Write a short LinkedIn post reflecting on how unpredictable weather teaches resilience in AI research. Use today's weather in Boston as a metaphor. Keep it insightful, no more than 4 sentences

```
cd /Users/sreebhargavibalija/Library/Application Support/claude
```

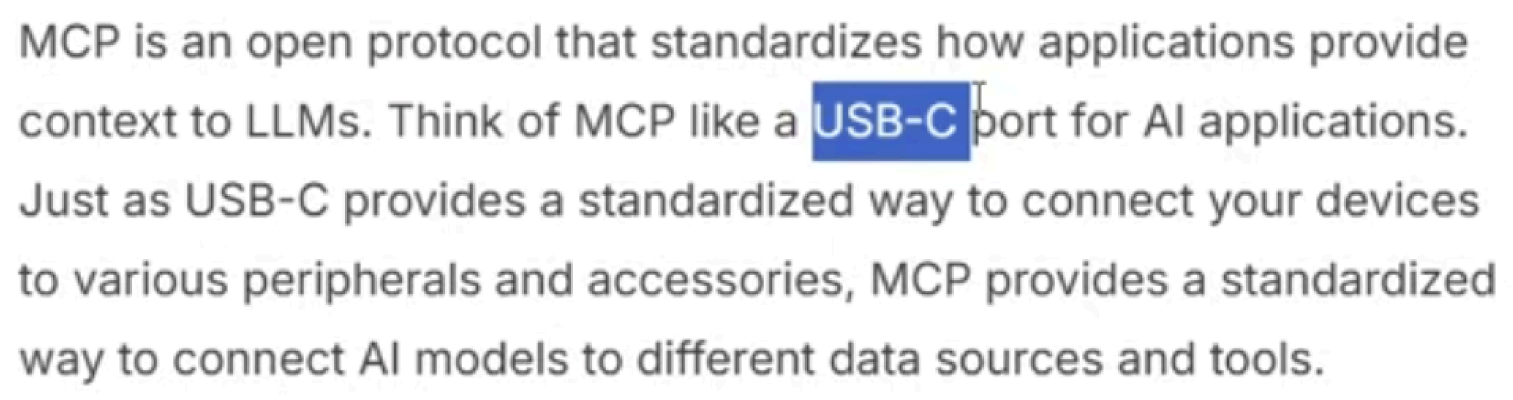
final command

```
./run_mcp_servers.sh > log.txt 2>&1
tail -f log.txt
```

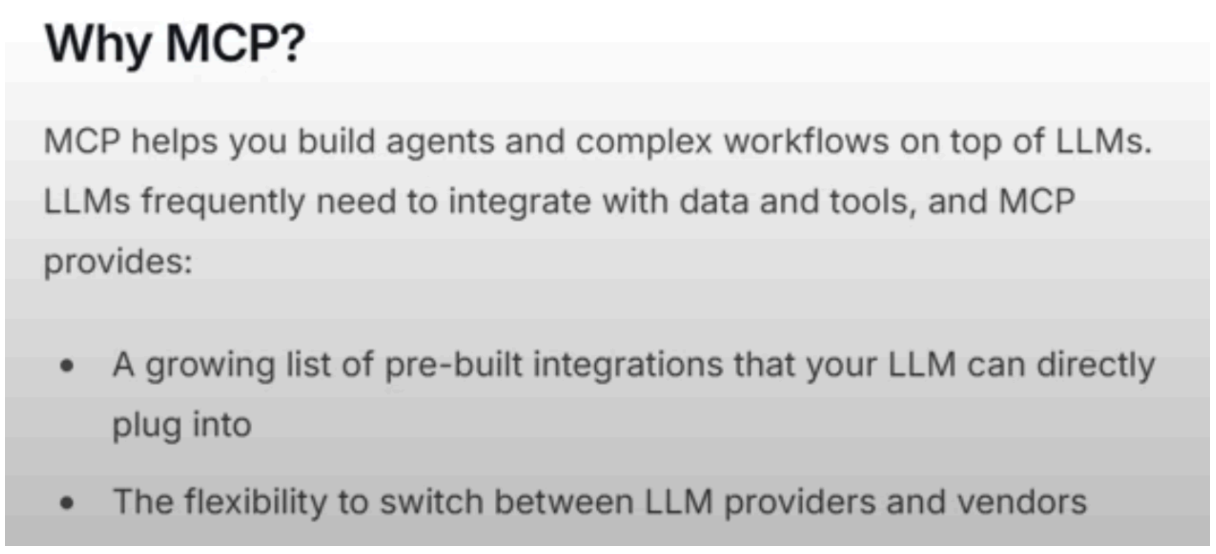
This code sets up a **FastMCP server** named `"weather"` that provides tools to fetch **weather alerts** and **forecasts** from the **National Weather Service (NWS) API**.
It defines an async helper function `make_nws_request()` to make HTTP GET requests with error handling.
The `get_alerts()` tool retrieves active weather alerts for a given US state using its two-letter code.
The `get_forecast()` tool fetches the forecast for a specific geographic location based on latitude and longitude.
When run directly, the MCP server starts and communicates via **stdio**, enabling use in platforms like Claude's Desktop Agent.

first demo

mcp is standardized



MCP is an open protocol that standardizes how applications provide context to LLMs. Think of MCP like a USB-C port for AI applications. Just as USB-C provides a standardized way to connect your devices to various peripherals and accessories, MCP provides a standardized way to connect AI models to different data sources and tools.



Why MCP?

MCP helps you build agents and complex workflows on top of LLMs. LLMs frequently need to integrate with data and tools, and MCP provides:

- A growing list of pre-built integrations that your LLM can directly plug into
- The flexibility to switch between LLM providers and vendors

Without `claude_desktop_config`, Claude Desktop wouldn't know **how to find, interpret, or invoke** your MCP-based tool. It's like a manifest file that makes your tool **plug-and-play** with the Claude ecosystem.

second demo

🎤 [Start of Demo] — Setting the Stage (~1 min)

"Now let's see Agentic AI in action.
What you're about to see is a **LangGraph-powered agent** that can **remember**, **reason**, and **use tools** to autonomously assist

a user — just like a human collaborator."

"Think of this agent not as a chatbot, but as an intelligent system that **maintains memory**, **uses APIs**, and **adapts over time**. And everything you're seeing is built using open-source LangGraph and LangChain."

“We’ve set up a clean Streamlit interface where the agent and user can interact naturally. It includes a sidebar to reset memory and a main area to chat.”

“At the heart of the agent is **ConversationBufferMemory**. This memory lets the agent retain everything you say across turns — which makes it intelligent, not stateless like typical chatbots.”

"Let’s begin by giving the agent a task: planning a trip to Paris.
I’ll say: *‘I’m planning a trip to Paris this summer. Can you help me?’*"

"Now I’ll follow up without repeating the full context:
‘What are some offbeat things to do near the Eiffel Tower?’
Notice how it uses **memory** to maintain context — just like a human would."

"Now I’ll ask:
‘Can you remind me what I said earlier?’
This shows that the agent isn’t stateless — it remembers the user’s intent."

“Notice that the agent keeps track of our earlier conversation and uses that memory in future responses — no need to repeat yourself. This is real contextual awareness.”

🧰 [Part 2: Tool Use]

"Next, let’s activate a tool. I’ll ask the agent:
‘Can you search what’s happening in Paris this July?’
Behind the scenes, the agent is invoking a **custom `Search` tool** — simulating what a real-world autonomous agent would do by calling APIs."

"This tool-using ability makes the agent more than just an LLM — it’s a **functional unit** capable of interacting with its environment."

“Here the agent is invoking a **custom search tool**, defined in the code as `custom_search()`.
It simulates calling an external API like Tavily, returning live-like results that enhance the agent’s utility.
This shows the agent can go beyond language — it’s capable of tool use like a functional system.”

🛠️ [Part 3: Goal-Directed Reasoning]

"Now I’ll challenge the agent with a goal:
‘Plan a 3-day itinerary based on what we discussed.’
This is where LangGraph shines — managing the flow, invoking memory, and guiding the LLM to stay goal-aligned."

"The agent combines **dialogue history**, **tools**, and **planning**, to generate personalized, multi-step responses."

“This is where LangGraph and LangChain shine — the agent uses **tools + memory** to reason over past interactions and generate a goal-aligned, multi-step plan.
The logic here isn’t hardcoded — the agent is empowered to decide when to use this itinerary tool.”

🧩 [Part 4: Reflection & Intent Inference (optional)]

"Agents can even reflect — imagine asking:
What do you think is my goal in this conversation?
With the right nodes, LangGraph can model **self-awareness** through a reflection module — a step closer to true autonomous behavior."

“Here, the agent uses the `ReflectGoal` tool to **infer user intent** based on conversation history.

This is an early form of reflection — where the agent attempts to reason about what you want and why you're asking questions.

✨ [Closing the Demo]

> "This wasn't just a chatbot demo — this was a **modular, memory-powered, multi-agent system**.
> It reasons. It plans. It adapts.
> This is Agentic AI — a new paradigm where we build agents that think and act over time, just like us."
>
> "And the beauty? It's all programmable, inspectable, and extensible. You can add a planner, a reflector, a multi-tool orchestrator — LangGraph gives us that flexibility."

✅ Pro Tip for Closing Slide

> "We're moving from **prompt engineering** to **agent architecture engineering**.
> This isn't about feeding the right prompt — it's about **designing the right behavior**."

This script sets up an LLM-powered assistant using the `agno` framework. It loads API keys for OpenAI and Groq from a `.env` file using `dotenv`, and prints them for confirmation. The agent is configured to use OpenAI's `gpt-4` model and the `DuckDuckGoTools` for real-time web search. The assistant is given a description and enabled to format responses in Markdown. Finally, it queries: *"Who won the IPL 2025 final and who was the Player of the Match?"* and prints the response.

4o

third demo

n8n — telegram bot

This n8n template is a **crypto news sentiment analysis bot** that integrates OpenAI GPT-4o with Telegram and multiple crypto RSS feeds. Here's a concise 10-line description:

1. It listens for incoming messages on Telegram using a Telegram Trigger node.
2. The user sends a crypto keyword (e.g., "Bitcoin") or company name.
3. An AI agent extracts the keyword from the user's message.
4. This keyword is used to filter articles from 10+ major crypto news RSS sources (e.g., Cointelegraph, Coindesk).
5. All RSS articles are merged and filtered for relevance using JavaScript.
6. A dynamic GPT-4o prompt is constructed with these filtered articles.
7. GPT-4o then summarizes the news, analyzes sentiment, and formats the result.
8. The summarized output includes a news digest, sentiment overview, and clickable article links.
9. The summary is sent back to the user on Telegram via the `Sends Response` node.
10. The template is modular, supports session tracking, and allows scalable news monitoring automation.

foruth demo

MAIA is the first multimodal agent that automates model interpretability by composing and executing vision-language experiments to explain neural behaviors.

It generates hypotheses, synthesizes or edits images, probes deep models, and iteratively updates its understanding—just like a human interpretability researcher.

Equipped with tools like image generation, activation analysis, and visual summarization, MAIA explains neurons and identifies failure modes or biases without retraining.

MAIA’s explanations rival expert human annotations and outperform prior automated baselines, even on synthetic neurons with ground-truth labels.

By bridging experimentation and explanation, MAIA sets a new paradigm for scalable, automated understanding of complex AI systems.

With great autonomy comes greater responsibility. Trust remains the biggest challenge—can we rely on agents to follow goals without supervision? Tool use poses real-world security risks if not sandboxed. Agents can hallucinate outputs or misinterpret user intent during long plans. Debugging these workflows is tough—unlike static code, agents adapt on the fly. Lastly, we lack proper evaluation metrics to assess how "intelligent" or safe an agent really is. Solving these will be key to unlocking the next era of trustworthy AI.

As we look ahead, LangGraph is evolving into a modular system for agentic workflows—soon supporting streaming data, persistent memory, and multi-agent graphs. Meanwhile, Claude Agents are redefining desktop interactivity by giving AI tools the ability to use apps like you and me. OpenAgents is pushing toward transparent, reproducible autonomous stacks. The big shift? Agents becoming the new operating system layer—running apps, browsing web, editing files, and even talking to each other. We're moving from LLMs answering questions to LLMs achieving goals autonomously.