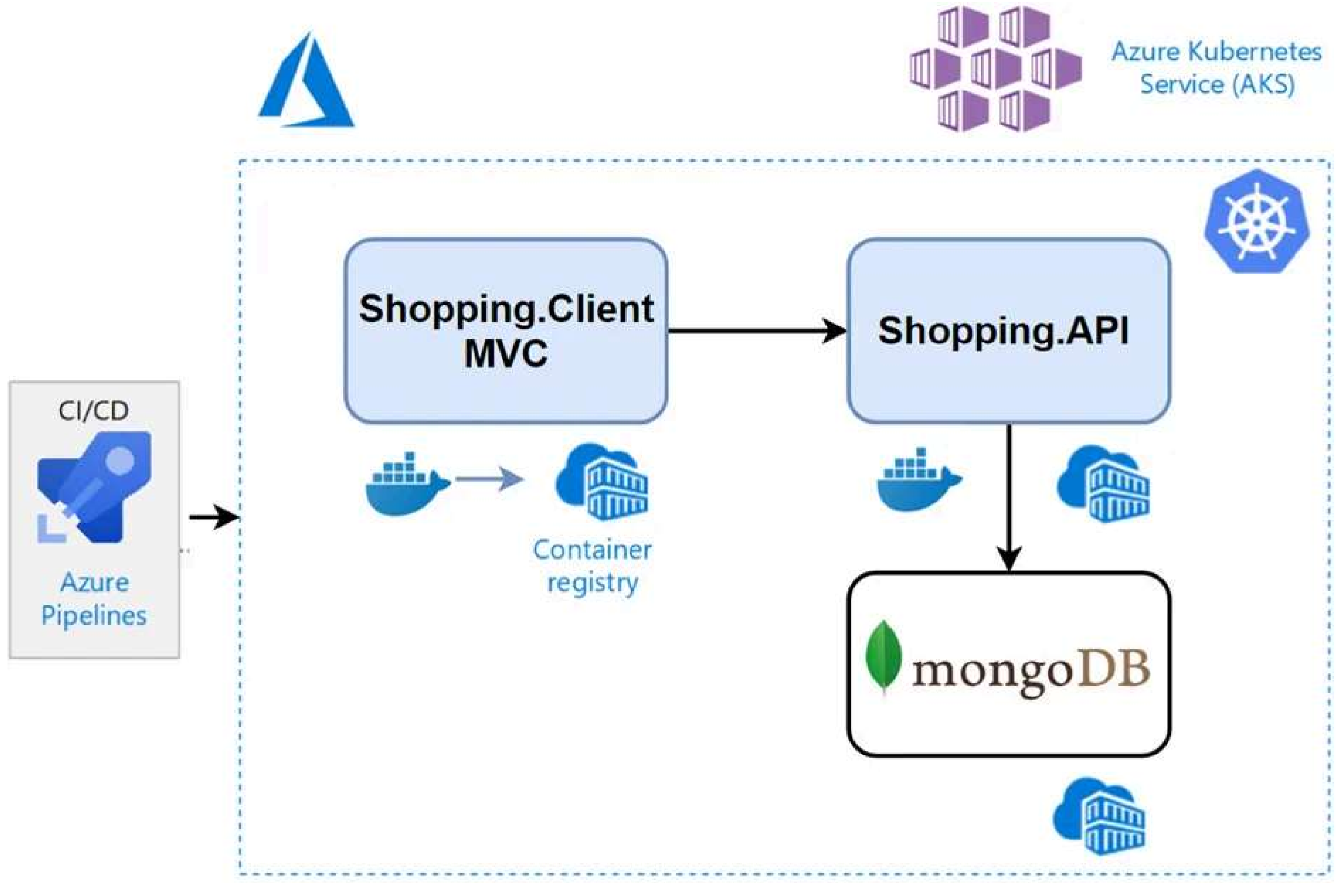
Preparing Multi- Microservices Applications for Deployment

Mehmet Ozkaya · Follow
Published in aspnetrun · 14 min read · Jan 13, 2021

29 3

In this article, we're going to prepare our **multi-container microservices** application. We will develop from scratch and create **DockerFiles**, Build images and orchestrate with **docker-compose** yaml files.



See the overall picture. You can see that we will have 3 microservices which we are going to develop and deploy together.

Shopping MVC Client Application

First of all, we are going to develop Shopping MVC Client Application For Consuming Api Resource which will be the Shopping.Client Asp.Net MVC Web Project. But we will start with developing this project as a standalone Web application which includes own data inside it. And we will add container support with **DockerFile**, push docker images to Docker hub and see the deployment options like “Azure Web App for Container” resources for 1 web application.

Shopping API Application

After that we are going to develop Shopping.API Microservice with MongoDb and Compose All Docker Containers.

This API project will have Products data and performs CRUD operations with exposing api methods for consuming from Shopping Client project.
We will containerize API application with creating dockerfile and push images to Azure Container Registry.

Mongo Db

Our API project will manage product records stored in a no-sql mongodb database as described in the picture.
we will pull **mongodb** docker image from docker hub and create connection with our API project.

At the end of the section, we will have 3 microservices whichs are **Shopping.Client — Shopping.API — MongoDb** microservices.

As you can see that, we have

- Created docker images,
- Compose docker containers and tested them,

Background

This is the first article of the series. You can follow the series with below links.

- **0- Deploying .Net Microservices**
- **1- Preparing Multi-Container Microservices Applications for Deployment**
- **2- Deploying Microservices on Kubernetes**

- 3- Deploy Microservices into Cloud Azure Kubernetes Service (AKS) with using Azure Container Registry (ACR)
- 4- Automate Deployments with CI/CD pipelines on Azure Devops

Step by Step Development w/ Course

IT & Software › Other IT & Software › Kubernetes

Deploying .Net Microservices with K8s, AKS and Azure DevOps

Deploying .Net Microservices to Kubernetes, move cloud Azure Kubernetes Services(AKS), Automating with Azure DevOps

I have just published a new course — Deploying .Net Microservices with K8s, AKS and Azure DevOps.

In this course, we're going to learn how to **Deploying .Net Microservices** into **Kubernetes**, and moving deployments to the cloud **Azure kubernetes services (AKS)** with using **Azure Container Registry(ACR)** and last section is we will learn how to Automating Deployments with **CI/CD pipeline of Azure DevOps** and GitHub.

Source Code

Get the Source Code from AspnetRun Microservices Github — Clone or fork this repository, if you like don't forget the star. If you find or ask anything you can directly open issue on repository.

Developing Your First Microservice

In this section we are going to Building MVC Client Application For Consuming Api Resource which will be the Shopping.Client Asp.Net MVC Web Project. But we will start with developing simple MVC application before integrate with API project.

As you can see the overall picture, we are in here and start to developing Shopping MVC Web Application project.
First, We will develop this project as a standalone Web application which includes own data inside it. And we will add container support with DockerFile, push docker images to Docker hub and see the deployment options like “Azure Web App for Container” resources for 1 web application.

So in this section, we are going to develop Shopping.Client MVC application which will be standalone web application and includes own data.
Next sections we will consume the Shopping.API project.

— First, we are going to create;
Create New Project
New Blank Solution

and Give the Name of the solution is “Shopping”

— Create Asp.Net Core Web Project :
Right Click — Add new Web Project — Asp.Net Core Web Application — Web Application (Model-View-Controller)
— not select HTTPS
— n : Shopping.Client

— Run the application
Set a Startup Project

Change Run Profile to “Shopping.Client” and Run the application
See 5001 http port worked — http://localhost:5001

Adding Model Class into MVC Application

In Solution Explorer, select the project. Create Models folder.
Right-click the Models folder and select Add > Class. Name the class Product and select Add.

Create Product Model into Models folder

```
namespace Shopping.Client.Models
 {
 public class Product
 {
 public string Id { get; set; }
 public string Name { get; set; }
 public string Category { get; set; }
 public string Description { get; set; }
 public string ImageFile { get; set; }
 public decimal Price { get; set; }
 }
 }
```

Developing Shopping.Client Microservices Data Model and Context Objects

Every microservice should have its own database, so we should create data store for Shopping.Client Microservices.

We should create a context class which should store Discount entity objects. So I am going to;

Create Data Folder
Add ProductContext.cs

```
public static class ProductContext
 {
 public static readonly List<Product> Products = new List<Product>
 {
 new Product()
 {
 Name = “IPhone X”,
 Description = “This phone is the company’s biggest change to its
flagship smartphone in years. It includes a borderless.”,
 ImageFile = “product-1.png”,
 Price = 950.00M,
 Category = “Smart Phone”
 },
 new Product()
 {
 Name = “Samsung 10”,
 Description = “This phone is the company’s biggest change to its
flagship smartphone in years. It includes a borderless.”,
 ImageFile = “product-2.png”,
 Price = 840.00M,
 Category = “Smart Phone”
 },
 new Product()
 {
 Name = “Huawei Plus”,
 Description = “This phone is the company’s biggest change to its
flagship smartphone in years. It includes a borderless.”,
 ImageFile = “product-3.png”,
 Price = 650.00M,
 Category = “White Appliances”
 },
 new Product()
 {
 Name = “Xiaomi Mi 9”,
 Description = “This phone is the company’s biggest change to its
flagship smartphone in years. It includes a borderless.”,
 ImageFile = “product-4.png”,
 Price = 470.00M,
 Category = “White Appliances”
 },
 new Product()
 {
 Name = “HTC U11+ Plus”,
 Description = “This phone is the company’s biggest change to its
flagship smartphone in years. It includes a borderless.”,
 ImageFile = “product-5.png”,
 Price = 380.00M,
 Category = “Smart Phone”
 },
 new Product()
 {
 Name = “LG G7 ThinQ New8”,
 Description = “This phone is the company’s biggest change to its
flagship smartphone in years. It includes a borderless.”,
 ImageFile = “product-6.png”,
 Price = 240.00M,
 Category = “Home Kitchen”
 }
 }
 }
```

Listing Products on Index Page of Shopping.Client Microservice

We are going to Listing Products on Index Page of Shopping.Client Microservice.

First, we are going to update home page of MVC Application;

```
HomeController.cs
 public IActionResult Index()
 {
 return View(ProductContext.Products); — ADDED parameter
 }
```

So we have now Products on Index page, we should map this data into html list objects.
After that, let me modify Index.cshtml

Update Index.cshtml

```
@model IEnumerable<Shopping.Client.Models.Product>

@{
 ViewData[“Title”] = “Home Page”;
 }
```

```
<h1>Products</h1>
 <table class="table">
 <thead>
 <tr>
 <th>
 @Html.DisplayNameFor(model => model.Name)
 </th>
 <th>
 @Html.DisplayNameFor(model => model.Category)
 </th>
 <th>
 @Html.DisplayNameFor(model => model.Description)
 </th>
 <th>
 @Html.DisplayNameFor(model => model.ImageFile)
 </th>
 <th>
 @Html.DisplayNameFor(model => model.Price)
 </th>
 <th></th>
 </tr>
 </thead>
 <tbody>
 @foreach (var item in Model)
 {
 <tr>
 <td>
 @Html.DisplayFor(modelItem => item.Name)
 </td>
 <td>
 @Html.DisplayFor(modelItem => item.Category)
 </td>
 <td>
 @Html.DisplayFor(modelItem => item.Description)
 </td>
 <td>
 @Html.DisplayFor(modelItem => item.ImageFile)
 </td>
 <td>
 @Html.DisplayFor(modelItem => item.Price)
 </td>
 </tr>
 }
 </tbody>
 </table>
```

Finally, we can Run the application.

Set a Startup Project

Change Run Profile to "Shopping.Client" and Run the application
See 5001 http port worked — http://localhost:5001

Create Docker Container for Shopping.Client Microservice

We are going to Create Docker Container for our Shopping.Client Microservice. For creation docker container, we need to create Dockerfile for our MVC Application. This is very easy to use vs tooling.

First, we are going to Add Docker Support on our vs solution;

Right click — Add Docker Support — Linux

So this operation, create a new Dockerfile for us.
Let's see the Dockerfile
Examine DockerFile

The Dockerfile file, which we will make the necessary settings for Docker, automatically appears.
The purpose of creating the Dockerfile file; When we ask docker to extract the image of our project, it will search for a file named "Dockerfile" in the project. This will make our application work according to the settings in our file.

See file;

```
FROM mcr.microsoft.com/dotnet/aspnet:5.0-buster-slim AS base
 WORKDIR /app
 EXPOSE 80
 EXPOSE 443

FROM mcr.microsoft.com/dotnet/sdk:5.0-buster-slim AS build
 WORKDIR /src
 COPY ["Shopping.Client/Shopping.Client.csproj", "Shopping.Client/"]
 RUN dotnet restore "Shopping.Client/Shopping.Client.csproj"
 COPY . .
 WORKDIR "/src/Shopping.Client"
 RUN dotnet build "Shopping.Client.csproj" -c Release -o /app/build

FROM build AS publish
 RUN dotnet publish "Shopping.Client.csproj" -c Release -o
/app/publish

FROM base AS final
 WORKDIR /app
 COPY — from=publish /app/publish .
 ENTRYPOINT ["dotnet", "Shopping.Client.dll"]
```

Dockerfile consists of 2 main parts. The first is to build the application and the second is to run the application. If we look at what the commands are in the file;

FROM part where the base image is specified in whichever library the FROM project is used in. We will use dotnet 5 SDK image in this project.

WORKDIR It is the part where we specify the folder under which the docker container will copy the files of our project.

COPY is the command used to copy project files from local file system to image. In our project, we will first copy and restore the csproj file, then copy all these files again and create our application by running the dotnet publish command.

RUN: It is used for commands that need to run while Docker containers are being prepared. First, it is ensured that the build is taken and then it is published.

ENTRYPOINT is the command that we specify the first command and parameters that will run when the container is up. While the container is running, DockerExample.dll will be executed with the dotnet command.

Check Output Logs

Once we added Docker Support, in output its automatilcy build docker images

Check docker images output logs

```
 Starting up container(s)…
 docker build -f
“C:\Users\ezozkme\source\repos\swnzen\swnzen\Shopping.Client\Dockerfi
le” — force-rm -t shoppingclient:dev — target base — label
“com.microsoft.created-by=visual-studio” — label
“com.microsoft.visual-studio.project-name=Shopping.Client”
“C:\Users\ezozkme\source\repos\swnzen\swnzen”
 Sending build context to Docker daemon 4.386MB

Step 1/6 : FROM mcr.microsoft.com/dotnet/aspnet:5.0-buster-slim AS
base
 — -> 5f9a6a778eac
 Step 2/6 : WORKDIR /app
 — -> Running in d7b126c32d92
 Removing intermediate container d7b126c32d92
 — -> d23b0bf721d8
 Step 3/6 : EXPOSE 80
 — -> Running in 39654ab99d8b
 Removing intermediate container 39654ab99d8b
 — -> 4c66963a83fe
 Step 4/6 : EXPOSE 443
 — -> Running in efbdcf6019d2
 Removing intermediate container efbdcf6019d2
 — -> 6588782be6b6
 Step 5/6 : LABEL com.microsoft.created-by=visual-studio
 — -> Running in 8394a8470229
 Removing intermediate container 8394a8470229
 — -> 4a876be8a155
 Step 6/6 : LABEL com.microsoft.visual-studio.project-
name=Shopping.Client
 — -> Running in 2861da6e1c69
```

```
Removing intermediate container 2861da6e1c69
— -> 377350c2c197
```

— As you can see that run “docker build” command. we can also run manually. But vs has great tooling experience for docker support.

Push Docker Hub Container Registry to Shopping.Client Microservice Docker Image

We are going to Push Docker Hub Container Registry to Shopping.Client Microservice Docker Image.

We should login the docker over the command line. Before push the image we should login the system.

- Login Docker
- before push
  docker login
  username
  pass

```
docker login
Login with your Docker ID to push and pull images from Docker Hub. If
you don’t have a Docker ID, head over to https://hub.docker.com to
create one.
Username: mehmetozkaya
Password:
Login Succeeded
```

Tag Docker Image For Docker Hub

Before push the image, the image should have tag, so we should tag the image. It is mandatory step to apply tag for image that will be push to Docker

Hub or any container registry, the image should have tag.

We are going to Tag from "lastest" one not dev one. That means ready for production deployment tag.

Lets check existing image;

```
docker images

REPOSITORY TAG IMAGE ID CREATED SIZE
shoppingclient latest cc2a573482cf 45 minutes ago 210MB
```

If there is no latest tag shoppingclient, you can create with Running VS release mod on Docker run profile. Otherwise you can also make it manually with docker build commands.

So once we see the latest shoppingclient docker image, we should tag the lastest one with dockerhub repo name;

```
docker tag cc2 mehmetozkaya/swnzen
```

As you can see that we give the name exactly same as with docker hub repository profile, otherwise it can't match when push docker hub.

Let check now;

```
docker images
```

```
REPOSITORY TAG IMAGE ID CREATED SIZE
shoppingclient latest cc2a573482cf 46 minutes ago 210MB
mehmetozkaya/swnzen latest cc2a573482cf 46 minutes ago 210MB
```

see its tagged.

Push Docker Hub

docker push mehmetozkaya/swnzen:latest

```
The push refers to repository [docker.io/mehmetozkaya/swnzen]
 b57cdc9e8ec8: Pushed
 d066a90a6a65: Pushed
 024230939f4e: Pushed
 ea4124eb3c7e: Pushed
 8ed87ee178f4: Pushing [=======================================> ]
58.3MB/75.66MB
 0916aa79e133: Pushing [===================================> ]
28.54MB/41.33MB
 87c8a1d8f54f: Pushing [====================> ] 28.68MB/69.23MB
```

It takes some time. When we push the image docker search for the tag name exist then push the image to the docker hub.

See DockerHub

As you can see that we can successfully push our image to docker hub.

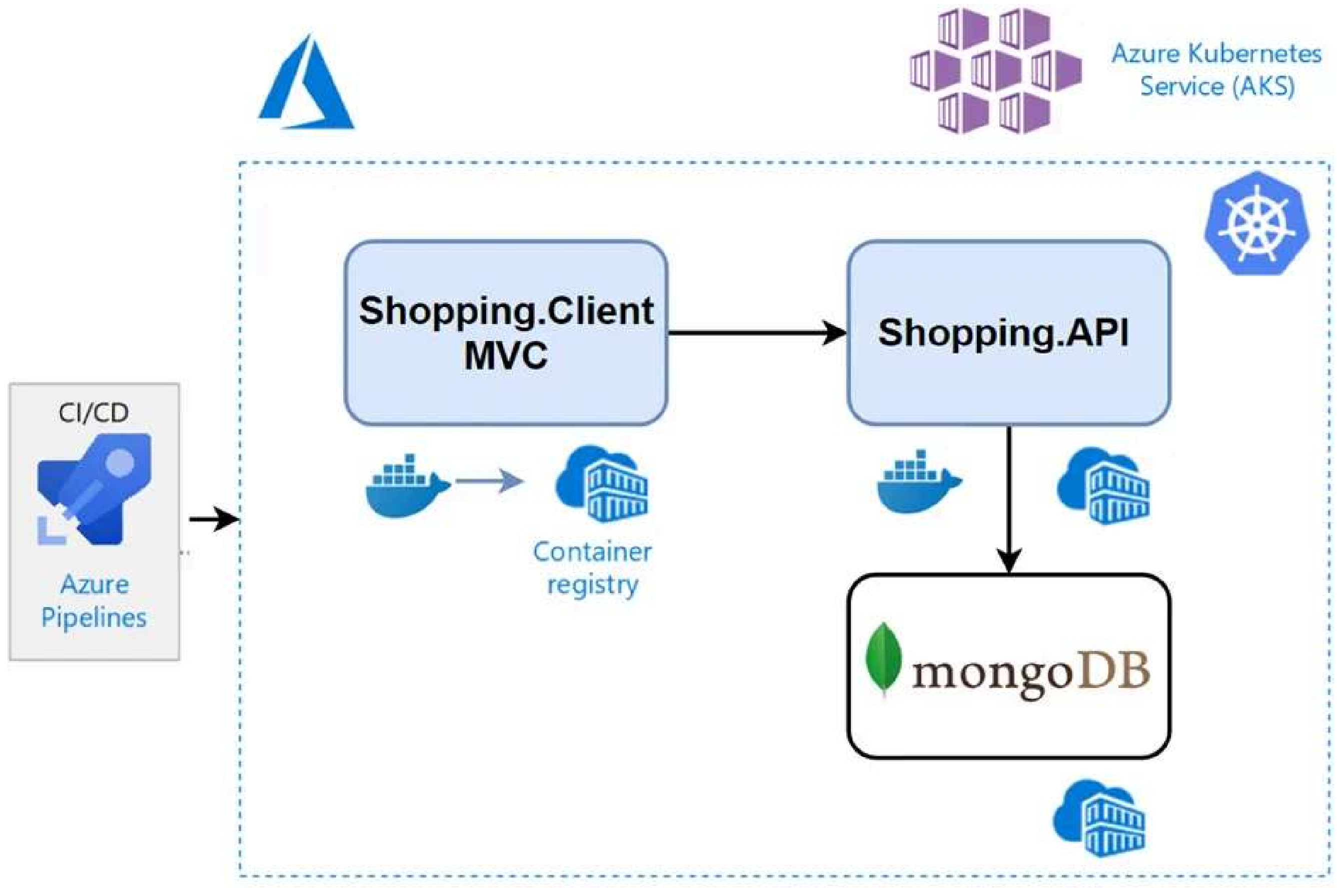
mehmetozkaya/swnzen
Last pushed: 12 minutes ago

As you can see that we have pushed the our shopping.client image to the DockerHub successfully.

Developing Shopping.API Microservice with MongoDb and Compose All Docker Containers

In this section, we are going to develop Shopping.API Microservice with MongoDb and Compose All Docker Containers.
At the end of the section, we will have 3 microservices whichs are Shopping.Client — Shopping.API — MongoDb microservices.

We have developed Shopping.Client and now we are going to start with developing the Shopping.API project.
This API project will have Products data and performs CRUD operations with exposing api methods for consuming from clients.



Our API project will manage product records stored in a no-sql **mongodb** database as described in the picture.

First we will develop API project which using Mongo providers.
After that, we will pull mongodb docker image from docker hub and create connection with our API project.
And of course, we will refactor Shopping.Client MVC application, instead of using hard value Product list, Shopping Client will consume Products from API project.

Create Asp.Net Core Web API Project For Shopping.API Microservice

First, we are going to create;

Create Asp.Net Core Web API Project :
Right Click Solution — Add new Web API — NO HTTPS — name : Shopping.API

Create Product Controller Class for Shopping.API Microservice

We are going to create a new web api project For Shopping.API Microservice.

Go to Controller folder
Add ProductController class

```
[ApiController]
 [Route("[controller]")]
 public class ProductController : ControllerBase
 {
 private readonly ILogger<ProductController> _logger;

public ProductController(ILogger<ProductController> logger)
 {
 _logger = logger;
 }
```

Run application
Change Run Profile to "Shopping.API" and Run the application

swagger and
https://localhost:5000/product

Consume Shopping.API from Shopping.Client Microservices with using HttpClientFactory

We are going to Consume Shopping.API from Shopping.Client Microservices with using HttpClientFactory Object.

First, We should go to Shopping.Client Project.
Locate Startup.cs
We will use http client factory, Add register http client into Aspnet DI;

```
public void ConfigureServices(IServiceCollection services)
 {
 services.AddHttpClient("ShoppingAPIClient", client =>
 {
 client.BaseAddress = new Uri("http://localhost:5000/"); //
Shopping.API url
 });

services.AddControllersWithViews();
 }
```

After that we are going to Inject HttpClient into home controller in order to consume API project when retrieving products data.

```
public class HomeController : Controller
 {
 private readonly HttpClient _httpClient;
 private readonly ILogger<HomeController> _logger;

public HomeController(IHttpClientFactory httpClientFactory,
ILogger<HomeController> logger)
 {
 _logger = logger ?? throw new ArgumentNullException(nameof(logger));
_httpClient = httpClientFactory.CreateClient("ShoppingAPIClient");
 }
```

After that, we are going to Implement Index Get method with calling /products api from shopping.api project.

```
public async Task<IActionResult> Index()
 {
 var response = await _httpClient.GetAsync("/product");
 var content = await response.Content.ReadAsStringAsync();
 var productList =
JsonConvert.DeserializeObject<IEnumerable<Product>>(content); —
Install latest Newtonsoft software

return View(productList);
 }
```

Setup Mongo Docker Database

In this section we are going to Setup Mongo Docker Database. We are in here and finished to development of Shopping MVC and Shopping.API project. Now its time to create a real database which is no-sql mongo db. After that we will create connection from API project.

Setup Mongo Docker Database for Shopping.API Microservices

First, We should go to DockerHub and find mongodb official image.

Setup mongodb docker
Go to docker hub mongo image

Examine documentations, ports and so on.
Pull the mongo docker image in your local docker.

docker pull mongo
Run and Start mongodb

docker run -d -p 27017:27017 — name shopping-mongo mongo

See that we are forwarding the same port — 27017.
For troubleshooting, we have same commands;

docker logs -f shopping-mongo
docker exec -it shopping-mongo /bin/bash

Dockerize Microservices with Creating Multi-Container App using Docker Compose

In this section we are going to Dockerize all Microservices with Creating Multi-Container App using Docker Compose.

Docker Compose is a Docker tool that enables complex applications to be defined and run. With Docker Compose, you can make multiple container definitions in a single file, and run the application by raising all the requirements your application needs with a single command.
We have finished to developments of our microservices on local also test it. So before we deploy these microservices, we are going to create docker images and test on **docker container environment**.

You'll learn how to manage more than one container and communicate between them when using **Container Tools in Visual Studio**.
Managing multiple containers requires container orchestration and requires an orchestrator such as **Docker Compose, Kubernetes, or Service Fabric.**

Here, we'll use Docker Compose. Docker Compose is great for local debugging and testing in the project of the development cycle.

Adding Docker-Compose File for Shopping Microservices Solution

We are going to add Docker-Compose File for Shopping Microservices Solution. First, We should go Shopping.API project. As you know that we have already created DockerFile for Shopping.Client appliction before.
So now we need to add for Shopping.API.
But this time we also need orcestration with Client-API and mongodb, so thats why we need to create docker-compose with dockerfile.

In the Shopping.API project,
choose Add > Container Orchestrator Support.
The Docker Support Options dialog appears.
Choose Docker Compose.
Choose Linux.

DockerFile and docker-compose created.

Visual Studio creates a docker-compose.yml and override file in the docker-compose node in the solution,
and that project shows in boldface font, which shows that it's now the startup project.

See the file;

```
docker-compose.yml

version: ‘3.4’

services:
 shopping.api:
```

```
 image: ${DOCKER_REGISTRY-}shoppingapi
 build:
 context: .
 dockerfile: Shopping.API/Dockerfile

docker-compose.override.yml

version: ‘3.4’

services:
 shopping.api:
 environment:
 — ASPNETCORE_ENVIRONMENT=Development
 — ASPNETCORE_URLS=https://+:443;http://+:80
 ports:
 — “80”
 — “443”
 volumes:
 — ${APPDATA}/ASP.NET/Https:/root/.aspnet/https:ro
```

Docker Compose file defines all the services to be deployed in docker environment. These services rely on either a DockerFile or an existing container image.
In our case we have also Shopping.Client application, we already created DockerFile but also we need to add into docker compose file.
For that purpose, you can add manually in these file like API projects but I would like to generate from Visual studio.

In the Shopping.Client project,
again right-click on the project node,
choose Add > Container Orchestrator Support.
Choose Docker Compose, and then select the Linux.

Visual Studio makes changes to your docker compose YML file. Now both services are included.

As you can see that, we have created docker-compose and add 2 microservices configuration but 1 more left. we will add Mongodb database.

Run multi-container application with Docker Compose

We are going to Run multi-container application with Docker Compose.

We need to Run docker-compose

we have 2 options

1- Close all dockers and run with below command on that location;

docker-compose -f docker-compose.yml -f docker-compose.override.yml up — build

2- Run visual studio docker-compose run — actualy its also run the same command

choose option 2
Set a Startup Project for docker-compose file
Click Run button
See output window

```
1>docker-compose -f
“C:\Users\ezozkme\source\repos\swnzen\swnzen\docker-compose.yml” -f
“C:\Users\ezozkme\source\repos\swnzen\swnzen\docker-
compose.override.yml” -p dockercompose12550600008089957795 — no-ansi
config
```

It Worked !!

Also right click — clean solution — it run docker-compose down command and stop docker ps

```
docker-compose -f
“C:\Users\ezozkme\source\repos\swnzen\swnzen\docker-compose.yml” -f
“C:\Users\ezozkme\source\repos\swnzen\swnzen\docker-
compose.override.yml” -f
“C:\Users\ezozkme\source\repos\swnzen\swnzen\obj\Docker\docker-
compose.vs.debug.partial.g.yml” -p dockercompose12550600008089957795
— no-ansi down — rmi local — remove-orphans
```

See the generated images;

docker ps

```
C:\Users\ezozkme>docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
52b980ee1a0d shoppingclient:dev “tail -f /dev/null” 27 minutes ago Up
27 minutes 443/tcp, 0.0.0.0:8001->80/tcp shoppingclient
c784c0e0a7f5 shoppingapi:dev “tail -f /dev/null” 27 minutes ago Up 27
minutes 443/tcp, 0.0.0.0:8000->80/tcp shoppingapi
c86d72bf1ebb mongo “docker-entrypoint.s…” 27 minutes ago Up 27
minutes 0.0.0.0:27017->27017/tcp shoppingdb
```

Test application

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

http://localhost:8000/product

As you can see that, we have finally dockerize all microservices and running on our local machines. Next article we are going to talk about Kubernetes.

For the next articles ->

- 2- Deploying Microservices on Kubernetes

References

https://docs.docker.com/get-started/overview/
https://docs.docker.com/get-started/
https://medium.com/batech/docker-nedir-docker-kavramlar%C4%B1-avantajlar%C4%B1-901b37742ee0
https://www.mediaclick.com.tr/tr/blog/docker-nedir-docker-ne-ise-yarar
https://www.docker.com/resources/what-containe

Microservices · Docker · Docker Compose · Kubernetes · Azure Container Registry

Published in aspnetrun

Follow

597 Followers · Last published Apr 20, 2021

The best path to leverage your aspnet skills. Onboarding to .Net Software Architect jobs. Developing production-ready enterprise .Net applications with applying latest architectures and best practices.

Written by Mehmet Ozkaya

Follow

8.3K Followers · 325 Following

Software Architect | Udemy Instructor | AWS Community Builder | Cloud-Native and Serverless Event-driven Microservices https://github.com/mehmetozkaya

Responses (3)

What are your thoughts?

Respond

Haseeb Awan
Apr 11, 2022

hello

Reply

Haseeb Awan
Apr 11, 2022

Product

hello

Reply

Ruben Veldman
Oct 5, 2021 (edited)

For everyone that, just like me, want to copy paste the ProductContext.cs, without errors:

using Shopping.Client.Models;

using System.Collections.Generic;

namespace Shopping.Client.Data

{

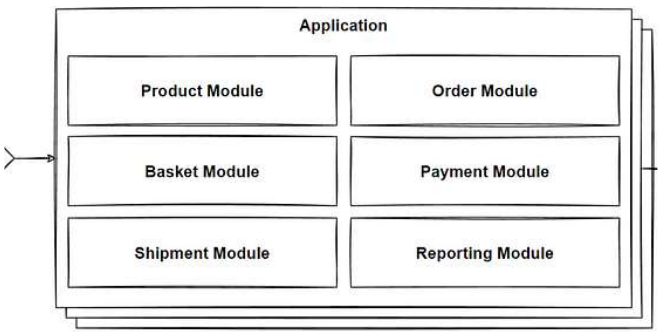
public class ProductContext

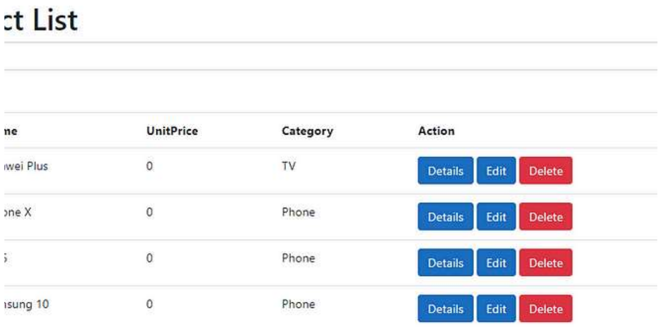
{

public static readonly List<Product>... more

Reply

More from Mehmet Ozkaya and aspnetrun




ct List

ne	UnitPrice	Category	Action
wei Plus	0	TV	Details Edit Delete
one X	0	Phone	Details Edit Delete
5	0	Phone	Details Edit Delete
nsung 10	0	Phone	Details Edit Delete

In Design Microservices Architectur... by Mehme...

Microservices Killer: Modular Monolithic Architecture

In this article, we are going to learn Modular Monolithic Architecture and Best Practices...

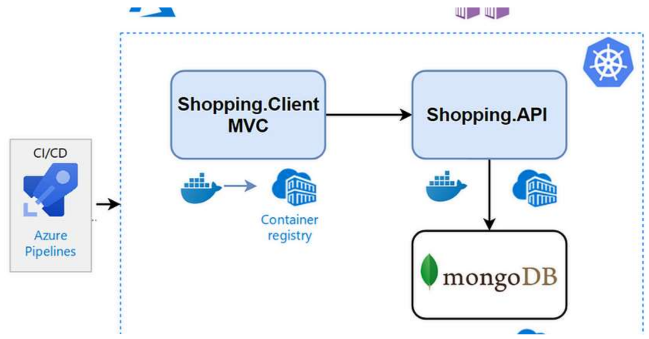
Feb 16, 2023 472 10

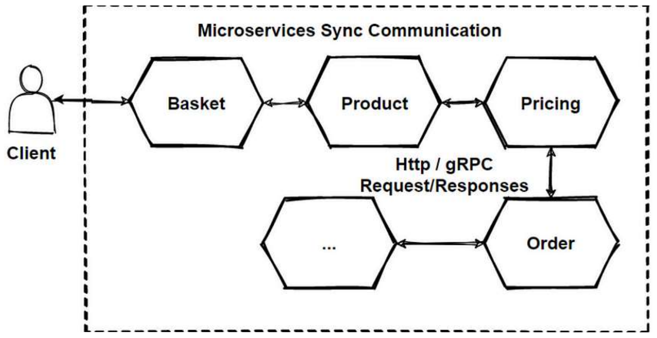
In aspnetrun by Mehmet Ozkaya

Layered Architecture with ASP.NET Core, Entity Framework Core and...

This article explains aspnetrun core repository of github. This series of articles...

Dec 13, 2019 289 6




Microservices Sync Communication
Client
Basket
Product
Pricing
Http / gRPC Request/Responses
...
Order

In aspnetrun by Mehmet Ozkaya

In Design Microservices Architectur... by Mehme...

Deploying Microservices on Kubernetes

In this article, we are going to Deploy our Shopping Microservices on Kubernetes....

Jan 14, 2021 113 3

Microservices Communications

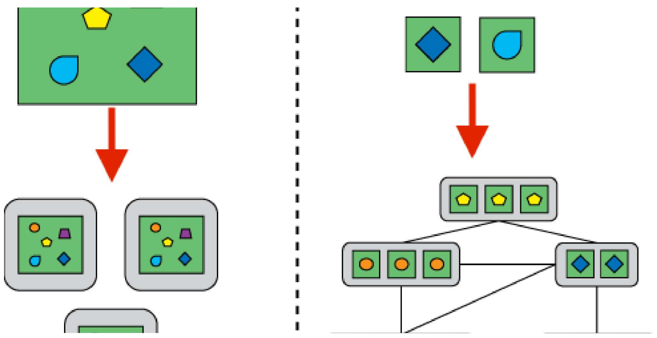
In this article, we're going to learn Microservices Communications. We will lear...

Sep 7, 2021 1.3K 11

See all from Mehmet Ozkaya

See all from aspnetrun

Recommended from Medium

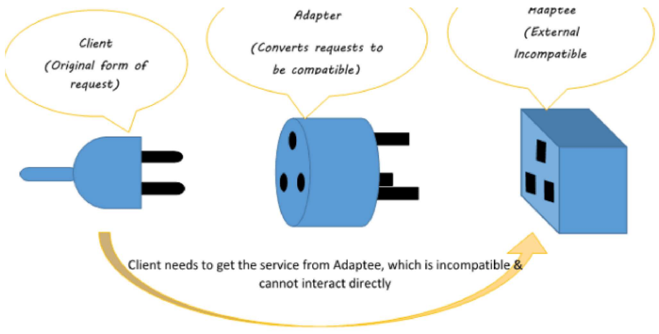


In Artificial Intelligence in Plain Engl... by Amit Sin...

Service-Oriented Architecture (SOA) vs. Microservices

In today's fast-evolving tech landscape, building scalable, maintainable, and flexible...

Sep 26, 2024 184 2


Client (Original form of request)
Adapter (Converts requests to be compatible)
Adaptee (External Incompatible
Client needs to get the service from Adaptee, which is incompatible & cannot interact directly

Ravi Patel

Understanding the Adapter Pattern: A Comprehensive Guide...

Design patterns are critical for solving common software design challenges and...

Sep 23, 2024

Lists

Coding & Development
11 stories · 1018 saves

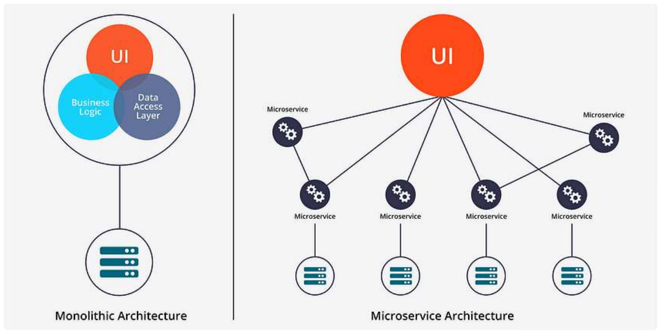
Natural Language Processing
1958 stories · 1602 saves

Ali Zeynalli

10 Must-Know Cloud Native Architecture Patterns

Sidecar/Sidekick, Ambassador, Scatter/Gather, BFF, Anti-Corruption Layer,...

Jan 18 52 1

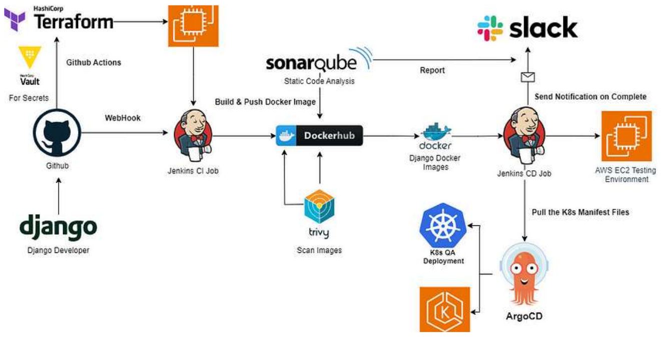


Vinotech

Monolithic and Microservice Architectures in Spring Boot

Monolithic architecture

Sep 30, 2024 51



In Django Unleashed by Joel Wembo

Technical Guide: End-to-End CI/CD DevOps with Jenkins, Docker,...

Building an end-to-end CI/CD pipeline for Django applications using Jenkins, Docker,...

Ramesh Fadatare

5 Microservices Design Patterns You Must Know in 2025

Here are five important microservices design patterns you should know in 2025, explained...

Apr 13, 2024 1.2K 21

Jan 25 223 2

See more recommendations