Microservices

Enterprise Application Development

Agenda

01 Monolithic Application Architecture

02 Microservices Architecture

03 Monolith vs Microservices Architecture

04 Characteristics of Microservices Architecture

05 Microservices vs SOA

06 Communication between Microservices

07 Advantages & Disadvantages of Microservices

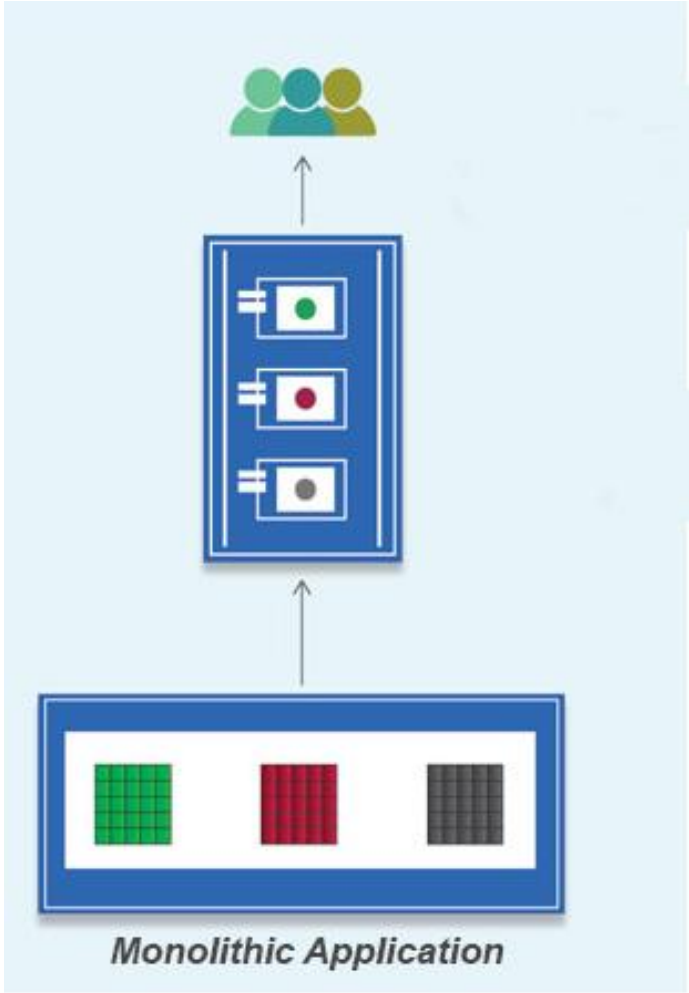
IFS

Monolithic Application Architecture

Monolithic Application Architecture

Monolithic Application

- All business capabilities / modules will run in a single process. This may be a WAR / EAR file.
- Classes of each of those business capabilities / modules may communicate with each other and may share same libraries.
- One module may be a library or dependency for another module.
- Connects to one single database containing tables related to all business capabilities



IFS

Monolithic Application

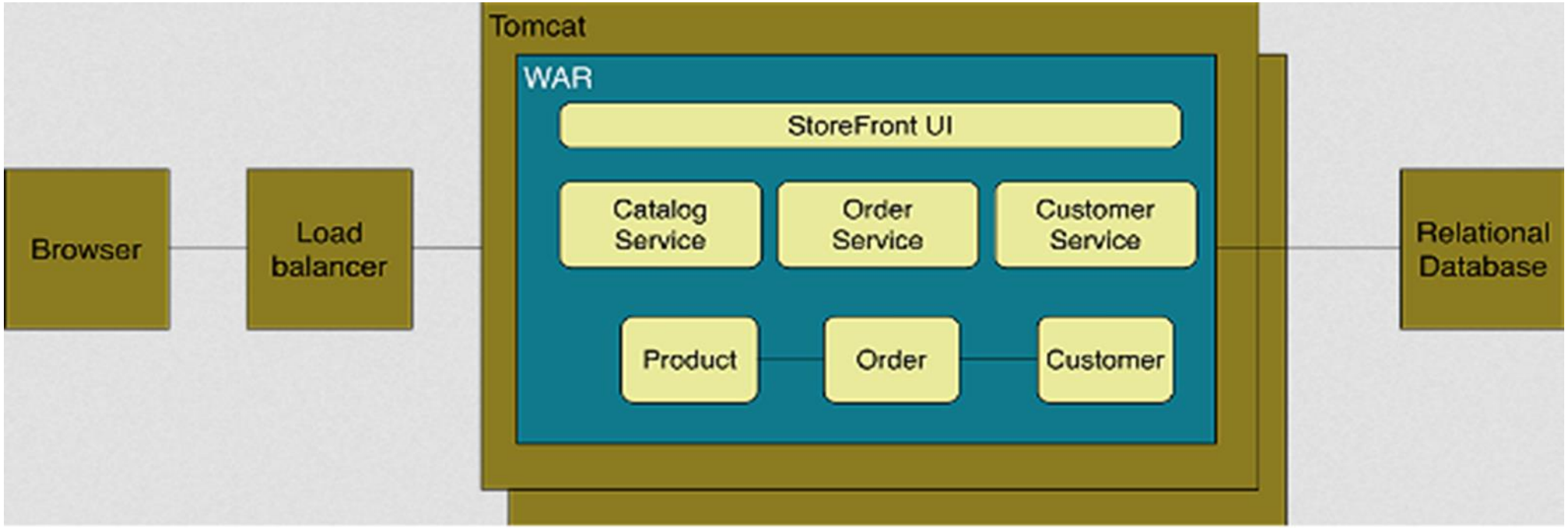
Architecture

- A Monolithic application is built as a single unit.
- It includes all its functionality into a single process.
- May contain 3 tiers such a client tier, server (middle/business) tier and database tier.

Monolithic Application

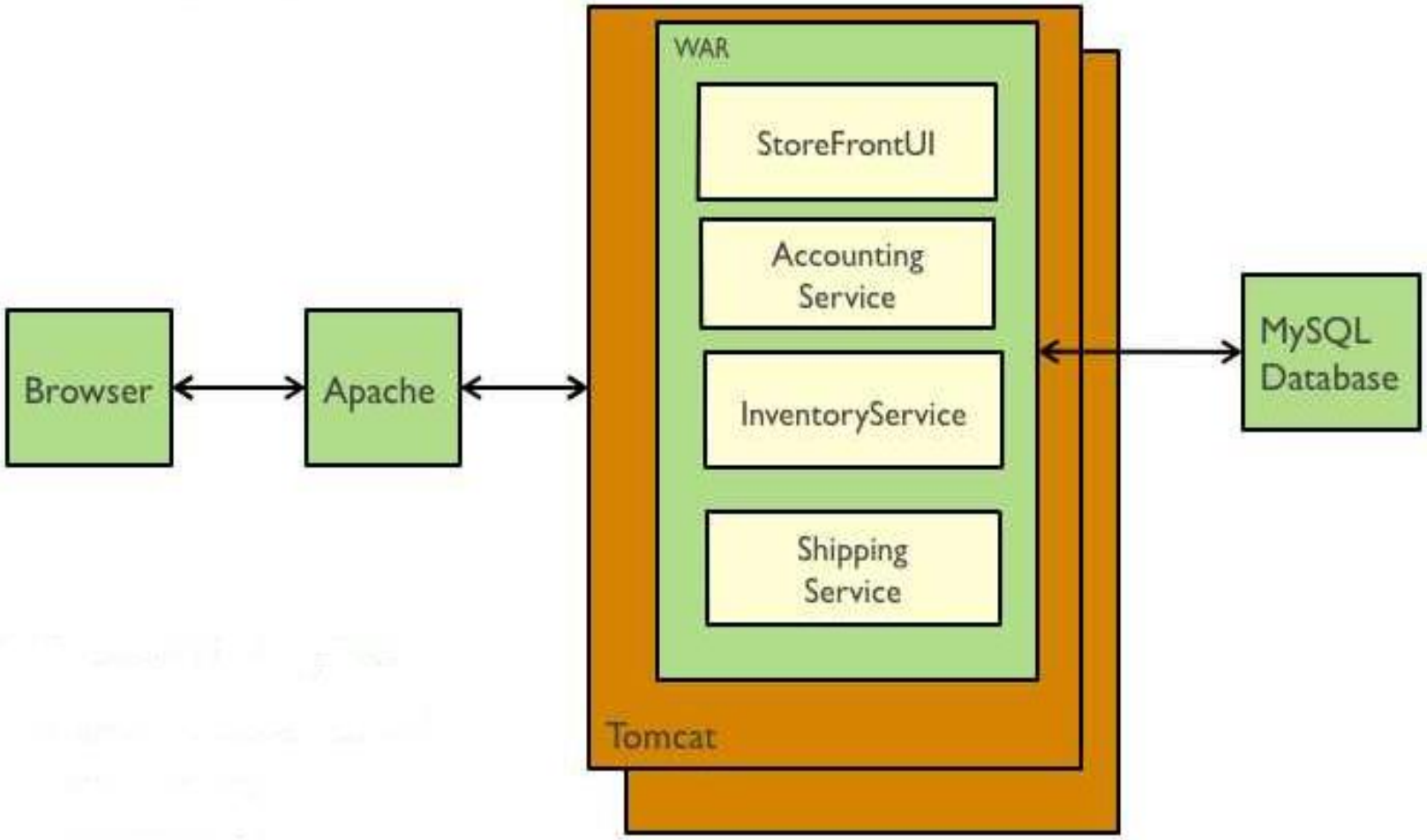
Architecture

- The Server Tier may contain all the business logic which spans across various business domains across the application.



Monolithic Application

Architecture



Monolithic Application

Architecture

- These business logic may be written in classes, and they may be grouped into namespaces or packages according to the business capabilities or technical boundaries.

  package com.example.shoppingcartservice

  package com.example.usermanagement

Monolithic Application

Architecture

- Function in a particular business domain may call another function of a different business domain.

```
9
10  public class ProductCatalog {
11
12      Public Product viewProduct(int id) {
13          ...
14          ...
15          RecommendationEngine recEngine = new RecommendationEngine();
16          recEngine.getRecommendedProducts(category);
17          ...
18          ...
19      }
20  }
```

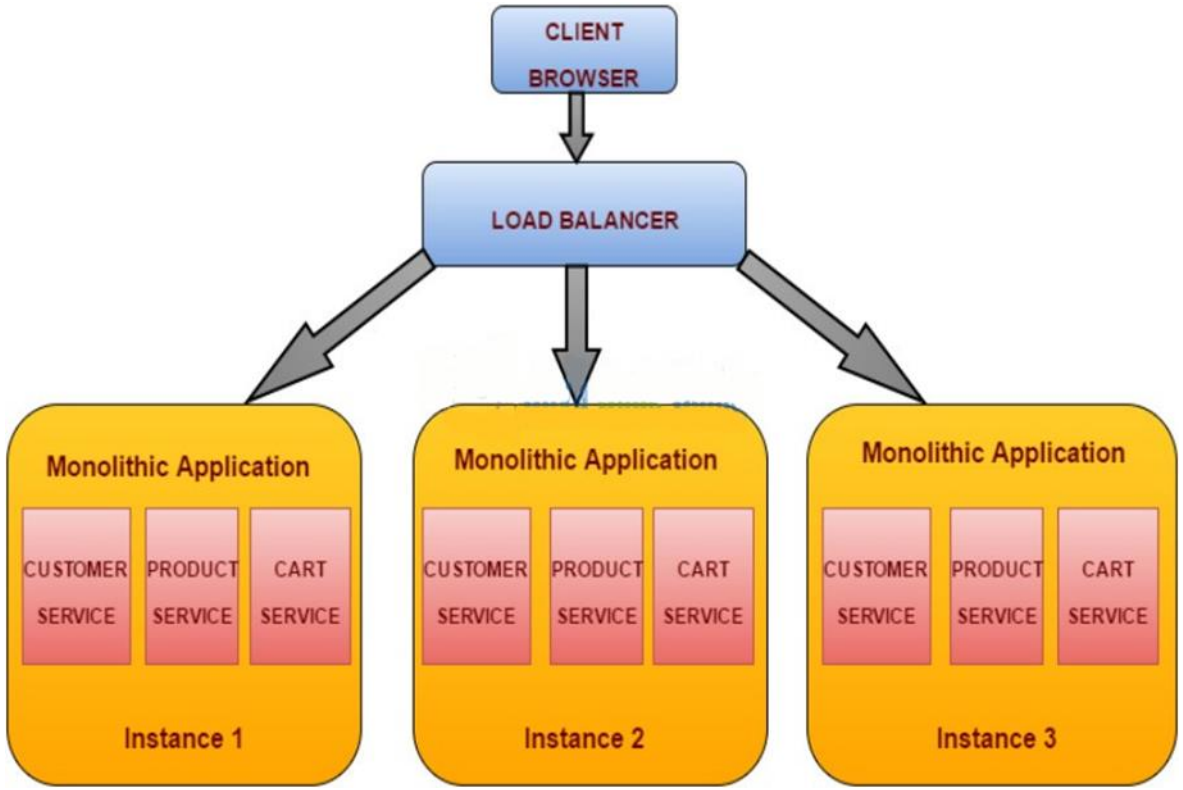
- This introduces coupling and interdependencies among the business modules of the enterprise application.

IFS

Monolithic Application

Architecture

- Monolithic application can be horizontally scaled by running many instances behind a load balancer, but each instance contain all functionalities even though there are components having less demand.



Monolithic Application

Architecture

- Monolithic application can be hosted in different geographical locations.

- However, the entire application modules has to be deployed even though some of the modules do not have high demand.

Monolithic Application

Architecture

- Monolithic architecture approach suits most of the software applications but have become inadequate for some large enterprise applications.

- Some of the major worldwide business companies such as Netflix and Amazon has migrated from Monolithic architecture to Microservice architecture.

Advantages of Monolithic Architecture

- Simplicity
- Easier code Base Access
- Easier inter- module refactoring
- Easier Deployments

Monolithic Application Architecture

Simplicity

- Monolith is relatively a simple and familiar architecture.
- Will suit for most of the simple and enterprise applications as well and we have been using it to solve the problems with the help of SOA.
- Can be implemented with small amount of resources (people & technology).
- If properly designed there will be no (or less amount of) redundant classes.
- Solution for the entire application will be one single unit (JAR / WAR / EAR).

Monolithic Application Architecture

Easier Code Base Access

- All business logic will reside in one single place.

- No need to maintain different or multiple version control systems to maintain the codebase.

Monolithic Application Architecture

Easier Inter-Module Refactoring

- You can do changes to the code base which could affect multiple modules.

  (You can move classes from one module to another)

Monolithic Application Architecture

Easier Deployments

- Final delivery of a Monolithic application server will most probably be a single deployment unit such as a JAR, WAR or EAR file.

- So you just have to worry about deploying one single file.

Note: Some may argue that it is easier to deploy microservices since it concerns about one single business capability. So there is nothing right or wrong about these arguments since it depends on the task and the expectation of the person.

Microservices Architecture

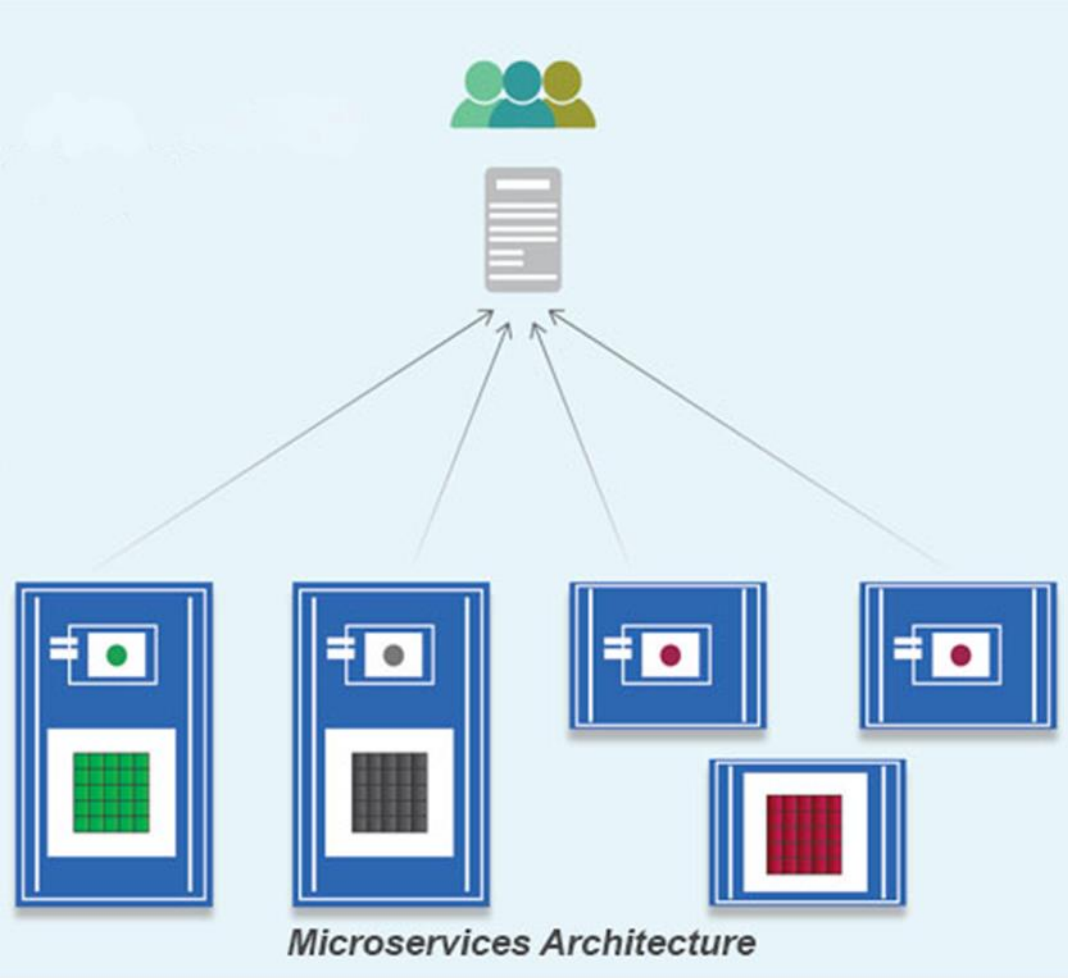
Microservices

Overview

- A self-contained process that provides a unique business capability.
- An approach for 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.
- Built around business capabilities and independently deployable.
- Each Microservice may be written in different programming languages and use different data storage technologies.
- Microservices communicate with each other through a well-defined interface such as REST or Message Queues.
- Every Microservice is responsible for its own data model and data.
- Microservice splits individual capabilities of a Monolithic architecture and implement within its own process.

Microservices

Architecture

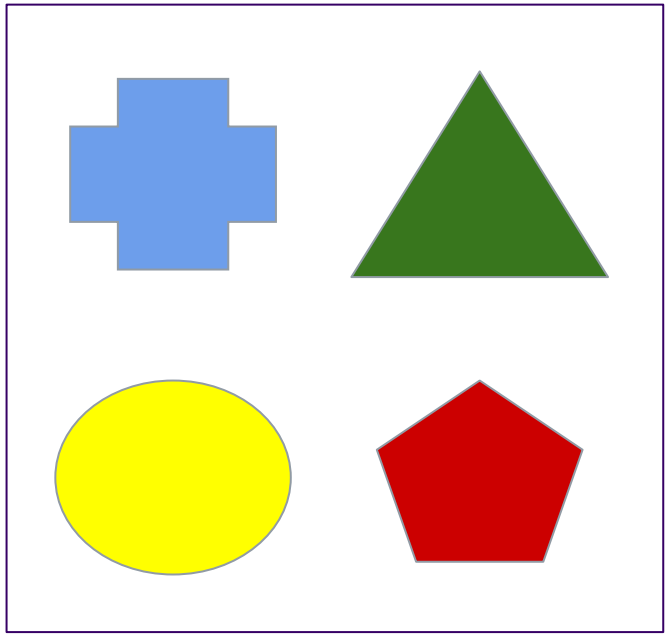


Monolith vs Microservices Architecture

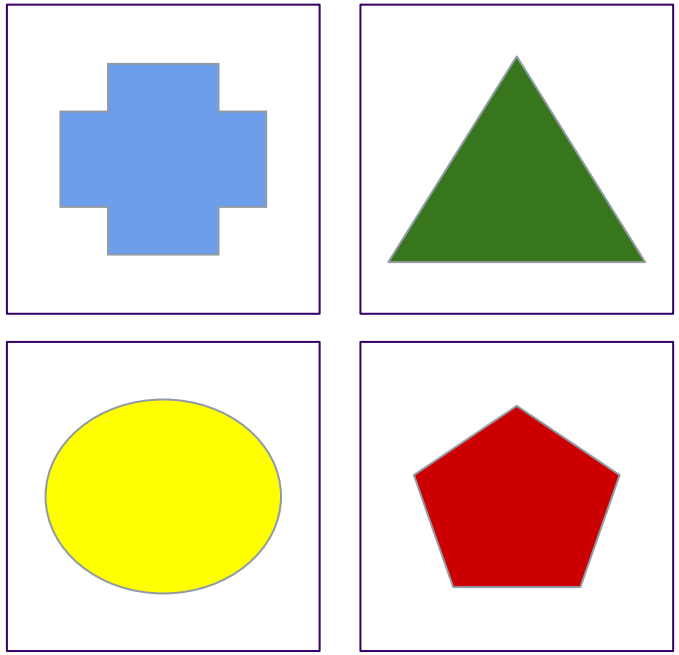
IFS

Monolith vs Microservices

A monolith application puts all its functionality into a single process

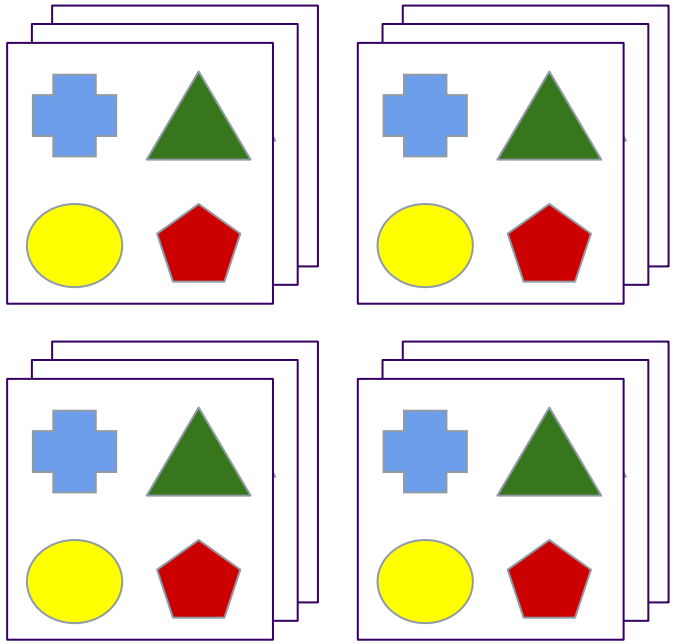


A microservices architecture puts each element of functionality into a separate process

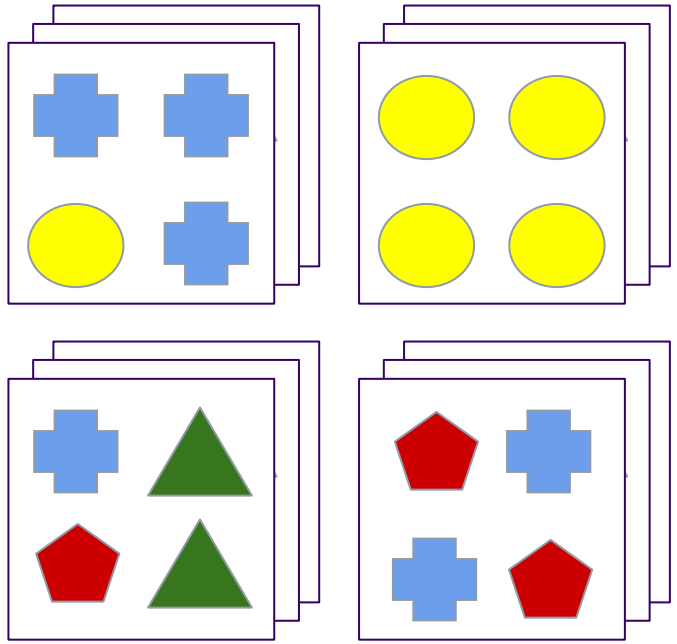


Monolith vs Microservices

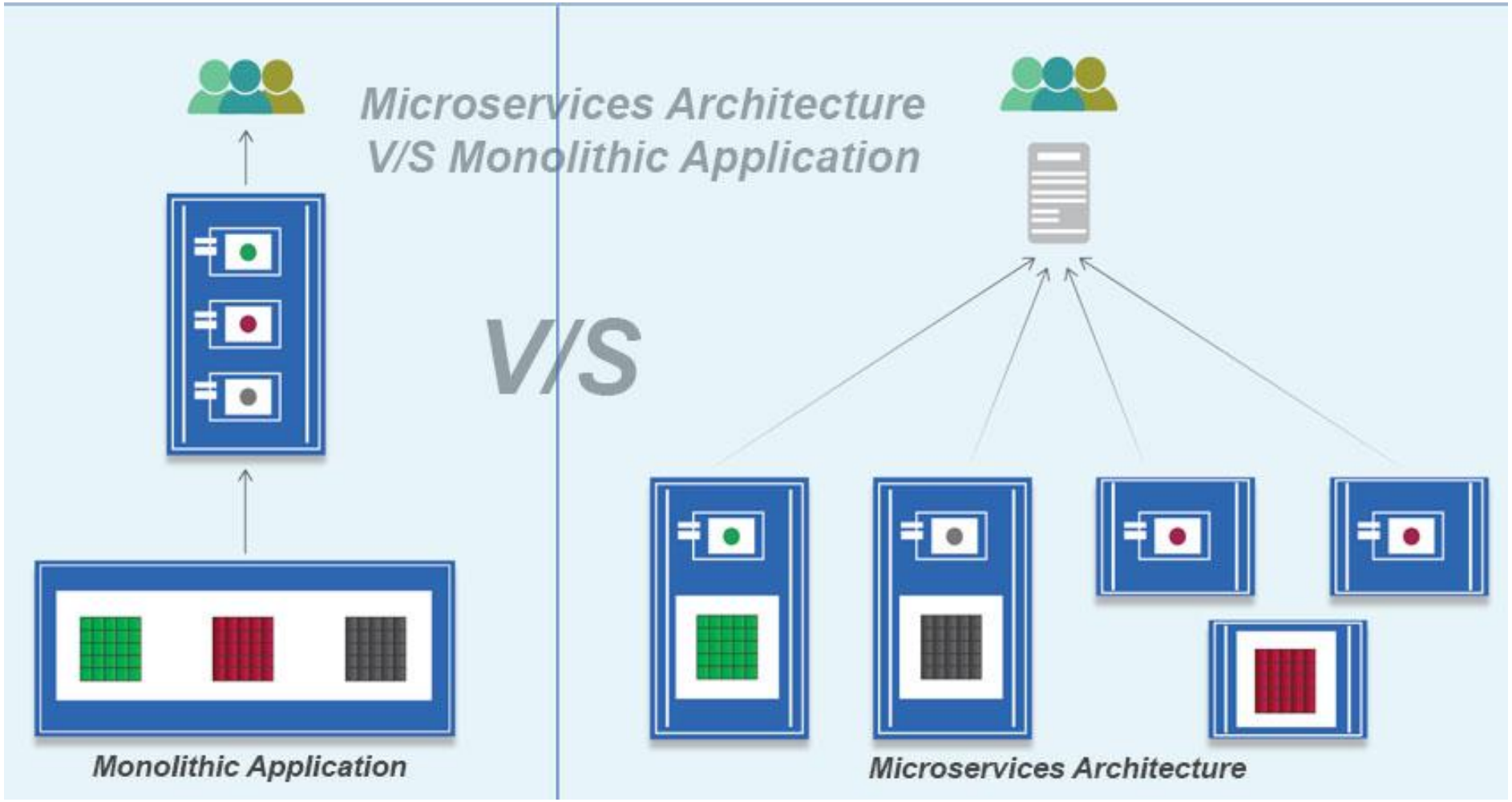
Monolith scales by replicating the whole thing on multiple servers



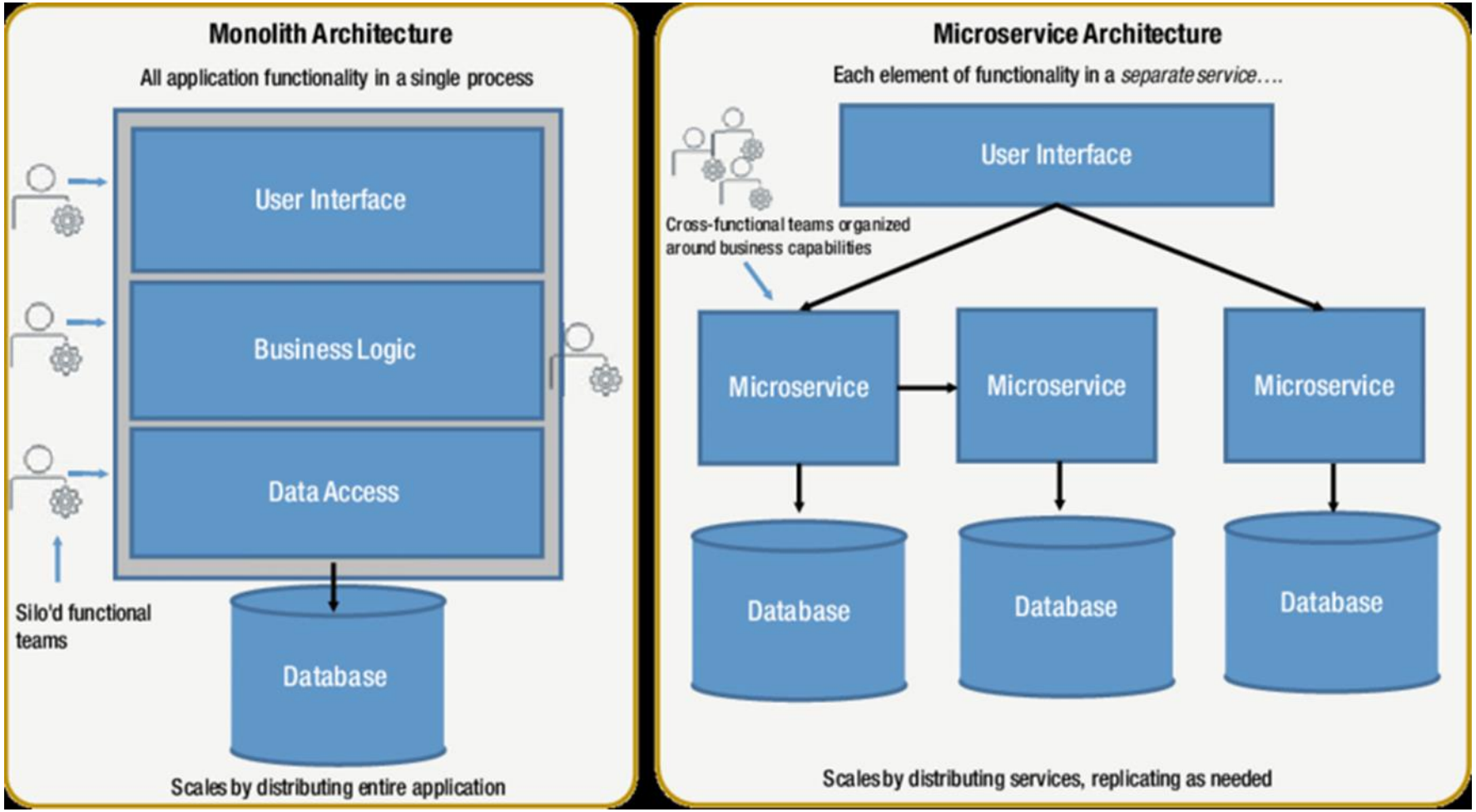
Microservices scales by distributing these services across servers, replicating as needed.



Monolith vs Microservices



Monolith vs Microservices



Characteristics of Microservices Architecture

IFS

Characteristics

Microservices Architecture

- Componentization via services
- Organized around business capabilities
- Products not Projects
- Smart endpoints and dumb pipes
- Decentralized Governance
- Decentralized Data Management
- Infrastructure Automation
- Design for failure

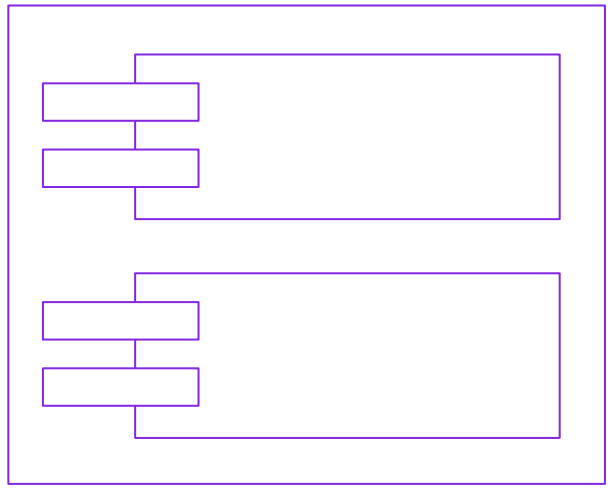
Characteristics

Componentization via Services

- Component is a unit of software that is independently replaceable and upgradeable.

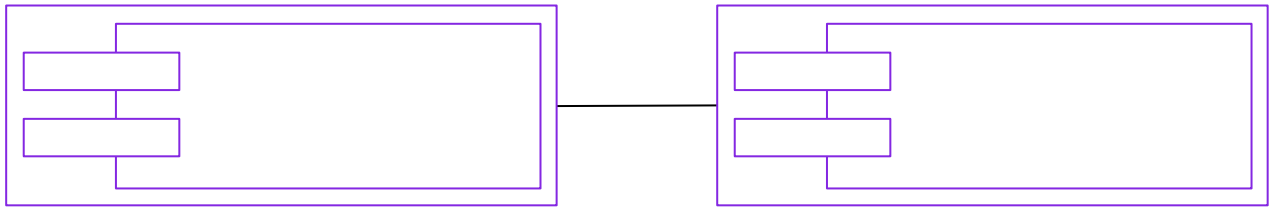
- Component can be a Library or a Service.

Component



Java 7 / Windows Environment

Services



Java 7 / Windows Environment

Java 8 / Sun Solaris

Characteristics

Componentization via Services

- If the component is a library, it is difficult to upgrade it since it affects to the whole application.

  Ex: Upgrading one component to a newer version of Java needs the whole application to be upgraded

- Communication between library components introduce high coupling.

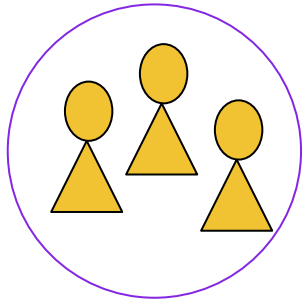
- If the components are services, they could be running in different runtimes, and you may use web services or messaging queues to communicate with them.

- Those services can be independently replaceable and upgradeable.

Organized Around

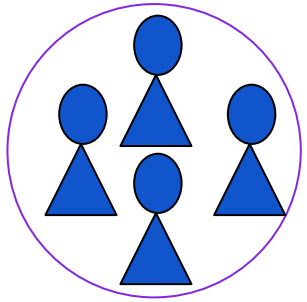
Business Capabilities

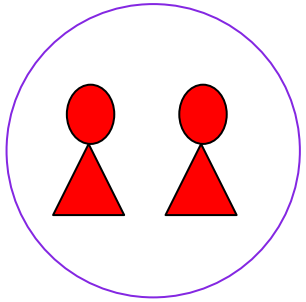
- Most of the software development organizations organize the team according to the technology.



Front End Team / UI

Middle tier Team / Server



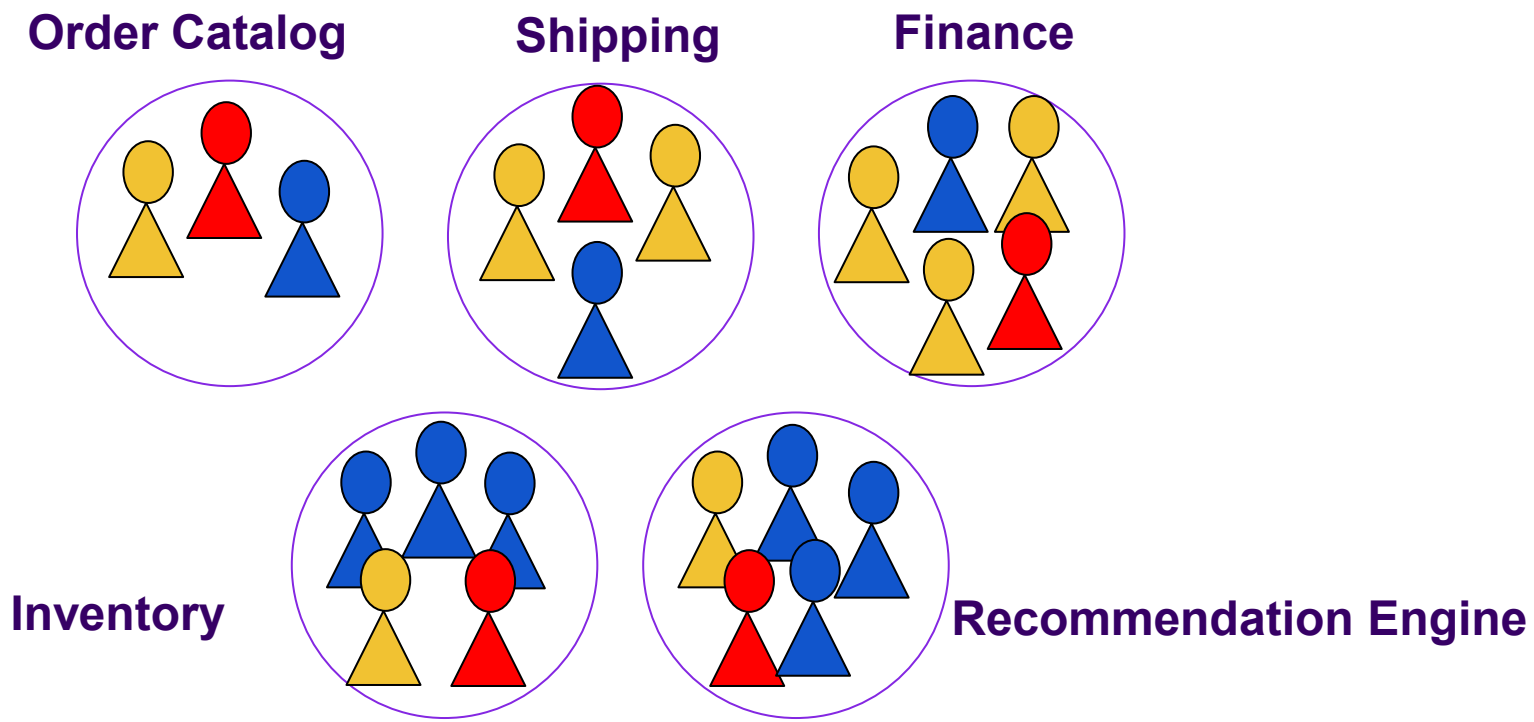


Database Team / DBA

Organized Around

Business Capabilities

- But when implementing a solution according to the Microservices approach, you should organize the team according to the business capabilities.



Organized Around

Business Capabilities

- These teams are cross functional, including full range of skills in all aspects of development, user experience, project management, ...etc.

- How big a particular team should be?

  It depends on the business capability and usually it should be no more than a 2 pizza team (About 24 people)

- Each team is responsible for all aspect of software development including the communication right up to the end user experience.

Characteristics

Products not Projects

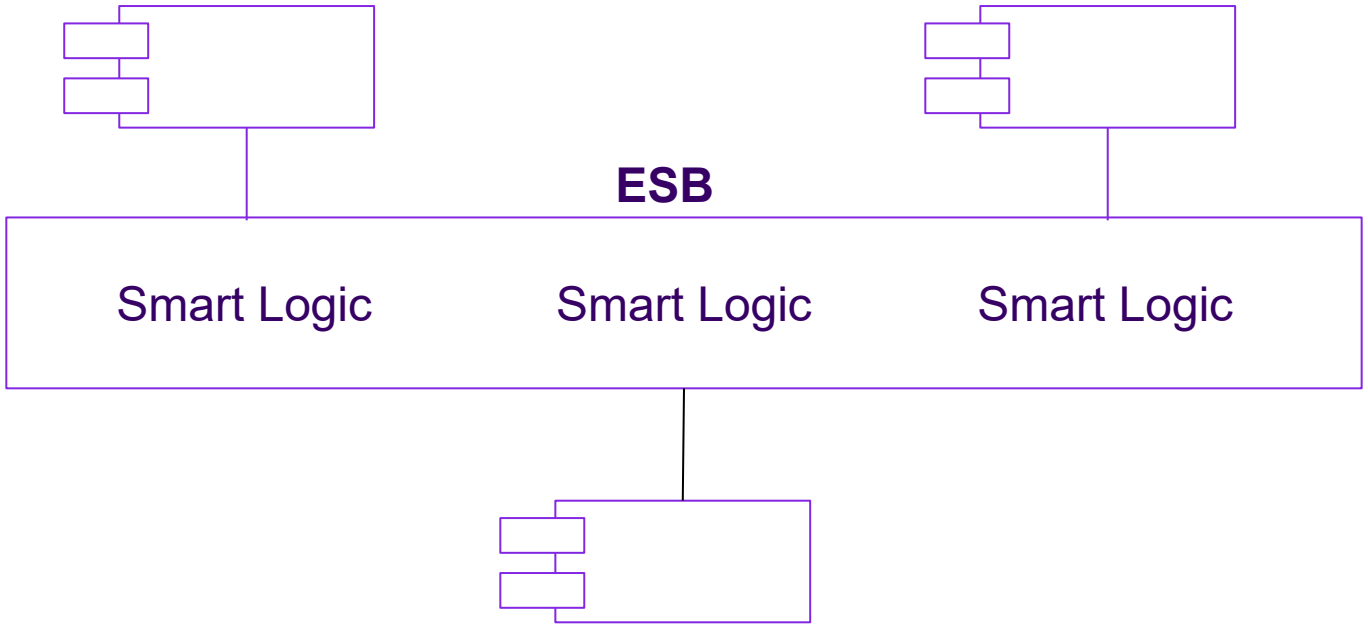
- In traditional project model the ultimate goal is to deliver a piece of software which is then considered as completed.
- After that it is handed over to the maintenance and support organization.
- But in Microservices approach the team who developed the component owns it over its full lifetime.
- Hence the components to be built in Microservices approach is considered as products.
- Product team is responsible for monitor it in production, its availability, new enhancements, bug fixing, end user experience and some of the support work.

Characteristics

Smart Endpoints and Dumb Pipes

- ESB is a powerful piece of middleware used in SOA which can route messages, apply business rules, message transformations, ...etc.

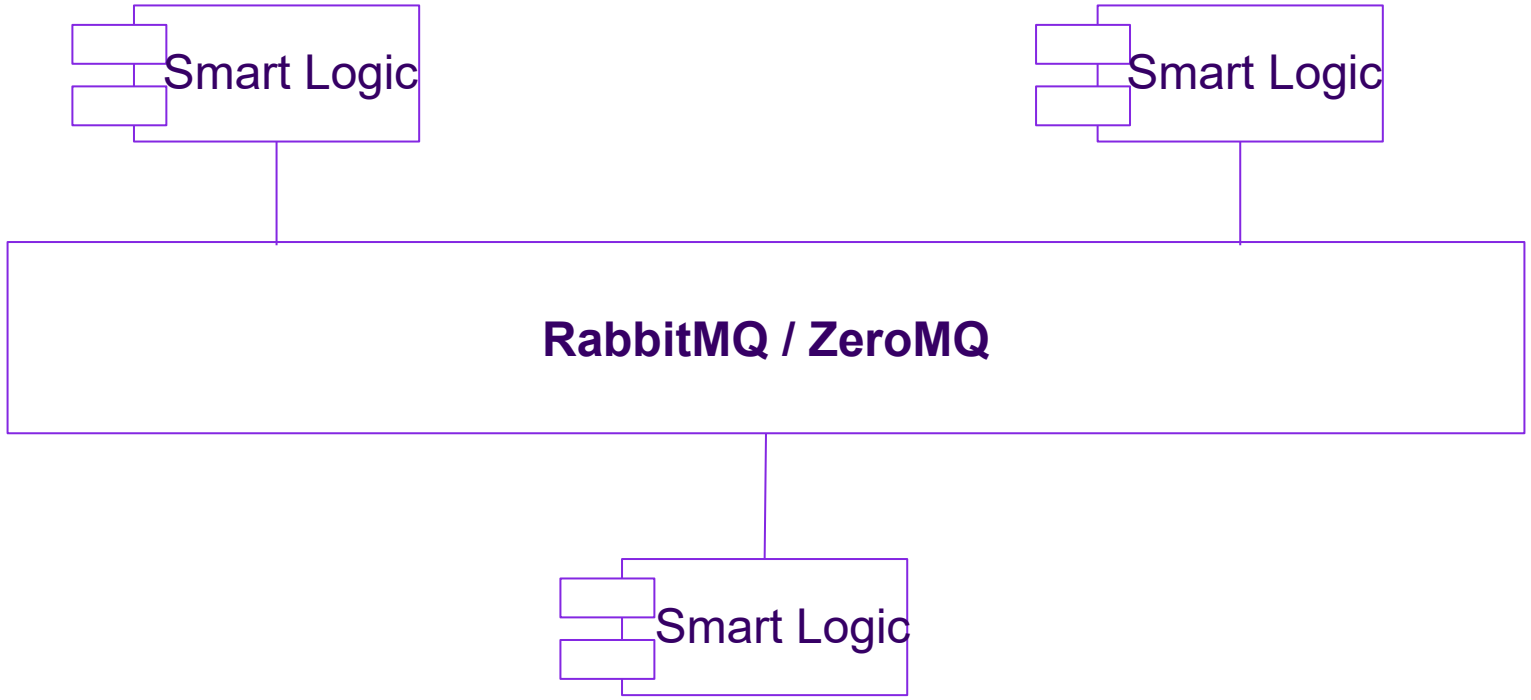
- So, most of the intelligent logic (smarts) is written in the ESB itself.



Characteristics

Smart Endpoints and Dumb Pipes

- But Microservices approach rejects this notion and moves the smarts to the endpoints.



Characteristics

Smart Endpoints and Dumb Pipes

- The main reason for that is Microservices approach favor in building applications as decoupled and as cohesive as possible.

- So, they rely on intelligent endpoints and a dumb but efficient piping mechanism.

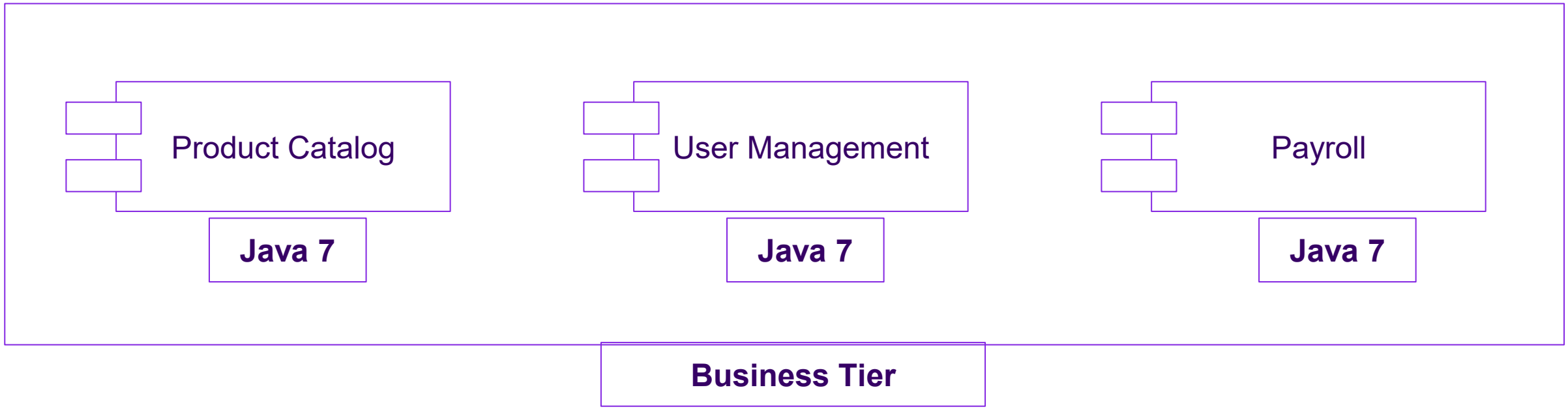
- The main inspiration for this is the internet itself which works very well because its a dumb set of pipes where the intelligence is implemented at the endpoints.

Characteristics

Decentralized Governance

- Most of the Monolithic approaches employ a centralized governance where there are strict set of rules regarding the technologies and patterns to be followed.

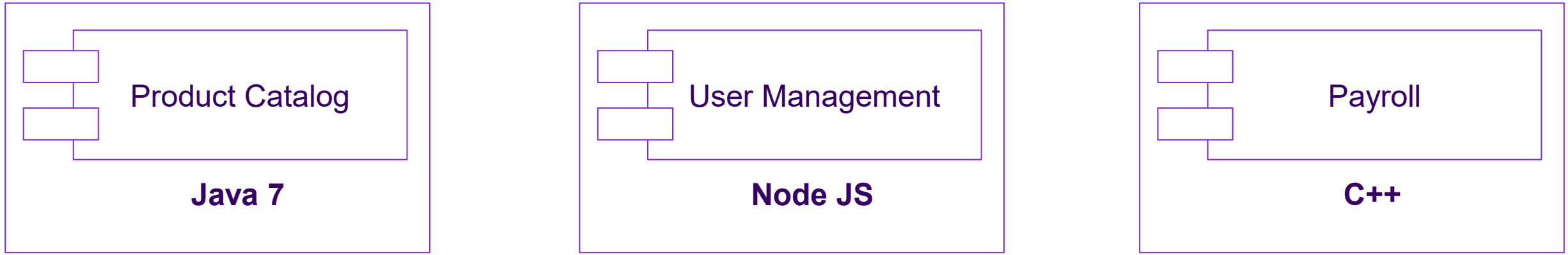
- At least all modules of the application should be written in same language.



Characteristics

Decentralized Governance

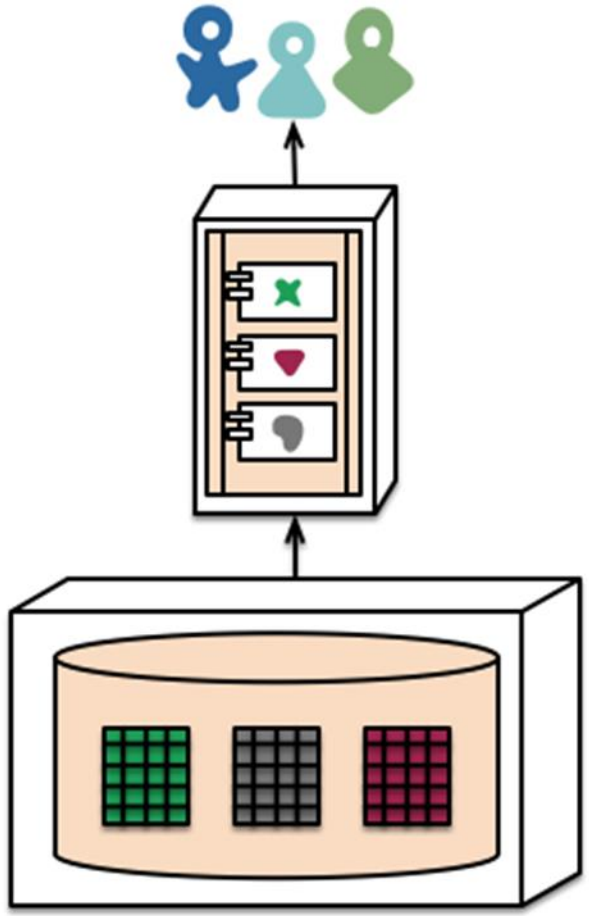
- Since Microservices approach has separate services where each running in its own process, the product team can decide on which language to write the components and what tools and technologies to use.



Characteristics

Decentralized Data Management

- In Monolithic approach, all data persist in one database.
- Any upgrade to the database will affect all business components of the application.

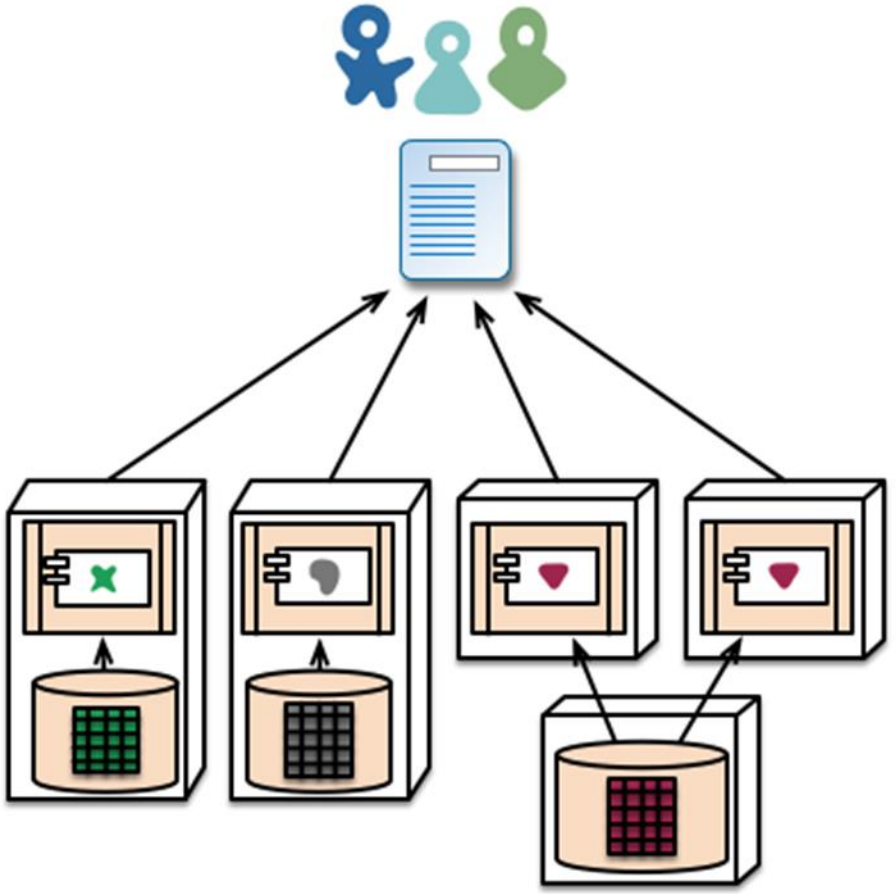


monolith - single database

Characteristics

Decentralized Data Management

- Using Microservices approach, every service is responsible for its own data and own persistence.
- However, a particular service cannot talk to another services data store directly.
- Communication should always occur through the other service's API.
- This provides the freedom for each product team to decide on which datastore to choose according to the business capability they are implementing.

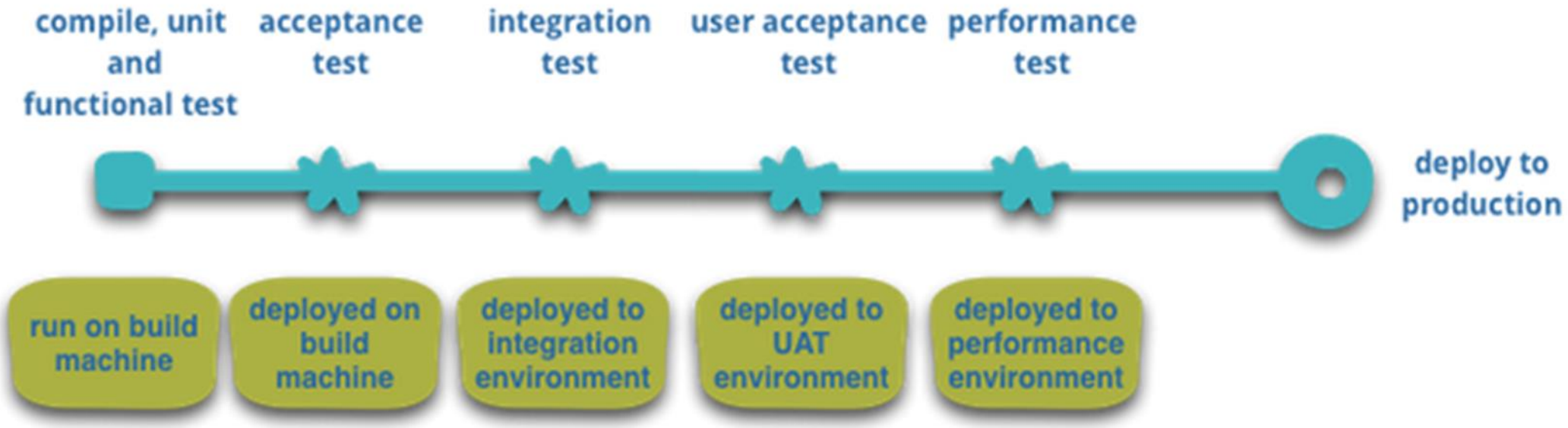


microservices - application databases

Characteristics

Infrastructure Automation

- In order to make everything working smoothly in Microservices, infrastructure automation has become an absolute necessity.



Characteristics

Infrastructure Automation

- Some of the features such as continuous delivery and blue green deployment has become mandatory if you are choosing the Microservices approach.

- Since it involves several services running with zero downtime and ensuring the consistency throughout the application, monitoring has also become an absolute necessity too.

Characteristics

Design for Failure

- Any service may fail in a given time in Microservices approach.
- Since all services communicate with each other via their APIs, failure in one service may affect the business flow of another service as well.
- Therefore, it is utmost important to have certain tools to monitor the health of the services so that when a service is down, the product team can work on it as soon as possible.
- In addition to that Microservices approach use certain tools to randomly making the services down in order to test how resilient their application is and how quickly they can recover the damage.

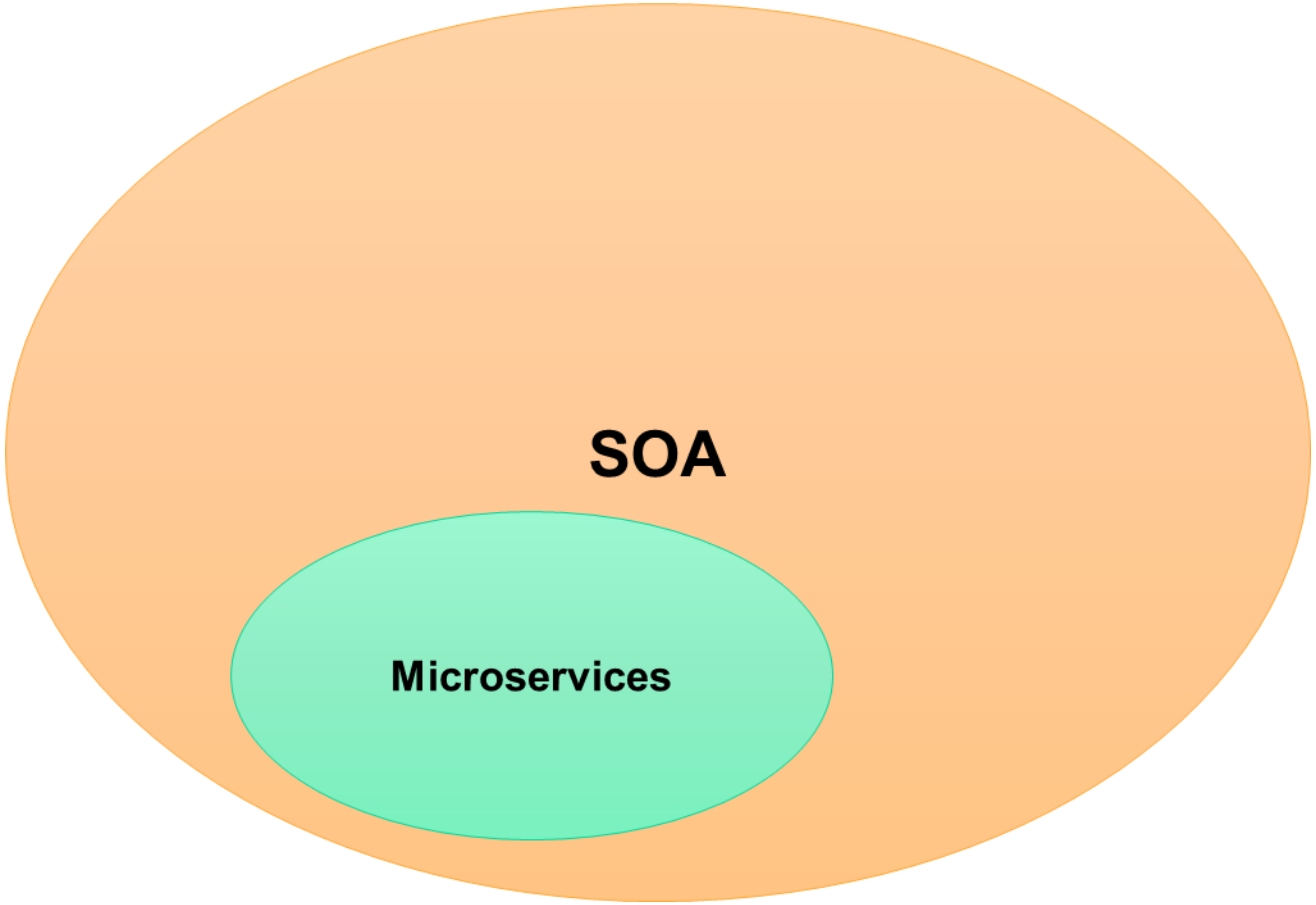
  Ex: Chaos Monkey by Netflix

Microservices vs SOA

IFS

Microservices vs SOA

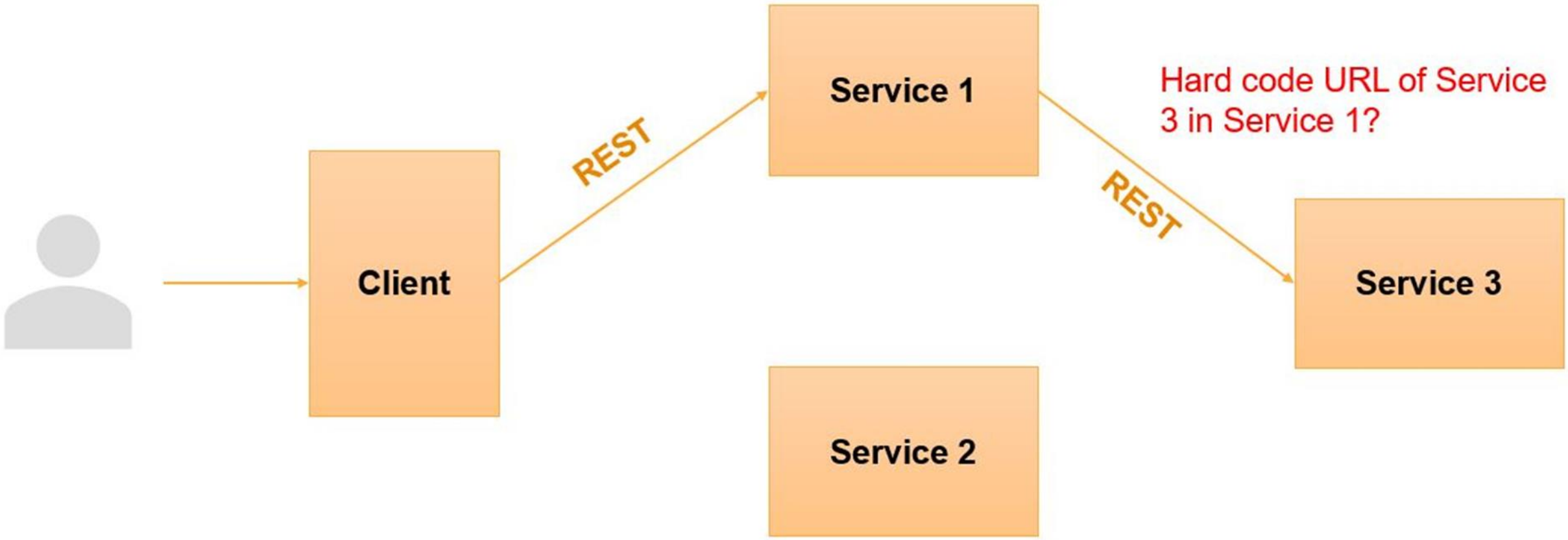
- Are microservices just SOA?



Communication Between Microservices

IFS

Communication Between Microservices

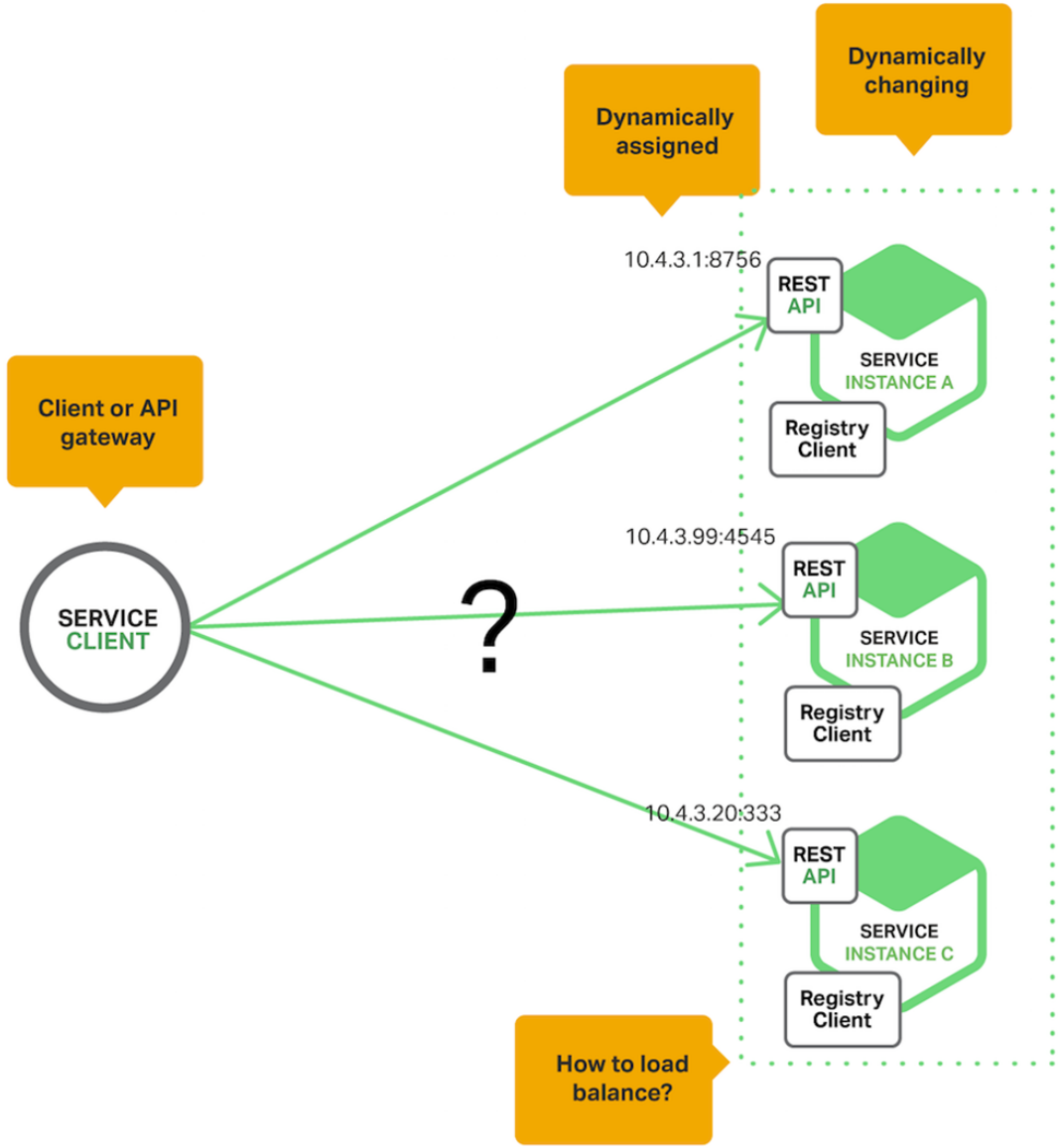


Communication Between Microservices

Why hard coding of URLs in Microservices are bad?

- Changes in URL of a microservice will require code updates in others.
- Dynamic URLs in the cloud.
- Load balancing
- Multiple environments (DEV, TEST, PROD)

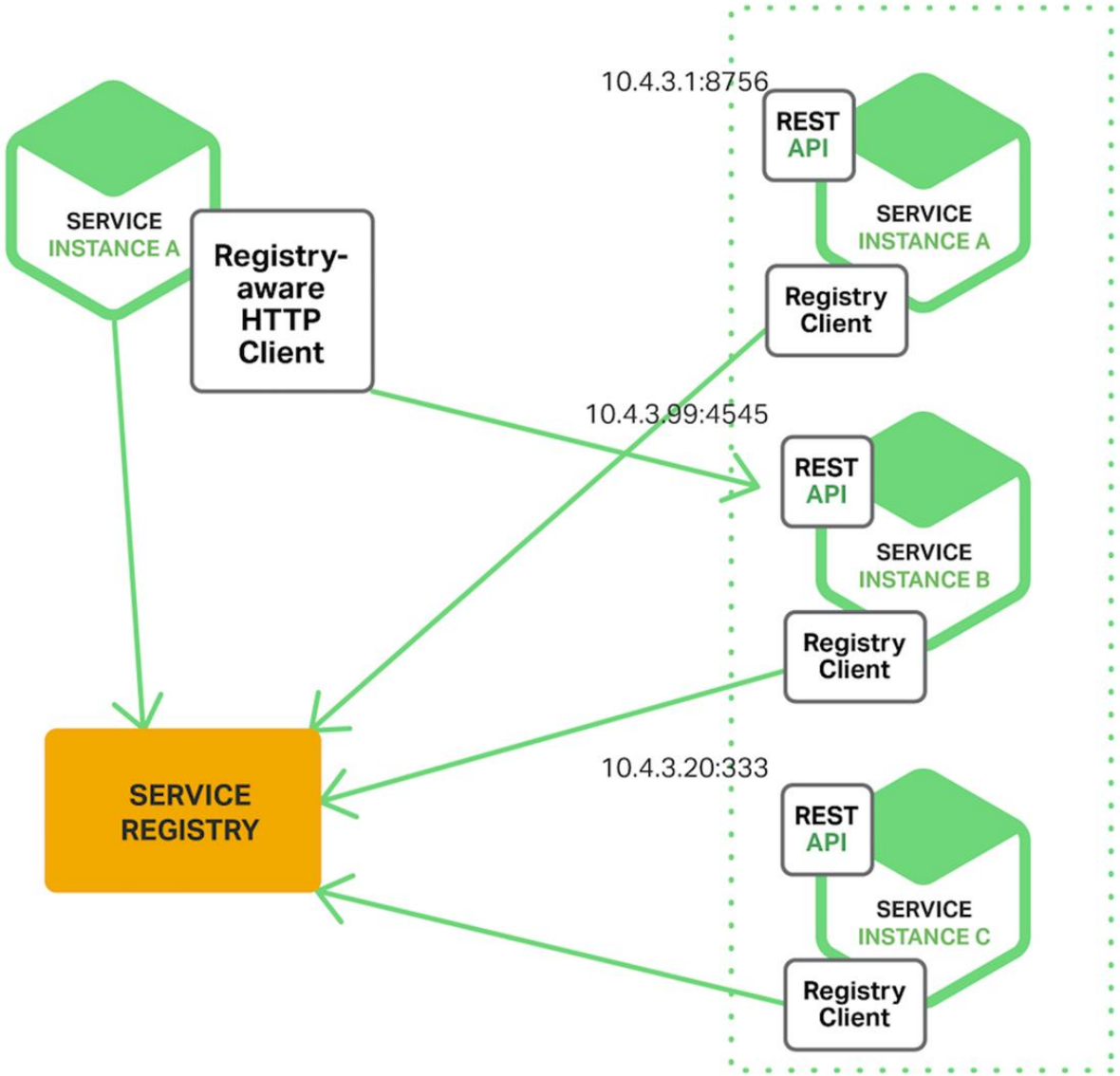
Communication Between Microservices



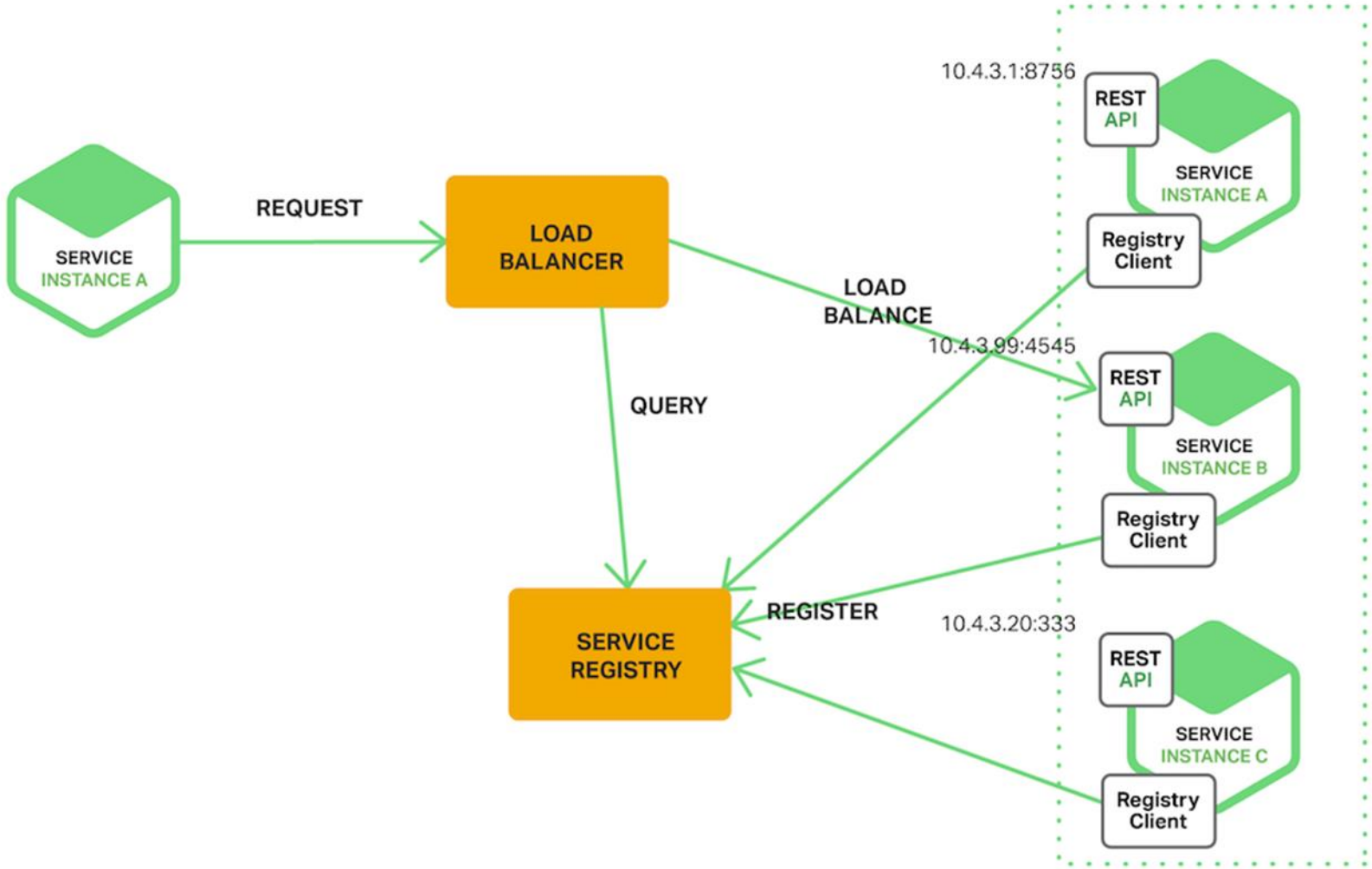
Service Discovery in Microservices

- Automatically registers microservices in a service registry.
- This is itself a microservice and we call it the discovery server.
- All microservices can refer the service registry when they need to talk to another microservice.

Client-Side Service Discovery



Server-Side Service Discovery



Advantages & Disadvantages of Microservices

IFS

Advantages of Microservices

Partial deployment

- You can upgrade services without affecting others.

High availability

- Even though your recommendation engine is shut down, you can still use the shopping cart functionality.

High scalability

- Demanding services can be deployed in multiple servers

Preserve modularity

- Business capabilities are separated, and coupling is low because of following the single responsibility principle

Freedom to choose technology

- Best possible technology can be used to write a specific business capability.

Advantages of Microservices

Faster deployments

Easier to understand the codebase

- Microservice focuses on one business capability.

Improves fault isolation

- Application may remain largely unaffected by the failure of a single module.

Disadvantages of Microservices

- Difficult to implement multiple database transaction logic
- Testing a Microservices based application can be cumbersome.
- Poor consistency
- Complex to understand the whole architecture of the application.
- Redundant code (classes) may span across multiple Microservices.

Thank you!

#MOMENTOFSERVICE