National College of Ireland

Configuration Manual

MSc Research Project
Msc In Cloud Computing

Adhithya Narendran Sundar
Student ID: X23421215

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:	Adhithya Narendran Sundar
Student ID:	X23421215
Programme:	Msc In Cloud Computing
Year:	2025
Module:	MSc Research Project
Supervisor:	Sean Heeney
Submission Due Date:	11/12/2025
Project Title:	Configuration Manual
Word Count:	2,880
Page Count:	18

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:	
Date:	11th 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

Adhithya Narendran Sundar
X23421215

1 Introduction

1.1 Purpose of This Configuration Manual

This Configuration Manual gives a brief, end-to-end description of the deployment and operation of the research system:

"An AI Driven Unified QA Framework for Serverless APIs: Automated Functional, Behavioural and Performance Testing in CI/CD Pipelines."

Its goal is to allow assessors and practitioners to reproduce the framework reliably by providing the key steps for setting up the serverless workloads, testing tools, observability stack, and CI/CD automation involved in supporting the project. The manual also serves as a reference for extending or reconfiguring components for future development.

1.2 Scope

The Unified QA Framework is a complete serverless quality assurance environment, which is a combination of compute, testing, metrics, and adaptive automation. This manual addresses configuration and deployment for:

- **CRUD Lambda API** (Python + DynamoDB)
- **Containerised ML Inference Lambda** (Docker/ECR)
- **API Gateway routing** for CRUD and inference endpoints
- **Custom CloudWatch metrics:** RequestsProcessed, ColdStartCount, InferenceLatency, ModelColdStartCount
- **Prometheus, Pushgateway, and Grafana** monitoring stack on EC2
- **Automated testing:** PyTest, Robot Framework, Locust
- **Adaptive QA Controller** for metrics-driven test triggers
- **GitHub Actions CI/CD pipeline** with automated deployment and re-testing

1.3 Objectives

The aims and objectives of this manual are to provide:

- A reproducible setup for functional, behavioural, and performance testing
- Clear and concise deployment steps for CRUD and ML serverless components
- Instructions for integrating CloudWatch with Prometheus and Grafana
- Instructions for setting up the Adaptive QA Controller for detecting anomalies
- Streamlined CI/CD workflow between testing, observability, and automated deployment

1.4 High-Level Architecture

The framework consists of five integrated layers:

1. **Serverless Compute:** CRUD and ML inference Lambda functions
2. **database:** DynamoDB for CRUD API's data; S3 for ML model(ML api) artefacts
3. **Observability:** CloudWatch metrics, Pushgateway publishing, Prometheus scraping, Grafana dashboards
4. **Testing:** Automated validation using PyTest, Robot Framework, and Locust
5. **Adaptive Intelligence:** Controller that analyses metrics and triggers targeted re-testing in CI/CD

2 System Overview

2.1 Core Architecture

The Unified QA Framework consists of two AWS Lambda workloads that comprises both a lightweight and compute-intensive execution paths.

2.1.1 QAFrameworkCRUD Lambda

A Python 3.11 Lambda providing CRUD operations backed by DynamoDB.

- Operations: Create, Read, Update, Delete
- Backend: DynamoDB table (`QAFrameworkDB`)
- Custom metrics: `RequestsProcessed`, `ColdStartCount`
- Deployment: ZIP package

2.1.2 QAFrameworkMLInferenceContainer Lambda

A container-based Lambda - ML deployed through ECR for real-time ML inference.

- Image: `ml-inference:latest`
- Model: XGBoost classifier (stored in S3)
- Endpoints: `/predict`, `/health`
- Metrics: `InferenceCount`, `InferenceLatency`, `ModelColdStartCount`

2.2 API Gateway

API Gateway exposes a unified HTTP interface:

- `/crud` — CRUD operations
- `/predict` — ML inference
- `/health` — ML readiness

This routing thus enables consistent functional and performance testing across both workloads.

2.3 Observability

Telemetry flows through a CloudWatch–Prometheus pipeline.

- CloudWatch collects metrics emitted by both Lambdas.
- CI/CD publishes metrics to Pushgateway on EC2 (`54.224.224.239:9091`).
- Prometheus scrapes Pushgateway at fixed intervals.
- Grafana visualises cold starts, throughput, latency, and Adaptive QA activity.

2.4 Testing Layer

Three frameworks provide functional, behavioural, and performance validation:

- **PyTest** — functional testing for CRUD and ML endpoints
- **Robot Framework** — workflow and behavioural testing
- **Locust** — load, concurrency, and performance testing

All tests execute sequentially inside GitHub Actions.

2.5 Adaptive QA Layer

The Adaptive QA Controller extracts and evaluates the runtime metrics to trigger targeted re-testing.

- Monitored metrics: `ColdStartCount`, `RequestsProcessed`, `InferenceLatency`, `ModelColdStartCount`

Triggered actions include:

- Locust performance runs
- Robot behavioural re-validation
- ML readiness warm-up checks

2.6 Technology Stack

Layer	**Technology**
Compute	AWS Lambda (Python 3.11, Docker/ECR)
Storage	DynamoDB, Amazon S3
API Routing	Amazon API Gateway
Observability	CloudWatch, Prometheus, Pushgateway, Grafana
CI/CD	GitHub Actions (OIDC authentication)
Testing	PyTest, Robot Framework, Locust
Machine Learning	XGBoost model (containerised Lambda)

3 Configuration Details

3.1 AWS Configuration

This section comprises the core AWS resources required to deploy and reproduce the Unified QA Framework.

3.1.1 QAFrameworkCRUD Lambda

Python 3.11 Lambda providing DynamoDB-backed CRUD operations.

- Handler: `app.lambda_handler`
- Memory: 256 MB Timeout: 10 s
- Deployment: ZIP archive
- Metrics: `RequestsProcessed`, `ColdStartCount`

3.1.2 DynamoDB Table

- Name: `QAFrameworkDB`
- Partition key: `id` (String)
- Billing mode: On-demand

3.1.3 QAFrameworkMLInferenceContainer Lambda

Container-based Lambda for real-time ML inference.

- Image: `ml-inference:latest` (ECR)
- Memory: 1024 MB Timeout: 30 s
- Ephemeral storage: 512 MB
- Model: XGBoost (stored in S3)
- Metrics: `InferenceCount`, `InferenceLatency`, `ModelColdStartCount`

3.2 API Gateway Configuration

API Gateway provides a unified public interface:

Endpoint	Method	Integrated Function
`/crud`	POST, GET, PUT, DELETE	QAFrameworkCRUD
`/predict`	POST	ML Inference Lambda
`/health`	GET	ML Inference Lambda

3.3 CloudWatch Metrics

Both Lambdas emit custom metrics used by the Adaptive QA Controller.

3.3.1 CRUD Metrics

- `RequestsProcessed` — successful CRUD operations
- `ColdStartCount` — cold starts

3.3.2 ML Metrics

- `InferenceCount` — inference requests
- `InferenceLatency` — end-to-end inference time
- `ModelColdStartCount` — container cold starts

3.4 Prometheus and Pushgateway

Prometheus scrapes custom QA metrics published to Pushgateway.

3.4.1 EC2 Prometheus Node

- Public IP: `54.224.224.239`
- Pushgateway: `http://54.224.224.239:9091`

3.4.2 Scrape Job

```
- job_name: 'pushgateway'
  honor_labels: true
  static_configs:
    - targets: ['54.224.224.239:9091']
```

3.5 CI/CD Configuration

GitHub Actions automates testing, metric handling, and deployment.

3.5.1 Pipeline Workflow

1. Run PyTest (functional tests)
2. Run Robot Framework (behavioural tests)
3. Run Locust (load tests)
4. Retrieve CloudWatch metrics
5. Push metrics to Pushgateway
6. Execute Adaptive QA Controller
7. Update Lambda code / container images
8. Promote alias `dev` → `live`

3.5.2 Required Environment Variables

Stored in GitHub Secrets:

- `PUSHGATEWAY_URL`
- `API_ENDPOINT_CRUD`
- `API_ENDPOINT_ML`
- `AWS_ROLE_TO_ASSUME`

3.6 Adaptive QA Controller

The Adaptive QA Controller evaluates metrics and triggers targeted re-testing.

3.6.1 Rule Engine

Metric	Condition	Action
`ColdStartCount`	¿ 2	Run Locust test
`RequestsProcessed`	¡ 1	Run Robot suite
`InferenceLatency`	¿ threshold	Trigger ML stress test
`ModelColdStartCount`	¿ 0	Warm ML model

3.6.2 Controller Output Metrics

- adaptive_actions_total
- adaptive_triggered_tests

3.7 Controller Structure and Configuration

3.7.1 Directory Structure

```
controller/
    adaptive_controller.py
    config/
        thresholds.json
        rules.json
    model/
        regression_model.pkl
    utils/
        cloudwatch_client.py
        prometheus_client.py
```

3.7.2 Installation

```
pip install boto3 requests pandas numpy scikit-learn \
            prometheus-client python-dateutil
```

3.7.3 Configuration Files

thresholds.json

```
{ "ColdStartCount": 2, "RequestsProcessed": 1,
  "InferenceLatency": 500, "ModelColdStartCount": 0 }
```

rules.json

```
{ "ColdStartCount": "locust_load_test",
  "RequestsProcessed": "robot_suite",
  "InferenceLatency": "ml_stress_test",
  "ModelColdStartCount": "warm_model" }
```

3.7.4 Execution Workflow

1. Load thresholds and rules
2. Retrieve latest CloudWatch metrics
3. Evaluate against thresholds
4. Trigger associated actions
5. Push controller metrics to Pushgateway

3.7.5 Local and CI/CD Execution

```
DRY_RUN=true python controller/adaptive_controller.py
```

```
- name: Run Adaptive QA Controller
  run: python controller/adaptive_controller.py
```

3.7.6 Optional: Model-Based Anomaly Detection

The controller may load `regression_model.pkl` to support adaptive thresholding.

3.8 Testing Frameworks

- **PyTest** — functional tests
- **Robot Framework** — behavioural tests
- **Locust** — load/performance tests

4 Installation Guide

This section comprises of the steps required to install and reproduce the Unified QA Framework, the local dev setup, AWS configuration, container image creation and observability tools.

4.1 System Requirements

- **Python 3.11** — Lambda development and testing
- **AWS CLI v2** — manage Lambda, DynamoDB, API Gateway, CloudWatch, ECR
- **Docker Engine** — build and test ML Lambda container
- **Git** — retrieve repository
- **AWS Account** with permissions for Lambda, DynamoDB, ECR, API Gateway, CloudWatch

Optional tools:

- AWS SAM CLI (template deployments)
- Node.js/npm (CDK-based workflows)

4.2 Installing Core Tools

4.2.1 Ubuntu / Linux

```
sudo apt update
sudo apt install -y python3 python3-venv python3-pip awscli docker.io
sudo systemctl enable --now docker
```

4.2.2 macOS (Homebrew)

```
brew install python awscli
brew install --cask docker
```

4.2.3 Windows

Install:

- Python 3.11
- AWS CLI MSI installer
- Docker Desktop (requires WSL2)

4.2.4 Verify installs

```
python3 --version
aws --version
docker --version
git --version
```

4.3 Python Dependencies

Dependencies are defined in `requirements.txt`, including:

```
boto3
pytest
robotframework
locust
prometheus-client
numpy
pandas
scikit-learn
xgboost
```

4.4 Local Installation Steps

1. Clone the repository:

   ```
   git clone <repository-url>
   cd <project-folder>
   ```

2. Create and activate virtual environment:

   ```
   python3 -m venv venv
   source venv/bin/activate          # macOS/Linux
   venv\Scripts\activate             # Windows
   ```

3. Install dependencies:

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

4. Confirm test tools:

```
pytest --version
robot --version
locust --version
```

5. Configure AWS CLI:

```
aws configure
```

6. Authenticate Docker to ECR:

```
aws ecr get-login-password --region <region> \
| docker login --username AWS --password-stdin \
  <aws-id>.dkr.ecr.<region>.amazonaws.com
```

7. Build the ML inference image:

```
docker build -t ml-inference .
```

8. (Optional) Test container locally:

```
docker run -p 8080:8080 ml-inference
curl http://localhost:8080/health
```

9. Run local unit tests:

```
pytest -v
```

4.5 Installing Prometheus, Pushgateway, and Grafana on EC2

4.5.1 Install Prometheus

```
wget https://github.com/prometheus/prometheus/releases/download/v2.49.1/ \
prometheus-2.49.1.linux-amd64.tar.gz
tar -xvf prometheus-*.tar.gz
sudo mv prometheus-*/ /usr/local/prometheus
```

4.5.2 Install Pushgateway

```
wget https://github.com/prometheus/pushgateway/releases/download/v1.10.0/ \
pushgateway-1.10.0.linux-amd64.tar.gz
tar -xvf pushgateway-*.tar.gz
sudo mv pushgateway-*/ /usr/local/pushgateway
```

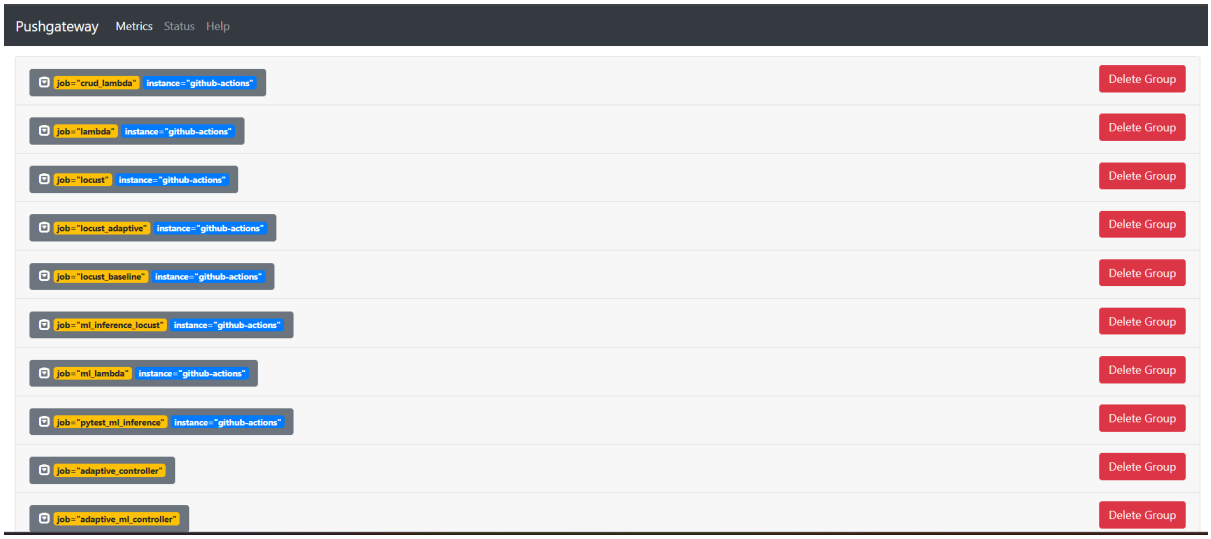

Figure 1: PushGateway metrics

4.5.3 Install Grafana

```
sudo apt-get install -y adduser libfontconfig1
wget https://dl.grafana.com/oss/release/grafana_10.2.3_amd64.deb
sudo dpkg -i grafana_*.deb
sudo systemctl enable --now grafana-server
```

4.5.4 Start services

```
cd /usr/local/prometheus
./prometheus &

cd /usr/local/pushgateway
./pushgateway --web.listen-address=":9091" &
```

4.6 Connectivity and Verification

- Prometheus metrics:

  ```
  curl http://<ec2-ip>:9090/metrics
  ```

- Pushgateway status:

  ```
  curl http://<ec2-ip>:9091/metrics
  ```

- Grafana UI:

  ```
  http://<ec2-ip>:3000
  ```

- AWS identity check:

  ```
  aws sts get-caller-identity
  ```

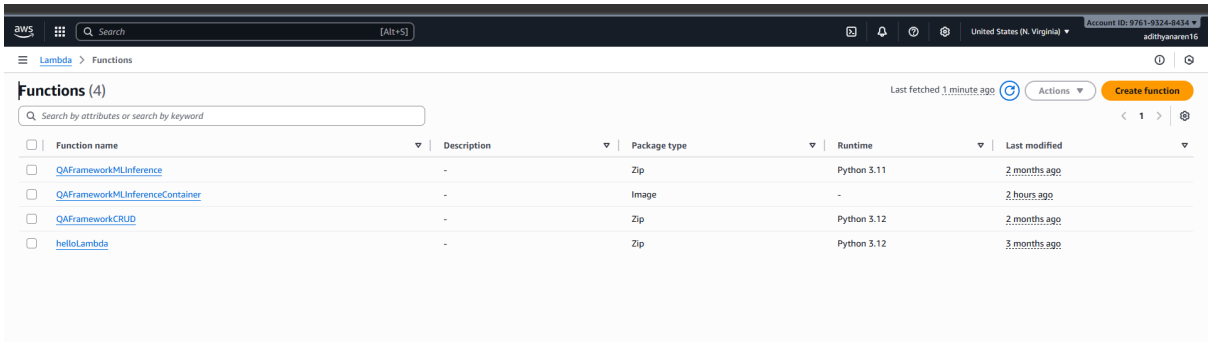

Figure 2: AWS Lambda Console - Deployed Functions for the Unified QA Framework

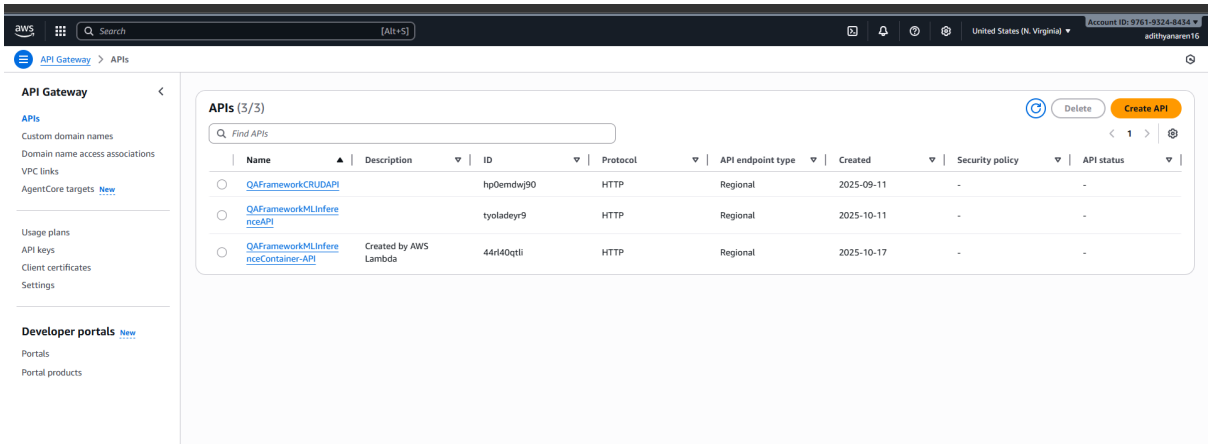

Figure 3: API Gateway Console - Configured APIs for CRUD and ML Inference Endpoints

4.7 Notes on System Setup

- Ensure that Docker is running before building/testing containers.
- Windows users require WSL2 for Docker.
- Ensure GitHub OIDC role is not blocked by MFA policies.

5 Deployment Guide

This section discusses the steps to deploy the CRUD Lambda, ML Inference Lambda, and API Gateway configuration.

5.1 Prerequisites

- Verify AWS authentication:

  ```
  aws sts get-caller-identity
  ```

- Lambda execution role with:
  - `AWSLambdaBasicExecutionRole`

  - CloudWatch metric publishing
  - DynamoDB access (CRUD)
  - S3 read access (ML)

- ECR repository:

```
aws ecr create-repository --repository-name ml-inference
```

5.2 Deploying the CRUD Lambda Function

1. Package

```
cd lambda_crud
pip install -r requirements.txt -t .
zip -r function.zip .
```

2. Create (first deployment)

```
aws lambda create-function \
  --function-name QAFrameworkCRUD \
  --runtime python3.11 \
  --handler app.lambda_handler \
  --zip-file fileb://function.zip \
  --role arn:aws:iam::<account-id>:role/<lambda-role>
```

3. Update (subsequent deployments)

```
aws lambda update-function-code \
  --function-name QAFrameworkCRUD \
  --zip-file fileb://function.zip
```

4. Validate

```
aws lambda invoke --function-name QAFrameworkCRUD response.json
cat response.json
```

5.3 Deploying the ML Inference Container Lambda

1. Authenticate to ECR

```
aws ecr get-login-password --region <region> \
| docker login --username AWS --password-stdin \
  <aws-id>.dkr.ecr.<region>.amazonaws.com
```

2. Build, tag, push

```
docker build -t ml-inference .
docker tag ml-inference:latest \
  <aws-id>.dkr.ecr.<region>.amazonaws.com/ml-inference:latest
docker push <aws-id>.dkr.ecr.<region>.amazonaws.com/ml-inference:latest
```

3. Create (first deployment)

```
aws lambda create-function \
  --function-name QAFrameworkMLInferenceContainer \
  --package-type Image \
  --code ImageUri=<aws-id>.dkr.ecr.<region>.amazonaws.com/ml-inference:latest \
  --role arn:aws:iam::<account-id>:role/<lambda-role> \
  --timeout 30 --memory-size 1024
```

4. Update (subsequent deployments)

```
aws lambda update-function-code \
  --function-name QAFrameworkMLInferenceContainer \
  --image-uri <aws-id>.dkr.ecr.<region>.amazonaws.com/ml-inference:latest
```

5. Validate

```
aws lambda invoke \
  --function-name QAFrameworkMLInferenceContainer response.json
cat response.json
```

5.4 Deploying API Gateway

1. Create API (if required)

```
aws apigateway create-rest-api --name UnifiedQAFrameworkAPI
```

2. Import OpenAPI specification

```
aws apigateway import-rest-api \
  --fail-on-warnings \
  --body file://openapi-spec.json
```

3. (If needed) Attach integrations

- `/crud` → `QAFrameworkCRUD`
- `/predict`, `/health` → `QAFrameworkMLInferenceContainer`

4. Deploy stage

```
aws apigateway create-deployment \
  --rest-api-id <api-id> \
  --stage-name prod
```

5.5 Verifying Deployment

1. CRUD API

```
curl -X POST https://<api>.execute-api.<region>.amazonaws.com/prod/crud \
  -H "Content-Type: application/json" \
  -d '{"id":"123","name":"test"}'
```

2. ML Inference API

```
curl -X POST https://<api>.execute-api.<region>.amazonaws.com/prod/predict \
  -H "Content-Type: application/json" \
  -d '{"age":55,"cholesterol":240,"thalach":150}'
```

3. Health

```
curl https://<api>.execute-api.<region>.amazonaws.com/prod/health
```

4. CloudWatch logs

```
aws logs tail /aws/lambda/QAFrameworkCRUD --follow
aws logs tail /aws/lambda/QAFrameworkMLInferenceContainer --follow
```

5. Metrics and configuration

```
aws cloudwatch list-metrics --namespace UnifiedQA
aws lambda get-function --function-name QAFrameworkCRUD
aws lambda get-function --function-name QAFrameworkMLInferenceContainer
```

6 Testing Execution

This section explains how to execute the automated test suites that are being used in the Unified QA Framework. These tests are used to validate functional behaviour, correctness of the workflow, and performance characteristics for both serverless APIs. Tests can be run locally or using GitHub Actions. Before test execution, set the API endpoints:

```
export API_ENDPOINT_CRUD="https://<api>.execute-api.<region>.amazonaws.com/prod/crud"
export API_ENDPOINT_ML="https://<api>.execute-api.<region>.amazonaws.com/prod/predict
```

6.1 Running PyTest (Functional Testing)

Run all tests

```
pytest tests/pytest
```

Run a specific file

```
pytest tests/pytest/test_crud.py
```

Verbose / CI output

```
pytest -vv --maxfail=1
pytest --junitxml=pytest-results.xml
```

Debugging

```
pytest --last-failed -vv
```

6.2 Running Robot Framework (Behavioural Testing)

Run all suites

```
robot tests/robot
```

Run a specific suite

```
robot tests/robot/crud_workflow.robot
```

Generated files

- `output.xml`
- `log.html`
- `report.html`

6.3 Running Locust (Performance and Load Testing)

Interactive mode

```
locust -f tests/locust/locustfile.py
# UI: http://localhost:8089
```

Headless mode (CI)

```
locust -f tests/locust/locustfile.py \
  --headless -u 50 -r 5 -t 2m \
  --host https://<api>.execute-api.<region>.amazonaws.com/prod
```

CSV output

```
locust ... --csv=locust_results
```

Optional warm-up

```
locust -f tests/locust/ml_warmup_locustfile.py \
  --headless -u 5 -r 1 -t 30s
```

6.4 Test Output Summary

- **PyTest:** console output, optional JUnit XML
- **Robot Framework:** `output.xml`, `log.html`, `report.html`
- **Locust:** web UI (manual), CSV files (CI)

6.5 Debugging and Verification

Check API accessibility

```
curl https://<api>.execute-api.<region>.amazonaws.com/prod/health
```

Inspect CloudWatch logs

```
aws logs tail /aws/lambda/QAFrameworkCRUD --follow
aws logs tail /aws/lambda/QAFrameworkMLInferenceContainer --follow
```

Verify CloudWatch metrics

```
aws cloudwatch list-metrics --namespace UnifiedQA
```

References

Amazon Web Services (2024a). *Amazon API Gateway Developer Guide.*
URL: *https://docs.aws.amazon.com/apigateway/latest/developerguide/*

Amazon Web Services (2024b). *Amazon CloudWatch User Guide.*
URL: *https://docs.aws.amazon.com/cloudwatch/*

Amazon Web Services (2024c). *Amazon DynamoDB Developer Guide.*
URL: *https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/*

Amazon Web Services (2024d). *Amazon ECR User Guide.*
URL: *https://docs.aws.amazon.com/AmazonECR/latest/userguide/*

Amazon Web Services (2024e). *AWS Lambda Developer Guide.*
URL: *https://docs.aws.amazon.com/lambda/latest/dg/welcome.html*

Amazon Web Services (2025a). Amazon api gateway documentation, `https://docs.aws.amazon.com/apigateway/`.

Amazon Web Services (2025b). Amazon cloudwatch metrics and monitoring guide, `https://docs.aws.amazon.com/cloudwatch/`.

Amazon Web Services (2025c). Amazon dynamodb developer guide, `https://docs.aws.amazon.com/dynamodb/`.

Amazon Web Services (2025d). Aws lambda documentation, `https://docs.aws.amazon.com/lambda/`.

GitHub (2024). *GitHub Actions: Configuring OpenID Connect in AWS.*
URL: *https://docs.github.com/actions/deployment/security-hardening-your-deployments/configuring-openid-connect-in-amazon-web-services*

GitHub (2025). Github actions documentation, `https://docs.github.com/actions`.

Grafana Labs (2024). *Grafana Documentation.*
URL: *https://grafana.com/docs/*

Grafana Labs (2025). Grafana observability platform documentation, `https://grafana.com/docs/`.

Locust Developers (2024). *Locust Load Testing Framework Documentation.*
URL: *https://docs.locust.io/*

Prometheus Authors (2024). *Prometheus Documentation*.
URL: *https://prometheus.io/docs/*

Prometheus Authors (2025a). Prometheus documentation, `https://prometheus.io/docs/`.

Prometheus Authors (2025b). Prometheus pushgateway documentation, `https://github.com/prometheus/pushgateway`.

pytest developers (2024). *pytest Documentation*.
URL: *https://docs.pytest.org/*

Robot Framework Foundation (2024). *Robot Framework User Guide*.
URL: *https://robotframework.org/*

XGBoost Developers (2024). *XGBoost Documentation*.
URL: *https://xgboost.readthedocs.io/*