LOWCADEMY

.NET Development Setup Guide MacBook (macOS)

A beginner-friendly, step-by-step installation and verification handbook for VS Code, direct Azure SQL development access, and later Azure deployment.

Designed by Ankit Gangrade
Lowcademy | www.lowcademy.com
Audience: Freshers and developers setting up a new machine.

What you will build

A .NET 10 Web API running on your computer. It connects securely over the internet to an Azure SQL development database. Later, you will publish the API to Azure App Service. No local SQL Server or Docker is needed.

```
Your computer -> VS Code -> .NET Web API -> Azure SQL (DEV)
                                         |
                                  Later: Azure App Service
```

Only the required software and cloud services

Install on computer	Use
.NET 10 SDK	Build and run ASP.NET Core applications
Visual Studio Code	Edit and run the project
C# Dev Kit extension	C# language support and debugging
SQL Server (mssql) extension	Connect to and query Azure SQL
Azure CLI	Sign in to Azure and deploy later
One AI coding extension (Codex OR Claude Code)	AI-assisted coding inside VS Code

Azure services: Azure subscription + Azure SQL logical server + Azure SQL DEV database (now); Azure App Service + App Service Plan (when deploying). These are cloud resources, not local installations.

Not required: Visual Studio IDE, local SQL Server, SSMS, Docker, Node.js, Homebrew, Git, Postman, or an Azure DevOps account for this first setup.

1. Before you start

• Use a computer with administrator rights, a stable internet connection, and enough free disk space (approximately 5 GB or more is a practical starting point).

• Have a Microsoft Azure subscription or access to a subscription where you can create SQL and App Service resources.

• Know which Azure SQL DEV database you should use. If your organization manages it, ask the administrator for the server name, database name, and your login/access.

• Use only a development database. Never test inserts or deletes against production.

2. Install .NET 10 SDK

2.1 Download and install

Open Microsoft .NET 10 downloads. Under SDK, download the macOS Arm64 installer for Apple Silicon (M-series), or macOS x64 for an Intel Mac. Open the downloaded .pkg file and follow the prompts.

2.2 Check the installation

```
dotnet --version
```

> Installation check: Terminal should print a version starting with 10. (for example, 10.0.xxx). If Terminal says command not found, close and reopen Terminal, then try again.

3. Install Visual Studio Code

3.1 Download VS Code

Visit Visual Studio Code. Download the macOS version. Move VS Code into the Applications folder and launch it.

> Installation check: VS Code opens and shows the Welcome page. You can select File > Open Folder.

4. Install the two VS Code extensions

4.1 Install C# Dev Kit

In VS Code, open Extensions (four-square icon on the left). Search C# Dev Kit. Install the extension published by Microsoft. Accept any required dependencies.

> Installation check: The extension appears under Installed and is enabled.

4.2 Install SQL Server (mssql)

In the same Extensions view, search SQL Server (mssql) and install the extension published by Microsoft.

> Installation check: You can open the SQL Server/MSSQL view or find MS SQL: Connect from the Command Palette (Ctrl+Shift+P on Windows; Cmd+Shift+P on macOS).

5. Install Azure CLI

5.1 Install from Microsoft

Follow the macOS installer (PKG) method on Microsoft Azure CLI for macOS. Download the official package and complete the installer. Homebrew is not required for this method.

5.2 Verify Azure CLI

In Terminal, run:

```
az --version
```

> Installation check: You see Azure CLI version information. The command should not report “not found” or “not recognized”.

5.3 Sign in to Azure

Run the command below. Complete sign-in in the browser. Select the correct subscription if prompted.

```
az login
az account show --output table
```

> Installation check: The second command displays your Azure subscription details. If you have more than one subscription, use az account set --subscription "SUBSCRIPTION_ID".

6. Choose your AI coding assistant

Install one of these VS Code extensions. Neither is required for .NET itself, but one is needed for the AI-assisted workflow in this guide.

• Codex: Search Codex in VS Code Extensions; choose the official OpenAI extension, install it, and sign in. See official Codex documentation.

• Claude Code: Search Claude Code in VS Code Extensions; choose the official Anthropic extension, install it, and sign in. See official Claude Code documentation.

> Check: Open the assistant panel in VS Code and ask: “Explain how to create a .NET 10 Web API.” If it replies, your sign-in and extension are working. Subscription or usage charges may apply.

7. Create or obtain the Azure SQL DEV database

You can use an existing development database. If it already exists, skip the creation steps and request the connection details from your Azure administrator.

7.1 Create the database in Azure (only if needed)

Open Azure Portal, search SQL databases, and choose Create. Select the subscription/resource group, enter a database name such as NorviaDevDB, and create/select a logical SQL server. Choose an affordable compute configuration and review the estimated cost. Use SQL authentication for this beginner walkthrough; your organization may require Microsoft Entra authentication instead.

7.2 Allow access from your computer

Open the logical SQL server in Azure Portal, then Networking. Configure public network access for selected networks and add your current public IP address as a firewall rule. Save. The network must permit outbound TCP port 1433. If your organization uses private endpoints, ask IT for VPN/private-network access instead.

> Installation check: The server firewall shows your current public IP address as an allowed rule. This is not the same as your local 192.168.x.x address.

7.3 Write down these details

You will need the server name (for example, norvia-dev-sql.database.windows.net), database name (NorviaDevDB), SQL username, and password. Never paste a real password into source code or an AI prompt.

8. Test Azure SQL from VS Code

8.1 Connect

In VS Code, open the Command Palette and select MS SQL: Connect (or add a connection in the SQL Server extension). Use the values below:

- Server: norvia-dev-sql.database.windows.net (replace with yours)
- Database: NorviaDevDB (replace with yours)
- Authentication: SQL Login (for this walkthrough)
- User/password: Your Azure SQL development credentials
- Encryption: Enabled; Trust server certificate: False

8.2 Run a safe test query

Open a new query connected to your DEV database and execute:

```
SELECT DB_NAME() AS DatabaseName, 1 AS ConnectionOK;
```

Installation check: One row appears. DatabaseName matches your DEV database and ConnectionOK is 1. No tables need to be created for this test.

Common error: “Client IP address is not allowed” means the Azure SQL firewall needs your current public IP. “Login failed” usually means the username/password or permissions are incorrect. A timeout can indicate blocked port 1433, wrong server name, or private-network restrictions.

9. Create and run your first .NET Web API

Open Terminal. Choose a folder where you want to keep your project. Run these commands one at a time:

```
mkdir -p ~/Projects/Lowcademy
cd ~/Projects/Lowcademy
dotnet new webapi -n NorviaApi
cd NorviaApi
```

9.1 Start the API

```
dotnet run
```

> Installation check: The terminal displays Now listening on: http://localhost:PORT (your port may vary). Open that exact address in a browser. A new minimal Web API may display a basic response or return 404 at / until you add a root route; either outcome means the web server responded.

Press Ctrl+C in the terminal to stop the application before the next steps.

10. Connect your .NET API to Azure SQL DEV

10.1 Install the SQL client library

In the NorviaApi project folder, run:

```
dotnet add package Microsoft.Data.SqlClient
```

> Installation check: You see a successful package restore. dotnet build should finish with Build succeeded.

10.2 Enable local secrets

Use .NET User Secrets so your SQL password is not stored in the project files.

```
dotnet user-secrets init
```

> Installation check: The command completes successfully and the project file contains a UserSecretsId value.

Set the connection string with the following command. Replace the server, database, username, and password with your own DEV details. Run this only on your own trusted computer.

```
dotnet user-secrets set 'ConnectionStrings:DevSql' \
  'Server=tcp:YOUR_SERVER.database.windows.net,1433;Initial Catalog=YOUR_DEV_DB;User ID=YOUR_USER;Pass
word=YOUR_PASSWORD;Encrypt=True;TrustServerCertificate=False;Connection Timeout=30;'
```

Security: Do not commit credentials, include them in screenshots, or share them with Codex/Claude. For team or production access, prefer Microsoft Entra ID / managed identity when possible.

11. Add a database connectivity test endpoint

In VS Code, open your NorviaApi folder. Open Program.cs and replace its content with the code below. This temporary endpoint confirms that the running API can connect to Azure SQL.

```
using Microsoft.Data.SqlClient;

var builder = WebApplication.CreateBuilder(args);
var app = builder.Build();

app.MapGet("/", () => "Norvia API is running");

app.MapGet("/db-check", async (IConfiguration config) =>
{
    var cs = config.GetConnectionString("DevSql")
        ?? throw new Exception("DevSql is not configured");
    await using var connection = new SqlConnection(cs);
    await connection.OpenAsync();
    await using var command = new SqlCommand("SELECT DB_NAME()", connection);
    var dbName = (string?)await command.ExecuteScalarAsync();
    return Results.Ok(new { connected = true, database = dbName });
});

app.Run();
```

11.1 Build and run

```
dotnet build
dotnet run
```

Installation check: The build succeeds and the terminal prints the local HTTP address. Leave the app running while testing.

11.2 Test from a browser

Copy the exact local address shown in the terminal and append /db-check. Example: http://localhost:5123/db-check (use your actual port).

Installation check: You should receive JSON similar to {"connected":true,"database":"NorviaDevDB"}. This proves the local .NET app connected to Azure SQL.

> If the check fails: Confirm the DevSql secret name, database credentials, Azure SQL firewall rule, and server name. Try the same connection in the VS Code MSSQL extension first. Remove or protect /db-check before publishing; it is a temporary learning endpoint.

12. Prepare for Azure deployment (later)

Once your application works locally, you can host it using Azure App Service. You do not need to install IIS or SQL Server on your computer.

• In Azure Portal, create an App Service Plan and a Web App with a supported .NET 10 runtime. Choose a Linux or Windows runtime supported by your chosen App Service configuration.

• Use Azure CLI (az login) to deploy a published build, or configure a deployment workflow later. Keep development and production environments separate.

• In the Web App configuration, set ConnectionStrings__DevSql to the database connection string only for a development/test deployment. Do not publish your local User Secrets file.

• Allow Azure App Service to reach the database using approved Azure SQL networking rules. Prefer managed identity and private networking for production.

• Open the Web App HTTPS URL and test a normal API endpoint. Check App Service logs if startup fails. Remove the temporary database test endpoint before production release.

Optional command-line deployment example

After creating the Azure Web App and signing in, you can publish locally and deploy the ZIP package. Replace all placeholders. The deployment method and available runtime depend on your Azure configuration.

```
dotnet publish -c Release -o ./publish
cd publish
zip -r ../app.zip .
cd ..
```

```
az webapp deploy --resource-group YOUR_RESOURCE_GROUP \
  --name YOUR_WEB_APP_NAME --src-path app.zip --type zip
```

Deployment check: In Azure Portal, confirm the deployment succeeds, open https://YOUR_WEB_APP_NAME.azurewebsites.net/, and check that the API responds. Verify the Azure app can reach its database using a protected application operation.

13. Final readiness checklist

• ☐ dotnet --version returns 10.x

• ☐ VS Code opens; C# Dev Kit and SQL Server extensions are enabled

• ☐ az --version works and az account show displays your subscription

• ☐ One AI coding extension is installed and authenticated

• ☐ VS Code MSSQL query returns the correct DEV database name

• ☐ dotnet build succeeds

- ☐ The local /db-check endpoint returns connected: true
- ☐ You understand that Azure App Service is created only when ready to deploy

Official references

Microsoft .NET | VS Code | Azure SQL connectivity | Azure App Service .NET

Lowcademy | www.lowcademy.com
Designed by Ankit Gangrade