Microservices Application Documentation

Alvaro Lopera Mendizabal

30/05/2024

Introduction

This document provides comprehensive documentation for a microservices application developed using Java Spring Boot, Docker, and Kubernetes. The application is comprised of three microservices: *Shopfront*, *Product Catalogue*, and *Stock Manager*. It offers an overview of the project structure, technologies utilized, and instructions for running the application.

Project Structure

The project structure is divided into three main microservices:

Shopfront

The *Shopfront* microservice serves as the primary entry point for end-users, providing both a web-based user interface and an API-driven interface.

Product Catalogue

The *Product Catalogue* microservice offers product details such as name and price.

Stock Manager

The *Stock Manager* microservice provides inventory-related information including SKU and quantity.

Scripts and Files

Several important scripts and files are included in the project:

- **prepare-app.sh:** A script to prepare the application and create all related services.
- **remove-app.sh:** A script to remove the application and all related services.
- **Docker compose & Docker files:** Docker files for building images and composing containers.

Technologies Used

- Java
- Spring Boot 3.2.4
- Maven 4.0.0
- Docker
- Kubernetes

About Maven & Spring Boot

Maven

Apache Maven is a build automation tool used primarily for Java projects. It manages project dependencies and facilitates project management tasks such as building, testing, and deployment. Maven uses a Project Object Model (POM) file to describe the project configuration and dependencies.

Spring Boot

Spring Boot is an open-source Java-based framework used to create standalone, production-ready Spring-based applications with minimal configuration. It simplifies the development process by providing defaults for commonly used configurations and allows developers to focus more on business logic rather than boilerplate code.

Dependencies Analysis

Let's now analyze the dependencies specified in the provided `pom.xml` file for the *shopfront* microservice:

- **spring-boot-starter-thymeleaf**: This dependency provides integration with Thymeleaf, a modern server-side Java template engine for web and standalone environments. Thymeleaf enables seamless development of web applications by allowing developers to write HTML templates with dynamic content using natural templates.
- **spring-boot-starter-actuator**: Spring Boot Actuator provides production-ready features to help monitor and manage the application. It includes built-in endpoints for metrics, health checks, and various monitoring-related tasks.
- **spring-cloud-starter-hystrix**: Hystrix is a latency and fault tolerance library designed to isolate points of access to remote systems, services, and third-party libraries. It helps prevent cascading failures and provides fallback options when failures occur.
- **spring-cloud-starter-eureka**: Eureka is a service registration and discovery framework. It allows microservices to register themselves with the registry and discover other services registered with it. This enables dynamic scaling and load balancing in distributed systems.
- **spring-boot-starter-test**: This dependency provides support for testing Spring Boot applications. It includes various testing libraries and utilities such as JUnit, Mockito, and Hamcrest, making it easier to write and execute tests for Spring Boot applications.

Additionally, the `pom.xml` file defines properties for the project, including the encoding used for source files (`UTF-8`), and versions for Dropwizard (`1.3.27`) and Guice (`4.2.3`).

About Docker & Kubernetes

Docker

Docker is a containerization platform that enables developers to package applications and their dependencies into lightweight, portable containers. These containers are isolated environments that encapsulate everything needed to run an application, including code, runtime, system tools, libraries, and settings. Docker provides consistency across different environments and simplifies the process of building, shipping, and running applications.

Kubernetes

Kubernetes is an open-source container orchestration platform for automating the deployment, scaling, and management of containerized applications. It provides features for container deployment, scaling, load balancing, self-healing, and more, allowing developers to deploy and manage containerized applications

at scale seamlessly. Kubernetes abstracts away the underlying infrastructure, enabling applications to be deployed and run consistently across different environments.

Usage for this Application

For this application, Docker and Kubernetes are used to containerize and orchestrate the microservices:

- **Docker**: The provided Dockerfile for each microservice specifies how to build a Docker image containing the application code and its dependencies. This image is then used to create Docker containers that can be deployed and run consistently across different environments.
- **Kubernetes**: Kubernetes is used to deploy and manage the Docker containers in a cluster environment. The provided YAML files define Kubernetes resources such as Services and Deployments for each microservice. These resources ensure that the microservices are highly available, scalable, and can be accessed reliably.

Additionally, each microservice in the application exposes specific ports:

- **Shopfront Microservice**: Exposes port 8010 for HTTP communication. This port is used to access the Shopfront microservice and interact with the user interface or API.
- **Product Catalogue Microservice**: The port configuration for the Product Catalogue microservice is not explicitly provided in the snippet, but it should expose a port for HTTP communication similar to the Shopfront microservice.
- **Stock Manager Microservice**: The port configuration for the Stock Manager microservice is also not explicitly provided, but it should expose a port for HTTP communication similar to the other microservices.

Spring Boot + Maven with Docker + Kubernetes

Combining Spring Boot and Maven with Docker and Kubernetes offers a powerful toolset for developing, packaging, deploying, and managing microservices-based applications.

Development Workflow

1. **Development with Spring Boot**: Spring Boot simplifies the development process by providing default configurations and conventions, reducing boilerplate code, and enabling rapid application development. Developers can focus on writing business logic while Spring Boot handles the underlying infrastructure.
2. **Dependency Management with Maven**: Maven is used for dependency management and project build automation. It simplifies the process of managing project dependencies, building the project, and packaging it into a distributable format. Developers can specify project dependencies in the `pom.xml` file, and Maven resolves and downloads them automatically.

Containerization with Docker

1. **Packaging with Docker**: Docker is used to package the Spring Boot applications and their dependencies into lightweight, portable containers. Developers create a Dockerfile for each microservice, specifying the base image, dependencies, and runtime configurations. Docker builds an image from the Dockerfile, encapsulating the application and its environment into a container.
2. **Image Distribution and Deployment**: Docker images can be distributed and deployed across different environments consistently. Developers can push Docker images to container registries like Docker Hub, and Kubernetes can pull these images to deploy them in a Kubernetes cluster.

Orchestration with Kubernetes

1. **Deployment with Kubernetes**: Kubernetes orchestrates the deployment, scaling, and management of Docker containers in a cluster environment. Developers define Kubernetes resources such as Services and Deployments using YAML files, specifying desired state and configurations for the microservices.
2. **Automatic Scaling and Self-healing**: Kubernetes provides features for automatic scaling and self-healing, ensuring that the microservices are highly available and resilient. Kubernetes monitors the health of containers and automatically restarts or reschedules them if they fail or become unresponsive.
3. **Service Discovery and Load Balancing**: Kubernetes facilitates service discovery and load balancing by dynamically assigning IP addresses to containers and routing traffic to healthy instances. Services can be accessed consistently using DNS names or Service IPs, regardless of the underlying infrastructure.

By leveraging Spring Boot + Maven with Docker + Kubernetes, developers can streamline the development, packaging, deployment, and management of microservices-based applications, enabling rapid iteration, scalability, and resilience.