Section-1: Introduction

Part-1 : Microservice Basics

- ➔ Microservices are small, loosely coupled applications or services that can fail independently from each other. When a microservice fails, only a single function or process in the system should become available while the rest of the system remains unaffected.
- ➔ From that definition, we can conclude that the ultimate goal of a microservice is independence.

- ➔ Each microservice should believe it's the only service in the world.

- ➔ Firstly, microservices should not share code or data.

- ➔ **Dry Principle** -> Do Not Repeat Yourself

- ➔ Independence and autonomy is more important than code reusability in microservices.

- ➔ Now, I'm not saying throw the DRY principle out of the door. That's still a very good principle. With microservices, if you design it correctly, it will have a bounded context and it will have data sovereignty over its specific data.

- ➔ Another principle that we should follow is to make sure that microservices do not communicate directly with each other. Because they will be tightly coupled.

Part-2 : CQRS

CQRS is a software design pattern that stands for command and query segregation principle.

Why We Need CQRS:

- ➢ Data is often more frequently queried than altered, or vice versa.
- ➢ CQRS allows you to scale the command and query APIs independently from each other.
- ➢ This could result in fewer lock contentions, which is generally as a result of executing command and query operations on the same model.
- ➢ CQRS allows you to optimize your reads and write data schemas with the schema of the read side can be optimized for queries while the schema on the write or command side can be optimized for writes or updates.
- ➢ For example, if you store a materialized view of your data in the database, your query API won't have to do any complex joints between tables or collections.
- ➢ It allows you to separate concerns. Generally, you will find that complex business logic is applied on the write model, while the read model is usually quite simple.
- ➢ Finally, you can improve your data security by ensuring that only the relevant command API is allowed to perform write operations on the given write database or event store.

Part-3 : Event Sourcing

Event sourcing is a software design pattern that is commonly combined with CQRS. Event sourcing defines an approach where all the changes that are made to an object or entity are stored as a sequence of immutable events to an event store as opposed to just storing the current state.

Benefits Of Event Sourcing

- The event store basically contains a complete auditable log.
  In other words, all the state changes that were applied to the object or entity instead of just storing the latest or current state, as is common in traditional applications.

- The state of an object, usually the aggregate can be recreated by replaying the event store.

- It improves write performance since all events are simply appended to the event store.In other words, we never do any update or delete operations on the event store.

- In the case of failure, the event store can be used to restore the entire read database.

Part-5 : Apache Kafka

Apache Kafka is an open source event streaming platform that enables the creation of real time event driven applications.

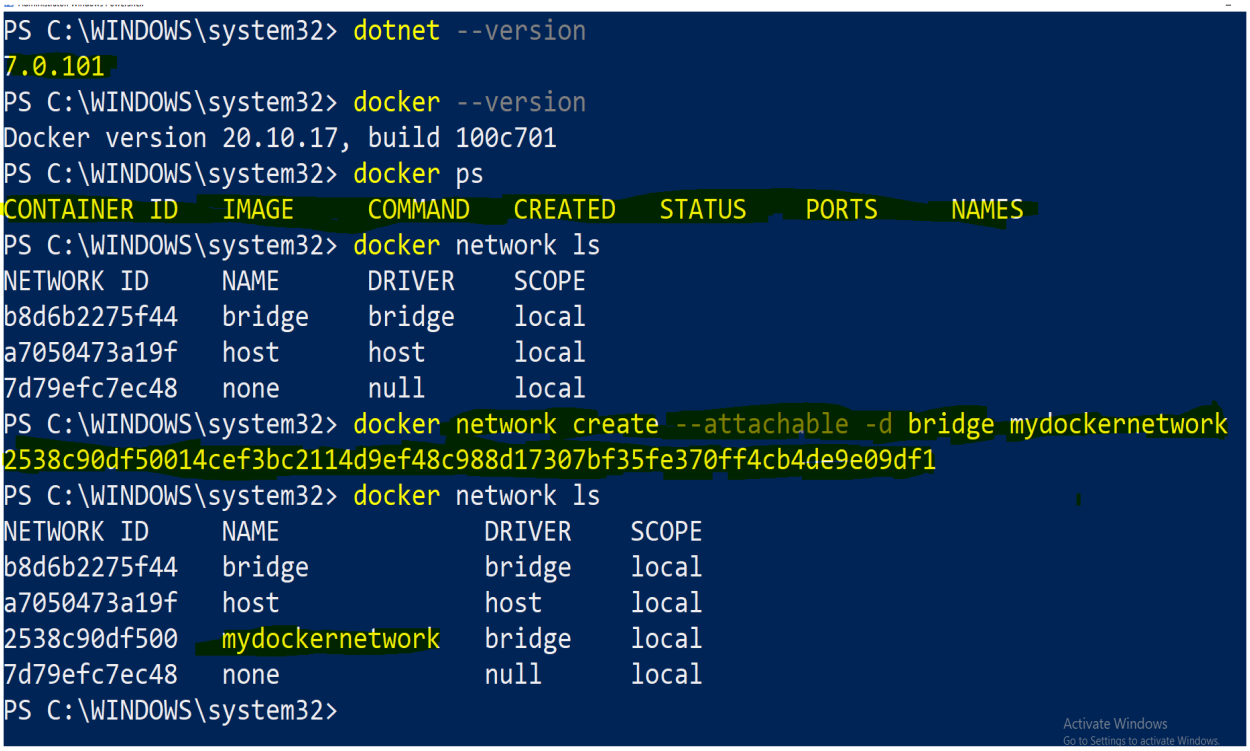
Kafka was developed by LinkedIn in 2011 as a high throughput message broker for its own internal use. It was then open sourced and donated to the Apache Foundation. Today, Kafka has evolved into the most widely used streaming platform and is capable of ingesting and processing trillions of records per day without any noticeable performance lag.

Section-2: Setup & Structure

Part-7 : Prerequisites

But as you can see, there are no containers to deploy to Docker yet.
We are now going to run a few of our prerequisites as Docker containers instead of installing them directly onto our computers in the traditional fashion.
But before we do that, let's create a Docker network which will enable our microservices to communicate with Apache, Kafka, MongoDB and Microsoft SQL Server, which we are going to install as Docker containers

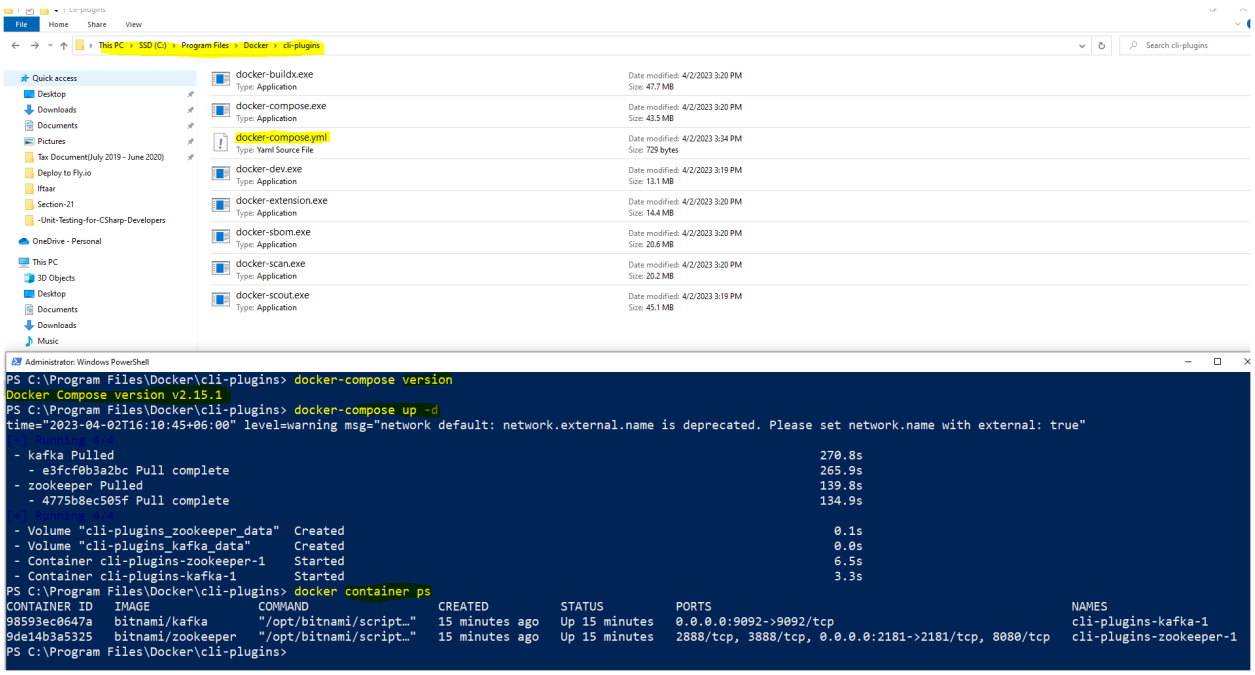
Let's create a Docker network which will enable our microservices to communicate with Apache, Kafka, MongoDB and Microsoft SQL Server, which we are going to install as Docker containers



Part-8 : Run Kafka In Docker

About ZooKeeper:
Src: https://dattell.com/data-architecture-blog/what-is-zookeeper-how-does-it-support-kafka/

Apache Zookeeper is very important because the Kafka broker's are stateless and Zookeeper is responsible for managing the Kafka cluster and also for electing the lead broker. But we will only add a single broker, so that's why we have the zookeeper service.

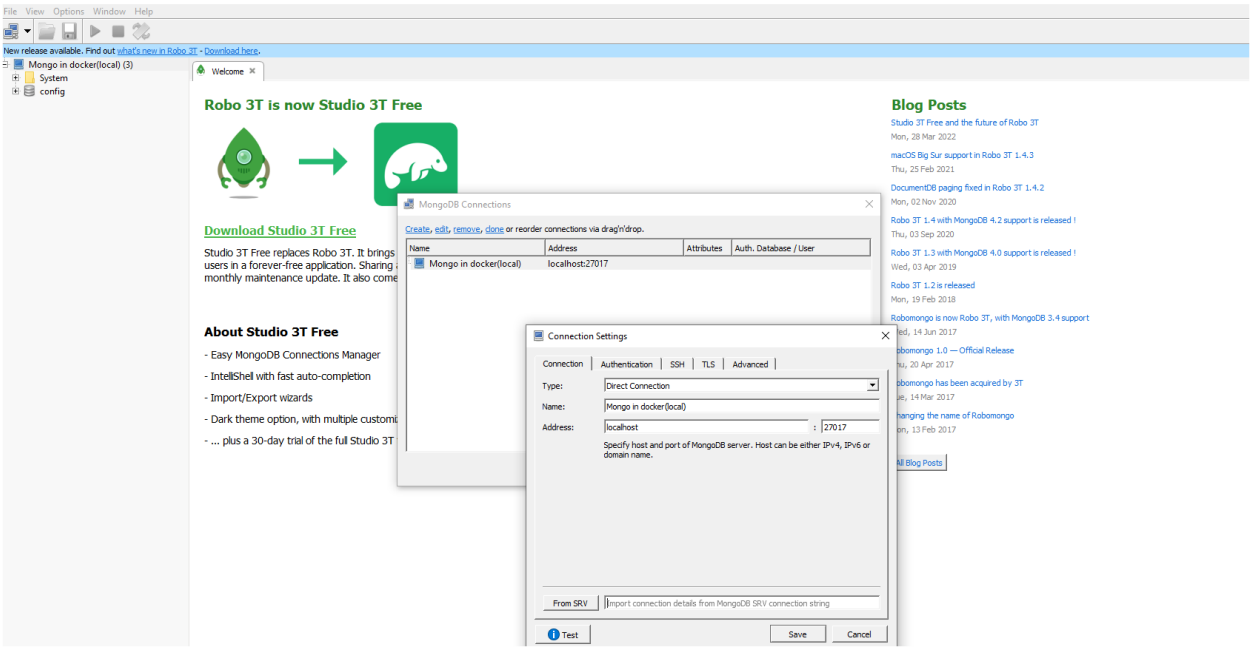


docker compose up -d
Now, Dash d is very important because it will run it as detached.
If you do not add dash d, it will run Kafka and Apache Zookeeper in your terminal session.

Part-9 : Run MongoDB In Docker

-v allows us to specify a Docker volume. Now in order to be able to save or persist data and also to share data between containers Docker came up with the concept of volumes. Quite simply, volumes are directories of files that are outside of the default union file system and exist as normal directories and files on the host file system.
So this is not in the container itself because if it was, we would lose the data every time the container is recreated.

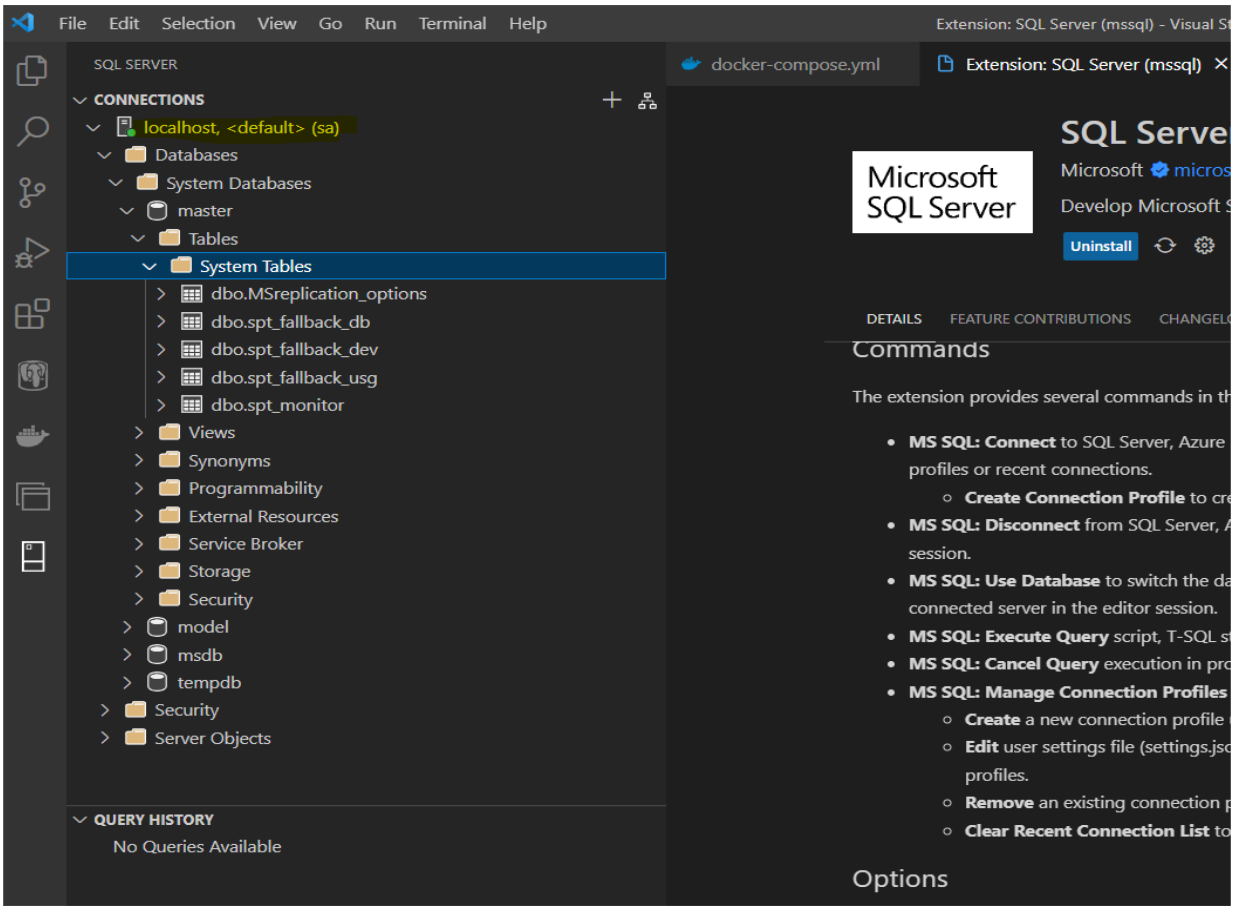
```
Administrator: Windows PowerShell
PS C:\WINDOWS\system32> docker run -it -d --name mongo-container -p 27017:27017  --network mydockernetwork  --restart always -v mongodb_data_container:/data/db  mongo:latest
Unable to find image 'mongo:latest' locally
latest: Pulling from library/mongo
74ac377868f8: Pull complete
7bf92c33f8cb: Pull complete
3ad057cae032: Pull complete
94195d632f1b: Pull complete
df4a989974eb: Pull complete
a94fc3cc2ce1: Pull complete
e31c2ebced96: Pull complete
7af2dc62bb4b: Pull complete
ec5239bce583: Pull complete
Digest: sha256:cc4522f3f5c0d3435046eb51b1d8a633d8e24d8e661b6ba127a98e5519d11bde
Status: Downloaded newer image for mongo:latest
4c7a826da7fd7e3b25dd70b67a3e04bb2fbe346a09d129783cf581ae7da9375c
PS C:\WINDOWS\system32> docker ps
CONTAINER ID   IMAGE              COMMAND                  CREATED              STATUS          PORTS                                                 NAMES
4c7a826da7fd   mongo:latest       "docker-entrypoint.s…"   About a minute ago   Up 48 seconds   0.0.0.0:27017->27017/tcp                              mongo-container
98593ec0647a   bitnami/kafka      "/opt/bitnami/script…"   58 minutes ago       Up 58 minutes   0.0.0.0:9092->9092/tcp                                cli-plugins-kafka-1
9de14b3a5325   bitnami/zookeeper  "/opt/bitnami/script…"   58 minutes ago       Up 58 minutes   2888/tcp, 3888/tcp, 0.0.0.0:2181->2181/tcp, 8080/tcp  cli-plugins-zookeeper-1
PS C:\WINDOWS\system32>
```



Part-10 : Run Microsoft SQL In Docker

EULA : User License
SA: System Administrator Password

```
PS C:\WINDOWS\system32> docker run -d --name sql-container --network mydockernetwork --restart always -e 'ACCEPT_EULA=Y' -e 'SA_PASSWORD=$tr0ngS@P@ssw0rd02' -e 'MSSQL_PID=Express' -p 1433:1433 mcr.microsoft.com/mssql/server:2017-latest-ubuntu
Unable to find image 'mcr.microsoft.com/mssql/server:2017-latest-ubuntu' locally
2017-latest-ubuntu: Pulling from mssql/server
9f82a64a5b13: Pull complete
652747db0187: Pull complete
aafd94cf844c: Pull complete
0d4cc1ecf80c: Pull complete
fa70aa07bab4: Pull complete
d9afe5573d95: Pull complete
2718b5d7820f: Pull complete
e9b5ce20e8cb: Pull complete
Digest: sha256:24e58f079a083e975b0066f30bec7301d5a2cb0aad0cc50156218ae1c3d757c6
Status: Downloaded newer image for mcr.microsoft.com/mssql/server:2017-latest-ubuntu
44c7b458f3aee085e775e5f86c9b10af5e4200934e71502ff5aeaf6c43a27ddc
PS C:\WINDOWS\system32> docker ps
CONTAINER ID   IMAGE                                COMMAND                  CREATED          STATUS          PORTS                                                NAMES
44c7b458f3ae   mcr.microsoft.com/mssql/server:2017-latest-ubuntu   "/opt/mssql/bin/nonr…"   3 minutes ago    Up 3 minutes    0.0.0.0:1433->1433/tcp                               sql-container
4c7a826da7fd   mongo:latest                         "docker-entrypoint.s…"   2 hours ago      Up 2 hours      0.0.0.0:27017->27017/tcp                             mongo-container
98593ec0647a   bitnami/kafka                        "/opt/bitnami/script…"   3 hours ago      Up 3 hours      0.0.0.0:9092->9092/tcp                               cli-plugins-kafka-1
9de14b3a5325   bitnami/zookeeper                    "/opt/bitnami/script…"   3 hours ago      Up 3 hours      2888/tcp, 3888/tcp, 0.0.0.0:2181->2181/tcp, 8080/tcp   cli-plugins-zookeeper-1
PS C:\WINDOWS\system32>
```



Part-11 : Basic Project Setup

What is dotnet class library
Src: https://learn.microsoft.com/en-us/dotnet/standard/class-libraries

```
D:\.NET Microservices-CQRS & Event Sourcing with Kafka>mkdir CQRS-ES

D:\.NET Microservices-CQRS & Event Sourcing with Kafka>mkdir SM-Post

D:\.NET Microservices-CQRS & Event Sourcing with Kafka>cd SM-Post

D:\.NET Microservices-CQRS & Event Sourcing with Kafka\SM-Post>mkdir Post.Cmd

D:\.NET Microservices-CQRS & Event Sourcing with Kafka\SM-Post>mkdir Post.Query

D:\.NET Microservices-CQRS & Event Sourcing with Kafka\SM-Post>
```

-o will create a directory by the supplied name

```
D:\.NET Microservices-CQRS & Event Sourcing with Kafka\SM-Post>cd..

D:\.NET Microservices-CQRS & Event Sourcing with Kafka>cd CQRS-ES

D:\.NET Microservices-CQRS & Event Sourcing with Kafka\CQRS-ES>dotnet new classlib -o CQRS.core
The template "Class Library" was created successfully.

Processing post-creation actions...
Restoring D:\.NET Microservices-CQRS & Event Sourcing with Kafka\CQRS-ES\CQRS.core\CQRS.core.csproj:
  Determining projects to restore...
  Restored D:\.NET Microservices-CQRS & Event Sourcing with Kafka\CQRS-ES\CQRS.core\CQRS.core.csproj (in 66 ms
  ).
Restore succeeded.



D:\.NET Microservices-CQRS & Event Sourcing with Kafka\CQRS-ES>cd..

D:\.NET Microservices-CQRS & Event Sourcing with Kafka>cd SM-Post

D:\.NET Microservices-CQRS & Event Sourcing with Kafka\SM-Post>dotnet new sln
The template "Solution File" was created successfully.


D:\.NET Microservices-CQRS & Event Sourcing with Kafka\SM-Post>
```

```
D:\.NET Microservices-CQRS & Event Sourcing with Kafka\SM-Post>cd Post.Cmd

D:\.NET Microservices-CQRS & Event Sourcing with Kafka\SM-Post\Post.Cmd>dotnet new webapi -o Post.Cmd.Api
The template "ASP.NET Core Web API" was created successfully.

Processing post-creation actions...
Restoring D:\.NET Microservices-CQRS & Event Sourcing with Kafka\SM-Post\Post.Cmd\Post.Cmd.Api\Post.Cmd.Api.csproj:
  Determining projects to restore...
  Restored D:\.NET Microservices-CQRS & Event Sourcing with Kafka\SM-Post\Post.Cmd\Post.Cmd.Api\Post.Cmd.Api.csproj (in 216 ms).
Restore succeeded.



D:\.NET Microservices-CQRS & Event Sourcing with Kafka\SM-Post\Post.Cmd>dotnet new classlib -o Post.Cmd.Domain
The template "Class Library" was created successfully.

Processing post-creation actions...
Restoring D:\.NET Microservices-CQRS & Event Sourcing with Kafka\SM-Post\Post.Cmd\Post.Cmd.Domain\Post.Cmd.Domain.csproj:
  Determining projects to restore...
  Restored D:\.NET Microservices-CQRS & Event Sourcing with Kafka\SM-Post\Post.Cmd\Post.Cmd.Domain\Post.Cmd.Domain.csproj (in 56 ms).
Restore succeeded.



D:\.NET Microservices-CQRS & Event Sourcing with Kafka\SM-Post\Post.Cmd>dotnet new classlib -o Post.Cmd.Infrastructure
The template "Class Library" was created successfully.

Processing post-creation actions...
Restoring D:\.NET Microservices-CQRS & Event Sourcing with Kafka\SM-Post\Post.Cmd\Post.Cmd.Infrastructure\Post.Cmd.Infrastructure.csproj:
  Determining projects to restore...
  Restored D:\.NET Microservices-CQRS & Event Sourcing with Kafka\SM-Post\Post.Cmd\Post.Cmd.Infrastructure\Post.Cmd.Infrastructure.csproj (in 55 ms).
Restore succeeded.



D:\.NET Microservices-CQRS & Event Sourcing with Kafka\SM-Post\Post.Cmd>
```

```
D:\.NET Microservices-CQRS & Event Sourcing with Kafka\SM-Post\Post.Cmd>cd..

D:\.NET Microservices-CQRS & Event Sourcing with Kafka\SM-Post>cd Post.Query

D:\.NET Microservices-CQRS & Event Sourcing with Kafka\SM-Post\Post.Query>dotnet new webapi -o Post.Query.Api
The template "ASP.NET Core Web API" was created successfully.

Processing post-creation actions...
Restoring D:\.NET Microservices-CQRS & Event Sourcing with Kafka\SM-Post\Post.Query\Post.Query.Api\Post.Query.Api.csproj:
  Determining projects to restore...
  Restored D:\.NET Microservices-CQRS & Event Sourcing with Kafka\SM-Post\Post.Query\Post.Query.Api\Post.Query.Api.csproj (in 165 ms).
Restore succeeded.



D:\.NET Microservices-CQRS & Event Sourcing with Kafka\SM-Post\Post.Query>dotnet new classlib -o Post.Query.Domain
The template "Class Library" was created successfully.

Processing post-creation actions...
Restoring D:\.NET Microservices-CQRS & Event Sourcing with Kafka\SM-Post\Post.Query\Post.Query.Domain\Post.Query.Domain.csproj:
  Determining projects to restore...
  Restored D:\.NET Microservices-CQRS & Event Sourcing with Kafka\SM-Post\Post.Query\Post.Query.Domain\Post.Query.Domain.csproj (in 56 ms).
Restore succeeded.



D:\.NET Microservices-CQRS & Event Sourcing with Kafka\SM-Post\Post.Query>dotnet new classlib -o Post.Query.Infrastructure
The template "Class Library" was created successfully.

Processing post-creation actions...
Restoring D:\.NET Microservices-CQRS & Event Sourcing with Kafka\SM-Post\Post.Query\Post.Query.Infrastructure\Post.Query.Infrastructure.csproj:
  Determining projects to restore...
  Restored D:\.NET Microservices-CQRS & Event Sourcing with Kafka\SM-Post\Post.Query\Post.Query.Infrastructure\Post.Query.Infrastructure.csproj (in 55 ms).
Restore succeeded.



D:\.NET Microservices-CQRS & Event Sourcing with Kafka\SM-Post\Post.Query>
```

```
D:\.NET Microservices-CQRS & Event Sourcing with Kafka\SM-Post\Post.Query>cd..

D:\.NET Microservices-CQRS & Event Sourcing with Kafka\SM-Post>dotnet sln add ../CQRS-ES/CQRS.Core/CQRS.Core.csproj
Project `..\CQRS-ES\CQRS.Core\CQRS.Core.csproj` added to the solution.

D:\.NET Microservices-CQRS & Event Sourcing with Kafka\SM-Post>dotnet sln add Post.Cmd/Post.Cmd.Api/Post.Cmd.Api.csproj
Project `Post.Cmd\Post.Cmd.Api\Post.Cmd.Api.csproj` added to the solution.

D:\.NET Microservices-CQRS & Event Sourcing with Kafka\SM-Post>dotnet sln add Post.Cmd/Post.Cmd.Domain/Post.Cmd.Domain.csproj
Project `Post.Cmd\Post.Cmd.Domain\Post.Cmd.Domain.csproj` added to the solution.

D:\.NET Microservices-CQRS & Event Sourcing with Kafka\SM-Post>dotnet sln add Post.Cmd/Post.Cmd.Infrastructure/Post.Cmd.Infrastructure.csproj
Project `Post.Cmd\Post.Cmd.Infrastructure\Post.Cmd.Infrastructure.csproj` added to the solution.

D:\.NET Microservices-CQRS & Event Sourcing with Kafka\SM-Post>
```

```
D:\.NET Microservices-CQRS & Event Sourcing with Kafka\SM-Post>dotnet sln add Post.Query/Post.Query.Api/Post.Query.Api.csproj
Project `Post.Query\Post.Query.Api\Post.Query.Api.csproj` added to the solution.

D:\.NET Microservices-CQRS & Event Sourcing with Kafka\SM-Post>dotnet sln add Post.Query/Post.Query.Domain/Post.Query.Domain.csproj
Project `Post.Query\Post.Query.Domain\Post.Query.Domain.csproj` added to the solution.

D:\.NET Microservices-CQRS & Event Sourcing with Kafka\SM-Post>dotnet sln add Post.Query/Post.Query.Infrastructure/Post.Query.Infrastructure.csproj
Project `Post.Query\Post.Query.Infrastructure\Post.Query.Infrastructure.csproj` added to the solution.

D:\.NET Microservices-CQRS & Event Sourcing with Kafka\SM-Post>
```

Part-12 : Adding Project References

```
D:\.NET Microservices-CQRS & Event Sourcing with Kafka\SM-Post>dotnet add Post.Cmd/Post.Cmd.Api/Post.Cmd.Api.csproj reference ../CQRS-ES/CQRS.Core/CQRS.Core.csproj
Reference `..\..\..\CQRS-ES\CQRS.Core\CQRS.Core.csproj` added to the project.

D:\.NET Microservices-CQRS & Event Sourcing with Kafka\SM-Post>dotnet add Post.Cmd/Post.Cmd.Api/Post.Cmd.Api.csproj reference Post.Cmd/Post.Cmd.Domain/Post.Cmd.Domain.csproj
Reference `..\Post.Cmd.Domain\Post.Cmd.Domain.csproj` added to the project.

D:\.NET Microservices-CQRS & Event Sourcing with Kafka\SM-Post>dotnet add Post.Cmd/Post.Cmd.Api/Post.Cmd.Api.csproj reference Post.Cmd/Post.Cmd.Infrastructure/Post.Cmd.Infrastructure.csproj
Reference `..\Post.Cmd.Infrastructure\Post.Cmd.Infrastructure.csproj` added to the project.

D:\.NET Microservices-CQRS & Event Sourcing with Kafka\SM-Post>dotnet new classlib -o Post.Common
The template "Class Library" was created successfully.

Processing post-creation actions...
Restoring D:\.NET Microservices-CQRS & Event Sourcing with Kafka\SM-Post\Post.Common\Post.Common.csproj:
  Determining projects to restore...
  Restored D:\.NET Microservices-CQRS & Event Sourcing with Kafka\SM-Post\Post.Common\Post.Common.csproj (in 56 ms).
Restore succeeded.


D:\.NET Microservices-CQRS & Event Sourcing with Kafka\SM-Post>dotnet add Post.Cmd/Post.Cmd.Api/Post.Cmd.Api.csproj reference Post.Common/Post.Common.csproj
Reference `..\..\Post.Common\Post.Common.csproj` added to the project.

D:\.NET Microservices-CQRS & Event Sourcing with Kafka\SM-Post>
```

```
D:\.NET Microservices-CQRS & Event Sourcing with Kafka\SM-Post>dotnet add Post.Cmd/Post.Cmd.Domain/Post.Cmd.Domain.csproj reference ../CQRS-ES/CQRS.Core/CQRS.Core.csproj
Reference `..\..\..\CQRS-ES\CQRS.Core\CQRS.Core.csproj` added to the project.

D:\.NET Microservices-CQRS & Event Sourcing with Kafka\SM-Post>dotnet add Post.Cmd/Post.Cmd.Domain/Post.Cmd.Domain.csproj reference Post.Common/Post.Common.csproj
Reference `..\..\Post.Common\Post.Common.csproj` added to the project.

D:\.NET Microservices-CQRS & Event Sourcing with Kafka\SM-Post>dotnet add Post.Cmd/Post.Cmd.Infrastructure/Post.Cmd.Infrastructure.csproj reference ../CQRS-ES/CQRS.Core/CQRS.Core.csproj
Reference `..\..\..\CQRS-ES\CQRS.Core\CQRS.Core.csproj` added to the project.

D:\.NET Microservices-CQRS & Event Sourcing with Kafka\SM-Post>dotnet add Post.Cmd/Post.Cmd.Infrastructure/Post.Cmd.Infrastructure.csproj reference Post.Cmd/Post.Cmd.Domain/Post.Cmd.Domain.csproj
Reference `..\Post.Cmd.Domain\Post.Cmd.Domain.csproj` added to the project.

D:\.NET Microservices-CQRS & Event Sourcing with Kafka\SM-Post>
```

```
D:\.NET Microservices-CQRS & Event Sourcing with Kafka\SM-Post>dotnet add Post.Common/Post.Common.csproj reference ../CQRS-ES/CQRS.Core/CQRS.Core.csproj
Reference `..\..\CQRS-ES\CQRS.Core\CQRS.Core.csproj` added to the project.
```

```
D:\.NET Microservices-CQRS & Event Sourcing with Kafka\SM-Post>dotnet add Post.Query/Post.Query.Api/Post.Query.Api.csproj reference ../CQRS-ES/CQRS.Core/CQRS.Core.csproj
Reference `..\..\..\CQRS-ES\CQRS.Core\CQRS.Core.csproj` added to the project.

D:\.NET Microservices-CQRS & Event Sourcing with Kafka\SM-Post>dotnet add Post.Query/Post.Query.Api/Post.Query.Api.csproj reference Post.Query/Post.Query.Domain/Post.Query.Domain.csproj
Reference `..\Post.Query.Domain\Post.Query.Domain.csproj` added to the project.

D:\.NET Microservices-CQRS & Event Sourcing with Kafka\SM-Post>dotnet add Post.Query/Post.Query.Api/Post.Query.Api.csproj reference Post.Query/Post.Query.Domain/Post.Query.Infrastructure.csp
roj
Could not find project or directory `Post.Query/Post.Query.Domain/Post.Query.Infrastructure.csproj`.

D:\.NET Microservices-CQRS & Event Sourcing with Kafka\SM-Post>dotnet add Post.Query/Post.Query.Api/Post.Query.Api.csproj reference Post.Query/Post.Query.Infrastructure/Post.Query.Infrastruc
ture.csproj
Reference `..\Post.Query.Infrastructure\Post.Query.Infrastructure.csproj` added to the project.

D:\.NET Microservices-CQRS & Event Sourcing with Kafka\SM-Post>dotnet add Post.Query/Post.Query.Api/Post.Query.Api.csproj reference Post.Common/Post.Common.csproj
Reference `..\..\Post.Common\Post.Common.csproj` added to the project.

D:\.NET Microservices-CQRS & Event Sourcing with Kafka\SM-Post>
```

```
D:\.NET Microservices-CQRS & Event Sourcing with Kafka\SM-Post>dotnet add Post.Query/Post.Query.Domain/Post.Query.Domain.csproj reference ../CQRS-ES/CQRS.Core/CQRS.Core.csproj
Reference `..\..\..\CQRS-ES\CQRS.Core\CQRS.Core.csproj` added to the project.

D:\.NET Microservices-CQRS & Event Sourcing with Kafka\SM-Post>dotnet add Post.Query/Post.Query.Domain/Post.Query.Domain.csproj reference Post.Common/post.Common.csproj
Reference `..\..\Post.Common\post.Common.csproj` added to the project.

D:\.NET Microservices-CQRS & Event Sourcing with Kafka\SM-Post>dotnet add Post.Query/Post.Query.Infrastructure/Post.Query.Infrastructure.csproj reference ../CQRS-ES/CQRS.Core/CQRS.Core.csproj
j
Reference `..\..\..\CQRS-ES\CQRS.Core\CQRS.Core.csproj` added to the project.

D:\.NET Microservices-CQRS & Event Sourcing with Kafka\SM-Post>dotnet add Post.Query/Post.Query.Infrastructure/Post.Query.Infrastructure.csproj reference Post.Query/Post.Query.Domain/Post.Qu
ery.Domain.csproj
Reference `..\Post.Query.Domain\Post.Query.Domain.csproj` added to the project.

D:\.NET Microservices-CQRS & Event Sourcing with Kafka\SM-Post>
```

Part-13 : Adding NuGet Packages

NuGet is the official package manager for the DOTNet Developer Platform and NuGet package is basically a compiled library with some descriptive metadata. It is a mechanism through which dot net developers can create, share and consume useful code.

Confluent : Running or flowing together
"dotnet restore"is used to restore dependencies, such as NuGet packages, of a .NET project.

```
D:\.NET Microservices-CQRS & Event Sourcing with Kafka\SM-Post>dotnet restore
  Determining projects to restore...
  Restored D:\.NET Microservices-CQRS & Event Sourcing with Kafka\SM-Post\Post.Query\Post.Query.Domain\Post.Query.Domai
  n.csproj (in 251 ms).
  Restored D:\.NET Microservices-CQRS & Event Sourcing with Kafka\SM-Post\Post.Query\Post.Query.Api\Post.Query.Api.cspr
  oj (in 252 ms).
  Restored D:\.NET Microservices-CQRS & Event Sourcing with Kafka\SM-Post\Post.Cmd\Post.Cmd.Domain\Post.Cmd.Domain.cspr
  oj (in 252 ms).
  Restored D:\.NET Microservices-CQRS & Event Sourcing with Kafka\SM-Post\Post.Query\Post.Query.Infrastructure\Post.Que
  ry.Infrastructure.csproj (in 252 ms).
  Restored D:\.NET Microservices-CQRS & Event Sourcing with Kafka\SM-Post\Post.Cmd\Post.Cmd.Api\Post.Cmd.Api.csproj (in
   251 ms).
  Restored D:\.NET Microservices-CQRS & Event Sourcing with Kafka\SM-Post\Post.Common\Post.Common.csproj (in 252 ms).
  Restored D:\.NET Microservices-CQRS & Event Sourcing with Kafka\SM-Post\Post.Cmd\Post.Cmd.Infrastructure\Post.Cmd.Inf
  rastructure.csproj (in 252 ms).
  1 of 8 projects are up-to-date for restore.

D:\.NET Microservices-CQRS & Event Sourcing with Kafka\SM-Post>dotnet restore
  Determining projects to restore...
  Restored D:\.NET Microservices-CQRS & Event Sourcing with Kafka\SM-Post\Post.Query\Post.Query.Domain\Post.Query.Domain.csproj (in 203 ms).
  Restored D:\.NET Microservices-CQRS & Event Sourcing with Kafka\SM-Post\Post.Query\Post.Query.Api\Post.Query.Api.csproj (in 223 ms).
  Restored D:\.NET Microservices-CQRS & Event Sourcing with Kafka\SM-Post\Post.Query\Post.Query.Infrastructure\Post.Query.Infrastructure.csproj (in 222 ms).
  5 of 8 projects are up-to-date for restore.

D:\.NET Microservices-CQRS & Event Sourcing with Kafka\SM-Post>
```

```
D:\.NET Microservices-CQRS & Event Sourcing with Kafka\SM-Post>dotnet build
MSBuild version 17.4.0+18d5aef85 for .NET
  Determining projects to restore...
  Restored D:\.NET Microservices-CQRS & Event Sourcing with Kafka\SM-Post\Post.Cmd\Post.Cmd.Api\Post.Cmd.Api.csproj (in
   344 ms).
  Restored D:\.NET Microservices-CQRS & Event Sourcing with Kafka\SM-Post\Post.Query\Post.Query.Domain\Post.Query.Domai
  n.csproj (in 344 ms).
  Restored D:\.NET Microservices-CQRS & Event Sourcing with Kafka\SM-Post\Post.Cmd\Post.Cmd.Infrastructure\Post.Cmd.Inf
  rastructure.csproj (in 344 ms).
  Restored D:\.NET Microservices-CQRS & Event Sourcing with Kafka\SM-Post\Post.Query\Post.Query.Infrastructure\Post.Que
  ry.Infrastructure.csproj (in 346 ms).
  Restored D:\.NET Microservices-CQRS & Event Sourcing with Kafka\SM-Post\Post.Query\Post.Query.Api\Post.Query.Api.cspr
  oj (in 346 ms).
  Restored D:\.NET Microservices-CQRS & Event Sourcing with Kafka\SM-Post\Post.Cmd\Post.Cmd.Domain\Post.Cmd.Domain.cspr
  oj (in 344 ms).
  Restored D:\.NET Microservices-CQRS & Event Sourcing with Kafka\SM-Post\Post.Common\post.Common.csproj (in 344 ms).
  1 of 8 projects are up-to-date for restore.
  CQRS.Core -> D:\.NET Microservices-CQRS & Event Sourcing with Kafka\CQRS-ES\CQRS.Core\bin\Debug\net7.0\CQRS.Core.dll
  post.Common -> D:\.NET Microservices-CQRS & Event Sourcing with Kafka\SM-Post\Post.Common\bin\Debug\net7.0\post.Commo
  n.dll
  Post.Cmd.Domain -> D:\.NET Microservices-CQRS & Event Sourcing with Kafka\SM-Post\Post.Cmd\Post.Cmd.Domain\bin\Debug\
  net7.0\Post.Cmd.Domain.dll
  Post.Query.Domain -> D:\.NET Microservices-CQRS & Event Sourcing with Kafka\SM-Post\Post.Query\Post.Query.Domain\bin\
  Debug\net7.0\Post.Query.Domain.dll
  Post.Cmd.Infrastructure -> D:\.NET Microservices-CQRS & Event Sourcing with Kafka\SM-Post\Post.Cmd\Post.Cmd.Infrastru
  cture\bin\Debug\net7.0\Post.Cmd.Infrastructure.dll
  Post.Query.Infrastructure -> D:\.NET Microservices-CQRS & Event Sourcing with Kafka\SM-Post\Post.Query\Post.Query.Inf
  rastructure\bin\Debug\net7.0\Post.Query.Infrastructure.dll
  Post.Cmd.Api -> D:\.NET Microservices-CQRS & Event Sourcing with Kafka\SM-Post\Post.Cmd\Post.Cmd.Api\bin\Debug\net7.0
  \Post.Cmd.Api.dll
  Post.Query.Api -> D:\.NET Microservices-CQRS & Event Sourcing with Kafka\SM-Post\Post.Query\Post.Query.Api\bin\Debug\
  net7.0\Post.Query.Api.dll

Build succeeded.
    0 Warning(s)
    0 Error(s)

Time Elapsed 00:00:06.33

D:\.NET Microservices-CQRS & Event Sourcing with Kafka\SM-Post>
```

Section-3: Messages

Part-15 : What Is A command

Three types of messages:

- Command
- Event
- Queries

What is a command:

A command is a combination of expressed intent. In other words, it describes an action that you want to be performed. It also contains the information that is required to undertake the desired action. Commands are always named with the verb in the imperative mood.
For example, NewPostCommand, LikePostCommand or AddCommentCommand.

Part-17 : What Is An Event

Events are objects that describe something that has occurred in the application.
A typical source of the event is the aggregate. When something important happens in the aggregate, it will raise an event. Events are named with the past-particle verb.
For example, PostCreatedEvent, PostLikedEvent, CommentAddedEvent

What is aggregate ?

Section-4: Command Dispatching

Part-19 : The Mediator Pattern

Keyword: Dispatch, Behavioral Design Pattern , Mediator

Mediator Pattern

- The mediator pattern is a behavioral design pattern.
- It promotes loose coupling by preventing objects from referring to each other explicitly.
- It is used to simplify communication between objects in an application by introducing a single object known as the mediator that manages the distribution of messages among other objects.

Future Task

➢ See this video Again

Part-20 : ICommandDispatcher Interface(The Mediator)

Func Delegate:
Func<T1, T2> => The last parameter will always be out parameter. Here T2.

C# programming language comply with the **Liskov-Substitution** principle that states that a concrete class should be substitutable by the base class without affecting the workings of the program.

Part-21 : CommandDispatcher (The Concrete Mediator)

Dictionary<Type, Func<BaseCommand, Task>> _handlers = **new()**;
_handlers.Add(typeof(T), **x => handler((T)x)**);
_handlers.**TryGetValue(command.GetType(), out Func<BaseCommand, Task> handler)**

Failed In Quiz:

- ➔ The **CommandDispatcher** is the concrete mediator that is responsible for coordinating the colleague objects.

- ➔ The **ICommandDispatcher** interface is the mediator that manages the distribution of messages (commands) among other objects.

Future Task

- ➢ See this video Again to understand it's functionality
- ➢ Overall there is a lack of understanding in this section(4). So try later.

Section-5: Aggregate

Part-22 : What Is An Aggregate

What is the aggregate:
The aggregate is an entity or group of entities that is always kept in a consistent state, and the aggregate root is the entity within the aggregate that is responsible for maintaining that state.

- ➢ The aggregate can be viewed as the domain entity on the write or command side of a CQRS and **Event Sourcing** based application or service, similar to the domain entity that you find on the read or query side.

- ➢ When you create the aggregate class, you will see that it is difficult at first glance to view the aggregate as a domain entity, because unlike the domain entity on the read side, it is not a simple plain old C# object that contains only state and no behavior.

Important:

- ➔ The fundamental differences in its structure is due to the fundamental difference in how the data is stored in the write database or event store versus how it is stored in the database.

- ➔ The read side is simple, One instance of the domain entity represents one record in the database.

- ➔ The write side is more complex because there we store the data as a sequence of immutable events over time. In other words, we store all state changes and we save these state changes in the form of events that are versioned.

- ➔ The design of the aggregate should therefore allow you to be able to use these events to recreate or replay the latest state of the Aggregate, so that you do not have to query the read database for the latest state, else the hard separation of commands and queries would be in vain.

Part-23 : AggregateRoot

Remember, the AggregateRoot is the entity within the aggregate that is responsible for keeping the aggregate in a consistent state.

But how does the aggregate achieve this?

- ➔ By assisting the aggregate to raise events.
- ➔ Apply changes to the aggregate state.
- ➔ Keeping track of uncommitted changes and replaying the latest state of the aggregate.

@ => void Foo(int @string)
It's just a way to allow declaring reserved keywords as vars.(Link)
Read more here

Instance of Interface or Abstract cannot be cannot be created.

Future Task:

- Need to learn about reflection. Why did we need reflection in the ApplyChange Method?
- How does Polymorphism work in the RaiseEvent function?
- Why AggregateRoot needed to be an abstract class?

Part-24 : PostAggregate Part1

Keyword: Tuple in C#, this in C#

Future Task:

- Failed in the quiz. So retake the quiz again and write down the answers of the questions of the quiz

Section-6: Event Store

Part-26 : What is An Event Store

The event store, also known as the write database, is a database where the data is stored as a sequence of immutable events Over time.

The event store is a key component of event sourcing, and the following are key considerations when designing an event store.

- ➢ An event store must be an append only store. No update or delete. Operations should be allowed.
- ➢ Each event that is saved, should represent the version or state of an aggregate at any given point in time.
- ➢ Events should be stored in chronological order and new events should be appended to the previous event. ***
- ➢ The state of the aggregate should be recreateable by replaying the event store.
- ➢ Implement optimistic concurrency control.

Part-27 : EventModel

The purpose of the event model is to represent the schema of the event store and each instance of the event model will represent a record in the event store or more accurately, Since we are using MongoDB for our event store, it is better to say that each instance of the event model will represent a document in the event store collection, and each document again will represent an event that is versioned that can alter the state of the aggregate.

Thanks to polymorphism, we can assign an instance of a concrete event object to the base event there, and that will ensure that we have all of the event information and that is most important there for replaying the event store.***
public BaseEvent EventData { get; set; }

Part-28 : IEventStoreRepository Interface

The repository pattern is used to create an abstraction layer between the data access and business logic layers.

we're going to create the event store repository interface, which sole purpose is to provide an interface abstraction for us to interact with our events store database.

We are not going to define an update nor a delete operation because remember that I've said that an event store should be immutable or unchangeable, so we should only be able to create data and retrieve data.

Part-29 : EventStoreRepository

IOptions<MongoDbConfig> config builder.Services.**Configure<**MongoDbConfig>(builder.Configuration.GetSection(nameof(MongoDbConfig)));

ConfigureAwait(false)
This is used to avoid forcing the callback to be invoked on the original context or scheduler. There are some benefits such as improving performance and avoiding deadlocks.

Future Task:

- How to configure mongo config for multiple databases and multiple collections.
- Learn More About IOption (Link), Option Pattern
- Learn more about ConfigureAwait

Part-31 : EventStore

A **Scoped** service will create a new instance for each unique HTTP request.
A **Singleton** service, which will only create a single instance of events store repository for the entire application.
A **Transient** service which will create a new instance everywhere we use it.

Future Task:

- **Optimistic concurrency** is applied in the SaveEventAsync Method. How?
- If we extend Exception class and pass a string message then there is no constructor to receive it in Exception class
- Rewatch this section again and take the quiz

From Quiz:
The Event Store is used on the write or command side of a CQRS and Event Sourcing based application, and it is used to store data as a sequence of immutable events over time.

Section-7: Command Handling

Part-37 : Register Command Handlers

builder.Services.**BuildServiceProvider().GetRequiredService<ICommandHandler>()**;

The reason behind this implementation is because of dependency injection in the CommandHandler class.

builder.Services.**AddSingleton**<ICommandDispatcher>(**_ => dispatcher**);

Future Task:

- Retake the quiz.

Section-8: Event Producer

Part-39 : EventProducer Interface & Implementation

- builder.Services.Configure<**ProducerConfig**>(builder.Configuration.GetSection(nameof(ProducerConfig)));

We can override a property value by configuring a service like this.Here we override the value of **BootstrapServers**

- Value = JsonSerializer.Serialize(@event, @event.GetType())

How Polymorphism and Liskov-Substitution comply in this line of code.

Part-40 : Producing An Event

Setting Environment Variable , IConfiguration vs System.Environment

Now you might say, yeah, what if we fail to successfully produce the event to Kafka?
And in this case we have already successfully persisted the event to the event store. Now, I would suggest here that you add a transaction over the code where you save to MongoDB as well as over the produceAsync method. And then only if both the persisting to the event store and the producing to Kafka succeeds, then you can actually go and commit the transaction now.

MongoDB does support transactions, but the reason that I haven't included it in my code is because I'm only running a single instance of MongoDB on my machine. And for you to use MongoDB transactions, you need MongoDB to run as part of a replica set. So obviously I only have a single instance, but if you do have a replica set or if you want to add a replica set on your machine or on a server and you want to know how to write the code for starting a transaction, committing and aborting a transaction with MongoDB, then feel free to reach out to me and I'll share that code with you.

Future Task:

- Implement Transaction
- Rethink about the qs no 2 in quiz (utf-8)

Quiz:

➔ What is Kafka Topic

A topic can be viewed as a channel through which event data is streamed. Producers always send or produce messages to a topic, while consumers consume events from topics that they subscribe to.

➔ UTF-8 Serializers are used to serialize which type of data?

String.

The **Confluent.Kafka.Message** represents the data type that is produced and consumed to and from Kafka, but serialization occurs prior to the construction of this message. Therefore, the type we are looking for is a general system type rather than a Confluent.Kafka type.

Section-9: Domain Layer

Part-41 : Important DDD Concepts

What is domain driven design?

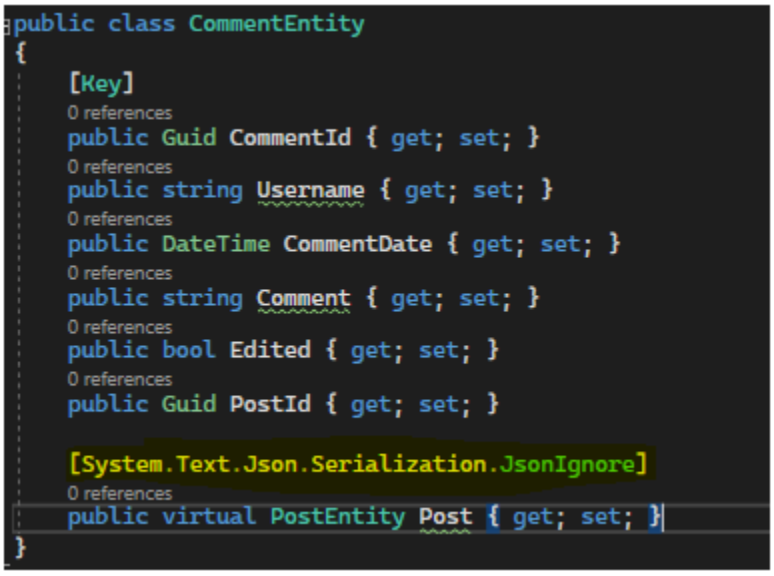
- ➢ The term domain driven design was coined by Erik Evans in 2003.
- ➢ It is an approach to structure and model software in a way that it matches the business domain.
- ➢ It places the primary focus of a software project on the core area of the business, also known as the core domain.
- ➢ It refers to problems as domains and aims to establish a common language to talk about these problems.
- ➢ It describes independent problem areas as bounded contexts.

What is a bounded context?

- ➢ It is an independent problem area.
- ➢ It describes a logical boundary within which a particular model is defined and applicable.
- ➢ Each bounded context correlates to a microservice.
  For example social media post microservice.

Part-42 : Domain Entities

Virtual Keyword and Lazy Loading



```
public class CommentEntity
{
    [Key]
    0 references
    public Guid CommentId { get; set; }
    0 references
    public string Username { get; set; }
    0 references
    public DateTime CommentDate { get; set; }
    0 references
    public string Comment { get; set; }
    0 references
    public bool Edited { get; set; }
    0 references
    public Guid PostId { get; set; }

    [System.Text.Json.Serialization.JsonIgnore]
    0 references
    public virtual PostEntity Post { get; set; }
}
```

Prevent circular reference exceptions.

Section-10: Read Database

Part-46 : DatabaseContext

A DB context instance represents a session with the database and can be used to query and save instances of your entities.

DbSet can be used to query and save instances of the specified entity.
LINQ queries against a DbSet will be translated by entity framework core into queries against the database.

Action Delegate => Encapsulates a method that has a single parameter and does not return a value.

Microsoft.EntityFrameworkCore.Proxies
UseLazyLoadingProxies() => EF Core will then enable lazy loading for any navigation property that can be overridden--that is, it must be **virtual** and on a class that can be inherited from.

Is it correct to define a design for the infrastructure layer in the domain layer?

Navigation properties represent related entities to a principal entity.(From quiz)

Future Task:

- Why we needed DatabaseContextFactory class and configuration in it and injected it as a singleton
- Why do we actually need Microsoft.EntityFrameworkCore.Proxies and what is actually Lazy loading? Because the Navigation entity is not going to get returned if we do not use include in the query.

Part-47 : Programmatically Create DB & Tables On Startup

TrustServerCertificate=True, dataContext.Database.EnsureCreated();

```
Use SocialMedia
GO

IF NOT EXISTS(SELECT * FROM sys.server_principals WHERE name =
'SMUser')
BEGIN
    CREATE LOGIN SMUser WITH PASSWORD=N'SmPA$$06500',
DEFAULT_DATABASE=SocialMedia
END

IF NOT EXISTS(SELECT * FROM sys.database_principals WHERE name =
'SMUser')
BEGIN
    EXEC sp_adduser 'SMUser', 'SMUser', 'db_owner';
END
```

Part-48 : PostRepository

Using(Read More) , discard operator(Read More)

Why is there a database contextfactory implementation?

AsNoTracking => **AsNoTracking queries execute faster** because there's no need to set up change tracking information. It's useful when the results are used in a read only scenario. So in the case of the ListAllAsync method, we are using it in a read only scenario, but in the top method the GetByIdAsync method we do want to track the changes.So there we will not use AsNoTracking because with the AsNoTracking you cannot track the changes on the entity and in the GetByIdAsync method we want to track, but in the ListAllAsync method we are simply querying the data in a read only fashion.

Future Task:

- Why we didn't use AsNoTracking in GetByIdAsync Method. Validate the point that has been made in this lecture.

- Do we need AsNoTracking if we update an entity directly

Section-11: Event Handling

Part-52 : EventHandler

post.Likes++;
No default value for Likes has been set. So how it can be incremented

Future Task:

- Retake the quiz

Section-12: Event Consumer

Part-53 : Kafka Consumer

Rewatch this section

Part-55 : EventJsonConverter

Keyword: ref, out

Future Task:

- Didn't understand this lecture at all. So try later.

Part-56 : EventConsumer

Future Task:

- Didn't understand this lecture at all. So try later.

Part-57 : ConsumerHostedService

We are now going to implement the consumer hosted service, which is a hosted service that represents a background task. Now at the moment, the post query API is just the normal rest API that is stateless and which exposes a restful interface through which its logic can be invoked. Now a microservice usually takes on the form of either a small restful API or an asynchronous event driven consumer service. Now a consumer service is created as a hosted service that you start listening for new event messages from the message bus as soon as the microservice starts up.

In this lecture, we're going to turn the post query API into consumer service as well. In other words, it'll have a background task that listens for new event messages to consume from Kafka, and it will still expose a restful interface through which we will be able to query data that relates to social media posts.

Using make sure that the scope gets disposed of once we are done with it.

Future Task:

- What is stateless rest API
- builder.Services.**AddHostedService**<ConsumerHostedService>();
- Rewatch this whole section
- Failed in the quiz(**How does Kafka track the consumer offset?**)

Section-13: Command Controllers

Part-60 : NewPostController

Future Task:

- How command dispatcher dispatch command

Part-61 : Creating a New Social Media Post

Future Task:

- AggregateIdentifier is not same in MongoDB
- Rewatch this section again

Part-63 : Edit Message of a Social Media Post

Now all of our concrete events extend BaseEvent. So we are obviously using **polymorphism** here. Now MongoDB in its default state does not support polymorphism, but fortunately you can go ahead and set the BsonClassMap and that will make sure that we can tell it that all of our concrete event object types do extend base events.

Future Task:

- Why is the BaseEvent class is abstract? Can't it be concrete ?
- Learn More about BsonClassMap in program.cs file

Part-68 : EditCommentController

Future Task:

- Why we need to replayEvents

Part-72 : DeletePostController

Future Task:

- How cascade delete is working in here

Quiz - 13

PUT is generally used to update an entire resource. PATCH is used for partial updates. PUT can also be used for upserting, in cases where it should create the resource if it does not yet exist.

HTTP status code 400 is used to indicate a Bad Request. This is when the client causes an error rather than the server/API.

Section-14: Queries & Query Dispatching

Part-76 : IQueryDispatcher Interface(The Mediator)

And the reason that that works is because of the **Liskov-Substitution Principle** that states that the base should be substitutable for a concrete or specialized type without affecting the workings of the application.

So if you ask me why am I not using Liskov in the register handler.
Now the only reason is I just wanted to show you two popular ways of using generics or the Liskov-Substitution principle, that's all.
But if you prefer to use list cough in both or generics and both, go ahead and do that.

Quiz - 14

The QueryDispatcher class is the concrete mediator that is responsible for coordinating the query handler methods (colleague objects).

Section-17: A Powerful Ending

Part-90 : Rapidly Change The Read Database Type

Since we installed postgres before, we skipped the installation part.