Quickstart

# For Server Developers

Copy page

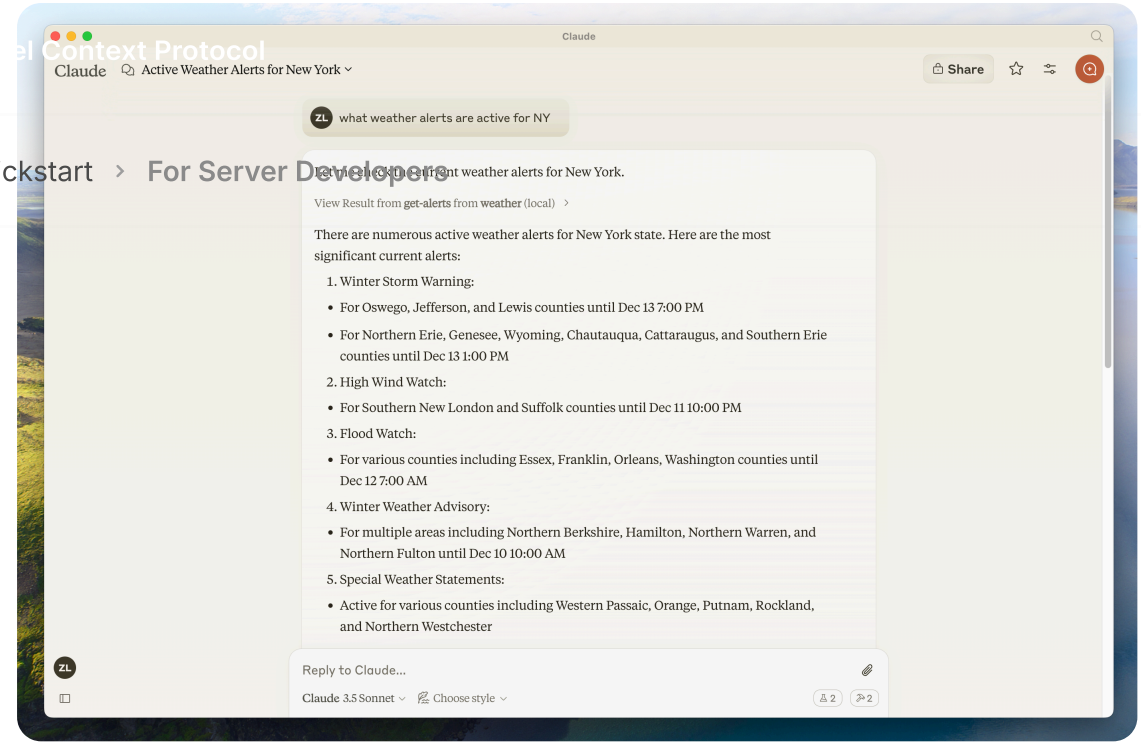
Get started building your own server to use in Claude for Desktop and other clients.

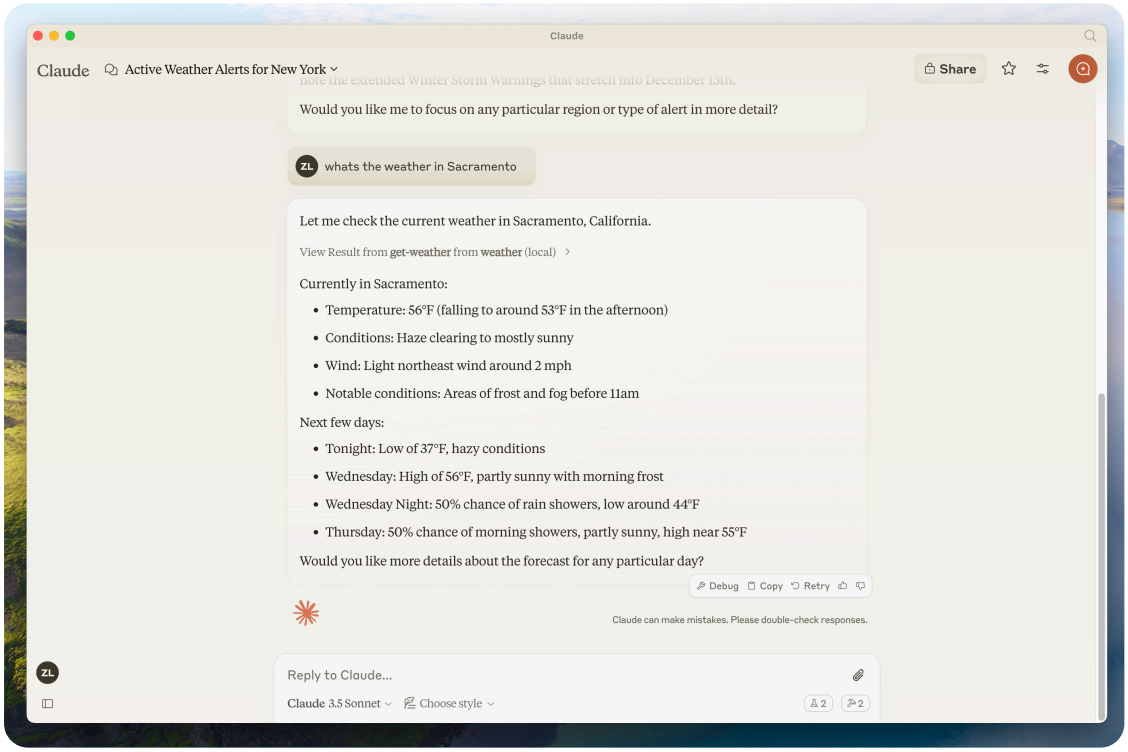
In this tutorial, we'll build a simple MCP weather server and connect it to a host, Claude for Desktop. We'll start with a basic setup, and then progress to more complex use cases.

## What we'll be building

Many LLMs 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):




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

### 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 Java Kotlin C#

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

Python 3.10 or higher installed.

You must use the Python MCP SDK 1.2.0 or higher.

## 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[cli]" httpx

# Create our server file
touch weather.py
```

Now let's dive into building your server.

## Building your server

### Importing packages and setting up the instance

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

```
from typing import Any
import httpx
from mcp.server.fastmcp import FastMCP

# Initialize FastMCP server
mcp = FastMCP("weather")

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

The FastMCP class uses Python type hints and docstrings to automatically generate tool definitions, making it easy to create and maintain MCP tools.



## 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(url: str) -> dict[str, Any] | None:
    """Make a request to the NWS API with proper error handling."""
    headers = {
        "User-Agent": USER_AGENT,
        "Accept": "application/geo+json"
    }
    async with httpx.AsyncClient() as client:
        try:
            response = await client.get(url, headers=headers, time
            response.raise_for_status()
            return response.json()
        except Exception:
            return None

def format_alert(feature: dict) -> str:
    """Format an alert feature into a readable string."""
    props = feature["properties"]
    return f"""
Event: {props.get('event', 'Unknown')}
Area: {props.get('areaDesc', 'Unknown')}
Severity: {props.get('severity', 'Unknown')}
Description: {props.get('description', 'No description available')
Instructions: {props.get('instruction', 'No specific instructions
"""
```

## Implementing tool execution

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

```python
@mcp.tool()
async def get_alerts(state: str) -> str:
    """Get weather alerts for a US state.

    Args:
        state: Two-letter US state code (e.g. CA, NY)
    """
    url = f"{NWS_API_BASE}/alerts/active/area/{state}"
    data = await make_nws_request(url)

    if not data or "features" not in data:
        return "Unable to fetch alerts or no alerts found."

    if not data["features"]:
        return "No active alerts for this state."

    alerts = [format_alert(feature) for feature in data["features"]
    return "\n---\n".join(alerts)

@mcp.tool()
async def get_forecast(latitude: float, longitude: float) -> str:
    """Get weather forecast for a location.

    Args:
        latitude: Latitude of the location
        longitude: Longitude of the location
    """
    # First get the forecast grid endpoint
    points_url = f"{NWS_API_BASE}/points/{latitude},{longitude}"
    points_data = await make_nws_request(points_url)

    if not points_data:
        return "Unable to fetch forecast data for this location."

    # Get the forecast URL from the points response
    forecast_url = points_data["properties"]["forecast"]
    forecast_data = await make_nws_request(forecast_url)

    if not forecast_data:
        return "Unable to fetch detailed forecast."

    # Format the periods into a readable forecast
```

```
        periods = forecast_data["properties"]["periods"]
        forecasts = []
        for period in periods[:5]:  # Only show next 5 periods
            forecast = f"""
{period['name']}:
Temperature: {period['temperature']}°{period['temperatureUnit']}
Wind: {period['windSpeed']} {period['windDirection']}
Forecast: {period['detailedForecast']}
"""
            forecasts.append(forecast)

        return "\n---\n".join(forecasts)
```

### Running the server

Finally, let's initialize and run the server:

```
if __name__ == "__main__":
    # Initialize and run the server
    mcp.run(transport='stdio')
```

Your server is complete! Run `uv run weather.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_conf ‌.j
```

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.py"
            ]
        }
    }
}
```

> ⚠ You may need to put the full path to the `uv` executable in the `command` field. You can get this by running `which uv` on MacOS/Linux or `where uv` on Windows.

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

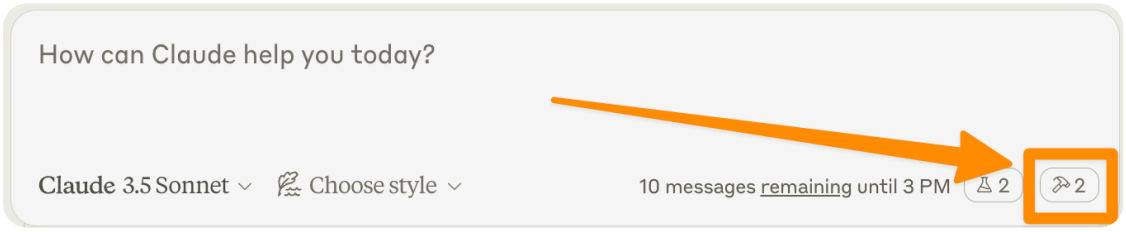
This tells Claude for Desktop:

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

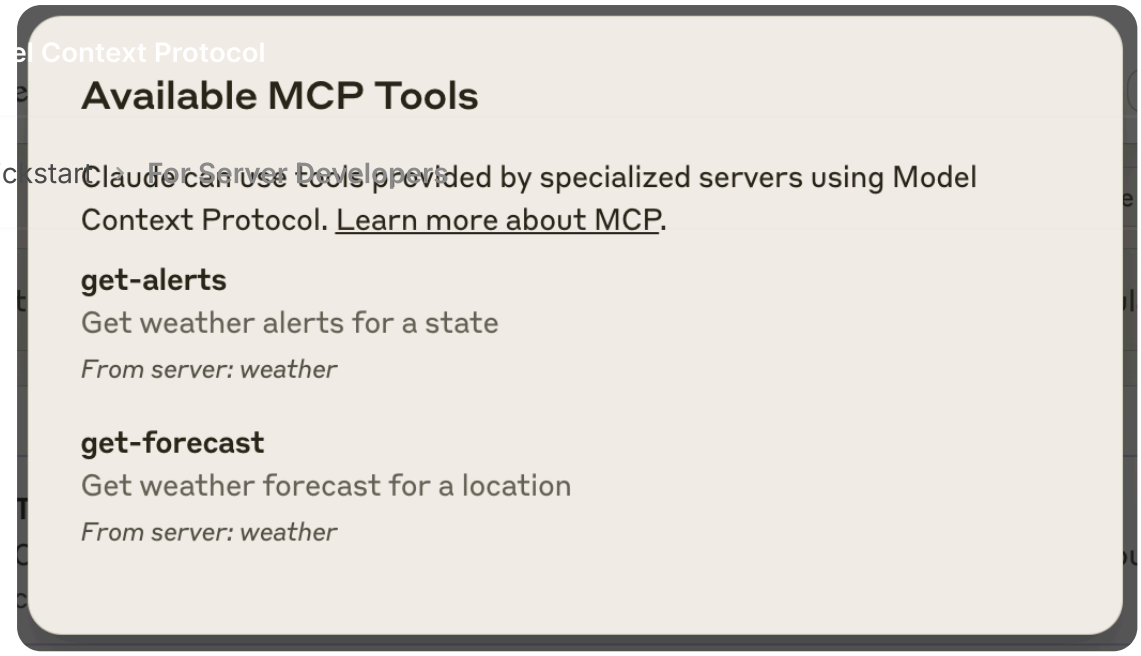
Save the file, and restart **Claude for Desktop**.

## 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:

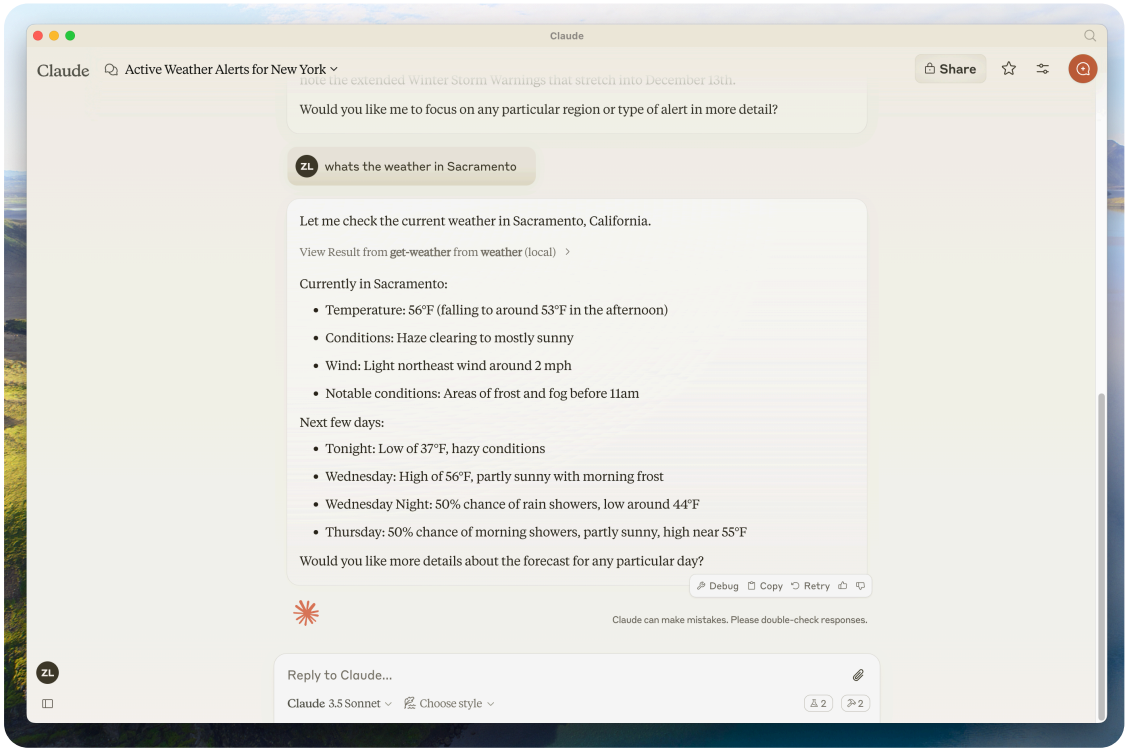


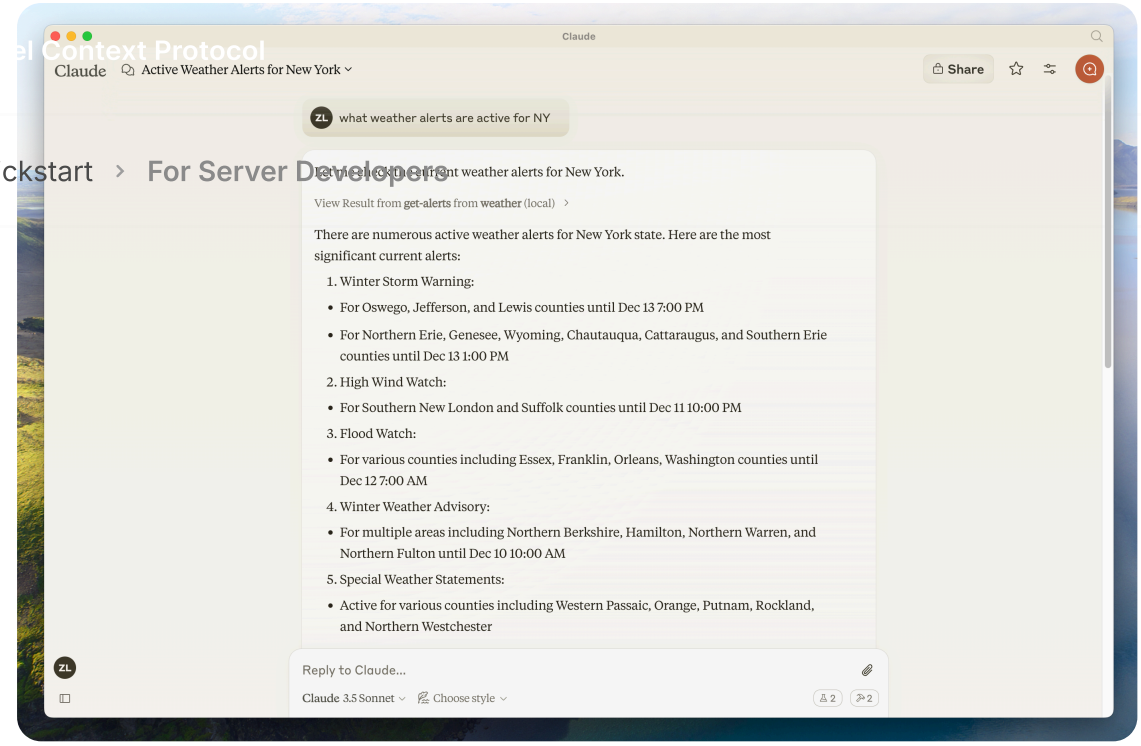

If your server isn't being picked up by Claude for Desktop, proceed to the Troubleshooting section for debugging tips.

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?






> 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 own MCP client that can connect to your server

### Example servers

Check out our gallery of official MCP servers and implementations

### 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