

Lab: Working with MCP Connectors

Module: T2 - Agents, Skills & MCP

Duration: 25 minutes (+ optional extension)

Difficulty: Intermediate

Overview

In this lab, you will explore the Model Context Protocol (MCP) and learn how to connect Claude to external systems. You will review existing connectors, set up a GitHub integration, and understand how to build custom MCP servers.

Objectives

After completing this lab, you will be able to:

- Understand the MCP connector ecosystem and available integrations
- Configure and authenticate the GitHub MCP connector
- Use Claude to interact with GitHub repositories via MCP
- Understand the structure of a custom MCP server

Prerequisites

- Claude Code installed and authenticated (completed in Lab T1)
- GitHub account with at least one repository
- Basic understanding of APIs and authentication concepts
- (Optional) Python 3.8+ for the custom MCP server section

Scenario

Your organization wants to integrate Claude with your development tools to streamline workflows. You will set up a GitHub connector so developers can use natural language to interact with repositories, issues, and pull requests directly from Claude.

Part 1: Exploring Existing Connectors

Duration: 10 minutes

In this part, you will explore the MCP connector ecosystem and understand what integrations are available.

Task 1: View Available MCP Connectors

1. Open your terminal and navigate to any project directory.
2. List the available MCP connectors:

```
```bash
```

```
claude mcp list
```

```
```
```

Expected output: A list of available connectors such as:

- github - GitHub repository integration
- slack - Slack messaging
- notion - Notion workspace
- filesystem - Local file access
- (and others depending on your environment)

3. Get detailed information about a specific connector:

```
```bash
```

```
claude mcp info github
```

```
```
```

Expected output: Description, available tools, required permissions, and configuration options.

Task 2: Review Connector Documentation

4. Open your web browser and visit: [**modelcontextprotocol.io/servers**](https://modelcontextprotocol.io/servers)
5. Browse the available server implementations. Note the following categories:

- **Official connectors:** Maintained by Anthropic
- **Community connectors:** Contributed by the community
- **Reference implementations:** Examples for building your own

6. Click on the GitHub connector to review its capabilities:

- **Tools:** list_repos, get_repo, create_issue, list_pull_requests, etc.
- **Resources:** Repository contents, issue details, PR diffs
- **Authentication:** OAuth with GitHub

Task 3: Understand the Three MCP Primitives

Review the three building blocks of MCP:

| Primitive | Purpose | Example |
|-----------|---------|---------|
|-----------|---------|---------|

| | | |
|---------------|----------------------------|-------------------------------------|
| **Tools** | Actions Claude can execute | `create_issue`,
`send_message` |
| **Resources** | Data Claude can read | `repo://org/name`,
`file://path` |
| **Prompts** | Templates for interactions | `code_review`, `summarize` |

Discussion: Think about your organization's systems. Which ones would benefit from:

- Tool access (Claude can take action)?
- Resource access (Claude can read data)?
- Both?

Validation Checkpoint

- [] You can list available MCP connectors
 - [] You understand the three MCP primitives (Tools, Resources, Prompts)
 - [] You have identified potential use cases for MCP in your organization
-

Part 2: GitHub Connector Setup

Duration: 15 minutes

In this part, you will configure the GitHub MCP connector and use Claude to interact with your repositories.

***Note:** If MCP connectors are not yet enabled in your Claude Enterprise environment, follow along as a conceptual exercise or use demo mode.*

Task 1: Configure the GitHub Connector

7. Add the GitHub connector to your Claude configuration:

```
```bash
```

```
claude mcp add github
```

```
```
```

8. You will be prompted to configure the connector. Select the appropriate options:

- **Authentication method:** OAuth (recommended)
- **Scope:** Repository read access (minimum), or read/write if you want to create issues

9. A browser window will open for GitHub OAuth authentication.

Task 2: Authenticate with GitHub

10. In the browser, sign in to your GitHub account if prompted.
11. Review the permissions being requested:
 - **Repository access:** Read access to code and metadata
 - **Issues:** Read access to issues and pull requests
 - (Optionally) Write access for creating issues
12. Click **Authorize** to grant access.
13. Return to your terminal. You should see a confirmation message.

Expected output: "GitHub connector configured successfully"

***Troubleshooting:** If OAuth fails, verify:*

- Your GitHub account has access to the repositories you want to query
- Your organization allows OAuth apps (check with your GitHub admin)
- No VPN or firewall is blocking the OAuth callback

Task 3: Test the GitHub Connection

14. Start Claude Code:

```bash

claudie

```

15. Ask Claude to list your repositories:

```

*List my GitHub repositories*

```

Expected behavior: Claude will call the GitHub MCP connector and return a list of your accessible repositories.

16. Get information about a specific repository:

```

*Show me information about the [repository-name] repository*

```

Replace [repository-name] with one of your actual repositories.

17. Query issues in a repository:

```

*What are the open issues in [repository-name]?*

```

18. Summarize recent activity:

```

*Summarize the last 5 commits in [repository-name]*

```

Expected behavior: Claude will fetch commit data and provide a human-readable summary.

Task 4: Understand MCP in Action

When Claude answers these questions, observe what's happening:

19. **Tool Invocation:** Claude decides which MCP tool to use (e.g., `list_repos`, `get_issues`)
20. **API Call:** The MCP server makes the actual GitHub API call
21. **Response Processing:** Claude receives the data and formats it for you
22. **Natural Language:** You interact in plain English while Claude handles the technical details

[SCREENSHOT: Terminal showing Claude's response with GitHub data]

Validation Checkpoint

- [] GitHub connector is configured and authenticated
 - [] Claude can list your repositories
 - [] Claude can retrieve issues and commit information
 - [] You understand how MCP enables Claude to access external systems
-

Part 3: Custom MCP Server Walkthrough (Optional)

Duration: 15+ minutes

In this optional part, you will review a sample MCP server implementation to understand how to build custom integrations.

****Note:** This section is conceptual/educational. Building production MCP servers requires additional considerations.**

Task 1: Review the Sample MCP Server

Navigate to the sample MCP server directory:

```
cd labs/sample-mcp-server
```

Open `server.py` in your code editor and review the code:

```
from mcp.server import Server
from mcp.server.stdio import stdio_server

# Create the MCP server
server = Server("demo-server")

@server.tool()
def get_weather(city: str) -> str:
    """Get current weather for a city.

    Args:
        city: Name of the city to get weather for

    Returns:
        Weather information as a string
    """
    # In a real implementation, this would call a weather API
    return f"Weather in {city}: 72F, Sunny"

@server.tool()
def create_task(title: str, priority: str = "medium") -> dict:
    """Create a task in the task management system.

    Args:
        title: Title of the task
        priority: Priority level (low, medium, high)

    Returns:
        Dictionary with task details
    """
    # In a real implementation, this would call your task API
    return {
        "id": "task-123",
        "title": title,
        "priority": priority,
        "status": "created"
    }

@server.resource("employee://{{employee_id}}")
```

```

def get_employee(employee_id: str) -> str:
    """Fetch employee information as a resource.

    Args:
        employee_id: The employee's ID

    Returns:
        JSON string with employee data
    """

    # In a real implementation, this would query your HR system
    import json
    return json.dumps({
        "id": employee_id,
        "name": "John Doe",
        "department": "Engineering",
        "email": "john.doe@company.com"
    })

async def main():
    """Run the MCP server."""
    async with stdio_server() as streams:
        await server.run(*streams)

if __name__ == "__main__":
    import asyncio
    asyncio.run(main())

```

Task 2: Understand the Key Components

Review each component of the MCP server:

Component	Purpose	Example
`Server("name")`	Creates the MCP server instance	`Server("demo-server")`
`@server.tool()`	Exposes a function as a callable tool	Claude can invoke `get_weather`
`@server.resource()`	Exposes data as a readable resource	Claude can read `employee://123`
`stdio_server()`	Handles communication with Claude	Standard input/output protocol

Task 3: Review the Dependencies

Open requirements.txt:

```
mcp>=0.1.0
httpx>=0.24.0
```

The `mcp` package provides:

- Server framework
- Protocol handling
- Tool and resource decorators

The `httpx` package is useful for making HTTP requests to external APIs.

Task 4: Run the Server (Optional)

If you have Python 3.8+ installed:

23. Create a virtual environment:

```
```bash
python -m venv venv
source venv/bin/activate # On Windows: venv\Scripts\activate
````
```

24. Install dependencies:

```
```bash
pip install -r requirements.txt
````
```

25. Run the server:

```
```bash
python server.py
````
```

***Note:** In production, the MCP server would be registered with Claude and invoked automatically. This standalone run is for testing purposes.*

Task 5: Design Your Own Integration

Think about an internal system you'd like to connect to Claude. Answer these questions:

26. **What tools would you expose?**

- Example: `search_tickets`, `create_incident`, `get_customer_info`

27. **What resources would you provide?**

- Example: `ticket://12345`, `customer://acme-corp`

28. **What permissions are needed?**

- Example: Read-only for queries, write for creation

29. **What security considerations apply?**

- Example: OAuth authentication, rate limiting, audit logging

Validation Checkpoint

- [] You understand the structure of an MCP server
 - [] You can identify the purpose of tools vs. resources
 - [] You have ideas for custom integrations in your organization
-

Discussion Questions

Use these questions for group discussion or self-reflection:

30. **Internal Systems:** What internal systems in your organization would benefit from MCP connectors? Consider:

- Ticketing systems (Jira, ServiceNow)
- Documentation platforms (Confluence, SharePoint)
- Communication tools (Slack, Teams)
- Custom internal applications

31. **Permission Boundaries:** For each system you identified:

- What permissions would you grant? (Read-only? Read-write?)
- What data should Claude NOT have access to?
- Who should be able to configure these integrations?

32. **Sensitive Data:** How would you handle sensitive data in MCP responses?

- Should certain fields be masked or redacted?
- What audit logging would you implement?
- How would you handle PII or confidential information?

33. **Security Considerations:** Review the threat model:

- Prompt injection risks
 - Over-permissioning dangers
 - Data exfiltration concerns
 - How would you mitigate each?
-

Troubleshooting Guide

| Issue | Solution |
|-----------------------------|--|
| "Connector not found" | Run `claude mcp list` to see available connectors. The connector may not be enabled in your environment. |
| OAuth authentication fails | Verify GitHub credentials. Check if your organization allows OAuth apps. Contact your GitHub admin. |
| "Permission denied" for MCP | Check Claude Enterprise admin console for MCP enablement. Contact your Claude admin. |
| Slow responses from MCP | MCP calls add network latency - this is expected. First calls may be slower due to authentication. |
| "Rate limit exceeded" | GitHub has API rate limits. Wait a few minutes and try again, or reduce query frequency. |
| Server won't start (Python) | Ensure Python 3.8+. Check virtual environment is activated. Verify all dependencies are installed. |

Summary

In this lab, you have learned how to:

34. **Explore the MCP ecosystem** and understand available connectors
35. **Configure and authenticate** the GitHub MCP connector
36. **Use natural language** to query GitHub repositories, issues, and commits
37. **Understand MCP server architecture** and how to build custom integrations

Key Takeaways

- **MCP standardizes** how Claude connects to external systems
- **Three primitives:** Tools (actions), Resources (data), Prompts (templates)
- **Security first:** Always apply least-privilege access and audit logging
- **Ecosystem growing:** Check modelcontextprotocol.io for new connectors

Next Steps

- Experiment with other available MCP connectors in your environment
- Propose custom MCP integrations for your team's internal systems
- Review the MCP security guidelines before production deployment
- Explore the MCP Python and TypeScript SDKs for custom development

Additional Resources

- MCP Specification: modelcontextprotocol.io
- MCP Servers Repository: github.com/modelcontextprotocol/servers
- MCP Python SDK: mcp-python.readthedocs.io
- Claude Enterprise Admin: admin.claude.ai
- Anthropic Documentation: docs.anthropic.com