

# Configuration Manual

MSc. Research Project  
Cloud Computing

Mohammad Amaan Shaikh  
Student ID: X23186925

School of Computing  
National College of Ireland

Supervisor: Dr. Ahmed Makki

**National College of Ireland  
Project Submission Sheet  
School of Computing**



<b>Student Name:</b>	Mohammad Amaan Shaikh
<b>Student ID:</b>	X23186925
<b>Programme:</b>	Cloud Computing
<b>Year:</b>	2024
<b>Module:</b>	MSc. Research Project
<b>Supervisor:</b>	Dr. Ahmed Makki
<b>Submission Due Date:</b>	12/12/2024
<b>Project Title:</b>	Configuration Manual
<b>Word Count:</b>	1800
<b>Page Count:</b>	12

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

**ALL** internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

<b>Signature:</b>	
<b>Date:</b>	12th December 2024

**PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:**

Attach a completed copy of this sheet to each project (including multiple copies).	<input type="checkbox"/>
<b>Attach a Moodle submission receipt of the online project submission</b> , to each project (including multiple copies).	<input type="checkbox"/>
<b>You must ensure that you retain a HARD COPY of the project</b> , both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	<input type="checkbox"/>

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

<b>Office Use Only</b>	
Signature:	
Date:	
Penalty Applied (if applicable):	

# Configuration Manual

Mohammad Amaan Shaikh  
X23186925

## 1 Introduction

Introduction: This configuration manual provides detailed steps that were utilized to set up and automate the robust CI/CD pipeline. It provides detailed understanding of configuration and integration steps utilized. Additionally, it provides the downloading and installation steps for each tool used and provides the details of configuration and OS image used for each server. This configuration manual ensures instructions are cleared for the future researcher who wishes to contribute to the study.

## 2 Configuration of EKS Cluster

### 2.1 Step 1: IAM Role for EKS Cluster:

- Go to IAM console in AWS
- Roles > create role
- Choose EKS service from the available AWS service list.
- Click next and attach policy: AmazonEKSClusterPolicy

### 2.2 Step 2: IAM Role for Node Groups (Worker Nodes)

- Go to IAM console
- Roles > create roles.
- Choose EC2 service from the available AWS service list.
- Attach the below policies to IAM role:
  - AmazonEKSWorkerNodePolicy
  - AmazonEC2ContainerRegistryReadOnly
  - AmazonEKS\_CNI\_Policy

## 2.3 Step 3: Cluster creation

- From the AWS Console create AWS master node or EKS cluster, for this study the default VPC and subnets were used and AmazonEKSClusterPolicy created in step 1 was attached to EKS cluster.
- Once the EKS cluster is ready, under the compute section of the newly created EKS cluster, create a new node group and attach it to the cluster. Below is the detailed configuration of node group that was utilized and attached to the EKS cluster.

For this study Amazon Linux 2 instance of instance type t3.medium was configured and desired size of 2, minimum size of 1, Maximum size of 5 was added. This ensures a stable starting point to operate and at increased workloads nodes can be scaled to maximum size of 5 making the setup scalable. Additionally, IAM role created in step 2 was attached to giving permission to worker node to interact with ECR registry, container related permissions, and worker node policies.

The image shows two panels from the AWS Management Console for configuring an EKS Node Group.

**Node group compute configuration**  
These properties cannot be changed after the node group is created.

- AMI type** Info: Select the EKS-optimized Amazon Machine Image for nodes.   
Dropdown menu: Amazon Linux 2 (AL2\_x86\_64)
- Capacity type**: Select the capacity purchase option for this node group.   
Dropdown menu: On-Demand
- Instance types** Info: Select instance types you prefer for this node group.   
Search bar: Enter an instance type   
Selected: t3.medium (vCPU: 2 vCPUs, Memory: 4 GiB, Network: Up to 5 Gbps, Max ENI: 3, Max IPI: 18)
- Disk size**: Select the size of the attached EBS volume for each node.   
Input: 20 GiB

**Node group scaling configuration**

- Desired size**: Set the desired number of nodes that the group should launch with initially.   
Input: 2 nodes   
Note: Desired node size must be greater than or equal to 0
- Minimum size**: Set the minimum number of nodes that the group can scale in to.   
Input: 1 nodes   
Note: Minimum node size must be greater than or equal to 0
- Maximum size**: Set the maximum number of nodes that the group can scale out to.   
Input: 5 nodes   
Note: Maximum node size must be greater than or equal to 1 and cannot be lower than the minimum size

Figure 1: EKS Worker Node Configurations

## 3 Jenkins Server setup:

To set up a Jenkins server Amazon linux AMI, with instance type t2.medium was used. A key pair named x23186925-thesis.pem was generated and attached to this Jenkins server.

- Step 1: `ssh ec2-user@server-ip -i x23186925-thesis.pem`
- Step 2: Jenkins has a software requirement of Java to run. Hence utilizing below command specific version of java can be installed.
  - `sudo amazon-linux-extras enable corretto8`
  - `sudo yum install java-1.8.0-amazon-corretto -y`
- Step 3: Once the installation is complete, verify installation of java using command: `java -version`
- Step 4: Furthermore, once the java is installed successfully, using below command one can install Long Term Support release of Jenkins.



## 4 Setting up S3 bucket

Amazon S3 was used to store the Helm charts using OCI mechanism and store the report generated by KubeScore. To setup the Amazon S3 buckets below steps were followed.

- Navigate to AWS Management console, navigate to S3 console and click on create bucket as shown in below Figure 4. Enter the name of Bucket, and keep the versioning enabled and finally click on the create button.

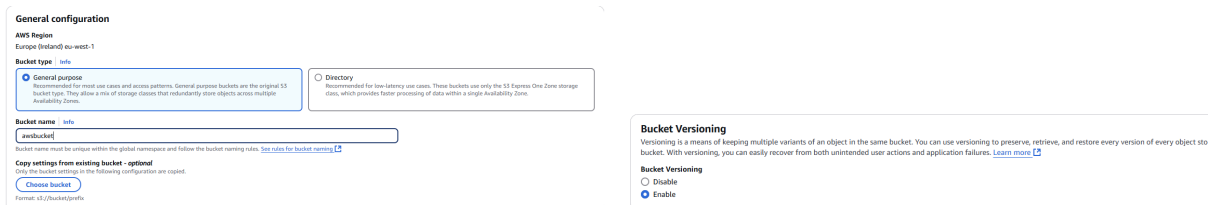


Figure 4: S3 Bucket configuration

- Two bucket's named: x23186925-kubescape-results and x23186925-thesis-helm-charts were created in the eu-west-1 region

## 5 Installation of kubectl and Helm:

Below are the steps to install Kubectl and Helm.

### 5.1 Installing Kubectl

- kubectl was used to interact with the EKS cluster. Below are the command to Download and verify the installation of kubectl.
  - `curl -LO "https://dl.k8s.io/release/v1.28.1/bin/linux/amd64/kubectl"` (Replace v1.28.1 with the desired version)
  - `chmod +x ./kubectl` (Providing kubectl execution permission)
  - `sudo mv ./kubectl /usr/local/bin/kubectl` (Moving kubectl to System PATH)
  - `kubectl version --client --output=yaml` (To verify installation)

### 5.2 Installing Helm

- Helm was installed using an installer script that grabs the latest version of Helm and installs the Helm after execution of script. Below are the commands to fetch the script and install helm.
  - `curl -fsSL -o get_helm.sh https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3`
  - `chmod 700 get_helm.sh`
  - `./get_helm.sh`
  - `helm version` (To verify the installation of Helm)

## 6 Installation of KubeScore and Sonar-Scanner

### 6.1 KubeScore setup:

- KubeScore was used to analyze the Helm chart configuration. KubeScore generated a report, which was further studied and changes in Helm chart configuration were done. Application was redeployed using Helm increasing the reliability and security. Below are the steps to install KubeScore.
  - `curl -Lo kube-score https://github.com/zegl/kube-score/releases/download/v1.19.0/kube-score_1.19.0_linux_amd64`
  - `chmod +x kube-score`
  - `sudo mv kube-score /usr/local/bin/`
  - kube-score version)

### 6.2 Sonar-Scanner setup:

- `sonarScannerRepo="https://binaries.sonarsource.com/Distribution/sonar-scanner-cli/"`
- `sonarScannerZip=(curl -s $sonarScannerRepo -- grep "sonar-scanner-cli-.*linux.zip" -- sed 's/.*"*/p;d')`
- `curl -sS -O $sonarScannerRepo/sonarScannerZip`
- `wget https://binaries.sonarsource.com/Distribution/sonar-scanner-cli/sonar-scanner-cli-6.2.1.4610-linux-x64.zip`
- `unzip sonar-scanner-cli-6.2.1.4610-linux-x64.zip -d /opt`
- `mv /opt/sonar-scanner-6.2.1.4610-linux-x64/ /opt/sonar-scanner`
- `export PATH=$PATH:/opt/sonar-scanner/bin`
- `sonar-scanner -version`

## 7 Kubernetes configuration file and Jenkins pipeline.

Below are the steps that were used to create the Helm charts and configuration employed.

- Step1: First create a Helm chart using below command
  - `helm create nginx-app`
- Step2: Navigate to the chart created
  - `cd nginx-app`
- Step3: Below Figure 5 is the Directory structure of Helm chart.

```
[root@ip-172-31-20-58 nginx-app]# tree
.
├── Chart.yaml
├── charts
├── templates
│   ├── NOTES.txt
│   ├── _helpers.tpl
│   ├── deployment.yaml
│   ├── hpa.yaml
│   ├── ingress.yaml
│   ├── networkpolicy.yaml
│   ├── pdb.yaml
│   ├── service.yaml
│   ├── serviceaccount.yaml
│   └── tests
│       └── test-connection.yaml
└── values.yaml

3 directories, 12 files
```

Figure 5: Helm Directory Structure

```
[root@ip-172-31-20-58 templates]# cat deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: {{ include "nginx-app.fullname" . }}
  labels:
    app: {{ .Values.appName }}
spec:
  # replicas: {{ .Values.replicaCount }} # Leave out if using HPA
  selector:
    matchLabels:
      app: {{ .Values.appName }}
  template:
    metadata:
      labels:
        app: {{ .Values.appName }}
    spec:
      containers:
        - name: {{ .Values.appName }}
          image: {{ .Values.image.repository }}:{{ .Values.image.tag }}
          ports:
            - containerPort: {{ .Values.containerPort }}
          readinessProbe:
            httpGet:
              path: /
              port: {{ .Values.containerPort }}
            initialDelaySeconds: 5
            periodSeconds: 10
          securityContext:
            runAsUser: 1000
            runAsGroup: 3000
            allowPrivilegeEscalation: false
          resources:
            requests:
              cpu: "500m"
              memory: "512Mi"
            limits:
              cpu: "1000m"
              memory: "1024Mi"
            ephemeral-storage: "2Gi"
          imagePullSecrets:
            - name: {{ .Values.imagePullSecrets }}
```

```
[root@ip-172-31-20-58 nginx-app]# cat values.yaml
appName: nginx

replicaCount: 3

image:
  repository: 250738637992.dkr.ecr.eu-west-1.amazonaws.com/x23186925-thesis
  tag: latest
  pullPolicy: IfNotPresent

containerPort: 8000

imagePullSecrets: ecr-secret

service:
  type: NodePort
  port: 80
  nodePort: 30009

autoscaling:
  enabled: true
  minReplicas: 1
  maxReplicas: 5
  targetCPUUtilizationPercentage: 80

serviceAccount:
  create: true
  name: ""

ingress:
  enabled: false
  annotations: {}
  hosts:
    - host: chart-example.local
      paths:
        - /
  tls: []
```

Figure 6: Configuration of Deployment.yaml and values.yaml file

```
[root@ip-172-31-20-58 templates]# cat ingress.yaml
{{- if .Values.ingress.enabled }}
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: {{ include "nginx-app.fullname" . }}
  labels:
    {{- include "nginx-app.labels" . | nindent 4 }}
  annotations:
    {{- with .Values.ingress.annotations }}
    {{- toYaml . | nindent 4 }}
    {{- end }}
spec:
  rules:
    {{- range .Values.ingress.hosts }}
    - host: {{ .host }}
      http:
        paths:
          {{- range .paths }}
          - path: {{ . }}
            pathType: ImplementationSpecific
            backend:
              service:
                name: {{ include "nginx-app.fullname" . }}
                port:
                  number: {{ .Values.service.port }}
          {{- end }}
        {{- end }}
    {{- end }}
  {{- if .Values.ingress.tls }}
  tls:
    {{- range .Values.ingress.tls }}
    - hosts:
        {{- range .hosts }}
        - {{ . }}
        {{- end }}
        secretName: {{ .secretName }}
    {{- end }}
    {{- end }}
  {{- end }}
{{- end }}
```

```
[root@ip-172-31-20-58 templates]# cat hpa.yaml
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: {{ include "nginx-app.fullname" . }}-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: {{ include "nginx-app.fullname" . }}
  minReplicas: {{ .Values.autoscaling.minReplicas }}
  maxReplicas: {{ .Values.autoscaling.maxReplicas }}
  metrics:
    - type: Resource
      resource:
        name: cpu
        target:
          type: Utilization
          averageUtilization: {{ .Values.autoscaling.targetCPUUtilizationPercentage }}
```

Figure 7: Configuration of Deployent.yaml and values.yaml file

```
[root@ip-172-31-20-58 templates]# cat service.yaml
apiVersion: v1
kind: Service
metadata:
  name: {{ include "nginx-app.fullname" . }}
spec:
  type: {{ .Values.service.type }} # Update 'Values.yaml' to set this to ClusterIP or LoadBalancer if possible
  selector:
    app: {{ .Values.appName }}
  ports:
    - port: {{ .Values.service.port }}
      targetPort: {{ .Values.containerPort }}
      nodePort: {{ .Values.service.nodePort }}
```

Figure 8: Helm Directory Structure

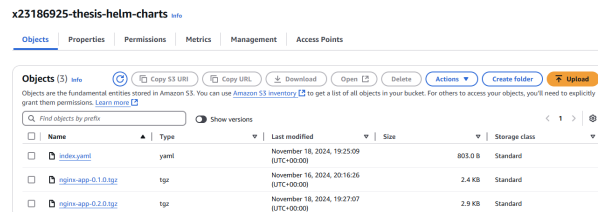
- Step4: Based on the suggestions or analysis made by KubeScore Chart.yaml, values.yaml, deployment.yaml, service.yaml, hpa.yaml, ingress.yaml file were created and updated to resolve CRITICAL issues.
- `aws eks --region eu-west-1 update-kubeconfig --name x23186925-thesis --profile MSCCLOUD-250738637992` (This command creates a kubeconfig file essential to manage the kubernetes cluster at path: `/.kube/config`)
- Step 5: Package and deploy the chart to EKS cluster:
  - Once the Helm chart is created, it was package and first deployed manually using command:
  - `helm package .`
  - `helm install my-releas ./nginx-app` (Utilizes credentials stored in `/.kube/config` file)

## 8 Helm charts versioning

By export `'HELM_EXPERIMENTAL_OCI=1'` you enable Open Container Initiative of Helm. It let you store, pull and manage Helm charts as OCI artifacts. To use S3 bucket as a Helm repository, below steps were followed.

- Step1: Package your Helm chart, depending on the name of chart and version number present in Chart.yaml ".tgz" file will be created, binded with specific version
- Step2: Upload the packaged chart (.tgz file) to the S3 bucket you created in Section 2 (x23186925-thesis-helm-charts) using command
  - `aws s3 cp nginx-app-0.1.0.tgz s3://your-bucket-name/`
- Step3: Generate index.yaml file for the repository
- Index.yaml file will contain metadata about the charts, each time the Helm chart is updated, index.yaml should be also edited. Updating index.yaml will reflect the versions of helm in it as a metadata.
  - `helm repo index . --url https://x23186925-thesis-helm-charts.s3.amazonaws.com`
- Step4: Upload the index.yaml file to your s3 bucket using command:
  - `aws s3 cp index.yaml s3://x23186925-thesis-helm-charts/`
- Step5: Now that your .tgz file of Helm chart and index.yaml file are in S3 bucket configure add S3 bucket as Helm repository.
  - `helm repo add my-s3-repo https://x23186925-thesis-helm-charts.s3.amazonaws.com`
- Step6: Use the Repository

- helm repo update (Update the Helm Repository)
- helm search repo my-s3-repo (Search for the chart)
- helm install my-releas my-s3-repo/nginx-app --version 0.1.0 (Install the chart)
- helm upgrade my-releas my-s3-repo/nginx-app --version 0.1.0 (upgrade the chart)



Name	Type	Last modified	Size	Storage class
index.yaml	yaml	November 18, 2024, 19:25:09 (UTC+00:00)	803.0 B	Standard
nginx-app-0.1.0.tgz	tgz	November 16, 2024, 20:16:26 (UTC+00:00)	2.4 KB	Standard
nginx-app-0.2.0.tgz	tgz	November 18, 2024, 19:27:07 (UTC+00:00)	2.9 KB	Standard

Figure 9: Versions of Helm chart

## 9 Sonar Cloud Configuration

Below are the steps to setup quality gates and SonarCloud project.

- Step1: SonarCloud Account
  - At <https://www.sonarsource.com/products/sonarcloud/> sign up using preferred method(Github,Gitlab, Bitbucket etc). Once signup is successful, Log in to the SonarCloud dashboard.
- Step2: Create a new Organization and project
  - From the sonar cloud dashboard Click + Create an organization
  - Select version control system as Github
  - Follow the instruction and Click Create Organization
- Step3: Once the Organization is created, create the Project using
  - Navigate to project and Click + add a project
  - Choose the repository from the available repositories.
  - Also copy the provided sonar-project.properties configuration file. This file will be used by CI/CD environment to trigger the analysis
- Step3: Configure quality gates
  - From your dashboard, go to Quality Gates section
  - Click on Create to create a new quality gate
  - Figure 10 shows the conditions for Quality gates that were defined.
  - And finally save the quality gates
  - Once the quality gates are created, Go to Project Settings– Quality Gate. and select the newly created quality gate to apply Quality Gate to the project.

Metric	Operator	Value
Coverage	is less than	80.0%
Duplicated Lines (%)	is greater than	3.0%
Maintainability Rating	is worse than	A
Reliability Rating	is worse than	A
Security Hotspots Reviewed	is less than	100%
Security Rating	is worse than	A

Figure 10: Quality Gate Conditions

## 10 Setup of Jenkins pipeline and final configurations

- Credential Handling: GitHub Token, AWS Access Key, AWS Secret Access Key, SonarToken are securely stored in Jenkins Credentials Manager. These credentials are used in pipeline stages to authenticate different tools and services.
- This stage of pipeline authenticates with AWS, updates the kubeconfig file, providing access to EKS cluster. Additionally, using GitHub Token, GitHub repository is cloned on Jenkins server.

```
stage('Perform AWS Actions') {
  steps {
    withCredentials([string(credentialsId: 'GITHUB_TOKEN', variable: 'GITHUB_TOKEN')]) {
      sh '''
        aws sts get-caller-identity
        aws eks --region eu-west-1 update-kubeconfig --name x23186925-thesis
        git clone https://$GITHUB_TOKEN@github.com:mdamaan08/msc-thesis-2024.git
        #kubectl apply -f /var/lib/jenkins/workspace/testtt/msc-thesis/Deployment.yml
        #kubectl apply -f /var/lib/jenkins/workspace/testtt/msc-thesis/service.yml
      '''
    }
  }
}
```

Figure 11: Code Cloning and updation of kubernetes credentials

- This stage of pipeline navigates to sonar-project.properties file and triggers sonar scanner for static code analysis.

```
stage('Static Analysis of Code'){
  steps{
    sh '''
      cd /var/lib/jenkins/workspace/testtt/msc-thesis-2024
      sonar-scanner
    '''
  }
}
```

Figure 12: Static Analysis stage

Sonar-project.properties file was stored at a root of the project in GitHub, below are the configuration of sonar-project.properties file

- This stage authenticates with Amazon Elastic Container Registry (ECR) for pulling and pushing the application image

```

msc-thesis-2024 / sonar-project.properties
root add sonarcloud properties folder ✓

Code Blame 8 lines (8 loc) · 328 Bytes Code 55% faster with GitHub Copilot
1 sonar.projectKey=mdamaan08_msc-thesis-2024
2 sonar.organization=msc-thesis
3 # Pointing sources to the examples directory
4 sonar.sources=examples
5 # If you want to include all file types in the examples directory
6 sonar.inclusions=examples/**/*
7 sonar.host.url=https://sonarcloud.io
8 sonar.login=c087c44f9328e859d0821befc7062ebc9d9a22a2

```

Figure 13: sonar-project properties file

```

stage('ECR Authentication'){
  steps(
    sh ...
    ...
    ...
  )
}

```

aws ecr get-login-password --region eu-west-1 | docker login --username AWS --password-stdin 250738637992.dkr.ecr.eu-west-1.amazonaws.com

cd /var/lib/jenkins/workspace/testtt/msc-thesis-2024/examples/python-web-app

Figure 14: ECR authentication stage

```

msc-thesis-2024 / examples / python-web-app / Dockerfile
root 2nd commit

Code Blame 19 lines (10 loc) · 299 Bytes Code 55% faster with GitHub Copilot
1 FROM ubuntu
2
3 WORKDIR /app
4
5 COPY requirements.txt /app
6 COPY devops /app
7
8 RUN apt-get update && \
9     apt-get install -y python3 python3-pip && \
10     pip install --break-system-packages -r requirements.txt && \
11     cd devops
12
13
14 ENTRYPOINT ["python3"]
15 CMD ["manage.py", "runserver", "0.0.0.0:8000"]

```

Figure 15: DockerFile

```

stage('Docker Image build and publishing'){
  steps(
    sh ...
    ...
    ...
  )
}

```

cd /var/lib/jenkins/workspace/testtt/msc-thesis-2024/examples/python-web-app

docker build -t x23186925-thesis .

docker tag x23186925-thesis:latest 250738637992.dkr.ecr.eu-west-1.amazonaws.com/x23186925-thesis:latest

docker push 250738637992.dkr.ecr.eu-west-1.amazonaws.com/x23186925-thesis:latest

Figure 16: Build and Publish Docker image stage

- Build and Publish Docker Image stage: The pipeline builds a Docker image via Dockerfile present in a directory that was cloned from GitHub. DockerFile is stored securely in GitHub with application code.
- Helm Configuration Check stage: This stage uses kube-score to validate the Kubernetes Helm charts configuration, generates a report and upload it to S3 Bucket. The first iteration of report was created manually using ‘kube-score’ command directly on Jenkins server. And from second iteration onward, evaluation was automated in jenkins pipeline generating a updated-kube-score-results.txt file and uploading it to S3 bucket (x23186925-kubescore-results)

```
stage('Helm configuration check'){
  steps(
    sh '''
      cd /var/lib/jenkins/workspace/testt/msc-thesis-2024
      helm get manifest my-releas -n default > updated-kubescore-manifest22222.yaml
      kube-score score updated-kubescore-manifest.yaml --output-format ci > updated-kube-score-results.txt
      aws s3 cp updated-kube-score-results.txt s3://x23186925-kubescore-results
    '''
  )
}
```

Figure 17: Configuration check of Helm charts

- Deployment using Helm: The final stage of pipeline deploys the Helm chart to EKS cluster and ensures application is running with the latest Helm package stored in S3-based Helm repository.

```
stage('Deployment using helm'){
  steps(
    sh '''
      helm repo add my-s3-repo https://x23186925-thesis-helm-charts.s3.amazonaws.com
      helm repo update
      helm upgrade my-releas my-s3-repo/nginx-app --version 0.2.0
    '''
  )
}
```

Figure 18: Deployment stage

## 11 Conclusion

- Application code, DockerFile, sonar-project.properties file were stored in GitHub
- Configuration related to static analysis was mentioned in sonar-project.properties file.
- Result generated by KubeScore were stored in S3 bucket ((x23186925-kubescore-results))
- Helm charts are stored securely in S3 bucket (x23186925-thesis-helm-charts), S3 bucket which is configured as a Helm repository.
- The Jenkins pipeline was structured in stages to perform AWS action, static analysis, create and push Docker image, configuration check of Helm and finally deployment of application stage.