CPC357 ASSIGNMENT 2

IOT ARCHITECTURE AND SMART APPLICATIONS

SCHOOL OF COMPUTER SCIENCE

UNIVERSITI SAINS MALAYSIA

SEMESTER 1 2025/2026

ASSIGNMENT 2

Lecturer: Dr. Mohammad Nadhir Ab Wahab

Group Members:

Name	Matric No
HARINETHIRAN A/L NAGARETINAN	22300638
ARUNRAJ A/L ARMUGAM	22300686

Submission: 18th January 2026

Table of Contents

1. Introduction

The **Real-Time Weather Monitoring System** is an Internet of Things (IoT)-based one which is aimed at recording and reporting on the weather conditions as they are. The system offers proper and dependable real-time information concerning vital weather conditions, namely, temperature, humidity and precipitation. This information is needed in a variety of practical uses such as agricultural planning, urban infrastructure management and disaster preparedness. The system is economical and effective because it involves **using the DHT11 sensor** of temperature and humidity and a **rain sensor** of precipitation, which will be used to monitor the weather in real-time.

The system combines these sensors and cloud services to allow **real-time transmission of data and processing**. Through this integration, it is made possible such that the data recorded by the physical sensors which will be part of weather data is relayed in a smooth manner to a centralized platform where **data can be stored** and **analyzed and visualized**. The scalability and reliability of the systems can be promoted with the help of the cloud-based integration and the monitoring station is able to process the growing data and to provide the same performance and accessibility.

The increasing need in real time weather details to make informed decisions is the solution to this project. The real-time availability of temperature, humidity, and rainfall could be used in the farm sector **to identify the most efficient irrigation times** to ensure the crop is not lost due to adverse weather patterns. Urban landscape In urban setup, real time weather data can **help in controlling drainage, flood control and planning of infrastructure**. Also on-time weather information is useful in emergencies and disaster planning and response situations where the prompt availability of the correct information can enhance preparedness and response efficiency.

The system emphasizes the combination of software and hardware materials so as to deliver effective and viable functionality. The hardware layer is tasked with the responsibility of sensing and gathering weather data and the software layer is the one that transmits, stores and analyses data using cloud-based services. Such a layered architecture provides a clear way of communication among system components, as well as allows expansion in the future. Other sensors or capabilities can be added with little modification to the current system.

All in all, the Real-Time Weather Monitoring System shows that IoT types of technologies can inspire effective interconnections between physical sensory devices and online platforms to provide timely and practical weather data. An effective, scalable and real-time solution to the contemporary weather surveillance, the system can provide insight into agile and proactive decision-making in various fields.

2. High-Level System Architecture Diagram

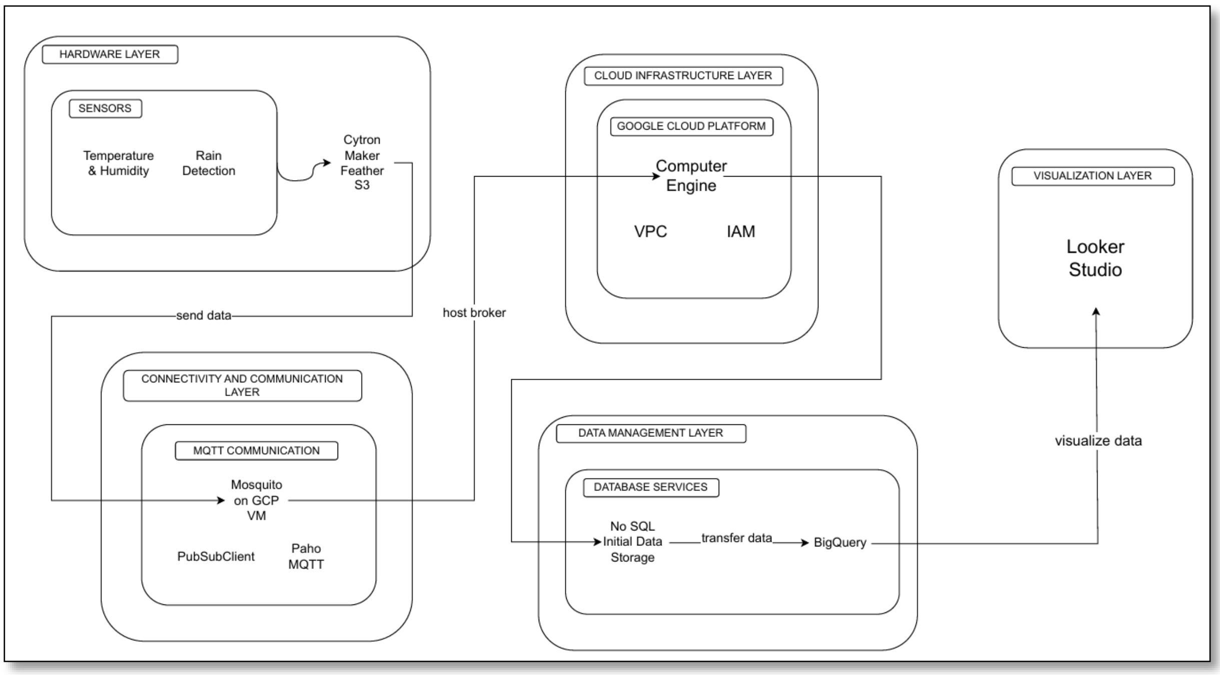


Figure 1: Shows High-Level Architecture Diagram of the System

2.1 Hardware Layer

The hardware layer is the basic part of the **Real Time Weather Monitoring System** and includes the actual IoT devices engaged in the gathering of environmental data. The main unit of this layer is a **Cytron Maker Feather S3 microcontroller** that will serve as a central unit of data collection and transmission. The two sensors fitted in this microcontroller can be used to measure the following parameters of the environment:

- **DHT11 Sensor:** This sensor is tasked with the operation of measuring the temperature and the humidity of the environment. It offers digitally and is characterized by simplicity and low cost so that it can be employed in IoT applications.

- **Rain Sensor:** This is a sensor that measures the falling of rain and sends live information about rain. It is able to predict the occurrence and the severity of rain thus the system can be used to keep track of weather conditions.

These sensors are then collated into the raw environmental data which is sent to the microcontroller which sends it to the cloud where it is further processed and analyzed. The choice of effective and reliable hardware components will provide correct data collection, and this will be the foundation on the overall functionality of the system.

2.2 Connectivity & Communication Layer

The connection and communications layer is an essential part of the Real Time Weather Monitoring System that will be in charge of the safe and efficient relay of sensor information (IoT devices) to the cloud framework. Some of the important technologies that are used in this layer include the following:

- **Mosquitto MQTT Broker:** The Mosquitto MQTT broker is deployed on a virtual machine of Google Cloud Platform (GCP) Compute Engine. It is the central messaging point, where the data between the server and IoT devices are published and subscribed to.
- **MQTT Protocol:** MQTT (Message Queuing Telemetry Transport) protocol is implemented to provide the opportunity to transmit the data with lightweight and efficiency. It is also very much applicable in IoT applications because it does not require high bandwidth. Transport Layer Security (TLS) is applied to port 8883 to achieve data integrity and security when the data is being transferred.
- **PubSubClient Library:** At the microcontroller communication, PubSubClient library is used to create MQTT communication. It allows the microcontroller to broadcast the sensor data obtained to the MQTT broker.
- **Paho MQTT Library:** The Paho MQTT library enables communication with the MQTT server on the server side to enable the system to subscribe or get incoming sensor data that can be further used.

2.3 Cloud Infrastructure Layer

The cloud infrastructure level is to be used to put up and run the critical services needed in the Real Time Weather Monitoring System. It uses a number of services provided by Google Cloud Platform (GCP) to provide scalability, security, and reliability:

- **Compute Engine**: To support data communication in real-time, a Compute Engine virtual machine which is hosting the Mosquitto MQTT broker is the high-performance, customizable computing environment.
- **Virtual Private Cloud (VPC)**: VPC service is used to provide a safe and autonomous network in the system. It handles the communication of the network and implements a firewall policy, which helps to avoid the unauthorized access of MQTT broker and other cloud resources.
- **Identity and Access Management (IAM)**: IAM is implemented in order to create appropriate access control and permissions in the system. BigQuery has been configured with a dedicated service account that has the privileges of the BigQuery Admin role allowing authorized management and querying of the data warehouse as well as improving the global system security.

This is used as an infrastructure layer that offers secure, scalable and reliable cloud environment that supports continuous data transmission, processing and storage of the weather monitoring solution.

2.4 Data Management Layer

A data management layer takes care of storage, processing and formatting of sensor data retrieved by the IoT devices in the Real Time Weather Monitoring System. This layer entails the following components:

- **MongoDB**: Raw sensor data are first of all saved into a database in MongoDB on a Compute engine virtual machine. Being a NoSQL database, the MongoDB database is flexible in servicing unstructured and semi-structured data. This acts as a short-lived storage measure as it allows the effective ingestion and manipulation of real-time sensor information.

- **BigQuery**: MongoDB sensor data is then sent to a data warehouse service provider, BigQuery, which is a fully managed and highly scaled service that belongs to Google Cloud Platform (GCP). BigQuery allows quick and effective querying and analysis of large volumes of data, which is why it could be applied to process and analyze a lot of data produced by the system.

This multi-layered data management strategy provides good real-time data ingestion, as well as, long-term storage and high volume analysis of weather data.

2.5 Visualization Layer

The visualization layer has a role of presenting the sensor data that has undergone the processing process in the Real Time Weather Monitoring System in a manner that is intuitive and insightful. This is done through the help of Looker Studio, an efficient business intelligence software that happens to be a product of Google Cloud Platform (GCP):

- **Interactive Dashboards**: The interactive dashboards are created with the help of Looker Studio and allow people to visualize and analyze the information related to weather in a proper manner. These dashboards provide an interactive way of searching through trends, patterns, and insights on the sensor data.

- **Integration to BigQuery**: Can be done seamlessly. Looker Studio is compatible with BigQuery, which can be accessed on the fly and has the processed information on the data warehouse. The visualizations in this integration are dynamic and automatically updated with the ingestion of new data to provide timely and correct insights.

Through the Looker Studio, the system provides actionable intelligence and a holistic outlook of the weather conditions to the benefit of improving informed decision-making by the stakeholders.

Generally, this Real Time Weather Monitoring System architecture is a well-organized and impressive IoT, which uses various services of Google Cloud Platform (GCP) to gather, transmit, store, process, and visualize the environmental data. Through a layered architecture, espoused in terms of hardware, connection and communication, cloud infrastructure, data management, and visualization layers, the system is scalable, secure and flexible. This design allows providing useful pieces of information regarding weather, which means that it is a powerful and effective tool in real-time monitoring of the environment.

3. Design Consideration

3.1. Factors Influencing Design Decisions

3.1.1 Sensor Selection

DHT11 Sensor: DHT11 sensor was chosen because it is cheap, easy to use and has enough accuracy to measure temperature and humidity. It delivers a digital signal, which is particularly advantageous in situations with the IoT when the low cost and reliability of data obtained is the requirement. The DHT11 does not compete with more sophisticated sensors like the DHT22 in terms of accuracy; however, since the sensor is needed to fulfill basic weather monitoring tasks, it provides sufficient results to be considered a cost-effective solution.

Rain Sensor: The reason why the rain sensor was selected was because of its ability to give versatile outputs of digitized and analogous values to enable it gauge the occurrence and intensity of rainfall. Although it might not be as accurate as professional rain gauges, its flexibility, ability to be easily integrated, and affordability will make it applicable to the real-time weather conditions that the system needs.

3.1.2 Hardware Selection

Cytron Maker Feather S3 Microcontroller: This microcontroller has been chosen since it has built-in Wi-Fi and Bluetooth functions to allow it to easily connect to cloud services. Its integration with MQTT communication protocols as well as the ability to implement lightweight data processing make it work efficiently and reliably. It is a balanced blend of functionality, energy use, and integration simplicity as compared to the less advanced microcontrollers, and so it is suitable in IoT based systems like weather monitoring station.

3.1.3 Scalability

The architecture used in the system would permit the system to **expand in the future** with the ability to add new sensors such **as wind speed or air quality detectors** without making many major changes to the current setup. The services of the **Google Cloud Platform (GCP)** such as **BigQuery** and **Looker Studio** are used to guarantee that the system is able to process the

growing amount of data efficiently, as well as allow **dynamic and real-time visualization**. The scalable design offers flexibility to support the future needs and changing application requirements without the compromise of system performance.

3.1.4 Data Security

The system encompasses several **security protocols** that will guarantee that data is not compromised. **Transport Layer Security (TLS)** has been applied to MQTT to ensure protection against the transmission of data between the IoT devices and the cloud, as well as to ensure data encryption. Firewall rules are implemented with the help of a **Virtual Private Cloud (VPC)**, and the cloud resources are secured against illegal access to the network. Besides, **Identity and Access Management (IAM)** implements the **role-based access control (RBAC)** that means that authorized users and applications have access to system resources only. Even though these security techniques add more complexity to the setup, it is rather much more consistent with the best practices of IoT security and increases the overall rigor and reliability of the system greatly.

3.1.5 Connectivity & Communication

The **Mosquitto MQTT Broker** that is deployed on a **Google Cloud Platform (GCP) Compute Engine** virtual machine is an excellent solution to a reliable and centralized messaging hub of the system. Lightweight **protocol of MQTT** and **PubSubClient** library at microcontroller side and Paho MQTT library at server side protocols allow efficient and **reliable communication** between the hardware layer and the cloud infrastructure. This effective communication system helps to reduce latency and minimise bandwidth usage which are crucial conditions in real-time weather monitoring systems based on IoT.

3.1.6 Cost Effectiveness

Such **low-cost sensors** as the DHT11 sensor and the rain sensor coupled with the **pay-as-you-go** pricing model of Google Cloud Platform will keep the system affordable to a large user base. Mongodb components have been shown to easily ingest real-time data and Big Query provides large-scale data analytics without expending large amounts of money on on-premises infrastructure. The system is able to run without unwarranted spending by using cloud services that are managed and by utilizing affordable hardware. Even though the chosen components and settings have a **better balance of quality**, the trade-off between functionality, performance, and the overall cost, it can be stated that the advantage lies in more sophisticated sensors and better cloud options which might increase performance.

3.2 Compromises

3.2.1 Design Compromises Between Cost and Performance

The choice of **cost-effective sensors** like the **DHT11** and the **simple rain sensor** was taken deliberately because they are **less precise** than the expensive ones. An example is the **DHT11 which is not as accurate as the DHT22** but is **accurate enough to give real time weather conditions**. The rain sensor, on the same note, is only useful in detecting precipitation, but **not providing quantitative precipitation data** as advanced rain gauges would do. The trade-offs on these designs were necessary in order to have a cheaper real time weather monitoring system at the expense of the essential functionality of the system.

3.2.2 Usability vs. Security Considerations

The use of **advanced security settings**, which is tight **firewall rules** and **encrypted MQTT** communication, made it **more complicated to deploy** and **maintain a system**. Even though such measures are important to guarantee good protection of data, it **needs further configuration** and **technical support**, which can **promote the learning curve of users**. A choice or viewpoint in which it is clear that **security is more than ease of use depicts** the significance of **data integrity and assurance** in the **user of IoT applications**. This tradeoff on design makes sure that the system is **secure and reliable** even though this will **minimally decrease accessibility** with respect to **non-technical users.**

3.2.3 Real-Time Data vs. Long-Term Analysis

The system architecture focuses on the **real-time data corruption** with **MongoDB** and **Big Query** is used to store and **analyze data in the long run**. Even though it is more **efficient** and **scalable** in **terms of approach**, it **adds some complexity** to the data pipeline. This **trade-off** in design is explained by the fact that such system allows supporting **real-time operation** as well as **analysis of historical data** fulfilling its requirements.

3.2.4 Flexibility vs. Specialization

The choice in favor of **general-purpose components** including **Maker Feather S3** and a **rain sensor** has a tradeoff against **precision and specialization** and leads to **system flexibility**. An example of this is that implementing a specific weather station kit would be easy to integrate, however, it will **decrease the options of tailoring the system** to meet a particular use case.

4. Development Process

4.1 Setting Up Google Cloud Platform Services

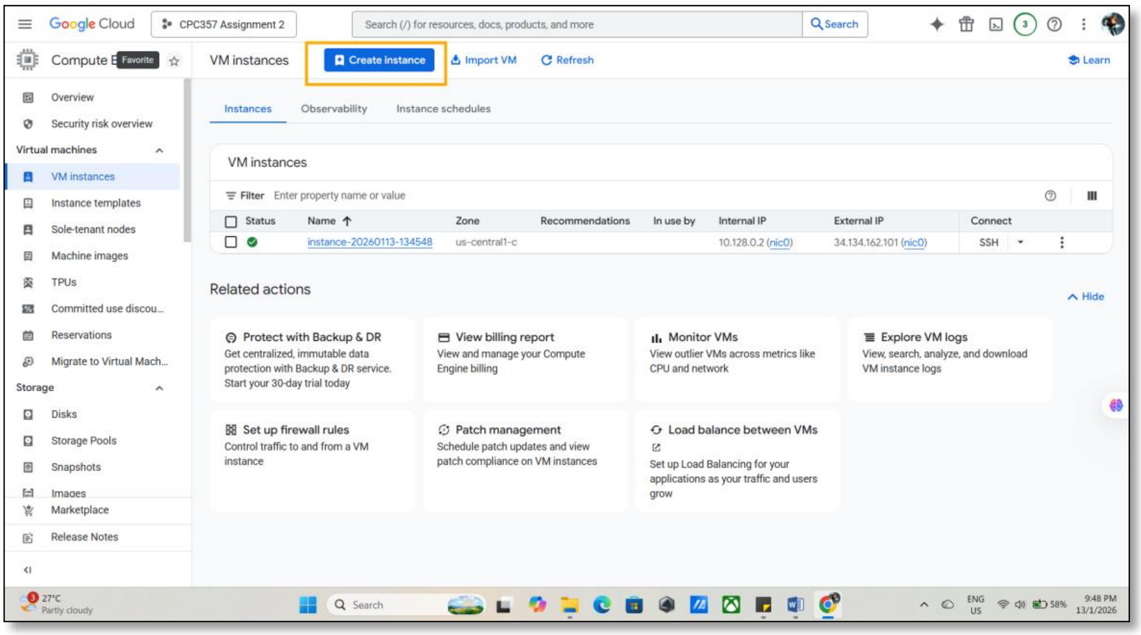
Services that are used for the development:

- Compute Engine (VM Instances)
- VPC Network (Firewall Rule)
- IAM & Admin (Service Accounts)
- BigQuery
- Looker Studio

4.1.1 Compute Engine

Compute Engine is a scalable compute engine enabling you to establish and operate virtual machines on the Google cloud. We have the option of having ready-to-do and pre-built configurations of VMs to start any machine quickly or we can opt to have custom types of machines and build VMs with the most required processing power and memory depending on our requirement. Click this link to enter into the Google Cloud Console. This will require a few more steps before you will be able to continue the rest of the session if you are working with Compute Engine the first time. Depending on the console, please create a project and enable the Compute Engine API. It is also important to mention that to use the service, you will also have to create a billing account.

Next, go to the navigation menu and select **Compute Engine** > **Virtual Machines** > **VM Instances.** Click the **Create Instance** button to create a new virtual machine.



VM instances

Status	Name	Zone	Recommendations	In use by	Internal IP	External IP	Connect
✓	instance-20260113-134548	us-central1-c			10.128.0.2 (nic0)	34.134.162.101 (nic0)	SSH

Related actions

Protect with Backup & DR
Get centralized, immutable data protection with Backup & DR service. Start your 30-day trial today

View billing report
View and manage your Compute Engine billing

Monitor VMs
View outlier VMs across metrics like CPU and network

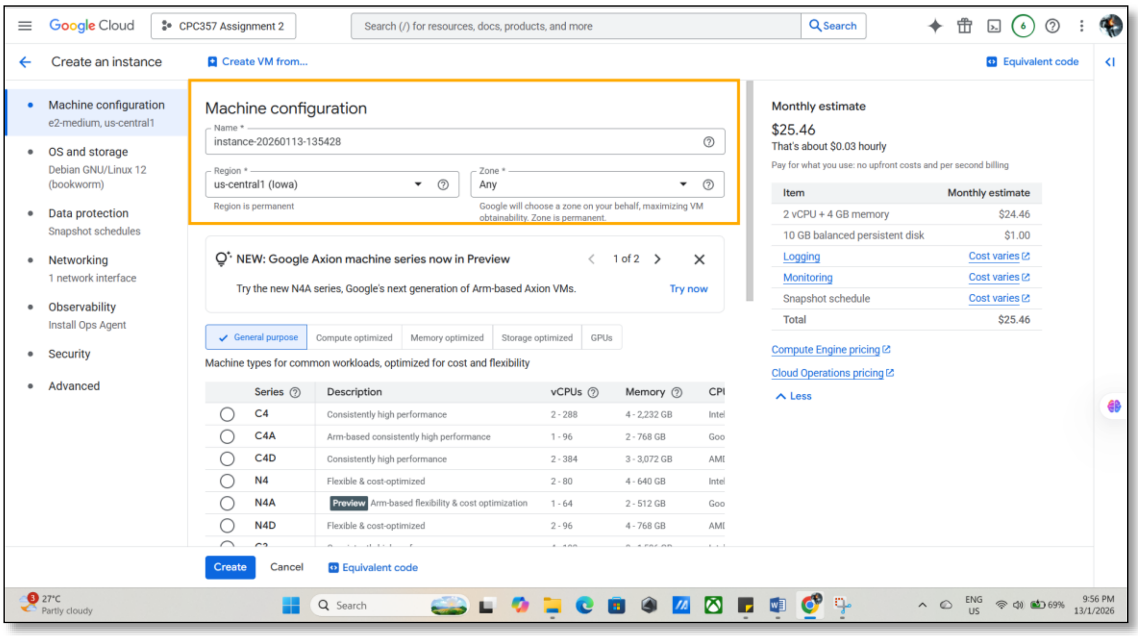
Explore VM logs
View, search, analyze, and download VM instance logs

Set up firewall rules
Control traffic to and from a VM instance

Patch management
Schedule patch updates and view patch compliance on VM instances

Load balance between VMs
Set up Load Balancing for your applications as your traffic and users grow

Under **Machine configuration**, you can set the region and zone that runs your virtual machine. Next, go to **OS and storage** in the navigation menu to choose the operating system. Click the Change button and **select Ubuntu** for the operating system and **Ubuntu 24.04 LTS** for the version.



Machine configuration

Name *: instance-20260113-135428

Region *: us-central1 (Iowa)
Region is permanent

Zone *: Any
Google will choose a zone on your behalf, maximizing VM obtainability. Zone is permanent.

NEW: Google Axion machine series now in Preview — 1 of 2
Try the new N4A series, Google's next generation of Arm-based Axion VMs. Try now

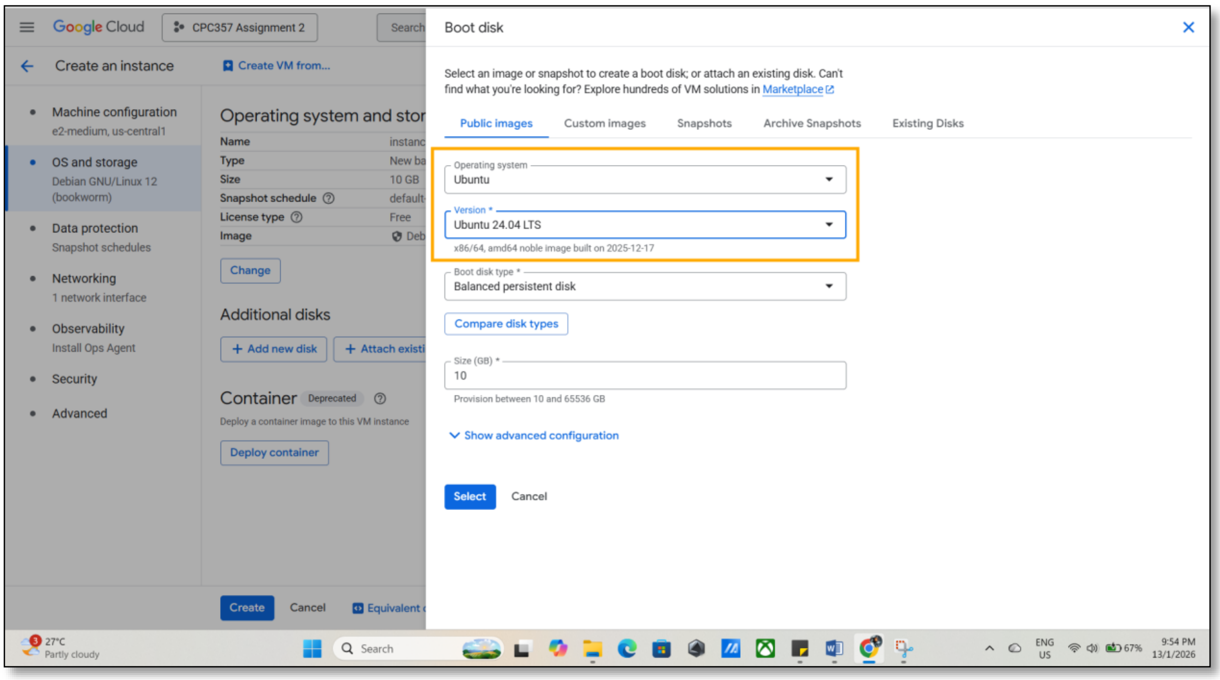
Machine types for common workloads, optimized for cost and flexibility

	Series	Description	vCPUs	Memory	CPU
○	C4	Consistently high performance	2 - 288	4 - 2,232 GB	Intel
○	C4A	Arm-based consistently high performance	1 - 96	2 - 768 GB	Goo
○	C4D	Consistently high performance	2 - 384	3 - 3,072 GB	AMI
○	N4	Flexible & cost-optimized	2 - 80	4 - 640 GB	Intel
○	N4A	Preview Arm-based flexibility & cost optimization	1 - 64	2 - 512 GB	Goo
○	N4D	Flexible & cost-optimized	2 - 96	4 - 768 GB	AMI

Monthly estimate
$25.46
That's about $0.03 hourly
Pay for what you use: no upfront costs and per second billing

Item	Monthly estimate
2 vCPU + 4 GB memory	$24.46
10 GB balanced persistent disk	$1.00
Logging	Cost varies
Monitoring	Cost varies
Snapshot schedule	Cost varies
Total	$25.46

Compute Engine pricing
Cloud Operations pricing


Boot disk

Select an image or snapshot to create a boot disk; or attach an existing disk. Can't find what you're looking for? Explore hundreds of VM solutions in Marketplace

Public images | Custom images | Snapshots | Archive Snapshots | Existing Disks

Operating system: Ubuntu

Version *: Ubuntu 24.04 LTS

x86/64, amd64 noble image built on 2025-12-17

Boot disk type *: Balanced persistent disk

Compare disk types

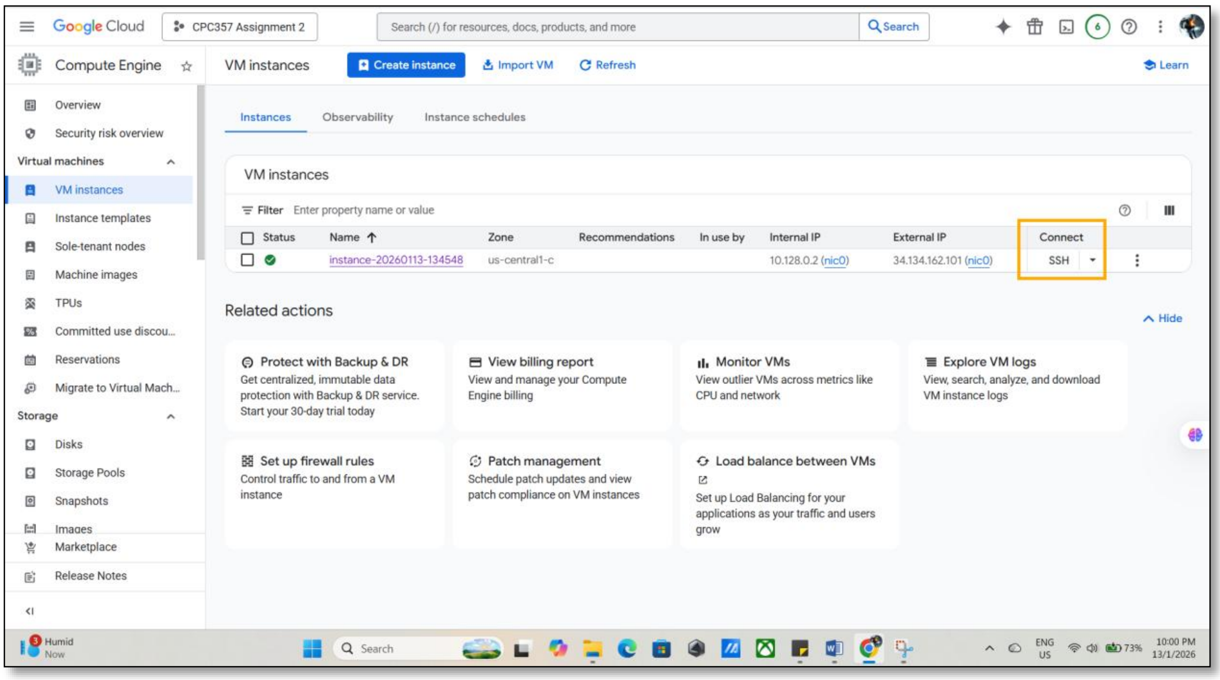
Size (GB) *: 10

Provision between 10 and 65536 GB

Show advanced configuration

Select | Cancel

Click the **Create** button to finish up the instance creation. Upon completion, you should be able to find your newly created VM instance in the list. Next, click the **SSH** button next to the VM instance to open up the SSH-in-browser.


VM instances

	Status	Name	Zone	Recommendations	In use by	Internal IP	External IP	Connect
☐	✓	instance-20260113-134548	us-central1-c			10.128.0.2 (nic0)	34.134.162.101 (nic0)	SSH

Related actions

Protect with Backup & DR Get centralized, immutable data protection with Backup & DR service. Start your 30-day trial today

View billing report View and manage your Compute Engine billing

Monitor VMs View outlier VMs across metrics like CPU and network

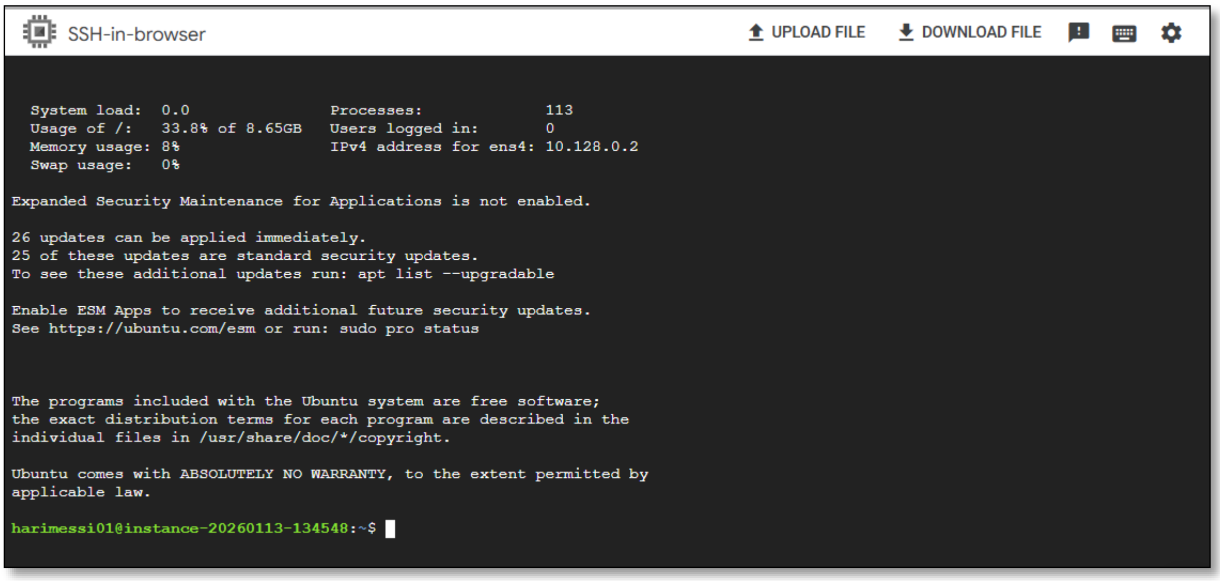
Explore VM logs View, search, analyze, and download VM instance logs

Set up firewall rules Control traffic to and from a VM instance

Patch management Schedule patch updates and view patch compliance on VM instances

Load balance between VMs Set up Load Balancing for your applications as your traffic and users grow

A pop-up window will appear and click the **Authorize** button to give permission for the **SSH-in-browser** to connect to the VM.



```
  System load:  0.0               Processes:             113
  Usage of /:   33.8% of 8.65GB   Users logged in:       0
  Memory usage: 8%                IPv4 address for ens4: 10.128.0.2
  Swap usage:   0%

Expanded Security Maintenance for Applications is not enabled.

26 updates can be applied immediately.
25 of these updates are standard security updates.
To see these additional updates run: apt list --upgradable

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status



The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

harimessi01@instance-20260113-134548:~$
```

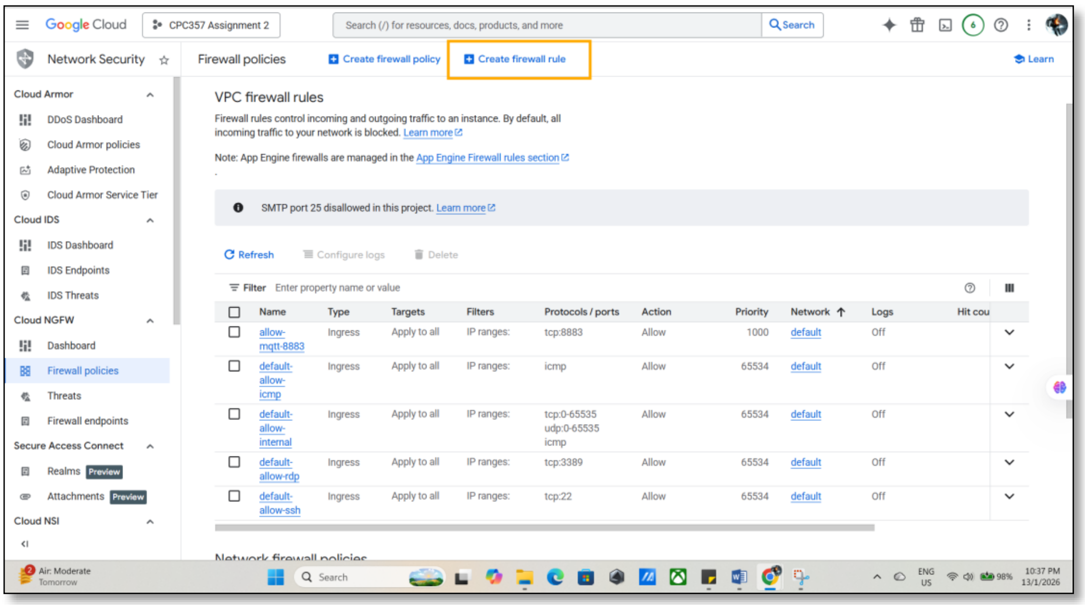
After that, run the following commands in the SSH-in-browser to setup the VM:

```
$ sudo apt-get update
```

```
$ sudo apt-get upgrade
```

4.1.2 Firewall

To provide connectivity for your Compute Engine VM instances, you need to configure the Virtual Private Cloud (VPC) network and set up the VPC firewall rules accordingly. To put it simply, VPC firewall rules let you allow or deny traffic to or from VM instances in a VPC network based on port number, tag, or protocol. First, go to the navigation menu and select **VPC Network** > **Firewall**. Then, click the **Create Firewall Rule** button at the top.



VPC firewall rules

Firewall rules control incoming and outgoing traffic to an instance. By default, all incoming traffic to your network is blocked. Learn more

Note: App Engine firewalls are managed in the App Engine Firewall rules section

SMTP port 25 disallowed in this project. Learn more

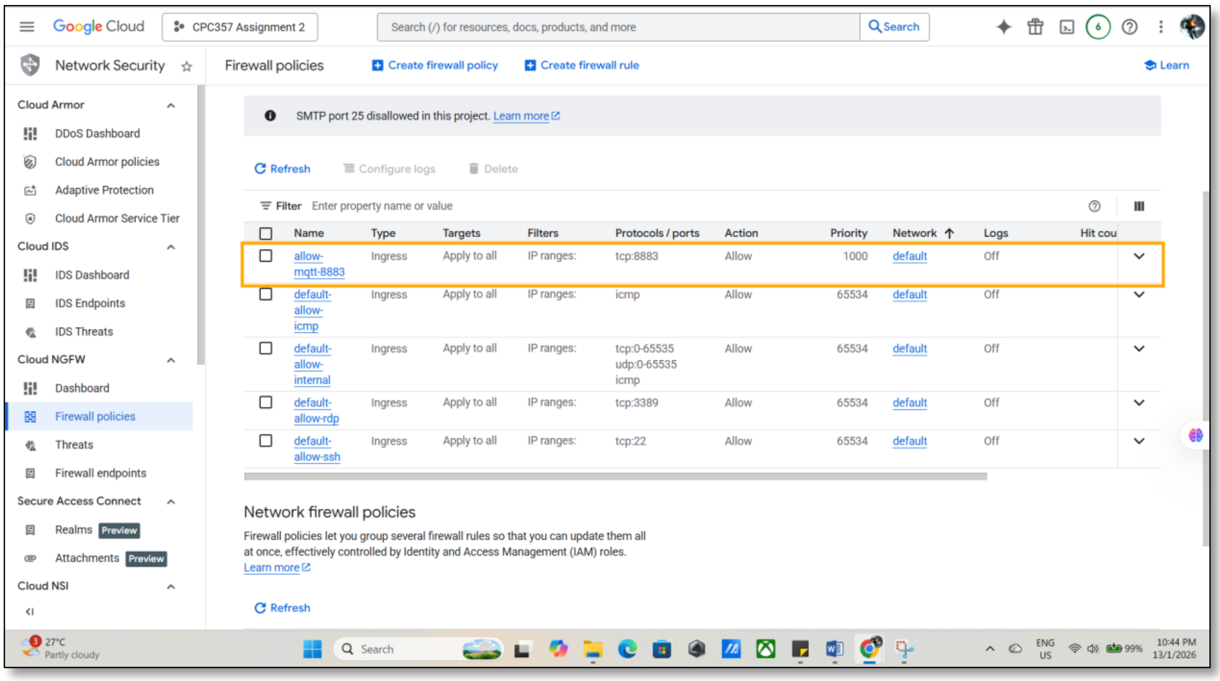
Name	Type	Targets	Filters	Protocols / ports	Action	Priority	Network	Logs	Hit cou
allow-mqtt-8883	Ingress	Apply to all	IP ranges:	tcp:8883	Allow	1000	default	Off	
default-allow-icmp	Ingress	Apply to all	IP ranges:	icmp	Allow	65534	default	Off	
default-allow-internal	Ingress	Apply to all	IP ranges:	tcp:0-65535 udp:0-65535 icmp	Allow	65534	default	Off	
default-allow-rdp	Ingress	Apply to all	IP ranges:	tcp:3389	Allow	65534	default	Off	
default-allow-ssh	Ingress	Apply to all	IP ranges:	tcp:22	Allow	65534	default	Off	

Network firewall policies

Configure the VPC firewall rule according to the specifications below:

- Name: allow-mqtt-8883
- Description: Allow traffic on port 8883
- Logs: off
- Network: default
- Priority: 1000
- Direction of traffic: Ingress
- Action on match: Allow
- Targets: All instances in the network
- Source filter: IPv4 ranges
- Source IPv4 ranges: 0.0.0.0/0
- Second source filter: None
- Destination filter: None
- Protocols and ports: Specified protocols and ports (TCP Port 8883)

Once you're done, click on the Create button to save and create the VPC firewall rule and upon completion, the newly created VPC firewall rule will be displayed in the list.



SMTP port 25 disallowed in this project. Learn more

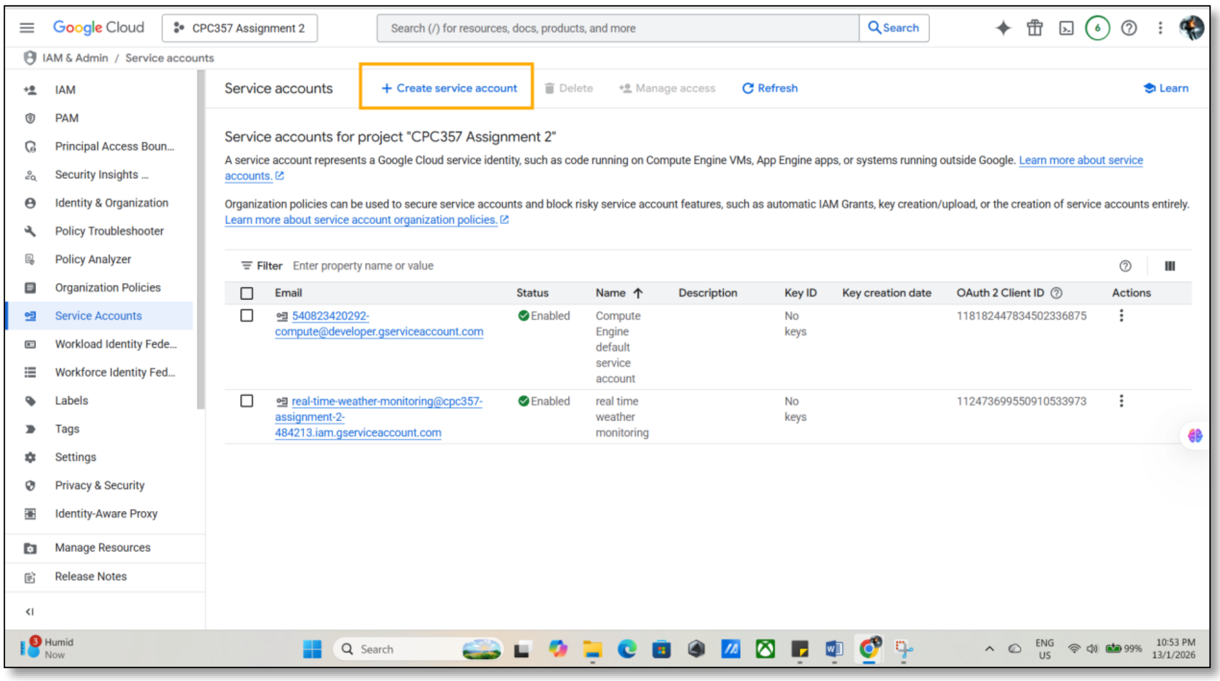
Name	Type	Targets	Filters	Protocols / ports	Action	Priority	Network	Logs	Hit cou
allow-mqtt-8883	Ingress	Apply to all	IP ranges:	tcp:8883	Allow	1000	default	Off	
default-allow-icmp	Ingress	Apply to all	IP ranges:	icmp	Allow	65534	default	Off	
default-allow-internal	Ingress	Apply to all	IP ranges:	tcp:0-65535 udp:0-65535 icmp	Allow	65534	default	Off	
default-allow-rdp	Ingress	Apply to all	IP ranges:	tcp:3389	Allow	65534	default	Off	
default-allow-ssh	Ingress	Apply to all	IP ranges:	tcp:22	Allow	65534	default	Off	

Network firewall policies

Firewall policies let you group several firewall rules so that you can update them all at once, effectively controlled by Identity and Access Management (IAM) roles.
Learn more

4.1.3 Service Account

A service account represents a Google Cloud service identity, such as code running on Compute Engine VMs, App Engine apps, or systems running outside Google. Organization policies can be used to secure service accounts and block risky service account features, such as automatic IAM Grants, key creation/upload, or the creation of service accounts entirely. Go to the navigation menu and select **IAM & Admin** > **Service Accounts**. Then, click the **Create Service Account** button at the top.

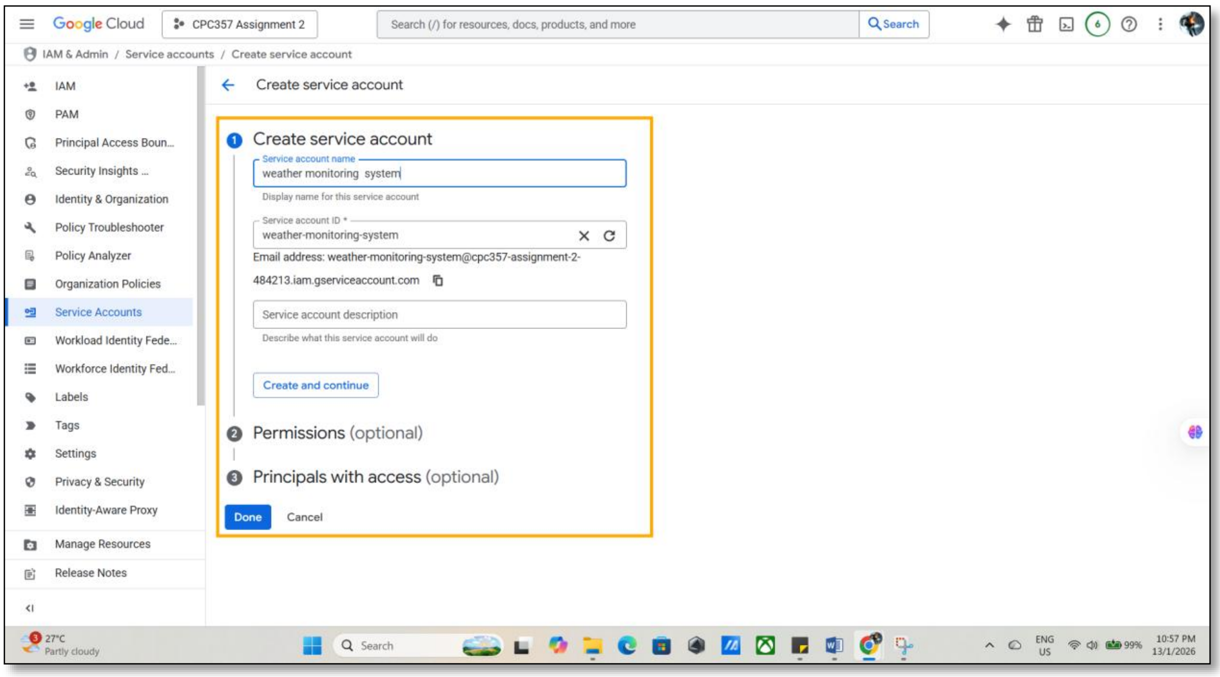

Service accounts for project "CPC357 Assignment 2"

A service account represents a Google Cloud service identity, such as code running on Compute Engine VMs, App Engine apps, or systems running outside Google. Learn more about service accounts.

Organization policies can be used to secure service accounts and block risky service account features, such as automatic IAM Grants, key creation/upload, or the creation of service accounts entirely. Learn more about service account organization policies.

Email	Status	Name ↑	Description	Key ID	Key creation date	OAuth 2 Client ID	Actions
540823420292-compute@developer.gserviceaccount.com	Enabled	Compute Engine default service account		No keys		118182447834502336875	⋮
real-time-weather-monitoring@cpc357-assignment-2-484213.iam.gserviceaccount.com	Enabled	real time weather monitoring		No keys		112473699550910533973	⋮

Under **Service account details**, give your account a name and click **Create and Continue**. Then, grant a role to your service account which is **BigQuery Data Editor** (access to edit all the contents of datasets. After that, create the service account by clicking **Done**.


Create service account

Service account name: weather monitoring system

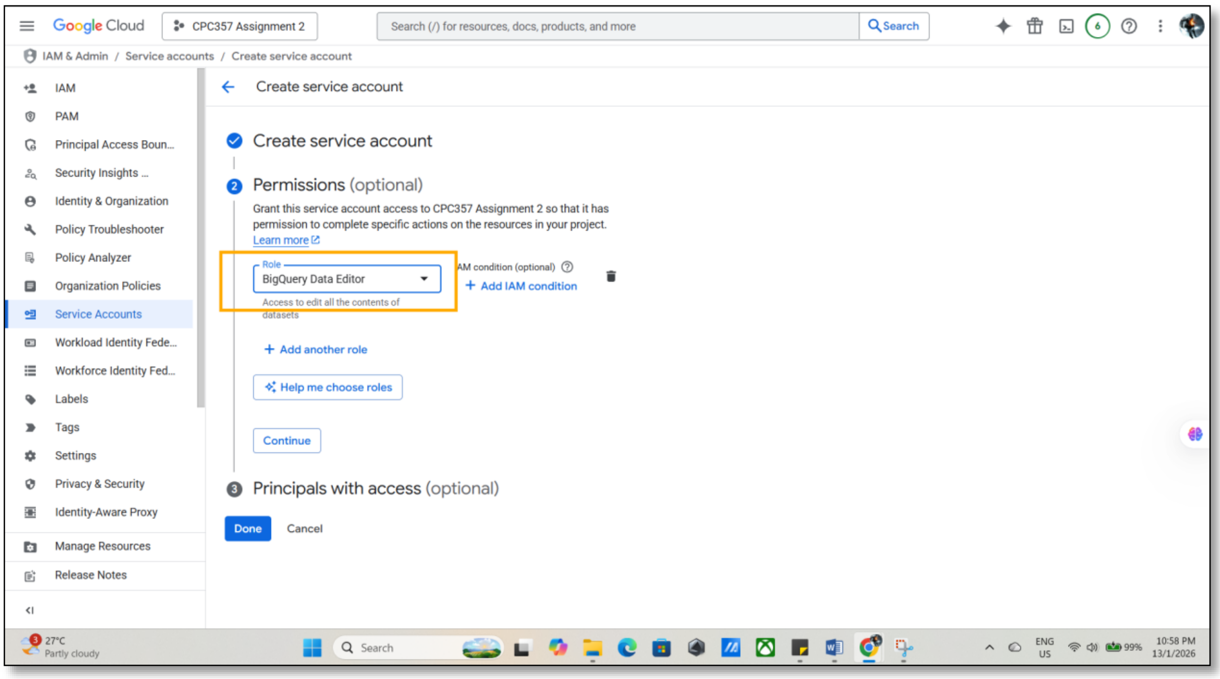
Display name for this service account

Service account ID *: weather-monitoring-system

Email address: weather-monitoring-system@cpc357-assignment-2-484213.iam.gserviceaccount.com

Service account description

Describe what this service account will do



Create service account

Permissions (optional)

Grant this service account access to CPC357 Assignment 2 so that it has permission to complete specific actions on the resources in your project. Learn more

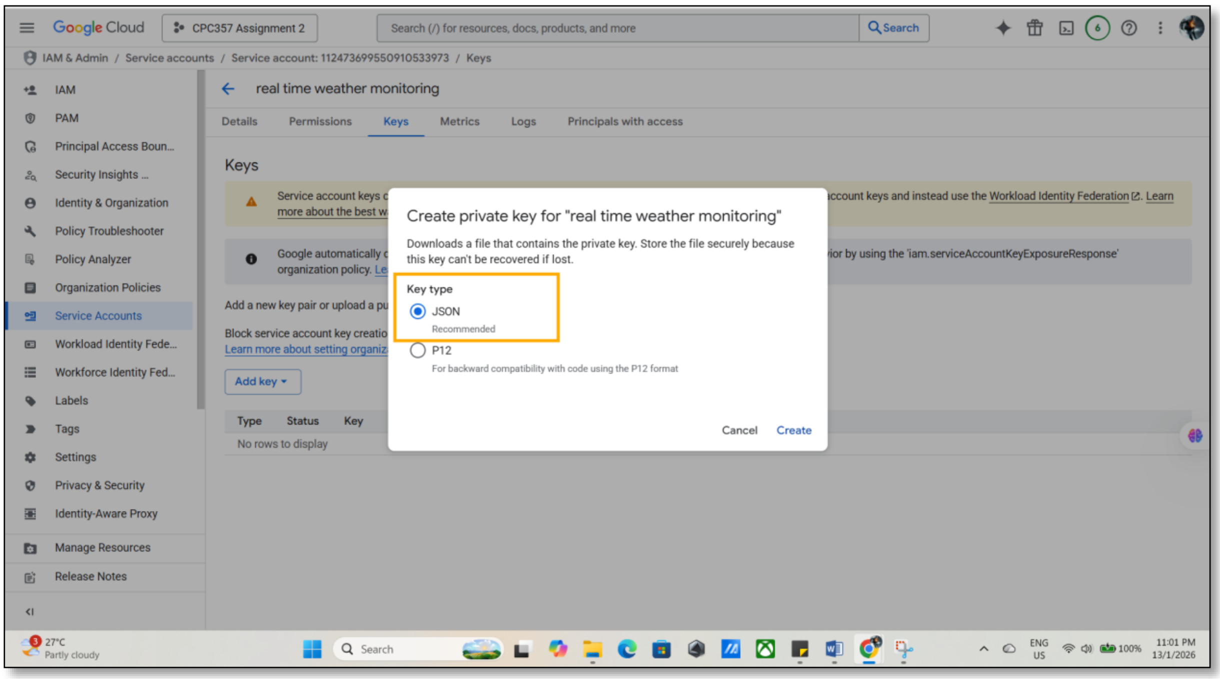
Role: BigQuery Data Editor

Access to edit all the contents of datasets

IAM condition (optional)

Principals with access (optional)

After creating the service account, the account will be displayed in the list. Click the account and navigate to **Keys** and **click Add Key** > **Create New Key**. Then, in the pop-up window, choose **JSON** as the key type and click **Create.** After clicking, a JSON file containing the service account's key will be downloaded automatically. Keep the file safe because we will use that key in our python script later.



Keys

Create private key for "real time weather monitoring"

Downloads a file that contains the private key. Store the file securely because this key can't be recovered if lost.

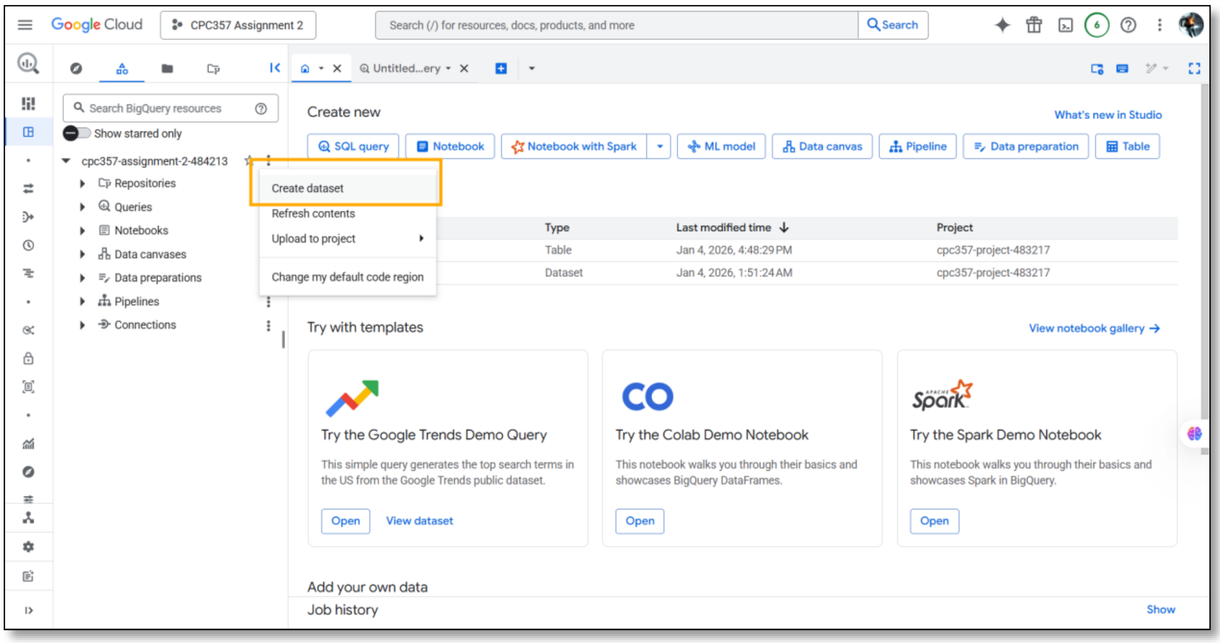
Key type

[✓ JSON] Recommended

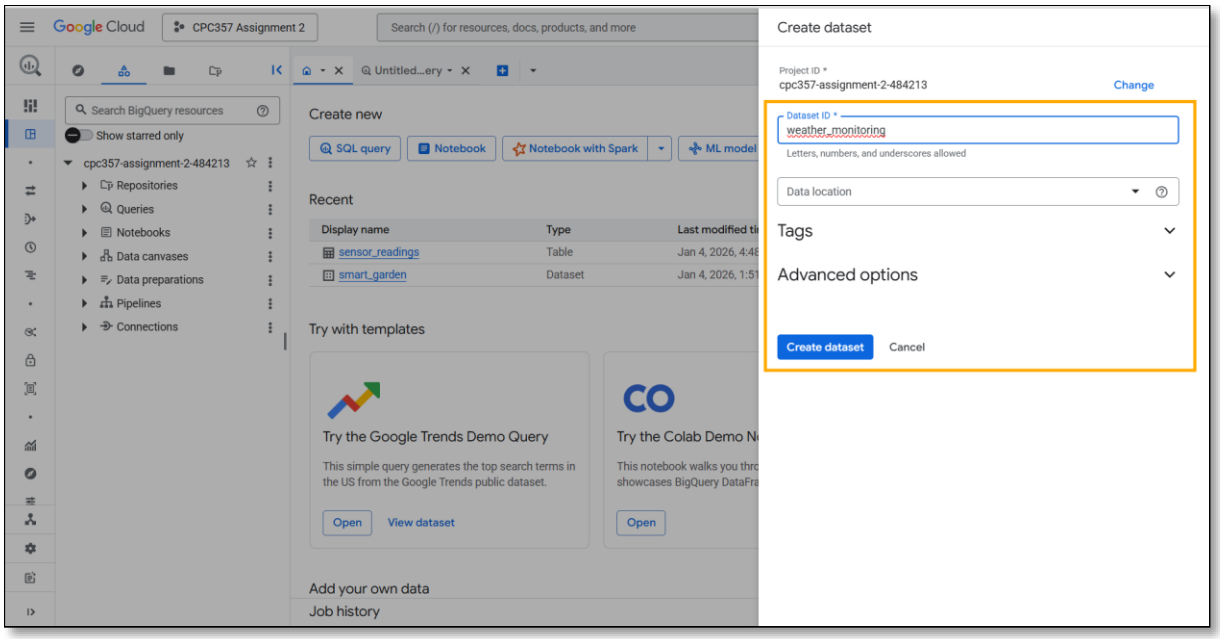
P12 For backward compatibility with code using the P12 format

4.1.4 BigQuery

BigQuery is a serverless, cost-effective, and multicloud data warehouse designed to help you turn big data into valuable business insights. Navigate to BigQuery and click **Create dataset** under your project-id. Then, give enter your **Dataset ID** and specify the location type that you want to use to store your data. After that, click **Create.**


Google Cloud
CPC357 Assignment 2
Search (/) for resources, docs, products, and more
Search
Search BigQuery resources
Show starred only
cpc357-assignment-2-484213
Repositories
Queries
Notebooks
Data canvases
Data preparations
Pipelines
Connections
Create dataset
Refresh contents
Upload to project
Change my default code region
Create new
What's new in Studio
SQL query
Notebook
Notebook with Spark
ML model
Data canvas
Pipeline
Data preparation
Table
Try with templates
View notebook gallery
Try the Google Trends Demo Query
This simple query generates the top search terms in the US from the Google Trends public dataset.
Open
View dataset
Try the Colab Demo Notebook
This notebook walks you through their basics and showcases BigQuery DataFrames.
Try the Spark Demo Notebook
This notebook walks you through their basics and showcases Spark in BigQuery.
Add your own data
Job history
Show

Type	Last modified time	Project
Table	Jan 4, 2026, 4:48:29 PM	cpc357-project-483217
Dataset	Jan 4, 2026, 1:51:24 AM	cpc357-project-483217



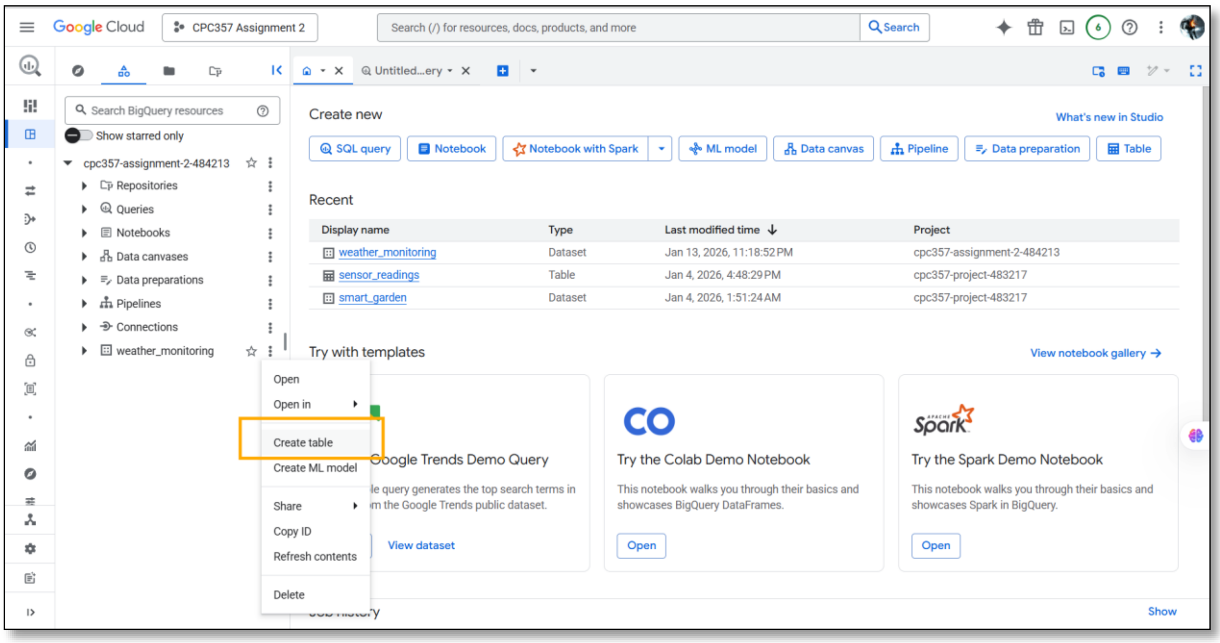
Display name	Type	Last modified ti
sensor_readings	Table	Jan 4, 2026, 4:48
smart_garden	Dataset	Jan 4, 2026, 1:51

Project ID *
cpc357-assignment-2-484213

Dataset ID *
weather_monitoring

Upon completion, **Create table** under the newly created dataset and give table a name. After that, create four fields as below table under the **Schema** section:

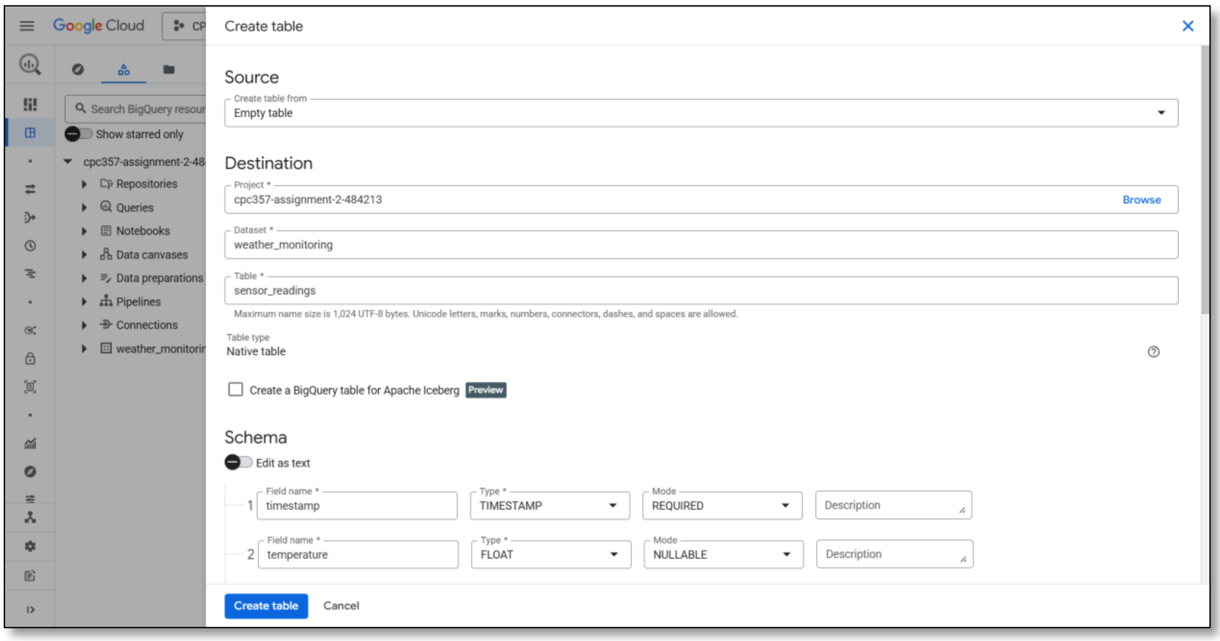
Field name	Type	Mode
timestamp	TIMESTAMP	REQUIRED
temperature	FLOAT	NULLABLE
humidity	FLOAT	NULLABLE
rain	INTEGER	NULLABLE



Display name	Type	Last modified time	Project
weather_monitoring	Dataset	Jan 13, 2026, 11:18:52 PM	cpc357-assignment-2-484213
sensor_readings	Table	Jan 4, 2026, 4:48:29 PM	cpc357-project-483217
smart_garden	Dataset	Jan 4, 2026, 1:51:24 AM	cpc357-project-483217

This notebook walks you through their basics and showcases BigQuery DataFrames.

This notebook walks you through their basics and showcases Spark in BigQuery.



Create table from: Empty table

Project *: cpc357-assignment-2-484213

Dataset *: weather_monitoring

Table *: sensor_readings

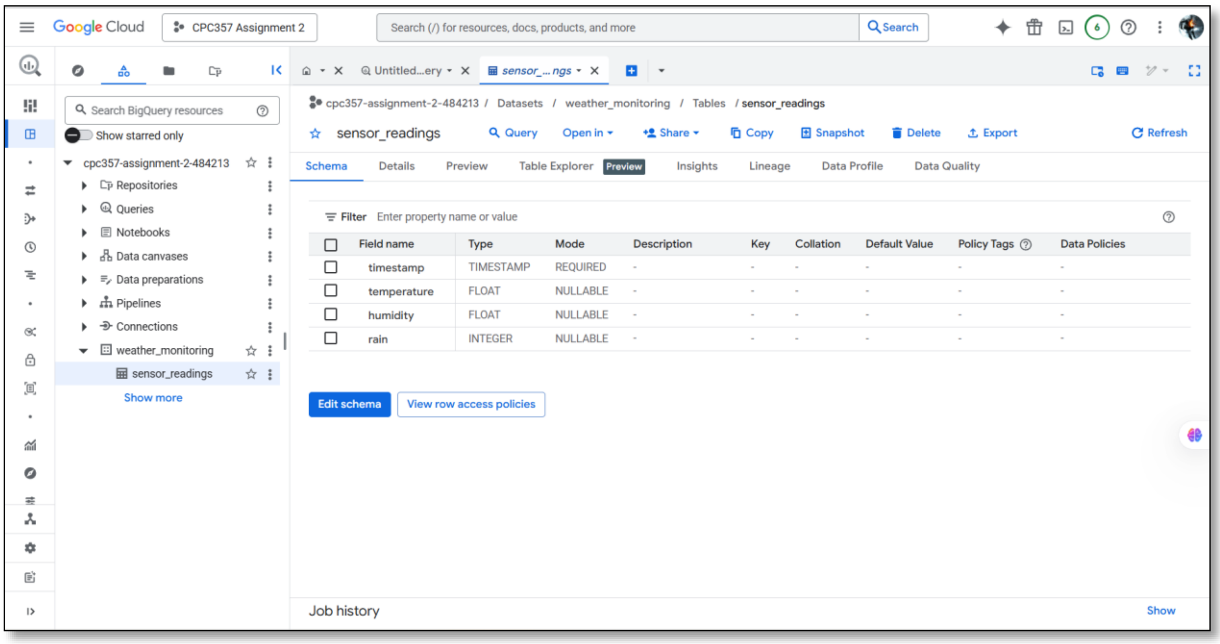
Maximum name size is 1,024 UTF-8 bytes. Unicode letters, marks, numbers, connectors, dashes, and spaces are allowed.

Table type: Native table

Create a BigQuery table for Apache Iceberg Preview

	Field name *	Type *	Mode	Description
1	timestamp	TIMESTAMP	REQUIRED	
2	temperature	FLOAT	NULLABLE	

Then click **Create Table**. Now you can see the created table with the column names and their types.



cpc357-assignment-2-484213 / Datasets / weather_monitoring / Tables / sensor_readings

sensor_readings

Field name	Type	Mode	Description	Key	Collation	Default Value	Policy Tags	Data Policies
timestamp	TIMESTAMP	REQUIRED	-	-	-	-	-	-
temperature	FLOAT	NULLABLE	-	-	-	-	-	-
humidity	FLOAT	NULLABLE	-	-	-	-	-	-
rain	INTEGER	NULLABLE	-	-	-	-	-	-

4.2 Set Up the Hardware

4.2.1 Arduino Code

Copy the code from the GitHub Repo:

https://github.com/harinethiran1306/Real-Time-Weather-Monitoring-System/blob/fa80ae496471f8addeb1b6d2ea7fc827dace89d7/CPC357_Assignment2_Arduino_Code.ino

Paste it in your Arduino IDE. Before uploading into your microcontroller, make sure you download and enable PubSubClient library in Arduino IDE.

4.3 Set Up Environment in Google Virtual Machine

Run all the following commands before creating the python script:

- **sudo apt-get install mosquito**
- **sudo apt-get install mosquitto-clients**
- **sudo apt install python3-pip**
- **$ pip install paho-mqtt**
- **sudo apt-get install -y mongodb**
- **pip install pymongo**
- **pip3 install pymongo google-cloud-bigquery paho-mqtt**

4.4 Create python script & Run it

Use the python script from GitHub Repository Link: https://github.com/harinethiran1306/Real-Time-Weather-Monitoring-System/blob/fa80ae496471f8addeb1b6d2ea7fc827dace89d7/mongo-by.py

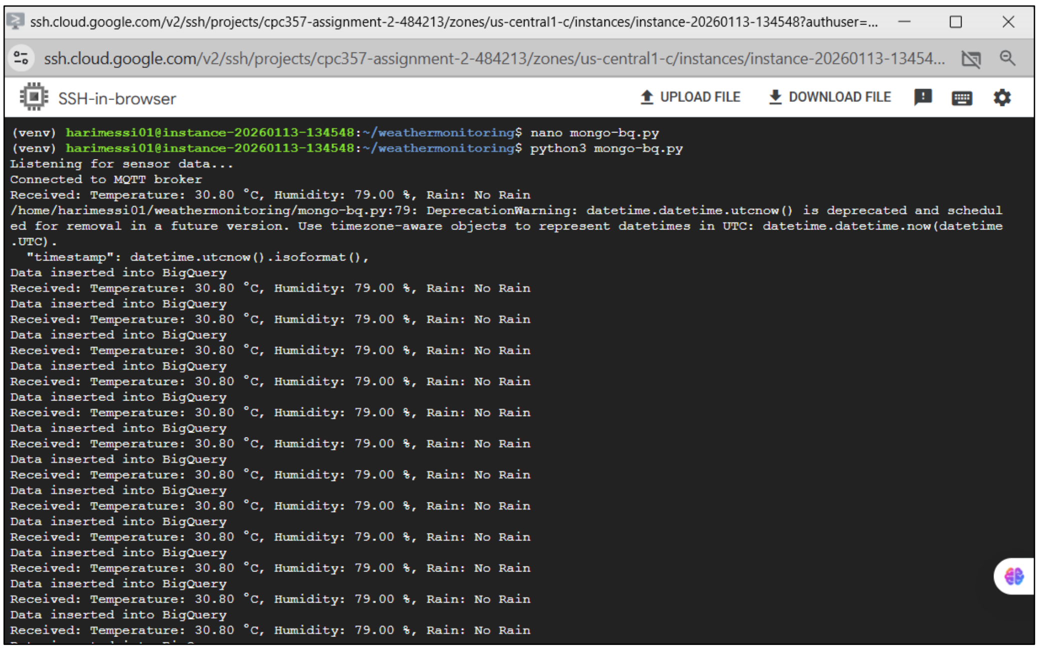
Simple follow the below command to create and run the script.

- **nano mongo-bq.py**

Then copy the code and paste into the nano editor. After that, click **Ctrl+O** to save and **Ctrl+X** to exit the nano editor. You have to include your service account key in the same directory too. Copy the content from the JSON file, create a **service-account-key.json** file using the same command that used to create the python script, and paste it into the file. Save and close the file then run the below command to execute python script.

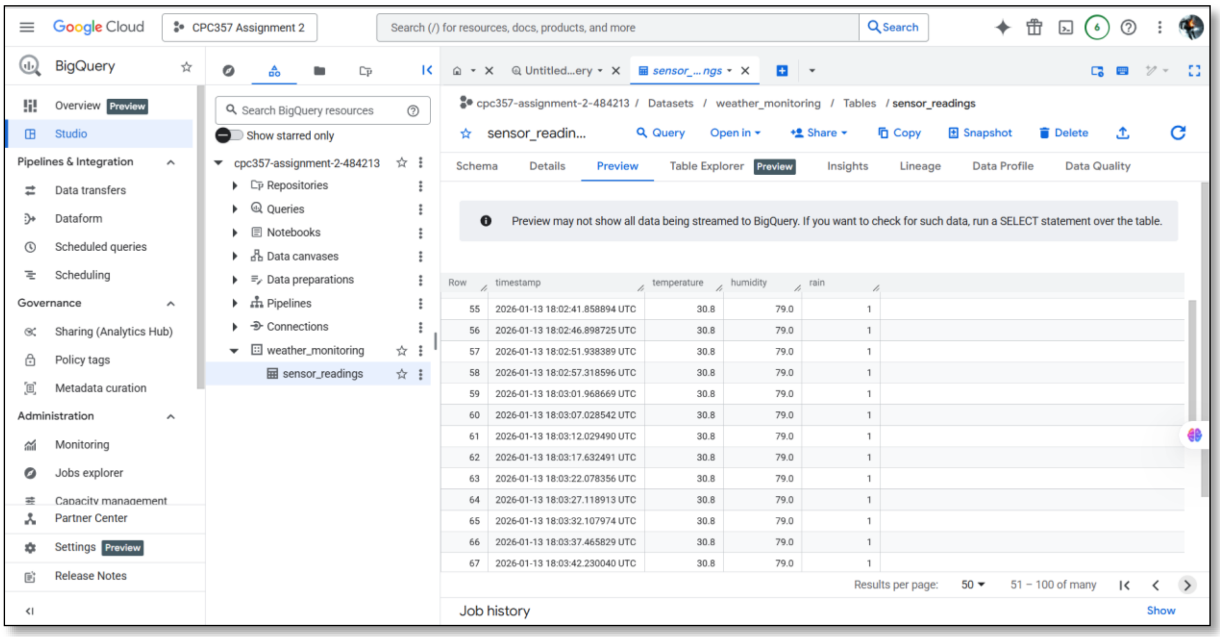
- **python3 mongo-bq.py**

You will see the below output indicating the sensor readings are successfully inserted into BigQuery table.


ssh.cloud.google.com/v2/ssh/projects/cpc357-assignment-2-484213/zones/us-central1-c/instances/instance-20260113-134548?authuser=...
ssh.cloud.google.com/v2/ssh/projects/cpc357-assignment-2-484213/zones/us-central1-c/instances/instance-20260113-13454...
SSH-in-browser
UPLOAD FILE
DOWNLOAD FILE

```
(venv) harimessi01@instance-20260113-134548:~/weathermonitoring$ nano mongo-bq.py
(venv) harimessi01@instance-20260113-134548:~/weathermonitoring$ python3 mongo-bq.py
Listening for sensor data...
Connected to MQTT broker
Received: Temperature: 30.80 °C, Humidity: 79.00 %, Rain: No Rain
/home/harimessi01/weathermonitoring/mongo-bq.py:79: DeprecationWarning: datetime.datetime.utcnow() is deprecated and schedul
ed for removal in a future version. Use timezone-aware objects to represent datetimes in UTC: datetime.datetime.now(datetime
.UTC).
  "timestamp": datetime.utcnow().isoformat(),
Data inserted into BigQuery
Received: Temperature: 30.80 °C, Humidity: 79.00 %, Rain: No Rain
Data inserted into BigQuery
Received: Temperature: 30.80 °C, Humidity: 79.00 %, Rain: No Rain
Data inserted into BigQuery
Received: Temperature: 30.80 °C, Humidity: 79.00 %, Rain: No Rain
Data inserted into BigQuery
Received: Temperature: 30.80 °C, Humidity: 79.00 %, Rain: No Rain
Data inserted into BigQuery
Received: Temperature: 30.80 °C, Humidity: 79.00 %, Rain: No Rain
Data inserted into BigQuery
Received: Temperature: 30.80 °C, Humidity: 79.00 %, Rain: No Rain
Data inserted into BigQuery
Received: Temperature: 30.80 °C, Humidity: 79.00 %, Rain: No Rain
Data inserted into BigQuery
Received: Temperature: 30.80 °C, Humidity: 79.00 %, Rain: No Rain
Data inserted into BigQuery
Received: Temperature: 30.80 °C, Humidity: 79.00 %, Rain: No Rain
Data inserted into BigQuery
Received: Temperature: 30.80 °C, Humidity: 79.00 %, Rain: No Rain
Data inserted into BigQuery
Received: Temperature: 30.80 °C, Humidity: 79.00 %, Rain: No Rain
Data inserted into BigQuery
Received: Temperature: 30.80 °C, Humidity: 79.00 %, Rain: No Rain
```

Now navigate to the BigQuery table and click the **Export > Explore with Looker Studio**. You will be redirect to Looker Studio where you can create charts according to your preference.



Preview may not show all data being streamed to BigQuery. If you want to check for such data, run a SELECT statement over the table.

Row	timestamp	temperature	humidity	rain
55	2026-01-13 18:02:41.858894 UTC	30.8	79.0	1
56	2026-01-13 18:02:46.898725 UTC	30.8	79.0	1
57	2026-01-13 18:02:51.938389 UTC	30.8	79.0	1
58	2026-01-13 18:02:57.318596 UTC	30.8	79.0	1
59	2026-01-13 18:03:01.968669 UTC	30.8	79.0	1
60	2026-01-13 18:03:07.028542 UTC	30.8	79.0	1
61	2026-01-13 18:03:12.029490 UTC	30.8	79.0	1
62	2026-01-13 18:03:17.632491 UTC	30.8	79.0	1
63	2026-01-13 18:03:22.078356 UTC	30.8	79.0	1
64	2026-01-13 18:03:27.118913 UTC	30.8	79.0	1
65	2026-01-13 18:03:32.107974 UTC	30.8	79.0	1
66	2026-01-13 18:03:37.465829 UTC	30.8	79.0	1
67	2026-01-13 18:03:42.230040 UTC	30.8	79.0	1

5. Security Consideration

5.1 Security Measure Implemented

5.1.1 Transport Layer Security (TLS) for MQTT Communication

All messages between the **microcontroller** and the MQTT broker go through encryption using **TLS over port 8883** to guarantee **data confidentiality** and prevent the occurrence of **eavesdropping and man-in-the-middle attacks** during a transmission. Such an encryption means that data packets which have been intercepted **cannot be deciphered** or **modified** correctly without authorization decryption keys. Besides that, TLS certificates also approach the client and server authentication, creating a **reliable connection** between them and avoiding communication with **unauthorized servers.**

5.1.2 Firewall Rules in Virtual Private Cloud (VPC)

Firewall policies in the **Virtual Private Cloud (VPC**) are set to **control the traffic in and out** so as to assure that only the **approved devices and IP ranges** can target the **MQTT broker** and other system resources. In order to be less exposed to vulnerable points, there is a list of ports that are opened:

- **Only encrypted MQTT communication is enabled in Port 8883.**
- **The SSH (port 22) is only limited to the particular IP addresses**.

Intense access control has a significant effect on **minimizing the attack surface** of the system and **eliminating non-authorized access** to cloud elements, which can guarantee **network access security**.

5.1.3 Identify & Access Management (IAM)

The system will take advantage of the **Google Cloud IAM** contributory to **execute role-based access control (RBAC)**, can grant each service and user the bare **minimum permissions** to execute their tasks. The **principle of least privilege** as it is applied minimizes the chance of **unintentional or malicious abuse** of resources.

Examples include:

- **Designation of the role of BigQuery Admin to service accounts that solely handle the management of the data and data queries.**
- **Limiting the end-user accounts to read-only access of the Looker Studio dashboards, excluding any possibilities of altering or deleting the data.**

The IAM policies are continuously revised in order to keep in pace with the changing requirements of the systems, thus **ensuring enhanced security** in the long run.

5.1.4 Data Encryption in Transit & at Rest

TLS has been deployed in **MQTT** communications and **HTTPS** in accessing dashboards, which helps in encrypting data during these interactions. Such encryption discourages the revelation of delicate information as data transfers between parts of the systems. Also, every piece of data in both **BigQuery** and the **MongoDB** is encrypted in place with the default **encryption of AES-256** that is provided by the Google Cloud. This is so that in case of any physical attacks on the storage devices, there will be no security risk on the data and the data will not be readable in an unipolar manner without the decryption keys. The encryption features of Google Cloud are up to industry standards that give users the system of **trust and reliability**.

5.1.5 Authentication & Authorization

There is authentication of the microcontroller, MQTT broker and cloud services using secure **API keys** and **OAuth tokens**. These tokens allow users to be sure that only **true devices** and **users** can log in to the endpoints of the system. Also, API keys are regularly changed and the lifetime of tokens is limited to ensure that they are not abused in case of a compromise. Each of the **API endpoints** is set to deny unauthenticated requests to give a structure allowing the **blocking of unauthorized admittance**.

5.2 Strategies for Data Protection & Access Control

5.2.1 Monitoring & Alerts

Real-time monitoring is established using **Google Cloud Monitoring** that monitors the activity of the system and detects abnormal patterns. Alerts are set to inform the administrators that a **possible breach** has occurred, like a request to access the information without an authorization, or an irregularity in the data flows. Logs of **Cloud Audit Logs** provide administrators with an opportunity to detect and act upon suspicious activity in a short period of time. Such constant tracking guarantees that there are **timely interventions** to reduce risks and guarantee security of data.

5.2.2 Backup & Disaster Recovery

A **powerful backup strategy** is used in the system, where key data is saved on safe places on a regular basis. This is to guarantee fast restoration of data in case they are breached or the system fails. There is also a **disaster recovery plan** to recover services in **minutes** so that there is minimum downtimes. Such measures ensure the system is not affected by **data loss** and offers operational durability in case of **unforeseen incidents**.

5.2.3 Access Control Policies

The system implements **granular access control** to provide that only **authorized devices** and authorized users can access certain resources. **Google Cloud Identity** and **Access Management (IAM)** is used to implement **role-based access control (RBAC)** to allocate roles with minimal permissions needed to do tasks. As an example, the role of the **BigQuery Admin** provides access to the data stored information to be managed, whereas **end-user accounts** are allowed to view the visualizations in **Looker Studio** only without writing something. These policies play a vital role in ensuring that there is **integrity in the system** and unauthorized access is avoided.

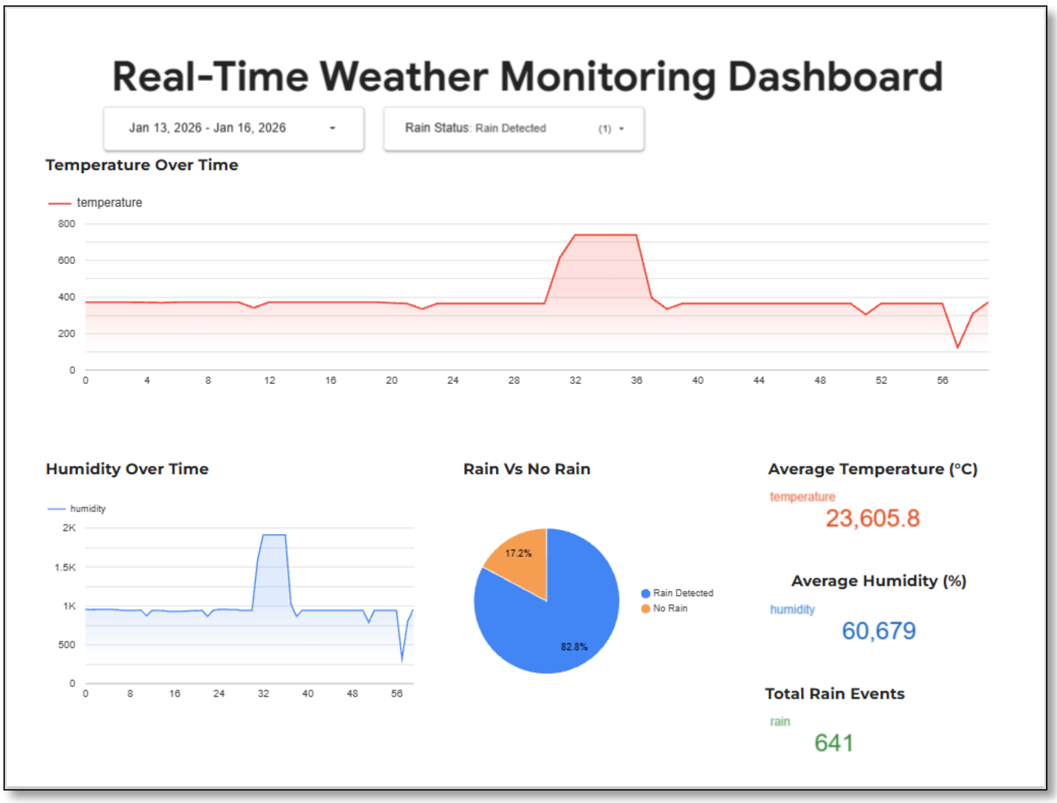
5.2.4 Regular Key Rotation

To reduce the chances of breached credentials, the system will carry out **frequent key rotation** of all **encryption keys** and **API keys.** This habit also means that in case a key is left exposed, it has a restricted time during which it would be used. Automation of this process by the system will lighten the load of the administration and improve the overall **security posture**. This is also an offensive way of ensuring **data confidentiality** and reducing the exposure to possible threats.

5.2.5 Penetration Testing & Vulnerability

Periodic **penetration testing** of the system is done to discover and eliminate vulnerabilities that may be exploited by the attackers. Automated **vulnerability scans** are also done to identify **misconfigurations,** old libraries or open endpoints. These weaknesses can be mitigated in advance to guarantee the **integrity of the data** and lower the chances of successful cyberattacks.

6. Appendix



- GitHub Repository Link: https://github.com/harinethiran1306/Real-Time-Weather-Monitoring-System.git