National College of Ireland

Configuration Manual

MSc Research Project
Cloud Computing

Jiyoung Kim
Student ID: 24142816

School of Computing
National College of Ireland

Supervisor: Shreyas Setlur Arun

National College of Ireland
Project Submission Sheet
School of Computing

National College of Ireland

Student Name:	Jiyoung Kim
Student ID:	24142816
Programme:	Cloud Computing
Year:	2025
Module:	MSc Research Project
Supervisor:	Shreyas Setlur Arun
Submission Due Date:	11/12/2025
Project Title:	Configuration Manual
Word Count:	2257
Page Count:	9

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:	Jiyoung Kim
Date:	5th December 2025

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

Jiyoung Kim
24142816

1 Software Requirements

To configure this proposed pipeline, it requires access to several cloud-based tools and services that support automated pipeline. Prior to the commencement of set up, the following are necessary:

1.1 Cloud & External Services

- **AWS Learner Lab** – It is a temporary AWS environment that is used for testing and deployment. It provides temporary IAM credentials and gives the right to service such as Elastic Beanstalk, DynamoDB, and CloudWatch.
- **GitHub Repository** – It stores the source code, Terraform files, and CI workflows. It also records all the modifications to the project, or any changes are logged with version history. Moreover, GitHub Actions is applied as a trigger of automation in the pipeline.
- **SonarCloud** – It performs a static analysis of the Django application code, and it generates reports in JSON format, which is analyzed in the metrics step.
- **Terraform** – It is used to define and create cloud resources by following the Infrastructure-as-Code approach.
- **tfsec** – It scans the Terraform files and examines the security misconfigurations or rule violations of IaC layer.
- **OWASP ZAP (Full Scan)** – A full scan mode is more time consuming, as it crawls and tests a wide range of endpoints.
- **OWASP Juice Shop** – It is a deliberately vulnerable web application that is used as high-risk evaluation target for static and dynamic stages of the pipeline. (sources used: `https://github.com/juice-shop/juice-shop`)

1.2 Necessary Accounts

This pipeline requires several accounts or access tokens to operate the implementation. These must be set up in advance:

- **GitHub Account** – The repository needs to be created, source code is to be maintained, and GitHub Actions should be used as part of CI workflow.

- **SonarCloud Account** – The project workspace must be created, and the project token must be generated that enables the pipeline to perform a static analysis using the CI environment.
- **AWS Learner Lab Access** – The access is required to launch the Django application in Elastic Beanstalk and perform monitoring of the logs and metrics in CloudWatch during testing.

Each account must be activated and qualified of generating the API tokens or credentials, if needed.

1.3 Tool Versions

These following versions were used to implement the framework. The latest version can also be used and can be guaranteed to yield the same results across the CI process of implementation and deployment.

- **Terraform**: 1.13.4
- **tfsec**: 1.28.14
- **OWASP ZAP**: 2.14.0 (Full Scan Mode)
- **GitHub Actions runner**: ubuntu-latest
- **Python**: 3.9 (Cloud9 environment default)
- **Django**: 4.2.25

Maintaining the same version or the latest ensures that the consistent results are derived in the execution and deployment process of the CI. Even though variability in tool versions can influence the check time and output, it has no effect on the pipeline structure.

2 System Architecture Overview

This section describes the general architecture adopted by the proposed security automation framework. The system interconnects a Django application with Infrastructure-as-Code resources, a CI/CD, and various security tools to conduct automated static, IaC, and dynamic security checks to be completed prior to deployment. The entire process of components is depicted in the following diagram.

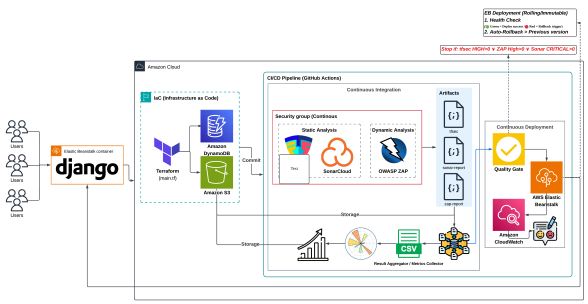

Figure 1: System Architecture Diagram

The architecture diagram above is divided to six layers and each layer takes care of a specific section of the pipeline:

- **Application layer** – This layer contains the Django application deployed to AWS Elastic Beanstalk. It is the primary point of both dynamic and static testing.
- **Infrastructure-as-Code Layer (Terraform)** – This layer provisions DynamoDB and S3 resources. The S3 bucket is implemented in Terraform, but it cannot be used for storing the outputs due to the limitation on access rights such as permanent credential keys.
- **CI/CD Layer (GitHub Actions)** – This pipeline is automatically used whenever developers push code changes. It performs a static analysis (SonarCloud), IaC scanning (tfsec), dynamic analysis (OWASP ZAP), metrics aggregation, and Quality Gate validation.
- **Security Automation Layer** – In this step, it consists of two organizations with static analysis and dynamic scanning. While both tfsec and SonarCloud analyze the potential issues of structural code, OWASP ZAP Full Scan is scanned during runtime.
- **Deployment and Monitoring Layer** – In case of the passing of the Quality Gate, the validated build is deployed to AWS Elastic Beanstalk. CloudWatch is considered to view logs and performance metrics.

These layers collectively constitute the entire DevSecOps process applied in the implementation.

3 Step-by-Step Configuration Process

This section describes the steps required for configuration, from configuring the repository to connecting the deployment to Elastic Beanstalk, as well as each security tool in the proposed pipeline.

3.1 Repository Setup

1. Create and clone a public repository
   Before clone into AWS Cloud9, it needs to create a new GitHub repository.

   ```
   git clone https://github.com/Jiyoung0716/Thesis
   cd thesis-project
   ```

2. Organize the project structure
   The repository is made up of the following main folders and files:

   ```
   .github/workflows/       # Contains the GitHub Actions workflow for CI
   infra/terraform/         # Terraform files defining the cloud setup
   metrics_output/          # Output reports and charts
   static/ , staticfiles/   # Static files prepared for Django deployment
   thesis/                  # Django project source code
   Procfile                 # Elastic Beanstalk for deployment
   ```

```
sonar-project.properties # Configuration file for SonarCloud
analyze_security.py      # Script for collecting and processing scan results
quality_gate.py          # Script that applies the Quality Gate conditions
requirements.txt         # Dependencies on this project
```

3. Commit and push the initial structure
   After organizing the project structure, an initial commit for all files is pushed to the main branch.

   ```
   git add .
   git commit -m "<written any changes for recognition>"
   git push origin main
   ```

3.2 Terraform Configuration Setup

1. Verify Terraform installation

   - The Terraform version is checked in the Cloud9 environment using the *terraform* command:

     ```
     terraform -version
     ```

   - If Terraform was not in the environment, it should be installed in order to perform:

     ```
     sudo yum install -y terraform
     ```

2. Build the Terraform both directory and file
   Make use of the designed path at the terminal to create all regarding IaC files:

   ```
   infra/terraform/
   ```

   Through the path, files are defined in a file named *main.tf*.

3. Create the *main.tf* file defined
   Update *main.tf* file, including the following one:

   ```
   provider "aws" {
     region = var.region
   }

   resource "aws_dynamodb_table" "signups" {
     name         = "thesis-signups"
     billing_mode = "PAY_PER_REQUEST"

     hash_key  = "role"
     range_key = "username"

     attribute {
       name = "role"
       type = "S"
     }

     attribute {
   ```

```
    name = "username"
    type = "S"
  }

  point_in_time_recovery {
    enabled = true
  }

  server_side_encryption {
    enabled = true
  }
}
```

Both `point_in_time_recovery` and `server_side_encryption` prevent reporting tables as insecure by **tfsec**.

4. Initialise Terraform
   Use the *cd* command, move into Terraform workspace and then initialize it using *terraform init*.

   ```
   cd infra/terraform
   terraform init
   ```

   This is to set up the working directory and downloads the required providers.

3.3 Static Application Security Testing (SonarCloud)

1. Create the SonarCloud project
   The SonarCloud project should be connected to the same GitHub repository. Only the CI workflow should be run in the scan, and the SonarCloud automatic analysis option must be turned off.

2. Add the Sonarcloud configuration file
   A `sonarproject.properties` file should be placed in the repository root. After creating the file, it is composed of the following code:

   ```
   sonar.projectKey=Jiyoung0716_Thesis
   sonar.organization=jiyoung0716
   sonar.host.url=https://sonarcloud.io

   sonar.sources=.
   sonar.python.version=3.10
   ```

3. Register the secret token for authentication
   The generated SonarCloud token is added to the GitHub repository as a token with the name `SONAR_TOKEN`. It is important to note that this token is given once only.

4. Include the scan step in the CI workflow
   Insert the following step to yaml file within the `.githubworkflows`:

   ```
   - name: SonarCloud Scan
     uses: SonarSource/sonarcloud-github-action@v2
     env:
       SONAR_TOKEN: ${{ secrets.SONAR_TOKEN }}
   ```

3.4 IaC Static Security Testing (tfsec)

1. check tfsec installation

   - In Cloud9, the version of tfsec is verified using the following command:

     ```
     tfsec --version
     ```

   - In case of absence of tfsec, it can be installed by:

     ```
     sudo yum install -y tfsec
     ```

2. Add tfsec to the CI workflow
   The following command is required to to scan the Terraform directory to the GitHub Actions workflow:

```
- name: Run tfsec
  uses: aquasecurity/tfsec-action@v1.0.3
  with:
    working_directory: infra/terraform
    additional_args: --format json --out tfsec.json
    soft_fail: true
```

3.5 Dynamic Security Testing (OWASP ZAP Full Scan)

Since the scan is performed through the official GitHub Actions, OWASP ZAP does not require a local machine.

1. Scan the application
   Django application is launched within the workflow to be accessible by ZAP:

```
- name: Start Django app
  run: |
    python manage.py migrate --noinput || true
    python manage.py runserver 0.0.0.0:8000 &
    for i in $(seq 1 30); do
      if curl -sSf http://localhost:8000/ >/dev/null; then
        break
      fi
      sleep 2
    done
```

2. Run the ZAP Full Scan
   The dynamic analysis is conducted by the official full-scan action:

```
- name: ZAP Full Scan
  uses: zaproxy/action-full-scan@v0.12.0
  continue-on-error: true
  with:
    target: 'http://localhost:8000'
    fail_action: false
    allow_issue_writing: false
```

3.6 Metrics Aggregation & Visualization Script

This is a step for gathering the results of all the security tools and creating aggregated metrics and charts.

1. Download all scan reports
   The outputs of the past jobs (SonarCloud, tfsec and OWASP ZAP) are downloaded into a local directory within the workflow:

   ```
   - name: Download all reports artifacts
     uses: actions/download-artifact@v4
     with:
       path: reports
   ```

   Subdirectories in the reports folder after this step include `sonarcloud-report`, `tfsec-report`, and `zap-report`.

2. Install related dependencies
   A Python environment is set up with the necessary libraries of metrics aggregation and plotting:

   ```
   - name: Install analysis dependencies
     run: |
       python -m pip install --upgrade pip
       pip install matplotlib pandas
   ```

3. Run the metrics script
   The following script is run `analyze_security.py`, which reads the output of the JSON and produces metrics:

   ```
   - name: Run metrics & graph generator
     run: |
       python3 analyze_security.py
   ```

   The script performs the following tasks:

   - Populates the JSON report of the tfsec, SonarCloud, and OWASP ZAP loads under the `reports` folder.
   - Counts findings by severity for each tool and generates two CSV files both named `metrics.csv` for a summary and `metrics_detailed.csv` for the sake of detail.
   - Generates bar charts on each tool individually and combined severity charts which are stored as PNG images in the `metrics_output` directory.

   Tools can be compared, and the overall security posture of the application can be summarised using the resulting CSV files and images in `metrics_output`.

3.7 Quality Gate

This phase decides whether the pipeline should be deployed in accordance with the combined security measures or not.

1. Download the metrics artifact
   All metrics created in the last step are downloaded to the desired directory:

```
- name: Download metrics artifact
  uses: actions/download-artifact@v4
  with:
    name: metrics-report
    path: metrics_output
```

2. Run the Quality Gate script
   Once the Python environment is set up, the script `quality_gate.py` is executed to evaluate the severity score and determine whether the gate has been passed.

```
- uses: actions/setup-python@v5
  with:
    python-version: '3.10'

- name: Run Quality Gate Check
  run: |
    python3 quality_gate.py
```

   The script also loads the summary and detailed CSV files generated during the metrics stage and add the findings together according to severity. The inspection focuses on the next level of blocking:

```
BLOCKING_SEVERITIES = ["CRITICAL", "HIGH"]
```

   The blocking count does not include a small set of recurring ZAP alerts:

```
ALLOWED_ZAP_HIGH_MESSAGES = [
    'Server Leaks Version Information via "Server" HTTP Response Header Field',
    "CSP: Failure to Define Directive with No Fallback",
    "GET for POST",
]
```

   In case there are any non-excluded results at a blocking severity then the script terminates with `exit 1`, leading to the Quality Gate failure. In case none are used, the script closes with the `exit 0`, which lets the pipeline continue with the deployment.

3.8 Deployment to AWS Elastic Beanstalk

1. Prepare the deployment package
   The application is packaged as a **Zip** file before uploading it to Elastic Beanstalk.

   The folders that are not actually needed when the project is being deployed, such as *virtual environments*, *cached files*, *Git metadata*, or *analysis outputs*, can be omitted to make the size of the package minimal and to decrease the time to deploy.

   The archive should contain the Django project, `requirements.txt`, and the `Procfile` at the project root.

2. Configure the Procfile
   Elastic Beanstalk uses a `Procfile` to start the application. The file is put under root directory and includes:

```
web: gunicorn thesis.wsgi:application
```

The code above not only prevents Gunicorn issues during deployment, but also helps load the Django WSGI process.

3. WSGIPath in Elastic Beanstalk
   Once the deployment package has been uploaded, the environment should be made to refer to the appropriate WSGI file.
   In the AWS console:
   `Elastic Beanstalk → Environments → <your-environment> → Configuration → Updates, monitoring, and logging → WSGIPath`
   Set the following value: `thesis/wsgi.py` This configuration is essential without which the application will not boot up.

4. Establish the Elastic Beanstalk environment

   - Go to a Elastic Beanstalk console.
   - Push the create button to make a new Elastic Beanstalk application.
   - Choose the Python 3.9 platform, which is equivalent to Cloud9.
   - Post the zip file created as the version of the application.

5. Deploy and check the application When the environment gets **Healthy**, open the given URL (CNAME) to ensure that Django application is operating successfully.
   The updates in the future may be provided by building a new zip file (using the same exclusion pattern) and uploading it as a new version of the application.

4 Troubleshooting Examples

This section outlines typical problems that can be experienced during configuration.
In case of any of the following problems, consult the corresponding section of this manual.

- SonarCloud scan does not run
  Make sure the configuration instructions are followed in Section 4.3.

- tfsec fails to produce
  Go to the IaC configuration process in Section 4.4

- Elastic Beanstalk is unhealthy.
  Reconfirm the location of the WSGI path in Section 4.8 and re-verify the deployment settings.