Get Started

Quickstart

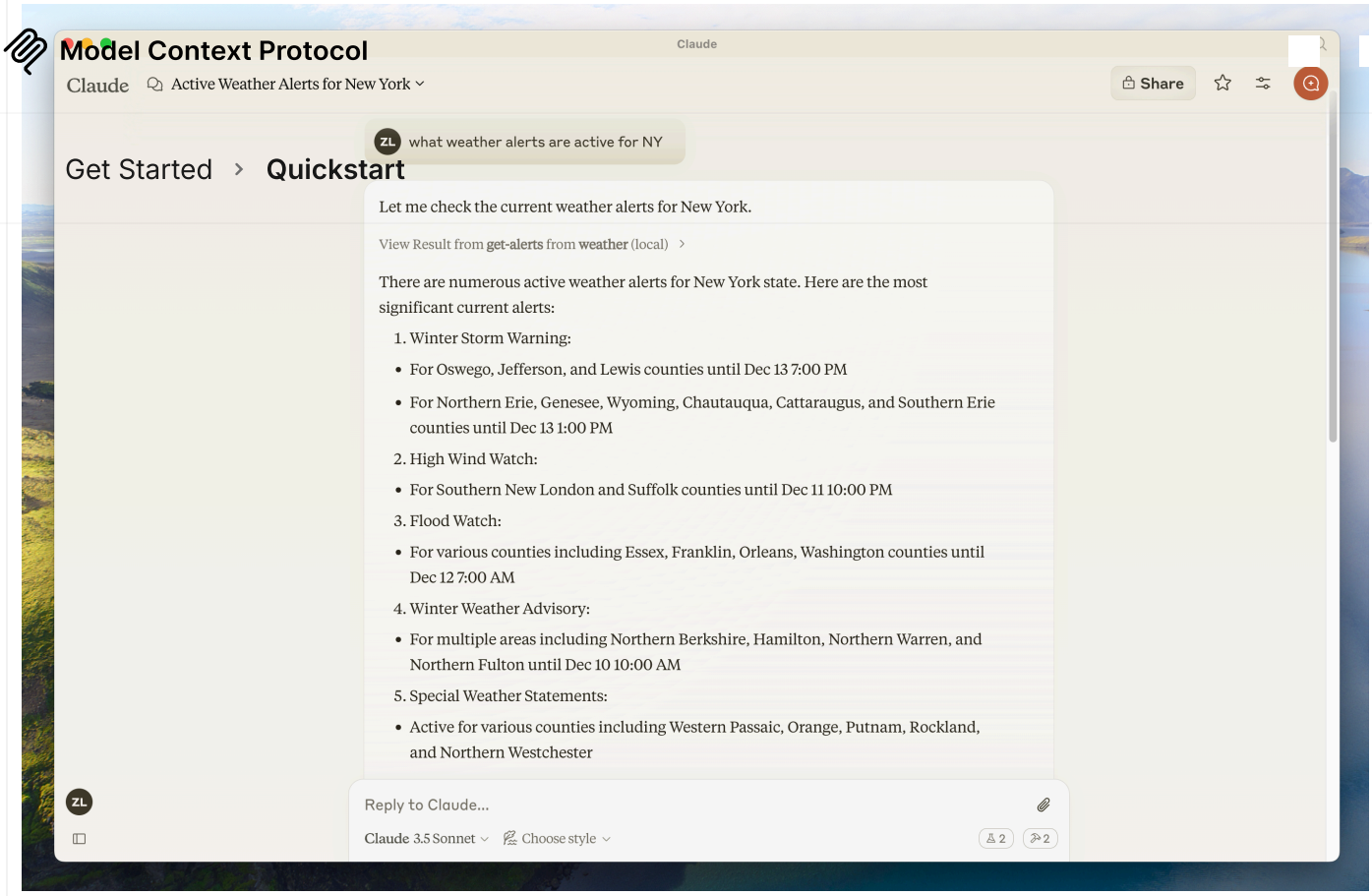
Get started with building your first MCP server and connecting it to a host

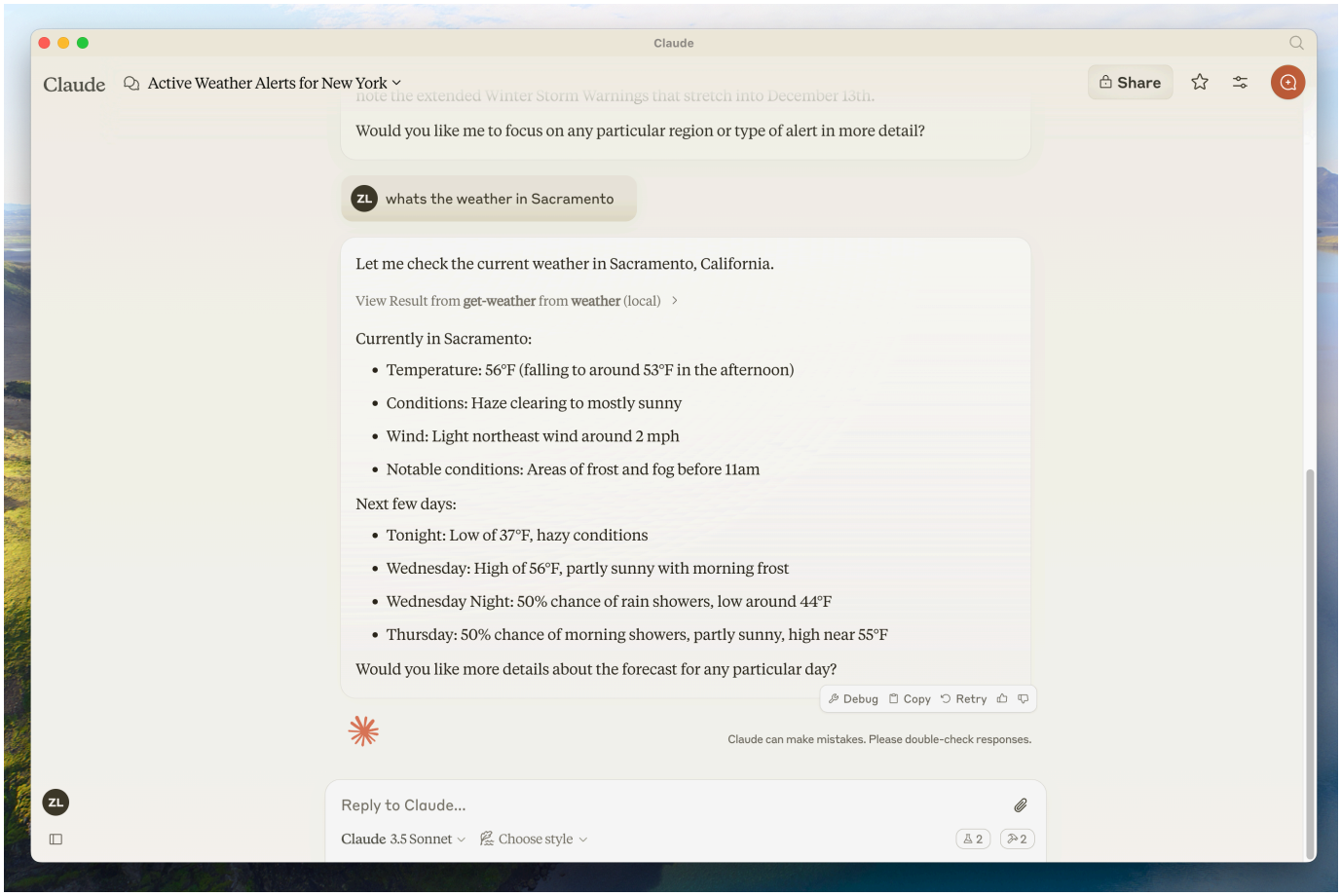
In this tutorial, we'll build a simple MCP weather server and connect it to a host, Claude for Desktop. We'll start with a basic setup, and then progress to more complex use cases.

What we'll be building

Many LLMs (including Claude) do not currently have the ability to fetch the forecast and severe weather alerts. Let's use MCP to solve that!

We'll build a server that exposes two tools: `get-alerts` and `get-forecast`. Then we'll connect the server to an MCP host (in this case, Claude for Desktop):


Model Context Protocol
Claude
Active Weather Alerts for New York
Share
Get Started › Quickstart
what weather alerts are active for NY
Let me check the current weather alerts for New York.
View Result from get-alerts from weather (local)
There are numerous active weather alerts for New York state. Here are the most significant current alerts:
1. Winter Storm Warning:
• For Oswego, Jefferson, and Lewis counties until Dec 13 7:00 PM
• For Northern Erie, Genesee, Wyoming, Chautauqua, Cattaraugus, and Southern Erie counties until Dec 13 1:00 PM
2. High Wind Watch:
• For Southern New London and Suffolk counties until Dec 11 10:00 PM
3. Flood Watch:
• For various counties including Essex, Franklin, Orleans, Washington counties until Dec 12 7:00 AM
4. Winter Weather Advisory:
• For multiple areas including Northern Berkshire, Hamilton, Northern Warren, and Northern Fulton until Dec 10 10:00 AM
5. Special Weather Statements:
• Active for various counties including Western Passaic, Orange, Putnam, Rockland, and Northern Westchester
Reply to Claude...
Claude 3.5 Sonnet
Choose style


Claude
Active Weather Alerts for New York
Share
Would you like me to focus on any particular region or type of alert in more detail?
whats the weather in Sacramento
Let me check the current weather in Sacramento, California.
View Result from get-weather from weather (local)
Currently in Sacramento:
• Temperature: 56°F (falling to around 53°F in the afternoon)
• Conditions: Haze clearing to mostly sunny
• Wind: Light northeast wind around 2 mph
• Notable conditions: Areas of frost and fog before 11am
Next few days:
• Tonight: Low of 37°F, hazy conditions
• Wednesday: High of 56°F, partly sunny with morning frost
• Wednesday Night: 50% chance of rain showers, low around 44°F
• Thursday: 50% chance of morning showers, partly sunny, high near 55°F
Would you like more details about the forecast for any particular day?
Debug
Copy
Retry
Claude can make mistakes. Please double-check responses.
Reply to Claude...
Claude 3.5 Sonnet
Choose style

Model Context Protocol

Servers can connect to any client. We've chosen Claude for Desktop here for simplicity, but e also have guides on **building your own client** as well as a **list of other clients here**.

Get Started › **Quickstart**

Why Claude for Desktop and not Claude.ai?

Core MCP Concepts

MCP servers can provide three main types of capabilities:

1. **Resources**: File-like data that can be read by clients (like API responses or file contents)
2. **Tools**: Functions that can be called by the LLM (with user approval)
3. **Prompts**: Pre-written templates that help users accomplish specific tasks

This tutorial will primarily focus on tools.

Python **Node**

Let's get started with building our weather server! **You can find the complete code for what we'll be building here.**

Prerequisite knowledge

This quickstart assumes you have familiarity with:

Python

LLMs like Claude

System requirements

For Python, make sure you have Python 3.9 or higher installed.

Set up your environment

First, let's install `uv` and set up our Python project and environment:

MacOS/Linux Windows

```
curl -LsSf https://astral.sh/uv/install.sh | sh
```

Make sure to restart your terminal afterwards to ensure that the `uv` command gets picked up.

Now, let's create and set up our project:

MacOS/Linux Windows

```
# Create a new directory for our project
uv init weather
cd weather

# Create virtual environment and activate it
uv venv
source .venv/bin/activate

# Install dependencies
uv add mcp httpx

# Remove template file
rm hello.py

# Create our files
mkdir -p src/weather
touch src/weather/__init__.py
touch src/weather/server.py
```

Add this code to `pyproject.toml`:

```
...rest of config

[build-system]
```

```
requires = [ "hatchling",]
build-backend = "hatchling.build"

[project.scripts]
weather = "weather:main"
```

Add this code to `__init__.py`:

src/weather/__init__.py

```
from . import server
import asyncio

def main():
    """Main entry point for the package."""
    asyncio.run(server.main())

# Optionally expose other important items at package level
__all__ = ['main', 'server']
```

Now let's dive into building your server.

Building your server

Importing packages

Add these to the top of your `server.py`:

```
from typing import Any
import asyncio
import httpx
from mcp.server.models import InitializationOptions
import mcp.types as types
from mcp.server import NotificationOptions, Server
import mcp.server.stdio
```

Setting up the instance

Model Context Protocol

Then initialize the server instance and the base URL for the NWS API:

Get Started › **Quickstart**

```
NWS_API_BASE = "https://api.weather.gov"
USER_AGENT = "weather-app/1.0"

server = Server("weather")
```

Implementing tool listing

We need to tell clients what tools are available. The `list_tools()` decorator registers this handler:

```
@server.list_tools()
async def handle_list_tools() -> list[types.Tool]:
    """
    List available tools.
    Each tool specifies its arguments using JSON Schema validation.
    """
    return [
        types.Tool(
            name="get-alerts",
            description="Get weather alerts for a state",
            inputSchema={
                "type": "object",
                "properties": {
                    "state": {
                        "type": "string",
                        "description": "Two-letter state code (e.g. CA, NY)",
                    },
                },
                "required": ["state"],
            },
        ),
        types.Tool(
            name="get-forecast",
```

```
            description="Get weather forecast for a location",
            inputSchema={
                "type": "object",
                "properties": {
                    "latitude": {
                        "type": "number",
                        "description": "Latitude of the location",
                    },
                    "longitude": {
                        "type": "number",
                        "description": "Longitude of the location",
                    },
                },
                "required": ["latitude", "longitude"],
            },
        ),
    ]
```

This defines our two tools: `get-alerts` and `get-forecast`.

Helper functions

Next, let's add our helper functions for querying and formatting the data from the National Weather Service API:

```
async def make_nws_request(client: httpx.AsyncClient, url: str) -> dict[str,   ny]
    """Make a request to the NWS API with proper error handling."""
    headers = {
        "User-Agent": USER_AGENT,
        "Accept": "application/geo+json"
    }

    try:
        response = await client.get(url, headers=headers, timeout=30.0)
        response.raise_for_status()
        return response.json()
```

```
        except Exception:
            return None

def format_alert(feature: dict) -> str:
    """Format an alert feature into a concise string."""
    props = feature["properties"]
    return (
        f"Event: {props.get('event', 'Unknown')}\n"
        f"Area: {props.get('areaDesc', 'Unknown')}\n"
        f"Severity: {props.get('severity', 'Unknown')}\n"
        f"Status: {props.get('status', 'Unknown')}\n"
        f"Headline: {props.get('headline', 'No headline')}\n"
        "---"
    )
```

Implementing tool execution

The tool execution handler is responsible for actually executing the logic of each tool. Let's add it:

```
@server.call_tool()
async def handle_call_tool(
    name: str, arguments: dict | None
) -> list[types.TextContent | types.ImageContent | types.EmbeddedResource]:
    """
    Handle tool execution requests.
    Tools can fetch weather data and notify clients of changes.
    """
    if not arguments:
        raise ValueError("Missing arguments")

    if name == "get-alerts":
        state = arguments.get("state")
        if not state:
            raise ValueError("Missing state parameter")

        # Convert state to uppercase to ensure consistent format
        state = state.upper()
```

```
        if len(state) != 2:
            raise ValueError("State must be a two-letter code (e.g. CA, NY) ,

        async with httpx.AsyncClient() as client:
            alerts_url = f"{NWS_API_BASE}/alerts?area={state}"
            alerts_data = await make_nws_request(client, alerts_url)

            if not alerts_data:
                return [types.TextContent(type="text", text="Failed to retrieve a

            features = alerts_data.get("features", [])
            if not features:
                return [types.TextContent(type="text", text=f"No active alerts fo

            # Format each alert into a concise string
            formatted_alerts = [format_alert(feature) for feature in features[:20
            alerts_text = f"Active alerts for {state}:\n\n" + "\n".join(formatted

            return [
                types.TextContent(
                    type="text",
                    text=alerts_text
                )
            ]
    elif name == "get-forecast":
        try:
            latitude = float(arguments.get("latitude"))
            longitude = float(arguments.get("longitude"))
        except (TypeError, ValueError):
            return [types.TextContent(
                type="text",
                text="Invalid coordinates. Please provide valid numbers for latit
            )]

        # Basic coordinate validation
        if not (-90 <= latitude <= 90) or not (-180 <= longitude <= 180):
            return [types.TextContent(
                type="text",
                text="Invalid coordinates. Latitude must be between -90 and 90, l
            )]
```

```
    async with httpx.AsyncClient() as client:
        # First get the grid point
        lat_str = f"{latitude}"
        lon_str = f"{longitude}"
        points_url = f"{NWS_API_BASE}/points/{lat_str},{lon_str}"
        points_data = await make_nws_request(client, points_url)

        if not points_data:
            return [types.TextContent(type="text", text=f"Failed to retrieve

        # Extract forecast URL from the response
        properties = points_data.get("properties", {})
        forecast_url = properties.get("forecast")

        if not forecast_url:
            return [types.TextContent(type="text", text="Failed to get foreca

        # Get the forecast
        forecast_data = await make_nws_request(client, forecast_url)

        if not forecast_data:
            return [types.TextContent(type="text", text="Failed to retrieve f

        # Format the forecast periods
        periods = forecast_data.get("properties", {}).get("periods", [])
        if not periods:
            return [types.TextContent(type="text", text="No forecast periods

        # Format each period into a concise string
        formatted_forecast = []
        for period in periods:
            forecast_text = (
                f"{period.get('name', 'Unknown')}:\n"
                f"Temperature: {period.get('temperature', 'Unknown')}°{period
                f"Wind: {period.get('windSpeed', 'Unknown')} {period.get('win
                f"{period.get('shortForecast', 'No forecast available')}\n"
                "---"
            )
            formatted_forecast.append(forecast_text)
```

```
            forecast_text = f"Forecast for {latitude}, {longitude}:\n\n" +  \n".j
            return [types.TextContent(
                type="text",
                text=forecast_text
            )]
        else:
            raise ValueError(f"Unknown tool: {name}")
```

Running the server

Finally, implement the main function to run the server:

```
async def main():
    # Run the server using stdin/stdout streams
    async with mcp.server.stdio.stdio_server() as (read_stream, write_stream):
        await server.run(
            read_stream,
            write_stream,
            InitializationOptions(
                server_name="weather",
                server_version="0.1.0",
                capabilities=server.get_capabilities(
                    notification_options=NotificationOptions(),
                    experimental_capabilities={},
                ),
            ),
        )

# This is needed if you'd like to connect to a custom client
if __name__ == "__main__":
    asyncio.run(main())
```

Your server is complete! Run `uv run src/weather/server.py` to confirm that everything's working.

Let's now test your server from an existing MCP host, Claude for Desktop.

Testing your server with Claude for Desktop

> Claude for Desktop is not yet available on Linux. Linux users can proceed to the **Building a client** tutorial to build an MCP client that connects to the server we just built.

First, make sure you have Claude for Desktop installed. **You can install the latest version here.** If you already have Claude for Desktop, **make sure it's updated to the latest version.**

We'll need to configure Claude for Desktop for whichever MCP servers you want to use. To do this, open your Claude for Desktop App configuration at `~/Library/Application Support/Claude/claude_desktop_config.json` in a text editor. Make sure to create the file if it doesn't exist.

For example, if you have **VS Code** installed:

MacOS/Linux **Windows**

```
code ~/Library/Application\ Support/Claude/claude_desktop_config.json
```

You'll then add your servers in the `mcpServers` key. The MCP UI elements will only show up in Claude for Desktop if at least one server is properly configured.

In this case, we'll add our single weather server like so:

MacOS/Linux **Windows**

Python

```
{
    "mcpServers": {
        "weather": {
```

```
            "command": "uv",
            "args": [
                "--directory",
                "/ABSOLUTE/PATH/TO/PARENT/FOLDER/weather",
                "run",
                "weather"
            ]
        }
    }
}
```

Make sure you pass in the absolute path to your server.

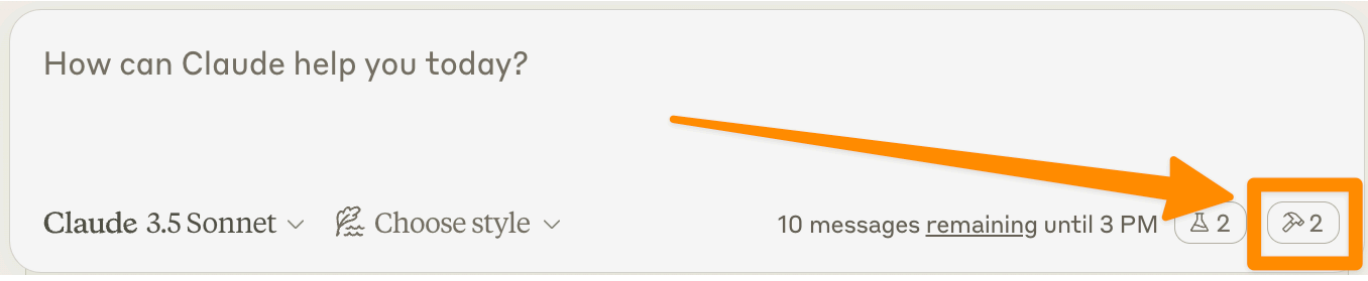
This tells Claude for Desktop:

1. There's an MCP server named "weather"
2. To launch it by running `uv --directory /ABSOLUTE/PATH/TO/PARENT/FOLDER/weather run weather`

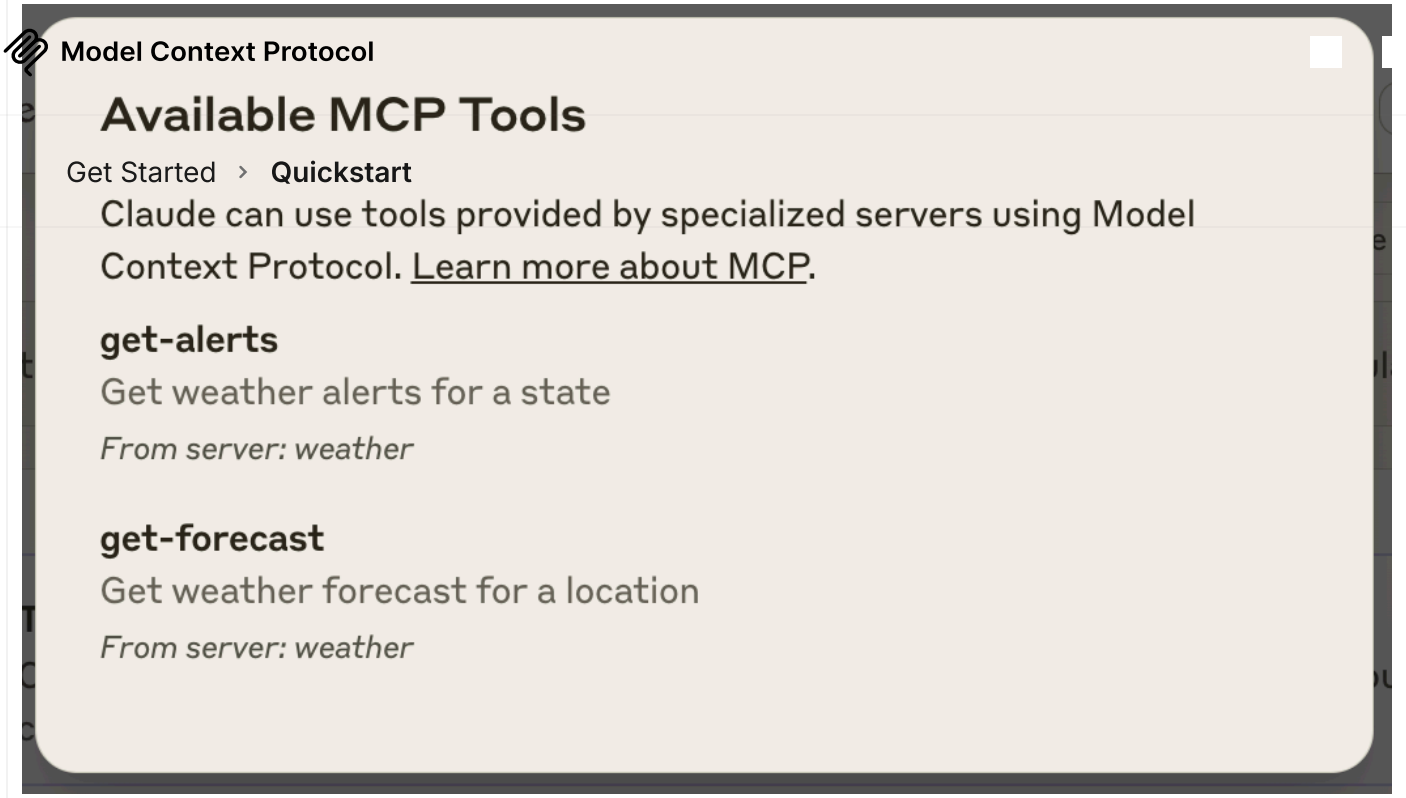
Save the file, and restart **Claude for Desktop**.

Test with commands

Let's make sure Claude for Desktop is picking up the two tools we've exposed in our `weather` server. You can do this by looking for the hammer icon:



After clicking on the hammer icon, you should see two tools listed:



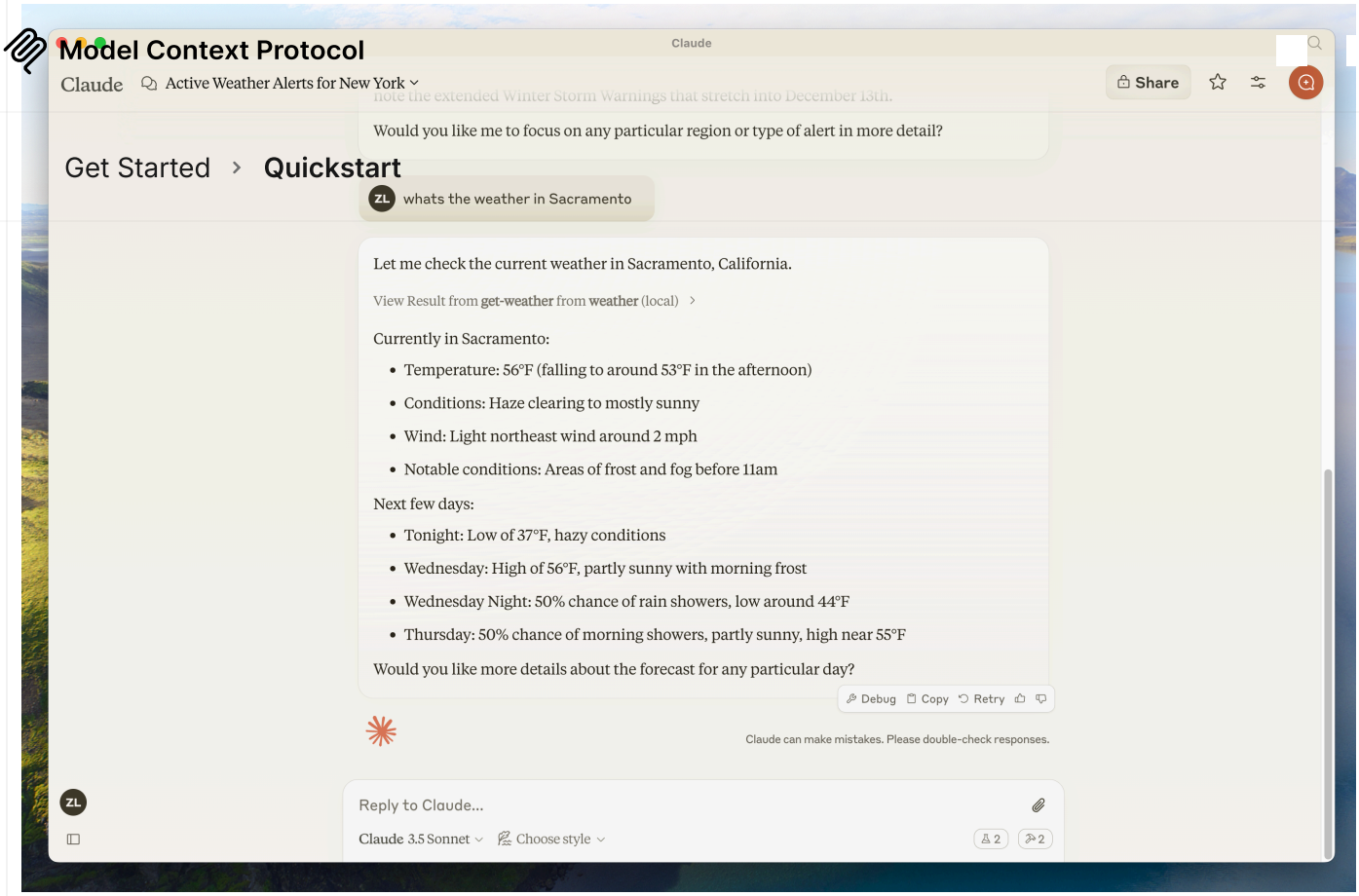
If your server isn't being picked up by Claude for Desktop, proceed to the **Troubleshooting** section for debugging tips.

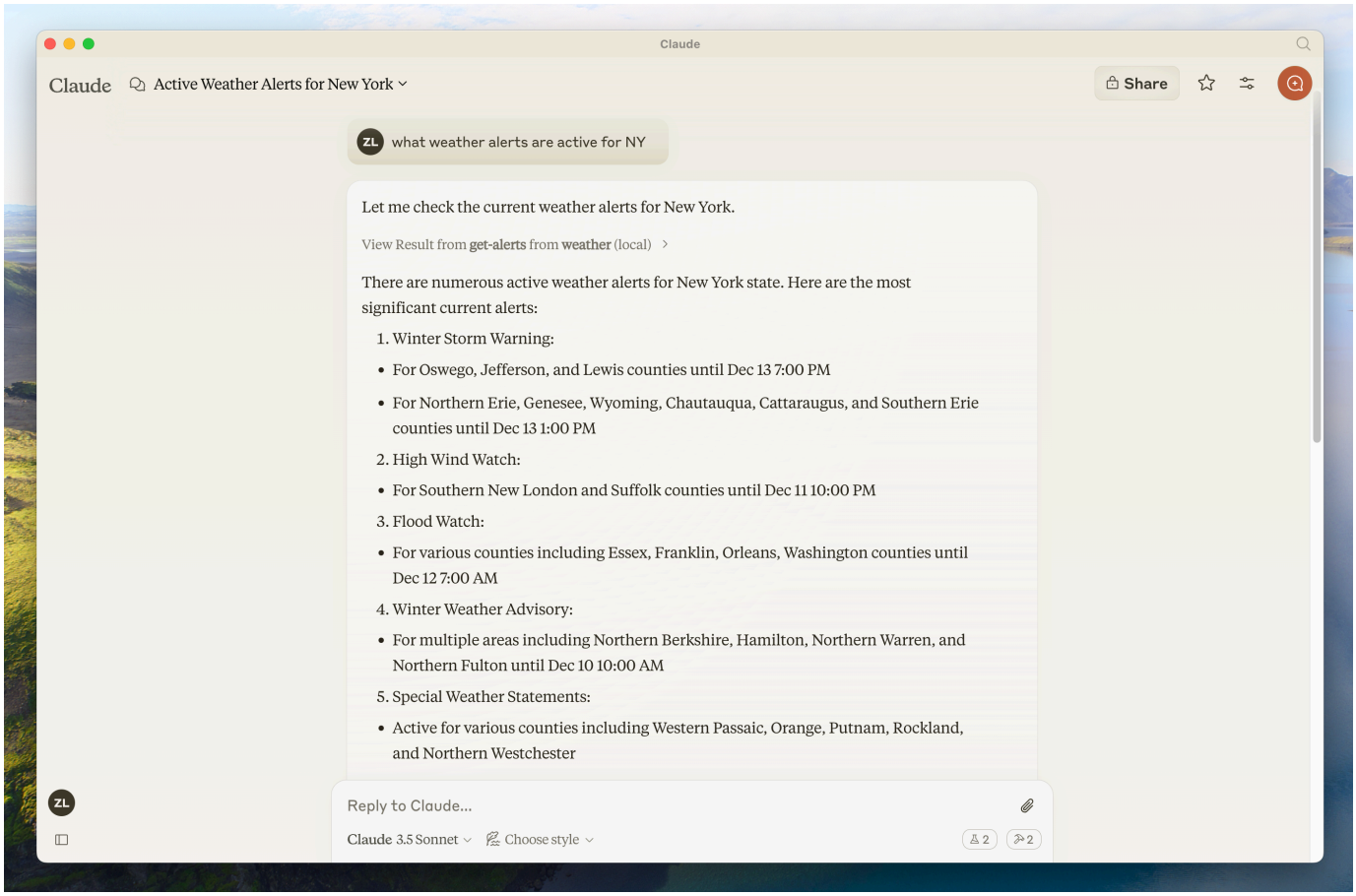
If the hammer icon has shown up, you can now test your server by running the following commands in Claude for Desktop:

> What's the weather in Sacramento?
>
> What are the active weather alerts in Texas?

Model Context Protocol

Get Started › Quickstart





Since this is the US National Weather service, the queries will only work for US locations.

What's happening under the hood

When you ask a question:

1. The client sends your question to Claude
2. Claude analyzes the available tools and decides which one(s) to use
3. The client executes the chosen tool(s) through the MCP server
4. The results are sent back to Claude
5. Claude formulates a natural language response
6. The response is displayed to you!

Troubleshooting

- Claude for Desktop Integration Issues
- Weather API Issues

For more advanced troubleshooting, check out our guide on **Debugging MCP**

Next steps

Building a client

Learn how to build your an MCP client that can connect to your server

Example servers

Check out our gallery of official MCP servers and implementations

Model Context Protocol

Get Started › Quickstart

Debugging Guide

Learn how to effectively debug MCP servers and integrations

Building MCP with LLMs

Learn how to use LLMs like Claude to speed up your MCP development

Was this page helpful? Yes No

‹ Introduction

Examples ›