Lab 4: Use Model Context Protocol (MCP) as Tools.

Tools for extending agent capabilities

- Strands Agents build-in Tools

```
# Add tools to our agent
agent = Agent(
    tools=[calculator, file_read, shell]
)
```

```
print(agent.tool_names)

print(agent.tool_config)
```

```
agent = Agent(tools=["/path/to/my_tool.py"])
```

Tools for extending agent capabilities

- Strands Agents build-in Tools
- Python Tools

```python
# Define a function to convert Fahrenheit to Celsius
@tool
def fahrenheit_to_celsius(params: Dict[str, Any]) -> Dict[str, Any]:
    """
    Convert temperature from Fahrenheit to Celsius.

    Args:
        params: A dictionary containing:
            - fahrenheit: The temperature in Fahrenheit to convert

    Returns:
        A dictionary containing:
            - celsius: The temperature in Celsius
            - original_fahrenheit: The original Fahrenheit value
    """
    try:
        fahrenheit = float(params.get("fahrenheit"))
        celsius = (fahrenheit - 32) * 5/9
        return {
            "celsius": round(celsius, 2),
            "original_fahrenheit": fahrenheit
        }
    except (TypeError, ValueError):
        return {
            "error": "Invalid input. Please provide a valid number for fahrenheit."
        }
```

Tools for extending agent capabilities

- Strands Agents build-in Tools
- Python Tools
- Model Context Protocol (MCP) Tools

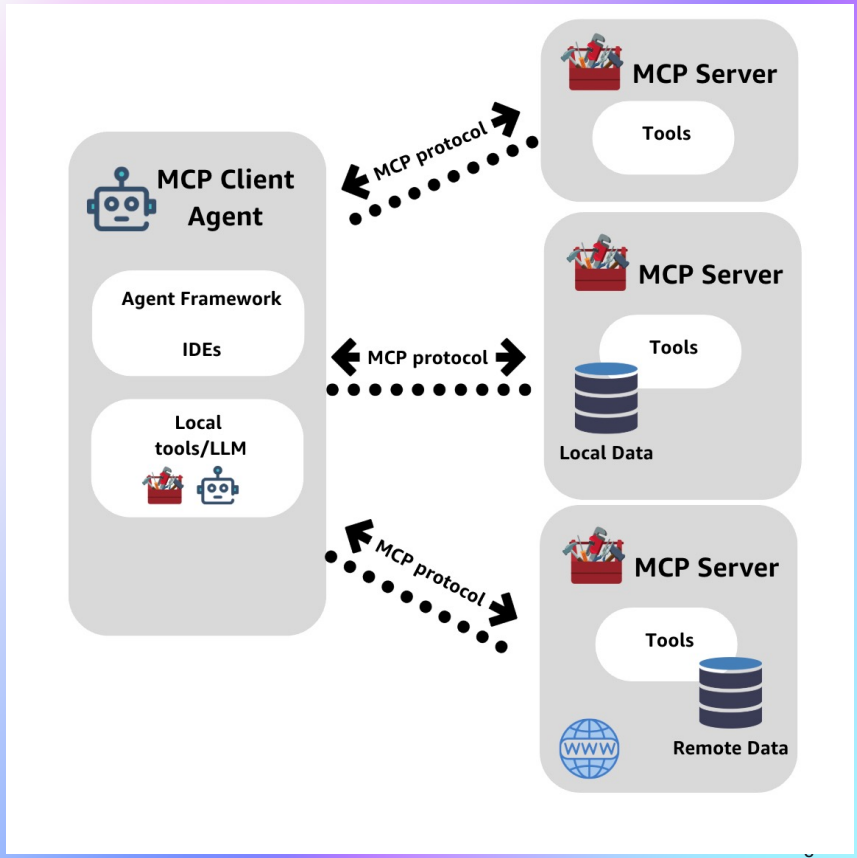
```python
from mcp.server import FastMCP

mcp = FastMCP("Calculator Server")

@mcp.tool(description="Add two numbers together")
def add(x: int, y: int) -> int:
"""Add two numbers and return the result."""
    return x + y
```

MCP Architecture Components

- **MCP Client:** Integrated in Strands Agent Framework
- **MCP Server:** Independent service hosting tools
- **MCP Tools:** Specific functionalities (calculator, weather, etc.)



MCP Server Connection Options

MCP Server Connection Options

- Standard I/O (stdio)

```python
from mcp import stdio_client, StdioServerParameters
from strands import Agent
from strands.tools.mcp import MCPClient

# Connect to an MCP server using stdio transport
# Note: uvx command syntax differs by platform

# For macOS/Linux:
stdio_mcp_client = MCPClient(lambda: stdio_client(
    StdioServerParameters(
        command="uvx",
        args=["awslabs.aws-documentation-mcp-server@latest"]
    )
))

# For Windows:
stdio_mcp_client = MCPClient(lambda: stdio_client(
    StdioServerParameters(
        command="uvx",
        args=[
            "--from",
            "awslabs.aws-documentation-mcp-server@latest",
            "awslabs.aws-documentation-mcp-server.exe"
        ]
    )
))

# Create an agent with MCP tools
with stdio_mcp_client:
    # Get the tools from the MCP server
    tools = stdio_mcp_client.list_tools_sync()

    # Create an agent with these tools
    agent = Agent(tools=tools)
    agent("What is AWS Lambda?")
```

MCP Server Connection Options

MCP Server Connection Options

- Standard I/O (stdio)
- Streamable HTTP

```python
from mcp.client.streamable_http import streamablehttp_client
from strands import Agent
from strands.tools.mcp.mcp_client import MCPClient

streamable_http_mcp_client = MCPClient(lambda: streamablehttp_client("http://localhost:8000/mcp"))

# Create an agent with MCP tools
with streamable_http_mcp_client:
    # Get the tools from the MCP server
    tools = streamable_http_mcp_client.list_tools_sync()

    # Create an agent with these tools
    agent = Agent(tools=tools)
```

MCP Server Connection Options

MCP Server Connection Options

- Standard I/O (stdio)
- Streamable HTTP
- Server-Sent Events (SSE)

```python
from mcp.client.sse import sse_client
from strands import Agent
from strands.tools.mcp import MCPClient

# Connect to an MCP server using SSE transport
sse_mcp_client = MCPClient(lambda: sse_client("http://localhost:8000/sse"))

# Create an agent with MCP tools
with sse_mcp_client:
    # Get the tools from the MCP server
    tools = sse_mcp_client.list_tools_sync()

    # Create an agent with these tools
    agent = Agent(tools=tools)
```

MCP Server Connection Options

MCP Server Connection Options

- Standard I/O (stdio)
- Streamable HTTP
- Server-Sent Events (SSE)
- Custom Transport with MCPClient

```python
from typing import Callable
from strands import Agent
from strands.tools.mcp.mcp_client import MCPClient
from strands.tools.mcp.mcp_types import MCPTransport

# Define a function that returns your custom transport
def custom_transport_factory() -> MCPTransport:
    # Implement your custom transport mechanism
    # Must return a tuple of (read_stream, write_stream)
    # Both must implement the AsyncIterable and AsyncIterator protocols
    ...
    return read_stream, write_stream

# Create an MCPClient with your custom transport
custom_mcp_client = MCPClient(transport_callable=custom_transport_factory)

# Use the server with context manager
with custom_mcp_client:
    # Get the tools from the MCP server
    tools = custom_mcp_client.list_tools_sync()

    # Create an agent with these tools
    agent = Agent(tools=tools)
```

Tool Design Best Practices

- Clearly explain the tool's purpose and functionality
- Specify when the tool should be used
- Detail the parameters it accepts and their formats
- Describe the expected output format
- Note any limitations or constraints

Thank You