3.0 AGENTIC AI & ORCHESTRATION OVERVIEW

LEARNING OBJECTIVES

By the end of Day 3 (morning theory block), you should be able to:

- Explain what **Agentic AI** is and how it differs from “single-shot” LLM calls.
- Describe common **agent patterns**: tool calling, ReAct, plan–act, multi-agent and graph-based flows.
- Map these patterns onto concrete frameworks:
  - **CrewAI** – multi-agent orchestration in Python.
  - **Langflow** – visual builder for LangChain-style flows.
  - **LangGraph** – graph-based orchestration for complex workflows.
  - **watsonx Orchestrate** – IBM’s production-grade agent platform.
- Understand how your **RAG accelerator** turns into a reusable tool for agents.

3.1 WHAT IS AGENTIC AI?

So far we mostly used LLMs like functions:

Prompt in → answer out

Agentic AI adds **reasoning + action**:

- The LLM doesn't just answer — it:
  - Decides **which tools to use** (APIs, RAG services, calculators…).
  - Plans a sequence of **steps**.
  - Executes tools, reads results, and continues reasoning.
- The result is an **agent**:
  - Has a goal ("help user with workshop questions").
  - Has **capabilities** (tools, collaborators, knowledge bases).
  - Uses an LLM for planning and reflection.

Agentic AI is powerful when:

- You need **multi-step workflows** (e.g. search → filter → call RAG → summarize).
- You need to integrate with **existing systems** (ticketing, CRM, data lakes).
- You want **traceability**: which tools were used, which docs were read, etc.

3.2 CORE AGENT PATTERNS

We'll refer to these patterns throughout Day 3:

1. TOOL-CALLING AGENT

- LLM chooses **one tool** at a time based on user input.
- Example:
  - `rag_service_tool(question)` → calls your accelerator `/ask` endpoint.
  - `calculator_tool(expression)` → safe arithmetic evaluation.
- Agent picks a tool, calls it, and formats the result back to the user.

2. REACT (REASON + ACT)

- LLM alternates between **thinking** and **acting**:
  - **Thought:** "I should call the RAG service to get context."
  - **Action:** Use tool `rag_service_tool`.
  - **Observation:** Tool output.
  - Repeat…

Many frameworks (CrewAI, LangGraph, watsonx Orchestrate "react style") build on this idea.

3. PLAN–ACT

- LLM **plans a sequence** of steps, then executes them:

3.3 FRAMEWORK TOUR

In the morning we'll conceptually walk through four frameworks that implement these patterns.

3.3.1 CREWAI

- **What it is:** A Python library for multi-agent "crews".
- **Mental model:**
  - `Agent` – role, goal, backstory, tools.
  - `Task` – what needs doing, expected output.
  - `Crew` – group of agents + tasks + process (`Process.sequential`, `Process.hierarchical`, ...).
- **Where it shines:**
  - Faster prototyping of **multi-agent** patterns.
  - Narrative workflows (research → writing → editing).
- **Workshop angle:**
  - A single CrewAI agent using your **accelerator RAG API + calculator**.

3.3.2 LANGFLOW

- **What it is:** A visual builder for LangChain flows.
- **Mental model:**
  - Drag-and-drop components (LLM, retriever, tools, routers, prompts).
  - Connect them as a **graph**.

3.4 MORNING FLOW (APPROX. 4H)

Suggested agenda (adapt to your group):

1. **Intro & recap (30–45 min)**
   - Why agents on top of RAG?
   - Quick recap of RAG architecture from Day 2.
2. **Agent patterns walkthrough (45 min)**
   - Tool-calling, ReAct, plan–act, multi-agent, graph-based.
3. **Framework lightning demos (90–120 min)**
   - **CrewAI mini-demo:**
     - One agent with a “support engineer” role.
     - Tools: calculator + stubbed RAG tool.
   - **Langflow mini-demo:**
     - Visual RAG chain that calls an LLM and shows citations.
   - **LangGraph mini-demo:**
     - Simple graph: retrieve → generate.
   - **Orchestrate concept demo:**
     - Show YAML / ADK structure for a “Hello World Agent”.
4. **Bridge to afternoon labs (15–30 min)**
   - Show how the accelerator `/ask` endpoint becomes **the** key tool.

3.5 HOW THIS CONNECTS TO LABS

- **Day 2 → Day 3 bridge:**
  - Day 2 gave you a **production-like RAG service** (`/ask` endpoint).
  - Day 3 gives you **agents** that call that service as a tool.
- **Lab 3.1 – Local Agent in simple-watsonx-environment:**
  - Agent has two tools: RAG service, calculator.
  - Planner LLM chooses which tool to call, then composes final answer.
- **Agent frameworks (CrewAI / LangGraph / Orchestrate):**
  - You can re-express the same logic in different frameworks:
    - CrewAI for Python multi-agent setups.
    - LangGraph for stateful workflows and evaluation.
    - Orchestrate for production deployment and governance.

By the end of Day 3, your mental model should be:

Docs → RAG (Day 2) → Agent on top (Day 3) → Orchestrated & governed in watsonx (beyond the workshop).