National College of Ireland

Configuration Manual

MSc Research Project
In Cloud Computing

Ravina Mestry
Student ID: x22177264

School of Computing
National College of Ireland

Supervisor: Vikas Sahni

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: …….Ravina Mestry……

Student ID: ……x22177264……

Programme: ……MSc in Cloud Computing…… **Year:** ……2023…….

Module: ……Research Project in Cloud Computing……

Lecturer: ……Vikas Sahni……

Submission Due Date: ……14/08/2023……

Project Title: Securing the Speed: Balancing Security and Deployment Velocity in DevOps

Word Count: ……2165…… **Page Count:** ……16……

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: ……Ravina Mestry……

Date: ……10/08/2023……

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

Ravina Mestry
Student ID: x22177264

1 Introduction

The system requirements, setup, software, and installation specifications used in this research will be better understood by readers of this configuration manual. Additionally, a detailed explanation of the procedures to be followed when carrying out this research project is provided in this manual. Table 1 lists the tools version and URLs discussed in detail in Section 3.

Prerequisites

- AWS and Django Python knowledge.
- AWS, Docker, and GitHub login.
- Infrastructure as code knowledge.

No.	Tools	Version	URL
1	Terraform	1.5.3	Overview | x22177264-research-project-webapp | nci-research-project | Terraform Cloud
2	Docker	3.8	ravinamestry/x22177264_ravina_mestry general | Docker Hub
3	OWASP ZAP	2.13.0	http:// <X22177264_ ravina_mestry_research_project_zap_ PUBLIC_IP>:8080/
4	Snyk	-	https://docs.snyk.io/integrations/ci-cd-integrations/github-actions-integration
5	Datadog	v7	Agent 7 | Datadog (datadoghq.eu)
6	SonarQube	3	http:// <X22177264_ ravina_mestry_research_project_sonarqube _PUBLIC_IP>:9000/
7	GitHub	2.34.1	ravinamestry/x22177264_ravina_mestry_research_project (github.com)

Table 1: AWS EC2 Instance Specification

2 Before you begin

2.1 Web Application

The web application Employee Contacts was developed in Django Python using the Django framework. Django offers the first setup for a virtual environment. In addition to CRUD features like create, update, and delete shown in Figure 1, Employee Contacts also offer non-CRUD functionality like data validation for fields in the add new employee page form. The database that is utilized by the application is SQLite. The employee contacts web app is

installed on Gunicorn wsgi application server and SQLite database is migrated using below commands.

python3 manage.py makemigrations
python3 manage.py migrate

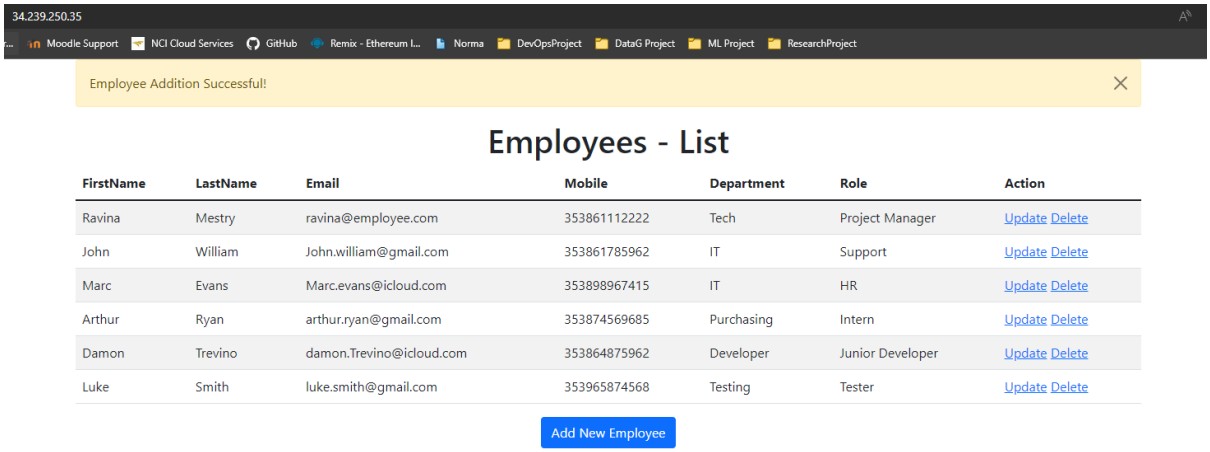

Figure 1: Web Application

- Created a virtual Python environment on local machine to install Django.
- Activated the virtual environment.

cd /mnt/c/work/workspace/django-workspace
python3 -m venv ravina_mestry_research_project_webapp _gunicorn _venv
source ravina_mestry_research_project_webapp_gunicorn_venv/bin/activate

- Installed Python 3.10.6.
- Used pip to install Django 3.2.13.
- Started the Django project and started with below command.

cd ravina_mestry_research_project_webapp
python3 manage.py runserver 8080

- After adding the URL generated by running the server in settings.py 'ALLOWED_HOSTS' the Django application is shown running successfully.
- The web application is then deployed on AWS EC2 instance in Development (Dev), Staging (Stage) and Production (Prod) using Docker image.

3 Tools/Cloud-based services

3.1 AWS EC2

- AWS EC2 instances are used for deploying dev, stage and prod environment for running the Web application, also for installing SonarQube and Zap discussed in Section 3.4 and 3.6 respectively.
- AWS Instances are created using Terraform, discussed in Section 3.2.

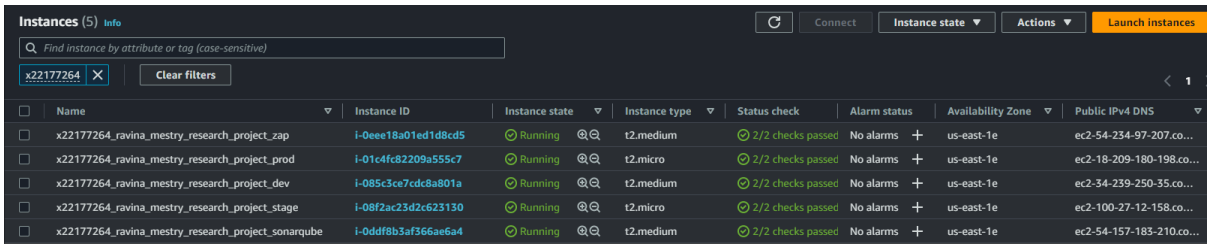

Figure 2: AWS EC2 Instances

- The Terraform-generated EC2 instances for this project are listed shown in Figure 2 and details of Instance Type and AMI ID are listed in Table 2.

AWS Instance Name	Instance Type	AMI ID
x22177264_ravina_mestry_research_project_sonarqube	t2.medium	ami-04505e74c0741db8d
x22177264_ravina_mestry_research_project_zap	t2.medium	ami-04505e74c0741db8d
x22177264_ravina_mestry_research_project_dev	t2.micro	ami-053b0d53c279acc90
x22177264_ravina_mestry_research_project_stage	t2.micro	ami-053b0d53c279acc90
x22177264_ravina_mestry_research_project_prod	t2.micro	ami-053b0d53c279acc90

Table 2: AWS EC2 Instance Specification

3.2 Terraform

- Created new Terraform Organization nci-research-project and workspaces x22177264-research-project, x22177264-research-project-sonarqube and x22177264-research-project-zap shown in Figure 3.
- Saved the AWS variables for above three workspaces in Terraform shown in Figure 4.

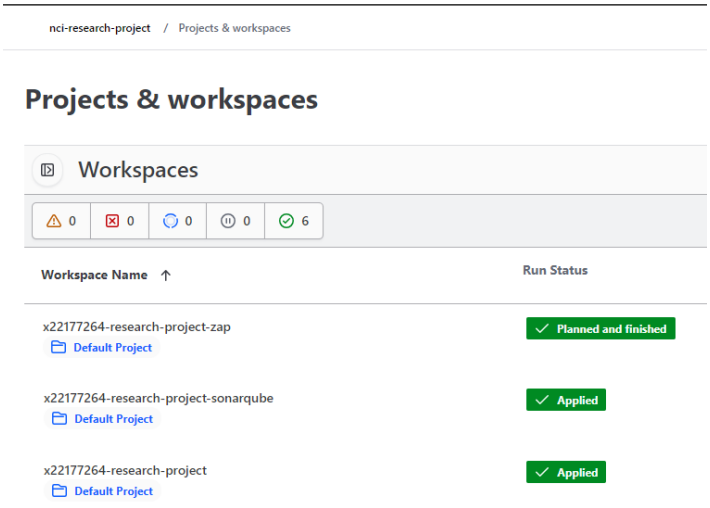

Figure 3: Terraform organization and Workspaces

- In main.tf for Terraform mentioned the EC2 instance details in Figure 5 for creating development, Staging, and production environments with the Key name, Security group ID and Tag name.
- Terraform also notes Public_IP for the EC2 instances and terraform_create.yml saving the Public_IP in GitHub Actions in Figure 6.
- The Public_ID noted in Secrets of GitHub in Figure 7 is retrieved by deploy-dev.yml and deploy.yml to deploy the Web Application to respective environments using Docker.
- Docker Images which consist of Web Application are copied to the Public_IP saved in GitHub Secrets.

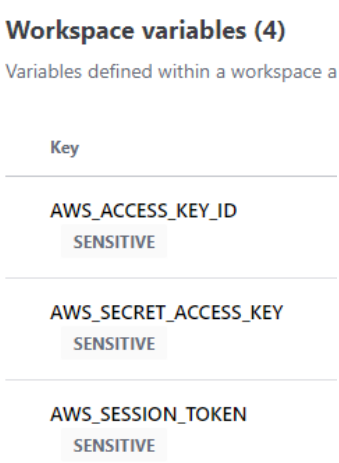

Figure 4: Terraform AWS Variables

```
#dev instance creation
resource "aws_instance" "x22177264_ravina_mestry_research_project_dev" {
  ami = "ami-04505e74c0741db8d"
  #ami = "ami-053b0d53c279acc90"
  instance_type = "t2.medium"
  subnet_id = "subnet-07f96f862eb5dbc50"
  associate_public_ip_address = true
  key_name = "x22177264_ravina_mestry_research_project_key"
  vpc_security_group_ids = ["sg-00f05bacad60c315d"]

  tags = {
    Name = "x22177264_ravina_mestry_research_project_dev"
  }
```

Figure 5: Terraform AWS Instance details

```
output "x22177264_ravina_mestry_research_project_dev_public_ip" {
  value = "${aws_instance.x22177264_ravina_mestry_research_project_dev.public_ip}"
}
```

Resources 3 | Outputs 3 | Curre

NAME ↓	TYPE	VALUE
x22177264_ravina_mestry_research_project_dev_public_ip	string	"52.91.148.130"
x22177264_ravina_mestry_research_project_prod_public_ip	string	"18.209.180.198"
x22177264_ravina_mestry_research_project_stage_public_ip	string	"100.27.12.158"

Figure 6: Terraform saving Public_IP in GitHub Secrets

Figure 7: Terraform creating Instances for Dev/Stage/Prod

3.3 Docker

- Created the repository ravina-mestry/x22177264_ravina_mestry to save the Docker Image for Web app shown in Figure 9.
- Docker Image is built and pushed to Docker Hub by Terraform in .GitHub/workflows/build.yml workflow in Figure 8.

```
- name: Build and Push webapp
  uses: docker/build-push-action@v2
  with:
    push: true
    context: ./webapp
    tags: ravinamestry/x22177264_ravina_mestry:ncirl_cloud_research_webapp_x22177264_ravina_mestry
```

Figure 8: Docker Image Tag Specification

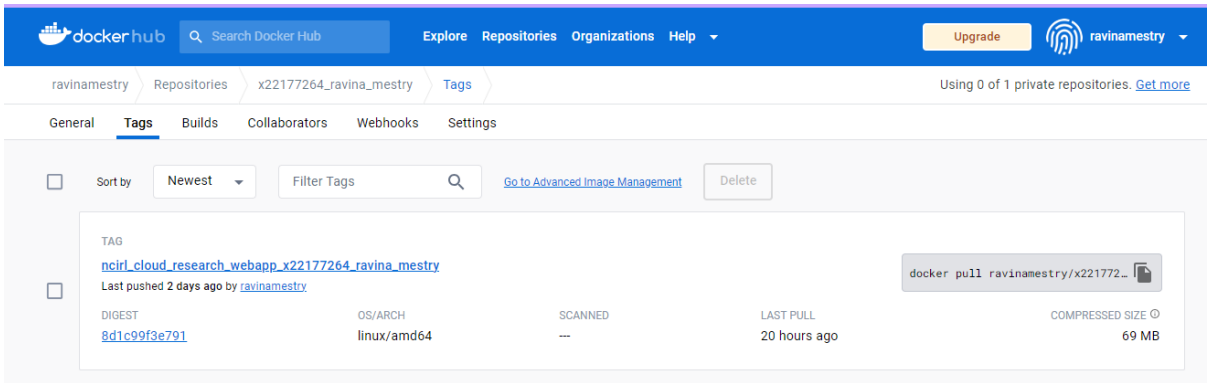

Figure 9: Docker Image pushed in Docker Hub

- Docker-compose consists of Docker Image to be pulled and ran in Docker container which runs the Web app on Dev/Stage/Prod environments in Figure 10 using below command. Docker login is stored in GitHub Action Secrets.

sudor docker-compose up -d

```
container_name: webapp
build: ./webapp
image: ravinamestry/x22177264_ravina_mestry:ncirl_cloud_research_webapp_x22177264_ravina_mestry
ports:
```

Deploying in AWS EC2

```
#9 1.897 Collecting pytz==2022.1
#9 1.905   Downloading pytz-2022.1-py2.py3-none-any.whl (503 kB)
#9 1.953 Collecting sqlparse==0.4.2
#9 1.959   Downloading sqlparse-0.4.2-py3-none-any.whl (42 kB)
#9 2.003 Collecting typing-extensions==4.1.1
#9 2.009   Downloading typing_extensions-4.1.1-py3-none-any.whl (26 kB)
#9 2.045 Requirement already satisfied: setuptools>=3.0 in /usr/local/lib/python3.6/site-packages (from gunicorn==20.1.0->-r requirements.txt (line 3)) (41.6.0)
#9 2.125 Installing collected packages: typing-extensions, sqlparse, pytz, asgiref, gunicorn, Django
#9 4.876 Successfully installed Django-3.2.13 asgiref-3.4.1 gunicorn-20.1.0 pytz-2022.1 sqlparse-0.4.2 typing-extensions-4.1.1
#9 4.878 WARNING: Running pip as the 'root' user can result in broken permissions and conflicting behaviour with the system package manager. It is recommended to u
#9 DONE 5.2s

#10 [6/6] COPY . .
#10 DONE 0.2s

#11 exporting to image
#11 exporting layers
#11 exporting layers 1.3s done
#11 writing image sha256:deb6c0d26d81588d4b1c93c29df2cf3084b8eec66395d9768bb4d15240a6d050 done
#11 naming to docker.io/***/x22177264_ravina_mestry:ncirl_cloud_research_webapp_x22177264_ravina_mestry done
#11 DONE 1.3s
Image for service webapp was built because it did not already exist. To rebuild this image you must use `docker-compose build` or `docker-compose up --build`.
Creating webapp ...
Creating webapp ... done
```

Figure 10: Docker Image ran on AWS Instance for Dev/Stage/Prod

3.4 SonarQube

- Created AWS EC2 t2.medium Instance for SonarQube using Terraform in Section 3.1.
- Added the configuration into docker-compose.yaml.
- Increased the Elasticsearch limit by adding the following command in installdocker.sh.

sudo sysctl -w vm.max_map_count=262144

- Built and started the docker containers.

sudo docker-compose up -d

- Navigated to <InstanceIP>:9000 and created the project x22177264_ravina_mestry and configured for GitHub Actions by creating SONAR_TOKEN and SONAR_HOST_URL secrets and sonar-project.properties file in GitHub repository.
- Ran the .github/workflows/build_sonarqube.yml to show the analysis of code in Figure 11.

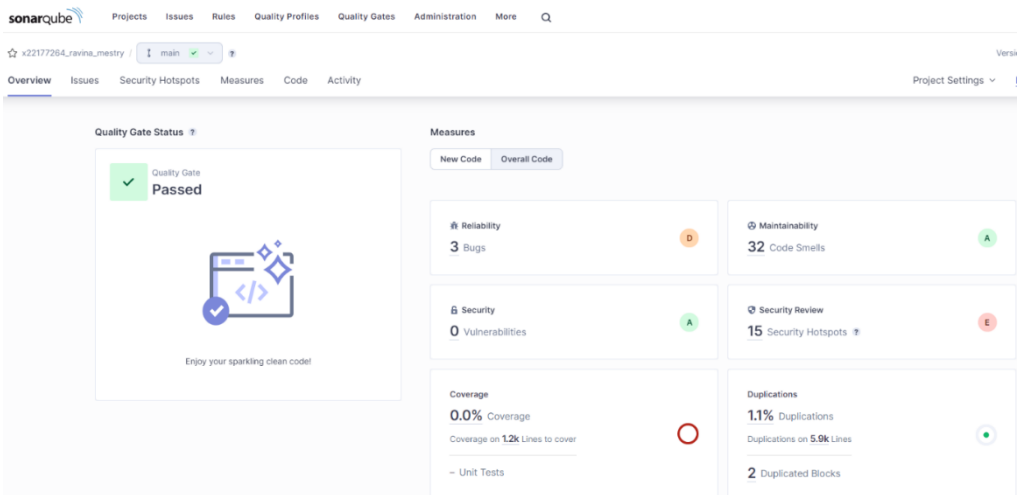

Figure 11: SonarQube Code analysis

3.5 GitHub and GitHub Action

- Created GitHub repository x22177264_ravina_mestry _research_project.
- Added the code for webapp discussed in Section 2.1.
- Added terraform/main.tf to create infrastructure in AWS EC2 in Section 3.2.
- Included SonarQube scan, Super-Linter, Snyk scan, and Datadog monitoring tools on the CI-CD pipeline.
- Created Workflows shown in Figure 12 for creating and destroying infrastructure, building Docker Image, and deploying the web app in Dev/Stage/Prod environments.

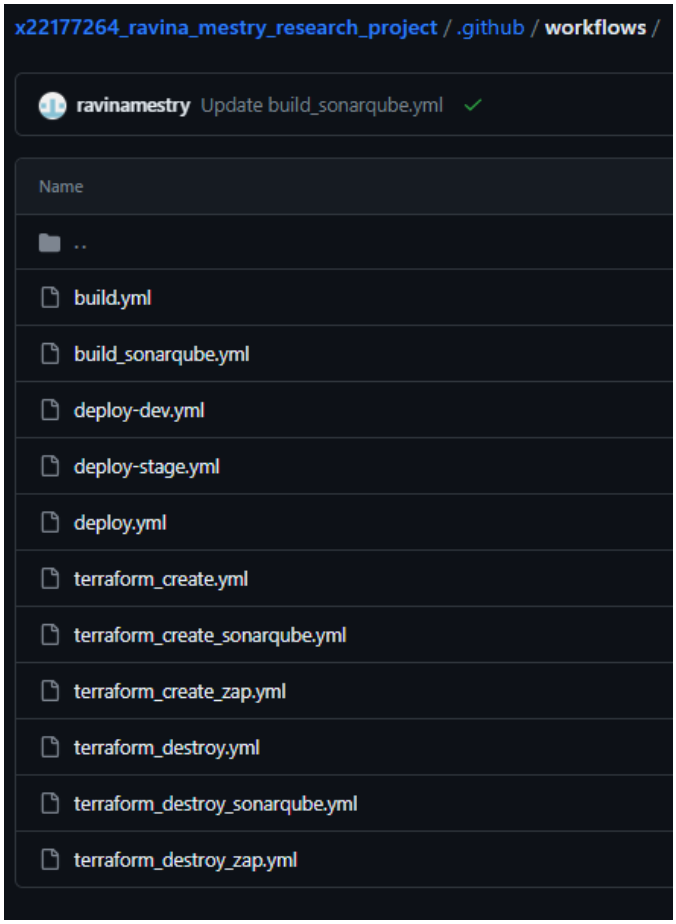

Figure 12: GitHub Actions Workflows

3.6 ZAP

- Created the AWS EC2 t2.medium Instance for Zap using Terraform discussed in Section 3.1.
- Added owasp/zap2docker-stable in Docker file with entry point zap-webswing.sh.
- Ran the docker containers with the ports 8080 or 8090 in Docker-compose.yml.

3.6.1 Steps to run a Quick Start Automated scan

- Started ZAP <x22177264_ravina_mestry_research_project_zap_PUBLIC_IP> and clicked the Quick Start tab of the Workspace.
- Clicked the large Automated Scan button.
- In the URL to attack text box, entered the full URL of the web application.
- Clicked the Attack.
- Figure 13 shows the scanner attacking passively the web application.

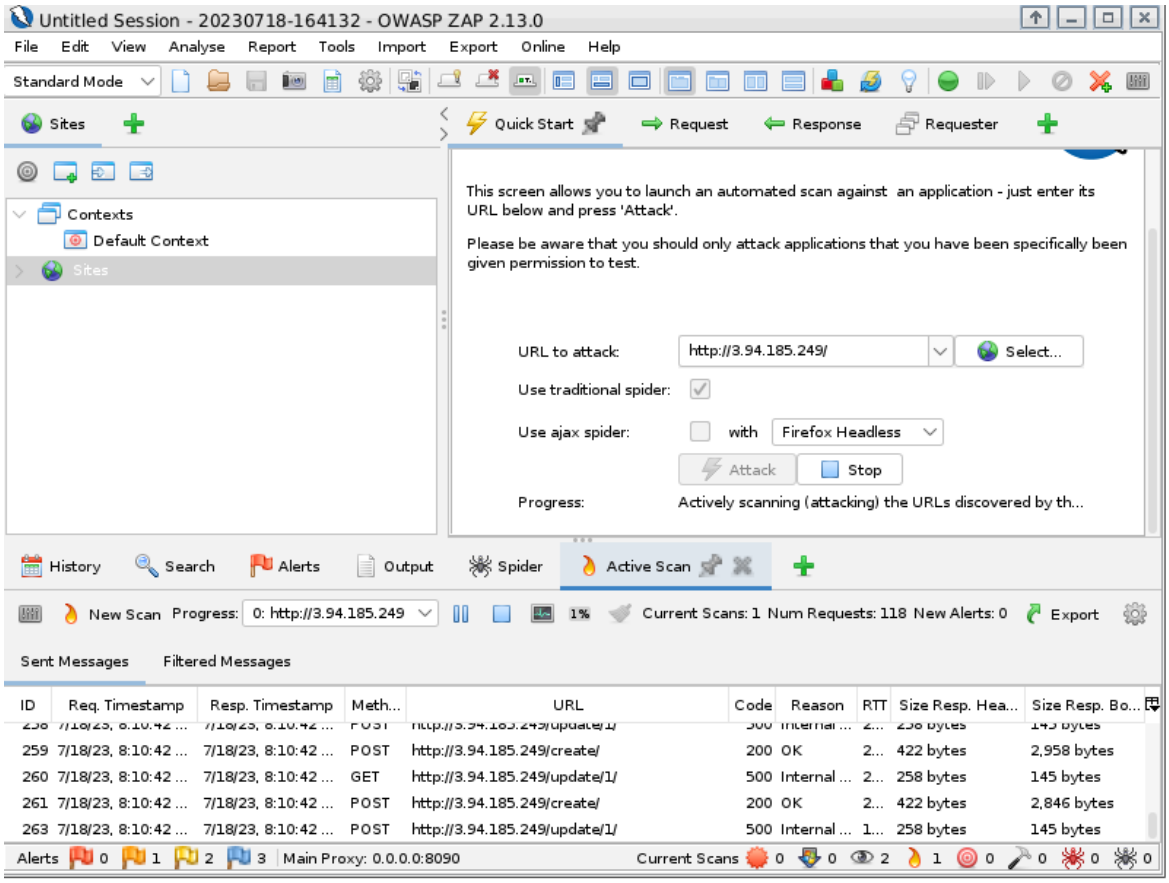

Figure 13: GitHub Actions Workflows

3.7 Snyk

- Created Snyk account with organization name ‘ravinamestry’ and generated Snyk API Token.
- Integrated Snyk with GitHub Actions and Docker Hub. Enables Repository access for GitHub Actions and enables Detect application vulnerabilities for Docker Hub in Figure 15.
- Created GitHub Action Secret for SNYK_TOKEN which is used in Figure 14 env section in .github/workflows/build.yml workflow for Scanning the web app.

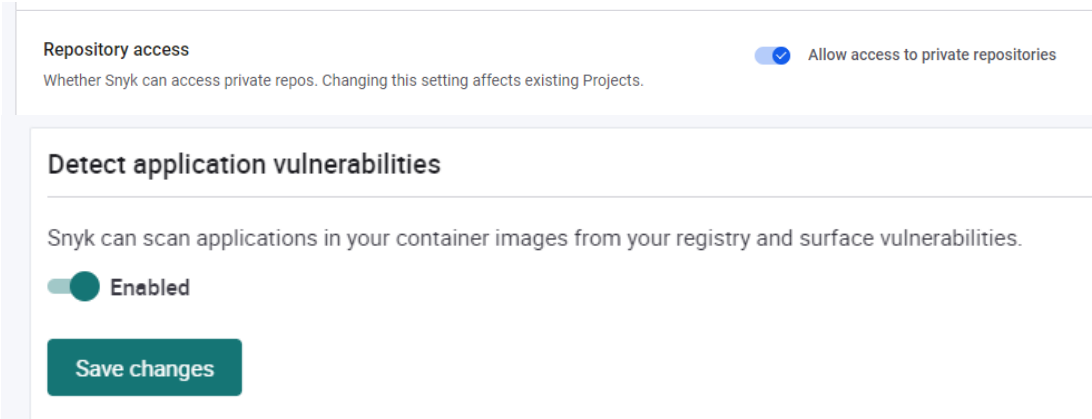

Figure 14: Snyk GitHub Actions and Docker Hub Integration

- Snyk scans the image ravinamestry/x22177264_ravina_mestry:ncirl_cloud_research_webapp_x22177264_ravina_mestry for high severity vulnerabilities mentioned in Figure 15.
- Used --sarif-file-output and the GitHub SARIF upload action, when GitHub creates code scanning alerts in a repository using information from Static Analysis Results Interchange Format (SARIF) files, SARIF files are uploaded to a repository using GitHub Actions.
- Continue-on-error is true so that when Snyk Action fails when vulnerabilities are found this would not prevent the SARIF upload action from running.

```
- name: Scan Docker image
  uses: snyk/actions/docker@master
  continue-on-error: true
  with:
    image: ravinamestry/x22177264_ravina_mestry:ncirl_cloud_research_webapp_x22177264_ravina_mestry
    args: --file=Dockerfile --severity-threshold=high --sarif-file-output=snyk.sarif
  env:
    SNYK_TOKEN: ${{ secrets.SNYK_TOKEN }}
- name: Upload Snyk report as sarif
  uses: github/codeql-action/upload-sarif@v2
  with:
    sarif_file: snyk.sarif
```

Figure 15: Snyk configuration

- Figure 16 shows the Docker image and GitHub repo scanned for vulnerabilities and Snyk tracks and flags Pull Requests in the top-most vulnerable projects with severity levels critical, high, medium, and low.

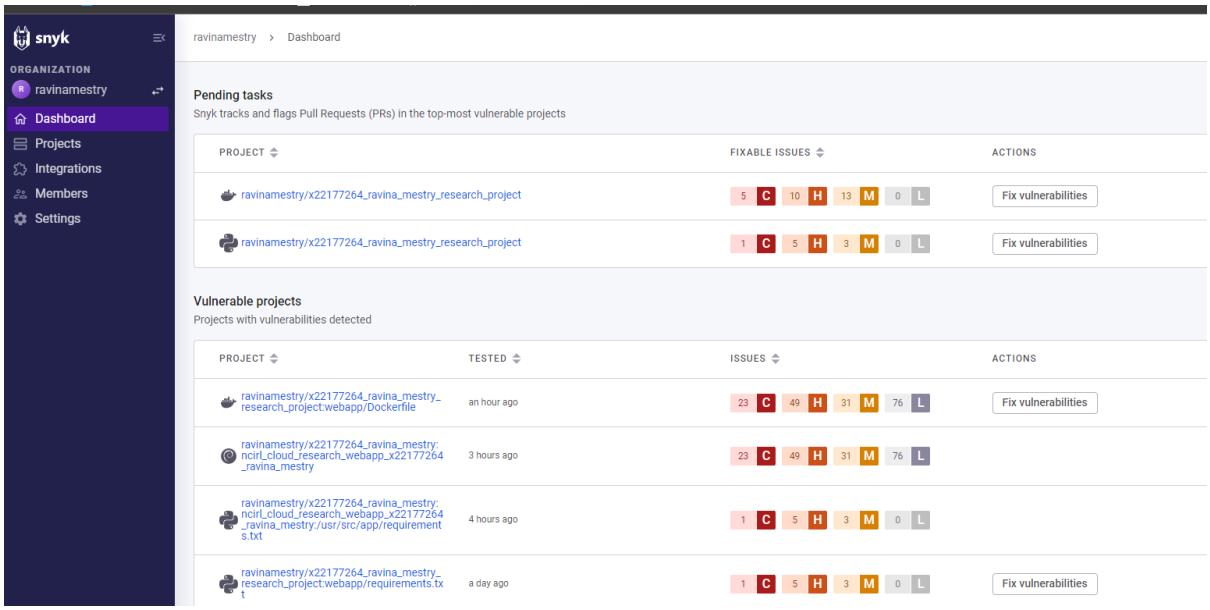

Figure 16: Snyk dashboard

3.8 Datadog

3.8.1 Set up for Tracing on GitHub Actions Workflows

- Configured the GitHub App name ‘Datadog – ResearchProject’ in Datadog integration in Figure 17.
- Edited the Permissions to grant Actions: Read access.
- Configured tracing for GitHub Actions for Enabling CI Visibility for the research repository in Figure 18.
- The Pipelines page in Datadog shows Pipelines and Pipeline Execution with duration and CI status in Figure 19.

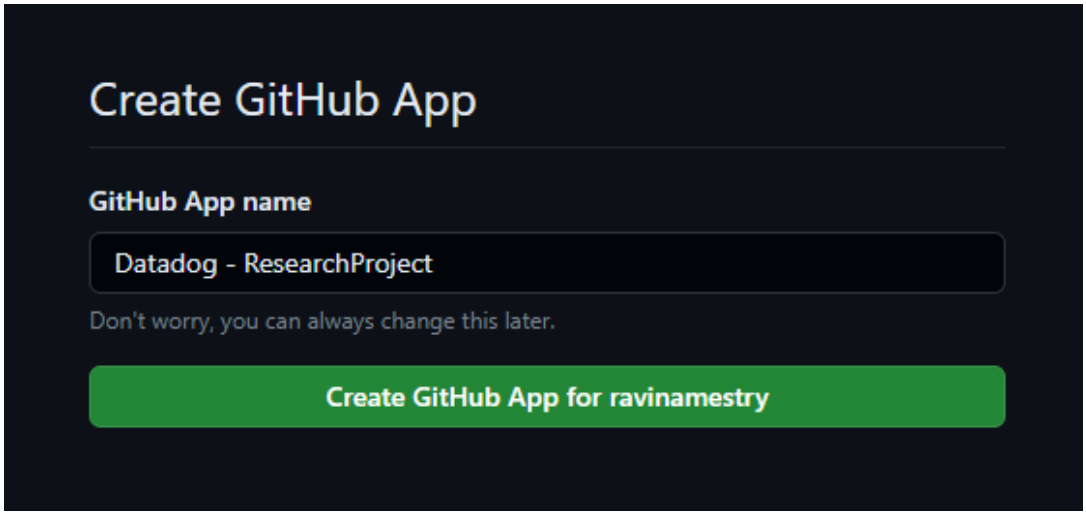

Figure 17: Created GitHub App in Datadog

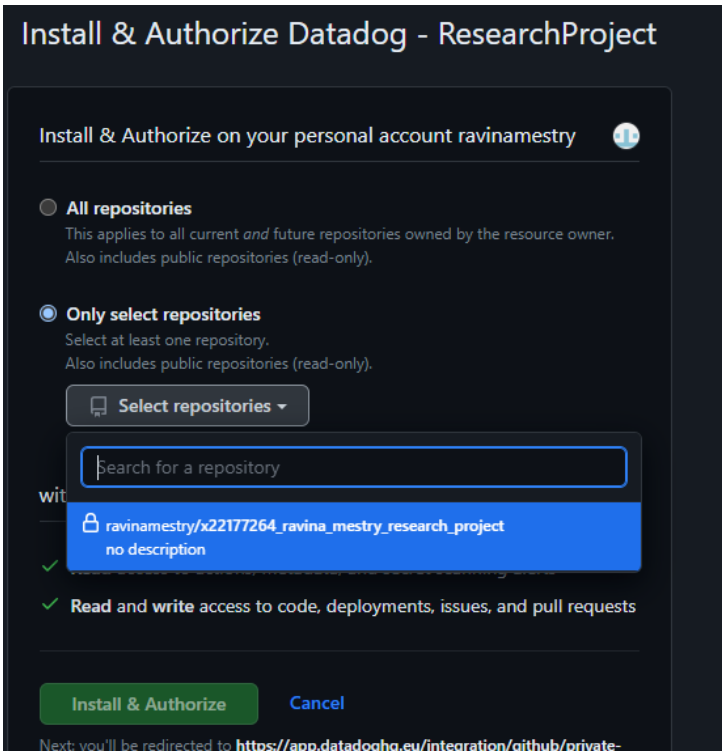

Figure 18: Enabling CI Visibility in Datadog

URL: [Set up Tracing on GitHub Actions Workflows (datadoghq.com)]

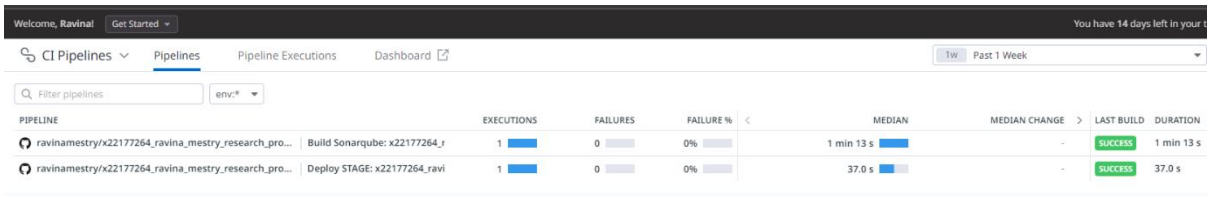

Figure 19: Datadog Pipeline Visibility

3.8.2 Set up for Cloud Security Posture Management (CSPM)

- Datadog Integrations tile clicked on AWS and selected AWS region and Datadog API Key.
- Enabled Cloud Security Posture Management to scan the cloud environment, hosts, and containers.
- Clicked on Launch CloudFormation Template and created stack.
- After the stack is created, on the AWS integration tile in Datadog and clicked Ready.

URL: [Setting Up CSPM (datadoghq.com)]

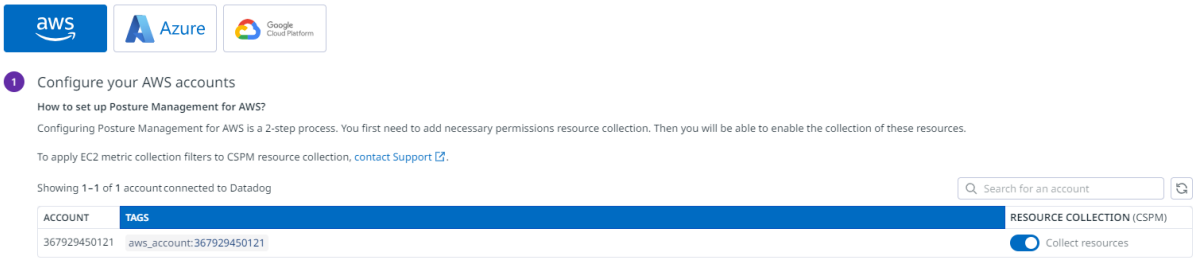

Figure 20: Datadog CSPM Configuration

- Enabled Collect Resources toggle for AWS Resource in Figure 20 in Security>Setup.
- Enabled Cloud Security Posture Management Collection to enable resource collection for CSPM in AWS Integration tile.
- Figure 22 shows the Cloud Security Posture with misconfigured resources and posture score.

```
15
16    DD_API_KEY=${{ secrets.Datadog_API_Key }} DD_SITE="datadoghq.eu" bash -c "$(curl -L https://s3.amazonaws.com/dd-agent/scripts/install_script_agent7.sh)"
17    sudo apt-get update -y
18    sudo apt-get install apt-transport-https curl gnupg -y
19    sudo sh -c "echo 'deb [signed-by=/usr/share/keyrings/datadog-archive-keyring.gpg] https://apt.datadoghq.com/ stable 7' > /etc/apt/sources.list.d/datadog.list"
20    sudo touch /usr/share/keyrings/datadog-archive-keyring.gpg -y
21    sudo chmod a+r /usr/share/keyrings/datadog-archive-keyring.gpg -y
22    curl https://keys.datadoghq.com/DATADOG_APT_KEY_CURRENT.public | sudo gpg --no-default-keyring --keyring /usr/share/keyrings/datadog-archive-keyring.gpg --import --batch
23    curl https://keys.datadoghq.com/DATADOG_APT_KEY_C0962C7D.public | sudo gpg --no-default-keyring --keyring /usr/share/keyrings/datadog-archive-keyring.gpg --import --batch
24    curl https://keys.datadoghq.com/DATADOG_APT_KEY_F14F620E.public | sudo gpg --no-default-keyring --keyring /usr/share/keyrings/datadog-archive-keyring.gpg --import --batch
25    curl https://keys.datadoghq.com/DATADOG_APT_KEY_382E94DE.public | sudo gpg --no-default-keyring --keyring /usr/share/keyrings/datadog-archive-keyring.gpg --import --batch
26    sudo cp -a /usr/share/keyrings/datadog-archive-keyring.gpg /etc/apt/trusted.gpg.d/ -y
27    sudo apt-get update -y
28    sudo apt-get install datadog-agent datadog-signing-keys -y
29    sudo sh -c "sed 's/api_key:.*/api_key: ${{ secrets.Datadog_API_Key }}/' /etc/datadog-agent/datadog.yaml > /etc/datadog-agent/datadog.yaml"
30    sudo -u dd-agent -- datadog-agent import /etc/dd-agent /etc/datadog-agent -y
31    sudo sh -c "chown dd-agent:dd-agent /etc/datadog-agent/datadog.yaml && chmod 640 /etc/datadog-agent/datadog.yaml"
32    sudo systemctl restart datadog-agent.service
```

Figure 21: Datadog agent installation in terraform

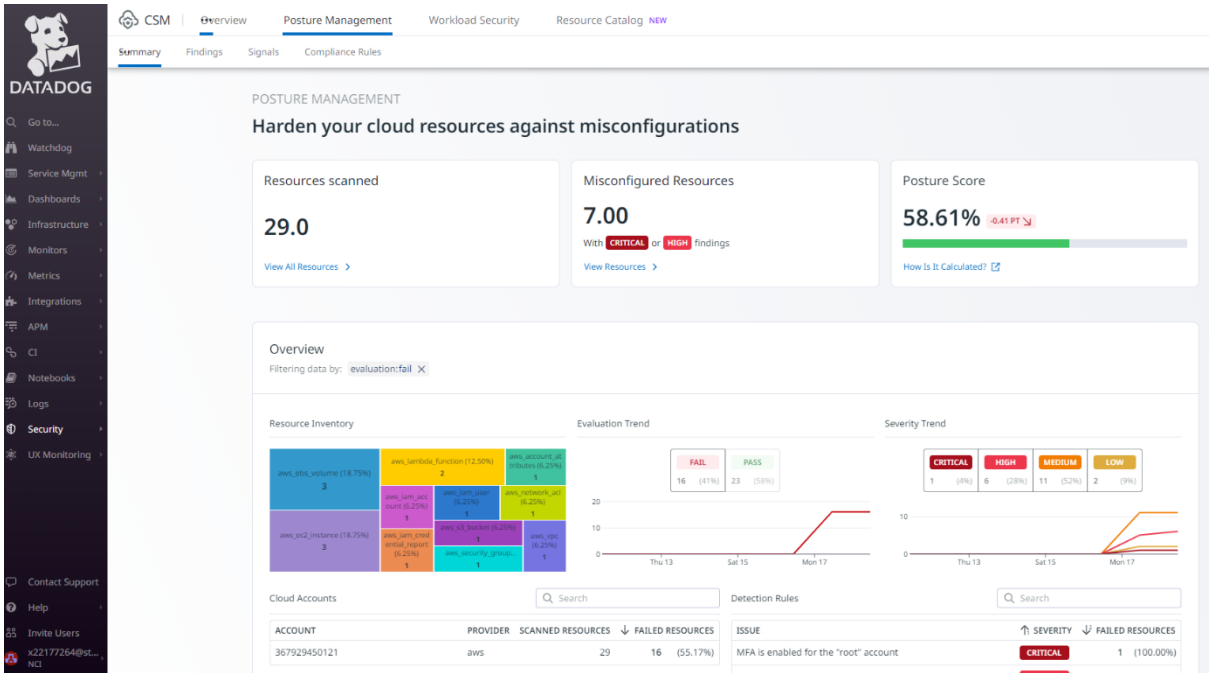

Figure 22: Datadog Posture Management

3.8.3 Set up for Datadog Agent

- Got the steps for installing Datadog agent from the URL mentioned in Table 1 and added the steps in the terraform terraform/installdocker.sh file shown in Figure 21.
- Below command shows the status of the Datadog agent running for the host.

sudo service datadog-agent status

3.9 Super-Linter

- Entered the code in Figure 23 in the .GitHub/workflows/build.yml workflow from the URL mentioned below.
- GitHub Actions will automatically run the Super-Linter workflow whenever there is a pull request or push event to the specified branches.

URL: https://github.com/marketplace/actions/super-linter

```yaml
superlint:
  # Name the Job
  name: Lint Code Base
  # Set the agent to run on
  runs-on: ubuntu-latest
  needs: BuildStart

  ##########################################
  # Grant status permission for MULTI_STATUS #
  ##########################################
  permissions:
    contents: read
    packages: read
    statuses: write

  ################
  # Load all steps #
  ################
  steps:
    ##########################
    # Checkout the code base #
    ##########################
    - name: Checkout Code
      uses: actions/checkout@v3
      with:
        # Full git history is needed to get a proper
        # list of changed files within `super-linter`
        fetch-depth: 0

    ################################
    # Run Linter against code base #
    ################################
    - name: Lint Code Base
      uses: super-linter/super-linter@v5
      env:
        VALIDATE_ALL_CODEBASE: false
        DEFAULT_BRANCH: master
        GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
```

Figure 23: Super-Linter code in build

4 Configuring the Pipeline

- Figure 24 illustrates the summary of CI-CD pipeline with integrated Security tools on various stages of the pipeline designed for the research paper explained in sections.

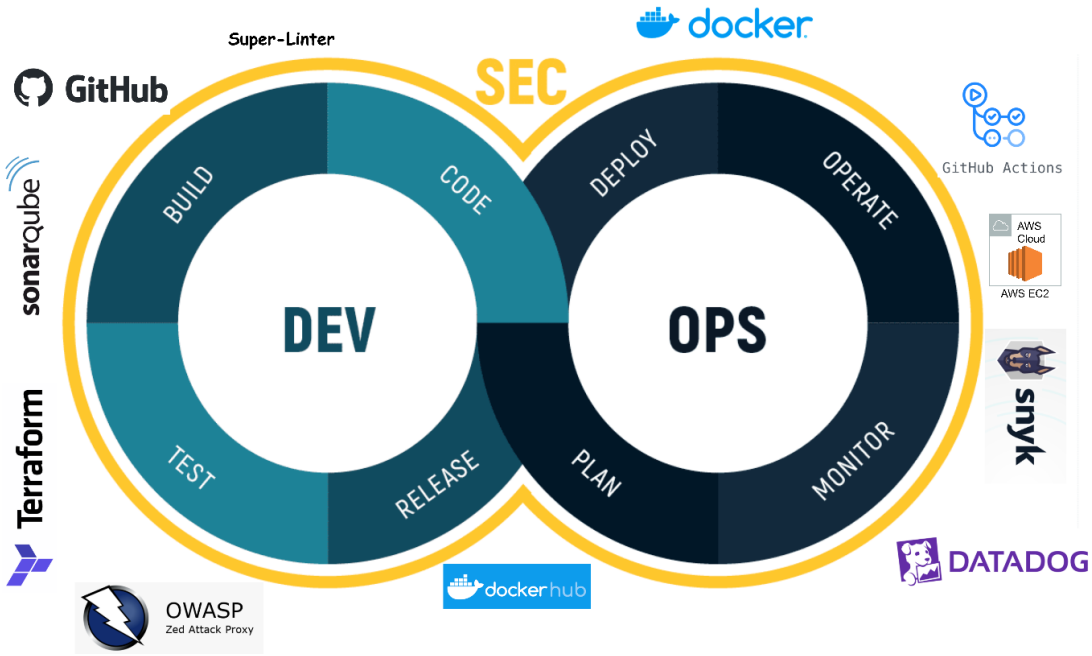

Figure 24: CI-CD Pipeline with Integrated Security Tools

4.1 Infrastructure Deployment Workflow

- Below workflow in Figure 25 is created for infrastructure development, GitHub actions will trigger terraform_create.yml workflow when there are changes in infrastructure requirements from terraform terraform/main.tf path.

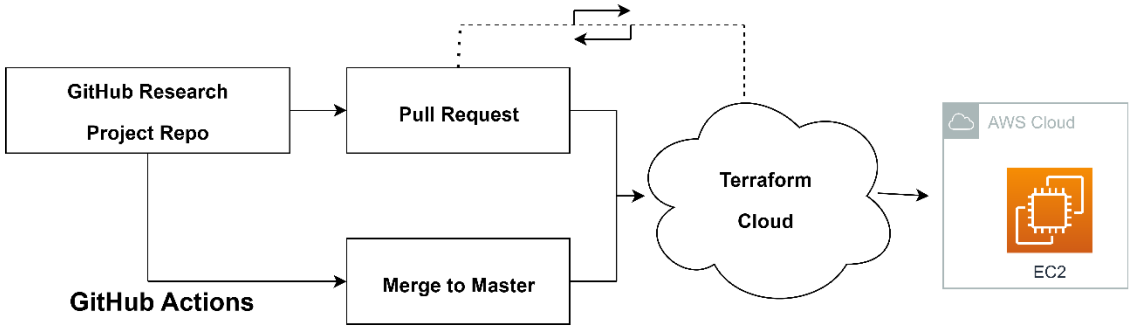

Figure 25: Infrastructure Deployment Workflow

4.1.1 Pull requests steps:

- Figure 26 shows pull request steps and merge to develop and merge to master steps in terraform plan and apply execution.
- On pull request, terraform initiates the directory which has terraform Configuration files and validates the Configuration.
- Terraform Plan shows the plan which has the actions Terraform will take in order.

- Terraform plan status returns whether a plan was successfully generated or not.

4.1.2 Merge to master steps:

- Terraform runs the Apply when Pull request is merged into master branch.
- Terraform Apply the plan and creates the infrastructure according to the plan.
- It retrieves the public IPs of instances and saves them in GitHub secrets shown in Figure 27.

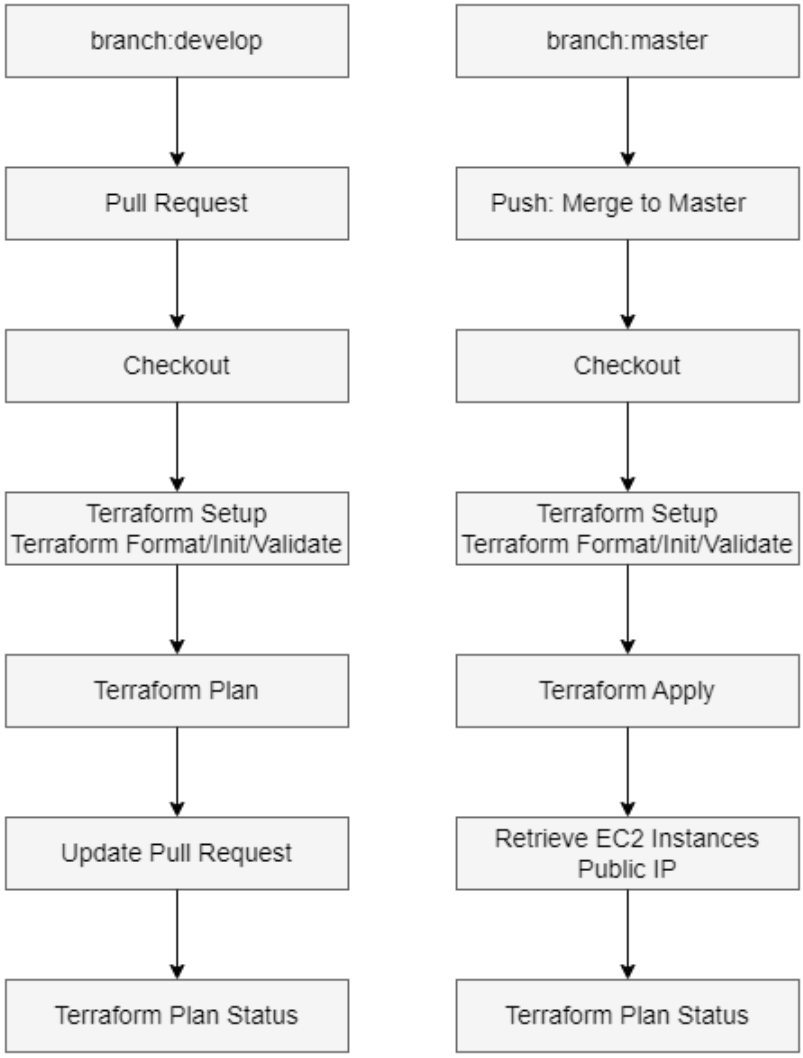

Figure 26: Terraform Plan and Apply Execution

4.2 Application Deployment Workflow

- Develop branch from GitHub repo consists of code change which is to be added to the development and then to Staging environments.
- When the Pull request is generated on Develop branch, GitHub actions will trigger build.yml workflow on develop branch for webapp illustrated in Figure 28.

Figure 27: GitHub Secrets for Public IP

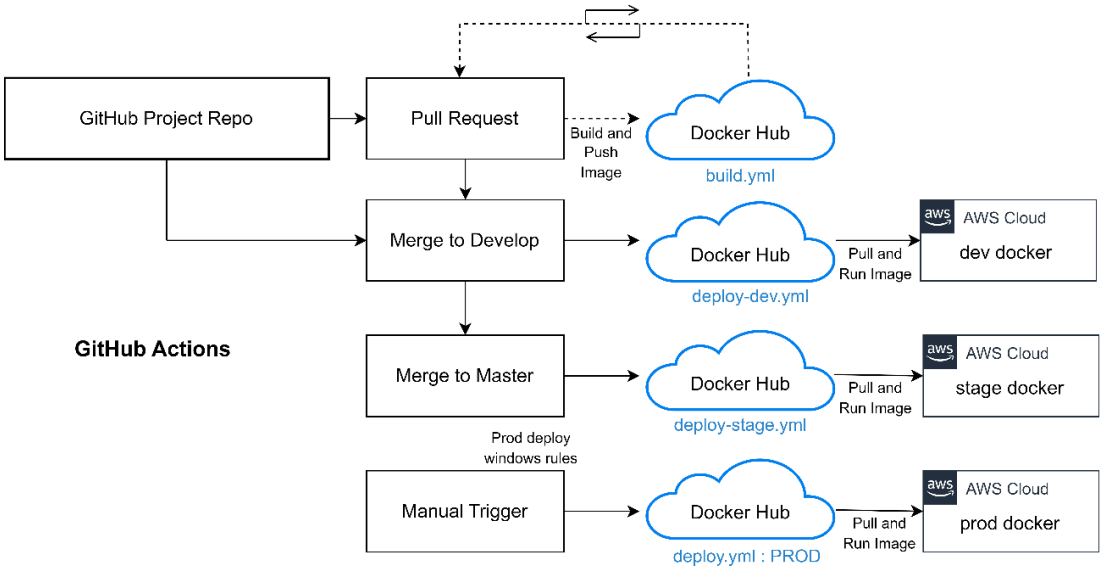

Figure 28: Application Deployment Workflow

- It builds and push the web app and Docker Hub image with tag 'x22177264_ravina_mestry/project:ncirl_cloud_research_webapp_x22177264_ravina_mestry'.
- When the develop branch is merged into develop branch shown in Figure 28, GitHub actions will trigger deploy-dev.yml workflow on develop branch. It deploys the application in development environment from GitHub Action secrets x22177264_ravina_mestry_research_project_dev_PUBLIC_IP'.
- It gets docker-compose.yml file and removes existing Docker container and image, gets Login for Docker, and runs Docker to deploy the application in development environment from GitHub Action secrets x22177264_ravina_mestry_research_project_dev_PUBLIC_IP'.

- When the develop branch is merged into master branch, GitHub actions will trigger deploy-stage.yml workflow on master branch.
- When GitHub actions is manually triggered with input PROD in the workflow in Figure 29, removes existing Docker container and image and the deploy.yml workflow and deploys the application in Production environment from GitHub Action secrets x22177264_ravina_mestry_research _project_prod_PUBLIC_IP'.

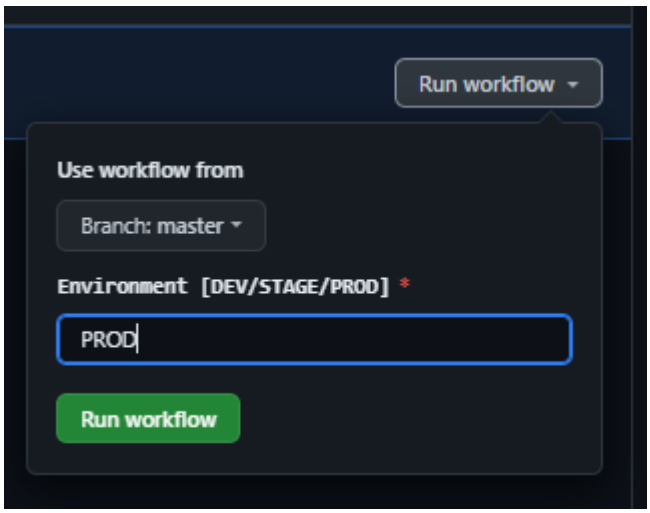

Figure 29: GitHub Action Deploy workflow

4.3 Destroy Infra Workflow

- The destroy infra workflow destroys no longer needed infrastructure. Resources like Dev or Stage are destroyed and created when needed. The Terraform destroy command terminates the resources in workspace. Figure 30 shows the terraform_destroy.yml execution destroying the dev, stage, and prod AWS EC2 instances.

terraform_destroy.yml
terraform_destroy_sonarqube.yml
terraform_destroy_zap.yml

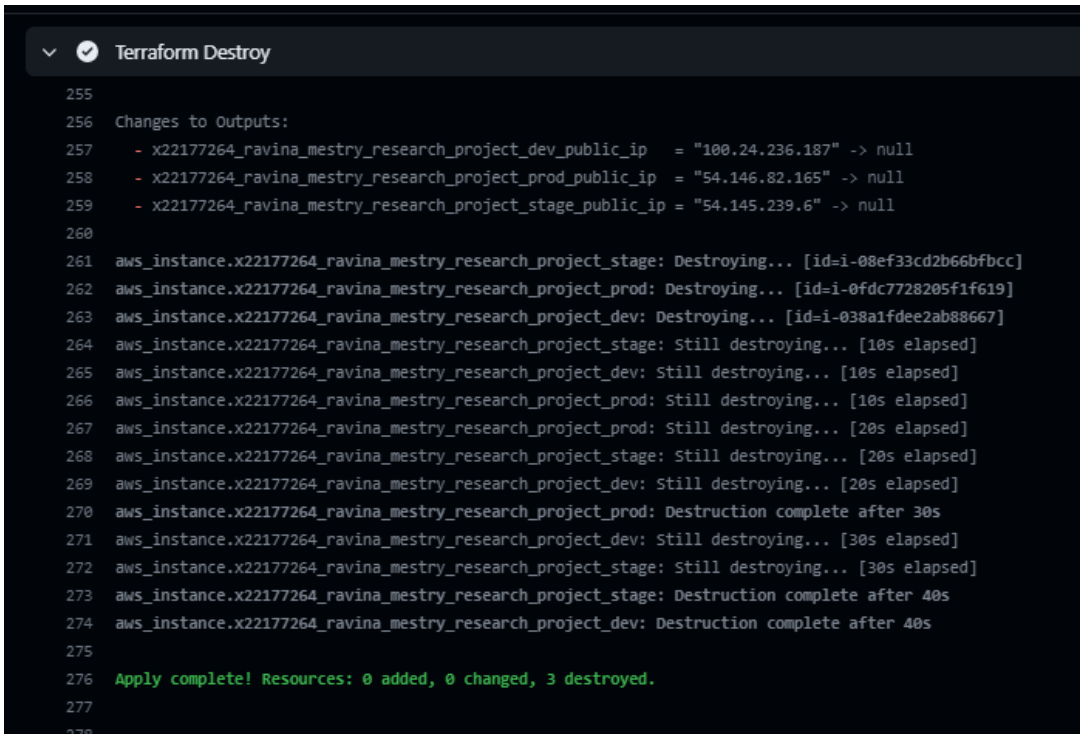

Figure 30: Destroy Infrastructure Workflow