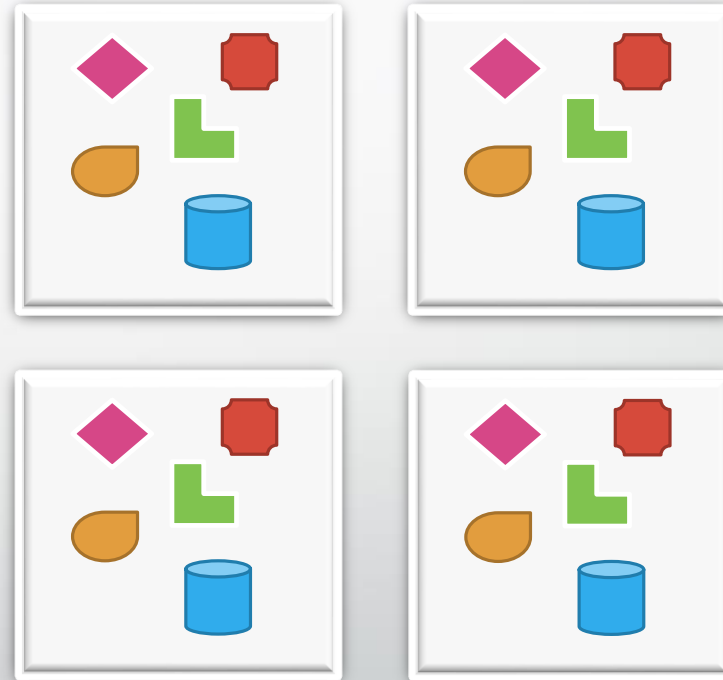
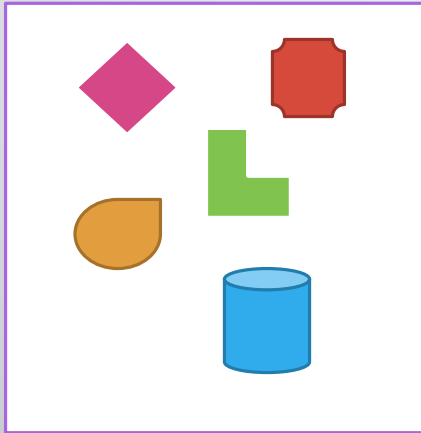


Mikroservisi i ASP.NET Core

SERVISNO-ORIJENTISANE ARHITEKTURE

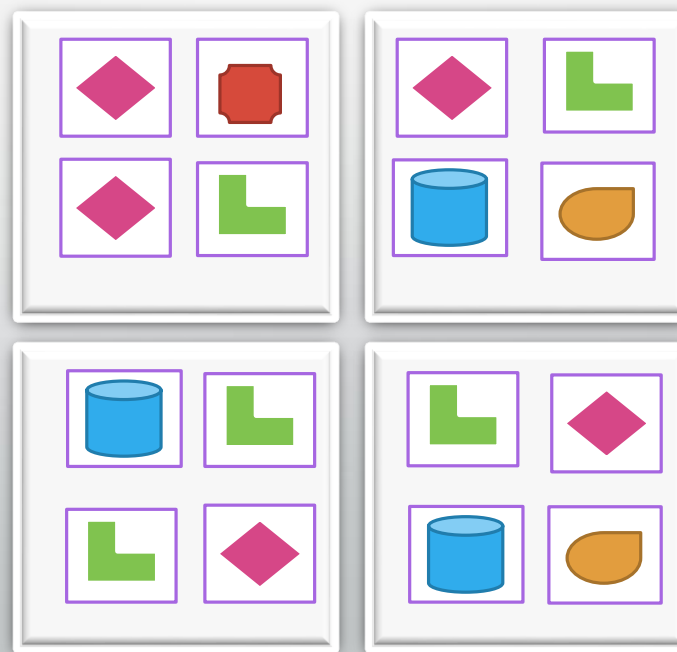
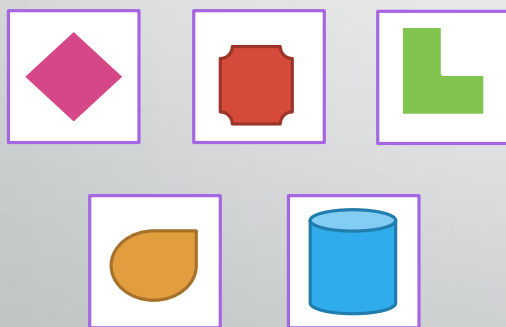
Monolitna arhitektura



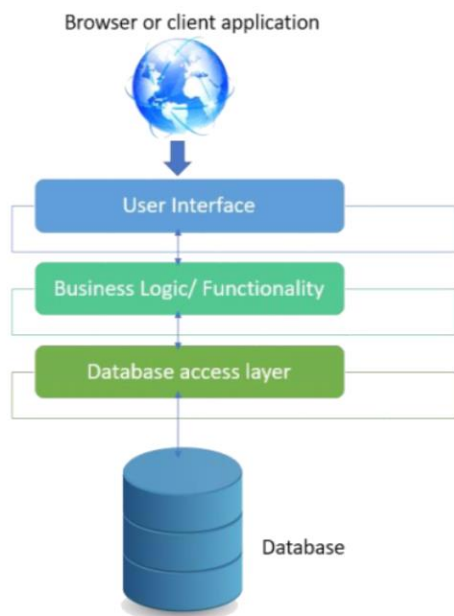
Mikroservisna arhitektura

Mikroservisna arhitektura je princip razvoja aplikacija u okviru malih, izdvojenih servisa, pri čemu svaki servis ima svoj proces i ostvaruje komunikaciju putem jednostavnih mehanizama kao što je HTTP API

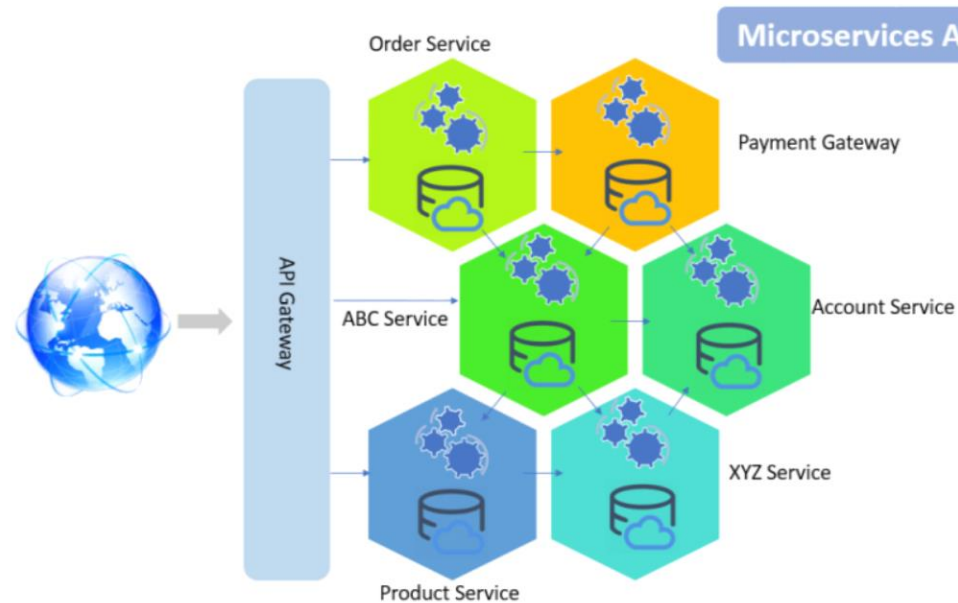
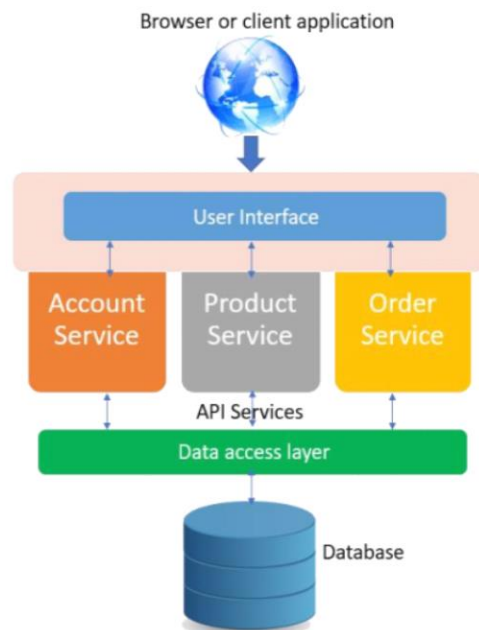
Martin Fowler, ThoughtWorks



Monolitna vs. Mikroservisna arhitektura



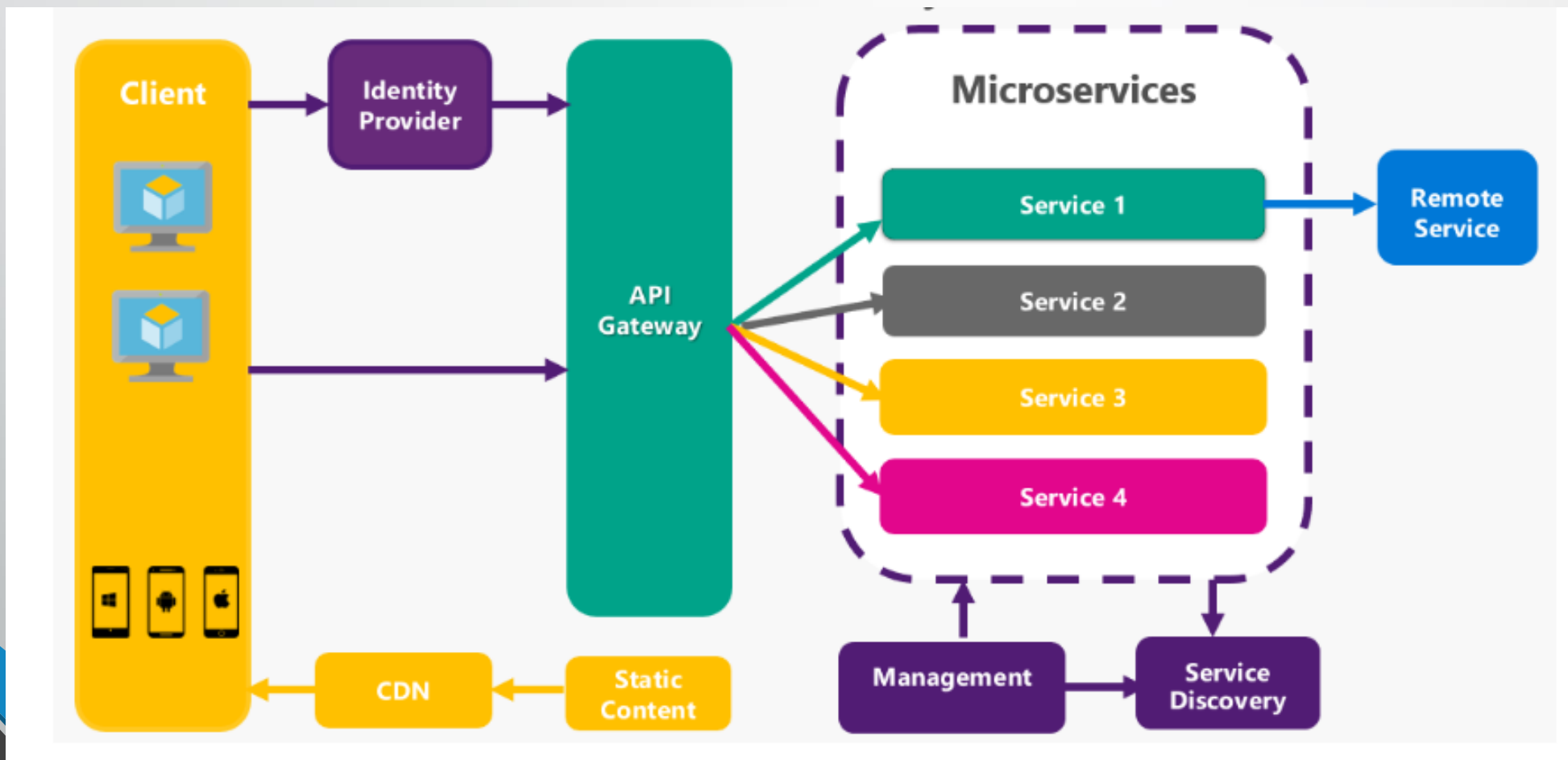
Or



Mikroservisna arhitektura

- Svaki mikroservis implementira specifičnu logiku u domenu tog mikroservisa i svaki mikroservis treba da bude razvijan autonomno i deplojovan nezavisno.
- Kod kreiranja mikroservisa nije ključna njihova veličina, već je cilj kreirati slabo spregnute servise da bismo postigli individualno razvijanje, deplojovanje i skaliranje svakog servisa.

Mikroservisna arhitektura



Zašto koristiti mikroservisnu arhitekturu ?

- Mikroservisna arhitektura omogućava bolje održavanje kompleksnih i velikih sistema, tako što nameće kreiranje aplikacija bazirano na većem broju nezavisnih servisa.
- Dodatna prednost – *skaliranje* – umesto jedne monolitne aplikacije koju treba skalirati kao celinu, korišćenjem mikroservisne arhitekture možemo skalirati mikroservise individualno.

Mikroservisna arhitektura - prednosti

- Različiti razvojni timovi mogu razvijati i isporučivati komponente relativno samostalno
- Komponente mogu biti pisane u različitim programskim jezicima i koristiti različite tehnologije
- Moguće je koristiti različite baze podataka
- Centralizacija upravljanja je minimalna
- Continuous Integration i Continuous Delivery
- Moguće je često izdavanje i ažuriranje verzija komponenti uz održavanje stabilnosti ostatka sistema

Mikroservisna arhitektura - mane

- Složenost međuservisne komunikacije i distribuiranih sistema
- Nepostojanje distribuiranih transakcija
- Složenost operativnih procesa
- Otežani integration testovi
- Potreba za robusnim upravljanjem grešaka i automatskim oporavkom od istih
- Potrebno sofisticirano nadgledanje rada celog sistema

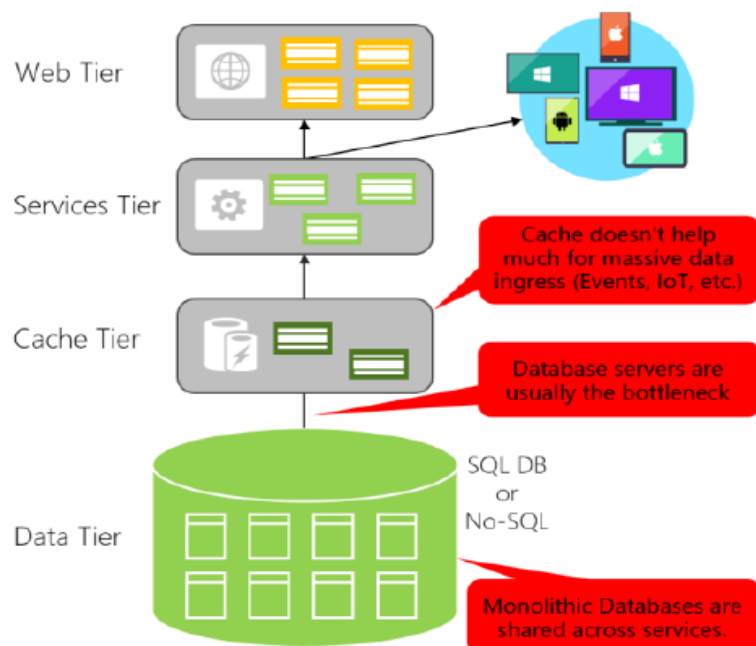
Podaci u mikroservisnoj arhitekturi

- Važno pravilo u mikroservisnoj arhitekturi je da svaki mikroservis mora da sadrži svoje domenske podatke i logiku. Baš kao što kompletna aplikacija sadrži podatke i logiku svaki mikroservis treba da sadrži to isto.
- Ovo znači da će se modeli podataka razlikovati između mikroservisa.
- S druge strane u monolitnoj arhitekturi bismo najčešće koristili jedinstvenu centralizovanu bazu.

Podaci u mikroservisnoj arhitekturi

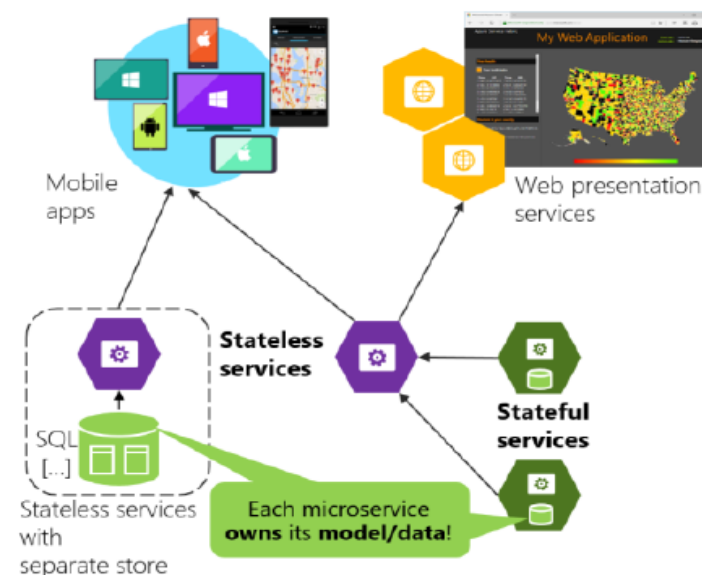
Data in Traditional approach

- Single monolithic database
- Tiers of specific technologies



Data in Microservices approach

- Graph of interconnected microservices
- State typically scoped to the microservice
- Remote Storage for cold data



Podaci u mikroservisnoj arhitekturi

- Inicijalno centralizovana baza i jedna aplikacija možda deluju prostije i omogućavaju višestruko korišćenje entiteta u različitim podsistemima.
- Realnost je malo drugačija...
- Zapravo ćemo završiti sa ogromnim tabelama koje su korišćene od strane mnogo različitih podsistema i koje sadrže attribute i kolone koje nisu neophodne u većem broju slučajeva.
- Prednost monolitne aplikacije sa tipično jednom relacionom bazom – ACID i SQL.

Podaci u mikroservisnoj arhitekturi

- Svaki od mikroservisa treba da ima podatke koji su za njega privatni i kojima se pristupa korišćenjem API-ja tog mikroservisa. Ako dva ili više servisa deli perzistentne podatke (jedna baza) nameće se problem pažljivog koordinisanja promena nad šemom podataka što dovodi do usporavanja razvoja aplikacije.
- Nameće se nekoliko rešenja:
 - 1. Private-tables-per-service
 - 2. Schema-per-service
 - 3. Database-per-service

Implementacija mikroservisa

- Mikroservis može biti implementiran korišćenjem različitih tehnologija u različitim programskim jezicima.
- Kao jedno od rešenja nameće se .NET Core.
 - Kros-platformsko rešenje
 - Open-source framework
 - Velika podrška za razvijanje u vidu kvalitetnih alata
 - Brz razvoj
 - Docker podrška

Docker

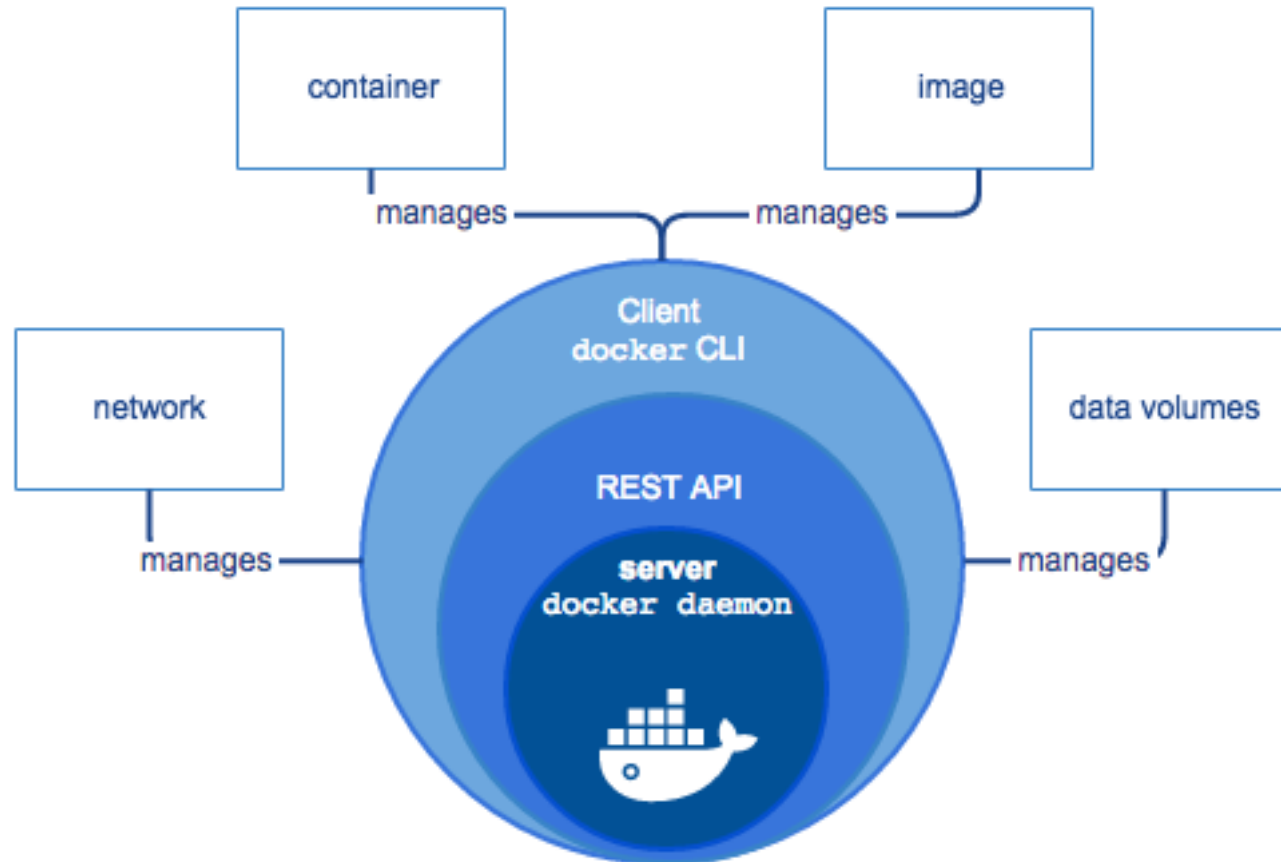


- Docker je tehnologija za kreiranje softverskih kontejnera - paketi pojedinačnih aplikacija koji sadrže sve neophodno za pokretanje i izvršavanje.
 - Način da „zapakujemo“ sve što je aplikaciji neophodno za izvršavanje i pokretanje.
- Kao osnovni koncept prilikom korišćenja Dokera ističe se pojam **image**.
- **Image** podrazumeva sam kod naše aplikacije, ali i sve zavisnosti i pakete koje su aplikaciji neophodne za izvršenje.
- Kreiranjem docker slike na osnovu neke aplikacije omogućavamo njeno pokretanje u okviru docker kontejnera.

Docker Engine

- Docker engine je klijent-server aplikacija sa tri glavne komponente:
 - Server (long-running program) – još se naziva i daemon proces
 - REST API – nudi interfejs koji programi mogu da koriste za rad sa demonom i da mu odrede šta treba da radi.
 - Command line interface (CLI) – docker komanda

Docker Engine



Docker i .NET Core

- .NET Core se može veoma jednostavno izvršavati u Docker kontejneru.
- Kontejneri nam omogućavaju lightweight način da izolujemo aplikaciju od host sistema, deljenjem samo kernela i korišćenjem resursa dodeljenih našoj aplikaciji.

Instalacija dokera

- Uspešnost instalacije možete proveriti izvršenjem komande:
 - `docker --version`
- Ako ova komanda prikaže informacije o verziji, znači da je instalacija prošla uspešno.




Mikroservisi i ASP.NET Core

Primer

Create a new project

Recent project templates

 ASP.NET Core Web Application C#

 Empty Project C++

asp.net core web application



Clear all

All languages

All platforms

All project types



ASP.NET Core Web Application

Project templates for creating ASP.NET Core web apps and web APIs for Windows, Linux and macOS using .NET Core or .NET Framework. Create web apps with Razor Pages, MVC, or Single Page Apps (SPA) using Angular, React, or React + Redux.

C#

Linux

macOS

Windows

Cloud

Service

Web



ASP.NET Core Web Application

New

Project templates for creating ASP.NET Core web apps and web APIs for Windows, Linux and macOS using .NET Core or .NET Framework. Create web apps with Razor Pages, MVC, or Single Page Apps (SPA) using Angular, React, or React + Redux.

F#

Linux

macOS

Windows

Web



ASP.NET Web Application (.NET Framework)

New

Project templates for creating ASP.NET applications. You can create ASP.NET Web Forms, MVC, or Web API applications and add many other features in ASP.NET.

Visual Basic

Windows

Cloud

Web



ASP.NET Web Application (.NET Framework)

New

Project templates for creating ASP.NET applications. You can create ASP.NET

Back

Next

Configure your new project

ASP.NET Core Web Application

C#

Linux

macOS

Windows

Cloud

Service

Web


Project name

ProductMicroservice

Location

F:\nastava\soa\WebApp1

...

Solution name 

ProductMicroservice

☐ Place solution and project in the same directory

Back

Create



Create a new ASP.NET Core web application

.NET Core

ASP.NET Core 5.0

- ASP.NET Core Empty**
An empty project template for creating an ASP.NET Core application. This template does not have any content in it.
- ASP.NET Core Web API**
A project template for creating an ASP.NET Core application with an example Controller for a RESTful HTTP service. This template can also be used for ASP.NET Core MVC Views and Controllers.
- ASP.NET Core Web App**
A project template for creating an ASP.NET Core application with example ASP.NET Razor Pages content.
- ASP.NET Core Web App (Model-View-Controller)**
A project template for creating an ASP.NET Core application with example ASP.NET Core MVC Views and Controllers. This template can also be used for RESTful HTTP services.
- ASP.NET Core with Angular**
A project template for creating an ASP.NET Core application with Angular
- ASP.NET Core with React.js**

[Get additional project templates](#)

Authentication

No Authentication

[Change](#)

Advanced

- ☐ Configure for HTTPS
- ☒ Enable Docker Support
(Requires [Docker Desktop](#))


Linux

- ☒ Enable OpenAPI support

Author: Microsoft
Source: Templates 5.0.0

Back

Create

 Solution 'ProductMicroservice' (1 of 1 project)

▲  **ProductMicroservice**

☁ Connected Services

▲  Dependencies

▶  Analyzers


▶  Frameworks

▶  Packages


▲  Properties

 launchSettings.json

▶  Controllers

▶  appsettings.json

▶  Dockerfile

▶  Program.cs

▶  Startup.cs

▶  WeatherForecast.cs

Kreiranje modela

- Napraviti folder "Model"
- U njemu dodati klase Product i Category

```
namespace ProductMicroservice.Model
{
    0 references
    public class Product
    {
        0 references
        public int Id { get; set; }
        0 references
        public string Name { get; set; }
        0 references
        public string Description { get; set; }
        0 references
        public decimal Price { get; set; }
        0 references
        public int CategoryId { get; set; }
    }
}
```

```
namespace ProductMicroservice.Model
{
    0 references
    public class Category
    {
        0 references
        public int Id { get; set; }
        0 references
        public string Name { get; set; }
        0 references
        public string Description { get; set; }
    }
}
```

EF Core

- Instalirati pakete:
 - Microsoft.EntityFrameworkCore
 - Microsoft.EntityFrameworkCore.Tools
 - Microsoft.EntityFrameworkCore.Sqlite
- Dodati database context u projekat

Database context

- U projektu kreirati folder "DbContext"
- Dodati klasu ProductContext

```
using Microsoft.EntityFrameworkCore;
using ProductMicroservice.Models;

namespace ProductMicroservice.DBContexts
{
    3 references
    public class ProductContext : DbContext
    {
        4 references
        public DbSet<Product> Products { get; set; }
        0 references
        public DbSet<Category> Categories { get; set; }
        0 references
        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            modelBuilder.Entity<Category>().HasData(
                new Category
                {
                    Id = 1,
                    Name = "Electronics",
                    Description = "Electronic Items",
                },
                new Category
                {
                    Id = 2,
                    Name = "Clothes",
                    Description = "Dresses",
                },
                new Category
                {
                    Id = 3,
                    Name = "Grocery",
                    Description = "Grocery Items",
                }
            );
        }
        0 references
        protected override void OnConfiguring(DbContextOptionsBuilder options)
        => options.UseSqlite("Data Source=DBFileName.db");
    }
}
```

SQLite provider

```
namespace ProductMicroservice
{
    2 references
    public class Startup
    {
        0 references
        public Startup(IConfiguration configuration)
        {
            Configuration = configuration;

            using (var db = new ProductContext())
            {
                db.Database.EnsureCreated();
                db.Database.Migrate();
            }
        }

        1 reference
        public IConfiguration Configuration { get; }

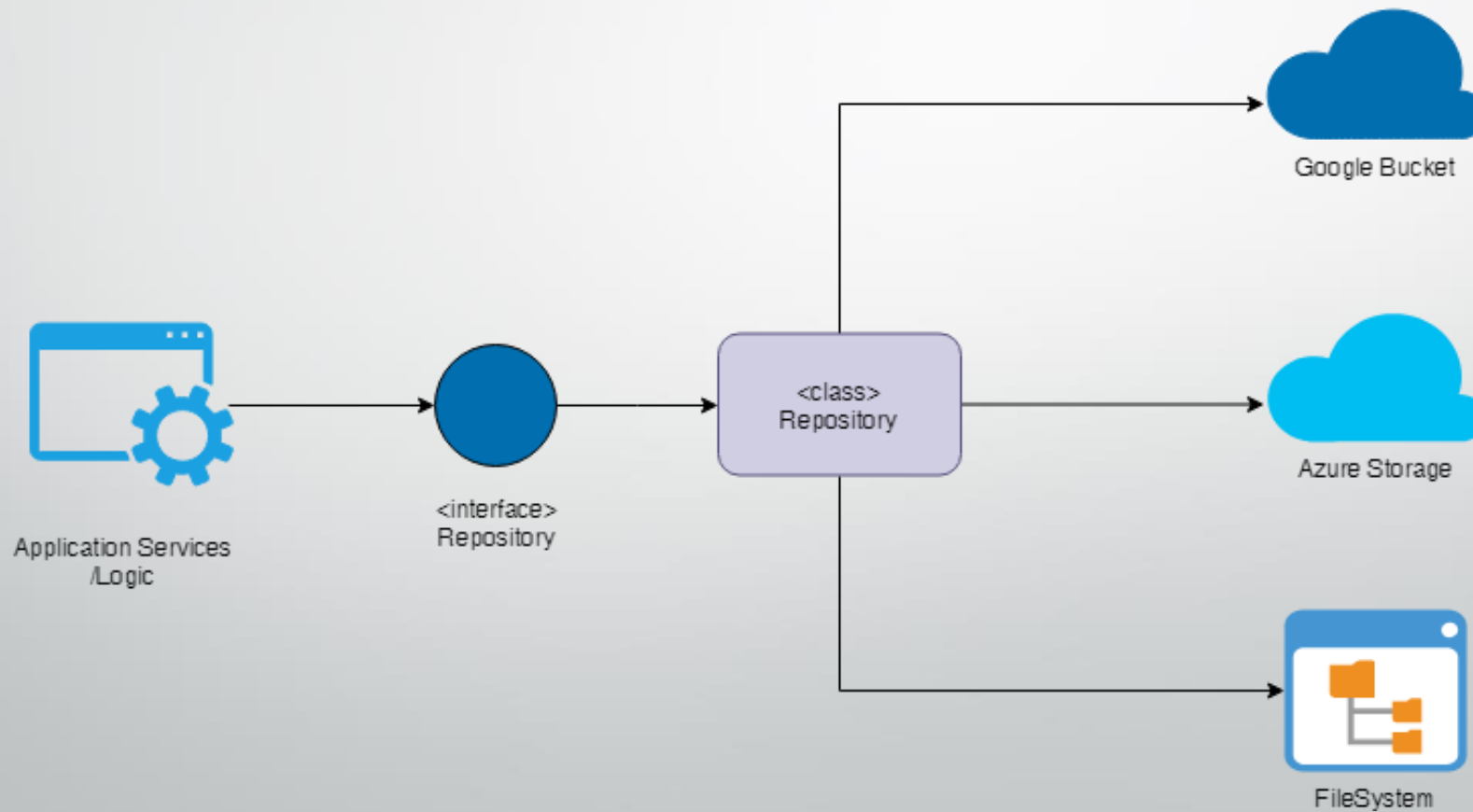
        // This method gets called by the runtime. Use this method to add services to the container.
        0 references
        public void ConfigureServices(IServiceCollection services)
        {
            services.AddControllers();
            services.AddSwaggerGen(c =>
            {
                c.SwaggerDoc("v1", new OpenApiInfo { Title = "ProductMicroservice", Version = "v1" });
            });

            services.AddEntityFrameworkSqlite().AddDbContext<ProductContext>();

        }

        // This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
        0 references
        public void Configure(IApplicationBuilder app, IWebHostEnvironment env) ...
    }
}
```

Repository pattern - podsetnik



Dodavanje repozitorijuma

- Dodati novi folder "Repository"
- Dodati interfejs IProductRepository
 - Dodati metode za CRUD operacije za Product mikroservis

```
using ProductMicroservice.Models;
using System.Collections.Generic;

namespace ProductMicroservice.Repository
{
    4 references
    public interface IProductRepository
    {
        2 references
        IEnumerable<Product> GetProducts();
        2 references
        Product GetProductByID(int ProductId);
        2 references
        void InsertProduct(Product product);
        2 references
        void DeleteProduct(int ProductId);
        2 references
        void UpdateProduct(Product Product);
        4 references
        void Save();
    }
}
```

Dodati konkretnu klasu ProductRepository u folder Repository

```
using Microsoft.EntityFrameworkCore;
using ProductMicroservice.DBContexts;
using ProductMicroservice.Models;
using System.Collections.Generic;
using System.Linq;

namespace ProductMicroservice.Repository
{
    1 reference
    public class ProductRepository : IProductRepository
    {
        private readonly ProductContext _dbContext;
        0 references
        public ProductRepository(ProductContext dbContext)
        {
            _dbContext = dbContext;
        }
        1 reference
        public void DeleteProduct(int productId)
        {
            var product = _dbContext.Products.Find(productId);
            _dbContext.Products.Remove(product);
            Save();
        }
    }
}
```

```
public Product GetProductByID(int productId)
{
    return _dbContext.Products.Find(productId);
}
1 reference
public IEnumerable<Product> GetProducts()
{
    return _dbContext.Products.ToList();
}
1 reference
public void InsertProduct(Product product)
{
    _dbContext.Add(product);
    Save();
}
4 references
public void Save()
{
    _dbContext.SaveChanges();
}
1 reference
public void UpdateProduct(Product product)
{
    _dbContext.Entry(product).State = EntityState.Modified;
    Save();
}
}
```

ConfigureServices

```
using Microsoft.OpenApi.Models;
using ProductMicroservice.DBContexts;
using ProductMicroservice.Repository;

namespace ProductMicroservice
{
    2 references
    public class Startup
    {
        0 references
        public Startup(IConfiguration configuration)
        {
            Configuration = configuration;

            using (var db = new ProductContext())
            {
                db.Database.EnsureCreated();
                db.Database.Migrate();
            }
        }

        1 reference
        public IConfiguration Configuration { get; }

        // This method gets called by the runtime. Use this method to add services to the container.
        0 references
        public void ConfigureServices(IServiceCollection services)
        {
            services.AddControllers();
            services.AddSwaggerGen(c =>
            {
                c.SwaggerDoc("v1", new OpenApiInfo { Title = "ProductMicroservice", Version = "v1" });
            });

            services.AddEntityFrameworkSqlite().AddDbContext<ProductContext>();
            services.AddTransient<IProductRepository, ProductRepository>();
        }

        // This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
        0 references
        public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
    }
}
```


Dodavanje kontrolera

- Desni klik na folder Controllers
 - Kreiranje ProductController klase
 - (API Controller with r/w access)
 - Izmeniti default implementaciju

```

7 namespace ProductMicroservice.Controllers
8 {
9     [Route("api/[controller]")]
10    [ApiController]
11    public class ProductController : ControllerBase
12    {
13        private readonly IProductRepository _productRepository;
14        public ProductController(IProductRepository productRepository)
15        {
16            _productRepository = productRepository;
17        }
18        [HttpGet]
19        public IActionResult Get()
20        {
21            var products = _productRepository.GetProducts();
22            return new OkObjectResult(products);
23        }
24        [HttpGet("{id}", Name = "Get")]
25        public IActionResult Get(int id)
26        {
27            var product = _productRepository.GetProductByID(id);
28            return new OkObjectResult(product);
29        }

```

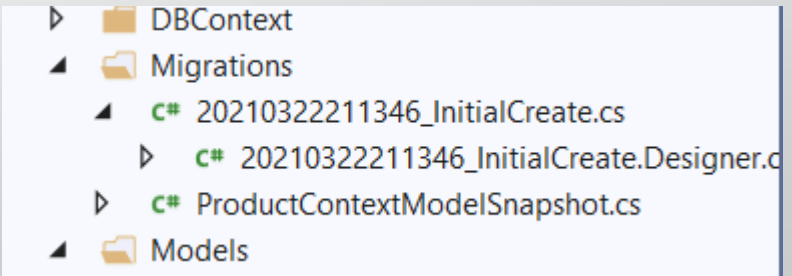
```

30    [HttpPost]
31    public IActionResult Post([FromBody] Product product)
32    {
33        _productRepository.InsertProduct(product);
34        return CreatedAtAction(nameof(Get),
35            new { id = product.Id }, product);
36    }
37    [HttpPut]
38    public IActionResult Put([FromBody] Product product)
39    {
40        if (product != null)
41        {
42            _productRepository.UpdateProduct(product);
43            return new OkResult();
44        }
45        return new NoContentResult();
46    }
47    [HttpDelete("{id}")]
48    public IActionResult Delete(int id)
49    {
50        _productRepository.DeleteProduct(id);
51        return new OkResult();
52    }
53 }
54

```

Entity Framework Core Migrations


- Otvoriti Package Manager konzolu (Tools/NuGet Package Manager/...)
- Ukucati komandu Add-Migration <ime> (npr InitialCreate)
- Komandom **update-database** primenjujemo migracije



A screenshot of a file explorer window showing the project structure. The 'Migrations' folder is expanded, showing two files: '20210322211346_InitialCreate.cs' and '20210322211346_InitialCreate.Designer.cs'. The 'DBContext' folder is also visible, containing 'ProductContextModelSnapshot.cs'. The 'Models' folder is also visible.

```
└─ DBContext
   └─ Migrations
      ├── 20210322211346_InitialCreate.cs
      └─ 20210322211346_InitialCreate.Designer.cs
   └─ ProductContextModelSnapshot.cs
   └─ Models
```

Pokretanje Products mikroservisa

 **Swagger**
Supported by SMARTBEAR

Select a definition ProductMicroservice v1

ProductMicroservice ^{v1} ^{OAS3}

</swagger/v1/swagger.json>

Product

GET

/api/Product

POST

/api/Product

PUT

/api/Product

GET

/api/Product/{id}

DELETE

/api/Product/{id}


Schemas

Product >

Pokretanje pomoću Docker-a

- Desni klik na sln / Add / Container Orchestrator Support / Docker Compose
- Sačuvati sln
- Otvoriti cmd u admin modu i navigirati u folder gde su projektni fajlovi
- **docker images**
- Pokrenuti aplikaciju preko Dockera
- **docker ps**
- Iz prikazanih pokrnutih kontejnera, pročitati port

Pokretanje Products mikroservisa

 **Swagger**
Supported by SMARTBEAR

Select a definition ProductMicroservice v1

ProductMicroservice ^{v1} ^{OAS3}

</swagger/v1/swagger.json>

Product

GET

/api/Product

POST

/api/Product

PUT

/api/Product

GET

/api/Product/{id}

DELETE

/api/Product/{id}

Schemas

Product >

Korisni linkovi

- <https://dotnet.microsoft.com/learn/aspnet/microservices-architecture>
- <https://github.com/dotnet-architecture/eShopOnContainers>
- <https://github.com/teodorislava/SOA-.NET-Core>
- <https://cs.elfak.ni.ac.rs/nastava/mod/resource/view.php?id=8247>
- <https://cs.elfak.ni.ac.rs/nastava/mod/resource/view.php?id=7072>