EE4500-CNS-2025-BL2 LAB 3 LoRaWAN Database

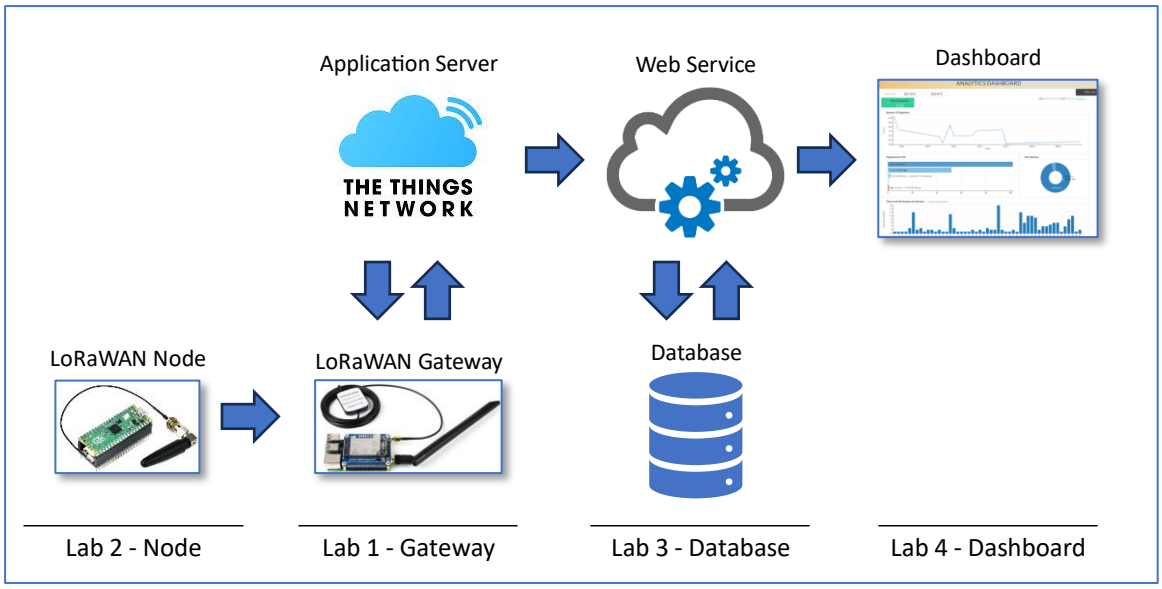
Course	25-EE4500_EE5510-CNS_TSV-CAM-BL2
Unit	Lab 3
Title	LoRaWAN Database
Version	1.5
Author(s)	Bruce Belson
Date created	1-Jun-2024
Date updated	25-Mar-2025

Contents

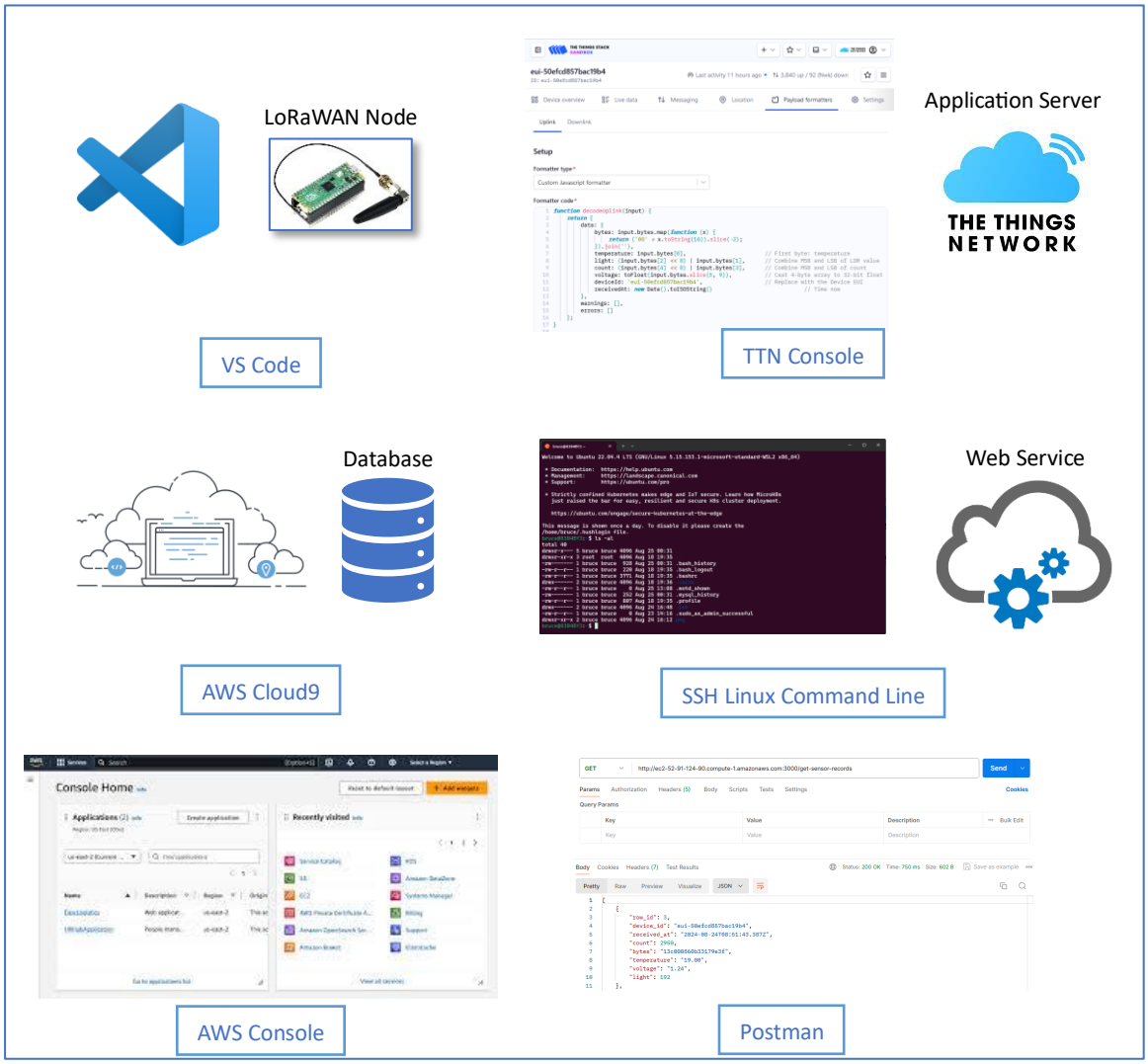
Objectives

The purpose of this lab exercise is to transmit data from a LoRaWAN Node via The Things Network to a relational database hosted on Amazon Web Services.

i. The node kit from Lab 2 is reconnected to the existing TTN application. Two new data fields – count & voltage – are added to the payload.
ii. A relational database is created on AWS.
iii. A node.js web server is created on an AWS virtual machine, to feed data to the database.
iv. An integration is set up between TTN and AWS, to transmit incoming data to the AWS web server.



This lab delivers much of the cloud infrastructure for the complete distributed application.



To develop, test and maintain a networked application requires a lot of different tools. The picture above summarises the tools we'll be using in Lab 3.

Deliverables

There are three target deliverables for the lab:

i) amended firmware for the LoRaWAN node & matching updates to the TTN application.
ii) MySQL database hosted on AWS.
iii) secure cloud-based pipeline delivering data to the database.

Deliverable 1: LoRaWAN node firmware

1. The node's firmware is updated so that the uploaded data includes two new fields, `count` & `voltage`:
    a. `count` is an unsigned 16-bit integer with an initial value of 0, which is incremented in each payload.
    b. `voltage` is a 4-byte floating-point number; it is the voltage supplied to the Pico.
2. The payload will now contain the following bytes:

Bytes	0	1-2	3-4	5-8
Value	Temperature	LDR value	count	voltage

3. The Uplink Payload Formatter of the TTN application will be updated to decode the new payload.

Deliverable 2: SQL database

The database is a MySQL relational database hosted on the AWS RDS service. Full details can be found below in ***Steps > Create a database instance***.

Deliverable 3: Pipeline from TTN to AWS

The pipeline consists of two components:

1. A webhook integration on TTN which POSTs data to a web server.
2. A node.js web server hosted on an AWS EC2 virtual Linux machine.

Websites

Raspberry Pi Pico	LearnJCU > Reading Resources > Raspberry Pi Pico
Waveshare product site	https://www.waveshare.com/wiki/Pico-LoRa-SX1262
AWS Academy Portal	As sent to you by AWS Academy
The Things Network console	https://au1.cloud.thethings.network/console
Postman	https://web.postman.co

IDs and Endpoints

Throughout this lab, you will need to note down values from various consoles and screens. Save them here for use later.

Item	Value
Cloud9 IP Address	
RDS database endpoint	
Web server public DNS	
Web server IP address	

Summary of steps

1. Update Pico firmware
2. Update TTN Uplink Payload Formatter
3. Start AWS Academy Sandbox
4. Start Cloud9
5. Set up Cloud9 environment
6. Create a database instance
7. Create a database table
8. Create a virtual machine
9. Create security group for web server
10. Connect to EC2 instance
11. Create a website
12. Test the web server
13. Add a webhook from TTN to AWS

Procedure

Update Pico firmware

1) Reconstruct the LoRaWAN node from Lab 2.
2) Update the firmware so that the uploaded data includes two new fields, count & voltage:
    a. count is an unsigned 16-bit integer with an initial value of 0, which is incremented in each payload.
    b. voltage is a 4-byte floating-point number (i.e. a C float) which contains the voltage read from adc after setting the input channel to 3, thus:
    ```
    adc_select_input(3);
    ```
3) The payload should now contain the following bytes:

Byte(s)	0	1-2	3-4	5-8
Value	Temperature	LDR value	count	voltage

Hints

1) Reading the supply voltage:
    a. Set GPIO pin 29 to input during initialisation
    b. Initialise GPIO pin 29 for ADC during initialisation
    c. Call adc_select_input(3) before calling adc_read().
    d. Apply a conversion factor of 3.3f * 3.0 / (1 << 12). Note that this is multiplied by 3.0 because the input from VSYS to ADC is divided by 3.
    e. See Raspberry Pi Pico Datasheet Section 2.1 Raspberry Pi Pinout for more information:
    `GPIO29 IP Used in ADC mode (ADC3) to measure VSYS/3`
    You can also look at the Pico schematic in the same document to see the voltage divider across VSYS in the bottom-left corner of the page.
2) In C, you can use memcpy() to copy the four bytes of the voltage into the buffer.

Update TTN Uplink Payload Formatter

1. Connect to TTN at https://au1.cloud.thethings.network/console/applications
2. Select the application you built in Lab 2.

Update your TTN application to decode the buffer to fields named as follows:

Name	Description
bytes	The raw input data as a hex array
count	Integer value in bytes 3-4 of input
deviceId	Set this from the known value of the end device's ID
light	Integer value in bytes 1-2 of input
receivedAt	The time when the formatter ran. Implemented as: `new Date().toISOString()`
temperature	Integer value in byte 0 of input
voltage	Copy of 4-byte (32-bit) float in bytes 5-8 of input

Hints

Delete the application-level payload formatter, and instead create a sensor-level payload formatter.

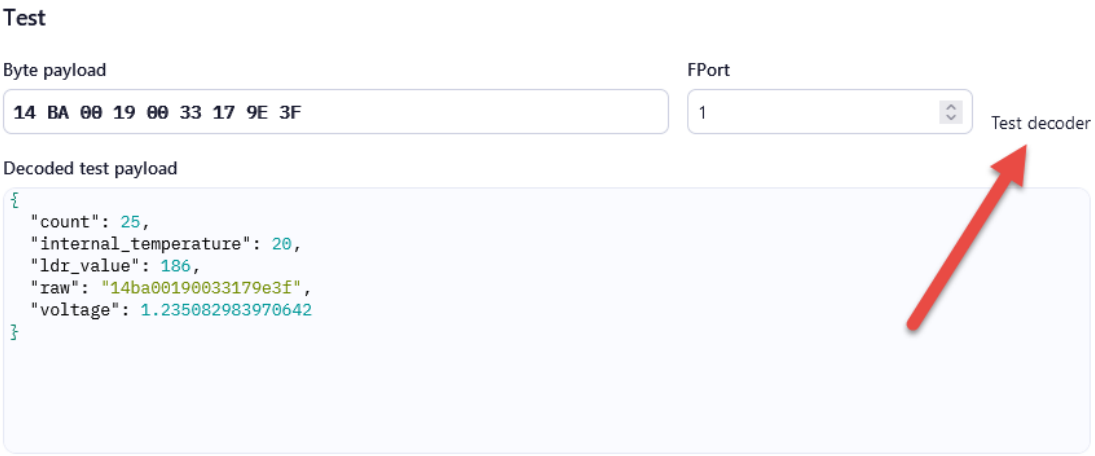
Consider the hints below as you amend the code of your payload formatter.

1) When testing your formatter code, first capture a hex value that you can use in the Byte payload of the Test panel, by updating the formatter as follows:

```
function decodeUplink(input) {
  return {
    data: {
      raw: input.bytes.map(function (x) {
                return ('00' + x.toString(16)).slice(-2);
              }).join('')
    },
    warnings: [],
    errors: []
  };
}
```

2) You can then copy the hex string from the decoded_payload field in the Live data display

3) Then paste the string into the test panel of the uplink payload formatter. Test your code using the panel until you get the expected decoded payload, e.g.:

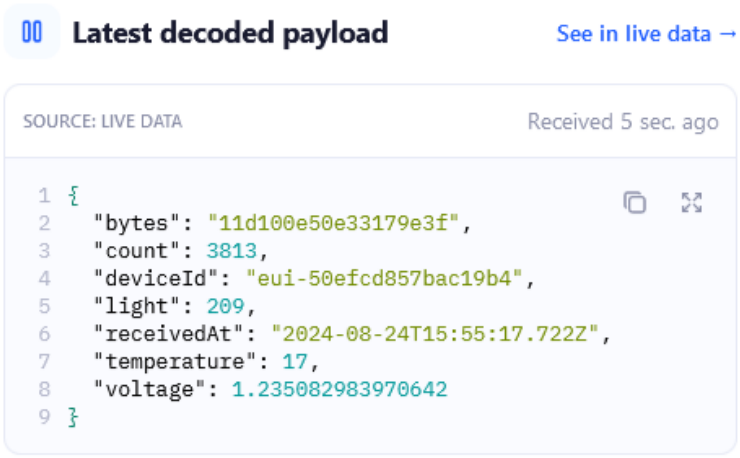


4) Hint: In Javascript, you can use the function below to convert a byte array to a float:

```
function toFloat(bytes) {
  const buffer = new ArrayBuffer(4);
  const f32 = new Float32Array(buffer);
  const ui8 = new Uint8Array(buffer);
  bytes.forEach(function (b, i) {
    ui8[i] = b;
  });
  return f32[0];
}
```

5) You can use the Javascript slice() function to copy the last four bytes of the array, thus:
`toFloat(input.bytes.slice(5, 9))`
6) If you create a formatter for the end device (rather than for the application) you can simply set the value of the device, e.g.:
`deviceId: 'eui-50efcd857bac19b4',`

Test that your application now has a decoded payload resembling the following:



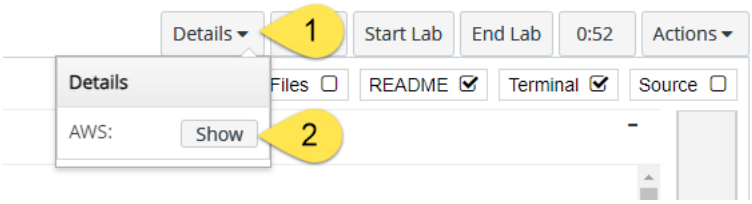
Start AWS Academy Sandbox

1. Log in at https://www.awsacademy.com/
2. Select course AWS Academy Cloud Foundations [114129]
3. Select ***Modules***
4. Scroll down and select ***Sandbox Environment***
5. Click ***Load Sandbox Environment in a new window***.

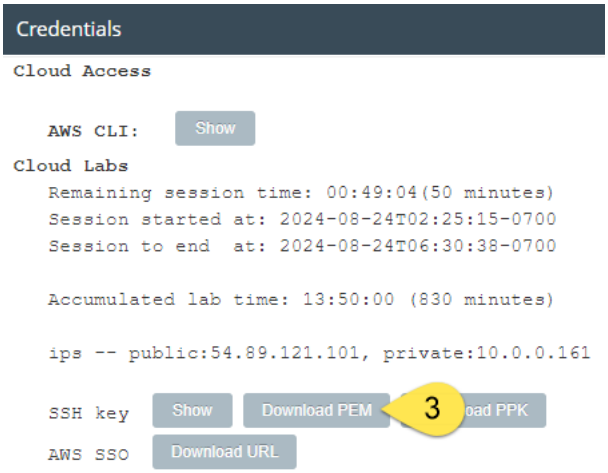
6. In the Lab page, click ***Start Lab*** at the top of the right-hand side.
7. Work elsewhere until Lab status moves from `in creation` to `ready`.

Once the Lab has started, save your credentials:

1. Return to the *Academy Workbench – Vocareum* window. Close the status popup.
2. Click ***Details > AWS: Show***



3. In the ***Credentials*** window, click ***Download PEM***.



4. Check that the file `labsuser.pem` has been downloaded to the `Downloads` folder on the PC.
5. Click ***AWS*** in the workbench.

Start Cloud9

1. Select ***Cloud9*** in the console.
2. Click ***Create environment***.
3. Enter the settings below.

Name	lab3
Description	Test and dev environment
Environment type	New EC2 instance
New EC2 instance	Default values
Network settings > Connection	Secure Shell (SSH)

4. Click ***Create***.
5. In the Environments list, wait for a message `Successfully created lab3`, then click the ***Open*** link next to `lab3`.

Set up Cloud9 environment

1. Find the bash window at the bottom of the IDE.
2. Type in the following three commands to install the MySQL tools:

```
sudo yum install -y \
    https://dev.mysql.com/get/mysql57-community-release-el7-11.noarch.rpm
```

```
sudo yum install -y mysql-community-client
sudo rpm --import https://repo.mysql.com/RPM-GPG-KEY-mysql-2022
```

3. Get the Cloud9 virtual machine's IP address by typing:
   `ifconfig`
   In the `enX0` section, find the value next to `inet` and save it as ***Cloud9 IP Address***.

Create a database instance

1. From the AWS console ***Services*** menu, open ***EC2***.
2. On the list at the left, select ***Network & Security > Security Groups***.
3. Click ***Create security group*** and set the fields as below:

Security group name	db
Description	Allow access to MySQL database
VPC	Default
Inbound rule	Click ***Add rule***
Type	MYSQL/Aurora
Protocol	TCP
Port range	3306
Source	Custom - The Cloud9 IP address found above, followed by /32
Description	Cloud9 IP address

4. Click ***Create security group***.
5. In the Services menu, open ***Aurora and RDS***.
6. Click on ***Create database*** and enter fields as below:

Attribute	Value
Creation method	Standard Create
Engine options	MySQL
Templates	Free tier
Instance identifier	database-1
Master username	admin
Credentials management	Self managed
Password	Cheesewh1z!
Instance configuration	db.t3.micro
Storage type	General purpose SSD (gp2)
Allocated storage	20 GiB
Compute resource	Don't connect to an EC2 compute resource
VPC	Default VPC
DB subnet group	Default
Public access	Yes
VPC security group	Choose existing
Security group name	db
Availability zone	No preference
Additional configuration > Database port	3306
Database authentication	Password authentication
Additional configuration > Database options > Initial database name	sensor_data

7. Click ***Create database***.
8. Wait for a while until the message `Creating database database-1` changes to `Successfully created database database-1`.
   (Avoid any attempts at up-selling during the wait.)
9. Now click the ***View connection details*** button and inspect the dialog. Make a copy of the endpoint and save it, as ***RDS database endpoint***.

Create a database table

1. Return to the console window in Cloud9 and select the Bash panel again.
2. Start the mysql client as follows, substituting the RDS database endpoint found above:
   mysql -h <endpoint> -P 3306 -u admin -p'Cheesewh1z!'
3. You should see the mysql command CLI as below:

```
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MySQL connection id is 26
Server version: 8.0.35 Source distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MySQL [(none)]>
```

4. Paste the SQL code you can find on the subject website in `ddl.sql`.
5. Test the database with the following SQL code:
   ```
   INSERT INTO devices_data (device_id, received_at, count, bytes, voltage,
     light) values ('device_1', NOW(), 1, '001122334455667788', 3.6, 192);
   ```
6. Inspect the result with:
   ```
   SELECT * FROM devices_data;
   ```
7. You should see a single record. Repeat steps 5 & 6 above.
8. Clean up the table as follows:
   ```
   DELETE FROM devices_data;
   ```

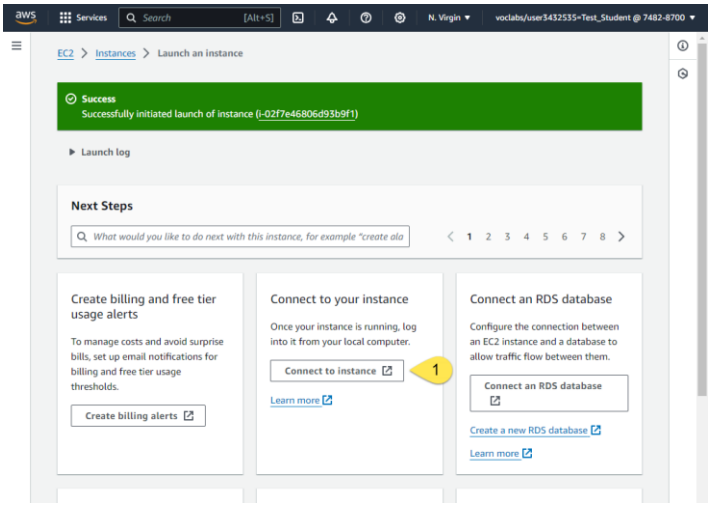
Create a virtual machine

1. In the AWS console use ***Services*** to open ***EC2***.
2. Click ***Launch instance*** and create an instance as below:

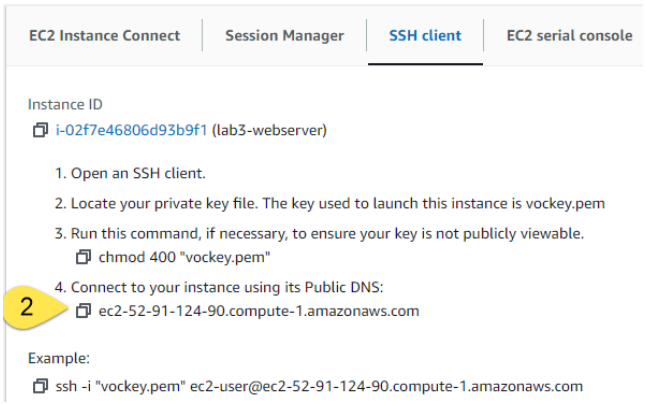
Name	lab3-webserver
Image	Amazon Linux 2023 AMI
Instance type	t2.micro
Key pair	vockey
Allow SSH traffic from	Anywhere

3. Click ***Launch instance***.

4. On the instance dashboard, click ***Connect to instance***.



5. In the new tab, select SSH client.
6. Copy the EC2 Public DNS and save it as ***Web server public DNS***.



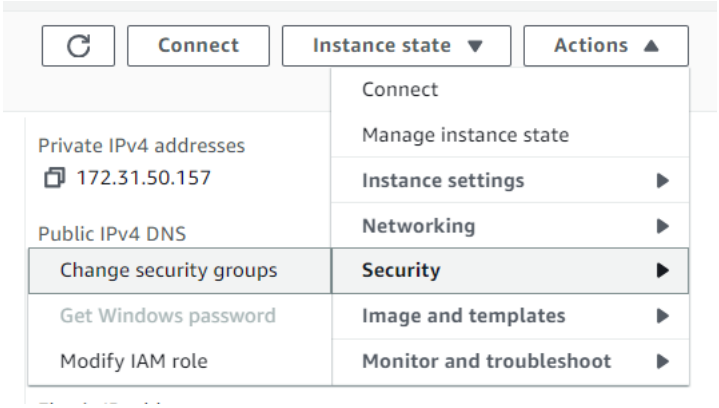
Create security group for web server

1. In AWS console, use ***Services > EC2 > Security Groups***.
2. Click ***Create security group*** and fill out the fields as below:

Security group name	webserver
Description	Allow access to EC2 Web Server
VPC	Default
Inbound rule	Click ***Add rule***
Type	Custom TCP
Protocol	TCP
Port range	3000
Source	Anywhere-IPv4
Description	Web API on Port 3000

3. Click ***Create security group***.
4. Click Instances in the left-hand menu and click on the Instance ID of lab3-webserver to open it.

5. Click ***Actions > Security > Change security groups***:



6. In the ***Associated security groups*** panel, select `webserver`.
7. Click the ***Add security group*** button.
8. Click ***Save***.

Connect to EC2 instance

Start up an SSH session from the PC to the EC2 instance.

1. On the PC, start a Windows command line session.
2. If necessary, navigate to your home folder, e.g.:
   `cd C:\Users\jc123456`
3. Start an SSH session on the EC2 instance, using the credentials downloaded earlier:
   `ssh -i Downloads\labsuser.pem ec2-user@<EC2 Public DNS>`
4. You may be asked to confirm the connection. Type Yes.

```
C:\Users\jc454533>ssh -i Downloads\labsuser.pem ec2-user@ec2-52-91-124-90.compute-1.amazonaws.com
The authenticity of host 'ec2-52-91-124-90.compute-1.amazonaws.com (52.91.124.90)' can't be established.
ED25519 key fingerprint is SHA256:[redacted]
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])?
```

5. You are now logged on to the EC2 Linux virtual machine instance.

Allow the EC2 instance – which will be hosting our web server – to connect to the database.

6. Get the EC2 virtual machine's IP address by typing:
   `ifconfig`
   In the `enX0` section, find the value next to `inet` and save it as ***Web server IP address***.
7. In AWS Console, Click ***Services > EC2 > Security Groups***.
8. In the list, click on the ID of the `db` Security Group.
9. Click ***Edit inbound rules***.
10. Add a second rule: `MYSQL/Aurora`, `TCP`, `3306`, with the Web server IP address (as found above) followed by /32. Description is: `Web server IP address`.
11. Click ***Save rules***.

Create a website

First, set up **node.js**, which we will use as the runtime for the web server:

1. At the SSH session, type the following:

```
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.7/install.sh |
bash
source ~/.bashrc
nvm install v22.7.0
node -e "console.log('Running Node.js ' + process.version)"
```

2. We should see the following at the command line:
   `Running Node.js v22.7.0`
3. Finally, type the following to download and install some essential node.js libraries:
   `npm install express mysql2 moment`

We now have the run-time environment. Next, create the website.

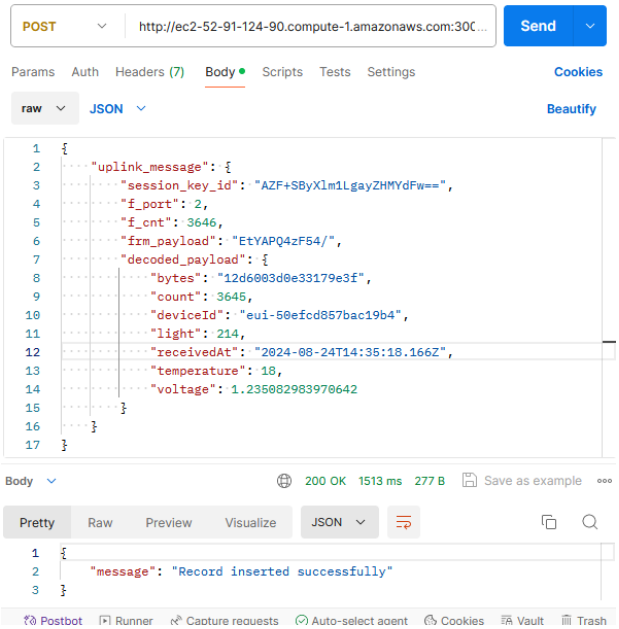
   a. Type the following commands to download the web server code:
   ```
   cd ~
   mkdir src
   cd src
   curl https://jcuengineering.com/files/cc4950/labs/lab3/app.js >
   app.js
   ```

4. Open the server code with an editor as follows:
   `nano app.js`
5. In the editor, locate the line containing:
   `host: 'insert the endpoint of the RDS database',`
   Replace the host value by the RDS database endpoint, saved in ***Create a database instance*** above, e.g.:
   `host: 'lab3-db.cjrksnjhley0.us-east-1.rds.amazonaws.com',`
6. Save changes (Ctrl+O) and exit nano (Ctrl+X).
7. Start the server:
   `node ./app.js`
8. You should see the following:
   `Server is running on port 3000`

Test the web server

1. Open the Postman app at https://web.postman.co.
2. If you don't have a free account, create one.
3. Set up a connection as follows:

Action	POST
URL	http://<Web server public DNS>:3000/insert-sensor-record
Body type	Raw, JSON
Body value	Copied from `payload.json` which can be found on the subject website



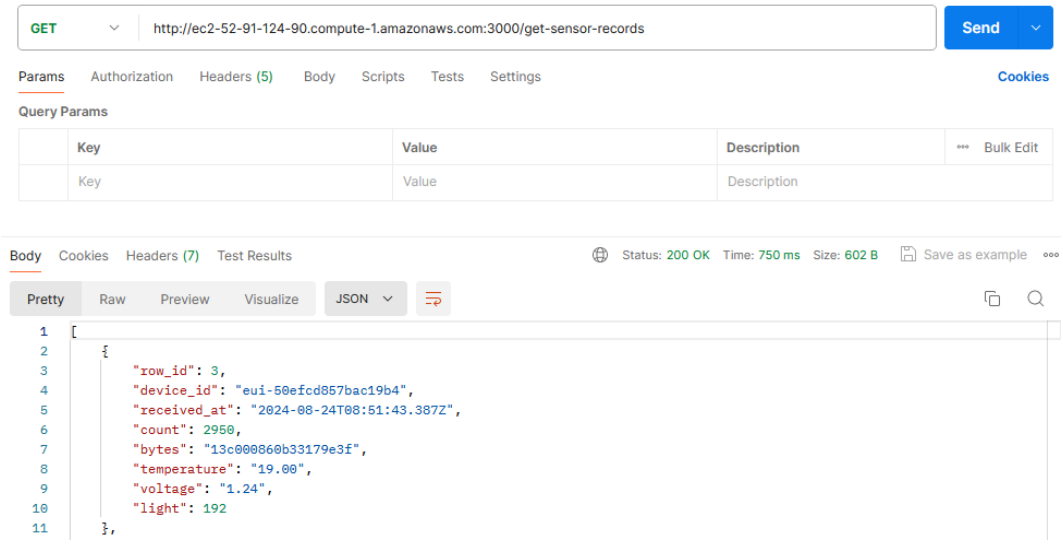
4. Click ***Send***. Observe the console output of the web server in the SSH session.
    a. Switch to the Cloud9 BASH window, and type the following in the MySQL console: SELECT * FROM devices_data;
5. You should see something like:

```
MySQL [sensor_data]> select * from devices_data;
+--------+--------------------+---------------------+-------+------------------+-------------+---------+-------+
| row_id | device_id          | received_at         | count | bytes            | temperature | voltage | light |
+--------+--------------------+---------------------+-------+------------------+-------------+---------+-------+
|      3 | eui-50efcd857bac19b4 | 2024-08-24 08:51:43.387 |  2950 | 13c000860b33179e3f |       19.00 |    1.24 |   192 |
+--------+--------------------+---------------------+-------+------------------+-------------+---------+-------+
1 row in set (0.001 sec)
```

6. In Postman, create another connection:

Action	GET
URL	http://<Web server public DNS>:3000/get-sensor-records

7. Click ***Send*** and observe the record you just posted, e.g.:



8. Now send the POST connection again and observe that it fails. This is because the `UNIQUE INDEX` constraint on the record has failed: the inputs for both `device_id` and `received_at` are identical to the first record.
9. Look at the SSH console to see details of the error.
10. Change the `receivedAt` time in the body of the POST and try again. This time it will succeed.

11. Send the GET connection again and observe that more than one record is retrieved.

Add a webhook from TTN to AWS

The final step is to connect TTN to the webserver, sending a POST each time a message is received.

1. Connect to TTN at https://au1.cloud.thethings.network/console/applications
2. Select the application you built in Lab 2.
3. Select Integrations > Webhooks > Add Webhook
4. Create a Custom Webhook as below:

Webhook ID	lab3-webhook
Webhook format	JSON
Base URL	As in Postman POST above
Filter event data	up.uplink_message.decoded_payload
Enabled event types	Uplink message

Use the GET connection in Postman to check that data is reaching the database every 30 seconds.

Final thoughts

Security & robustness

You could harden your web server, so that it starts automatically, with code like this:

```
npm install -g pm2
pm2 start app.js
pm2 save
pm2 startup
```

But because this server will be thrown away at the end of the lab, this step is omitted.

Credentials management could be improved. In particular, credentials that are hardcoded explicitly in source code could be held in the machine environment or – better – managed by AWS Secrets Manager.

Security groups could be improved: we could forbid access to the web service to everyone except our dashboard and TTN.

Debug & tracing

There are a lot of ways to monitor the data flowing through the system, including:

- **Putty** can display the trace output from the original node.
- **TTN Gateway** can show the raw data moving in to the TTN network.
- **TTN Application** can monitor the decoded and formatted uplink data messages.
- **Node.js server** in SSH terminal writes logs to the terminal console.
- **MySQL client** in Cloud9 can run: `select * from devices_data;`
- **Postman** can run a GET on the web server's get-sensor-records endpoint.

Cleaning up

Remember to return to the AWS Sandbox (labs.vocareum.com/…) and click End Lab. This will clean up all your resources.

To Do:

Multi-line instructions are misleading and they may break when copy/pasted.