<PROJECT 최종보고서>

AWS IoT Backend 를 이용한

Soil tracker System

고성빈, 권현석, 함지웅

엄현상 교수님

윤석찬 (아마존웹서비스 코리아 테크 에반젤리스트)

Table of Contents

1. Abstract

최근 컴퓨팅 기기의 소형화에 힘입어 사물인터넷(IoT = Internet of Things) 기술이 발달하고 있습니다. 신용카드 만한 크기의 디바이스에서 다양한 작업을 할 수 있게 되었고, 클라우드 서비스의 발달에 힘입어 인터넷 연결을 통해 더욱 더 많은 유용한 서비스를 만들 수 있게 되었습니다. 본 프로젝트 팀은 이런 IoT 환경과 클라우드 서비스인 AWS(Amazon Web Services)를 이용하여 식물 생장에 필요한 토양 환경을 모니터링 할 수 있는 시스템을 제작하였습니다.

사용자는 이 시스템을 활용하여 원격으로 토양 환경을 관리할 수 있으며, 문제가 발생한 경우 알림을 받을 수 있습니다. 함께 제공되는 모바일 앱은 편의성을 더욱 높여줄 것입니다.

2. Introduction

토양 품질은 식물의 생장에 매우 중요한 영향을 끼칩니다. 따라서 이를 지속적으로 관리하고 적절한 조치를 취하는 것은 식물들을 키우는 사용자의 중요한 역할 중 하나라고 할 수 있습니다.

본 프로젝트는 이러한 사용자들을 주 목표로 하여 손쉽게 그리고 언제 어디서나 식물이 자라는 토양 환경을 모니터링 할 수 있는 수단을 제공하는 것을 목적으로 합니다.

사람이 직접 토양 옆에 머물면서 혹은 일정 시간마다 항상 확인하기는 어렵습니다. 특히 개인 재배자의 경우에는 다른 일로 바쁜 경우 아무런 도움 없이는 토양 관리가 어려울 수 밖에 없으며, 규모가 넓은 경우에도 모든 면적을 사람이 일일이 관리하기는 쉽지 않습니다.

본 프로젝트는 이러한 문제를 해결하기 위해 IoT 디바이스를 이용한 해결 방안을 제시합니다. 관리하고자 하는 토양에 IoT 디바이스를 배치하고, Pre-Set 상태로 제공되는 간단한 웹 서비스 설정만으로 언제 어디서나 스마트폰이 있다면 토양 환경을 관리할 수 있도록 하는 것입니다.

보고서는 다음과 같은 순서로 구성됩니다.

먼저 관련된 기존의 접근 방법에 비해 본 프로젝트의 접근이 가지는 이점에 대한 분석과 프로젝트 개발 환경에 대한 설명이 있고, 이후에 프로젝트가 해결하고자 하는 문제와 해결 방법을 구현한 프로젝트 전체의 구성을 소개합니다. 이후에는 각 모듈에 대한 설명과 데이터의 입력과 사용자 단으로의 출력, 그리고 실제로 어떻게 프로젝트가 구현되었는지에 대한 설명이 소개됩니다. 마지막으로 실제 테스트 결과와 분석, 추후 개선점 등에 대한 내용이 이어질 예정입니다.

3. Background Study

A. 관련 접근방법/기술 장단점 분석

토양 환경을 비롯해 식물 생장의 전반적인 요소들을 모니터링 하는 시스템이 아예 없는 것은 아닙니다. 하지만 대부분 현재 존재하는 솔루션들은 대규모 재배 사용자를 목표로 하며, 따라서 일반 사용자가 구입하기에 가격이 저렴한 편이 아닙니다. 이는 기존 솔루션이 대규모 재배와 함께 상업적인 재배를 목적으로 하기 때문이기도 합니다. 상업적인 재배의 경우 챙겨야 할 요소들이 소규모 개인 사용자들에 비해 많을 수밖에 없고, 따라서 솔루션의 규모와 가격 역시 그에 비례하는 만큼 증가하는 것입니다.

하지만 본 프로젝트는 기존 솔루션 수준의 관리가 필요 없거나, 소규모 관리 혹은 취미로 식물을 재배하는 사용자들을 위한 소규모의, 그러나 효율적인 솔루션을 제공합니다. 상대적으로 저렴한 가격에 식물 생장에 필수적인 요소들을 모니터링 할 수 있고, 필요한 경우에는 더 많은 IoT 디바이스를 활용해 확장이 가능한 솔루션을 제공합니다.

B. 프로젝트 개발환경

본 프로젝트는 다음과 같은 개발 환경에서 개발을 진행했습니다.

IoT 디바이스 : Intel Edison, Arduino Board

모니터링 센서 : Grove Sensors (Light, Moisture, Temperature, UV)

기타 장치 : Grove RGB LCD

개발 컴퓨터 : No preference (Windows, Mac, Linux all OK)

모바일 앱 : Android (Web App with wrapper)

4. Goal/Problem & Requirements

본 프로젝트가 해결하고자 하는 문제는 이미 서술했듯이, 비교적 저렴한 가격과 적은 시간을 통해 소규모 사용자가 효율적으로 토양 환경을 모니터링 할 수 있는 솔루션을 제공하는 데 있습니다. 이를 위해 다음과 같은 것들이 요구됩니다.

1) 사용자가 신경을 쓰지 않아도 IoT 디바이스가 안정적으로 작동할 것

2) 토양 환경을 모니터링 한 결과를 모바일 앱을 통해 확인할 수 있을 것

3) 확장성을 위해 한 사용자가 여러 디바이스를 모니터링 할 수 있을 것

4) 사용자가 원하는 조건을 벗어나는 경우 알림을 제공할 수 있을 것

5) 저렴한 가격을 통해 효율성을 추구할 것

5. Approach

[프로젝트 수행 목표 또는 문제 정의 및 접근방법]

본 프로젝트는 위에서 서술한 요구 사항을 만족하기 위해, 3개의 모듈로 나누어져 구성되었습니다.

실제 토양 환경을 모니터링 하여 서버로 전송하는 디바이스 파트, 디바이스가 수집한 데이터를 보관하고 가공하여 사용할 수 있도록 서버의 역할을 하는 AWS 파트, 마지막으로 사용자의 편리한 토양 모니터링을 위한 모바일 앱 파트입니다.

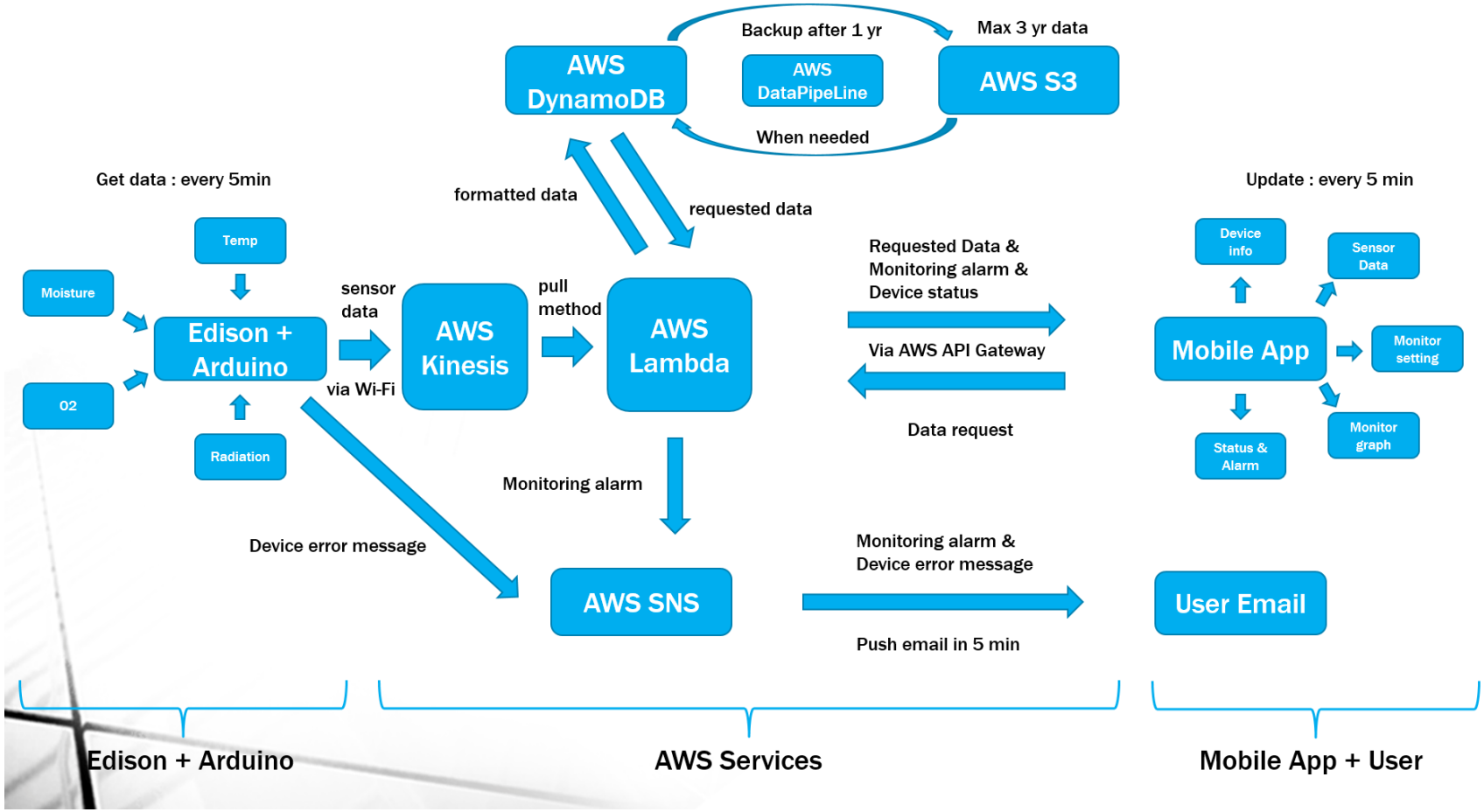
디바이스에서 수집한 데이터는 AWS의 다양한 서비스를 거쳐 처리되고 별도의 데이터베이스에 저장됩니다. 또한 데이터베이스에는 디바이스와 모니터링 조건, 사용자 정보를 담고 있는 테이블도 존재하는데, 이는 사용자가 단순히 하나의 디바이스만을 사용하는 것이 아니라 여러 환경에서 여러 디바이스를 사용하는 경우를 고려한 접근입니다. 모바일 앱은 사용자 편의를 위해 존재하며, 현재 디바이스의 상태를 보고, 디바이스와 모니터링 조건을 관리하며, 그래프를 통해 누적된 데이터를 볼 수 있도록 다양한 기능을 제공합니다.

6. Project Architecture

A. Architecture Diagram

본 프로젝트의 전체 아키텍처는 다음 페이지의 그림으로 대체합니다.

파트 B에서 아키텍처의 각 부분에 대한 자세한 설명을 진행합니다.


AWS DynamoDB
Backup after 1 yr
AWS DataPipeLine
When needed
Max 3 yr data
AWS S3
Get data : every 5min
formatted data
requested data
Update : every 5 min
Temp
Moisture
O2
Radiation
Edison + Arduino
sensor data
via Wi-Fi
AWS Kinesis
pull method
AWS Lambda
Requested Data & Monitoring alarm & Device status
Via AWS API Gateway
Data request
Device info
Sensor Data
Mobile App
Monitor setting
Monitor graph
Status & Alarm
Monitoring alarm
Device error message
AWS SNS
Monitoring alarm & Device error message
Push email in 5 min
User Email
Edison + Arduino
AWS Services
Mobile App + User

B. Architecture Description

본 프로젝트의 아키텍쳐에 대한 자세한 설명은 다음과 같습니다.

1) Edison + Arduino 파트

이 파트는 데이터를 수집하고, AWS 서비스로 데이터를 보내는 역할을 합니다. 수집하는 데이터는 4종류이며, 5분 간격으로 데이터를 수집하게 됩니다. 토양 환경의 경우 급격하게 변하는 경우가 거의 없다고 봐도 무방하므로, 짧은 데이터 수집 간격은 오히려 쓸모 없는 데이터의 수집을 불러일으키고, 이를 처리해야 하는 AWS 서비스 또한 많이 사용하게 되므로 가격 효율성 면에서도 떨어지게 됩니다. 데이터 전송은 AWS에서 제공하는 대용량 데이터 스트림인 AWS Kinesis 서비스를 이용합니다. 한편, 디바이스는 몇 가지 종류의 에러를 자체적으로 검출할 수 있고, 에러가 검출된 경우 사용자에게 푸시 알림을 보내게 됩니다. 자세한 사항은 8-B Implementation Details에서 서술합니다.

2) AWS Services 파트

이 파트는 디바이스에서 수집한 데이터를 처리하여 데이터베이스에 저장하고, 모바일 앱에서 데이터 요청이 있을 경우 적절한 데이터를 적절히 가공하여 제공하는 역할을 합니다. 또한 디바이스에서 수집한 데이터가 현재 사용자가 설정한 모니터링 범위를 넘어서는 경우, 이에 대한 푸시 알림을 사용자에게 전송합니다.

3) Mobile App 파트

이 파트는 사용자의 편의성을 위한 파트입니다. AWS Service에 저장된 데이터는 직접 커맨드라인 인터페이스를 사용해서도 볼 수 있지만 이는 본 프로젝트가 목표로 하는 사용자의 편의성 증진에 어긋나므로, 사용자가 언제 어디에서나 인터넷만 있다면 현재 상태를 모니터링 할 수 있도록 모바일 앱을 제공합니다. 모바일 앱에서는 현재 디바이스 상태 확인, 센서 데이터 그래프 확인, 디바이스 관리, 모니터링 조건 관리, 모바일 앱 옵션 설정 등의 작업을 수행할 수 있습니다.

7. Implementation Spec

A. Input/Output Interface

본 프로젝트의 가장 중요한 두 가지 입력은 디바이스의 센서 데이터와 사용자의 모바일 앱 조작입니다.

디바이스의 센서 데이터 입력은 아날로그 센서를 통해 수집한 데이터를 디지털 값으로 Edison +

Arduino 시스템에서 변환하여 시작됩니다. 변환된 데이터는 인터넷 연결을 통해 5분 간격으로 kinesis 스트림에 써지며, AWS Lambda가 이를 가져와 처리하고 데이터베이스에 저장하게 됩니다.

모바일 앱 조작은 사용자가 사용한 기능 (예를 들어 디바이스 추가, 모니터링 조건 수정 등)에 대한 API를 호출하여 해당하는 AWS Lambda 함수를 트리거하고, 트리거의 결과를 다시 모바일 앱이 받아서 사용자에게 돌려주는 방식으로 입력이 처리됩니다.

본 프로젝트의 중요한 출력은 모바일 앱에서 이루어지는 사용자 조작의 결과물입니다. 바로 전 문단에서 서술했듯이, 모바일 앱은 사용자가 조작한 기능에 대한 API를 호출하여 결과를 받아옵니다. 그리고 이 결과에 따라 먼저 HTTP Status Code를 먼저 확인하여 제대로 결과가 반환되었는지 보고, 에러가 있다면 적절한 에러 메시지를, 정상이라면 사용자가 요청한 데이터를 반환합니다.

B. Inter Module Communication Interface

각 파트간의 소통은 다음과 같이 이루어집니다.

1) IoT 디바이스 & AWS Service

이 커뮤니케이션은 Wi-Fi 네트워크를 기반으로 하며, Kinesis에 IoT 디바이스가 데이터를 쓰는 형식으로 진행됩니다. 이를 위해 putKinesis() 함수가 있으며, 이를 뒷받침하는 다양한 함수들이 AWSHelperFunctions.cpp 파일에 구현되어 있습니다.

2) AWS Service & 모바일 앱

이 커뮤니케이션은 API를 통해 이루어집니다. 각 AWS Lambda 함수에 대해 각각의 API가 존재하며, 모바일 앱이 이 API를 호출하여 AWS Lambda 함수를 실행시키고, 결과 역시 API의 호출의 반환값으로 받게 됩니다.

3) IoT 디바이스 & 사용자

이 커뮤니케이션은 IoT 디바이스에 에러가 발생한 경우, 사용자에게 직접 푸시 이메일을 보내는 방식으로 작동합니다. 이를 위해 putSns() 함수가 있으며, 디바이스는 이 함수를 호출해 AWS SNS 서비스에 있는 정보를 이용하여 사용자에게 에러 정보를 담은 푸시 이메일을 보내게 됩니다.

4) AWS Service & 사용자

이 커뮤니케이션은 IoT 디바이스에서 수집한 데이터가 사용자가 설정한 모니터링 조건을 벗어났을 때 푸시 이메일을 보내는 방식으로 작동합니다. 이를 위해 AWS Lambda는 디바이스에서 수집한 데이터를 처리할 때 현재 모니터링 조건에 부합하는지 여부를 체크하고, 그렇지 않다면 AWS

SNS 서비스를 invoke하여 사용자에게 모니터링 정보와 현재 센서 정보를 담은 푸시 이메일을 전송하게 됩니다.

C. Modules

아키텍쳐 부분에서 이미 서술했듯이, 본 프로젝트는 3개의 모듈로 이루어집니다. 각각은 IoT 디바이스 모듈, AWS 서비스 모듈, Mobile App 모듈이며, 이 중에서 디바이스 모듈과 Mobile App 모듈은 사용자의 요구에 따라 여러 개가 존재할 수 있습니다. 사용자가 디바이스를 여러 개 관리하고 싶은 경우, 간단하게 디바이스를 여러 개 설정하고, 모바일 앱에서 등록하는 것으로 완료됩니다. 또한, 여러 명의 사용자가 하나의 AWS 서비스를 사용하여 여러 개의 디바이스를 관리하고 싶은 경우, 모바일 앱에서 사용자 이름을 다르게 설정하고 각 사용자의 디바이스를 등록하는 것으로 쉽게 관리할 수 있게 됩니다.

8. Solution

A. Implementation Details

1) IoT 디바이스 파트

이 파트는 Arduino IDE와 C++ 언어를 사용하여 개발되었습니다.

직접 구현한 코드는 "Soiltracker.ino"(C++) 파일에 집중되어 있으며, 기본적으로 AWS나 Grove 센서의 제작사에서 제공되는 라이브러리도 필요에 맞게 수정하였습니다.

디바이스 파트에서는 가장 중요하게 고려한 부분이 얼마나 디바이스가 안정적으로 서버에 데이터를 전송하는가와 에러가 발생했을 때 얼마나 알려줄 수 있고 얼마나 복구가 가능한지 입니다.

안정적으로 서버에 데이터를 전송하기 위해서는 서버와의 연결이 안정적이어야 하고, 데이터 수집 또한 정상적으로 이루어져야 합니다. 서버 연결의 안정성을 위해서는 디바이스 부팅 과정에서 Wi-Fi에 연결하는 과정과 Kinesis 스트림에 데이터를 쓰는데 가장 중요한 타임 스탬프 문제를 해결하기 위한 Time Sync Server와의 연결에 대해 실패하면 일정 회수를 다시 시도하도록 프로그램을 작성하였습니다. 이를 통해 처음에 연결을 실패하더라도 지속적으로 시도함으로써 개발 도중에 문제가 발생했던 연결 초기화 문제를 어느 정도 해결할 수 있었습니다.

데이터 수집에 대해서는, 센서가 수집하는 데이터가 안정화 될 때까지 시간이 어느 정도 필요하다는 점에 착안하여 센서로부터 데이터를 수집하는 간격은 실제로 데이터를 Kinesis 스트림에 쓰

는 간격보다 훨씬 짧게 5초로 설정하였습니다. 이를 통해 안정적인 데이터 수집과 조금 더 빠른 센서 에러 검출이 가능해졌습니다.

에러가 발생했을 때, 개발 중간까지의 디바이스 구성으로는 아무런 정보를 알 수 없었습니다. IoT 디바이스는 간결한 기능만을 가지고 있으므로, 자체적으로 디스플레이를 갖고 있거나 하지 않아 어떻게 작동하고 있는지 보려면 Serial 연결을 통해 컴퓨터에 연결한 상태여야만 했습니다. 이를 해결하고 사용자가 디바이스에 에러가 발생했는지, 발생했다면 어떤 에러가 발생했는지 쉽게 알 수 있도록 각 에러 케이스에 대해 에러 코드를 정의하고, 디바이스에 RGB LCD를 장착하여 현재 에러 상태를 글과 RGB 색으로 동시에 제공할 수 있도록 구성하였습니다.

또한 에러가 발생한 경우 지속적인 모니터링을 위해 복구 방법이 필요한데, 현재는 잘못된 데이터의 저장을 막기 위해 데이터 포스팅 간격을 매우 길게 변경하여 데이터 저장을 막고 사용자에게 알리는 선에서 그쳐있습니다. 하지만 보완을 통해 스스로 에러를 복구하고 정상 상태로 돌아갈 수 있도록 하는 코드를 추가할 예정입니다.

2) AWS Service 파트

AWS 서비스는 직접 코드를 작성한 부분은 AWS Lambda function들이며, 다른 서비스는 서비스 설정을 직접 진행하는 방식으로 개발되었습니다.

AWS Lambda 서비스는 본 프로젝트의 가장 중심을 이루는 부분이며, 데이터 처리와 데이터 가공 및 데이터 제공 기능을 담당합니다. 디바이스 데이터 처리를 위한 함수, 모바일 앱에서 데이터를 요청하는 경우 원하는 데이터를 찾아 정제하여 제공하기 위한 함수들을 모두 합쳐 15개의 Lambda fucntion이 제작되었으며, 각각의 함수는 모두 다른 기능을 제공합니다. 제공되는 함수들은 크게 "디바이스 데이터 처리", "사용자 정보 관리", "디바이스 정보 관리", "모니터링 정보 관리" 함수들로 나눠볼 수 있습니다. 자세한 Lambda function code는 본 프로젝트의 Github 링크에서 찾아볼 수 있으며, 각 Lambda function 의 정보는 부록에서 찾아볼 수 있습니다.

AWS DynamoDB 서비스는 본 프로젝트에서 사용되는 데이터베이스 서비스로, 본 프로젝트가 데이터베이스에 쓰는 작업과 데이터를 읽는 작업이 많고, 로그 스타일의 데이터가 지속적으로 기록된다는 점을 고려하여 비관계형 데이터베이스 서비스를 선택했습니다. 테이블은 "센서 데이터 정보", "센서 최신 데이터 정보", "사용자 정보", "사용자 디바이스 정보", "사용자 모니터링 조건 정보", "디바이스 정보", "디바이스 모니터링 정보"의 7개의 테이블로 구성되어 있습니다. 자세한 데이터베이스 구성은 부록에서 찾아볼 수 있습니다.

AWS API Gateway 서비스는 AWS 서비스와 모바일 앱이 통신하는 역할을 담당합니다. 디바이스

데이터를 처리하는 Lambda function을 제외한 나머지 모든 함수들은 각각의 API를 갖고 있으며, Mobile App은 이를 invoke하여 원하는 데이터를 얻고, 사용자, 디바이스, 모니터링 조건 정보를 관리할 수 있습니다. 모두 14개의 API가 제작되었습니다.

AWS SNS 서비스는 사용자에게 푸시 이메일 서비스를 제공하는 역할을 담당합니다. 사용자가 AWS SNS 서비스에 자신의 이메일을 등록하면, 디바이스에서 에러가 발생하거나, 모니터링 조건 범위를 벗어나는 데이터가 수집된 경우 등록된 이메일로 에러 정보를 담은 푸시 이메일이 발송됩니다. 모두 2개의 토픽으로 구성되어 있으며, 각각 디바이스 에러, 모니터링 범위 초과 알림을 담당합니다.

B. Implementations Issues

1) IoT 디바이스 – Time Sync 문제

디바이스에서 AWS Kinesis 스트림에 데이터를 쓰기 위해서는 유효한 타임스탬프가 필요합니다. 하지만 IoT 디바이스를 처음 설정했을 때는 자동으로 시간을 맞추지 않으므로, 현재 시간과 맞지 않는 타임 스탬프가 생성되는데, 이를 해결하기 위해 NTP 서버(http://2.kr.pool.ntp.org)를 이용한 시간 동기화를 진행하였고, 서버에서 요구하는 시간대에 맞추기 위해 현재 시간과는 다르지만 UTC로 동기화를 진행했습니다.

2) IoT 디바이스 – 에러 검출 및 표시 문제

디바이스에서 어떤 에러가 발생했는지 알기는 쉽지 않습니다. 따라서 RGB LCD를 장착하고, 에러 코드를 정의하여 이를 바탕으로 최소한의 정보를 제공하도록 디바이스를 구성했습니다. 에러가 발생한 경우 사용자는 디바이스의 LCD를 이용하여 어떤 에러인지 파악하고 그에 맞는 조치를 취할 수 있습니다.

3) IoT 디바이스 – 에러 자동 복구 문제

디바이스에서 에러가 발생한 경우 현재 구현으로 초기 Wi-Fi 설정 문제, 시간 동기화 문제를 해결할 수 있습니다. 실제 에러가 발생했던 경우들에서 대부분 딜레이를 어느 정도 주고 다시 연결을 시도하면 잘 되었기 때문에, 실패한 경우 다시 시도하도록 구성하여 복구 가능하도록 하였습니다. 또한 센서 에러의 경우에는 발생했을 때 Kinesis 포스팅을 멈추고 센서 안정성을 유지하기 위한 데이터 수집만 수행하도록 하며, 사용자가 다시 센서를 고쳤을 때, 추가적 시간 지연 없이 바로 정상적으로 데이터를 전송할 수 있도록 하여 데이터 손실을 최소화했습니다. Kinesis에 데이터를 쓸 때 에러가 발생한 경우에는 일정 시간 지연 후 Kinesis를 다시 초기화한 후, 같은 데이터를 다시 전송하도록 진행하였습니다. 위의 방법들로 완전한 에러 자동 복구가 가능하지는 않겠지만 간

단한 에러들은 스스로 복구하거나 사용자가 최소한의 역할만으로 복구할 수 있도록 합니다.

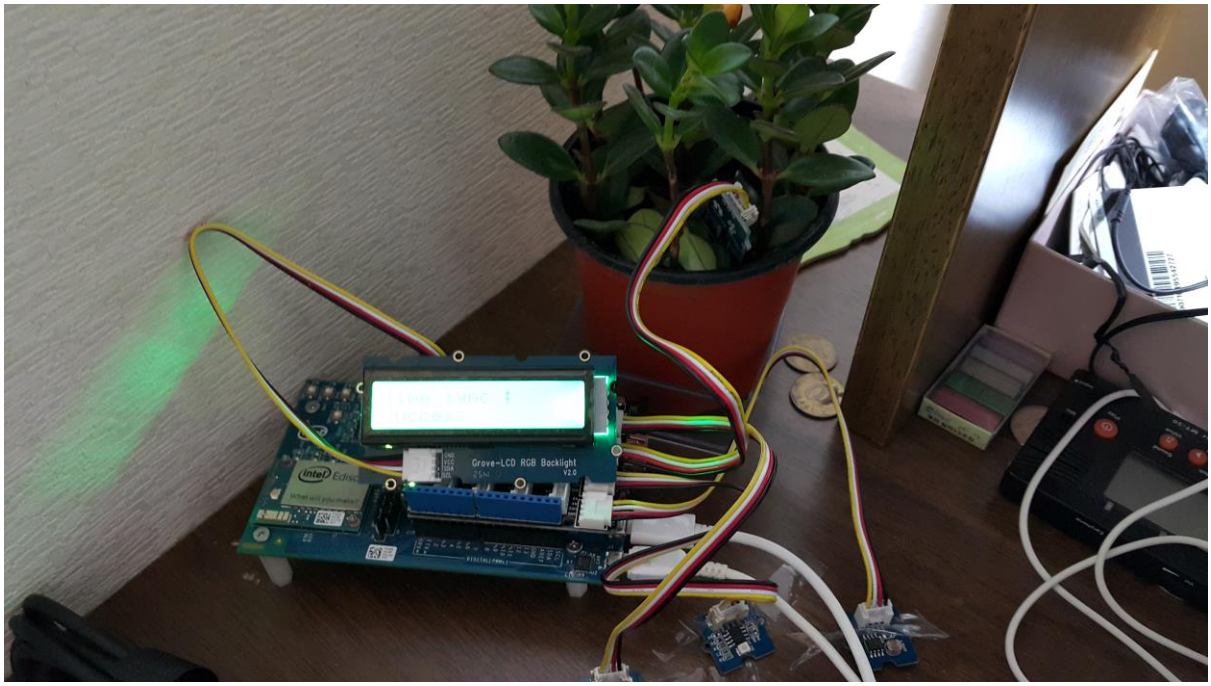
4) AWS Service – node.js 코드 복잡도 문제

AWS Lambda 함수는 모두 node.js로 작성되었습니다. 그러나 node.js는 서버 구현을 위한 비동기성을 염두에 둔 언어로, 현재까지 개발자가 다루었던 언어들과는 조금 다른 개발 패턴을 보였습니다. 이를 해결하기 위해 callback 함수를 적절히 사용하는 방법을 공부하는 등의 노력을 기울였고, 대부분의 Lambda 함수들에서 코드를 간결하게 하고 callback function이 쓰이는 부분을 최대한 적은 양으로 제한함으로써 가독성을 높이고 유지 보수가 쉽도록 변경하였습니다.

9. Results

A. Experiments

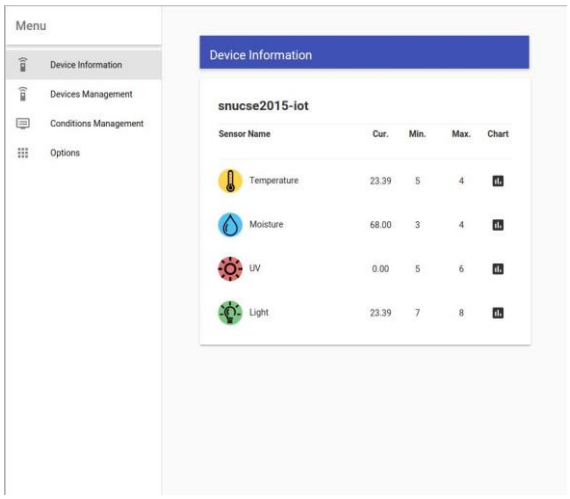
모든 센서를 장착하고 작동하고 있는 디바이스의 모습은 다음과 같습니다.

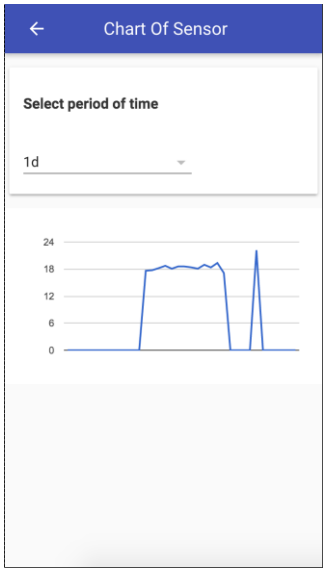


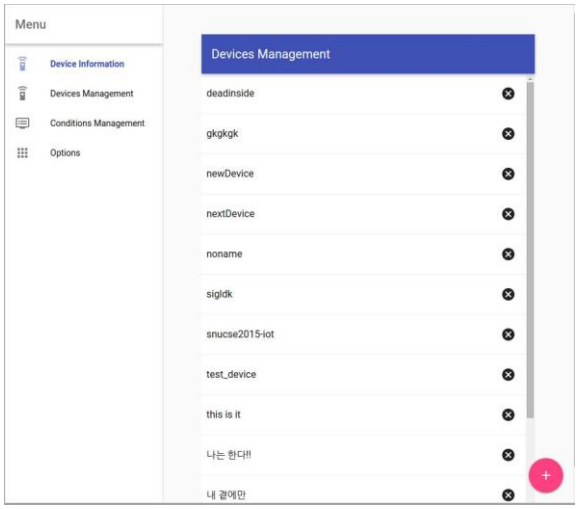
Moisture 센서는 토양의 습도를 직접 측정해야 하므로 토양에 직접 꽂혀있고, 나머지 센서는 토양 근처의 값을 측정하면 충분하므로, 주변에 배치되어 있습니다. 또한 윗부분에는 RGB LCD가 위치하여 사용자가 상태를 식별할 수 있도록 했습니다. 현재 상태는 에러가 발생하지 않은 상태이며, 이를 나타내기 위해 화면에 글씨가 쓰여있고(촬영상 문제로 글씨는 잘 보이지 않습니다.) 초록색 불빛이 나오고 있습니다.

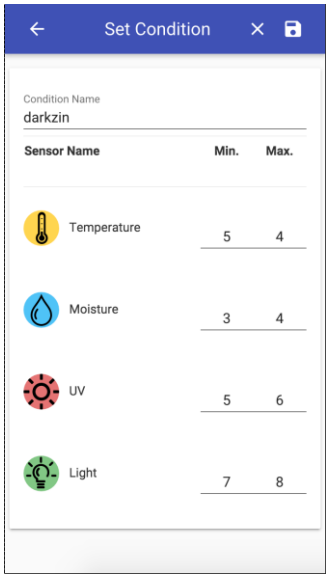
모바일 앱 캡쳐 화면은 다음과 같습니다.

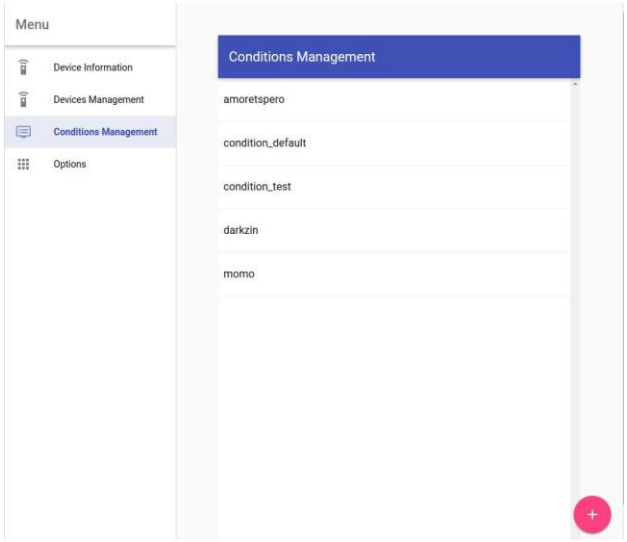
순서는 '현재 디바이스 정보' – '센서 데이터 그래프' – '디바이스 리스트' – '모니터링 조건 리스트' – '모니터링 조건 세부 정보' – '어플리케이션 옵션' 순서입니다.

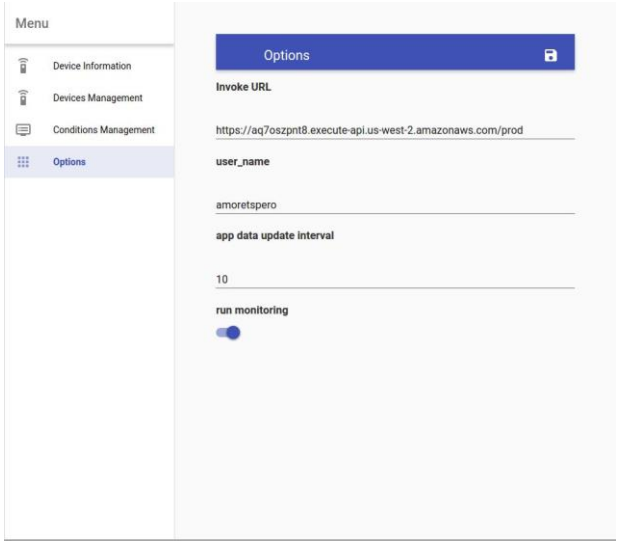







Set Condition
Condition Name
darkzin
Sensor Name
Min.
Max.
Temperature
5
4
Moisture
3
4
UV
5
6
Light
7
8


Menu
Device Information
Devices Management
Conditions Management
Options
Conditions Management
amoretspero
condition_default
condition_test
darkzin
momo


Menu
Device Information
Devices Management
Conditions Management
Options
Options
Invoke URL
https://aq7oszpnt8.execute-api.us-west-2.amazonaws.com/prod
user_name
amoretspero
app data update interval
10
run monitoring

B. Result Analysis and Discussion

위의 결과들을 통해 전체적인 시스템이 잘 작동함을 알 수 있다. 하지만 현재는 완전히 문제 상황에 대응이 가능한 상태가 아니므로 지속적인 보완이 필요할 것이다. 또한 그런 경우는 매우 드물겠지만, AWS 서비스 자체가 불능이 될 경우를 대비한 복구 코드 또한 필요할 것이다. 이를 통해 조금 더 안정적인 서비스를 구축할 수 있을 것이다. 한편, 전체 시스템에서 데이터를 처리하고 데이터를 입력/출력하고, 저장하는 부분은 모두 AWS 서비스를 통해 이루어진다. 그리고 각 파트가 서로 밀접한 연관이 있기보다는 각각의 기능을 충실히 수행하고 안정적인 AWS 서비스를 통해 최소한의 커뮤니케이션만으로 기능을 구현함으로써 decoupled된 경향이 강하고 Cloud native한 시스템을 구축하였다. 이는 더 안정적으로 작동하는 시스템을 구현하는데 큰 도움이 되었으며, 대량의 디바이스와 많은 사용자를 처리하기 수월한 시스템을 구성하는 데도 큰 도움이 되었다.

10. Division & Assignment of Work

항목	담당자
Arduino Code Development	고성빈
Communication with AWS	고성빈, 함지웅
AWS Lambda Development	함지웅
AWS Service Setup	함지웅
Communication with other parts	모두 참여
Mobile App Dev	권현석
Communication with AWS	권현석

11. Conclusion

본 프로젝트는 소규모 사용자들에게 간단하고 효율적인, 토양 품질 모니터링 시스템을 제공하는 것을 목적으로 하고 효율성과 간단함 그리고 안정성에 초점을 맞추어 진행했다. 효율성은 AWS 서비스를 적절히 활용하여 달성할 수 있었으며, 구축 비용 면에서도 AWS Lambda 함수의 수행 시간 비례 비용 청구 제도 등을 통해서 저렴하게 구현하는데 성공하였다. 또한 미리 Pre-Set 된 코드와 모바일 앱을 사용자에게 제공함으로써 사용자가 쉽게 서비스를 설정하고 사용할 수 있도록 하여 간단함의 측면을 달성할 수 있었다. 안정성 측면에서는 위에서 언급한대로 전체적인 시스템이 decoupled, Cloud native하도록 구현되어서 시스템 전체적인 안정성을 확보하였으며, 디바이스의 안정성이 더 확보된다면 더 좋은 시스템이 될 것이다. AWS 서비스는 이 프로젝트에서 처음 써보았는데, 다양한 기능들이 많은 만큼, 이후에 개발하는 다른 프로젝트에서도 이를 적절히 활용한다면 효율적으로 개발이 가능할 것이라는 생각을 할 수 있었다. 새로운 서비스와 새로운

언어 그리고 최근 부각되는 IoT 디바이스까지 최신 경향을 파악해 볼 수 있는 프로젝트였다.

◆ [Appendix] User Manual & AWS Service info

A. AWS Service Info

1. AWS Lambda function Info

Name	Handler	Role	Memory(MB)	Timeout
snucse-iot-soiltracker-fetch-data	snucse-iot-soiltracker-fetch-data.requesthandler	snucse-iot-soiltracker	256	0min 7sec
snucse-iot-soiltracker-fetch-latest-data	snucse-iot-soiltracker-fetch-latest-data.requesthandler	snucse-iot-soiltracker	128	0min 5sec
snucse-iot-soiltracker-set-monitoring-condition	snucse-iot-soiltracker-set-monitoring-condition.requesthandler	snucse-iot-soiltracker	128	0min 10sec
snucse-iot-soiltracker-set-device-condition	snucse-iot-soiltracker-set-device-condition.requesthandler	snucse-iot-soiltracker	128	0min 3sec
snucse-iot-soiltracker-set-user-option	snucse-iot-soiltracker-set-user-option.requesthandler	snucse-iot-soiltracker	128	0min 7sec
snucse-iot-soiltracker-get-user-condition	snucse-iot-soiltracker-get-user-condition	snucse-iot-soiltracker	128	0min 3sec
snucse-iot-soiltracker-get-monitoring-condition	snucse-iot-soiltracker-get-monitoring-condition.requesthandler	snucse-iot-soiltracker	128	0min 3sec
snucse-iot-soiltracker-get-device-condition	snucse-iot-soiltracker-get-device-codition	snucse-iot-soiltracker	128	0min 3sec
snucse-iot-soiltracker-add-device	snucse-iot-soiltracker-add-device.requesthandler	snucse-iot-soiltracker	128	0min 5sec
snucse-iot-soiltracker-add-user	snucse-iot-soiltracker-add-user.requesthandler	snucse-iot-soiltracker	128	0min 5sec
snucse-iot-soiltracker-get-user-device	snucse-iot-soiltracker-get-user-	snucse-iot-soiltracker	128	0min 5sec

	device.requesthandler			
snucse-iot-soiltracker-get-user-option	snucse-iot-soiltracker-get-user-option.requesthandler	snucse-iot-soiltracker	128	0min 3sec
snucse-iot-soiltracker-remove-monitoring-condition	snucse-iot-soiltracker-remove-monitoring-condition.requesthandler	snucse-iot-soiltracker	256	0min 10sec
snucse-iot-soiltracker-remove-device	snucse-iot-soiltracker-remove-device.requesthandler	snucse-iot-soiltracker	512	1min 0sec
snucse-iot-soiltracker-process-data	snucse-iot-soiltracker-process-data.datahandler	snucse-iot-soiltracker	256	0min 5sec

2. AWS DynamoDB Table Info

Name	Primary Key(type)	Sort Key(type)	Read capacity	Write capacity
snucse-iot-soiltracker-device-info	device_id(String)	X	1	1
snucse-iot-soiltracker-monitoring-data	device_id(String)	X	2	2
snucse-iot-soiltracker-monitoring-info	user_name(String)	condition_name(String)	1	1
snucse-iot-soiltracker-sensor-data	device_id(String)	time(Number)	5	5
snucse-iot-soiltracker-sensor-data-latest	device_id(String)	X	2	2
snucse-iot-soiltracker-user-device-info	user_name(String)	device_id(String)	1	1
snucse-iot-soiltracker-user-info	user_name(String)	X	1	1

B. User Manual (Next Page)

AWS IoT Backend Soil Tracker System

User Manual

Version : 1.0 (Final)

Date : 2015-12-18

Written by Jiung Hahm

Contact : amoretspero@gmail.com

Contents

1. Introduction to Soil Tracker System

* 본 시스템은 2015년 가을학기 서울대학교 창의적 통합설계1 과목의 프로젝트의 하나로 기획된 것이며, 개발 기간은 2015년 9월부터 2015년 12월 입니다.

1.1. Background

최근 컴퓨팅 기기의 소형화에 힘입어 사물인터넷(IoT = Internet of Things) 기술이 발달하고 있습니다. 신용카드 만한 크기의 디바이스에서 다양한 작업을 할 수 있게 되었고, 클라우드 서비스의 발달에 힘입어 인터넷 연결을 통해 더욱 더 많은 유용한 서비스를 만들 수 있게 되었습니다. 본 프로젝트 팀은 이런 IoT 환경과 클라우드 서비스인 AWS(Amazon Web Services)를 이용하여 식물 생장에 필요한 토양 환경을 모니터링 할 수 있는 시스템을 제작하였습니다.

사용자는 이 시스템을 활용하여 원격으로 토양 환경을 관리할 수 있으며, 문제가 발생한 경우 알림을 받을 수 있습니다. 함께 제공되는 모바일 앱은 편의성을 더욱 높여줄 것입니다.

1.2. Overall Architecture

본 시스템의 전체 구조는 크게 3부분으로 나눠지며 다음과 같습니다.

Intel Edison + Arduino Board / AWS Services / Mobile App

Intel Edison + Arduino Board : 이 부분은 토양 환경 정보를 수집하는 역할을 담당합니다. 환경 정보를 수집한 후, 연결된 무선 인터넷을 통해 정보를 서버로 전송합니다.

AWS Services : 이 부분은 서버의 역할을 담당합니다. 디바이스로부터 수집된 정보를 처리하여 DB에 저장하고, 모바일 앱에서 요청하는 데이터를 제공하며, 알림, API 관리 등의 역할을 수행합니다.

Mobile App : 이 부분은 사용자와 가장 가까이 있는 부분으로, 인터넷이 있는 어디에서나 사용자가 원격으로 편리하게 토양 환경을 관리할 수 있도록 도와줍니다. 디바이스 정보, 현재 상태, 데이터 그래프 등 다양한 기능을 제공합니다.

전체적인 구조를 다이어그램을 나타내면 Fig.1과 같습니다.

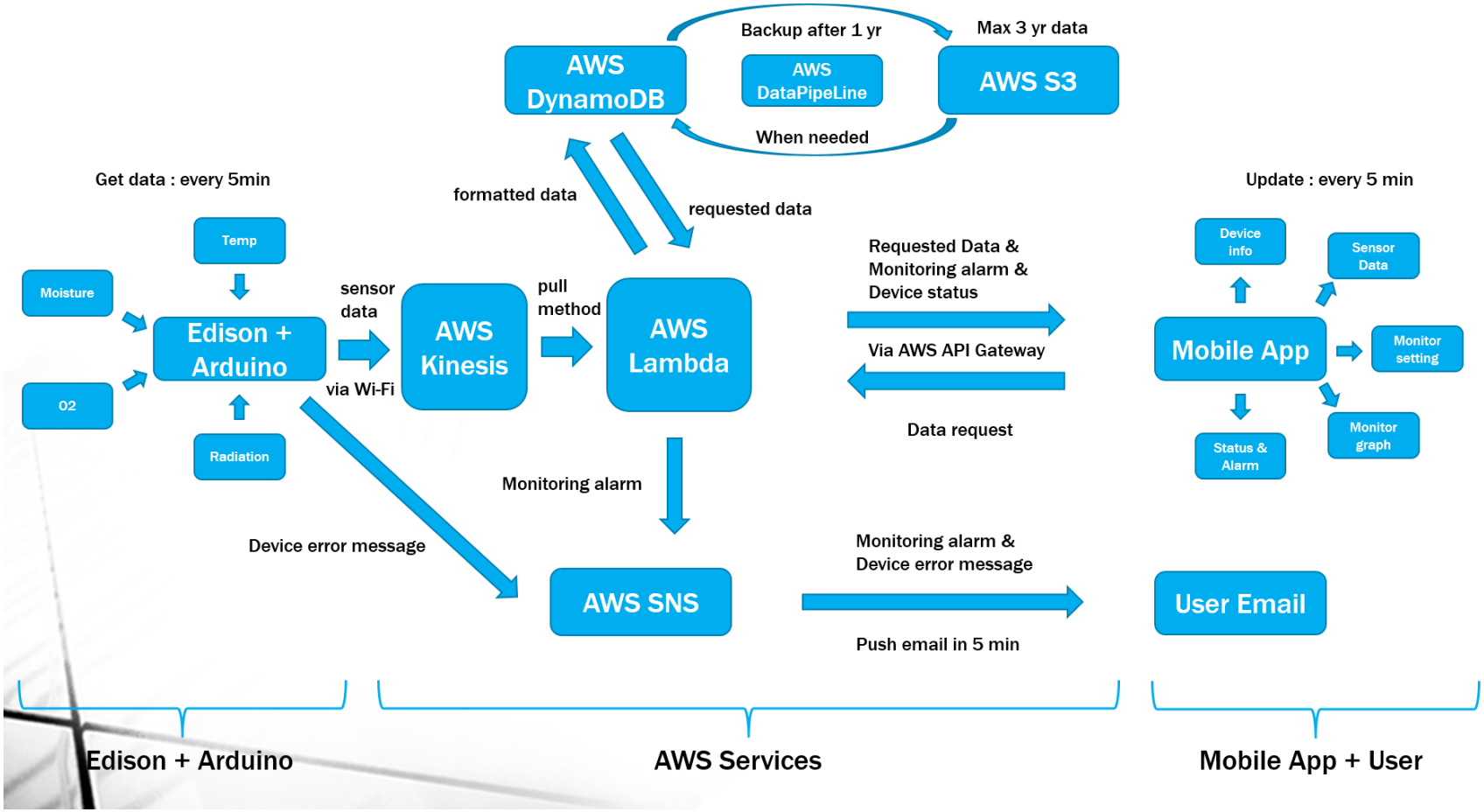

AWS DynamoDB
Backup after 1 yr
AWS DataPipeLine
Max 3 yr data
AWS S3
When needed
Get data : every 5min
formatted data
requested data
Update : every 5 min
Temp
Moisture
Edison + Arduino
O2
Radiation
sensor data
via Wi-Fi
AWS Kinesis
pull method
AWS Lambda
Requested Data & Monitoring alarm & Device status
Via AWS API Gateway
Data request
Device info
Sensor Data
Mobile App
Monitor setting
Monitor graph
Status & Alarm
Monitoring alarm
Device error message
AWS SNS
Monitoring alarm & Device error message
Push email in 5 min
User Email
Edison + Arduino
AWS Services
Mobile App + User

fig.1 – Overall Architecture

2. Intel Edison + Arduino Board System

2.1. Intel Edison Setup

이 파트에서는 Intel Edison 디바이스를 설정하는 방법을 설명합니다.

1) Intel Edison Board를Arduino Board에 결합합니다. 결합 후에 J9 Jumper를 아래쪽 위치로 변경합니다. '아래쪽'의 기준은 Arduino Board를 정방향으로 두었을 때가 기준입니다. 그 후에 동봉된 Micro USB Cable을 두 개 모두 연결하고 컴퓨터에 Intel Edison을 연결합니다.

자세한 정보는 https://software.intel.com/en-us/assembling-intel-edison-board-with-arduino-expansion-board 를 참고하시기 바랍니다.

2) Intel Edison에 운영체제를 설치합니다. 링크(https://software.intel.com/en-us/iot/hardware/edison/downloads)에서 "ww18-15" 버전의 Yocto Linux 이미지와 Image Flash Tool을 다운로드합니다. Windows 사용자의 경우 64-bit Installer를 사용하는 것을 권장합니다. 한편, Image Flash Tool의 이름이 'Phone Flash Tool Lite'인 경우, 정상이니 사용하면 됩니다.

3) Flash Tool을 실행합니다. 그리고 Intel Edison이 디바이스 리스트에 나열되어 있는지 확인합니다. 그 후에 다음 과정을 따릅니다.

1. 2번에서 다운로드 받은 Yocto Linux 이미지를 선택합니다.

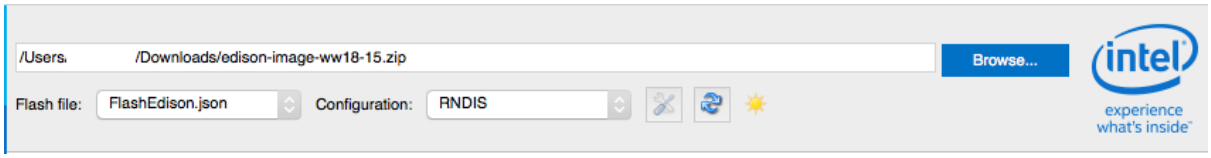


Fig.2 – Selecting Yocto Linux Image.

2. "Start to Flash" 버튼을 클릭합니다. 만약 "disconnect and reconnect your Edison" 메시지가 나오면 USB 케이블을 연결 해제 한 후에 다시 연결합니다.

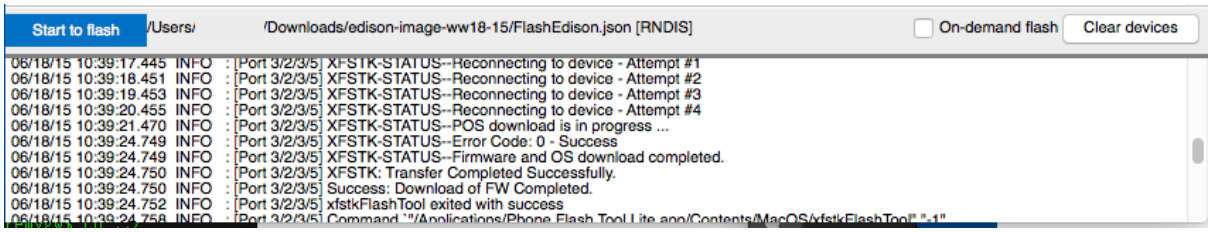


Fig.3 – Flashing Linux Image.

3. Image Flashing이 끝날 때까지 기다립니다. 완료되면 Intel Edison을 재부팅합니다.

4) Intel Edison과 컴퓨터를 연결합니다.

1. Mac의 경우 : 터미널 윈도우를 열고 "screen /dev/tty.usbs"를 입력합니다. 명령어 뒤에 "115200 –L"을 추가로 입력하고 Enter 키를 두 번 누릅니다.

2. Windows의 경우 : PuTTy를 설치한 다음에 "serial"을 선택한 후, baud rate는 115200으로 설정합니다. 장치 관리자에서 COM port 중에서 "Edison Virtual Com Port"가 아닌 "USB" 레이블을 갖고 있는 포트를 선택하여 접속합니다.

3. Linux의 경우 : screen 커맨드를 사용해 접속하기 전에, screen을 설치하고, "screen /dev/ttyUSB0 115200" 커맨드를 사용해 접속합니다.

5) "configure_edison –setup" 커맨드를 사용해 Intel Edison을 설정합니다.

이제 Intel Edison 설정이 완료되었습니다.

2.2. Arduino Board Setup

이 파트에서는 Arduino Board를 설정하는 방법에 대해 설명합니다.

1. Light, Moisture, Temperature, UV 센서와 LCD 디스플레이, 연결 선을 준비합니다.

2. Grove Base Shield를 Arduino Expansion Board에 연결합니다.

3. Light 센서는 A3포트에, Moisture 센서는 A1포트에, Temperature 센서는 A0포트에, UV 센서는 A2포트에, LCD 디스플레이는 I2C포트에 연결합니다.

4. 전원을 연결하고 제대로 LED가 점등되는지 확인합니다.

5. https://github.com/cloudtrack/lustitia 에 접속하여 "Download Zip" 버튼을 클릭해 코드 데이터를 받고, 원하는 곳에 압축을 해제합니다.

6. https://www.arduino.cc/en/Main/Software 에 접속하여 Windows Installer를 다운받고, 설치합니다.

7. 5번에서 압축을 해제한 폴더 안의 "Edison/SoilTracker" 폴더에 들어가 "SoilTracker.ino" 파

일을 실행합니다. 6번에서 설치한 Arduino IDE가 열립니다.

8. 라이브러리 설정을 진행합니다. "Sketch" 메뉴에서 "Include Library"로 들어가 "Add .zip Library"를 선택합니다. 그리고 5번에서 압축을 해제한 폴더 내부의 "Edison/AWSArduinoLibrary" 폴더를 선택하고 "Open" 버튼을 클릭합니다. 동일 폴더 내부의 "AWSEdisonLibrary" 폴더, "HardwareLibrary" 폴더, "Time" 폴더에 대해서도 같은 작업을 해줍니다.

9. 이제 "Verify" 버튼을 누르고 제대로 컴파일이 되는지 확인합니다.

10. 제대로 컴파일이 되었다면 AWS 계정을 만들기 전까지의 모든 작업은 완료되었습니다. AWS 계정을 설정한 후에 사용자 정보를 입력하고 디바이스에 프로그램을 업로드 합니다. 만약 제대로 컴파일 되지 않았다면 GitHub의 https://github.com/cloudtrack/lustitia의 Issue Tracker에 질문을 등록해주세요.

이제 Arduino Board 설정이 완료되었습니다.

2.3. Where to get the devices

본 시스템을 사용하기 위해서는 Intel Edison Board와 Arduino Expansion Board가 필요합니다.

국내에서 "Intel Edison Kit for Arduino Package"를 구매하면 Intel Edison과 Arduino Expansion Board를 얻을 수 있습니다.

2.4. Where to get the sensors

본 시스템을 사용하기 위해서는 Grove sensor들이 필요합니다. 국내에서 "Grove Starter Kit for Arduino" 제품과 "Grove UV Sensor"를 구매하면 됩니다. 만약 국내에서 구매가 불가능하다면 제조사인 SeeedStudio의 홈페이지http://www.seeedstudio.com/depot 에서 구매할 수 있습니다.

이로써 Intel Edison + Arduino Board System 설정을 마칩니다.

3. AWS System

3.1. AWS Account Setup

https://aws.amazon.com 에 접속하여 AWS에 가입합니다. AWS에서는 Free tier라고 하여 첫 가입 후 12개월동안 다양한 서비스를 무료로 제공합니다. 본 시스템은 이러한 무료 제공 자원을 적극적으로 활용하며, AWS Kinesis 이외에는 과금이 되지 않을 것입니다. 가입 후에 다시 위의 링크로 들어가서 로그인을 진행하면 AWS Console에 들어갈 수 있습니다.

3.2. AWS IAM Setup

이제 "Identity & Access Management" 서비스로 들어갑니다. 이 곳에서는 AWS의 계정 및 서비스의 액세스 권한과 사용자 인증과 관련된 것들을 설정할 수 있습니다.

1) Group 설정

Group은 특정 권한을 공통적으로 가진 사용자들의 집합이라고 볼 수 있습니다. 본 설명서에서는 하나의 그룹을 생성하여 그 그룹에 사용자를 추가합니다.

Groups 보드에서 "Create New Group" 버튼을 클릭합니다. Group Name에 "SNUCSE-IOT-SOILTRACKER"를 입력하고, "Next Step" 버튼을 클릭합니다. 이제 이 그룹에 속한 사용자들이 가질 수 있는 권한을 설정하는 화면이 나옵니다. "AWSLambdaFullAccess", "AmazonS3FullAccess", "AmazonDynamoDBFullAccess", "AmazonKinesisFullAccess", "AWSLambdaInvocation-DynamoDB", "AmazonSNSFullAccess", "AWSDataPipelineFullAccess"를 검색하여 추가하고, "Next Step" 버튼을 클릭합니다. 설정을 확인하고, "Create Group" 버튼을 눌러 완성합니다.

2) User 설정

User는 그룹에 속한 개개인의 사용자입니다. 그룹이 가진 권한을 그대로 상속받으며, 자신만의 권한을 가질 수도 있습니다. 본 설명서에서는 하나의 사용자를 생성하여 위에서 만든 group에 추가합니다.

Users 보드에서 "Create New Users" 버튼을 클릭합니다. 1번 칸에 원하는 이름을 입력합니다. "Generate an access key for each user" 버튼은 체크된 상태로 유지하고, "Create" 버튼을 눌러 완성합니다.

이제 다시 Users 보드로 돌아와 방금 만든 사용자의 이름을 클릭하면 이 사용자의 요약 정보를 보여주는 화면이 나옵니다. 이 화면에서 "Permissions" 탭으로 들어가 "Attach Policy" 버튼을 클릭합니다. "AmazonAPIGatewayInvokeFullAccess", "AmazonAPIGatewayAdministrator"를 검색하여 추가합니다.

이제 "Security Credentials" 탭으로 들어갑니다. "Create Access Key"를 클릭하여 새 Access Key를 발급받습니다. 이 Access Key는 해당 사용자가 AWS의 여러 서비스들을 이용할 때 사용자가 인증된 사용자임을 증명하는데 사용됩니다. 만들 때, Access Key와 Secret Access Key는 반드시 별도의 공간에 메모해 둡니다. Secret Access Key는 분실하면 다시 볼 수 없으므로 무조건 재발급 받아야 합니다.

3) Role 설정

Role은 AWS Service들이 가질 수 있는 권한을 의미합니다. 왼쪽의 "Roles" 메뉴로 들어가서 "Create New Role" 버튼을 클릭하여 새 Role을 생성합니다.

첫 단계에서는 이름을 지정합니다. "snucse-iot-soiltracker"를 입력하고 "Next Step"을 클릭합니다.

두 번째 단계에서는 Role의 Type을 지정합니다. 본 시스템에서는 AWS Lambda function에 주로 할당하게 될 것이므로, "AWS Service Roles" 의 "AWS Lambda"를 "Select"하여 다음 단계로 이동합니다.

세 번째 단계에서는 Role이 가질 수 있는 권한을 설정합니다. "AWSLambdaFullAccess", "AmazonS3FullAccess", "AmazonDynamoDBFullAccess", "AWSLambdaDynamoDBExecutionRole", "AmazonKinesisFullAccess", "AmazonSNSFullAccess", "AWSDataPipelineFullAccess"를 선택하여 추가합니다. "Next Step"을 클릭해 마지막 단계로 이동합니다.

마지막 단계에서는 위에서 설정한 것들이 제대로 되어있는지 확인한 후, "Create Role" 버튼을 클릭하여 생성을 완료합니다.

이제 AWS IAM 설정이 완료되었습니다. 위에서 만든 사용자의 이름과 Access Key, Secret Access Key는 매우 중요한 정보이므로 반드시 별도의 공간에 저장해두고 잃어버리지 않도록 조심합니다.

3.3. AWS Kinesis Setup

이 부분에서는 AWS Kinesis 서비스를 설정합니다.

1) AWS Management Console에서 Kinesis 서비스로 접속합니다.

2) 상단의 "Create Stream" 버튼을 클릭하여 새 Stream을 생성합니다. "Stream Name"에는 "snucse-iot-soiltracker"를, "Number of Shards"에는 1을 넣어줍니다. Shard는 한 번에 Kinesis가 처리할 수 있는 데이터의 양을 정해줍니다. 이 서비스에서는 1개로 충분합니다. 만약 많은 디바이스를 관리한다면 더 높은 숫자를 설정해주면 됩니다.

이제 AWS Kinesis 설정이 완료되었습니다.

3.4. AWS DynamoDB Setup

이 부분에서는 AWS DynamoDB 서비스를 설정합니다.

1) AWS Management Console에서 DynamoDB 서비스로 접속합니다.

2) 상단의 "Create Table" 버튼을 클릭하여 새 데이터베이스 테이블을 생성합니다. 각 테이블의 이름과 생성 정보는 아래와 같습니다.

Name	Primary Key(type)	Sort Key(type)	Read capacity	Write capacity
snucse-iot-soiltracker-device-info	device_id(String)	X	1	1
snucse-iot-soiltracker-monitoring-data	device_id(String)	X	2	2
snucse-iot-soiltracker-monitoring-info	user_name(String)	condition_name(String)	1	1
snucse-iot-soiltracker-sensor-data	device_id(String)	time(Number)	5	5
snucse-iot-soiltracker-sensor-data-latest	device_id(String)	X	2	2
snucse-iot-soiltracker-user-device-info	user_name(String)	device_id(String)	1	1
snucse-iot-soiltracker-user-info	user_name(String)	X	1	1

"Create table" 메뉴에서 Sort Key와 Read capacity, Write capacity를 설정하기 위해서는 "Add Sort Key" 체크박스에 체크를 하고, "Use default settings" 체크박스에 체크 해제하면 됩니다.

3) 이제 위에서 생성한 7개의 테이블 중에서 "snucse-iot-soiltracker-user-info" 테이블을 제외한 6개의 테이블에 대해서, Stream을 Enable합니다. 테이블 이름을 클릭한 후, Overview 메뉴에서 "Manage Stream"을 클릭하여 "New and old images" 옵션을 선택한 후 "Enable"을 클릭하여 Stream을 Enable합니다.

이제 AWS DynamoDB 설정이 완료되었습니다.

3.5. AWS SNS Setup

이 부분에서는 AWS SNS 서비스를 설정합니다.

1) AWS Management Console에서 SNS 서비스로 접속합니다.

2) 왼쪽 메뉴의 "Topics"로 들어갑니다.

3) "Create new topic" 버튼을 클릭하여 Topic을 생성합니다. "snucse-iot-soiltracker-device-direct-alarm"과 "snucse-iot-soiltracker-monitoring-alarm" 의 2개의 Topic을 생성합니다. Display name은 사용하지 않으므로 빈 칸으로 유지합니다. "Create topic" 버튼을 눌러 Topic을 생성합니다.

4) 이제 알림을 받을 Email을 설정합니다. 위의 3번 과정에서 만든 Topic을 클릭하고 "Actions" 에서 "Subscribe to topic"을 클릭합니다. "Protocol"은 "Email"로 설정하고, 알림을 받을 Email을 "Endpoint" 칸에 입력합니다. "Create Subscription" 버튼을 눌러 완료합니다. 3번 과정에서 만든 2개의 Topic 모두에 대해 설정해줍니다.

5) 4번 과정에서 사용한 Email 계정으로 접속하여 Subscription 메일을 확인하고 Confirm합니다.

6) 왼쪽 "Subscription" 메뉴로 들어가 Subscription이 제대로 되었는지 확인합니다.

이제 AWS SNS 설정이 완료되었습니다.

3.6. AWS Lambda Setup

이 부분에서는 AWS Lambda 서비스를 설정합니다.

1) AWS Management Console에서 AWS Lambda 서비스로 접속합니다.

2) "Create a Lambda function" 버튼을 클릭하여 Lambda function을 생성합니다.

3) Step 1은 "Select blueprint"이며, 이 과정은 생략합니다. 바로 "Skip"을 클릭해 다음 단계로 이동합니다.

4) Step 2는 "Configure function" 입니다. AWS Lambda Setup에서는 모두 15개의 Lambda function을 만들게 됩니다. 각 function의 Source code는 이 문서의 **3.7. Where to get the source code** 에서 얻을 수 있습니다.

각 함수의 Source Code는 ZIP 파일로 제공되며, 다음 항목에서 다루게 됩니다.

모든 Lambda function의 Runtime language는 Node.js로 동일합니다.

나머지 옵션의 값은 다음 표와 같습니다.

Name	Handler	Role	Memory(MB)	Timeout
snucse-iot-soiltracker-fetch-data	snucse-iot-soiltracker-fetch-data.requesthandler	snucse-iot-soiltracker	256	0min 7sec
snucse-iot-soiltracker-fetch-latest-data	snucse-iot-soiltracker-fetch-latest-data.requesthandler	snucse-iot-soiltracker	128	0min 5sec
snucse-iot-soiltracker-set-monitoring-condition	snucse-iot-soiltracker-set-monitoring-condition.requesthandler	snucse-iot-soiltracker	128	0min 10sec
snucse-iot-soiltracker-set-device-condition	snucse-iot-soiltracker-set-device-condition.requesthandler	snucse-iot-soiltracker	128	0min 3sec
snucse-iot-soiltracker-set-user-option	snucse-iot-soiltracker-set-user-option.requesthandler	snucse-iot-soiltracker	128	0min 7sec
snucse-iot-soiltracker-get-user-condition	snucse-iot-soiltracker-get-user-condition	snucse-iot-soiltracker	128	0min 3sec
snucse-iot-soiltracker-get-monitoring-condition	snucse-iot-soiltracker-get-monitoring-condition.requesthandler	snucse-iot-soiltracker	128	0min 3sec
snucse-iot-soiltracker-get-device-condition	snucse-iot-soiltracker-get-device-codition	snucse-iot-soiltracker	128	0min 3sec
snucse-iot-soiltracker-add-device	snucse-iot-soiltracker-add-	snucse-iot-soiltracker	128	0min 5sec

	device.requesthandler			
snucse-iot-soiltracker-add-user	snucse-iot-soiltracker-add-user.requesthandler	snucse-iot-soiltracker	128	0min 5sec
snucse-iot-soiltracker-get-user-device	snucse-iot-soiltracker-get-user-device.requesthandler	snucse-iot-soiltracker	128	0min 5sec
snucse-iot-soiltracker-get-user-option	snucse-iot-soiltracker-get-user-option.requesthandler	snucse-iot-soiltracker	128	0min 3sec
snucse-iot-soiltracker-remove-monitoring-condition	snucse-iot-soiltracker-remove-monitoring-condition.requesthandler	snucse-iot-soiltracker	256	0min 10sec
snucse-iot-soiltracker-remove-device	snucse-iot-soiltracker-remove-device.requesthandler	snucse-iot-soiltracker	512	1min 0sec
snucse-iot-soiltracker-process-data	snucse-iot-soiltracker-process-data.datahandler	snucse-iot-soiltracker	256	0min 5sec

5) 이제 각 Lambda function에 대해 필요한 경우 Event source mapping을 설정합니다.

여기에서는 "snucse-iot-soiltracker-process-data" 함수에 대해서 Event source mapping을 설정합니다. "snucse-iot-soiltracker-process-data" 함수 이름을 클릭하여 해당 함수 화면으로 진입합니다. "Event Sources" 탭에서 "Add event source" 버튼을 클릭합니다. "Event source type"으로 "Kinesis"를 클릭하면 위에서 만든 Kinesis stream인 "snucse-iot-soiltracker"를 "Kinesis stream" 항목에서 찾을 수 있습니다. 이 항목으로 설정한 후, 나머지 설정은 기본값(Batch size : 100, Starting position : Trim horizon, "Enable event source": "Enable now")으로 유지합니다. "Submit" 버튼을 눌러서 마무리합니다.

이제 AWS Lambda 설정이 완료되었습니다.

3.7. AWS API Gateway Setup

이 부분에서는 AWS API Gateway를 설정합니다.

1) AWS Management Console에서 API Gateway로 접속합니다.

2) 상단의 "Create API"버튼을 클릭하여 API를 생성합니다. 이름은 "snucse-iot-soiltracker"로 설정하고, "Clone from API" 항목은 "Do not clone from existing API"를 유지합니다. 하단의 "Create API" 버튼을 클릭하여 완료합니다.

3) 만들어진 API의 이름을 클릭하여 해당 API의 메뉴로 진입합니다.

4) 이제 Resource를 생성합니다. 각 resource마다 생성하는 방법은 같고, 생성해야 하는 resource의 이름은 다음과 같습니다. "set-device-condition", "add-device", "add-user", "get-device-condition", "set-monitoring-condition", "set-user-option", "get-monitoring-condition", "get-user-condition", "get-user-device", "get-user-option", "fetch-latest-data", "fetch-data", "remove-device", "remove-monitoring-condition".

5) 이제 각 resource를 생성하는 방법에 대해 설명합니다. "set-device-condition" resource를 대표로 설명합니다. 나머지 resource들에 대해서도 같은 방법으로 설정하면 됩니다.

6) "/"(root)에서 "Create Resource"를 클릭합니다. 이때 "/"에는 method가 없는 상태입니다. "No methods defined for the resource"를 화면에서 확인할 수 있습니다. "Create Resource" 버튼을 클릭한 다음 resource 이름을 "Resource Name" 항목에 넣습니다. Resource Path는 자동으로 생성되므로 따로 입력할 필요는 없습니다. 이제 "Create Resource" 버튼을 눌러 생성을 마무리합니다.

7) 정상적으로 생성되었다면, 왼쪽 메뉴에서 "/" 밑의 "/(Resource 이름)" 항목을 확인할 수 있습니다. 이제 이 항목을 클릭하여 들어갑니다.

8) 오른쪽 상단의 "Create Method"를 클릭합니다. 그러면 왼쪽 메뉴의 (Resource 이름) 밑에 선택 가능한 칸이 나타납니다. "OPTIONS"을 클릭하고 바로 옆의 체크 표시를 클릭합니다.

9) 바뀐 화면에서 "Integration type"은 "Mock Integration"을 체크하고, "Save" 버튼을 클릭하면 OPTIONS method 생성이 완료됩니다. 이제 해당 method의 개략적인 작동이 그려진 화면이 나타납니다. "Integration Request" 항목을 클릭하여 들어갑니다. "Mapping Templates"를 클릭하여 "application/json"이 있는지 확인합니다. 있으면 그대로 유지하고, 없다면 "Add mapping template" 버튼을 클릭하여 새로 만들어줍니다.

10) 이제 다시 "/(Resource 이름)"을 클릭하여 해당 항목의 메뉴로 진입합니다. 화면에서 방금 만든 "OPTIONS" method의 정보를 확인할 수 있습니다. "Create Method"를 클릭하고 왼쪽 메뉴의 (Resource 이름) 밑에 선택 가능한 칸이 나타나면 "POST"를 클릭하고 바로 옆의 체크 표시를 클

릭합니다.

11) 바뀐 화면에서 "Integration type"은 "Lambda Function"을 클릭하고 아래쪽에 Region은 "us-west-2"를 선택합니다. "Lambda Function"은 "set-device-condition"으로 설정합니다. 자동완성이 되므로 편리하게 입력할 수 있습니다. 다 되었으면 "Save"를 클릭합니다.

12) 이제 다시 "/(Resource 이름)"을 클릭하여 resource의 화면으로 돌아갑니다. 우측 상단에 있는 "Enable CORS" 버튼을 누르고, 조건은 아무것도 건드리지 않은 채로 "Enable CORS and replace existing CORS headers" 버튼을 클릭합니다. 팝업 메시지가 뜨면 "Yes, replace existing values"를 클릭하여 설정을 완료합니다.

13) 4번에서 언급한, 생성해야 하는 다른 resource들에 대해서 6~12를 반복합니다.

14) 다 되었으면 "/"로 이동합니다. 왼쪽 상단에 "Deploy API"를 클릭하여 API를 배포합니다. "Deployment stage"는 "prod"로, "Deployment description"은 빈 상태로 유지하고 "Deploy" 버튼을 눌러 배포합니다.

15) Deploy가 완료되면 새로운 페이지로 이동하며, 상단에 "Invoke URL"이 표시됩니다. 이 URL을 잘 적아 별도의 장소에 보관합니다. 이후 모바일 앱 사용시 필요한 정보입니다.

이제 AWS API Gateway 설정이 완료되었습니다.

3.9. Arduino AWS Info Setup

이 부분에서는 Arduino Board에 AWS 사용자 정보를 입력하는 과정에 대해 설명합니다.

위의 Arduino Board Setup 과정에서는 AWS 사용자 정보를 입력하지 않았지만, Arduino Board가 AWS 서비스와 통신하기 위해서는 정보 입력이 필수적입니다.

1) 2.2.단계에서 다운로드 받은 Arduino Board 파일을 저장한 폴더로 이동하여 "Soiltracker" 폴더로 이동합니다.

2) "keys.cpp" 파일을 엽니다.

3) "awsKeyID" 변수를 3.2.단계에서 얻은 사용자의 access key로 설정합니다.

4) "awsSecKey" 변수를 3.2.단계에서 얻은 사용자의 secret access key로 설정합니다.

5) "ssid" 변수를 디바이스가 작동할 환경에서 사용하는 Wi-Fi 네트워크의 이름으로 설정합니다.

6) "pass" 변수를 디바이스가 작동할 환경에서 사용하는 Wi-Fi 네트워크의 비밀번호로 설정합니다.

7) "TARGET_ARN" 변수를 3.5.에서 설정한 SNS 서비스의 "snucse-iot-soiltracker-device-direct-alarm" 토픽의 ARN으로 설정합니다. ARN은 AWS SNS 메인 화면에서 왼쪽의 "Topics" 메뉴로 이동하면 확인할 수 있습니다. 이때, 변수를 설정하는 과정에서 ":"을 모두 "%3A"로 변경하는 과정이 필요합니다.

8) "HASH_KEY_VALUE"의 값을 원하는 이름으로 설정합니다. 영어와 숫자, 기호만 가능하며 이 이름으로 정보가 DB에 기록됩니다.

9) 이제 이 정보를 저장합니다.

10) 2.2.에서 설정한 Arduino IDE를 다시 실행합니다. "SoilTracker.ino" 파일을 더블클릭하면 자동으로 실행됩니다.

11) 다시 체크 버튼을 눌러 컴파일하고 에러가 없는지 확인합니다.

12) Intel Edison + Arduino Board를 조립한 시스템을 같이 제공된 USB 케이블 2개로 컴퓨터에 연결합니다.

13) 장치관리자에 들어가서 "Edison Virtual COM Port"로 이름이 붙은 COM 포트 번호를 확인합니다.

14) 이제 "Tools"의 "Port" 메뉴로 들어가서 13에서 확인한 포트 번호를 찾아 클릭하여 설정하고, 오른쪽 화살표(->) 버튼을 클릭하여 디바이스에 업로드합니다.

이것으로 Arduino AWS Info 설정이 완료되었습니다.

3.9. Where to get the source code

AWS service 설정 과정에서 AWS Lambda 서비스의 경우 각 Lambda function들의 코드가 필요합니다. 해당 코드는 Github https://github.com/cloudtrack/lustitia에서 받으실 수 있습니다.

받는 방법 :

1) 위의 링크에 접속합니다.

2) 우측 상단의 "Download ZIP" 버튼을 클릭하여 파일을 저장합니다.

3) 압축을 풀면, "AWSLambdaFunctionCodes" 폴더에 각 Lambda function들의 코드가 .zip 파일로 저장되어 있습니다. 해당 코드를 이름에 맞게 사용하시면 됩니다.

이로써 AWS System 설정을 마칩니다.

4. Mobile App System

4.1. Download & Install

이 부분에서는 Mobile App을 다운로드하고, 기기에 설치하는 과정에 대해 설명합니다.

Github https://github.com/cloudtrack/lustitia에서 soiltracker-app.apk를 다운로드받아 설치합니다. 설치 후 앱을 실행시키시면 username과 AWS lambda URL을 입력하는 양식을 보실 수 있습니다. 여기에 제공받은 username과 URL을 입력하시면, username에 할당되어 있는 device들 중 데이터가 존재하는 첫번째 디바이스의 페이지로 이동합니다.

4.2. Main Page

이 부분에서는 Mobile App의 디바이스 페이지에 대해 설명합니다.

Device Page에서는 현재 선택된 Device의 최근 센서 데이터와, 각 센서 데이터의 제한조건을 보실 수 있습니다. 제한 조건을 위반 시에는 옵션에 따라 사용자에게 알림이 전달됩니다. Device를 바꾸고 싶으시면 Drawer Menu에서 Devices Management로 이동한 뒤, 이용가능한 Device 중 원하는 Device를 선택하면 됩니다. 각 센서 정보 옆에 그래프 버튼을 선택하면 현재 Device의 해당 센서값을 그래프 차트로 보여주는 Data Chart Page로 이동합니다.

4.3. Data Chart

이 부분에서는 Mobile App의 데이터 차트에 대해 설명합니다.

데이터 차트는 사용자가 원하는 기간동안 선택된 Device의 센서 데이터를 그래프로 보여주는 페이지입니다. 화면 상단의 드롭다운 메뉴를 누르시면 원하는 기간을 선택할 수 있습니다. 기간을 선택하면 화면 하단의 라인 차트에 현재부터 원하는 기간 전까지의 센서 데이터가 표시됩니다. 선택 가능한 기간은 1일, 3일, 6일, 일주일, 1달, 3달, 6달입니다. 헤더 패널의 좌측 화살표 아이콘을 누르시면 Device Page로 이동합니다.

4.4. Monitoring Condition

이 부분에서는 Mobile App에서 모니터링 조건을 조회하고, 설정하는 방법에 대해 설명합니다.

Condition Management Page는 현재 사용자가 갖고있는 조건들을 열람하고, 수정 및 추가하는 페이지입니다. 또 사용자는 현재 디바이스에 적용하고 싶은 조건을 선택하여 디바이스 센서의 제한 조건을 바꿀 수 있습니다.

목록 내에서 조건을 선택하면 해당조건의 내용이 화면에 표시됩니다. 화면 상단의 헤더패널 중 V 표시된 버튼을 누르면 현재 열람한 조건이 현재 디바이스에 적용되고, Device Page로 이동합니다. 그리고 옆의 연필모양의 아이콘을 누르면 현재 조건을 편집할 수 있습니다.

편집화면에서는 각 센서의 최소조건과 최대조건, 조건의 이름을 변경할 수 있습니다. 화면 상단의 헤더패널 중 X 모양 아이콘을 누르면 현재 조건이 삭제되고 Condition Management Page로 이동합니다. 그리고 디스크 모양의 아이콘을 누르면 현재 조건이 저장되어 서버에 반영되고, 현재 조건이 추가된 Condition Management Page로 이동합니다.

4.5. Device Management

이 부분에서는 Mobile App에서 디바이스를 관리하는 방법에 대해 설명합니다.

Devices Management Page에서는 사용자에게 등록된 모든 디바이스가 표시됩니다. 사용자는 목록 내 항목을 선택하여 원하는 Device의 정보를 볼 수 있습니다. 그리고 항목 옆에 x표시된 버튼을 누르면 해당 Device의 삭제가 가능합니다. 화면 우측 하단의 + 버튼을 누르면 새로운 디바이스를 추가할 수 있습니다.

새로운 디바이스를 추가하는 페이지에서는 디바이스의 이름을 입력합니다. 그리고 화면 상단의 헤더 패널 중 디스크 모양의 아이콘을 누르면 새로운 디바이스가 저장되며, 저장된 내용이 반영된 Device Management Page로 이동합니다.

4.6. Settings

이 부분에서는 Mobile App의 설정을 관리하는 방법에 대해 설명합니다.

Settings에서는 앱의 최초 실행 시 설정된 username과 url, 그리고 앱 데이터의 업데이트 주기와 모니터링 여부를 변경할 수 있습니다. 앱 데이터의 업데이트 주기를 변경하면 그 주기에 맞춰 현

재 디바이스의 센서데이터가 업데이트됩니다. 또 모니터링을 켜놓으면 센서 데이터가 조건을 위반했을 때 서버에서 사용자에게 이메일을 전송함으로써 사용자가 현재 식물의 이상여부를 알 수 있습니다.

원하는 내용으로 세팅을 변경한 뒤 화면 상단의 헤더 패널 중 디스크 모양의 아이콘을 누르면 변경한 설정이 서버에 저장됩니다.

이로써 Mobile App System에 대한 설명을 마칩니다.

5. Contact

시스템 사용 도중 문제가 발생하거나, 개선을 건의하고 싶을 때 다음 방법으로 연락이 가능합니다.

1) Email : amoretspero@gmail.com

2) Github Issue Tracker : https://github.com/cloudtrack/lustitia/issues

6. License