# CSC8112 - Internet of Things

Assessment 1

## Assessment Overview

This assessment contributes 40% towards the total mark for this module. Out of this, 70% marks are assigned for implementation tasks and 30% marks are for the final report. It is an individual exercise: no group work is permitted for the assessment. You are advised to read and view all the instructional tutorial resources before you start implementing the solutions for the coursework Tasks 1 to 3. Each Task has been assigned a specific mark, which you will be awarded once you successfully demonstrate the completion of the same.

Once you complete Tasks 1-3, you will need to prepare the final Report (Task 4). **This coursework Report must be submitted on NESS by 3 pm on November 14, 2025**. In this Report, you will need to provide an in-depth discussion of how you implemented the solutions (e.g., code and commands) to solve Tasks 1-4. Additionally, you will be required to demonstrate successful executions of Tasks 1-4. Before the report submission deadline, you will be provided with a 30 mins slot to conduct **live demonstration**. In case of unforeseen disruptions (e.g., further lockdowns), we may also allow recorded demonstrations.

**Final marks** for this coursework will be decided by your performance in the live demonstrations and the technical details you will provide in the final report. While the final report needs to be submitted to NESS by 3 pm on November 14, 2025, the live demonstration session will be organised on November 10, 2025.

**You are required to complete Tasks 1-3 using the command line interface ([https://docs.docker.com/engine/reference/commandline/cli/]), provided by the Docker Engine, as well as by implementing a programmatic solution in Python language.**

## Objectives

The learning outcomes of this coursework include the following:

- Understand how to process Internet of Things (IoT) sensor data in the edge-cloud setting?
- Be able to develop a machine learning-based IoT data processing pipeline ( data collection, data preprocessing, prediction and visualisation) in the edge-cloud setting?
- Be able to use a lightweight virtualisation technology stack, such as Docker, to implement IoT data processing pipeline in the edge-cloud setting?

A high-level picture showing the overall system design scope of the coursework is shown in Figure 1, a short explanation of the components is given below:

**IoT tier**:

- *Newcastle Urban Observatory (NCL UO) [https://urbanobservatory.ac.uk/]*: The largest set of publicly available real-time urban data in the UK. NCL UO sensors are gathering data across Newcastle city. With over 50 data types and counting, there are lots of live data for you to access.

**Edge tier:**

- *Data Injector*: This will be a software component that you will design and implement in Task 1, focusing on (i) reading data from Urban Observatory API and (ii) transmitting data to the machine learning pipeline.

- *EMQX*: A broker of MQTT protocol, a message queuing system given to you as a Docker image, which forms the basis for enabling asynchronous service-to-service communication in a complex Machine Learning (ML)-based IoT data processing pipeline.

- *Data Preprocessing Operator*: A software component that you will develop in Task 2, Responsible for preparing training data of Machine Learning model.

**Cloud tier:**

- *RabbitMQ*: A cloud-based message queuing system based on the AMQP protocol.

- *Machine Learning Model/Classifier/Engine*: A software component that can be trained to predict particular types of future events.

- *Visualization*: A component that will visualize the trend of raw time-series data and the prediction results (input from the Machine Learning Model/Classifier/Engine).

After successfully completing the coursework, you will be able to gain hands-on experience in the following interrelated technology stacks, including:

- configuring a Docker-based IoT data processing pipeline;
- pulling images from the Docker Hub (a global repository of software components' images maintained by the developers);
- creating and deploying a self-contained IoT data processing service, which is often referred to as microservices;

- training a machine-learning-based predictor based on real-world data streams available from Newcastle Urban Observatory;
- implementing a machine learning-based air-quality prediction micro-service;
- visualizing time-series data using graphs.

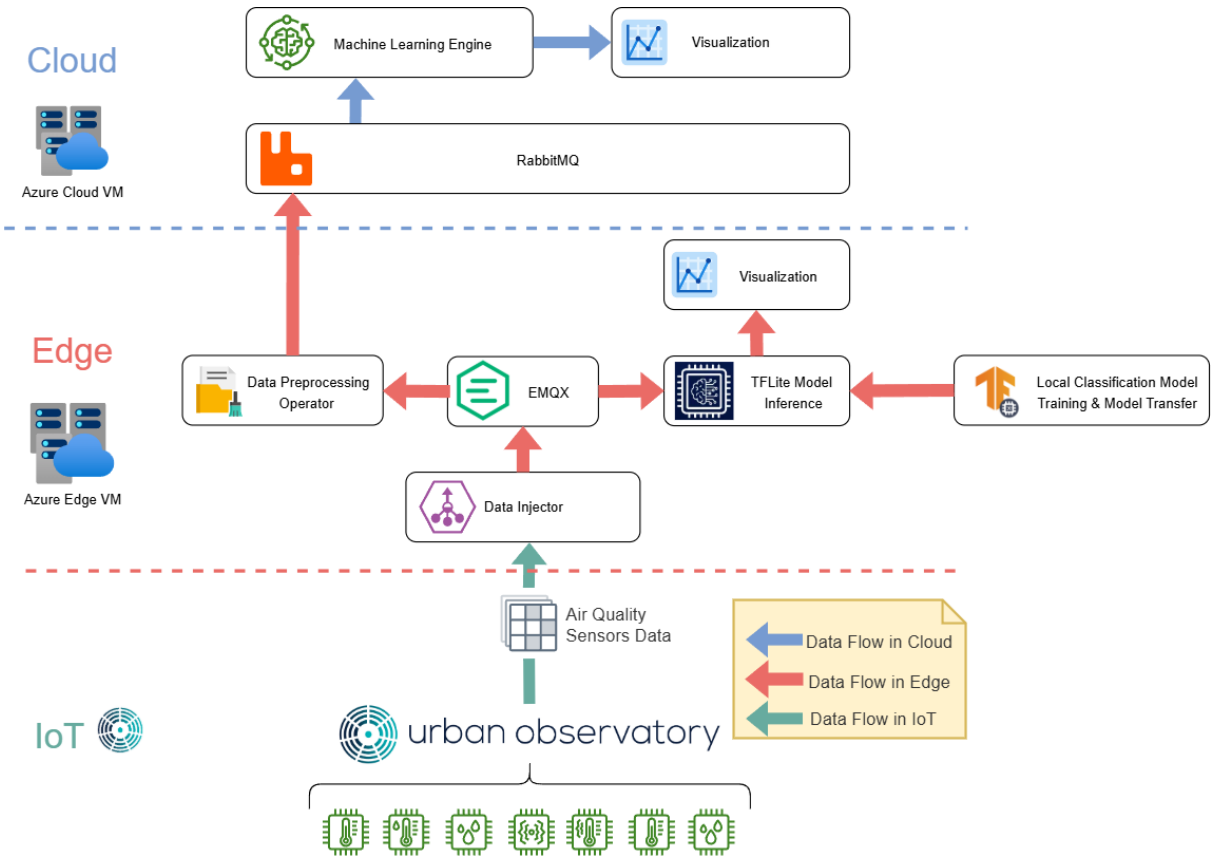


Figure 1: System overview

## Pre-Requisites

Before starting the coursework, you are advised to carefully go through the training content covered in Lecture 1 and extra supplements provided at [https://github.com/ncl-iot-team/CSC8112]. Together, these provide in-depth detail on:

- how to access and start Azure VMs, as shown in Figure 2;
- how to download and run a docker image on Azure Labs;
- how to run your experiments on Azure Labs;
- some hints for system structures of every task.

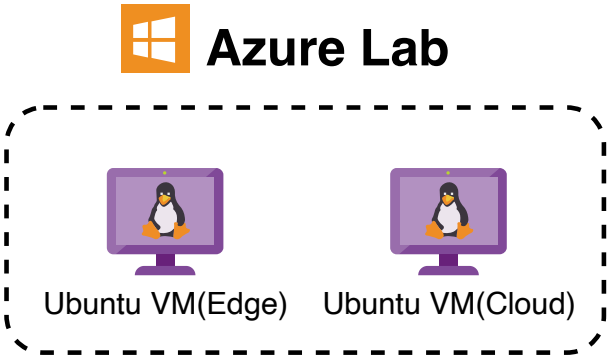


Figure 2: Relationship structure of Azure Lab and Ubuntu VMs.

## Specification of Tasks

The coursework consists of 4 tasks. **Please note that Tasks 1-4 need to be done by both the command line and implementing the logic using Python language.**

**Task 1: Design a data injector component by leveraging Newcastle Urban Observatory IoT data streams (20 Marks)**

**Task Objectives:** Understand and learn how to pull and run a Docker image from Docker Hub using the command line interface, how to collect real-world IoT data streams by invoking the Application Programming Interface (API) of Newcastle Urban Observatory, how to send data via MQTT (With EMQX as the broker for IoT applications), and how to re-compile and build Docker image using the command line and programmatic interfaces.

**Hints:** To download the EMQX docker image, please go to the following link [https://hub.docker.com/r/emqx/emqx]. To install python dependency package "requests" for sending HTTP request, please use the following Python package. [https://pypi.org/project/requests/]. Finally, the Pyton MQTT SDK ["paho.mqtt"] is available from [https://pypi.org/project/paho-mqtt/].

1. Pull and run the Docker image "emqx/emqx" from Docker Hub in the virtual machine running on Azure lab (Edge). Perform this task first using the command line interface (CLI).
2. Develop a data injector component with the following functions (Code) in Azure Lab (Edge) or the Azure Lab localhost:
   (a) Collect data from Urban Observatory platform by sending HTTP request to the following url ([https://github.com/ncl-iot-team/CSC8112/raw/refs/heads/main/data/uo_data.min.json]). Following that, please print out the raw data streams that you collected on the console.
   (b) Although the raw air quality data you collected from the Urban Observatory API contains many metrics including $NO_2$, NO, $CO_2$, PM2.5, and PM10, among others,

for the purpose of this coursework you only need to store and analyze PM2.5 data. While many meta-data are available for PM2.5 data, such as sensor name, timestamp, value, and location, you only need to store the metrics related to the Timestamp and Value meta-data fields.

(c) Send all PM2.5 data to be used by Task 2.2 (a) and Task 4.5 (b) via MQTT of Azure lab (Edge).

(d) Package the data injector as a Docker image. Each run on the Docker image should trigger the send operation once.

**Task 2: Data preprocessing operator design** **(20 Marks)**

**Task Objectives:** Understand how to clean and prepare data for machine learning training by applying data processing operations, such as outliers cleaning and data reformatting. Moreover, you will also learn how to collect/send data using various message queuing protocols (e.g., MQTT and AMQP), which are central to IoT data stream management. This task will also help you understand how native *Docker Compose techniques* can be leveraged to manage and deploy a complex IoT application stack/pipeline.

**Hints:** To install a python dependency package for sending messages via AMQP (a message queue protocol that the RabbitMQ broker uses), please download "pika" from [https://pypi.org/project/pika/].

1. Define a *Docker compose file* which contains the following necessary configurations and instructions for deploying and instantiating the following set of Docker images (as shown in Figure 1) on Azure lab (Cloud):

   (a) Download and run RabbitMQ image (rabbitmq:management);

2. Design a data preprocessing operator with the following functions (code) in *Azure Lab (Edge)*:

   (a) Collect all PM2.5 data published by Task 1.2 (c) from EMQX service, and please print out the PM2.5 data to the console (this operator will run as a Docker container, so the logs can be seen in the docker logs console automatically).

   (b) Filter out outliers (the value greater than 50), and please print out outliers to the console (docker logs console).

   (c) Average the PM2.5 data over 24-hour periods. Ensure that each daily average aligns correctly with the start and end of the day based on the Unix timestamp, and print the results to the console (docker logs console).

   (d) Transfer all results (averaged PM2.5 data) to be used by Task 3.2 (a) via AMQP to *Azure lab (Cloud).*

3. Define a Dockerfile to migrate your "data preprocessing operator" source code into a Docker image and then define a *docker-compose file* to run it as a container locally on the Azure lab (Edge).

**Task 3: Time-series data prediction and visualization** **(20 Marks)**

**Task Objectives:** Understand how to use a machine learning model/classifier with time-series sensor data, that you prepared in Task 2, to make a prediction, and how to visualize those data and predicted results.

**Hints:** To install the python dependency package "matplotlib" (a data visualization tool) use the following library including [https://pypi.org/project/matplotlib/]. To download the package "prophet" (a machine learning tool) use the following link: [https://pypi.org/project/prophet/].

1. Download a pre-defined Machine Learning (ML) engine code from [https://github.com/ncl-iot-team/CSC8112_MLEngine].
2. Design a PM2.5 prediction operator with the following functions (code) in Azure Lab (Cloud) or the Azure Lab localhost:
   (a) Collect all averaged daily PM2.5 data computed by Task 2.2 (d) from RabbitMQ service, and please print out them to the console.
   (b) Convert timestamp to date time format ( year-month-day hour:minute:second), and please print out the PM2.5 data with the reformatted timestamp to the console.
   (c) Use the line chart component of matplotlib to visualize averaged PM2.5 daily data, directly display the figure or save it as a file.
   (d) Feed averaged PM2.5 data to machine learning model to predict the trend of PM2.5 for the next 15 days (this predicted time period is a default setting of provided machine learning predictor/classifier model).
   (e) Visualize predicted results from ML predictor/classifier model, directly display the figure or save as it a file (pre-defined in the provided Machine Learning code).

**Task 4: Local Model Training and Edge Inference for Air Quality Classification (20 Marks)**

**Task Objectives:** Understand how to train a classification model locally, convert it to TensorFlow Lite for edge deployment, transfer the optimized model to edge infrastructure, and implement air quality classification using historical IoT data.

**Hints:** You can use the keras sequential package for the classification model. No need for specific models. Just explain your decisions in the report as we're more concerned on the task completion and not on model and inference quality. Distribution of PM2.5 values over the classes on IoT Sensor data from Urban Observatory is 7927 for Green, 772 for Yellow, and 12 for Red class, with total count being 8711. Here's the solution overview - https://github.com/ncl-iot-team/CSC8112/blob/main/img/task4.png

1. Use the provided labeled air quality dataset with PM2.5 values [https://github.com/ncl-iot-team/CSC8112/blob/main/data/PM2.5_labelled_data.csv], and train a classification model locally in your machine or google colab or kaggle environment with cpu/gpu(optional) using TensorFlow into corresponding 3 classifications as Red, Yellow, and Green. You can perform label-encoding, normalization, sampling to the labeled data if needed to improve model quality.

2. Convert the trained model to TensorFlow Lite format (.tflite), apply quantization to reduce model size and save it.

3. Evaluate model performance (accuracy, precision, recall, confusion matrix) and visualize the model size comparison graph(original vs TFLite optimized).

4. Transfer the .tflite model and any other necessary file(if needed) to Azure Lab (Edge VM) using copy-paste or file upload.

5. Create a Python script on Edge VM that:

   (a) Loads the TensorFlow Lite model.

   (b) Subscribe to the MQTT broker

   (c) Perform batch-inference on the json data from the MQTT broker.

   (d) Classifies each PM2.5 reading as Green/Yellow/Red using .tflite model file.

   (e) Logs classification results with the PM2.5 value to console.

   (f) Visualize results with a bar-chart showing count of Green/Yellow/Red classifications, a Time-series plot of PM2.5 values with color-coded classification points, PM2.5 value frequency distribution, and a summary of classification.

**Task 5: Report** **(20 Marks)**

Prepare the Final Report in plain English. There is no word or page limit; however, we appreciate a clear, concise, and focused presentation style. The report should consist of:

1. Detailed response to each task and related sub-tasks.

2. Screenshots of running services in the Docker Environment.

3. Screenshots of Code Snippets and/or Docker console.

4. Plots of data and prediction results by using Matplotlib.

5. Analytical discussion of the results and related conclusions.