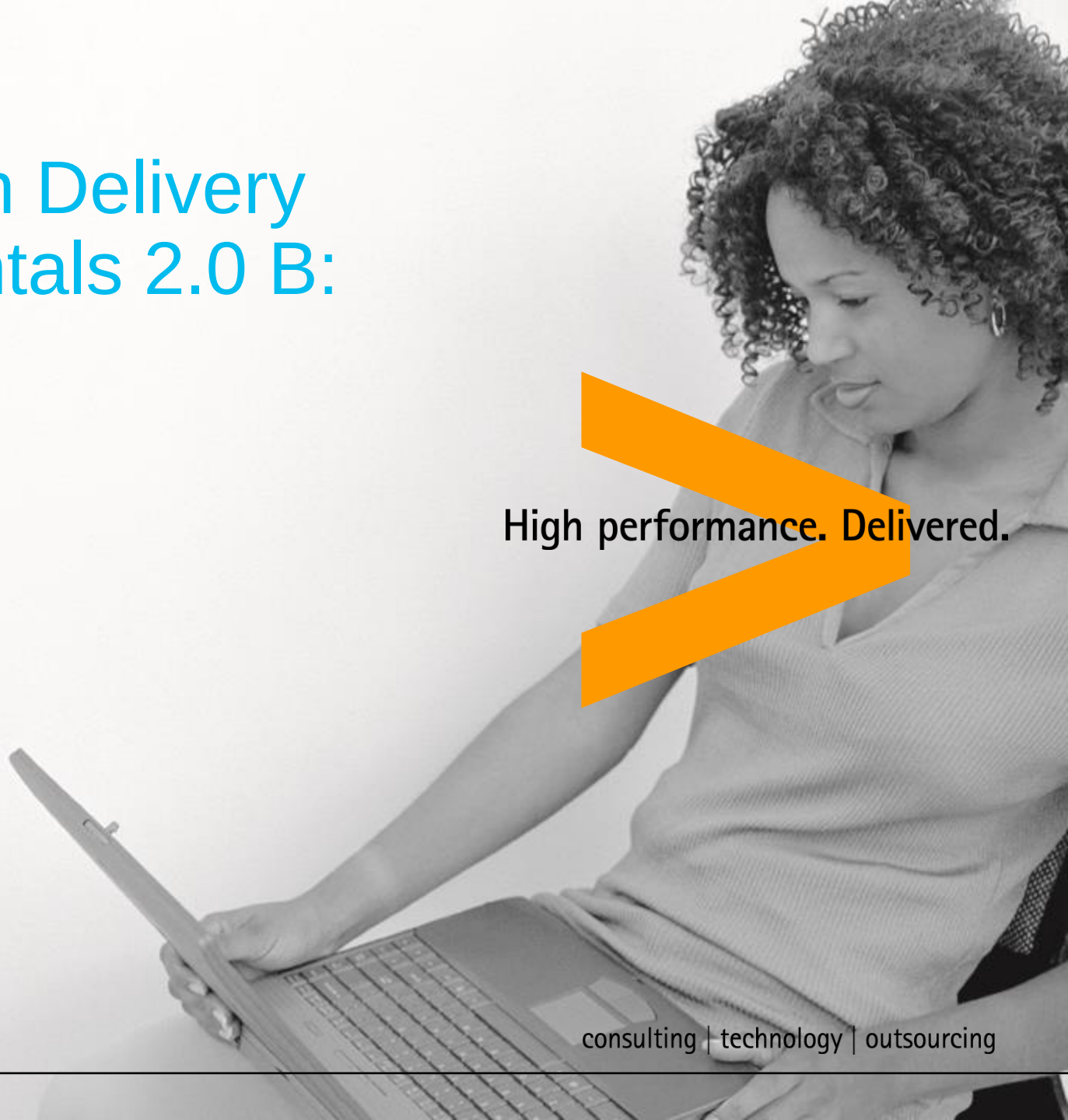
Application Delivery Fundamentals 2.0 B: Java

Micro Service



consulting | technology | outsourcing

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

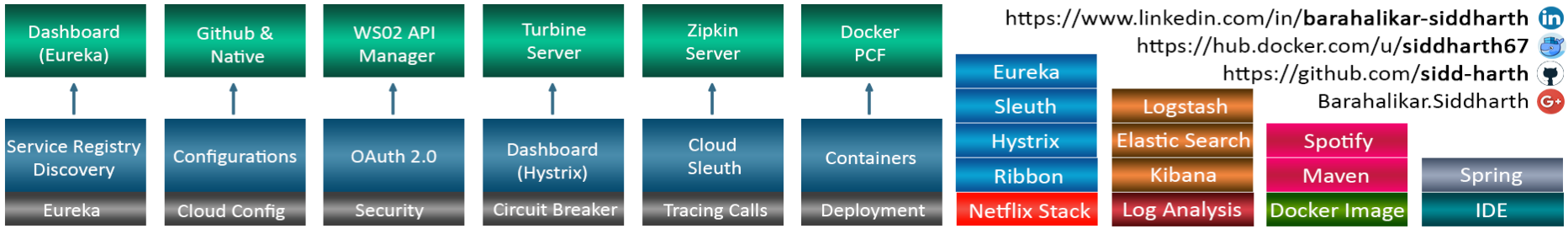
Micro Service

Goals

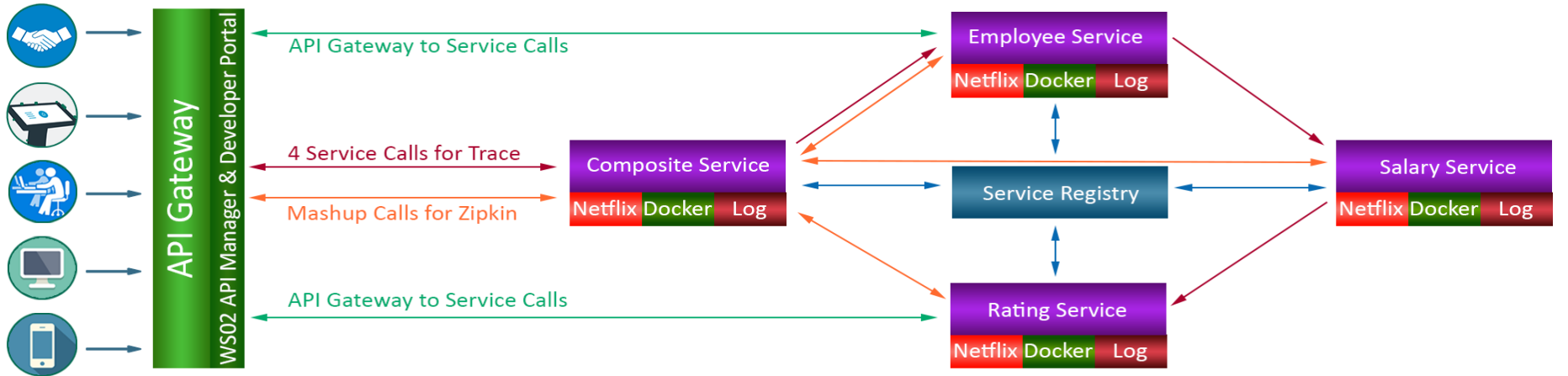
- Protection of system availability from individual Instance failure
- Automatic replacement of component instances when they fail (self-healing)
- Easy scaling of components by adjusting a simple parameter value
- Canary testing (A **canary deployment / canary test** allows you to gradually release 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)

Microservices

Spring Boot Microservices Managed via Spring Cloud, Netflix OSS, ELK Stack, Docker & WS02 APIM



https://www.linkedin.com/in/**barahalikar-siddharth**
https://hub.docker.com/u/**siddharth67**
https://github.com/**sidd-harth**
Barahalikar.Siddharth



- 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

WS02 - 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
- WS02 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.

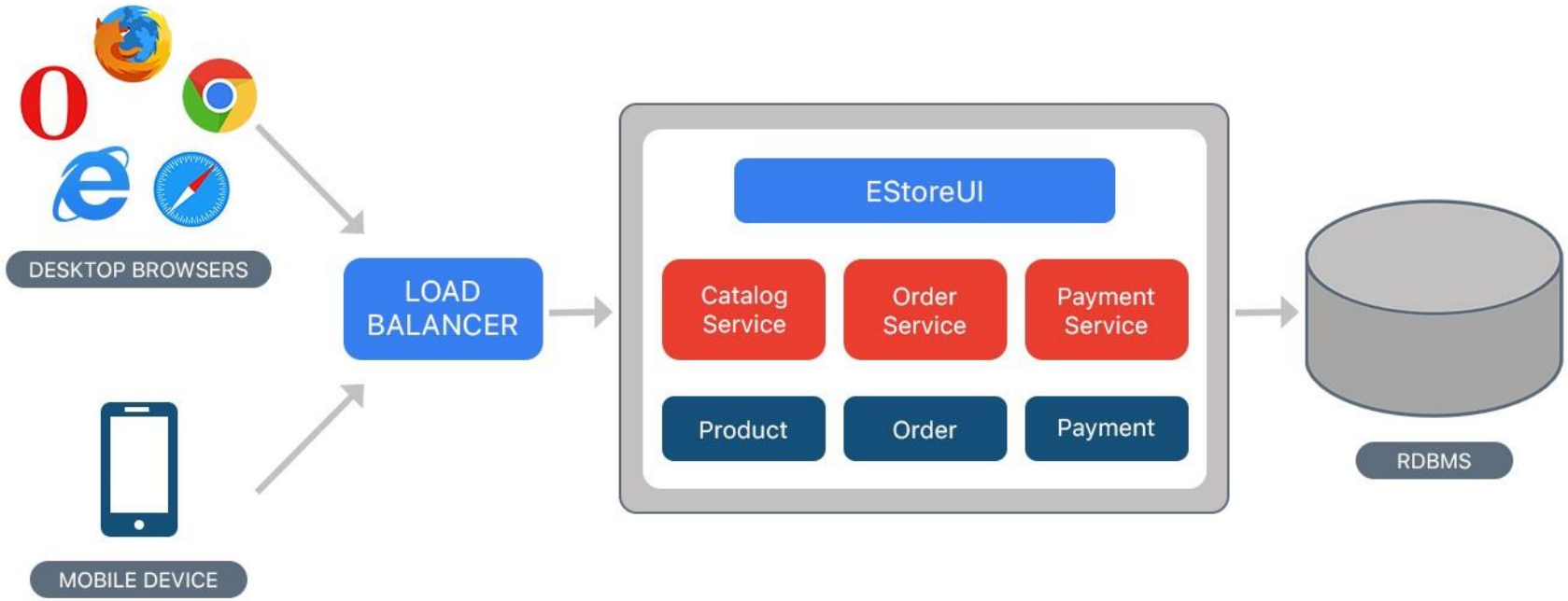
Monolithic Architecture

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

Microservices



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

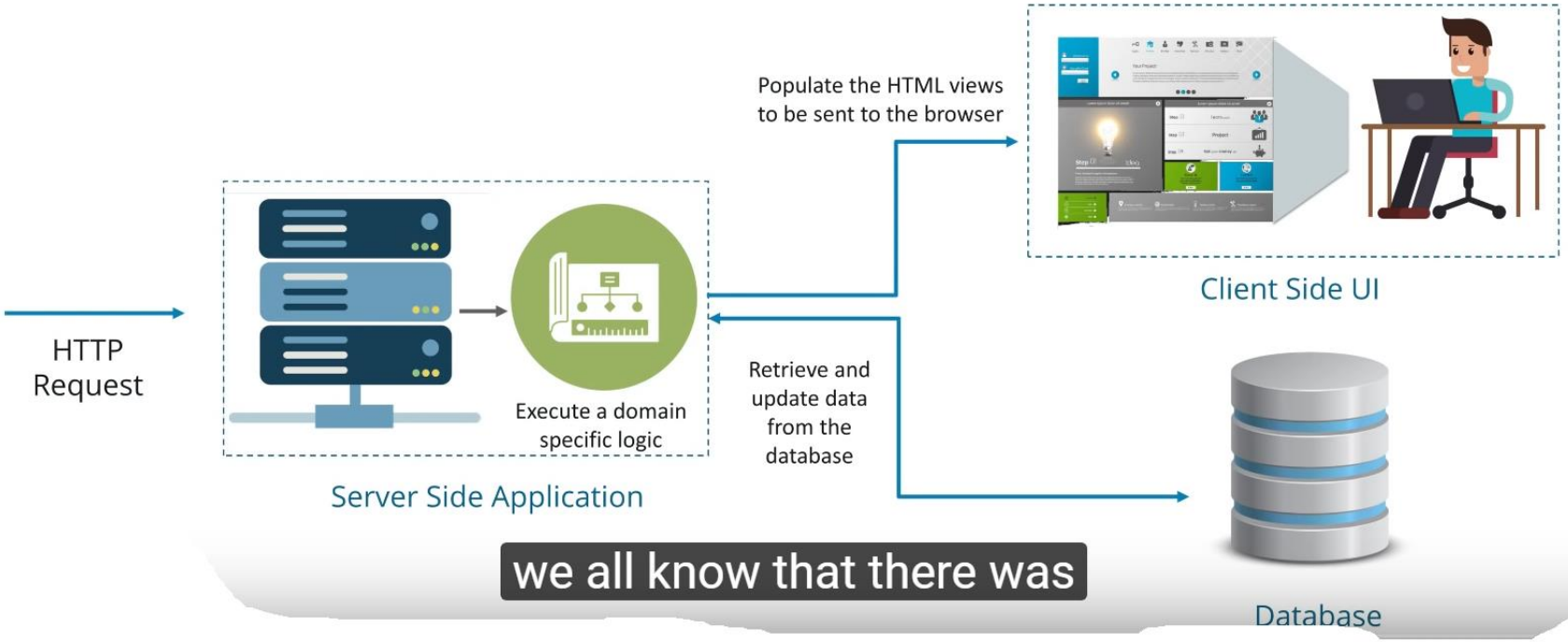
Drawbacks

- 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

Microservices

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

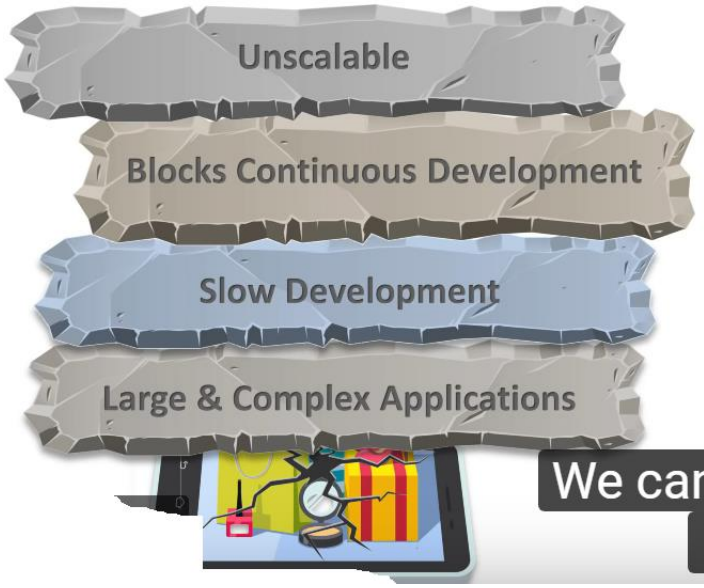


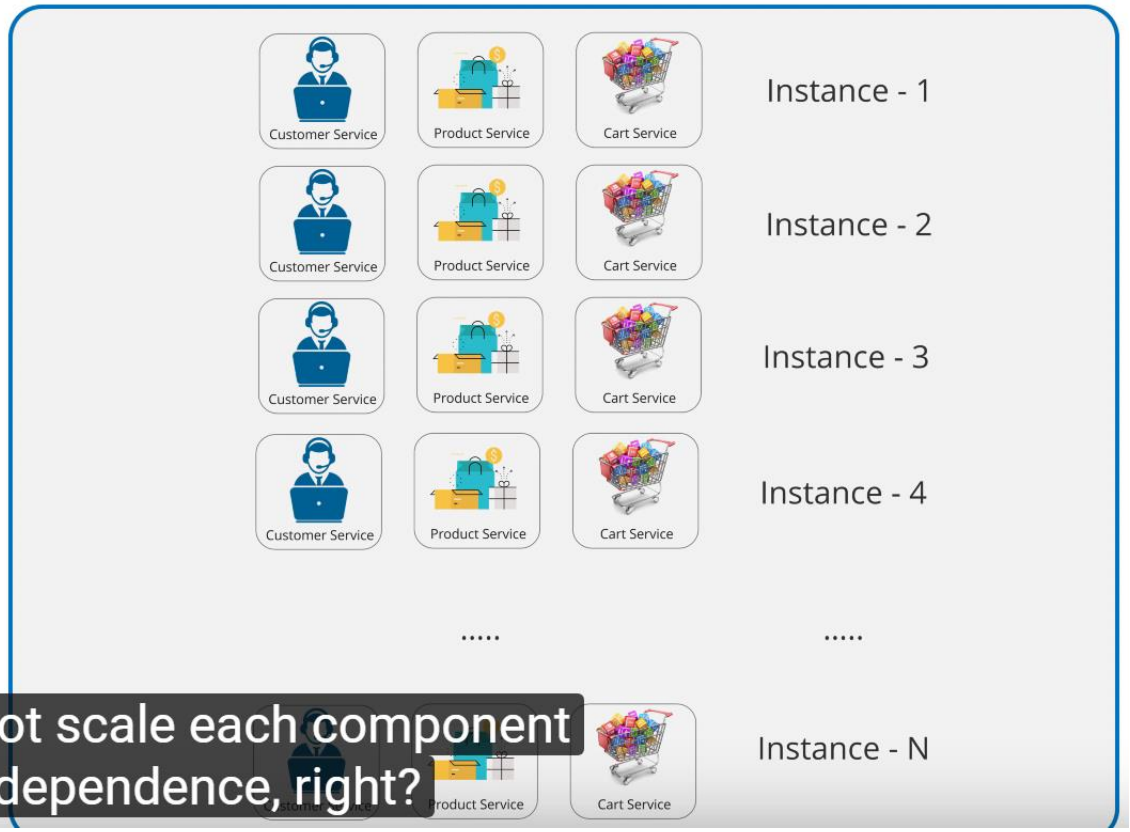
Monolithic Architecture

Microservices

Monolithic Architecture

Microservices

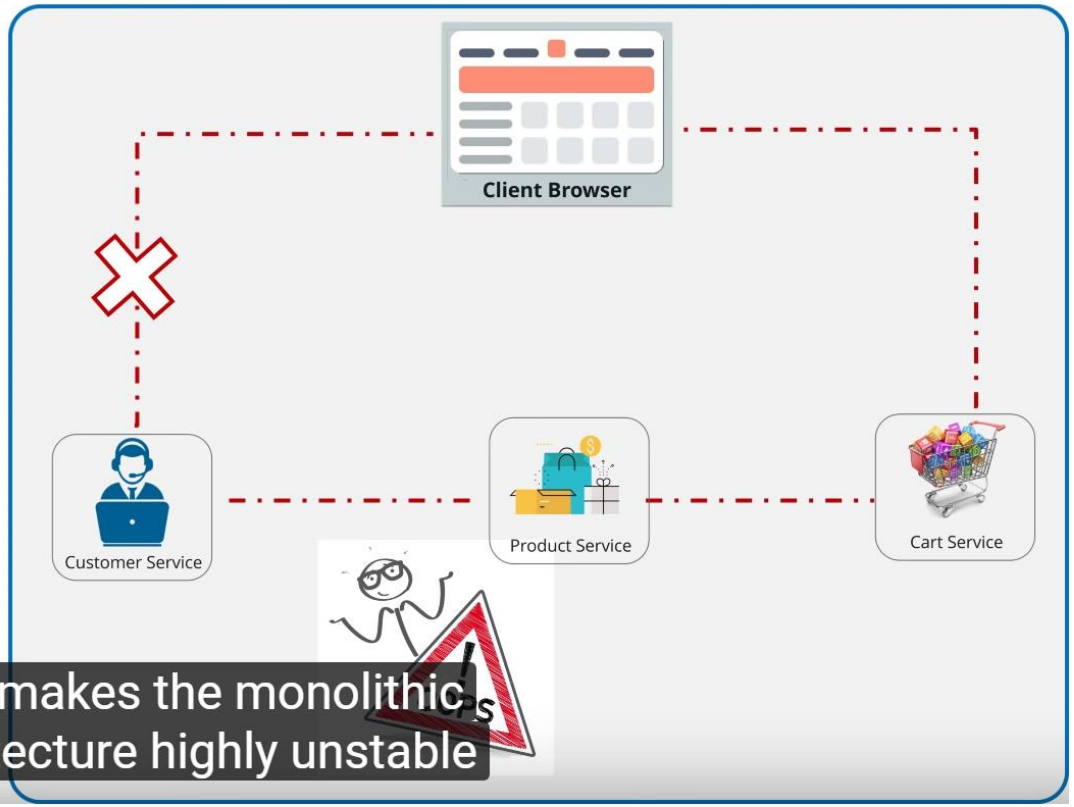




We cannot scale each component
Independence, right?

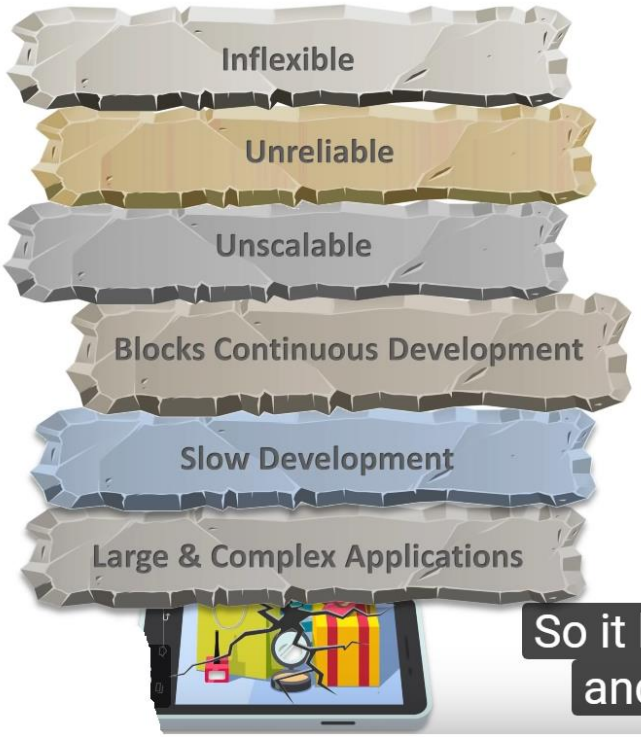
Monolithic Architecture

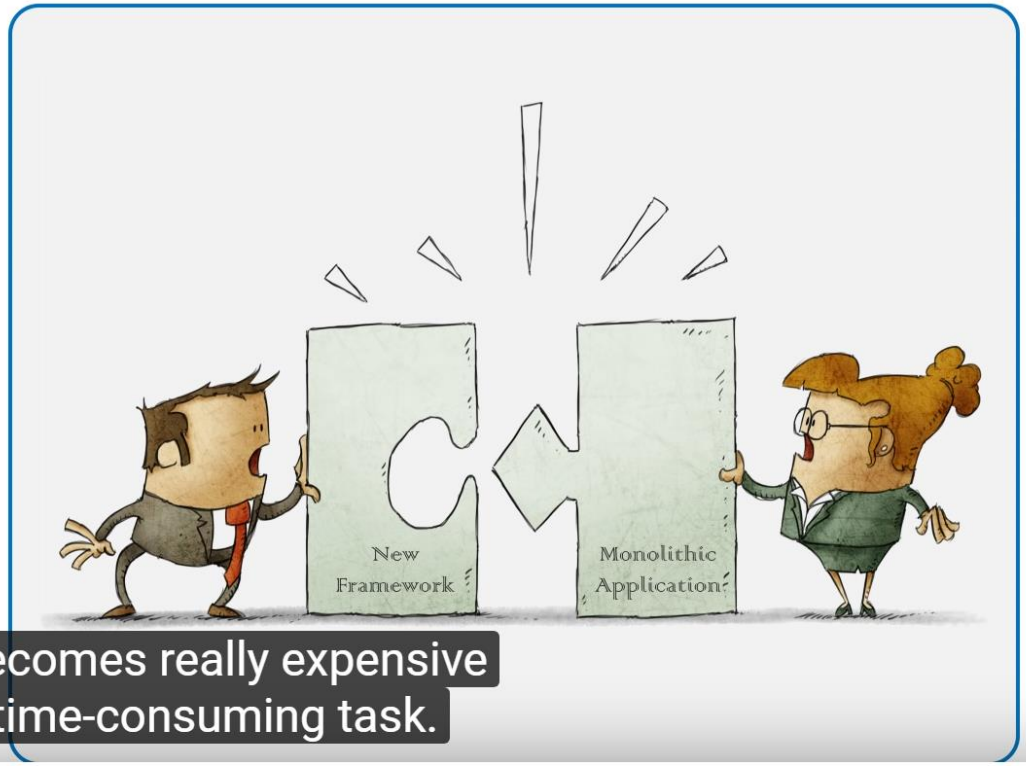
Microservices



Monolithic Architecture

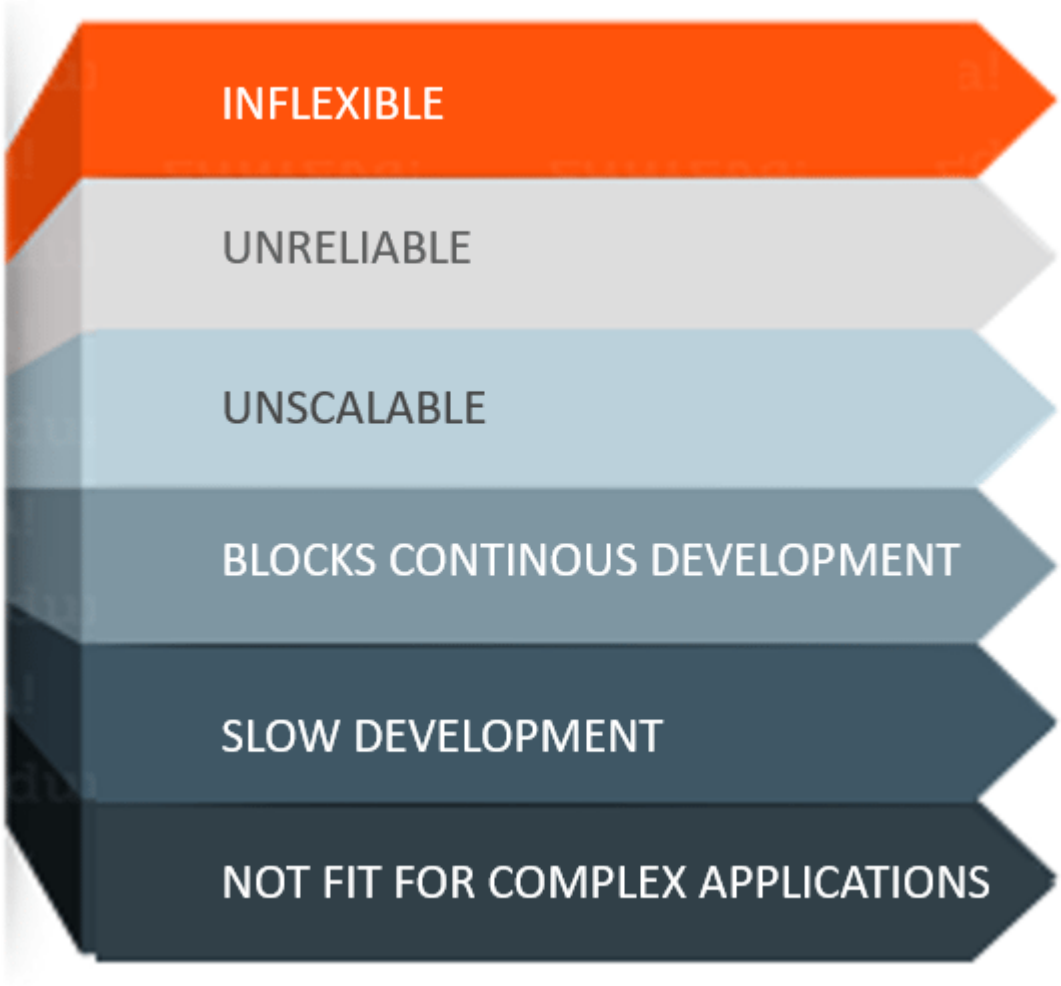
Microservices





Monolithic Architecture

Microservices

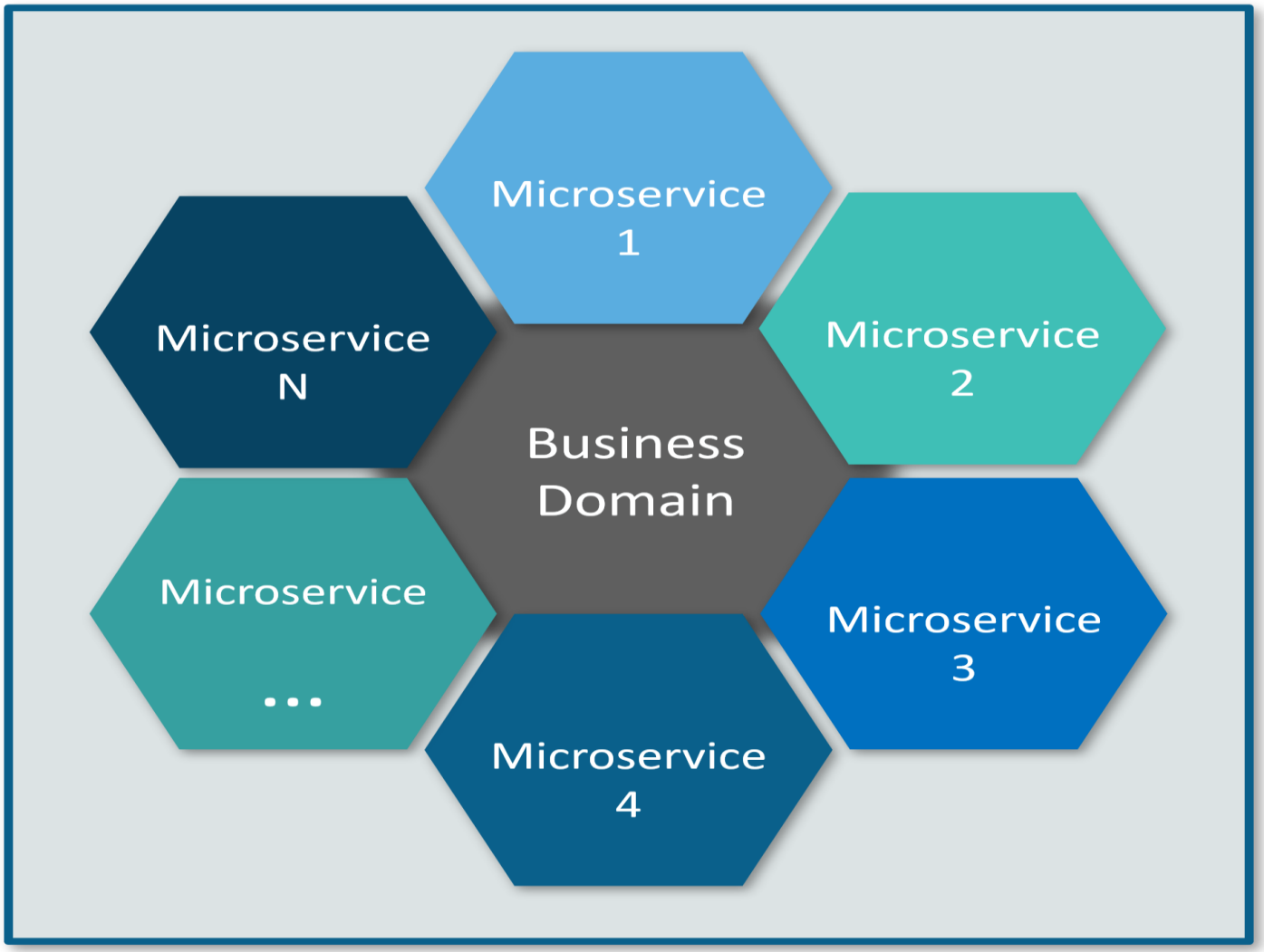


Micro Service

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

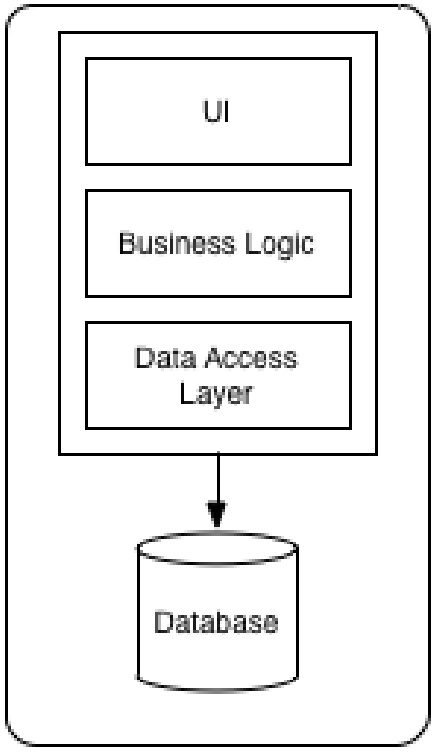
Micro Service

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

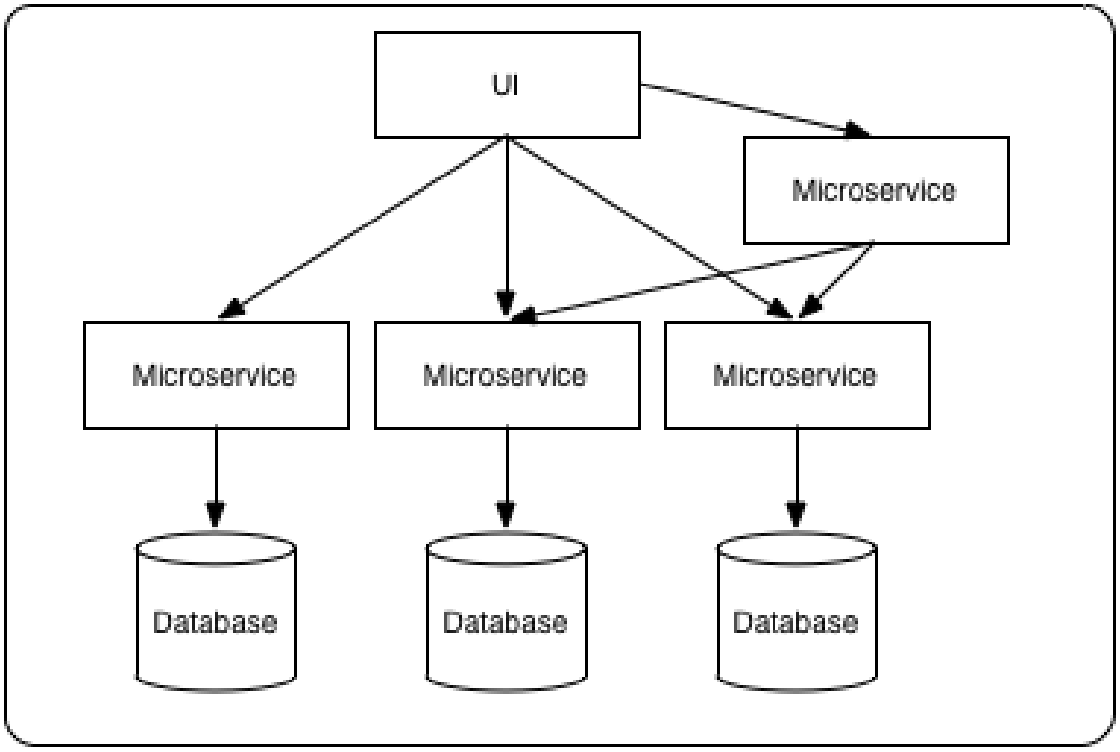
Micro Service

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

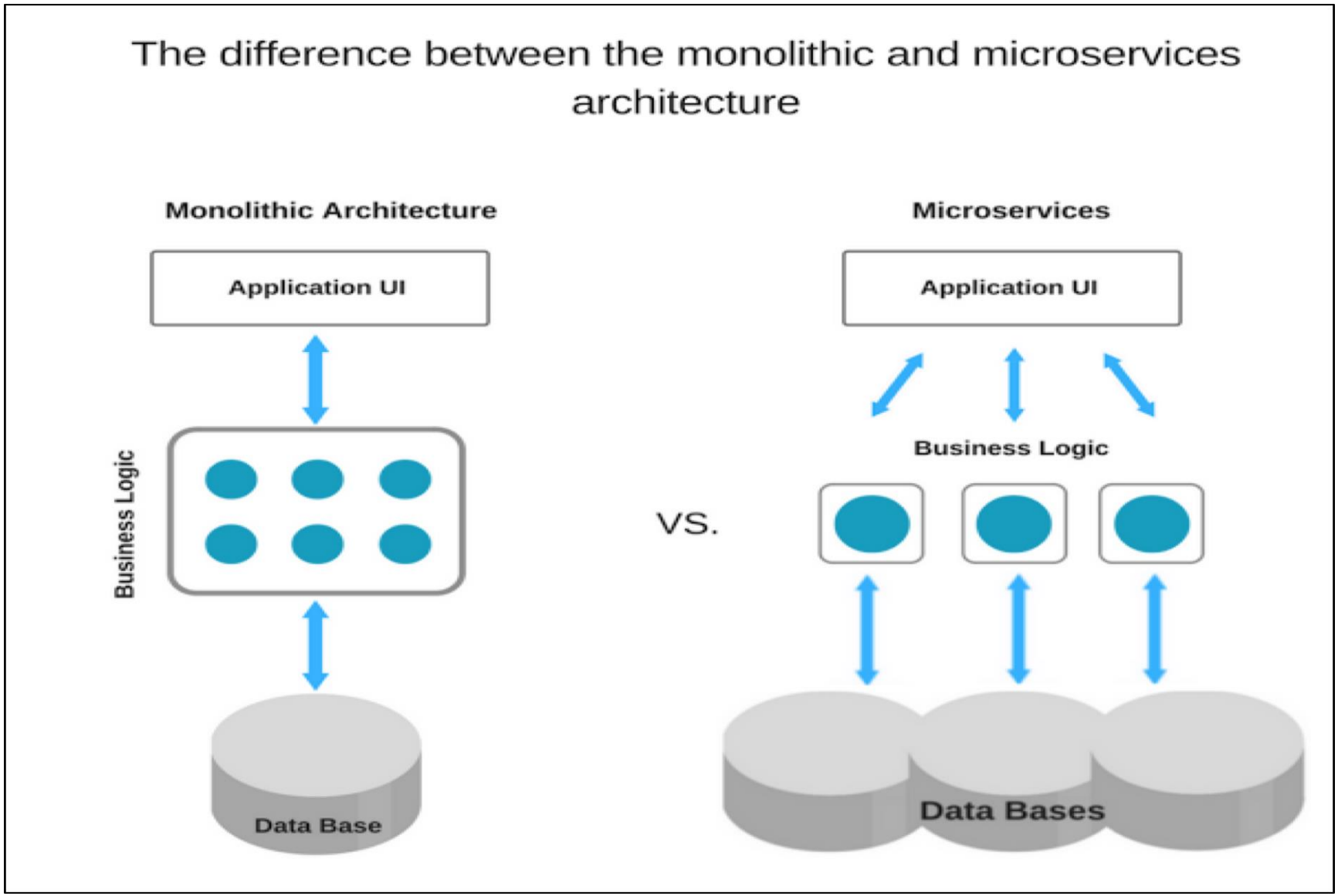


Monolithic Architecture



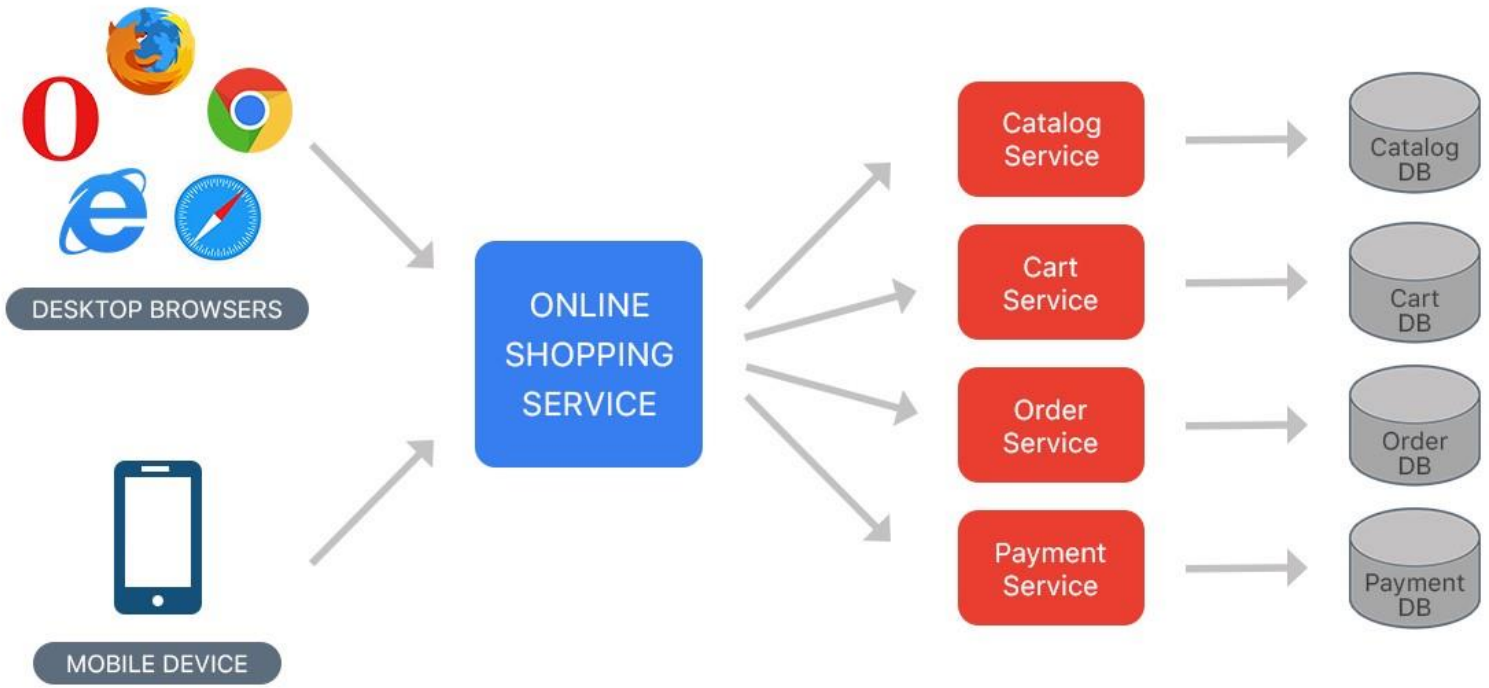
Microservices Architecture

MicroService



Micro Service

Microservices

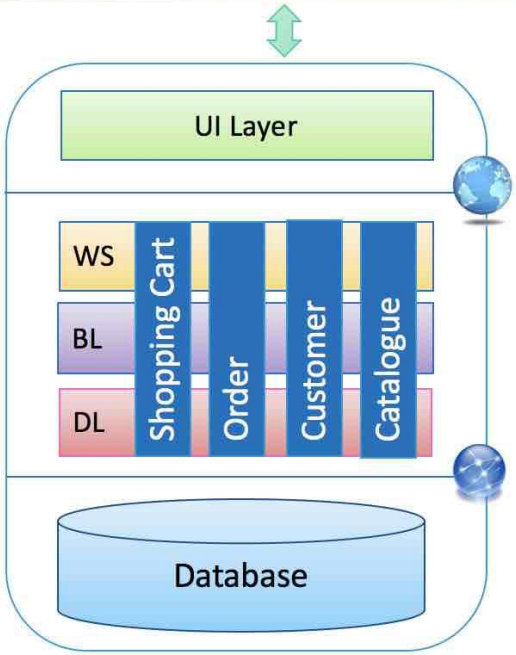


Monolithic vs. Micro Services Example

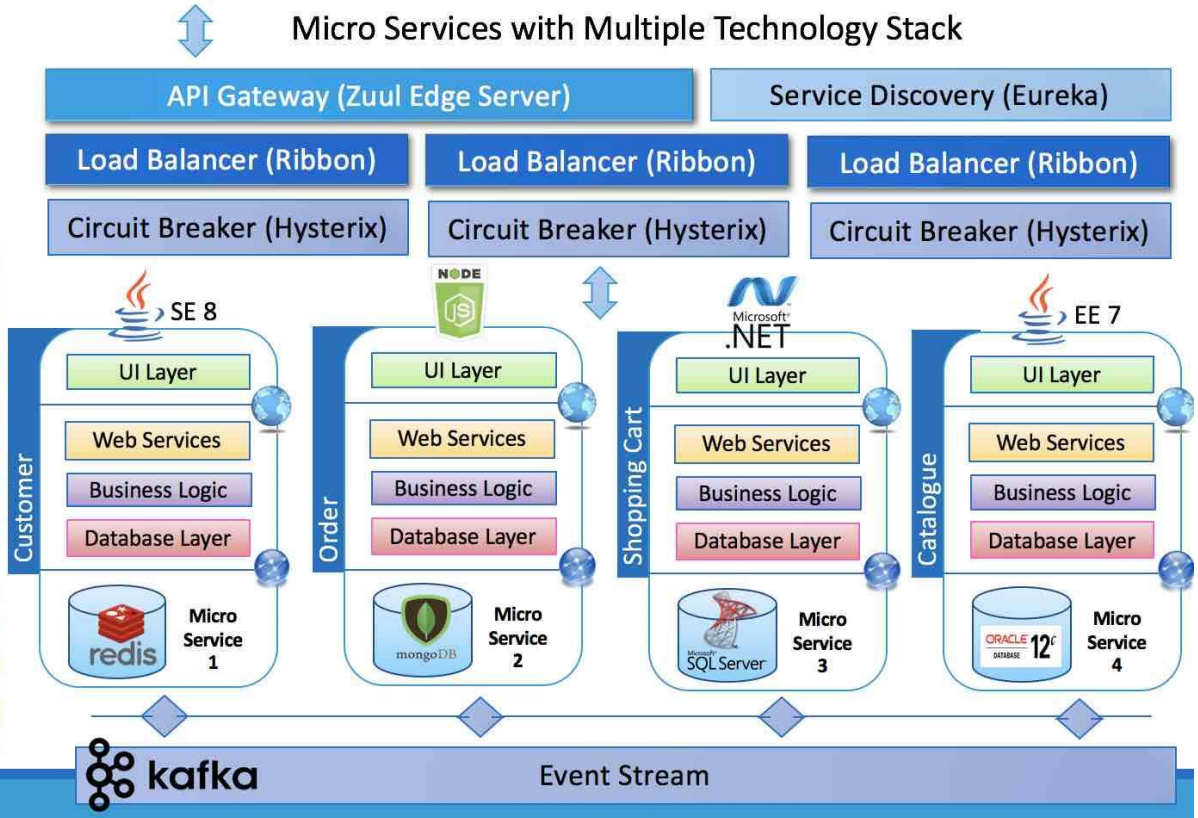
Existing aPaaS vendors creates Monolithic Apps.

This 3 tier model is obsolete now.

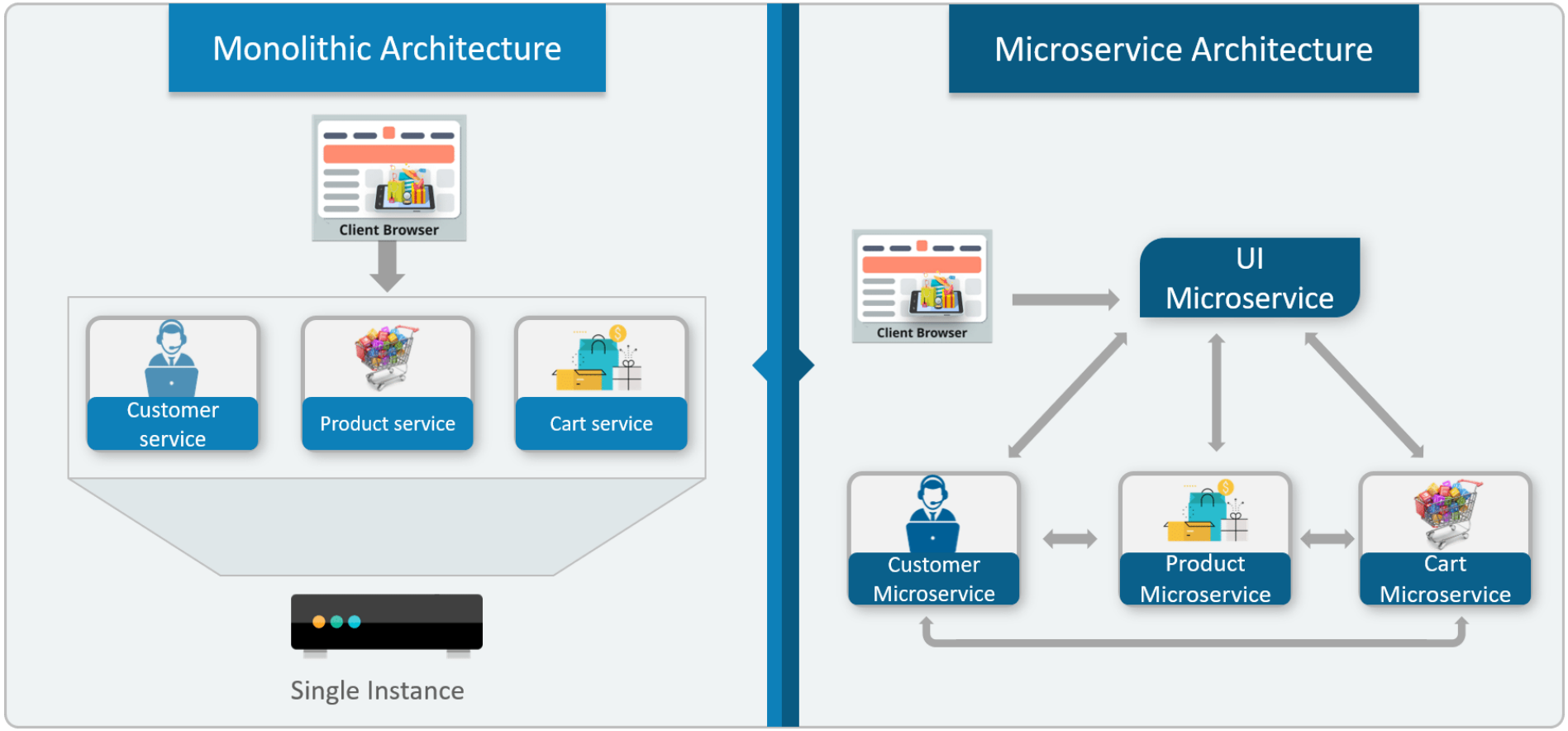
Source: Gartner Market Guide for Application Platforms Nov 23, 2016



Traditional Monolithic App using Single Technology Stack

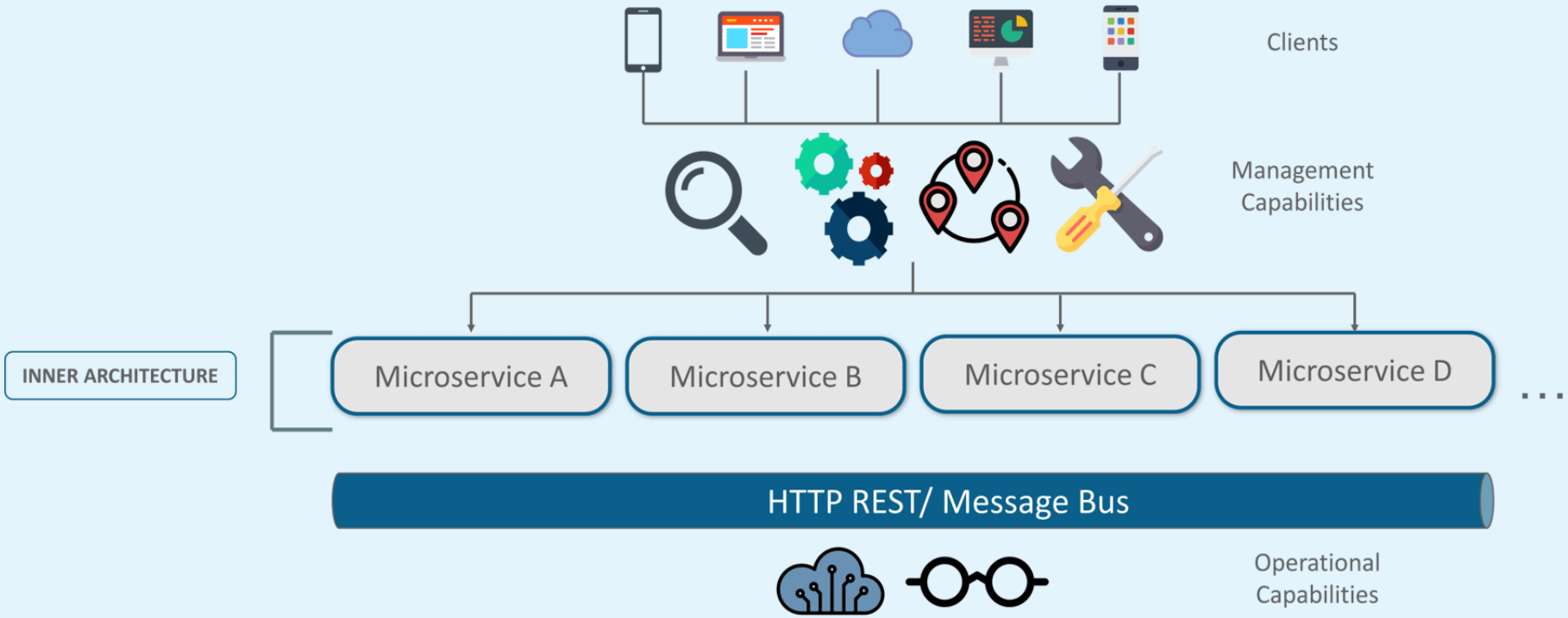



Microservices


Monolithic Architecture
Client Browser
Customer service
Product service
Cart service
Single Instance
Microservice Architecture
Client Browser
UI Microservice
Customer Microservice
Product Microservice
Cart Microservice

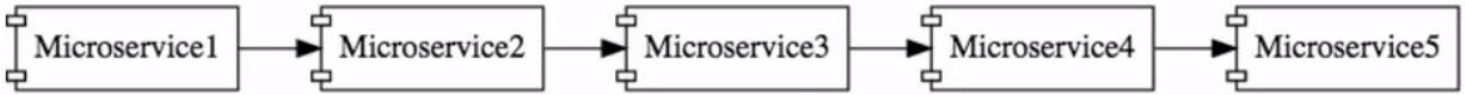
Microservice Architecture

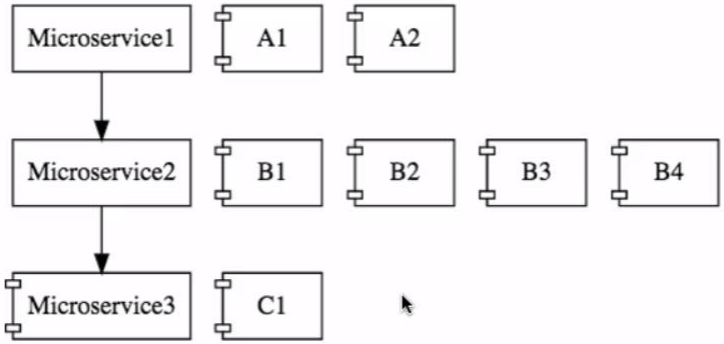
Microservices



Micro Service

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





What is Microservice

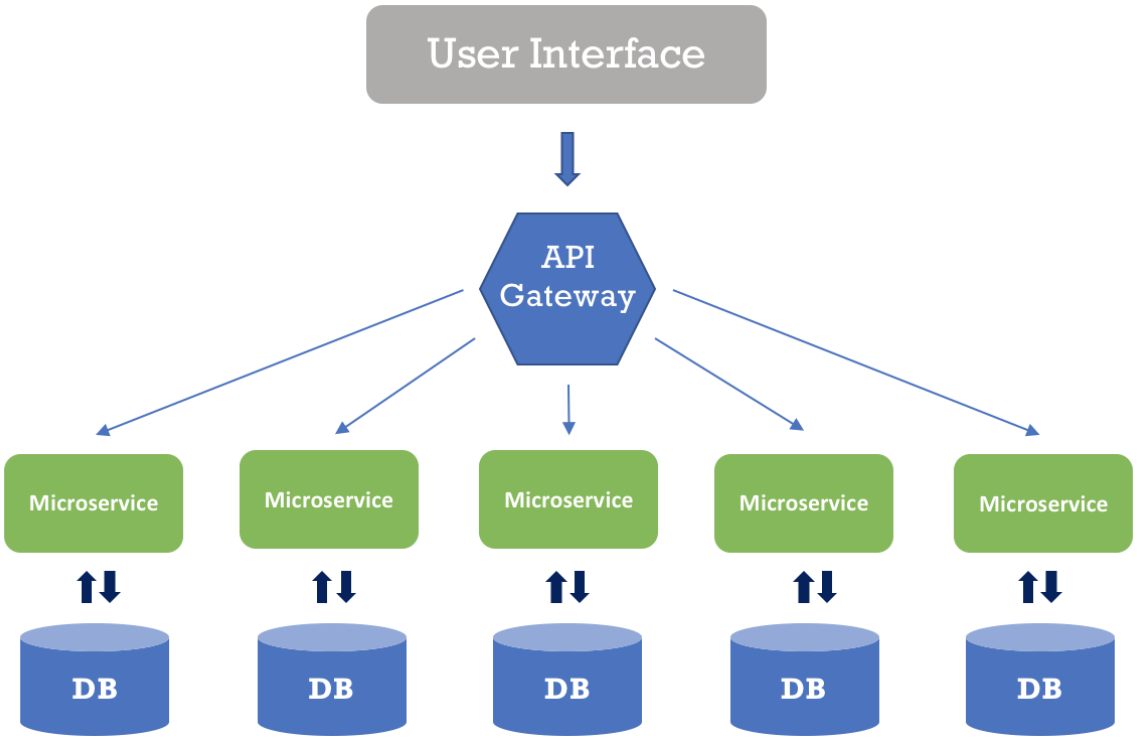
- **Microservices** is a variant of the service-oriented architecture (SOA) architectural style that structures an application as a collection of loosely coupled services.

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

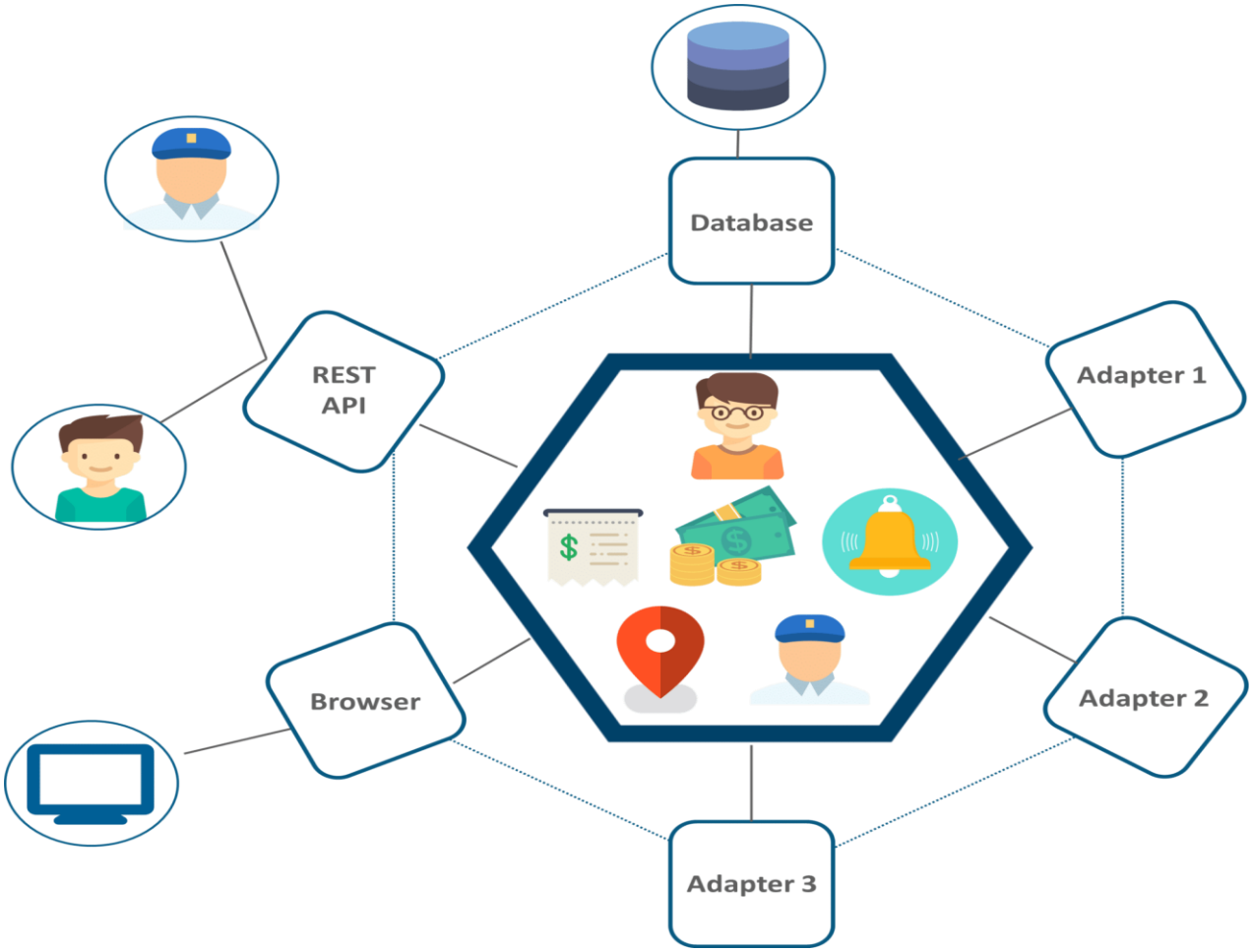
- It also parallelizes development by enabling small autonomous teams to develop, deploy and scale their respective services independently.



UBER CASE STUDY

Microservices

UBER's Previous Architecture

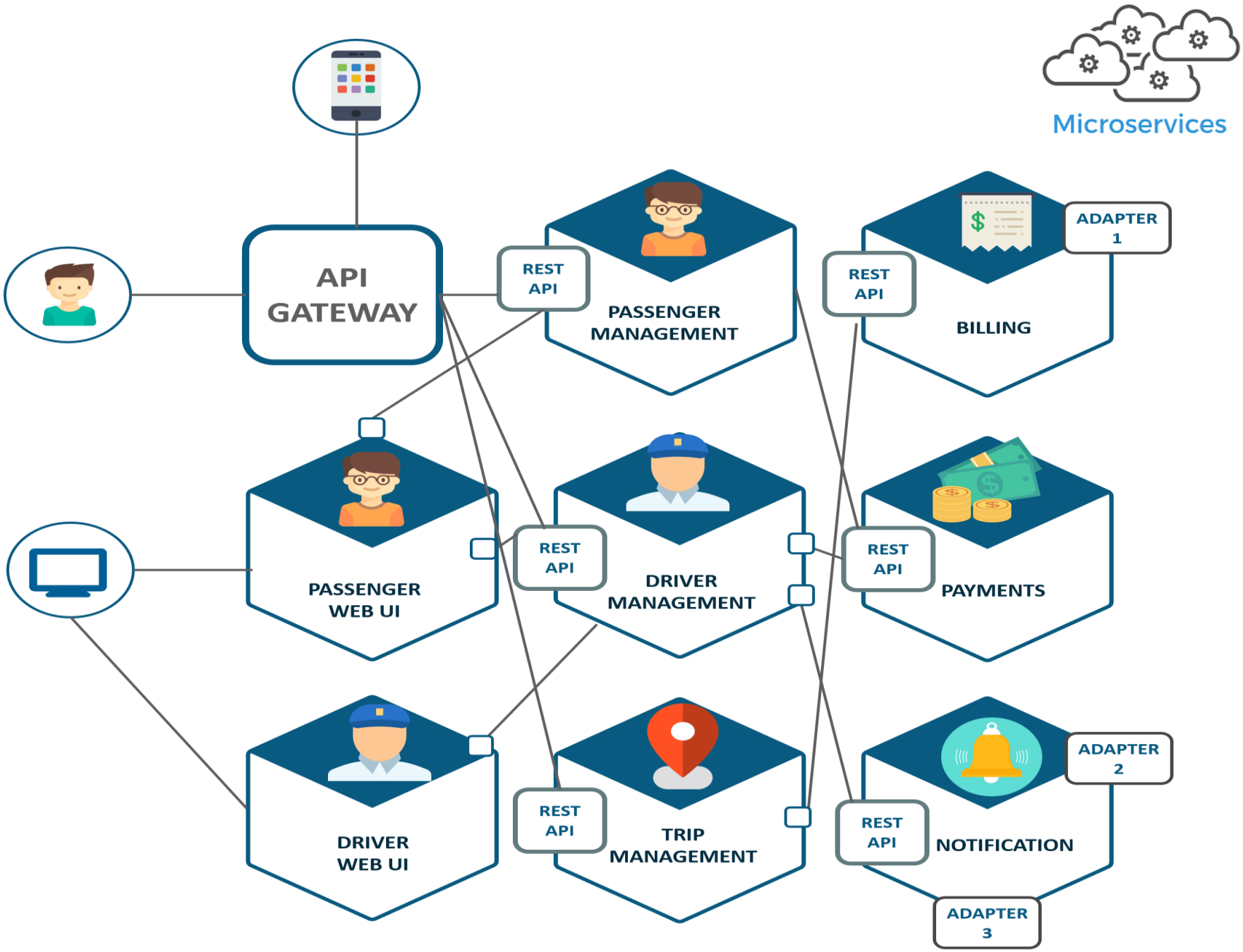


Problem Statement

- 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.

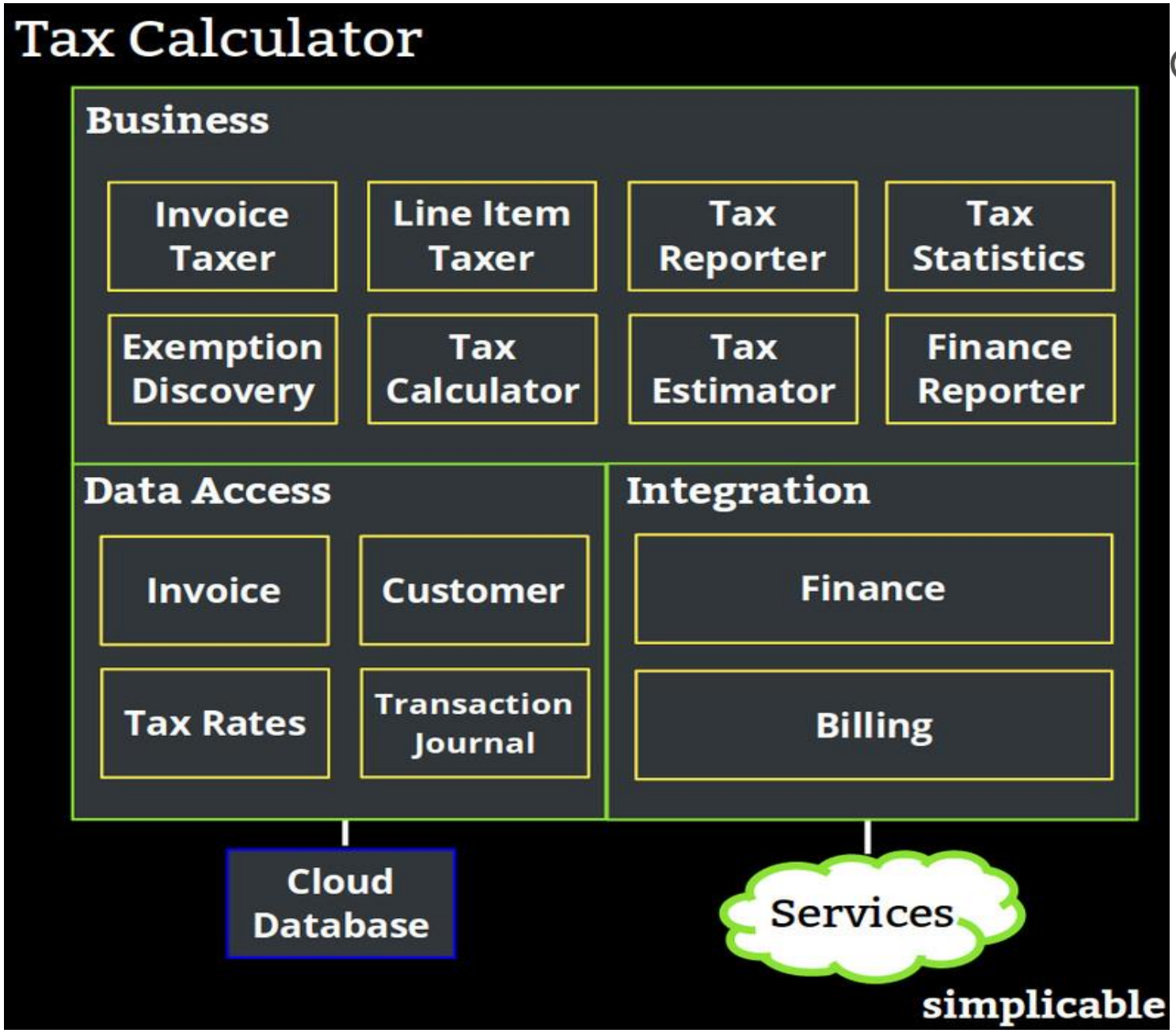
Solution

- 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.


Microservices
API GATEWAY
REST API
PASSENGER MANAGEMENT
REST API
BILLING
ADAPTER 1
PASSENGER WEB UI
REST API
DRIVER MANAGEMENT
REST API
PAYMENTS
DRIVER WEB UI
REST API
TRIP MANAGEMENT
REST API
NOTIFICATION
ADAPTER 2
ADAPTER 3

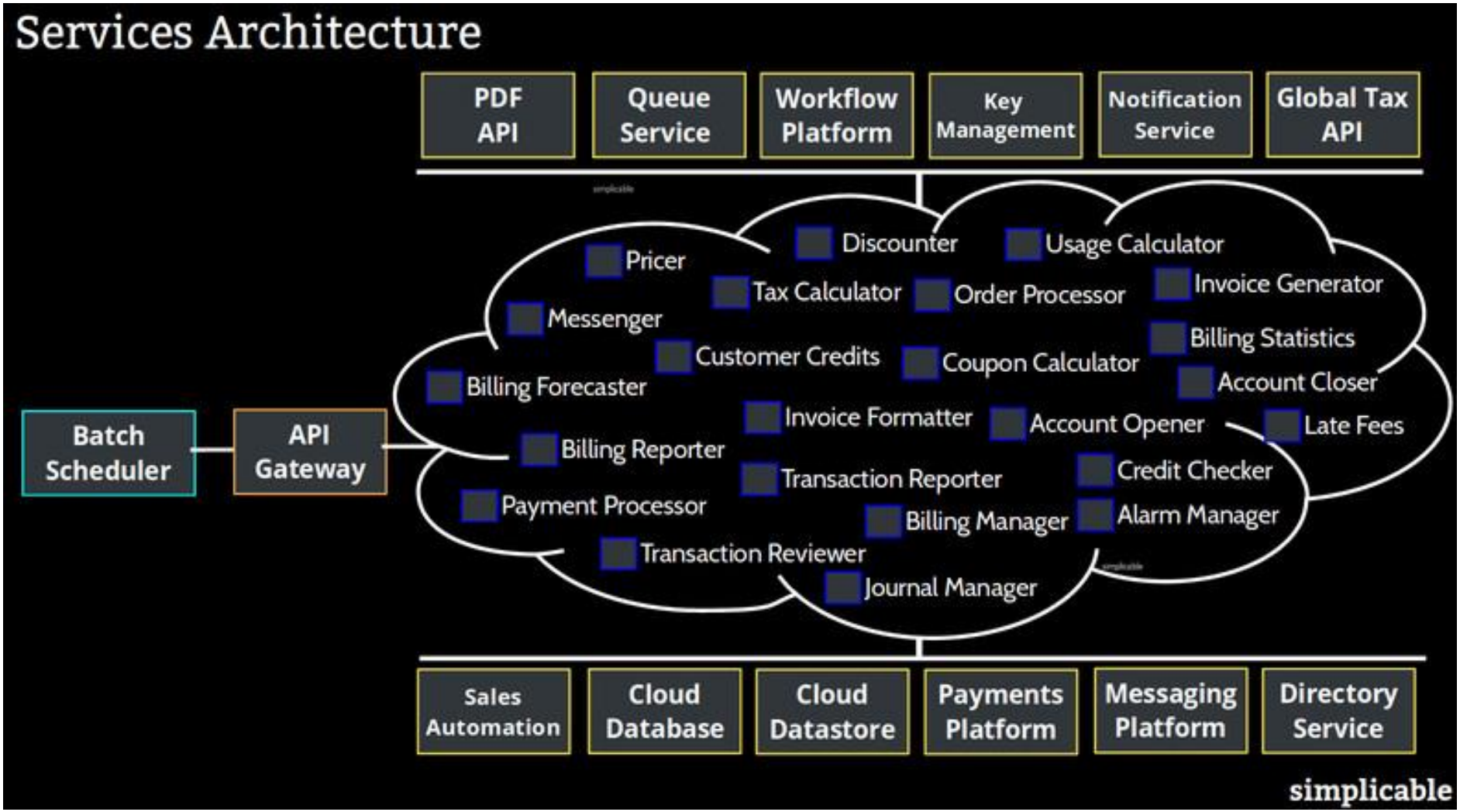
Existing Scenarios

- 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.

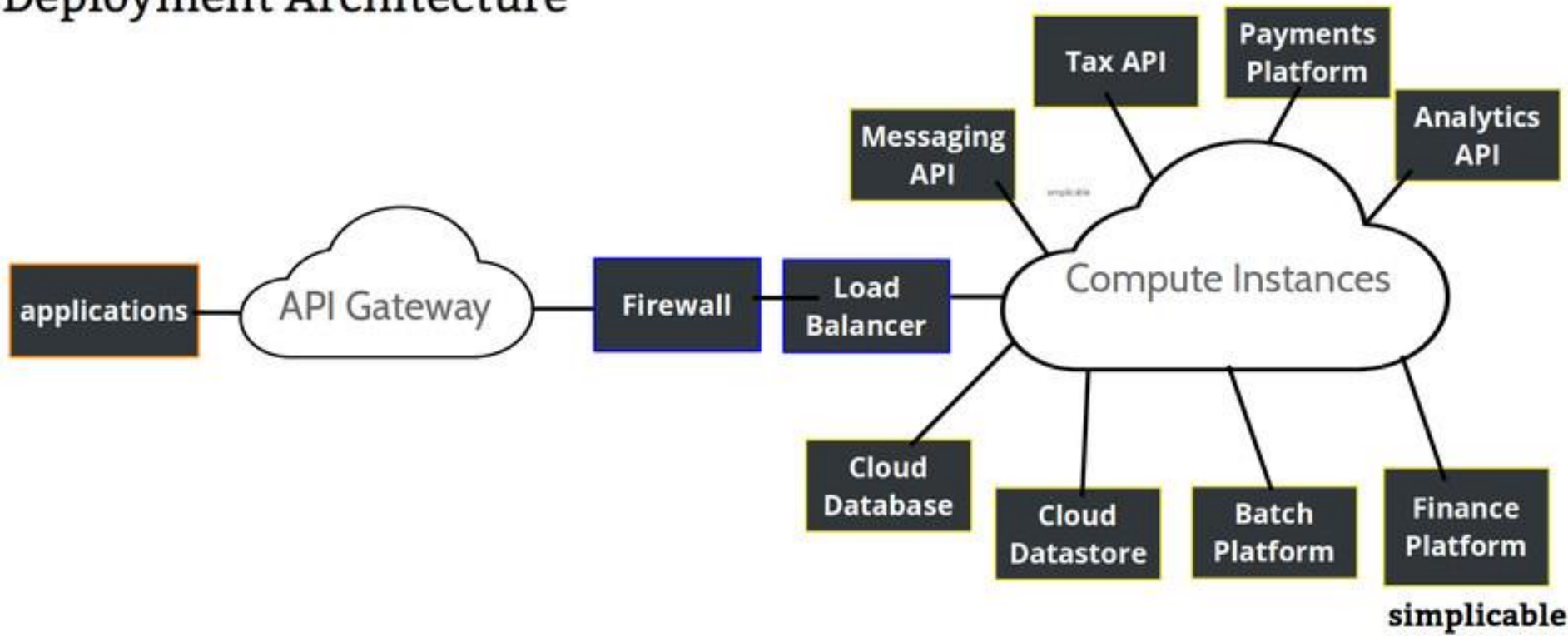


Microservices

Microservices



Deployment Architecture

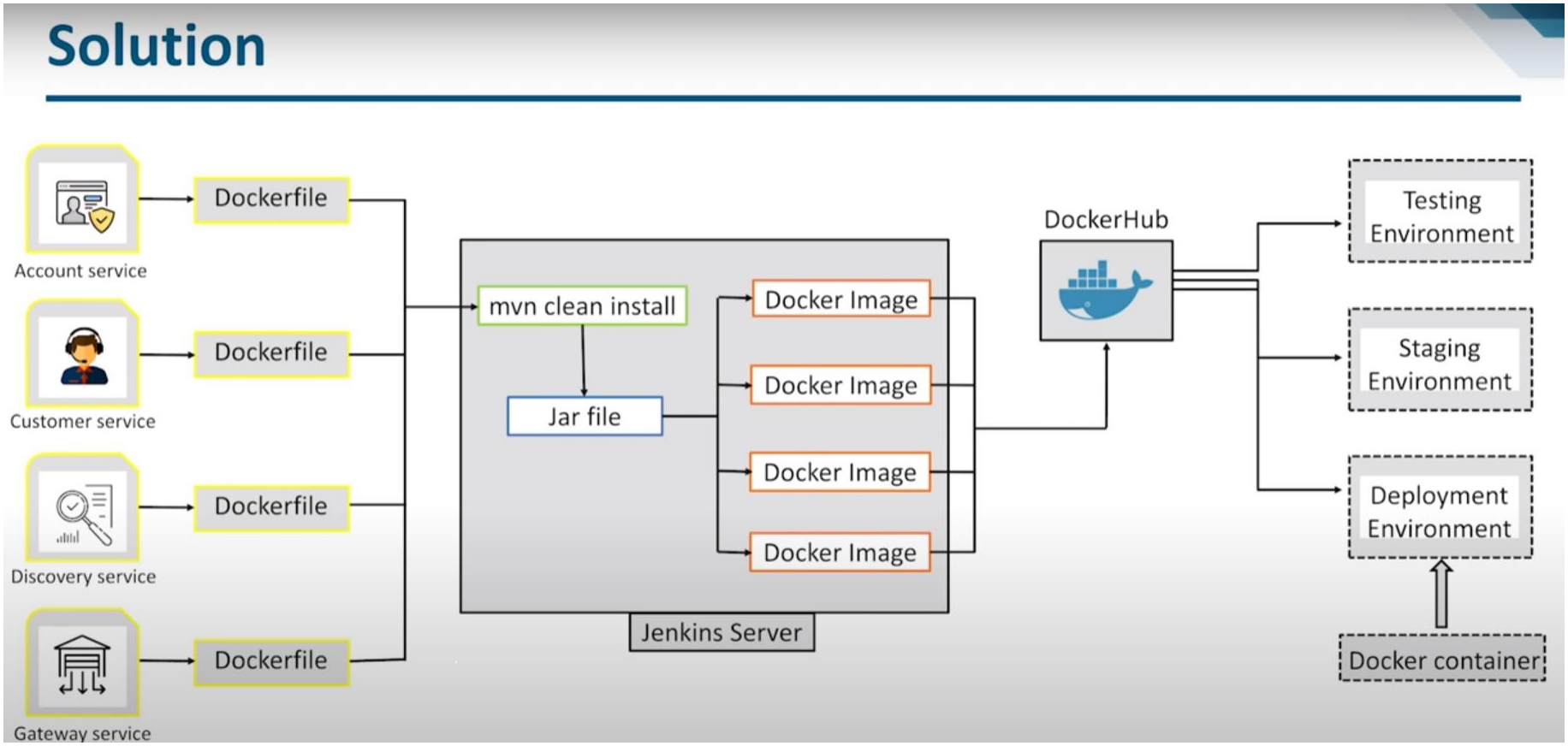


Principles of Micro service

- 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

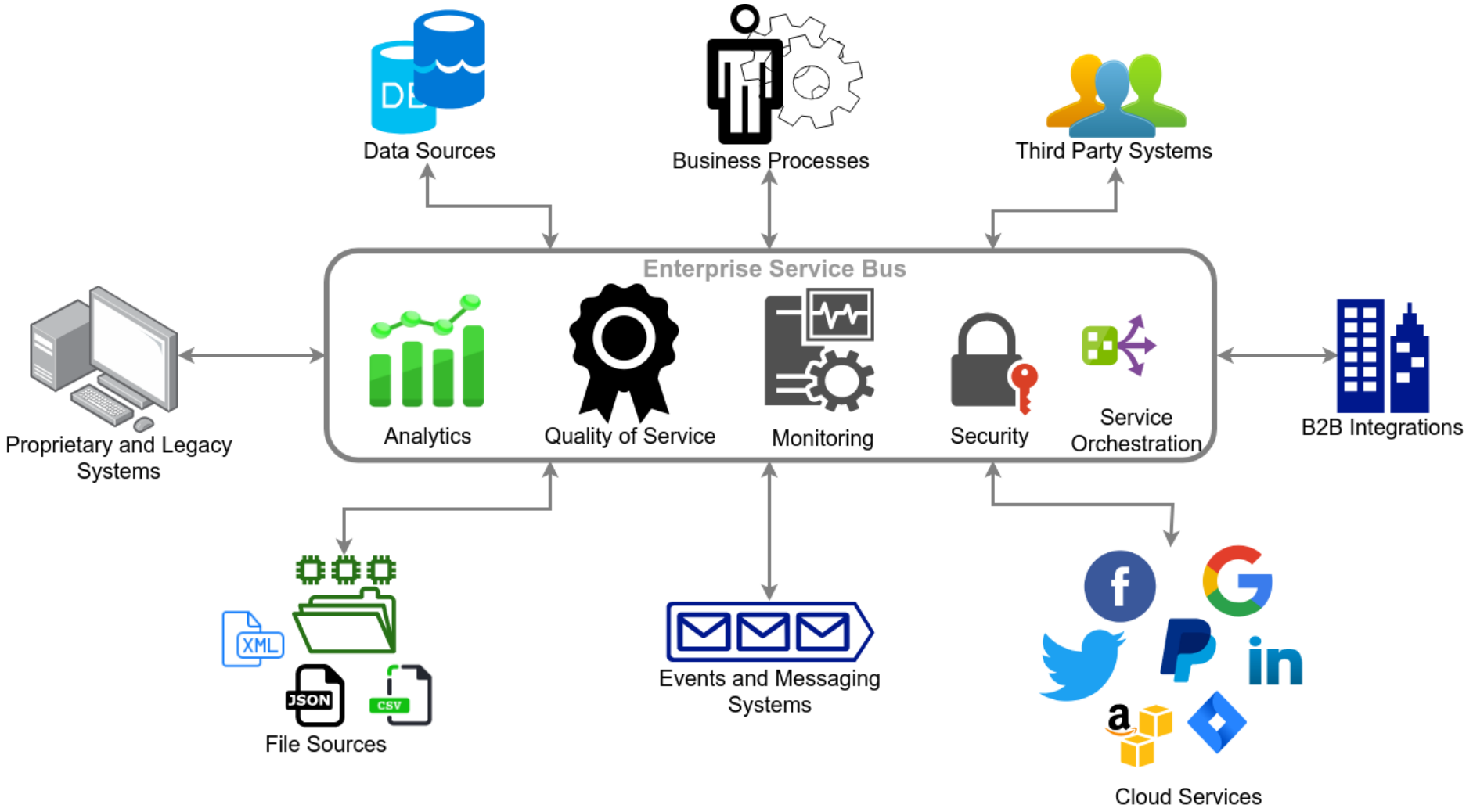
Microservices



API Integration ESB or Microservices

ESB for API Integration

Microservices

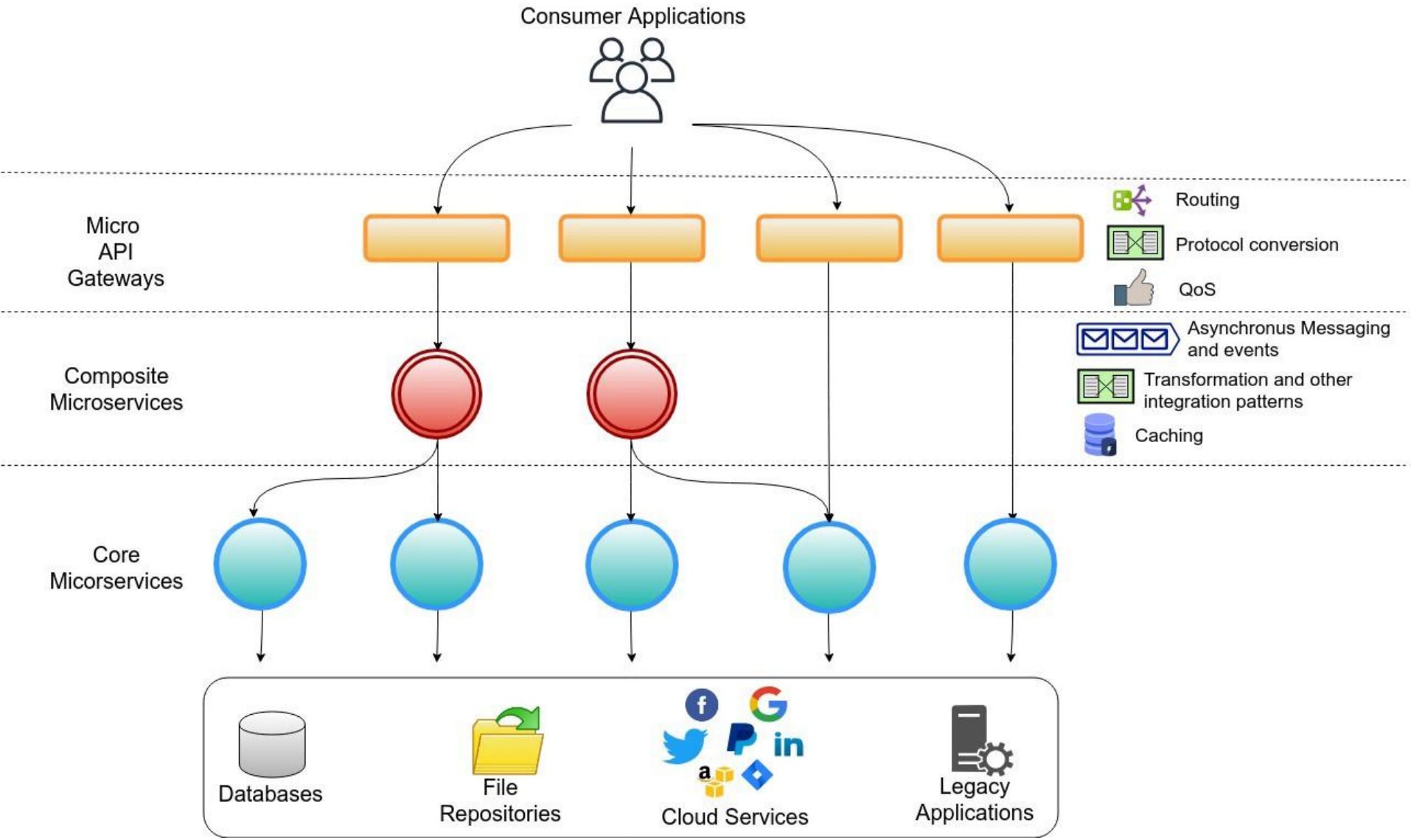


Disadvantages of ESB

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

Microservice for API Integration

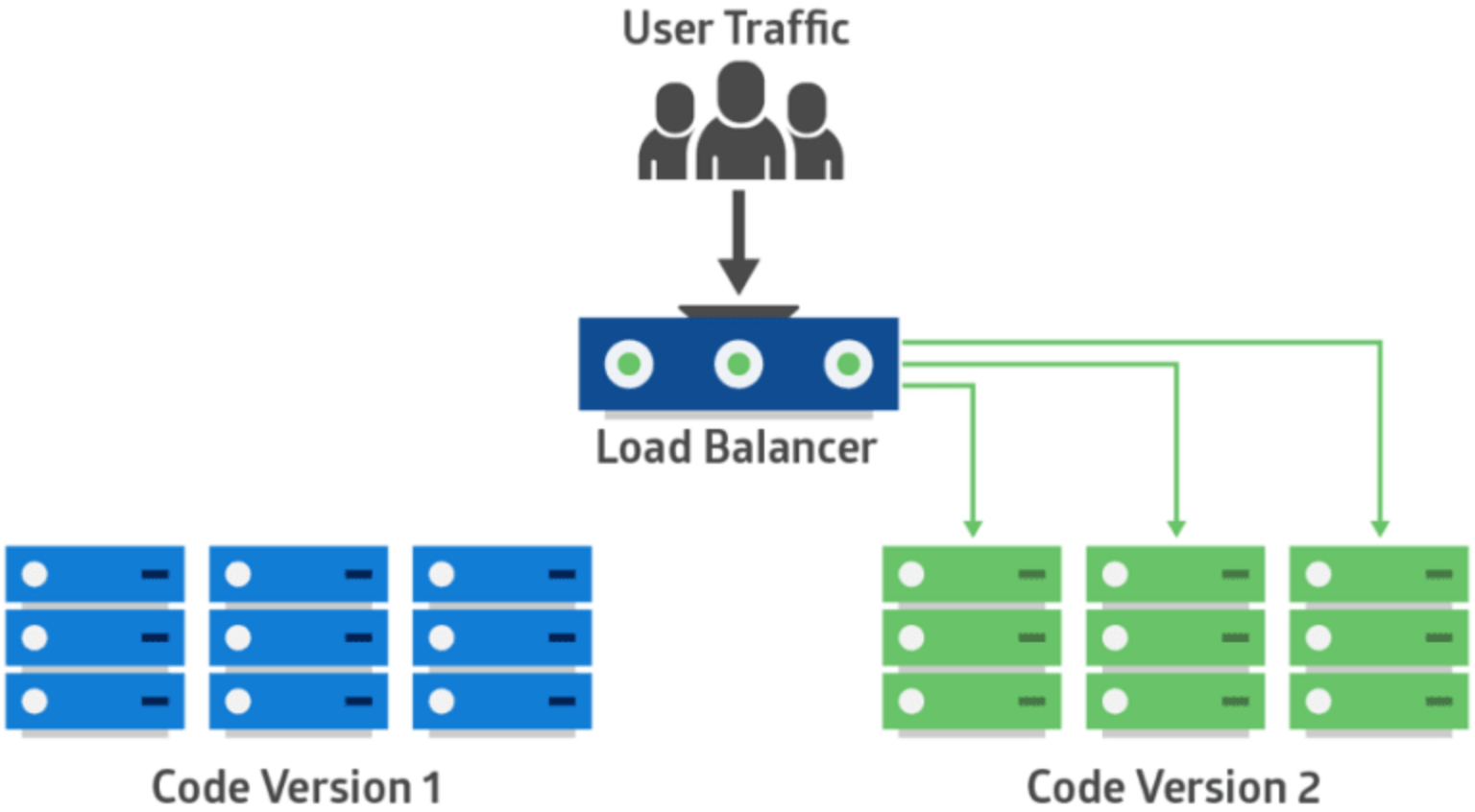
Microservices



ESB or Microservice

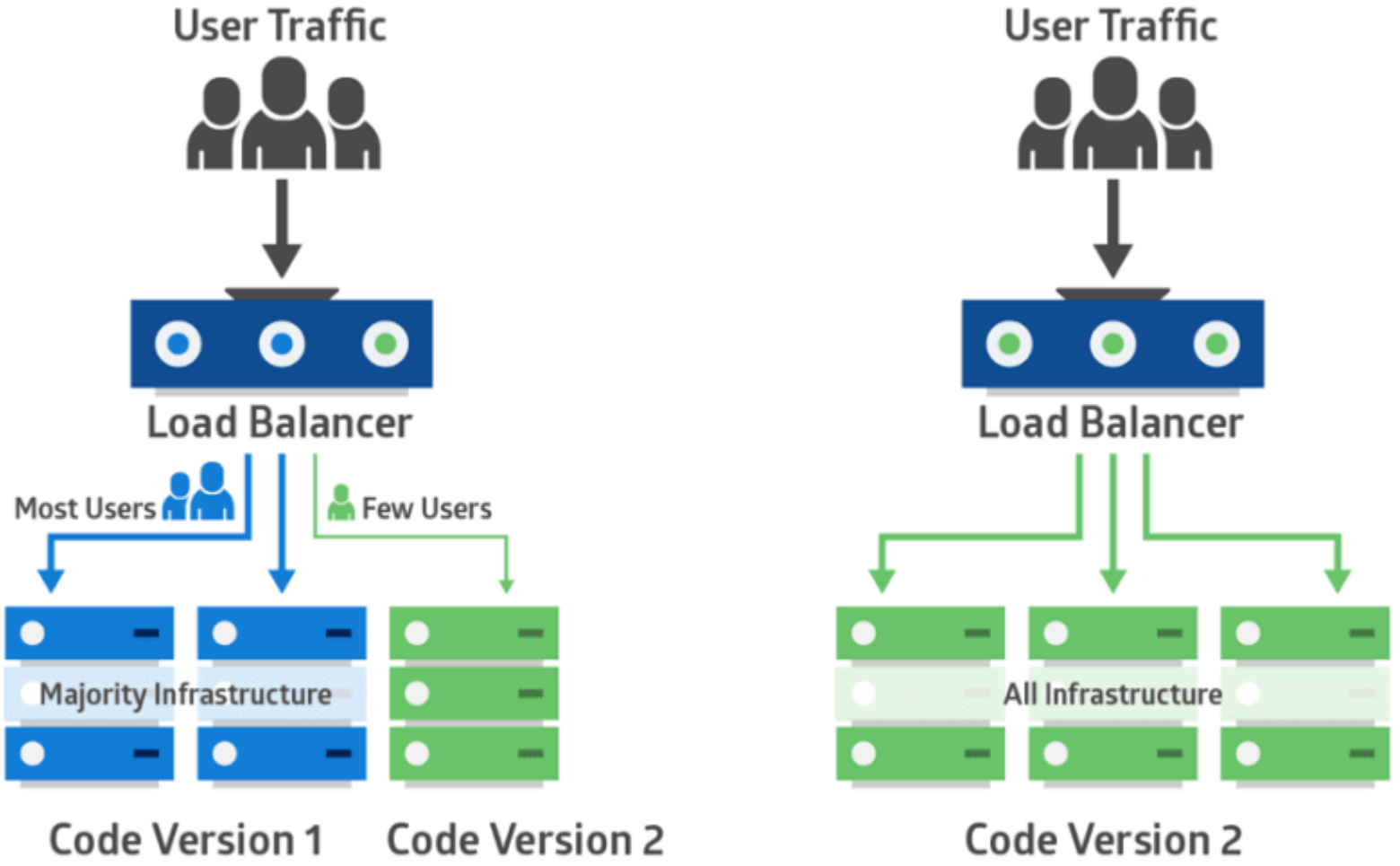
- When you select the suitable solution for the organization, following factors need to be assessed.
- Types of services to be integrated
- Communication protocols required
- Performance numbers
- Transaction requirements
- Security restrictions
- Inter-dependencies of services
- Business timelines
-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

Blue Green Deployment



Canary Testing

Microservices



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 *(Ref: https://www.12factor.net/)*	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: https://www.12factor.net/)*	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

Microservices

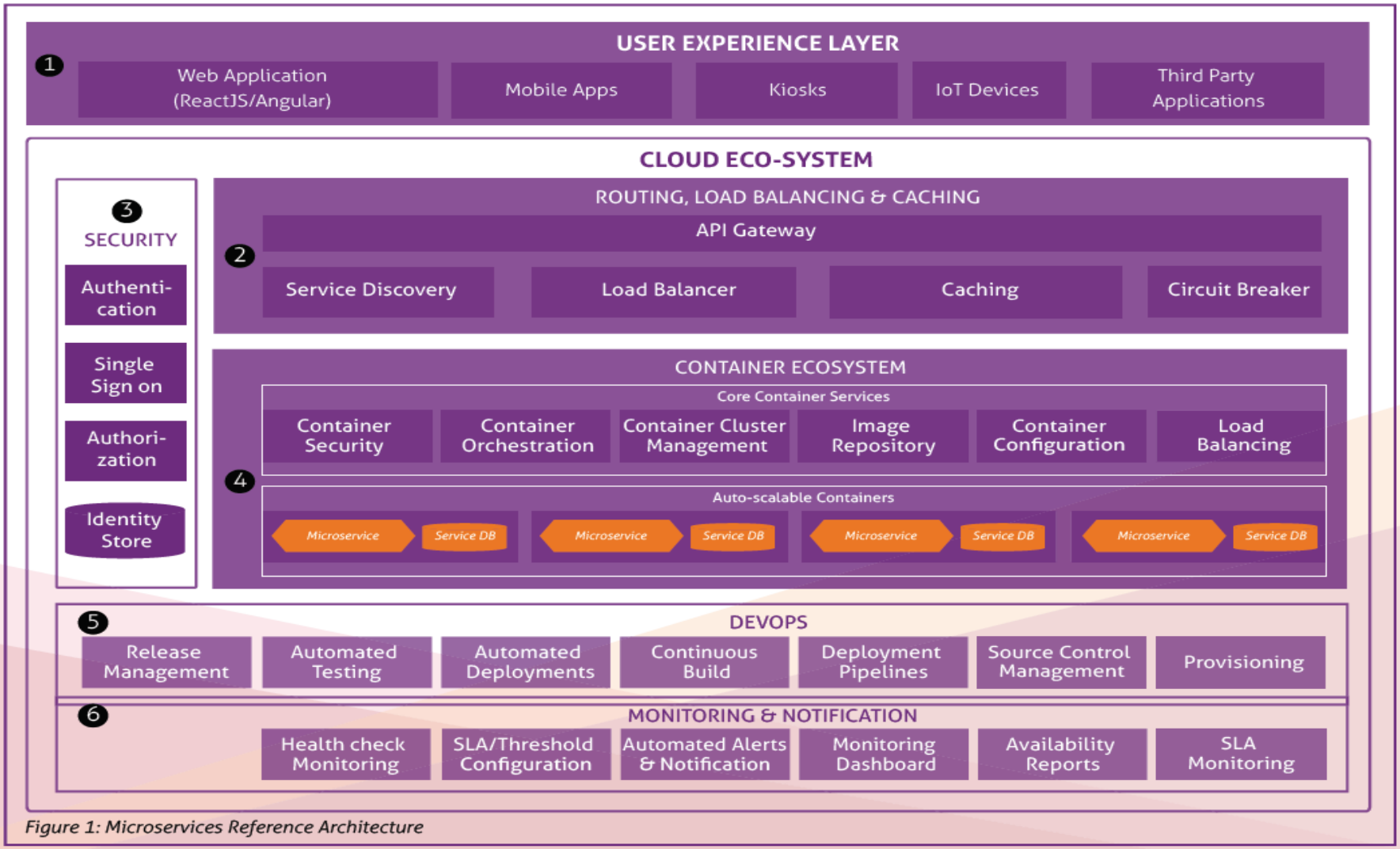


Figure 1: Microservices Reference Architecture

Sample Microservice Interaction

Microservices

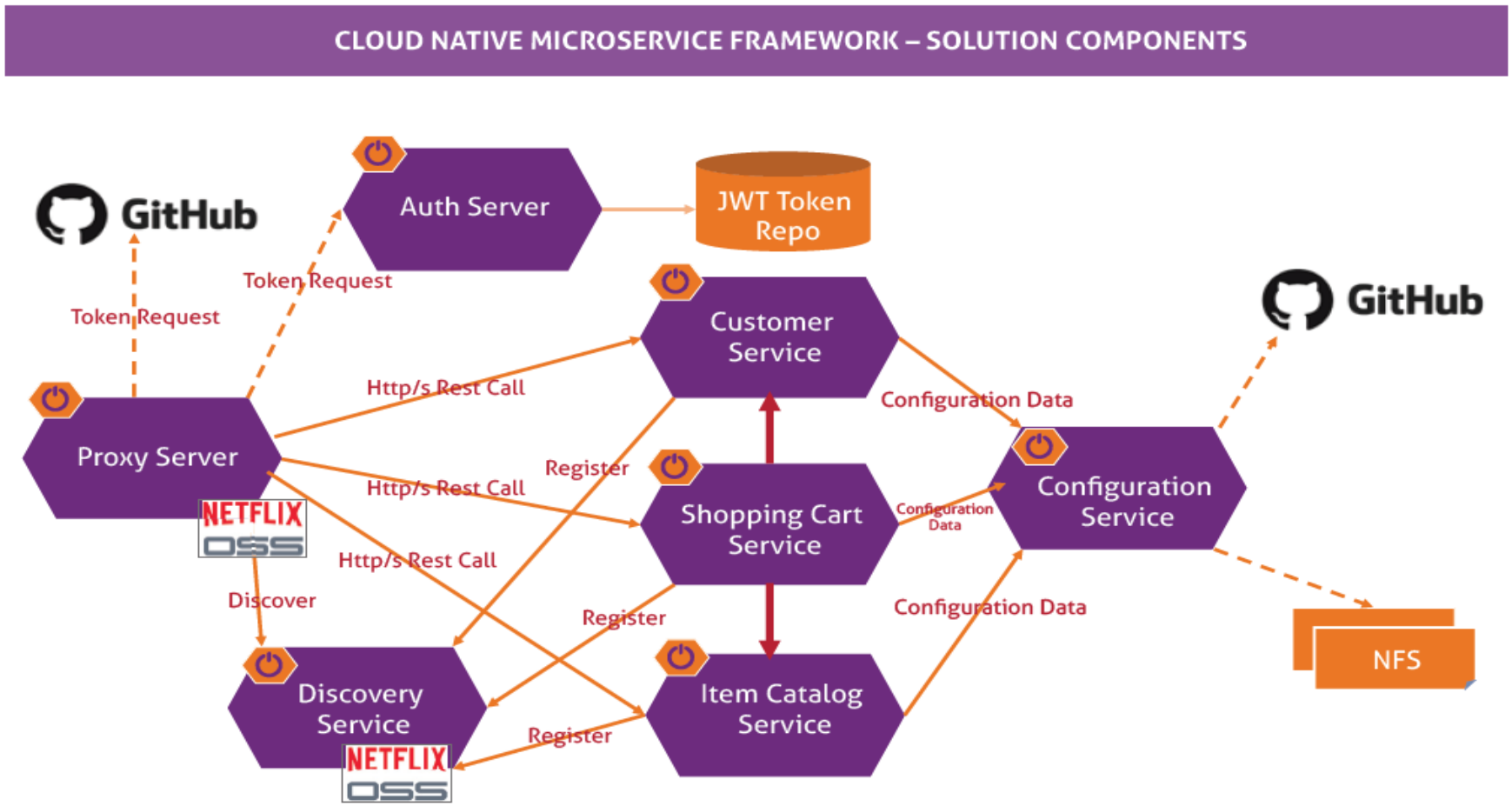


Figure 2: Sample Microservice Interaction Diagram

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 Sever	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

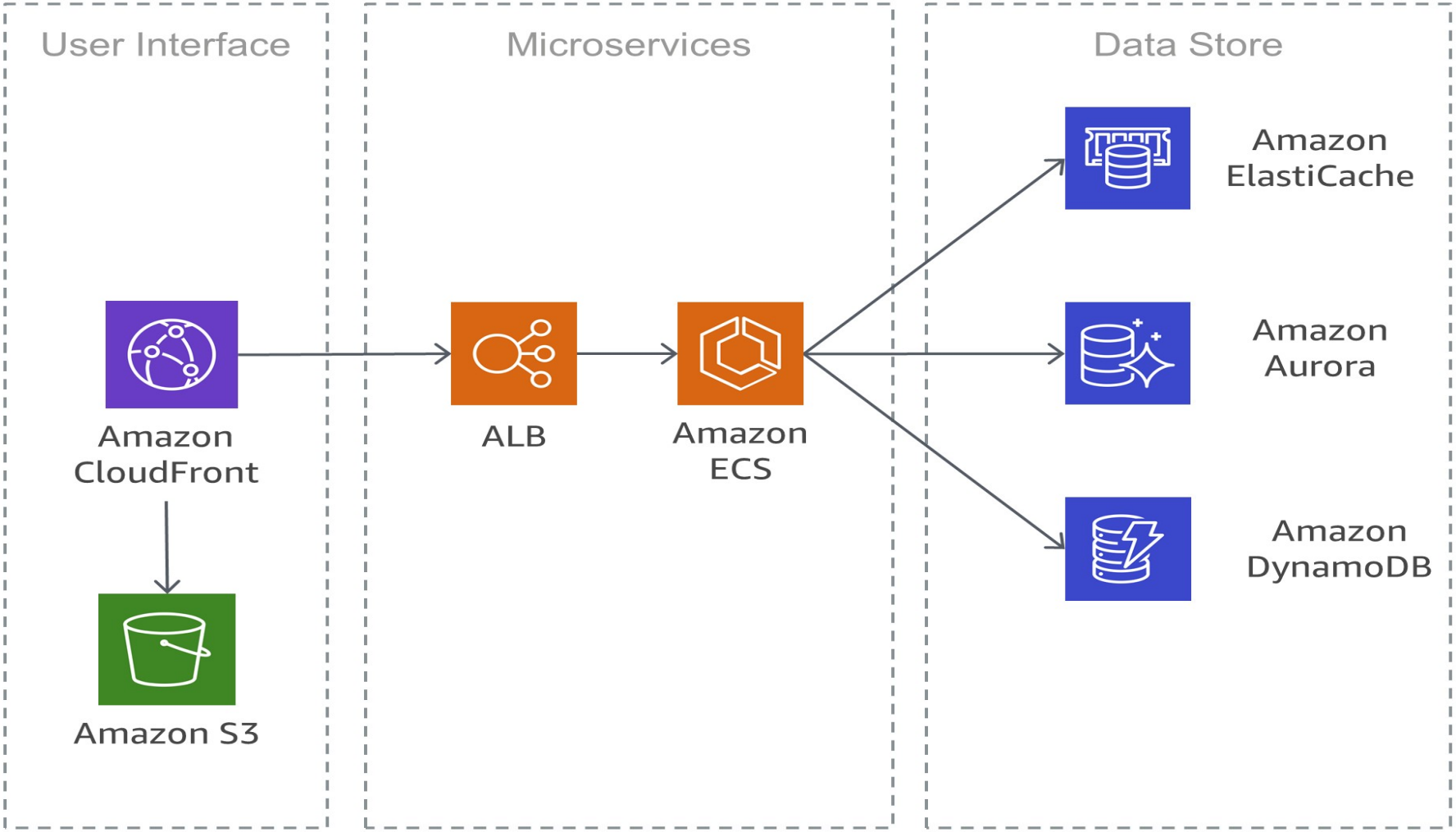
ARCHITECTURE COMPONENT	SAMPLE PRODUCT STACK
Api Gateway	Netflix OSS- Zuul
Authentication Service	Spring – Security Oauth2, OpenID Connect
Service Discovery	Netflix OSS- Eureka, Apache Zookeeper
Configuration Service	Spring – Cloud Config Server
Microservice	Spring – Boot, Vert.x, Dropwizard
Monitoring	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 orchesrator

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

Microservices

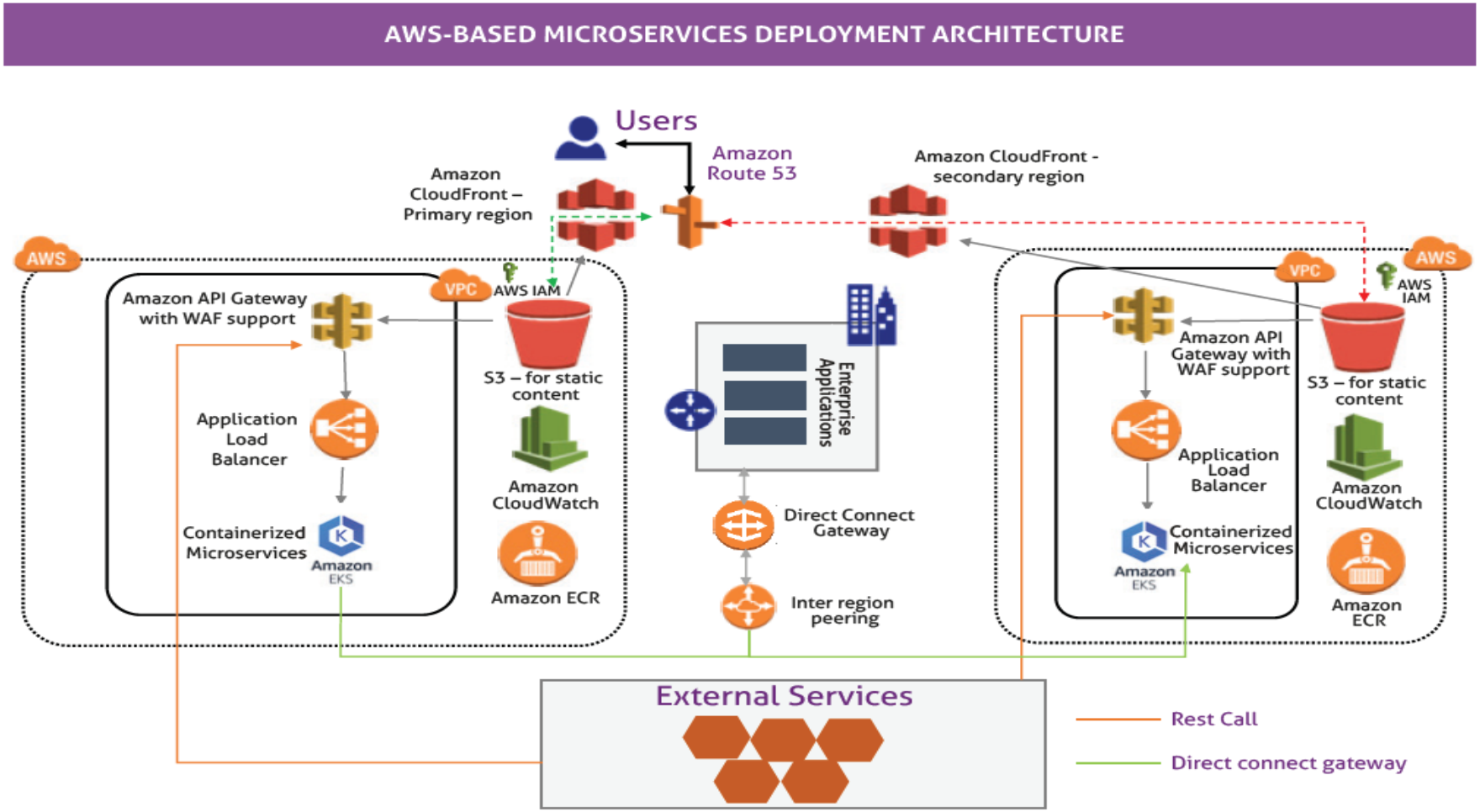


Figure 3: Sample AWS-based Microservices Deployment Architecture

Cloud Deployment Architecture - Azure

Microservices

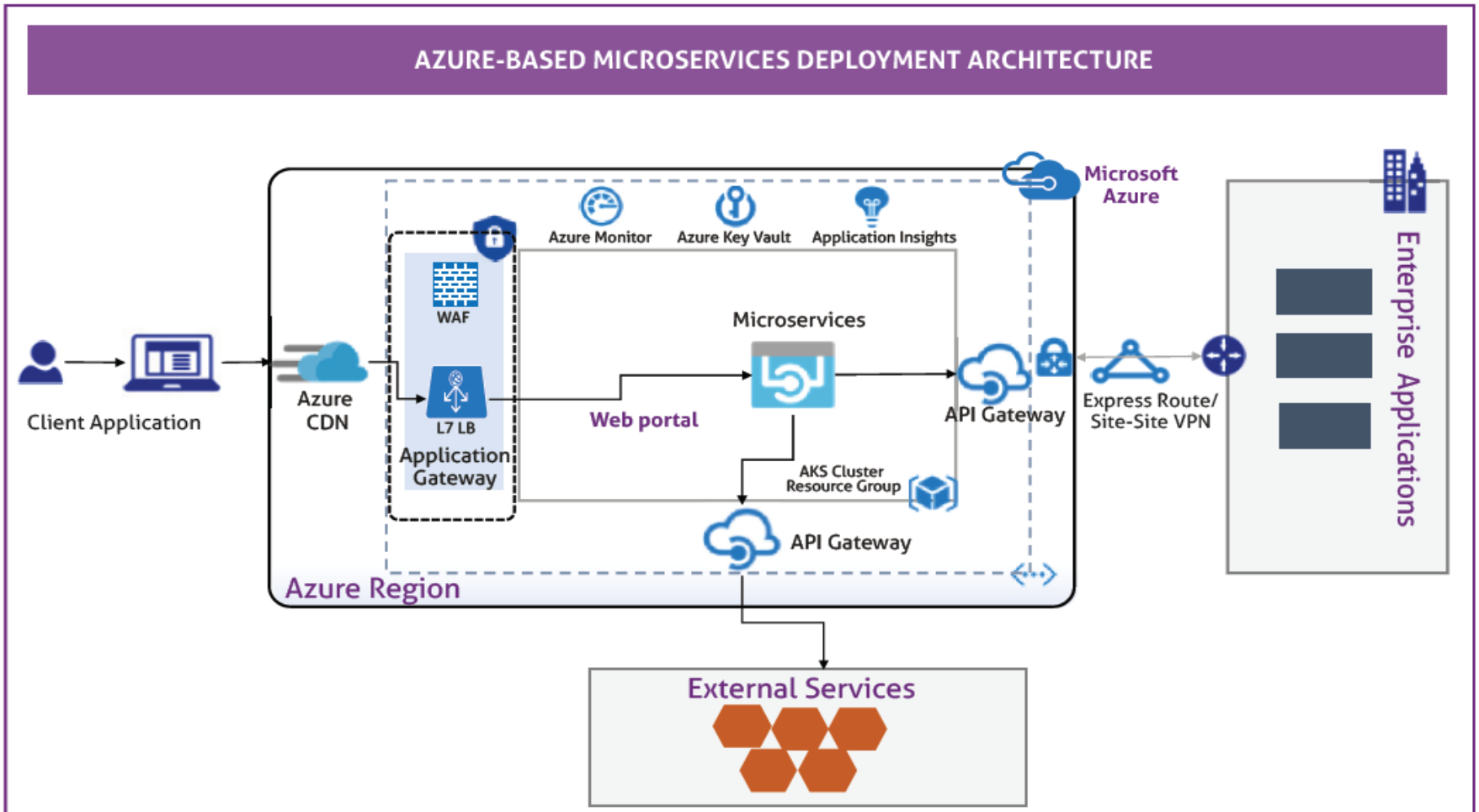
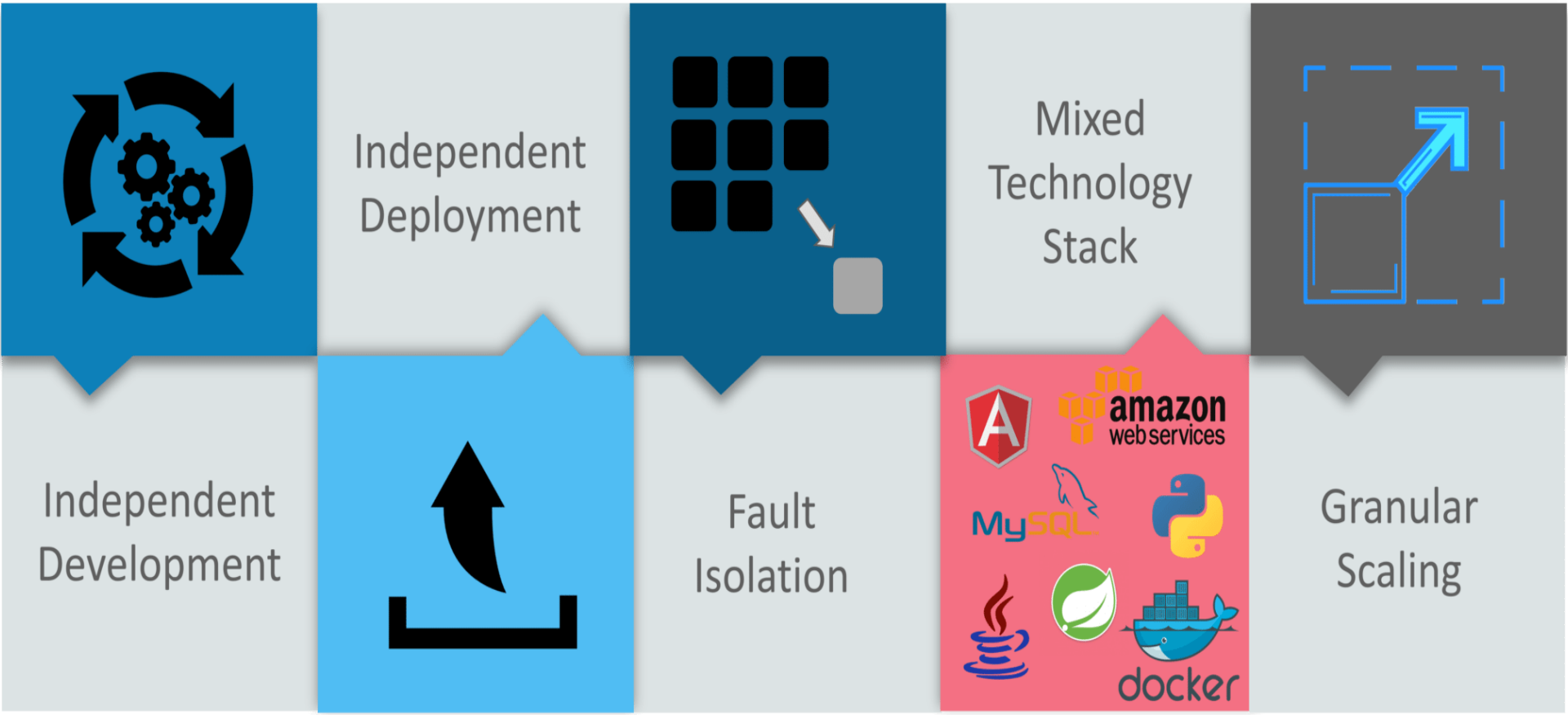


Figure 4: Sample Azure-based Microservices Deployment Architecture

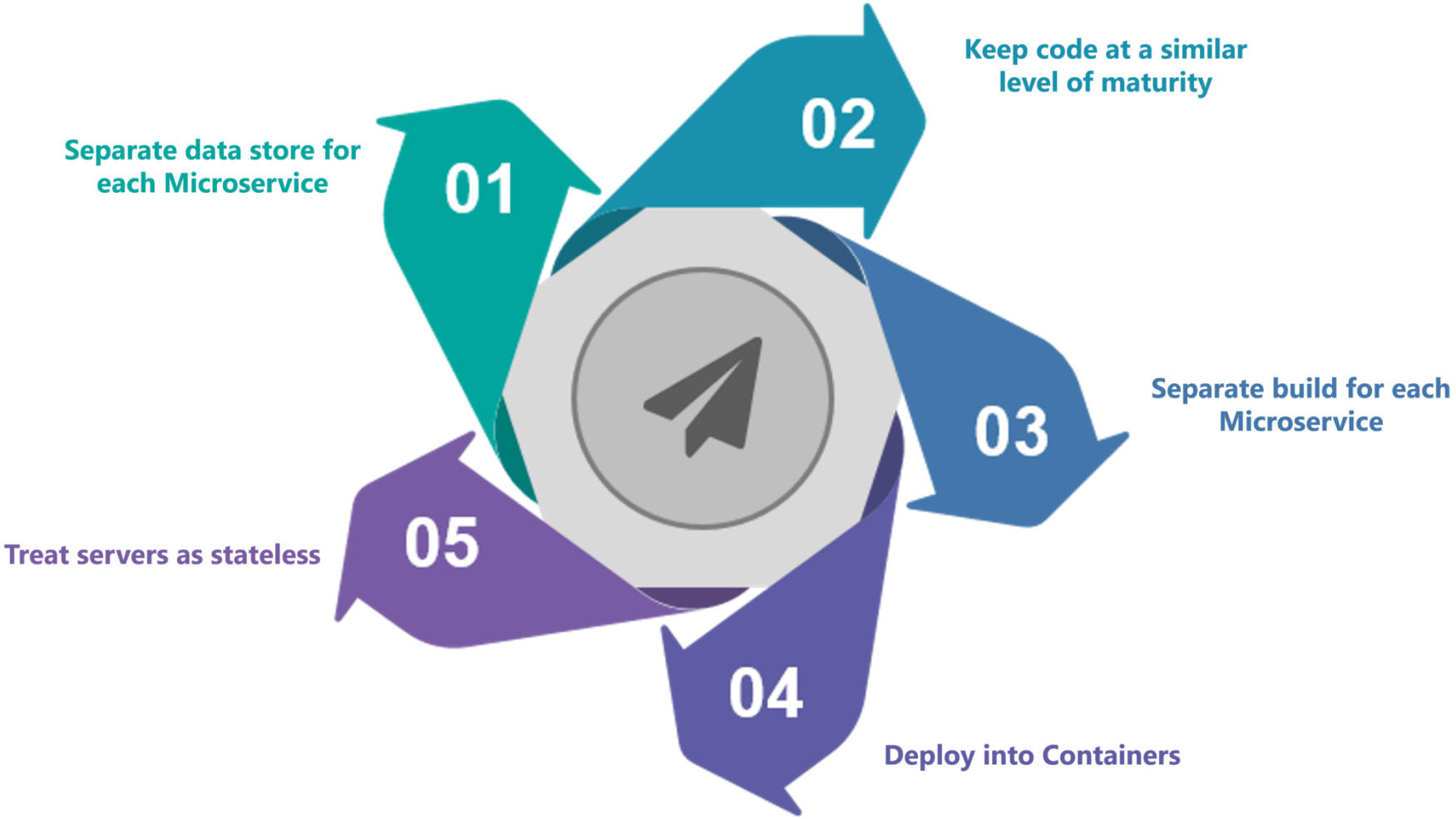
Microservices Features

Advantages Of Microservices

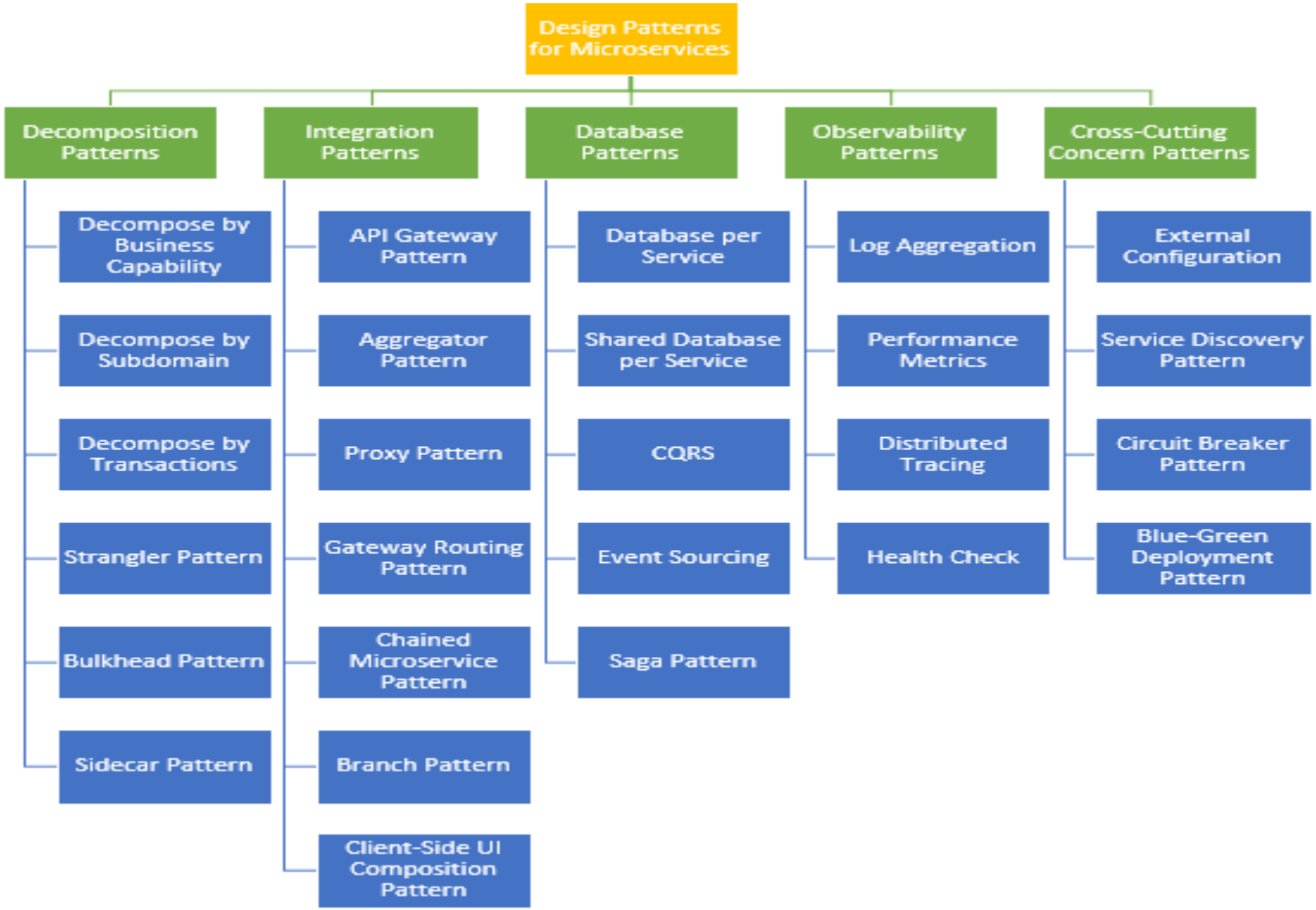


Best Practices To Design Microservices

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

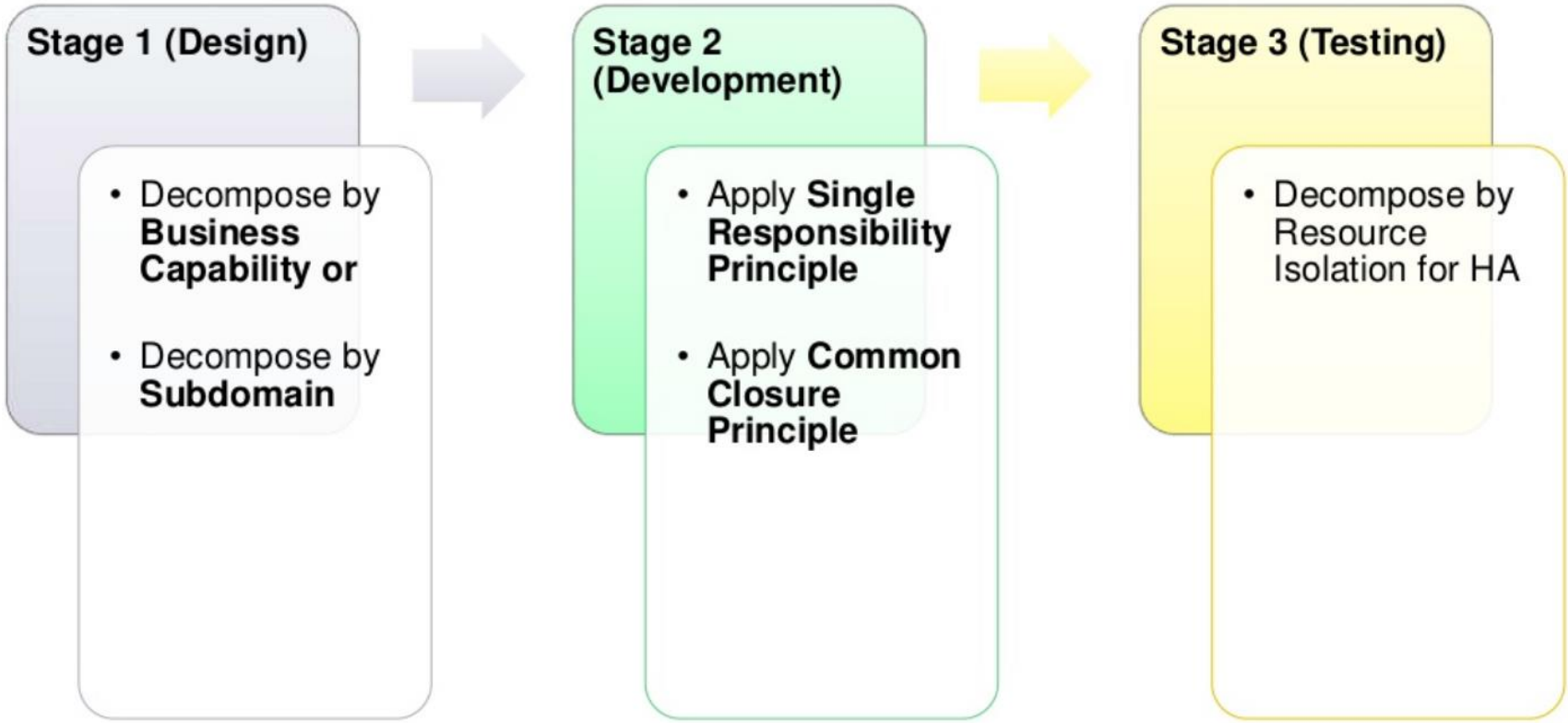
- **Solution**
- One strategy is to decompose by business capability.
- A business capability is something that a business does in order to generate value.
- The set of capabilities for a given business depend on the type of business.
- For example, the capabilities of an insurance company typically 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

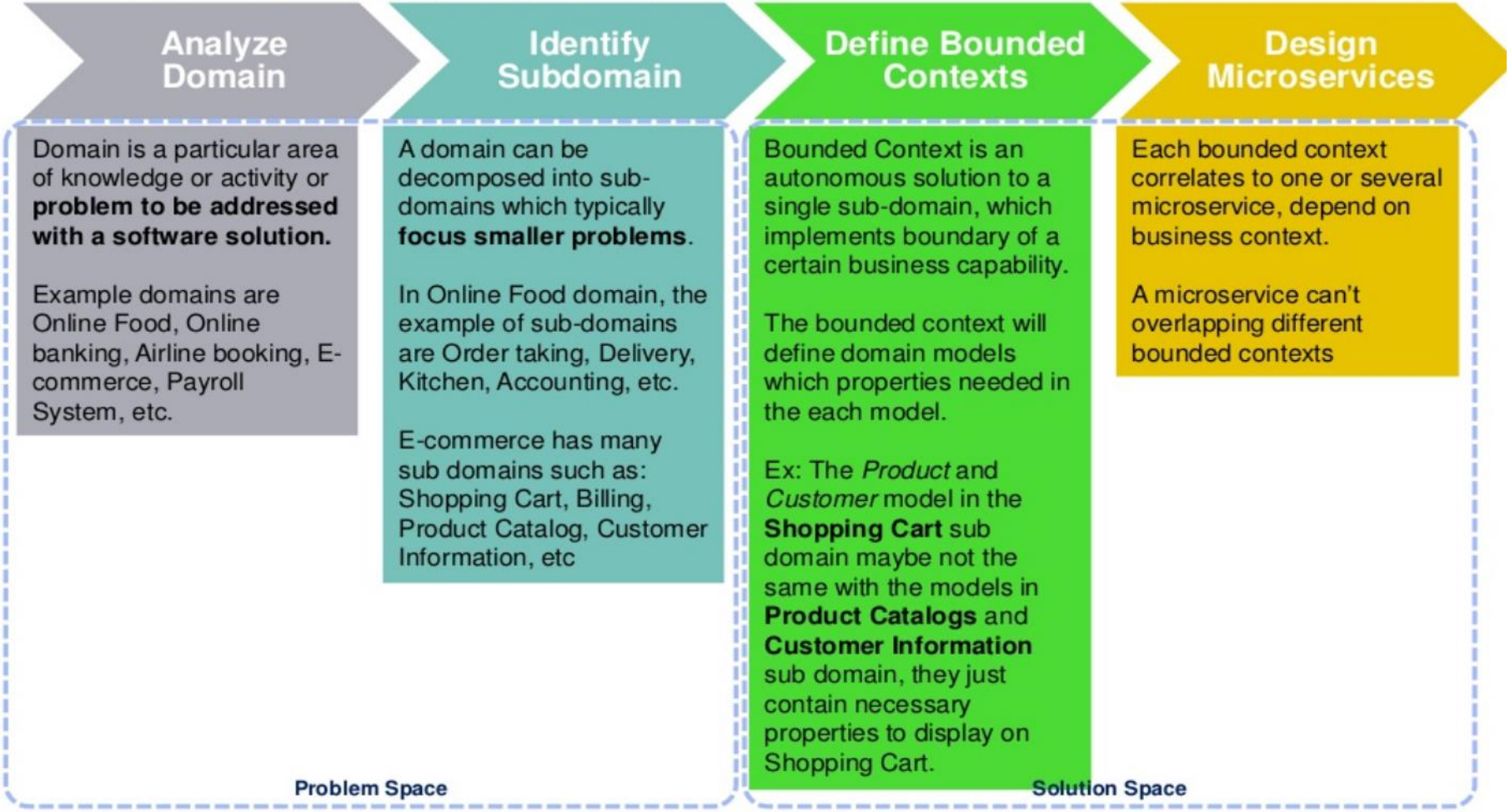
Decompose by Subdomain

- 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

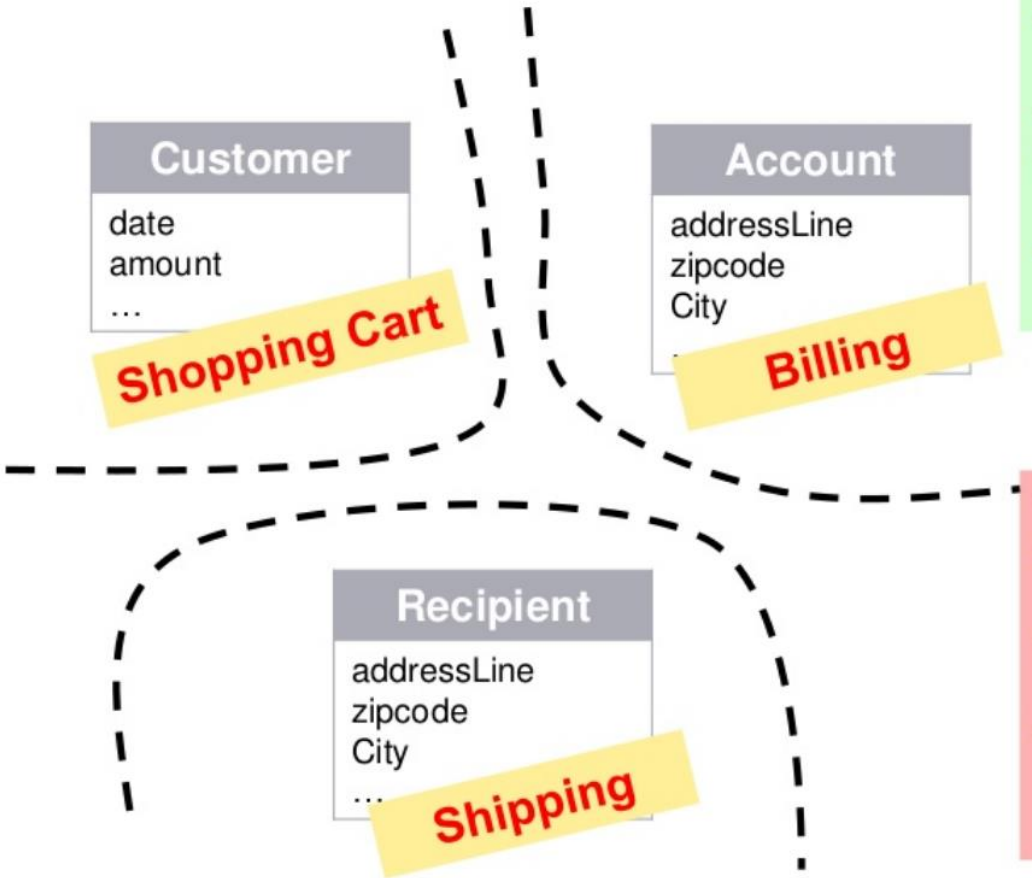
Domain-driven design help to design Microservices that loosely coupled and cohesive.



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

Microservices

2

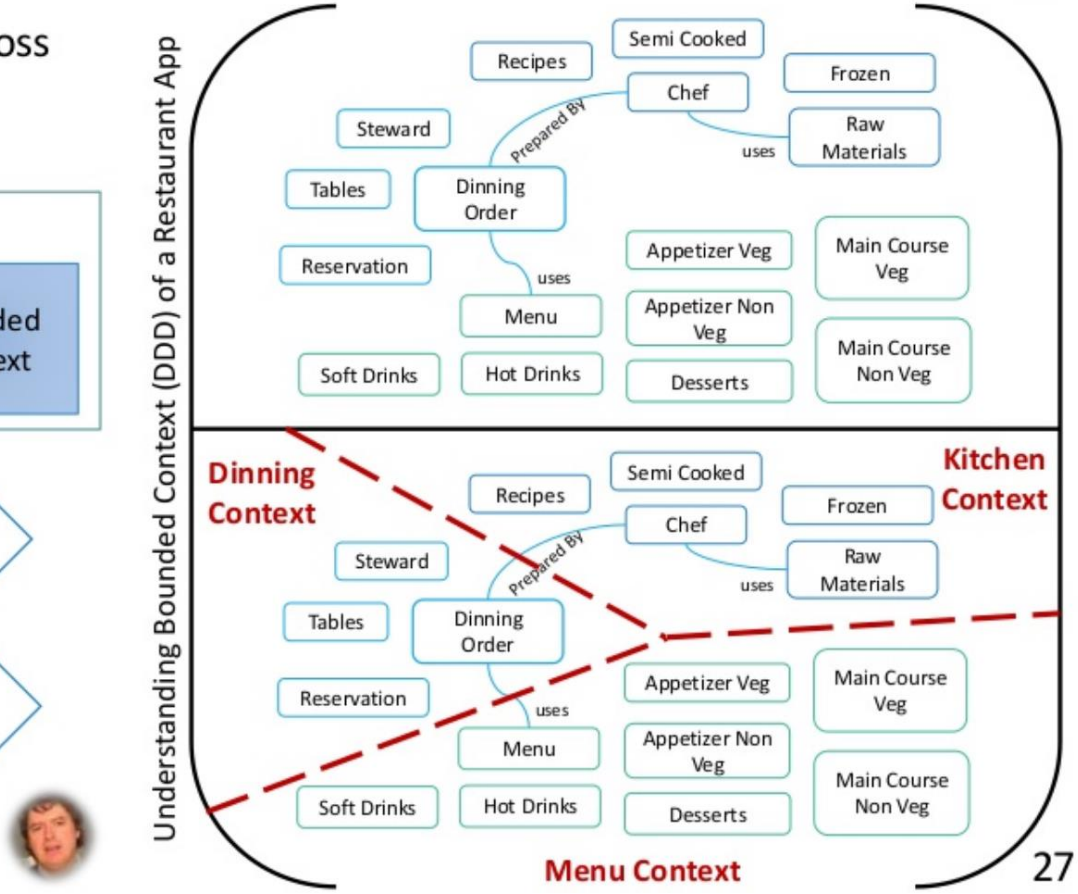
DDD: Bounded Context – Strategic Design

Clip slide

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

8/10/2018

27

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/{}).

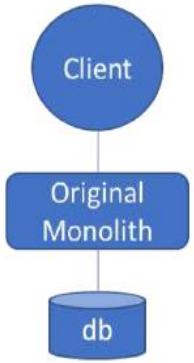
Strangler Pattern

- 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.

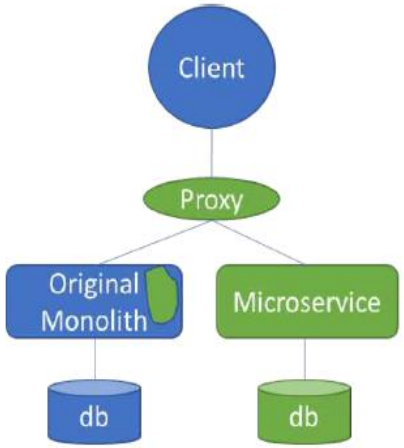
Strangler Pattern

Microservices

Transform – Create a parallel microservice



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



Eliminate –eliminate the legacy module