

# 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

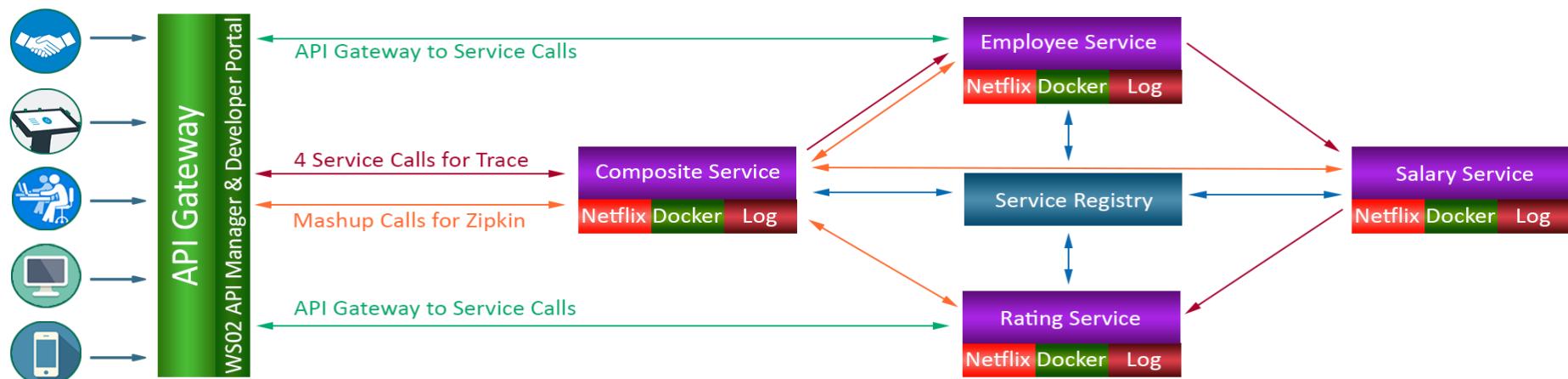
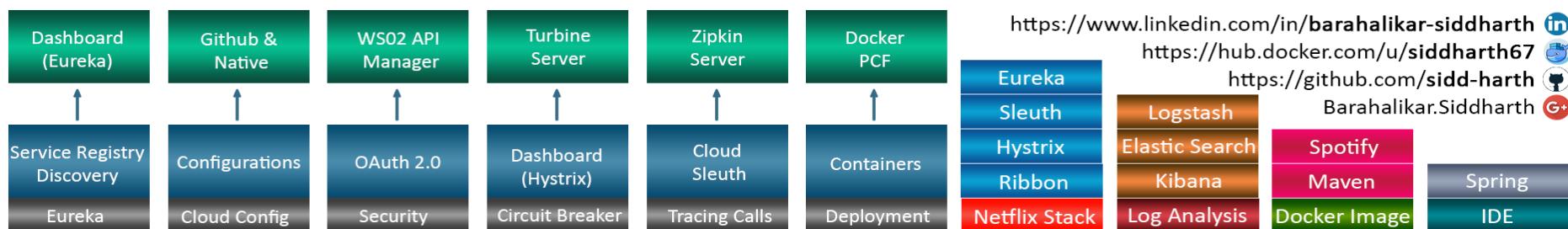
---

## 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

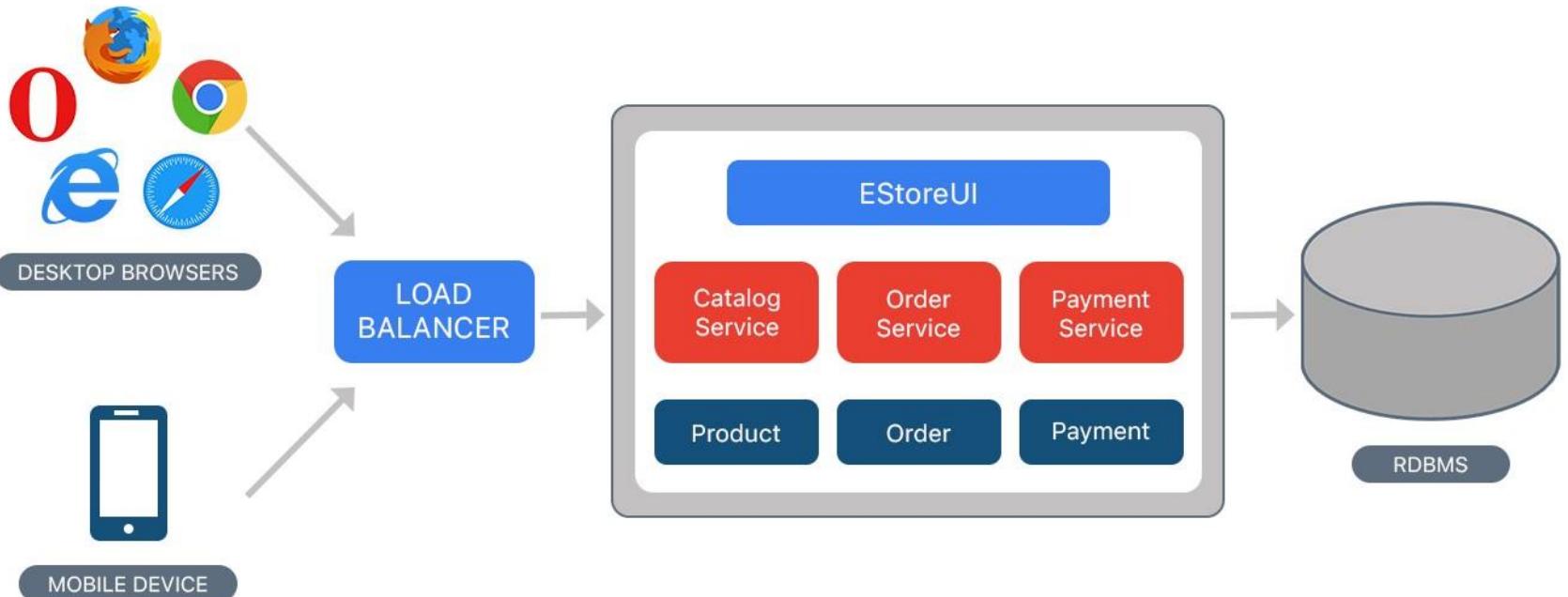
---

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

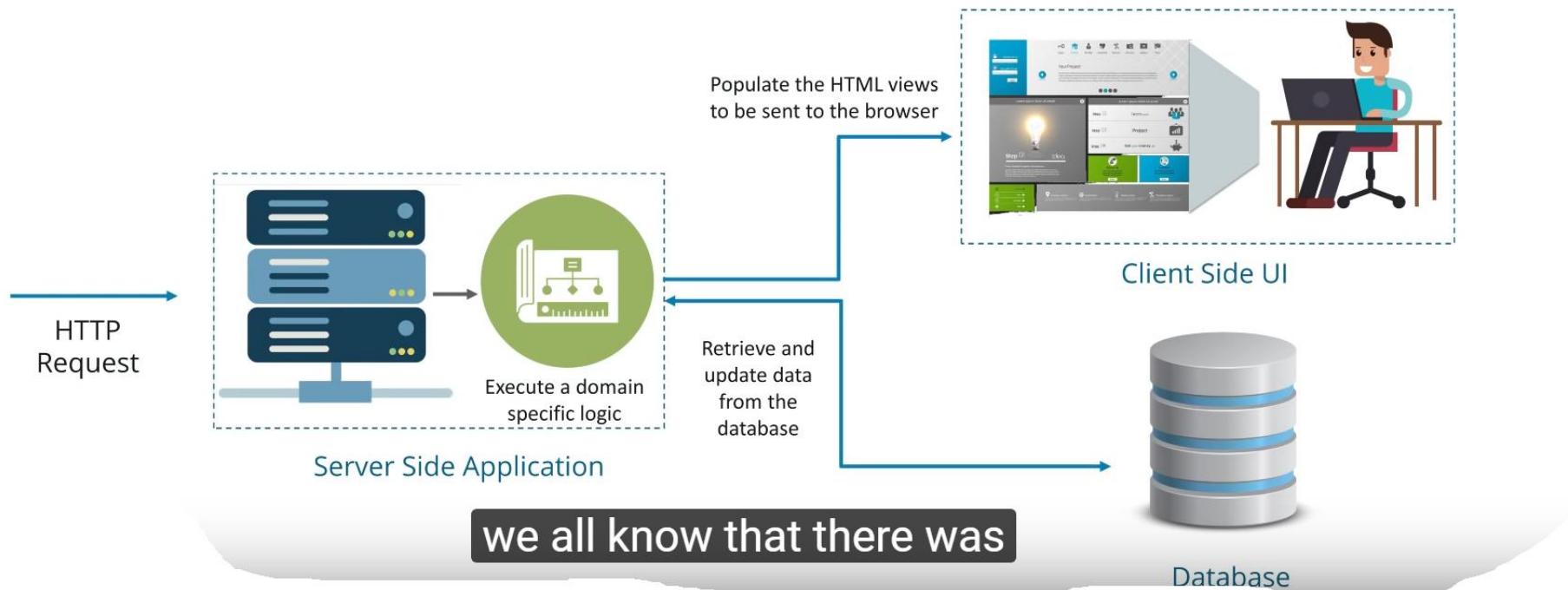
---

- 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



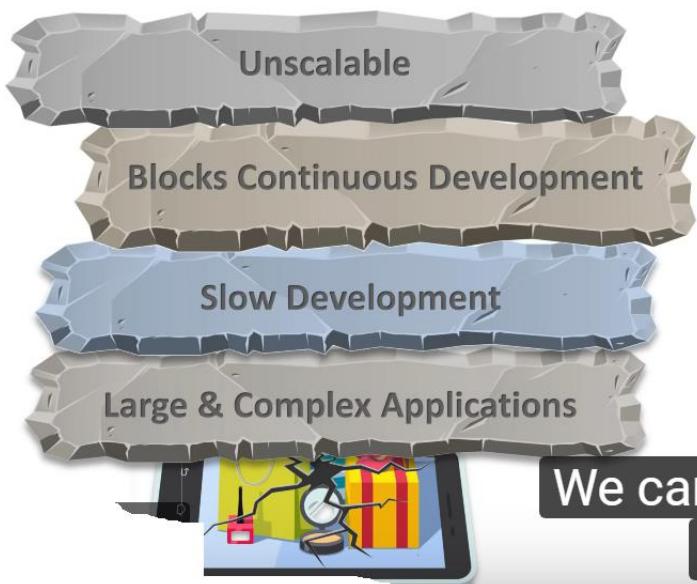


# Monolithic Architecture



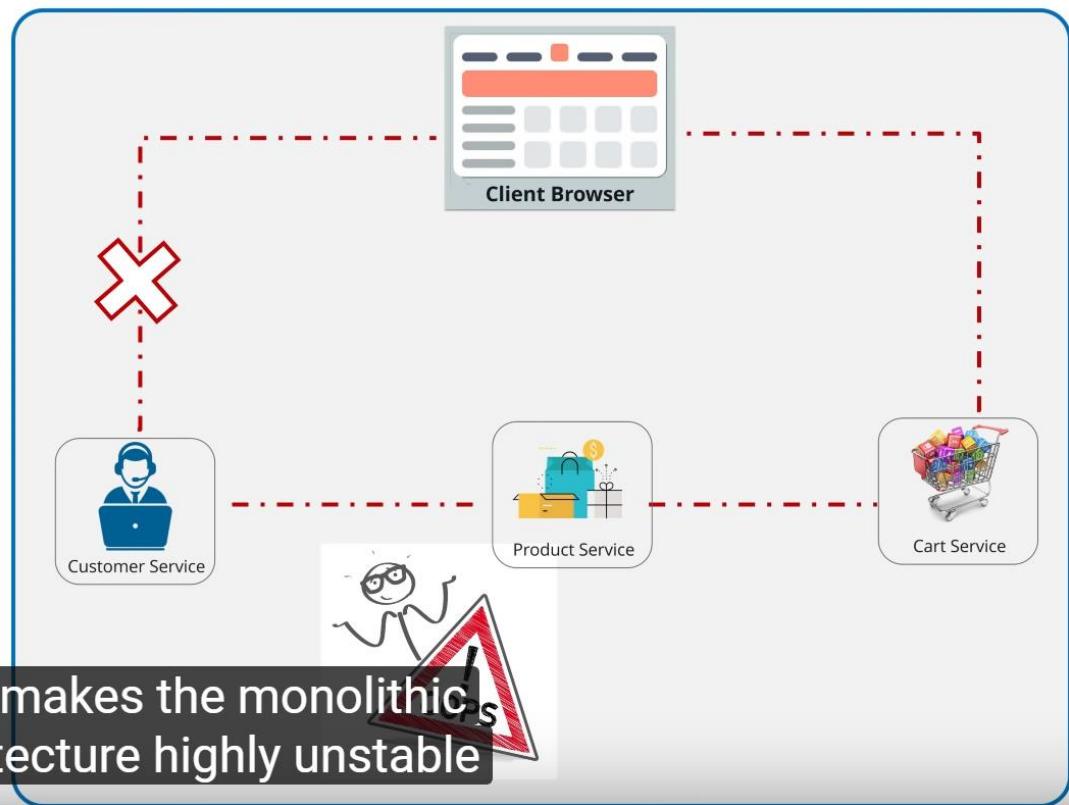


# Monolithic Architecture



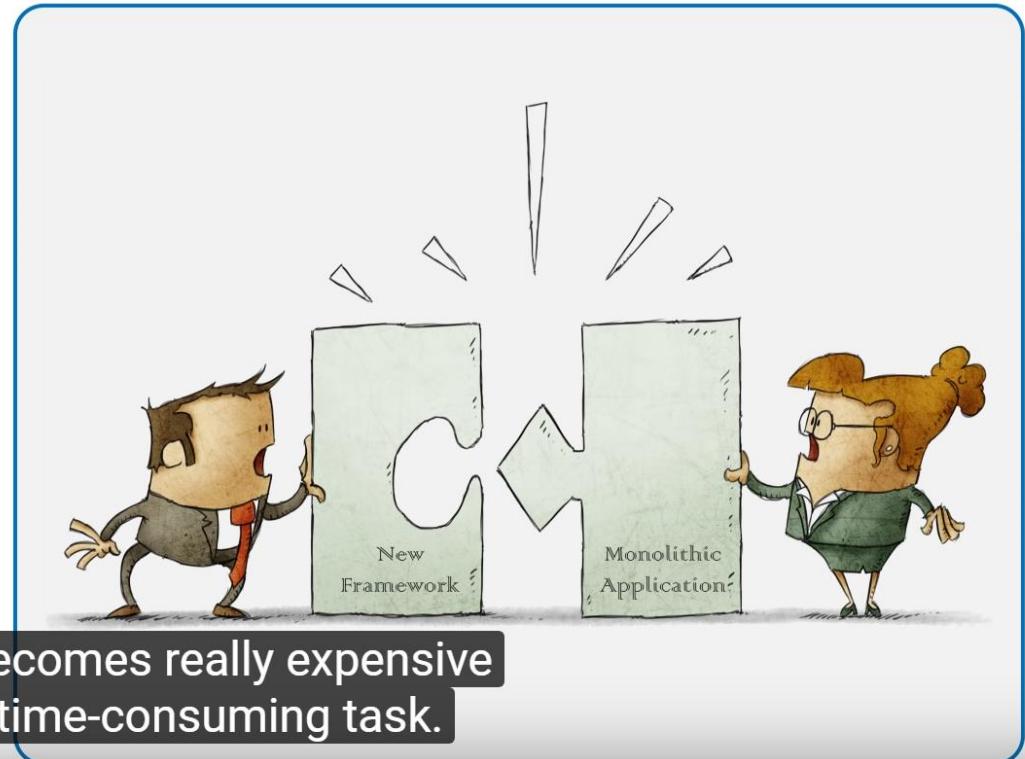


# Monolithic Architecture





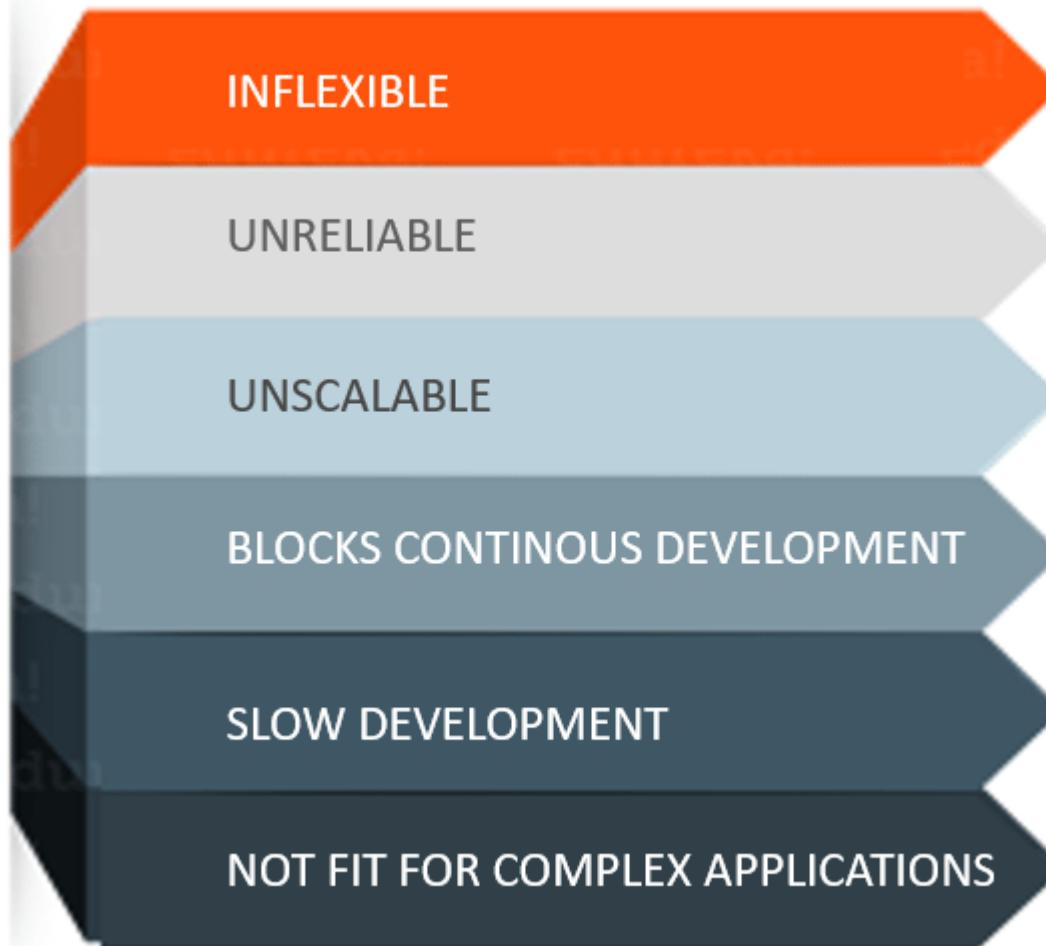
# Monolithic Architecture



So it becomes really expensive  
and time-consuming task.



# Monolithic Architecture



# Micro Service

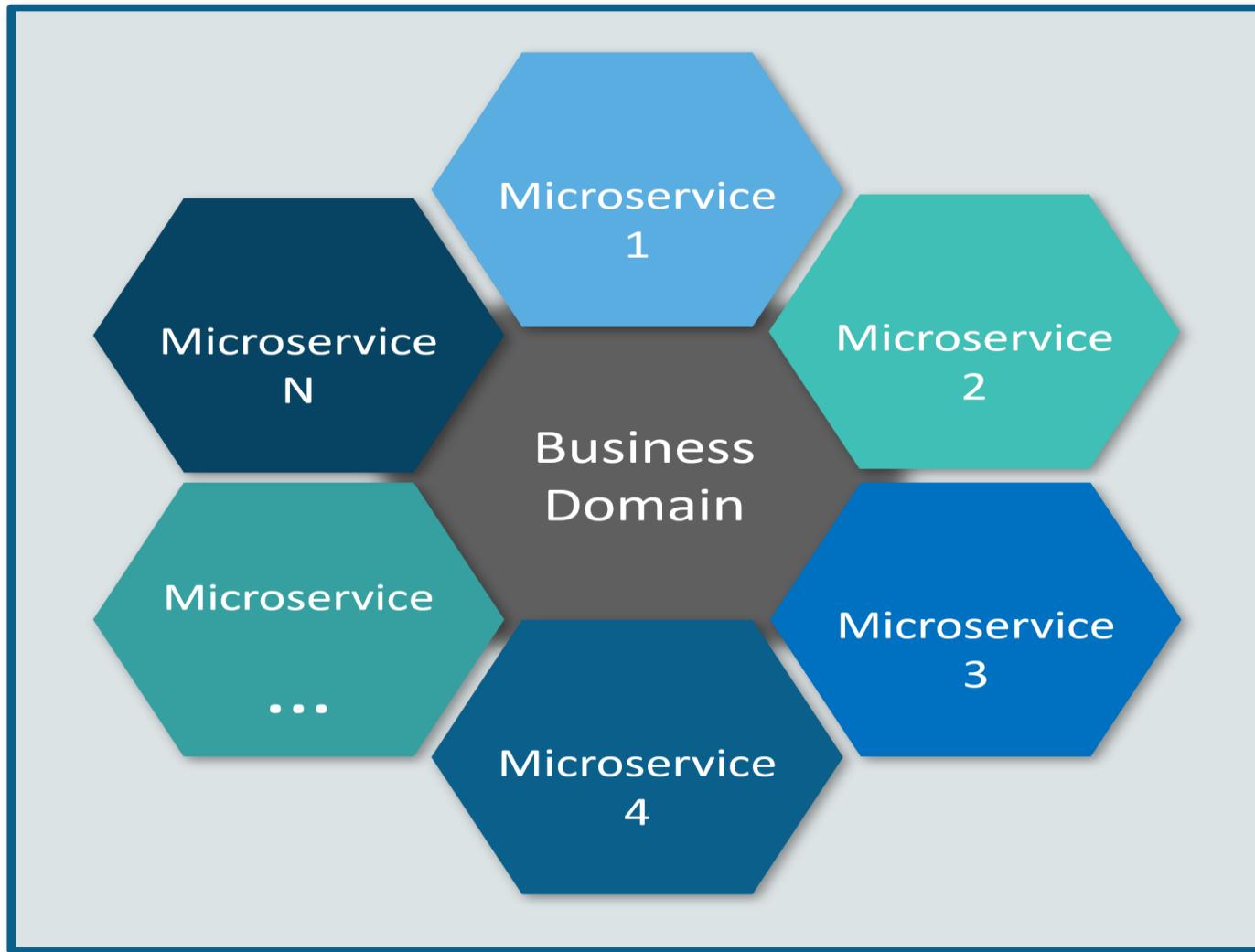


*Small autonomous services that work together - Sam Newman*

*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*



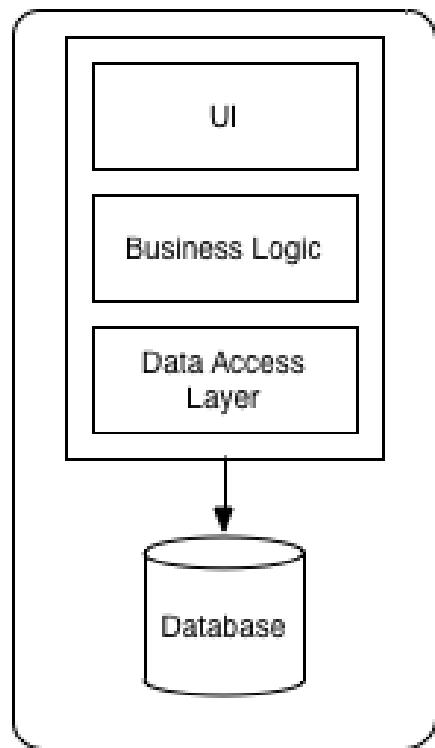
# Micro Service



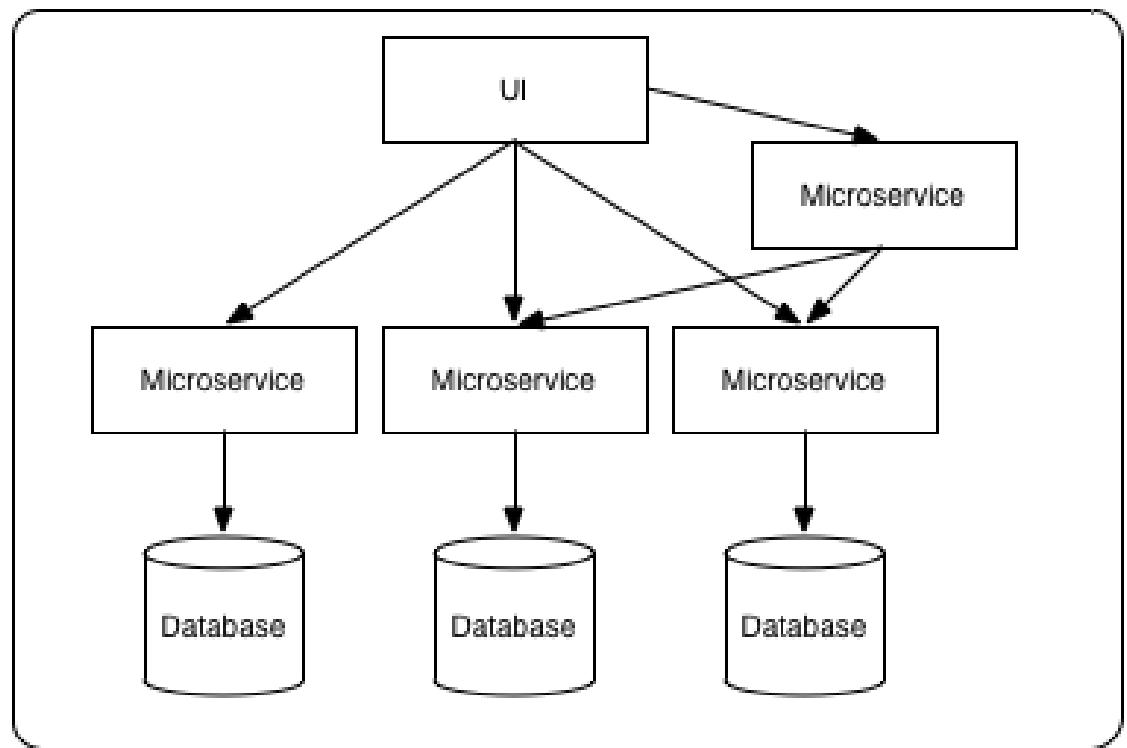
*There is a bare minimum of centralized management of these services, which may be written in different programming languages and use different data storage technologies - James Lewis and Martin Fowler*



# Microservice



Monolithic Architecture

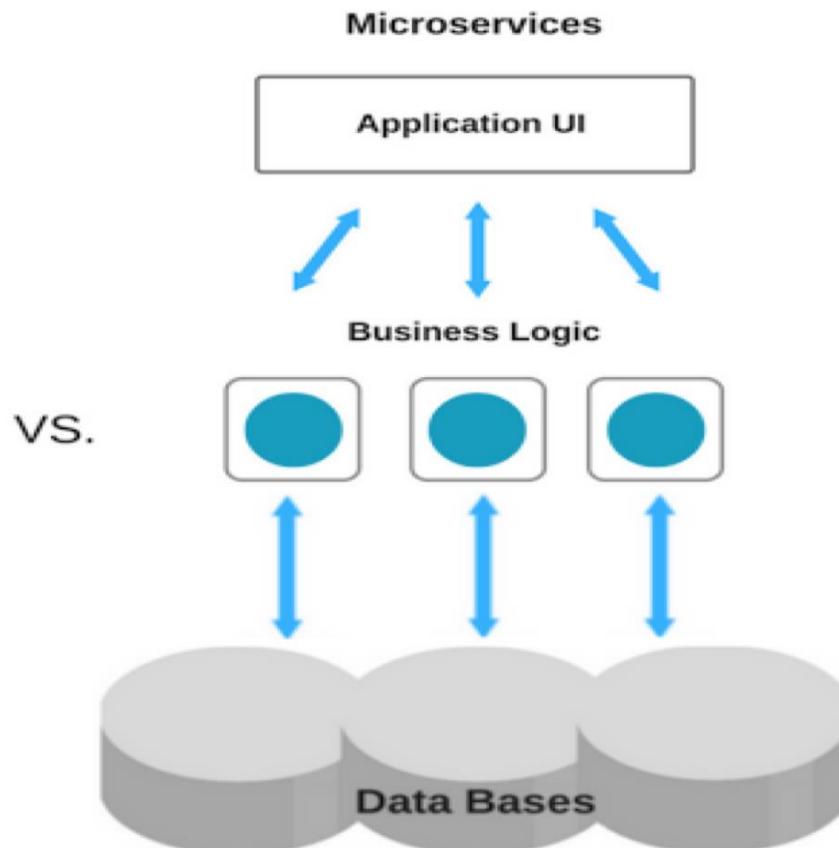
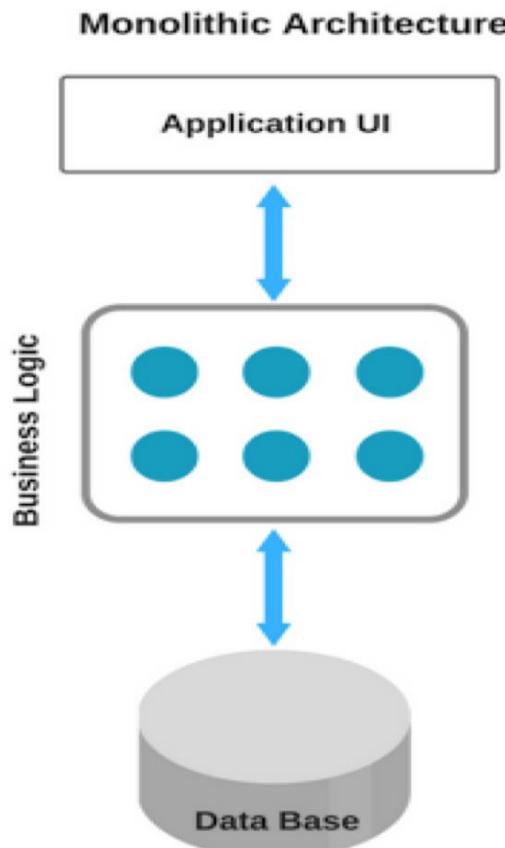


Microservices Architecture

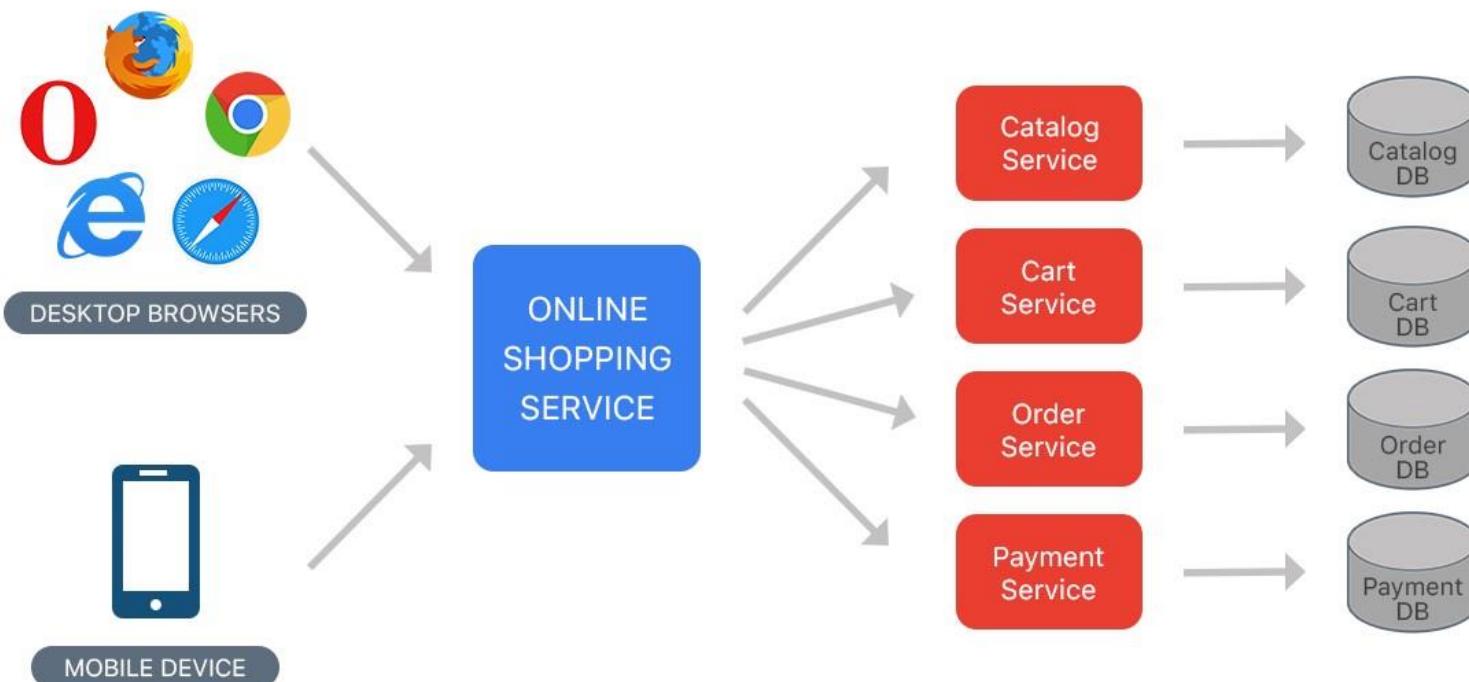
# MicroService



The difference between the monolithic and microservices architecture

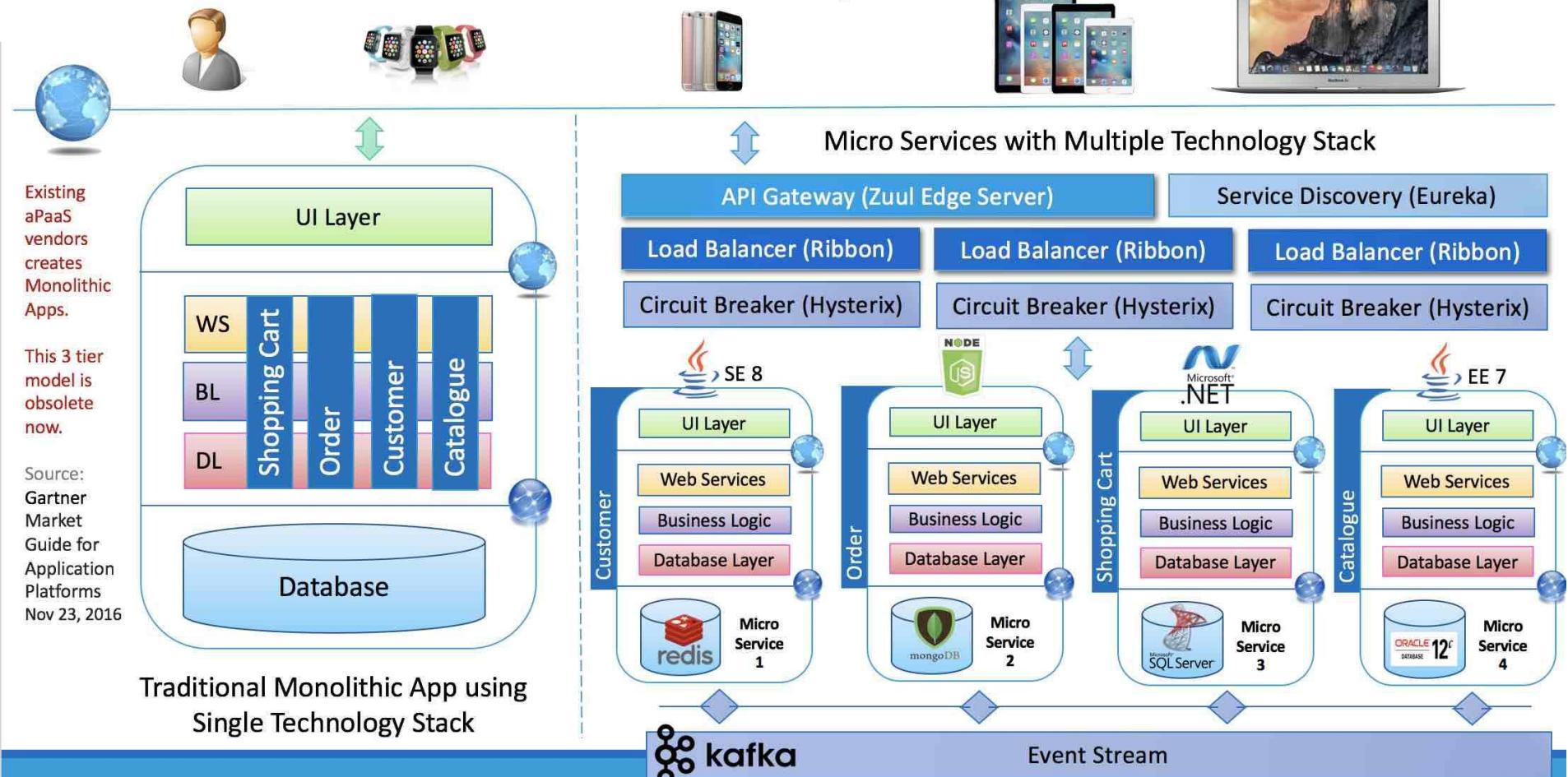


# Micro Service



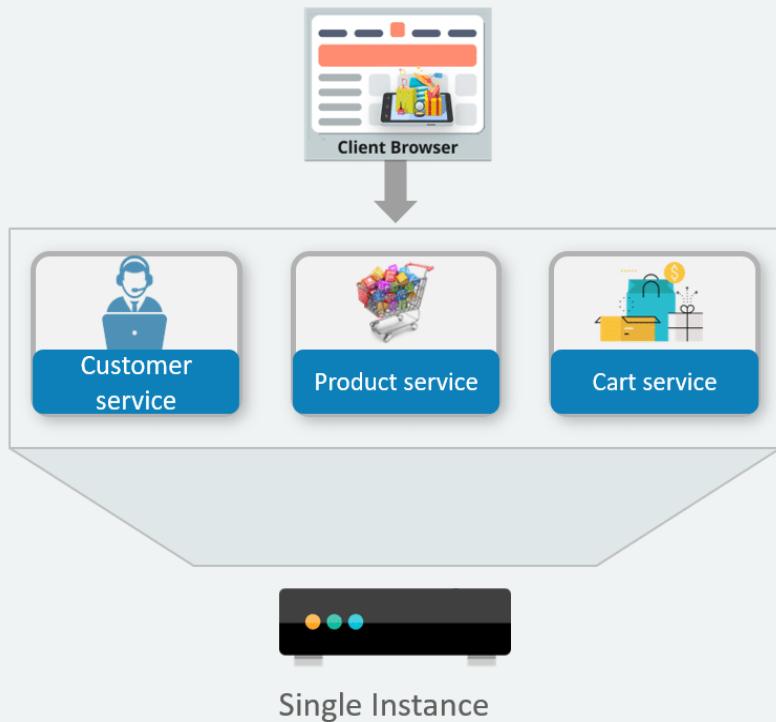


## Monolithic vs. Micro Services Example

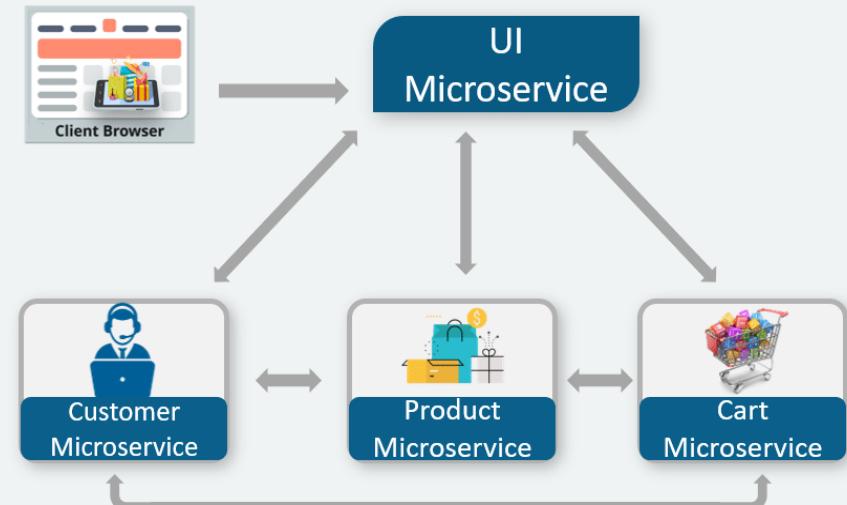




## Monolithic Architecture

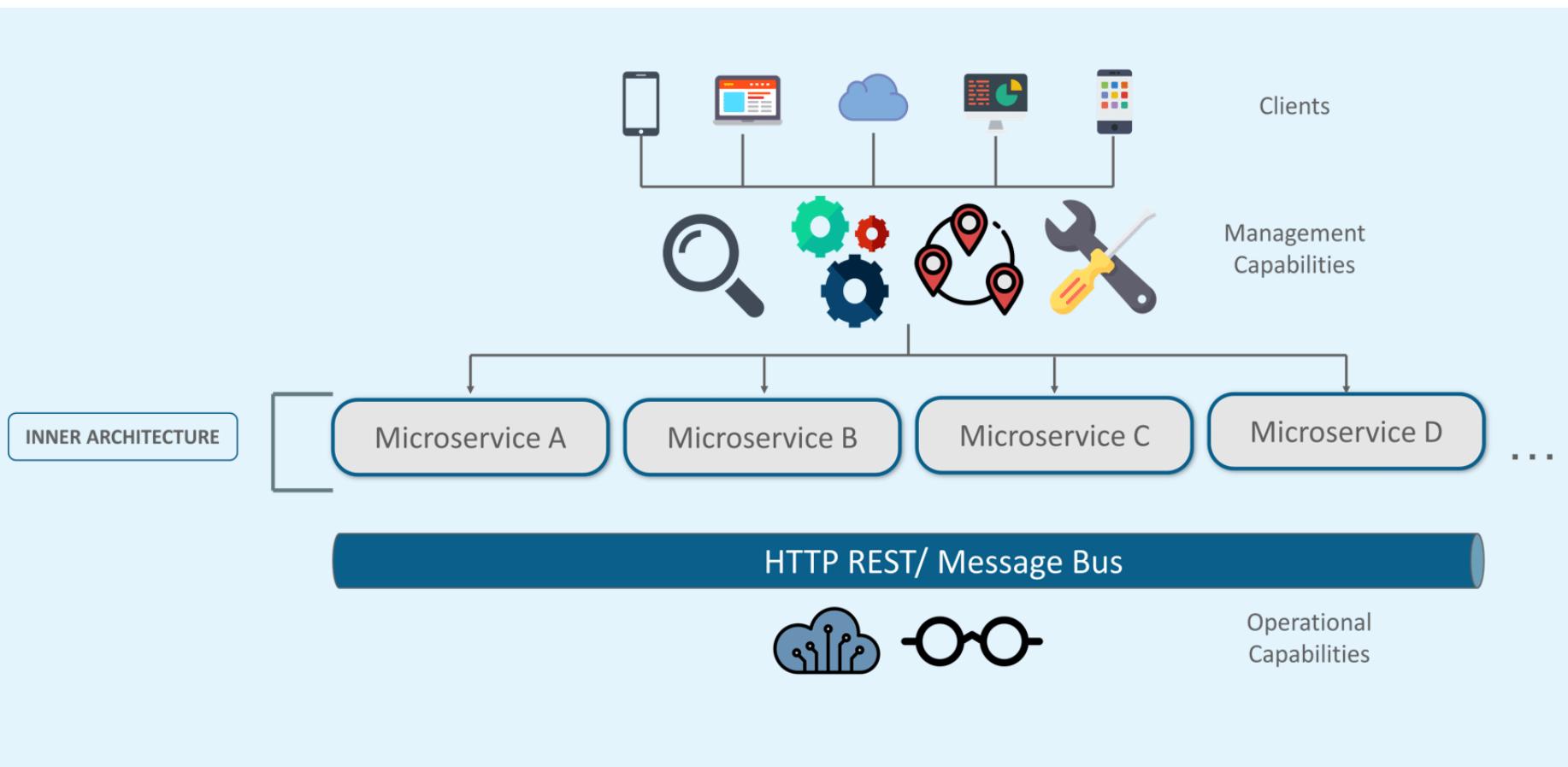


## Microservice Architecture





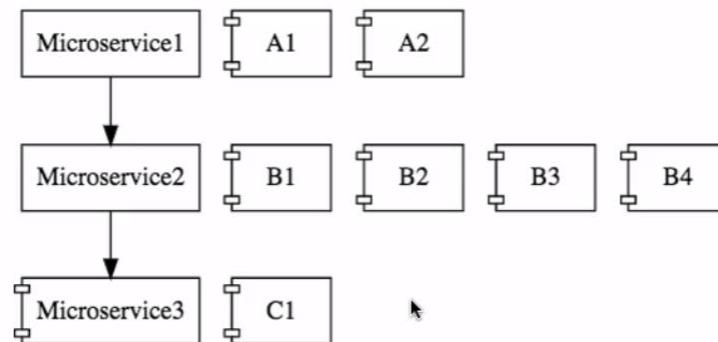
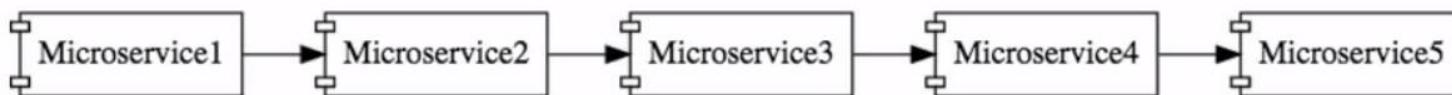
# Microservice Architecture



# 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

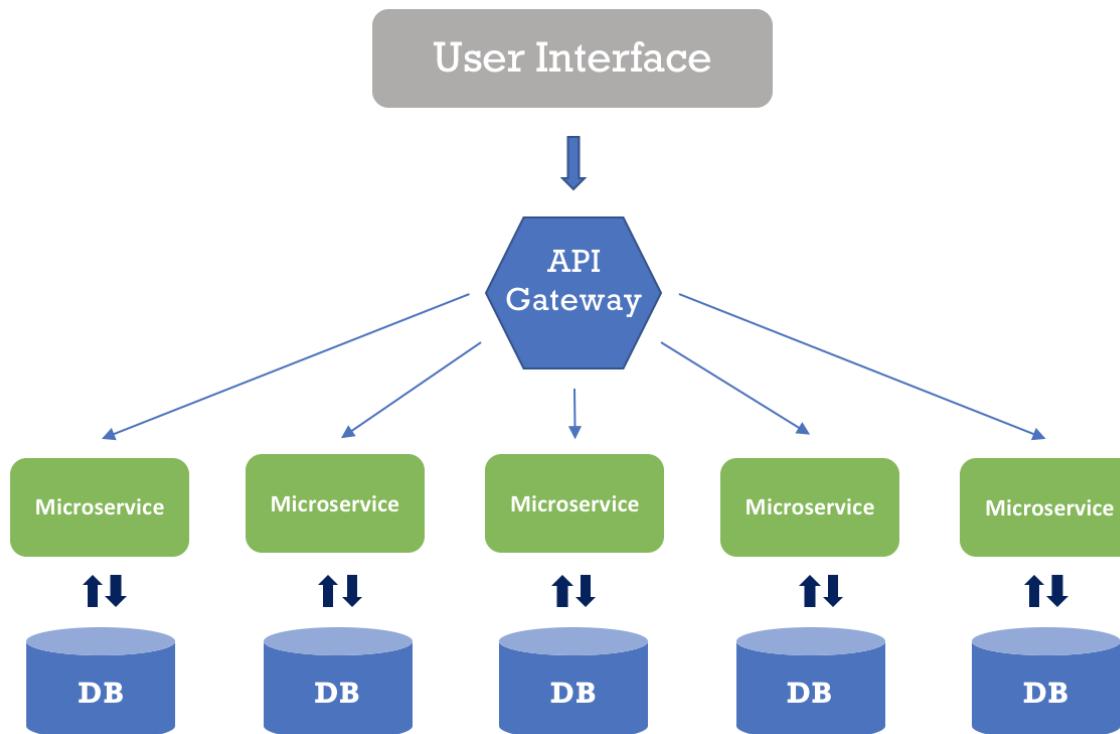
---

- 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.

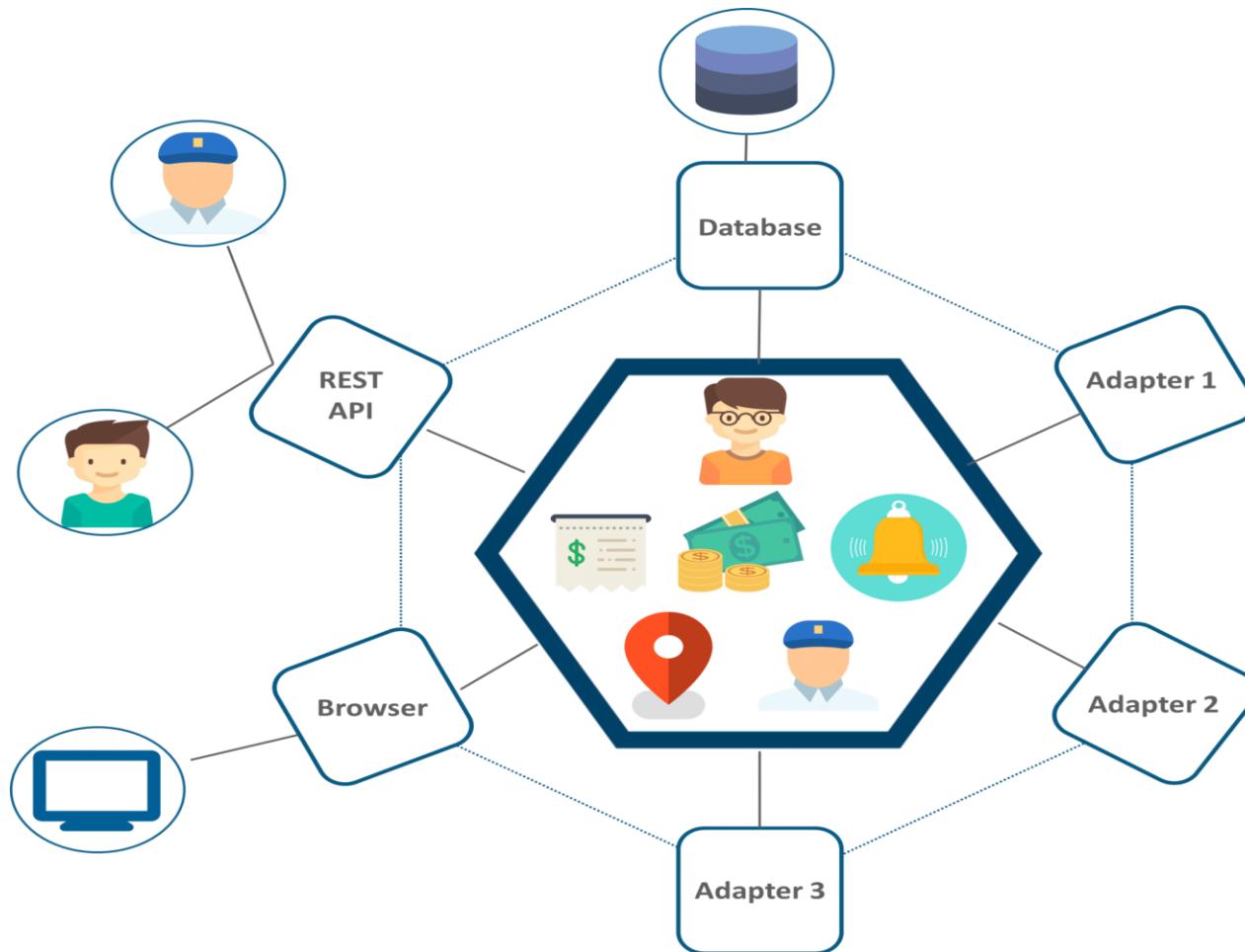


# What is Microservices

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



## UBER's Previous Architecture





## Problem Statement

---

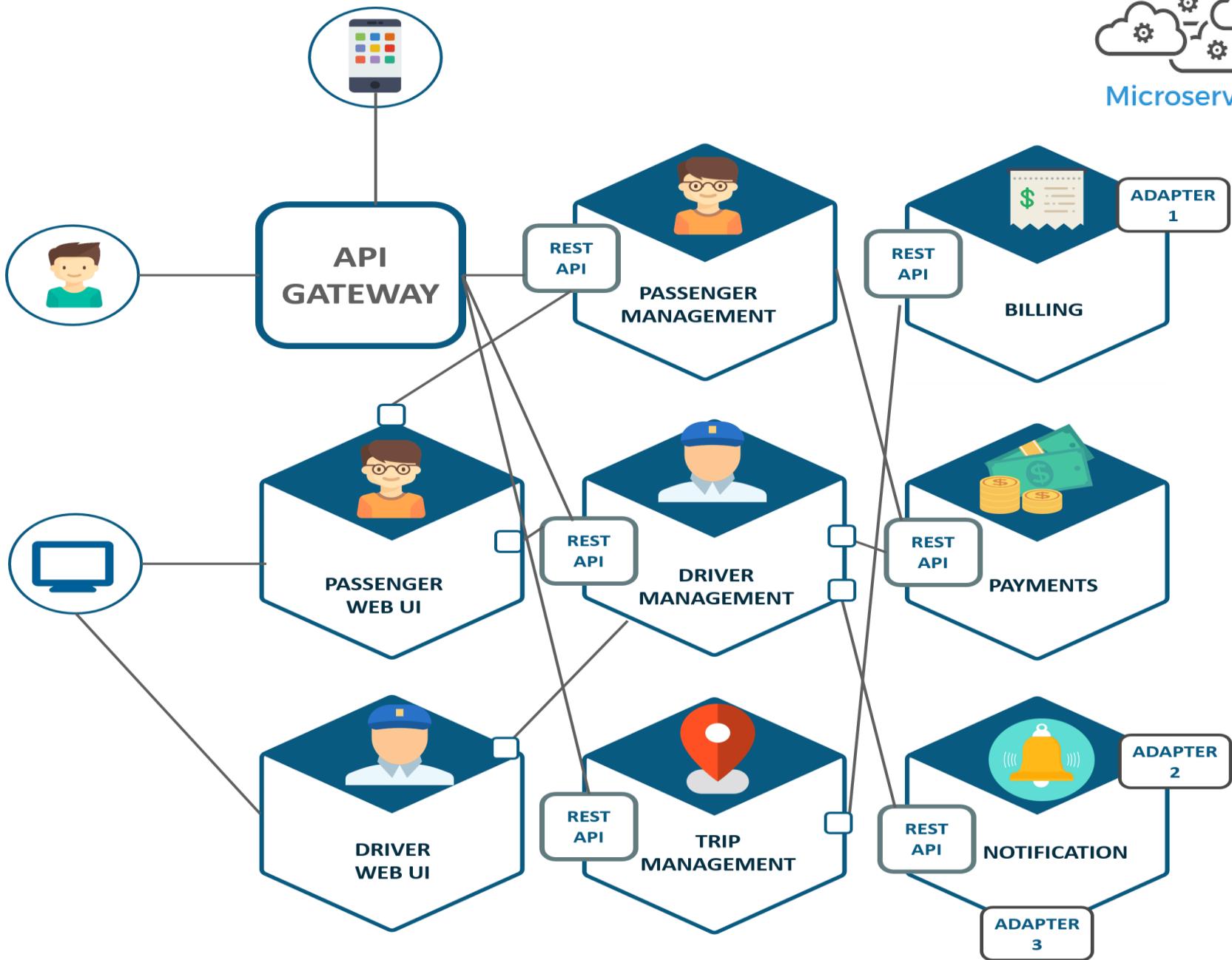
- While UBER started expanding worldwide this kind of framework introduced various challenges. The following are some of the prominent challenges
- All the features had to be re-built, deployed and tested again and again to update a single feature.
- Fixing bugs became extremely difficult in a single 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.



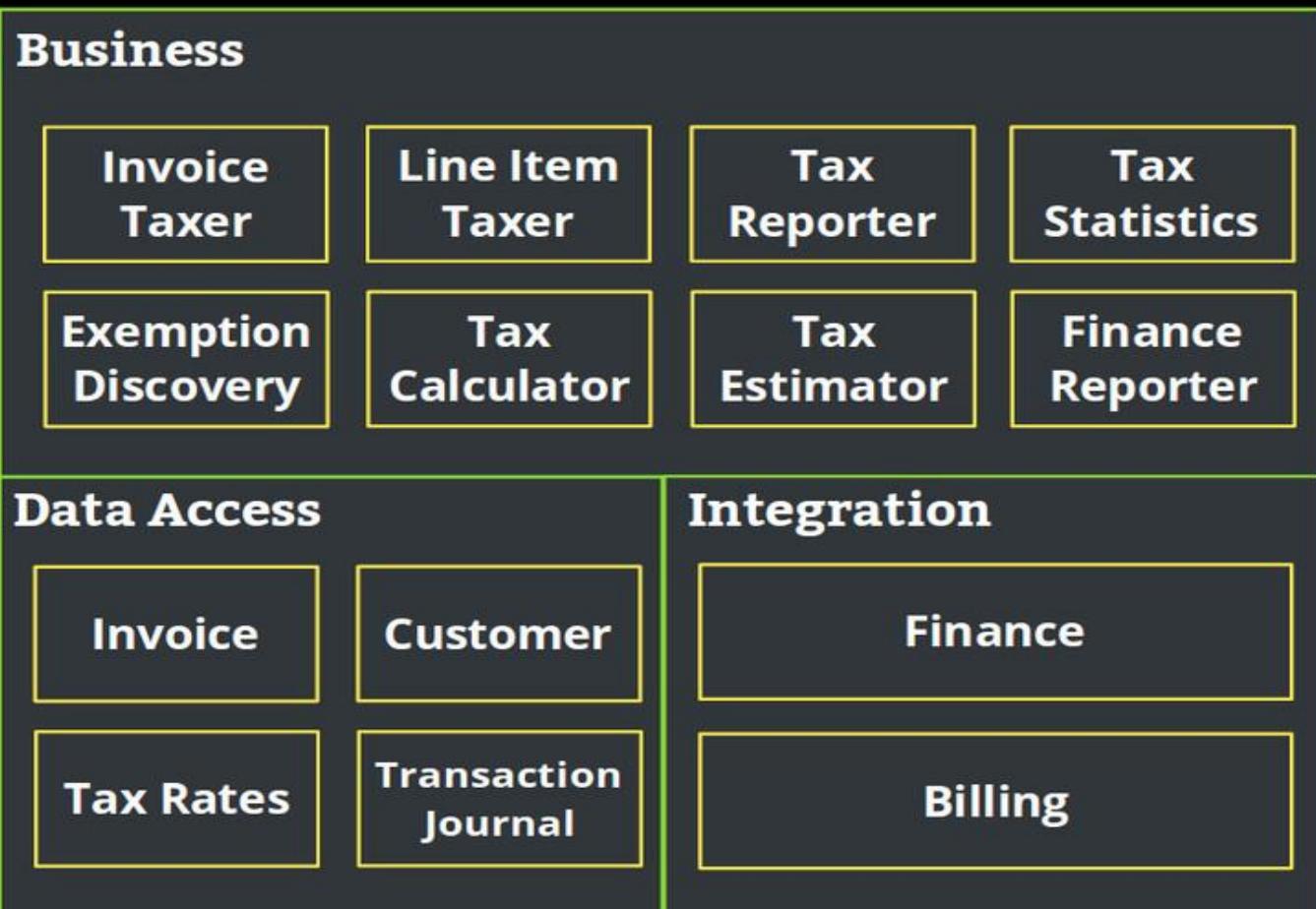


# Existing Scenarios

---

- 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.

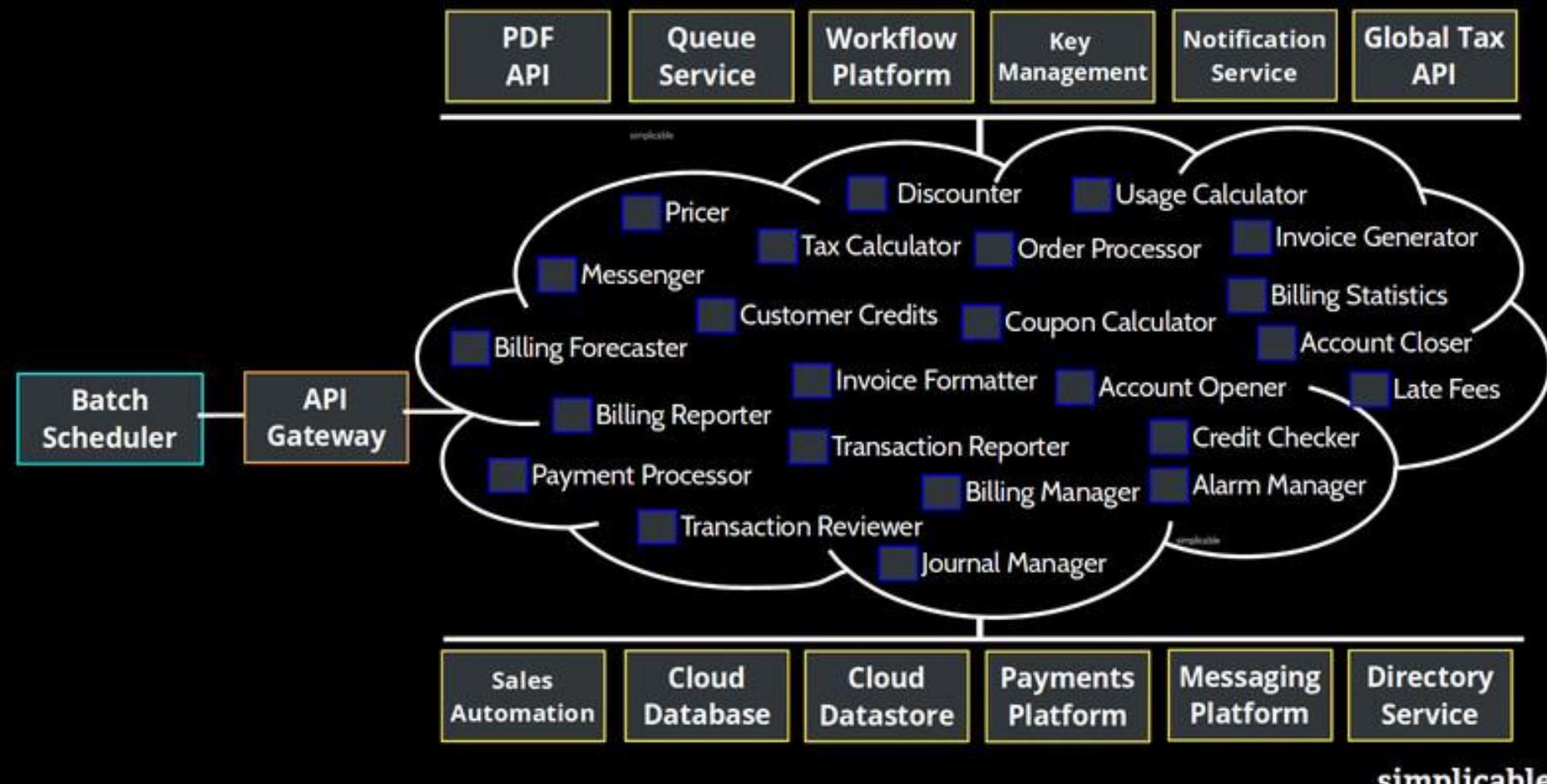
# Tax Calculator



simplicable

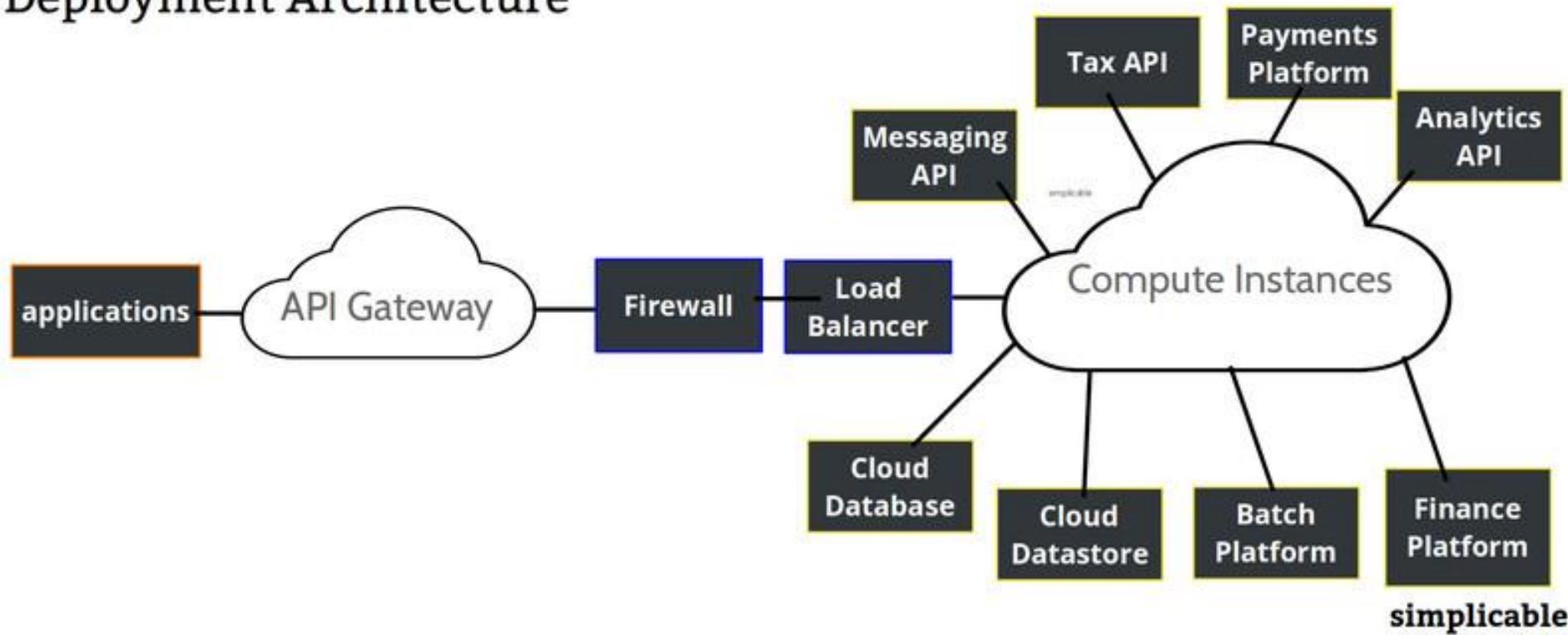


# Services Architecture





## Deployment Architecture





# Principles of Micro service

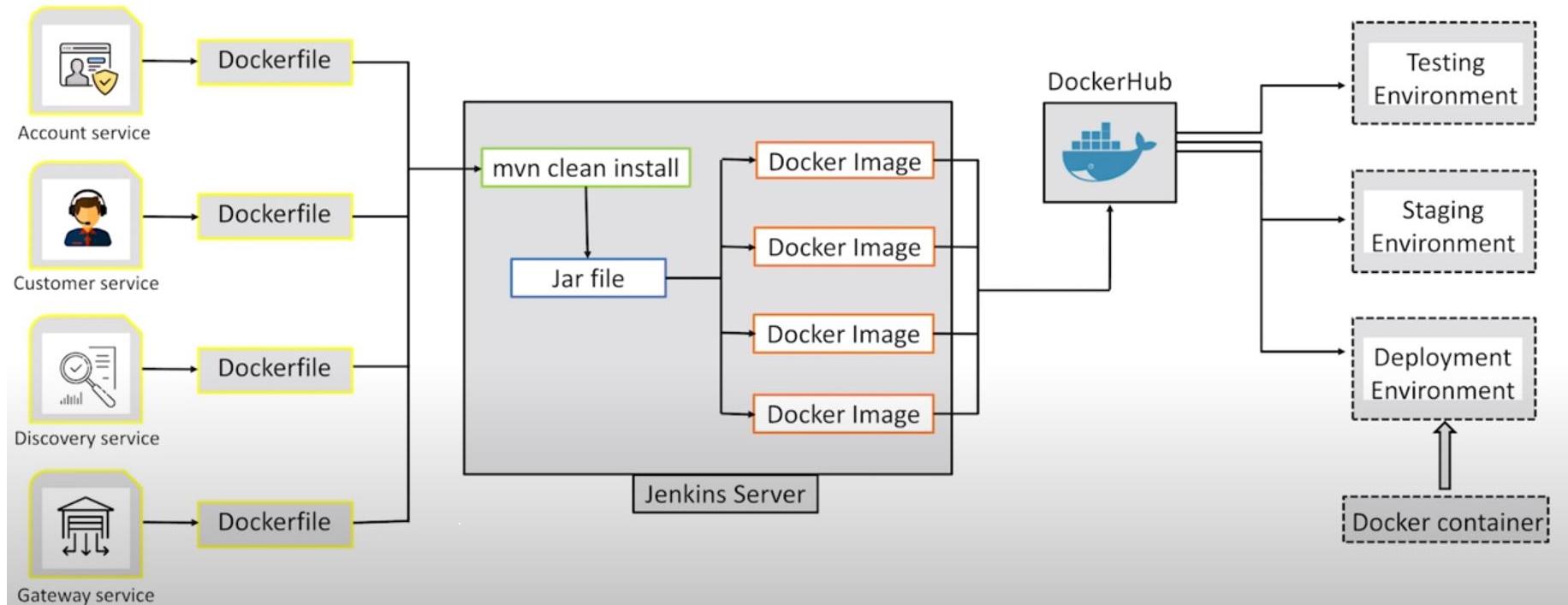
---

- 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



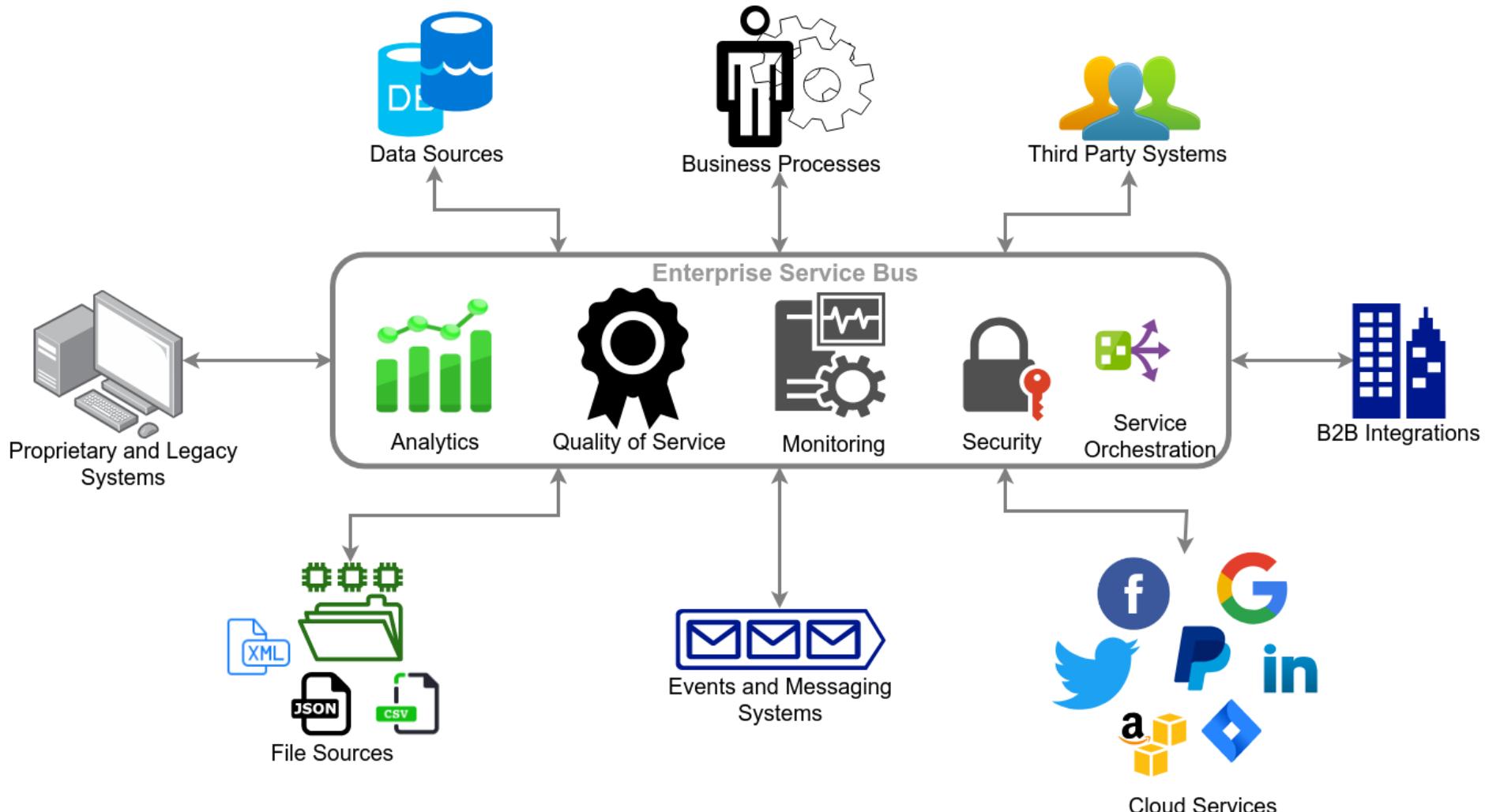
## Solution



# API Integration ESB or Microservices



# ESB for API Integration



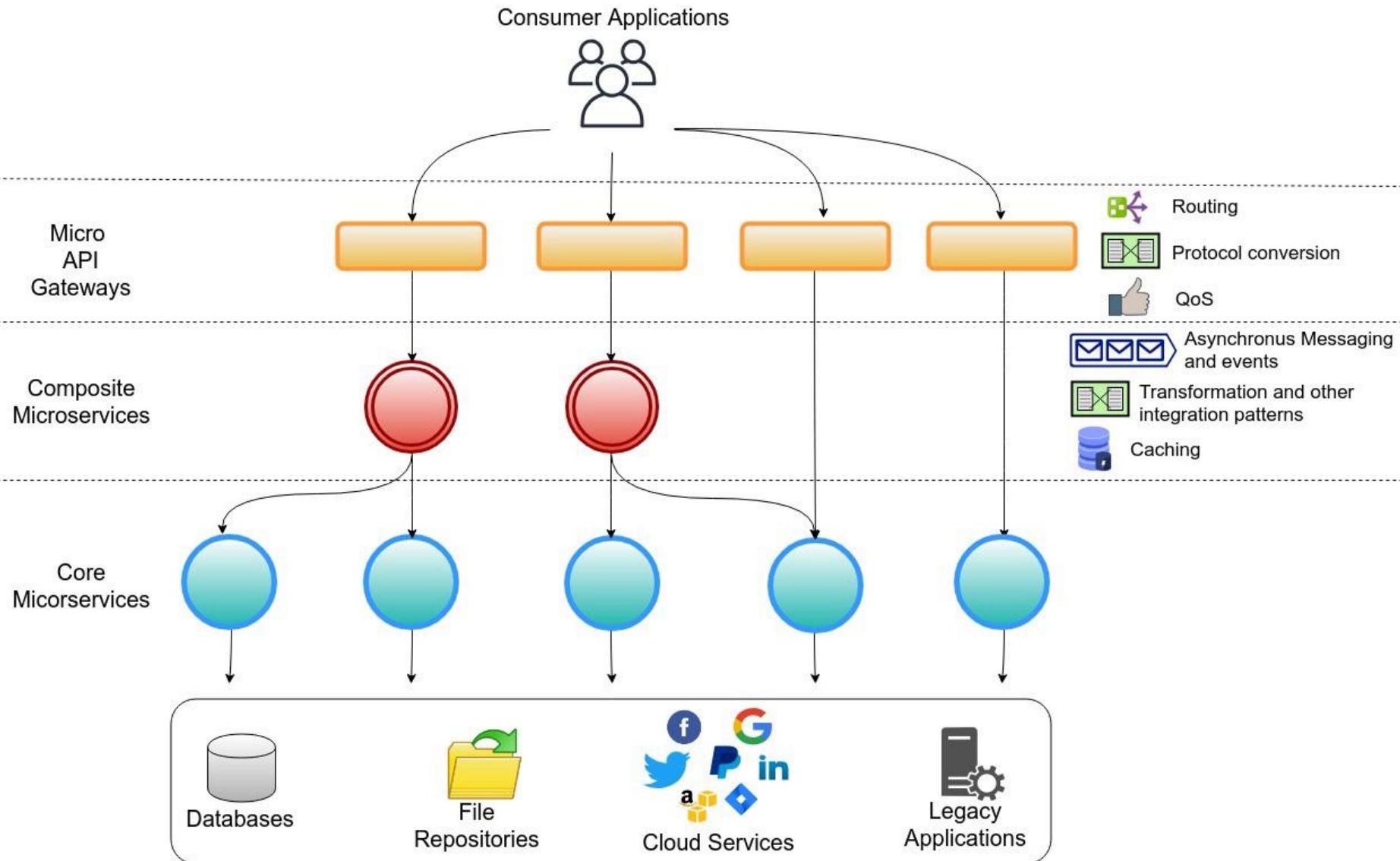
# Disadvantages of ESB



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



# Microservice for API Integration





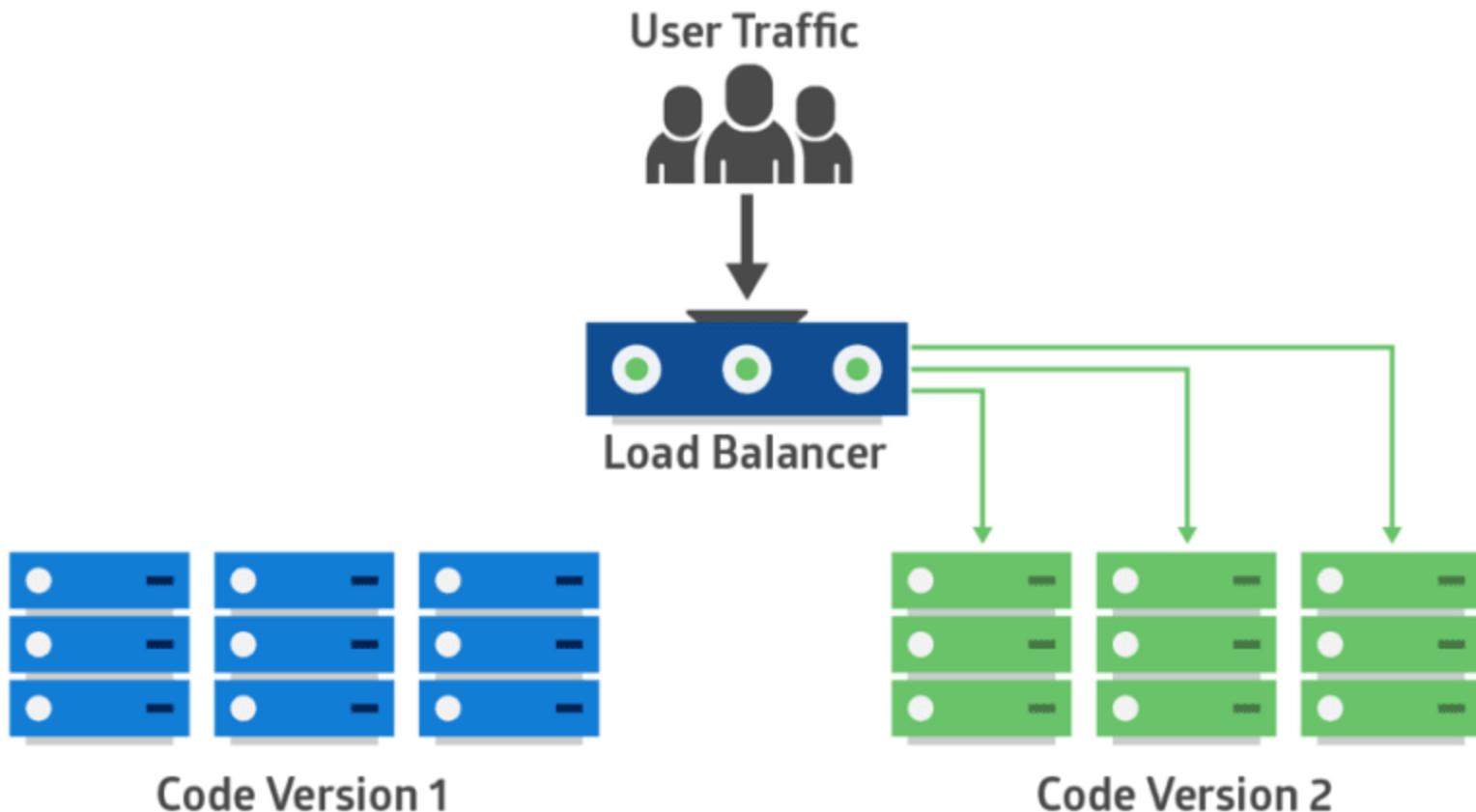
# ESB or Microservice

---

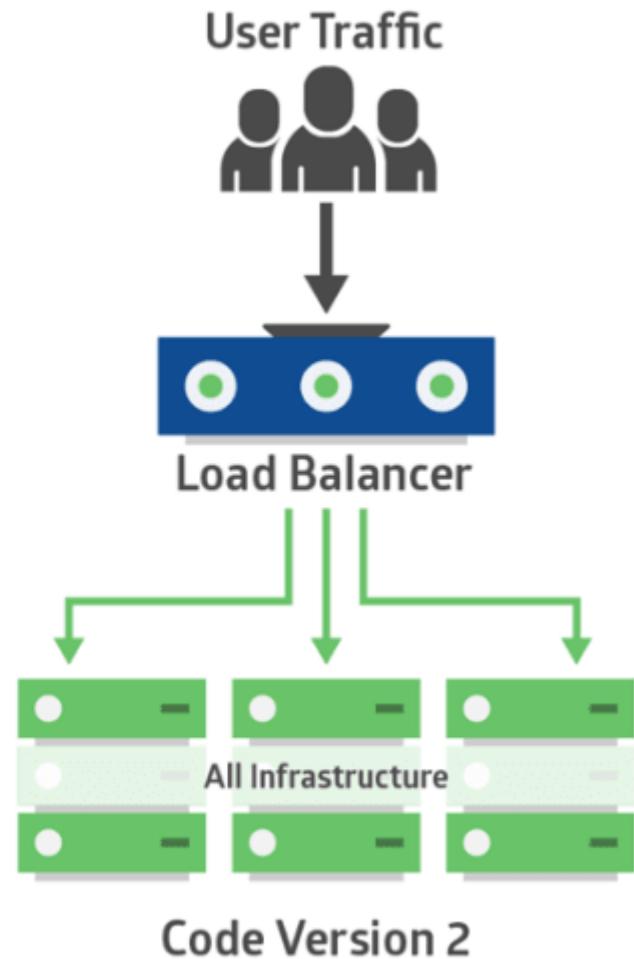
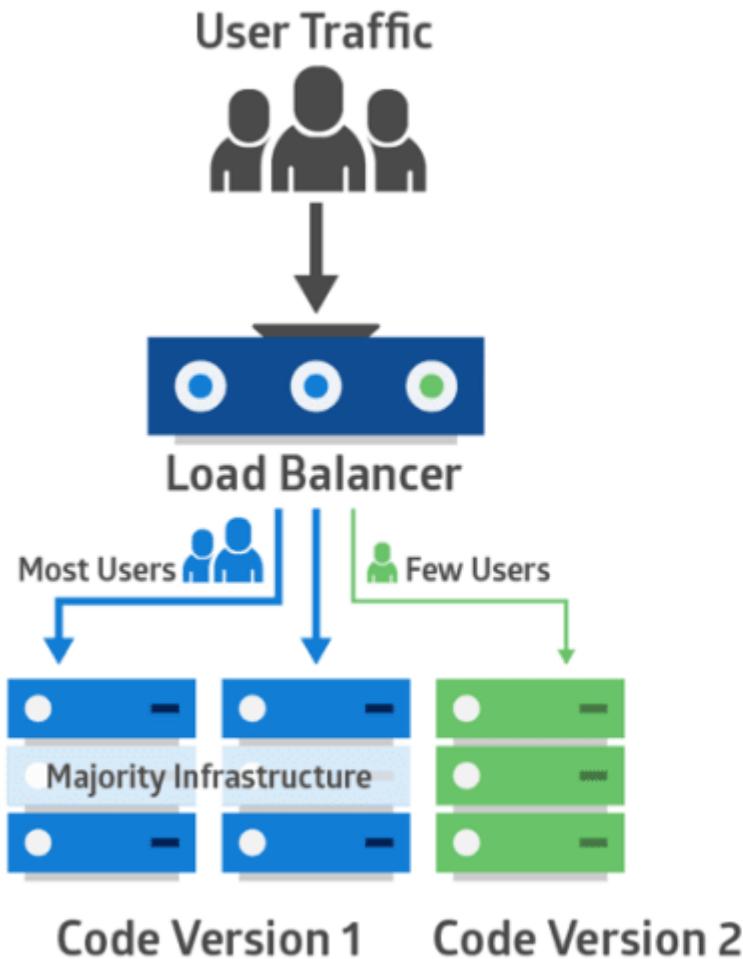
- When you select the suitable solution for the organization, following factors need to be assessed.
- Types of services to be integrated
- Communication protocols required
- Performance numbers
- Transaction requirements
- Security restrictions
- Inter-dependencies of services
- Business timelines
-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

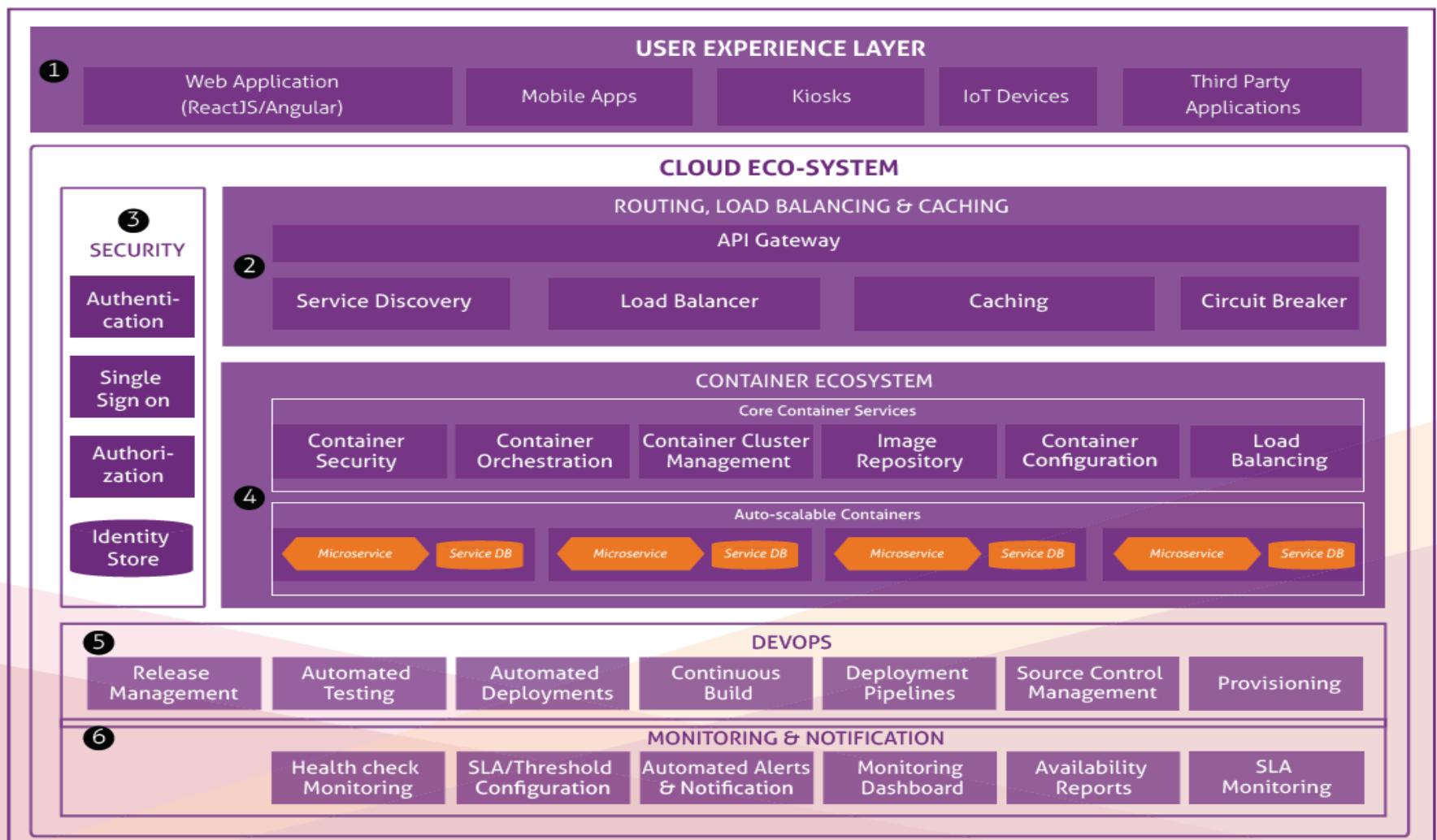


Figure 1: Microservices Reference Architecture



# Sample Microservice Interaction

## CLOUD NATIVE MICROSERVICE FRAMEWORK – SOLUTION COMPONENTS

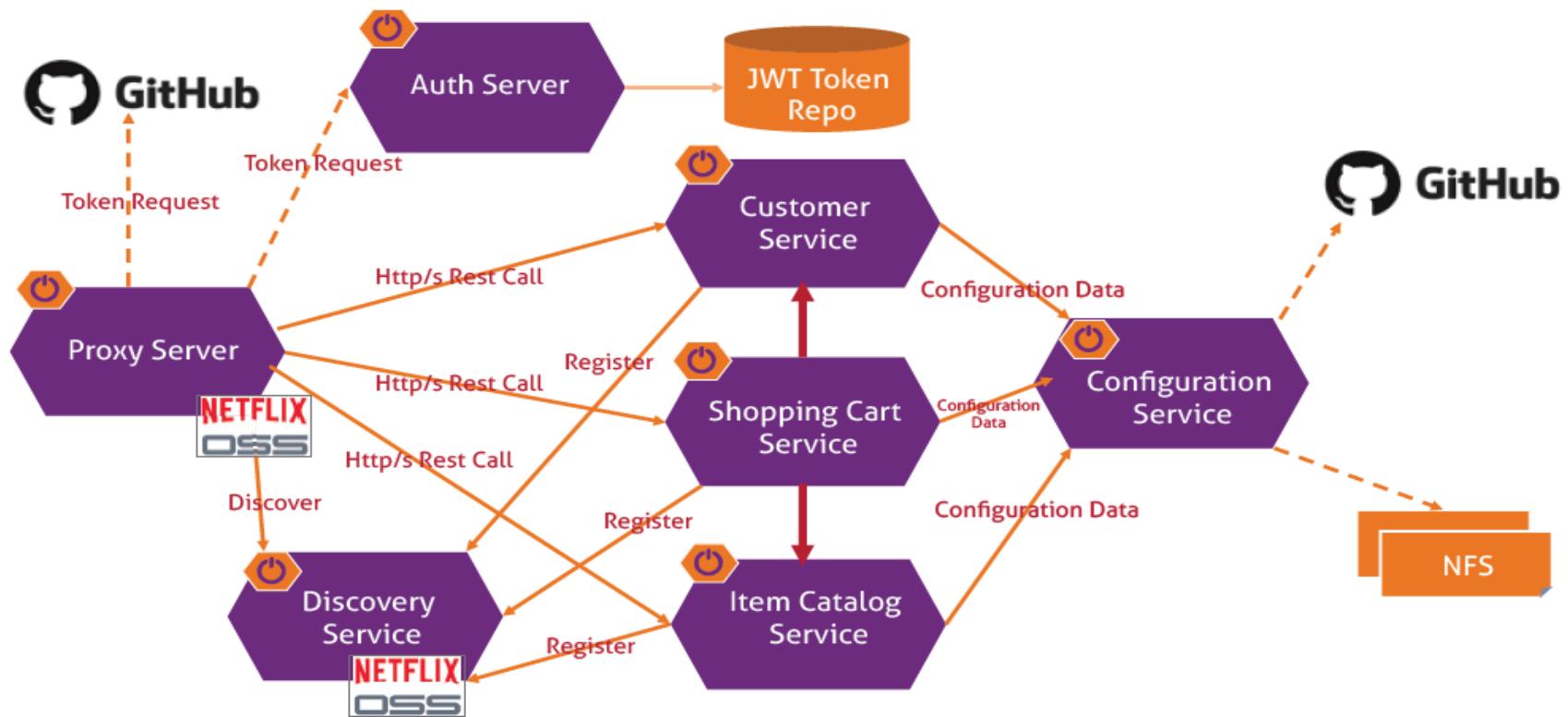


Figure 2: Sample Microservice Interaction Diagram



# 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

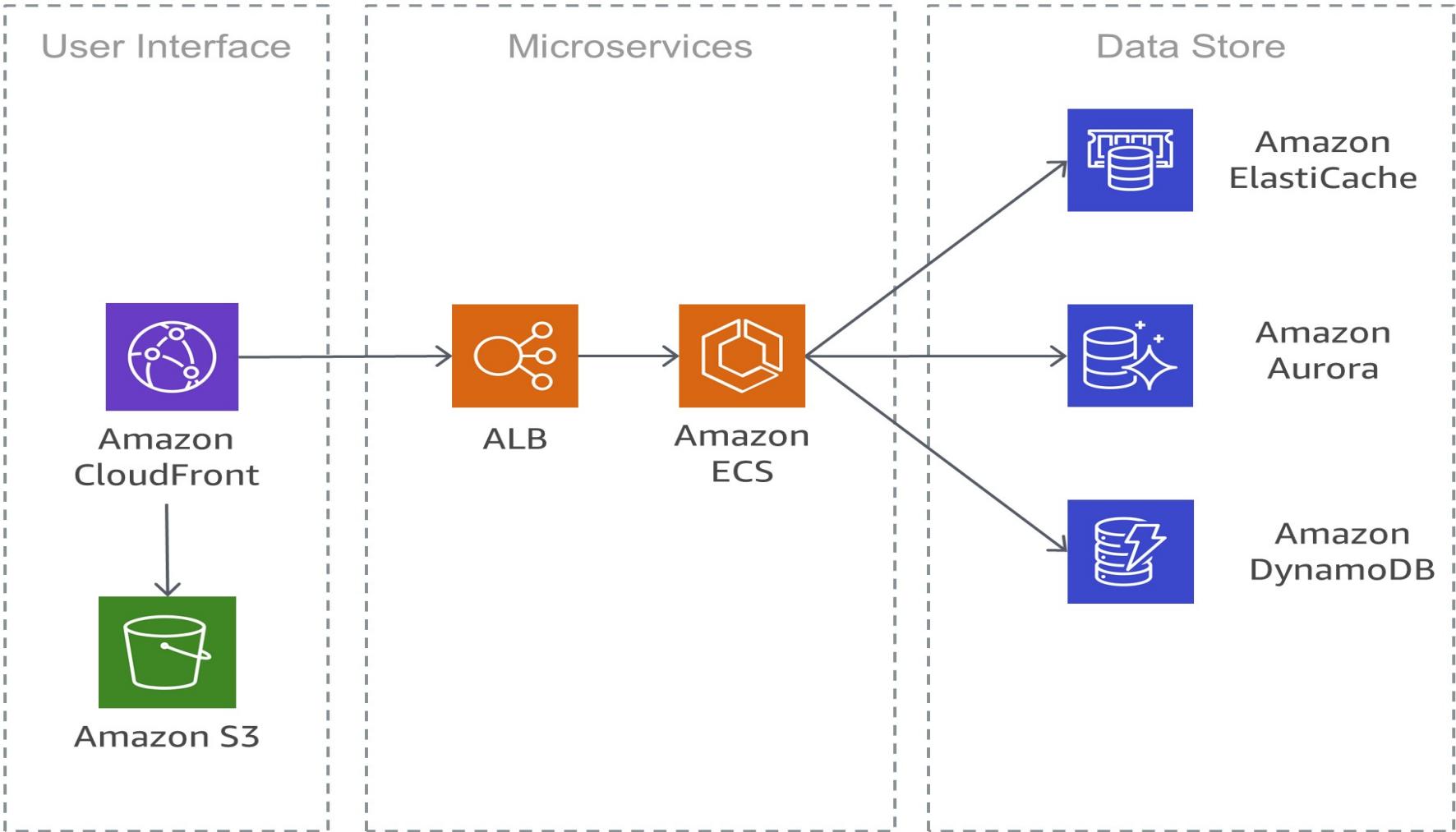


# Cloud Deployment Architecture - AWS

**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

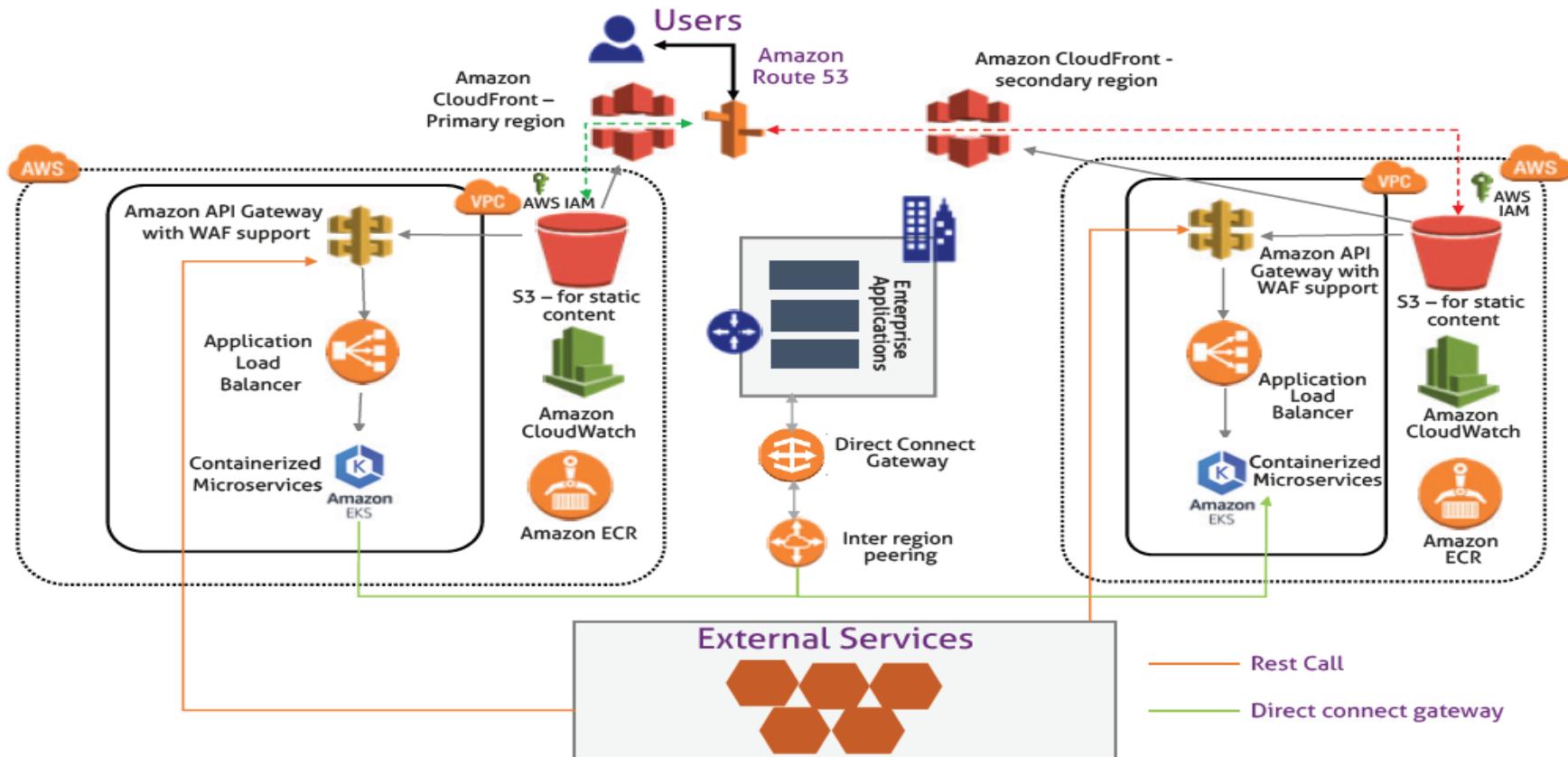


Figure 3: Sample AWS-based Microservices Deployment Architecture

# Cloud Deployment Architecture - Azure



## AZURE-BASED MICROSERVICES DEPLOYMENT ARCHITECTURE

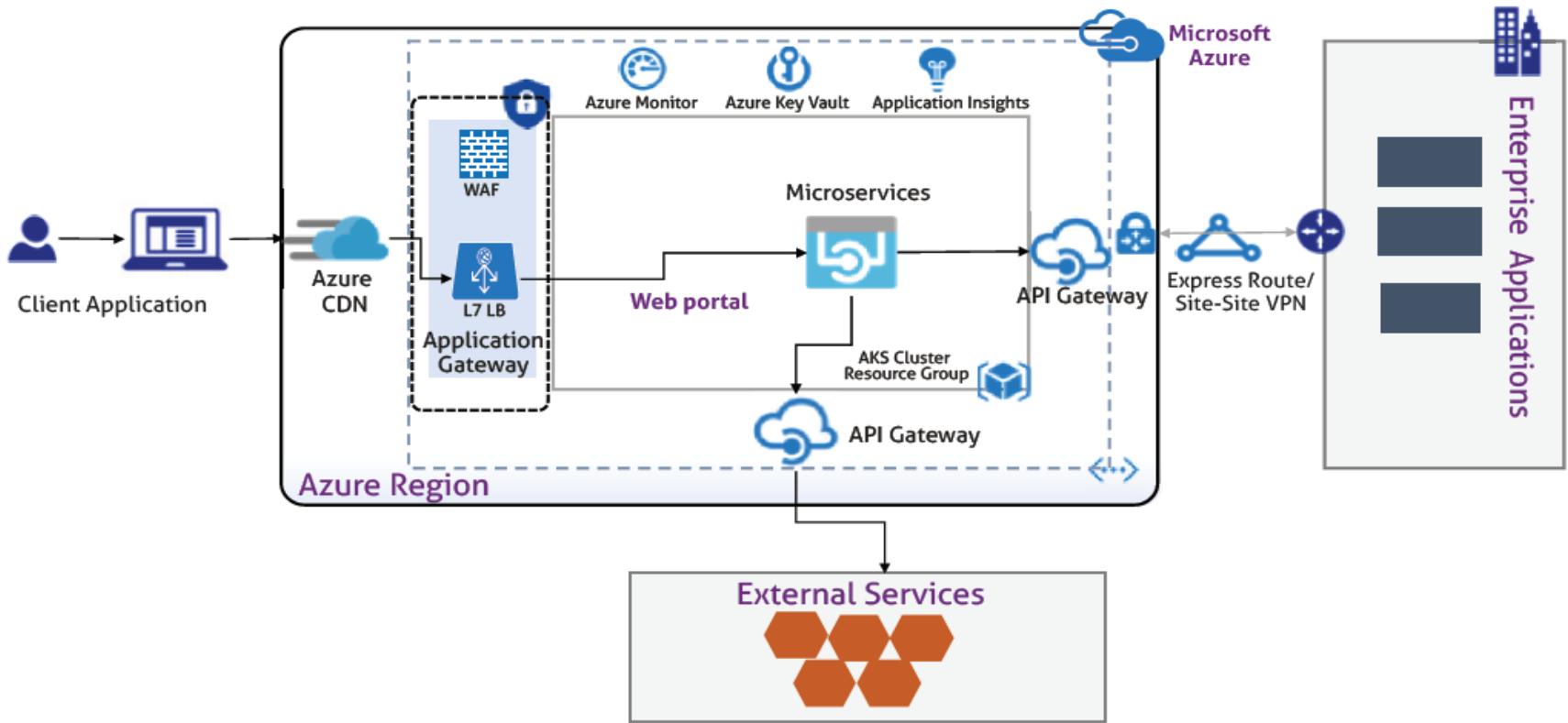


Figure 4: Sample Azure-based Microservices Deployment Architecture



# Microservices Features

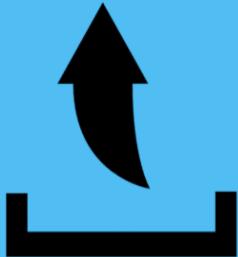




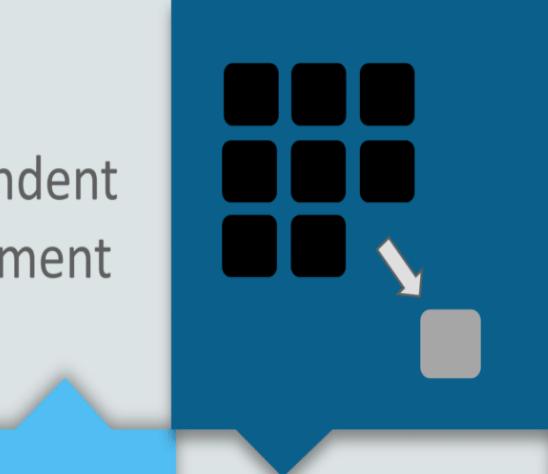
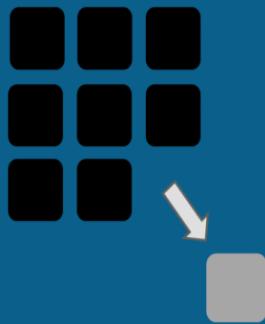
# Advantages Of Microservices



Independent Development



Independent Development



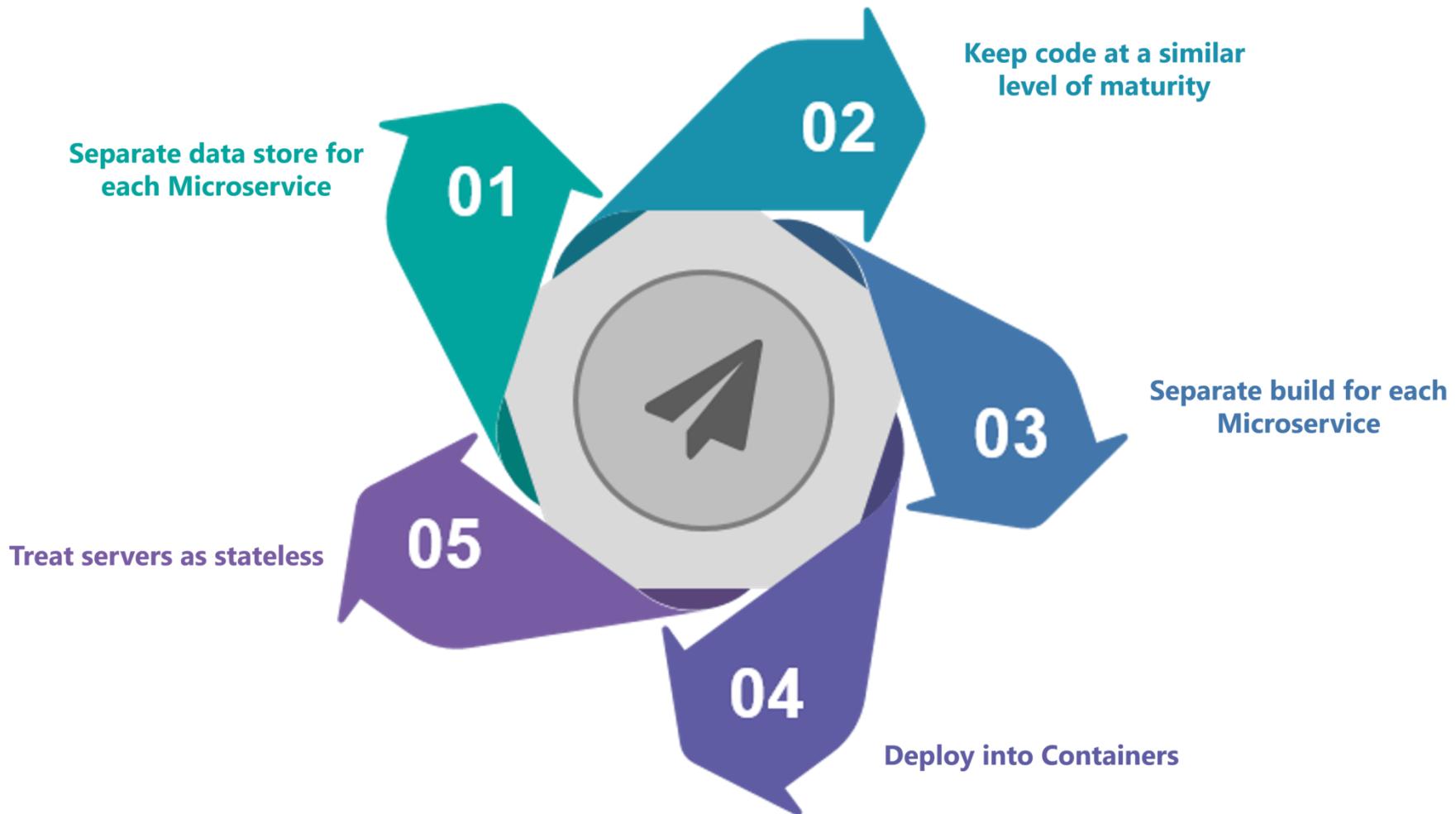
Fault Isolation

Mixed Technology Stack



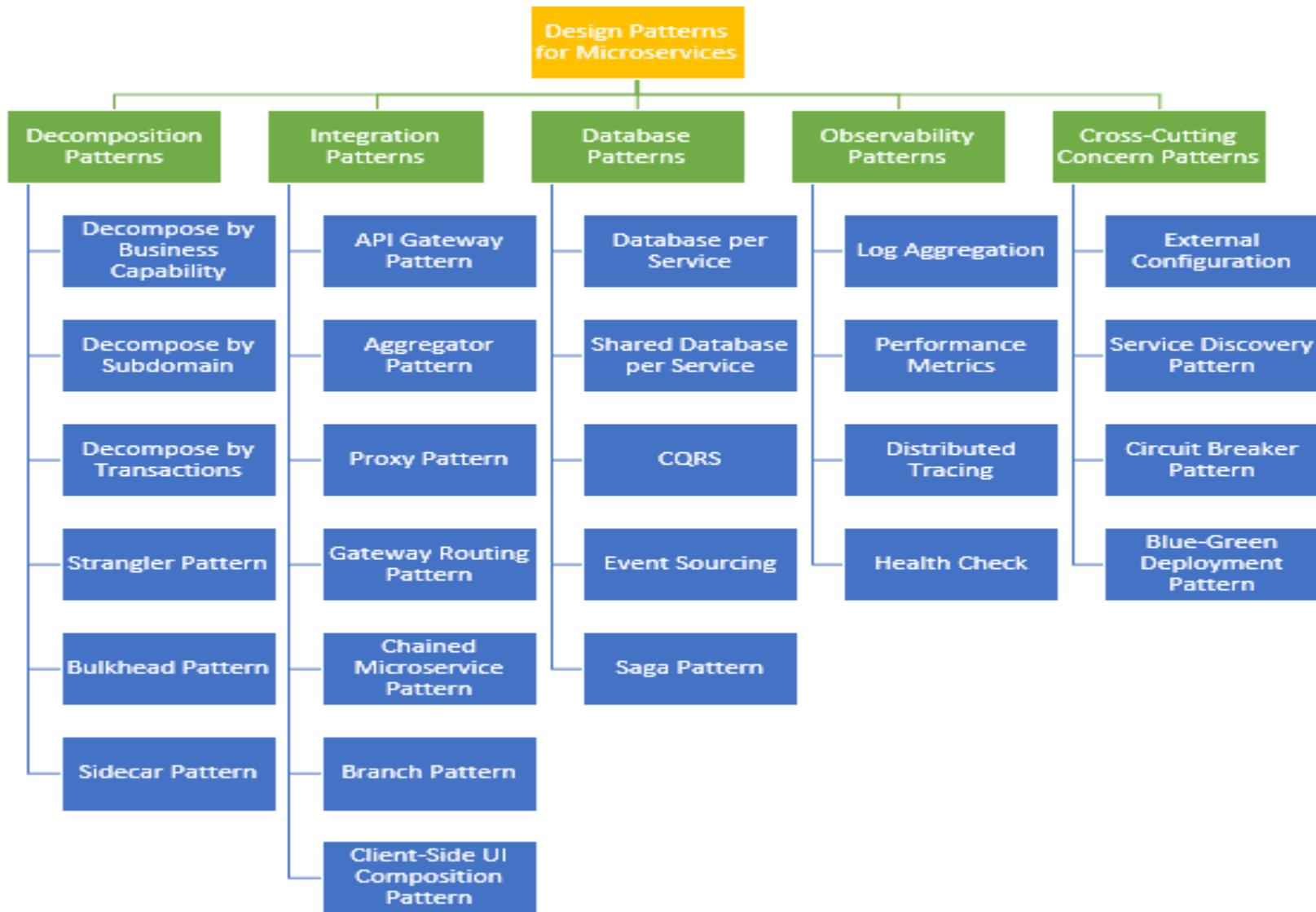
Granular Scaling

# Best Practices To Design Microservices





# Design Patterns of Micro services



# Questions



# Module Summary

---

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

