2020.06.15 클라우드IOT

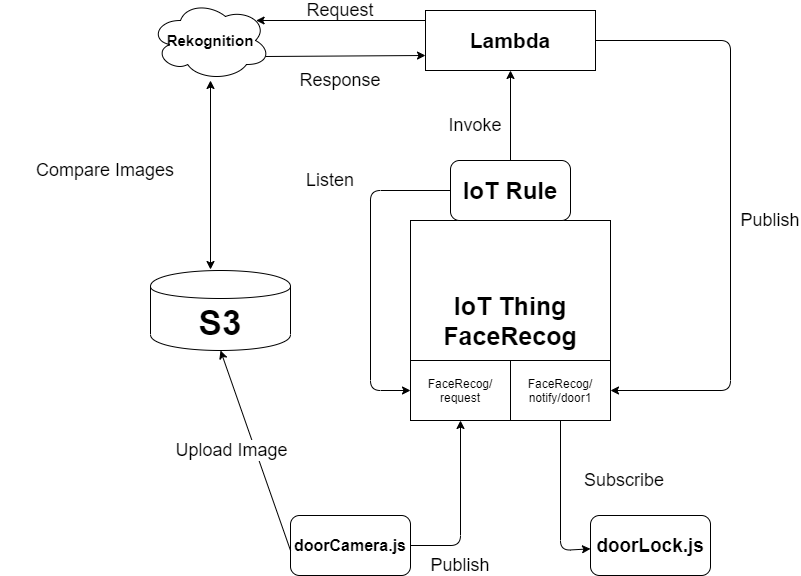
**Homework Assignment #5**

**Smart Door System**



|  |  |
| --- | --- |
| **과목** | **클라우드IOT** |
| **교수명** | **정갑주 교수님** |
| **학과** | **컴퓨터공학과** |
| **학번** | **201411275** |
| **이름** | **박진호** |

1. **System Design**

****

1. **System Implementation**

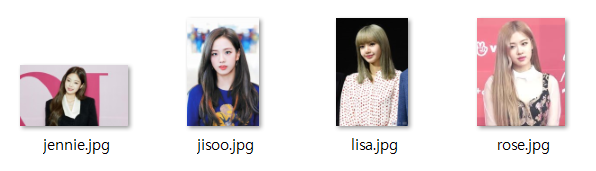
* **Task #1**

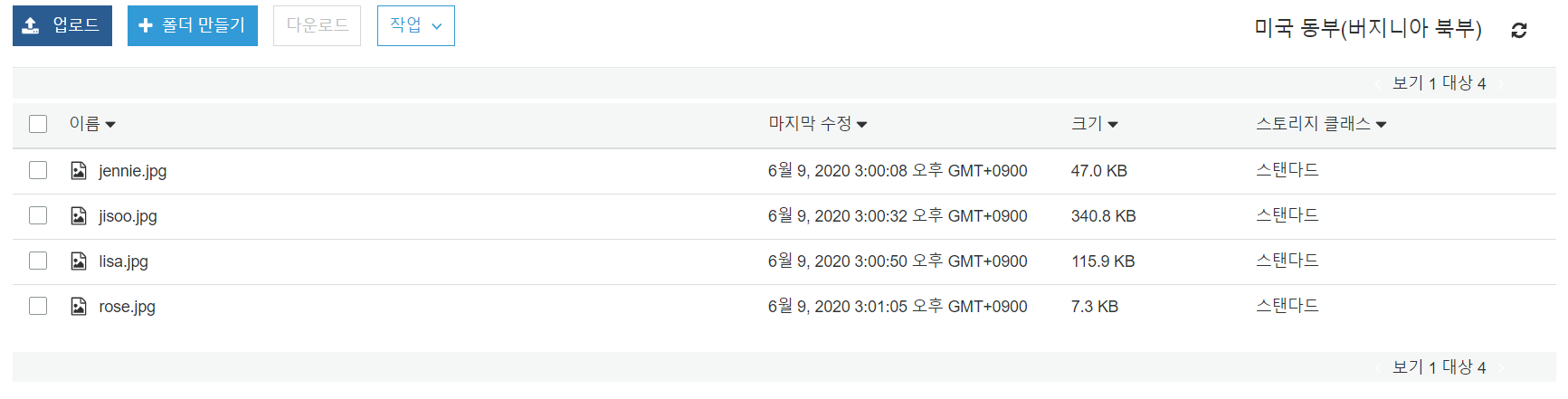
1. Image DB



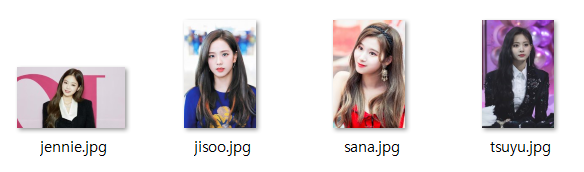
인증된 사용자들의 얼굴 Image를 담는 버킷인 image-db 버킷과 Test App으로 인증을 받을 사용자들의 얼굴 Image를 담는 image-visitor 버킷 두개를 생성한다.

인가된 사용자인 image-db 버킷에는 블랙핑크 4명의 이미지를 각 4개의 객체로 생성하였다.





Test App용 방문자의 Image를 담은 Visitor 버킷에는 위에서 사용한 제니, 지수 사진 두장과 사나, 쯔위의 사진을 업로드하였다. 즉, 반은 인가된 사용자이고, 나머지 반은 비인가된 사용자이다.





1. Face Recognition Lambda Service

얼굴인식 서비스를 개발하기 위해 node.js 프로젝트를 생성한다. Lambda에 올리기 위해 handler를 코딩하고 export한다.

const AWS = require("aws-sdk");

const keys = require("./credential-keys.js");

exports.handler = async (event) => {

  AWS.config.update({ region: "us-east-1" });

  const config = new AWS.Config({

    accessKeyId: keys.credential.accessKeyId,

    secretAccessKey: keys.credential.secretAccessKey,

  });

  const client = new AWS.Rekognition({

    accessKeyId: keys.credential.accessKeyId,

    secretAccessKey: keys.credential.secretAccessKey,

  });

AWS-SDK 의 s3 와 Rekognition을 사용하기 위해 credential을 import 해서 AWS 객체를 생성하고, Rekognition과 Config 객체를 생성해준다.

  const visitor = { bucket: event.bucket, name: event.name };

  const DB = ["jennie.jpg", "jisoo.jpg", "lisa.jpg", "rose.jpg"];

  const FaceCompare = async (index) => {

    const params = {

      SourceImage: {

        S3Object: {

          Bucket: "facerecog-image-db",

          Name: DB[index],

        },

      },

      TargetImage: {

        S3Object: {

          Bucket: visitor.bucket,

          Name: visitor.name,

        },

      },

      SimilarityThreshold: 70,

    };

    const result = await client

      .compareFaces(params, function (err, data) {

        if (err) console.log(err, err.stack);

        else {

          return data.FaceMatches;

        }

      })

      .promise();

    return result;

  };

  const facematch = [];

  for (var i = 0; i < 4; i++) {

    facematch[i] = await FaceCompare(i);

    if (facematch[i].FaceMatches != "") return true;

  }

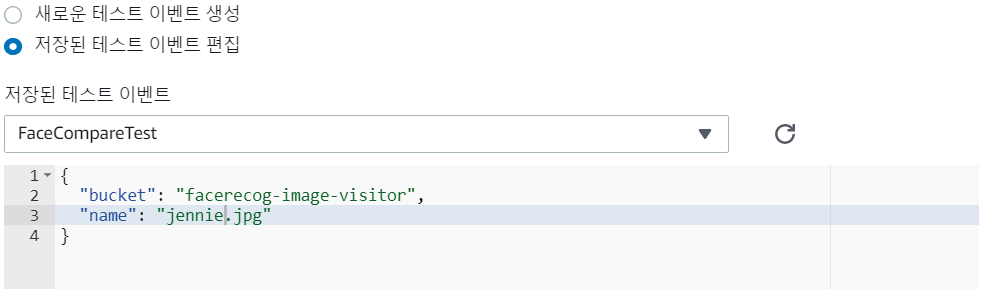
  return false;

};

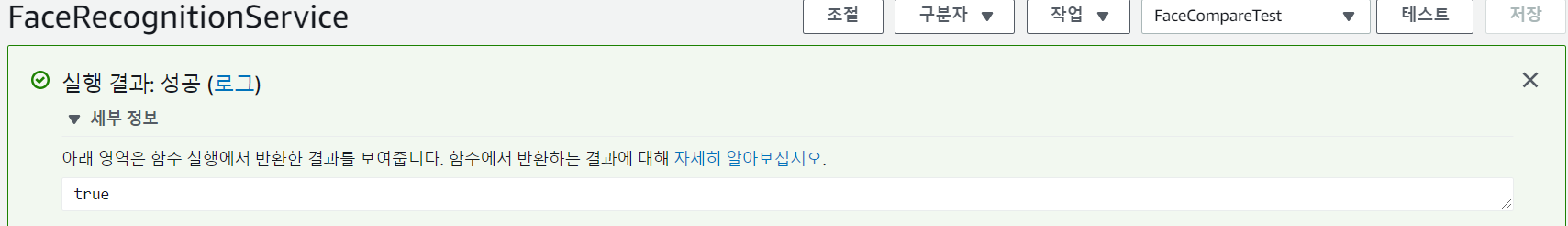
event 인자로 받은 visitor의 bucket이름과 객체이름인 name을 visitor 변수에 저장하고, String 배열인 DB에 image db에 올렸던 객체의 이름들(jennie.jpg, jisoo.jpg, lisa.jpg, rose.jpg) 를 넣어준다. FaceCompare 라는 async function을 생성하고, 인자를 받아서 해당 인자에 해당하는 name을 가진 객체와 visitor의 Image를 parameter로 하여 Rekognition의 함수인 compareFaces로 비교한다. 이후 data.FaceMatches를 promise로 return해준다.

Image DB에는 총 4개의 사진이 들어있으므로 4번을 반복하는 반복문에서 얼굴을 비교한 결과를 받아서 FaceMatches 객체가 공백이 아니면 (만약 일치하는 이미지이면 FaceMatches에 Similarity등의 정보가 담겨있지만, 일치하지 않는 이미지이면 FaceMatches는 공백의 객체이다) true를 return한다. 반복문을 다 돌았는데도 return 하지 않으면 일치하는 이미지가 없는 것이므로 false를 return 하도록 했다.

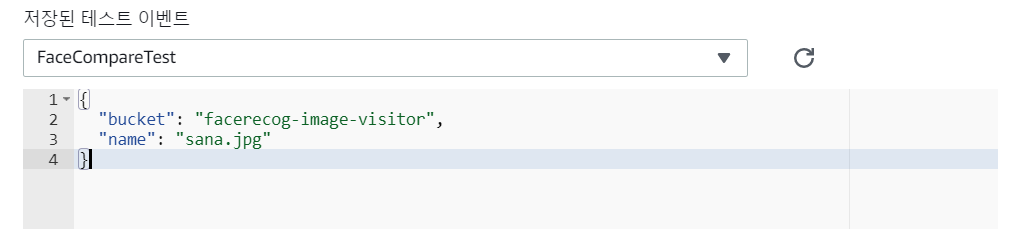
1. Face Recognition Service Test



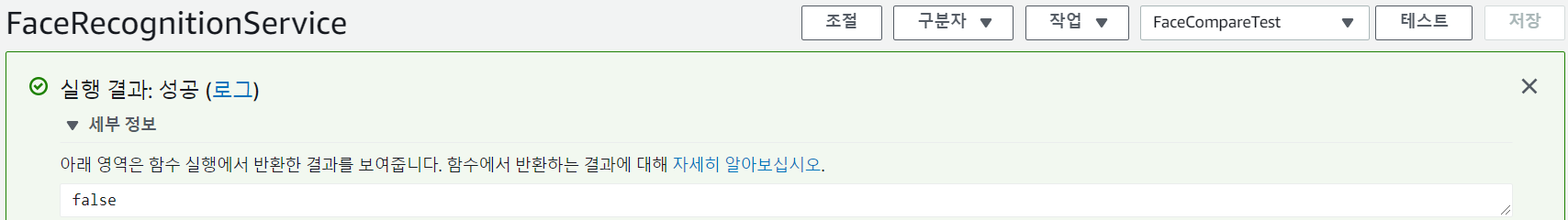
내가 만든 코드를 Lambda에 올리고 Test를 하기위해 event 를 작성한다. 이때 방문자의 bucket인 facerecog-image-visitor와 그 안에있는 성공케이스인 jennie.jpg를 name으로 해서 성공 케이스를 테스트 해보았다.



jennie.jpg는 image db에 등록된 인증된 방문자이므로 true가 출력됨을 확인할수 있다.



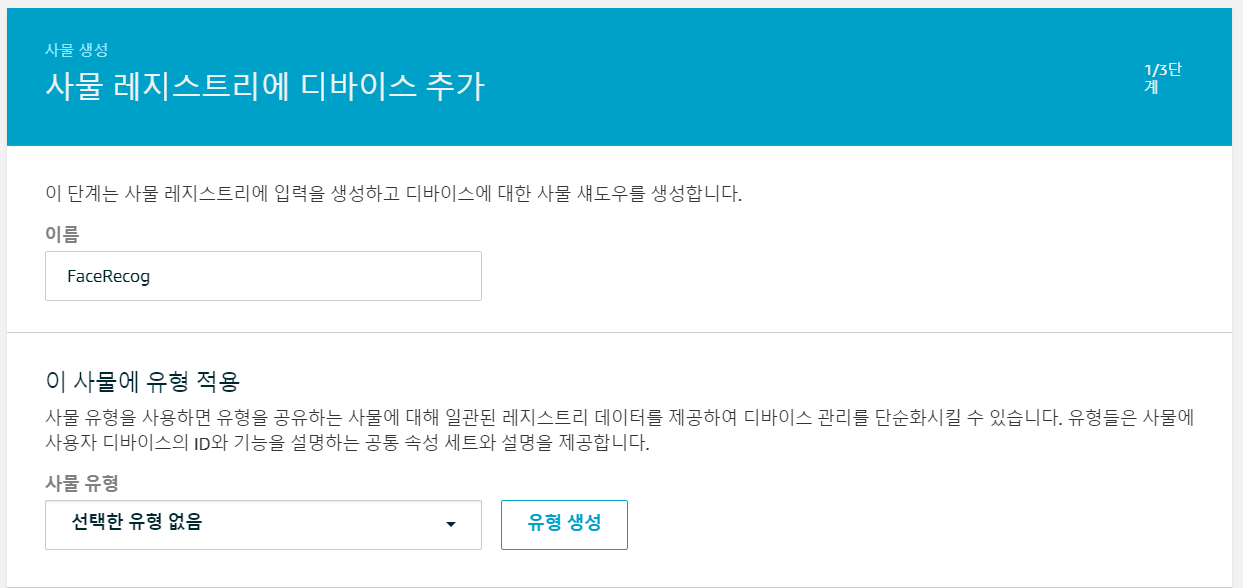
다음은 실패 케이스를 Test하기 위해 Image DB에 없는 sana.jpg를 event에 넣고 Test를 해보았다.

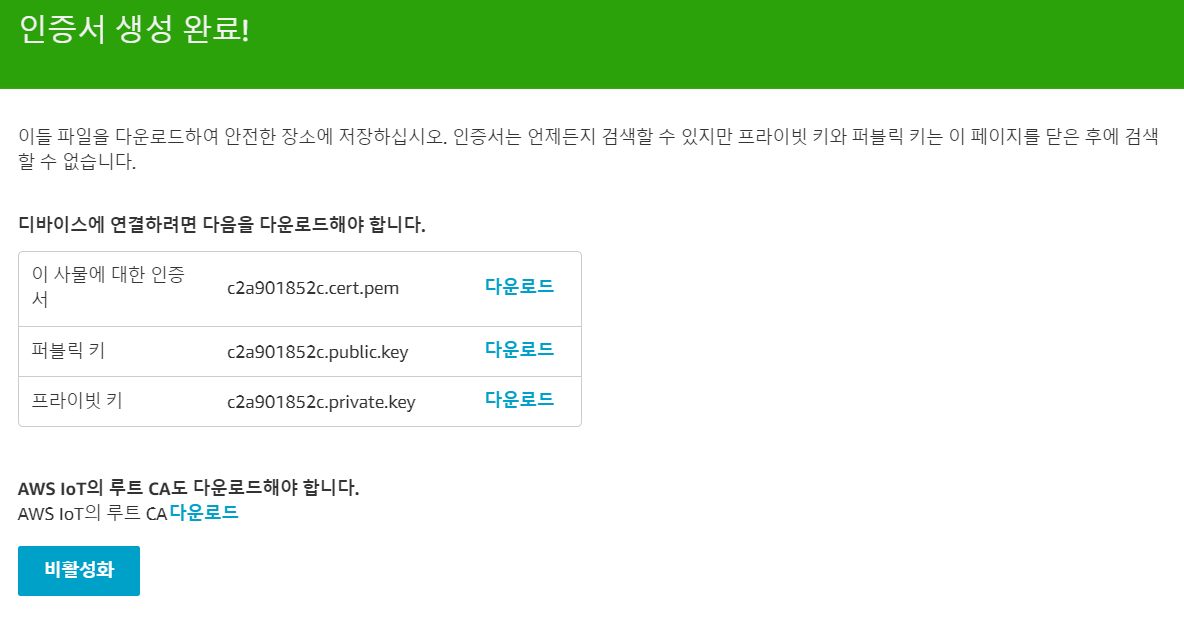


sana.jpg 는 인증되지 않은 방문자이므로 결과가 false로 나오는것을 확인할 수 있다.

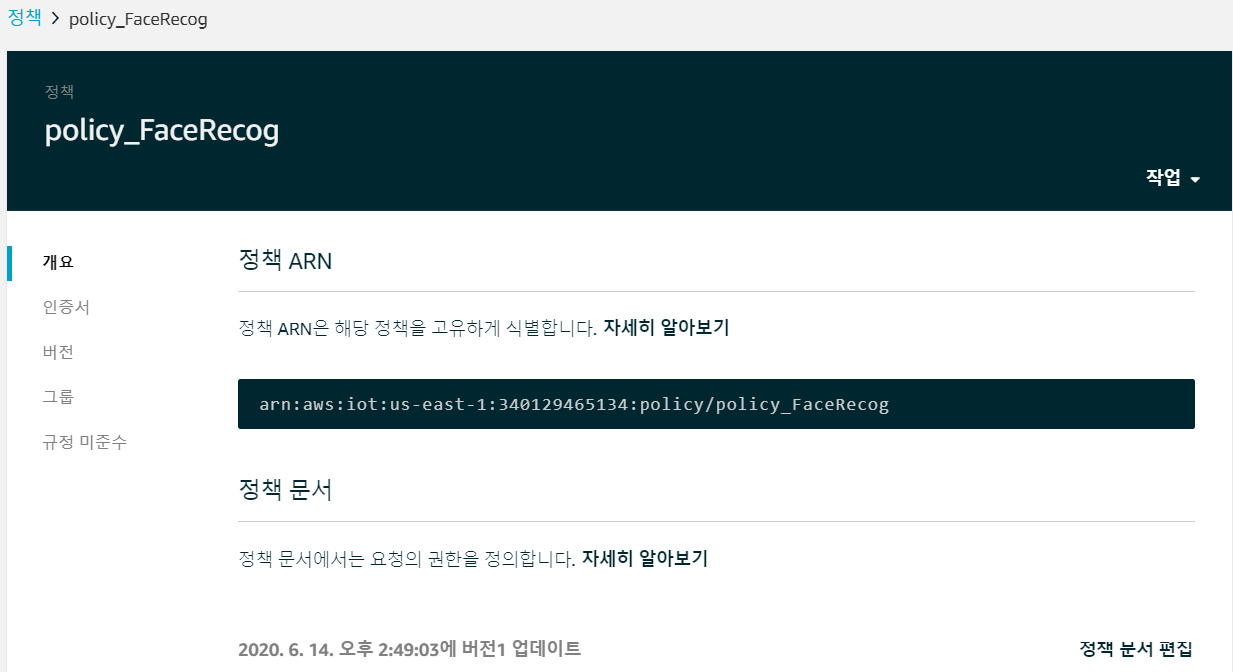
* **Task #2**

1. 사물 등록





FaceRecog 사물을 생성하고 인증서들을 다운로드 받고 활성화한다.



FaceRecog의 정책을 생성한다.

FaceRecog에 필요한 정책은 Connect(doorCamera, doorLock,FaceRecognitionService),

FaceRecog/request 의 Publish(doorCamera.js), Subscribe(IoT Rule), Receive(IoT Rule),

FaceRecog/notify/door1 의 Publish(Lambda), Subscribe(doorLock.js), Receive(doorLock.js)

총 8개가 필요하다.

해당 정책으로 생성된 JSON은 아래와 같다.

{

    "Version": "2012-10-17",

    "Statement": [

      {

        "Effect": "Allow",

        "Action": "iot:Connect",

        "Resource": "arn:aws:iot:us-east-1:340129465134:client/doorCamera"

      },

      {

        "Effect": "Allow",

        "Action": "iot:Connect",

        "Resource": "arn:aws:iot:us-east-1:340129465134:client/doorLock"

      },

      {

        "Effect": "Allow",

        "Action": "iot:Connect",

        "Resource": "arn:aws:iot:us-east-1:340129465134:client/FaceRecognitionService"

      },

      {

        "Effect": "Allow",

        "Action": "iot:Publish",

        "Resource": "arn:aws:iot:us-east-1:340129465134:topic/FaceRecog/request"

      },

      {

        "Effect": "Allow",

        "Action": "iot:Subscribe",

        "Resource": "arn:aws:iot:us-east-1:340129465134:topicfilter/FaceRecog/request"

      },

      {

        "Effect": "Allow",

        "Action": "iot:Receive",

        "Resource": "arn:aws:iot:us-east-1:340129465134:topic/FaceRecog/request"

      },

      {

        "Effect": "Allow",

        "Action": "iot:Publish",

        "Resource": "arn:aws:iot:us-east-1:340129465134:topic/FaceRecog/notify/door1"

      },

      {

        "Effect": "Allow",

        "Action": "iot:Subscribe",

        "Resource": "arn:aws:iot:us-east-1:340129465134:topicfilter/FaceRecog/notify/door1"

      },

      {

        "Effect": "Allow",

        "Action": "iot:Receive",

        "Resource": "arn:aws:iot:us-east-1:340129465134:topic/FaceRecog/notify/door1"

      }

    ]

  }

위와같이 정책을 생성하고 FaceRecog 사물과 연결한다.

1. doorCamera.js

const awsIot = require("aws-iot-device-sdk");

const AWS = require("aws-sdk");

const fs = require("fs");

const keys = require("./credentials/credential-keys.js");

const readline = require("readline");

AWS.config.region = "us-east-1";

AWS.config.apiVersions = {

  s3: "2006-03-01",

};

const s3 = new AWS.S3({

  accessKeyId: keys.credential.accessKeyId,

  secretAccessKey: keys.credential.secretAccessKey,

});

const doorCamera = awsIot.device({

  keyPath: "./credentials/c2a901852c-private.pem.key",

  certPath: "./credentials/c2a901852c-certificate.pem.crt",

  caPath: "./credentials/AmazonRootCA.pem",

  clientId: "doorCamera",

  host: "ai4oovsp79d43-ats.iot.us-east-1.amazonaws.com",

});

function createObject(params) {

  return new Promise(function (resolve, reject) {

    s3.upload(params, function (err, data) {

      if (err) reject(err);

      else resolve(data);

    });

  });

}

const upload = async function (image\_name) {

  try {

    const my\_object = {

      Bucket: "facerecog-image-visitor",

      Key: image\_name,

      Body: fs.createReadStream("./image/" + image\_name),

    };

    const result = await createObject(my\_object);

    console.log(result);

  } catch (err) {

    console.log(err);

  }

};

doorCamera.on("connect", function () {

  console.log("Door Camera connected");

  const rl = readline.createInterface({

    input: process.stdin,

    output: process.stdout,

  });

  rl.on("line", function (line) {

    upload(line);

    console.log("Uploaded image to S3");

    const message = {

      notify: "FaceRecog/notify/door1",

      bucket: "facerecog-image-visitor",

      name: line,

    };

    doorCamera.publish("FaceRecog/request", JSON.stringify(message));

    console.log("Published to FaceRecog/request" + JSON.stringify(message));

  }).on("close", function () {

    process.exit();

  });

});

doorCameara.js 에서는 readline으로 image의 이름을 입력받아서 같은 폴더에 있는 하위 폴더 /image에 있는 image를 s3 객체로 업로드 하고 FaceRecog/request에 publish 하여 IoT Rule 에 의해 Lambda를 호출하는 역할을 한다. 이 코드에서 연속적으로 이미지를 request 할수 있고 “close”를 입력하면 프로그램을 종료하게 처리했다.

1. doorLock.js

const awsIot = require("aws-iot-device-sdk");

const doorLock = awsIot.device({

  keyPath: "./credentials/c2a901852c-private.pem.key",

  certPath: "./credentials/c2a901852c-certificate.pem.crt",

  caPath: "./credentials/AmazonRootCA.pem",

  clientId: "doorLock",

  host: "ai4oovsp79d43-ats.iot.us-east-1.amazonaws.com",

});

doorLock.on("connect", function () {

  console.log("Door Lock connected");

  doorLock.subscribe("FaceRecog/notify/door1", function () {

    console.log("subscribing to the topic FaceRecog/notify/door1 !");

  });

  doorLock.on("message", function (topic, message) {

    if (topic == "FaceRecog/notify/door1") {

      var noti = JSON.parse(message.toString());

      console.log(noti);

      if (noti.command == "unlock") console.log(noti.name, ": unlock door1");

      else console.log(noti.name, ": unauthenticated person");

    }

  });

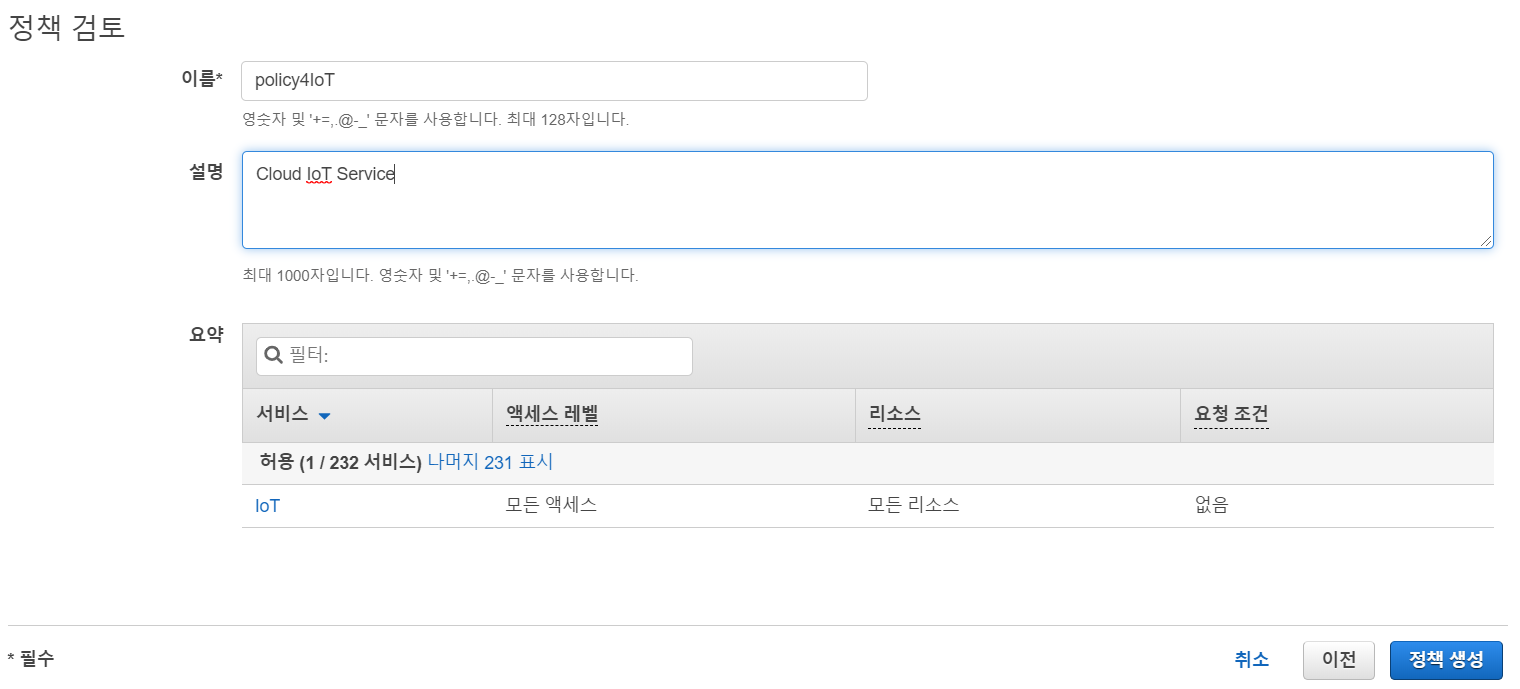
});

doorLock.js 는 doorCamera에서 lambda를 호출하고 lambda에서 해당 이미지에 대한 인증 여부를 command (unlock, reject) 를 FaceRecog/notify/door1에 publish 하면 그 메시지를 받아서 unlock인지, reject인지를 출력하는 역할을 한다.

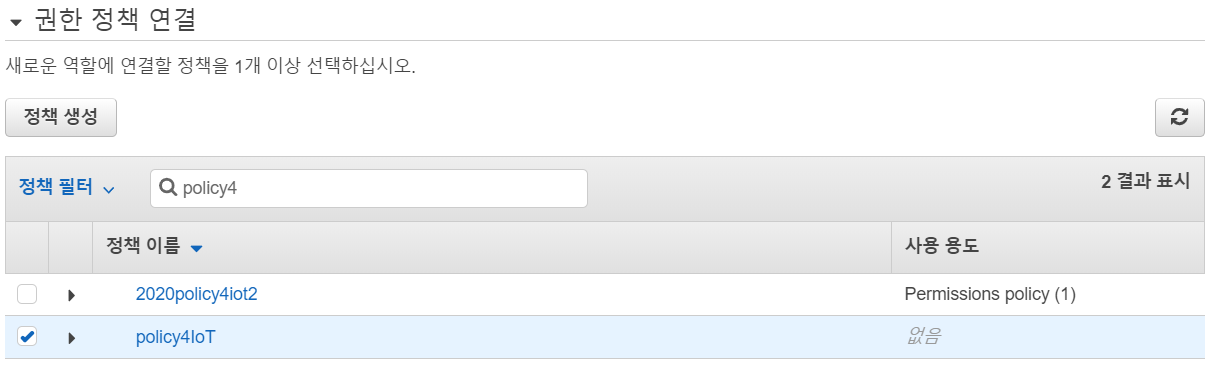
1. IAM Role

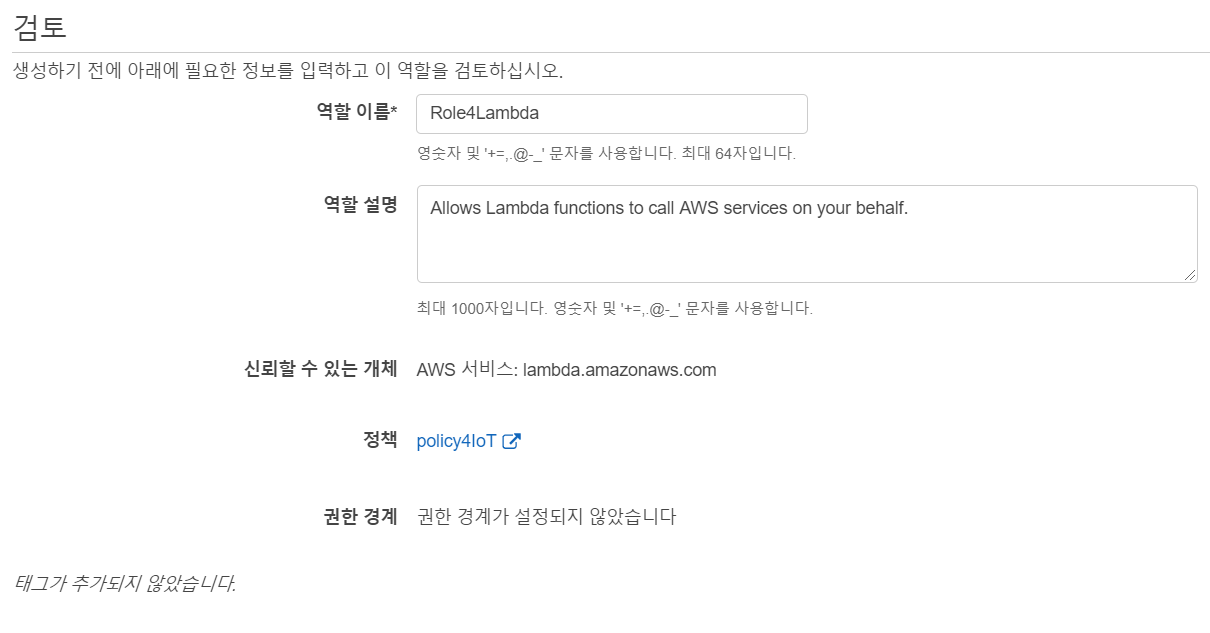
Lambda에서 IoT 를 Access하기 위한 Permission을 담은 Role을 생성한다.



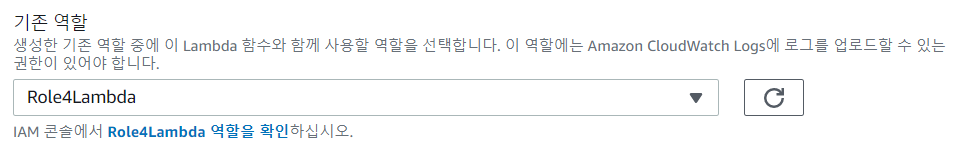


IoT에 대한 모든 권한을 갖는 Policy를 생성한 후에 이 Policy를 포함하는 Role을 생성한다.

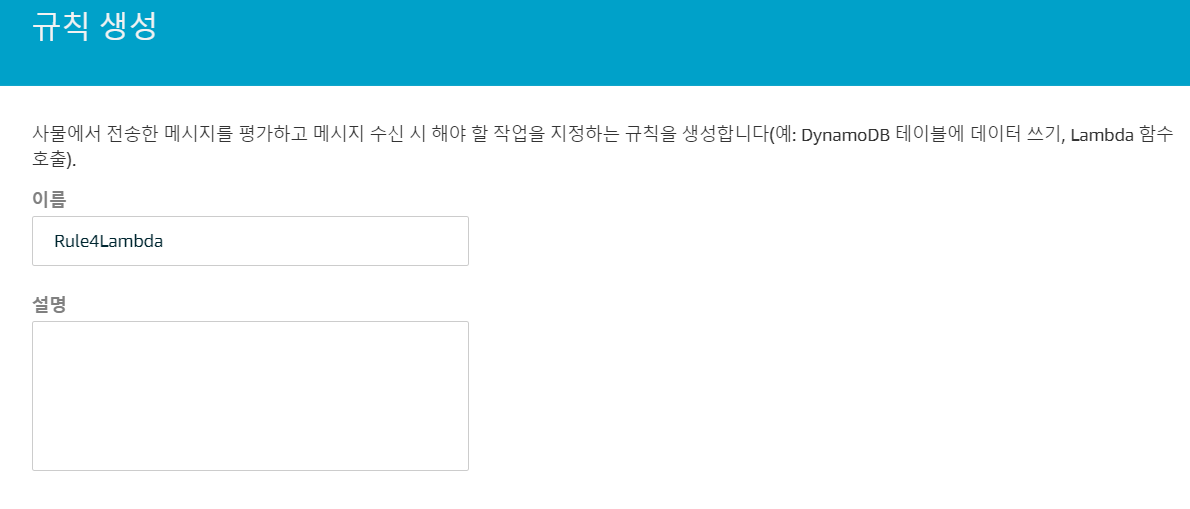




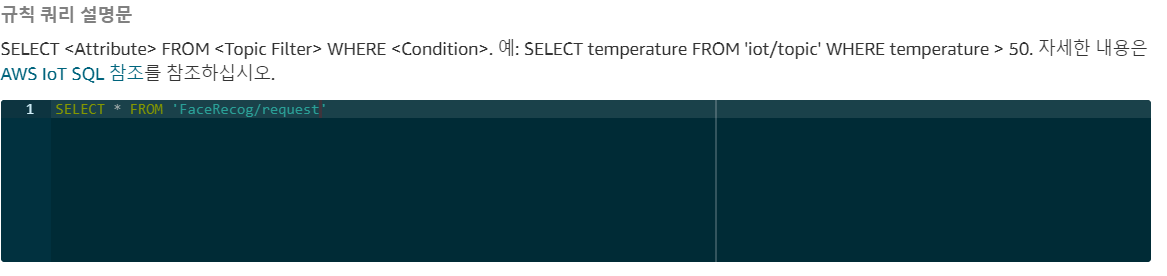
이 Role은 Lambda가 IoT를 사용할 수 있는 Permission 같은 역할을 해준다. 생성을 완료 한다음 Lambda 에서 이 Role을 연결해준다.

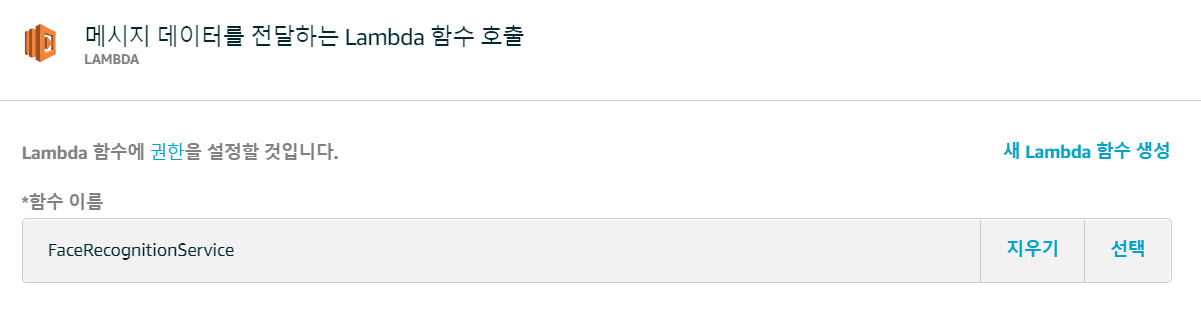


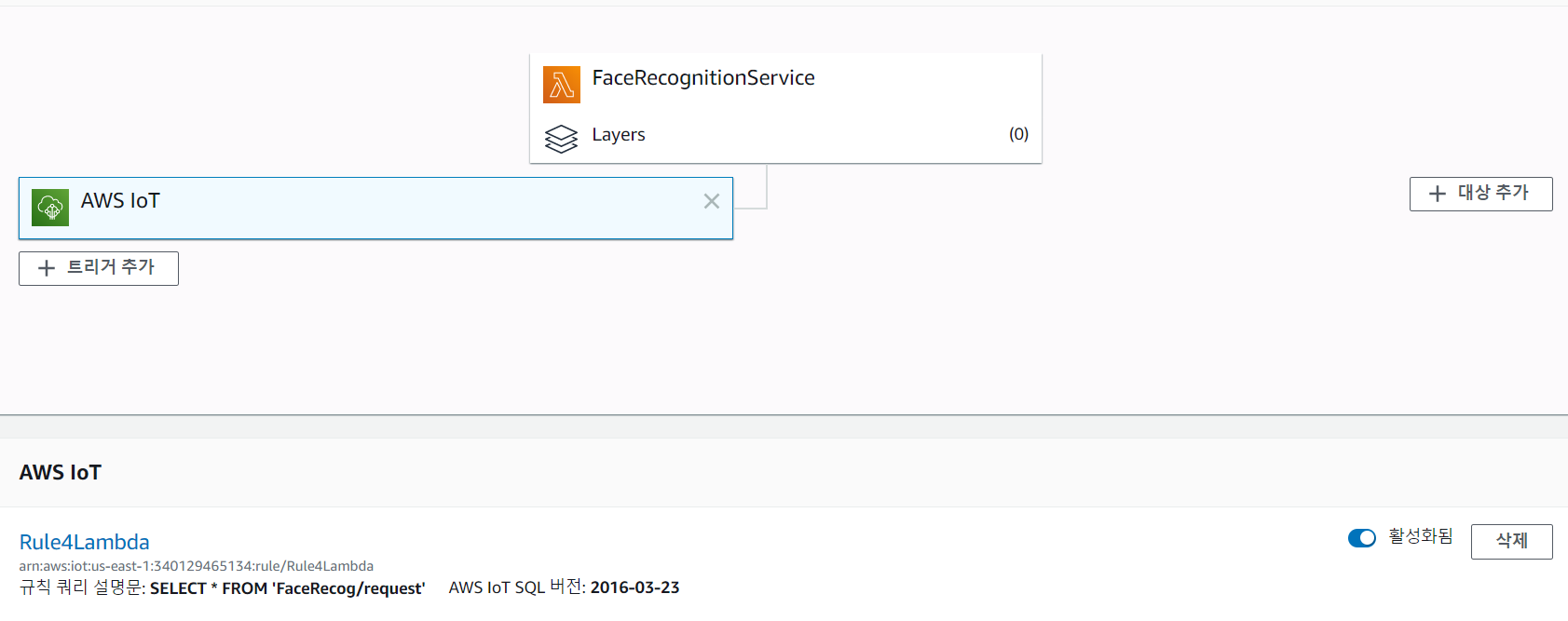
1. IoT Rule



FaceRecog/request에 메시지가 퍼블리시 되면 Lambda를 호출하기 위한 IoT Rule을 생성한다.







Lambda console에서 확인하면 아래 트리거에 AWS IoT의 Rule4Lambda가 추가되었다.

1. Lambda Function (FaceRecognitionService)

Task#1에서 작성한 Function은 IoT와 연결되어있지 않았으므로, 최종적으로 Lambda에 탑재할 코드를 다시 작성한다.

const AWS = require("aws-sdk");

const keys = require("./credentials/credential-keys.js");

AWS.config.update({ region: "us-east-1" });

const iotdata = new AWS.IotData({

  endpoint: "ai4oovsp79d43-ats.iot.us-east-1.amazonaws.com",

  accessKeyId: keys.credential.accessKeyId,

  secretAccessKey: keys.credential.secretAccessKey,

});

const client = new AWS.Rekognition({

  accessKeyId: keys.credential.accessKeyId,

  secretAccessKey: keys.credential.secretAccessKey,

});

exports.handler = async function (event, context) {

  const visitor = { bucket: event.bucket, name: event.name };

  const DB = ["jennie.jpg", "jisoo.jpg", "lisa.jpg", "rose.jpg"];

  const FaceCompare = async (index) => {

    const params = {

      SourceImage: {

        S3Object: {

          Bucket: "facerecog-image-db",

          Name: DB[index],

        },

      },

      TargetImage: {

        S3Object: {

          Bucket: visitor.bucket,

          Name: visitor.name,

        },

      },

      SimilarityThreshold: 70,

    };

    const result = await client

      .compareFaces(params, function (err, data) {

        if (err) console.log(err, err.stack);

        else {

          return data.FaceMatches;

        }

      })

      .promise();

    return result;

  };

  const facematch = [];

  for (var i = 0; i < 4; i++) {

    facematch[i] = await FaceCompare(i);

    if (facematch[i].FaceMatches != "") {

      const params = {

        topic: "FaceRecog/notify/door1",

        payload: JSON.stringify({ name: event.name, command: "unlock" }),

        qos: 0,

      };

      const res = await iotdata.publish(params).promise();

      return { statusCode: 200, result: res };

    }

  }

  const params = {

    topic: "FaceRecog/notify/door1",

    payload: JSON.stringify({ name: event.name, command: "reject" }),

    qos: 0,

  };

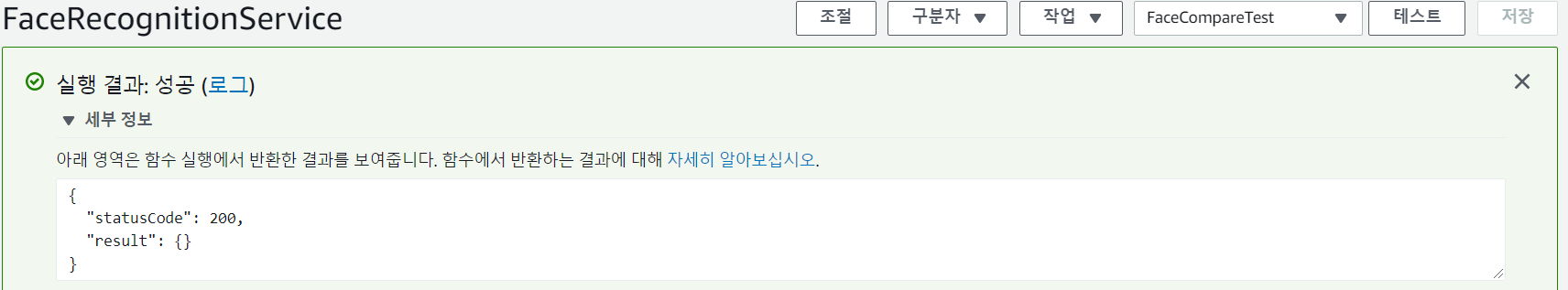
  const res = await iotdata.publish(params).promise();

  return { statusCode: 200, result: res };

};

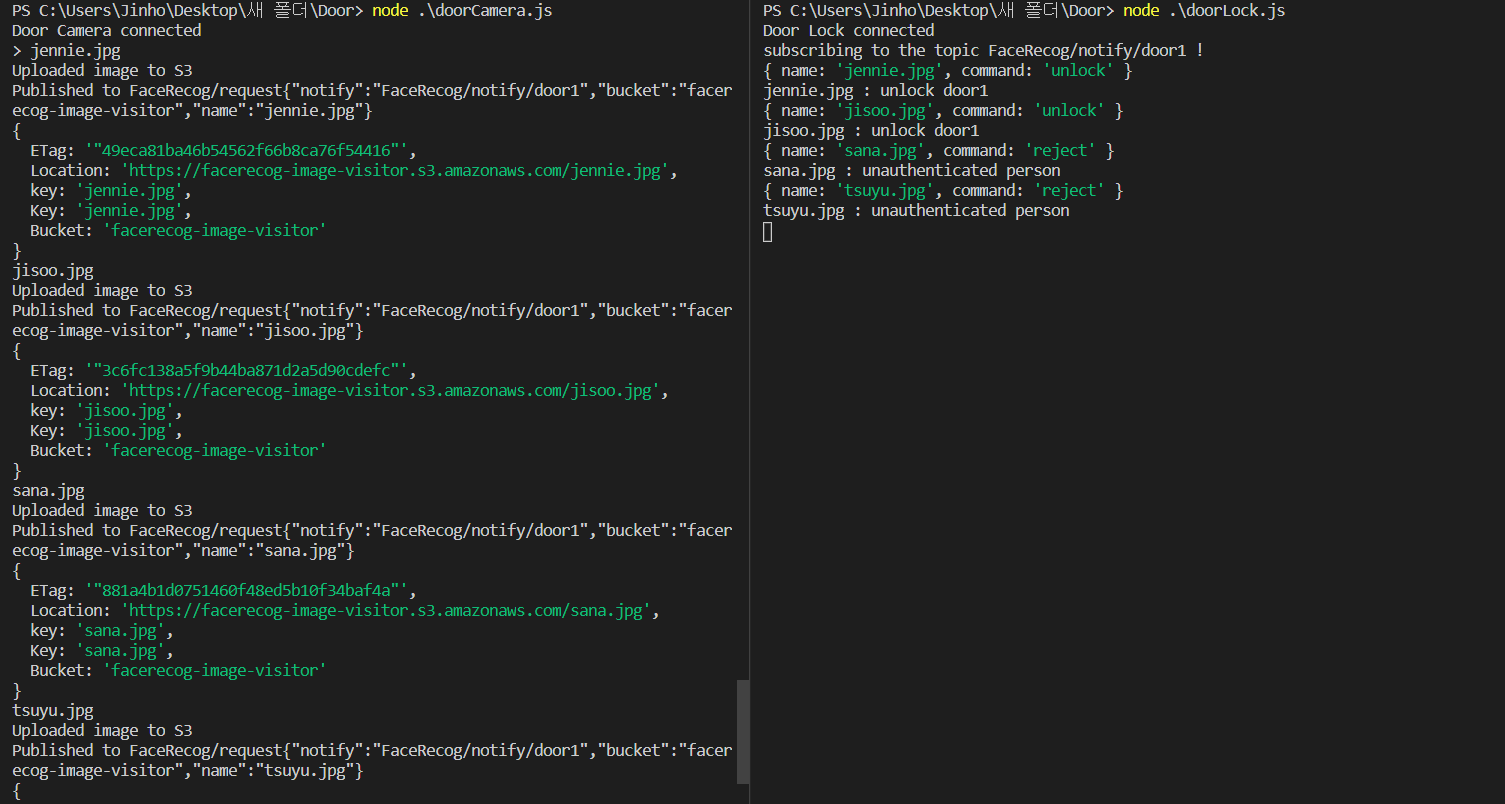
Task#1에서의 코드에 IoTData 객체를 생성한 후 Rekognition에서 얼굴을 비교한 결과 인증된 방문자이면 unlock을 command로 해서 publish 하고, 인증되지 않은 방문자이면 reject를 command로 해서 publish한다. 이때 publish를 하기도 전에 return 되지 않도록 promise()를 사용해서 동기화를 시킨다.

* **Test**





Lambda에서 먼저 테스트를 실행해보면 정상적으로 IoT Core의 테스트에서 FaceRecog/notify/door1 으로 메시지가 오는것을 알 수 있다.



doorCamera 에서 방문자 제니, 지수, 사나, 쯔위를 차례대로 입력하면 블랙핑크 멤버인 제니와 지수는 오른쪽 doorLock에서 unlock이 출력됨을 볼수 있고 블랙핑크 멤버가 아닌 사나와 쯔위는 unauthenticated person이 출력됨을 볼 수 있다.