OpenAI Agents SDK - Open Source

Playlist link
https://www.youtube.com/playlist?list=PL0vKVrkG4hWovpr0FX6Gs-06hfsPDEUe6

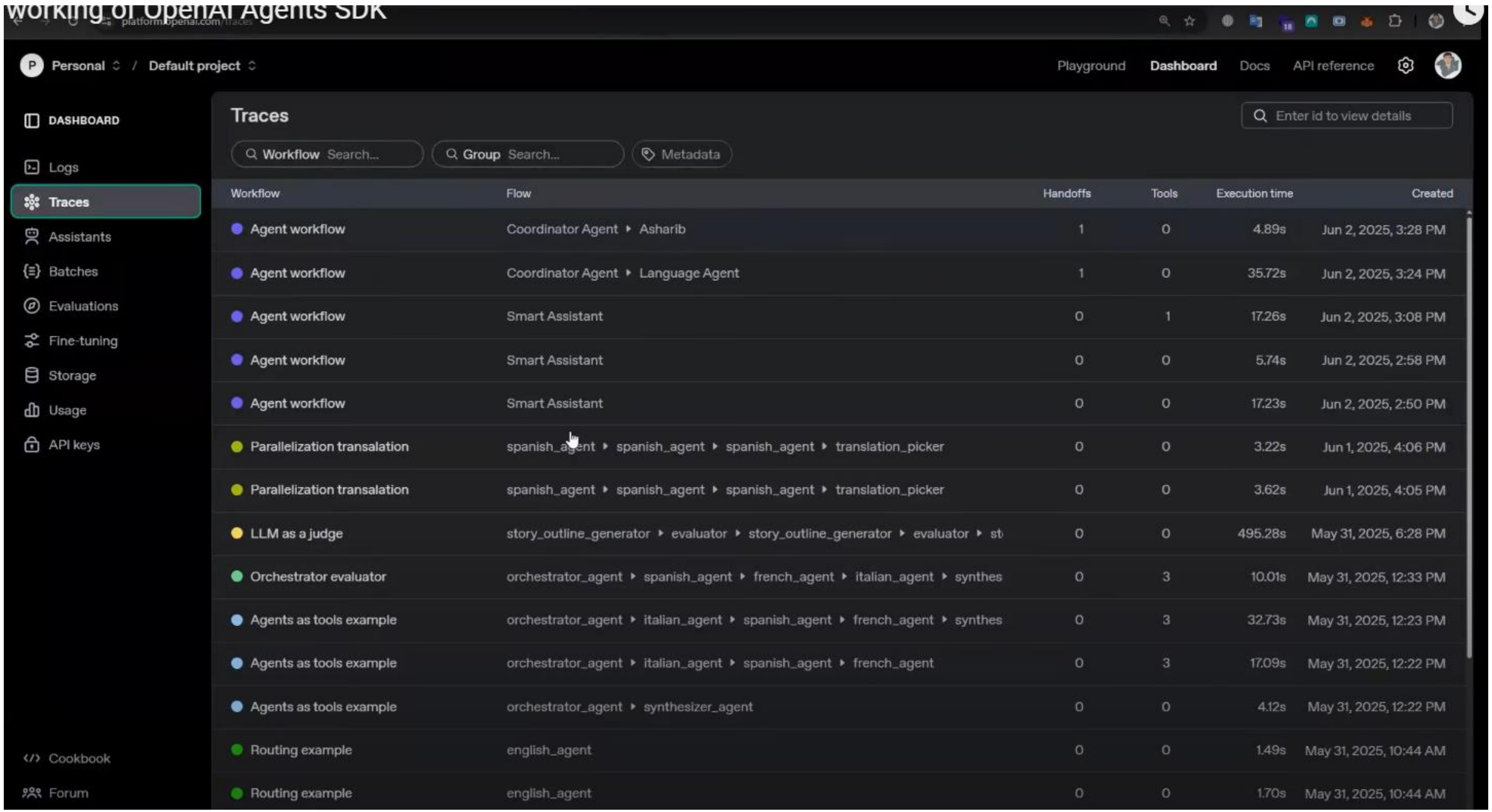
Inner working of OpenAI Agents SDK | Deep Dive

- Sir Asharib class link ➔ https://www.youtube.com/live/5RIADVKVEd8 (unlisted)
- Complete video link ➔ https://www.youtube.com/watch?v=Xkg6JBUFkPY
- Sir made guide for inner working ➔ https://github.com/AsharibAli/agentic-ai-projects/blob/main/openai_agents_sdk.md

Core Concepts of Agent

- To see **tracing** (while you are using openAI API) go to this link https://platform.openai.com/docs/overview ➔ dashboard ➔ tracing section, here you will see the entire workflow of agents you run till date.
- When you used openai models (which are paid) to make agents, you will get tracing functionality by default on above link
- If you used Gemini or any other model in openai sdk, you will need to used some tracing provider in order to trace it.

Core Concepts of Agent



Tools

- Tools are simple python functions which will build to implement custom logic
- Litellm is simple which helps to access different provider tools, like Gemini, Anthropic etc.
- OpenAI called their latest APIs as:
  - Response API ➔ https://platform.openai.com/docs/api-reference/responses
  - Completion API ➔ https://platform.openai.com/docs/guides/completions
  - Assistant API ➔ https://platform.openai.com/docs/assistants/

Open AI SDK

What is the OpenAI SDK?

- A **Software Development Kit** that simplifies integration with OpenAI’s APIs.
- Offers tools and abstractions to interact with models like **GPT-4, DALL-E, Whisper**, etc.
- Enables rapid development of AI-powered applications in various programming environments.
- Release on 11th March 2025 by OpenAI

Open AI SDK

Key Features

- Easy-to-use API client in popular languages (Python, JavaScript, etc.)
- Supports:
    - Chat Completions (GPT models)
    - Image Generation (DALL·E)
    - Audio Transcription (Whisper)
    - Embeddings
    - File and fine-tuning operations
- Built-in support for streaming, error handling, and retries.
- Python-First Design
- Built-in Agent Loop
- Interoperability
- Simplified Multi-Agent Workflows
- Real-World Applications

Open AI SDK

Why Use It?

- Speeds up development with minimal boilerplate code.
- Officially maintained and optimized by OpenAI.
- Ensures compatibility with OpenAI's latest features and API changes.

How to Get Started

- Install via package manager (e.g., pip install openai)
- Authenticate with API key
- Use pre-built functions for common tasks

Core Concepts : The Power of Simplicity in Design

The Agents SDK is designed to be **easy to use** but also **powerful**. It’s built around 4 main parts:

1. **Agents**

Smart AI assistants (LLMs) with instructions, tools (like web search), and safety rules. They understand tasks and take actions.

- Pre-built AI models with custom instructions and tools.
- Can understand tasks and respond intelligently.
- Follow built-in safety rules for reliable output.

Core Concepts : The Power of Simplicity in Design

2. Handoffs

Agents can pass tasks to other agents if needed—like teamwork between AI assistants.

- Agents can transfer tasks to other, more specialized agents.
- Enables teamwork between multiple agents.
- Helps handle complex or multi-step tasks more efficiently.

3. Guardrails

Safety checks that control what agents can say or do, helping avoid mistakes or risks.

- Safety checks on what the agent can say or do.
- Prevents harmful, incorrect, or risky outputs.
- Keeps the AI working within set boundaries.

Core Concepts : The Power of Simplicity in Design

4. Tracing & Observability

Helps you see what the agent is doing, step by step—great for debugging and improving your app.

- Shows a clear step-by-step view of what the agent is doing.
- Helps developers debug, monitor, and improve performance.
- Makes it easier to understand and control agent behavior.

OpenAI's SWARM framework

Origins & Concept

- The idea of multi-agent collaboration has been explored in AI research for years.
- OpenAI began experimenting with agent collaboration to handle complex, multi-step tasks more efficiently.
- Initial concept of **SWARM** emerged in 2023, around the same time OpenAI introduced its Agents and tool-use features.
- Internally, OpenAI ran experiments where multiple agents (GPT-based) worked together to solve tasks like coding, planning, and research.
- The term **"SWARM"** became more publicly known in **late 2023 to early 2024** through: Research talks, OpenAI demonstrations, Community interest in **multi-agent collaboration**
- SWARM is still **experimental**, not yet a formal product.

OpenAI’s SWARM framework

What is SWARM?

- SWARM is a new experimental framework from OpenAI for building multi-agent systems.
- It lets multiple AI agents work together like a team to solve complex tasks.
- Inspired by how humans collaborate, using division of labor, communication, and coordination.

Key Goals

- **Scalability:** Break large tasks into smaller subtasks for agents to handle.
- **Specialization:** Use different agents with different skills.
- **Autonomy + Collaboration:** Agents work independently but coordinate as a team.

OpenAI’s SWARM framework

How SWARM Works

OpenAI’s experimental Swarm framework is meant more as an educational tool than a production-ready system—but it introduces several key design patterns for orchestrating multi-agent systems. In Swarm, agents aren’t monolithic; instead, they’re designed with specialized roles and communicate through clearly defined patterns. Here are some of the core design patterns:

1. Prompt Chaining (Chain Workflow):

This pattern involves breaking down complex tasks into a sequence of simpler, manageable steps, where each step builds upon the previous one. The Agents SDK supports this by allowing developers to define agents that execute specific functions in a predetermined order, ensuring a structured approach to task completion.

2. Routing:

Routing entails directing tasks to the most appropriate agent based on the task's nature. The Agents SDK facilitates this through its handoff mechanism, enabling agents to transfer control to other agents better suited to handle specific subtasks, thereby optimizing task management.

OpenAI’s SWARM framework

3. Parallelization:

This pattern focuses on executing multiple subtasks concurrently to enhance efficiency. With the Agents SDK, developers can design agents that operate in parallel, leveraging the SDK's orchestration capabilities to manage simultaneous processes effectively.

4. Orchestrator-Workers:

In this design, an orchestrator agent decomposes a complex task into smaller subtasks and assigns them to worker agents. The Agents SDK's architecture supports this by allowing an orchestrator agent to oversee the workflow and delegate tasks to specialized worker agents, ensuring coordinated task execution.

5. Evaluator-Optimizer:

This pattern involves iterative improvement through feedback loops, where an evaluator agent assesses the performance of other agents and suggests optimizations. The Agents SDK's guardrails feature enables the implementation of such evaluative mechanisms, allowing for continuous performance enhancement and adherence to desired behaviors.

OpenAI's SWARM framework

Core Components

Component	Description
Agent	Language models with specific instructions or skills.
Tasks	Jobs assigned to agents, can be split or passed.
Workspace	Shared space where agents read/write info (like a whiteboard).
Tools	External actions agents can use (e.g., web, code, memory).
Communication	Messaging system between agents to share progress or ask for help.

Benefits of SWARM

- **Handles complex, multi-step problems**
- **Encourages AI teamwork**
- Easier to **debug and understand** workflows
- Supports **modular development** (add or replace agents as needed)

OpenAI’s SWARM framework

Example Use Cases

- Research assistants collaborating on a report.
- Customer service agents working together on complex tickets.
- Writing, coding, or brainstorming as a group of specialists

Current Status

- Still in experimental stages (as of 2024–2025)
- Not public yet — demonstrated internally by OpenAI
- Possible future feature in OpenAI’s Agents SDK

UV

uv is a modern Python package manager and build system developed by Astral (formerly part of the pdm project). It is designed to be extremely fast, reliable, and easy to use, and is built in Rust for performance.

Key Features of uv:

- **Ultra-Fast**: Much faster than pip and poetry due to being written in Rust.
- **Unified Tooling**: Acts as a drop-in replacement for pip, virtualenv, and pip-tools.
- **Deterministic Installs**: Ensures reproducible builds by resolving dependencies into lockfiles.
- **PEP 582 Support**: Supports local package installation without virtual environments.

Chainlit

Chainlit is an open-source Python framework designed to build and share conversational AI apps powered by LLMs (Large Language Models). It allows developers to quickly create, test, and deploy AI assistants with a front-end interface — all from Python.

Key Features of uv:

- **UI Built-In:** Auto-generates a chat user interface — no front-end needed.
- **LLM Support**: Works with OpenAI, Hugging Face, LangChain, and others.
- **Fast Prototyping**: Build LLM-powered apps in minutes.
- **Realtime Interaction**: Supports async messages, tool use, and streaming.
- **Developer Tools:** Logs, debugging, and interaction tracing included.

Making first Agent using Gemini API

Learn_agentic_ai/01_ai_agent_first:

- We have covered **step 05_chainlit** and **06_chatbot/chatbot** in it
- We have made chatbot one on google colab and other on VS code/cursor using chainlit
- Below is the link of google colab working

https://colab.research.google.com/drive/1mkYAOwlC0yaV0ho2N2BbMZrgXDRGWnaX#scrollTo=-j2Nfiz_C83g

- We have also worked on hello_agent using chainlit when present in folder

07_streaming

- Streaming means the chatbot shows you its response word-by-word or phrase-by-phrase, almost as it's thinking, instead of making you wait for the entire answer to be calculated and then displayed all at once.

Why is this good?

- ***Faster perception:*** You see a response sooner, which feels faster even if the total time is the same.
- ***More engaging:*** It's more like a conversation with a real person because you're seeing the answer develop.
- ***Handles longer answers better:*** Long responses feel less daunting when they appear gradually.

Check folder and repo for code
Repo link ➔ https://github.com/panaversity/learn-agentic-ai/tree/main/01_ai_agents_first/07_streaming

08_tools

- The OpenAI Agents SDK provides a robust framework for integrating various tools into agents, enabling them to perform tasks such as data retrieval, web searches, and code execution. Here's an overview of the key points regarding tool integration:

Types of Tools:

1. Hosted Tools: These are pre-built tools running on OpenAI's servers, accessible via the [OpenAIResponsesModel]. Examples include:

  - WebSearchTool: Enables agents to perform web searches.
    - Try it in Colab: File Search Tool Example
  - FileSearchTool: Allows retrieval of information from OpenAI Vector Stores.
    - Try it in Colab: Computer Tool Example
  - ComputerTool: Facilitates automation of computer-based tasks.
    - We will use model=computer-use-preview-2025-03-11
    - Note: The model "computer-use-preview" is not available.

2. Function Calling: This feature allows agents to utilize any Python function as a tool, enhancing their versatility.

3. Agents as Tools: Agents can employ other agents as tools, enabling hierarchical task management without transferring control.

Check Repo for detail ➔ https://github.com/panaversity/learn-agentic-ai/tree/main/01_ai_agents_first/08_tools

OPENAI Documentation Github Page

- Below is the link of document

https://openai.github.io/openai-agents-python/

Why use the Agents SDK

The SDK has two driving **design principles**:

1. Enough features to be worth using, but few enough primitives to make it quick to learn.
2. Works great out of the box, but you can customize exactly what happens.

OPENAI Documentation Github Page

Here are the **main features of the SDK**:

- **Agent loop:** Built-in agent loop that handles calling tools, sending results to the LLM, and looping until the LLM is done.
- **Python-first**: Use built-in language features to orchestrate and chain agents, rather than needing to learn new abstractions.
- **Handoffs:** A powerful feature to coordinate and delegate between multiple agents.
- **Guardrails:** Run input validations and checks in parallel to your agents, breaking early if the checks fail.
- **Function tools:** Turn any Python function into a tool, with automatic schema generation and Pydantic-powered validation.
- **Tracing:** Built-in tracing that lets you visualize, debug and monitor your workflows, as well as use the OpenAI suite of evaluation, fine-tuning and distillation tools.

@dataclass

Understanding Python Dataclasses

- Dataclasses, introduced in Python 3.7, are a powerful way to create classes primarily used to store data.
- They reduce boilerplate code often associated with defining classes for data storage, such as __init__, __repr__, __eq__, and __hash__ methods.

Why Use Dataclasses?

- Before dataclasses, you might have used plain classes or namedtuple for data structures. While functional, they often required a lot of repetitive code.
- Dataclasses simplify this by automatically generating common methods based on type hints.

@dataclass

Benefits of Dataclasses:

- **Less Boilerplate:** Automatically generates __init__, __repr__, __eq__, __hash__ (if mutable=False), and __str__.
- **Readability:** Clearly defines the fields and their types, making the code easier to understand.
- **Type Hinting:** Integrates seamlessly with type hints, improving static analysis and code clarity.
- **Mutable by Default:** Unlike namedtuple, dataclasses are mutable by default, but you can make them immutable.
- **Default Values:** Easy to assign default values to fields.

Basic Usage

- To create a dataclass, you import the dataclass decorator from the dataclasses module and apply it to your class.
- We can define methods, class variable and class methods in @dataclass

@dataclass

- Official Code Link ➔ https://github.com/panaversity/learn-agentic-ai/tree/main/00_openai_agents/00_python_syntax
- Code also saved in folder ➔ G:\osamabinadnan_files\giaic\quarter_04\OpenAI_SDK\OpenAISDK_Working_from_YTPlaylist\Video03_Divingin_sourcecodeof_OpenAIAgentsSDK

```python
from dataclasses import dataclass

@dataclass
class Book:
    title: str
    author: str
    pages: int
    price: float

# Creating an instance
book1 = Book("The Great Gatsby", "F. Scott Fitzgerald", 180, 12.99)
print(book1)
print(f"Title: {book1.title}, Author: {book1.author}")

# Dataclasses are mutable by default
book1.price = 10.50
print(book1)
```

@dataclass

- System Prompt vs user Prompt
- https://openai.github.io/openai-agents-python/ref/agent/
- Callable method

```
    name: str
    """The name of the agent."""

    instructions: (
        str
        | Callable[
            [RunContextWrapper[TContext], Agent[TContext]],
            MaybeAwaitable[str],
        ]
        | None
    ) = None
```

Understand behind the scene code of Runner

- RSI ➔ Recursive self improvement, in future agent will improve itself by using it memory
- run, run_sync and run_stream are class level static method which help to run flow of agents
- Runner works on loop, which is called **Agent loop**
- RunResultStreaming is also a @dataclass

Dataclass, Generics and Callable in Python to understand OpenAI Agents SDK Source Code

Dataclass (@dataclass)

- A dataclass is a shortcut for creating classes that store data. It automatically creates things like the __init__ (constructor) and __repr__ (string representation) methods for you.Think of it like this:
- Instead of writing a whole class just to store some variables, you can use @dataclass to make your code shorter and cleaner.

Dataclass, Generics and Callable in Python to understand OpenAI Agents SDK Source Code

Dataclass (@dataclass)

Example:

```python
from dataclasses import dataclass

@dataclass
class User:
    name: str
    age: int
```

Now you can easily create a user:

```python
u = User(name="Alice", age=25)
print(u)  # Output: User(name='Alice', age=25)
```

Dataclass, Generics and Callable in Python to understand OpenAI Agents SDK Source Code

Generics

- Generics let you write code that can work with any type, without being specific about which one. It makes your code reusable and type-safe. Think of it like this:
- You're saying, “I don't care what type it is yet — I’ll fill that in later.”

Dataclass, Generics and Callable in Python to understand OpenAI Agents SDK Source Code

Generics

python Copy Edit

```python
from typing import TypeVar, Generic

T = TypeVar('T')  # This is a placeholder for any type

class Box(Generic[T]):
    def __init__(self, content: T):
        self.content = content
```

Now you can make a box of anything:

python Copy Edit

```python
box1 = Box("Hello")    # Box[str]
box2 = Box(123)        # Box[int]
```

Dataclass, Generics and Callable in Python to understand OpenAI Agents SDK Source Code

Callable

- A Callable is just something you can call like a function. In Python, functions are callables, and so are objects with a __call__ method. Think of it like this:
- If you can do something(), then something is a callable.

Dataclass, Generics and Callable in Python to understand OpenAI Agents SDK Source Code

Callable

python

Copy Edit

```python
from typing import Callable

def greet(name: str) -> str:
    return f"Hello, {name}!"

def use_function(f: Callable[[str], str]):
    print(f("World"))

use_function(greet)  # Output: Hello, World!
```

You're saying: "Give me a function that takes a string and returns a string."

Agent Loop | Tool Call | Hands off | Memory | Guardrails

- Agent is nothing but LLM call
- OpenAI brought ChatCompletionAPI first time
- Then many companies made wrapper on OpenAI's ChatCompletionAPI like LangChain, EasyLLM etc.
- At the bottom is RestAPI ➔ on it ChatCompletion API (which adopted by most of the companies) to chat with LLM ➔ then Langchain
- You just import openai library, change base url, instead to go to openai server, it will go somewhere else (on mentioned URL)
- We can do handoff in LangGraph, AutoGen and in OpenAI Agent SDK as well. Best is OpenAI SDK

Agent Loop | Tool Call | Hands off | Memory | Guardrails

OpenAI Agent Core Concept

- Story begin with LLMs, different companies made it, user asked AGI level questions to them
- OpenAI made it standard at first place when it achieved AGI level
- Whoever (Google, Meta etc.) achieved AGI wrote its SDK.
- It means that they made package in python to talk with LLMs
- Client (Your PC) ➔ send request to model's server
- Your PC send request to model server, the server response it and give you answer.

Agent Loop | Tool Call | Hands off | Memory | Guardrails

- Earlier, this work you can do by making http request restAPI and talk to it
- Then these companies made easy way to used LLMs which they called **SDK**.
- You just need to install python package of SDK; implement it function and ask to your agent.
- The functions of SDK, we can call it ChatCompletion, Assistant API, Responsive API etc.

Agent Loop | Tool Call | Hands off | Memory | Guardrails

A Good Software (OpenAI Agent SDK)

- Good software will be made in layers, otherwise it becomes confusing, same is the case for OpenAI SDK
- First layer is LLM, → on which it wrap with wrapper for example FastAPI for inference → then you call it from client using RestAPI, although we can call in any language but industry is going toward python then we are calling it in python
- LLM → API Server → client (ChatCompletion API which is calling Rest) → Agent SDK

Tool Call

- Tool calling means that an AI agent (like ChatGPT) can use special tools (like a calculator, web search, or code runner) to help it solve problems or get answers.

Example:

- Imagine an AI that can't do math very well in its head.
- So, when you ask it: "What’s 37 x 82?", it says, “Let me use my calculator tool!”.
- It calls a tool (the calculator), gets the answer, then replies to you.

Tool Call

Does LLMs have access of tools schemas, if yes, how it looks like and works??

- Yes, **LLMs (like ChatGPT or other AI agents)** can have access to **tools** via what's called a **tool schema** (also called function schema, API schema, or OpenAPI spec). This tells the LLM **what tools are available**, **how to use them**, and **what inputs/outputs** they take.

What is a Tool Schema?

- A tool schema is like a menu or instruction sheet that tells the LLM:
  - "Here is a tool you can use. This is what it does. Here's how to use it."
- It's usually defined in JSON format, like this (see below picture):

Tool Call

```json
{
  "name": "get_weather",
  "description": "Get the current weather for a given city.",
  "parameters": {
    "type": "object",
    "properties": {
      "city": {
        "type": "string",
        "description": "The name of the city to get the weather for."
      }
    },
    "required": ["city"]
  }
}
```

Tool Call

How Does the LLM Use It?

1. Reads the tool schema during setup or runtime.
2. When you ask something like:

   "What's the weather in Paris?", the LLM thinks:

   "I have a tool called get_weather and it needs a city. Let me call it with city: 'Paris'."
3. It generates this tool call: ➔

```json
{
  "tool_name": "get_weather",
  "arguments": {
    "city": "Paris"
  }
}
```

Tool Call

4. The system (not the LLM itself) executes the tool, gets the result (e.g., 22°C and sunny), and gives that back to the LLM.

5. The LLM then replies:

    "It's 22°C and sunny in Paris right now!"

Where Are These Schemas Used?

- **OpenAI Function Calling**
- **LangChain Tools**
- **AutoGPT / Agentic frameworks**
- **Custom APIs** in enterprise AI setups

Stateless Nature of the API

- LLMs are stateless, it means it don’t have memory to recall last conversation until and unless you made it stateful.
- To make stateful LLMs send all conversation in every request.
- So, OpenAI made OpenAI SDK over ChatCompletionAPI (stateless layer), which remember all session. You don’t need to send all your previous conversion to it.

Agent Loop

An Agent Loop is when an AI agent (like ChatGPT acting like a smart assistant) keeps doing a cycle of:

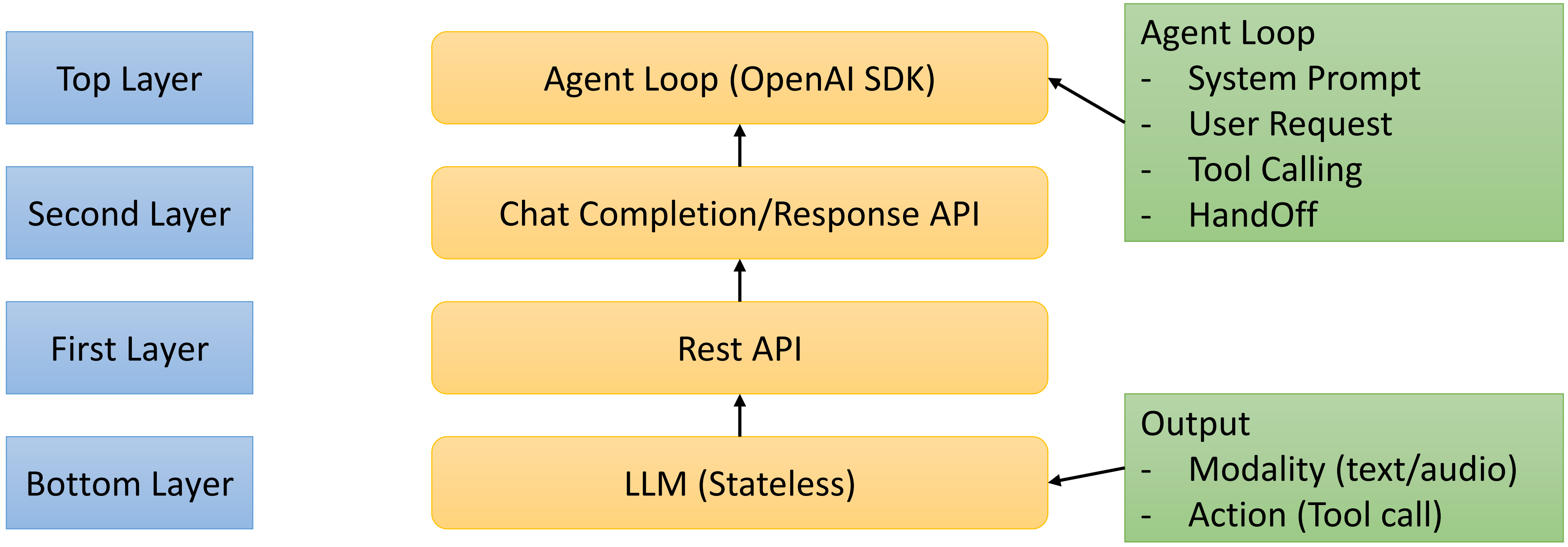
1. Thinking 🧠
2. Taking action (like calling a tool) 🛠️
3. Observing the result 👀
4. Thinking again based on the new result...
5. Repeat 🔁

Until it reaches a final answer or goal ✅.

Thought → Action → Observation → [repeat until done]

Agent Loop

- Agent has short term state, mean all conversation, so whatever task agent will do using tools, it will do by itself.
- Lang chain and Autogen also were also doing the same



Agent Loop / stateless LLMs

- All tools/functions will be called within this loop and all will be done by using OpenAI SDK.
- ***First thing is*** "Tools are called by Agent loop itself, depend on the user query."
- As we set LLMs have stateless protocol, so **how can we make agents which don't remember anything?**
- To resolve this problem, LLMs set that you can proceed 2 types of requests simultaneously, first is called **system prompt** other is **user prompt**.
- So, the ***second thing is***, "There are two types of prompts"

Agent Loop / stateless LLMs

System Prompt

- A **system prompt** is like a set of instructions or rules given to the AI before it starts the conversation.
- Example system prompt:
  - "You are a helpful and friendly assistant who speaks simply and clearly."🧠
- The agent reads this first and keeps it in mind throughout the chat.

User Prompt

- A **user prompt** is what the user says or asks during the conversation.
- Example user prompt:
  - "Can you explain how gravity works in simple terms?"
- The AI uses both the system prompt and user prompt together to decide how to answer.

Agent Loop / stateless LLMs

Do you need to send the system prompt every time in Chat Completion API?

Yes, if you want the AI to remember the instructions.

The Chat Completion API does not store memory between requests. So:

- If you want consistent behavior, you should include the system prompt in every request as the first message (with "role": "system").
- Think of it like reminding the AI: “Here’s who you are again, and now here’s what the user wants.”
- ***Third thing*** is, “function calling”
- All these three things are **abstract** (hide) in OpenAI SDK.

Handoff

- In the OpenAI SDK, especially in the context of multi-agent workflows or tool-using agents, a handoff means passing control or a task from one agent (or function/tool) to another based on logic or capability.
- **Handoff** = When one agent (or function within an agent) **decides it can't complete a task** and **delegates it** — either to:
  - A human (human handoff),
  - Another AI agent (agent-to-agent handoff),
  - Or a tool/function (tool/function handoff).
- Agent is the combination of system prompt, user prompt, tool description/calling.
- Handoff is done in tool calling and handoff itself is a kind of tool calling but it abstraction is separate.

Handoff

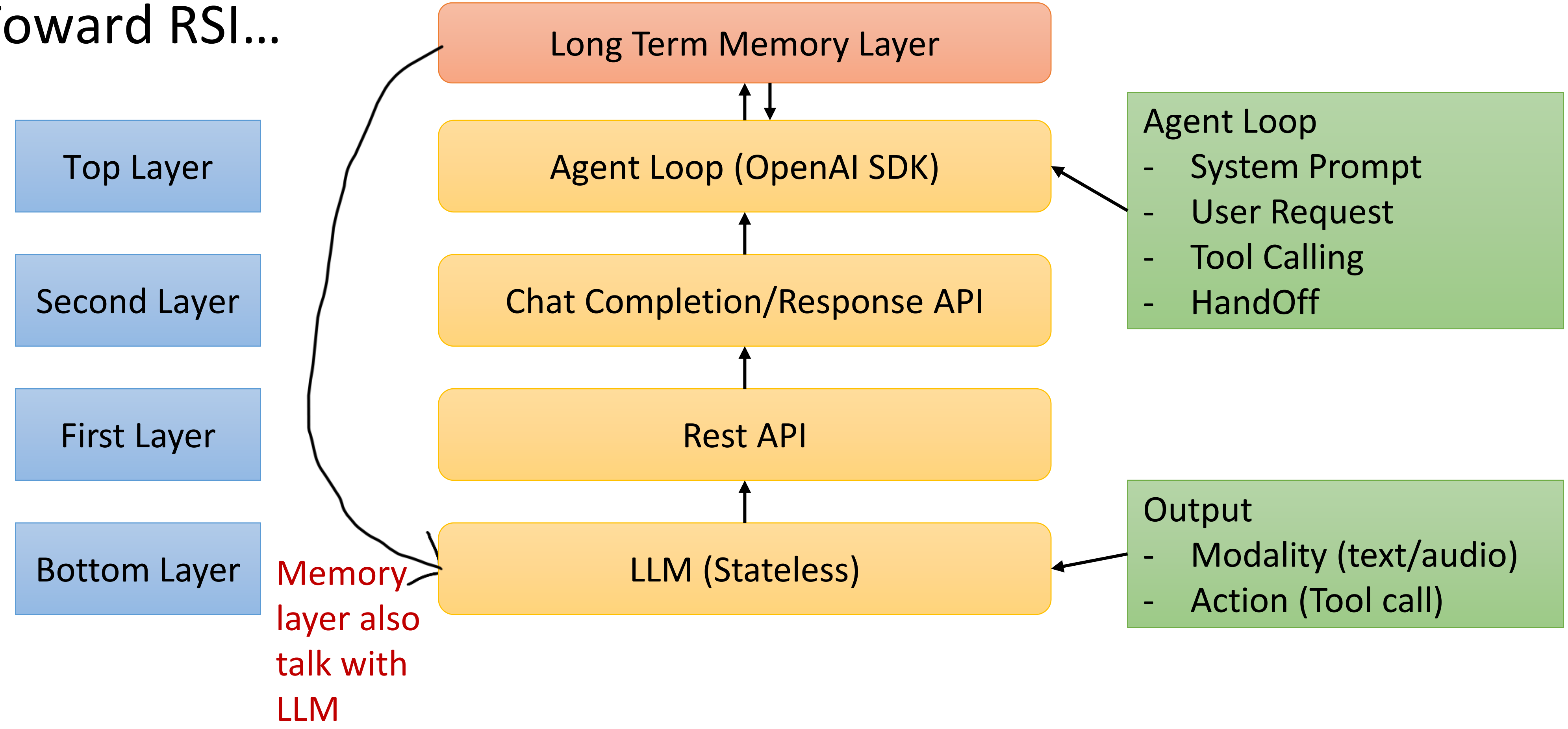
- The handoff and tool calling implementation are same from inside because same API is used in both.
- There are 4 types of messages to understand loop, it’s all about talk between agent and LLMs

1. System Prompt/Instruction/Persona
2. User Prompt/Your Questions
3. Tool Message (process tool to give answer)
4. Assistant message/AI message/Agent Reply to user

Memory (Long term)

Memory Layer

- Agent learn from interactions.
- Toward RSI…



Memory (Long term)

- In the **OpenAI Assistants / Agents system**, "memory" means:
- The ability for an agent to **remember facts, preferences, or events** from **past conversations** and use them in future interactions — like a human would.
- This allows the agent to **build context over time**, not just within one conversation.
- You can available memory in tool calling and in system prompt
- Memory layer is also talk with LLM

Memory (Long term)

What is RSI in this context?

- **RSI = Retrieval System Interface**
- It’s a part of the architecture that supports memory by retrieving relevant past information (like user history, prior chats, facts) when the agent needs it.
- ✅ *In simple terms:*
- RSI is the system that helps the agent "remember" things by retrieving stored information (from vector databases, notes, past interactions, etc.) and feeding it into the current conversation.

Memory (Long term)

How It Works (Simplified Flow):

- User: *"Remind me what I said about my favorite vacation spot?"*
- Agent uses **RSI** to **search its memory store** (e.g., vector DB or long-term notes).
- It finds: *“User said they love Bali in June.”*
- That info is inserted into the context or used to answer.

In memory, we will call LLM in order to reduced the usage of token and just get the context of previous conversations, not full conversation, it means memory will also used LLMs

Memory Management

Memory management is very important:

- To memorize communication between agents
- To remember work while agent performs individual task, keeping in short term or in long term memory
- To remember what tool has done and remember it answer/reply

All these type of stuffs manage via memory layer and to manage this memory in efficient way, there is a package which we called **LangMem**

Work Flow of Agents

- There is a certain difference between work flow and Agentic workflow
- According to Anthropic design pattern article's findings, agents and LLMs decide what will be the sequence of workflow through handoff and tool calling.
- Article link ➔ https://www.anthropic.com/engineering/building-effective-agents
- Workflow and agent could be of short term or long term, short term means one session, long term means keeping flowing and state

Work Flow of Agents

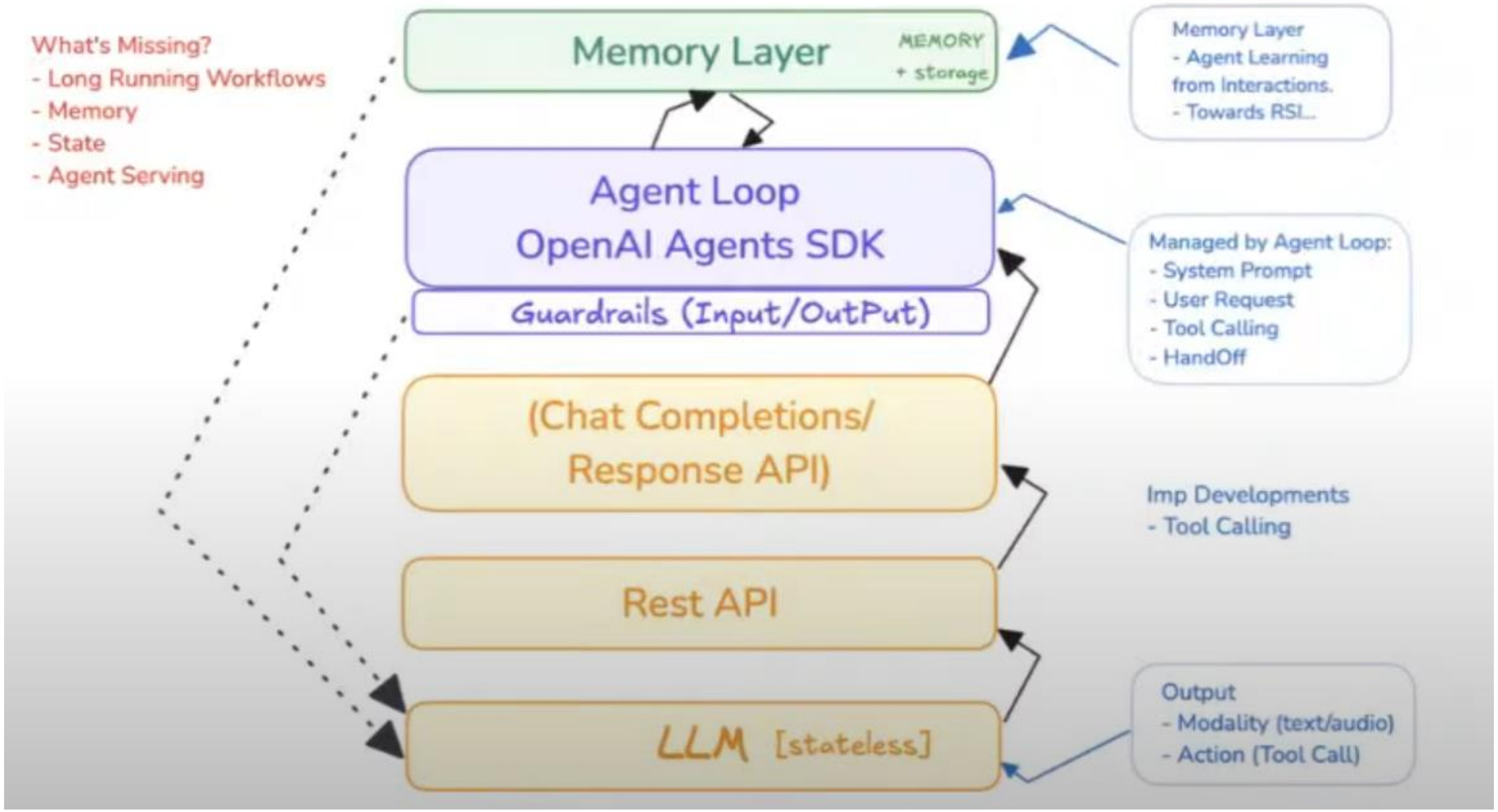
- There are few implementation for long term memory till this lecture date.

1. Either you can used AWS step function
2. Or you can used AWS long running container
3. Or you can used LangGraph → for medium
4. Or you can used Temporal services → https://temporal.io

Guardrails

- Guardrails are rules, filters, or controls that help keep AI agents safe, reliable, and on-topic during generation or decision-making.
- They limit or guide what the AI can say or do, especially to:
  - Prevent harmful outputs
  - Enforce task boundaries
  - Stay within brand or domain rules
  - Improve trust and accuracy
- Guardrails also talk with LLM.

Guardrails



What's Missing?
- Long Running Workflows
- Memory
- State
- Agent Serving
Memory Layer
MEMORY
+ storage
Memory Layer
- Agent Learning
from Interactions.
- Towards RSI...
Agent Loop
OpenAI Agents SDK
Managed by Agent Loop:
- System Prompt
- User Request
- Tool Calling
- HandOff
Guardrails (Input/OutPut)
(Chat Completions/
Response API)
Imp Developments
- Tool Calling
Rest API
LLM [stateless]
Output
- Modality (text/audio)
- Action (Tool Call)

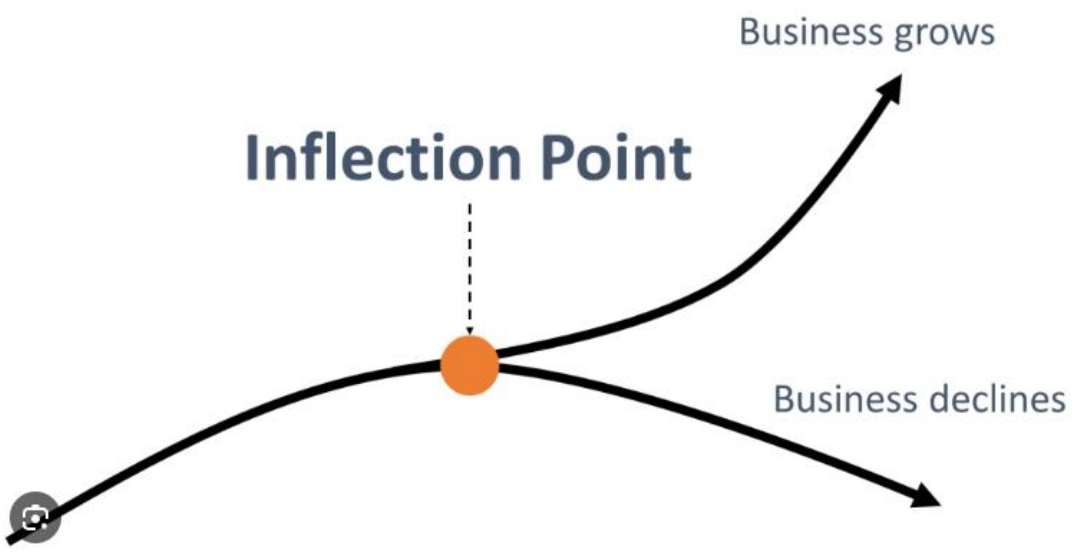
OpenAI SDK

OpenAI SDK provides everything except:

- Memory layer
- And little to no possibly to define long running workflows, it will be implemented by yourself using different external libraries.

Inflection point

- In business, Andy Grove, the former CEO of Intel, defined a strategic inflection point as a significant event that causes a major change in a company or industry, requiring a fundamental shift in strategy.
- It's a turning point where the competitive environment changes, potentially leading to either significant growth or decline. Grove used the term to describe a moment where a company must adapt its approach or face obsolescence.



OpenAI SDK Discussion

- In June 2023, openai introduced function calling and added in chat completion AI.
- Microsoft's Autogen made agent framework before their competitor and released version 0.2
- There are 4 options to make agents
  - **CrewAI:** ➔ comparably easy but lack of logic. Crew ➔ agents ➔ tasks ➔ and crews itself in flows, debugging is very hard/complicated
  - **LangGraph:** ➔ hard to understand states, they mixed up 2 things, i.e., short term agentic (work) flow and long-term workflow.
  - **AutoGen:** Made decision to make agent once again with improve design and workflow which causes division in team so they apart their ways.
  - **OpenAI SDK:**

OpenAI SDK Discussion

- To do analysis properly and in good way, write good prompt
- Few LLMs are become more intelligent, ChatGPT 4.5, Gemini 2.0 pro, Grok 3.
- OpenSDK ➔ 30% framework, 60% python, 10% external libraries for memory and other functionality.

Workflow vs Agents

Workflows (Procedural Execution Style)

- A **workflow** is a **predefined sequence of steps** that the system follows to complete a task. It's deterministic and modular.

Key Characteristics:

- **Static or semi-dynamic:** Steps are known beforehand; the flow may include conditionals or branches, but the control logic is hard-coded or scripted.
- **Orchestrated:** A controller (like a workflow engine or planner) explicitly decides the next step.
- **Predictable:** Good for high-reliability, compliance, or constrained environments.
- **Task decomposition:** Often uses subtasks and tool calls as modular, well-defined operations.

Workflow vs Agents

Workflow Example

Example:

To write and send an email:

markdown

Copy Edit

```markdown
1. Ask user for subject
2. Ask user for body
3. Format the email
4. Send via API
```

Workflow vs Agents

Agents (Autonomous Execution Style)

An agent is an autonomous, goal-driven entity that uses reasoning to decide what to do next, possibly exploring and adapting dynamically.

Key Characteristics:

- **Dynamic planning:** Agents choose the next action based on current state, tools, and goal.
- **Looping control:** Agents often use think-act-observe cycles (e.g., ReAct or MRKL patterns).More autonomy: Agents decide how to complete the goal, not just follow steps.
- **Stateful:** Agents remember intermediate states, outcomes, failures.
- **Emergent behavior:** Sometimes solutions arise that weren't explicitly programmed.

Workflow vs Agents

Example Agents (Autonomous Execution Style)

Example:

Given a goal like “book a flight to NYC,” the agent might:

markdown Copy Edit

```markdown
1. Search for flights
2. Extract data
3. Decide it's too expensive
4. Ask user for alternative dates
5. Retry
6. Book via API
```

No predefined script — the agent decides each step based on the current context and goal.

Short-Term Workflows vs Long-Term Workflows

Short-Term Workflows (Microplans)

Definition:

- **Templated sequences** of **steps** to achieve subgoals within a single agent loop or session.

Characteristics:

- Focused on **immediate tasks** (e.g., "summarize a document," "extract a table").
- Often cached or retrieved from toolkits (e.g., function graphs).
- Agent **calls or instantiates** them dynamically.

How Agents Use Them:

- As building blocks to solve parts of larger goals.
- Invoked when a known pattern matches the current subgoal.
- Can be **executed as-is**, modified, or aborted mid-way.

Short-Term Workflows vs Long-Term Workflows

Long-Term Workflows (Macroplans)

Definition:

- **Extended, goal-level strategies** or plans that span **multiple steps, sessions, or time horizons**.

Characteristics:

- Represent high-level plans (e.g., "launch a product," "plan a vacation").
- May persist across sessions or agent memory.
- Include checkpoints, dependencies, and progress tracking.

How Agents Use Them:

- As **strategic scaffolding**: guides the agent in sequencing major stages.
- Enables **resumability** and **temporal reasoning** over days/weeks.
- Often stored in vector memory, graph structures, or long-term state.

🧠 In Practice: Agent + Workflow Fusion

- Agents don’t replace workflows — they use them:
- **Short-term workflows**: Like calling a subroutine.
- **Long-term workflows**: Like following a project plan.
- The agent chooses **when** to execute them, **how** to adapt them, or **whether** to abandon them based on reasoning and observations.

Open AI SDK Discussion

OpenAI SDK:

- Is not workflow but agent.
- Does dynamic routing
- Is short term

This make easy to use and you can use it for smaller single session task due to short term.

MCP (Model Context Protocol)

- **MCP (Model Context Protocol)** is an emerging concept/protocol designed to **enable coordination between large language models (LLMs), agentic systems, tools, and environments**.
- It's most prominently associated with **OpenAI's research into AI agents**, introduced around early 2024.
- MCP is a **high-level abstraction layer or interface** that helps manage **context, state, and communication** in complex AI-agent architectures.

What is MCP (Model Context Protocol)?

MCP stands for **Model Context Protocol**. It is a **communication and coordination protocol** designed to:

- **Standardize the interaction between models and their environment** (tools, memory, world states).
- Allow **agents to persist knowledge, state, and goals** across steps or tasks.
- Help **orchestrate multiple agents or model instances** working together, potentially asynchronously.
- Support **long-lived, autonomous agentic behaviors** beyond single-turn prompts.

Think of it as a **“software bus” or orchestration layer** for agent-like systems using LLMs.

🔧 What does MCP do with Agents?

MCP is a **key enabling infrastructure for building advanced AI agents**, particularly those that:

- **Need memory** (beyond what a single model context can hold).
- **Perform multi-step reasoning or planning**.
- **Interact with multiple tools**, APIs, or real-world systems.
- **Persist identity, goals, or context across time**.
- Coordinate **multiple model instances** (e.g., planner, executor, critic, etc.).

Concretely, MCP does things like:

- **Tracks agent state**: Goals, plans, world model, memories, tasks in progress.
- **Manages context**: So, the model can resume tasks with relevant context without reloading everything.
- **Routes messages**: Between agents, tools, APIs, or humans.
- **Encapsulates prompts + metadata**: Such as system message, tool responses, or user instructions.

Key Capabilities

🧠 Key Capabilities

Feature	Description
Context Management	Loads and persists relevant information between agent steps.
Tool Routing	Manages how the model calls and receives outputs from external tools.
Memory Integration	Lets the agent remember past interactions, tasks, or experiences.
Task Decomposition	Coordinates how agents break down and execute complex goals.
Agent-to-Agent Messaging	Enables agents to collaborate or specialize (e.g., sub-agents for planning vs acting).

🧪 Example Use Case

Imagine an AI executive assistant that:

1. Remembers your calendar.
2. Plans a trip.
3. Books hotels, reschedules meetings.
4. Learns from feedback.
5. Coordinates with sub-agents (e.g., a flight booking agent, a budget optimizer).

MCP allows this assistant to maintain context over hours/days, route tasks to the right agents or tools, and operate autonomously without forgetting what it's doing between steps.

MCP Discussion

- You can give external info to LLMs by below 3 ways:
  1. You can train model on new data ➔ which is obviously a hard job
  2. You can make RAG (Retrieval Augmented Generation) system, like convert all your document data into vector and place in vector database. **RAG retrieves relevant external documents** (usually stored in a vector database) and then **uses them to generate more accurate and grounded responses**.
  3. You can used external data using tool/function calling, the hardest thing in function calling is to make schema of function, i.e., name, parameter, required parameters, what function does?, what response function will give after processing etc.
- MCP found this (schema issue) gap, MCP hired 1100 sources so developer can connect them to get external info which is very hard tp do and time consuming.

MCP Discussion

- It was release in Dec 2024 and it is getting maturity day by day.
- For those 1100 sources, developer had to write tools and schema, that is why MCP was released, **a protocol in which all 1100 sources tools and schema has been written in standard way.**
- MCP operates on a **client server architecture**

MCP operates on a client-server architecture:

- **MCP Hosts**: These are the AI applications (like a chatbot or an IDE plugin) that need access to external data or capabilities.
- **MCP Clients**: These sit within the host and manage secure, one-to-one connections to servers.
- **MCP Servers**: These are lightweight programs that expose specific tools, data, or resources (e.g., a GitHub server might provide repository access) to the AI.

Development to Deployment & Introduction to DACA Design Pattern

Discussion

- If we created virtual machine, it’s obvious it is stateful (memory), but the problem is we can’t scale it up.
- Cloud owners determined that the virtual machine is not as good respect to scalability and made it light weight version which is ‘containers’, containers are stateless.
- You give request to container, container ups ➔ load state from database ➔ do your assign work ➔ it’s saved some stuff database ➔ Give response back to you ➔ then forget everything about you.
- So, the design pattern which give you maximum scalability those are stateless containers of ‘**Docker**’. These are also called ‘*<u>Serverless Containers</u>*’

Development to Deployment & Introduction to DACA Design Pattern

Discussion

- As of today, the latest state-of-the-art stuff of cloud are serverless container (managed) or stateless container.
- OpenAI SDK has record of complete conversation, when you call it in stateless method then you can save all conversation in database by single call, by this you can make scalable agent how? Stateless protocol, you made stateless FastAPI server which is inside container, when the request comes, it pick detail from database
- To make applications, we have to merge cloud native technology and agentic AI, by merging these we can even make systems of Microsoft level.

Development to Deployment & Introduction to DACA Design Pattern

Discussion

- When we try to make our system serverless then it core root in Docker, we make microservices from docker and to manage these microservices (containers) using Kubernetes.
- **Kubernetes** auto manage containers, if million, billion users come to your app, it will up million billion containers/microservices
- Although, Kubernetes is very hard to learn and we have to make serverless system, so we will learn Docker (as a base for serverless) and **DAPR**.

DACA (Dapr Agentic Cloud Ascent)

Dapr Agentic Cloud Ascent (DACA) is a design pattern for building and scaling agentic AI systems using a minimalist, cloud-first approach. It integrates the OpenAI Agents SDK for agent logic, MCP for tool calling, Dapr for distributed resilience, and a staged deployment pipeline that ascends from local development to planetary-scale production. DACA emphasizes:

- **AI-First Agentic Design**: Autonomous AI agents, powered by the OpenAI Agents SDK, perceive, decide, and act, with **MCP** enabling tool access and **A2A** facilitating intelligent agent-to-agent dialogues.
- **Agent-Native Cloud Scalability**: Stateless containers deploy on cloud platforms (e.g., Azure Container Apps, Kubernetes), leveraging managed services optimized for agent interactions.
- **Stateless Design**: Containers that scale efficiently without retaining state.
- **Dapr Sidecar**: Provides state management, messaging, and workflows.
- **Cloud-Free Tiers**: Leverages free services for cost efficiency.
- **Progressive Scaling**: From local dev to Kubernetes with self-hosted LLMs.

DACA (Dapr Agentic Cloud Ascent)

Discussion:

- **Cloud Ascent** refers to the process of developing an application locally—such as on a laptop during the development phase—and seamlessly scaling it to support millions of users in the cloud as demand grows.
- And this cloud is for agents
- Is DACA is framework or design?? Check appendix III in below link
- https://github.com/panaversity/learn-agentic-ai/blob/01e344ba85ec36134c783b5ceef45a12a9bb7e68/comprehensive_guide_daca.md#appendix-iii-daca-a-design-patter-or-framework

DACA (Dapr Agentic Cloud Ascent)

Discussion:

- The **Dapr Agentic Cloud Ascent (DACA)** is best classified as a **design pattern**, though it has elements that might make it feel framework-like in certain contexts. Let's break this down to clarify its nature and why it fits the design pattern label, while also addressing the nuances that might lead to confusion.
    - **Framework:** A reusable, pre-written set of code or libraries that provides a structure for developing applications. It often dictates the flow (e.g., React, Django).
    - **Design Pattern:** A general, reusable solution to a common software design problem. It's a conceptual template, not actual code (e.g., Singleton, Observer).
    - **In short:** <u>*Framework = implementation + structure*</u>. <u>*Design Pattern = conceptual solution*</u>

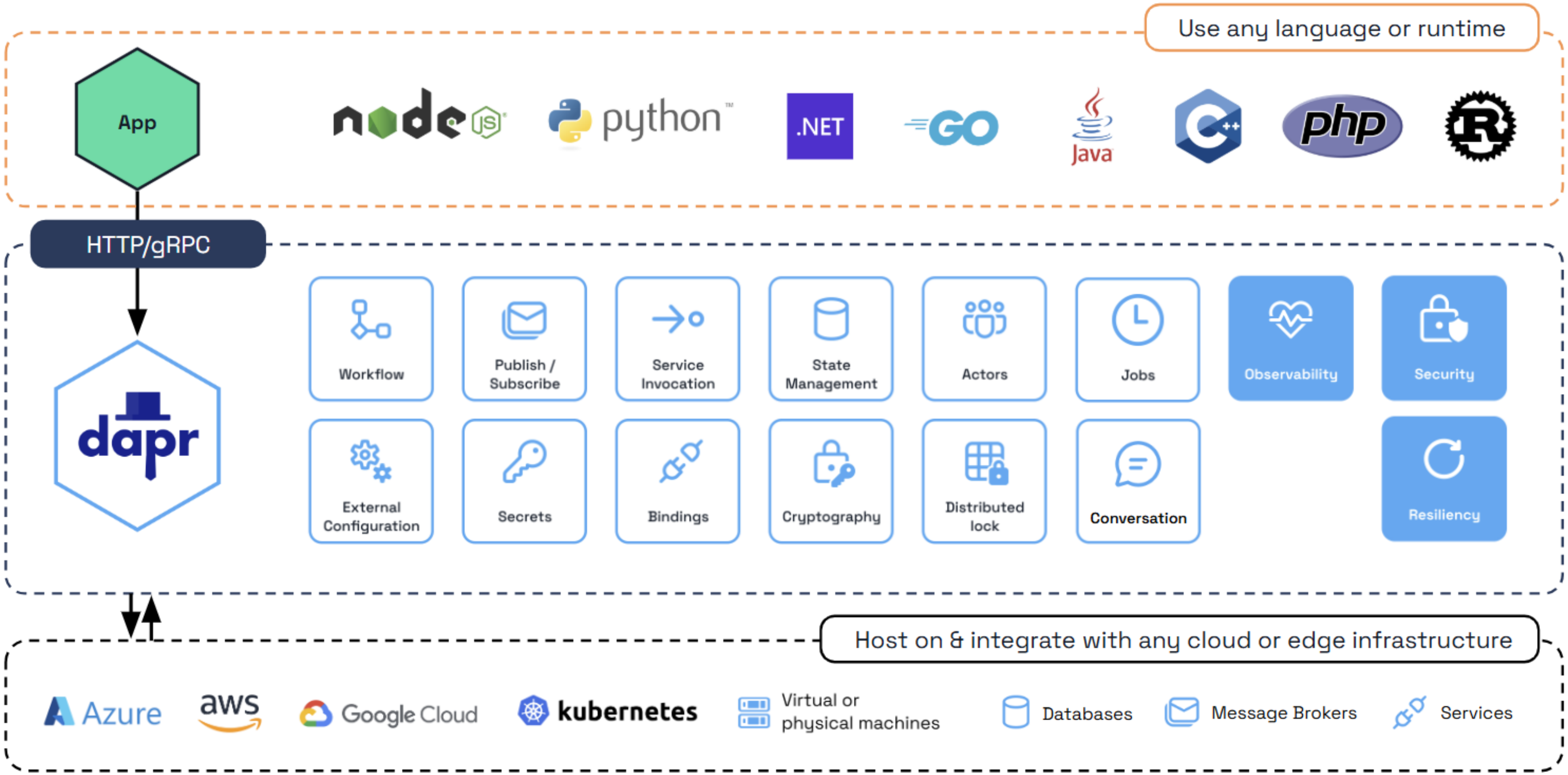
DACA (Dapr Agentic Cloud Ascent)

Discussion:

- In DACA, we are using DAPR (DAPR is tech which is used to distribute microservices which communicate with each other), Agents (any tech used to make agents) and Cloud (any tech for cloud native)
- **DACA as a Pattern**: It’s a high-level strategy for agentic AI systems, focusing on architecture (three-tier, EDA), principles (statelessness, HITL), and deployment stages (local to planet-scale). It doesn’t provide a runtime or library—you build the system following its guidance.
- **Not a Framework**: DACA doesn’t offer a pre-built runtime, APIs, or enforced conventions. While it suggests tools (e.g., Dapr, FastAPI), these are optional, and the pattern’s core is about *how* to structure the system, not *what* to use.

DACA (Dapr Agentic Cloud Ascent)

Discussion:



DACA (Dapr Agentic Cloud Ascent)

Discussion:

- DAPR is an open source, you can call it framework, tool or library which we can used using Python, Node, and other languages mentioned in above picture
- You make your software in chuck and each chunk in distributed in microservices, you are continuously adding feature as a separate microservice
- **DAPR is responsible for communication between these microservices**

DACA (Dapr Agentic Cloud Ascent)

Discussion:

- DAPR up container in stateless position, it means you up container in which live streaming feature is uploaded but different users are streaming at the same time but state will come from user’s system, which user will use streaming microservices suppose on Facebook.
- Think of your container as a "bike" running your microservice. DAPR acts as a "sidecar," providing additional features. For example, if your container fails, DAPR automatically starts another container to ensure the task is completed.

DACA (Dapr Agentic Cloud Ascent)

Discussion:

- You can deploy it on any cloud where Kubernetes are present
- Kubernetes in orchestrator which up and down the container when event comes
- For tool calling, we will use new version of MCP which is remote and will be running in some container.

Agent to Agent Communication & Development, Protoyping, Production using DACA

Discussion:

- Imagine a world where everything is an AI agent, from your coffee machine to your car, from businesses to entire cities. Picture a world transformed into Agentia—a dynamic, living network of intelligent AI agents seamlessly integrated into our daily lives.
- From our homes and offices to entire cities, systems no longer communicate through outdated APIs but through sophisticated, intelligent dialogues driven by state-of-the-art AI frameworks.
- Agentia scales effortlessly across the globe, thanks to its foundation in cloud-native technologies. Agentia is more than digital—it's also physical, brought to life by robots that serve as embodied agents interacting with and enhancing our physical world.

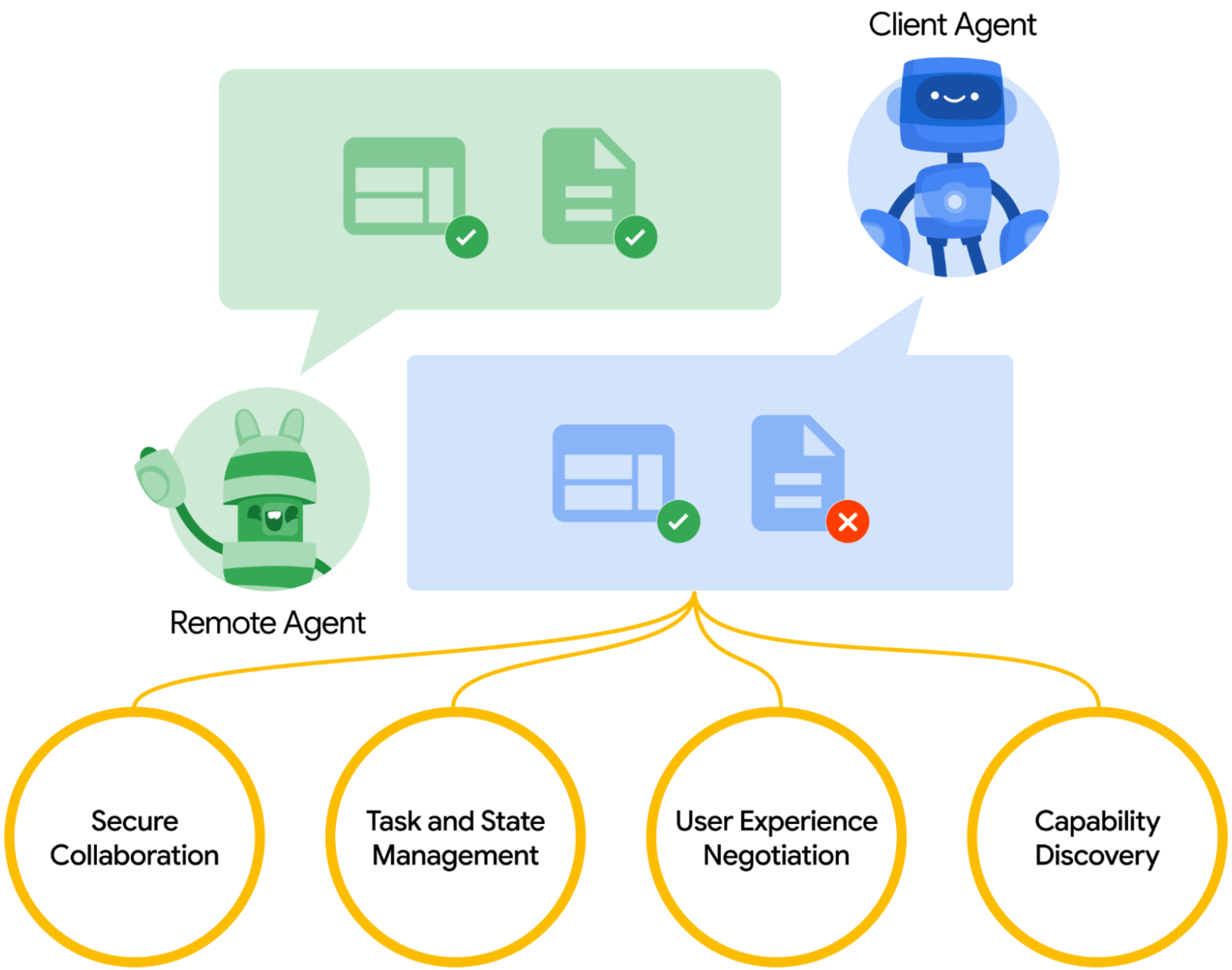
Agent to Agent Communication & Development, Protoyping, Production using DACA

Discussion:

- Google launched A2A with collaboration with other companies in April 2025.
- A2A, launched by Google with over 50 partners, is integral to DACA. It uses HTTP, SSE, and JSON-RPC to enable secure, modality-agnostic (text, audio, video) agent communication. Key A2A features in DACA include:
  - **Agent Cards**: JSON files (/.well-known/agent.json) advertise capabilities, enabling discovery.
  - **Task Management**: Agents initiate and process tasks with real-time feedback via A2A endpoints.
  - **Interoperability**: Connects agents across platforms, supporting Agentia's vision of a global network.
  - **Security**: Enterprise-grade authentication ensures trust in cross-domain dialogues.

Agent to Agent Communication & Development, Protoyping, Production using DACA

Discussion:



Agent to Agent Communication & Development, Protoyping, Production using DACA

Discussion:

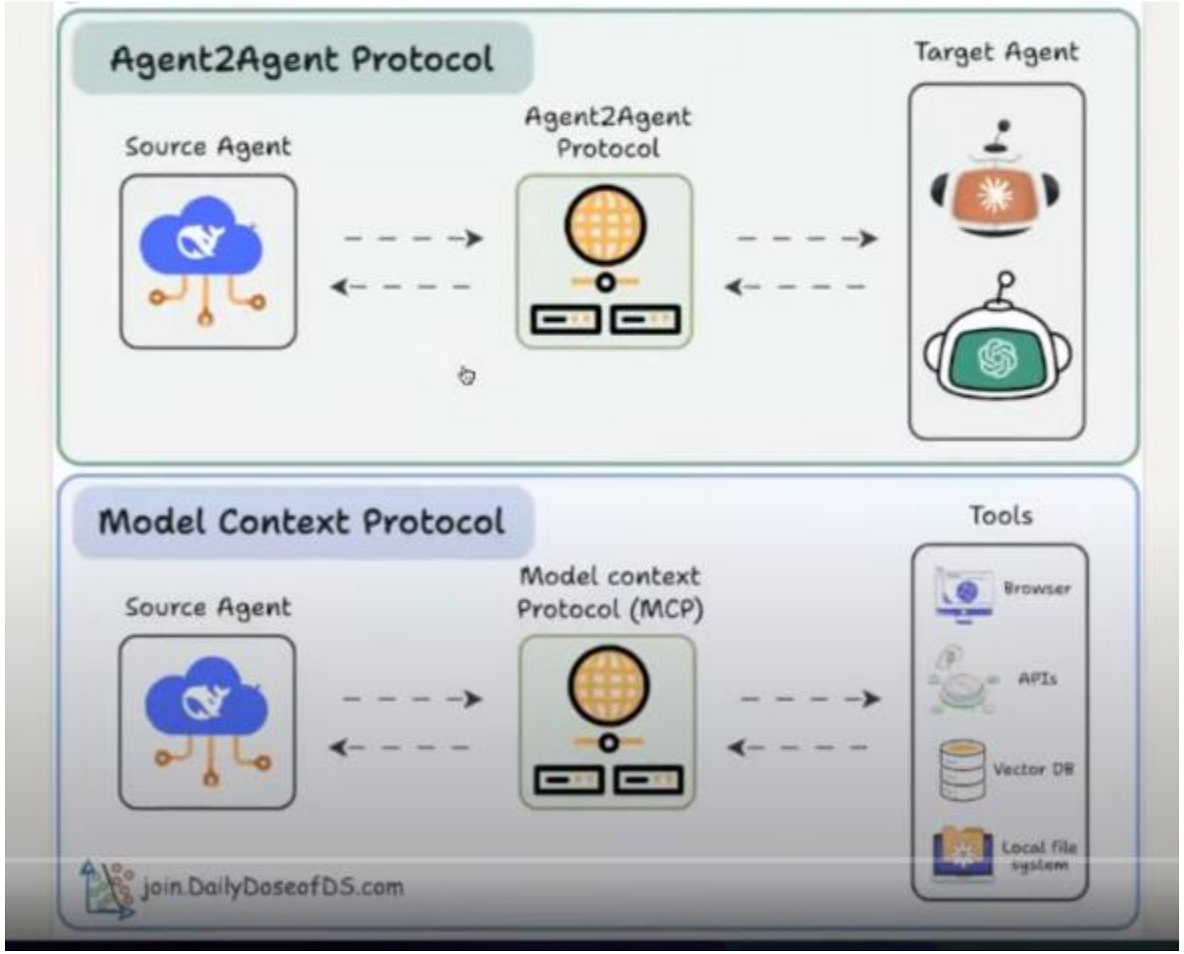
A2A (Agent-to-Agent) lets two agents work together — a **client agent** and a **remote agent**. The client agent sends tasks, and the remote agent carries them out to give back results or take action.

Here's how it works:

- **Finding the right agent (Capability discovery):** Each agent has a digital "Agent Card" listing its skills. The client agent uses this to pick the best remote agent for the job.
- **Handling tasks (Task management):** Tasks can be short or long. For longer ones, the agents keep talking to stay in sync. The final result of a task is called an **artifact**.
- **Working together (Collaboration):** The agents send messages to share info, progress, and instructions.
- **Adapting for the user (User experience negotiation):** Messages include content like images, videos, or forms. The agents agree on the right format depending on what the user's system supports.

Why we need A2A when we already have MCP?

Discussion:



Why we need A2A when we already have MCP?

Discussion:

- In the context of MCP, agents are provided with tools rather than being required to develop new ones. These tools can be pre-existing and are deployed to the MCP server for use by agents. In this model, agents are exposed through the MCP server and made available as callable resources.
- Regarding A2A (Agent-to-Agent) communication within MCP, there is a single supported interaction pattern: one agent utilizes another agent as a tool. This implies that agent-to-agent communication in MCP is unidirectional and hierarchical — one agent acts as the client, while the other functions as a tool or service provider. Therefore, any scenario involving A2A communication through MCP is structured such that one agent is effectively embedded into the workflow of another.

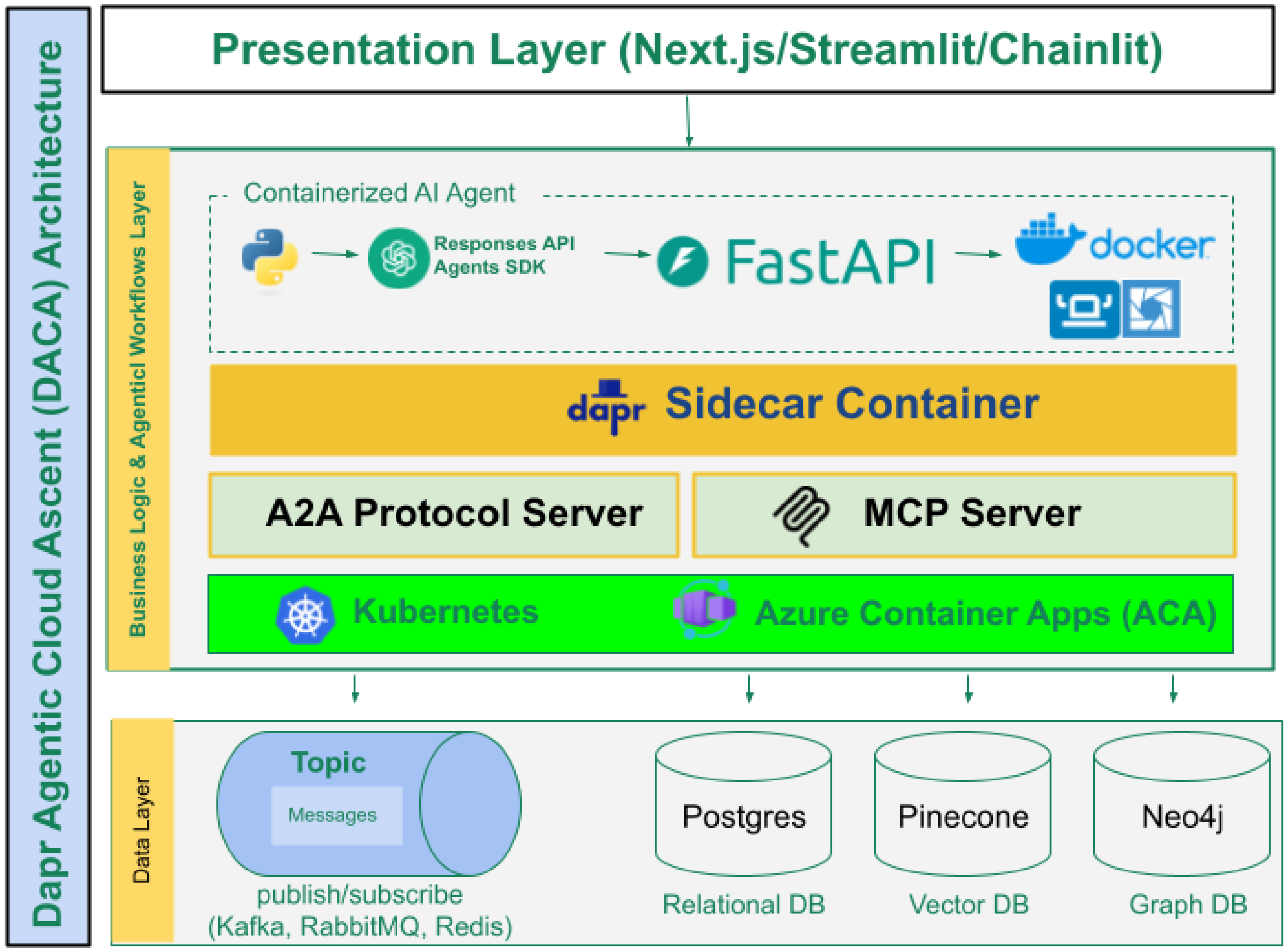
Why we need A2A when we already have MCP?

Discussion:

- In the context of A2A (Agent-to-Agent) communication, the framework offers the flexibility to build truly dynamic and scalable multi-agent systems. Unlike traditional tool-based interactions, A2A enables agents to operate with a higher degree of autonomy.
- Agents are not limited to functioning solely as tools for other agents. Instead, they are capable of **self-discovery**, allowing one agent to identify and connect with other agents at runtime. Through **capability sharing mechanisms**—such as the exchange of structured "Agent Cards"—agents can evaluate the skills and functionalities of others and determine the most appropriate collaboration path. This fosters a decentralized, intelligent environment where agents can initiate communication, delegate tasks, and cooperate seamlessly based on real-time needs and capabilities.

DACA Architecture Diagram Breakdown

Discussion:



DACA Architecture Diagram Breakdown

Discussion:

- We use Python and the OpenAI SDK to build our agentic system. To interface with other systems or users, we expose the agent through an API framework—specifically, FastAPI.
- In simple terms, the agent is wrapped with an API to make it accessible externally. This agent runs in the cloud, and as discussed previously, containerization is the ideal approach for cloud deployment.
- Using Docker, we create a containerized environment that includes everything—from Python and the OpenAI SDK to FastAPI—ensuring consistency across local and cloud environments. This setup allows us to deploy the agent on any cloud platform with ease.
- This whole process is called **Containerized AI Agent**.

DACA Architecture Diagram Breakdown

Discussion:

- DACA Architecture is also called 3 tiers architecture
  - First layer is **‘Presentation layer’**
  - Second layer is **‘Business Logic layer’**
  - Third layer is called **‘Data layer’**
- DACA built on 3 tiers application
- Presentation layer could be of nextjs, streamlit, chainlit or any other mock server from FastAPI etc.
- To write business logic layer, we learn Python programming, OpenAI SDK, and learn FastAPI to expose agent as a web server or API

DACA Architecture Diagram Breakdown

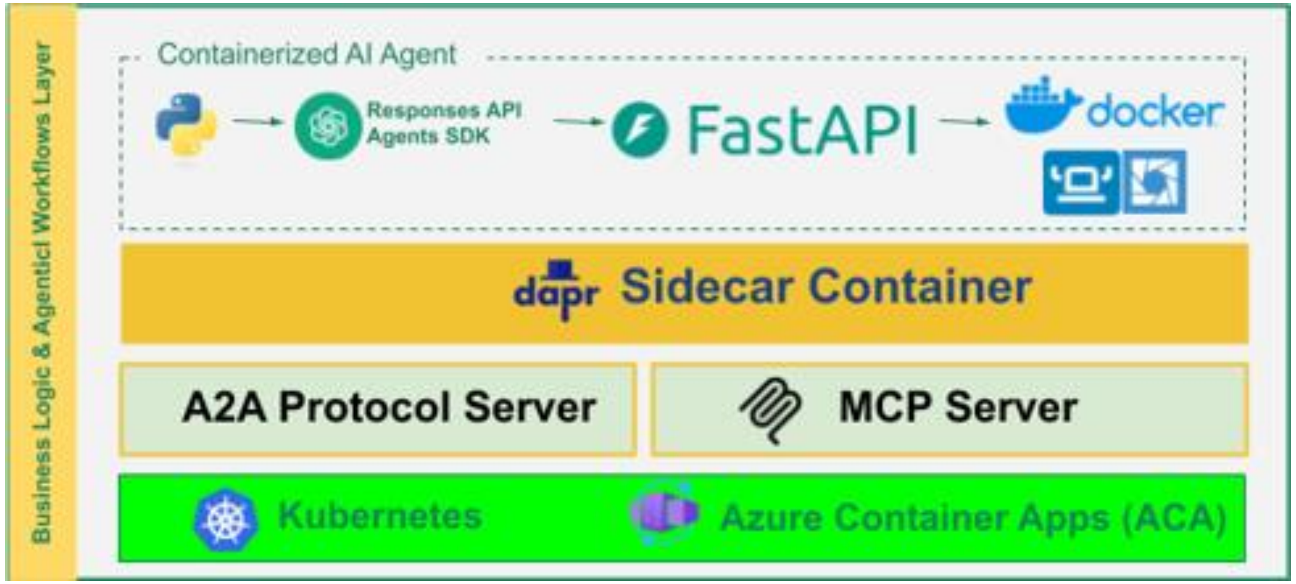
Discussion:

- Sidecar container (DAPR) used to communicate between microservices
- Sidecar is responsible to make stateful your stateless microservice which is up due to some user interaction.
- If microservice fail to up then sidecar has info about task who need to be done so it will up another microservice for it. That is called **Resilience/Fault tolerance**.
- As its name shows, sidecar is supporting microservices i.e., to make microservice stateful, communication between microservices, keep it resilience, manage logs etc.

DACA Architecture Diagram Breakdown

Discussion:

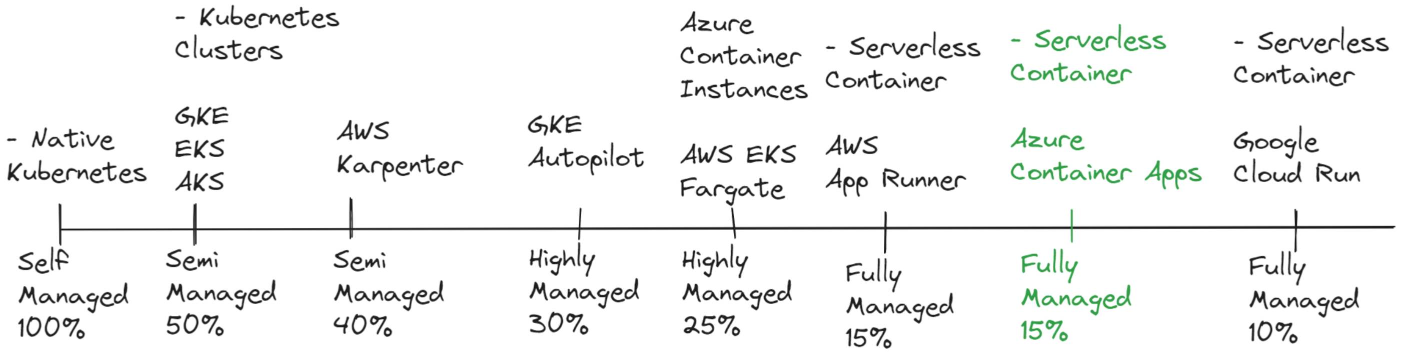
- If agent in FastAPI want to talk with another agent it will used A2A protocol and if agent want to talk with tools like Database, API then it will used MCP protocol.

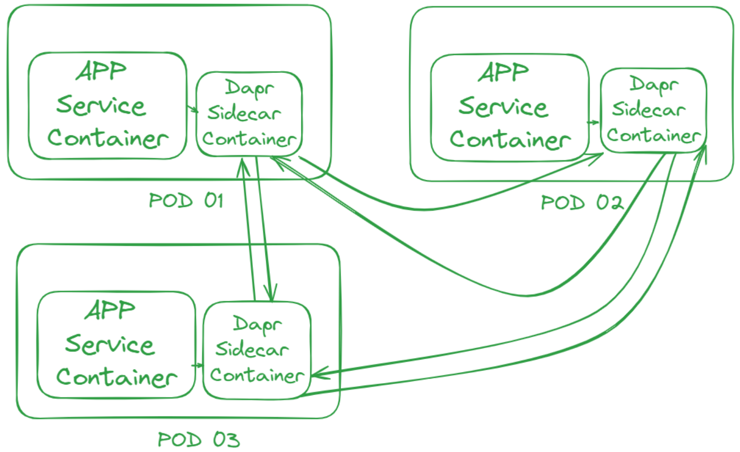


DACA Architecture Diagram Breakdown

Discussion:

- Now we have Kubernetes and Azure layer (check video from 55 min to 1.23 hr)





DACA Architecture Diagram Breakdown

Discussion:

- Python code is running in container (logical visualization / logical separate boundary), why we have to run python code in container? It has 2 reasons
  - Our native should run, native means refers to applications or services specifically **designed and built to run in a cloud environment**, leveraging its full capabilities such as scalability, elasticity, distributed architecture, and managed services. These apps are **not just migrated** to the cloud—they are **optimized for it from the ground up**.
  - If we want to orchestrate/manage something with Kubernetes that it is the requirement of it to run your code in container, otherwise Kubernetes will not manage your code.

DACA Architecture Diagram Breakdown

Discussion:

- Container also has few requirement like
  - Network connection to connect with it
  - Container needs hardware i.e., CPU, storage, RAM etc.
- Kubernetes also make logical network service and named as **POD;** the aim of this POD is to run container. Kubernetes is a big system, in Kubernetes POD is responsible for running container because POD provides:
  - Network to container
  - Address to container
  - Spec (hardware, RAM, CPU, storage)

DACA Architecture Diagram Breakdown

Discussion:

- But in highly scalable system, there is problem if 20 millions users landed on my app if one POD enough for me?
- Suppose as per given scenario I increase the quantity of containers, but how Kubernetes will know that I increase containers to suppose 5?
- First of all, Increasing and decreasing of container is in Kubernetes domain, not us, it doesn't directly manage container but POD on the behalf of Kubernetes controls containers.
- If 20 million traffic come to your app then Kubernetes increases PODs (containers are inside the PODs)

DACA Architecture Diagram Breakdown

Discussion:

- If users decline on your app, then Kubernetes also scale down no of PODs in order to save resources
- As developers, when we build highly scalable systems, our Python code often needs to interact with external components like databases or state management services.
- While Kubernetes can understand the resource requirements of the Python application and allocate them accordingly, a question arises: <u>how does Kubernetes know that the Python code is making calls to external systems, such as databases? How does it detect these interactions, and how does it manage or track accountability for these external calls?</u>

DACA Architecture Diagram Breakdown

Discussion:

- If Kubernetes were to track accountability, how could it help reduce redundancy—for example, when five containers running the same code are all making identical calls to the database?
- Sir Aleem suggested a plug-and-play approach. This means when deploying a container, although POD is providing resources (hardware, RAM etc.) we should inform the Pod about the necessary external dependencies—such as which database to connect to, which storage to use, and which APIs to call.
- However, the Pod's responsibility is limited to managing the container and its lifecycle; it doesn’t handle external integrations. This is where **Dapr** (Distributed Application Runtime) comes in. Dapr acts as a sidecar to the application, handling service calls, state management, pub/sub, and more—enabling a clean, modular, and reusable approach to external system interaction.

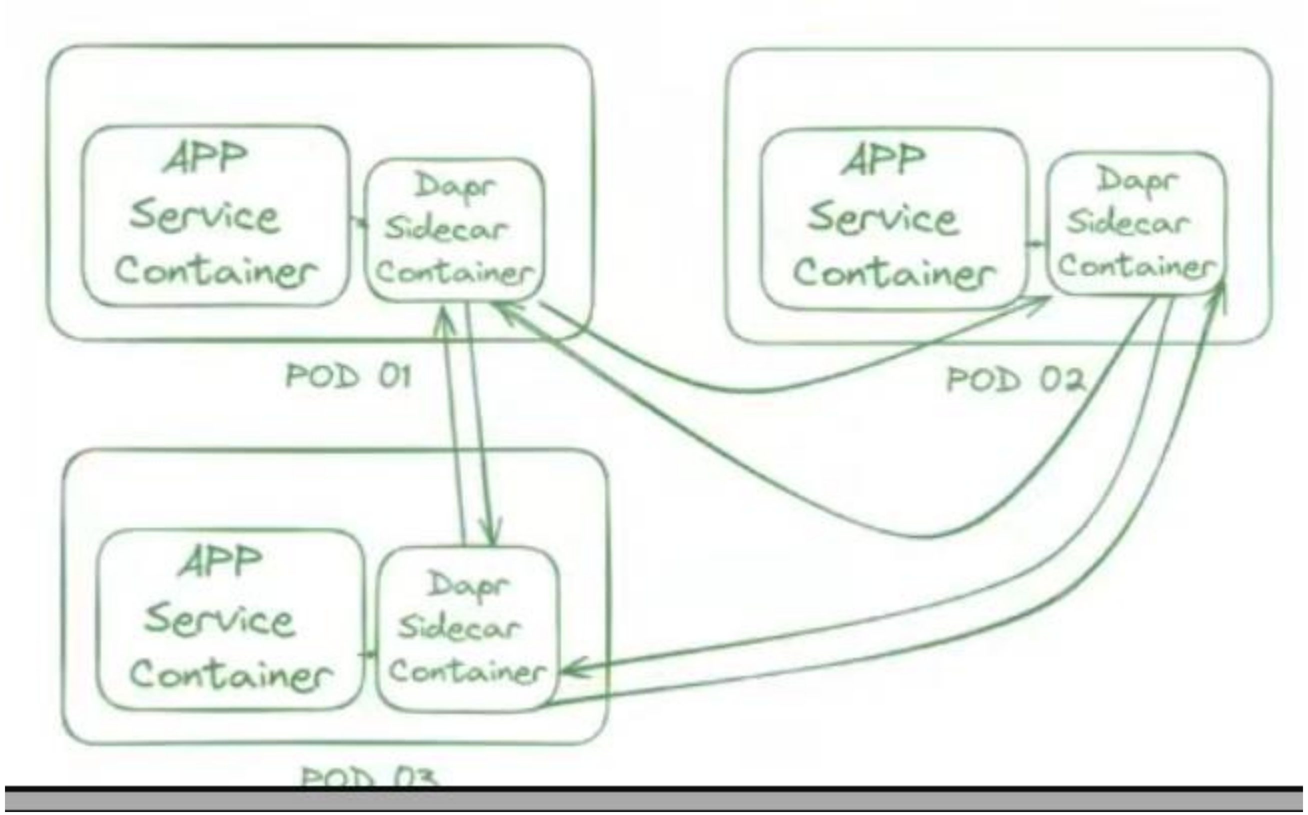
DACA Architecture Diagram Breakdown

Discussion:

- The term **"sidecar"** in this context refers to running an additional container alongside the main application container within the same Pod.
- This sidecar container acts as a supporting service that helps the main application manage external interactions—such as connecting to databases, handling state, or making API calls.
- In the case of **Dapr**, it runs as a sidecar container. While your main code container focuses on business logic, Dapr handles the communication with external systems but we have to inform Dapr to connect with which resource.
- This separation of concerns allows for a more modular, scalable, and reusable system design.

DACA Architecture Diagram Breakdown

Discussion:



DACA Deployment Stages: The Ascent

1. Local Development: Open-Source Stack (Discussion:)

- Github link for detail ➔ https://github.com/panaversity/learn-agentic-ai/blob/main/comprehensive_guide_daca.md#daca-deployment-stages-the-ascent
- We should make container which will run on local development, hugging face, azure container app and on Kubernetes, only the configuration of code is different but code remain same when running on different platforms.
- **Goal**: Rapid iteration with production-like features.

DACA Deployment Stages: The Ascent

1. Local Development: Open-Source Stack (Discussion:)

- Now what can we do? Your container is running in which you have your FastAPI and agent code, same as you have another container having both are running, both container communicating with each other using Dapr and in between Rapid MQ as well, then you need database as well, considering these scenarios, you have 2 ways for local development.
  - Docker compose: Run the agent app, Dapr sidecar, A2A endpoints and local services (Old suggestion – Apr 2025)
  - **Rancher Desktop with Lens**: Runs the agent app, Dapr sidecar, A2A endpoints and local services on local Kubernetes. (Newly sugguested – July 2025)

DACA Deployment Stages: The Ascent

2. Prototyping: Free Deployment (Discussion:)

- For prototyping, like to show your work to customer or friend and for testing, you need free deployment service, go to Hugging face docker spaces.
- Hugging Face Spaces offers the ability to host custom applications using Docker containers, known as **Docker Spaces**. This feature allows users to deploy machine learning demos and applications that go beyond the capabilities of the default Streamlit and Gradio SDKs provided by Hugging Face.
- https://huggingface.co/docs/hub/en/spaces-sdks-docker

DACA Deployment Stages: The Ascent

2. Prototyping: Free Deployment (Discussion:)

- Hugging face docker space in completely (as of date), you can deploy as many as containers in it.
- In hugging face docker space, container can handle up to 100 users.
- On free tier, you can 2 CPUs and 8 to 16 GB ram. By spending 0.03 USD per hour, you can get 8 CPUs and 32 GB ram but it is for the situation when customer wants it to show their consumers.
- For this you need cron-job as well, a **CronJob** in Kubernetes is a way to run a scheduled task (a job) at a specific time or interval — similar to the Linux cron system.
- A **CronJob** is used to **automatically run a container at a scheduled time**, such as:
    - Every hour
    - Every day at midnight
    - Every Monday at 8 AM

DACA Deployment Stages: The Ascent

2. Prototyping: Free Deployment (Discussion:)

- **Goal**: Test and validate with minimal cost.
- **Setup**:
  - **Containers**: Deploy to Hugging Face Docker Spaces (free hosting, CI/CD). Both FastAPI, MCP Server, and A2A endpoints in containers.
  - **LLM APIs**: Google Gemini (free tier), Responses API.
  - **Messaging**: CloudAMQP RabbitMQ (free tier: 1M messages/month, 20 connections).
  - **Scheduling**: https://cron-job.org (free online scheduler).
  - **Database**: CockroachDB Serverless (free tier: 10 GiB, 50M RU/month).
  - **In-Memory Store**: Upstash Redis (free tier: 10,000 commands/day, 256 MB).
  - **Dapr**: Use [Managed Dapr Service Catalyst by Diagrid]() free-tier.
- **Scalability**: Limited by free tiers (10s-100s of users, 5-20 req/s).
- **Cost**: Fully free, but watch free tier limits (e.g., Upstash's 7 req/min cap).

DACA Deployment Stages: The Ascent

3. Medium Enterprise Scale: Azure Container Apps (ACA) (Discussion:)

- Suppose customer said, there will be 1000 users landed on my app simultaneously, that will fall in medium enterprise scale
- **Goal**: Scale to thousands of users with cost efficiency.
- **Setup**:
  - **Containers**: Deploy containers (FastAPI and MCP Servers) to ACA with Dapr support (via KEDA).
  - **Scaling**: ACA's free tier (180,000 vCPU-s, 360,000 GiB-s/month) supports ~1-2 always-on containers, auto-scales on HTTP traffic or KEDA triggers.
  - **LLM APIs**: OpenAI Chat Completion, Responses API.
  - **Messaging**: CloudAMQP RabbitMQ (paid tier if needed).
  - **Scheduling**: ACA Jobs for scheduled tasks.
  - **Database**: CockroachDB Serverless (scale to paid tier if needed).
  - **In-Memory Store**: Upstash Redis (scale to paid tier if needed).
- **Scalability**: Thousands of users (e.g., 10,000 req/min), capped by OpenAI API limits (10,000 RPM = 166 req/s). Using Google Gemini will more economical.
- **Cost**: Free tier covers light traffic; paid tier ~$0.02/vCPU-s beyond that.

DACA Deployment Stages: The Ascent

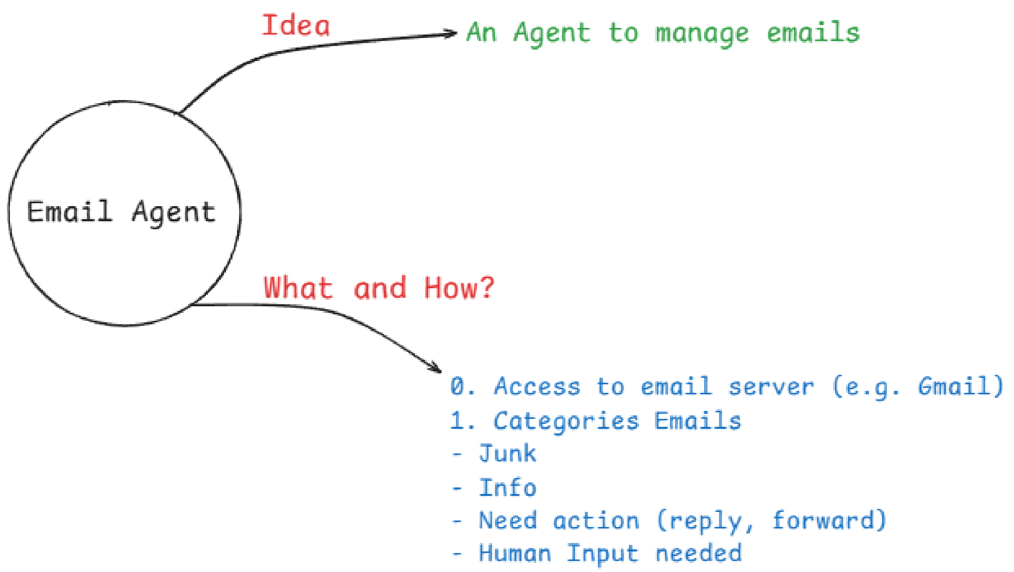
4. Planet-Scale: Kubernetes with Self-Hosted LLMs (Discussion:)

- **Goal**: Achieve planetary scale with no API limits.
- **Setup**:
  - **Containers**: Kubernetes cluster (e.g., on Oracle Cloud's free VMs: 2 AMD VMs or 4 Arm VMs). Both FastAPIs and MCP containers.
  - **LLM APIs**: Self-hosted LLMs (e.g., LLaMA, Mistral) with OpenAI-compatible APIs (via vLLM or llama.cpp).
  - **Messaging**: Kafka on Kubernetes (high-throughput, multi-broker).
  - **Scheduling**: Kubernetes CronJobs.
  - **Database**: Postgres on Kubernetes.
  - **In-Memory Store**: Redis on Kubernetes.
  - **Dapr**: Deployed on Kubernetes for cluster-wide resilience.
- **Training**: Use Oracle Cloud's free tier to train devs on Kubernetes DevOps, ensuring skills for any cloud (AWS, GCP, Azure).
- **Scalability**: Millions of users (e.g., 10,000 req/s on 10 nodes with GPUs), limited by cluster size.
- **Cost**: Compute-focused ($1-2/hour/node), no API fees.

Conceptualizing an Email Agent: Our Mind Map Brainstorm

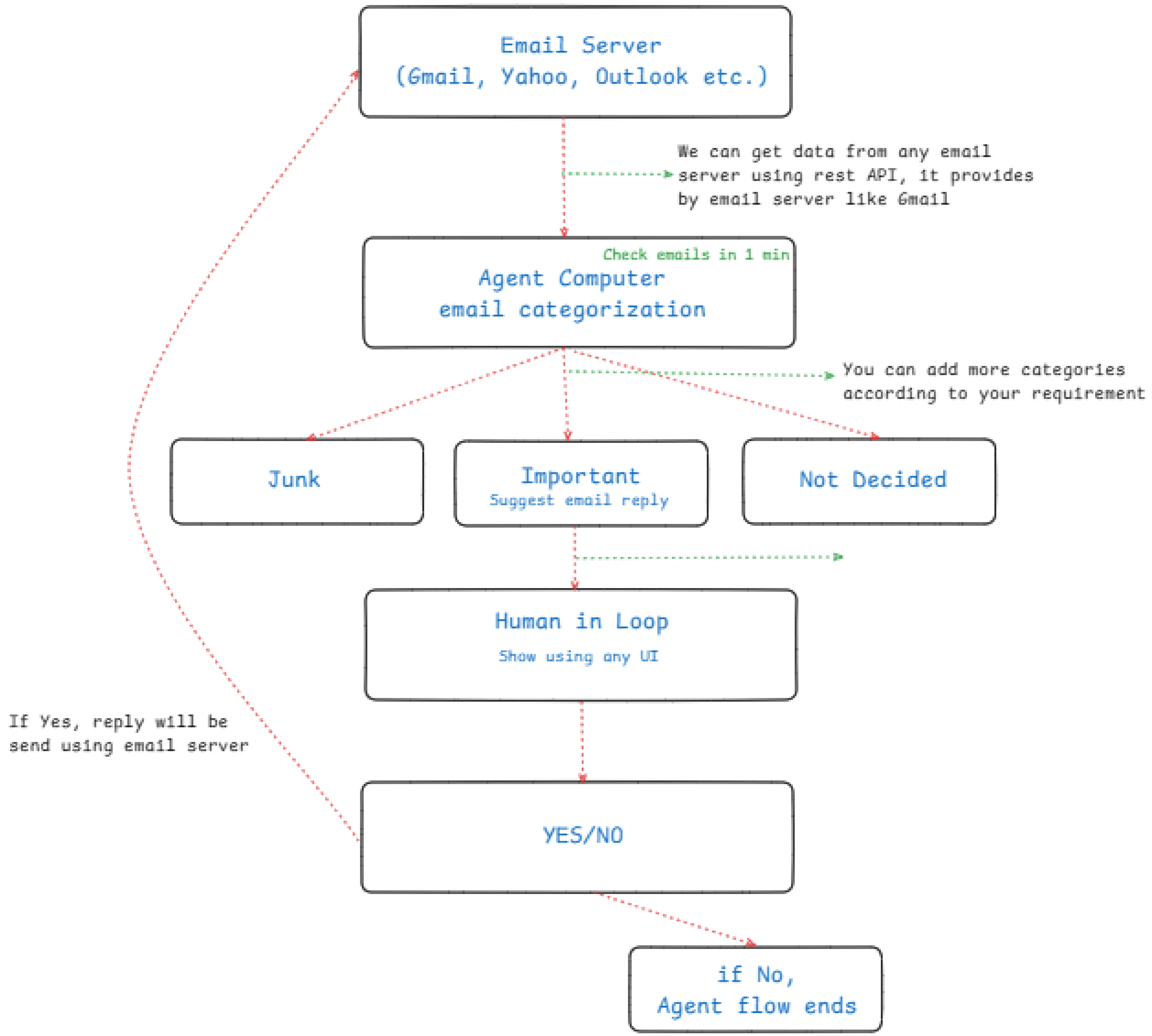
Discussion:

- Mind mapping for email agent
- We have to mind mapping email agent before execution, like my agent will be categorizing email in 3 or 4 categories i.e., important, less important, junk etc.



Conceptualizing an Email Agent: Our Mind Map Brainstorm

Discussion



Conceptualizing an Email Agent: Our Mind Map Brainstorm

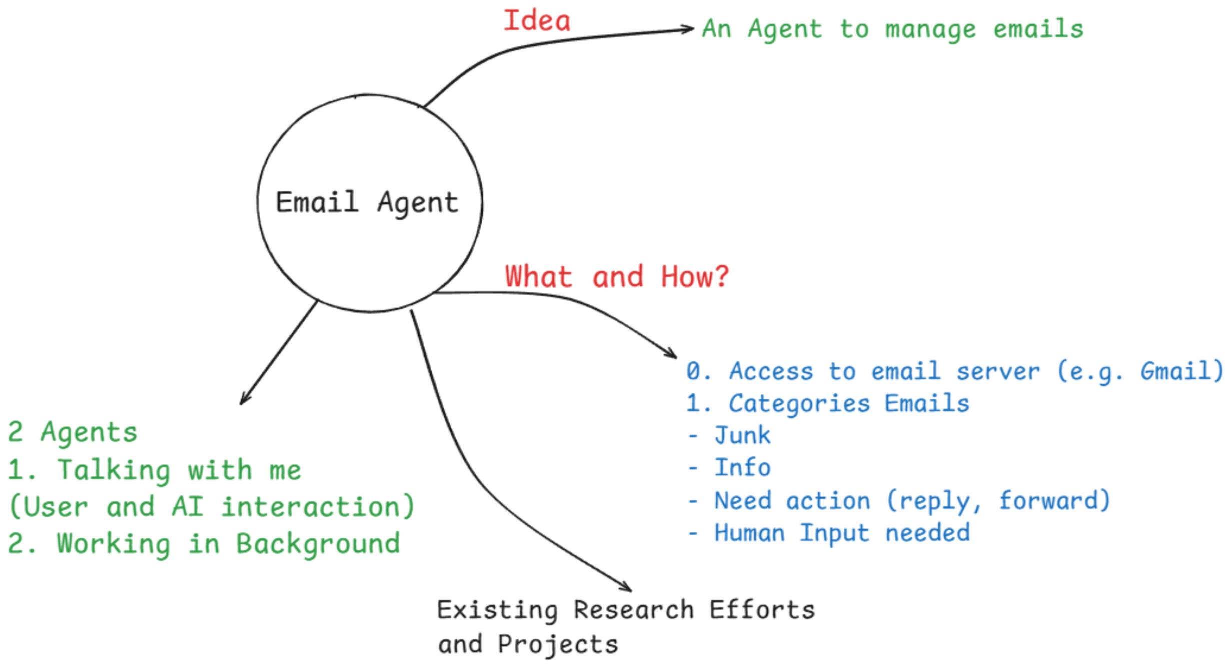
Discussion:

- Above flowchart is just an example of email agent, we can improve it and can add new features.
- Same is the case for below diagram

Conceptualizing an Email Agent: Our Mind Map Brainstorm

Discussion (cont.):

- There is a difference between ERD (Entity Relationship diagram) and mind mapping, ERD is unstructured and use to define database schema.
- Sir Zia suggested that there could be 2 agents, one is for user and agent interaction and other will be working in background.

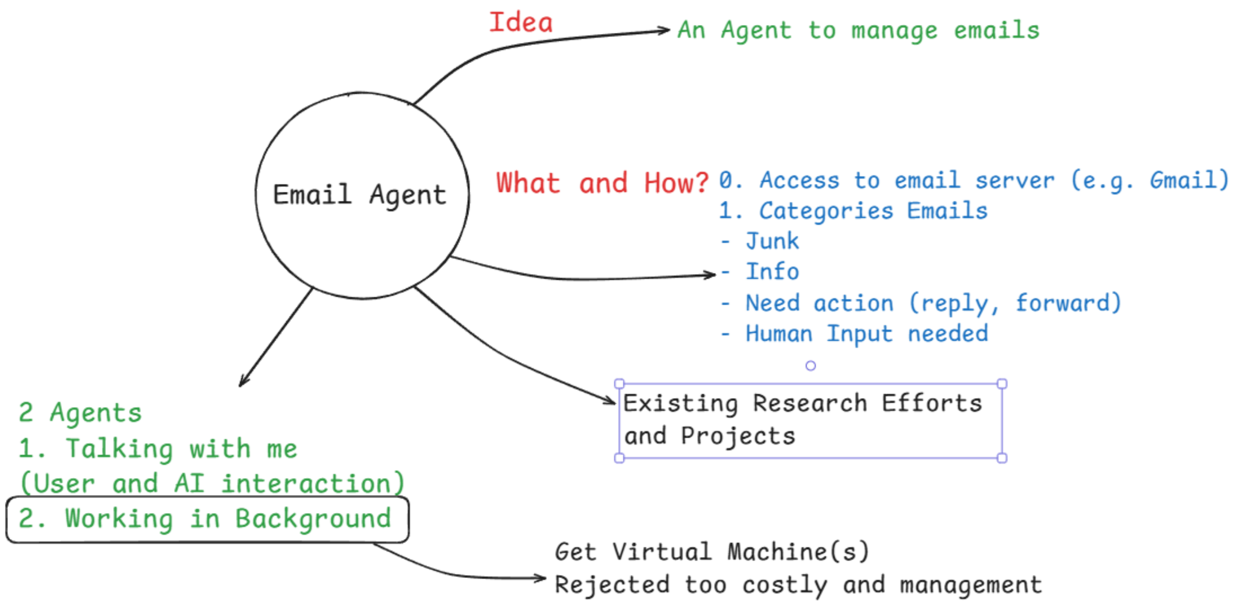


Conceptualizing an Email Agent: Our Mind Map Brainstorm

Discussion (cont.):

- But the question is, how will you implement and manage background agent which is running autonomously? You should think about cloud point of view, it is easy to implement it on your machine because of less users and when internet or your machine close, agent will stop working.

- You get virtual machine from amazon or some other provider but it may get error or any other technical issue then it will be close/not work, this idea is rejected



Conceptualizing an Email Agent: Our Mind Map Brainstorm

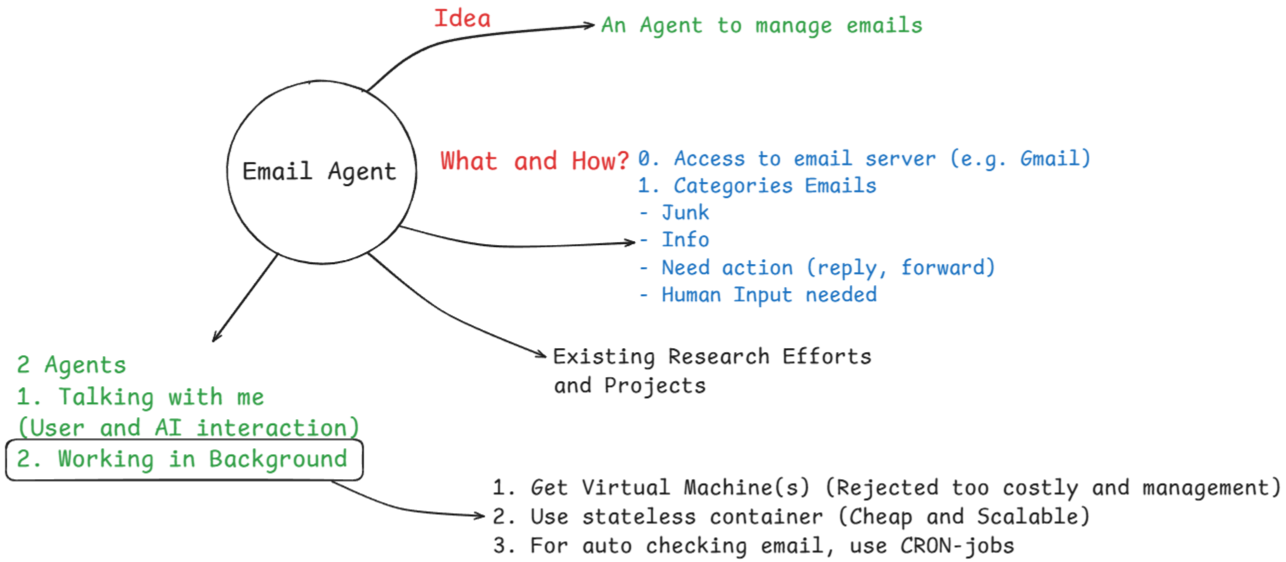
Discussion (cont.):

- Now a days, everyone using is stateless container, so we will used the same in this case

- If we used container then we have to make sure when will agent take detail from email server because although stateless server are cheap but it incurred some cost when container is up.

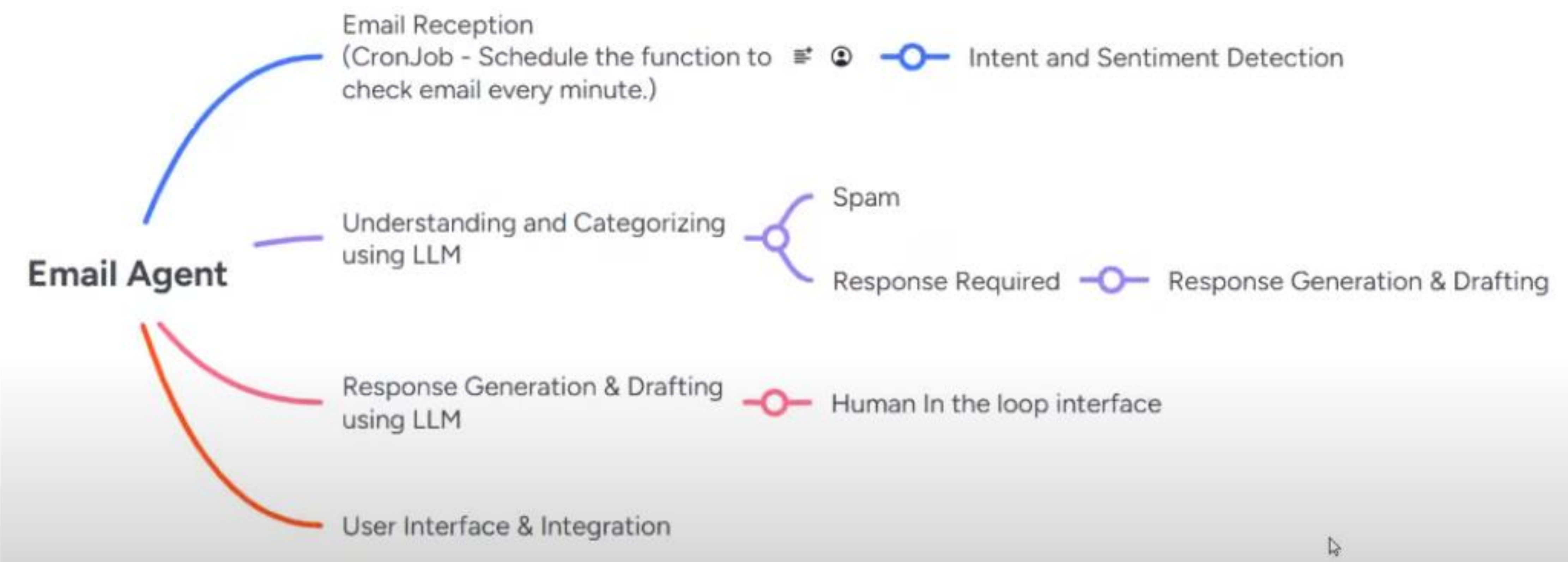
- 2 types of stateless container 1) User Event based 2) Schedule Event based, which are called CRON-jobs.

- CRON-jobs helps to trigger event



Conceptualizing an Email Agent: Our Mind Map Brainstorm

Discussion (cont.) Sir Ameen Take:



Conceptualizing an Email Agent: Our Mind Map Brainstorm

Discussion (cont.) Sir Qasim take on it:

- We work on 3 tier application architecture
  - On first layer, we make user interface
  - On second layer, we write business logic like where it will run?
  - On last layer, we have data layer
- Whatever API you used like OpenAI, Gemini etc. they all give you plan of pay as you go, it means you only pay for resources which you used. This is possible due to serverless container/stateless container.
- If you want to run code 24x7, it will incur some cost, we want to make such systems when user come and assign task to our server (using UI) then small machine or computer on/up, process user request and give back answer, then it off/down the machine/computer

Conceptualizing an Email Agent: Our Mind Map Brainstorm

Discussion (cont.) Sir Qasim take on it:

- When request receive it should be on/up, as soon as response delivered, it should be off/down.
- If this functionality you can apply on your computer or server, that is called **serverless computing**.
- Serverless computing because of ‘**microservices architecture plus container**’.
- Request received ➔ computer on/up ➔ data which user want us to process is in some data layer ➔ process that data ➔ give response to user ➔ simultaneously save that data in data layer ➔ computer off/down again

Conceptualizing an Email Agent: Our Mind Map Brainstorm

Discussion (cont.) Sir Qasim take on it:

- This is how serverless container works, by using this logic scalability is possible in today's world doesn't matter million or billion users comes to your app.
- This serverless computing (EDA → Event Driven Architecture) will possible due to DAPR, which we will discuss later.

Conceptualizing an Email Agent: Our Mind Map Brainstorm

Discussion (cont.):

- For asynchronous communication, we used Kafka, Rabbit MQ, Redis Streams etc. these all are message broker software.
- For synchronous communication, we will used RestAPI.
- In a scalable system, we use something called a **Load Balancer**. This sits at the top of the system and handles all user requests.
- When a user sends a request, it first goes to the load balancer. The load balancer checks which server (or container) has space. Then, it sends the user to that container.
- If a container stops working (crashes or shuts down), the load balancer will move the user to another working container.

Conceptualizing an Email Agent: Our Mind Map Brainstorm

Discussion (cont.):

But what happens to the user's data when the container crashes?

Here's how it works:

- Our system is **stateless**. This means we **do not save user data inside the container**.
- Instead, when a function runs and returns some data (called **state**), we **save that state in a separate storage**, **outside** the container.
- So, even if a container crashes, the user's data is safe in that outside storage.

- When the user is moved to a new container, the system can get the saved data from that outside storage and continue as normal.
- There is also something called a **stickiness policy**. It tries to keep the user connected to the same container, so if the user reloads the page, they still see the same data. But if the container crashes, the user is moved to a new one, and the data is loaded again from storage.

Unleash AI Agents at Scale: Cloud-Native Strategies & Real-World Use Cases Explored

Discussion:

- **“How do we design AI Agents that can handle 10 million concurrent AI Agents without failing?”**
- GKE ➔ Google Kubernetes Engine
- CKAD➔ Certification Kubernetes Application Developer
  - https://training.linuxfoundation.org/certification/certified-kubernetes-application-developer-ckad/ ➔ $ 445 cost
  - It’s all about Linex, standard in one-two

Unleash AI Agents at Scale: Cloud-Native Strategies & Real-World Use Cases Explored

Discussion (cont.):

- Way of Preparation of CKAD:
  - Sir provide setup Linux commands list
  - Then we need Kubernetes cluster to run services and code
- That is the reason, sir added Rancher in our course and remove Docker, Rancher is a manager, it will install K3S (small Kubernetes) on your machine.

Condensed Argument with Proof and Logic:

1. Kubernetes Scalability:

- Kubernetes supports up to 5,000 nodes (machine) and 150,000 pods per cluster, container are in pods (Kubernetes docs), with real-world examples like PayPal scaling to 4,000 nodes and 200,000 pods (InfoQ, 2023) and KubeEdge managing 100,000 edge nodes and 1 million pods (KubeEdge case studies).For more detail → https://github.com/panaversity/learn-agentic-ai/blob/main/README.md

Condensed Argument with Proof and Logic:

2. Dapr’s Efficiency for Agentic AI:

- **Evidence:** Dapr’s actor model supports thousands of virtual actors per CPU core with double-digit millisecond latency (Dapr docs, 2024). Case studies show Dapr handling millions of events, e.g., Tempestive’s IoT platform processing billions of messages (Dapr blog, 2023) and DeFacto’s system managing 3,700 events/second (320 million daily) on Kubernetes with Kafka (Microsoft case study, 2022).
- **Logic:** Agentic AI relies on stateful, low-latency agents. Dapr Agents, built on the actor model, can represent 10 million users as actors, distributed across a Kubernetes cluster. Dapr’s state management (e.g., Redis) and pub/sub messaging (e.g., Kafka) ensure efficient coordination and resilience, with automatic retries preventing failures. Sharding state stores and message brokers scales to millions of operations/second.

Condensed Argument with Proof and Logic:

3. Handling AI Workloads:

- **Evidence**: LLM inference frameworks like vLLM and TGI serve thousands of requests/second per GPU (vLLM benchmarks, 2024). Kubernetes orchestrates GPU workloads effectively, as seen Kubernetes manages GPU workloads, as seen in NVIDIA's AI platform scaling to thousands of GPUs (NVIDIA case study, 2023).
- **Logic**: Assuming each user generates 1 request/second requiring 0.01 GPU, 10 million users need ~100,000 GPUs. Batching, caching, and model parallelism reduce this to a feasible ~10,000–20,000 GPUs, achievable in hyperscale clouds (e.g., AWS). Kubernetes' resource scheduling ensures optimal GPU utilization.

Condensed Argument with Proof and Logic:

4. Networking and Storage:

- **Evidence**: EMQX on Kubernetes handled 1 million concurrent connections with tuning (EMQX blog, 2024). C10M benchmarks (2013) achieved 10 million connections using optimized stacks. Dapr’s state stores (e.g., Redis) support millions of operations/second (Redis benchmarks, 2024).
- **Logic**: 10 million connections require ~100–1,000 Gbps bandwidth, supported by modern clouds. High-throughput databases (e.g., CockroachDB) and caching (e.g., Redis Cluster) handle 10 TB of state data for 10 million users (1 KB/user). Kernel bypass (e.g., DPDK) and eBPF-based CNIs (e.g., Cilium) minimize networking latency.

Condensed Argument with Proof and Logic:

5. Resilience and Monitoring:

- **Evidence**: Dapr’s resiliency policies (retries, circuit breakers) and Kubernetes’ self-healing (pod restarts) ensure reliability (Dapr docs, 2024). Dapr’s OpenTelemetry integration scales monitoring for millions of agents (Prometheus case studies, 2023).
- **Logic**: Real-time metrics (e.g., latency, error rates) and distributed tracing prevent cascading failures. Kubernetes’ liveness probes and Dapr’s workflow engine recover from crashes, ensuring 99.999% uptime.

Condensed Argument with Proof and Logic:

Feasibility with Constraints:

- **Challenge**: No direct benchmark exists for 10 million concurrent users with Dapr/Kubernetes in an agentic AI context. Infrastructure costs (e.g., $10M–$100M for 10,000 nodes) are prohibitive for low-budget scenarios.
- **Solution**: Use open-source tools (e.g., Minikube, kind) for local testing and cloud credits (e.g., AWS Educate) for students. Simulate 10 million users with tools like Locust on smaller clusters (e.g., 100 nodes), extrapolating results. Optimize Dapr's actor placement and Kubernetes' resource quotas to maximize efficiency on limited hardware. Leverage free-tier databases (e.g., MongoDB Atlas) and message brokers (e.g., RabbitMQ).
- **Conclusion**: Kubernetes with Dapr can handle 10 million concurrent users in an agentic AI system, supported by their proven scalability, real-world case studies, and logical extrapolation. For students with minimal budgets, small-scale simulations, open-source tools, and cloud credits make the problem tractable, though production-scale deployment requires hyperscale resources and expertise.

Supply Chain Optimization Agent

https://github.com/panaversity/learn-agentic-ai/blob/main/-01_lets_get_started/03_from_llms_to_stateful_long_runningl_multi_agents/02b_supply_chain_agent_architecture.md

Requirements for the Supply Chain Optimization Agent

Functional Requirements

- **Inventory Monitoring and Alerts**:
  - Continuously monitor inventory levels across warehouses.
  - Detect low stock or overstock conditions and suggest restocking or redistribution actions.
  - Notify managers with suggested actions for approval.
- **Route Optimization**:
  - Automatically optimize delivery routes based on real-time data (e.g., traffic, order volume).
  - Suggest optimized routes to drivers or fleet managers for approval before execution.
- **Action Approval and Execution**:
  - Allow managers to approve, modify, or reject suggested inventory actions or routes.
  - Execute approved actions (e.g., dispatch trucks, order stock).
- **Manual Route Adjustment**:
  - Enable managers to request custom route adjustments (e.g., prioritize a VIP customer).
  - Agent verifies and optimizes the request, suggesting improvements, and seeks approval before finalizing.

Healthcare Patient Monitoring Agent Architecture

https://github.com/panaversity/learn-agentic-ai/blob/main/-01_lets_get_started/03_from_llms_to_stateful_long_runningl_multi_agents/02c_patient_monitoring_agent_architecture.md

Requirements for the Healthcare Patient Monitoring Agent

Functional Requirements

- **Vitals Monitoring and Anomaly Detection**:
  - Continuously monitor patient vitals (e.g., heart rate, blood pressure, oxygen levels) from wearable devices or hospital sensors.
  - Detect anomalies (e.g., high heart rate, low oxygen) and suggest interventions (e.g., "Administer oxygen").
  - Notify healthcare professionals with suggested actions for approval.
- **Intervention Suggestions**:
  - Analyze vitals data and recommend actions based on medical guidelines or AI models (e.g., "Increase dosage," "Order ECG").
  - Present suggestions to doctors or nurses for approval.
- **Action Approval and Execution**:
  - Allow healthcare professionals to approve, modify, or reject suggested interventions.
  - Execute approved actions (e.g., update patient records, notify staff).
- **Manual Follow-Up Requests**:
  - Enable doctors to request custom follow-ups (e.g., "Schedule a blood test").
  - Agent verifies the request against patient data, suggests optimizations (e.g., timing, test type), and seeks approval before scheduling.

Financial Trading Agent Architecture

https://github.com/panaversity/learn-agentic-ai/blob/main/-01_lets_get_started/03_from_llms_to_stateful_long_runningl_multi_agents/02d_financial_trading_agent_architecture.md

Requirements for the Financial Trading Agent

Functional Requirements

- **Market Monitoring and Trade Suggestions**:
  - Continuously monitor real-time market data (e.g., stock prices, forex rates, volume).
  - Detect trading opportunities based on strategies (e.g., moving average crossover, arbitrage) and suggest buy/sell actions.
  - Notify traders with suggested trades for approval.
- **Trade Analysis and Suggestions**:
  - Analyze market conditions and suggest trade parameters (e.g., price, quantity, stop-loss).
  - Present suggestions to traders for review and approval.
- **Trade Approval and Execution**:
  - Allow traders to approve, modify, or reject suggested trades.
  - Execute approved trades via a brokerage API.
- **Manual Trade Requests**:
  - Enable traders to request custom trades (e.g., "Buy 100 shares of AAPL").
  - Agent assesses risk, optimizes the trade (e.g., adjusts timing or quantity), and seeks approval before execution.

Personalized Learning Agent Architecture

https://github.com/panaversity/learn-agentic-ai/blob/main/-01_lets_get_started/03_from_llms_to_stateful_long_runningl_multi_agents/02e_personalized_learning_agent_architecture.md

Requirements for the Personalized Learning Agent

Functional Requirements

- **Student Progress Monitoring and Suggestions**:
  - Continuously monitor student activity (e.g., quiz scores, time spent on lessons, engagement metrics).
  - Detect learning gaps or strengths and suggest personalized activities (e.g., "Practice fractions quiz," "Watch algebra video").
  - Notify teachers and/or students with suggested activities for approval.
- **Learning Activity Suggestions**:
  - Analyze student performance and preferences to recommend tailored content or interventions.
  - Present suggestions to teachers or students for review and approval.
- **Activity Approval and Assignment**:
  - Allow teachers (or students, depending on context) to approve, modify, or reject suggested activities.
  - Assign approved activities to students via the platform.
- **Manual Assignment Requests**:
  - Enable teachers to request custom assignments or interventions (e.g., "Assign essay on WWII").
  - Agent assesses student readiness, optimizes the request (e.g., adjusts difficulty), and seeks approval before assigning.
- **Non-Functional Requirements**

Blood Bank ERP Agent Architecture

https://github.com/panaversity/learn-agentic-ai/blob/main/-01_lets_get_started/03_from_llms_to_stateful_long_runningl_multi_agents/02f_blook_bank_erp_agent_architecture.md

Requirements for the Enhanced Blood Bank ERP Agent

Functional Requirements

- **Blood Tracking with Barcodes**:
  - Track blood units from donation to distribution using unique barcodes.
  - Monitor status (e.g., collected, tested, stored, dispatched, expired).
  - Notify staff of expiring units or low inventory.
- **Donor Relationship Management**:
  - Maintain donor profiles (e.g., contact info, donation history, blood type, preferences).
  - Calculate and notify donors when they can safely donate again (e.g., 56 days for whole blood, 7 days for platelets).
  - Automate personalized donor engagement (e.g., thank-you messages, campaigns).
- **Inventory Optimization**:
  - Automatically suggest restocking, redistribution, or disposal based on inventory levels and expiry dates.
  - Predict blood demand using historical data and external factors (e.g., seasonal trends, emergencies).
- **Demand Forecasting and Alerts**:
  - Forecast blood needs for hospitals or regions.
  - Alert staff and donors during shortages (e.g., "Urgent need for O- blood").
- **Action Approval and Execution**:
  - Allow staff to approve, modify, or reject suggested actions (e.g., discard units, contact donors).
  - Execute approved actions (e.g., update inventory, send notifications).
- **Manual Staff Requests**:
  - Enable staff to request donor outreach or blood unit actions (e.g., "Transfer 20 units to Hospital Y").
  - Agent optimizes requests (e.g., prioritizes donors, checks stock) and seeks approval.

Shopping Cart and Inventory Management Agent Architecture

https://github.com/panaversity/learn-agentic-ai/blob/main/-01_lets_get_started/03_from_llms_to_stateful_long_runningl_multi_agents/02g_shopping_cart_agent_architecture.md

Requirements for the Enhanced Blood Bank ERP Agent

Functional Requirements

- **Shopping Cart Management**:
  - Allow customers to add/remove items to/from their cart.
  - Check real-time inventory availability before adding items.
  - Notify customers if items are low stock or out of stock with alternatives.
- **Inventory Management**:
  - Track inventory levels across warehouses or stores using product IDs.
  - Suggest restocking, redistribution, or markdowns based on stock levels.
  - Notify staff of inventory issues (e.g., low stock, overstock).
- **Action Approval and Execution**:
  - Allow staff to approve, modify, or reject suggested inventory actions (e.g., "Restock 50 units").
  - Execute approved actions (e.g., update inventory, notify suppliers).
- **Manual Staff Requests**:
  - Enable staff to request inventory adjustments (e.g., "Transfer 20 units to Store A") or promotions (e.g., "Discount overstocked items").
  - Agent optimizes requests (e.g., prioritizes nearby stock, suggests discount rates) and seeks approval.

Social Media Account Management Agent Architecture

https://github.com/panaversity/learn-agentic-ai/blob/main/-01_lets_get_started/03_from_llms_to_stateful_long_runningl_multi_agents/02h_social_media_agent_architecture.md

Requirements for the Social Media Account Management Agent

Functional Requirements

- **Account Activity Monitoring**:
  - Monitor user social media accounts for activity (e.g., mentions, comments, follower growth).
  - Suggest responses to comments/mentions or new posts based on trends and engagement.
  - Notify users of suggested actions for approval.
- **Content Management**:
  - Suggest post ideas or captions based on user preferences, trending topics, or analytics.
  - Notify users when content is ready for review and posting.
- **Action Approval and Execution**:
  - Allow users to approve, edit, or reject suggested posts or responses.
  - Execute approved actions (e.g., post content, reply to comments) via platform APIs.
- **Manual User Requests**:
  - Enable users to request content schedules (e.g., "Plan 5 posts for next week") or engagement actions (e.g., "Reply to followers").
  - Agent optimizes requests (e.g., adjusts timing, enhances content) and seeks approval.

Customer Acquisition and Management Agent Architecture

https://github.com/panaversity/learn-agentic-ai/blob/main/-01_lets_get_started/03_from_llms_to_stateful_long_runningl_multi_agents/02i_customer_acquisition_management_agent_architecture.md

Requirements for the Customer Acquisition and Management Agent (LinkedIn)

Functional Requirements

- **Customer Identification**:
  - Monitor LinkedIn for potential customers based on profiles (e.g., job titles, industries, interests) and activity (e.g., posts, comments).
  - Suggest prospects matching the company's target audience (e.g., "IT Manager at TechCorp").
  - Notify sales teams with prospect details for outreach approval.
- **Outreach and Sales**:
  - Suggest personalized connection requests, messages, or InMails to engage prospects.
  - Notify sales reps with proposed outreach content for approval before sending.
- **Relationship Management**:
  - Track interactions (e.g., messages, meetings) and suggest follow-ups or nurturing actions (e.g., "Share industry article").
  - Notify sales reps to maintain relationships with prospects/customers.
- **Manual Sales Requests**:
  - Allow sales reps to request outreach campaigns (e.g., "Target 50 CFOs") or specific actions (e.g., "Follow up with John Doe").
  - Agent optimizes requests (e.g., refines target list, crafts messages) and seeks approval.

Other Focus Areas for Agentic AI

https://github.com/panaversity/learn-agentic-ai/blob/main/-01_lets_get_started/03_from_llms_to_stateful_long_runningl_multi_agents/02j_other_areas.md

- Agriculture and Farming
- Transportation and Logistics (Beyond Supply Chain)
- Environmental Monitoring
- Entertainment and Media
- Human Resources (HR)
- Real Estate
- Gaming Industry
- Legal and Compliance
- Travel and Tourism
- Cybersecurity
- Fitness and Wellness
- Government and Public Services
- Manufacturing
- Event Planning
- Charity and Non-Profit

OpenRouter

- A unified interface for 50+ free LLMs
- You can connect by both ways, via chat interface and via API
- Docs link → https://openrouter.ai/docs/quickstart#using-the-openrouter-api-directly