Parking Lot System Manager

HOST: GUY TEL ZUR

STUDENTDS: ADI TZURDECKER, OMER MARK.

Agenda

- What is IoT-based Smart Parking System?
- Background & Related work
- Challenges
- System Architecture
- Implementation
- Future Ideas
- Summary

What is IoT-based Smart Parking System?

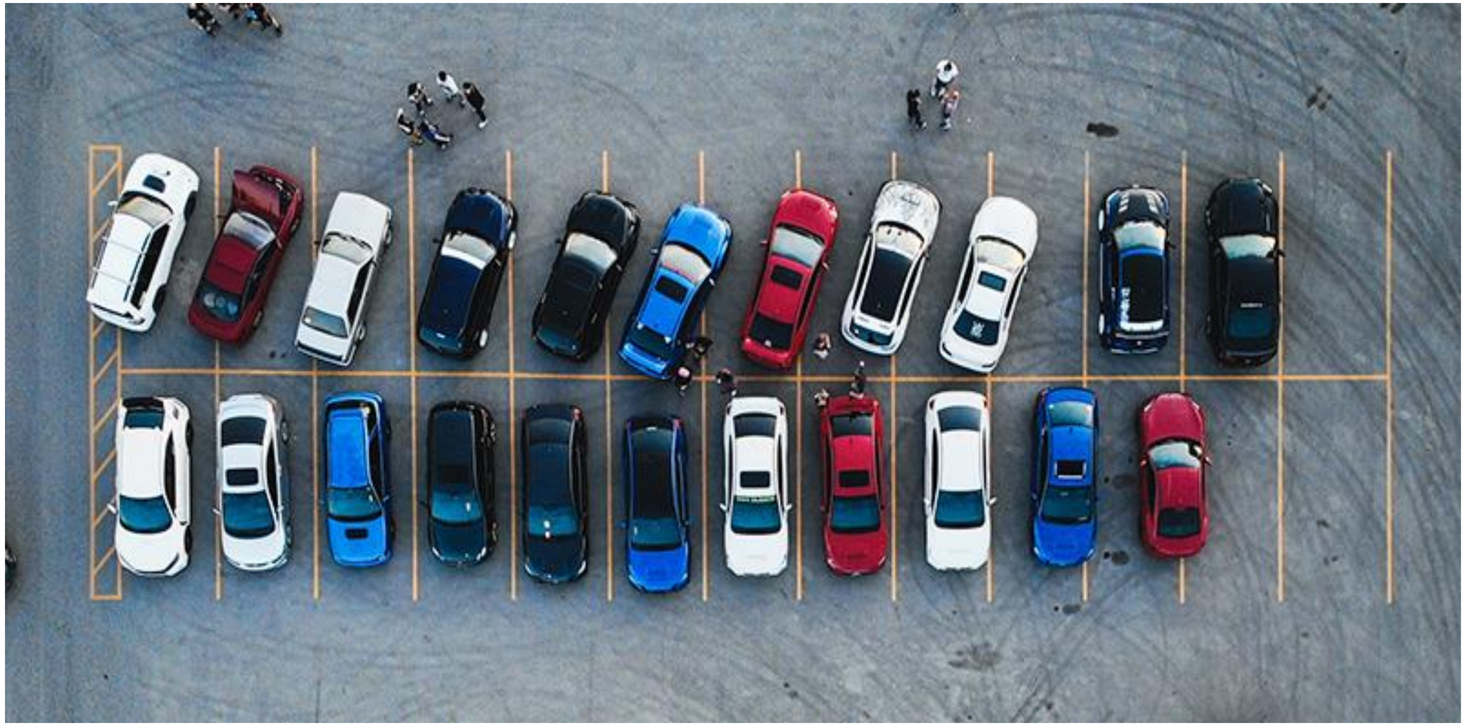
What is IoT-based Smart Parking System?

IoT-based smart parking system, is a management system design to monitor the parking lot occupancy, getting real time-data from IoT devices found in the parking lot communicating over network.

What is IoT-based Smart Parking System?

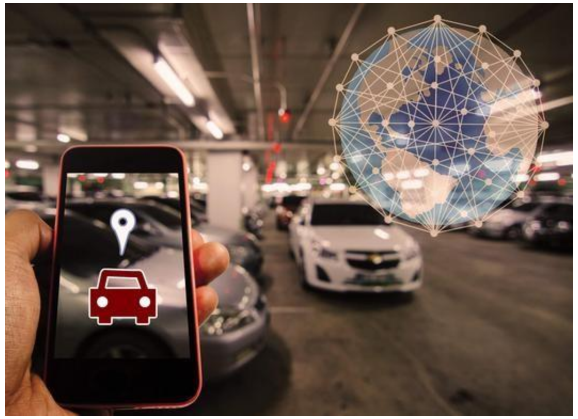
Issues

- Parking availability
- Parking spot recommendations
- Data gathering, storage and mining
- UI for easy management and monitor
- User application for drivers

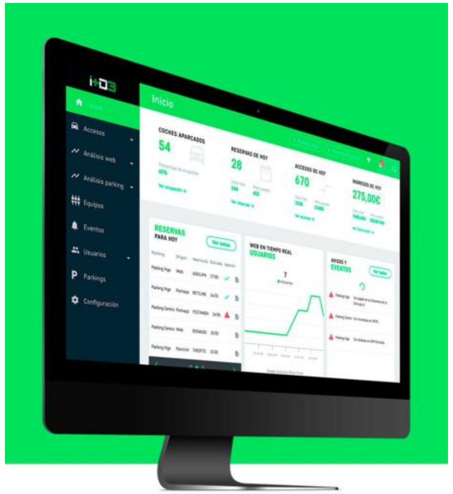


What is IoT-based Smart Parking System?

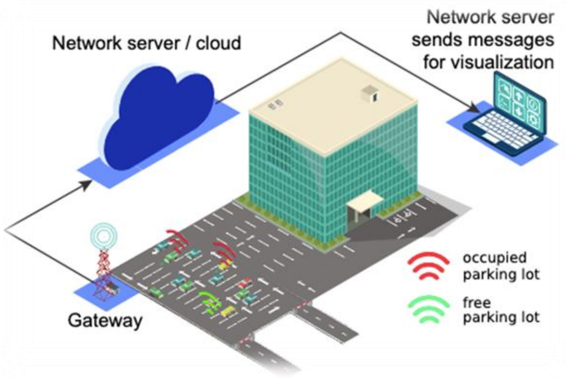
Quick peek to the future



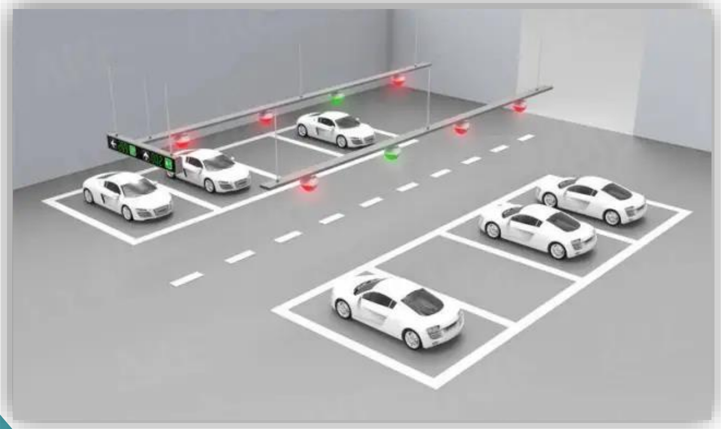
User web/mobile application



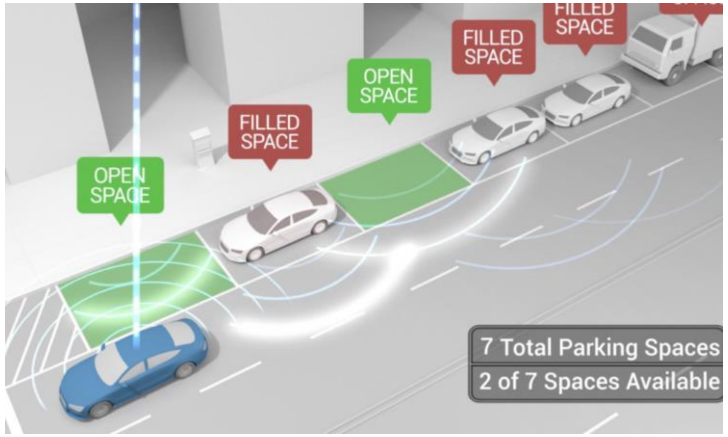
UI for manage and monitoring system



Cloud integration



Data spots recommendations

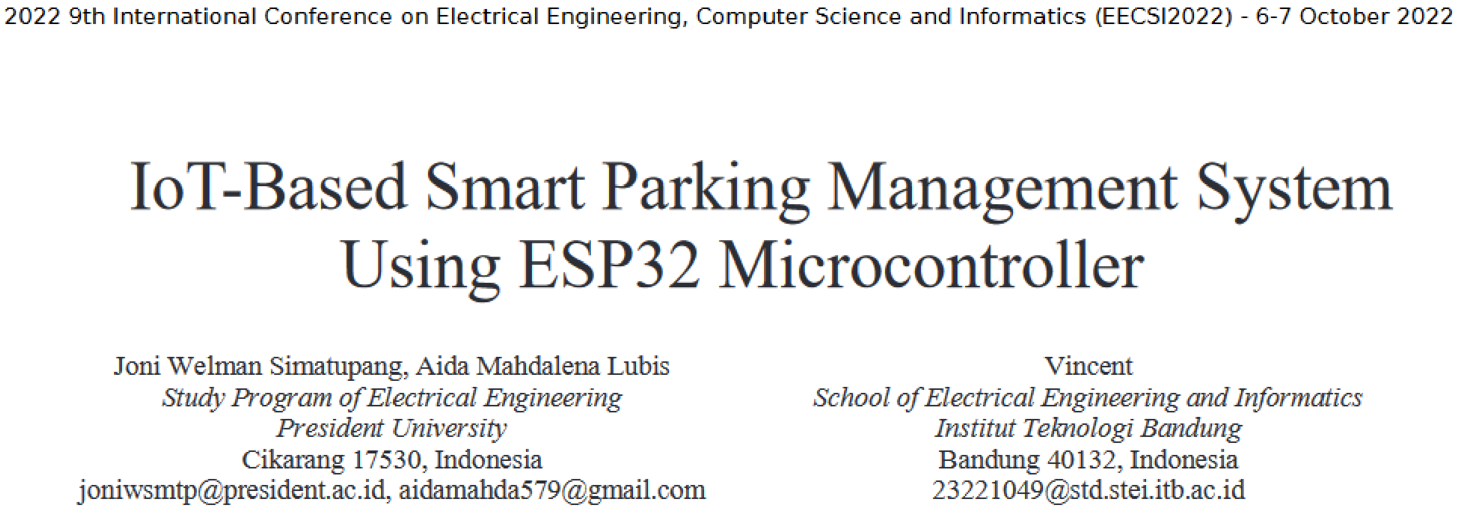


Parking availability in real time

Literature Reviews

Background & Related work

- Title: IoT-based smart parking management system using ESP32 microcontrollers
- Authors: Joni Welman Simatupang, Aida Mahdalena Lubis
- Publish date: 7/10/22
- Publisher: IEEE



2022 9th International Conference on Electrical Engineering, Computer Science and Informatics (EECSI2022) - 6-7 October 2022

IoT-Based Smart Parking Management System Using ESP32 Microcontroller

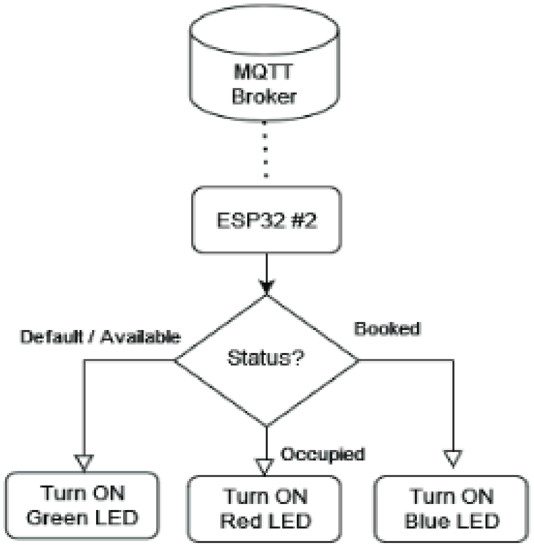
Joni Welman Simatupang, Aida Mahdalena Lubis
Study Program of Electrical Engineering
President University
Cikarang 17530, Indonesia
joniwsmtp@president.ac.id, aidamahda579@gmail.com

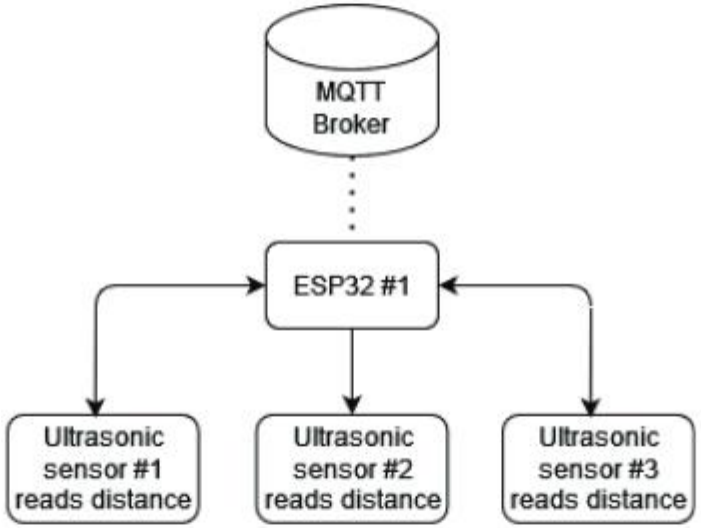
Vincent
School of Electrical Engineering and Informatics
Institut Teknologi Bandung
Bandung 40132, Indonesia
23221049@std.stei.itb.ac.id

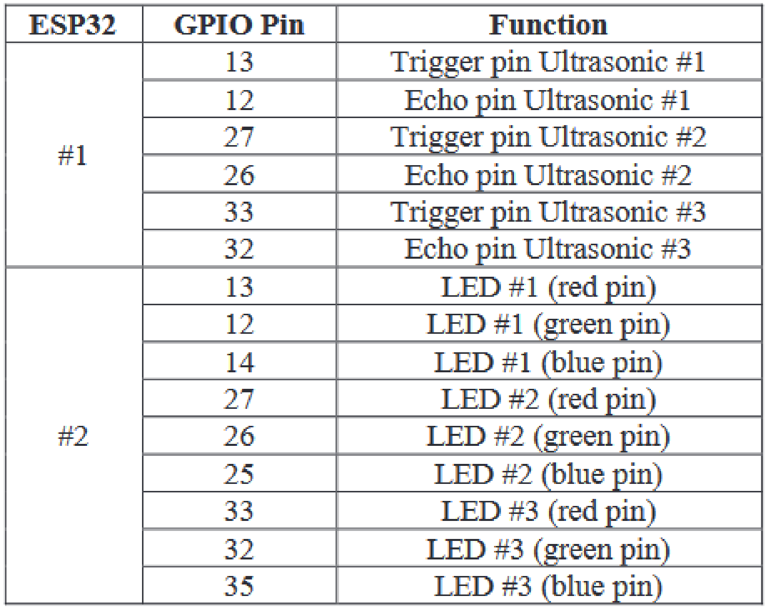
Article focuses about the problems of the ‘new world’ (pollution, traffic, gasoline…)
Challenges being dealt with are: cost and power consumption, website for the system, sensors accuracy.

Background & Related work

System configuration and Implementation




MQTT Broker
ESP32 #1
Ultrasonic sensor #1 reads distance
Ultrasonic sensor #2 reads distance
Ultrasonic sensor #3 reads distance



ESP32	GPIO Pin	Function
#1	13	Trigger pin Ultrasonic #1
	12	Echo pin Ultrasonic #1
	27	Trigger pin Ultrasonic #2
	26	Echo pin Ultrasonic #2
	33	Trigger pin Ultrasonic #3
	32	Echo pin Ultrasonic #3
#2	13	LED #1 (red pin)
	12	LED #1 (green pin)
	14	LED #1 (blue pin)
	27	LED #2 (red pin)
	26	LED #2 (green pin)
	25	LED #2 (blue pin)
	33	LED #3 (red pin)
	32	LED #3 (green pin)
	35	LED #3 (blue pin)

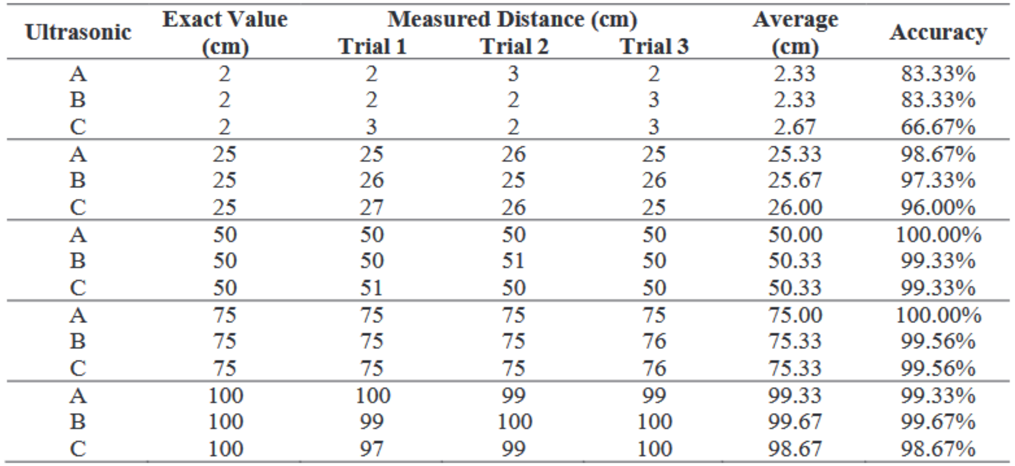
- 2 ESP32 for 3 parking slots
- Ultra sonicsensor, RGB LED.
- MQTT communication (Broker configuration wasn’t specified).

Background & Related work

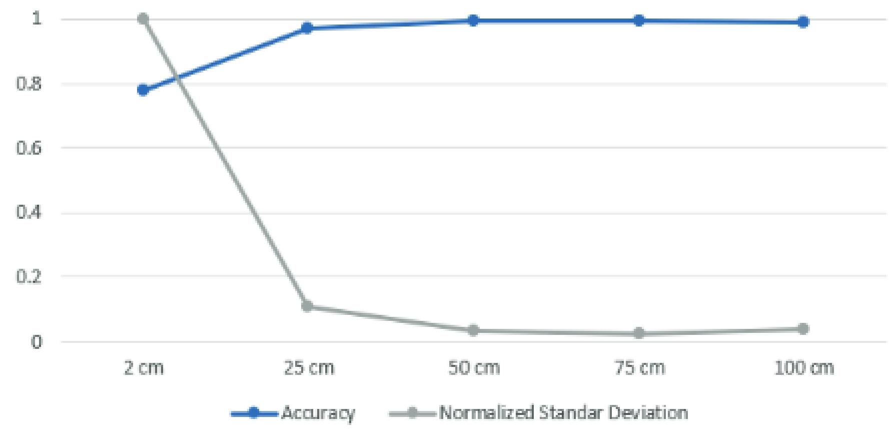
Results

- Threshold configured to if(vehicle _distance < 100) {park = TRUE}
- Ultrasonic accuracy is generally worse for shorter range.
- Results were evaluated for a web page dedicated to park reservations.

TABLE IV. ULTRASONIC TRIAL RESULTS



Ultrasonic	Exact Value (cm)	Measured Distance (cm)			Average (cm)	Accuracy
		Trial 1	Trial 2	Trial 3		
A	2	2	3	2	2.33	83.33%
B	2	2	2	3	2.33	83.33%
C	2	3	2	3	2.67	66.67%
A	25	25	26	25	25.33	98.67%
B	25	26	25	26	25.67	97.33%
C	25	27	26	25	26.00	96.00%
A	50	50	50	50	50.00	100.00%
B	50	50	51	50	50.33	99.33%
C	50	51	50	50	50.33	99.33%
A	75	75	75	75	75.00	100.00%
B	75	75	75	76	75.33	99.56%
C	75	75	75	76	75.33	99.56%
A	100	100	99	99	99.33	99.33%
B	100	99	100	100	99.67	99.67%
C	100	97	99	100	98.67	98.67%



Background & Related work

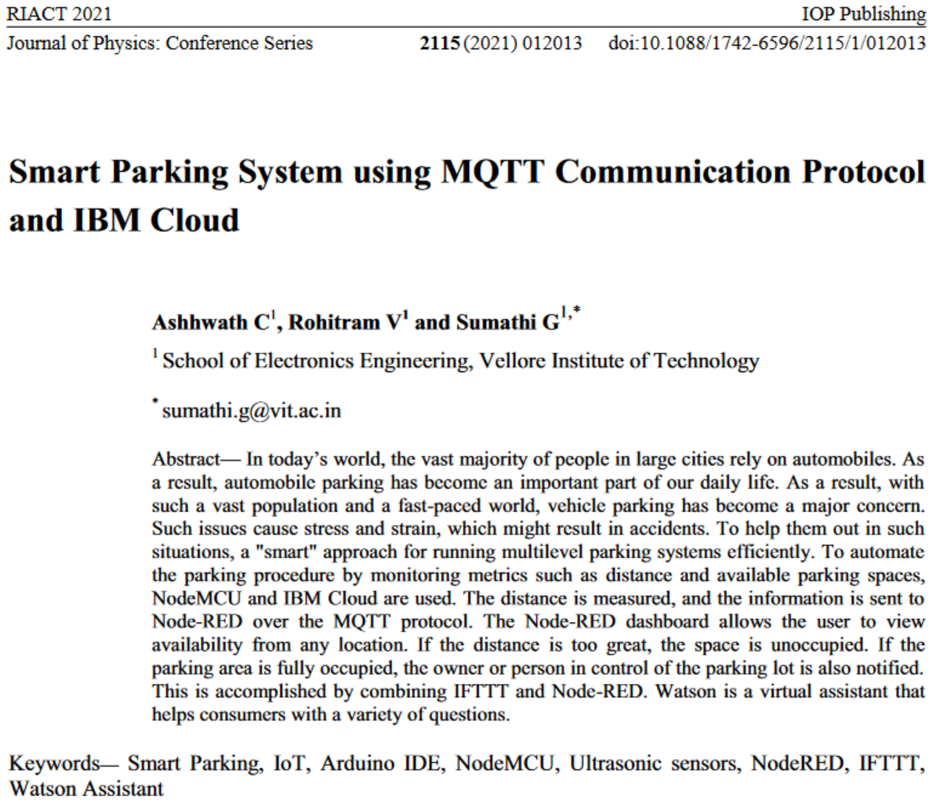
Summary

- MQTT broker configuration wasn’t specified.
- Data collection and gathering wasn’t discussed.
- Cloud integration wasn’t mentioned, scalability as well.
- System configuration isn’t clear where it comes to overcoming security issues.

Smart Parking System using MQTT Communication Protocol and IBM Cloud

Background & Related work

- Title: Smart Parking System using MQTT Communication Protocol and IBM Cloud
- Authors: Ashhwath C, Rohitram V and Sumathi G.
- Publish date: 2021
- Publisher: IOP Publishing Ltd.



RIACT 2021 IOP Publishing

Journal of Physics: Conference Series **2115** (2021) 012013 doi:10.1088/1742-6596/2115/1/012013

Smart Parking System using MQTT Communication Protocol and IBM Cloud

Ashhwath C[1], Rohitram V[1] and Sumathi G[1,*]

[1] School of Electronics Engineering, Vellore Institute of Technology

[*] sumathi.g@vit.ac.in

Abstract— In today's world, the vast majority of people in large cities rely on automobiles. As a result, automobile parking has become an important part of our daily life. As a result, with such a vast population and a fast-paced world, vehicle parking has become a major concern. Such issues cause stress and strain, which might result in accidents. To help them out in such situations, a "smart" approach for running multilevel parking systems efficiently. To automate the parking procedure by monitoring metrics such as distance and available parking spaces, NodeMCU and IBM Cloud are used. The distance is measured, and the information is sent to Node-RED over the MQTT protocol. The Node-RED dashboard allows the user to view availability from any location. If the distance is too great, the space is unoccupied. If the parking area is fully occupied, the owner or person in control of the parking lot is also notified. This is accomplished by combining IFTTT and Node-RED. Watson is a virtual assistant that helps consumers with a variety of questions.

Keywords— Smart Parking, IoT, Arduino IDE, NodeMCU, Ultrasonic sensors, NodeRED, IFTTT, Watson Assistant

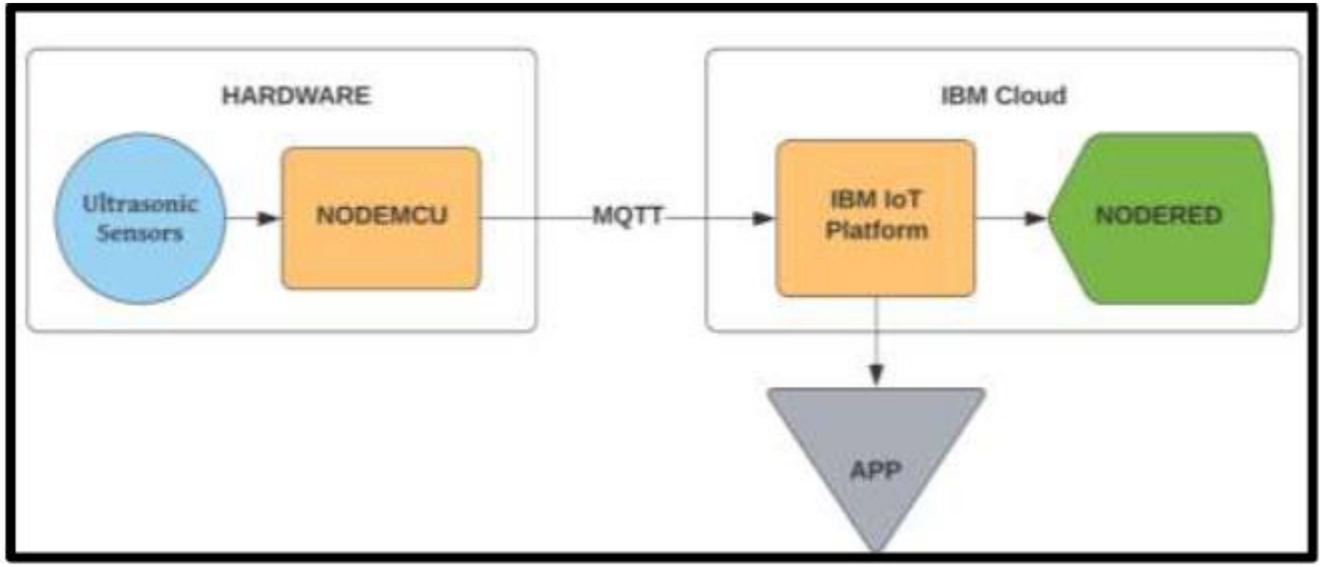
Article focuses about the problems such as time wasting over free parking spot seeking and traffic. Challenges being dealt with are: anywhere monitoring availability, actuators.

Smart Parking System using MQTT Communication Protocol and IBM Cloud

Background & Related work

System configuration and Implementation

- NodeMCU microcontroller
- Ultrasonic sensor, LED indicators.
- MQTT communication (Broker configuration wasn't specified).
- IBM IoT platform
- Node Red with IFTTT.
- App for drivers for track parking location.
- Watson assistant – queries for customers.



Background & Related work

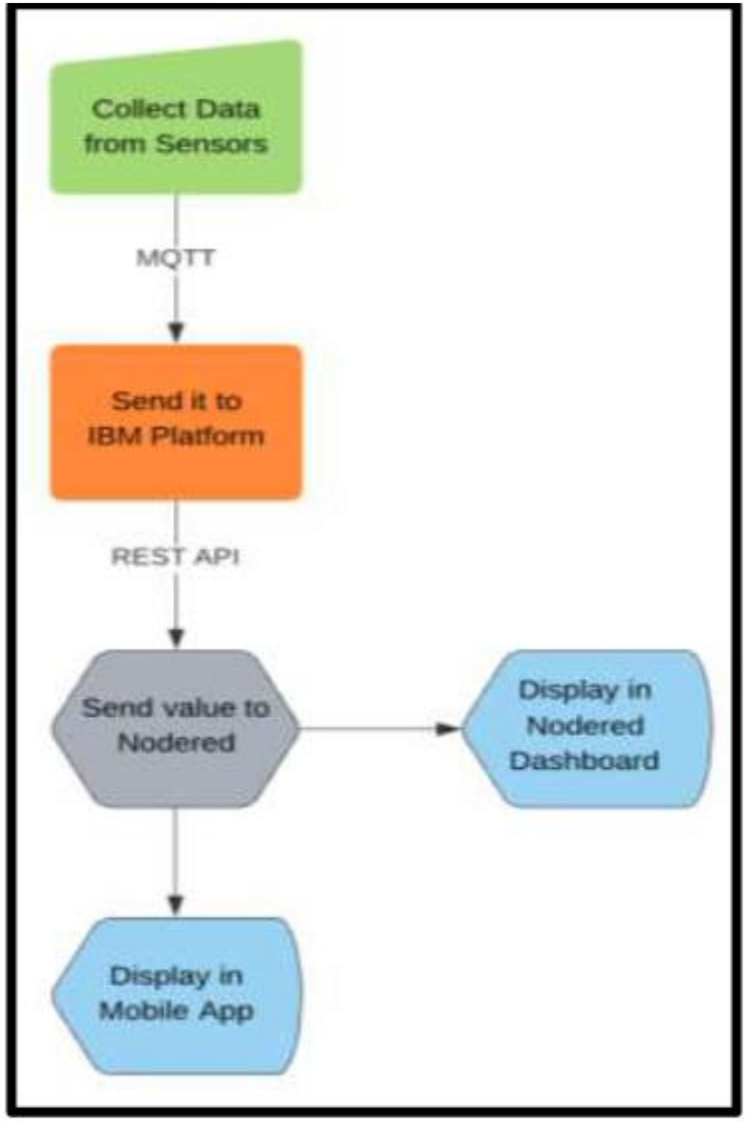


Figure 2. Workflow diagram

System configuration and Implementation

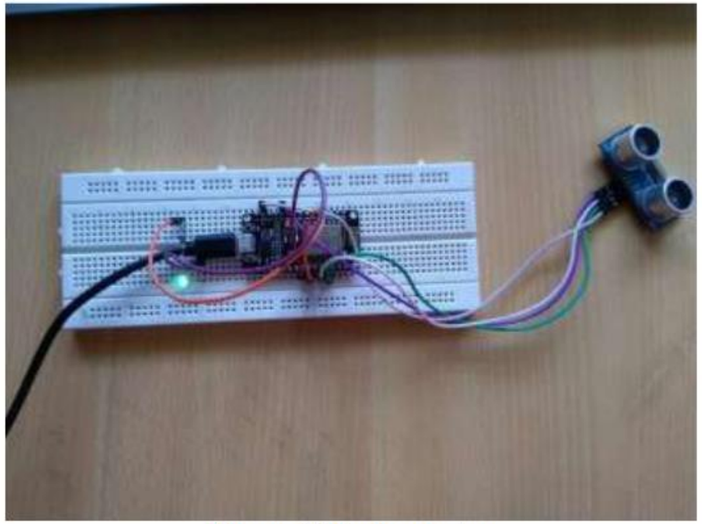
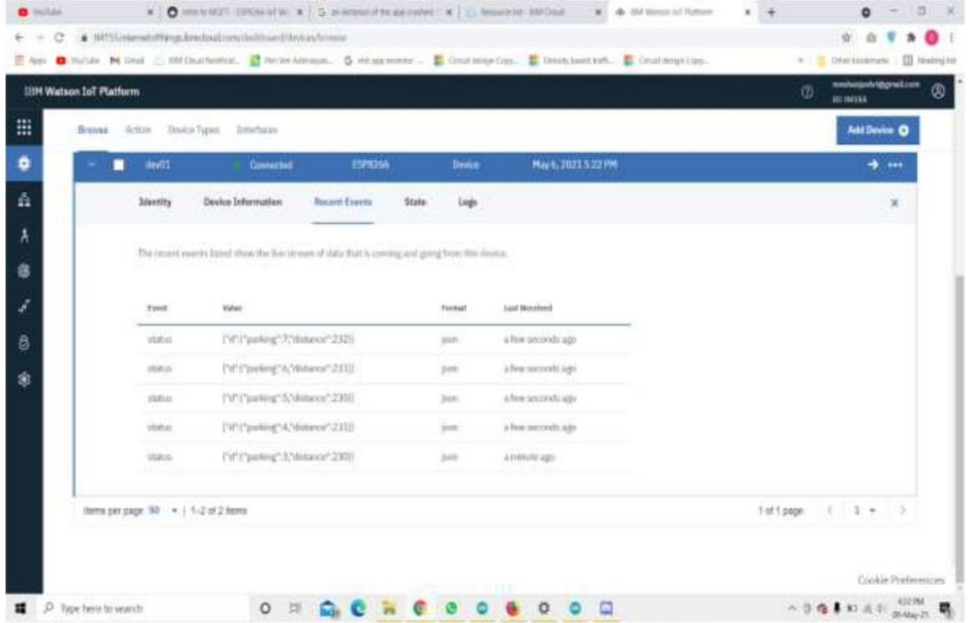


Figure 3. Hardware setup

NODEMCU device
Ultrasonic sensor
LED sensor

Recent MQTT published messaged emitted by the IoT thing broker on IBM cloud.



The recent events listed show the live stream of data that is coming and going from this device.

Event	Value	Format	Last Received
status	{"d":{"parking":7,"distance":232}}	json	a few seconds ago
status	{"d":{"parking":6,"distance":231}}	json	a few seconds ago
status	{"d":{"parking":5,"distance":230}}	json	a few seconds ago
status	{"d":{"parking":4,"distance":231}}	json	a few seconds ago
status	{"d":{"parking":3,"distance":230}}	json	a minute ago

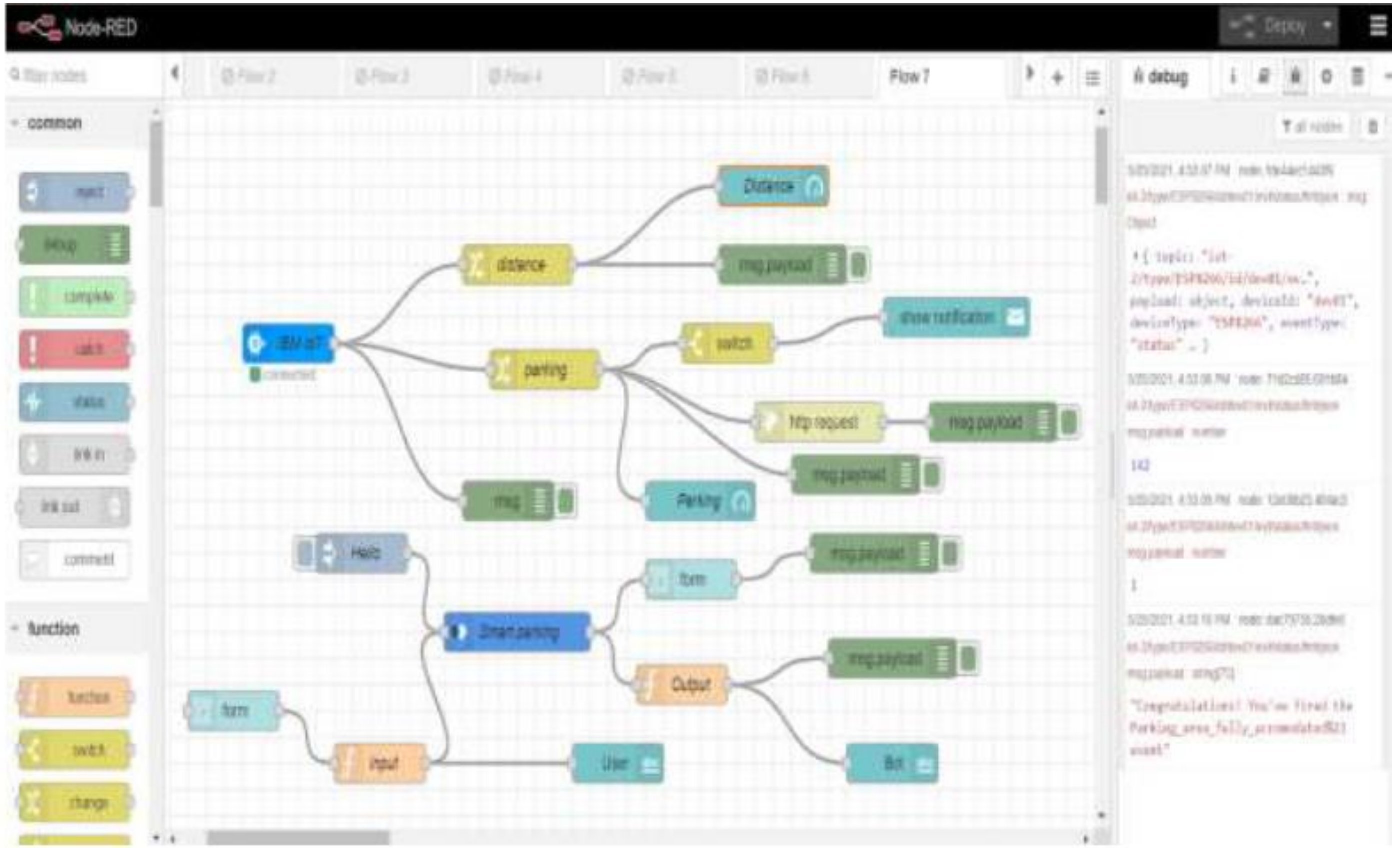
Items per page 50 | 1-2 of 2 items   1 of 1 page

Figure 5. Recent events in IBM Cloud Platform

Background & Related work

System configuration and Implementation

Node-RED has a significant usage in the system configuration, directing sensor data to various location in form of flow. Node-RED UI displays data is greatly useful for owner/security.



```
5/25/2021 4:53:07 PM  node: [illegible]
[illegible] msg : Object
▸{ topic: "iot-2/type/ESP8266/id/dev01/ev...", payload: object, deviceId: "dev01", deviceType: "ESP8266", eventType: "status" ... }
5/25/2021 4:53:08 PM  node: [illegible]
[illegible]
msg.payload : number
142
5/25/2021 4:53:09 PM  node: [illegible]
[illegible]
msg.payload : number
1
5/25/2021 4:53:10 PM  node: [illegible]
[illegible]
msg.payload : string[71]
"Congratulations! You've fired the Parking_area_fully_accomodated[illegible] event"
```

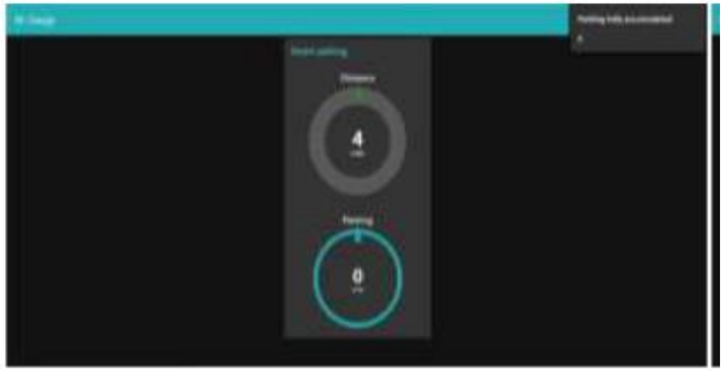


Figure 7. Node-RED Dashboard – Gauges and Web UI – Notification

Background & Related work

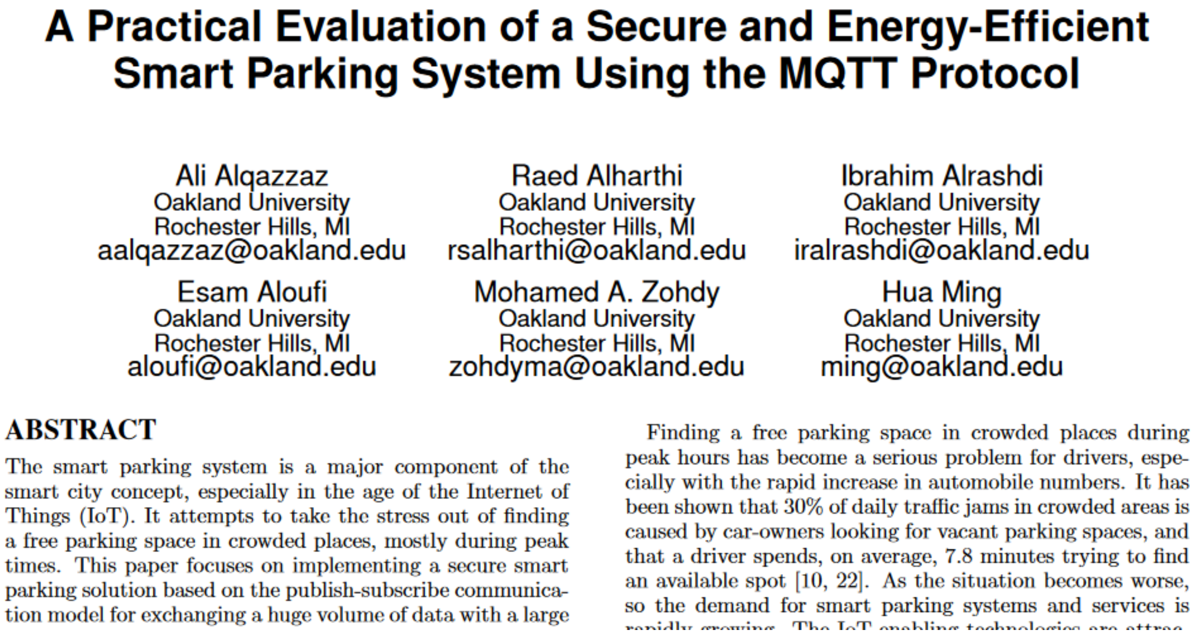
Summary

- ✓ System’s scalability is possible due to cloud integration.
- ✓ Monitoring real time data using Node-Red ‘gauges’ nodes .
- ✓ Mobile Application for customer’s feedback is also useful.

- X Mobile application – is it useful?
- X Data gathering and long-term storing isn’t discussed.
- X Security issues aren’t defined and dealt with.

Background & Related work

- Title: A Practical Evaluation of a Secure and Energy-Efficient Smart Parking System Using the MQTT Protocol
- Authors: Ali Alqazzaz, Raed Alharthi, Ibrahim Alrashdi, Esam Aloufi, Mohamed A. Zohdy, Hua Ming
  All from Oakland University
- Publish date: 6/4/2019
- Publisher: Association for Computing Machinery.



A Practical Evaluation of a Secure and Energy-Efficient Smart Parking System Using the MQTT Protocol

Ali Alqazzaz
Oakland University
Rochester Hills, MI
aalqazzaz@oakland.edu

Raed Alharthi
Oakland University
Rochester Hills, MI
rsalharthi@oakland.edu

Ibrahim Alrashdi
Oakland University
Rochester Hills, MI
iralrashdi@oakland.edu

Esam Aloufi
Oakland University
Rochester Hills, MI
aloufi@oakland.edu

Mohamed A. Zohdy
Oakland University
Rochester Hills, MI
zohdyma@oakland.edu

Hua Ming
Oakland University
Rochester Hills, MI
ming@oakland.edu

ABSTRACT

The smart parking system is a major component of the smart city concept, especially in the age of the Internet of Things (IoT). It attempts to take the stress out of finding a free parking space in crowded places, mostly during peak times. This paper focuses on implementing a secure smart parking solution based on the publish-subscribe communication model for exchanging a huge volume of data with a large

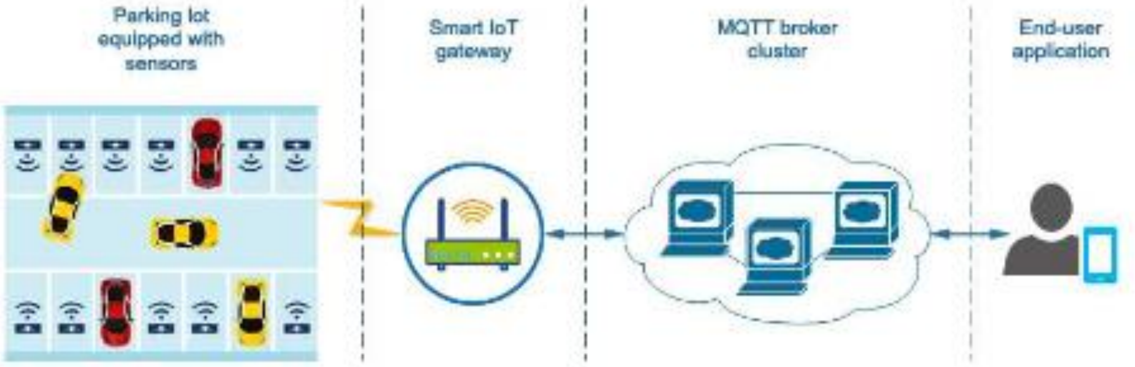
Finding a free parking space in crowded places during peak hours has become a serious problem for drivers, especially with the rapid increase in automobile numbers. It has been shown that 30% of daily traffic jams in crowded areas is caused by car-owners looking for vacant parking spaces, and that a driver spends, on average, 7.8 minutes trying to find an available spot [10, 22]. As the situation becomes worse, so the demand for smart parking systems and services is rapidly growing. The IoT enabling technologies are attrac-

Article focuses about the problems of highly traffic cities and smart cities process where smart parking lot is an integral part of them, and highly necessary.
Article main goal is to verify the efficiency and suitability of the SecSPS framework* and reduce power consumption and CPU utilization.

Background & Related work

System configuration and Implementation

- Ultrasonic Sensors
- SBC* clients/broker – Raspberry Pi 3 model B+ (OS – Raspbia)
- TP-Link TL-SG108 (Access points for WiFi)
- MQTT with using TLS with OpenSSL self generated keys and certificates.



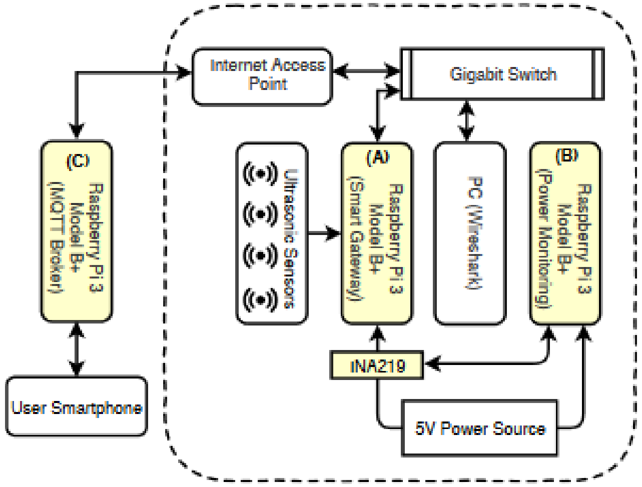

Internet Access Point
Gigabit Switch
(C) Raspberry Pi 3 Model B+ (MQTT Broker)
Ultrasonic Sensors
(A) Raspberry Pi 3 Model B+ (Smart Gateway)
PC (Wireshark)
(B) Raspberry Pi 3 Model B+ (Power Monitoring)
INA219
User Smartphone
5V Power Source

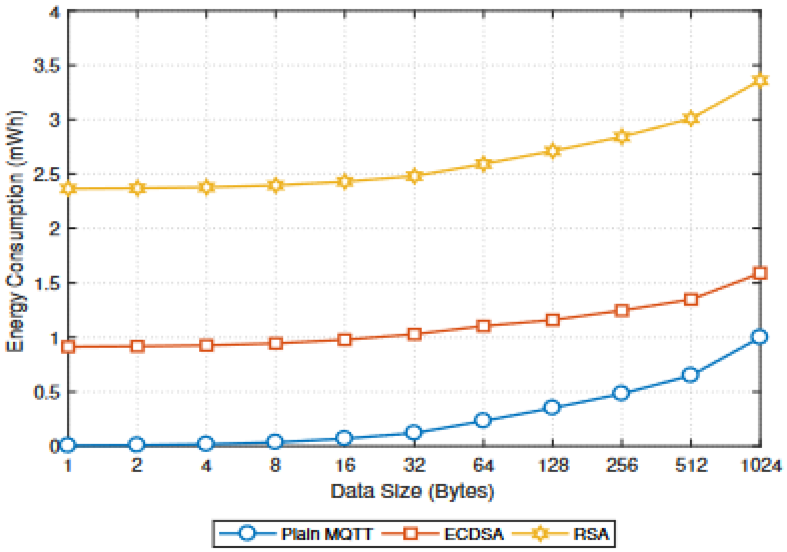
Figure 2: Detailed testbed architecture

Background & Related work

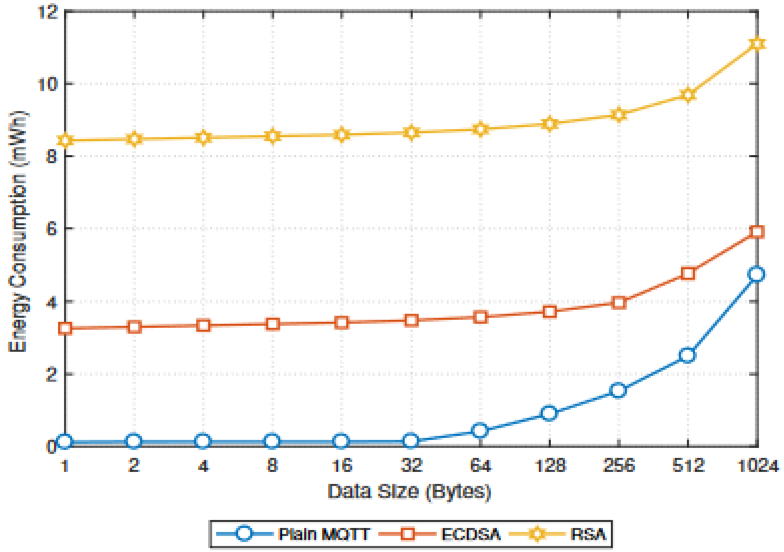
Results

Energy consumptions for data size

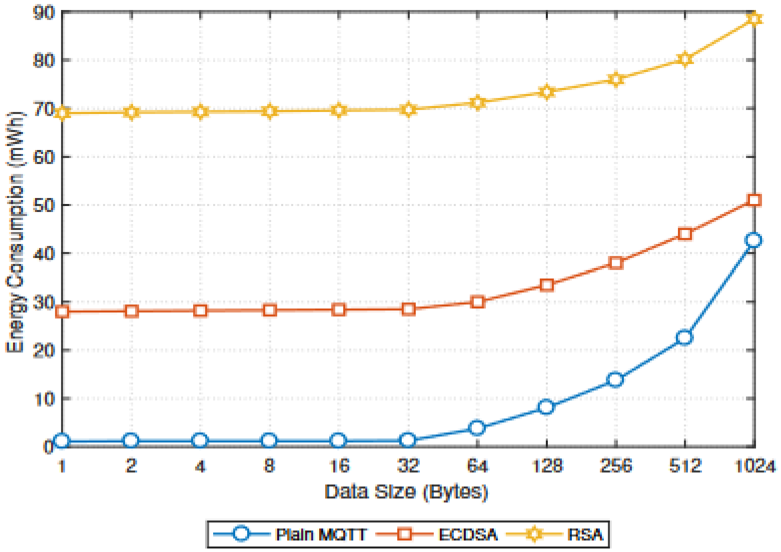
1000 concurrent clients



5000 concurrent clients



10000 concurrent clients



- 54-57 % power consumption decrease when using ECC instead of RSA.
- 35% reduction for 1kb payload.

Background & Related work

Results
CPU utilization

TLS handshake process (unlike regular TCP) consume a lot of CPU. But, unlike HTTP protocol for instance (as used in earlier work) TLS connections established only once for a whole session.

We observe same CPU utilization for all scenarios once the connection is established.

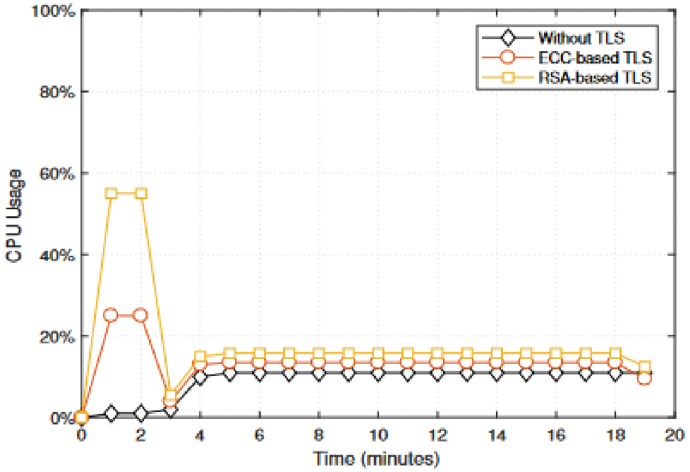


Figure 7: The CPU utilization for scenarios with and without TLS

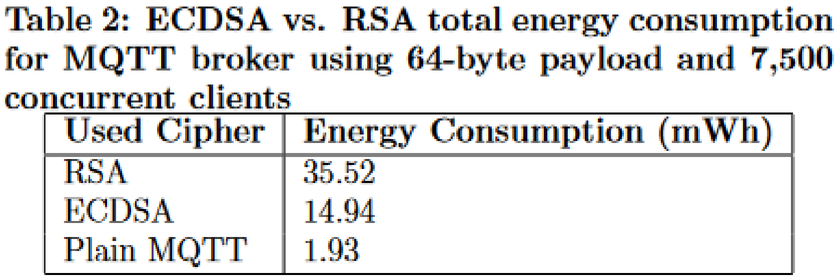


Table 2: ECDSA vs. RSA total energy consumption for MQTT broker using 64-byte payload and 7,500 concurrent clients

Used Cipher	Energy Consumption (mWh)
RSA	35.52
ECDSA	14.94
Plain MQTT	1.93

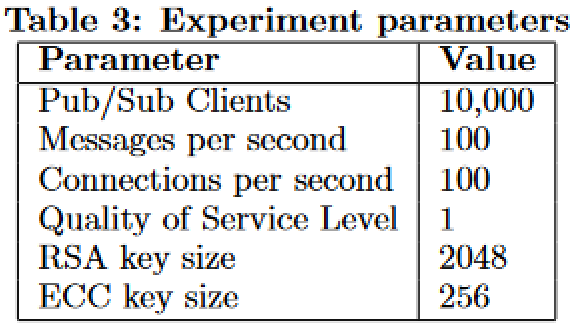


Table 3: Experiment parameters

Parameter	Value
Pub/Sub Clients	10,000
Messages per second	100
Connections per second	100
Quality of Service Level	1
RSA key size	2048
ECC key size	256

Background & Related work

Summary

- ✓ System security issue was well defined and handled great.
- ✓ System scalability proven easy to handle and affordable (cost, power consumption, data traffic, CPU utilization wises).

- X Data gathering, storing and visualize wasn't part of the implementation (cloud integration wasn't considered).
- X Monitoring and data visualize wasn't discussed – visualization tools can handle 1k-10k MQTT clients?

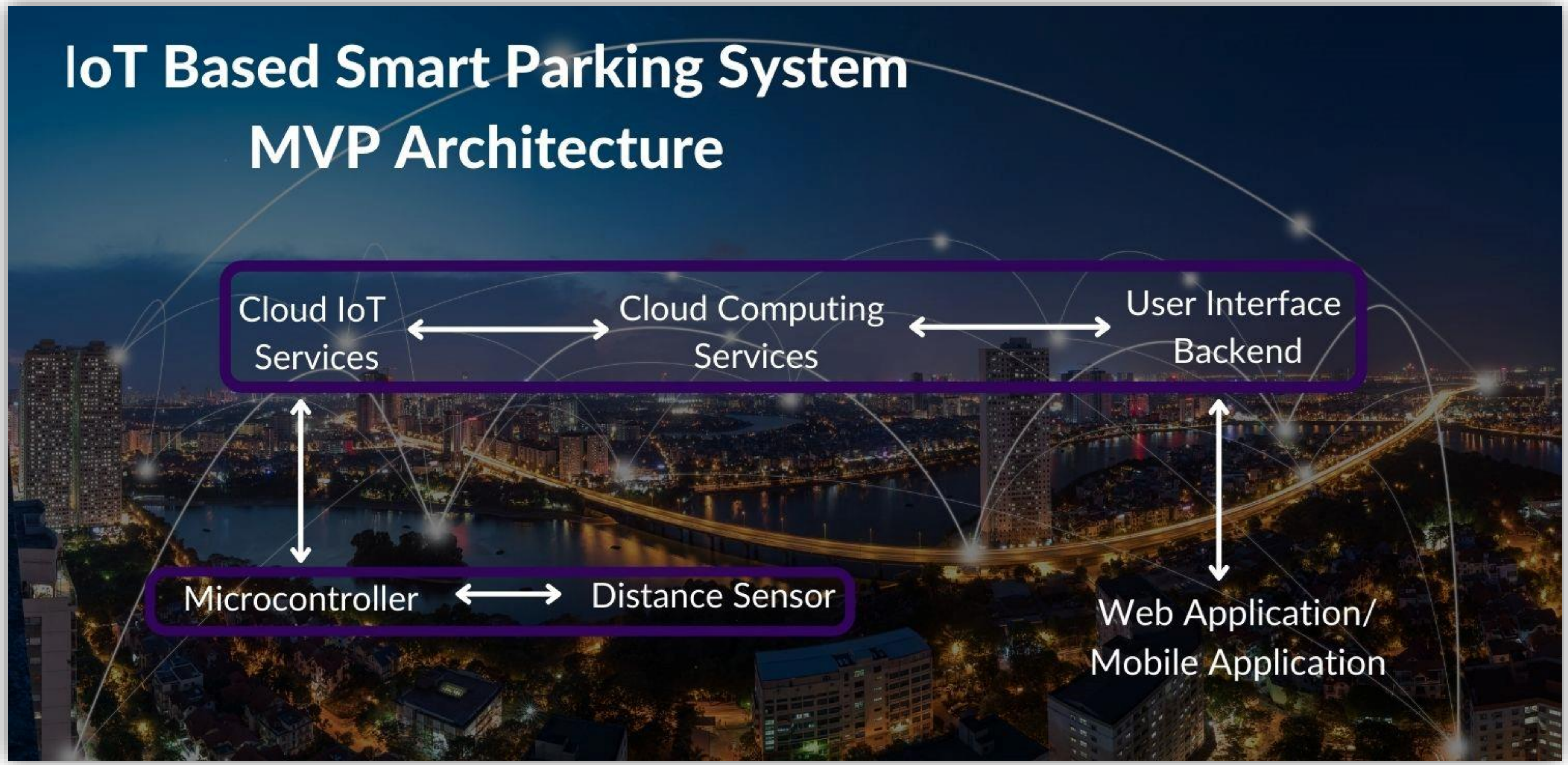
Challenges

Challenges

As we study from previous work, lots of challenges encountered when trying to implement IoT smart parking system

- **Scalability** – can the system's architecture holds up to 100, 1000 and more parking spots? Network traffic and consuming, keeping track, etc…
- **Security** – communications in IoT is a great challenge, this case included. Packets are going back and forth, is the system vulnerable? The data?
- **Price worthy** – is this product worth the effort for all parking lots sizes.
- **Data handling** – Is gathering the whole data possible? How can we store and mining it correctly?

System Architecture



IoT Based Smart Parking System
MVP Architecture

Cloud IoT Services ⟷ Cloud Computing Services ⟷ User Interface Backend

Cloud IoT Services ⟷ Microcontroller

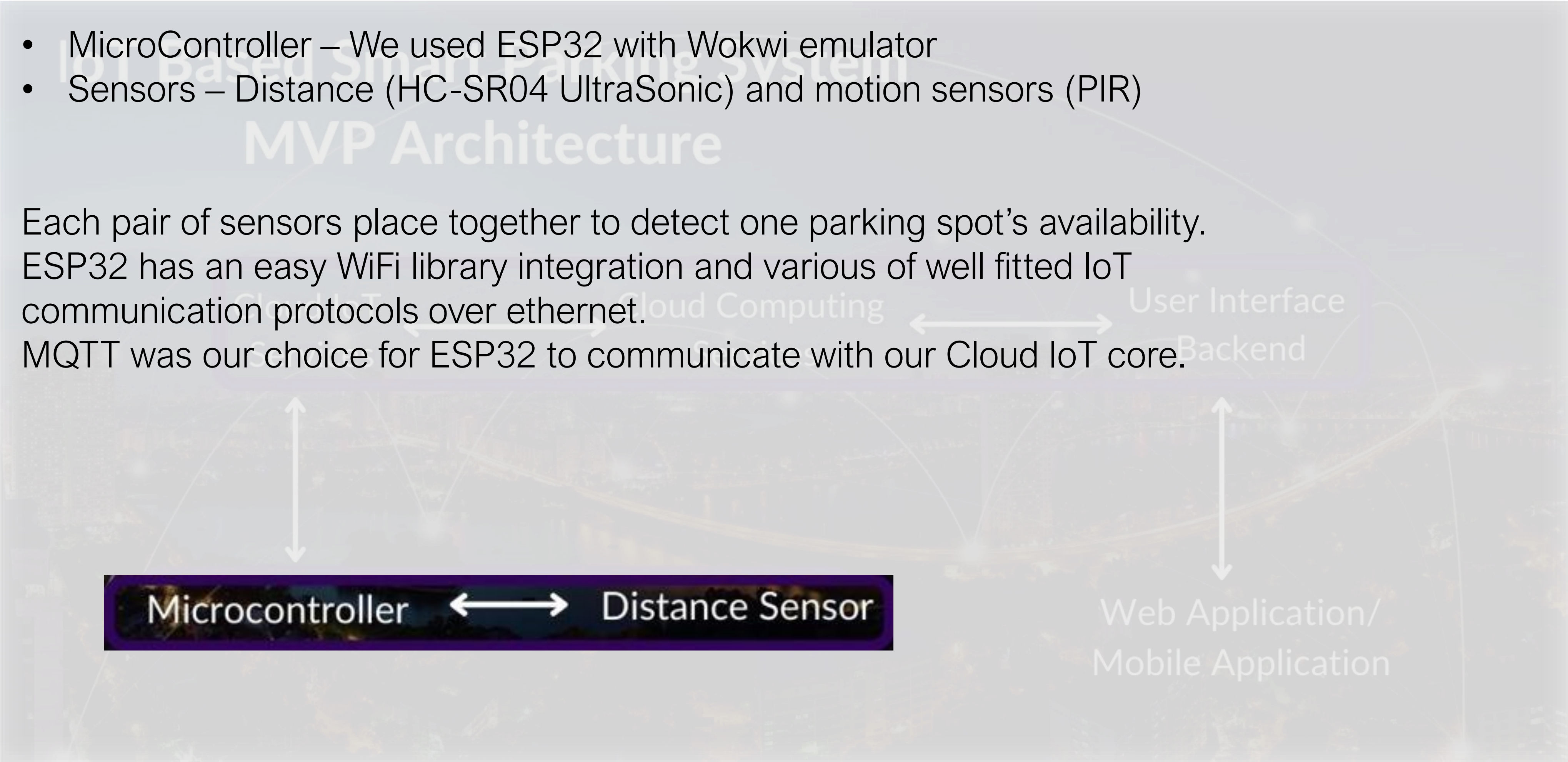
Microcontroller ⟷ Distance Sensor

User Interface Backend ⟷ Web Application/ Mobile Application

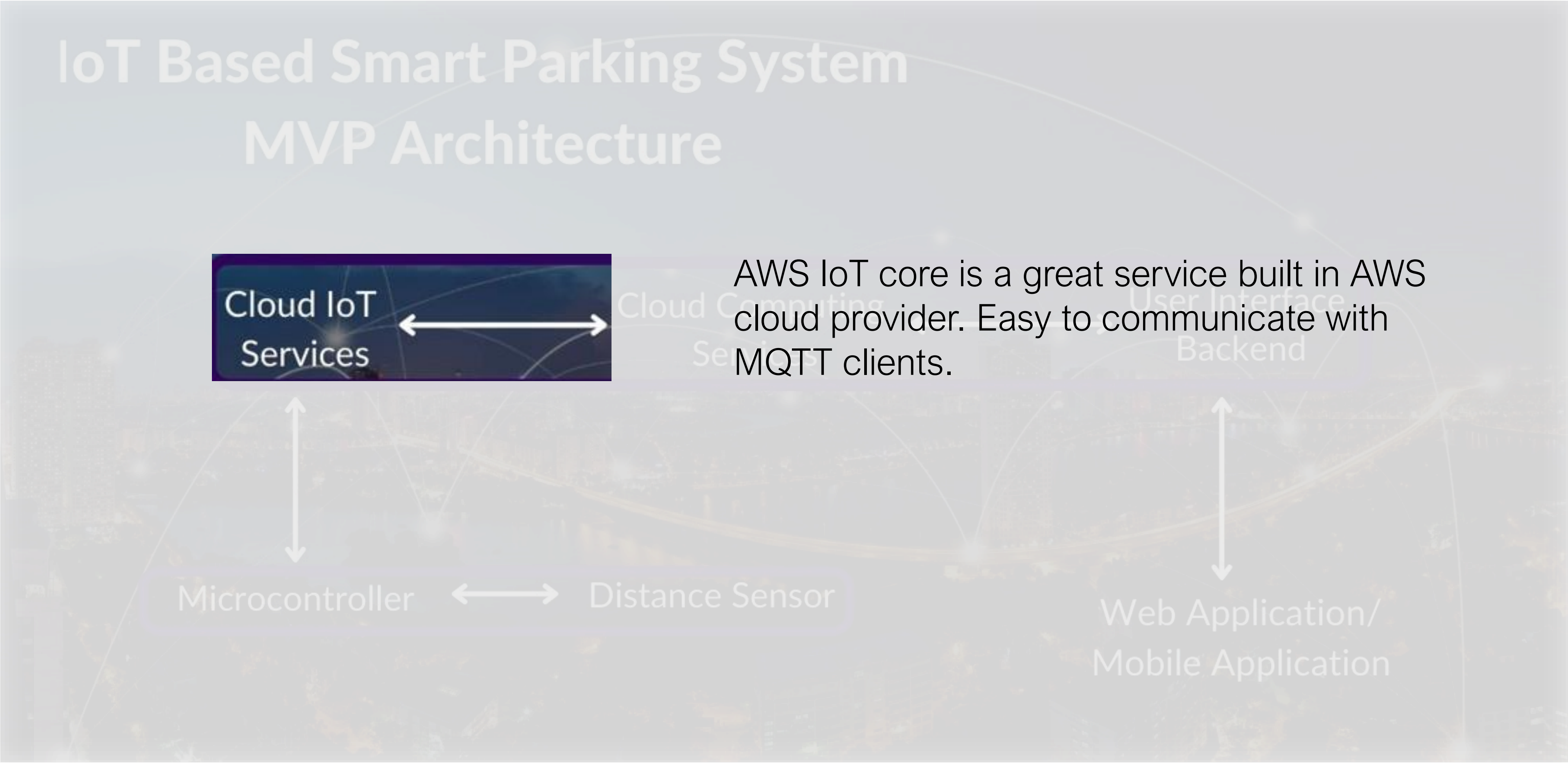
System Architecture

- MicroController – We used ESP32 with Wokwi emulator
- Sensors – Distance (HC-SR04 UltraSonic) and motion sensors (PIR)

Each pair of sensors place together to detect one parking spot’s availability.
ESP32 has an easy WiFi library integration and various of well fitted IoT communication protocols over ethernet.
MQTT was our choice for ESP32 to communicate with our Cloud IoT core.

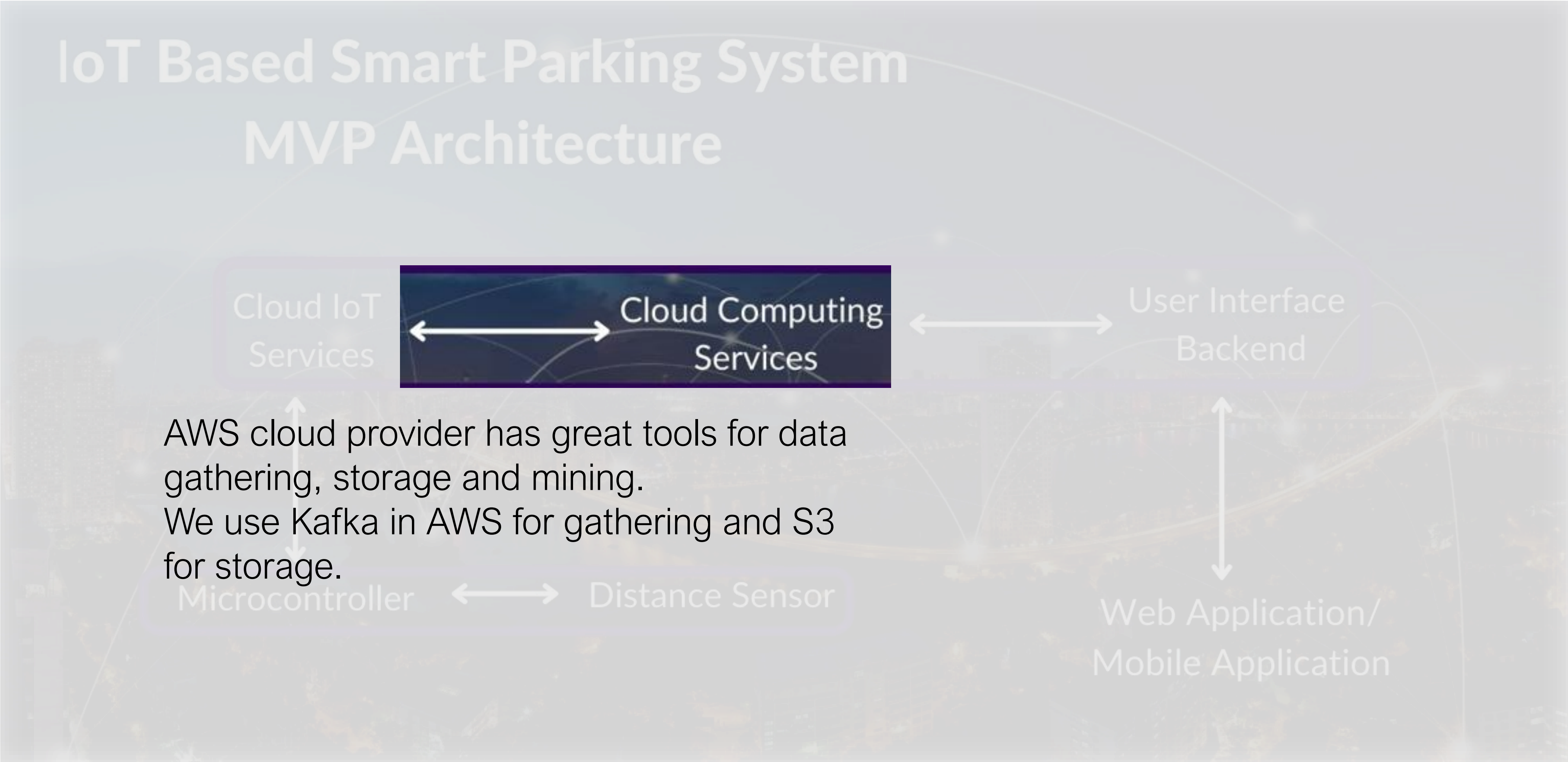


System Architecture



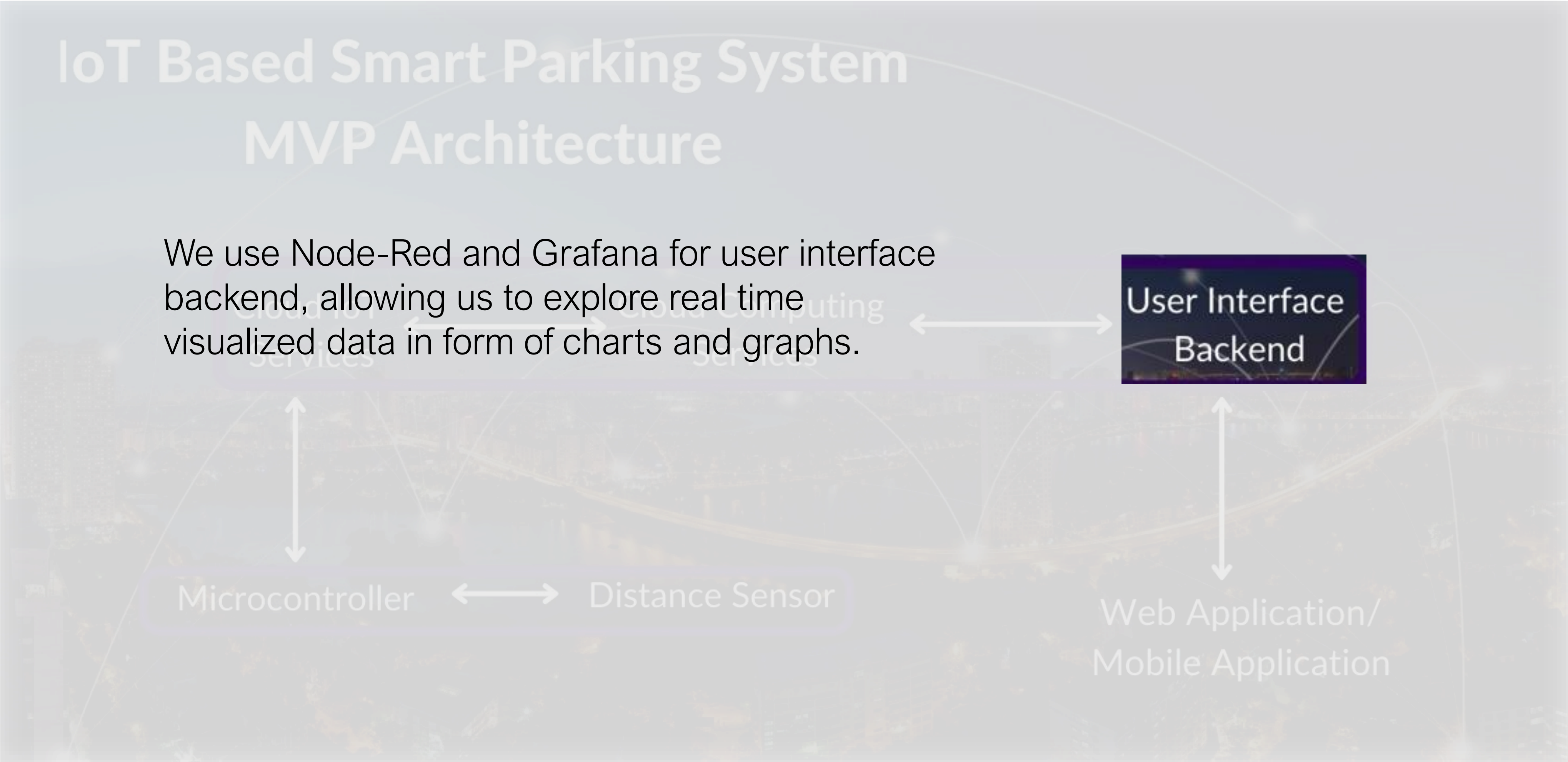
AWS IoT core is a great service built in AWS cloud provider. Easy to communicate with MQTT clients.

System Architecture



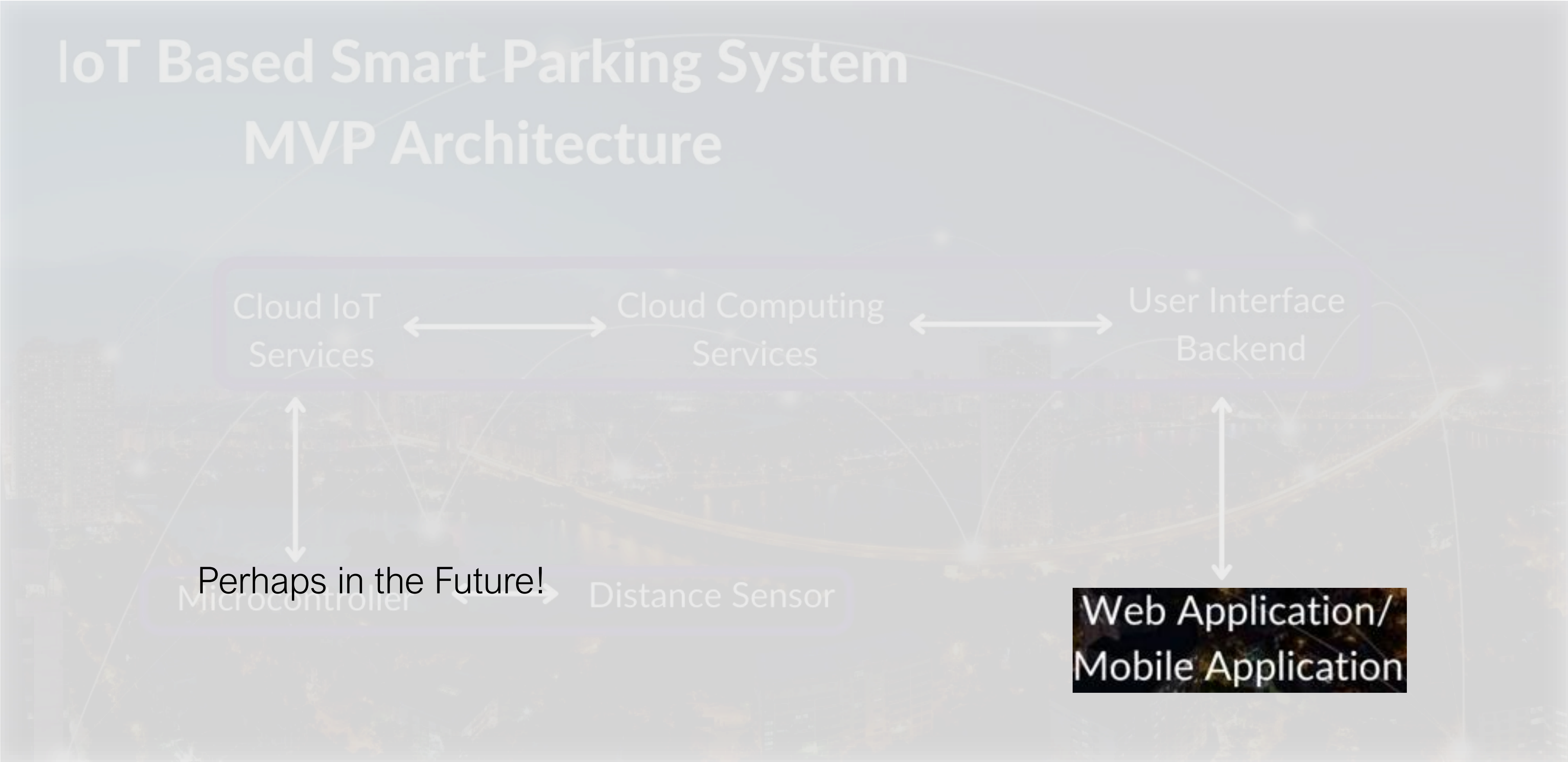
AWS cloud provider has great tools for data gathering, storage and mining.
We use Kafka in AWS for gathering and S3 for storage.

System Architecture



We use Node-Red and Grafana for user interface backend, allowing us to explore real time visualized data in form of charts and graphs.

System Architecture

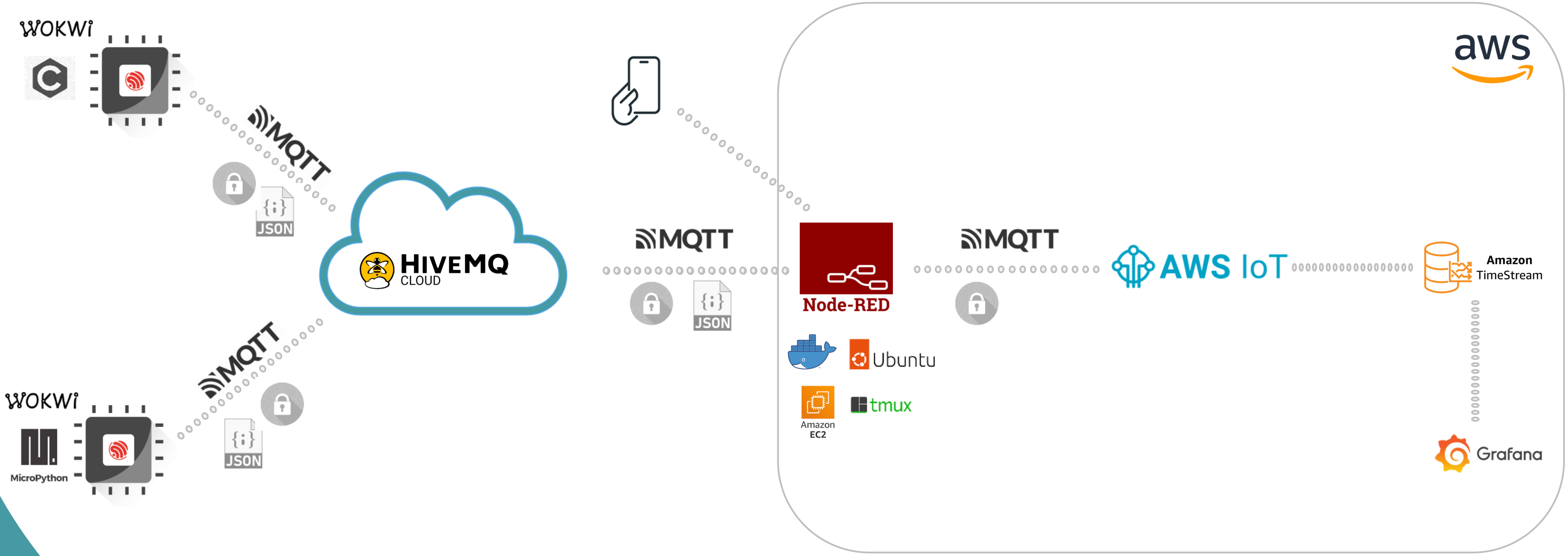


IoT Based Smart Parking System

MVP Architecture

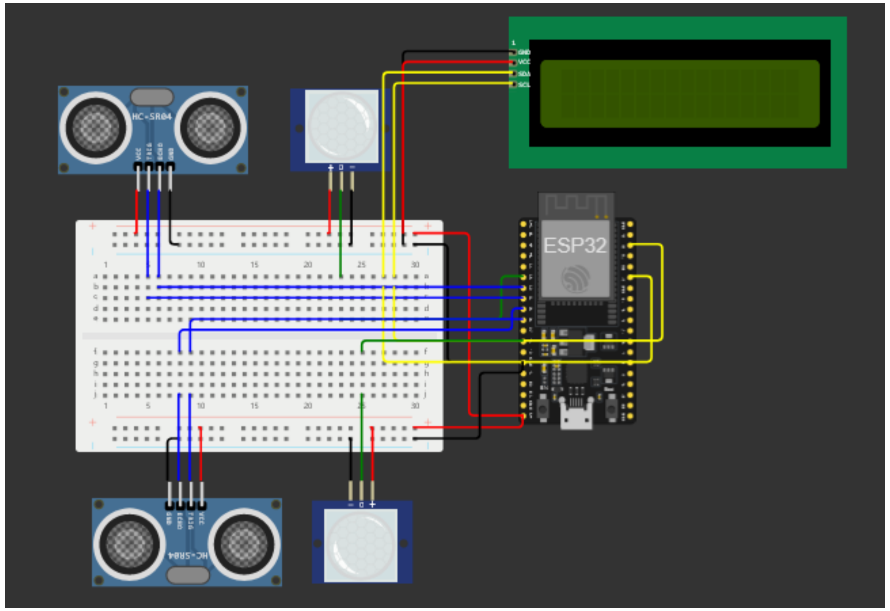
Perhaps in the Future!

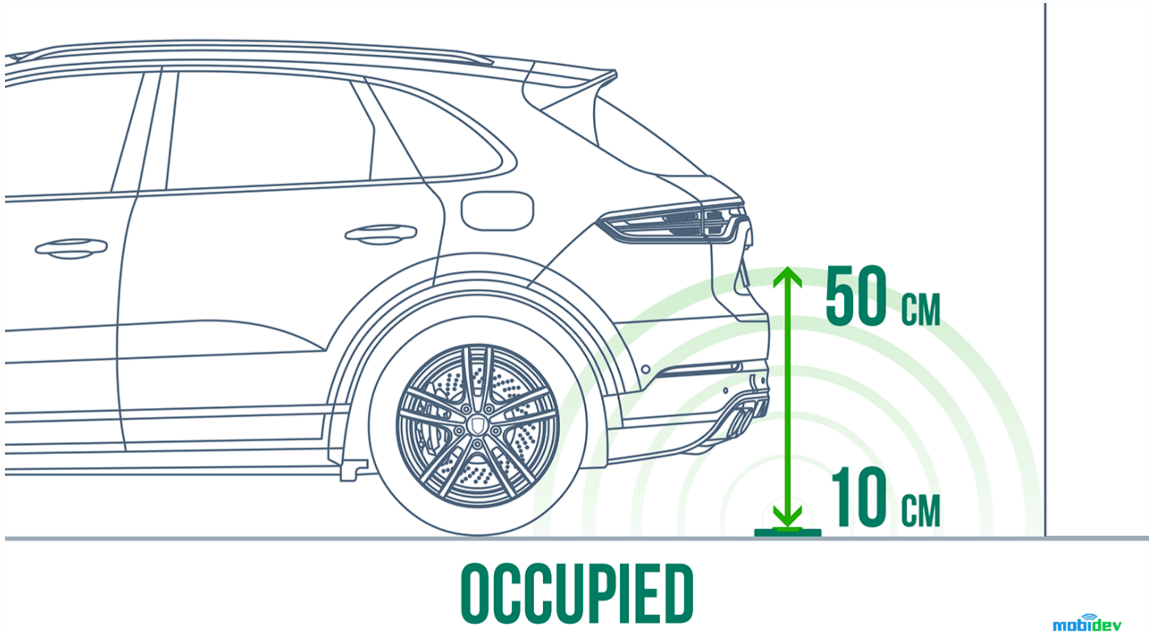
Flow


WOKWi
MQTT
JSON
WOKWi
MicroPython
MQTT
JSON
HIVEMQ
CLOUD
MQTT
JSON
aws
Node-RED
Ubuntu
Amazon
EC2
tmux
MQTT
AWS IoT
Amazon
TimeStream
Grafana

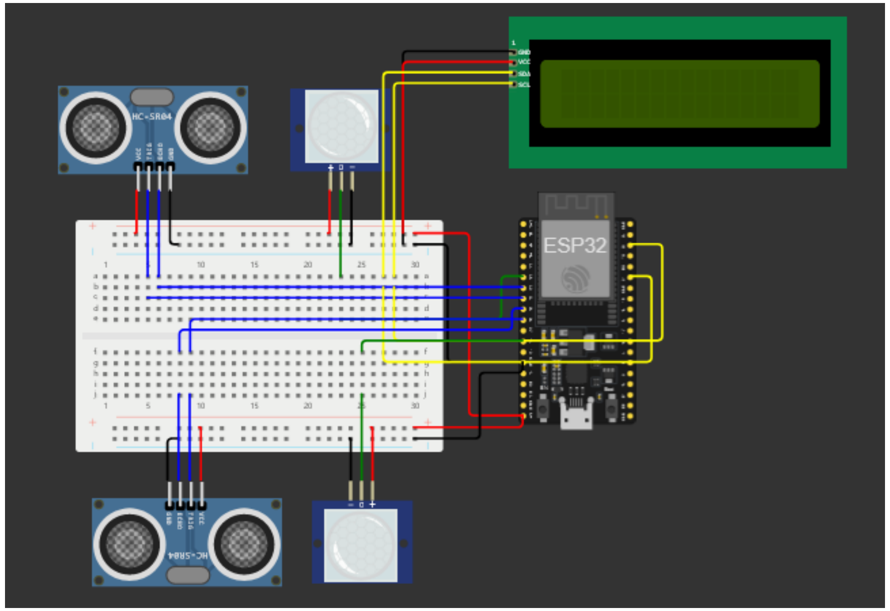
Parking Sensor

Parking Sensor





Parking Sensor

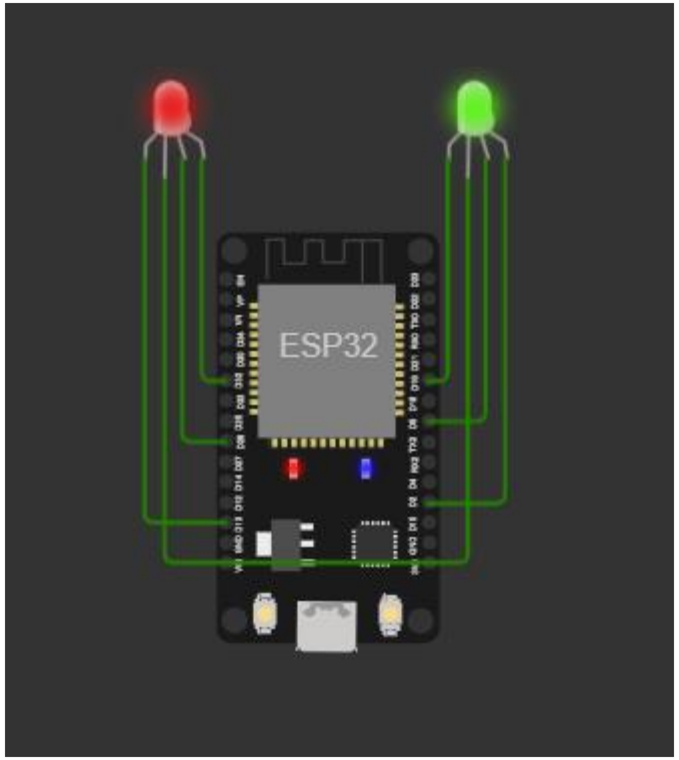


- MQTT protocol based.
- This is one client, holds 2 parking spots (2 pair of sensors).
- Client connected to HiveMQ Cloud MQTT broker, over secured port 8883, using private key, CA and certification - all are local files.
- Client publish to data/parking and subscribe to topic/clients (for future actuators and such).

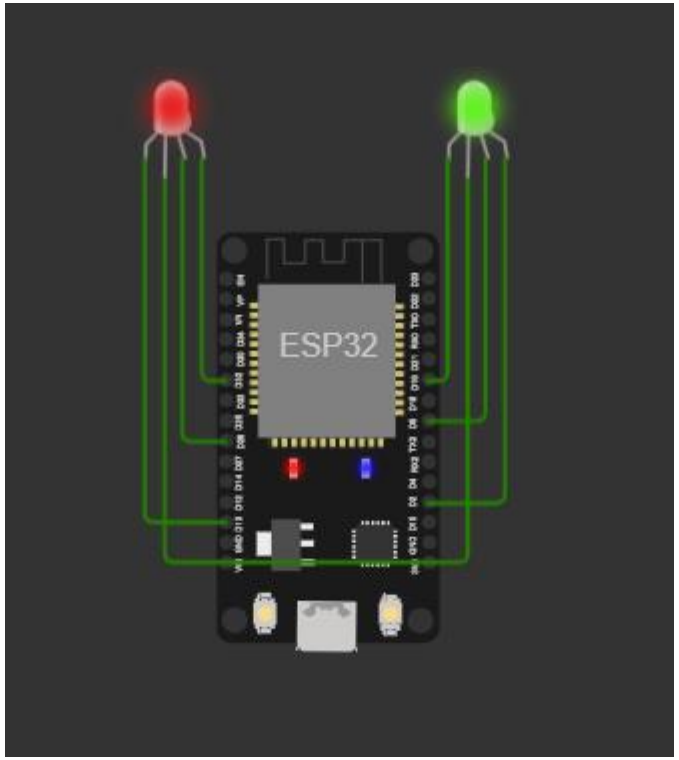
- Json client publish example:

```
{
        "clientid": "ESP8266Client-1341",
        "sensor_number": "1“
        "spot": "1",
        "status": "taken",
}
```

Led Sensor



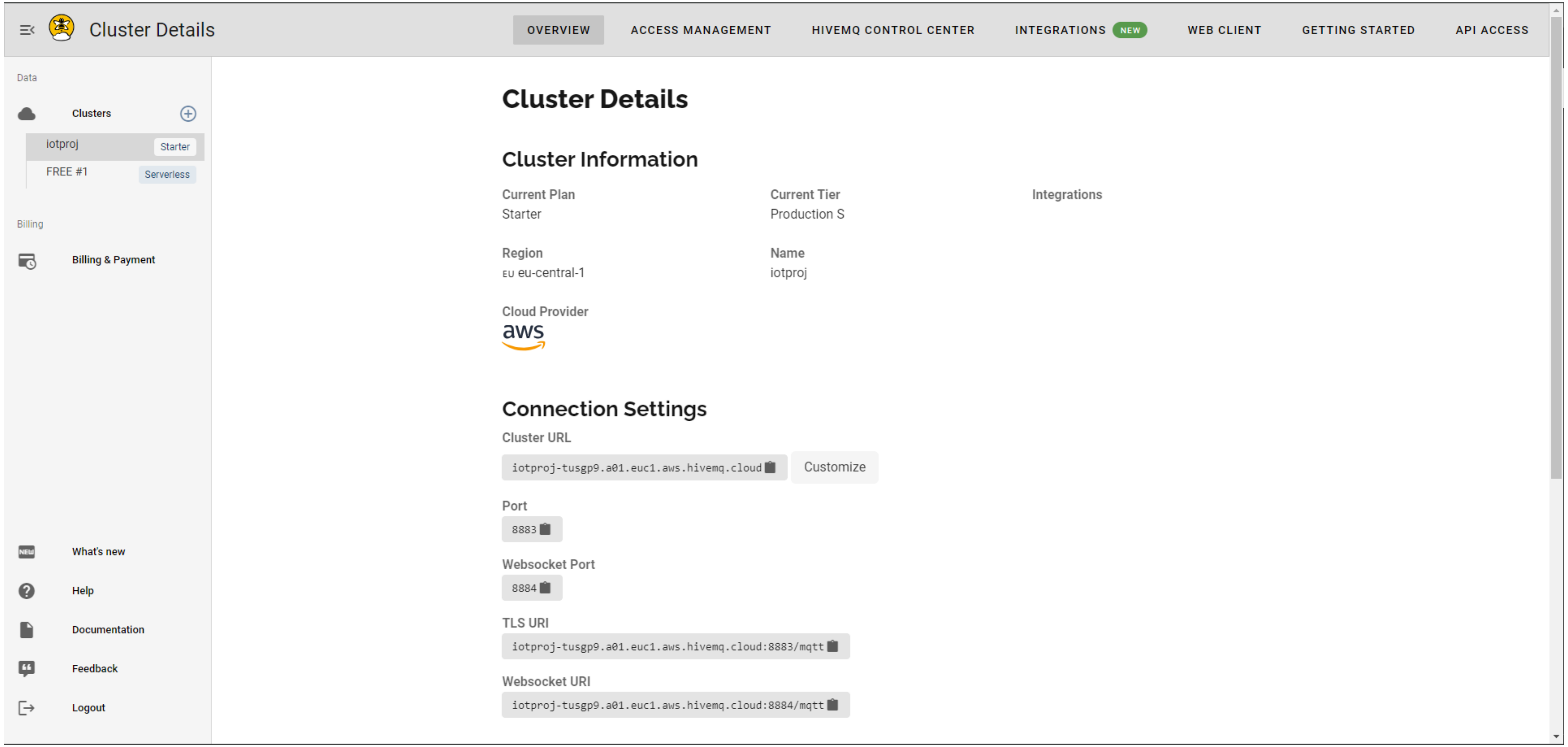
Led Sensor



- MQTT protocol based.
- This is one client, holds 2 led sensors for 2 parking spots.
- Write in Micro Python
- Client connected to HiveMQ Cloud MQTT broker, over secured port 8883, using private key, CA and certification - all are local files.
- Client listen to topic: data/parking and fetch which spot and which status the led need to be
- Json client publish example:

```
{
        "clientid": "ESP8266Client-1341",
        "sensor_number": "1“
        "spot": "1",
        "status": "taken",
}
```

MQTT Broker



Cluster Details

Cluster Information

Current Plan
Starter

Current Tier
Production S

Integrations

Region
EU eu-central-1

Name
iotproj

Cloud Provider
aws

Connection Settings

Cluster URL
`iotproj-tusgp9.a01.euc1.aws.hivemq.cloud`

Port
`8883`

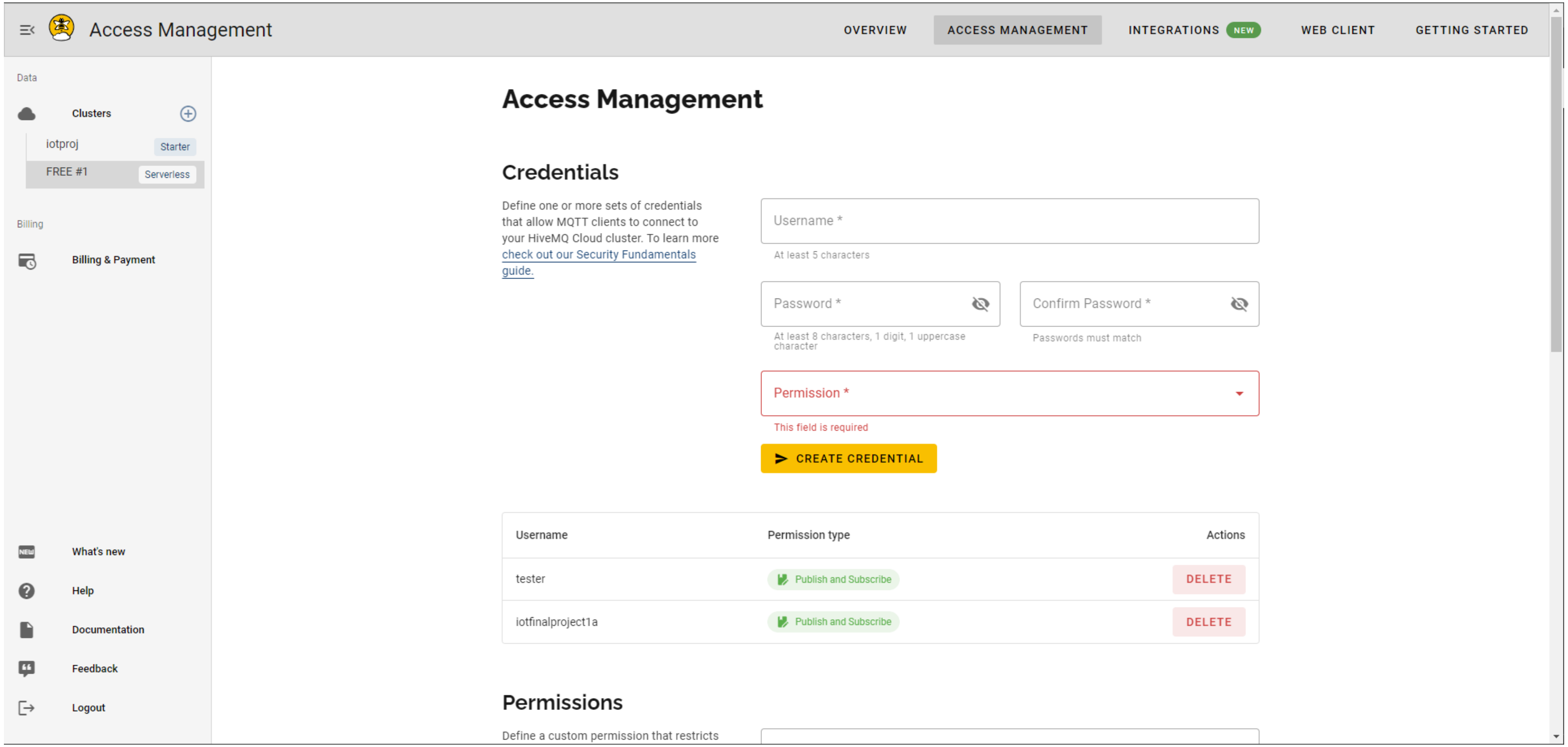
Websocket Port
`8884`

TLS URI
`iotproj-tusgp9.a01.euc1.aws.hivemq.cloud:8883/mqtt`

Websocket URI
`iotproj-tusgp9.a01.euc1.aws.hivemq.cloud:8884/mqtt`

HIVEMQ CLOUD

MQTT Broker



Access Management

Credentials

Define one or more sets of credentials that allow MQTT clients to connect to your HiveMQ Cloud cluster. To learn more check out our Security Fundamentals guide.

Username *

At least 5 characters

Password *

At least 8 characters, 1 digit, 1 uppercase character

Confirm Password *

Passwords must match

Permission *

This field is required

CREATE CREDENTIAL

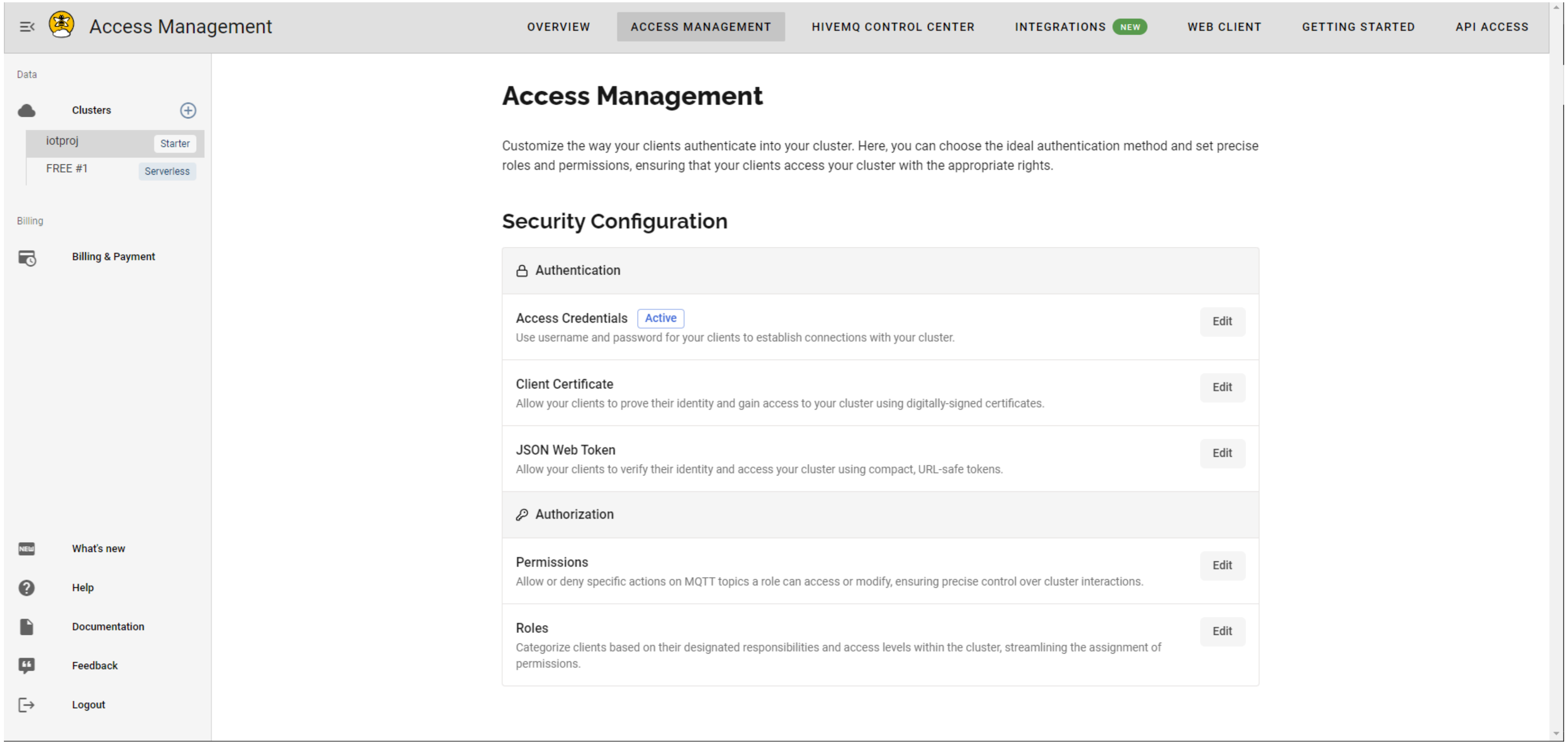
Username	Permission type	Actions
tester	Publish and Subscribe	DELETE
iotfinalproject1a	Publish and Subscribe	DELETE

Permissions

Define a custom permission that restricts

HIVEMQ CLOUD

MQTT Broker



Access Management

Customize the way your clients authenticate into your cluster. Here, you can choose the ideal authentication method and set precise roles and permissions, ensuring that your clients access your cluster with the appropriate rights.

Security Configuration

Authentication

Access Credentials Active — Edit
Use username and password for your clients to establish connections with your cluster.

Client Certificate — Edit
Allow your clients to prove their identity and gain access to your cluster using digitally-signed certificates.

JSON Web Token — Edit
Allow your clients to verify their identity and access your cluster using compact, URL-safe tokens.

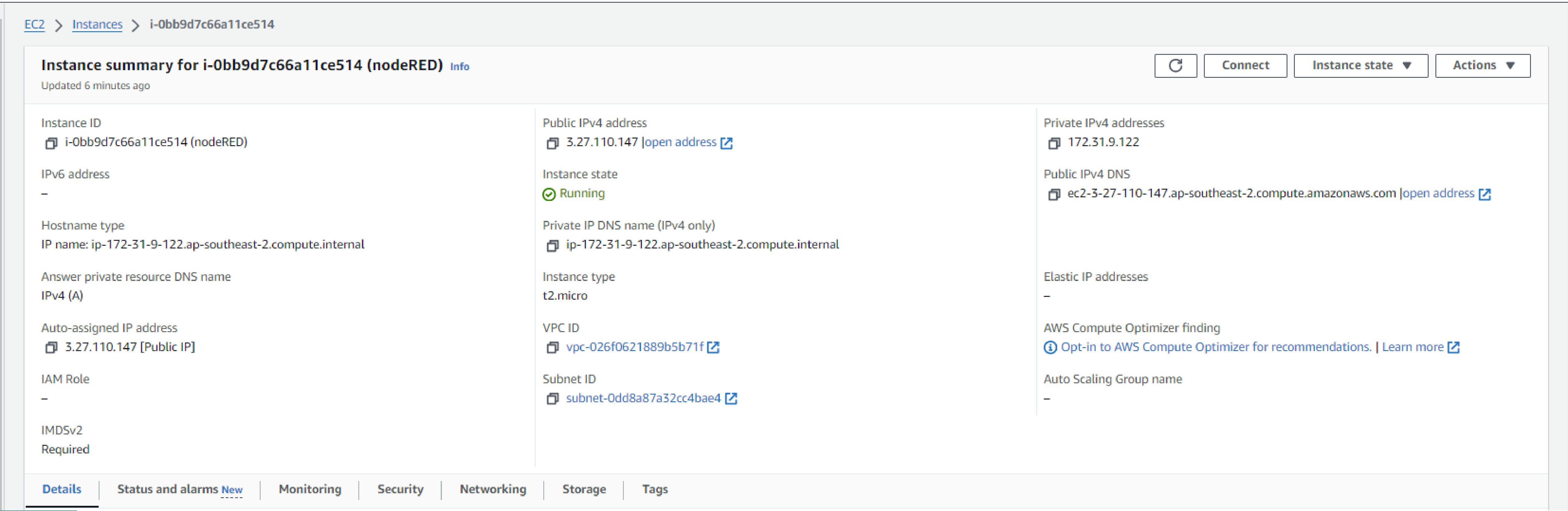
Authorization

Permissions — Edit
Allow or deny specific actions on MQTT topics a role can access or modify, ensuring precise control over cluster interactions.

Roles — Edit
Categorize clients based on their designated responsibilities and access levels within the cluster, streamlining the assignment of permissions.

HIVEMQ CLOUD

AWS EC2 Instance



Instance summary for i-0bb9d7c66a11ce514 (nodeRED) Info

Updated 6 minutes ago

Instance ID
i-0bb9d7c66a11ce514 (nodeRED)

IPv6 address
–

Hostname type
IP name: ip-172-31-9-122.ap-southeast-2.compute.internal

Answer private resource DNS name
IPv4 (A)

Auto-assigned IP address
3.27.110.147 [Public IP]

IAM Role
–

IMDSv2
Required

Public IPv4 address
3.27.110.147 |open address

Instance state
Running

Private IP DNS name (IPv4 only)
ip-172-31-9-122.ap-southeast-2.compute.internal

Instance type
t2.micro

VPC ID
vpc-026f0621889b5b71f

Subnet ID
subnet-0dd8a87a32cc4bae4

Private IPv4 addresses
172.31.9.122

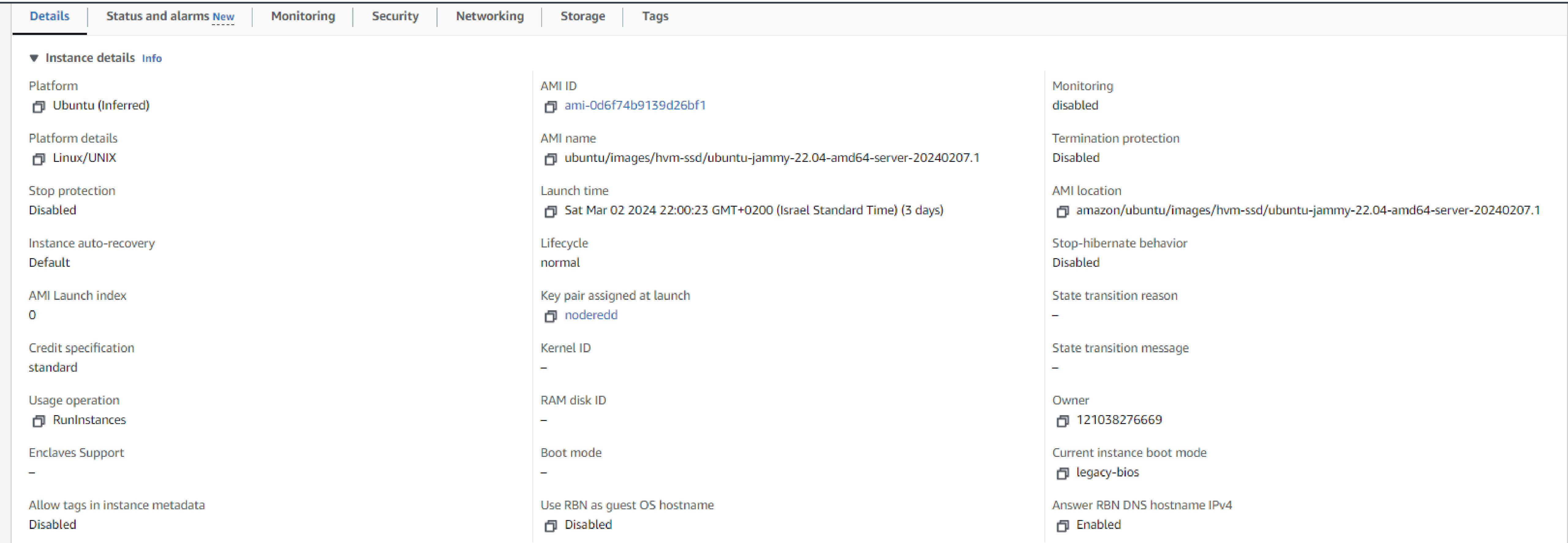
Public IPv4 DNS
ec2-3-27-110-147.ap-southeast-2.compute.amazonaws.com |open address

Elastic IP addresses
–

AWS Compute Optimizer finding
Opt-in to AWS Compute Optimizer for recommendations. | Learn more

Auto Scaling Group name
–

AWS EC2 Instance



▼ Instance details Info

Platform
Ubuntu (Inferred)

Platform details
Linux/UNIX

Stop protection
Disabled

Instance auto-recovery
Default

AMI Launch index
0

Credit specification
standard

Usage operation
RunInstances

Enclaves Support
–

Allow tags in instance metadata
Disabled

AMI ID
ami-0d6f74b9139d26bf1

AMI name
ubuntu/images/hvm-ssd/ubuntu-jammy-22.04-amd64-server-20240207.1

Launch time
Sat Mar 02 2024 22:00:23 GMT+0200 (Israel Standard Time) (3 days)

Lifecycle
normal

Key pair assigned at launch
noderedd

Kernel ID
–

RAM disk ID
–

Boot mode
–

Use RBN as guest OS hostname
Disabled

Monitoring
disabled

Termination protection
Disabled

AMI location
amazon/ubuntu/images/hvm-ssd/ubuntu-jammy-22.04-amd64-server-20240207.1

Stop-hibernate behavior
Disabled

State transition reason
–

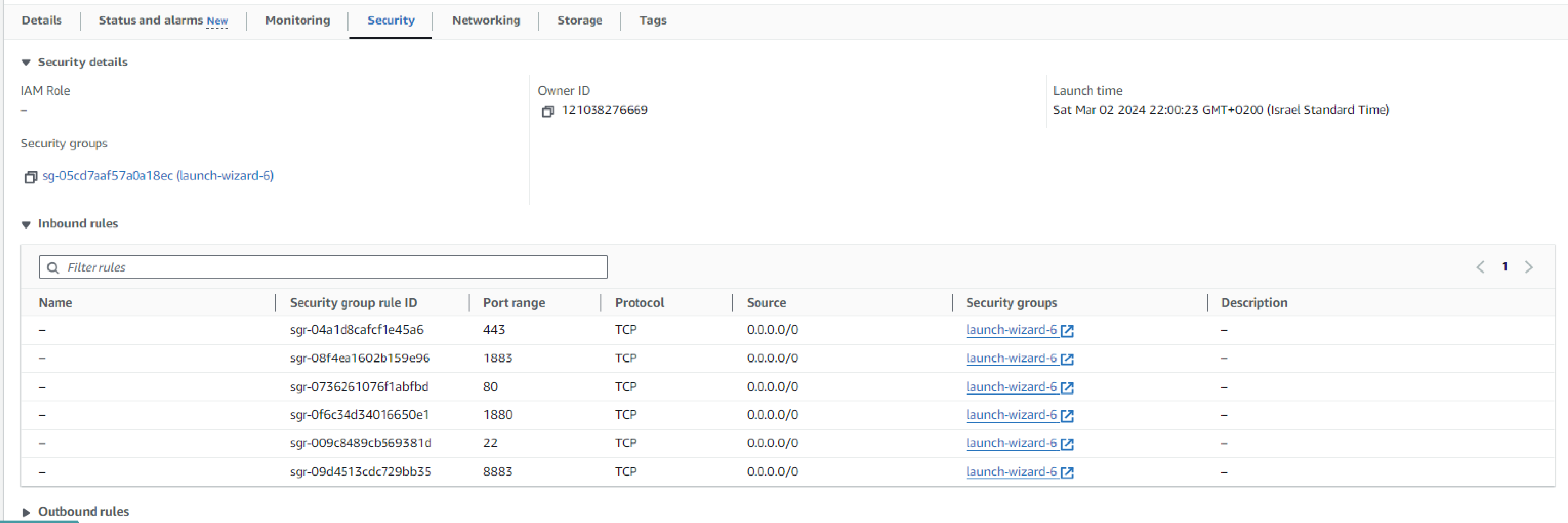
State transition message
–

Owner
121038276669

Current instance boot mode
legacy-bios

Answer RBN DNS hostname IPv4
Enabled

AWS EC2 Instance



Security details

IAM Role
–

Owner ID
121038276669

Launch time
Sat Mar 02 2024 22:00:23 GMT+0200 (Israel Standard Time)

Security groups
sg-05cd7aaf57a0a18ec (launch-wizard-6)

Inbound rules

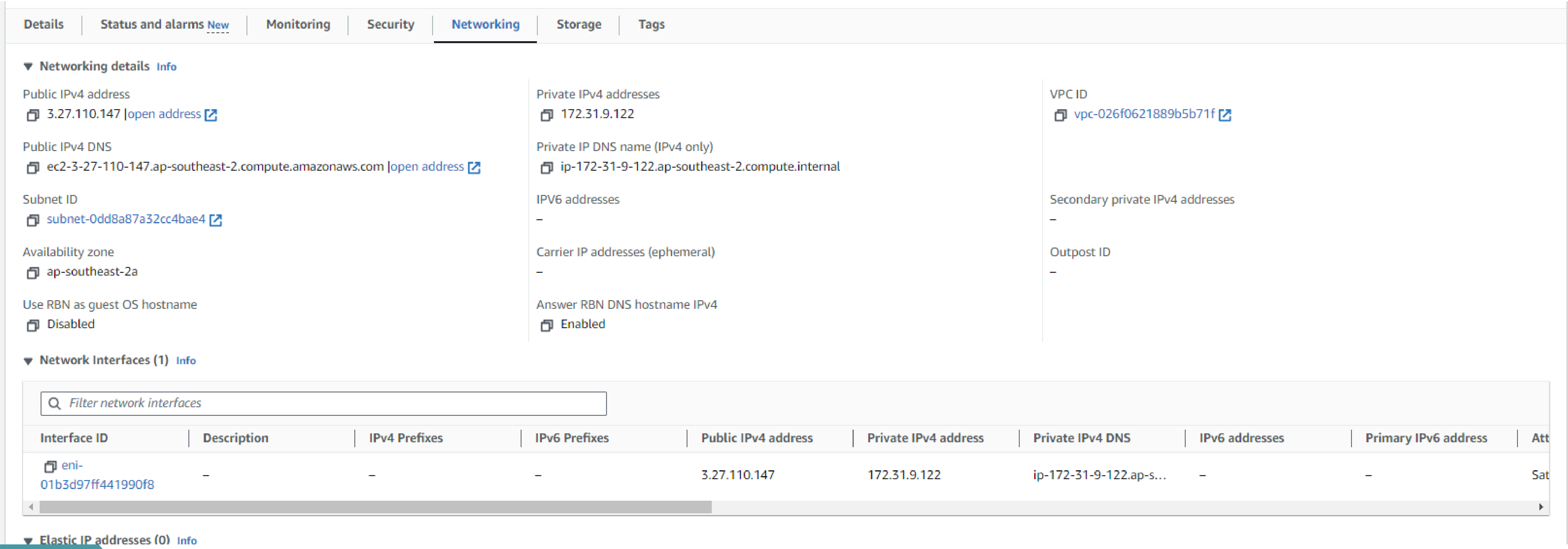
Name	Security group rule ID	Port range	Protocol	Source	Security groups	Description
–	sgr-04a1d8cafcf1e45a6	443	TCP	0.0.0.0/0	launch-wizard-6	–
–	sgr-08f4ea1602b159e96	1883	TCP	0.0.0.0/0	launch-wizard-6	–
–	sgr-0736261076f1abfbd	80	TCP	0.0.0.0/0	launch-wizard-6	–
–	sgr-0f6c34d34016650e1	1880	TCP	0.0.0.0/0	launch-wizard-6	–
–	sgr-009c8489cb569381d	22	TCP	0.0.0.0/0	launch-wizard-6	–
–	sgr-09d4513cdc729bb35	8883	TCP	0.0.0.0/0	launch-wizard-6	–

Outbound rules

Amazon EC2

aws

AWS EC2 Instance



▼ Networking details Info

Public IPv4 address
3.27.110.147 |open address

Public IPv4 DNS
ec2-3-27-110-147.ap-southeast-2.compute.amazonaws.com |open address

Subnet ID
subnet-0dd8a87a32cc4bae4

Availability zone
ap-southeast-2a

Use RBN as guest OS hostname
Disabled

Private IPv4 addresses
172.31.9.122

Private IP DNS name (IPv4 only)
ip-172-31-9-122.ap-southeast-2.compute.internal

IPV6 addresses
–

Carrier IP addresses (ephemeral)
–

Answer RBN DNS hostname IPv4
Enabled

VPC ID
vpc-026f0621889b5b71f

Secondary private IPv4 addresses
–

Outpost ID
–

▼ Network Interfaces (1) Info

Filter network interfaces

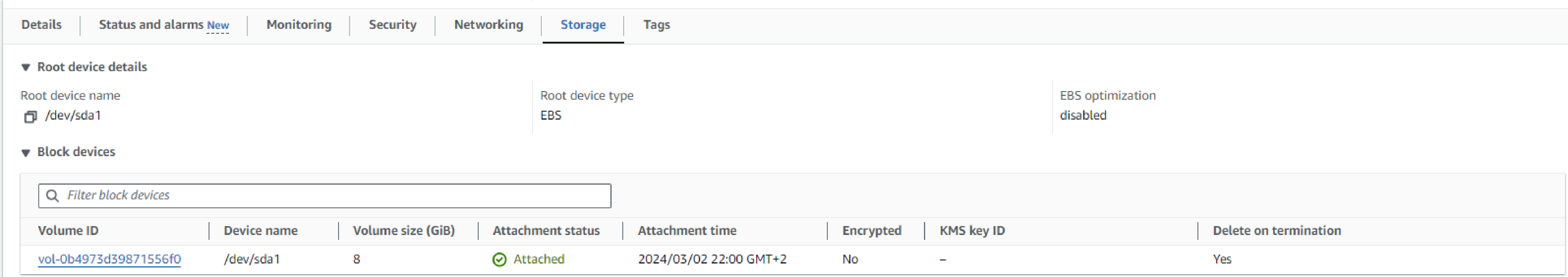
Interface ID	Description	IPv4 Prefixes	IPv6 Prefixes	Public IPv4 address	Private IPv4 address	Private IPv4 DNS	IPv6 addresses	Primary IPv6 address	Att
eni-01b3d97ff441990f8	–	–	–	3.27.110.147	172.31.9.122	ip-172-31-9-122.ap-s...	–	–	Sat

▼ Elastic IP addresses (0) Info

Amazon EC2

aws

AWS EC2 Instance



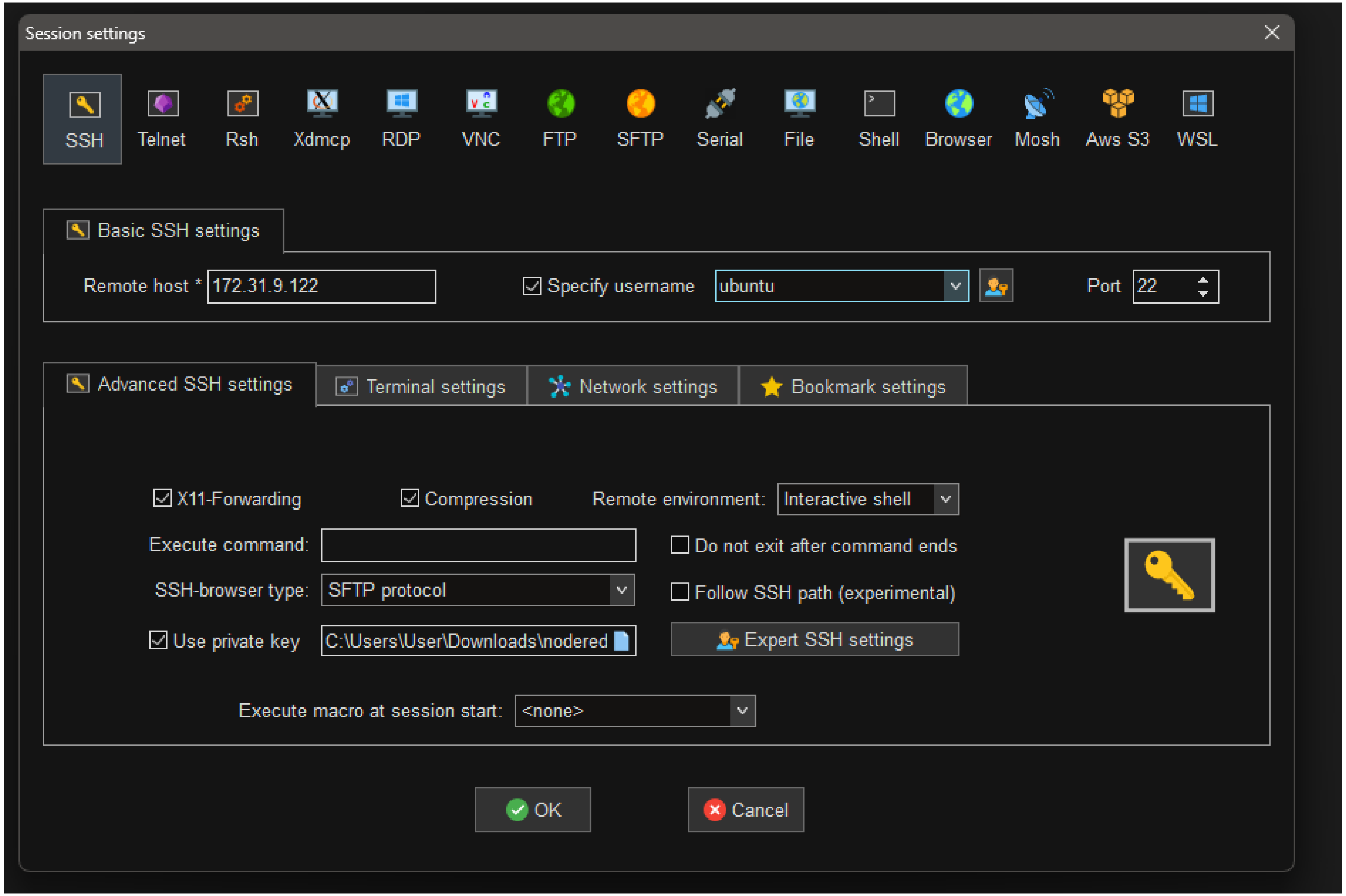
▼ Root device details

Root device name	Root device type	EBS optimization
/dev/sda1	EBS	disabled

▼ Block devices

Volume ID	Device name	Volume size (GiB)	Attachment status	Attachment time	Encrypted	KMS key ID	Delete on termination
vol-0b4973d39871556f0	/dev/sda1	8	✓ Attached	2024/03/02 22:00 GMT+2	No	–	Yes

SSH



Session settings

Basic SSH settings

Remote host * 172.31.9.122

☑ Specify username ubuntu

Port 22

Advanced SSH settings | Terminal settings | Network settings | Bookmark settings

☑ X11-Forwarding

☑ Compression

Remote environment: Interactive shell

Execute command:

□ Do not exit after command ends

SSH-browser type: SFTP protocol

□ Follow SSH path (experimental)

☑ Use private key C:\Users\User\Downloads\nodered

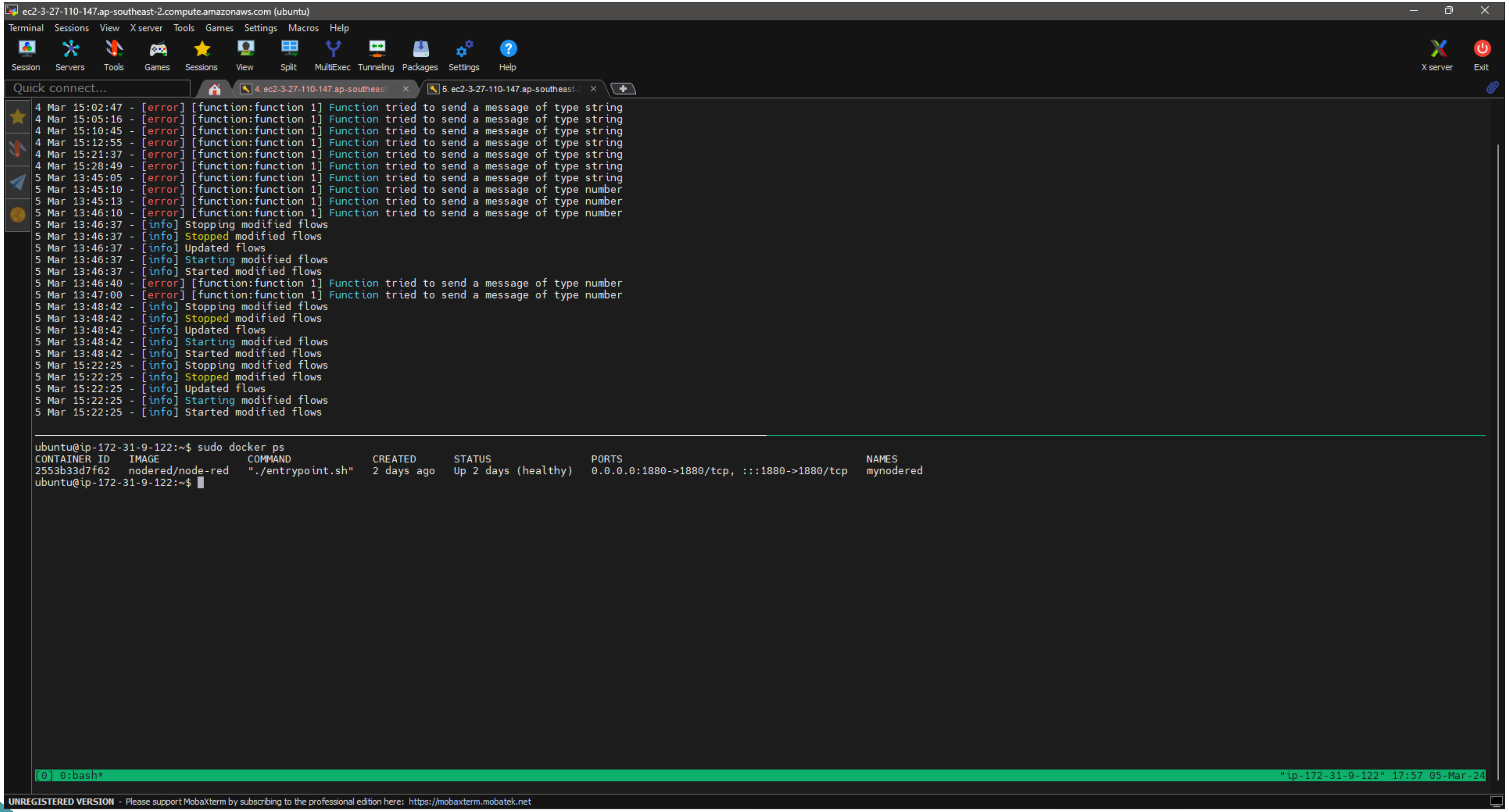
Expert SSH settings

Execute macro at session start: <none>

Amazon EC2

aws

Docker (& tmux)



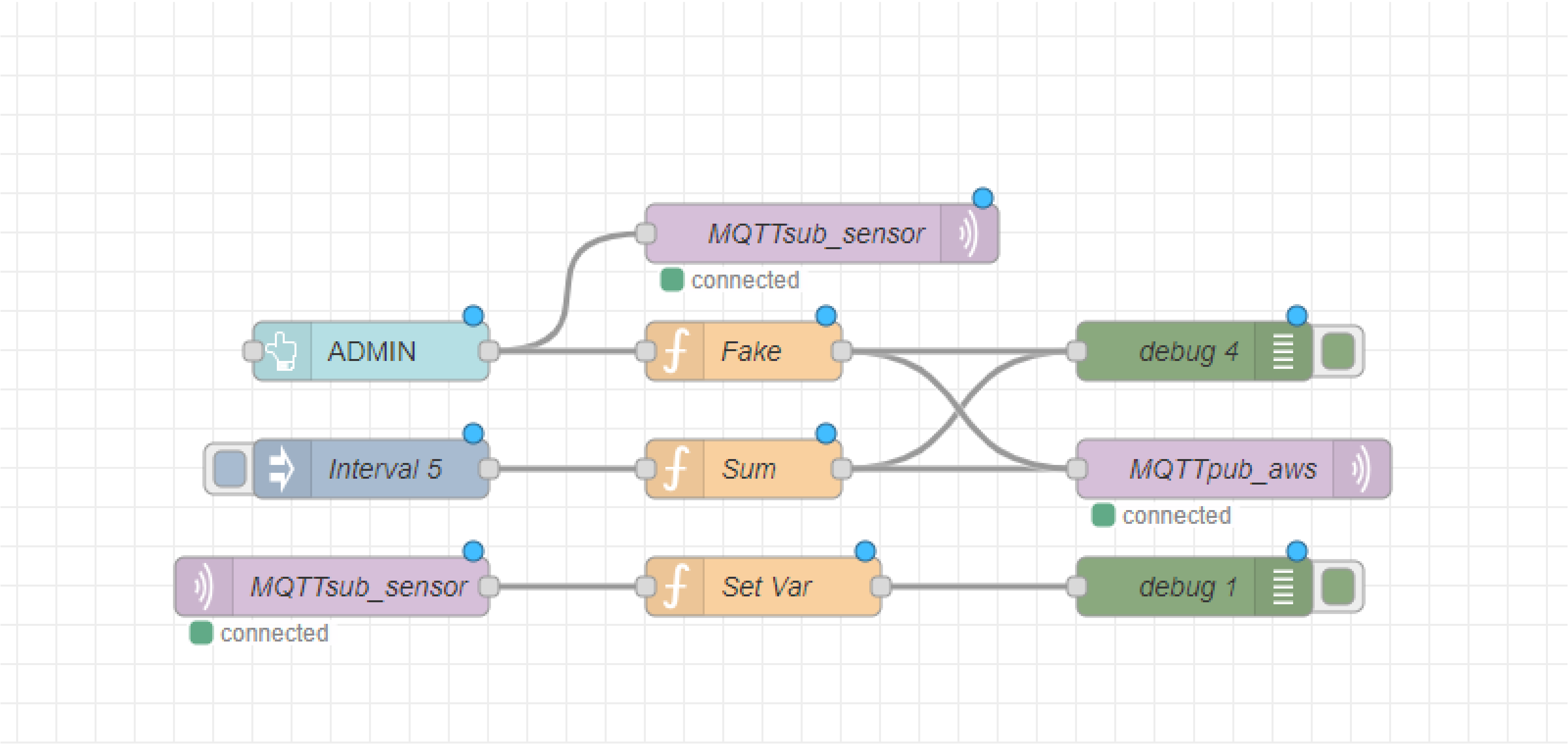
```
4 Mar 15:02:47 - [error] [function:function 1] Function tried to send a message of type string
4 Mar 15:05:16 - [error] [function:function 1] Function tried to send a message of type string
4 Mar 15:10:45 - [error] [function:function 1] Function tried to send a message of type string
4 Mar 15:12:55 - [error] [function:function 1] Function tried to send a message of type string
4 Mar 15:21:37 - [error] [function:function 1] Function tried to send a message of type string
4 Mar 15:28:49 - [error] [function:function 1] Function tried to send a message of type string
5 Mar 13:45:05 - [error] [function:function 1] Function tried to send a message of type string
5 Mar 13:45:10 - [error] [function:function 1] Function tried to send a message of type number
5 Mar 13:45:13 - [error] [function:function 1] Function tried to send a message of type number
5 Mar 13:46:10 - [error] [function:function 1] Function tried to send a message of type number
5 Mar 13:46:37 - [info] Stopping modified flows
5 Mar 13:46:37 - [info] Stopped modified flows
5 Mar 13:46:37 - [info] Updated flows
5 Mar 13:46:37 - [info] Starting modified flows
5 Mar 13:46:37 - [info] Started modified flows
5 Mar 13:46:40 - [error] [function:function 1] Function tried to send a message of type number
5 Mar 13:47:00 - [error] [function:function 1] Function tried to send a message of type number
5 Mar 13:48:42 - [info] Stopping modified flows
5 Mar 13:48:42 - [info] Stopped modified flows
5 Mar 13:48:42 - [info] Updated flows
5 Mar 13:48:42 - [info] Starting modified flows
5 Mar 13:48:42 - [info] Started modified flows
5 Mar 15:22:25 - [info] Stopping modified flows
5 Mar 15:22:25 - [info] Stopped modified flows
5 Mar 15:22:25 - [info] Updated flows
5 Mar 15:22:25 - [info] Starting modified flows
5 Mar 15:22:25 - [info] Started modified flows
```

```
ubuntu@ip-172-31-9-122:~$ sudo docker ps
CONTAINER ID   IMAGE               COMMAND             CREATED        STATUS                 PORTS                                       NAMES
2553b33d7f62   nodered/node-red    "./entrypoint.sh"   2 days ago     Up 2 days (healthy)    0.0.0.0:1880->1880/tcp, :::1880->1880/tcp   mynodered
ubuntu@ip-172-31-9-122:~$ 
```

```
[0] 0:bash*                                                                     "ip-172-31-9-122" 17:57 05-Mar-24
```

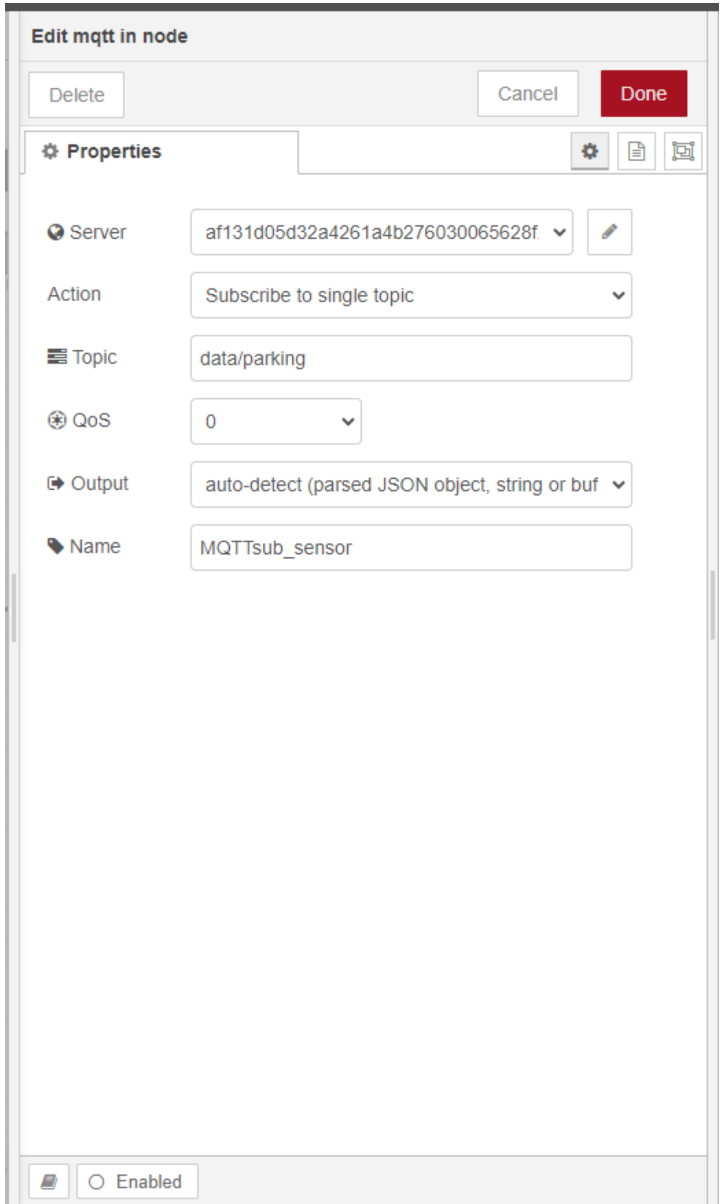
UNREGISTERED VERSION - Please support MobaXterm by subscribing to the professional edition here: https://mobaxterm.mobatek.net

NodeRED



Node-RED

NodeRED



Edit mqtt in node

Server: af131d05d32a4261a4b276030065628f

Action: Subscribe to single topic

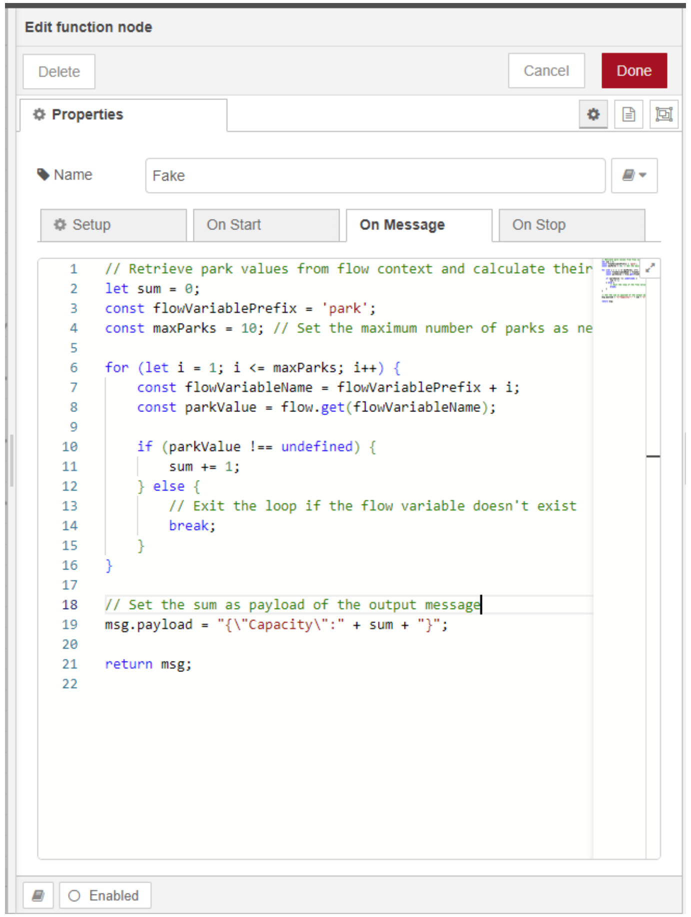
Topic: data/parking

QoS: 0

Output: auto-detect (parsed JSON object, string or buf

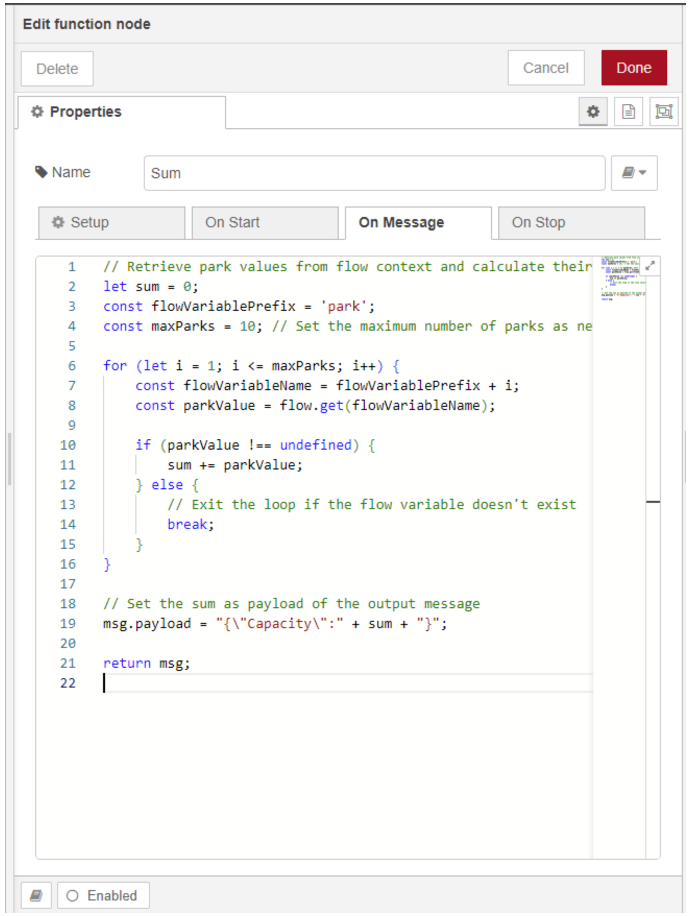
Name: MQTTsub_sensor

NodeRED



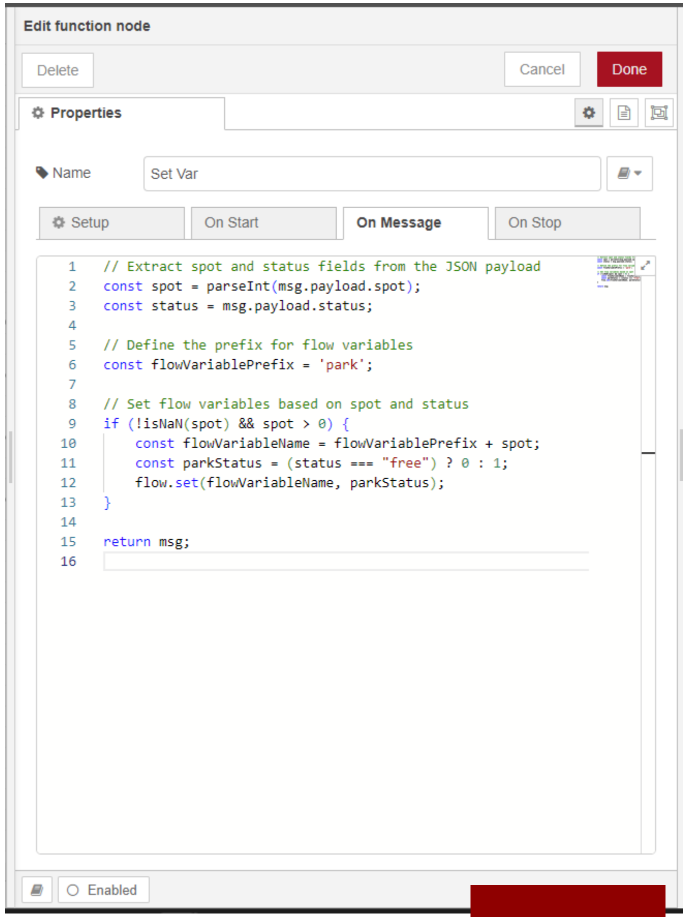
Name: Fake

```
// Retrieve park values from flow context and calculate their
let sum = 0;
const flowVariablePrefix = 'park';
const maxParks = 10; // Set the maximum number of parks as ne

for (let i = 1; i <= maxParks; i++) {
    const flowVariableName = flowVariablePrefix + i;
    const parkValue = flow.get(flowVariableName);

    if (parkValue !== undefined) {
        sum += 1;
    } else {
        // Exit the loop if the flow variable doesn't exist
        break;
    }
}

// Set the sum as payload of the output message
msg.payload = "{\"Capacity\":" + sum + "}";

return msg;

```



Name: Sum

```
// Retrieve park values from flow context and calculate their
let sum = 0;
const flowVariablePrefix = 'park';
const maxParks = 10; // Set the maximum number of parks as ne

for (let i = 1; i <= maxParks; i++) {
    const flowVariableName = flowVariablePrefix + i;
    const parkValue = flow.get(flowVariableName);

    if (parkValue !== undefined) {
        sum += parkValue;
    } else {
        // Exit the loop if the flow variable doesn't exist
        break;
    }
}

// Set the sum as payload of the output message
msg.payload = "{\"Capacity\":" + sum + "}";

return msg;

```

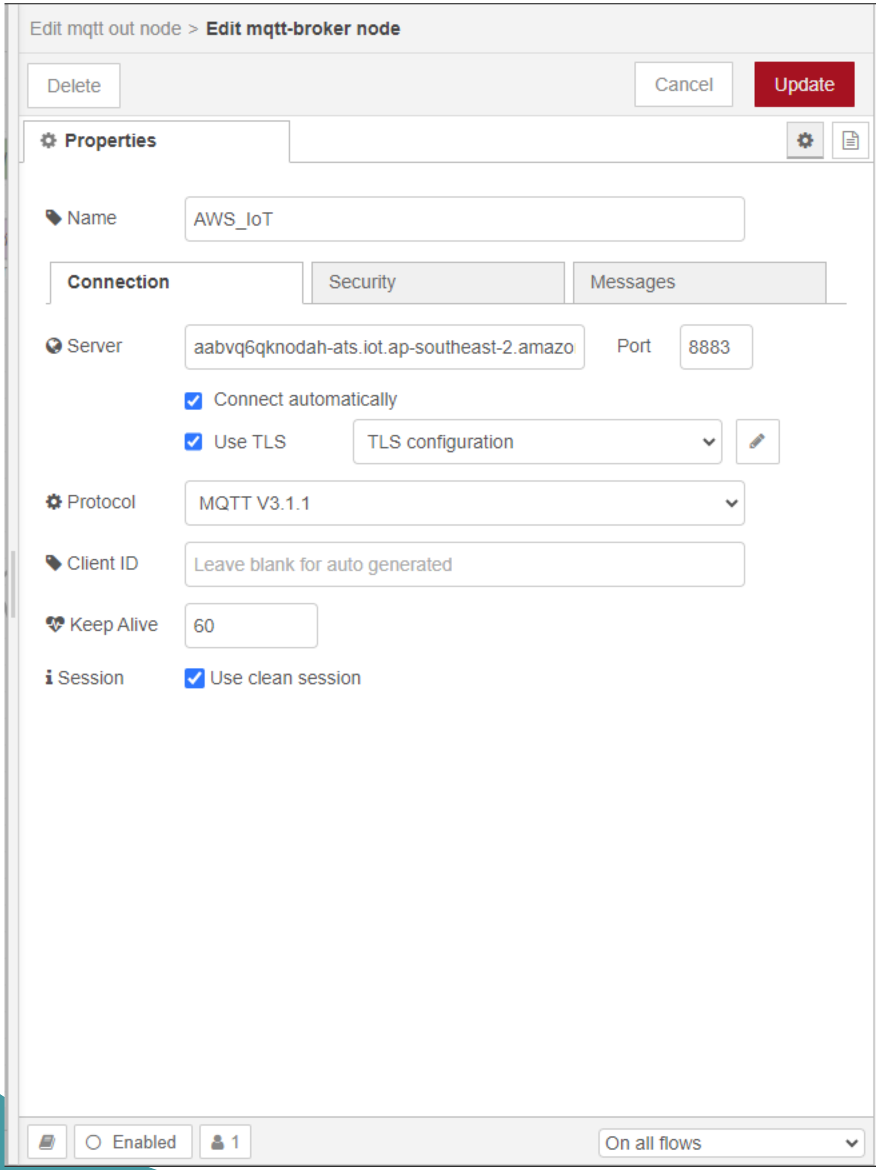


Name: Set Var

```
// Extract spot and status fields from the JSON payload
const spot = parseInt(msg.payload.spot);
const status = msg.payload.status;

// Define the prefix for flow variables
const flowVariablePrefix = 'park';

// Set flow variables based on spot and status
if (!isNaN(spot) && spot > 0) {
    const flowVariableName = flowVariablePrefix + spot;
    const parkStatus = (status === "free") ? 0 : 1;
    flow.set(flowVariableName, parkStatus);
}

return msg;

```

Node-RED

NodeRED



Edit mqtt out node > **Edit mqtt-broker node**

Name: AWS_IoT

Server: aabvq6qknodah-ats.iot.ap-southeast-2.amazo Port: 8883

☑ Connect automatically

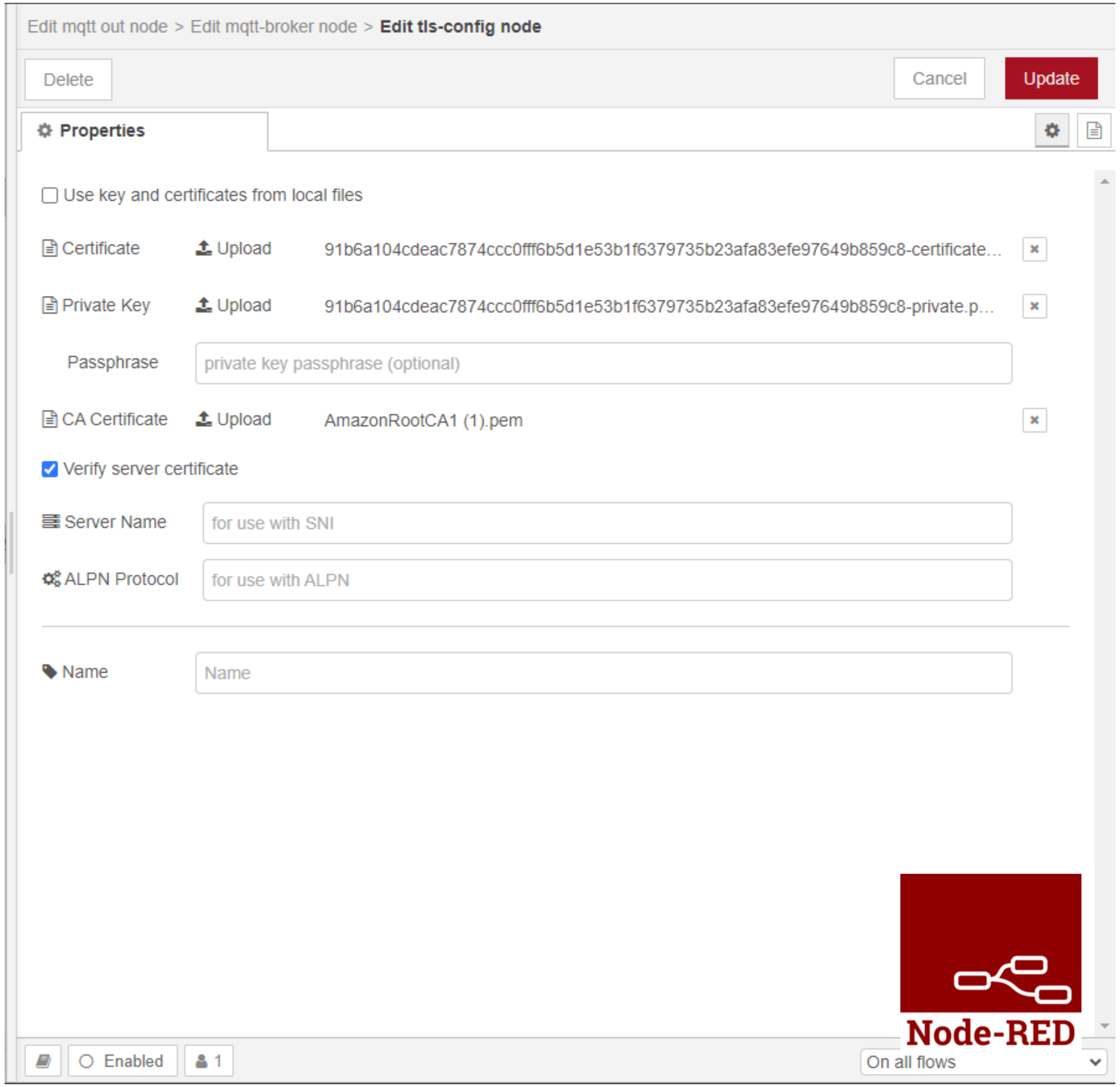
☑ Use TLS: TLS configuration

Protocol: MQTT V3.1.1

Client ID: Leave blank for auto generated

Keep Alive: 60

Session: ☑ Use clean session



Edit mqtt out node > Edit mqtt-broker node > **Edit tls-config node**

☐ Use key and certificates from local files

Certificate: Upload 91b6a104cdeac7874ccc0fff6b5d1e53b1f6379735b23afa83efe97649b859c8-certificate...

Private Key: Upload 91b6a104cdeac7874ccc0fff6b5d1e53b1f6379735b23afa83efe97649b859c8-private.p...

Passphrase: private key passphrase (optional)

CA Certificate: Upload AmazonRootCA1 (1).pem

☑ Verify server certificate

Server Name: for use with SNI

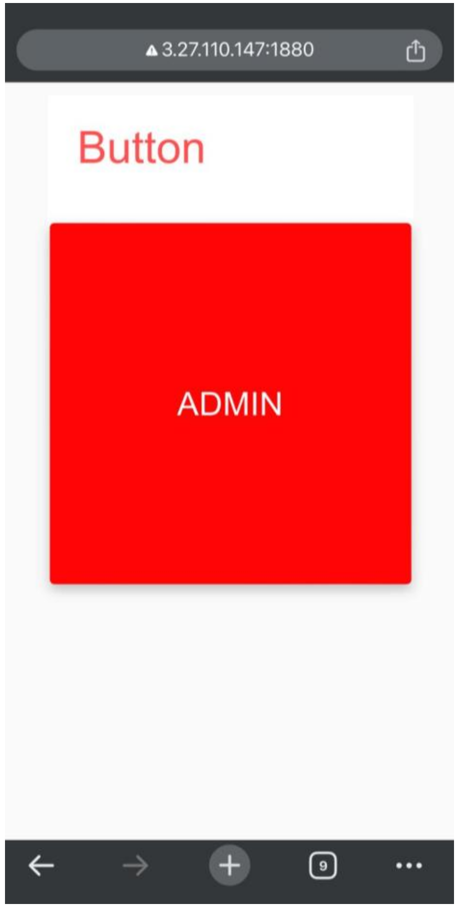
ALPN Protocol: for use with ALPN

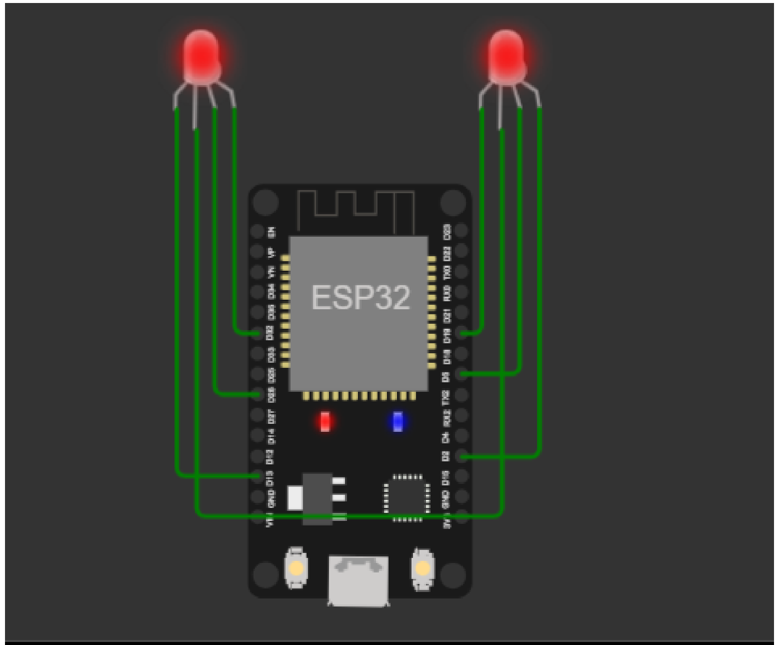
Name: Name

Node-RED

NodeRED

corrupt admin

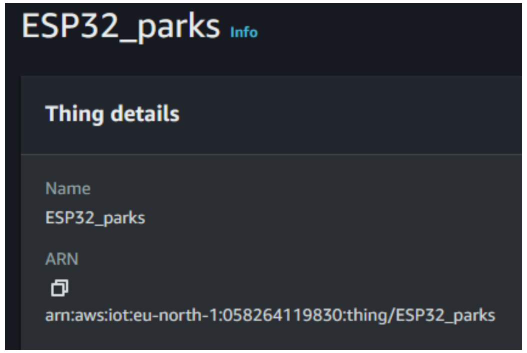




```
40078000,len:12344
il 12 room 4
40080400,len:4124
x40080680
ing to WiFi.................. Connected!
ing to MQTT server... Connected!
 Message received on topic: b'admin/force'
```

Implementation

Thing – An MQTT broker on AWS IoT core



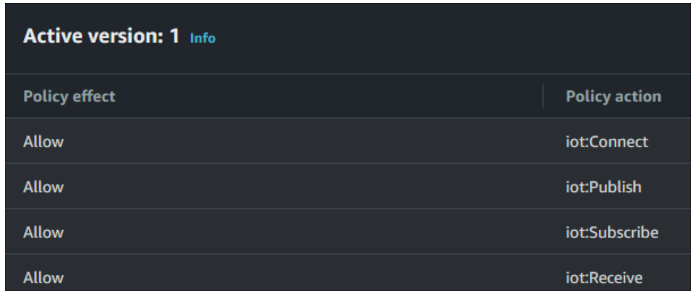
ESP32_parks Info

Thing details

Name
ESP32_parks

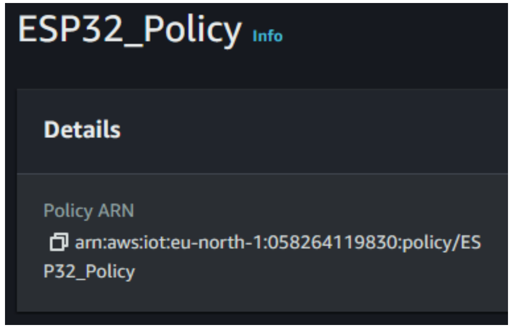
ARN
arn:aws:iot:eu-north-1:058264119830:thing/ESP32_parks

Policy – defined policies for MQTT thing, for connect, publish and subscribe. For this demonstration we allow all who has the certificate and key.



Active version: 1 Info

Policy effect	Policy action
Allow	iot:Connect
Allow	iot:Publish
Allow	iot:Subscribe
Allow	iot:Receive

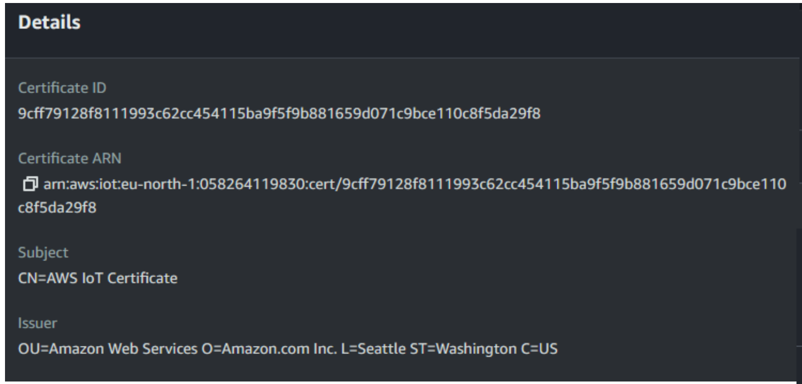


ESP32_Policy Info

Details

Policy ARN
arn:aws:iot:eu-north-1:058264119830:policy/ESP32_Policy

Certificates – We can see the activated certificate attached to the policy and the Thing of our choice.



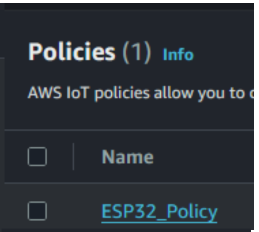
Details

Certificate ID
9cff79128f8111993c62cc454115ba9f5f9b881659d071c9bce110c8f5da29f8

Certificate ARN
arn:aws:iot:eu-north-1:058264119830:cert/9cff79128f8111993c62cc454115ba9f5f9b881659d071c9bce110c8f5da29f8

Subject
CN=AWS IoT Certificate

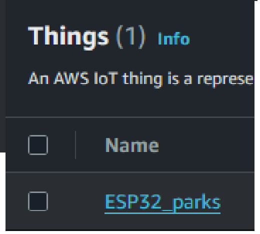
Issuer
OU=Amazon Web Services O=Amazon.com Inc. L=Seattle ST=Washington C=US



Policies (1) Info

AWS IoT policies allow you to c

□	Name
□	ESP32_Policy



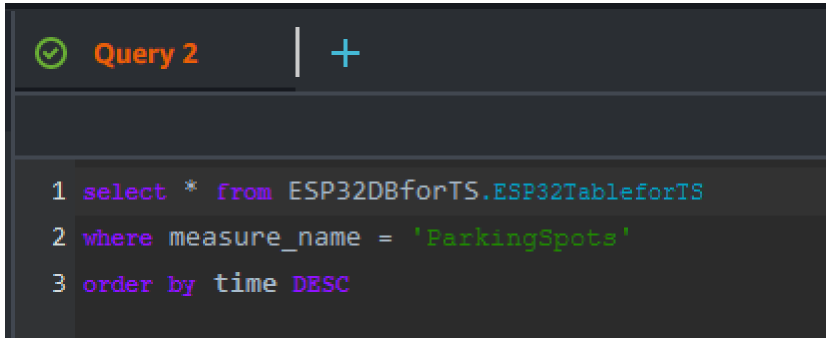
Things (1) Info

An AWS IoT thing is a represe

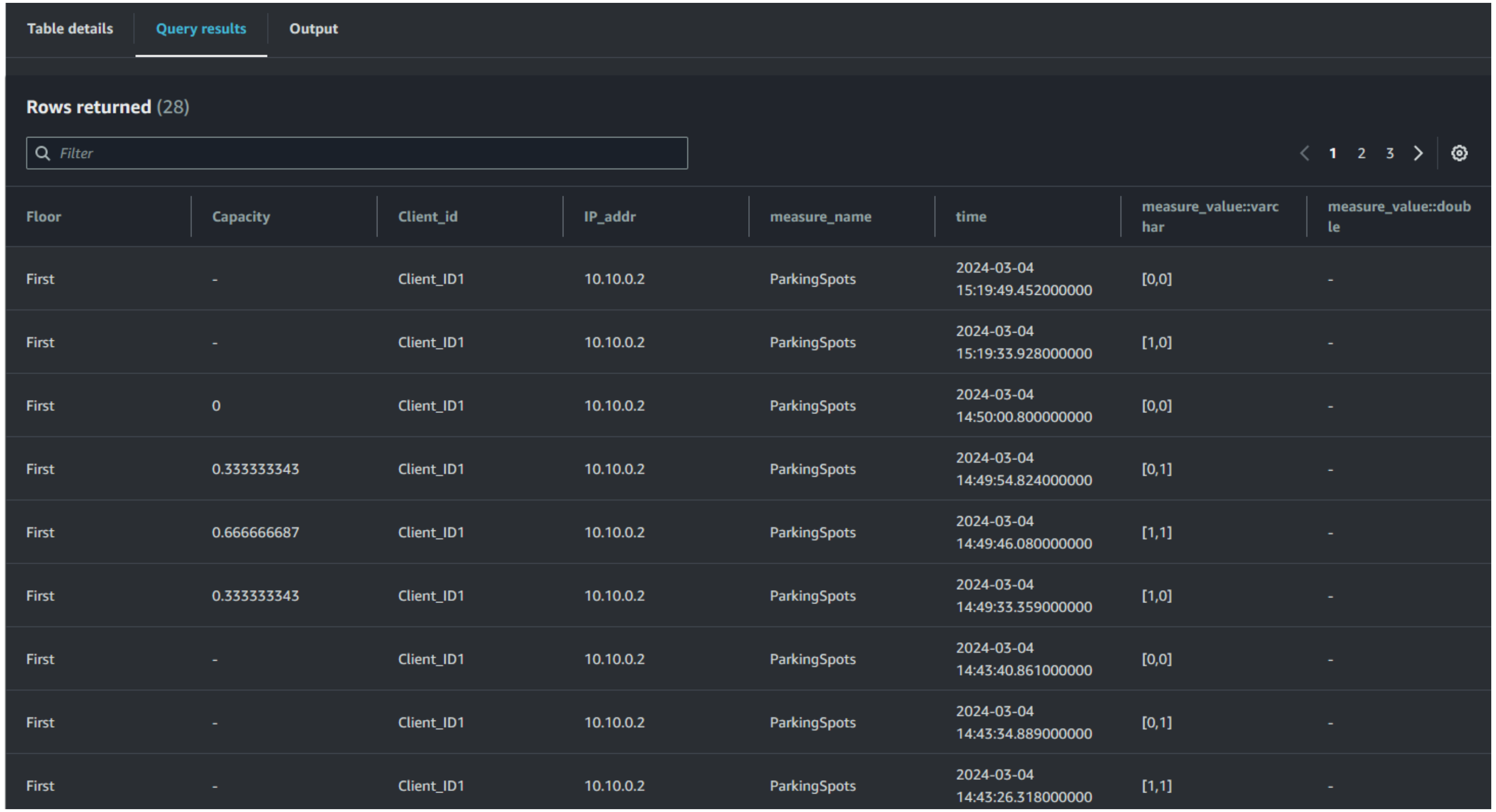
□	Name
□	ESP32_parks

Implementation

Timestream service allow as to take MQTT messages were published to our AWS broker, store in a database with timestamps and query the database table:



```
select * from ESP32DBforTS.ESP32TableforTS
where measure_name = 'ParkingSpots'
order by time DESC
```



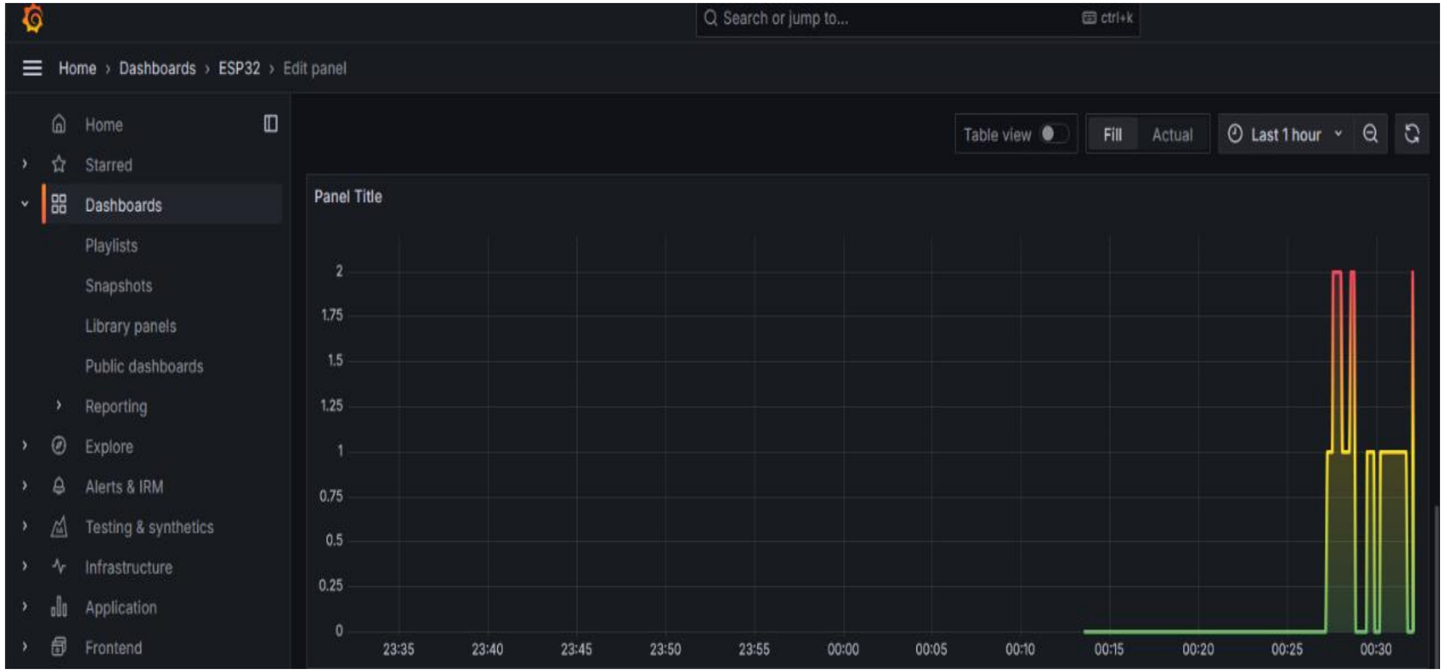
Rows returned (28)

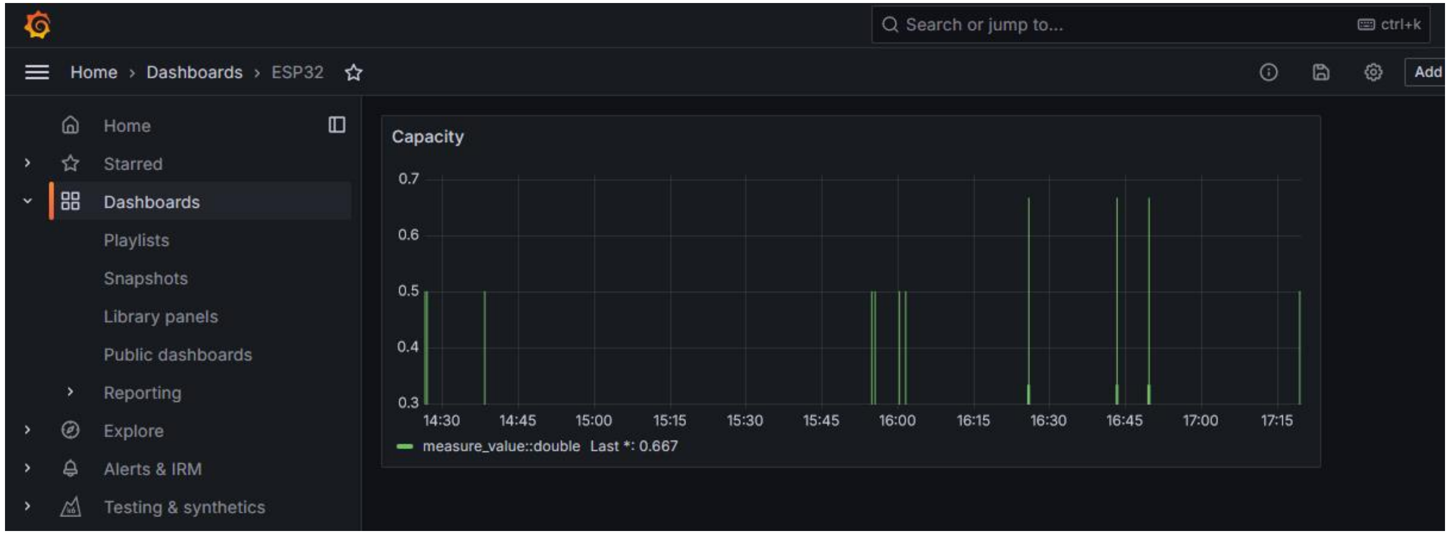
Floor	Capacity	Client_id	IP_addr	measure_name	time	measure_value::varchar	measure_value::double
First	-	Client_ID1	10.10.0.2	ParkingSpots	2024-03-04 15:19:49.452000000	[0,0]	-
First	-	Client_ID1	10.10.0.2	ParkingSpots	2024-03-04 15:19:33.928000000	[1,0]	-
First	0	Client_ID1	10.10.0.2	ParkingSpots	2024-03-04 14:50:00.800000000	[0,0]	-
First	0.333333343	Client_ID1	10.10.0.2	ParkingSpots	2024-03-04 14:49:54.824000000	[0,1]	-
First	0.666666687	Client_ID1	10.10.0.2	ParkingSpots	2024-03-04 14:49:46.080000000	[1,1]	-
First	0.333333343	Client_ID1	10.10.0.2	ParkingSpots	2024-03-04 14:49:33.359000000	[1,0]	-
First	-	Client_ID1	10.10.0.2	ParkingSpots	2024-03-04 14:43:40.861000000	[0,0]	-
First	-	Client_ID1	10.10.0.2	ParkingSpots	2024-03-04 14:43:34.889000000	[0,1]	-
First	-	Client_ID1	10.10.0.2	ParkingSpots	2024-03-04 14:43:26.318000000	[1,1]	-

Implementation

Grafana is a great visualization, easy integrated with AWS, secured tool can visualize data from database's table, in form of our choice.

In the snapshots we can see the capacity over time published from ESP32 client to our AWS broker.





Future ideas

Future ideas

IoT-based smart parking system could feet great in a big scale world of our own, and can be easily integrated with more features to solve more problems:

- **Payment methods** – various of payment methods around the world are very acceptable and could be integrated into IoT-based parking lot as well, all using personal smartphone with no extra application needed to be installed.
- **Parks reservation –** can be very useful for not-every day drivers (and for every-day drivers as well).
- **Smart Cities –** smart parking lots are essential for smarts cities and highly populated areas.

Summary

Summary

- World is getting bigger, faster and its need for scalable, smart and affordable solutions is greatly increase. IoT-based smart parking system are low-cost, using off-the-shelf products such as microcontrollers, cheap and reliable sensors, cloud services and environment, and easy to develop programs.
- In the world's data race where data is valuable, yet your personal data is everywhere, its important to implement systems that protect the customer's personal data with reliable security protocols, without damage it's user experience.
- World's population enlarges every second, and solutions as exhibit must be implemented ASAP.

ANY QUESTIONS?