Microservices Development Bootcamp with immersive project (using Spring Boot and Docker) Training (5 Days)

This training course will enable the attendees to understand the value proposition of Microservices as well as the implementation of this technology. You will learn about the pros and cons of breaking up the monolithic applications prevalent in the enterprise space and converting them into Microservices-based solutions.
Spring Boot is a technology stack that builds on the popular Spring Framework to allow Spring-based applications as stand-alone jar files that host their own web servers. This approach works nicely with deployment automation and rapid scaling. In this course, we will implement Microservices using Spring Boot.
Linux containers are changing the way companies think about service development and deployment. Containers play a vital role in the modern data-center, and Docker is leading the way. In this course, we will show how to package and deploy your Microservices using Docker.

Objectives

- Introduction to DevOps practices and GitFlow
- Breaking up Monoliths into Microservices
- Build using Gradle
- Creating RESTful services with Spring Boot
- Using databases and JPA in Spring Boot
- Security patterns and best practices in Spring Boot
- Deploying resilient and scalable services
- Traffic routing patterns
- Metrics and tracing of Microservices

Audience

Developers, solution architects and technical team leads.

Prerequisites

- Knowledge of Java – Equivalent to https://www.webagesolutions.com/courses/WA2494-introduction-to-java-8-using-eclipse
- Git - Equivalent to https://www.webagesolutions.com/courses/WA2410-git-training-introduction-to-version-control-with-git

Course Outline

Chapter 1. DevOps Fundamentals

- Why DevOps
- What is DevOps?
- Collaborative, Matrixed and Cross-Functional Teams
- Key Components of Successful DevOps Teams
- DevOps-ification
- DevOps Vocabulary
- DevOps Goals
- Not DevOps - Crush Buzzwords
- Driving Business Outcomes with DevOps
- Technology-Enabled Business
- DevOps Key Enabler for Digital Transformation
- Core Values and Mission
- Core Values - Culture
- Core Values - Automation
- Core Values - Measurement
- Core Values - Sharing
- Communication
- Collaboration
- Value Stream Mapping
- Behavioral Patterns for Success
- DevOps Org Structures
- DevOps Team - Separate
- DevOps Merged Organization
- DevOps Overlapped Organization
- Organizational Structure Leadership
- What Does Continuous Delivery Mean?
- Deployment Pipelines
- Your Organization is Doing CD if …
- Pipelining for CD
- Continuous Integration
- CI Pipeline
- CD & CI Methodologies
- Key Tool Categories for CI/CD

Git

Chapter 2. Introduction to GitFlow

- What is GitFlow
- Contrast gitFlow with other comparable best practices in the market
- Make sure it connects to CI/CD
- Benefits
- How GitFlow works?
- How GitFlow works? (Contd.)

- What is GitFlow? (Contd.)
- How GitFlow works? (Contd.)
- GitFlow Extension
- Initializing GitFlow
- Features
- Release
- Hotfixes

Developing Microservices

Chapter 3. Breaking Up Monoliths – Pros and Cons

- Traditional Monolithic Applications and Their Place
- Disadvantages of Monoliths
- Developer's Woes
- Architecture Modernization
- Architecture Modernization Challenges
- Microservices Architecture is Not a Silver Bullet!
- What May Help?
- In-Class Discussion

Chapter 4. Microservice Development

- What are Microservices?
- Microservices vs Classic SOA
- Principles of Microservices Architecture Design
- Business Domain-Centric Design
- Designing for failure
- Microservices Architecture – Pros
- Microservices Architecture – Cons
- Docker and Microservices
- Microservice Deployment with Docker – Workflow
- Writing Dockerfile
- Kubernetes
- What is OpenShift
- OpenShift Architecture
- Microservices and Various Applications
- Web Applications
- Web Applications – Reference Architecture
- Web Applications – When to use?
- Single Page Applications
- Single Page Applications – Benefits
- Traditional Enterprise Application Architecture
- Sample Microservices Architecture
- Serverless & Event-driven Microservice – AWS Lambda

Chapter 5. Twelve factor Applications

- Twelve-factor Applications

- Twelve Factors, Microservices, and App Modernization
- 12-Factor Microservice Codebase
- 12-Factor Microservice Dependencies
- 12-Factor Microservice Config
- 12-Factor Microservice Backing Services
- 12-Factor Microservice Continuous Delivery
- 12-Factor Microservice Processes
- 12-Factor Microservice Data Isolation
- 12-Factor Microservice Concurrency
- 12-Factor Microservice Disposability
- 12-Factor Microservice Environment Parity
- 12-Factor Microservice Logs
- 12-Factor Microservice Admin Processes

Chapter 6. REST Services

- Many Flavors of Services
- Understanding REST
- Principles of RESTful Services
- REST Example – Create
- REST Example – Retrieve
- REST Example – Update
- REST Example – Delete
- REST Example – Client Generated ID
- SOAP Equivalent Examples
- REST Example – JSON
- REST vs. gRPC
- RESTful Services Usage
- Additional Resources

Gradle and Docker

Chapter 7. Introduction to Gradle

- What is Gradle
- Why Groovy
- Build Script
- Sample Build Script
- Task Dependency
- Plugins
- Dependency Management
- Gradle Command-Line Arguments

Chapter 8. Docker Introduction

- What is Docker
- Where Can I Ran Docker?
- Docker and Containerization on Linux
- Linux Kernel Features: cgroups and namespaces

- The Docker-Linux Kernel Interfaces
- Docker Containers vs Traditional Virtualization
- Docker Integration
- Docker Services
- Competing Systems
- Docker Command-line
- Starting, Inspecting, and Stopping Docker Containers
- Docker Application Container Public Repository
- Registries
- Your Own Docker Image Registry
- One Process per Container
- The Dockerfile
- A Sample Dockerfile

Spring Framework

Chapter 9. Introduction to Spring Boot

- What is Spring Boot?
- How is Spring boot related to Spring?
- Spring Boot Main Features
- Understanding Java Annotations
- Spring MVC Annotations
- Example of Spring MVC-based RESTful Web Service
- Spring Booting Your RESTful Web Service
- Spring Boot Skeletal Application Example
- Converting a Spring Boot Application to a WAR File
- Externalized Configuration
- Starters
- The 'pom.xml' File
- Spring Boot Gradle Plugin
- HOWTO: Create a Spring Boot Application

Chapter 10. Overview of Spring Database Integration

- DAO Support in Spring
- Spring Data Access Modules
- Spring JDBC Module
- Spring ORM Module
- DataAccessException
- @Repository Annotation
- Using DataSources
- DAO Templates
- DAO Templates and Callbacks
- ORM Tool Support in Spring

Chapter 11. Using Spring with JPA or Hibernate

- Spring JPA

- Benefits of Using Spring with ORM
- Spring @Repository
- Using JPA with Spring
- Configure Spring Boot JPA EntityManagerFactory
- Application JPA Code
- "Classic" Spring ORM Usage
- Spring JpaTemplate
- Spring JpaCallback
- JpaTemplate Convenience Features
- Spring Boot Considerations
- Spring Data JPA Repositories
- Database schema migration for CI using Liquibase
- Spring REST Services

Chapter 12. REST Services With Spring MVC

- Spring MVC Components
- Spring MVC @RequestMapping with REST
- Working With the Request Body and Response Body
- @RestController Annotation
- Implementing JAX-RS Services and Spring
- JAX-RS Annotations
- Spring Security
- Spring Security Options
- Spring Security Features
- Java Clients Using RestTemplate
- RestTemplate Methods

Chapter 13. Spring Security

- Securing Web Applications with Spring Security 3.0
- Spring Security 3.0
- Authentication and Authorization
- Programmatic v Declarative Security
- Getting Spring Security from Gradle
- Spring Security Configuration
- Spring Security Configuration Example
- Authentication Manager
- Using Database User Authentication
- LDAP Authentication
- Security Assertion Markup Language (SAML)
- Setting up an SSO provider
- Setting up the project
- The pom.xml file setup
- The application.yml file setup
- The Spring Security configuration files
- The resources folder setup
- Running and testing the application
- Microservices - Dealing with the State [WA2755]

- Microservices - How Can I Maintain State? [WA2755]
- The OAuth2 specification
- Access token
- Grant types
- JSON Web Tokens (JWT)
- OAuth2 support in Spring Security
- Example – Microservice 1 creates a JWT token and Microservice 2 authenticates using that token to do anything

Chapter 14. Traffic Routing Patterns

- Service Discovery in Microservices
- Load Balancing in Microservices
- Rate Limiting
- Circuit breakers
- Existing solutions (nginx/HAProxy/API Gateways – AWS API Gateway)

Chapter 15. Service Implementations (logs vs. metrics)

- Logging best practices
- Correlate Requests With a Unique ID
- Include a Unique ID in the Response
- Send Logs to a Centralized Location
- Structure Your Log Data
- Add Context to Every Request
- Write Logs to Local Storage
- Log Useful and Meaningful Data to Avoid Regret
- Metrics using Prometheus
- Overview
- Prometheus
- Service discovery
- Exposing metrics in services
- Querying in Prometheus
- Grafana
- Tracing using Jaeger
- OpenTracing and the fundamental concepts – span, trace
- Jaeger – a distributed tracing system, from Uber
- Jaeger Client Libraries
- Agent
- Collector
- Query
- Ingester

Project Details

Sprint #1 (First 2 days)

Form pairs to work on the application (pair programming). Lay out a plan for the process of creating the registration application. Your approach should align with the fundamentals of DevOps and Microservices. Discuss the application components and lay out a logical layout, a feature set and a backlog of tasks to be completed during the development. Remember there will be limited time to develop the application and you will need to display a functional PoC by the end of the first sprint. By the end of this session you should have a code repository where the Integration can get started. Extend your integration plan from session 1 by designing a build process using the Gradle build tool. Develop a functional RESTful service that can respond to queries and test the build process. The end result of Sprint one should be a build process and an operational Proof-of-concept RESTful service. Since this is the end of a sprint, the last hour will be reserved for the teams to discuss lessons learned.

Sprint #2 (Next 2 days)

Using the Spring Framework develop a stateful back-end for your RESTful frontend that will be provided to you so that data being sent to the REST service can be stored in the database layer. Your RESTful service at this point should be stateless and should be able to communicate with as well as store all persistent data in the database. Deploy this service in Docker.
Create a new Microservice that generates a JWT token for authentication and authorization of your current Microservice. Make sure that your current Microservice cannot perform any tasks without the JWT security measure provided by the new Microservice. The teams will need to develop integration test to ensure that the service is able to work properly with the database. We will not discuss lesson learned at the end of this sprint. The presentation on Day 5 will discuss that.

Sprint #3 (Last Day)

Introduce a service discovery configuration that will make the application available and usable by the end users. Design a load balancing approach for your application. Deploy the application service with the RESTful service API into separate load balancing groups. Configure the applications to talk to the load balancer instead of communicating directly. Your team has the freedom to choose the tools to complete these tasks. The teams will also need to provide system tests to ensure the new infrastructure works as expected and can survive failures.