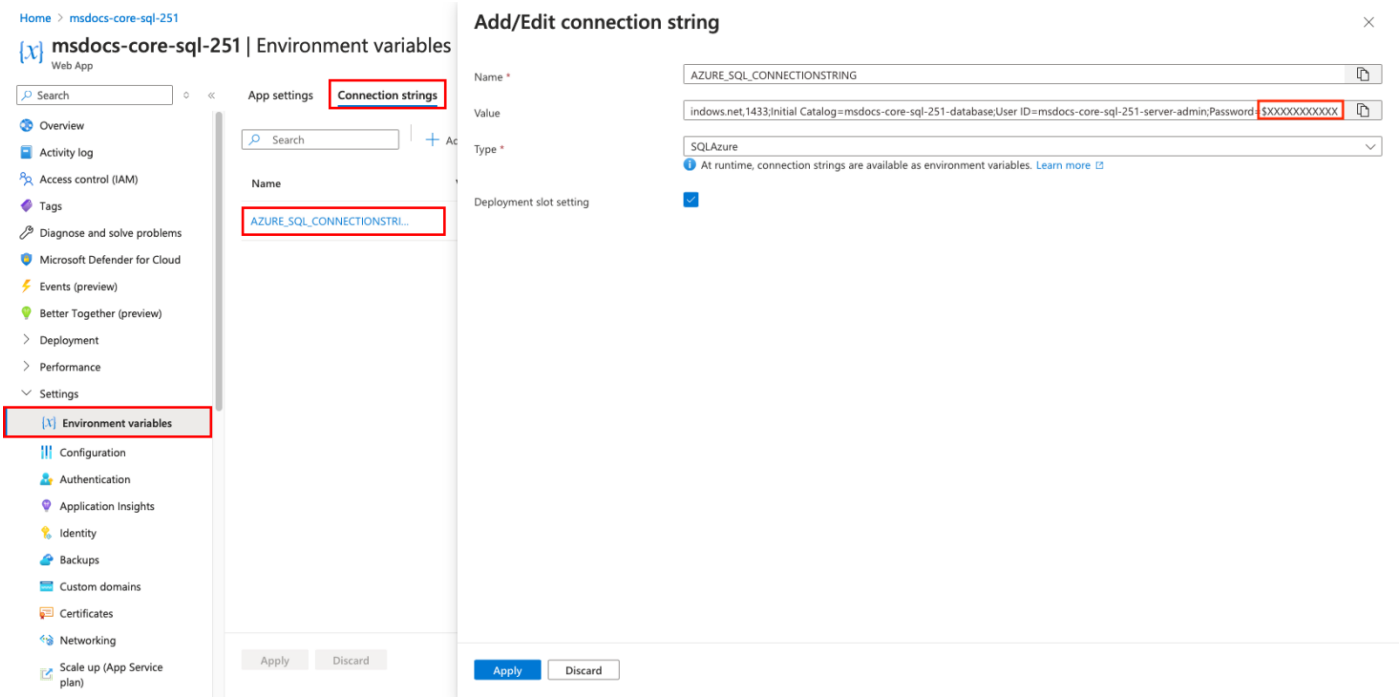
DEPLOY AN ASP.NET CORE AND AZURE SQL DATABASE APP TO AZURE APP SERVICE -2

3. SECURE CONNECTION SECRETS

The creation wizard generated the connectivity variable for you already as .NET connection strings and app settings. However, the security best practice is to keep secrets out of App Service completely. You'll move your secrets to a key vault and change your app setting to Key Vault references with the help of Service Connectors.

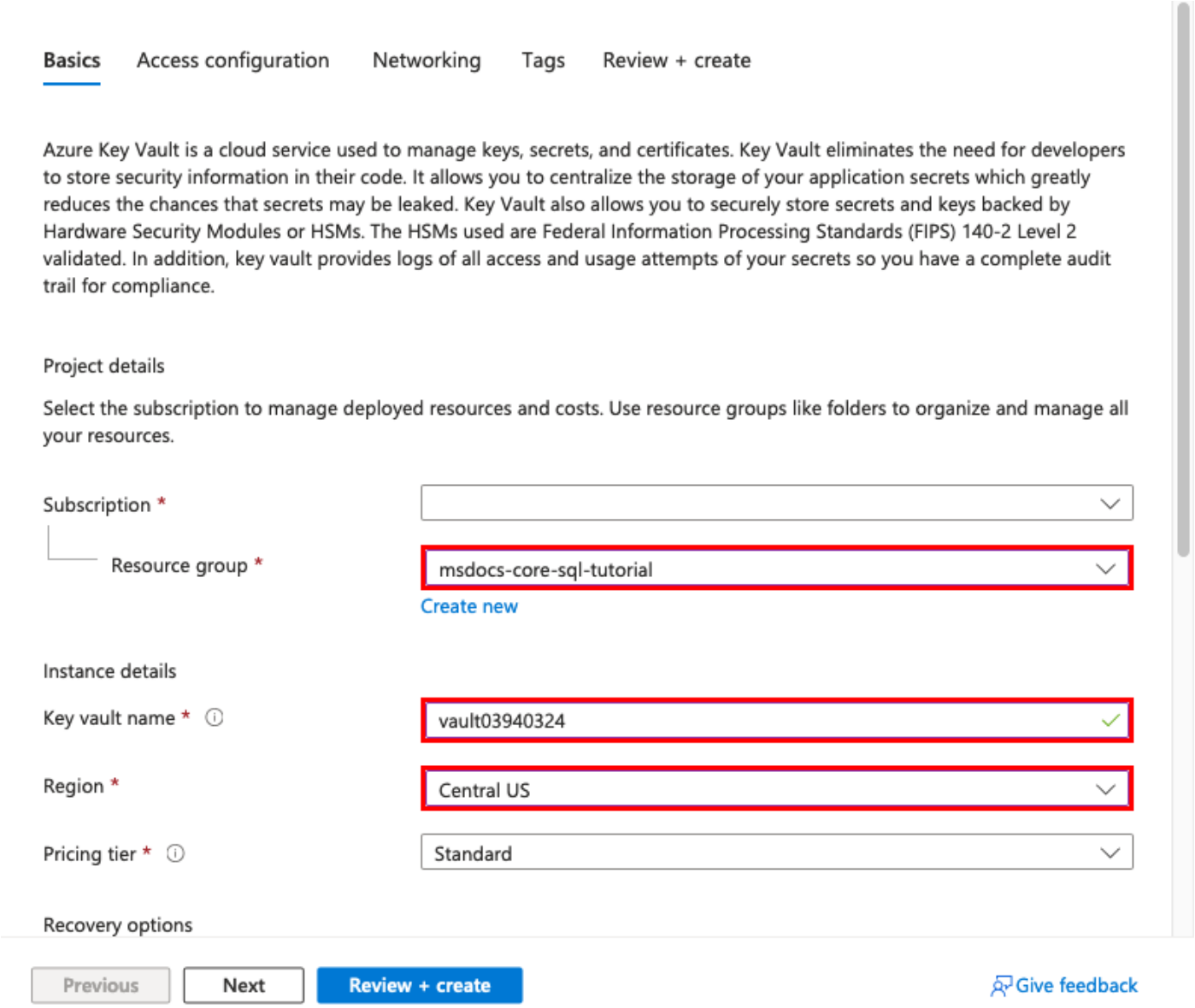
STEP 1: RETRIEVE THE EXISTING CONNECTION STRING

1. In the left menu of the App Service page, select **Settings > Environment variables > Connection strings**.
2. Select **AZURE_SQL_CONNECTIONSTRING**.
3. In **Add/Edit connection string**, in the **Value** field, find the *Password=* part at the end of the string.
4. Copy the password string after *Password=* for use later. This connection string lets you connect to the SQL database secured behind a private endpoint. However, the secrets are saved directly in the App Service app, which isn't the best. Likewise, the Redis cache connection string in the **App settings** tab contains a secret. You'll change this.



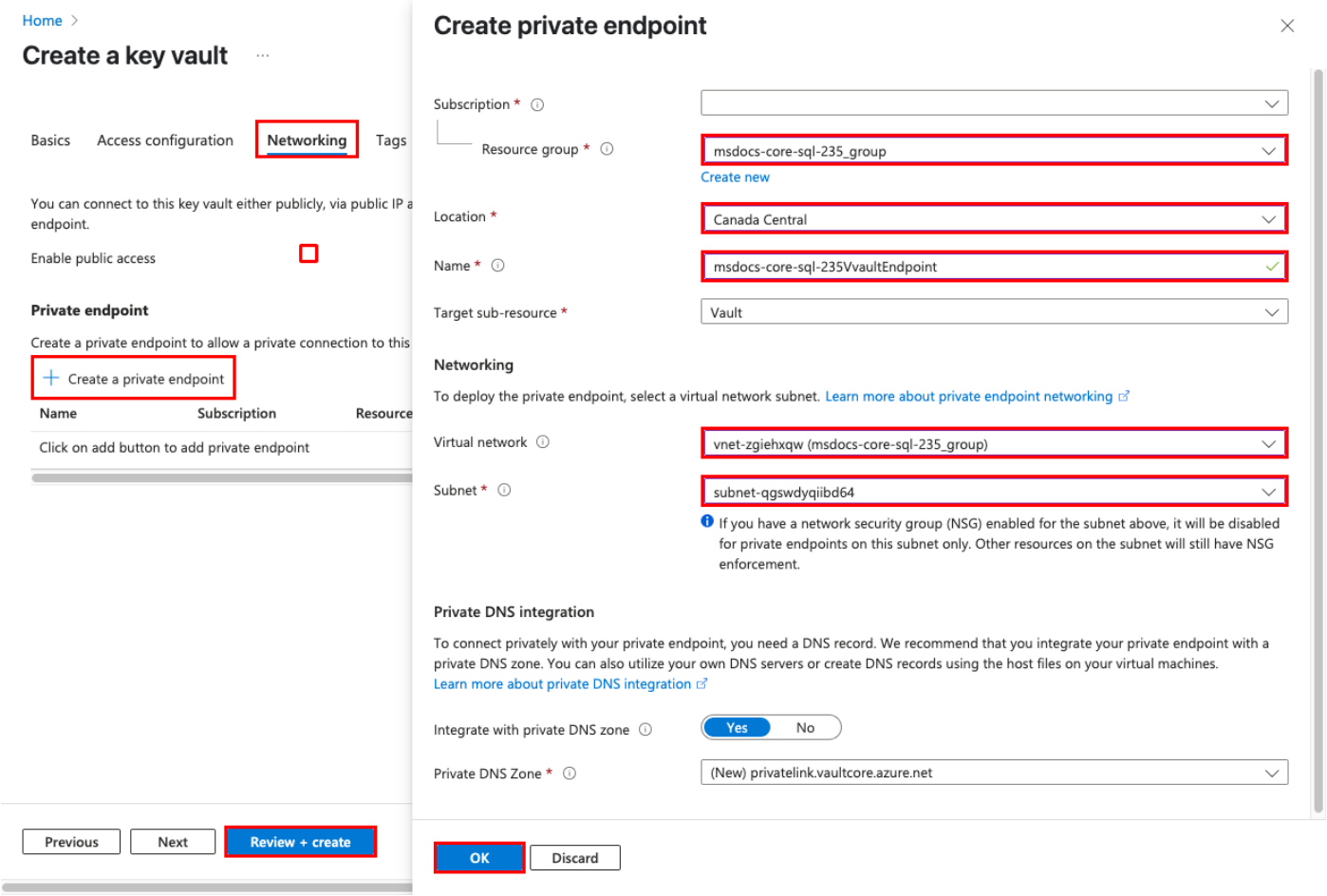
STEP 2: CREATE A KEY VAULT FOR SECURE MANAGEMENT OF SECRETS

1. In the top search bar, type "*key vault*", then select **Marketplace** > **Key Vault**.
2. In **Resource Group**, select **msdocs-core-sql-XYZ_group**.
3. In **Key vault name**, type a name that consists of only letters and numbers.
4. In **Region**, set it to the same location as the resource group.



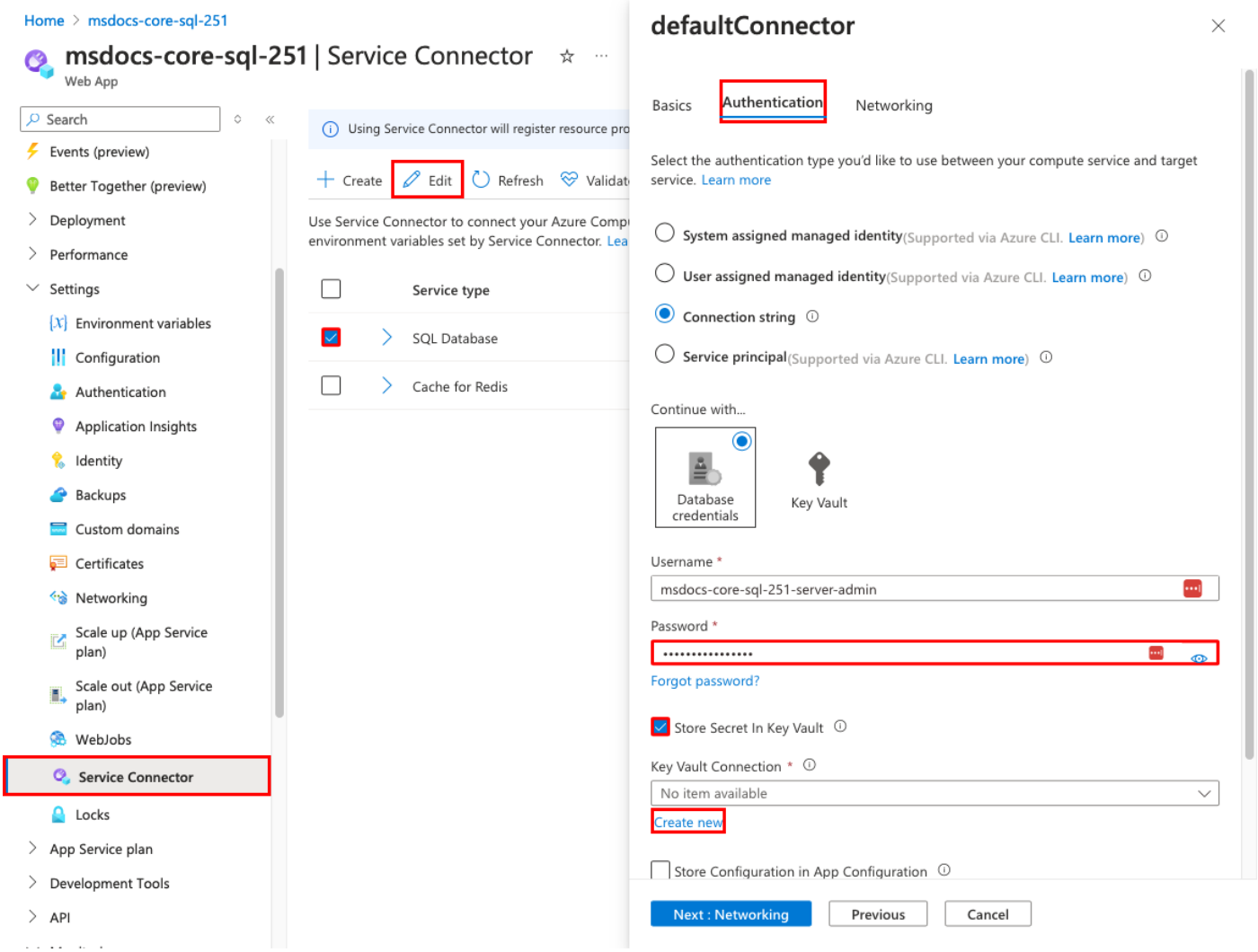
STEP 3: SECURE THE KEY VAULT WITH A PRIVATE ENDPOINT

1. Select the **Networking** tab.
2. Unselect **Enable public access**.
3. Select **Create a private endpoint**.
4. In **Resource group**, select **msdocs-core-sql-XYZ_group**.
5. In the dialog, in **Location**, select the same location as your App Service app.
6. In **Name**, type **msdocs-core-sql-XYZVvaultEndpoint**.
7. In **Location**, select the same location as your App Service app.
8. In **Virtual network**, select the virtual network in the **msdocs-core-sql-XYZ_group** group.
9. In **Subnet**, select the available compatible subnet. The Web App wizard created it for your convenience.
10. Select **OK**.
11. Select **Review + create**, then select **Create**. Wait for the key vault deployment to finish. You should see "Your deployment is complete."



STEP 4: SQL DATABASE CONNECTOR

1. In the top search bar, type *msdocs-core-sql*, then the App Service resource called **msdocs-core-sql-XYZ**.
2. In the App Service page, in the left menu, select **Settings > Service Connector**. There are already two connectors, which the app creation wizard created for you.
3. Select checkbox next to the SQL Database connector, then select **Edit**.
4. Select the **Authentication** tab.
5. In **Password**, paste the password you copied earlier.
6. Select **Store Secret in Key Vault**.
7. Under **Key Vault Connection**, select **Create new**. A **Create connection** dialog is opened on top of the edit dialog.

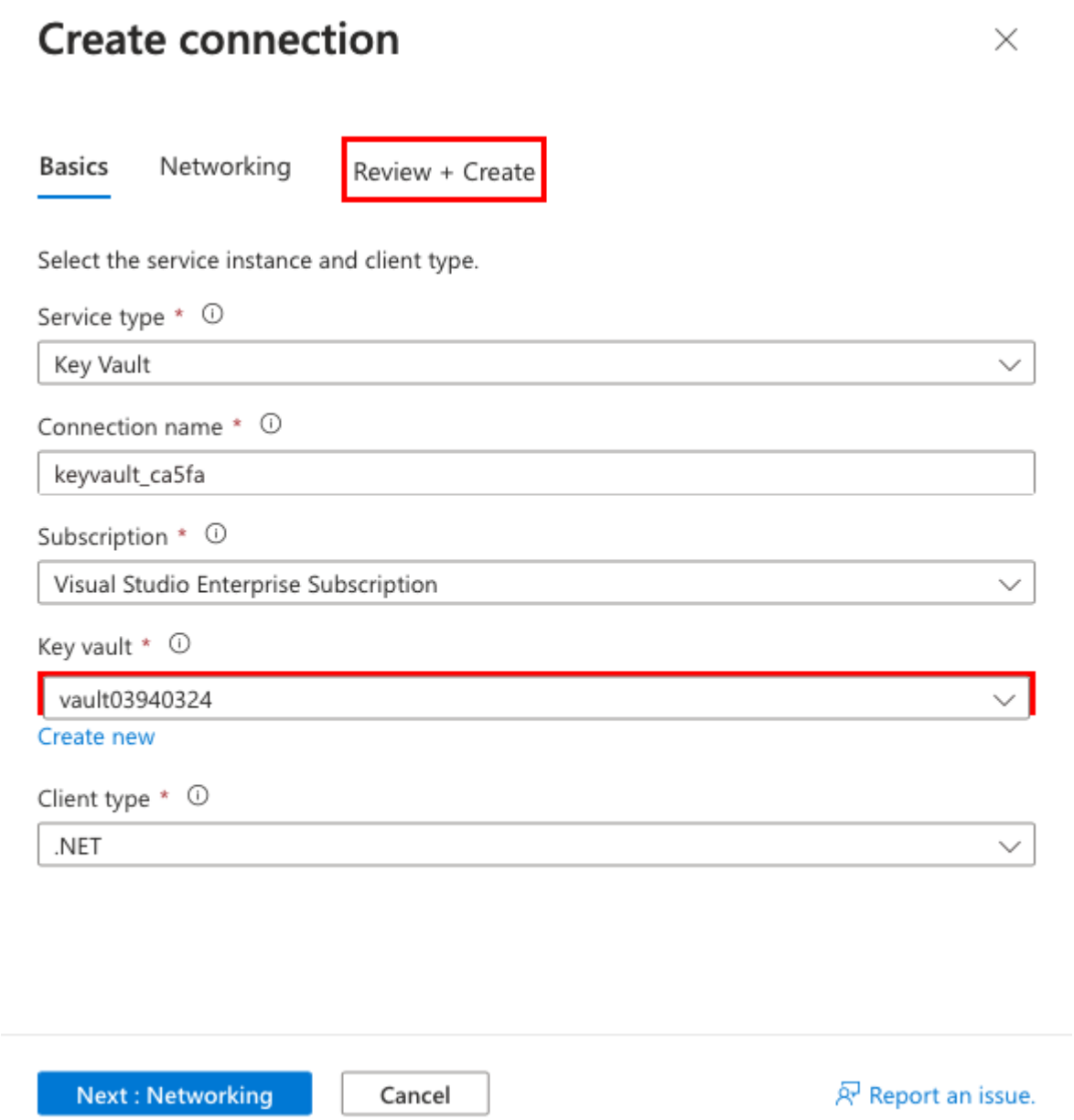


STEP 5: ESTABLISH THE KEY VAULT CONNECTION

1. In the **Create connection** dialog for the Key Vault connection, in **Key Vault**, select the key vault you created earlier.

2. Select **Review + Create**.

3. When validation completes, select **Create**.



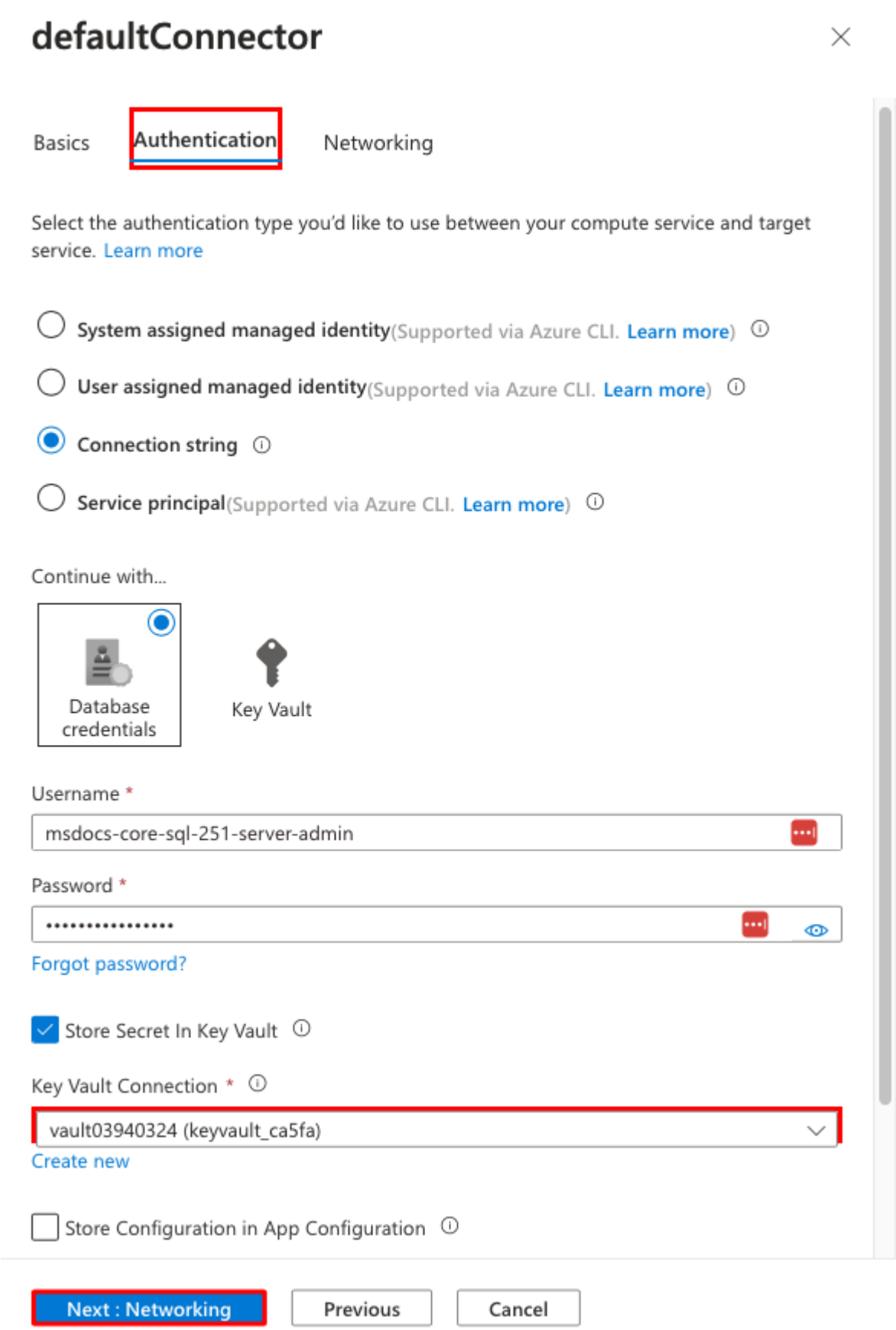
STEP 6: FINALIZE THE SQL DATABASE CONNECTOR SETTINGS

1. You're back in the edit dialog for **defaultConnector**. In the **Authentication** tab, wait for the key vault connector to be created. When it's finished, the **Key Vault Connection** dropdown automatically selects it.

2. Select **Next: Networking**.

3. Select **Configure firewall rules to enable access to target service**. The app creation wizard already secured the SQL database with a private endpoint.

4. Select **Save**. Wait until the **Update succeeded** notification appears.



STEP 7: CONFIGURE THE REDIS CONNECTOR TO USE KEY VAULT SECRETS

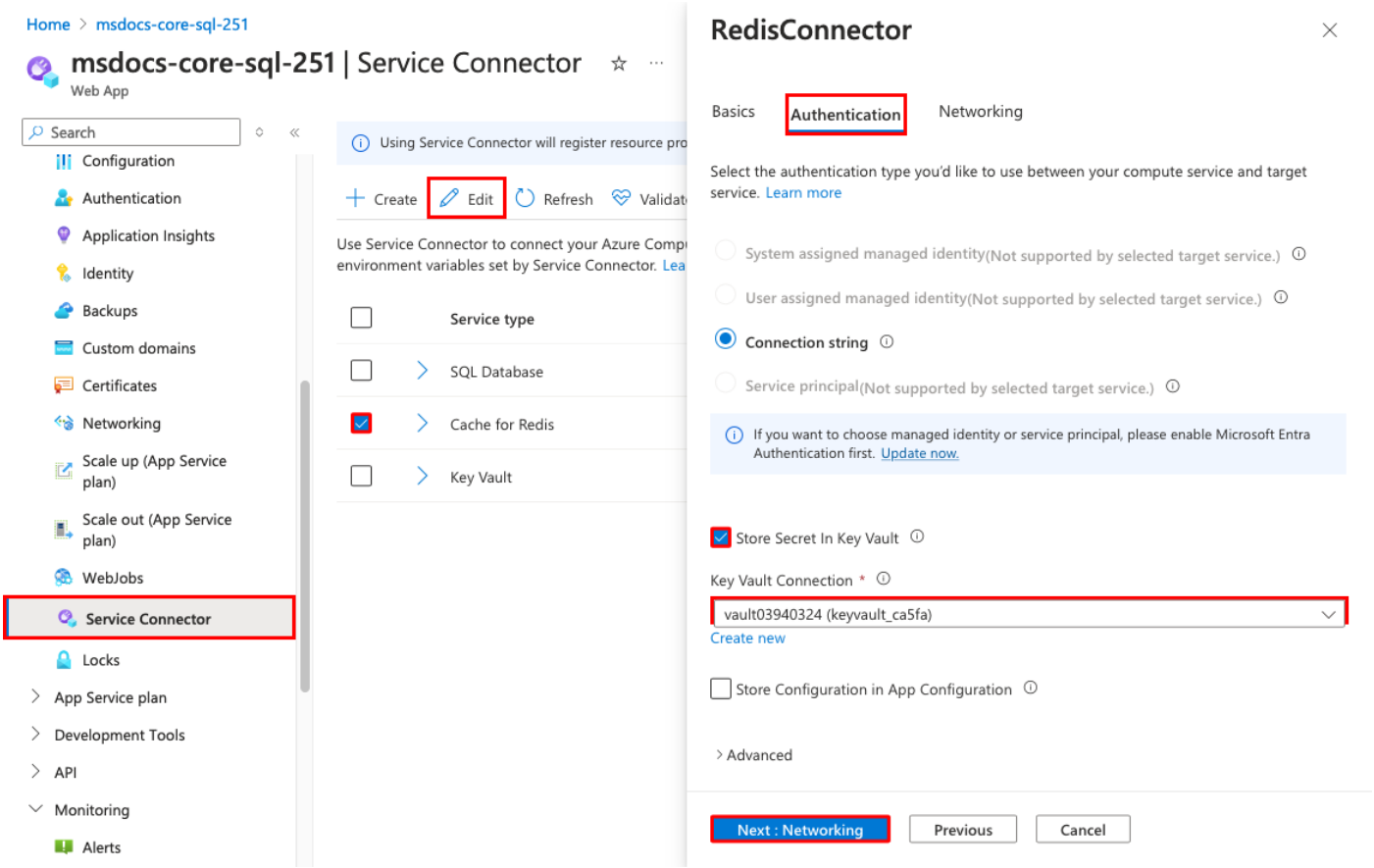
1. In the Service Connector page, select the checkbox next to the Cache for Redis connector, then select **Edit**.
2. Select the **Authentication** tab.
3. Select **Store Secret in Key Vault**.

4. Under **Key Vault Connection**, select the key vault you created.

5. Select **Next: Networking**.

6. Select **Configure firewall rules to enable access to target service**.

7. Select **Save**. Wait until the **Update succeeded** notification appears.

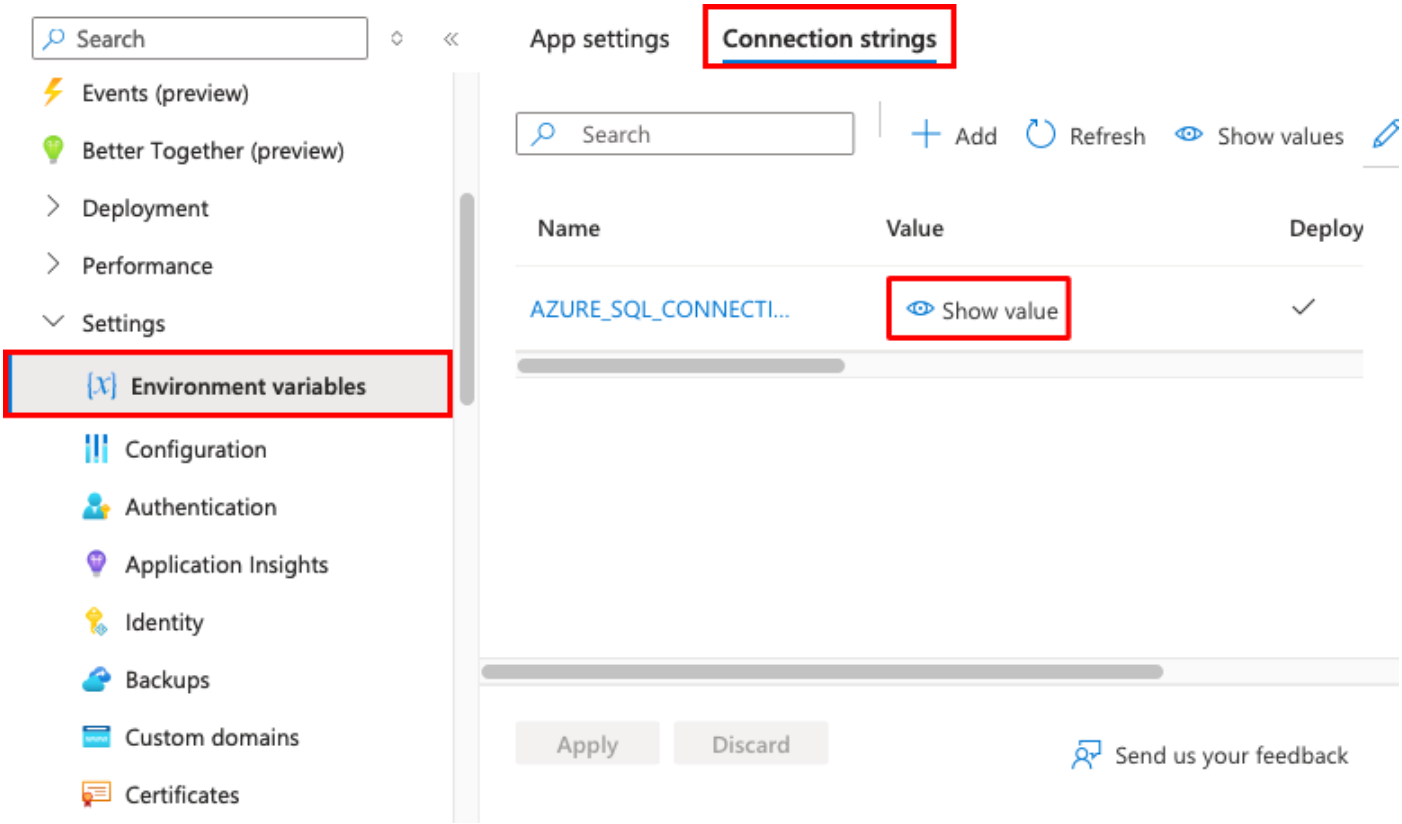


STEP 8: VERIFY THE KEY VAULT INTEGRATION

1. From the left menu, select **Settings > Environment variables > Connection strings** again.

2. Next to **AZURE_SQL_CONNECTIONSTRING**, select **Show value**. The value should be @Microsoft.KeyVault(...), which means that it's a key vault reference because the secret is now managed in the key vault.

3. To verify the Redis connection string, select the **App setting** tab. Next to **AZURE_REDIS_CONNECTIONSTRING**, select **Show value**. The value should be @Microsoft.KeyVault(...) too.



To summarize, the process for securing your connection secrets involved:

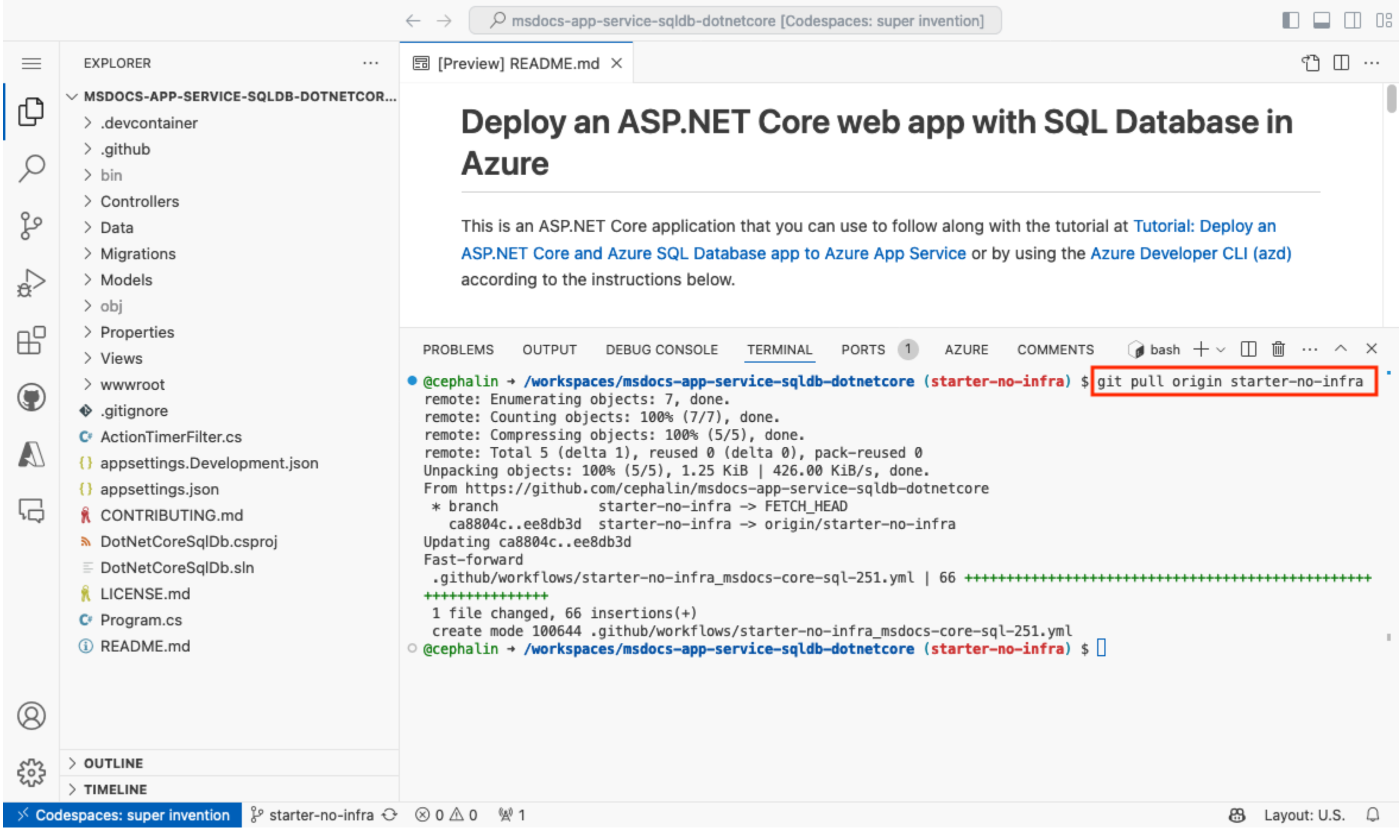
- Retrieving the connection secrets from the App Service app's environment variables.
- Creating a key vault.
- Creating a Key Vault connection with the system-assigned managed identity.
- Updating the service connectors to store the secrets in the key vault.

4. DEPLOY SAMPLE CODE

In this step, you configure GitHub deployment using GitHub Actions. It's just one of many ways to deploy to App Service, but also a great way to have continuous integration in your deployment process. By default, every git push to your GitHub repository kicks off the build and deploy action.

STEP 1:

Back in the GitHub codespace of your sample fork, run git pull origin starter-no-infra. This pulls the newly committed workflow file into your codespace.



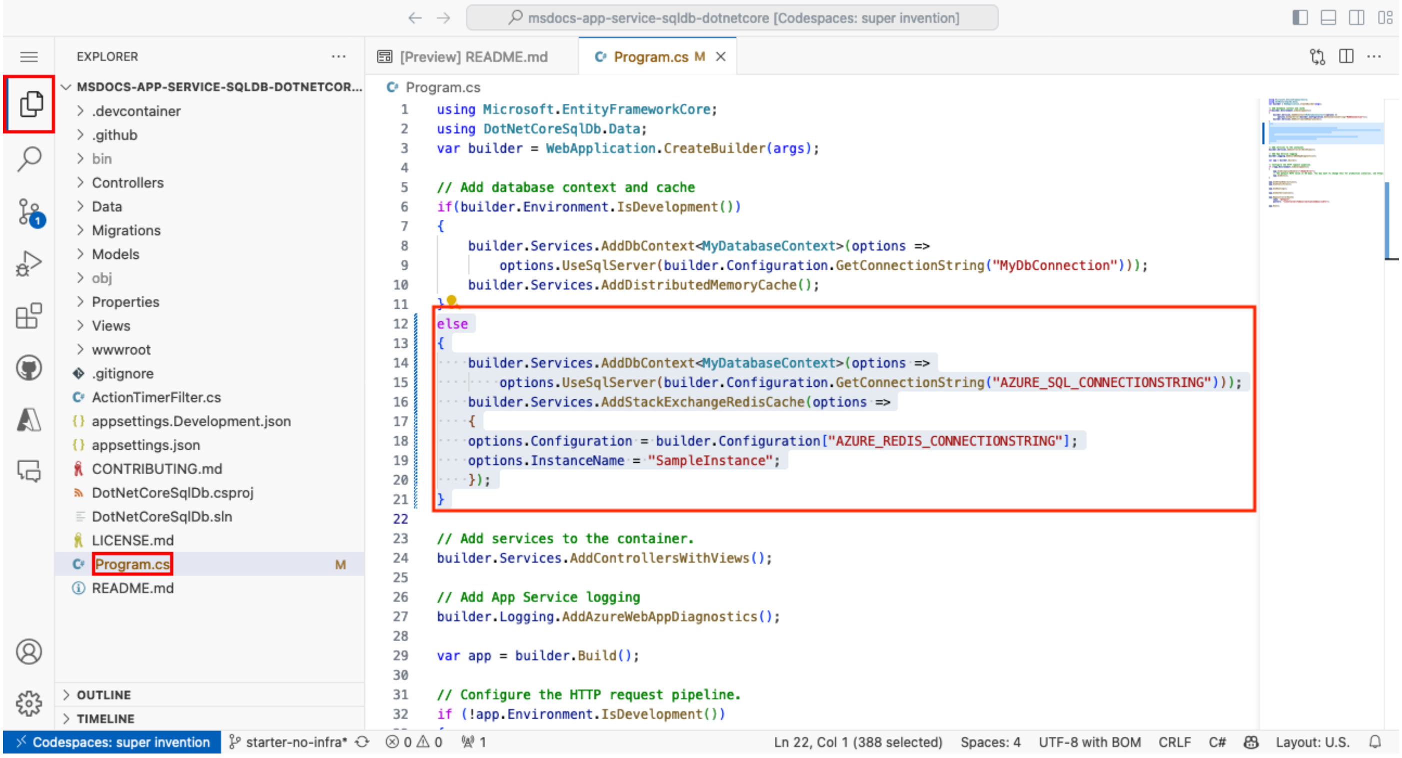
Deploy an ASP.NET Core web app with SQL Database in Azure

This is an ASP.NET Core application that you can use to follow along with the tutorial at Tutorial: Deploy an ASP.NET Core and Azure SQL Database app to Azure App Service or by using the Azure Developer CLI (azd) according to the instructions below.

```
@cephalin → /workspaces/msdocs-app-service-sqldb-dotnetcore (starter-no-infra) $ git pull origin starter-no-infra
remote: Enumerating objects: 7, done.
remote: Counting objects: 100% (7/7), done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 5 (delta 1), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (5/5), 1.25 KiB | 426.00 KiB/s, done.
From https://github.com/cephalin/msdocs-app-service-sqldb-dotnetcore
 * branch            starter-no-infra -> FETCH_HEAD
   ca8804c..ee8db3d  starter-no-infra -> origin/starter-no-infra
Updating ca8804c..ee8db3d
Fast-forward
 .github/workflows/starter-no-infra_msdocs-core-sql-251.yml | 66 ++++++++++++++++++++++++++++++++++++++++++++++++++
+++++++++++++++
 1 file changed, 66 insertions(+)
 create mode 100644 .github/workflows/starter-no-infra_msdocs-core-sql-251.yml
@cephalin → /workspaces/msdocs-app-service-sqldb-dotnetcore (starter-no-infra) $ 
```

STEP 2:

1. Open *Program.cs* in the explorer.
2. Find the commented code (lines 12-21) and uncomment it. This code connects to the database by using AZURE_SQL_CONNECTIONSTRING and connects to the Redis cache by using the app setting AZURE_REDIS_CONNECTIONSTRING.



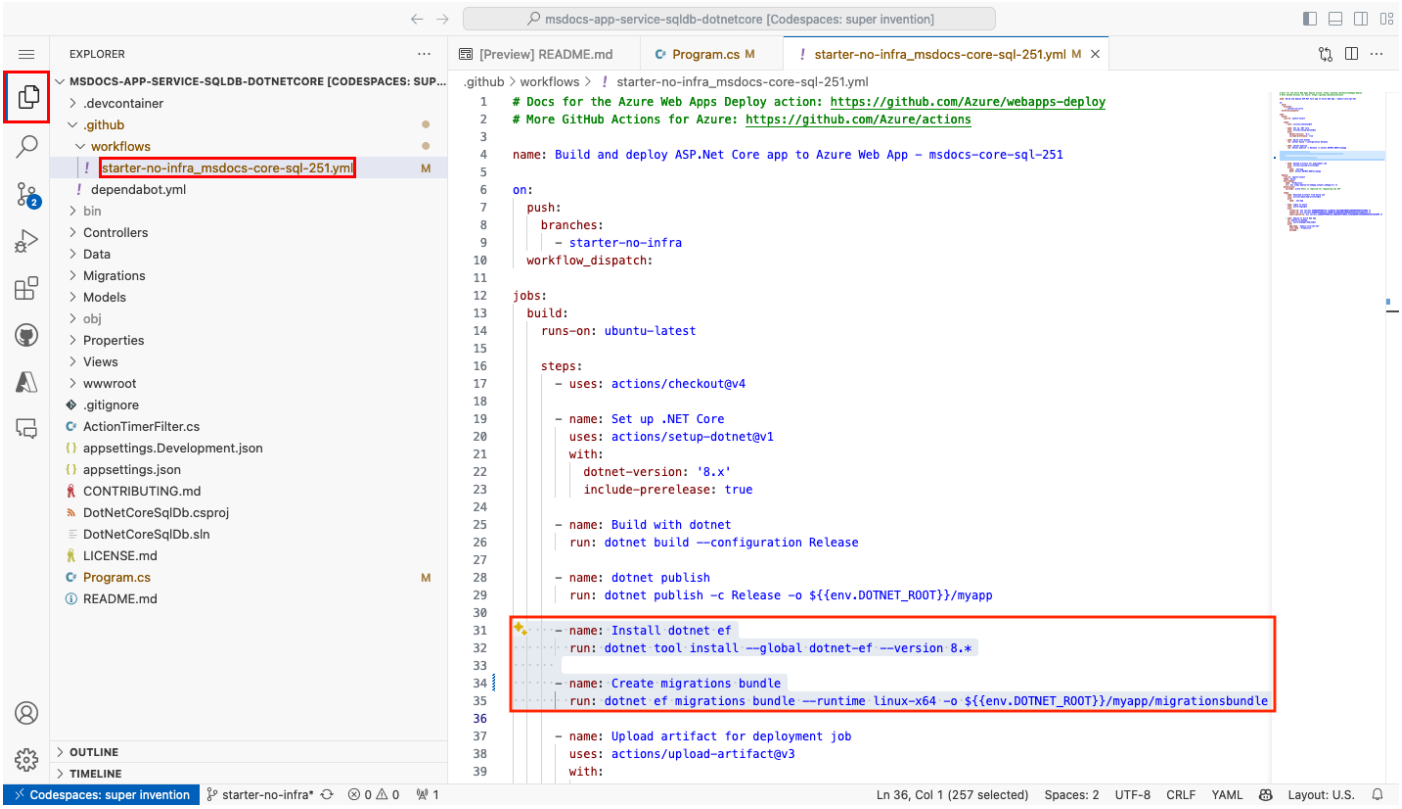
```
1   using Microsoft.EntityFrameworkCore;
2   using DotNetCoreSqlDb.Data;
3   var builder = WebApplication.CreateBuilder(args);
4
5   // Add database context and cache
6   if(builder.Environment.IsDevelopment())
7   {
8       builder.Services.AddDbContext<MyDatabaseContext>(options =>
9           options.UseSqlServer(builder.Configuration.GetConnectionString("MyDbConnection")));
10      builder.Services.AddDistributedMemoryCache();
11  }
12  else
13  {
14      builder.Services.AddDbContext<MyDatabaseContext>(options =>
15          options.UseSqlServer(builder.Configuration.GetConnectionString("AZURE_SQL_CONNECTIONSTRING")));
16      builder.Services.AddStackExchangeRedisCache(options =>
17      {
18      options.Configuration = builder.Configuration["AZURE_REDIS_CONNECTIONSTRING"];
19      options.InstanceName = "SampleInstance";
20      });
21  }
22
23  // Add services to the container.
24  builder.Services.AddControllersWithViews();
25
26  // Add App Service logging
27  builder.Logging.AddAzureWebAppDiagnostics();
28
29  var app = builder.Build();
30
31  // Configure the HTTP request pipeline.
32  if (!app.Environment.IsDevelopment())
```

STEP 3:

1. Open *.github/workflows/starter-no-infra_msdocs-core-sql-XYZ* in the explorer. This file was created by the App Service create wizard.

2. Under the dotnet publish step, add a step to install the Entity Framework Core tool with the command dotnet tool install -g dotnet-ef --version 8.*.

3. Under the new step, add another step to generate a database migration bundle in the deployment package: dotnet ef migrations bundle --runtime linux-x64 -o ${{env.DOTNET_ROOT}}/myapp/migrationsbundle. The migration bundle is a self-contained executable that you can run in the production environment without needing the .NET SDK. The App Service linux container only has the .NET runtime and not the .NET SDK.



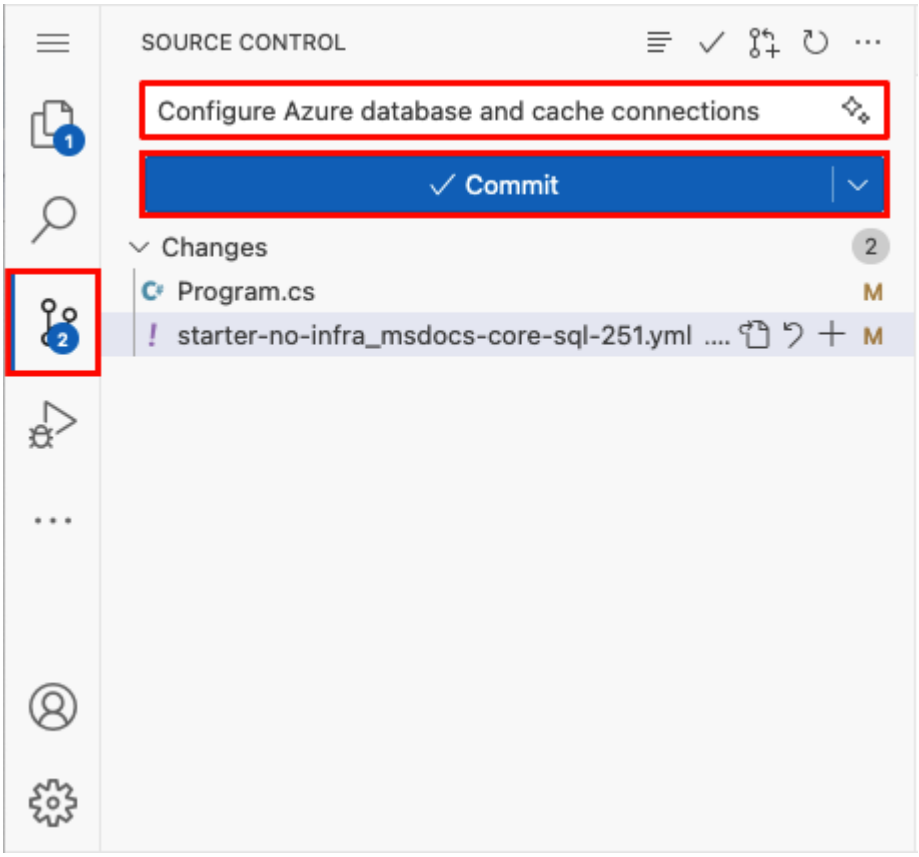
STEP 4:

1. Select the **Source Control** extension.

2. In the textbox, type a commit message like Configure Azure database and cache connections. Or, select and let GitHub Copilot generate a commit message for you.

3. Select **Commit**, then confirm with **Yes**.

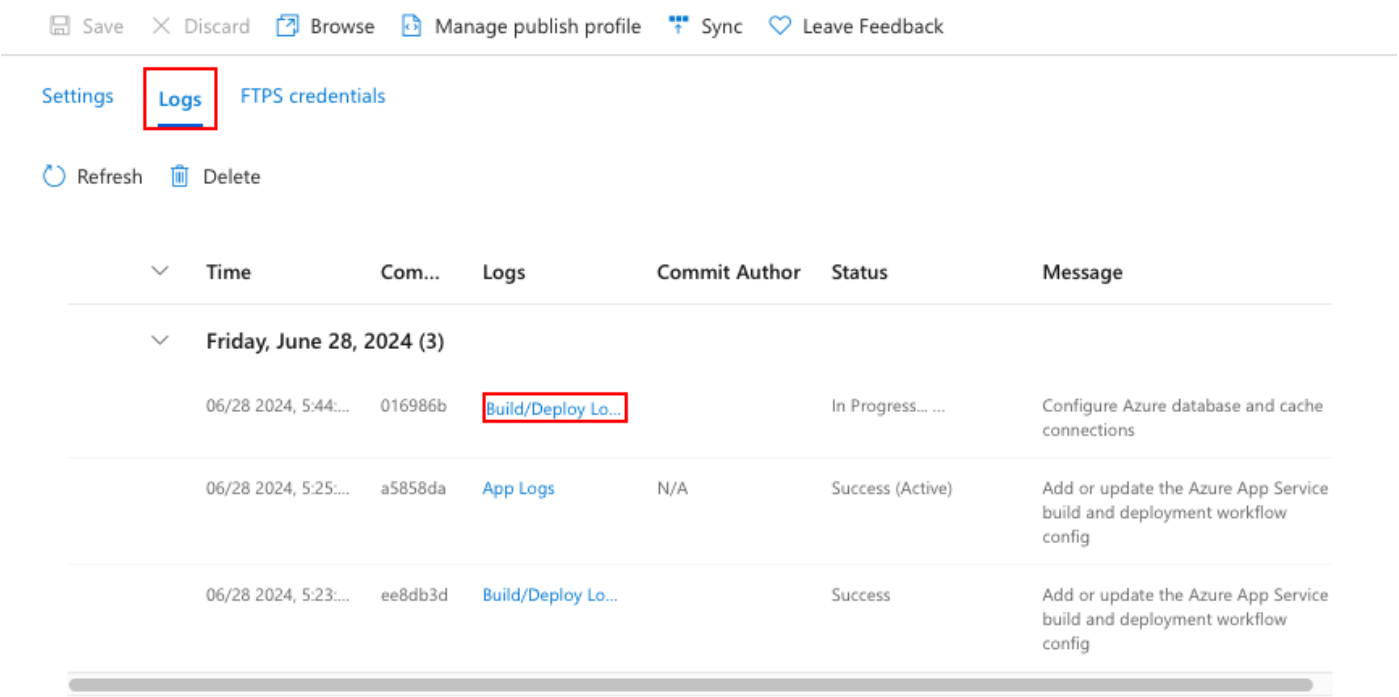
4. Select **Sync changes 1**, then confirm with **OK**.



STEP 5:

Back in the Deployment Center page in the Azure portal:

1. Select the **Logs** tab, then select **Refresh** to see the new deployment run.
2. In the log item for the deployment run, select the **Build/Deploy Logs** entry with the latest timestamp.



STEP 6:

You're taken to your GitHub repository and see that the GitHub action is running. The workflow file defines two separate stages, build and deploy. Wait for the GitHub run to show a status of **Success**. It takes about 5 minutes.