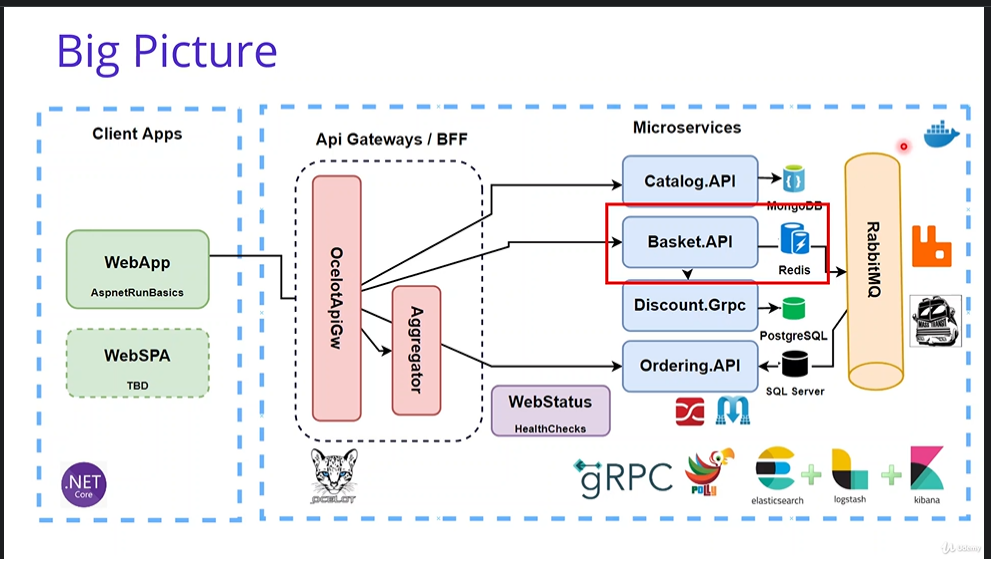
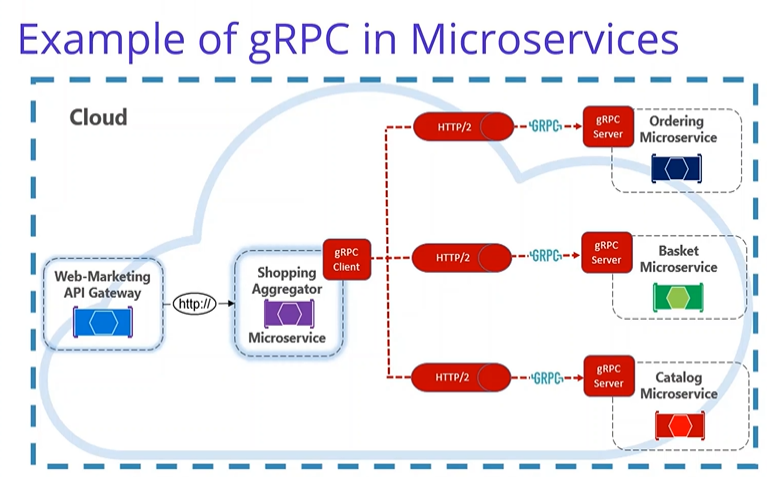
Building Microservices on .Net which used Asp.Net Web API, Docker, RabbitMQ,Ocelot API Gateway, MongoDB,Redis,SqlServer

**Create Repository in GIthub**

* Created repository “aspnetcore-microservices” in Github (<https://github.com/yogesh-narkhede>)
* Clone repository using Visual Studio.

**Create Solution “aspnetcore-microservices”**

****



**Catalog API Service:**

* Created folder structure for src/Services/Catalog.
* Created new project Catalog.API under src/Services/Catalog.
* Created Catalog.API profile under Properties => Debug with URL: <http://localhost:5000>

**MongoDB:**

* Right clicked on Solution and Open in Terminal.
* Go to hub.docker.com and searched for Mongo to get mongo pull image command
* Run Command: docker pull mongo
* Show running images: docker ps
* Show stopped images : docker ps –a
* Run mongo docker image using command: docker run -d -p 27017:27017 --name shopping-mongo mongo
* Run command to check logs: docker logs –f shopping-mongo
* Interactive terminal for MongoDB connection: docker exec -it shopping-mongo /bin/bash
* List command to show all files: ls
* To run mongo command: mongo
  + use CatalogDb
  + db.createCollection(‘Products’)
  + db.Products.insertMany([{'Name': 'Asus Laptop','Category': 'Computers','Summary': 'Summary','Description': 'Description','ImageFile': 'ImageFile','Price': 54.93,},{'Name': 'HP Laptop','Category': 'Computers','Summary': 'Summary','Description': 'Description','ImageFile': 'ImageFile','Price': 88.93}])
  + db.Products.find({}).pretty()
  + db.Products.remove({})

**Add Nuget Packages:**

* Visit website to get install command: <https://www.nuget.org/packages/MongoDB.Driver/>
* Open package manager and execute command: Install-Package MongoDB.Driver
* Update-Package -ProjectName Catalog.API

**Implementation:**

* Created Entities folder and Product class with all fields.
* Created Data folder and ICatalogContext, CatalogContext and CatalogContextSeed classes with implementation.
* Created Repositories folder and ProductRepository interface and class with implementation.
* Created CatalogController with CRUD APIs implementation.
* Add docker-compose
  + Right click on project and Add “Container Orchestrator Support”.
  + Select Docker Compose.
  + Select Operating System as Linux
  + Added/updated catalogdb & catalog.api configuration in docker-compose.yml and docker-compose.override.yml

**Run docker-compose:**

docker-compose -f .\docker-compose.yml -f .\docker-compose.override.yml up -d

**Down docker-compose:**

docker-compose -f .\docker-compose.yml -f .\docker-compose.override.yml down

**Docker commands:**

* docker ps –aq
* Stop all running conatiners: docker stop $(docker ps -aq)
* Remove all containers: docker rm $(docker ps -aq)
* Remove all images: docker rmi $(docker ps -aq)
* Remove unnamed resources: docker system prune
* Show running images: docker ps
* Show stopped images : docker ps –a
* Start existing images: docker start <containerId>
* Stop running container: docker stop <containerId>
* Remove container: docker rm <containerId>
* Show docker images: docker images
* Remove container image: docker rmi <containerId>

**Test swagger in browser:**

<http://localhost:8000/swagger/index.html>

**Mongo GUI options for MongoDb Docker Image**

* <https://hub.docker.com/r/mongoclient/mongoclient>
* Run command in Terminal: docker run -d -p 3000:3000 mongoclient/mongoclient
* Open url in browser: <http://localhost:3000/>

**Basket API Service:**

* Created folder structure for src/Services/Basket.
* Created new project Basket.API under src/Services/Basket.
* Created Basket.API profile under Properties => Debug with URL: <http://localhost:5001>

**Redis Database:**

* Right clicked on docker-compose and Open in Terminal.
* Go to hub.docker.com and searched for Redis to get redis pull image command
* Run Command: docker pull redis
* Run redis docker image using command: docker run -d -p 6379:6379 --name aspnetrun-redis redis
* Interactive terminal for Redis connection: docker exec -it aspnetrun-redis /bin/bash
* Run command to enter into cli: redis-cli
  + set key value
  + get key
  + set name Yogesh
  + get name

**Add Nuget Packages:**

* Visit website to get install command: https://www.nuget.org/packages/Microsoft.Extensions.Caching.StackExchangeRedis/
* Open package manager and execute command: Install-Package Microsoft.Extensions.Caching.StackExchangeRedis
* Update-Package -ProjectName Basket.API

**Implementation:**

* Created Entities folder and ShoppingCart and ShoppingCartItem classes with all fields.
* Created Repositories folder and BasketRepository interface and class with implementation.
* Created BasketController with CRUD APIs implementation.
* Register BasketRepository interface and class in startup class.
* Add docker-compose
  + Right click on project and Add “Container Orchestrator Support”.
  + Select Docker Compose.
  + Select Operating System as Linux
  + Added/updated basketdb & basket.api configuration in docker-compose.yml and docker-compose.override.yml
  + Added portainer configuration to pull portainer/portainer-ce docker image in docker-compose.yml and docker-compose.override.yml
  + Run command: docker-compose -f .\docker-compose.yml -f .\docker-compose.override.yml up -d
  + Access portainer using url: <http://localhost:9000>

**Discount API Service:**

* Created folder structure for src/Services/Discount.
* Created new project Discount.API under src/Services/Discount.
* Created Discount.API profile under Properties => Debug with URL: [http://localhost:5002](http://localhost:5001)

**PostgreSQL Database:**

* Right clicked on docker-compose and Open in Terminal.
* Go to hub.docker.com and searched for postgres to get postgres pull image command
* Run Command: docker pull postgres
* Added discountdb configuration in docker-compose.yml and docker-compose.override.yml
* Added pgadmin configuration in docker-compose.yml and docker-compose.override.yml
* Run command: docker-compose -f .\docker-compose.yml -f .\docker-compose.override.yml up –d
* Access pgadmin using url: <http://localhost:5050/>
* Login with email address and password mentioned in docker-compose.override.yml file.
* Add Server
  + Set Name: DiscountServer
  + Go to connection tab
  + Enter discountdb as hostname
  + Enter username and password which is mentioned in docker-compose.override.yml file.
* Queries:
  + CREATE TABLE Coupon(

ID SERIAL PRIMARY KEY NOT NULL,

ProductName VARCHAR(24) NOT NULL,

Description TEXT,

Amount INT

);

* INSERT INTO coupon (ProductName, Description, Amount) VALUES ('IPhone X', 'IPhone Discount', 150)
* INSERT INTO coupon (ProductName, Description, Amount) VALUES ('Samsung 10, 'Samsung Discount', 100)
* SELECT \* FROM public.coupon ORDER BY id ASC

**Add Nuget Packages:**

* Open package manager and execute commands:
  + Install-Package Npgsql
  + Install-Package Dapper
  + Update-Package -ProjectName Discount.API

**Implementation:**

* Created Entities folder and Coupon class with all fields.
* Created Repositories folder and DiscountRepository interface and class with implementation.
* Created DiscountController with CRUD APIs implementation.
* Register DiscountRepository interface and class in startup class.
* Add docker-compose
  + Right click on project and Add “Container Orchestrator Support”.
  + Select Docker Compose.
  + Select Operating System as Linux
  + Added/updated discount.api configuration in docker-compose.yml and docker-compose.override.yml
  + Run command: docker-compose -f .\docker-compose.yml -f .\docker-compose.override.yml up -d

**Discount GRPC Service:**

* Created new project Discount.Grpc of Grpc type under src/Services/Discount.
* Created Discount.Grpc profile under Properties => Debug with URL: <http://localhost:5003>

**Add Nuget Packages:**

* Open package manager and execute commands:
  + Install-Package Npgsql
  + Install-Package Dapper
  + Update-Package -ProjectName Discount.Grpc
  + Install-Package AutoMapper.Extensions.Microsoft.DependencyInjection

**Implentation:**

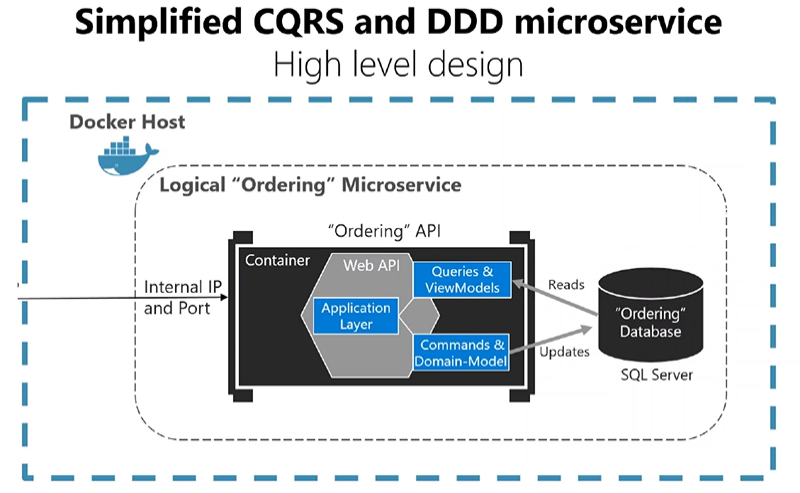
* Copy below folders from Discount.API to Discount.Grpc
  + Entities
  + Repositories
  + Extensions
* Update namespace in all copied classes. Replace Discount.API by Discount.Grpc manually.
* Register DiscountRepository interface and class in Startup class.
* Copy DatabaseSettings from Discount.API\appsettings.json to Discount.Grpc\appsettings.json
* Copy Main method from Discount.API\Program.cs to Discount.Grpc\Program.cs
* Created Coupon.proto file in Protos folder.
* Implemented all CRUD apis of discount in discount.proto file.
* Created DiscountService class in Services folder and implement DiscountProtoServiceBase class and override its methods.
* MapGrpcService DiscountService in Configure method in Startup class.
* Created Mapper folder and DiscountProfile class in it.

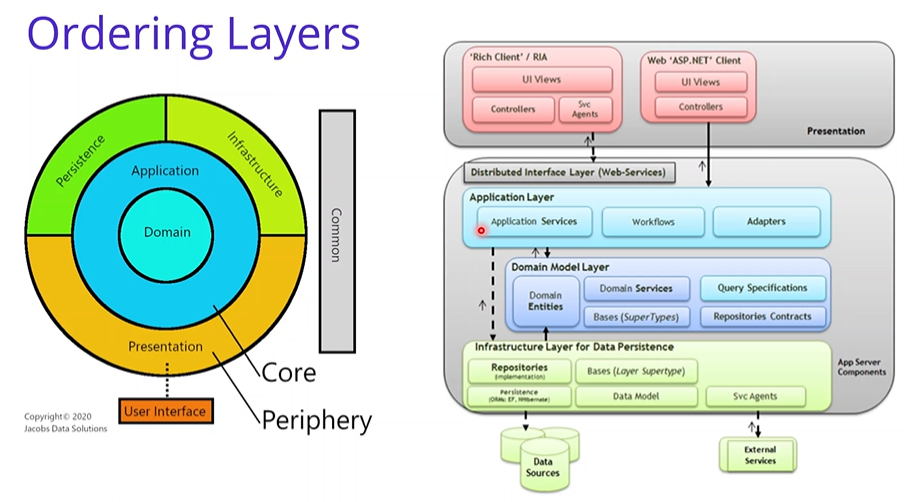
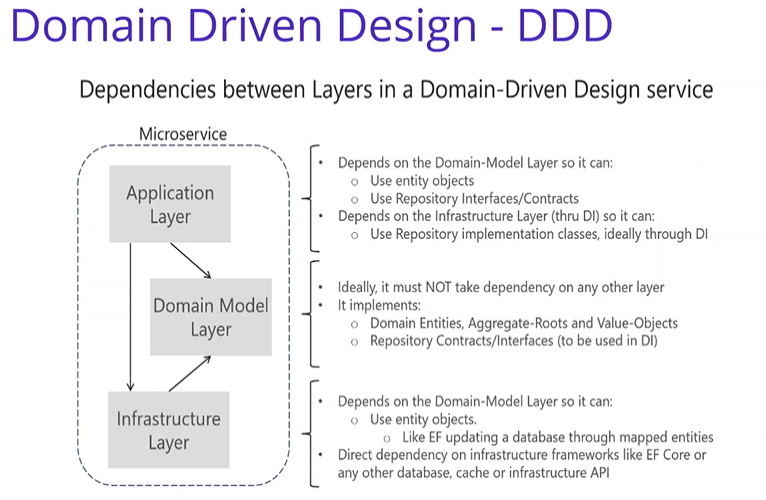
**Consume Discount Grpc Service into Basket API Service**

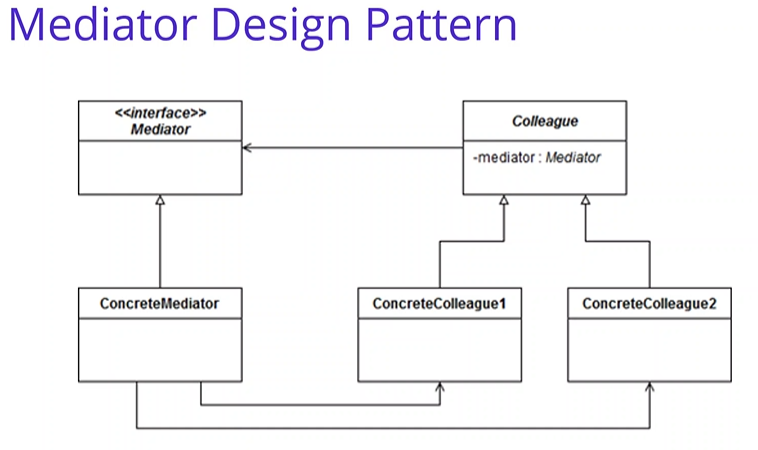
* Right click on Basket.API project and Add Connected Services.
* In Service References, Add gRPC
  + File: \aspnetcore-microservices\src\Services\Discount\Discount.Grpc\Protos\discount.proto
  + Type of class to be generated: client
* Created GrpcServices folder and DiscountGrpcService class in it.
* Implemented GetDiscount method in DiscountGrpcService class
* Consumed GetDiscount method of DiscountGrpcService in UpdateBasket action.
* AddGrpcClient<DiscountProtoService.DiscountProtoServiceClient> in startup class. Also registered DiscountGrpcService.
* Added GrpcSettings in appsettings.json.
* To test Basket.API with Discount.Grpc
  + Right click on Solution and select Properties.
  + Select Multiple Startup Projects
    - Select Start for Basket.API and Discount.Grpc projects.
    - Click Apply and Ok.
  + Click Start button to run these projects and test APIs.
* Add docker-compose
  + Right click on Discount.Grpc project and Add “Container Orchestrator Support”.
  + Select Docker Compose.
  + Select Operating System as Linux
  + Added/updated discount.grpc configuration in docker-compose.yml and docker-compose.override.yml
  + Added GrpcSetting in environment variable of basket.api in docker-compose.override.yml file.
  + Run command with build to rebuild images: docker-compose -f .\docker-compose.yml -f .\docker-compose.override.yml up --build

**Ordering API Service:**

* Created folder structure for src/Services/Ordering.
* Created new project Ordering.API under src/Services/Ordering.
* Created Ordering.API profile under Properties => Debug with URL: <http://localhost:5004>
* Created new class library project Ordering.Domain under src/Services/Ordering.
* Created new class library project Ordering.Application under src/Services/Ordering.
* Created new class library project Ordering.Infrastructure under src/Services/Ordering.
* Add Project References
  + Ordering.Application => Add Ordering.Domain reference.
  + Ordering.Infrastructure => Add Ordering.Application reference.
  + Ordering.API => Add Ordering.Application & Ordering.Infrastructure references.
* **Ordering.Domain:**
  + Created Common folder.
  + Created EntityBase and ValueObject classes in Common folder.
  + Created Entities folder and Order class in it with all fields.
* **Ordering.Application:**
  + Created Contracts, Features & Behaviours folders.
  + Created Persistence folder under Contracts folder.
  + Created IAsyncRepository & IOrderRepository classes in Persistence folder.
  + Created Infrastructure folder under Contracts folder.
  + Created IEmailService class in Infrastructure folder.
  + Created Models folder.
  + Created Email and EmailSettings classes in Models folder.
  + Created Mappings folder and MappingProfile class in it.
  + Created Orders folder under Feature folder.
  + Created Commands and Queries folders under Feature/Orders folder.
  + Created GetOrdersList folder under Queries folder.
  + Created GetOrdersListQuery, GetOrdersListQueryHandler & OrdersVm classes in GetOrdersList folder.
  + Install-Package MediatR.Extensions.Microsoft.DependencyInjection
  + Created CheckoutOrder folder under Command folder.
  + Created CheckoutOrderCommand, CheckoutOrderCommandHandler & CheckoutOrderCommandValidator classes in CheckoutOrder folder.
  + Created UpdateOrder folder under Command folder.
  + Created UpdateOrderCommand, UpdateOrderCommandHandler & UpdateOrderCommandValidator classes in UpdateOrder folder.
  + Created DeleteOrder folder under Command folder.
  + Created DeleteOrderCommand & DeleteOrderCommandHandler classes in DeleteOrder folder.
  + Created Exceptions folder.
  + Created NotFoundException & ValidationExeption classes in Exceptions folder.
  + Created ValidationBehaviour & UnhandledExceptionBehaviour classes in Behaviours folder.
  + Created ApplicationServiceRegistration extension class in Ordering.Application
  + Created OrderController in Ordering.API\Controller and implemented REST APIs.
  + Created Persistence folder in Ordering.Infrastructure project.
  + Created OrderContext and OrderContextSeed classes in Persistence folder.
  + Created Repositories folder in Ordering.Infrastructure project.
  + Created RepositoryBase and OrderRepository classes in repositories folder.
  + Created Mail folder in Ordering.Infrastructure project and EmailService class in it.
  + Created ApplicationServiceRegistration class in Ordering.Application and InfrastructureServiceRegistration class in Ordering.Infrastructure project.
  + Install-Package Microsoft.EntityFrameworkCore.Tools in Ordering.Infrastructure
  + Add-Migration InitialCreate in Ordering.Infrastructure
  + Create Extensions folder and HostExtensions class in it.
  + Modify Program class to use HostExtensions method.
  + Add docker-compose
    - Right click on Ordering.API project and Add “Container Orchestrator Support”.
    - Select Docker Compose.
    - Select Operating System as Linux
    - Added/updated ordering.api configuration in docker-compose.yml and docker-compose.override.yml
    - Added ConnectionString in environment variable of ordering.api in docker-compose.override.yml file.
    - Run command: docker-compose -f .\docker-compose.yml -f .\docker-compose.override.yml up --d







**Microservices async communication with RabbitMQ & MassTransit:**

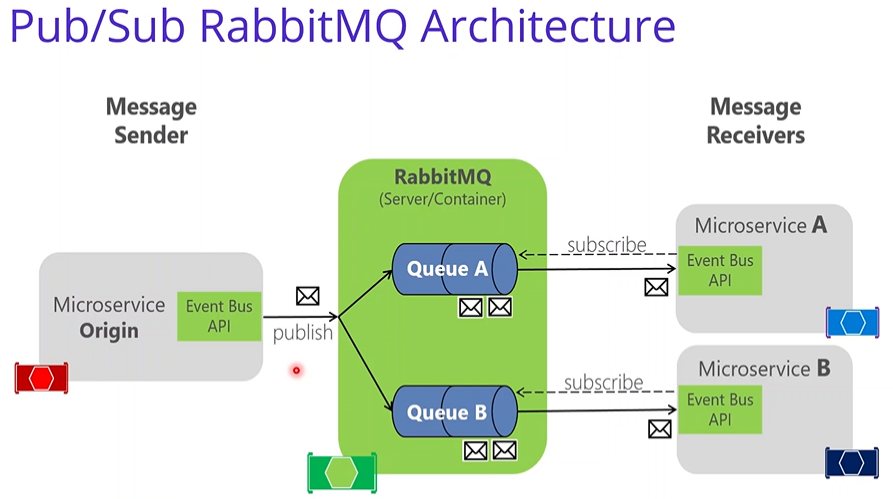
* Added rabbitmq configuration in docker-compose.yml and docker-compose.override.yml
* Run command: docker-compose -f .\docker-compose.yml -f .\docker-compose.override.yml up --d
* Access RabbitMQ portal using url: <http://localhost:15672/> with default username/password as guest/guest.

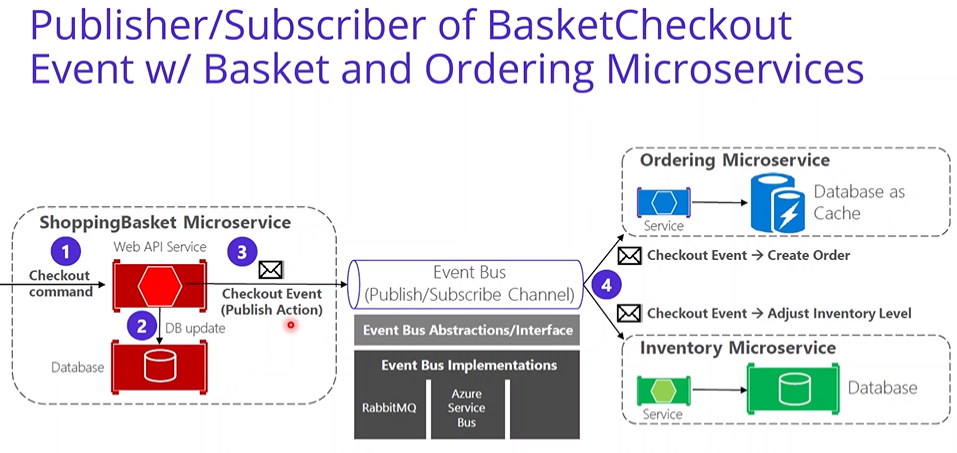
**Add Nuget Packages:**

* Open package manager, select Basket.API project and execute commands:
  + Install-Package MassTransit
  + Install-Package MassTransit.RabbitMQ
  + Install-Package MassTransit.AspNetCore
  + Install-Package AutoMapper.Extensions.Microsoft.DependencyInjection
  + Update-Package -ProjectName Basket.API
* Open package manager, select Ordering.API project and execute commands:
  + Install-Package MassTransit
  + Install-Package MassTransit.RabbitMQ
  + Install-Package MassTransit.AspNetCore
  + Install-Package AutoMapper
  + Install-Package AutoMapper.Extensions.Microsoft.DependencyInjection

**Implentation:**

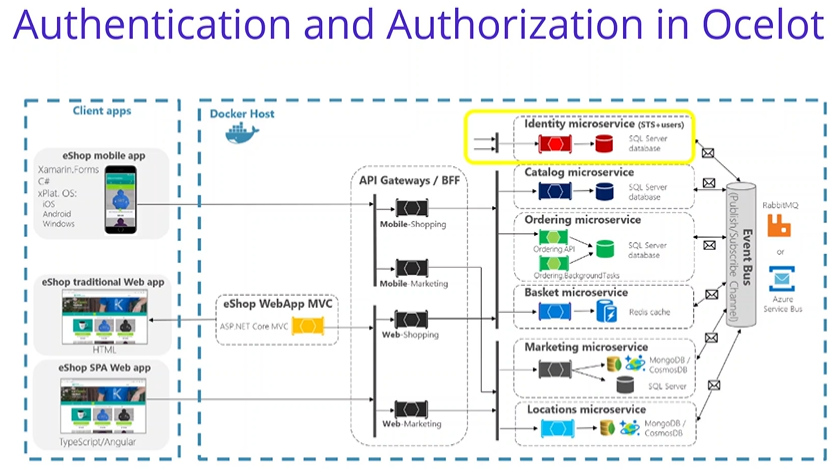
* Created BuildingBlocks folder in Solution.
* Created EventBus.Messages class library project in BuildingBlocks folder.
* Created Events folder under EventBus.Messages project and also created IntegrationBaseEvent & BasketCheckoutEvent classes in it.
* Created Common folder and EventBusConstants class in it.
* Basket.API => Add Project Reference => EventBus.Messages
* Added MassTransit-RabbitMQ configuration in startup class of Basket.API
* Created Checkout action in BasketController.
* Created Mapper folder in Basket.API project and BasketProfile class in it.
* Ordering.API => Add Project Reference => EventBus.Messages
* Added MassTransit-RabbitMQ configuration in startup class of Ordering.API
* Created Mappings folder under Ordering.API project and OrderingProfile class in it.
* To update dockerfile:
  + Delete dockerfile from Basket.API & Ordering.API
  + Right click on Basket.API & Ordering.API and select Docker Support.





**API Gateways with Ocelot:**

* Created ApiGateways folder in solution.
* Created OcelotApiGw ASP.Net Core empty project.
* Created ocelot.json, ocelot.Development.json & ocelot.Local.json
* Defined Routes in ocelot.Local.json file.
* Defined RateLimitOptions for Basket API.
* Install-Package Ocelot.Cache.CacheManager
* Configured some api with Cache option.
* Created OcelotApiGw profile under Properties => Debug with URL: <http://localhost:5010>
* Defined Routes in ocelot.Development.json file with docker configs.
* Add docker-compose
  + Right click on OcelotApiGw project and Add “Container Orchestrator Support”.
  + Select Docker Compose.
  + Select Operating System as Linux
  + Added/updated ocelotapigw configuration in docker-compose.yml and docker-compose.override.yml



**API Gateways – Requests Aggregation Pattern:**

* Created Shopping.Aggregator aspnet core web api project in ApiGateways folder.
* Created Models folder and below classes in it.
  + CatalogModel
  + BasketModel
  + BasketItemExtendedModel
  + OrderResponseModel
  + ShoppingModel
* Created Services folder and below interfaces & classes in it.
  + ICatalogService
  + CatalogService
  + IBasketService
  + BasketService
  + IOrderService
  + OrderService
* Created Extensions folder and HttpClientExtensions class in it.
* Created ShoppingController class in Controller folder and implemented GetShopping method in it.
* Created Shopping.Aggregator profile under Properties => Debug with URL: <http://localhost:5005>
* Add docker-compose
  + Right click on Shopping.Aggregator project and Add “Container Orchestrator Support”.
  + Select Docker Compose.
  + Select Operating System as Linux
  + Added/updated shopping.aggregator configuration in docker-compose.yml and docker-compose.override.yml

**WebApps:**

* Created WebApps folder in src folder.
* Downloaded readymade project from github.
  + <https://github.com/aspnetrun/run-aspnetcore-basics>
* Add downloaded project as existing project in our project.
* Delete Data, Entities, Migration and Repositories folders.
* Remove all nugget packages from project file.
* Removed connection string from appsettings and added ApiSettings.
* Created Models folder and created below classes in it.
  + BasketModel
  + BasketItemModel
  + BasketCheckoutModel
  + CatalogModel
  + OrderResponseModel
* Created Services folder and below interface & classes in it.
  + IBasketService
  + BasketService
  + ICatalogService
  + CatalogService
  + IOrderService
  + OrderService
* Created Extensions folder and HttpClientExtensions class in it.
* Implemented and resolved all issues in Pages.
* Updated Program and Startup classes.
* Created AspnetRunBasics profile under Properties => Debug with URL: <http://localhost:5006>
* Add docker-compose
  + Right click on AspnetRunBasics project and Add “Container Orchestrator Support”.
  + Select Docker Compose.
  + Select Operating System as Linux
  + Added/updated aspnetrunbasics configuration in docker-compose.yml and docker-compose.override.yml

**Microservices Observability with Distrubuted Logging with ElasticSearch:**

* Added elasticsearch & kibana configuration in docker-compose.yml and docker-compose.override.yml
* Created Common.Logging class library project in BuildingBlocks folder.
* Created SeriLogger class and implemented with Configure.
* Added Serilog & ElasticConfiguration in AppSettings of AspNetRunBasics project.
* Created LoggingDelegatingHandler class in Common.Logging project and implemented it.

**Add Nuget Packages:**

* Open package manager, select Common.Logging project and execute commands:
  + Install-Package Serilog.AspNetCore
  + Install-Package Serilog.Enrichers.Environment
  + Install-Package Serilog.Sinks.Elasticsearch

**Implentation:**

* AspNetRunBasics
  + Added Common.Logging project reference
  + Configured UseSerilog on CreateDefaultBuilder method in Program class.
  + Added Serilog and ElasticConfiguration in appsettings.json
  + Added LoggingDelegatingHandler as AddHttpMessageHandler in Startup class for all http client.
  + Delete dockerfile and Add => Docker Support => Linux.
* Shopping.Aggregator
  + Added Common.Logging project reference
  + Configured UseSerilog on CreateDefaultBuilder method in Program class.
  + Added Serilog and ElasticConfiguration in appsettings.json
  + Added LoggingDelegatingHandler as AddHttpMessageHandler in Startup class for all http client.
  + Delete dockerfile and Add => Docker Support => Linux.
* Catalog.API
  + Added Common.Logging project reference
  + Configured UseSerilog on CreateDefaultBuilder method in Program class.
  + Added Serilog and ElasticConfiguration in appsettings.json
  + Delete dockerfile and Add => Docker Support => Linux.
* Basket.API
  + Added Common.Logging project reference
  + Configured UseSerilog on CreateDefaultBuilder method in Program class.
  + Added Serilog and ElasticConfiguration in appsettings.json
  + Delete dockerfile and Add => Docker Support => Linux.
* Discount.API
  + Added Common.Logging project reference
  + Configured UseSerilog on CreateDefaultBuilder method in Program class.
  + Added Serilog and ElasticConfiguration in appsettings.json
  + Delete dockerfile and Add => Docker Support => Linux.
* Discount.Grpc
  + Added Common.Logging project reference
  + Configured UseSerilog on CreateDefaultBuilder method in Program class.
  + Added Serilog and ElasticConfiguration in appsettings.json
  + Delete dockerfile and Add => Docker Support => Linux.
* Ordering.API
  + Added Common.Logging project reference
  + Configured UseSerilog on CreateDefaultBuilder method in Program class.
  + Added Serilog and ElasticConfiguration in appsettings.json
  + Delete dockerfile and Add => Docker Support => Linux.
* OcelotApiGw
  + Added Common.Logging project reference
  + Configured UseSerilog on CreateDefaultBuilder method in Program class.
  + Added Serilog and ElasticConfiguration in appsettings.json
  + Delete dockerfile and Add => Docker Support => Linux.
* Added ElasticConfiguration environment variable in docker-compose.override.yml for all required services.

**Microservices Resilience and Fault Tolerance using Polly:**

* Install-Package Microsoft.Extensions.Http.Polly in Shopping.Aggregator & AspNetRunBasics projects.
* Install-Package Polly in Ordering.API, Discount.API & Discount.Grpc projects.
* Implemented Retry and CircuitBreaker policies in Startup class of Shopping.Aggregator & AspNetRunBasics projects.
* Added PolicyHandler on HttpClient.
* Implemented Retry policy in HttpHostExtensions class in Ordering.API, Disocunt.API & Discount.Grpc projects.

**Resources:**

* <https://github.com/>
* <https://hub.docker.com>
* <https://www.nuget.org/packages>
* <https://docs.portainer.io/v/ce-2.11/start/intro> (admin/admin@123)
* <https://www.pgadmin.org/>
* https:// medium.com/aspnetrun/microservices-architecture-on-net-3b4865eea03f
* <https://github.com/aspnetrun/run-aspnetcore-microservices>
* <https://github.com/mehmetozkaya/AspnetMicroservices>
* <https://github.com/aspnetrun/run-aspnet-grpc>
* <https://medium.com/aspnetrun/cqrs-and-event-sourcing-in-event-driven-architecture-of-ordering-microservices-fb67dc44da7a>
* <https://medium.com/software-alchemy/a-brief-intro-to-clean-architecture-clean-ddd-and-cqrs-23243c3f31b3>
* <https://github.com/jasontaylordev/CleanArchitecture>
* <https://www.youtube.com/watch?app=desktop&v=5OtUm1BLmG0mG0>
* <https://twitter.com/gillcleeren/status/1367863587163766787>
* <https://www.youtube.com/watch?v=BxtHt7tsX-c&t=4s>
* <https://docs.microsoft.com/en-us/aspnet/core/fundamentals/http-requests?view=aspnetcore-5.0>
* <https://docs.microsoft.com/en-us/dotnet/architecture/microservices/implement-resilient-applications/use-httpclientfactory-to-implement-resilient-http-requests>
* <https://stackoverflow.com/questions/40027299/where-is-the-postasjsonasync-method-in-asp-net-core>
* <https://github.com/aspnetrun/run-aspnet-identityserver4>
* <https://medium.com/aspnetrun/securing-microservices-with-identityserver4-with-oauth2-and-openid-connect-fronted-by-ocelot-api-49ea44a0cf9e>
* <https://github.com/Shop-Microservices/AspnetMicroservices>
* <https://github.com/mansoorafzal/SecureMicroservices>
* <https://github.com/aspnetrun/run-aspnetcore-basics>
* <https://github.com/aspnetrun/run-devops>
* <https://github.com/felipecembranelli/run-aspnetcore-microservices>
* <https://medium.com/aspnetrun/microservices-observability-resilience-monitoring-on-net-a5dfbdbb0fbd>
* <https://medium.com/aspnetrun/microservices-observability-with-distributed-logging-using-elasticsearch-and-kibana-79df919997d2>
* <https://docs.microsoft.com/en-us/aspnet/core/fundamentals/logging/?view=aspnetcore-5.0>
* <https://github.com/thecarlo/elastic-kibana-netcore-serilog/blob/master/src/docker/docker-compose.yml>
* <https://github.com/mehmetozkaya/AspnetMicroservices_CrossCutting/blob/main/src/docker-compose.override.yml>
* <https://docs.microsoft.com/en-us/aspnet/web-api/overview/advanced/httpclient-message-handlers>
* <https://medium.com/aspnetrun/microservices-resilience-and-fault-tolerance-with-applying-retry-and-circuit-breaker-patterns-c32e518db990>
* <https://github.com/App-vNext/Polly>
* <https://docs.microsoft.com/en-us/dotnet/architecture/cloud-native/>
* <https://docs.microsoft.com/en-us/dotnet/architecture/microservices/implement-resilient-applications/use-httpclientfactory-to-implement-resilient-http-requests>
* <https://github.com/App-vNext/Polly/wiki/Polly-and-HttpClientFactory#using-polly-with-ihttpclientfactory>
* <https://medium.com/aspnetrun/microservices-monitoring-with-health-checks-using-watchdog-6b16fdae0349>
* <https://github.com/Xabaril/AspNetCore.Diagnostics.HealthChecks#ui-storage-providers>
* <https://docs.microsoft.com/en-us/dotnet/architecture/microservices/implement-resilient-applications/monitor-app-health>
* <https://github.com/Xabaril/AspNetCore.Diagnostics.HealthChecks/blob/master/src/HealthChecks.UI/Configuration/Options.cs>
* <https://devblogs.microsoft.com/dotnet/improvements-in-net-core-3-0-for-troubleshooting-and-monitoring-distributed-apps/>
* <https://devblogs.microsoft.com/dotnet/monitoring-and-observability-in-cloud-native-asp-net-core-apps/>
* <https://github.com/mansoorafzal/AspnetMicroservices>