# Microservices in .NET Core

Trusted partner for your Digital Journey

Atos | Syntel

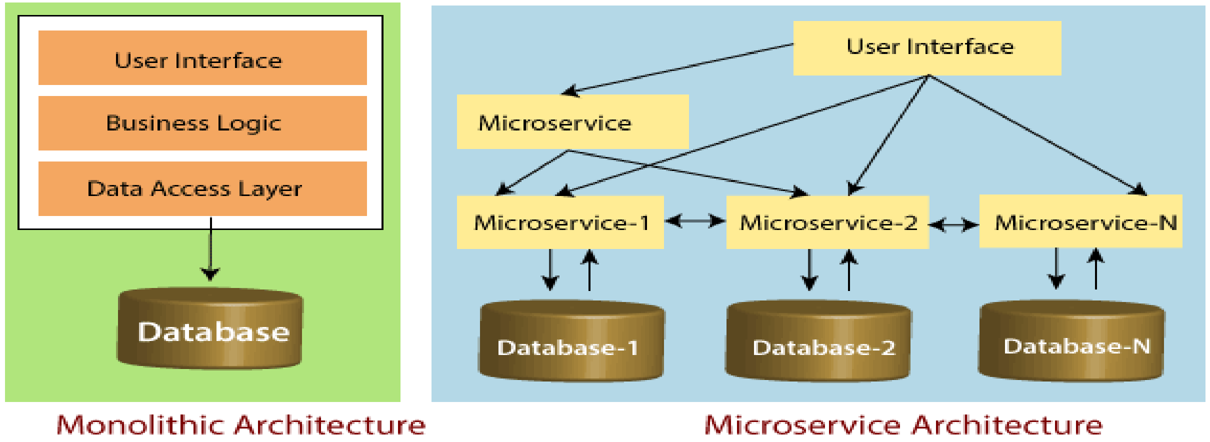
1

## Introduction
## Monolith vs Microservices Architecture

Atos | Syntel

## Microservices and Introduction to .NET Core
### Monolithic vs Microservices Architecture

A Service oriented architecture is a design helps us to build distributed systems, and these systems deliver services to other applications through the protocol.



Monolithic vs Microservice Architecture

# Microservices and Introduction to .NET Core
## Monolithic vs Microservices Architecture

**Monolithic Architecture**
UI, Business Logic  and Data Access Layer has been combined in one complete package which is communicating with one single database.

**Microservices Architecture**
These are small tiny services communicating to each other.

Microservices is an architectural style to build the applications.

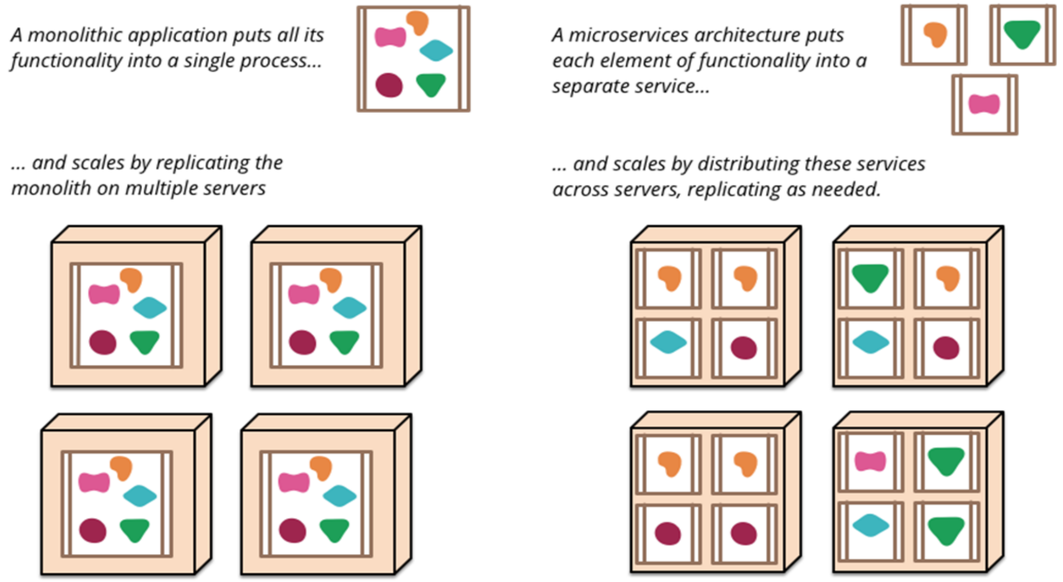
It is next to Service-Oriented Architecture (SOA).

The most important feature of the microservice-based architecture is that it can perform continuous delivery of a large and complex application.

Microservices can be designed using WCF services, REST(Representational State Transfer Services using ASP.NET Core Web API or Java Web Services

# Microservices and Introduction to .NET Core

## Monolithic vs Microservices Architecture



A monolithic application puts all its functionality into a single process...
... and scales by replicating the monolith on multiple servers
A microservices architecture puts each element of functionality into a separate service...
... and scales by distributing these services across servers, replicating as needed.

For Monolithic applications, Scales is done by cloning the app on multiple servers.

For microservices, scales is done by deploying each independently with multiple instances across servers

# Microservices and Introduction to .NET Core

## Monolithic vs Microservices Architecture

- **Choosing a monolithic architecture**
  - **If your company is a small team.** This way you don’t have to deal with the complexity of deploying a microservice architecture.
  - **If you want a quicker launch.** Monolithic architecture requires less time to launch. This system will require more time later on to update your system, but the initial launch is quicker.

- **Choosing a microservices architecture**
  - **If you want to develop a more scalable application.** Scaling a microservices architecture is far easier. New capabilities and modules can be added with much ease and speed.
  - **If your company is larger or plans to grow.** Using microservices is great for a company that plans to grow, as a microservices architecture is far more scalable and easier to customize over time.

## Principles of Microservices

2

Atos | Syntel

# Microservices and Introduction to .NET Core

## Principles of Microservices

- Single Responsibility Principle
  - The single responsibility principle states that a class or a module in a program should have only one responsibility. Any microservice cannot serve more than one responsibility, at a time.

- Modelled around business domain
  - Microservice never restrict itself from accepting appropriate technology stack or database. The stack or database is most suitable for solving the business purpose.

- Isolated Failure
  - The large application can remain mostly unaffected by the failure of a single module. It is possible that a service can fail at any time. So, it is important to detect failure quickly, if possible, automatically restore failure.

## Microservices and Introduction to .NET Core

### Principles of Microservices

- Infrastructure Automation
  - The infrastructure automation is the process of scripting environments. With the help of scripting environment, we can apply the same configuration to a single node or thousands of nodes. It is also known as configuration management, scripted infrastructures, and system configuration management.

- Deploy independently
  - Microservices are platform agnostic. It means we can design and deploy them independently without affecting the other server.

3

## Benefits & Drawbacks

# Microservices and Introduction to .NET Core
## Benefits & Drawbacks

- **Benefits**
  - Improves Scalability and Productivity
  - Integrates well with legacy systems
  - Sustainable development
  - Cross-functionality

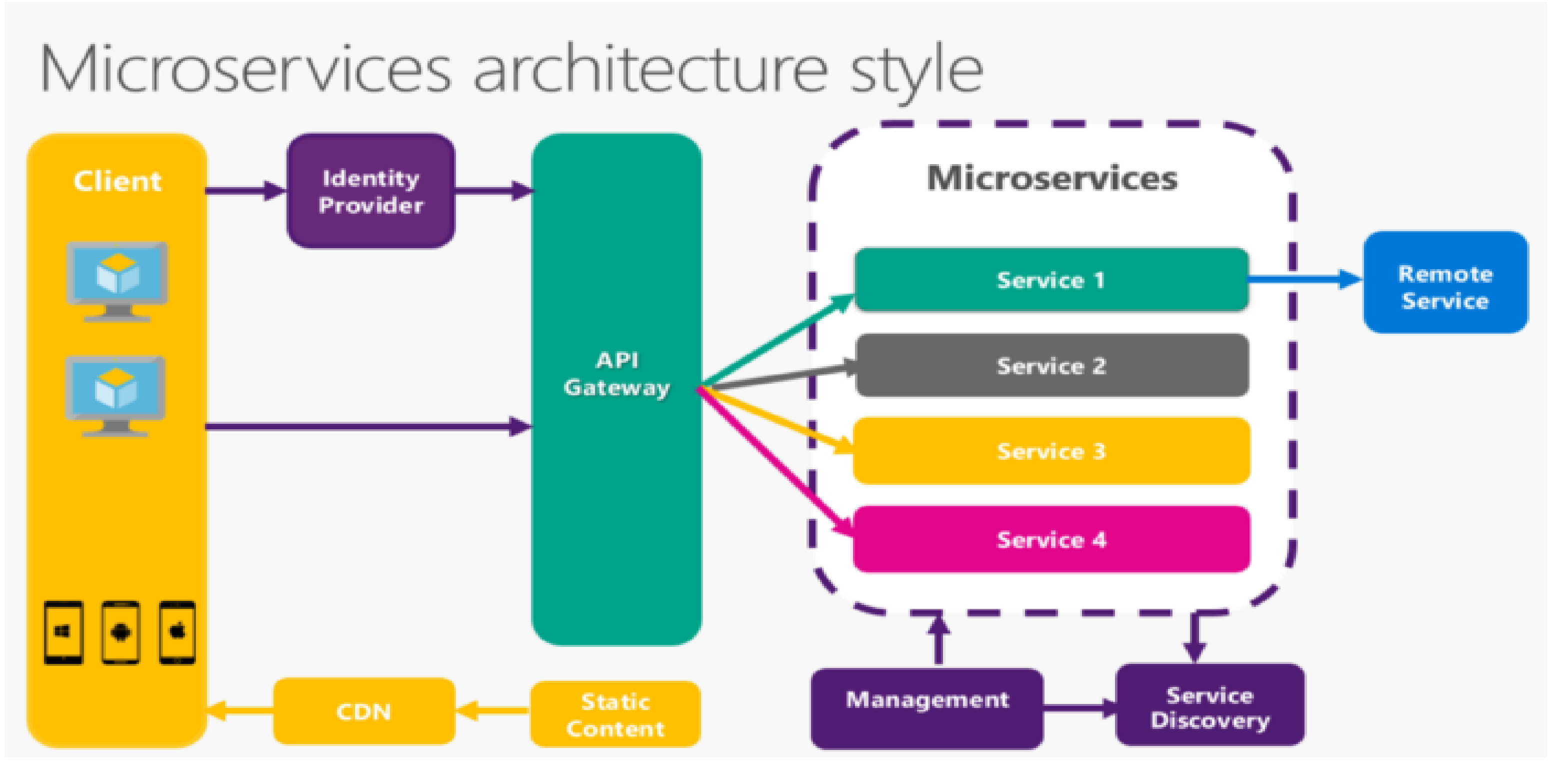
- **Drawbacks**

  - Deployment requires more effort
  - Testing must be independent
  - Difficult to change multiple microservices

# Microservices Architecture

# Microservices and Introduction to .NET Core

## Microservices Architecture



# Microservices and Introduction to .NET Core

## Microservices Architecture

- **Management**. Maintains the nodes for the service.
- **Identity Provider**. Manages the identity information and provides authentication services within a distributed network.
- **Service Discovery**. Keeps track of services and service addresses and endpoints.
- **API Gateway**. Serves as client's entry point. Single point of contact from the client which in turn returns responses from underlying microservices and sometimes an aggregated response from multiple underlying microservices.
- **CDN**. A content delivery network to serve static resources for e.g. pages and web content in a distributed network
- **Static Content** The static resources like pages and web content

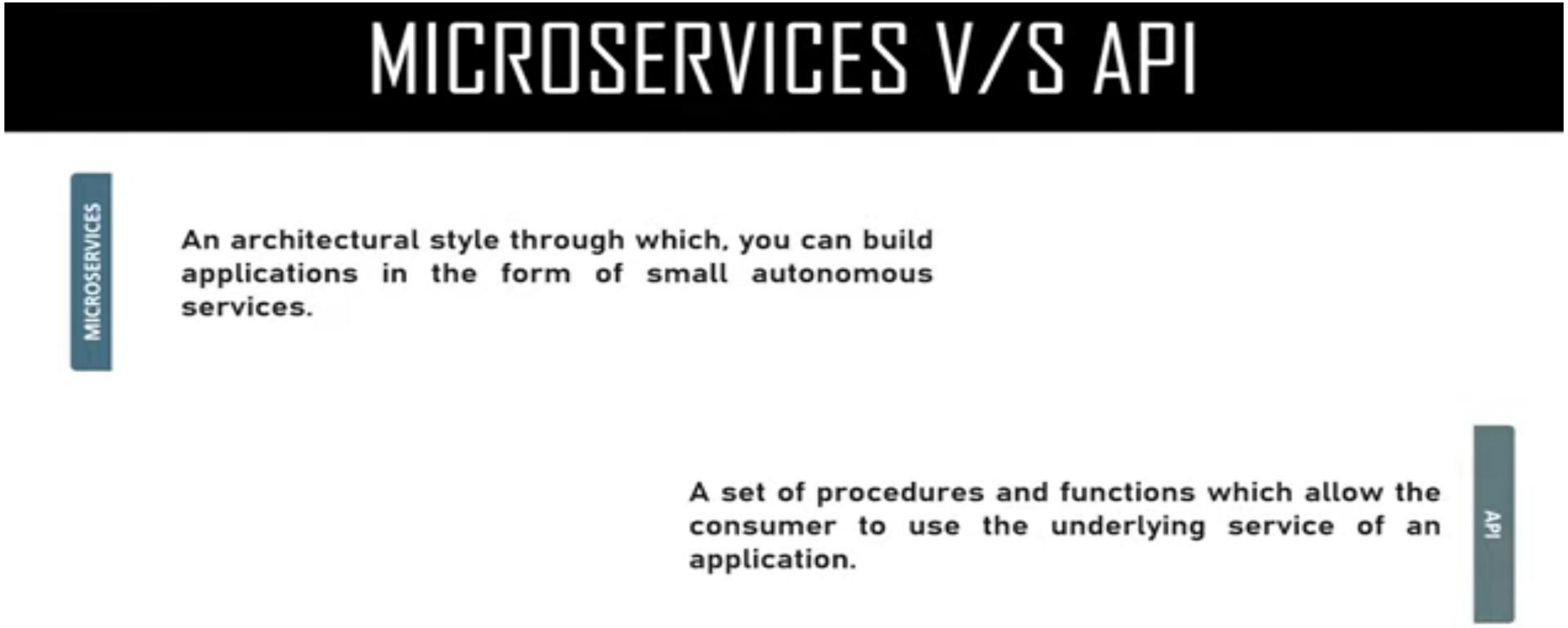
5

## Microservices Vs API

Atos | Syntel

# Microservices and Introduction to .NET Core

## Microservices Vs API



MICROSERVICES V/S API

MICROSERVICES

An architectural style through which, you can build applications in the form of small autonomous services.

A set of procedures and functions which allow the consumer to use the underlying service of an application.

API

## Microservices and Introduction to .NET Core
### Microservices Vs API

- Application Program Interface or APIs is a way through which you can make sure two or more applications communicate with each other to process the client request.

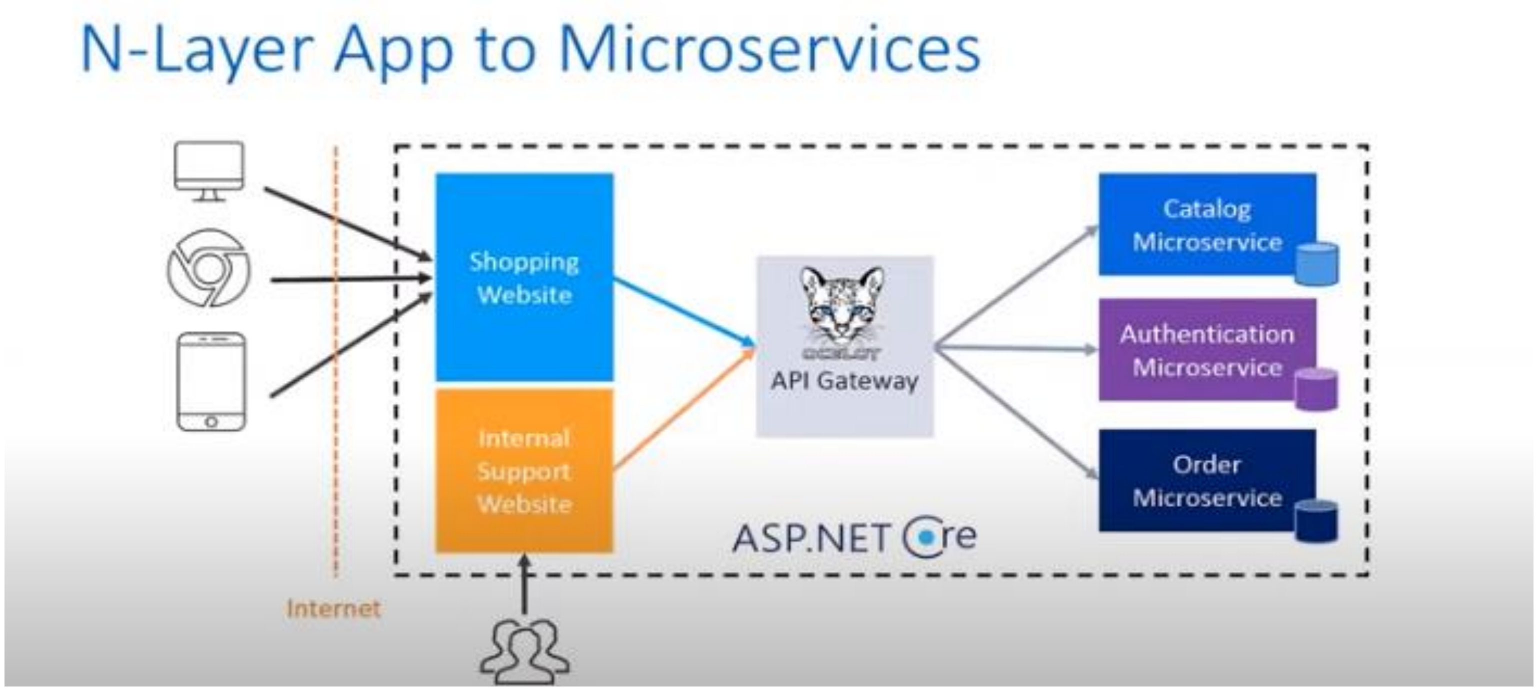
- APIs are developed in REST full style those are nothing but HTTP methods like
  - Create → Post
  - Read → Get
  - Update → Put
  - Delete → Delete

6

## N-Layer App To Microservices

# Microservices and Introduction to .NET Core

## N-Layer App to Microservices



# Microservices and Introduction to .NET Core

## N-Layer App to Microservices

- Microservices can be designed using WCF services, REST Services using ASP.NET Core Web API or any Java Web Services

- API Gateway helps to give single endpoint to client to communicate with various microservices.
  - Azure API Management
  - Ocelot API Gateway (Open Source)
  - Express API Gateway
  - Loopback API Gateway

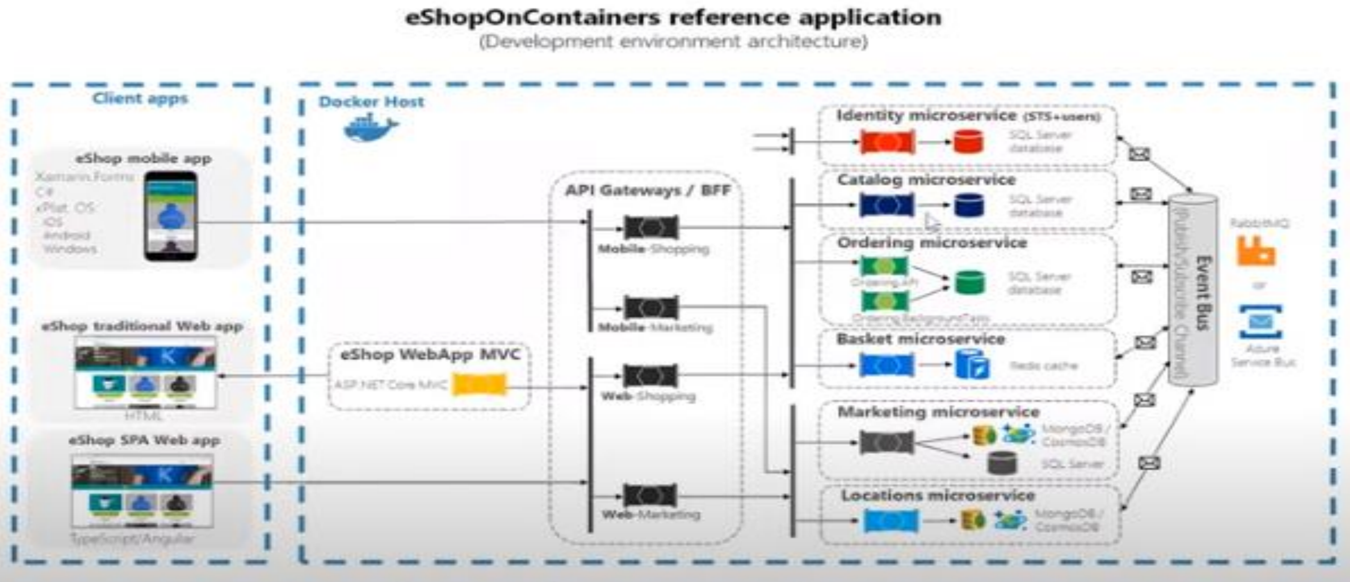
- Authentication and Authorization also can be implemented at API Gateway level.

7

## Complex Microservices Architecture

# Microservices and Introduction to .NET Core
## Complex Microservices Architecture

Microservices has their independent database or two or more microservices can share the database.



*Reference: https://github.com/dotnet-architecture/eShopOnContainers/tree/master*

8

## Microservices Challenges

## Microservices and Introduction to .NET Core

### Microservices Challenges

- Difficult to achieve strong consistency across services
- ACID transactions do not span multiple processes.
- Distributed System so hard to debug and trace the issues
- Greater need for an end to end testing
- Required cultural changes in across teams like Dev and Ops working together even in the same team.

# Introduction to .Net Core

9

Atos | Syntel

## Introduction to .NET Core

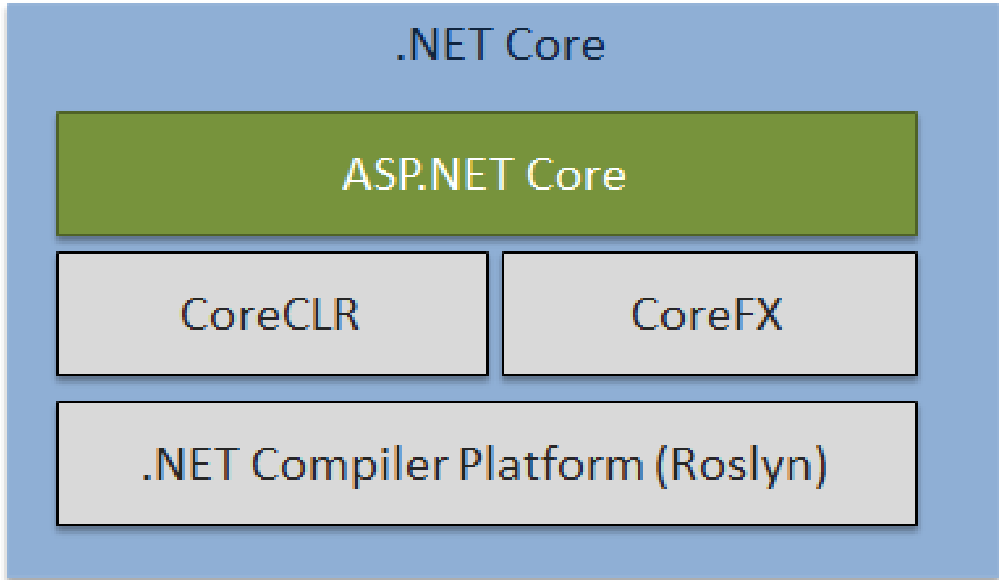
- .NET Core is the latest general purpose, open source development platform maintained by Microsoft.
- It works across different platforms and has been redesigned in a way that makes .NET fast, flexible and modern.
- .NET Core applications now can be build not only windows but on Android, iOS, Linux, Mac etc.
- Flexible Deployment
  - Self Contained Deployment
    - Publishing your app as *self-contained* produces an application that includes the .NET runtime and libraries, and your application and its dependencies. Users of the application can run it on a machine that doesn't have the .NET runtime installed.

## Introduction to .NET Core

- Framework Dependent Deployment
  - Publishing your app as *framework-dependent* produces an application that includes only your application itself and its dependencies. Users of the application have to separately install the .NET runtime.

- ▶ Modular
- ▶ Built in support for dependency injection
- ▶ Provides dotent cli(Command Line Interface) do develop applications on other platforms too.
- ▶ Target Multiple Frameworks

## Microservices and Introduction to .NET Core
### Complex Microservices Architecture



# Introduction to .NET Core

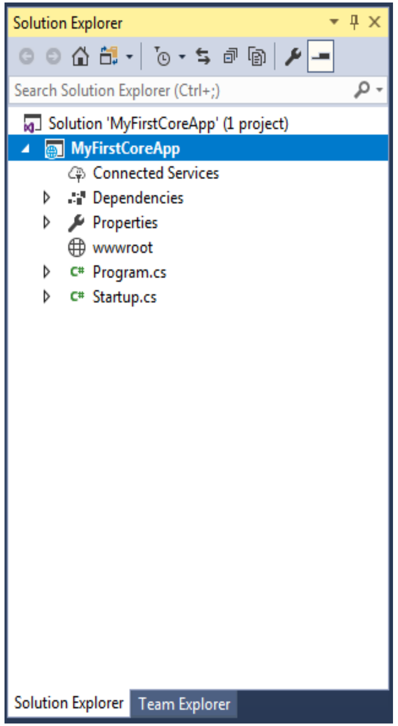
- . ASP.NET Core is a free, open-source and cloud optimized web framework which can run on Windows, Linux, or Mac.

- ASP.NET Core is a modular framework distributed as NuGet packages. This allows us to include packages that are required in our application.

- ASP.NET Core applications run on both, .NET Core and traditional .NET framework (.NET Framework 4.x).
- Built in support for dependency injection

# Microservices and Introduction to .NET Core

## ASP.NET Core Project Structure



By default, the **wwwroot** folder in the ASP.NET Core project is treated as a web root folder. Static files can be stored in any folder under the web root and accessed with a relative path to that root.

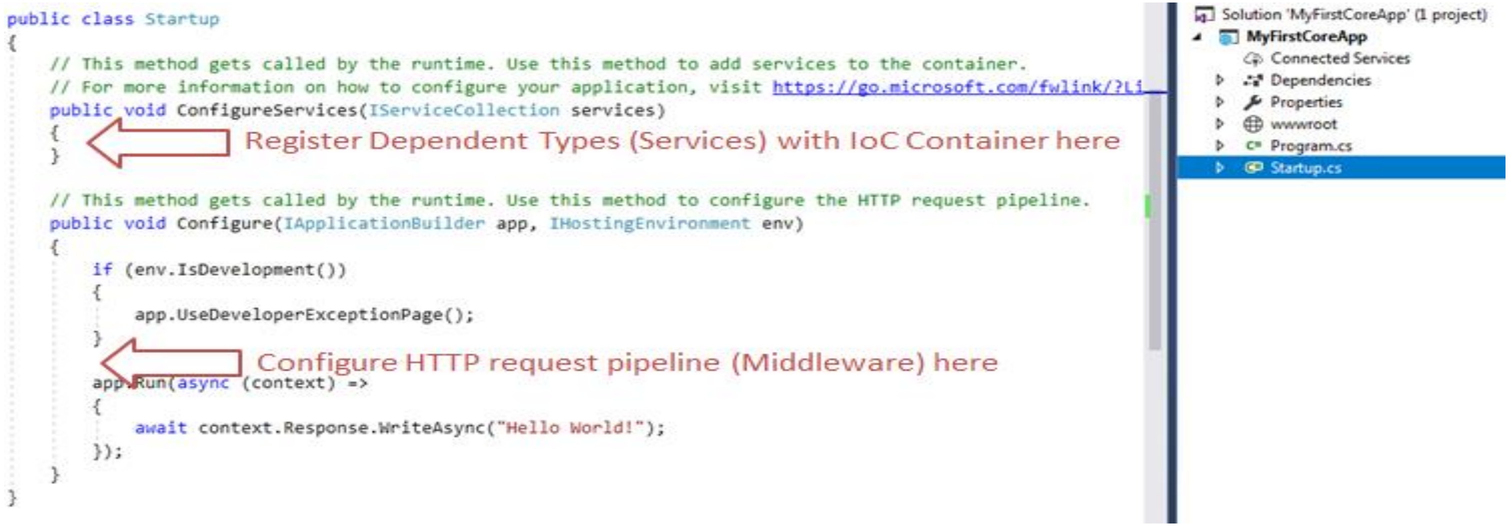
Only those files that are in the web root - wwwroot folder can be served over an http request. All other files are blocked and cannot be served by default.

You can rename wwwroot folder to any other name as per your choice and set it as a web root while preparing hosting environment in the program.cs.

For example,wwwroot folder has been renamed  to Content folder.
Call UseWebRoot() method to configure Content folder as a web root folder in the Main() method of Program class

ASP.NET Core web application is actually a console project which starts executing from the entry point public static void Main() in Program class where we can create a host for the web application.

# Microservices and Introduction to .NET Core
## ASP.NET Core Project Structure



```
public class Startup
{
    // This method gets called by the runtime. Use this method to add services to the container.
    // For more information on how to configure your application, visit https://go.microsoft.com/fwlink/?Li
    public void ConfigureServices(IServiceCollection services)
    {
    }

    // This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
    public void Configure(IApplicationBuilder app, IHostingEnvironment env)
    {
        if (env.IsDevelopment())
        {
            app.UseDeveloperExceptionPage();
        }

        app.Run(async (context) =>
        {
            await context.Response.WriteAsync("Hello World!");
        });
    }
}
```

Register Dependent Types (Services) with IoC Container here
Configure HTTP request pipeline (Middleware) here
Solution 'MyFirstCoreApp' (1 project)
MyFirstCoreApp
Connected Services
Dependencies
Properties
wwwroot
Program.cs
Startup.cs

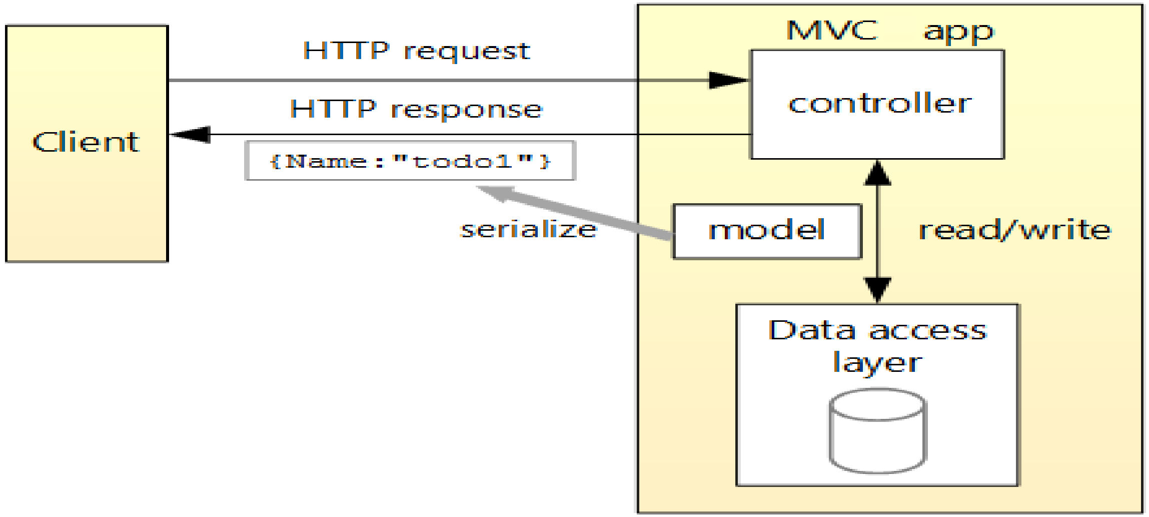
The Configure method includes three parameters IApplicationBuilder to define Http Request Pipeline, IHostingEnvironment, and ILoggerFactory by default. These services are framework services injected by built-in IoC container.
At run time, the ConfigureServices method is called before the Configure method. This is so that you can register your custom service with the IoC container which you may use in the Configure method.

# ASP.NET Core Web API Basics

# Microservices and Introduction to .NET Core

## ASP.NET Core Web API Basics

- ASP.NET Core Web API is a framework for building HTTP services that can be accessed from any client including browsers and mobile devices. It is an ideal platform for building RESTful applications on the .NET Core.
- It doesn't have any user interface or view.



## Microservices and Introduction to .NET Core
### ASP.NET Core Web API Basics

- Database - Here we store data and nothing more, no logic.
- DAL - To access the data, we use the Unit of Work pattern and, in the implementation, we use the ORM EF Core with code first and migration patterns.
- Business logic
- REST API - The actual interface through which clients can work with our API will be implemented through ASP.NET Core. Route configurations are determined by attributes.

# Thank you

For more information please contact:
T+ 33 1 98765432
M+ 33 6 44445678
firstname.lastname@atos.net

Atos | Syntel