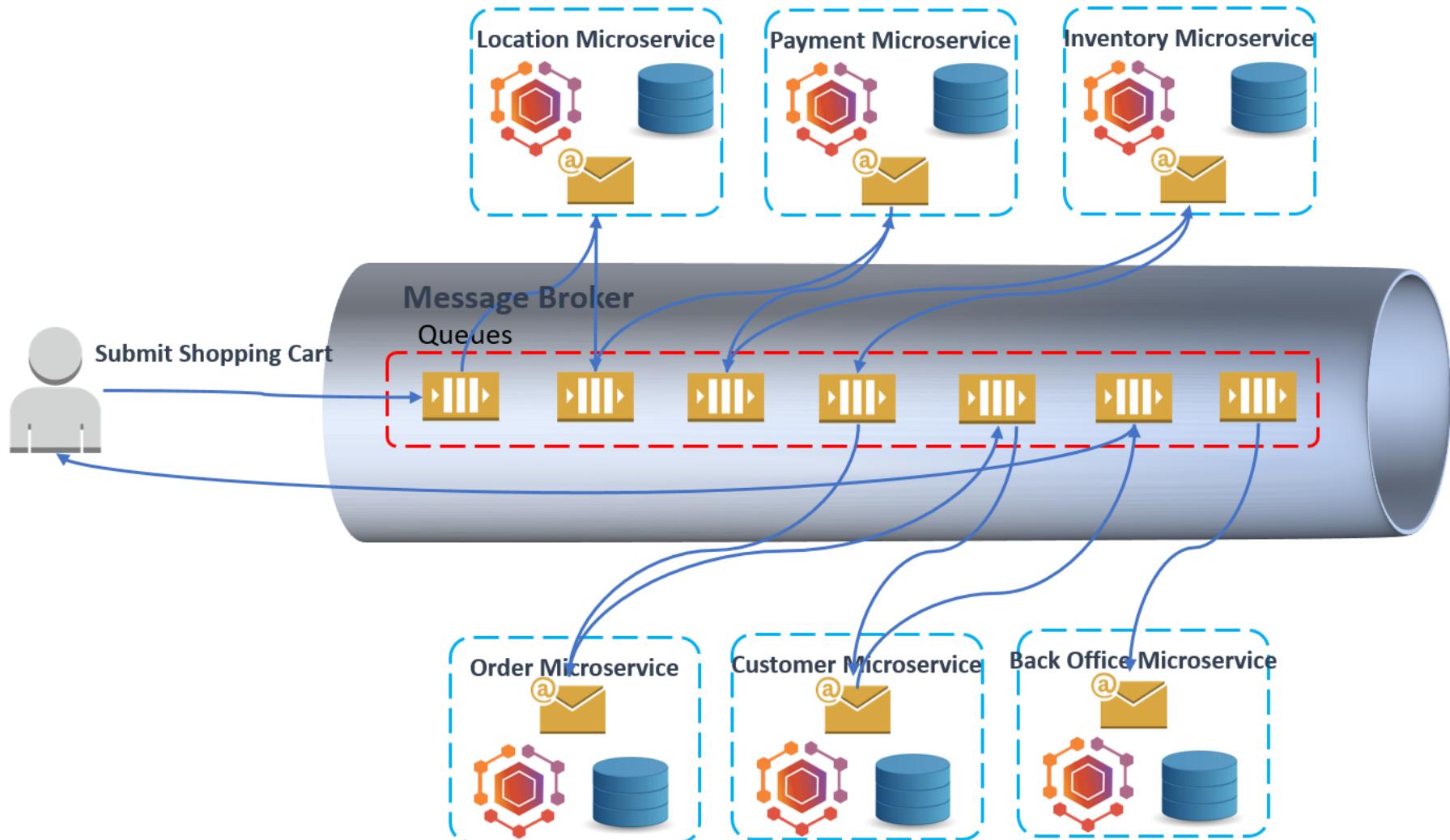


# Event Choreography





# Event Choreography



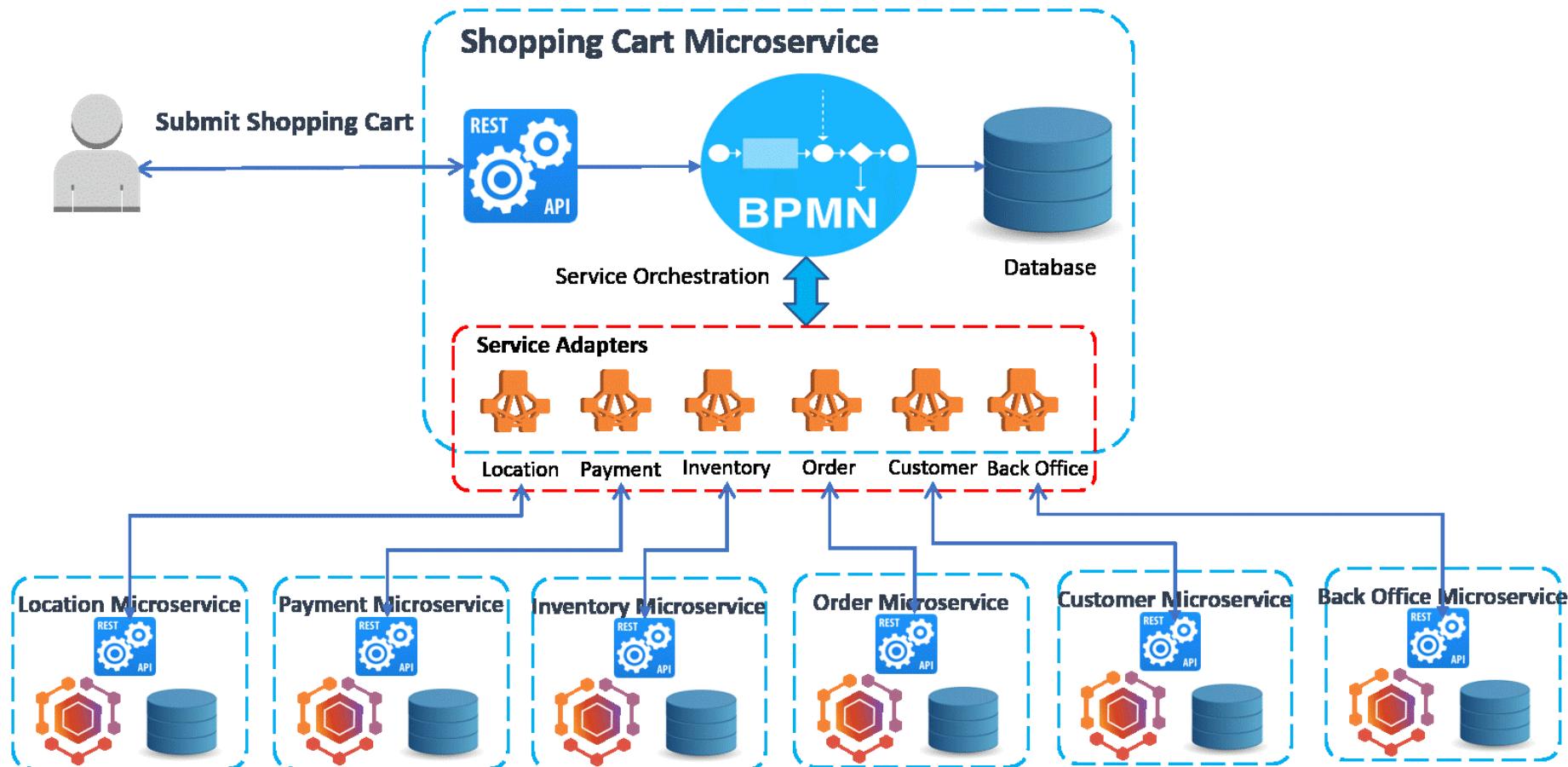
# Event-Driven Service Choreography



- Difficult to maintain: Due to the decentralized solution, the business flow is spreading across multiple services. Any business change to the process flow will lead to changes in multiple services.
- Difficult to manage: The runtime state of the process instance is stored separately.
- Difficult to rollback: The business process is choreographed by multiple services which leave the transaction's ACID becomes extremely difficult.



# Centralized Service Orchestration





# Centralized Service Orchestration

- Easy to maintain: the business flow is modeled as graphical BPMN process in a centralized orchestration microservice. The standard Business Process Model and Notation (BPMN) grants the businesses with the capability of understanding their internal business procedures in a graphical notation. Furthermore, organizations can better communicate and report these procedures in a standard manner. Any further changes to the process flow will be mitigated by implementing the workflow diagrams.
- Easy to manage: BPMN process engine keeps track of all process instances, their state, audit and statistics information.
- Easy to rollback: Transaction's ACID can be guaranteed by the compensation flow as the BPMN out-of-the-box feature.

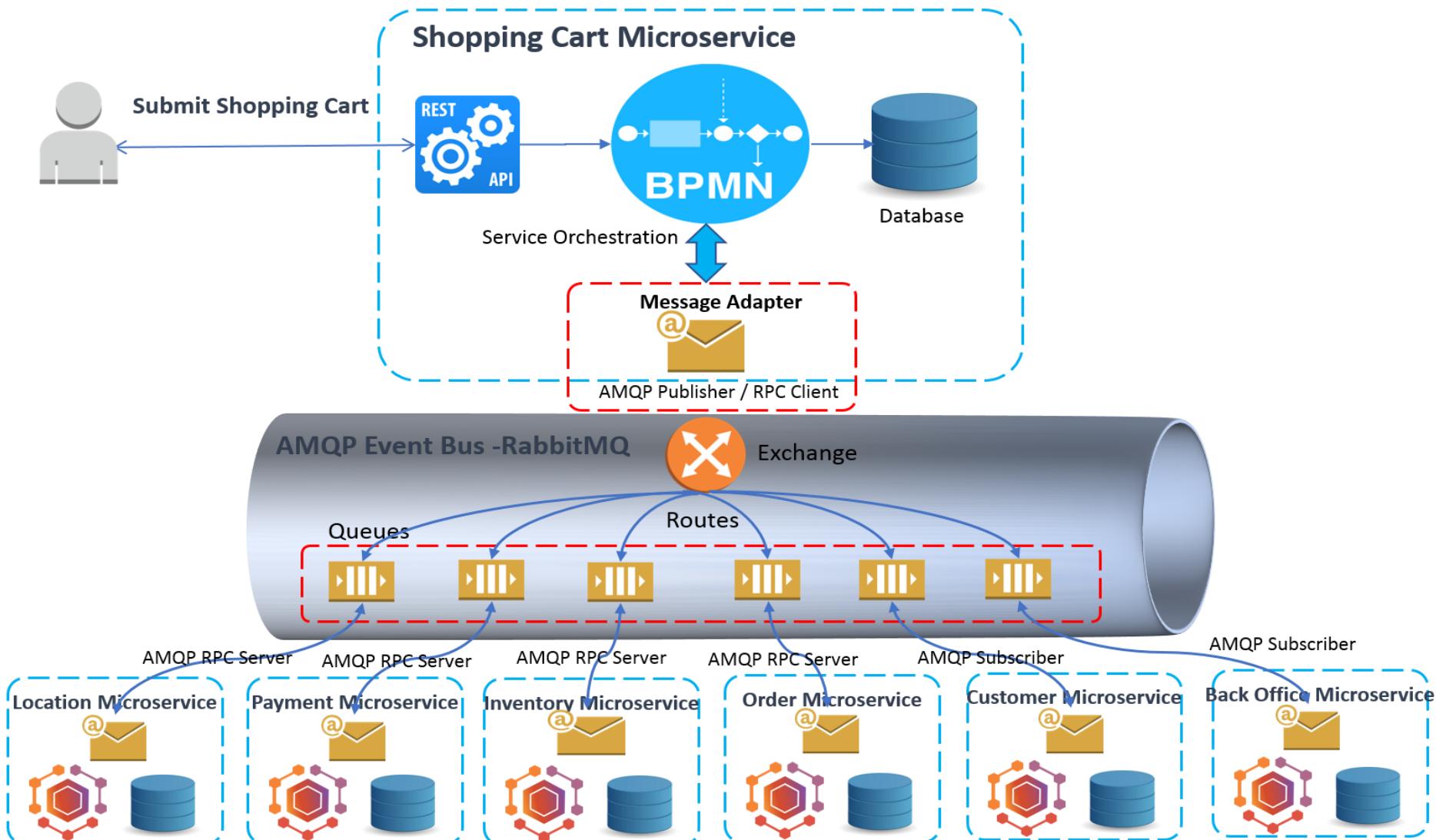


# Centralized Service Orchestration

- Tight-coupling: The orchestration pattern must build and maintain a point-to-point connection between the services. Point-to-point means that one service calls the API of another service which results in a mesh of communication paths between all services. Integrating, changing or removing services from the service repository will be a hard task since you must be aware of each connection between the services.
- Complexity in building service adapters: Orchestration service need to develop adapters to the peer services, which must maintain all details of the service communications, such as service location, service interfaces, and data model translation. Whenever a peer service changes, the adapter will be impacted.



# Centralized Service Orchestration



# Observability Pattern



# Log Aggregation

---

- Since microservices are deployed into individual containers, the logs generated by each of the containers (a.k.a pods) need to be aggregated to create a centralized log repository.
- Microservices can log to standard output or to a log file.
- The log management systems such as Splunk or Kibana can aggregate the log stream in real time to a centralized log repository and we can query the real-time logs.



# Performance Metrics

---

- Performance monitoring services such as Prometheus, AppDynamics and NewRelic can be used to monitor the performance metrics of microservices.
- The performance metrics are depicted visually and we can configure the performance thresholds and notification triggers.



# Distributed Tracing

---

- When the request flows across various layers and microservices, it is necessary to trace the request end-to-end for error handling and for performance troubleshooting scenarios.
- In distributed tracking, we create a unique request ID (such as x-request-id) that is passed across all layers and microservices and logged for troubleshooting purposes.



## Health Checks

---

- In order to properly distribute the load and route the traffic accordingly, each microservice has to publish health check endpoint (such as /health) that provides the status of the overall health of the service.
- The health check service should check the status of dependent systems (such as databases, storage systems) and host connectivity to provide the overall health status of the service.

# CROSS-CUTTING CONCERN PATTERNS



# External Configuration

---

- All environment-specific configurations such as connection strings, application properties and URLs should be loaded from an external configuration file.
- The CI/CD pipeline can inject the environment-specific configuration values during the build.

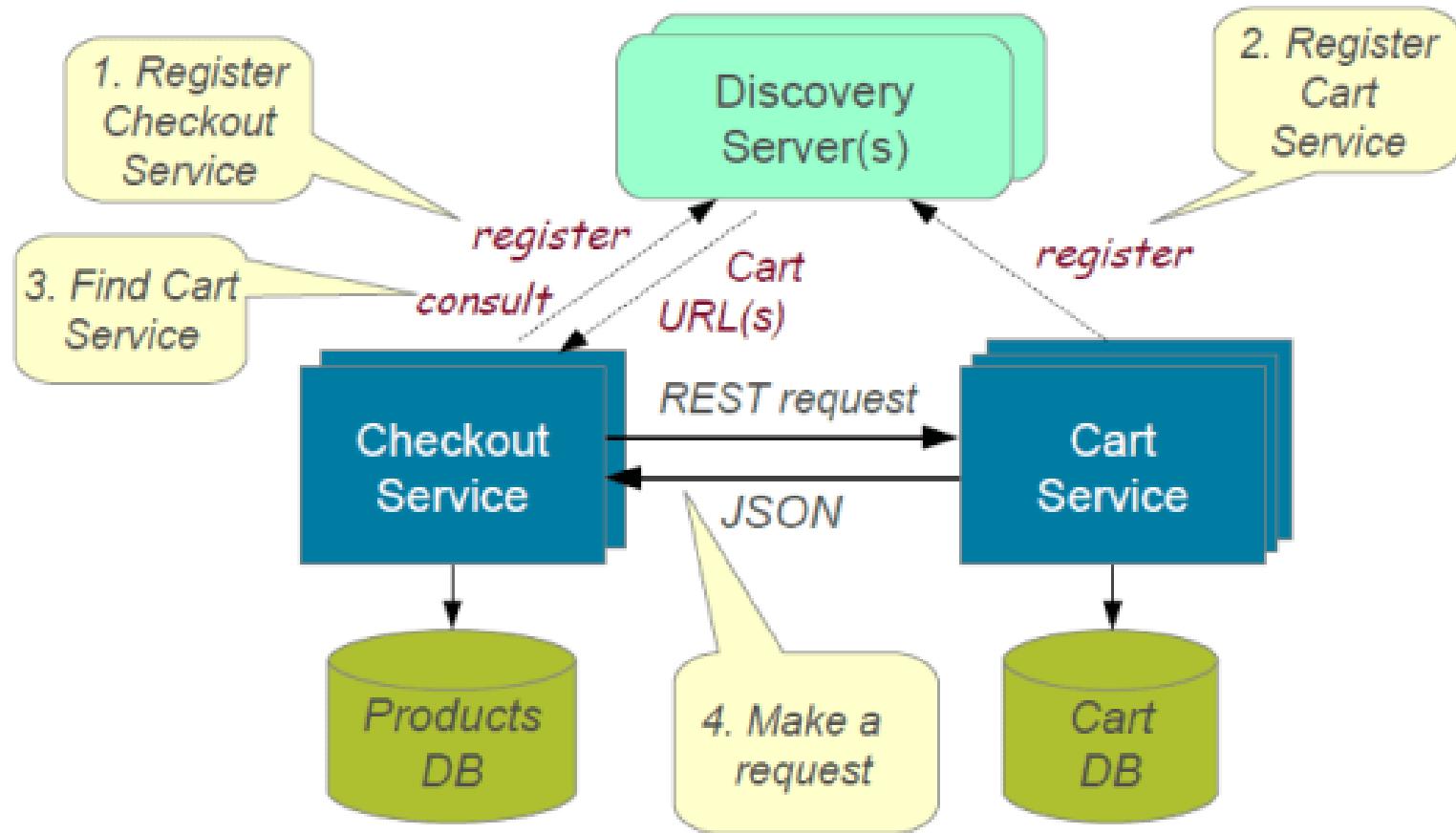


# Service Discovery

---

- Centralized service discovery module should handle the responsibilities such as service registration/ de-registration and request routing, based on service health.
- For client side service discovery service registry is used for load balancing and for server side service discovery, server side load balancing is used.

# Service Discovery

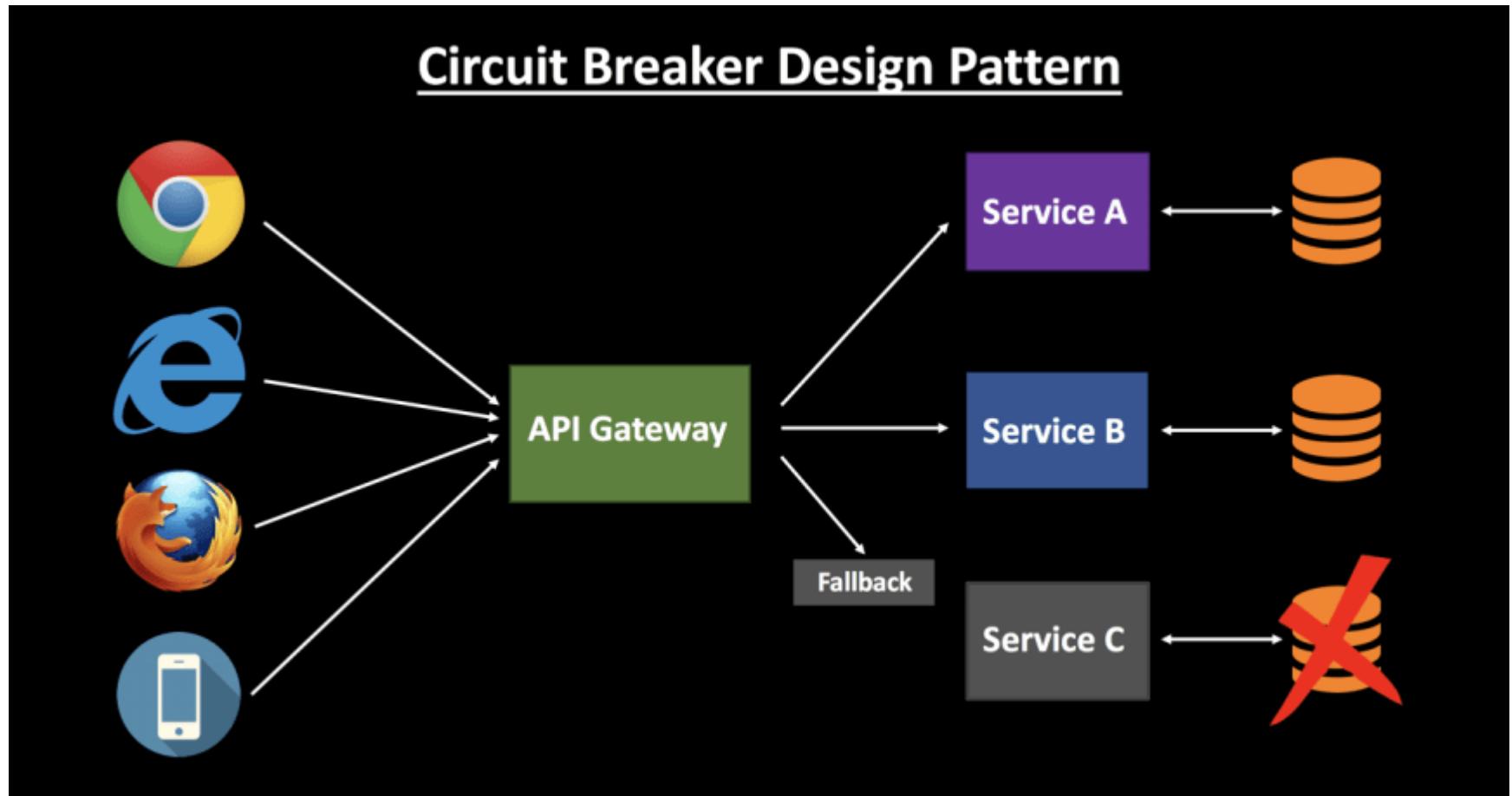




# The circuit breaker

---

- When one of the services in the request-processing pipeline fails, the circuit breaker is responsible in terms of handling the failure and preventing the cascading of the error.
- The circuit breaker can monitor the error from a dependent service and fallback to a default handler in case of error.
- Netflix Hystrix is an example of a circuit breaker



# Solutions to Challenges with Microservice Architectures



- **Spring Boot**
- *Enable building production ready applications quickly*
- Provide non-functional features
- embedded servers (easy deployment with containers)
- metrics (monitoring)
- health checks (monitoring)
- externalized configuration

# Examples of Microservices Frameworks for Java



- There are several microservices frameworks that you can use for developing for Java. Some of these are:
- **Spring Boot.** This is probably the best Java microservices framework that works on top of languages for Inversion of Control, Aspect Oriented Programming, and others.
- **Jersey.** This open source framework supports JAX-RS APIs in Java is very easy to use.
- **Swagger.** Helps you in documenting API as well as gives you a development portal, which allows users to test your APIs.



## Pros

---

- Microservice architecture gives developers the freedom to independently develop and deploy services
- A microservice can be developed by a fairly small team
- Code for different services can be written in different languages (though many practitioners discourage it)
- Easy integration and automatic deployment (using open-source continuous integration tools such as Jenkins, Hudson, etc.)



## Pros

---

- Easy to understand and modify for developers, thus can help a new team member become productive quickly
- The developers can make use of the latest technologies
- The code is organized around business capabilities
- Starts the web container more quickly, so the deployment is also faster



## Pros

---

- When change is required in a certain part of the application, only the related service can be modified and redeployed—no need to modify and redeploy the entire application
- Better fault isolation: if one microservice fails, the other will continue to work (although one problematic area of a monolith application can jeopardize the entire system)
- Easy to scale and integrate with third-party services
- No long-term commitment to technology stack



## Cons

---

- Due to distributed deployment, testing can become complicated and tedious
- Increasing number of services can result in information barriers
- The architecture brings additional complexity as the developers have to mitigate fault tolerance, network latency, and deal with a variety of message formats as well as load balancing
- Being a distributed system, it can result in duplication of effort



## Cons

---

- When number of services increases, integration and managing whole products can become complicated
- In addition to several complexities of monolithic architecture, the developers have to deal with the additional complexity of a distributed system
- Developers have to put additional effort into implementing the mechanism of communication between the services



## Cons

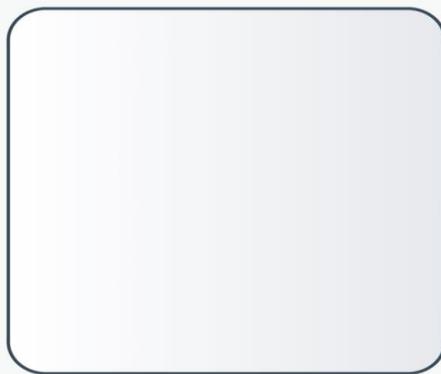
---

- Handling use cases that span more than one service without using distributed transactions is not only tough but also requires communication and cooperation between different teams
- The architecture usually results in increased memory consumption
- Partitioning the application into microservices is very much an art.



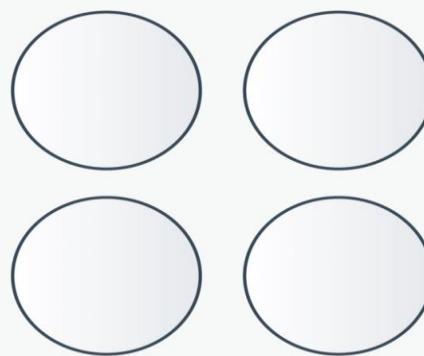
# SOA vs Microservices

## Monolithic vs. SOA vs. Microservices



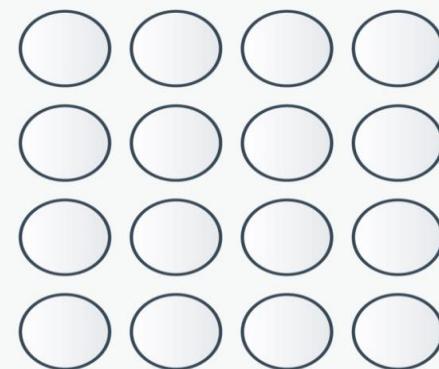
**Monolithic**

Single Unit



**SOA**

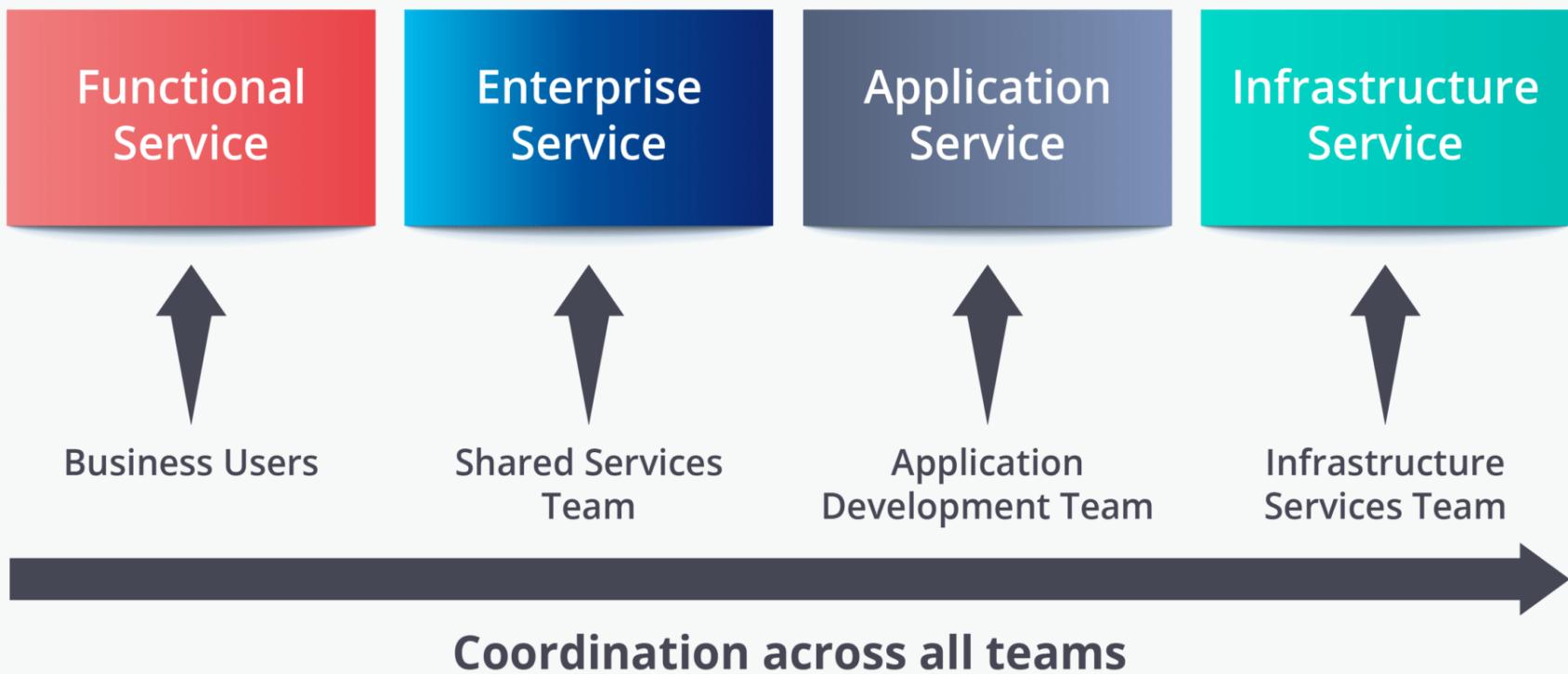
Coarse-grained



**Microservices**

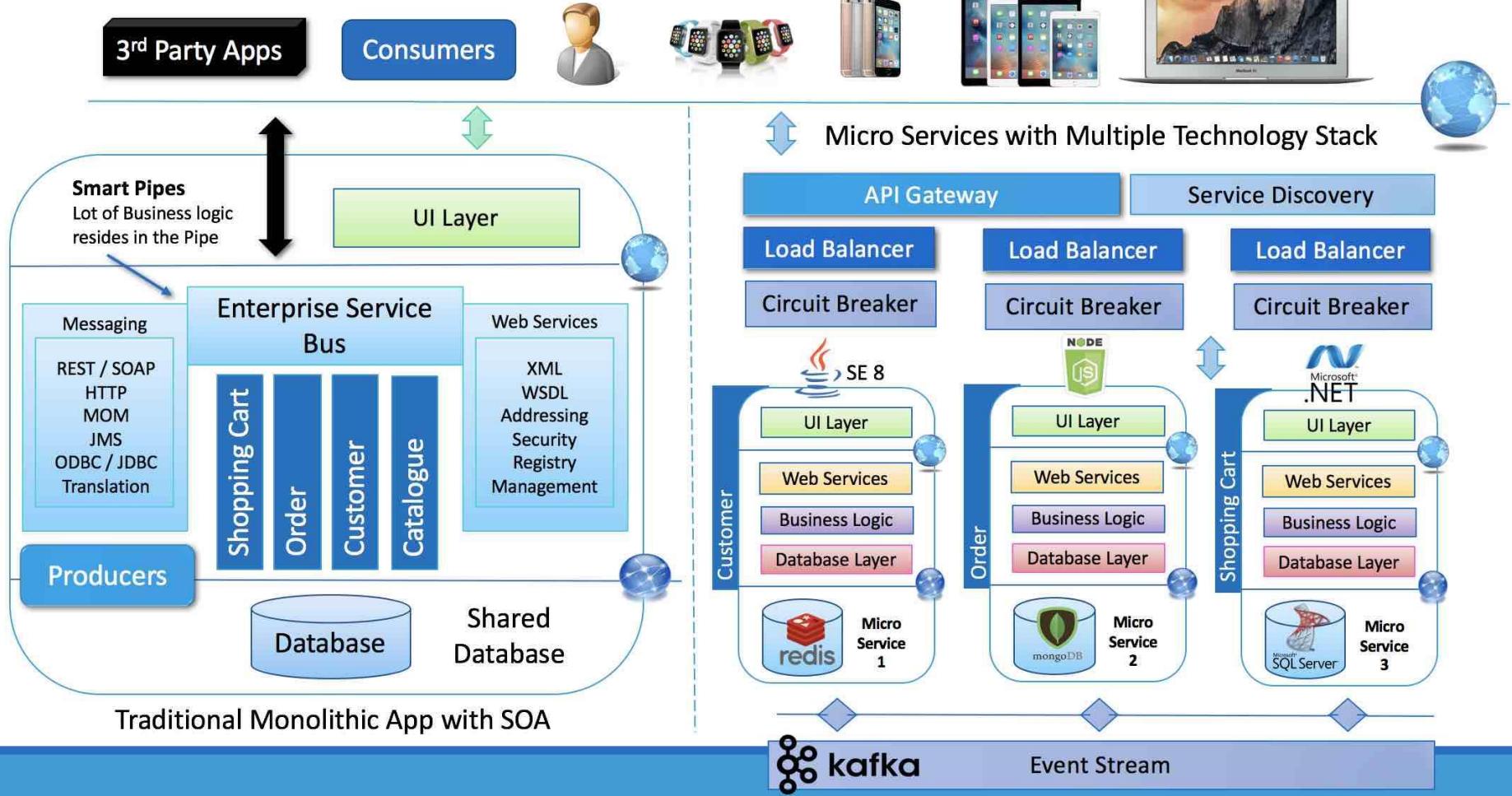
Fine-grained

# Service Oriented Architecture





## SOA vs. Micro Services Example





## SOA ARCHITECTURE



Services/APIs



Application



Application Server



Managed Runtime Environment



Operating System



Hypervisor



Storage



Network

## MICROSERVICES

Services/APIs



Managed Runtime Environment



Operating System



Hypervisor



Storage



Network



SOA	MSA
Follows “ <b>share-as-much-as-possible</b> ” architecture approach	Follows “ <b>share-as-little-as-possible</b> ” architecture approach
Importance is on <b>business functionality</b> reuse	Importance is on the concept of “ <b>bounded context</b> ”
They have <b>common governance and standards</b>	They focus on <b>people collaboration</b> and freedom of other options
Uses <b>Enterprise Service bus (ESB)</b> for communication	Simple messaging system
They support <b>multiple message protocols</b>	They use <b>lightweight protocols</b> such as <b>HTTP/REST</b> etc.
<b>Multi-threaded</b> with more overheads to handle I/O	<b>Single-threaded</b> usually with the use of Event Loop features for non-locking I/O handling
Maximizes application service reusability	Focuses on <b>decoupling</b>
<b>Traditional Relational Databases</b> are more often used	<b>Modern Relational Databases</b> are more often used
A systematic change requires modifying the monolith	A systematic change is to create a new service
DevOps / Continuous Delivery is becoming popular, but not yet mainstream	Strong focus on DevOps / Continuous Delivery

# Challenges with Microservice Architectures



- Quick Setup needed : You cannot spend a month setting up each microservice. You should be able to create microservices quickly.
- Automation : Because there are a number of smaller components instead of a monolith, you need to automate everything - Builds, Deployment, Monitoring etc.
- Visibility : You now have a number of smaller components to deploy and maintain. Maybe 100 or maybe 1000 components. You should be able to monitor and identify problems automatically. You need great visibility around all the components.



# Challenges with Microservice Architectures

---

- Bounded Context : Deciding the boundaries of a microservice is not an easy task. Bounded Contexts from Domain Driven Design is a good starting point. Your understanding of the domain evolves over a period of time. You need to ensure that the microservice boundaries evolve.
- Configuration Management : You need to maintain configurations for hundreds of components across environments. You would need a Configuration Management solution
- Dynamic Scale Up and Scale Down : The advantages of microservices will only be realized if your applications can scaled up and down easily in the cloud.
- Pack of Cards : If a microservice at the bottom of the call chain fails, it can have knock on effects on all other microservices. Microservices should be fault tolerant by Design.

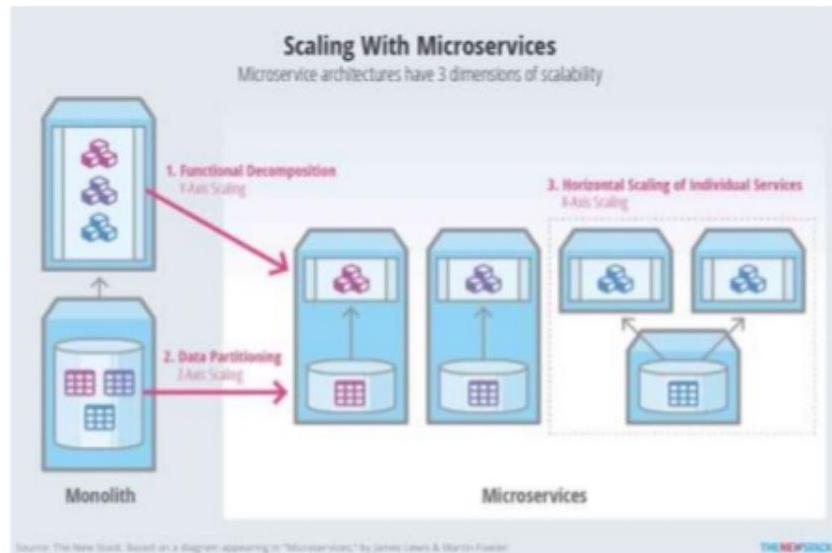
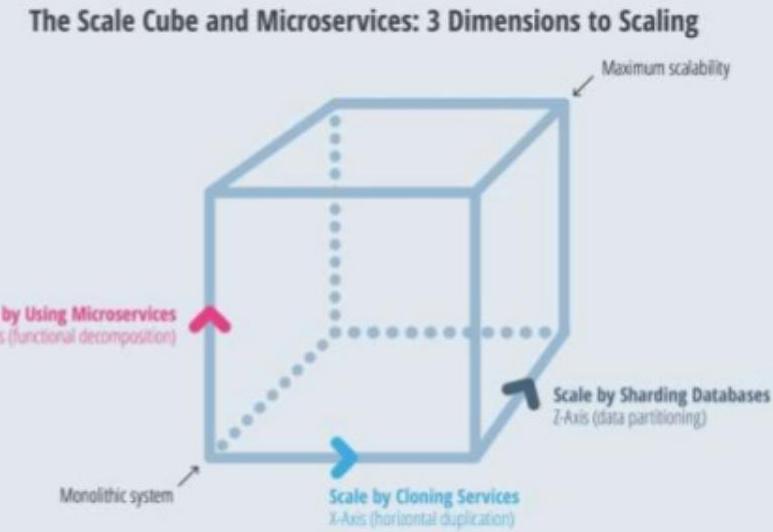


# Challenges with Microservice Architectures

---

- Debugging : When there is a problem that needs investigation, you might need to look into multiple services across different components. Centralized Logging and Dashboards are essential to make it easy to debug problems.
- Consistency : You cannot have a wide range of tools solving the same problem. While it is important to foster innovation, it is also important to have some decentralized governance around the languages, platforms, technology and tools used for implementing/deploying/monitoring microservices.

# Scale Cube and Micro Services



1. Y Axis Scaling – Functional Decomposition : Business Function as a Service
2. Z Axis Scaling – Database Partitioning : Avoid locks by Database Sharding
3. X Axis Scaling – Cloning of Individual Services for Specific Service Scalability

# Scaling Challenge

## X-AXIS SCALING

Network name: Horizontal scaling, scale out



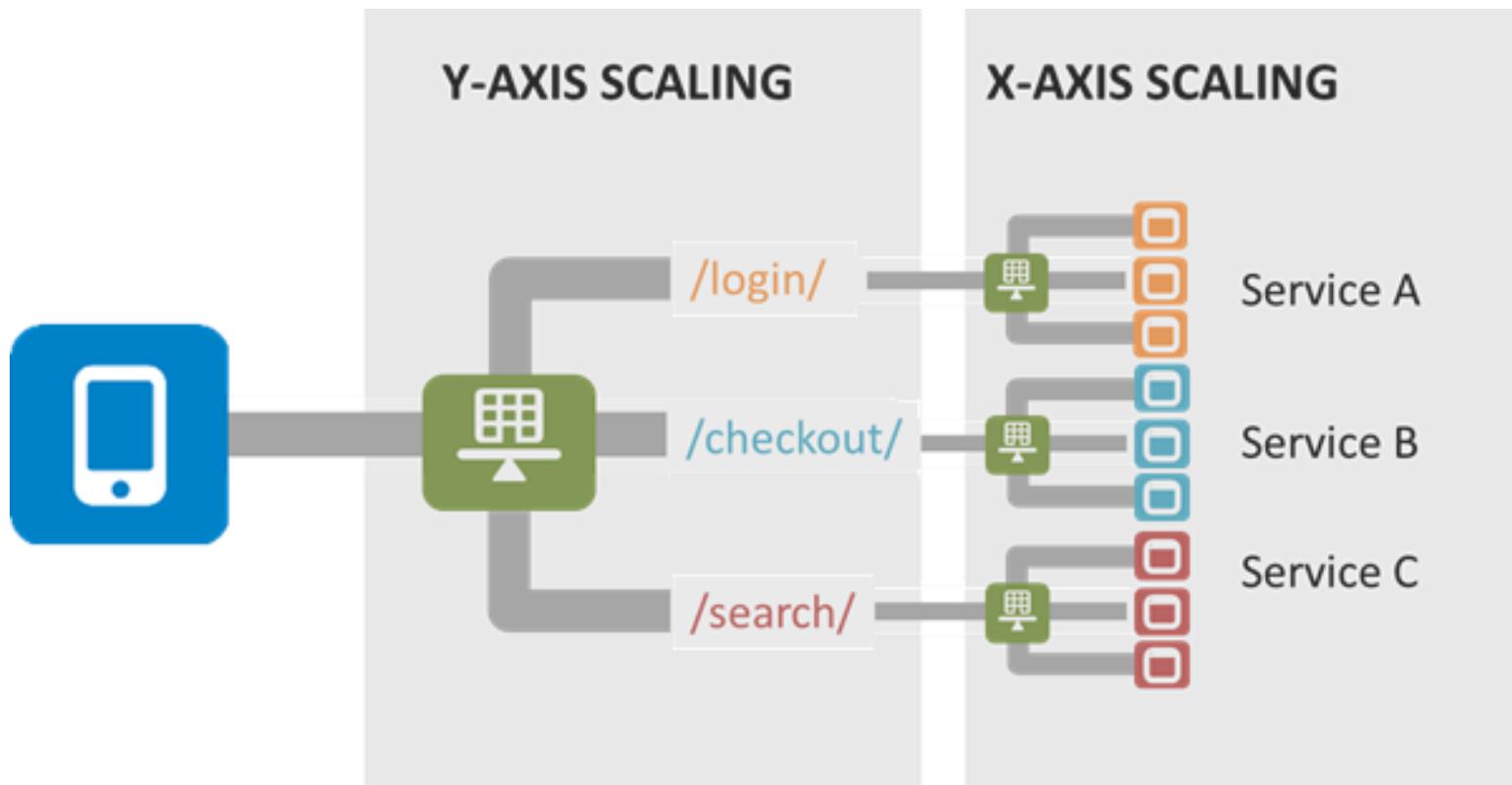
## Y-AXIS SCALING

Network name: Layer 7 Load Balancing, Content switching, HTTP Message Steering





# Scaling Challenge



# Scaling Challenge

## Z-AXIS SCALING

Network name: Layer 7 Load Balancing, Content switching, HTTP Message Steering





# Proposed Solutions to Handle Challenges

- **1. Data Synchronization** — We have event sourcing architecture to address this issue using the async messaging platform. The saga design pattern can address this challenge.
- **2. Security** — An API Gateway can solve these challenges. Kong is very popular and is open-source, and is being used by many companies in production. Custom solutions can also be developed for API security using JWT token, Spring Security, and Netflix Zuul/ Zuul2. There are enterprise solutions available, too, like Apigee and Okta (2-step authentication). Openshift is used for public cloud security for its top features, like Red Hat Linux Kernel-based security and namespace-based app-to-app security.
- **3. Versioning** — This will be taken care of by API registry and discovery APIs using the dynamic Swagger API, which can be updated dynamically and shared with consumers on the server.
- **4. Discovery** — This will be addressed by API discovery tools like Kubernetes and OpenShift. It can also be done using Netflix Eureka at the code level. However, doing it in with the orchestration layer will be better and can be managed by these tools rather doing and maintaining it through code and configuration.



# Proposed Solutions to Handle Challenges

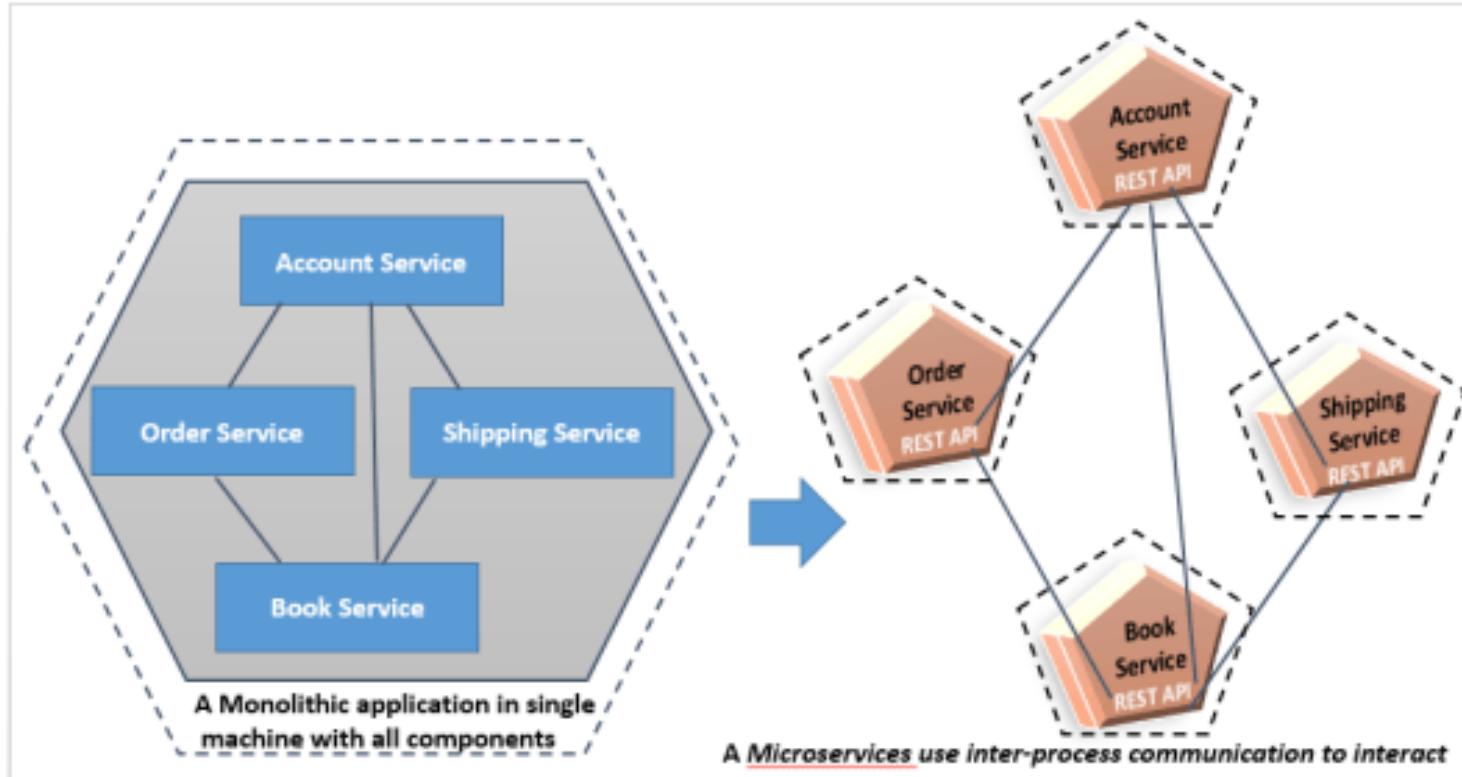
- **5. Data Staleness** — The database should be always updated to give recent data. The API will fetch data from the recent and updated database. A timestamp entry can also be added with each record in the database to check and verify the recent data. Caching can be used and customized with an acceptable eviction policy based on business requirements.
- **6. Debugging and Logging** — There are multiple solutions for this. Externalized logging can be used by pushing log messages to an async messaging platform like Kafka, Google PubSub, etc. A correlation ID can be provided by the client in the header to REST APIs to track the relevant logs across all the pods/Docker containers. Also, local debugging can be done individually on each microservice using the IDE or checking the logs.
- **7. Testing** — This issue can be addressed with unit testing by mocking REST APIs or integrated/dependent APIs which are not available for testing using WireMock, BDD, Cucumber, integration testing, performance testing using JMeter, and any good profiling tool like Jprofiler, DynaTrace, YourToolKit, VisualVM, etc.



## Proposed Solutions to Handle Challenges

- **8. Monitoring** — Monitoring can be done using open-source tools like Prometheus in combination with Grafana by creating gauge and matrices, Kubernetes/OpenShift, Influx DB, Apigee, combined with Grafana, and Graphite.
- **9. DevOps Support** — Microservices deployment and support-related challenges can be addressed using state-of-the-art DevOps tools like GCP, Kubernetes, and OpenShift with Jenkins.
- **10. Fault Tolerance** — Netflix Hystrix can be used to break the circuit if there is no response from the API for the given SLA/ETA.

# Microservices Inter-Service Communication



# Microservices Inter-Service Communication



- In Microservice Architecture, we can classify our inter-service communication into two approaches like the following:
- Synchronous communication style
- Asynchronous communication style

# Synchronous communication style



- The client sends a request to the server and waits for a response from the service (Mostly JSON over HTTP).
- For example, Spring Cloud Netflix provides the most common pattern for synchronous REST communication such as Feign or Hystrix.

# Synchronous communication style

---



- The synchronous communication approach does have some drawbacks, such as timeouts and strong coupling.
- There are numerous protocols, such as REST, gRPC, and Apache Thrift, that can be used to interact with services synchronously.

# Synchronous one-to-one communication style

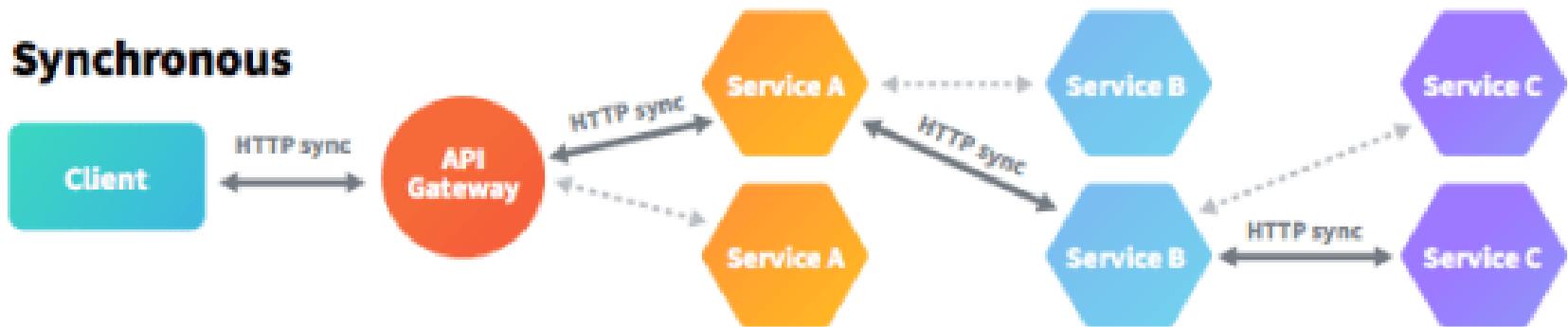


- Most synchronous communications are one-to-one.
- In synchronous one-to-one communication, you can also use multiple instances of a service to scale the service.
- However, if you do, you have to use a load-balancing mechanism on the client side.
- Each service contains meta-information about all instances of the calling service.
- This information is provided by the service discovery server, an example of which is Netflix Eureka.

# Synchronous one-to-one communication style



- There are several load-balancing mechanisms you can use.
- One of these is Netflix Ribbon, which carries out load-balancing on the client side, as illustrated in the following diagram:



# Synchronous one-to-one communication style



- As you can see in the preceding diagram, we have multiple instances of a particular service, but the services are still communicating one-to-one.
- That means that each service communicates to an instance of another service.
- The load balancer chooses which method should be called.
- The following is a list of some of the most common load-balancing methods available:
  - Round-Robin: This is the simplest method that routes requests across all the instances sequentially.
  - Least Connections: This is a method in which the request goes to the instance that has the fewest number of connections at the time.
  - Weighted Round-Robin: This is an algorithm that assigns a weight to each instance and forwards the connection according to this weight.
  - IP Hash: This is a method that generates a unique hash key from the source IP address and determines which instance receives the request.

# Asynchronous communication style

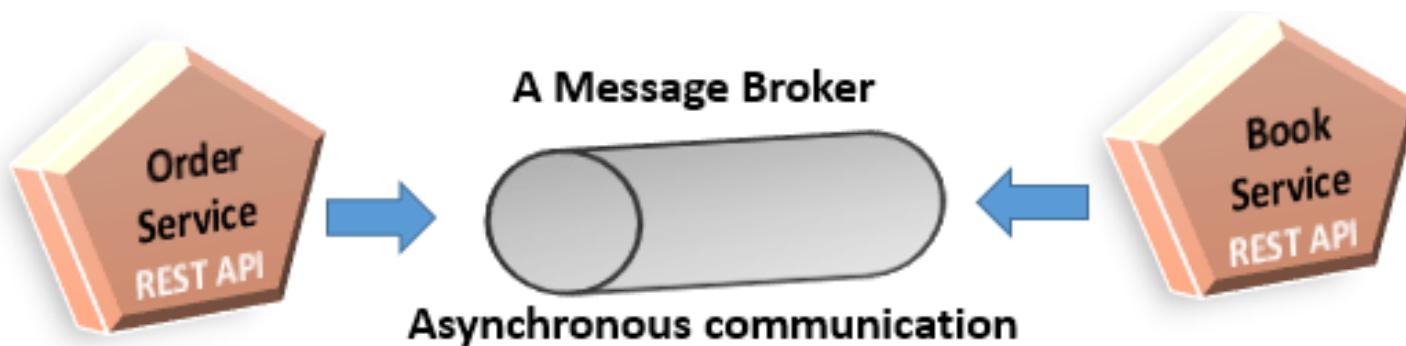


- In this communication style, the client service doesn't wait for the response coming from another service.
- So, the client doesn't block a thread while it is waiting for a response from the server. Such type of communications is possible by using lightweight messaging brokers.
- The message producer service doesn't wait for a response.
- It just generates a message and sends message to the broker.
- It waits for the only acknowledgement from the message broker to know the message has been received by a message broker or not.

# Asynchronous communication style



- The Order Service generates a message to A Message Broker and then forgets about it.
- The Book Service that subscribes to a topic is fed with all the messages belonging to that topic.
- The services don't need to know each other at all, they just need to know that messages of a certain type exist with a certain payload.



# Asynchronous communication style



- There are various tools to support lightweight messaging, you just choose one of the following message brokers that is delivering your messages to consumers running on respective microservices:
  - RabbitMQ
  - Apache Kafka
  - Apache ActiveMQ
  - NSQ
- The above tools are based on the AMQP (Advanced Message Queuing Protocol).
- This protocol provides messaging based inter-service communication.
- Spring Cloud Stream also provides mechanisms for building message-driven microservices using either the RabbitMQ or the Apache Kafka.

# Asynchronous communication style

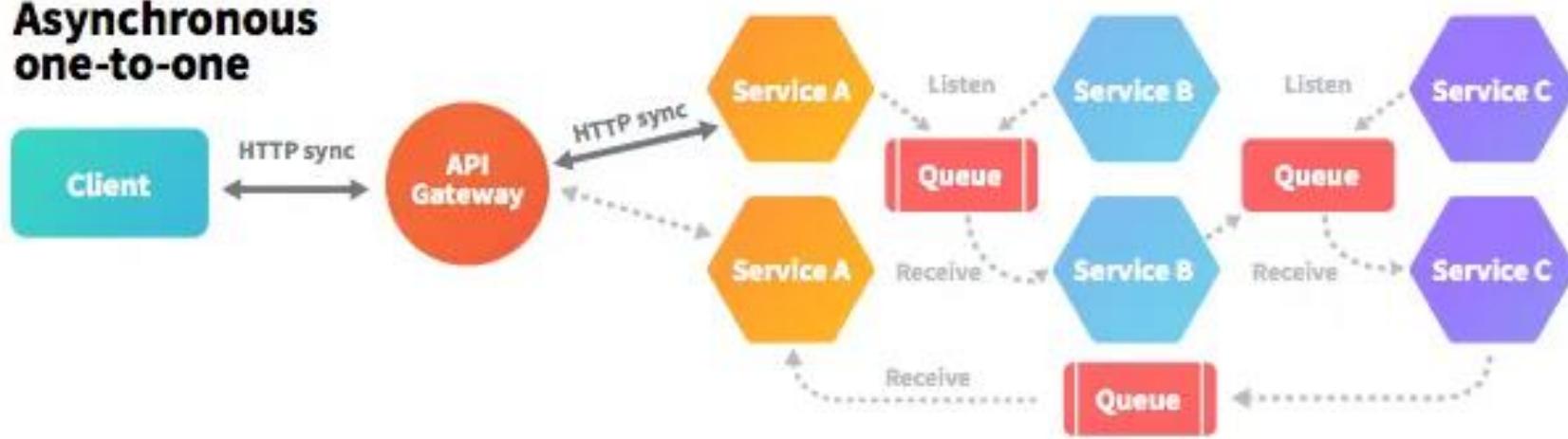


- Asynchronous one-to-one communication style
- Asynchronous one-to-many communication style

# Asynchronous one-to-one communication style



## Asynchronous one-to-one



# Asynchronous one-to-many communication style

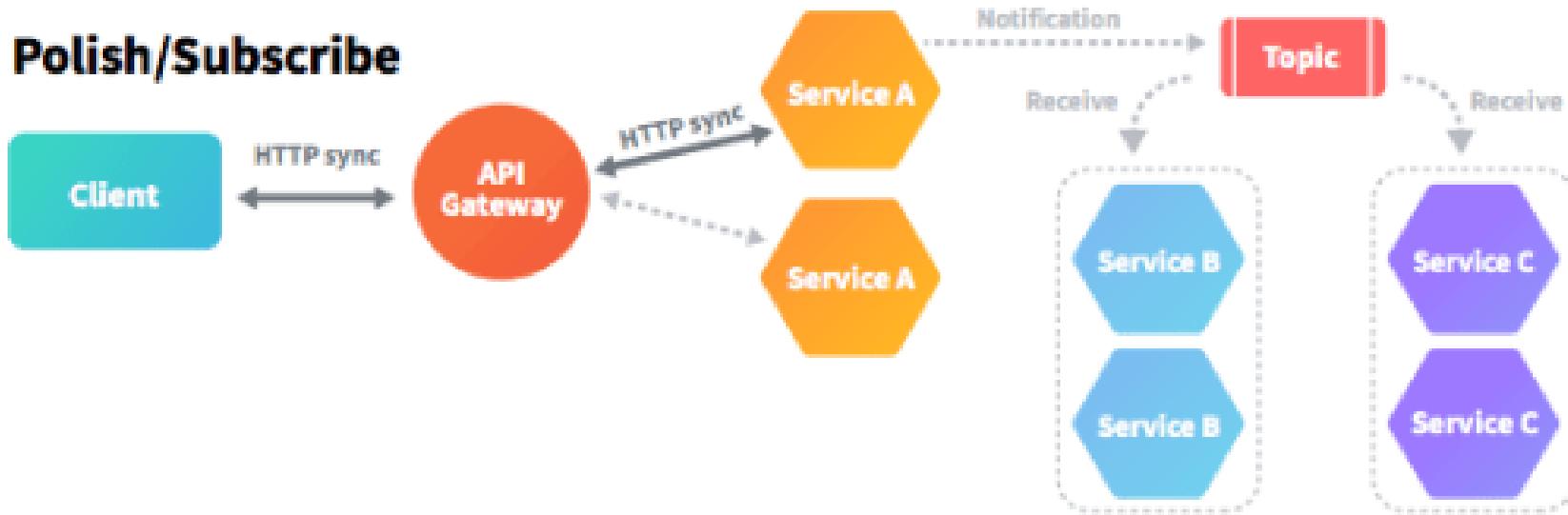


- Publish/subscribe: In this approach, a Client publishes a notification message to the message broker and this notification message is consumed by zero or more interesting services.
- Publish/async responses: In this approach, a Client publishes a request message and then waits for a certain amount of time for responses from interested services

# Asynchronous one-to-many communication style



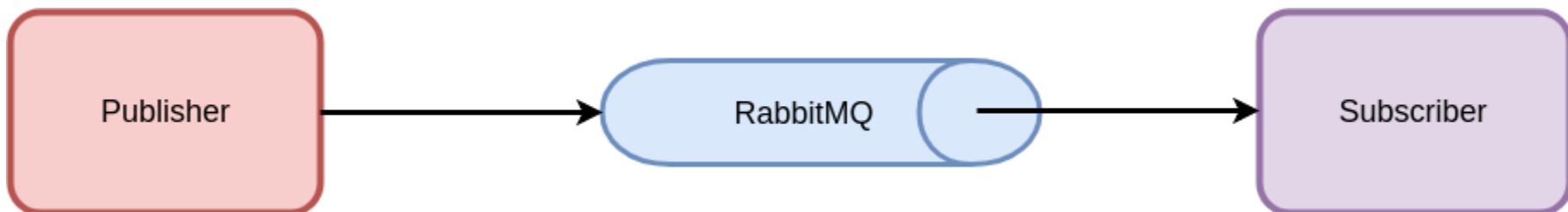
## Push/Subscribe



# Connecting Microservices Through Messaging



- Spring Cloud Stream
  - It is a framework for building highly scalable event-driven microservices connected with shared messaging systems.
  - The framework provides a flexible programming model built on already established and familiar Spring idioms and best practices, including support for persistent pub/sub semantics, consumer groups, and stateful partitions.





# Binder Implementations

---

- Spring Cloud Stream supports a variety of binder implementations
  - RabbitMQ
  - Apache Kafka
  - Kafka Streams
  - Amazon Kinesis
  - Google PubSub (*partner maintained*)
  - Solace PubSub+ (*partner maintained*)
  - Azure Event Hubs (*partner maintained*)
  - Apache RocketMQ (*partner maintained*)

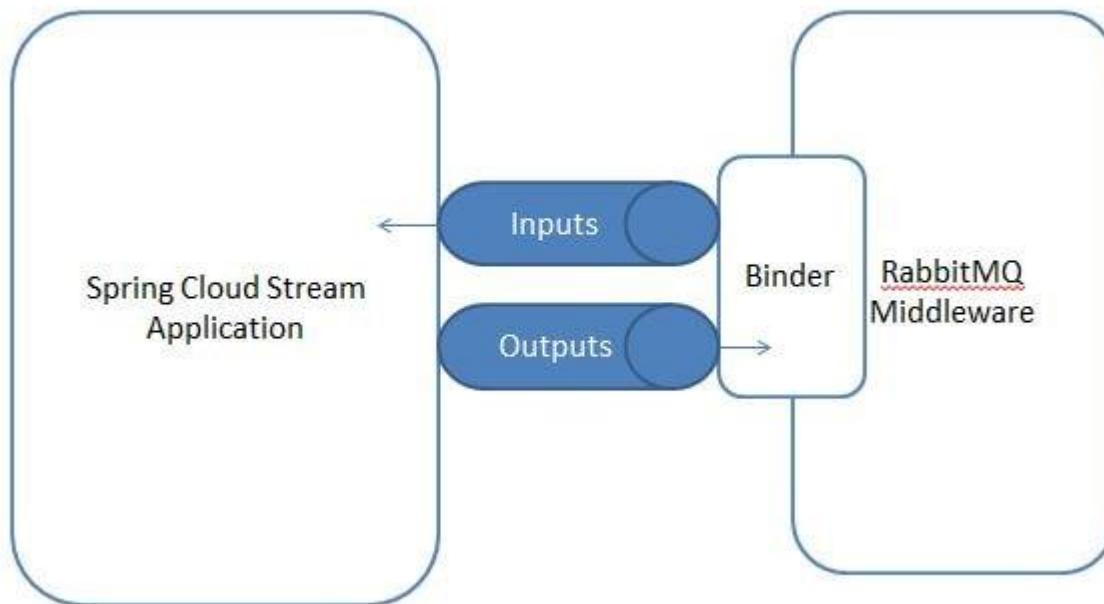


# Core Components of Cloud Stream

- The core building blocks of Spring Cloud Stream are:
- Destination Binders: Components responsible to provide integration with the external messaging systems.
- Destination Bindings: Bridge between the external messaging systems and application provided Producers and Consumers of messages (created by the Destination Binders).
- Message: The canonical data structure used by producers and consumers to communicate with Destination Binders (and thus other applications via external messaging systems).

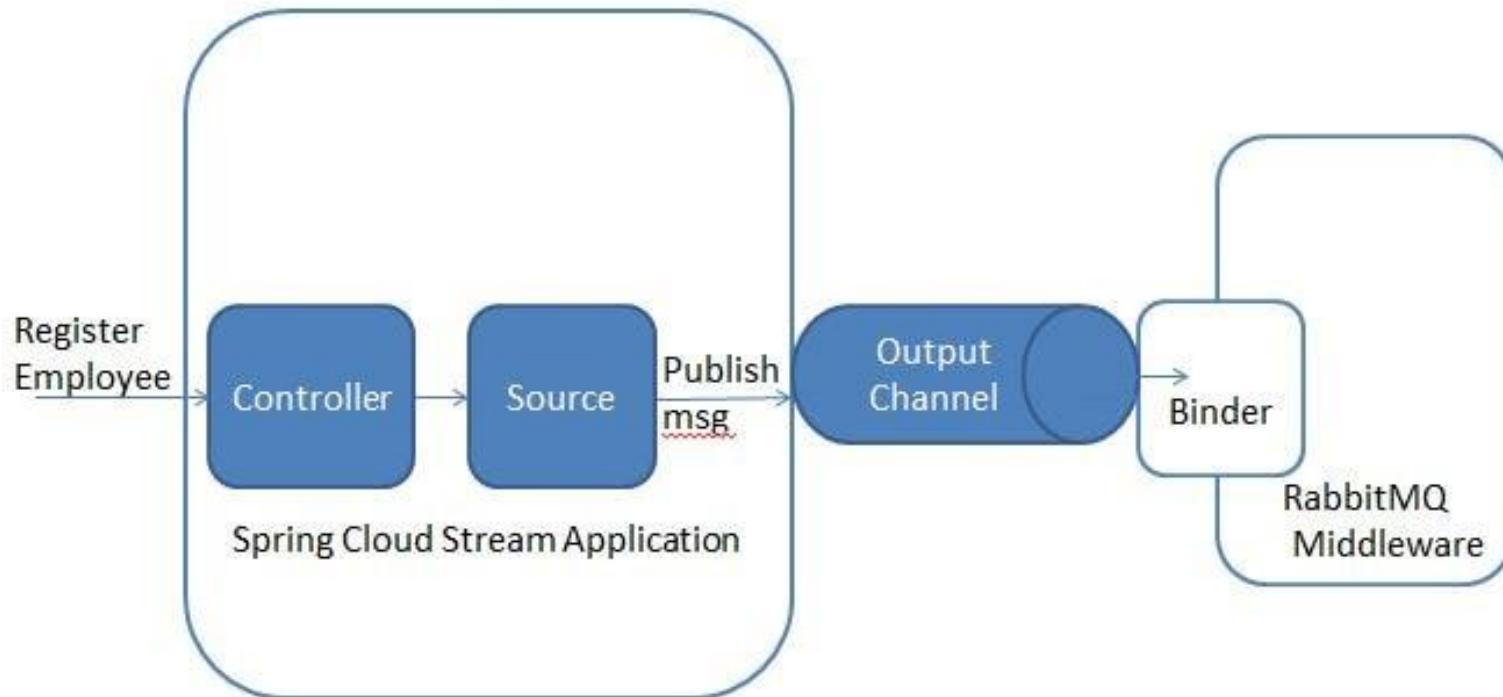


# Core Components of Cloud Stream





# Core Components of Cloud Stream



# Asynchronous communication in Microservices

---



- Kafka
- RabbitMQ
- Google Pub/Sub
- Amazon Services
- ActiveMQ
- Azure Services



# What is ZooKeeper

- ZooKeeper is a centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services.
- All of these kinds of services are used in some form or another by distributed applications.
- Each time they are implemented there is a lot of work that goes into fixing the bugs and race conditions that are inevitable.
- Because of the difficulty of implementing these kinds of services, applications initially usually skimp on them, which make them brittle in the presence of change and difficult to manage.
- Even when done correctly, different implementations of these services lead to management complexity when the applications are deployed.

# What is ZooKeeper?



- Apache ZooKeeper plays the very important role in system architecture as it works in the shadow of more exposed Big Data tools, as Apache Spark or Apache Kafka.
- Originally, the ZooKeeper framework was built at “Yahoo!”. Because it helps to access their applications in an easy manner.



## Why Apache ZooKeeper ?



1  
Naming Service

2  
Configuration Management

3  
Synchronization Service

4  
Group Services





# Why Zookeeper

---

- It allows for mutual exclusion and cooperation between server processes.
- It ensures that your application runs consistently.
- The transaction process is never completed partially. It is either given the status of Success or failure. The distributed state can be held up, but it's never wrong.
- Irrespective of the server that it connects to, a client will be able to see the same view of the service
- Helps you to encode the data as per the specific set of rules

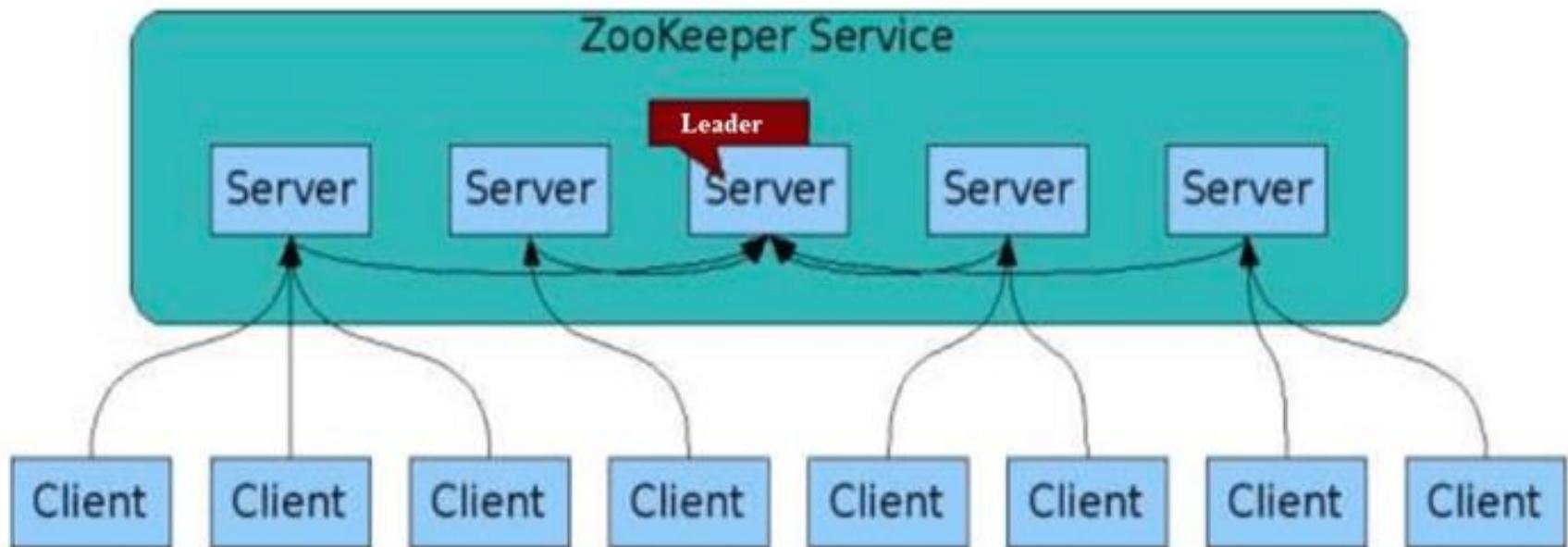


# Why Zookeeper

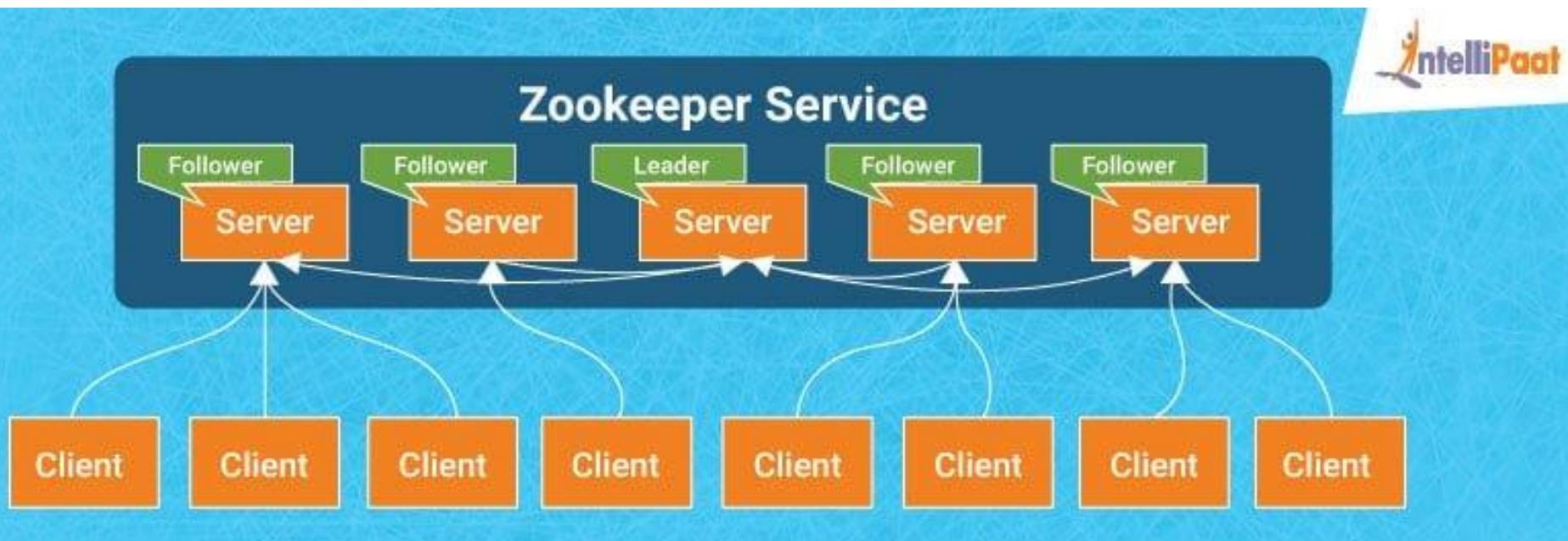
---

- It helps to maintain a standard hierarchical namespace similar to files and directories.
- Computers, which run as a single system which can be locally or geographically connected.
- It allows to Join/leave node in a cluster and node status at the real time
- You can increase performance by deploying more machines.
- It allows you to elect a node as a leader for better coordination.
- ZooKeeper works fast with workloads where reads to the data are more common than writes.

# ZooKeeper Architecture: How it works?



# ZooKeeper Architecture: How it works?





# ZooKeeper Architecture

- Server: The server sends an acknowledgement when any client connects. In the case when there is no response from the connected server, the client automatically redirects the message to another server.
- Client: Client is one of the nodes in the distributed application cluster. It helps you to access information from the server. Every client sends a message to the server at regular intervals that helps the server to know that the client is alive.



# ZooKeeper Architecture

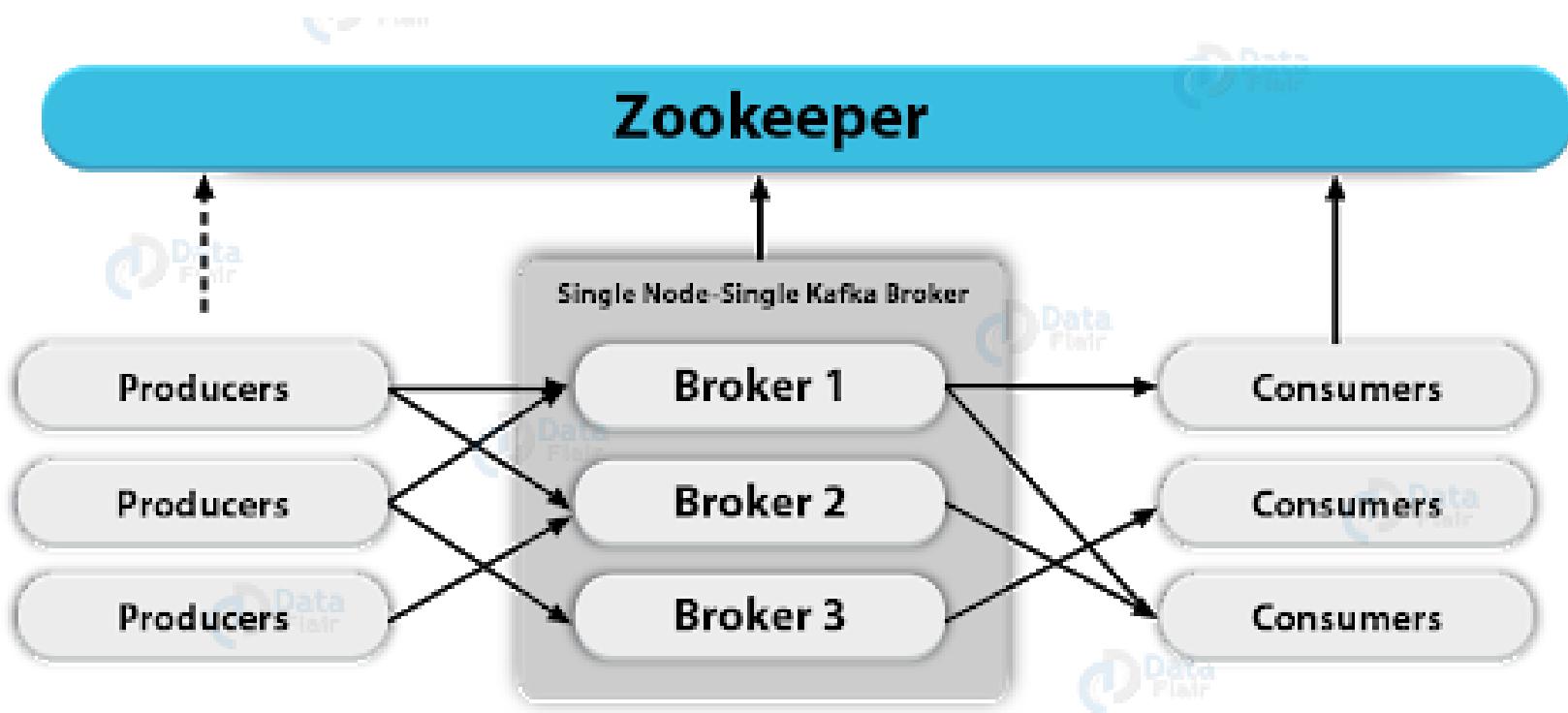
- Leader: One of the servers is designated a Leader. It gives all the information to the clients as well as an acknowledgment that the server is alive. It would perform automatic recovery if any of the connected nodes failed.
- Follower: Server node which follows leader instruction is called a follower.
- Client read requests are handled by the correspondingly connected Zookeeper server
- The client writes requests are handled by the Zookeeper leader.



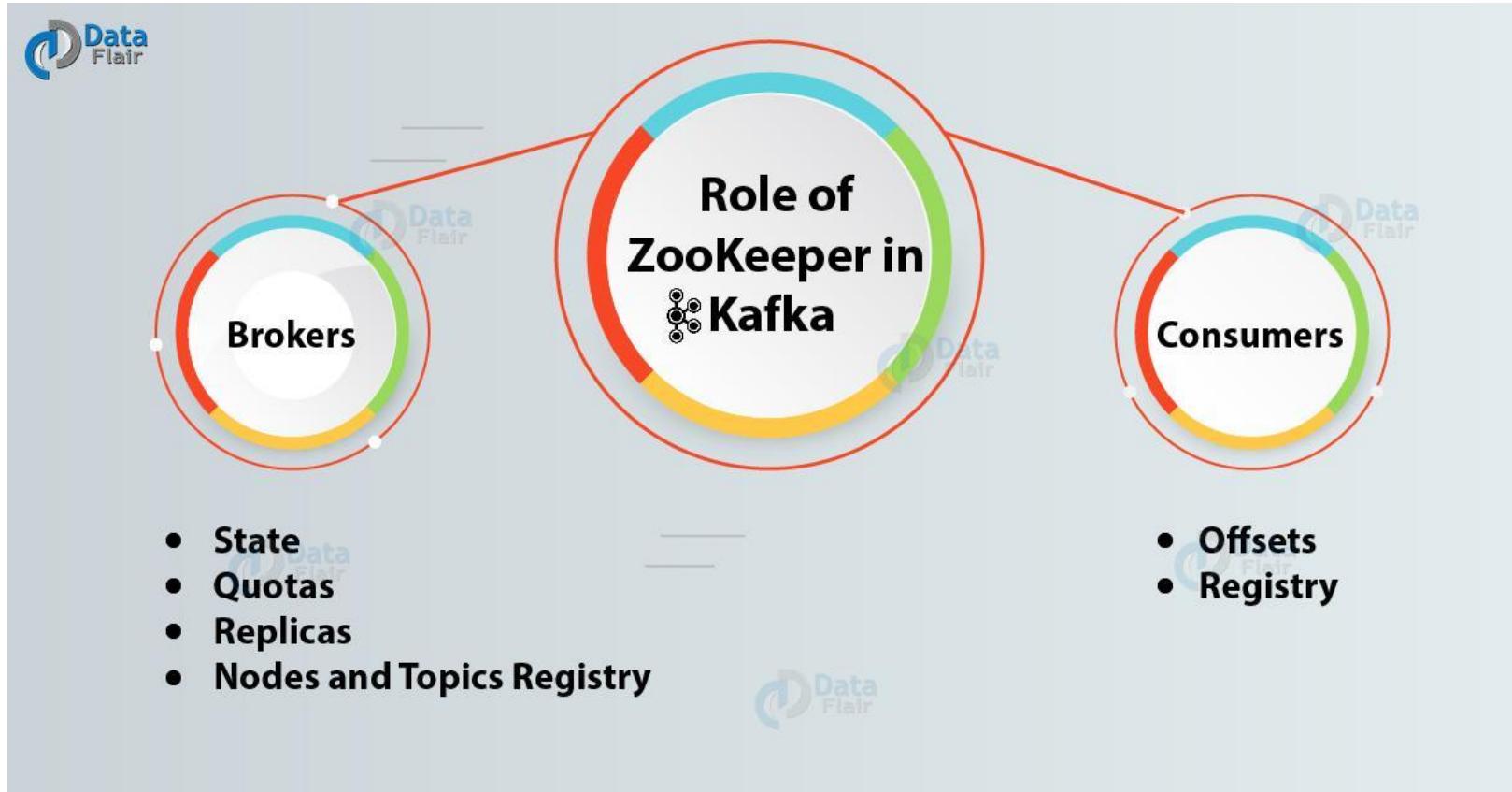
# ZooKeeper Architecture

- Ensemble/Cluster: Group of Zookeeper servers which is called ensemble or a Cluster. You can use ZooKeeper infrastructure in the cluster mode to have the system at the optimal value when you are running the Apache.
- ZooKeeper WebUI: If you want to work with ZooKeeper resource management, then you need to use WebUI. It allows working with ZooKeeper using the web user interface, instead of using the command line. It offers fast and effective communication with the ZooKeeper application.

# What is ZooKeeper?



# ZooKeeper in Kafka



# Companies using Zookeeper



- Yahoo
- Facebook
- eBay
- Twitter
- Netflix
- Zynga
- Nutanix



# What is Kafka

---

- Kafka consists of Records, Topics, Consumers, Producers, Brokers, Logs, Partitions, and Clusters. Records can have key (optional), value and timestamp.
- Kafka Records are immutable.
- A Kafka Topic is a stream of records ("/orders", "/user-signups").
- You can think of a Topic as a feed name.
- A topic has a Log which is the topic's storage on disk.
- A Topic Log is broken up into partitions and segments.



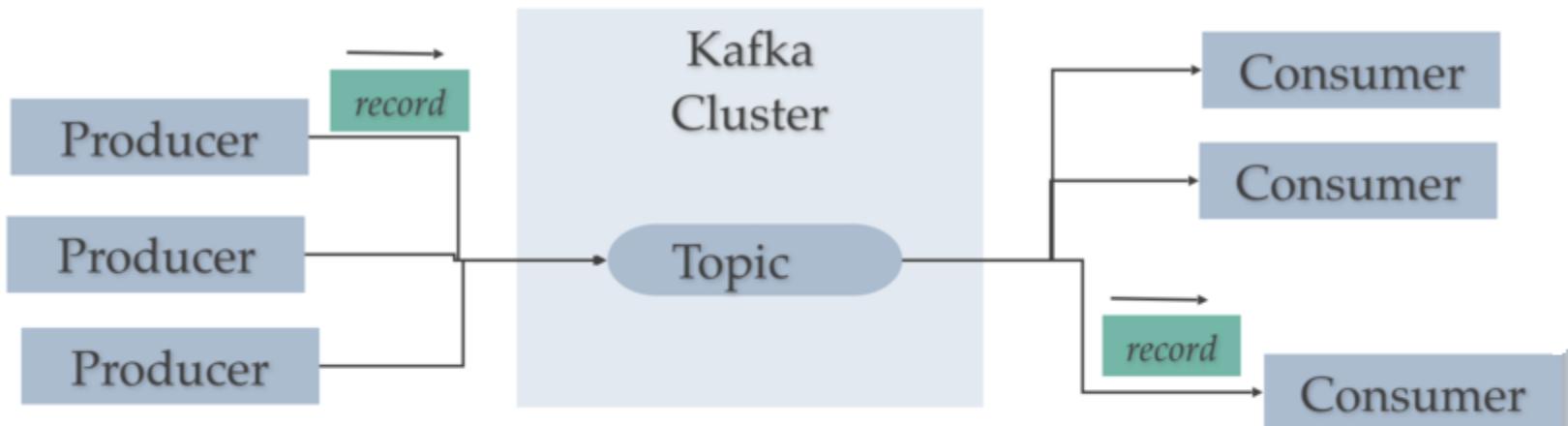
# What is Kafka

---

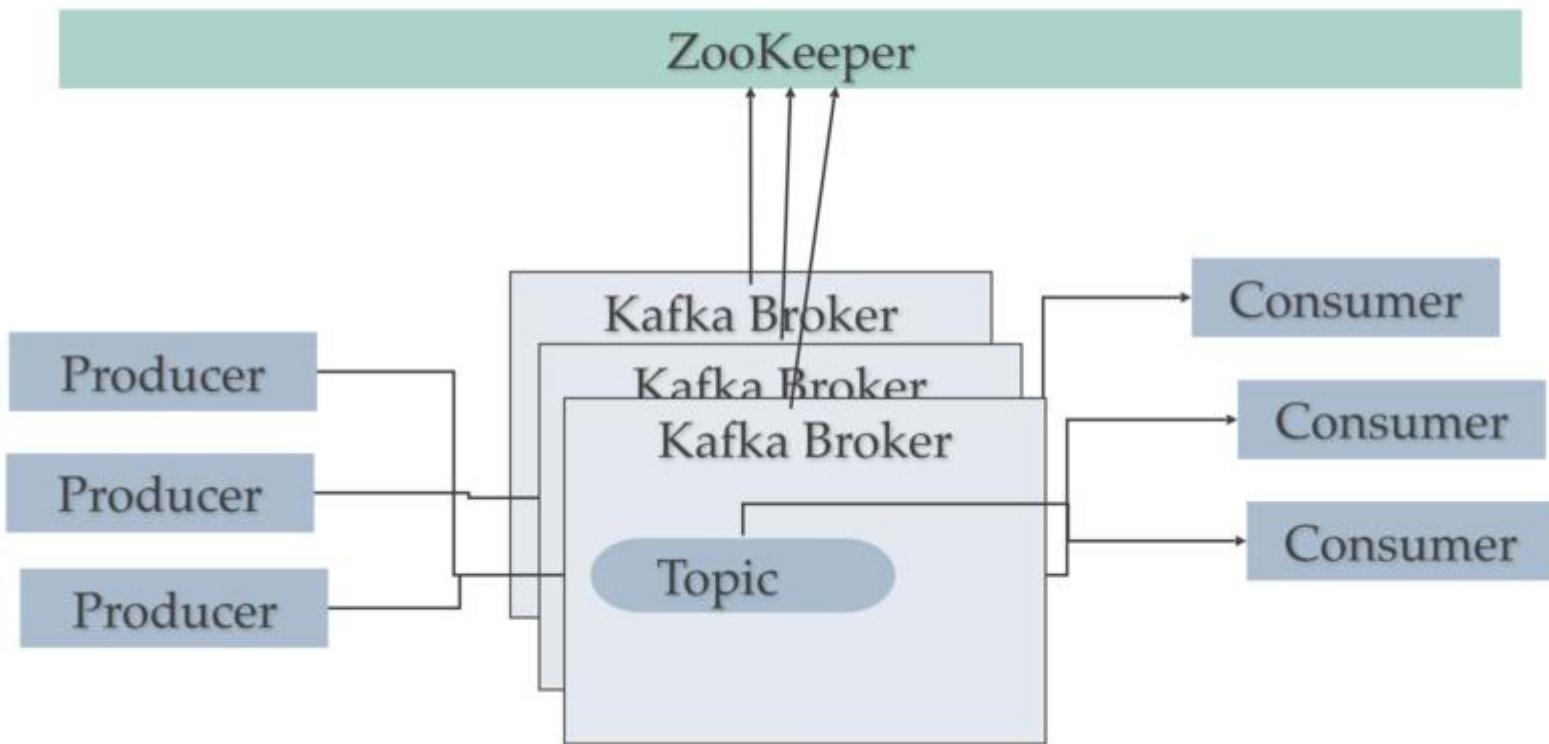
- The Kafka Producer API is used to produce streams of data records.
- The Kafka Consumer API is used to consume a stream of records from Kafka.
- A Broker is a Kafka server that runs in a Kafka Cluster.
- Kafka Brokers form a cluster.
- The Kafka Cluster consists of many Kafka Brokers on many servers.
- Broker sometimes refer to more of a logical system or as Kafka as a whole.



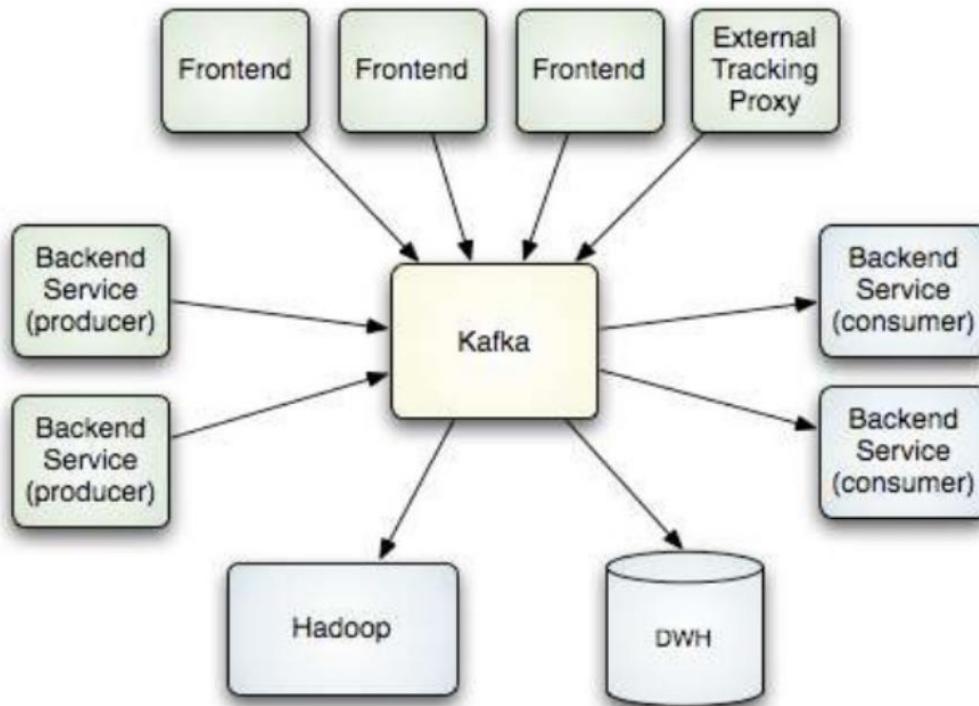
# Kafka: Topics, Producers, and Consumers



## ZooKeeper does coordination for Kafka Cluster

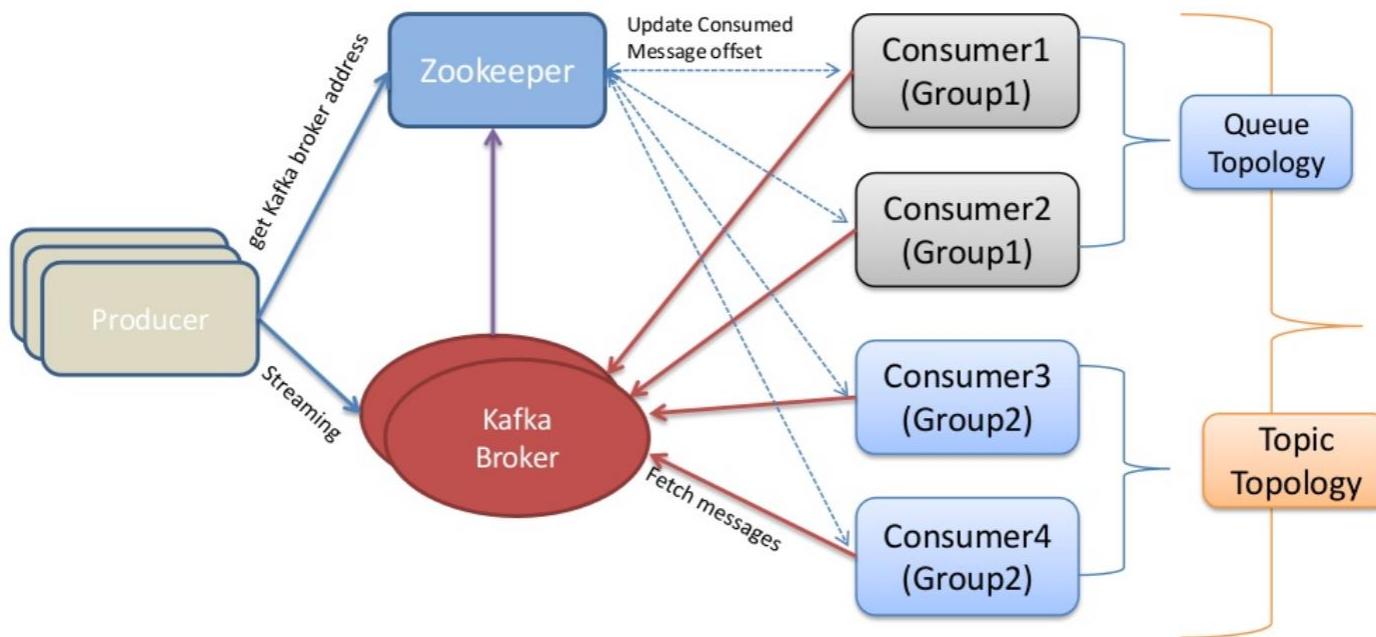


# How it works



Credit : <http://kafka.apache.org/design.html>

# Real time transfer



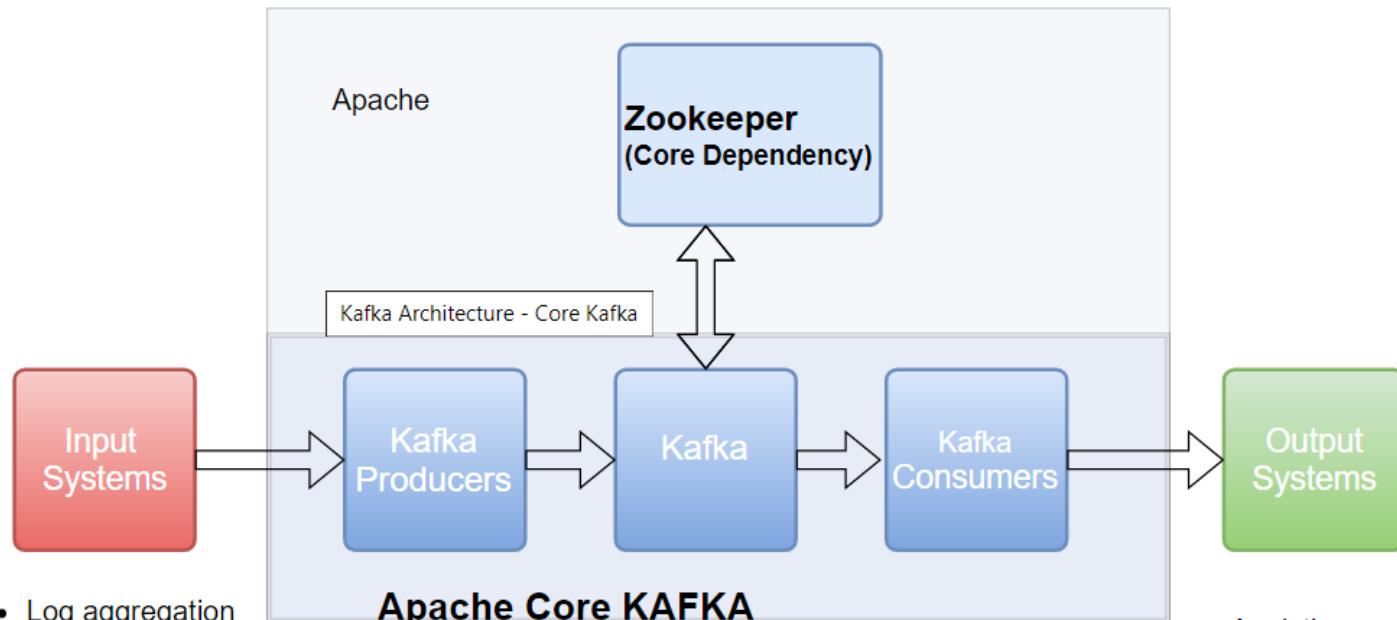
# ZooKeeper in Kafka



- Kafka needs ZooKeeper
- Kafka uses Zookeeper to do leadership election of Kafka Broker and Topic Partition pairs.
- Kafka uses Zookeeper to manage service discovery for Kafka Brokers that form the cluster.
- Zookeeper sends changes of the topology to Kafka, so each node in the cluster knows when a new broker joined, a Broker died, a topic was removed or a topic was added, etc.
- Zookeeper provides an in-sync view of Kafka Cluster configuration.



## Kafka Architecture: Core Kafka



- Log aggregation
- Metrics
- KPIs
- Batch imports
- Audit trail
- User activity logs
- Web logs

*Not part of core*

- Schema Registry
- Avro
- Kafka REST Proxy
- Kafka Connect
- Kafka Streams

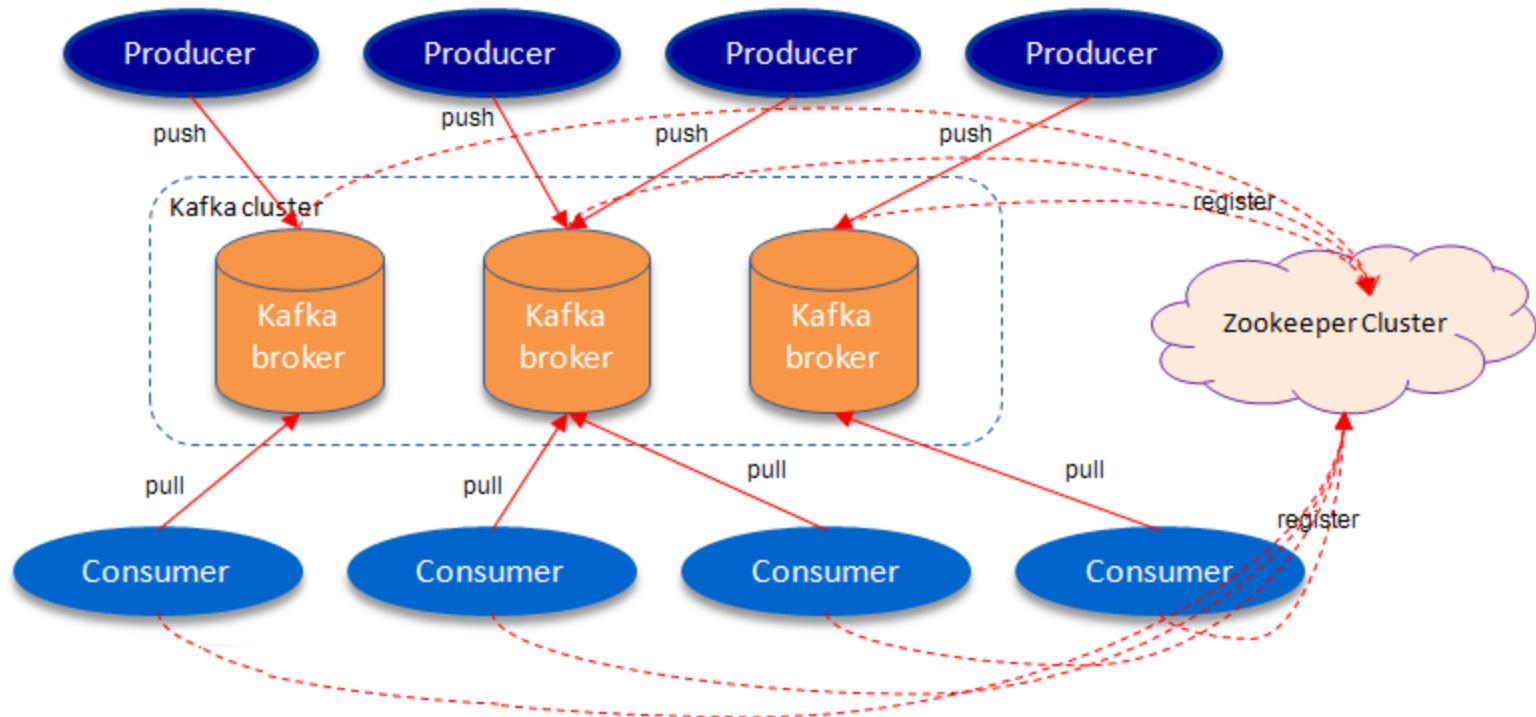
*Apache Kafka Core*

- Server/Broker
- Scripts to start libs
- Script to start up Zookeeper
- Utils to create topics
- Utils to monitor stats

- Analytics
- Databases
- Machine Learning
- Dashboards
- Indexed for Search
- Business Intelligence

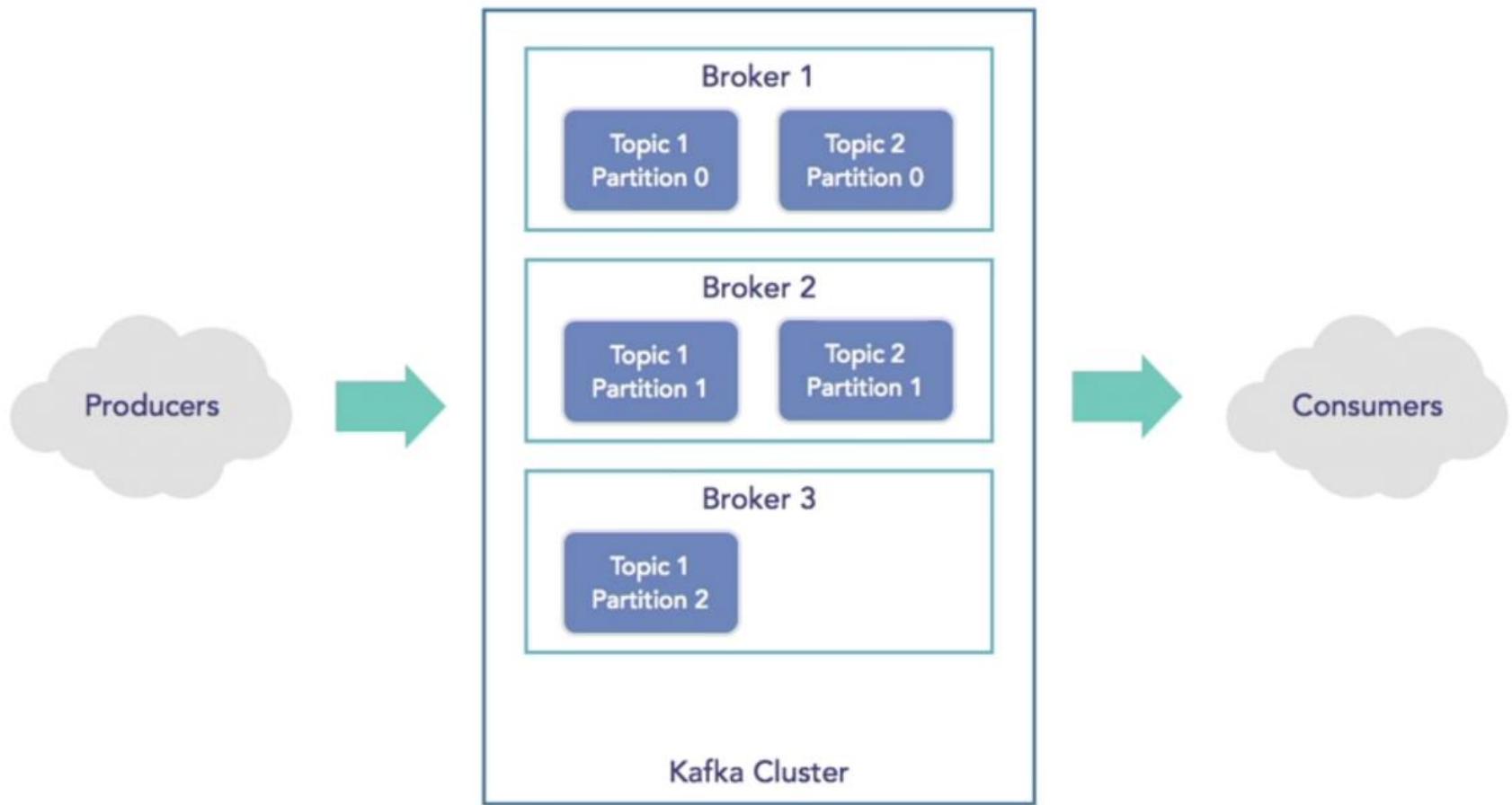


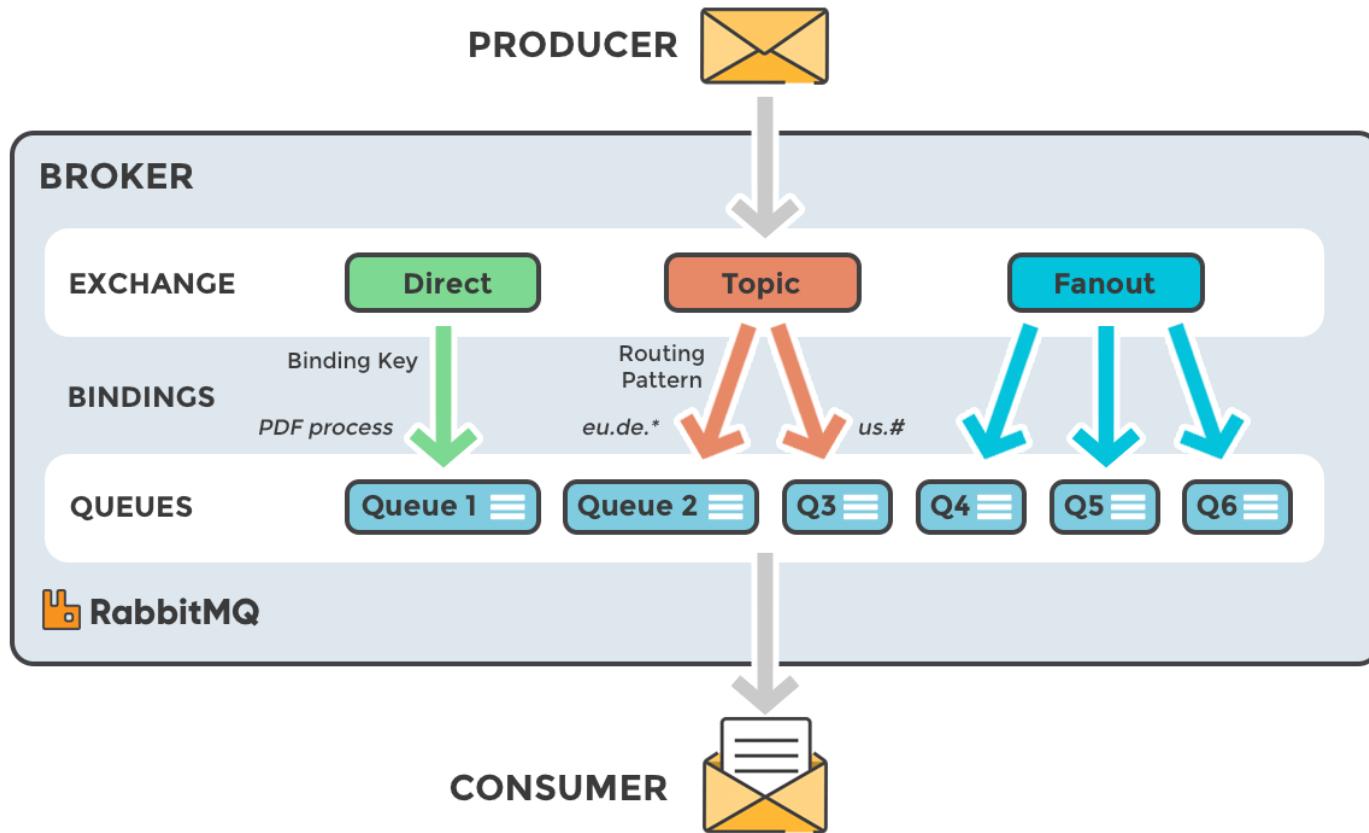
# Kafka architecture





# Kafka architecture







## Zkserver from command prompt

```
2019-11-03 11:48:12,127 [myid:] - WARN  [NIOWorkerThread-1:NIOServerCnxn@370] - Exception^
  causing close of session 0x0: null
2019-11-03 11:48:12,992 [myid:] - WARN  [NIOWorkerThread-2:NIOServerCnxn@370] - Exception^
  causing close of session 0x0: null
2019-11-03 11:48:13,906 [myid:] - WARN  [NIOWorkerThread-4:NIOServerCnxn@370] - Exception^
  causing close of session 0x0: null
2019-11-03 11:48:15,023 [myid:] - WARN  [NIOWorkerThread-6:NIOServerCnxn@370] - Exception^
  causing close of session 0x0: null
2019-11-03 11:48:16,051 [myid:] - WARN  [NIOWorkerThread-9:NIOServerCnxn@370] - Exception^
  causing close of session 0x0: null
2019-11-03 11:48:17,642 [myid:] - WARN  [NIOWorkerThread-13:NIOServerCnxn@370] - Exception^
  n causing close of session 0x0: null
2019-11-03 11:48:18,605 [myid:] - WARN  [NIOWorkerThread-12:NIOServerCnxn@370] - Exception^
  n causing close of session 0x0: null
2019-11-03 11:48:19,724 [myid:] - WARN  [NIOWorkerThread-16:NIOServerCnxn@370] - Exception^
  n causing close of session 0x0: null
2019-11-03 11:48:20,792 [myid:] - WARN  [NIOWorkerThread-1:NIOServerCnxn@370] - Exception^
  causing close of session 0x0: null
2019-11-03 11:48:21,807 [myid:] - WARN  [NIOWorkerThread-5:NIOServerCnxn@370] - Exception^
  causing close of session 0x0: null
```



C:\Windows\System32\cmd.exe - ./bin/windows/kafka-server-start.bat ./config/server.properties

```
tention.ms -> 86400000, cleanup.policy -> [delete], flush.ms -> 9223372036854775807, segm  
ent.ms -> 604800000, segment.bytes -> 1073741824, retention.ms -> 604800000, message.time  
stamp.difference.max.ms -> 9223372036854775807, segment.index.bytes -> 10485760, flush.me  
ssages -> 9223372036854775807}. (kafka.log.LogManager)  
[2019-11-03 11:43:35,072] INFO [Partition test11-0 broker=0] No checkpointed highwatermar  
k is found for partition test11-0 (kafka.cluster.Partition)  
[2019-11-03 11:43:35,073] INFO Replica loaded for partition test11-0 with initial high wa  
termark 0 (kafka.cluster.Replica)  
[2019-11-03 11:43:35,074] INFO [Partition test11-0 broker=0] test11-0 starts at Leader Ep  
och 0 from offset 0. Previous Leader Epoch was: -1 (kafka.cluster.Partition)  
[2019-11-03 11:48:44,857] INFO [GroupCoordinator 0]: Preparing to rebalance group console  
-consumer-8213 in state PreparingRebalance with old generation 0 (__consumer_offsets-33)  
(reason: Adding new member consumer-1-217673c0-2a77-44ea-b5dc-5b486eefded2) (kafka.coordin  
ator.group.GroupCoordinator)  
[2019-11-03 11:48:44,868] INFO [GroupCoordinator 0]: Stabilized group console-consumer-82  
13 generation 1 (__consumer_offsets-33) (kafka.coordinator.group.GroupCoordinator)  
[2019-11-03 11:48:44,882] INFO [GroupCoordinator 0]: Assignment received from leader for  
group console-consumer-8213 for generation 1 (kafka.coordinator.group.GroupCoordinator)  
[2019-11-03 11:51:33,455] INFO [GroupMetadataManager brokerId=0] Removed 0 expired offset  
s in 3 milliseconds. (kafka.coordinator.group.GroupMetadataManager)
```





Administrator: RabbitMQ Command Prompt (sbin dir) - kafka-console-producer.bat --broker-list localhost:9092 --topic test

[2019-11-03 11:43:19,657] ERROR org.apache.kafka.common.errors.TopicExistsException: Topic 'test' already exists.  
(kafka.admin.TopicCommand\$)

E:\software\A08\file\kafka\bin\windows>kafka-topics.bat --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic test11  
Created topic test11.

E:\software\A08\file\kafka\bin\windows>kafka-console-producer.bat --broker-list localhost:9092 --topic test  
>hi  
>how are you  
>enjoy  
>





Administrator: RabbitMQ Command Prompt (sbin dir) - kafka-console-consumer.bat --bootstrap-server localhost:9092 --topic test --from-beginning  
consumption.

E:\software\A08\file\kafka\bin\windows>kafka-console-consumer.bat --bootstrap-server localhost:2181 --topic test  
Processed a total of 0 messages  
Terminate batch job (Y/N)? y

E:\software\A08\file\kafka\bin\windows>kafka-console-consumer.bat --bootstrap-server localhost:2181 --topic test  
Processed a total of 0 messages  
Terminate batch job (Y/N)? y

E:\software\A08\file\kafka\bin\windows>kafka-console-consumer.bat --bootstrap-server localhost:9092 --topic test --from-beginning  
hi  
how are you  
enjoy





# RabbitMQ

---

- The super simplified overview:
- Publishers send messages to exchanges
- Exchanges route messages to queues and other exchanges
- RabbitMQ sends acknowledgements to publishers on message receipt
- Consumers maintain persistent TCP connections with RabbitMQ and declare which queue(s) they consume
- RabbitMQ pushes messages to consumers
- Consumers send acknowledgements of success/failure
- Messages are removed from queues once consumed successfully



<b>Category</b>	<b>Kafka</b>	<b>RabbitMQ</b>	<b>Amazon SQS</b>	<b>Google Pub/Sub</b>
Message Rate (msgs/sec)	100k+	20k+	Varies (can be scaled)	10k+
Message Acknowledgements	No	Yes	Yes	Yes
Built in UI	No	Yes	Yes	Yes
Protocol	Kafka	AMQP	HTTP REST	HTTP REST
Additional features	Apache Storm, etc	Several plugins available to add more features	In-flight messages	-
Use cases	High ingestion platforms where speed, scale and efficiency is the prime concern	Robust systems where features like acknowledgements, deeper management are important	Useful when deploying applications on AWS EC2, Elastic Beanstalk to facilitate low-latency communication	Works well with applications deployed on GCE

# Solutions to Challenges with Microservice Architectures



- **Spring Cloud**
- **Spring Cloud provides solutions to cloud enable your microservices.**
- **It leverages and builds on top of some of the Cloud solutions opensourced by Netflix (Netflix OSS).**

# Spring Cloud



A screenshot of a web browser showing the Spring Cloud project page. The URL in the address bar is [projects.spring.io/spring-cloud/](https://projects.spring.io/spring-cloud/). The browser has multiple tabs open, including one titled "03 - Introduction to Spring Cloud". The main content area features the Spring logo and navigation links for DOCS, GUIDES, PROJECTS (which is highlighted in green), BLOG, and QUESTIONS. A large, semi-transparent circular graphic of two interlocking green leaves is overlaid on the page. The text on the page describes Spring Cloud's purpose in building distributed systems.

PROJECTS

## Spring Cloud

Spring Cloud provides tools for developers to quickly build some of the common patterns in distributed systems (e.g. configuration management, service discovery, circuit breakers, intelligent routing, micro-proxy, control bus, one-time tokens, global locks, leadership election, distributed sessions, cluster state). Coordination of distributed systems leads to boiler plate patterns, and using Spring Cloud developers can quickly stand up services and applications that implement those patterns. They will work well in any distributed environment, including the developer's own laptop, bare metal data centres, and managed platforms such as Cloud Foundry.

QUICK START



# Spring Cloud

- Spring Cloud provides tools for developers to quickly build some of the common patterns in distributed systems (e.g. configuration management, service discovery, circuit breakers, intelligent routing, micro-proxy, control bus, one-time tokens, global locks, leadership election, distributed sessions, cluster state).
- Coordination of distributed systems leads to boiler plate patterns, and using Spring Cloud developers can quickly stand up services and applications that implement those patterns.
- They will work well in any distributed environment, including the developer's own laptop, bare metal data centres, and managed platforms such as Cloud Foundry.



# Spring Cloud Main Projects

---

- [Spring Cloud Config](#)
  - Centralized external configuration management backed by a git repository.
  - The configuration resources map directly to Spring Environment but could be used by non-Spring applications if desired.
- [Spring Cloud Netflix](#)
  - Integration with various Netflix OSS components (Eureka, Hystrix, Zuul, Archaius, etc.).



# Spring Cloud Main Projects

---

- Spring Cloud Bus
- An event bus for linking services and service instances together with distributed messaging. Useful for propagating state changes across a cluster (e.g. config change events).
- Spring Cloud Cloudfoundry
- Integrates your application with Pivotal Cloud Foundry. Provides a service discovery implementation and also makes it easy to implement SSO and OAuth2 protected resources.



# Spring Cloud Main Projects

---

- Spring Cloud Open Service Broker
- Provides a starting point for building a service broker that implements the Open Service Broker API.
- Spring Cloud Cluster
- Leadership election and common stateful patterns with an abstraction and implementation for Zookeeper, Redis, Hazelcast, Consul.



# Spring Cloud Main Projects

---

- Spring Cloud Consul
- Service discovery and configuration management with Hashicorp Consul.
- Spring Cloud Security
- Provides support for load-balanced OAuth2 rest client and authentication header relays in a Zuul proxy.
- Spring Cloud Sleuth
- Distributed tracing for Spring Cloud applications, compatible with Zipkin, HTrace and log-based (e.g. ELK) tracing.



# Spring Cloud Main Projects

---

- Spring Cloud Data Flow
- **A cloud-native orchestration service for composable microservice applications on modern runtimes. Easy-to-use DSL, drag-and-drop GUI, and REST-APIs together simplifies the overall orchestration of microservice based data pipelines.**
- Spring Cloud Stream
- **A lightweight event-driven microservices framework to quickly build applications that can connect to external systems. Simple declarative model to send and receive messages using Apache Kafka or RabbitMQ between Spring Boot apps.**



# Spring Cloud Main Projects

---

- **Spring Cloud Stream App Starters**
- Spring Cloud Stream App Starters are Spring Boot based Spring Integration applications that provide integration with external systems.
- **Spring Cloud Task**
- A short-lived microservices framework to quickly build applications that perform finite amounts of data processing. Simple declarative for adding both functional and non-functional features to Spring Boot apps.



# Spring Cloud Main Projects

---

- **Spring Cloud Task App Starters**
- Spring Cloud Task App Starters are Spring Boot applications that may be any process including Spring Batch jobs that do not run forever, and they end/stop after a finite period of data processing.
- **Spring Cloud Zookeeper**
- Service discovery and configuration management with Apache Zookeeper.



# Spring Cloud Main Projects

---

- **Spring Cloud AWS**
- Easy integration with hosted Amazon Web Services. It offers a convenient way to interact with AWS provided services using well-known Spring idioms and APIs, such as the messaging or caching API. Developers can build their application around the hosted services without having to care about infrastructure or maintenance.
- **Spring Cloud Connectors**
- Makes it easy for PaaS applications in a variety of platforms to connect to backend services like databases and message brokers (the project formerly known as "Spring Cloud").



# Spring Cloud Main Projects

---

- **Spring Cloud Starters**
  - Spring Boot-style starter projects to ease dependency management for consumers of Spring Cloud.  
(Discontinued as a project and merged with the other projects after Angel.SR2.)
- **Spring Cloud CLI**
  - Spring Boot CLI plugin for creating Spring Cloud component applications quickly in Groovy
- **Spring Cloud Contract**
  - Spring Cloud Contract is an umbrella project holding solutions that help users in successfully implementing the Consumer Driven Contracts approach.



# Spring Cloud Main Projects

---

- **Spring Cloud Gateway**
  - Spring Cloud Gateway is an intelligent and programmable router based on Project Reactor.
- **Spring Cloud OpenFeign**
  - Spring Cloud OpenFeign provides integrations for Spring Boot apps through autoconfiguration and binding to the Spring Environment and other Spring programming model idioms.
- **Spring Cloud Pipelines**
  - Spring Cloud Pipelines provides an opinionated deployment pipeline with steps to ensure that your application can be deployed in zero downtime fashion and easily rolled back if something goes wrong.



# Spring Cloud Main Projects

---

- **Spring Cloud Function**
- Spring Cloud Function promotes the implementation of business logic via functions. It supports a uniform programming model across serverless providers, as well as the ability to run standalone (locally or in a PaaS).
- .



# Spring Cloud Config Server

- Spring Cloud Config provides server and client-side support for externalized configuration in a distributed system.
- With the Config Server you have a central place to manage external properties for applications across all environments.
- The concepts on both client and server map identically to the Spring Environment and PropertySource abstractions, so they fit very well with Spring applications, but can be used with any application running in any language.



# Spring Cloud Config Server

- As an application moves through the deployment pipeline from dev to test and into production you can manage the configuration between those environments and be certain that applications have everything they need to run when they migrate.
- The default implementation of the server storage backend uses git so it easily supports labelled versions of configuration environments, as well as being accessible to a wide range of tooling for managing the content. It is easy to add alternative implementations and plug them in with Spring configuration.



## Features

---

- Spring Cloud Config Server features:
- HTTP, resource-based API for external configuration (name-value pairs, or equivalent YAML content)
- Encrypt and decrypt property values (symmetric or asymmetric)
- Embeddable easily in a Spring Boot application using `@EnableConfigServer`



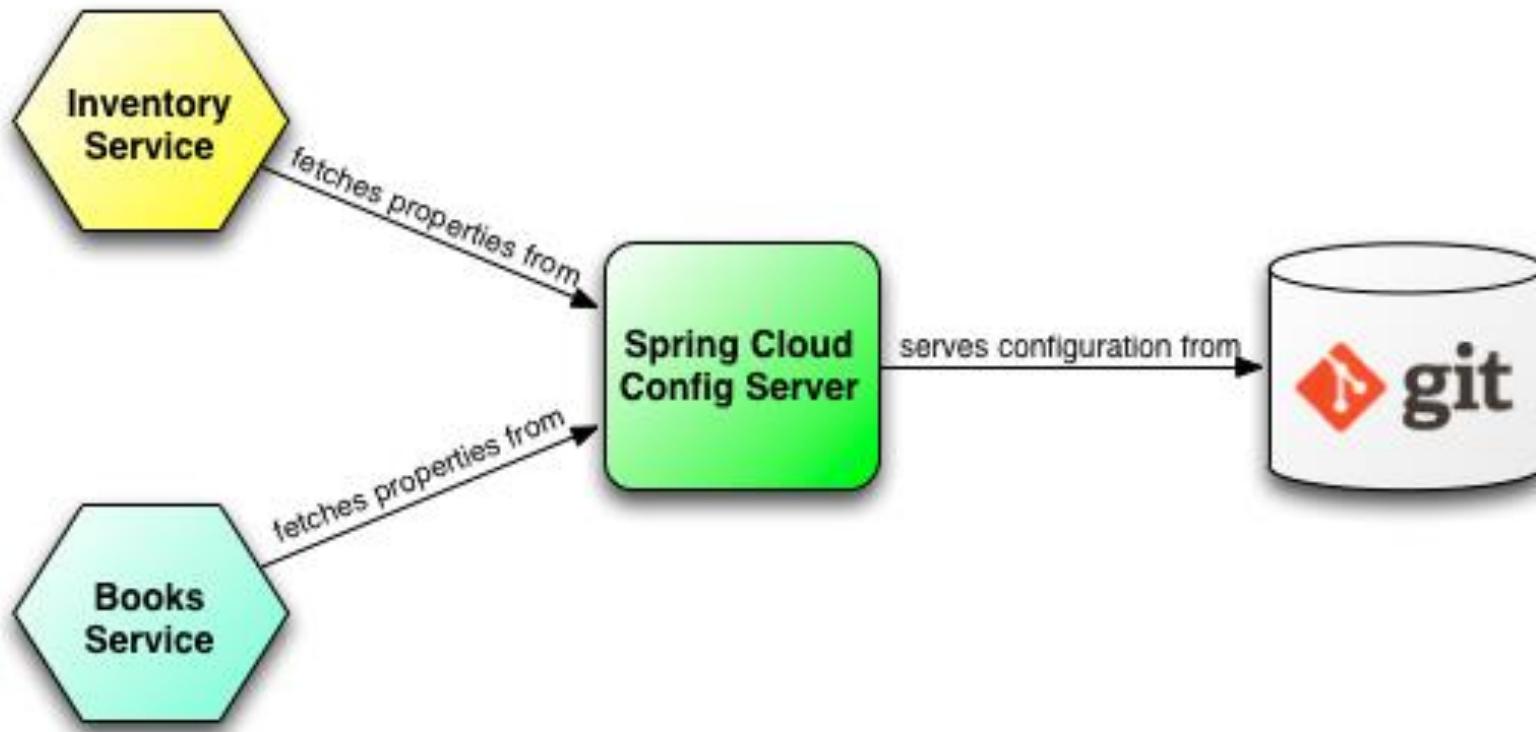
## Client side Features

---

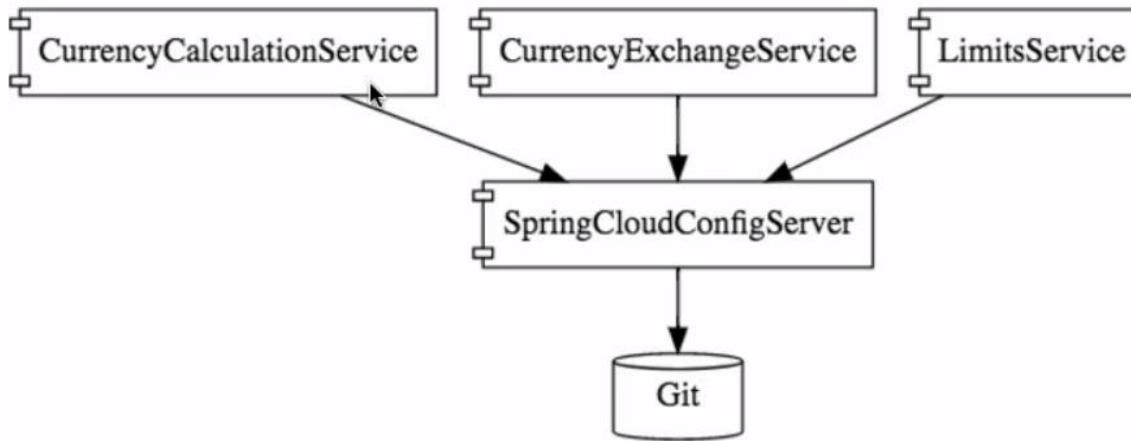
- Config Client features (for Spring applications):
- Bind to the Config Server and initialize Spring Environment with remote property sources
- Encrypt and decrypt property values (symmetric or asymmetric)



# Spring Cloud Config Server

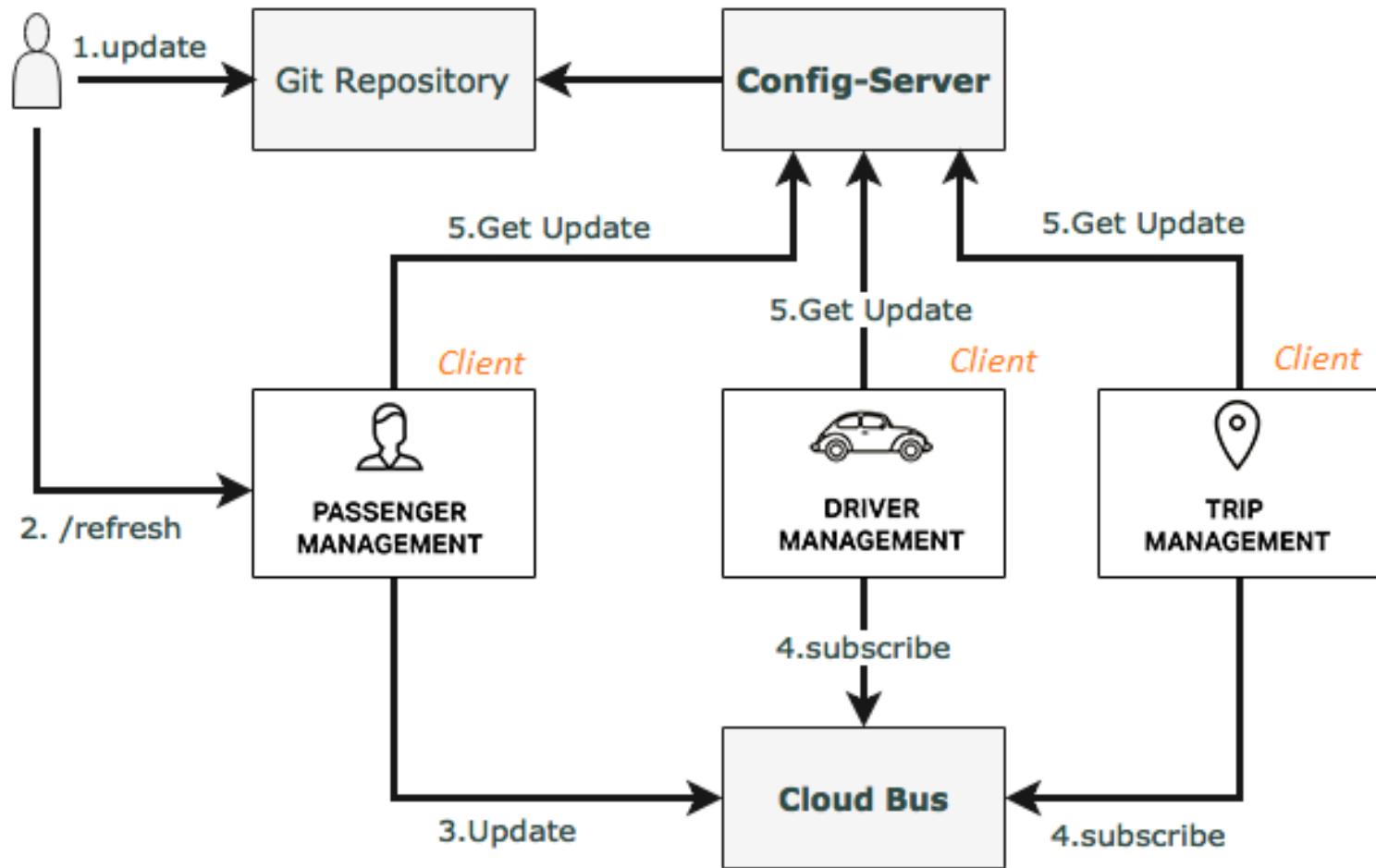


# Spring Cloud



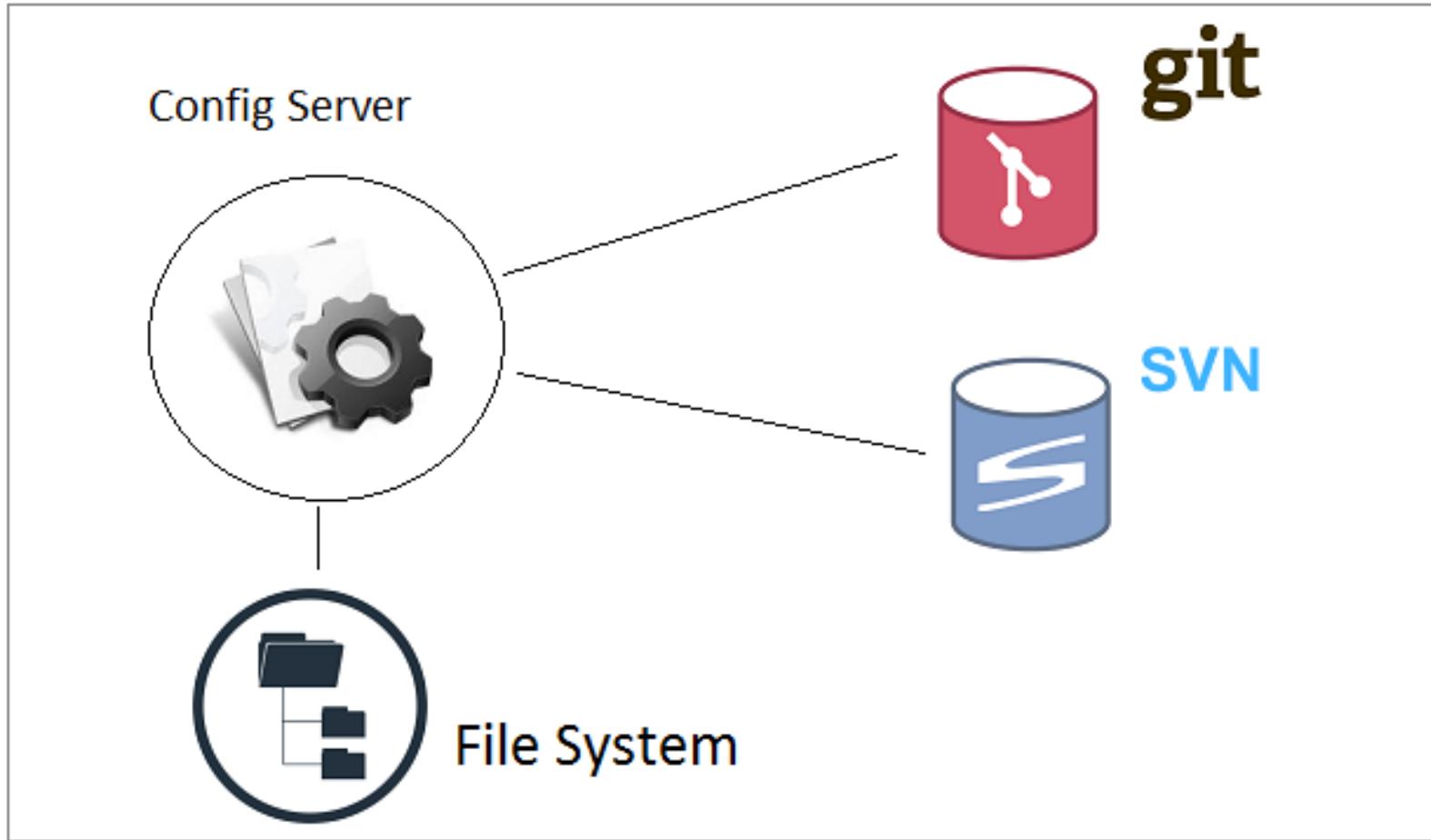
*Spring Cloud Config Server*

# Spring Cloud Bus





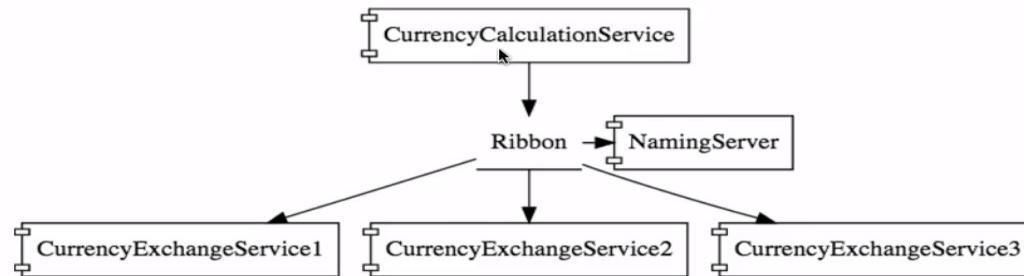
# How does Config Server Store Data



## DYNAMIC SCALE UP AND DOWN

- Naming Server (Eureka)
- Ribbon (Client Side Load Balancing)
- Feign (Easier REST Clients)

# Spring Cloud



*Ribbon Load Balancing*

## DYNAMIC SCALE UP AND DOWN

- Naming Server (Eureka)
- Ribbon (Client Side Load Balancing)
- Feign (Easier REST Clients)

## VISIBILITY AND MONITORING

- Zipkin Distributed Tracing
- Netflix API Gateway

# Important Spring Cloud Modules



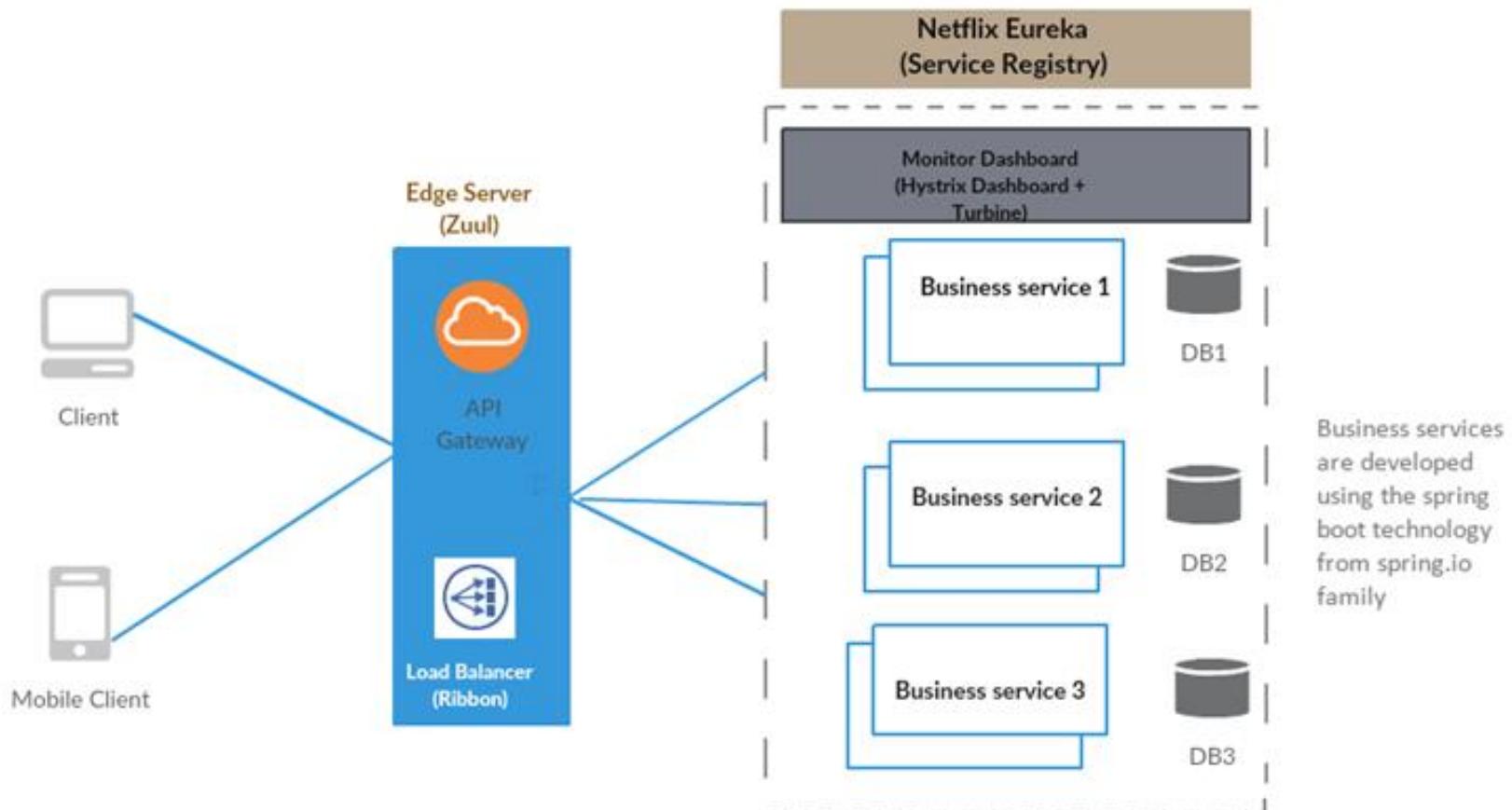
- Dynamic Scale Up and Down. Using a combination of
- Naming Server (Eureka)
- Ribbon (Client Side Load Balancing)
- Feign (Easier REST Clients)
- Visibility and Monitoring with
- Zipkin Distributed Tracing
- Netflix API Gateway
- Configuration Management with
- Spring Cloud Config Server
- Fault Tolerance with
- Hystrix

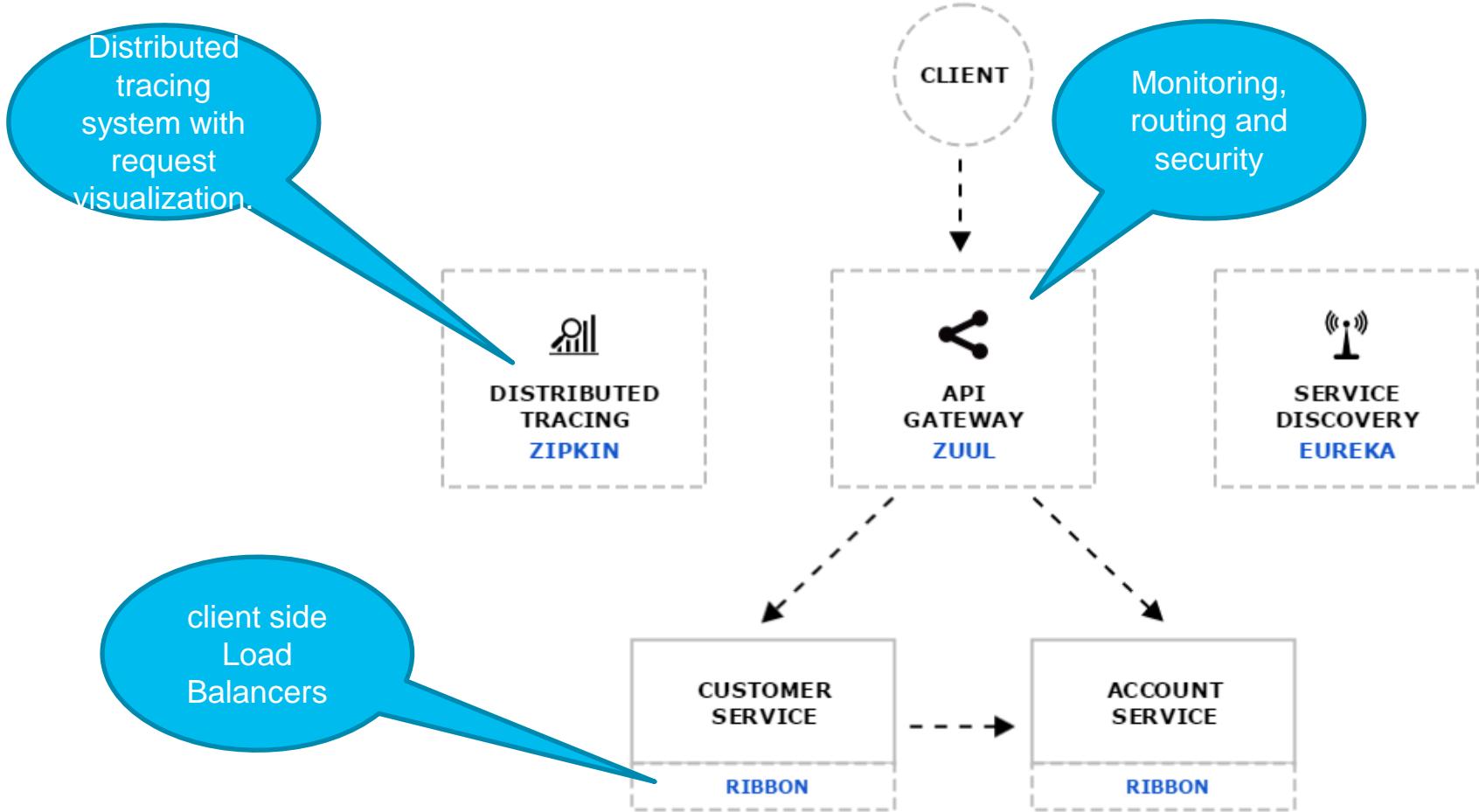


# Standardized ports

## Ports

Application	Port
Limits Service	8080, 8081, ...
Spring Cloud Config Server	8888
Currency Exchange Service	8000, 8001, 8002, ..
Currency Conversion Service	8100, 8101, 8102, ...
Netflix Eureka Naming Server	8761
Netflix Zuul API Gateway Server	8765
Zipkin Distributed Tracing Server	9411







Create stand-alone Spring applications

Embed Tomcat, Jetty or Undertow directly (no need to deploy WAR files)

Provide opinionated 'starter' POMs to simplify your Maven configuration

Automatically configure Spring whenever possible

Provide production-ready features such as metrics, health checks and externalized configuration

Absolutely no code generation and no requirement for XML configuration

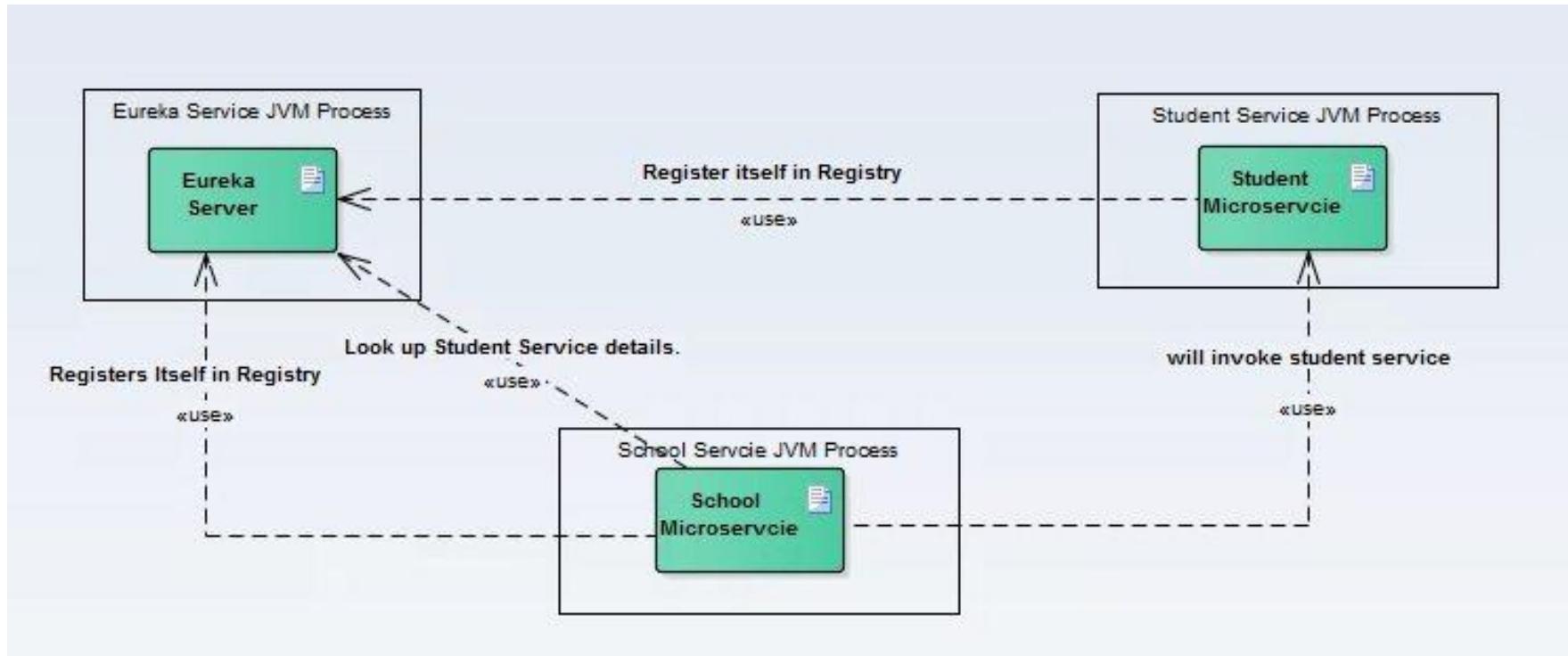


## Why Eureka Server

---

- In the distributed computing are there is a concept called 'Service registration and discovery' where one dedicated server is responsible to maintain the registry of all the Microservice that has been deployed and removed. This will act like a phone book of all other applications/microservices.

# Spring Cloud Service Discovery with Netflix Eureka





# Server Configuration

---

- server:
- port: \${PORT:8761} # Indicate the default PORT where this service will be started
- 
- eureka:
- client:
  - registerWithEureka: false #telling the server not to register himself in the service registry
  - fetchRegistry: false
- server:
- waitTimeInMsWhenSyncEmpty: 0 #wait time for subsequent sync



# Server Configuration

Name	Description
spring.application.name	Unique name for a Eureka server service.
eureka.client.serviceUrl.defaultZone	It consults with other Eureka servers to sync the service registry. As it is in standalone mode, I am giving the local server address.
server.port	In which port the server will be bound.
eureka.client.register-with-eureka	This determines if this server registers itself as a client; as I said earlier, the Eureka server is also acting as a client so that it can sync the registry. The value being false means it prevents itself from acting as a client.
eureka.client.fetch-registry	Does not register itself in the service registry.



# Client Configuration

---

- server:
- port: 8098 #default port where the service will be started
- 
- eureka: #tells about the Eureka server details and its refresh time
- instance:
- leaseRenewalIntervalInSeconds: 1
- leaseExpirationDurationInSeconds: 2
- client:



# Client Configuration

---

- serviceUrl:
- defaultZone: http://127.0.0.1:8761/eureka/
- healthcheck:
- enabled: true
- lease:
- duration: 5
- 
- spring:
- application:
- name: student-service #current service name to be used by the eureka server
-



# Client Configuration

---

- management:
- security:
- enabled: false #disable the spring security on the management endpoints like /env, /refresh etc.
- 
- logging:
- level:
- com.example.howtodoinjava: DEBUG



# Client Configuration

- If `registerWithEureka` is true, an instance registers with a Eureka server using a given URL; then onwards, it sends heartbeats every 30s (configurable by `eureka.instance.leaseRenewalIntervalInSeconds`).
- If the server doesn't receive a heartbeat, it waits 90s (configurable by `eureka.instance.leaseExpirationDurationInSeconds`) before removing the instance from registry and there by disallowing traffic to that instance.
- Sending heartbeat is an asynchronous task;
- if the operation fails, it backs off exponentially by a factor of 2 until a maximum delay of `eureka.instance.leaseRenewalIntervalInSeconds * eureka.client.heartbeatExecutorExponentialBackOffBound` is reached.
- There is no limit to the number of retries for registering with Eureka.



# Eureka Admin Server

Spring Boot Admin

Wallboard Applications Journal About en ▾

APPLICATIONS

2

INSTANCES

2

STATUS

all up



UP

✓ Customer-Service,virtusa-banking,config-server-client

1m <http://DESKTOP-55AGI0I:7070/>

✓ virtusa-eureka-server

2m <http://DESKTOP-55AGI0I:8761/>



# Eureka Admin Server

Spring Boot Admin

Wallboard Applications Journal About en ▾

3m

Customer-  
Service,virtusa-  
banking,config-  
server-client

1 instance

5m

virtusa-eureka-  
server

1 instance

# Consul Service Registration and Discovery Example

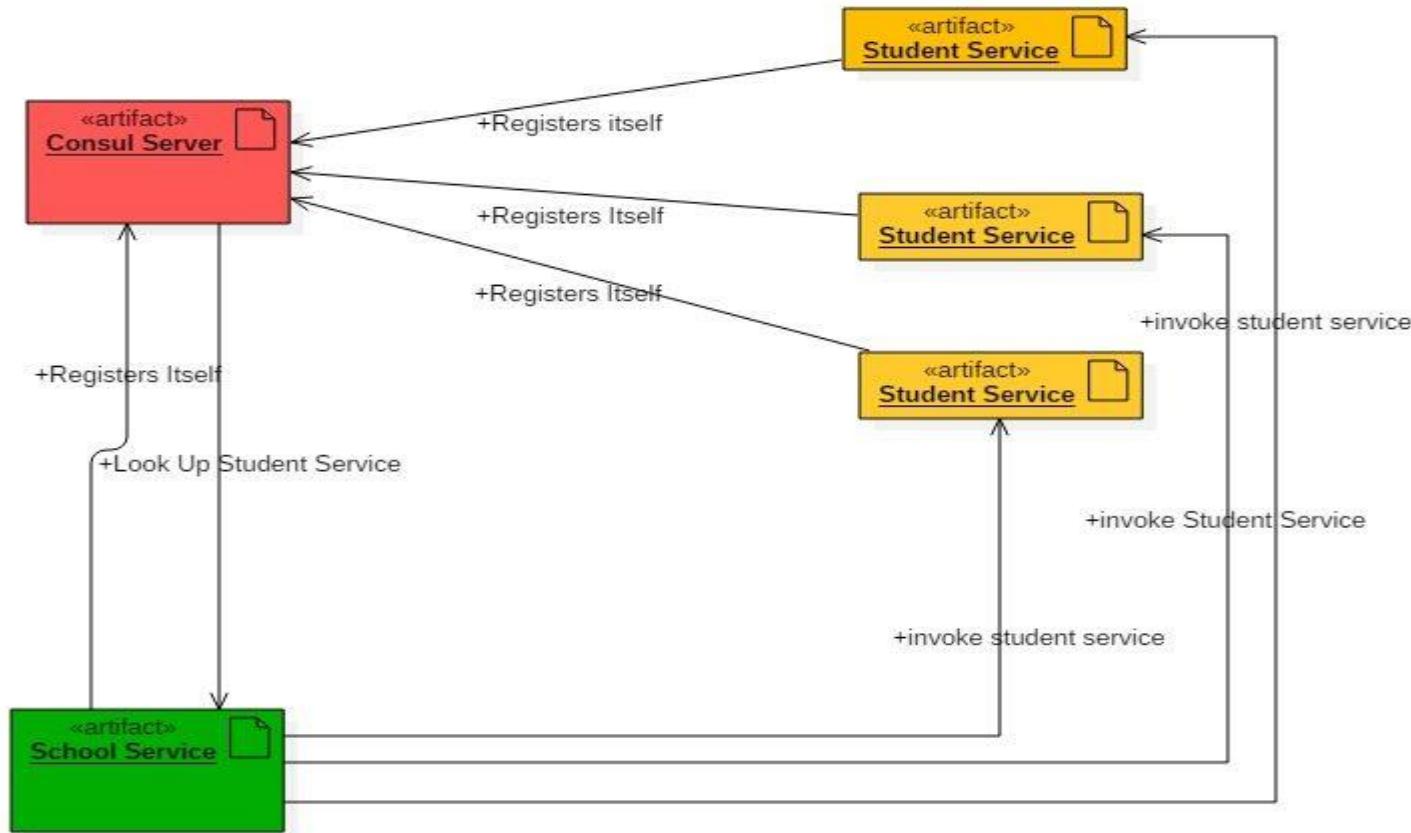


- Consul provides multiple features like service discovery, configuration management, health checking and key-value store etc.

# Consul Service Registration and Discovery Example



- **Consul Agent** – running on localhost acting as discovery/registry server functionality.



# Configuring Consul in Local workstation



- Download from Consul portal. Choose particular package based on the operating System. Once downloaded the zip, we need to unzip it to desired place.
- Start Consul Agent in local workstation – The Zip file that we have unzipped, has only one exe file called `consul.exe`. We will start a command prompt here and use below command to start the agent.
- `consul agent -server -bootstrap-expect=1 -data-dir=consul-data -ui -bind=192.168.99.1`

# Configuring Consul in Local workstation



- Make sure you enter the correct bind address, it would be different depending on the LAN settings. Do a ipconfig in command prompt to know your IPv4 address and use it here.

A screenshot of a Windows Command Prompt window. The title bar says "C:\Windows\system32\cmd.exe". The command line shows "F:\Study\installations\consul\_0.9.0\_windows\_amd64>ipconfig". The window is mostly blank, indicating no output has been displayed yet.

# Configuring Consul in Local workstation



```
C:\Windows\system32\cmd.exe - consul agent -server -bootstrap-expect=1 -data-dir=consul-data -ui -bind=192.168.6.1

F:\Study\installations\consul_0.9.0_windows_amd64>consul agent -server -bootstrap-expect=1 -data-dir=consul-data -ui -bind=192.168.6.1
==> WARNING: BootstrapExpect Mode is specified as 1; this is the same as Bootstrap mode.
==> WARNING: Bootstrap mode enabled! Do not enable unless necessary
==> Starting Consul agent...
==> Consul agent running!
  Version: 'v0.9.0'
    Node ID: 'f8db8d09-ac67-e997-9306-aeb20e4ecd3e'
    Node name: 'Sajal-HP'
    Datacenter: 'dc1'
      Server: true <Bootstrap: true>
    Client Addr: 127.0.0.1 <HTTP: 8500, HTTPS: -1, DNS: 8600>
    Cluster Addr: 192.168.6.1 <LAN: 8301, WAN: 8302>
  Gossip encrypt: false, RPC-TLS: false, TLS-Incoming: false

==> Log data will now stream in as it occurs:

2017/07/20 13:31:42 [INFO] raft: Initial configuration <index=1>: [{Suffrage:Voter ID:192.168.6.1:8300 Address:192.168.6.1:8300}]
2017/07/20 13:31:42 [INFO] raft: Node at 192.168.6.1:8300 [Follower] entering Follower state <Leader: "">
2017/07/20 13:31:42 [INFO] serf: EventMemberJoin: Sajal-HP.dc1 192.168.6.1
2017/07/20 13:31:42 [WARN] serf: Failed to re-join any previously known node
2017/07/20 13:31:42 [INFO] serf: EventMemberJoin: Sajal-HP 192.168.6.1
2017/07/20 13:31:42 [INFO] consul: Handled member-join event for server "Sajal-HP.dc1" in area "wan"
2017/07/20 13:31:42 [INFO] consul: Adding LAN server Sajal-HP <Addr: tcp/192.168.6.1:8300> <DC: dc1>
2017/07/20 13:31:42 [WARN] serf: Failed to re-join any previously known node
2017/07/20 13:31:42 [INFO] agent: Started DNS server 127.0.0.1:8600 <udp>
2017/07/20 13:31:42 [INFO] agent: Started DNS server 127.0.0.1:8600 <tcp>
2017/07/20 13:31:42 [INFO] agent: Started HTTP server on 127.0.0.1:8500
2017/07/20 13:31:49 [ERR] agent: failed to sync remote state: No cluster leader
2017/07/20 13:31:49 [ERR] http: Request GET /v1/catalog/services?wait=2s&index=169, error: No cluster leader from=127.0.0.1:53785
2017/07/20 13:31:50 [ERR] http: Request GET /v1/catalog/services, error: No cluster leader from=127.0.0.1:53789
2017/07/20 13:31:51 [ERR] http: Request GET /v1/catalog/services, error: No cluster leader from=127.0.0.1:53792
2017/07/20 13:31:52 [WARN] raft: Heartbeat timeout from "" reached, starting election
2017/07/20 13:31:52 [INFO] raft: Node at 192.168.6.1:8300 [Candidate] entering Candidate state in term 66
2017/07/20 13:31:52 [INFO] raft: Election won. Tally: 1
2017/07/20 13:31:52 [INFO] raft: Node at 192.168.6.1:8300 [Leader] entering Leader state
2017/07/20 13:31:52 [INFO] consul: cluster leadership acquired
2017/07/20 13:31:52 [INFO] consul: New leader elected: Sajal-HP
2017/07/20 13:31:52 [WARN] agent: Check 'service:student-service-9099' is now critical
2017/07/20 13:31:52 [WARN] agent: Check 'service:student-service-9097' is now critical
2017/07/20 13:31:54 [INFO] agent: Synced check 'service:student-service-9097'
2017/07/20 13:31:54 [INFO] agent: Synced check 'service:school-service-8098'
2017/07/20 13:31:54 [INFO] agent: Synced check 'service:student-service-9099'
2017/07/20 13:31:54 [INFO] agent: Synced check 'service:student-service-9098'
```

# Configuring Consul in Local workstation



- **Test whether Consul Server is running** – Consul runs on default port and once agent started successfully, browse <http://localhost:8500/ui> and you should see a console screen like –

A screenshot of a web browser window displaying the Consul UI. The address bar shows the URL <http://localhost:8500/ui/#/dc1/services>. The browser's toolbar includes icons for Apps, Bookmarks, Suggested Sites, and various links like myemail.accenture.co, Stocks, Imported From IE, Study, Gmail, YouTube, Stock/Share Market, The Economic Times, and social media links for Facebook and Google+. The main interface has a navigation bar with tabs: SERVICES (highlighted in pink), NODES, KEY/VALUE, ACL, DC1 (highlighted in green with a dropdown arrow), and a settings gear icon. Below the tabs are filters: 'Filter by name' (text input), 'any status' (dropdown menu), and 'EXPAND' (button). A search bar contains the text 'consul'. To the right, it says '1 passing'. The background of the page is white.



# Consul Client

---

- **Create Student Project**
- Create a Spring boot project from initializer portal with four dependencies i.e.
- Actuator
- Web
- Rest Repositories
- Consul Discovery



# Service Configuration

- `server.port=9098` – will start the service in default 9098 port.
- `spring.application.name: student-service` – will registers itself in consul server using student-service tag and also other services will lookup this service with this name itself.
- `management.security.enabled=false` – is not actually required for this exercise, but it will disable spring security in the management endpoints provided by actuator module.

# Hystrix Circuit Breaker Pattern – Spring Cloud



- It is generally required to enable fault tolerance in the application where some underlying service is down/throwing error permanently, we need to fall back to different path of program execution automatically.
- This is related to distributed computing style of Eco system using lots of underlying Microservices.
- This is where circuit breaker pattern helps and Hystrix is an tool to build this circuit breaker.

# What is Circuit Breaker Pattern?



- If we design our systems on microservice based architecture, we will generally develop many Microservices and those will interact with each other heavily in achieving certain business goals.
- Now, all of us can assume that this will give expected result if all the services are up and running and response time of each service is satisfactory.

# What is Circuit Breaker Pattern?



- Now what will happen if any service, of the current Eco system, has some issue and stopped servicing the requests.
- It will result in timeouts/exception and the whole Eco system will get unstable due to this single point of failure.

# What is Circuit Breaker Pattern?



- Here circuit breaker pattern comes handy and it redirects traffic to a fall back path once it sees any such scenario.
- Also it monitors the defective service closely and restore the traffic once the service came back to normalcy.

# What is Circuit Breaker Pattern?



- So circuit breaker is a kind of a wrapper of the method which is doing the service call and it monitors the service health and once it gets some issue, the circuit breaker trips and all further calls goto the circuit breaker fall back and finally restores automatically once the service came back !! That's cool right?



# Circuit Breaker States

---

- The circuit breaker has 3 distinct states, Closed, Open, and Half-Open.
- Closed – When everything is normal, the circuit breaker remains in the closed state and all calls pass through to the services.
- When the number of failures exceeds a predetermined threshold the breaker trips, and it goes into the Open state.

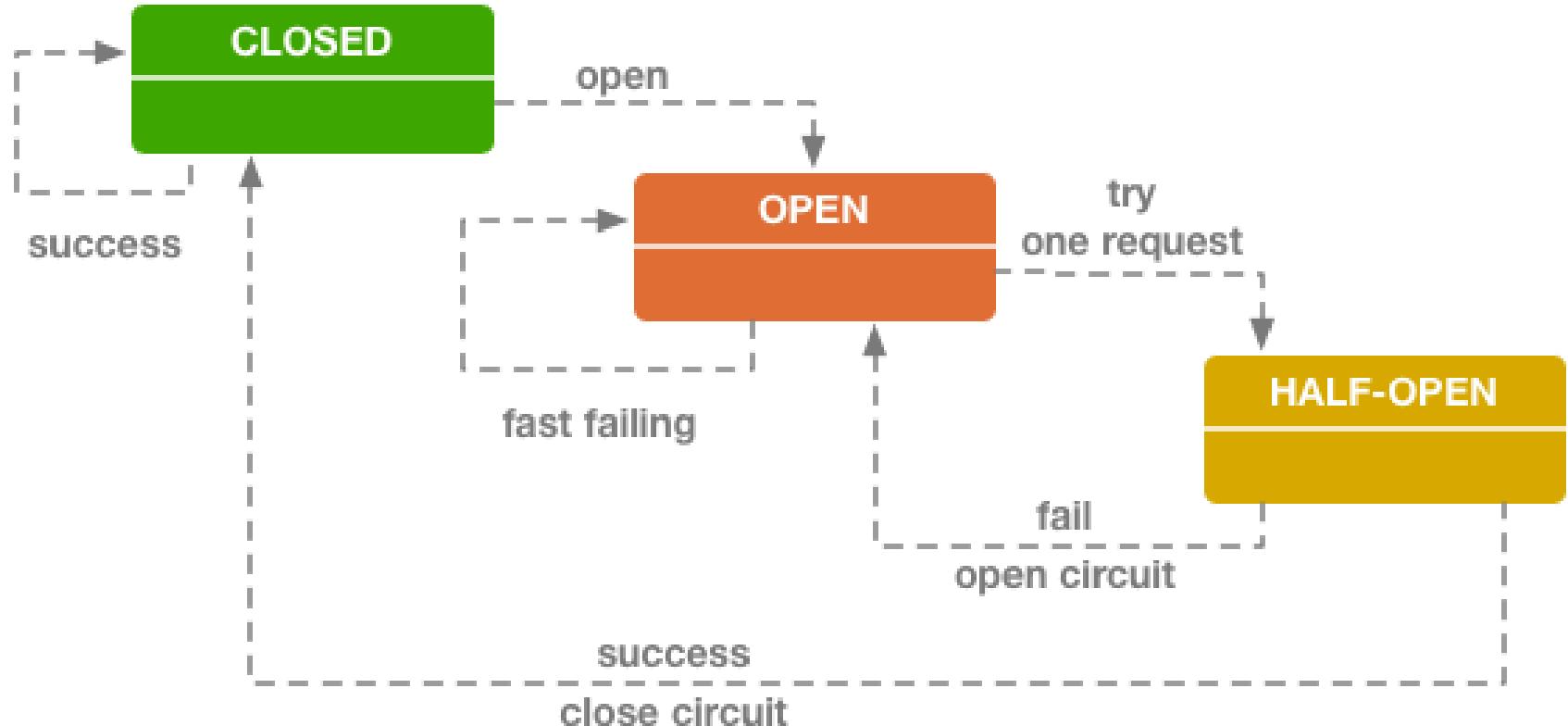


# Circuit Breaker States

---

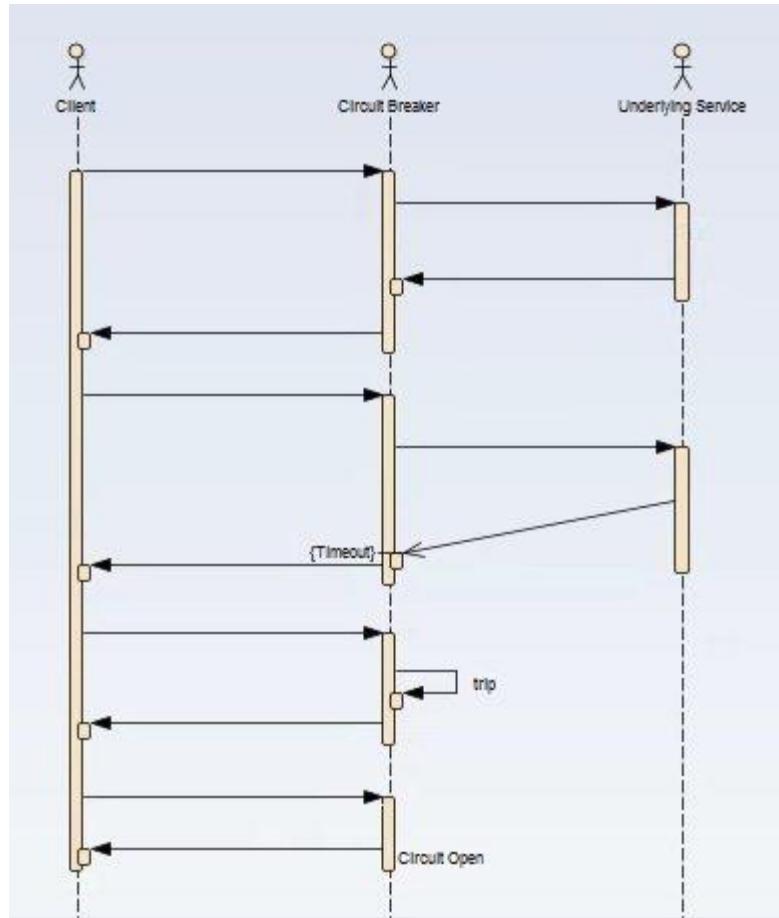
- Open – The circuit breaker returns an error for calls without executing the function.
- Half-Open – After a timeout period, the circuit switches to a half-open state to test if the underlying problem still exists.
- If a single call fails in this half-open state, the breaker is once again tripped. If it succeeds, the circuit breaker resets back to the normal closed state

# Circuit Breaker States



Circuit Breaker State Diagram

# What is Circuit Breaker Pattern?



# Hystrix Dashboard



- **<http://localhost:9091/hystrix>**
- **<http://localhost:9091/actuator/hystrix.stream>** – It's a continuous stream that Hystrix generates. It is just a health check result along with all the service calls that are being monitored by Hystrix. Sample output will look like in browser –



Eureka | localhost:8765/ | Hystrix Circuit E | localhost:9091/ | Consul by Hash | Spring Initializr | New Tab | localhost:9091/ | Hystrix Dashboard | + | - | X

localhost:9091/hystrix

Insert title here Empire New Tab How to use Assertion Browser Automation node.js - How can I fi Freelancer-dev-8104 Courses



## Hystrix Dashboard

<http://localhost:9091/actuator/hystrix.stream>

*Cluster via Turbine (default cluster): http://turbine-hostname:port/turbine.stream  
Cluster via Turbine (custom cluster): http://turbine-hostname:port/turbine.stream?cluster=[clusterName]  
Single Hystrix App: http://hystrix-app:port/actuator/hystrix.stream*

Delay:  ms Title:





Eureka | localhost:8765/ | Hystrix Circuit E | localhost:9091/ | Consul by Hash | Spring Initializr | New Tab | localhost:9091/ | Hystrix Monitor | + | - | X

localhost:9091/hystrix/monitor?stream=http%3A%2F%2Flocalhost%3A9091%2Factuator%2Fhystrix.stream

Apps Insert title here Empire New Tab How to use Assertion Browser Automation node.js - How can I fi Freelancer-dev-8104 Courses

## Hystrix Stream: http://localhost:9091/actuator/hystrix.stream



**Circuit** Sort: [Error then Volume](#) | [Alphabetical](#) | [Volume](#) | [Error](#) | [Mean](#) | [Median](#) | [90](#) | [99](#) | [99.5](#)

[Success](#) | [Short-Circuited](#) | [Bad Request](#) | [Timeout](#) | [Rejected](#) | [Failure](#) | [Error %](#)

callProductServiceAndGetData

Host: 0.0/s  
Cluster: 0.0/s  
Circuit **Closed**

Hosts	1	90th	0ms
Median	0ms	99th	0ms
Mean	0ms	99.5th	0ms

**Thread Pools** Sort: [Alphabetical](#) | [Volume](#) |

ProductDelegate

Host: 0.0/s  
Cluster: 0.0/s

Active	0	Max Active	0
Queued	0	Executions	0
Pool Size	1	Queue Size	5

Type here to search

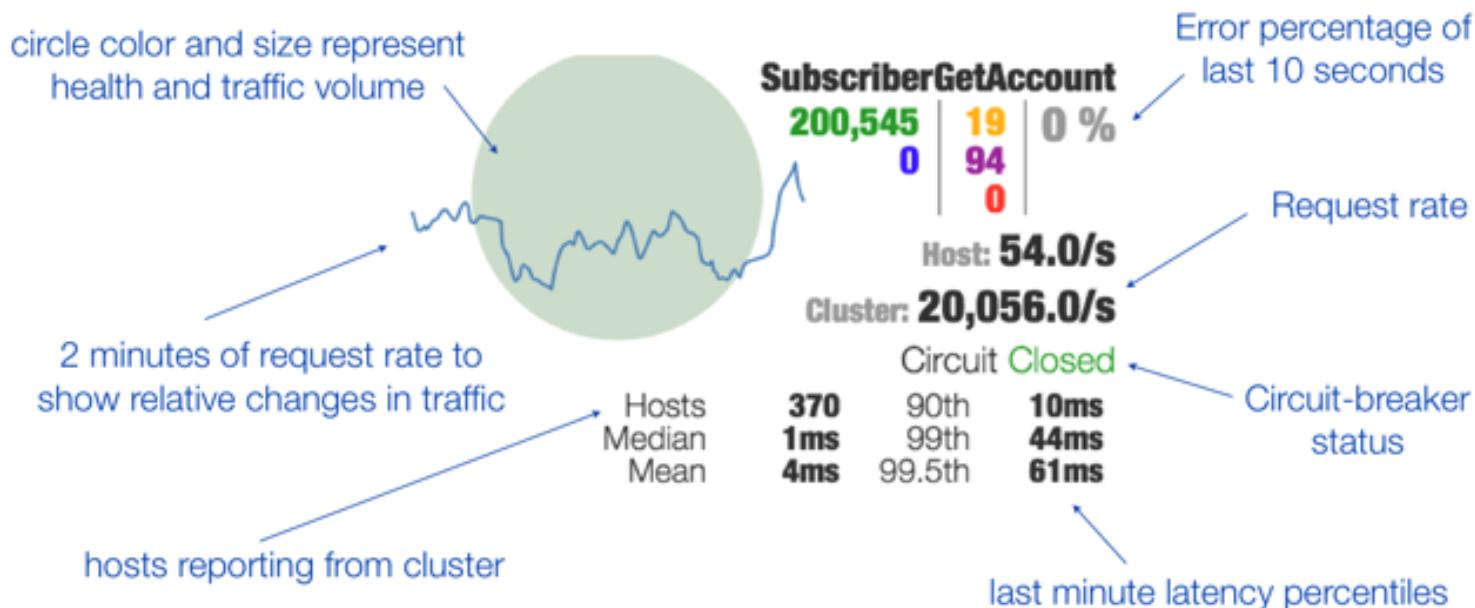
00:04 20/11/2018



Name	Description	Triggers Fallback?
EMIT	value delivered (HystrixObservableCommand only)	NO
SUCCESS	execution complete with no errors	NO
FAILURE	execution threw an Exception	YES
TIMEOUT	execution started, but did not complete in the allowed time	YES
BAD_REQUEST	execution threw a HystrixBadRequestException	NO
SHORT_CIRCUITED	circuit breaker <b>OPEN</b> , execution not attempted	YES
THREAD_POOL_REJECTED	thread pool at capacity, execution not attempted	YES
SEMAPHORE_REJECTED	semaphore at capacity, execution not attempted	YES



Name	Description	Throws Exception?
FALLBACK_EMIT	fallback value delivered (HystrixObservableCommand only)	NO
FALLBACK_SUCCESS	fallback execution complete with no errors	NO
FALLBACK_FAILURE	fallback execution threw an error	YES
FALLBACK_REJECTION	fallback semaphore at capacity, fallback not attempted	YES
FALLBACK_MISSING	no fallback implemented	YES



Rolling 10 second counters  
with 1 second granularity

Successes	<b>200,545</b>	Thread timeouts
Short-circuited (rejected)	<b>0</b>	Thread-pool Rejections
	<b>0</b>	Failures/Exceptions



# Circuit Breaker

---

- `@HystricCommand(fallbackMethod="defaultProductList", commandProperties = {  
 @HystrixProperty(name="execution.isolation.thread.timeoutInMilliseconds", value="500")})`

# Using Blue-Green Deployment to Reduce Downtime and Risk

---



- Blue-green deployment is a technique that reduces downtime and risk by running two identical production environments called Blue and Green.
- At any time, only one of the environments is live, with the live environment serving all production traffic. For this example, Blue is currently live and Green is idle.

# Using Blue-Green Deployment to Reduce Downtime and Risk

---

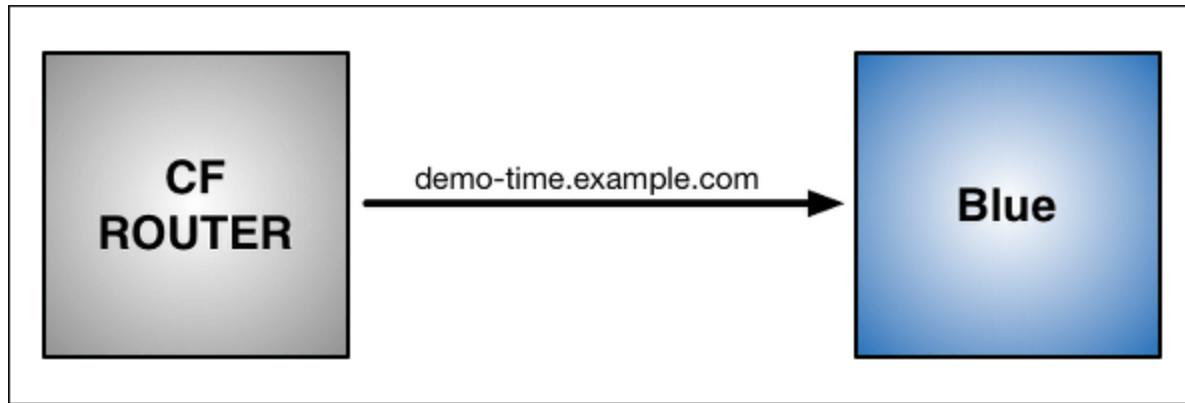


- As you prepare a new version of your software, deployment and the final stage of testing takes place in the environment that is *not* live: in this example, Green.
- Once you have deployed and fully tested the software in Green, you switch the router so all incoming requests now go to Green instead of Blue. Green is now live, and Blue is idle.

# Blue-Green Deployment with Cloud Foundry



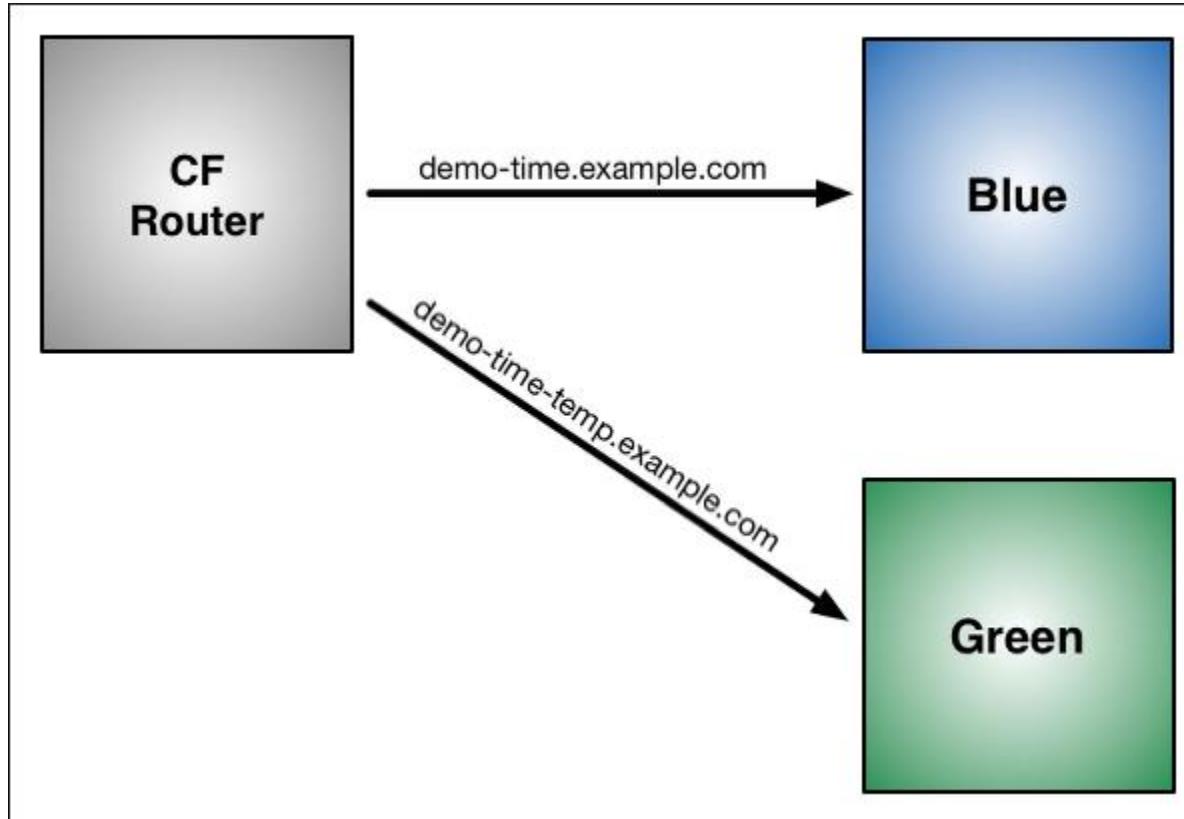
- \$ cf push Blue -n demo-time



# Blue-Green Deployment with Cloud Foundry



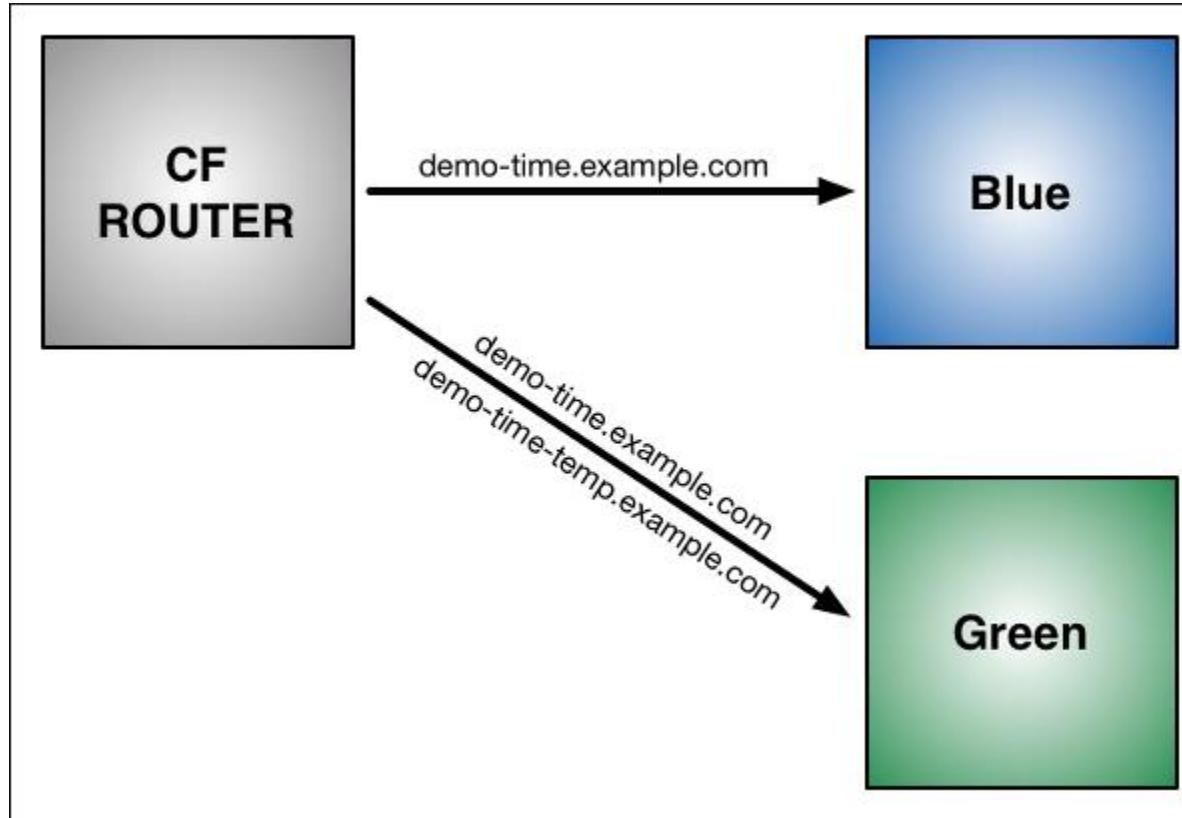
- \$ cf push Green -n demo-time-temp



# Blue-Green Deployment with Cloud Foundry



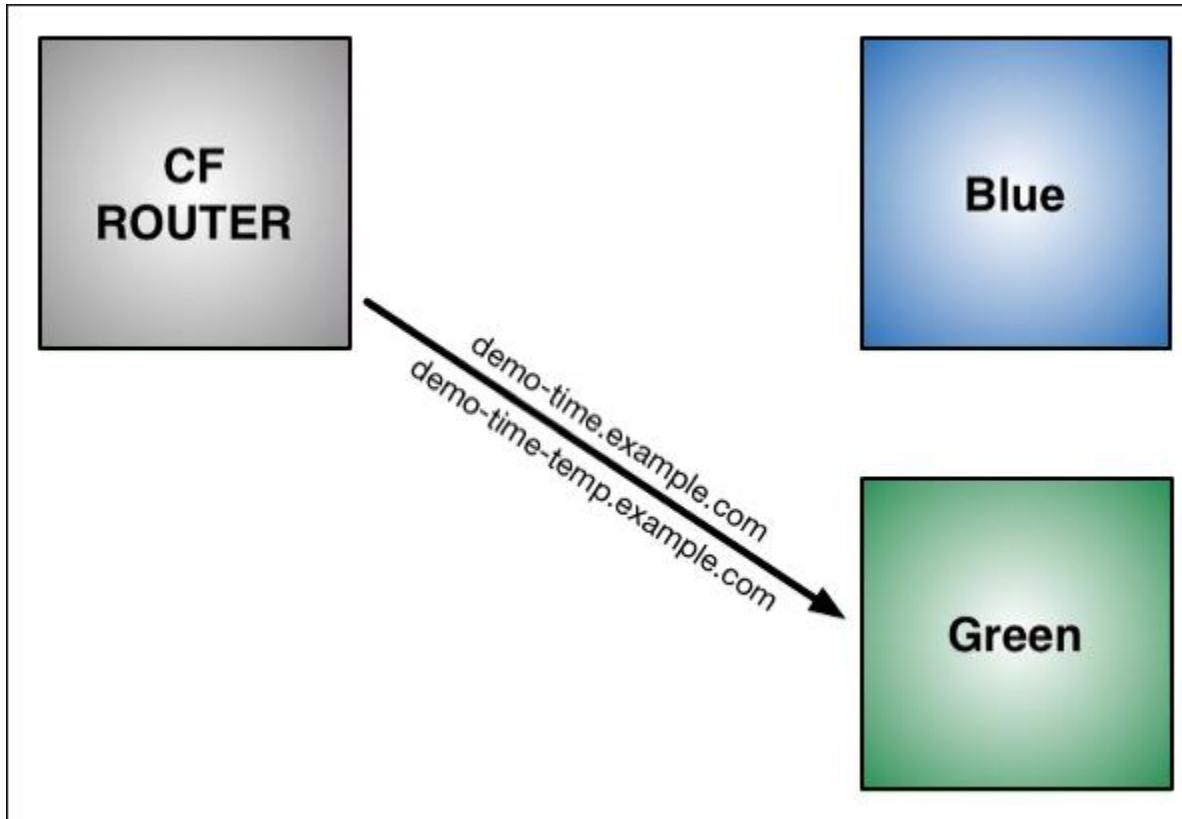
- \$ cf map-route Green example.com -n demo-time
- Binding demo-time.example.com to Green... OK



# Blue-Green Deployment with Cloud Foundry



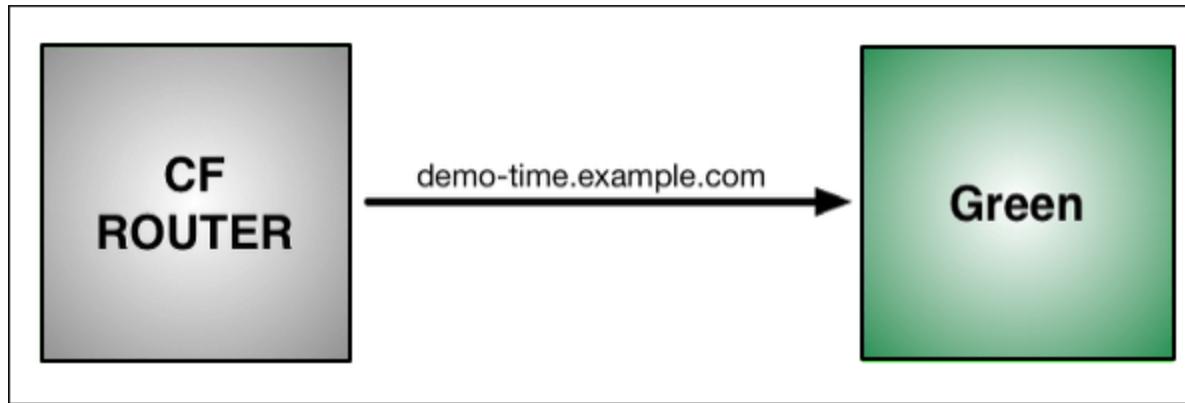
- \$ cf unmap-route Blue example.com -n demo-time
- Unbinding demo-time.example.com from blue... OK



# Blue-Green Deployment with Cloud Foundry



- cf delete-route



# How to Deploy Spring Boot Application to Cloud Foundry Platform

---



- **Cloud Foundry** is one of the Cloud Providers. It is a PaaS service where we can easily deploy and manage our applications and the Cloud Foundry will take care of the rest of the cloud based offerings like scalability, high availability etc.

# What is Cloud Foundry



- Cloud Foundry is an open-source platform as a service (PaaS) that provides you with a choice of clouds, developer frameworks, and application services.
- It is open source and it is governed by the Cloud Foundry Foundation.
- The original Cloud Foundry was developed by VMware and currently it is managed by Pivotal, a joint venture company by GE, EMC and VMware.

# What is Cloud Foundry



- Now since Cloud Foundry is open source product many popular organizations currently provides this platform separately and below are the list of current certified providers.
- Pivotal Cloud Foundry
- IBM Bluemix
- HPE Helion Stackato 4.0
- Atos Canopy
- CenturyLink App Fog
- GE Predix
- Huawei FusionStage
- SAP Cloud Platform
- Swisscom Application Cloud

# Cloud Foundry Installation for Windows



- Download the [CF Windows installer](#). It will prompt for the download. Save the zip file distribution.
- Unpack the zip file to a suitable place in your workstation.
- After successfully **unzip** operation, double click on the cf CLI executable.
- When prompted, click **Install**, then Close. Here are the sample steps for the same. This is very straight forward, you can select the default values.

# Cloud Foundry Installation for Windows



- Verify the installation by opening a terminal window and type cf. If your installation was successful, the cf CLI help listing appears. This indicates that you are ready to go with any cloud foundry platform from your local workstation.

# Setup PWS Console



Secure | https://account.run.pivotal.io/z/uaa/sign-up

**Pivotal.**

Create your Pivotal Account

First name

Last name

Email address

Password

Password confirmation

**Sign Up**

Already have an account? [Sign In](#)

# Login and logout from PWS Console using CLI



- Login to PWS – We will use **cf login -a api.run.pivotal.io** command to login to pivotal web service console from CLI tool that we have installed in our local workstation. It will logon the CLI tool to PWS platform so that we can deploy and manage our applications from our workstation. After giving command, it will ask for registered email and password and once provided successfully, it will logon to the platform.
- Logout from PWS Console – We will use command **cf logout** to logout from the platform, once we have all the work done for that session.



# Create Database in cloud foundry

Pivotal Account | M Inbox (1,408) - parameswaribala | R Welcome to Rediffmail | Pivotal Web Services | Spring Initializr | The HAL Browser (for Spring) | +

Apps Insert title here Empire New Tab How to use Assertion Browser Automation node.js - How can I fi Freelancer-dev-8104 Courses

press / eswaribala@rediffmail.com

Pivotal Web Services Search apps, services, spaces, & orgs

Home / eswaribala-org / development / kyc-cf

APP kyc-cf Starting

VIEW APP

Overview Service (1) Route (1) Logs Tasks Settings Buildpack: N/A

Bound Services

BIND SERVICE NEW SERVICE

Service	Instance Name	Binding Name
ClearDB MySQL Database free - Spark DB	virtusa_2018db	:

GIVE FEEDBACK

Pivotal © 2018 Pivotal Software Inc. All rights reserved. Terms | Privacy  
Last login: 11/20/18 8:33 pm

Type here to search

21:09 ENG 20/11/2018

# Push Application to Console



- cf push kyc-cf -p target\spring-helloworld-cf-0.0.1-SNAPSHOT.jar

```
Administrator: Command Prompt - cf push kyc-cf -p F:\Scope_SpringBoot_2018\KYCCFService\target\KYCCFService-0.0.1-SNAPSHOT.jar
User:          eswaribala@rediffmail.com
Org:          eswaribala-org
Space:        development

C:\WINDOWS\system32>cf push kyc-cf -p F:\Scope_SpringBoot_2018\KYCCFService\target\KYCCFService-0.0.1-SNAPSHOT.jar
Pushing app kyc-cf to org eswaribala-org / space development as eswaribala@rediffmail.com.
..
Getting app info...
Creating app with these attributes...
+ name:      kyc-cf
  path:      F:\Scope_SpringBoot_2018\KYCCFService\target\KYCCFService-0.0.1-SNAPSHOT.jar

  routes:
+   kyc-cf.cfapps.io

Creating app kyc-cf...
Mapping routes...
Comparing local files to remote cache...
Packaging files to upload...
Uploading files...
  320.00 KiB / 409.37 KiB [=====>-----] 78.17%
```



# Verify Application Deployed

Secure | https://console.run.pivotal.io/organizations/d9ba791e-2a9f-4107-a71b-7d7e464ac5d8

Pivotal Web Services

Search by App Name

sajal.chakraborty@gmail.com

Your org, "sajal.chakraborty", is still in trial. To get access to 25GB of memory and paid service plans, upgrade now

ORG  
sajal.chakraborty

SPACES  
development

Marketplace

Docs

Support

Tools

Blog

Status

ORG QUOTA  
sajal.chakraborty 1 GB / 2 GB 50%

Increase Quota

Billing Statement

Space (1) Domains (2) Member (1) Settings

development

APPS	SERVICES
1	0

+ Add a Space

50% of Org Quota

A screenshot of the Pivotal Web Services console. The left sidebar shows navigation links like 'ORG', 'SPACES', 'Marketplace', etc. The main area displays organization details: 'Your org, "sajal.chakraborty", is still in trial.' A message encourages upgrading to access more memory and paid service plans. It shows the current quota: 1 GB / 2 GB (50%). The 'development' space is selected, showing 1 app and 0 services. A red circle highlights the number '1' under 'APPS' in the development space summary.

# What is Zuul?



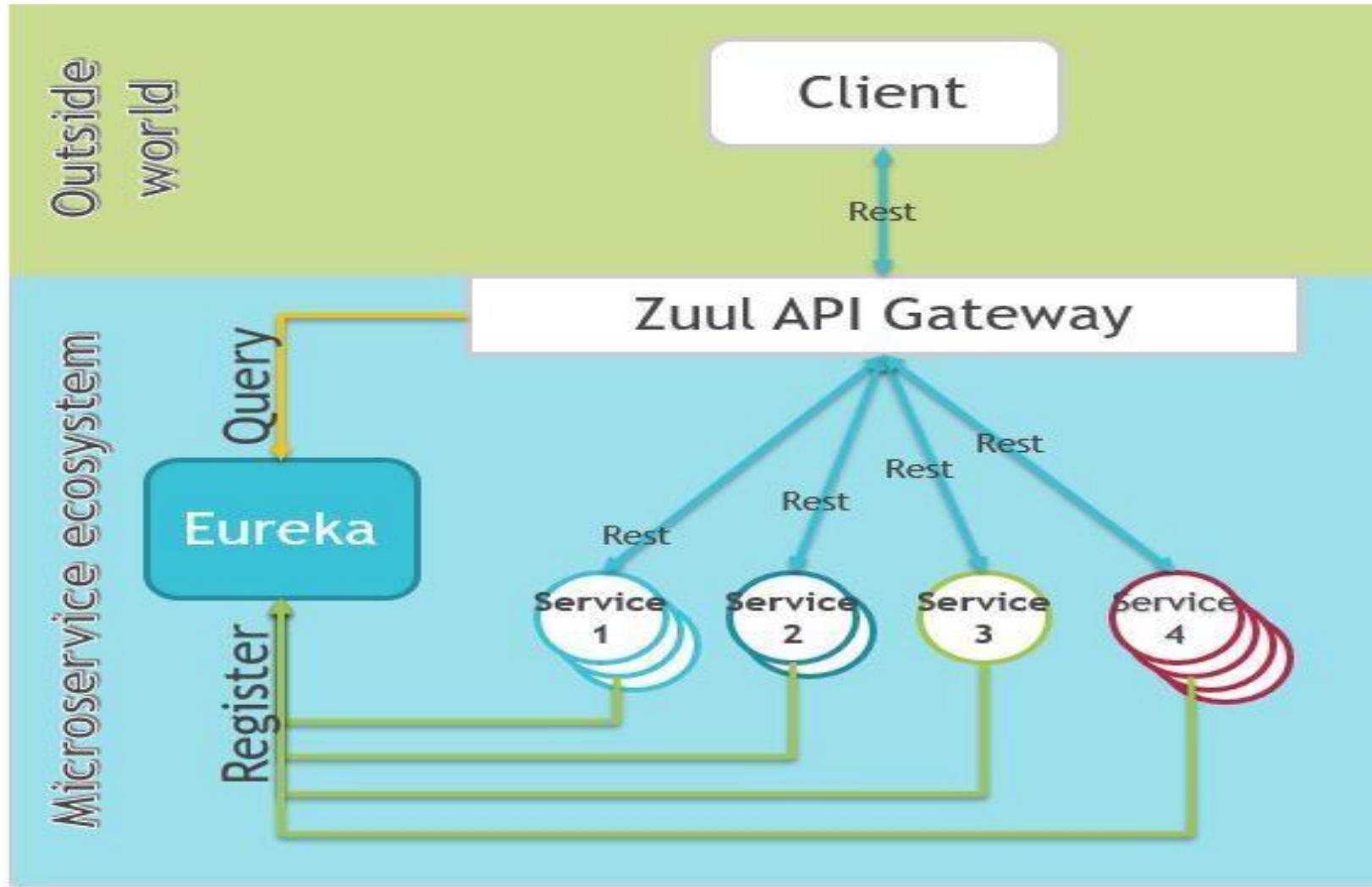
- *Zuul Server is an API Gateway application.*
- *It handles all the requests and performs the dynamic routing of microservice applications.*
- *It works as a front door for all the requests. It is also known as Edge Server.*
- *Zuul is built to enable dynamic routing, monitoring, resiliency, and security.*
- *It can also route the requests to multiple Amazon Auto Scaling Groups.*

# What is Zuul?

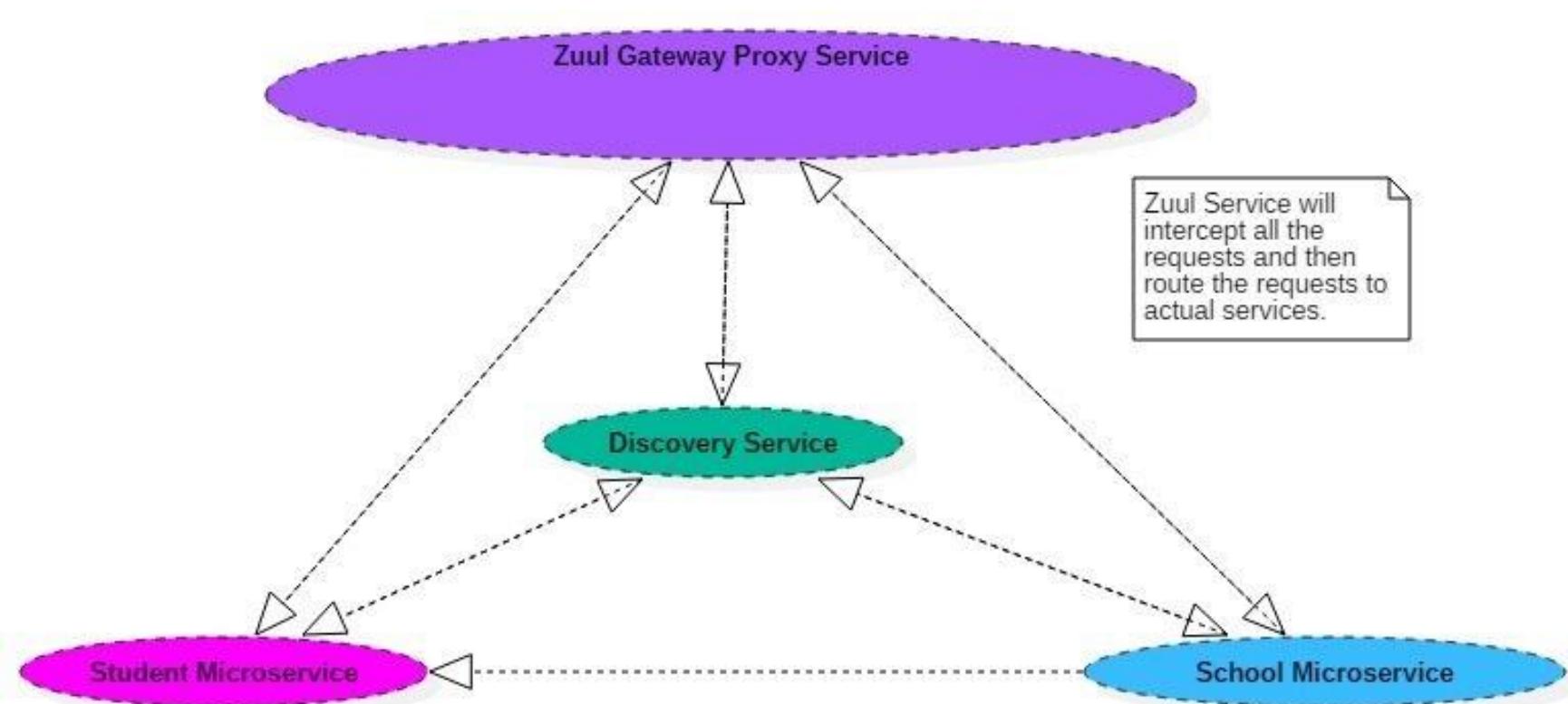


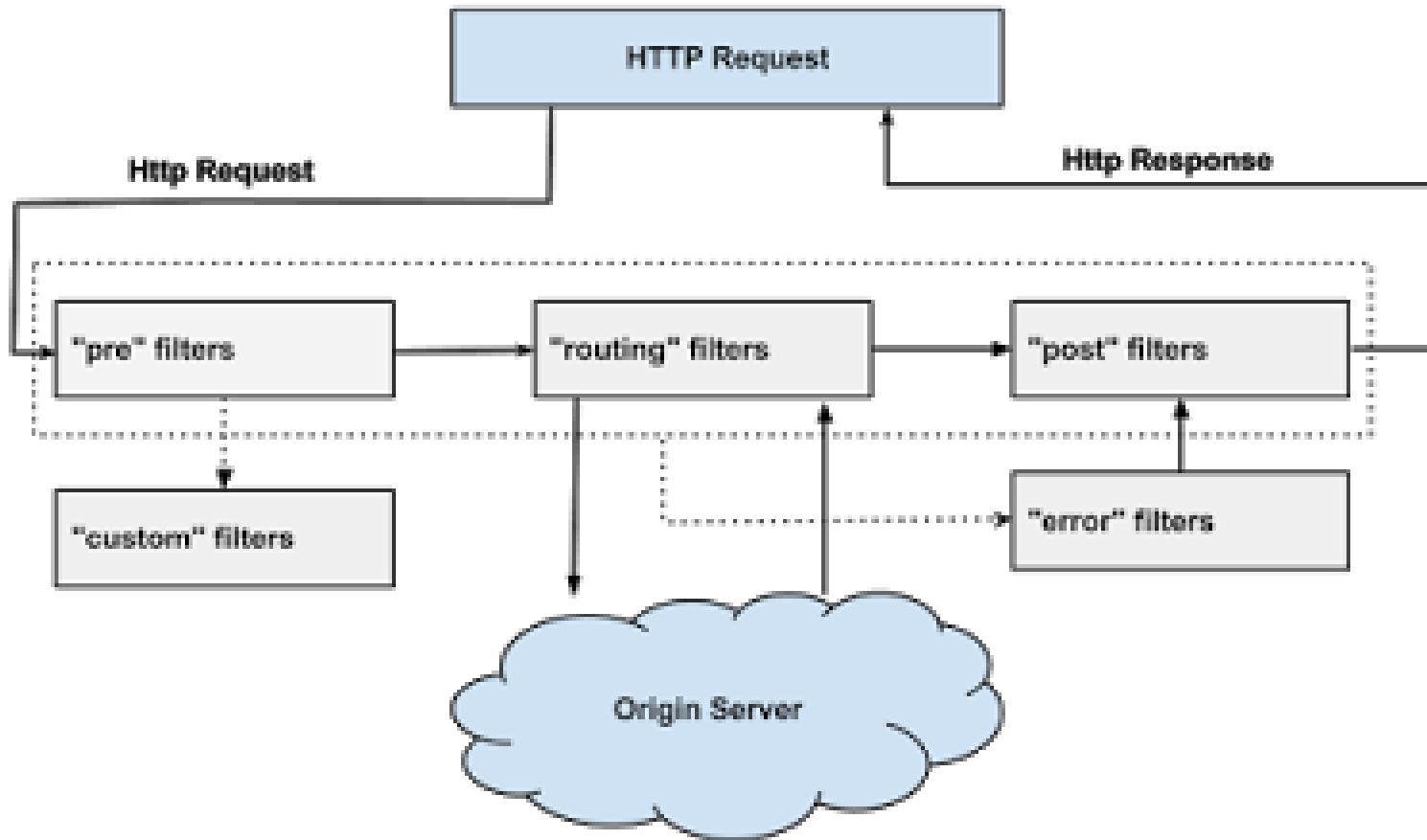
- *For Example, /api/products are mapped to the product service and /api/user is mapped to the user service. The Zuul Server dynamically routes the requests to the respective backend application.*

# What is Zuul?



# What is Zuul?





# *Why did we build Zuul?*



- *The volume and diversity of Netflix API traffic sometimes results in production issues arising quickly and without warning.*
- *We need a system that allows us to rapidly change behaviour in order to react to these situations.*

# Zuul Components

---



- Zuul has mainly four types of filters that enable us to intercept the traffic in different timeline of the request processing for any particular transaction.
- We can add any number of filters for a particular url pattern.
- **pre filters** – are invoked before the request is routed.
- **post filters** – are invoked after the request has been routed.
-

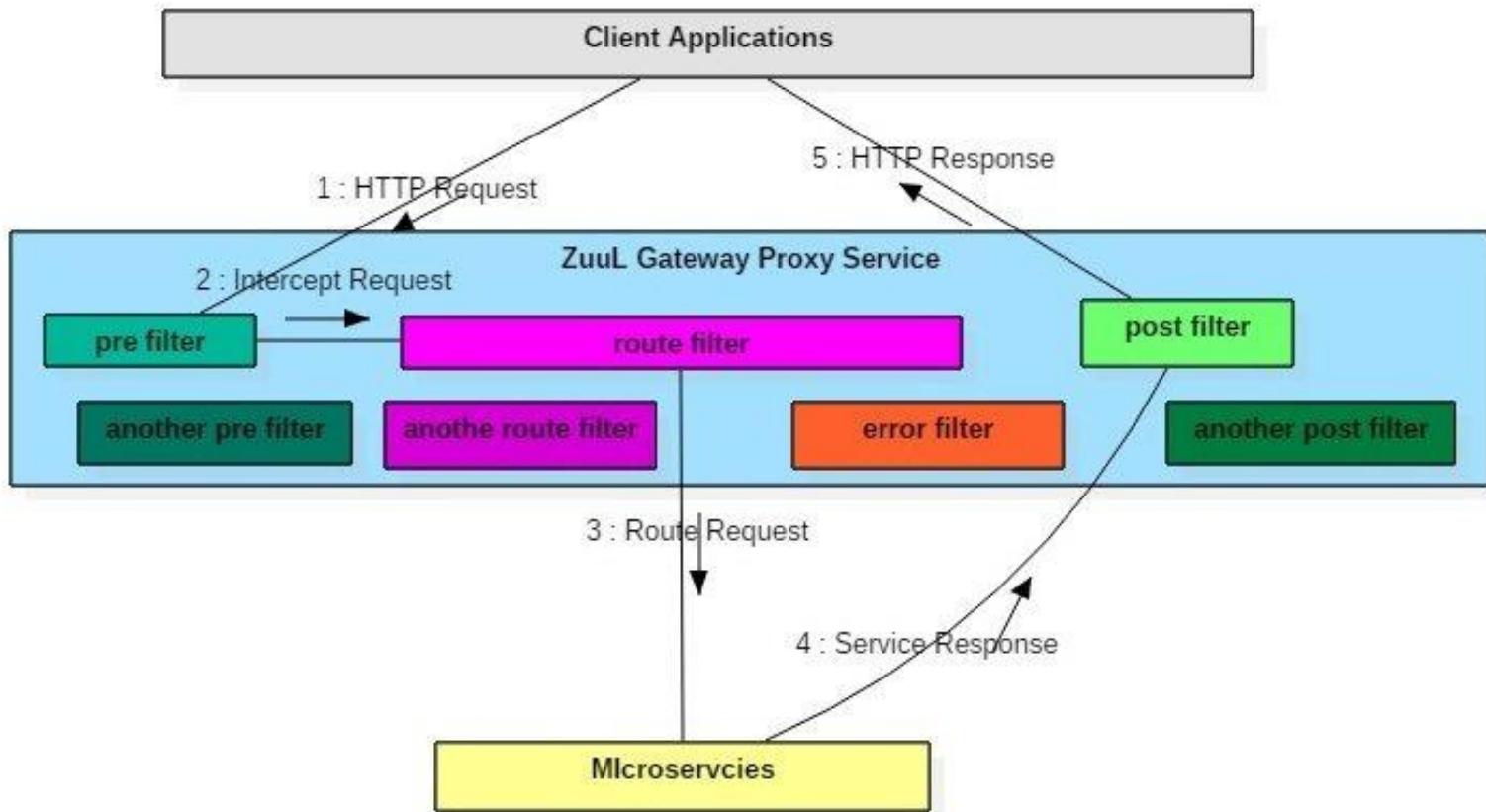
# Zuul Components

---



- **route filters** – are used to route the request.
- **error filters** – are invoked when an error occurs while handling the request.
-

# Zuul Components



# Zuul Filters Responsibilities



- Apply **microservice authentication and security** in the gateway layer to protect the actual services
- We can do **microservices insights and monitoring** of all the traffic that are going in to the ecosystem by enabling some logging to get meaningful data and statistics at the edge in order to give us an accurate view of production.

# Zuul Filters Responsibilities



- **Dynamic Routing** can route requests to different backend clusters as needed.
- We can do **runtime stress testing** by gradually increasing the traffic to a new cluster in order to gauge performance in many scenarios e.g. cluster has new H/W and network setup or that has new version of production code deployed.

# Zuul Filters Responsibilities



- We can do **dynamic load shedding** i.e. allocating capacity for each type of request and dropping requests that go over the limit.
- We can apply **static response handling** i.e. building some responses directly at the edge instead of forwarding them to an internal cluster for processing.

# Zuul Filters Responsibilities



- Authentication and Security — identifying authentication requirements for each resource and rejecting requests that do not satisfy them.
- Insights and Monitoring — tracking meaningful data and statistics at the edge in order to give us an accurate view of production.
- Dynamic Routing — dynamically routing requests to different backend clusters as needed.
- Stress Testing — gradually increasing the traffic to a cluster in order to gauge performance.

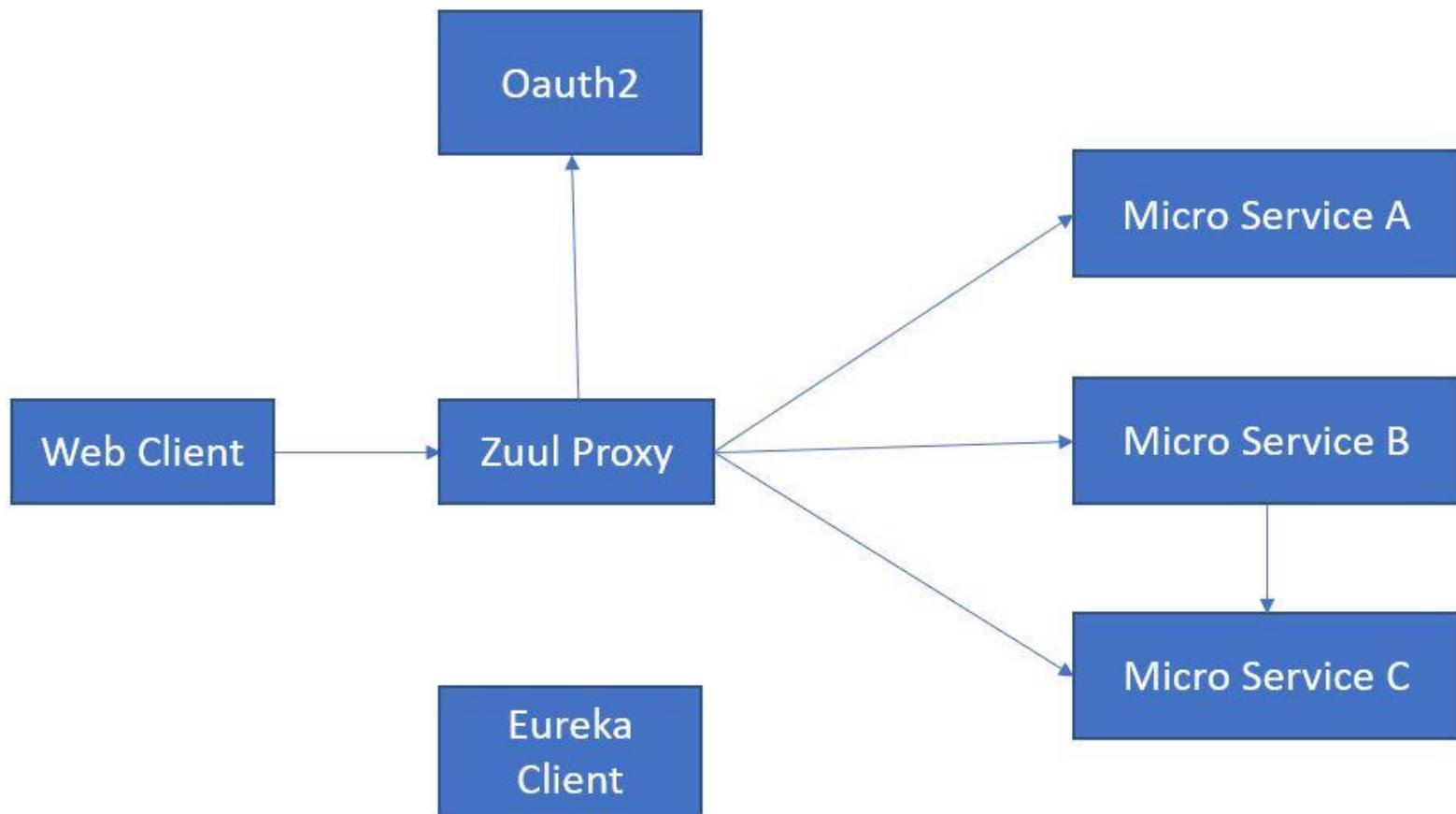
# Zuul Filters Responsibilities



- Load Shedding — allocating capacity for each type of request and dropping requests that go over the limit.
- Static Response handling — building some responses directly at the edge instead of forwarding them to an internal cluster
- Multiregion Resiliency — routing requests across AWS regions in order to diversify our ELB usage and move our edge closer to our members



# Zuul Authentication and Authorization





# What Is Redis?

---

- Redis is an open-source (BSD licensed), in-memory data structure store, used as a database, cache, and message broker.
- It supports data structures such as string, hashes, lists, sets, sorted sets with range queries, bitmaps, hyper logs, and geospatial indexes with radius queries.

# Why Redis?



- Redis is basically used for cache management.
- It reduces the client workload and speeds up the application.



# Calling an External Application API

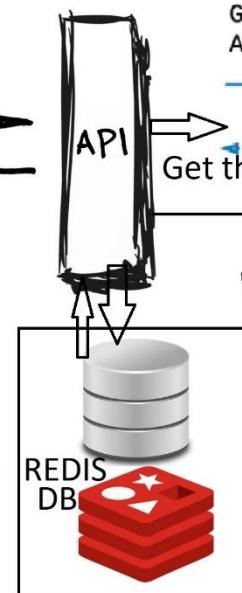
Web app  
in  
Browser



Request  
Response

First check response is present in  
redis if yes , will return from here  
else call to web service and put  
the response in redis so next  
time response will be return from  
redis.

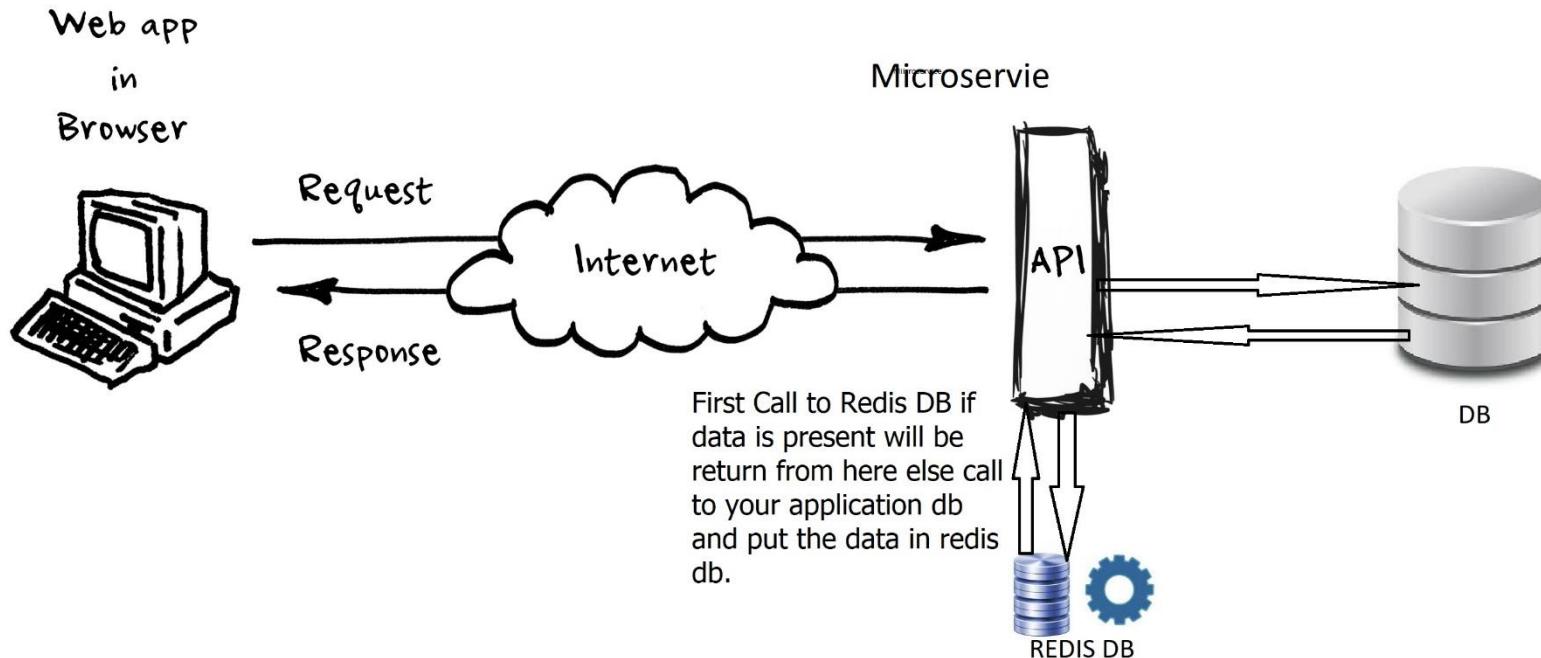
Microservie



External Application



# Frequently Querying the Reference Table or Master Table in the Database





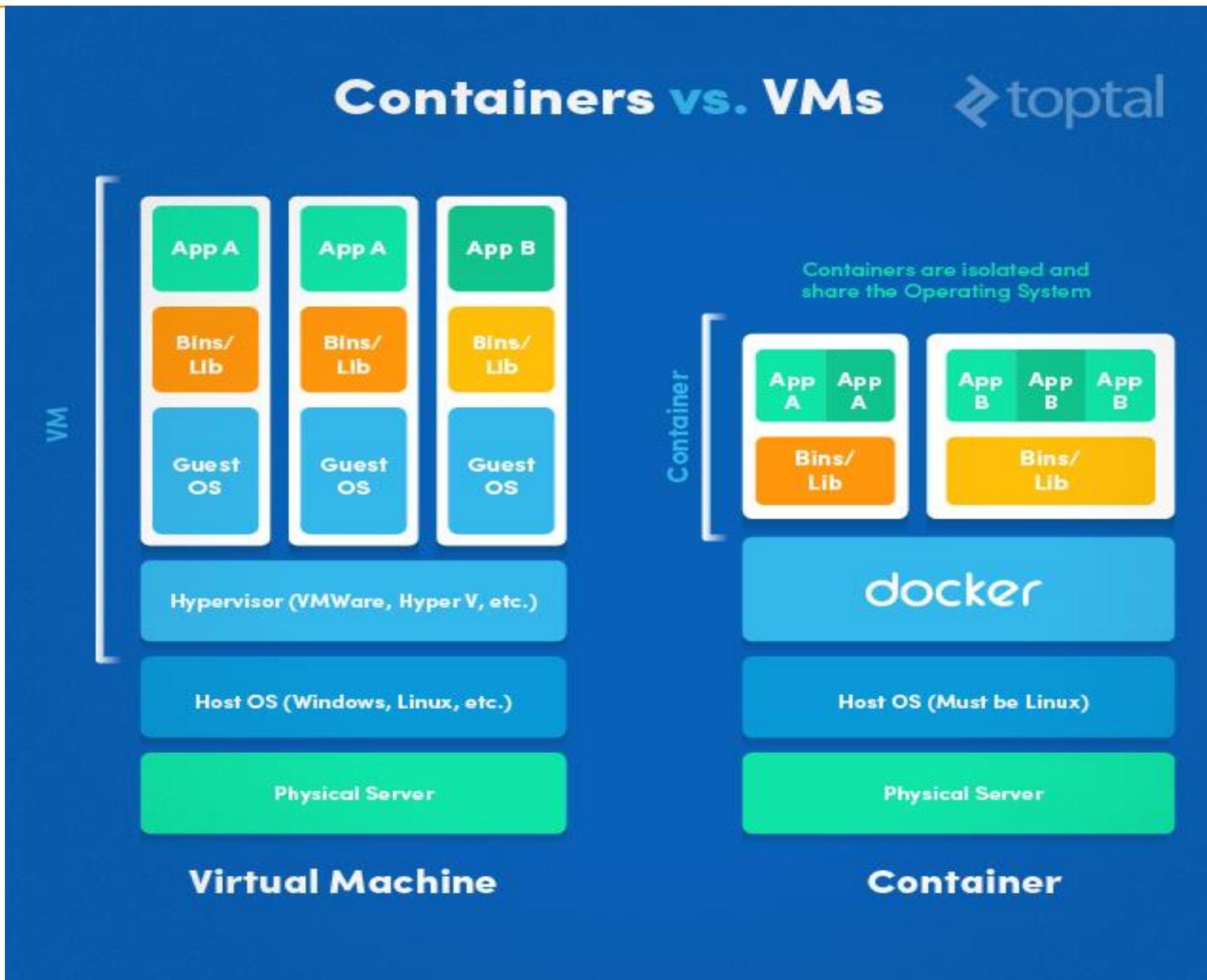
- Docker is an excellent tool for managing and deploying microservices.
- Each microservice can be further broken down into processes running in separate Docker containers, which can be specified with Dockerfiles and Docker Compose configuration files.
- Combined with a provisioning tool such as Kubernetes, each microservice can then be easily deployed, scaled, and collaborated on by a developer team.
- Specifying an environment in this way also makes it easy to link microservices together to form a larger application.

# Docker



## Containers vs. VMs

toptal



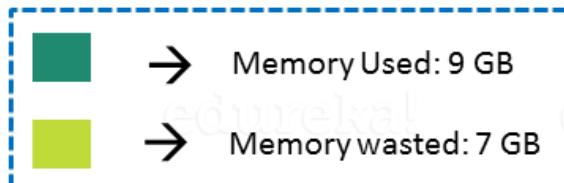
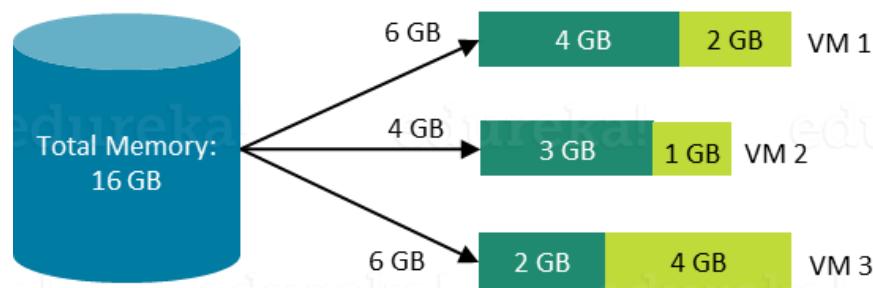


# Hypervisor

---

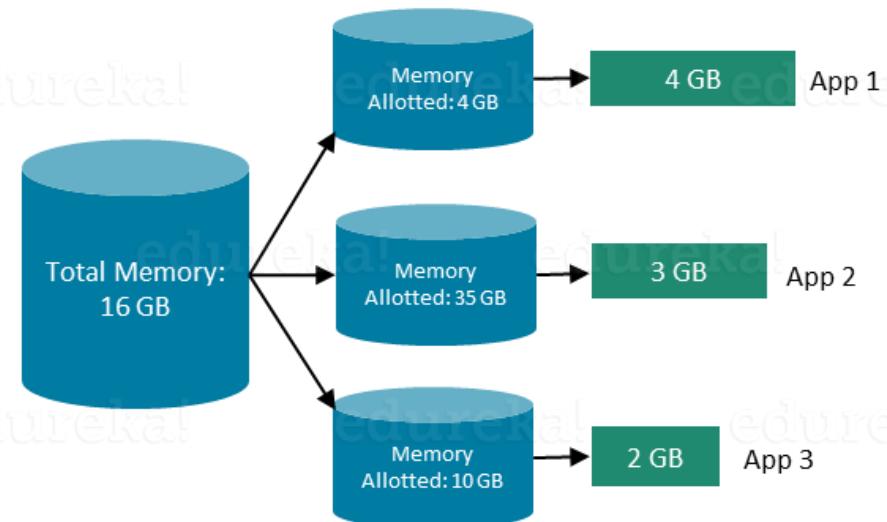
- A hypervisor is a software that makes virtualization possible.
- It is also called Virtual Machine Monitor.
- It divides the host system and allocates the resources to each divided virtual environment.
- You can basically have multiple OS on a single host system.
- There are two types of Hypervisors:
- **Type 1**: It's also called Native Hypervisor or Bare metal Hypervisor.
  - It runs directly on the underlying host system.
  - It has direct access to your host's system hardware and hence does not require a base server operating system.
- **Type 2**: This kind of hypervisor makes use of the underlying host operating system. It's also called Hosted Hypervisor.

## In case of Virtual Machines



7 Gb of Memory is blocked and cannot be allotted to a new VM

## In case of Docker



Only 9 GB memory utilized;  
 7 GB can be allotted to a new Container



# What is Docker

---

- What is Docker?
- At its heart, Docker is software which lets you create an *image* and then run instances of that image in a *container*.
- Docker maintains a vast repository of images, called the Docker Hub which you can use as starting points or as free storage for your own images.
- You can install Docker, choose an image you'd like to use, then run an instance of it in a container.



# Difference between VM and Container

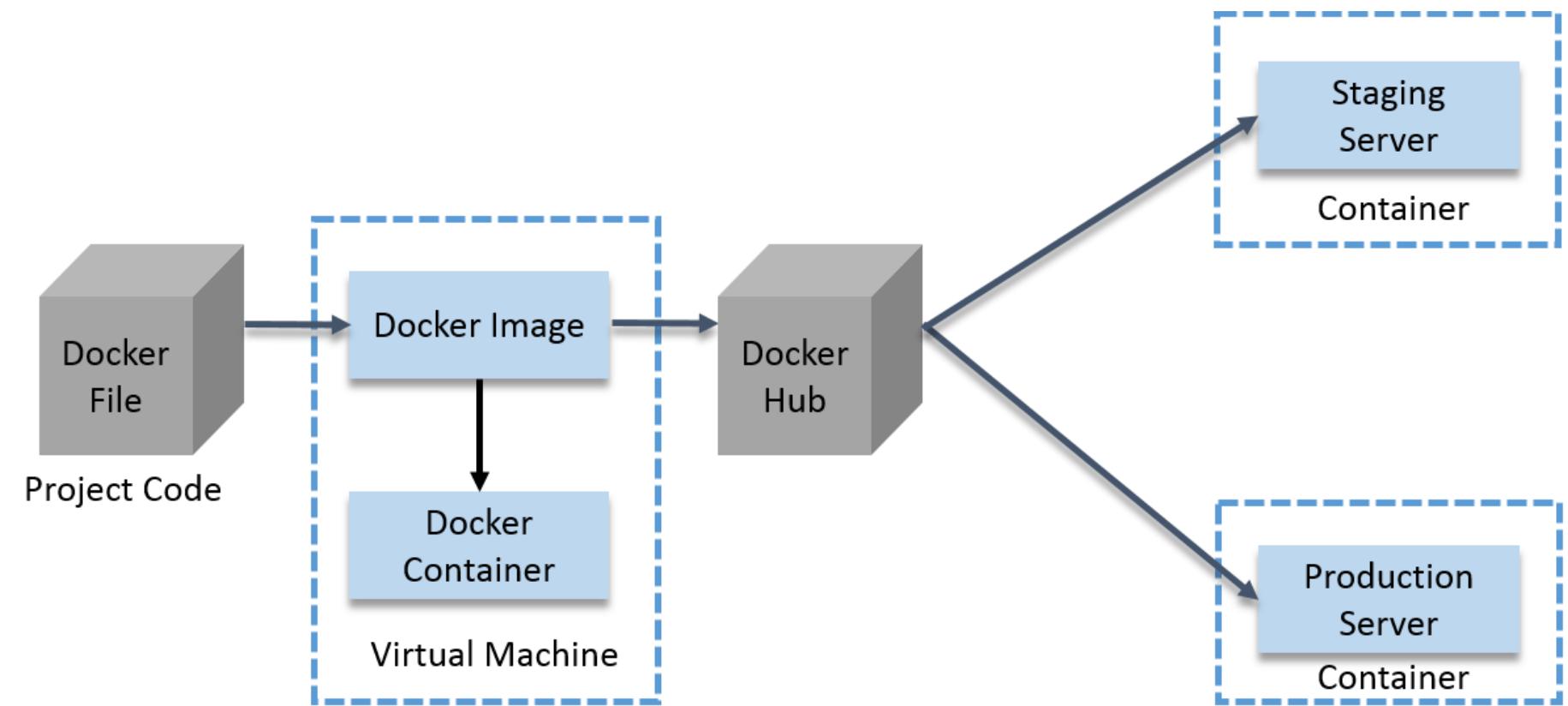
- Applications running in virtual machines, apart from the hypervisor, require a full instance of the operating system and any supporting libraries.
- Containers, on the other hand, share the operating system with the host.
- Hypervisor is comparable to the container engine (represented as Docker on the image) in a sense that it manages the lifecycle of the containers.
- The important difference is that the processes running inside the containers are just like the native processes on the host, and do not introduce any overheads associated with hypervisor execution.
- Additionally, applications can reuse the libraries and share the data between containers.

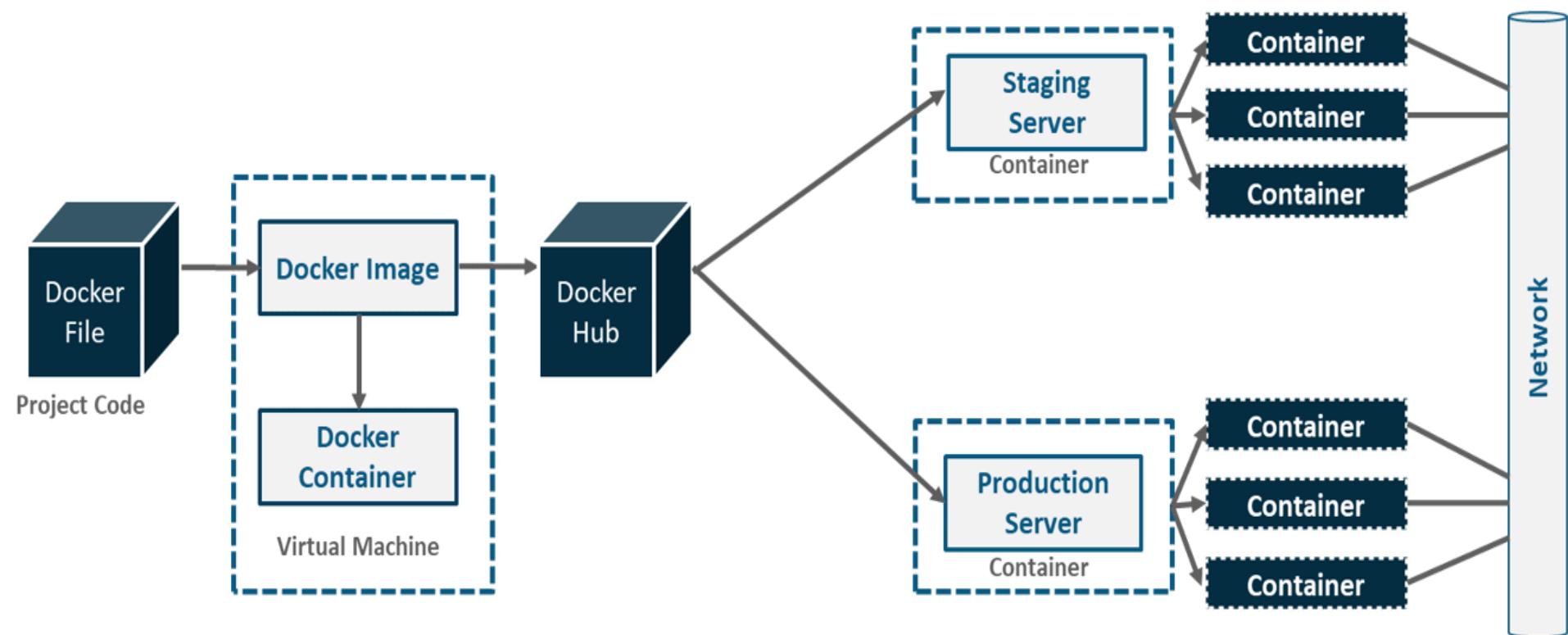
# Dockerfile, Docker Image And Docker Container



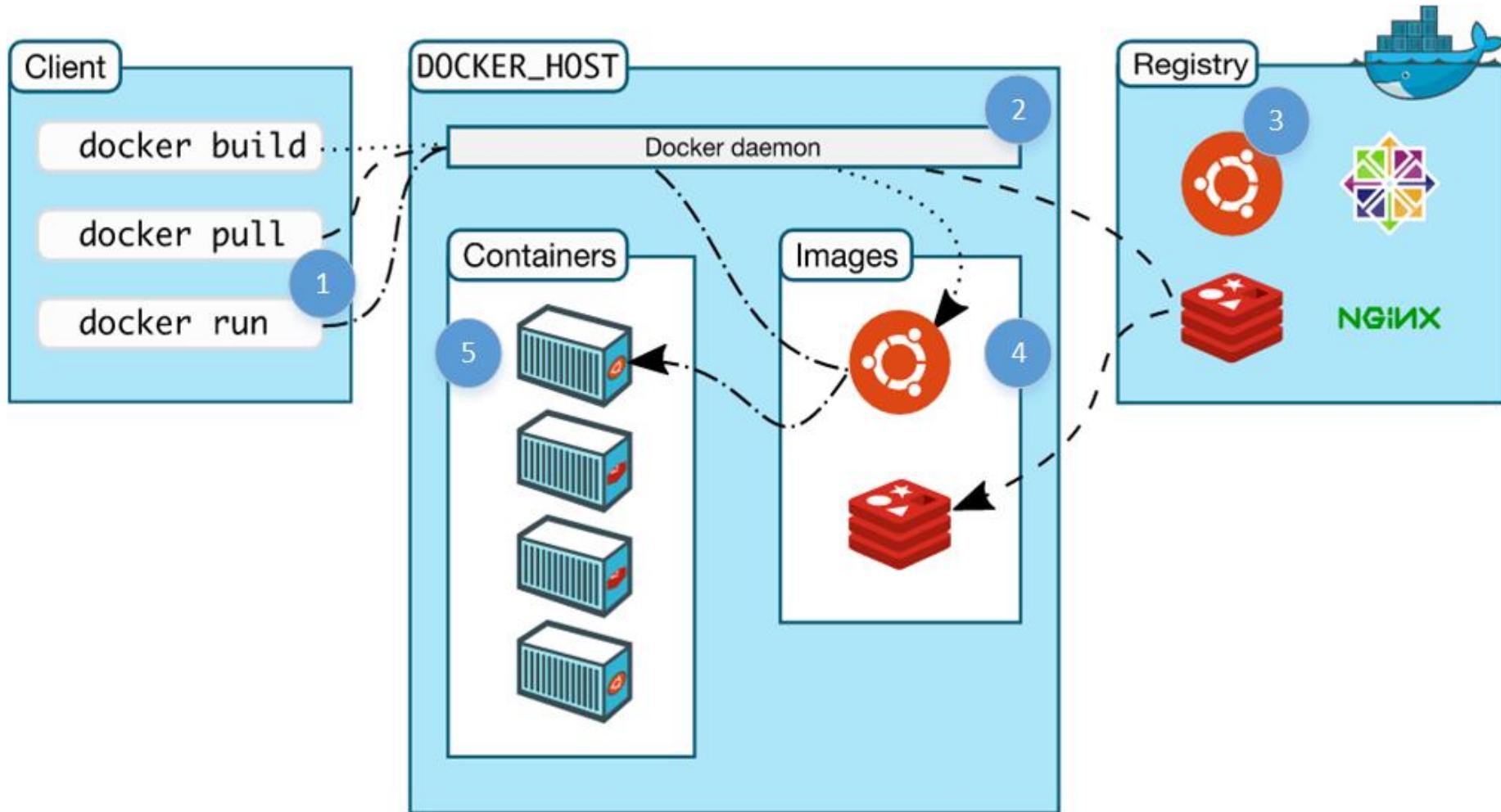
- A Docker Image is created by the sequence of commands written in a file called as Dockerfile.
- When this Dockerfile is executed using a docker command it results into a Docker Image with a name.
- When this Image is executed by “docker run” command it will by itself start whatever application or service it must start on its execution.





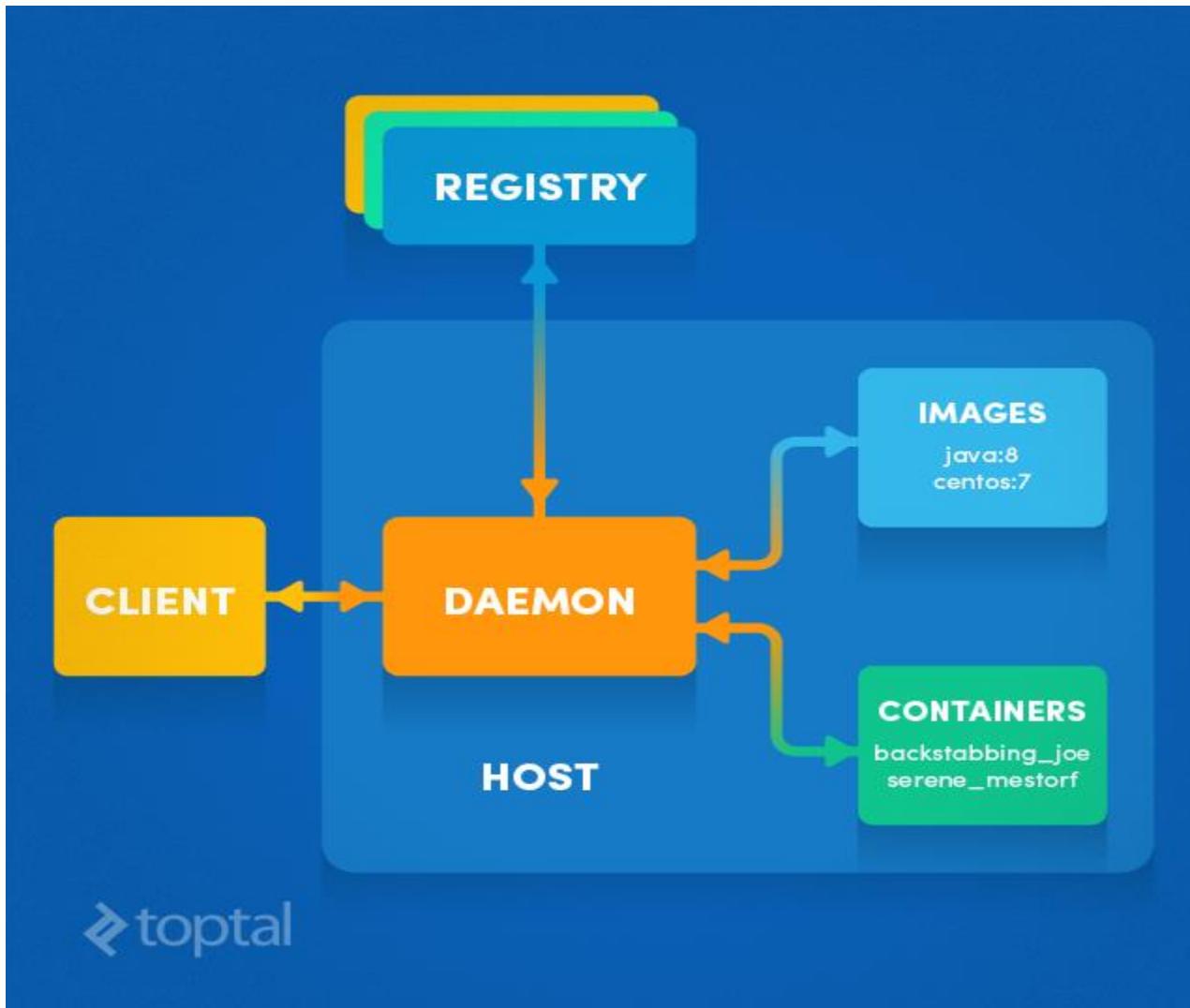


# Docker





# Docker Architecture

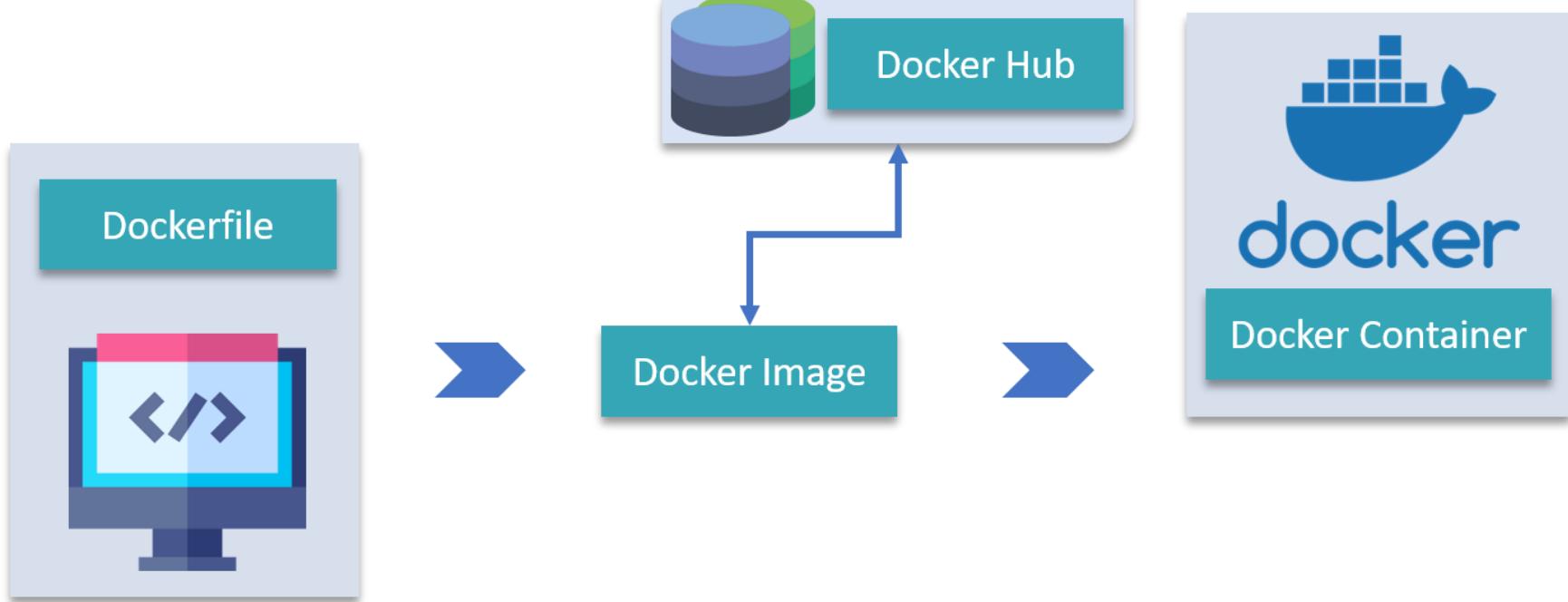


## Docker Hub:



- Docker Hub is like GitHub for Docker Images. It is basically a cloud registry where you can find Docker Images uploaded by different communities, also you can develop your own image and upload on Docker Hub, but first, you need to create an account on DockerHub.

# Docker Hub





# Docker Architecture

---

- It consists of a Docker Engine which is a client-server application with three major components:
- A server which is a type of long-running program called a daemon process (the docker command).
- A REST API which specifies interfaces that programs can use to talk to the daemon and instruct it what to do.
- A command line interface (CLI) client (the docker command).
- The CLI uses the Docker REST API to control or interact with the Docker daemon through scripting or direct CLI commands. Many other Docker applications use the underlying API and CLI.



# Docker Architecture

---

- By default, the main registry is the Docker Hub which hosts public and official images.
- Organizations can also host their private registries if they desire.
- Images can be downloaded from registries explicitly (`docker pull imageName`) or implicitly when starting a container. Once the image is downloaded it is cached locally.
- Containers are the instances of images - they are the living thing. There could be multiple containers running based on the same image.



# Docker Architecture

---

- At the center, there is the Docker daemon responsible for creating, running, and monitoring containers.
- It also takes care of building and storing images.
- Finally, on the left-hand side there is a Docker client. It talks to the daemon via HTTP.
- Unix sockets are used when on the same machine, but remote management is possible via HTTP based API.



# Goals of Docker Networking





## Creating Test Database Server

---

- This is a great Docker use case.
- We might not want to run our production database in Docker (perhaps we'll just use Amazon RDS for example), but we can spin up a clean MySQL database in no time as a Docker container for development - leaving our development machine clean and keeping everything we do controlled and repeatable.



# Docker Compose

---

- Docker Compose is basically used to run multiple Docker Containers as a single server. Let me give you an example:
- Suppose if I have an application which requires WordPress, Maria DB and PHP MyAdmin. I can create one file which would start both the containers as a service without the need to start each one separately. It is really useful especially if you have a microservice architecture.



# Docker Container

Column	Description
Container ID	The unique ID of the container. It is a SHA-256.
Image	The name of the container image from which this container is instantiated.
Status	The status of the container (created, restarting, running, removing, paused, exited, or dead).
Ports	The list of container ports that have been mapped to the host.
Names	The name assigned to this container (multiple names are possible).



# Docker Client and Docker Engine

---

- **Docker Client** : This is the utility we use when we run any docker commands e.g. docker run (docker container run) , docker images , docker ps etc. It allows us to run these commands which a human can easily understand.
- **Docker Daemon/Engine**: This is the part which does rest of the magic and knows how to talk to the kernel, makes the system calls to create, operate and manage containers, which we as users of docker dont have to worry about.



# Creating Test Database Server

- docker run --name olddb -e MYSQL\_ROOT\_PASSWORD=vignesh -e MYSQL\_DATABASE=virtusa\_2018db -e MYSQL\_USER=root -p 3306:3306 mysql/mysql-server:5.5
- docker run tells the engine we want to run an image (the image comes at the end, mysql:vlatest)
- --name db names this container db.
- -d detach - i.e. run the container in the background.
- -e MYSQL\_ROOT\_PASSWORD=123 the -e is the flag that tells docker we want to provide an environment variable. The variable following it is what the MySQL image checks for setting the default root password.
- -p 3306:3306 tells the engine that we want to map the port 3306 from inside the container to out port 3306.



MINGW64/d/Program Files/Docker Toolbox

```
Warning: Unable to load '/usr/share/zoneinfo/iso3166.tab' as time zone. Skipping it.
Warning: Unable to load '/usr/share/zoneinfo/leapseconds' as time zone. Skipping it.
Warning: Unable to load '/usr/share/zoneinfo/tzdata.zi' as time zone. Skipping it.
Warning: Unable to load '/usr/share/zoneinfo/zone.tab' as time zone. Skipping it.
Warning: Unable to load '/usr/share/zoneinfo/zone1970.tab' as time zone. Skipping it.
[Entrypoint] Not creating mysql user. MYSQL_USER and MYSQL_PASSWORD must be specified to create a mysql user.
[Entrypoint] ignoring /docker-entrypoint-initdb.d/*
[Entrypoint] Server shut down
[Entrypoint] MySQL init process done. Ready for start up.
[Entrypoint] Starting MySQL 5.5.62-1.1.8
181120 19:49:03 [Note] mysqld (mysqld 5.5.62) starting as process 1 ...
[jruby-9.1.1] Balasubramaniam@DESKTOP-55AGI0I MINGW64 /d/Program Files/Docker Toolbox
[kibana-6.3.0] docker kill $(docker ps -q)
304981f38447
[logstash-6.3.0] Balasubramaniam@DESKTOP-55AGI0I MINGW64 /d/Program Files/Docker Toolbox
[metricbeat-6.3.0] docker kill $(docker ps -a -q)
Error response from daemon: Cannot kill container: 304981f38447: Container 304981f384473c7caf3235c2fa998da2f0240efff01a1ed0d00381c25180b1dd is not running
[microservices-0.0.1-SNAPSHOT] Balasubramaniam@DESKTOP-55AGI0I MINGW64 /d/Program Files/Docker Toolbox
[nssm-2.24] docker rm $(docker ps -a -q)
304981f38447
[OpenSSL-1.0.2] Balasubramaniam@DESKTOP-55AGI0I MINGW64 /d/Program Files/Docker Toolbox
[Oracle] Balasubramaniam@DESKTOP-55AGI0I MINGW64 /d/Program Files/Docker Toolbox
[Pega Robot] Balasubramaniam@DESKTOP-55AGI0I MINGW64 /d/Program Files/Docker Toolbox
[PostgreSQL]
[Program Files]
[Program Files (x86)]
[Python]
[Ruby25-x64]
[scope]
[SQLExpr_x86_ENU]
[SQLServer2017Media]
[temp]
[winlogbeat-6.3.0-windows-x86_64]
[wwwroot]
[auditbeat-6.3.0-windows-x86_64.zip]
```

3 items

Type here to search

File Home

DATA (D): alfresco-community cygwin64 docker ElasticSearch filebeat-6.3.0 ftpsite heroku-apt jruby-9.1.1 kibana-6.3.0 logstash-6.3.0 material-ui metricbeat microservices nssm-2.24 OpenSSL-1.0.2 Oracle Pega Robot PostgreSQL Program Files Program Files (x86) Python Ruby25-x64 scope SQLExpr\_x86\_ENU SQLServer2017Media temp winlogbeat-6.3.0-windows-x86\_64 wwwroot auditbeat-6.3.0-windows-x86\_64.zip

Search resources



# Creating Test Database Server

- Docker ps
- docker exec -it olddb /bin/bash
- mysql –u root –p

A screenshot of a terminal window titled "MINGW64/d/Program Files/Docker Toolbox". The window displays a MySQL command-line interface. It shows the creation of a new database named "virtual\_2018db" and a subsequent "show databases" command that lists all databases, including the newly created one.

```
MINGW64/d/Program Files/Docker Toolbox
Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| mysql |
| performance_schema |
| sys |
+-----+
4 rows in set (0.02 sec)

mysql> create database virtual_2018db;
Query OK, 1 row affected (0.10 sec)

mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| mysql |
| performance_schema |
| sys |
| virtual_2018db |
+-----+
```



# Springboot Docker

---

- <plugin>
- <groupId>com.spotify</groupId>
- <artifactId>docker-maven-plugin</artifactId>
- <version>VERSION GOES HERE</version>
- <configuration>
- <imageName>example</imageName>
- <baseImage>java</baseImage>
- <entryPoint>["java", "-jar", "\${project.build.finalName}.jar"]</entryPoint>
- <!-- copy the service's jar file from target into the root directory of the image -->
- <resources>
- <resource>
- <targetPath>/</targetPath>
- <directory>\${project.build.directory}</directory>
- <include>\${project.build.finalName}.jar</include>
- </resource>
- </resources>
- </configuration>
- </plugin>



# Springboot Docker

---

- Spring boot Project Build
- Goal package docker:build
- mvn clean install dockerfile:build
- Docker folder created and docker image name example deployed in docker machine
- Check docker images
- Docker ps



# Springboot Docker

- Docker run -t --name sampleapp --link v1db -p 8080:8080 example:latest
- Example:latest --- image name
- --name -- container reference
- --link -- db ref in container

```
docker run -h 192.168.99.100 -p 7070:7070 -t my-repo/example --name my-repo-image:latest
```



# Docker Volume

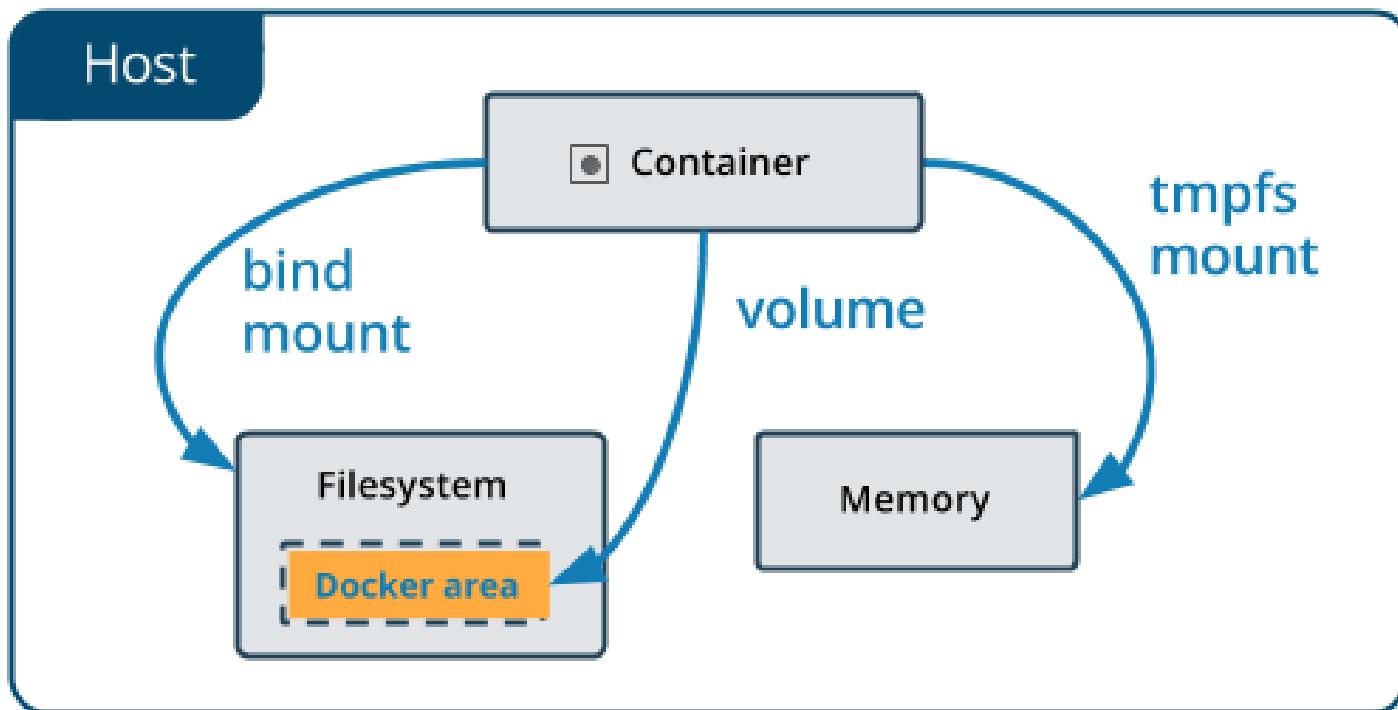
---

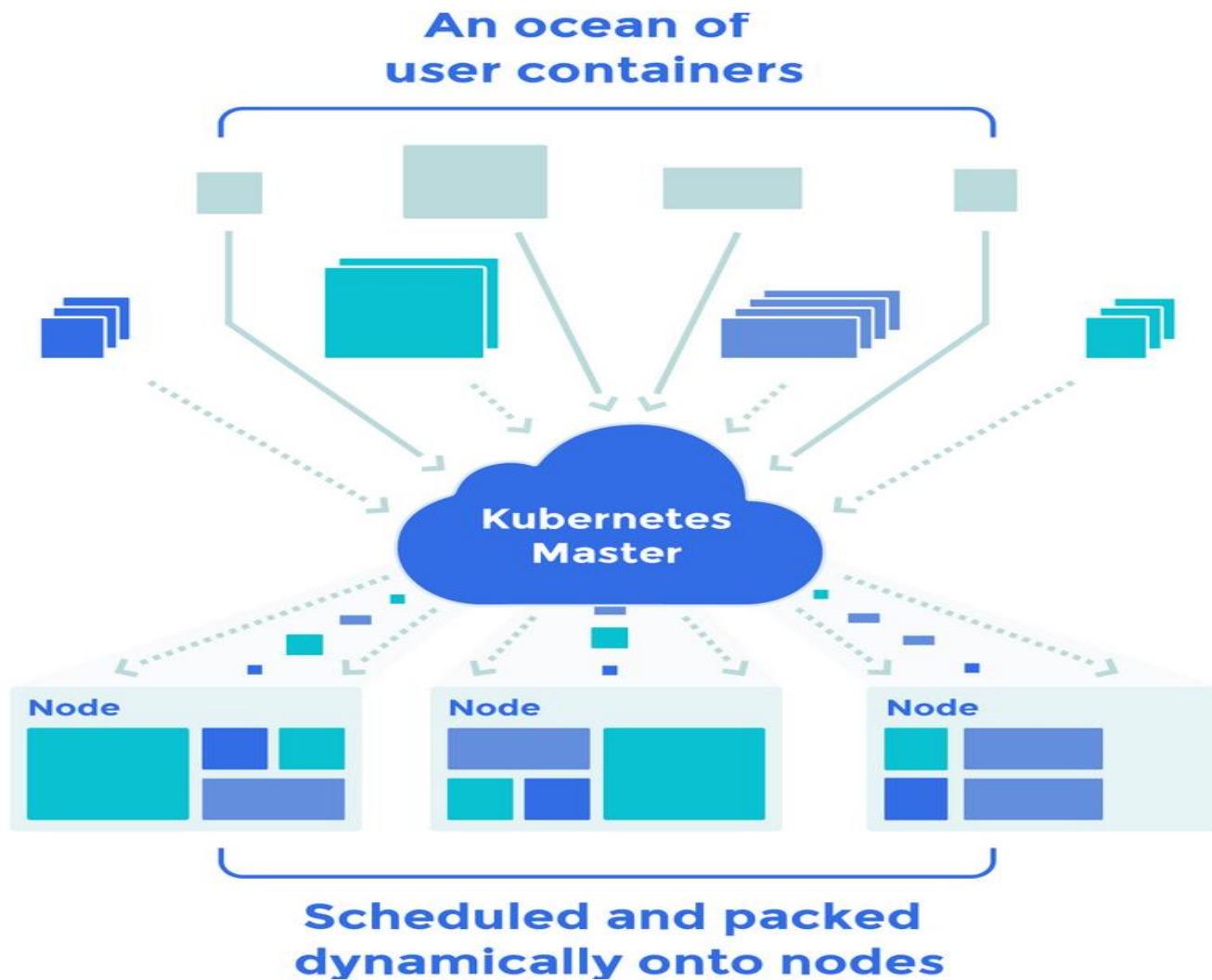
- Volumes are the preferred mechanism for persisting data generated by and used by Docker containers.
- While bind mounts are dependent on the directory structure of the host machine, volumes are completely managed by Docker.



# Docker Volume

- Volumes have several advantages over bind mounts:
  - Volumes are easier to back up or migrate than bind mounts.
  - You can manage volumes using Docker CLI commands or the Docker API.
  - Volumes work on both Linux and Windows containers.
  - Volumes can be more safely shared among multiple containers.
  - Volume drivers let you store volumes on remote hosts or cloud providers, to encrypt the contents of volumes, or to add other functionality.
  - New volumes can have their content pre-populated by a container

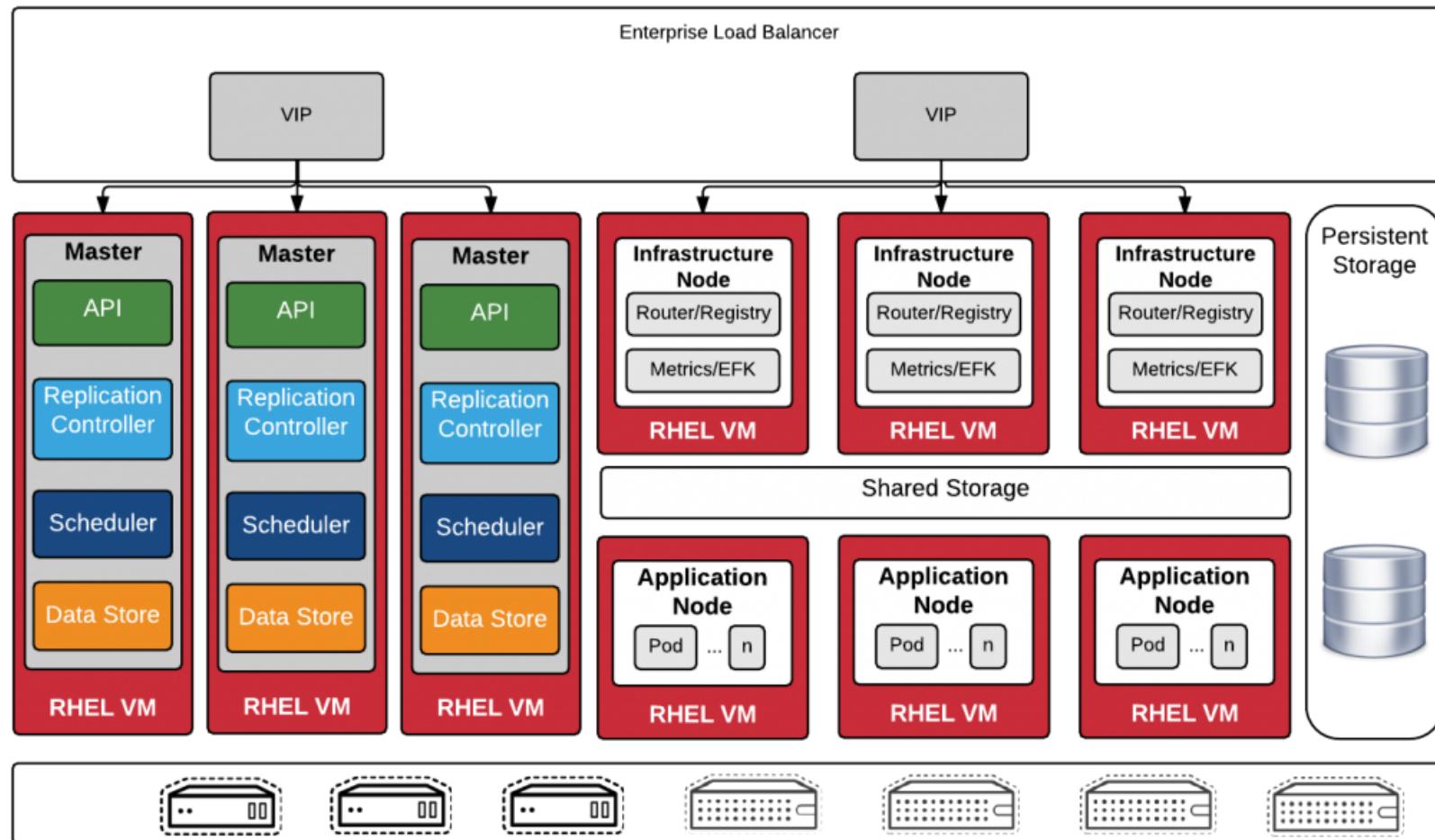




# Openshift

Openshift is built on top of kubernetes. Openshift project is maintained by Redhat. It has both open source (openshift origin) and enterprise version (openshift container platform).

Along with core Kubernetes features, it offers out of the box components for container management and orchestration.





# Docker Swarm

---

- Docker Swarm is Docker's native feature to support clustering of Docker machines.
- This enables multiple machines running Docker Engine to participate in a cluster, called Swarm.
- The Docker engines contributing to a Swarm are said to be running in Swarm mode.
- Machines enter into the Swarm mode by either initializing a new swarm or by joining an existing swarm.
- To the end user the swarm would seem like a single machine.
- A Docker engine participating in a swarm is called a node



# Docker Swarm

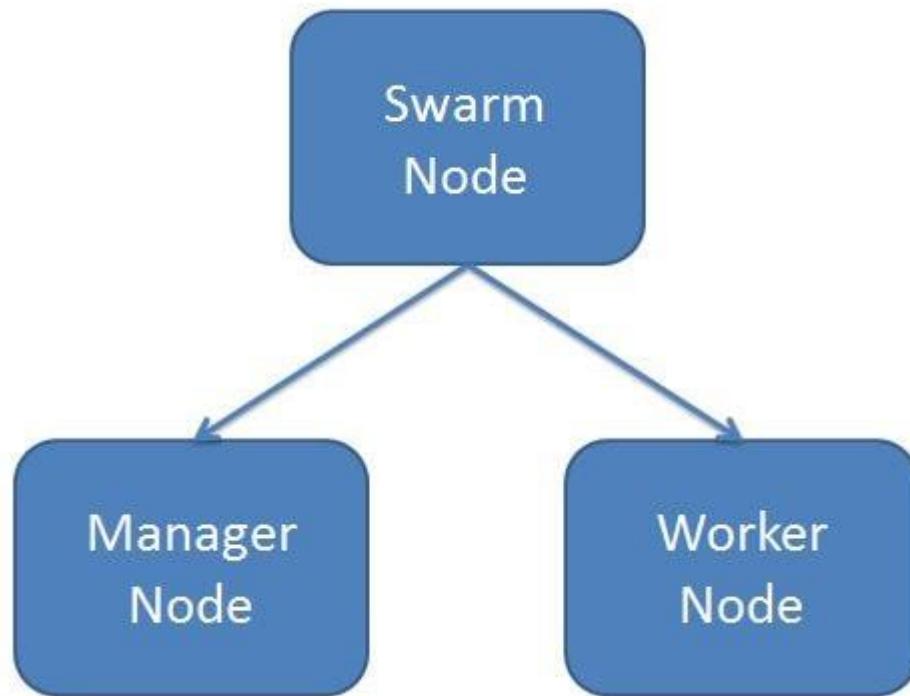
---

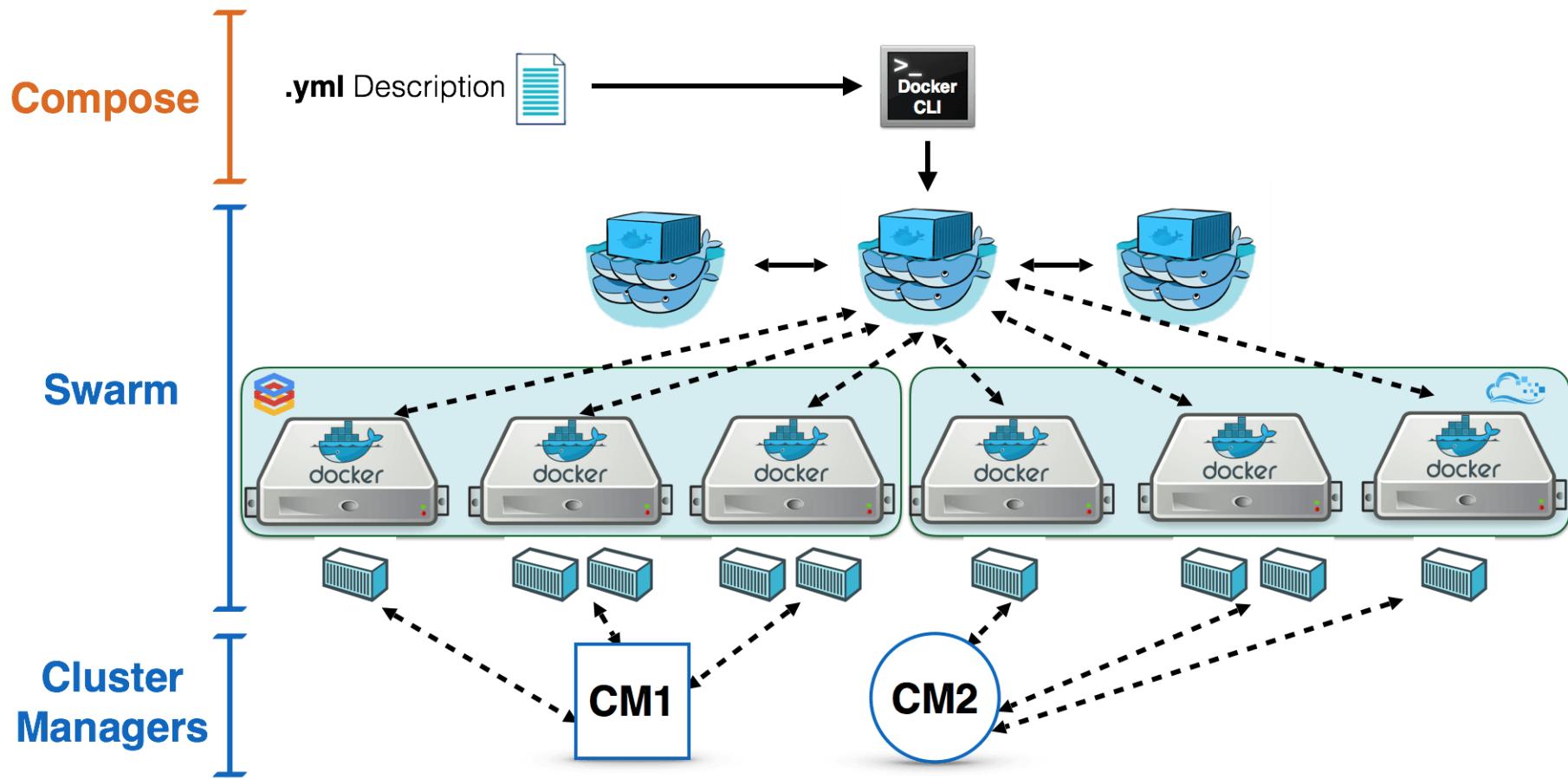
- A node can either be a manager node or a worker node.
- The manager node performs cluster management and orchestration while the worker nodes perform tasks allocated by the manager.
- A manager node itself, unless configured otherwise, is also a worker node.
- The central entity in the Docker Swarm infrastructure is called a service.
- A Docker swarm executes services. The user submits a service to the manager node to deploy and execute.
- A service is made up of many tasks. A task is the most basic work unit in a Swarm.
- A task is allocated to each worker node by the manager node.

# Docker Swarm



- The Docker ecosystem consists of tools from development to production deployment frameworks.
- In that list, docker swarm fits into cluster management.
- A mix of docker-compose, swarm, overlay network and a good service discovery tool such as etcd or consul can be used for managing a cluster of Docker containers.
- Docker swarm is still maturing in terms of functionalities when compared to other open-source container cluster management tools.
- Considering the vast docker contributors, it won't be so long for docker swarm to have all the best functionalities other tools possess.
- Docker has documented a good production plan for using docker swarm in production.







```
Balasubramaniam@DESKTOP-55AGI0I MINGW64 /d/Program Files/Docker Toolbox
$ docker service create --name eurekaswarm -p 8761:8761 eureka-app:latest
image eureka-app:latest could not be accessed on a registry to record
its digest. Each node will access eureka-app:latest independently,
possibly leading to different nodes running different
versions of the image.
```

```
0uw58z9g5vbjl13zzjop10k9
overall progress: 1 out of 1 tasks
1/1: running  [=====>]
verify: Service converged
```

```
Balasubramaniam@DESKTOP-55AGI0I MINGW64 /d/Program Files/Docker Toolbox
$ docker service ls
```

ID	NAME	MODE	REPLICAS	IMAGE	PORTS
0uw58z9g5vbj	eurekaswarm	replicated	1/1	eureka-app:latest	*:8761->8761/tcp

```
Balasubramaniam@DESKTOP-55AGI0I MINGW64 /d/Program Files/Docker Toolbox
$ docker service scale eurekaswarm=3
eurekaswarm scaled to 3
overall progress: 3 out of 3 tasks
1/3: running  [=====>]
2/3: running  [=====>]
3/3: running  [=====>]
verify: Service converged
```

```
Balasubramaniam@DESKTOP-55AGI0I MINGW64 /d/Program Files/Docker Toolbox
$ docker service ls
```

ID	NAME	MODE	REPLICAS	IMAGE	PORTS
0uw58z9g5vbj	eurekaswarm	replicated	3/3	eureka-app:latest	*:8761->8761/tcp

```
Balasubramaniam@DESKTOP-55AGI0I MINGW64 /d/Program Files/Docker Toolbox
```

# What Is Kubernetes? An Introduction To Container Orchestration Tool



- **What Is Kubernetes?**
- Kubernetes is an open-source container management (orchestration) tool. Its container management responsibilities include container deployment, scaling & descaling of containers & container load balancing.
- Download from
- <https://github.com/kubernetes/minikube>
- Under curl - windows
- <https://kubernetes.io/docs/tasks/tools/install-kubectl/#install-kubectl>



# What is Kubernetes



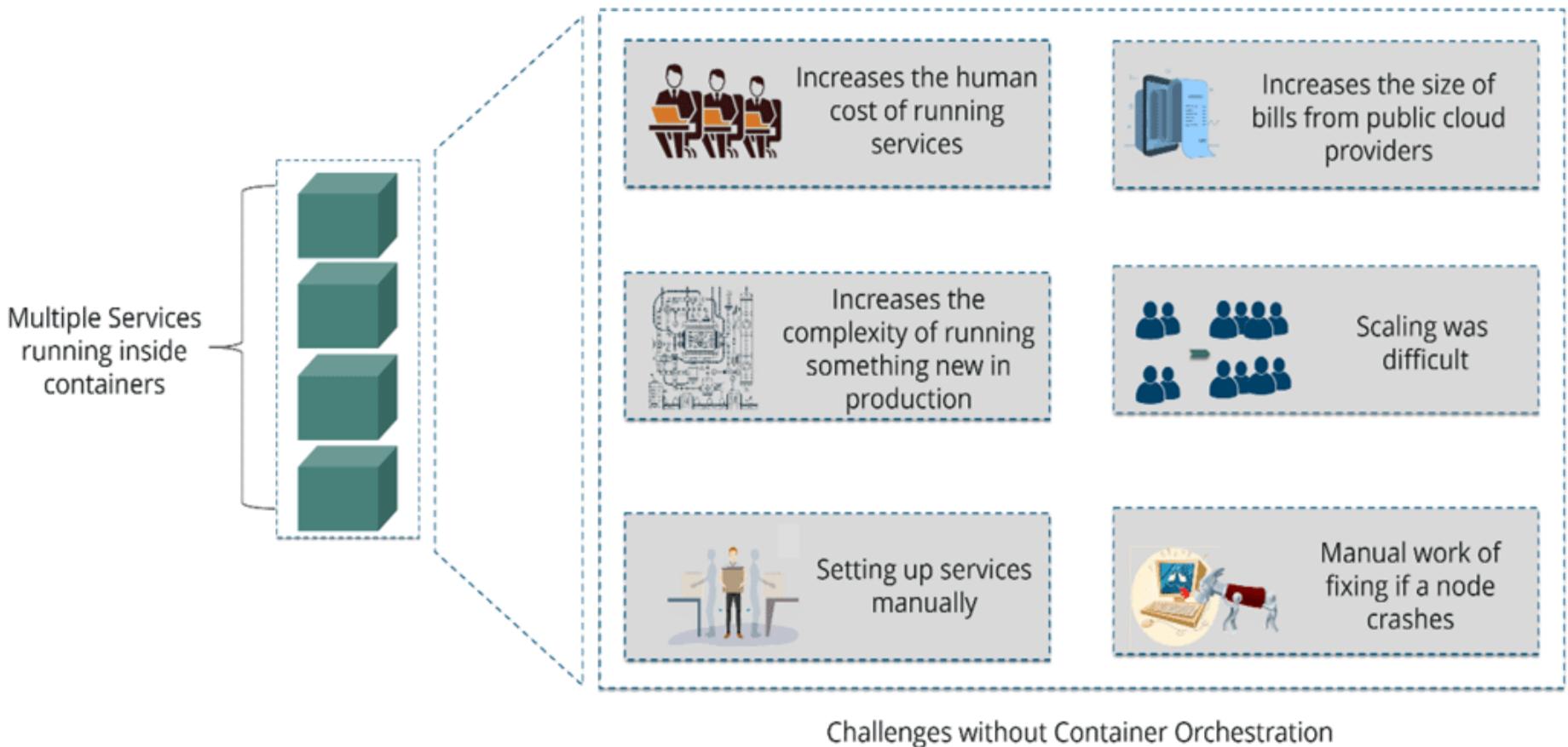
# What Is Kubernetes? An Introduction To Container Orchestration Tool

---



- **Why Use Kubernetes?**
- Companies out there maybe using Docker or Rocket or maybe simply Linux containers for containerizing their applications. But, whatever it is, they use it on a massive scale. They don't stop at using 1 or 2 containers in Prod. But rather, **10's or 100's** of containers for load balancing the traffic and ensuring high availability.
- .

# What is the need for Container Orchestration?



# Kubernetes vs Docker Swarm



Features	Kubernetes	Docker Swarm
<b>Installation &amp; Cluster Config</b>	Setup is very complicated, but once installed cluster is robust.	Installation is very simple, but the cluster is not robust.
<b>GUI</b>	GUI is the Kubernetes Dashboard.	There is no GUI.
<b>Scalability</b>	Highly scalable and scales fast.	Highly scalable and scales 5x faster than Kubernetes.
<b>Auto-scaling</b>	Kubernetes can do auto-scaling.	Docker swarm cannot do auto-scaling.
<b>Load Balancing</b>	Manual intervention needed for load balancing traffic between different containers and pods.	Docker swarm does auto load balancing of traffic between containers in the cluster.
<b>Rolling Updates &amp; Rollbacks</b>	Can deploy rolling updates and does automatic rollbacks.	Can deploy rolling updates, but not automatic rollback.
<b>DATA Volumes</b>	Can share storage volumes only with the other containers in the same pod.	Can share storage volumes with any other container.
<b>Logging &amp; Monitoring</b>	In-built tools for logging and monitoring.	3rd party tools like ELK stack should be used for logging and monitoring.

# What are the features of Kubernetes?



01

Automated Scheduling

Kubernetes provides advanced scheduler to launch container on cluster nodes

02

Self Healing Capabilities

Rescheduling, replacing and restarting the containers which are died.

03

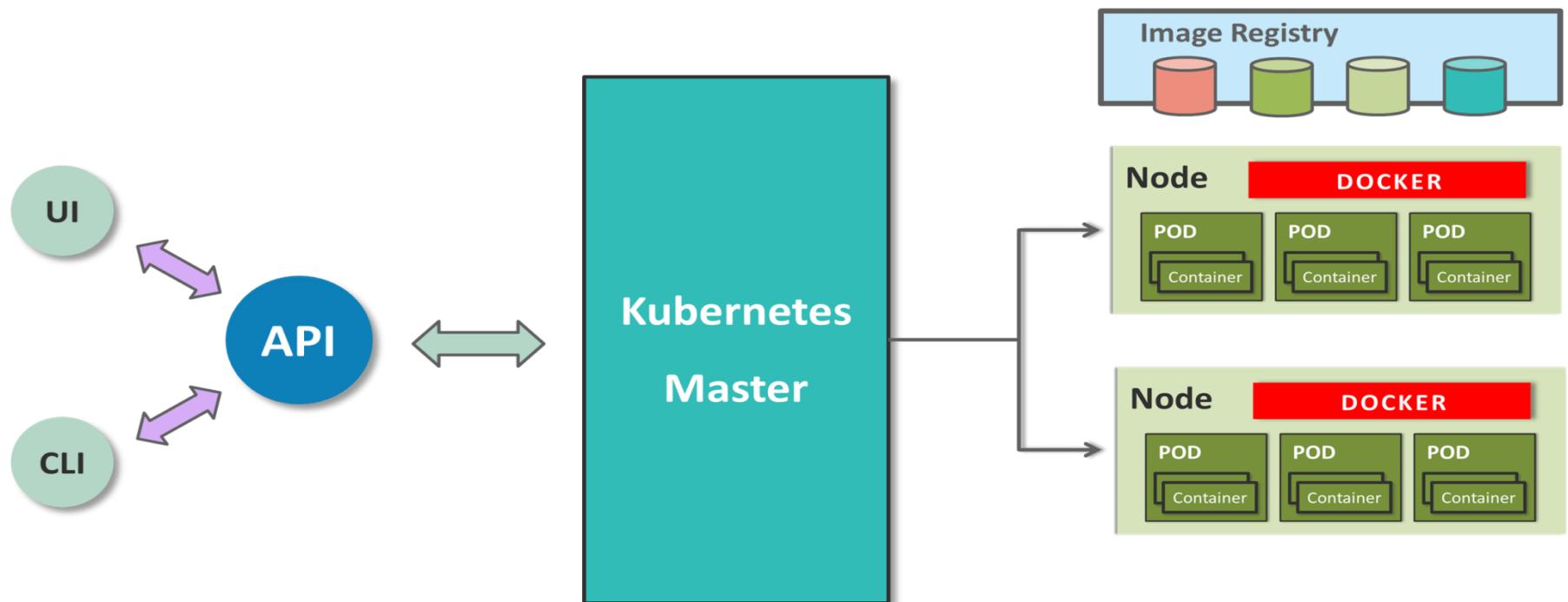
Automated rollouts and rollback

Kubernetes supports rollouts and rollbacks for the desired state of the containerized application

04

Horizontal Scaling and Load Balancing

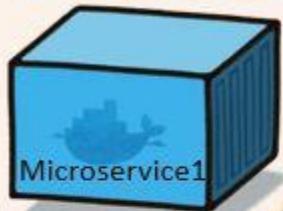
Kubernetes can scale up and scale down the application as per the requirements



Pod has group of containers that are run on the same host. So, if we regularly deploy single containers, then our container and the pod will be one and the same.



**POD1**



Microservice1

**POD2**



Microservice2

**POD3**

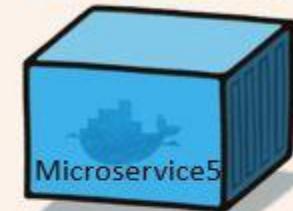


Microservice3



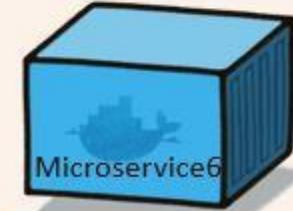
Microservice4

**POD4**



Microservice5

**POD5**



Microservice6

...



# Representation Of Kubernetes Cluster

**App.yaml**

**Deployment**

Pod1

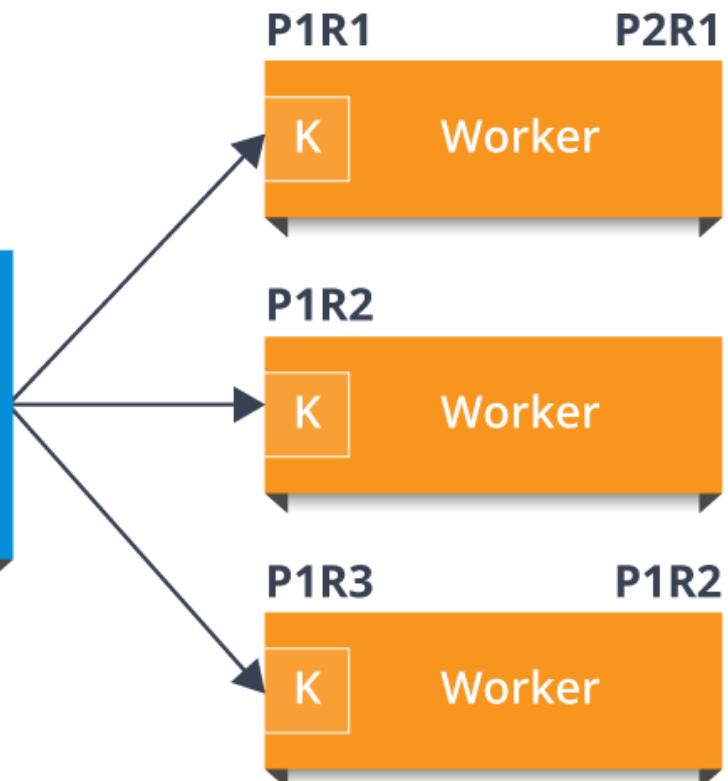
Container Image 1 Container Image 2

Replicas = 3

Pod2

Container Image 3

Replicas = 2



**P - Pod R - Replica**

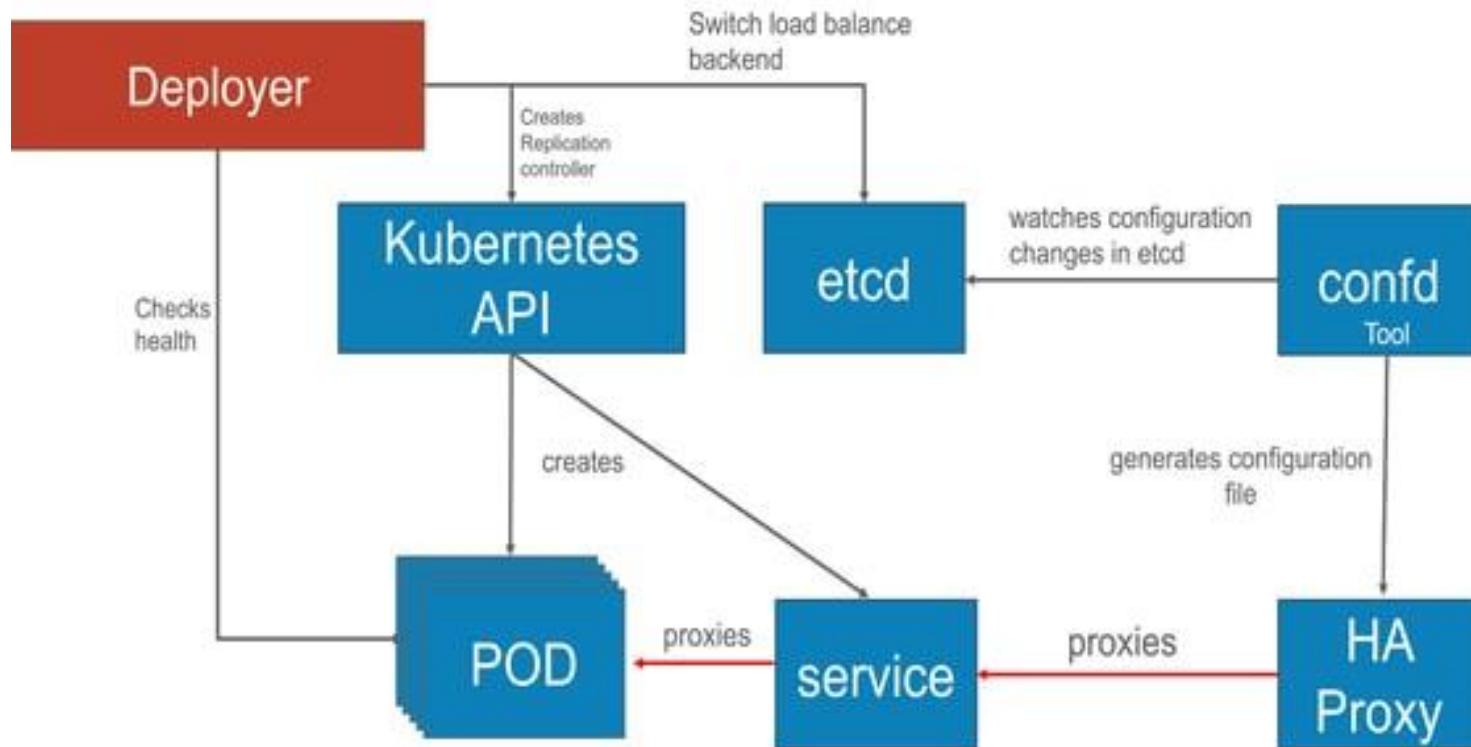


# Different types of services in Kubernetes

Cluster IP	Node Port	Load Balancer	External Name
<ul style="list-style-type: none"><li>• Exposes the service on a cluster-internal IP.</li><li>• Makes the service only reachable from within the cluster.</li><li>• This is the default Service Type.</li></ul>	<ul style="list-style-type: none"><li>• Exposes the service on each Node's IP at a static port.</li><li>• A Cluster IP service to which Node Port service will route, is automatically created.</li></ul>	<ul style="list-style-type: none"><li>• Exposes the service externally using a cloud provider's load balancer.</li><li>• Services, to which the external load balancer will route, are automatically created.</li></ul>	<ul style="list-style-type: none"><li>• Maps the service to the contents of the External Name field by returning a CNAME record with its value.</li><li>• No proxying of any kind is set up.</li></ul>



# Kubernetes architecture



# Minikube –p cluster1 dashboard



Hello Minikube - Kubernetes   Overview - Kubernetes Dashboard

127.0.0.1:52161/api/v1/namespaces/kube-system/services/http:kubernetes-dashboard:/proxy#!/overview?namespace=default

Paused

Apps Insert title here Empire New Tab How to use Assert... Browser Automatio... node.js - How can I ... Freelancer-dev-810... Courses New Tab Google

**kubernetes** + CREATE

☰ Overview

Cluster

Namespaces

Nodes

Persistent Volumes

Roles

Storage Classes

Namespace

default

Overview

Workloads

Cron Jobs

Daemon Sets

Deployments

Jobs

Pods

Replica Sets

Replication Controllers

Discovery and Load Balancing

Services

Name	Labels	Cluster IP	Internal endpoints	External endpoints	Age
kubernetes	component: apiserver provider: kubernetes	10.96.0.1	kubernetes:443 TCP	-	6 minutes

Config and Storage

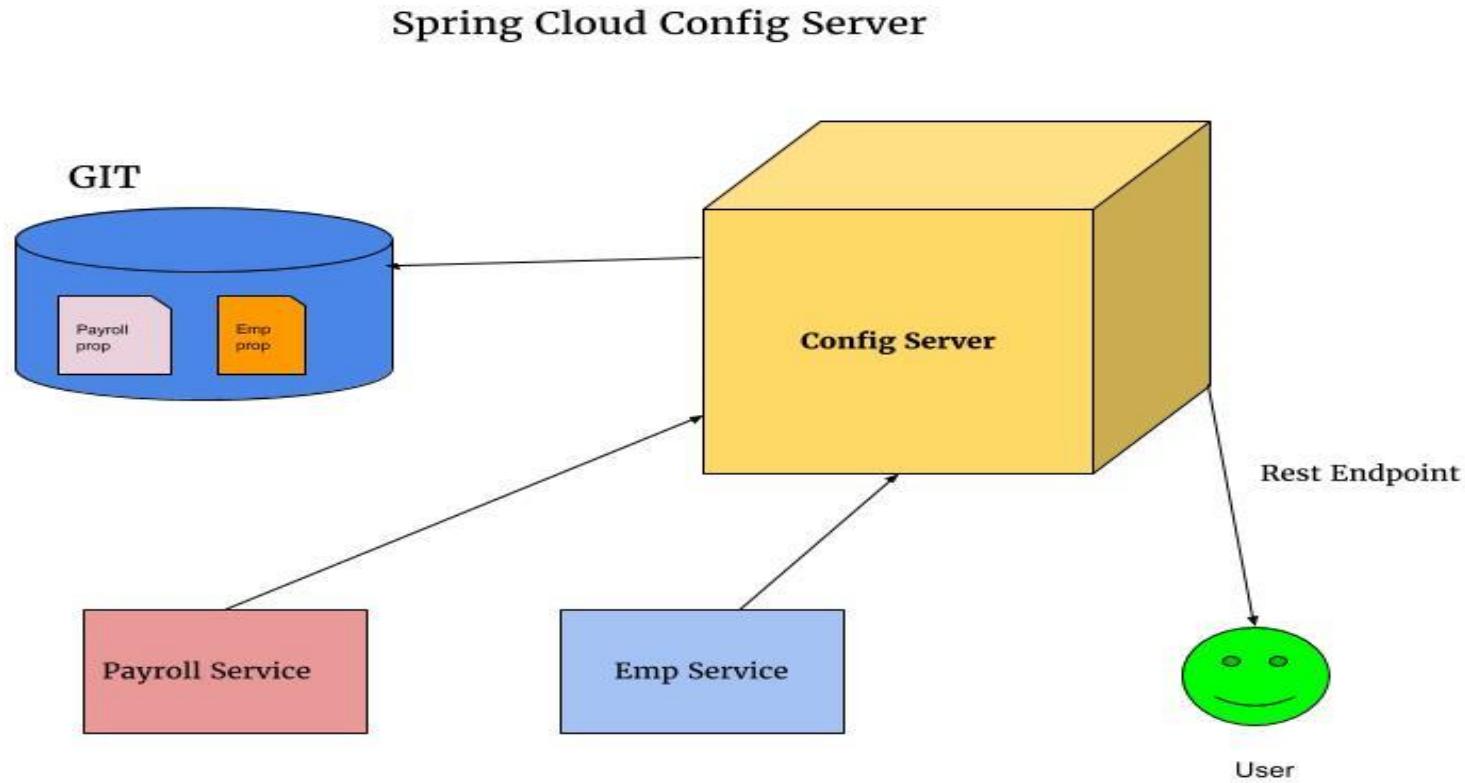
Secrets

Name	Type	Age
default-token-r827j	kubernetes.io/service-account-token	6 minutes

Type here to search

22:50 20/02/2019

# Spring Cloud Config Server with git





# Spring Cloud Config Server with git

SPRING INITIALIZR bootstrap your application now

Generate a  with  and Spring Boot

## Project Metadata

Artifact coordinates

Group

com.in28minutes.microservices

Artifact

spring-cloud-config-server

## Dependencies

Add Spring Boot Starters and dependencies to your application

Search for dependencies

Web, Security, JPA, Actuator, Devtools...

Selected Dependencies

DevTools ✕ Config Server ✕

Generate Project 

Don't know what to look for? Want more options? [Switch to the full version.](#)



# Spring Cloud Config Server with git

```
pository
Rangas-MacBook-Pro:03.microservices rangaraokaranam$ mkdir git-localconfig-repo
Rangas-MacBook-Pro:03.microservices rangaraokaranam$ cd git-localconfig-repo/
Rangas-MacBook-Pro:git-localconfig-repo rangaraokaranam$ git init
Initialized empty Git repository in /in28Minutes/git/spring-micro-services/03.mi
croservices/git-localconfig-repo/.git/
Rangas-MacBook-Pro:git-localconfig-repo rangaraokaranam$ git add -A
Rangas-MacBook-Pro:git-localconfig-repo rangaraokaranam$ git commit -m "first co
mmit"
[master (root-commit) 0898c54] first commit
Committer: Ranga Rao Karanam <rangaraokaranam@Rangas-MacBook-Pro.local>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly:
```

```
git config --global user.name "Your Name"
git config --global user.email you@example.com
```

After doing this, you may fix the identity used for this commit with:

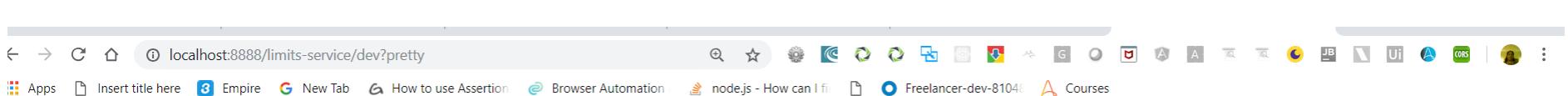
```
git commit --amend --reset-author
```

```
1 file changed, 2 insertions(+)
create mode 100644 limits-service.properties
Rangas-MacBook-Pro:git-localconfig-repo rangaraokaranam$
```



# Spring Cloud Config Server with git

```
1spring.application.name=spring-cloud-config-server
2server.port=8888
3spring.cloud.config.server.git.uri=file:///D:/Microservices/ConfigServerGit
```



```
{"name":"limits-service","profiles":["dev"],"label":null,"version":"b07af9f124423cd6326ac8e10c7d310c95f5d7ab","state":null,"propertySources": [{"name":"file:///D:/Microservices/ConfigServerGit/limits-service-dev.properties","source":{"limits-service.minimum":"1"}}, {"name":"file:///D:/Microservices/ConfigServerGit/limits-service.properties","source":{"limits-service.minimum":"88","limits-service.maximum":"888"}]}]
```



# Micro service With Multiple Port

```
10 // @JsonIgnoreProperties(value = { assetValue })  
11  
12 //dynamic filter  
13 @JsonFilter(value="AssetFilter")  
14 public class Asset {  
15     @Id  
16     @GeneratedValue(strategy=GenerationType.IDENTITY)  
17     @Column(name="Asset_Id")  
18     private int assetId;  
19     @Column(name="Asset_Name",nullable=false,length=50)  
20     private String(assetName);  
21     @Column(name="Asset_Value")  
22     private long assetValue;  
23     @Transient  
24     private int port;  
25  
26     public int getPort() {  
27         return port;  
28     }  
29     public void setPort(int port) {  
30         this.port = port;  
31     }  
32     public int getAssetId() {  
33         return assetId;  
34     }
```



# Micro service With Multiple Port

```
49
50
51 }
52 @RequestMapping(path="/getassetbyid/{id}",method=RequestMethod.GET,produces=MediaType.APPLICATION_JSON)
53 public @ResponseBody MappingJacksonValue getAssetInfo(@PathVariable int id)
54 {
55     Asset asset = assetService.findById(id);
56     asset.setPort(Integer.parseInt(environment.getProperty("local.server.port")));
57     SimpleBeanPropertyFilter propertyFilter=SimpleBeanPropertyFilter.filterOutAllExcept("assetId");
58     FilterProvider filters = new SimpleFilterProvider().addFilter("AssetFilter", propertyFilter);
59     MappingJacksonValue mapping=new MappingJacksonValue(asset);
60     mapping.setFilters(filters);
61     return mapping;
62
63 }
64 @CrossOrigin(origins = "*")
65 @RequestMapping(path="/getassetbyname/{name}",method=RequestMethod.GET,produces=MediaType.APPLICATION_JSON)
66 public @ResponseBody Asset getAssetByNameInfo(@PathVariable String name)
67 {
68     System.out.println(name);
69     return assetService.findByName(name);
```



# Micro service With Multiple Port

Run Configurations

## Create, manage, and run configurations

Run a Java application

The screenshot shows the 'Run Configurations' dialog with the following details:

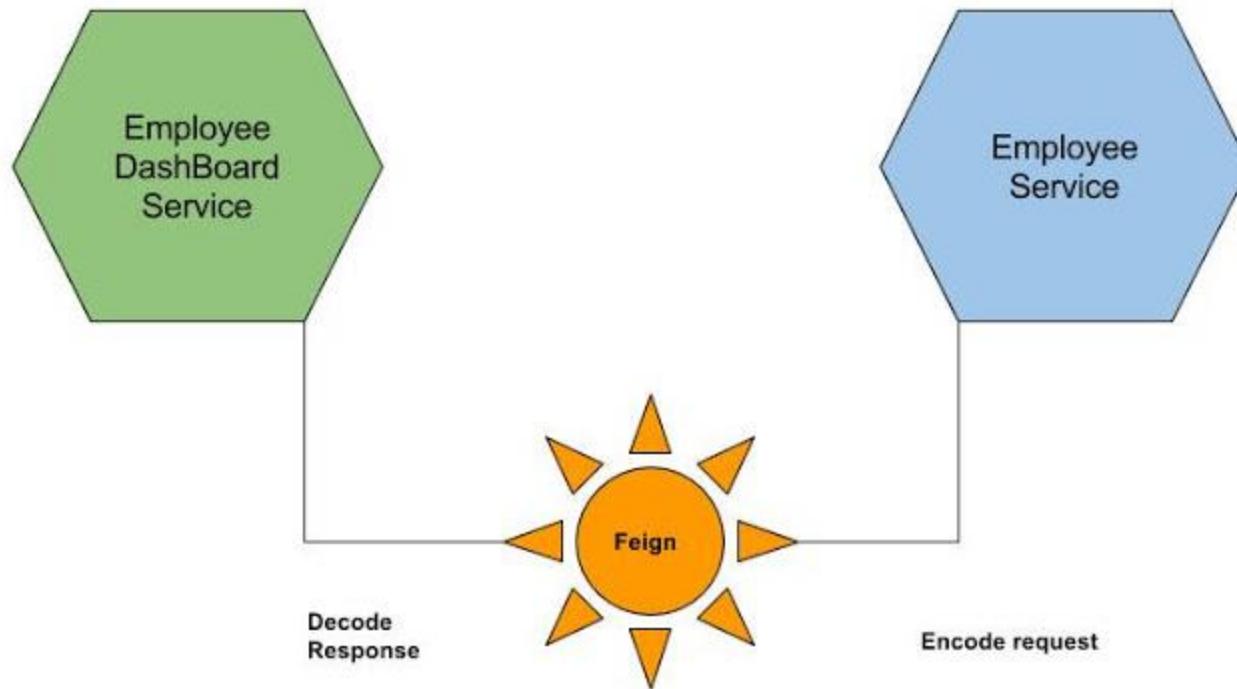
- Name:** ActuatorDemoApplication 6061
- Main tab selected:** Arguments
- Program arguments:** -Dserver.port=6061
- VM arguments:** (empty)
- Working directory:** \${workspace\_loc:jpademo}
- Buttons at the bottom:** Revert, Run (highlighted), Close

The left sidebar lists various application types and their configurations:

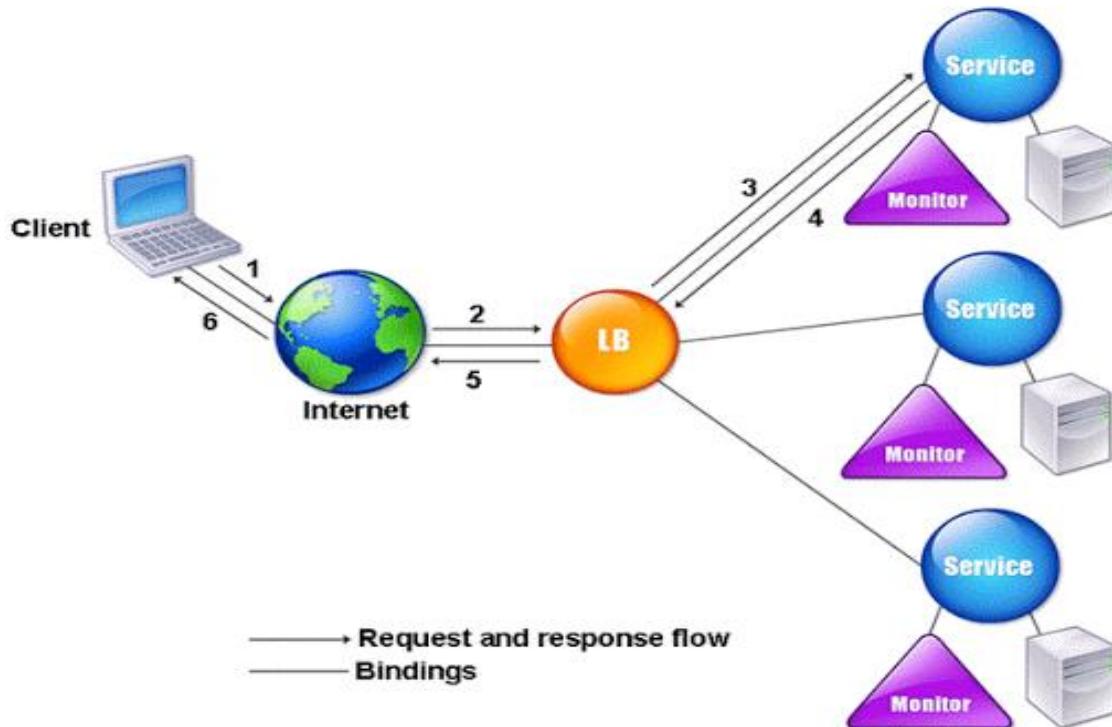
- Java Applet
- Java Application
  - ActuatorDemoApplication 6060
  - ActuatorDemoApplication 6061 (selected)
  - JpademoApplication8000
  - SpringCloudConfigServerDemoApplication
- JUnit
- JUnit Plug-in Test
- Launch Group
- Maven Build
- Mwe2 Launch
- Node.js Application
- OSGi Framework
- Scala Application
- Scala Interpreter
- Task Context Test
- XSL

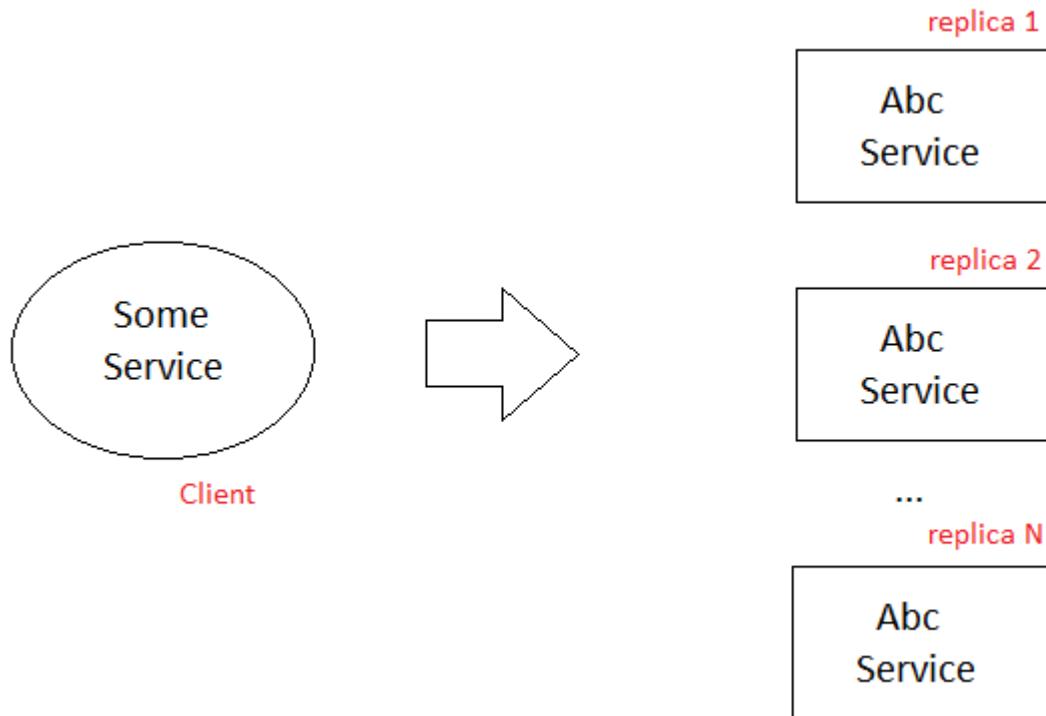
At the bottom left, it says "Filter matched 28 of 39 items".

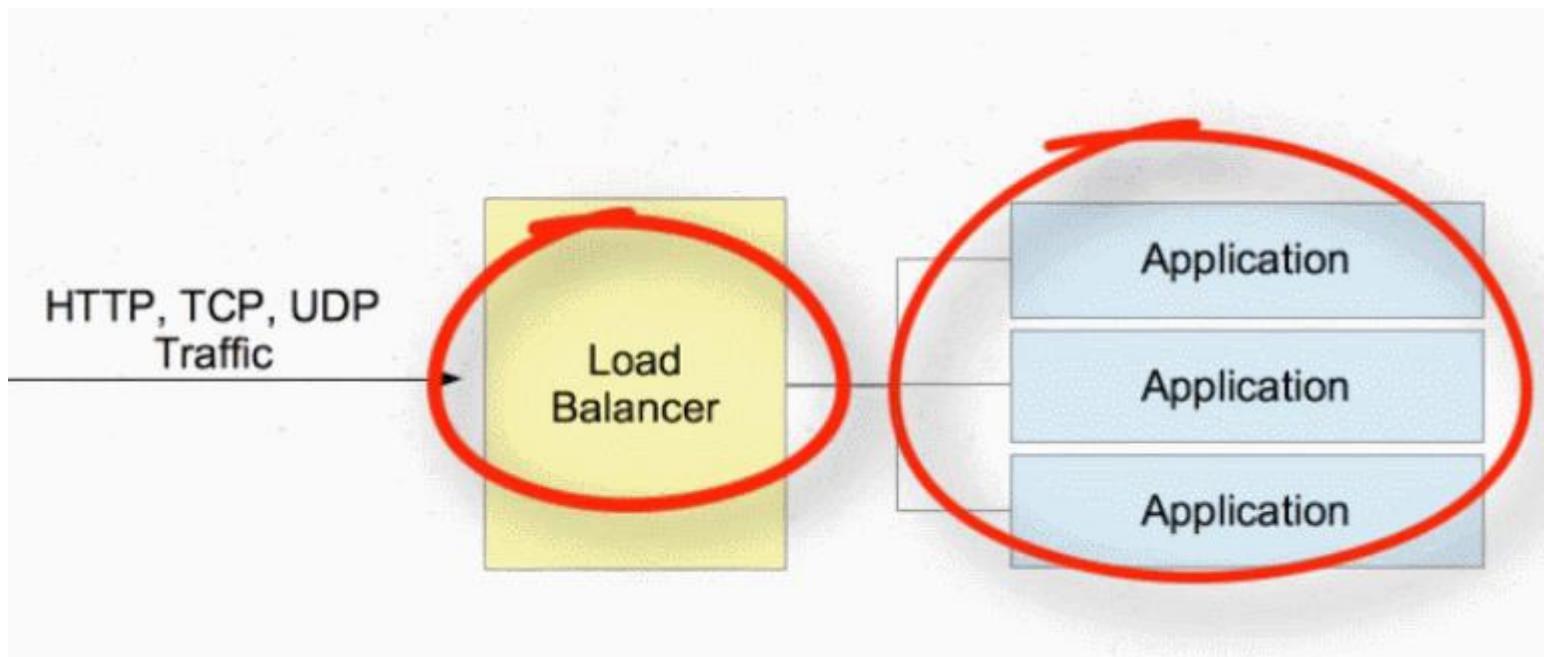
# Feign Client

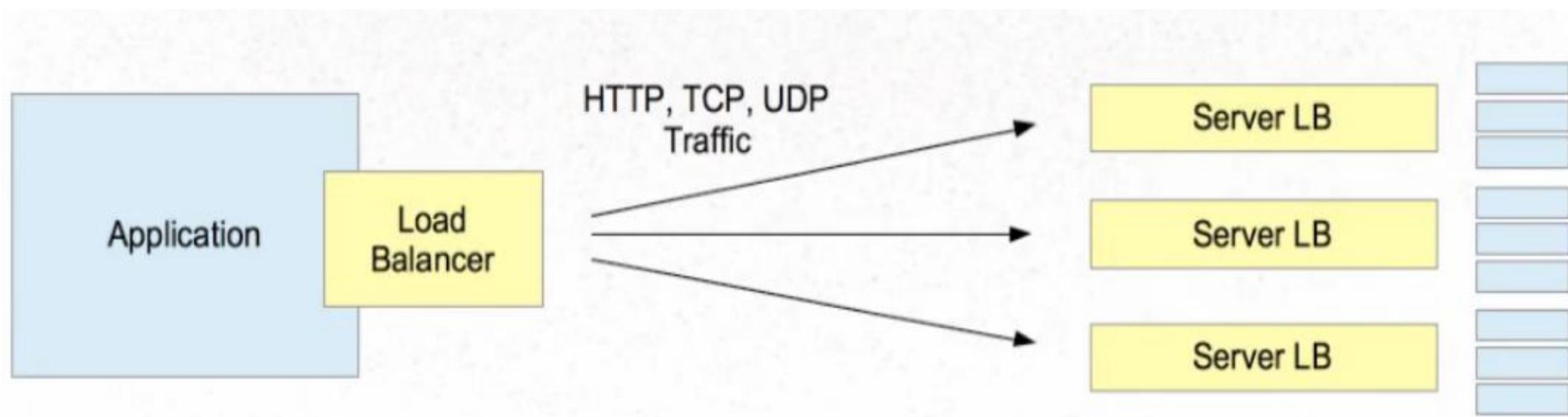


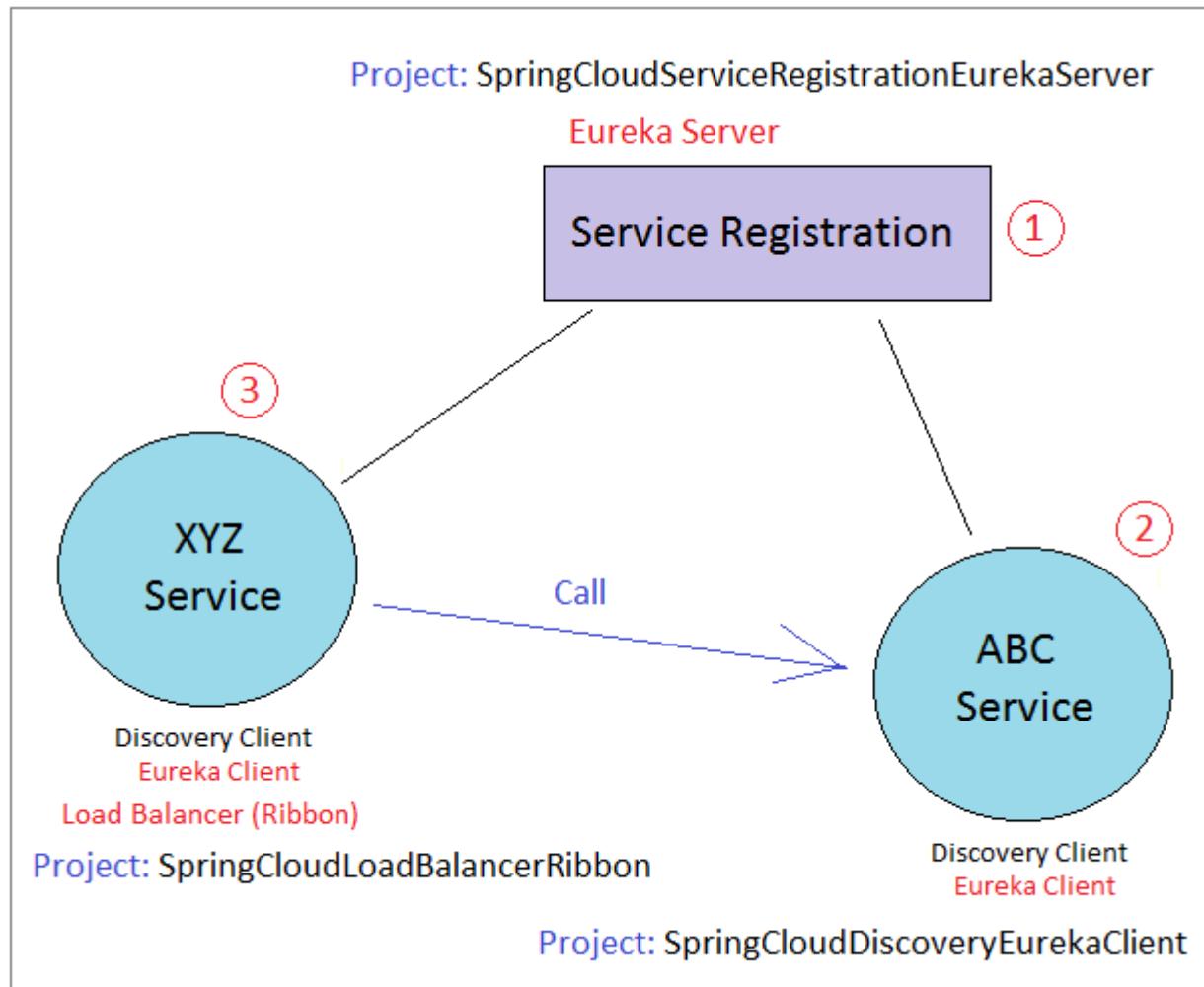
# RIBBON LOAD BALANCER

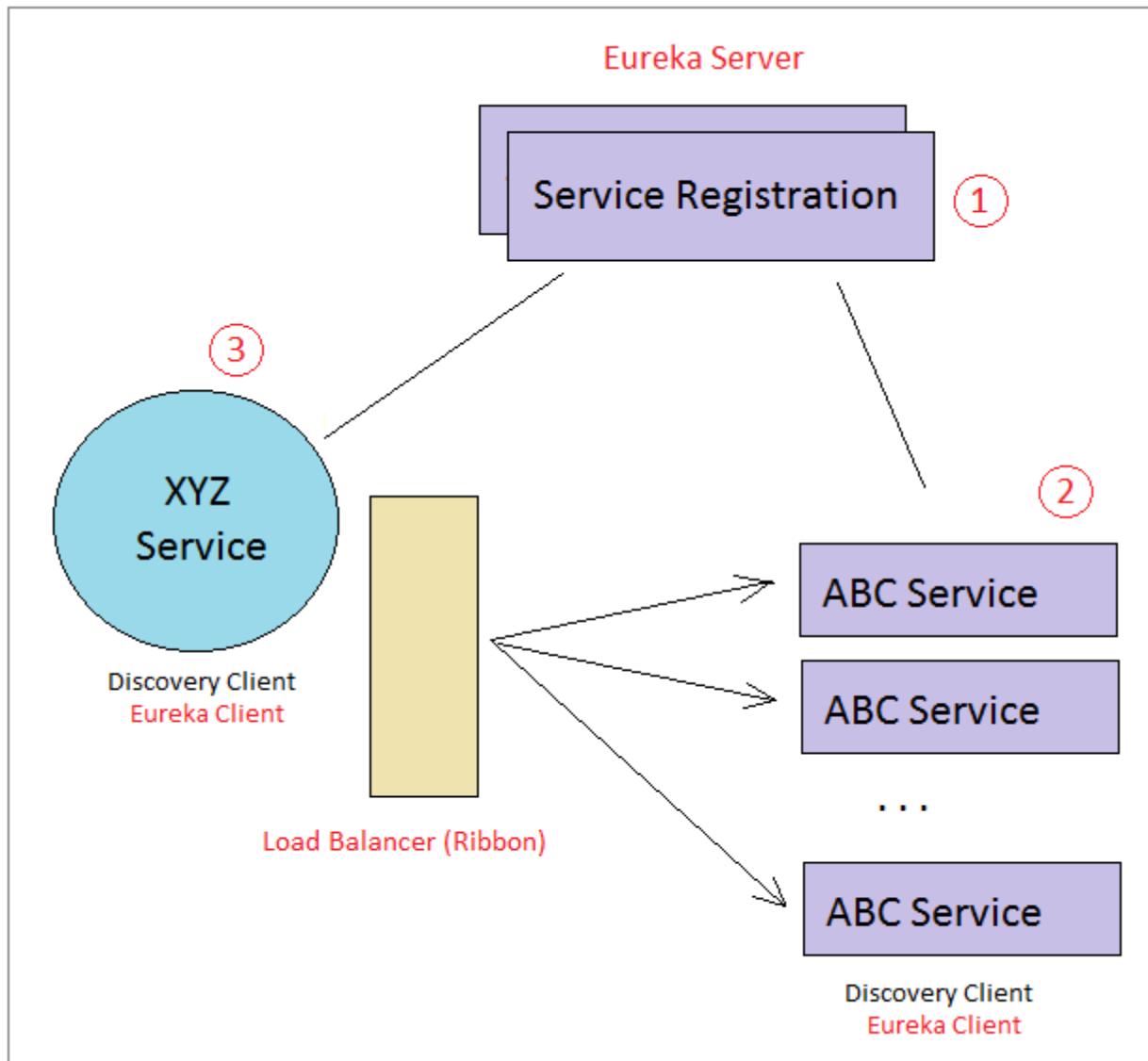














# RIBBON LOAD BALANCER

---

- Ribbon is a client-side load balancer, which gives you a lot of control over the behavior of HTTP and TCP clients.
- Ribbon's Client component offers a good set of configuration options such as connection timeouts, retries, retry algorithm (exponential, bounded back off) etc.
- Ribbon comes built in with a pluggable and customizable Load Balancing component.



# RIBBON LOAD BALANCER

---

- Some of the load balancing strategies offered are listed below:
- Simple Round Robin LB
- Weighted Response Time LB
- Zone Aware Round Robin LB
- Random LB



# RIBBON LOAD BALANCER

---

- Ribbon provides the following features:
- Load balancing
- Fault tolerance
- Multiple protocols (HTTP, TCP, UDP) support in an asynchronous and reactive model
- Caching and batching



# RIBBON LOAD BALANCER

Bean Type	Bean Name	Class Name
IClientConfig	ribbonClientConfig	DefaultClientConfigImpl
IRule	ribbonRule	ZoneAvoidanceRule
IPing	ribbonPing	DummyPing
ServerList<Server>	ribbonServerList	ConfigurationBasedServerList
ServerListFilter<Server>	ribbonServerListFilter	ZonePreferenceServerListFilter
ILoadBalancer	ribbonLoadBalancer	ZoneAwareLoadBalancer
ServerListUpdater	ribbonServerListUpdater	PollingServerListUpdater



# RIBBON LOAD BALANCER

The properties file (sample-client.properties)

```
# Max number of retries on the same server (excluding the first try)
sample-client.ribbon.MaxAutoRetries=1

# Max number of next servers to retry (excluding the first server)
sample-client.ribbon.MaxAutoRetriesNextServer=1

# Whether all operations can be retried for this client
sample-client.ribbon.OkToRetryOnAllOperations=true

# Interval to refresh the server list from the source
sample-client.ribbon.ServerListRefreshInterval=2000

#
```



# RIBBON LOAD BALANCER

Connect timeout used by Apache HttpClient  
sample-client.ribbon.ConnectTimeout=3000

# Read timeout used by Apache HttpClient  
sample-client.ribbon.ReadTimeout=3000

# Initial list of servers, can be changed via Archaius dynamic property at runtime  
sample-client.ribbon.listOfServers=www.microsoft.com:80, www.yahoo.com:80, www.google.com:80



# API Gateway

---

- Authentication, authorization and security
- Rate Limits
- Fault Tolerance
- Service Aggregation



# Zipkin

```
MINGW64:/d/Program Files/Docker Toolbox
##          .
## ## ##      ==
## ## ## ## ## ===
/''''''''''''\_\_/_ ===
~~~ {~~~ ~~~ ~~~ ~~~ ~~~ ~ / === ~~~
     \_\_ o
         \_\_ \_\_
docker is configured to use the default machine with IP 192.168.99.100
For help getting started, check out the docs at https://docs.docker.com

Start interactive shell

Balasubramaniam@DESKTOP-55AGI0I MINGW64 /d/Program Files/Docker Toolbox
$ docker run -d -p 9411:9411 openzipkin/zipkin
Unable to find image 'openzipkin/zipkin:latest' locally
latest: Pulling from openzipkin/zipkin
ff3a5c916c92: Pulling fs layer
a8906544047d: Pulling fs layer
1590b87a38029: Pulling fs layer
5a45314016bd: Waiting
20b47c038743: Waiting
9af51dacdfe7: Waiting
```

Security Sc... Code Docume...



# AWS Lambda Serverless Environment

---

- **What is AWS Lambda?**
- Amazon explains, AWS Lambda ( $\lambda$ ) as a ‘serverless’ compute service, meaning the developers, don’t have to worry about which AWS resources to launch, or how will they manage them, they just put the code on lambda and it runs, it’s that simple! It helps you to focus on core-competency i.e. App Building or the code.



# AWS Lambda Serverless Environment

---

- **Where will I use AWS Lambda?**
- AWS Lambda executes your backend code, by automatically managing the AWS resources. When we say 'manage', it includes launching or terminating instances, health checkups, auto scaling, updating or patching new updates etc.



# AWS Lambda Serverless Environment

---

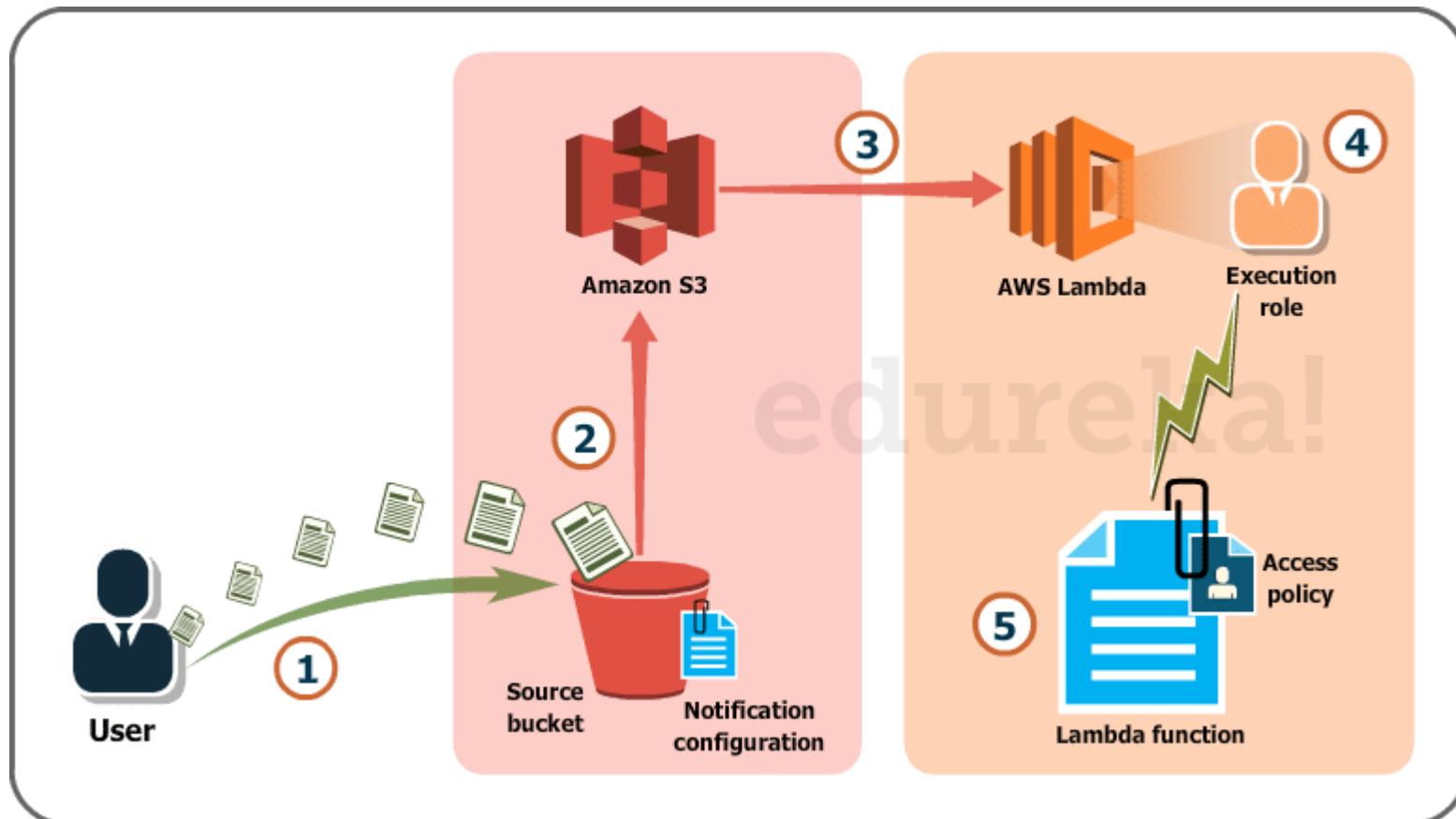
- **So, how does it work?**
- The code that you want Lambda to run is known as a **Lambda function**. Now, as we know a function runs only when it is called, right? Here, **Event Source** is the entity which triggers a Lambda Function, and then the task is executed.



# AWS Lambda Serverless Environment

---

- Let's take an example to understand it more clearly.
- Suppose you have an app for image uploading. Now when you upload an image, there are a lot of tasks involved before storing it, such as resizing, applying filters, compression etc.
- So, this task of uploading of an image can be defined as an **Event Source** or the 'trigger' that will call the Lambda Function, and then all these tasks can be executed via the Lambda function.



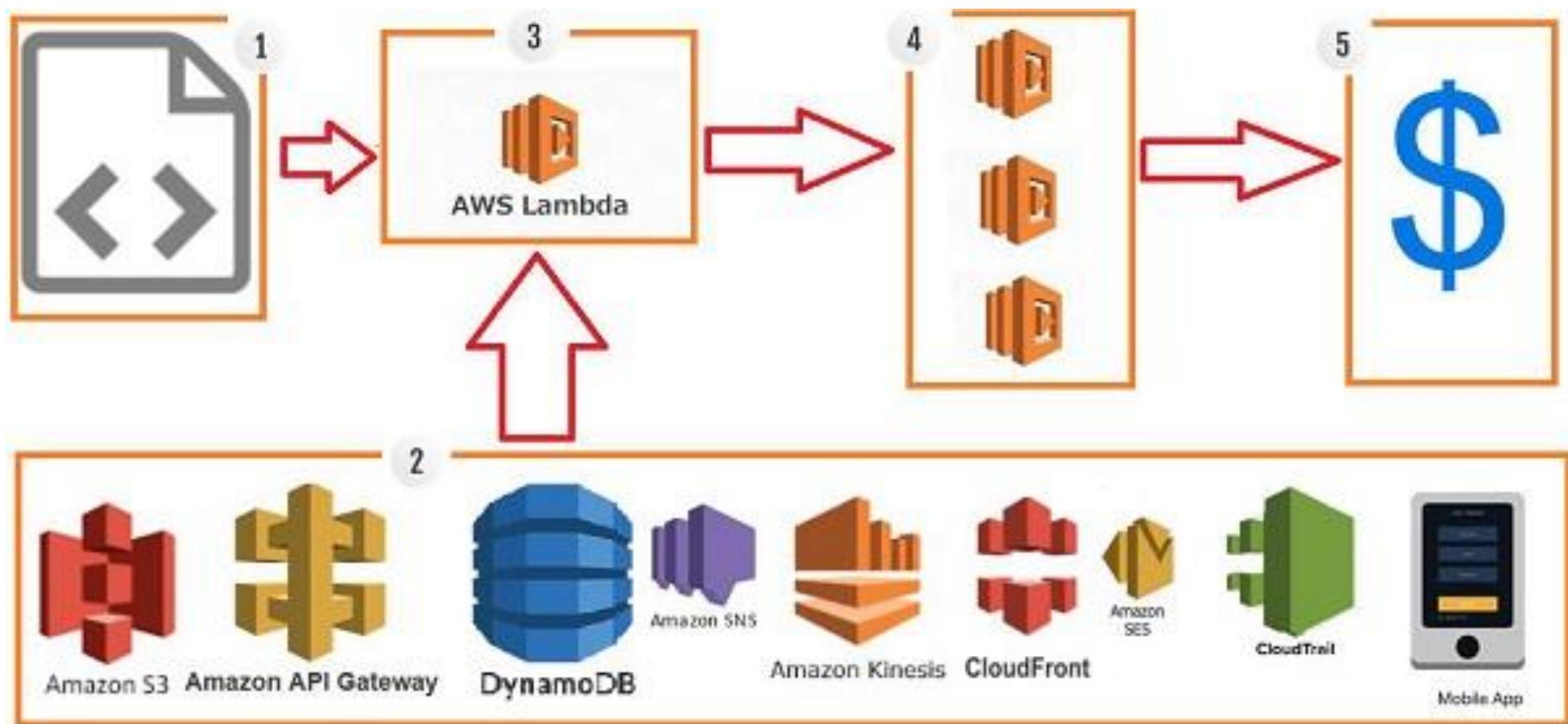


# AWS Lambda Serverless Environment

- The whole process, as you can see in the diagram, is divided into 5 steps, let's understand each one of them.
- User uploads an image (object) to a source bucket in S3 which has notification attached to it, for Lambda.
- The notification is read by S3 and it decides where to send that notification.
- S3 sends the notification to Lambda, this notification acts as an invoke call of the lambda function.
- Execution role in Lambda can be defined by using IAM (Identity and Access Management) to give access permission for the AWS resources, for this example here it would be S3.
- Finally, it invokes the desired lambda function which works on the object which has been uploaded to the S3 bucket.



# How AWS Lambda Works?





# AWS Lambda Steps

- **Step 1** – Upload AWS lambda code in any of languages AWS lambda supports, that is NodeJS, Java, Python, C# and Go.
- **Step 2** – These are few AWS services on which AWS lambda can be triggered.
- **Step 3** – AWS Lambda which has the upload code and the event details on which the trigger has occurred. For example, event from Amazon S3, Amazon API Gateway, Dynamo dB, Amazon SNS, Amazon Kinesis, CloudFront, Amazon SES, CloudTrail, mobile app etc.



# AWS Lambda Steps

- **Step 4 –** Executes AWS Lambda Code only when triggered by AWS services under the scenarios such as –
  - User uploads files in S3 bucket
  - http get/post endpoint URL is hit
  - data is added/updated/deleted in dynamo dB tables
  - push notification
  - data streams collection
  - hosting of website
  - email sending
  - mobile app, etc.
- **Step 5 –** Remember that AWS charges only when the AWS lambda code executes, and not otherwise.

# Advantages of using AWS Lambda



- Ease of working with code(Infrastructure Burden Free)
- Log Provision
- Billing based on Usage
- Multi Language Support
- Ease of code authoring and deploying
-

# Disadvantages of using AWS Lambda



- It is not suitable for small projects.
- You need to carefully analyze your code and decide the memory and timeout. Incase if your function needs more time than what is allocated, it will get terminated as per the timeout specified on it and the code will not be fully executed.
- Since AWS Lambda relies completely on AWS for the infrastructure, you cannot install anything additional software if your code demands it.

# Events that Trigger AWS Lambda



- Entry into a S3 object
- Insertion, updation and deletion of data in Dynamo DB table
- Push notifications from SNS
- GET/POST calls to API Gateway
- Headers modification at viewer or origin request/response in CloudFront
- Log entries in AWS Kinesis data stream
- Log history in CloudTrail

# Use Cases of AWS Lambda



- S3 Object and AWS Lambda
- Amazon S3 passes the event details to AWS Lambda when there is any file upload in S3. The details of the file upload or deletion of file or moving of file is passed to the AWS Lambda.
- The code in AWS Lambda can take the necessary step for when it receives the event details. For Example creating thumbnail of the image inserted into S3.

# Use Cases of AWS Lambda



- DynamoDB and AWS Lambda
- DynamoDB can trigger AWS Lambda when there is data added, updated and deleted in the table. AWS Lambda event has all the details of the AWS DynamoDB table about the insert /update or delete.
- API Gateway and AWS Lambda
- API Gateway can trigger AWS Lambda on GET/POST methods. We can create a form and share details with API Gateway endpoint and use it with AWS Lambda for further processing, for Example, making an entry of the data in DynamoDB table.

# Use Cases of AWS Lambda



- SNS and AWS Lambda
- SNS is used for push notification, sending SMS etc. We can trigger AWS Lambda when there is any push notification happening in SNS. We can also send SMS to the phone number from AWS Lambda when it receives the trigger.

# Use Cases of AWS Lambda



- Scheduled Events and AWS Lambda
- Scheduled Events can be used for cron jobs. It can trigger AWS Lambda to carry out the task at regular time pattern.
- CloudTrail and AWS Lambda
- CloudTrail can be helpful in monitoring the logs on the account. We can use AWS Lambda to further process the CloudTrail logs .

# Use Cases of AWS Lambda



- Kinesis and AWS Lambda
- Kinesis is used to capture/store real time tracking data coming from website clicks, logs, social media feeds and a trigger to AWS Lambda can do additional processing on this logs.



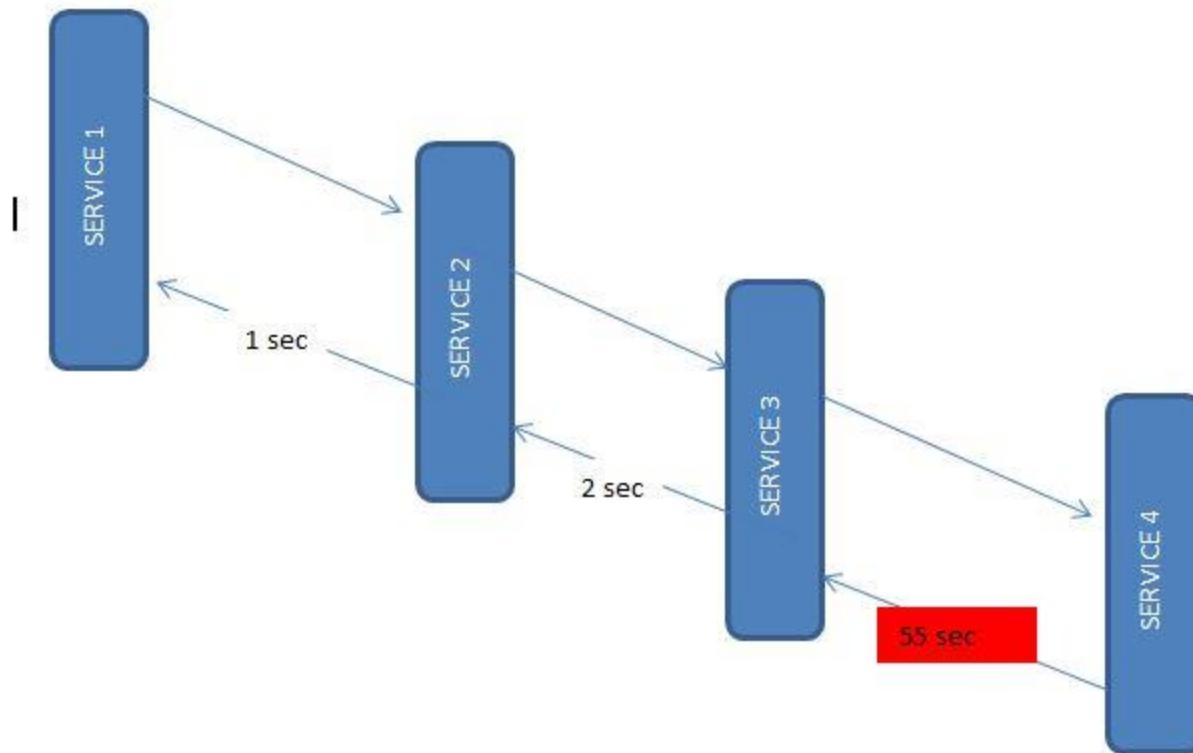
# Zipkin server

---

- Download from
- download the latest Zipkin server from maven repository and run the executable jar file using below command.
- `java -jar zipkin-server-2.12.1-exec.jar`
- Once Zipkin is started, we can see the Web UI at <http://localhost:9411/zipkin/>



# Microservices Interactions





# Prometheus and grafana

---

- `prometheus.exe --config.file="prometheus.yml" --web.listen-address=:9092`



# Spring Data Flow





# Spring Data Flow

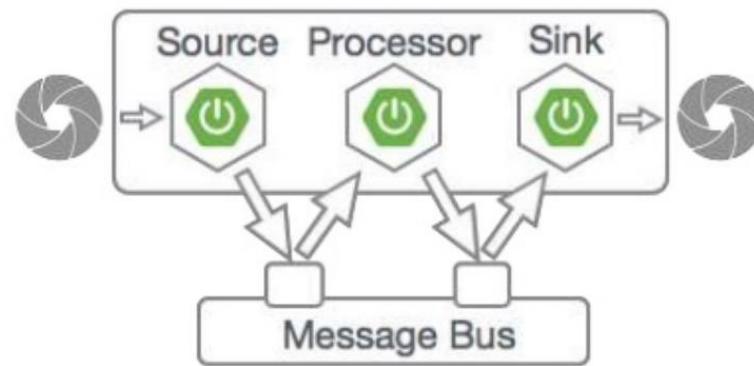
---

- Spring Cloud Data Flow is a toolkit to build real-time data integration and data processing pipelines by establishing message flows between Spring Boot applications that could be deployed on top of different runtimes.
- Long lived applications require Stream Applications while Short lived applications require Task Applications.
- In this example we make use of Stream Applications. Previously we had already developed Spring Cloud Stream applications to understand the concept of Spring Cloud Stream Source and Spring Cloud Sink and their benefit.

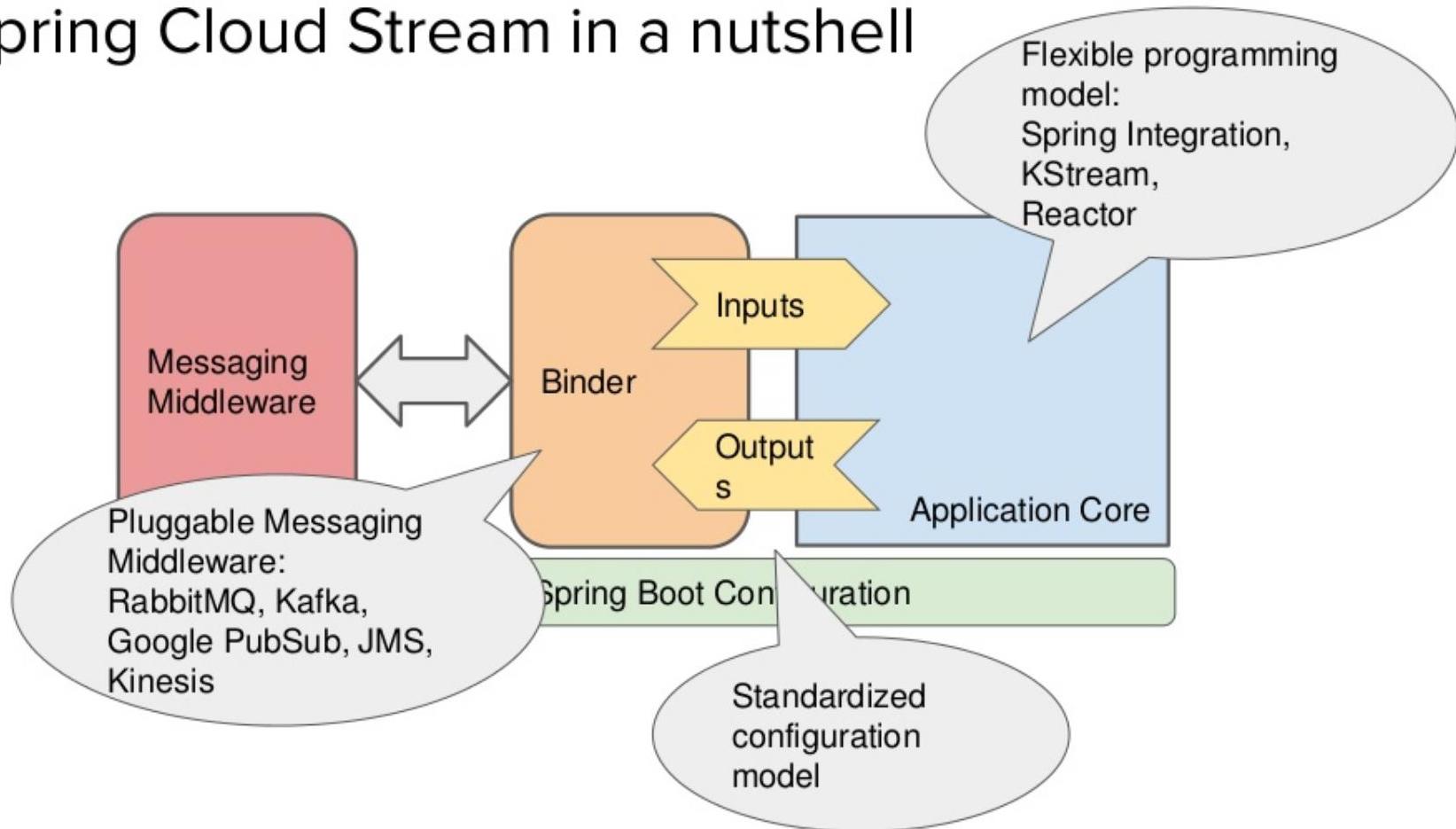


# Microservices are not just for web apps

- Spring Cloud Stream
  - Event driven based Microservices
  - Loose coupling via pub/sub messaging
  - Opinionated Primitives
  - For **integration** and **stream processing** use-cases



# Spring Cloud Stream in a nutshell





# Microservices are not just for web apps

- Spring Cloud Task
  - Tasks are short lived Boot Microservices
  - System tracks invocations, exit-status
  - Spring Batch Jobs are wrapped as Tasks
  - Useful for ETL between databases and filesystems





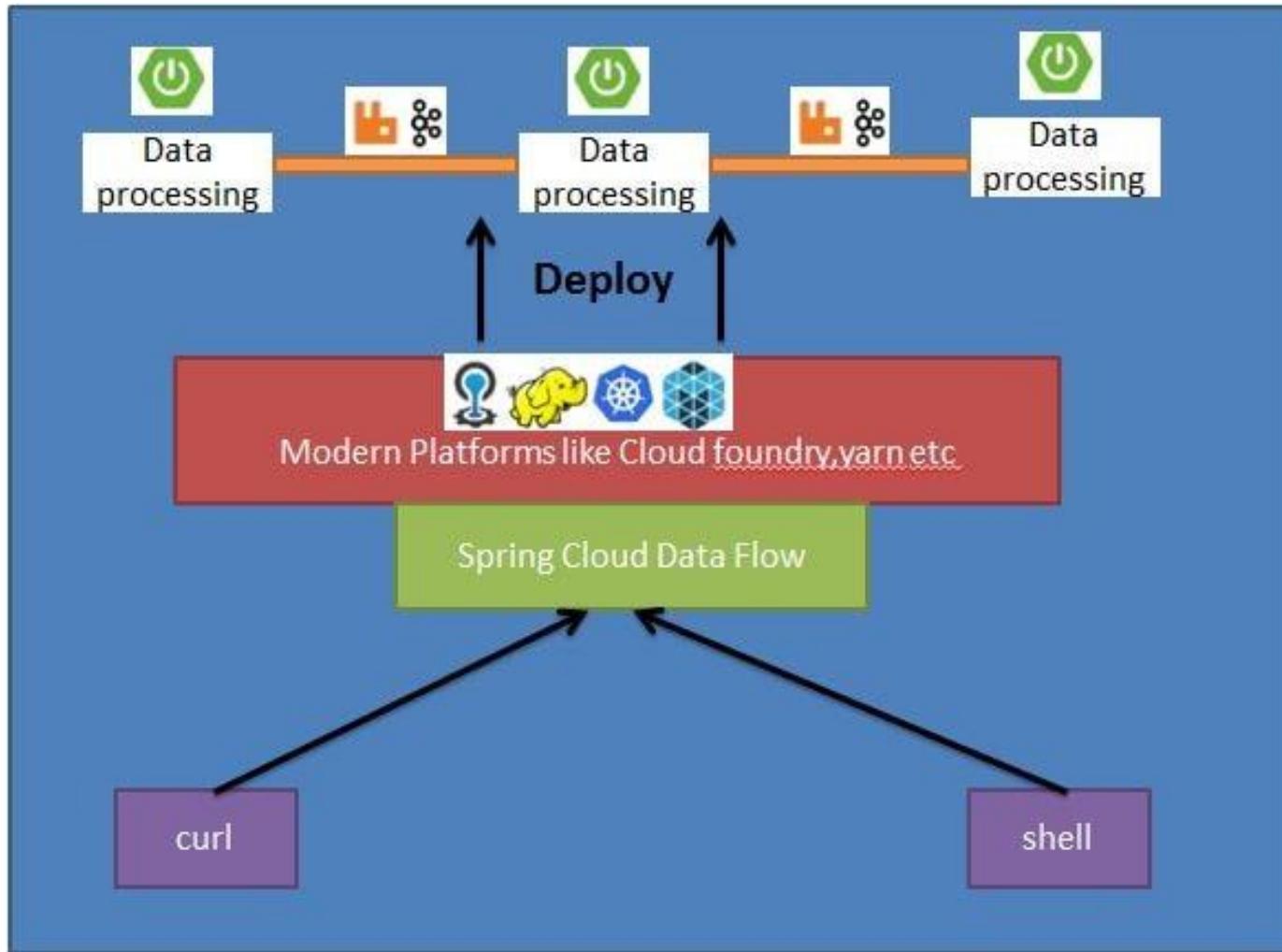
# Spring Data Flow

---

- Microservice based Streaming and Batch data processing for Cloud Foundry and Kubernetes.
- Spring Cloud Data Flow provides tools to create complex topologies for streaming and batch data pipelines.
- The data pipelines consist of Spring Boot apps, built using the Spring Cloud Stream or Spring Cloud Task microservice frameworks.
- Spring Cloud Data Flow supports a range of data processing use cases, from ETL to import/export, event streaming, and predictive analytics.



# Spring Data Flow





# Spring Data Flow

- The Spring Cloud Data Flow server uses Spring Cloud Deployer, to deploy data pipelines made of Spring Cloud Stream or Spring Cloud Task applications onto modern platforms such as Cloud Foundry and Kubernetes.
- Custom stream and task applications, targeting different middleware or data services, can be built using the familiar Spring Boot style programming model.



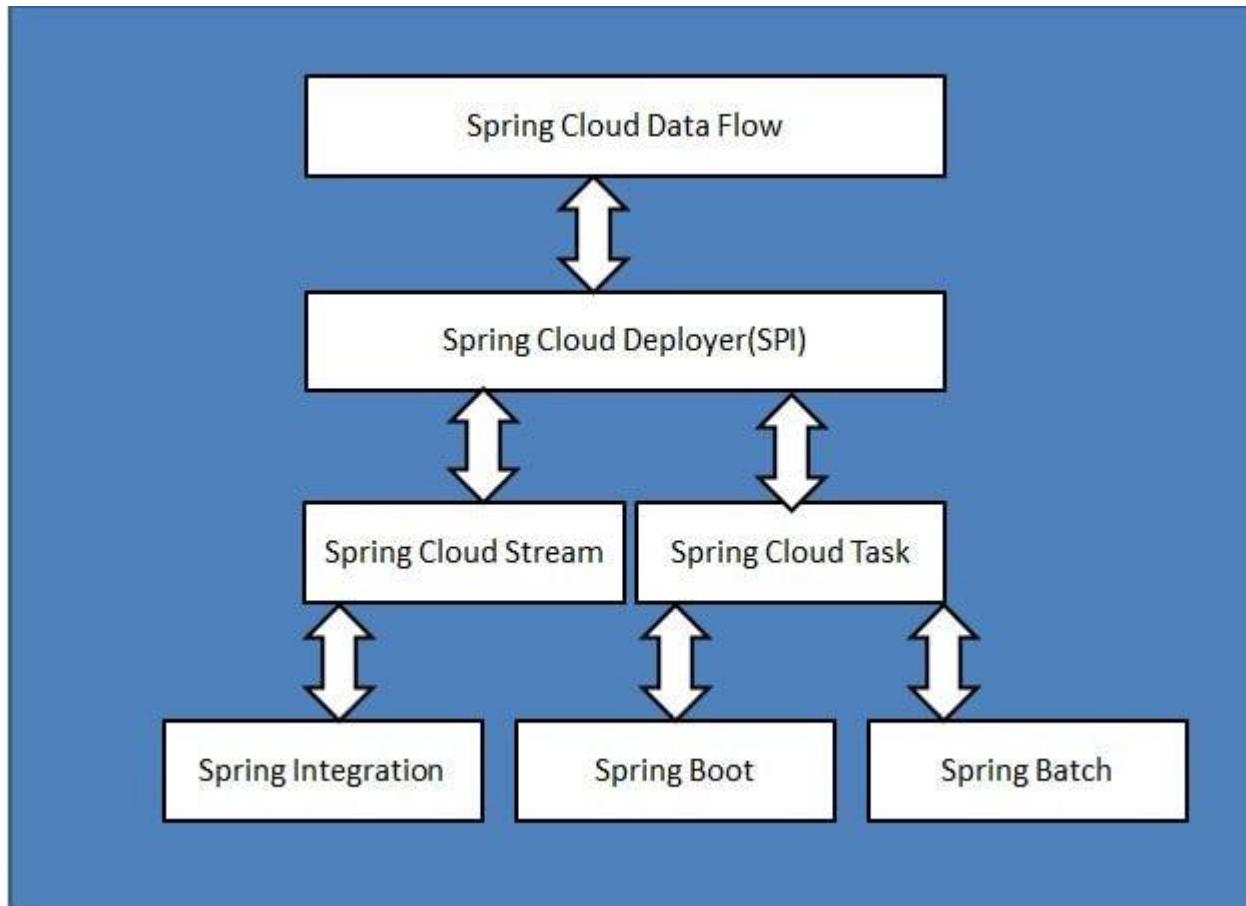
# Spring Data Flow

---

- Pipelines consist of Spring Boot apps, built using the Spring Cloud Stream or Spring Cloud Task microservice frameworks.
- SCDF can be accessed using the REST API exposed by it or the web UI console.
- We can make use of metrics, health checks, and the remote management of each microservice application .
- Also we can scale stream and batch pipelines without interrupting data flows.
- With SCDF we build data pipelines for use cases like data ingestion, real-time analytics, and data import and export.

# Spring Data Flow

- SCDF is composed of the following Spring Projects-





# Spring Data Flow

---

- `java -jar spring-cloud-dataflow-server-local-1.3.0.M1.jar`
- `java -jar spring-cloud-dataflow-shell-1.3.0.M1.jar`



# Spring data flow

---

- app register --name myprocessor --type processor --uri file:///F:/virtusa\_microservices\_oct2019/source-0.0.1-SNAPSHOT.jar



c:\ dataflow 1.3.0.M1

0 Dir(s) 66,305,060,864 bytes free

```
F:\virtusa_microservices_oct2019>java -jar spring-cloud-dataflow-shell-1.3.0.M1.jar
```

1.3.0.M1

```
Welcome to the Spring Cloud Data Flow shell. For assistance hit TAB or type "help".  
dataflow:>_
```





```
dataflow 1.3.0.M1
dataflow:>app register --name source-app --type source --uri file:///F:/virtusa_microservices_oct2019/source-0.0.1-SNAPSHOT.jar
Successfully registered application 'source:source-app'
dataflow:>app register --name processor-app --type processor --uri file:///F:/virtusa_microservices_oct2019/processor-0.0.1-SNAPSHOT.jar
Successfully registered application 'processor:processor-app'
dataflow:>app register --name sink-app --type sink --uri file:///F:/virtusa_microservices_oct2019/sink-0.0.1-SNAPSHOT.jar
Successfully registered application 'sink:sink-app'
dataflow:>stream create --name log-data --definition 'source-app|processor-app|sink-app'

Command failed org.springframework.cloud.dataflow.rest.client.DataFlowClientException: Cannot create stream log-data because another one has already been created with the same name

dataflow:>stream create --name log-data --definition 'source-app|processor-app|sink-app'

Created new stream 'log-data'
dataflow:>stream deploy --name log-data
Deployment request has been sent for stream 'log-data'
dataflow:>
```





J Spring Cloud Tutorial - Stream Pr x | 6 Practical Uses for a Microservice x | Google Download the jar using http://re... x | Spring Cloud Data Flow x +

localhost:9393/dashboard/#/apps

Apps Insert title here Empire New Tab How to use Assertions... Browser Automation... node.js - How can I... Freelancer-dev-810... Courses New Tab hi airtel Airtel 4G Hotspot nt8F83

Paused

## spring

Apps Runtime Streams Tasks Jobs Analytics About

### Apps

This section lists all the available applications and provides the control to register/unregister them (if applicable).

REGISTER APPLICATION(S)			NO APP SELECTED TO UNREGISTER		BULK IMPORT APPLICATIONS		Filter items	↻
	Name	Type	URI				Actions	
<input type="checkbox"/>	source-app	source	maven://com.virtusa.source:source-0.0.1-SNAPSHOT.jar					
<input type="checkbox"/>	processor-app1	processor	maven://com.virtusa.processor:processor-0.0.1-SNAPSHOT.jar					
<input type="checkbox"/>	sink1-app	sink	maven://com.virtusa.sink:sink-0.0.1-SNAPSHOT.jar					

« Previous 1 Next »

© 2017 Pivotal Software, Inc.  
All Rights Reserved.

PROJECT  
[Project Page](#)  
[Issue Tracker](#)

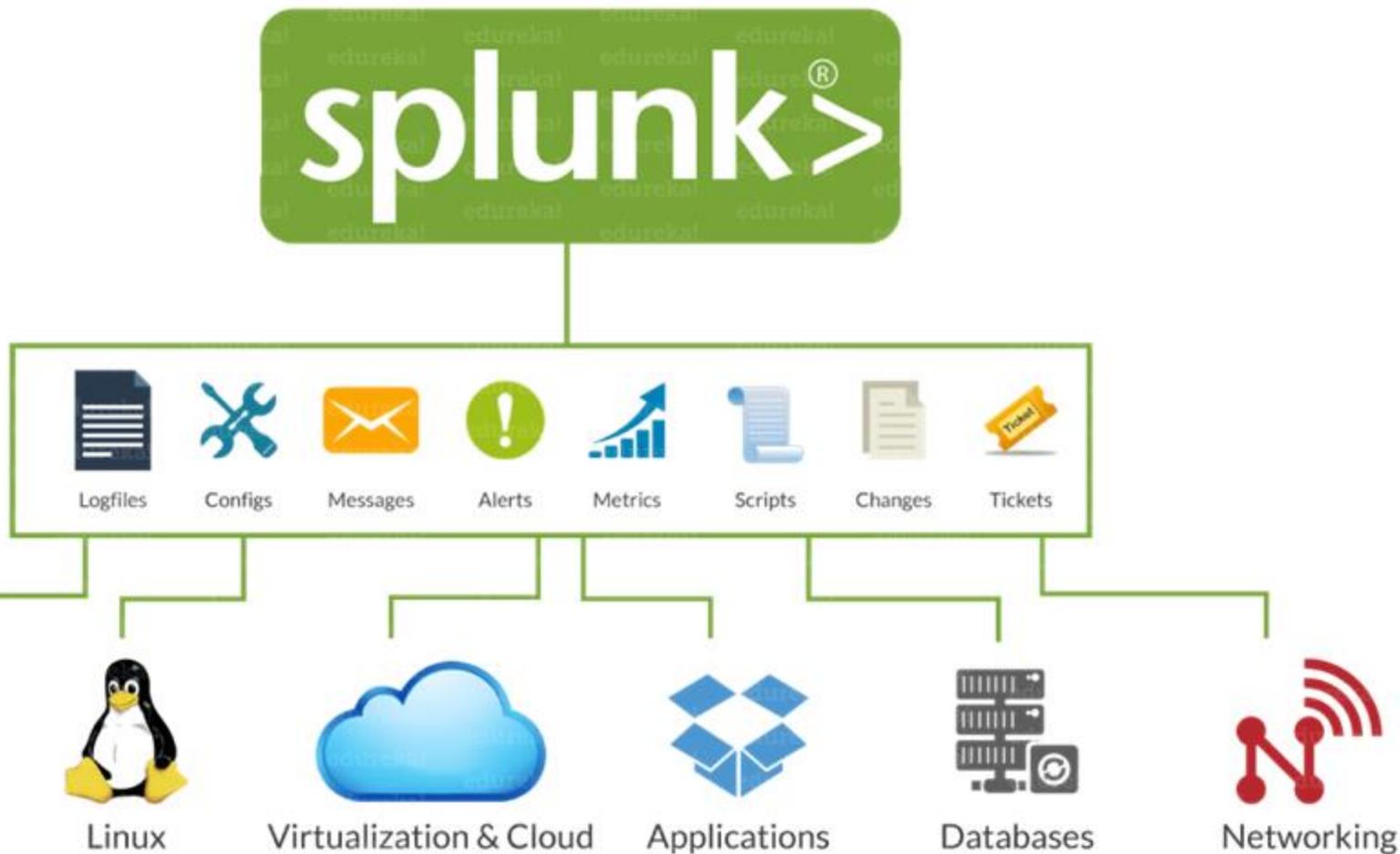
DOCUMENTATION  
[Docs](#)  
[Sources](#)  
[Api Docs](#)

NEED HELP?  
[For questions + support:](#)  
[Stackoverflow](#)



- `stream create --name log-data --definition 'source-app | processor-app| sink-app'`
- `stream deploy --name log-data`
- Go to Spring Cloud Data Flow UI Console -  
`http://localhost:9393/dashboard`

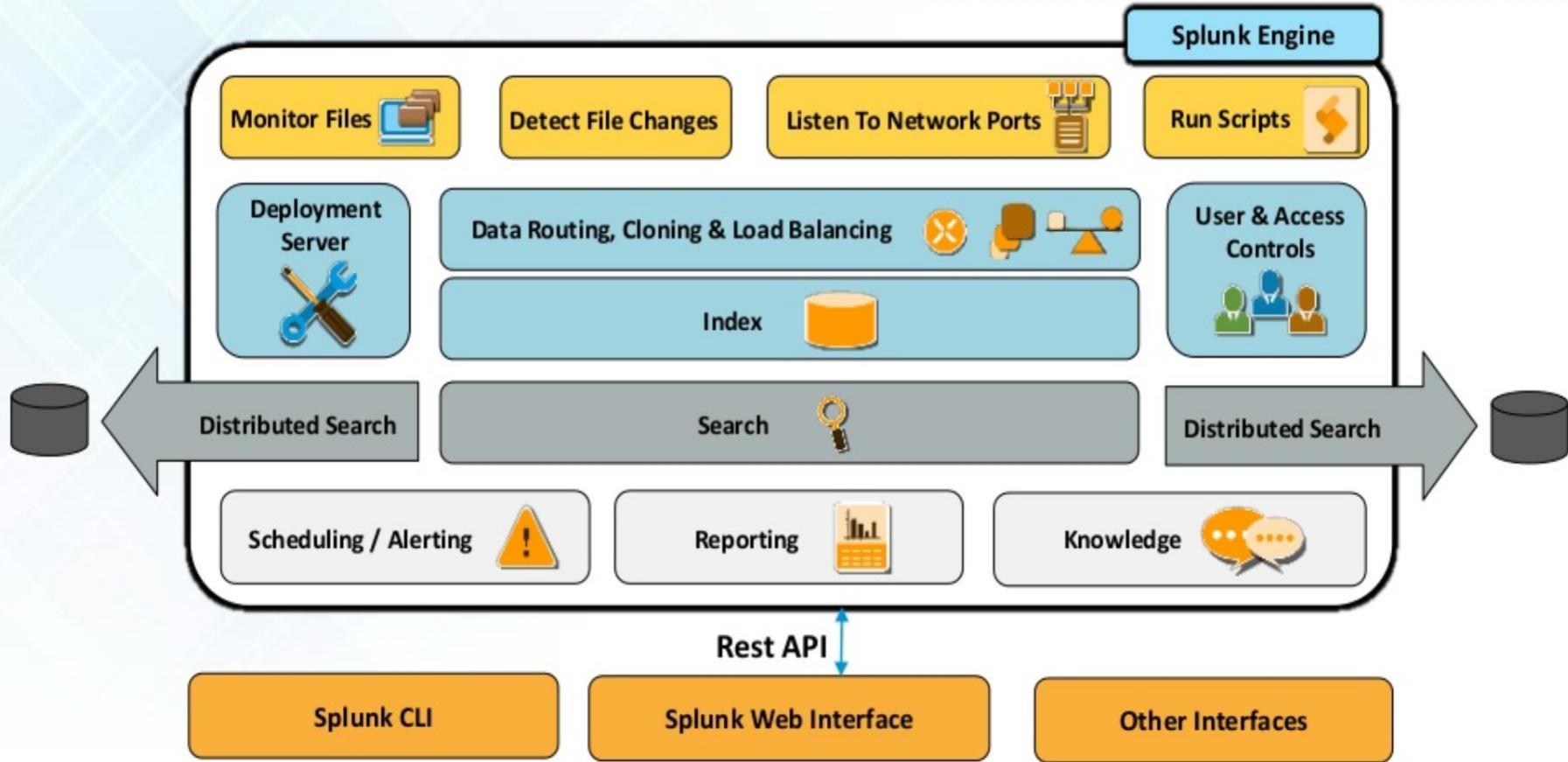
- Splunk is a one-stop solution as it automatically pulls data from various sources and accepts data in any format such as .csv, json, config files, etc.
- Splunk is the easiest tool to install and allows functionality like: searching, analyzing, reporting as well as visualizing machine data.
- It has a huge market in the IT infrastructure and business.
- Many big players in the industry are using Splunk such as Dominos, Adobe, Bosch, Vodafone, Coca-Cola etc.



- Splunk collects data in real-time from multiple systems.
- It accepts data in any form, example- log file, .csv, json, config etc.
- Splunk can pull data from database, cloud and any other OS
- It analyze and visualize the data for better performance.
- Splunk give alerts/ event notifications.
- Provides real-time visibility.
- It satisfies industry needs like horizontal scalability (using many systems in parallel)



# Splunk Architecture





# Splunk Architecture

- Splunk CLI/ splunk web interface or any other interface interacts with the search head.
- This communication happens via Rest API.
- It can be used to search head to make distributed searches, setup knowledge objects for operational intelligence, perform scheduling/alerting and create reports or dashboards for visualization.
- It can also be used to run scripts for automating data forwarding from remote Splunk forwarders to pre-defined network ports.
- After that you can monitor the files that are coming at real time and analyze if there are any anomalies and set alert/ reminders accordingly.
- It can be used to perform routing, cloning and load balancing of the data that is coming in from the forwarder, before they are stored in an indexer.
- It can also be used to create multiple users to perform various operations on the indexed data.

# Splunk Users



Persona	Industry Role	Activities
Administrator	network engineer, system administrator	<ul style="list-style-type: none"><li>Configures, administers, optimizes, and secures the Splunk Enterprise deployment.</li><li>Sets up user accounts and permissions.</li><li>Gets data into Splunk Enterprise.</li></ul>
Knowledge Manager	data analyst, system administrator	<ul style="list-style-type: none"><li>Oversees knowledge object creation, normalization, and usage across teams, departments, and deployments.</li><li>Gets the data into Splunk Enterprise, or works with the administrator to do so.</li><li>Creates and shares data models.</li></ul>
Search User	data analyst, IT professional, network engineer, security analyst, system administrator	<ul style="list-style-type: none"><li>Uses Search to investigate server problems, understand configurations, monitor user activities, and troubleshoot escalated problems.</li><li>Builds reports and dashboards to monitor the health, performance, activity, and capacity of their IT infrastructure.</li><li>Identifies patterns and trends that are indicators of routine problems.</li></ul>
Pivot User	business professional, data analyst, executive, IT professional, manager, system administrator	<ul style="list-style-type: none"><li>Uses Pivot to build reports based on data models created by the Knowledge Manager.</li><li>Creates reports and dashboards to monitor their businesses.</li><li>Identifies trends in the health and performance of their businesses.</li></ul>
Developer	system integrator, professional developer	<ul style="list-style-type: none"><li>Integrates data and functionality of applications with Splunk Enterprise.</li><li>Builds Splunk apps and add-ons with custom dashboards and data visualizations.</li></ul>



# Splunk vs Other Tools

Features	Splunk	Sumo Logic	ELK
Searching	✓	✓	Only possible with Integrations
Analysis	✓	✓	Only possible with Integrations
Visualization Dashboard	✓	✓	Only possible with Integrations
SaaS Setup	✓	✓	✓
On Premise Setup	✓	✗	✓
Input any data type	✓	✓	Needs plugins
Plugins & Integration	✓	✓	✓
Customer Support	✓	Available; but not proficient	Available; but not proficient
Documentation & Community	✓	✗	✓



# Case: Domino's



# Case: Domino's



## Interactive map

- Shows all the orders coming from across US in real time
- Brought employee satisfaction



## Real-time Feedback

- Employees constantly see what customers are saying
- Helped them understand customer expectations



## Dashboard

- Used to keep score and set targets
- Compare performance with previous week



## Payment Process

- Analysed the speed of different payment modes
- Determine error free payments modes



## Promotional Support

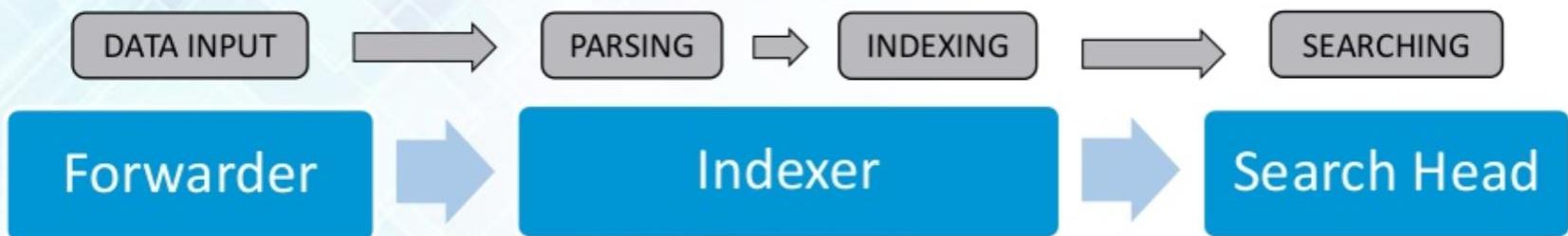
- Track how various promotional offers are impacting in real-time
- Initially, determining the impact of promotions took almost a day



## Performance Monitor

- Monitor the performance of Domino's in-house developed point of sales systems

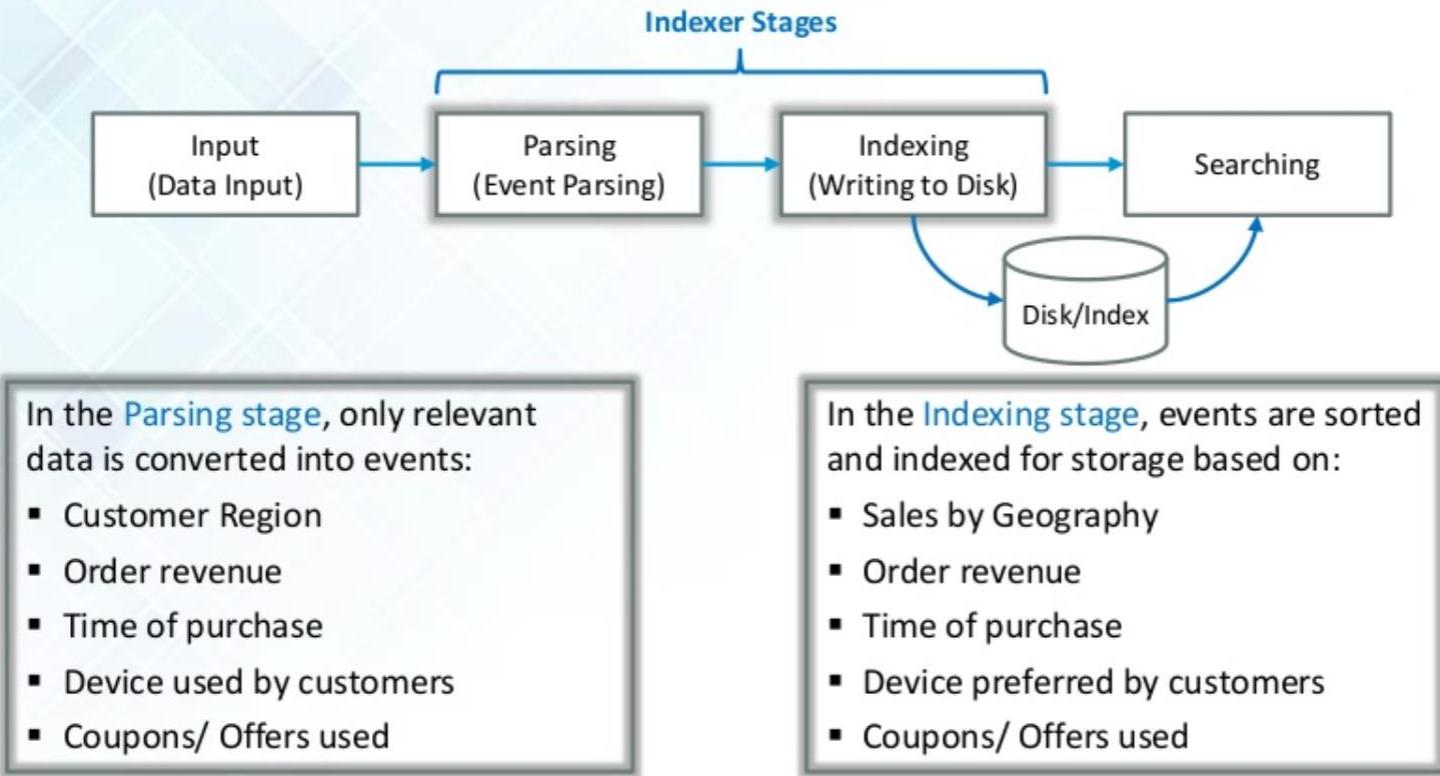
# Splunk Components



- Collects the data from remote machines
- Forwards the data to the Indexer in real-time
- Processes the incoming data in real-time
- Stores & Indexes the data on disk
- End users interact with Splunk through Search Head
- Allows users to do searching, analysis & visualization



# Indexer for Data Storage and Processing



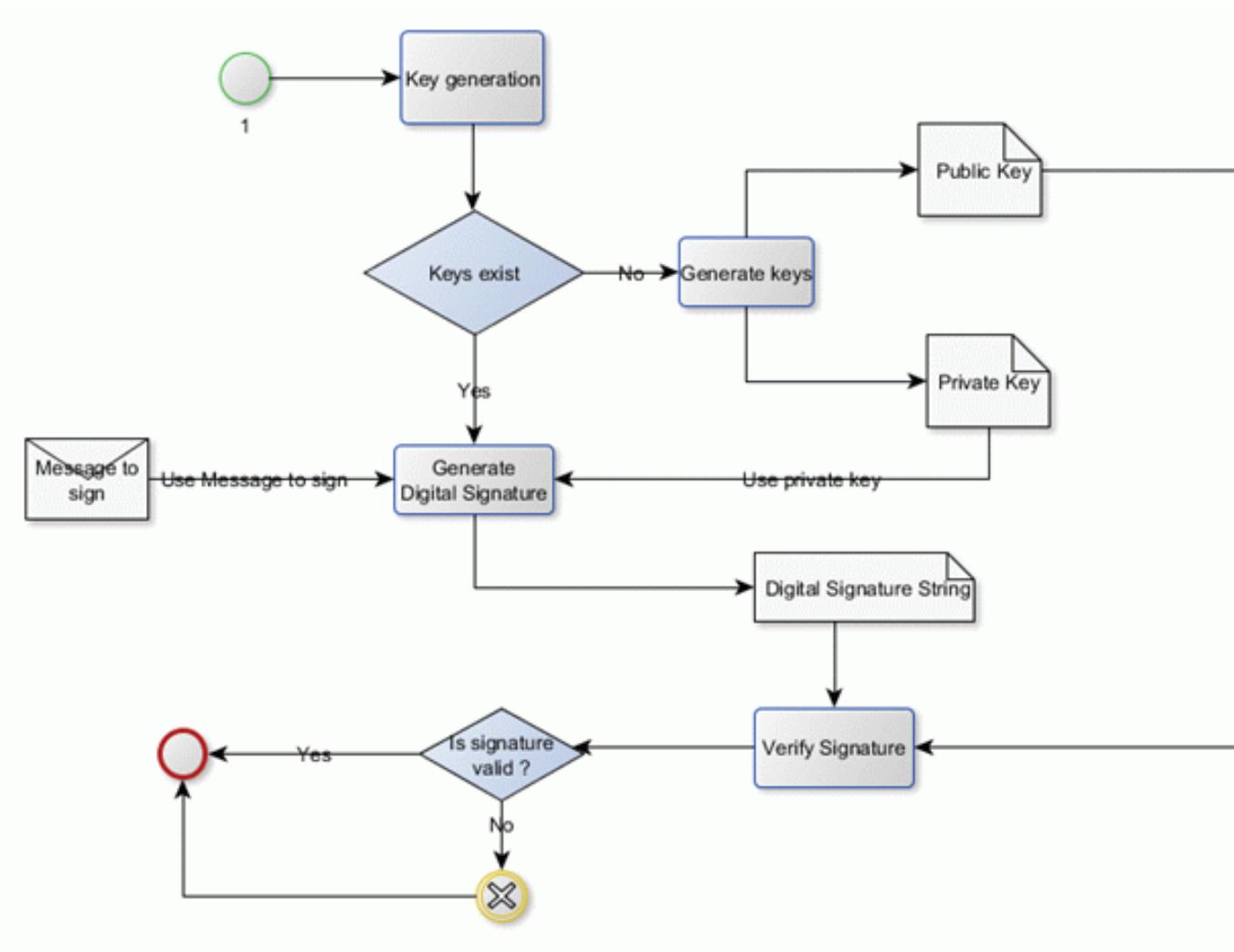
\* The details mentioned in this slide are representative in nature and data present might not be accurate.

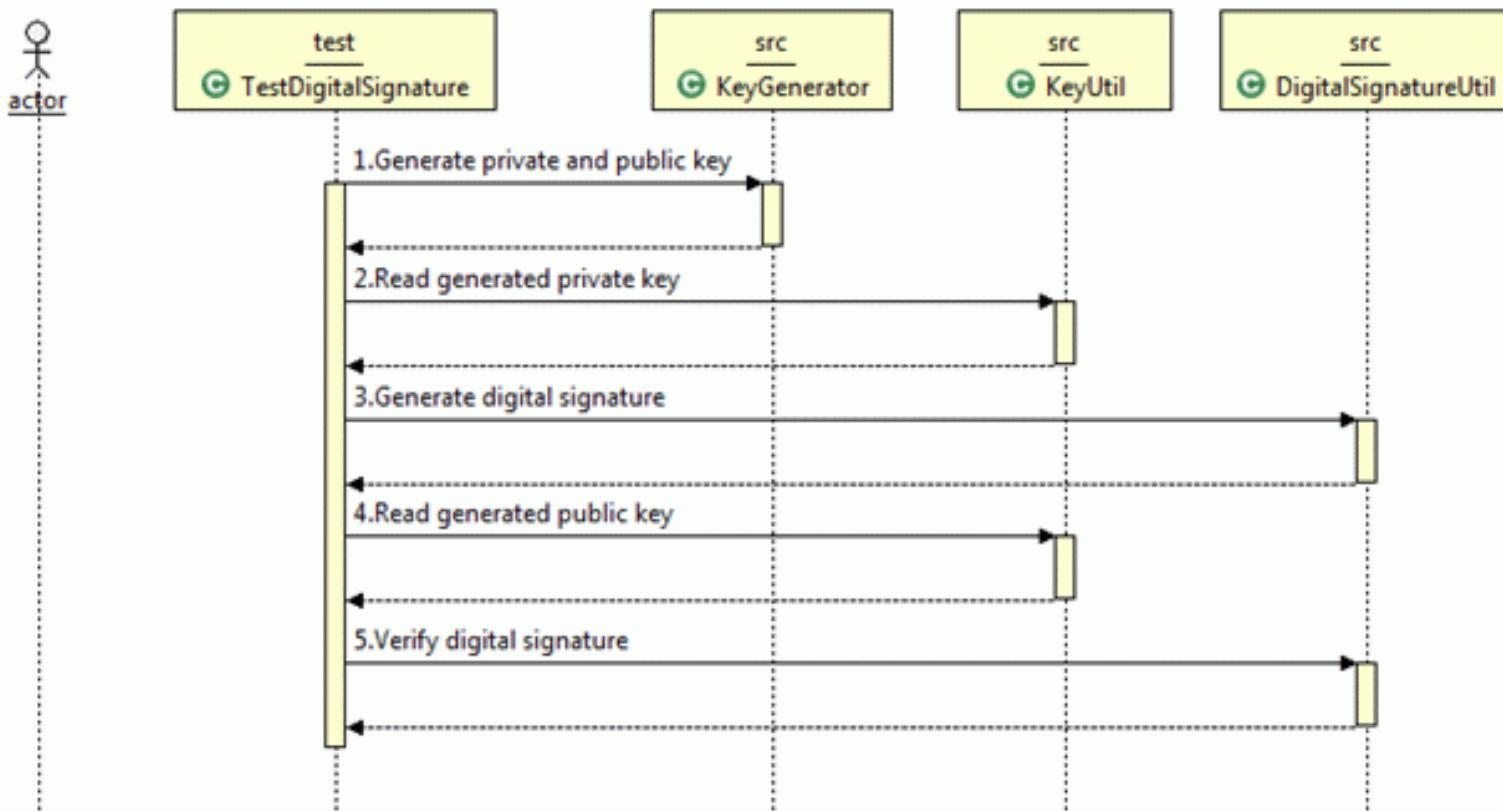


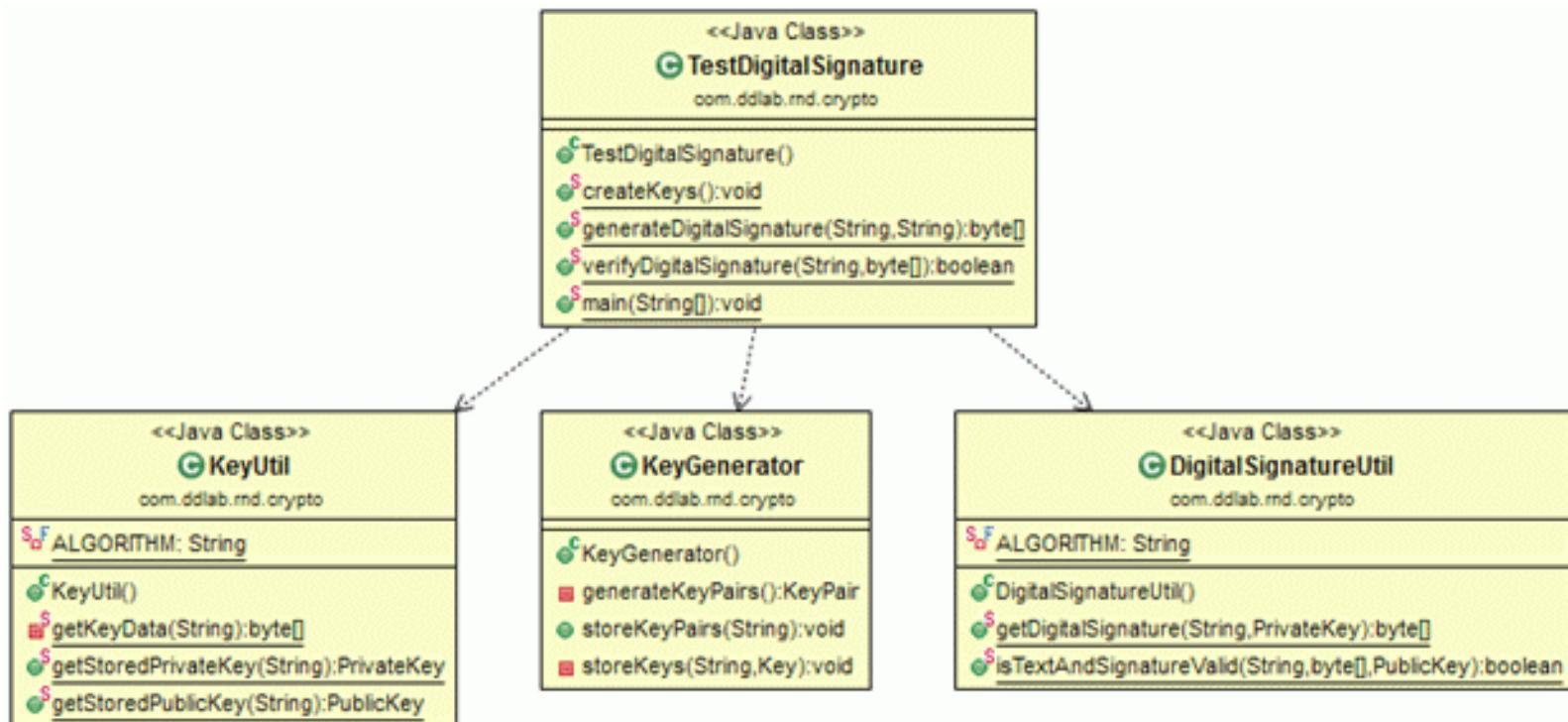
# Digital Signature

---

- Create a pair of keys called Private and Public keys
- Use the private key and your text message to generate a digital signature
- Send the public key, actual text message and digital signature separately to the destination
- Use the public key, text message and digital signature to verify the message
- If the verification is successful then process the message otherwise throw an exception and discard the message.







# Questions



# Module Summary

---

- Spring Integration Framework.
- Message, Channel and Adapter
- Understood the different Component Integration
- Understood the Event-Driven Architecture

