National College of Ireland

ML-Powered Cloud Task Failure Prediction and Scalable Deployment on AWS

MSc Research Project

MSc in Cloud Computing

Harikrishnan Satheeshkumar

Student ID: x23297948

School of Computing

National College of Ireland

Supervisor: Aqeel Kazmi

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	……. Harikrishnan Satheeshkumar……
Student ID:	……x23297948……..……
Programme:	……MSc in Cloud Computing…… **Year:** …2024…….
Module:	……Research Project……..……
Supervisor:	……Aqeel Kazmi…..……
Submission Due Date:	……15-09-2025……..……
Project Title:	ML-Powered Cloud Task Failure Prediction and Scalable Deployment on AWS
Word Count:	……802…… **Page Count**……3……..……..

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: ……Harikrishnan Satheeshkumar……

Date: ……15-09-2025……..……..……

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

---

1. Introduction

This configuration manual provides the detailed steps required to set up the environment and deploy the **ML-Powered Cloud Task Failure Prediction and Scalable Deployment on AWS** project. The guide covers the necessary prerequisites, software installation, model training, and the complete deployment pipeline on Amazon Web Services (AWS). It is designed for developers, researchers, and students who wish to reproduce the experiment, validate the results, or extend the project's functionality. The final output is a scalable, real-time API for predicting task failures, orchestrated with modern cloud-native tools.

---

2. Prerequisites

Before beginning the setup process, please ensure your system meets the following requirements:

- **AWS Account**: An active Amazon Web Services account with permissions to manage IAM, S3, ECR, and ECS is essential.
- **Operating System**: A Linux-based OS (like Ubuntu 20.04+), macOS, or Windows with WSL2 is recommended for compatibility with Docker and shell commands.
- **Python Version**: Python 3.10 or later.
- **Docker**: Docker Desktop or Docker Engine installed and running on your local machine.
- **Git**: Git installed for cloning the project repository.
- **AWS CLI**: The AWS Command Line Interface configured with credentials for your account.

---

3. Libraries and Packages

The project relies on several key Python libraries for data processing, machine learning, and serving the API. The core dependencies are listed in the requirements.txt file.

- `pandas`: For data manipulation and reading the CSV dataset.
- `scikit-learn`: Used for the Random Forest Classifier model and data splitting.
- `numpy`: For numerical operations.
- `boto3`: The AWS SDK for Python, used by the Flask app to dynamically load the trained model from an S3 bucket.
- `Flask`: A micro web framework for building the `/predict` API endpoint.
- `gunicorn`: A robust WSGI HTTP Server to run the Flask application in the Docker container.

---

4. Configuration Steps

Follow these steps sequentially to set up, train, and deploy the entire system.

Step 1: Data Preparation

The model is trained on the Google Borg workload traces.

1. Download the dataset (`borg_traces_data.csv`).
2. Place the downloaded CSV file into a `data/` directory within the project's root folder.

Step 2: Train the Machine Learning Model

Run the training script to process the data and create the model artifact. This script will perform data cleaning, feature engineering, and train the Random Forest classifier.

```
# Run the training script
python train.py
```

This will generate the trained model file, `rf_classifier.joblib`, in the project's root directory.

Step 3: AWS S3 and IAM Setup

The trained model needs to be stored in S3 for the application to access it.

1. Create a new S3 bucket in your AWS account.
2. Upload the `rf_classifier.joblib` file to the S3 bucket you just created.
3. Create an IAM role that grants ECS tasks read-only access to this specific S3 bucket. Note the ARN of this role for later use.

Step 4: Build and Push Docker Image to ECR

Containerize the Flask application using Docker and push it to the Amazon Elastic Container Registry (ECR).

```
# Build the Docker image
docker build -t cloud-failure-predictor .

# Authenticate Docker to your AWS ECR registry
aws ecr get-login-password --region <your-region> | docker login --username
AWS --password-stdin <your-aws-account-id>.dkr.ecr.<your-
region>.amazonaws.com

# Create an ECR repository if you haven't already
aws ecr create-repository --repository-name cloud-failure-predictor

# Tag the image for ECR
```

```
docker tag cloud-failure-predictor:latest <your-aws-account-
id>.dkr.ecr.<your-region>.amazonaws.com/cloud-failure-predictor:latest

# Push the image to ECR
docker push <your-aws-account-id>.dkr.ecr.<your-region>.amazonaws.com/cloud-
failure-predictor:latest
```

Step 5: Deploy to AWS ECS with an ALB

Deploy the container as a service on ECS using the Fargate launch type for a serverless experience.

1. **Create an ECS Task Definition**: Define the task by pointing it to the ECR image URL from the previous step. Assign the IAM role created in Step 4 to this task definition.
2. **Create an Application Load Balancer (ALB)**: Set up an ALB to expose the service to the internet. Configure a target group that forwards traffic to port 5000 of the container.
3. **Launch the ECS Service**: Create a new service in your ECS cluster, select the task definition, and associate it with the ALB's target group.

Step 6: Configure Auto Scaling

Attach an Auto Scaling policy to the ECS service to handle variable loads.

1. Navigate to your newly created ECS service.
2. Under the "Auto Scaling" tab, create a policy.
3. Configure the policy to scale the number of tasks based on the average CPU utilization, for example, scale out if CPU exceeds 75%.

Step 7: Live Testing

Once the service is running, you can test the live prediction endpoint using `curl`.

1. Find the DNS name of your Application Load Balancer from the EC2 console.
2. Send a POST request with feature data to the `/predict` endpoint.

```
curl -X POST http://<YOUR-ALB-DNS-NAME>/predict \
-H "Content-Type: application/json" \
-d '{"features": [274800000000, 0.0, 1.0, 0.000415, 0.0, 0.0]}'
```

A successful response will be a JSON object containing the prediction, such as `{"prediction": 0}`.