**IOT CA3**

**The Smart Coaster (Beta Release)**

A logo with coffee beans on it

Description automatically generated

Jamie Lawlor

Caitlin Maguire

Luke Hilliard

Shahzad Shabeer

**Version 3**

**December 2024**

**Table Of Contents**

1. [**SYSTEM ARCHITECTURE……...…………………………...…………………………... 3**](#SystemArchitecture)
2. [**DEPLOYED TO AWS…….………………………………………………….......................**](#AWSAlpha)

**2.1** [Custom Domain..…………………………………………………………………………..……….](#CustomDomain)

**2.2** [HTTPS……….…………………………………………………………………………..................](#AWS2)

**2.3** Available after submission………………………………………………………………………….

1. [**PUBNUB……………………...………………………….…………………………….…….**](#Pubnub)
   1. [Access Manager…………………………………………………………………………...….……](#Communication)
   2. SSL…………………………………………………………………………………………………
   3. Communication…………………………………………………………………………………….
2. [**SECURITY……………………...……………………….……………………………….….**](#Security)
   1. [Device………..……..……………………………………………………………………...………](#IoTDevice)
   2. [Communication Channels……………………...…………………………………………………..](#CommunicationChannels)
   3. [Database…………………………………………………………………………………...……….](#DatabaseSecurity)
   4. [Webserver…………………………………………………………………………………………](#ServerSecurity)
   5. [Data in flow……………………………………………………………………………………..](#DataInTransit)
3. **System Architecture**

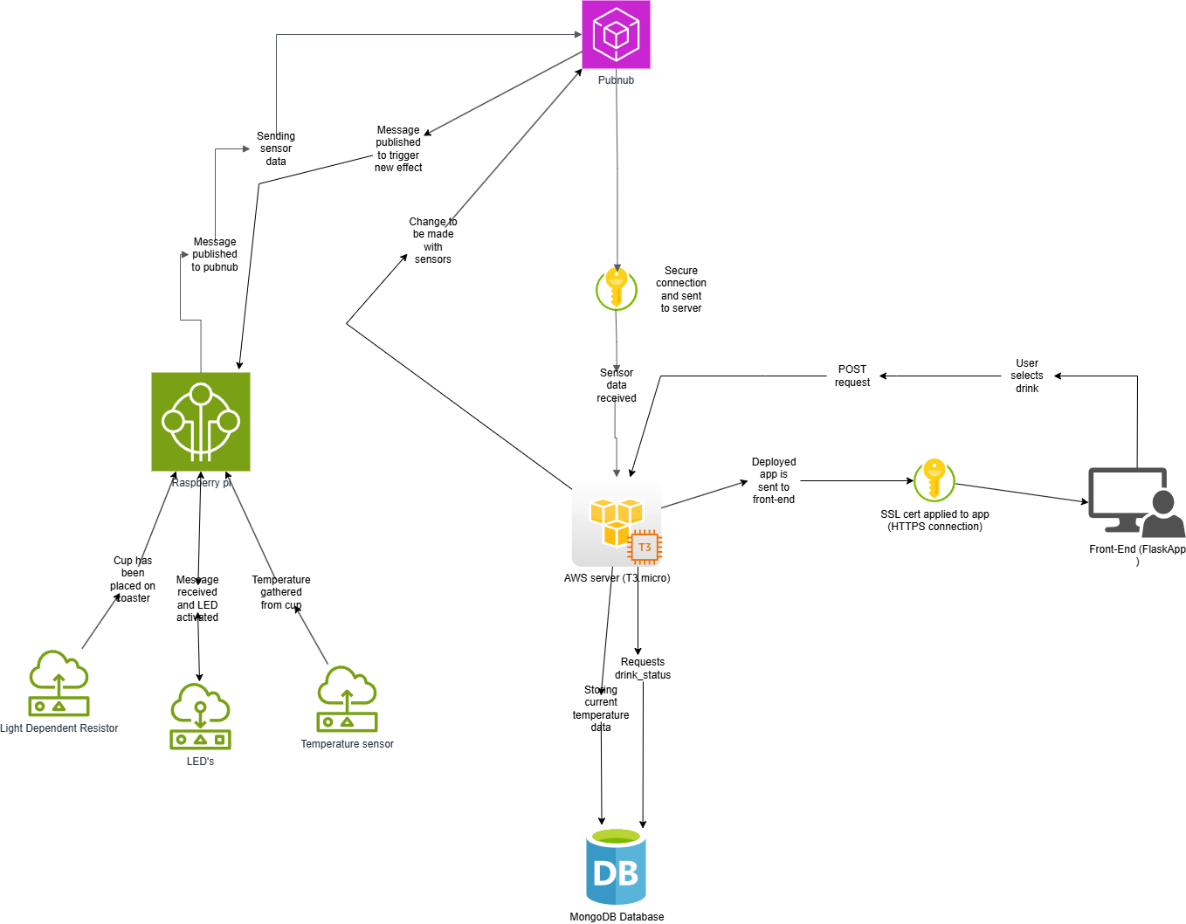
****

Figure Updated System Architecture Diagram

1. **Deployed to AWS**
   1. **Custom Domain – Jamie**

**Godaddy.com**

Using godaddy.com I was able to purchase a custom domain for the project. I had been given a number of different options for the domain name and after a long discussion with the team, we decided on sipify.site. After gaining access to the domain I created a hosted zone for the domain on Route53. Using this I was given access to AWS name servers to route traffic to my domain. I created a record for my domain and assigned the Ipv4 address from my instance to route the traffic to connect my project with the domain.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

I then changed the godaddy nameservers for my domain to be the custom nameservers that route53 provided to successfully route the domain to the instance. I then created a record to allow www. to be used to access the website as well. This record allowed the website to be accessed using sipify.site or [www.sipify.site](http://www.sipify.site).

A screen shot of a computer

Description automatically generated

Figure Accessing website without www

A screen shot of a computer

Description automatically generated

Figure Accessing website using www

**2.2 HTTPS – Jamie**

When first creating the AWS server and deploying the project, the website only had unsecure http access. In reality most users avoid accessing http websites as they are unsecure and can provide risk to being attacked. When accessing the website originally, Google Chrome provided this message before continuing to the page.

A screenshot of a computer

Description automatically generated

To secure the website and provide a https connection, I used letsencrypt.org to generate an SSL cert for the domain. In the AWS instance I added the inbound rule to allow https requests to the server and redirect all http traffic to https. After using certbot to generate the certificate I ran a test on ssllabs.com/ssltest to ensure that the certificate was successfully generated and that my domain was now secure.

A screenshot of a computer

Description automatically generated

This provides an extra layer of security both to the website and the user as the SSL certificate provides a secure connection just like all the other websites you would use on a daily basis.

**2.3 Available after Submission – Jamie**

A big issue I experienced when working with the AWS server was stopping and starting the instance, which gave new Ipv4 addresses. I will be keeping the AWS instance running until the beginning of semester 2 so it may be accessed by anyone who wishes to view the website, There will be an override button included on the website for John so that he can bypass the ldr detection and view the rest of the project.

**3. Pubnub**

**3.1 Access Manager – Jamie**

Unfortunately I was unable to have access manager completed for this submission. One of the main problems I focused was figuring out a way of granting a token to a user without the use of a login. The access manager documentation on pubnub is very vague and there isn’t a lot of detail on how to approach this. What I would have like to have done was to grant a token to a user when they logged into the website which would grant read and write access so that they may write the drink they have selected and read in the responses that pubnub have sent to them. I have both a “Sipify-channel” channel for the ldr and temperature messages and a “Get-notification” channel which sends the notification to the pi and the pi sends a message based on what LED is triggered.

**3.2 SSL/TLS – Jamie**

Pubnub takes security extremely seriously for their users so it comes with SSL/TLS protocol already enabled. When messages are being sent to pubnub they are encrypted automatically which ensures that any information being sent between the client and server remains confidential and cannot be accessed by any unauthorised users. This was extremely useful as I didn’t have to worry about creating the TLS security

**3.3 Communication – Jamie**

I established two-way communication with pubnub for the project. In the beginning the data from the ldr and temperature sensor are published from the pi to pubnub and then received on the server to be utilised. The ldr data was used to display the correct screen on the front-end for whether a cup has been placed onto the coaster or not. The temperature data was sent to the database along with the drink that the user selected to be stored. Using the temperature data the correct notification is displayed on the front-end to the user but is also published to pubnub on a separate channel. This message is received on the pi which triggers the corresponding LED to visually display to the user if the drink is too hot, too cold or at the perfect temperature. This two way communication was essential when developing the interaction between the server and the hardware and the server with the front-end.

A screenshot of a computer

Description automatically generated

Figure Two-way connection with 2 channels

**4. Security**

**4.1 Device – Luke**

I took on a slight security role when constructing the coaster as the role itself was neglected. I tried to incorporate some security designs to ensure the device could not be opened so easily. To do this I cut a very small hole that fits a toothpick, holding both trays together while locking it from being separated.

A cardboard box on a white surface

Description automatically generated

In the above image you can see the two matching holes where the toothpick would enter. To the right you can also see a small slit which is just big enough to fit the power cable. This is another way I tried to incorporate security into my design by exposing as little of the Raspberry Pi as possible. With this design virtually none of the Pi is visible.

The images below show how the Raspberry Pi sits if it were to be sealed up and plugged in. When it is plugged in the wire acts as an anchor, stopping the Pi from moving around the box.

A small piece of electronic device

Description automatically generated with medium confidence

A cardboard box with a white cord plugged into it

Description automatically generated

* 1. **Communication channels – Jamie**
  2. **Database – Caitlin**

A large part of database security is having a backup running of the database. This can be easy to do using a MySQL database, however we are using a NoSQL database therefore this was more difficult. The cluster I created for the project is under a free tier, within the free tier you cannot access the built in backup features of MongoDB Atlas.

Following some research I have read up about different methods to back up a mongo database, and I decided to follow the approach of this website:  <https://medium.com/@andrewskangah/how-to-create-a-middle-man-automatic-backup-for-mongodb-atlas-mo-sandbox-cluster-tier-5e7276d66ee2>

First of all, I created a local backup of the database on my desktop. I did this by creating a folder called mongobackup, in the terminal I went into this location and pasted in the export connection string from the MongoDB Atlas website. In the connection string I included my username and password for the admin user, as I have full access to the database. I also needed to edit the connection string by including the database name and the specified collection, I started with the drink status collection.

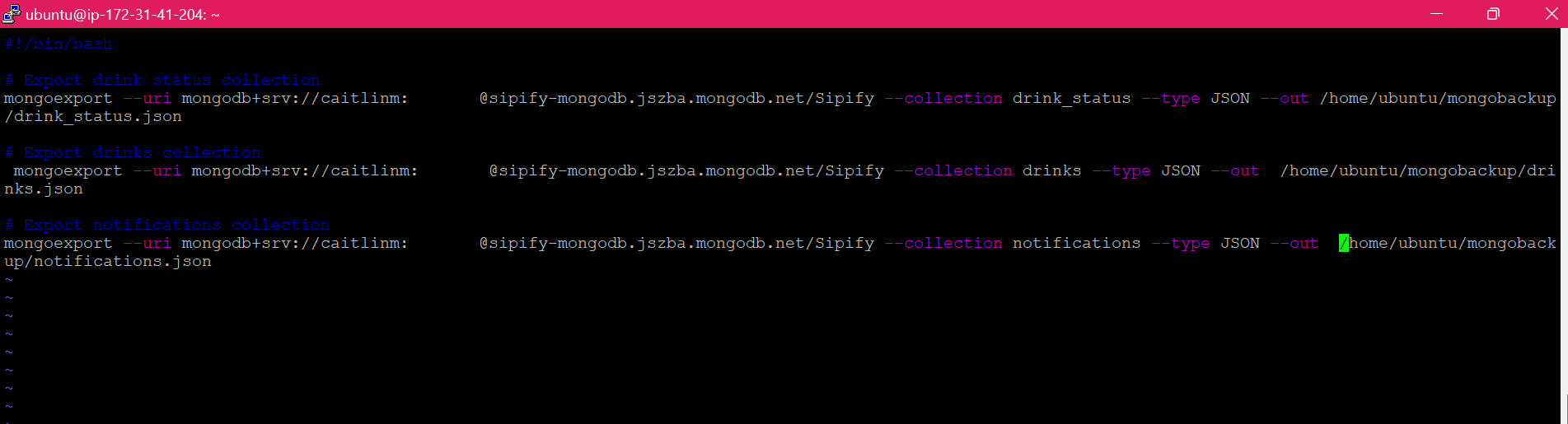
**Local Database Backup:**

**A screenshot of a computer

Description automatically generated**

After adding in the connection string in the terminal, I was getting an error that mongo-export was not recognised. I researched how to get mongo export installed in the terminal, and found I can download the MongoDB tools from the mongo website. I downloaded the tools into  my program files and created a path in the environment variables to the mongo tools. Once I did this the mongo export command was recognised and I was connected, the drink status records were coming into the local mongobackup folder. I repeated this process for my other collections drinks and notifications.

Once I was able to connect to atlas and create a local backup, I started the process of creating a backup on the AWS instance using a cron job. To do this first I created the script file called mongobackup.sh.

**Script:**

Next, I ran the script file to ensure it was connecting properly to atlas, then I moved on to creating the cron job. To test that the cron was running as it should, I set it to run every minute.

**Cron:**

**A screen shot of a computer

Description automatically generated**

**Log:**

To check the backup was running correctly I ran a command to view the syslogs.

**A screen shot of a computer

Description automatically generated**

 From the logs we can see the cronjob is running every minute as it should, now to check that the correct contents have come in to the mongobackup folder.

**Mongobackup Folder:**

**A screen shot of a computer

Description automatically generated**A copy of each of the files has been added the drink status , drinks and notifications all for each of the collections in the database.

**MongoDB Atlas Drink Status Collection:**

**A screenshot of a computer

Description automatically generated**

**Drink Status JSON:**

**A screen shot of a computer screen

Description automatically generated**

 By comparing the records from the drink status collection in MongoDB Atlas and in the backup drink\_status.json, we can see that the records have come in correctly . There are 17 records in the database and 17 values in the Json.

Now to test the backup, I am going to add in a new record into the MongoDB Atlas and check to see if it is added into the next backup on the server.

**New Record:**

**A screenshot of a computer

Description automatically generated**

A new record with the drink type Mocha has been added to the drink status collection, now we wait one minute to ensure the record is added to the next backup of the database in the server using the cron job.

A screen shot of a black screen

Description automatically generated

In the logs we can see the backup has ran again as expected.

**Updated Drink Status JSON:**

A screen shot of a computer

Description automatically generated

The bottom record shows that the new drink "Mocha" has been added into the back up of the database, using cron.

As we are not dealing with a large set of data, I think an appropriate number of backups would be once a week. I have changed the cron job to run every Friday at 12am as shown below.

**Updated Cron: A screenshot of a computer

Description automatically generated**

I used the website <https://crontab.cronhub.io/> to learn how to use cron, in the code above a Sudo command will be run on the script file that I created. The script file includes each of the mongo export connection strings to the collections. Without calling the sudo command on the script file, the cron job was not running correctly, and the records were not coming in.

For our database security we now have both a local backup and a backup running on a cron job on the AWS instance. By using the backup on the instance this ensures the drink status collection will be updated as new drinks are added. The drinks and notification collections are also included in the backup; however, the two collections will most likely not be changed unless necessary as this data is used as a comparison for the drink status.

**Drinks Collection Backup:**

**A screen shot of a computer

Description automatically generated**

**Notifications Collection Backup:**

**A screen shot of a computer

Description automatically generated**

 For the security of the database I have also created different users in Atlas, I am set to the admin user, and I have created users for my team members.

**Cup Detection Backup:**

A screen shot of a computer

Description automatically generated

**Users In Atlas: A screenshot of a computer

Description automatically generated**

**4.4 Webserver – Jamie**

**4.5 Data in flow - Jamie**