National College of Ireland

Configuration Manual

MSc Research Project
MSc Cloud Computing

Pradnya Deshmukh
Student ID: X23149604

School of Computing
National College of Ireland

Supervisor: Shivani Jaswal

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: ...Pradnya Deshmukh

Student ID: ...X23149604

Programme: .....MSc Cloud Computing..... **Year:** .....2024.....

Module: Research Project

Lecturer: ...Shivani Jaswal...

Submission Due Date: 12-12-2024

Project Title: Adaptive Serverless FaaS Dataflow Framework for Real-Time IoT Analytics across the Cloud-Edge Continuum

Word Count: .....**PageCount:** .....11.....

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: .....Pradnya Deshmukh...

Date: 12-12-2024

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

Pradnya Deshmukh
X23149604

1 Introduction

This configuration guide will give a hands-on guide to setup and run the services and code on AWS. The system architecture consists of the following key components:

- Data Generation and Ingestion: The sensor data from IoT devices are stored in S3 storage for further processing.
- Machine Learning Pipeline: This research uses three models: anomaly detector for predicting anomalies in the preprocessed data, LSTM for resource allocation decisions, and RL Orchestrator for making placement decisions on the cloud-edge continuum.
- Real-time Inference: Model inference by AWS Lambda functions with API Gateway.

2 Prerequisites

Hardware Configuration for the local run:

- Processor: Intel 11th Gen Core i5-1135 @2.4 GHz
- RAM: 16 GB DDR4 RAM 3200MHz
- Storage (SSD): 512GB
- Operating System: Windows 11, 64-bit

Software Packages for the local run:

- Python 3.8+
- PyCharm IDE Community Edition 2021.3
- Jupyter Lab

3 AWS Account Requirements

- AWS Account Requirements
  An active AWS account with appropriate permissions
- Access to the following services:
  a. AWS Lambda
  b. Amazon S3
  c. Amazon SageMaker
  d. AWS API Gateway

4 IAM Access Roles and Permissions

4.1 Lambda Basic Execution Role

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": "logs:CreateLogGroup",
         "Resource": "arn:aws:logs:us-east-1:767397664235:*"
      },
      {
         "Effect": "Allow",
         "Action": [
            "logs:CreateLogStream",
            "logs:PutLogEvents"
         ],
         "Resource": [
            "arn:aws:logs:us-east-1:767397664235:log-
group:/aws/lambda/iot_orchestrator_lambda:*"
         ]
      }
   ]
}
```

4.2 Lambda IoT Thing Access Policy

```
{
        "Version": "2012-10-17",
        "Statement": [
                {
                        "Effect": "Allow",
                        "Action": [
                                "iot:GetThingShadow",
                                "iot:UpdateThingShadow"
                        ],
                        "Resource": "arn:aws:iot:us-east-1:767397664235:thing/*"
                }
        ]
}
```

4.3 Sagemaker S3 Access Policy

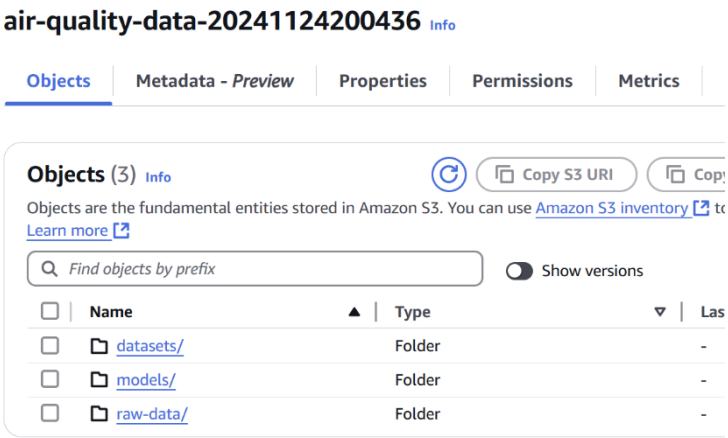
```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
```

```
            "Action": [
                "s3:GetObject",
                "s3:PutObject",
                "s3:DeleteObject",
                "s3:ListBucket"
            ],
            "Resource": [
                "arn:aws:s3:::*"
            ]
        }
    ]
}
```

4.4 Sagemaker IoT Shadow Access Policy

```
{
        "Version": "2012-10-17",
        "Statement": [
                {
                        "Effect": "Allow",
                        "Action": [
                                "iot:GetThingShadow",
                                "iot:UpdateThingShadow",
                                "iot:ListThings"
                        ],
                        "Resource": "*"
                }
        ]
}
```

5 S3 Storage Bucket



6 Model Development

1. Create SageMaker Domain for code development

2. Prepare Training Data - Create an S3 bucket to store training data.
3. Upload historical sensor data to this bucket in json format
4. Create SageMaker Notebook Instance
5. Click on JupyterLab to create your project spaces, air_quality_ml
6. Create sub-folders for the project
7. Code and train your models.

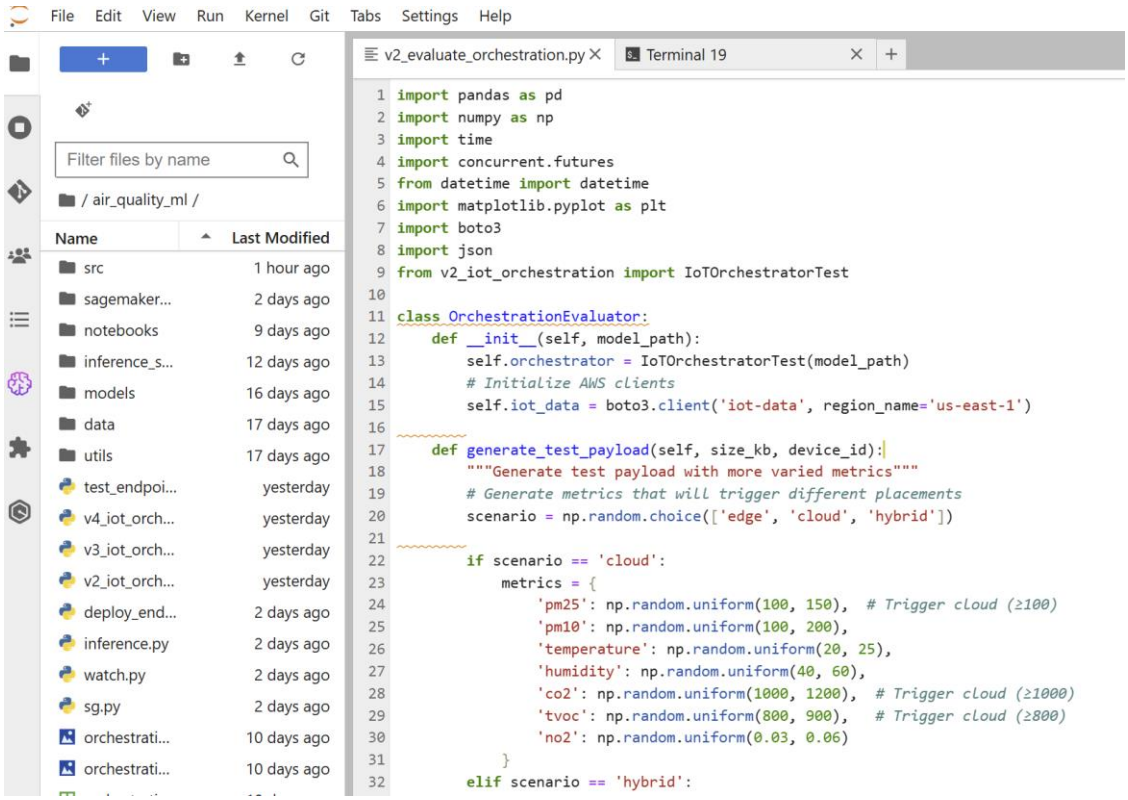


Figure: Model Training and Deployment Window – Jupyterlab

7 Code Development

7.1 **Training Data Generator –** To generate training_data.csv file for model training

```
def main():
  """Main function to prepare training data"""
  try:
    # Initialize preparator
    preparator = TrainingDataPreparator()

    # Prepare data
    input_path = 's3://air-quality-data-
20241124200436/datasets/historical/20241124_143440.csv'
    output_path = preparator.prepare_data(input_path)
```

```
        print(f"\nData preparation completed successfully!")
        print(f"Training data saved to: {output_path}")
        print("Check data_preparation.log for detailed information.")

    except Exception as e:
        print(f"Error: {str(e)}")
        raise

if __name__ == "__main__":
    main()
```

7.2 Anomaly Detection Script – Trains on training_data.csv and saves a model that can detect anomalies in the sensor data.

```
class AnomalyDetectionTrainer:
        def __init__(self, input_size):
            self.model = AnomalyDetector(input_size)
            self.criterion = nn.MSELoss()
            self.optimizer = optim.Adam(self.model.parameters(), lr=0.001)
            self.scaler = StandardScaler()  # Will use own scaler instead of
    preprocessor

        def preprocess_data(self, data):
            """Preprocess input data using internal scaler"""
            return self.scaler.fit_transform(data)

        def train(self, data, epochs=200, batch_size=32):
            """Train the anomaly detector"""
            data_tensor = torch.FloatTensor(data)

            print(f"Training on data shape: {data_tensor.shape}")
            losses = []

            for epoch in range(epochs):
                self.optimizer.zero_grad()
                outputs = self.model(data_tensor)
                loss = self.criterion(outputs, data_tensor)
                loss.backward()
                self.optimizer.step()

                losses.append(loss.item())

                if (epoch + 1) % 10 == 0:
                    print(f'Epoch [{epoch+1}/{epochs}], Loss: {loss.item():.4f}')

            return losses
```

7.3 LSTM Script – Trains on training_data.csv and saves a model that can make resource allocation decisions on the IoT sensor data

```
class LSTMPredictor:
  def __init__(self, input_size, sequence_length=24):
    self.sequence_length = sequence_length
    self.scaler = MinMaxScaler()
    self.model = self._build_model(input_size)

  def _build_model(self, input_size):
    """Build LSTM model"""
    model = tf.keras.Sequential([
      tf.keras.layers.LSTM(128, return_sequences=True,
                  input_shape=(self.sequence_length, input_size)),
      tf.keras.layers.Dropout(0.2),
      tf.keras.layers.LSTM(64),
      tf.keras.layers.Dropout(0.2),
      tf.keras.layers.Dense(input_size)
    ])

    model.compile(optimizer='adam',
          loss='mse',
          metrics=['mae'])
    return model
```

7.4 RL Orchestration Script – Trains on training_data.csv and saves a model that can make placement decisions (hybrid, edge, cloud) on the IoT sensor data

```
class OrchestratorModel:
  def __init__(self, state_size, action_size):
    self.state_size = state_size
    self.action_size = action_size
    self.memory = deque(maxlen=500) #2000
    self.gamma = 0.95    # discount rate
    self.epsilon = 1.0   # exploration rate
    self.epsilon_min = 0.01
    self.epsilon_decay = 0.995
    self.model = self._build_model()

  def _build_model(self):
    model = Sequential([
      Dense(64, input_dim=self.state_size, activation='relu'),
      Dropout(0.2),
      Dense(32, activation='relu'),
      Dropout(0.2),
```

```
        Dense(self.action_size, activation='linear')
    ])
    model.compile(optimizer='adam', loss='mse')
    return model
```

7.5 **RL Orchestration Evaluation** – Trains on the saved rl_model.keras and makes predictions on the sensor data for placement decisions

```
class OrchestrationEvaluator:
  def __init__(self, model_path):
    self.orchestrator = IoTOrchestratorTest(model_path)
    # Initialize AWS clients
    self.iot_data = boto3.client('iot-data', region_name='us-east-1')


def main():
  # Initialize evaluator
  evaluator = OrchestrationEvaluator('/home/sagemaker-
user/air_quality_ml/models/rl/rl_model.keras')

  # Run evaluation
  device_id = "device001"
  results_df = evaluator.run_evaluation(device_id)

  # Save results
  results_df.to_csv('orchestration_evaluation_results.csv', index=False)

  # Create plots
  evaluator.plot_results(results_df)
```

7.6 RL Orchestration Output

```
Starting evaluation...
Testing payload size: 100KB, Concurrent requests: 1
Updated shadow with scenario: hybrid, metrics: {'pm25': 50.119708641081004, 'pm10':
51.03893129252317, 'temperature': 31.36526683763491, 'humidity': 71.22804361234374,
'co2': 781.5271475748289, 'tvoc': 404.39807784816736, 'no2': 0.029951161594292133}
Updated shadow with scenario: hybrid, metrics: {'pm25': 56.34440510845337, 'pm10':
97.65387500885791, 'temperature': 31.236259346134343, 'humidity': 77.32514835259808,
'co2': 698.2191675521649, 'tvoc': 542.9192837242713, 'no2': 0.038127718906702364}
Updated shadow with scenario: edge, metrics: {'pm25': 38.4545210188514, 'pm10':
31.30901716250246, 'temperature': 19.33260152193791, 'humidity': 56.111164615087354,
'co2': 361.92882107279047, 'tvoc': 239.5090947129933, 'no2': 0.010457358256759957}
Updated shadow with scenario: hybrid, metrics: {'pm25': 59.97349934543843, 'pm10':
143.69838630585076, 'temperature': 34.04850269324211, 'humidity': 74.06926622158008,
'co2': 611.5978524939853, 'tvoc': 629.3157454763943, 'no2': 0.04738624696713416}
```

Updated shadow with scenario: edge, metrics: {'pm25': 1.050173821254948, 'pm10': 32.08939830220883, 'temperature': 21.317065910733685, 'humidity': 34.97460872675319, 'co2': 458.6490004664163, 'tvoc': 232.53184254754018, 'no2': 0.016166110430412998}
Testing payload size: 100KB, Concurrent requests: 20
Updated shadow with scenario: edge, metrics: {'pm25': 29.880896644472724, 'pm10': 63.5453272389975, 'temperature': 22.158943545480003, 'humidity': 37.85692417866186, 'co2': 756.9782501595016, 'tvoc': 344.761594995, 'no2': 0.01427305051125045}
Updated shadow with scenario: hybrid, metrics: {'pm25': 74.65258187324008, 'pm10': 137.0010640980583, 'temperature': 33.74835522506439, 'humidity': 74.21421499765079, 'co2': 509.1682350624015, 'tvoc': 569.3053298188457, 'no2': 0.036769914104435385}
Updated shadow with scenario: edge, metrics: {'pm25': 29.051218737272123, 'pm10': 73.67101139763976, 'temperature': 15.926044391246956, 'humidity': 35.947662527130454, 'co2': 545.3734821162675, 'tvoc': 375.5030524685509, 'no2': 0.021578345024267524}
Updated shadow with scenario: edge, metrics: {'pm25': 30.568450140146222, 'pm10': 66.55394201900556, 'temperature': 21.434158822778475, 'humidity': 50.81528902071889, 'co2': 349.5362329494122, 'tvoc': 428.1888281872783, 'no2': 0.019802093540241632}

7.7 IoT Orchestration Lambda Execution

```
Function Logs:
START RequestId: 26397a17-7d67-49ab-84fb-4e405c1d5cb8 Version: $LATEST
device001
{'confidence': 0.9, 'device_id': 'device001', 'metrics': {'co2':
733.1878644881135, 'humidity': 75.14541691906045, 'no2': 0.0332934271020415,
'pm10': 90.8683335573566, 'pm25': 84.9049902607766, 'temperature':
30.052947726142225, 'tvoc': 572.9200736205215}, 'placement': 'cloud',
'processing_rules': {'compressionLevel': 'low', 'dataAggregation': '5s',
'filterThreshold': 0.3, 'localProcessing': False}, 'resource_allocation':
{'batch_size': 64, 'cpu': 0.3, 'memory': 0.3}, 'timestamp': '2024-12-
11T21:57:06.000069'}
formatted_response
{'device_info': {'device_id': 'device001', 'timestamp': '2024-12-11 21:57:06
UTC'}, 'metrics': {'air_quality': {'pm2.5': '84.9 µg/m³', 'pm10': '90.9
µg/m³', 'co2': '733 ppm', 'tvoc': '573 ppb', 'no2': '0.033 ppm'},
'environment': {'temperature': '30.1°C', 'humidity': '75.1%'}}, 'decision':
{'location': 'cloud', 'confidence': '90.0%', 'description': 'Process data in
the cloud for maximum computational power'}, 'resource_allocation':
{'cpu_usage': '30.0%', 'memory_usage': '30.0%', 'batch_size': 64},
'processing_config': {'local_processing': False, 'data_aggregation_interval':
'5s', 'filter_threshold': '30%', 'compression_level': 'low'},
'status_summary': {'air_quality_status': 'alert', 'processing_load': 'high'}}
END RequestId: 26397a17-7d67-49ab-84fb-4e405c1d5cb8
REPORT RequestId: 26397a17-7d67-49ab-84fb-4e405c1d5cb8  Duration: 154.10 ms
Billed Duration: 155 ms Memory Size: 1024 MB    Max Memory Used: 47 MB

Request ID: 26397a17-7d67-49ab-84fb-4e405c1d5cb8
```

References

1. https://docs.aws.amazon.com/
2. https://marketsplash.com/