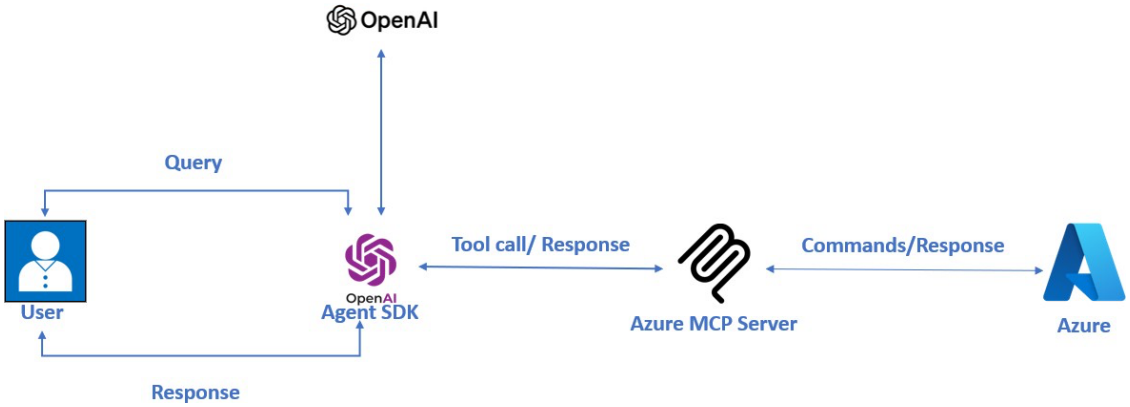
# Azure MCP Server Agent

## Overview




The application provides an interactive interface where users can:

1. Connect to Azure services using natural language commands
2. List Azure resources (Storage, Cosmos DB, etc.)
3. Execute Azure CLI commands
4. Query and explore Azure resources without remembering syntax

## Prerequisites

Before you begin, ensure you have the following installed:

- Python 3.10 or higher
- Node.js and npx (for the MCP Azure server)
- Azure subscription with appropriate permissions
- Azure CLI
- OpenAI API key

# Project Setup

## Installing uv Package Manager

If you don't have uv installed yet:

```
pip install uv
```

## Creating a Project Environment

Create a new project environment using uv:

```
uv init azure-mcp-assistant
cd azure-mcp-assistant
```

## Installing Required Packages

Install all required packages using uv:

```
uv add python-dotenv openai-agents==0.0.12 "mcp[cli]>=1.6.0"
```

## Environment Configuration

Create a .env file in your project directory with the following configuration:

```
OPENAI_API_KEY=your_openai_api_key
```

## Running the Application

1. Place the app.py file in your project directory.
2. Ensure you're logged in to Azure CLI:

```
az login
```

3. Run the application:

```
python app.py
```

4. Enter your queries when prompted

## Example Queries

- "List my Azure storage accounts"
- "Show me all my Cosmos DB databases"

- "List my resource groups"
- "Show me the tables in my Storage account"
- "Query my Log Analytics workspace"
- "List my App Configuration stores"

## Understanding the Code

Let's break down the main components of the app.py script:

### 1. Imports and Configuration

```
import os
import asyncio
from agents import Agent, Runner
from agents.mcp import MCPServerStdio
```

This section imports necessary libraries and loads environment variables from your .env file.

### 2. MCP Azure Server

```
async with MCPServerStdio(
    name="Azure MCP Server",
    params={"command": "npx", "args": ["-y", "@azure/mcp@latest", "server", "start"]},
) as server:
```

This configures the Model Context Protocol (MCP) server for Azure access. The server allows the application to interact with your Azure resources.

### 3. Agent Setup

```
agent = Agent(
    name="Assistant",
    instructions="You are a helpful assistant. Use the available tools to answer the user's questions about Azure resources and services.",
    mcp_servers=[mcp_server]
)
```

This creates an agent with access to Azure MCP tools and the specified Azure OpenAI model.

### 4. Main Application Loop

The run() function establishes connections and handles user interaction:

```
async def run(mcp_server: MCPServerStdio):
    # Implementation details
```

This function:

- Sets up an agent with Azure MCP tools
- Manages the chat loop, handling user input and displaying responses
- Maintains conversation history

# Technical Details

## Authentication Methods

The Azure MCP Server uses Azure Identity's DefaultAzureCredential, which tries these credentials in order:

1. Environment Variables
2. Shared Token Cache
3. Visual Studio Credentials
4. Azure CLI (Recommended)
5. Azure PowerShell
6. Azure Developer CLI
7. Interactive Browser

## MCP Azure Tools

The application provides access to these Azure services:

1. **Azure Resource Management**
    - List and manage resource groups
    - Manage individual resources
2. **Azure Storage**
    - List storage accounts
    - Manage blob containers and blobs
    - Work with storage tables
3. **Azure Cosmos DB**
    - List accounts and databases
    - Execute SQL queries against containers
4. **Azure Monitor**
    - Query logs using KQL

- List available tables and workspaces

5. **Azure App Configuration**
    - Manage key-value pairs
    - Handle labeled configurations
6. **Azure CLI Integration**
    - Execute any Azure CLI command directly