INFRACONTROL: A CLOUD-NATIVE AWS INFRASTRUCTURE VISIBILITY AND SECURITY ANALYSIS PLATFORM

Name: **Chepyala Sai Koushik**
LinkedIn: https://www.linkedin.com/in/sai-koushik-chepyala
GitHub: https://github.com/saikoushikchepyala1
Portfolio: https://sai-koushik1.vercel.app/
Project Link: https://github.com/saikoushikchepyala1/InfraControl

ABSTRACT

Cloud infrastructure has become the backbone of modern software systems, enabling scalability, reliability, and cost efficiency. However, managing and monitoring cloud resources across regions remains a challenge, especially for students, small teams, and organizations without centralized visibility tools. INFRACONTROL is a cloud-based infrastructure monitoring platform designed to provide clear visibility into AWS resources such as EC2, S3, Lambda, DynamoDB, and IAM across regions. The system offers a unified dashboard that displays configuration risks, resource inventory, regional distribution, and service usage in real time using AWS APIs.

Keywords: AWS, Terraform, AWS EKS, Kubernetes, Docker, FastAPI, CI/CD, IRSA, DockerHub, AWS APIs

List of Figures

2.1 Workflow of the InfraControl 5

4.1 Terraform Apply Execution 12
4.2 Terraform Output Details 13
4.3 IAM IRSA Role Created 13
4.4 EKS Cluster Provisioned 14
4.5 Kubernetes Pod Status in EKS 14
4.6 Application Running (Local/Docker) 14
4.7 Application Scan Results - Part 1 15
4.8 Application Scan Results - Part 2 15
4.9 DockerHub Repository Push Confirmation 16
4.10 GitHub Actions CI/CD Pipeline Success 16

5.1 Repository Structure 17

Table of Contents

List of Figures . . . . .

Title **Page No.**

CHAPTER 1 Introduction 1

1.1 Introduction 1

1.2 Problem Statement 1

1.3 Objectives of the Project Work 2

1.4 AWS Services and Technologies Used 3

CHAPTER 2 Architecture 4

2.1 System Architecture 4

2.2 Operational Workflow 6

2.3 Multiple Execution Environments 6

CHAPTER 3 Implementation 7

3.1 Overview 7

3.2 Prerequisites 7

3.3 Method A — Local Run 8

3.4 Method B — Docker Run 8

3.5 Method C — AWS EKS Deployment 9

3.5.1 Initial Infrastructure Setup 9

3.5.2 Automated Deployment Using CI/CD Pipeline 10

3.5.3 Security and Account Isolation 11

CHAPTER 4 Results 12

4.1 Infrastructure Provisioning 12

4.2 Application Deployment and Execution 14

4.3 AWS Scan Visualization and CI/CD 15

4.4 Summary of Results 16

CHAPTER 5 Conclusion 17

5.1 Repository Structure 17

CHAPTER 1

Introduction

1.1 Introduction

Cloud platforms such as Amazon Web Services (AWS) provide a wide range of services that enable organizations to deploy applications quickly and scale them efficiently. As cloud adoption grows, the number of deployed resources also increases, making it difficult to maintain visibility and control over infrastructure.

Most beginners and small teams lack a simple tool to understand what resources are running, in which region they exist, and how services are distributed across the account. InfraControl addresses this problem by offering a centralized and easy-to-understand dashboard that visualizes AWS infrastructure resources in real time.

InfraControl, a cloud-native application designed to scan and analyze AWS infrastructure resources and present the results through a web-based dashboard. This project automates provisioning, deployment and secure operation of a microservice application on AWS. It demonstrates Infrastructure as Code (Terraform), containerization (Docker), Kubernetes orchestration (EKS), secure IAM and IRSA usage, and CI/CD automation with GitHub Actions.

1.2 Problem Statement

As organizations increasingly adopt Amazon Web Services (AWS) for deploying applications and managing infrastructure, the number of active cloud resources across regions continues to grow. In many cases, these resources are created dynamically through Infrastructure as Code, experimentation, or automated pipelines. Over time, this leads to fragmented visibility, unmanaged services, and potential security and cost risks.

AWS provides service-specific dashboards; however, it does not offer a simplified view that combines inventory visibility with basic security and configuration checks across multiple regions in a single interface as a result:

- Users may be unaware of running compute resources that continuously incur cost.
- Critical security configurations such as encryption, MFA, or flow logs may remain disabled.
- Default infrastructure components (e.g., default VPCs) may be used without proper hardening.
- Monitoring resources across multiple regions becomes operationally complex.

1.3 Objectives of the Project Work

The primary objective of InfraControl is to design and implement a cloud-native AWS Infrastructure Visibility Dashboard that provides real-time inventory and basic security analysis of AWS resources.

The specific objectives of this project are:

- To develop a system that scans AWS accounts across multiple regions and aggregates resource information.
- To display service-wise inventory details including EC2, DynamoDB, Lambda, RDS, VPC, EBS, EIP, ELB, S3, and IAM.
- To detect common configuration and security issues such as:
  - Running EC2 instances incurring cost
  - Unencrypted S3 buckets and DynamoDB tables
  - VPC Flow Logs not enabled
  - Use of default VPC configurations
  - IAM users without Multi-Factor Authentication (MFA)
- To categorize detected issues based on severity levels (Low, Medium, High) and provide recommended remediation steps.
- To deploy the application using containerization (Docker) and Kubernetes orchestration on Amazon EKS.
- To implement Infrastructure as Code (Terraform) and CI/CD automation (GitHub Actions) for reproducible and version-controlled deployments.

1.4 AWS Services and Technologies Used

InfraControl is implemented using a combination of cloud services, container technologies, and automation tools. The major services and technologies used in this project are:

- **Amazon EC2** – Used internally by Amazon EKS worker nodes to run application containers.
- **Amazon S3, IAM, RDS, DynamoDB and VPC** – Target AWS services scanned by the InfraControl application for inventory and configuration analysis.
- **Amazon EKS (Elastic Kubernetes Service)** – Provides the managed Kubernetes environment where the application is deployed in the cloud.
- **IAM and IRSA** – Used to grant secure and limited AWS permissions to the Kubernetes application without storing long-term credentials.
- **Terraform** – Implements Infrastructure as Code for automated and repeatable provisioning of AWS infrastructure.
- **Docker** – Packages the backend and frontend into a portable container image for consistent execution.
- **Kubernetes** – Manages container deployment, scaling, and service exposure inside the EKS cluster.
- **GitHub Actions** – Automates build, container push, and deployment through a CI/CD pipeline.
- **FastAPI and Python** – Used to develop the backend API responsible for AWS scanning and result processing.

CHAPTER 2

Architecture

2.1 System Architecture

InfraControl follows a cloud-native, container-based architecture designed to provide secure and scalable visibility into AWS infrastructure resources.

The system is structured to support local execution, containerized execution, and production deployment on Amazon Elastic Kubernetes Service (EKS) while maintaining a consistent application workflow.

The architecture consists of the following major components:

- **User Browser Interface** – allows the user to initiate AWS account scanning and view detected resources and security findings.
- **FastAPI Backend Application** – performs AWS API calls, collects infrastructure data, analyzes configurations, and generates results.
- **Docker Containerization** – packages the backend application and its dependencies to ensure consistent execution across environments.
- **Kubernetes on Amazon EKS** – provides orchestration, scalability, and managed runtime for production deployment.
- **IAM Role with IRSA (IAM Roles for Service Accounts)** – enables secure and temporary access from Kubernetes pods to AWS APIs without exposing static credentials.
- **CI/CD Pipeline using GitHub Actions** – automates image building, pushing to a container registry, and deployment to the EKS cluster.

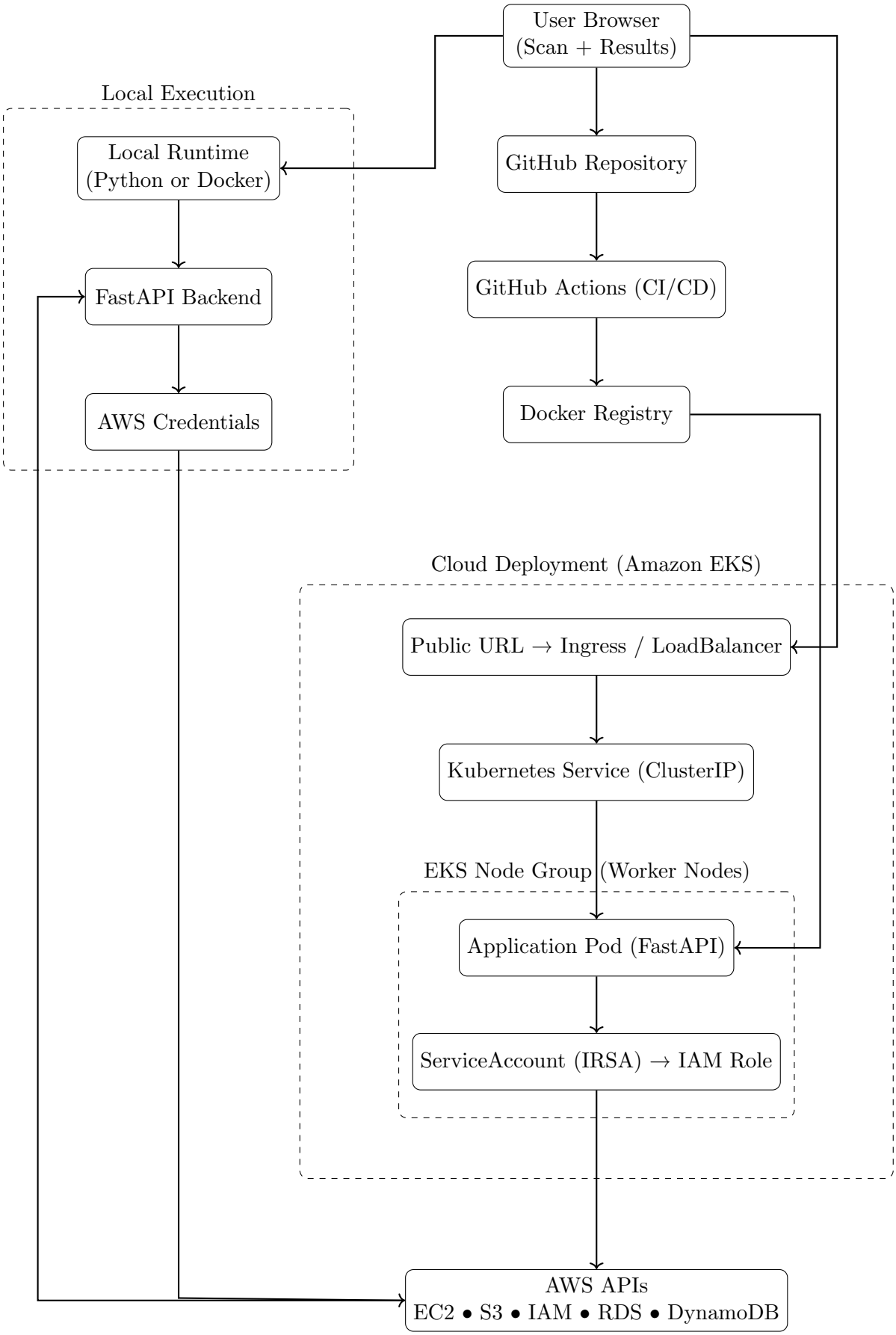

Figure 2.1: Workflow of the InfraControl

2.2 Operational Workflow

The operational flow of InfraControl begins when the user opens the web interface and initiates an account scan. The request is received by the FastAPI backend, which authenticates using either:

- Locally configured AWS credentials (during local or Docker execution), or
- IRSA-based IAM role permissions (during EKS deployment).

Using these permissions, the backend queries multiple AWS service APIs such as EC2, S3, IAM, RDS, and DynamoDB to gather resource inventory and configuration details. The collected data is then analyzed to identify security risks, configuration gaps, and active cost-incurring resources.

Finally, the processed results are returned to the web interface, where they are displayed in a structured dashboard showing:

- Detected resources in various regions,
- Issue severity levels, and
- Recommended fixes.

2.3 Multiple Execution Environments

- **Local Execution** – The backend runs directly on the user's machine using locally configured AWS credentials. Scanning is limited strictly to the user's own AWS account.
- **Docker-based Execution** – The same application runs inside a Docker container with mounted AWS credentials. This ensures portability and consistent runtime behavior across systems.
- **Cloud Deployment on Amazon EKS** – The containerized application is deployed to a managed Kubernetes cluster. Instead of static credentials, the running pod securely accesses AWS services through an IRSA-linked IAM role, enabling security and scalability.

CHAPTER 3

Implementation

3.1 Overview

This chapter gives step-by-step instructions to set up, run, and deploy InfraControl. It explains three ways to run the project:

- Local (run directly on a developer machine),
- Docker (container run on a machine), and
- EKS (cluster deployment using Terraform and Kubernetes).

Each method uses the same application code and produces the same web interface. The only difference is how credentials are provided to the application.

3.2 Prerequisites

Before starting, make sure you have:

- An AWS account (for EKS and Terraform provisioning).
- AWS CLI v2 installed and configured (if you use local or Terraform).
- Terraform v1.3+ installed.
- Docker installed and logged in (for building and pushing the image).
- kubectl installed (for EKS deployments).
- A GitHub account (for CI/CD with GitHub Actions).
- Basic Linux shell knowledge (or Windows WSL).

Verify:

1. Copy the repo: git clone https://github.com/saikoushikchepyala1/InfraControl.git
2. Change into project: `cd InfraControl`
3. Verify Docker works: `docker --version`
4. Verify AWS CLI works: `aws sts get-caller-identity`

3.3 Method A — Local Run

The application will use AWS credentials configured on the local machine.

Step 1: Configure AWS on your machine
Enter Access Key ID, Secret Access Key, region (ap-south-1).

```
aws configure
```

Step 2: Install Python virtual environment and dependencies (optional if using Docker)

```
python -m venv venv
source venv/bin/activate
pip install fastapi uvicorn boto3
```

Step 3: Start backend locally (if not using Docker)

```
uvicorn backend.app.main:app –host 0.0.0.0 –port 8000
```

Open the browser at http://localhost:8000 and click Scan account.

Notes: When running locally, the application reads credentials from `/.aws/credentials` and will only access the AWS account those credentials belong to.

3.4 Method B — Docker Run

Step 1: Build Docker Image: From the repository root, run:

```
docker build -t infracontrol
```

Step 2: Run Docker Container (Option 1: Mount AWS credentials)

```
docker run -p 8000:8000 -v  /.aws:/root/.aws infracontrol
```

Open the browser at http://localhost:8000 and click Scan account.

Note: When running with Docker, the application reads credentials from the mounted file `/.aws/credentials`.

3.5 Method C — AWS EKS Deployment

This method deploys InfraControl to an Amazon EKS Kubernetes cluster. Unlike local or Docker execution, the application runs inside cloud infrastructure and is accessible through a public URL.

3.5.1 Initial Infrastructure Setup

The following steps are required only once to create the cloud infrastructure in the user's AWS account.

Step 1: Clone the repository

```
git clone https://github.com/saikoushikchepyala1/InfraControl.git
cd InfraControl
```

Step 2: Configure AWS credentials locally

```
aws configure
```

Provide:

- AWS Access Key ID
- AWS Secret Access Key
- Region: `ap-south-1`

Step 3: Provision infrastructure using Terraform

```
cd terraform
terraform init
terraform apply
```

This step creates:

- VPC and networking resources
- IAM roles and policies
- Amazon EKS cluster
- Worker node group

Step 4: Connect `kubectl` to the EKS cluster

```
aws eks update-kubeconfig –name infracontrol-eks –region ap-south-1
kubectl get nodes
```

Note: If nodes appear in the output, the cluster is ready.

Step 5: Deploy the application to Kubernetes

```
kubectl apply -f k8s/
kubectl get pods
kubectl get svc
kubectl get ingress
```

Note: Wait until the pod status becomes `Running`.

Step 6: Access Application via Public URL

Retrieve the external URL from the Service or Ingress resource and open it in a browser. InfraControl will now run inside the AWS cloud environment.

3.5.2 Automated Deployment Using CI/CD Pipeline

After the initial setup, deployment becomes automatic. A GitHub Actions workflow builds the Docker image, pushes it to Docker Hub, updates kubeconfig, and applies Kubernetes manifests to the EKS cluster.

Step 1: Push code changes to GitHub

```
git add .
git commit -m "update"
git push origin main
```

Step 2: GitHub Actions pipeline executes automatically

- Builds the Docker image
- Pushes the image to Docker Hub
- Connects to the EKS cluster
- Deploys the updated application to Kubernetes

Step 3: Application updates in the cloud

Opening the same public URL will show the latest deployed version without any manual deployment steps.

3.5.3 Security and Account Isolation

Each user must deploy InfraControl within their own AWS account. Terraform provisioning, IAM roles, and the EKS cluster all belong to that account. Therefore:

- Other users cannot access or scan your AWS account.
- The scan results always correspond to the AWS account where the application is deployed.

When running on EKS, the application uses an IAM Role for Service Accounts (IRSA), which provides temporary and limited AWS permissions. This approach avoids storing long-term credentials inside containers and follows cloud security practices.

CHAPTER 4

Results

This chapter presents the practical outcomes obtained after implementing and deploying the InfraControl platform. The results confirm that the system infrastructure, application deployment, and AWS scanning workflow operate correctly in all configured environments.

4.1 Infrastructure Provisioning

The cloud infrastructure required for the project was successfully created using Terraform. This included the virtual network configuration, IAM roles, and the Amazon EKS cluster with worker nodes.

After execution of the Terraform configuration, the outputs confirmed that all required resources were provisioned without errors and the Kubernetes cluster became ready for application deployment.

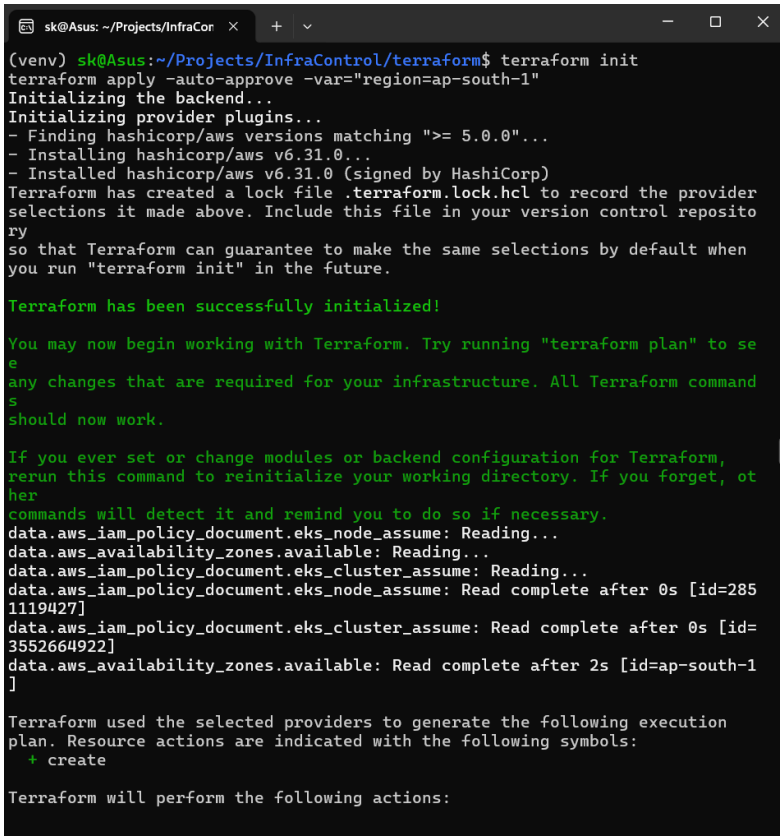

Figure 4.1: Terraform Apply Execution

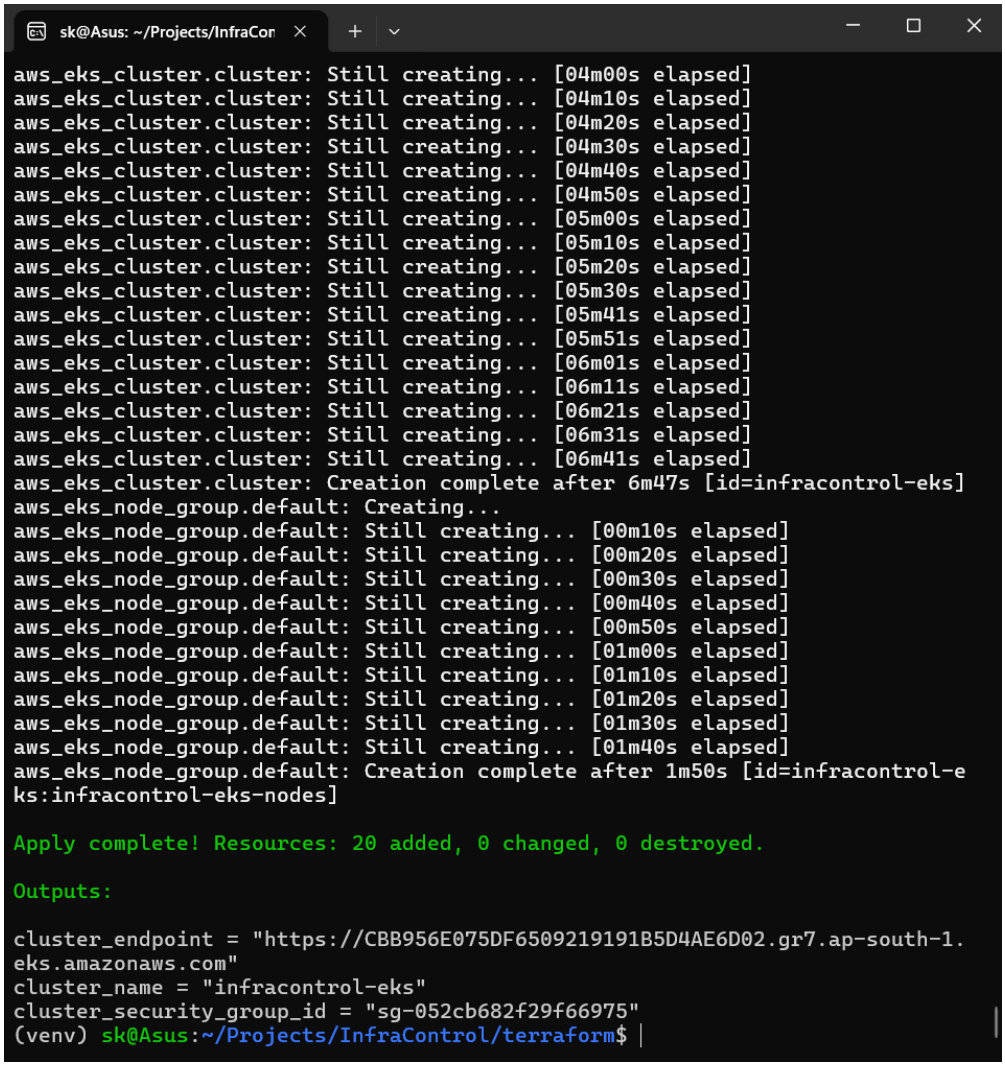

Figure 4.2: Terraform Output Details

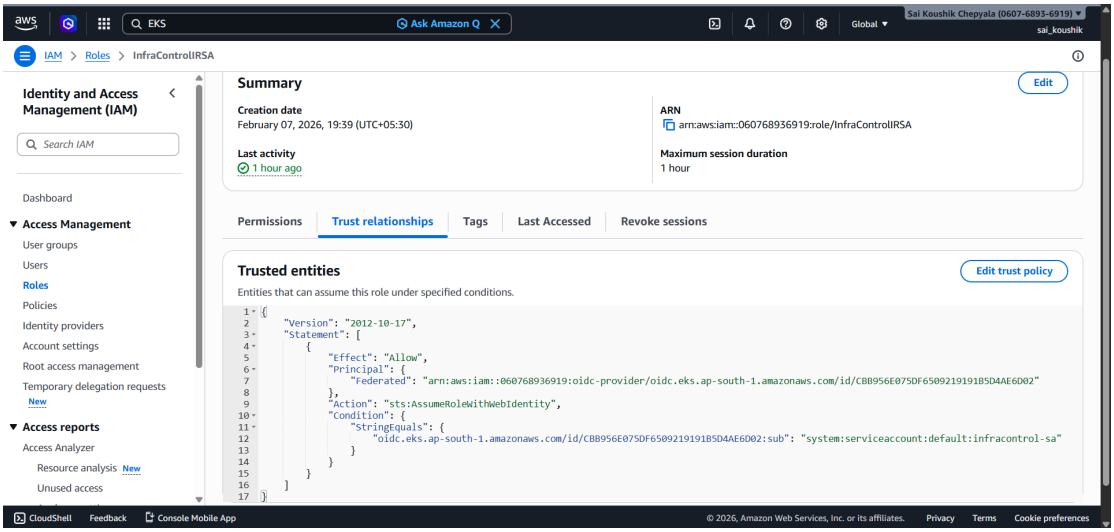

Figure 4.3: IAM IRSA Role Created

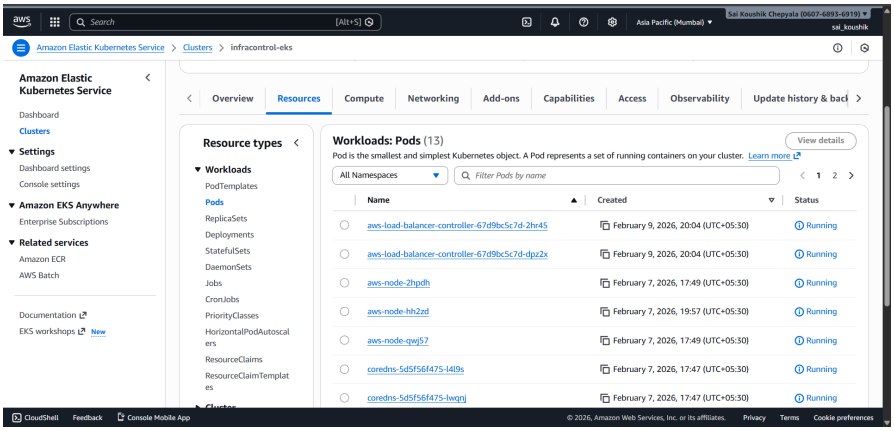

Figure 4.4: EKS Cluster Provisioned

4.2 Application Deployment and Execution

The InfraControl application was deployed successfully inside the Amazon EKS cluster using Kubernetes manifests. The backend container started properly, pods entered the running state, and the service exposure allowed the application for access.

```
sk@Asus:~/Projects/InfraControl$ kubectl get nodes
NAME                                           STATUS   ROLES    AGE    VERSION
ip-10-100-1-196.ap-south-1.compute.internal    Ready    <none>   4d3h   v1.29.15-eks-ecaa3a6
ip-10-100-2-12.ap-south-1.compute.internal     Ready    <none>   4d5h   v1.29.15-eks-ecaa3a6
ip-10-100-3-122.ap-south-1.compute.internal    Ready    <none>   4d5h   v1.29.15-eks-ecaa3a6
sk@Asus:~/Projects/InfraControl$ kubectl get pods
NAME                                    READY   STATUS    RESTARTS   AGE
infracontrol-backend-648bf6fccd-l6g78   1/1     Running   0          137m
sk@Asus:~/Projects/InfraControl$ kubectl get deployment
NAME                 READY   UP-TO-DATE   AVAILABLE   AGE
infracontrol-backend   1/1     1            1           4d
sk@Asus:~/Projects/InfraControl$ kubectl get svc
NAME              TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)   AGE
infracontrol-svc   ClusterIP   172.20.127.12   <none>        80/TCP    2d3h
kubernetes        ClusterIP   172.20.0.1      <none>        443/TCP   4d5h
sk@Asus:~/Projects/InfraControl$ kubectl get ingress
NAME                 CLASS   HOSTS   ADDRESS                                                        PORTS   AGE
infracontrol-ingress   alb     *       k8s-default-infracon-b5c109805e-606604844.ap-south-1.elb.amazonaws.com   80      2d3h
sk@Asus:~/Projects/InfraControl$
```

Figure 4.5: Kubernetes Pod Status in EKS

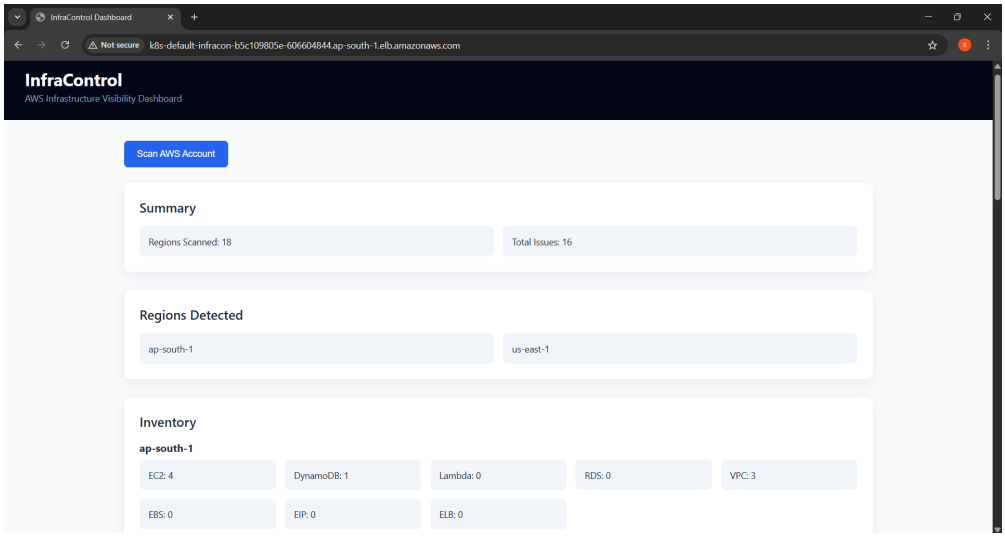

Figure 4.6: Application Running (Local/Docker)

4.3 AWS Scan Visualization and CI/CD

After deployment, the web interface successfully displayed AWS resource information, detected configuration issues, and presented the results in a dashboard view.

The Docker image was pushed to Docker Hub and the automated deployment workflow executed successfully through GitHub Actions, demonstrating a complete CI/CD pipeline.

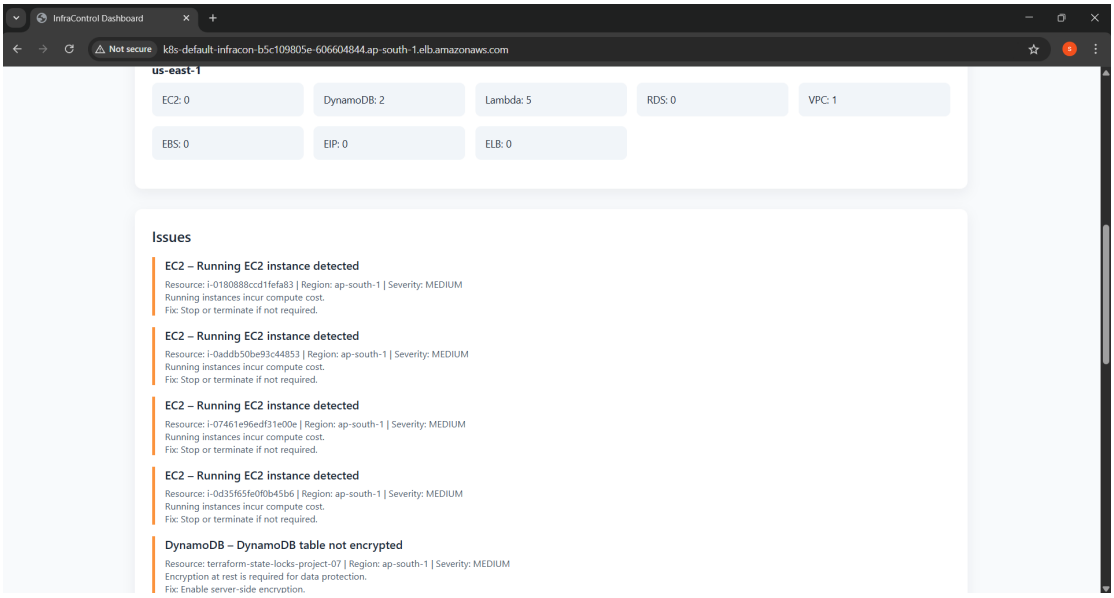

Figure 4.7: Application Scan Results - Part 1

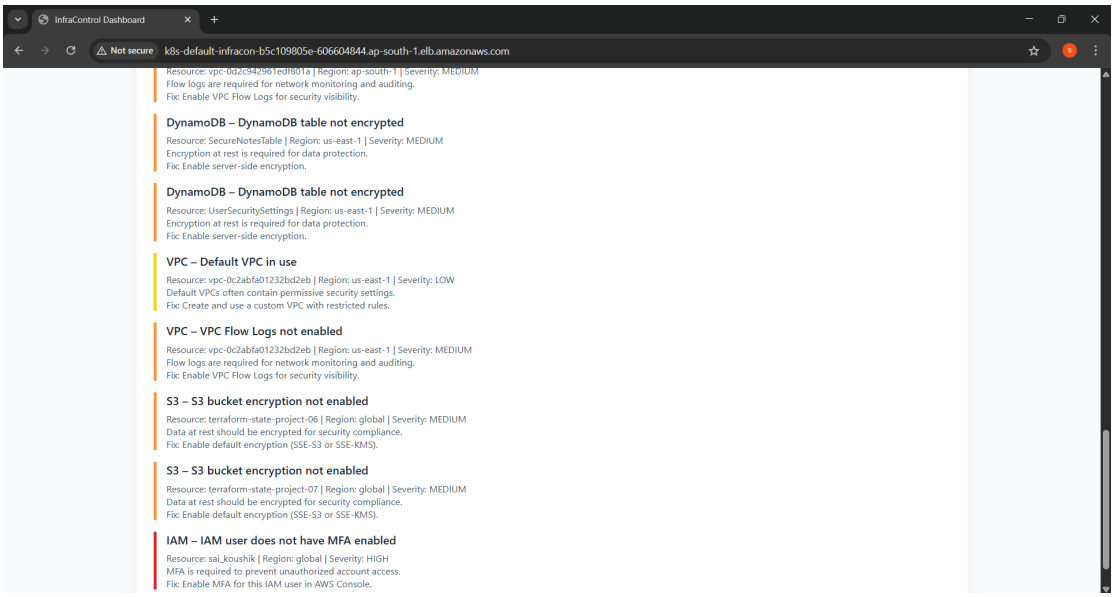

Figure 4.8: Application Scan Results - Part 2

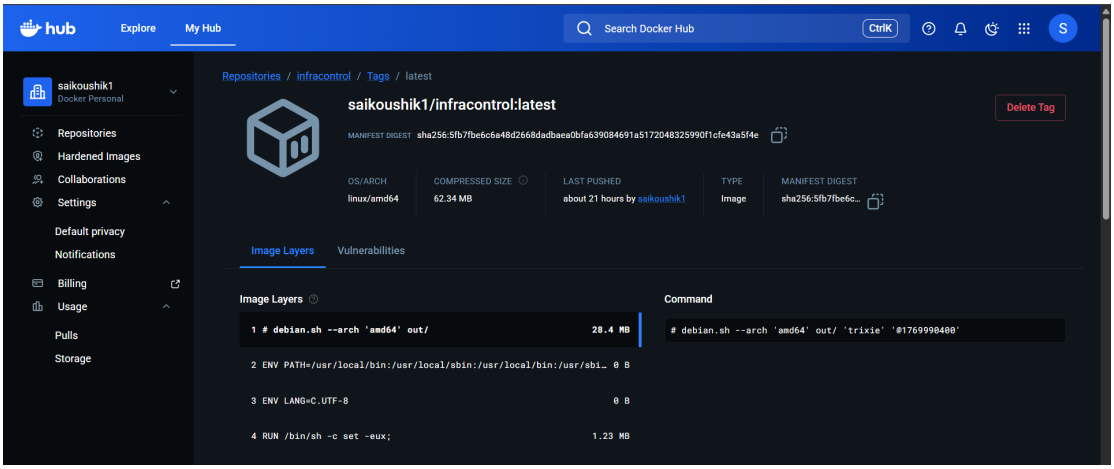

Figure 4.9: DockerHub Repository Push Confirmation

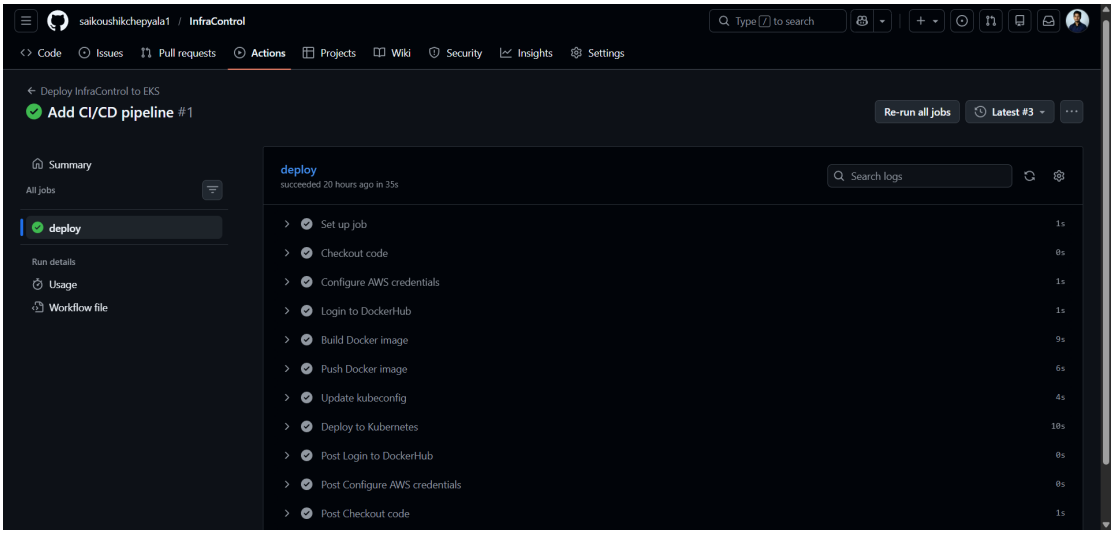

Figure 4.10: GitHub Actions CI/CD Pipeline Success

4.4 Summary of Results

The obtained results confirm that:

- Cloud infrastructure provisioning using Terraform works correctly.
- The application deploys and runs successfully on Amazon EKS.
- AWS resources and configuration issues are detected and displayed in the dashboard.
- Docker packaging and GitHub Actions CI/CD automation function as expected.

These outcomes demonstrate that InfraControl operates as a complete cloud-native AWS infrastructure visibility and analysis platform.

CHAPTER 5

Conclusion

5.1 Repository Structure

Figure 5.1: Repository Structure

- **.github/workflows**
  Contains GitHub Actions workflow files used to automate build, Docker image push, and deployment to Amazon EKS.

- **backend/app**
  Implements the FastAPI backend service responsible for scanning AWS resources, processing results, and exposing API endpoints to the frontend.

- **frontend**
  Provides the web user interface (HTML, CSS, JavaScript) that displays scan results and interacts with the backend APIs.

- **k8s**
  Includes Kubernetes manifest files for deploying the application on Amazon EKS, such as Deployment, Service, Ingress, and ServiceAccount with IRSA configuration.

- **terraform**
  Holds Infrastructure-as-Code definitions used to provision AWS networking, IAM roles, and the Amazon EKS cluster required for cloud deployment.

- **.dockerignore**
  Specifies files and folders excluded from the Docker build context to keep the container image lightweight and secure.

- **.gitignore**
  Defines files that should not be tracked in version control, such as credentials, build artifacts, and local environment files.

- **Dockerfile**
  Describes the steps required to build the InfraControl container image, including dependency installation and application startup configuration.

- **alb-policy.json**
  IAM policy document defining permissions required for AWS Load Balancer Controller or related ALB operations within the cluster.

- **scanner-policy.json**
  IAM policy granting read-only access to AWS services needed for infrastructure scanning and analysis performed by the backend.

InfraControl, a cloud-native application designed and implemented to provide clear visibility into AWS infrastructure resources and basic configuration risks through a simple web interface.

InfraControl scan and analyze AWS infrastructure resources and present the results through a web-based dashboard. This project automates provisioning, deployment and secure operation of a microservice application on AWS. It demonstrates Infrastructure as Code (Terraform), containerization (Docker), Kubernetes orchestration (EKS), secure IAM and IRSA usage, and CI/CD automation with GitHub Actions.

Project Repository: https://github.com/saikoushikchepyala1/InfraControl

Docker Image: https://hub.docker.com/r/saikoushik1/infracontrol

Author Portfolio: https://sai-koushik1.vercel.app/

Cloud-DevOps-Projects: https://github.com/saikoushikchepyala1/Cloud-DevOps-Projects