

Agentic Frameworks & Multi-Agent Systems

Build production-grade agentic systems with Python, Kafka, RAG, and full-stack observability in 2-days

FA-0018 · 2-days

Agentic & Generative AI · Software Development · DevOps, Cloud & Infrastructure

Introduction

Agentic systems have crossed a line: they are no longer experimental assistants but operational software components that plan, delegate, retrieve knowledge, execute tools, and coordinate with other agents under real-world constraints. When these systems fail, they fail like distributed systems—through cascading retries, message backlogs, silent degradations, and untraceable decisions. This course is built for engineers who need to **design, build, and operate agentic systems that hold up under pressure**, not demos that work once.

This is a **high-intensity, 2-day immersion** focused on hard-core agentic development using Python on Linux, Kafka for modular communication, hybrid LLM deployment (cloud endpoints and on-prem), RAG at scale, and deep observability with Grafana-centric tooling. The instructor brings **over 30 years of industry experience** and teaches this from the perspective of production systems that ship, break, get audited, and must be fixed fast—**industry-demanded engineering, not academic abstraction**.

Learning outcomes

By the end of the course, participants will be able to:

- Architect agentic and multi-agent systems as **distributed software**, with explicit state, contracts, and failure handling.
- Implement **graph-based and role-based multi-agent orchestration** with bounded autonomy and deterministic recovery paths.

- Build **Kafka-backed agent communication layers** using event-driven and request-reply patterns that tolerate retries and partial failure.
 - Design and deploy **hybrid LLM infrastructure**, seamlessly routing between hosted endpoints and on-prem inference.
 - Engineer **RAG pipelines** specifically for multi-agent environments with retrieval routing, context governance, and citation discipline.
 - Instrument agentic systems end-to-end using **metrics, logs, and traces** suitable for postmortems and live debugging.
 - Evaluate agent performance using **offline regression suites and online canaries**, tied to latency, cost, and correctness.
 - Operate agentic platforms with confidence through dashboards, alerts, runbooks, and rollback strategies.
-

Prerequisites

Prerequisites (strict – not optional)

This is not an introductory course. Participants **must already have**:

Strong, production-level **Python experience**

- Async programming (async/await)
- Typed codebases
- Packaging and dependency management
- Writing and maintaining tests

Daily working proficiency with **Linux**

- Shell scripting
- Process management
- Networking fundamentals
- Reading logs and system metrics

Prior hands-on experience with **distributed systems concepts**

- Message queues or event streams
- Backpressure, retries, and timeouts
- Idempotency and eventual consistency

Practical familiarity with **Kafka or an equivalent system**

- Topics, partitions, consumer groups
- Message serialization concepts

Working knowledge of **LLMs and embeddings**

- Prompting basics
- Vector search concepts
- REST/HTTP APIs

Ability to run **containerized environments** locally

- Docker or equivalent
- Comfort reading and modifying compose files

Participants who are new to Python, Linux, Kafka, or backend engineering will struggle and are strongly advised not to enroll.

Training outline

1. Agentic systems as production software

1. What distinguishes agentic systems from workflows and chatbots
2. Autonomy, delegation, and decision boundaries
3. Determinism vs probabilistic execution
4. Failure surfaces unique to agentic behavior
5. Design axioms for production-grade agents

2. Core agent architecture

1. Agent responsibilities and lifecycle
2. Role separation: planner, executor, retriever, reviewer
3. Tool abstraction and execution authority

Memory models

1. Ephemeral context
 2. Persistent state
 3. Externalized memory stores
5. State ownership and mutation rules

3. Multi-agent orchestration patterns

1. Supervisor-worker hierarchies

2. Router-based delegation to specialist agents
3. Debate, critique, and verification loops
4. Hierarchical planning and task decomposition

Graph-based orchestration

1. Nodes, edges, guards, and reducers
2. Interrupts, checkpoints, and resume semantics

Bounding agent behavior

1. Step limits
2. Tool limits
3. Time budgets
4. Escalation paths

4. Deployment / development model

1. Repository structure for agentic platforms
2. Service boundaries and shared contracts
3. Python runtime management strategies
4. Environment parity between local and production
5. Secrets and configuration management

5. Agent communication backbone

1. Why message-driven architectures matter for agents
2. Command vs event semantics
3. Topic design and naming discipline
4. Schema evolution and compatibility rules
5. Correlation and causation tracking
6. Idempotency keys and deduplication strategies
7. Consumer group design for agent workers
8. Backpressure, lag management, and flow control
9. Dead-letter topics and recovery workflows
10. Request-reply patterns over Kafka
11. Saga-style coordination and compensation logic

6. LLM access layer design

1. Provider-agnostic client interfaces

2. Timeouts, retries, and circuit breakers
3. Token, latency, and cost budgeting
4. Streaming responses and partial results
5. Error classification and fallback logic

7. Hybrid LLM deployment

1. Cloud-hosted endpoints vs on-prem inference
2. OpenAI-compatible APIs for local model serving
3. Model routing and selection strategies
4. Using small models for routing and control
5. Using larger models for synthesis and reasoning
6. Operational tradeoffs: cost, latency, privacy, reliability

8. Prompting and contracts in multi-agent systems

1. Prompts as executable specifications
2. Role prompts vs task prompts
3. Structured outputs and schema validation
4. Guardrails against prompt drift
5. Context budgeting per agent hop

9. Tooling layer engineering

Tool classification

1. Pure computation
 2. External reads
 3. Side-effecting writes
2. Tool contracts and schemas
 3. Validation and error envelopes
 4. Async execution and concurrency limits
 5. Retry behavior and idempotent writes
 6. Capturing artifacts and execution provenance
 7. Tool security and access control

10. RAG for multi-agent environments

1. Retrieval as a first-class system component
2. Query rewriting and decomposition
3. Retrieval routing by agent role
4. Index design and metadata discipline
5. Chunking strategies and versioning
6. Context filtering and deduplication
7. Citation enforcement and claim grounding
8. Passing context by reference in message-driven systems
9. Cache design and reuse across agent hops

11. Reliability engineering for agent graphs

1. Designing for retries and partial failure
2. Compensation actions and rollback logic
3. Deterministic replays and forensic debugging
4. Handling poisoned messages and stuck workflows
5. Load shedding and graceful degradation

12. Observability for agentic systems

1. Structured logging with correlation IDs
2. Distributed tracing across services and queues

Metrics that matter for agents

1. Throughput
 2. Error rates
 3. Queue lag
 4. Saturation
4. Instrumenting agent decisions and tool calls
 5. Prompt and response redaction strategies
 6. Dashboards for agent health and system health
 7. Alerting on silent failures and runaway behavior

13. Performance, cost, and governance

1. Concurrency tuning and worker sizing

2. Queue and buffer sizing
3. Caching strategies
4. Model routing optimization
5. Cost attribution per agent and per workflow
6. Release gates and rollback criteria

14. Evaluation strategy

1. Offline evaluation harness design
2. Scenario-based regression suites
3. Failure clustering and root-cause analysis
4. Online canaries and synthetic traffic
5. Drift detection and long-term quality signals

15. Capstone production specification

1. End-to-end multi-agent system design
2. Agent roles and interaction map
3. Kafka topic and message contract layout
4. Tool catalog and execution rules
5. RAG indexes and retrieval policies
6. Observability requirements and SLOs
7. Operational runbooks for failure scenarios