

Configuration Manual

MSc Research Project
Programme Name

Manjula Kore
Student ID: x23203986

School of Computing
National College of Ireland

Supervisor: Diego Lugones

National College of Ireland
Project Submission Sheet
School of Computing



Student Name:	Manjula Kore
Student ID:	x23203986
Programme:	Programme Name
Year:	2024
Module:	MSc Research Project
Supervisor:	Diego Lugones
Submission Due Date:	12/12/2014
Project Title:	Configuration Manual
Word Count:	620
Page Count:	9

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Manjula Kore
Date:	27th January 2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	<input type="checkbox"/>
Attach a Moodle submission receipt of the online project submission , to each project (including multiple copies).	<input type="checkbox"/>
You must ensure that you retain a HARD COPY of the project , both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	<input type="checkbox"/>

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Manjula Kore
x23203986

1 Introduction

This manual outlines the procedure for using an AWS Greengrass Core Device to receive sensor data from a Greengrass ESP32-Core running on the Wokwi simulator. The sensor data is published to AWS IoT Core through MQTT, then analyzed by the AWS Greengrass. The idea behind this configuration is to assimilate data from the connected sensor via ESP32; disseminate to the MQTT topics/subscribe to AWS Greengrass technologies. This makes data processing at the edge without undue delays, thus allowing real-time analytics and automation. The project applies on the ESP32 development board, Wokwi simulator, AWS Greengrass Core, and AWS IoT Core services in this aspect.

2 Tools and Technologies Required

Category	Tool/Technology	Description
Virtual Platform	Wokwi Simulation	Microcontroller for collecting sensor data and sending it to AWS IoT Core via MQTT.
Cloud Services	AWS Lambda	Serverless compute service for processing sensor data at the Greengrass core or cloud.
	AWS IoT Core	For managing the connection between the ESP32 and the cloud, using MQTT for data exchange.
	AWS Greengrass (Greengrass CLI (for deployment and management))	For local processing and management of AWS IoT devices and Lambda functions.
	AWS CloudWatch	For monitoring and storing logs generated by Lambda functions, Greengrass, and AWS SNS.
	AWS SNS	For sending alerts or notifications to subscribed users or systems.
	AWS S3	Object storage service to store and retrieve sensor data or logs, if required.
Programming Languages	YAML	Configuration file format for AWS IoT, Greengrass, and Lambda setup.
	Python	AWS Lambda functions or local processing scripts that interact with AWS services.
	C/C++	For developing firmware on the ESP32 using the Arduino IDE.

Table 1: Tools and Technologies

3 System Specifcations

3.1 AWS IoT Setup

AWS IoT Core Documentation (n.d.)

1. Create a new IoT Thing on the AWS IoT Core console and download the generated certificates AS SHOWN IN Fig. 1
2. Attach a policy to allow IoT actions (e.g., publish, subscribe).
3. Setting up MQTT Topics in IoT Core (Publish : 'iotfrontier/sub' and Subscribe Topic: 'iotfrontier/pub') as per Fig. 2
4. Test MQTT Connection : Use the Test section in the AWS IoT Core console to subscribe to 'iotfrontier/pub' and ensure messages are received.

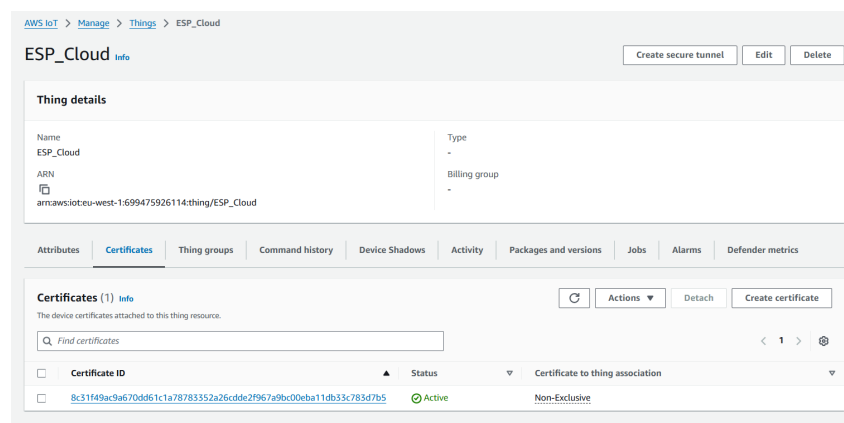


Figure 1: Thing Creation

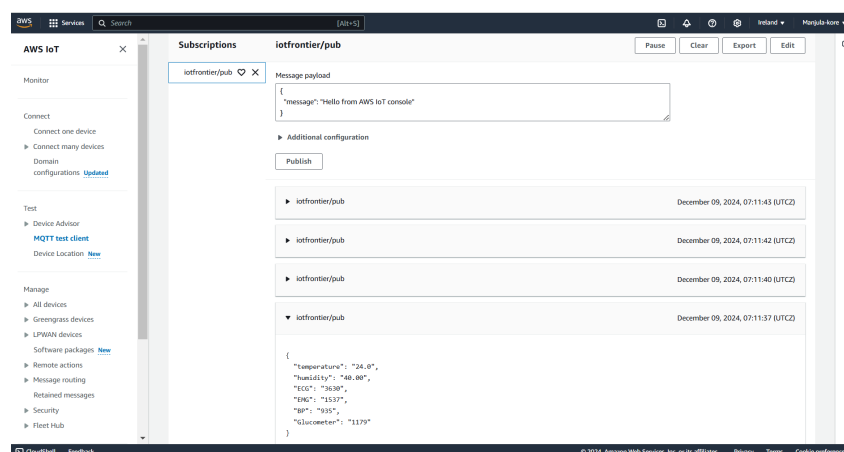


Figure 2: MQTT Client Test

3.2 Setting Up ESP32 and Wokwi Simulation

Welcome to Wokwi! | Wokwi Docs (n.d.)

1. Create a Wokwi Project
2. ESP32 Setup in Wokwi
3. Install the required libraries for your sensor and WiFi and MQTT connectivity.
4. Prepare a config file for AWS configurations as shown below Fig. 3 and replace your AWS details like Thing Name, endpoint, and certificates such as AmazonRootA, Private key cert and AWS cert created while creating Thing on AWS IoT Core.

```
#include <pgmspace.h>
#define SECRET
#define THINGNAME "ESP_Cloud"
const char WIFI_SSID[] = "Wokwi-GUEST";
const char WIFI_PASSWORD[] = "";
const char AWS_IOT_ENDPOINT[] = "agvydrva1bxn5-ats.iot.eu-west-1.amazonaws.com";
// Amazon Root CA 1
static const char AWS_CERT_CA[] PROGMEM = R"EOF(
-----BEGIN CERTIFICATE-----
-----END CERTIFICATE-----
)EOF";
// Device Certificate
static const char AWS_CERT_CRT[] PROGMEM = R"KEY(
-----BEGIN CERTIFICATE-----
-----END CERTIFICATE-----
)KEY";
// Device Private Key
static const char AWS_CERT_PRIVATE[] PROGMEM = R"KEY(
-----BEGIN RSA PRIVATE KEY-----
-----END RSA PRIVATE KEY-----
)KEY";
```

Figure 3: Config file : secrets.h

5. Write code in the file named "sketch.ino" for sensor data and MQTT connection, where sensors PIN will as shown below Fig. 4

```
// Pin definitions
const int DHT_PIN = 15;
const int ECG_PIN = 34; // ECG sensor output pin
const int EMG_PIN = 35; // EMG sensor output pin
const int BP_PIN = 36; // BP sensor output pin
const int GLUCOMETER_PIN = 39; // Glucometer sensor output pin

// Sensor objects
DHTesp dhtSensor;

// AWS IoT topics
#define AWS_IOT_PUBLISH_TOPIC "iotfrontier/pub"
#define AWS_IOT_SUBSCRIBE_TOPIC "iotfrontier/sub"
```

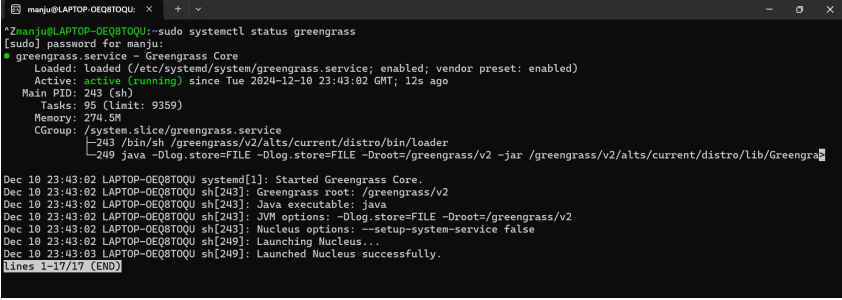
Figure 4: Sensor Pin setup

6. Run the Simulation to generate real-time data for the sensors attached.

3.3 AWS Greengrass Setup

AWS IoT Greengrass Documentation (n.d.)

1. Install Greengrass Core on local 5
 - (a) `curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-latest.zip >greengrass-nucleus-latest.zip && unzip greengrass-nucleus-latest.zip -d GreengrassInstaller`
 - (b) `sudo -E java -Droot="/greengrass/v2" -Dlog.store=FILE -jar ./GreengrassInstaller/lib/Greengrass.jar --aws-region eu-west-1 --thing-name Greengrass-Core-Device --thing-group-name greengrass-core-group --component-default-user ggc_user:ggc_group --provision true --setup-system-service true --deploy-dev-tools true`
 - (c) `sudo /greengrass/v2/bin/greengrass-cli --version`
 - (d) Locate `config.yaml`
system:
certificateFilePath: "/path/to/device-certificate.pem.crt"
privateKeyPath: "/path/to/device-private.pem.key"
rootCaPath: "/path/to/AmazonRootCA1.pem"
iotDataEndpoint: "your-iot-endpoint"
iotCredEndpoint: "your-cred-endpoint"
region: "eu-west-1"



```
manju@LAPTOP-OE8TOQU: ~$ sudo systemctl status greengrass
[sudo] password for manju:
* greengrass.service - Greengrass Core
   Loaded: loaded (/etc/systemd/system/greengrass.service; enabled; vendor preset: enabled)
   Active: active (running) since Tue 2024-12-10 23:43:02 GMT; 12s ago
     Main PID: 243 (sh)
       Tasks: 95 (limit: 9359)
      Memory: 274.5M
      CGroup: /system.slice/greengrass.service
              └─243 /bin/sh /greengrass/v2/alts/current/distro/bin/loader
                 └─249 java -Dlog.store=FILE -Dlog.store=FILE -Droot=/greengrass/v2 -jar /greengrass/v2/alts/current/distro/lib/Greengrass.jar

Dec 10 23:43:02 LAPTOP-OE8TOQU systemd[1]: Started Greengrass Core.
Dec 10 23:43:02 LAPTOP-OE8TOQU sh[243]: Greengrass root: /greengrass/v2
Dec 10 23:43:02 LAPTOP-OE8TOQU sh[243]: Java executable: java
Dec 10 23:43:02 LAPTOP-OE8TOQU sh[243]: JVM options: -Dlog.store=FILE -Droot=/greengrass/v2
Dec 10 23:43:02 LAPTOP-OE8TOQU sh[243]: Nucleus options: --setup-system-service false
Dec 10 23:43:02 LAPTOP-OE8TOQU sh[243]: Launching Nucleus...
Dec 10 23:43:03 LAPTOP-OE8TOQU sh[249]: Launched Nucleus successfully.
lines 1-17/17 (END)
```

Figure 5: Greengrass CLI

2. Create Greengrass Core Device with correct AWS IoT Thing name and IoT endpoint. (from AWS IoT Core console).
3. Create Deployable Greengrass Core with attached local Greengrass Core 6
4. Prepare Lambda for Local Execution
5. Attach Roles and policies to both Greengrass Core and Lambda Function.

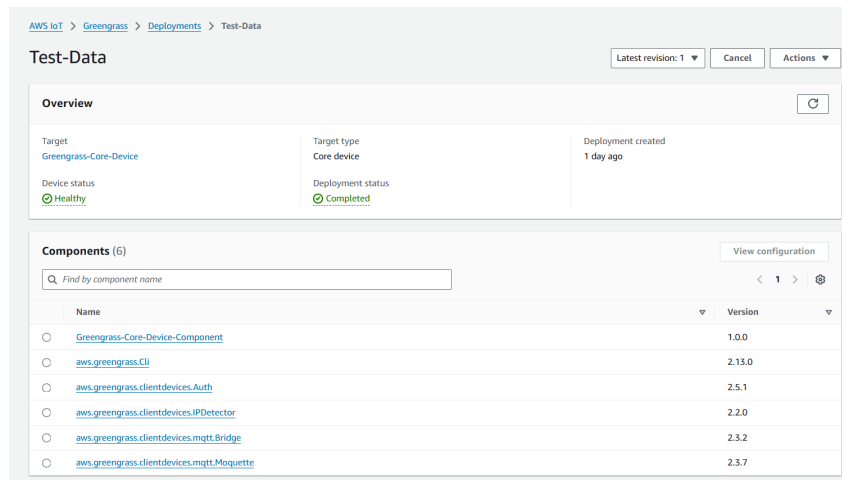


Figure 6: Greengrass Component

3.4 Deploy Lambda to Greengrass

1. In the AWS Lambda Console, create the "lambda.function.py" function and include code as per Fig. 7, Fig. 8, Fig. 9.

```
def analyze_patient_data(event):
    """Analyze sensor data locally."""
    try:
        # Extract sensor data
        temperature = float(event.get('temperature', 0))
        blood_pressure = float(event.get('blood_pressure', 0))
        heart_rate = float(event.get('heart_rate', 0))
        sugar_level = float(event.get('sugar_level', 0))
        movement = float(event.get('movement', 0))
    except ValueError as e:
        logging.error(f"Error converting event data: {e}")
        return {'error': 'Invalid data format'}

    # Analyze patient data
    patient_condition = {}
```

Figure 7: Analyze Sensor Data Locally

2. Add all conditions on sensor data recieved from IoT core.
3. Use a local Python environment to simulate Lambda function inputs.
4. Configure the function to allow Greengrass execution.
5. Add Lambda function to Greengrass Deployment.
6. Set Lambda execution parameters.
7. Deploy the updated configuration to the Greengrass Core device.
8. Test local lambda function on AWS Lambda Console.

```

# Analyze temperature
if temperature < 98.3:
    patient_condition['temperature'] = 'Low - Serious'
elif temperature > 91.3:
    patient_condition['temperature'] = 'High - Serious'
else:
    patient_condition['temperature'] = 'Normal'

# Analyze blood pressure
if blood_pressure < 60:
    patient_condition['blood_pressure'] = 'Low - Serious'
elif 60 <= blood_pressure <= 140:
    patient_condition['blood_pressure'] = 'Moderate - Not Serious'
else:
    patient_condition['blood_pressure'] = 'High - Serious'

# Analyze heart rate
if heart_rate < 60:
    patient_condition['heart_rate'] = 'Low - Serious'
elif 60 <= heart_rate <= 100:
    patient_condition['heart_rate'] = 'Moderate - Not Serious'
else:
    patient_condition['heart_rate'] = 'High - Serious'

```

Figure 8: Sensor Data Anamolies

```

def lambda_handler(event, context):
    """Handle incoming sensor data."""
    logging.info(f"Received event: {json.dumps(event)}")

    # Analyze data locally
    patient_condition = analyze_patient_data(event)
    logging.info(f"Analyzed patient condition: {patient_condition}")

    # Publish to IoT Core
    logging.info("Starting IoT Core publish")
    publish_to_cloud(patient_condition)
    logging.info("Finished IoT Core publish")

```

Figure 9: Publish to IoT Core

3.5 Amazon S3 for Data Storage

Amazon Simple Storage Service Documentation (n.d.)

1. Create an S3 bucket in the AWS Management Console. Fig. 10
2. Attach policies.
3. Add configuration of S3 object to the lambda function and Deploy updated lambda function.
4. Include S3 policies and IAM Role to the Lambda function's.
5. Update Lambda Function by adding s3 object configuration. Fig. 11
6. Enable versioning for tracking changes to uploaded files.


```
def save_to_s3(patient_condition):
    """Save the patient condition data to S3."""
    timestamp = datetime.utcnow().strftime('%Y-%m-%d_%H-%M-%S')
    s3_key = f'patient_condition_{timestamp}.json'
    try:
        s3_client.put_object(
            Bucket=S3_BUCKET_NAME,
            Key=s3_key,
            Body=json.dumps(patient_condition),
            ContentType='application/json'
        )
        logging.info(f"Data stored in S3: {s3_key}")
        print(f"Data stored in S3: {s3_key}")
    except Exception as e:
        logging.error(f"Error storing data in S3: {str(e)}")
```

Figure 10: Function to save data in S3

iot-lambda-healthcare-data [info](#)

[Objects](#) [Properties](#) [Permissions](#) [Metrics](#) [Management](#) [Access Points](#)

Objects (240) [info](#) [Copy S3 URI](#) [Copy URL](#) [Download](#) [Open](#) [Delete](#) [Actions](#) [Create folder](#) [Upload](#)

Objects are the fundamental entities stored in Amazon S3. You can use [Amazon S3 inventory](#) to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. [Learn more](#)

Find objects by prefix

<input type="checkbox"/>	Name	Type	Last modified	Size	Storage class
<input type="checkbox"/>	patient_condition_2024-12-10_04-58-26.json	json	December 10, 2024, 04:58:27 (UTC+00:00)	169.0 B	Standard
<input type="checkbox"/>	patient_condition_2024-12-11_02-31-52.json	json	December 11, 2024, 02:31:53 (UTC+00:00)	102.0 B	Standard
<input type="checkbox"/>	patient_condition_2024-12-11_02-32-58.json	json	December 11, 2024, 02:32:59 (UTC+00:00)	102.0 B	Standard
<input type="checkbox"/>	patient_condition_2024-12-11_02-33-50.json	json	December 11, 2024, 02:33:51 (UTC+00:00)	103.0 B	Standard
<input type="checkbox"/>	patient_condition_2024-12-11_02-34-26.json	json	December 11, 2024, 02:34:27 (UTC+00:00)	103.0 B	Standard
<input type="checkbox"/>	patient_condition_2024-12-11_02-36-03.json	json	December 11, 2024, 02:36:04 (UTC+00:00)	103.0 B	Standard
<input type="checkbox"/>	patient_condition_2024-12-11_02-37-04.json	json	December 11, 2024, 02:37:05 (UTC+00:00)	102.0 B	Standard
<input type="checkbox"/>	patient_condition_2024-12-11_02-47-24.json	json	December 11, 2024, 02:47:25 (UTC+00:00)	109.0 B	Standard

Figure 11: S3 Bucket Object

3.6 AWS SNS for Alerts and Notifications

Amazon Simple Notification Service Documentation (n.d.)

1. Create an SNS topic Fig. 12
2. Add subscribers to the topic
3. Add HTTP/S endpoint for custom integrations.
4. Update Lambda Function and add function for email send. Fig. 13 and Fig. 14

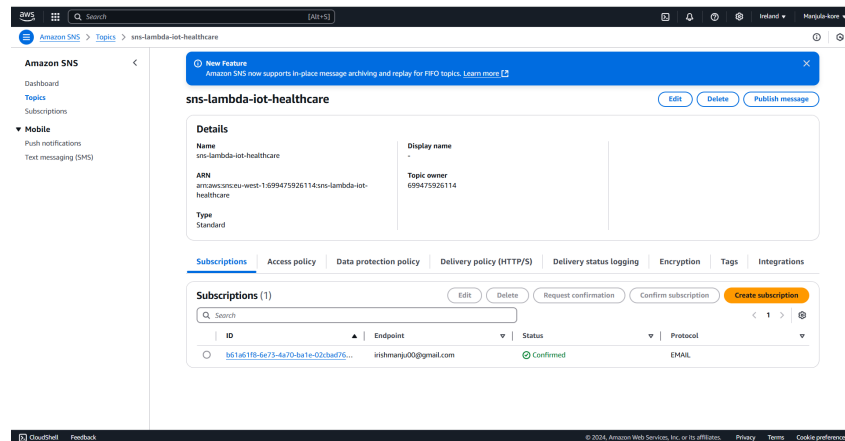


Figure 12: SNS Configuration

```
def send_sns_notification(patient_condition):
    """Send notifications for critical conditions via SNS."""
    try:
        if 'Low - Serious' in patient_condition.values() or 'High - Serious' in patient_condition.values():
            sns_client.publish(
                TopicArn=SNS_TOPIC_ARN,
                Message=json.dumps(patient_condition),
                Subject="Healthcare Alert - Patient Condition"
            )
            logging.info("SNS notification sent")
            print("SNS notification sent")
    except Exception as e:
        logging.error(f"Error sending SNS notification: {str(e)}")
```

Figure 13: Email sending function in Lambda

3.7 AWS CloudWatch for Monitoring

Amazon CloudWatch Documentation (n.d.)

1. Add CloudWatch permissions to the Lambda function's IAM role.
2. Lambda automatically creates a log group in CloudWatch for each function. You can view logs under the group: `/aws/lambda/"lambda-function-name"`. Fig. 15

```
# Initialize AWS clients
sns_client = boto3.client('sns')
s3_client = boto3.client('s3')
ipc_client = client.connect()

# Configuration
SNS_TOPIC_ARN = 'arn:aws:sns:eu-west-1:699475926114:sns-lambda-iot-healthcare'
S3_BUCKET_NAME = 'iot-lambda-healthcare-data'

# Set up logging
logging.basicConfig(level=logging.INFO)
```

Figure 14: S3 and SNS Client added in lambda function

Timestamp	Message
2024-12-11T02:59:56.398Z	START RequestId: f4edc396-8705-4cfd-a122-3f72001c731a Version: \$LATEST
2024-12-11T02:59:56.398Z	Received event: {"temperature": "38.8", "humidity": "48.88", "ECG": "1566", "DWC": "2485", "HR": "184", "glucose": "2481"}
2024-12-11T02:59:56.413Z	Data stored in S3: patient_condition_2024-12-11_02-59-55.json
2024-12-11T02:59:56.418Z	END RequestId: f4edc396-8705-4cfd-a122-3f72001c731a
2024-12-11T02:59:56.418Z	REPORT RequestId: f4edc396-8705-4cfd-a122-3f72001c731a Duration: 26.58 ms Billed Duration: 29 ms Memory Size: 128 MB Peak Memory Used: 81 MB
2024-12-11T02:59:56.378Z	START RequestId: 84219d5c-f487-45c9-ba36-d39599811094 Version: \$LATEST
2024-12-11T02:59:56.379Z	Received event: {"temperature": "38.8", "humidity": "48.88", "ECG": "1457", "DWC": "2382", "HR": "1955", "glucose": "2488"}
2024-12-11T02:59:56.411Z	Data stored in S3: patient_condition_2024-12-11_02-59-56.json
2024-12-11T02:59:56.418Z	END RequestId: 84219d5c-f487-45c9-ba36-d39599811094
2024-12-11T02:59:56.418Z	REPORT RequestId: 84219d5c-f487-45c9-ba36-d39599811094 Duration: 40.13 ms Billed Duration: 41 ms Memory Size: 128 MB Peak Memory Used: 81 MB
2024-12-11T02:59:57.480Z	START RequestId: 46605f5f-26c4-4774-a700-d8c443468506 Version: \$LATEST

Figure 15: CloudWatch Logs

References

Amazon CloudWatch Documentation (n.d.).

URL: <https://docs.aws.amazon.com/cloudwatch/>

Amazon Simple Notification Service Documentation (n.d.).

URL: <https://docs.aws.amazon.com/sns/>

Amazon Simple Storage Service Documentation (n.d.).

URL: <https://docs.aws.amazon.com/s3/>

AWS IoT Core Documentation (n.d.).

URL: <https://docs.aws.amazon.com/iot/>

AWS IoT Greengrass Documentation (n.d.).

URL: <https://docs.aws.amazon.com/greengrass/>

Welcome to Wokwi! | Wokwi Docs (n.d.).

URL: <https://docs.wokwi.com/>