Microsoft

Azure Master Architect Program: Building Intelligent API Ecosystems

Participant guide

February 2026

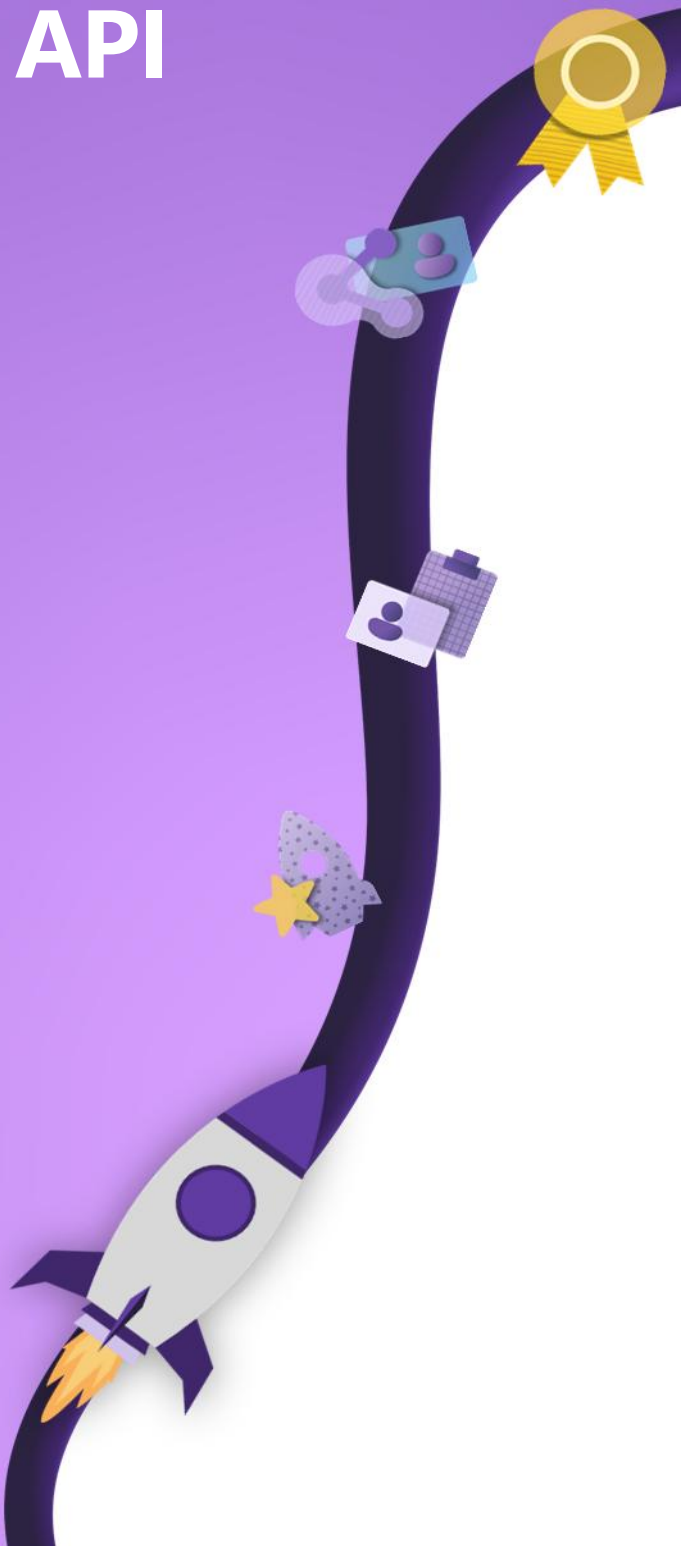


Table of contents

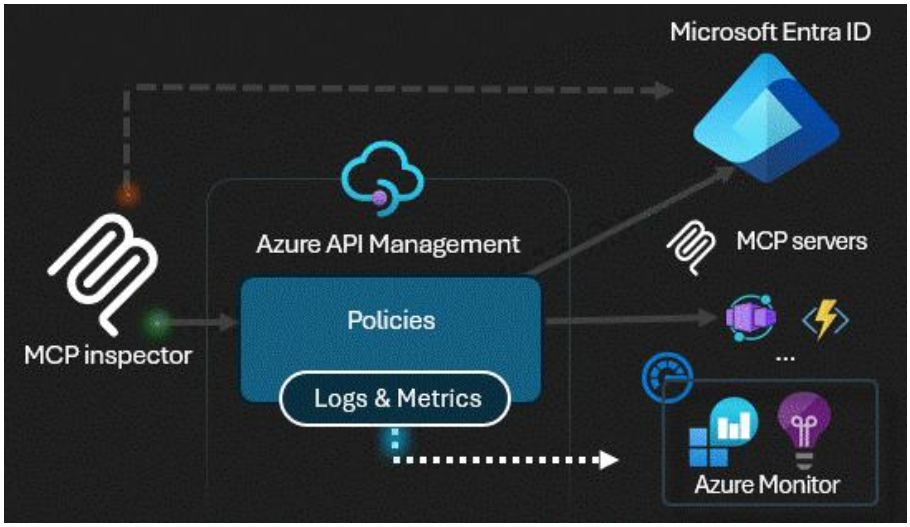
Getting ready

To complete the activities in this workshop, you need:

- An Azure subscription with sufficient capacity to perform workshop activities.
- A GitHub account linked to your Microsoft account, with GitHub Copilot enabled.
- Visual Studio Code, with the Jupyter Notebook extension installed.
- The Azure command line interface (CLI).
    - Run the installer at https://aka.ms/installazurecliwindows.
- The Azure Developer CLI.
    - In PowerShell, run the command
      `winget install Microsoft.azd`

Activity 1: Model Context Protocol tools with APIM

In this activity you will build a remote Model Context Protocol (MCP) server using Azure API Management (APIM) as your authentication gateway. The APIM gateway enforces policies for secure calls to MCP servers, and you can use an MCP inspector to examine the behavior and use Azure Monitor to capture log and metric information.

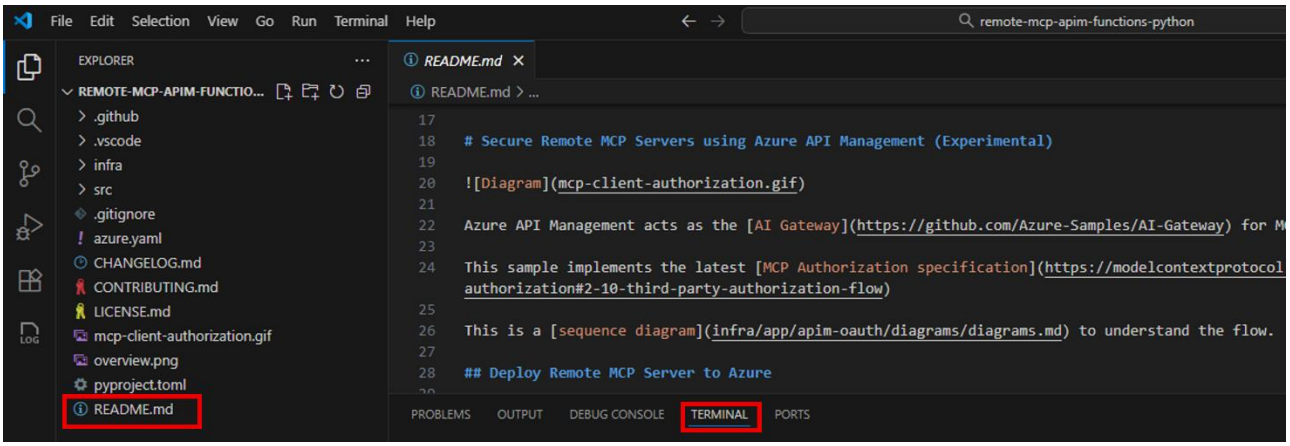


Build and test an MCP server

Begin by cloning a GitHub repository and then run terminal commands to build the architecture.

1. Open the https://aka.ms/mcp-remote-apim-auth GitHub repository.

2. Select **Code** (a green button on the GitHub repository). On the **Local** tab, select **HTTPS** and then select the **Copy** icon next to the URL field.
3. Open Git Bash or a PowerShell terminal and enter the following command:
   `git clone https://github.com/Azure-Samples/remote-mcp-apim-functions-python.git`
   - Alternatively, you can use GitHub Desktop, or use VS Code to clone the repository.
4. Open the cloned repository folder in VS Code and then open a terminal pane. From this point, you'll be following directions in the `README.md` file.

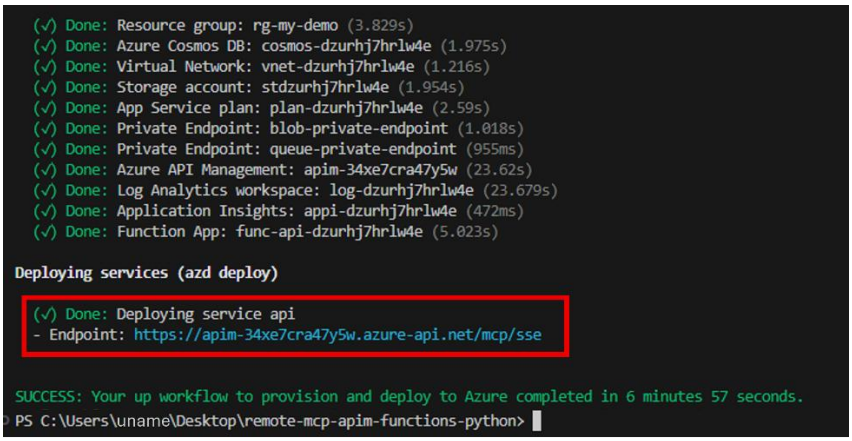


5. To provision the API management (APIM) service and other resources, run the command **`azd up`**
   - If the command **`azd`** is not recognized, then you need to install the Azure Developer CLI and restart the shell window. Refer to Getting ready in this document.
   - You will need to provide your Azure subscription key.
   - You will need to select an Azure region. The `East US` region, for example, may fail to provision for this exercise due to Cosmos DB constraints. Try using `Sweden central` or other regions that may have less demand.
   - The infrastructure scripts create an Azure resource group. In the Azure portal, you can examine the new `rg-mcp-demo-dev` resource group to discover what was deployed.
   - The script may fail due to restrictions on MCAPS subscriptions via policies, with the following error:

     ```
     DeploymentScriptOperationFailed: Key based authentication is not permitted on this storage account.
     ```

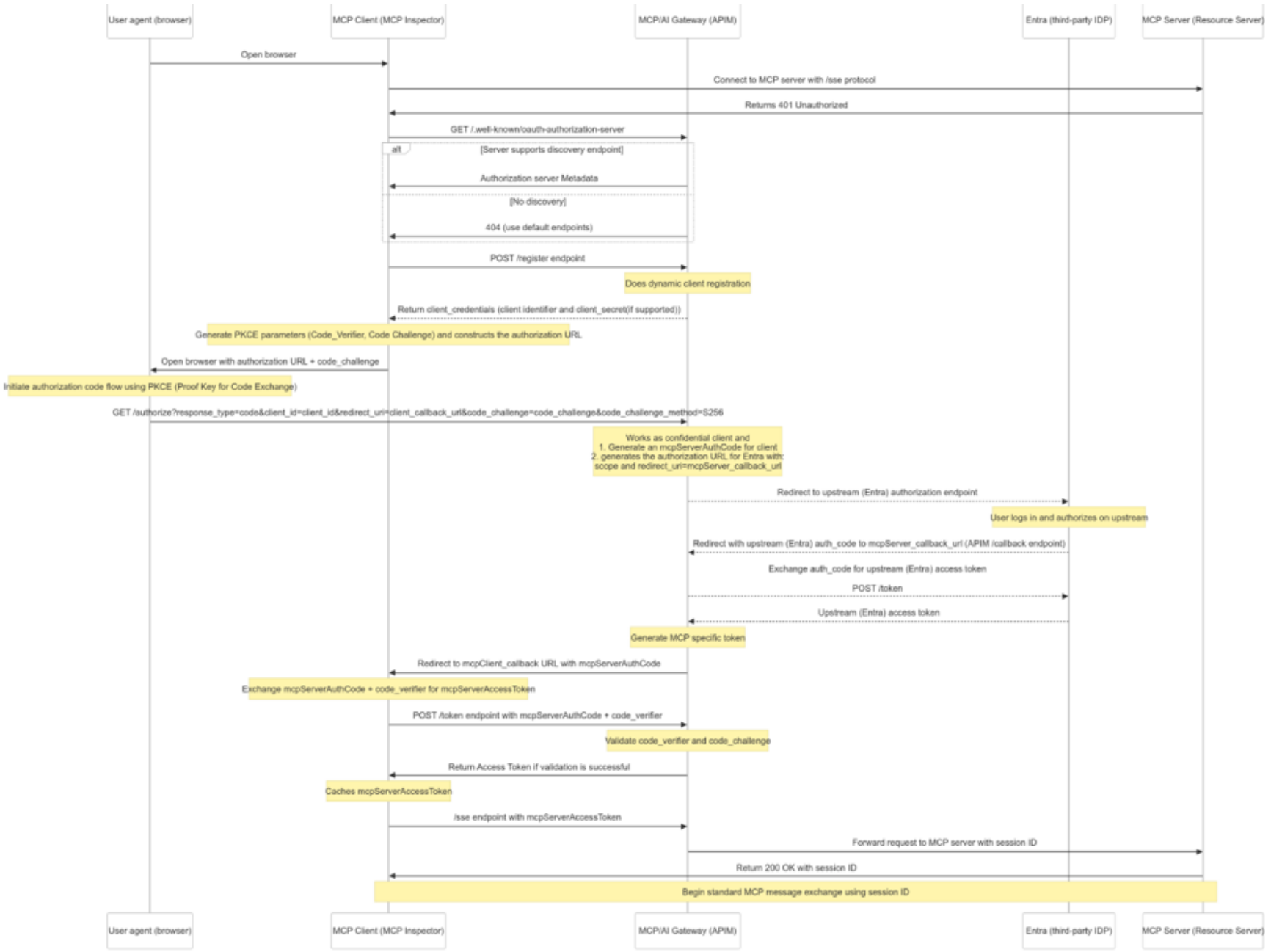
     Ask the instructor for assistance in creating a policy exception in the Azure resource group.

   When the **azd up** command is complete, you should see a list of deployed resources and an API endpoint, such as this example (your endpoint URL will differ):



6. Once the previous **azd** deployment step finishes successfully, continue following the directions in the `README.md` file.
7. Install and run an MCP Inspector web app to test the APIM gateway, with the following command in the terminal pane:
   `npx @modelcontextprotocol/inspector`

   When the command returns a URL for the MCP Inspector app, `CTRL`+left-click the URL to launch the app.
   - You'll be prompted to log in with your Azure subscription identity.
   - When the MCP Inspector app asks for details, such as the server send events (SSE) endpoint, find the configuration details in the terminal pane in the response to the earlier **`azd up`** command. Select **Connect** to run testing. There can be a significant delay (10 seconds or more) the first time you do this, so please be patient.
   - Read the rest of the `README.md` file to learn about APIM features and functions.
   - The sequence diagram on the next page specifies the flow between the MCP client, APIM, and the MCP server. (This diagram is also available from a link in the `README.md` file.)


User agent (browser)
MCP Client (MCP Inspector)
MCP/AI Gateway (APIM)
Entra (third-party IDP)
MCP Server (Resource Server)
Open browser
Connect to MCP server with /sse protocol
Returns 401 Unauthorized
GET /.well-known/oauth-authorization-server
alt
[Server supports discovery endpoint]
Authorization server Metadata
[No discovery]
404 (use default endpoints)
POST /register endpoint
Does dynamic client registration
Return client_credentials (client identifier and client_secret(if supported))
Generate PKCE parameters (Code_Verifier, Code Challenge) and constructs the authorization URL
Open browser with authorization URL + code_challenge
Initiate authorization code flow using PKCE (Proof Key for Code Exchange)
GET /authorize?response_type=code&client_id=client_id&redirect_uri=client_callback_url&code_challenge=code_challenge&code_challenge_method=S256
Works as confidential client and
1. Generate an mcpServerAuthCode for client
2. generates the authorization URL for Entra with
scope and redirect_uri=mcpServer_callback_url
Redirect to upstream (Entra) authorization endpoint
User logs in and authorizes on upstream
Redirect with upstream (Entra) auth_code to mcpServer_callback_url (APIM /callback endpoint)
Exchange auth_code for upstream (Entra) access token
POST /token
Upstream (Entra) access token
Generate MCP specific token
Redirect to mcpClient_callback URL with mcpServerAuthCode
Exchange mcpServerAuthCode + code_verifier for mcpServerAccessToken
POST /token endpoint with mcpServerAuthCode + code_verifier
Validate code_verifier and code_challenge
Return Access Token if validation is successful
Caches mcpServerAccessToken
/sse endpoint with mcpServerAccessToken
Forward request to MCP server with session ID
Return 200 OK with session ID
Begin standard MCP message exchange using session ID
User agent (browser)
MCP Client (MCP Inspector)
MCP/AI Gateway (APIM)
Entra (third-party IDP)
MCP Server (Resource Server)

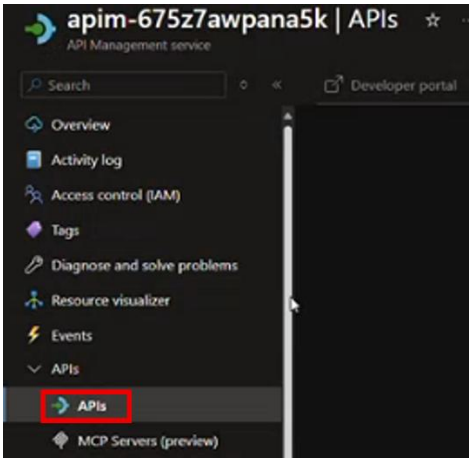
Activity 2: Exposing existing APIs as MCP Servers with APIM

In this activity you will practice converting an existing API to an MCP Server.

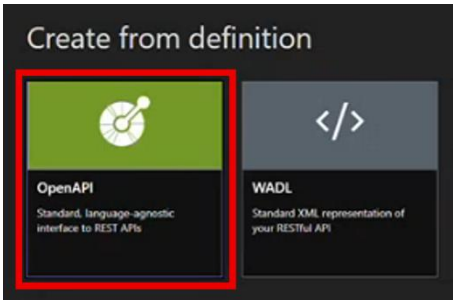
Create APIs in Azure Portal

Begin by importing a set of APIs in a JSON file into the Azure APIM instance you created earlier.

1. Download the file `sample.json` from https://aka.ms/AzureMasterArchitect_Resources.
2. In the Azure portal, navigate to the API Management instance you created in the last activity.
3. In the sidebar menu, under **APIs**, select **APIs**.

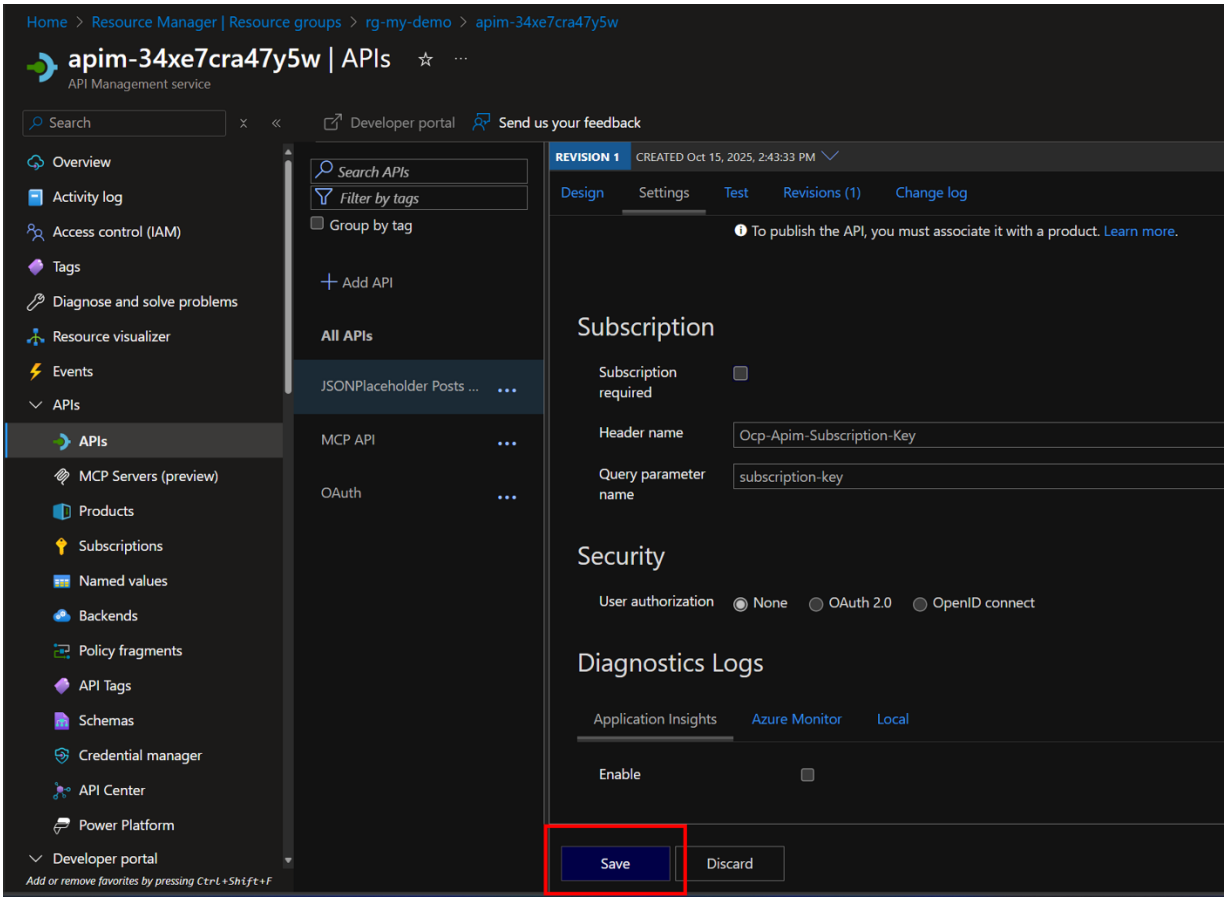


4. In the **Create from definition** area, select **OpenAI**.



5. In the **Create from OpenAI specification** dialog box, select the JSON file that you downloaded.
    - Enter a **Display name** (can contain spaces and special characters) and **Name** (used in code references) for the API.
    - In the **API URL suffix** field, enter a path segment that will be appended to your APIM base URL to form the full endpoint for your API. It must be unique within the APIM instance. For example, if your APIM base URL is `https://contoso.azure-api.net`, and your API URL suffix is `sample`, then your API will be accessible at `https://mycompany.azure-api.net/openai-chat`.
    - Select **Create**.

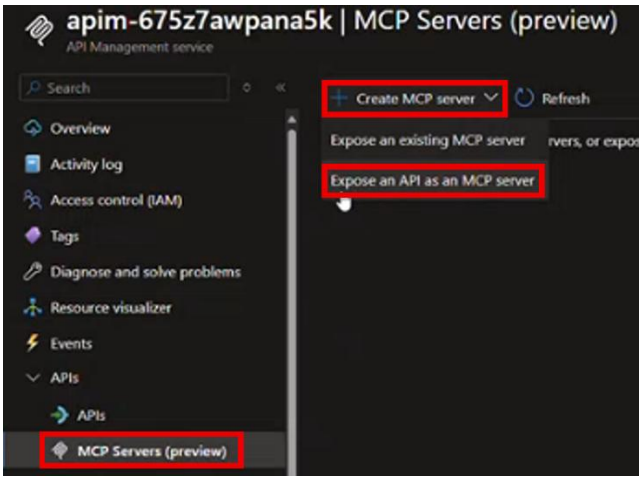
6. Under the **All APIs** heading, select the new RESTful API to examine its structure. You can test the functions here in the Azure Portal interface.
7. To avoid an **Access denied** error when testing this simple example, disable the requirement to provide a subscription key when calling the API.
    - In the API view, on the **Settings** pane, clear the selection of the **Subscription required** checkbox.
    - Select **Save** to save this configuration change.



Expose APIs as MCP Servers

Use Azure Portal to expose the APIs as MCP Servers in APIM.

1. In the Azure portal, navigate to your API Management instance, the one you created in the last activity. (You're already there if you're continuing from the last exercise.)
2. In the sidebar menu, under **APIs**, select **MCP Servers** and then select **+ Create MCP server**.
3. Select **Expose an API as an MCP server**.



4. In the **Backend MCP server** area:
    a. Select the **API** that you just created.
    b. Select one or more **API operations** to expose as tools. You can select all operations or only specific operations.
5. In the **New MCP server** pane:
    a. Enter a **Name** for the MCP server in API Management.
    b. Optionally, enter a **Description** for the MCP server.
6. Select **Create**.
7. To test the new MCP Server with the MCP Inspector:
    a. Select the new MCP Server in the **MCP Server** list in Azure Portal.
    b. Copy the URL from the **MCP Server URL** field.
    c. Run the MCP Inspector in the **Terminal** pane of VS Code.
    d. In the MCP Inspector, change the **Transport type** from SSE to streamable HTTP. Paste the URL and select **Connect** to access the new MCP Server and test it.
8. When you are done with this activity, or when you no longer want to revisit this sample configuration, delete all the resources you created. In the terminal pane of VS Code, enter the command

    ```
    azd down
    ```

    You do not need these resources for the next activity.

Activity 3: MCP Server Registry and API Center

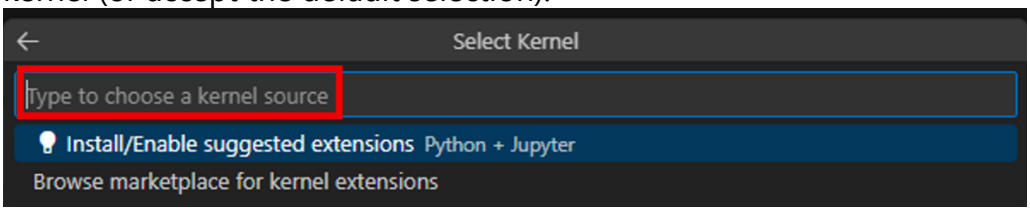
In this activity you will create a resource group, configure several MCP Servers, and then explore the capabilities of MCP Server Registry and Azure API Center.

Explore MCP Server Registry and API Center

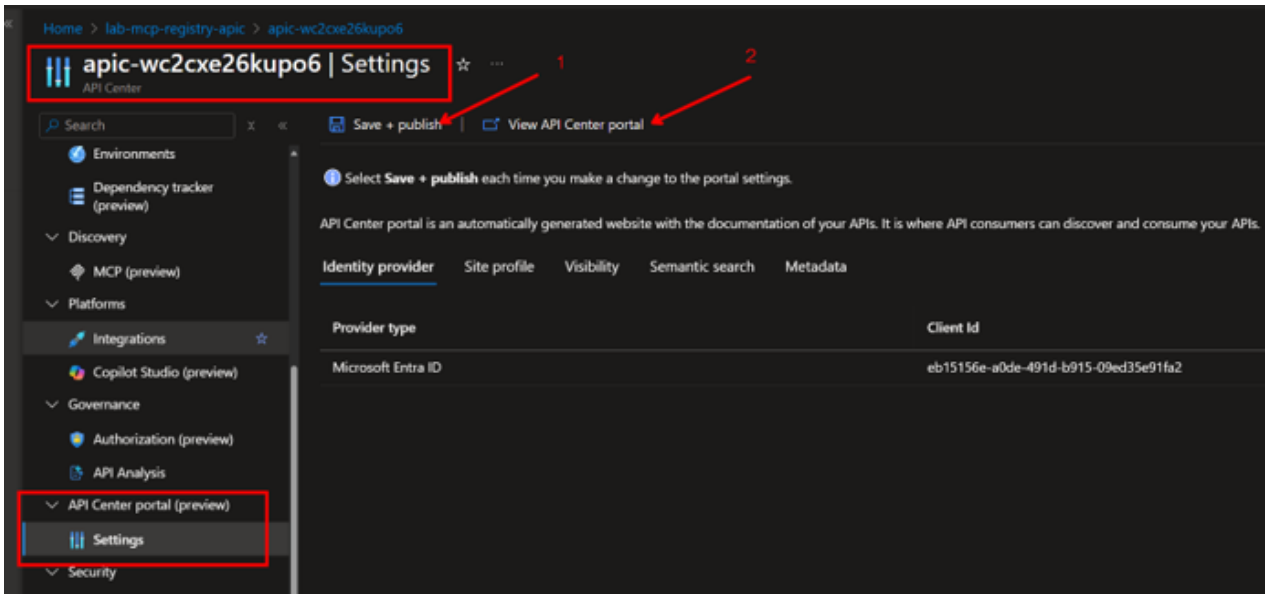
Begin by cloning a GitHub repository and then run an interactive Python notebook (.ipynb).

1. Open the https://github.com/Azure-Samples/AI-Gateway/tree/main GitHub repository.
2. Clone the repository and then open the cloned repository folder in VS Code.
3. In the sidebar navigation pane, select **`labs`**, then **`mcp_registry_apic`**, and then **`mcp_registry_apic.ipynb`** to open an interactive Python notebook.
4. The interactive Python notebook contains a series of *code cells*, a snippet of code that you can run interactively. At each code cell, hover over the code cell and then select **Execute cell** (the triangular Play button) to run the code (or tab to the code cell and press `SHIFT+ENTER`.)

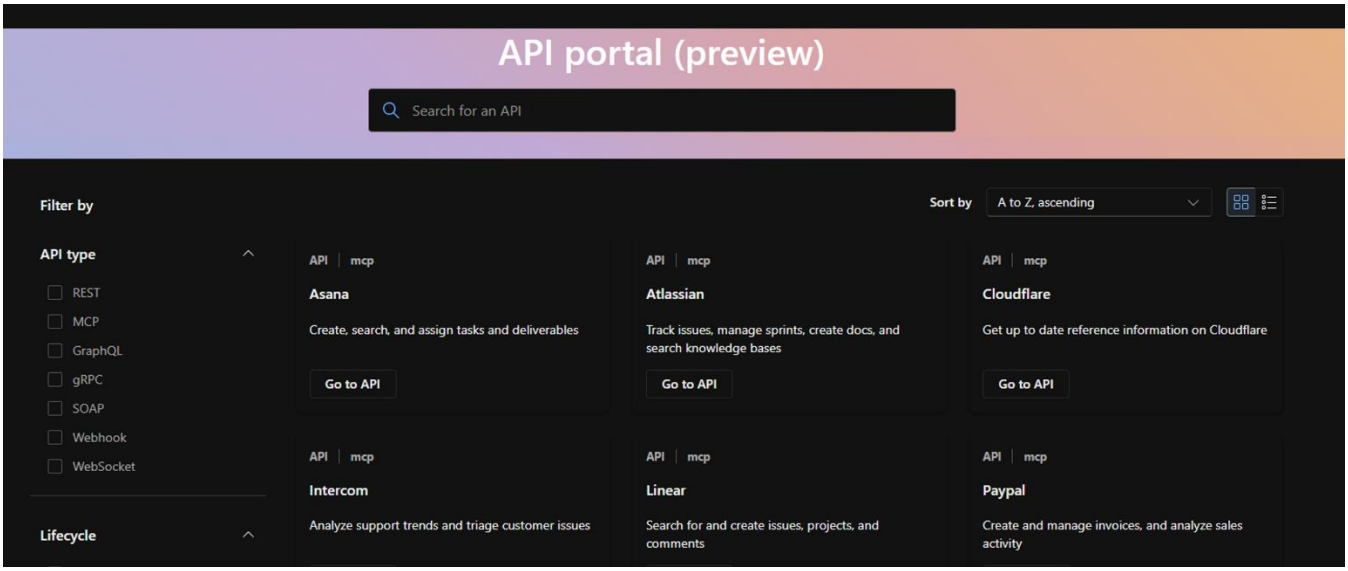
5. When you run the first code cell, the notebook prompts you for environmental variables. For example, at the top of the screen, in the **Select Kernel** pop-up window, select the appropriate kernel (or accept the default selection).



6. Continue through the interactive notebook, following instructions, running the code cells, and observing the effects.
7. When the interactive notebook creates the Azure API Center, go to the Azure portal to publish and explore the API Center.
    - In the sidebar, select **API Center portal** and then select **Settings**.
    - Select **Save + publish** to publish the API Center as provisioned.



8. Select **View API Center portal** to explore APIs and to test the published MCP Server.



9. After you complete the notebook, remove the deployed resources from Azure by running the **`clean-up-resources.ipynb`** notebook in the same folder of the repository.