National College of Ireland

Configuration Manual

MSc Research Project
MSc Cybersecurity

Aniket Vishwakarma
Student ID: x23211768

National College of Ireland

Supervisor: Prof Jawad Salahuddin

National College of Ireland

MSc Project Submission Sheet

Student Name:	Aniket Vishwakarma	
Student ID:	x23211768	
Programme:	Msc Cybersecurity	**Year: 2025**
Module:	Research Practicum	
Supervisor:	Professor Jawad Salahuddin	
Submission Due Date:	15th September 2025	
Project Title:	Automating Threat Intelligence: The Use of AI in Cyber Threat Prediction and Mitigation	
Word Count:	1497 **Page Count:** 6	

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: Aniket Vishwakarma

Date: 15th September 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

Aniket Vishwakarma
x23211768

1 Introduction

1.1 Purpose of this Manual

This document provides detailed step-by-step instructions for configuring and deploying the AI-powered Cyber Threat Prediction system. The purpose of this manual is to guide a user through the complete setup process, from establishing a local development environment to deploying the final application on a cloud infrastructure using a fully automated CI/CD pipeline. By following these instructions, a user can replicate the entire production-ready system.

1.2 Intended Audience

This manual is intended for system administrators, DevOps engineers, and developers who have a foundational understanding of Python, cloud computing (specifically AWS), version control with Git, and command-line interfaces. Familiarity with the concepts of web applications and CI/CD pipelines is also beneficial.

1.3 System Overview

The system has the following three main components that will be configured:

1. Machine Learning Application: A web application written in Python with the Flask framework that operates a trained Random Forest model to predict cyber threats.
2. Cloud Infrastructure: A collection of resources allocated from the Amazon Web Services (AWS) to host the application, such as a custom VPC, a public subnet, and an EC2 instance.
3. Automation Pipeline: A Continuous Integration and Continuous Deployment (CI/CD) workflow that, thanks to GitHub Actions, enables automatic deployment for the application to AWS infrastructure on code changes.

2 Prerequisites

Before configuration, ensure the following software and accounts are in place.

2.1 Software and Tools

- Python (3.11 or later): The programming language in which the application is written.
- Git: The versioning system for managing source code.
- AWS CLI: The tool for command-line interaction and provisioning of AWS resources.

2.2 Account Requirements

- AWS Account: An active account under Amazon Web Services with permissions to create IAM users, VPCs, EC2 instances, etc.
- GitHub Account: An account on GitHub to host the source code repository and manage the CI/CD pipeline.

3 Local Development Environment Setup

This section explains the setup of the project on your local computer.

3.1 Cloning the Repository

First, clone the GitHub repository to get the source code.

```
git clone https://github.com/Aniket-Vishwakarma-ops/ids.git
cd ids
```

3.2 Setting up a Python Virtual Environment

Best practice is to create a virtual environment to control the project's dependencies.

```
# Create virtual environment
python3.11 -m venv venv

# Activate the virtual environment
# On macOS/Linux:
source venv/bin/activate
# On Windows:
venv\Scripts\activate
```

3.3 Installing Dependencies

The project includes a requirements.txt file listing all necessary Python libraries.

```
pip install -r requirements.txt
```

This will install Flask, Joblib, Scikit-learn, Pandas, and other required packages.

4 AWS Infrastructure Provisioning

This section details the steps to create the necessary cloud infrastructure on AWS using the AWS CLI.

4.1 Configuring the AWS CLI

You must first configure the AWS CLI with your credentials. Create an IAM user in your AWS console with AdministratorAccess and generate an access key.

CRITICAL SECURITY NOTE: Never expose your secret access keys. The credentials provided in the original research document should be considered compromised and must be deleted immediately. Always use secure methods like IAM roles for production environments.

Run the following command and enter your credentials when prompted:

```
aws configure
AWS Access Key ID [None]: YOUR_ACCESS_KEY_ID
AWS Secret Access Key [None]: YOUR_SECRET_ACCESS_KEY
Default region name [None]: eu-north-1
Default output format [None]: json
```

4.2 Creating an EC2 Key Pair

This key pair is required for secure SSH access to your EC2 instance.

```
aws ec2 create-key-pair --key-name idskeypair --query 'KeyMaterial' --output text > idskeypair.pem
```

This command creates a key pair named idskeypair and saves the private key to a file named idskeypair.pem. Secure this file:

```
# On macOS/Linux:
chmod 400 idskeypair.pem
```

```
# On Windows (using PowerShell):
icacls.exe idskeypair.pem /inheritance:r
icacls.exe idskeypair.pem /grant:r "$($env:username):(r)"
```

4.3 Setting up the Network Infrastructure

Execute the following commands in order. **You must replace the placeholder IDs with the actual IDs returned by the previous command.**

1. **Create a VPC:**

```
aws ec2 create-vpc --cidr-block 10.0.0.0/16 --tag-specifications "ResourceType=vpc,Tags=[{Key=\"Name\",Value=\"MyVPC\"}]"
# Note the "VpcId" from the output (e.g., vpc-0ebb9f8b34e9b43b1)
```

2. **Create a Subnet:**

```
aws ec2 create-subnet --vpc-id <VPC_ID_FROM_STEP_1> --cidr-block 10.0.1.0/24 --tag-specifications "ResourceType=subnet,Tags=[{Key=\"Name\",Value=\"MySubnet\"}]"
# Note the "SubnetId" from the output (e.g., subnet-0674d5bf31814ab7c)
```

3. **Create and Attach an Internet Gateway:**

```
aws ec2 create-internet-gateway --tag-specifications "ResourceType=internet-gateway,Tags=[{Key=\"Name\",Value=\"MyIGW\"}]"
# Note the "InternetGatewayId" (e.g., igw-0e42a90d3a773fb94)

aws ec2 attach-internet-gateway --internet-gateway-id <IGW_ID> --vpc-id <VPC_ID>
```

4. **Create and Configure a Route Table:**

```
aws ec2 create-route-table --vpc-id <VPC_ID> --tag-specifications "ResourceType=route-table,Tags=[{Key=\"Name\",Value=\"MyRouteTable\"}]"
# Note the "RouteTableId" (e.g., rtb-0225b93403133d8ac)

aws ec2 create-route --route-table-id <ROUTE_TABLE_ID> --destination-cidr-block 0.0.0.0/0 --gateway-id <IGW_ID>

aws ec2 associate-route-table --subnet-id <SUBNET_ID> --route-table-id <ROUTE_TABLE_ID>
```

4.4 Configuring the Security Group

This acts as a firewall for the EC2 instance.

1. **Create the Security Group:**

```
aws ec2 create-security-group --group-name MySG --description "Allow SSH, HTTP, and App access" --vpc-id <VPC_ID>
# Note the "GroupId" (e.g., sg-053ae4539d69b17e9)
```

2. **Add Inbound Rules:**

```
# Allow SSH (Port 22)
aws ec2 authorize-security-group-ingress --group-id <GROUP_ID> --protocol tcp --port 22 --cidr 0.0.0.0/0

# Allow HTTP (Port 80)
aws ec2 authorize-security-group-ingress --group-id <GROUP_ID> --protocol tcp --port 80 --cidr 0.0.0.0/0

# Allow Flask App (Port 5000)
aws ec2 authorize-security-group-ingress --group-id <GROUP_ID> --protocol tcp --port 5000 --cidr 0.0.0.0/0
```

4.5 Launching the EC2 Instance

Use an appropriate Amazon Machine Image (AMI) for your region. The ID ami-00c8ac9147e19828e is for Amazon Linux in eu-north-1.
aws ec2 run-instances --image-id ami-00c8ac9147e19828e --count 1 --instance-type t3.micro --key-name idskeypair --security-group-ids <GROUP_ID> --subnet-id <SUBNET_ID> --associate-public-ip-address --tag-specifications "ResourceType=instance,Tags=[{Key=\"Name\",Value=\"MyEC2\"}]"
After the instance is running, find its Public IPv4 address from the AWS EC2 console.

5 GitHub and CI/CD Pipeline Configuration

5.1 Repository Setup

Push your local project code to a new repository on your GitHub account.

5.2 Configuring GitHub Secrets

In your GitHub repository, navigate to Settings > Secrets and variables > Actions. Add the following repository secrets:

- **EC2_HOST**: The Public IPv4 address of your EC2 instance (e.g., 13.61.179.124).
- **EC2_USER**: The default username for your EC2 instance's AMI (e.g., ec2-user for Amazon Linux).
- **EC2_KEY**: The entire content of your idskeypair.pem file. Open the file, copy all the text (including -----BEGIN RSA PRIVATE KEY----- and -----END RSA PRIVATE KEY ----), and paste it as the value for this secret.

5.3 Creating the GitHub Actions Workflow

Create a file at the following path in your repository: .github/workflows/deploy.yml. Paste the following content into the file:

```
name: Deploy Flask App to EC2

on:
  push:
    branches:
      - main

jobs:
  deploy:
    runs-on: ubuntu-latest

    steps:
    - name: Checkout source
      uses: actions/checkout@v4

    - name: Save EC2 private key
      run: |
        echo "${{ secrets.EC2_KEY }}" > idskeypair.pem
        chmod 600 idskeypair.pem

    - name: Deploy Flask App via SSH
      run: |
        ssh -o StrictHostKeyChecking=no -i idskeypair.pem ${{ secrets.EC2_USER }}@${{ secrets.EC2_HOST }} << 'EOF'
```

```
# Install dependencies and Python if not present
sudo dnf groupinstall -y "Development Tools"
sudo dnf install -y gcc openssl-devel bzip2-devel libffi-devel zlib-devel wget make git

if ! command -v python3.11 &> /dev/null; then
  cd /usr/src
  sudo wget https://www.python.org/ftp/python/3.11.5/Python-3.11.5.tgz
  sudo tar xzf Python-3.11.5.tgz
  cd Python-3.11.5
  sudo ./configure --enable-optimizations
  sudo make -j$(nproc)
  sudo make altinstall
fi

# Clean old directory and clone fresh
rm -rf ~/ids
git clone https://github.com/Aniket-Vishwakarma-ops/ids.git ~/ids

# Kill existing Flask process if running
pkill -f "python3.11 app.py" || true

cd ~/ids

# Install dependencies
python3.11 -m pip install -r requirements.txt

# Start Flask app in the background
nohup python3.11 app.py > flask.log 2>&1 &
EOF
```

6 Deployment and Verification

6.1 Triggering the Deployment

To deploy the application, simply commit and push a change to the main branch of your GitHub repository.

```
git add .
git commit -m "Initial deployment"
git push origin main
```

This will trigger the GitHub Actions workflow. You can monitor its progress in the "Actions" tab of your repository.

6.2 Verifying the Application

Once the workflow completes successfully (indicated by a green checkmark), the application should be live. Open a web browser and navigate to: http://<YOUR_EC2_PUBLIC_IP>:5000

You should see the home page of the threat detection application. You can test its functionality using the prediction forms.

6.3 Troubleshooting

If the application is not accessible or not working correctly:

1. **Check the GitHub Actions Log:** Look for any errors in the workflow execution log.

2. **Check the Application Log on EC2:** Connect to your EC2 instance via SSH and check the log file for any Python errors.

```
ssh -i idskeypair.pem ec2-user@<YOUR_EC2_PUBLIC_IP>
# Once connected:
cat ~/ids/flask.log
```

7 Conclusion

By completing the steps outlined in this manual, we have successfully configured and deployed a sophisticated, AI-driven cyber threat detection system. We have provisioned a secure and isolated cloud network on AWS, launched a server to host the application, and established a fully automated CI/CD pipeline. The result is a production-ready system where any update to the source code is automatically deployed, ensuring that the application is always running the latest version. This setup not only demonstrates a powerful application of machine learning but also embodies modern best practices in cloud infrastructure management and DevOps, creating a system that is robust, scalable, and maintainable.