



# Introduction to Microservices

Ravindu Nirmal Fernando

SLIIT | March 2025

# Foundations of Modern Software Architecture: Paving the Way for Microservices

- Influential Concepts and Technologies
  - **Domain-Driven Design:** Emphasizing the importance of reflecting real-world complexities in our code for better system modeling.
  - **Continuous Delivery:** Revolutionizing software deployment, making every code check-in a potential release candidate.
  - **Web Communication Advancements:** Enhancing how machines interact, leading to more efficient and robust systems.
- Architectural Shifts
  - **From Layered to Hexagonal:** Moving away from traditional layered architectures to avoid hidden complexities in business logic.
  - **Embracing Virtualization:** Utilizing on-demand provisioning and resizing of resources for greater flexibility with cloud computing.
- Organizational Practices
  - **Small Autonomous Teams:** Inspired by tech giants like Amazon and Google, promoting ownership and lifecycle management of services.
  - **Learning from Netflix:** Building resilient, scalable systems that can withstand and adapt to change.

# Microservices: A Natural Progression

- Emergence from Real-World Use: Microservices weren't pre-planned but evolved as a response to practical needs in software development.
- Responding to Change: Offering the agility and flexibility to adapt to new technologies and market demands.

# Monolithic Applications

- **Basic Structure**

- Single-Tiered Structure: Built as a single, unified unit.
- Combined Modules: Functional modules like UI, server logic, and database interactions are combined.

- **Design and Construction**

- Modular Architecture: Follows a modular structure within a single unit, aligning with object-oriented principles.
- Programming Constructs: Defined using language-specific constructs (e.g., Java packages).
- Build Artifacts: Built as a single artifact, such as a Java JAR file.

- **Characteristics**

- Inter-module Dependencies: Modules are tightly coupled and interdependent.
- Unified Deployment: Deployed as a single entity.

- **Scalability**

- Scalability Approach: Scaling involves replicating the entire application, not individual components.

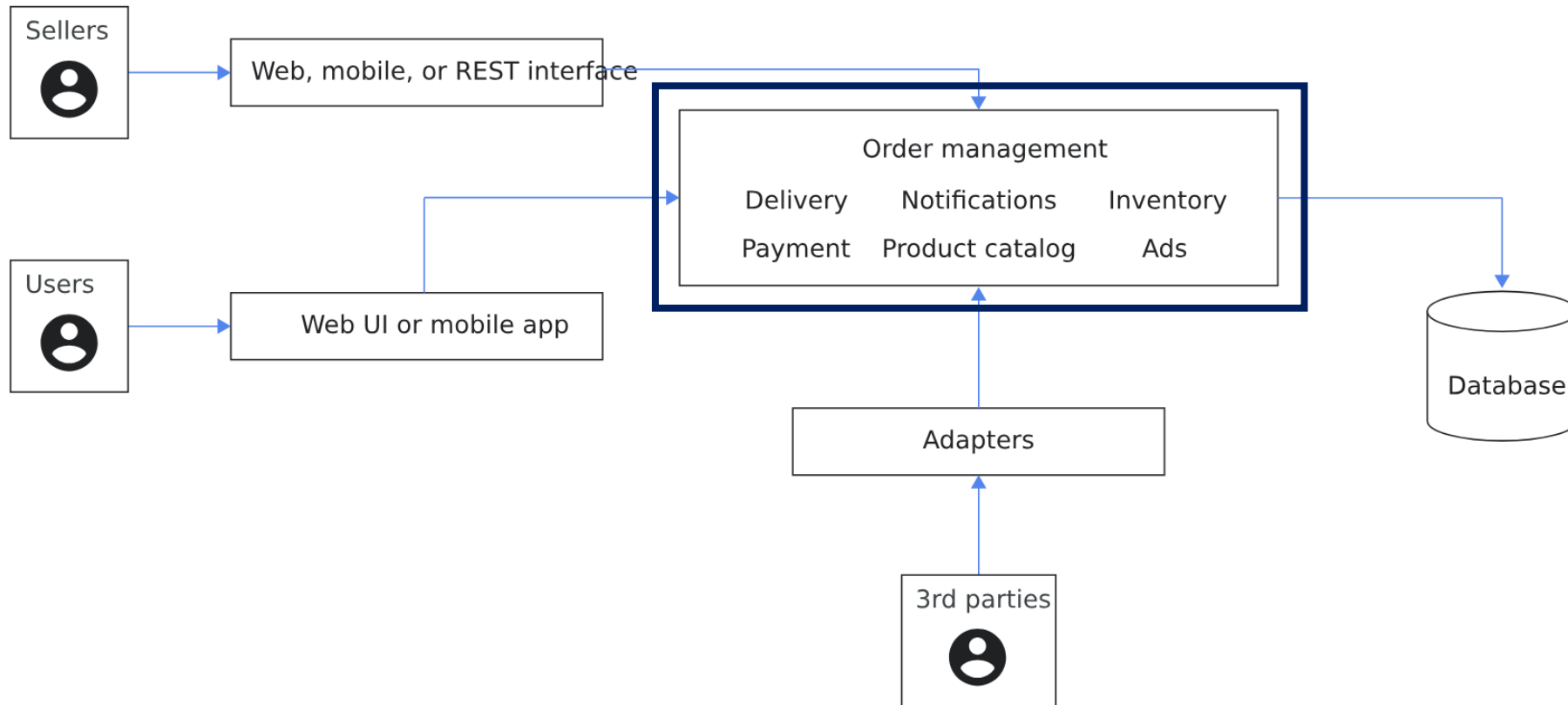


Diagram of a monolithic ecommerce application with several modules using a combination of programming language constructs. (<https://cloud.google.com/architecture/microservices-architecture-introduction>)

- **Benefits of Monolithic Architecture**

- **Simplified Testing:** Tools like Selenium enable end-to-end testing of the entire application.
- **Ease of Deployment:** Deployment involves simply copying the packaged application to a server.
- **Resource Sharing:** All modules share memory, space, and resources, streamlining cross-cutting concerns like logging, caching, and security.
- **Intra-Process Communication:** Direct module-to-module calls can offer performance advantages over network-dependent microservices.

- **Challenges of Monolithic Architecture**

- **Scalability Issues:** Difficulty in scaling when different modules have conflicting resource requirements.
- **Complexity in Maintenance and Updates:** As the application grows, implementing changes becomes more complicated due to tightly coupled modules.
- **CI/CD Complications:** Continuous integration and deployment become challenging as any update requires redeploying the entire application.
- **Vulnerability to System Failures:** A bug in any module, like a memory leak, can crash the entire system.
- **Technological Rigidity:** Adopting new frameworks or languages is costly and time-consuming, as it often requires rewriting the entire application.

# Understanding Microservices

- **Core Characteristics**

- **Small and Focused:** Aimed at doing one thing well, avoiding sprawling codebases.
- **Cohesion and Single Responsibility:** Adhering to the principle of grouping related code and separating unrelated functionalities.

- **Size and Scope**

- **No Fixed Size:** Size varies based on language expressiveness and domain complexity.
- **Team Alignment:** Ideally sized to be managed by a small team.
- **Balance in Size:** Smaller services maximize benefits but increase complexity.

- **Autonomy**

- **Independent Entities:** Deployed separately, can be different technologies, possibly as isolated services on a PAAS or as individual operating system processes.
- **Network Communication:** Services communicate via network calls, ensuring separation and reducing tight coupling.

- **Deployment and Change Management**

- **Independent Deployment:** Services can be deployed independently without impacting others.
- **API-Centric Interaction:** Services expose APIs for interaction, emphasizing decoupled, technology-agnostic interfaces.

- **Decoupling**

- **Key to Microservices:** Essential for maintaining independence and achieving the benefits of microservices architecture.
- **Change and Deployment:** Ability to change and deploy a service independently is crucial.



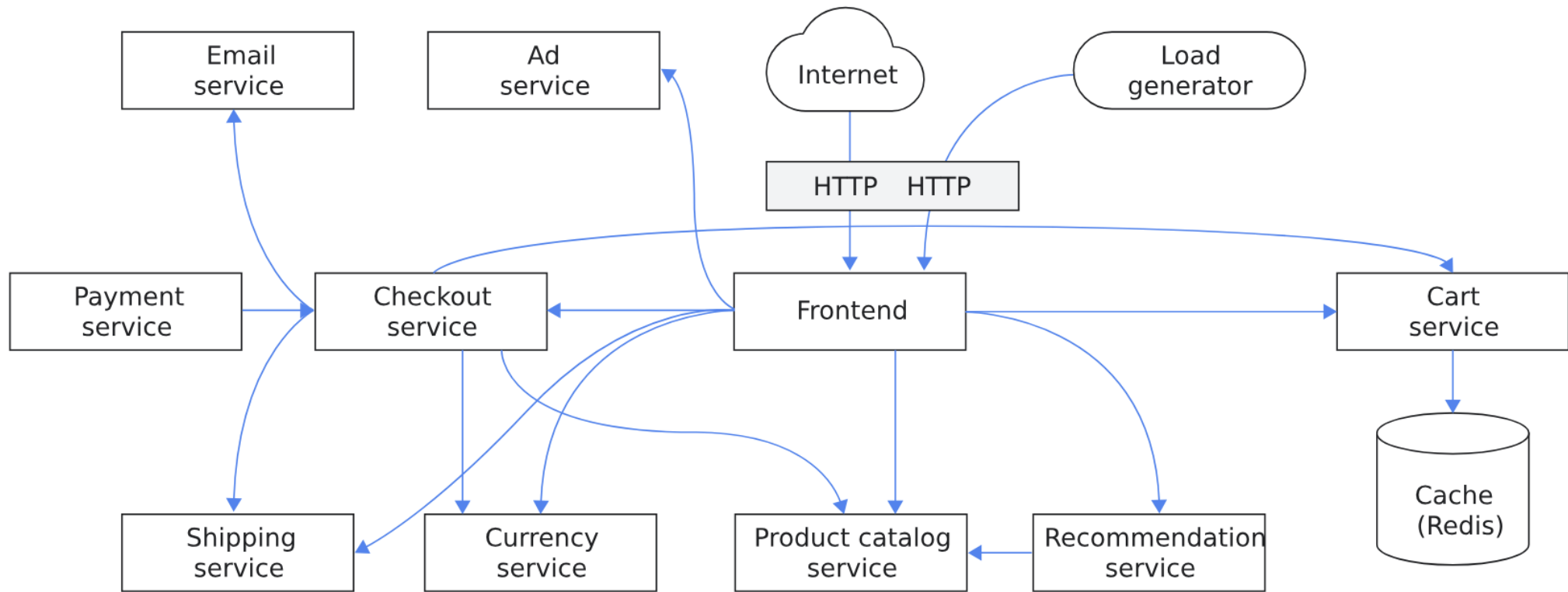
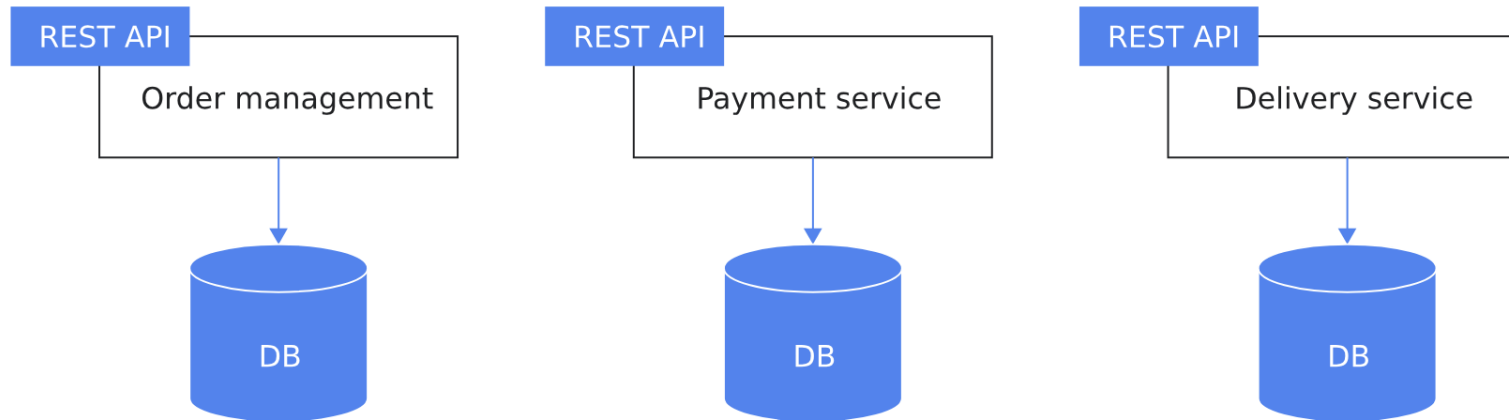


Diagram of an ecommerce application with functional areas implemented by microservices.  
(<https://cloud.google.com/architecture/microservices-architecture-introduction>)

- **Database Relationship**

- **Service-Specific Databases:** Each microservice has its own database tailored to its requirements.
- **Loose Coupling:** This approach ensures loose coupling by routing data requests through service APIs instead of a shared database.
- **Independent Data Management:** Each service manages its data independently, enhancing autonomy and reducing interdependencies.



# Benefits of Microservices Architecture

Aspect	Benefit Details
<b>Enhanced Development and Maintenance</b>	<ul style="list-style-type: none"><li>- Breaks application into smaller, manageable chunks.</li><li>- Clear boundaries with defined APIs.</li><li>- Quicker development, easier understanding and maintenance.</li></ul>
<b>Team Autonomy and Efficiency</b>	<ul style="list-style-type: none"><li>- Independent development of services by teams.</li><li>- Full lifecycle ownership of services.</li><li>- Flexibility to use different programming languages (Polyglot Development).</li></ul>
<b>Improved Scalability and Market Responsiveness</b>	<ul style="list-style-type: none"><li>- Independent scaling based on service needs.</li><li>- Hardware optimization for resource requirements.</li><li>- Faster product delivery and improved time to market.</li></ul>

# Challenges of Microservices Architecture

Challenge Category	Challenge Details
<b>Complexity in Distributed Systems</b>	<ul style="list-style-type: none"><li>- Necessity of choosing and implementing inter-service communication mechanisms.</li><li>- Managing partial failures and service unavailability.</li></ul>
<b>Transaction Management Across Services</b>	<ul style="list-style-type: none"><li>- Handling atomic operations across multiple microservices (Distributed Transactions).</li><li>- Maintaining data consistency during failures (Consistency Issues).</li></ul>
<b>Testing and Deployment Complexities</b>	<ul style="list-style-type: none"><li>- Requirement for comprehensive testing across multiple services.</li><li>- Complexities in managing multiple service deployments and service discovery.</li></ul>
<b>Operational Overhead</b>	<ul style="list-style-type: none"><li>- Increased need for monitoring and alerting across more services.</li><li>- Higher risk of failure due to more points of service-to-service communication.</li><li>- Challenges in productionizing and maintaining robust operations infrastructure.</li></ul>
<b>Performance and Suitability Considerations</b>	<ul style="list-style-type: none"><li>- Potential latency issues due to network calls between services.</li><li>- Not suitable for all types of applications, especially those requiring real-time data processing.</li><li>- Importance of clear communication and service boundary planning.</li></ul>

# Migrating from Monolithic to Microservices: Key Considerations

Consideration Category	Details
Assessing the Need for Migration	<ul style="list-style-type: none"><li>- Evaluate if microservices align with business goals and pain points.</li><li>- Consider simpler alternatives like autoscaling or enhanced testing.</li></ul>
Starting the Migration Process	<ul style="list-style-type: none"><li>- Begin with extracting and deploying one service independently.</li><li>- Adopt an iterative approach, learning and adapting with each service migration.</li></ul>
Strategic Implementation	<ul style="list-style-type: none"><li>- Recognize varying approaches to microservice size and quantity among teams.</li><li>- Emphasize continuous learning and strategy refinement.</li></ul>
Future Learning and Strategies	<ul style="list-style-type: none"><li>- Explore strategies for detailed refactoring from monolithic to microservices.</li><li>- Plan for ongoing education and adaptation of methods.</li></ul>

# References

- <https://cloud.google.com/architecture/microservices-architecture-introduction>
- Building Microservices, Sam Newman