

(APPLICATION MONITORING & AUTOMATIC DEPLOYMENT)

Submitted by

Lehar Agrawal

For the award of the degree of

MCA (Masters in Computer Applications)

Under the supervision of

Supervisor

(Ankur Sharma, Senior DevOps Engineer, 8Bit System Private Ltd)



Faculty of Mathematics and Computing

Banasthali Vidyapith

Banasthali - 304022

Session: 2024



CIN No. U74999RJ2021PTC076221

TAN NO. : JPRB07041G

REF#2024/25/INT/0005

TO WHOM IT MAY CONCERN

This is to certify that **Lehar Agarwal** has successfully completed an internship with **8Bit System Pvt. Ltd.** as a JAVA & DevOps Intern from **10-JAN-2024 to 02-JUN-2024**.

During the internship period, **Lehar Agarwal** demonstrated a strong understanding of concern technology principles and actively contributed to various projects and tasks.

Lehar Agarwal showed excellent technical skills, a willingness to learn, and a collaborative attitude throughout the internship. Their contributions were valuable, and they consistently demonstrated a commitment to quality and best practices in concern technology.

We wish **Lehar Agarwal** all the best in their future endeavors and believe that she has a bright future ahead in the field of concern technology.

Attendance	Discipline	Project Grade
A++	Excellent	A++

With best wishes,

Form 8Bit System Private Limited



www.8bit.co.in

Abstract

This project aims to develop an automated Continuous Integration and Continuous Deployment (CI/CD) pipeline for a Spring Boot application. The pipeline is designed to automate the entire process from building the application, creating Docker images, pushing these images to a Nexus repository, and deploying them on a Kubernetes cluster. Leveraging ArgoCD, the pipeline ensures automatic tracking and deployment of changes upon new code commits. Continuous monitoring is achieved using the ELK (Elasticsearch, Logstash, Kibana) stack, and automatic updates of deployments are managed through ArgoCD, which ensures that any changes in the Git repository are automatically reflected in the Kubernetes cluster. The integration with tools like Git, Jenkins, Docker, Maven, Kubernetes, and ArgoCD, In ArgoCD you can monitor logs and data and ensures a cohesive and efficient workflow, significantly reducing manual intervention and accelerating the release process. The setup ensures high availability, easy scalability, and comprehensive monitoring, making it suitable for modern microservices-based architectures. This automated pipeline not only supports continuous delivery but also ensures that the application is always up-to-date with the latest code changes, thereby maintaining consistency and reliability across development and production environments.

Acknowledgement

I would like to extend my sincere gratitude to everyone who supported the successful completion of this project. I am especially thankful to my project advisor, Ankur Sharma, for their expert guidance and unwavering support throughout this endeavor.

I deeply appreciate the efforts of the mentor behind Jenkins, Docker, Nexus, Kubernetes, and ArgoCD for providing such reliable tools, which were crucial to this project's success. I also thank 8Bit System Private Limited for offering the essential resources and infrastructure necessary for the project's execution.

Lastly, I am profoundly thankful to my family for their constant support and understanding during the course of this project. Their encouragement has been invaluable to me.

Table of Contents

- Objective
- Requirement Analysis (SRS)
 - Requirement Specification
 - H/w and S/w Requirements
 - Feasibility Study
 - Product Functions
- System Design (SDS)
 - High-level Design
 - Activity Diagrams
 - Sequence Diagrams
- Coding
 - Only main Modules (if available)
- Testing
 - Test cases
- User Interfaces
- Appendices
- References

Objective

The objective of this project is to establish an automated CI/CD (Continuous Integration and Continuous Deployment) pipeline for a Spring Boot application. This pipeline will leverage Jenkins for continuous integration, Docker for containerization, Nexus for artifact storage, Kubernetes for container orchestration, and ArgoCD for automate the deployment process. Specifically, the project aims to:

Automate the Build and Test Process:

Use Jenkins to automatically trigger builds and tests upon detecting new commits in the Git repository, ensuring code quality and functionality before deployment.

Create and Store Docker Images:

Generate Docker images for the Spring Boot application, tag them appropriately, and push them to a Nexus repository for centralized storage and version control.

Deploy Docker Images to Kubernetes:

Define and manage deployment configurations using Kubernetes YAML files, and deploy the Docker images as pods within a Kubernetes cluster, ensuring scalability and high availability.

Implement Continuous Monitoring and Automated Updates:

Utilize the ELK (Elasticsearch, Logstash, Kibana) stack for comprehensive logging and monitoring of deployed applications.

Use ArgoCD to continuously monitor the Git repository for changes in deployment configurations and automatically update Kubernetes deployments, ensuring that the live environment is always in sync with the latest code changes.

The ultimate goal is to streamline and automate the deployment process, reduce manual intervention, and improve the speed and reliability of delivering software updates. By integrating these tools and technologies, the project aims to provide a robust, scalable, and efficient CI/CD pipeline suitable for modern cloud-native application architectures.

Requirement Analysis (SRS)

Requirement Specification

- Automate build and deployment process.
- Ensure seamless integration with Git, Jenkins, Nexus, Kubernetes, and ArgoCD.
- Automatically update Kubernetes deployments on new commits.

The project requires a set of functional and non-functional requirements. Functionally, it necessitates seamless data flow from GitHub to ArgoCD. Additionally, the system should support automated deployment and configuration using Jenkins, Git and ArgoCD. Non-functionally, the system must be scalable, reliable, and secure.

Functional Requirements

Automated Build Process:

The system should automatically trigger a build process upon detecting new commits in the Git repository.

Jenkins should use Maven to compile the Spring Boot application and run unit and integration tests.

Docker Image Creation:

The system should create Docker images of the Spring Boot application after a successful build.

Docker images should be tagged appropriately (e.g., with version numbers or build IDs).

Artifact Storage:

The Docker images should be pushed to a Nexus repository for centralized storage and version control.

Deployment to Kubernetes:

The system should deploy the Docker images to a Kubernetes cluster as pods.

Kubernetes should manage deployment configurations via YAML files, specifying the desired state, replicas, and resource limits.

Continuous Monitoring:

The ELK stack should collect and visualize logs from the application pods for comprehensive monitoring.

Kibana should provide dashboards for real-time log analysis and troubleshooting.

Automated Deployment Updates:

ArgoCD should continuously monitor the Git repository for changes in deployment configurations.

ArgoCD should automatically sync changes and update Kubernetes deployments to reflect the latest configurations.

Credentials Management:

The system should securely manage and store credentials for accessing Git, Nexus, and Kubernetes.

Non-Functional Requirements

Scalability:

The system should support horizontal scaling of application pods based on CPU utilization or other metrics.

Kubernetes should provide load balancing to distribute traffic across multiple pods.

Availability:

The system should ensure high availability of the application by deploying multiple replicas and using Kubernetes' self-healing capabilities.

Kubernetes should automatically replace failed pods to maintain the desired state.

Performance:

The CI/CD pipeline should complete the build, test, and deployment processes within an acceptable time frame to support rapid development cycles.

The system should handle concurrent builds and deployments efficiently.

Security:

Docker images should be scanned for vulnerabilities before being pushed to the Nexus repository.

Sensitive information, such as credentials and environment variables, should be encrypted and securely managed.

Reliability:

The system should provide consistent and reliable deployment processes to minimize downtime and errors.

Automated tests should ensure the quality and functionality of the application before deployment.

Maintainability:

The system should be easy to maintain, with clear documentation and modular components that can be updated independently.

Jenkins pipelines, Kubernetes configurations, and ArgoCD setups should be version-controlled and documented.

Extensibility:

The CI/CD pipeline should support the deployment of multiple microservices, each with its own Docker image and Kubernetes deployment.

Jenkins and ArgoCD should support plugins and integrations to extend functionality as needed.

Monitoring and Alerting:

The system should provide real-time monitoring and alerting for build failures, deployment issues, and application errors.

Alerts should be configurable and sent to appropriate channels (e.g., email).

H/w and S/w Requirements**Hardware:**

- Sufficient RAM and CPU resources to run Docker containers effectively
- Storage space for storing log data, depending on the expected volume.
- Network connectivity for communication between components.
- RAM: 16GB
- **Jenkins server:**

CPU: Multi-core processor (at least 4 cores; 8 or more cores recommended)

Memory: Minimum of 16 GB RAM, ideally 32 GB or more

Storage: SSD with a minimum of 100 GB of available space

Network: High-speed network connection (1 Gbps or higher)

- **Kubernetes Cluster Nodes:**

Master Node:

CPU: Multi-core processor (minimum 4 cores; 8 or more cores recommended)

Memory: Minimum of 16 GB RAM, ideally 32 GB or more

Storage: SSD with a minimum of 100 GB of available space

Network: High-speed network connection (1 Gbps or higher)

Worker Nodes:

CPU: Multi-core processor (minimum 4 cores per node)

Memory: Minimum of 16 GB RAM per node

Storage: SSD with a minimum of 100 GB of available space per node

Network: High-speed network connection (1 Gbps or higher)

- **Nexus Repository Server:**

CPU: Multi-core processor (minimum 4 cores)

Memory: Minimum of 16 GB RAM

Storage: SSD with a minimum of 200 GB of available space

Network: High-speed network connection (1 Gbps or higher)

- **SSD:**

- a) Capacity: At least 256 GB or higher to accommodate system files, logs, and other data.

- b) Interface: SATA III or NVMe for faster data transfer rates.
- c) Read/Write Speeds: Look for SSDs with high read/write speeds to ensure efficient data processing.
- d) Endurance: Consider SSDs with higher endurance ratings if the system will be subjected to heavy read/write operations.

Software Requirements:

Jenkins:

Version: Latest stable version

Plugins: Git, Docker, Maven, Kubernetes, Pipeline, ArgoCD, among others.

Docker:

Version: Latest stable version

Maven:

Version: Latest stable version

Spring Boot:

Version: Latest stable version compatible with your project

Kubernetes:

Version: Latest stable version

Tools: kubectl, kubeadm, and optionally Helm

ArgoCD:

Version: Latest stable version

Nexus Repository Manager:

Version: Latest stable version

ELK Stack (Elasticsearch, Logstash, Kibana):

Version: Latest stable versions

Additional Software Tools

Git:

Version: Latest stable version

Java Development Kit (JDK):

Version: JDK 11 or later (depending on your Spring Boot application requirements)

Operating System:

Server OS: Linux distributions such as Ubuntu or CentOS

Container OS: Linux-based distributions like Alpine or Debian

Networking and Security Requirements

Network:

Reliable, high-speed network connectivity between servers

Appropriate network segmentation for enhanced security and performance

Security:

Firewalls to protect servers

Secure access controls such as SSH keys and VPNs

SSL/TLS for secure communications

Feasibility Study

The feasibility study demonstrates the practicality and viability of implementing the proposed system. It assesses factors such as technical feasibility, economic viability, and operational feasibility to determine the project's likelihood of

success. With the availability of open-source tools and the flexibility of Docker containerization, the project appears feasible from both a technical and economic standpoint.

Technical Feasibility:

All components (Jenkins, Docker, Nexus, Kubernetes, ArgoCD) are compatible and support the required integrations.

Technology Assessment: Evaluate the suitability of selected technologies such as Elasticsearch, Kibana, Logstash, Filebeat, Apache Kafka, Jenkins, Ansible, and Shell Scripting for the log monitoring requirements.

Integration Challenges: Assess the compatibility and interoperability of various components and tools to ensure seamless integration and data flow.

Scalability and Performance: Analyze the ability of the system to scale with increasing log volumes and maintain optimal performance under various load conditions.

Security Considerations: Address security concerns related to data privacy, access control, encryption, and compliance with industry regulations.

Operational Feasibility:

Streamlined workflow from code commit to deployment reduces manual intervention and speeds up the release cycle.

User Acceptance: Assess the willingness and readiness of users to adopt and utilize the log monitoring system effectively.

Training Needs: Identify training requirements for administrators, operators, and end-users to ensure proper utilization and maintenance of the system.

Change Management: Plan for organizational changes and adjustments required to integrate the log monitoring system into existing workflows and processes.

Maintenance and Support: Determine the availability of resources and expertise for ongoing maintenance, troubleshooting, and support of the system.

Economic Feasibility:

Initial setup costs are justified by the reduction in manual deployment efforts and faster release cycles.

Cost Analysis: Evaluate the cost of implementing and maintaining the log monitoring system, including hardware, software licenses, and ongoing operational expenses.

Return on Investment (ROI): Estimate the potential benefits such as improved productivity, reduced downtime, and enhanced troubleshooting capabilities against the initial and ongoing investment.

Budget Allocation: Determine if the project aligns with the available budget and financial resources of the organization.

Cost-Benefit Analysis: Compare the costs and benefits to assess whether the project is economically viable and justifiable.

Product Functions

- Automated build and test of Spring Boot application.
- Creation and storage of Docker images in Nexus repository.
- Deployment of Docker images to Kubernetes.
- Continuous monitoring of ELK Stack and automatic updates of deployments using ArgoCD.

The CI/CD pipeline for the Spring Boot application is designed to automate various stages of development, building, deployment, and monitoring. Below are detailed descriptions of each function:

Automated Build and Test of Spring Boot Application:

Description: Jenkins is configured to automatically trigger builds when new commits are detected in the Git repository. The build process involves pulling the latest code, compiling the application using Maven, and running unit and integration tests.

Steps:

Git Commit: Developers commit code changes to the Git repository.

Jenkins Build Trigger: Jenkins detects the commit and initiates the build job.

Maven Build: Jenkins uses Maven to compile the code and execute tests.

Test Results: Jenkins aggregates and reports the results of the tests.

Benefits: Ensures code quality and functionality before deployment, minimizing the risk of bugs in production.

Creation and Storage of Docker Images in Nexus Repository:

Description: Upon successful build and testing, Jenkins creates a Docker image of the Spring Boot application. This image is tagged and pushed to the Nexus repository for storage and version control.

Steps:

Dockerfile Execution: Jenkins uses a Dockerfile to create the Docker image.

Image Tagging: The Docker image is tagged with a version number or build ID.

Push to Nexus: Jenkins pushes the tagged Docker image to the Nexus repository.

Benefits: Ensures consistent application environments, simplifies deployment, and provides a centralized repository for Docker images.

Deployment of Docker Images to Kubernetes:

Description: Kubernetes is used to deploy the Docker images as pods within a cluster. Deployment configurations are specified in Kubernetes YAML files, defining the desired state of the application, including the number of replicas, resource limits, and environment variables.

Steps:

Pull Image from Nexus: Kubernetes pulls the Docker image from the Nexus repository.

Pod Deployment: The Docker image is deployed as pods in the Kubernetes cluster.

Service Exposure: The application is exposed via Kubernetes services for internal and external access.

Benefits: Automates the deployment process, ensures high availability, and facilitates easy scaling of applications.

Continuous Monitoring of ELK Stack and Automatic Updates of Deployments using ArgoCD:

Description: The Elastic Stack (ELK) is used for logging and monitoring the deployed applications. ArgoCD continuously monitors the Git repository for configuration changes and automatically updates the Kubernetes deployments.

Steps:

Log Aggregation: Elasticsearch collects logs from the application pods, Logstash processes the logs, and Kibana provides a dashboard for visualization.

ArgoCD Monitoring: ArgoCD monitors the Git repository for changes in deployment configurations.

Automatic Sync: When changes are detected, ArgoCD automatically syncs the changes and updates the Kubernetes deployments.

Status Reporting: ArgoCD provides a real-time status of the deployment and any discrepancies.

Benefits: Provides comprehensive monitoring and logging, facilitates quick troubleshooting, ensures configurations are up-to-date, and automates deployment updates.

Additional Considerations:-

Security:

Credentials Management: Secure storage and management of credentials for accessing Git, Nexus, and Kubernetes.

Image Scanning: Automated scanning of Docker images for vulnerabilities before pushing to Nexus.

Scalability:

Horizontal Pod Autoscaling: Kubernetes can automatically scale the number of pods based on CPU utilization or other metrics.

Load Balancing: Kubernetes services provide load balancing to distribute traffic across multiple pods.

Resilience:

Self-Healing: Kubernetes automatically replaces failed pods to maintain the desired state.

Resource Management: Configurations for resource limits and requests ensure efficient utilization of cluster resources.

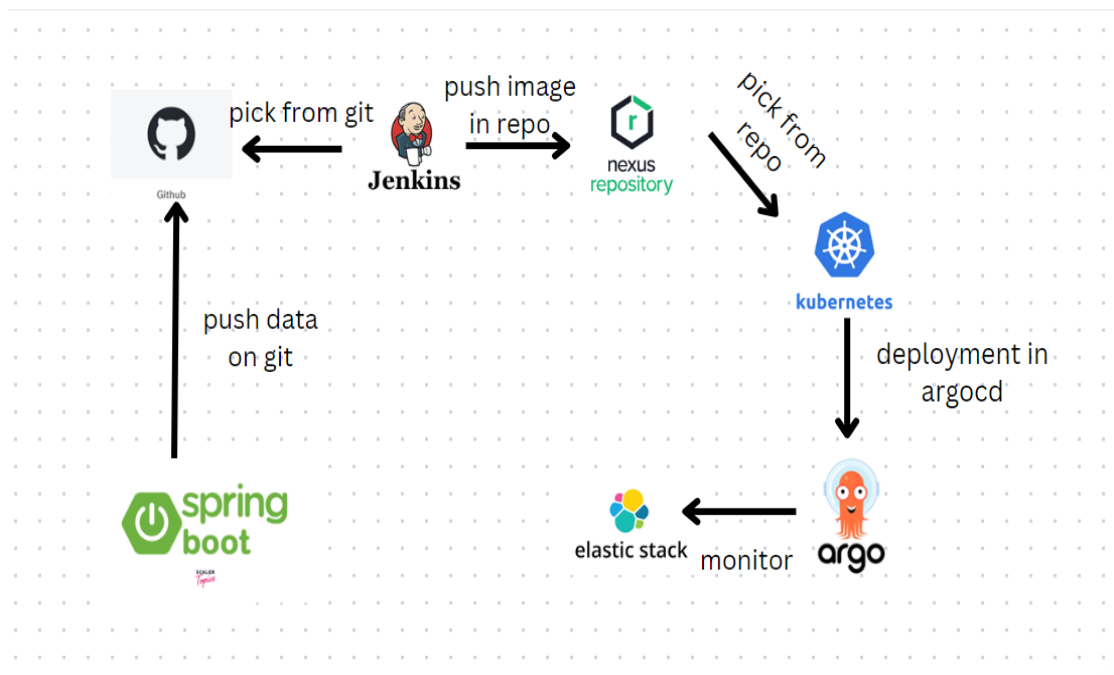
Extensibility:

Microservices: The pipeline supports the deployment of multiple microservices, each with its own Docker image and Kubernetes deployment.

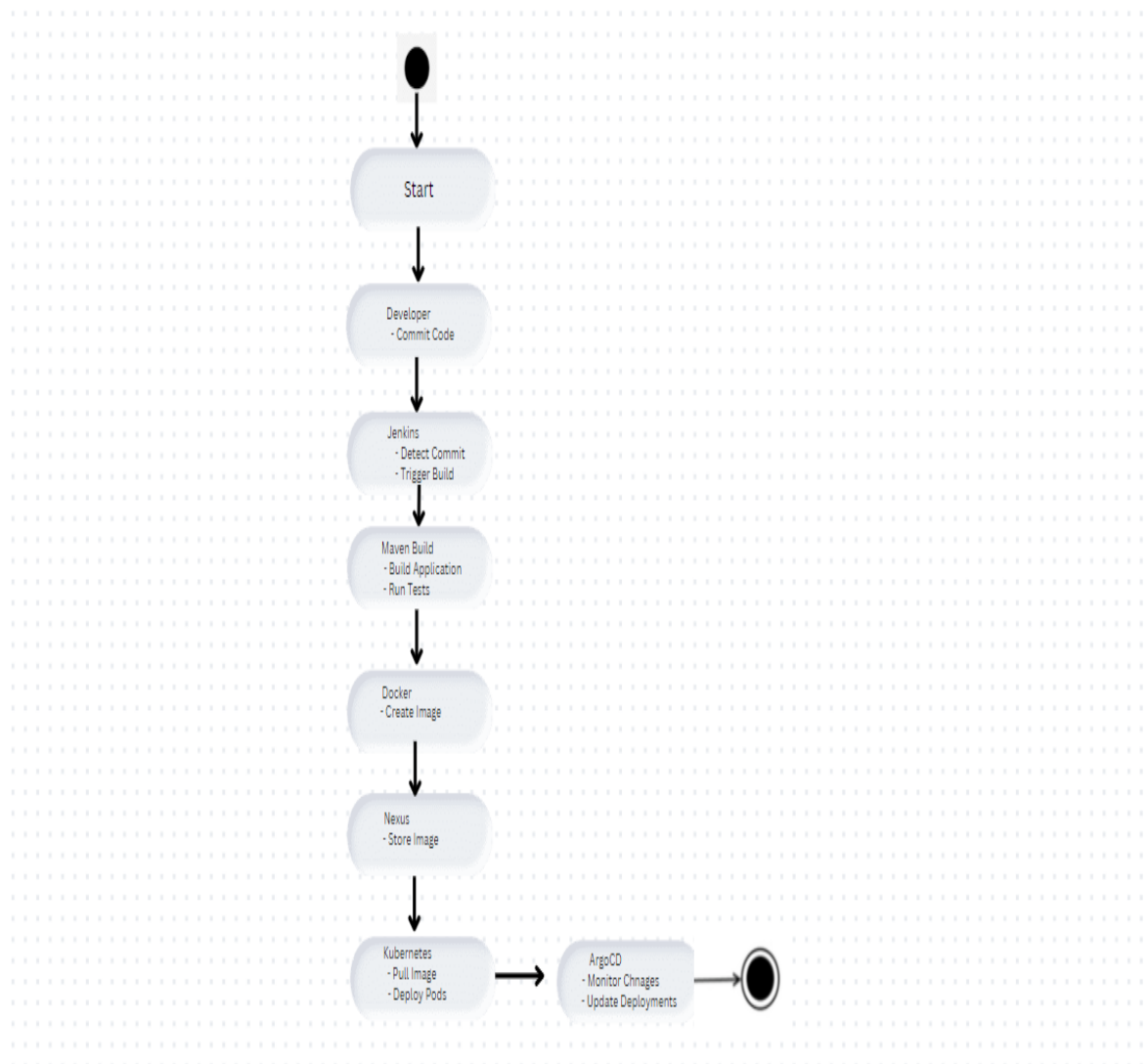
Plugins: Jenkins and ArgoCD support various plugins and integrations to extend functionality.

System Design (SDS)

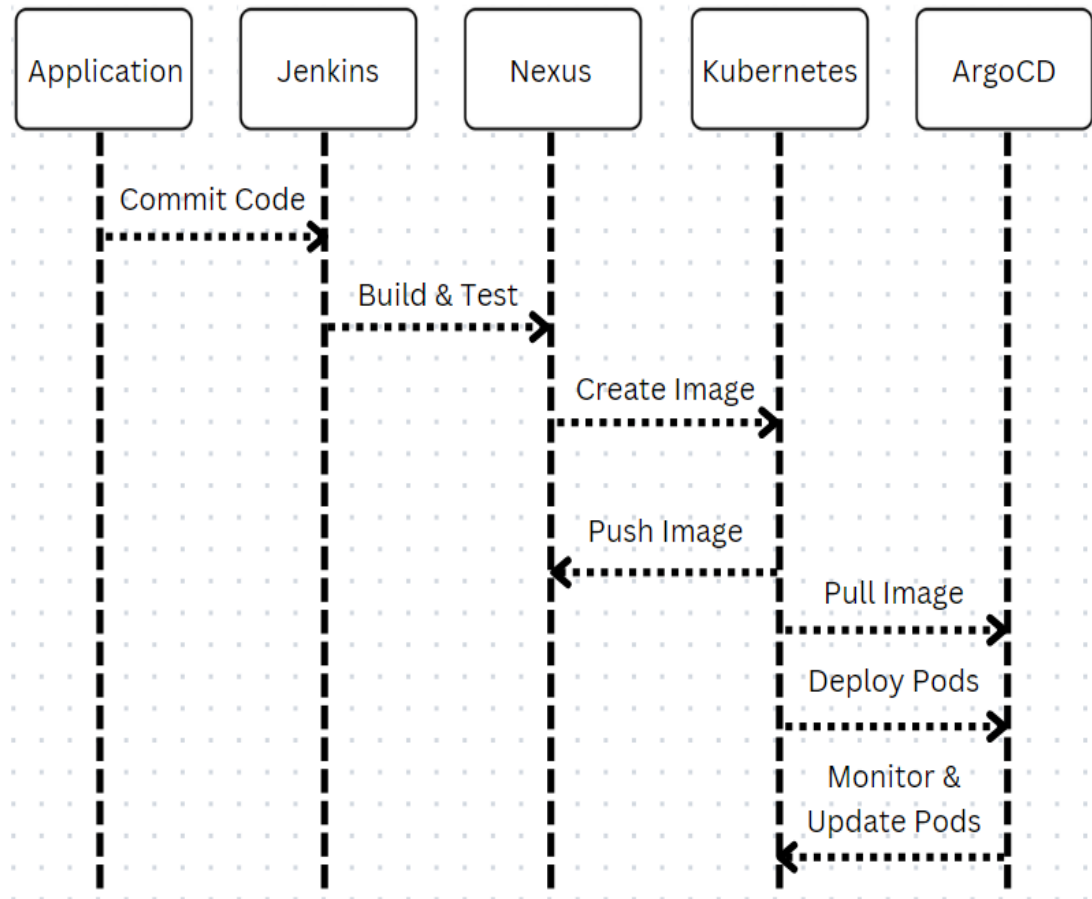
High-level Design



Activity Diagram



Sequence Diagram



Coding

Jenkins Pipeline

```
pipeline {

    agent any

    environment {

        springBootImageName = "newbuild"

        springBootImageTag = "1.0.${env.BUILD_NUMBER}"

        esImageName = "elasticsearch"

        esImageTag = "7.15.${env.BUILD_NUMBER}"

        fbImageName = "filebeat"

        fbImageTag = "7.15.${env.BUILD_NUMBER}"

        lsImageName = "logstash"

        lsImageTag = "7.15.${env.BUILD_NUMBER}"

        nexusUrl = "192.168.1.41:8082"

        nexusRepository = "docker-hosted"

        nexusrepourl = "http://192.168.1.41:8082/repository/docker-hosted/"

    }

    stages {

        stage('Get Code') {

            steps {

                git branch: 'main', url:
'https://github.com/Lehar1107/nexus_jenkins_docker_springboot.git'
```

```

    }

}

stage('Build Spring Boot') {

    steps {

        bat 'mvn clean install -DskipTests=true'

    }

}

stage('Build Spring Boot Docker Image') {

    steps {

        script {

            def dockerfile = """

            FROM openjdk:17-jdk

            COPY target/springboot-docker-nexus-0.0.1-SNAPSHOT.jar
/app/my-spring-boot-app.jar

            WORKDIR /app

            ENTRYPOINT ["java", "-jar", "my-spring-boot-app.jar"]

            """

            writeFile file: 'Dockerfile', text: dockerfile

            bat "docker build -t
${springBootImageName}:${springBootImageTag} ."

            bat "docker image ls
${springBootImageName}:${springBootImageTag}"

        }

    }

}

```

```

    }

    stage('Push Spring Boot Docker Image to Nexus') {

        environment {

            NEXUS_CREDENTIALS = credentials('nexus-user-credentials')

        }

        steps {

            script {

                def dockerImage =
                "${springBootImageName}:${springBootImageTag}"

                def nexusImage =
                "${nexusUrl}/${nexusRepository}/${springBootImageName}:${springBootImageTag}"

                withCredentials([usernamePassword(credentialsId: 'nexus-user-credentials', usernameVariable: 'NEXUS_USERNAME', passwordVariable: 'NEXUS_PASSWORD')]) {

                    withEnv(["DOCKER_LOGIN=${NEXUS_USERNAME}",
                    "DOCKER_PASSWORD=${NEXUS_PASSWORD}"]) {

                        bat "docker login -u $NEXUS_USERNAME -p $NEXUS_PASSWORD $nexusrepourl"

                        bat "docker tag ${dockerImage} ${nexusImage}"

                        bat "docker push ${nexusImage}"

                        bat "docker logout $nexusrepourl"

                    }

                }

            }

        }

    }

```



```

    }

}

stage('Build Elasticsearch Docker Image') {

    steps {

        script {

            def esDockerfile = """

                FROM docker.elastic.co/elasticsearch/elasticsearch:7.15.0

                # Any custom modifications to the Elasticsearch image can be
added here

            """

            writeFile file: 'Dockerfile.elasticsearch', text: esDockerfile

            bat "docker build -t ${esImageName}:${esImageTag} -f
Dockerfile.elasticsearch ."

            bat "docker image ls ${esImageName}:${esImageTag}"

        }

    }

}

stage('Push Elasticsearch Docker Image to Nexus') {

    environment {

        NEXUS_CREDENTIALS = credentials('nexus-user-credentials')

    }

    steps {

        script {

```

```

        def esDockerImage = "${esImageName}:${esImageTag}"

        def esNexusImage =
            "${nexusUrl}/${nexusRepository}/${esImageName}:${esImageTag}"

            withCredentials([usernamePassword(credentialsId: 'nexus-user-
credentials', usernameVariable: 'NEXUS_USERNAME', passwordVariable:
'NEXUS_PASSWORD')]) {

                withEnv(["DOCKER_LOGIN=${NEXUS_USERNAME}",
"DOCKER_PASSWORD=${NEXUS_PASSWORD}"]) {

                    bat "docker login -u $NEXUS_USERNAME -p
$NEXUS_PASSWORD $nexusrepourl"

                    bat "docker tag ${esDockerImage} ${esNexusImage}"

                    bat "docker push ${esNexusImage}"

                    bat "docker logout $nexusrepourl"

                }

            }

        }

    }

}

stage('Build Filebeat Docker Image') {

    steps {

        script {

            def fbDockerfile = ""

            FROM docker.elastic.co/beats/filebeat:7.15.0

            # Any custom modifications to the Filebeat image can be added
here

```

```

"""

writeFile file: 'Dockerfile.filebeat', text: fbDockerfile

bat "docker build -t ${fbImageName}:${fbImageTag} -f
Dockerfile.filebeat ."

bat "docker image ls ${fbImageName}:${fbImageTag}"

}

}

}

stage('Push Filebeat Docker Image to Nexus') {

    environment {

        NEXUS_CREDENTIALS = credentials('nexus-user-credentials')

    }

    steps {

        script {

            def fbDockerImage = "${fbImageName}:${fbImageTag}"

            def fbNexusImage =
"${nexusUrl}/${nexusRepository}/${fbImageName}:${fbImageTag}"

            withCredentials([usernamePassword(credentialsId: 'nexus-user-
credentials', usernameVariable: 'NEXUS_USERNAME', passwordVariable:
'NEXUS_PASSWORD')]) {

                withEnv(["DOCKER_LOGIN=${NEXUS_USERNAME}",
"DOCKER_PASSWORD=${NEXUS_PASSWORD}"]) {

                    bat "docker login -u $NEXUS_USERNAME -p
$NEXUS_PASSWORD $nexusrepourl"

                    bat "docker tag ${fbDockerImage} ${fbNexusImage}"

```

```

        bat "docker push ${fbNexusImage}"

        bat "docker logout $nexusrepurl"

    }

}

}

}

}

}

stage('Build logstash Docker Image') {

    steps {

        script {

            def lsDockerfile = ""

            FROM docker.elastic.co/logstash/logstash:7.15.0

            # Any custom modifications to the logstash image can be added
here

            ""

            writeFile file: 'Dockerfile.logstash', text: lsDockerfile

            bat "docker build -t ${lsImageName}:${lsImageTag} -f
Dockerfile.logstash ."

            bat "docker image ls ${lsImageName}:${lsImageTag}"

        }

    }

}

stage('Push logstash Docker Image to Nexus') {

```

```

environment {

    NEXUS_CREDENTIALS = credentials('nexus-user-credentials')

}

steps {

    script {

        def lsDockerImage = "${lsImageName}:${lsImageTag}"

        def lsNexusImage =
"${nexusUrl}/${nexusRepository}/${lsImageName}:${lsImageTag}"

        withCredentials([usernamePassword(credentialsId: 'nexus-user-
credentials', usernameVariable: 'NEXUS_USERNAME', passwordVariable:
'NEXUS_PASSWORD')]) {

            withEnv(["DOCKER_LOGIN=\${NEXUS_USERNAME}",
"DOCKER_PASSWORD=\${NEXUS_PASSWORD}"]) {

                bat "docker login -u $NEXUS_USERNAME -p
$NEXUS_PASSWORD $nexusrepurl"

                bat "docker tag ${lsDockerImage} ${lsNexusImage}"

                bat "docker push ${lsNexusImage}"

                bat "docker logout $nexusrepurl"

            }

        }

    }

}

```

```

stage('Prepare YAMLS') {

    steps {

        script {

            def elasticYamlContent = readFile('nexus/elastic-playbook.yaml')

            def filebeatYamlContent = readFile('nexus/filebeat-playbook.yaml')

            def logstashYamlContent = readFile('nexus/logstash-playbook.yaml')

            def modifiedElasticContent =
elasticYamlContent.replaceAll("\\${BUILD_NUMBER}\\",
env.BUILD_NUMBER)

            def modifiedFilebeatContent =
filebeatYamlContent.replaceAll("\\${BUILD_NUMBER}\\",
env.BUILD_NUMBER)

            def modifiedLogstashContent =
logstashYamlContent.replaceAll("\\${BUILD_NUMBER}\\",
env.BUILD_NUMBER)

            writeFile file: 'nexus/elastic-playbook.yaml', text:
modifiedElasticContent

            writeFile file: 'nexus/filebeat-playbook.yaml', text:
modifiedFilebeatContent

            writeFile file: 'nexus/logstash-playbook.yaml', text:
modifiedLogstashContent

        }

    }

}

stage('List Files') {

    steps {

```

```

    dir('nexus') {

        bat 'dir'

    }

}

stage('Deploy Applications in Kubernetes') {

    environment {

        NEXUS_CREDENTIALS = credentials('nexus-user-credentials')

    }

    steps {

        script {

            bat "kubectl --kubeconfig=C:/Users/LEHAR/.kube/config apply -f
nexus/elastic-playbook.yaml"

            bat "kubectl --kubeconfig=C:/Users/LEHAR/.kube/config apply -f
nexus/filebeat-playbook.yaml"

            bat "kubectl --kubeconfig=C:/Users/LEHAR/.kube/config apply -f
nexus/logstash-playbook.yaml"

        }

    }

}

```

Deployments Code

Elastic-playbook.yaml

apiVersion: v1

kind: ConfigMap

metadata:

name: elasticsearch-config

namespace: elastic-deploy

data:

elasticsearch.yml: |

Add your Elasticsearch configuration here

cluster.name: my-cluster

node.name: my-node

network.host: 0.0.0.0

apiVersion: apps/v1

kind: Deployment

metadata:

name: elasticsearch

namespace: elastic-deploy

spec:

selector:

matchLabels:

app: elasticsearch

template:

metadata:

labels:

app: elasticsearch

spec:

containers:

- name: elasticsearch

image: 192.168.1.41:8082/docker-
hosted/elasticsearch:7.15.\${BUILD_NUMBER}

env:

- name: ES_JAVA_OPTS

value: "-Xms512m -Xmx512m"

- name: discovery.type

value: single-node

ports:

- containerPort: 9200

name: http

- containerPort: 9300

name: transport

volumeMounts:

- name: elasticsearch-data

mountPath: /usr/share/elasticsearch/data

- name: config-volume

mountPath: /usr/share/elasticsearch/config/elasticsearch.yml

subPath: elasticsearch.yml

volumes:

- name: elasticsearch-data

emptyDir: { }

- name: config-volume

configMap:

name: elasticsearch-config

apiVersion: v1

kind: Service

metadata:

name: elasticsearch

namespace: elastic-deploy

spec:

selector:

app: elasticsearch

ports:

- name: http

protocol: TCP

port: 9200

targetPort: 9200

- name: transport

protocol: TCP

port: 9300

targetPort: 9300

apiVersion: v1

kind: Secret

metadata:

name: nexus-docker-secret

namespace: elastic-deploy

data:

.dockerconfigjson:

ew0KICAgICJhdXRocyI6IHsNCiAgICAgICxOTIuMTY4LjEuNDE6ODA4MiI
6IHsNCiAgICAgICAgInVzZXJuYW1lIjogImFkbWluIiwNCiAgICAgICAgInBhc
3N3b3JkIjogIjY4MGM0YWYzLWNhYjktNDMyOC04MjBILWEwODI4YjNlNj
FlNSIsDQogICAgICAgICJhdXRoIjogIllXUnRhVzQ2Tmpnd1l6Umhaak10WTJ
GaU9TMDBNekk0TFRneU1HVXRZVEE0TWpoaU0yVTJNV1UxIg0KICAgIC
AgfQ0KICAgIH0NCiAgfQ0KDQogIA0K

type: kubernetes.io/dockerconfigjson

Filebeat.yaml

apiVersion: v1

kind: ConfigMap

metadata:

name: my-filebeat

namespace: elastic-deploy

data:

my-file.log: |

This

You

Make

apiVersion: v1

data:

filebeat.yml: |

filebeat.inputs:

- type: log

paths:

- /usr/share/filebeat/my-file.log

output.logstash:

hosts: ['logstash:5044']

kind: ConfigMap

metadata:

name: filebeat-configmap

```
    namespace: elastic-deploy

---

apiVersion: apps/v1

kind: Deployment

metadata:

  name: filebeat

  namespace: elastic-deploy

spec:

  selector:

    matchLabels:

      app: filebeat

  template:

    metadata:

      labels:

        app: filebeat

    spec:

      containers:

        - name: filebeat

          image: 192.168.1.41:8082/docker-
hosted/filebeat:7.15.${BUILD_NUMBER}

          volumeMounts:

            - name: filebeat-config

              mountPath: /usr/share/filebeat/filebeat.yml
```

subPath: filebeat.yml

- name: my-filebeat

mountPath: /usr/share/filebeat/my-file.log

subPath: my-file.log

volumes:

- name: filebeat-config

configMap:

name: filebeat-configmap

- name: my-filebeat

configMap:

name: my-filebeat

apiVersion: v1

kind: Secret

metadata:

name: nexus-docker-secret

namespace: elastic-deploy

data:

.dockerconfigjson:

ew0KICAgICJhdXRocyI6IHsNCiAgICAgICxOTluMTY4LjEuNDE6ODA4MiI
6IHsNCiAgICAgICAgInVzZXJuYW1lIjogImFkbWluIiwNCiAgICAgICAgInBhc
3N3b3JkIjogIjY4MGM0YWYzLWNhYjktNDMyOC04MjBILWEwODI4YjNINj
FINSIsDQogICAgICAgICJhdXR0IjogIllXUnRhVzQ2Tmpnd1l6Umhaak10WTJ
GaU9TMDBNekk0TFRneU1HVXRZVEE0TWpoaU0yVTJNV1UxIg0KICAgIC
AgfQ0KICAgIH0NCiAgfQ0KDQogIA0K

type: kubernetes.io/dockerconfigjson

Logstash.yaml

apiVersion: v1

kind: ConfigMap

metadata:

name: logstash-configmap

namespace: elastic-deploy

data:

logstash.conf: |

input {

beats {

port => 5044

}

}

output {

elasticsearch {

hosts => ["http://elasticsearch:9200"]

index => "filebeat-data"

}

}

apiVersion: apps/v1

kind: Deployment

metadata:

name: logstash

namespace: elastic-deploy

spec:

selector:

matchLabels:

app: logstash

template:

metadata:

labels:

app: logstash

spec:

containers:

- name: logstash

image: 192.168.1.41:8082/docker-
hosted/logstash:7.15.\${BUILD_NUMBER}

ports:

- containerPort: 5044 # Port for receiving logs from other containers

name: logstash

volumeMounts:

- name: logstash-config

mountPath: /usr/share/logstash/pipeline/logstash.conf

subPath: logstash.conf

volumes:

- name: logstash-config

configMap:

name: logstash-configmap

kind: Service

apiVersion: v1

metadata:

name: logstash

namespace: elastic-deploy

spec:

selector:

app: logstash

ports:

- protocol: TCP

port: 5044

targetPort: 5044

apiVersion: v1

kind: Secret

metadata:

name: nexus-docker-secret

namespace: elastic-deploy

data:

.dockerconfigjson:

ew0KICAgICJhdXRocyI6IHsNCiAgICAgICxOTluMTY4LjEuNDE6ODAA4MiI
6IHsNCiAgICAgICAgInVzZXJuYW1lIjogImFkbWluIiwNCiAgICAgICAgInBhc
3N3b3JkIjogIjY4MGM0YWYyZLWNhYjktNDMyOC04MjBILWEwODI4YjNINj
FINSIsDQogICAgICAgICJhdXRoIjogIiIXUnRhVzQ2Tmpnd1l6Umhaak10WTJ
GaU9TMDBNekk0TFRneU1HVXRZVEE0TWpoaU0yVTJNV1UxIg0KICAgIC
AgfQ0KICAgIH0NCiAgfQ0KDQogIA0K

type: kubernetes.io/dockerconfigjson

Plugins of an applications

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
```

```
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
```

```
    https://maven.apache.org/xsd/maven-4.0.0.xsd">
```

```
    <modelVersion>4.0.0</modelVersion>
```

```
    <parent>
```

```
        <groupId>org.springframework.boot</groupId>
```

```
        <artifactId>spring-boot-starter-parent</artifactId>
```

```
        <version>3.2.5</version>
```

```
        <relativePath/> <!-- lookup parent from repository -->

</parent>

<groupId>com.example</groupId>

<artifactId>springboot-docker-nexus</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>springboot-docker-nexus</name>

<description>Demo project for Spring Boot</description>

<properties>

    <java.version>17</java.version>

</properties>

<dependencies>

    <dependency>

        <groupId>org.springframework.boot</groupId>

        <artifactId>spring-boot-starter-web</artifactId>

    </dependency>

    <dependency>

        <groupId>org.springframework.boot</groupId>

        <artifactId>spring-boot-starter-test</artifactId>

        <scope>test</scope>

    </dependency>

</dependencies>
```

```
<build>

  <plugins>

    <plugin>

      <groupId>org.springframework.boot</groupId>

      <artifactId>spring-boot-maven-plugin</artifactId>

    </plugin>

  </plugins>

</build>

</project>
```

Configs

```
{

  "auths": {

    "192.168.1.41:8082": {

      "username": "admin",

      "password": "680c4af3-cab9-4328-820e-a0828b3e61e5",

      "auth":
        "YWRtaW46NjgwYzRhZjMtY2FiOS00MzI4LTgyMGUtYTA4MjhiM2U2MWU1"

    }

  }

}
```

Testing

Test Cases

1. Build and Test Application

- Objective: Ensure the Spring Boot application builds successfully.
- Test Case ID: TC01
- Preconditions: Code is committed to Git repository.
- Steps:
 1. Trigger a build in Jenkins.
 2. Monitor the build process.
 3. Check build logs for errors.
- Expected Result: Application builds successfully without errors.

2. Create Docker Image

- Objective: Verify Docker image creation.
- Test Case ID: TC02
- Preconditions: Successful build of the application.
- Steps:
 1. Jenkins triggers Docker image creation.
 2. Monitor Docker image creation process.
 3. Check Docker logs for errors.
- Expected Result: Docker image is created successfully.

3. Push Docker Image to Nexus

- Objective: Ensure Docker image is pushed to Nexus repository.

- Test Case ID: TC03
- Preconditions: Successful Docker image creation.
- Steps:
 1. Jenkins pushes Docker image to Nexus.
 2. Check Nexus repository for the new image.
 3. Verify image integrity and details.
- Expected Result: Docker image is successfully pushed to Nexus.

4. Deploy to Kubernetes

- Objective: Confirm deployment of Docker image to Kubernetes.
- Test Case ID: TC04
- Preconditions: Docker image available in Nexus repository.
- Steps:
 1. Kubernetes pulls Docker image from Nexus.
 2. Deploy image as pods in Kubernetes cluster.
 3. Check pod status and logs.
- Expected Result: Docker image is deployed successfully as Kubernetes pods.

5. Monitor with ArgoCD

- Objective: Verify ArgoCD monitoring and updates.
- Test Case ID: TC05
- Preconditions: Deployment in Kubernetes is successful.
- Steps:
 1. Make a new commit in Git repository.

2. ArgoCD detects the new commit.
 3. ArgoCD updates the deployment in Kubernetes.
- Expected Result: ArgoCD successfully monitors and updates the deployment.

User Interfaces

1. Jenkins Interface:

- Purpose: To monitor build and deployment processes.
- Components: Dashboard, Job Configurations, Console Output, Build History.
- Features: Trigger builds, view logs, configure jobs.

2. Nexus Repository Manager:

- Purpose: To store and manage Docker images.
- Components: Repository Browser, Upload/Download Artifacts, User Management.
- Features: View repositories, manage images, set up access control.

3. Kubernetes Dashboard:

- Purpose: To manage and monitor Kubernetes clusters.
- Components: Nodes, Pods, Deployments, Services.
- Features: View cluster status, manage deployments, monitor resource usage.

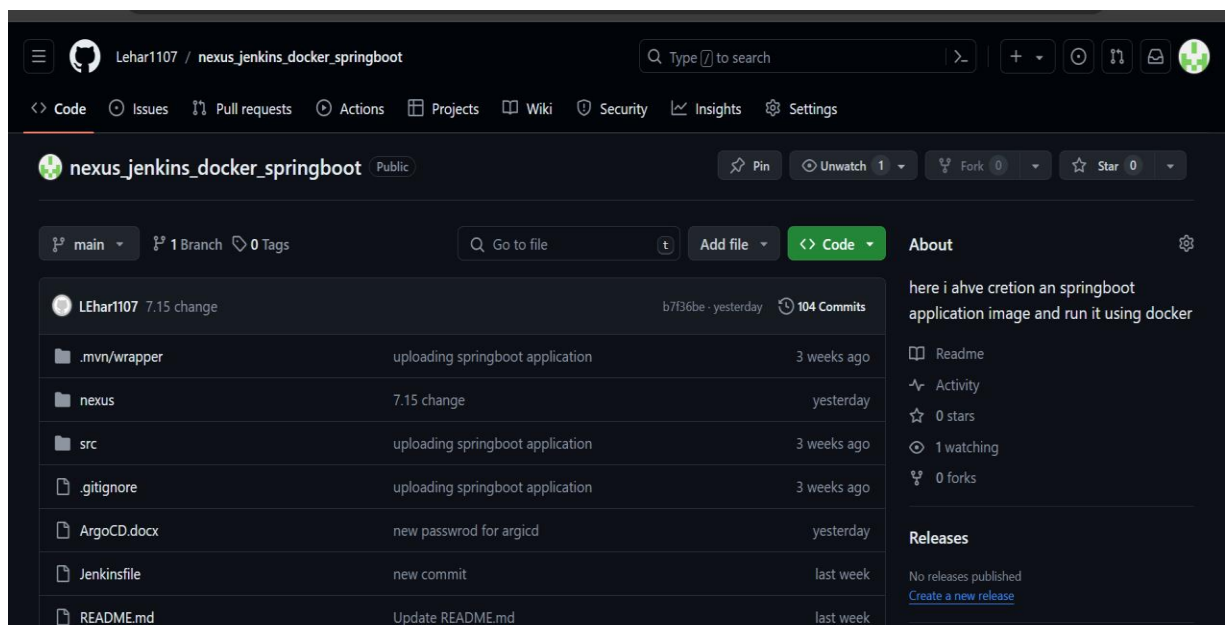
4. ArgoCD Interface:

- Purpose: To monitor and manage continuous deployment.
- Components: Applications, Projects, Sync Status, Event Logs.

- Features: View application status, trigger syncs, monitor deployment status.

Pictures

GitHub



Through Jenkins

Dashboard > deploy_elkStack_on_kubernetes > Configuration

Configure

General

Advanced Project Options

Pipeline

Advanced

Pipeline

Definition

Pipeline script

Script

```
43 ~
44 /
45 stage('Push Spring Boot Docker Image to Nexus') {
46   environment {
47     NEXUS_CREDENTIALS = credentials('nexus-user-credentials')
48   }
49   steps {
50     script {
51       def dockerImage = "${springBootApplication}:${springBootImageTag}"
52       def nexusImage = "${nexusUrl}/${nexusRepository}/${springBootApplication}:${springBootImageTag}"
53       withCredentials([usernamePassword(credentialsId: 'nexus-user-credentials', usernameVariable: 'NEXUS_USERNAME', passwordVariable: 'NEXUS_PASSWORD')]) {
54         bat "docker login -u $NEXUS_USERNAME -p $NEXUS_PASSWORD $nexusRepository"
55         bat "docker tag $dockerImage $nexusImage"
56         bat "docker push $nexusImage"
57       }
58     }
59   }
60 }
```

☒ Use Groovy Sandbox

Run

Dashboard > deploy_elkStack_on_kubernetes

Status

Changes

Build Now

Configure

Delete Pipeline

Full Stage View

Rename

Pipeline Syntax

deploy_elkStack_on_kubernetes

Add description

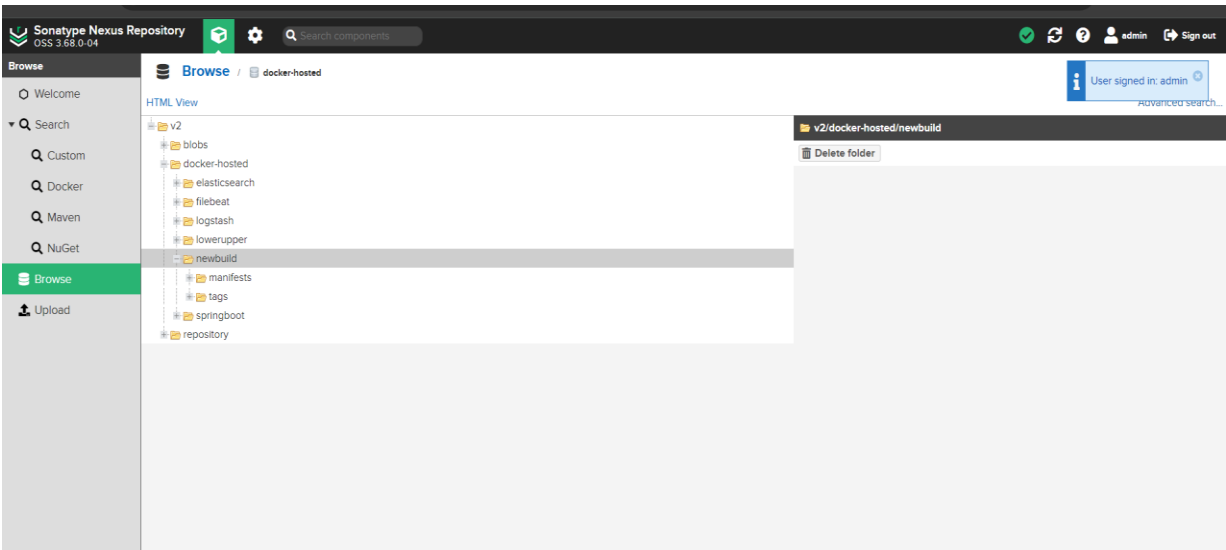
Disable Project

Stage View

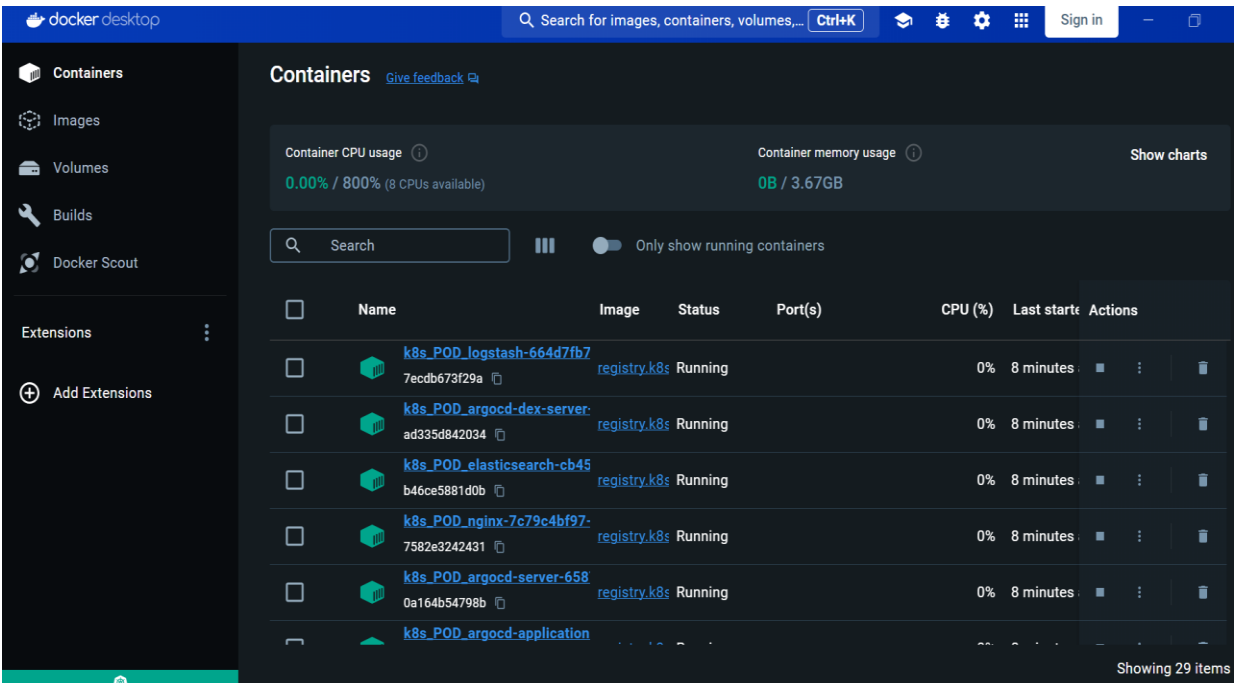
Average stage times:
(Average full run time: ~2min 52s)

	Get Code	Build Spring Boot	Build Spring Boot Docker Image	Push Spring Boot Docker Image to Nexus	Build Elasticsearch Docker Image	Push Elasticsearch Docker Image to Nexus	Build Filebeat Docker Image	Push Filebeat Docker Image to Nexus	Build logstash Docker Image	Push logstash Docker Image to Nexus	Prepare YAMLS	List Files	Deploy Applications in Kubernetes
#17	2s	9s	9s	6s	5s	2s	3s	2s	3s	2s	499ms	466ms	3s
#16	1s	8s	11s	5s	4s	2s	4s	2s	4s	2s	480ms	496ms	3s
#15	2s	10s	9s	5s	5s	2s	4s	2s	5s	3s	486ms	525ms	3s
#14	3s	9s	8s	6s	6s	3s	4s	2s	4s	3s	521ms	517ms	5s

Image in Repository



Docker Desktop is running or not



Check Pods and deployment in Powershell

```
PS C:\Users\LEHAR\Docke\argocd> kubectl get pods -n elastic-deploy
NAME                                READY   STATUS    RESTARTS   AGE
elasticsearch-cb45cc95f-qthgs      1/1     Running   1 (3m28s ago)    26h
filebeat-66f8b59bcf-jszbp         1/1     Running   1 (3m28s ago)    26h
logstash-664d7fb7b-k65tg           1/1     Running   1 (3m28s ago)    26h
PS C:\Users\LEHAR\Docke\argocd> kubectl get deploy -n elastic-deploy
NAME            READY   UP-TO-DATE   AVAILABLE   AGE
elasticsearch   1/1     1             1            26h
filebeat        1/1     1             1            26h
logstash        1/1     1             1            26h
```

ConfigMaps

```
logstash-664d7fb7b-k65tg 1/1 1 1 26h
PS C:\Users\LEHAR\Docke\argocd> kubectl get cm -n elastic-deploy
NAME          DATA   AGE
elasticsearch 1       27h
filebeat-configmap 1       27h
kube-root-ca.crt 1       27h
logstash-configmap 1       27h
my-filebeat    1       27h
PS C:\Users\LEHAR\Docke\argocd> kubectl get svc -n elastic-deploy
NAME            TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)          AGE
elasticsearch   ClusterIP   10.96.154.181 <none>        9200/TCP,9300/TCP 27h
logstash        ClusterIP   10.101.171.164 <none>        5044/TCP          27h
```

Services

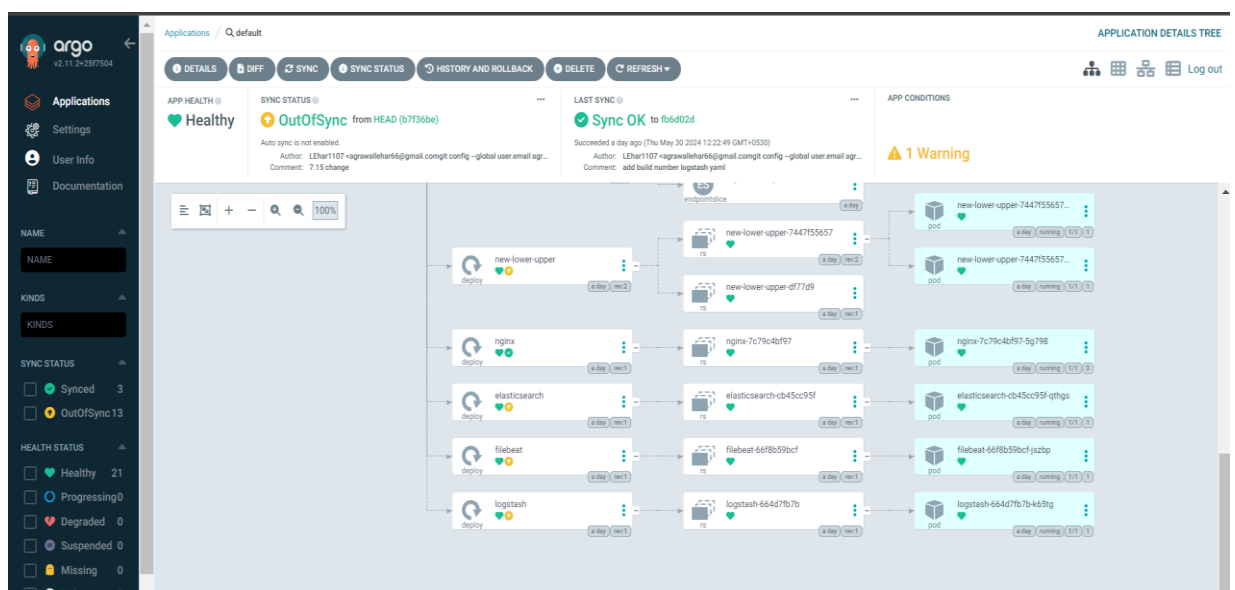
```
PS C:\Users\LEHAR\Docke\argocd> kubectl get svc -n argocd
NAME                                TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)          AGE
argocd-applicationset-controller    ClusterIP   10.109.232.213 <none>        7000/TCP,8080/TCP 27h
argocd-dex-server                   ClusterIP   10.100.250.48 <none>        5556/TCP,5557/TCP,5558/TCP 27h
argocd-metrics                      ClusterIP   10.109.29.99 <none>        8082/TCP          27h
argocd-notifications-controller-metrics ClusterIP   10.107.212.55 <none>        9001/TCP          27h
argocd-redis                       ClusterIP   10.102.165.133 <none>        6379/TCP          27h
argocd-repo-server                  ClusterIP   10.101.112.174 <none>        8081/TCP,8084/TCP 27h
argocd-server                       ClusterIP   10.111.33.75 <none>        80/TCP,443/TCP    27h
argocd-server-metrics               ClusterIP   10.101.169.105 <none>        8083/TCP          27h
lower-upper                         LoadBalancer 10.103.248.74 localhost      3000:31921/TCP    27h
nginx-service                       ClusterIP   10.99.40.177 <none>        80/TCP            27h
PS C:\Users\LEHAR\Docke\argocd> kubectl get pods -n argocd
NAME                                READY   STATUS    RESTARTS   AGE
argocd-application-controller-0      1/1     Running   3 (3m57s ago)    27h
argocd-applicationset-controller-7dc76d94b4-lq8rn 1/1     Running   3 (3m57s ago)    27h
argocd-dex-server-85f7d69f5f-vcffg  1/1     Running   3 (3m57s ago)    27h
argocd-notifications-controller-7ffccb884-p86gt  1/1     Running   3 (3m57s ago)    27h
argocd-redis-8564565c6b-pvv68       1/1     Running   3 (3m57s ago)    27h
argocd-repo-server-b9c7b5f85-9w8hn  1/1     Running   3 (3m57s ago)    27h
argocd-server-6587989799-qrpqs       1/1     Running   3 (3m57s ago)    27h
new-lower-upper-7447f55657-sldgf     1/1     Running   1 (3m57s ago)    23h
new-lower-upper-7447f55657-tvd8b    1/1     Running   1 (3m57s ago)    23h
nginx-7c79c4bf97-5g798              1/1     Running   3 (3m57s ago)    27h
```

```


PS C:\Users\LEHAR\Docke\argocd> kubectl get deploy -n argocd
NAME                                READY    UP-TO-DATE    AVAILABLE    AGE
argocd-applicationset-controller    1/1      1              1            27h
argocd-dex-server                   1/1      1              1            27h
argocd-notifications-controller    1/1      1              1            27h
argocd-redis                        1/1      1              1            27h
argocd-repo-server                 1/1      1              1            27h
argocd-server                      1/1      1              1            27h
new-lower-upper                   2/2      2              2            27h
nginx                              1/1      1              1            27h
PS C:\Users\LEHAR\Docke\argocd>

```

ArgoCD – Deployment



Data in ElasticVue

 Elasticvue

default cluster

HOME

NODES


SHARDS


INDICES

SEARCH

REST


SNAPSHOTS








Indices 



NEW INDEX


[Index templates](#)

Filter... 





<input type="checkbox"/>	Name 	Health	Status	UUID	Aliases	Shards	Segments	Docs	Storage	Created	
<input type="checkbox"/>	filebeat-data	yellow	open	zhZDdXdWQVYVYojesVn34Q	[]	1p 1r	1	3	14.5 kB	5/30/2024, 1:13:12 PM	  

Records per page 10 1-1 of 1  

BULK ACTION 

Data

Filter CURRENT PAGE only  

<input type="checkbox"/>	_index	_type	_id	_score	@timestamp	@version	agent
<input type="checkbox"/>	filebeat-data	_doc	iiNzyI8BcIAfYsmW9CzN	1	5/30/2024, 1:11:39 PM	1	{"name":"filebeat-66f8b59bcf-jszbp","type":"filebeat"}
<input type="checkbox"/>	filebeat-data	_doc	iyNzyI8BcIAfYsmW9Czb	1	5/30/2024, 1:11:39 PM	1	{"name":"filebeat-66f8b59bcf-jszbp","type":"filebeat"}
<input type="checkbox"/>	filebeat-data	_doc	jCNzyI8BcIAfYsmW9Czb	1	5/30/2024, 1:11:39 PM	1	{"name":"filebeat-66f8b59bcf-jszbp","type":"filebeat"}
<input type="checkbox"/>	filebeat-data	_doc	YbYLzo8BMKD8ITis6Dgj	1	5/31/2024, 3:14:03 PM	1	{"ephemeral_id":"1d682332-fbdc-4322-86b9-406c6394da5"}
<input type="checkbox"/>	filebeat-data	_doc	YrYLzo8BMKD8ITis6DhE	1	5/31/2024, 3:14:03 PM	1	{"ephemeral_id":"1d682332-fbdc-4322-86b9-406c6394da5"}
<input type="checkbox"/>	filebeat-data	_doc	Y7YLzo8BMKD8ITis6DhE	1	5/31/2024, 3:14:03 PM	1	{"ephemeral_id":"1d682332-fbdc-4322-86b9-406c6394da5"}

Appendices

1. Appendix A: Acronyms and Abbreviations

- CI/CD: Continuous Integration/Continuous Deployment
- SCM: Source Code Management
- VCS: Version Control System
- UI: User Interface
- SDS: System Design Specification

2. Appendix B: Tools and Technologies Used

- Jenkins: An open-source automation server.
- Docker: A platform for developing, shipping, and running applications in containers.
- Maven: A build automation tool for Java projects.
- Nexus: A repository manager for managing binary artifacts.
- Kubernetes: An open-source system for automating the deployment, scaling, and management of containerized applications.
- ArgoCD: A declarative, GitOps continuous delivery tool for Kubernetes.

3. Appendix C: Glossary

- Pipeline: A set of automated processes to manage the lifecycle of an application.
- Pod: The smallest deployable units of computing in Kubernetes.
- Repository: A central location in which data is stored and managed.
- Commit: Saving changes to the source code repository.

References

1. Books and Articles:

- Humble, Jez, and David Farley. Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. Addison-Wesley, 2010.
- Burns, Brendan, et al. Kubernetes: Up & Running: Dive into the Future of Infrastructure. O'Reilly Media, 2019.

2. Websites:

- Jenkins: <https://www.jenkins.io/>
- Docker: <https://www.docker.com/>
- Nexus: <https://www.sonatype.com/nexus/repository-oss>
- Kubernetes: <https://kubernetes.io/>
- ArgoCD: <https://argoproj.github.io/argo-cd/>

3. Documentation:

- Jenkins Documentation: <https://www.jenkins.io/doc/>
- Docker Documentation: <https://docs.docker.com/>
- Nexus Repository OSS Documentation:
<https://help.sonatype.com/repomanager3>
- Kubernetes Documentation: <https://kubernetes.io/docs/home/>
- ArgoCD Documentation: <https://argo-cd.readthedocs.io/en/stable/>