Get started with AI agent development on Azure

AI agents represent the next generation of intelligent applications. Learn how they can be developed and used on Microsoft Azure.

Learning objectives

By the end of this module, you'll be able to:

- Describe core concepts related to AI agents
- Describe options for agent development
- Create and test an agent in the Azure AI Foundry portal

Introduction

As generative AI models become more powerful and ubiquitous, their use has grown beyond simple "chat" applications to power intelligent agents that can operate autonomously to automate tasks. Increasingly, organizations are **using generative AI models to build agents** that orchestrate business processes and coordinate workloads in ways that were previously unimaginable.

This module discusses some of the **core concepts related to AI agents**, and introduces some of the technologies that developers can use to build agentic solutions on Microsoft Azure.

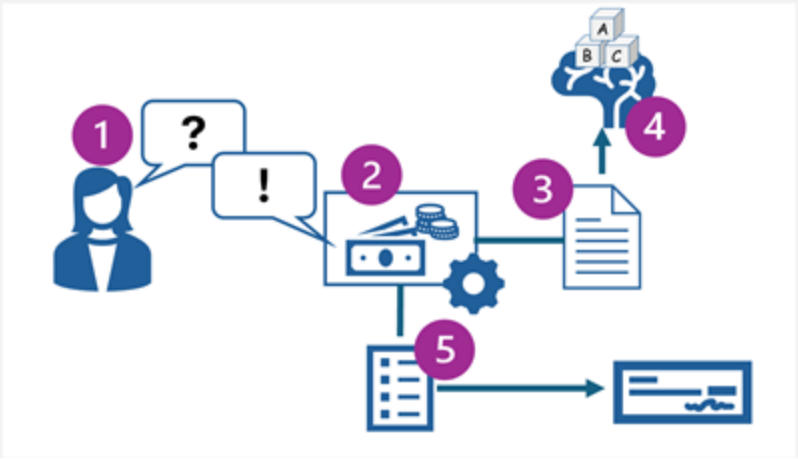
What are AI agents?

AI agents are **smart software services that combine generative AI models with contextual data and the ability to automate tasks based on user input and environmental factors that they perceive.**

For example, an organization might build an AI agent to help employees manage expense claims. The agent might use a generative model combined with corporate expenses policy documentation to

answer employee questions about what expenses can be claimed and what limits apply. Additionally, the agent could use a programmatic function to automatically submit expense claims for regularly repeated expenses (such as a monthly cellphone bill) or intelligently route expenses to the appropriate approver based on claim amounts.

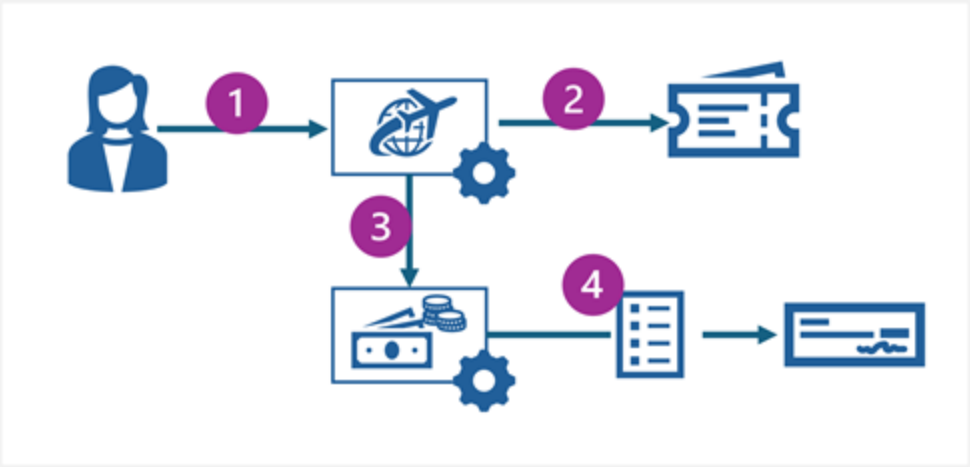
An example of the expenses agent scenario is shown in the following diagram.



The diagram shows the following process:

1. A user asks the expense agent a question about expenses that can be claimed.
2. The expenses agent accepts the question as a prompt.
3. The agent uses a knowledge store containing expenses policy information to ground the prompt.
4. The grounded prompt is submitted to the agent's language model to generate a response.
5. The agent generates an expense claim on behalf of the user and submits it to be processed and generate a check payment.

In more complex scenarios, organizations can develop **multi-agent solutions** in which multiple agents coordinate work between them. For example, *a travel booking agent could book flights and hotels for employees* and *automatically submit expense claims with appropriate receipts to the expenses agent*, as shown in this diagram:



The diagram shows the following process:

1. A user provides details of an upcoming trip to a travel booking agent.
2. The travel booking agent automates the booking of flight tickets and hotel reservations.
3. The travel booking agent initiates an expense claim for the travel costs through the expense agent.
4. The expense agent submits the expense claim for processing.

Options for agent development

There are many ways that developers can create AI agents, including **multiple frameworks and SDKs**.

Azure AI Foundry Agent Service

Azure AI Foundry Agent Service is a managed service in Azure that is designed to provide a framework for **creating, managing, and using AI agents within Azure AI Foundry**. The service is based on the OpenAI Assistants API but with increased choice of models, data integration, and enterprise security; enabling you to use both the **OpenAI SDK** and the **Azure Foundry SDK** to **develop agentic solutions**.

> Tip: For more information about Foundry Agent Service, see the Azure AI Foundry Agent Service documentation.

OpenAI Assistants API

The OpenAI Assistants API provides a subset of the features in Foundry Agent Service, and can **only be used with OpenAI models**. In Azure, you can use the Assistants API with Azure OpenAI, though in practice the Foundry Agent Service provides greater flexibility and functionality for agent development on Azure.

Semantic Kernel

Semantic Kernel is a **lightweight, open-source development kit** that you can use to build AI agents and orchestrate multi-agent solutions. The **core Semantic Kernel SDK** is designed for all kinds of generative AI development, while the **Semantic Kernel Agent Framework** is a platform specifically optimized for creating agents and implementing agentic solution patterns.

AutoGen

AutoGen is an open-source framework for developing agents rapidly. It's **useful as a research and ideation tool** when experimenting with agents.

Microsoft 365 Agents SDK

Developers can create self-hosted agents for delivery through a wide range of channels by using the **Microsoft 365 Agents SDK**. Despite the name, agents built using this SDK are not limited to Microsoft 365, but can be delivered through channels like **Slack** or **Messenger**.

Microsoft Copilot Studio

Microsoft Copilot Studio provides a **low-code development environment** that "citizen developers" can use to quickly build and deploy agents that integrate with a Microsoft 365 ecosystem or commonly used channels like Slack and Messenger. The visual design interface of Copilot Studio makes it a good choice for building agents when you have little or no professional software development experience.

Copilot Studio agent builder in Microsoft 365 Copilot

Business users can use the **declarative Copilot Studio agent builder tool** in Microsoft 365 Copilot to author basic agents for common tasks. The declarative nature of the tool enables users to create an agent by describing the functionality they need, or they can use an intuitive visual interface to specify options for their agent.

Choosing an agent development solution

With such a wide range of available tools and frameworks, it can be challenging to decide which ones to use. Use the following **considerations** to help you identify the right choices for your scenario:

- For business users with **little or no software development experience**, **Copilot Studio agent builder in Microsoft 365 Copilot Chat** provides a way to create simple declarative agents that automate everyday tasks. This approach can empower users across an organization to benefit from AI agents with minimal impact on IT.
- If business users have sufficient technical skills to build **low-code solutions using Microsoft Power Platform technologies**, Copilot Studio enables them to combine those skills with their business domain knowledge and build agent solutions that extend the capabilities of Microsoft 365 Copilot or add agentic functionality to common channels like Microsoft Teams, Slack, or Messenger.

- When an organization needs more complex extensions to Microsoft 365 Copilot capabilities, **professional developers** can use the **Microsoft 365 Agents SDK** to build agents that target the same channels as Copilot Studio.
- To develop agentic solutions that use Azure back-end services with a wide choice of models, custom storage and search services, and integration with Azure AI services, professional developers should use **Foundry Agent Service**.
- Start with **Foundry Agent Service to develop single, standalone agents**. When you need to build **multi-agent solutions**, use **Semantic Kernel to orchestrate the agents** in your solution.

User	Scenario	Tool
Little to no software dev experience	simple declarative agents that automate everyday work	Copilot Studio agent builder in Microsoft 365 Copilot Chat
Users with sufficient tech skills	Build low-code solutions using Microsoft Power Platform technologies	Microsoft 365 Copilot
Professional developers	Build agents that target the same channels as Copilot Studio	Microsoft 365 Agents SDK
Professional developers	Develop agentic solutions that use Azure back-end services with a wide choice of models, custom storage and search services	Foundry Agent Service
Professional developers	Develop single, standalone agents	Foundry Agent Service
Professional developers	multi-agent solutions	Semantic Kernel

> Note: There's overlap between the capabilities of each agent development solution, and in some cases factors like existing familiarity with tools, programming language preferences, and other considerations will influence the decision.

Azure AI Foundry Agent Service

Azure AI Foundry Agent Service is a service within Azure that you can use to **create, test, and manage AI agents**. It provides both a visual agent development experience in the Azure AI Foundry

portal and **a code-first development experience using the Azure AI Foundry SDK**.

Components of an agent

Agents developed using Foundry Agent Service have the following elements:

- **Model**: A deployed generative AI model that enables the agent to reason and generate natural language responses to prompts. You can use common OpenAI models and a selection of models from the **Azure AI Foundry model catalog**.
- **Knowledge**: data sources that enable the agent to ground prompts with contextual data. Potential knowledge sources include *Internet search results from Microsoft Bing, an Azure AI Search index, or your own data and documents*.
- **Tools**: Programmatic functions that enable the agent to automate actions. Built-in tools to access knowledge in Azure AI Search and Bing are provided as well as a code interpreter tool that you can use to generate and run Python code. You can also create custom tools using your own code or Azure Functions.

Conversations between users and agents take place on a thread, which retains a history of the messages exchanged in the conversation as well as any data assets, such as files, that are generated.

Exercise - Explore AI Agent development

In this exercise, you use the Azure AI Agent service in the Azure AI Foundry portal to create a simple AI agent that assists employees with expense claims.

Explore AI Agent development

In this exercise, you use the Azure AI Agent service in the Azure AI Foundry portal to create a simple AI agent that assists employees with expense claims.

Module assessment

1. Which of the following best describes an AI agent? **A software service that uses AI to assist users with information and task automation**.

2. Which AI agent development service offers a choice of generative AI models from multiple vendors in the Azure AI Foundry model catalog? **Azure AI Foundry Agent Service**.
3. What element of an Foundry Agent Service agent enables it to ground prompts with contextual data? **Knowledge**.

Summary

In this module, you learned about AI agents and some of the options available for developing them. You also learned how to create a simple agent using the visual tools for **Foundry Agent Service in the Azure AI Foundry portal**.

> Tip: For more information about Foundry Agent Service, see Azure AI Foundry Agent Service documentation.

Develop an AI agent with Azure AI Foundry Agent Service

This module provides engineers with the skills to begin **building agents with Azure AI Foundry Agent Service**.

Learning objectives

By the end of this module, you'll be able to:

- Describe the **purpose of AI agents**
- Explain the **key features of Azure AI Foundry Agent Service**
- **Build an agent** using the Foundry Agent Service
- **Integrate an agent** in the Foundry Agent Service into your own application

Introduction

Azure AI Foundry Agent Service is *a fully managed service* designed to empower developers to securely **build, deploy, and scale high-quality, extensible AI agents** without needing to manage the underlying compute and storage resources.

Imagine you're working in the healthcare industry, where there's a need to automate patient interactions and streamline administrative tasks. Your organization wants to develop an AI agent that can handle patient inquiries, schedule appointments, and provide medical information based on real-time data. However, managing the infrastructure and ensuring data security are significant challenges. Azure AI Foundry Agent Service offers a solution by allowing you to create AI agents tailored to your needs through custom instructions and advanced tools. This service **simplifies the development process, reduces the amount of code required, and manages the underlying infrastructure, enabling you to focus on building high-quality AI solutions**.

In this module, you'll learn how to use the Foundry Agent Service to develop agents.

What is an AI agent

An AI agent is **a software service that uses generative AI to understand and perform tasks on behalf of a user or another program**. These agents use advanced AI models to **understand context, make decisions, utilize grounding data, and take actions to achieve specific goals**. Unlike traditional applications, AI agents can operate independently, executing complex workflows and automating processes without the need of constant human intervention. The evolution of generative AI enables agents to behave intelligently on our behalf, transforming how we can use and integrate these agents.

Understanding what an AI agent is and how to utilize them is crucial for effectively using AI to automate tasks, make informed decisions, and enhance user experiences. This knowledge enables organizations to deploy AI agents strategically, maximizing their potential to drive innovation, improve efficiency, and achieve business objectives.

Why Are AI agents useful?

AI agents are incredibly useful for several reasons:

- **Automation of Routine Tasks**: AI agents can handle repetitive and mundane tasks, freeing up human workers to focus on more strategic and creative activities. This leads to increased productivity and efficiency.
- **Enhanced Decision-Making**: By processing vast amounts of data and providing insights, AI agents support better decision-making. They can **analyze trends, predict outcomes, and offer recommendations based on real-time data**. AI Agents can even use advanced decision-making algorithms and machine learning models to analyze data and make informed decisions autonomously. This allows them to handle complex scenarios and provide actionable insights, whereas generative AI chat models primarily focus on generating text-based responses.
- **Scalability**: AI agents can scale operations without the need for proportional increases in human resources. This is beneficial for businesses looking to grow without significantly increasing operational costs.
- **24/7 Availability**: Like all software, AI agents can operate continuously without breaks, ensuring that tasks are completed promptly and customer service is available around the clock.

Agents are built to simulate human-like intelligence and can be applied in various domains such as customer service, data analysis, automation, and more.

Examples of AI agent use cases

AI agents have a wide range of applications across various industries. Here are some notable examples:

Personal productivity agents

Personal productivity agents assist individuals with **daily tasks** such as **scheduling meetings, sending emails, and managing to-do lists**. For instance, Microsoft 365 Copilot can help users draft documents, create presentations, and analyze data within the Microsoft Office suite.

Research agents

Research agents continuously **monitor market trends, gather data, and generate reports**. These agents can be used in financial services to **track stock performance**, in healthcare to stay updated with the latest **medical research**, or in marketing to analyze consumer behavior.

Sales agents

Sales agents automate lead generation and qualification processes. They can **research potential leads, send personalized follow-up messages, and even schedule sales calls**. This automation helps sales teams focus on closing deals rather than administrative tasks.

Customer service agents

Customer service agents **handle routine inquiries, provide information, and resolve common issues**. They can be integrated into chatbots on websites or messaging platforms, offering instant support to customers. For example, Cineplex uses an AI agent to process refund requests, significantly reducing handling time and improving customer satisfaction.

Developer agents

Developer agents help in **software development** tasks such as **code review, bug fixing, and repository management**. They can automatically **update codebases, suggest improvements**, and ensure that **coding standards are maintained**. GitHub Copilot is a great example of a developer agent.

> Tip: To learn more about GitHub Copilot, explore the GitHub Copilot fundamentals learning path.

> Note: You can explore more about agents in general with the Fundamentals of AI agents module.

How to use Azure AI Foundry Agent Service

Azure AI Foundry Agent Service is **a fully managed service designed to empower developers to securely build, deploy, and scale high-quality, extensible AI agents without needing to manage the underlying compute and storage resources**. This unit covers the purpose, benefits, key features, and integration capabilities of Azure AI Foundry Agent Service.

Purpose of Azure AI Foundry Agent Service

The Foundry Agent Service allows developers to **create AI agents tailored to their needs through custom instructions and advanced tools like code interpreters and custom functions**. These agents can answer questions, perform actions, or automate workflows by combining generative AI models with tools that interact with real-world data sources. The service simplifies the development process by reducing the amount of code required and managing the underlying infrastructure.

Previously, developers could create an agent-like experience by using standard APIs in Azure AI Foundry and connect to custom functions or other tools, but doing so would take a significant coding effort. Foundry Agent Service handles all of that for you through AI Foundry to build agents via the portal or in your own app in fewer than 50 lines of code. The exercise in the module explores both methods of building an agent.

Foundry Agent Service is ideal for **scenarios requiring advanced language models for workflow automation**. It can be used to:

- Answer questions using real-time or proprietary data sources.
- Make decisions and perform actions based on user inputs.
- Automate complex workflows by combining generative AI models with tools that interact with real-world data.

For example, an AI agent can be created to generate reports, analyze data, or even interact with users through apps or chatbots, making it suitable for customer support, data analysis, and automated reporting.

Key features of Foundry Agent Service

Foundry Agent Service offers several key features:

- **Automatic tool calling**: The service handles the entire tool-calling lifecycle, including running the model, invoking tools, and returning results.

- **Securely managed data**: Conversation states are securely managed using threads, eliminating the need for developers to handle this manually.
- **Out-of-the-box tools**: The service includes tools for file retrieval, code interpretation, and interaction with data sources like Bing, Azure AI Search, and Azure Functions.
- **Flexible model selection**: Developers can choose from various models, including Azure OpenAI models and others like Llama 3, Mistral, and Cohere.
- **Enterprise-grade security**: The service ensures data privacy and compliance with secure data handling and keyless authentication.
- **Customizable storage solutions**: Developers can use either platform-managed storage or bring their own Azure Blob storage for full visibility and control.

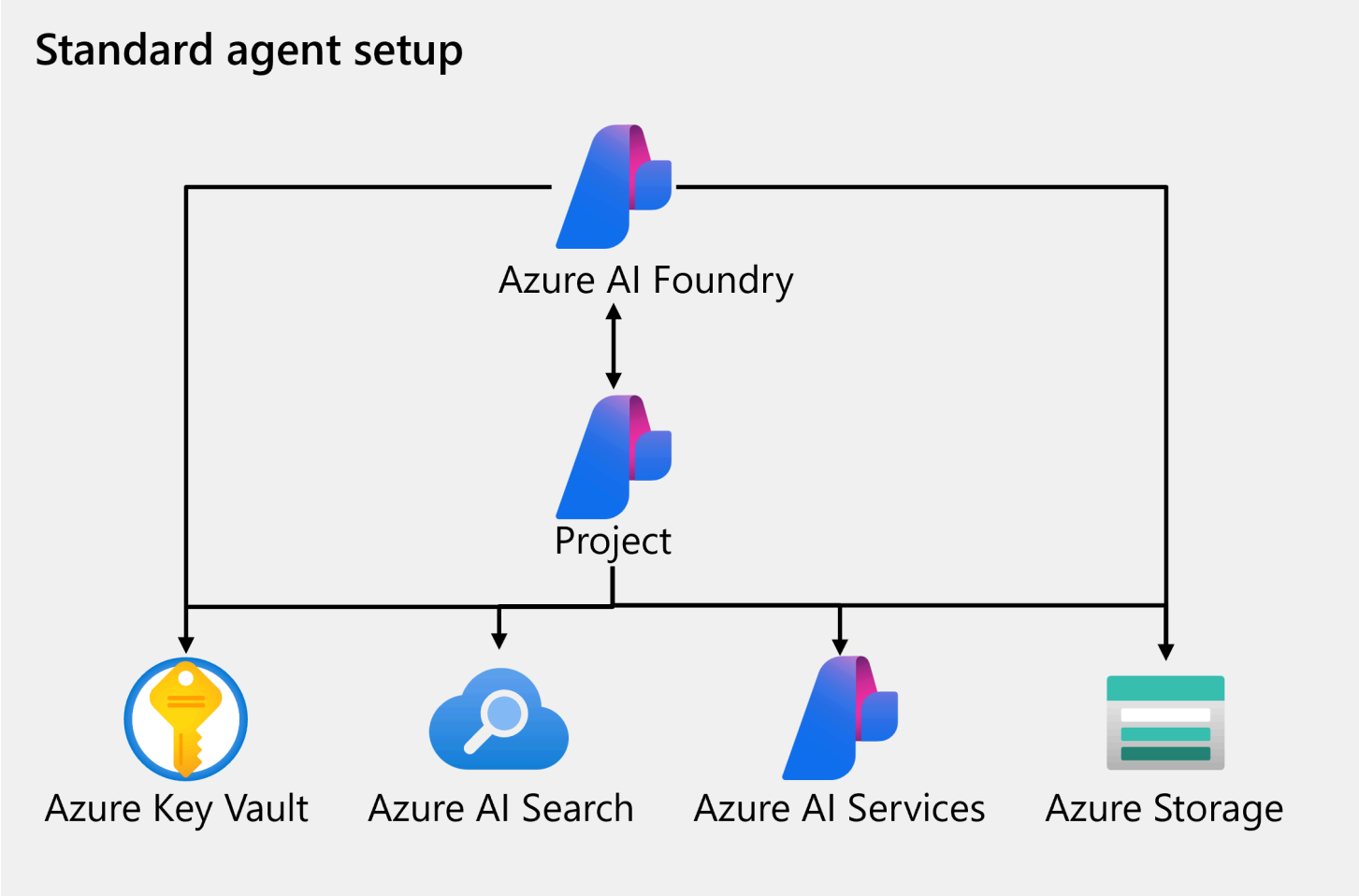
Foundry Agent Service provides a more streamlined and secure way to build and deploy AI agents compared to developing with the Inference API directly.

Foundry Agent Service resources

Foundry Agent Service is fully managed and designed to help developers build agents without having to worry about underlying resources. Through Azure, AI Foundry and the Agent Service will provision the necessary cloud resources. If desired, you can choose to connect your own resources when building your agent, giving you the flexibility to utilize Azure however works best for you.

At a minimum, you need to **create an Azure AI hub with an Azure AI project** for your agent. You can add more Azure services as required. You can c**reate the resources using the Azure AI Foundry portal**, or you can **use predefined bicep templates to deploy the resources** in your subscription. Two common architectures for Foundry Agent Service solutions are:

- Basic agent setup: A minimal configuration that includes Azure AI hub, Azure AI project, and Azure AI Services resources.
- Standard agent setup: A more comprehensive configuration that includes the basic agent setup plus Azure Key Vault, Azure AI Search, and Azure Storage.



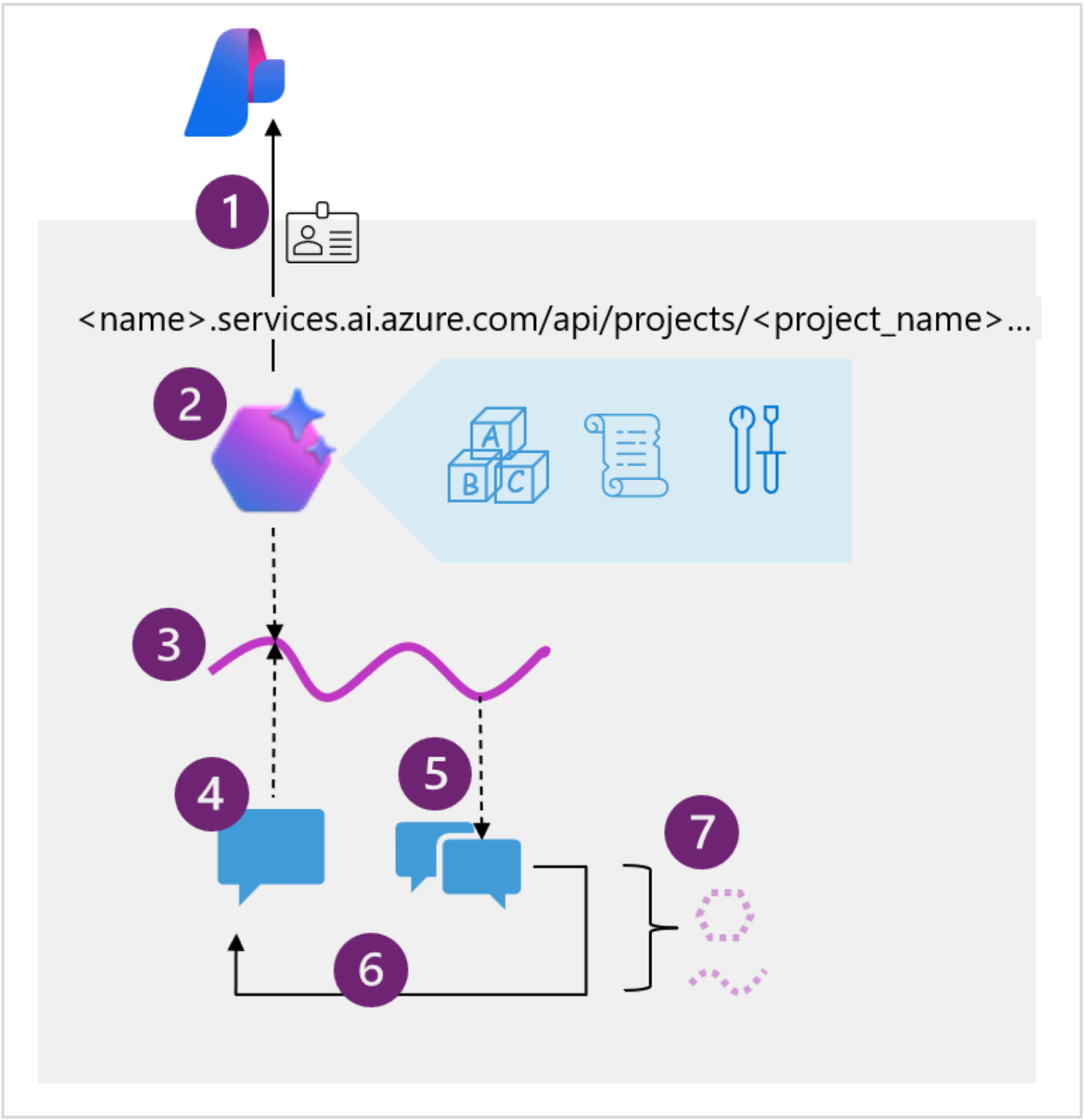
Develop agents with the Azure AI Foundry Agent Service

Previous solutions to achieve an agent-like experience took hundreds of lines of code to do things like referencing grounding data or connecting to a custom function. The Agent Service now simplifies all of that, supporting client-side function calling with just a few lines of code and connections to Azure Functions or an OpenAPI defined tool.

> Note: Foundry Agent Service offers several advantages to building agents, but isn't always the right solution for your use case. For example, if you're trying to **build an integration with Microsoft 365** you might choose the **Copilot Studio agent builder** and if you're trying to **orchestrate multiple agents**, you might choose the **Semantic Kernel Agents Framework**. This Fundamentals of AI Agents unit explores more of the options for building agents.

Developing apps that use agents

Foundry Agent Service provides several **SDKs and a REST API** for you to integrate agents into your app using your preferred programming language. The exercise later in this module focuses on Python, but the overall pattern is the same for REST or other language SDKs.



The diagram shows the following high-level steps that you must implement in your code:

1. Connect to the *AI Foundry project* for your agent, using the project **endpoint and Entra ID authentication**.
2. Get a reference to an existing agent that you created in the Azure AI Foundry portal, or create a new one specifying:
    - The **model deployment** in the project that the agent should use to interpret and respond to prompts.
    - **Instructions** that determine the functionality and behavior of the agent.
    - **Tools** and **resources** that the agent can use to perform tasks.
3. Create a **thread** for a chat session with the agent. All conversations with an agent are conducted on a stateful thread that retains message history and data artifacts generated during the chat.
4. Add **messages** to the thread and invoke it with the agent.
5. Check the thread **status**, and when ready, retrieve the messages and data artifacts.
6. Repeat the previous two steps as a **chat loop** until the conversation can be concluded.
7. When finished, delete the agent and the thread to clean up the resources and delete data that is no longer required.

Tools available to your agent

Much of the enhanced functionality of an agent comes from the agent's ability to determine when and how to use **tools**. Tools make additional functionality available to your agent, and if the conversation or task warrants the use of one or more of the tools, the agent calls that tool and handle the response.

You can assign tools when creating an agent in the Azure AI Foundry portal, or when defining an agent in code using the SDK.

For example, one of the tools available is the **code interpreter**. This tool enables your agent to run custom code it writes to achieve something, such as MATLAB code to create a graph or solve a data analytics problem.

Available tools are split into two categories:

1. Knowledge tools

Knowledge tools enhance the context or knowledge of your agent. Available tools include:

- **Bing Search**: Uses Bing search results to ground prompts with real-time live data from the web.
- **File search**: Grounds prompts with data from files in a vector store.
- **Azure AI Search**: Grounds prompts with data from Azure AI Search query results.
- **Microsoft Fabric**: Uses the **Fabric Data Agent** to ground prompts with data from your Fabric data stores.

> Tip: You can also integrate third-party licensed data by using the OpenAPI Spec action tool (discussed below).

2. Action tools

Action tools perform an action or run a function. Available tools include:

- **Code Interpreter**: A sandbox for model-generated Python code that can access and process uploaded files.
- **Custom function**: Call your custom function code – you must provide function definitions and implementations.
- **Azure Function**: Call code in serverless Azure Functions.
- **OpenAPI Spec**: Call external APIs based on the OpenAPI 3.0 spec.

By connecting built-in and custom tools, you can allow your agent to perform countless tasks on your behalf.

Exercise - Build an AI agent

Now it's your opportunity to build an agent in Azure AI Foundry. In this exercise, you create an agent and test it in Agent playground. You'll then develop your own app that integrates with an agent through Foundry Agent Service.

Develop an AI agent

In this exercise, you'll use Azure AI Agent Service to create a simple agent that analyzes data and creates charts. The agent uses the built-in Code Interpreter tool to dynamically generate the code required to create charts as images, and then saves the resulting chart images.

Module assessment

1. What is the first step in setting up Azure AI Foundry Agent Service? **Create an Azure AI Foundry Project**.
2. Which element of an agent definition is used to specify its behavior and restrictions? **Instructions**.

3. Which tool should you use to enable an agent to dynamically generate code to perform tasks or access data in files? **Code Interpreter**.

Summary

AI agents represent a significant advancement in the field of artificial intelligence, offering numerous benefits to businesses and individuals alike. By **automating routine tasks, enhancing decision-making, and providing scalable solutions**, AI agents are transforming how we work and interact with technology. As these agents continue to evolve, their potential applications will only expand, driving further innovation and efficiency across various sectors.

In this module, you learned about the purpose of **Foundry Agent Service**, its key features, the setup process, and its integration capabilities with other Azure AI services. We also addressed the challenge of building, deploying, and scaling AI agents. Foundry Agent Service solves several of these challenges, providing a fully managed environment for creating high-quality, extensible AI agents with minimal coding and infrastructure management.

The techniques covered in this module demonstrate several advantages, including **automatic tool calling, secure data management, and flexible model selection**. These features enable developers to focus on creating intelligent solutions while ensuring enterprise-grade security and compliance. The business impact includes streamlined development processes, reduced operational overhead, and enhanced AI capabilities.

After completing this module, you're now able to:

- Describe the ***purpose of AI agents***
- Explain the **key features of Foundry Agent Service**
- **Build an agent** using the Foundry Agent Service
- **Integrate an agent** in the Foundry Agent Service into your own app

More reading:

- Quickstart Guide for Azure AI Foundry Agent Service.
- What's new in Azure AI Foundry Agent Service.
- For detailed instructions, see the quickstart guide.

2.3 Develop AI agents with the Azure AI Foundry extension in Visual Studio Code

Learn how to build, test, and deploy AI agents using the **Azure AI Foundry extension in Visual Studio Code**.

Learning objectives

By the end of this module, you'll be able to:

- Configure and deploy AI agents using the **agent designer**
- Add tools and capabilities to extend your agents' functionality
- Test agents using the integrated playground
- Generate sample code to integrate agents into applications

Introduction

As generative AI models become more powerful and accessible, developers are moving beyond simple chat applications to build intelligent agents that can automate complex tasks. These AI agents combine large language models with specialized tools to **access data, perform actions, and complete entire workflows** with minimal human intervention.

Visual Studio Code is a powerful platform for AI agent development, especially with the Azure AI Foundry extension. This extension brings enterprise-grade AI capabilities directly into your development environment. With the Azure AI Foundry for Visual Studio Code extension, you can discover and deploy models, create and configure agents, and test them in interactive playgrounds—all without leaving your code editor.

This module introduces the core concepts of AI agent development and shows you how to use the Azure AI Foundry extension for Visual Studio Code to build, test, and deploy intelligent agents. You'll learn how to use Azure AI Foundry's Agent Service to **create agents that can handle complex tasks, use various tools, and integrate with your existing applications**.

Get started with the Azure AI Foundry extension

The Azure AI Foundry for Visual Studio Code extension transforms your development environment into a comprehensive platform for building, testing, and deploying AI agents. This extension provides direct access to the capabilities of Azure AI Foundry's Agent Service without leaving your code editor, streamlining the entire agent development workflow.

What is the Azure AI Foundry for Visual Studio Code extension?

The Azure AI Foundry for Visual Studio Code extension is a powerful tool that brings enterprise-grade AI agent development capabilities directly into Visual Studio Code. It provides an integrated experience for:

- **Agent Discovery and Management** - Browse, create, and manage AI agents within your Azure AI Foundry projects
- **Visual Agent Designer** - Use an intuitive interface to configure agent instructions, tools, and capabilities
- **Integrated Testing** - Test agents in real-time using the built-in playground without switching contexts
- **Code Generation** - Generate sample integration code to connect agents with your applications
- **Deployment Pipeline** - Deploy agents directly to Azure AI Foundry for production use

Key features and capabilities

The extension offers several key features that accelerate AI agent development:

Agent Designer Interface

A visual designer that simplifies agent creation and configuration. You can define agent instructions, select appropriate models, and configure tools through an intuitive graphical interface.

Built-in Playground

An integrated testing environment where you can interact with your agents in real-time, test different scenarios, and refine agent behavior before deployment.

Tool Integration

Seamless integration with various tools including:

- **RAG (Retrieval-Augmented Generation)** for knowledge-based responses
- **Search** capabilities for information retrieval
- **Custom actions** for specific business logic
- **Model Context Protocol (MCP)** servers for extended functionality

Project Integration

Direct connection to your Azure AI Foundry projects, allowing you to work with existing resources and deploy new agents to your established infrastructure.

Installing the extension

To get started with the Azure AI Foundry for Visual Studio Code extension, first you need to have Visual Studio Code installed on your machine. You can download it from the Visual Studio Code website.

You can install the Azure AI Foundry extension directly from the Visual Studio Code Marketplace:

1. Open Visual Studio Code.
2. Select Extensions from the left pane, or press Ctrl+Shift+X.
3. Search for and select Azure AI Foundry.
4. Select Install.
5. Verify the extension is installed successfully from the status messages.

Getting started workflow

The typical workflow for using the Azure AI Foundry extension follows these steps:

1. **Install and configure** the extension in Visual Studio Code
2. **Connect** to your Azure AI Foundry project
3. **Create or import** an AI agent using the designer
4. **Configure agent** instructions and add necessary tools
5. **Test the agent** using the integrated playground
6. **Iterate** on the design based on test results
7. **Generate code** for application integration

This streamlined workflow enables rapid prototyping and deployment of AI agents, making it easier for developers to build intelligent automation solutions that can handle complex real-world tasks.

The Azure AI Foundry for Visual Studio Code extension represents a significant step forward in making AI agent development more accessible and efficient, providing developers with enterprise-grade tools

in a familiar development environment.

Develop AI agents in Visual Studio Code

Creating and configuring AI agents in Visual Studio Code using the Azure AI Foundry extension provides a streamlined development experience that combines the power of Azure AI Foundry Agent Service with the familiar Visual Studio Code environment. This approach enables you to design, configure, and test agents without leaving your development environment.

Understanding Azure AI Foundry Agent Service

Azure AI Foundry Agent Service is **a managed service in Azure designed to provide a comprehensive framework for creating, managing, and deploying AI agents**. The service builds on the OpenAI Assistants API foundation while offering enhanced capabilities including:

- **Expanded model choice** - Support for multiple AI models beyond OpenAI
- **Enterprise security** - Built-in security features for production environments
- **Advanced data integration** - Seamless connection to Azure data services
- **Tooling ecosystem** - Access to a various built-in and custom tools

The Visual Studio Code extension provides direct access to these capabilities through an intuitive interface that simplifies the agent development process.

Creating agents with the extension

The Azure AI Foundry extension provides multiple ways to create AI agents, whether you're starting from scratch or building on existing work. The flexible approach accommodates different development preferences and scenarios.

Prerequisites for agent creation

Before creating an agent, complete the following steps:

1. Complete the extension setup and sign in to your Azure account
2. Create a default Azure AI Foundry project, or select an existing one
3. Select and deploy the model for your agent to use, or use an existing deployment

Creating a new agent

To create a new AI agent, follow these steps:

1. Open the Azure AI Foundry Extension view in Visual Studio Code
2. Navigate to the Resources section
3. Select the + (plus) icon next to the Agents subsection to create a new AI agent
4. Configure the agent properties in the Agent Designer view that opens

When you create an agent, the extension opens both the agent .yaml file and the Designer view, providing you with both a visual interface and direct access to the configuration file.

Configuring agent properties

Once you create an agent, the extension provides comprehensive configuration options to define how your agent behaves and interacts with users. The Agent Designer provides an intuitive interface for setting up these properties.

Basic Configuration

In the Agent Designer, configure the following essential properties:

- **Agent name** - Enter a descriptive name for your agent in the prompt
- **Model selection** - Choose your model deployment from the dropdown (this is the deployment name you chose when deploying a model)
- **Description** - Add a clear description of what your agent does
  System instructions - Define the agent's behavior, personality, and response style
- **Agent ID** - Automatically generated by the extension

Understanding the agent YAML file

Your AI agent is defined in a YAML file that contains all configuration details. Here's an example structure:

```yaml
yaml-language-server: $schema=https://aka.ms/ai-foundry-vsc/agent/1.0.0
version: 1.0.0
name: my-agent
description: Description of the agent
id: ''
metadata:
  authors:
    - author1
    - author2
  tags:
    - tag1
    - tag2
model:
  id: 'gpt-4o-1'
  options:
    temperature: 1
    top_p: 1
instructions: Instructions for the agent
tools: []
```

This YAML file is opened automatically alongside the Designer view, allowing you to work with either the visual interface or edit the configuration directly.

Agent instruction design

Well-crafted instructions are the foundation of effective AI agents. They define how your agent understands its role, responds to users, and handles various scenarios.

Best practices for instructions

When writing system instructions for your agent:

- **Be specific and clear** - Define exactly what the agent should do and how it should behave
- **Provide context** - Explain the agent's role and the environment it operates in
- **Set boundaries** - Clearly define what the agent should and shouldn't do
- **Include examples** - Show the agent examples of desired interactions when helpful
- **Define personality** - Establish the tone and style of responses

Instruction examples

For a customer service agent, effective instructions might include:

- The agent's role and purpose
- Guidelines for handling different types of customer inquiries
- Escalation procedures for complex issues
- Tone and communication style preferences

Deploying agents

Once you configure your agent, you can deploy it to Azure AI Foundry.

Deployment process

To deploy your agent:

- **Select the "Create on Azure AI Foundry" button** in the bottom-left of the Designer
- **Wait for deployment completion** - The extension handles the deployment process
- **Refresh the Azure Resources view** in the Visual Studio Code navbar
- **Verify deployment** - The deployed agent appears under the Agents subsection

Managing deployed agents

After deployment, you can:

- **View agent details** - Select the deployed agent to see the Agent Preferences page
- **Edit the agent** - Select "Edit Agent" to modify configuration and redeploy with the Update on Azure AI Foundry button
- **Generate integration code** - Select "Open Code File" to create sample code for using the agent
- **Test in playground** - Select "Open Playground" to interact with the deployed agent

Testing and iteration

The integrated playground enables immediate testing of your agent configuration, allowing you to validate behavior and make adjustments in real-time.

Using the playground

After configuring your agent, you can test it using the built-in playground:

- **Real-time conversations** - Chat with your agent to test responses
- **Instruction validation** - Verify the agent follows its configured instructions
- **Behavior testing** - Test how the agent handles different types of requests
- **Iterative refinement** - Make adjustments based on testing results

Working with agent threads

When you interact with deployed agents, **the system creates threads to manage conversation sessions**:

- **Threads** - Conversation sessions between an agent and user that store messages and handle context management
- **Messages** - Individual interactions that can include text, images, and files
- **Runs** - Single executions of an agent that use the agent's configuration and thread messages

You can view and manage these threads through the Azure Resources view in the extension.

Creating and configuring AI agents with the Azure AI Foundry Visual Studio Code extension provides a powerful yet accessible approach to agent development. The extension provides visual design tools, direct YAML editing, comprehensive configuration options, and integrated testing capabilities. These features enable developers to rapidly prototype and deploy sophisticated AI agents that can handle complex real-world scenarios.

Extend AI agent capabilities with tools

One of the most powerful features of AI agents is their ability to use tools that extend their capabilities beyond simple text generation. The Azure AI Foundry for Visual Studio Code extension makes it easy to add and configure tools for your agents. These tools enable agents to perform actions, access data, and integrate with external systems.

Understanding agent tools

Tools are programmatic functions that enable agents to automate actions and access information beyond their training data. When an agent determines that a tool is needed to respond to a user request, it can automatically invoke the appropriate tool, process the results, and incorporate them into its response. This capability transforms agents from simple text generators into powerful automation systems that can interact with real-world data and services.

Built-in tools

Azure AI Foundry provides several built-in tools that you can easily add to your agents without any additional configuration or setup. These tools are production-ready and handle common use cases that many agents require.

- **Code Interpreter** - Enables agents to **write and execute Python code for mathematical calculations, data analysis, chart generation, file processing, and complex problem-solving**
- **File Search** - Provides retrieval-augmented generation by uploading and indexing documents, searching knowledge bases, and supporting various file formats (PDF, Word, text files)
- **Grounding with Bing Search** - Allows agents to search the internet for real-time data, current events, and trending topics while providing citations and sources
- **OpenAPI Specified Tools** - Connects agents to external APIs and services through OpenAPI 3.0 specifications
- **Model Context Protocol (MCP)** - Standardized tool interfaces for extended functionality and community-driven tools

Adding tools in Visual Studio Code

The Azure AI Foundry extension provides an intuitive interface for adding tools to your agents through a streamlined process. The visual interface makes it easy to browse, configure, and test tools without writing any code:

- Select your agent in the extension
- Navigate to the Tools section in the configuration panel
- Browse available tools from the tool library
- Configure tool settings as needed
- Test tool integration using the playground

When you add a tool, you can also add any new assets it needs. For example, if you add a File Search tool, you can use an existing vector store asset or make a new asset for your vector store to host your uploaded files.

Model Context Protocol (MCP) servers

MCP servers provide a standardized way to add tools to your agents using an open protocol. This approach enables you to use community-built tools and create reusable components that work across different agent implementations.

Key benefits include:

- **Standardized protocol** for consistent tool communication
- **Reusable components** that work across different agents
- **Community-driven tools** available through MCP registries
- **Simplified integration** with consistent interfaces

The extension supports adding MCP servers through registry browsing, custom server addition, configuration management, and testing and validation.

Tool management and best practices

Effective tool management ensures your agents perform reliably and efficiently in production environments. Following best practices helps you avoid common pitfalls and optimize agent performance:

Tool Selection Guidelines

- Identify what capabilities your agent requires
- Start with built-in tools before adding custom solutions
- Test thoroughly to validate tool behavior in various scenarios
- Monitor performance to track tool usage and effectiveness

Adding tools and extending agent capabilities through the Azure AI Foundry Visual Studio Code extension enables you to create sophisticated AI agents that can handle complex real-world tasks. By combining built-in tools with custom functions and MCP servers, you can build agents that seamlessly integrate with your existing systems and business processes while maintaining enterprise-grade security and performance.

Exercise - Build an AI agent using the Azure AI Foundry extension

If you have an Azure subscription, you can explore the Azure AI Foundry Extension for Visual Studio Code by completing this hands-on exercise. This exercise guides you through creating and deploying an AI agent using the extension.

Develop an AI agent with VS Code extension

In this exercise, you'll use the Azure AI Foundry VS Code extension to create an agent that can use Model Context Protocol (MCP) server tools to access external data sources and APIs. The agent will be able to retrieve up-to-date information and interact with various services through MCP tools.

Module assessment

1. When you create a new agent in the Azure AI Foundry extension, what two views are automatically opened? The YAML file and the Designer view
2. What is a key benefit of using Model Context Protocol (MCP) servers for agent tools? They provide reusable components that work across different agents
3. How does the Azure AI Foundry Agent Service manage conversation sessions when users interact with deployed agents? The system creates individual threads for each conversation to manage context and message history.

Summary

In this module, you learned how to develop AI agents using the Azure AI Foundry for Visual Studio Code extension. You explored how to create, configure, and test agents directly within your development environment. You also learned to extend their capabilities with various tools and deploy them to production environments.

> Tip: For more information, see Work with the Azure AI Foundry for Visual Studio Code extension.

2.4 Integrate custom tools into your agent

Built-in tools are useful, but they may not meet all your needs. In this module, learn how to extend the capabilities of your agent by **integrating custom tools** for your agent to use.

Learning objectives

By the end of this module, you'll be able to:

- Describe the benefits of **using custom tools** with your agent.
- Explore the different options for custom tools.
- **Build an agent that integrates custom tools** using the **Azure AI Foundry Agent Service**.

Introduction

Azure AI Foundry Agent Service offers a seamless way to build an agent without needing extensive AI or machine learning expertise. By using tools, you can provide your agent with functionality to execute actions on your behalf.

The AI Agent Service provides built-in tools for gathering knowledge and generating code, which provide your agent with some powerful functionality. However, sometimes your agent needs to be able to complete specific tasks or actions that an AI model would struggle to handle on its own. To accomplish these actions, you can **provide your agent a custom tool based on your own code or a third-party service or API**.

Imagine you're working in the retail industry, and your company is struggling with managing customer inquiries efficiently. The customer support team is overwhelmed with repetitive questions, leading to delays in response times and decreased customer satisfaction. By using Foundry Agent Service with custom tools, you can **create a custom FAQ agent that handles common inquiries**. This agent can be provided with a set of custom tools to look up customer orders, freeing up your support team to focus on more complex issues.

In this module, you'll learn how to **utilize custom tools in Foundry Agent Service** to enhance productivity, improve accuracy, and create tailored solutions for specific needs.

Why use custom tools

Azure AI Foundry Agent Service offers a powerful platform for integrating custom tools to enhance productivity and provide tailored solutions for specific business needs. By using these custom tools, businesses can achieve greater efficiency and effectiveness in their operations.

Why use custom tools?

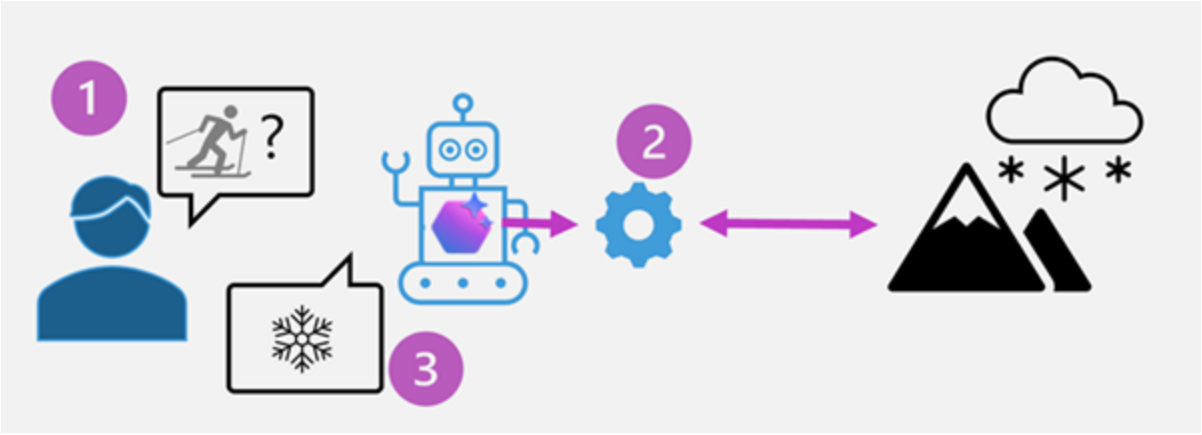
Custom tools in Azure AI services can significantly **enhance productivity** by automating repetitive tasks and streamlining workflows that are specific to your use case. These tools improve accuracy by providing precise and consistent outputs, reducing the likelihood of human error. Additionally, custom tools offer tailored solutions that address specific business needs, enabling organizations to optimize their processes and achieve better outcomes.

- **Enhanced productivity**: Automate repetitive tasks and streamline workflows.
- **Improved accuracy**: Provide precise and consistent outputs, reducing human error.
- **Tailored solutions**: Address specific business needs and optimize processes.

Adding tools makes custom functionality available for the agent to use, depending on how it decides to respond to the user prompt. For example, consider how **a custom tool to retrieve weather data from an external meteorological service** could be used by an agent.

The diagram shows the process of an agent choosing to use the custom tool:

1. A user asks an agent about the weather conditions in a ski resort.
2. The agent determines that it has access to a tool that can use an API to get meteorological information, and calls it.
3. The tool returns the weather report, and the agent informs the user.



Common scenarios for custom tools in agents

Custom tools within the Foundry Agent Service enable users to extend the capabilities of AI agents, tailoring them to meet specific business needs. Some example use cases that illustrate the versatility and impact of custom tools include:

Customer support automation

- **Scenario**: A retail company integrates a custom tool that connects the Azure AI Agent to their **customer relationship management (CRM) system**.
- **Functionality**: The AI agent can **retrieve customer order histories, process refunds, and provide real-time updates on shipping statuses**.
- **Outcome**: Faster resolution of customer queries, reduced workload for support teams, and improved customer satisfaction.

Inventory management

- **Scenario**: A manufacturing company develops a custom tool to link the AI agent with their **inventory management system**.
- **Functionality**: The AI agent can **check stock levels, predict restocking needs using historical data, and place orders with suppliers automatically**.
- **Outcome**: Streamlined inventory processes and optimized supply chain operations.

Healthcare appointment scheduling

- **Scenario**: A healthcare provider integrates a **custom scheduling tool** with the AI agent.
- **Functionality**: The AI agent can **access patient records, suggest available appointment slots, and send reminders to patients**.
- **Outcome**: Reduced administrative burden, improved patient experience, and better resource utilization.

IT Helpdesk support

- **Scenario**: An IT department develops a custom tool to integrate the AI agent with their **ticketing and knowledge base systems**.
- **Functionality**: The AI agent can **troubleshoot common technical issues, escalate complex problems, and track ticket statuses**.
- **Outcome**: Faster issue resolution, reduced downtime, and improved employee productivity.

E-learning and training

- **Scenario**: An educational institution creates a custom tool to connect the AI agent with their **learning management system (LMS)**.
- **Functionality**: The AI agent can **recommend courses, track student progress, and answer questions about course content**.
- **Outcome**: Enhanced learning experiences, increased student engagement, and streamlined administrative tasks.

These examples demonstrate how **custom tools within the Foundry Agent Service** can be used across industries to address unique challenges, drive efficiency, and deliver value.

Options for implementing custom tools

Azure AI Foundry Agent Service offers various custom tools that enhance the capabilities and efficiency of your AI agents. These tools allow for scalable interoperability with various applications, making it easier to integrate with existing infrastructure or web services.

Custom tool options available in Azure AI Foundry Agent Service

Azure AI services provide several custom tool options, including **OpenAPI specified tools, Azure Functions, and function calling**. These tools enable seamless integration with **external APIs, event-driven applications, and custom functions**.

- **Custom function**: Function calling allows you to *describe the structure of custom functions to an agent and return the functions that need to be called along with their arguments*. The agent can dynamically identify appropriate functions based on their definitions. This feature is useful for integrating custom logic and workflows, in a selection of programming languages, into your AI agents.
- **Azure Functions**: Azure Functions enable you to create intelligent, event-driven applications with minimal overhead. They support **triggers and bindings**, which simplify how your AI Agents interact with external systems and services. **Triggers determine when a function executes, while bindings facilitate streamlined connections to input or output data sources**.
- **OpenAPI specification tools**: These tools allow you to connect your Azure AI Agent to an external API using an **OpenAPI 3.0 specification**. This provides standardized, automated, and scalable API integrations that enhance the capabilities of your agent. OpenAPI specifications

describe HTTP APIs, enabling people to understand how an API works, generate client code, create tests, and apply design standards.
- **Azure Logic Apps**: This action provides **low-code/no-code solutions** to add workflows and connects apps, data, and services with the **low-code Logic App**.

This flexibility to integrate custom functionality in multiple ways enables a wide range of extensibility possibilities for your Foundry Agent Service agents.

How to integrate custom tools

Custom tools in an agent can be defined in a handful of ways, depending on what works best for your scenario. You may find that your company already has **Azure Functions** implemented for your agent to use, or a **public OpenAPI specification** gives your agent the functionality you're looking for.

Function Calling

Function calling allows agents to execute predefined functions dynamically based on user input. This feature is ideal for scenarios where agents need to perform specific tasks, such as retrieving data or processing user queries, and can be done in code from within the agent. Your function may call out to other APIs to get additional information or initiate a program.

Example: Defining and using a function

Start by defining a function that the agent can call. For instance, here's a fake snowfall tracking function:

```
import json

def recent_snowfall(location: str) -> str:
    """
    Fetches recent snowfall totals for a given location.
    :param location: The city name.
    :return: Snowfall details as a JSON string.
    """
    mock_snow_data = {"Seattle": "0 inches", "Denver": "2 inches"}
    snow = mock_snow_data.get(location, "Data not available.")
    return json.dumps({"location": location, "snowfall": snow})

user_functions: Set[Callable[..., Any]] = {
    recent_snowfall,
}
```

Register the function with your agent using the Azure AI SDK:

```
# Initialize agent toolset with user functions
functions = FunctionTool(user_functions)
toolset = ToolSet()
toolset.add(functions)
agent_client.enable_auto_function_calls(toolset=toolset)

# Create your agent with the toolset
agent = agent_client.create_agent(
    model="gpt-4o-mini",
    name="snowfall-agent",
    instructions="You are a weather assistant tracking snowfall. Use the provided functions to a
    toolset=toolset
)
```

The agent can now call `recent_snowfall` dynamically when it determines that the prompt requires information that can be retrieved by the function.

Azure Functions

Azure Functions provide **serverless computing capabilities for real-time processing**. This integration is ideal for event-driven workflows, **enabling agents to respond to triggers such as HTTP requests or queue messages**.

Example: Using Azure Functions with a queue trigger

First, develop and deploy your Azure Function. In this example, imagine we have a function in our Azure subscription to fetch the snowfall for a given location.

When your Azure Function is in place, integrate add it to the agent definition as an Azure Function tool:

```
storage_service_endpoint = "https://<your-storage>.queue.core.windows.net"

azure_function_tool = AzureFunctionTool(
    name="get_snowfall",
    description="Get snowfall information using Azure Function",
    parameters={
            "type": "object",
            "properties": {
                "location": {"type": "string", "description": "The location to check snowfall."}
            },
            "required": ["location"],
        },
    input_queue=AzureFunctionStorageQueue(
        queue_name="input",
        storage_service_endpoint=storage_service_endpoint,
    ),
    output_queue=AzureFunctionStorageQueue(
        queue_name="output",
        storage_service_endpoint=storage_service_endpoint,
    ),
)

agent = agent_client.create_agent(
    model=os.environ["MODEL_DEPLOYMENT_NAME"],
    name="azure-function-agent",
    instructions="You are a snowfall tracking agent. Use the provided Azure Function to fetch sn
    tools=azure_function_tool.definitions,
)
```

The agent can now send requests to the Azure Function via a storage queue and process the results.

OpenAPI Specification

OpenAPI defined tools allow agents to interact with external APIs using standardized specifications. This approach simplifies API integration and ensures compatibility with various

services. The Foundry Agent Service uses **OpenAPI 3.0** specified tools.

> Tip: Currently, three authentication types are supported with OpenAPI 3.0 tools: **anonymous, API key**, and **managed identity**.

Example: Using an OpenAPI specification

First, create a JSON file (in this example, called `snowfall_openapi.json`) describing the API.

Then, register the OpenAPI tool in the agent defintion:

```
from azure.ai.agents.models import OpenApiTool, OpenApiAnonymousAuthDetails

with open("snowfall_openapi.json", "r") as f:
    openapi_spec = json.load(f)

auth = OpenApiAnonymousAuthDetails()
openapi_tool = OpenApiTool(name="snowfall_api", spec=openapi_spec, auth=auth)

agent = agent_client.create_agent(
    model="gpt-4o-mini",
    name="openapi-agent",
    instructions="You are a snowfall tracking assistant. Use the API to fetch snowfall data.",
    tools=[openapi_tool]
)
```

The agent can now use the OpenAPI tool to fetch snowfall data dynamically.

> Note: One of the concepts related to agents and custom tools that developers often have difficulty with is the declarative nature of the solution. **You don't need to write code that explicitly calls your custom tool functions** - **the agent itself decides to call tool functions based on messages in prompts**. By providing the agent with functions that have meaningful names and well-documented parameters, the agent can "figure out" when and how to call the function all by itself!

By using one of the available custom tool options (or any combination of them), you can create powerful, flexible, and intelligent agents with Foundry Agent Service. These integrations enable seamless interaction with external systems, real-time processing, and scalable workflows, making it easier to build custom solutions tailored to your needs.

Exercise - Build an agent with custom tools

Now it's your opportunity to build an agent with custom tools. In this exercise, you create an agent in code and connect the tool definition to a custom tool function.

Use a custom function in an AI agent

In this exercise you'll explore creating an agent that can use custom functions as a tool to complete tasks.

You'll build a simple technical support agent that can collect details of a technical problem and generate a support ticket.

Module assessment

1. What are custom tools, and how can they help you develop effective agents with Azure AI Foundry Agent Service? **Callable functions that an agent can use to extend its capabilities**.
2. You need to integrate functionality from an OpenAPI 3.0-based web service into an agent solution. What should you do? **Add the web services as an OpenAPI specification tool to the agent definition**.
3. Your agent application code includes a local function that you want the agent to call. What kind of tool should you add to the agent's definition? **Function calling**.

Summary

In this module, we covered the benefits of **integrating custom tools into Foundry Agent Service** to boost productivity and provide tailored business solutions. By providing custom tools to our agent, we can optimize processes to meet specific needs, resulting in better responses from your agent.

The techniques learned in this module enable businesses to generate marketing materials, improve communications, and analyze market trends more effectively, all through custom tools. The integration of various tool options in the AI Agent Service, from Azure Functions to OpenAPI specifications, allows for the creation of intelligent, event-driven applications that use well-established patterns already used in many businesses.

Further reading

- AI Agents for beginners tool use
- Azure AI Foundry Agent Service function calling
- Introduction to Azure Functions
- OpenAPI Specification

Develop a multi-agent solution with Azure AI Foundry Agent Service

Break down complex tasks with intelligent collaboration. Learn how to design multi-agent solutions using **connected agents**.

Learning objectives

After completing this module, you'll be able to:

- Describe how **connected agents** enable modular, collaborative workflows.
- Design a **multi-agent solution** by defining **main agent tools** and **connected agent roles**.
- Build and run a connected agent solution

Introduction

Azure AI Foundry Agent Service can help you develop sophisticated, **multi-agent systems** that can break down complex tasks into smaller, specialized roles. Using **connected agents**, you can design intelligent solutions where a primary agent delegates work to sub-agents without the need for custom orchestration logic or hardcoded routing. This modular approach boosts efficiency and maintainability across a wide range of scenarios.

Imagine you're on an engineering team that receives a constant flow of support tickets including bugs, feature requests, and infrastructure issues. Manually reviewing and sorting each one takes time and slows down your team's ability to respond quickly. With a **multi-agent approach**, you can build a triage assistant that assigns different tasks to specialized agents. The sub-agents might perform tasks such as classifying ticket type, setting priority, and suggesting the right team for the work. With a multi-agent approach, these specialized agents can work together to streamline the ticketing process.

In this module, you learn how to use connected agents in Azure AI Foundry. You also practice building **an intelligent ticket triage system using a collaborative mutli-agent solution**.

Understand connected agents

As AI solutions become more advanced, managing complex workflows gets harder. A single agent can handle a wide range of tasks, but this approach can become unmanageable as the scope expands. That's why Azure AI Foundry Agent Service lets you connect multiple agents, each with a focused role, to work together in a cohesive system.

What are connected agents?

Connected agents are a feature in the Azure AI Foundry Agent Service that allows you to break large tasks into smaller, specialized roles without building a custom orchestrator or hardcoding routing logic. Instead of relying on one agent to do everything, you can create multiple agents with clearly defined responsibilities that collaborate to accomplish tasks.

At the center of this system, there's **a main agent that interprets user input and delegates tasks to connected sub-agents**. **Each sub-agent is designed to perform a specific function**, such as to summarize a document, validate a policy, or retrieve data from a knowledge source.

This division of labor helps you:

- Simplify complex workflows
- Improve agent performance and accuracy
- Make systems easier to maintain and extend over time

Why use connected agents?

Rather than scaling a single agent to handle every user request or data interaction, using connected agents lets you:

- Build modular solutions that are easier to develop and debug
- Assign specialized capabilities to agents that can be reused across solutions
- Scale your system in a way that aligns with real-world business logic

This approach is especially useful in scenarios where agents need to perform sensitive tasks independently, such has handling private data or generating personalized content.

Using connected agents to automate workflows offers many benefits, for example:

- **No custom orchestration required** - The main agent uses natural language to route tasks, eliminating the need for hardcoded logic.

- **Improved reliability and traceability** - The clear separation of responsibilities makes issues easier to debug since agents can be tested individually.
- **Flexible and extensible** - Add or swap agents without reworking the entire system or modifying the main agent.

Connected agents make it easier to build modular, collaborative systems without complex orchestration. By assigning focused roles and using natural language delegation, you can simplify workflows, improve reliability, and scale your solutions more effectively.

Design a multi-agent solution with connected agents

In a connected agent solution, **success depends on clearly defining the responsibilities of each agent**. **The central agent is also responsible for how the agents will collaborate**. Let's explore how to design a multi-agent program using Azure AI Foundry Agent Service.

Main agent (orchestrator) responsibilities

The main agent acts as the orchestrator. It *interprets the intent behind a request and determines which connected agent is best suited to handle it*. The main agent is responsible for:

- Interpreting user input
- Selecting the appropriate connected agent
- Forwarding relevant context and instructions
- Aggregating or summarize results

Connected agent responsibilities

Connected agents designed to focus on **a single domain of responsibility**. A connected agent is responsible for:

- Completing a specific action based on a clear prompt
- Using tools (if needed) to complete their task
- Returning the results to the main agent

Connected agents should be designed with a **single responsibility** in mind. This makes your system easier to debug, extend, and reuse.

Set up a multi-agent solution with connected agents

1. **Initialize the agents client**
   First, you create a client that connects to your Azure AI Foundry project.
2. **Create an agent to connect to the main agent**
   Define an agent you want to connect to the main agent. You can do this using the `create_agent` method on the `AgentsClient` object.
   For example, your connected agent might retrieve stock prices, summarize documents, or validate compliance. Give the agent clear instructions that define its purpose.
3. **Initialize the connected agent tool**
   Use your agent definition to create a `ConnectedAgentTool` . Assign it a name and description so the main agent knows when and how to use it.
4. **Create the main agent**
   Create the main agent using the `create_agent` method. Add your connected agents using the `tools` property and assign the `ConnectedAgentTool` definitions to the main agent.
5. **Create a thread and send a message**
   Create the **agent thread** that is used to manage the conversation context. Then create a message on the thread that contains the request you want the agent to fulfill.
6. **Run the agent workflow**
   Once the message is added, create a run to process the request. The main agent uses its tools to delegate tasks as needed and compile a final response for the user.
7. **Handle the results**
   When the run completes, you can review the main agent's response. The final output may incorporate insights from one or more connected agents. Only the main agent's response is visible to the end user.

Designing a connected agent system involves **defining focused agents, registering them as tools, and configuring a main agent to route tasks** intelligently. This modular approach gives you a flexible foundation for building collaborative AI solution that scale as your needs grow.

Exercise - Develop a multi-agent app with Azure AI Foundry

Develop a multi-agent solution

In this exercise, you'll create a project that orchestrates multiple AI agents using **Azure AI Foundry Agent Service**. You'll design an AI solution that assists with ticket triage. The connected agents will assess the ticket's priority, suggest a team assignment, and determine the level of effort required to complete the ticket. Let's get started!

Module assessment

1. What is the role of the main agent in a connected agent system? **To coordinate user input and route tasks to the appropriate connected agents**.
2. How do you connect an agent to a main agent using the Azure AI Projects client library? **Add the agent as a `ConnectedAgentTool` to the main agent's tool definition**.
3. How does the main agent decide which connected agent to use? **It uses prompt instructions and natural language understanding**.

Summary

In this module, you learned how to **design and implement multi-agent solutions** using **Azure AI Foundry Agent Service**.

Connected agents let you break down complex tasks by assigning them to specialized agents that work together within a coordinated system. You explored how to define clear roles for main and connected agents, delegate tasks using natural language, and design modular workflows that are easier to scale and maintain. You also practiced building a multi-agent solution. Great work!

Integrate MCP Tools with Azure AI Agents

Enable dynamic tool access for your Azure AI agents. Learn how to **connect MCP-hosted tools and integrate them seamlessly into agent workflows**.

Learning objectives

After completing this module, you're able to:

- Explain the roles of the **MCP server and client** in tool discovery and invocation.
- Wrap **MCP tools** as asynchronous functions and **register them with Azure AI agents**.
- Build an Azure AI agent that dynamically accesses and calls MCP tools during runtime.

Introduction

AI agents are capable of performing a wide range of tasks, but many tasks still require them to **interact with tools outside the large language model**. Agents may need to access APIs, databases, or internal services. Manually integrating and maintaining these tools can quickly become complex, especially as your system grows, or changes frequently.

Model Context Protocol (MCP) servers can help solve this problem by integrating with AI agents. Connecting an Azure AI Agent to a Model Context Protocol (MCP) server can provide your agent with a catalog of tools accessible on demand. This approach makes your AI solution more robust, scalable, and easier to maintain.

Suppose you're working for a retailer that specializes in cosmetics. Your team wants to build an AI assistant that can help manage inventory by checking product stock levels and recent sales trends. Using an MCP server, you can connect the assistant to a set of tools that can make inventory assessments and provide recommendations to the team.

In this module, you learn how to **set up an MCP server and client**, and connect tools to an Azure AI Agent dynamically. You also practice **creating your own AI MCP tool solution** with Azure AI Foundry Agent Service.

Understand MCP tool discovery

As AI agents become more capable, the range of tools and services they can access also grows. However, registering new tools, managing, updating, and integrating them can quickly become complex and time-consuming. **Dynamic tool discovery** helps solve this problem by enabling agents to find and use tools automatically at runtime.

What is *dynamic tool discovery*?

Dynamic tool discovery is **a mechanism that allows an AI agent to discover available external tools without needing hardcoded knowledge of each one**. Instead of manually adding or updating every tool your agent can use, the agent queries a centralized Model Context Protocol (MCP) server. This server acts as a live catalog, exposing tools that the agent can understand and call.

This approach means:

- Tools can be added, updated, or removed centrally without modifying the agent code.
- Agents can always use the latest version of a tool, improving accuracy and reliability.
- The complexity of managing tools shifts away from the agent and into a dedicated service.

How does MCP enable dynamic tool discovery?

An MCP server hosts a set of functions that are exposed as tools using the `@mcp.tool` decorator. **Tools are a primitive type in the MCP that enables servers to expose executable functionality to clients**. A client can connect to the server and fetch these tools dynamically. The client then generates function wrappers that are added to the Azure AI Agent's tool definitions. This setup creates a flexible pipeline:

- The **MCP server hosts available tools**.
- The **MCP client dynamically discovers the tools**.
- The **Azure AI Agent uses the available tools** to respond to user requests.

Why use dynamic tool discovery with MCP?

This approach provides several benefits:

- **Scalability**: Easily add new tools or update existing ones without redeploying agents.
- **Modularity**: Agents can remain simple, focusing on delegation rather than managing tool details.
- **Maintainability**: Centralized tool management reduces duplication and errors.
- **Flexibility**: Supports diverse tool types and complex workflows by aggregating capabilities.

Dynamic tool discovery is especially useful in environments where tools evolve rapidly or where many teams manage different APIs and services. Using tools allows AI agents to adapt to changing capabilities in real time, interact with external systems securely, and perform actions that go beyond language generation.

Integrate agent tools using an MCP server and client

To dynamically connect tools to your Azure AI Agent, you first need a functioning Model Context Protocol (MCP) setup. This includes both the **MCP server, which hosts your tool catalog**, and the **MCP client, which fetches those tools and makes them usable by your agent**.

What is the MCP Server?

The **MCP server acts as a registry for tools** your agent can use. You can initialize your MCP server using `FastMCP("server-name")`. The FastMCP class uses Python type hints and document strings to automatically generate tool definitions, making it easy to create and maintain MCP tools. These definitions are then served over HTTP when requested by the client. Because tool definitions live on the server, you can update or add new tools at any time, without having to modify or redeploy your agent.

What is the MCP Client?

A standard **MCP client acts as a bridge between your MCP server and the Azure AI Agent Service**. The client initializes an MCP client session and connects to the server. Afterwards, it performs three key tasks:

- **Discovers available tools from the MCP server** using `session.list_tools()`.
- **Generates Python function stubs** that wrap the tools.
- **Registers those functions with your agent**.

This allows the agent to call any tool listed in the MCP catalog as if it were a native function, all without hardcoded logic.

Register tools with an Azure AI Agent

When an MCP client session is initialized, the client can dynamically pull in tools from the MCP server. An MCP tool can be invoked using `session.call_tool(tool_name, tool_args)`. **The tools should each be wrapped in an async function so that the agent is able to invoke them**. Finally, those

functions are bundled together and become part of the agent's toolset and are available during runtime for any user request.

Overview of MCP agent tool integration

- The **MCP server** hosts **tool definitions** decorated with `@mcp.tool`.
- The **MCP client** initializes an MCP client connection to the server.
- The **MCP client fetches the available tool definitions** with `session.list_tools()`.
- **Each tool is wrapped in an async function** that invokes `session.call_tool`
- The tool functions are bundled into `FunctionTool` that makes them usable by the agent.
- The `FunctionTool` is registered to the agent's toolset.

Now your agent is able to access and invoke your tools through natural language interaction. By setting up the MCP server and client, you create a clean separation between tool management and agent logic—enabling your system to adapt quickly as new tools become available.

Use Azure AI agents with MCP servers

You can **enhance your Azure AI Foundry agent by connecting it to Model Context Protocol (MCP) servers**. MCP servers provide tools and contextual data that your agent can use to perform tasks, extending its capabilities beyond built-in functions. Azure AI Agent Service includes support for remote MCP servers, allowing your agent to quickly connect to your server and access tools.

When you use the Azure AI Foundry Agent Service to connect to your MCP server, you don't need to manually create an MCP client session or add any function tools to your agent. Instead, you **create an MCP tool object that connects to your MCP server**. Then you add information about the MCP server to the agent thread when invoking a prompt. This also allows you to connect and use different tools from multiple servers depending on your needs.

Integrating remote MCP servers

To connect to an MCP server, you need:

- A remote MCP server endpoint (example).
- An Azure AI Foundry agent configured to use the MCP tool.

You can connect to multiple MCP servers by adding them as separate tools, each with:

- `server_label`: A unique identifier for the MCP server (e.g., GitHub).

- `server_url` : The MCP server's URL.
- `allowed_tools` (*optional*): A list of specific tools the agent is allowed to access.

The MCP tool also supports custom headers, which let you pass:

- Authentication keys (e.g., API keys, OAuth tokens).
- Other required headers for the MCP server.
    - These headers are included in `tool_resources` during each run and are not stored between runs.

Invoking tools

When using the Azure MCP Tool object, you don't need to wrap function tools or invoke `session.call_tool` . Instead, the tools are automatically invoked when necessary during an agent run. To automatically invoke MCP tools:

- Create the `McpTool` object with the server label and URL.
- Use `update_headers` to apply any headers required by the server.
- Use the `set_approval_mode` to determine whether approval is required. Supported values are:
    - `always` : A developer needs to provide approval for every call. If you don't provide a value, this one is the default.
    - `never` : No approval is required.
- Create a `ToolSet` object and add the `McpTool` object
- Create an agent run and specify the `toolset` property
- When the run completes, you should see the results of any invoked tools in the response.

If the model tries to invoke a tool in your MCP server with approval required, you get a run status of `requires_action` . - In the `requires_action` field, you can get more details on which tool in the MCP server is called and any arguments to be passed. - Review the tool and arguments so that you can make an informed decision for approval. - Submit your approval to the agent with `call_id` by setting approve to true.

MCP integration is a key step toward creating richer, more context-aware AI agents. As the MCP ecosystem grows, you'll have even more opportunities to bring specialized tools into your workflows and deliver smarter, more dynamic solutions.

Exercise - Connect MCP tools to Azure AI Agents

Connect AI agents to tools using Model Context Protocol (MCP)

In this exercise, you'll create an agent that can connect to an MCP server and automatically discover callable functions.

You'll build a simple inventory assessment agent for a cosmetics retailer. Using the MCP server, the agent will be able to retrieve information about the inventory and make restock or clearance suggestions.

Module assessment

1. What role does the MCP server play in the MCP agent tool integration? **Hosts tool definitions and makes them available for discovery by the client**.
2. How does an MCP client retrieve available tools from the MCP server? **By calling `session.list_tools()` to get the current tool catalog.**.
3. Why should MCP tools be wrapped in async functions on the client-side? **To enable asynchronous invocation so the agent can call tools without blocking**.

Summary

In this module, you learned how to **integrate external tools with Azure AI Foundry Agent Service using the Model Context Protocol (MCP)**.

By connecting your agent to an MCP server, you can **dynamically discover and register tools at runtime without hardcoding APIs or redeploying your agent**. **Using an MCP client, you generated function wrappers from discovered tools and connected them directly to your agent**. This integration allows your agent to adapt to evolving toolsets, and create more flexible AI solutions that can grow alongside your applications.

> Tip: To learn more about MCP, see the Model Context Protocol User Guide and AI Agents MCP Integration. To learn more about using Azure AI Foundry Agent Service with MCP, visit Connect to Model Context Protocol servers.

Develop an AI agent with Microsoft Agent Framework

This module provides engineers with the skills to begin **building Azure AI Foundry Agent Service agents with Microsoft Agent Framework**.

Learning objectives

By the end of this module, you'll be able to:

- Use **Microsoft Agent Framework** to connect to an Azure AI Foundry project.
- Create Azure AI Foundry Agent Service agents using the **Microsoft Agent Framework SDK**.
- Integrate plugin functions with your AI agent.

Introduction

AI agents are transforming how applications interact with users and automate tasks. Unlike traditional programs, AI agents use generative AI to interpret data, make decisions, and complete tasks with minimal human intervention. These agents use large language models to streamline complex workflows, making them ideal for automating business processes.

Developers can build AI agents using different tools, including the **Microsoft Agent Framework**. This open-source SDK simplifies the integration of AI models into applications. The Microsoft Agent Framework **supports different types of agents from multiple providers**, including **Azure AI Foundry, Azure OpenAI, OpenAI, Microsoft Copilot Studio, and Anthropic agents**. This module focuses on Azure AI Foundry Agents, which provide enterprise-grade capabilities using the Azure AI Foundry Agent Service.

Azure AI Foundry Agent Service is a fully managed service that enables developers to securely build, deploy, and scale high-quality extensible AI agents. Using the Foundry Agent Service, developers don't need to manage the underlying compute or storage resources. The Microsoft Agent Framework enables developers to quickly build agents on the Foundry Agent Service, supporting natural language processing and providing access to built-in tools in just a few lines of code.

While Foundry Agent Service provides a powerful foundation for building AI agents, the **Microsoft Agent Framework offers more flexibility and scalability**. If your solution requires multiple types of agents, using the Microsoft Agent Framework ensures consistency across your implementation. Finally, if you're planning to develop multi-agent solutions, the framework's workflow orchestration features allow you to coordinate collaborative agents efficiently—a topic covered in more detail in a later module.

Suppose you need to develop an AI agent that automatically formats and emails expense reports for employees. Your AI agent can extract data from submitted expense reports, format them correctly, and send them to the appropriate recipients when you use the Microsoft Agent Framework. The tools and functions feature allows your AI agent to interact with APIs, retrieve necessary data, and complete tasks.

In this module, you learn about the core features of the **Microsoft Agent Framework SDK**. You also learn how to create your own AI agents and extend their capabilities with tool functions.

After completing this module, you're now able to:

- Use the Microsoft Agent Framework to connect to an Azure AI Foundry project.
- Create Azure AI Foundry agents using the Microsoft Agent Framework.
- Integrate tool functions with your AI agent.

Understand Microsoft Agent Framework AI agents

An AI agent is a program that uses generative AI to interpret data, make decisions, and perform tasks on behalf of users or other applications. AI agents rely on large language models to perform their tasks. Unlike conventional programs, AI agents can function autonomously, handling complex workflows and automating processes without requiring continuous human oversight.

AI Agents can be developed using many different tools and platforms, including the **Microsoft Agent Framework**. The Microsoft Agent Framework is an open-source SDK that enables developers to easily integrate the latest AI models into their applications. **This framework provides a comprehensive foundation for creating functional agents that can use natural language processing to complete tasks and collaborate with other agents**.

Microsoft Agent Framework core components

The Microsoft Agent Framework offers different components that can be used individually or combined.

- **Chat clients** - provide abstractions for connecting to AI services from different providers under a common interface. Supported providers include Azure OpenAI, OpenAI, Anthropic, and more through the `BaseChatClient` abstraction.
- **Function tools** - containers for custom functions that extend agent capabilities. Agents can automatically invoke functions to integrate with external APIs and services.
- **Built-in tools** - prebuilt capabilities including **Code Interpreter** for Python execution, **File Search** for document analysis, and **Web Search** for internet access.
- **Conversation management** - structured message system with roles (USER, ASSISTANT, SYSTEM, TOOL) and `AgentThread` for persistent conversation context across interactions.
- **Workflow orchestration** - supports sequential workflows, concurrent execution, group chat, and handoff patterns for complex multi-agent collaboration.

The Microsoft Agent Framework helps streamline the creation of agents and allows multiple agents to work together in conversations while including human input. The framework supports different types of agents from multiple providers, including Azure AI Foundry, Azure OpenAI, OpenAI, Microsoft Copilot Studio, and Anthropic agents.

What is an Azure AI Foundry Agent?

Azure AI Foundry Agents provide enterprise-level capabilities using the Azure AI Foundry Agent Service. These agents offer advanced features for complex enterprise scenarios. Key benefits include:

- **Enterprise-level capabilities** – Built for Azure environments with advanced AI features including **code interpreter, function tools integration, and Model Context Protocol (MCP)** support.
- **Automatic tool invocation** – Agents can automatically call and execute tools, integrating seamlessly with Azure AI Search, Azure Functions, and other Azure services.
- **Thread and conversation management** – Provides built-in mechanisms for managing persistent conversation states across sessions, ensuring smooth multi-agent interactions.
- **Secure enterprise integration** – Enables secure and compliant AI agent development with **Azure CLI authentication, RBAC, and customizable storage** options.

When you use Azure AI Foundry Agents, you get the full power of enterprise Azure capabilities combined with the features of the Microsoft Agent Framework. These features can help you create robust AI-driven workflows that can scale efficiently across business applications.

Agent framework core concepts

- **BaseAgent** - the foundation for all agents with consistent methods, providing a unified interface across all agent types.
- **Agent threads** - manage persistent conversation context and store conversation history across sessions using the `AgentThread` class.
- **Chat messages** - organized structure for agent communication using role-based messaging (**USER, ASSISTANT, SYSTEM, TOOL**) that enables smooth communication and integration.
- **Workflow orchestration** - supports sequential workflows, running multiple agents in parallel, group conversations between agents, and transferring control between specialized agents.
- **Multi-modal support** - allows agents to work with text, images, and structured outputs, including vision capabilities and type-safe response generation.
- **Function tools** - let you add custom capabilities to agents by including custom functions with automatic schema generation from Python functions.
- **Authentication methods** - supports multiple authentication methods including **Azure CLI credentials, API keys, MSAL for Microsoft business authentication, and role-based access control**.

This framework supports autonomous, multi-agent AI behaviors while maintaining a flexible architecture that lets you mix and match agents, tools, and workflows as needed. The design lets you switch between OpenAI, Azure OpenAI, Anthropic, and other providers without changing your code, making it easy to build AI systems—from simple chatbots to complex business solutions.

Create an Azure AI agent with Microsoft Agent Framework

Azure AI Foundry Agent is a specialized agent within the Microsoft Agent Framework, designed to provide enterprise-level conversational capabilities with seamless tool integration. It automatically handles tool calling, so you don't need to manually parse and invoke functions. The agent also securely manages conversation history using threads, which reduces the work of maintaining state. **The Azure AI Foundry Agent supports many built-in tools**, including **code interpreter, file search, and web search**. It also provides integration capabilities for Azure AI Search, Azure Functions, and other Azure services.

Creating an AzureAIAgent

An Azure AI Foundry Agent includes all the core capabilities you typically need for enterprise AI applications, like function execution, planning, and memory access. This agent acts as a self-contained runtime with enterprise-level features.

To use an Azure AI Foundry Agent:

1. Create an Azure AI Foundry project.
2. Add the project connection string to your Microsoft Agent Framework application code.
3. Set up authentication credentials.
4. Create a `ChatAgent` with an `AzureAIAgentClient`.
5. Define tools and instructions for your agent.

Here's the code that shows how to create an Azure AI Foundry Agent:

```
from agent_framework import AgentThread, ChatAgent
from agent_framework.azure import AzureAIAgentClient
from azure.identity.aio import AzureCliCredential

def get_weather(
    location: Annotated[str, Field(description="The location to get the weather for.")],
) -> str:
    """Get the weather for a given location."""
    return f"The weather in {location} is sunny with a high of 25°C."

# Create a ChatAgent with Azure AI client {#create-a-chatagent-with-azure-ai-client }
async with (
    AzureCliCredential() as credential,
    ChatAgent(
        chat_client=AzureAIAgentClient(async_credential=credential),
        instructions="You are a helpful weather agent.",
        tools=get_weather,
    ) as agent,
):
    # Agent is now ready to use
```

Once your agent is created, you can **create a thread to interact with your agent** and get responses to your questions. For example:

```
# Create the agent thread for ongoing conversation {#create-the-agent-thread-for-ongoing-convers
thread = agent.get_new_thread()

# Ask questions and get responses {#ask-questions-and-get-responses }
first_query = "What's the weather like in Seattle?"
print(f"User: {first_query}")
first_result = await agent.run(first_query, thread=thread)
print(f"Agent: {first_result.text}")
```

Azure AI Foundry Agent key components

The Microsoft Agent Framework Azure AI Foundry Agent uses the following components to work:

- **AzureAIAgentClient** - **manages the connection to your Azure AI Foundry project**. This client lets you access the services and models associated with your project and provides enterprise-level authentication and security features.
- **ChatAgent** - the **main agent** class that combines the client, instructions, and tools to create a working AI agent that can handle conversations and complete tasks.
- **AgentThread** - automatically keeps track of **conversation history between agents and users**, and manages the conversation state. You can create new threads or reuse existing ones to maintain context across interactions.
- **Tools integration** - support for custom functions that extend agent capabilities. Functions are automatically registered and can be called by agents to connect with external APIs and services.
- **Authentication credentials** - supports **Azure CLI credentials, service principal authentication, and other Azure identity options** for secure access to Azure AI services.
- **Thread management** - provides flexible options for thread creation, including automatic thread creation for simple scenarios and explicit thread management for ongoing conversations.

These components work together to let you create enterprise-level agents with instructions to define their purpose and get responses from AI models while maintaining security, scalability, and conversation context for business applications.

Add tools to Azure AI agent

In the Microsoft Agent Framework, tools allow your AI agent to use existing APIs and services to perform tasks it couldn't do on its own. Tools work through function calling, allowing AI to automatically request and use specific functions. The framework routes the request to the appropriate function in

your codebase and returns the results back to the large language model (LLM) so it can generate a final response.

To enable automatic function calling, tools need to provide details that describe how they work. The function's input, output, and purpose should be described in a way that the AI can understand, otherwise, the AI can't call the function correctly.

How to use tools with Azure AI Foundry Agent

The Microsoft Agent Framework supports both **custom function tools** and **built-in tools** that are ready to use out of the box.

Built-in tools

Azure AI Foundry Agents come with several built-in tools that you can use immediately:

- **Code Interpreter** - executes Python code for calculations, data analysis, and more
- **File Search** - searches through and analyzes documents
- **Web Search** - retrieves information from the internet

These tools are automatically available and don't require any extra setup.

Custom function tools

When creating custom tools for your Azure AI Foundry Agent, you need to understand several key concepts:

1. Function definition and annotations
   Create your tool by defining a regular Python function with proper type annotations. Use Annotated and Field from Pydantic to provide detailed descriptions that help the AI understand the function's purpose and how to use its parameters. The more descriptive your annotations, the better the AI can understand when and how to call your function.
2. Adding tools to your agent
   Pass your custom functions to the `ChatAgent` during creation using the tools parameter. You can add a single function or a list of multiple functions. The framework automatically registers these functions and makes them available for the AI to call.
3. Tool invocation through conversation
   Once your tools are registered with the agent, you don't need to manually invoke them. Instead, ask the agent questions or give it tasks that would naturally require your tool's functionality. The AI automatically determines when to call your tools based on the conversation context and the tool descriptions you provided.

4. Multiple tools and orchestration

   You can **add multiple tools to a single agent**, and **the AI automatically chooses which tool to use based on the user's request**. The framework handles the orchestration, calling the appropriate functions and combining their results to provide a comprehensive response.

Best practices for tool development

- **Clear descriptions**: Write clear, detailed descriptions for your functions and parameters to help the AI understand their purpose
- **Type annotations**: Use proper Python type hints to specify expected input and output types
- **Error handling**: Implement appropriate error handling in your tool functions to gracefully handle unexpected inputs
- **Return meaningful data**: Ensure your functions return data that the AI can effectively use in its responses
- **Keep functions focused**: **Design each tool to handle a specific task** rather than trying to do too many things in one function

By following these concepts, you can extend your Azure AI Foundry Agent with both built-in and custom tools, allowing it to interact with APIs and perform advanced tasks. This approach makes your AI more powerful and capable of handling real-world applications efficiently.

Exercise - Develop an Azure AI agent with the Microsoft Agent Framework SDK

Now you're ready to build an agent with the Microsoft Agent Framework. In this exercise, you use the Microsoft Agent Framework SDK to create an AI agent that creates an expense claim email.

Develop an Azure AI chat agent with the Microsoft Agent Framework SDK

In this exercise, you'll use Azure AI Agent Service and Microsoft Agent Framework to create an AI agent that processes expense claims.

Knowledge check

1. What are the key steps to create an AzureAIAgent? Create an `AzureAIAgentClient`, define a `ChatAgent` with instructions and tools, and create an `AgentThread` for conversations.
2. Which component in the Agent Framework manages conversation state and stores messages? `AgentThread`
3. Which step is necessary to enable an AzureAIAgent to use a plugin? Create Python functions with proper type annotations and descriptions, then pass them to the `ChatAgent`'s tools parameter.

Summary

In this module, you learned how the Microsoft Agent Framework enables developers to build AI agents. You learned about the components and core concepts of the **Microsoft Agent Framework**. You also learned how to create custom tools to extend your agent's capabilities. By applying these concepts and skills, you can use the Microsoft Agent Framework to create dynamic, adaptable AI solutions that enhance user interactions and automate complex tasks.

Orchestrate a multi-agent solution using the Microsoft Agent Framework

Learn how to use the **Microsoft Agent Framework SDK** to develop your own AI agents that can collaborate for a multi-agent solution.

Learning objectives

By the end of this module, you'll be able to:

- Build AI agents using the Microsoft Agent Framework SDK
- Understand **how and when to use different orchestration patterns**
- Develop multi-agent solutions

Introduction

AI agents offer a powerful combination of technologies, able to complete tasks with the use of generative AI. However, in some situations, the task required might be larger than is realistic for a single agent. For those scenarios, consider a **multi-agent solution**. A multi-agent solution allows agents to collaborate within the same conversation.

Imagine you're trying to address common DevOps challenges such as monitoring application performance, identifying issues, and deploying fixes. A multi-agent system could consist of four specialized agents working collaboratively:

- The **Monitoring Agent** continuously ingests logs and metrics, detects anomalies using natural language processing (NLP), and triggers alerts when issues arise.
- The **Root Cause Analysis Agent** then correlates these anomalies with recent system changes, using machine learning models or predefined rules to pinpoint the root cause of the problem.
- Once the root cause is identified, the **Automated Deployment Agent** takes over to implement fixes or roll back problematic changes by interacting with CI/CD pipelines and executing deployment scripts.
- Finally, the **Reporting Agent** generates detailed reports summarizing the anomalies, root causes, and resolutions, and notifies stakeholders via email or other communication channels.

This modular, scalable, and intelligent multi-agent system streamlines the DevOps process. The agents collaborate to reduce manual intervention and improve efficiency while ensuring timely communication and resolution of issues.

In this module, you'll explore how to use the powerful capabilities of the Microsoft Agent Framework to design and orchestrate intelligent agents that work collaboratively to solve complex problems. You'll also learn about the **different types of orchestration patterns** available, and use the Microsoft Agent Framework to develop your own AI agents that can collaborate for a multi-agent solution.

After completing this module, you'll be able to:

- Build AI agents using the **Microsoft Agent Framework SDK**
- Use tools and plugins with your AI agents
- Understand different types of orchestration patterns
- Develop multi-agent solutions

Understand the Microsoft Agent Framework

The Microsoft Agent Framework is an open-source SDK that enables developers to integrate AI models into their applications. This framework provides comprehensive support for creating AI-powered agents that can work independently or collaborate with other agents to accomplish complex tasks.

What is the Microsoft Agent Framework?

The Microsoft Agent Framework is designed to help developers build AI-powered agents that can process user inputs, make decisions, and execute tasks autonomously by leveraging large language models and traditional programming logic. The framework provides structured components for defining AI-driven workflows, enabling agents to interact with users, APIs, and external services seamlessly.

Core concepts

The Microsoft Agent Framework provides a flexible architecture with the following key components:

- **Agents**: Agents are intelligent, AI-driven entities capable of reasoning and executing tasks. They use large language models, tools, and conversation history to make decisions dynamically and respond to user needs.

- **Agent orchestration**: Multiple agents can collaborate towards a common goal using different **orchestration patterns**. The Microsoft Agent Framework supports several orchestration patterns with a unified interface for construction and invocation, allowing you to easily switch between patterns without rewriting your agent logic.

The framework includes several core features that power agent functionality:

- **Chat clients**: Chat clients provide abstractions for connecting to AI services from different providers under a common interface. Supported providers include Azure OpenAI, OpenAI, Anthropic, and more through the `BaseChatClient` abstraction.
- **Tools and function integration**: Tools enable agents to extend their capabilities through custom functions and built-in services. Agents can automatically invoke tools to integrate with external APIs, execute code, search files, or access web information. The framework supports both **custom function tools and built-in tools** like *Code Interpreter, File Search, and Web Search*.
- **Conversation management**: Agents can maintain conversation history across multiple interactions using `AgentThread`, allowing them to track previous interactions and adapt responses accordingly. The structured message system uses **roles (USER, ASSISTANT, SYSTEM, TOOL)** for persistent conversation context.

Types of agents

The Microsoft Agent Framework supports several different types of agents from multiple providers:

- **Azure AI Foundry Agent** - a specialized agent within the Microsoft Agent Framework designed to provide enterprise-grade conversational capabilities with seamless tool integration. It automatically handles tool calling and securely **manages conversation history using threads**, reducing the overhead of maintaining state. Azure AI Foundry Agents support built-in tools and provide integration capabilities for Azure AI Search, Azure Functions, and other Azure services.
- **ChatAgent**: designed for general conversation and task completion interfaces. The `ChatAgent` type provides natural language processing, contextual understanding, and dialogue management with support for custom tools and instructions.
- **OpenAI Assistant Agent**: designed for advanced capabilities using **OpenAI's Assistant API**. This agent type supports goal-driven interactions with features like code interpretation and file search through the OpenAI platform.
- **Anthropic Agent**: provides access to **Anthropic's Claude models** with the framework's unified interface, supporting advanced reasoning and conversation capabilities.

Why you should use the Semantic Kernel Agent Framework

The Microsoft Agent Framework offers a robust platform for building intelligent, autonomous, and collaborative AI agents. **The framework can integrate agents from multiple sources, including Azure AI Foundry Agent Service, and supports both multi-agent collaboration and human-agent interaction**. Agents can work together to orchestrate sophisticated workflows, where each agent specializes in a specific task, such as data collection, analysis, or decision-making. The framework also facilitates human-in-the-loop processes, enabling agents to augment human decision-making by providing insights or automating repetitive tasks. The provider-agnostic design allows you to switch between different AI providers without changing your code, making it suitable for building adaptable AI systems from simple chatbots to complex enterprise solutions.

Understand agent orchestration

The **Microsoft Agent Framework SDK's agent orchestration framework** makes it possible to design, manage, and scale complex multi-agent workflows without having to manually handle the details of agent coordination. Instead of relying on a single agent to manage every aspect of a task, you can combine multiple specialized agents. Each agent with a unique role or area of expertise can collaborate to create systems that are more robust, adaptive, and capable of solving real-world problems collaboratively.

By orchestrating agents together, you can take on tasks that would be too complex for a single agent—from running parallel analyses, to building multi-stage processing pipelines, to managing dynamic, context-driven handoffs between experts.

Why multi-agent orchestration matters

Single-agent systems are often limited in scope, constrained by one set of instructions or a single model prompt. Multi-agent orchestration addresses this limitation by allowing you to:

- **Assign** distinct skills, responsibilities, or perspectives to each agent.
- **Combine outputs** from multiple agents to improve decision-making and accuracy.
- **Coordinate steps in a workflow** so each agent's work builds on the last.
- **Dynamically route control** between agents based on context or rules.

This approach opens the door to more flexible, efficient, and scalable solutions, especially for real-world applications that require collaboration, specialization, or redundancy.

Supported orchestration patterns

Microsoft Agent Framework provides several orchestration patterns directly in the SDK, each offering a different approach to coordinating agents. These patterns are designed to be technology-agnostic so you can adapt them to your own domain and integrate them into your existing systems.

- **Concurrent orchestration** - Broadcast the same task to multiple agents at once and collect their results independently. Useful for **parallel analysis**, independent subtasks, or ensemble decision making.
- **Sequential orchestration** - Pass the output from one agent to the next in a fixed order. Ideal for **step-by-step** workflows, pipelines, and progressive refinement.
- **Handoff orchestration** - **Dynamically transfer control** between agents based on context or rules. Great for escalation, fallback, and expert routing where one agent works at a time.
- **Group chat orchestration** - Coordinate a shared conversation among multiple agents (and optionally a human), managed by a chat manager that chooses who speaks next. Best for **brainstorming**, collaborative problem solving, and building consensus.
- **Magentic orchestration** - A manager-driven approach that plans, delegates, and adapts across specialized agents. Suited to **complex, open-ended problems** where the solution path evolves.

A unified orchestration workflow

Regardless of which orchestration pattern you choose, the Microsoft Agent Framework SDK provides a consistent, developer-friendly interface for building and running them. The typical flow looks like this:

1. **Define your agents** and describe their capabilities.
2. **Select and create an orchestration pattern**, optionally adding a manager agent if needed.
3. **Optionally configure callbacks or transforms** for custom input and output handling.
4. **Start a runtime** to manage execution.
5. **Invoke the orchestration** with your task.
6. **Retrieve results** in an **asynchronous**, nonblocking way.

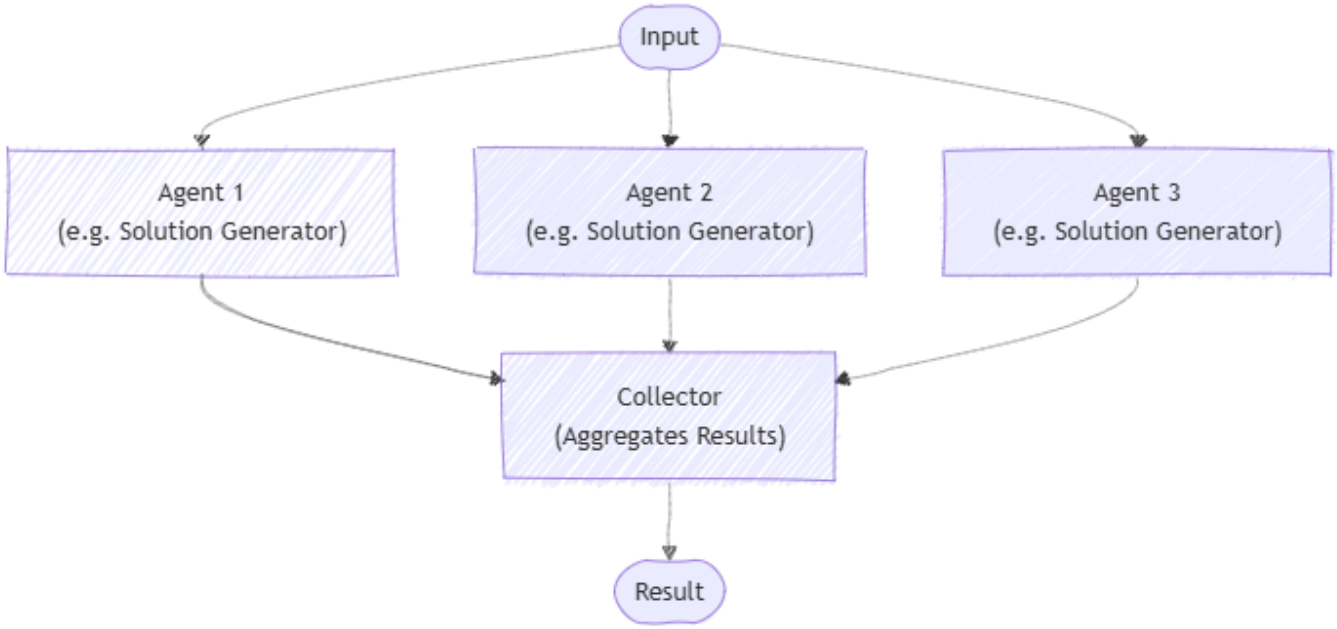
Because all patterns share the same core interface, you can easily experiment with different orchestration strategies without rewriting agent logic or learning new APIs. The SDK abstracts the complexity of agent communication, coordination, and result aggregation so you can focus on designing workflows that deliver results.

Multi-agent orchestration in the Microsoft Agent Framework SDK provides a flexible, scalable way to build intelligent systems that combine the strengths of multiple specialized agents. With built-in orchestration patterns, a unified development model, and runtime features for managing execution, you can quickly prototype, refine, and deploy collaborative AI workflows. The framework provides the

tools to turn multiple agents into a cohesive problem-solving team, whether you're running parallel processes, sequential workflows, or dynamic conversations.

Use concurrent orchestration

Concurrent orchestration lets multiple agents work on the same task at the same time. **Each agent handles the task independently, and then their outputs are gathered and combined**. This method works especially well when you want diverse approaches or solutions, like during **brainstorming, group decision-making, or voting**.



This pattern is useful when you need different approaches or ideas to solve the same problem. Instead of having agents work one after another, they **all work at the same time**. This speeds up the process and covers the problem from many angles.

Usually, the results from each agent are combined to create a final answer, but this isn’t always necessary. Each agent can also produce its own separate result, like calling tools to complete tasks or updating different data stores independently.

Agents work on their own and don't share results with each other. However, an agent can call other AI agents by running its own orchestration as part of its process. Agents need to know which other agents are available to work on tasks. This pattern allows you to either call all registered agents every time or choose which agents to run based on the specific task.

When to use concurrent orchestration

You may want to consider using the concurrent orchestration pattern in these situations:

- When tasks can run at the same time, either by using a fixed group of agents or by selecting AI agents dynamically based on what the task needs.
- When the task benefits from different specialized skills or approaches (for example, technical, business, or creative) that all work independently but contribute to solving the same problem.

This kind of teamwork is common in multi-agent decision-making methods such as:

- Brainstorming ideas
- Combining different reasoning methods (ensemble reasoning)
- Making decisions based on voting or consensus (quorum)
- Handling tasks where speed matters and running agents in parallel cuts down wait time

When to avoid concurrent orchestration

You may want to avoid using the concurrent orchestration pattern in the following scenarios:

- Agents need to build on each other's work or **depend on shared context** in a specific order.
- The task requires a **strict sequence** of steps or predictable, repeatable results.
- Resource limits, like model usage quotas, make running agents in parallel inefficient or impossible.
- Agents can't reliably coordinate changes to shared data or external systems while running at the same time.
- There's **no clear way to resolve conflicts or contradictions** between results from different agents.
- Combining results is too complicated or ends up lowering the overall quality.

Implement concurrent orchestration

Implement the concurrent orchestration pattern with the Microsoft Agent Framework:

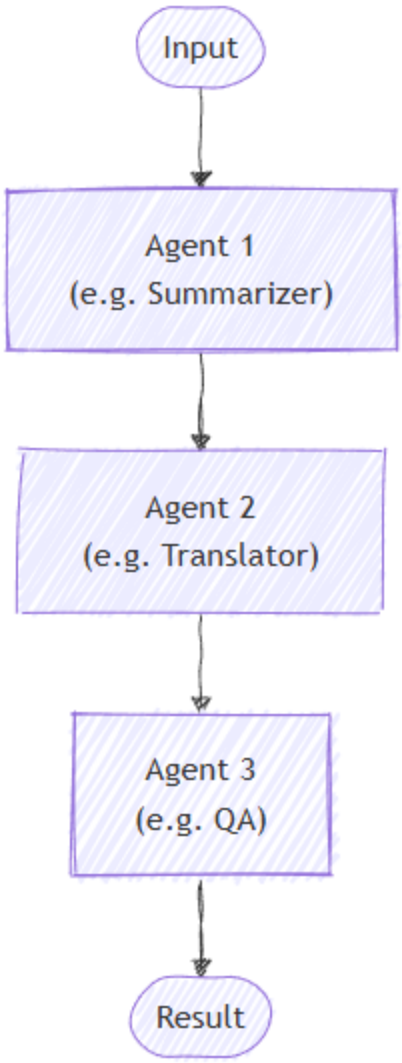
1. **Create your chat client**: Set up a chat client (for example, `AzureOpenAIChatClient`) with appropriate credentials to connect to your AI service provider.
2. **Define your agents**: Create agent instances using the chat client's `create_agent` method. Each agent should have specific instructions and a name that defines its role and expertise area.
3. **Build the concurrent workflow**: Use the `ConcurrentBuilder` class to create a workflow that can run multiple agents in parallel. Add your agent instances as participants using the `participants()` method, then call `build()` to create the workflow.

4. **Run the workflow**: Call the workflow's `run` method with the task or input you want the agents to work on. The workflow runs all agents concurrently and returns events containing the results.
5. **Process the results**: Extract the outputs from the workflow events using `get_outputs()`. The results contain the combined conversations from all agents, with each agent's response included in the final output.
6. **Handle the aggregated responses**: Process the aggregated messages from all agents. **Each message includes the author name and content**, allowing you to identify which agent provided each response.

Concurrent orchestration is a powerful pattern for using multiple AI agents **simultaneously**, enabling faster and more diverse problem-solving. By running agents in parallel, you can explore different approaches at once, improve efficiency, and gain richer insights. However, it's important to choose this pattern when tasks can truly run independently and to be mindful of resource constraints and coordination challenges. When implemented thoughtfully with the Microsoft Agent Framework SDK, concurrent orchestration can greatly enhance your AI workflows and decision-making processes.

Use sequential orchestration

In sequential orchestration, agents are arranged in a **pipeline** where each agent processes the task one after another. **The output from one agent becomes the input for the next**. This pattern is ideal for workflows where each step depends on the previous one, such as document review, data transformation pipelines, or multi-stage reasoning.



Sequential orchestration works best for tasks that need to be done **step-by-step**, with each step improving on the last. The order in which agents run is fixed and decided beforehand, and agents don't decide what happens next.

When to use sequential orchestration

Consider using the sequential orchestration pattern when your workflow has:

- Processes made up of **multiple steps** that must happen in a **specific order**, where each step relies on the one before it.
- Data workflows where each stage adds something important that the next stage needs to work properly.
- Tasks where stages can't be done at the same time and must run one after another.
- Situations that require **gradual improvements**, like drafting, reviewing, and polishing content.

- Systems where you know how each agent performs and can handle delays or failures in any step without stopping the whole process.

When to avoid sequential orchestration

Avoid this pattern when:

- **Stages can be run independently and in parallel** without affecting quality.
- A single agent can perform the entire task effectively.
- Early stages may fail or produce poor output, and there's no way to stop or correct downstream processing based on errors.
- Agents need to collaborate dynamically rather than hand off work sequentially.
- The workflow requires iteration, backtracking, or dynamic routing based on intermediate results.

Implement sequential orchestration

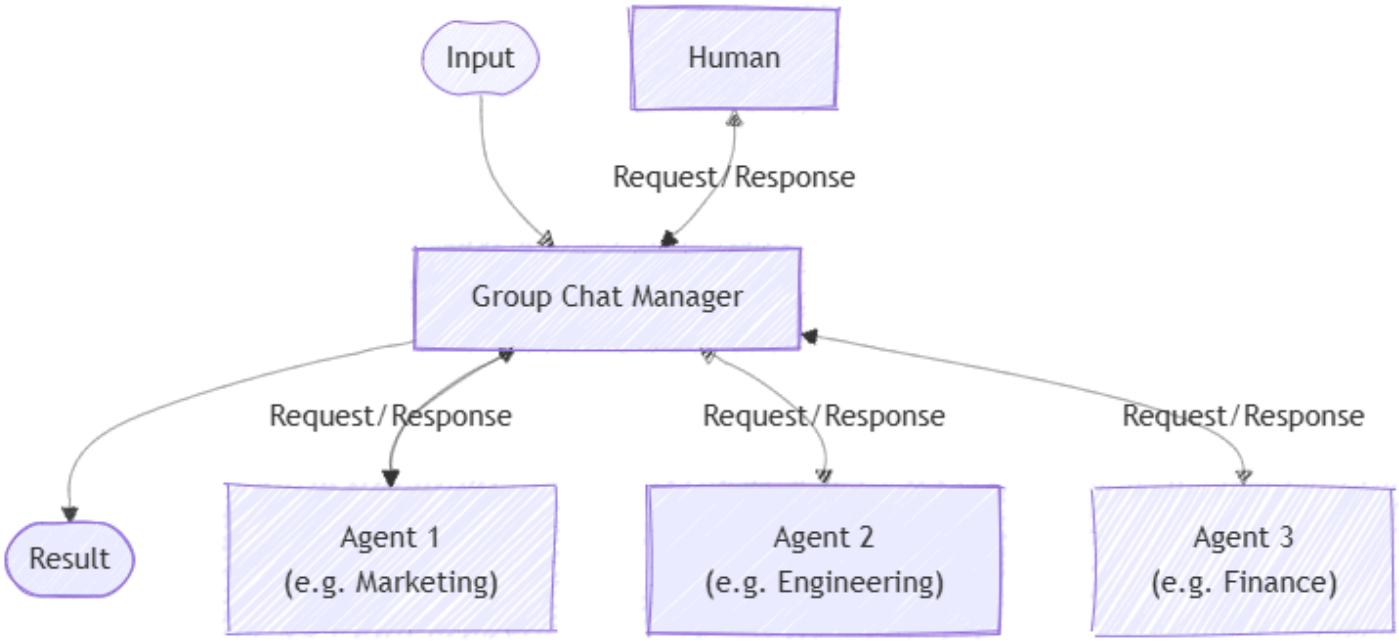
Implement the sequential orchestration pattern with the Microsoft Agent Framework SDK:

1. **Create your chat client**: Set up a chat client (for example, `AzureOpenAIChatClient`) with appropriate credentials to connect to your AI service provider.
2. **Define your agents**: Create agent instances using the chat client's `create_agent` method. Each agent should have specific instructions and a name that defines its role and expertise area in the pipeline.
3. **Build the sequential workflow**: Use the `SequentialBuilder` class to create a workflow that executes agents one after another. Add your agent instances as participants using the `participants()` method, then call `build()` to create the workflow.
4. **Run the workflow**: Call the workflow's `run_stream` method with the task or input you want the agents to work on. The workflow processes the task through all agents sequentially, with each agent's output becoming input for the next.
5. **Process the workflow events**: Iterate through the workflow events using an async loop. Look for `WorkflowOutputEvent` instances, which contain the results from the sequential processing.
6. **Extract the final conversation**: Collect the final conversation from the workflow outputs. The result contains the complete conversation history showing how each agent in the sequence contributed to the final outcome.

Sequential orchestration is ideal when your task requires clear, ordered steps where each agent builds on the previous one's output. This pattern helps improve output quality through stepwise refinement and ensures predictable workflows. When applied thoughtfully with the Microsoft Agent Framework SDK, it enables powerful multi-agent pipelines for complex tasks like content creation, data processing, and more.

Use group chat orchestration

Group chat orchestration models a collaborative conversation among multiple AI agents, and optionally a human participant. A central chat manager controls the flow, deciding which agent responds next and when to request human input. This pattern is useful for simulating meetings, debates, or collaborative problem-solving.



The group chat pattern works well for scenarios where **group discussion** or **iterative collaboration** is key to reaching decisions. It supports different interaction styles, from free-flowing ideation to formal workflows with defined roles and approval steps. Group chat orchestration is also great for **human-in-the-loop** setups where a human may guide or intervene in the conversation. Typically, agents in this pattern don't directly change running systems—they mainly contribute to the conversation.

When to use group chat orchestration

Consider using group chat orchestration when your scenario involves:

- Spontaneous or guided collaboration among agents (and possibly humans)
- Iterative maker-checker loops where agents take turns creating and reviewing
- **Real-time human oversight or participation**
- Transparent and auditable conversations since **all output is collected in a single thread**

Common scenarios include:

- Creative brainstorming where agents build on each other's ideas
- Decision-making that benefits from debate and consensus
- Complex problems requiring cross-disciplinary dialogue
- Quality control and validation requiring multiple expert perspectives
- Content workflows with clear separation between creation and review

When to avoid group chat orchestration

Avoid this pattern when:

- Simple task delegation or straightforward linear pipelines suffice
- Real-time speed requirements make discussion overhead impractical
- Hierarchical or deterministic workflows are needed without discussion
- The chat manager can't clearly determine when the task is complete
- Managing conversation flow becomes too complex, especially with many agents (limit to three or fewer for easier control)

Maker-checker loops

A common special case is the maker-checker loop. Here, **one agent (the maker) proposes content or solutions, and another agent (the checker) reviews and critiques them**. The checker can send feedback back to the maker, and this cycle repeats until the result is satisfactory. **This process requires a turn-based sequence managed by the chat manager**.

Implement group chat orchestration

Implement the group chat orchestration pattern with the Microsoft Agent Framework SDK:

1. **Create your chat client**: Set up a chat client (for example, `AzureOpenAIChatClient`) with appropriate credentials to connect to your AI service provider.
2. **Define your agents**: Create agent instances using the chat client's `create_agent` method. Each agent should have **specific instructions and a name** that defines its role and expertise area.
3. **Build the group chat workflow**: Use the `GroupChatBuilder` class to create a workflow that can run multiple agents in parallel. Add your agent instances as participants using the `participants()` method, then call `build()` to create the workflow.
4. **Run the workflow**: Call the workflow's `run` method with the task or input you want the agents to work on. The workflow runs all agents concurrently and returns events containing the results.
5. **Process the results**: Extract the outputs from the workflow events using `get_outputs()`. The results contain the combined conversations from all agents, with each agent's response included in the final output.

6. **Handle the aggregated responses**: Process the aggregated messages from all agents. Each message includes the author name and content, allowing you to identify which agent provided each response.

Customizing the group chat manager

You can create a custom group chat manager by extending the base `GroupChatManager` class. This approach lets you control:

- How conversation results are filtered or summarized
- How the next agent is selected
- When to request user input
- When to terminate the conversation

Custom managers let you implement specialized logic tailored to your use case.

Group chat manager call order

During each round of the conversation, the chat manager calls methods in this order:

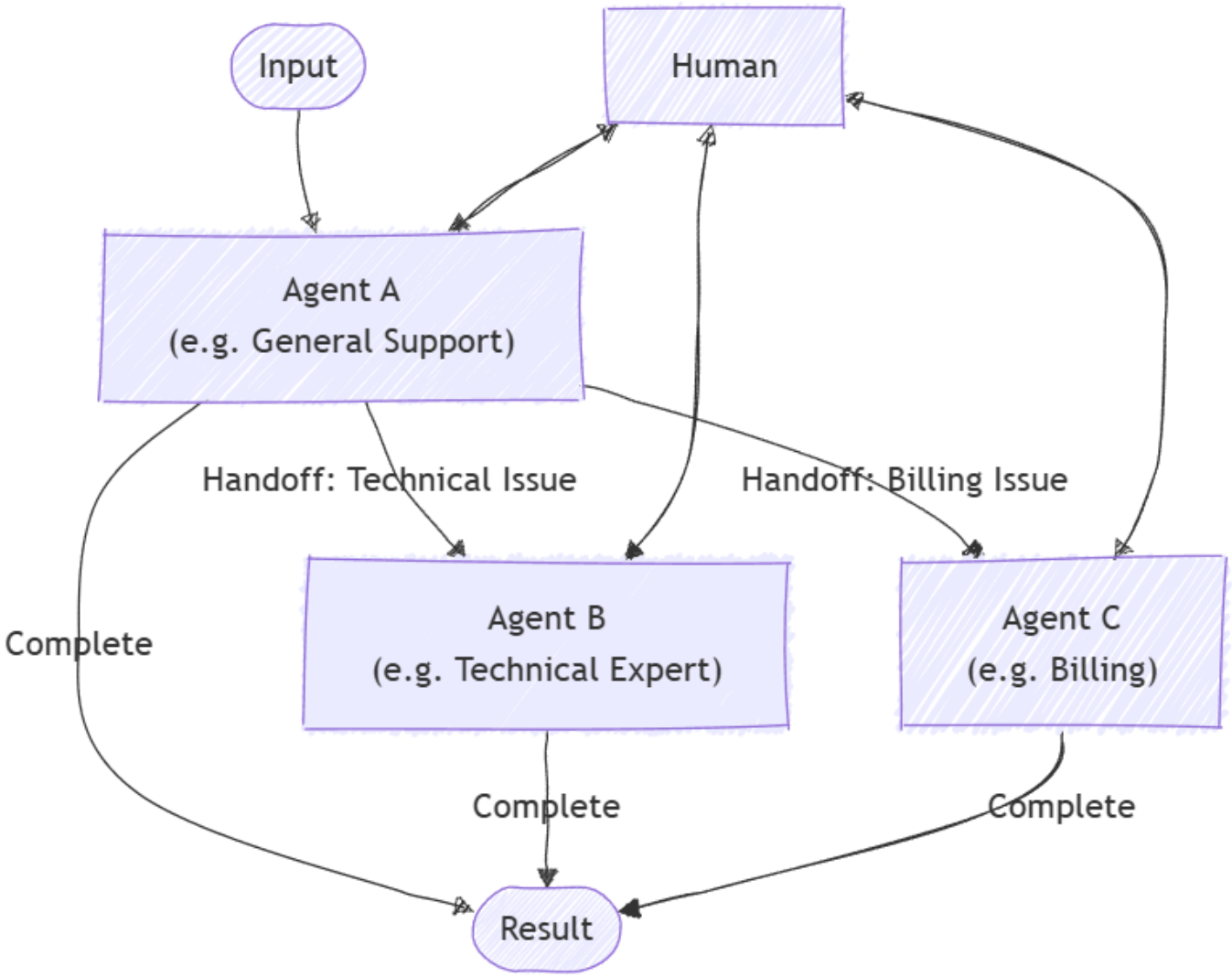
1. `should_request_user_input` - Checks if human input is needed before the next agent responds.
2. `should_terminate` - Determines if the conversation should end (for example, max rounds reached).
3. `filter_results` - If ending, summarizes or processes the final conversation.
4. `select_next_agent` - If continuing, chooses the next agent to speak.

This ensures user input and termination conditions are handled before moving the conversation forward. Override these methods in your custom manager to change behavior.

Group chat orchestration enables multiple AI agents—and optionally humans—to collaborate through guided conversation and iterative feedback. It's ideal for complex tasks that benefit from diverse expertise and dynamic interaction. While it requires careful management, this pattern offers transparency and flexibility in decision-making and creative workflows. The Microsoft Agent Framework SDK makes it easy to implement and customize group chat orchestration for your needs.

Use handoff orchestration

Handoff orchestration lets AI agents transfer control to one another based on the task context or user requests. Each agent can "handoff" the conversation to another agent with the right expertise, **making sure the best-suited agent handles each part of the task**. This pattern is ideal for customer support, expert systems, or any situation where dynamic delegation is needed.



This pattern fits scenarios where the best agent isn't known upfront or where the task requirements become clearer during processing. Unlike parallel patterns, agents work one at a time, fully handing off control from one to the next.

When to use handoff orchestration

You may want to consider using the handoff orchestration pattern in these scenarios:

- Tasks need specialized knowledge or tools, but the number or order of agents can't be determined in advance.
- Expertise requirements emerge dynamically during processing, triggering task routing based on content analysis.
- Multiple-domain problems require different specialists working sequentially.
- You can define clear signals or rules indicating when an agent should transfer control and to whom.

When to avoid handoff orchestration

You may want to avoid using the handoff orchestration pattern in these scenarios:

- The involved agents and their order are known upfront and fixed.
- Task routing is simple and rule-based, not needing dynamic interpretation.
- Poor routing decisions might frustrate users.
- Multiple operations must run at the same time.
- Avoiding infinite handoff loops or excessive bouncing between agents is difficult.

Implementing handoff orchestration

The handoff orchestration pattern can be implemented in the Microsoft Agent Framework SDK using control workflows. In a control workflow, each agent processes the task in sequence, and based on its output, the **workflow decides which agent to call next**. This routing is done using a **switch-case structure** that routes the task to different agents based on classification results.

1. **Set up data models and chat client**:
   - Create your chat client for connecting to AI services
   - Define Pydantic models for AI agents' structured JSON responses
   - Create simple data classes for passing information between workflow steps
   - Configure agents with specific instructions and `response_format` parameter for structured JSON output
2. **Create specialized executor functions**:
   - **Input storage executor** - saves incoming data to shared state and forwards to classification agent
   - **Transformation executor** - converts agent's JSON response into typed routing object
   - **Handler executors** - separate executors for each classification outcome with guard conditions to verify correct message processing
3. **Build routing logic**:
   - Create factory functions that generate condition checkers for each classification value

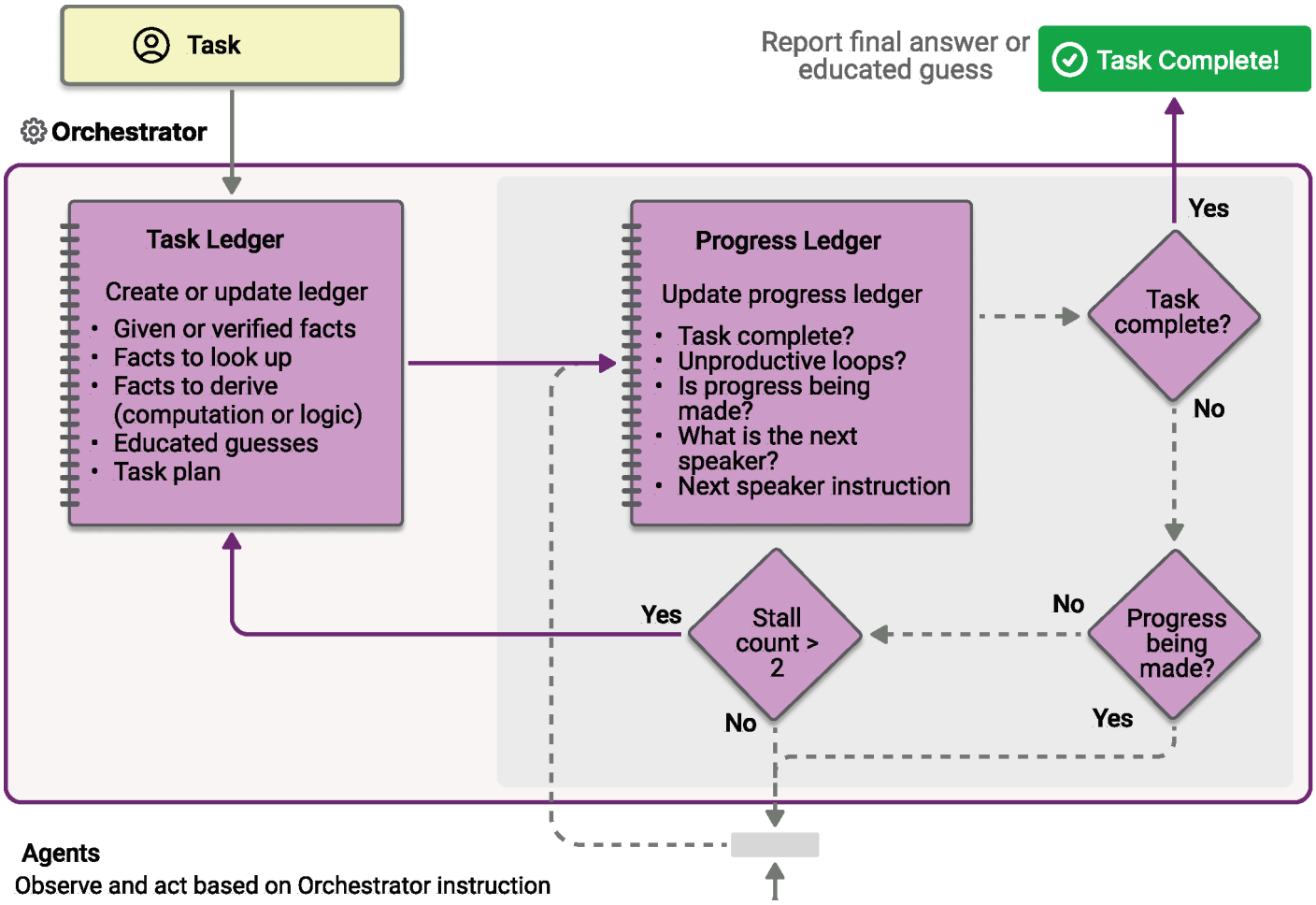
- Design conditions to examine incoming messages and return true for specific classification results
- Use conditions with Case objects in **switch-case edge groups**
- Always include **a Default case as fallback** for unexpected scenarios

4. **Assemble the workflow**:
    - Use WorkflowBuilder to connect executors with regular edges
    - Add switch-case edge group for routing based on classification results
    - Configure workflow to follow first matching case or fall back to default
    - Set up **terminal executor** to yield final output

Handoff orchestration provides a flexible way to route tasks dynamically among specialized AI agents, ensuring that each part of a workflow is handled by the **best-suited expert**. It works well for complex, evolving tasks like **customer support or multi-domain problem solving** where expertise needs change during the conversation. When you use the Microsoft Agent Framework SDK, you can build adaptable systems that seamlessly transfer control between agents—and include human input when needed—for smooth and efficient task completion.

Use Magentic orchestration

Magentic orchestration is a flexible, **general-purpose multi-agent pattern designed for complex, open-ended tasks that require dynamic collaboration**. This pattern uses a dedicated Magentic manager to coordinate a team of specialized agents. The manager decides which agent should act next based on the evolving context, task progress, and agent capabilities.



The Magentic manager maintains a shared context, tracks progress, and adapts the workflow in real time. This approach allows the system to break down complex problems, assign subtasks, and iteratively refine solutions. The process focuses as much on building and documenting the approach as it does on delivering the final solution. A dynamic task ledger is built and refined as the workflow progresses, recording goals, subgoals, and execution plans.

When to use Magentic orchestration

Consider using the Magentic orchestration pattern in these scenarios:

- **The problem is complex or open-ended with no predetermined solution path**.
- Input and feedback from multiple specialized agents are needed to shape a valid solution.
- **The system must generate a documented plan of approach for human review**.
- Agents have tools that can directly interact with external systems and resources.
- A step-by-step, dynamically built execution plan adds value before running the tasks.

When to avoid Magentic orchestration

You may want to avoid this pattern when:

- The solution path is fixed or deterministic.
- There's no need to produce a ledger or plan of approach.
- The task is simple enough for a more lightweight orchestration pattern.
- Speed is the priority, as this method emphasizes planning over fast execution.
- You expect frequent stalls or loops without a clear resolution path.

Implementing Magentic orchestration

Implement the Magentic orchestration pattern with the Microsoft Agent Framework:

1. **Define specialized agents**: Create agent instances (for example, `ChatAgent`) with specific instructions and chat clients. Each agent should have a specialized role and capabilities suited for different aspects of the complex task.
2. **Set up event handling callback**: Define an async callback function to handle different types of events during orchestration, including orchestrator messages, agent streaming updates, agent messages, and final results.
3. **Build the Magentic workflow**: Use the `MagenticBuilder` class to create the orchestration. Add your agent instances as participants, configure the event callback with streaming mode, and set up the standard manager with appropriate parameters like `max round count` and `stall limits`.
4. **Configure the standard manager**: The standard manager coordinates agent collaboration using a chat client for planning and progress tracking. Configure parameters like `maximum round count`, `stall count`, and `reset count` to control the orchestration behavior.
5. **Run the workflow**: Call the workflow's `run_stream` method with your complex task. The workflow dynamically plans, delegates work to appropriate agents, and coordinates their collaboration to solve the problem.
6. **Process workflow events**: Iterate through the workflow events using an async loop. Handle different event types including `WorkflowOutputEvent`, which contains the final results from the orchestration.
7. **Extract the final result**: Collect the final output from the workflow events. The result contains the complete solution developed through the collaborative effort of all participating agents.

Magentic orchestration excels at **solving complex, evolving problems that require real-time coordination between specialized agents**. It's ideal for **tasks where the plan can't be defined in advance and must adapt as new information emerges**. Using the Microsoft Agent Framework, you can build systems that dynamically design, refine, and execute solution paths through intelligent agent collaboration.

Exercise - Develop a multi-agent solution

Now it's your opportunity to b**uild a multi agent solution with the Semantic Kernel Agent Framework**. In this exercise, you create an application that automatically triages and resolves issues presented in log files of a system. Using Azure AI Agents, you create an *incident manager agent* and a *devops agent* that collaborates to fix the issues.

Develop a multi-agent solution

In this exercise, you'll practice using the **sequential orchestration pattern** in the Microsoft Agent Framework SDK. You'll create a simple pipeline of three agents that work together to process customer feedback and suggest next steps. You'll create the following agents:

- The **Summarizer agent** will condense raw feedback into a short, neutral sentence.
- The **Classifier agen**t will categorize the feedback as Positive, Negative, or a Feature request.
- Finally, the **Recommended Action agent** will recommend an appropriate follow-up step.

You'll learn how to use the Microsoft Agent Framework SDK to break down a problem, route it through the right agents, and produce actionable results.

Knowledge check

1. What's the first step in the Microsoft Agent Framework's unified orchestration workflow? Define your agents and describe their capabilities
2. For brainstorming and collaborative problem solving among multiple agents, which orchestration pattern is most suitable? Group Chat
3. Which pattern dynamically transfers control between agents based on context or rules? Handoff

Summary

In this module, you explored how to design and manage multi-agent orchestration workflows using the Microsoft Agent Framework SDK.

You learned how multi-agent systems provide advantages over single-agent approaches, including improved scalability, specialization, and collaborative problem solving. You also learned several

different orchestration patterns—**concurrent, sequential, handoff, group chat, and magentic**—and reviewed guidance on when and how to use each.

You also saw how the SDK provides a unified interface for defining agents, running orchestrations, handling structured data, and retrieving results asynchronously, enabling you to build flexible, reliable, and maintainable multi-agent workflows.

Discover Azure AI Agents with A2A

Learn how to implement the **A2A protocol** to enable **agent discovery, direct communication, and coordinated task execution** across remote agents.

Learning objectives

After completing this module, you're able to:

- Understand the A2A protocol and its role in multi-agent orchestration.
- Design **discoverable agents** for modular, collaborative problem-solving.
- Implement A2A strategies to discover and invoke remote agents.

Introduction

AI agents are powerful on their own, but many real-world tasks require **collaboration across multiple agents**. Coordinating these interactions manually can be complex, especially when agents are remote or distributed.

The **Agent-to-Agent (A2A) protocol** addresses this challenge by providing a standardized framework for agent discovery, communication, and coordinated task execution. By implementing A2A, you can easily manage connections to remote agents, delegate requests to the appropriate agent, and enables seamless communication between agents in a standardized, secure way.

For example, imagine a technical writer who wants to create compelling blog content. One agent generates compelling article headlines, while another creates detailed outlines. Using A2A, a routing agent coordinates the workflow: it sends the user's request to the title agent, passes the generated title to the outline agent, and returns the final outline to the user—all automatically.

In this module, you learn how to **implement the A2A protocol with Azure AI Agents**. You also practice configuring a routing agent, registering remote agents, and building a coordinated workflow that allows multiple agents to collaborate effectively.

Define an A2A agent

Before an A2A agent can participate in multi-agent workflows, it needs to explain what it can do. **Agent Skills and how other agents or clients can discover those capabilities are exposed through an Agent Card**.

Agent Skills

An Agent Skill describes a specific capability or function that the agent can perform. Think of it as a building block that communicates to clients or other agents what tasks the agent is designed to handle.

Key elements of an Agent Skill include:

- **ID**: A unique identifier for the skill.
- **Name**: A human-readable name describing the skill.
- **Description**: A detailed explanation of what the skill does.
- **Tags**: Keywords for categorization and easier discovery.
- **Examples**: Sample prompts or use cases to illustrate the skill in action.
- **Input/Output Modes**: Supported data formats or media types (for example, text, JSON).

When defining a skill for your agent, consider the tasks it should perform, how to describe them clearly, and how other agents or clients might use them. For example, a simple "Hello World" skill could return a basic greeting in text format, whereas a blog-writing skill might accept a topic and return a suggested title or outline.

Agent Card

The Agent Card is like a digital business card for your agent. It's a structured document that a routing agent or client can retrieve to discover your agent's capabilities and how to interact with it.

Key elements of an Agent Card include:

- **Identity Information**: Name, description, and version of the agent.
- **Endpoint URL**: Where the agent's A2A service can be accessed.
- **Capabilities**: Supported A2A features such as streaming or push notifications.
- **Default Input/Output Modes**: The primary media types the agent can handle.
- **Skills**: A list of the agent's skills that other agents can invoke.
- **Authentication Support**: Indicates if the agent requires credentials for access.

When creating an Agent Card, ensure it accurately represents your **agent's skills and endpoints**. This allows clients or routing agents to discover the agent, understand what it can do, and interact with it appropriately.

Putting it together

Once an agent defines its skills and publishes an Agent Card:

- Other agents or clients can discover the agent automatically.
- Requests can be routed to the agent's appropriate skill.
- Responses are returned in supported formats, enabling smooth collaboration across multiple agents.

For example, in a technical writer workflow, one agent could define skills for generating article titles, and another for creating outlines. The routing agent retrieves each agent's card to discover these capabilities and orchestrates a workflow where a title generated by one agent feeds into the outline agent, producing a cohesive final response.

Implement an agent executor

The Agent Executor is a core component of an A2A agent. **It defines how your agent processes incoming requests, generates responses, and communicates with clients or other agents**. Think of it as the bridge between the A2A protocol and your agent's specific business logic.

Understand the Agent Executor

The `AgentExecutor` interface **handles all incoming requests sent to your agent**. It receives information about the request, processes it according to the agent's capabilities, and sends responses or events back through a communication channel.

Key responsibilities:

- Execute tasks requested by users or other agents.
- Stream responses or send individual messages back to the client.
- Handle task cancellation if supported.

Implement the interface

An Agent Executor typically defines two primary operations:

Execute

- Processes incoming requests and generates responses.
- Accesses request details (for example, user input, task context).
- Sends results back via an event queue, which may include messages, task updates, or artifacts.

Cancel

- Handles requests to cancel an ongoing task.
- May not be supported for simple agents.

The executor uses the **RequestContext** to understand the incoming request and an **EventQueue** to communicate results or events back to the client.

Request handling flow

Consider a "Hello World" agent workflow:

1. The agent has a small helper class that implements its core logic (for example, returning a string).
2. The executor receives a request and calls the agent’s logic.
3. The executor wraps the result as an event and places it on the event queue.
4. The routing mechanism sends the event back to the requester.

For cancellation, a basic agent might only indicate that cancellation isn't supported.

The Agent Executor is central to making your A2A agent functional. It defines how the agent executes tasks and communicates results, providing a standardized interface for clients and other agents. Properly implemented executors enable seamless integration and collaboration in multi-agent workflows.

Host an A2A server

Once your agent defines its skills and Agent Card, the next step is to host it on a server. **Hosting makes your agent accessible to clients and other agents over HTTP**, enabling real-time interactions and multi-agent workflows.

Hosting an agent allows it to:

- Expose its capabilities through its Agent Card, which clients and other agents can discover.
- Receive incoming A2A requests and forward them to your *Agent Executor* for processing.
- Manage task lifecycles, including streaming responses and stateful interactions.

Effectively, **the server acts as a bridge between your agent's logic and the external world**, ensuring it can participate in coordinated workflows.

Core components of the agent server

To host an agent, you need three essential components working together:

1. Agent Card

- Describes the agent's capabilities, skills, and input/output modes.
- Exposed at a standard endpoint (typically /.well-known/agent-card.json) so clients and other agents can discover your agent.
- Can include multiple versions or an "extended" card for authenticated users.

2. Request Handler

- Routes incoming requests to the appropriate methods on your Agent Executor (for example, `execute` or `cancel`).
- Manages the task lifecycle using a Task Store, which tracks tasks, streaming data, and resubscriptions.
- Even simple agents require a task store to handle interactions reliably.

3. Server Application

- Built using a web framework (`Starlette` in Python) to handle HTTP requests.
- Combined with an ASGI server (like `Uvicorn`) to start listening on a network interface and port.
- Exposes the agent card and request handler endpoints, enabling clients to interact with your agent.

Set up the A2A agent server

1. **Define** your agent's skills and Agent Card.
2. **Initialize a request handler** that links your Agent Executor with a Task Store.
3. **Set up the server application**, providing the Agent Card and request handler.
4. **Start the server using an ASGI server (Uvicorn)** to make it accessible on the network.

5. Once running, **the agent listens for incoming requests and responds** according to its defined skills.

A "Hello World" agent may expose a basic greeting skill. Once hosted, it can respond to any requests sent to its endpoint. A more complex agent can serve multiple skills or an extended Agent Card for authenticated users.

Hosting an A2A agent combines the **Agent Card**, **request handler**, and **agent executor** to make it available for client and agent interactions. This setup ensures tasks are managed correctly and responses are delivered reliably, enabling your agent to participate in multi-agent workflows.

Connect to your A2A agent

Once your A2A agent server is running, the next step is understanding how a client can interact with it. **A client acts as the bridge between your application and the agent server**.

The client responsibilities include:

- **Discovering the Agent Card**, which contains metadata about the agent and its endpoints.
- **Sending requests to the agent** for processing.
- **Receiving and interpreting the agent's responses**, which can be either direct messages or task-based results.

Connect to your agent server

- The client must know the base URL of the server.
- The client typically retrieves the Agent Card from a well-known endpoint on the server.
- Once the Agent Card is obtained, the client can be initialized with it, establishing a connection ready to send messages.

Send requests to the agent

There are two main types of requests a client can make:

- **Non-Streaming Requests**: The client sends a message and **waits for a complete response**. This type of request is suitable for simple interactions or when a single response is expected.
- **Streaming Requests**: The client sends a message and receives responses incrementally as the agent processes the request. This type of request is useful for long-running tasks or when you

want to update the user in real-time.

In both cases, requests usually include a `role` (for example, user) and the message content. More complex agents may return **task objects** instead of immediate messages, allowing for task tracking or cancellation.

Handle the agent response

Agent responses may include:

- **Direct messages**: Immediate outputs from the agent, such as text or structured content.
- **Task-based responses**: Objects representing ongoing tasks, which may require follow-up calls to check status or retrieve results.

Clients should be prepared to handle both response types and interpret the returned data appropriately.

Interacting with the agent

- Each request should be uniquely identifiable, often using a generated ID.
- Streaming responses are asynchronous and may provide partial results before the final output.
- Simple agents may return messages directly, while more advanced agents may manage multiple tasks simultaneously.

Connecting a client to your agent server involves **fetching the Agent Card**, **establishing a connection**, **sending requests**, and **handling responses**. By grasping these core concepts, you can confidently interact with your remote agent, whether you're sending simple messages or managing complex tasks.

Exercise - Connect to remote Azure AI Agents with the A2A protocol

Connect to remote agents with A2A protocol

In this exercise, you'll use Azure AI Agent Service with the A2A protocol to create simple remote agents that interact with one another. These agents will assist technical writers with preparing their developer blog posts. A title agent will generate a headline, and an outline agent will use the title to develop a concise outline for the article. Let's get started

Module assessment

1. What is the primary role of an A2A server? It routes requests between clients and connected agents.
2. What does the Agent Executor do in an A2A agent? Processes incoming requests and generates responses or events.
3. What is an agent card used for in A2A? It provides metadata about the agent, such as its capabilities and available functions.

Summary

In this module, you learned how to connect Python clients to Azure AI Agents using the **Agent-to-Agent (A2A) protocol**.

By running an A2A server and connecting your client, you explored **how agents are discovered and communicated with dynamically using the Agent Card**. You learned about **executors, which handle agent requests**, and **how messages—both streaming and non-streaming—flow between clients and agents**. Understanding these concepts allows you to build flexible, discoverable agent networks that can delegate tasks and respond to requests across distributed environments.