Building AI Agents and Apps with Microsoft 365

A comprehensive developer's guide to designing, building, and deploying intelligent AI agents within the Microsoft 365 ecosystem — from environment setup to production deployment.

</> DEVELOPER GUIDE

BEST PRACTICES

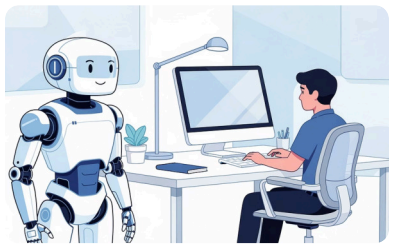
365apps.pro

CHAPTER 1

Understanding the AI Agent Landscape in Microsoft 365

Before writing a single line of code, every developer needs a solid mental model of what Microsoft 365 agents are, how they fit into the broader ecosystem, and what makes them distinct from traditional integrations and bots.

What are Microsoft 365 Agents?

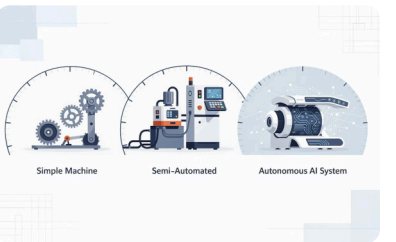


Automated Business Process Execution

AI-powered tools that automate and execute complex business processes end-to-end, reducing manual effort and human error across workflows.

Flexible Collaboration Modes

Function seamlessly alongside or entirely on behalf of individuals, teams, or whole organisations, adapting to the required level of involvement.



Full Spectrum of Autonomy

Range from simple prompt-and-response interactions all the way to fully autonomous agents capable of multi-step reasoning and action without human intervention.

The Power of Agents in Microsoft 365

Deep Ecosystem Integration

Agents connect natively with Microsoft 365 Copilot, Teams, Outlook, and the full suite of Office apps — meeting users exactly where they already work, with no additional context switching required.

Why This Matters for Developers

By leveraging existing Microsoft 365 data, services, and user contexts, developers can build agents that feel native and intelligent from day one. This dramatically shortens time-to-value and reduces integration complexity.

- Boost individual and team productivity
- Streamline end-to-end workflows
- Tap into Graph API and M365 data sources
- Deliver consistent cross-app experiences

Key Components: Agents SDK & Agents Toolkit

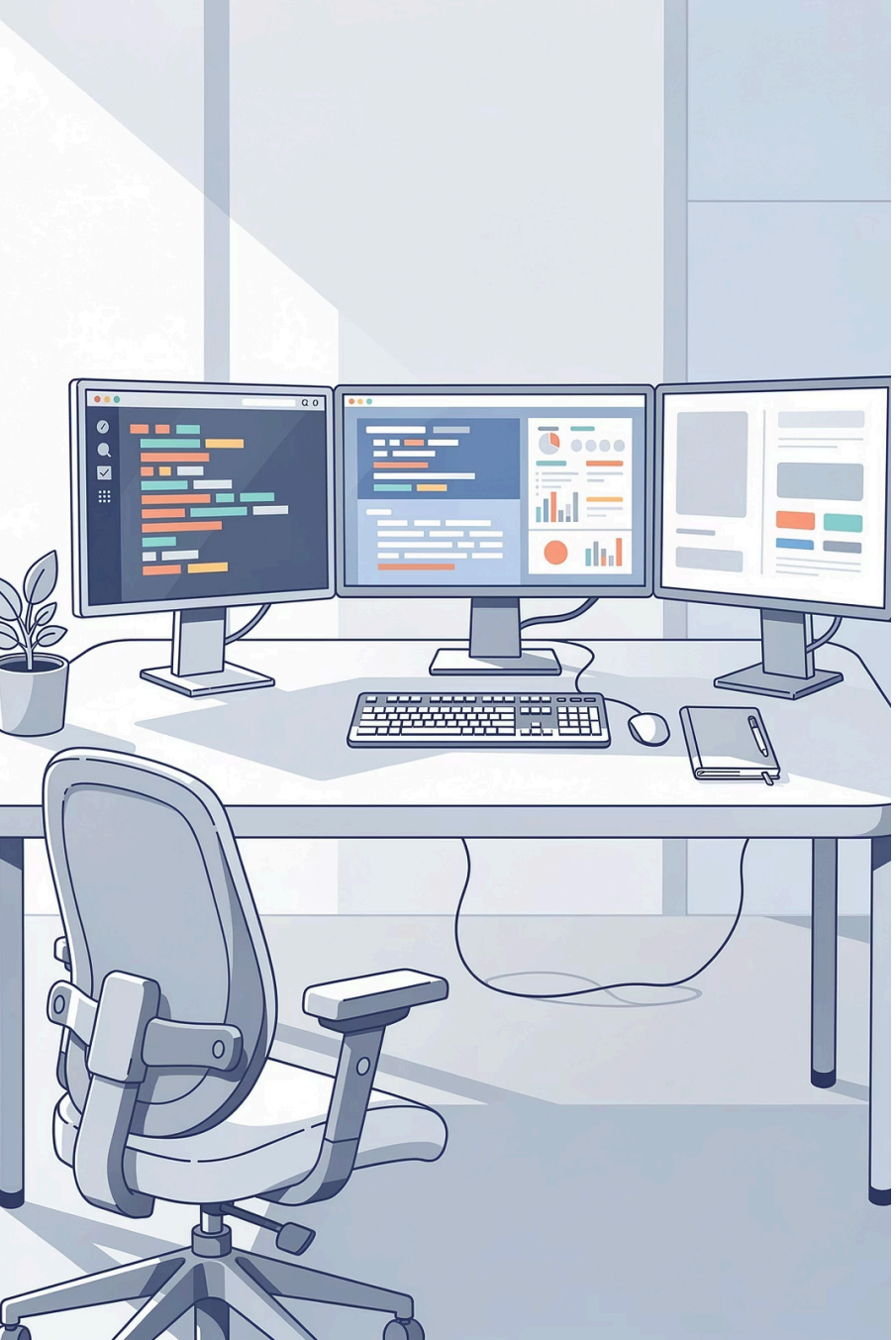
Microsoft 365 Agents SDK

Enables building agents using **your AI stack of choice** — it is entirely model-agnostic and orchestrator-agnostic. Agents can be deployed across multiple channels simultaneously, from Teams to custom web surfaces, without rewriting core logic. The SDK provides foundational constructs for state, storage, and activity handling.

Microsoft 365 Agents Toolkit

An evolution of the popular **Teams Toolkit**, this VS Code extension provides everything needed to scaffold, configure, test, and deploy enterprise-ready agents and apps. It removes boilerplate setup, letting developers focus on business logic and AI integration rather than infrastructure concerns.

Together, the SDK and Toolkit form a cohesive, end-to-end developer experience — from first scaffold to production deployment.



CHAPTER 2

Setting Up Your Development Environment

A properly configured development environment is the foundation of a smooth agent-building experience. This chapter walks through every essential tool and configuration step before writing your first agent.

Essential Tools for Agent Development

Visual Studio Code

The primary IDE. Install the **Microsoft 365 Agents Toolkit** extension from the VS Code Marketplace to unlock scaffolding, debugging, and deployment features.

Node.js

A JavaScript runtime environment required for JavaScript-based agents. Ensure you are running a **compatible version** (v18+) as specified in the Toolkit documentation.

Microsoft Teams

Used for real-world collaboration and **end-to-end testing** of your deployed agents within a live Microsoft 365 environment, including sideloading during development.

M365 Agents Toolkit

Install directly into VS Code. Provides **project templates, local testing tools**, and one-click provisioning and deployment workflows to Azure and Microsoft 365.

Leveraging Azure OpenAI

While optional, integrating **Azure OpenAI** is strongly recommended for any agent requiring advanced language understanding, reasoning, or generative capabilities. Azure OpenAI provides enterprise-grade access to GPT models with security, compliance, and scalability built in.

Getting Started

1. Sign in to the **Azure Portal** and create an Azure OpenAI service resource
2. Choose your preferred GPT model deployment (e.g., GPT-4o)
3. Generate your **API key and endpoint** from the resource settings
4. Store credentials securely using environment variables or Azure Key Vault

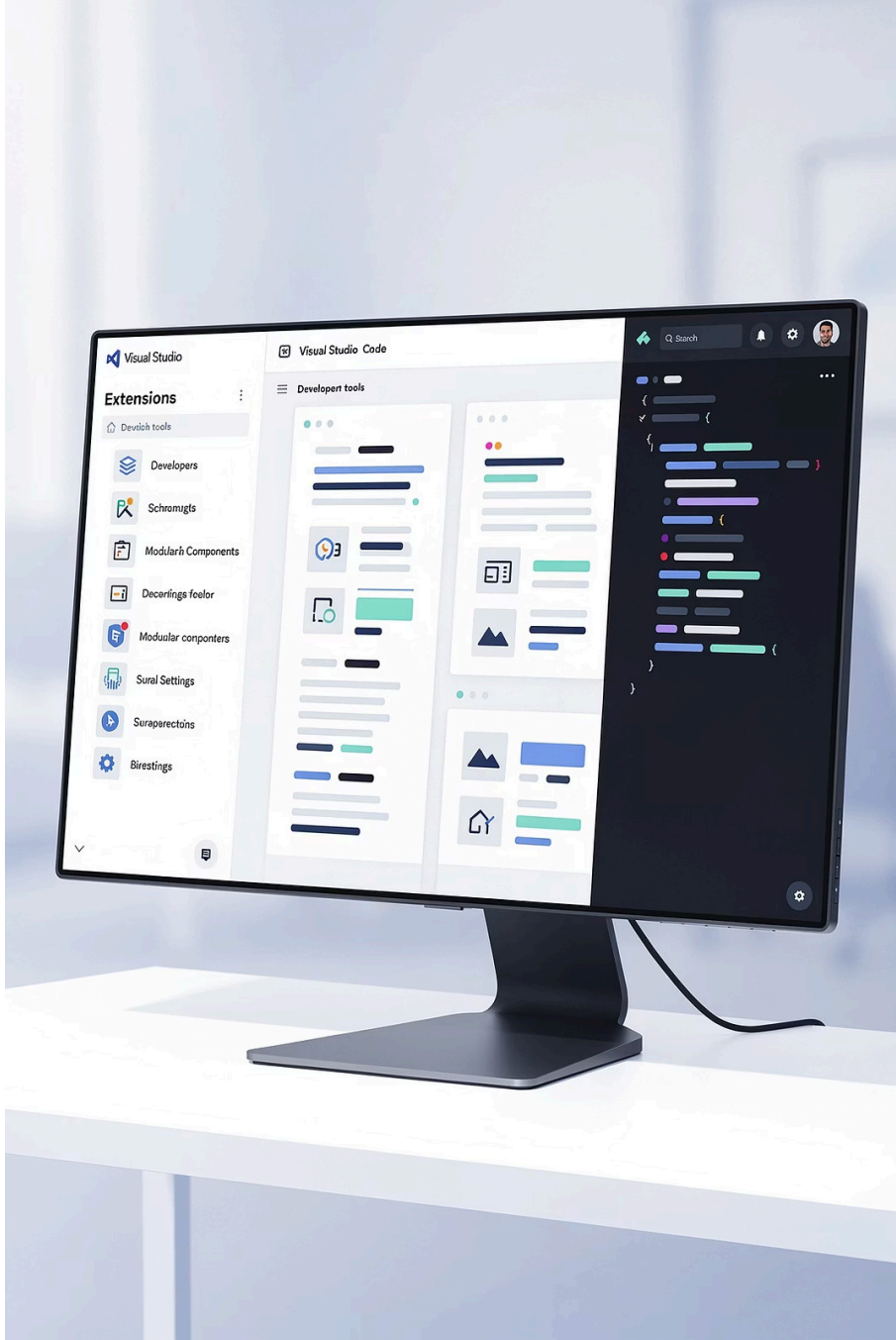
Why Azure OpenAI over Direct OpenAI?

- Data residency and compliance controls
- Private networking with VNet integration
- Microsoft Responsible AI content filtering
- Enterprise SLAs and support
- Seamless integration with other Azure services

Your AI Agent Development Hub

Visual Studio Code, paired with the Microsoft 365 Agents Toolkit extension, provides a fully integrated development environment. From project creation and local debugging to provisioning cloud resources and deploying to Teams — everything is accessible within a single, familiar interface.

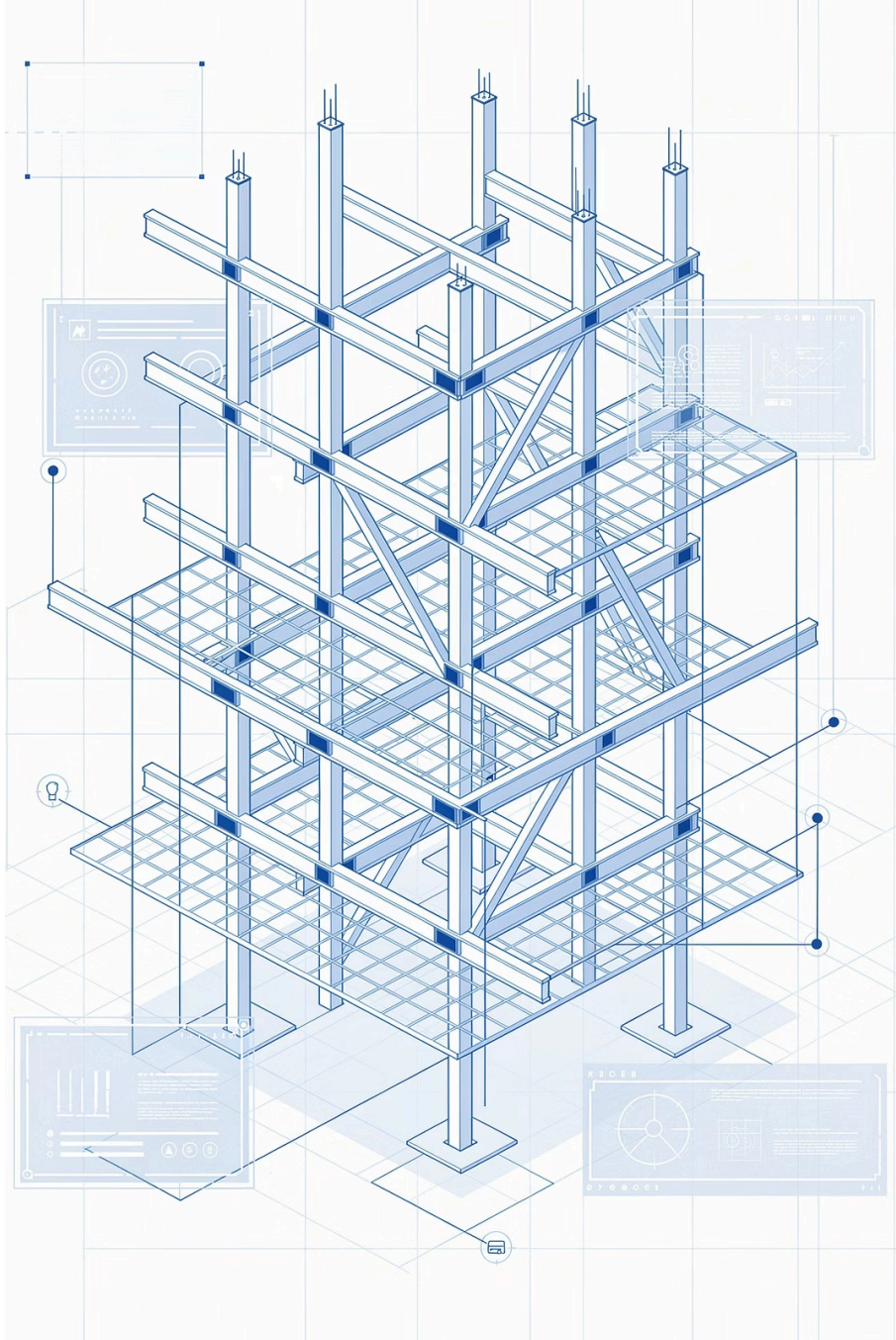
Tip: Use the Toolkit's **Accounts** panel to sign in to both your Microsoft 365 tenant and Azure subscription simultaneously, enabling seamless provisioning without leaving VS Code.



CHAPTER 3

Building Your First Agent with the Agents Toolkit

With your environment ready, it is time to scaffold and build your first Microsoft 365 agent. The Agents Toolkit guides you through every step, from template selection to running your agent locally.



Project Scaffolding with the Agents Toolkit

Open Visual Studio Code

Launch VS Code and ensure the **Microsoft 365 Agents Toolkit** extension is installed and activated. You should see the Toolkit icon in the Activity Bar on the left.

Create a New Agent or App

Navigate to the Toolkit panel and select **"Create a New Agent/App"**. This launches a guided wizard that walks you through selecting your agent type, language, and AI service integrations.

Choose Your Template

Select the most appropriate starting template for your use case — such as **Echo Agent** for a bare-bones scaffold or **Weather Agent** for a more complete reference implementation.

Understanding Agent Templates

Echo Agent / Empty Agent

A deliberately minimal baseline with the core scaffolding in place but no opinionated business logic. Ideal for developers who want full control over their agent architecture from the very beginning. All fundamental SDK patterns are present to learn from and build upon.

Weather Agent

A richer reference implementation demonstrating real-world integration patterns. This template shows how to connect to **Azure AI Foundry** or **OpenAI**, utilise orchestrators like **Semantic Kernel** or **LangChain**, and handle tool calls — providing an excellent blueprint for production-grade agents.

Start with the Echo Agent if you are new to the SDK. Graduate to the Weather Agent once you are comfortable with the fundamental turn/activity model.

Supported Languages and SDKs

The Microsoft 365 Agents SDK is designed with flexibility at its core, supporting multiple programming languages to meet developers where they are most productive.

C#

Using the **.NET 8.0 SDK**. Ideal for enterprise teams with existing C# codebases and deep .NET expertise. Excellent tooling support in Visual Studio and VS Code.

JavaScript

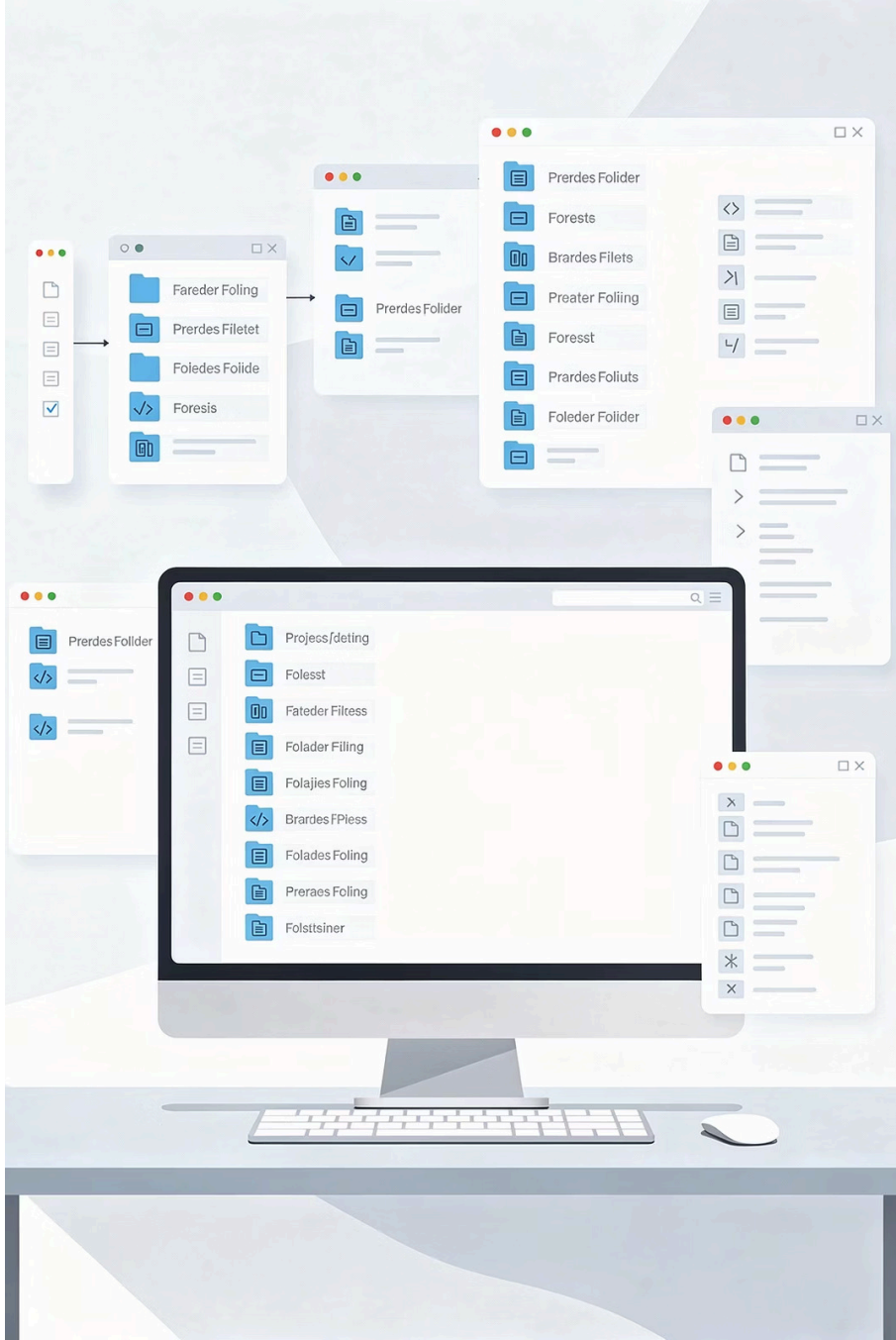
Using **Node.js v18+**. Perfect for teams familiar with the JavaScript ecosystem. Lightweight and fast for rapid prototyping and iteration.

Python

Supporting **versions 3.9–3.11**. The natural choice for data science and ML-heavy teams, and for integrating with Python-first AI libraries and frameworks.

Project Structure at a Glance

The Agents Toolkit generates a well-organised project structure immediately upon scaffolding. You will find clearly separated directories for your agent source code, configuration files (including Teams App manifest), environment variable definitions, and infrastructure-as-code templates for Azure provisioning — giving you a clean, maintainable foundation from day one.



CHAPTER 4

Implementing Agent Logic and AI Integration

This chapter dives into the heart of agent development — understanding the SDK's core concepts, wiring up AI models, and implementing the logic that makes your agent genuinely intelligent and useful.

The Microsoft 365 Agents SDK in Action

The SDK provides a robust container for your agent, handling the heavy lifting of state management, storage, and activity routing — so you can focus on building business value rather than infrastructure plumbing.

Core SDK Concepts

Turn

A single, discrete unit of work performed by the agent in response to an incoming activity. Each turn has a context object providing access to state and channel services.

Activity

An interaction type managed by the agent. Activities can be messages, conversation updates, reactions, and more — each with a distinct handler in your agent code.

Message

The most common activity type. Represents a text or rich-media message sent to or from the agent within a conversation channel.

Technology Agnostic Design

One of the SDK's greatest strengths is its neutrality. You are never locked into a specific AI model, orchestrator, or cloud provider. This means your agent logic remains portable and future-proof as the AI landscape evolves.

- Swap between GPT-4o, Claude, Gemini freely
- Use Semantic Kernel, LangChain, or custom logic
- Combine agents built with different stacks
- Deploy to Azure, on-premises, or hybrid

Integrating AI Services

Connect to AI Models

Wire your agent to any AI model via API — Azure OpenAI, OpenAI directly, or open-source models hosted on Azure AI Foundry. The SDK is entirely model-agnostic; simply inject your model client into the agent's turn handler and call it as needed during a conversation turn.

Orchestrators: Semantic Kernel & LangChain

For agents requiring complex, multi-step reasoning or tool use, orchestrators like **Semantic Kernel** (Microsoft's open-source framework) or **LangChain** provide planner and plugin patterns. These enable agents to break down user goals into sequential sub-tasks and execute them reliably.

Multi-Agent Composition

The SDK supports combining multiple agents built with **entirely different technologies** into a single coherent application. This enables specialised agent roles — for example, a reasoning agent, a retrieval agent, and an action agent — working together to handle complex user requests.

Example: A Simple Echo Agent

The Echo Agent is the "Hello, World" of Microsoft 365 agent development. Despite its simplicity, it demonstrates the complete request-response cycle within the SDK — from receiving an incoming activity to constructing and sending a reply.

How It Works

1. The SDK receives a **message activity** from the channel (e.g., Teams)
2. The agent's `OnMessageActivityAsync` handler is invoked with the turn context
3. Your code extracts the incoming text from the activity
4. A reply is constructed using `MessageFactory.Text()`
5. The reply is sent back via `SendActivityAsync()`

```
await turnContext.SendActivityAsync(
  MessageFactory.Text($"Echo: {text}"),
  cancellationToken);
```

What This Teaches You

- The turn context object and its capabilities
- Activity handler registration patterns
- How to construct and send outbound messages
- The async/await pattern used throughout the SDK
- Foundation for adding AI model calls

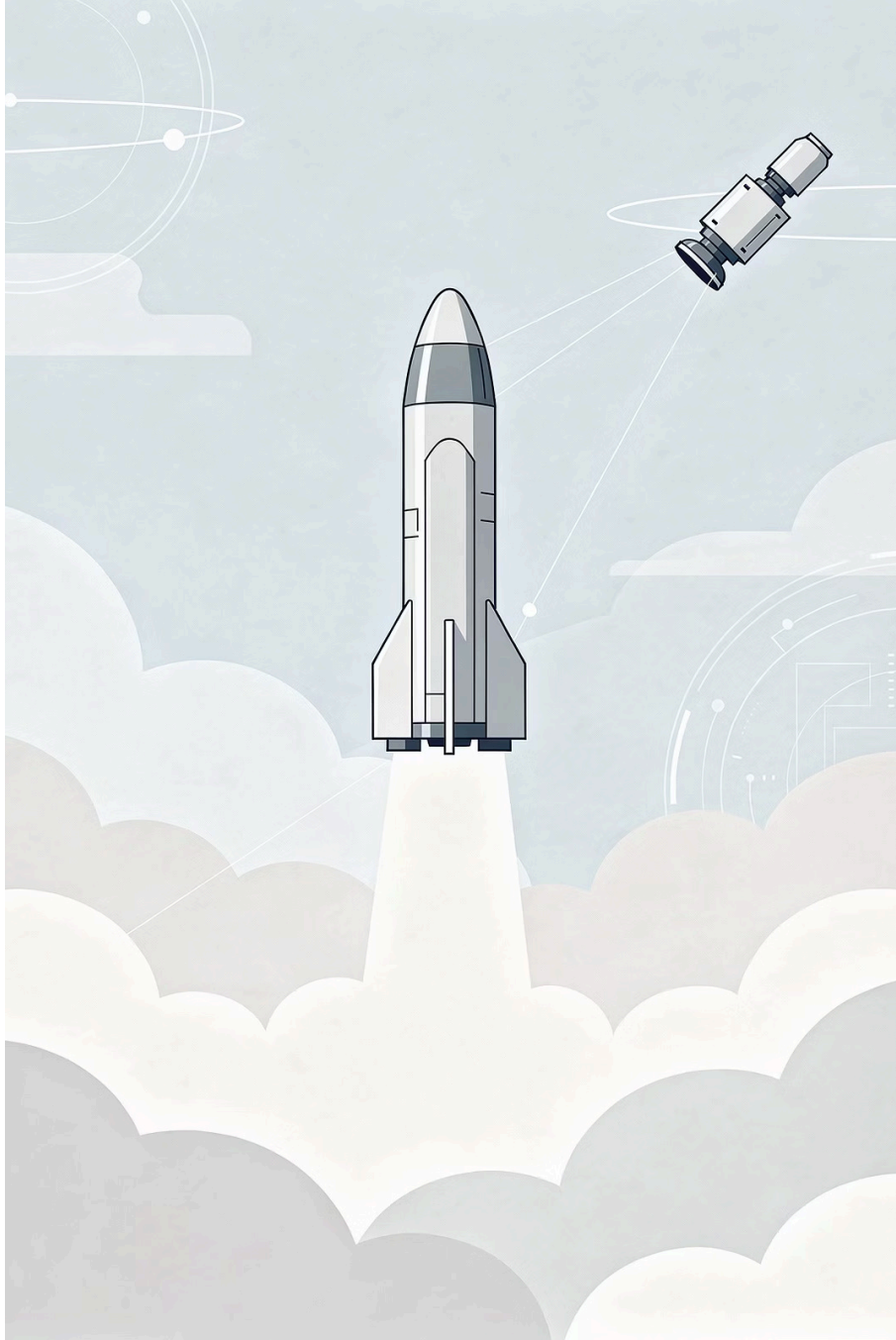
Agent Data Flow Architecture

Understanding this end-to-end data flow is critical for debugging and optimising your agents. Each hop in the flow introduces latency and potential failure points — the SDK abstracts much of this complexity, but developers should be aware of each stage when building production-grade agents.

CHAPTER 5

Testing, Deployment, and Best Practices

Building an agent is only half the journey. Rigorous testing, thoughtful deployment strategy, and adherence to best practices are what separate a proof-of-concept from a production-ready solution that users trust and rely on.



Local Testing and Debugging

Use the Agents Playground

The Agents Toolkit includes a built-in **"agents playground"** — a lightweight local chat interface that lets you interact with your agent without needing a full Teams deployment. This dramatically speeds up the inner development loop.

Debug with VS Code Breakpoints

The Toolkit's launch configuration enables **full breakpoint debugging** inside your agent handler code. Step through turn logic, inspect the activity object, and examine state values in real time directly within VS Code.

Test Varied Input Scenarios

Deliberately test your agent against **edge cases and unexpected inputs** — empty messages, very long inputs, non-English text, and malformed requests. Robust input handling is essential before any production deployment.

⚠ Never test AI agents only with "happy path" inputs. Adversarial and unexpected inputs will reveal brittle logic that could cause failures or unintended responses in production.

Deployment Options

Microsoft 365 Admin Centre

Publish your agent for **organisation-wide distribution**. Administrators can approve, manage, and scope the agent's availability to specific users or groups through the Integrated Apps section of the M365 Admin Centre — ideal for enterprise rollouts with governance controls.

Microsoft Teams Store

Submit your agent to the **Teams App Store** (public or private organisational catalogue) to make it discoverable within Teams. The store provides a polished installation experience and integrates with Teams' permission model for seamless adoption.

Azure Hosting

Deploy your agent's backend services to **Azure App Service, Azure Container Apps, or Azure Functions** for scalable, resilient cloud hosting. The Agents Toolkit can provision the necessary Azure resources directly and configure environment variables automatically.

Declarative Agents for Microsoft 365 Copilot

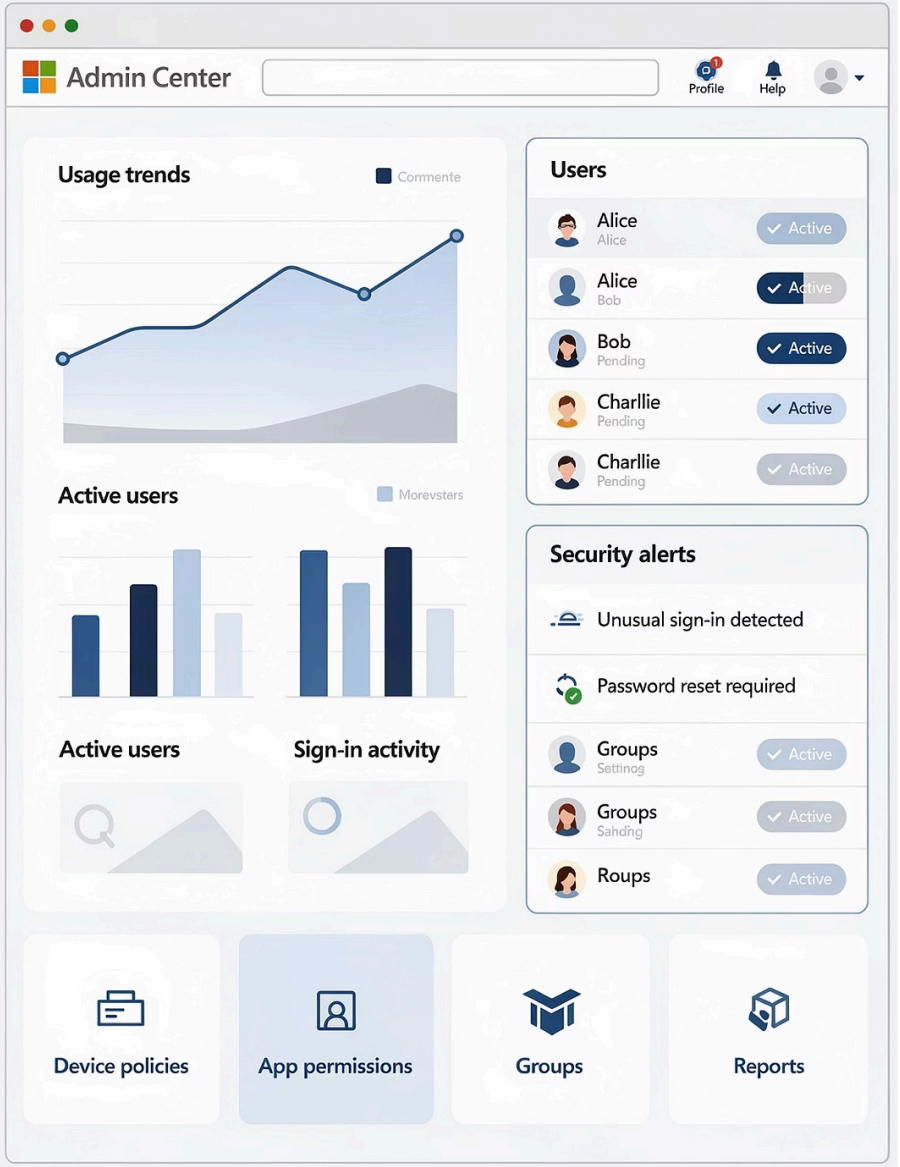
What Are Declarative Agents?

Rather than writing imperative code, declarative agents allow you to **configure an agent's behaviour** by declaring its instructions, available actions, and knowledge sources in a structured manifest. Copilot then interprets and executes these declarations at runtime.

Key Characteristics

- Targets **licensed Microsoft 365 Copilot users** — no separate model hosting required
- Define custom **system prompts and personas** to tailor the Copilot experience
- Attach **knowledge sources** such as SharePoint sites or Graph connectors
- Register **actions** (API plugins) that Copilot can call on the user's behalf
- Currently supported within **Word and PowerPoint** in addition to Teams and Copilot Chat

Declarative agents are the fastest path to a customised Copilot experience — no model fine-tuning or complex orchestration code required.



Deploying Your Agent to the Microsoft 365 Ecosystem

The Microsoft 365 Admin Centre provides IT administrators with full visibility and control over which agents and apps are available to users across the organisation. From this single portal, admins can approve new agent submissions, configure permissions, restrict to specific security groups, and monitor usage — ensuring governance and compliance are maintained throughout the agent lifecycle.

Empowering Developers with AI in Microsoft 365

3

Languages Supported

C#, JavaScript, and Python — build in the language your team knows best.

∞

AI Models Compatible

Fully model-agnostic — connect to any AI service or swap models without rewriting your agent.

3

Deployment Targets

Admin Centre, Teams Store, and Azure — publish once, reach users everywhere.

> The Microsoft 365 Agents Toolkit and SDK provide a powerful, flexible, and enterprise-ready platform. Build intelligent agents that genuinely transform workflows, enhance productivity, and deliver lasting value within the Microsoft ecosystem you and your users already know.

Set Up Your Environment

VS Code, Node.js, and the Agents Toolkit extension.

Scaffold Your Agent

Choose a template, pick your language, wire up your AI service.

Test Thoroughly

Use the playground, debug locally, cover edge cases.

Deploy with Confidence

Publish via Admin Centre, Teams Store, or Azure.