## **Path 3: MCP Client Builder - Detailed Journey**

### **Step 1: Environment Access**

**What the AI Engineer Does:**

* Logs into **SageMaker Domain Studio**
* Opens either:
  + **Jupyter Hub** (notebook interface), OR
  + **VSCode extension** integrated with SageMaker Domain Studio

**What They Get:**

* Pre-configured environment with SageMaker instance
* Connectivity to enterprise artifactory for downloading approved libraries
* Network connectivity to Line-of-Business (LOB) APIs from builder account
* Access to Bedrock models via application inference profiles tied to their SageMaker role

### **Step 2: MCP Framework Setup**

**What the AI Engineer Does:**

* Downloads MCP SDK and related libraries from enterprise artifactory
* Reviews LOB API documentation and connectivity guide
* Identifies which LOB APIs they need to integrate with

**Assumptions:**

* Network connectivity from builder account to LOB systems is already established
* LOB API endpoints, authentication, and access are pre-configured

### **Step 3: Build MCP Client**

**What the AI Engineer Does:**

* Writes MCP client code to call LOB APIs
* Implements:
  + MCP server/client protocol handlers
  + LOB API integration logic
  + Request/response transformation
  + Error handling and retry logic
  + Authentication flows (if needed) STRECH GOAL

**What They're Building:**

* Bridge between MCP protocol and LOB API calls
* Custom tooling for Claude or other LLM agents to access LOB data
* Integration layer for enterprise systems

### **Step 4: Local Testing Within SageMaker**

**What the AI Engineer Does:**

* Tests MCP client **locally within SageMaker instance**
* Validates:
  + LOB API connectivity
  + MCP protocol compliance
  + Request/response accuracy
  + Error handling behavior
* Iterates on implementation until satisfied

**What This Achieves:**

* Proves the MCP client works in Builder with No CI/CD
* Verifies LOB API integration
* No deployment needed for basic testing and validation

### **Step 5: Deployment Scenarios (Stretch Goal)**

**When This Is Needed:**

* Engineer needs MCP client running as a persistent service
* Wants it accessible to other teams or applications
* Needs scheduled execution or event-driven invocation

**Deployment Option 1: Lambda (MCP-Specific)** **What the AI Engineer Does:**

* Runs **Lambda deployment utility script**

**What the Script Does:**

* Packages MCP client code as Lambda function
* Configures Lambda execution role and environment
* Sets up API Gateway or EventBridge triggers (if needed)
* Provides Lambda function ARN and invocation details

**Result:**

* MCP client runs serverless on demand
* Cost-effective for sporadic or event-driven workloads
* Scales automatically

**Deployment Option 2: ECS Service** **What the AI Engineer Does:**

* Runs **ECS utility script** for ECS deployment

**What the Script Does:**

* Packages MCP client code into Docker container
* Pushes container to ECR (Elastic Container Registry)
* Deploys the container as an ECS service
* Provides service endpoint

**Result:**

* MCP client runs as always-on service
* Better for high-frequency or continuous workloads
* Can be scaled and load-balanced

## **Our Team's Deliverables for Path 3**

**Core (Must Have by October 31st):**

* LOB API connectivity guide and network configuration documentation
* MCP SDK setup and integration examples
* Sample MCP client templates
* Local testing examples with LOB API mocks

**Stretch Goals:**

* Lambda deployment utility script (MCP-specific)
* ECS deployment utility script
* Deployment decision guide (Lambda vs ECS)