Faysal Aziz

Design Patterns with Kotlin & Containerizations

Hands-on mastery of robust software patterns, containerization, and orchestration—taught with real Linux environments and real-world workflows in 2 days

FA-0166 · 2 days

Software Development · DevOps, Cloud & Infrastructure

Introduction

The digital battlefield has evolved. In boardrooms across Silicon Valley and beyond, CTOs no longer ask if they should embrace cloud-native architectures—they ask how fast they can transform their legacy systems before competitors leave them in the dust.

Welcome to the epicenter of that transformation.

This isn't another theoretical deep-dive into microservices buzzwords or another surface-level tutorial on containerization. This is where three decades of hard-won expertise meets the cutting edge of modern software architecture. Where Kotlin's elegant expressiveness dances with Docker's containerization prowess. Where Kubernetes orchestrates not just your applications, but your competitive advantage.

You're here because mediocrity isn't an option.

Every lesson unfolds in the trenches of real Linux environments—the same servers humming quietly in data centers that power the applications millions depend on daily. No sanitized sandbox environments. No toy problems. Just the raw, unforgiving reality of production systems where uptime isn't negotiable and scalability isn't optional.

Your instructor has witnessed the evolution from monolithic mainframes to today's distributed mesh of intelligent services. Survived the dot-com crash. Architected systems that scaled from thousands to millions of users. Debugged failures at 3 AM when millions of dollars hung in the balance.

This is that wisdom, distilled.

Every pattern explored has been battle-tested in the fires of real-world deployment. Every technique demonstrated has solved actual business problems. Every line of code written together will echo the rhythms of industry-leading practices that separate the architects from the amateurs.

Step into the arena where theory meets reality, where your expertise will be forged in the same crucible that has shaped the digital infrastructure powering our modern world.

Learning outcomes

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

- Understand and apply key creational, structural, and behavioral design patterns in Kotlin.
- Containerize Kotlin applications using Docker with best practices.
- Deploy and manage containerized apps using Kubernetes on Linux servers.
- Leverage Kubernetes-specific container orchestration patterns such as Sidecar, Init, Operator, and Probe patterns.
- Integrate design patterns seamlessly with Docker and Kubernetes workflows.
- Debug, iterate, and optimize deployments within real Linux environments.

Prerequisites

Participants should have:

- Intermediate knowledge of Kotlin programming.
- Familiarity with Linux command line and basic server operations.
- Basic understanding of Docker and Kubernetes concepts (e.g., containers, pods, services).
- Access to Linux servers (physical or virtual) to practice Docker and Kubernetes hands-on.
- Java tooling and a build system like Maven or Gradle available.

Lab Requirements:

- VS Code with the SSH plugin.
- Ports 21 and 22 (outgoing for remote connection establishment).
- Terminal / Console.
- Unrestricted internet access (so that students can SSH to remote training servers).
- Alternatively, students may also use WSL2 containers.

Training outline

1. 1. Fundamentals of Design Patterns in Kotlin

Kotlin idioms vs. traditional patterns

- Comparison of Kotlin's functional features with classical OOP design patterns

Creational Patterns:

- Singleton (object declarations), Factory, Abstract Factory, Builder, Prototype

Structural Patterns:

- Adapter, Decorator, Proxy, Facade, Composite

Behavioral Patterns:

- Strategy, Observer, State, Template Method, Command

Concurrency Patterns specific to Kotlin:

- Coroutines-based patterns for async tasks and synchronization

2. 2. Containerization with Docker

- Docker fundamentals: images, containers, layers
- Building images efficiently with multi-stage Dockerfiles
- Dockerizing Kotlin applications (e.g., Spring Boot app in Kotlin)
- Best practices: slim base images, build profiles, tagging, layering

3. 3. Kubernetes Overview & Pattern Integration

- Kubernetes core concepts: Pods, Deployments, Services, ConfigMaps, Secrets

  Essential container patterns in Kubernetes:

  - **Sidecar** – for logging, monitoring, helper processes
  - **Init Container** – for pre-boot steps
  - **Adapter & Ambassador** – protocol translation, external service abstraction
  - **ConfigMap / Secret Pattern** – for config and secrets management
  - **Work Queue, Health Probe, Predictable Demands (resource specs)**

  Higher-level patterns:

  - **Controller / Operator Pattern** – encapsulating operational logic in custom controllers

4. 4. Real-World Application: From Code to Cluster

- Set a baseline: deploy a simple Kotlin microservice in Docker

- Deploy to Kubernetes cluster (e.g., Minikube or local K8s) as real servers
- Use Init container to seed data, Sidecar for logging or metrics collection
- Manage configs and secrets via ConfigMaps/Secrets
- Set up readiness and liveness probes, define resource requests/limits
- Scale services, perform rolling updates, simulate failure and recovery

5. 5. Advanced Pattern: Kotlin Operators in Kubernetes

- Introduction to the Operator pattern in Kubernetes
- How to design a Custom Resource Definition (CRD) in Kotlin
- Implementing a Controller in Kotlin using fabric8 Java/Kotlin client
- Deploy and test a simple operator that manages “WebServer” CRD resources

6. 6. End-to-End Integration & Best Practices

- Combine all pieces into a cohesive microservice flow: design patterns, container build, Kubernetes patterns, Operator logic
- Debugging techniques on Linux servers: logs, kubectl exec, image issues, pod health
- CI/CD pipeline overview: build, test, push Docker images, automate deployment (optional advanced demo)

© 2026 Faysal Aziz · Kuala Lumpur