WLUG
COMMUNITY | KNOWLEDGE | SHARE

MICROMANIA

THE FOSS FILES SEASON 3 | EPISODE 3

Ops Unleashed

Aditya Aparadh

Ameya Unchagonkar

Tushar Rathod

Aradhya Pitlawar

CONTENTS

BEFORE MICROSERVICES

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

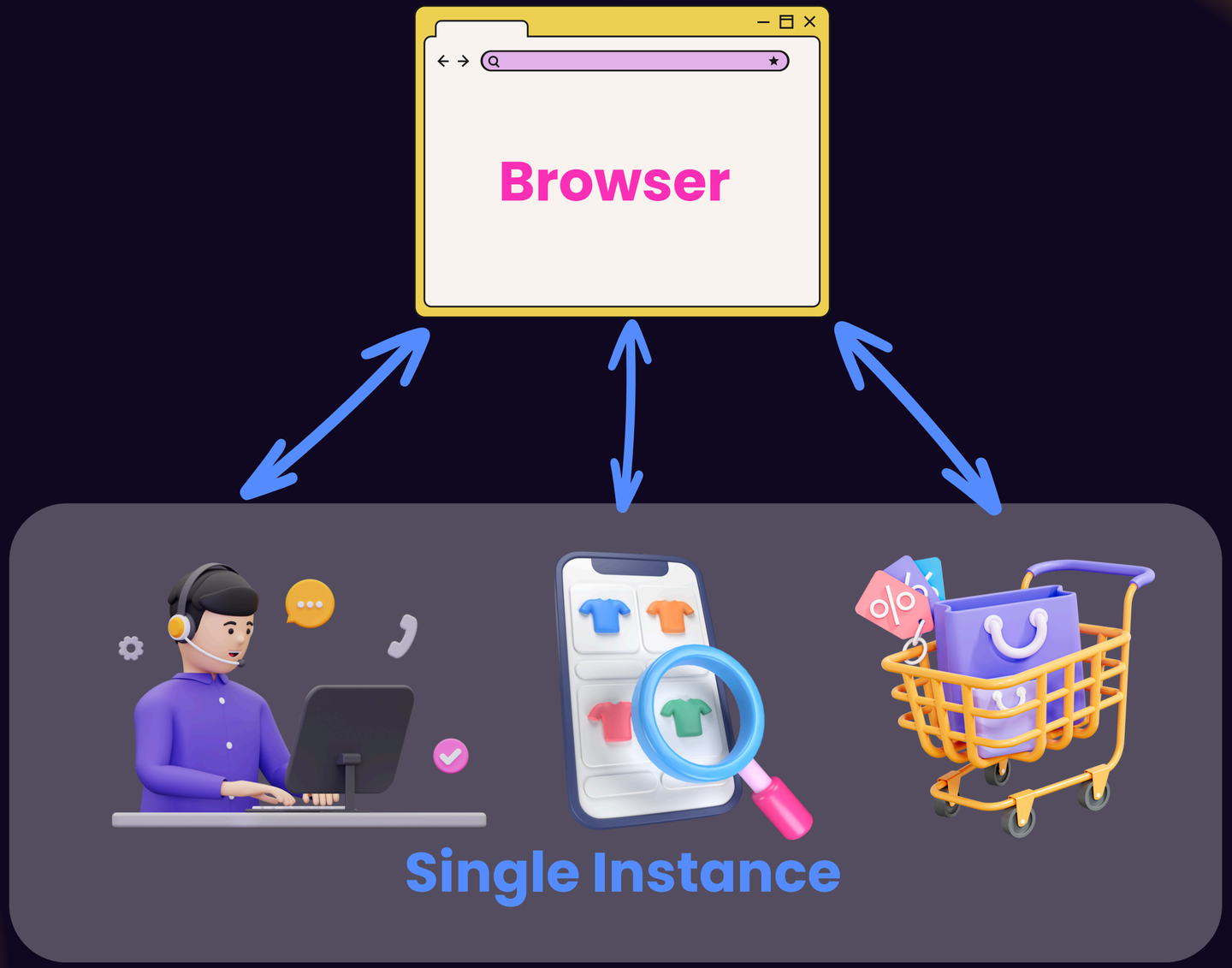
MONOLITHIC

BEFORE MICROSERVICES

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

MONOLITHIC

Example



EXAMPLE


BUGS IN OUR 10 YEARS
OLD MONOLITHIC APPLICATION
EVER GIVEN
OUR DEV TEAM


TRIED USING MONOLITHIC
ARCHITECTURE
SEEMS TO WORK FINE I GUESS

WLUG
COMMUNITY | KNOWLEDGE | SHARE

ADVANTAGES

- Based on the idea that "Software is a single entity with no modularization"
- Single indivisible unit.
- Straightforward and easy to implement
- Easy to develop and deploy
- Cost effective.

ADVANTAGES

CHALLENGES

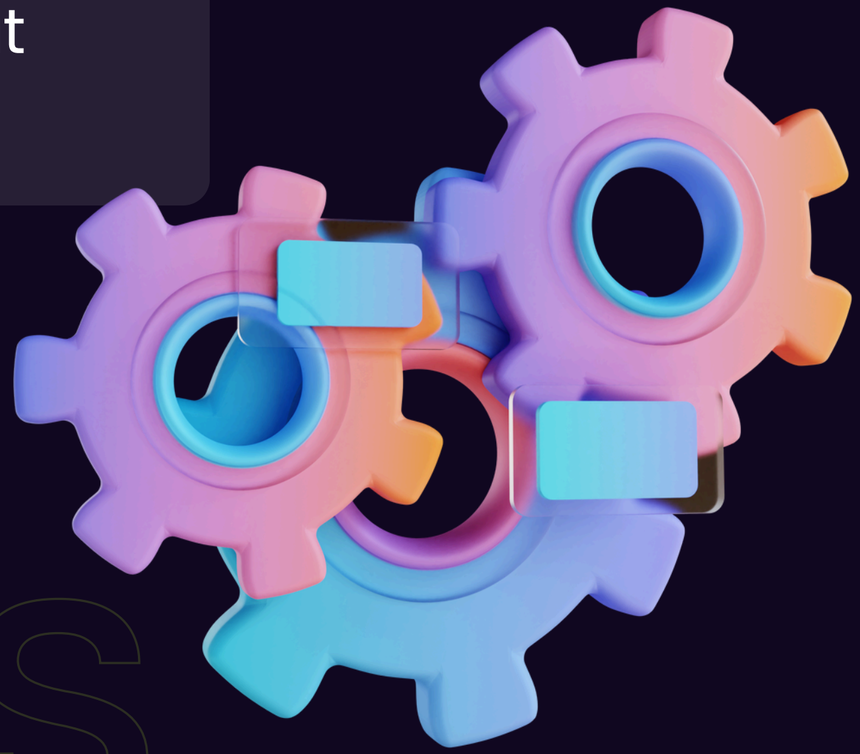
1. Large and Complex
2. Slow Development
3. Unscalable
4. Unreliable
5. Unflexible

CHALLENGES

WHAT ARE SERVICES?

- A service is like a helpful task or job that someone or something does for you.

eg. Messaging Service, Food Delivery Service, Streaming Service,



SERVICES

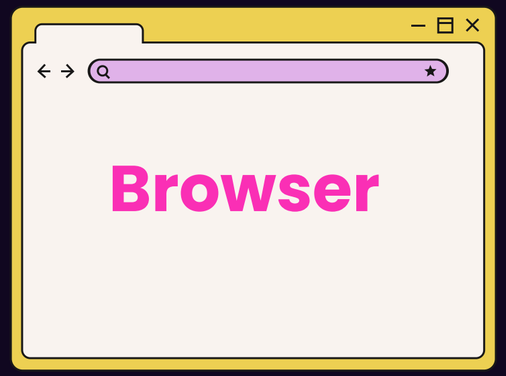
WLUG
COMMUNITY | KNOWLEDGE | SHARE

WHAT ARE MICROSERVICES?

Microservices is a way of building software applications as a collection of small, independent services that work together to create a larger and more complex application.

MICROSERVICES

Example



EXAMPLE


Pros
Cons
1.
2.

Advantages

Scalability

Flexibility

Rapid Deployment

Team Autonomy

Disadvantages

Distributed System Overhead

Cost of Technology

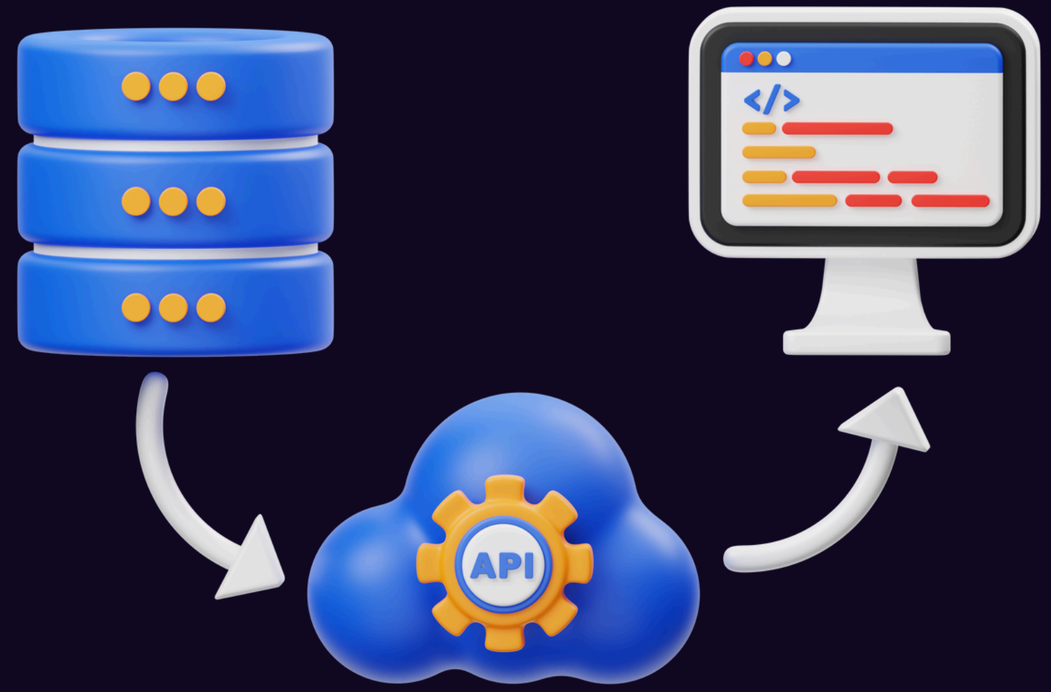
Testing Challenges

Data Consistency

Microservices Communication

COMMUNICATION VIA API

- Each service has its own API
- They can talk to each other, by sending requests to the respective API endpoint
- Synchronous Communication



API

COMMUNICATION VIA API

User account

Cart

HTTP Response

HTTP Request

Payment

Checkout

HTTP Response

HTTP Request

API

COMMUNICATION VIA MESSAGE BROKER

- Asynchronous Communication
- Communication via messages
- Common distribution patterns: Publish/subscribe and Point-to-point messaging

API

CODE MANAGEMENT

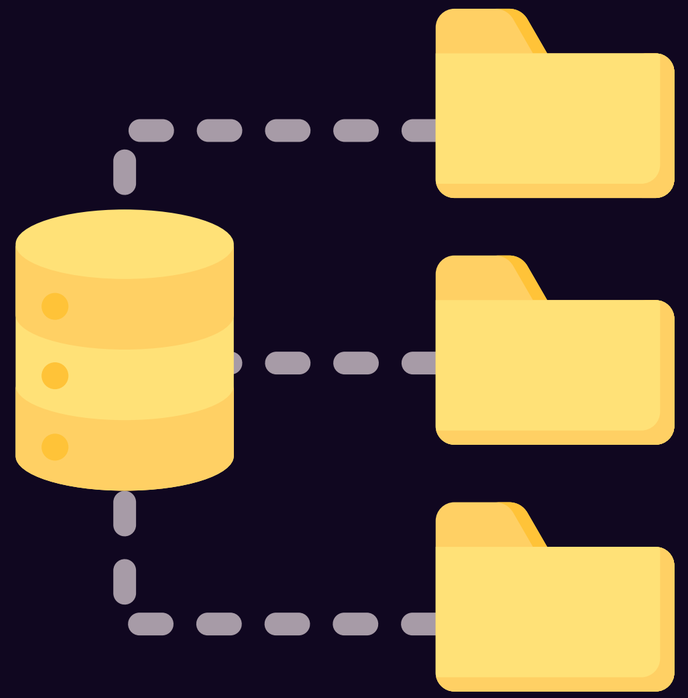
MONOREPO VS POLYREPO

CODE

WLUG
COMMUNITY | KNOWLEDGE | SHARE

MONOREPO

- Makes the code management and development easier
- Clone and work only with 1 repo
- Changes can be tracked together, tested together and released together
- Share code and configurations



MONOREPO

WLUG
COMMUNITY | KNOWLEDGE | SHARE

POLYREPO

- Makes the code management and development easier
- Clone and work only with 1 repo
- Changes can be tracked together, tested together and released together
- Share code and configurations

- My Online Shop


Shopping Cart


Payment


Notifications

POLYREPO

WLUG
COMMUNITY | KNOWLEDGE | SHARE

Monolith vs Microservices

WLUG
COMMUNITY | KNOWLEDGE | SHARE

ARCHITECTURE

- Single, unified application where all components are tightly integrated and share the same codebase.
- Modifying part requires redeploying the entire system, which increases interconnection and complexity in development and deployment.
- Decomposed into small, independent services, each with its own codebase and database.
- Microservices allow teams to independently develop, deploy, and scale services, promoting autonomy and reducing dependencies.

WLUG
COMMUNITY | KNOWLEDGE | SHARE

SCALABILITY

- Scaled as a whole, even if only a specific component requires additional resources.
- Monoliths typically scale vertically by increasing the resources (CPU, RAM) of the entire application.
- Individual services can be scaled independently, allowing for more efficient resource allocation.
- Resources can be allocated more efficiently as only the necessary microservices are scaled, reducing waste.

WLUG
COMMUNITY | KNOWLEDGE | SHARE

DEVELOPMENT AND DEPLOYMENT

- Single development and deployment unit, making updates and releases slower and riskier.
- Deployed as a single unit, requiring coordinated releases of the entire application.
- Independent development and deployment of services, enabling faster, more frequent updates.
- Independent deployment allowing teams to release updates to individual services without affecting the entire application.

WLUG
COMMUNITY | KNOWLEDGE | SHARE

Fault Isolation

- Components are tightly coupled, meaning a fault in one part can affect the entire application
- A failure in one module or component can potentially bring down the entire monolithic application.

- Failures are contained within specific services, limiting the impact on the overall system.
- Provide better fault isolation, as issues in one service are less likely to impact others due to their independent nature.

WLUG
COMMUNITY | KNOWLEDGE | SHARE

Technology Stack

- Uniform technology stack throughout the application.
- Constrained to the technology choices made during the initial development of the monolith.
- Supports diverse technologies and programming languages for different services.
- Each microservice can use its own database technology, enabling the use of the most suitable database for each service.

WLUG
COMMUNITY | KNOWLEDGE | SHARE

Communication

- Components communicate directly through in-process method calls or function invocations.
- Inter-service communication often involves lightweight protocols like HTTP OR APIs

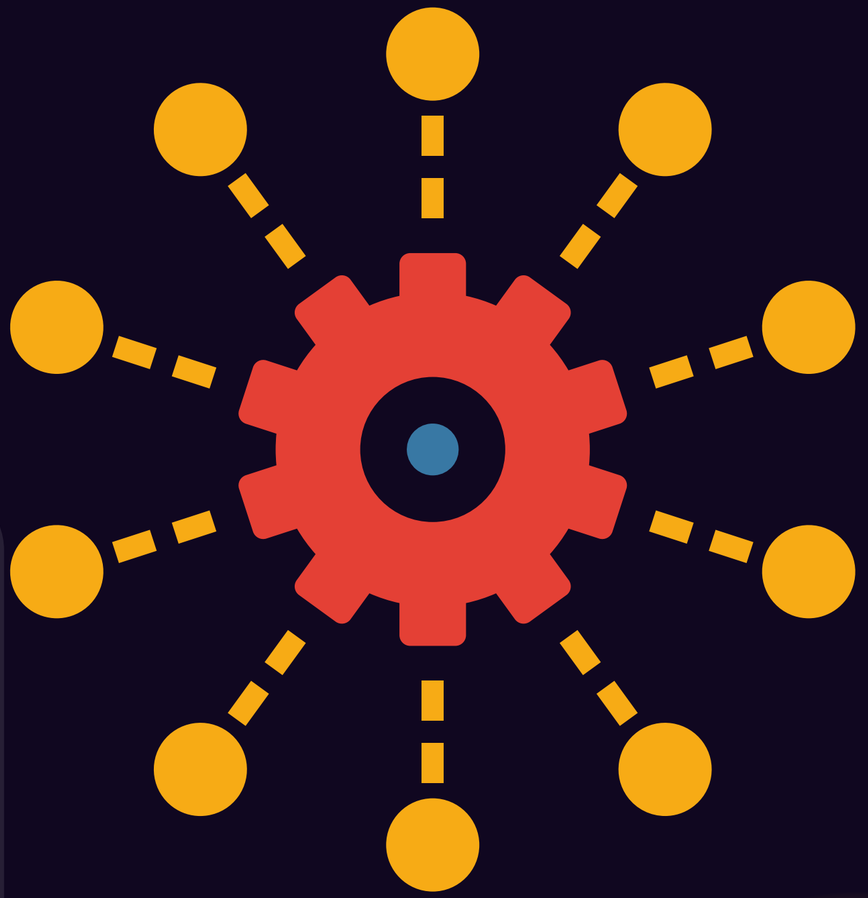
WLUG
COMMUNITY | KNOWLEDGE | SHARE

Flexibility and Modularity

- Lack of modularity, making it challenging to adapt to new technologies.
- Teams working on a monolith may face bottlenecks as they need to coordinate changes in a shared codebase.
- Modular design allows for easier adaptation to new frameworks and languages.
- Different teams can work on different microservices independently, allowing for faster development cycles.

WLUG
COMMUNITY | KNOWLEDGE | SHARE

MICROSERVICES FRAMEWORKS



- Microservices frameworks are a set of tools that simplify the development, deployment, and management of microservices-based architectures
- There are many frameworks across programming languages

WLUG
COMMUNITY | KNOWLEDGE | SHARE

FRAMEWORKS

WLUG
COMMUNITY | KNOWLEDGE | SHARE

NODEJS

- Modular Architecture
- Dependency Injection
- Decorators and Metadata
- Built-in Microservices Support
- WebSocket Support and Real-time Communication

NODEJS

WLUG
COMMUNITY | KNOWLEDGE | SHARE

SPRING

- Simplified Configuration
- Embedded Web Server
- Microservices Support
- Spring Ecosystem Integration
- Built-in Production-ready Features

SPRING

WLUG
COMMUNITY | KNOWLEDGE | SHARE

DJANGO

- Django REST Framework (DRF)
- Separation of Concerns
- Database Sharding and Separation
- Celery for Asynchronous Processing
- Message Brokers for Inter-Service Communication

DJANGO

WLUG
COMMUNITY | KNOWLEDGE | SHARE

GO

- Concurrency and Goroutines
- Standard Library for HTTP Handling
- gRPC for Efficient Communication
- Dependency Management with Go Modules
- Containerization with Docker

GO

WLUG
COMMUNITY | KNOWLEDGE | SHARE

CONCLUSION

“Small parts working together, making big things happen in software."

MICROSERVICES

WLUG
COMMUNITY | KNOWLEDGE | SHARE

THANK YOU

Community | Knowledge | Share

Aditya Aparadh

Ameya Unchagonkar

Tushar Rathod

Aradhya Pitlawar