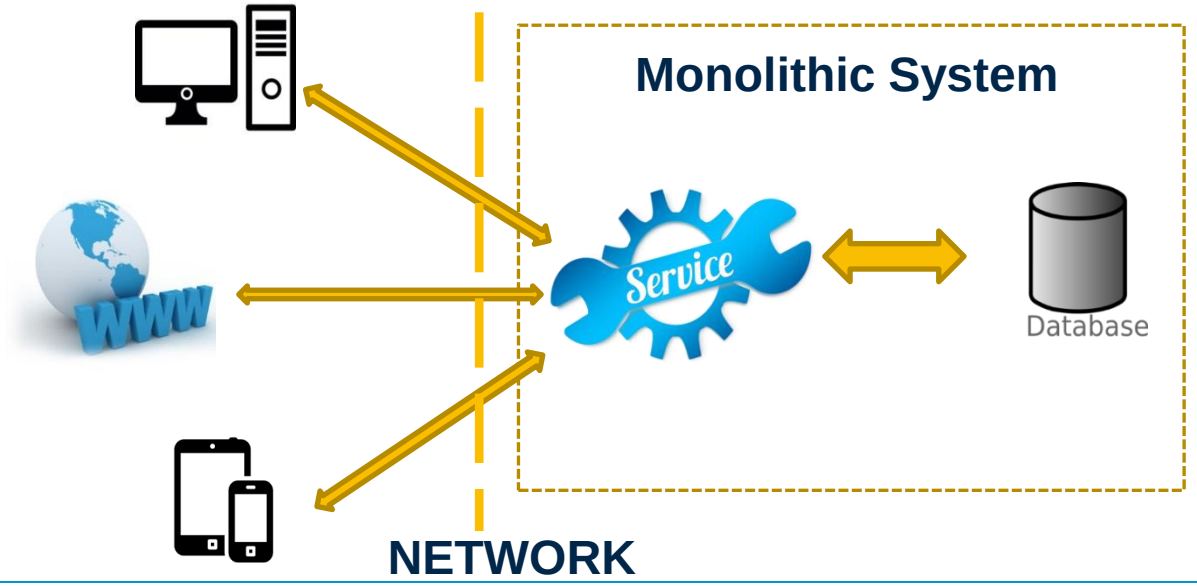
Introduction to Microservices

Lesson Objectives

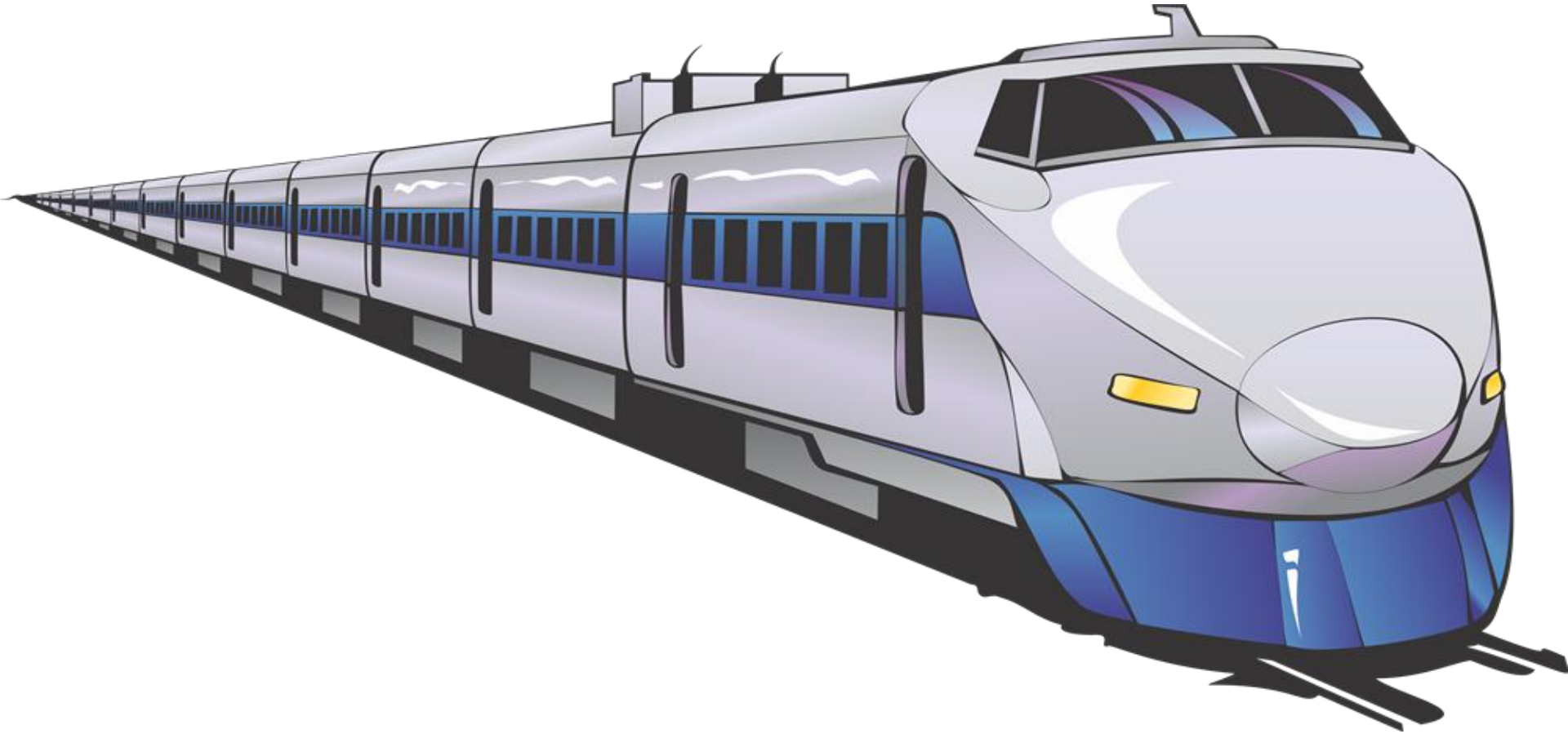
- At the end of this module you will be able to:
  - Understand limitations of Monolithic Systems
  - Understand the need and importance of Microservices
  - Design principles need to be followed in Microservices

Service Introduction

- A Service is an application component which provides functionality to other components (web app, mobile app, desktop app, or even another service)
- Service-Oriented Architecture is an architectural pattern in which components provide services to other components via a communications protocol over a network.



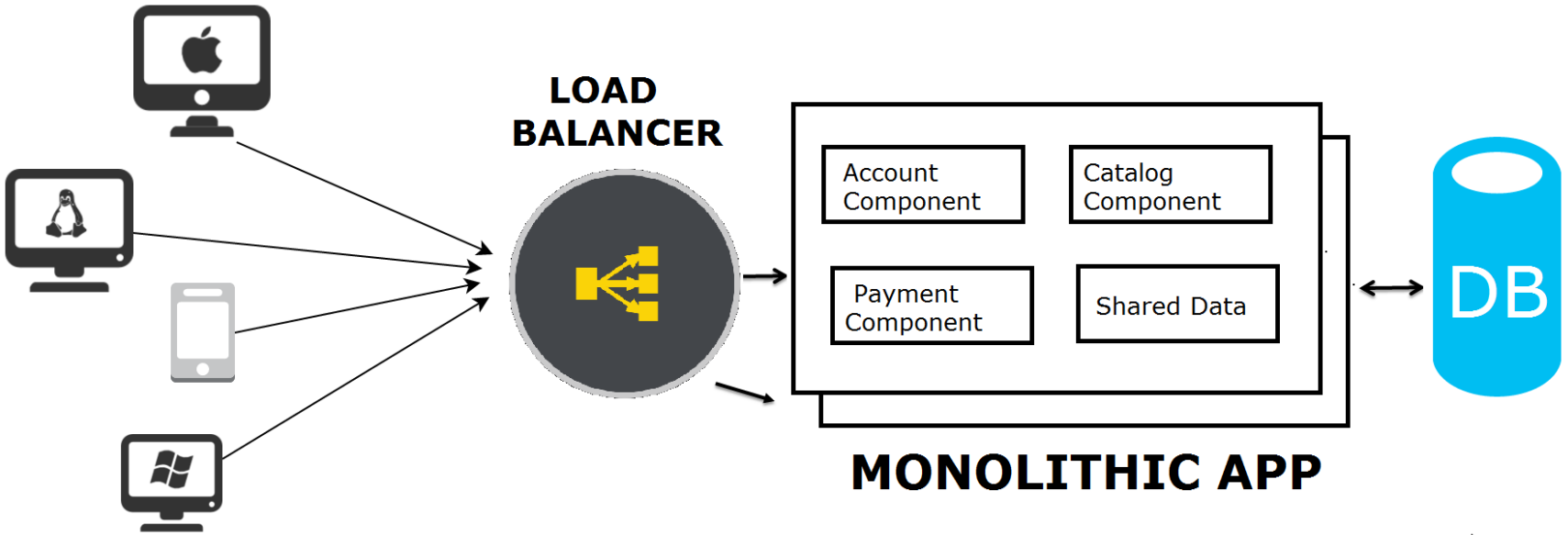
Monolithic System



Monolithic System

- Monolithic system is a typical enterprise application.
- There's always one package which basically contains everything which tends to end up with large code base.
- When application expands, It keeps grows with no physical division and no restriction in size.
- Developers takes longer time to develop new functionality with in application and it's difficult to make a change without affecting other parts of the system.
- Deployment of a large system can also be challenging, because even for a small bug fix, a new version of the entire system need to be redeployed.
- New technologies cannot be adopted for a specific functionality, since it is available in a overall package.

Monolithic Architecture

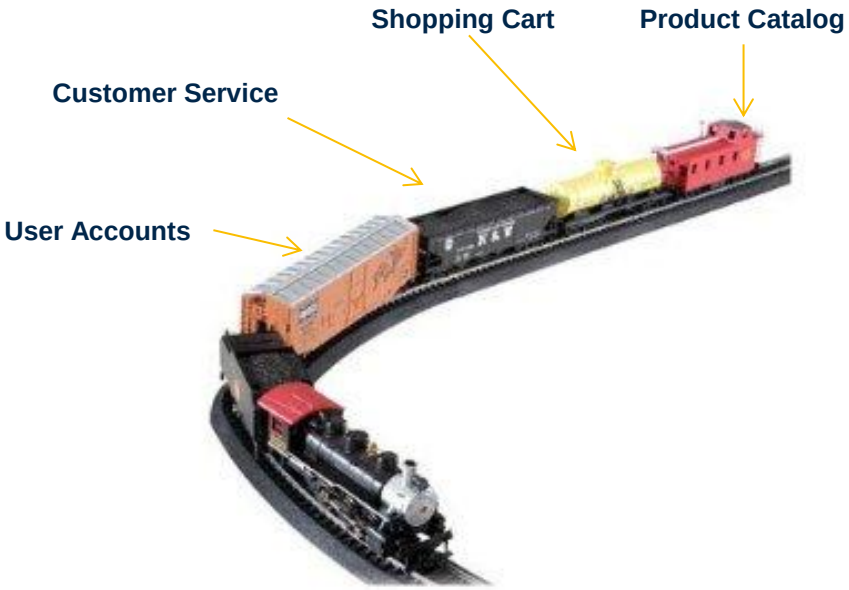


Monolithic System Advantages

- Single codebase
  - Easy to develop/debug/deploy
  - Good IDE support
- Easy to scale horizontally (but can only scale in an “un-differentiated” manner)
- A Central Ops team can efficiently handle

Monolithic System Limitations

- As codebase increases …
  - Tends to increase “tight coupling” between components (system stops working, if one component stop working)
  - Just like the cars of a train
- All components have to be coded in the same language
- Scaling is “undifferentiated”
  - Cant scale “Product Catalog” differently from “Customer Service”



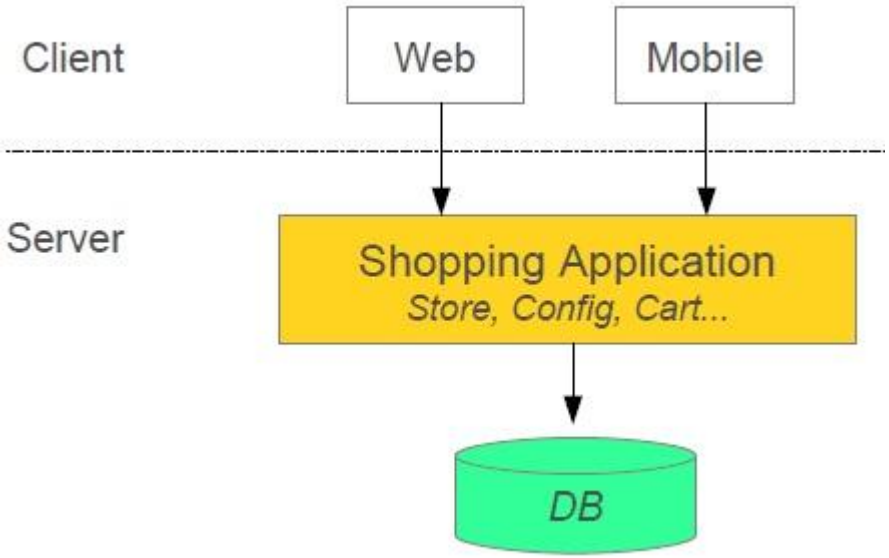
Microservices Introduction

- Microservices are small software components that are specialized in one task and work together to achieve a higher-level task.
- A microservice is an independent unit of work that can execute one task without interfering with other parts of the system.
- Micro-service normally has a single focus. i.e. It does one thing and it does it well.
- It is basically an improved version of service-oriented architecture.
- Microservices are business units that model the company processes.
- They are smart endpoints that contain the business logic and communicate using simple channels and protocols

Microservices Advantages

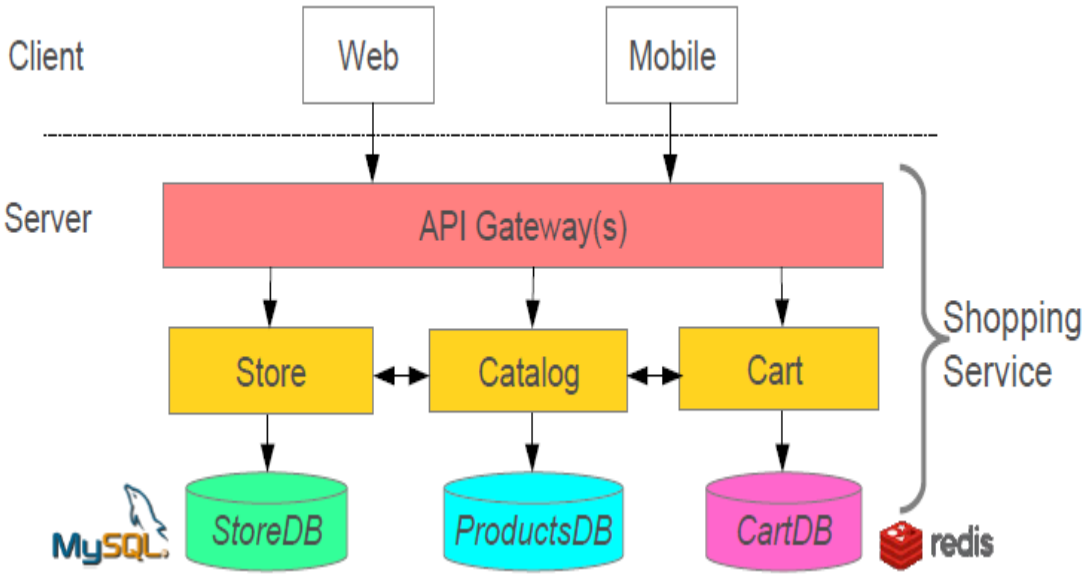
- Individual parts can be enhanced based on market needs.
- Microservices can be tested using automated testing tools
- Deployment is super easy with the release and deployment tools.
- Virtual machines can host microservices on-demand
- Embraces new technology
- Supports Asynchronous Communication technology
- Simpler server side and client side technology
- Shorter development times
- Highly scalable and better performance
- Enables frequent updates and fast issue resolution

Shopping system without Microservices (Monolith architecture)



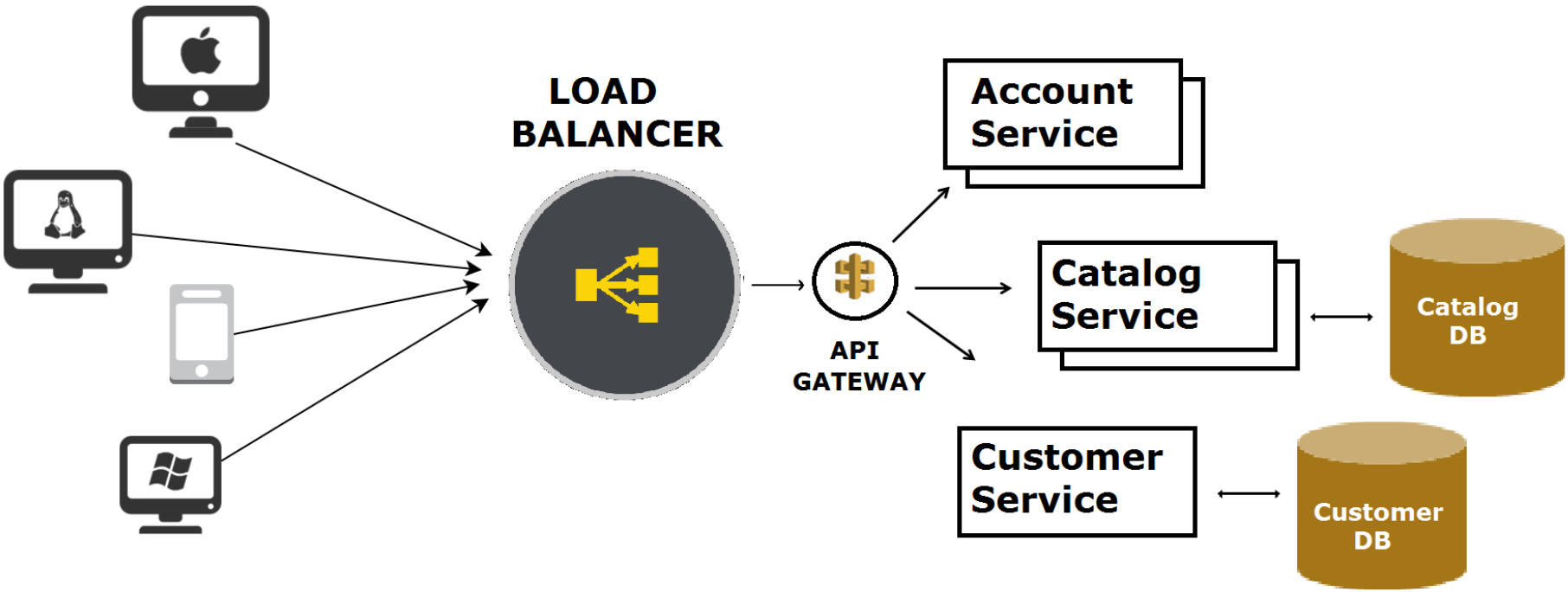
In this architecture we are using Monolith architecture i.e. all collaborating components combine all in one application.

Shopping system with Microservices



In this architecture style the main application divided in a set of sub applications called microservices.
One large Application divided into multiple collaborating processes as below.

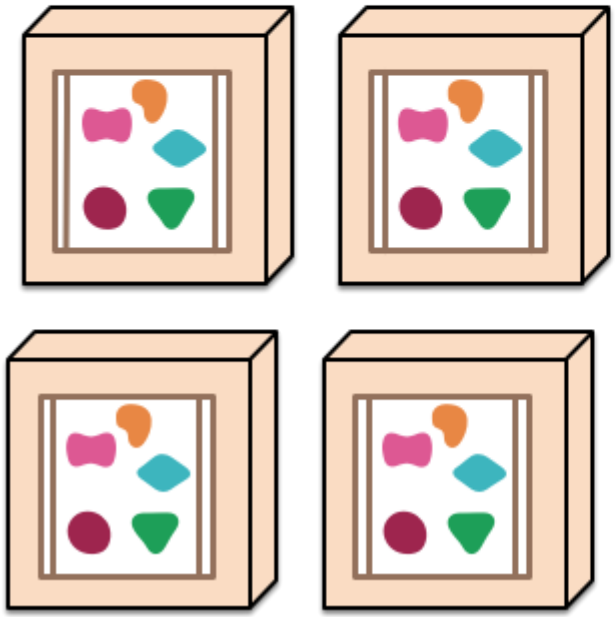
Microservice Architecture



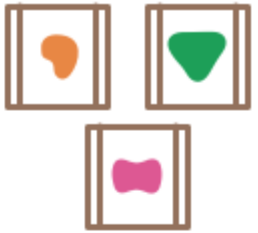
Monolithic vs Microservices

A monolithic application puts all its functionality into a single process...

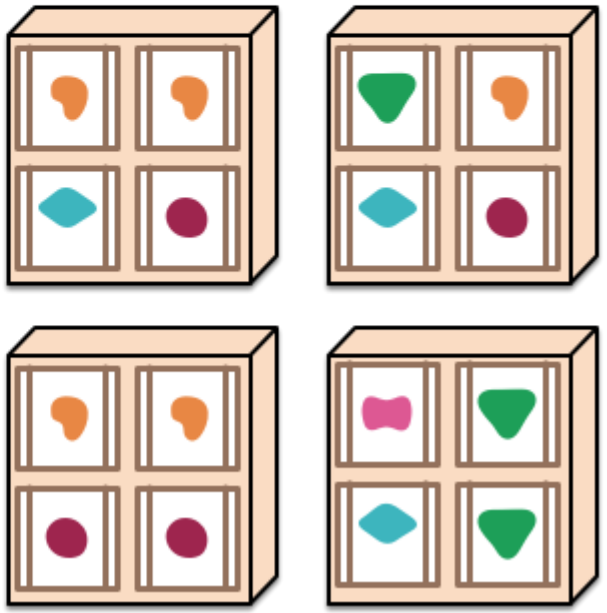
... and scales by replicating the monolith on multiple servers



A microservices architecture puts each element of functionality into a separate service...



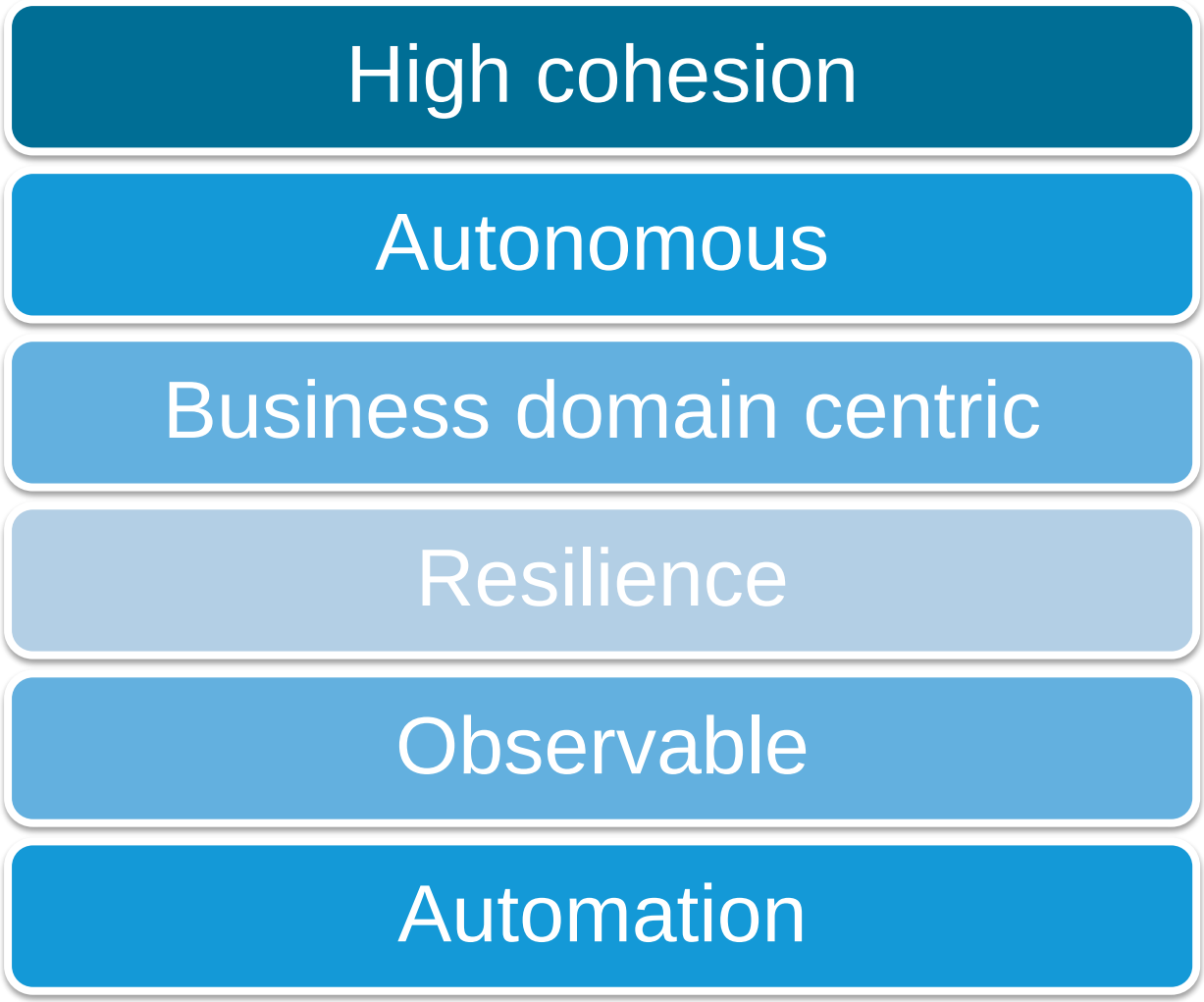
... and scales by distributing these services across servers, replicating as needed.



Microservices Challenges

- Can lead to chaos if not designed right
- Distributed Systems are inherently Complex
- Operational Overhead(DevOps Model required)
- MicroServices does not automatically mean better Availability, unless Fault Tolerant Architecture is followed.

Microservices Key design principles



High Cohesion

- High cohesion principle controls the size of the microservice and the scope of the contents of the microservice
  - Microservices content and functionality in terms of input and output must be comprehensible.
  - Must have a single focus.
  - It makes overall system highly scalable, flexible, and reliable

Autonomous

- Microservices should also be autonomous
  - A microservice should not be subject to change because of an external system it interacts with or an external system that interacts with it. i.e. It should be loosely coupled.
- Microservice should also be stateless
- Microservice should support backwards compatibility
- Microservice can be independently changeable and independently deployable
- Contracts between the services must be clearly defined
  - Concurrently developed by several teams

Business Domain Centric

- A microservice should be a business domain centric
  - The overall idea is to have a microservice represent a business function or a business domain
  - This helps to scope the service and control the size of the service
  - When business changes or the organization changes or the functions within the business change, microservices should change in the same way because the system is broken up into individual parts which are business domain centric

Resilience

- Microservice need to embrace failure when it happens.
  - Microservice needs to embrace the failure by degrading the functionality within it or by using default functionality.
  - If one component of a system fails, the failure should not cascade, the problem should be isolated and the rest of the system should work.
  - Microservices can be more resilient by having multiple instances which registers themselves on start up, and deregister themselves on failure to have an awareness of fully functioning microservices.
  - Microservices should validate incoming data, and it shouldn't fall over because they've received something in an incorrect format.

Observable

- System built with Microservice architecture must be observerable.
  - System's health must be observed in terms of logs, system status and errors.
  - Logging must be centralized and monitored from a centralized place. Which help us to achieve
    - Solving the problem quickly
    - Monitor business data
    - Data used for scaling and capacity planning.
    - Distributed transaction details

Automation

- Microservice architecture should support the feature of automation with the help of automation tools.
  - Automated testing will reduce the amount of time required for manual regression testing, and the time taken to test integration between services and clients, and also the time taken to set up test environments.
  - Continuous Integration tools

Microservices Best Practices

- Isolate your services (Loosely Coupled)
- Use Client Side Smart LoadBalancers
- Dependency Calls
  - Guard your dependency calls
  - Cache your dependency call results
  - Consider Batching your dependency calls
  - Increase throughput via Async/ReactiveX patterns
- Test Services for Resiliency
  - Latency/Error tests
  - Dependency Service Unavailability
  - Network Errors

Spring and Microservices

- **Spring enables separation-of-concerns**

Loose Coupling- Effect of changes isolated

Tight Cohesion- Code perform a single well defined task

Microservices provide the same strength as Spring provide

Loose Coupling- Application build from collaboration services or processes, so any process change without effecting another processes.

- ***Tight Cohesion-An individual service or process that deals with a single view of data.***

Microservices Benefits

- Smaller code base is easy to maintain.
- Easy to scale as individual component.
- Technology diversity i.e. we can mix libraries, databases, frameworks etc.
- Fault isolation i.e. a process failure should not bring whole system down.
- Better support for smaller and parallel team.
- Independent deployment
- Deployment time reduce

Microservices Challenges

- Difficult to achieve strong consistency across services
- ACID transactions do not span multiple processes.
- Distributed System so hard to debug and trace the issues
- Greater need for end to end testing
- Required cultural changes in across teams like Dev and Ops working together even in same team.

Microservices Infrastructure

- Platform as a Service like Pivotal Cloud Foundry help to deployment, easily run, scale, monitor etc.
- It support for continuous deployment, rolling upgrades fo new versions of code, running multiple versions of same service at same time.

Microservices Tooling Supports

- Setup new service by using Spring Boot
- Expose resources via a RestController
- Consume remote services using RestTemplate

Summary

- Service-oriented architecture (SOA) is a design approach where multiple services collaborate to provide some set of capabilities.
- A monolithic application puts all its functionality in a single process where as microservices architecture puts each functionality into a separate service.
- Monolithic apps – good for small organizations
- MicroServices have its challenges, but the benefits are many