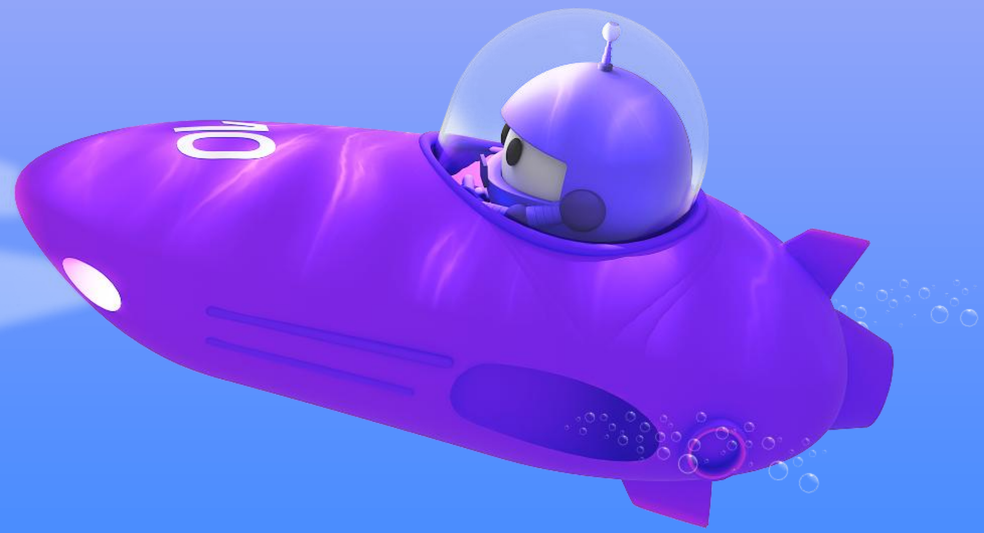
DEVCafe
CONNECT AND SHARE

.NET

Multi-Agent Workflows in .NET with Microsoft Agent Framework, A2A, MCP and AG-UI

Thang Chung – Microsoft Azure MVP

29 November 2025



Agenda

01 What’s the problem with building an enterprise application?

02 Workflow types

03 Microsoft Agent Framework workflow

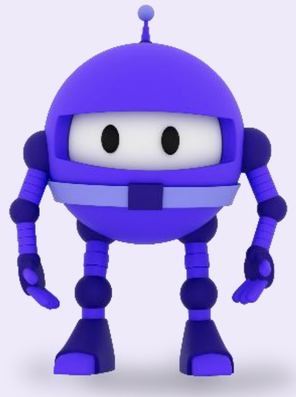
04 DEMO

05 Appendix: GenAI Reference architecture

About me

- I’m a humble Solution Architect for the Tiger Tribe company.
- Seven-time consecutive Microsoft MVP, honored in the areas of Azure Innovation Hub and Web Development. My public profile: https://mvp.microsoft.com/en-US/mvp/profile/9f03f4fd-be27-ea11-a810-000d3a8ccb3e
- Creator of Vietnam Microservices Group on Facebook (20k+ members): https://www.facebook.com/groups/645391349250568
- Professional Experience: Over 18 years of proven expertise in software consulting, architecture, development, and deployment across outsourcing firms, product companies, and startups.
- Technical Specialization: Deep expertise in cloud computing, cloud-native platforms, serverless architectures, WebAssembly/WASI, and Agentic AI systems.
- Connect with Me:
  - GitHub: https://github.com/thangchung
  - Blog: https://dev.to/thangchung
  - LinkedIn: https://www.linkedin.com/in/thangchungatwork

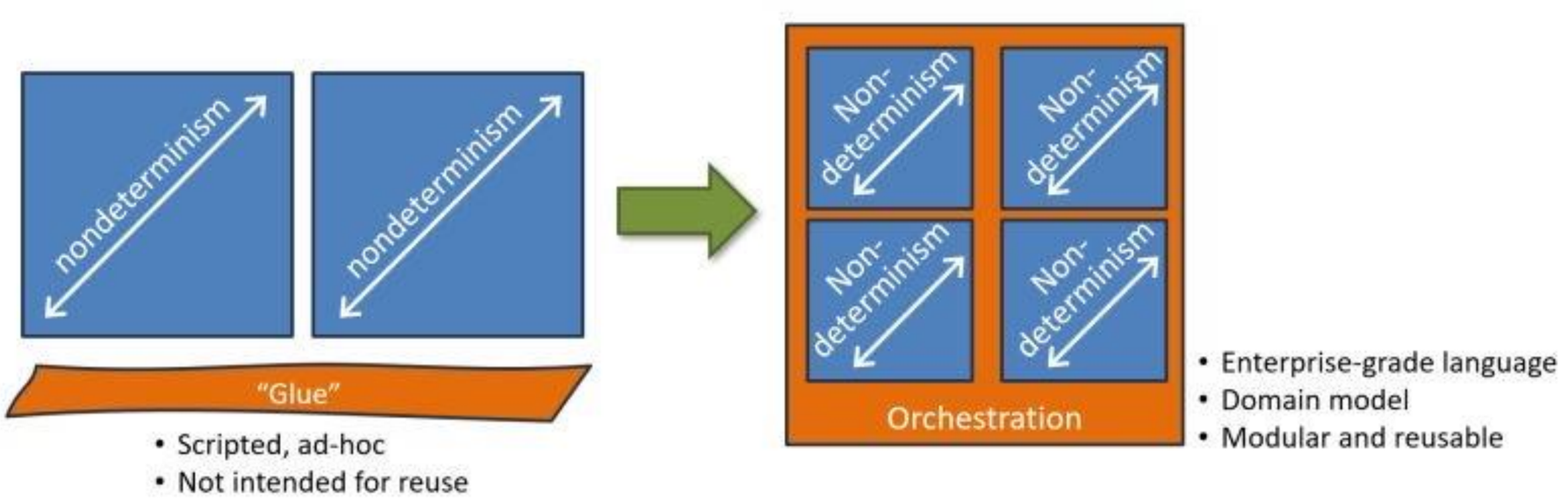
01 What’s the problem with building an enterprise application?



Enterprise application builder:

How to deliver *intelligent reasoning without sacrificing reliability, traceability, and operational control*?

GenAI: From Lighthouse to Enterprise-Grade



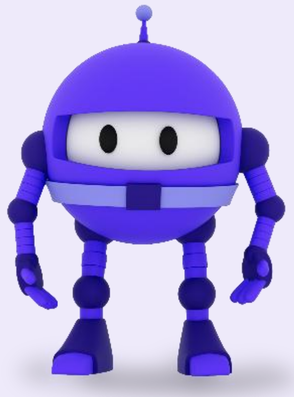
Gregor Hohpe: https://www.linkedin.com/posts/ghohpe_genai-embabel-goto-activity-7379119150205632512-3xb2

Workflow (i.e., orchestration) - deterministic: structure, type safety, and fault tolerance

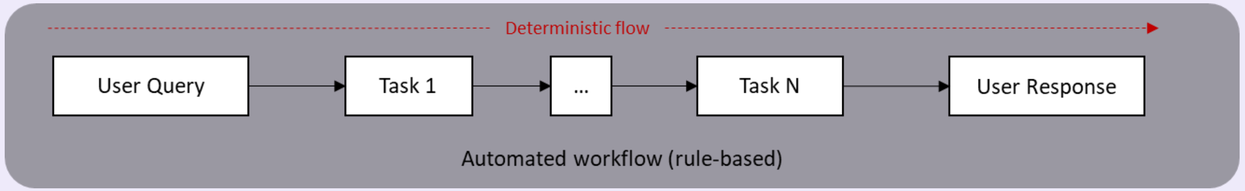
AI Agents – non-deterministic: domain-specific reasoning and contextual insight

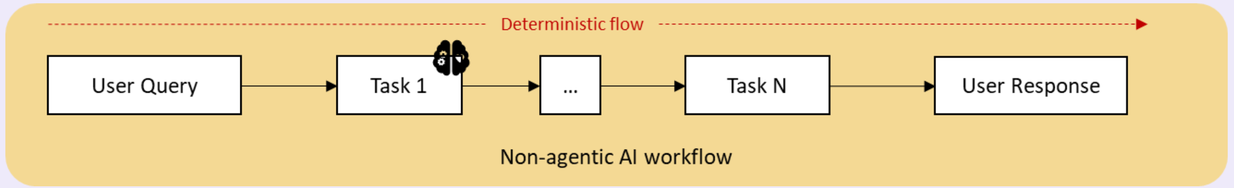
→ LLM + Agent Architecture + Prompt + Knowledge + Tools

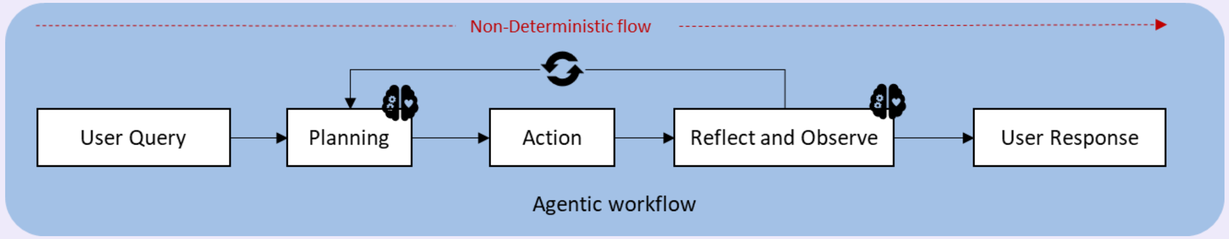
02 Workflow types

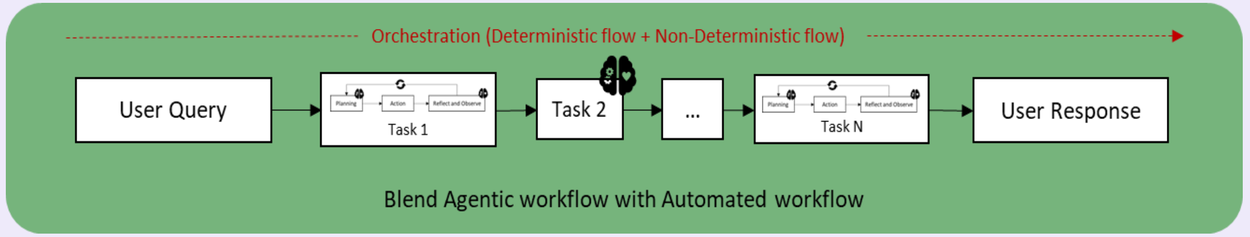


Workflow Types

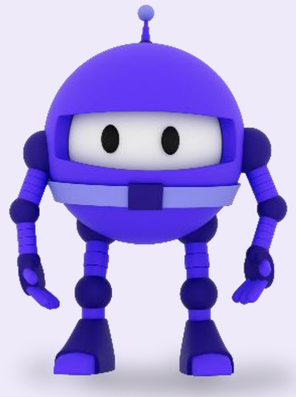






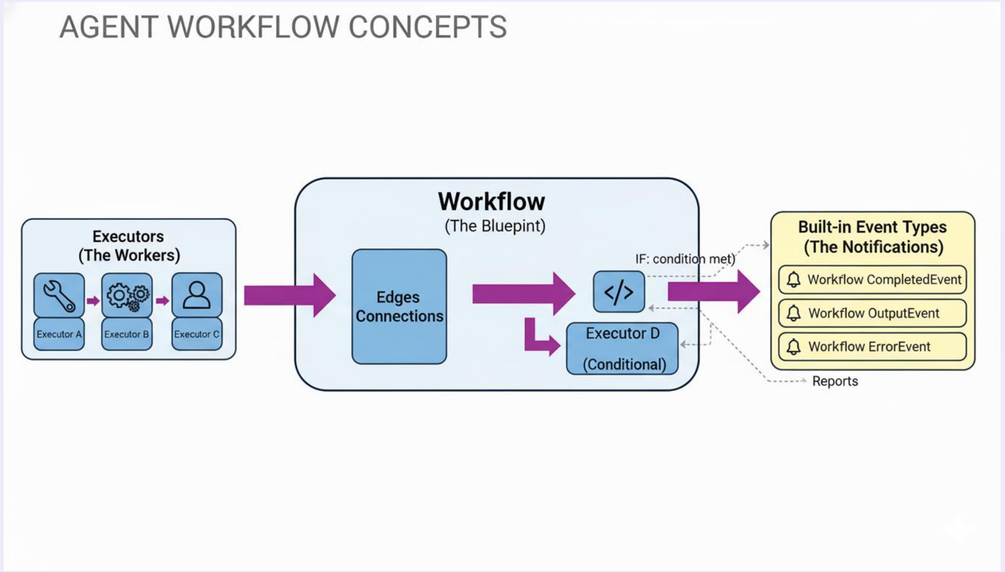


03 Microsoft Agent Framework workflow



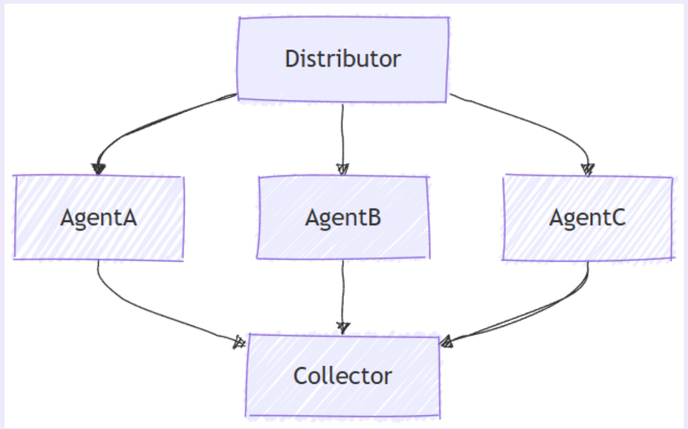
Microsoft Agent Framework a.k.a MAF – Core components

- **Executors**: The workers in a workflow (AI agents or custom code) that receive input, do a task, and produce output.
- **Edges**: The connections between executors that define message flow and can add conditions for routing.
- **Workflows**: The overall blueprint (a directed graph) of executors and edges, showing the process from start to finish.
- **Built-in Event Types**: Notifications that signal specific moments or changes (like start, completion, or error) in the workflow or executor's lifecycle.



https://learn.microsoft.com/en-us/agent-framework/user-guide/workflows/overview

MAF – Workflow orchestration - Concurrent

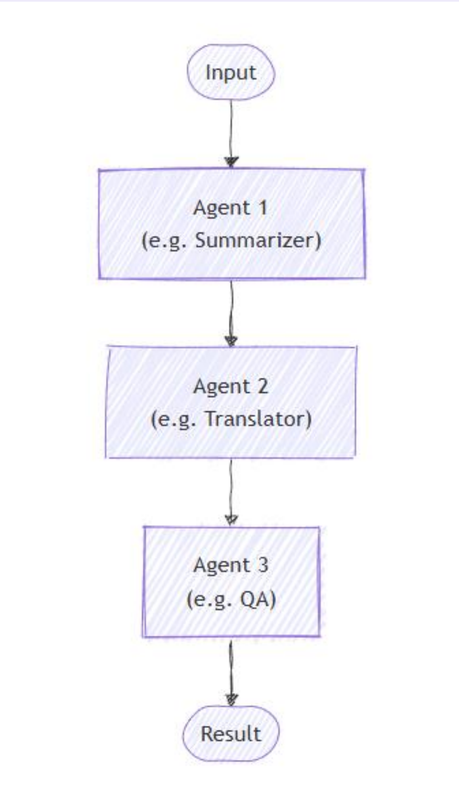


- Well-suited for scenarios where diverse perspectives or solutions are valuable, such as brainstorming, ensemble reasoning, or voting systems.

```csharp
// 1) Set up the Azure OpenAI client
var endpoint = Environment.GetEnvironmentVariable("AZURE_OPENAI_ENDPOINT") ??
    throw new InvalidOperationException("AZURE_OPENAI_ENDPOINT is not set.");
var deploymentName = Environment.GetEnvironmentVariable("AZURE_OPENAI_DEPLOYMENT_NAME") ?? "gpt-4o-mini";
var client = new AzureOpenAIClient(new Uri(endpoint), new AzureCliCredential())
    .GetChatClient(deploymentName)
    .AsIChatClient();

// 2) Helper method to create translation agents
static ChatClientAgent GetTranslationAgent(string targetLanguage, IChatClient chatClient) =>
    new(chatClient,
        $"You are a translation assistant who only responds in {targetLanguage}. Respond to any " +
        $"input by outputting the name of the input language and then translating the input to
{targetLanguage}.");

// Create translation agents for concurrent processing
var translationAgents = (from lang in (string[])["French", "Spanish", "English"]
                         select GetTranslationAgent(lang, client));

// 3) Build concurrent workflow
var workflow = AgentWorkflowBuilder.BuildConcurrent(translationAgents);

// 4) Run the workflow
var messages = new List<ChatMessage> { new(ChatRole.User, "Hello, world!") };

StreamingRun run = await InProcessExecution.StreamAsync(workflow, messages);
await run.TrySendMessageAsync(new TurnToken(emitEvents: true));

List<ChatMessage> result = new();
await foreach (WorkflowEvent evt in run.WatchStreamAsync().ConfigureAwait(false))
{
    if (evt is AgentRunUpdateEvent e)
    {
        Console.WriteLine($"{e.ExecutorId}: {e.Data}");
    }
    else if (evt is WorkflowCompletedEvent completed)
    {
        result = (List<ChatMessage>)completed.Data!;
        break;
    }
}
```

https://learn.microsoft.com/en-us/agent-framework/user-guide/workflows/orchestrations/concurrent?pivots=programming-language-csharp

MAF – Workflow orchestration - Sequential



- Ideal for workflows where each step builds upon the previous one, such as document review, data processing pipelines, or multi-stage reasoning

```csharp
// 1) Set up the Azure OpenAI client
var endpoint = Environment.GetEnvironmentVariable("AZURE_OPENAI_ENDPOINT") ??
    throw new InvalidOperationException("AZURE_OPENAI_ENDPOINT is not set.");
var deploymentName = Environment.GetEnvironmentVariable("AZURE_OPENAI_DEPLOYMENT_NAME") ?? "gpt-4o-mini";
var client = new AzureOpenAIClient(new Uri(endpoint), new AzureCliCredential())
    .GetChatClient(deploymentName)
    .AsIChatClient();

// 2) Helper method to create translation agents
static ChatClientAgent GetTranslationAgent(string targetLanguage, IChatClient chatClient) =>
    new(chatClient,
        $"You are a translation assistant who only responds in {targetLanguage}. Respond to any " +
        $"input by outputting the name of the input language and then translating the input to {targetLanguage}.");

// Create translation agents for sequential processing
var translationAgents = (from lang in (string[])["French", "Spanish", "English"]
                         select GetTranslationAgent(lang, client));

// 3) Build sequential workflow
var workflow = AgentWorkflowBuilder.BuildSequential(translationAgents);

// 4) Run the workflow
var messages = new List<ChatMessage> { new(ChatRole.User, "Hello, world!") };

StreamingRun run = await InProcessExecution.StreamAsync(workflow, messages);
await run.TrySendMessageAsync(new TurnToken(emitEvents: true));

List<ChatMessage> result = new();
await foreach (WorkflowEvent evt in run.WatchStreamAsync().ConfigureAwait(false))
{
    if (evt is AgentRunUpdateEvent e)
    {
        Console.WriteLine($"{e.ExecutorId}: {e.Data}");
    }
    else if (evt is WorkflowCompletedEvent completed)
    {
        result = (List<ChatMessage>)completed.Data!;
        break;
    }
}
```

https://learn.microsoft.com/en-us/agent-framework/user-guide/workflows/orchestrations/sequential?pivots=programming-language-csharp

MAF – Workflow orchestration – Group Chat

- Group chat orchestration models a collaborative conversation among multiple agents, coordinated by a manager that determines speaker selection and conversation flow
- Characteristics:

  –Centralized Coordination: Unlike handoff patterns where agents directly transfer control, group chat uses a manager to coordinate who speaks next

  –Iterative Refinement: Agents can review and build upon each other's responses in multiple rounds

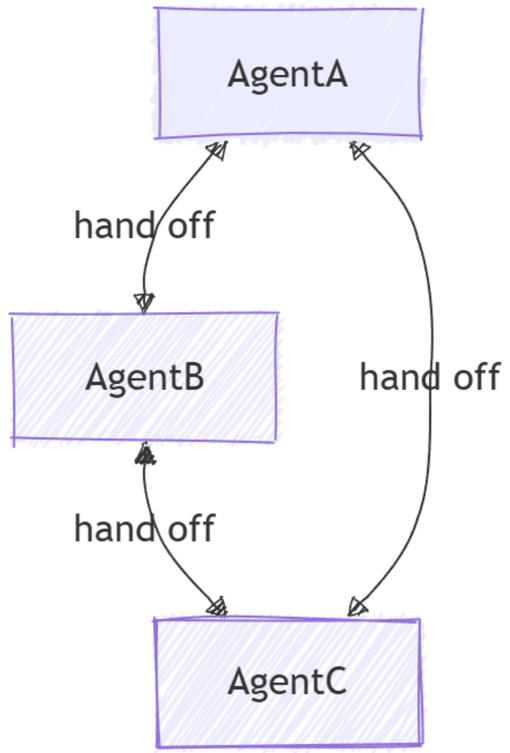
  –Flexible Speaker Selection: The manager can use various strategies (round-robin, prompt-based, custom logic) to select speakers

  –Shared Context: All agents see the full conversation history, enabling collaborative refinement
- Ideal for scenarios requiring iterative refinement, collaborative problem-solving, or multi-perspective analysis

```csharp
// Set up the Azure OpenAI client
var endpoint = Environment.GetEnvironmentVariable("AZURE_OPENAI_ENDPOINT") ??
    throw new InvalidOperationException("AZURE_OPENAI_ENDPOINT is not set.");
var deploymentName = Environment.GetEnvironmentVariable("AZURE_OPENAI_DEPLOYMENT_NAME") ?? "gpt-4o-mini";
var client = new AzureOpenAIClient(new Uri(endpoint), new AzureCliCredential())
    .GetChatClient(deploymentName)
    .AsIChatClient();

// Create a copywriter agent
ChatClientAgent writer = new(client,
    "You are a creative copywriter. Generate catchy slogans and marketing copy. Be concise and impactful.",
    "CopyWriter",
    "A creative copywriter agent");

// Create a reviewer agent
ChatClientAgent reviewer = new(client,
    "You are a marketing reviewer. Evaluate slogans for clarity, impact, and brand alignment. " +
    "Provide constructive feedback or approval.",
    "Reviewer",
    "A marketing review agent");

// Build group chat with round-robin speaker selection
// The manager factory receives the list of agents and returns a configured manager
var workflow = AgentWorkflowBuilder
    .CreateGroupChatBuilderWith(agents =>
        new RoundRobinGroupChatManager(agents)
        {
            MaximumIterationCount = 5  // Maximum number of turns
        })
    .AddParticipants(writer, reviewer)
    .Build();

// Start the group chat
var messages = new List<ChatMessage> {
    new(ChatRole.User, "Create a slogan for an eco-friendly electric vehicle.")
};

StreamingRun run = await InProcessExecution.StreamAsync(workflow, messages);
await run.TrySendMessageAsync(new TurnToken(emitEvents: true));

await foreach (WorkflowEvent evt in run.WatchStreamAsync().ConfigureAwait(false))
{
    if (evt is AgentRunUpdateEvent update)
    {
        // Process streaming agent responses
        AgentRunResponse response = update.AsResponse();
        foreach (ChatMessage message in response.Messages)
        {
            Console.WriteLine($"[{update.ExecutorId}]: {message.Text}");
        }
    }
    else if (evt is WorkflowOutputEvent output)
    {
        // Workflow completed
        var conversationHistory = output.As<List<ChatMessage>>();
        Console.WriteLine("\n=== Final Conversation ===");
        foreach (var message in conversationHistory)
        {
            Console.WriteLine($"{message.AuthorName}: {message.Text}");
        }
        break;
    }
}
```

https://learn.microsoft.com/en-us/agent-framework/user-guide/workflows/orchestrations/group-chat?pivots=programming-language-csharp

MAF – Workflow orchestration - Handoff



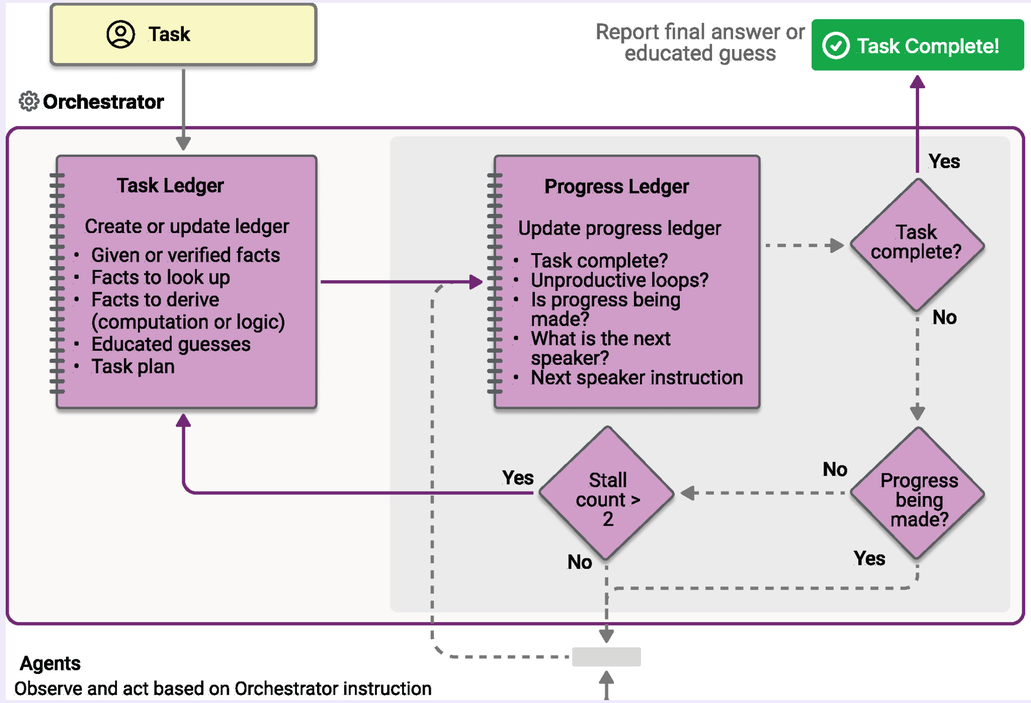
- Useful in customer support, expert systems, or any scenario requiring dynamic delegation

```csharp
// 1) Set up the Azure OpenAI client
var endpoint = Environment.GetEnvironmentVariable("AZURE_OPENAI_ENDPOINT") ??
    throw new InvalidOperationException("AZURE_OPENAI_ENDPOINT is not set.");
var deploymentName = Environment.GetEnvironmentVariable("AZURE_OPENAI_DEPLOYMENT_NAME") ?? "gpt-4o-mini";
var client = new AzureOpenAIClient(new Uri(endpoint), new AzureCliCredential())
    .GetChatClient(deploymentName)
    .AsIChatClient();

// 2) Create specialized agents
ChatClientAgent historyTutor = new(client,
    "You provide assistance with historical queries. Explain important events and context clearly. Only respond
about history.",
    "history_tutor",
    "Specialist agent for historical questions");

ChatClientAgent mathTutor = new(client,
    "You provide help with math problems. Explain your reasoning at each step and include examples. Only respond
about math.",
    "math_tutor",
    "Specialist agent for math questions");

ChatClientAgent triageAgent = new(client,
    "You determine which agent to use based on the user's homework question. ALWAYS handoff to another agent.",
    "triage_agent",
    "Routes messages to the appropriate specialist agent");

// 3) Build handoff workflow with routing rules
var workflow = AgentWorkflowBuilder.StartHandoffWith(triageAgent)
    .WithHandoffs(triageAgent, [mathTutor, historyTutor]) // Triage can route to either specialist
    .WithHandoff(mathTutor, triageAgent)                  // Math tutor can return to triage
    .WithHandoff(historyTutor, triageAgent)               // History tutor can return to triage
    .Build();

// 4) Process multi-turn conversations
List<ChatMessage> messages = new();

while (true)
{
    Console.Write("Q: ");
    string userInput = Console.ReadLine()!;
    messages.Add(new(ChatRole.User, userInput));

    // Execute workflow and process events
    StreamingRun run = await InProcessExecution.StreamAsync(workflow, messages);
    await run.TrySendMessageAsync(new TurnToken(emitEvents: true));

    List<ChatMessage> newMessages = new();
    await foreach (WorkflowEvent evt in run.WatchStreamAsync().ConfigureAwait(false))
    {
        if (evt is AgentRunUpdateEvent e)
        {
            Console.WriteLine($"{e.ExecutorId}: {e.Data}");
        }
        else if (evt is WorkflowCompletedEvent completed)
        {
            newMessages = (List<ChatMessage>)completed.Data!;
            break;
        }
    }

    // Add new messages to conversation history
    messages.AddRange(newMessages.Skip(messages.Count));
}
```

https://learn.microsoft.com/en-us/agent-framework/user-guide/workflows/orchestrations/handoff?pivots=programming-language-csharp

MAF – Workflow orchestration - Magentic

- Magentic orchestration is designed based on the Magentic-One system invented by AutoGen.
- It is a flexible, general-purpose multi-agent pattern designed for complex, open-ended tasks that require dynamic collaboration.
- A dedicated Magentic manager coordinates a team of specialized agents, selecting which agent should act next based on the evolving context, task progress, and agent capabilities.
- Coming soon for .NET/C# version

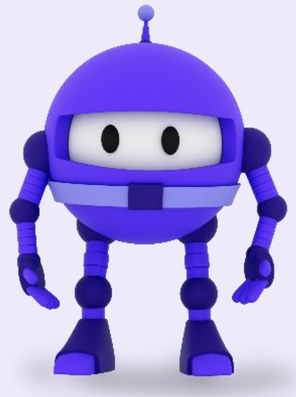


https://learn.microsoft.com/en-us/agent-framework/user-guide/workflows/orchestrations/magentic?pivots=programming-language-csharp

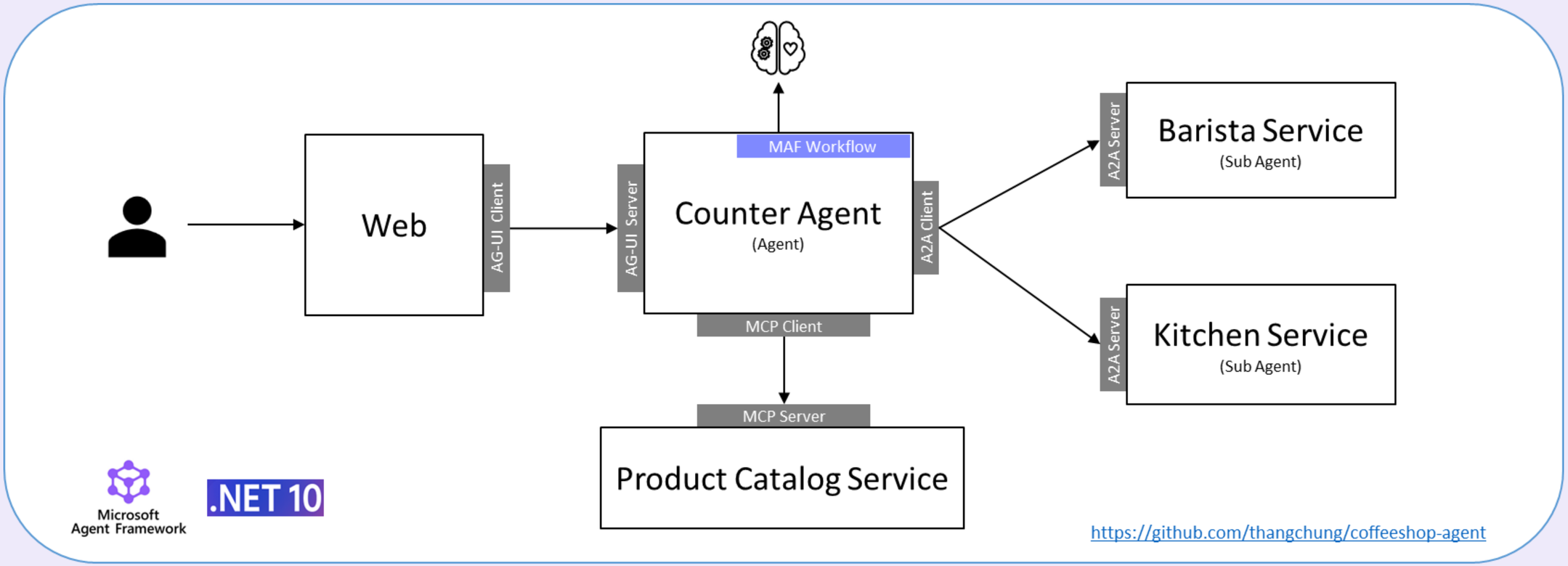
MAF- Other patterns and supports

- Workflow as Agents
- Requests and Responses
- Shared States
- Checkpoints
- Observability
- Visualization (devui)

04 DEMO



Smart CoffeeShop Agent

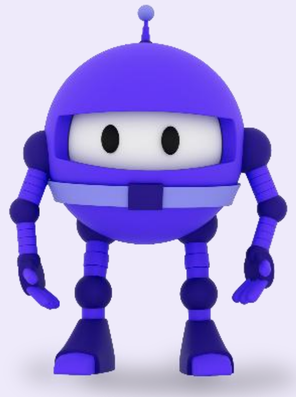


MCP gives agent tools

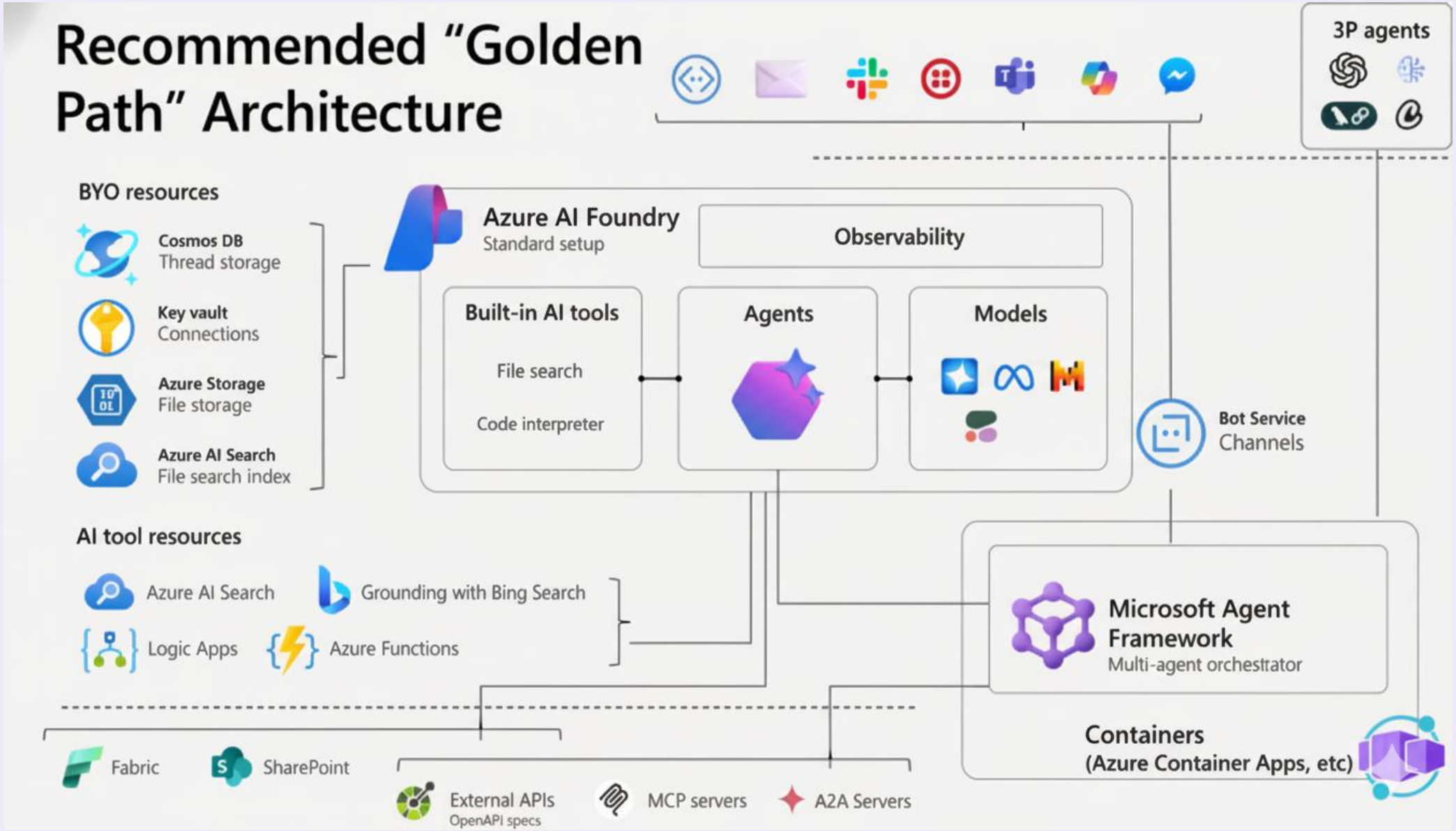
A2A allows agents to communicate with other agents

AG-UI brings agents into user-facing applications

05 *Appendix*: GenAI Reference architecture



Appendix: GenAI Reference Architecture - Microsoft



Thank you