Customer API – Full-Stack .NET 8.0 Web API with Azure Deployment Walkthrough

Step 1: Install Prerequisite

SDK: https://dotnet.microsoft.com/en-us/download

Visual Studio Code: https://code.visualstudio.com/

Azure CLI: https://learn.microsoft.com/en-us/cli/azure/install-azure-cli

In Terminal: **brew install azure-cli**

Azure Functions Core Tools: **npm install -g azure-functions-core-tools@4 --unsafe-perm true**

Git: https://git-scm.com/downloads

Step 2: Create ASP.NET Core Web API

2. 1 New API Project

```
dotnet new webapi -n CustomerApi
cd CustomerApi
code .
```

2.2 Set Up Program.cs

- Configure Swagger for API documentation
- Add a sample WeatherForecast endpoint for testing

```
using Microsoft.OpenApi.Models;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container
```

```
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen(options =>
{
options.SwaggerDoc("v1", new OpenApiInfo { Title = "Customer API", Version = "v1" });
});

var app = builder.Build();

// Enable Swagger UI
app.UseSwagger();
app.UseSwaggerUI(c =>
{
c.SwaggerEndpoint("/swagger/v1/swagger.json", "Customer API V1");
c.RoutePrefix = string.Empty; // Serves Swagger UI at application root
(http://localhost:5000/)
});

// Define a sample endpoint
var summaries = new[]
{
"Freezing", "Bracing", "Chilly", "Cool", "Mild", "Warm", "Balmy", "Hot", "Sweltering",
"Scorching"
};

app.MapGet("/weatherforecast", () =>
{
var forecast = Enumerable.Range(1, 5).Select(index =>
new
{
Date = DateTime.Now.AddDays(index).ToShortDateString(),
TemperatureC = Random.Shared.Next(-20, 55),
Summary = summaries[Random.Shared.Next(summaries.Length)]
})
.ToArray();
return forecast;
})
.WithName("GetWeatherForecast");

app.Run();
```

2.3 Run Locally

dotnet run

Visit https://localhost:5001/swagger

Step 3: Define Data Models

3.1 Install Packages

```
dotnet add package Microsoft.EntityFrameworkCore

dotnet add package Microsoft.EntityFrameworkCore.SqlServer

dotnet add package Microsoft.EntityFrameworkCore.Tools

dotnet add package Microsoft.VisualStudio.Web.CodeGeneration.Design

dotnet add package Swashbuckle.AspNetCore
```

3.2 Create Customer Model in Models/Customer.cs

```
namespace CustomerApi.Models
{
public class Customer
{
public int Id { get; set; }
public required string Name { get; set; }
public required string Email { get; set; }
public required string Phone { get; set; }
}
}
```

3.3 Update Program.cs

- Add in-memory customer list
- Implement CRUD endpoints for Customers
- Enable Swagger UI only in development environment

```
using CustomerApi.Models;
using Microsoft.OpenApi.Models;

var builder = WebApplication.CreateBuilder(args);

// Enable Swagger (OpenAPI)
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen(options =>
{
options.SwaggerDoc("v1", new OpenApiInfo { Title = "Customer API", Version = "v1" });
});

var app = builder.Build();

// Enable Swagger UI in development
if (app.Environment.IsDevelopment())
{
app.UseSwagger();
app.UseSwaggerUI();
}

// Fake in-memory list of customers
var customers = new List<Customer>
{
new Customer { Id = 1, Name = "Jeet Patel", Email = "jeet@example.com" },
new Customer { Id = 2, Name = "Alex Lee", Email = "alex@example.com" }
};

// Routes

// GET all customers
app.MapGet("/customers", () => customers);

// GET customer by id
app.MapGet("/customers/{id}", (int id) =>
{
var customer = customers.FirstOrDefault(c => c.Id == id);
return customer is not null ? Results.Ok(customer) : Results.NotFound();
});
```

```
// POST new customer
app.MapPost("/customers", (Customer newCustomer) =>
{
newCustomer.Id = customers.Max(c => c.Id) + 1;
customers.Add(newCustomer);
return Results.Created($"/customers/{newCustomer.Id}", newCustomer);
});

// PUT update customer
app.MapPut("/customers/{id}", (int id, Customer updatedCustomer) =>
{
var existing = customers.FirstOrDefault(c => c.Id == id);
if (existing is null) return Results.NotFound();

existing.Name = updatedCustomer.Name;
existing.Email = updatedCustomer.Email;
return Results.Ok(existing);
});

// DELETE customer
app.MapDelete("/customers/{id}", (int id) =>
{
var customer = customers.FirstOrDefault(c => c.Id == id);
if (customer is null) return Results.NotFound();

customers.Remove(customer);
return Results.NoContent();
});

app.Run();
```

3.4 Define EF Core DbContext in Data/AppDbContext.cs

```
using Microsoft.EntityFrameworkCore;
using CustomerApi.Models;
```

```
namespace CustomerApi.Data {
public class AppDbContext : DbContext {
public AppDbContext(DbContextOptions<AppDbContext> options) : base(options) { }
public DbSet<Customer> Customers { get; set; }
}
}
```

Step 4: Set Up Azure SQL & Connection

4.1 Login in to Azure CLI

az login

4.2 Create a Resource Group

az group create --name myResourceGroup --location centralus

4.3 Create SQL Server & Database

```
az sql server create \
  --name myuniquesqlserver1234567 \
  --resource-group myResourceGroup \
  --location centralus \
  --admin-user azureuser \
  --admin-password MyP@ssword123!
```

4.4 Add local IP to the firewall rules

```
az sql server firewall-rule create \
  --resource-group myResourceGroup \
  --server myuniquesqlserver1234567 \
  --name AllowMyIP \
  --start-ip-address $(curl -4 -s ifconfig.me) \
  --end-ip-address $(curl -4 -s ifconfig.me)
```

4.5 Update appsettings.json with connection string from Azure Portal

```
{
"Logging": {
        "LogLevel": {
                "Default": "Information",
                "Microsoft.AspNetCore": "Warning"
                }
        },
        "AllowedHosts": "*",
        "ConnectionStrings": {
                "DefaultConnection":
        "Server=tcp:myuniquesqlserver1234567.database.windows.net,1433;Initial
        Catalog=mySampleDB;Persist Security Info=False;User
        ID=azureuser;Password=MyP@ssword123!;MultipleActiveResultSets=False;Encryp
        t=True;TrustServerCertificate=False;Connection Timeout=30;"
        }
}
```

Step 5: Register DB Context and Scaffold API

5.1 Install EF Core Packages and Tools

```
dotnet add package Microsoft.EntityFrameworkCore.SqlServer

dotnet add package Microsoft.EntityFrameworkCore.Tools

dotnet tool install --global dotnet-ef

dotnet add package Microsoft.EntityFrameworkCore.Design
```

5.2 Create and Apply Migrations

```
dotnet ef migrations add InitialCreate

dotnet ef database update
```

5.3 Update Program.cs

- Register AppDbContext with SQL Server connection
- Replace in-memory list with EF Core DbContext for CRUD operations

```
using CustomerApi.Models;
using CustomerApi.Data; // Your EF Core DbContext namespace
using Microsoft.EntityFrameworkCore;
using Microsoft.OpenApi.Models;

var builder = WebApplication.CreateBuilder(args);

// Add DbContext with SQL Server connection
builder.Services.AddDbContext<AppDbContext>(options =>
options.UseSqlServer(builder.Configuration.GetConnectionString("DefaultConnection")));

// Add Swagger services
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen(options =>
{
options.SwaggerDoc("v1", new OpenApiInfo { Title = "Customer API", Version = "v1" });
});

var app = builder.Build();

// Enable Swagger UI at root URL
app.UseSwagger();
app.UseSwaggerUI(c =>
{
c.SwaggerEndpoint("/swagger/v1/swagger.json", "Customer API V1");
c.RoutePrefix = string.Empty;
});

// CRUD endpoints for Customers using EF Core

app.MapGet("/customers", async (AppDbContext db) =>
await db.Customers.ToListAsync());

app.MapGet("/customers/{id}", async (int id, AppDbContext db) =>
{
var customer = await db.Customers.FindAsync(id);
return customer is not null ? Results.Ok(customer) : Results.NotFound();
});

app.MapPost("/customers", async (Customer newCustomer, AppDbContext db) =>
{
db.Customers.Add(newCustomer);
await db.SaveChangesAsync();
return Results.Created($"/customers/{newCustomer.Id}", newCustomer);
});

app.MapPut("/customers/{id}", async (int id, Customer updatedCustomer, AppDbContext
db) =>
{
var existing = await db.Customers.FindAsync(id);
if (existing is null) return Results.NotFound();

existing.Name = updatedCustomer.Name;
existing.Email = updatedCustomer.Email;
existing.Phone = updatedCustomer.Phone;
await db.SaveChangesAsync();

return Results.Ok(existing);
});

app.MapDelete("/customers/{id}", async (int id, AppDbContext db) =>
{
var customer = await db.Customers.FindAsync(id);
if (customer is null) return Results.NotFound();

db.Customers.Remove(customer);
await db.SaveChangesAsync();

return Results.NoContent();
});
```

```
app.Run();
```

5.4 App Service Plan

az provider register --namespace Microsoft.Web

```
az appservice plan create \
  --name customerApiPlanLinux \
  --resource-group myResourceGroup \
  --sku FREE \
  --is-linux
```

5.5 Web App (Azure App Service)

```
az webapp create \
  --name customer-api-jeet123 \
  --resource-group myResourceGroup \
  --plan customerApiPlanLinux \
  --runtime "DOTNETCORE:8.0"
```

Step 6: Initialize and Push Code

6.1 Configure local Git deployment on Azure Web App

az webapp deployment source config-local-git --name customer-api-jeet123 --resource-group myResourceGroup "url": "https://None@customer-api-jeet123.scm.azurewebsites.net/customer-api-jeet123.git"

6.2 Create GitHub Respitory

Name: Customer-Api

6.3 Initialize Git Locally

cd ~/CustomerApi

git init

git add .

git commit -m "Initial commit"

6.4 Add Git Remote

git remote add origin https://github.com/Jeet52/CustomerApi.git

6.5 Create Personal Access Token

a) Go to Github Setting

b) Click Generate New Token

c) Name: CustomerApi Deployment Token

d) Generate Token

e) Copy Token

6.6 Push Code to Git

git branch -M main

git push -u origin main

Username: Jeet52

Password: *token*

Step 7: Provision Azure App Service

7.1 Register Microsoft Web provider

```
az provider register --namespace Microsoft.Web
```

7.2 Create an App Service Plan

```
az webapp create \
  --name customer-api-app-jeet123 \
  --resource-group myResourceGroup \
  --plan myLinuxPlan \
  --runtime "DOTNETCORE:8.0"
```

7.3 Create the Web App

```
az webapp create \
  --name customer-api-app-jeet123 \
  --resource-group myResourceGroup \
  --plan myLinuxPlan \
  --runtime "DOTNETCORE:8.0"
```

7.4 Vertify the App is Created

```
az webapp show --name customer-api-app-jeet123 --resource-group myResourceGroup
```

7.5 Browse App

```
az webapp browse --name customer-api-app-jeet123 --resource-group myResourceGroup
```

Step 8: Configure CI/CD Pipeline

8.1 Azure Credential for GitHub Actions

a)Go to Azure Portal

b) Go to the Service App (EX)

c) Get Publish Profile

d) Download the .PublishSetting file

e) Go to GitHub Repo

f) Click on Settings

g) Click on Actions

h) Click New repository Secret

i) Name: AZURE_WEBAPP_PUBLISH_PROFILE

j) Paste the entire publish profile XML content

8.2 GitHub Action Workflow YAMl

- Make a folder .github/workflows/ in your repo root
- Create a file named deploy.yml inside github/workflows/

```
===========================================================================
name: Build and Deploy ASP.NET Core API to Azure Web App

on: push: branches: - main # or your default branch

jobs: build-and-deploy: runs-on: ubuntu-latest

steps:
  - name: Checkout Code
    uses: actions/checkout@v3

  - name: Setup .NET SDK
    uses: actions/setup-dotnet@v3
    with:
      dotnet-version: '8.0.x'

  - name: Restore Dependencies
    run: dotnet restore

  - name: Build
    run: dotnet build --configuration Release --no-restore

  - name: Publish
    run: dotnet publish -c Release -o published

  - name: Deploy to Azure Web App
    uses: azure/webapps-deploy@vv2
    with:
      app-name: 'customer-api-app'         # your Azure Web App name
      publish-profile: ${{ secrets.AZURE_WEBAPP_PUBLISH_PROFILE }}  # your secret name
      package: ./published
```

==

8.3 Push and Trigger Pipeline

```
git add .github/workflows/deploy.yml

git commit -m "Add GitHub Actions workflow for Azure deployment"

git push origin main
```

8.4 Azure SQL Connection String

a) Go to your Azure SQL Database in the Azure Portal

b) Find the Connection strings section in the left menu

c) Copy the *ADO.NET* connection string

```
Server=tcp:.database.windows.net,1433;Initial Catalog=;Persist Security Info=False;User ID=;Password=;MultipleActiveResultSets=False;Encrypt=True;TrustServerCertificate=False;Connection Timeout=30;
```

8.5 Configure Connection String in Azure App Service

a) Go to Azure Portal

b) Go to App Services

c) Go to Your App

d) Open Configurations

e) Find the Connection String tab

f) Add a New String Configuration

g) Name: DefaultConnection (Match Code)

h) Value: “your full Azure SQL connection string”

i) Type: SQL Azure

j) Save Changes

k) Restart the App

l) URL: https://customer-api-app-jeet123.azurewebsites.net/swagger

8.6 Update appsettings.json

```
{
"Logging": {
"LogLevel": {
"Default": "Information",
"Microsoft.AspNetCore": "Warning"
}
},
"AllowedHosts": "*",
"ConnectionStrings": {
"DefaultConnection":
"Server=tcp:myuniquesqlserver1234567.database.windows.net,1433;Initial
Catalog=mySampleDB;Persist Security Info=False;User
ID=azureuser;Password=MyP@ssword123!;MultipleActiveResultSets=False;Encrypt=True;
TrustServerCertificate=False;Connection Timeout=30;"
}
}
```

8.7 Modify Program.cs

```
using CustomerApi.Models;
using CustomerApi.Data; // Your EF Core DbContext namespace
using Microsoft.EntityFrameworkCore;
using Microsoft.OpenApi.Models;

var builder = WebApplication.CreateBuilder(args);

// Add DbContext with SQL Server connection
builder.Services.AddDbContext<AppDbContext>(options =>
options.UseSqlServer(builder.Configuration.GetConnectionString("DefaultConnection")));

// Add Swagger services
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen(options =>
{
options.SwaggerDoc("v1", new OpenApiInfo { Title = "Customer API", Version = "v1" });
});

var app = builder.Build();

// Serve a nice custom HTML page at root "/"
app.MapGet("/", () => Results.Content(@"
<!DOCTYPE html>
<html lang='en'>
<head>
<meta charset='UTF-8' />
<meta name='viewport' content='width=device-width, initial-scale=1' />
<title>Welcome to Customer API</title>
<style>
body { font-family: Arial, sans-serif; background: #f0f4f8; text-align: center; padding:
50px; }
h1 { color: #2a9df4; }
p { font-size: 18px; color: #555; }
a.button {
display: inline-block;
margin-top: 20px;
padding: 12px 25px;
font-size: 18px;
color: white;
background-color: #2a9df4;
border-radius: 6px;
text-decoration: none;
}
a.button:hover {
background-color: #1c7ed6;
}
</style>
</head>
<body>
<h1>Welcome to Customer API</h1>
<p>This API lets you manage customers with ease.</p>
<a href='/swagger' class='button'>View API Documentation</a>
</body>
</html>", "text/html"));

// Enable Swagger UI at "/swagger"
app.UseSwagger();
app.UseSwaggerUI(c =>
{
c.SwaggerEndpoint("/swagger/v1/swagger.json", "Customer API V1");
c.RoutePrefix = "swagger"; // Swagger UI is at /swagger now
});

// CRUD endpoints for Customers using EF Core

app.MapGet("/customers", async (AppDbContext db) =>
await db.Customers.ToListAsync());

app.MapGet("/customers/{id}", async (int id, AppDbContext db) =>
{
var customer = await db.Customers.FindAsync(id);
return customer is not null ? Results.Ok(customer) : Results.NotFound();
});

app.MapPost("/customers", async (Customer newCustomer, AppDbContext db) =>
{
db.Customers.Add(newCustomer);
await db.SaveChangesAsync();
return Results.Created($"/customers/{newCustomer.Id}", newCustomer);
});

app.MapPut("/customers/{id}", async (int id, Customer updatedCustomer, AppDbContext
db) =>
{
var existing = await db.Customers.FindAsync(id);
if (existing is null) return Results.NotFound();

existing.Name = updatedCustomer.Name;
existing.Email = updatedCustomer.Email;
existing.Phone = updatedCustomer.Phone;
await db.SaveChangesAsync();

return Results.Ok(existing);
});

app.MapDelete("/customers/{id}", async (int id, AppDbContext db) =>
{
var customer = await db.Customers.FindAsync(id);
if (customer is null) return Results.NotFound();

db.Customers.Remove(customer);
await db.SaveChangesAsync();

return Results.NoContent();
});

app.Run();
```

8.8 Final Website

https://customer-api-app-jeet123.azurewebsites.net/swagger/index.html

Step 8 (Azure DevOps) [Alternative]

Pre: DevOps Sign Up

https://aex.dev.azure.com/

Project Name: CustomerApiProject

https://dev.azure.com/JeetPatelCompSci/CustomerApiProject

1. Go to your Azure DevOps Project

- Open your Azure DevOps project portal:
  https://dev.azure.com/<your-org>/<your-project>

2. Create a Service Connection Using Wizard

- Navigate:
  Project Settings (bottom left) → Service connections → New service connection
- Select Azure Resource Manager
- Choose the option:
  "Automatic" (Recommended)
- Sign in with your Azure account and pick your subscription
- Select the resource group where your app is (or leave blank for subscription-wide access)
- Name your service connection (e.g., CustomerApiAutoSP)
- Check Grant access permission to all pipelines
- Click Save

Azure DevOps will create the Service Principal for you behind the scenes. No manual CLI needed.

3. Create azure-pipelines.yml in your project root

yaml

CopyEdit

```
trigger:
 - main

 pool:
   vmImage: 'windows-latest'

 variables:
   buildConfiguration: 'Release'

 steps:
 - task: UseDotNet@2
   inputs:
     packageType: 'sdk'
     version: '8.x'

 - task: DotNetCoreCLI@2
   inputs:
     command: 'restore'
     projects: '**/*.csproj'

 - task: DotNetCoreCLI@2
   inputs:
     command: 'build'
     arguments: '--configuration $(buildConfiguration)'

 - task: DotNetCoreCLI@2
   inputs:
     command: 'publish'
     publishWebProjects: true
     arguments: '--configuration $(buildConfiguration) --output $(Build.ArtifactStagingDirectory)'

 - task: PublishBuildArtifacts@1
   inputs:
     pathToPublish: '$(Build.ArtifactStagingDirectory)'
     artifactName: 'drop'

 - task: AzureWebApp@1
   inputs:
     azureSubscription: 'CustomerApiAutoSP'  # Use the exact name you gave the service connection here
     appName: 'customer-api-jeet123'         # Your Azure App Service name
     package: '$(Build.ArtifactStagingDirectory)/**/*.zip'
```

4. Push azure-pipelines.yml to your repository

bash

CopyEdit

```
git add azure-pipelines.yml
 git commit -m "Add Azure DevOps pipeline"
 git push origin main
```

5. Create and run the pipeline in Azure DevOps

- Go to Pipelines in Azure DevOps portal → New pipeline
- Select your repo (GitHub or Azure Repos)
- Select YAML
- Point it to your azure-pipelines.yml file
- Run the pipeline

6. Pipeline runs and deploys your app

- • If you hit "No hosted parallelism" error, submit this form: https://aka.ms/azpipelines-parallelism-request