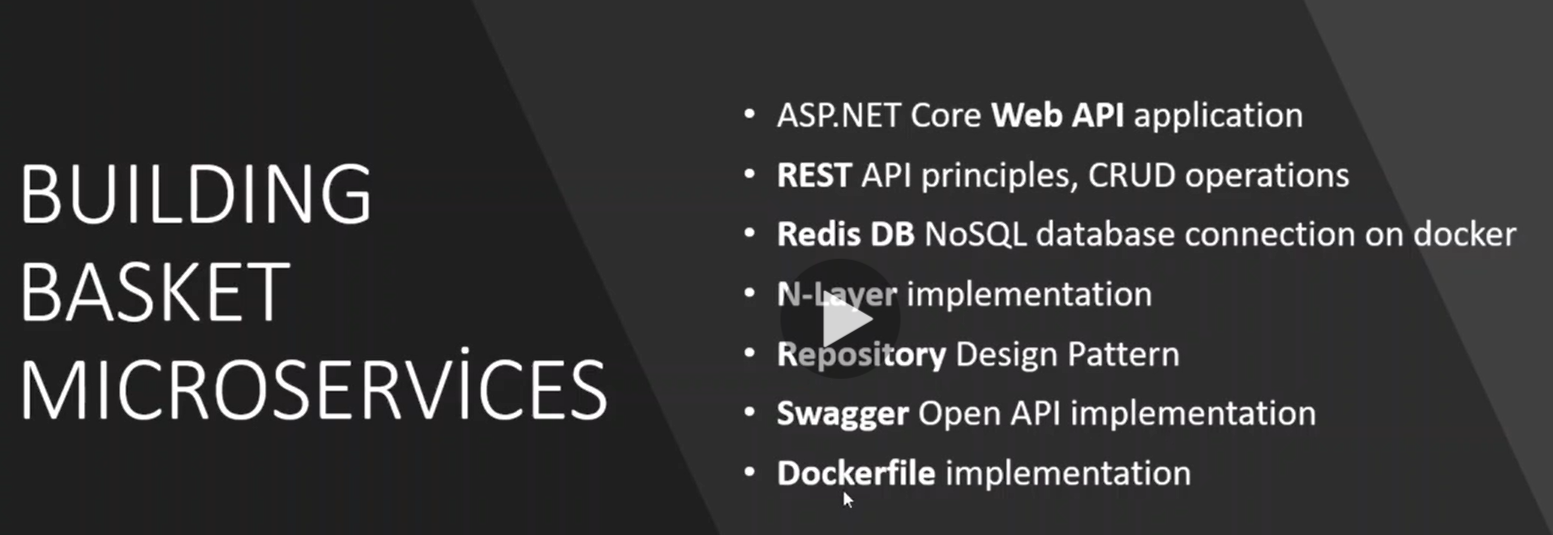
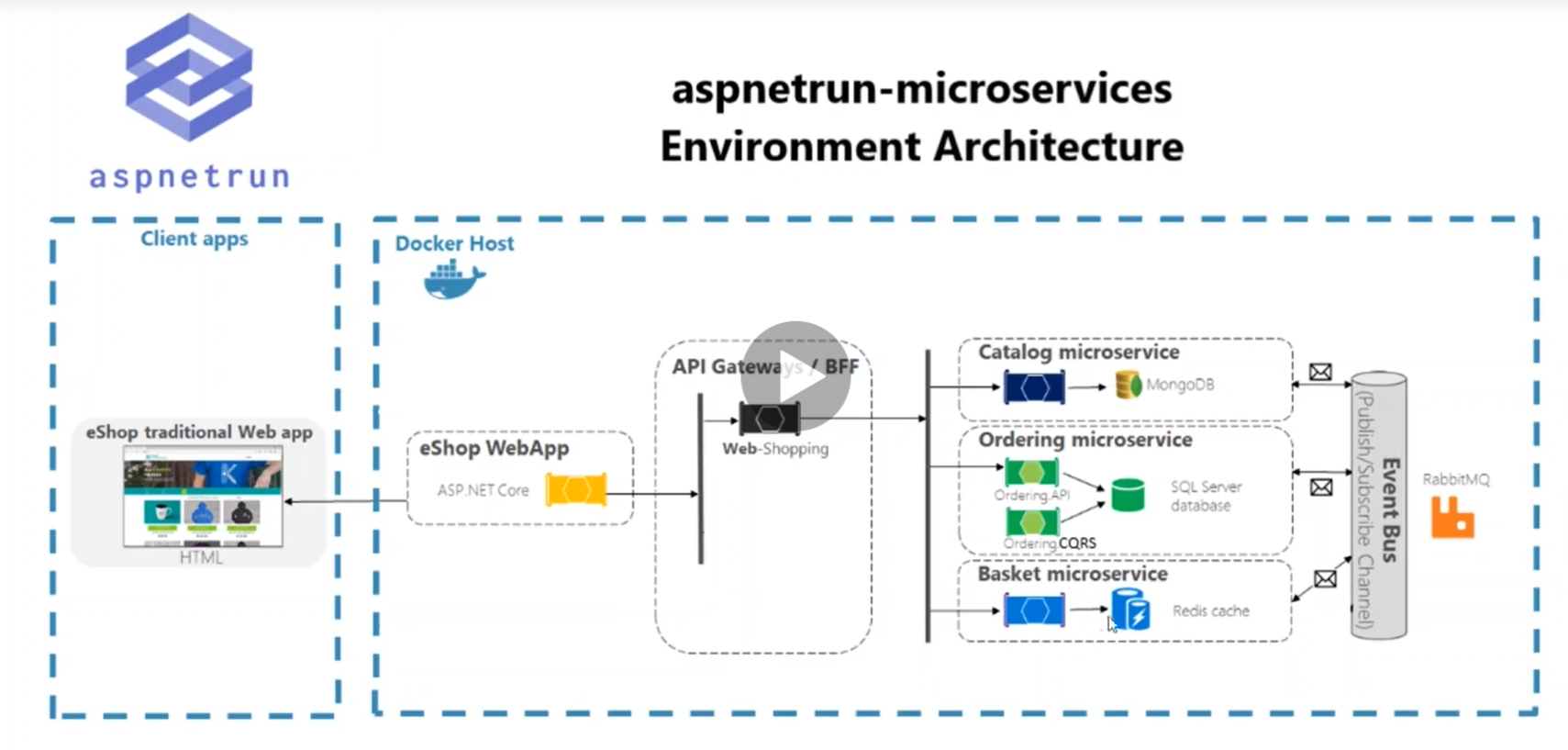
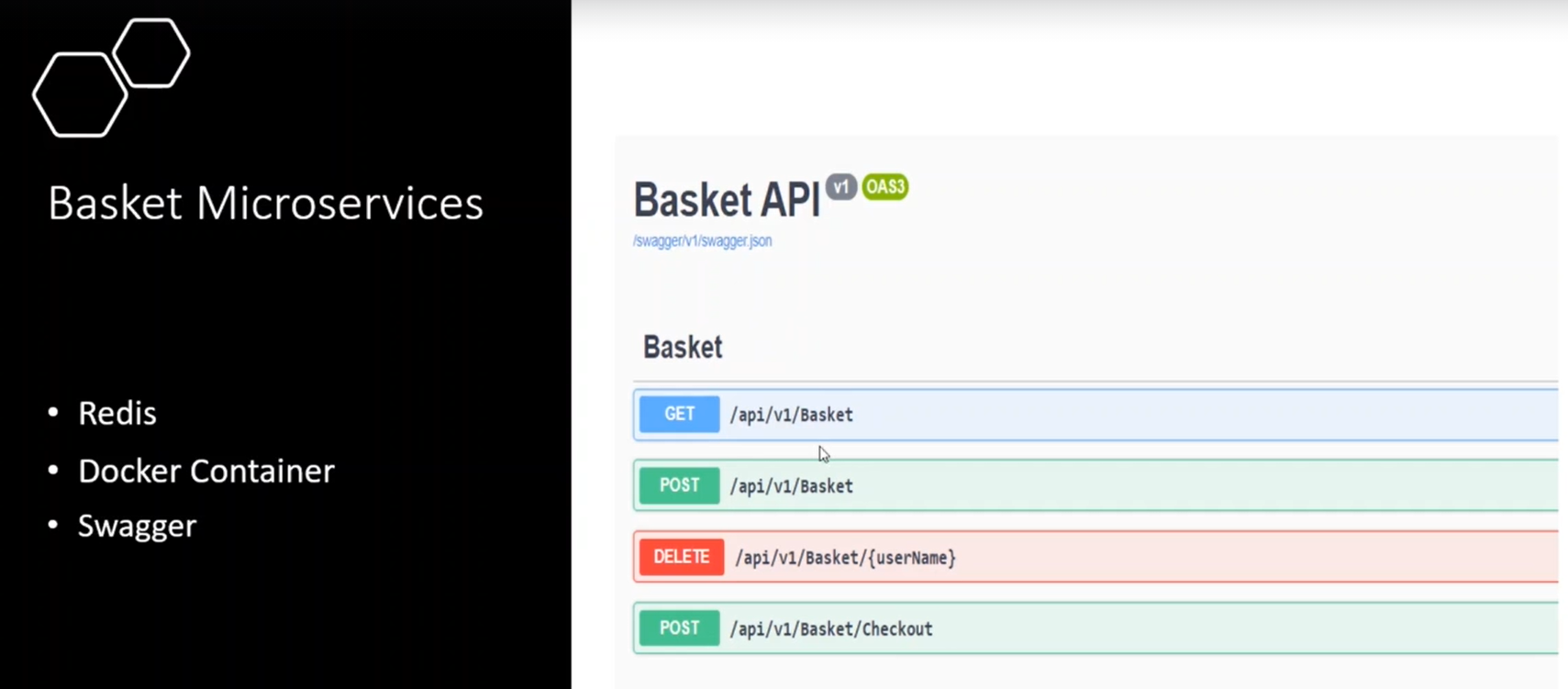
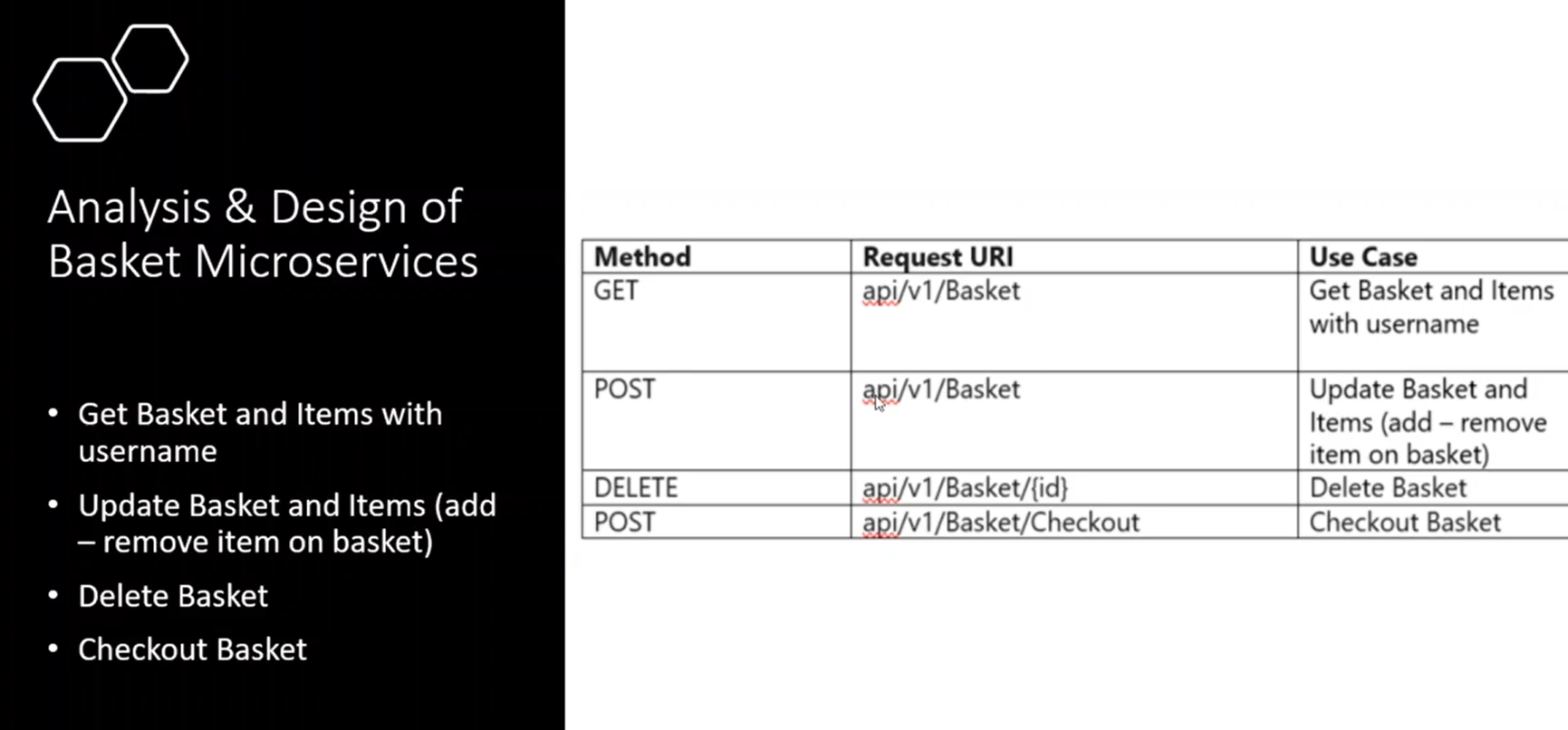
**Section03 building Basket Micro services**

**Notes:-**

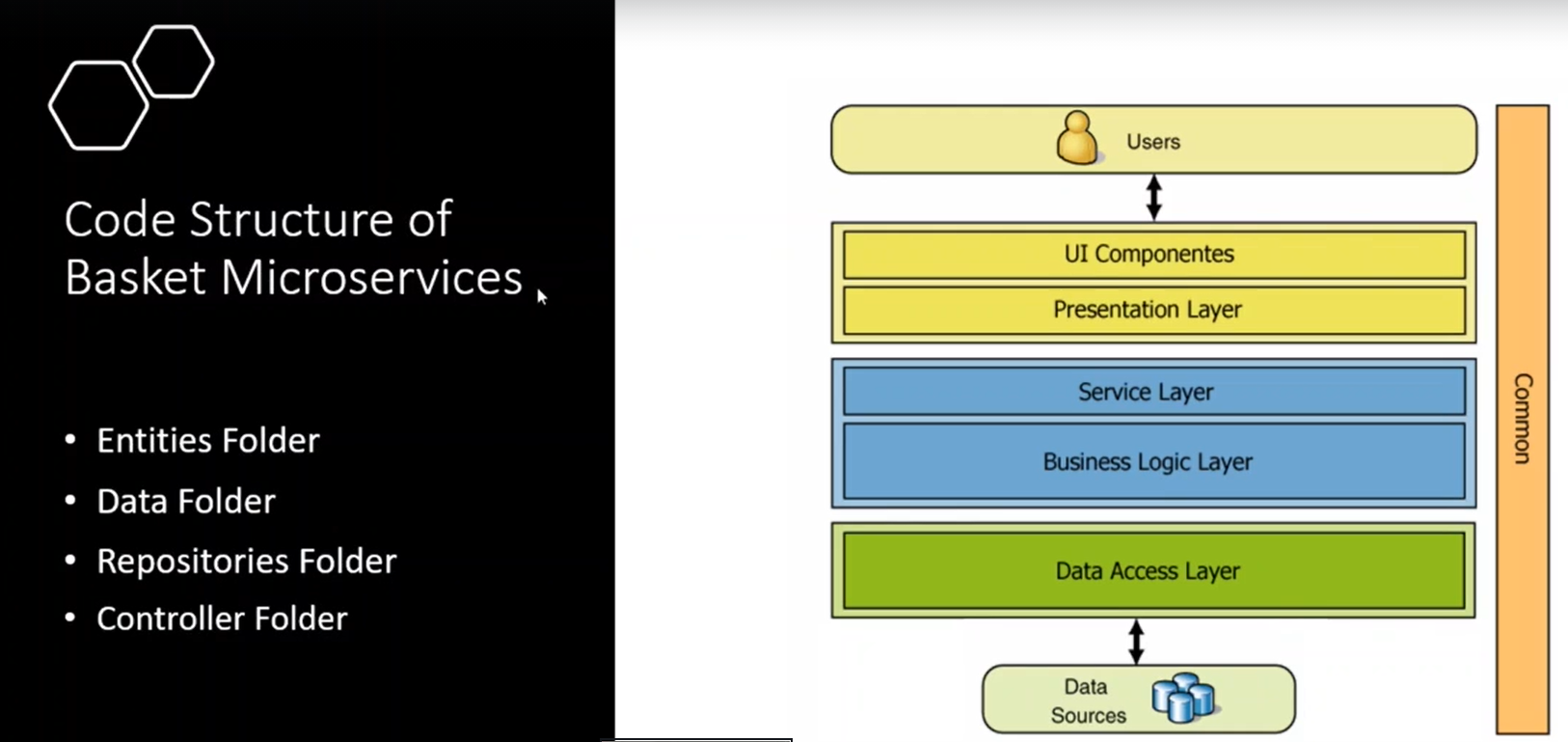




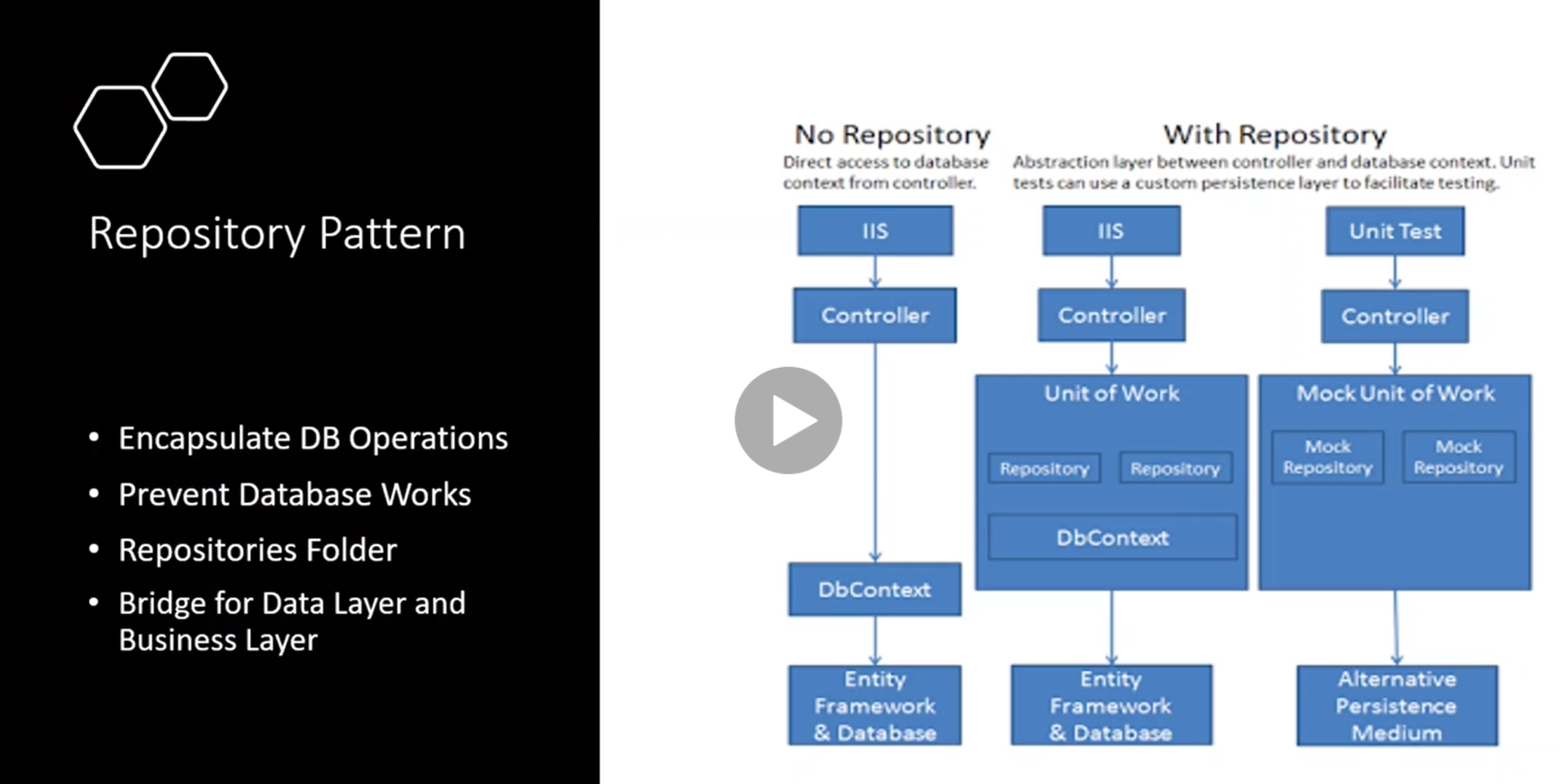




**Code Structure of Basket MicroServices**



**We using N-layer as we use in the Catalog API**



**What is Redis**



**Redis is No-SQL database and have high security ability to store data**

**Redis store data in the server RAM**

**Redis reduce read / write operations because instead to access database , we can access it through RAM**

**Steps of install redis database**

**Notes:-**

**1-go to the docker hub and type on search redis and execute the following commands**

**docker pull redis**

**//create container called aspnetrun-redis from the image redis**

**docker run -d -p 6379:6379 --name aspnetrn-redis redis**

**//to see all logs on the redis database**

**docker logs -f aspnetrun-redis**

**//to access to the redis database**

**docker exec -it aspnetrun-redis /bin/bash**

**redis-cli**

**//to add new key with value**

**set key value**

**//to get key**

**get key**

**//to add new key with value**

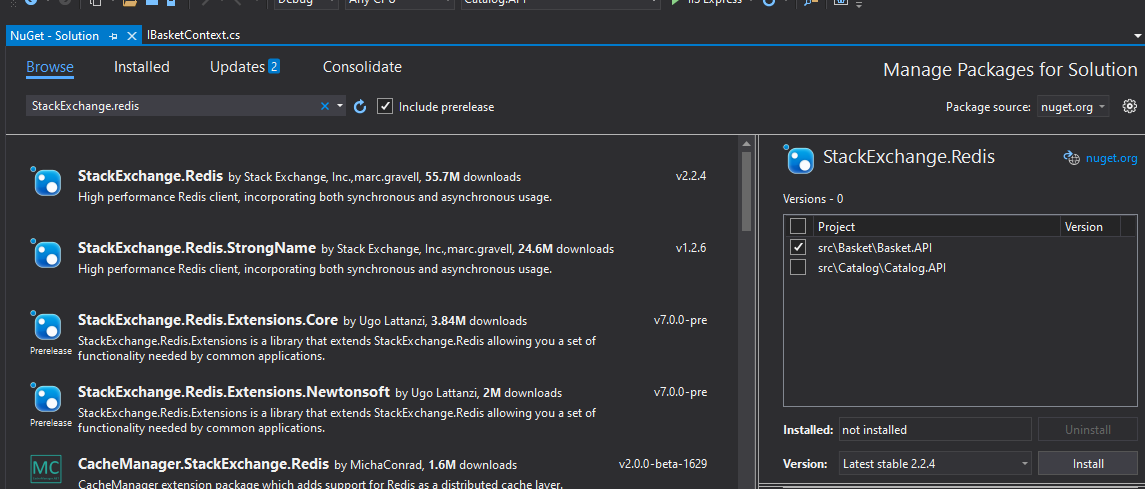
**set fullName MohammedEnbeh**

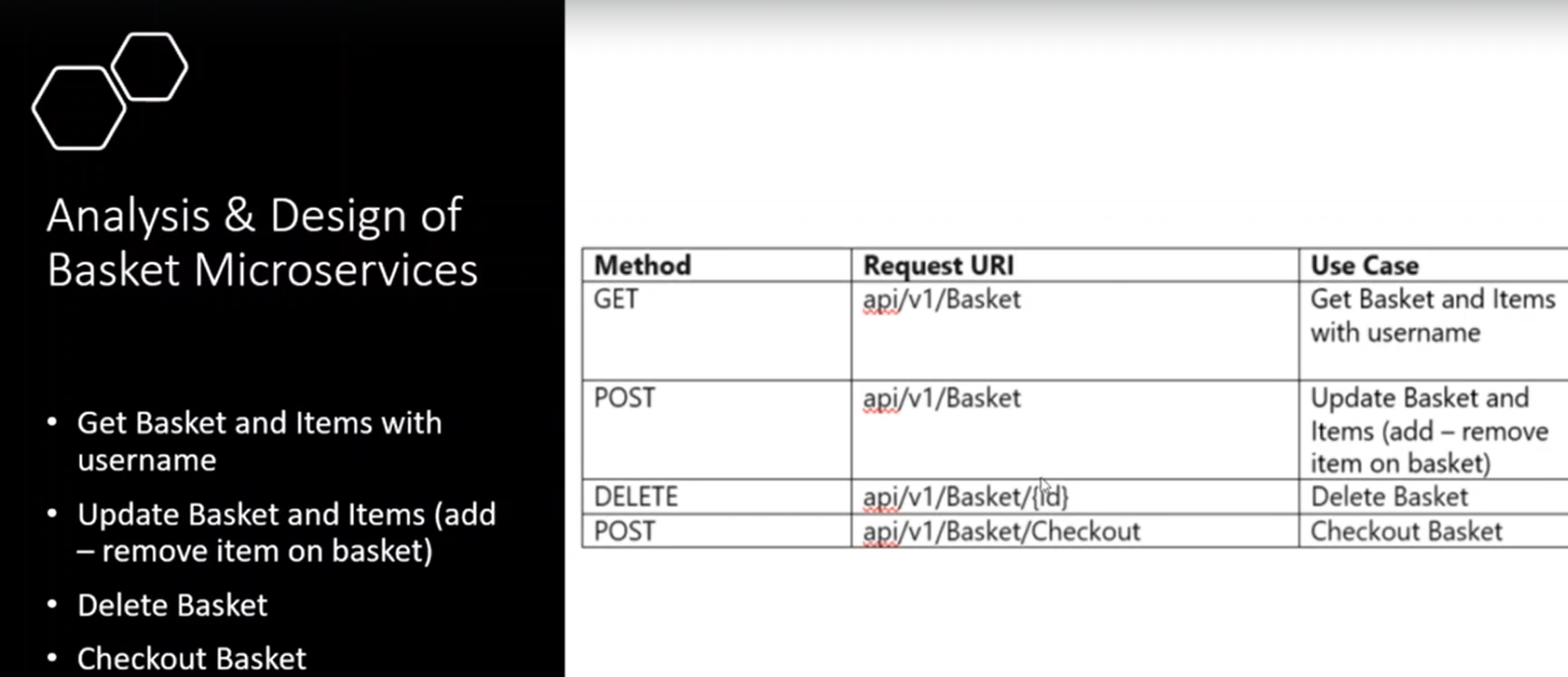
**//to get key**

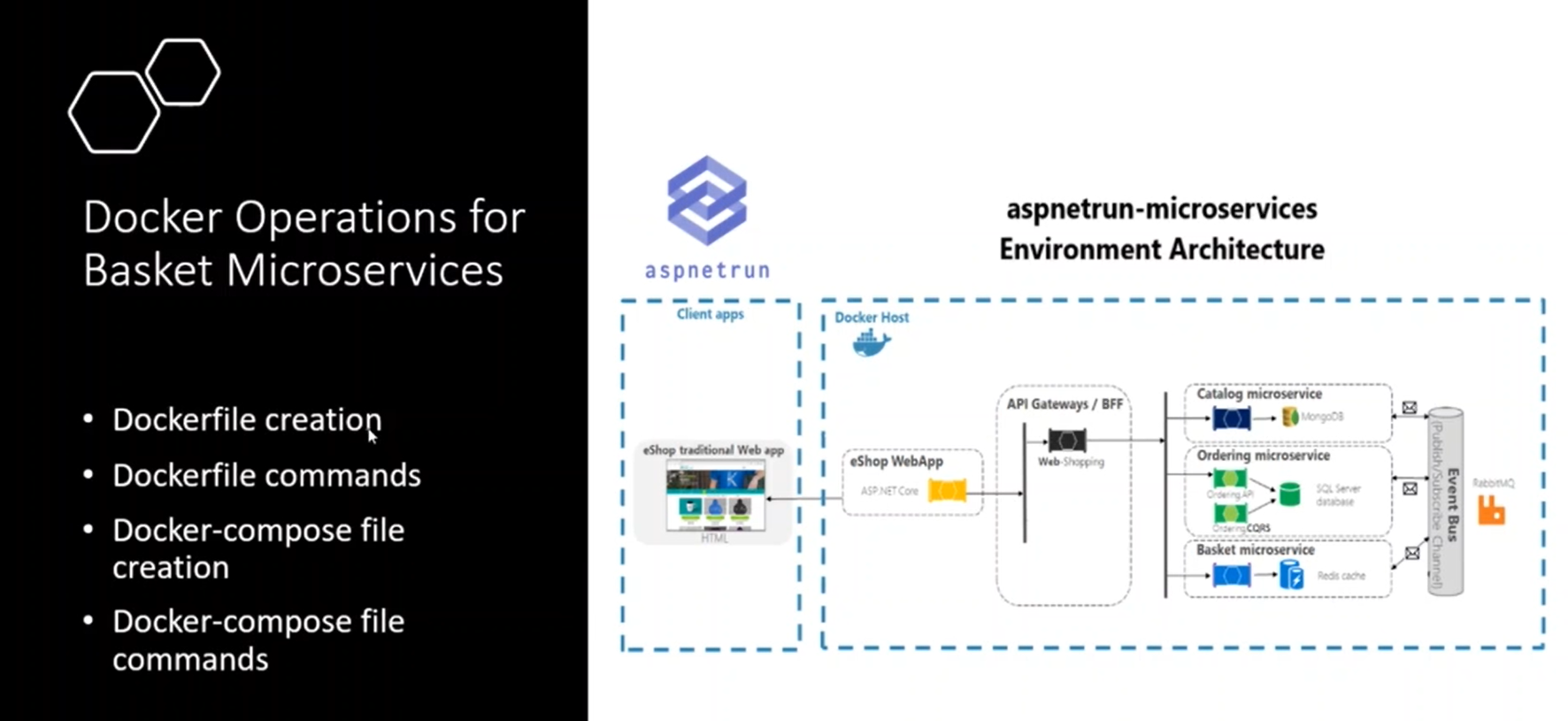
**get fullName**

**2-on VS 2019 install from Nuget the following package**

**StackExchange.Redis**



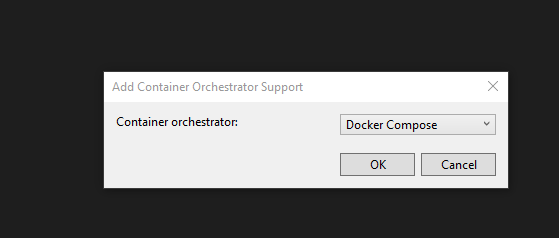




**Docker-Compose File generated in Basket API**

**Notes:-**

**1-on the Basket.API 🡪 right click 🡪 Add 🡪 Container orchestrator support**



**2-on the Docker-compose we set the following code**

version: '3.4'

services:

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

catalogdb:

image: mongo

basketdb:

image: redis

#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

basket.api:

image: ${DOCKER\_REGISTRY-}basketapi

build:

context: .

dockerfile: src/Basket/Basket.API/Dockerfile

**in the Docker-compose.override 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"

basketdb:

container\_name: basketdb

restart: always

ports:

- "6379:6379"

#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"

basket.api:

container\_name: basketapi

environment:

- ASPNETCORE\_ENVIRONMENT=Development

- "ConnectionStrings:Redis=basketdb:6379"

depends\_on:

- basketdb

volumes:

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

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

ports:

- "8001:80"

**Run the following command**

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

**steps:-**

**1-on the appsettings.json we configure the Redis and the service bus configuration**

{

"ConnectionStrings": {

"Redis": "localhost:6379"

},

"EventBus": {

"HostName": "localhost",

"UserName": "guest",

"Password": "guest"

},

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft": "Warning",

"Microsoft.Hosting.Lifetime": "Information"

}

},

"AllowedHosts": "\*"

}

**2-on the startup class we configure Redis Connection and service bus and apply DI and BasketContext and BasketRepository**

**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){**

**//we map the appsetting configuration with the configuration and establish connect with the Redis database**

**services.AddSingleton<ConnectionMultiplexer>(sp =>{**

**var configuration = ConfigurationOptions.Parse(Configuration.GetConnectionString("Redis"),true);**

**return ConnectionMultiplexer.Connect(configuration);});**

**//register the DI of the repository and context classes**

**services.AddTransient<IBasketRepository, BasketRepository>();**

**services.AddTransient<IBasketContext, BasketContext>();**

**//to register the AutoMapper on the service collection**

**services.AddAutoMapper(typeof(Startup));**

**services.AddControllers();**

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

**//we create the interface IRabbitMQConnection that inject into the RabbimtMQConnection with pass**

**//DI of the IConnectionFactory to apply DI for the IConnection Factory**

**services.AddSingleton<IRabbitMQConnection>(sp => {**

**var factory = new ConnectionFactory(){HostName = Configuration["EventBus:HostName"]};**

**if (!string.IsNullOrEmpty(Configuration["EventBus:UserName"])){**

**factory.UserName = Configuration["EventBus:UserName"];}**

**if(!string.IsNullOrEmpty(Configuration["EventBus:Password"])){**

**factory.UserName = Configuration["EventBus:Password"];}**

**return new RabbitMQConnection(factory);});**

**//we register the EventBusRabbitMQProducer**

**services.AddSingleton<EventBusRabbitMQProducer>();}**

**// 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();**

**app.UseSwagger();**

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

**app.UseRouting();**

**app.UseAuthorization();**

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

**3-on the BasketAPI > Repositories > Interfaces**

**public interface IBasketRepository{**

**Task<BasketCart> GetBasket(string userName);**

**Task<BasketCart> UpdateBasket(BasketCart basket);**

**Task<bool> DeleteBasket(string userName);}**

**//create the repository inherit from IBasketRepository**

**public class BasketRepository : IBasketRepository{**

**private readonly IBasketContext \_context;**

**public BasketRepository(){}**

**public BasketRepository(IBasketContext context){**

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

**public async Task<BasketCart> GetBasket(string userName){**

**var basket = await \_context.Redis.StringGetAsync(userName);**

**if (string.IsNullOrEmpty(basket))**

**return null;**

**return JsonConvert.DeserializeObject<BasketCart>(basket);}**

**public async Task<BasketCart> UpdateBasket(BasketCart basket){**

**var updated = await \_context.Redis.StringSetAsync(basket.UserName, JsonConvert.SerializeObject(basket));**

**if (!updated)**

**return null;**

**return await GetBasket(basket.UserName);}**

**public async Task<bool> DeleteBasket(string userName){**

**return await \_context.Redis.KeyDeleteAsync(userName);}}**

**4-on the Mapping > BasketMapping.cs we using to mapping from the BasketCheckout to the BasketCheckoutEvent**

**public class BasketMapping : Profile{**

**public BasketMapping(){CreateMap<BasketCheckout, BasketCheckoutEvent>().ReverseMap();}}**

**5-on the BasketAPI > Entities**

**public class BasketCart{**

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

**public List<BasketCartItem> Items { get; set; } = new List<BasketCartItem>();**

**public BasketCart(){}**

**public BasketCart(string userName){UserName = userName;}**

**public decimal TotalPrice{**

**get{**

**decimal totalPrice = 0;**

**foreach (var item in Items){totalPrice += item.Price \* item.Quantity;}**

**return totalPrice;}}}**

**public class BasketCartItem{**

**public int Quantity { get; set; }**

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

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

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

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

**public class BasketCheckout{**

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

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

**//BilingAddress**

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

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

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

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

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

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

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

**//Payment**

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

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

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

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

**public int PaymentMethod { get; set; }}**

**6-on the Data > Interfaces > create interface IBasketContext**

**public interface IBasketContext{IDatabase Redis { get; }}**

**public class BasketContext : IBasketContext{**

**//using for the communicate with the Redis database**

**private readonly ConnectionMultiplexer \_redisConnection;**

**public BasketContext(ConnectionMultiplexer redisConnection){**

**\_redisConnection = redisConnection ?? throw new ArgumentException(nameof(redisConnection));**

**Redis = redisConnection.GetDatabase();}**

**//represent the redis database**

**public IDatabase Redis { get; }}**

**7-on the Controllers we define BasketController**

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

**[ApiController]**

**public class BasketController : ControllerBase{**

**//apply DI of the IMapper on the BasketController**

**private readonly IBasketRepository \_repository;**

**private readonly IMapper \_mapper;**

**private readonly EventBusRabbitMQProducer \_eventBus;**

**public BasketController(IBasketRepository repository,IMapper mapper, EventBusRabbitMQProducer eventBus){**

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

**\_mapper = mapper ?? throw new ArgumentNullException(nameof(mapper));**

**\_eventBus = eventBus ?? throw new ArgumentNullException(nameof(eventBus));}**

**//when you add submit on product form checkout it will get product then add new item into the product then update basket**

**[HttpGet]**

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

**public async Task<ActionResult> GetBasket(string userName){**

**var basket = await \_repository.GetBasket(userName);**

**return Ok(basket ?? new BasketCart(userName));}**

**[HttpPost]**

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

**public async Task<ActionResult> UpdateBasket([FromBody] BasketCart basket){**

**return Ok(await \_repository.UpdateBasket(basket));}**

**[HttpDelete("{userName}")]**

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

**public async Task<ActionResult> DeleteBasket(string userName){**

**return Ok(await \_repository.DeleteBasket(userName));}**

**//used to send the basket information after checkout the cart page**

**//becouse we have two method of http post we see that we have to set custom route for the checkout**

**[Route("[action]")]**

**[HttpPost]**

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

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

**public async Task<IActionResult> Checkout([FromBody] BasketCheckout basketCheckout){**

**//get total price of basket**

**//remove the basket**

**//send checkout event to rabbitmq**

**var basket = await \_repository.GetBasket(basketCheckout.UserName);**

**if (basket is null)**

**return BadRequest();**

**var basketRemoved = await \_repository.DeleteBasket(basket.UserName);**

**if (!basketRemoved){return BadRequest();}**

**//we will mapp the basketCheckout object to BasketCheckout Event by using Auto Mapper**

**var eventMessage = \_mapper.Map<BasketCheckoutEvent>(basketCheckout);**

**eventMessage.RequestId = Guid.NewGuid();**

**eventMessage.TotalPrice = basket.TotalPrice;**

**try{**

**\_eventBus.PublishBasketCheckout(EventBusConstants.BasketCheckoutQueue,eventMessage);}**

**catch{throw;}**

**return Accepted();}}**