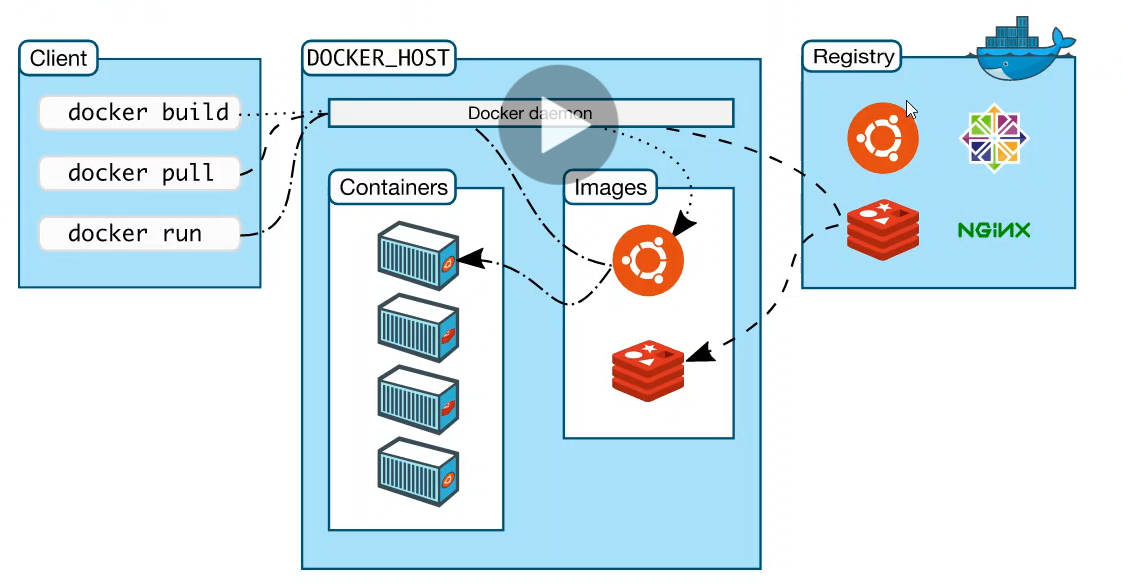
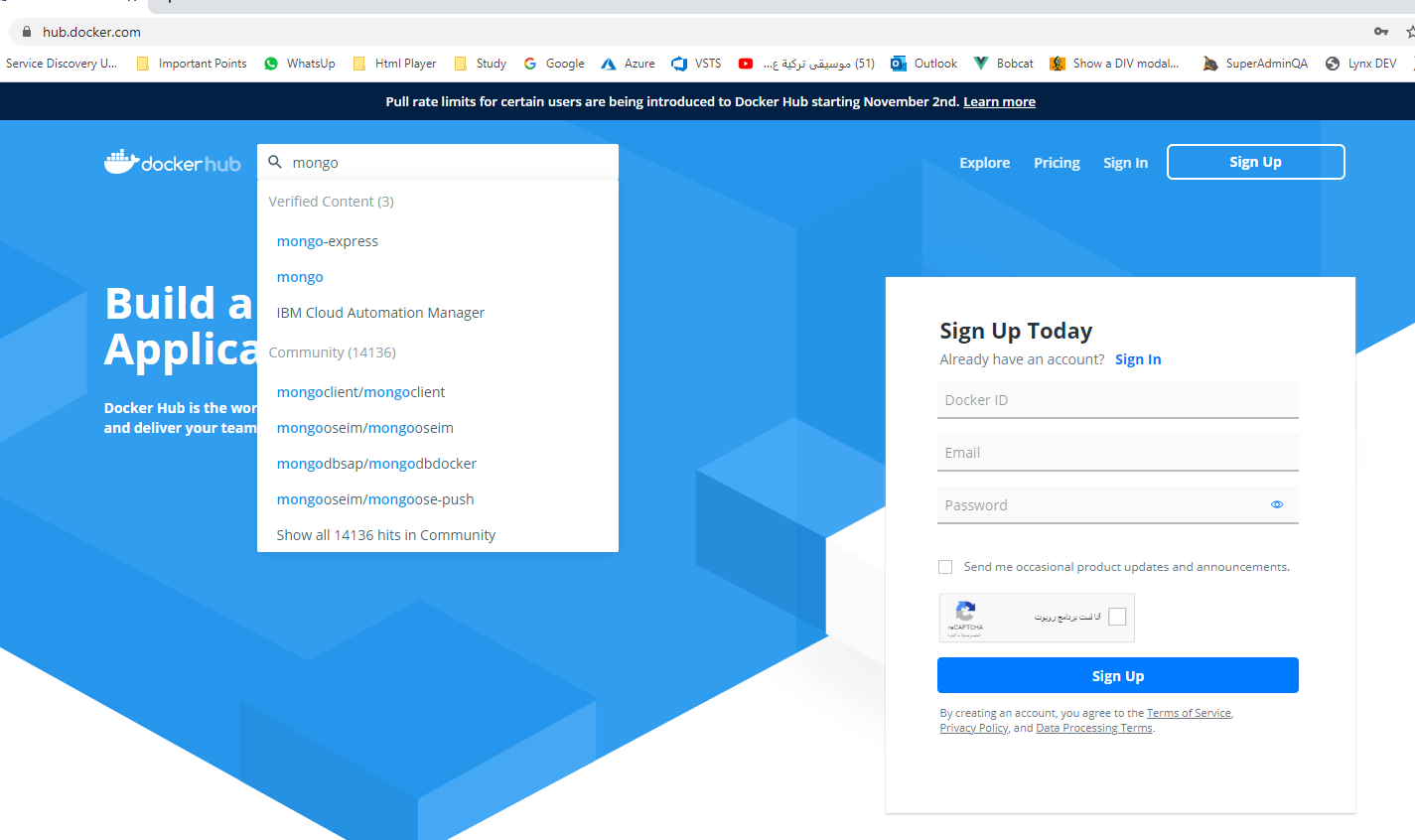
**Section0 building Catalog Micro services**

**Notes:-**

**1-we see in the Docker architecture we have Docker hub which contain the docker registry that holds all the online images and we have docker host which hold the images that can hold multiple containers as below**

**2-to download online image such as mongo go to the website URL** [**https://hub.docker.com/**](https://hub.docker.com/) **and**



**3-on the mongo you will see that you run the following command to download mongodb image**

**docker pull mongo //to download mongo-db image**

**docker images //to get list of the images**

**docker ps //to get all the containers running**

**//it will run the mongo-db image as container on the port 27017**

**docker run –d –p 27017:27017 --name aspnetrun-mongo mongo**

**//to enter to the mongo image we type the below two commands as below**

**Use**

**mongo**

**Commands in mongo**

**//to show all database**

**show dbs**

**//switch and create new database if not exist but not shown until create new collection**

**use CatalogDb**

**db.createCollection(‘Prodcuts’)**

**//it will insert two records to the collection 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}])**

**//it will get all products in pretty JSON format**

**db.Products.find({}).pretty()**

**//to remove all items from products collection**

**db.Products.remove({})**

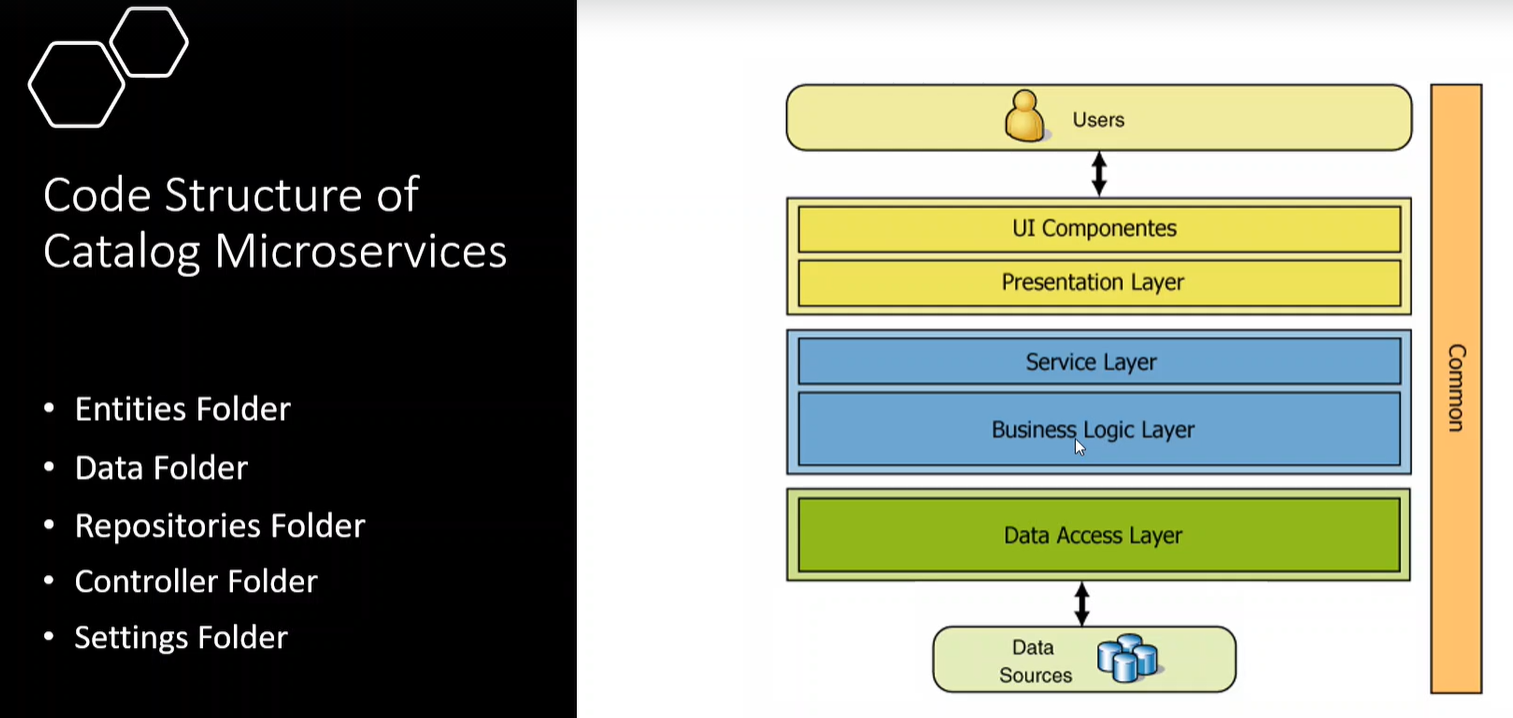
**//to show all database**

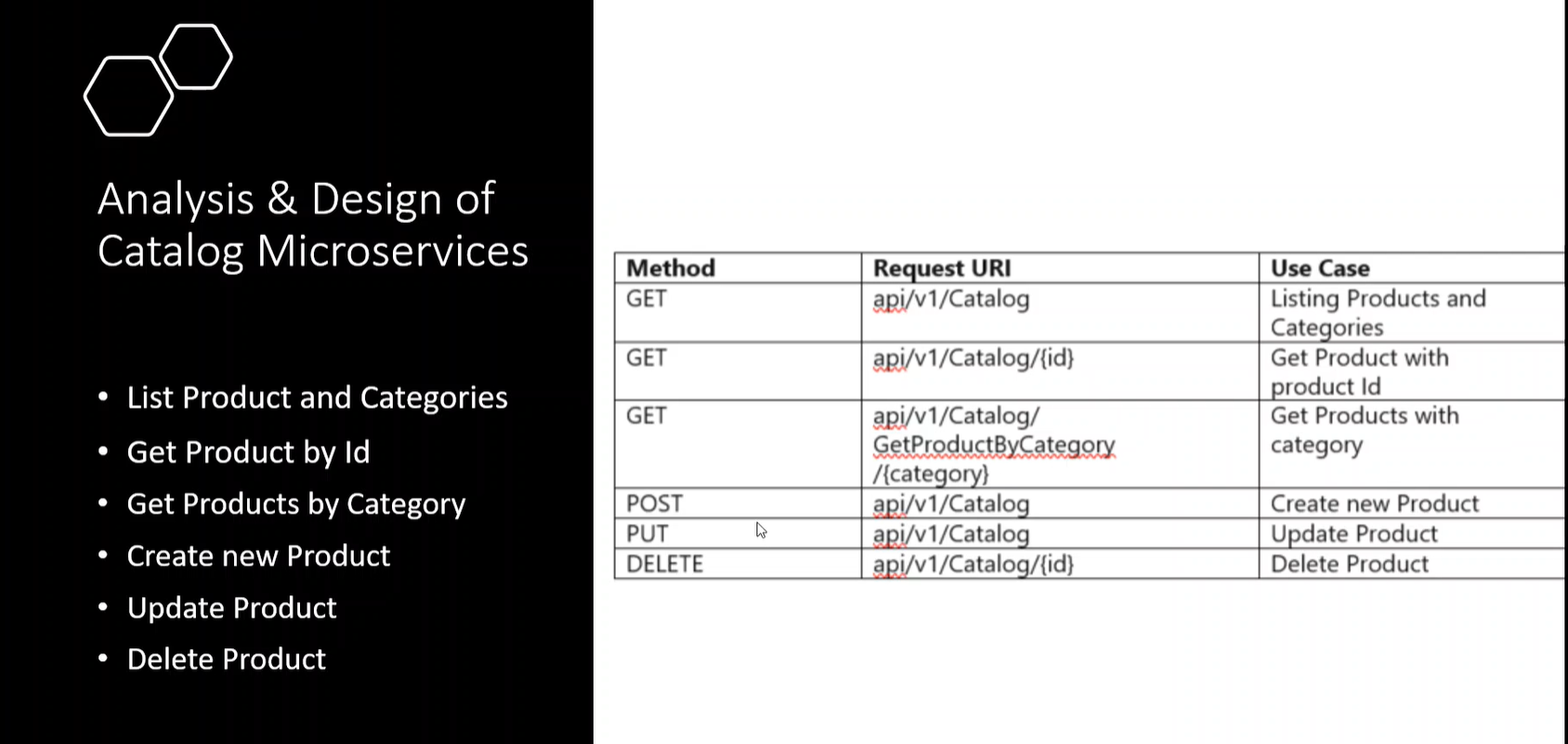
**show dbs**

**//to show all collection**

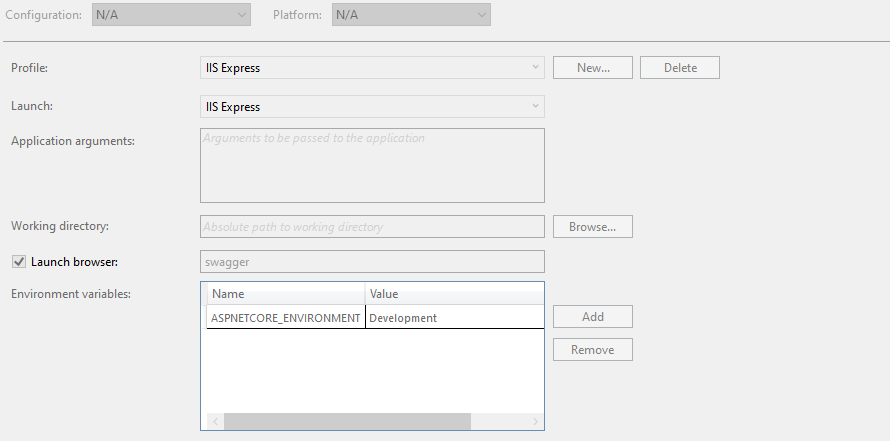
**show collections**

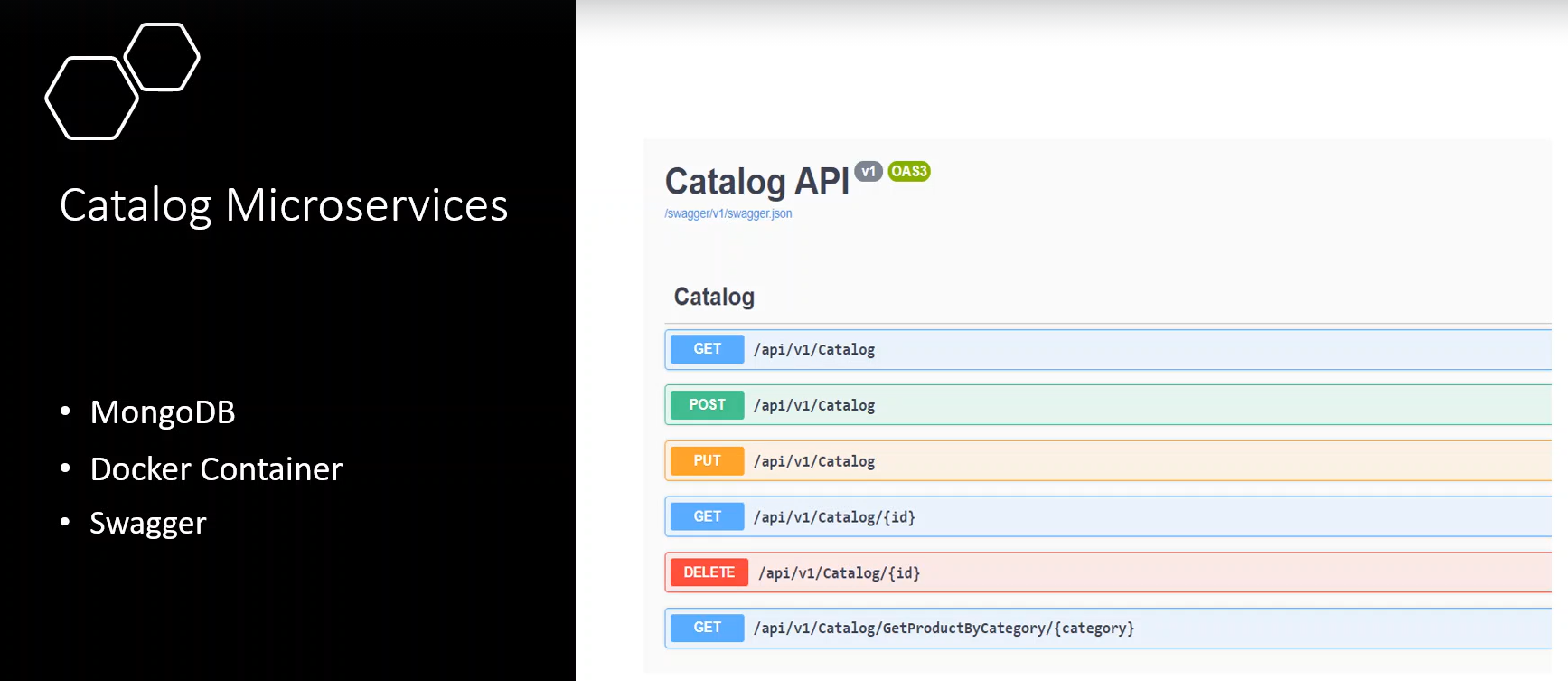
**Code Structure in Catalog Micro services**



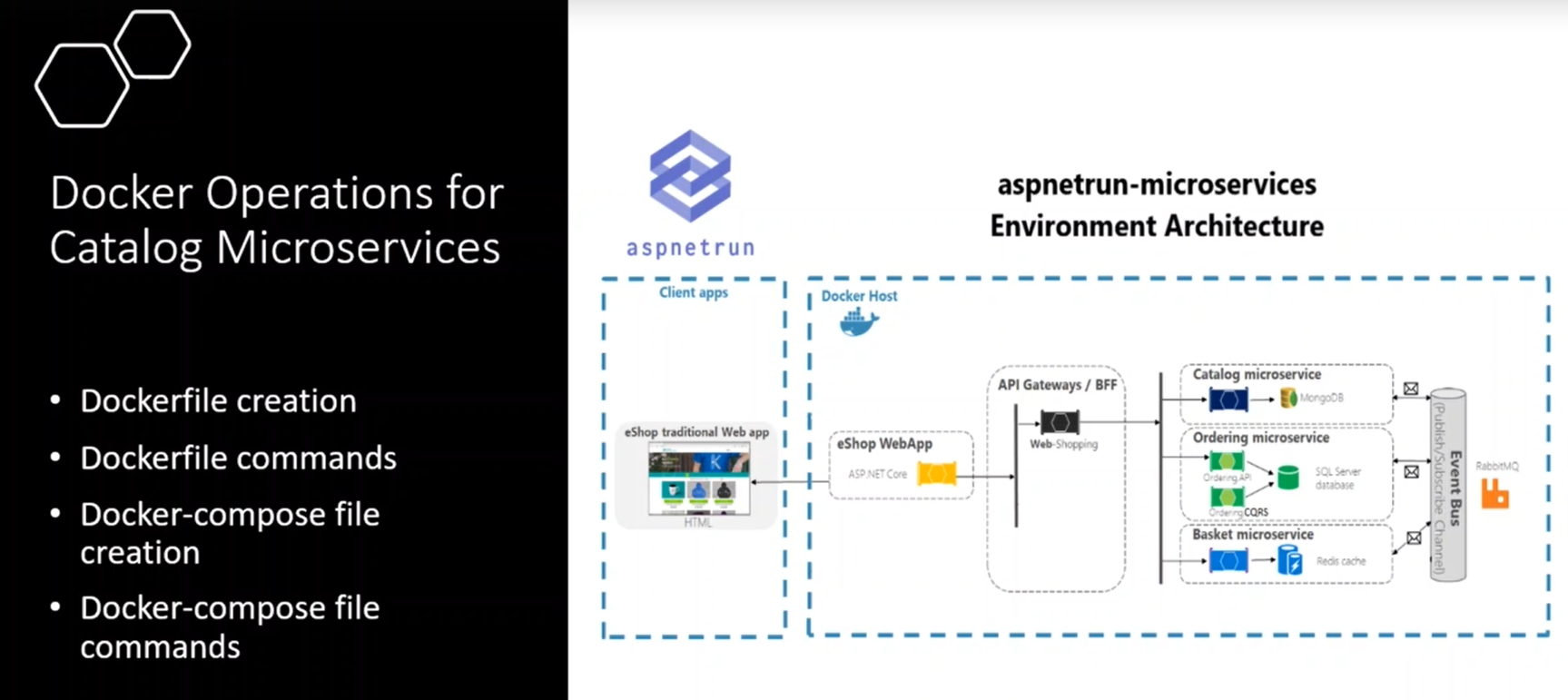


**To set the startup launch page or part you can configure it in the project properties as below**



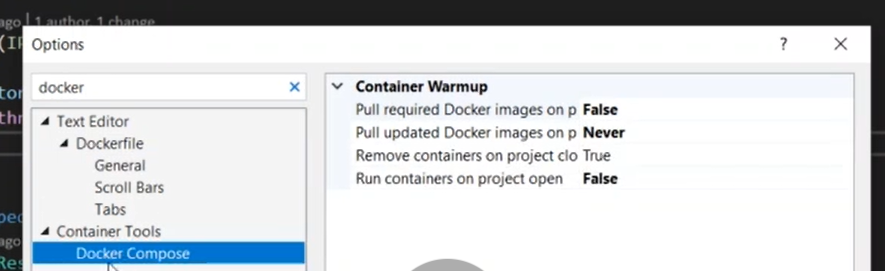


**Docker Compose Configuration and generation**



**Steps:-**

**1-on vs 2019 🡪 Tools 🡪 type docker and set**



**2-on the project 🡪 right click 🡪 Add 🡪 (There are two options)**

**A-Container Orchestrator Support 🡪 which create docker file and docker compose configuration file**

**B-Docker Support 🡪 which only create docker compose configuration part**

**3-Choose Docker Compose and then select Linux**

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

**Docker file: include the configuration of the project and dependencies and publish parts and the base image that will build from it which is aspnet core 3.1**

**In Docker file we set the following configuration**

#See https://aka.ms/containerfastmode to understand how Visual Studio uses this Dockerfile to build your images for faster debugging.

#it will take the aspnet 5.0 from Docker repository online and configure the port that working with

FROM mcr.microsoft.com/dotnet/aspnet:5.0-buster-slim AS base

WORKDIR /app

EXPOSE 80

#it will take the aspnet 5.0 and build the image and set the working directory

FROM mcr.microsoft.com/dotnet/sdk:5.0-buster-slim AS build

WORKDIR /src

#it will copy the Catalog.API.csproj and paste it in the Catalog.API

COPY ["src/Catalog/Catalog.API/Catalog.API.csproj", "src/Catalog/Catalog.API/"]

#it will run and generate image from the folder the copy the csproj

RUN dotnet restore "src/Catalog/Catalog.API/Catalog.API.csproj"

#copy the generated iamge to the docker registry

COPY . .

#set the working directory that make build the container from that image

WORKDIR "/src/src/Catalog/Catalog.API"

#build the generated container

RUN dotnet build "Catalog.API.csproj" -c Release -o /app/build

#run the generated container

FROM build AS publish

RUN dotnet publish "Catalog.API.csproj" -c Release -o /app/publish

#set entry point for the iamge and run the image and set the entry point as Catalog.API.dll

FROM base AS final

WORKDIR /app

COPY --from=publish /app/publish .

ENTRYPOINT ["dotnet", "Catalog.API.dll"]

**In Docker.compose.yml file we set the following configuration**

version: '3.4'

services:

#declare the image for the catalogdb that represent the mongodb image that downloaded from docker hub

catalogdb:

image: mongo

#declare the catalog api image that refer to local docker file configuration in Catalog.API Project

catalog.api:

image: ${DOCKER\_REGISTRY-}catalogapi

build:

context: .

dockerfile: src/Catalog/Catalog.API/Dockerfile

**In Docker-compose.override.yml we set the following code**

version: '3.4'

services:

#define the catalogdb configuration configuration (container name,port,restart state)

#volumes represnet the storage folder for the image in local machine

catalogdb:

container\_name: catalogdb

restart: always

volumes:

- ${WEBAPP\_STORAGE\_HOME}/site:/data/db

#- ./data:/data/db

ports:

- "27017:27017"

#we see that the microservice that use mongodb we provide the connection string

#define the catalog.api configuration such as port number , dependencies like catalogdb,volumes

catalog.api:

container\_name: catalogapi

environment:

- ASPNETCORE\_ENVIRONMENT=Development

- "CatalogDbSettings:ConnectionString=mongodb://catalogdb:27017"

depends\_on:

- catalogdb

volumes:

- ${HOME}/.microsoft/usersecrets/:/root/.microsoft/usersecrets

- ${HOME}/.aspnet/https:/root/.aspnet/https/

ports:

- "8000:80"

**Running Command for the Docker Compose**

**Steps:-**

**1-open the folder contains the docker-compose.yml by command prompt and type the following commands as below**

**//it will execute the docker-compose.yml with apply docker-compose.override.yml**

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

**Steps:-**

**1-on the appsettings.json we set the following code**

**{**

**"CatalogDbSettings": {**

**"ConnectionString": "mongodb://localhost:27017",**

**"DatabaseName": "CatalogDb",**

**"CollectionName": "Products"**

**},**

**"Logging": {**

**"LogLevel": {**

**"Default": "Information",**

**"Microsoft": "Warning",**

**"Microsoft.Hosting.Lifetime": "Information"**

**}**

**},**

**"AllowedHosts": "\*"**

**}**

**2-on the startup.cs we set the following code**

**public class Startup{**

**public Startup(IConfiguration configuration){Configuration = configuration;}**

**public IConfiguration Configuration { get; }**

**// This method gets called by the runtime. Use this method to add services to the container.**

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

**services.AddControllers();**

**//it will generate swagger with detect the version with set title and using it over the whole application**

**services.AddSwaggerGen(c =>{c.SwaggerDoc("v1", new OpenApiInfo { Title = "Catalog.API", Version = "v1" });});**

**//it will map the CatalogDbSettings that exist in the appsettings.json into the CatalogDbSettings**

**services.Configure<CatalogDbSettings>(Configuration.GetSection(nameof(CatalogDbSettings)));**

**//when you are seeing the ICatalogDatabaseSettings run the below command in any constructor get the values from appSettings.json**

**services.AddSingleton<ICatalogDbSettings>(sp => sp.GetRequiredService<IOptions<CatalogDbSettings>>().Value);**

**//if you see and ICatalogContext on any constructor on controller crete new instance of the CatalogContext as below**

**services.AddTransient<ICatalogContext, CatalogContext>();**

**//if you inject IProductRepository in the constructor ,create new instance of the ProductRepository**

**services.AddTransient<IProductRepository, ProductRepository>();}**

**// This method gets called by the runtime. Use this method to configure the HTTP request pipeline.**

**public void Configure(IApplicationBuilder app, IWebHostEnvironment env){**

**if (env.IsDevelopment()){**

**app.UseDeveloperExceptionPage();**

**//it will use swagger with set the EndPoint swagger.json**

**app.UseSwagger();**

**app.UseSwaggerUI(c => c.SwaggerEndpoint("/swagger/v1/swagger.json", "Catalog.API v1"));}**

**app.UseRouting();**

**app.UseAuthorization();**

**app.UseEndpoints(endpoints =>{endpoints.MapControllers();});}}**

**3-in appsettings.json we set the following code**

**//we configure the connection settings**

**{**

**"CatalogDbSettings": {**

**"ConnectionString": "mongodb://localhost:27017",**

**"DatabaseName": "CatalogDb",**

**"CollectionName": "Products"**

**},**

**"Logging": {**

**"LogLevel": {**

**"Default": "Information",**

**"Microsoft": "Warning",**

**"Microsoft.Hosting.Lifetime": "Information"**

**}**

**},**

**"AllowedHosts": "\*"**

**}**

**4-on the CatalogAPI > Settings > create ICatalogDbSettings,CatalogDbSettings**

**public interface ICatalogDbSettings{**

**public string ConnectionString { get; set; }**

**public string DatabaseName { get; set; }**

**public string CollectionName { get; set; }}**

**public class CatalogDbSettings : ICatalogDbSettings{**

**public string ConnectionString { get; set; }**

**public string DatabaseName { get; set; }**

**public string CollectionName { get; set; }}**

**5-on the Repositories > Interface > create IProductRepository**

**public interface IProductRepository{**

**Task<IEnumerable<Product>> GetProducts();**

**Task<Product> GetProduct(string id);**

**Task<IEnumerable<Product>> GetProductName(string name);**

**Task<IEnumerable<Product>> GetProductByCategory(string categoryName);**

**Task Create(Product product);**

**Task<bool> Update(Product product);**

**Task<bool> Delete(string id);}**

**6-on the Repositories > ProductRepository we apply the interface IProductRepository**

**public class ProductRepository : IProductRepository{**

**//we will injection ICatalogContext to allow access to the Mongo database**

**private readonly ICatalogContext \_context;**

**public ProductRepository(ICatalogContext context){**

**\_context = context ?? throw new ArgumentNullException(nameof(context));}**

**public async Task<IEnumerable<Product>> GetProducts(){**

**//it will get all products without filtering**

**return await \_context.Products.Find(p => true).ToListAsync();}**

**public async Task<Product> GetProduct(string id){**

**return await \_context.Products.Find(p => p.Id == id).FirstOrDefaultAsync();}**

**public async Task<IEnumerable<Product>> GetProductName(string name){**

**var filter = Builders<Product>.Filter.Eq(p => p.Name , name);**

**return await \_context.Products.Find(filter).ToListAsync();}**

**//in order to make contains we have to change the filter with using regex with categroy name**

**public async Task<IEnumerable<Product>> GetProductByCategory(string category){**

**var filter = new BsonDocument { { "Category", new BsonDocument { { "$regex", category }, { "$options", "i" } } } };**

**return await \_context.Products.Find(filter).ToListAsync();}**

**public async Task Create(Product product){await \_context.Products.InsertOneAsync(product);}**

**public async Task<bool> Update(Product product){**

**//it will get the old json object and replace it with the new object**

**var updateReulst = await \_context.Products.ReplaceOneAsync(filter: g => g.Id == product.Id, replacement: product);**

**return updateReulst.IsAcknowledged && updateReulst.ModifiedCount > 0;}**

**public async Task<bool> Delete(string id){**

**FilterDefinition<Product> filter = Builders<Product>.Filter.Eq(p => p.Id,id);**

**DeleteResult deleteResult = await \_context.Products.DeleteOneAsync(filter);**

**return deleteResult.IsAcknowledged && deleteResult.DeletedCount > 0;}}**

**7-on the CatalogAPI > Entities > create class Product**

**public class Product{**

**//we make it string becouse mongodb generate generate bsonID in the Id column**

**//it will make auto generation of the Id column such as primary key in the MongoDB**

**[BsonId]**

**[BsonRepresentation(BsonType.ObjectId)]**

**public string Id { get; set; }**

**[BsonElement("Name",Order =2)]**

**public string Name { get; set; }**

**public string Summary { get; set; }**

**public string Description { get; set; }**

**[BsonElement("Category", Order = 5)]**

**public string Category { get; set; }**

**public string ImageFile { get; set; }**

**public decimal Price { get; set; }}**

**8-on the CatalogAPI > Data > Interfaces > create interface ICatalogContext**

**public interface ICatalogContext{IMongoCollection<Product> Products { get; }}**

**//on the Data we create CatalogContext that apply the connection with mongo database**

**public class CatalogContext : ICatalogContext{**

**public CatalogContext(ICatalogDbSettings settings){**

**//by inject the ICatalogDbSettings that mapped in the configureservices that map to the mongo connection settings**

**var client = new MongoClient(settings.ConnectionString);**

**//we get the database and collection name to map with the IMongoCollection Products**

**var database = client.GetDatabase(settings.DatabaseName);**

**Products = database.GetCollection<Product>(settings.CollectionName);**

**//we call the seed data**

**CatalogContextSeed.SeedData(Products);}**

**public IMongoCollection<Product> Products { get; }}**

**9-on the Data > CatalogContextSeed we apply seeding**

**public class CatalogContextSeed{**

**public static void SeedData(IMongoCollection<Product> productCollection){**

**bool existProduct = productCollection.Find(p => true).Any();**

**if (!existProduct){productCollection.InsertManyAsync(GetPreConfiguredProducts());}}**

**private static IEnumerable<Product> GetPreConfiguredProducts(){**

**return new List<Product>(){**

**new Product(){**

**Name = "IPhone X",**

**Summary = "This phone is the company's biggest change to its flagship smartphone in years. It includes a borderless.",**

**Description = "Lorem ipsum dolor sit amet, consectetur adipisicing elit. Ut, tenetur natus doloremque laborum quos iste ipsum rerum obcaecati impedit odit illo dolorum ab tempora nihil dicta earum fugiat. Temporibus, voluptatibus. Lorem ipsum dolor sit amet, consectetur adipisicing elit. Ut, tenetur natus doloremque laborum quos iste ipsum rerum obcaecati impedit odit illo dolorum ab tempora nihil dicta earum fugiat. Temporibus, voluptatibus.",**

**ImageFile = "product-1.png",**

**Price = 950.00M,**

**Category = "Smart Phone"},**

**new Product(){**

**Name = "Samsung 10",**

**Summary = "This phone is the company's biggest change to its flagship smartphone in years. It includes a borderless.",**

**Description = "Lorem ipsum dolor sit amet, consectetur adipisicing elit. Ut, tenetur natus doloremque laborum quos iste ipsum rerum obcaecati impedit odit illo dolorum ab tempora nihil dicta earum fugiat. Temporibus, voluptatibus. Lorem ipsum dolor sit amet, consectetur adipisicing elit. Ut, tenetur natus doloremque laborum quos iste ipsum rerum obcaecati impedit odit illo dolorum ab tempora nihil dicta earum fugiat. Temporibus, voluptatibus.",**

**ImageFile = "product-2.png",**

**Price = 840.00M,**

**Category = "Smart Phone"},**

**new Product(){**

**Name = "Huawei Plus",**

**Summary = "This phone is the company's biggest change to its flagship smartphone in years. It includes a borderless.",**

**Description = "Lorem ipsum dolor sit amet, consectetur adipisicing elit. Ut, tenetur natus doloremque laborum quos iste ipsum rerum obcaecati impedit odit illo dolorum ab tempora nihil dicta earum fugiat. Temporibus, voluptatibus. Lorem ipsum dolor sit amet, consectetur adipisicing elit. Ut, tenetur natus doloremque laborum quos iste ipsum rerum obcaecati impedit odit illo dolorum ab tempora nihil dicta earum fugiat. Temporibus, voluptatibus.",**

**ImageFile = "product-3.png",**

**Price = 650.00M,**

**Category = "White Appliances"},**

**new Product(){**

**Name = "Xiaomi Mi 9",**

**Summary = "This phone is the company's biggest change to its flagship smartphone in years. It includes a borderless.",**

**Description = "Lorem ipsum dolor sit amet, consectetur adipisicing elit. Ut, tenetur natus doloremque laborum quos iste ipsum rerum obcaecati impedit odit illo dolorum ab tempora nihil dicta earum fugiat. Temporibus, voluptatibus. Lorem ipsum dolor sit amet, consectetur adipisicing elit. Ut, tenetur natus doloremque laborum quos iste ipsum rerum obcaecati impedit odit illo dolorum ab tempora nihil dicta earum fugiat. Temporibus, voluptatibus.",**

**ImageFile = "product-4.png",**

**Price = 470.00M,**

**Category = "White Appliances"},**

**new Product(){**

**Name = "HTC U11+ Plus",**

**Summary = "This phone is the company's biggest change to its flagship smartphone in years. It includes a borderless.",**

**Description = "Lorem ipsum dolor sit amet, consectetur adipisicing elit. Ut, tenetur natus doloremque laborum quos iste ipsum rerum obcaecati impedit odit illo dolorum ab tempora nihil dicta earum fugiat. Temporibus, voluptatibus. Lorem ipsum dolor sit amet, consectetur adipisicing elit. Ut, tenetur natus doloremque laborum quos iste ipsum rerum obcaecati impedit odit illo dolorum ab tempora nihil dicta earum fugiat. Temporibus, voluptatibus.",**

**ImageFile = "product-5.png",**

**Price = 380.00M,**

**Category = "Smart Phone"},**

**new Product(){**

**Name = "LG G7 ThinQ",**

**Summary = "This phone is the company's biggest change to its flagship smartphone in years. It includes a borderless.",**

**Description = "Lorem ipsum dolor sit amet, consectetur adipisicing elit. Ut, tenetur natus doloremque laborum quos iste ipsum rerum obcaecati impedit odit illo dolorum ab tempora nihil dicta earum fugiat. Temporibus, voluptatibus. Lorem ipsum dolor sit amet, consectetur adipisicing elit. Ut, tenetur natus doloremque laborum quos iste ipsum rerum obcaecati impedit odit illo dolorum ab tempora nihil dicta earum fugiat. Temporibus, voluptatibus.",**

**ImageFile = "product-6.png",**

**Price = 240.00M,**

**Category = "Home Kitchen"}};}}**

**10-on the CatalogController we implement the methods as below**

**[Route("api/v1/[controller]")]**

**//to conver this class as API controller you have inherit from ControllerBase and set attribute APiController**

**[ApiController]**

**public class CatalogController : ControllerBase{**

**// dsafsafdwqewqewqe**

**private readonly IProductRepository \_repository;**

**private readonly ILogger<CatalogController> \_logger;**

**public CatalogController(IProductRepository repository,ILogger<CatalogController> logger){**

**\_repository = repository ?? throw new ArgumentNullException(nameof(repository));**

**\_logger = logger ?? throw new ArgumentNullException(nameof(logger));}**

**//when open the product page**

**//to secure the action and make sure that return Product and return Ok result**

**[HttpGet]**

**[ProducesResponseType(typeof(IEnumerable<Product>),(int)HttpStatusCode.OK)]**

**public async Task<ActionResult> GetProductsAsync(){**

**var products = await \_repository.GetProducts();**

**return Ok(products);}**

**//go to the product details**

**//we make check and filtering the parameter passed to be string and max length character of 24 characters as below**

**[HttpGet("{id:length(24)}",Name = "GetProduct")]**

**[ProducesResponseType((int)HttpStatusCode.NotFound)]**

**[ProducesResponseType(typeof(IEnumerable<Product>), (int)HttpStatusCode.OK)]**

**public async Task<ActionResult> GetProductAsync(string id){**

**var product = await \_repository.GetProduct(id);**

**if (product is null){**

**\_logger.LogError($"Product with Id {id} not found");**

**return NotFound();}**

**return Ok(product);}**

**//search product by category searchbox**

**//there is another way to render routing name to be match the name of the action by set the [action]**

**[HttpGet]**

**[Route("[action]/{category}")]**

**[ProducesResponseType(typeof(IEnumerable<Product>), (int)HttpStatusCode.OK)]**

**public async Task<ActionResult> GetProductByCategory(string category){**

**var product = await \_repository.GetProductByCategory(category);**

**return Ok(product);}**

**//create product**

**[HttpPost]**

**[ProducesResponseType(typeof(IEnumerable<Product>), (int)HttpStatusCode.OK)]**

**public async Task<ActionResult> CreateProduct([FromBody] Product product){**

**await \_repository.Create(product);**

**//it will redirect as action in mvc and return the product object as below**

**return CreatedAtRoute("GetProduct", new { id = product.Id }, product);}**

**//update product**

**[HttpPut]**

**[ProducesResponseType(typeof(IEnumerable<Product>), (int)HttpStatusCode.OK)]**

**public async Task<ActionResult> UpdateProduct([FromBody] Product product){**

**return Ok(await \_repository.Update(product));}**

**//delete product**

**[HttpDelete("{id:length(24)}")]**

**[ProducesResponseType(typeof(Product), (int)HttpStatusCode.OK)]**

**public async Task<ActionResult> Delete(string id){**

**return Ok(await \_repository.Delete(id));}}**

**11-on the Docker-compose file we set the configuration as below**

catalogdb:

image: mongo

#declare the catalog api image that refer to local docker file configuration in Catalog.API Project

catalog.api:

image: ${DOCKER\_REGISTRY-}catalogapi

build:

context: .

dockerfile: src/Catalog/Catalog.API/Dockerfile

services:

#define the catalogdb configuration configuration (container name,port,restart state)

#volumes represnet the storage folder for the image in local machine

catalogdb:

container\_name: catalogdb

restart: always

volumes:

- ${WEBAPP\_STORAGE\_HOME}/site:/data/db

#- ./data:/data/db

ports:

- "27017:27017"

#we see that the microservice that use mongodb we provide the connection string

#define the catalog.api configuration such as port number , dependencies like catalogdb,volumes

catalog.api:

container\_name: catalogapi

restart: on-failure

environment:

- ASPNETCORE\_ENVIRONMENT=Development

- "CatalogDbSettings:ConnectionString=mongodb://catalogdb:27017"

depends\_on:

- catalogdb

volumes:

- ${HOME}/.microsoft/usersecrets/:/root/.microsoft/usersecrets

- ${HOME}/.aspnet/https:/root/.aspnet/https/

ports:

- "5000:80"