AWS CLOUD BASED ARCHITECTURAL IMPLEMENTATION DOCUMENT FOR SMART CITY PARKING IOT USECASE

Author: Akram Sheriff

BUSINESS PROBLEM STATEMENT - IOT SMART CITY PARKING PROBLEMS IDENTIFIED:

REQUIREMENTS:

1) Peak hours parking Spot <x,y> Identification issue
2) Locating the Parking spots in roundabout within a small Time (latency) is an **issue (ML model latency issue, UX issue on Mobile App) & Network issue as well – Less time to Locate**
3) Cascaded effect of more turns by different cars in the Roundabout queue of a smart city is contributing to that additional turnaround time to identify the correct parking spot.
4) HTTPS POST of the telemetry data from the Smart Phone mobile app of user about current location of a Car to identify the most optimum location to park a car via the serving AI/ML model is taking longer time. **(ML Model serving latency problem, ML Model output location prediction accuracy problem <x,y>).**
5) As per Smart city Business requirement the SLA for response with optimum location for parking spot should be <=3 seconds.

PHASE – 1: COLLECTION OF LORA SENSOR DATA VIA REAL TIME STREAMING METHOD INTO CLOUD DATAWAREHOUSE (AWS S3 BUCKET) FROM CISCO SMART CITY DEPLOYMENTS .

Duration: 2-3 Months (inclusive of Cisco LoRa gateway deployments) .

Technology Stack: Cisco IOT LoRaWAN gateway, Cisco IOS, Abeeway LoRa parking sensors, Apache flink, AWS Lambda, AWS IOT Core, AWS DynamoDB et al.

https://www.cisco.com/c/en/us/products/routers/wireless-gateway-lorawan/index.html

https://www.abeeway.com/abeeway-geolocation-module/

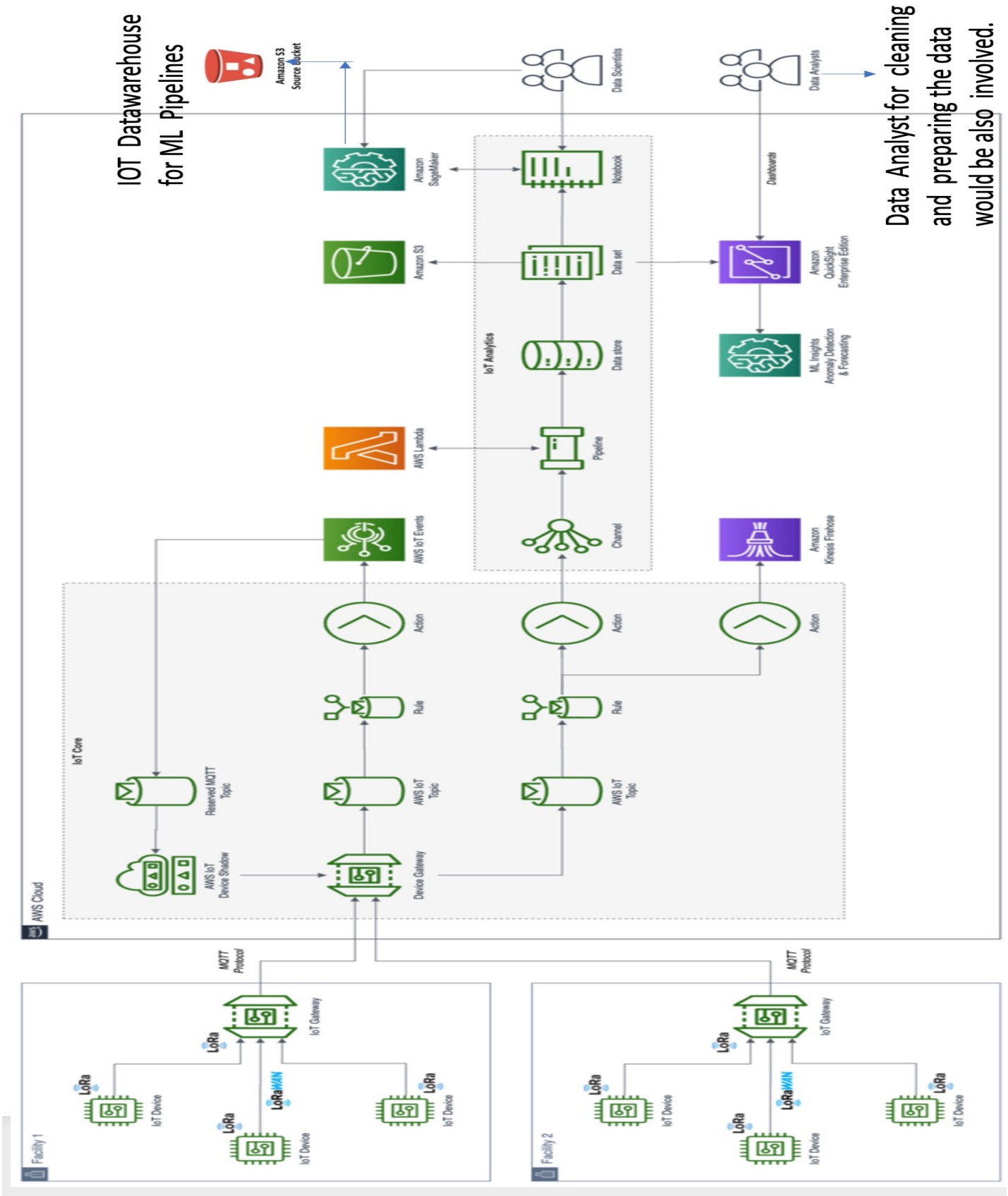
PHASE – 2: AWS Sagemaker based AI ML Data Pre-processing, cleaning, EDA pipeline, Feature extraction pipeline, ML model training/build pipeline, ML deployment pipeline and ML Model serving pipeline (PROD)**.**

Duration: 5-7 months for moving the final ML model into Production/serving as a HTTP End point

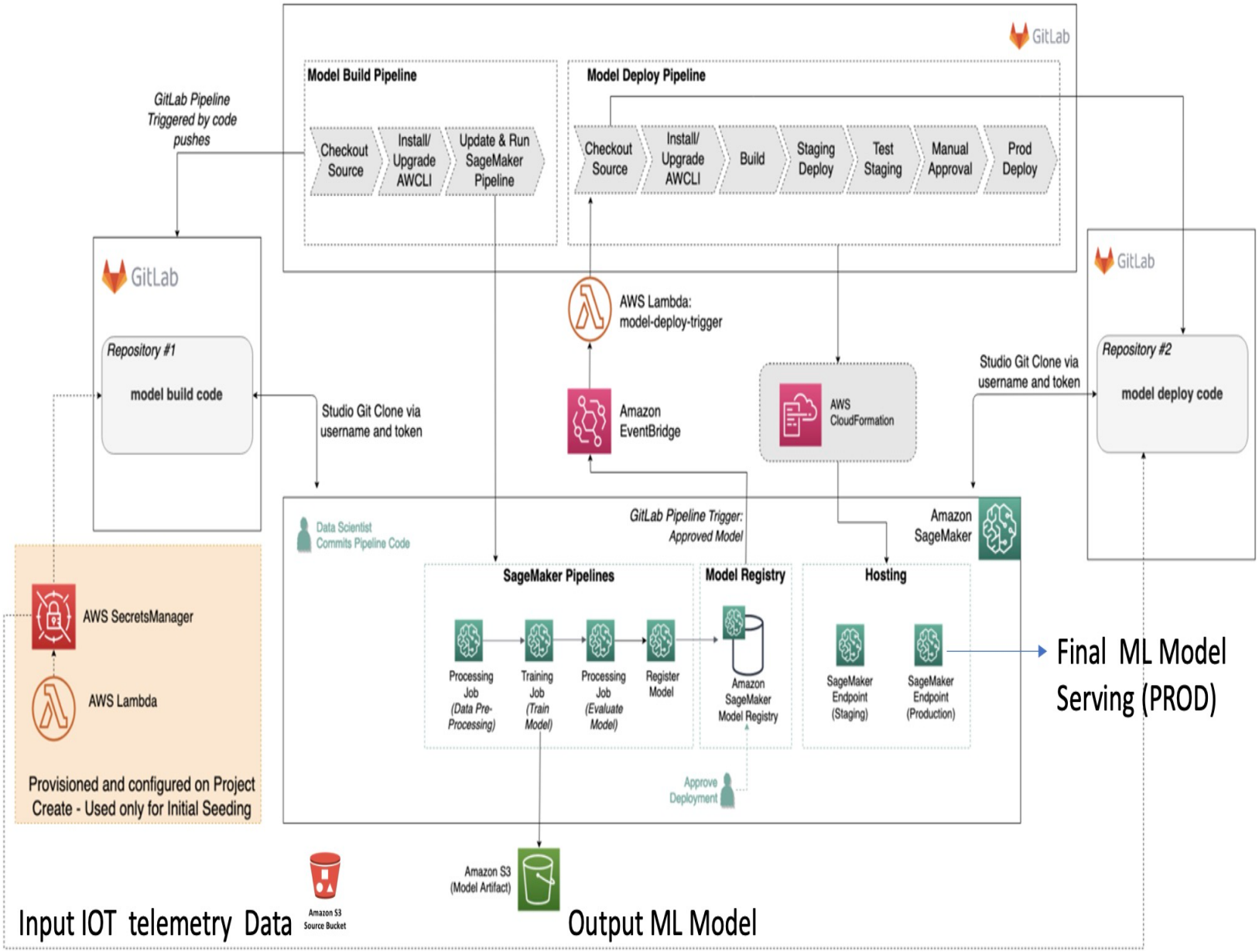
(Technology Stack: Python, AWS Lambda/Serverless, AWS Sagemaker Studio (with SageMaker projects enabled) Cloudformation template, Sagemaker pipelines (CI/CD) Github actions / Gitlab pipelines, AWS Cloudwatch, IAM, DynamoDB et al.

https://github.com/akramIOT/Smart_City_Parking_IOT/

PHASE 1: REAL TIME IOT DATA STREAMING ARCHITECTURE (CISCO LoRaWAN GATEWAY AND LORA SENSOR)



CLOUD ARCHITETCURE OF ML MODEL TRAINING AND DEPLOYMENT PIPELINE WITH AWS SAGEMAKER PIPELINES



IOT SMART CITY ML MODEL PERFORMANCE BENCHMARKING FRAMEWORK ANALYSIS:

- **PROGRAMMABILITY OF ML Model :** For this Smart city IOT parking usecase there is no need for DL or CNN based ML models and hence there is no GPU hardware required for training the model with the available datasets. A simple polynomial regression is used to estimate the optimum parking location (x,y) by comparing the incoming data location attributes from Smart phone and the parking sensor availability status.
- **LATENCY:** The latency of the ML Model's response is very important and critical for this Smart city parking IOT usecase as it will affect the end user experience (On Smart city mobile app) and an increase in the latency to process multiple incoming requests would cause to increase in Car traffic which is circling in the parking roundabouts leading to traffic snarls. Latency is a strict business requirement with an SLA driven approach for this usecase.
- **ACCURACY:** ML Model accuracy for estimating the optimum parking location <X,Y> is very critical for this usecase as an increase in the deviation of estimated location would cause an increase in the number of turns/rounds the car owners would have to take for parking their Car's in the parking lot roundabout. ML model accuracy would affect the UX (User experience) in this usecase as per business requirement.
- **SIZE OF ML MODEL:** Since the trained ML Model for this usecase is NOT deployed in the Cisco LoRaWAN IOT gateway at the edge of the network where the CPU, Memory/RAM resources are limited as it's an embedded gateway product, there is NO hard requirement on the size of the "final trained ML model" moving to Production. The final trained ML model is saved in **pickled** format and then we can load the Sklearn model into memory using **pickle API** and register the loaded model (MLFlow registry) or with the **AWS Sagemaker ML Model Registry (AWS Sagemaker** used this approach in this project solution architecture implementation). Since the ML model is exposed as a HTTP End point (REST Call) with a wrapped function on the trained ML model , the size of ml model is not that critical parameter.
- **THROUGHPUT:** In general throughput describes the #number of inferences that can be delivered given the size of the ML model and input dataset required. In this Smart city IOT parking business usecase scenario the throughput is a very important measure of performance of the ML model. During normal traffic times as per testing the Trained ML model is able to handle about ~ 290-300 requests per minute to respond back with good **a) Parking location accuracy and b) within acceptable latency as per Business requirement (<= 3 Seconds).**Sometimes during peak volume of incoming traffic (HTTP/HTTPS requests) to the ML Model served as an API endpoint , there is a **Tradeoff** required between **Latency, Throughput and location accuracy of the Model.** As per business requirement and discussion with Smart city customer stakeholder, during peak volume of incoming traffic, ML model's performance in terms of Accuracy and latency is given importance with a tradeoff on the **reduction in Max throughput** which can be handled dropping to ~ 230-250 requests per minute.
- **ML MODEL EXPLAINABILITY:** All the data, metadata, extracted features and the ML model is versioned in the repository (using dvc and github based repo) for tracking the "Model lineage" over the complete time period (Time travel of the ML model). Any bias or issues if observed during Model serving period then we can use this data checkpoint to validate the "ML Model explainability" in a end to end manner. So far no issues are observed in the final trained ML model which is moved into production (PROD).
- **ML MODEL ENERGY EFFICIENCY:** There is NO business requirement on the energy consumed (#Inferences per watt metric) for doing inferencing of the incoming real world data hitting the API gateway (HTTP API end point).
- **RATE OF LEARNING OF THE ML MODEL:**As per the observation in the last 4 months of the trained ML Model in production(PROD), there is no model drift observed in production/PROD in terms of estimated location accuracy. Proposed plan is to implement a **ML model drift based ML model re-training** pipeline with AWS sagemaker with new datasets.(real world datasets hitting the API end point from different smart city deployments).
- **Rate of learning** in general is measured in the following ways:

  For training phase: Improvements in throughput and model accuracy.

For production: phase Improvements in throughput, model accuracy, and latency

For both training and production: Improvements in programmability, size of model, and energy efficiency.

- **SECURITY OF ML MODEL:** Since this ML model is deployed in Smart city environment with the PROD ML Model served via the HTTP API end point it's very important to detect and mitigate the following security attacks a) Data poisoning security threat attacks which may happen easily or b) D-DOS attacks. Since in this project the ML model was originally "Offline Trained" model there is a proposed plan to do "Online training" with new real world data sets with AWS Sagemaker pipelines to do CI/CD with the Trigger for model re-training done by measuring the drift in the ML model. Any security threats detected would also account to a performance drift in the ML model which would used as a trigger to re-train with ML model with fresh set of real world data (features) and then deploy it. Hence MLOps and CI/CD approach is used to address the security threat of the ML model.

CLOUD SOLUTION - HIGH LEVEL DESIGN CRITERIA:

• Simplicity & Agility

1) Simple and easy to access Web application or Mobile application for End users to look into the parking spot information by leveraging AWS Cognito.
2) Easier and REST API based 3rd part application development at a quick pace.
3) By leveraging AWS IaaS services like EC2, Intuitive web-applications can be built for exposing this parking sensor data.

• Scale

1) To Seamlessly expand infrastructure regionally or globally to meet the Smart city operation requirements in an automated mode, proposal is to use AWS Cloudformation.
2) By using AWS CloudFormation tool and (.py to Cloudformation JSON Conversion code with Troposphere Library).
3) we could easily replicate this designed Model in another Green field Smart City Parking deployment in a New AWS region.
4) Scale of the # of Messages sent by the LoRa or LTE-M or WIFI Parking sensor in the Uplink direction and the Density of sensors deployed in a Particular region. **• *Cost***

1) Abeeway or Libelium Vendor's LoRa based parking sensor is low cost $1 each for > 10,000 Parking sensors bought. Reference : https://www.alibaba.com/product-detail/Powerfull-more-than-10km-Long-range_60724613680.html?spm=a2700.7724857.normalList.15.1296abd7zMNd7s
2) Engineering e-BOM cost is reduced with the Cloudformation based approach as the same JSON or .yml model can be used on another AWS region for doing Greenfield Smart city deployment.

• Security

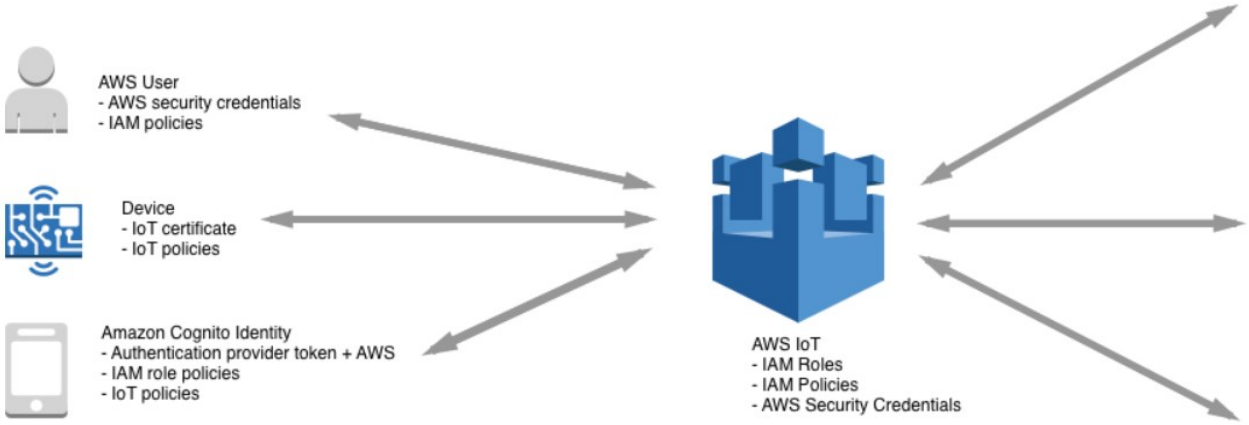
1) Secure communication from IOT End Parking Sensor/Device through cloud while maintaining compliance and iterating rapidly for developing new applications & Dashboards.
2) IAM, Cognito , Cloudwatch based security End to End security.
3) Sensor Network Security & End User (Web or Mobile) application security are considered in this Design. As per the reference architecture below the solution architecture proposal is to

use AWS IOT Device Defender for monitoring the Config's of Device gateway. Since AWS IoT Device Defender makes it easy to maintain and enforce IoT configurations, such as ensuring device identity, authenticating and authorizing devices, and encrypting device data to Cloud. AWS IoT Device Defender continuously audits the IoT configurations on the Parking sensor/ devices against a set of predefined security best practices/rules . AWS IoT Device Defender sends an alert if there are any gaps in the IoT configuration that might create a security risk such as client identity certificates if wrongly shared by the Smart city system Integrator for usage across Multiple different Parking Sensors.

4) **MQTT (over TLS 1.2) with X.509 certificate-based mutual authentication**.

THREE TIER SECURITY REFERENCE ARCHITECTURE FOR SMART CITY IOT PARKING SOLUTION

1) Device Security, 2) End User authorization & Security via Cognito, 3) Device gateway security by mapping proper AWS Roles & Policies.



IOT NETWORK ARCHITECTURE CRITERIA: (Star or Hub & Spoke Topology)

1) Lower Power consumption for Sensor& Lower Duty Cycle for sending the Data Payload.
2) Increased scale of Parking sensors sending the data to an IOT Gateway in Star or a Mesh Topology (Either LoRa Sensor based or WIFI-Mesh based).
3) Ease of use to Track the Sensor Location on a UI Map - For better End User experience.
4) Sensor <Lat, Long> Coordinates based Trilateration Geofence Application can be easily built. (This will minimize the Number of turn cycles for identifying the Parking spot in the Roundabout). This web application to be exposed to end users Mobile Clients & Web App based.

LOGIN CREDENTIALS:

User name,Password,Access key ID,Secret access key,Console login link

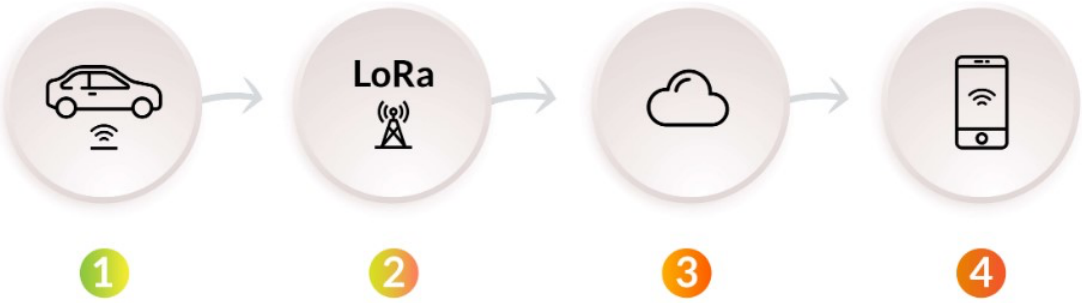
Smart_Parking_MQTT, AKIA45RT6TUKLPZKOU6P,DhXmsk3nOXs0iyTZoByhVyaJauPLsXKK/4jSmo9m,

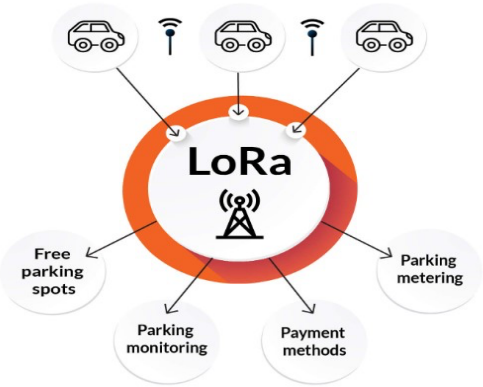
https://888093449492.signin.aws.amazon.com/console

Usr_Name: Smart_Parking_MQTT

Access key ID: AKIA45RT6TUKLPZKOU6P
Secret access key: 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
Consolepwd: iot123

SMART PARKING IOT NETWORK ARCHITECTURE (Sensor+ IOT Gateway + Cloud + Application)





Action #1. In order to detect the absence, arrival, presence, and departure of a vehicle a batterypowered occupancy sensor is installed in each parking space. Their low-power design allows the units to run up to 5 years and require no external wiring to install.

Action #2. When a vehicle activity is detected an occupancy sensor with an embedded LoRa transceiver sends a short data packet containing the change in status to any wireless Internet gateway within its range.

Action #3. For now, users can securely use your smartphone for smart parking payment or finding a free parking spot.

Action #4. The gateways forward the packets to a Cloud-based or dedicated server. By the way, don't hesitate to consider AWS as a service provider for your cloud computing. It provisions organizations with computing power, databases, storage, and other resources in minutes and provides the flexibility to choose the development platform or programming model that will benefit each particular project.

Action #5. Over some period, data are sent to the city planning officials and parking management institutions

NETWORK CONNECTIVITY BANDWIDTH ANALYSIS:

Uplink stream of data is designed in such a way that lowest possible overhead is there in the implemented IOT Architecture.

15 Byte of data Payload in Upstream Direction by each Parking Sensor for every 90 Seconds once. (Based on the JSON Telemetry data stream provided)

1 Bit – For Parking occupancy state(0-Not Present or 1- Present) 2

Bits – For Sensor Numerical Number as per given JSON data.

4 Bytes Float or 4 Bytes Integer – (Lat, Long coordinates of Parking Sensor)

8 Bytes – For DeviceEUID (LoRa)

Number of Bytes sent in LoRa Spreading Factor (SF) for every 1 hour = 15* 40 = 600 Bytes.

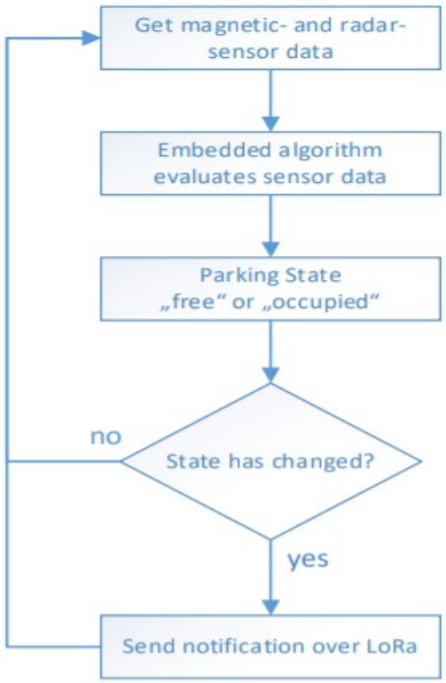
Let's assume **20 Parking Sensors** are present in the Roundabout then , Total message Bytes = 600 * 20 = 12000 Bytes = **1.2 KB Per Hour.**

- **Total Number of Bytes per day = 1.2 KB * 24 = 28.8 KB**
- **This message size data is used to decide the AWS Kinesis shards to be used for Data ingestion from AWS IOT Core.**
- **Based on Device End Point state, since Device Shadow will be implemented in AWS and since filtering of data will be done at LoRa IOT Gateway, the total # of KB sent to the AWS Cloud per day will be around = 20 KB.**

LoRa Class A Device to be used for sending Parking sensor data & since the duty cycle of Class A is minimum and since most of the messages are sent in Uplink direction - **0.1% ducy cycle / 10mW from 863 to 870 MHz. (EU & USA region Freq spec)**

LoRa DevEUI can be uniquely mapped to AWS Client ID for sending JSON Data specific to tht Sensor. (By doing some data Transformations).

LORA BASED PARKING SENSOR ALGORITHM STATE MACHINE



End to End LoRa Network Latency (QOS) – Network Connectivity Design consideration

1) This data point is used to decide the **MQTT Broker running on AWS IOT Core's QOS value (either 0 or 1)** for the Retransmissions. Considering the additional delay caused by the LoRa transmission and possibly re-transmissions, transmission time limitation of each device, transmission from the Gateway to the LoRa network and processing by the LoRa network, the complete delay from the new parking status until the state is visible in the LoRa application server may be 40 or more seconds.
2) Hardware/Sensor technology to identify if a parking spot is occupied. - **Magnetic Field Pattern Technology for occupancy detection using EM Field or Light Blocking Technology for Core Sensor Technology.**
3) Connectivity technology to transmit current state of the parking spot. - **LoRa Parking Sensors or NB-IOT or LTE- M Parking Sensors data -Network Connectivity Technology**.
4) A scalable data processing architecture in AWS to maintain parking data. - **(Refer Diagram)** IOT Core + Device gateway + Lambda + Device Shadow +
5) Parking data is made available to 3rd party app developers to build interesting parking applications. - Refer API Section

6) Platform security and data protection. – **(Refer Diagram)**, IAM, Cognito , Cloudwatch based security End to End security.

7) Edge Computing LoRa or WIFI IOT Gateway implementing MQTT Topic aggregator for using IOT Applications (Device Gateway).

1) **Since the End user should be able to access the Parking Data via a REST API from a web service or from a Mobile client , S3 is used to store eventual transformed JSON data.**
2) **AES-256 Encryption is enabled to ensure security of data stored in S3 Objects.**
3) **Cloudwatch alerts are enabled for the data stored in S3.**
4) **Securely coupled access policy for different web services are mapped to the Roles created to ensure that End to End security is ensured and the solution becomes foolproof.**

AWS Implementation Section: (Workflow)

- Setting up different AWS IoT Core rules to ingest incoming JSON Parking sensor data into AWS Lambda service.
- AWS Rules Engine is used in this Implementation for Filtering & Transforming the incoming
- Randomized data with the given input Sample JSON model is constructed and published into the MQTT Publisher . This is implemented in First Lambda function.
- First Lambda function calls one record of such data and stores into DynamoDB with Table Name "Sensor_State"
- Whenever the external REST API (http/https POST) comes into the API-gateway , an API- endpoint deployed in Triggered to invoke the second Lambda function. All the incoming data and headers are allowed through passthrough method.
- Second Lambda function queries the DynamoDB for that Parking Sensor's entry based on Serial Number as Primary Key.
- Implementing the AWS S3 data fetch python job into the data processing pipeline.
- Creating the AWS ML Model build, test , deploy pipeline. (Sagemaker pipelines based)
- Creating AWS Sagemaker ML Model deployment pipeline via AWS event Bridge, gitlab pipelines.

SMART PARKING IOT USECASE:

1) **AWS IOT Rules based:** Based on given parking sensors <LAT, LONG> coordinates. Create a Geofence region for the parking spots along the Roundabout and the moment a car/passenger comes into the Geofence region (based on LAT, LONG) Coordinates push a HTTPS POST Notification via AWS API Gateway + Lambda Trigger to the backend Serverless Lambda applications to process the data.
2) **Depending on the outcome of Location calculation, Occupancy_state of Sensor the state of sensor is flipped in DynamoDB by write method executed by Lambda function.**
3) To begin with along the Smart city downtown roundabout , a Geofence map can be constructed with the geo coordinates as per the location of Parking sensors.

4) If a car comes into the Geofence Map, A https POST is done to mimic the simulation of this usecase and the parking Slot is allotted or Not allotted as per this outcome.

AWS IOT RULE SETUP: (MQTT Pub/Sub model based & Based on given JSON Telemetry data)

Sensor Location Topic Rule:
$aws/<location>/<Sensor_id_number>/state>/<city>/<street_name>/<lat,long>/ **Sensor**

State Topic Rule:

$aws/<Sensor_Presence>/ Sensor_id_number/<Timestamp>/<Occupancy_State>/

Leveraging AWS IOT Core: (Javascript & Python SDK) https://github.com/aws/aws-iot-device-sdk-python

https://github.com/boto/boto3

SOFTWARE CODE/ IMPLEMENTAITON DETAILS:

Creating a AWS CloudFormation Template leveraging Troposphere Library:

(Python code to .JSON Cloudformation conversion)

All the Code is uploaded into github - https://github.com/akramIOT/Smart_City_Parking_IOT/

```
# -*- coding: utf-8 -*-
"""
Created on Tue Feb  4 19:51:44 2020
@author: akram sheriff
"""
from troposphere import Ref, Template
import troposphere.ec2 as ec2 t =
Template() instance =
ec2.Instance("IOT")
#instance.InstanceId = "i-019f43e8695aa6af8"
instance.ImageId = "ami-03caa3f860895f82e"
instance.InstanceType = "t2.micro"
t.add_resource(instance) print(t.to_json())
```

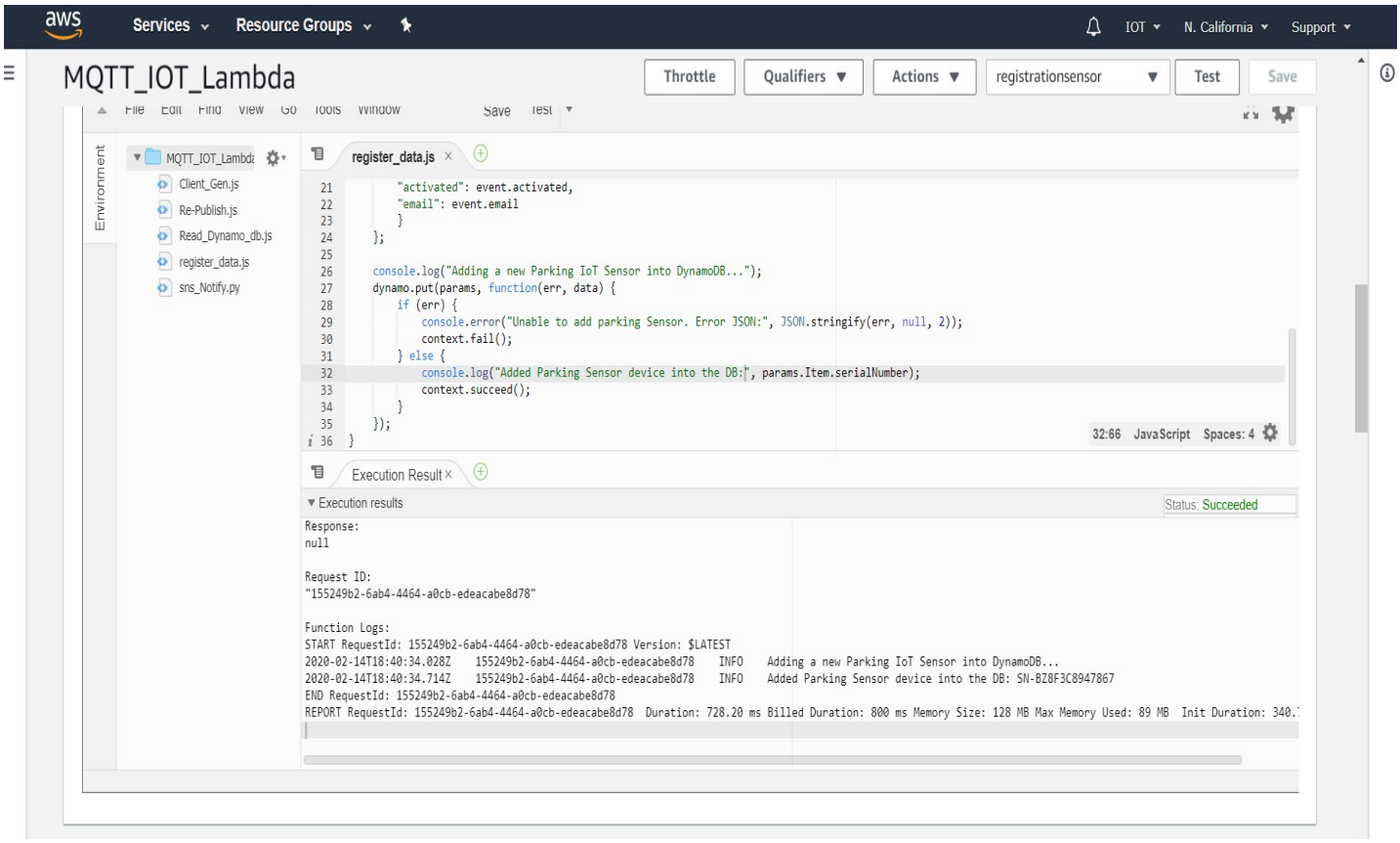
```
  "Resources": {
    "IOT": {
      "Properties": {
        "ImageId": "ami-03caa3f860895f82e",
        "InstanceType": "t2.micro"
      },
      "Type": "AWS::EC2::Instance"
    }
  }
}
```


AWS CLI OUTPUT:


```
[root@akram-lnx-vm python-paho-mqtt-for-aws-iot]#
[root@akram-lnx-vm python-paho-mqtt-for-aws-iot]# aws iot list-things
{
  "things": [
    {
      "thingArn": "arn:aws:iot:us-west-1:888093449492:thing/IOT_thing",
      "version": 1,
      "thingName": "IOT_thing",
      "attributes": {}
    },
    {
      "thingArn": "arn:aws:iot:us-west-1:888093449492:thing/Akram_IOT",
      "version": 2,
      "thingName": "Akram_IOT",
      "attributes": {
        "environment": "awstakehome"
      }
    },
```

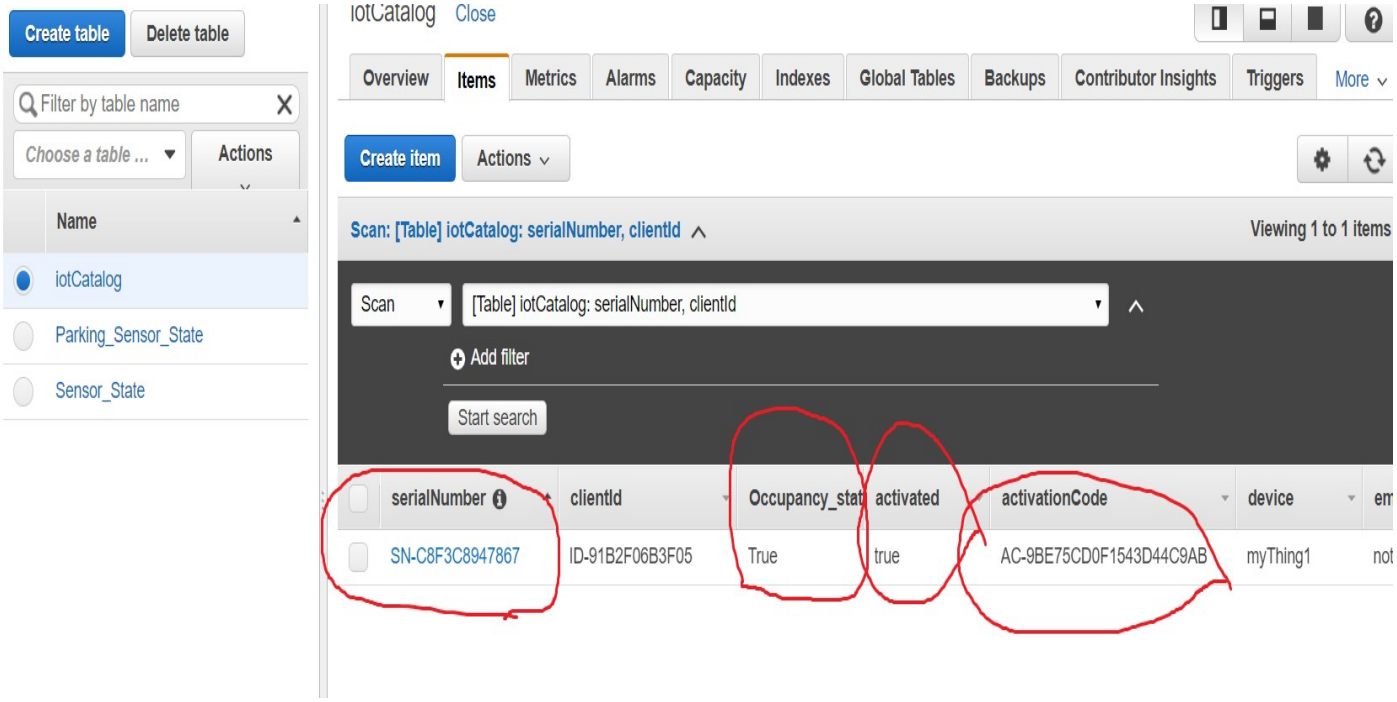
```
    {
      "thingArn": "arn:aws:iot:us-west-1:888093449492:thing/IOT_Endpoint",
      "version": 2,
      "thingName": "IOT_Endpoint",
      "attributes": {
        "environment": "awstakehome"
      }
    }
  ]
}
```


EXECUTION IMPLEMENTATION LOGIC:

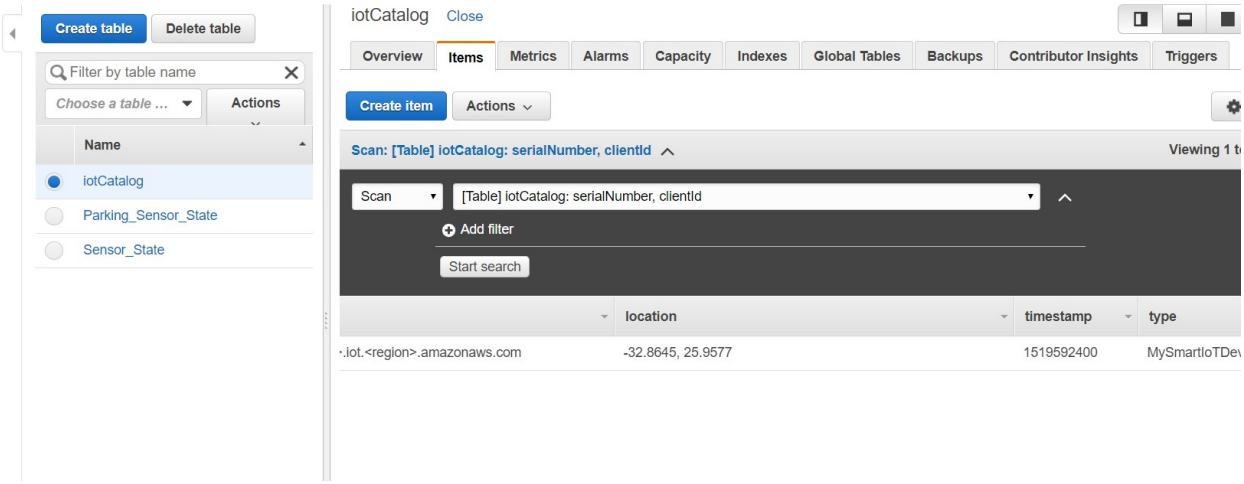


OUTPUT SNAPSHOTS: (Workflow)

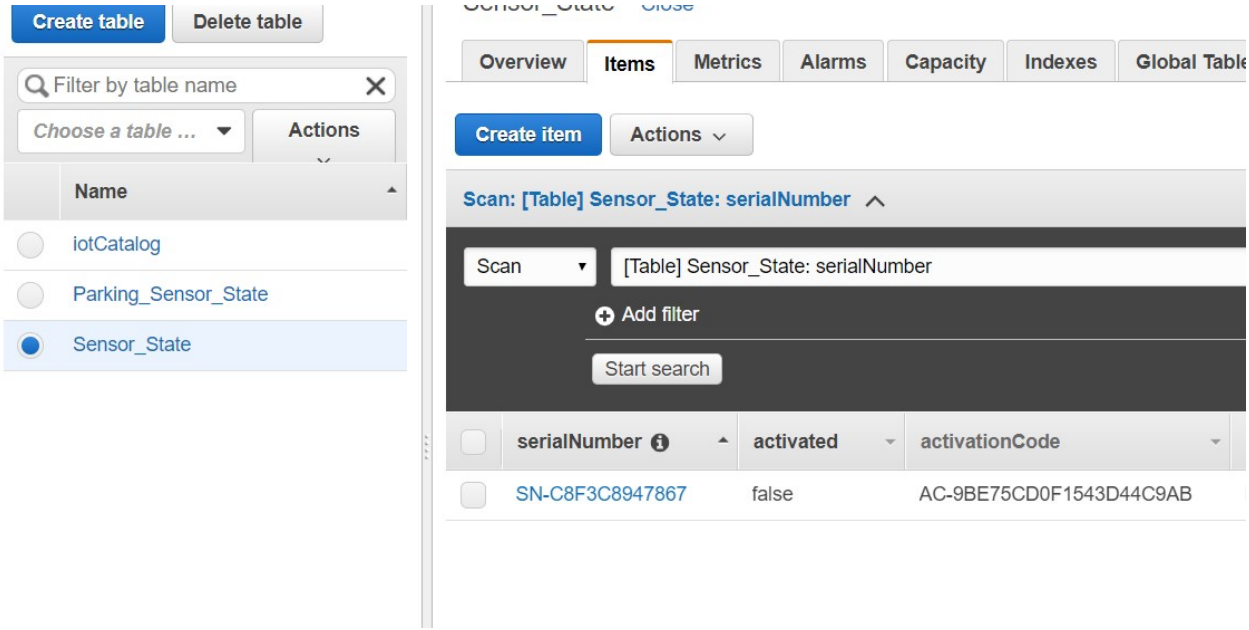
1)Sample Data Record pushed into DynamoDB Table



2) The Parking Sensor's location state is shown below.



3) Currently the parking Sensor Spot is Un-occupied as shown below state.



- Incoming External End Point https POST request/Query uses Car's location to identify the Sensor with less distance as the appropriate one and then Queries into DynamoDB(by using the Sensor's Serial Number as query string) to check/ compare the Occupancy state if its available or NOT.
- If the State is False, then the Parking spot is allocated to that Car. If the state is TRUE which means the Parking Spot is already occupied and so the next parking sensor is iterated.
- All this is implemented in the backend Lambda Logic (2nd Lambda function).
- Correspondingly the Occupancy state of that sensor is flipped in DynamoDB to Occupancy_State- True so that no one else can reserve that spot or False, somone occupied
- ML Model estimated location for parking:

TEST_VALIDATION_RESULTS :