













Microservices using ASP.NET Core Web API

 **New Batch:** 8th July 2025, 6:30 AM to 8:00 AM IST
 **Duration:** 3 Months (Monday to Friday)
 **Course Fees:** ₹10,000 INR or \$125 USD
 **Registration Link:** <https://forms.gle/7yiT3tZhrhMbWDTE9>
 **Join our Telegram Group:** <https://telegram.me/microservicesusingaspnetcore>
 **Mobile / WhatsApp Number:** +91 70218 01173
 **Course Details:** <https://dotnettutorials.net/lesson/microservices-using-asp-net-core-online-training/>

Demo Sessions:

 8th, and 9th July 2025
 6:30 AM to 8:30 AM IST

Demo Zoom Credentials:

 **Zoom Link:** <https://us06web.zoom.us/j/89542142270?pwd=02WUwBgCdlaKJ9r4N30l4uTOdtWol3.1>
 **Meeting ID:** 895 4214 2270
 **Passcode:** 284676

Module 1: Foundations & Fundamentals

This module builds a strong foundation by introducing you to the core concepts and principles of microservices. You will explore the evolution from monolithic to microservices architectures, understand key design principles such as loose coupling and decentralized data management, and learn about Clean Architecture and Domain-Driven Design (DDD). These fundamentals will prepare you to design scalable and maintainable microservice systems using ASP.NET Core Web API.

Chapter 1: Introduction to Microservices Architecture

In this chapter, you will learn the fundamental concepts of microservices and how they differ from monolithic systems. You will learn about the characteristics, advantages, and challenges of microservices, along with real-world examples like Netflix, Amazon, and Uber. By the end, you will understand when and why to transition from a monolithic to a microservices architecture, as well as how to identify the right use cases for microservices.

Topics to be Covered:

- What applications are and how they are structured into different layers.
- Types of applications (web, mobile, desktop) and their real-world examples.
- Monolithic architecture, its layered structure, benefits, and challenges, illustrated with an e-commerce example.
- Microservices architecture, how it works, its advantages, and its challenges, again explained with practical e-commerce scenarios.
- Real-world case studies from companies like Netflix, Amazon, and Uber, showing why organizations transition from monolithic systems to microservices.

Chapter 2: Core Microservices Design Principles

In this chapter, you will learn the foundational principles of microservices that make Microservices architecture successful, such as single responsibility, independent deployment, decentralized data management, event-driven communication, and scalability. These principles form the backbone of robust and flexible microservices.

Topics to be Covered:

- Single Responsibility Principle (SRP)
- Independent Deployment
- Decentralized Data Management
- Loose Coupling and High Cohesion
- Event-Driven Communication

- Fault Isolation and Scalability
- Technology Diversity

Chapter 3: Clean Architecture

In this chapter, you will learn about the Clean Architecture principles, which emphasize the separation of concerns and maintainability. You will learn how to structure microservices using layers (Presentation, Application, Domain, and Infrastructure) and see a real-world product management example implemented in ASP.NET Core Web API.

Topics to be Covered:

- What Is Clean Architecture?
- Why is Clean Architecture Important?
- Understanding the Layers of Clean Architecture
- Real-time Example: Product Management System using Clean Architecture in ASP.NET Core Web API.
- Benefits of Clean Architecture in ASP.NET Core Web API

Chapter 4: Domain-Driven Design (DDD)

In this chapter, you will learn about Domain-Driven Design (DDD), which is a software development approach that emphasizes the importance of the business domain in application design. It focuses on creating a shared understanding of the domain between developers and domain experts, encapsulating complex business rules in rich domain models, and organizing code around domain concepts.

Topics to be Covered:

- What Is Domain-Driven Design (DDD)?
- Why is Domain-Driven Design Important?
- Understanding the Layers of Domain-Driven Design
- Real-time Example using Domain-Driven Design in ASP.NET Core Web API
- Benefits of Domain-Driven Design in ASP.NET Core Web API

Module 2: Building Core Microservices

Transitioning from theory to practice, this module guides you through the hands-on development of essential microservices for an e-commerce platform. We will set up the project structure in Visual Studio and implement fully functional User, Product, Order, Payment, and Notification microservices using Entity Framework Core and SQL Server. Emphasis is placed on creating secure, robust, and independent services that collectively form a scalable solution.

Chapter 5: Microservices Project Setup in Visual Studio

In this chapter, you will learn how to set up a scalable microservices solution in Visual Studio for an e-commerce platform. This chapter guides you through creating a well-structured Visual Studio solution that follows **Clean Architecture and Domain-Driven Design (DDD)** principles, preparing it for independent microservice development and deployment.

Topics to be Covered:

- Overview of microservices in modern e-commerce systems
- Key microservices in an e-commerce platform: User, Product, Order, Payment, Notification
- Importance of modularization, separation of concerns, and independent deployment
- Comparing development approaches: Monorepo (single solution) vs. Polyrepo (multiple solutions)
- Industry best practice: One solution/repository per microservice with Clean Architecture and DDD
- Course approach: Starting with a single Visual Studio solution containing all microservices projects
- Detailed steps for creating the root Visual Studio solution
- Creating solution folders for each microservice
- Adding Domain, Application, Infrastructure, API, and Tests projects per microservice
- Project referencing and dependencies setup
- Naming conventions and folder structure

Chapter 6: Creating the User Microservice

In this chapter, we will focus on managing user authentication, registration, and profiles with JWT security. You will learn about JWT tokens, role-based authorization, and secure profile management. By the end of the chapter, we will implement the User Service with authentication, login, and profile management capabilities.

Topics to be Covered:

- **User Registration & Authentication:** Support user signup, login, password reset, and multi-factor authentication (MFA).
- **User Profile Management:** CRUD operations for user profiles including name, email, address, phone number.
- **Role & Permission Management:** Support roles like Admin, Customer, Vendor, with role-based authorization.
- **Address Book:** Manage multiple shipping/billing addresses per user.
- **Session Management & Security:** JWT or OAuth2 token-based authentication, refresh tokens, secure password hashing.
- **User Activity Tracking:** Track user logins, last seen, and other security events.
- **Account Verification:** Email verification and phone number verification via OTP.
- **Social Login Integration:** Support for Google, Facebook, or other OAuth providers.
- **API for User Data:** Secure APIs for other microservices (e.g., Order Microservice) to retrieve user details when needed.

Chapter 7: Building the Product Microservice

In this chapter, we will develop a Product Microservice using ASP.NET Core Web API. You will learn about the Product domain, how to manage relationships like Product-Category, and the importance of data validation using FluentValidation and Data Annotation. You will also get hands-on experience in implementing CRUD operations using EF Core Code First Approach with SQL Server Database.

Topics to be Covered:

- **Product Management:** CRUD operations for products (add, update, delete, view).
- **Category & Subcategory Management:** Organize products into categories for easier browsing.
- **Inventory Management:** Track product stock levels, availability status, and automatic stock updates.
- **Product Images:** Upload, store, and serve product images.
- **Search & Filtering:** Enable search by product name, category, price, rating, and other filters.
- **Price & Discount Management:** Manage product pricing, discounts, and promotional offers.
- **Product Reviews & Ratings:** Manage customer reviews and star ratings.
- **API Endpoints:** Public APIs to expose product data for other microservices or frontend consumption.

Chapter 8: Building the Order Microservice

In this chapter, we will develop the Order Microservice, focusing on managing customer orders, items, and order statuses. The chapter will cover repository patterns, Entity Framework Core for data persistence, and how to handle relationships between entities. We will implement key features like order placement, tracking, status update, cancellation, etc.

Topics to be Covered:

- **Order Placement:** Handle creation of new orders with validation of cart items, quantities, and user details.
- **Order Tracking:** Maintain order status lifecycle (Placed, Confirmed, Packed, Shipped, Delivered, Cancelled, Returned).
- **Order History:** Allow users to view their past orders.
- **Order Validation:** Verify product availability and prices before order confirmation.
- **Cart Management:** Maintain user's current shopping cart and support CRUD for cart items.
- **Order Cancellation & Returns:** Support order cancellation before shipment and returns management.

- **Integration with Payment Microservice:** Update order payment status and trigger workflows accordingly.
- **Integration with Notification Microservice:** Send order confirmation, shipment, and delivery notifications.
- **Invoice Generation:** Generate and store order invoices.
- **API for Orders:** APIs for placing, updating, retrieving orders, and for internal communication with other microservices.

Chapter 9: Creating the Payment Microservice

In this chapter, we will build a Payment Microservice, simulating payment gateways such as cash, card, and UPI. You will learn about payment flows, handle payment status updates, and integrate the Payment service with the Order service. In this chapter, we will also cover how to secure payment endpoints using API keys and authentication.

Topics to be Covered:

- **Payment Gateway Integration:** Support multiple payment methods (Credit/Debit Cards, Net Banking, UPI, Wallets).
- **Payment Processing:** Handle payment authorization, capture, refund, and failure.
- **Transaction Management:** Store payment transaction details with statuses.
- **Secure Payment Data:** Ensure PCI compliance, encryption of sensitive data.
- **Refund Processing:** Handle partial and full refunds.
- **Payment Status Updates:** Notify Order Microservice of payment success/failure.
- **Retry Mechanism:** Retry failed payment attempts or offer alternative payment methods.
- **API Endpoints:** APIs for initiating payments, checking status, refunds, and reports.

Chapter 10: Creating the Notification Microservice

In this chapter, we will build a notification system that listens to events (like Order Placement, Payment Success, Order Delivered, Order Cancelled, etc.) and sends multi-channel notifications via Email and SMS. Here, we will implement templating, scheduling, retry logic, and user preference management to enhance communication reliability and user engagement.

Topics to be Covered:

- **Multi-Channel Notification:** Support notifications via Email and SMS and possibly in-app notifications.
- **Template Management:** Manage reusable templates for order confirmations, shipment updates, payment receipts, promotional messages.
- **Event-Driven Architecture:** Listen for events from other microservices (Order placed, Payment success, Shipment dispatched).
- **Notification Scheduling:** Schedule notifications for future delivery (e.g., reminders, promotions).
- **Retry & Failure Handling:** Retry failed notification attempts and log failures.
- **User Preferences:** Allow users to opt-in/out of notification types and channels.
- **API Endpoints:** APIs to send notifications, manage templates, and get notification history.

Module 3: Inter-Service Communication

Master the art of communication between distributed services. This module covers both synchronous (REST, gRPC) and asynchronous (RabbitMQ, Kafka) patterns, as well as advanced API management, service discovery, security, implementing rate limiting for protection, and caching strategies, empowering you to build robust, high-performance, and secure microservice interactions.

Chapter 11: Inter-Service Communication in Microservices

In this chapter, we will introduce the concepts of Inter-Service Communication in a Microservices architecture. You will learn about (**HTTP/REST**, **gRPC**) and asynchronous (**RabbitMQ**, **Kafka**) communication techniques, comparing HTTP REST, gRPC, and messaging.

Topics to be Covered:

- Need for inter-service communication in microservices
- Synchronous communication: HTTP REST APIs, gRPC overview, pros and cons

- Asynchronous communication: Messaging basics (RabbitMQ, Kafka) and event-driven architecture
- Designing APIs for internal vs. external consumption
- Basic error handling and HTTP status codes
- Timeouts and cancellation handling in service calls
- Best practices for service-to-service communication in ASP.NET Core Web API

Hands-on Lab:

- Implement RESTful HTTP communication between Product and Order microservices
- Integrate RabbitMQ for asynchronous messaging between Order and Notification microservices
- Test synchronous and asynchronous communication flows end-to-end with Postman and logs

Chapter 12: Data Management Strategies in Microservices

In this chapter, you will learn about managing distributed data in microservices architecture. You will understand the concepts of distributed transactions. You will also understand how to implement the Saga Pattern to handle distributed workflows across multiple services and ensure data consistency in microservices.

Topics to be Covered:

- Challenges of data management in microservices (distributed transactions, consistency)
- Why shared databases are an anti-pattern in microservices
- Database per microservice approach: schema isolation and independent scaling
- Data consistency challenges: eventual vs. strong consistency
- Patterns for data consistency overview:
 1. Saga Pattern (concept, choreography vs orchestration)
 2. Two-Phase Commit (2PC) overview and limitations
 3. Event Sourcing fundamentals
 4. Command Query Responsibility Segregation (CQRS) overview
- Handling distributed transactions and compensating actions
- Query patterns: API Composition and CQRS
- Real-time example: Handling distributed order and payment data

Hands-on Lab:

- Implement Saga pattern (conceptual) using messaging to coordinate distributed transactions
- Develop compensating actions for rollback scenarios
- Create sample API Composition endpoints aggregating data from User, Order, and Product microservices

Chapter 13: API Gateway and Service Discovery

In this chapter, you will learn the role of API Gateways for routing, authentication, and aggregation. Implement Ocelot in ASP.NET Core, set up service discovery with Eureka or Consul, and perform efficient communication between clients and multiple microservices.

Topics to be Covered:

- What is an API Gateway? Why is it important in microservices architecture?
- Common features: Routing, aggregation, authentication, rate limiting, load balancing.
- Popular API Gateway implementations (Ocelot, YARP, Kong) in ASP.NET Core ecosystem
- Setting up Ocelot API Gateway in ASP.NET Core.
- Service Discovery concepts: Client-side vs. Server-side discovery patterns.
- Service Registry & Discovery (Eureka, Consul, custom solutions): How microservices find each other.
- Edge vs. internal API Gateway scenarios.
- Securing microservices via the gateway.

Hands-on/Lab:

- Set up and configure Ocelot API Gateway to route requests to Product and Order microservices based on paths and headers.
- Implement JWT-based centralized authentication at the API Gateway level.

- Register microservices with a service registry like Consul or Eureka and configure service discovery in API Gateway.
- Create simple request aggregation APIs at the gateway that combine data from multiple services.
- Test gateway routing, authentication, and aggregation flows using Postman.

Chapter 14: Rate Limiting / Throttling

In this chapter, you will learn how to protect your microservices from overload and abuse by implementing rate limiting and throttling using different algorithms and layers, including API Gateway and middleware strategies, while maintaining good user experience.

Topics to be Covered:

- Why rate limiting and throttling are important in microservices
- Common scenarios: load spikes, denial of service (DoS) prevention, fair usage policies
- Types of rate limiting: fixed window, sliding window, token bucket, leaky bucket algorithms
- Rate limiting at different layers: API Gateway, microservice middleware, client-side
- Implementing rate limiting with Ocelot API Gateway in ASP.NET Core
- Using middleware for throttling inside microservices
- Configuring limits by IP, user, API key, or route
- Handling rate limit exceeded responses (HTTP 429) and client notifications
- Monitoring and logging rate limit events
- Best practices to balance protection and user experience

Hands-on Lab:

- Configure Ocelot to enforce rate limiting policies for APIs
- Implement middleware-based throttling in ASP.NET Core microservices
- Test rate limiting under simulated load scenarios using Postman or load testing tools

Chapter 15: Security in Microservices

In this chapter, you will learn how to master security essentials in microservices including JWT authentication, OAuth2, RBAC (Role-Based Access Control), secure inter-service communication, secrets management, and protecting APIs behind gateways to build a robust and secure microservice ecosystem.

Topics to be Covered:

- Security challenges in microservices architecture
- Authentication vs Authorization
- Authentication strategies:
 1. JWT token-based authentication (with IdentityServer4 or ASP.NET Core Identity)
 2. OAuth2 and OpenID Connect basics
- Authorization approaches: Role-based (RBAC), Policy-based, Claims-based.
- Secure inter-service communication: API keys, mutual TLS, token propagation.
- Protecting APIs behind the API Gateway.
- Handling secrets: Configuration management, Key Vaults, environment variables.
- Best practices: Rate limiting, IP filtering, logging, auditing.

Hands-on/Lab:

- Implement JWT authentication for User microservice and secure other microservices with token validation.
- Propagate JWT tokens through API Gateway to backend services.
- Set up role-based authorization (RBAC) for Admin, Customer, and Vendor roles.
- Secure inter-service communication using API keys or mutual TLS.
- Use HTTPS for all inter-service calls.
- Test authentication and authorization flows with Postman, including negative test cases.

Chapter 16: Performance Optimization with Caching

In this chapter, you will learn the importance of caching in microservices to optimize performance and reduce redundant requests. We will cover Redis as a distributed caching solution, explaining how to implement caching, manage TTL (time-to-live), and invalidate cache when necessary.

Topics to be Covered:

- Why caching is essential in microservices.
- Caching strategies: Client-side, server-side, distributed caching.
- Cache invalidation patterns and challenges.
- Setting up distributed cache with Redis in ASP.NET Core.
- Caching at API Gateway vs. microservice level.
- Cache aside, read-through, and write-through strategies.
- Avoiding common pitfalls: Stale data, cache stampede, consistency.
- Real-world examples: Caching product catalog, user sessions, token validation.

Hands-on/Lab:

- Integrate Redis distributed cache with the Product microservice for frequently accessed data such as product listings.
- Implement response caching at the API Gateway (Ocelot) for GET endpoints to reduce load on microservices.
- Design and implement cache expiration policies and TTL (time-to-live).
- Handle cache invalidation scenarios to prevent stale data.
- Measure and compare API response times with and without caching.

Module 4: Advanced Communication, Messaging & Integration

This advanced module covers advanced communication and integration techniques for microservices. You will explore high-performance gRPC communication, event-driven architectures with messaging platforms, advanced messaging using MassTransit, and sophisticated data handling through CQRS and Event Sourcing. GraphQL integration will also be covered to build flexible and efficient APIs that aggregate data from multiple services.

Chapter 17: High-Performance Service Communication with gRPC

In this chapter, you will learn how to use gRPC for high-performance communication between services. You will learn about gRPC message structures, serialization with Protobuf, securing gRPC with TLS, and the differences between gRPC and REST for various microservice interaction scenarios.

Topics to be Covered:

- Introduction to Remote Procedure Call (RPC) and why gRPC for Microservices
- Overview of gRPC and Protocol Buffers
- Defining Service Contracts Using .proto Files
- Advantages of gRPC: Performance, contract-first design, streaming capabilities, and use cases
- Setting Up and Implementing gRPC Services in ASP.NET Core
- Types of gRPC Calls: Unary, client streaming, server streaming, and bidirectional streaming (including synchronous and asynchronous streaming)
- Interoperability with Other Languages/Platforms
- Versioning and Backward Compatibility in gRPC Contracts
- Securing gRPC Communication: TLS and authentication
- gRPC vs. REST: Performance, payload size, streaming, and when to use each

Hands-on/Lab:

- Create a simple gRPC service and client in ASP.NET Core for microservice communication.
- Implement unary and streaming RPC calls (client streaming, server streaming, bidirectional streaming) between Order and Inventory microservices.
- Compare the performance of gRPC calls against REST APIs for common microservice interactions.
- Secure gRPC services using TLS and implement authentication.

Chapter 18: Building Event-Driven Microservices Architecture

In this chapter, you learn how to design scalable, responsive systems using event-driven architecture. You will understand how to implement event buses, asynchronous event publishing, and idempotent event handlers using RabbitMQ and Kafka.

Topics to be Covered:

- Principles of Event-Driven Architecture (EDA)
- Benefits of EDA: Scalability, responsiveness, resilience
- Core Concepts: Events, producers, consumers, topics, and queues
- Event Brokers and Messaging Platforms Overview: RabbitMQ, Apache Kafka
- Designing Events and Event Schemas
- Handling Event Versioning and Schema Evolution
- Event Processing Patterns: Competing consumers, publish-subscribe, and basics of event sourcing
- Asynchronous Processing, Event Publishing, and Subscription
- Implementing Event-Driven Communication in ASP.NET Core Microservices
- Eventual Consistency and Designing Idempotent Event Handlers
- Error Handling and Dead-Letter Queues
- Use Cases: Order placed, payment received, stock updated, notification sent

Hands-on/Lab:

- Implement an OrderPlaced event in the Order microservice and publish it to RabbitMQ.
- Create consumers in Notification and Inventory microservices to process the event asynchronously.
- Design idempotent event handlers and handle potential duplicates.
- Simulate event replay and test event-driven workflows end-to-end.

Chapter 19: Apache Kafka for Microservices

In this chapter, you will learn how to use Apache Kafka as a distributed streaming platform to implement scalable, fault-tolerant, and real-time event-driven microservices. The chapter covers Kafka architecture, key concepts, and how to integrate Kafka with ASP.NET Core Web API microservices.

Topics to be Covered:

- Introduction to Apache Kafka: What and Why?
- Kafka Architecture: Topics, Partitions, Brokers, Producers, Consumers, and Consumer Groups
- Kafka vs Traditional Messaging Queues (RabbitMQ, MSMQ)
- Setting up Kafka locally and on cloud
- Kafka Producer and Consumer APIs in .NET
- Kafka message serialization: Avro, JSON, Protobuf
- Managing Kafka topics and partitions
- Implementing event-driven communication with Kafka in microservices
- Kafka Consumer Groups and Offset management
- Fault tolerance and message durability
- Kafka Streams and KSQL overview
- Monitoring and managing Kafka clusters
- Best practices for Kafka integration in ASP.NET Core Microservices

Hands-on Lab:

- Setup Apache Kafka locally or use a managed Kafka service.
- Create a Producer microservice that publishes events (e.g., ProductAdded).
- Create a Consumer microservice that subscribes to Kafka topics and processes events.
- Handle message serialization and deserialization.
- Implement retry and error handling strategies for Kafka consumers.
- Monitor Kafka message flow using Kafka Manager or equivalent tools.

Chapter 20: Advanced Messaging with MassTransit

In this chapter, you will learn about advanced messaging techniques using MassTransit, focusing on message reliability, retries, and fault handling. You will understand how to implement message routing,

error handling, and how MassTransit ensures that messages are reliably delivered in production systems.

Topics to be Covered:

- Introduction to MassTransit and its advantages over raw message brokers
- Overview of MassTransit framework and key features
- Supported transports (RabbitMQ, Azure Service Bus, etc.)
- Setting up MassTransit in ASP.NET Core microservices
- Key concepts: messages, consumers, Sagas, state machines, message contracts
- Implementing Sagas for long-running workflows and distributed transactions (e.g., Order fulfillment, Payment, Inventory)
- Message routing, retries, and fault handling
- Scheduled messages and delayed delivery
- Integrating with ASP.NET Core Dependency Injection and configuration
- Monitoring, logging, and diagnostics/tracing in MassTransit workflows

Hands-on Lab:

- Set up MassTransit with RabbitMQ in ASP.NET Core microservices
- Build a Saga workflow involving Order, Payment, and Inventory microservices
- Implement request-response and publish-subscribe messaging patterns
- Use MassTransit monitoring and diagnostics tools to track message flow and troubleshoot issues

Chapter 21: Implementing CQRS and Event Sourcing

In this chapter, you will learn how to separate command and query responsibilities using CQRS. This chapter will introduce to command handlers, query handlers, and how to structure their services for better scalability and maintainability.

Topics to be Covered:

- Introduction to CQRS: rationale, benefits, and challenges
- Separating commands (writes) and queries (reads)
- Roles of Command Handlers and Query Handlers
- Event Sourcing fundamentals: capturing all state changes as an immutable event log
- Benefits and challenges of Event Sourcing including auditability and complexity
- Implementing Event Stores in .NET (using EventStoreDB, MediatR, or custom solutions)
- Techniques for rebuilding application state from event logs and snapshots
- Designing read and write models for scalability and performance
- Handling eventual consistency and synchronization between models
- Best practices: event schema changes, event versioning, safe event replay
- Real-world use cases: order state transitions, audit logging, complex business workflows

Hands-on Lab:

- Implement CQRS in Order microservice by separating command and query models
- Add event sourcing to capture all state changes as events and rebuild state
- Use MediatR or EventStoreDB to manage event streams
- Handle eventual consistency between read and write models
- Test command handling, event persistence, and state rebuilding

Chapter 22: OData (Open Data Protocol)

In this chapter, you will learn about OData, a standardized RESTful protocol for querying and updating data using a uniform and metadata-driven approach. You will explore how OData supports rich querying capabilities like filtering, sorting, paging, and data shaping using URL conventions, making APIs more flexible and interoperable. This chapter also covers how to implement OData services and clients using ASP.NET Core Web API.

Topics to be Covered:

- What is OData? Overview and history
- OData architecture and metadata

- Querying data using OData URL conventions: filtering, sorting, pagination, and selecting fields
- CRUD operations through OData endpoints
- OData data formats: JSON and Atom/XML
- Benefits of OData for API design and client interoperability
- Implementing OData services with ASP.NET Core Web API
- Consuming OData APIs in clients (e.g., Blazor, Angular, Postman)
- OData vs. REST: similarities and differences
- Security considerations and best practices when using OData

Hands-on Lab:

- Create a simple ASP.NET Core Web API project exposing OData endpoints
- Implement filtering, sorting, and pagination on a sample Product entity using OData query options
- Test OData queries using tools like Postman or browser
- Secure OData endpoints using JWT authentication
- Consume the OData API from a sample client application

Chapter 23: GraphQL in Microservices

In this chapter, you will learn how to use GraphQL to build flexible and efficient APIs that can fetch data from multiple microservices. This chapter covers how GraphQL differs from REST APIs, the structure of GraphQL queries, mutations, and subscriptions, and how to integrate GraphQL into a microservices architecture for optimized data retrieval.

Topics to be Covered:

- Introduction to GraphQL and comparison with REST APIs
- Understanding GraphQL schema: queries, mutations, and subscriptions
- Benefits of using GraphQL in microservices architecture
- Designing GraphQL schemas for multiple microservices
- Querying and aggregating data from multiple services using GraphQL
- Implementing GraphQL server with ASP.NET Core (e.g., using HotChocolate or GraphQL.NET)
- Handling GraphQL subscriptions for real-time updates
- Security considerations in GraphQL APIs (authorization, query complexity)
- Performance optimization: batching, caching, and persisted queries
- Versioning and evolution of GraphQL schemas
- Integrating GraphQL with existing microservices and APIs
- Testing and debugging GraphQL endpoints

Hands-on Lab:

- Add a GraphQL server using libraries like HotChocolate or GraphQL.NET.
- Define GraphQL schema types (queries, mutations, subscriptions) that aggregate data from multiple microservices (e.g., Product, Order, User).
- Implement resolvers to fetch data from existing REST/gRPC microservices or databases.
- Create queries that support filtering, sorting, and pagination.
- Implement mutations for creating and updating resources through GraphQL.
- Add real-time subscriptions to receive updates (e.g., order status changes).
- Secure the GraphQL endpoint using authentication and authorization policies.
- Optimize performance with batching and caching techniques.
- Test GraphQL queries, mutations, and subscriptions using tools like GraphiQL or Postman.
- Document the GraphQL API and schema using introspection and playground features.

Module 5: Failure Handling, Observability, & Versioning

Reliability and maintainability are vital in distributed systems. This module focuses on designing fault-tolerant microservices using resilience patterns like retries and circuit breakers, implementing observability through centralized logging and distributed tracing, and managing API versioning to ensure backward compatibility. You will also learn practical tools and techniques to maintain uptime and diagnose issues in production environments.

Chapter 24: Failure Handling in Microservices

In this chapter, you will learn how to design resilient microservices by implementing retry patterns, circuit breakers, and fallback methods using Polly. You will also learn how to simulate and handle failures in microservices to ensure system robustness in production.

Topics to be Covered:

- Understanding types of failures in distributed microservices
- Transient vs permanent faults
- Retry patterns: immediate retry, exponential backoff, jitter
- Circuit Breaker pattern: concepts and use cases
- Timeout and Bulkhead isolation patterns
- Fallback strategies to ensure graceful degradation
- Implementing resilience policies with Polly in ASP.NET Core
- Combining multiple resilience policies (retry + circuit breaker + fallback)
- Testing resilience using Chaos Engineering principles
- Simulating failures and measuring system behaviour
- Best practices for designing fault-tolerant microservices

Hands-on Lab:

- Implement retry policies with exponential backoff and jitter using Polly
- Add circuit breaker and fallback policies for critical microservice calls
- Combine multiple Polly policies in service calls
- Use Chaos Engineering tools to inject faults and observe system behaviour
- Test system resilience and fallback scenarios

Chapter 25: Observability & Distributed Tracing

In this chapter, you will learn the concept of observability and how to implement distributed tracing in microservices using tools like Serilog, OpenTelemetry, and Jaeger. We will also explore metrics collection and dashboard creation with Grafana to monitor and track the performance of their microservices.

Topics to be Covered:

- What is Observability and why it matters in microservices
- Pillars of Observability: Logging, Metrics, and Tracing
- Centralized logging with Serilog and integration with ELK Stack
- Distributed tracing fundamentals and challenges in microservices
- Implementing OpenTelemetry instrumentation in ASP.NET Core
- Visualizing traces with Jaeger
- Collecting and monitoring metrics using Prometheus
- Creating dashboards and alerts in Grafana
- Correlating logs, traces, and metrics for root cause analysis
- Monitoring microservices health and performance in real-time
- Best practices for observability-driven development

Hands-on Lab:

- Integrate Serilog for centralized logging and configure structured log output.
- Instrument microservices with OpenTelemetry SDK to collect distributed traces.
- Visualize traces using Jaeger and correlate logs with trace IDs.
- Collect metrics using Prometheus and create Grafana dashboards.
- Set up alerts for service health and performance anomalies.

Chapter 26: Versioning and Backward Compatibility

In this chapter, you will learn how to manage API versioning in microservices, ensuring backward compatibility while supporting newer versions. We will explore strategies for deprecating old versions and maintaining compatibility across different versions of the same service.

Topics to be Covered:

- Importance of API versioning in microservices architecture

- API versioning strategies: URI versioning, query parameters, headers, content negotiation
- Designing backward-compatible APIs and handling breaking changes
- Deprecation policies and communicating changes to API consumers
- Implementing API versioning in ASP.NET Core Web API
- Versioning considerations for gRPC services and message contracts
- Managing multiple active versions in production environments
- Testing and validating API versions
- Automating documentation with Swagger/OpenAPI for versioned APIs
- Strategies for smooth client migration and minimizing downtime during upgrades

Hands-on Lab:

- Implement URI-based and header-based API versioning in ASP.NET Core Web API.
- Design backward-compatible API changes and version deprecation strategies.
- Configure Swagger/OpenAPI documentation to support multiple API versions.
- Version gRPC service contracts with protobuf file changes.
- Test multiple versions of the same API running side by side and client compatibility.

Module 6: Containerization & Load Balancing

In this final module, you will gain expertise in deploying and managing microservices at scale using containerization and orchestration technologies. You will learn Docker fundamentals, containerize microservices, and use Docker Compose for local multi-container setups. The module then advances to Kubernetes for production-grade orchestration, service discovery, and load balancing, and CI/CD automation for continuous delivery.

Chapter 27: Docker for Microservices

In this chapter, you will learn the basics of containerization with Docker. You will understand how to containerize microservices, write Dockerfiles, build Docker images, and use Docker Compose to run multi-container setups. This chapter lays the foundation for deploying microservices in isolated, reproducible environments.

Topics to be Covered:

- Introduction to containerization and its benefits over traditional virtualization
- Docker architecture: Docker Engine, Docker Hub, Docker CLI
- Understanding Docker images and containers
- Writing Dockerfiles for ASP.NET Core microservices
- Building and tagging Docker images
- Running containers locally and managing container lifecycle
- Networking basics in Docker containers
- Using Docker Compose for multi-container applications
- Defining services, networks, and volumes in docker-compose.yml
- Building a complete microservices solution with Docker Compose
- Best practices for Dockerfile creation and image optimization
- Debugging and troubleshooting Docker containers

Hands-on Lab:

- Write Dockerfiles for ASP.NET Core microservices and build Docker images.
- Run microservices as containers locally using docker run.
- Create a docker-compose.yml to orchestrate multi-container microservice environments.
- Manage networks, volumes, and environment variables in Docker Compose.
- Debug and troubleshoot containerized microservices.

Chapter 28: Kubernetes (K8s) for Microservices

In this chapter, you will learn how to use Kubernetes for managing and orchestrating containerized microservices. You will learn about Kubernetes architecture, deploying microservices on Kubernetes clusters, and managing services with Kubernetes using YAML manifests. You will also gain hands-on experience with basic Kubernetes commands.

Topics to be Covered:

- Overview of Kubernetes and container orchestration fundamentals
- Kubernetes architecture: Master node, worker nodes, API Server, Scheduler, Controller Manager
- Key Kubernetes objects: Pods, Deployments, ReplicaSets, Services, ConfigMaps, Secrets
- Deploying ASP.NET Core microservices to Kubernetes clusters
- Writing Kubernetes YAML manifests for deployments and services
- Managing deployments: rolling updates, rollbacks, and scaling
- Service discovery and load balancing in Kubernetes
- Using kubectl CLI for cluster management and troubleshooting
- Configuring persistent storage for microservices
- Namespaces and resource quotas for multi-tenant cluster management
- Introduction to Helm charts for Kubernetes application packaging
- Monitoring Kubernetes clusters with built-in tools

Hands-on Lab:

- Deploy microservices to a local Kubernetes cluster (e.g., Minikube or Docker Desktop).
- Write Kubernetes manifests (Deployment, Service, ConfigMap, Secret) for microservices.
- Use kubectl commands to manage pods, deployments, and services.
- Perform rolling updates, rollbacks, and horizontal pod autoscaling.
- Configure persistent storage and namespaces for multi-tenant management.
- Package microservices as Helm charts and deploy via Helm.

Chapter 29: Load Balancing

In this chapter, you will learn understand the importance of load balancing in microservices. The chapter will cover various load-balancing strategies to ensure that traffic is efficiently routed. Learn traffic routing strategies like blue-green and canary deployments.

Topics to be Covered:

- Importance of load balancing in microservices architecture
- Types of load balancing: client-side, server-side, and DNS-based
- Layer 4 (Transport) vs Layer 7 (Application) load balancing
- Load balancing algorithms: round-robin, least connections, IP hash, weighted load balancing
- Load balancing in Kubernetes using Services of type LoadBalancer and Ingress controllers
- Implementing load balancing with cloud providers (AWS ELB, Azure Load Balancer)
- Traffic routing strategies: blue-green deployment, canary releases, and A/B testing
- Health checks and failover mechanisms in load balancers
- Configuring load balancing for stateful vs stateless microservices
- Performance considerations and scaling with load balancing

Hands-on Lab:

- Configure Kubernetes Service of type LoadBalancer and Ingress controller to expose microservices.
- Implement different load balancing algorithms (round-robin, least connections) using Ingress rules or cloud load balancers.
- Set up health probes and readiness/liveness checks for pod monitoring.
- Simulate traffic routing strategies like blue-green deployments and canary releases.
- Monitor load balancer performance and failover behavior.

Chapter 30: Service Mesh in Microservices

In this chapter, you will learn Service Mesh architecture and tools like Istio and Linkerd. Implement advanced traffic management, security with mutual TLS, observability, and policy enforcement to manage microservices at scale.

Topics to be Covered:

- What is a Service Mesh and why it matters in microservices
- Core features of service meshes: traffic routing, load balancing, retries, circuit breaking
- Security features: mutual TLS (mTLS) for secure service-to-service communication

- Observability: metrics, distributed tracing, and logging integration
- Policy enforcement: access control, rate limiting, and quota management
- Popular Service Mesh tools overview: Istio, Linkerd, Consul Connect
- Architecture components: control plane and data plane
- Integrating a Service Mesh with Kubernetes-based microservices
- Use cases and benefits for microservice deployments
- Challenges and considerations when adopting Service Mesh

Hands-on Lab:

- Deploy a sample microservices application on Kubernetes with Istio or Linkerd
- Configure mTLS between services for secure communication
- Set up traffic routing rules and fault injection policies
- Collect and analyze telemetry data (metrics and traces) via service mesh dashboards

Chapter 31: CI/CD Pipelines for Microservices

In this chapter, you will learn how to automate the deployment process using CI/CD pipelines. This chapter covers the concepts of continuous integration and delivery, as well as how to set up pipelines using tools like GitHub Actions to automate the build, test, and deployment process for their microservices.

Topics to be Covered:

- Introduction to Continuous Integration (CI) and Continuous Deployment/Delivery (CD)
- Benefits of CI/CD in microservices environments
- Overview of CI/CD tools and platforms with focus on GitHub Actions
- Setting up GitHub repositories and workflows for ASP.NET Core microservices
- Automating build and test processes in CI pipelines
- Configuring multi-stage pipelines for build, test, and deployment
- Managing secrets and environment variables securely in CI/CD pipelines
- Deployment strategies: rolling updates, blue-green deployments, canary releases
- Integrating automated tests (unit, integration, end-to-end) into pipelines
- Monitoring and alerting on pipeline status and deployment health
- Troubleshooting common CI/CD pipeline issues
- Best practices for scaling CI/CD across multiple microservices

Hands-on Lab:

- Set up a GitHub repository for your ASP.NET Core microservices solution.
- Create a GitHub Actions workflow file (.github/workflows/ci-cd.yml) to automate the build process: restore dependencies, build the project, and run unit tests.
- Extend the workflow to include deployment stages for different environments (e.g., staging and production) using Docker containers or Kubernetes manifests.
- Configure secure storage and usage of secrets and environment variables in GitHub Actions.
- Automate integration and end-to-end tests as part of the pipeline.
- Implement multi-stage pipelines with approval gates for deployment.
- Monitor pipeline runs, troubleshoot failed jobs, and set up notifications.
- Test rolling deployments and rollback mechanisms through the pipeline.

Chapter 32: Testing Strategies for Microservices

In this chapter, you will understand the differences between unit, integration, and end-to-end testing in a distributed system and learn how to implement automated tests to ensure reliability and maintainability of microservices.

Topics to be Covered:

- Introduction to Microservices Testing Challenges
- Unit Testing
- Integration Testing
- End-to-End (E2E) Testing
- Test Data Management
- Performance and Load Testing

Hands-on Lab:

- Create unit tests for domain models and service layers using xUnit.
- Develop integration tests for API controllers using ASP.NET Core TestServer and in-memory databases.
- Write end-to-end API tests for an order placement workflow across multiple services using Postman.
- Test asynchronous message handling in the Notification microservice with MassTransit in-memory transport.

Chapter 33: .NET Aspire

This chapter/module will focus on the future-ready skills, advanced .NET features, and career pathways for microservices developers. It will empower learners to aspire for excellence in building scalable and maintainable microservices using the latest .NET ecosystem innovations.

Topics to be Covered:

- Overview of the latest .NET ecosystem updates relevant for microservices (e.g., .NET 8/9 features)
- Advanced C# features and performance optimizations for microservices
- Exploring cloud-native .NET capabilities: Azure Functions, Durable Functions, and serverless
- Building resilient microservices with advanced fault-tolerance and telemetry
- Emerging technologies: gRPC improvements, minimal APIs, Hot Reload, and source generators
- Best practices for scalable, maintainable, and secure microservices development
- Modern DevOps and GitOps workflows for .NET microservices
- Career pathways and certifications in .NET and cloud microservices architecture
- Community and ecosystem resources: open-source projects, blogs, conferences
- Tips for continuous learning and keeping up with .NET innovations

Hands-on Lab:

- Create a new microservices solution using .NET Aspire templates
- Add multiple services (API, Worker, Redis, etc.) and orchestrate locally
- Configure distributed tracing and health checks out of the box
- Secure secrets with local and cloud vaults
- Deploy the Aspire-managed solution to Azure Container Apps
- Monitor services using Aspire's built-in dashboard

Note: If we missed any topics, if any new features are introduced, or if you want to learn any concepts not available in this Microservices using ASP.NET Core Web API Syllabus, please let us know, and that will also be included in this course. If you have any questions, please contact us.