## **Path 2: Agent 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
* Access to Bedrock models via application inference profiles tied to their SageMaker role
* Access to already-provisioned OpenSearch (if using RAG-based agent) or can provision new one

### **Step 2: Agent Framework Setup**

**What the AI Engineer Does:**

* Chooses their agent framework from enterprise artifactory
  + **LangChain** (example - supports both RAG and ReAct agents)
  + **LlamaIndex**
  + Other approved agent frameworks

**Agent Types They Can Build:**

1. **RAG-based Agent**: Uses OpenSearch for knowledge retrieval + reasoning
2. **ReAct Agent**: Uses tool calling and iterative reasoning without vector storage

### **Step 3: Infrastructure Connection**

**What the AI Engineer Does:**

* If building RAG-based agent, connects to OpenSearch:
  + Uses existing OpenSearch from Path 1, OR
  + Runs **OpenSearch serverless provisioning utility** to create new instance
* Runs **DynamoDB provisioning utility script** for conversation memory management
* Connects to Bedrock models via inference profiles

**What They're Connecting:**

* OpenSearch (vector store for knowledge base)
* DynamoDB (conversation memory persistence)
* Bedrock models (reasoning engine)
* Optional: External tools/APIs for ReAct pattern

### **Step 4: Build Agent Application**

**What the AI Engineer Does:**

* Writes agent logic in their chosen framework
* Implements:
  + Agent reasoning loops
  + Tool definitions (for ReAct agents)
  + RAG retrieval logic (for RAG agents)
  + Conversation memory management (DynamoDB integration)
  + Error handling and retry logic

### **Step 5: Local Testing Within SageMaker**

**What the AI Engineer Does:**

* Tests agent **locally within SageMaker instance** using simple client code
* Instruments application with **Arize AI** for observability:
  + Runs **Arize AI instrumentation script** (uses admin API key)
  + Script automatically creates project in Arize AI on the fly
  + Registers application in Arize AI for monitoring
* Validates:
  + Agent reasoning quality
  + Tool execution accuracy
  + Response coherence
  + Performance and latency
* Iterates on agent logic until satisfied

**What This Achieves:**

* Proves the agent works in Builder with No CI/CD
* Full observability and monitoring through Arize AI
* No deployment needed for basic testing and validation

### **Step 6: Deployment Scenarios (Stretch Goal)**

**When This Is Needed:**

* Engineer needs the agent as a **Streamlit chat interface** (interactive UI)
* Needs to expose agent as a **FastAPI REST API** (programmatic access)
* Wants it running **outside** their SageMaker session

**What the AI Engineer Does:**

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

**What the Script Does:**

* Packages agent code (with Streamlit or FastAPI wrapper) into Docker container
* Pushes container to ECR (Elastic Container Registry)
* Deploys the container as an ECS service
* Provides:
  + Web UI endpoint (for Streamlit deployments)
  + API endpoint (for FastAPI deployments)

**Result:**

* Agent runs as production service accessible to other users/systems
* Can be scaled, monitored, and maintained
* No longer dependent on their SageMaker session being active

## **Our Team's Deliverables for Path 2**

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

* Agent connection guide (OpenSearch + Bedrock + DynamoDB configuration)
* DynamoDB provisioning utility script
* Sample agent templates (RAG-based and ReAct patterns)
* Local testing examples with simple client code
* Arize AI instrumentation script and guide
* Documentation on agent framework options and selection