

# Kafka Integration and Deployment Technologies

## Advanced lab practice with clients, containers and orchestration

FA-0735 · 5 days

DevOps, Cloud & Infrastructure · Software Development

---

## Introduction

Connect existing networking and programming skills through a supplied Kafka application and deployment lab. Review containers, Kafka architecture and resilience, selected Java/Python clients and an introductory Kubernetes deployment design.

The five-day course assumes technical foundations and uses a controlled environment. It does not promise complete production mastery, automatic interoperability or comprehensive programming instruction in both client languages. Remote or cloud access is supplied only as arranged for the training lab.

## Learning outcomes

- Explain Kafka brokers, KRaft controllers, topics, partitions, replication and consumer groups.
- Operate selected Docker and Kafka lab components and inspect configuration/logs.
- Investigate a controlled failure, lag or performance example using comparable evidence.
- Implement a selected Java or Python producer/consumer flow and explain serialisation and offsets.
- Package a small application and compare Compose with Kubernetes orchestration.
- Recognise stateful storage, security and recovery requirements and a Spark/data-pipeline integration context.

## Prerequisites

- Understand networking systems

- Remoting capability

Comfort with a suitable terminal, Linux/file operations and either Java or Python; the other language is shown through comparison examples.

Compatible Docker, Kafka and Kubernetes training environments with approved access, required resources and lab-only permissions. Remote access is not mandatory for every real-world deployment.

# Training outline

## 1. Day 1 — Containers and the lab environment

- Docker images, containers, processes, ports and volumes; supported platform-specific installation/setup.
- Pull/build and run selected supplied web/container examples, including PHP/Apache or Nginx where useful.
- Inspect port mapping, persistent data and basic lifecycle commands.
- Distinguish application configuration and credentials from image content; exercise and review the isolated lab.

## 2. Day 2 — Kafka architecture and basic operation

- Brokers, KRaft metadata/controller roles, topics, record structure, partitions and producer/consumer responsibilities.
- ZooKeeper ensembles are historical Kafka context; use a compatible current KRaft setup rather than assume current upstream Kafka requires ZooKeeper.
- Access the supplied Unix-like lab, install or inspect compatible runtime/images and use an approved remote editor where needed.
- Produce/consume console records, create selected partitions and inspect offsets and record/log storage.
- Compare multiple producers/consumers and explain boundaries between separate clusters.

## 3. Day 3 — Replication, failures and groups

- Inspect a controlled multiple-broker setup, configuration, logs and replicated partitions.
- Demonstrate a bounded failure/recovery scenario and document its assumptions; distinguish replication from a full backup or continuity strategy.
- Review storage management and client observations during failure.
- Consumer-group membership, partition assignment and idle members; inspect appropriate offsets and lag.

- Introduce inter-cluster replication as a separate design decision, not an automatic producer behaviour; assess the example.

## 4. Day 4 — Client development and measured behaviour

- Run a small baseline performance/lag experiment and compare a selected configuration change under stated conditions.
- Use Java or Python for the main client exercise; inspect setup, serializers, send/receive errors and resource cleanup.
- Compare a supplied example in the other language rather than promise two complete language courses.
- Consumer offset commit and partition assignment are consumer concerns, not producer controls.
- Refactor and test the small client, inspect its failure/retry assumptions and review results.

## 5. Day 5 — Packaging, orchestration and data integration

- Package the client/application with Docker; review image/registry use, staging and Compose multi-service configuration. Docker is not a language package manager, and Compose is not PHP Composer.
- Introduce Kubernetes YAML, Pods, Deployments/ReplicaSets for suitable stateless clients, networking and service boundaries.
- For Kafka/stateful components, review stable identity, persistent volumes and an appropriate StatefulSet/operator approach; ordinary replicas alone do not guarantee durable broker state.
- Inspect a supplied orchestration demonstration and discuss operations, permissions, TLS, monitoring and recovery requirements.
- Introduce selected Spark/Hadoop consumption or ETL/ELT integration and verify the small pipeline's output; document the limits of the five-day lab.