MockCert

Study guide | Microsoft | AI-500

Microsoft Certified: Multi-Agent AI Solutions Expert

Notes for every domain of the Microsoft Certified: Multi-Agent AI Solutions Expert exam, organized by the official exam guide's domains and weights.

By MockCert about 59 min read Updated November 23, 2026

CONTENTS

DOMAIN 1 · 15–20%
Architect multi-agent solutions 2

DOMAIN 2 · 30–35%
Develop multi-agent solutions in Azure 9

DOMAIN 3 · 20–25%
Evaluate, optimize, and monitor multi-agent solutions 22

DOMAIN 4 · 20–25%
Secure, govern, and deploy multi-agent solutions 32

MockCert is an independent certification preparation platform. It is not affiliated with, endorsed by or operated by Microsoft. Exam domains and weights follow the official exam guide. These notes were last updated on November 23, 2026; the notes at mockcert.com are kept up to date as the exam changes.

Domain 1 | 15–20% of the exam | 10 min read

Architect multi-agent solutions

Overview

Every point below comes from Microsoft's official documentation; each section lists its sources.

Design logical architecture for multi-agent solutions

Start here

THINK FIRST

Given a business goal, how would you split it into agents, workflows and tools, and decide where humans and memory fit?

Why it matters: Questions describe a goal or workflow and ask which pattern, tool scope, protocol, approval point or memory design fits it.

A complex goal becomes several specialized agents, each with a tight scope and a fitting model. Multiagent workflows give natural checkpoints where humans review and intervene. MCP gives models tools and context, and A2A lets agents communicate across service boundaries.

Key terms

Term	Meaning
Agents as tools	Composing agents so an outer agent calls an inner agent the way it calls a function tool.
RequestPort	An Agent Framework channel where a workflow pauses until an outside response arrives.
MCP	Model Context Protocol, an open standard for giving LLMs tools and context data.
A2A	Agent-to-Agent, a standard protocol for agents across service and organization boundaries.
Project connection	Where Foundry keeps tool authentication details, such as API keys, instead of app code.

Decomposing the goal

Split a complex problem into specialized, manageable agents. Each agent can use its own model: larger models for complex reasoning and smaller, faster ones for simple work. Agents can also power individual

workflow steps as executors. In the no-code Agent Service approach, a system prompt defines behavior and relationships, and you do not control the compute.

Choosing a workflow pattern

Prompt chaining feeds each agent loop's output into the next loop. An Agent Framework sequential workflow is the same idea: a pipeline from one agent to the next. When you can't predict the subtasks in advance, orchestrator-workers splits the task as it runs.

Agent workflow patterns and when they fit

Pattern	How it works	Fits when
Routing	Sends each request to a matching agent loop or model	Basic and complex questions need different loops
Orchestrator-workers	Breaks the task into subtasks as it runs	Subtasks cannot be predicted in advance
Evaluator-optimizer	One LLM call drafts, another evaluates and gives feedback	A response needs iterative improvement
Handoff	Passes control between loops with context and state	Escalation or expertise transfer is needed

Agents as tools

An inner agent keeps its own instructions, tools and expertise. The outer agent calls it like a function tool and sees only its final text. It does not see the inner agent's reasoning or tool calls, so use tracing to observe them.

EXAM TRAP

An inner agent does not inherit the outer agent's history or context. It gets input only through the tool call arguments.

Human-in-the-loop points

Multi-agent workflows give natural checkpoints for human review. At a RequestPort, the workflow pauses and emits a request until an outside reply arrives. An approval-required tool call also pauses it with a `RequestInfoEvent`. Checkpoints save pending requests and emit them again on restore. A single agent run that needs input ends by naming the input required.

EXAM TRAP

Unlike handoff, which is interactive by default, sequential, concurrent and group chat orchestrations do not pause for free-form user input on their own. Pair them with a RequestPort in a custom workflow.

Approvals and tool gates

A Foundry MCP tool's `require_approval` defaults to always. In Python, `approval_mode="always_require"` on `@tool` forces approval. By default, an approval response counts

only if it matches a pending request in the same `AgentSession`. Standing approvals live in session state. Progressive tool exposure gates which tools the agent sees during a run. The tool list resets to its original set on every `agent.run()`, so each gate re-arms for the next turn. Ordering such as get before update needs no workflow.

EXAM TRAP

In Python, child tool approvals cannot be resumed interactively through a parent agent-as-tool call. Use a workflow when approval must be interactive, delayed or durable.

Tool scopes and authentication

Authentication belongs to the connection, not the agent's code. Store API keys or bearer tokens in a project connection. Agent identity scopes access to one agent, so agents can get different rights on one MCP server. For per-user access, `oauth2` runs an OAuth flow, and `user-entra-token` passes an Entra token. Register MCP and A2A connections with `--kind remote-tool`.

EXAM TRAP

An empty `allowed_tools` list does not block tools. Omitting it or passing `[]` exposes every tool, so list tool names to restrict access.

Communication protocols

Use MCP for tools and A2A between agents. A Toolbox endpoint is MCP-compatible, so any MCP runtime can use it. Treat remote MCP tool descriptions and results as untrusted, since they can carry indirect prompt injection. Toolbox guardrails screen every tool's inputs and outputs.

EXAM TRAP

Agent Service accepts only remote MCP endpoints. Host a local MCP server on Azure Container Apps or Azure Functions first.

Short-term and long-term memory

Short-term memory is the session; long-term memory spans sessions. The orchestration framework usually manages session context. Foundry stores user profile, chat summary and procedural memory. Static memories load at the start of each conversation, and contextual ones are fetched each turn. In multi-agent systems, segment memory logically and operationally.

EXAM TRAP

Low-level memory APIs do not resolve the user automatically. Set scope in every request; automatic resolution needs the memory search tool with `{{$userId}}`.

Key takeaways

- Use orchestrator-workers when subtasks cannot be predicted in advance.
- An agent used as a tool sees only tool call arguments and returns only final text.

- Foundry MCP tools require approval by default, and an empty `allowed_tools` list exposes all tools.
- Keep tool credentials in project connections, not in agent code.
- Treat remote MCP tool descriptions and results as untrusted input.

Sources: Single Agent Loop Versus Multiple Agent Loops - Azure Logic Apps | Microsoft Learn (opens in a new tab) · Agents as Tools | Microsoft Learn (opens in a new tab) · Connect agents to MCP server endpoints - Microsoft Foundry | Microsoft Learn (opens in a new tab) · Microsoft Agent Framework Workflows - Human-in-the-loop (HITL) | Microsoft Learn (opens in a new tab) · Toolbox authentication in Microsoft Foundry - Microsoft Foundry | Microsoft Learn (opens in a new tab) · Agents in Workflows | Microsoft Learn (opens in a new tab) · Build a Multiple-Agent Workflow Automation Solution by using Microsoft Agent Framework - Azure Architecture Center | Mic (opens in a new tab) · Using function tools with human in the loop approvals | Microsoft Learn (opens in a new tab)

Specify technology components to include in a multi-agent solution

Start here

THINK FIRST

Which Foundry and Azure components would you pick to connect, secure, observe and build a production multi-agent system?

Why it matters: Questions describe a multi-agent requirement and ask which component, setting or role satisfies it.

A multi-agent system is assembled from parts: toolboxes and A2A connections for tools and agents, Foundry IQ for knowledge, and Control Plane for monitoring. Foundry Compliance brings policy, Defender for Cloud and Purview insights into one view. Foundry DevPack installs Foundry Toolkit and azd as your development environment.

Key terms

Term	Meaning
Toolbox	A Foundry group of reusable tools exposed behind a managed MCP endpoint.
Knowledge source	A content source a Foundry IQ knowledge base can query; a base can reference several.
Trace Replay	A Foundry view for inspecting and stepping through an agent conversation's trace.
Foundry Control Plane	Fleet monitoring of health, cost and anomalies across many kinds of agents.
Shared state	AG-UI state that the client keeps and sends back to the agent when needed.

Connecting tools and agents

A toolbox puts reusable tools behind one managed MCP endpoint. A2A connections, in preview, let a prompt agent call an A2A-compatible agent as a tool, directly or through a toolbox. Toolbox support in prompt agents is also in preview and off by default. Approval settings on an MCP tool do not grant access to the service behind it.

EXAM TRAP

A prompt agent cannot mix a toolbox with individual Foundry tools. Attaching a toolbox replaces the individual tools.

Routing to knowledge sources

Low and medium effort can query only some knowledge sources. Minimal retrieval queries every candidate source. Retrieval instructions in the knowledge base or retrieve action guide which sources the model picks or skips. Indexed and remote sources differ in citations and permissions.

Indexed and remote knowledge sources compared

Feature	Indexed source	Remote SharePoint source
Content index in Foundry IQ	Yes, a backing search index	No; retrieval uses the Copilot Retrieval API
Citation URLs	Service-generated URLs to indexed documents	Not returned
Document access	Purview sensitivity labels, when ingested for a blob, indexed OneLake or indexed SharePoint source	The user's own SharePoint permissions

Zero Trust boundaries

A shared index still needs document-level permissions. Use separate pipelines and indexes for content that needs its own governance. API Management does not check where it forwards a managed identity token, so you must keep tokens on trusted backends. Behind a Key Vault firewall, API Management must use its system-assigned identity. Serverless Foundry IQ lacks private indexer execution.

EXAM TRAP

Policy editing in API Management is a privileged right. A policy editor could change a policy to leak the managed identity's token and reach its resources.

Mapping compliance controls

Foundry Compliance gathers policy and security findings in one view. It combines policy information, asset compliance, Defender for Cloud recommendations and Purview insights. Its network egress area covers limits on outbound connections and destinations.

EXAM TRAP

Foundry Compliance does not replace Azure Policy evaluation. It only gives visibility into the relevant results.

Shared state with AG-UI

The client keeps shared state and sends it back when needed. A `STATE_SNAPSHOT` sets or replaces state, while deltas carry small changes. In .NET, mapping tool results to shared state is opt-in.

EXAM TRAP

Client state is not trusted. Validate its shape and values before it reaches prompts, routing or privileged operations.

Tracing and replay

Control Plane needs OpenTelemetry GenAI conventions in agent logs. Trace Replay opens from any page showing a Conversation ID or Trace ID. Its Trajectories view shows each reasoning step, tool call and turn, with bars by duration or tokens. The High filter shows spans over **2k** tokens. Viewing traces needs a Log Analytics reader role.

EXAM TRAP

Playthrough does not work on every trace. It needs **two** or more spans, so a single-span trace cannot be replayed step by step.

Fleet monitoring and gates

Control Plane tracks fleet health, cost and anomalies. It covers Foundry agents, Azure SRE Agent, Logic Apps agent loops and registered custom agents. Health metrics show active agents, completion rates and error trends. Anomaly detection flags cost spikes and slowdowns. Cost data needs Cost Management Reader. `azd ai eval` can grade emitted traces, and `--fail-on` turns results into a build gate.

EXAM TRAP

Agents on resources without Application Insights get no health metrics, cost tracking or drill-down traces in Control Plane.

Developer tools and compute

Foundry DevPack installs the full developer toolset. It adds Foundry Toolkit, azd and azd ai, Foundry Canvas and the Foundry Skill. The Toolkit builds prompt agents or code-based hosted agents. Start with a prompt agent, and choose a hosted agent for more execution control. Versions are immutable, and Save to Foundry creates one without publishing an endpoint.

EXAM TRAP

Saving a prompt locally in Agent Builder does not make inference local. That prompt may still call a cloud model.

Key takeaways

- Attaching a toolbox to a prompt agent replaces its individual Foundry tools.
- Remote SharePoint sources use the user's permissions and return no citation URLs.
- Foundry Compliance shows policy results but does not replace Azure Policy evaluation.
- Without Application Insights, Control Plane has no health, cost or trace data for an agent.
- Start with a prompt agent; move to a hosted agent when you need execution control.

Sources: Create a prompt agent with Microsoft Foundry Toolkit for Visual Studio Code - Microsoft Foundry | Microsoft Learn (opens in a new tab) · What is a Knowledge Source? - Foundry IQ (Azure AI Search) | Microsoft Learn (opens in a new tab) · Monitor AI agent fleet health and performance - Microsoft Foundry | Microsoft Learn (opens in a new tab) · Monitor AI governance and compliance in Microsoft Foundry - Microsoft Foundry | Microsoft Learn (opens in a new tab) · Install Microsoft Foundry Toolkit for Visual Studio Code - Microsoft Foundry | Microsoft Learn (opens in a new tab) · Review agent interactions with Trace Replay - Microsoft Foundry | Microsoft Learn (opens in a new tab) · State Management with AG-UI | Microsoft Learn (opens in a new tab) · Use Managed Identities in Azure API Management | Microsoft Learn (opens in a new tab)

Domain 2 | 30–35% of the exam | 19 min read

Develop multi-agent solutions in Azure

Overview

Every point below comes from Microsoft's official documentation; each section lists its sources.

Design and implement advanced prompt engineering strategies in multi-agent solutions

Start here

THINK FIRST

When should you improve an agent through its prompt and context, and when is fine-tuning the model worth the cost?

Why it matters: Questions describe an agent's prompt, context or model problem and ask which prompt technique, context setting or training approach fits.

Few-shot examples steer a model for one request without changing it. Versioning keeps each prompt tied to the model it was tested on. Fine-tuning changes the model's weights, and you must redo it when you move to a new model revision.

Key terms

Term	Meaning
Few-shot examples	Sample user and assistant turns placed after the system message to show the desired answers.
Recency bias	The tendency for the end of a prompt to sway the output more than its start.
Context provider	An Agent Framework component that injects instructions, messages or tools before a run.
Fine-tuning	Training that adapts a pretrained model's weights with task-specific data.

Examples and prompt layout

Few-shot examples steer only the current request. With Chat Completions, add them as user and assistant turns after the system message. In RAG prompts, place retrieved context before the user's question, since models attend more reliably to earlier context. Because of recency bias, try repeating key instructions at the end.

Suggested RAG prompt budgets

Part	Budget	Guidance
System message	**200** to **500** tokens	Keep it concise
Few-shot examples	**300** to **800** tokens	Two to three examples
Example coverage	Not a token limit	A full answer, a partial answer and a no-answer case

Defensive guidelines

Tell the model to treat retrieved context as data only. Retrieved documents can hide instructions, which is indirect prompt injection. Also give the model a way out, such as answering "not found" when the text lacks the answer. That alternative path helps it avoid inventing responses.

EXAM TRAP

A system message is not a guarantee. It influences the model but does not ensure compliance, so layer it with filtering and evaluation.

Prompt lifecycle and versions

Tie each prompt version to the model version it targets. Record the prompt text, its evaluation results and settings such as temperature, top-p and max tokens. Saved prompt-agent versions are immutable, so every change creates a new version. To avoid untested upgrades, pin model versions or set Azure OpenAI deployments to No Auto Upgrade. For Azure OpenAI and serverless API models, existing deployments of a retired model return HTTP errors.

Optimizing a prompt agent

The optimizer improves instructions, tool descriptions or model choice. It scores the baseline and every candidate on the same dataset. That dataset must already use the evaluators' column names, since the wizard cannot map columns. It rewrites function tool descriptions but cannot run those client-side tools. Ephemeral Responses API agents cannot be optimized.

EXAM TRAP

Promoting a candidate creates a new version, but a pinned agent sends it no traffic. Set the new version as active first.

Injecting dynamic context

Context providers add instructions, messages or tools before a run. They can also extract state after a run. HarnessAgent turns on its todo, mode, file memory and skills providers by default, then adds yours after them. In Python, FileMemoryProvider shares memory across sessions only when you pass a stable scope such as a user ID.

EXAM TRAP

Do not keep session state inside an AIContextProvider. One instance serves every session of the agent, so store per-session state in the AgentSession.

Context across agents

A sub-agent shares the caller's session only when told to. With `as_tool()`, add `propagate_session=True` to share the AgentSession. Application state then flows down and merges back, but approval state stays with its own agent. In C#, keep per-run data in `AdditionalProperties` and lasting state in the session's `StateBag`. Only one Python history provider should load messages.

Choosing a training method

Fine-tuning changes weights instead of adding prompt examples. DPO trains on preferred and rejected responses to the same input. RFT suits tasks with many valid answers whose quality you can score reliably.

Preference and reinforcement fine-tuning compared

Method	What you supply	How it is billed
Preference fine-tuning (DPO)	Preferred and rejected responses to the same input	Tokens in the training file times epochs
Reinforcement fine-tuning (RFT)	Prompts and a grader that scores responses	Training time, plus any model-grading costs

Training cost and frequency

Reassess a tuned model as data changes or its base model retires. Retraining and redeployment may follow. The Developer training tier saves money on idle capacity but can be preempted. Developer deployments are removed after **24** hours. Failed jobs, queue time and data safety checks are not charged. An RFT job that hits its spending limit pauses with a deployable checkpoint.

EXAM TRAP

A fine-tuned model does not carry over to a new model revision. You must fine-tune the new version again before using it.

Key takeaways

- Few-shot examples condition only the current inference; fine-tuning changes the model's weights.
- In RAG prompts, place the context before the user's question and treat it as data only.
- Version each prompt with its evaluation results and send it only to the model version it targets.
- Keep session state in the AgentSession, never in a shared context provider instance.
- DPO learns from preferred and rejected responses; RFT learns from a grader's scores.

Sources: Prompt engineering techniques - Microsoft Foundry | Microsoft Learn (opens in a new tab) · Design to Support Foundation Model Life Cycles - Azure Architecture Center | Microsoft Learn (opens in a new tab) · Training in Microsoft Foundry - Microsoft Foundry | Microsoft Learn (opens in a new tab) · Context Providers |

Microsoft Learn (opens in a new tab) · Develop a RAG Solution on Azure - Prompt Engineering - Azure Architecture Center | Microsoft Learn (opens in a new tab) · System message design for Azure OpenAI - Microsoft Foundry | Microsoft Learn (opens in a new tab) · Agent development lifecycle in Microsoft Foundry - Microsoft Foundry | Microsoft Learn (opens in a new tab) · Quickstart: Optimize a prompt agent - Microsoft Foundry | Microsoft Learn (opens in a new tab)

Design and implement agent memory, context management, and knowledge integration

Start here

THINK FIRST

How do you decide what an agent should remember, what it should look up, and how to keep each user's data apart?

Why it matters: Questions describe an agent's memory, history or retrieval need and ask which memory type, setting or knowledge source fits.

Short-term memory holds the current conversation, while long-term memory keeps durable facts across sessions. Retrieval brings in outside knowledge on demand without storing it as memory. Scopes keep isolated collections of memory items, for example one per customer.

Key terms

Term	Meaning
Short-term memory	The current session's conversation, usually managed by the orchestration framework.
Long-term memory	Durable knowledge Foundry Agent Service extracts from conversations and keeps across sessions.
Scope	A partition of a memory store with its own isolated memory items, such as one per customer.
Knowledge base	What every agentic retrieval scenario needs, together with at least one knowledge source.

Short-term and long-term memory

Foundry Agent Service memory is built for the long term. It turns conversations into durable knowledge available across sessions. It stores **three** types: user profile, chat summary and procedural memory. Static memories load at the start of each conversation, and contextual memories are fetched every turn. Fetch stable profile details early so answers feel personal.

EXAM TRAP

Foundry Agent Service memory is designed for long-term memory across sessions. The current session's conversation is short-term memory, usually managed by the orchestration framework.

Storing and reducing history

History lives in the local session or in the service. Local session state keeps the full chat in the session. With service-managed storage, the session keeps only a remote conversation ID. When local history outgrows model limits, apply a reducer. For token-aware reduction, use a compaction strategy instead of message counts.

EXAM TRAP

Reducer settings only affect in-memory history providers. With service-managed history, reduction depends on the provider or service.

Memory scopes and lifecycle

Each scope is an isolated set of memory items. A store allows **100** scopes and **10,000** memories per scope. Setting scope to `{{$userId}}` reads the `x-memory-user-id` header, or else the caller's Entra IDs. A store can set a default TTL in seconds. Memory writes complete only after the `update_delay` quiet period.

EXAM TRAP

A remember command does not override TTL. If the store has a TTL, items added that way can still expire.

Securing memory and sessions

Screen what enters and leaves memory for prompt injection. Use Azure AI Content Safety prompt injection detection on memory traffic. Memory stores do not support virtual network integration. For many users on one key, keep conversation IDs server-side and verify ownership before resuming. In shared hosted sessions, the platform stops cross-user response chains and filters `context.get_history()`.

EXAM TRAP

The platform does not partition data your container stores itself in a shared session. Key that data by session ID and user ID, not session ID alone.

Chunking and embeddings

Chunk by document structure, not fixed character counts. The Document Layout skill keeps headings and splits text by paragraphs and sentences. Its indexer needs `allowSkillsetToReadFileData` set to true and `parsingMode` set to default. With Content Understanding, `chunkingProperties.method` set to semantic follows paragraph and heading boundaries.

EXAM TRAP

Do not mix embedding models between indexing and querying. Use the same endpoint, deployment and model in the embedding skill and the query vectorizer.

Agentic retrieval

Query planning splits a complex question into subqueries. Every scenario needs a knowledge base and at least **one** knowledge source. Planning can use chat history. Subqueries run at once as keyword, vector or hybrid search, each semantically reranked. Low or medium planning needs an LLM. In the 2026-10-01 GA contract, set limits on the retrieve request.

Indexed, remote and MCP sources

MCP Server sources query live data with no ingestion. Remote sources have no backing index, so they don't return citation URLs. For document-level permissions, set `query_source_credential` to the caller's credential to trim results per user. If the SDK lacks query-source authorization, Agent Framework fails closed with a `ValueError`.

EXAM TRAP

MCP Server knowledge sources do not support minimal retrieval reasoning effort. Use low or medium instead.

Adding Foundry IQ to agents

Add Foundry IQ through a toolbox attached as an MCP tool. The tool's default `query_type` is `vector_semantic_hybrid`. With a private virtual network, use project managed identity, since keys are not supported. Agent Framework retrieves this content before the model runs, not as memory.

Foundry IQ modes in Agent Framework

Feature	Semantic mode	Agentic mode
What it searches	An existing index	A Foundry IQ knowledge base
How it retrieves	Can combine keyword and vector retrieval	Query planning and multi-hop retrieval

EXAM TRAP

The Foundry IQ tool for Foundry agents cannot search several indexes at once. It can only target one index.

Key takeaways

- Foundry Agent Service memory is long-term; the framework handles the current session.
- Partition memory with scopes, up to **100** per store.
- In shared sessions, key container data by session ID and user ID.

- Use the same embedding model for indexing and querying.
- MCP Server sources read live data with no ingestion but need low or medium effort.

Sources: What is Memory? - Microsoft Foundry | Microsoft Learn (opens in a new tab) · Foundry IQ (Azure AI Search) | Microsoft Learn (opens in a new tab) · Create and use memory in Foundry Agent Service - Microsoft Foundry | Microsoft Learn (opens in a new tab) · Agentic Retrieval Overview - Foundry IQ (Azure AI Search) | Microsoft Learn (opens in a new tab) · Storage | Microsoft Learn (opens in a new tab) · Multiplex multiple users in one hosted agent session - Microsoft Foundry | Microsoft Learn (opens in a new tab) · Chunk and Vectorize by Document Layout (Document Layout Skill) - Foundry IQ (Azure AI Search) | Microsoft Learn (opens in a new tab) · Chunk and Vectorize with Content Understanding - Foundry IQ (Azure AI Search) | Microsoft Learn (opens in a new tab)

Build and integrate tool ecosystems

Start here

THINK FIRST

When an agent needs outside tools, how do you choose where they run, how they are secured and what happens when one fails?

Why it matters: Questions describe a tool integration and ask which hosting option, MCP setting, auth method or error-handling feature fits.

Simple tools can run in-process through function calling. Tools that need separate development and deployment can move to Azure Functions, where an MCP tool trigger defines tool endpoints. Retry policies and error sections then handle failures.

Key terms

Term	Meaning
Function calling	Tools defined in your agent code that run in the same process as the agent.
MCP tool trigger	An Azure Functions trigger that defines tool endpoints that agents and models can call.
Custom handler	How a self-hosted MCP server built with an official MCP SDK runs on Azure Functions.
on-error section	The API Management policy section that runs when request processing hits an error.

In-process or hosted tools

Host tools on Azure Functions to separate business logic. Hosted tools can be built, tested and deployed apart from the agent. All function calling features, including parallel calls, need a model released after **November 6, 2023**. A queue-based Functions tool receives a message on an input queue. Its reply carries the original CorrelationId so the agent matches it to the right call.

EXAM TRAP

The queue-based Azure Functions tool does not work with the basic agent setup. It needs the standard setup's queue resources and managed network integration.

Two ways to host MCP on Functions

Use the MCP extension or deploy a self-hosted SDK server. Setting `configurationProfile` to `mcp-custom-handler` applies the host settings needed to run an MCP server on Functions. The extension supports only the isolated worker model for C# and does not support PowerShell.

MCP extension and self-hosted MCP servers on Azure Functions

Feature	MCP extension	Self-hosted SDK server
How it is built	Functions with MCP tool triggers that define tool endpoints	A server built with an official MCP SDK, run as a custom handler
State and hosting	The option to pick when the server must be stateful	Stateless, streamable-http, Flex Consumption plan (preview)
Key for key-based access	`mcp_extension` system key	`default` host key

Transports and endpoints

Use Streamable HTTP unless a client needs SSE. The extension serves Streamable HTTP at `/runtime/webhooks/mcp`. When you do use SSE, the extension relies on Queue storage in the default host storage account, `AzureWebJobsStorage`.

EXAM TRAP

SSE is not the modern default. Newer MCP protocol versions deprecate it, so use Streamable HTTP unless a client specifically needs SSE.

Securing a Functions MCP server

Without the `mcp_extension` key, callers get 401 Unauthorized. Clients send the key in `x-functions-key` or the `code` query string. Setting the webhook authorization level to Anonymous removes the key requirement. Built-in MCP authentication also turns off key-based access. For Entra sign-in from Foundry, only project managed identity is supported. Use OAuth identity passthrough when each user needs their own identity.

EXAM TRAP

Anonymous access is not for business data. Keep it for tools that return read-only public information, or for private development.

Defining tools and MCP Apps

Mark a property required only when the tool needs it. MCP Apps combine tool and resource triggers so a tool can return an interactive UI. The tool declares a `ui.resourceUri` that points to a UI resource. The MCP host renders that resource in a sandboxed iframe in the chat.

EXAM TRAP

Tool properties in the MCP extension are optional by default. If you forget to mark a vital property required, clients may omit it.

MCP servers in API Management

API Management can protect an MCP server with a subscription key. Clients can send a subscription key in `Ocp-Apim-Subscription-Key`. The credential manager injects OAuth 2.0 tokens into backend calls. Tools can change, by adding, renaming or removing them, with no need to recreate the server. SSE passthrough needs **two** endpoints, while streamable passthrough needs **one**.

EXAM TRAP

Deleting a REST API operation that an MCP tool still references fails with a dependency error. Delete the tool first.

Retries and fallbacks

Logic Apps retries on timeouts and 408, 429 or 5xx responses. For most operations, the Default policy is exponential, with up to **4** retries capped between **5** and **45** seconds. Fixed and exponential policies allow **1** to **90** retries. An action runs only after the previous one succeeds, by default. In API Management, an error jumps straight to `on-error`, where `context.LastError` holds the details.

EXAM TRAP

API Management policies have no on-error section by default. Without one, callers just receive a **400** or **500** response.

Checking tool results

Turn on payload logging to see tool arguments and results. API Management does not capture them by default; enable frontend and backend logging on the MCP server. Failed requests record `error.message` and `error.type`. In Logic Apps, `result()` returns the top-level action results of a scope. In the playground, review each tool's name and arguments, then select Approve.

EXAM TRAP

Do not read `context.Response.Body` in policies at MCP scope. MCP responses stream, and reading the body breaks the stream.

Key takeaways

- A Functions tool reply must carry the original CorrelationId so the agent can match it.
- Self-hosted MCP servers on Functions must be stateless; pick the MCP extension for state.
- Use Streamable HTTP; SSE is deprecated in newer MCP protocol versions.
- Logic Apps retries timeouts and **408**, **429** and **5xx** failures under its retry policy.
- Add an on-error section in API Management, or callers only see **400** or **500**.

Sources: Integrate Azure Functions with Foundry Agents - Microsoft Foundry | Microsoft Learn (opens in a new tab) · MCP tool trigger for Azure Functions | Microsoft Learn (opens in a new tab) · Handle Workflow Errors and Exceptions - Azure Logic Apps | Microsoft Learn (opens in a new tab) · Error handling in Azure API Management policies | Microsoft Learn (opens in a new tab) · Self-hosted remote MCP server on Azure Functions (public preview) | Microsoft Learn (opens in a new tab) · Model context protocol bindings for Azure Functions | Microsoft Learn (opens in a new tab) · Connect an MCP server on Azure Functions to Foundry Agent Service | Microsoft Learn (opens in a new tab) · Build an MCP Apps server using Azure Functions | Microsoft Learn (opens in a new tab)

Implement multi-agent orchestration

Start here

THINK FIRST

How do you choose an orchestration pattern, and where do human approvals, MCP tools and middleware fit around it?

Why it matters: Questions describe a multi-agent task and ask which Agent Framework pattern, approval setting, MCP option or middleware fits.

Agent Framework patterns differ in who decides the next step: a fixed order, parallel work, a shared chat, a manager or direct handoffs. A workflow can pause after an agent responds so a human reviews the work. Middleware wraps every run with shared logic such as logging and security.

Key terms

Term	Meaning
Magentic manager	A planning manager agent that coordinates specialized agents dynamically.
Handoff	A pattern where agents pass control directly to each other based on context.
Hosted MCP tool	An MCP server the AI service calls on the service side, not code that runs locally.
Middleware	Code that intercepts agent interactions to add logging, security or other shared behavior.

Choosing a pattern

Pick the pattern by who decides what happens next. Sequential runs agents in a set order, while concurrent runs them all at once. Magentic adds a manager that plans as the work goes.

Agent Framework orchestration patterns compared

Pattern	How agents coordinate	Note
Sequential	One after another in a set order	Each agent gets the previous full conversation by default
Concurrent	In parallel	Suits brainstorming, ensemble reasoning and voting
Group Chat	In one shared conversation	Preferred for simple coordination without planning
Magentic	A manager coordinates specialists as it goes	For tasks whose solution path is unknown upfront
Handoff	Agents pass control to each other	A mesh of agents with no orchestrator

Sequential workflows

The terminal output holds only the last agent's response. Set `chain_only_agent_responses=True` so each agent reads only the prior agent's reply. Add `.with_request_info()` to pause after chosen agents for human review. Agent names there must match exactly, including case, or construction raises `ValueError`.

EXAM TRAP

Sequential orchestration has no built-in pause for free-form feedback between agents. It also cannot send control back to an earlier agent.

Concurrent workflows

Results merge into one response, one message per agent. A custom aggregator can process the results differently. By default only the aggregator's output is a terminal event. Use `intermediate_output_from` to surface chosen participants' outputs as intermediate events.

Controlling a Magentic manager

Limits stop the manager from looping forever. You can cap coordination rounds, consecutive stalls before replanning and plan resets. Rounds without progress raise a stall counter, and passing its limit forces a reset and replan. In plan review, a human calls `approve()` or `revise(...)` with feedback.

EXAM TRAP

Two workflows running at once should not share a stateful explicit manager. Use `manager_factory` so each workflow keeps its own manager state.

Handoff behavior

Handoff is interactive by default. If an agent answers without handing off, the user gets the next turn. A user reply goes back to the agent that asked, unless you disable return-to-previous routing. In autonomous mode, the default cap is **50** turns. Only user and agent messages are shared, not tool calls. Participants must be agents that run tools locally.

EXAM TRAP

Handoff is not the same as agents as tools. In handoff the receiving agent owns the task, while with agents as tools the primary agent keeps responsibility.

Human approval for tools

Mark tools that need a person to approve each call. In Python, set `approval_mode` to `"always_require"` on the `@tool` decorator. In .NET, wrap the function in `ApprovalRequiredAIFunction`. A `RequestPort` pauses a workflow until an outside reply arrives. Checkpoints keep pending requests and emit them again as `RequestInfoEvent` objects on restore.

EXAM TRAP

A local approval response is not accepted on its own. By default it must match a pending request in the same `AgentSession`, or it is ignored with a warning.

Connecting MCP tools

Hosted MCP tools run on the service side. With `RequireApproval` set to `"always"`, every call needs user approval, while `"never"` runs tools automatically. OpenAI Responses requires approval for every hosted MCP call by default. For a server that needs authentication, set an Authorization header and load the secret from a store, never source control.

EXAM TRAP

Leaving out `allowed_tools` sends no filter at all. Passing `[]` sends an explicit empty allowlist instead.

Middleware for shared concerns

Agent middleware wraps run middleware. With agent middleware A1, A2 and run middleware R1, R2, the order is A1, A2, R1, R2, then the agent. Middleware suits error handling, retries and graceful degradation. It can stop early by setting `context.result` and raising `MiddlewareTermination`. In .NET, the builder returns a new agent and leaves the original unchanged.

EXAM TRAP

Chat middleware does not run only once per request. It runs inside the function invocation loop, so it fires for every model call, including tool-result calls.

Key takeaways

- Use Magentic when the solution path is unknown; prefer Group Chat when no planning is needed.
- A sequential workflow returns only the last agent's response as its terminal output.
- In handoff the receiving agent owns the task; with agents as tools the primary agent keeps it.
- Require approval with `approval_mode="always_require"` in Python or `ApprovalRequiredAIFunction` in .NET.
- Put logging, security checks and retries in middleware so every agent shares them.

Sources: Workflow orchestrations in Agent Framework | Microsoft Learn (opens in a new tab) · Microsoft Agent Framework Workflows Orchestrations - Sequential | Microsoft Learn (opens in a new tab) · Adding middleware to agents | Microsoft Learn (opens in a new tab) · Microsoft Agent Framework Workflows Orchestrations - Magentic | Microsoft Learn (opens in a new tab) · Microsoft Agent Framework Workflows Orchestrations - Handoff | Microsoft Learn (opens in a new tab) · Hosted MCP Tools | Microsoft Learn (opens in a new tab) · Microsoft Agent Framework Workflows Orchestrations - Concurrent | Microsoft Learn (opens in a new tab) · Microsoft Agent Framework Workflows - Human-in-the-loop (HITL) | Microsoft Learn (opens in a new tab)

Domain 3 | 20–25% of the exam | 14 min read

Evaluate, optimize, and monitor multi-agent solutions

Overview

Every point below comes from Microsoft's official documentation; each section lists its sources.

Design and implement evaluation and validation strategies for multi-agent solutions

Start here

THINK FIRST

How would you combine human reviewers and automated evaluators to judge whether a multi-agent solution is ready for production?

Why it matters: Questions describe an evaluation goal and ask which Foundry evaluator, role, template setting or comparison step fits it.

Automated evaluators score tool use, grounding and safety, but automated safety labels can be wrong. Human reviewers, such as domain experts, approve or flag results before release. Pinned versions keep scores comparable across runs, and t-testing makes run comparisons more reliable.

Key terms

Term	Meaning
Evaluation template	The set of questions human reviewers answer about agent responses in Foundry.
Tool Use Quality	A preview composite evaluator that scores the whole tool-use process in one judge call.
Groundedness	A RAG quality evaluator you can apply to an agent's final text response to assess knowledge grounding.
Judge model	The chat-completion deployment an AI-assisted evaluator uses to score responses.

Why humans stay in the loop

Automated safety labels can be wrong, so humans check them. An AI-generated severity label or its reasoning may contain errors. The portal's human feedback column lets a reviewer give a thumbs up or thumbs down to approve or flag each instance. Microsoft recommends domain experts review results before you deploy to end users. Safety evaluations also do not replace manual red teaming for your own domain.

EXAM TRAP

Passing the automated safety evaluations is not full coverage. They augment testing but were not designed to replace manual red teaming aimed at your use cases and users.

Human evaluation templates

Only one template is active at a time. A template can mix slider, multiple choice, free-form text and thumbs up/down questions. A new template starts as Inactive, and you activate it before reviewers begin. Activating another template switches the previous one off automatically.

EXAM TRAP

You cannot run two human evaluation templates side by side. Activating a second template deactivates the first one, so plan one template per review round.

Roles for human review

Template owners need more rights than reviewers. Managing templates takes Foundry Project Manager or higher. Reviewers need Foundry User on the project and Reader on the account. Viewing results in the portal needs Foundry User on the project.

Minimum roles for each human review task

Task	Role on the project	Role on the account
Create and manage templates	Foundry Project Manager or higher	Not stated
Submit reviewer feedback in the web app	Foundry User	Reader
View evaluation results in the portal	Foundry User	Not stated

EXAM TRAP

Foundry User and Azure AI User are the same role. The roles were renamed, but role IDs and core permissions did not change.

Review results and data

Results export as a CSV file per template. Reviewers may skip some responses or rate the same response more than once. Select Download Results to get every evaluation for a template, with all fields from the JSON view. The data lives in Application Insights and follows its retention policy.

Evaluating tool calls

Tool Use Quality batches five tool evaluators into one judge call. It covers Tool Selection, Tool Call Accuracy, Tool Input Accuracy, Tool Call Success and Tool Output Utilization, which cuts cost and latency. For strict checks, Tool Input Accuracy tests parameters against **six** criteria and returns a binary Pass or Fail.

EXAM TRAP

Tool-focused evaluators and groundedness have limited support for Bing Grounding and Foundry IQ calls. Avoid them when the conversation includes those tools.

Evaluating knowledge and memory

Use Relevance and Groundedness for knowledge grounding. These RAG evaluators judge the final text response of an agent. An unexpectedly low Groundedness score points to incomplete retrieval or context. Ground agents on curated content with a Foundry IQ knowledge base rather than memory. Protect memory against prompt injection and corruption, because bad stored data can steer later responses.

Judges and multi-agent scope

System evaluators target the orchestrator or final agent. In a multi-agent system, score the main orchestrator or the agent that completes the task. Every AI-assisted evaluator takes its judge in `initialization_parameters`, and the judge must be a chat-completion deployment in your project. Target-based evaluation only works with synchronous, non-streaming agents on the responses or invocations protocol. Use trace evaluation for A2A, Activity, long-running or streaming agents.

Comparing runs fairly

Pin versions so scores stay comparable. Pin datasets, model deployments, evaluator versions and rubric versions across runs. Foundry compares runs with t-testing, and ImprovedStrong means p of **0.001** or lower in the desired direction. A result is Inconclusive with too few examples or p of **0.05** or more. The comparison view is not saved, so reselect the runs and select Compare again.

Key takeaways

- Human reviewers, ideally domain experts, should validate automated evaluation results before release.
- Only **one** human evaluation template can be active, and a new template starts Inactive.
- Template management needs Foundry Project Manager; reviewers need Foundry User plus Reader on the account.
- Tool Use Quality scores the whole tool-use process in **one** judge call.
- An unexpectedly low Groundedness score suggests incomplete retrieval or context.

Sources: Agent Evaluators for Generative AI - Microsoft Foundry | Microsoft Learn (opens in a new tab) ·

Microsoft Foundry risk and safety evaluations Transparency Note - Microsoft Foundry | Microsoft Learn (opens in a new tab) · Microsoft Foundry evaluation | Microsoft Learn (opens in a new tab) · See Evaluation Results in Microsoft Foundry portal - Microsoft Foundry | Microsoft Learn (opens in a new tab) · Human Evaluation for Microsoft Foundry Agents - Microsoft Foundry | Microsoft Learn (opens in a new tab) · Quickstart: Evaluate your hosted agent - Microsoft Foundry | Microsoft Learn (opens in a new tab) · Run benchmark evaluations in Microsoft Foundry - Microsoft Foundry | Microsoft Learn (opens in a new tab) · What is Memory? - Microsoft Foundry | Microsoft Learn (opens in a new tab)

Optimize prompt and model performance for multi-agent solutions

Start here

THINK FIRST

When a multi-agent solution gets slow, forgetful or stale, how do you tell whether to fix parallelism, context or the prompt itself?

Why it matters: Questions describe a slow, rate-limited or forgetful agent workflow and ask which change or improvement loop fixes it.

Parallel agents finish sooner, but a single model endpoint or shared knowledge store can rate-limit them. Context grows with every agent, so compact it between agents or keep long-running state in a durable store. A judge model then scores each agent response against your criteria.

Key terms

Term	Meaning
TPM and RPM	Tokens and requests per minute. Without subscription-level quota, these limits are set per region, subscription and model.
Compaction	Summarizing or trimming older history so a conversation stays within token limits.
Context provider	A source whose context sits in the prompt on every call, unlike a tool called on demand.
Composite score	The agent optimizer's overall score for a candidate, from **0.0** to **1.0**.
Eval model	The judge model that reads each response and criterion and returns a binary score.

Parallelism and its cost

Concurrent agents cut run time but spike model use. In concurrent orchestration, agents work in parallel rather than one after another. Many simultaneous model calls can then hit limits on a single model endpoint or shared knowledge store.

EXAM TRAP

Parallel is not always faster. Avoid concurrent orchestration when model quota or other resource limits make parallel work inefficient or impossible.

Rate limits and 429 errors

HTTP 429 means a TPM or RPM limit was exceeded. Retry with exponential backoff and honor the Retry-After header. Increase load gradually instead of in sharp jumps. Models without their own row default to **400,000** TPM, **1,000** RPM and **300** concurrent requests. For sustained high throughput, consider provisioned throughput deployments.

EXAM TRAP

Under subscription-level quota, Global Standard deployments of one model and version share a single pool across all regions. Adding a region does not add quota.

Hosted agent startup time

Readiness time covers process start to HTTP 200 on `/readiness`. To check startup, send `x-ms-debug-latency-enabled: true` and watch `x-ms-debug-latency-container-readiness-ms` on several cold requests. Keep model calls, migrations, tool discovery and downloads out of startup, and run independent setup in parallel. For .NET, ReadyToRun speeds startup but enlarges the image and build.

EXAM TRAP

Low readiness time with slow requests means startup is not the problem. The delay is in request handling.

When context overflows

Past the window limit, the oldest or least relevant context is cut. That truncation can drop facts the agent still needs. Very long windows can also degrade model performance. If an agent needs only new instructions, pass those alone instead of the earlier context.

EXAM TRAP

A context provider is not free between calls. It spends tokens on every invocation, while a tool spends tokens only when called.

Compaction and durable state

Summarize or prune context between agents. Each agent adds reasoning, tool results and outputs, so multi-agent context grows fast. Monitor its size and compact older history while keeping key information. For long or multi-session work, store progress, intermediate results and history in a durable store, not only in memory.

Retrieval that recalls well

File search combines keyword and semantic search, then reranks. By default it uses **800**-token chunks with **400** tokens of overlap and adds at most **20** chunks to context. If a conversation's vector store is still processing files, a response waits up to **60** seconds. That wait does not apply to the agent's vector store.

The agent optimizer loop

Candidates are ranked by a composite score from 0.0 to 1.0. The eval model can be any deployed chat-completion model, but the optimization model must be on the supported list. The optimizer makes **5** candidates by default and learns from earlier ones. Tool optimization rewrites descriptions only, not types, defaults or required fields. Hosted agents need the Responses protocol.

Reading agent optimizer results

Result	What it means	What to do
Starred candidate	Highest composite score	Review it as the optimizer's recommendation
Gain under **0.03**	Noise, not a meaningful improvement	Do not treat it as a real gain
Every candidate below baseline	Nothing beat the current setup	Deploy none; the baseline stays active

Prompts, data and user feedback

Each reoptimization builds on the latest optimized text. Prompt Optimizer handles text instructions only, not tools or knowledge sources. The portal wizard takes data from agent traces, a registered dataset or an uploaded JSONL file. User feedback on a Likert scale passes at **3.0** or above by default. Score nondeterministic outputs with rubrics or a model judge, not exact matches.

Key takeaways

- Parallel agents shorten run time, but a single model endpoint or shared knowledge store can rate-limit them.
- On HTTP **429**, retry with exponential backoff using the Retry-After value.
- Compact context between agents and keep long-running state in a durable store.
- An optimizer gain under **0.03** is noise; if every candidate loses, keep the baseline.

Sources: AI Agent Orchestration Patterns - Azure Architecture Center | Microsoft Learn (opens in a new tab) · Agent optimizer in Foundry Agent Service overview - Microsoft Foundry | Microsoft Learn (opens in a new tab) · Microsoft Foundry Models quotas and limits - Microsoft Foundry | Microsoft Learn (opens in a new tab) · Adding Context Providers | Microsoft Learn (opens in a new tab) · Optimize agent instructions, skills, tools, and models in Foundry Agent Service - Microsoft Foundry | Microsoft Learn (opens in a new tab) · Quotas and limits for Microsoft Foundry Agent Service - Microsoft Foundry | Microsoft Learn (opens in a new tab) · Reduce startup latency for hosted agents - Microsoft Foundry | Microsoft Learn (opens in a new tab) · Vector stores for file search in Foundry Agent Service - Microsoft Foundry | Microsoft Learn (opens in a new tab)

Design and implement observability and monitoring for multi-agent solutions

Start here

THINK FIRST

How would you know, within minutes, that a multi-agent solution is failing, drifting or overspending, and trace the cause?

Why it matters: Questions describe a monitoring, tracing, quota or cost need and ask which Foundry or Azure feature, setting or role meets it.

Traces flow into Application Insights, where dashboards track tokens, latency, error rates and quality scores. Trace context propagation correlates client-side spans with server-side spans. Alerts and scheduled evaluations with test datasets help you catch problems and drift after deployment.

Key terms

Term	Meaning
Server-side tracing	Traces Foundry records automatically once Application Insights is connected, with no code changes.
Trace context propagation	Adding W3C `traceparent` and `tracestate` headers so client spans join server spans.
Continuous evaluation	Quality and safety scoring of a sample of production traffic.
PTU deployment	A Provisioned Throughput deployment with dedicated capacity and a latency SLA.
Copilot harness	A preview harness that manages some response settings itself, such as `max_output_tokens`, so requests must not send them.

Turning tracing on

Connect Application Insights to start server-side tracing. Foundry then records traces with no code changes, and you get built-in traces for the past **90** days. Agents built with Agent Framework or Semantic Kernel emit traces with no extra packages. Only a Foundry Account Owner or Foundry Owner can enable collection. Telemetry storage can cost money based on volume and retention.

EXAM TRAP

Turning tracing off does not delete what was collected. Earlier trace data stays under the Application Insights retention policies.

Correlating client and server spans

Trace context propagation joins client spans to server spans. In the Python SDK it is on by default once tracing is enabled. Set `AZURE_EXPERIMENTAL_ENABLE_GENAI_TRACING` to true before calling `AIProjectInstrumentor().instrument()`. Spans can go to Application Insights, the console or any OTLP backend. Hosted agent packages tag every span with project, agent name, version and ID.

EXAM TRAP

In a workflow, the target executor's `executor.process` span is not a child of the source's `message.send` span. It is linked to it, because execution is not nested.

What spans record

Record message content in development, not in production. Content recording shows full requests and responses while you debug; turn it off in production to protect sensitive data. On `execute_tool` spans, `gen_ai.tool.call.arguments` holds tool arguments. With sensitive capture off, workflow failures keep ERROR status but drop exception messages. Querying telemetry needs Log Analytics Reader.

EXAM TRAP

Turning off message capture does not hide custom function data. The `@trace_function` decorator always records parameters and return values.

Dashboard warning signs

Watch success rate, latency and tokens on the dashboard. Application Insights dashboards track tokens, latency, error rates and quality scores in real time. Find traces by Response ID or Trace ID. The thresholds below suggest where to look first.

Agent Monitoring Dashboard signals

Signal	Warning level	What it may mean
Run success rate	Below **95%**	Failed runs need investigation
Latency	Above **10** seconds	Throttling, complex tool calls or network issues
Token usage	High for the time range	Verbose prompts or responses to optimize

Catching drift and regressions

Scheduled evaluation with test datasets detects drift. Continuous evaluation scores sampled live traffic. Link evaluation run IDs to traces to see quality and performance together. Alerts can fire on latency, token usage, scores or red team findings. Insights flag recurring failures, hallucinations and token spikes, each with a likely cause and next action.

EXAM TRAP

Playground evaluations are not free or off by default. They are on for every project and billed by consumption until you unselect all evaluators.

Availability and SLAs

ModelAvailabilityRate is successful calls over total calls. It subtracts server errors from total calls. When latency is a core need, PTU gives dedicated capacity and a latency SLA. API Management with a circuit breaker can route around unhealthy backends. Global Standard and Data Zone add resilience but do not replace a multi-region failover plan.

EXAM TRAP

Azure Service Health reports platform problems and maintenance, not slow regressions in your own workload. Use Azure Monitor metric alerts for those.

Token and tool limits

Raise `max_tokens` to stop truncated responses. For GPT-4o it defaults to **4,096**. An agent can register at most **128** tools. A vector store file can hold up to **2,000,000** tokens. These file, message and tool limits are fixed and cannot be increased. The Copilot harness manages `max_output_tokens`, `max_tool_calls` and `parallel_tool_calls`, so do not send them.

Quotas and cost

Quota is enforced per subscription, not per tenant. For sustained **429** errors, back off and honor Retry-After, spread load through an API Management AI gateway, or ask for more quota. For `regional_session_quota_exceeded`, back off or use another region. Idle or stopped hosted sessions keep state but do not count toward session quota.

EXAM TRAP

Approved PTU quota is a limit, not reserved capacity. It does not guarantee capacity in the region you asked for.

Key takeaways

- Connecting Application Insights turns on server-side tracing with no code changes.
- Trace context propagation correlates client and server spans with W3C headers.
- Investigate when run success falls below **95%** or latency passes **10** seconds.
- Scheduled evaluation detects drift; Service Health does not catch workload regressions.
- PTU quota is a limit, not a capacity reservation.

Sources: Observability in Generative AI - Microsoft Foundry | Microsoft Learn (opens in a new tab) · Add client-side tracing to Foundry agents - Microsoft Foundry | Microsoft Learn (opens in a new tab) · Monitor agents with the Agent Monitoring Dashboard - Microsoft Foundry | Microsoft Learn (opens in a new tab) · Set Up Tracing for AI Agents in Microsoft Foundry - Microsoft Foundry | Microsoft Learn (opens in a new tab) · Stay

informed about service health regressions - Microsoft Foundry | Microsoft Learn (opens in a new tab) · Quotas and limits for Microsoft Foundry Agent Service - Microsoft Foundry | Microsoft Learn (opens in a new tab) · Configure tracing for AI agent frameworks - Microsoft Foundry | Microsoft Learn (opens in a new tab) · Microsoft Foundry Tracing and Data Handling - Microsoft Foundry | Microsoft Learn (opens in a new tab)

Domain 4 | 20–25% of the exam | 16 min read

Secure, govern, and deploy multi-agent solutions

Overview

Every point below comes from Microsoft's official documentation; each section lists its sources.

Design and implement security for multi-agent solutions

Start here

THINK FIRST

How would you give each agent in a multi-agent solution only the access it needs, and prove it is safe before release?

Why it matters: Questions describe an agent's access, credential or network need and ask which identity, role, Key Vault setting or test fits.

Every agent gets its own Entra identity with no permissions, and you grant roles at resource or resource-group scope where possible. Azure RBAC is the recommended way to authorize the Key Vault data plane. The AI Red Teaming Agent then probes the solution before deployment.

Key terms

Term	Meaning
Agent identity	A special Microsoft Entra service principal that represents one agent instance when it calls downstream resources.
Agent identity blueprint	The parent object agent identities are created under; it is allowed to impersonate them.
Delegated access	On-behalf-of access where the agent acts for the signed-in user instead of as itself.
Attack Success Rate	The share of red teaming attacks that draw an undesirable response; the key scan metric.

Agent identities start empty

A new agent identity has no resource permissions. Foundry assigns no RBAC roles or downstream rights to it, so you grant access in each target service. Agent Service uses a federated credential on the blueprint to

create identities and get tokens. No client secret or certificate is needed, and agent code never holds the credential.

EXAM TRAP

Creating an agent identity does not give it access to storage, search or other services. Until you assign roles in the target service, its calls are not authorized.

Least privilege and boundaries

Scope roles to a resource or resource group. Prefer the narrowest Azure RBAC scope that meets the agent's needs over subscription-wide access. Permissions also stop at each boundary. Agent identities can take Conditional Access policies, identity protection and network access controls.

EXAM TRAP

Permission to manage an agent does not let a principal call it at runtime. A runtime caller also gains no automatic access to resources behind the agent.

Authentication flows

An agent acts as itself or on behalf of a user. With agent identity, Agent Service requests a token scoped to the target service's audience, which is its OAuth resource identifier. With delegated access, both the agent and the user must be authorized.

Agent identity and delegated access compared

Feature	Agent identity	Delegated (on-behalf-of)
Who the agent acts as	Its own agent identity	The signed-in user
Who needs access	The agent identity, granted in the target service	The agent for the delegated permissions, and the user for the resource
How it works	Agent Service runs an automatic OAuth 2.0 token exchange with Entra ID	In preview; needs agent blueprint endpoint access

Managed identity over keys

Prefer Entra ID with a managed identity to shared keys. For Azure Storage, disabling Shared Key authorization where compliance allows removes key rotation entirely. In API Management, the `authentication-managed-identity` policy gets a token and sends it as a Bearer value in the Authorization header. Without `client-id`, it uses the system-assigned identity. On a self-hosted gateway, set `authentication-provider="WorkloadIdentity"` and omit `client-id`.

EXAM TRAP

API Management does not check which backend receives the managed identity token. Sending tokens only to trusted backends is your responsibility.

Network boundaries

The agent subnet must be /27 or larger. Start with **/24** or larger for busy production workloads. Several Foundry resources can share a virtual network, but each needs its own agent subnet. The Foundry resource sits in the same region as the virtual network, while Cosmos DB, Foundry IQ and Storage may be elsewhere. People who build agents need the Foundry User role on the project.

EXAM TRAP

You cannot add virtual network injection to an existing Foundry account for hosted agents. Include it when you first create the account.

Key Vault access models

Azure RBAC is the recommended Key Vault data plane model. From API version 2026-02-01 it is also the default for new vaults. Under the legacy access policy model, broad roles can grant themselves data access. RBAC works with Privileged Identity Management, so privileged rights expire. Changing `enableRbacAuthorization` needs `Microsoft.KeyVault/vaults/write`, and the portal also needs role assignment rights to avoid lockout.

Key Vault permission models compared

Feature	Access policy model	Azure RBAC
Who can grant data plane access	Contributor, Key Vault Contributor or any role with vaults/write	Owner or User Access Administrator
When it applies	Vaults whose `enableRbacAuthorization` is null keep using it	Default for new vaults from API version 2026-02-01

EXAM TRAP

Updating an existing vault with API version 2026-02-01 or later does not switch it to Azure RBAC. It changes only when you set `enableRbacAuthorization` yourself.

Secrets, certificates and rotation

Key Vault can trigger rotation 30 days before expiry. It publishes a near-expiry event to Event Grid, which can call a rotation function. With one credential, the secret fails during the lag before the service updates. The dual-key pattern alternates primary and secondary keys as secret versions instead. Rotating a storage key revokes account-level SAS tokens made from it, so regenerate them.

Shift left with red teaming

The AI Red Teaming Agent finds known risks before deployment. Do a manual round of red teaming first, then measure and mitigate. A default scan sends **10** prompts for each of **4** risk categories, **40** in all. Judge results by Attack Success Rate. The agent supports single-turn, text-only interactions, and agentic risk categories run only in cloud red teaming.

EXAM TRAP

Agentic risk categories, such as prohibited actions or sensitive data leakage, are not available in local scans. They run only in cloud red teaming.

Key takeaways

- Grant each agent identity narrow roles yourself; Foundry assigns none automatically.
- Delegated access needs authorization for both the agent and the signed-in user.
- Use Azure RBAC for Key Vault; existing vaults switch only when you change `enableRbacAuthorization`.
- Each Foundry resource needs its own agent subnet of **/27** or larger.
- Measure red teaming scans by Attack Success Rate, after an initial manual round.

Sources: Agent identity concepts in Microsoft Foundry - Microsoft Foundry | Microsoft Learn (opens in a new tab) · Azure role-based access control (Azure RBAC) vs. access policies | Microsoft Learn (opens in a new tab) · AI Red Teaming Agent - Microsoft Foundry | Microsoft Learn (opens in a new tab) · Run AI Red Teaming Agent Locally (Azure AI Evaluation SDK) - Microsoft Foundry | Microsoft Learn (opens in a new tab) · Identity and access overview for Foundry Agent Service - Microsoft Foundry | Microsoft Learn (opens in a new tab) · Rotation tutorial for resources with two sets of credentials | Microsoft Learn (opens in a new tab) · Azure API Management policy reference - authentication-managed-identity | Microsoft Learn (opens in a new tab) · Set up private networking for Foundry Agent Service - Microsoft Foundry | Microsoft Learn (opens in a new tab)

Design and implement guardrails for multi-agent solutions

Start here

THINK FIRST

Where in a multi-agent workflow would you place guardrails, and how would you prove they actually block what they should?

Why it matters: Questions describe an agent's risks and ask which intervention point, guardrail setting, approval step or test fits.

Harmful content can enter at the user's prompt, a tool call, a tool's reply or the final answer. A layered guardrail checks each of these points. You then test that guardrail on a non-production agent and measure again after every change.

Key terms

Term	Meaning
Intervention point	A place where guardrail controls run: user input, tool call, tool response or output.
Guided setup	A Foundry flow that turns your answers about users, data and tools into targeted controls.

Term	Meaning
Task Adherence	A check that flags tool use that does not match the agent's task, through `taskRiskDetected`.
Spotlighting	A defense Foundry turns on with tool response validation against prompt injection in tool outputs.

Four intervention points

Guard user input, tool calls, tool responses and output. Intermediate agents can introduce or pass on harmful content, so one check at the edge is not enough. Tool call controls run before the agent invokes a tool. The tool call point is in preview and works for agents only.

What a hosted agent does when each point blocks

Intervention point	What is checked	When it blocks
User input	The incoming request	HTTP 400 with `content_filter` before the agent runs
Tool call	Tool name, arguments and content the agent proposes	The tool is not run; the agent waits for new user input
Tool response	Content a supported tool returns	The agent stops; nothing is saved to agent memory
Streamed output	The agent's response	Status stays 200; the stream ends with a `content_filter` error event

EXAM TRAP

Tool call and tool response controls only work for tools that support moderation, such as Azure Functions, OpenAPI and Bing Grounding.

Which guardrail applies

An agent's own guardrail fully overrides the model's. Assign a custom guardrail to an agent and that guardrail is used. With none assigned, the agent inherits its model deployment's guardrail. You can also pass a custom guardrail in a request header to override the deployment guardrail for one call.

EXAM TRAP

The agent and model guardrails are not merged. Risk detection follows only the guardrail assigned to the agent.

Building a custom guardrail

Default guardrails are read-only, so create a custom one. You cannot edit Microsoft Default guardrails such as Default.V2. The Violence, Hate, Sexual and Self-harm controls on inputs and outputs can be overridden but not deleted. The High blocking level is the strictest, blocking low, medium and high severity.

EXAM TRAP

Document attack detection is off in the default safety policies. Core-harm filters and user prompt attack detection are on, so enable document attacks yourself.

Guided setup per agent

Run guided setup separately for each agent. Agents on the same model can need different guardrails. Your answers become controls at the right points. Saying the data includes PII turns on PII detection, and code generation adds protected material and code safety controls.

Guided setup answers and the controls they add

If the agent	Foundry adds	To reduce
Serves public users	Stricter content filtering and jailbreak protection	Exposure to unexpected or adversarial input
Calls external tools	Tool response validation and spotlighting	Prompt injection through tool outputs
Takes consequential actions	Task adherence controls and action validation	Misaligned or unintended actions

Hosted agent policy pitfalls

A bad policy setup can let everything through. Without `rai_config`, a hosted agent has no content safety guardrail. With `rai_config` but no `rai_policy_name`, Microsoft.DefaultV2 applies. On the invocations protocol, a policy without `invocations_moderation` screens nothing, yet the agent still returns HTTP **200**.

EXAM TRAP

A successful deploy does not prove the policy ID is valid. An agent can reference a missing policy, report active and filter nothing, because the guardrail fails open.

Gating tool use

Validate model-generated tool calls before side effects run. Use approval and least privilege for high-risk tools. Agent Framework Tool Approval adds a human decision before the model sees the result. When Task Adherence returns `taskRiskDetected` as true, block the call or escalate to a human. Between agents, check output and retry, ask for clarification or halt.

Testing and validating

Test on a non-production model or agent. Assigning a guardrail changes behavior at once, and Try in Playground needs that assignment. An Annotate and block hit shows the risk and intervention point. Task Adherence's Run test flags each sample and adds a reason when it detects a risk. Red-team the setup and measure again after each change. Use rubrics or a model judge, not exact matches.

EXAM TRAP

With `azd ai agent invoke`, a message argument goes as plain text, so an `inputContentType` of json cannot parse it and it passes unscreened. Send the body as a file.

Latency, streaming and languages

Each intervention point adds about 50 to 100 ms. Default mode buffers output until it is approved. Asynchronous filter mode streams while moderation runs, so content can arrive before a violation is caught. A violating streamed prompt still gets HTTP **400** before any event. The harm classifiers are tested on **8** languages, so check errors in each language you serve.

Key takeaways

- Place controls at user input, tool calls, tool responses and output, not only at the edges.
- A guardrail assigned to an agent fully replaces the model deployment's guardrail.
- Document attack detection is off by default and must be enabled; core-harm and prompt attack filters are on.
- A missing or inert hosted agent policy fails open, so test that blocking really happens.
- Test guardrails on non-production agents and re-measure after every change.

Sources: AI Agent Orchestration Patterns - Azure Architecture Center | Microsoft Learn (opens in a new tab) · Configure guided guardrail set-up for an agent - Azure AI services | Microsoft Learn (opens in a new tab) · How to configure guardrails and controls in Microsoft Foundry - Microsoft Foundry | Microsoft Learn (opens in a new tab) · Agentic Workflows Task Adherence (preview) - Microsoft Foundry | Microsoft Learn (opens in a new tab) · Guardrails and controls overview in Microsoft Foundry - Microsoft Foundry | Microsoft Learn (opens in a new tab) · Hosted agent content safety overview - Microsoft Foundry | Microsoft Learn (opens in a new tab) · Intervention points concepts - Microsoft Foundry | Microsoft Learn (opens in a new tab) · Default safety policies in Microsoft Foundry - Microsoft Foundry | Microsoft Learn (opens in a new tab)

Deploy multi-agent solutions to Azure

Start here

THINK FIRST

How would you move a multi-agent solution from development to production safely, and get back quickly if a release goes wrong?

Why it matters: Questions describe a release, rollback or test need and ask which deployment method, pipeline step or tool fits.

Each environment gets its own isolated resources, and the same tested commit moves forward through them. A canary sends a small share of traffic to a new version, while a staging slot is verified and then swapped into production. Rollback re-runs a known good release.

Key terms

Term	Meaning
Canary deployment	Routing a small part of live traffic to a new candidate version while the stable version serves the rest.
Deployment slot	A staging copy of an app that you verify and then swap into production.
Roll forward	Undoing a change with a new commit and deployment, such as `git revert`.
Mock data	Simulated results that let workflow unit tests run without calling outside services.
Blue-green deployment	A strategy for zero-downtime releases, used with revisions on Functions on Container Apps, where slots aren't available.

Isolating each environment

Bind each environment to its own Foundry project. Package and deploy hosted agent source for each environment, not one shared build. Test the same commit in the test stage with regression and integration tests before production approval. In GitHub Actions, chain dev, test and production jobs, and protect production with environment rules.

EXAM TRAP

Environment names do not isolate Azure resources. Without separate Foundry projects, test and production can share the same resources.

Canary releases for hosted agents

A canary and the stable version share one production endpoint. Use `azd ai agent endpoint update` to change the traffic split without redeploying or making a version. If a canary check fails, stop raising traffic and set the stable version to **100%**. A `400` saying the version is still provisioning means poll until active, then retry.

EXAM TRAP

The `--version` flag on `azd ai agent invoke` picks a version for your test session only. It does not change the traffic percentages.

Deploying hosted agent code

A new version appears only when the code or definition changes. Versioning is content-addressable, so reposting identical code returns the latest version. The service keeps the zip's SHA-256 and returns it on download, which reveals drift. In a pipeline, `--deploy-mode code` selects source deployment and needs `--runtime` and `--entry-point`.

Slots and blue/green on Functions

Deploy to a staging slot, verify it, then swap. To undo a swap, swap again. If any instance fails to restart during a swap, the swap reverts and stops. Hosting plans differ in what they offer.

Release options by Azure Functions hosting choice

Hosting choice	Deployment slots	Safe release approach
Consumption plan	**2**: one production, one staging	Verify in staging, then swap
Flex Consumption plan	Not supported	Rolling updates; a separate app as staging
Azure Container Apps	Not available	Revisions with a blue-green strategy

EXAM TRAP

A slot swap does not guarantee zero downtime. For zero-downtime Functions releases, use the Flex Consumption plan with rolling updates.

Logic Apps Standard slots

Workflows run only in the production slot. The dispatcher is off in other slots, and traffic distribution is disabled. Slots still let you roll back to an earlier version. Deploying from VS Code overwrites the slot's content. Swap with preview needs deployment slot settings enabled. Behind a private-endpoint storage firewall, create slots with Bicep or ARM.

EXAM TRAP

Standard logic app deployment slots do not support blue-green deployment. Plan another release method if you need it.

Rolling back a release

Roll back by re-running a previous successful pipeline run. The re-run checks out that run's original commit, not the branch head. It reuses the old YAML, but secrets and variables take current values. It does not restore outside resource settings, downstream data or Key Vault secrets. A postdeploy health check can trigger this re-run automatically.

EXAM TRAP

A `git revert` is not a rollback. It creates a new commit and a new deployment, so it is a roll forward.

Unit testing workflows

Logic Apps unit tests use mocks instead of real services. Create a `UnitTestExecutor` and run `RunWorkflowAsync()`. `Setup()` makes a new executor before each test. The mock file's `triggerMocks` section is required to start a run. Building tests from runs needs a locally executed workflow and skips Data Mapper, custom code, XML and Liquid actions.

EXAM TRAP

Workflow unit tests do not call real actions. Only C# is supported, and every action in the execution path must be mocked.

CI/CD and infrastructure as code

Bicep replaces all app settings on each deployment. It needs no cleanup step, and what-if detects drift. Wait **30** seconds after deploying settings, because the app restarts and code deploys can fail. OIDC uses a managed identity, so GitHub stores no secrets. The Logic Apps extension generates ARM-based pipelines, but you connect them to Azure DevOps.

EXAM TRAP

A Standard logic app's `local.settings.json` is not deployed. Its settings apply only to local runs, and Azure ignores the file.

Key takeaways

- Give every environment its own Foundry project; names alone do not isolate resources.
- Undo a failed canary by setting the stable version back to **100%** of traffic.
- Re-running a previous successful run is a rollback; `git revert` is a roll forward.
- Slot swaps do not guarantee zero downtime on Azure Functions.
- Logic Apps unit tests need every action in the execution path mocked.

Sources: Promote hosted agents to production with the Azure Developer CLI - Microsoft Foundry | Microsoft Learn (opens in a new tab) · Continuous deployment for Azure Functions | Microsoft Learn (opens in a new tab) · Recover a deployment to a Flex Consumption plan in Azure Functions | Microsoft Learn (opens in a new tab) · Create Unit Tests from Standard Workflow Runs with Visual Studio Code - Azure Logic Apps | Microsoft Learn (opens in a new tab) · Deploy a hosted agent from source code - Microsoft Foundry | Microsoft Learn (opens in a new tab) · Azure Functions deployment slots | Microsoft Learn (opens in a new tab) · Set Up Deployment Slots for Zero Downtime - Azure Logic Apps | Microsoft Learn (opens in a new tab) · Azure Logic Apps Automated Test SDK - Azure Logic Apps | Microsoft Learn (opens in a new tab)

MockCert

Take a Microsoft Certified: Multi-Agent AI Solutions Expert mock exam the way the real one feels

5 timed mock exams · 60 questions in 100 minutes each · one is free

- Free to start: study notes, explained quizzes, flashcards and the first mock exam
- Timed mock exams built to the official exam guide
- A verifiable Certificate of Assessment when you pass a premium mock

Explore Microsoft Certified: Multi-Agent AI Solutions Expert on M...



Open it on your phone

mockcert.com/microsoft-ai-500/

Scan the code with your phone's camera.