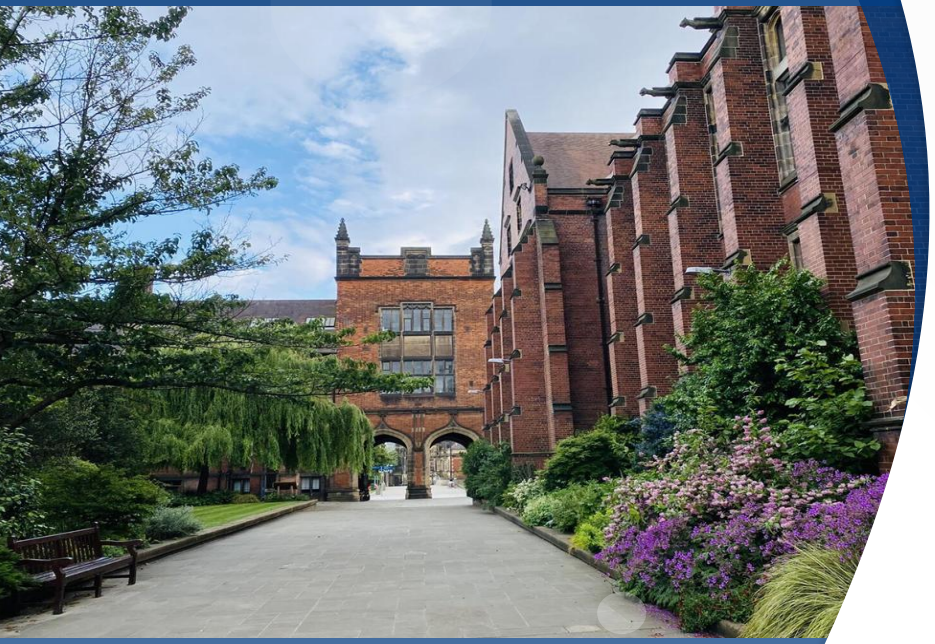
# CSC8112 IoT Module Coursework

Rui Sun

Newcastle University



# Outline

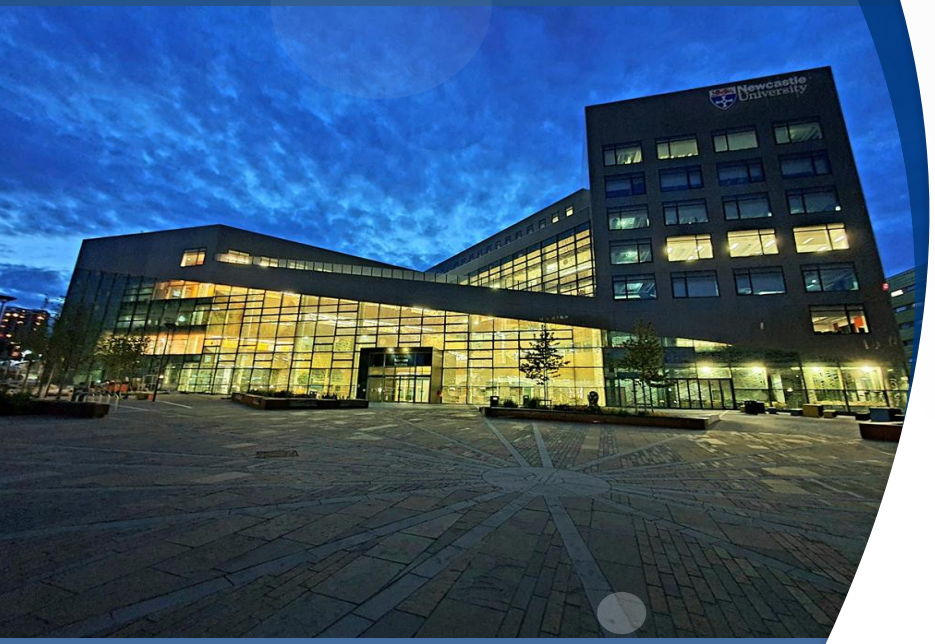
# Objectives

**To utilise IoT data to finish a specific task in a real application, which involves:**

- Data collection
- Data preprocessing
- Time-series sensor data prediction
- Visualisation

**To understand the pipeline implementation with data flow and deploying Docker-based application hosting environment**

Final report needs to be submitted to NESS by 15:00 pm on November 15, 2024
Live demonstration session will be organised on either November 14 or 15, 2024 (30min per student)




# Outline

# Microservices

# What is Microservices

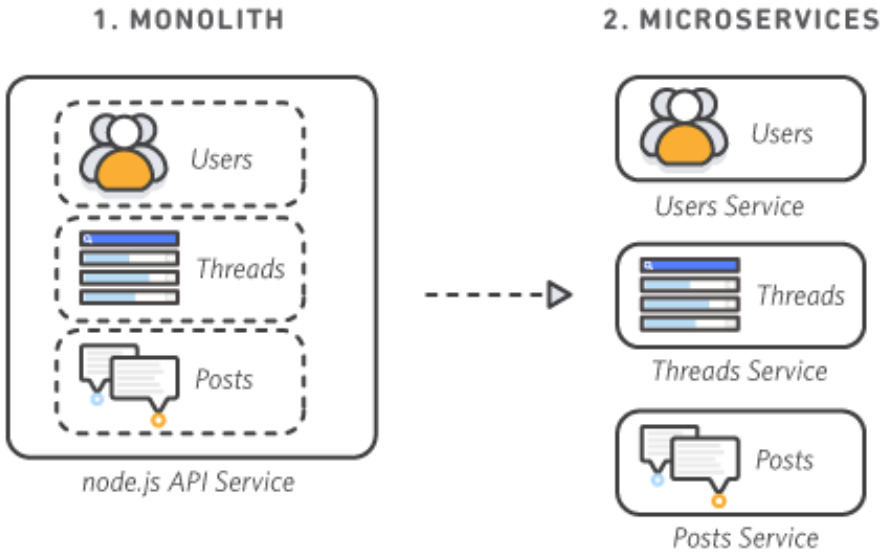
**Microservices** –
also known as the microservice architecture - is an architectural style that structures an application as a collection of services that are

- Highly maintainable and testable
- Loosely coupled
- Independently deployable
- Organized around business capabilities
- Owned by a small team

microservice architecture enables the rapid, frequent and reliable delivery of large, complex applications.

https://aws.amazon.com/microservices/

# What is Microservices




## Monolithic vs. Microservices Architecture

With monolithic architectures, all processes are tightly coupled and run as a single service.

With a microservices architecture,
an application is built as independent components that run each application process as a service

https://aws.amazon.com/microservices/

# Docker

# Docker

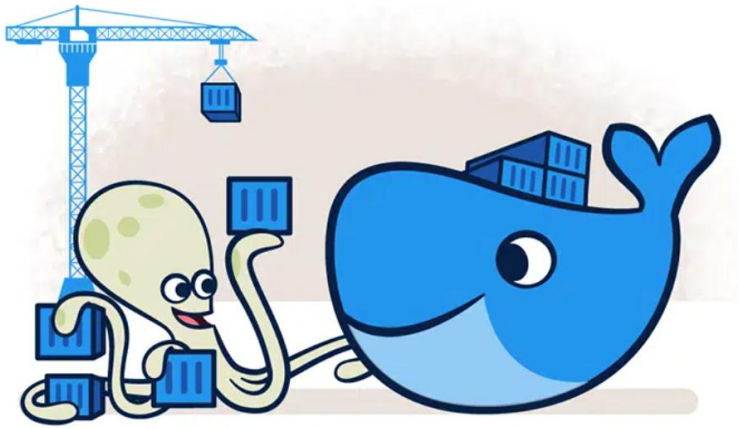
**What is Docker:**

- An open platform to help developers build, share, and run modern applications
- Separate applications from the infrastructure to deliver software quickly

Docker makes development efficient and predictable

Download: https://www.docker.com/

# Docker

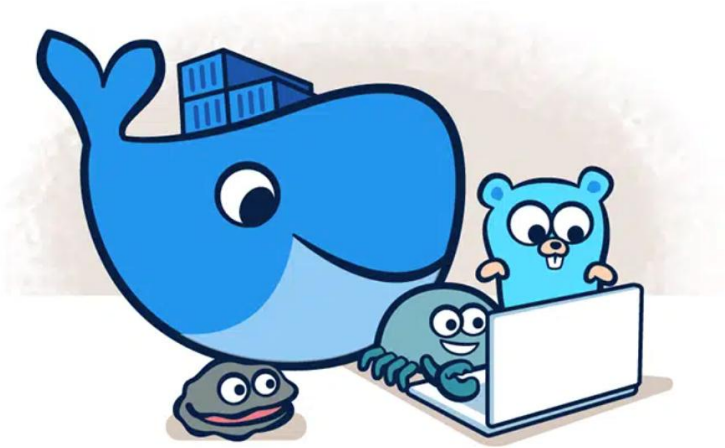


## Build:

- Get a head start on your coding by leveraging Docker images to efficiently develop your own unique applications on Windows and Mac.  Create your multi-container application using Docker Compose.
- Integrate with your favorite tools throughout your development pipeline – Docker works with all development tools you use including VS Code, CircleCI and GitHub.
- Package applications as portable container images to run in any environment consistently from on-premises Kubernetes to AWS ECS, Azure ACI, Google GKE and more.

https://www.docker.com/

# Docker



## Share:

- Leverage Docker Trusted Content, including Docker Official Images and images from Docker Verified Publishers from the Docker Hub repository.
- Innovate by collaborating with team members and other developers and by easily publishing images to Docker Hub.
- Personalize developer access to images with roles based access control and get insights into activity history with Docker Hub Audit Logs.

https://www.docker.com/

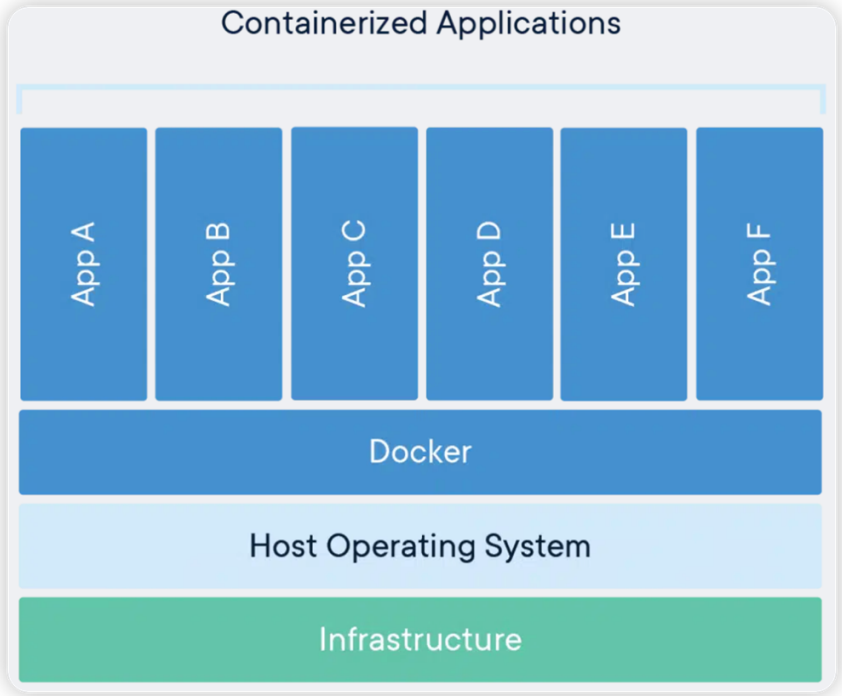
# Docker


## Run:

- Deliver multiple applications hassle free and have them run the same way on all your environments including design, testing, staging and production – desktop or cloud-native.
- Deploy your applications in separate containers independently and in different languages. Reduce the risk of conflict between languages, libraries or frameworks.
- Speed development with the simplicity of Docker Compose CLI and with one command, launch your applications locally and on the cloud with AWS ECS and Azure ACI.

https://www.docker.com/

# Docker container



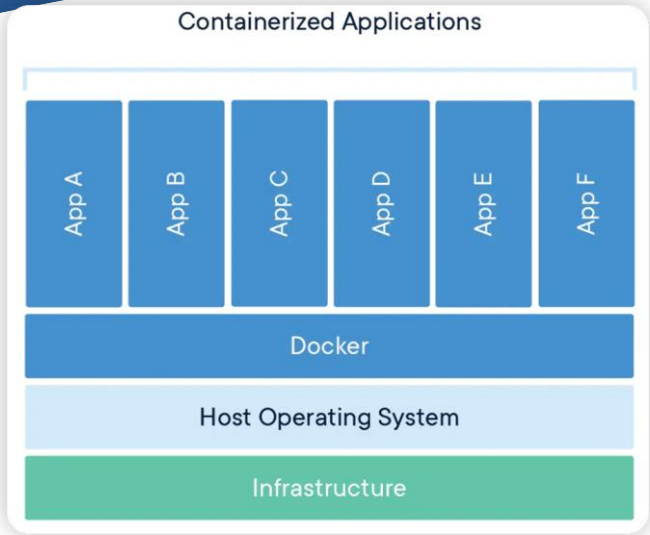

## What is docker container:

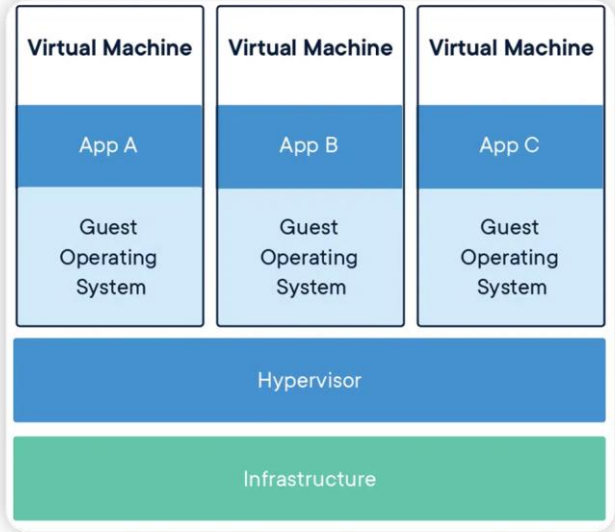
A container is a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another.

- **Standard:** Docker created the industry standard for containers, so they could be portable anywhere
- **Lightweight:** Do not require an OS per application and no licensing costs.
- **Secure:** Strongest default isolation capabilities in the industry.

https://www.docker.com/resources/what-container/

# Docker vs. Virtual Machines




## Containers

1. packages code and dependencies together.
2. Multiple containers can share OS kernel.
3. isolated processes.
4. less space (tens of MBs).
5. can handle more applications and require fewer VMs and OS.

## Virtual Machines

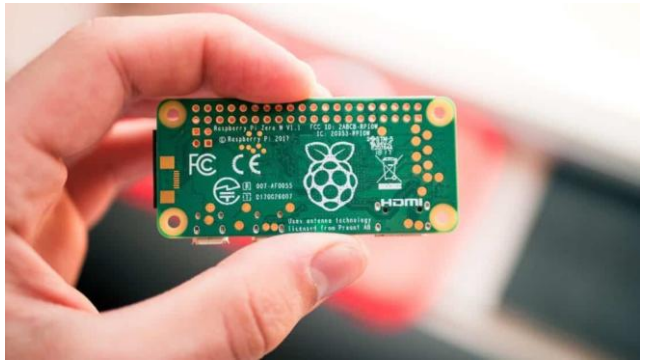
1. Physical hardware turns one server into many servers.
2. The hypervisor allows multiple VMs to run on a single machine.
3. Full copy of an OS, the application, necessary binaries and libraries (tens of GBs).
4. Slow to boot.


https://www.docker.com/resources/what-container/

# Containerized Microservices with IoT devices

## Why IoT Development needs Microservices and Containerization?

- Easy developing, integrating and scaling applications in large-scale IoT system
- Microservices and containerization enable efficient and faster development by breaking down IoT functionalities into small, modular and independent units that work in isolation without affecting the overall performance of the IoT ecosystem
- Docker container is lightweight and friendly with IoT devices
- Application update and iterate faster



https://www.docker.com/blog/from-edge-to-mainstream-scaling-to-100k-iot-devices/
https://www.einfochips.com/blog/why-iot-development-needs-microservices-and-containerization/

# Docker container

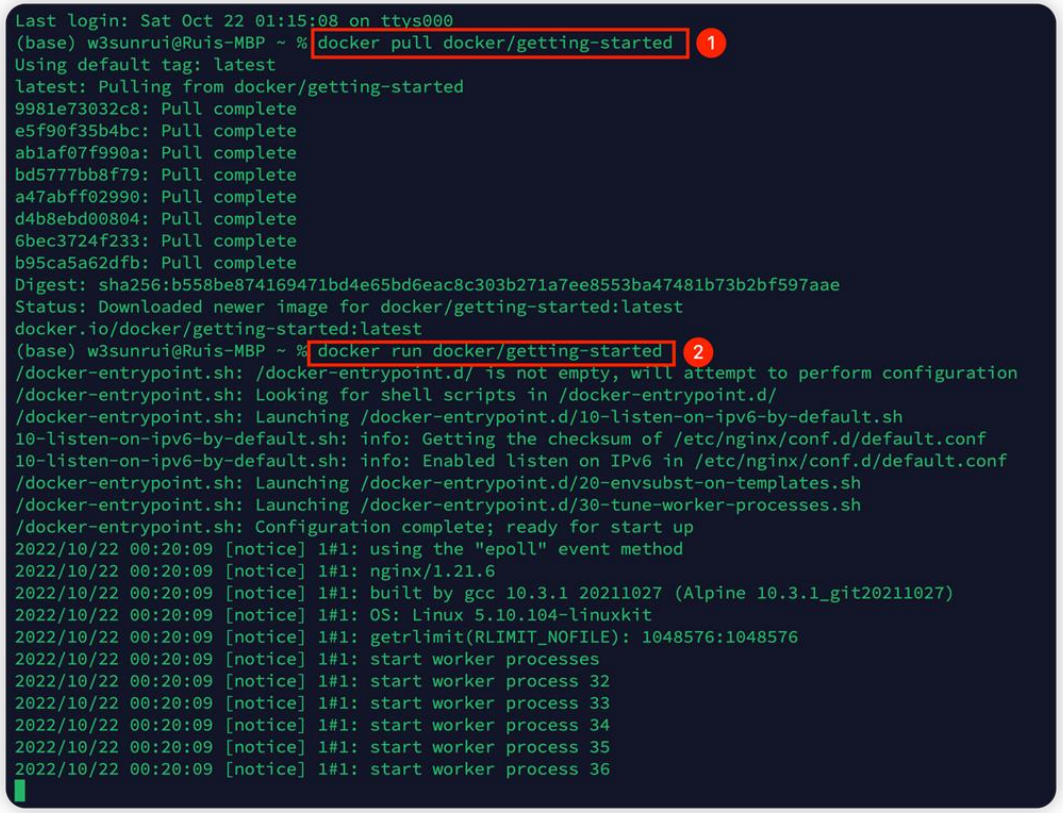
## How to run a Docker container from a public image:

### 1. Pull a docker image:

$ docker pull docker/getting-started

### 2. Run image as container:

$ docker run docker/getting-started




https://www.docker.com/resources/what-container/

# Docker Compose File

## Docker compose file:

The Compose file is a YAML file defining services, networks, and volumes for a Docker application.

## YAML file:

YAML is a data serialization language that is often used for writing configuration files.

```
{
  "doe": "a deer, a female deer",
  "ray": "a drop of golden sun",
  "pi": 3.14159,
  "xmas": true,
  "french-hens": 3,
  "calling-birds": [
     "huey",
     "dewey",
     "louie",
     "fred"
  ],
  "xmas-fifth-day": {
  "calling-birds": "four",
  "french-hens": 3,
  "golden-rings": 5,
  "partridges": {
    "count": 1,
    "location": "a pear tree"
  },
  "turtle-doves": "two"
  }
}
```

JSON

```
doe: "a deer, a female deer"
ray: "a drop of golden sun"
pi: 3.14159
xmas: true
french-hens: 3
calling-birds:
  - huey
  - dewey
  - louie
  - fred
xmas-fifth-day:
  calling-birds: four
  french-hens: 3
  golden-rings: 5
  partridges:
    count: 1
    location: "a pear tree"
  turtle-doves: two
```

YAML

# Docker Compose File

**How to automatically pull and start multiple services by defining a Docker Compose file.**

```
student@edge:~$ sudo apt  install docker-compose
[sudo] password for student:
Reading package lists ... Done
Building dependency tree
Reading state information ... Done
The following additional packages will be installed:
  python3-attr python3-cached-property python3-distut
  python3-importlib-metadata python3-jsonschema pytho
  python3-pyrsistent python3-setuptools python3-textt
Suggested packages:
  python-attr-doc python-jsonschema-doc python-setupt
Recommended packages:
  docker.io
The following NEW packages will be installed
  docker-compose python3-attr python3-cached-property
  python3-docopt python3-importlib-metadata python3-j
  python3-pyrsistent python3-setuptools python3-textt
0 to upgrade, 16 to newly install, 0 to remove and 4
```

1. Install Docker-Compose tool

docker-compose file example

```yaml
version: "3"
services:
  mongo:
    image: mongo
    deploy:
      replicas: 1
    ports:
      - '27017:27017'
```

2. Define docker-compose file

```
(ubifl) w3sunrui@Ruis-MBP component_demo % sudo docker-compose up
[+] Running 10/10
 ⠿ mongo Pulled
   ⠿ 514fa78e57ce Already exists
   ⠿ 808a7df5fbbc Pull complete
   ⠿ 8d5a5e151a7f Pull complete
   ⠿ bbed2c86a740 Pull complete
   ⠿ 6a5c6dc442fc Pull complete
   ⠿ 70d00e8640a3 Pull complete
   ⠿ d11bc600eb8d Pull complete
   ⠿ 02b720c57555 Pull complete
   ⠿ 194b45d19c19 Pull complete
[+] Running 2/2
 ⠿ Network component_demo_default    Created
 ⠿ Container component_demo-mongo-1  Created
```

3. Run docker-compose file

# How to build your code

## 1. Define DockerFile

Dockerfile

```
# Base on image_full_name (e.g., ubuntu:18.04) docker image
FROM python:3.8.12

# Switch to root
USER root

# Copy all sources files to workdir
ADD <your project directory> /usr/local/source

# Change working dir
WORKDIR /usr/local/source

# Prepare project required running system environments
#   requirements.txt is a document that pre-define any
#   python dependencies with versions required of your code
RUN pip3 install -r requirements.txt

# Start task
CMD python3 <your main .py file>
```

## 2. Run DockerFile to build your code into image

Build code

```
sudo docker build -t <name:tag> <source directory (relative)>
```

## 3. Define docker-compose file

docker-compose configuration file

```
version: "3"
services:
  data_injector:
    image: data_injector:latest
```

## 4. Run docker-compose file to start your code as a container

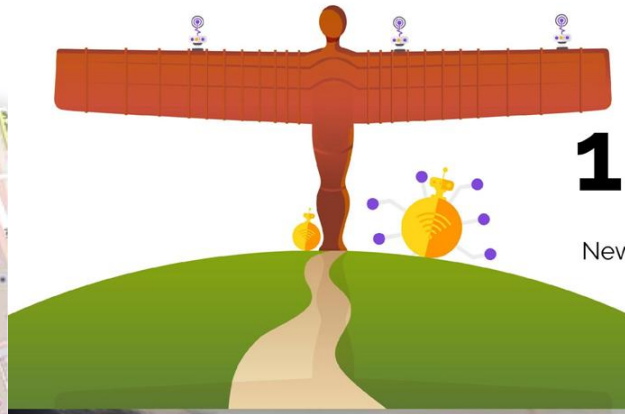
Start a image

```
sudo docker-compose up
```

Dockerfile https://docs.docker.com/engine/reference/builder/

# Urban Observation Platform

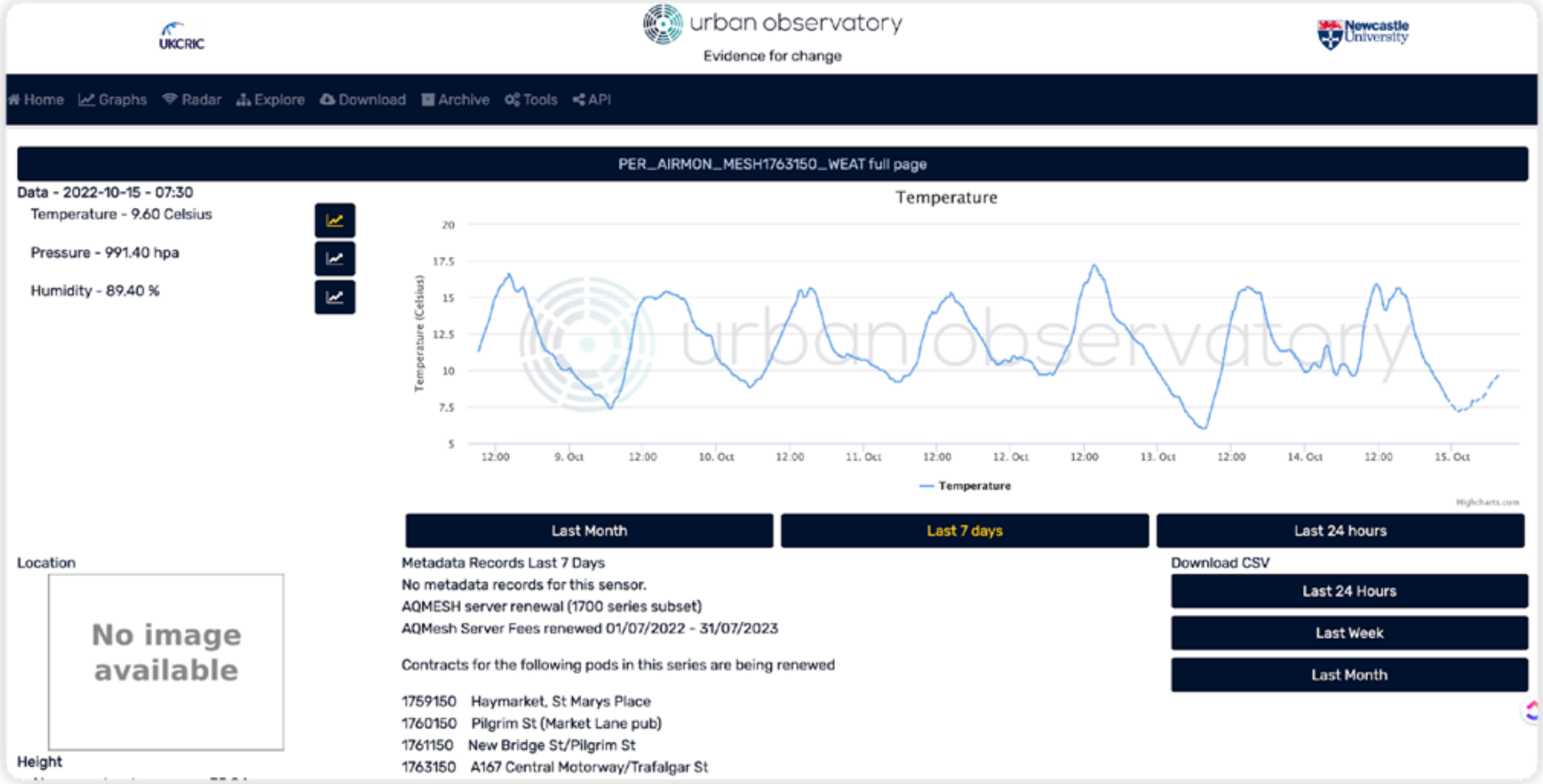
# Urban Observatory Platform

Deploying
**1,000+**
sensors across
Newcastle and Gateshead

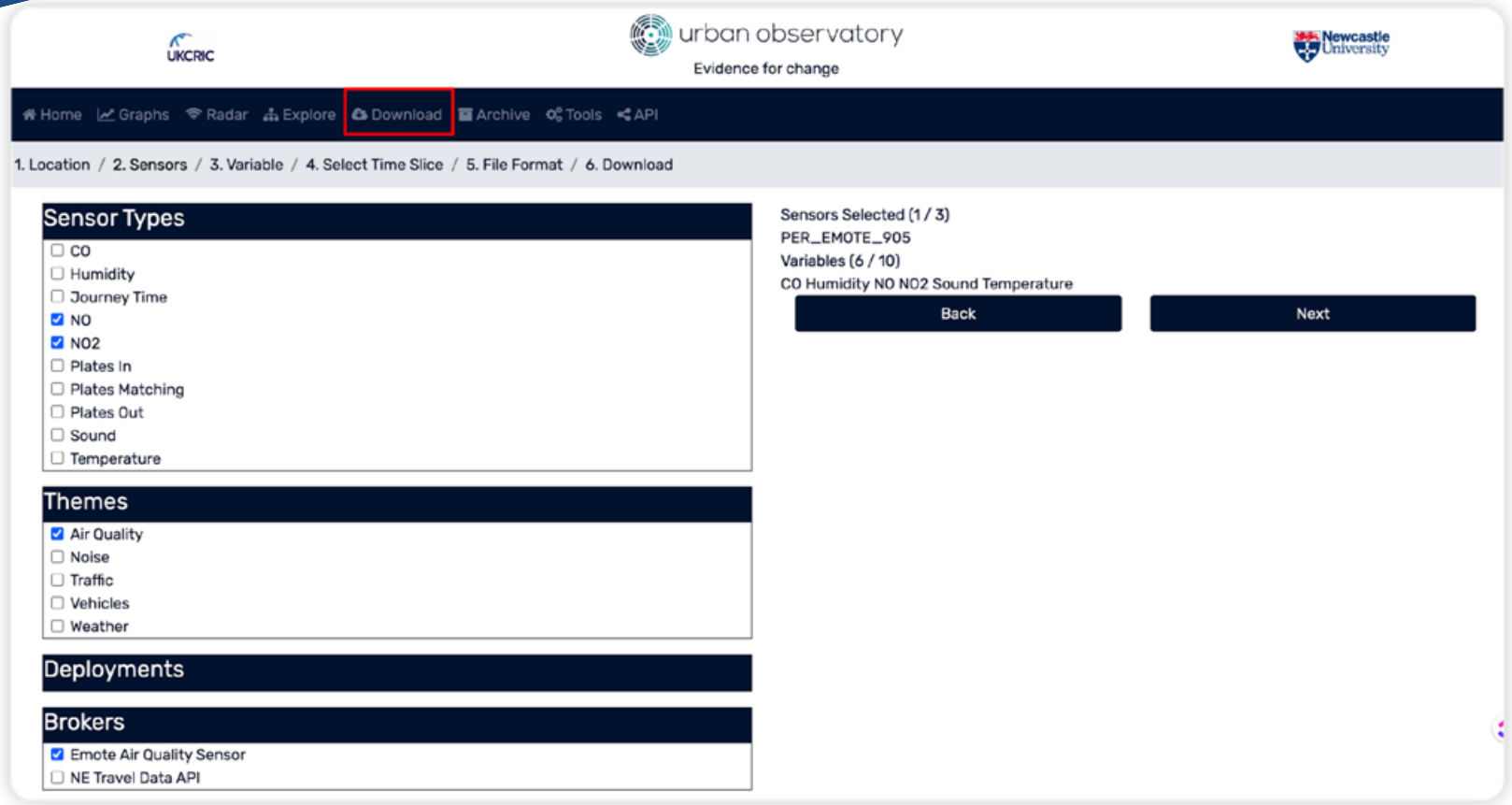

urban observatory https://urbanobservatory.ac.uk/

# Urban Observatory Platform




urban observatory https://urbanobservatory.ac.uk/

# Urban Observatory Platform

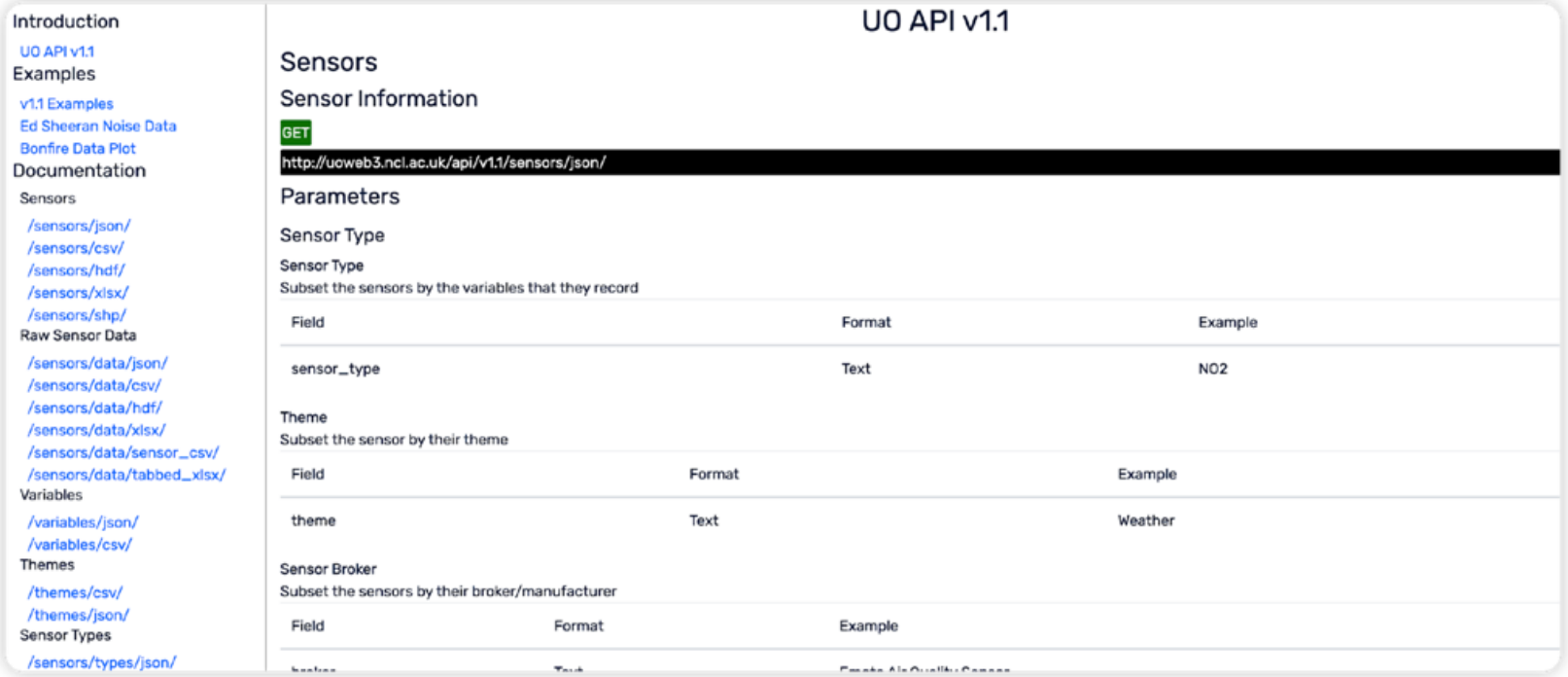



## Access data:

1. Directly download data as a csv file
2. HTTP API

urban observatory https://urbanobservatory.ac.uk/

# Urban Observatory Platform



Introduction
UO API v1.1
Examples
v1.1 Examples
Ed Sheeran Noise Data
Bonfire Data Plot
Documentation
Sensors
/sensors/json/
/sensors/csv/
/sensors/hdf/
/sensors/xlsx/
/sensors/shp/
Raw Sensor Data
/sensors/data/json/
/sensors/data/csv/
/sensors/data/hdf/
/sensors/data/xlsx/
/sensors/data/sensor_csv/
/sensors/data/tabbed_xlsx/
Variables
/variables/json/
/variables/csv/
Themes
/themes/csv/
/themes/json/
Sensor Types
/sensors/types/json/

UO API v1.1

Sensors

Sensor Information

GET

http://uoweb3.ncl.ac.uk/api/v1.1/sensors/json/

Parameters

Sensor Type

Sensor Type
Subset the sensors by the variables that they record

| Field | Format | Example |
|---|---|---|
| sensor_type | Text | NO2 |

Theme
Subset the sensor by their theme

| Field | Format | Example |
|---|---|---|
| theme | Text | Weather |

Sensor Broker
Subset the sensors by their broker/manufacturer

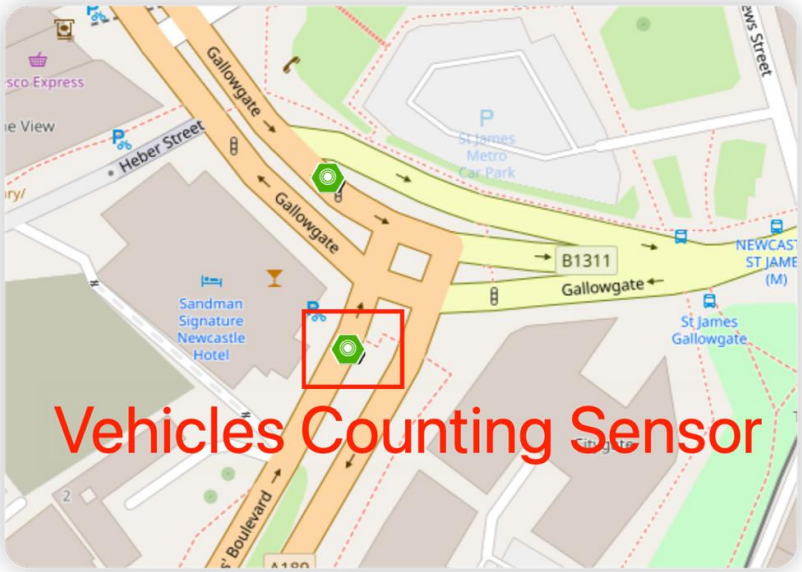
| Field | Format | Example |
|---|---|---|

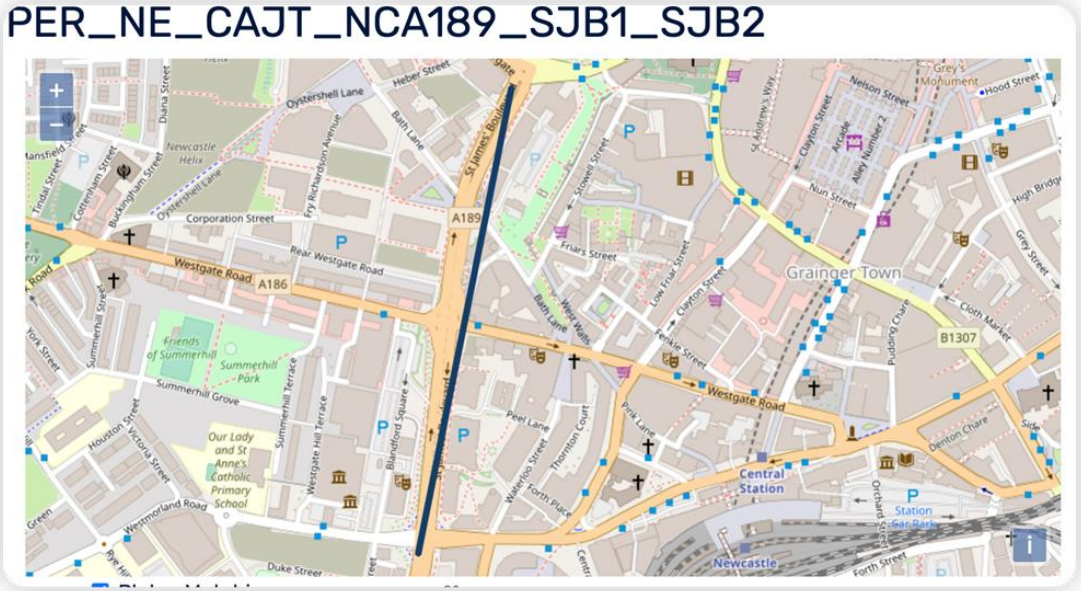
## Access data:

1. Directly download data as a csv file
2. HTTP API

urban observatory https://urbanobservatory.ac.uk/

# How to send HTTP request to get data by API




https://newcastle.urbanobservatory.ac.uk/sensors/sensor/PER_NE_CAJT_NCA189_SJB1_SJB2_VEHI/

# Pre-Requisites of HTTP request sending implementation

## Install requests a python package into your system

```
(ubifl) w3sunrui@Ruis-MBP ~ % pip install requests
Collecting requests
  Using cached requests-2.28.1-py3-none-any.whl (62 kB)
Requirement already satisfied: certifi>=2017.4.17 in ./opt/anaconda3/envs/ubifl/lib/python3.10/site-packages (from requests) (2022.6.15)
Requirement already satisfied: charset-normalizer<3,>=2 in ./opt/anaconda3/envs/ubifl/lib/python3.10/site-packages (from requests) (2.1.0)
Requirement already satisfied: idna<4,>=2.5 in ./opt/anaconda3/envs/ubifl/lib/python3.10/site-packages (from requests) (3.3)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in ./opt/anaconda3/envs/ubifl/lib/python3.10/site-packages (from requests) (1.26.10)
Installing collected packages: requests
Successfully installed requests-2.28.1
(ubifl) w3sunrui@Ruis-MBP ~ %
```

# How to send HTTP request to get data by API

HTTP Get

```python
import requests

if __name__ == '__main__':
    # Target URL
    url = "http://uoweb3.ncl.ac.uk/api/v1.1/sensors/PER_NE_CAJT_NCA189_SJB1_SJB2" \
          "/data/json/?starttime=20220901&endtime=20221001 "

    # Request data from Urban Observatory Platform
    resp = requests.get(url)

    # Convert response(Json) to dictionary format
    raw_data_dict = resp.json()

    print(raw_data_dict)

```


```
"sensors": [
    {
        "Broker Name": {
            "0": "NE Travel Data API"
        },
        "Sensor Centroid Latitude": {
            "0": 54.9708650063
        },
        "data": {
            "Plates Out": [
                {
                    "Flagged as Suspect Reading": false,
                    "Timestamp": 1661990640000,
                    "Value": 7.0,
                    "Variable": "Plates Out",
                    "Sensor Name": "PER_NE_CAJT_NCA189_SJB1_SJB2",
                    "Units": "Vehicles"
                },
                {
                    "Flagged as Suspect Reading": false,
                    "Timestamp": 1661990880000,
                    "Value": 1.0,
                    "Variable": "Plates Out",
                    "Sensor Name": "PER_NE_CAJT_NCA189_SJB1_SJB2",
                    "Units": "Vehicles"
                },
```

# Message Queue

# What is message queue?


A message queue is a form of asynchronous service-to-service communication used in serverless and microservices architectures

Messages are stored on the queue until they are processed and deleted. Each message is processed only once, by a single consumer.

Message queues can be used to decouple heavyweight processing, to buffer or batch work, and to smooth spiky workloads.


https://aws.amazon.com/message-queue/
https://www.cloudamqp.com/blog/what-is-message-queuing.html

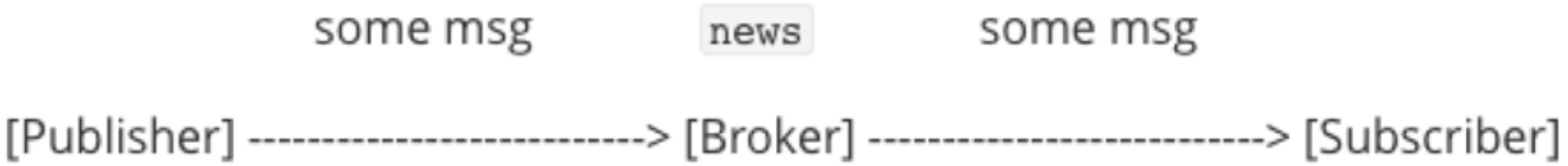
# MQTT Protocol

## MQTT protocol

an IoT communication protocol based on the Publish/Subscribe model, and it occupies half market of the Internet of Things protocol

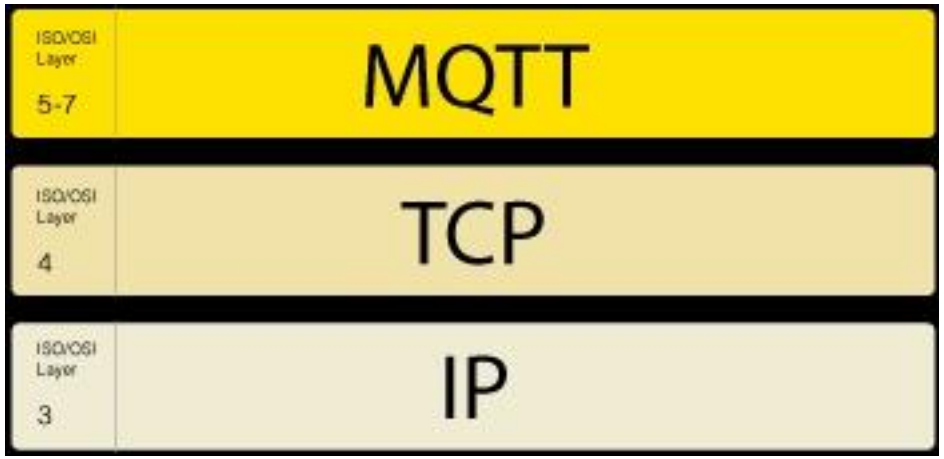
## Characteristics:

1. simple and easy implementation
2. support for QoS
3. and small size of packet



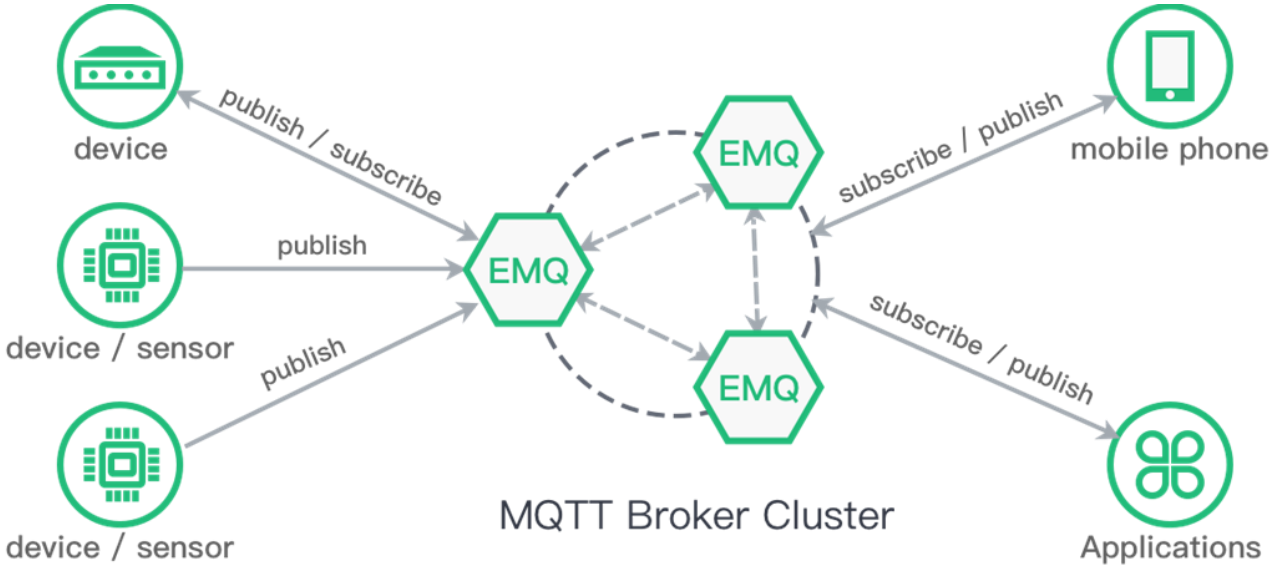

https://www.emqx.io/
https://info.hivemq.com/mqtt-essentials

# MQTT Protocol




The MQTT protocol is based on TCP/IP. Both the client and the broker need to have a TCP/IP stack.

The MQTT connection is always between one client and the broker. Clients never connect to each other directly.




https://www.emqx.io/
https://info.hivemq.com/mqtt-essentials

# EMQ X

**EMQ X:**
the most scalable and popular open-source MQTT broker with a high performance

### Massive Scale

Scale to **100 million** concurrent MQTT connections with a single EMQX 5.0 cluster.

### High Performance

Move and process **millions** of MQTT messages per second in a single broker.

### Low Latency

Guarantee **sub-millisecond latency** in message delivery with the soft real-time runtime.

### Fully MQTT 5.0

**100%** compliant with **MQTT 5.0 and 3.x** standard for better scalability, security, and reliability.

### High Availability

Achieve high availability and horizontal scalability with a masterless distributed architecture.

### Cloud-Native & K8s

Easy to deploy on-premises and in public clouds with **Kubernetes Operator** and **Terraform**.

**EMQ X** https://www.emqx.io/

# Pre-Requisites of MQTT client implementation

Install paho.mqtt a python package into your system

```
(ubifl) w3sunrui@Ruis-MBP ~ % pip install paho.mqtt
Collecting paho.mqtt
  Using cached paho_mqtt-1.6.1-py3-none-any.whl
Installing collected packages: paho.mqtt
Successfully installed paho.mqtt-1.6.1
```

# How to define MQTT subscriber in Python

MQTT subscriber

```python
import json

from paho.mqtt import client as mqtt_client

if __name__ == '__main__':
    mqtt_ip = "localhost"
    mqtt_port = 1883
    topic = "CSC8112"

    # Create a mqtt client object
    client = mqtt_client.Client()


    # Callback function for MQTT connection
    def on_connect(client, userdata, flags, rc):
        if rc == 0:
            print("Connected to MQTT OK!")
        else:
            print("Failed to connect, return code %d\n", rc)

    # Connect to MQTT service
    client.on_connect = on_connect
    client.connect(mqtt_ip, mqtt_port)

    # Callback function will be triggered
    def on_message(client, userdata, msg):
        print(f"Get message from publisher {json.loads(msg.payload)}")

    # Subscribe MQTT topic
    client.subscribe(topic)
    client.on_message = on_message

    # Start a thread to monitor message from publisher
    client.loop_forever()
```

# How to define MQTT publisher in Python

MQTT publisher

```python
import json
from paho.mqtt import client as mqtt_client

if __name__ == '__main__':
    mqtt_ip = "localhost"
    mqtt_port = 1883
    topic = "CSC8112"
    msg = "Hello!"

    # Create a mqtt client object
    client = mqtt_client.Client()

    # Callback function for MQTT connection
    def on_connect(client, userdata, flags, rc):
        if rc == 0:
            print("Connected to MQTT OK!")
        else:
            print("Failed to connect, return code %d\n", rc)

    # Connect to MQTT service
    client.on_connect = on_connect
    client.connect(mqtt_ip, mqtt_port)

    # Publish message to MQTT
    # Note: MQTT payload must be a string, bytearray, int, float or None
    msg = json.dumps(msg)
    client.publish(topic, msg)
```

# Test

## Run Subscriber -> Run Publisher

```
/Users/w3sunrui/opt/anaconda3/envs/
Connected to MQTT OK!
Get message from publisher Hello!
```

# RabbitMQ

## What is RabbitMQ?

RabbitMQ is a widely used open-source message broker that helps in scaling the application by deploying a message queuing mechanism in between the two applications.

## Main features of RabbitMQ:

- **Distributed Deployment:** Users can deploy RabbitMQ as clusters for high availability and throughput as well. The clusters are federated across multiple availability zones and regions to be available every time.
- **Tools and Plugins:** RabbitMQ offers a plethora of tools and plugins for continuous integration, operational metrics, and integration to other enterprise systems.
- **Enterprise and Cloud Ready:** RabbitMQ is lightweight and easy to deploy on the public as well as private clouds using pluggable authentication authorization.
- **Asynchronous Messaging:** RabbitMQ supports multiple messaging protocols, message queuing, delivery acknowledgment, routing to queues, and different exchange types.

RabbitMQ https://www.rabbitmq.com/



https://www.cloudamqp.com/blog/part1-rabbitmq-for-beginners-what-is-rabbitmq.htm
https://hevodata.com/learn/rabbitmq-queue/ l

# RabbitMQ

## Basic concepts of RabbitMQ:

- **Producer:** A producer is the one who sends messages to a queue based on the queue name.
- **Queue:** A Queue is a sequential data structure that is a medium through which messages are transferred and stored.
- **Consumer:** A consumer is the one who subscribes to and receives messages from the broker and uses that message for other defined operations.
- **Broker:** It is a message broker which provides storage for data produced. The data is meant to be consumed or received by another application that connects to the broker with the given parameters or connection strings.
- **Channel:** Channels offer a lightweight connection to a broker via a shared TCP connection.


RabbitMQ https://www.rabbitmq.com/



https://www.cloudamqp.com/blog/part1-rabbitmq-for-beginners-what-is-rabbitmq.htm
https://hevodata.com/learn/rabbitmq-queue/ l

# Pre-Requisites of RabbitMQ client implementation

Install pika a python package into your system

```
(ubifl) w3sunrui@Ruis-MBP ~ % pip install pika
Collecting pika
  Using cached pika-1.3.1-py3-none-any.whl (155 kB)
Installing collected packages: pika
Successfully installed pika-1.3.1
(ubifl) w3sunrui@Ruis-MBP ~ %
```

# How to define RabbitMQ consumer in Python

RabbitMQ consumer

```python
import json
import pika

if __name__ == '__main__':

    rabbitmq_ip = "localhost"
    rabbitmq_port = 5672
    # Queue name
    rabbitmq_queque = "CSC8112"

    def callback(ch, method, properties, body):
        print(f"Got message from producer msg: {json.loads(body)}")

    # Connect to RabbitMQ service with timeout 1min
    connection = pika.BlockingConnection(
        pika.ConnectionParameters(host=rabbitmq_ip, port=rabbitmq_port, socket_timeout=60))
    channel = connection.channel()
    # Declare a queue
    channel.queue_declare(queue=rabbitmq_queque)

    channel.basic_consume(queue=rabbitmq_queque,
                          auto_ack=True,
                          on_message_callback=callback)

    channel.start_consuming()
```

# How to define RabbitMQ producer in Python

RabbitMQ producer

```python
import pika
import json

if __name__ == '__main__':
    rabbitmq_ip = "localhost"
    rabbitmq_port = 5672
    # Queue name
    rabbitmq_queque = "CSC8112"
    msg = "Hello!"
    # Connect to RabbitMQ service
    connection = pika.BlockingConnection(pika.ConnectionParameters(host=rabbitmq_ip, port=rabbitmq_port))
    channel = connection.channel()

    # Declare a queue
    channel.queue_declare(queue=rabbitmq_queque)

    # Produce message
    channel.basic_publish(exchange='',
                          routing_key=rabbitmq_queque,
                          body=json.dumps(msg))

    connection.close()
```

# Test

Run Consumer -> Run Producer

```
/Users/w3sunrui/opt/anaconda3/envs/ubit
Got message from producer msg: Hello!
```

# Data Injector

# Data Injector

## What is Data Injector?

The data injector is responsible for processing, reformatting and transmitting the data before it is stored in a database or message queue.

Ref: https://www.matrixtsl.com/wiki/index.php?title=Using_the_Data_Injector#:~:text=The%20data%20injectors%20are%20designed,which%20is%20the%20embedded%20device.

# Machine Learning

# Machine Learning

## What is Machine Learning?

Machine learning (ML) is a type of artificial intelligence (AI) that use algorithms to become more accurate over time without human intervention. Instead of hard-coding or defining the outcome, a machine learning model uses data to learn how to make a decision and then incorporates feedback to improve accuracy over time.



https://streamsets.com/learn/machine-learning/

## Applications in Computer Vision

Object detection, semantic segmentation, pose estimation, and almost every other task is done with ML.

Figure 4. More results of **Mask R-CNN** on COCO test images, using ResNet-101-FPN and running at 5 fps, with 35.7 mask AP (Table 1).


DAQUAR 1553
**What is there in front of the sofa?**
Ground truth: table
IMG+BOW: table (0.74)
2-VIS+BLSTM: table (0.88)
LSTM: chair (0.47)

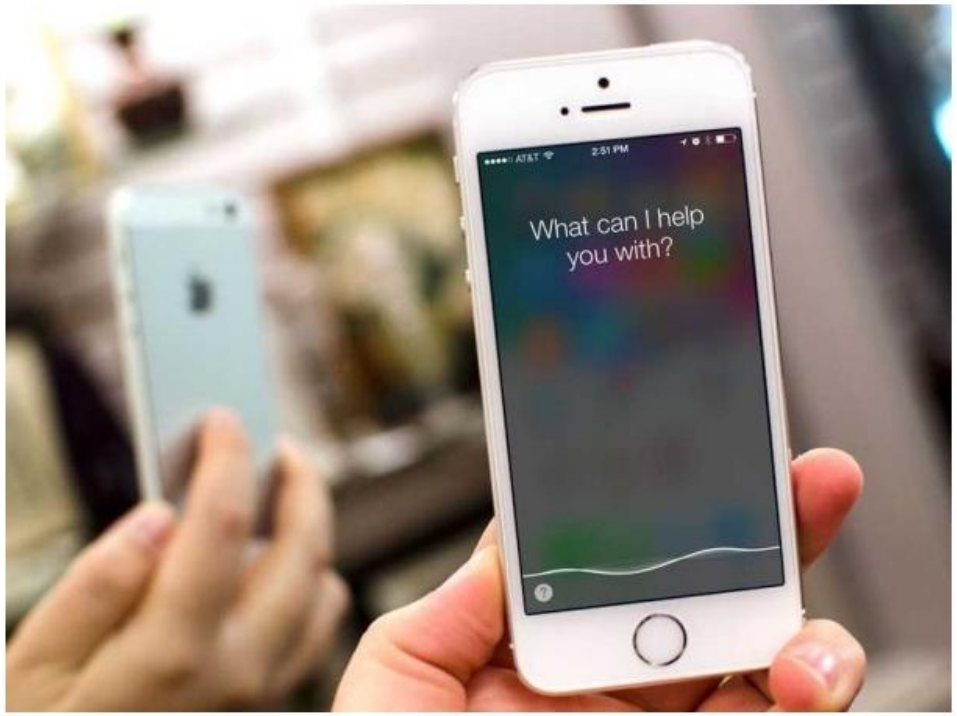
COCOQA 5078
**How many leftover donuts is the red bicycle holding?**
Ground truth: three
IMG+BOW: two (0.51)
2-VIS+BLSTM: three (0.27)
BOW: one (0.29)



https://www.cs.toronto.edu/~rgrosse/courses/csc311_f20/slides/lec01.pdf

## Applications in Speech

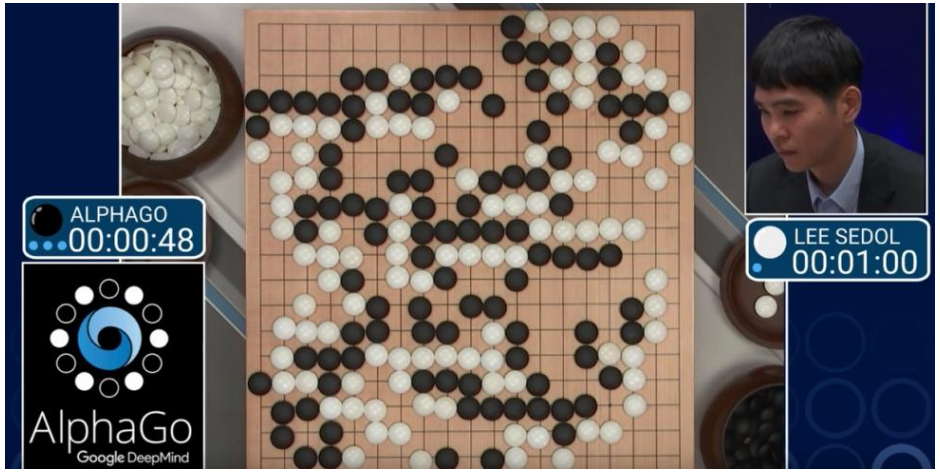
Speech to text, personal assistants, speaker identification...






https://www.cs.toronto.edu/~rgrosse/courses/csc311_f20/slides/lec01.pdf

**Applications in playing games and recommender systems**

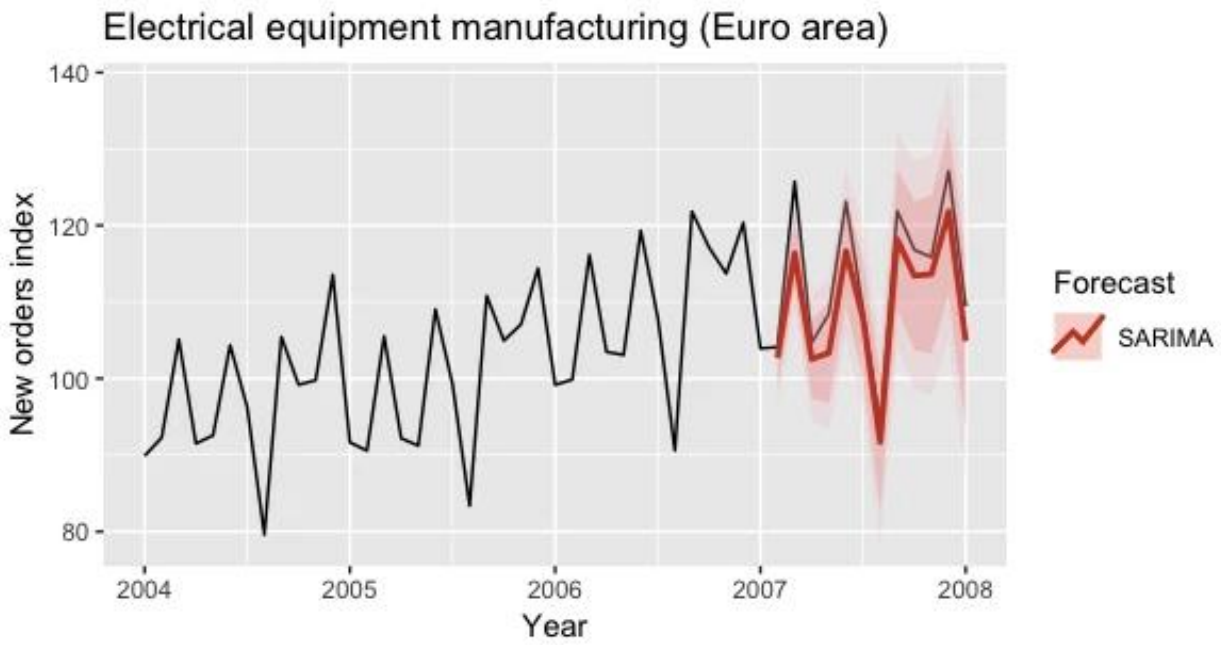





https://www.cs.toronto.edu/~rgrosse/courses/csc311_f20/slides/lec01.pdf

## What is time series data forecasting?

Time Series Forecasting makes use of the best fitting model essential to predicting the future observation based on complex processing current and previous data.



# Machine Learning Engine

## Input:

The  input data type should be pandas.DataFrame with Columns name ["Timestamp","Value"]
**Note:** All Timestamp data should be reformatted as DateTime (i.e. year-month-day hour:mint:second)

## What is Dataframe of Pandas

Dataframe is a 2-dimensional labelled data structure with columns of potentially different types. It like a spreadsheet or SQL table.

| | Timestamp | Value |
|---|---|---|
| 0 | 2022-06-01 00:00:00 | 5.616490 |
| 1 | 2022-06-02 00:00:00 | 6.219284 |
| 2 | 2022-06-03 00:00:00 | 5.633768 |
| 3 | 2022-06-04 00:00:00 | 4.195737 |
| 4 | 2022-06-05 00:00:00 | 5.400642 |

## Output:

A figure object of matplotlib.pyplot

Download Machine Learning Engine: https://github.com/ncl-iot-team/CSC8112_MLEngine

# Pre-Requisites of using Machine Learning Engine

## Install Pandas and Prophet python packages into your system

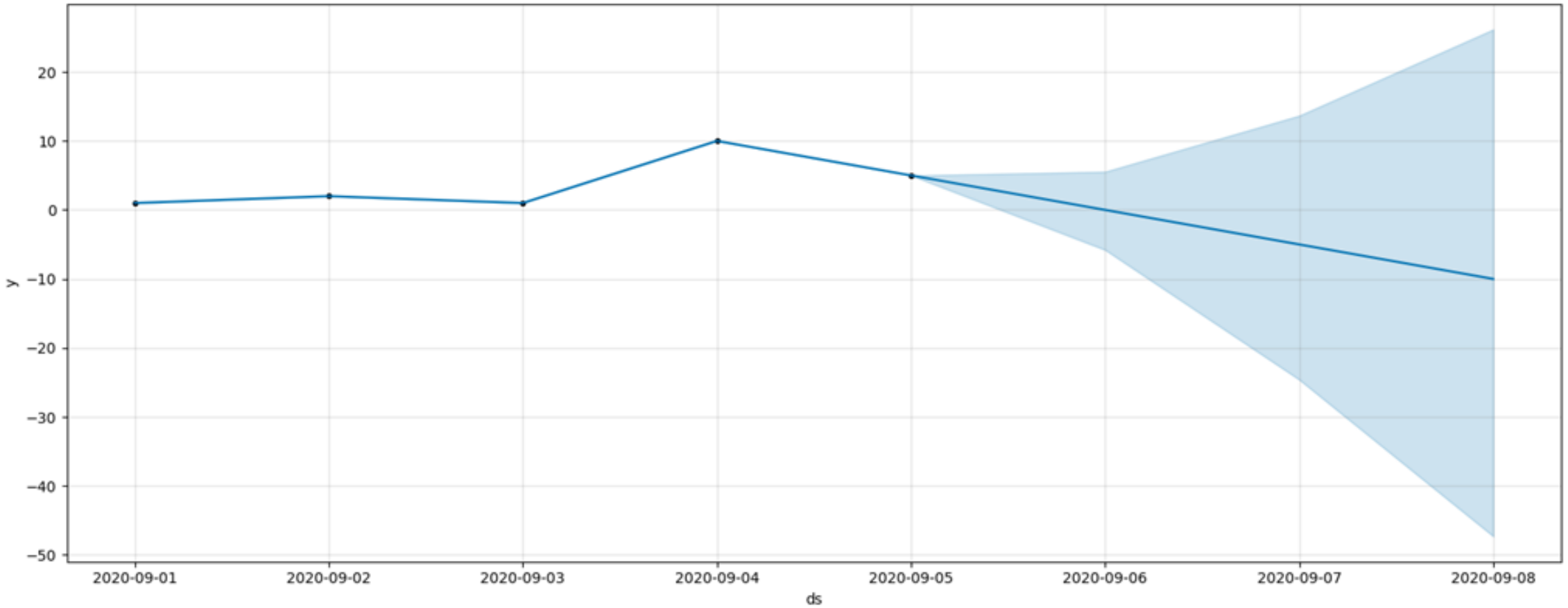
```
(ubifl) w3sunrui@Ruis-MBP ~ % pip install pandas
Collecting pandas
  Downloading pandas-1.5.1-cp310-cp310-macosx_10_9_x86_64.whl (12.0 MB)
     |████████████████████████████████| 12.0 MB 825 kB/s
Requirement already satisfied: python-dateutil>=2.8.1 in ./opt/anaconda3/envs/ubifl/lib/python3
Requirement already satisfied: numpy>=1.21.0 in ./opt/anaconda3/envs/ubifl/lib/python3.10/site-
Requirement already satisfied: pytz>=2020.1 in ./opt/anaconda3/envs/ubifl/lib/python3.10/site-p
Requirement already satisfied: six>=1.5 in ./opt/anaconda3/envs/ubifl/lib/python3.10/site-packa
1.16.0)
Installing collected packages: pandas
Successfully installed pandas-1.5.1
(ubifl) w3sunrui@Ruis-MBP ~ % pip install prophet
Collecting prophet
  Downloading prophet-1.1.1-cp310-cp310-macosx_10_9_x86_64.whl (6.8 MB)
     |████████████████████████████████| 6.8 MB 1.1 MB/s
Collecting convertdate>=2.1.2
  Downloading convertdate-2.4.0-py3-none-any.whl (47 kB)
```

# How to use Machine Learning Engine

Using Machine Learning Engine

```
from ml_engine import MLPredictor

if __name__ == '__main__':
    # Prepare data
    data = {
        'Timestamp': ['2020-09-01', '2020-09-02', '2020-09-03',
                      '2020-09-04', '2020-09-05'],
        'Value': [1, 2, 1, 10, 5]
    }
    data_df = pd.DataFrame(data)

    # Create ML engine predictor object
    predictor = MLPredictor(data_df)
    # Train ML model
    predictor.train()
    # Do prediction
    forecast = predictor.predict()

    # Get canvas
    fig = predictor.plot_result(forecast)
    fig.savefig("prediction.png")
    fig.show()
```

# Test


20
10
0
−10
−20
−30
−40
−50
y
2020-09-01
2020-09-02
2020-09-03
2020-09-04
2020-09-05
2020-09-06
2020-09-07
2020-09-08
ds

# Data Visualization

# Data Visualization

## Matplotlib

Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python.


Documents: https://matplotlib.org/stable/index.html

matplotlib https://matplotlib.org/

# Pre-Requisites of visualization implementation

## Install matplotlib a python package into your system

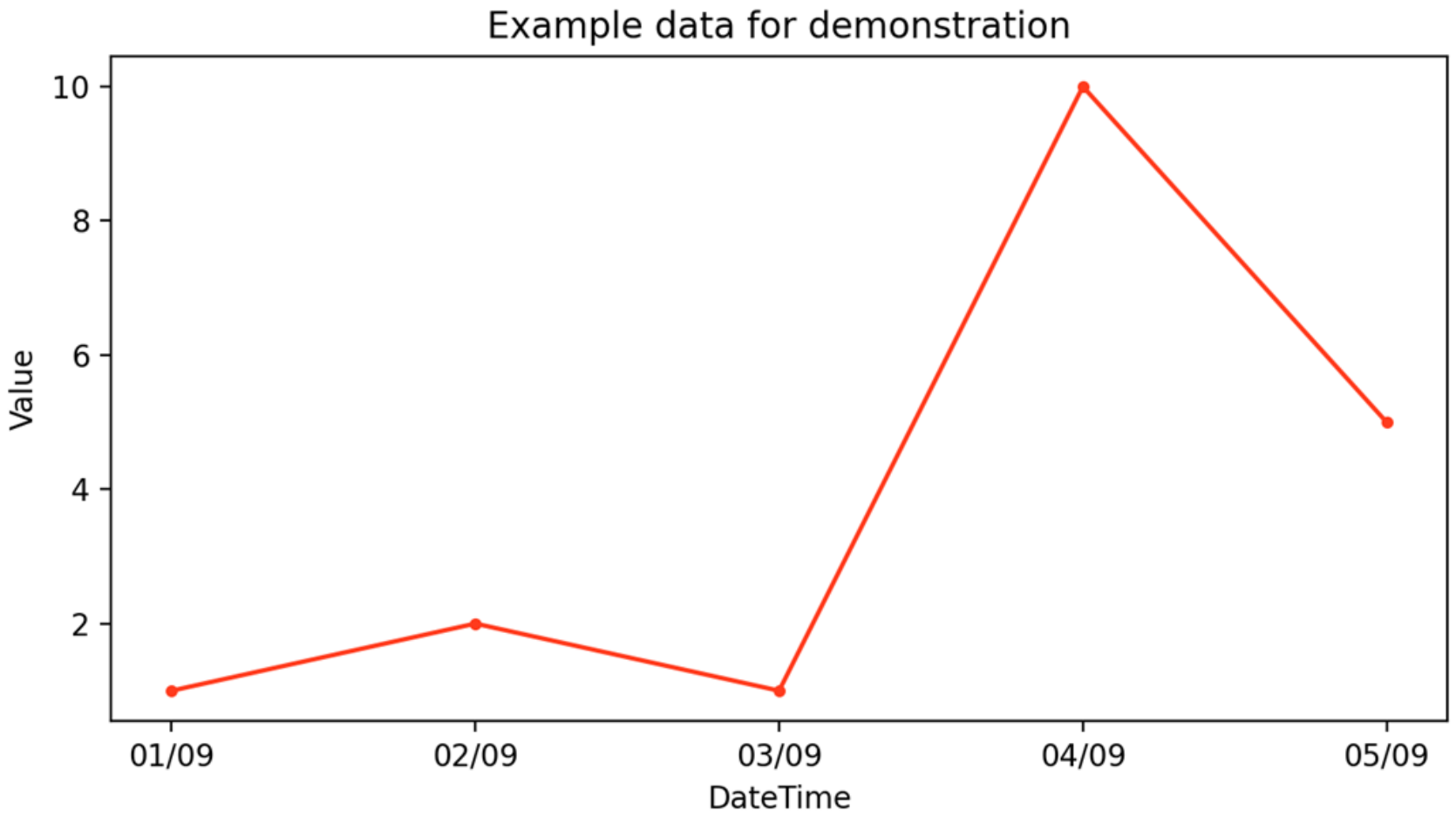
```
(ubifl) w3sunrui@Ruis-MBP ~ % pip install matplotlib
Collecting matplotlib
  Downloading matplotlib-3.6.1-cp310-cp310-macosx_10_12_x86_64.whl (7.3 MB)
     |████████████████████████████████| 7.3 MB 17 kB/s
Requirement already satisfied: kiwisolver>=1.0.1 in ./opt/anaconda3/envs/ubifl/lib/python3.10/site-packages (from matplotlib) (1.4.3)
Collecting contourpy>=1.0.1
  Downloading contourpy-1.0.5-cp310-cp310-macosx_10_9_x86_64.whl (241 kB)
     |████████████████████████████████| 241 kB 52 kB/s
Requirement already satisfied: cycler>=0.10 in ./opt/anaconda3/envs/ubifl/lib/python3.10/site-packages (from matplotlib) (0.11.0)
Requirement already satisfied: numpy>=1.19 in ./opt/anaconda3/envs/ubifl/lib/python3.10/site-packages (from matplotlib) (1.23.1)
Requirement already satisfied: pillow>=6.2.0 in ./opt/anaconda3/envs/ubifl/lib/python3.10/site-packages (from matplotlib) (9.2.0)
Requirement already satisfied: python-dateutil>=2.7 in ./opt/anaconda3/envs/ubifl/lib/python3.10/site-packages (from matplotlib) (2.8.2)
Requirement already satisfied: fonttools>=4.22.0 in ./opt/anaconda3/envs/ubifl/lib/python3.10/site-packages (from matplotlib) (4.34.4)
Requirement already satisfied: pyparsing>=2.2.1 in ./opt/anaconda3/envs/ubifl/lib/python3.10/site-packages (from matplotlib) (3.0.9)
Requirement already satisfied: packaging>=20.0 in ./opt/anaconda3/envs/ubifl/lib/python3.10/site-packages (from matplotlib) (21.3)
Requirement already satisfied: six>=1.5 in ./opt/anaconda3/envs/ubifl/lib/python3.10/site-packages (from python-dateutil>=2.7->matplotlib)
 (1.16.0)
Installing collected packages: contourpy, matplotlib
Successfully installed contourpy-1.0.5 matplotlib-3.6.1
(ubifl) w3sunrui@Ruis-MBP ~ %
```

# How to use Matplotlib to visualize a line chart

Matplotlib line chart

```python
import matplotlib.pyplot as plt
import pandas as pd

if __name__ == '__main__':
    # Prepare data
    data = {
        'Timestamp': ['01/09','02/09','03/09','04/09','05/09'],
        'Value': [1,2,1,10,5]
    }

    data_df = pd.DataFrame(data)

    # Initialize a canvas
    plt.figure(figsize=(20, 9), dpi=200)
    # Plot data into canvas
    plt.plot(data_df["Timestamp"], data_df["Value"], color="#FF3B1D", marker='.', linestyle="-")
    plt.title("Example data for demonstration")
    plt.xlabel("DateTime")
    plt.ylabel("Value")

    # Save as file
    plt.savefig("figure1.png")
    # Directly display
    plt.show()
```

# Test

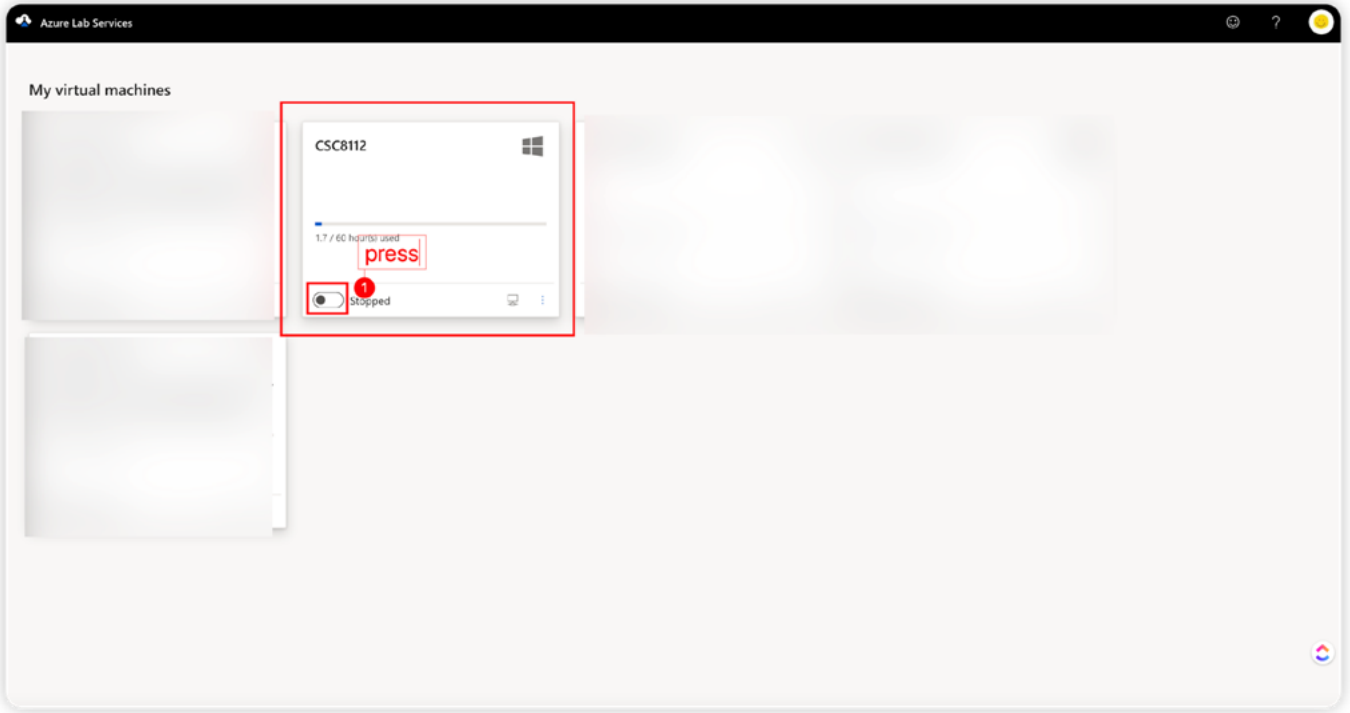

Example data for demonstration
10
8
6
4
2
Value
01/09
02/09
03/09
04/09
05/09
DateTime

# Azure Lab

# What is Azure Lab?

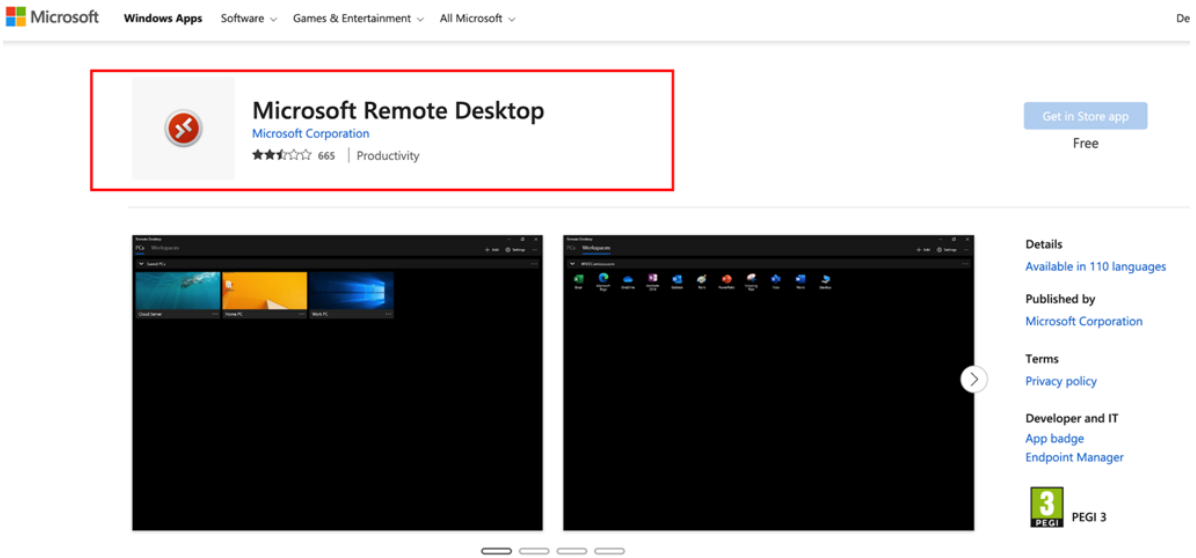
Azure Labs is a cloud based computer lab that allows schools and other organizations to run programs on preconfigured virtual machines.

# How to access Azure Lab

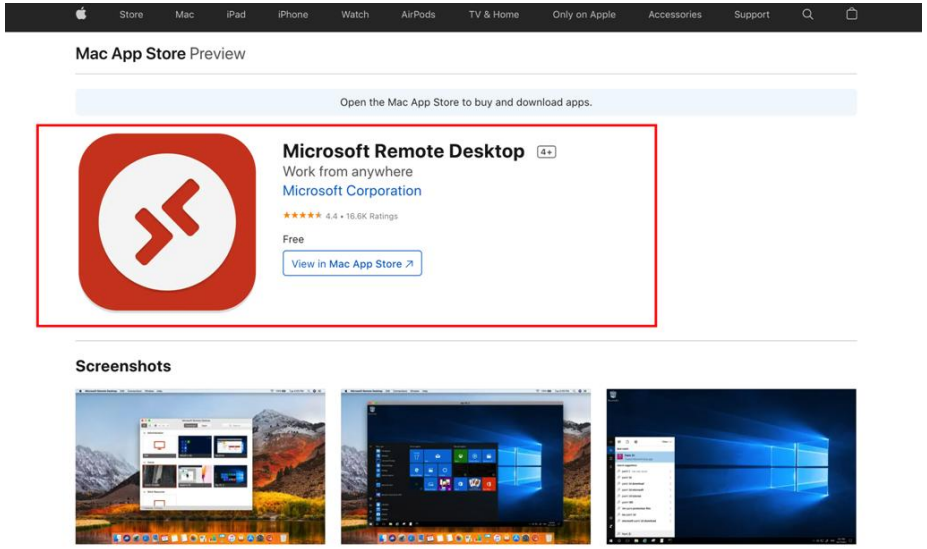



Login via:
https://labs.azure.com/virtualmachines

# How to access Azure Lab

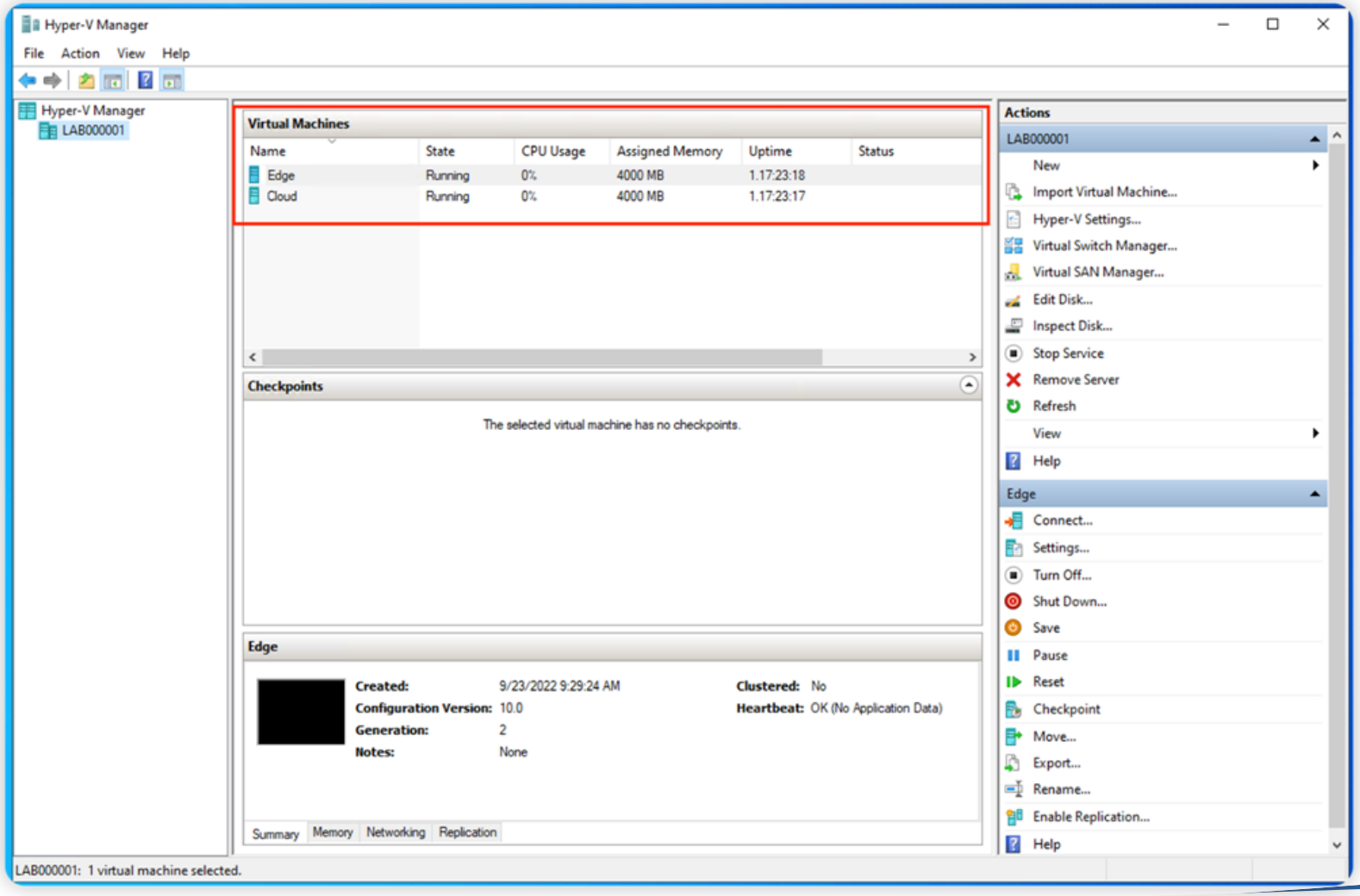


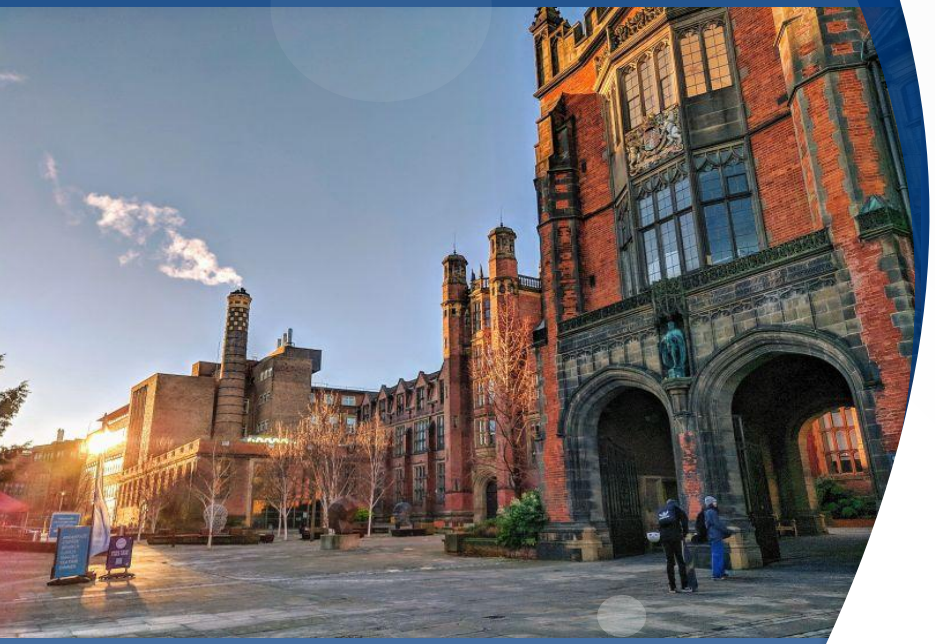

Windows




Mac OS

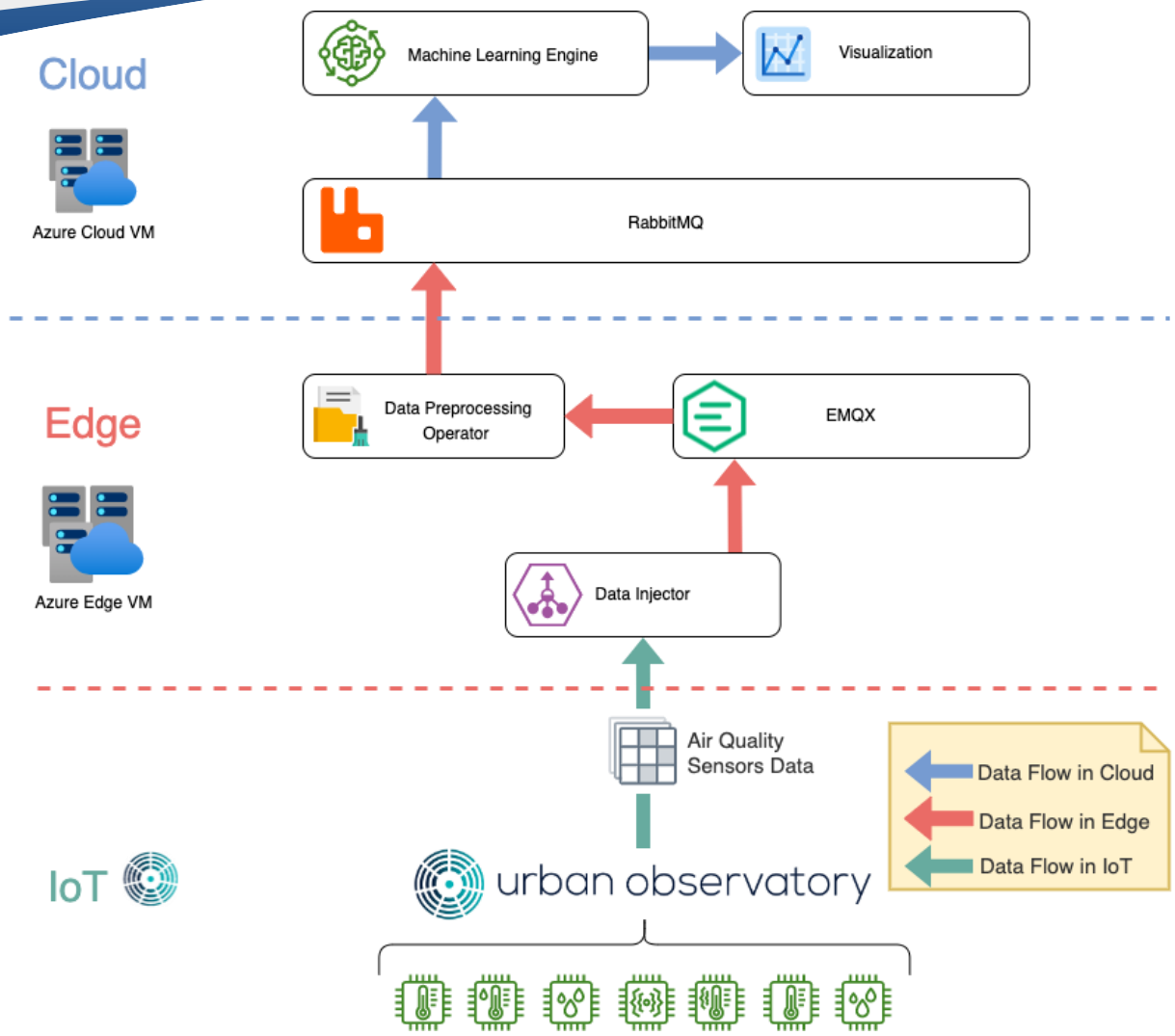
# How to access Azure Lab

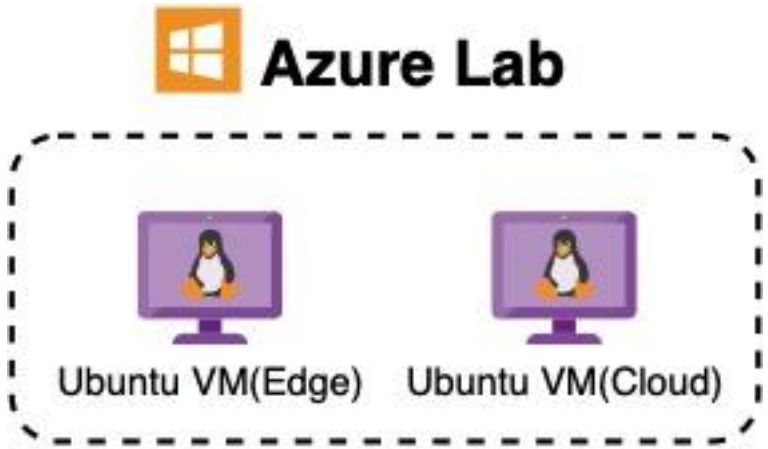




# Outline

# Overview Architecture

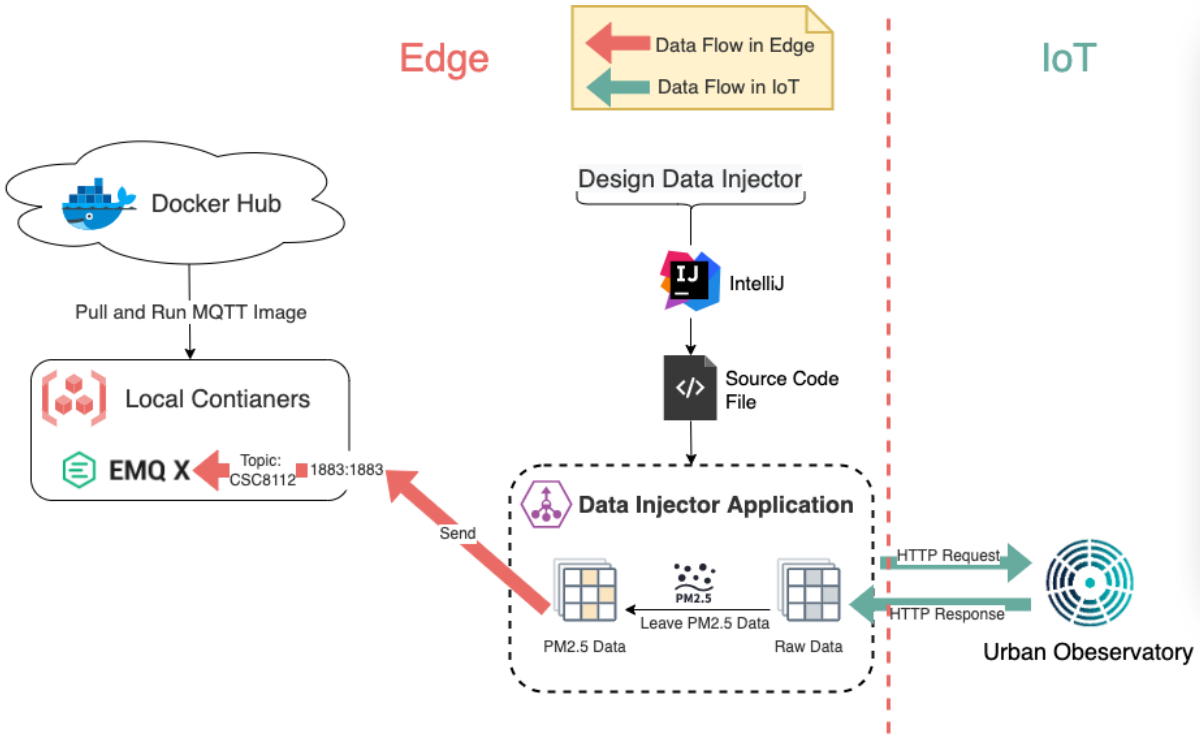



# Outline

# Task 1

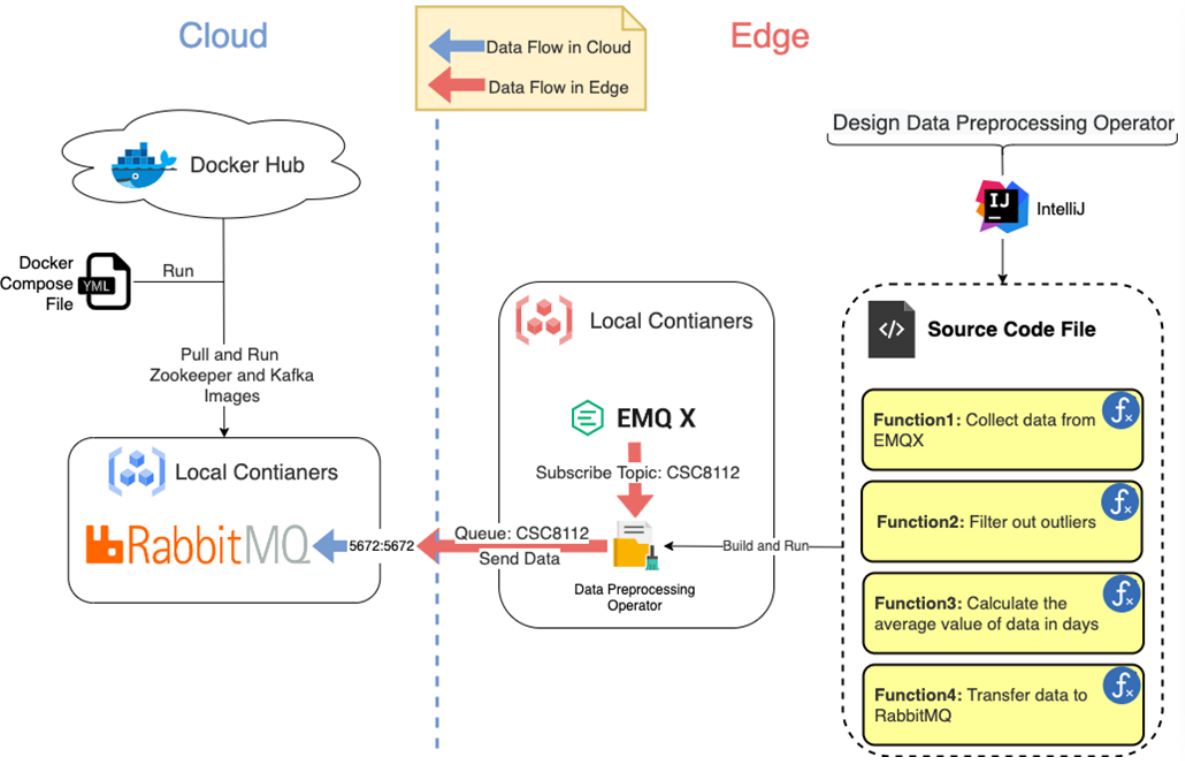
# Task 1: structure




1. Pull and run the Docker image "emqx/emqx" from Docker Hub in the virtual machine running on Azure lab (Edge). Perform this task using the command line interface (CLI).
2. Develop a data injector component with following functions (Code) in Azure Lab (Edge) or the Azure Lab localhost:
   (a) Collect data from Urban Observatory platform by sending HTTP request with url ([http://uoweb3.ncl.ac.uk/api/v1.1/sensors/PER_AIRMON_MONITOR1135100/data/json/?starttime=20220601&endtime=20220831]), and please print out the raw data that you get in console.
   (b) Filter out unnecessary data (only leave PM2.5(a metric of air quality) data with metrics (Timestamp and Value)), and please print out all PM2.5 data in console.
   (c) Send all PM2.5 data to EMQX service of Azure lab (Edge).

# Task 2

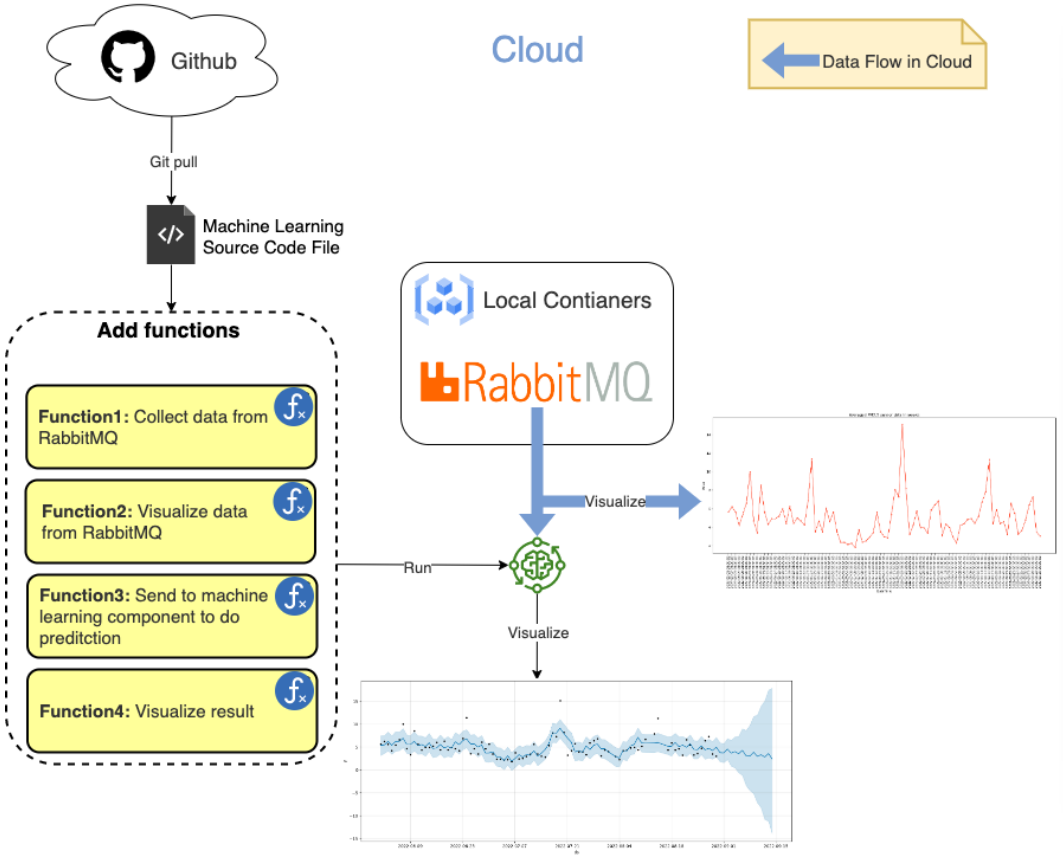
# Task 2: structure




1. Define a Docker compose file which contains the following necessary configurations and instructions for deploying and instantiating Docker images on Azure lab (Cloud):
   (a) Download and run RabbitMQ image (rabbitmq:management) in one replica;
2. Design a data preprocessing operator with following functions (code) in Azure Lab (Edge):
   (a) Collect all PM2.5 data from EMQX service, and please print out the PM2.5 data in console(docker logs console).
   (b) Filter out outliers (values greater than 50), and please print out outliers in console(docker logs console).
   (c) Calculate the averaging value of PM2.5 data in days, and please print out the result in console(docker logs console).
   (d) Transfer all results (averaged PM2.5 data) into RabbitMQ service on Azure lab (Cloud).
3. Build your "Data preprocessing operator" into Docker image, and then run it locally on the Azure lab (Edge).

# Task 3

# Task 3: structure




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 following functions (code) in Azure Lab (Cloud) or the Azure Lab localhost:
   (a) Collect all averaged PM2.5 daily data from RabbitMQ service, and please print out them in 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 in 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 trend of PM2.5 for next 15 days (a default setting of ML engine).
   (e) Visualize predicted results from Machine Learning engine, directly display the figure or save as it a file (pre-defined in ML engine).

# Task 4

# Task 4

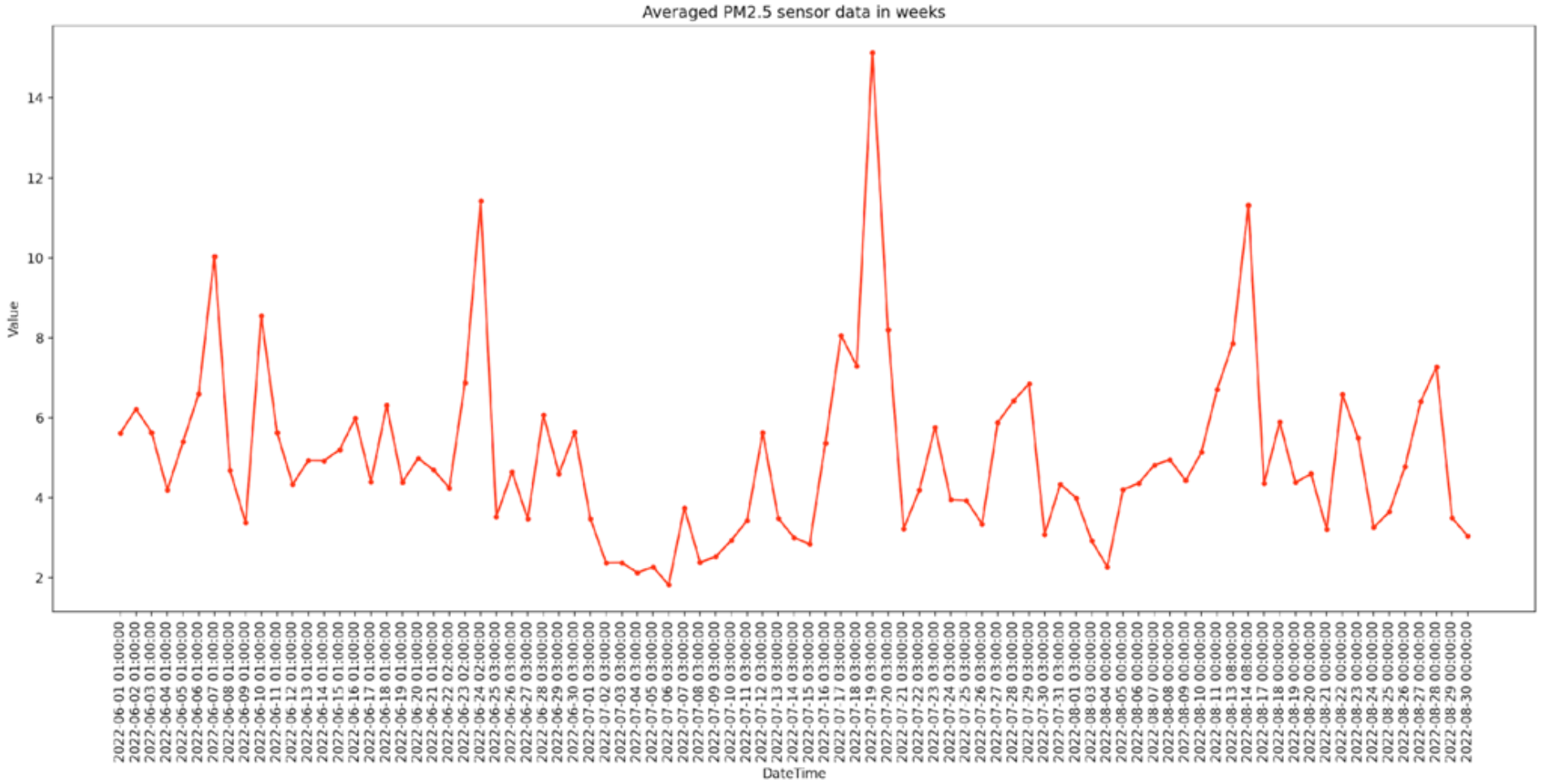
## Report

Prepare the Final Report in plain English. The report should consist of:

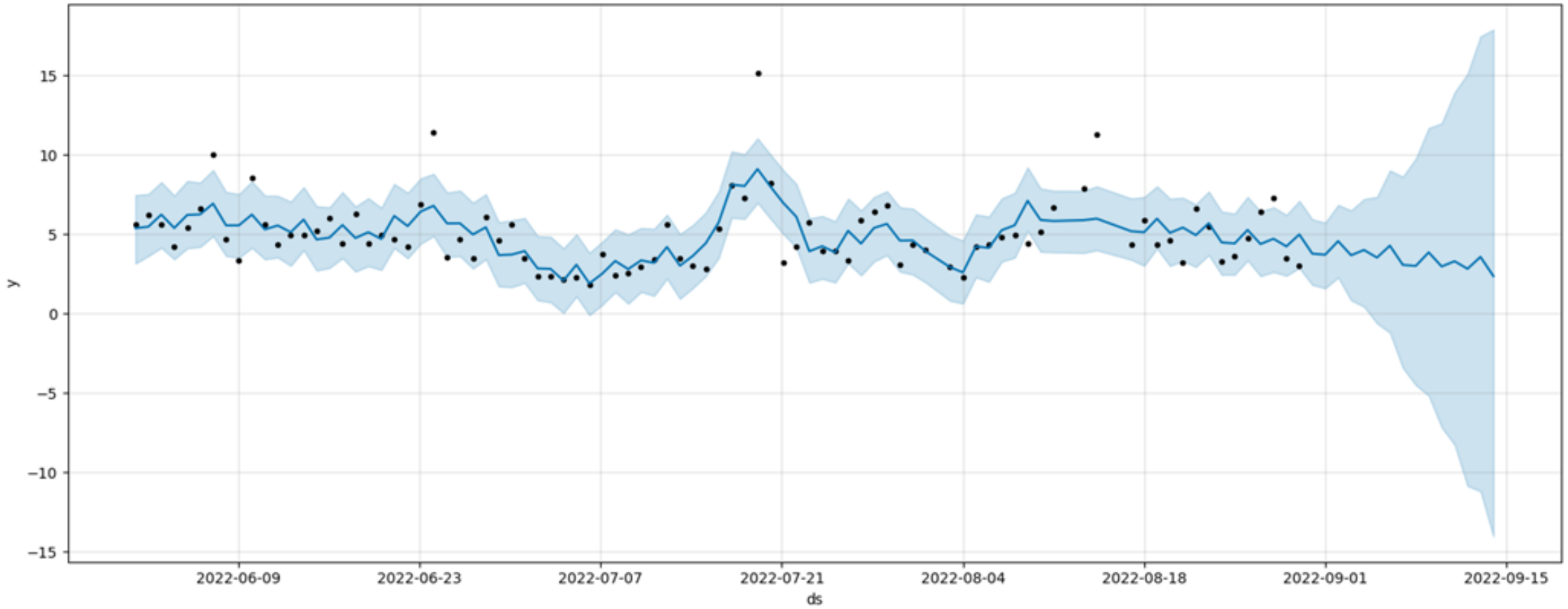
1. Detailed response to each task and related sub-tasks.
2. Screenshots of running services in the Docker Environment.
3. Screenshots of Code Snippets and/or Docker console;
4. Plots of data and prediction results by using Matplotlib.
5. Discussion of the results and related conclusions.

# Expected Results of Figures

# Visualization – Averaged PM2.5 Data

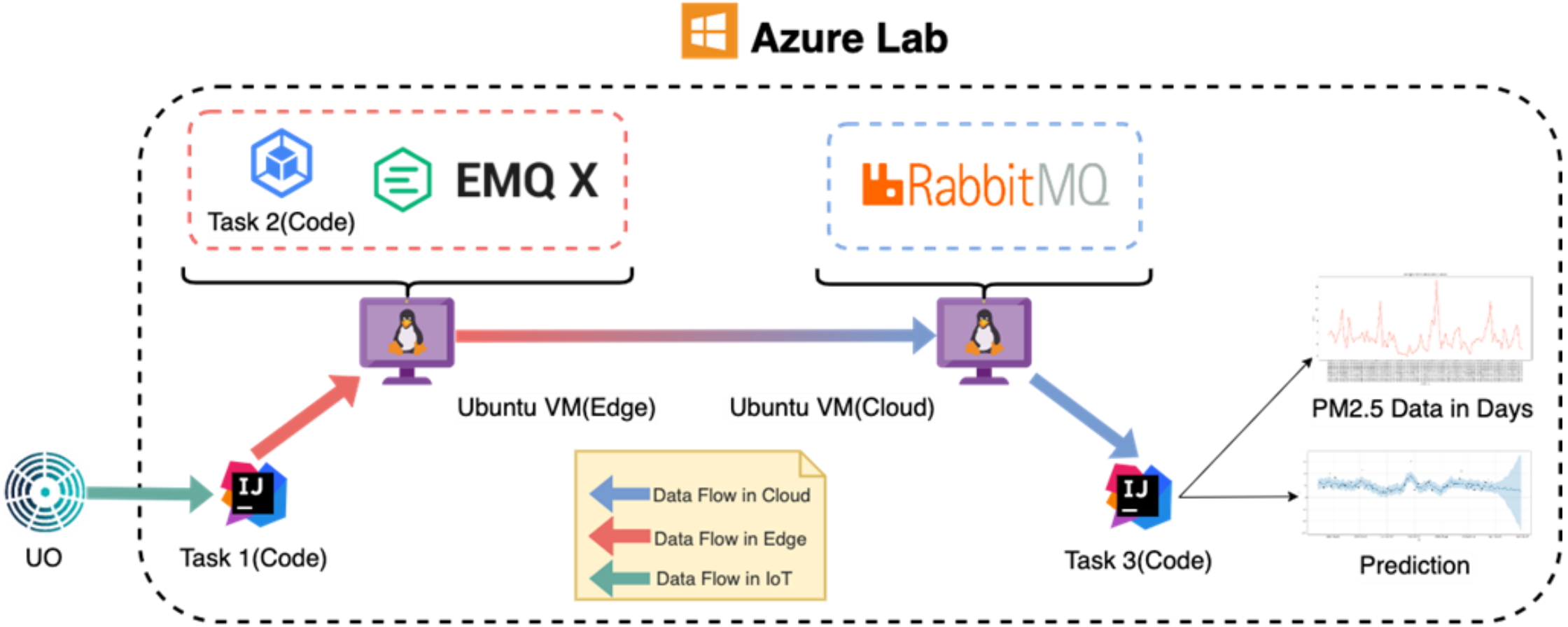


# Visualization - Prediction



# Full Workflow in Azure Lab

# Summarized structure




Newcastle University

# Thanks

## For Your Listening