Contents

[Collections 2](#_Toc76114969)

[stack vs heap memory 3](#_Toc76114970)

[Delegate amd multi cast delegate 3](#_Toc76114971)

[Abtraction v/s encapsulation 3](#_Toc76114972)

[How to pass from parent to child component 3](#_Toc76114973)

[Sharing Data from Child to Parent in Angular 8 Using @viewchild 4](#_Toc76114974)

[How to pass from parent to child grid 5](#_Toc76114975)

[How to dynamically implemnt naviagtion bar in angular 5](#_Toc76114976)

[What is docker and containers 5](#_Toc76114977)

[What is kubernates 5](#_Toc76114978)

[How to host web api and angular apps in azure 5](#_Toc76114979)

[Design Patterns 7](#_Toc76114980)

[Design pattern for toggle features 7](#_Toc76114981)

[Which are the three main categories of design patterns? 7](#_Toc76114982)

[Calling Store proc in EF 11](#_Toc76114983)

[\*\*\*Differences between a task and a thread. 24](#_Toc76114984)

[Sql peformance Improvements 24](#_Toc76114985)

[Preventing Sql Injection 24](#_Toc76114986)

[ASP.NET performance improvement 25](#_Toc76114987)

[Angular Performance Improvement 25](#_Toc76114988)

[token expiration and refresh web api angular 8 25](#_Toc76114989)

[MVC.net core life cycle 25](#_Toc76114990)

[ASP.NET CORE and WEB API 26](#_Toc76114991)

[The .NET Framework 28](#_Toc76114992)

[.NET Core 28](#_Toc76114993)

[What is Metapackages? 32](#_Toc76114994)

[What is the startup class in ASP.NET core? 32](#_Toc76114995)

[1. What is the use of ConfigureServices method of startup class? 32](#_Toc76114996)

[1. What is the use of the Configure method of startup class? 32](#_Toc76114997)

[How to enable Session in ASP.NET Core? 33](#_Toc76114998)

[1. What are the various JSON files available in ASP.NET Core? 33](#_Toc76114999)

[1. How to specify service lifetime for register service that added as a dependency? 33](#_Toc76115000)

[Singleton 34](#_Toc76115001)

[Transient 34](#_Toc76115002)

[Scoped 34](#_Toc76115003)

[Prevent Cross-Site Scripting (XSS) 36](#_Toc76115004)

[Best Practices against Cross Site Scripting Attacks 36](#_Toc76115005)

[End point routing 37](#_Toc76115006)

[Enable CORS with attributes 38](#_Toc76115007)

[ASP.NET Core - Middleware 39](#_Toc76115008)

[Throttling ASP.NET Web API calls 41](#_Toc76115009)

[Micro Services 42](#_Toc76115010)

[how do microservices communicate with each other 45](#_Toc76115011)

[API Gateway 46](#_Toc76115012)

[Get many resources in a single request 52](#_Toc76115013)

[Evolve your AP In without versions 52](#_Toc76115014)

[Cloud -AZURE 53](#_Toc76115015)

[What is a Resource group? 55](#_Toc76115016)

[What are Azure App Services? 55](#_Toc76115017)

[What are the different ways to host web sites in Azure? 56](#_Toc76115018)

[*Can I use json instead of yaml for my Compose file?* 58](#_Toc76115019)

[Yes. Yaml is a superset of json so any JSON file should be valid Yaml. To use a JSON file with Compose, specify the filename to use, for 58](#_Toc76115020)

[Deploying C# Web Applications with Docker 58](#_Toc76115021)

## Open Closed Principle (OCP)

## Software entities (classes, modules, functions, etc.) should be open for extension, but closed for modification. This principle suggests that the class should be easily extended but there is no need to change its core implementations. i.e. New features should be implemented using the new code, but not by changing existing code. The main benefit of adhering to OCP is that it potentially streamlines code maintenance and reduces the risk of breaking the existing implementation.

Let’s take an example of bank accounts like regular savings, salary saving, corporate etc. for different customers. As for each customer type, there are different rules and different interest rates. The code below violates OCP principle if the bank introduces a new Account type. Said code modifies this method for adding a new account type.

public class Account

{

public decimal Interest { get; set; }

public decimal Balance { get; set; }

// members and function declaration

public decimal CalcInterest(string accType)

{

if (accType == "Regular") // savings

{

Interest = (Balance \* 4) / 100;

if (Balance < 1000) Interest -= (Balance \* 2) / 100;

if (Balance < 50000) Interest += (Balance \* 4) / 100;

}

else if (accType == "Salary") // salary savings

{

Interest = (Balance \* 5) / 100;

}

else if (accType == "Corporate") // Corporate

{

Interest = (Balance \* 3) / 100;

}

return Interest;

}

}

We can apply OCP by using interface, abstract class, abstract methods and virtual methods when you want to extend functionality. Here I have used interface for example only but you can go as per your requirement.

interface IAccount

{

// members and function declaration, properties

decimal Balance { get; set; }

decimal CalcInterest();

}

//regular savings account

public class RegularSavingAccount : IAccount

{

public decimal Balance { get; set; } = 0;

public decimal CalcInterest()

{

decimal Interest = (Balance \* 4) / 100;

if (Balance < 1000) Interest -= (Balance \* 2) / 100;

if (Balance < 50000) Interest += (Balance \* 4) / 100;

return Interest;

}

}

//Salary savings account

public class SalarySavingAccount : IAccount

{

..etc …

## Liskov Substitution Principle (LSP)

LSP states that the child class should be perfectly substitutable for their parent class. If class C is derived from P then C should be substitutable for P. We can check using LSP that inheritance is applied correctly or not in our code

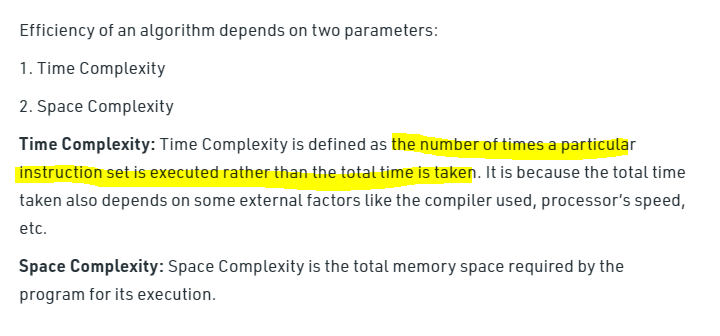
**DI Principle :** , the principle says that there should not be a tight coupling among components of software and to avoid that, the components should depend on abstraction.

The terms Dependency Injection (DI) and Inversion of Control (IoC) are generally used as interchangeably to express the same design pattern. The pattern was initially called IoC, but Martin Fowler (known for designing the enterprise software) anticipated the name as DI because all frameworks or runtime invert the control in some way and he wanted to know which aspect of control was being inverted.

**IoC** (**I**nversion **o**f **C**ontrol) :- It’s a generic term and implemented in several ways (container register, DI, events, delegates etc).

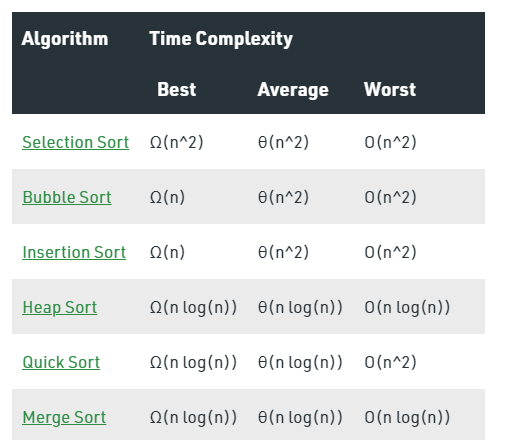
**DI** (**D**ependency **I**njection) :- DI is a sub-type of IoC and is implemented by constructor injection, setter injection or Interface injection.

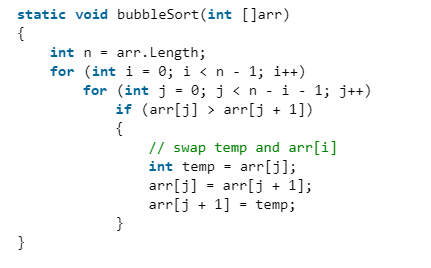
# Complexities of Algorithms



Both are calculated as the function of input size(n).

One important thing here is that in spite of these parameters the efficiency of an algorithm also depends upon the **nature**and **size of**the **input**





Return multiple o/p from method

Out param ,Ref param,Tuple,Collection,class

How to pass multiple args to method

Aruments , array ,class,tuple,json obj,…

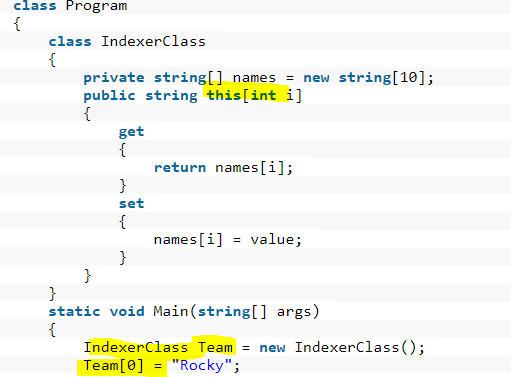
## Break will exit the loop completely, continue will just **skip** the current iteration.

## Collections

<https://www.codeproject.com/Articles/832189/List-vs-IEnumerable-vs-IQueryable-vs-ICollection-v>

### indexrs in C#

C# indexers are usually known as smart arrays. A C# indexer is a class property that allows you to access a member variable of a class or struct using the features of an array. In C#, indexers are created using this keyword. Indexers in C# are applicable on both classes and structs.



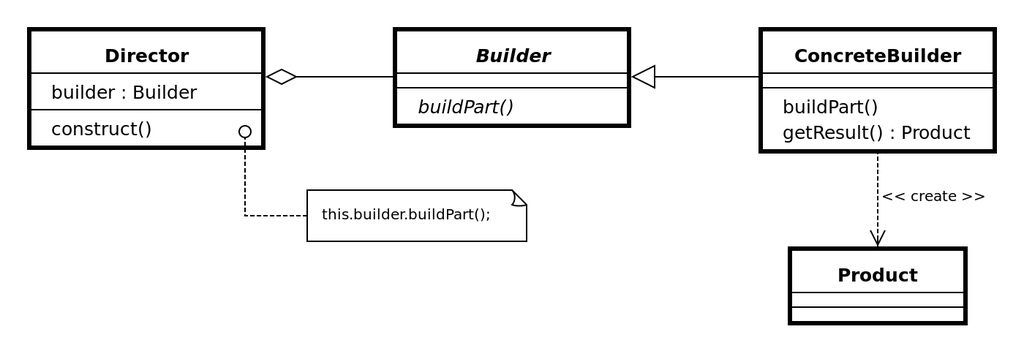


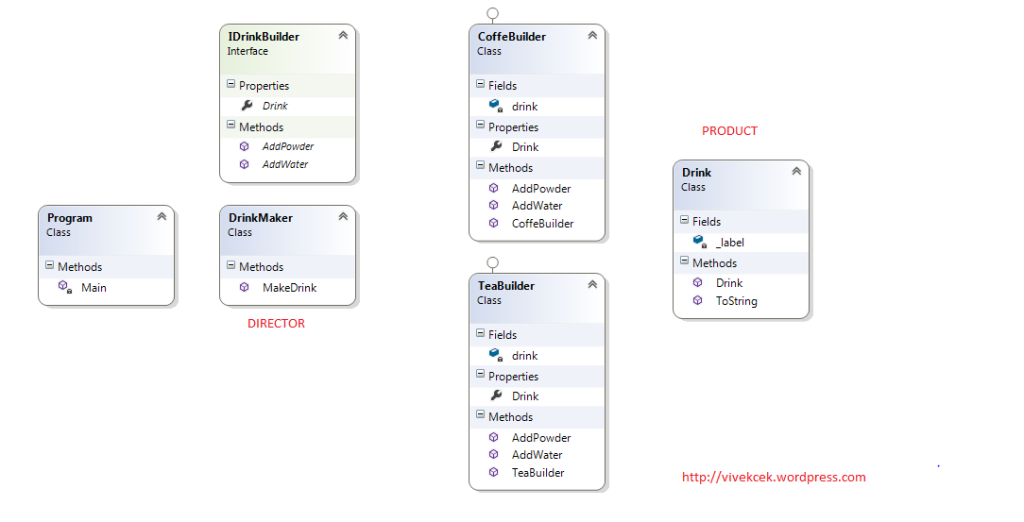
Extension Methods

Builder Pattren for tea/coffe making :

<https://vivekcek.wordpress.com/2014/06/28/new-vending-machine-at-office-and-builder-design-pattern/>

<https://dotnettutorials.net/lesson/builder-design-pattern-real-time-example/>





## stack vs heap memory

## Delegate amd multi cast delegate

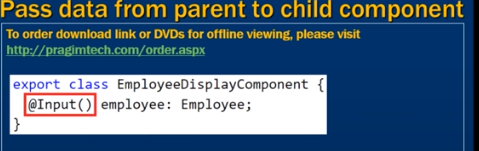
## Abtraction v/s encapsulation

## How to pass from parent to child component

<https://www.c-sharpcorner.com/blogs/update-the-child-component-and-parent-component-using-input-output-in-angular-4>

pass data from a parent component to a child directive or component using the @Input() decorator in the child component/directive.

To let Angular know that a property in a child component or directive can receive its value from its parent component we must use the @Input() decorator in the said child.



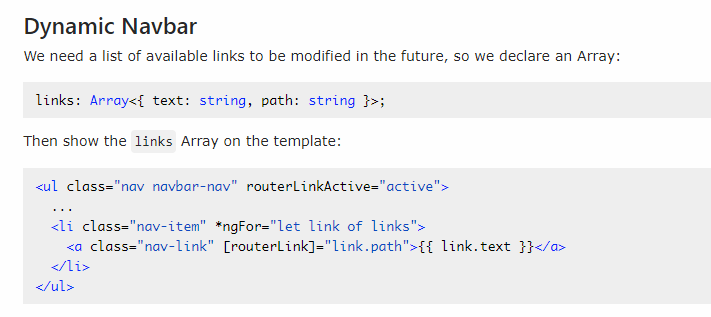
# Sharing Data from Child to Parent in Angular 8 Using @viewchild

<https://dzone.com/articles/sharing-data-from-child-to-parent-in-angular-8-usi>



## How to pass from parent to child grid

## How to dynamically implemnt naviagtion bar in angular



## What is docker and containers

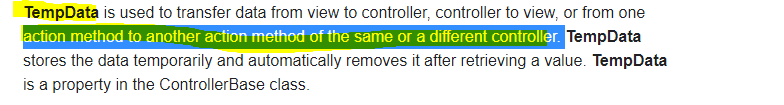
## What is kubernates

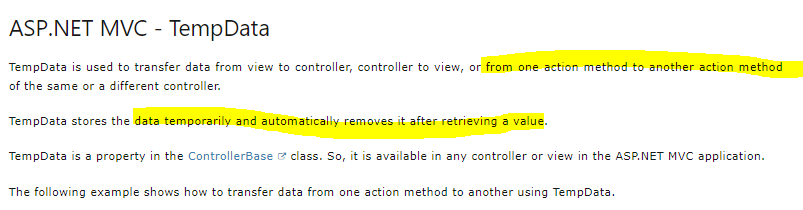
## How to host web api and angular apps in azure

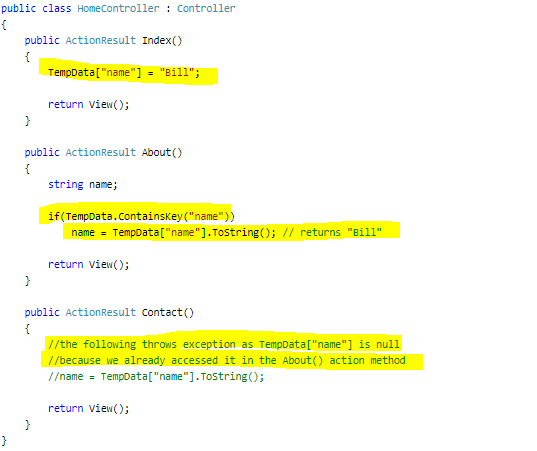
## Grasp 8 Angular 4 Lifecycle Hooks In Less Than 8 Minutes

***Are you familiar with third party libraries such as PrimeNG or ag-Grid? Have you used any of those in your projects?***

Angular material, PrimeNG: ======== PrimeNG is a collection of rich UI components for Angular. All widgets are open source and free to use under MIT License. PrimeNG is developed by PrimeTek Informatics ag-Grid: ======= The "ag" part of ag-Grid stands for "agnostic". The internal ag-Grid engine is implemented in TypeScript with zero dependencies. ag-Grid supports Angular through a wrapper component. The wrapper lets you use ag-Grid in your application like any other Angular component – you pass configuration through property bindings and handle events through event bindings. You can even use Angular components to customize the grid UI and cell contents / behavior.







## Design Patterns

## Design pattern for toggle features

<https://www.c-sharpcorner.com/UploadFile/questpond/design-pattern-interview-questions/>

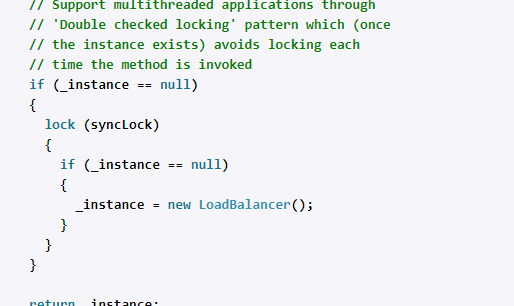
## Which are the three main categories of design patterns?

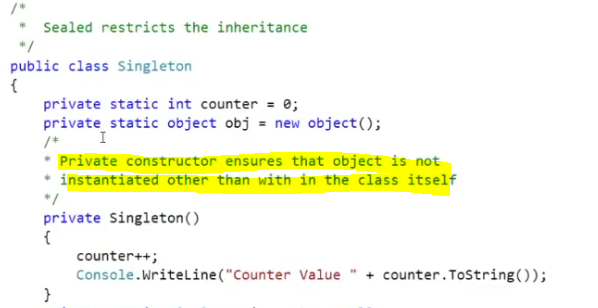
There are three basic classifications of patterns Creational, Structural, and Behavioral patterns.  
***Creational Patterns***

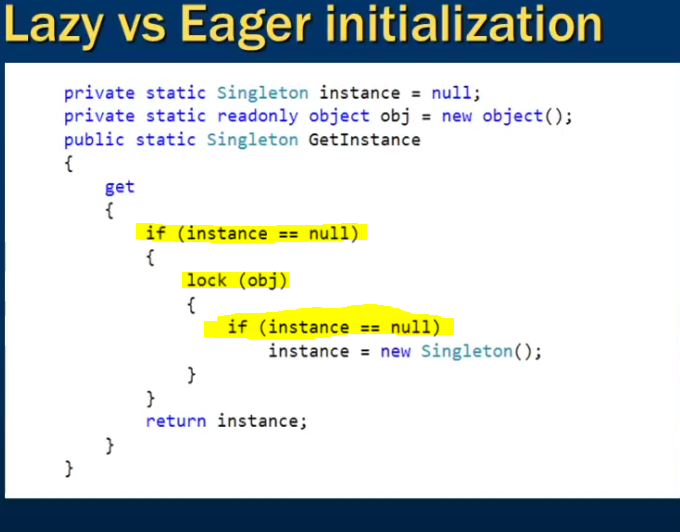
Factory Method : Creates an instance of several derived classes  
Abstract Factory : Creates an instance of several families of classes/superset of factory pattren  
Builder : Separates object construction from its representation  
Prototype : A fully initialized instance to be copied or cloned  
Singleton : A class in which only a single instance can exist  
 **Note**   
  
The best way to remember Creational pattern is by remembering ABFPS (Abraham Became First President of States).  
  
***Structural Patterns***  
Adapter : Match interfaces of different classes .  
Bridge : Separates an object's abstraction from its implementation.  
Composite : A tree structure of simple and composite objects.  
Decorator : Add responsibilities to objects dynamically.  
Flyweight : A fine-grained instance used for efficient sharing.  
Proxy : An object representing another object.  
  
**Note**  
To remember structural pattern best is (ABCDFFP)  
  
***Behavioral Patterns***  
Mediator : Defines simplified communication between classes.  
Memento : Capture and restore an object's internal state.  
Interpreter : A way to include language elements in a program.  
Iterator : Sequentially access the elements of a collection.  
Chain of Resp : A way of passing a request between a chain of objects.  
Command : Encapsulate a command request as an object.  
State : Alter an object's behavior when its state changes.  
Strategy : Encapsulates an algorithm inside a class.  
Observer : A way of notifying change to a number of classes.  
Template Method : Defer the exact steps of an algorithm to a subclass.  
Visitor : Defines a new operation to a class without change.

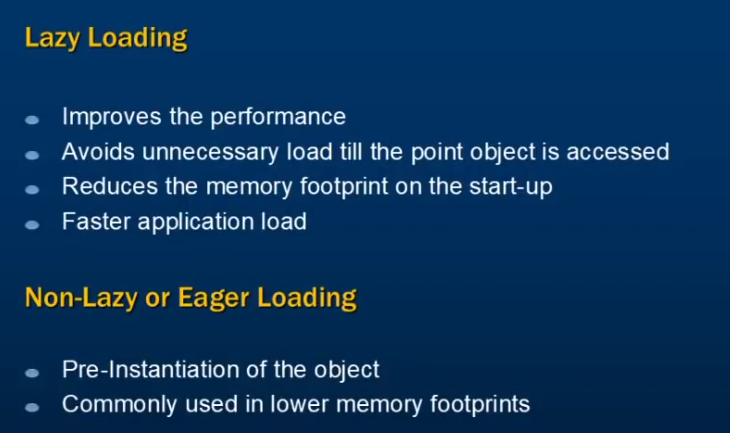
Singleton

<https://www.dofactory.com/net/singleton-design-pattern>

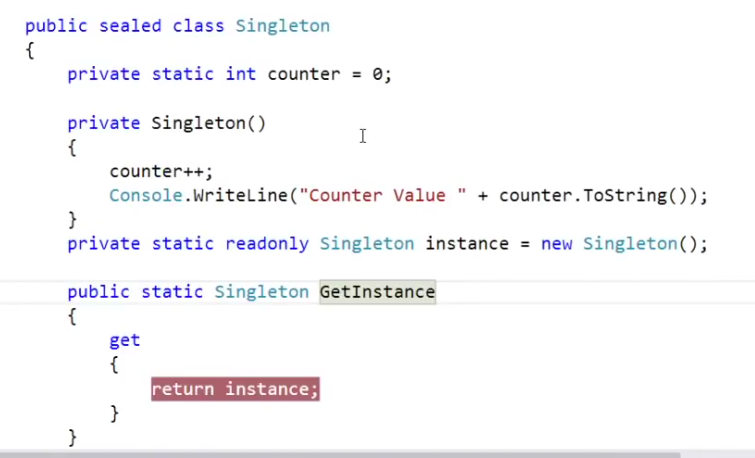


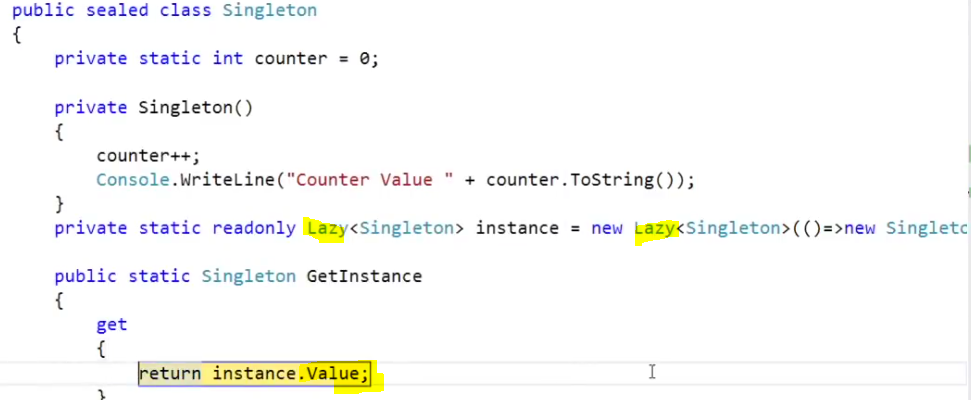




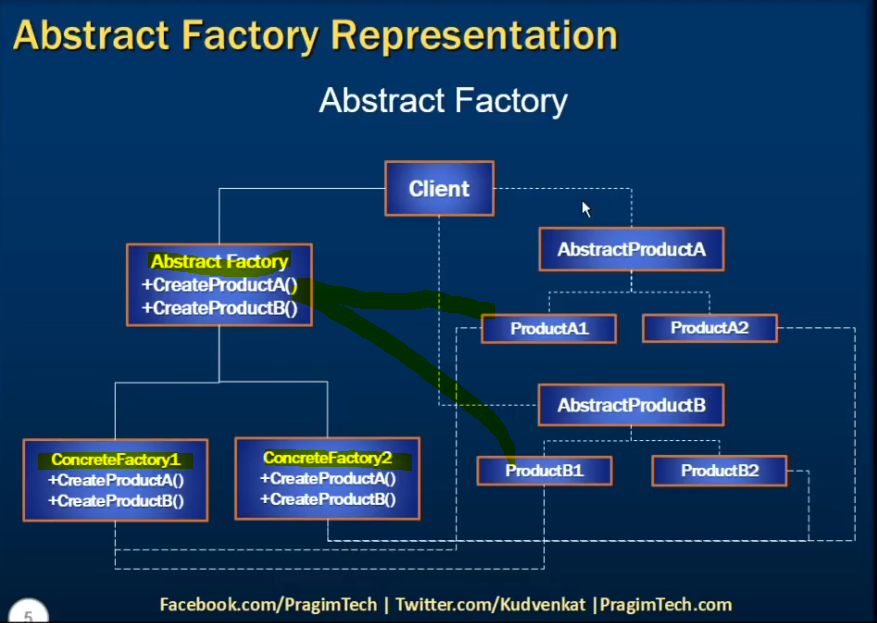


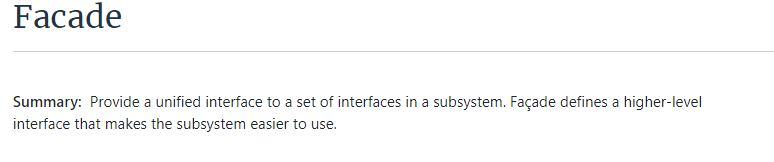
Non Lazy/Eager Loading











<https://www.dofactory.com/net/facade-design-pattern>

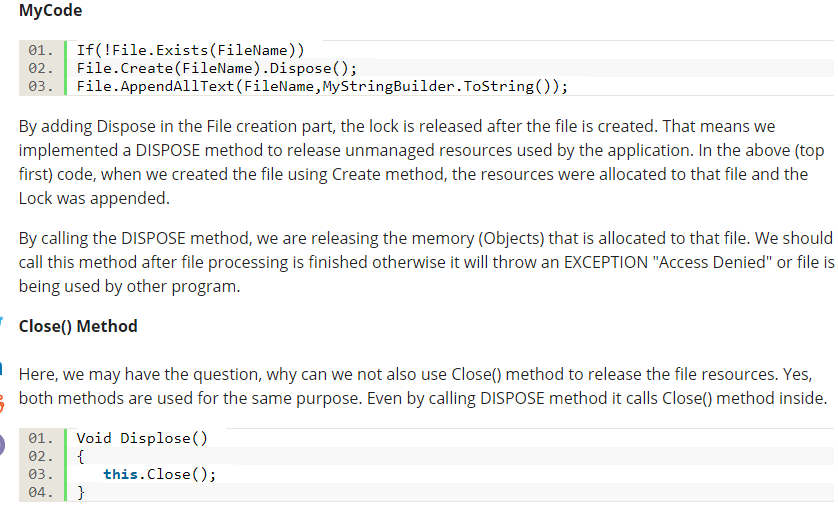
### Calling Store proc in EF

1. / Creating Custom class to hold result of Stored Procedure
2. **public** **class** EmployeeDetail
3. {
4. **public** **int** EmployeeID { get; set; }
5. **public** string EmployeeName { get; set; }
6. **public** string DepartmentName { get; set; }
7. }
9. // using Object Context (EF4.0)
10. using (Entities context = **new** Entities())
11. {
12. IEnumerable<EmployeeDetails> empDetails  =  context.ExecuteStoreQuery<EmployeeDetails>
13. ("exec GetEmployeeData").ToList();
14. }
16. // using DBContext (EF 4.1 and above)
17. using (Entities context = **new** Entities())
18. {
19. IEnumerable<EmployeeDetails> empDetails  =  context. Database.SqlQuery
20. < EmployeeDetails >("exec GetEmployeeData ", **null**).ToList();
21. }

<https://stackoverflow.com/questions/18690537/multi-processes-readwrite-one-file>



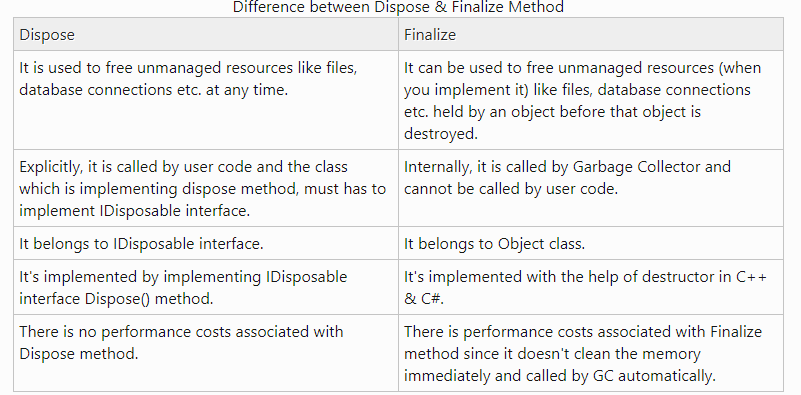
<https://www.c-sharpcorner.com/article/c-sharp-to-overcome-the-process-cannot-access-the-file-xyz-because-it-is-being-used/>



<https://bytes.com/topic/c-sharp/answers/493277-two-diffrent-process-tries-open-file-same-time>

<https://www.tutorialspoint.com/csharp/csharp_arrays.htm>

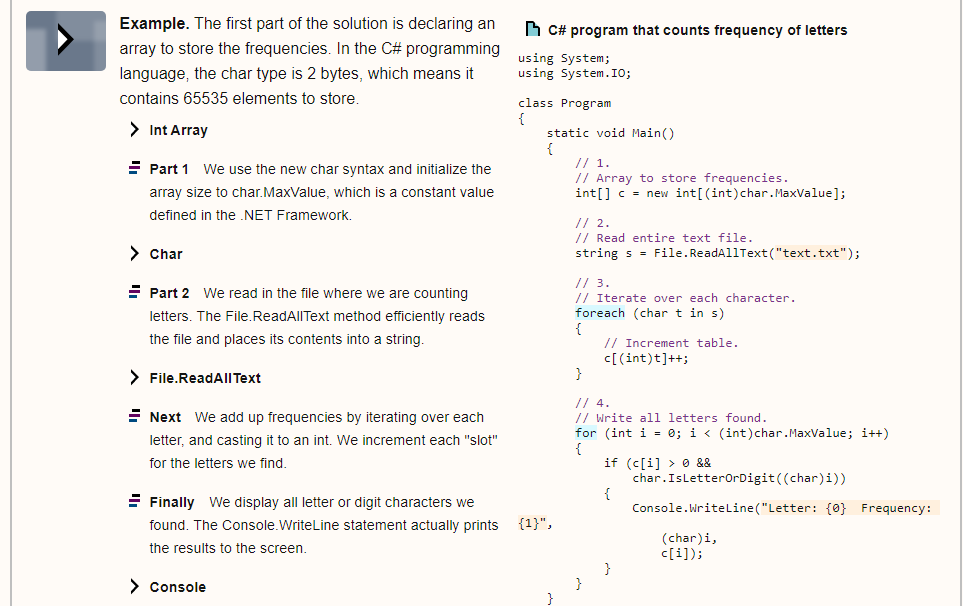
<https://www.tutorialspoint.com/csharp/csharp_jagged_arrays.htm>

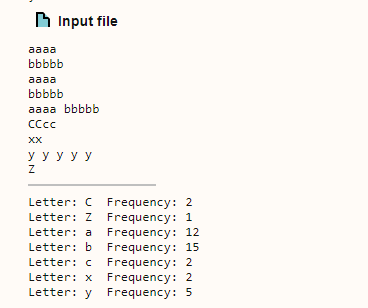


**C# Count Letter Frequencies**

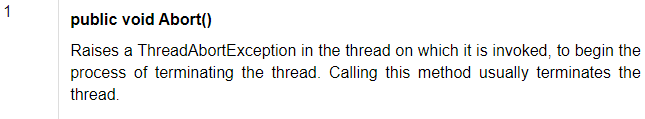
Implement a method that counts the frequencies of letters in text.

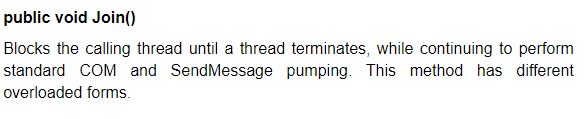
<https://www.dotnetperls.com/count-letter-frequencies>





<https://www.tutorialspoint.com/csharp/csharp_delegates.htm>

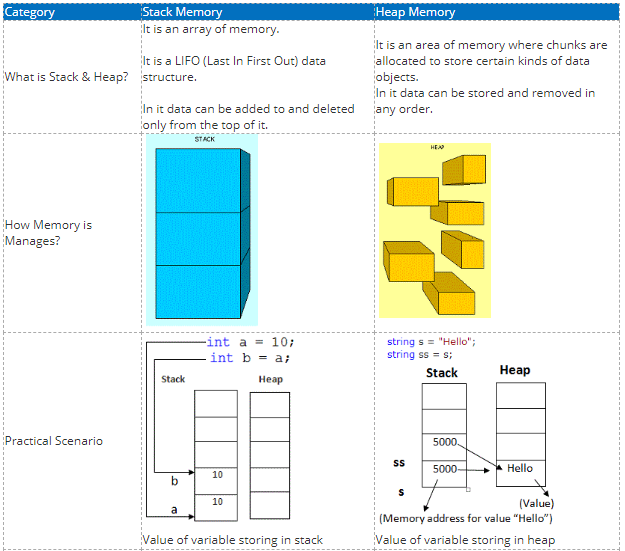




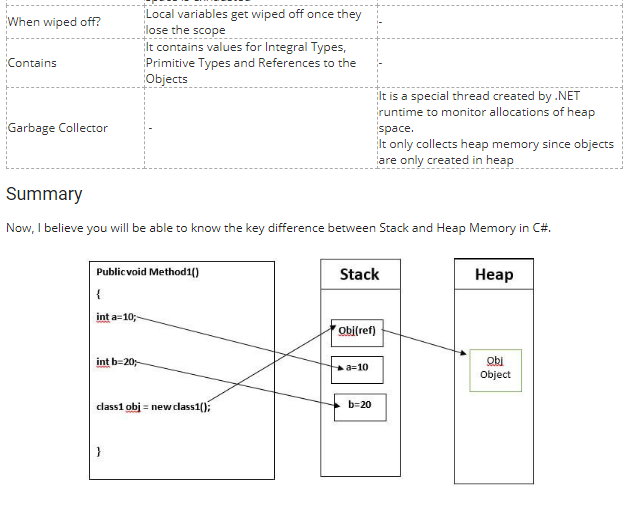
Compile time – Converts to execuable

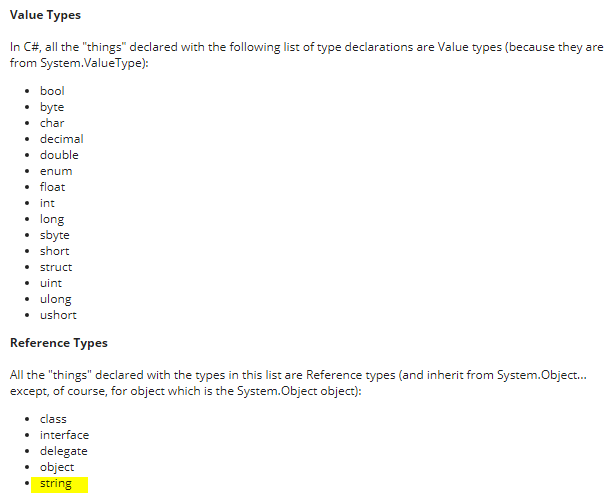
Run Time – Runs the executable









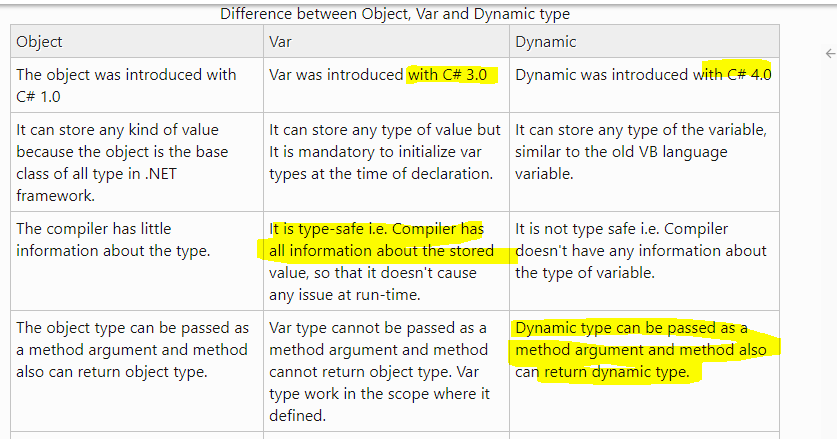


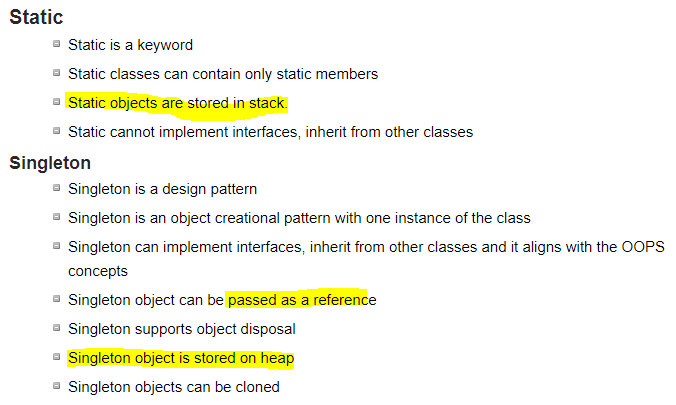
<https://www.c-sharpcorner.com/article/C-Sharp-heaping-vs-stacking-in-net-part-i/>

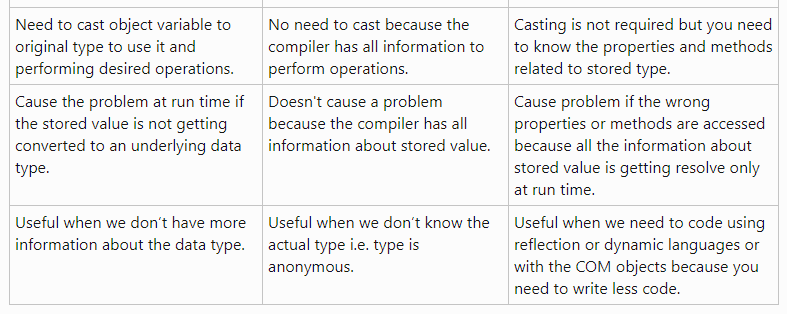
<https://www.tutorialspoint.com/Stack-and-Queue-in-Chash>

|  |  |
| --- | --- |
| Stacks | Queue |

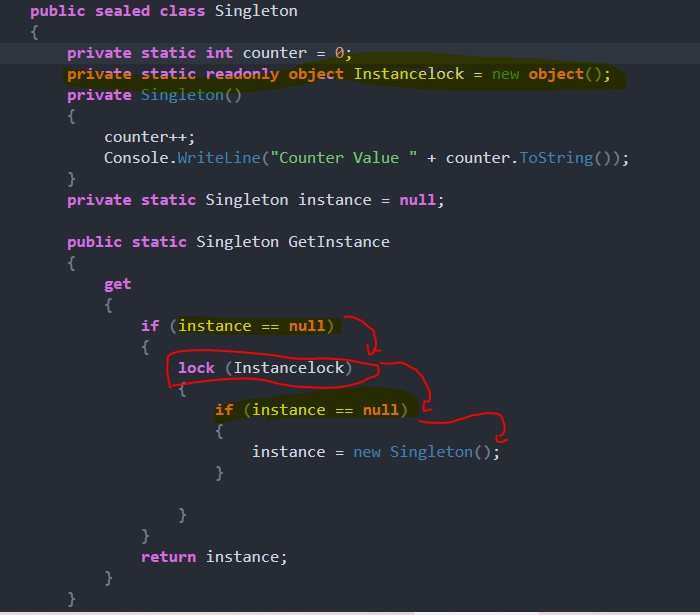
|  |  |
| --- | --- |
| Stacks | Queues |
| Stacks are based on the LIFO principle, i.e., the element inserted at the last, is the first element to come out of the list. | Queues are based on the FIFO principle, i.e., the element inserted at the first, is the first element to come out of the list. |
| Insertion and deletion in stacks takes place only from one end of the list called the top. | Insertion and deletion in queues takes place from the opposite ends of the list. The insertion takes place at the rear of the list and the deletion takes place from the front of the list. |
| Insert operation is called push operation. | Insert operation is called enqueue operation. |
| Delete operation is called pop operation. | Delete operation is called dequeue operation. |
| In stacks we maintain only one pointer to access the list, called the top, which always points to the last element present in the list. | In queues we maintain two pointers to access the list. The front pointer always points to the first element inserted in the list and is still present, and the rear pointer always points to the last inserted element. |
| Stack is used in solving problems works on recursion. | Queue is used in solving problems having sequential processing. |



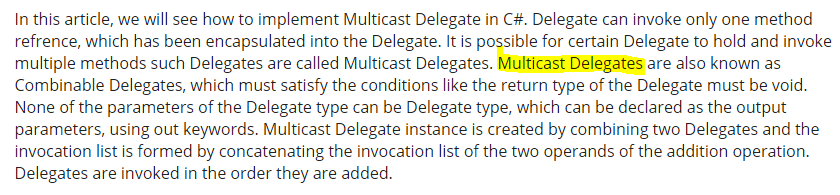


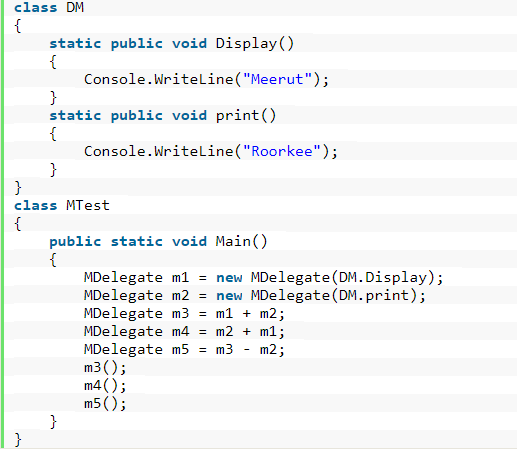


Thread Safe Singleton :

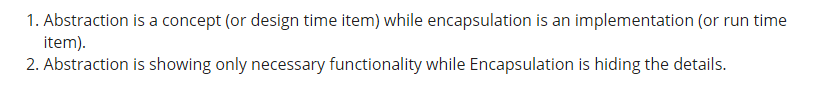


**Multi Cast Delegate**

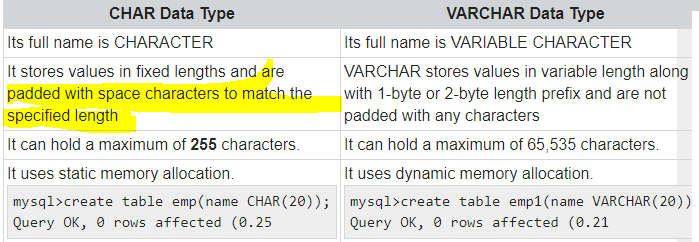




Exception handling in C#



<https://www.c-sharpcorner.com/article/abstraction-vs-encapsulation-in-oops/>



## \*\*\*Differences between a task and a thread.

1. The Thread class is used for creating and manipulating a [thread](http://msdn.microsoft.com/en-us/library/windows/desktop/ms684841%28v=vs.85%29.aspx) in Windows. A [Task](http://msdn.microsoft.com/en-us/library/vstudio/system.threading.tasks.task) represents some asynchronous operation and is part of the [Task Parallel Library](http://msdn.microsoft.com/en-us/library/dd460717%28v=vs.110%29.aspx), a set of APIs for running tasks asynchronously and in parallel.
2. The task can return a result. There is no direct mechanism to return the result from a thread.
3. Task supports cancellation through the use of cancellation tokens. But Thread doesn't.
4. ***A task can have multiple processes happening at the same time. Threads can only have one task running at a time.***
5. We can easily implement Asynchronous using ’async’ and ‘await’ keywords.
6. A new Thread()is not dealing with Thread pool thread, whereas Task does use thread pool thread.
7. A Task is a higher level concept than Thread.

*Is thread uses multicore ? no, its one task running at a time with context swtching*

IN testing /TDD considerring Dependency Injection in class to be tested is very important factor

## Sql peformance Improvements

<https://www.c-sharpcorner.com/UploadFile/ff2f08/tips-to-improve-sql-database-performance/>

<https://www.c-sharpcorner.com/UploadFile/f0b2ed/tips-to-increase-sql-server-query-performance-part-1/>

<https://www.c-sharpcorner.com/UploadFile/f0b2ed/tips-to-increase-sql-server-query-performance-part-2/>

## Preventing Sql Injection

1. Implement strong server side validation for all user inputs including cookie values.
2. Escape or filter the special characters in user inputs.
3. Use store procedures whenever possible.
4. Use parameterized queries or ORM-Entity Framework.
5. Avoid building SQL statements either in a class file or inside a procedure.
6. Avoid using exec command in SQL Server.
7. Avoid using sa account to connect database from the application.
8. Use low privileged account to execute queries.
9. Configure generic error page for the application and don’t display error information to user.
10. Catch all possible exceptions, implement global exception handler.

https://www.codeproject.com/Tips/706692/Preventing-SQL-Injection-Attacks-2

## ASP.NET performance improvement

<https://stackify.com/asp-net-performance-tuning/>

## Angular Performance Improvement

<https://www.simform.com/angular-performance/>

<https://medium.com/swlh/angular-performance-optimization-techniques-5b7ca0808f8b>

## token expiration and refresh web api angular 8

<https://jasonwatmore.com/post/2020/05/22/angular-9-jwt-authentication-with-refresh-tokens>

<https://www.codemag.com/article/1809031/Security-in-Angular-Part-2>

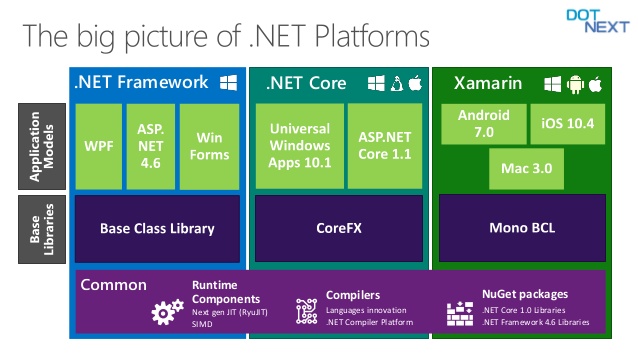
## MVC.net core life cycle

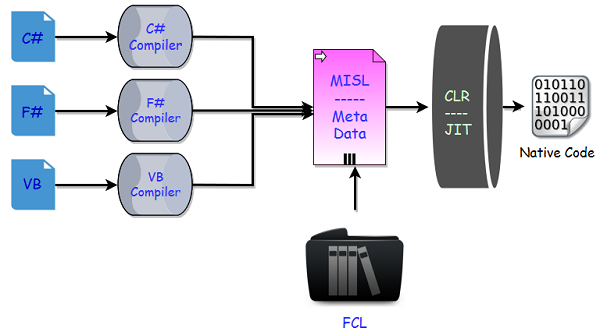
<https://www.c-sharpcorner.com/article/asp-net-core-mvc-request-life-cycle/>

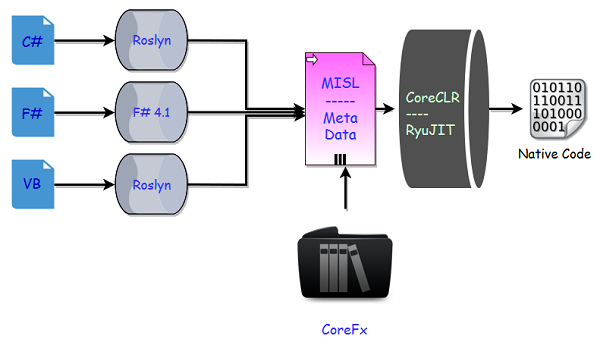
.net execution

# ASP.NET CORE and WEB API





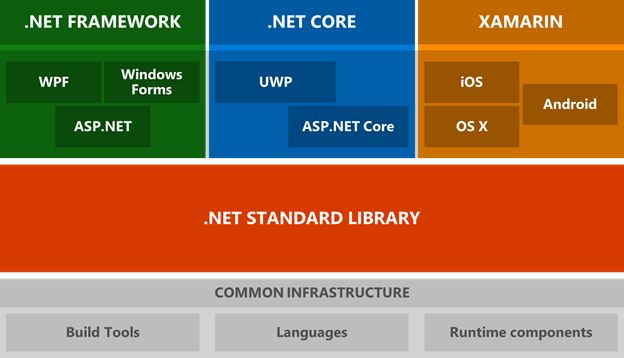




* In .NET Core now we have a new series of compilers, like we have **Roslyn** for C# and VB.
* You can also make use of the new F# 4.1 compiler if you want to use F# with .NET Core.
* Actually these tools are different and we can use Roslyn with .NET Framework as well if we are using C# 6 or later, because C# compiler can only support up to C# 5.
* In .NET Core, we don’t have a framework class libraries (FCL), so a different set of libraries are used and we now have **CoreFx**.
* CoreFx is the reimplementation of the class libraries for .NET Core.
* We also have a new run time with .NET Core known as CoreCLR and leverages a JIT Compiler.
* Now the question is why do we have the reimplementation of all these components that we already have in .NET framework.
* So the answer is the same as why Microsoft implemented .NET Core

.net core v/s .net framework

<https://www.c-sharpcorner.com/article/difference-between-net-framework-and-net-core/>



## The .NET Framework

supports Windows and Web applications. Today, you can use Windows Forms, WPF, and UWP to build Windows applications in .NET Framework. ASP.NET MVC is used to build Web applications in .NET Framework.

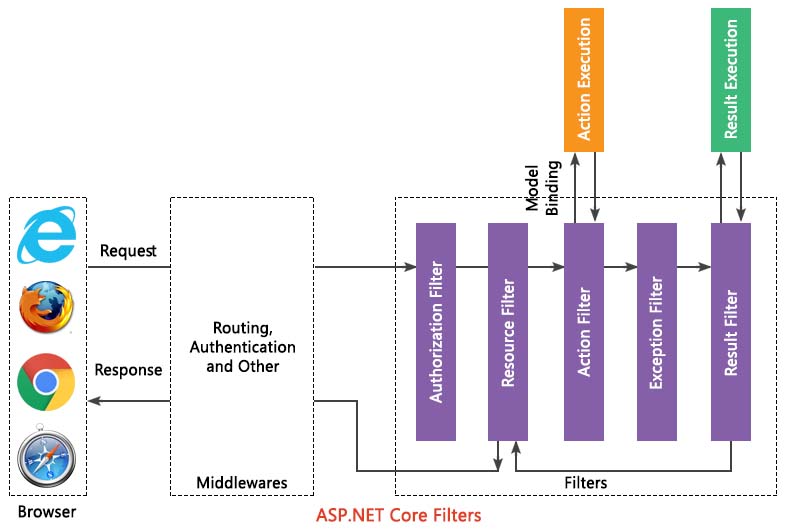
## .NET Core

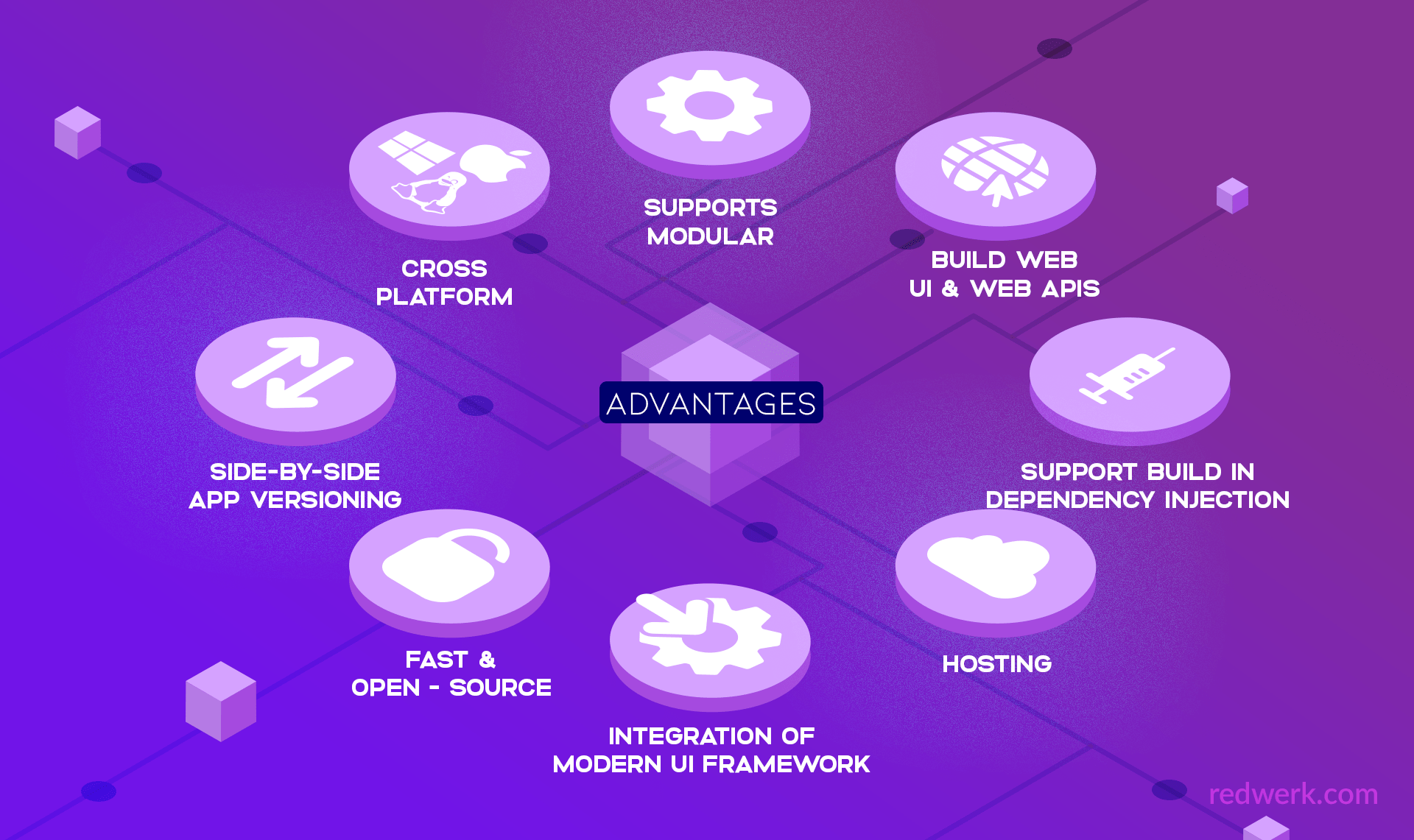
is the new open-source and cross-platform framework to build applications for all operating systems including Windows, Mac, and Linux. .NET Core supports UWP and ASP.NET Core only. UWP is used to build Windows 10 targets Windows and mobile applications. ASP.NET Core is used to build browser-based web applications.

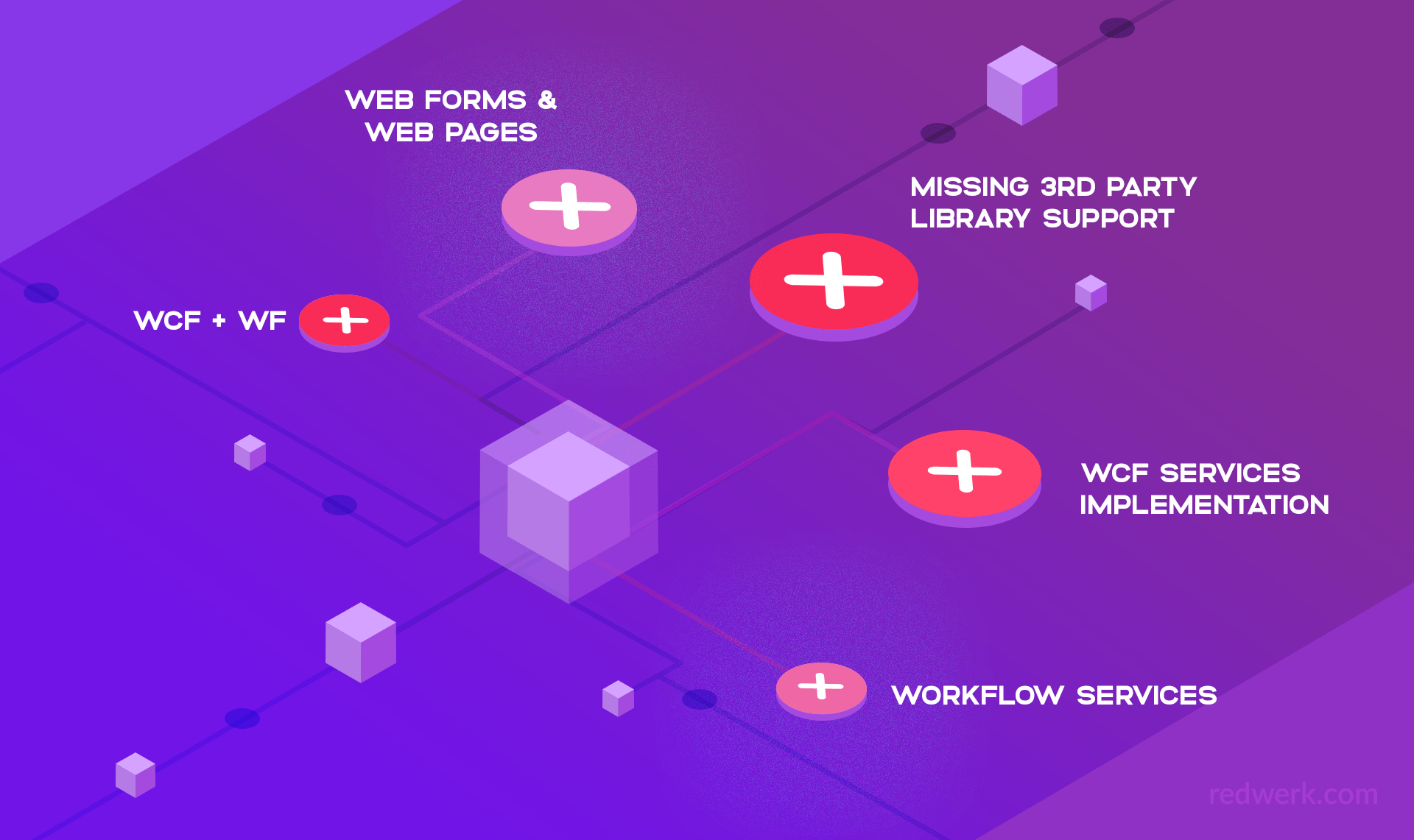
The following table may help you make your decision.

|  |  |
| --- | --- |
| A high-performance and scalable system without UI | .NET Core is much faster. |
| Docker containers support | Both, but .NET Core is born to live in a container. |
| Heavily rely on the command line | .NET Core has better support. |
| Cross-platform needs | .NET Core |
| Using Microservices | Both, but .NET Core is designed to keep today's needs in mind. |
| User interface centric Web applications | .NET Framework is better now until .NET Core catches up. |
| Windows client applications using Windows Forms and WPF | .NET Framework |
| Already have a pre-configured environment and systems | .NET Framework is better. |
| Stable version for an immediate need to build and deploy | .NET Framework has been around since 2001. .NET Core is just a baby. |
| Have existing experienced .NET team | .NET Core has a learning curve. |
| Time is not a problem. Experiments are acceptable. No rush to deployment. | .NET Core is the future of .NET. |

Web api filters







.net core entry point program.cs => starts startup.cs

// Entry point for the application.

public static void Main(string[] args) => WebApplication.Run<Startup>(args);

public class Startup {

public Startup() {

var builder = new ConfigurationBuilder()

.AddJsonFile("AppSettings.json");

Configuration = builder.Build();

}

public IConfiguration Configuration { get; set; }

**// 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 http://go.microsoft.com/fwlink/?LinkID=398940

public void **ConfigureServices**(IServiceCollection services) {

}

**// MIDDLEWARE**

**// This method gets called by the runtime.**

**// Use this method to configure the HTTP request pipeline.**

public void **Configure**(IApplicationBuilder app) {

app.UseIISPlatformHandler();

app.Run(async (context) => {

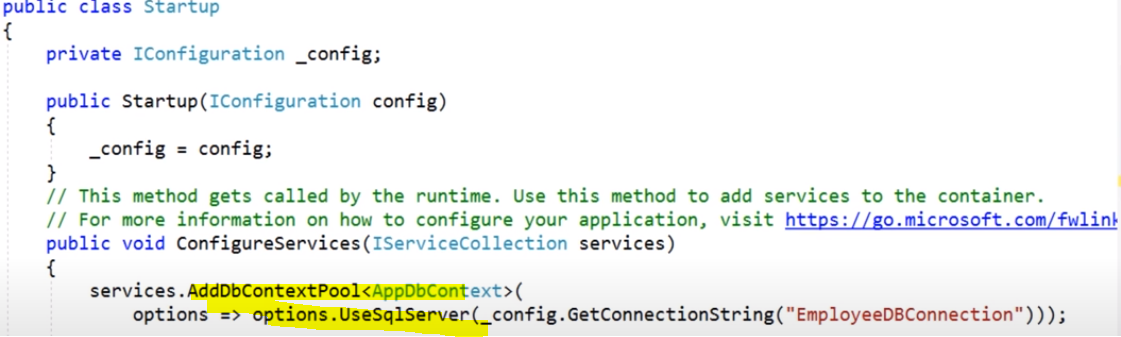
var msg = Configuration["message"];

await context.Response.WriteAsync(msg);

});

}

Db connection :



What are the features provided by ASP.NET Core?

* *Built-in supports for* [*Dependency Injection*](https://www.dotnettricks.com/learn/dependencyinjection)
* Built-in supports for the logging framework and it can be extensible
* Introduced new, fast and cross-platform web server - **Kestrel**. So, a web application can run without IIS, Apache, and Nginx.
* Multiple hosting ways are supported
* It supports modularity, so the developer needs to include the module required by the application. However, [.NET Core framework is also providing the meta package that includes the libraries](https://www.dotnettricks.com/learn/netcore)
* Command-line supports to create, build and run the application
* *There is no web.config file. We can store the custom configuration into an appsettings.json file*
* There is no Global.asax file. We can now register and use the services into startup class
* It has good support for asynchronous programming
* Support WebSocket and SignalR
* Provide protection against CSRF (Cross-Site Request Forgery)

## What is Metapackages?

The framework .NET Core 2.0 introduced Metapackage that includes all the supported package by ASP.NET code with their dependencies into one package. It helps us to do fast development as we don't require to include the individual ASP.NET Core packages. The assembly **Microsoft.AspNetCore.All** is a meta package provide by ASP.NET core.

 It is known that Microsoft.AspNetCore package .By adding this package to your project, you bring in all the relevant packages along with their dlls on which it depends and it is called a metapackage.

Specifically, the packages it lists are −

Microsoft.AspNetCore.Diagnostics

Microsoft.AspNetCore.Hosting

Microsoft.AspNetCore.Routing

Microsoft.AspNetCore.Server.IISIntegration

Microsoft.AspNetCore.Server.Kestrel

Microsoft.Extensions.Configuration.EnvironmentVariables

Microsoft.Extensions.Configuration.FileExtensions

Microsoft.Extensions.Configuration.Json

Microsoft.Extensions.Logging

Microsoft.Extensions.Logging.Console

Microsoft.Extensions.Options.ConfigurationExtensions

NETStandard.Library

## What is the startup class in ASP.NET core?

Startup class is the entry point of the ASP.NET Core application. Every .NET Core application must have this class. This class contains the application configuration rated items. It is not necessary that class name must "Startup", it can be anything, we can configure startup class in Program class.

## What is the use of ConfigureServices method of startup class?

This is an ***optional method*** of startup class. It can be used to configure the services that are used by the application. This method calls first when the application is requested for the first time. Using this method, ***we can add the services to the DI container, so services are available as a dependency in controller constructor*.**

## What is the use of the Configure method of startup class?

It defines how the application ***will respond to each HTTP request***. We can configure the request pipeline by configuring the middleware. It accepts IApplicationBuilder as a parameter and also it has two optional parameters: IHostingEnvironment and ILoggerFactory. Using this method, we can configure built-in [middleware](https://www.dotnettricks.com/learn/aspnetcore/middleware-custom-pipeline) such as routing, authentication, session, etc. as well as third-party middleware.

## How to enable Session in ASP.NET Core?

The middleware for the session is provided by the package Microsoft.AspNetCore.Session. To use the session in ASP.NET Core application, we need to add this package to csproj file and add the Session middleware to ASP.NET Core request pipeline.



## What are the various JSON files available in ASP.NET Core?

There are following JSON files in ASP.NET Core :

* + global.json
  + launchsettings.json
  + appsettings.json
  + bundleconfig.json
  + bower.json
  + package.json

## How to specify service lifetime for register service that added as a dependency?

ASP.NET Core allows us to specify the lifetime for registered services. The service instance gets disposed of automatically based on a specified lifetime. So, we do not care about the cleaning these dependencies, it will take care by ASP.NET Core framework. There is three type of lifetimes.

* [**Singleton**](https://www.qfles.com/interview-question/csharp-design-patterns-interview-questions) - Service with singleton lifetime is created once when first time the service is requested. For subsequent requests same instance is served by service containe
* **Transient** - Services with transient lifetime are created *each time they are requested* *from service container*. So it's best suited for stateless, light weight services.
* **Scoped** - Services with scoped lifetime are created once *per connection or per client http request*. When using scoped service in middleware then inject the service via invoke or invokeAsync method. You should not inject the service via constructor injection as it treats the service behavior like Singleton.

### Singleton

The service can be added as a singleton using AddSingleton method of IServiceCollection. ASP.NET Core creates service instance at the time of registration and subsequence request use this service instance. Here, we do not require to implement Singleton design pattern and single instance maintained by the ASP.NET Core itself.

**Example**

services.AddSingleton<IHelloWorldService, HelloWorldService>();

### Transient

The service can be added as Transient using AddTransient method of IServiceCollection. This lifetime can be used in stateless service. It is a way to add lightweight service.

**Example**

services.AddTransient<IHelloWorldService, HelloWorldService>();

### Scoped

The service can be added as scoped using an AddScoped method of IServiceCollection. *We need to take care while, service registered via Scoped in middleware and inject the service in the Invoke or InvokeAsync methods.* If we inject dependency via the constructor, it behaves like singleton object.

services.AddScoped<IHelloWorldService, HelloWorldService>();

**Middleware component** is program that's build into an app's pipeline to handle the request and response. Each middleware component can decide whether to pass the request to next component and to perform any operation before or after next component in pipelin

**Kestrel** :  
ASP.NET Core use the **Kestrel** web server by default. ASP.NET Core comes with:Default Kestrel web server that's cross platform HTTP server implementation.

 How Configuration works in ASP.NET Core?

In ASP.NET Core **Configuration** is implemented using various configuration providers. Configuration data is present in the form of key value pairs that can be read by configuration providers as key value from different configuration sources as below.

* appsettings.json - settings file
* Azure Key Vault
* Environment variables
* In-memory .Net objects
* Command Line Arguments
* Custom Providers
* How to read values from Appsettings.json file?
* You can read values from appsettings.json using below code.
* class Test{
* // requires using Microsoft.Extensions.Configuration;
* private readonly IConfiguration Configuration;
* public TestModel(IConfiguration configuration)
* {
* Configuration = configuration;
* }
* // public void ReadValues(){
* var val = Configuration["key"]; // reading direct key values
* var name = Configuration["Employee:Name"]; // read complex values
* }
* }



# Prevent Cross-Site Scripting (XSS)

Cross-Site Scripting (XSS) is a security vulnerability which enables an attacker to place client side scripts (usually JavaScript) into web pages. When other users load affected pages the attacker's scripts will run, enabling the attacker to steal cookies and session tokens, change the contents of the web page through DOM manipulation or redirect the browser to another page. XSS vulnerabilities generally occur when an application takes user input and outputs it to a page without validating, encoding or escaping it.

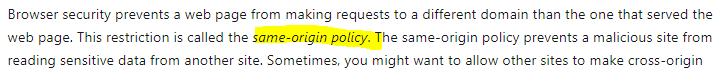
## Best Practices against Cross Site Scripting Attacks

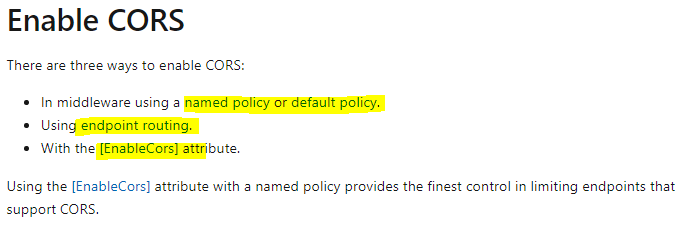
Preventing XSS attacks is pretty simple if you follow these best practices:

* Validate every user input, either reject or sanitize unknown character, for example, < or > which can be used to create
* Test every input from an external source
* Use HttpOnly for cookies so it is not readable by Javascript (therefore an attacker can’t use Javascript to read your cookies)
* Use markdown instead of HTML editors

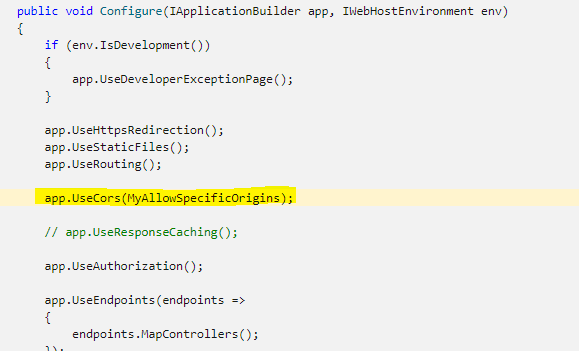
CORS :



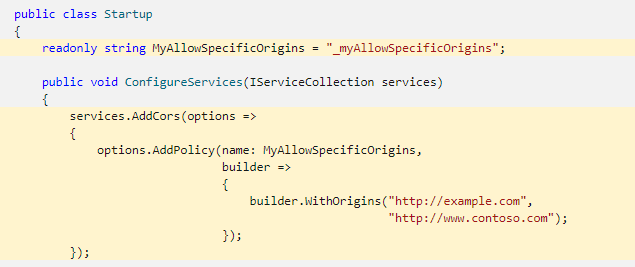








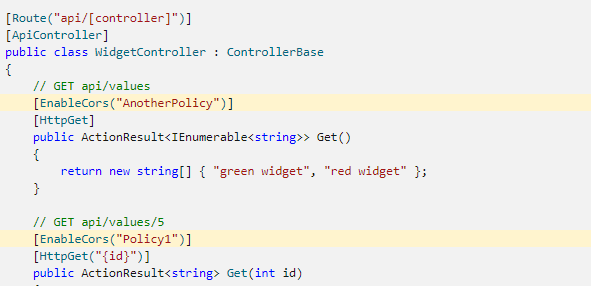
## End point routing





## Enable CORS with attributes

* [EnableCors] specifies the default policy.
* [EnableCors("{Policy String}")] specifies a named policy.

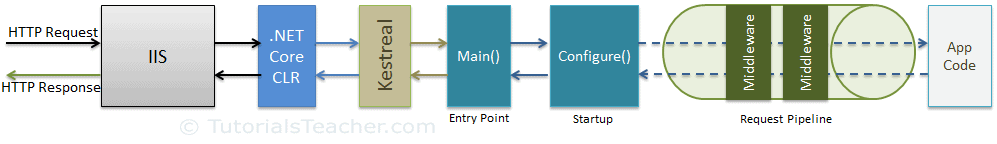


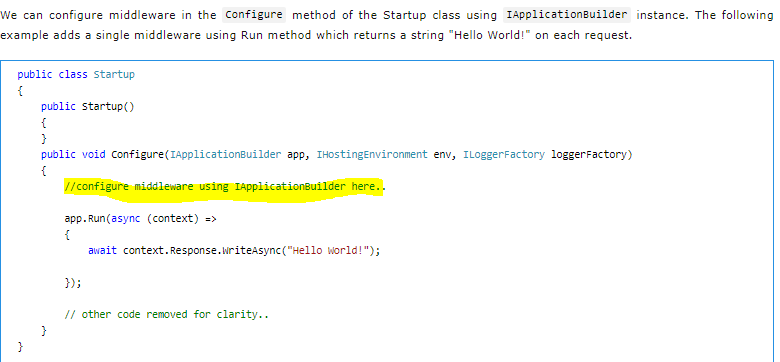
Swagger : Install-Package Swashbuckle.AspNetCore -Version 5.6.3

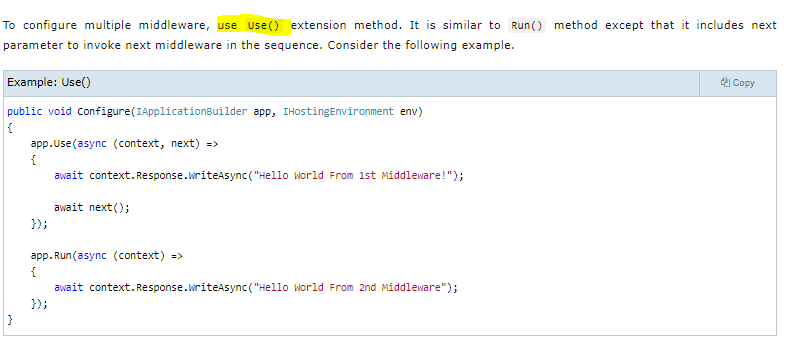


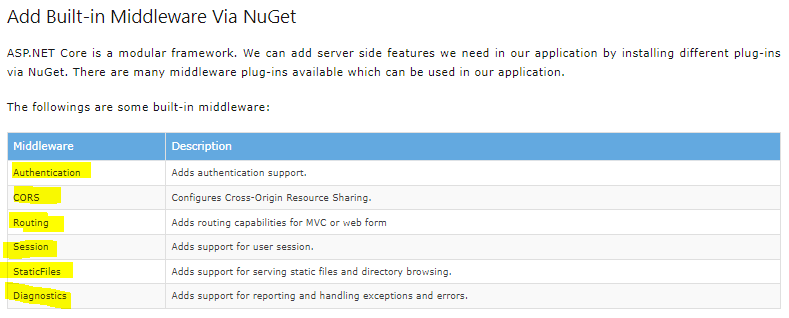
# ASP.NET Core - Middleware

ASP.NET Core introduced a new concept called **Middleware.** A middleware is nothing but a component (class) which is executed on every request in ASP.NET Core application. In the classic ASP.NET, HttpHandlers and HttpModules were part of request pipeline. Middleware is similar to HttpHandlers and HttpModules where both needs to be configured and executed in each request.









Repository Pattern:

Cofee/tea machine desighn pattren C#

ASP.NET Core 3.1 API - JWT Authentication with Refresh Tokens

#### Does ViewState affect performance? What is the ideal size of a ViewState? How can you compress a viewstate?

Viewstate stores the state of controls in HTML hidden fields. At times, this information can grow in size. This does affect the overall responsiveness of the page, thereby affecting performance. The ideal size of a viewstate should be not more than 25-30% of the page size. Viewstate can be compressed to almost 50% of its size. .NET also provides the GZipStream orDeflateStream to compress viewstate.

### Throttling ASP.NET Web API calls

Many API’s out there, such as [GitHub’s API](https://developer.github.com/v3/#rate-limiting), have a concept called “rate limiting” or “throttling” in place. Rate limiting is used to prevent clients from issuing too many requests over a short amount of time to your API. For example, we can limit anonymous API clients to a maximum of 60 requests per hour whereas we can allow more requests to authenticated clients. But how can we implement this

I would prefer plugging it earlier in the pipeline. Luckily for us, ASP.NET Web API also provides the concept of [message handlers](https://www.asp.net/web-api/overview/working-with-http/http-message-handlers). They accept an HTTP request and return an HTTP response and plug into the pipeline quite early.

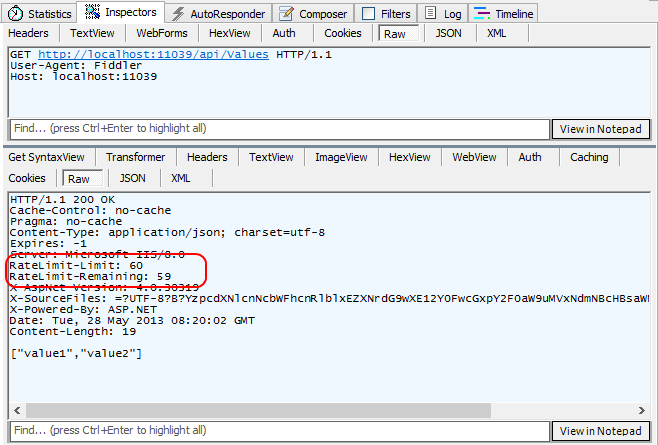
The easiest way of using the *ThrottlingHandler* is by registering it using simple parameters like the following

config.MessageHandlers.Add(new ThrottlingHandler(

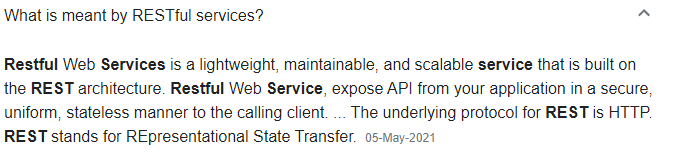
2 new InMemoryThrottleStore(),

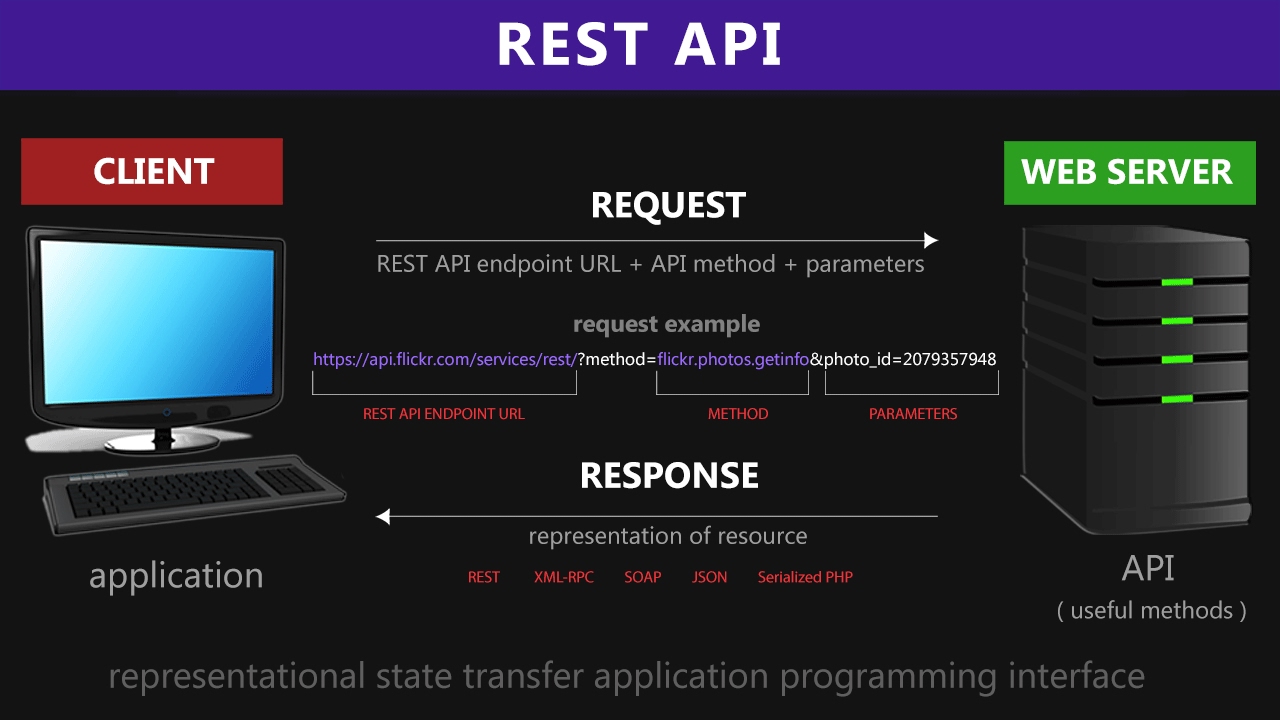
3 id => 60,

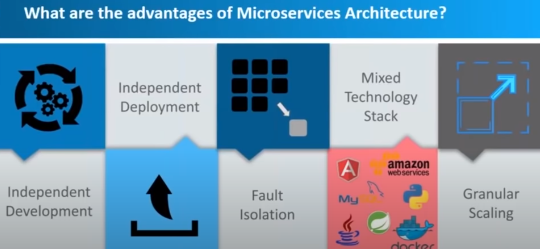
4 TimeSpan.FromHours(1)));



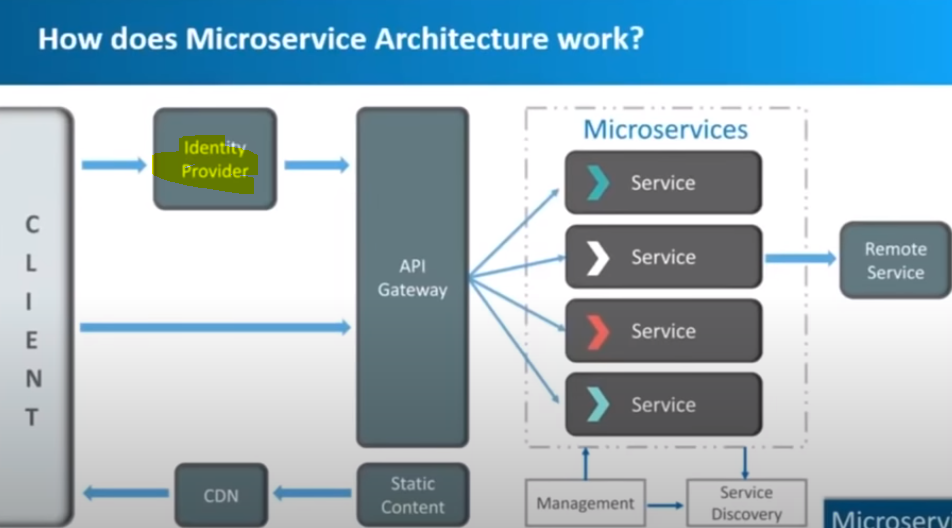
# Micro Services

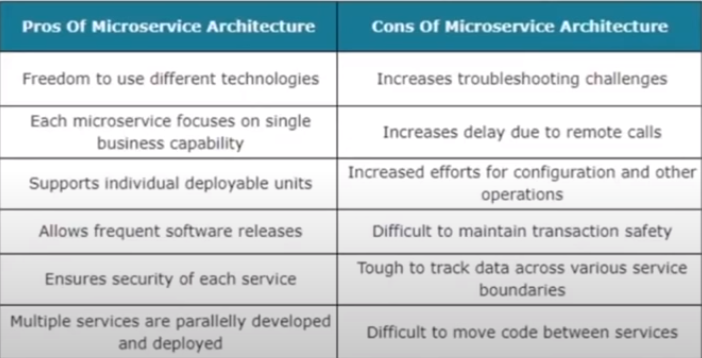






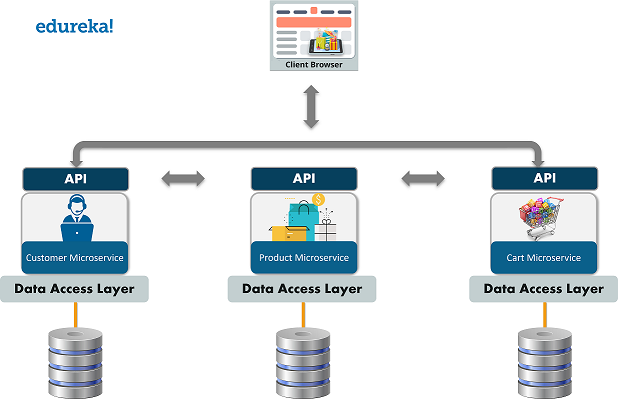












## how do microservices communicate with each other

Defining the communication mechanism for the micro-services is the main challenge while developing the architecture design for an application. Micro-services design depends on business aspects of a system, however communication links is technical aspect for that particular system.  
  
There are two modes by which micro services use to communicate with each other which is based on the receiver configuration whether it is single or multiple:

**Synchronous**:  
In this approach, client sends a request and waits for the response from a particular service. Multiple services are communicating with each other through HTTP sync.  
  
**Asynchronous**:  
This kind of communication is established with the help of message broker queues which flows the messages flowing from one service to another service. It depends on the concept of message flowing through the services.

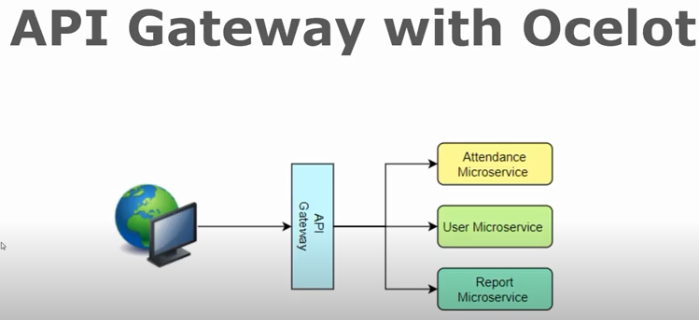
Message queues involved two components i.e Message produce and message consumer.

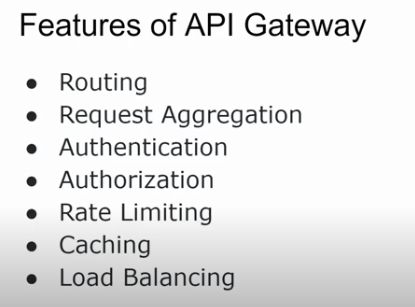
**Message producer:** It is responsible for sending out the requests and data and just waits for acknowledgement of data reception from consumer ones only.  
One to one communication is handled through queues and One to many communication is handled through Topics.  
  
Popular Message brokers used recently are RabbitMQ, Kafka etc. Software developers use these brokers as per their convenience.

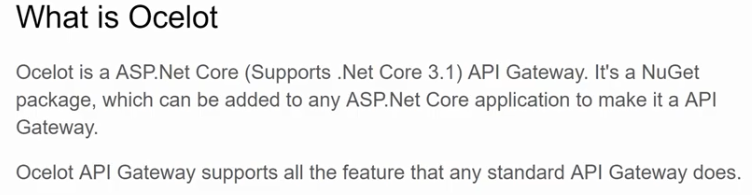
<https://blog.logrocket.com/methods-for-microservice-communication/>

## API Gateway

API Gateway is an API management tool that usually sits between the external caller (Web or Mobile) and the internal services. The API Gateway can provide multiple features like: 1. Routing 2. Request Aggregation 3. Authentication 4. Authorization 5. Rate Limiting 6. Caching 7. Load Balancing



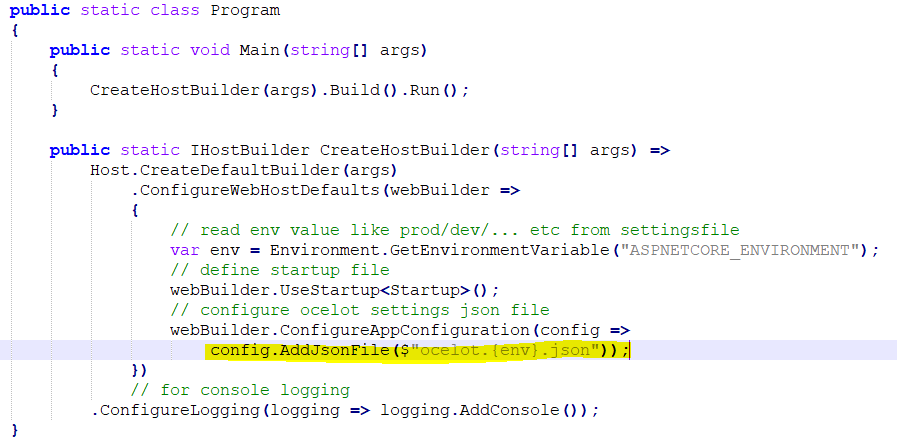




<https://ocelot.readthedocs.io/>

Ocelot is an ASP.Net Core (Supports .Net Core 3.1) API Gateway. It's a NuGet package, which can be added to any ASP.Net Core application to make it an API Gateway. Ocelot API Gateway supports all the features that any standard API Gateway does. I will cover the following features of Ocelot API Gateway in this video: 1. Routing (Basic routing to internal service) 2. Authentication (JWT Token-based authentication) 3. Response Caching (Using Ocelot.Cache.CacheManager NuGet package) 4. Rate limiting

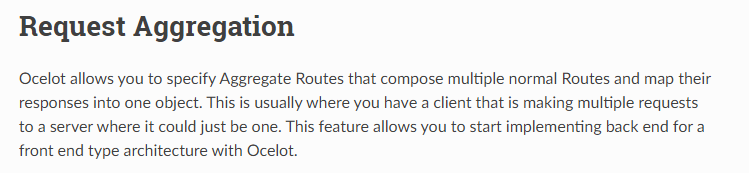


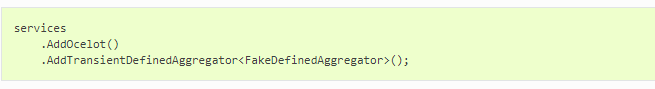


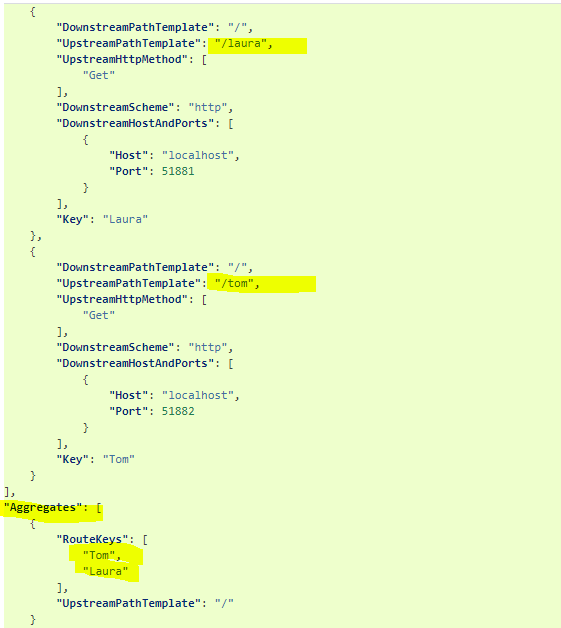




# 







## API backward compatibility and versioning example

 In other words, an API is backward compatible between releases if the clients should are able to work with a new version of the API seamlessly.1) Optional Params 2) versioning 3) Build small curd operations and agrregation on client side 4) canary deployment and feature toggle to safe guard in case of issues

* 1. Using optionalparams

1. [**HttpGet**]  
   [**Route**("GetOrders")]  
    **public** **IActionResult** **GetOrders**(**int** customerId, **int** orderId = 0)  
    {  
      **var** result = \_orderService.**GetOrdersForCustomer**(  
                    customerId, orderId);  
      **return** **Ok**(result);  
    }

how do we design our APIs the recommended way? How do we ensure that our RESTful API is backward compatible?

### Make sure that the unit tests pass Never change the behavior of HTTP response codes Never change parameters Version your API

### The robustness principle

* Every API endpoint should have a small, specific goal that follows only one of the [CRUD](https://en.wikipedia.org/wiki/Create,_read,_update_and_delete) operations. Clients should be in charge of aggregating multiple calls as needed.
* Servers should communicate expected message formats and schemas and adhere to them.
* New or unknown fields in message bodies should not cause errors in APIs, they should just be ignored.

[Versioning an API](https://restfulapi.net/versioning/) allows us to support different functionality for the same resource.

### Message3 Bus : Use headers and filters

Most message buses offer message headers. Just like HTTP headers, this is a great way to pass metadata without polluting the message payload. We can use this to our advantage in multiple ways. Just like with web APIs, we can publish messages with version information in the header

### Canary deployment

A canary deployment, also known as a blue/green, red/black, or purple/red deployment, is the idea of releasing a new version of an application and only allowing a small percentage of traffic to reach it.

The goal is to test new application versions with real traffic, while minimizing the impacts of any problems that might occur. If the new application works as expected, then the remaining instances can be upgraded. If something goes wrong, the single instance can be reverted and only a small portion of traffic is impacted.

### Backward Compatible Changes

An API is backward compatible if an unchanged client can interact with a changed API. The unchanged client should be able to use all the functionality that was offered by the old API.

If a change is supposed to be backward compatible, certain changes to the API are prohibited, others are possible. The following is a list of backward-compatible changes:

• Adding query parameters (they should always be optional).

• Adding header or form parameters, as long as they are optional.

• Adding new fields in JSON or XML data structures, as long as they are optional.

• Adding endpoints, e.g. a new REST resource.

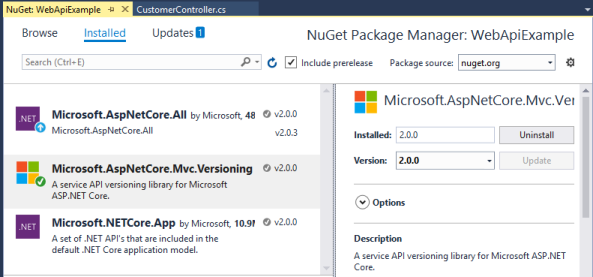
• Adding operations to an existing endpoint, e.g. when using SOAP.

• Adding optional fields to the request interfaces.

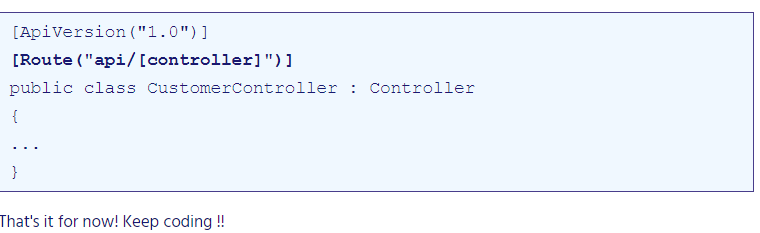
• Changing mandatory fields to optional fields in an existing API.

API Version in .core – 2 ways

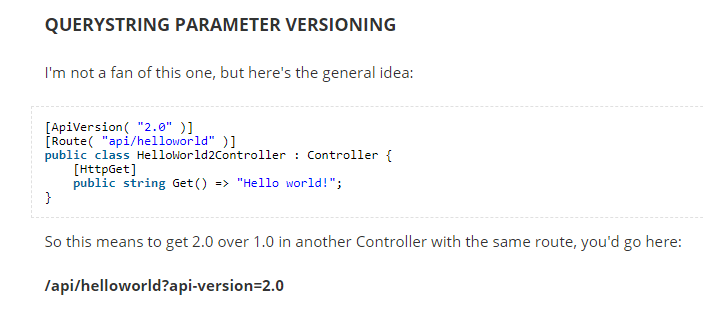
1. By attribute routing use query params  
   2. A better solution using [ApiVersion] attribute – header versioning

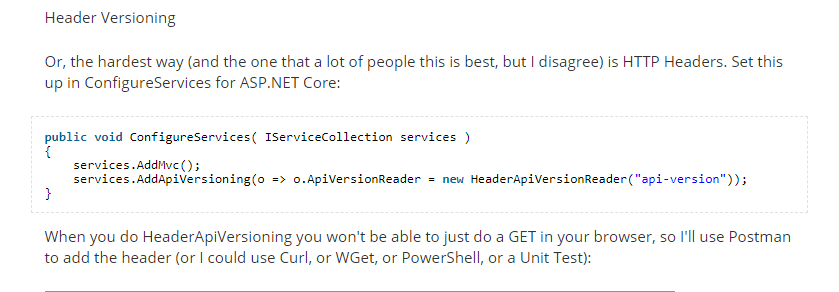


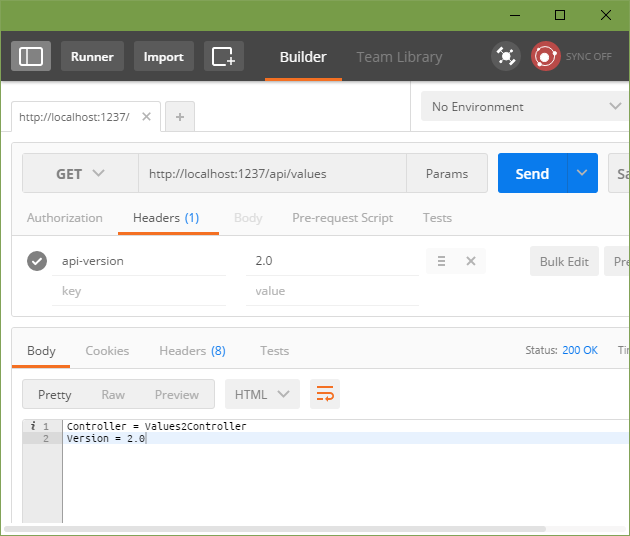




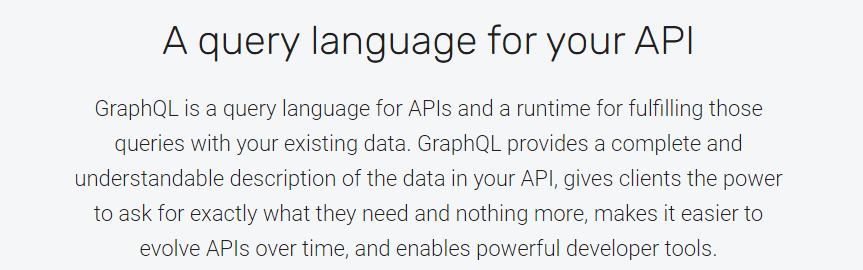












Send a GraphQL query to your API and get exactly what you need, nothing more and nothing less

## Get many resources in a single request

## Evolve your AP In without versions

**How does Web API handle large amounts of data?**

When it comes to streaming a large array of objects (for example, 10,000 objects), we are usually required to deal with two major performance issues:

1. Large memory allocation for objects
2. Long response time from server

To deal with the issues, we have two methods that can improve server side performance:

1. Iterative Pattern in C#
2. Chunked Transfer Encoding in HTTP

In the following sections, we will take a look at these methods to see how they help two issues out. We will also see two examples working on the array streaming, from server side to client side.

## Iterative Pattern in C#

It has been pretty well known that we can enable Iterative Pattern by using yield keyword within a method or property which has IEnumerable(T) return type. The idea behind the pattern is to enumerate each of the items instead of returning the whole collection.

C#

Copy Code

public IEnumerable<ReturnModel> Get()

{

*// An example of returning large number of objects*

foreach (var i in Enumerable.Range(0, 10000))

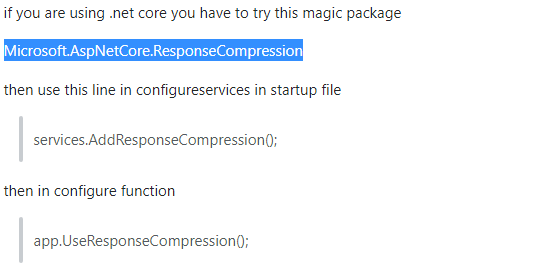
yield return new ReturnModel() { SequenceNumber = i, ID = Guid.NewGuid() };

}

Because the enumeration starts as soon as the foreach loop goes without waiting for all objects to be ready, we can expect

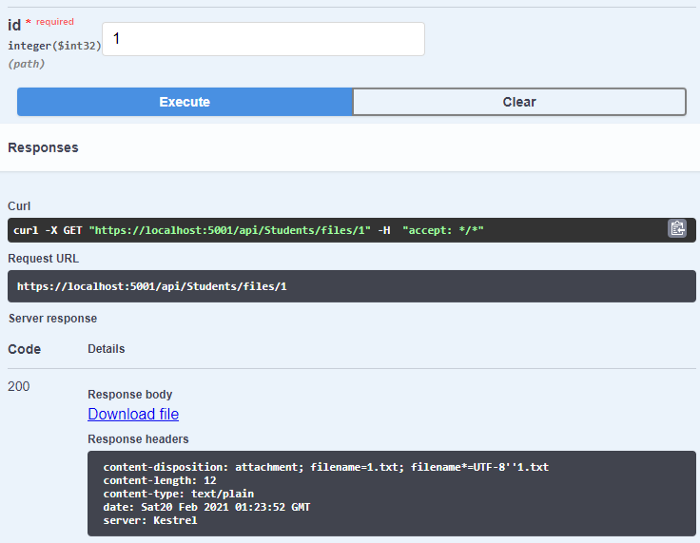
that the efficiency and memory use are better in general.



Compressing response message 

# Return a File in ASP.NET Core Web API

|  |
| --- |
| HttpGet("files/{id:int}")] |
|  | public async Task<ActionResult> DownloadFile(int id) |
|  | { |
|  | var filePath = $"{id}.txt"; // Here, you should validate the request and the existance of the file. |
|  |  |
|  | var bytes = await System.IO.File.ReadAllBytesAsync(filePath); |
|  | return File(bytes, "text/plain", Path.GetFileName(filePath)); |
|  | } |



## Micro srvices hosting deployment strategy :

<https://docs.openshift.com/container-platform/3.7/dev_guide/deployments/deployment_strategies.html>

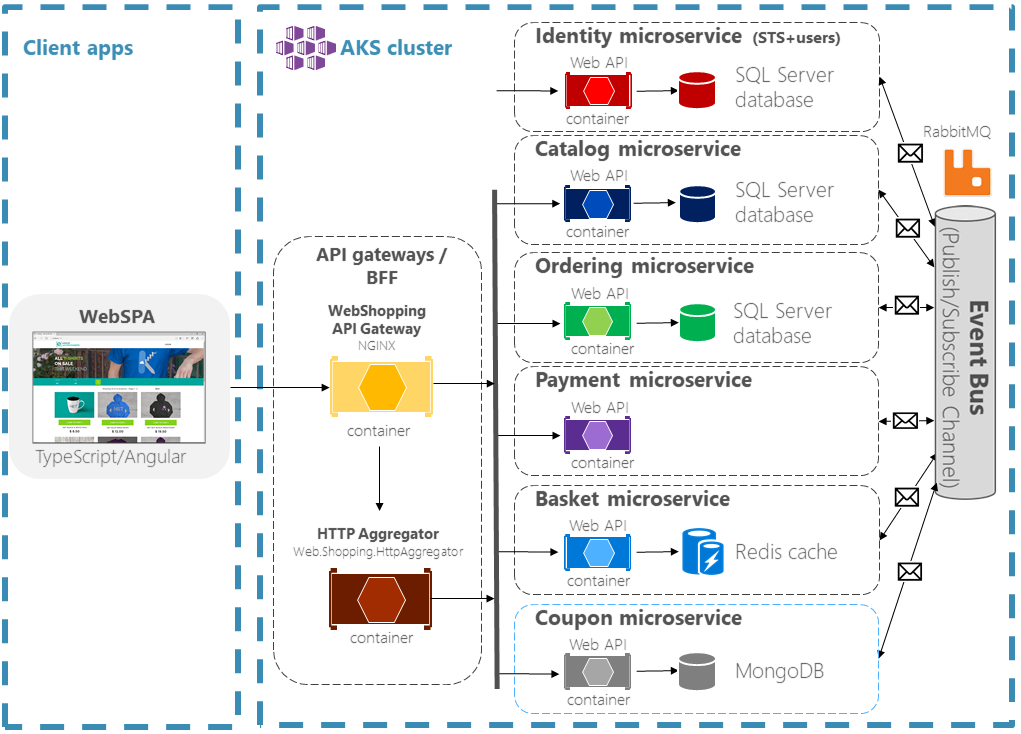
What Are Deployment Strategies?

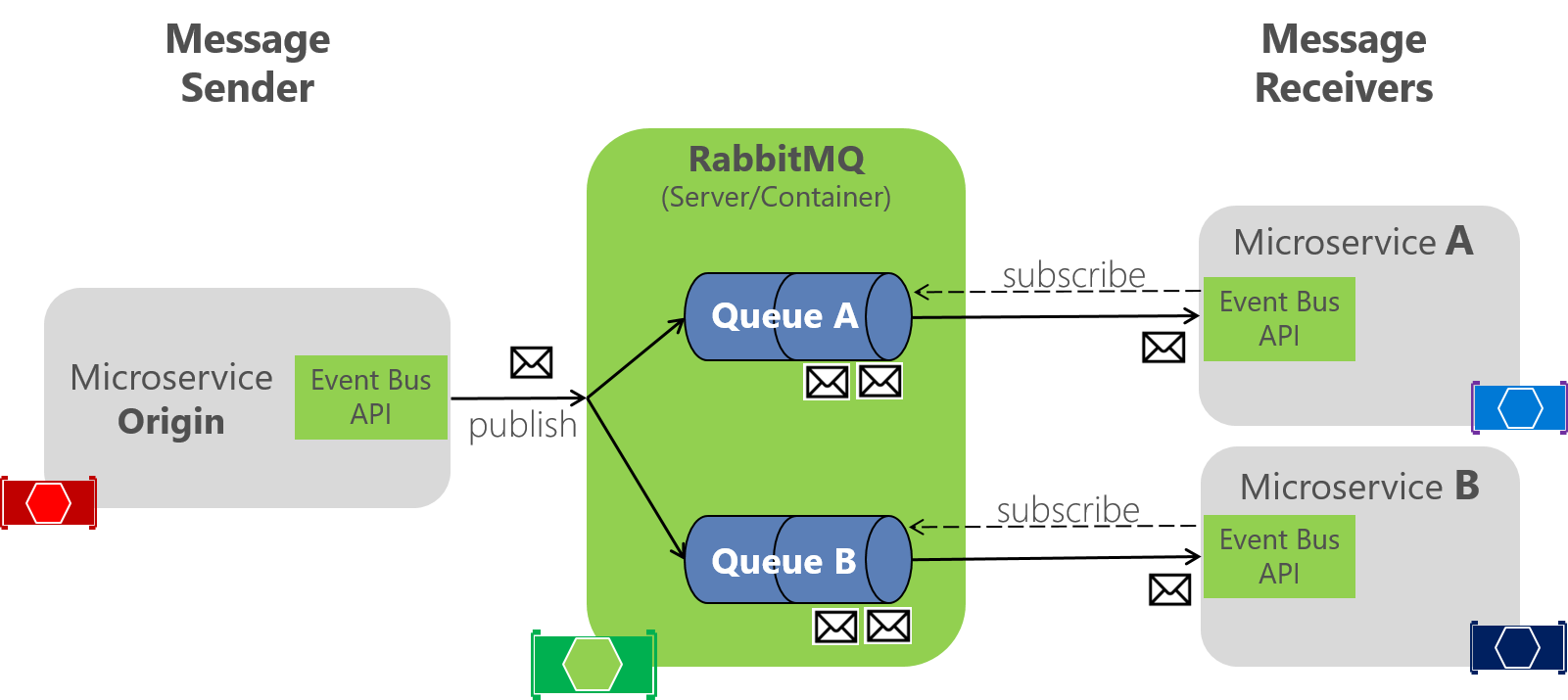
A deployment strategy is a way to change or upgrade an application. The aim is to make the change without downtime in a way that the user barely notices the improvements.

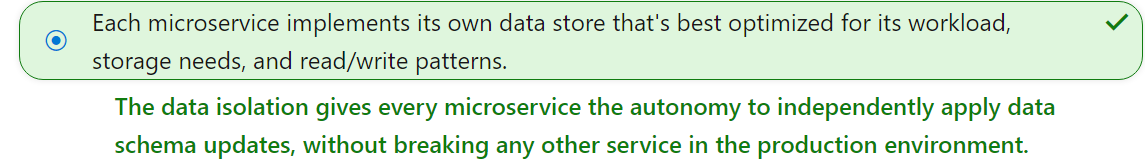
The most common strategy is to use a [blue-green deployment](https://docs.openshift.com/container-platform/3.7/dev_guide/deployments/advanced_deployment_strategies.html#advanced-deployment-strategies-blue-green-deployments). The new version (the blue version) is brought up for testing and evaluation, while the users still use the stable version (the green version). When ready, the users are switched to the blue version. If a problem arises, you can switch back to the green version.

A common alternative strategy is to use A/B versions that are both active at the same time and some users use one version, and some users use the other version. This can be used for experimenting with user interface changes and other features to get user feedback. It can also be used to verify proper operation in a production context where problems impact a limited number of users.

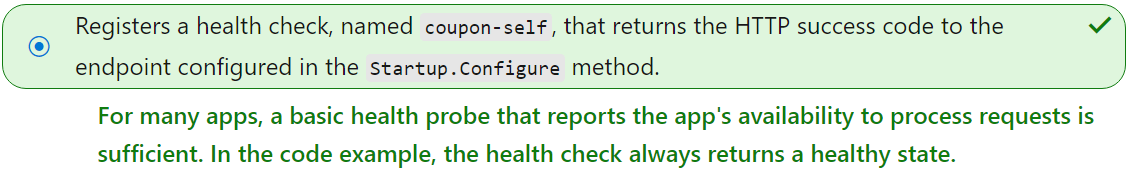
A canary deployment tests the new version but when a problem is detected it quickly falls back to the previous version. This can be done with both of the above strategies.



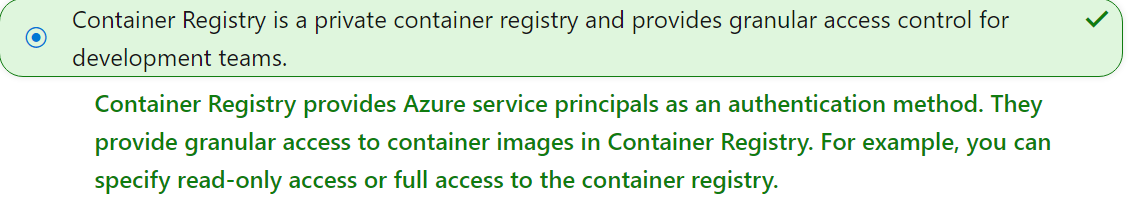
In a microservices architecture, how do individual services access their data?



2. For the ASP.NET Core Health Checks middleware, the code services.AddHealthChecks().AddCheck("coupon-self", () => HealthCheckResult.Healthy()); in Startup.ConfigureServices serves what purpose?







What is fault isolation?

## *Can I use json instead of yaml for my Compose file?*

## Yes. [Yaml is a superset of json](https://stackoverflow.com/a/1729545/444646) so any JSON file should be valid Yaml. To use a JSON file with Compose, specify the filename to use, for

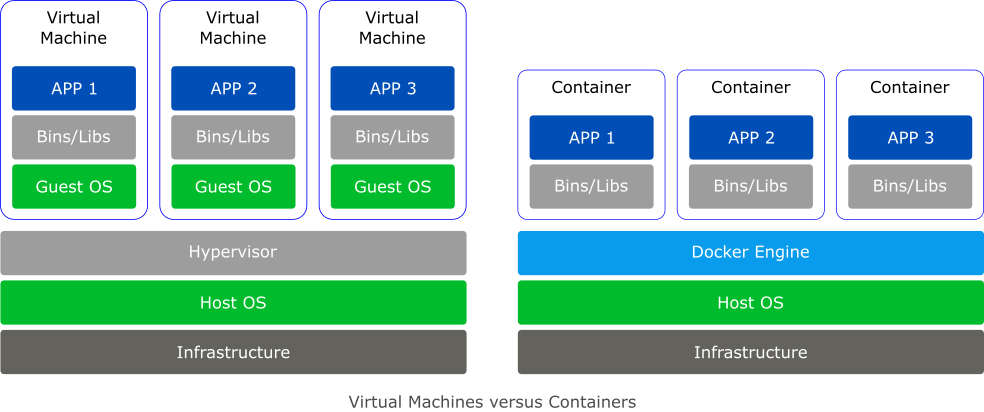
### [Deploying C# Web Applications with Docker](https://platform.uno/blog/deploying-c-web-applications-with-docker/)

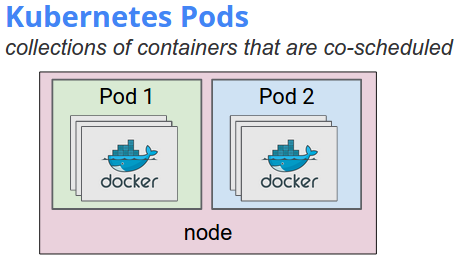
<https://platform.uno/blog/deploying-c-web-applications-with-docker/>

Docker is a type of container.

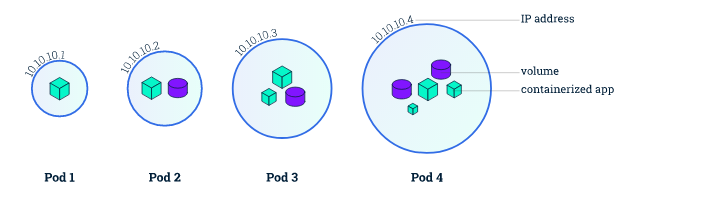
A container is a package that contains an application’s code and all of its dependencies, so that it can run quickly and reliably from one computing environment to the next.

Like a virtual machine (VM), containers are isolated and act as if they have their own file system, CPU, and RAM. Unlike a VM, however, they don’t have an OS. Instead, a container shares the kernel of the host OS.





PODSA Pod is a group of one or more application containers (such as Docker or rkt) that includes shared storage (volumes), a unique cluster IP address and information about how to run them (like container image version or specific ports). Containers within a Pod share an IP Address and port space.

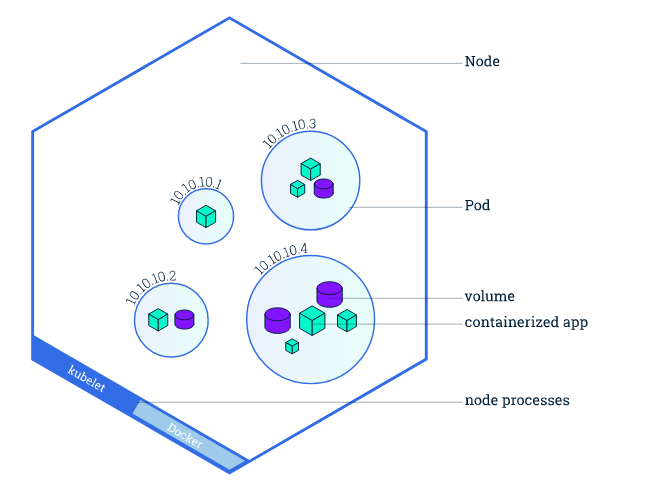


## Nodes

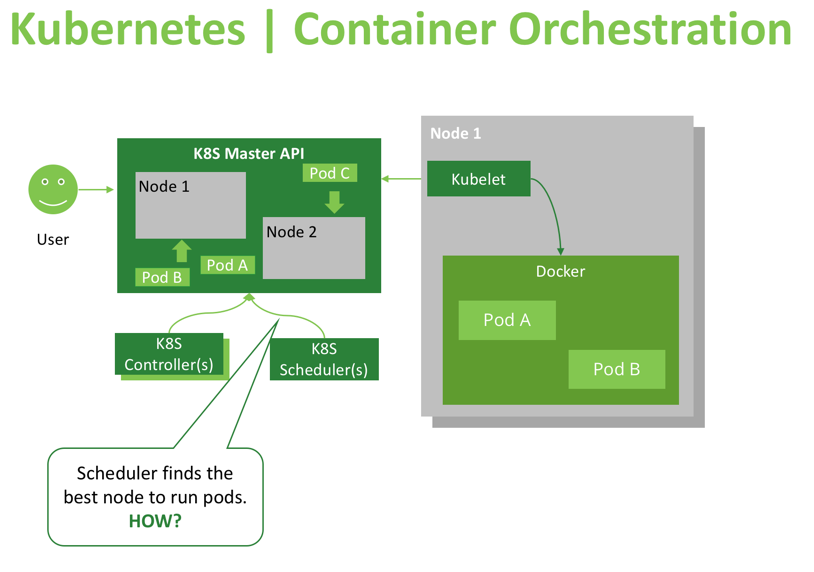
Pods always run on **Nodes**. A Node is a worker machine in Kubernetes and may be a VM or a physical machine, depending on the cluster. Each Node runs Pods and is managed by the Master.

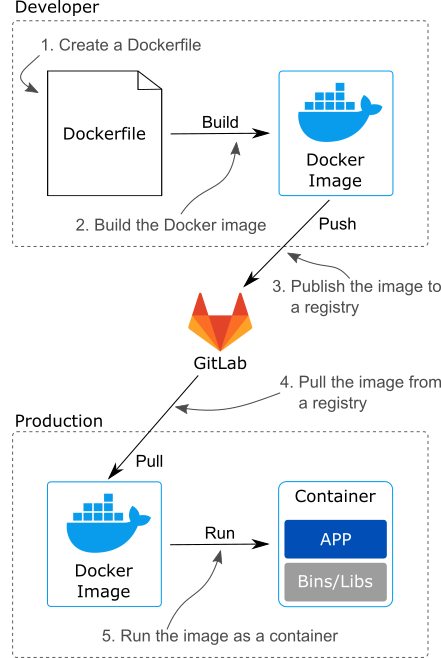
Every Kubernetes Node runs at least:

* A container runtime (like Docker, rkt) that will take care of pulling all your containers from a registry.
* Kubelet, that acts as a bridge between the Kubernetes Master and the Nodes; it manages the Pods and the containers running on a machine



* **kubectl get** - list resources
* **kubectl describe** - show detailed information about a resource
* **kubectl logs** - print the logs from a container in a pod
* **kubectl exec** - execute a command on a container in a pod





* Docker is a type of container. Containers are isolated packages that hold your code and dependencies so that they can run quickly and reliably from one computing environment to the next.
* Unlike a virtual machine, a container shares the kernel of the host OS allowing it to be smaller, faster, and use fewer resources.
* There are Linux and Windows Docker containers. To use Windows containers, you need to switch Docker Desktop to Windows containers. You also need to use Windows 10, Windows Server 2016, or Windows Server 2019 with either Hyper-V or WSL 2.
* Microsoft has four main base images (Windows Server Core, Nano Server, Windows, and Windows IoT Core) but they’ve also created several images derived from the four main ones including images with IIS, the .NET Framework, the .NET SDK, and the .NET Runtime.
* A Dockerfile is a special file that serves as a blueprint for creating a Docker image.
* A Docker image is a read-only template for creating a container. An image is built based on the instructions in a Dockerfile using the docker build command.
* The docker push command is used to push an image to a registry and docker pull is used to pull an image from a registry.