MCP Protocol

Model Context Protocol 2025

Complete Beginner Guide | Servers | Tools | Resources

■ What is MCP?

MCP (Model Context Protocol) = Open standard by Anthropic that connects AI models to external data sources and tools. Like a **USB-C for AI** - one protocol to connect everything!

Component	Description	Example
MCP Host	Application that uses AI (client)	Claude Desktop, VS Code, IDEs
MCP Client	Protocol connector in the host	Built into Claude Desktop
MCP Server	Service providing tools/resources	File system, GitHub, Database
Transport	Communication method	stdio, SSE, HTTP

Problem	MCP Solution
Every AI app builds custom integrations	One standard protocol for all
Tools locked to specific AI models	Tools work with any MCP-compatible AI
Complex setup for each connection	Simple JSON configuration
Security concerns with API access	Controlled, sandboxed access

■ Core Concepts (The 3 Primitives)

Primitive	Description	Control	Example
Tools	Functions the AI can call	Model-controlled	search_files(), run_query()
Resources	Data/content to read	App-controlled	file://docs/readme.md
Prompts	Pre-written prompt templates	User-controlled	summarize_code, explain_error

■ ***Tools = AI decides when to use | Resources = App loads into context | Prompts = User selects***

■■ Installation & Setup

Language	Package	Install Command
Python	mcp	pip install mcp
Python (with CLI)	mcp[cli]	pip install "mcp[cli]"
TypeScript	@modelcontextprotocol/sdk	npm install @modelcontextprotocol/sdk
TypeScript (global)	create-mcp-server	npx @modelcontextprotocol/create-server

■ Quick Start (Python)

```
# Install
pip install mcp

# Create a simple MCP server (my_server.py)
from mcp.server.fastmcp import FastMCP

# Create server instance
mcp = FastMCP('My First Server')

# Add a tool
@mcp.tool()
def greet(name: str) -> str:
'''Greet someone by name'''
return f'Hello, {name}!'

# Run the server
if __name__ == '__main__':
mcp.run()
```

■ Popular MCP Servers

Server	Purpose	Install/Config
filesystem	Read/write local files	npx -y @modelcontextprotocol/server-filesystem
github	GitHub repos, issues, PRs	npx -y @modelcontextprotocol/server-github
postgres	PostgreSQL database	npx -y @modelcontextprotocol/server-postgres
sqlite	SQLite database	npx -y @modelcontextprotocol/server-sqlite
brave-search	Web search via Brave	npx -y @modelcontextprotocol/server-brave-search
google-drive	Google Drive files	npx -y @anthropic/server-google-drive
slack	Slack messages/channels	npx -y @modelcontextprotocol/server-slack
memory	Persistent memory/notes	npx -y @modelcontextprotocol/server-memory
puppeteer	Browser automation	npx -y @anthropic/server-puppeteer
fetch	Fetch web pages	npx -y @anthropic/server-fetch
git	Git operations	npx -y @modelcontextprotocol/server-git
sequential-thinking	Step-by-step reasoning	npx -y @modelcontextprotocol/server-sequential-thinking

■■ Claude Desktop Configuration

■ ***Config file location: Mac: ~/Library/Application Support/Claude/claude_desktop_config.json | Windows: %APPDATA%/Claude/***

```
// claude_desktop_config.json
{
"mcpServers": {
"filesystem": {
"command": "npx",
"args": ["-y", "@modelcontextprotocol/server-filesystem", "/path/to/folder"]
},
"github": {
"command": "npx",
"args": ["-y", "@modelcontextprotocol/server-github"],
"env": {
"GITHUB_TOKEN": "your-github-token"
}
},
"postgres": {
"command": "npx",
"args": ["-y", "@modelcontextprotocol/server-postgres"],
"env": {
"DATABASE_URL": "postgresql://user:pass@localhost/db"
}
},
"my-python-server": {
"command": "python",
"args": ["/path/to/my_server.py"]
}
}
}
```

■ Creating Tools (Python)

```
from mcp.server.fastmcp import FastMCP
from typing import Annotated

mcp = FastMCP('Tool Server')

# Simple tool
@mcp.tool()
def add(a: int, b: int) -> int:
'''Add two numbers together'''
return a + b

# Tool with detailed parameters
@mcp.tool()
def search_files(
query: Annotated[str, 'Search query'],
max_results: Annotated[int, 'Maximum results to return'] = 10
) -> list[str]:
'''Search for files matching the query'''
# Implementation here
return ['file1.txt', 'file2.txt']

# Async tool
@mcp.tool()
async def fetch_data(url: str) -> str:
'''Fetch data from a URL'''
import aiohttp
async with aiohttp.ClientSession() as session:
async with session.get(url) as response:
return await response.text()
```

■ Creating Resources

```
from mcp.server.fastmcp import FastMCP

mcp = FastMCP('Resource Server')

# Static resource
@mcp.resource('config://app')
def get_config() -> str:
'''Application configuration'''
return '{"version": "1.0", "debug": true}'

# Dynamic resource with parameters
@mcp.resource('file://{path}')
def read_file(path: str) -> str:
'''Read a file from the filesystem'''
with open(path, 'r') as f:
return f.read()

# Resource with MIME type
@mcp.resource('data://users', mime_type='application/json')
def get_users() -> str:
'''Get all users as JSON'''
import json
users = [{'name': 'Alice'}, {'name': 'Bob'}]
return json.dumps(users)

# Binary resource (images, files)
@mcp.resource('image://{name}', mime_type='image/png')
def get_image(name: str) -> bytes:
'''Get an image file'''
with open(f'images/{name}.png', 'rb') as f:
return f.read()
```

■ Creating Prompts

```
from mcp.server.fastmcp import FastMCP

mcp = FastMCP('Prompt Server')

# Simple prompt
@mcp.prompt()
def summarize() -> str:
'''Summarize the provided content'''
return 'Please summarize the following content concisely:'

# Prompt with arguments
@mcp.prompt()
def code_review(language: str, style: str = 'detailed') -> str:
'''Review code in a specific language'''
return f'''Please review the following {language} code.
Style: {style}
Focus on:
- Code quality
- Best practices
- Potential bugs
- Performance issues'''

# Multi-message prompt
@mcp.prompt()
def debug_error(error_message: str) -> list:
'''Help debug an error'''
return [
{'role': 'system', 'content': 'You are a debugging expert.'},
{'role': 'user', 'content': f'Help me debug this error: {error_message}'}
]
```

■ Transport Protocols

Transport	Description	Use Case
stdio	Standard input/output	Local servers, Claude Desktop (default)
SSE	Server-Sent Events over HTTP	Web applications, remote servers
HTTP	HTTP POST requests	REST API style, stateless
WebSocket	Bidirectional real-time	Real-time apps (coming soon)

```
# stdio transport (default for Claude Desktop)
mcp.run() # Uses stdio by default

# SSE transport (for web apps)
mcp.run(transport='sse', port=8000)

# In Claude Desktop config for SSE server:
# {
# "my-server": {
# "url": "http://localhost:8000/sse"
# }
# }
```

■ TypeScript MCP Server

```
// Install: npm install @modelcontextprotocol/sdk

import { Server } from '@modelcontextprotocol/sdk/server/index.js';
import { StdioServerTransport } from '@modelcontextprotocol/sdk/server/stdio.js';

const server = new Server({
name: 'my-typescript-server',
version: '1.0.0',
}, {
capabilities: {
tools: {},
resources: {},
},
});

// Register a tool
server.setRequestHandler('tools/list', async () => ({
tools: [{
name: 'greet',
description: 'Greet someone',
inputSchema: {
type: 'object',
properties: { name: { type: 'string' } },
required: ['name']
}
}]
}));

server.setRequestHandler('tools/call', async (request) => {
if (request.params.name === 'greet') {
return { content: [{ type: 'text', text: `Hello, ${request.params.arguments.name}!` }] };
}
});

// Start server
const transport = new StdioServerTransport();
await server.connect(transport);
```

■ Context & Lifespan

```
from mcp.server.fastmcp import FastMCP, Context
from contextlib import asynccontextmanager

# Lifespan for setup/cleanup (database connections, etc.)
@asynccontextmanager
async def lifespan(server: FastMCP):
# Setup: runs when server starts
print('Server starting...')
db = await create_db_connection()
try:
yield {'db': db} # Available in context
finally:
# Cleanup: runs when server stops
await db.close()
print('Server stopped')

mcp = FastMCP('Stateful Server', lifespan=lifespan)

# Access context in tools
@mcp.tool()
async def query_db(sql: str, ctx: Context) -> str:
'''Run a database query'''
db = ctx.request_context.lifespan_context['db']
result = await db.execute(sql)
return str(result)
```

■ Complete Server Example

```
# weather_server.py - Complete MCP Server Example
from mcp.server.fastmcp import FastMCP
from typing import Annotated
import json

mcp = FastMCP('Weather Server')

# Tool: Get weather
@mcp.tool()
def get_weather(
city: Annotated[str, 'City name'],
units: Annotated[str, 'celsius or fahrenheit'] = 'celsius'
) -> str:
'''Get current weather for a city'''
# Mock implementation
return f'Weather in {city}: 22 degrees {units}, Sunny'

# Resource: Weather API config
@mcp.resource('config://weather-api')
def get_api_config() -> str:
'''Weather API configuration'''
return json.dumps({'api_version': '2.0', 'max_requests': 100})

# Prompt: Weather report
@mcp.prompt()
def weather_report(city: str) -> str:
'''Generate a weather report prompt'''
return f'Generate a detailed weather report for {city}'

if __name__ == '__main__':
mcp.run()
```

■ Debugging & Testing

Method	Command/Tool	Purpose
MCP Inspector	npx @modelcontextprotocol/inspector	Visual debugging UI
CLI Testing	mcp dev my_server.py	Test server locally

Logs	Check Claude Desktop logs	Debug connection issues
Python Logging	import logging; logging.basicConfig()	Server-side debugging

```
# Test with MCP Inspector
npx @modelcontextprotocol/inspector python my_server.py

# Test with mcp dev (if using mcp[cli])
mcp dev my_server.py

# Add logging to your server
import logging
logging.basicConfig(level=logging.DEBUG)

# Claude Desktop logs location:
# Mac: ~/Library/Logs/Claude/
# Windows: %APPDATA%/Claude/logs/
```

■ Best Practices

Category	Best Practice
Docstrings	Always write clear docstrings - they help the AI understand your tools
Error Handling	Return helpful error messages, not just exceptions
Type Hints	Use type hints and Annotated for parameter descriptions
Naming	Use clear, descriptive names for tools and resources
Security	Never expose sensitive data in resources without auth
Async	Use async for I/O operations (network, files)
Testing	Test with MCP Inspector before connecting to Claude
Logging	Add logging for debugging production issues
Config	Store API keys in env variables, not in code
Resources	Use appropriate MIME types for non-text resources
Granularity	Keep tools focused - one tool, one purpose

■ Quick Reference

Task	Code
Create Server	mcp = FastMCP('Name')
Add Tool	@mcp.tool() def my_tool(): ...
Add Resource	@mcp.resource('uri://path') def my_resource(): ...
Add Prompt	@mcp.prompt() def my_prompt(): ...
Run Server	mcp.run()
Run SSE Server	mcp.run(transport='sse', port=8000)
Type Annotation	param: Annotated[str, 'Description']
Async Tool	@mcp.tool() async def my_async_tool(): ...

Test Server	npx @modelcontextprotocol/inspector python server.py

■ MCP Ecosystem

Platform	MCP Support	Notes
Claude Desktop	Full support	Native integration
Claude.ai	Limited (connectors)	Some built-in MCP servers
Cursor IDE	Full support	Configure in settings
Continue.dev	Full support	VS Code extension
Cline	Full support	VS Code extension
Zed	Full support	Built-in support
Sourcegraph	Full support	Cody integration

Created by Manoj | The AI Dude Tamil ■

YouTube: The AI Dude Tamil | Master AI, Automation & Prompt Engineering

■ Connect AI to everything with MCP!