National College of Ireland

Configuration Manual

MSc Research Project
Cloud Computing

Sanal Sabu
Student ID: 23382651

School of Computing
National College of Ireland

Supervisor: Sean Heeney

National College of Ireland
Project Submission Sheet
School of Computing

National College of Ireland

Student Name:	Sanal Sabu
Student ID:	23382651
Programme:	Cloud Computing
Year:	2025
Module:	MSc Research Project
Supervisor:	Sean Heeney
Submission Due Date:	11/12/25
Project Title:	Configuration Manual
Word Count:	2765
Page Count:	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.

Signature:	Sanal Sabu
Date:	26th January 2026

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

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

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

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Sanal Sabu
23382651

26th January 2026

Contents

1 Introduction

1.1 Project Overview

This is an end-to-end ecosystem for developing, deploying, and managing secure cloud-native applications. The application is built following modern DevOps principles that include IaC (Infrastructure-as-Code), CI/CD (Continuous Integration and Continuous Deployment), PaC (Policy-as-Code), and observability.

The system consists of a containerized microservices application deployed on a Kubernetes cluster in AWS. The entire lifecycle—developed, deployed, and administered—has a rather pronounced emphasis on the shifting left of security by bringing automated security scanning and policy enforcement into the pipeline.

1.2 Purpose of This Manual

This manual comprehensively describes the setting up of the development environment, provisioning the cloud infrastructure necessary to deploy the application, and validating the functionality of the system and security controls. It is meant for developers, DevOps engineers, and system administrators in charge of operating the application.

1.3 System Architecture

The upper-layer architecture comprises some important components:

- AWS Cloud Infrastructure: VPC, EKS Cluster, RDS Database, and ECR Registries managed with Terraform.
- Microservices Application: A Python-based API service and a background Worker service containerized with Docker.
- CI/CD Pipeline: Automated workflows on GitHub Actions for building, testing, scanning, and deploying the application.
- Policy-as-Code framework: OPA Gatekeeper for Kubernetes and OPA CLI for Terraform to enforce security and operational policies.
- Observability Stack: Prometheus and Grafana for metric collection and real-time visualization of application performance and security compliance.

2 Prerequisites and Initial Setup

Before proceeding, ensure all required tools are installed and configured on your local machine.

2.1 Required Software

The following tools are required. Please ensure they are installed and available in your system's PATH.

- **Git:** For version control.

- **AWS CLI v2:** For interacting with AWS services.
- **Terraform (v1.5.0+):** For infrastructure provisioning.
- **kubectl:** For interacting with the Kubernetes cluster.
- **Helm (v3.12.0+):** For managing Kubernetes applications.
- **Docker Desktop:** For building and running containers.
- **OPA (Open Policy Agent):** For local policy validation.

2.2 AWS Account Configuration

First, configure the AWS CLI with credentials that have sufficient permissions (e.g., AdministratorAccess for this project) to create the required resources.

Listing 1: Configure AWS CLI

```
aws configure
```

You will be prompted to enter your AWS Access Key ID, Secret Access Key, default region (`us-east-1`), and default output format (`json`).

2.3 Repository Setup

Clone the project repository from GitHub to your local machine.

Listing 2: Clone Project Repository

```
git clone https://github.com/sanals77/Cloud-native-policy-as-
    code.git
cd Sanal_thesis
```

2.4 AWS Prerequisites Setup

The project uses an S3 bucket for Terraform's remote state and a DynamoDB table for state locking. Run the provided PowerShell script to set these up.

Listing 3: Run AWS Setup Script

```
./scripts/setup-aws.ps1
```

This script will create and configure the S3 bucket and DynamoDB table with the necessary security settings (encryption, versioning, and public access blocks).

3 Infrastructure Deployment with Terraform

This section details how to provision the core cloud infrastructure using Terraform.

3.1 Navigate and Initialize Terraform

Navigate to the Terraform directory and initialize the backend and providers.

Listing 4: Initialize Terraform

```
cd infrastructure/terraform
terraform init
```

3.2 Configure Environment Variables

Terraform uses a `terraform.tfvars` file to define environment-specific variables. Copy the example file and customize it if necessary. For the ‘dev‘ environment, the defaults are generally sufficient.

Listing 5: Copy Terraform Variables File

```
cp terraform.tfvars.example terraform.tfvars
```

3.3 Plan and Apply the Infrastructure

Generate a Terraform execution plan to review the resources that will be created. Then, apply the plan to provision the infrastructure.

Listing 6: Plan and Apply Terraform

```
# Create a plan for the 'dev' environment
terraform plan -var="environment=dev" -out=tfplan

# Apply the plan to create the resources
terraform apply "tfplan"
```

This process will take several minutes as it creates the VPC, EKS cluster, RDS instance, and other resources. Upon completion, Terraform will display a list of outputs, including the EKS cluster endpoint and RDS connection details.

4 Kubernetes Cluster Configuration

After the infrastructure is provisioned, configure your local environment to interact with the new EKS cluster.

4.1 Configure kubectl

Use the AWS CLI to update your local kubeconfig file with the connection details for the newly created EKS cluster. Terraform provides the exact command as an output.

Listing 7: Update kubeconfig for EKS

```
aws eks update-kubeconfig --region us-east-1 --name cloud-
   native-app-dev
\end{listing}

Verify the connection by listing the nodes in the cluster.
\begin{lstlisting}[language=bash, caption={Verify Cluster
   Connection}, label={lst:get-nodes}]
kubectl get nodes
```

4.2 Create Kubernetes Namespace

Create a dedicated namespace for the ‘dev‘ environment.

Listing 8: Create Namespace

```
kubectl create namespace dev
```

4.3 Create Database Credentials Secret

The application requires database credentials to be stored as a Kubernetes Secret. Retrieve the credentials from AWS Secrets Manager (created by Terraform) and create the secret in the ‘dev‘ namespace.

Listing 9: Create Database Secret

```
# 1. Get the Secret ARN from Terraform output
$DB_SECRET_ARN = "arn:aws:secretsmanager:..." # Paste ARN here

# 2. Retrieve the secret value from AWS
$DB_CREDS = aws secretsmanager get-secret-value --secret-id
   $DB_SECRET_ARN --query SecretString --output text |
   ConvertFrom-Json

# 3. Create the Kubernetes secret
kubectl create secret generic db-credentials `
  --from-literal=host=$($DB_CREDS.host) `
  --from-literal=username=$($DB_CREDS.username) `
  --from-literal=password=$($DB_CREDS.password) `
  --from-literal=dbname=$($DB_CREDS.dbname) `
  --namespace dev
```

5 Application Deployment

This section covers both the automated CI/CD deployment and the manual deployment process for local development and testing.

5.1 Automated Deployment via CI/CD

The primary method of deployment is through the GitHub Actions CI/CD pipeline.

5.1.1 GitHub Secrets Configuration

To enable the pipeline to interact with AWS, the following secrets must be configured in GitHub repository's settings :

- `AWS_ACCESS_KEY_ID`: Your AWS access key.
- `AWS_SECRET_ACCESS_KEY`: Your AWS secret key.

Note: For production, it is highly recommended to use OIDC to grant GitHub Actions temporary credentials without storing long-lived keys.

5.1.2 Deployment Triggers

The CD pipeline (`.github/workflows/cd.yml`) is configured to trigger deployments based on the following events:

- **Push to `main` branch:** Deploys to the `dev` environment.
- **Push of a tag (e.g., `v1.0.0`):** Deploys to `staging` and then to `prod`.
- **Manual Trigger (`workflow_dispatch`):** Allows manual deployment to a chosen environment.

5.2 Manual Deployment for Local Testing

To deploy the application manually from your local machine, follow these steps.

5.2.1 Authenticate with ECR

Log your Docker client into the Amazon ECR registry.

Listing 10: ECR Login

```
aws ecr get-login-password --region us-east-1 | docker login --
    username AWS --password-stdin 537651148488.dkr.ecr.us-east-1.
    amazonaws.com
```

5.2.2 Build and Push Docker Images

Build and push the images for both the API and Worker services. Replace ‘v1.0.0‘ with your desired image tag.

Listing 11: Build and Push API Service Image

```
# Navigate to the api-service directory
cd microservices/api-service

# Build and push the image
docker build -t 537651148488.dkr.ecr.us-east-1.amazonaws.com/
   cloud-native-app-dev-api-service:v1.0.0 .
docker push 537651148488.dkr.ecr.us-east-1.amazonaws.com/cloud-
   native-app-dev-api-service:v1.0.0
```

Listing 12: Build and Push Worker Service Image

```
# Navigate to the worker-service directory
cd microservices/worker-service

# Build and push the image
docker build -t 537651148488.dkr.ecr.us-east-1.amazonaws.com/
   cloud-native-app-dev-worker-service:v1.0.0 .
docker push 537651148488.dkr.ecr.us-east-1.amazonaws.com/cloud-
   native-app-dev-worker-service:v1.0.0
```

5.2.3 Deploy with Helm

Use Helm to deploy or upgrade the applications in the ‘dev‘ namespace, specifying the image tag you just pushed.

Listing 13: Deploy API Service with Helm

```
# Navigate to the project root directory
cd ../..

helm upgrade --install api-service ./infrastructure/kubernetes/
   helm/api-service \
  --namespace dev \
  --values ./infrastructure/kubernetes/helm/api-service/values-
     dev.yaml \
  --set image.tag=v1.0.0
```

Listing 14: Deploy Worker Service with Helm

```
helm upgrade --install worker-service ./infrastructure/
```

```
    kubernetes/helm/worker-service \
   --namespace dev \
   --set image.tag=v1.0.0
```

6 Observability Stack Setup

This section explains how to set up Prometheus and Grafana for monitoring.

6.1 Install kube-prometheus-stack

Create a ‘monitoring‘ namespace and install the stack using Helm.

Listing 15: Install Prometheus and Grafana

```
# Create namespace
kubectl create namespace monitoring

# Add Helm repositories
helm repo add prometheus-community https://prometheus-community
   .github.io/helm-charts
helm repo update

# Install the stack (sets the default Grafana password to '
   admin123')
helm install kube-prometheus-stack prometheus-community/kube-
   prometheus-stack \
  --namespace monitoring \
  --set grafana.adminPassword=admin123
```

6.2 Configure Application Monitoring

Apply the ‘ServiceMonitor‘ resource to configure Prometheus to scrape metrics from the ‘api-service‘.

Listing 16: Apply ServiceMonitor

```
kubectl apply -f observability/servicemonitor-api.yaml
```

6.3 Accessing Dashboards

Use ‘kubectl port-forward‘ to access the Grafana and Prometheus UIs locally.

Listing 17: Port-forward Grafana

```
kubectl port-forward -n monitoring svc/kube-prometheus-stack-
   grafana 3000:80
```

Access Grafana at `http://localhost:3000`. The custom dashboards "Cloud-Native Application Dashboard" and "Policy-as-Code Compliance Dashboard" will be automatically provisioned.

Listing 18: Port-forward Prometheus

```
kubectl port-forward -n monitoring svc/kube-prometheus-stack-
    prometheus 9090:9090
```

Access Prometheus at `http://localhost:9090`.

7 Policy-as-Code Setup (OPA Gatekeeper)

This section describes how to install and configure OPA Gatekeeper for runtime policy enforcement.

7.1 Install Gatekeeper

Install Gatekeeper into the cluster using its Helm chart.

Listing 19: Install Gatekeeper

```
# Create namespace
kubectl create namespace gatekeeper-system

# Add Helm repository
helm repo add gatekeeper https://open-policy-agent.github.io/
    gatekeeper/charts
helm repo update

# Install Gatekeeper
helm install gatekeeper gatekeeper/gatekeeper -n gatekeeper-
    system
```

7.2 Apply Policies

Apply the 'ConstraintTemplate' (which defines the policy logic) and the 'Constraint' (which applies the policy to specific resources).

Listing 20: Apply Gatekeeper Policies

```
# Apply the template that defines the 'required labels' policy
kubectl apply -f policies/kubernetes/gatekeeper-constraint-
    template.yaml

# Apply the constraint to enforce that deployments in 'dev'
    have 'app' and 'version' labels
kubectl apply -f policies/kubernetes/gatekeeper-constraint.yaml
```

8 Validation and Testing

This section provides commands to validate that the application and its surrounding ecosystem are functioning correctly.

8.1 Test API Connectivity

Use ‘kubectl run‘ to create a temporary pod inside the cluster and make a request to the ‘api-service‘.

Listing 21: Test GET /api/items Endpoint

```
kubectl run test-api --image=curlimages/curl:latest --rm -it --
    restart=Never -n dev -- curl http://api-service/api/items
```

8.2 Check Worker Service Logs

View the logs from the worker service to ensure it is running its periodic tasks.

Listing 22: View Worker Logs

```
kubectl logs deployment/worker-service -n dev --tail=10
```

8.3 Run Policy Experiments

The project includes a suite of experiments to validate the Policy-as-Code framework. These can be run via the provided script.

Listing 23: Run Policy Validation Experiments

```
# Navigate to the experiments directory
cd experiments

# Run all experiments
./run-all-experiments.ps1 -Experiment all
```

This script will execute tests for insecure container configurations, missing resource limits, vulnerable dependencies, and insecure Terraform code, providing detailed output for each.

A Troubleshooting

- **AWS Auth mismatch:** Ensure that your AWS CLI has been successfully configured using ‘aws configure‘. The crednals should have suitable IAM permissions.

- **kubectl Connection Refused:** Make sure you have set your kube context for that region and cluster correctly with ‘aws eks update-kubeconfig‘. Firewall might block the connection.
- **ImagePullBackOff Error in Kubernetes:** The mention of this error can be of errors during pulling the ECR Image. Can you check whether you have authenticated to ECR by ‘aws ecr get-login-password...‘, and are using the right image tags in your Helm command or Kubernetes manifest? You may need those images in your registry.
- **Helm Fails with Timeout:** An under-resourced cluster where pods are not ready for long in case of some misconfiguration happens (like- wrong secret or persistent volume) usually gives this error message. Go for ‘kubectl describe pod ¡pod-name¿ -n ¡namespace¿‘ for investigations.

B Key File and Directory Locations

- **Infrastructure Code:** `infrastructure/terraform/`
- **CI/CD Workflows:** `.github/workflows/`
- **Kubernetes Deployments (Helm):** `infrastructure/kubernetes/helm/`
- **Policy-as-Code Rules (Rego):** `policies/`
- **Application Source Code:** `microservices/`
- **Observability Configurations:** `observability/`
- **Validation Experiments:** `experiments/`
- **Deployment Scripts:** `scripts/`