Faculty of Computing and IT

CCS3300 Software Architecture

Week 3 : Software Quality Attributes: Building Robust and Reliable Systems

Learning Outcome Achieved:

ILO2 - Define and understand the quality attributes of software architecture.

by Asanka Ranasinghe

Contact : 070 7 432 737

Email : asanka.r@sltc.ac.lk

Defining Quality Attributes

Non-functional Requirements

These define how a system behaves. They impact user experience.

System Operation

They govern the system's overall function. This is beyond specific features.

Impact on Success

Directly affects user satisfaction. Crucial for reliability and maintainability.

Importance of Quality Attributes

Boost Customer Satisfaction

Increases loyalty through positive experiences.

Minimize Risks

Reduces failures and security breaches.

Reduce Long-Term Costs

Lowers maintenance and enhancement expenses.

Real-world Example

Amazon's uptime builds trust and revenue.

Key Quality Attributes: Security

Unauthorized Access

Protects against unwanted intrusions.

Data Breaches

Safeguards sensitive information.

Confidentiality

Ensures privacy of data.

Integrity

Maintains data accuracy and completeness.

Availability

Ensures access to data when needed.

Regulatory Compliance

Adheres to standards like GDPR and HIPAA.

Key Quality Attributes: Usability

Ease of Use

Simple and straightforward interactions.

Learnability

New users can quickly understand the system.

User Satisfaction

Positive experience promotes continued use.

Efficiency

Users perform tasks quickly and effectively.

Reduced Errors

Intuitive design minimizes mistakes.

Lower Support Costs

Fewer user issues mean less help needed.

Key Quality Attributes: Performance

Responsiveness

How quickly the system reacts.

Efficiency

How well resources are used.

Key Metrics

Measured by response time, throughput, latency.

Scalability Impact

Affects user experience and growth.

Performance Key Metrics

Response Time

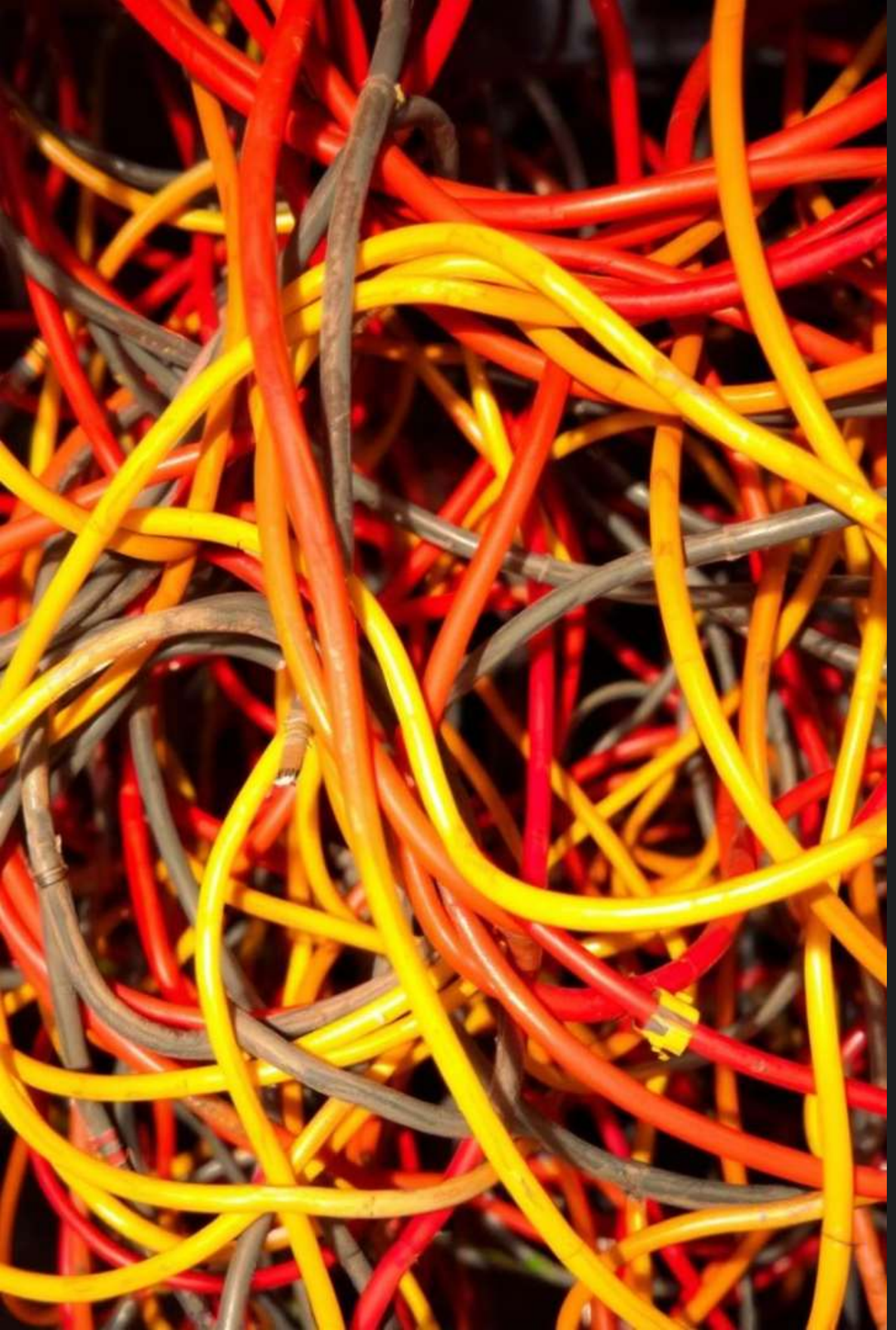
Time to complete a single request. Goal: Milliseconds or less for user-facing operations. Google search averages <500ms.

Throughput

Number of requests handled per unit time. Measured as requests per second (RPS). Netflix peaks at 400 Gbps.

Latency

Delay between request and response. High latency means poor user experience. Fiber optic cables reduce latency.



Architecture's Influence on Performance

Bottlenecks

Poor architecture creates bottlenecks and single points of failure.

Monolithic Architecture

Changes require full redeployment, leading to downtime.

Microservices

Independent deployment improves fault tolerance and reduces downtime by 80%.

Database Design

Inefficient queries can significantly slow response times.

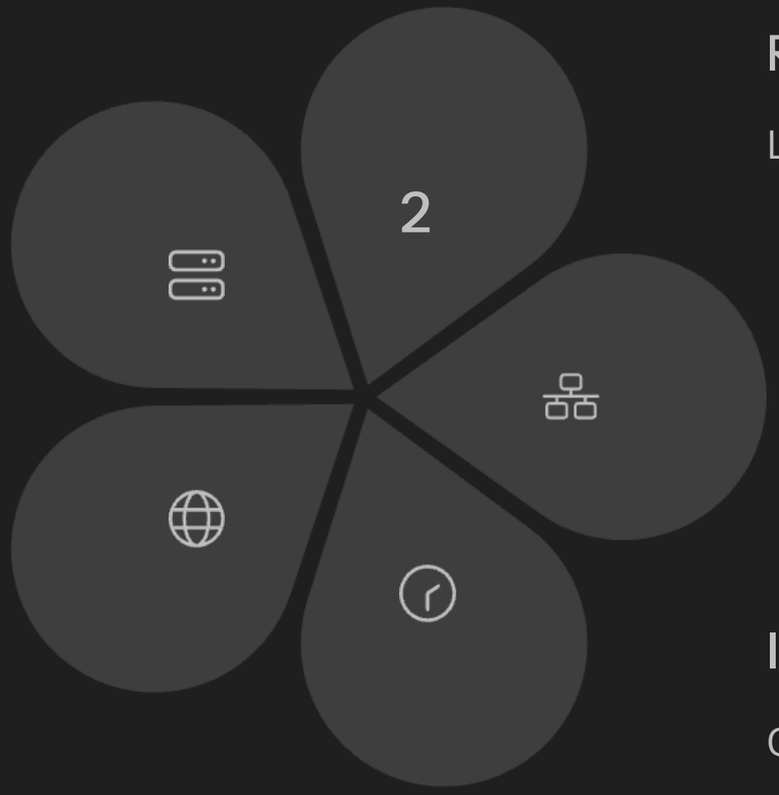
Performance Improvement Strategies: Caching

Caching Implementation

Various levels like client-side and server-side.

CDN Example

CDNs reduce load times by 60%.



Reduces Database Load

Less strain on core systems.

Lowers Network Latency

Faster data transfer across networks.

Improves Response Times

Quicker user interactions.

Performance Optimization Strategies: Load Balancing

Traffic Distribution

Distribute incoming traffic across multiple servers efficiently.

High Performance

HAProxy handles over 10,000 requests per second on commodity hardware.

Prevent Overload

Prevents single server overload, ensuring system availability.

Algorithm Choices

Utilize Round Robin, Least Connections, or IP Hash algorithms.

Performance Optimization Strategies: Database Indexing

↑

Improve Query Speed

Create indexes on frequently queried columns for faster data retrieval.

!

Careful Selection

Choose indexes carefully to avoid degrading write performance.

3

SQL Optimization

Optimizes SQL queries, speeding up data retrieval by 10x.

Defining Scalability

Handle Increased Load

Ability to handle increasing load without performance degradation.

Two Main Types

Vertical and Horizontal scaling are the primary methods.

Measured by Metrics

Load capacity, resource utilization, and cost efficiency determine scalability.

Real-World Example

Amazon scales to handle 10x traffic during peak shopping days.

Scalability: Vertical vs. Horizontal

1

Vertical Scaling

Increase single server resources (CPU, RAM). Hardware constraints limit growth. Suited for smaller applications.

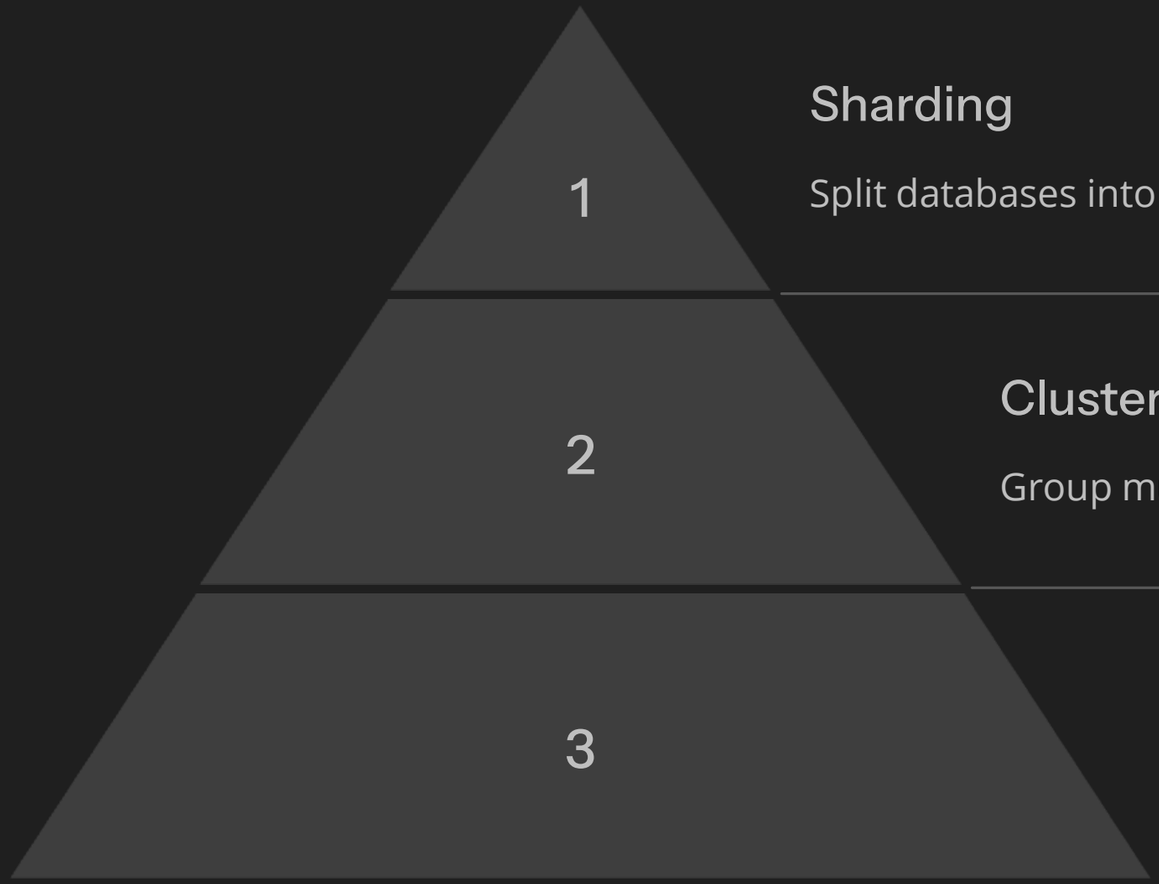
2

Horizontal Scaling

Add more servers to the pool. Offers high availability and fault tolerance. Preferred for large-scale applications.

Vertical scaling
vertical scaling

Architectural Approaches for Scalability



Sharding

Split databases into smaller, manageable pieces for distributed data.

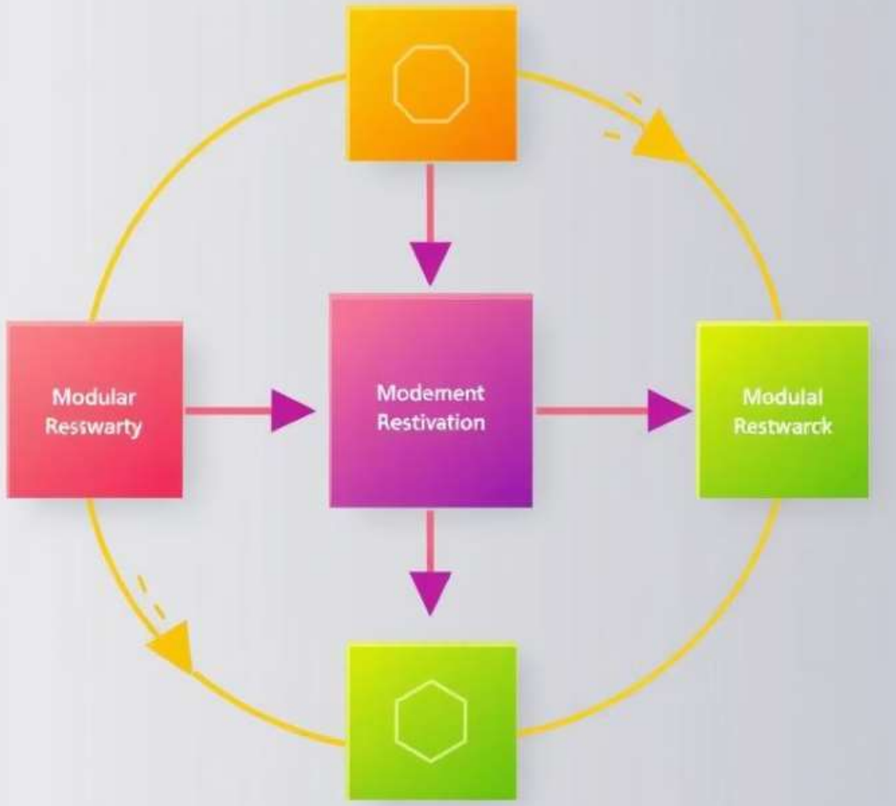
Clustering

Group multiple servers to work as a single, unified unit.

Partitioning

Divide data into logical segments for faster, more efficient retrieval.

These strategies are crucial for handling increased load. For instance, LinkedIn uses sharding to manage billions of user profiles effectively.



Defining Modifiability

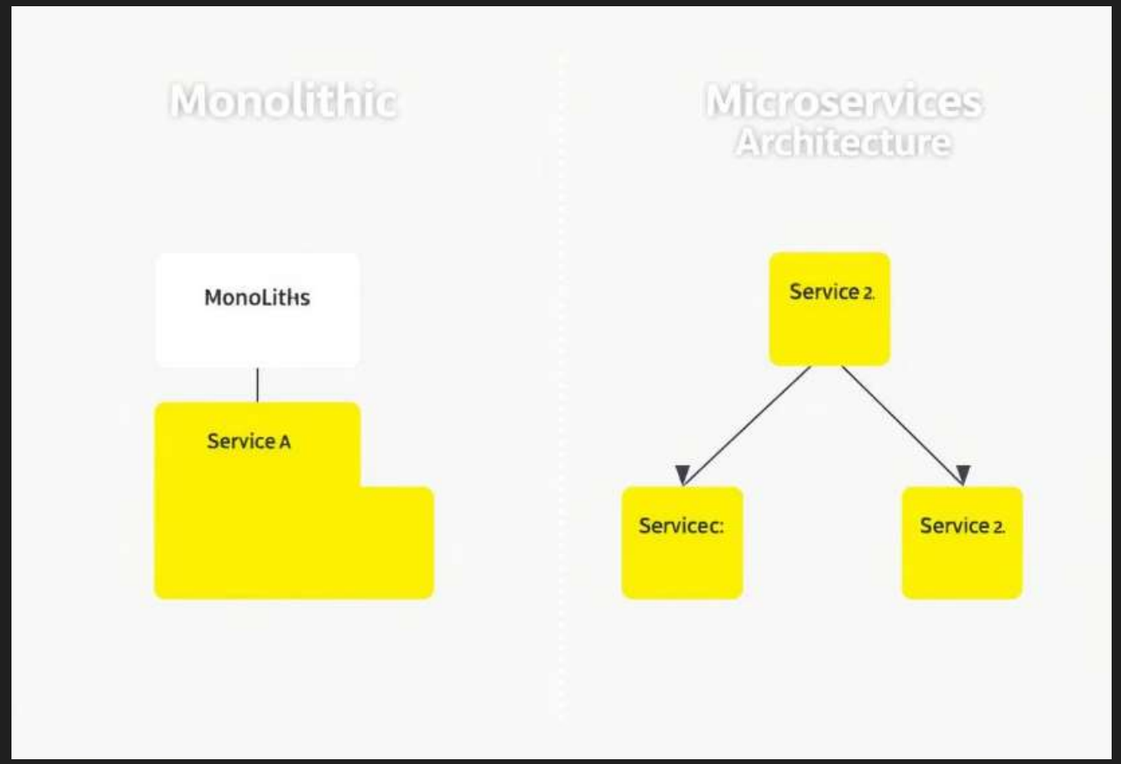
- Modifiability is how easily a system adapts to changes.
- It impacts development costs and time-to-market significantly.
- Poor modifiability leads to technical debt and fragility.
- Metric: time and effort for implementing a specific change.

Architectural Impact on Modifiability

Centralized vs. Microservices

Centralized architectures, like monoliths, hinder modifiability due to tight coupling. Microservices, conversely, promote it through loose coupling and independent deployments.

Consider an e-commerce platform: a monolithic system makes adding new payment types costly due to complex interdependencies, whereas a microservices architecture allows independent updates to payment, product, and cart services, reducing effort and risk.



Modularity and Abstraction

Modularity involves dividing a system into independent, interchangeable modules. Abstraction hides complex implementation details behind simplified interfaces.

Modularity

Breaking down a system into smaller, self-contained units.

Abstraction

Presenting essential information while hiding complexity.

API Gateway

An API Gateway acts as an abstraction layer for backend services, simplifying client interactions.

Design Patterns for Modifiability

Design patterns offer proven solutions to recurring design problems, enhancing system modifiability. Netflix utilizes many of these patterns.

Model-View-Controller (MVC)

Separates application concerns: data (Model), presentation (View), and user interaction logic (Controller).

Observer Pattern

Enables one-to-many dependency between objects, ensuring loose coupling for event handling.

Strategy Pattern

Allows interchangeable algorithms at runtime, promoting flexibility and extensibility.

Defining Availability

Availability signifies the probability a system remains operational at any given time, measured as uptime percentage.

99.99%

Availability

Corresponds to approximately 52 minutes of downtime per year.

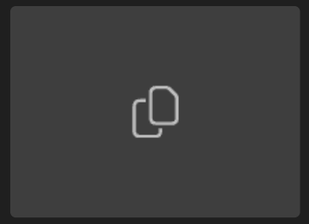
$

Downtime Costs

Includes financial losses, reputational damage, and SLA breaches.

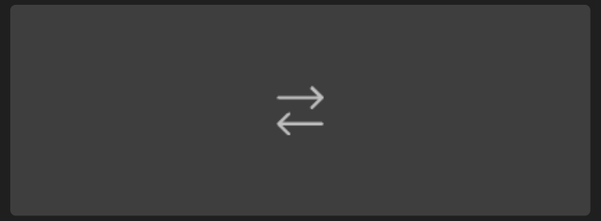
Architectural Solutions for High Availability

High availability ensures continuous operation through strategic architectural designs.



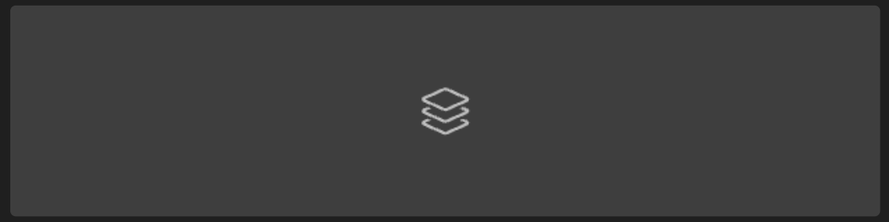
Redundancy

Duplicating components to eliminate single points of failure.



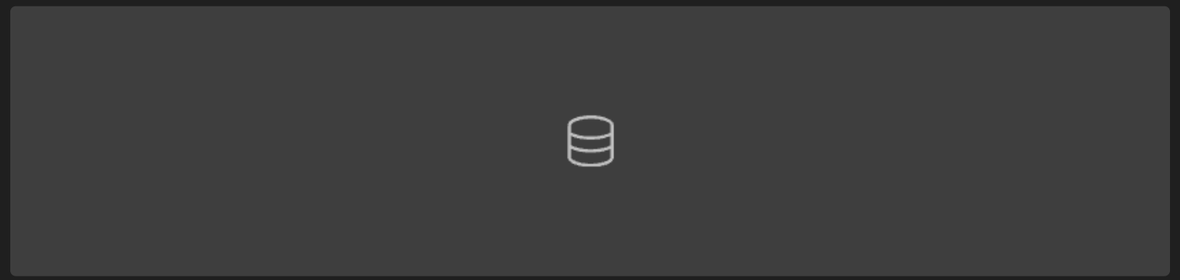
Failover Mechanisms

Automatic switching to backup systems during component failure.



Clustering

Running multiple application instances behind a load balancer.



Replication

Duplicating data across multiple servers (master-slave, multi-master).

Improving Availability

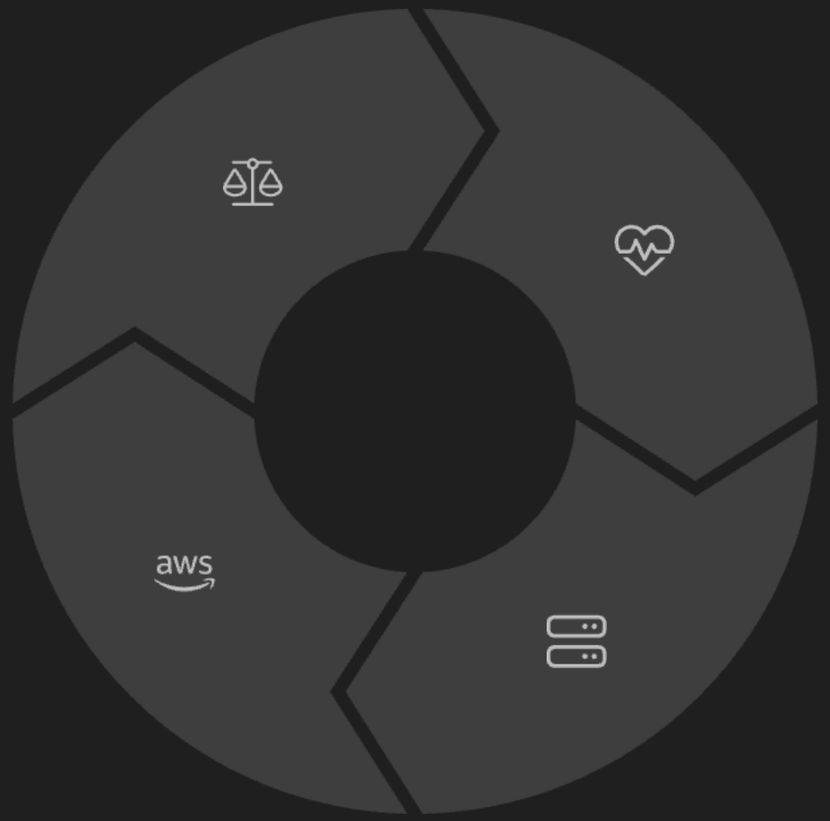
Enhance system uptime through strategic solutions.

Load Balancing

Distributes traffic across multiple servers efficiently.

Health Checks

Monitors system status for automated failover processes.



AWS Auto Scaling

Provides dynamic scaling and fault tolerance.

Database Replication

Ensures data durability and high availability.

Integration

Integration is crucial for connecting disparate systems and components, enabling seamless data flow and communication within a software ecosystem.

APIs

Standard interfaces like REST and GraphQL facilitate communication.

Message Queues

Enables asynchronous communication between services (e.g., Kafka).

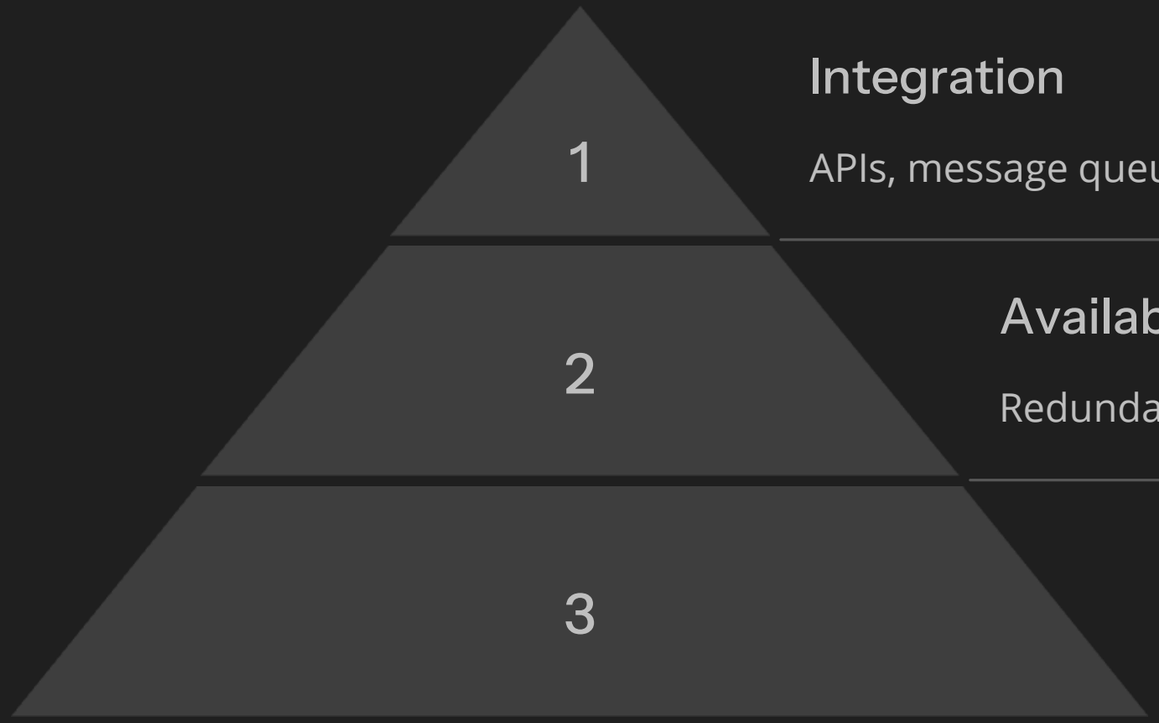
3

Data Integration

ETL processes consolidate data from multiple sources efficiently.

Key Architectural Considerations

Balancing these core principles creates robust and adaptable software systems, ensuring long-term success and resilience.



Integration

APIs, message queues, data integration strategies.

Availability

Redundancy, failover, clustering, replication.

Modifiability

Loose coupling, modularity, design patterns.

Why Integration Matters

Connected Systems

Modern systems rarely exist in isolation. They need to communicate.

Unified Data

Integration provides a single, unified view of critical information.

Business Agility

Systems adapt quickly to changing business needs and demands.

Enhanced Experience

Users enjoy seamless and smooth interactions across platforms.

The Challenge of Silos

Isolated Systems

Separate systems create redundant and duplicated data. This leads to inefficiencies.

Inconsistent Data

Discrepancies in data result in poor and misinformed business decisions.

Limited Visibility

Lack of overview hinders innovation and strategic planning. You can't see the full picture.

Breaking Silos

Integration solutions effectively break down these barriers. They connect disparate systems.

Service-Oriented Architecture (SOA)

Core Principles

SOA emphasizes services with well-defined interfaces. They are reusable.

Loose Coupling

Services evolve independently without affecting others. This ensures flexibility.

High Reusability

Leverage services across multiple applications. Avoid duplicate work.

An example includes integrating CRM, billing, and marketing systems. Each system is a service.

Microservices Architecture

Fine-Grained Services

Microservices are small, independently deployable units. They do one thing well.

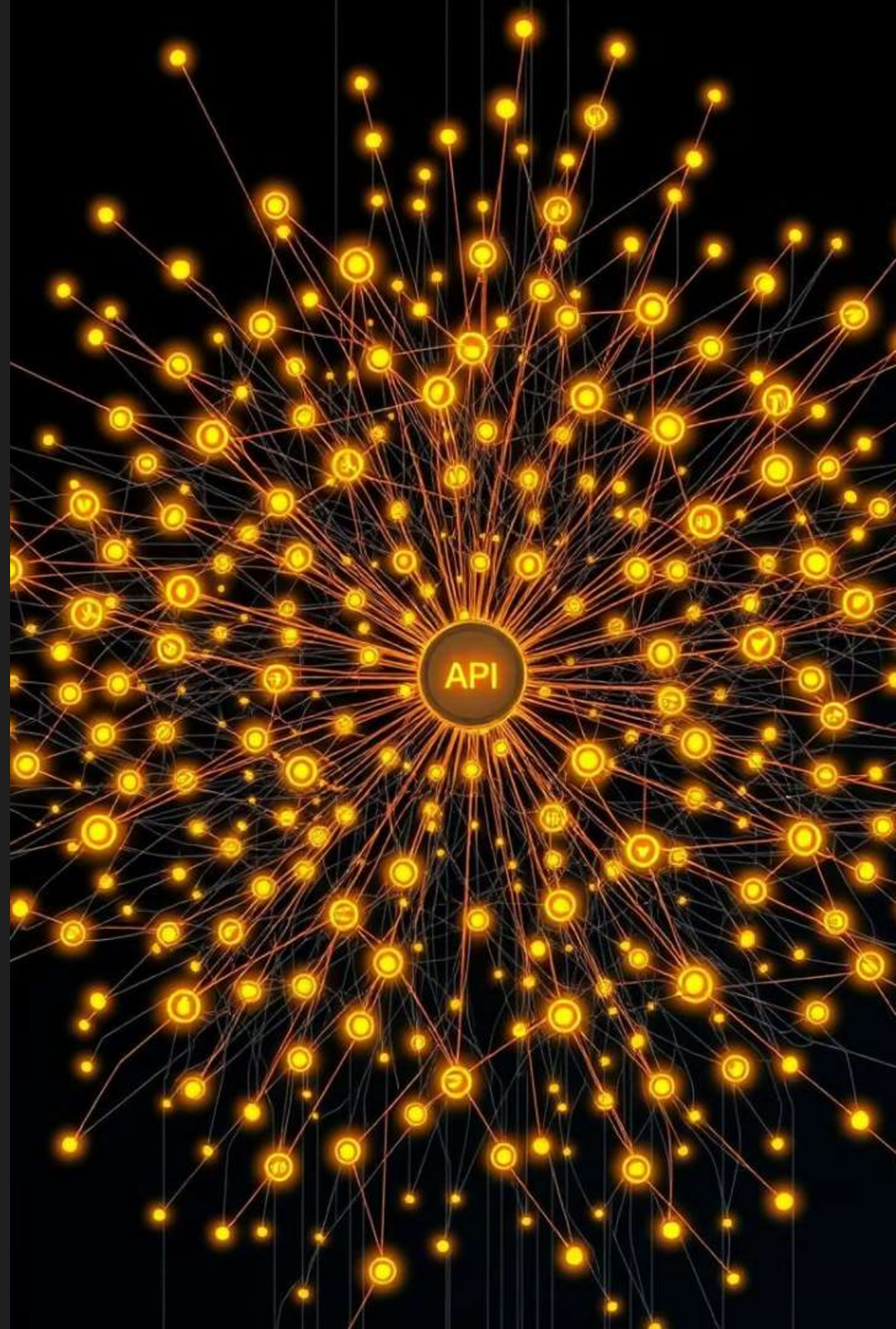
Decentralized Governance

Teams choose their own technologies. This fosters diversity.

API-First Approach

Emphasis is placed on clear APIs for seamless communication. Integration is key.

Netflix's architecture, composed of hundreds of small services, is a prime example of microservices in action. This enables scalability and resilience.



API Gateways

Single Entry Point

All external requests go through one gateway. This simplifies access.

Request Routing

Gateways direct requests to the correct backend microservices. Efficient delivery.

Security & Control

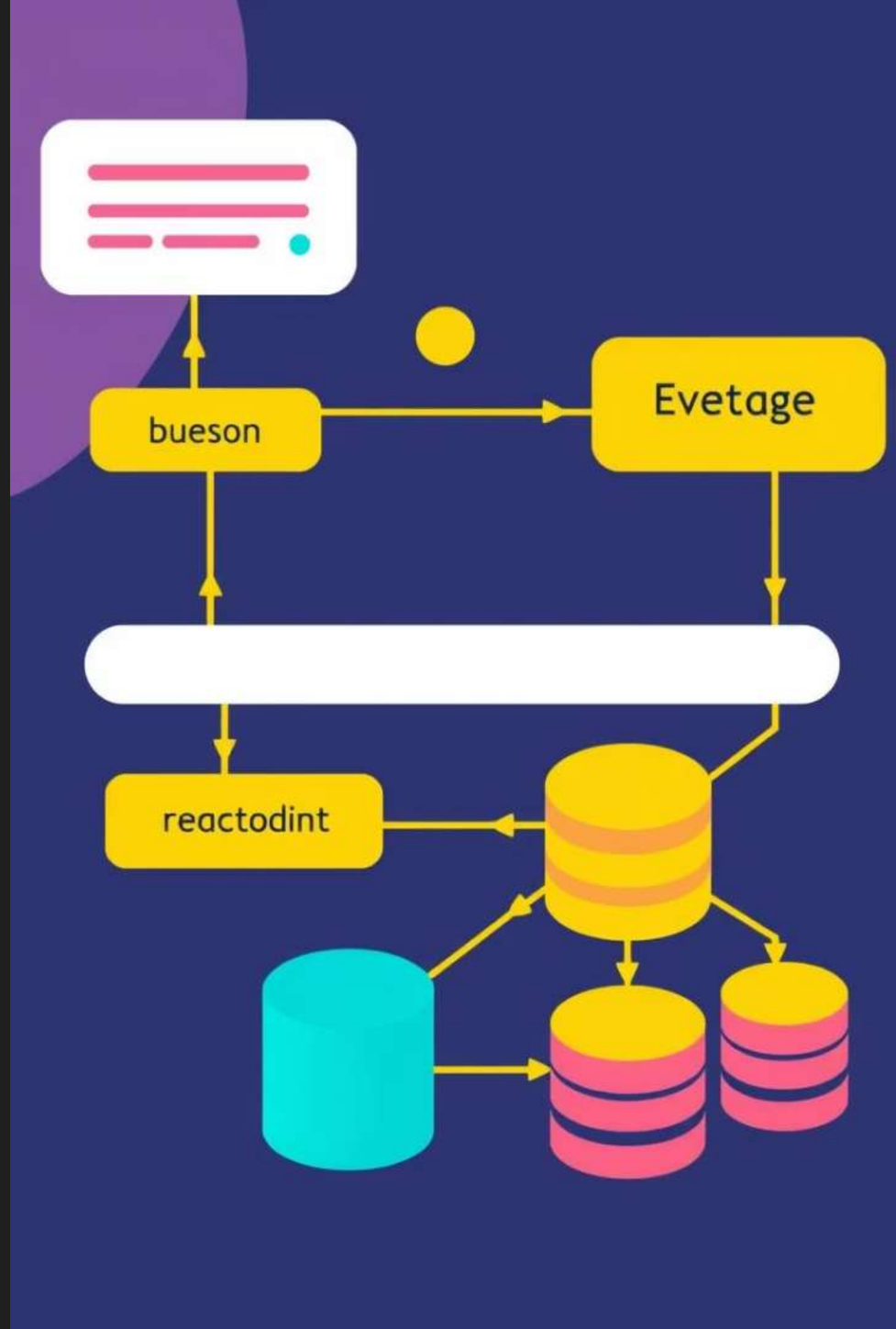
Handle authentication, rate limiting, and monitoring. Protect your services.

Kong API Gateway is a popular open-source choice for managing APIs. It provides robust features.

Integration Patterns

Message Queues	Enable asynchronous communication between systems.
Event-Driven Architecture	Systems react to real-time events for responsiveness.
Data Virtualization	Creates a unified access layer for diverse data sources.
APIs and Webhooks	Simplify external system integration and interaction.

These patterns offer different strategies for seamless system interaction. Choose the right tool for the job.



Balancing Quality Attributes for Success

High-Quality Software

Meets evolving user needs.

Optimize Systems

Balance performance, security, UX.

Strategic Approach

Essential for software development.

Q & A ?

Great work today, everyone!

Let’s keep building our knowledge, one lesson at a time. Until next time!