

<PROJECT 최종보고서>

AWS Lambda 에 기반한 수조관리 시스템

| 조 최용철, 이지훈, 우주

엄현상 교수님

윤석찬 담당자님(AWS)

Table of Contents

1.	Abstract.....	3
2.	Introduction	3
3.	Background Study.....	4
A.	관련 접근방법/기술 장단점 분석	4
B.	프로젝트 개발환경.....	7
4.	Goal/Problem & Requirements.....	7
5.	Approach	7
6.	Project Architecture.....	8
A.	Architecture Diagram	8
B.	Architecture Description.....	8
7.	Implementation Spec.....	9
A.	Input/Output Interface	9
B.	Inter Module Communication Interface.....	13
C.	Modules	13
8.	Solution.....	14
A.	Implementations Details.....	14
B.	Implementations Issues	18
9.	Results	19
A.	Experiments.....	19
B.	Result Analysis and Discussion	19
10.	Division & Assignment of Work.....	20
11.	Conclusion	20
◆	[Appendix] User Manual	21
1.	21

A.	21
---------	----

1. Abstract

이 프로젝트에서는 AWS의 여러 서비스를 사용하여 수조관리를 편리하게 할 수 있는 기계 및 모바일 어플리케이션을 제작하는 것을 목표로 한다. 우리가 계획한 것은 수조의 온도, 조도, pH를 실시간으로 측정해 AWS DynamoDB에 저장하고 값들의 통계치를 모바일 어플리케이션으로 볼 수 있게 하는 것이다. 또 사용자가 모바일 어플리케이션을 통해 물고기들의 급식 조절, 수조 주위의 조명 조절까지 할 수 있도록 하는 것이다.

2. Introduction

1) 프로젝트 수행 목표 달성을 배경 및 중요성

많은 사람들이 물고기를 키우고 집에 자신만의 어항을 가지고 있는 경우가 많다. 물고기를 키울 때 어항(수조) 관리가 중요하다. 이 때 어항 밀도, 여과기 관리, 주기적인 수질 점검, 주기적인 환수, pH 조절, 온도 조절, 조명 조절 등이 물고기를 키울 때 주의해야 하는 점이다. 여기서 어항 밀도, 여과기, 수질 점검 및 환수는 온라인으로 관리하기 어렵다. 하지만 pH, 온도, 조명 등을 모바일로 관리하고 모니터링 할 수 있게 할 수 있고 관리 부주의로 인해 물고기가 죽는 것을 어느 정도 막을 수 있을 것이다. 또 현재 시판되는 자동 급식 조절기는 자신이 주고 싶을 때 줄 수 있는 것이 아니라 일정 시간 간격으로 먹이를 주게 되어 있는 것이 많다. 그것도 12시간, 24시간 간격으로 설정 가능하다. 이런 불편함을 해소하기 위해 이 프로젝트를 수행하게 되었다.

2) 접근 방법

우리가 제작하려는 것은 시중의 키트(즉 아두이노 보드와 Intel Edison 및 Grove sensor)와 AWS, Android Application을 이용한 수조관리 시스템이다. 많은 사람들이 물고기를 키우는데 집 밖에서는 현재 물고기의 환경을 알 수 없다. 그런 불편함을 해소하기 위해 어디서든지 모바일 어플리케이션을 통해서 현재 수조의 환경을 확인할 수 있고, 급식 조절 및 조명 조절도 어플리케이션을 통해 할 수 있는 어플리케이션을 만들려고 한다. 이런 기능들을 구현하기 위해 아두이노 보드와 모바일 어플리케이션 사이를 AWS의 많은 서비스들을 사용하는 것, 특히 Lambda에 기반해서 만드는 것이 목표이다.

3) 보고서의 구성

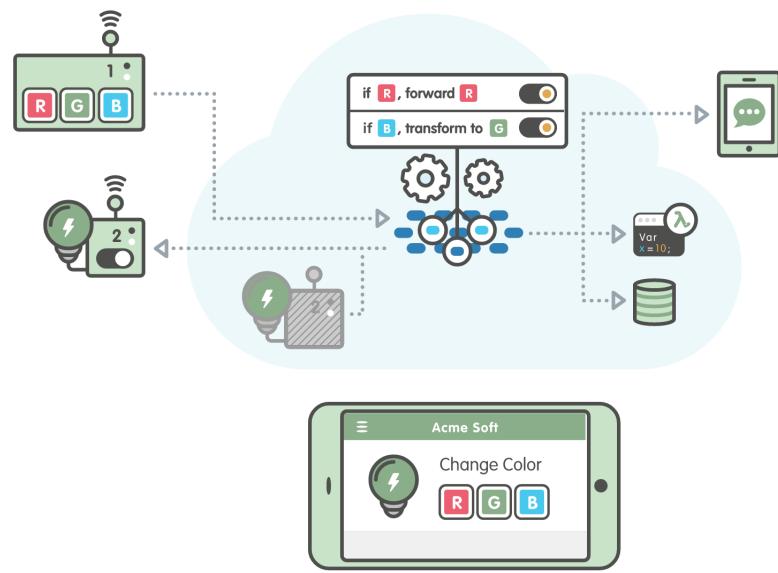
본 보고서는 이번 프로젝트를 하면서 사용한 AWS 서비스들에 대한 Background Study 내용과 Requirement, Approach를 차례로 설명한다. 또 프로젝트 architecture diagram과

어떤 식으로 작동하는지 설명한다. Implementation Spec에 자세히 명시되어 있으며, 성능 테스트의 결과 및 User Manual도 첨부하였다.

3. Background Study

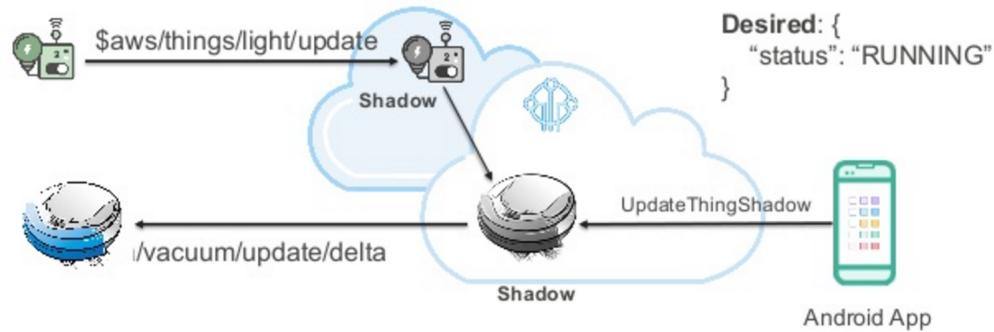
A. 관련 접근방법/기술 장단점 분석

1) AWS IoT



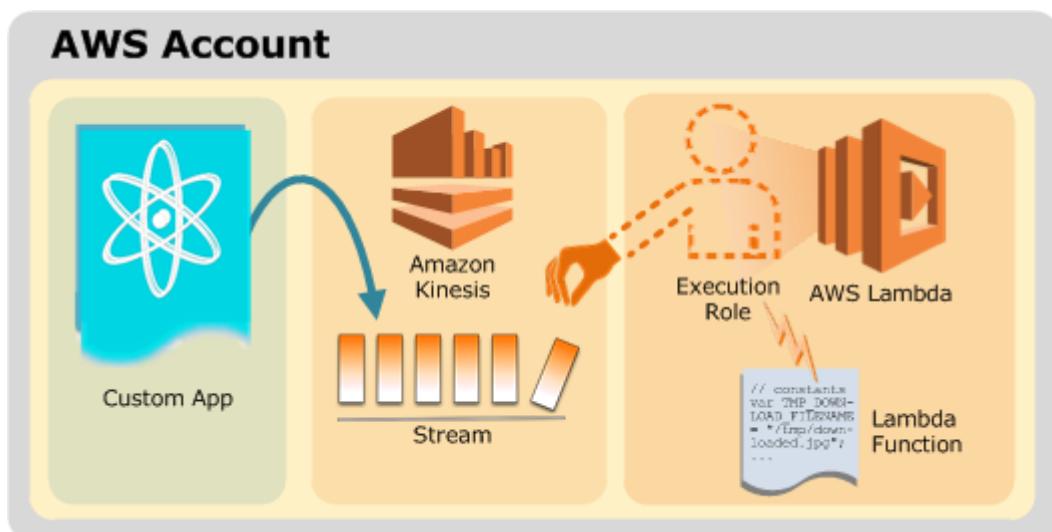
AWS IoT는 Amazon Web Service에서 이번에 새로 런칭한 서비스이다. AWS IoT는 디바이스를 클라우드에 쉽게 연결할 수 있게 해주는 서비스이다. 위 그림의 가운데 부분이 AWS IoT의 핵심이다. 위 그림을 설명하자면, 1번과 2번이 실제 기계이다. 1번을 통해서 2번의 불빛 색깔이 결정되는 그런 기계이다. 이를 Aws IoT에 연결하면, 관리가 쉬워진다. 먼저 만약 2번이 꺼져있다고 해도 AWS IoT에서는 shadow를 생성해 놓아서 shadow를 명령에 맞게 업데이트하고, 2번이 실제로 켜지면 shadow와 동기화 해준다. 위 그림에서 cloud 내부의 2번이 shadow이다. 또 AWS IoT는 Rule을 이용해서 AWS의 다른 서비스들과도 쉽게 연결할 수 있게 해준다. 위 그림에서도 DB와 Lambda function과 연결되어 있다. 마지막으로 아래 쪽에 있는 핸드폰에서도 불빛 색깔을 조절할 수 있게 Rest API를 제공해준다. 이 서비스를 이용해서 쉽게 디바이스와 모바일을 연결할 수 있다.

2) AWS IoT Shadow



AWS IoT Shadow에 대해서 좀 더 공부하였다. IoT Shadow는 기기의 상태 정보를 JSON 형식으로 저장하고 있다. 이 때 “reported” 값은 현재 센서의 상태를 의미한다. 예를 들어 `{"reported": {"servoOn": false, "relayOn": false}}` 이런 형식의 reported 값이 현재 state라고 하자. 이것이 의미하는 것은 servoOn 즉 서브모터가 꺼져있다는 뜻이다. 또 relay switch 도 off 상태라는 것을 의미한다. 또 다른 값으로는 “desired”와 “delta”가 있다. “desired”는 바꾸고 싶은 상태를 의미한다. 위에서 예를 들은 JSON 데이터 뒤에 `{"desired": {"servoOn": true, "relayOn": false}}`가 같이 있다고 하자. 그렇다면 서브모터를 켜고 싶다는 뜻이 된다. “delta”는 자동으로 생성되는데 “reported”와 “desired” 사이의 상태가 다른 것들의 정보를 담고 있다. 위의 예를 이어가자면, “reported”的 servoOn 값과 “desired”的 servoOn 값이 다르므로 “delta” JSON에 포함된다. 하지만 “relayOn”은 “reported”와 “desired”的 상태가 같기 때문에 “delta” JSON에 포함되지 않는다.

3) AWS Lambda



AWS Lambda는 이벤트에 의해서 해당 코드를 실행해주는 역할을 한다. AWS Lambda의

이벤트는 AWS의 다른 서비스들이 될 수 있다. SNS의 알림, Kinesis stream에 메시지 도착 등을 이벤트 리소스로 지정을 하면, 해당 이벤트가 발생될 때마다 자동으로 실행된다. 이번 프로젝트에서는 Lambda를 AWS 내부의 서비스들 사이의 연결고리로 사용한다. Kinesis와 DynamoDB, Kinesis와 SNS 사이를 Lambda가 연결해준다. 위 그림이 우리 서비스에서의 lambda 역할을 설명해준다. 또 제어 명령을 수행하는데 사용된다. 모바일 어플리케이션에서 Lambda function을 invoke 할 수가 있는데, invoke가 되면 AWS IoT thing shadow의 상태를 업데이트 해 실제 기구와 동기화하게 해준다. 이 때 update란 위에서 설명했듯이 “desired”를 특정 값으로 지정해주는 것을 의미한다. 이것은 aws sdk에서 지원하는 updateThingShadow 함수를 통해서 할 수 있다.

4) Intel Edison



Intel Edison은 wearable devices와 IoT 기기들을 제작하기 위한 소형 컴퓨터라고 할 수 있다. 사양을 설명하자면 500MHz Intel Atom Silvermount dual-core processor를 탑재했으며, 1GB DDR3 RAM, Flash 4GB, Wi-Fi, Bluetooth 등을 지원한다. Yocto Linux를 사용하며 C, C++, Python, NodeJS 등을 지원한다. 이번 프로젝트에서는 NodeJS를 이용하여 개발하였다.

5) Sensors

이번 프로젝트를 위해 총 4개의 센서를 사용하였다. Grove temperature sensor, Grove Light Sensor, DFRobot Analog pH meter, HC-SR04 ultrasonic sensor가 그것이다. 먼저 앞의 2개는 “johnny-five” 라이브러리를 이용해 센서 수집이 구현되었고, 나머지는 intel에서 제공하는 “upm” 라이브러리를 사용해 구현하였다.

B. 프로젝트 개발환경

버전 관리 및 소스코드 관리는 Github을 통해서 한다.

Android Application은 자바로 구현을 하고, Android Studio를 사용해서 제작한다.

Arduino Edison과 Grove Sensor로 값을 측정하고 AWS로 보내는 기능은 Node.js로 구현한다. 이 때 jsupm, jhonny-five 라이브러리를 사용한다.

AWS에서는 DynamoDB, IoT, SNS, Kinesis, Lambda 5가지 서비스를 주로 이용한다. Lambda 프로그래밍은 Node.js로 한다.

서버 모니터링 페이지는 Ruby on Rails를 이용하여 제작할 것이다.

4. Goal/Problem & Requirements

이번 프로젝트는 관리 부주의로 물고기가 잘 죽는다는 것에서 출발하였다. 물고기를 위해서는 주기적이고 지속적인 관리가 필요해서 수조의 환경을 관측하고 제어하는 기능을 구현해 사용자가 편리하게 수조의 환경을 관리하는 것을 목표로 한다. 요구 사항으로는, 먼저 수조의 수온, pH, 수위, 조도를 측정하는 것이다. 측정한 값을 데이터베이스에 저장하고 사용자가 쉽게 값을 확인할 수 있는 모바일 어플리케이션을 만드는 것이 두 번째 요구사항이다. 마지막으로 사용자가 어플리케이션을 통해서 급식 및 조명을 조절할 수 있는 기능을 구현할 것이다. 추가적으로 수조의 환경에서 대량의 데이터를 수집하여 어떤 환경에서 물고기가 잘 크는지 분석하는 것을 목표로 한다.

회사 평가표에 의한 정량적인 요구사항은 4개 이상의 AWS 서비스로 클라우드 백엔드 구성하기, 4개 이상의 센서 이용, 수조 시스템 스펙 4개에 대한 구현이 있다.

5. Approach

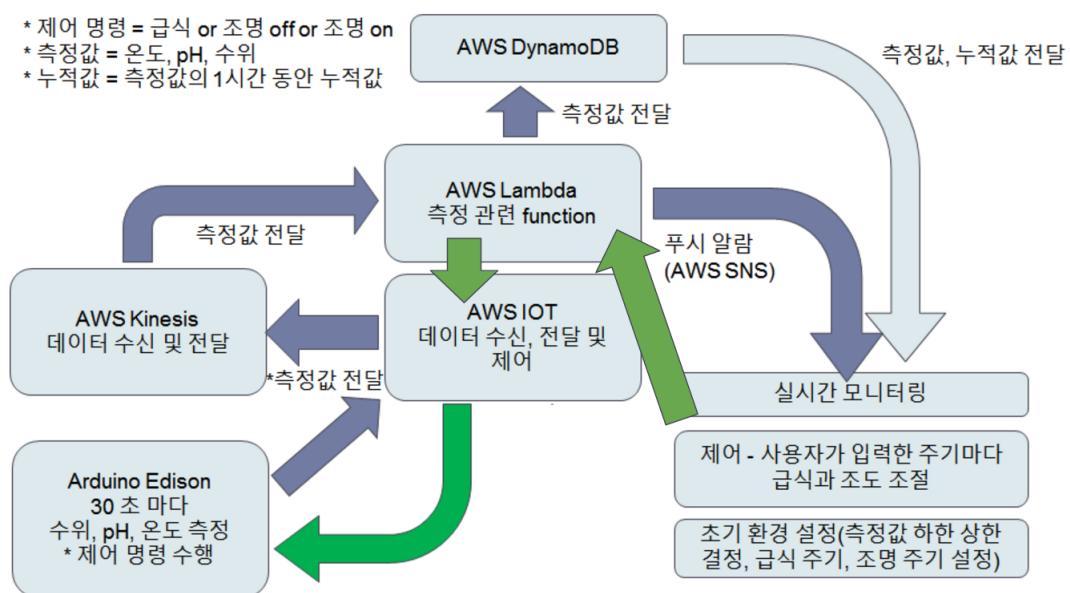
앞서 말했듯이 이번 프로젝트는 관리 부주의로 인해서 물고기가 잘 죽는다는 것에서 출발하였다. 이러한 관리 부주의가 일어나는 원인은 우리가 하루 종일 물고기를 관리할 수 없고 문제가 일어났을 때 바로 알 수 없어서이다. 그렇다면 수조 관리를 위한 값을 우리가 어디서든 확인할 수 있고, 현재 상태가 이상하다는 알림을 주고, 조절할 수 있다면 그런 문제는 줄어들 것이다. 이런 관점에서 이번 프로젝트의 목표는 사용자가 편리하게 수조를 관리할 수 있는 기계 및 모바일 어플리케이션을 제작하는 것을 목표로 한다.

필요한 기능들을 구현하기 위해서 AWS의 여러 서비스들을 적극적으로 이용한다. 먼저

아두이노의 센서들로 수집한 데이터를 처리하기 위해서 AWS IoT와 Kinesis를 사용한다. 또 Lambda 함수를 이용해서 Kinesis stream의 데이터를 DynamoDB에 저장한다. 데이터베이스에 저장된 값들은 나중에 최적 환경을 분석하기 위해서 사용되고, 모바일 어플리케이션에서 모니터링을 위해 사용된다. 모바일 어플리케이션에서 아두이노를 제어하는 것도 Lambda function에 의해 이루어진다. 또 이상값 알람을 주기 위해서는 AWS SNS 서비스를 이용해서 topic에 구독하고 있는 앱으로 푸시 알람을 준다.

6. Project Architecture

A. Architecture Diagram



B. Architecture Description

이번 프로젝트는 크게 아두이노 에디슨, AWS, 모바일 어플리케이션 세 부분으로 나눌 수 있다.

첫 번째로 아두이노 에디슨은 센서를 통해 수위, pH, 온도, 조도 등을 측정하여 AWS로 전송하는 역할을 한다. 아두이노 에디슨은 30초마다 값을 측정한다. 또 모바일 어플리케이션에서 보낸 제어 명령을 처리해 급식 조절과 조명 조절을 하는 역할을 한다.

두 번째로 AWS는 아두이노 에디슨과 모바일 어플리케이션의 연결고리 역할을 한다. 먼저 AWS IoT는 아두이노 에디슨의 측정값들이 가장 먼저 들어오는 서비스이다. AWS IoT를 통해 들어온 데이터들은 AWS IoT Rule을 통해 AWS Kinesis로 전송된다. 그 다음 AWS Lambda function이 실행된다. 이 Lambda function은 AWS Kinesis를 event source로 해서,

Kinesis로 값이 들어오면 실행되는 함수이다. Lambda에서는 Kinesis에서 받은 데이터를 decode한 다음 AWS DynamoDB에 저장한다. 이 때 만약 이상값이 들어온다면 AWS SNS를 통해서 모바일 어플리케이션에 푸시 알람을 주는 역할도 한다.

마지막으로 모바일 어플리케이션은 크게 실시간 모니터링, 제어, 초기 환경 설정으로 나뉜다. 실시간 모니터링은 모바일 어플리케이션이 AWS DynamoDB에 접근해서 데이터를 가져온 다음 그래프 형태로 표시해주는 기능이다. 그리고 제어 기능은 모바일 어플리케이션에서 Lambda function을 부르면 lambda function에서 command를 받아 각 명령에 맞는 상태로 AWS IoT thing shadow를 업데이트한다.

7. Implementation Spec

A. Input/Output Interface

1) Input

- AWS IoT

AWS IoT로의 input은 2가지로 나뉜다.

첫 번째로는 Arduino Edison에서 Sensor들로 측정된 값들이다. Sensor는 Grove light sensor, Grove temperature sensor, HC-SR04, DFRobot analog pH meter들로 총 4 가지이다. 온도는 섭씨로 측정이 되고 조도는 럭스(lux) 단위로 측정이 된다. pH meter는 일반적인 pH 단위로 측정이 되고, HC-SR04, 즉 거리 센서는 옵션에 따라서 센서 미터 또는 인치로 들어온다. 이 센서 input에 대한 자세한 정보는 뒤 Arduino Edison input에 있다. 이런 식으로 측정된 데이터는 JSON 형태로 AWS IoT로 들어온다. 아래 가 JSON 데이터의 예이다.

```
▼ 2015-12-17T03:53:53.558Z d4fcbe6c-2c5c-4a0c-a839-1274f525416d Decoded Data: { device_id: 'fishtank' , time: 1450324427269 , device: 'edison' , sensors: [ { light_level: 1275 } , { temperature: 43.81977028514734 } , { ph: 6.6217451095581055 } , { distance: 15.240549828178693 } ] }
```

두 번째로는 Lambda function에 의한 AWS IoT thing shadow update input이다. 이 인풋은 인자가 JSON 형태로 들어온다. JSON 형태로 {"desired": {"relayOn": true}} 이런 형식으로 들어오는데 relayOn, servoOn 두 가지 input이 있다. 이 때 relayOn은

릴레이 스위치를 on/off하는 input이고 즉 조명을 켜거나 끄는 역할을 한다. servoOn은 서브모터를 rotate 하는 input으로 급식 주기를 조절하는 역할을 한다.

- AWS Kinesis

Kinesis로의 input은 AWS IoT에서 전해주는 데이터이다. AWS IoT에서 정의한 ConnectedMaracaRule에 따라서 connected-maraca로 들어온 데이터, 즉 아두이노에 디스의 센서를 통해 수집한 값을 sql문으로 뽑아서 Kinesis Stream으로의 input으로 전달된다.

- AWS Lambda

ProcessKinesisStream 함수의 input은 위 Kinesis Stream이다. 위 background study에서 말했듯이 AWS Lambda는 이벤트 기반으로 invoke 되는 함수이다. ProcessKinesisStream 함수의 input, 즉 이벤트는 Kinesis Stream에 새로운 메시지가 들어오는 것이고 그 메시지 자체가 input으로 처리된다. 형식은 json이고 아래가 input의 예이다. 이 함수는 이런 input을 받아 이상 값을 발견 시 SNS에 publish하고 DynamoDB에 저장하는 역할을 한다.

```
{  
    "Records": [  
        {  
            "kinesis": {  
                "kinesisSchemaVersion": "1.0",  
                "partitionKey": "partitionKey",  
                "sequenceNumber": "49555882212401685393359815528289191185458542605284409346",  
                "data": "eyJkZXpY2VfaWQiOiJpb3RoYWNrMDAiLCJ0aWlIjoxNDQzNTI201WnZEWLcJkZXpY2UiO1JIZGzb24iLCJzZW5zb3JzIjpbeYJ0ZIwZXJhdHlyZS16NdcUNjYxMTQxNjM3NTA5  
MjczOTIw",  
                "eventSource": "aws:kinesis",  
                "eventVersion": "1.0",  
                "eventID": "shardId-000000000000:49555882212401685393359815528289191185458542605284409346",  
                "eventName": "aws:kinesis:record",  
                "invokerIdentityArn": "arn:aws:iam::240647427454:role/iot-hackseries-aws-loft/iot-hackseries-aws-loft-DeviceExecutionRole-1Y0LNLOAZPUI",  
                "awsRegion": "ap-northeast-1",  
                "eventSourceARN": "arn:aws:kinesis:ap-northeast-1:240647427454:stream/iot-hackseries-aws-loft-DeviceStream-0EUXUHYXDJT1"  
            }  
        ]  
    ]  
}
```

AwsIoUpdate 함수의 input은 모바일 어플리케이션이나 웹에서 전해준다. 이 때 input은 {"cmd": 0 or 1 or 2} 형식으로 들어온다. 이 때 0은 급식 주기 명령이고, 1은 조명 켜기, 2는 조명 끄기 input이다. AwsIoUpdate는 이 input을 받아 적절한 JSON 형식의 input을 만들어 AWS IoT Thing Shadow Update의 input으로 넘겨준다.

- AWS DynamoDB

Dynamo DB로의 input은 ProcessKinesisStream에서 처리한 데이터이다.

- AWS SNS

AWS SNS의 input은 Lambda function에서 해당 이벤트로 publish 하는 것이다. 해당

input은 이상값이 측정될 때만 발생한다.

● Arduino Edison

Edison으로의 input은 2가지가 있다.

첫 번째로는 Arduino Edison에서 Sensor들로 측정된 값들이다. Sensor는 Grove light sensor, Grove temperature sensor, HC-SR04, DFRobot analog pH meter들로 총 4 가지이다. 온도는 섭씨로 측정이 되고 조도는 럭스(lux) 단위로 측정이 된다. pH meter는 일반적인 pH 단위로 측정이 되고, HC-SR04, 즉 거리 센서는 옵션에 따라서 센치미터 또는 인치로 들어온다. 이번 프로젝트에서는 센치미터로 인풋을 받았다. 각 센서는 적절한 라이브러리를 사용했는데, 앞 2개 센서는 “johnny-five” 라이브러리를 사용했고, 뒤 2개는 “jsupm_[sensor name]” 라이브러리를 사용했다. 각각 하나씩 사용한 예를 들어 보자. ‘johnny-five’를 이용한 light sensor는 다음 코드를 변형해서 사용했다.

```
var five = require("johnny-five");
var Edison = require("edison-io");
var board = new five.Board({
  io: new Edison()
});

board.on("ready", function() {

  // Plug the Grove TSL2561 Light sensor module
  // into an I2C jack
  var light = new five.Light({
    controller: "TSL2561"
  });

  light.on("change", function() {
    console.log("Ambient Light Level: ", this.level);
  });
});
```

그리고 ‘jsupm’을 이용한 pH sensor는 다음 코드를 변형해서 사용했다.

```

var sensorObj = require('jsupm_dfrph');

// Instantiate a DFRPH sensor on analog pin A0, with an analog
// reference voltage of 5.0
var sensor = new sensorObj.DFRPH(0, 5.0);

// After calibration, set the offset (based on calibration with a pH
// 7.0 buffer solution). See the UPM sensor documentation for
// calibrations instructions.
sensor.setOffset(0.065);

// Every second, sample the pH and output it's corresponding
// analog voltage.

setInterval(function()
{
    console.log("Detected volts: " + sensor.volts());
    console.log("pH value: " + sensor.pH());
}, 1000);

```

두 번째는 AWS IoT thing shadow 업데이트에 의한 delta 이벤트 발생이다. Thing shadow가 업데이트 되면 delta 이벤트가 발생한다. edisonThingShadow.on('delta') 이런 식으로 delta 이벤트에 listen 하고 있으면 delta 이벤트가 발생하면 desired state로의 업데이트가 이뤄진다.

● Mobile Application

모바일 어플리케이션의 input은 사용자에 의해 발생한다. 사용자가 어플리케이션을 통해서 모니터링, 급식, 조명 조절을 위한 입력들이 input이 된다.

- 메인 페이지에서 DynamoDB에서 축적된 데이터들을 가져오고 불러온 값들을 그래프로 그려준다. 이 때 기간을 설정해서 input을 넣어주면 그 기간 사이의 값들을 가지고 그 그래프를 그려준다.

- 급식 조절 버튼을 클릭하면 AWS SDK를 이용하여 AwsIotUpdate function을 invoke 한다.

- 조명 on/off 토글 버튼을 클릭하면 AWS SDK를 이용하여 AwsIotUpdate function을 invoke 한다.

2) Output

이번 프로젝트에서 사용자가 직접 볼 수 있는 output은 2가지이다. 첫 번째로 모바일 어플리케이션에서 DynamoDB의 데이터로 그린 그래프이다. 온도, 조도, pH, 수위로 꺽은선 그래프를 그려서 보여준다. 그래프를 그릴 때는 javascript 라이브러리를 사용해서 그려준다. 두 번째 output은 급식 조절 및 조명 조절이다. 이 output은 모바일 어플리케이션의 input에서 시작된다. 모바일 어플리케이션에서 AWS Lambda function을 invoke한다. Invoke할 때는 arn을 특정 지어준다.

```
lambda = Aws::Lambda::Client.new( region: ENV['AWS_REGION'] )
resp = lambda.invoke({
  function_name: "arn:aws:lambda:ap-northeast-1:240647427454:function:AwsIoTUpdate", # required
  payload: '{"cmd": 1}'
})
```

B. Inter Module Communication Interface

앞서 말했듯이 모듈은 크게 3가지로 나눌 수 있다. 아두이노 에디슨 보드 및 센서, AWS 서비스, 안드로이드 어플리케이션이 그 3가지이다. AWS 내부에서도 여러 서비스들을 모듈로 나눌 수 있다. 모듈 간 통신은 기본적으로 AWS 서비스들로 이루어진다. 첫 번째로 아두이노와 AWS 사이의 통신은 AWS IoT를 통해서 이루어진다. 아두이노에서는 AWS SDK를 통해서 AWS IoT를 call할 수 있다. AWS SDK의 awsIoT module을 만들고 특정 thing의 topic에 데이터를 publish한다. 이 때 데이터는 json 형태로 보내서 해석하기 편하게 한다. 다음 AWS 내부에서는 위 input 부분에서 설명한대로 모듈 간 통신이 일어난다. AWS IoT의 Rule을 통해 Kinesis Stream에 메시지를 전달하면, AWS Lambda에서 해당 이벤트에 반응해 Kinesis의 메시지를 decode해 측정값들을 가져온다. 그런 다음 Dynamo DB에 값을 전달하고 SNS를 call하기도 한다. 이 모듈 간 통신은 모두 AWS SDK로 가능하다. 다음 AWS와 모바일 어플리케이션 사이의 통신은 AWS SDK를 통해서 일어난다. 모바일 어플리케이션과 아두이노 사이의 통신은 AWS IoT로 인해 이뤄진다. 위에서 설명했듯이 Lambda function으로 AWS IoT에 update 요청을 보내면 이를 shadow에 업데이트해주고, edison board에서 동기화한다.

C. Modules

1) 아두이노 에디슨

아두이노 에디슨은 센서를 통해 온도, 조도, pH, 수위를 측정하는 역할을 한다. 또 서브모터 등을 이용해서 급식 조절, 조명 조절을 한다.

2) AWS

AWS는 아두이노 에디슨에서 데이터를 받아 처리하고 저장한다. 또 모바일 어플리케이션

에서 제어 명령을 받아 아두이노 에디슨에 전달한다.

3) 모바일 어플리케이션

DynamoDB에서 값을 가져와 그래프 형태로 보여준다. 또 급식 조절, 조명 조절 명령을 Aws IoT로 전달한다.

8. Solution

A. Implementations Details

- Edison Node.js function – ConnectedMaraca.js

Intel Edison + Arduino 의 자세한 구현은 다음과 같다. Maraca.js 파일이 핵심적인 기능을 수행하며 그 외는 외부 라이브러리에 해당된다.

var five = require("johnny-five");	Johnny Five 라이브러리
var Edison = require("edison-io");	Edison 보드의 IO 를 관리하는 라이브러리
var awsIoT = require('aws-iot-device-sdk');	AWS-IoT SDK. thingShadow 와 같은 IoT 통신에 이용되는 모듈을 포함한다.
var songs = require("j5-songs");	
var groveSensor = require("jsupm_grove");	Grove sensor 라이브러리
var sensorObj = require('jsupm_dfrph');	pH 센서 라이브러리
var servoModule = require("jsupm_servo");	servo 라이브러리
var usonicModule = require("jsupm_hcsr04");	초음파 센서 라이브러리

Basic Modules

device	실제 IoT device 에 해당하는 module. 관련된 다양한 이벤트를 처리한다.
edisonThingShadow	서버와의 동기화를 위한 가상 device module. 관련된 다양한 이벤트를 처리한다.

board	보드에 해당하는 module.
temp	온도계 모듈에 해당된다. johnny-five 라이브러리의 temperature module 의 instance 이다. 섭씨 온도를 반환하는 역할을 한다.
light	조도계 모듈에 해당된다. Grove Sensor 라이브러리의 GroveLight module 의 instance 이다. Lux 로 나타낸 조도를 반환하는 역할을 한다.
ph	pH 센서 모듈에 해당된다. jsupm_dfrph 라이브러리의 DFRPH module 의 instance 이다. pH 농도를 반환하는 역할을 한다.
relay	조명 조절을 위한 릴레이 스위치에 해당하는 module. Johnny-Five 라이브러리의 Relay 모듈의 instance 에 해당한다. relay.on(), relay.off() 의 함수를 통해 릴레이 스위치를 다룰 수 있다.
servo	급식을 위한 서브 모터에 해당하는 module. Jsupm_servo 라이브러리의 instance 이다. servo.setAngle() 함수를 통하여 서브 모터를 조작한다.
usonic	초음파 센서에 해당하는 module. Jsupm_hcsr04 라이브러리의 instance 이다. usonic.getDistance(1) 함수를 통하여, cm 단위의 거리를 반환한다.

Basic Flow

device.on('connect, ...)	디바이스가 연결되었음을 알린다.
edisonThingShadow.on('delta', ...)	thingShadow 가 업데이트 되어 desired 값과 reported 값이 다를 때의 이벤트이다. 기본적으로 servo 나 relay 를 조작하는 역할을 한다.
edisonThingShadow.update()	thingShadow 를 업데이트한다. 이는 조작이 완료된 이후에 변화를 원위치 시키는 역할을 포함한다.

- AWS Lambda function – ProcessKinesisStream

Event.Records.forEach로 kinesis stream에서 읽어 온 데이터들을 처리한다. 그리고 이 상값이 들어올 경우 sns에 publish한다.

```

if(outlier) {
    sns.publish({
        TopicArn:'arn:aws:sns:ap-northeast-1:240647427454:universe-DeviceSNSTopic-4MS7BFUA81NT',
        Message: "Error",
        Subject: "Error",
        function(err,data){
            if (err) {
                console.log("Error sending a message "+err);
            } else {
                console.log("Sent message: "+data.MessageId);
            }
        });
}

```

또 dynamodb.putItem 함수로 db에 값을 저장한다.

```

dynamodb.putItem({
    "TableName": tableName,
    "Item": {
        "device_id": {
            "S": payload.device_id
        },
        "time": {
            "N": JSON.stringify(payload.time)
        },
        "device": {
            "S": payload.device
        },
        "light_level": {
            "N": JSON.stringify(payload.sensors[0].light_level)
        },
        "temperature": {
            "N": JSON.stringify(payload.sensors[1].temperature)
        },
        "pH" : {
            "N": JSON.stringify(payload.sensors[2].ph)
        },
        "distance" : {
            "N": JSON.stringify(payload.sensors[3].distance)
        },
        "source": {
            "S": "from lambda"
        }
    }
}

```

여기서 N은 number 즉 숫자 데이터를 의미하고 S는 string 데이터 타입을 의미한다.

- AWS Lambda function – AwsIoTUpdate

먼저 new AWS.IoTData로 shadow를 업데이트할 수 있는 객체를 하나 만든다.

```

var iotdata = new AWS.IoTData({
    endpoint: 'https://ANH2C5EYQYWLX.iot.ap-northeast-1.amazonaws.com',
    accessKeyId: 'AKIAJV7J0VJBXUK4TD7A',
    secretAccessKey: 'L+Tjnuf2qWxRQAQ/VUG0geCSVbHPubsTScQwIbDH'
});

```

그 다음 이 함수를 call한 어플리케이션에서 값을 받아온 후 알맞은 JSON string을 만든다.

```

var received = '{"state": {"desired": {';

if(event.cmd === 0){
    received += '"servoOn": true}}}' ;
}

else if(event.cmd === 1){
    received += '"relayOn": true}}}' ;
}

else{
    received += '"relayOn": false}}}' ;
}

```

마지막으로 shadow를 update 해준다.

```

var params = {
    payload: received, /* required */
    thingName: 'ConnectedMaraca' /* required */
};

iotdata.updateThingShadow(params, function(err, data) {
    if (err) console.log(err, err.stack); // an error occurred
    else {
        console.log(data);           // successful response
        context.succeed('SUCCESS');
    }
});

```

- Dashboard App

Sensor.scan	DynamoDB와 연결한 후 전체 item을 scan한다.
Sensor.query_latest(limit)	DynamoDB와 연결한 후 device_id가 fishtank인 것들을 최신 것부터 limit만큼 가져온다.
Sensor.query(start_time, end_time)	Start_time과 end_time 사이의 item 값을 가져온다.
Shadow.feed	AwsIotUpdate lambda function을 invoke한다. 급식 시작
Shadow.lightOn	AwsIotUpdate lambda function을 invoke한다. 조명 켜기
Shadow.lightOff	AwsIotUpdate lambda function을 invoke한다. 조명 끄기
ApiController sns	AWS SNS가 publish하는 주소
DashboardController index	메인 페이지

B. Implementations Issues

- 1) 가장 문제가 되었던 부분은 AWS-IoT 의 thingShadow 와의 동기화 문제였다. AWS에서 제공하는 샘플 코드를 사용하였으나, 업데이트가 되지 않는다는 사실을 발견했다. 디버깅을 하는 과정에서 time out detecting code 를 추가한 결과 업데이트 과정에서 타임아웃이 발생하는 것이 원인임을 알게 되었다. 보드 내에서 중복으로 레지스터를 하는 부분이 문제가 된다고 생각하고 board.on('ready') { ... } 내부에서 외부로 레지스터 코드를 옮긴 후에, 모든 업데이트가 완료 될 때마다 unregister 코드를 실행하는 것으로 문제를 해결하였다.
- 2) 센서와 보드의 호환성 문제로 인해 문제가 발생하였다. 특히, ds18b20 이라는 방수 온도계를 장착하고 Johnny-Five 라이브러리를 사용하려 하였으나, 인텔 에디슨 보드가 one-wire 을 지원하지 않는다는 이유로, 사용할 수 없었다. galileo-IO 를 이용하여, one-wire 프로토콜을 실행하는 것을 고려해보았으나, 구현 과정에서 실패하였고 방수가 되지 않는 grove temperature 센서를 이용하되, 걸을 얇은 비닐로 감싸 최대한 온도 전달이 늦어지지 않도록 하였다.
- 3) Servo 를 사용하는 과정에서 반복적으로 시스템이 멈추는 문제가 발생했다. 다양한 라이브러리를 적용해보았으나 같은 결과가 나타났고, 조사를 더 진행한 결과 Servo 가 순간적으로 사용하는 전력이 커서 시스템이 다운된 것이라는 사실을 알게 되었고, 9V 건전지로 외부 전원을 연결해주자 문제가 사라졌다. 이후 건전지의 전력이 소모되면 같은 문제가 발생하였다.

9. Results



A. Experiments

- 1) ConnectedMaraca 함수를 통해 publish 된 값이 어플을 통해 바로 확인할 수 있는지 테스트하였다.
- 2) 그래프를 기간 설정하여 확인할 수 있는지 테스트하였다.
- 3) 밥 주기와 조명 조절이 제대로 되는지 테스트하였다.
- 4) 측정값마다 데이터를 제대로 받아오는지 테스트하였다.
- 5) 그래프가 에러 없이 제대로 그려지는지 테스트하였다.

B. Result Analysis and Discussion

- 1) 가장 최근에 업데이트 된 측정값을 표시함을 확인 하였다. pH 센서를 갑자기 식초에 넣음으로서 가장 최근 값이 30초 이내로 업데이트 됨을 확인하였다. 조명을 끄면서 가장 최근의 조도가 30초 이내로 업데이트 됨을 확인하였다. 더불어, 극단적인 값을 입력하면 여러 메세지를 표시하는 것도 확인 되었다.
- 2) 그래프에 나타나는 기간을 표시할 수 있다. 사용자가 직접 입력을 통해서 설정할 수도 있

고, 버튼 식으로 설정할 수도 있다.

- 3) 모두 실시간으로 이루어지며, [밥 주기]의 경우 누르면 한 번만 실행된다. 조명 관련 버튼은 누르면 상태가 유지된다. 실험을 통해서 모두 30초 이내로 실행이 되는 것을 확인하였다. 실행 반응 속도는 3초부터 24초까지 다양한 분포를 띠었다.
- 4) 측정값을 바뀌면 그래프가 바뀌는 것을 확인하였다.
- 5) 설정 <2>, <4> 에 따라 그래프가 적절하게 표현되는 것을 확인하였다. <1>의 실험의 경우에는 pH 그래프가 급강하는 것이 확인되었다. 데이터가 없을 경우에 데이터 없음을 출력하는 것을 확인하였다.

10. Division & Assignment of Work

항목	담당자
Lambda 프로그래밍	우주
데이터 분석 시스템 개발	이지훈
기구 설계 및 제작	최용철
센싱 및 제어	이지훈
네트워크 기능 구현	우주
서버 모니터링 페이지	최용철
모니터링 기능	이지훈
제어 기능	우주
UI/UX 구성	최용철

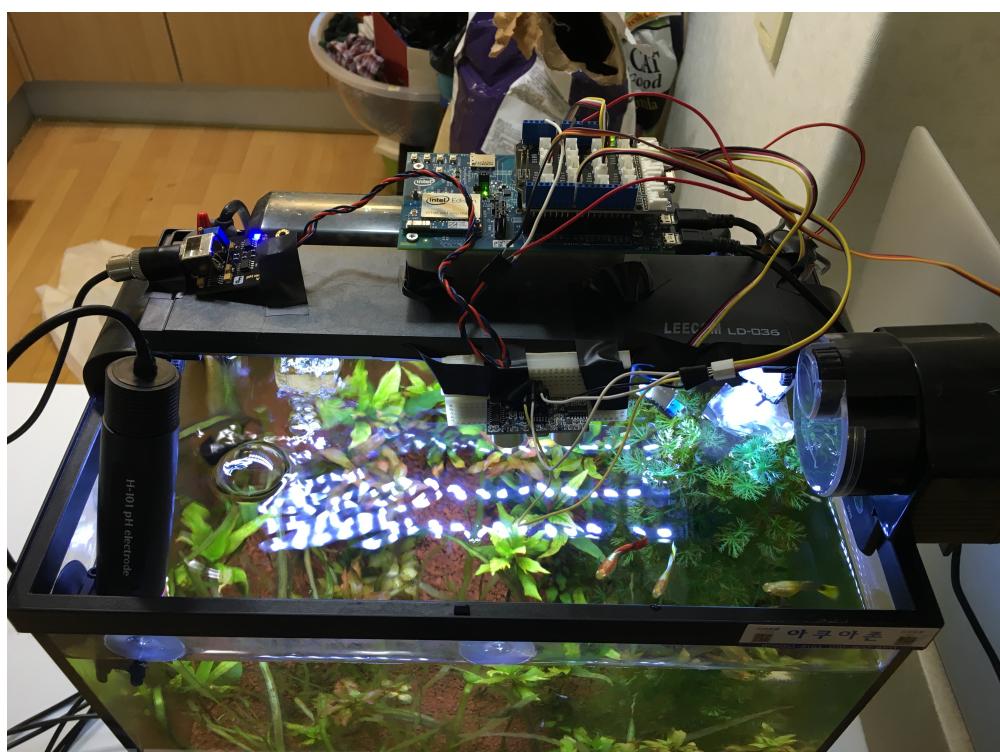
11. Conclusion

기존 수조 관리 시스템보다 발전된 형태의 좀 더 편하게 관리할 수 있는 기기를 만들 수 있었다. AWS의 여러 서비스들을 이용하면서 서버단의 안정성을 높일 수 있었고 AWS IoT 등의 서비스로 직접 구현하기는 힘든 기능을 구현할 수 있었다.

◆ [Appendix] User Manual

1. 기기 설치

- A. 조도 센서를 조명의 밑에 둔다.
- B. 수온 센서를 물 속에 고정한다.
- C. 급식 기기를 어항의 옆 면에 고정한다.
- D. 초음파 센서를 수면에 수평이 되도록 5cm 떨어뜨려 설치한다.
- E. 조명과 조명 조절 기기를 설치한다.
- F. pH 센서를 어항의 옆 면에 고정한다.
- G. 기기가 모두 설치되면 다음과 같다.



2. Application 조작

- A. 앱을 다운로드 받거나 혹은 <https://drfish.herokuapp.com/> 로 바로 접속한다.
- B. 앱 구성



<1>

가장 최근에 업데이트 된 수온, 조도, 수위, pH 농도를 표시해 준다. 만약 이중에서 이상값이 하나라도 확인되면, 푸시 알람이 표시된다.

<2>

그래프에 나타나는 기간을 표시할 수 있다. 사용자가 직접 입력을 통해서 설정할 수도 있고, 버튼 식으로 설정할 수도 있다.

<3>

기기 조절 명령을 나타낸다. 모두 실시간으로 이루어지며, [밥 주기]의 경우 누르면 한 번만 실행된다. 조명 관련 버튼은 누르면 상태가 유지된다.

<4>

표시를 원하는 측정값의 종류를 선택한다. 수온, 조도, 수위, pH 중 하나를 선택할 수 있다.

<5>

설정한 기간의 측정값을 그래프로 확인할 수 있다.