DevOps Global Project Report

CI/CD Pipeline with Docker, Kubernetes, Jenkins, and Monitoring

Authors :

- Khadija IMAD
- Chaima EDDIB
- Farah BOUHZAM

Supervisors :

- Mme. Karima MOUMANE
- Mr. Abdelkader EL MAHDAOUY

UM6P

Année Universitaire2024–2025

Contents

1 Introduction 3
1.1 Project Context . . . 3
1.2 Application Description . . . 3

2 GitHub – Source Control 4
2.1 Repository Setup . . . 4
2.2 Branching Strategy . . . 4
2.3 Commit History . . . 4

3 Jenkins – Continuous Integration 5
3.1 Installation and Configuration . . . 5
3.2 CI Pipeline . . . 5
3.2.1 Pipeline Execution and Results . . . 7
3.3 Build Logs . . . 8

4 SonarQube – Code Quality 12
4.1 Installation . . . 12
4.2 Project Analysis . . . 12
4.2.1 Impact of JaCoCo Integration . . . 14
4.3 Issues Resolution . . . 14

5 Docker – Containerization 15
5.1 Backend Dockerfile . . . 16
5.2 Frontend Dockerfile . . . 17
5.3 Docker Compose Configuration . . . 18
5.4 Building Docker Images . . . 19
5.5 Publishing to DockerHub . . . 19

6 Kubernetes – Deployment 21
6.0.1 Backend Deployment . . . 23
6.0.2 Backend Service . . . 24
6.0.3 Frontend Deployment . . . 25
6.0.4 Frontend Service . . . 26
6.0.5 MySQL Deployment . . . 27
6.0.6 MySQL Service . . . 28
6.0.7 Ingress Configuration . . . 29
6.1 Deployment on Minikube . . . 29
6.1.1 Verification with kubectl . . . 30

6.1.2 Accessing the Application . . . 31

7 Prometheus & Grafana – Monitoring 32

7.1 Deployment and Configuration . . . 32

7.2 Prometheus ServiceMonitor . . . 32

7.3 Prometheus Targets Health – Initial State . . . 33

7.3.1 Resolution:Spring Boot Actuator Integration . . . 33

7.4 Grafana Dashboard . . . 34

7.5 Grafana Drill-down . . . 35

7.5.1 Critical KPI and Alerts . . . 35

8 Conclusion 37

Chapter 1

Introduction

1.1 Project Context

This project is part of the Software engineering module. It aims to implement a complete CI/CD pipeline for a Java Enterprise Edition (Spring Boot) application. The main objectives are:

- Automate the build, testing, and deployment processes.
- Ensure code quality through static analysis tools.
- Containerize and deploy the application using Docker and Kubernetes.
- Monitor application performance with Prometheus and Grafana.

1.2 Application Description

The project uses a Spring Boot-based application. Key details include:

- **Project Name:** Employees Management System
- **Main Features:** Employee CRUD operations, Role management, Dashboard analytics
- **Architecture:** Multi-tier architecture with frontend, backend, and database layers
- **Repository:** https://github.com/chaimaeddib2005/Employee_Management

Chapter 2

GitHub – Source Control

2.1 Repository Setup

We created a repository on GitHub (public/private depending on access). The general code structure is organized as follows:

- /backend — Backend source code
- /frontend — Frontend source code
- /kubernetes — Deployment YAML files
- Jenkinsfile — Pipeline configuration

2.2 Branching Strategy

We adopted a Git Flow strategy with the following branches:

- master — Production-ready code

2.3 Commit History

Commits follow a clear and descriptive convention. Merge strategies include pull requests with code reviews to ensure code quality and traceability.

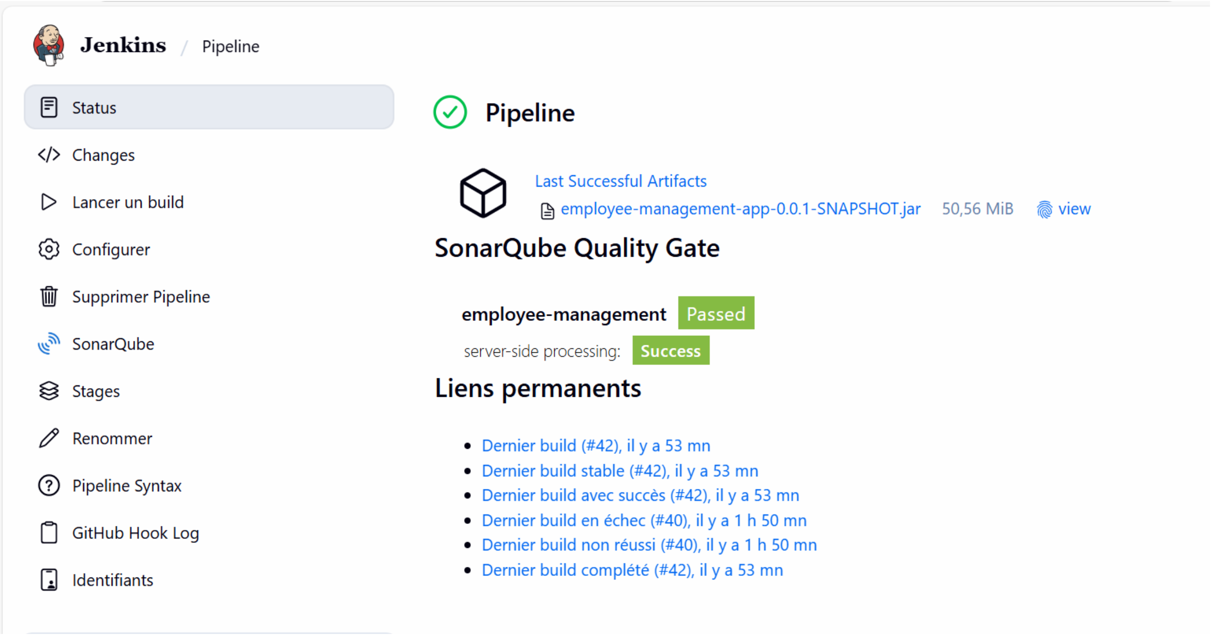
Chapter 3

Jenkins – Continuous Integration

3.1 Installation and Configuration

Jenkins was installed locally on a server with the necessary plugins for GitHub integration, Docker, Kubernetes ,SonarQube , Prometheus and Grafana .

3.2 CI Pipeline



Pipeline

Last Successful Artifacts

employee-management-app-0.0.1-SNAPSHOT.jar 50,56 MiB view

SonarQube Quality Gate

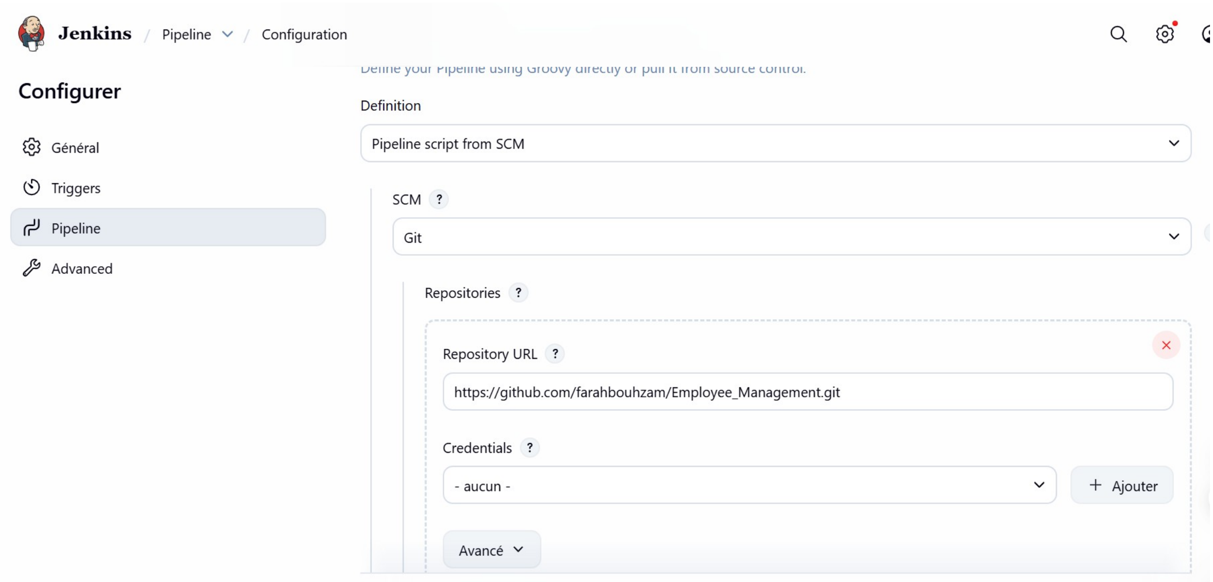
employee-management Passed

server-side processing: Success

Liens permanents

- Dernier build (#42), il y a 53 mn
- Dernier build stable (#42), il y a 53 mn
- Dernier build avec succès (#42), il y a 53 mn
- Dernier build en échec (#40), il y a 1 h 50 mn
- Dernier build non réussi (#40), il y a 1 h 50 mn
- Dernier build complété (#42), il y a 53 mn

Figure 3.1: Pipeline execution result with SonarQube Quality Gate status



Define your Pipeline using Groovy directly or pull it from source control.

Definition: Pipeline script from SCM

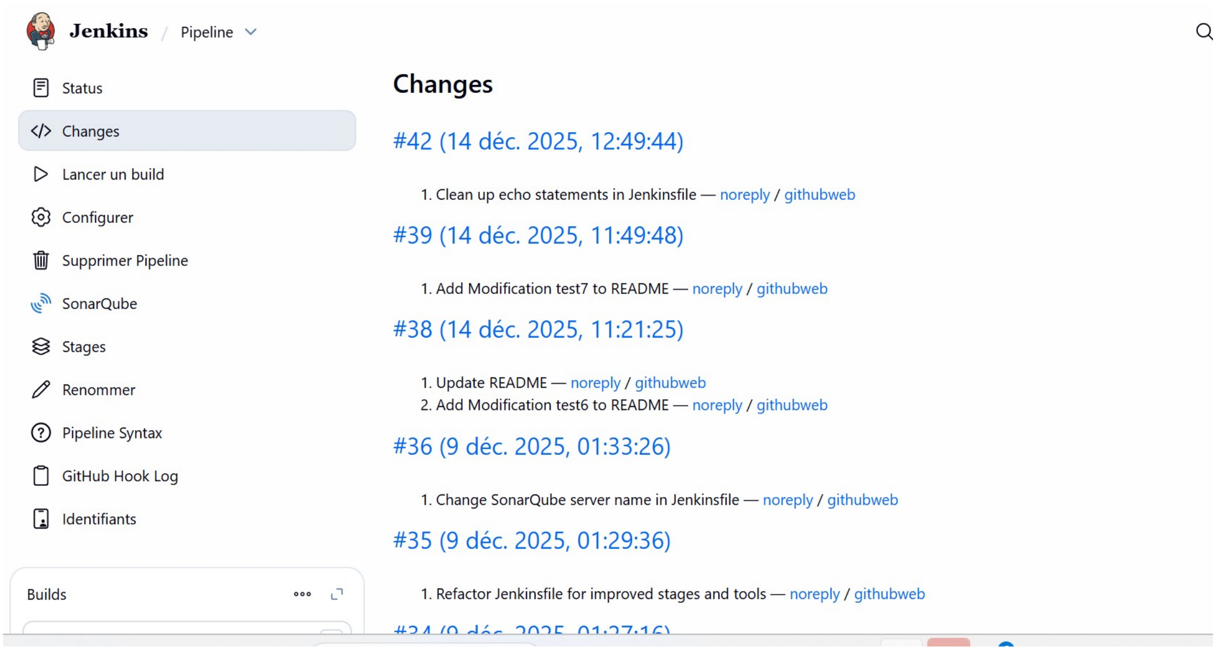
SCM: Git

Repositories

Repository URL: https://github.com/farahbouhzam/Employee_Management.git

Credentials: - aucun -

Figure 3.2: Pipeline configuration using Git SCM and Jenkinsfile from source control



Changes

#42 (14 déc. 2025, 12:49:44)

1. Clean up echo statements in Jenkinsfile — noreply / githubweb

#39 (14 déc. 2025, 11:49:48)

1. Add Modification test7 to README — noreply / githubweb

#38 (14 déc. 2025, 11:21:25)

1. Update README — noreply / githubweb
2. Add Modification test6 to README — noreply / githubweb

#36 (9 déc. 2025, 01:33:26)

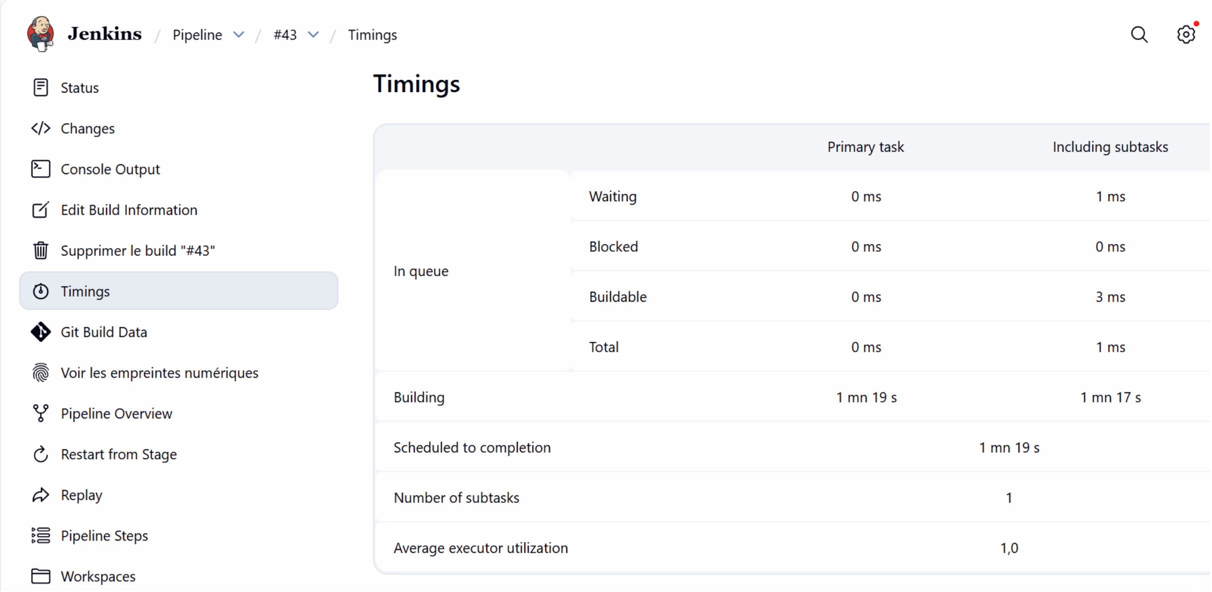
1. Change SonarQube server name in Jenkinsfile — noreply / githubweb

#35 (9 déc. 2025, 01:29:36)

1. Refactor Jenkinsfile for improved stages and tools — noreply / githubweb

#34 (9 déc. 2025, 01:27:16)

Figure 3.3: List of changes detected by Jenkins between pipeline executions



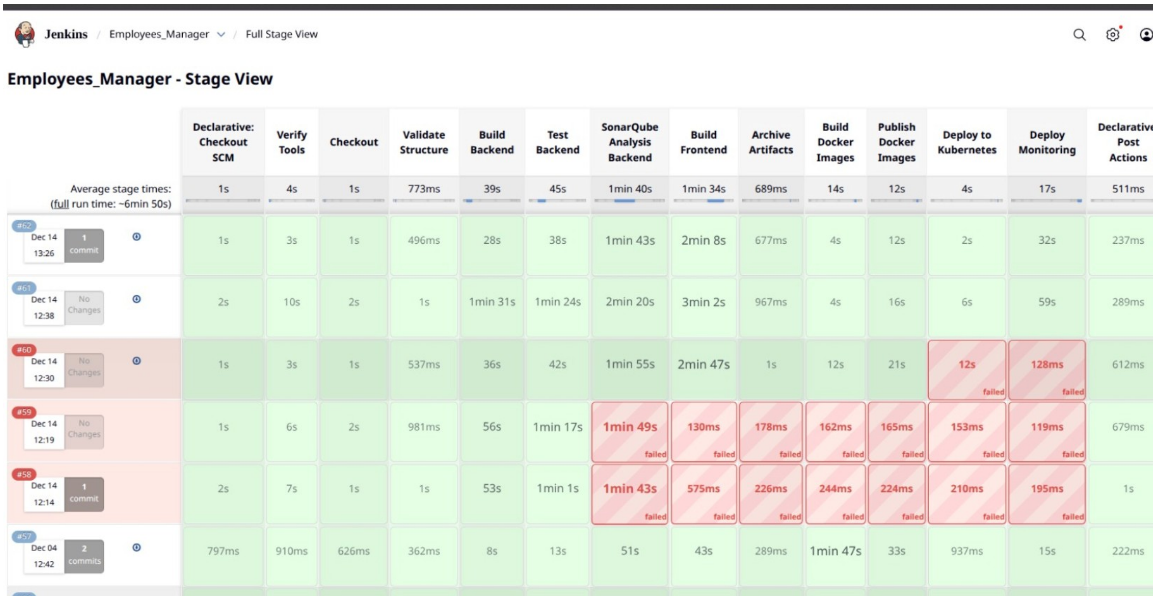
Timings

		Primary task	Including subtasks
In queue	Waiting	0 ms	1 ms
	Blocked	0 ms	0 ms
	Buildable	0 ms	3 ms
	Total	0 ms	1 ms
Building		1 mn 19 s	1 mn 17 s
Scheduled to completion		1 mn 19 s	
Number of subtasks		1	
Average executor utilization		1,0	

Figure 3.4: Detailed timing analysis of pipeline execution stages

3.2.1 Pipeline Execution and Results

- **Automatic trigger**
  The pipeline is automatically triggered on every push to the main branch using a GitHub webhook, enabling continuous integration.
- **Build results**
  The Jenkins console output shows:
  - **Successful compilation** – The project is compiled without errors.
  - **Tests passed** – All automated tests complete successfully.
  - **SonarQube analysis executed** – Quality Gate status:**Passed**.
  - **Docker image built and pushed** – The application image is published to the container registry.
  - **Deployment using Kubernetes** – The application is deployed as pods in a Kubernetes cluster.
  - **Monitoring using Prometheus and Grafana** – Metrics are collected and visualized through monitoring dashboards.



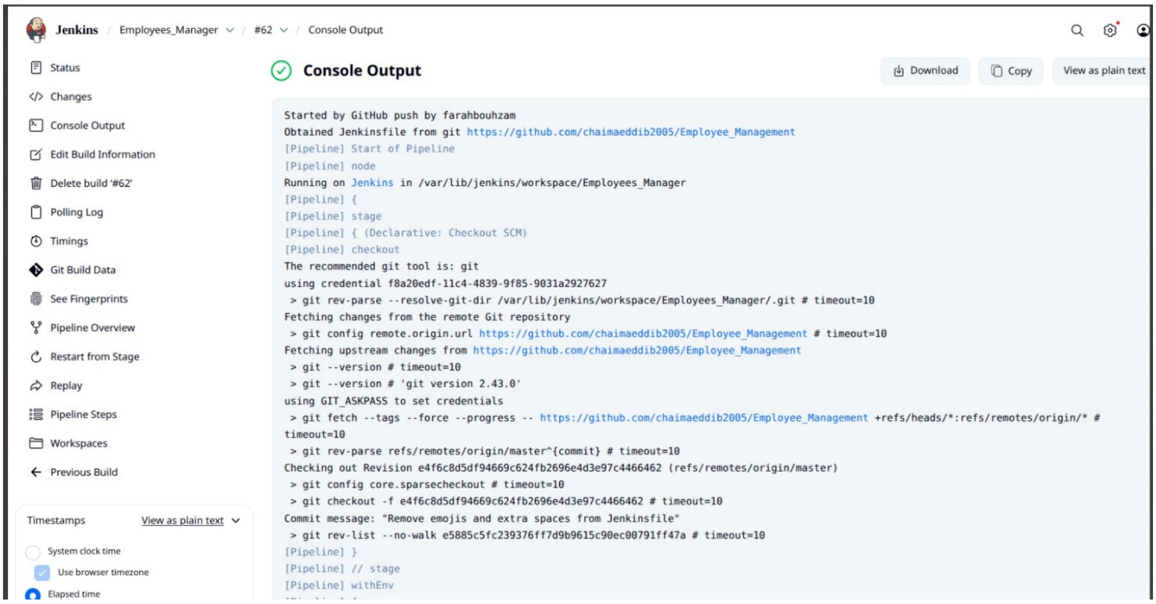
Employees_Manager - Stage View

	Declarative: Checkout SCM	Verify Tools	Checkout	Validate Structure	Build Backend	Test Backend	SonarQube Analysis Backend	Build Frontend	Archive Artifacts	Build Docker Images	Publish Docker Images	Deploy to Kubernetes	Deploy Monitoring	Declarative Post Actions
Average stage times: (full run time: ~6min 50s)	1s	4s	1s	773ms	39s	45s	1min 40s	1min 34s	689ms	14s	12s	4s	17s	511ms
#62 Dec 14 13:26 1 commit	1s	3s	1s	496ms	28s	38s	1min 43s	2min 8s	677ms	4s	12s	2s	32s	237ms
#61 Dec 14 12:38 No Changes	2s	10s	2s	1s	1min 31s	1min 24s	2min 20s	3min 2s	967ms	4s	16s	6s	59s	289ms
#60 Dec 14 12:30 No Changes	1s	3s	1s	537ms	36s	42s	1min 55s	2min 47s	1s	12s	21s	12s failed	128ms failed	612ms
#59 Dec 14 12:19 No Changes	1s	6s	2s	981ms	56s	1min 17s	1min 49s failed	130ms failed	178ms failed	162ms failed	165ms failed	153ms failed	119ms failed	679ms
#58 Dec 14 12:14 1 commit	2s	7s	1s	1s	53s	1min 1s	1min 43s failed	575ms failed	226ms failed	244ms failed	224ms failed	210ms failed	195ms failed	1s
#57 Dec 04 12:42 2 commits	797ms	910ms	626ms	362ms	8s	13s	51s	43s	289ms	1min 47s	33s	937ms	15s	222ms

Figure 3.5: Jenkins pipeline stage output — Java, Maven, and Node environment check

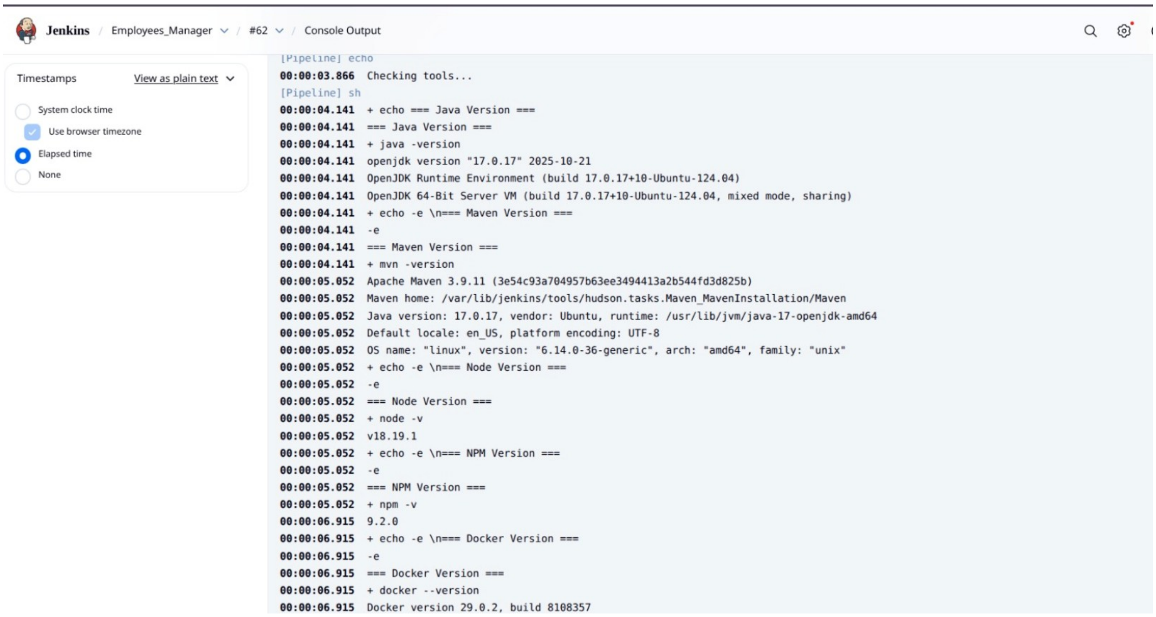
3.3 Build Logs

Build logs are captured in Jenkins to provide traceability and debugging information.



```
Started by GitHub push by farahbouhzam
Obtained Jenkinsfile from git https://github.com/chaimaeddib2005/Employee_Management
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in /var/lib/jenkins/workspace/Employees_Manager
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Declarative: Checkout SCM)
[Pipeline] checkout
The recommended git tool is: git
using credential f8a20edf-11c4-4839-9f85-9031a2927627
 > git rev-parse --resolve-git-dir /var/lib/jenkins/workspace/Employees_Manager/.git # timeout=10
Fetching changes from the remote Git repository
 > git config remote.origin.url https://github.com/chaimaeddib2005/Employee_Management # timeout=10
Fetching upstream changes from https://github.com/chaimaeddib2005/Employee_Management
 > git --version # timeout=10
 > git --version # 'git version 2.43.0'
using GIT_ASKPASS to set credentials
 > git fetch --tags --force --progress -- https://github.com/chaimaeddib2005/Employee_Management +refs/heads/*:refs/remotes/origin/* #
timeout=10
 > git rev-parse refs/remotes/origin/master^{commit} # timeout=10
Checking out Revision e4f6c8d5df94669c624fb2696e4d3e97c4466462 (refs/remotes/origin/master)
 > git config core.sparsecheckout # timeout=10
 > git checkout -f e4f6c8d5df94669c624fb2696e4d3e97c4466462 # timeout=10
Commit message: "Remove emojis and extra spaces from Jenkinsfile"
 > git rev-list --no-walk e5885c5fc239376ff7d9b9615c90ec00791ff47a # timeout=10
[Pipeline] }
[Pipeline] // stage
[Pipeline] withEnv
```

Figure 3.6: snippets of console .



○ System clock time

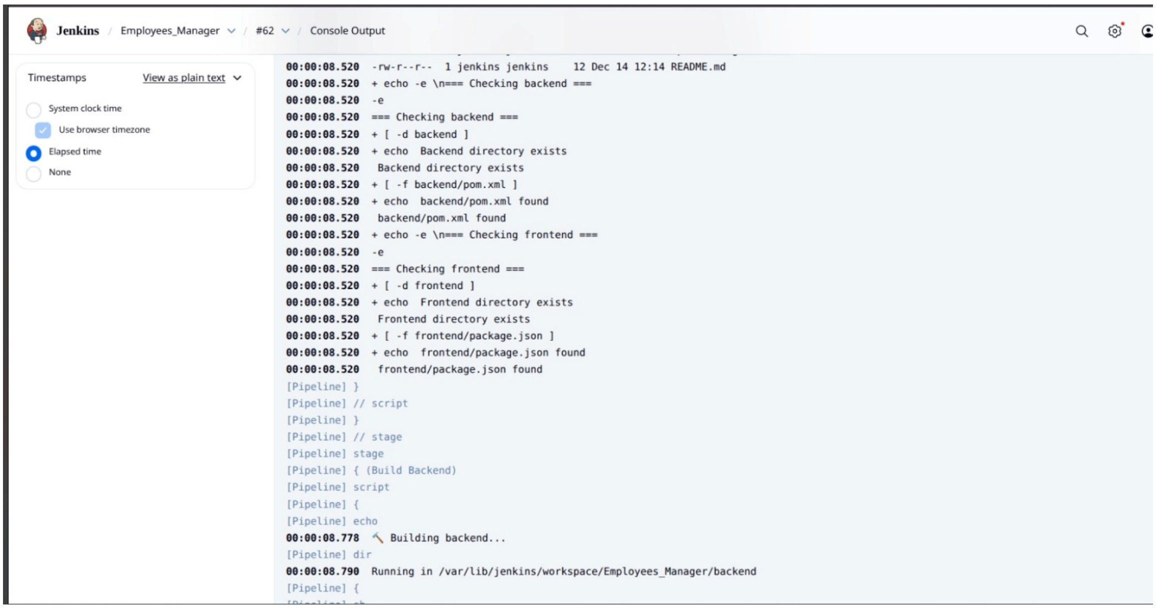
☑ Use browser timezone

● Elapsed time

○ None

```
[Pipeline] echo
00:00:03.866  Checking tools...
[Pipeline] sh
00:00:04.141  + echo === Java Version ===
00:00:04.141  === Java Version ===
00:00:04.141  + java -version
00:00:04.141  openjdk version "17.0.17" 2025-10-21
00:00:04.141  OpenJDK Runtime Environment (build 17.0.17+10-Ubuntu-124.04)
00:00:04.141  OpenJDK 64-Bit Server VM (build 17.0.17+10-Ubuntu-124.04, mixed mode, sharing)
00:00:04.141  + echo -e \n=== Maven Version ===
00:00:04.141  -e
00:00:04.141  === Maven Version ===
00:00:04.141  + mvn -version
00:00:05.052  Apache Maven 3.9.11 (3e54c93a704957b63ee3494413a2b544fd3d825b)
00:00:05.052  Maven home: /var/lib/jenkins/tools/hudson.tasks.Maven_MavenInstallation/Maven
00:00:05.052  Java version: 17.0.17, vendor: Ubuntu, runtime: /usr/lib/jvm/java-17-openjdk-amd64
00:00:05.052  Default locale: en_US, platform encoding: UTF-8
00:00:05.052  OS name: "linux", version: "6.14.0-36-generic", arch: "amd64", family: "unix"
00:00:05.052  + echo -e \n=== Node Version ===
00:00:05.052  -e
00:00:05.052  === Node Version ===
00:00:05.052  + node -v
00:00:05.052  v18.19.1
00:00:05.052  + echo -e \n=== NPM Version ===
00:00:05.052  -e
00:00:05.052  === NPM Version ===
00:00:05.052  + npm -v
00:00:06.915  9.2.0
00:00:06.915  + echo -e \n=== Docker Version ===
00:00:06.915  -e
00:00:06.915  === Docker Version ===
00:00:06.915  + docker --version
00:00:06.915  Docker version 29.0.2, build 8108357
```

Figure 3.7: snippets of console .



○ System clock time

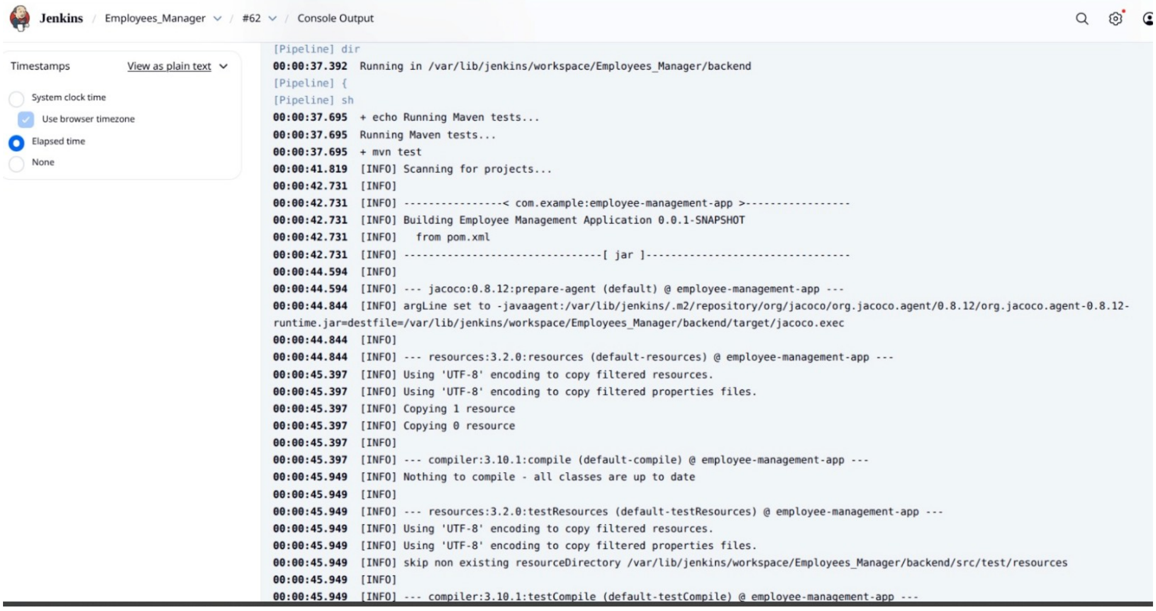
☑ Use browser timezone

● Elapsed time

○ None

```
00:00:08.520  -rw-r--r--  1 jenkins jenkins    12 Dec 14 12:14 README.md
00:00:08.520  + echo -e \n=== Checking backend ===
00:00:08.520  -e
00:00:08.520  === Checking backend ===
00:00:08.520  + [ -d backend ]
00:00:08.520  + echo  Backend directory exists
00:00:08.520   Backend directory exists
00:00:08.520  + [ -f backend/pom.xml ]
00:00:08.520  + echo  backend/pom.xml found
00:00:08.520   backend/pom.xml found
00:00:08.520  + echo -e \n=== Checking frontend ===
00:00:08.520  -e
00:00:08.520  === Checking frontend ===
00:00:08.520  + [ -d frontend ]
00:00:08.520  + echo  Frontend directory exists
00:00:08.520   Frontend directory exists
00:00:08.520  + [ -f frontend/package.json ]
00:00:08.520  + echo  frontend/package.json found
00:00:08.520   frontend/package.json found
[Pipeline] }
[Pipeline] // script
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Build Backend)
[Pipeline] script
[Pipeline] {
[Pipeline] echo
00:00:08.778  🔨 Building backend...
[Pipeline] dir
00:00:08.790  Running in /var/lib/jenkins/workspace/Employees_Manager/backend
[Pipeline] {
```

Figure 3.8: snippets of console .



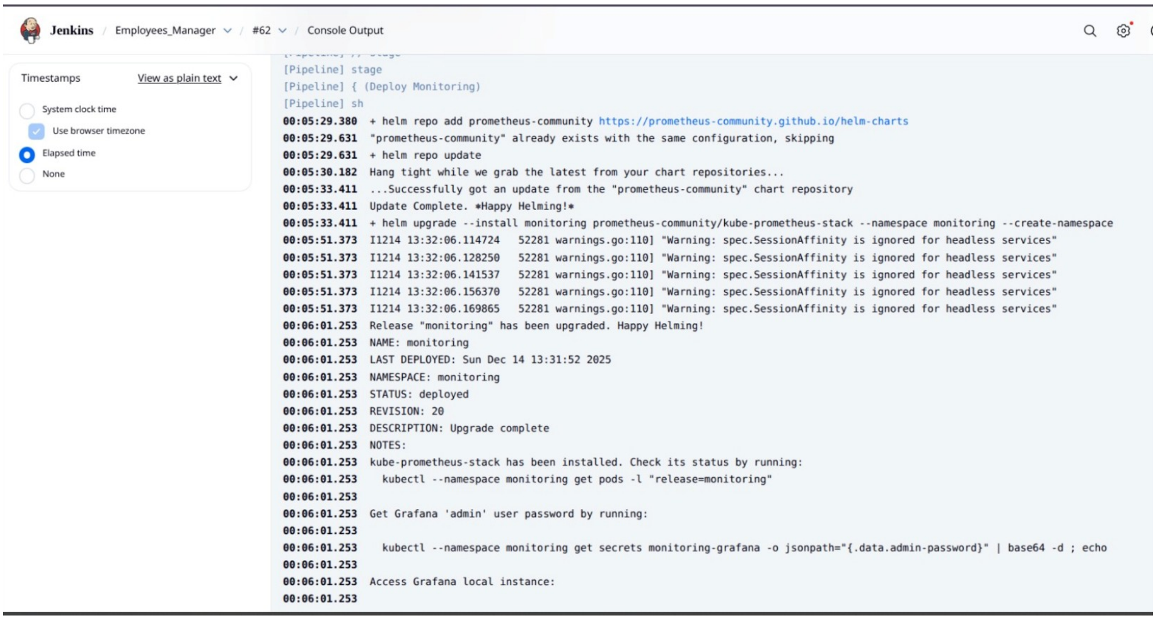
Jenkins / Employees_Manager / #62 / Console Output

Timestamps — View as plain text

○ System clock time
☑ Use browser timezone
● Elapsed time
○ None

```
[Pipeline] dir
00:00:37.392  Running in /var/lib/jenkins/workspace/Employees_Manager/backend
[Pipeline] {
[Pipeline] sh
00:00:37.695  + echo Running Maven tests...
00:00:37.695  Running Maven tests...
00:00:37.695  + mvn test
00:00:41.819  [INFO] Scanning for projects...
00:00:42.731  [INFO]
00:00:42.731  [INFO] ---------------< com.example:employee-management-app >----------------
00:00:42.731  [INFO] Building Employee Management Application 0.0.1-SNAPSHOT
00:00:42.731  [INFO]   from pom.xml
00:00:42.731  [INFO] --------------------------------[ jar ]---------------------------------
00:00:44.594  [INFO]
00:00:44.594  [INFO] --- jacoco:0.8.12:prepare-agent (default) @ employee-management-app ---
00:00:44.844  [INFO] argLine set to -javaagent:/var/lib/jenkins/.m2/repository/org/jacoco/org.jacoco.agent/0.8.12/org.jacoco.agent-0.8.12-
runtime.jar=destfile=/var/lib/jenkins/workspace/Employees_Manager/backend/target/jacoco.exec
00:00:44.844  [INFO]
00:00:44.844  [INFO] --- resources:3.2.0:resources (default-resources) @ employee-management-app ---
00:00:45.397  [INFO] Using 'UTF-8' encoding to copy filtered resources.
00:00:45.397  [INFO] Using 'UTF-8' encoding to copy filtered properties files.
00:00:45.397  [INFO] Copying 1 resource
00:00:45.397  [INFO] Copying 0 resource
00:00:45.397  [INFO]
00:00:45.397  [INFO] --- compiler:3.10.1:compile (default-compile) @ employee-management-app ---
00:00:45.949  [INFO] Nothing to compile - all classes are up to date
00:00:45.949  [INFO]
00:00:45.949  [INFO] --- resources:3.2.0:testResources (default-testResources) @ employee-management-app ---
00:00:45.949  [INFO] Using 'UTF-8' encoding to copy filtered resources.
00:00:45.949  [INFO] Using 'UTF-8' encoding to copy filtered properties files.
00:00:45.949  [INFO] skip non existing resourceDirectory /var/lib/jenkins/workspace/Employees_Manager/backend/src/test/resources
00:00:45.949  [INFO]
00:00:45.949  [INFO] --- compiler:3.10.1:testCompile (default-testCompile) @ employee-management-app ---
```

Figure 3.9: snippets of console .



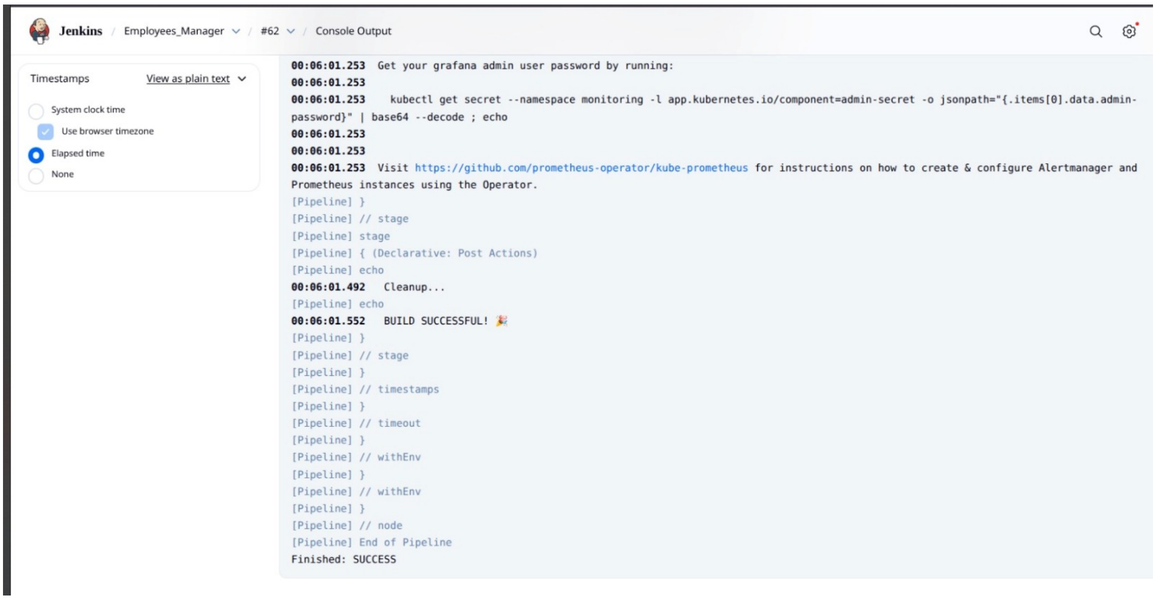
Jenkins / Employees_Manager / #62 / Console Output

Timestamps — View as plain text

○ System clock time
☑ Use browser timezone
● Elapsed time
○ None

```
[Pipeline] stage
[Pipeline] { (Deploy Monitoring)
[Pipeline] sh
00:05:29.380  + helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
00:05:29.631  "prometheus-community" already exists with the same configuration, skipping
00:05:29.631  + helm repo update
00:05:30.182  Hang tight while we grab the latest from your chart repositories...
00:05:33.411  ...Successfully got an update from the "prometheus-community" chart repository
00:05:33.411  Update Complete. *Happy Helming!*
00:05:33.411  + helm upgrade --install monitoring prometheus-community/kube-prometheus-stack --namespace monitoring --create-namespace
00:05:51.373  I1214 13:32:06.114724   52281 warnings.go:110] "Warning: spec.SessionAffinity is ignored for headless services"
00:05:51.373  I1214 13:32:06.128250   52281 warnings.go:110] "Warning: spec.SessionAffinity is ignored for headless services"
00:05:51.373  I1214 13:32:06.141537   52281 warnings.go:110] "Warning: spec.SessionAffinity is ignored for headless services"
00:05:51.373  I1214 13:32:06.156370   52281 warnings.go:110] "Warning: spec.SessionAffinity is ignored for headless services"
00:05:51.373  I1214 13:32:06.169865   52281 warnings.go:110] "Warning: spec.SessionAffinity is ignored for headless services"
00:06:01.253  Release "monitoring" has been upgraded. Happy Helming!
00:06:01.253  NAME: monitoring
00:06:01.253  LAST DEPLOYED: Sun Dec 14 13:31:52 2025
00:06:01.253  NAMESPACE: monitoring
00:06:01.253  STATUS: deployed
00:06:01.253  REVISION: 20
00:06:01.253  DESCRIPTION: Upgrade complete
00:06:01.253  NOTES:
00:06:01.253  kube-prometheus-stack has been installed. Check its status by running:
00:06:01.253    kubectl --namespace monitoring get pods -l "release=monitoring"
00:06:01.253
00:06:01.253  Get Grafana 'admin' user password by running:
00:06:01.253
00:06:01.253    kubectl --namespace monitoring get secrets monitoring-grafana -o jsonpath="{.data.admin-password}" | base64 -d ; echo
00:06:01.253
00:06:01.253  Access Grafana local instance:
00:06:01.253
```

Figure 3.10: snippets of console .



○ System clock time
☑ Use browser timezone
● Elapsed time
○ None

```
00:06:01.253  Get your grafana admin user password by running:
00:06:01.253
00:06:01.253    kubectl get secret --namespace monitoring -l app.kubernetes.io/component=admin-secret -o jsonpath="{.items[0].data.admin-
password}" | base64 --decode ; echo
00:06:01.253
00:06:01.253
00:06:01.253  Visit https://github.com/prometheus-operator/kube-prometheus for instructions on how to create & configure Alertmanager and
Prometheus instances using the Operator.
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Declarative: Post Actions)
[Pipeline] echo
00:06:01.492   Cleanup...
[Pipeline] echo
00:06:01.552   BUILD SUCCESSFUL! 🎉
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // timestamps
[Pipeline] }
[Pipeline] // timeout
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

Figure 3.11: snippets of console .

Chapter 4

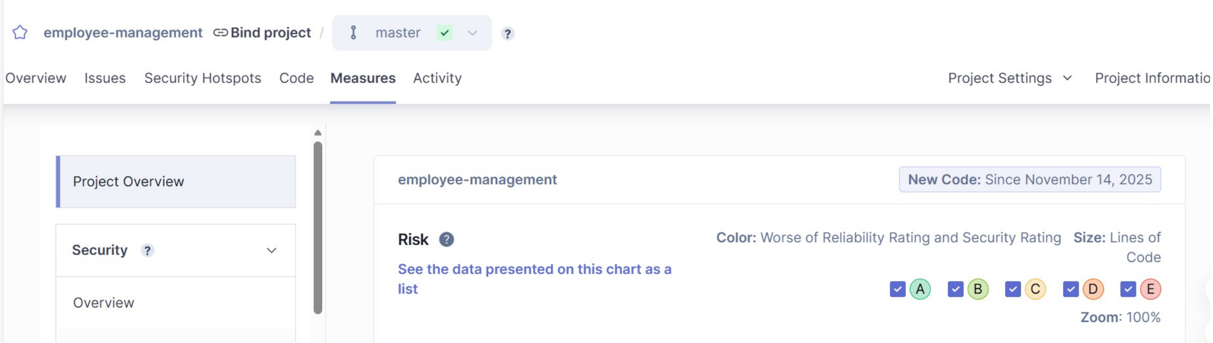
SonarQube – Code Quality

4.1 Installation

SonarQube server was installed and configured to analyze both frontend and backend projects.

4.2 Project Analysis

The dashboard provides metrics such as code coverage, bugs, vulnerabilities, and code smells.



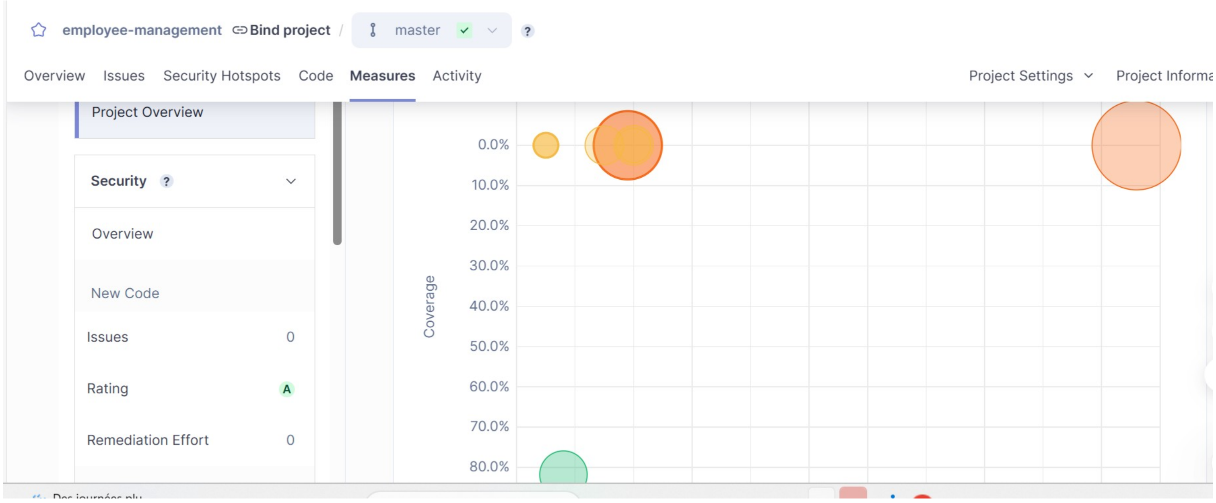


Figure 4.1: Detailed measures visualization .

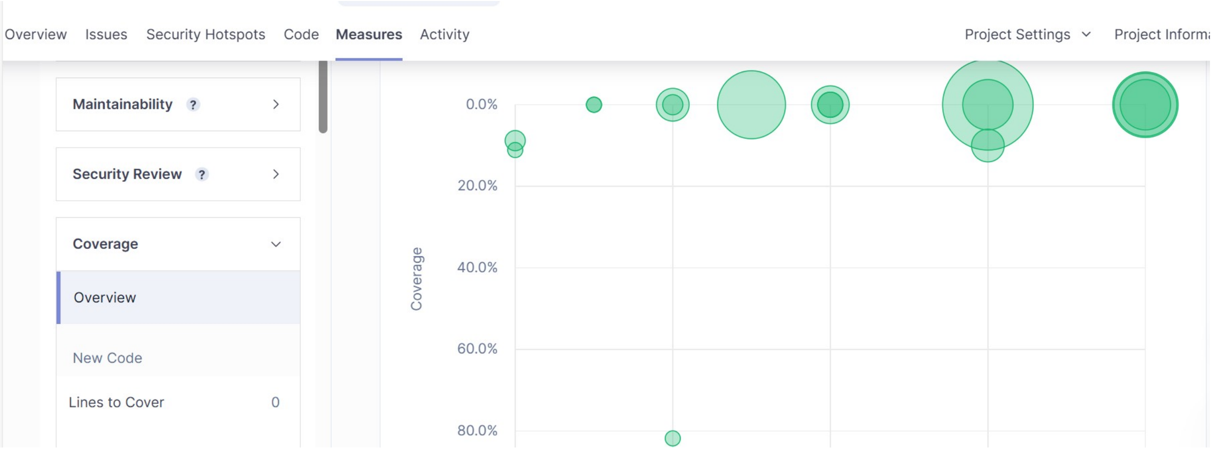
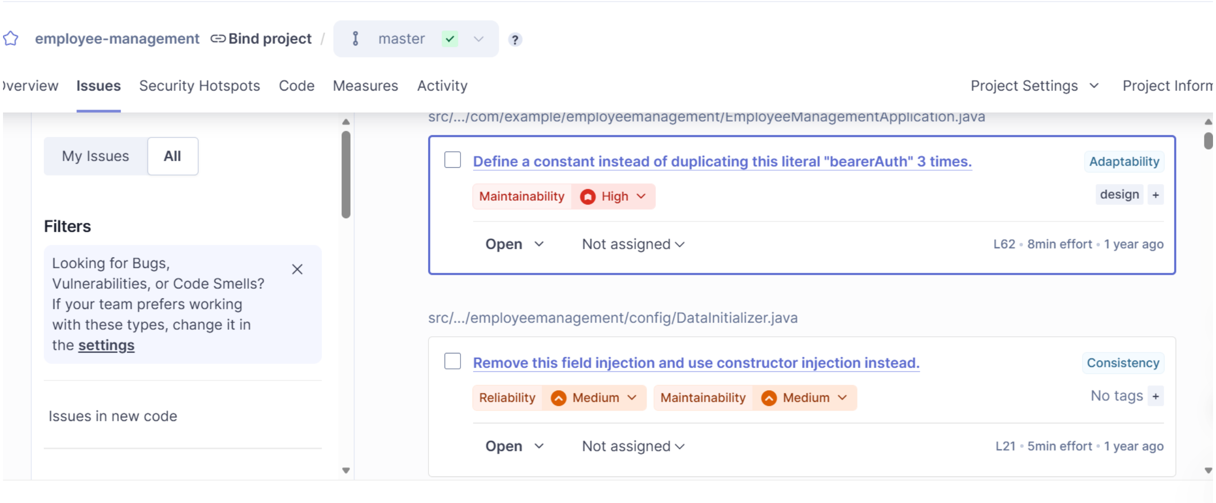


Figure 4.2: Maintainability, security review, and code coverage metrics visualized through SonarQube measure charts.



Looking for Bugs, Vulnerabilities, or Code Smells? If your team prefers working with these types, change it in the settings

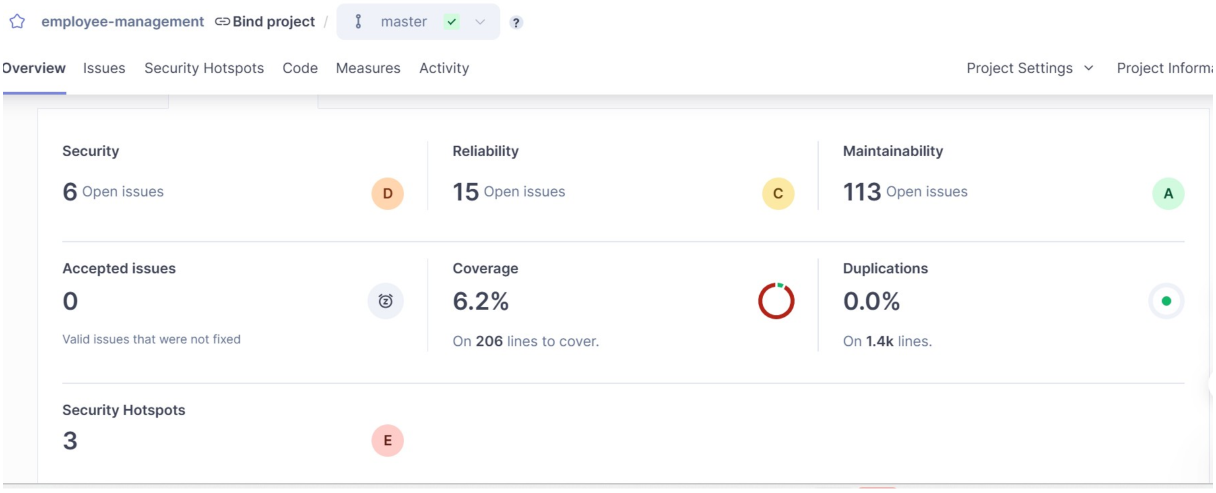
src/.../com/example/employeemanagement/EmployeeManagementApplication.java

□ Define a constant instead of duplicating this literal "bearerAuth" 3 times. Adaptability
Maintainability High design +
Open Not assigned L62 • 8min effort • 1 year ago

src/.../employeemanagement/config/DataInitializer.java

□ Remove this field injection and use constructor injection instead. Consistency
Reliability Medium Maintainability Medium No tags +
Open Not assigned L21 • 5min effort • 1 year ago

Figure 4.3: Examples of detected issues in SonarQube, including maintainability and reliability concerns with assigned severity levels.



employee-management ⊂⊃Bind project / master ✓ ?

Overview Issues Security Hotspots Code Measures Activity Project Settings Project Informa

Security
6 Open issues D

Reliability
15 Open issues C

Maintainability
113 Open issues A

Accepted issues
0
Valid issues that were not fixed

Coverage
6.2%
On 206 lines to cover.

Duplications
0.0%
On 1.4k lines.

Security Hotspots
3 E

Figure 4.4: Global SonarQube summary displaying security, reliability, maintainability ratings, test coverage, and duplication metrics.

4.2.1 Impact of JaCoCo Integration

- **Before JaCoCo integration**
  - Test coverage was **0%**, meaning no coverage metrics were detected.
- **After JaCoCo integration**
  - The JaCoCo plugin was integrated to enable test coverage computation in SonarQube.
  - Test coverage increased from **0% to 6.2%**, confirming that unit tests are now properly considered.
  - The analysis now provides better visibility into test quality and code reliability.

4.3 Issues Resolution

Type	Description	Correction Action
Code Smells	Unused imports detected by SonarQube	Removal of unnecessary imports to reduce code noise and improve readability
Code Smells	Use of field injection (@Autowired on attribute)	Replacing field injection with constructor injection to improve reliability and testability
Code Smells	Use of wildcard generic types (<?>)	Replacing general generic types with explicitly defined types to strengthen type safety

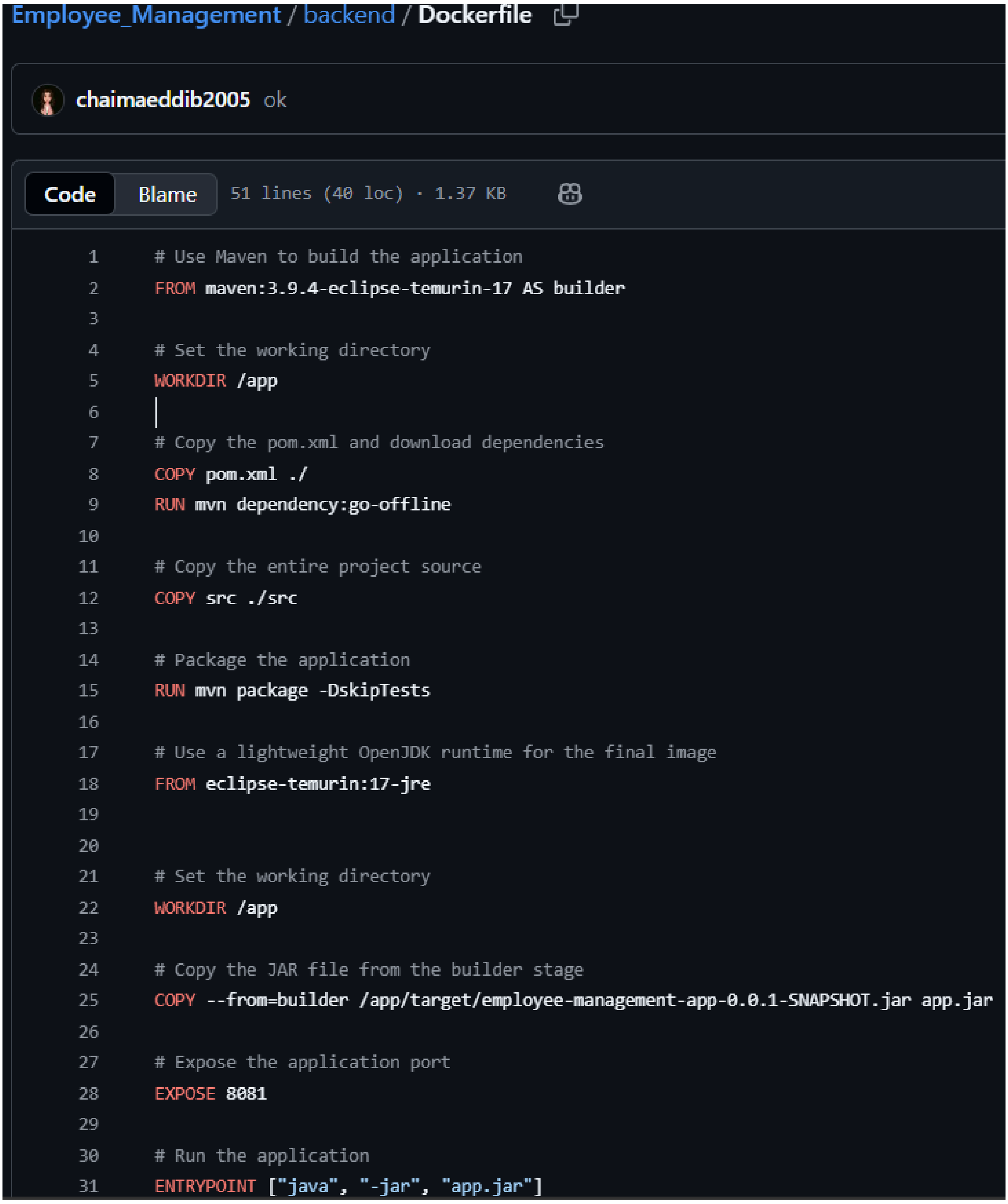
Table 4.1: Main SonarQube issues fixed after refactoring

Chapter 5

Docker – Containerization

After building and testing the backend and frontend, the applications are containerized using Docker.We created three Dockerfiles: one for the backend, one for the frontend, and a Docker Compose file to orchestrate all services.

5.1 Backend Dockerfile

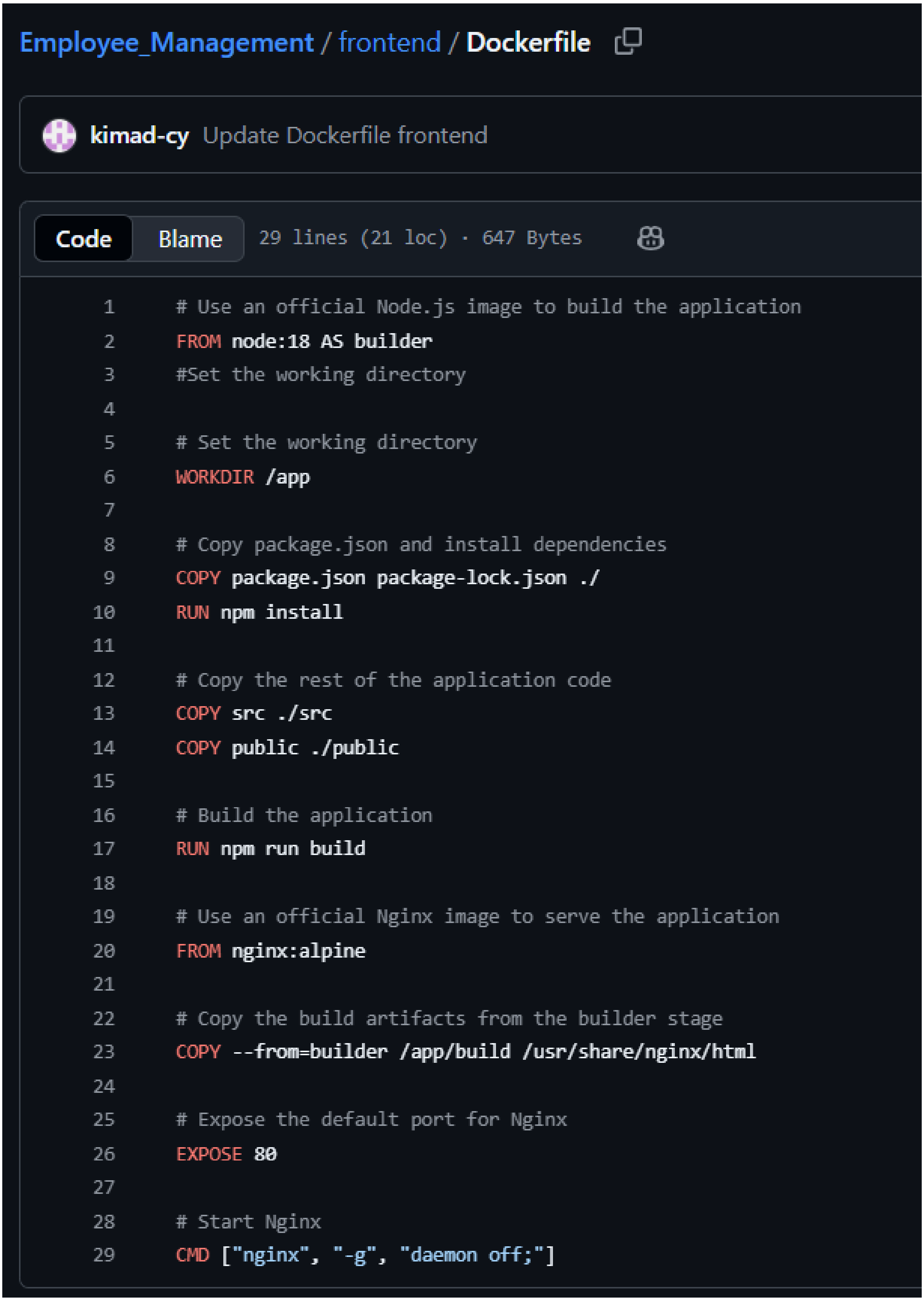


Employee_Management / backend / Dockerfile
chaimaeddib2005 ok
Code
Blame
51 lines (40 loc) · 1.37 KB

```
# Use Maven to build the application
FROM maven:3.9.4-eclipse-temurin-17 AS builder

# Set the working directory
WORKDIR /app

# Copy the pom.xml and download dependencies
COPY pom.xml ./
RUN mvn dependency:go-offline

# Copy the entire project source
COPY src ./src

# Package the application
RUN mvn package -DskipTests

# Use a lightweight OpenJDK runtime for the final image
FROM eclipse-temurin:17-jre


# Set the working directory
WORKDIR /app

# Copy the JAR file from the builder stage
COPY --from=builder /app/target/employee-management-app-0.0.1-SNAPSHOT.jar app.jar

# Expose the application port
EXPOSE 8081

# Run the application
ENTRYPOINT ["java", "-jar", "app.jar"]
```

Figure 5.1: Backend Dockerfile

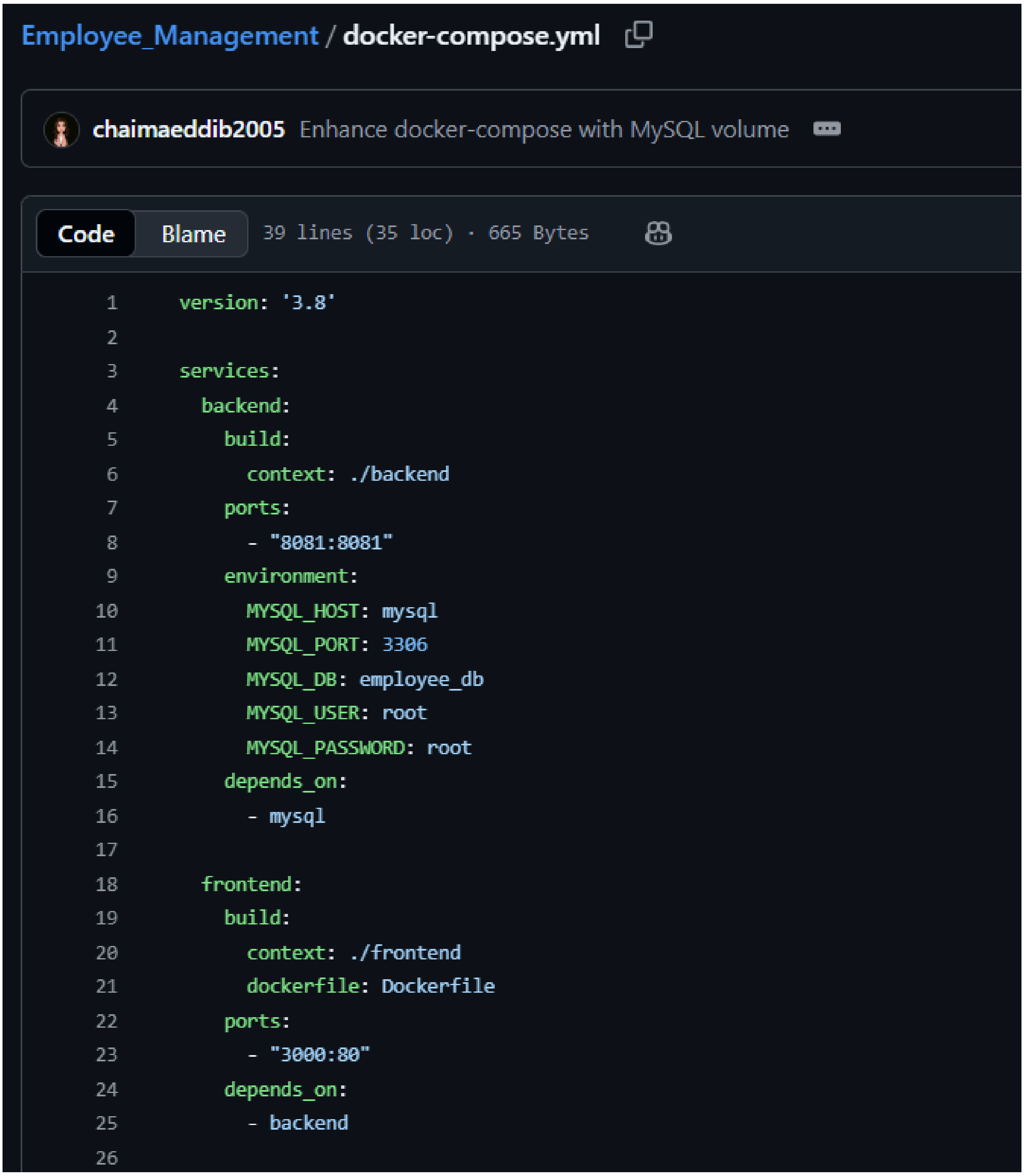
5.2 Frontend Dockerfile


Employee_Management / frontend / Dockerfile
kimad-cy Update Dockerfile frontend
Code
Blame
29 lines (21 loc) · 647 Bytes

```dockerfile
# Use an official Node.js image to build the application
FROM node:18 AS builder
#Set the working directory

# Set the working directory
WORKDIR /app

# Copy package.json and install dependencies
COPY package.json package-lock.json ./
RUN npm install

# Copy the rest of the application code
COPY src ./src
COPY public ./public

# Build the application
RUN npm run build

# Use an official Nginx image to serve the application
FROM nginx:alpine

# Copy the build artifacts from the builder stage
COPY --from=builder /app/build /usr/share/nginx/html

# Expose the default port for Nginx
EXPOSE 80

# Start Nginx
CMD ["nginx", "-g", "daemon off;"]
```

Figure 5.2: Frontend Dockerfile

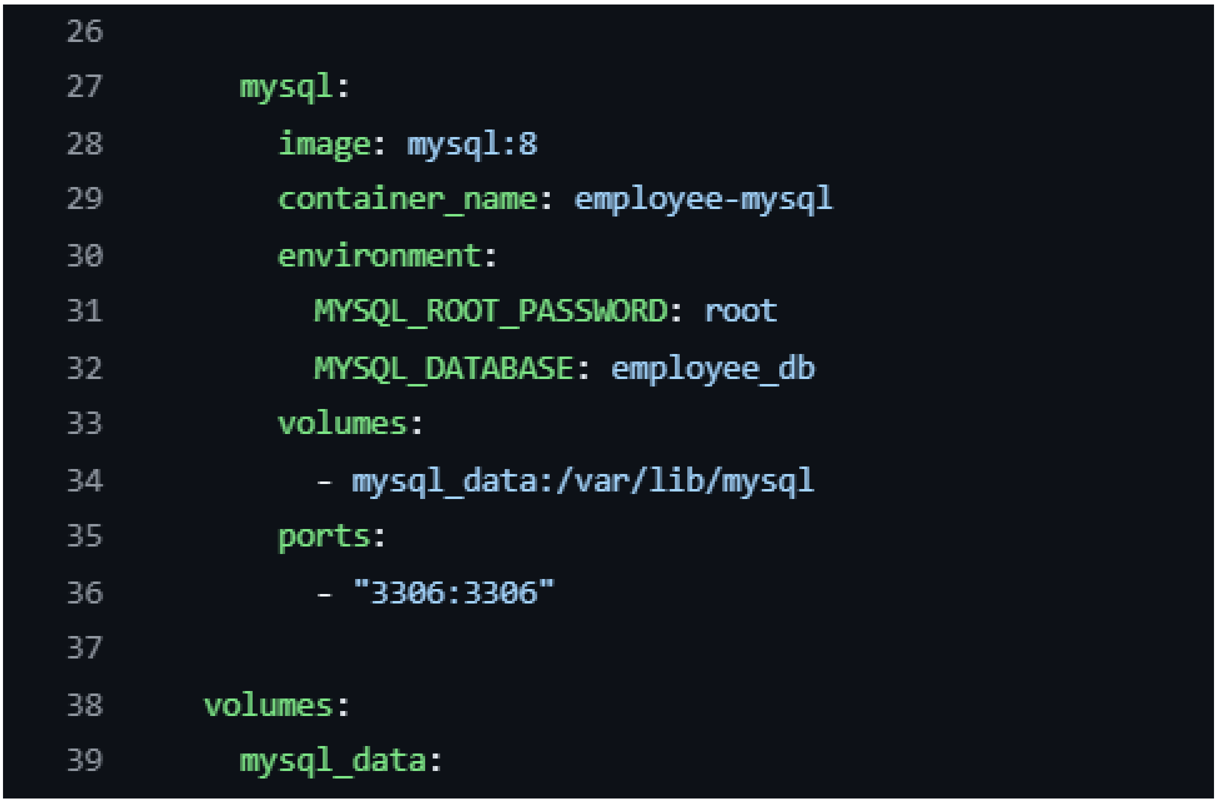
5.3 Docker Compose Configuration



Employee_Management / docker-compose.yml
chaimaeddib2005 Enhance docker-compose with MySQL volume
Code
Blame
39 lines (35 loc) · 665 Bytes

```yaml
version: '3.8'

services:
  backend:
    build:
      context: ./backend
    ports:
      - "8081:8081"
    environment:
      MYSQL_HOST: mysql
      MYSQL_PORT: 3306
      MYSQL_DB: employee_db
      MYSQL_USER: root
      MYSQL_PASSWORD: root
    depends_on:
      - mysql

  frontend:
    build:
      context: ./frontend
      dockerfile: Dockerfile
    ports:
      - "3000:80"
    depends_on:
      - backend

```

Figure 5.3: Docker Compose – Part 1



```yaml

  mysql:
    image: mysql:8
    container_name: employee-mysql
    environment:
      MYSQL_ROOT_PASSWORD: root
      MYSQL_DATABASE: employee_db
    volumes:
      - mysql_data:/var/lib/mysql
    ports:
      - "3306:3306"

volumes:
  mysql_data:
```

Figure 5.4: Docker Compose – Part 2

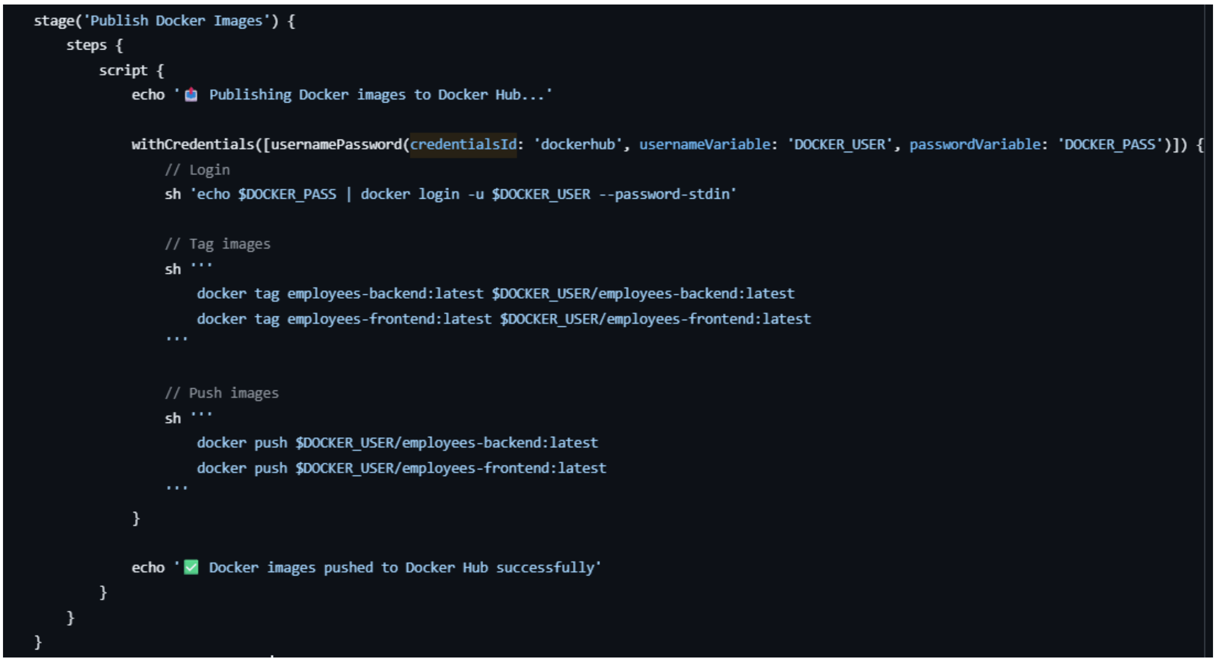
5.4 Building Docker Images

Commands used to build the images:

```
docker   build   -t   employees - backend : latest     ./ backend
docker   build   -t   employees - frontend : latest     ./ frontend
```

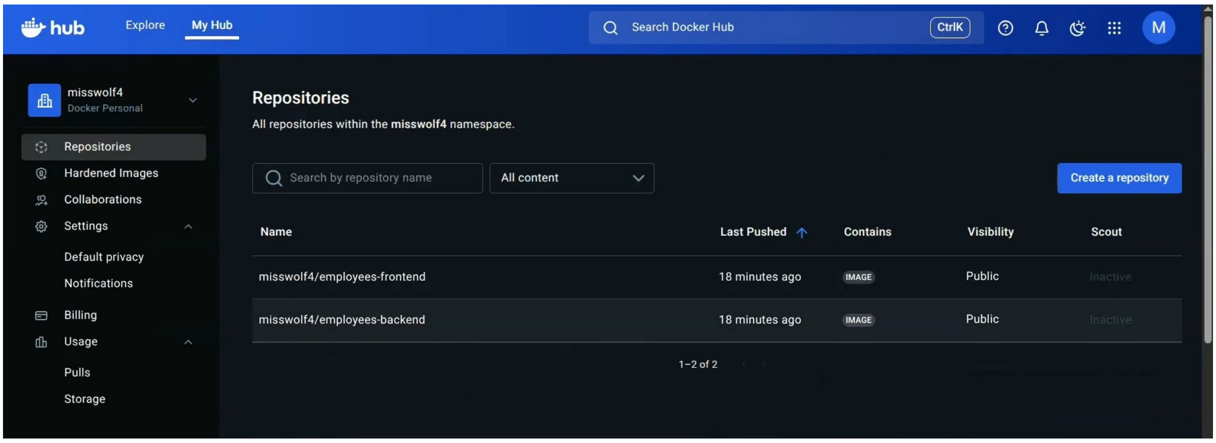
5.5 Publishing to DockerHub

Images are automatically pushed to DockerHub via Jenkins, using credentials securely stored in Jenkins.



```
stage('Publish Docker Images') {
    steps {
        script {
            echo '📤 Publishing Docker images to Docker Hub...'

            withCredentials([usernamePassword(credentialsId: 'dockerhub', usernameVariable: 'DOCKER_USER', passwordVariable: 'DOCKER_PASS')]) {
                // Login
                sh 'echo $DOCKER_PASS | docker login -u $DOCKER_USER --password-stdin'

                // Tag images
                sh '''
                    docker tag employees-backend:latest $DOCKER_USER/employees-backend:latest
                    docker tag employees-frontend:latest $DOCKER_USER/employees-frontend:latest
                '''

                // Push images
                sh '''
                    docker push $DOCKER_USER/employees-backend:latest
                    docker push $DOCKER_USER/employees-frontend:latest
                '''
            }

            echo '✅ Docker images pushed to Docker Hub successfully'
        }
    }
}
```

Figure 5.5: Jenkins stage for pushing Docker images to DockerHub



Repositories

All repositories within the **misswolf4** namespace.

Name	Last Pushed ↑	Contains	Visibility	Scout
misswolf4/employees-frontend	18 minutes ago	IMAGE	Public	Inactive
misswolf4/employees-backend	18 minutes ago	IMAGE	Public	Inactive

1–2 of 2

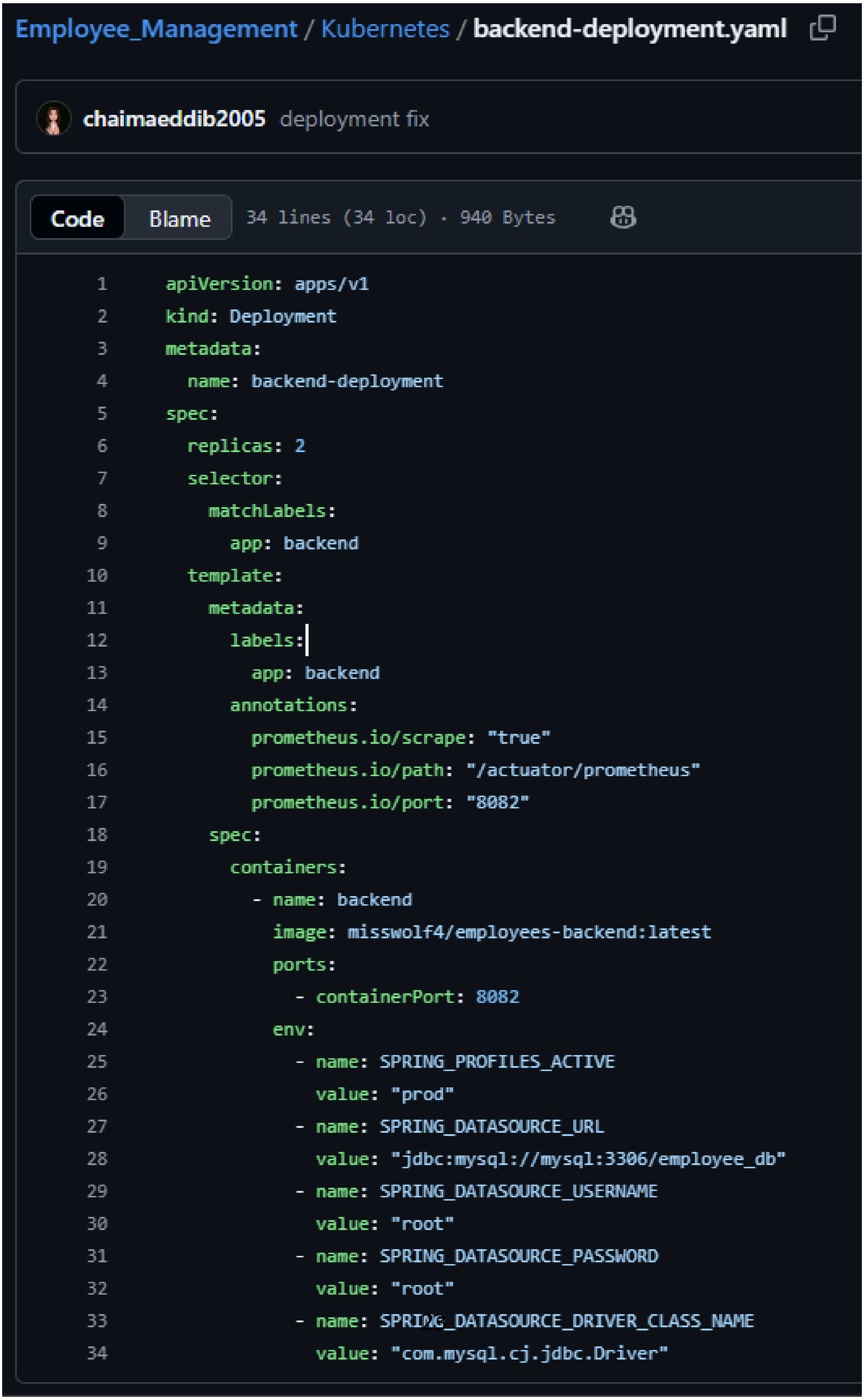
Figure 5.6: Published Docker image on DockerHub

Chapter 6

Kubernetes – Deployment

Kubernetes is used to deploy the backend, frontend, and MySQL database on Minikube. The YAML files define Deployments, Services, and Ingress.

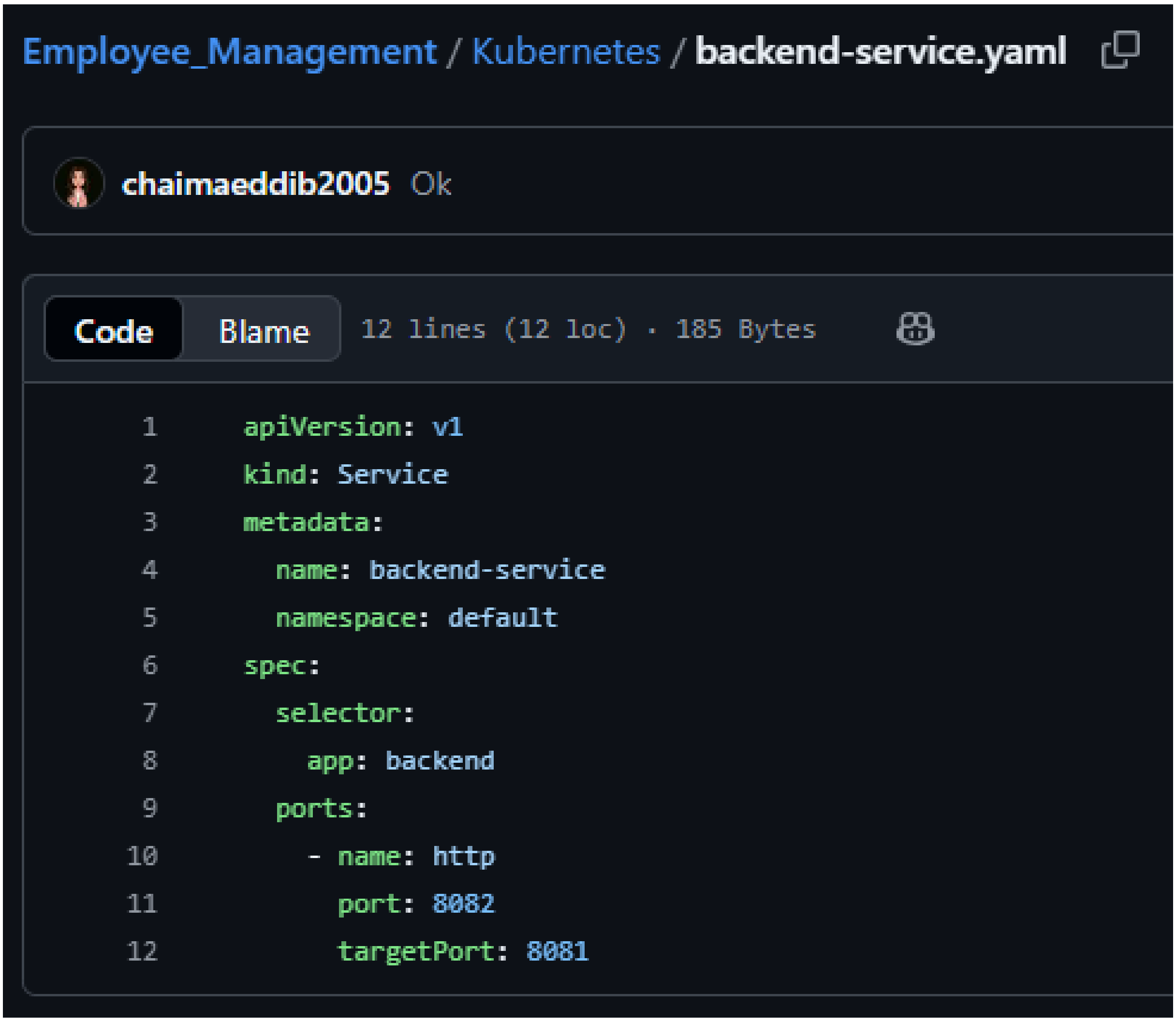
6.0.1 Backend Deployment



Employee_Management / Kubernetes / backend-deployment.yaml
chaimaeddib2005 deployment fix
Code
Blame
34 lines (34 loc) · 940 Bytes

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: backend-deployment
spec:
  replicas: 2
  selector:
    matchLabels:
      app: backend
  template:
    metadata:
      labels:
        app: backend
      annotations:
        prometheus.io/scrape: "true"
        prometheus.io/path: "/actuator/prometheus"
        prometheus.io/port: "8082"
    spec:
      containers:
        - name: backend
          image: misswolf4/employees-backend:latest
          ports:
            - containerPort: 8082
          env:
            - name: SPRING_PROFILES_ACTIVE
              value: "prod"
            - name: SPRING_DATASOURCE_URL
              value: "jdbc:mysql://mysql:3306/employee_db"
            - name: SPRING_DATASOURCE_USERNAME
              value: "root"
            - name: SPRING_DATASOURCE_PASSWORD
              value: "root"
            - name: SPRING_DATASOURCE_DRIVER_CLASS_NAME
              value: "com.mysql.cj.jdbc.Driver"
```

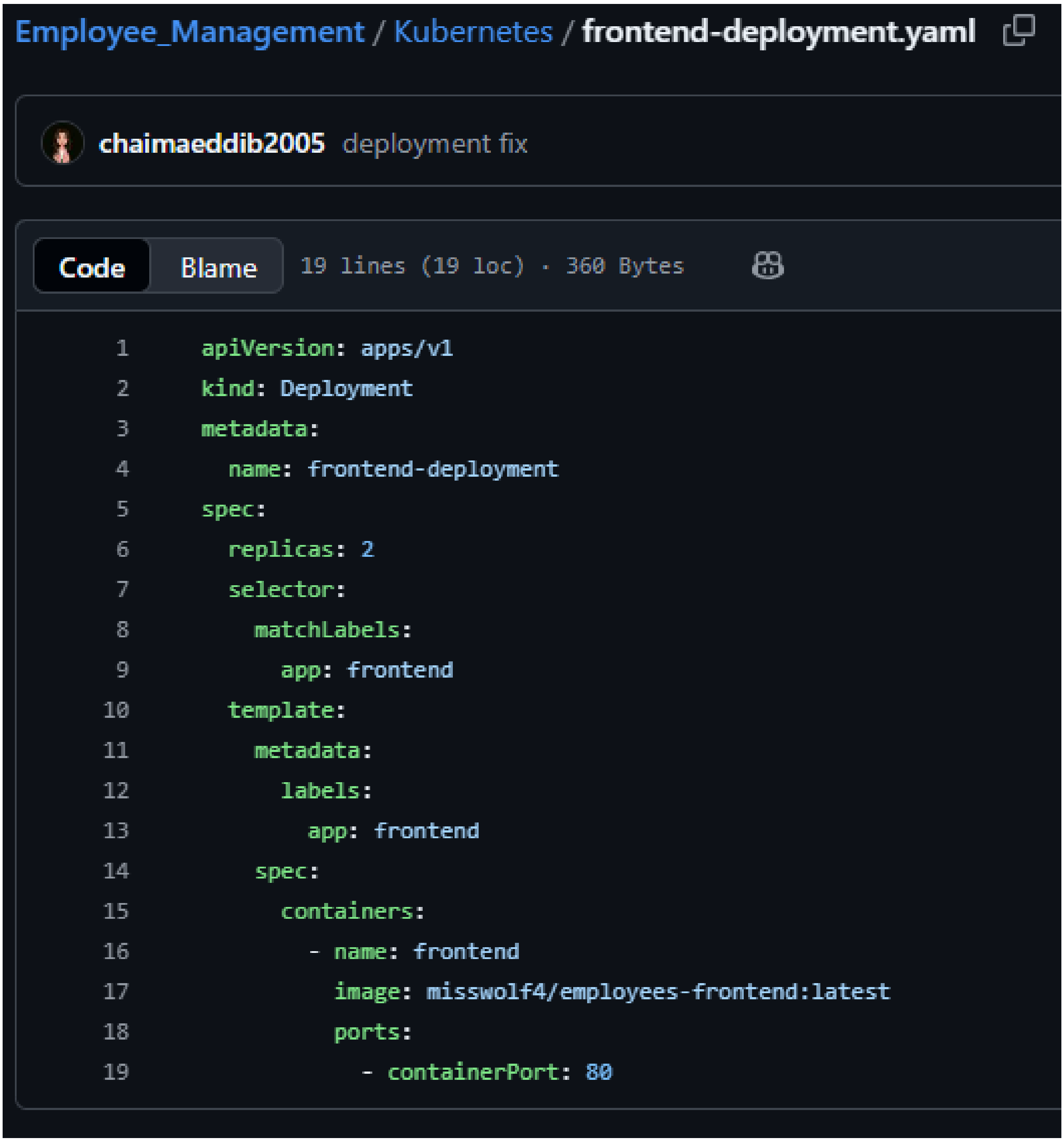
6.0.2 Backend Service


Employee_Management / Kubernetes / backend-service.yaml
chaimaeddib2005 Ok
Code
Blame
12 lines (12 loc) · 185 Bytes

```yaml
apiVersion: v1
kind: Service
metadata:
  name: backend-service
  namespace: default
spec:
  selector:
    app: backend
  ports:
    - name: http
      port: 8082
      targetPort: 8081
```

Figure 6.2: Backend Service YAML

6.0.3 Frontend Deployment

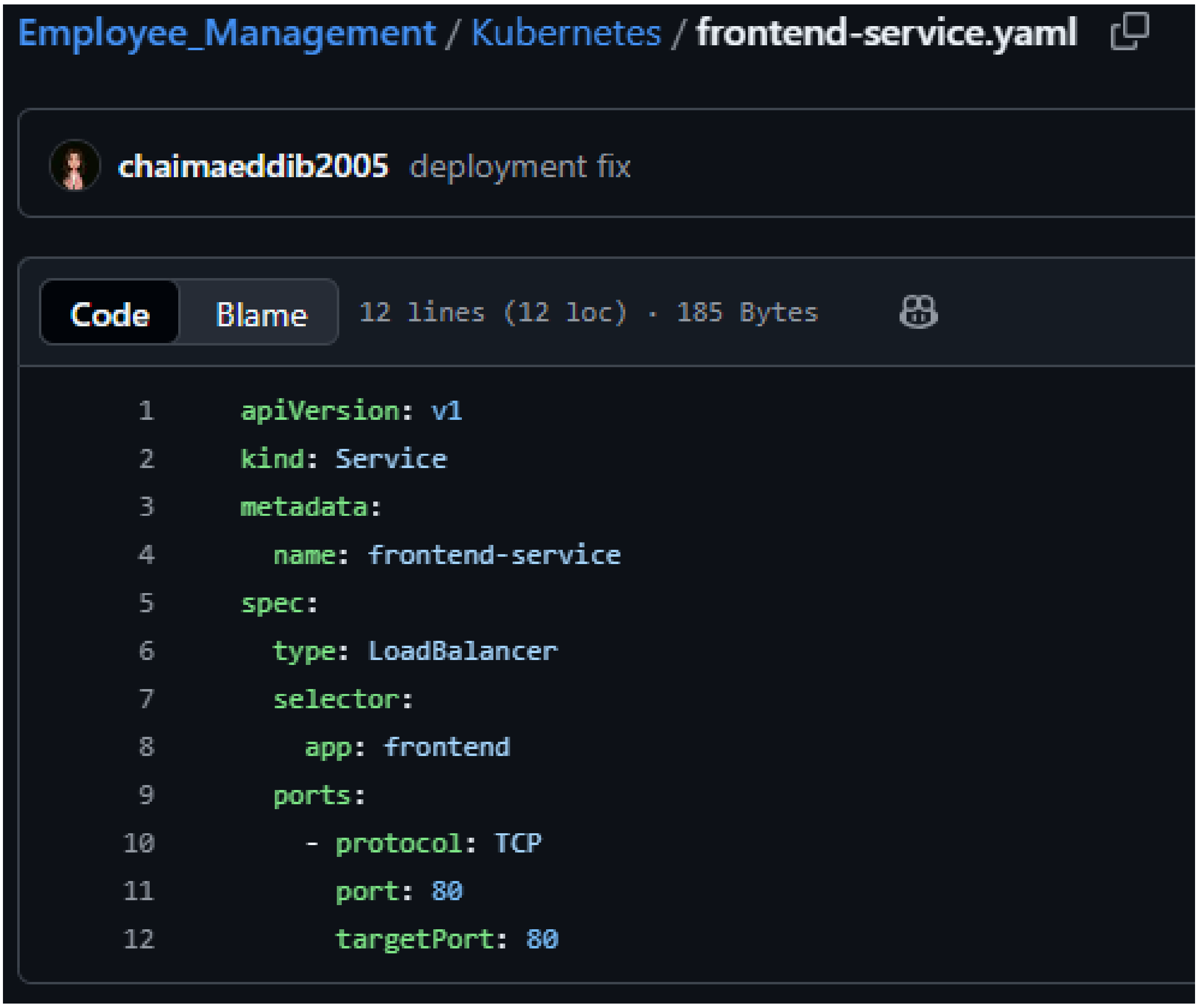


Employee_Management / Kubernetes / frontend-deployment.yaml
chaimaeddib2005 deployment fix
Code
Blame
19 lines (19 loc) · 360 Bytes

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: frontend-deployment
spec:
  replicas: 2
  selector:
    matchLabels:
      app: frontend
  template:
    metadata:
      labels:
        app: frontend
    spec:
      containers:
        - name: frontend
          image: misswolf4/employees-frontend:latest
          ports:
            - containerPort: 80
```

Figure 6.3: Frontend Deployment YAML

6.0.4 Frontend Service



Employee_Management / Kubernetes / frontend-service.yaml

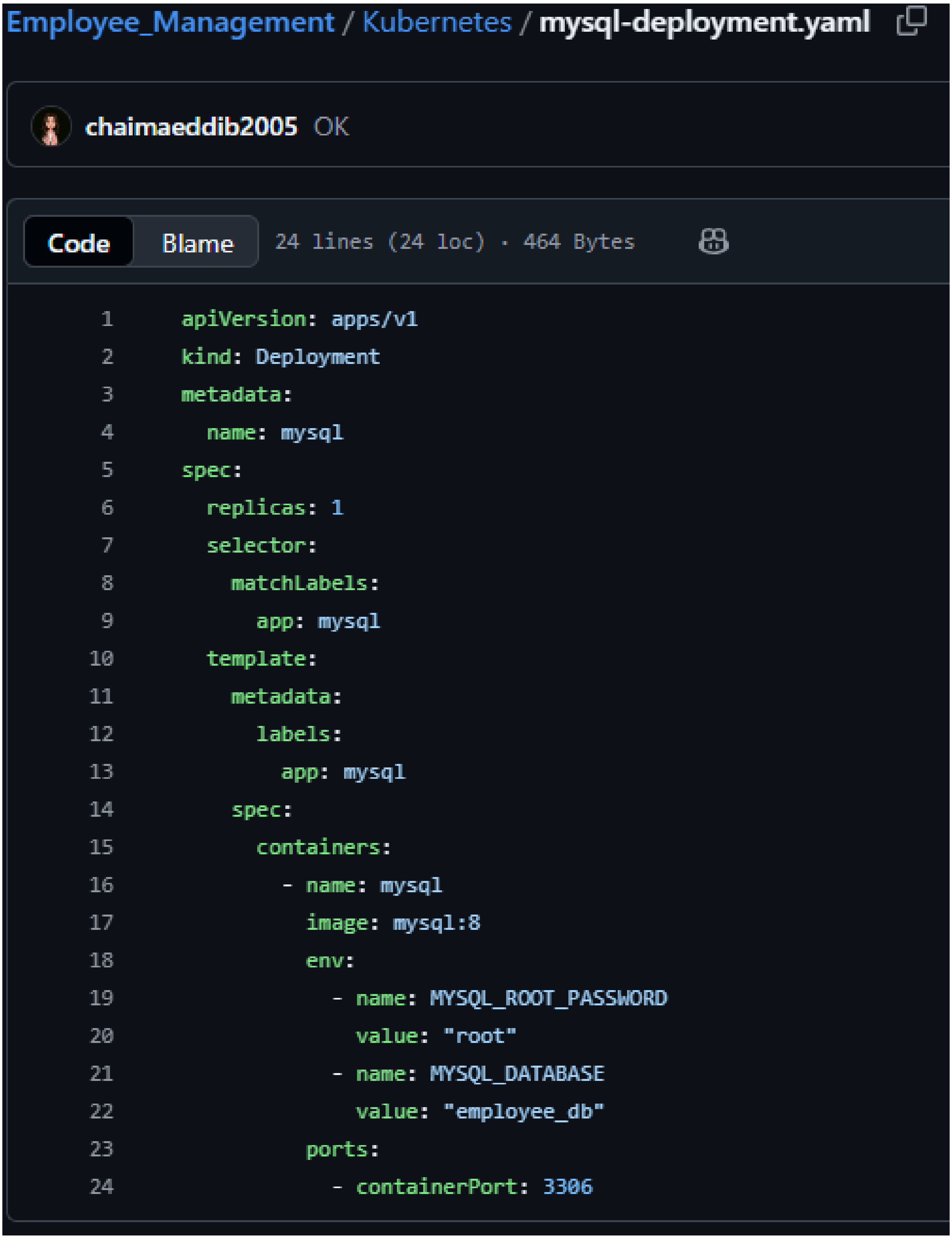
chaimaeddib2005 deployment fix

12 lines (12 loc) · 185 Bytes

```yaml
apiVersion: v1
kind: Service
metadata:
  name: frontend-service
spec:
  type: LoadBalancer
  selector:
    app: frontend
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
```

Figure 6.4: Frontend Service YAML

6.0.5 MySQL Deployment

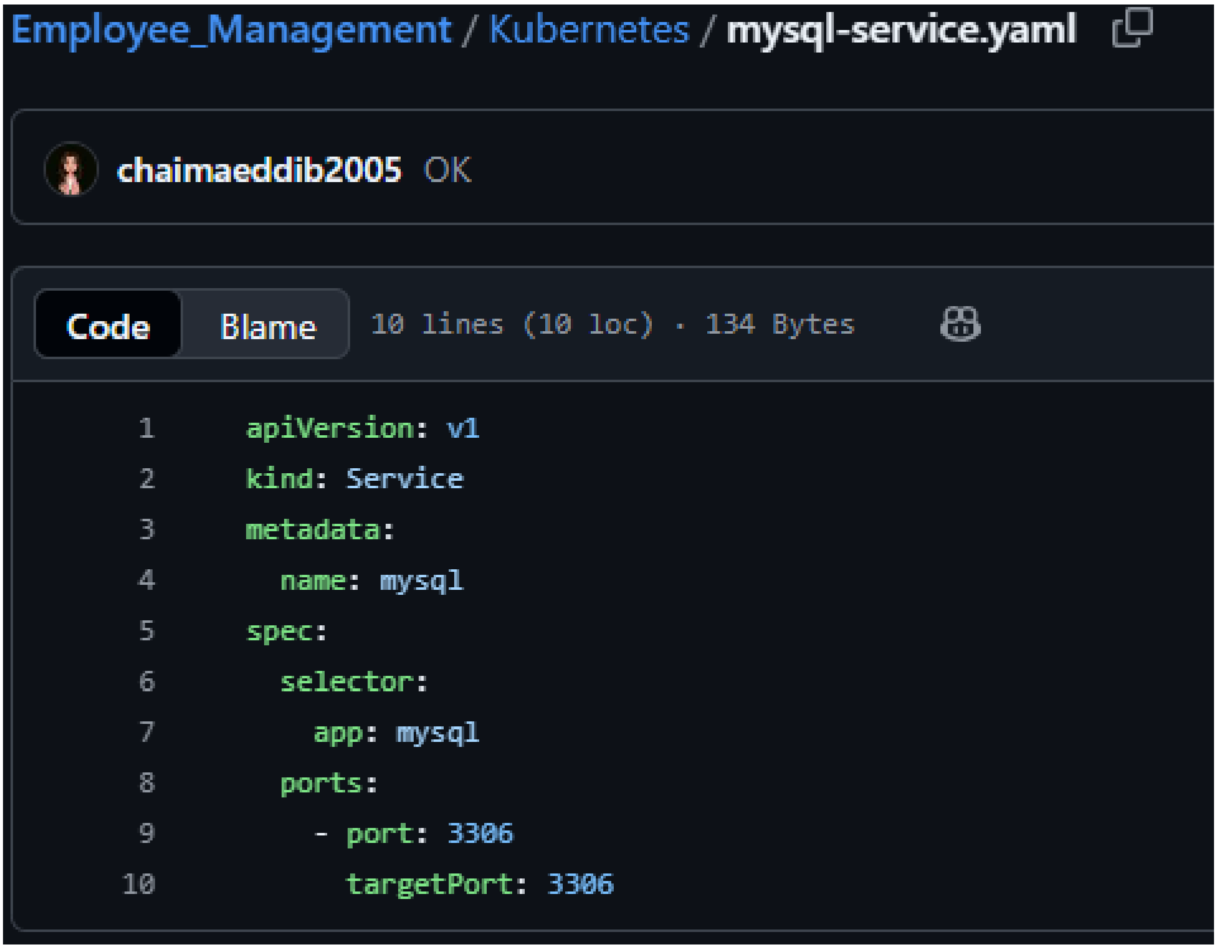


Employee_Management / Kubernetes / mysql-deployment.yaml
chaimaeddib2005 OK
Code
Blame
24 lines (24 loc) · 464 Bytes

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mysql
spec:
  replicas: 1
  selector:
    matchLabels:
      app: mysql
  template:
    metadata:
      labels:
        app: mysql
    spec:
      containers:
        - name: mysql
          image: mysql:8
          env:
            - name: MYSQL_ROOT_PASSWORD
              value: "root"
            - name: MYSQL_DATABASE
              value: "employee_db"
          ports:
            - containerPort: 3306
```

Figure 6.5: MySQL Deployment YAML

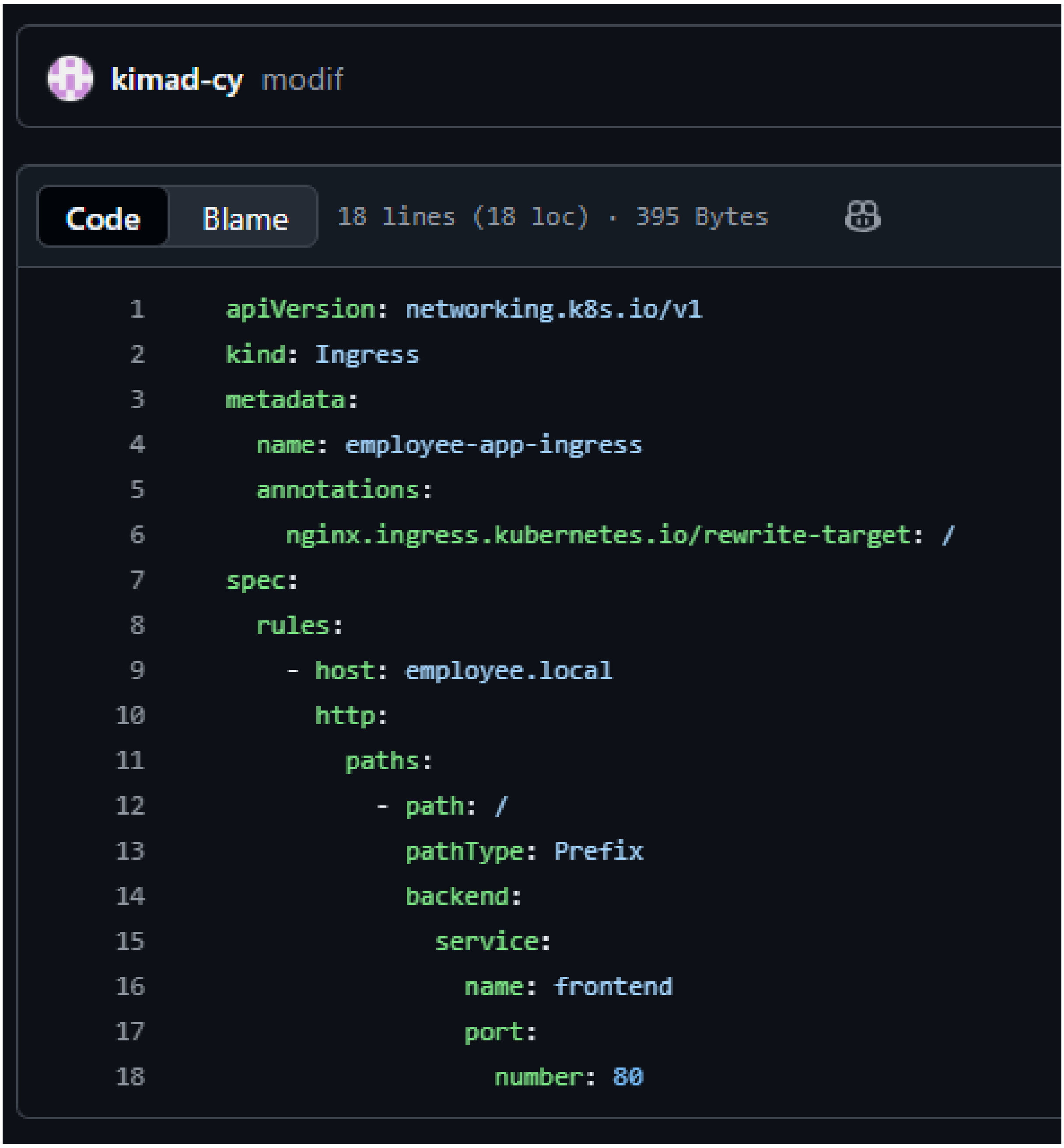
6.0.6 MySQL Service


Employee_Management / Kubernetes / mysql-service.yaml
chaimaeddib2005 OK
Code
Blame
10 lines (10 loc) · 134 Bytes

```yaml
apiVersion: v1
kind: Service
metadata:
  name: mysql
spec:
  selector:
    app: mysql
  ports:
    - port: 3306
      targetPort: 3306
```

Figure 6.6: MySQL Service YAML

6.0.7 Ingress Configuration

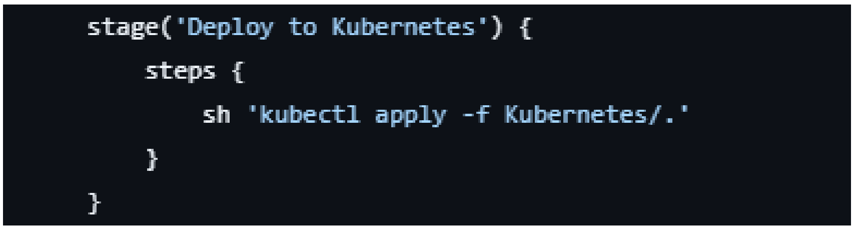


```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: employee-app-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
    - host: employee.local
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: frontend
                port:
                  number: 80
```

Figure 6.7: Ingress YAML configuration

6.1 Deployment on Minikube

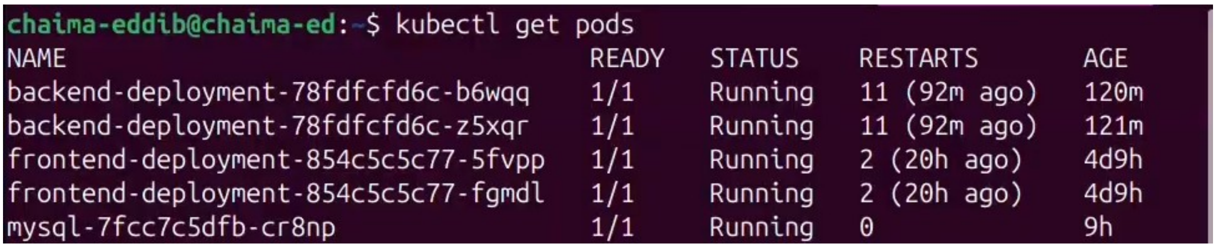
The Jenkins pipeline applies allYAML files from the /kubernetes directory automatically.



```
stage('Deploy to Kubernetes') {
    steps {
        sh 'kubectl apply -f Kubernetes/.'
    }
}
```

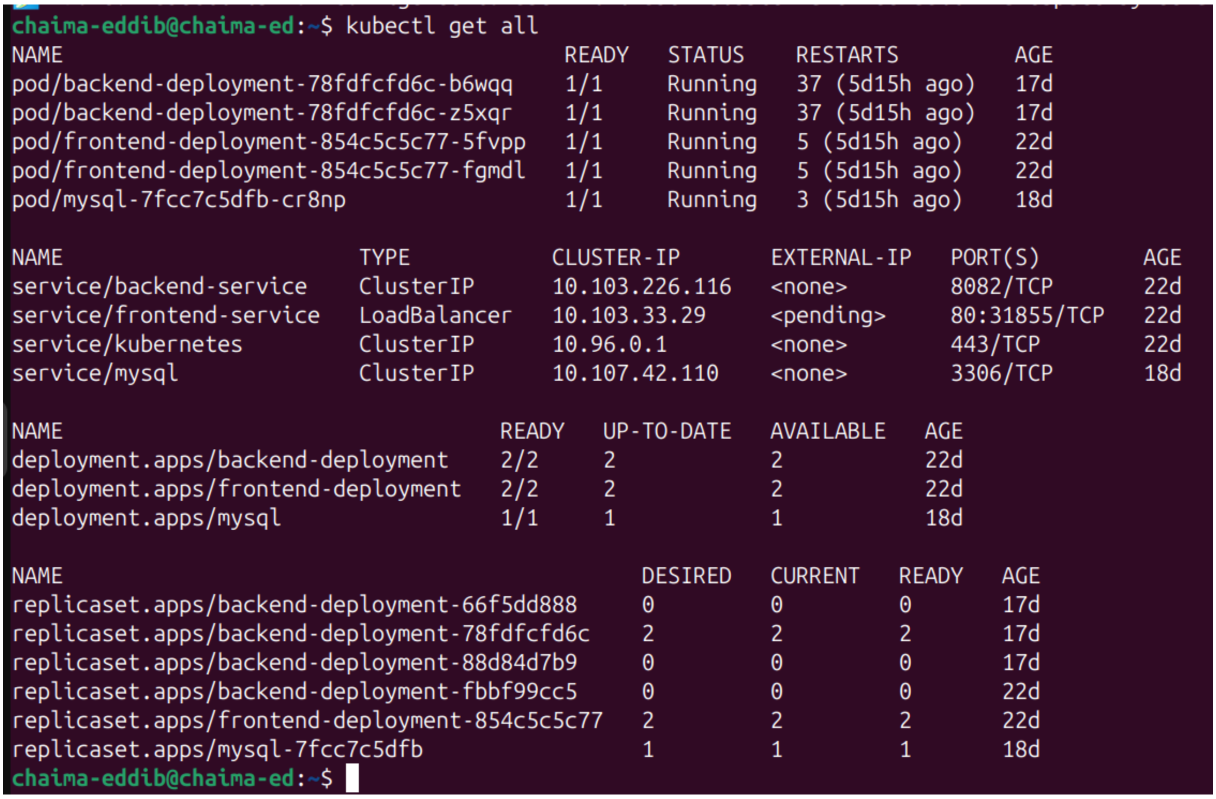
Figure 6.8: Jenkins stage for deploying to Minikube

6.1.1 Verification with kubectl



```
chaima-eddib@chaima-ed:~$ kubectl get pods
NAME                                  READY   STATUS    RESTARTS       AGE
backend-deployment-78fdfcfd6c-b6wqq   1/1     Running   11 (92m ago)   120m
backend-deployment-78fdfcfd6c-z5xqr   1/1     Running   11 (92m ago)   121m
frontend-deployment-854c5c5c77-5fvpp  1/1     Running   2 (20h ago)    4d9h
frontend-deployment-854c5c5c77-fgmdl  1/1     Running   2 (20h ago)    4d9h
mysql-7fcc7c5dfb-cr8np                1/1     Running   0              9h
```

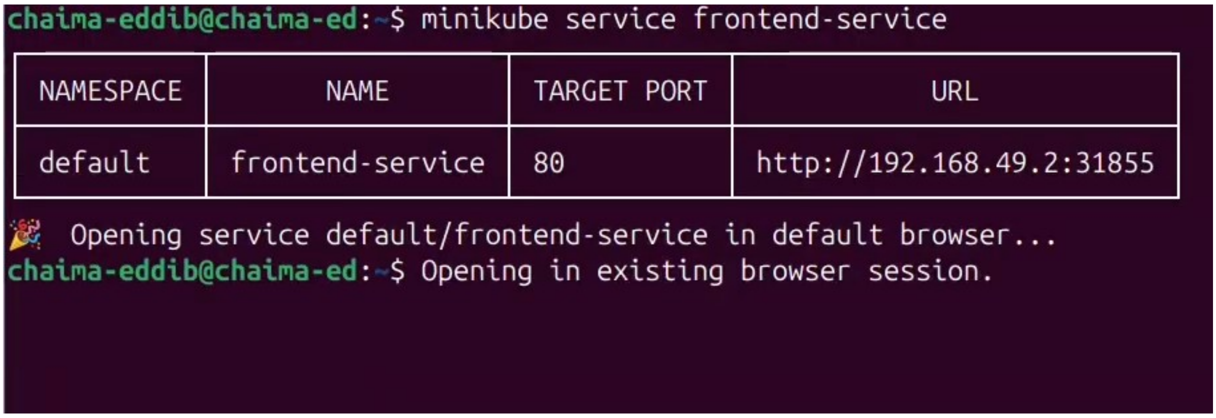
Figure 6.9: Output of kubectl get pods



```
chaima-eddib@chaima-ed:~$ kubectl get all
NAME                                      READY   STATUS    RESTARTS        AGE
pod/backend-deployment-78fdfcfd6c-b6wqq   1/1     Running   37 (5d15h ago)  17d
pod/backend-deployment-78fdfcfd6c-z5xqr   1/1     Running   37 (5d15h ago)  17d
pod/frontend-deployment-854c5c5c77-5fvpp  1/1     Running   5 (5d15h ago)   22d
pod/frontend-deployment-854c5c5c77-fgmdl  1/1     Running   5 (5d15h ago)   22d
pod/mysql-7fcc7c5dfb-cr8np                1/1     Running   3 (5d15h ago)   18d

NAME                       TYPE           CLUSTER-IP       EXTERNAL-IP   PORT(S)        AGE
service/backend-service    ClusterIP      10.103.226.116   <none>        8082/TCP       22d
service/frontend-service   LoadBalancer   10.103.33.29     <pending>     80:31855/TCP   22d
service/kubernetes         ClusterIP      10.96.0.1        <none>        443/TCP        22d
service/mysql              ClusterIP      10.107.42.110    <none>        3306/TCP       18d

NAME                                  READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/backend-deployment    2/2     2            2           22d
deployment.apps/frontend-deployment   2/2     2            2           22d
deployment.apps/mysql                 1/1     1            1           18d

NAME                                             DESIRED   CURRENT   READY   AGE
replicaset.apps/backend-deployment-66f5dd888     0         0         0       17d
replicaset.apps/backend-deployment-78fdfcfd6c    2         2         2       17d
replicaset.apps/backend-deployment-88d84d7b9     0         0         0       17d
replicaset.apps/backend-deployment-fbbf99cc5     0         0         0       22d
replicaset.apps/frontend-deployment-854c5c5c77   2         2         2       22d
replicaset.apps/mysql-7fcc7c5dfb                 1         1         1       18d
chaima-eddib@chaima-ed:~$
```

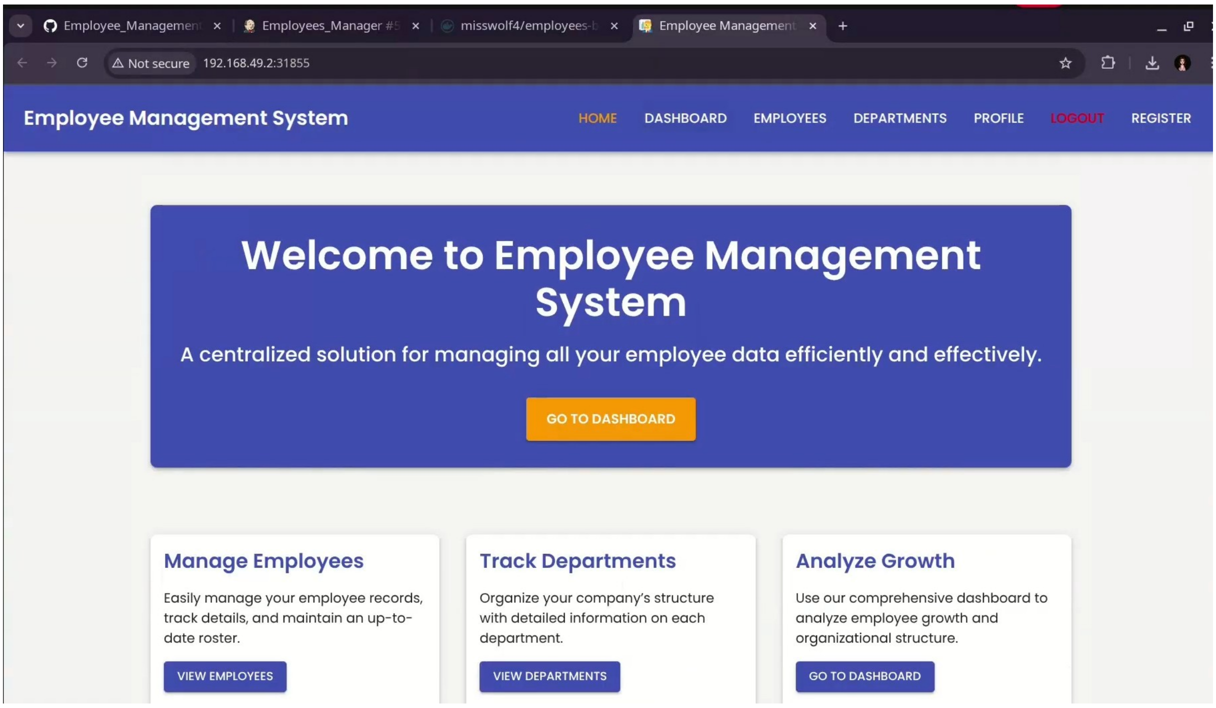
Figure 6.10: Output of kubectl get all

6.1.2 Accessing the Application



```
chaima-eddib@chaima-ed:~$ minikube service frontend-service
|-----------|------------------|-------------|---------------------------|
| NAMESPACE |       NAME       | TARGET PORT |            URL            |
|-----------|------------------|-------------|---------------------------|
| default   | frontend-service |          80 | http://192.168.49.2:31855 |
|-----------|------------------|-------------|---------------------------|
🎉  Opening service default/frontend-service in default browser...
chaima-eddib@chaima-ed:~$ Opening in existing browser session.
```

Figure 6.11: Minikube Ingress URL



Welcome to Employee Management System

A centralized solution for managing all your employee data efficiently and effectively.

Manage Employees

Easily manage your employee records, track details, and maintain an up-to-date roster.

Track Departments

Organize your company's structure with detailed information on each department.

Analyze Growth

Use our comprehensive dashboard to analyze employee growth and organizational structure.

Figure 6.12: Accessing the frontend application via browser

Chapter 7

Prometheus & Grafana – Monitoring

7.1 Deployment and Configuration

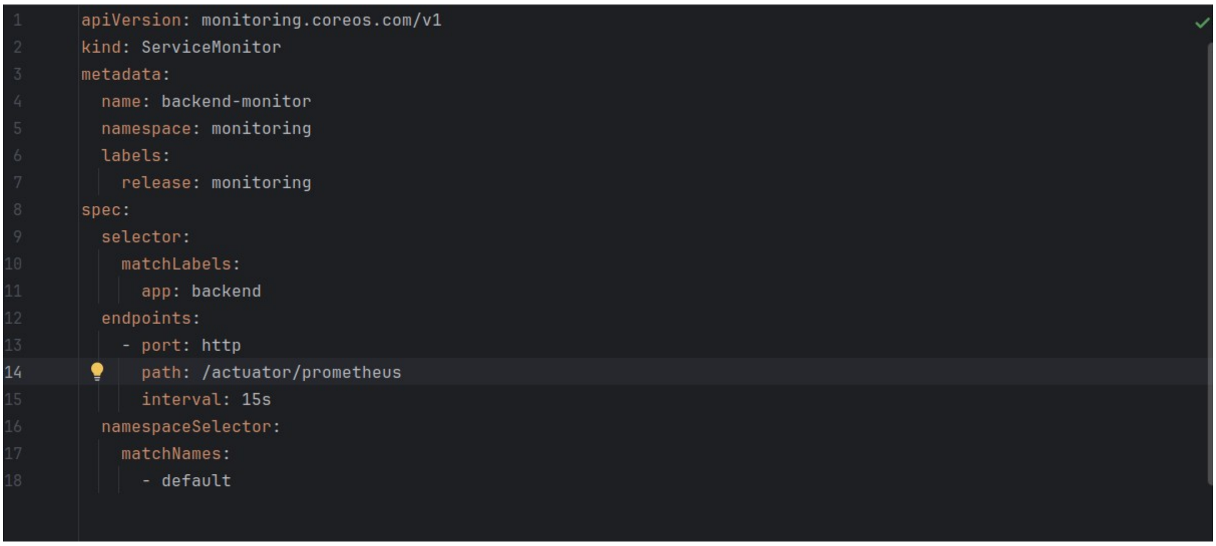
Prometheus and Grafana were deployed on the Minikube cluster using Helm charts. This setup allows monitoring of CPU, memory, service availability, and response times for all deployed applications.

The deployment was automated through Jenkins with the following stage:

```
stage ( ' Deploy    Monitoring ')    {
    steps   {
        sh  '''
            helm  repo  add  prometheus - community   https :// prometheus -
                community . github . io / helm - charts
            helm  repo  update

            helm  upgrade  -- install     monitoring    prometheus - community /
                kube - prometheus - stack    \
                  -- namespace   monitoring    -- create - namespace
        '''
    }
}
```

7.2 Prometheus ServiceMonitor

The monitoring of the backend application is configured using a ‘ServiceMonitor‘ resource:

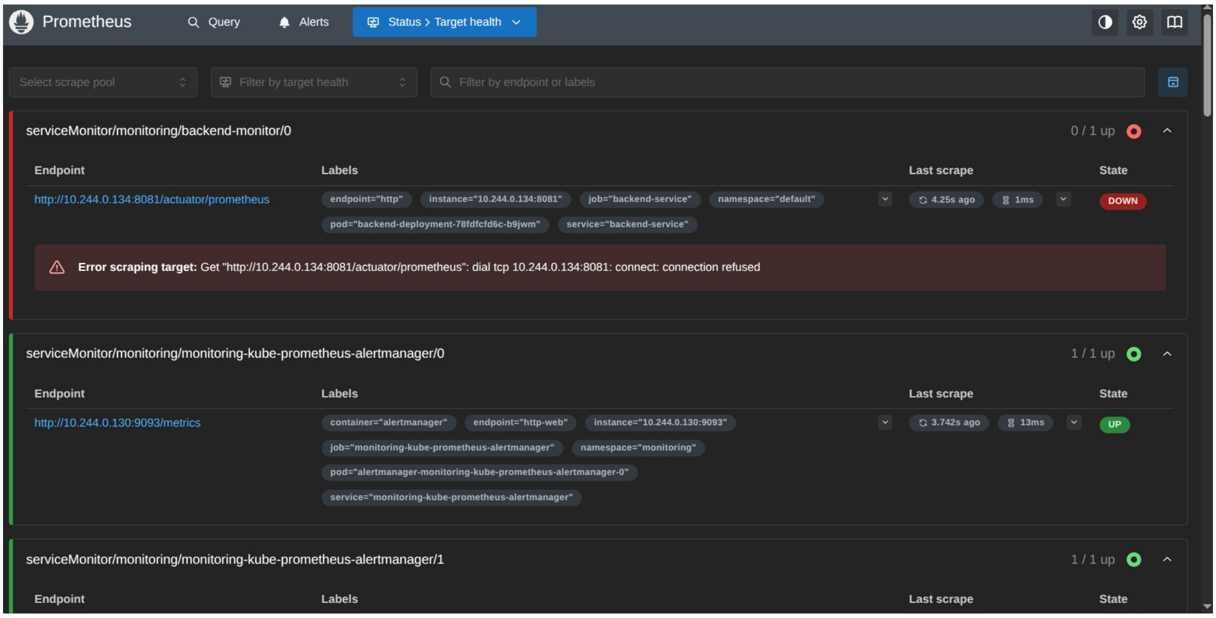


```yaml
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: backend-monitor
  namespace: monitoring
  labels:
    release: monitoring
spec:
  selector:
    matchLabels:
      app: backend
  endpoints:
    - port: http
      path: /actuator/prometheus
      interval: 15s
  namespaceSelector:
    matchNames:
      - default
```

Figure 7.1: ServiceMonitor YAML configuration for backend metrics

7.3 Prometheus Targets Health – Initial State

After deployment,Prometheus initially shows the backend target as **DOWN**This is because the Spring Boot application did not expose the required metrics endpoint.



serviceMonitor/monitoring/backend-monitor/0 — 0 / 1 up

Endpoint	Labels	Last scrape	State

http://10.244.0.134:8081/actuator/prometheus	endpoint="http" instance="10.244.0.134:8081" job="backend-service" namespace="default"
	pod="backend-deployment-78fdfcfd6c-b9jwm" service="backend-service"
		4.25s ago 1ms	DOWN

Error scraping target: Get "http://10.244.0.134:8081/actuator/prometheus": dial tcp 10.244.0.134:8081: connect: connection refused

serviceMonitor/monitoring/monitoring-kube-prometheus-alertmanager/0 — 1 / 1 up

Endpoint	Labels	Last scrape	State

http://10.244.0.130:9093/metrics	container="alertmanager" endpoint="http-web" instance="10.244.0.130:9093"
	job="monitoring-kube-prometheus-alertmanager" namespace="monitoring"
	pod="alertmanager-monitoring-kube-prometheus-alertmanager-0"
	service="monitoring-kube-prometheus-alertmanager"
		3.742s ago 13ms	UP

serviceMonitor/monitoring/monitoring-kube-prometheus-alertmanager/1 — 1 / 1 up

Endpoint	Labels	Last scrape	State

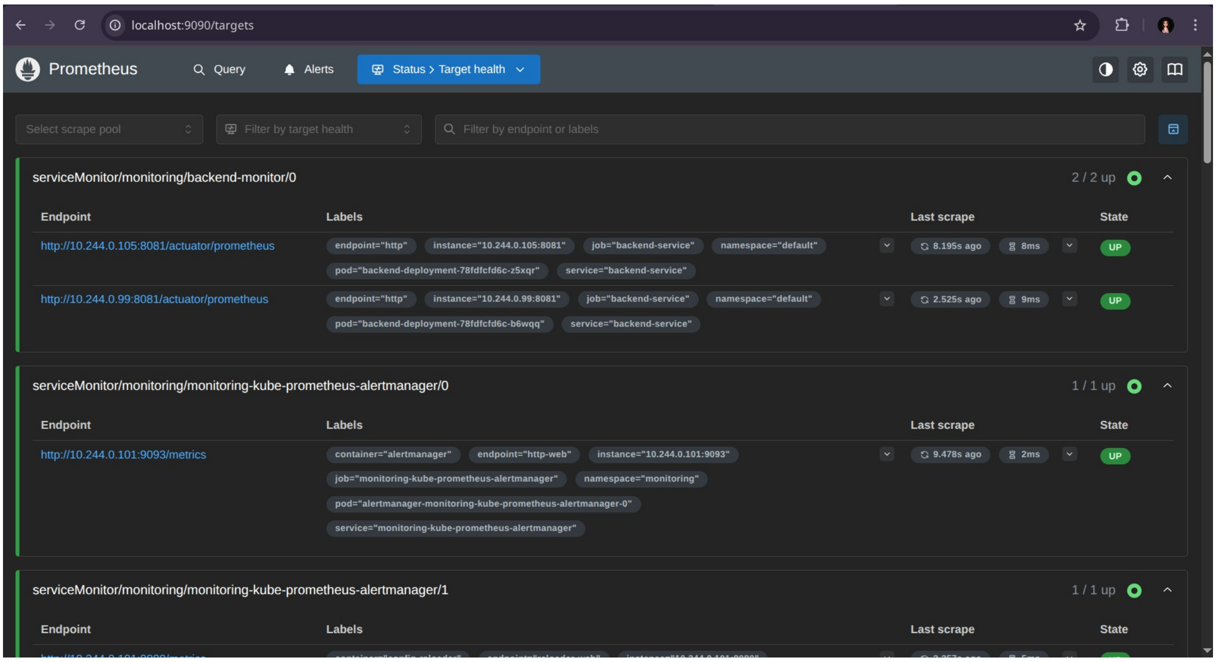
Figure 7.2: Prometheus target initially DOWN due to missing metrics endpoint

7.3.1 Resolution: Spring Boot Actuator Integration

To expose metrics, Spring Boot actuator was enabled with the Prometheus endpoint:

```
management . endpoints . web . exposure . include = health , info , prometheus
management . endpoint . prometheus . enabled = true
```

After redeploying the backend, Prometheus detects the target as **UP**.



serviceMonitor/monitoring/backend-monitor/0 — 2 / 2 up

Endpoint	Labels	Last scrape	State

http://10.244.0.105:8081/actuator/prometheus	endpoint="http"
	instance="10.244.0.105:8081"
	job="backend-service"
	namespace="default"
	pod="backend-deployment-78fdfcfd6c-z5xqr"
	service="backend-service"
		8.195s ago
		8ms
			UP

http://10.244.0.99:8081/actuator/prometheus	endpoint="http"
	instance="10.244.0.99:8081"
	job="backend-service"
	namespace="default"
	pod="backend-deployment-78fdfcfd6c-b6wqq"
	service="backend-service"
		2.525s ago
		9ms
			UP

serviceMonitor/monitoring/monitoring-kube-prometheus-alertmanager/0 — 1 / 1 up

Endpoint	Labels	Last scrape	State

http://10.244.0.101:9093/metrics	container="alertmanager"
	endpoint="http-web"
	instance="10.244.0.101:9093"
	job="monitoring-kube-prometheus-alertmanager"
	namespace="monitoring"
	pod="alertmanager-monitoring-kube-prometheus-alertmanager-0"
	service="monitoring-kube-prometheus-alertmanager"
		9.478s ago
		2ms
			UP

serviceMonitor/monitoring/monitoring-kube-prometheus-alertmanager/1 — 1 / 1 up

Endpoint	Labels	Last scrape	State

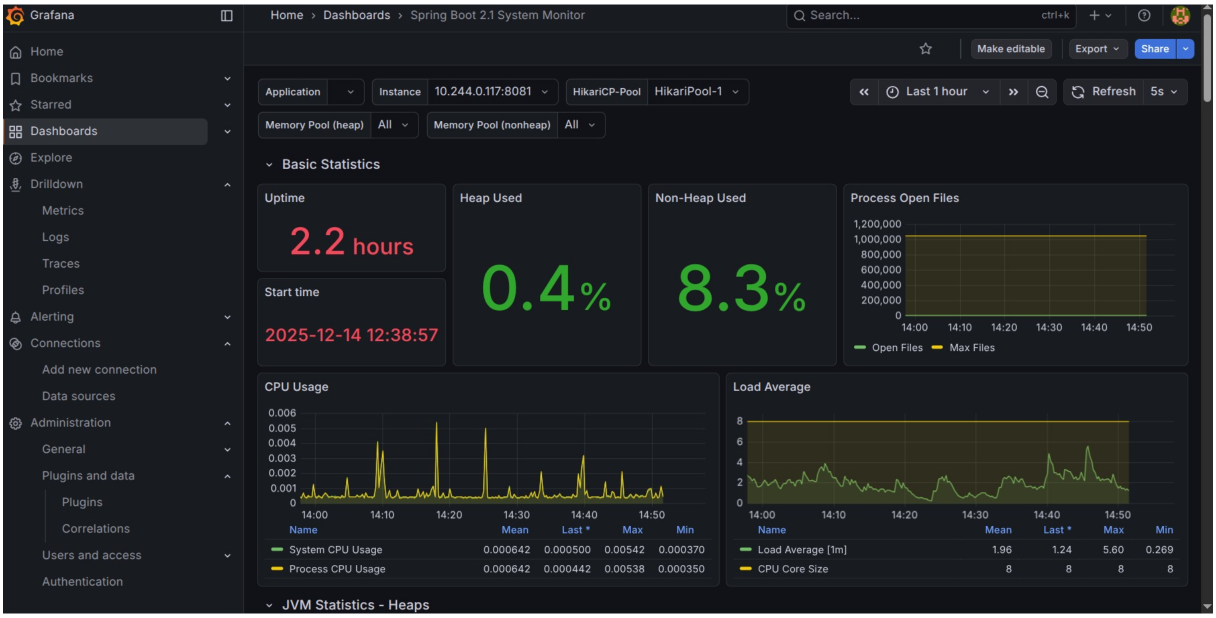
Figure 7.3: Prometheus target UP after enabling Spring Boot actuator

7.4 Grafana Dashboard

The Grafana dashboard displays key performance indicators (KPIs) such as:

- CPU usage
- RAM usage
- Service availability
- Response time

Instead of creating a dashboard from scratch, we used the built-in Spring Boot dashboard available on Grafana Dashboards: JustAI System Monitor (ID 11378). This dashboard was imported and configured to visualize the metrics collected from our backend application.



Application | Instance 10.244.0.117:8081 | HikariCP-Pool HikariPool-1

Memory Pool (heap) All | Memory Pool (nonheap) All

Uptime: 2.2 hours

Start time: 2025-12-14 12:38:57

Heap Used: 0.4%

Non-Heap Used: 8.3%

Name	Mean	Last *	Max	Min
System CPU Usage	0.000642	0.000500	0.00542	0.000370
Process CPU Usage	0.000642	0.000442	0.00538	0.000350

Name	Mean	Last *	Max	Min
Load Average [1m]	1.96	1.24	5.60	0.269
CPU Core Size	8	8	8	8

Figure 7.4: Grafana dashboard showing real-time metrics for applications

7.5 Grafana Drill-down

Grafana allows drill-down into specific services or pods for detailed analysis of metrics over time.

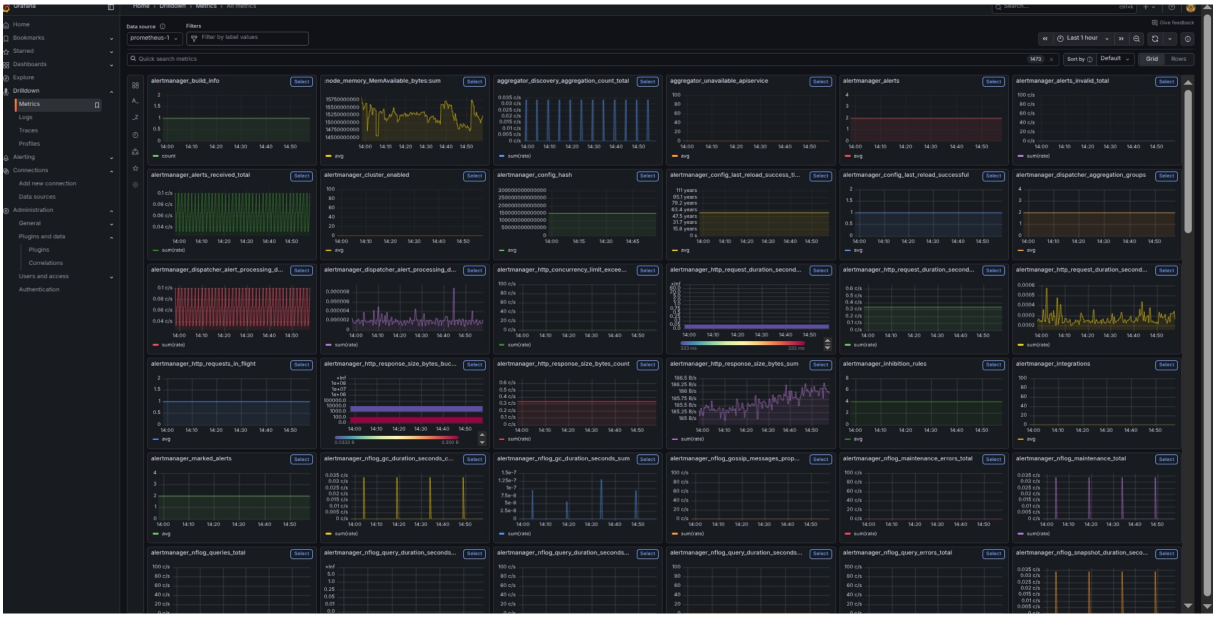

Data source
prometheus-1
Filters
Filter by label values
Quick search metrics
alertmanager_build_info
node_memory_MemAvailable_bytes:sum
aggregator_discovery_aggregation_count_total
aggregator_unavailable_apiservice
alertmanager_alerts
alertmanager_alerts_invalid_total
alertmanager_alerts_received_total
alertmanager_cluster_enabled
alertmanager_config_hash
alertmanager_config_last_reload_success_ti...
alertmanager_config_last_reload_successful
alertmanager_dispatcher_aggregation_groups
alertmanager_dispatcher_alert_processing_d...
alertmanager_dispatcher_alert_processing_d...
alertmanager_http_concurrency_limit_excee...
alertmanager_http_request_duration_second...
alertmanager_http_request_duration_second...
alertmanager_http_request_duration_second...
alertmanager_http_requests_in_flight
alertmanager_http_response_size_bytes_buc...
alertmanager_http_response_size_bytes_count
alertmanager_http_response_size_bytes_sum
alertmanager_inhibition_rules
alertmanager_integrations
alertmanager_marked_alerts
alertmanager_nflog_gc_duration_seconds_c...
alertmanager_nflog_gc_duration_seconds_sum
alertmanager_nflog_gossip_messages_prop...
alertmanager_nflog_maintenance_errors_total
alertmanager_nflog_maintenance_total
alertmanager_nflog_queries_total
alertmanager_nflog_query_duration_seconds...
alertmanager_nflog_query_duration_seconds...
alertmanager_nflog_query_duration_seconds...
alertmanager_nflog_query_errors_total
alertmanager_nflog_snapshot_duration_seco...
Select

Figure 7.5: Detailed Grafana drill-down view of backend service metrics

7.5.1 Critical KPI and Alerts

The most critical KPI for our backend application is **application availability**, represented by the Prometheus metric up. This metric is considered critical because if the application is down, all other metrics become irrelevant — users cannot access the service, and no business processes can be executed. Ensuring high availability is therefore the top priority.

In addition to availability, we monitor CPU usage, uptime, and heap memory usage to proactively detect performance degradation or resource saturation. The following table summarizes the selected alerts, their objectives, and thresholds:

Metric	**Objective**	**Warning Seuil**	**Critical Seuil**
Availability (up)	Ensure the application is reachable	N/A	0 (down)
CPU Usage	Prevent CPU saturation	70–80% average over 5 min	>90% average o
Uptime	Track application stability / restarts	>1 restart in 5 min	>3 restarts in 5 r
Heap Usage	Prevent JVM OutOfMemoryError and GC stalls	>70–75% of max heap	>85–90% of max

Table 7.1: Selected alerts and thresholds for the backend application

Monitoring these KPIs allows the team to detect critical issues early, maintain service reliability, and take proactive actions to prevent performance degradation.

Chapter 8

Conclusion

This project demonstrates the full DevOps lifecycle for a JEE application, from source control to CI/CD, containerization, Kubernetes deployment, and monitoring. Key achievements:

- Automated CI/CD pipelines using Jenkins
- High-quality code validated with SonarQube
- Containerized applications with Docker
- Deployed services on Kubernetes with Minikube
- Monitored metrics with Prometheus and Grafana

Challenges included configuring pipelines, managing dependencies, and integrating monitoring tools.Future improvements may include cloud deployment and scaling.