Lab Manual: Microservices in .NET

Overview

This lab manual accompanies the lecture "Understanding and Building Microservices in .NET." It guides students through hands-on exercises using code examples from the GitHub repository: https://github.com/horsdal/microservices-in-dotnetcore.

Learning Outcomes

By completing this lab, you will be able to:

- Build simple microservices using Nancy and OWIN
- Implement inter-service communication and resilience patterns
- Write unit tests using Nancy.Testing and xUnit
- Understand microservice data ownership
- Monitor and log service activities

Lab 1: Creating a Simple Microservice

Objective

Create a minimal microservice using Nancy and OWIN.

Steps

1. Clone the repository:
2. git clone https://github.com/horsdal/microservices-in-dotnetcore.git
3. cd microservices-in-dotnetcore/Chapter02
4. Open the solution in your IDE (e.g., Visual Studio or VS Code).
5. Examine the HomeModule.cs file:
6. public class HomeModule : NancyModule
7. {
8. public HomeModule()
9. {
10. Get("/", _ => "Hello from microservice!");
11. }
12. }
13. Run the application and verify you can access http://localhost:1234/.

Deliverables

- Screenshot of the running endpoint response.

- Brief explanation of how Nancy and OWIN interact.

Lab 2: Adding Inter-service Communication & Resilience

Objective

Extend the Shopping Cart microservice to fetch product info from ProductCatalog and handle potential failures.

Steps

1. Navigate to the ShoppingCart service.
2. Add an HTTP client to call the ProductCatalog service:
3. var productResponse = await httpClient.GetStringAsync("http://localhost:1235/api/products/42");
4. Implement Polly for retry:
5. var policy = Policy
6. .Handle<HttpRequestException>()
7. .WaitAndRetry(3, retryAttempt => TimeSpan.FromSeconds(2));
8. Add circuit breaker:
9. var breaker = Policy
10. .Handle<Exception>()
11. .CircuitBreaker(2, TimeSpan.FromMinutes(1));

Deliverables

- A working service that logs and recovers from service call failures.
- Code snippets showing retry and circuit breaker logic.

Lab 3: Data Ownership and Persistence

Objective

Implement local data storage in a microservice and enforce service boundaries.

Steps

1. Create a data model and in-memory store in the ShoppingCart service.
2. Store shopping cart items per user.
3. Avoid direct access to ProductCatalog's data — always use its API.

Deliverables

- Code for data model and store.

- Diagram showing service boundaries and communication flow.

Lab 4: Testing Microservices with Nancy.Testing and xUnit

Objective

Write and run unit tests for your service endpoints.

Steps

1. Create a new xUnit test project.
2. Install Nancy.Testing via NuGet.
3. Add a test for the Home endpoint:
4. [Fact]
5. public void Should_Return_Hello()
6. {
7. var browser = new Browser(with => with.Module<HomeModule>());
8. var result = browser.Get("/", with => with.HttpRequest());
9.
10. Assert.Equal(HttpStatusCode.OK, result.StatusCode);
11. }

Deliverables

- A passing unit test.
- Explanation of test pyramid and where this test fits.

Lab 5: Monitoring and Logging

Objective

Add logging and monitoring middleware to your microservice.

Steps

1. Install and configure Serilog:
2. Log.Logger = new LoggerConfiguration()
3. .WriteTo.Console()
4. .CreateLogger();
5. Add monitoring endpoint:
6. Get("/monitor", _ => "OK");

7. Log all incoming requests.

Deliverables

- Console output showing structured logs.
- Screenshot of /monitor endpoint response.

Final Exercise: Design a Microservice System

Objective

Plan and sketch a complete microservices system for the project that you are working on.

Tasks

1. Identify at least four microservices.
2. Define the API for each.
3. Determine which data each owns.
4. Plan communication patterns (sync/async).

Deliverables

- Architecture diagram
- Descriptions of each service and interaction
- API sample for at least one service

Submission

Please upload all code files, screenshots, and diagrams to week 8 under "Microservices Lab Submission."

Tips

- Use Postman to test endpoints.
- Check logs for errors.
- Start small and iterate incrementally.
- Collaborate with your project members on the final system design.