Azure Deployment Guide

Angular + .NET Web API + Azure Table Storage (Free Tier Only)

1. Objective

This guide helps you deploy a full-stack web application on Microsoft Azure using only free services.

By the end of this guide, you will have:

- An Angular frontend deployed on Azure
- A .NET Web API deployed on Azure
- Data stored in Azure Table Storage
- A fully working end-to-end cloud application

2. Final Architecture

Browser

  |

Angular Frontend → Azure Static Web Apps (Free)

  |

.NET Web API → Azure App Service (F1 – Free)

  |

Azure Table Storage (Free)

3. Prerequisites

Before starting, ensure you have:

- Azure account (Azure for Students preferred)
- GitHub account
- Node.js and Angular CLI installed
- .NET SDK 6 or 7 installed
- A working Angular project
- A working .NET Web API project

4. Backend Project Structure (Important)

Your .NET Web API project must follow this structure:

```
MyApiProject
│
├── Controllers
│   └── UsersController.cs
│
├── Models
│   └── UserEntity.cs
│
├── appsettings.json
├── Program.cs
└── MyApiProject.csproj
```

If the Controllers or Models folders do not exist, create them.

5. Create Azure Table Storage

Step 5.1: Create Storage Account

1. Open Azure Portal
2. Click Create a resource
3. Search for Storage account
4. Click Create

Fill the details:

- Resource Group: Create new
- Storage account name: Any unique name
- Performance: Standard
- Redundancy: LRS

Click Create

Step 5.2: Create Table

1. Open the created Storage Account
2. Go to Data Storage → Tables
3. Click + Table
4. Table name: Users
5. Click Create

6. Configure .NET Web API

Step 6.1: Install Required Package

Open terminal inside the API project folder and run:

```
dotnet add package Azure.Data.Tables
```

Step 6.2: Create Entity Model

1. Right-click on the project
2. Click Add → New Folder
3. Name the folder: Models
4. Right-click the Models folder
5. Click Add → Class
6. Name the file: UserEntity.cs

Paste the following code:

```
using Azure;
using Azure.Data.Tables;


namespace MyApiProject.Models;


public class UserEntity : ITableEntity
{
   public string PartitionKey { get; set; } = "USER";
   public string RowKey { get; set; } = Guid.NewGuid().ToString();
```

```
    public string Name { get; set; }
    public string Email { get; set; }

    public DateTimeOffset? Timestamp { get; set; }
    public ETag ETag { get; set; }
}
```

Step 6.3: Configure Table Storage Service

Open Program.cs.

Add the following before builder.Build():

```
builder.Services.AddSingleton(sp =>
{
    var config = sp.GetRequiredService<IConfiguration>();
    return new TableServiceClient(config["StorageConnection"]);
});
```

Step 6.4: Enable CORS

In Program.cs, add:

```
builder.Services.AddCors(options =>
{
    options.AddPolicy("AllowAll",
        policy => policy.AllowAnyOrigin()
```

```
                .AllowAnyHeader()
                .AllowAnyMethod());
});
```

After var app = builder.Build();, add:

```
app.UseCors("AllowAll");
```

Step 6.5: Create API Controller

1. Right-click the project
2. Click Add → New Folder
3. Name the folder: Controllers
4. Right-click Controllers
5. Click Add → Controller
6. Choose API Controller – Empty
7. Name it: UsersController

Paste the following code:

```
using Azure.Data.Tables;
using Microsoft.AspNetCore.Mvc;
using MyApiProject.Models;

namespace MyApiProject.Controllers;

[ApiController]
[Route("api/users")]
```

```
public class UsersController : ControllerBase
{
    private readonly TableClient _table;

    public UsersController(TableServiceClient serviceClient)
    {
        _table = serviceClient.GetTableClient("Users");
        _table.CreateIfNotExists();
    }

    [HttpPost]
    public async Task<IActionResult> AddUser(UserEntity user)
    {
        await _table.AddEntityAsync(user);
        return Ok(user);
    }

    [HttpGet]
    public Pageable<UserEntity> GetAllUsers()
    {
        return _table.Query<UserEntity>();
    }
}
```

Step 6.6: Local Configuration

Open appsettings.json and add:

"StorageConnection": "UseDevelopmentStorage=true"

Step 6.7: Run API Locally

dotnet run

Test in browser:

https://localhost:<port>/api/users

7. Deploy .NET API to Azure

Step 7.1: Create App Service

1. Azure Portal → Create resource
2. Search App Service
3. Click Create

Fill:

- Runtime: .NET 6 or 7
- OS: Windows
- Pricing Plan: F1 (Free)

Step 7.2: Deploy API

Option 1:

- Right-click project in Visual Studio
- Click Publish
- Choose Azure App Service

Option 2:

- Push code to GitHub
- Use GitHub Actions

Step 7.3: Configure Storage Connection in Azure

1. Open App Service
2. Go to Configuration
3. Add Application Setting:

Name	Value
StorageConnection	Storage account connection string

4. Save
5. Restart App Service

Step 7.4: Test API on Azure

https://your-api-name.azurewebsites.net/api/users

8. Angular Frontend Deployment

Step 8.1: Update API URL

Open:

src/environments/environment.prod.ts

```
export const environment = {
  apiUrl: 'https://your-api-name.azurewebsites.net'
};
```

Step 8.2: Build Angular App

```
ng build --configuration production
```

Step 8.3: Deploy Using Azure Static Web Apps

1. Azure Portal → Static Web Apps
2. Click Create
3. Connect GitHub repository
4. Framework: Angular
5. App location: /
6. Output location: dist/project-name
7. Tier: Free

9. Verification

- Open Static Web App URL
- Submit data
- Refresh page
- Data should persist from Azure Table Storage

10. Common Issues

Issue	Solution
CORS error	Verify CORS configuration
403 error	Check storage connection string
No data saved	Ensure PartitionKey exists
Angular page blank	Verify output folder
API works locally only	Restart App Service

11. Cleanup After Hackathon

To avoid any charges:

- Delete the entire Resource Group

12. Summary

You have successfully:

- Deployed an Angular frontend
- Deployed a .NET backend
- Used Azure Table Storage
- Built a complete cloud application using free Azure services