Hugging Face MCP Course - Unit 1 Summary (Expanded)

Hugging Face MCP Course - Unit 1 Summary (Expanded)

Title: Configuring MCP Clients and Servers

Unit 1 of the Hugging Face MCP course focuses on how to configure Model Context Protocol (MCP) servers and clients. MCP enables structured, tool-based modular AI systems, and configuration is central to setting up its environment.

Key Topics Covered:

1. MCP Configuration Overview:
   - MCP clients and servers use configuration files to manage connection logic.
   - These files specify which tools are available and how to reach them.

2. mcp.json File:
   - The core config file used to register servers and define transports.
   - Each server has a name and a transport method (e.g., stdio, sse).
   - Example:

```
    {
      "servers": [
        {
          "name": "Weather Service",
          "transport": {
            "type": "stdio",
            "command": "python3 server.py"
          }
        }
      ]
    }
```

3. Transport Types:

- stdio: Launches the server as a subprocess using stdin/stdout.
  - Best for short-lived or isolated tools.
- sse: Server-Sent Events, connects to a long-running HTTP-based server.
  - Best for persistent services.
- Custom transports can be added using plugins.

4. Starting an MCP Server:

- A server is typically created using FastMCP:

```
from mcp.server.fastmcp import FastMCP

mcp = FastMCP("Weather Service")

@mcp.tool()

def get_weather(location: str) -> str:

    return f"Weather in {location}: Sunny"

if __name__ == "__main__":

    mcp.run()
```

5. Connecting Clients:

- Clients use ToolCollection and MCPClient to load available tools.
- Example:

```
from mcp import ToolCollection
```

```
tools = ToolCollection.from_config("mcp.json")
print(tools["weather"].get_weather(location="Dhaka"))
```

6. Debugging and Validation:

- Validate JSON syntax before using it.
- Use dry runs to test connectivity.
- Print loaded tools to verify connection.

7. Environment Customization:

- Use multiple mcp.json files for staging, dev, or production.
- You can reference environment variables in your Python script but not inside mcp.json.

8. Tool Metadata and Aliases:

- ToolCollection supports discovery:
  - tools.list_tools()
  - tools["name"].description
- Enables dynamic workflows with AI agents and chaining.

9. Best Practices:

- Modularize server definitions.
- Keep each service in a separate repo or module if scaling.
- Use consistent naming across tools and configs.

Conclusion:

Unit 1 establishes the foundation for running modular AI applications using MCP. Configuration files such as mcp.json decouple tool logic from infrastructure, enabling interoperability, reuse, and clarity in multi-agent systems.