Outline

Planned topics for this lesson:

- What is Software Architecture ?

  THE BLUEPRINT !

- Why Microservices?

  COMPANIES LIKE NETFLIX, AMAZON, AND UBER HAVE SHIFTED TO MICROSERVICES TO MANAGE THEIR GLOBAL INFRASTRUCTURE MORE EFFICIENTLY

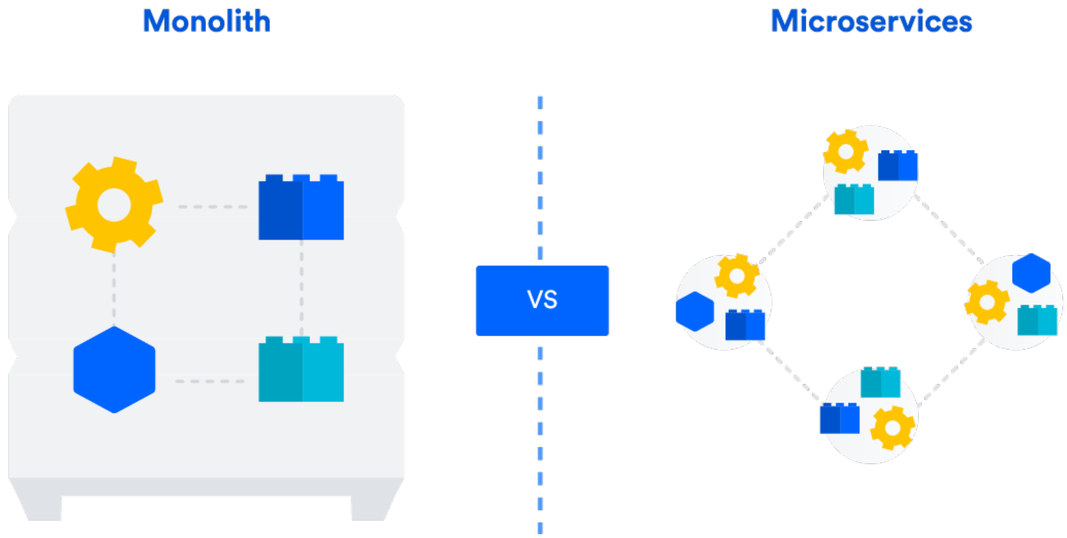
- Why NOT Monoliths?

  FOR SMALL APPLICATIONS, A MONOLITHIC APPROACH IS OFTEN SIMPLER AND EASIER TO MANAGE

Software Architecture

Introduction - What is it?

- **Software architecture** refers to the structure and organisation of a system's components and how they interact.
- It defines the **blueprint** for both functional and non-functional requirements.
- **Why it matters**: The right architecture ensures systems are **scalable**, **maintainable**, and **efficient**.



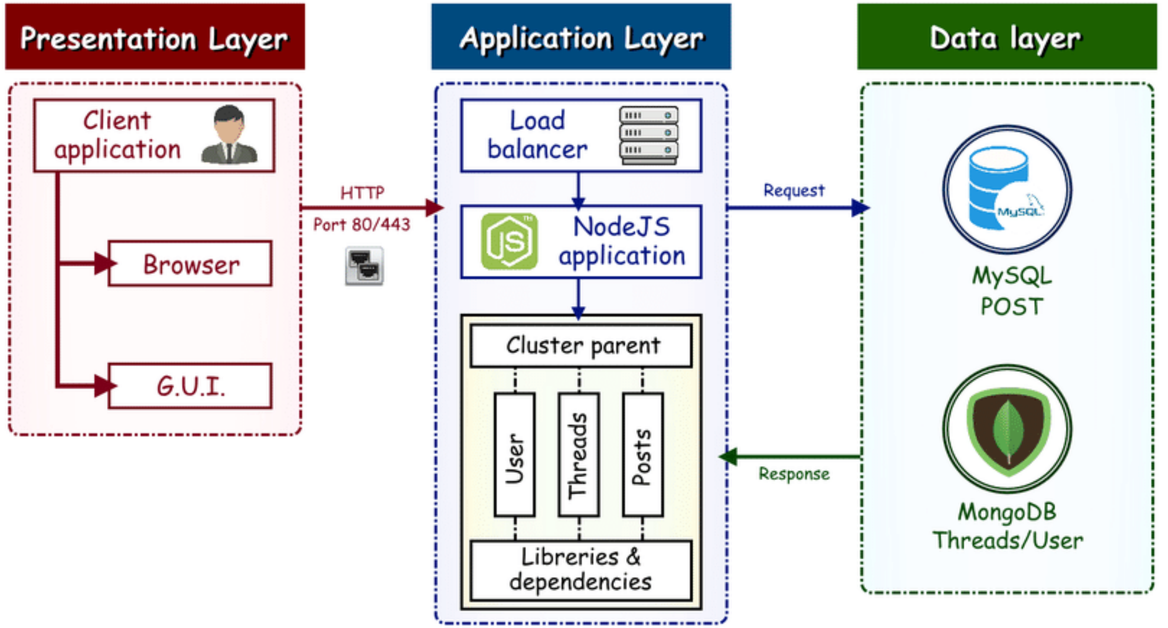
Software architecture is like the **blueprint** for a software system. Just as a building's architecture outlines the structure, rooms, and how they are connected, software architecture defines the structure of a software system, how its different parts interact, and how it will meet both current and future needs.

Monolithic Architecture

Overview

Monolithic architecture is a traditional way of building software where all components are tightly integrated into one single system.

- **Single codebase**: All functionality resides within a unified codebase.
- **Tight coupling**: Components are highly dependent on each other.
- **Single deployment unit**: The entire application is deployed as one.

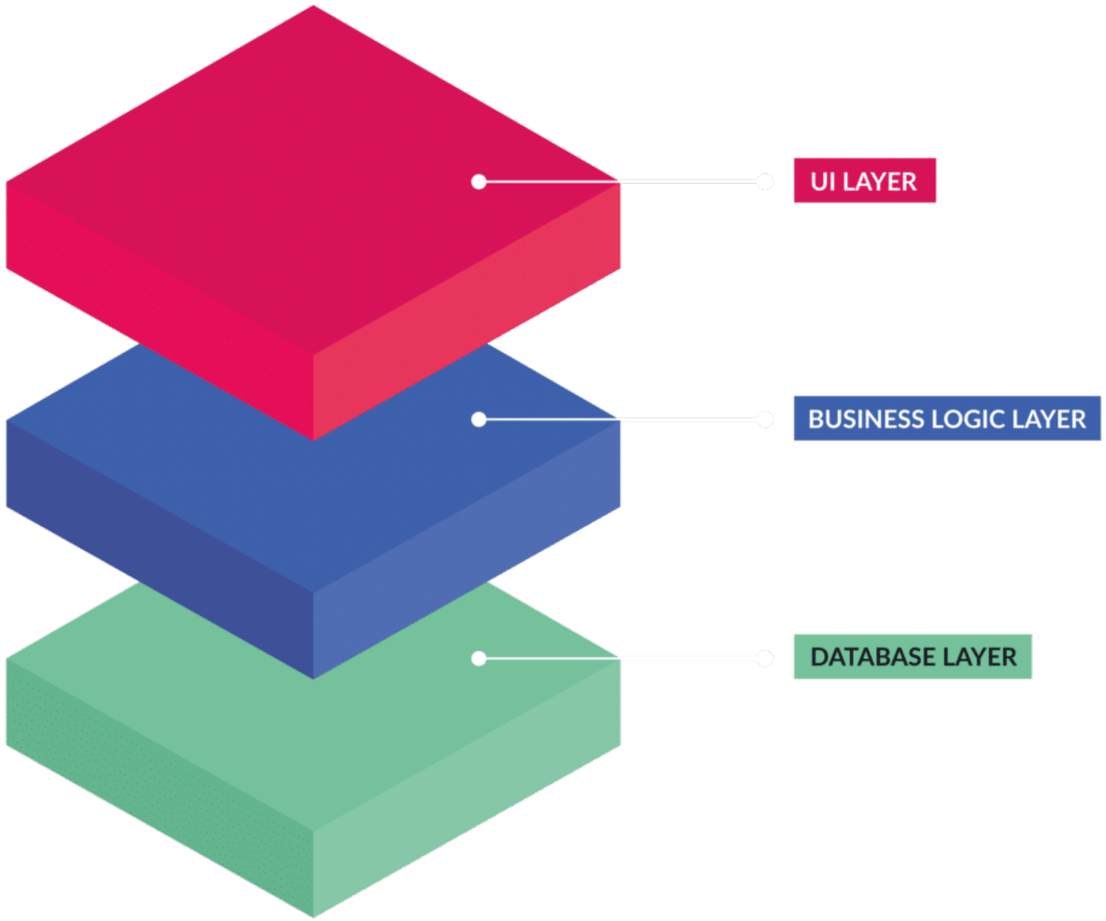


Example: A web application where the **UI**, **backend logic**, and **database interactions** are packaged together and deployed as one unit

Monolithic Architecture

Benefits

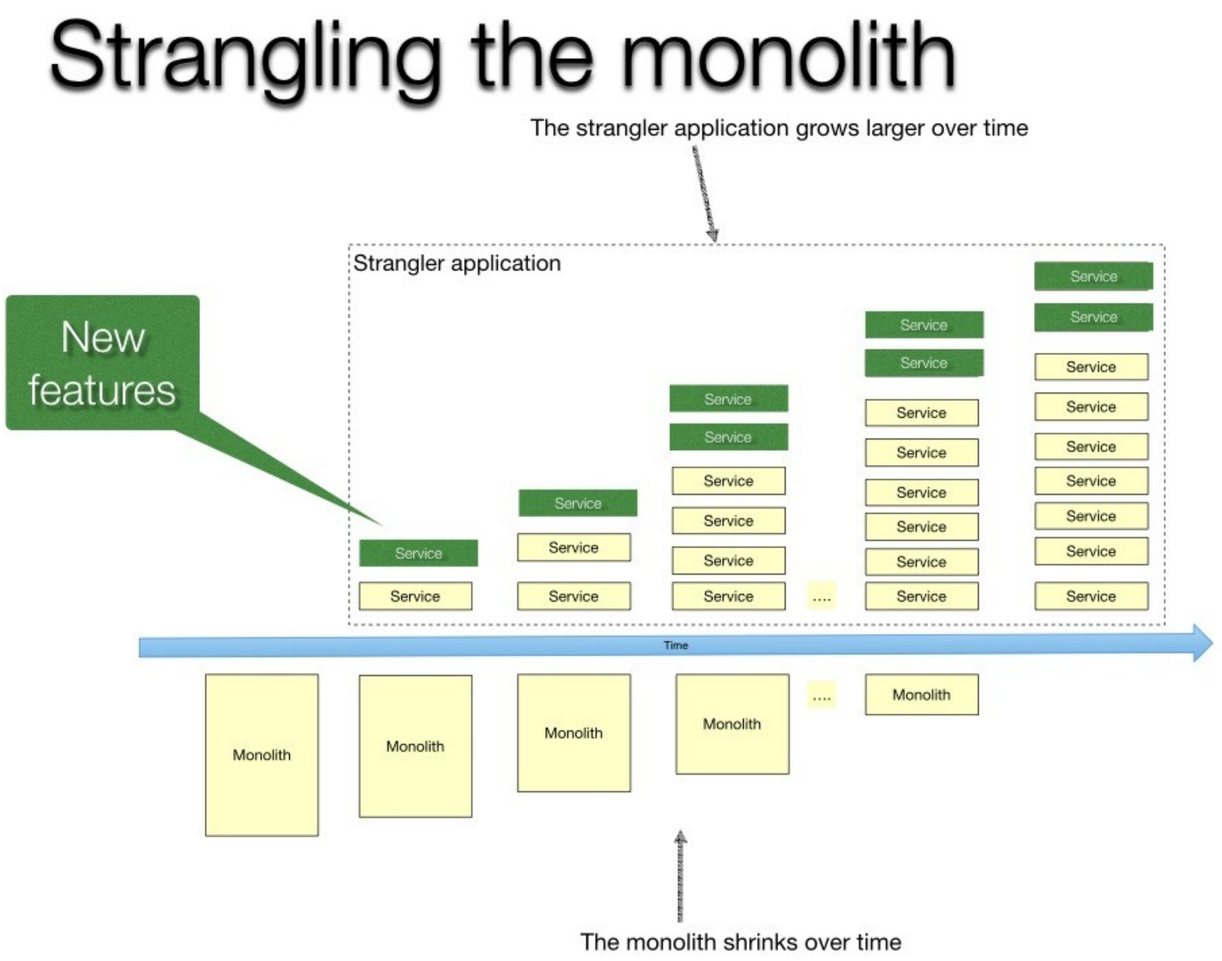
- **Simple development**: Since all components are in one place, managing and understanding the system is easier in early stages.
- **Easy to deploy**: Deployment is straightforward since the entire system is packaged into a single executable or container.
- **Easy debugging**: Debugging is centralised, meaning you can trace errors without worrying about interactions between different services.



Monolithic Architecture

Challenges

- **Scalability**: Hard to scale only specific parts of the system; you have to scale the entire system even if only one feature is resource-intensive.
- **Maintenance issues**: Over time, the codebase becomes more complex and harder to manage.
- **Limited flexibility**: Since components are tightly coupled, changing one part can impact the entire system.
- **Deployment bottlenecks**: As the system grows, deployment can become slower and riskier due to interdependencies.

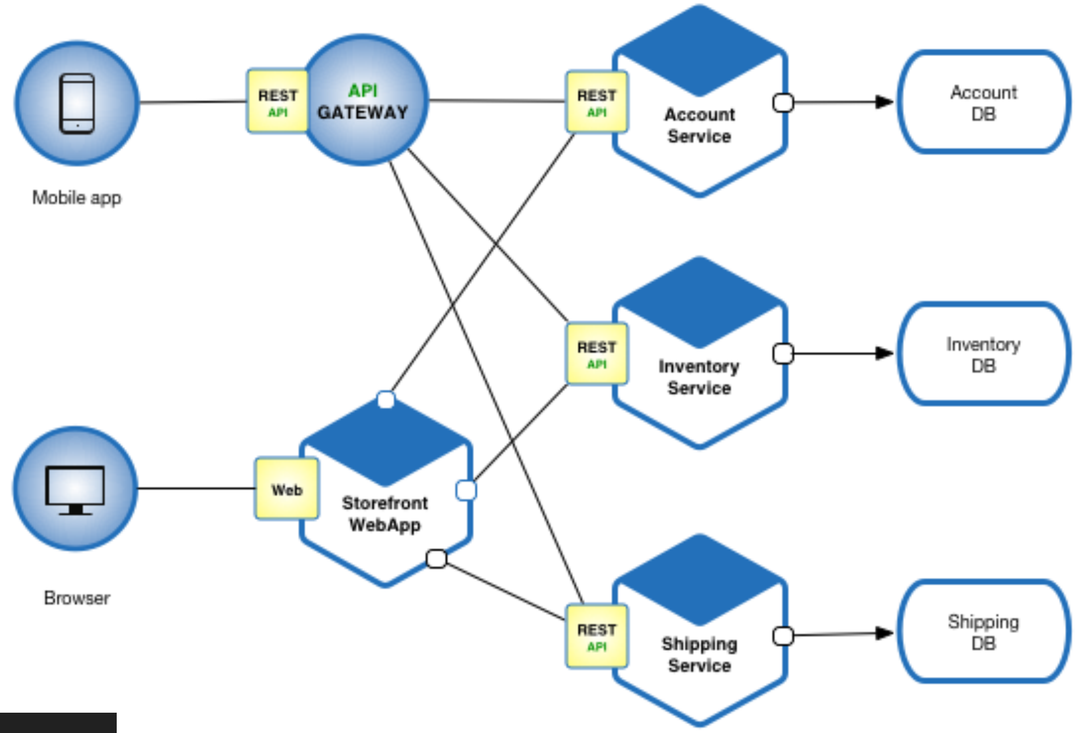


Microservices Architecture

Introduction

Microservices architecture breaks down a large application into smaller, independent services.

- **Loosely coupled**: Each service operates independently.
- **Single responsibility**: Each service focuses on one functionality or business domain.
- **Decentralised data management**: Services manage their own data, which may lead to having different databases for different services.

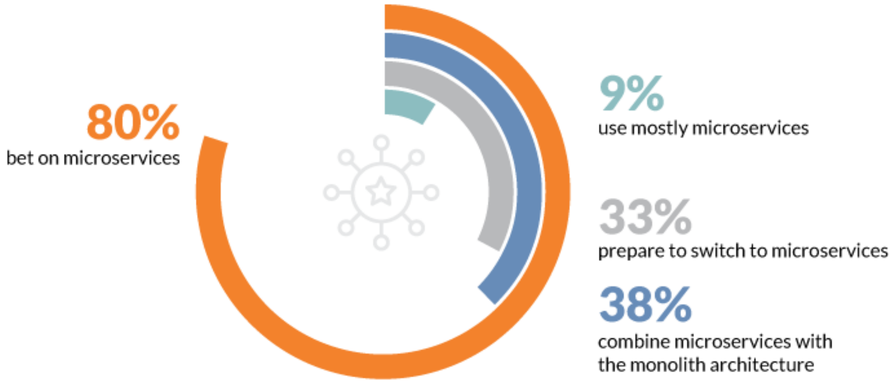


Imagine you're building a large house, but instead of constructing it all at once, you build each room separately. Each room is self-contained and functions independently. If you need to repair or modify one room, you can do so without affecting the rest of the house.

Microservices Architecture

Benefits

- **Scalability**: Services can be scaled independently based on demand. For example, you can scale the payment service without scaling the user management service.
- **Flexibility in technology**: Different services can be built using different programming languages or technologies.
- **Fault isolation**: If one service fails, it doesn't bring down the entire system.



Companies that moved to microservices

Airbnb

Netflix

Uber

LinkedIn

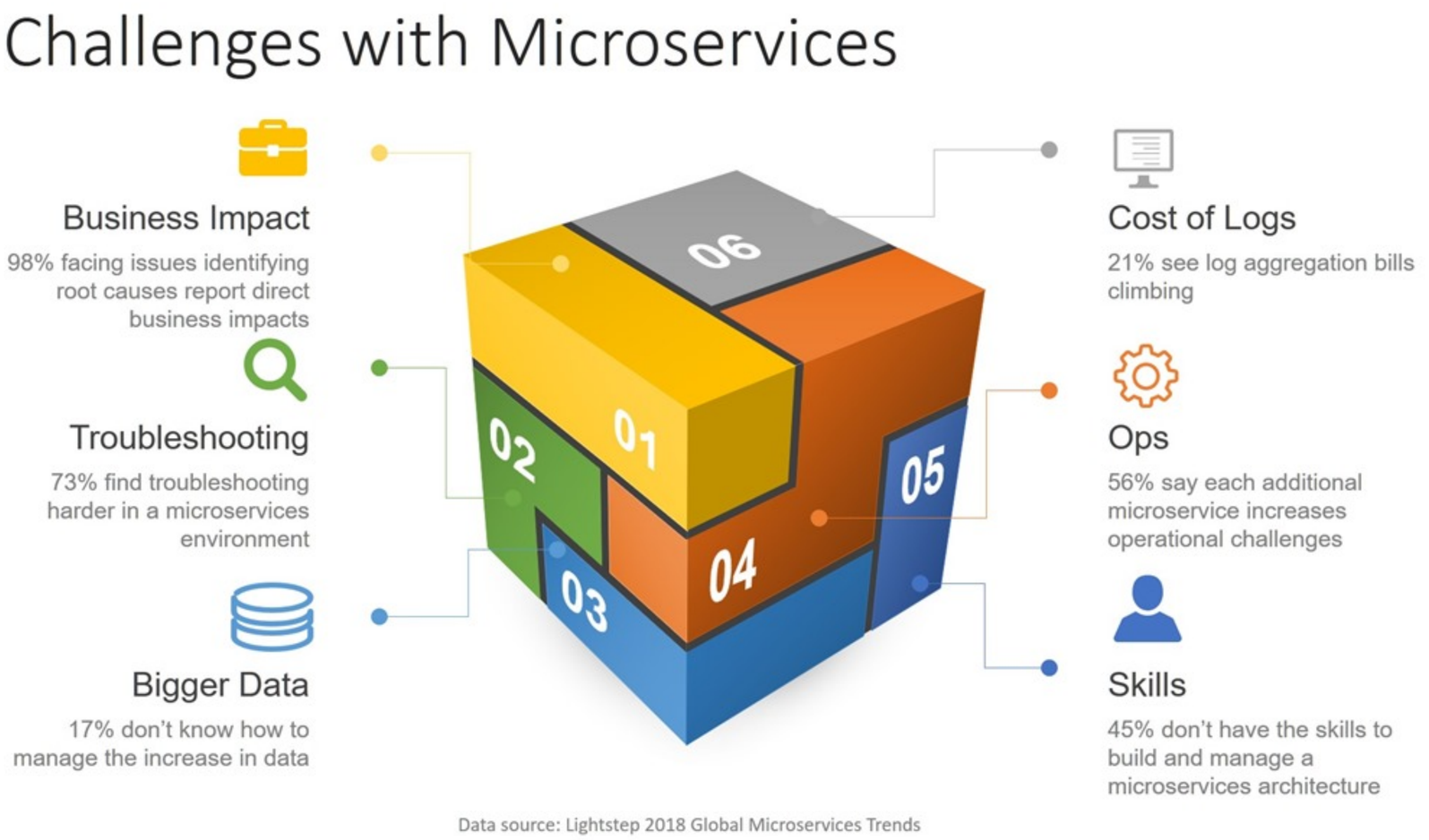
PayPal

The Guardian

eBay

Amazon

Microservices Architecture



Monolithic vs Microservices

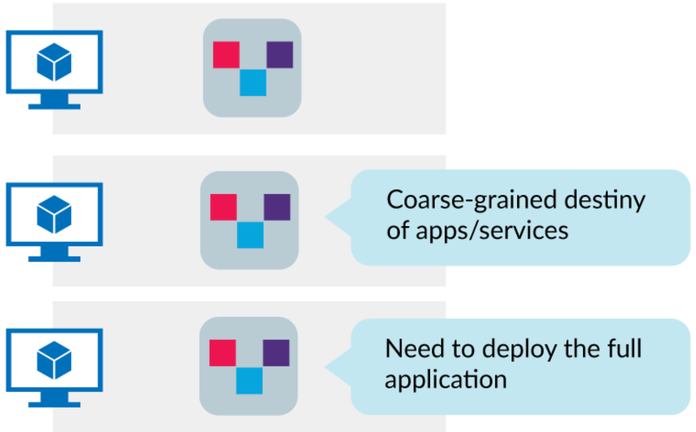
Aspect	Monolithic	Microservices
Codebase	Single, unified codebase	Multiple, independent codebases
Deployment	Deployed as a single unit	Each service is deployed independently
Scalability	Difficult to scale parts individually	Easy to scale individual services
Technology	Limited to one stack	Different services can use different stacks
Fault Isolation	Failures affect the whole system	Failures are isolated to individual services

Recap:

Monolithic deployment approach

A traditional application has most of its functionality within a few processes that are componentized with layers and libraries

Scales by cloning the app on multiple servers/VMs



Microservices application approach

A microservice application segregates functionality into separate smaller services.

Scales out by deploying each service independently with multiple instances across servers/VMs