Lab: Deploy an ASP.NET Core Web API from Visual Studio to Azure App Service

Goal

Create a .NET Web API, publish it to Azure App Service straight from Visual Studio, verify the API (including Swagger/OpenAPI JSON), and enable basic monitoring.

Prerequisites

- **Visual Studio 2022** (latest update) with **ASP.NET and web development** workload.
- **.NET SDK**: .NET 8 (LTS) or .NET 9 (works too).
- **Azure account** with permission to create Resource Groups and App Services.
- **Azure Developer CLI**/**Azure CLI** not required, but useful (optional).

Part 1 — Create a Web API Project (with Swagger)

1. **New Project**
   - Visual Studio → **Create a new project** → **ASP.NET Core Web API** → **Next**.
   - Project name: `Contoso.Todo.Api`
   - Framework: **.NET 8 (Long-term support)** (or .NET 9 if you prefer).
   - ✔ Enable **Use controllers** (optional but recommended).
   - ✔ **Enable OpenAPI support** (important for `/swagger` and `/swagger/v1/swagger.json`).
   - Create.
2. **Add a simple health endpoint** (easy to test after deploy):
   - Add `Controllers/HealthController.cs`:
   ```
   using Microsoft.AspNetCore.Mvc;

   [ApiController]
   [Route("api/[controller]")]
   public class HealthController : ControllerBase
   {
       [HttpGet]
       public IActionResult Get() => Ok(new { status = "Healthy", utc = DateTime.UtcNow });
   }
   ```
   - Run locally (Ctrl+F5). Test:
     - `https://localhost:xxxxx/swagger`
     - `https://localhost:xxxxx/swagger/v1/swagger.json`
     - `https://localhost:xxxxx/api/health`

Part 2 — Prepare Azure Resources (Portal route)

You can let Visual Studio provision these, but doing it once in the portal builds good muscle memory.

1. **Resource Group**
   - Azure Portal → **Resource groups** → **Create** → Name: `rg-contoso-webapi-dev` → Region close to you (e.g., **Central India** or **South India**).
2. **App Service Plan**
   - Create → **App Service plan** (Windows or Linux).
   - Name: `plan-contoso-dev`
   - OS:
     - **Windows** (simpler for Web Deploy) **or**
     - **Linux** (good for scaling/cost).
   - SKU: Start with **Free (F1)** if available or **B1 (Basic)** for more capabilities.
   - Region: Same as RG.
3. **App Service (Web App)**
   - Create → **Web App**
   - Publish: **Code**
   - Runtime stack:
     - For **Windows**: choose **.NET** (64-bit).
     - For **Linux**: choose **.NET 8 (LTS)** or **.NET 9**.
   - Operating System: match your plan.
   - Region: same as plan.
   - App name: `contoso-todo-api-<yourinitials>` (must be globally unique).
   - Plan: select `plan-contoso-dev`.
   - Review + Create.

The default public base URL will be:

```
https://<app_name>.azurewebsites.net
```

Part 3 — Publish from Visual Studio (one-click style)

1. **Right-click project → Publish**
2. **Choose a target**: **Azure → Azure App Service (Windows)** or **(Linux)** depending on your plan.
3. **Sign in** to Azure (if prompted).
4. **Select existing** Web App you created → **Finish**.
5. In the **Publish** profile:
   - **Configuration**: `Release`
   - **Target Framework**: should match your project
   - **Deployment Mode**: Framework-dependent
   - **File Publish Options**: leave defaults
6. Click **Publish**.

Visual Studio will build and push via Web Deploy/Zip Deploy and then open the site.

Part 4 — Verify Your Deployment & Find the Endpoint

1. **Base site check**
    - Open:
    - `https://<app_name>.azurewebsites.net`
    - For a minimal API template, the root may be blank. That's expected.
2. **Swagger UI**
    - If you enabled OpenAPI support:
    - `https://<app_name>.azurewebsites.net/swagger`
3. **OpenAPI JSON (your machine-readable spec)**
    - The conventional path:
    - `https://<app_name>.azurewebsites.net/swagger/v1/swagger.json`
    - This is the file clients/tools often ask for.
4. **Your Health endpoint**
    - Test:
    - `https://<app_name>.azurewebsites.net/api/health`
    - Expected 200 OK with `{ "status": "Healthy", ... }`
5. **What is my "API endpoint address"?**
    - The **base address** is your App Service URL:
    - `https://<app_name>.azurewebsites.net`
    - Combine with your route, e.g.:
        - `GET https://<app_name>.azurewebsites.net/api/health`
        - `GET https://<app_name>.azurewebsites.net/api/todos` (if you add a `TodosController`)
    - Swagger shows all routes & verbs in one place.

Part 5 — Common App Service Settings (post-deploy)

In Azure Portal → Your Web App:

1. **General Settings**
    - **Platform**: 64-bit
    - **HTTPS Only**: On
    - **Always On**: On (recommended for background tasks and faster warmups; needs B1+).
2. **Configuration**
    - **Application settings** → add environment variables (e.g., `ASPNETCORE_ENVIRONMENT=Production`).
    - **Connection strings** → store DB/server creds here (use `SqlAzure` type for Azure SQL). In code, read via `IConfiguration`.
3. **CORS** (if you have a SPA front-end)
    - **CORS** → Add allowed origins (e.g., `https://myspa.azurewebsites.net`).
4. **Health check**
    - **Health check** → Path: `/api/health` → Save.
5. **App Service Logs**

    - Turn on **Application logging (filesystem)** for quick troubleshooting (temporary).
    - Use **Log stream** to watch logs in real time.
6. **Application Insights** (recommended)
    - Enable from **Application Insights** blade and link or create a new instance to get request traces, failures, and performance metrics.

Part 6 — (Optional) Deploy via CI/CD from Visual Studio

1. In the **Publish** target dialog, choose **GitHub Actions** when available, or
2. In Azure Portal → Web App → **Deployment Center** → **GitHub Actions**.
3. Azure generates a workflow (`.github/workflows/…yml`) that builds & deploys on push to `main`.

Part 7 — Troubleshooting Checklist

- **HTTP 404 on `/swagger`**
    - Ensure project was created with **Enable OpenAPI support**.
    - In `Program.cs` verify:

    ```
    if (app.Environment.IsDevelopment())
    {
        app.UseSwagger();
        app.UseSwaggerUI();
    }
    ```

    If you want Swagger also in Production, remove the `IsDevelopment()` guard and call `UseSwagger()`/`UseSwaggerUI()` unconditionally (acceptable for dev/test; review before prod).

- **HTTP 500 after deploy**
    - Check **Log stream** and **Application Insights** traces.
    - Verify **Runtime stack**/.NET version in App Service matches your target (especially on Linux).
    - Confirm your **Connection strings** and required **App settings** exist in the portal.
- **Publish fails from Visual Studio**
    - Recreate the **Publish profile** (right-click Publish → New).
    - Ensure you selected the correct **Subscription/Resource Group/App**.
    - If using corporate proxy, ensure VS can reach Azure endpoints.
    - Try **Clean** + **Rebuild** before publishing.
- **CORS errors from browser SPA**
    - Configure **CORS** in App Service (allowed origins), not just in the API.
- **API Management integration issues**
    - First verify the API works directly at the App Service URL.
    - Then import the API in **API Management** from the **OpenAPI URL**:

    ```
    https://<app_name>.azurewebsites.net/swagger/v1/swagger.json
    ```

    - Start with a **new version** or **new API** to avoid conflicts.

Part 8 — Stretch Goals (useful in real projects)

- **Deployment Slots** (e.g., `staging`)
    - Create a slot, publish to `staging`, test, then **Swap** to `production` for zero-downtime releases.
- **Managed Identity** + Azure SQL
    - Enable **System-assigned managed identity** on the Web App.
    - In Azure SQL → Add Active Directory admin & grant the Web App's identity `db_datareader/db_datawriter` or custom roles.
    - Use `Authentication=Active Directory Default` in your connection string (no secrets).
- **Custom Domain + Free TLS**
    - Map your domain (e.g., `api.contoso.com`) and use **App Service Managed Certificate**.

Deliverables / Validation

- ✅ App Service URL responds:
    - `GET https://<app_name>.azurewebsites.net/api/health` → 200 OK JSON.
- ✅ Swagger UI:
    - `GET https://<app_name>.azurewebsites.net/swagger`
- ✅ OpenAPI JSON:
    - `GET https://<app_name>.azurewebsites.net/swagger/v1/swagger.json`
- ✅ Log stream shows requests hitting your endpoints.
- ✅ (Optional) App Insights displays requests, failures, and dependencies.

Cleanup (to avoid charges)

- Delete **Resource Group** `rg-contoso-webapi-dev` (this removes the plan and app).