

Modern Software Architecture

Backend • Frontend • DevOps

Muammar Zikri Aksana

Foundational Attributes

CORE CONCEPTS



Scalability

Ability to handle growth in users, data, or traffic without performance loss.



Reliability

Ensuring the system functions correctly even when components fail.



Maintainability

Ease of updating, fixing, and evolving the codebase over time.



Security

Protecting data and resources from unauthorized access and attacks.



Performance

Responsiveness and efficiency of the system under varying loads.



Observability

Understanding internal system state based on external outputs.

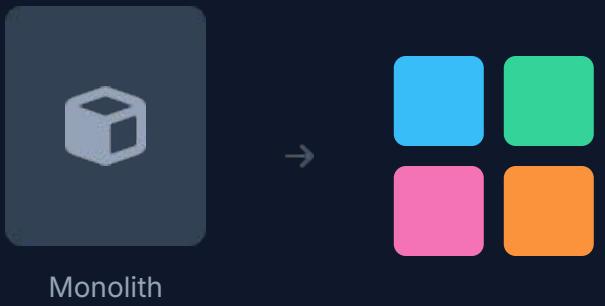
Architectural Styles

BACKEND

Monolithic vs. Microservices

The choice between a unified codebase and distributed services defines your scaling strategy.

- ✓ **Monolithic:** Simple to deploy and test, but tightly coupled.
Scaling requires replicating the entire app.
- ✓ **Microservices:** Independent deployment and scaling per service. Complex to manage networking and data consistency.



Database Scaling: Sharding

BACKEND DATA

Horizontal Scaling

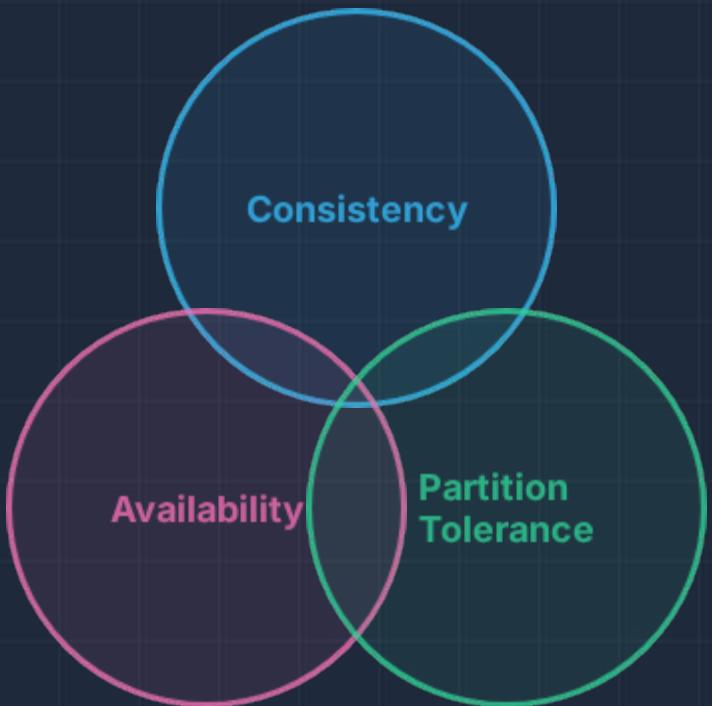
When a single database server cannot handle the load, we split the data across multiple servers (shards).

- ✓ **Sharding Key:** The data attribute (e.g., UserID) used to determine which shard holds the data.
- ✓ **Hot Partitions:** A risk where one shard receives disproportionate traffic.
- ✓ **Complexity:** Joins across shards are difficult and slow.



Distributed Consistency

BACKEND DATA



CAP Theorem

In a distributed system, you can only guarantee two of the three:

- ✓ **Consistency:** Every read receives the most recent write.
- ✓ **Availability:** Every request receives a response (even if data is old).
- ✓ **Partition Tolerance:** The system continues to operate despite network failures.
- ✓ Most modern internet systems choose **AP** (Availability + Partition Tolerance) and rely on **Eventual Consistency**.

Communication Patterns

BACKEND COMMS



Synchronous

Request-Response (REST, gRPC)

The client waits for the response. Simple but tightly couples services and creates blocking calls.



Event-Driven Architecture (EDA)

Asynchronous (Kafka, Pub/Sub)

Services publish events that others consume. Decouples services, allowing for better scalability and resilience.

Stability & Performance

BACKEND STABILITY



Caching

Storing hot data in memory

(Redis/Memcached) to reduce DB load and
latency.



Circuit Breaker

Detects failures and stops calling a failing
service to prevent cascading system
crashes.



Retries & Backoff

Automatically retrying failed requests with
exponential delays to handle transient
network blips.

Architectural Design Patterns

BACKEND DESIGN



CQRS

Command Query Responsibility Segregation

Separates the models used for updating (Commands) from the models used for reading (Queries). Optimizes performance for heavy read/write systems.



Saga

Distributed Transaction Management

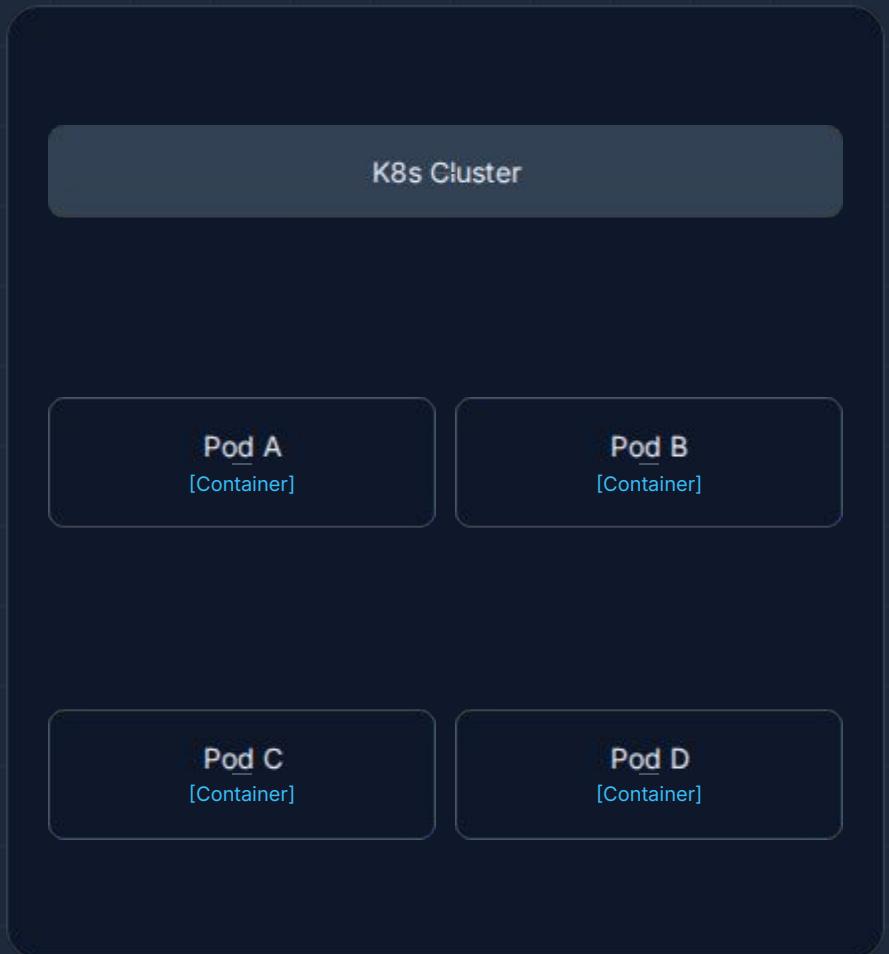
A sequence of local transactions. Used to maintain data consistency across multiple microservices without 2-phase commit.

Containers & Orchestration

DEVOPS

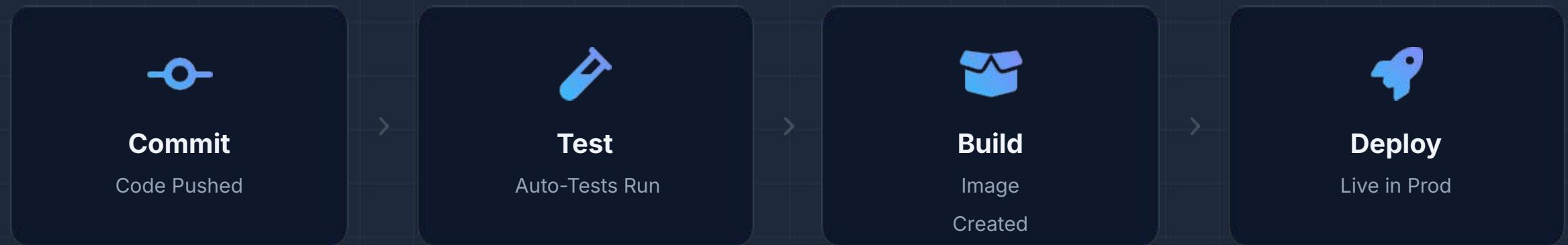
Running Anywhere

- ✓ **Docker (Containers):** Packages code and dependencies together. "It works on my machine" becomes "It works everywhere."
- ✓ **Kubernetes (Orchestration):** Manages thousands of containers. Handles scaling, self-healing, and load balancing automatically.



CI/CD & Automation

DEVOPS



- ✓ **Continuous Integration (CI):** Merging code changes frequently and verifying them with automated tests.
- ✓ **Infrastructure as Code (IaC):** Managing infrastructure (Terraform, Ansible) through code files, ensuring reproducibility.

Frontend Rendering

FRONTEND



CSR

Client-Side Rendering

Browser does the work. Fast interactions,
slow initial load.



SSR

Server-Side Rendering

Server sends HTML. Good for SEO and slow
devices.



SSG

Static Generation

Pre-built HTML. Instant load, difficult for
dynamic data.

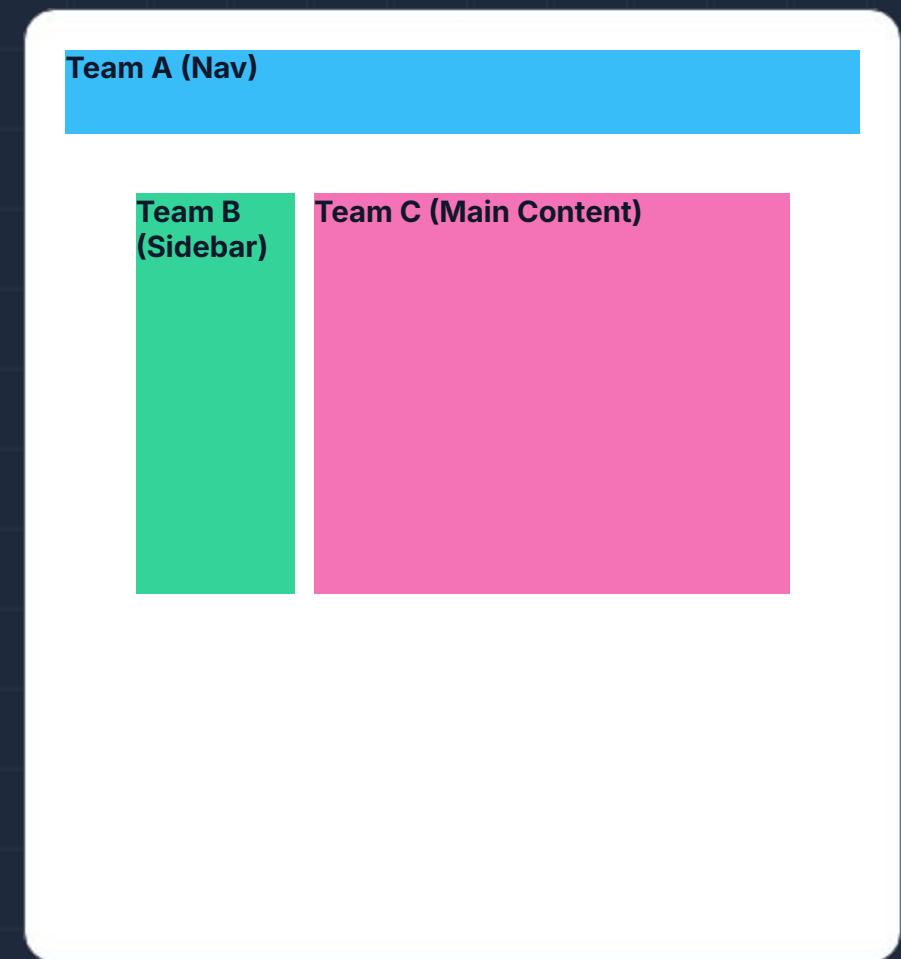
Micro-Frontends

FRONTEND

Scaling the Frontend

Just as microservices split the backend, micro-frontends split the UI into independent pieces.

- ✓ **Independence:** Different teams can build the Header, Checkout, and Profile using different frameworks (React, Vue).
- ✓ **Composition:** A container app stitches them together for the user.



Key Takeaways

SUMMARY

Architecture is a Journey

- ✓ **No Silver Bullet:** Every decision (Sharding, Microservices, SSR) has trade-offs.
- ✓ **Evolve:** Start simple (Monolith), measure bottlenecks, and then optimize.
- ✓ **Holistic View:** A modern architect understands the data, the infrastructure, and the user interface.



Goal

Build systems that are easy to change
and hard to break.