# C#/ASP.NET and Microservices for Beginners Syllabus

## 1. Introduction to C#:

* What is C#?
* C# is pronounced, "C-Sharp".
* It is an object-oriented programming language created by Microsoft that runs on the .NET Framework.
* C# has roots from the C family, and the language is close to other popular languages like C++ and Java.
* The first version was released in the year 2002. The latest version, C# 11, was released in November 2022.

C# is used for:

* Mobile applications
* Desktop applications
* Web applications
* Web services
* Web sites
* Games
* VR
* Database applications
* And much, much more!
* Environment setup (Visual Studio/VS Code).

This is a document about: [Get Started with C# (w3schools.com)](https://www.w3schools.com/cs/cs_getstarted.php)

|  |  |  |
| --- | --- | --- |
| **Milestones** | **Objective** | **User Story** |
| Setting up environments and starting the projects | Setting up necessary tools and starting empty projects for local development | As a software developer, I want to set up my development environment in Visual Studio/VS Code to efficiently work on a REST API project. This will allow me to write, test, and debug API endpoints effectively, ensuring that I can deliver high-quality code and functionality.  ----  Acceptance Criteria:  - C# IDE Install **-** C# Environment Install |

**Your task:** Following [**User Story**] above to install IDE and .NET on Visual Studio.

* Basic syntax, variables, and data types.

This is a Document about [C# Syntax (w3schools.com)](https://www.w3schools.com/cs/cs_syntax.php)

**Your task:** Completing exercises follow the link: [CSharpBasic\_Exercises.docx (sharepoint.com)](https://fptsoftware362-my.sharepoint.com/:w:/r/personal/thangdv29_fpt_com/_layouts/15/Doc.aspx?sourcedoc=%7BC2D14D1B-8529-4A90-9F7F-BE90E37C3CE3%7D&file=CSharpBasic_Exercises.docx&action=default&mobileredirect=true)

**Your task:** Completing exercises follow the link: [Exercise v3.0 (w3schools.com)](https://www.w3schools.com/cs/exercise.php?filename=exercise_syntax1)

## 2. OOP with C#:

* Classes and objects.

This is a document about [C# Classes and Objects (w3schools.com)](https://www.w3schools.com/cs/cs_classes.php)

This is a document about [C# Classes and Objects - Studytonight](https://www.studytonight.com/post/csharp-classes-and-objects)

* Inheritance, polymorphism, encapsulation, and abstraction.

This is a document about [C# OOP (Object-Oriented Programming) (w3schools.com)](https://www.w3schools.com/cs/cs_oop.php)

* Properties, indexers

This is a document about [C# Indexers and Properties - Studytonight](https://www.studytonight.com/post/csharp-indexers-and-properties)

**Your task:** Completing exercises about OOP follow the link: [OOP\_Exercises.docx (sharepoint.com)](https://fptsoftware362-my.sharepoint.com/:w:/r/personal/thangdv29_fpt_com/_layouts/15/Doc.aspx?sourcedoc=%7B064344A1-D1E4-4C64-A826-1FA1F8575EB5%7D&file=OOP_Exercises.docx&action=default&mobileredirect=true)

## 3. Advanced C#:

* Collections: Lists, Dictionaries, etc.

This is a document about [Collections in C# - GeeksforGeeks](https://www.geeksforgeeks.org/collections-in-c-sharp/)

**Your task:** Completing exercises about collections follow the link: [C# Professional - Collections - Exercises (codingame.com)](https://www.codingame.com/playgrounds/12961/c-professional---collections---exercises)

* LINQ basics.

This is a document about [Language Integrated Query (LINQ) in C# - C# | Microsoft Learn](https://learn.microsoft.com/en-us/dotnet/csharp/linq/)

This is a document about [C# LINQ - Studytonight](https://www.studytonight.com/post/csharp-linq)

**Your task:** Completing exercises about LINQ follow the link: [LINQ\_Exercises.docx (sharepoint.com)](https://fptsoftware362-my.sharepoint.com/personal/thangdv29_fpt_com/_layouts/15/Doc.aspx?sourcedoc=%7B33966910-ED0F-45B5-8C7E-9D35821044A7%7D&file=LINQ_Exercises.docx&action=default&mobileredirect=true)

* Exception handling.

This is a document about [Exception Handling - C# Programming Guide - C# | Microsoft Learn](https://learn.microsoft.com/en-us/dotnet/csharp/fundamentals/exceptions/exception-handling)

This is a document about [C# Exception Handling - Studytonight](https://www.studytonight.com/post/csharp-exception-handling)

**Your task:** Completing exercise about Exception follow the link: [Exception\_Exercises.docx (sharepoint.com)](https://fptsoftware362-my.sharepoint.com/:w:/r/personal/thangdv29_fpt_com/_layouts/15/Doc.aspx?sourcedoc=%7B55780EE4-71FD-4DC5-A214-EC1E2F449706%7D&file=Exception_Exercises.docx&action=default&mobileredirect=true)

* Delegates, lambdas, and events.

This is a document about [Delegates and lambdas - .NET | Microsoft Learn](https://learn.microsoft.com/en-us/dotnet/standard/delegates-lambdas)

This is a document about [Handling and Raising Events - .NET | Microsoft Learn](https://learn.microsoft.com/en-us/dotnet/standard/events/)

**Your task:** Completing exercise about Delegates, Lambdas, and events follow the link: [Delegates, lambdas, and events\_Exercises.docx (sharepoint.com)](https://fptsoftware362-my.sharepoint.com/:w:/r/personal/thangdv29_fpt_com/_layouts/15/Doc.aspx?sourcedoc=%7B39C3F033-7C48-490C-9676-FBFD73C10241%7D&file=Delegates%2C%20lambdas%2C%20and%20events_Exercises.docx&action=default&mobileredirect=true)

## 4. Introduction to Data Access with Entity Framework Core:

* What is Entity Framework Core?

This is a document about [Entity Framework Core Tutorials (entityframeworktutorial.net)](https://www.entityframeworktutorial.net/efcore/entity-framework-core.aspx)

This is a document about [Overview of Entity Framework Core - EF Core | Microsoft Learn](https://learn.microsoft.com/en-us/ef/core/)

* Code-first vs. Database-first approaches.
* Code-first: [What is Code-First? (entityframeworktutorial.net)](https://www.entityframeworktutorial.net/code-first/what-is-code-first.aspx)
* Database-first: [Entity Framework 6 (entityframeworktutorial.net)](https://www.entityframeworktutorial.net/entityframework6/introduction.aspx)

## 5. Basics of Web API with ASP.NET Core:

* What is ASP.NET Core?

This is a document about [Overview of ASP.NET Core | Microsoft Learn](https://learn.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-7.0)

Here are some key points about ASP.NET Core:

* **Cross-platform:** ASP.NET Core applications can be developed and run on Windows, macOS, and Linux.
* **Performance:** ASP.NET Core has been designed for high performance, and according to various benchmarks, it is one of the fastest web frameworks available. [ASP.NET Core performance | Microsoft Learn](https://learn.microsoft.com/en-us/aspnet/core/performance/overview?view=aspnetcore-7.0)
* **Modularity:** With the introduction of the .NET Core platform, the framework is now modular, which means you only include the parts of the framework that you need in your application. This leads to smaller, faster applications.
* **Integrated Dependency Injection**: ASP.NET Core has built-in support for Dependency Injection (DI) which helps in creating decoupled, testable, and maintainable applications. [Dependency injection in ASP.NET Core | Microsoft Learn](https://learn.microsoft.com/en-us/aspnet/core/fundamentals/dependency-injection?view=aspnetcore-7.0)
* **Kestrel Web Server:** ASP.NET Core introduces a new lightweight, high-performance, and cross-platform web server called Kestrel. It can be used as an edge server processing client requests or sit behind a reverse proxy like Nginx or IIS. [Web server implementations in ASP.NET Core | Microsoft Learn](https://learn.microsoft.com/en-us/aspnet/core/fundamentals/servers/?view=aspnetcore-7.0&tabs=windows)
* **Middleware:** In ASP.NET Core, you compose your request pipeline using Middleware components. This makes it easy to customize and configure how requests are processed. [ASP.NET Core Middleware | Microsoft Learn](https://learn.microsoft.com/en-us/aspnet/core/fundamentals/middleware/?view=aspnetcore-7.0)
* **Razor Pages:** Introduced in ASP.NET Core 2.0, Razor Pages is a new feature that provides a simpler way to organize and write page-focused web applications. It's an alternative to the traditional MVC pattern, though MVC is still fully supported and often used in ASP.NET Core. [Introduction to Razor Pages in ASP.NET Core | Microsoft Learn](https://learn.microsoft.com/en-us/aspnet/core/razor-pages/?view=aspnetcore-7.0&tabs=visual-studio)
* **Configuration:** The configuration system in ASP.NET Core is flexible, allowing for a variety of sources such as JSON files, environment variables, and command-line arguments. [Configuration in ASP.NET Core | Microsoft Learn](https://learn.microsoft.com/en-us/aspnet/core/fundamentals/configuration/?view=aspnetcore-7.0)
* **Built-in Support for SPA:** ASP.NET Core offers templates and integration for popular Single Page Application (SPA) frameworks like Angular, React, and Vue. [Overview of Single Page Apps (SPAs) in ASP.NET Core | Microsoft Learn](https://learn.microsoft.com/en-us/aspnet/core/client-side/spa/intro?view=aspnetcore-7.0)
* **Open Source:** ASP.NET Core is open-source, and its source code is available on GitHub. This promotes transparency and allows for community contributions.
* What is a Web API?

**ASP.NET Web API** is a framework for building HTTP services that can be accessed from different clients such as browsers, mobile devices, desktop applications, and IoTs.

Here are some key points about Web API in the context of ASP.NET:

* **HTTP Services**: Web API is used to create HTTP services, making it easy for developers to expose data and services over HTTP, allowing them to be consumed by a broad range of clients including web browsers, mobile devices, and other servers.
* **RESTful:** While Web API can be used to create both RESTful services and non-RESTful services, it is most commonly associated with creating RESTful APIs. This means that it uses standard HTTP methods like GET, POST, PUT, DELETE, etc., and can work with standard data formats like JSON and XML.
  + [REST API Introduction - GeeksforGeeks](https://www.geeksforgeeks.org/rest-api-introduction/)
  + [Tutorial: Create a web API with ASP.NET Core | Microsoft Learn](https://learn.microsoft.com/en-us/aspnet/core/tutorials/first-web-api?view=aspnetcore-7.0&tabs=visual-studio)
* **Convention Over Configuration**: Web API uses convention over configuration, which means it makes many decisions based on convention, so developers can focus on writing their application-specific code. [Convention over configuration - Wikipedia](https://en.wikipedia.org/wiki/Convention_over_configuration)
* **Content Negotiation**: One of the strong features of Web API is content negotiation, which allows a client to specify the format of the data they want (e.g., JSON, XML) and Web API will handle the conversion automatically. [Content Negotiation in ASP.NET Web API - ASP.NET 4.x | Microsoft Learn](https://learn.microsoft.com/en-us/aspnet/web-api/overview/formats-and-model-binding/content-negotiation)
* **Routing:** Web API uses routing to determine which controller and action should handle an incoming HTTP request. Developers can define custom routing patterns to handle different types of requests.
* [Routing in ASP.NET Core Web API - Dot Net Tutorials](https://dotnettutorials.net/lesson/routing-in-asp-net-core-web-api/)
* [Routing to controller actions in ASP.NET Core | Microsoft Learn](https://learn.microsoft.com/en-us/aspnet/core/mvc/controllers/routing?view=aspnetcore-7.0)
* **Integration with ASP.NET**: While Web API is a part of the ASP.NET framework, it can operate independently of other ASP.NET features. However, it can be seamlessly integrated with other ASP.NET features like MVC, Razor views, Identity for authentication, etc.
* **Cross-Platform:** With the advent of ASP.NET Core, the Web API capabilities are now cross-platform, allowing developers to create APIs that can run on Windows, Linux, and macOS.
* **Extensible:** Web API is designed to be extensible. Developers can add custom message handlers, formatters, filters, and more to customize the behavior of their API.
* **Self-Hosting**: While Web API services can be hosted in IIS (Internet Information Services), they can also be self-hosted in custom applications. This flexibility allows for a wide range of hosting scenarios.

It's worth noting that with the release of ASP.NET Core, the distinction between MVC and Web API has blurred, as both functionalities are now unified under a single programming model. So, in ASP.NET Core, you don't have a separate Web API framework; instead, you use the unified MVC framework to create both web applications and APIs.

## 6. Authentication & Authorization in Microservices:

* What is an Authentication & Authorization?

1. **Authentication:**

* **Purpose:** Determines who you are.
* **Process:** During authentication, a user or system proves their identity to the system. This is typically done by providing credentials such as a username and password, a fingerprint, a smart card, a token from a hardware or software-based authenticator, or some combination of these.
* **Outcome:** If authentication is successful, the system now knows who the user is but does not yet know what the user is allowed to do.

1. **Authorization:**

* **Purpose:** Determines what you are allowed to do.
* **Process:** Once a user is authenticated, the system then determines what resources the user can access and what operations they can perform on those resources. This is often based on predefined roles, permissions, or access controls associated with that user.
* **Outcome:** If authorization is successful, the user can access specific resources or perform certain operations. If not, the user's request is denied.

1. **Example:**

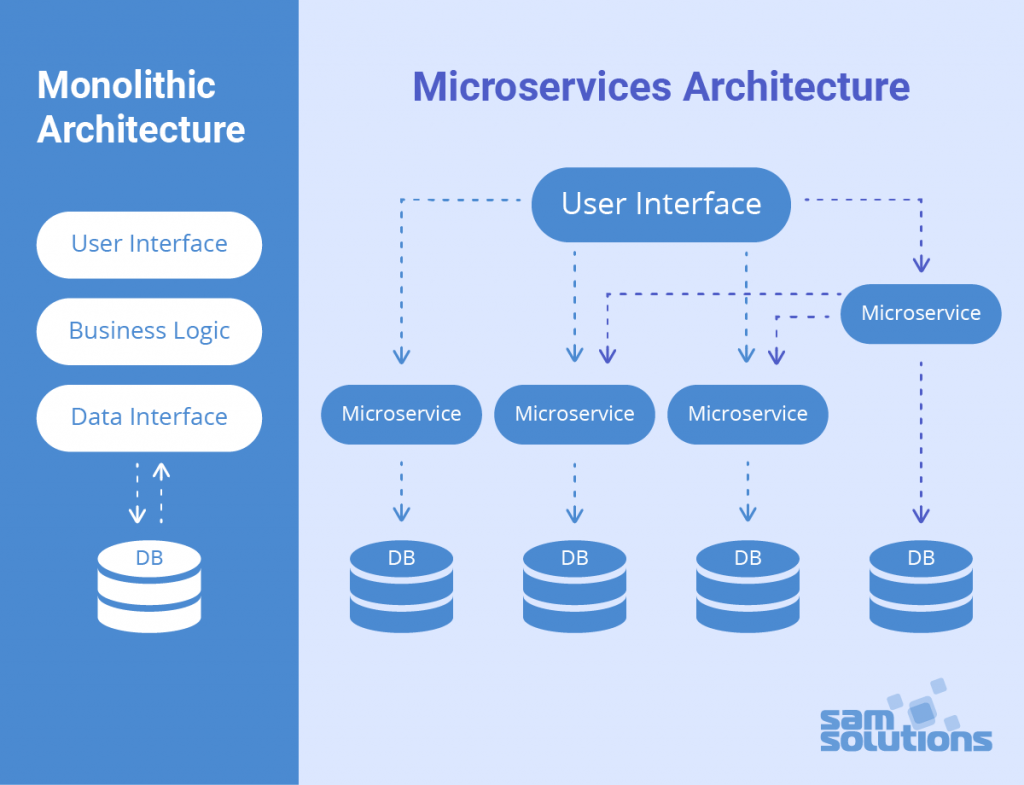
* **Authentication** is like the security desk checking your ID to see if you are indeed the person you claim to be.
* **Authorization** is like determining which rooms or areas of the building you're allowed to enter once your identity is confirmed.

This is a document about Authentication & Authorization in Microservices Architecture:

* [Authentication & Authorization in Microservices Architecture - Part I - DEV Community](https://dev.to/behalf/authentication-authorization-in-microservices-architecture-part-i-2cn0)
* JWT introduction (JSON Web Tokens)
  + What is JSON Web Token?
  + When should you use JSON Web Tokens?
  + What is the JSON Web Token structure?
  + How do JSON Web Tokens work?
  + Why should we use JSON Web Tokens?
* This is a document about JWT [JSON Web Token Introduction - jwt.io](https://jwt.io/introduction/)

## 8. Introduction to Microservices:

* Microservices architecture explained.



* **Microservices** are a software development approach where applications are composed of small, independent services that communicate over well-defined APIs. Microservices are distributed and loosely coupled, so they can be deployed and updated independently. Microservices are a cloud-native architecture that allows for faster and more flexible development and operations.
* Advantages and challenges.

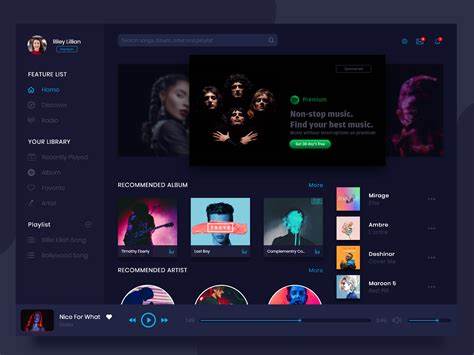
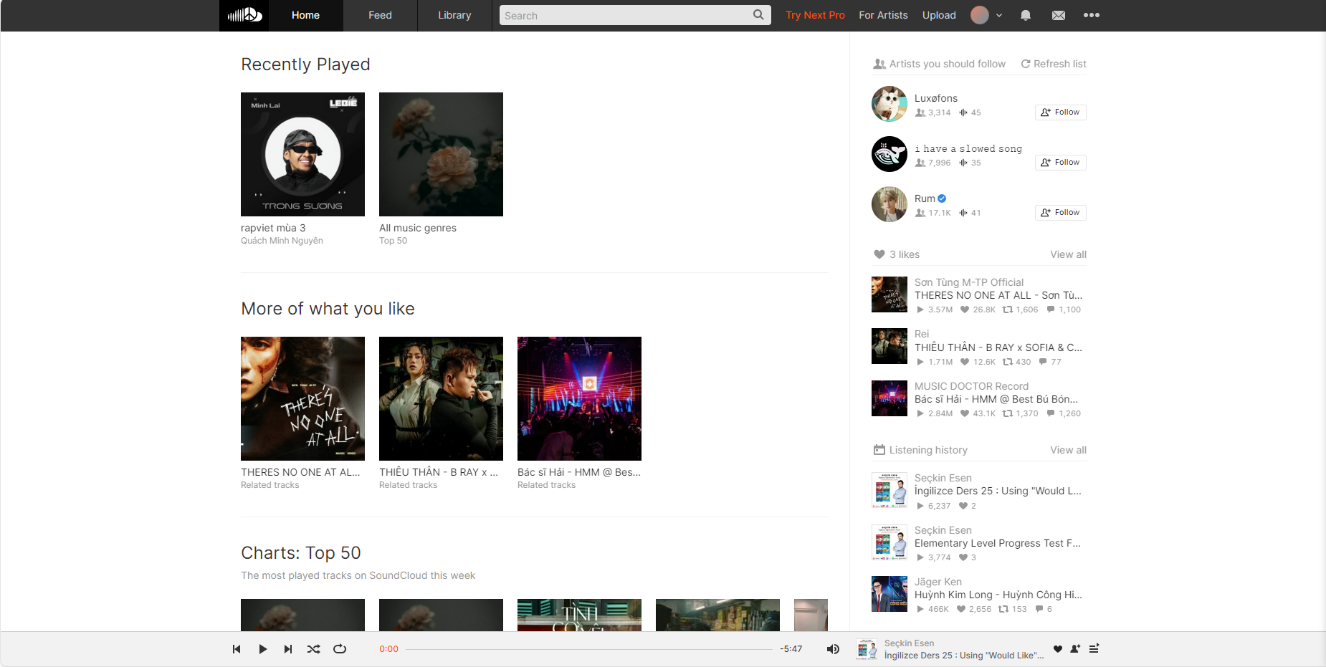
|  |  |  |
| --- | --- | --- |
| **#** | **Advantages of Microservices** | **Disadvantages of Microservices** |
| **1** | **Scalability:**  Advantages: Microservices can be individually scaled based on demand. This fine-grained scalability ensures efficient resource utilization.  **Example:** If one service within an e-commerce application experiences high traffic (e.g., product search), only that service can be scaled horizontally. | **Operational Complexity:**  Disadvantages: Managing multiple microservices can be operationally complex. Deployment, monitoring, and orchestration become more challenging.  **Example:** Coordinating the deployment of dozens of microservices in a production environment can be complex. |
| **2** | **Flexibility and Technology Diversity:**  Advantages: Each microservice can use the most appropriate technological stack for its specific task. This allows for innovation and the use of the best tools for each job.  **Example:** One microservice can use Java, while another uses Node.js, and they can communicate seamlessly. | **Distributed Systems Challenges:**  Disadvantages: Microservices involve network communication between services, which introduces latency and complexity. Handling distributed transactions can be challenging.  **Example:** Ensuring data consistency when an order service interacts with a payment service. |
| **3** | **Rapid Development:**  Advantages: Microservices enable parallel development by different teams, leading to faster feature development and time-to-market.  **Example:** One team can work on the payment service, while another team focuses on user authentication concurrently. | **Testing Complexity:**  Disadvantages: Testing can be more complex due to the distributed nature of microservices. End-to-end testing can be challenging.  **Example:** Ensuring that all services interact correctly in a complex workflow. |
| **4** | **Easy Maintenance:**  Advantages: Isolated microservices are easier to maintain and update. Changes to one service don't affect others, reducing the risk of introducing bugs.  **Example:** Updating the product catalog service does not require changes to the user authentication service. | **Resource Overhead:**  Disadvantages: Running multiple microservices consumes more resources than a single monolithic application, both in terms of memory and CPU.  **Example:** Running multiple instances of each microservice for redundancy. |
| **5** | **Resilience:**  Advantages: Isolated failures in one microservice do not necessarily affect the entire application. Properly designed microservices can be more fault-tolerant.  **Example:** If the recommendation service fails, other parts of the application can continue to function. | **Initial Development Overhead:**  Disadvantages: The initial setup and development of microservices may require more time and effort compared to starting with a monolith.  **Example:** Defining service boundaries and communication protocols. |
| **6** | **Scalable Development Teams:**  Advantages: Large projects can be divided into smaller, manageable teams, each responsible for one or more microservices.  **Example:** In a large e-commerce platform, different teams can manage the checkout, inventory, and user management services. | **Versioning Challenges:**  Disadvantages: Managing backward and forward compatibility between different versions of microservices can be complex.  **Example:** Ensuring that older clients can still use a newer version of a service without breaking. |
| **7** |  | **Cost and Infrastructure:**  Disadvantages: Microservices can incur higher infrastructure and operational costs due to the need for more resources and tools.  **Example:** Costs associated with maintaining and scaling multiple services. |

* Monolithic vs. Microservices.

|  |  |  |
| --- | --- | --- |
| **#** | **Monolithic Architecture** | **Microservices Architecture** |
| **1** | **Single Codebase:** In a monolithic architecture, the entire application is built as a single, tightly integrated codebase. | **Decomposed Services:** Microservices architecture decomposes the application into small, independently deployable services, each responsible for a specific piece of functionality. |
| **2** | **omponents:** The application consists of interconnected components, often organized into layers such as presentation, business logic, and data access. | **Components:** Each microservice is a standalone component that can have its own database, codebase, and technology stack. |
| **3** | **Development:** Easier to develop initially, as there is a single codebase to manage. | **Development:** Development is distributed across teams, and each team is responsible for one or more microservices. |
| **4** | **Deployment:** Deployment involves updating the entire application, which can be complex and risky. | **Deployment:** Easier deployment of individual microservices, allowing for faster updates and reduced risk. |
| **5** | **Scaling:** Scaling is typically done by replicating the entire application, which can be inefficient if some components require more resources than others. | **Scaling:** Each microservice can be scaled independently based on demand, resulting in efficient resource utilization. |
| **6** | **Maintenance:** Maintenance can become challenging as the application grows, and it may require significant effort to add new features or fix bugs. | **Maintenance:** Easier to maintain and update because changes to one microservice don't affect others. Adding new features or fixing issues is more straightforward. |
| **7** | **Technology Stack:** Uses a single technology stack and programming language throughout the application. | **Technology Stack:** Allows for flexibility in choosing the most appropriate technology stack for each microservice. |
| **8** | **Testing:** Testing can be simpler since all components are tightly coupled and share the same environment. | **Testing:** Testing can be more complex due to distributed nature, but each microservice can be tested in isolation. |

* Real-world examples of Microservices architecture
* **1. Netflix:** Netflix is a prime example of a company that has successfully implemented microservices architecture. Their architecture consists of numerous microservices that handle different functions, such as user authentication, recommendations, video streaming, and billing. This architecture allows them to scale and innovate rapidly.   
  
* **2. Amazon:** Amazon's e-commerce platform relies on microservices for various functionalities, including order processing, inventory management, and user authentication. This architecture enables them to handle high traffic during peak shopping seasons.

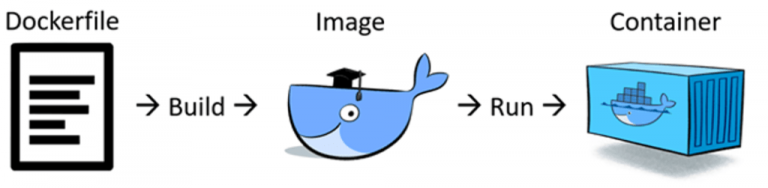


* **3. Spotify:** Spotify's music streaming service employs microservices to deliver personalized playlists, search functionality, user recommendations, and content delivery. This approach allows them to continuously enhance the user experience.
* **4. SoundCloud:** SoundCloud, the audio streaming platform, uses microservices to manage user accounts, content uploading, audio streaming, and social features. This architecture supports rapid feature development. 

## 9. Docker & Containerization:

This is a document about Docker [Overview of the get started guide | Docker Docs](https://docs.docker.com/get-started/)

* Introduction to Docker.
* Docker is a set of platforms as a service (PaaS) products that use the Operating system-level virtualization to deliver software in packages called containers. Containers are isolated from one another and bundle their own software, libraries, and configuration files; they can communicate with each other through well-defined channels. All containers are run by a single operating system kernel and therefore use fewer resources than a virtual machine.
* Developing, Shipping, and Running Applications
* Reduce time to production
* Run anywhere
* Container packages code and dependencies



* **Containers:** Containers are isolated environments that encapsulate an application and all its dependencies. They provide consistency across different environments (development, testing, production) and ensure that an application behaves the same way regardless of where it runs. Docker containers are based on a technology called containerization, with Docker being one of the most popular containerization platforms.
* **Images:** Docker images are read-only templates that serve as the basis for creating containers. An image contains everything needed to run an application, including the code, runtime, libraries, and environment variables. Images are typically defined by a Dockerfile, which is a script that specifies how an image should be built.
* **Dockerfile**: A Dockerfile is a text file that contains a set of instructions for building a Docker image. These instructions include specifying a base image, copying files into the image, installing software packages, and setting environment variables. Dockerfiles are used to create reproducible and automated image builds.

## 11. Microservices Communication:

* Synchronous vs. Asynchronous communication :
  + [Microservices patterns: synchronous vs asynchronous communication - greeeg.com](https://greeeg.com/en/issues/microservices-patterns-synchronous-vs-asynchronous)
  + [Communication in a microservice architecture - .NET | Microsoft Learn](https://learn.microsoft.com/en-us/dotnet/architecture/microservices/architect-microservice-container-applications/communication-in-microservice-architecture)
* RESTful communication between services: [Microservice Communications with HTTP REST Calls and .NET Core – andrewhalil.com](https://andrewhalil.com/2020/09/20/microservice-communication-using-http-rest-calls-in-net-core/)
* Message brokers communicate between services.

1. What is a Message broker?
2. Message broker architecture
3. Message Broker vs Message Queue
4. Message broker vs Event streaming Platform
5. Message broker vs Message Bus
6. Common Message brokers (e.g., RabbitMQ, Apache Kafka)

* This is a document about Message Broker: [What is a Message Broker? | VMware](https://www.vmware.com/topics/glossary/content/message-brokers.html#:~:text=A%20distributed%20message%20broker%20simplifies,in%20the%20form%20of%20messages.)
* This is a document how-to-use RabbitMQ in ASP.NET Core: [How to Use RabbitMQ in ASP.NET Core - Code Maze (code-maze.com)](https://code-maze.com/aspnetcore-rabbitmq/)

## 13. API Gateways in Microservices:

* What are an API Gateways in Microservices?
* Why use an API Gateway?
* How to implement API Gateways in Microservices?

This is a document about API Gateways in microservices:

* [API gateway pattern (microservices.io)](https://microservices.io/patterns/apigateway.html)
* [API Gateway Desing Pattern In Microservices (c-sharpcorner.com)](https://www.c-sharpcorner.com/article/microservices-design-using-gateway-pattern/)
* How to implement API Gateways with Ocelot in microservices architecture: [Implementing API Gateways with Ocelot - .NET | Microsoft Learn](https://learn.microsoft.com/en-us/dotnet/architecture/microservices/multi-container-microservice-net-applications/implement-api-gateways-with-ocelot)
* How to Implement JWT with Ocelot in microservices architecture: [Secure Microservices Using JWT With Ocelot in .NET Core (code-maze.com)](https://code-maze.com/dotnetcore-secure-microservices-jwt-ocelot/)