부산대학교
PUSAN NATIONAL UNIVERSITY

대규모 사물인터넷 단말 관리를 위한 클라우드 기반 OTA 기술 개발

부산대학교 정보컴퓨터공학부
2025학년도 전기 졸업과제 착수보고서

지도교수 김 태 운

팀 명 커피는생필품

팀 원 201924479 박준우

202255665 송승우

202255670 장민준

목 차

1. 과제 개요

1.1 개요

본 과제는 많은 산업 분야에 여전히 많이 운영되고 있는 전통적인 임베디드 시스템에서 IoT 및 OTA 기술을 사용할 수 있도록 하드웨어를 구성하고, 최근 IoT 기술에도 적극적으로 접목되기 시작한 클라우드 컴퓨팅 기술을 활용하여 OTA를 대규모 단말 환경에 효과적으로 적용할 수 있는 기술을 개발한다. 클라우드 기반의 대규모 IoT 단말 OTA 기술을 개발함으로써, 물리적으로 접근이 어려운 환경에서도 대규모의 IoT 단말을 효율적으로 관리할 수 있는 방안을 제시한다.

1.2 배경

현재 산업 현장에는 **IoT 기능이 적용되지 않은 기존 임베디드 시스템이 여전히 많이 운영**되고 있으며, 이 중 많은 수는 **작업자가 접근하기 어려운 위치에 설치**되곤 한다. 이러한 환경의 시스템들은 제한된 기능과 수동적인 운영으로 인해 지속적인 유지보수와 업데이트에 큰 제약이 있다.[1] 특히 많은 곳에 분산되어 설치된 시스템의 경우, 물리적인 방식으로 펌웨어 및 컨텐츠를 업데이트하거나 운영 정책을 변경하는데에는 막대한 시간과 인력이 소요된다.

최근에는 기존의 시스템에 IoT 기능을 후속으로 추가하고, 이를 기반으로 OTA(Over-the-air) 기술을 활용하고자 하는 수요가 빠르게 증가하고 있다. 보일러, 승강기, 음료 디스펜서 등 다양한 전통 산업 분야에서 기존 장비를 IoT화하고 OTA 기술을 통해 보안 패치, 기능 및 컨텐츠 업데이트, 설정 변경 등을 자동화하려는 시도가 확산하고 있다.[2]

스마트홈, 웨어러블 기기 등 관리 주체의 범위가 좁은 개인 및 가정용 IoT 환경의 경우, 효율적 관리보다 스마트폰 애플리케이션 등을 이용한, 편의성을 중시하는 방식을 통해 기기를 관리할 수 있으나, 전국 혹은 전 세계에 퍼져있는 IoT 단말을 관리해야하는 기업의 경우에는 **대규모 IoT 단말을 동일한 상태로 관리하고, 기기의 범위가 넓어지더라도 자동으로 확장되며, 중앙 집중식으로 관리 및 상태 모니터링을 할 수 있는 기술이 필요**하다.

1.3 과제 목표

이 프로젝트는 물리적 접근 없이도 전 세계에 분산된 IoT 기기들을 효율적으로 관리하고, 실시간으로 모니터링하며, 신속하게 소프트웨어와 컨텐츠를 업데이트할 수 있는 통합 플랫폼을 구축하는 것을 목표로 한다. 이를 위해 달성되어야 하는 세부 목표는 다음과 같다.

1. **웹 기반 통합 원격 관리 시스템 구축**
    - 중앙화된 클라우드 플랫폼을 통해 전 세계에 분산된 IoT 기기를 원격으로 관리
    - 역할 기반 접근 제어 시스템을 통한 관리자 권한 분리
    - 배포 일정 예약 및 자동화 기능 구현

2. **대규모 동시 배포 시스템 개발**
    - 수백~수천 대의 IoT 기기를 동시에 업데이트 할 수 있는 확장성 있는 시스템 구현
    - 부하 분산 및 효율적인 네트워크 자원 활용을 위한 배포 스케줄링 기능 구현
    - 오프라인 디바이스를 위한 자동 재배포 메커니즘
    - 디바이스별, 그룹별 등 선택적 배포 가능

3. **실시간 디바이스 모니터링 인프라 구축**
    - 각 IoT 기기의 상태 (펌웨어 버전, 작동 상태, 온도, 네트워크 상태 등)을 실시간으로 모니터링
    - 데이터 시각화를 통한 직관적인 모니터링 대시보드 제공
    - 배포 이력 및 모니터링 데이터 저장/분석 기능

4. **기존 임베디드 시스템의 IoT 기능 추가를 위한 하드웨어 모듈 개발**
    - 기존 산업용 임베디드 시스템에 부착 가능한 범용 IoT 확장 모듈 설계
    - 저전력 무선 통신(Wi-Fi) 기능 탑재
    - 기존 시스템 간섭 최소화 및 간편 설치를 위한 소형화 설계
    - 산업 환경에 적합한 내구성 및 안정성 확보

1.4 기대 효과

본 클라우드 기반 OTA 원격 관리 서비스 구축을 통해 다음과 같은 효과를 기대할 수 있다.

1. **운영 효율성 향상**
    - **현장 방문 비용 절감**: 물리적 방문 없이 원격으로 펌웨어 및 컨텐츠 업데이트가 가능해져 인력 및 출장 비용을 절감할 수 있다.
    - **관리 시간 단축**: 중앙화된 관리 시스템을 통해 수백~수천 대의 디바이스를 동시에 관리함으로써 업데이트 배포 시간이 단축될 것으로 예상된다.
    - **인적 오류 감소**: 자동화된 배포 및 모니터링 시스템을 통해 수동 작업 과정에서 발생할 수 있는 오류를 최소화할 수 있다.

2. **서비스 품질 제고**
    - **신속한 기능 개선**: 펌웨어 업데이트를 신속하게 배포할 수 있어 버그 수정 및 새로운 기능 추가가 즉시 적용되어 사용자 경험이 향상된다.
    - **일관된 사용자 경험**: 모든 디바이스가 최신 상태로 유지되어 위치에 관계없이 사용자에게 일관된 경험을 제공한다.

3. **비즈니스 민첩성 강화**
    - **마케팅 전략 유연성**: IoT 기기의 컨텐츠를 실시간으로 변경할 수 있어 시즌별, 지역별, 이벤트별 맞춤형 마케팅이 가능해진다.
    - **사업 확장 용이성**: 새로운 지역이나 고객에게 서비스를 확장할 때 중앙 관리 시스템을 통해 신속하게 통합 및 관리가 가능해진다.

2. 과제 구성

2.1 요구사항 분석

① 기능적 요구사항

표 2.1.a는 본 시스템이 수행해야 할 주요 기능적 요구사항을 정리한 것이다.

표 2.1.a 기능적 요구사항

기능	설명
펌웨어 업로드	관리자가 웹 클라이언트를 통해 새로운 펌웨어 파일을 업로드 할 수 있어야 한다.
펌웨어 OTA 배포	업로드된 펌웨어를 특정 디바이스 또는 그룹에 원격으로 배포할 수 있어야 한다.
배포 스케줄 설정	통합된 컨텐츠를 특정 기기 또는 디바이스 집단에 지정된 조건에 맞춰 배포할 수 있어야 한다.
디바이스별 펌웨어 조회	각 디바이스에 적용된 펌웨어 이력을 조회할 수 있어야 한다.
디바이스 KPI 모니터링	디바이스의 온도, 네트워크 상태, 작동 시간 등의 상태 정보를 확인할 수 있어야 한다.
디바이스 목록 확인	디바이스 ID, 위치 등의 목록 정보를 확인할 수 있어야 한다.
로그 수집	디바이스에서 발생한 로그나 오류 정보를 서버로 수집할 수 있어야 한다.
파일 메타데이터 관리	업로드한 펌웨어에 대한 버전, 작성자, 등록일의 정보를 기록할 수 있어야 한다.
MQTT 메시지 송수신	MQTT를 통해 디바이스에 다운로드 링크를 전달할 수 있어야 한다.
업데이트 실패 대응	업데이트 실패 시 롤백하거나 재시도를 할 수 있어야 한다.
CDN 연동	펌웨어를 CDN을 통해 빠르게 전달하고, 디바이스는 링크를 통해 다운로드할 수 있어야 한다.

② 비기능적 요구사항

표 2.1.b는 시스템이 충족해야 할 비기능적 요구사항을 나타낸다.

표 2.1.b 비기능적 요구사항

기능	설명
가용성	시스템은 24시간, 365일 안정적으로 운영되어야 한다.
성능	수백-수천 대의 디바이스에 요청을 보내더라도 성능 저하 없이 처리가 가능해야 한다.
이력 추적성	배포 내역, 오류, 성공 여부의 모든 중요 이벤트는 기록되어야 한다.
보안성	펌웨어 업로드 및 다운로드 시, 암호화된 통신으로 외부에 노출되서는 안된다. (-)
접근 제어	관리자만이 시스템을 사용할 수 있어야 한다.

2.2 시스템 구성

2.2.1 시스템 구성 개요

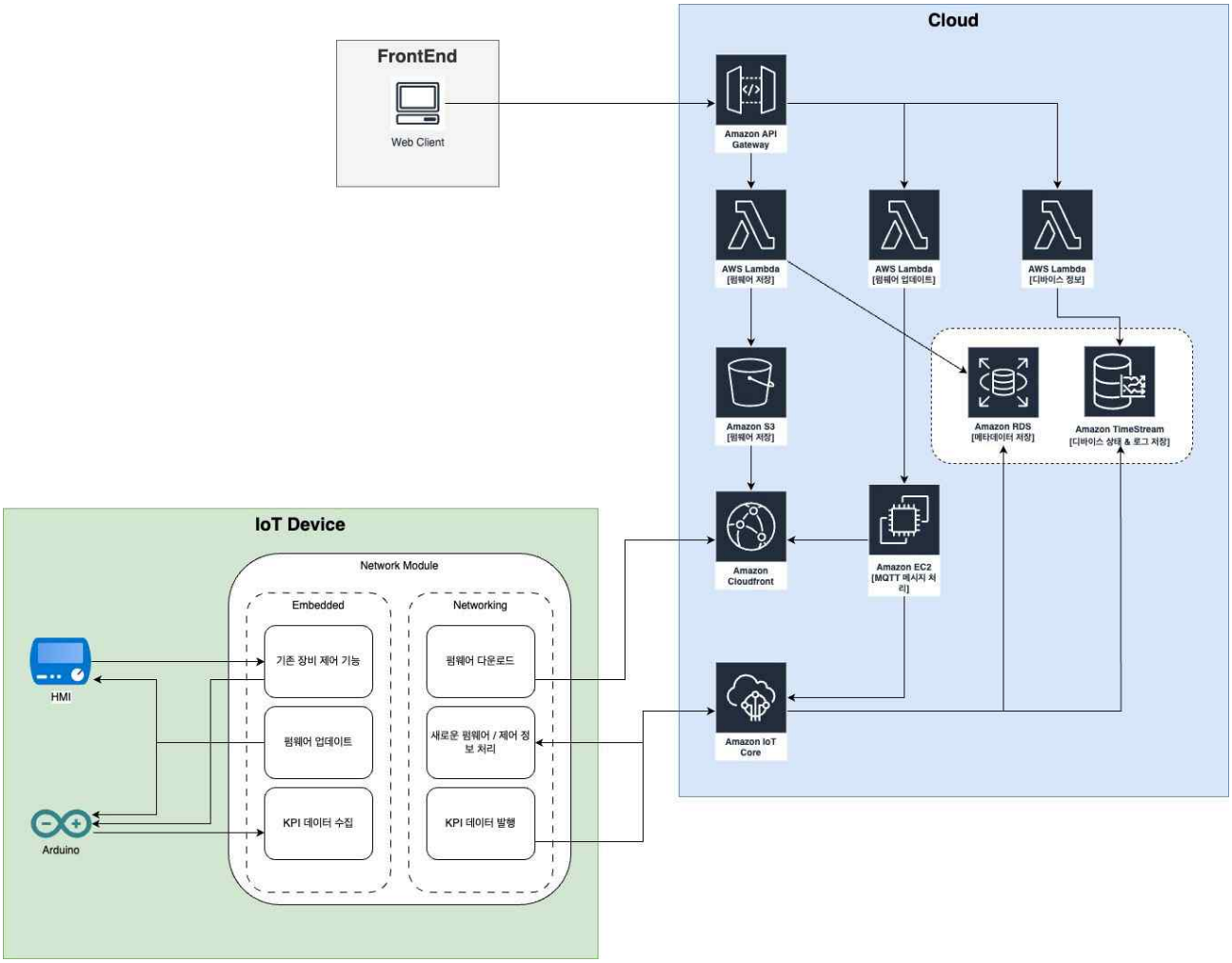

그림 2.2.1.a 시스템 구성 개요

2.2.2 웹 클라이언트

본 시스템의 프론트엔드는 사용자 친화적인 인터페이스를 구축을 목표로 다음과 같은 기술 스택 및 라이브러리를 사용하여 개발한다.

기술 스택 및 라이브러리

- 기본 프레임워크: Typescript, React
- 상태 관리: Redux Toolkit
- UI 디자인: Tailwind CSS, lucide-react
- 데이터 시각화: Recharts
- 코드 품질 관리: ESLint, Prettier, Jest

주요 페이지와 기능은 다음과 같다.

대시보드 페이지

대시보드는 시스템의 전반적인 상태와 핵심 지표를 한눈에 파악할 수 있는 중앙 허브 역할을 한다. 전체 디바이스의 현황을 요약하여 보여주며, 사용자가 시스템 전체 상태를 직관적으로 모니터링할 수 있다.

대시보드 페이지는 다음과 같은 기능을 제공한다.

- 전체 디바이스 현황 요약 정보 표시 (총 디바이스 개수, 온라인 디바이스 개수, 펌웨어 버전 등)
- 최근 업데이트 활동 로그 조회
- 주요 성능 지표(KPI) 시각화 (업데이트 성공률 등)

펌웨어 관리 페이지

펌웨어 관리 페이지는 IoT 디바이스에 배포될 펌웨어 패키지의 라이프사이클을 관리하는 공간이다. 사용자는 이 페이지를 통해 새로운 펌웨어 패키지를 시스템에 등록하고 관리할 수 있다.

펌웨어 관리 페이지는 다음과 같은 기능을 제공한다.

- 펌웨어 패키지 업로드 및 등록
- 버전 관리 및 변경 이력 추적
- 릴리즈 노트 및 메타데이터 편집
- 버전별 펌웨어 상세 정보 조회
- 펌웨어 삭제 및 아카이브

펌웨어 배포 페이지

펌웨어 배포 페이지는 등록된 펌웨어를 IoT 디바이스에 실제로 배포하는 프로세스를 관리한다. 사용자는 배포 대상, 일정, 전략 등을 설정하고 배포 과정을 모니터링할 수 있다.

펌웨어 배포 페이지는 다음과 같은 기능을 제공한다.

- 대상 디바이스 그룹 선택 및 필터링
- 배포 일정 설정 (즉시/예약 배포)
- 배포 진행 상황 실시간 모니터링
- 배포 중단 및 롤백 기능
- 배포 이력 조회
- 오프라인 디바이스 자동 재배포 설정

웹 클라이언트의 사용자 시나리오는 다음과 같다. 상태 조회는 “대시보드” 페이지에서 바로 확인 가능하며, 펌웨어 관련 사용자 시나리오는 다음과 같다.

주요 동작 시나리오

- **펌웨어 등록**
    1. 관리자로 로그인한다.
    2. 관리자는 “펌웨어 관리” > “펌웨어 등록” 버튼을 클릭한다.
    3. 관리자는 펌웨어 버전, 릴리즈 노트, 펌웨어 파일 등을 작성 & 첨부한다.
    4. 웹 클라이언트가 서버에 펌웨어 등록 API를 호출한다.
    5. 웹 클라이언트는 펌웨어가 성공적으로 업로드 되었는지를 알려준다.
- **펌웨어 업데이트 배포**
    1. 관리자가 로그인한 후, 사이드바에서 “펌웨어 관리”를 클릭한다.
    2. 웹 클라이언트는 현재 업로드 되어있는 펌웨어의 목록을 보여준다.
    3. 관리자는 배포하고 싶은 펌웨어를 클릭한다.
    4. 관리자는 “배포하기” 버튼을 클릭한다.
    5. 관리자는 배포 타겟을 설정할 수 있다. (리전 별, 그룹 별, 디바이스 별 등)
    6. 관리자는 “배포” 버튼을 클릭하여 배포를 진행한다.
    7. 웹 클라이언트는 해당 펌웨어의 “상세 정보” > “펌웨어 배포 정보”에 새롭게 등록한 배포 상태를 보여준다.
    8. 관리자는 배포 상태 (진행 중, 완료 등)을 확인할 수 있다.

2.2.3 서버

본 시스템은 유지보수성과 확장성을 고려하여 백엔드 구성요소를 기능별로 독립된 서비스 단위로 나누는 **마이크로서비스 아키텍처(MSA; Microservices Architecture)**를 적용하였다. 펌웨어 업로드, 펌웨어 업데이트 요청, 디바이스 정보 모니터링의 **각 기능은 독립된 Lambda 함수(서비스)로 구성**되어 있으며, 이들 서비스는 **AWS API Gateway를 통해 통합 관리**된다.

API Gateway는 클라이언트 요청을 해당 기능별 Lambda 서비스로 라우팅하는 역할을 하며, 보안 정책, 요청 검증, 등을 중앙에서 관리할 수 있도록 구성되어 있다. 이러한 구조는 각 기능별 서비스가 서로 영향을 받지 않고 독립적으로 개발, 배포, 확장될 수 있도록 하여, 대규모 IoT환경에서 발생할 수 있는 서비스 병목이나 장애 상황을 효과적으로 분산시킬 수 있는 장점을 제공한다.

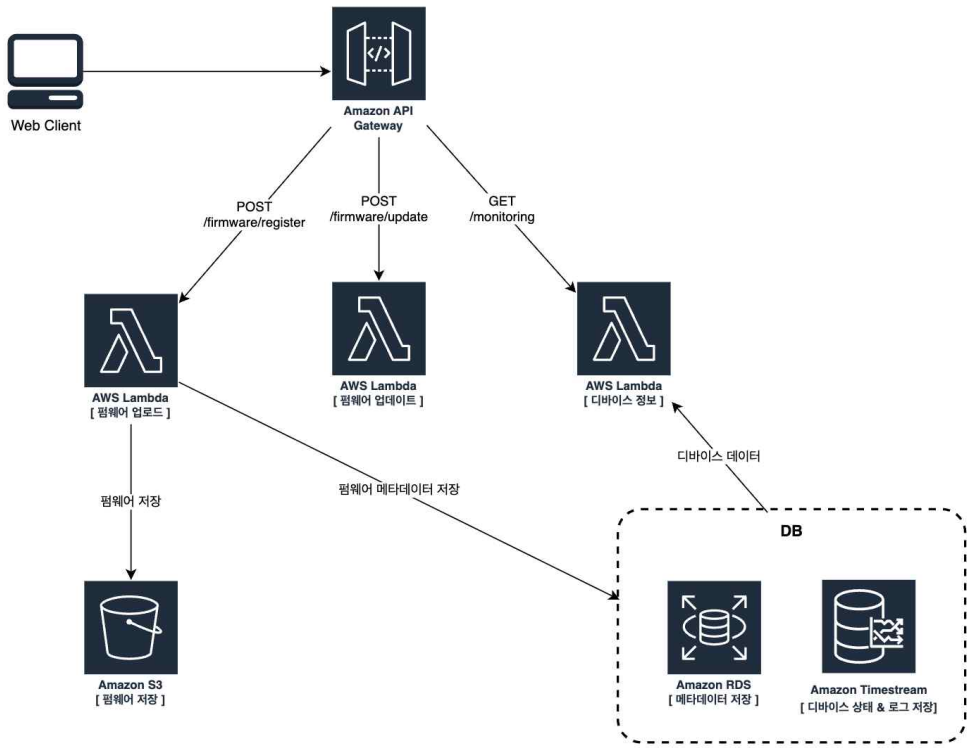

그림 2.2.3.a MSA 기반 서버 아키텍처

AWS API Gateway
AWS API Gateway는 클라이언트 요청을 받아 각 Lambda 함수로 라우팅하는 API 관문 서비스이다.

AWS API Gateway는 다음과 같은 기능을 제공한다.
- 인증, 로깅 등의 기능을 제공
- Lambda 및 AWS 리소스와의 연동이 자연스럽고 효율적
- 엔드포인트 관리 용이

AWS Lambda
AWS Lambda는 요청 기반으로 실행되는 Serverless컴퓨팅 서비스이다.

AWS Lambda는 다음과 같은 기능을 제공한다.
- 기능별로 독립된 Lambda 함수로 마이크로서비스 구성
- 트래픽에 따라 자동 확장되며, 서버 관리 불필요
- 펌웨어 배포, Presigned URL 발급, 메타데이터 저장 등 주요 로직을 처리

Amazon RDS
Amazon RDS는 펌웨어 메타데이터와 배포 정보 등을 저장하는 관계형 데이터베이스이다.

Amazon RDS는 다음과 같은 기능을 제공한다.
- 펌웨어 버전, 펌웨어 업로드 일자, 대상 디바이스 그룹 등의 정보 관리
- 관리자 웹 클라이언트에서 조회 및 배포 이력 확인에 사용

Amazon Timestream
Amazon Timestream은 각 단말에서 수집된 시간 기반 시계열 데이터를 저장하는 전용 데이터베이스이다.

Amazon Timestream은 다음과 같은 역할을 한다.
- 온도, 네트워크 상태 등의 실시간 데이터 저장
- 디바이스 상태 모니터링 및 대시보드 구성에 활용

Amazon S3 Bucket
Amazon S3 Bucket은 펌웨어 파일을 저장하는 객체 기반 저장소이다.

Amazon S3 Bucket은 다음과 같은 역할을 한다.
- Presigned URL 방식으로 안전하고 효율적인 대용량 파일 업로드 및 다운로드 지원
- CloudFront와 연동해 전 세계 엣지에서 컨텐츠 제공 가능

주요 동작 시나리오
- **펌웨어 업로드**
    1. 관리자가 Web Client에서 펌웨어 파일 업로드 요청
    2. API Gateway를 통해 Lambda에 요청 전달
    3. Lambda는 S3에 업로드할 수 있는 Presigned URL을 생성해 클라이언트에 응답
    4. 클라이언트는 해당 URL을 이용해 펌웨어 파일을 S3에 업로드
    5. Lambda는 RDS에 해당 펌웨어 파일의 메타데이터를 저장

- **펌웨어 업데이트 배포**
    1. 관리자가 Web Client에서 특정 디바이스 그룹을 선택해 배포 요청
    2. API Gateway를 통해 Lambda에 요청 전달
    3. Lambda가 IoT Core에 MQTT 메시지 송신

- **디바이스 모니터링**
    1. 관리자가 Web Client에서 디바이스 상태 조회 요청
    2. API Gateway를 통해 Lambda에 요청 전달
    3. Lambda는 RDS 및 TimeStream에 저장된 디바이스의 메타데이터, 상태 정보, 로그를 조회
    4. 조회된 디바이스 상태를 관리자에게 반환

2.2.4 클라우드

본 시스템은 IoT 기기 펌웨어의 효율적이고 신뢰성 있는 배포를 목표로 하며, 네트워크 지연 최소화, 보안, 상태 추적이 주요 고려사항이다. 따라서 클라우드 아키텍처는 다음과 같이 구성한다.

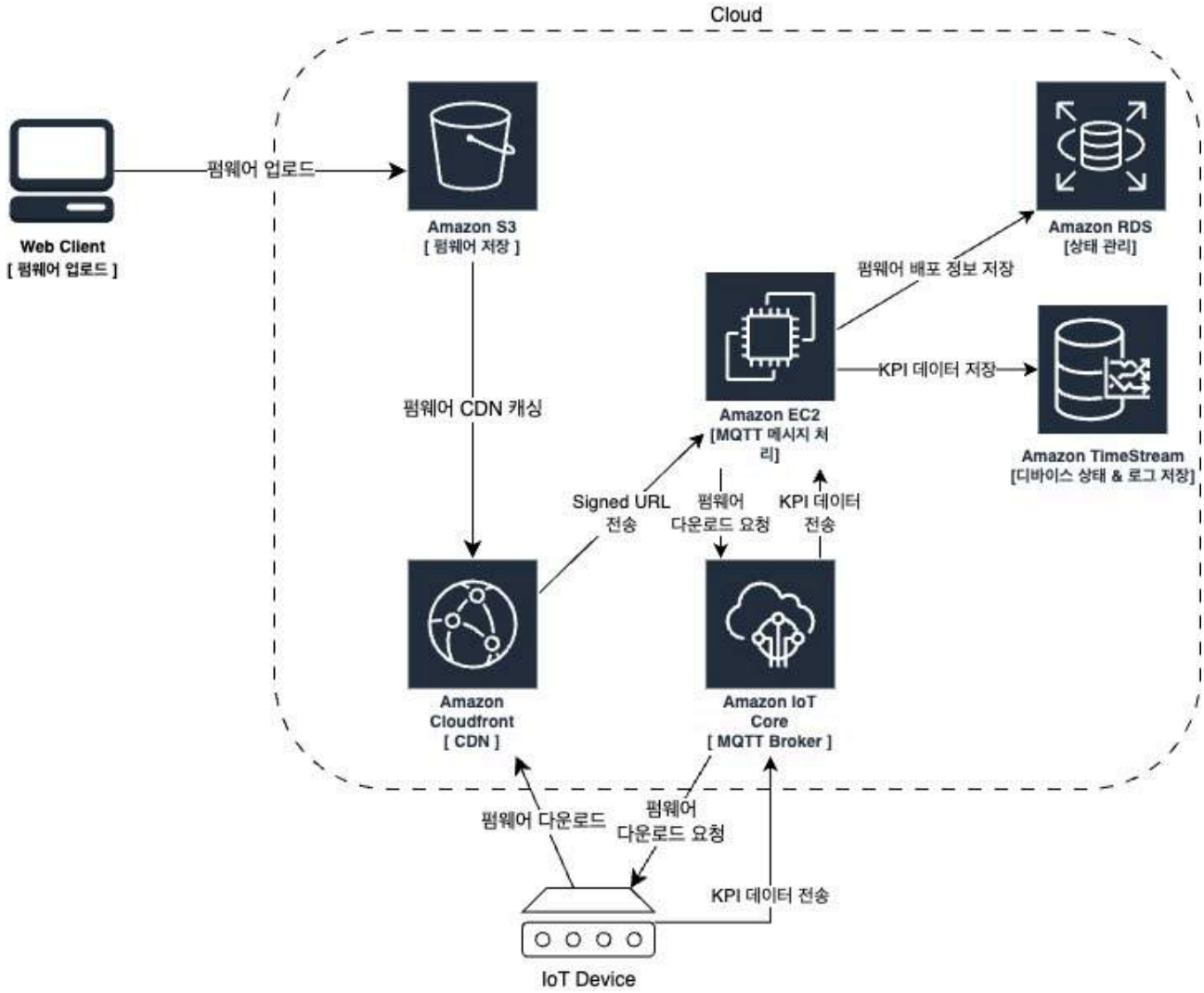

그림 2.2.4.a 클라우드 아키텍처

AWS CloudFront

Amazon CloudFront는 전 세계에 분산된 엣지 로케이션을 통해 콘텐츠를 캐싱하고 제공하는 CDN(Content Delivery Network) 서비스이다. IoT 기기들이 지리적으로 다양한 위치에 존재하는 환경에서, CloudFront는 펌웨어 파일을 기기와 가까운 위치에서 제공함으로써 네트워크 지연을 최소화하고 다운로드 속도를 향상시킨다.

AWS CloudFront는 다음과 같은 기능을 제공한다.

- S3에 저장된 펌웨어 파일을 CloudFront를 통해 배포
- 엣지 로케이션 기반으로 빠른 응답 제공
- 전 세계 수백~수천 대 기기의 동시 다운로드 요청을 효과적으로 처리

AWS IoT Core

AWS IoT Core는 수백만 대의 IoT 기기와 안전하게 통신할 수 있는 MQTT 기반 메시징 서비스를 제공한다. 이 시스템에서는 펌웨어 다운로드 URL 전달과 다운로드 완료 확인 등 양방향 통신에 IoT Core가 핵심 역할을 한다.

AWS IoT Core는 다음과 같은 기능을 제공한다.

- 경량화된 MQTT 프로토콜로 저전력 기기에서도 통신 가능
- 서버에서 기기로 Pre-signed URL 전달
- 기기로부터 다운로드 완료 메시지를 수신

특히 MQTT 토픽 구조를 활용하여 기기들을 다음과 같이 유연하게 타겟팅할 수 있다.

- Region별: firmware/update/region/{region_name}
- Group별: firmware/update/group/{group_id}
- 기기 ID별: firmware/update/device/{device_id}

이러한 구조를 통해 관리자는 특정 지역 또는 그룹, 혹은 특정 단일 기기를 대상으로 펌웨어 배포 메시지를 전송할 수 있다. 이는 기기 규모가 커질수록 메시지 트래픽을 효율적으로 관리하는 데 유리하다.

CloudFront Signed URL
펌웨어 파일을 저장한 Amazon S3 객체에 대해 미리 서명된 URL을 생성함으로써, 인증되지 않은 외부 요청을 방지하면서도 제한된 시간 동안 URL을 통해 다운로드를 허용할 수 있다.

Signed URL은 다음과 같은 기능을 제공한다.

- 기기 인증 없이도 안전한 다운로드 가능
- URL에는 만료 시간과 접근 권한이 포함되어 유출 위험 최소화
- MQTT를 통해 기기마다 고유한 Signed URL 전달 가능
- URL 생성 시 서버 측에서 다운로드 만료 시점 및 유효 범위 제어 가능

주요 동작 시나리오

- **펌웨어 업데이트 요청**
    1. 관리자가 웹 클라이언트에서 펌웨어 배포 요청을 생성한다.
    2. 서버는 해당 객체에 대해 Signed URL을 생성하고, 이를 MQTT 메시지를 통해 IoT 기기에 전달한다.
    3. IoT 기기는 CloudFront를 통해 펌웨어를 다운로드한다.
    4. 다운로드 완료 후, IoT 기기는 MQTT 메시지로 상태를 서버에 전달한다.
    5. 서버는 해당 메시지를 수신하여 상태 정보를 DB에 저장한다.

- **KPI & 디바이스 상태 저장**
    1. IoT 디바이스에서 KPI 데이터 & 상태를 MQTT 메시지로 전송한다.
    2. MQTT 메시지 처리 서비스가 AWS IoT Core로부터 메시지를 수신한다.
    3. KPI 데이터를 Amazon TimeStream에 저장한다.

2.2.5 IoT 기기

기존의 임베디드 시스템은 단독 동작을 전제로 설계되어 있어, 외부와의 실시간 연동이나 원격 제어 기능을 갖추지 않은 경우가 많다. 본 과제의 임베디드 파트에서는 이러한 **시스템에 IoT 기능을 후속으로 탑재**하고, **원격 OTA 업데이트**를 가능하게 하기 위한 구조를 설계한다.

이 과정에서 사용되는 네트워크 모듈은 기존 장비의 제어 기능을 유지하면서도, 네트워크 기능을 추가하여 클라우드 서버와의 통신이 가능하도록 구현한다. 이러한 구조는 기존 산업 기기에 **별도 마이그레이션 없이 외부 모듈만으로 IoT 기능을 부여**할 수 있다는 장점이 있으며, 복잡한 내장 시스템의 큰 수정 없이도 **빠른 적용이 가능**하다는 실용적 이점을 가진다.

네트워크 모듈에서 작동할 펌웨어는 아래와 같이 크게 클라우드와의 통신을 처리하는 **네트워킹 영역**과 사용자 정의 작업들을 처리하는 **임베디드 영역**으로 나누어진다.

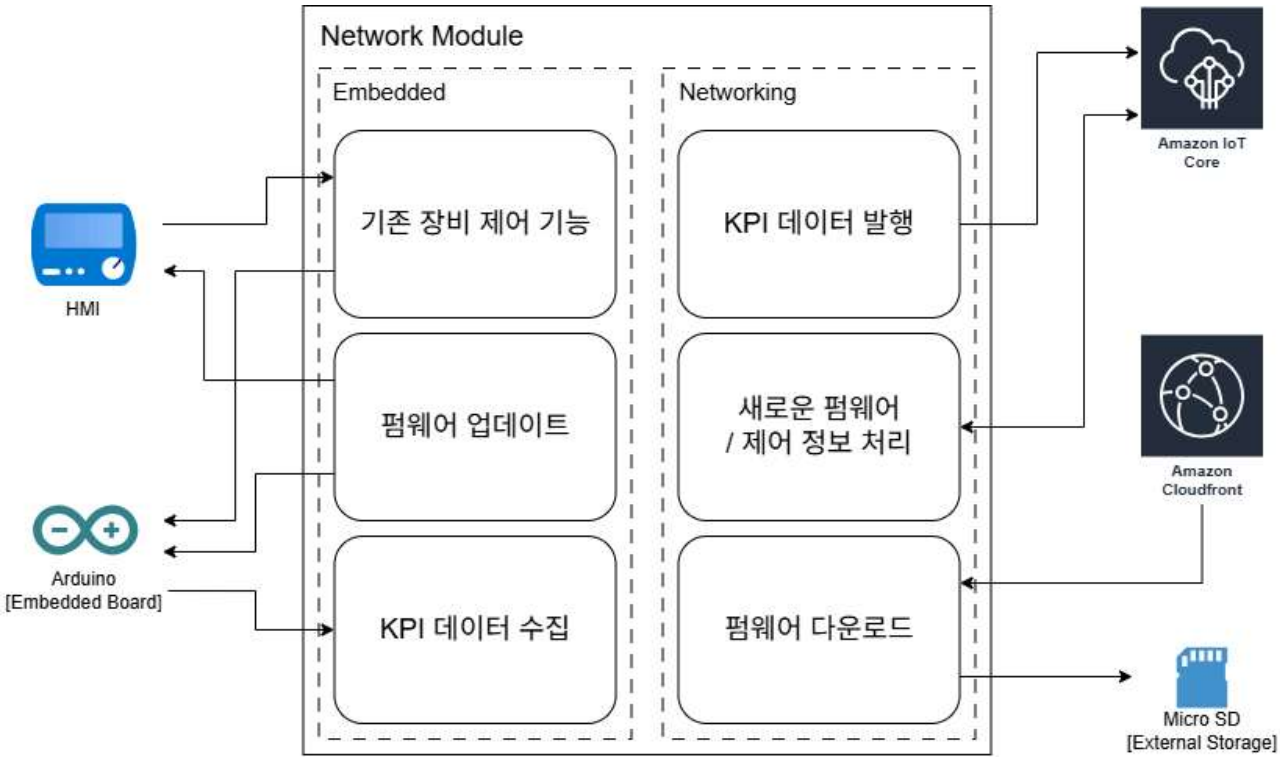

그림 2.2.5.a 네트워크 모듈 아키텍처

네트워킹 영역

- 새로운 펌웨어 및 제어 정보를 MQTT Broker(Amazon IoT Core)로부터 받아 처리
- CDN(Amazon Cloudfront)으로부터 새로운 펌웨어를 받아 외부 저장소에 저장
- 모듈이 제어 중인 기기에서 발생하는 KPI 데이터를 받아 MQTT로 발행

임베디드 영역

- 기존 임베디드 시스템에서 처리하던 제어 기능 중계
- 새로운 펌웨어가 준비되면 제어 중인 기기에 펌웨어 업데이트
- 모듈이 제어 중인 기기에서 발생하는 KPI 데이터 수집

2.3 논의 사항

[Server] 마이크로서비스 구현 방식

본 프로젝트는 펌웨어 파일의 업로드, 업데이트 요청, 디바이스 상태 모니터링 등의 기능이 명확히 분리되어 있으며, 서비스 간 결합도가 낮은 구조를 갖는다. 이에 따라 서버 구성 방식으로는 EC2 기반의 Kubernetes 클러스터(EKS)를 사용하는 방식과 AWS Lambda 기반의 Serverless 방식 두 가지를 검토하였다.

프로젝트의 주요 기능들은 대부분 단발성 이벤트 중심의 작업이며, 지속적인 트래픽이 발생하지 않기 때문에, 고정 리소스를 사용하는 EKS 방식보다는 Lambda 기반 Serverless 구조가 비용 및 운영 측면에서 유리하다고 판단하였다.

따라서 본 시스템은 AWS Lambda 기반 Serverless 아키텍처를 채택하였다.

[Server] API Gateway 구현 방식

클라이언트 요청을 각각의 마이크로서비스로 라우팅하고 인증, 로깅, 요청 제한 등의 처리를 담당하는 API Gateway 구현 방식으로는 Spring Cloud Gateway와 AWS API Gateway 두 가지 대안을 검토하였다.

Spring Cloud Gateway는 라우팅 커스터마이징에 강점이 있으나, 직접 서버를 운영해야 하며 AWS 서비스와의 연동이 복잡하다는 단점이 있다. 반면, AWS API Gateway는 Lambda와의 연동이 뛰어나고, 인증·확장·보안 기능을 AWS가 기본 제공하므로 Serverless 아키텍처에 적합하다.

이에 따라 API Gateway 구성 방식으로는 AWS API Gateway를 채택하였다.

[IoT] 네트워크 모듈의 구현 방법 & OTA 구현을 위한 하드웨어의 조건

실제 산업에서 사용되는 Arduino의 경우, 학습용으로 널리 사용되는 표준 Arduino와 달리 표준 12V로 작동하는 산업용 기기 제어를 위해 일반적으로 12V를 기준으로 동작하며, 그 외 방수 / 방진 강화, 기기 제어에 특화된 통신 포트 구성 등 각 기업의 필요에 특화된 구성을 가진 경우가 많아 **기능적 확장을 구현하기에 어려움**이 크다[가번호 2.3.1].

때문에 본 과제에서는 여러 환경의 PLC(Programmable Logic Circuit)에서 사용할 수 있도록 **별도의 임베디드 보드**를 두어 네트워크 작업을 처리하는 모듈로 개발하고자 한다.

산업용 Arduino와 최소 하나 이상의 산업 기기를 포함하는 임베디드 환경에서 OTA 기술을 구현하기 위해서는 추가 모듈로 사용할 하드웨어의 성능이 네트워크 작업 뿐 아니라 인터페이스 등 기타 추가 기기 제어를 처리할 수 있을 정도가 되어야 한다.

본 과제에서는 임베디드 보드의 네트워킹 확장 모듈에 많이 사용되는 Espressif사의 ESP계열 보드를 활용하여 네트워크 모듈을 구현할 예정이다. 펌웨어 다운로드 작업은 시간 복잡도가 높은 작업이기 때문에 타 작업을 그대로 수행하는 동시에 펌웨어 다운로드를 수행할 수 있도록 RTOS 기반 **실시간 처리가 가능한 듀얼코어 MCU를 사용할 필요**가 있는데, ESP32 보드의 경우 대부분의 개량판 보드들의 MCU가 기본으로 FreeRTOS 기반 실시간 처리를 지원하므로 **ESP32 MCU 기반 보드를 활용**한다.

그 외 OTA 구현을 위한 네트워크 모듈의 하드웨어 조건은 아래와 같다.

- Micro SD 소켓 등 **외부 저장소 확장**이 가능해야 함
- **Wi-Fi 모듈**과 (시스템 확장 가능성을 고려하여) Bluetooth 모듈 포함
- Arduino 및 HMI와 통신을 위한 **통신 포트 포함**
- TLS(HTTPS) 통신만 가능한 AWS CDN과 통신하기 위해 크기가 큰 인증서 처리와 대용량의 파일 처리를 수행할 수 있는 **충분한 메모리**
- Arduino 호환(Arudino IDE를 통한 개발 가능성)

2.4 한계 및 제약 사항

앞서 2.3절에서 설명하였듯이, OTA(Over-the-Air) 기술을 실제 산업 환경에 적용하기 위해서는 IoT 단말 측에서 **OTA 기능을 수용할 수 있는 하드웨어적 기반이 중요**하다. 본 과제에서는 기존 하드웨어의 기능 확장 한계를 고려하여, 별도의 네트워크 모듈을 추가하는 방식으로 OTA 기능을 구현하였다. 이를 통해 기존 시스템의 구조를 크게 변경하지 않고도 펌웨어 업데이트를 가능하게 하였지만, 해당 방식에는 여러 가지 제약 사항이 존재한다.

첫째, 외부 모듈로부터 펌웨어를 수신하고 이를 내부 마이크로컨트롤러에 전달하는 과정에서 상당한 **오버헤드가 발생**할 수 있다. 산업 현장에서 널리 사용되는 시리얼 통신 인터페이스(UART, SPI, I2C 등)는 일반적으로 센서 데이터와 같은 소용량, 저주기 통신을 전제로 설계된 경우가 많다. 이로 인해 펌웨어와 같이 수십 MB 이상의 대용량 데이터를 전송할 때 안정성과 속도가 저하되는 문제가 발생할 수 있다.

둘째, 많은 IoT 기기들은 **네트워크 연결이 불안정한 환경**에서 동작한다. 특히 산업 현장에서는 무선 간섭, 약한 신호 세기, 네트워크 커버리지 문제 등으로 인해 OTA 중단 또는 부분 다운로드 실패와 같은 상황이 발생할 수 있으며 이는 시스템 안정성에 직접적인 영향을 줄 수 있다. 따라서 OTA 기능을 구현할 때는 재시도 로직, 중단된 다운로드 복구 기능, 무결성 검증 등을 함께 고려해야 한다.

셋째, **IoT 단말의 메모리 자원 한계** 또한 OTA 구현에 중요한 제약 조건이다. 많은 IoT 기기들은 제한된 Flash 및 RAM 용량을 가지고 있으며, 다운로드 중 버퍼링, 무결성 검증 등을 위해서는 추가적인 저장 공간이 필요하다. 하드웨어 설계상 이러한 공간이 확보되지 않은 경우, OTA 구현 자체가 불가능하거나 제한적인 형태로만 가능하다.

이처럼 IoT 환경에서 OTA 기술을 안정적으로 적용하기 위해서는 하드웨어 사양, 통신 환경, 메모리 제약 등을 종합적으로 고려한 설계가 요구되며, 경우에 따라 하드웨어 업그레이드 또는 OTA 전략의 조정이 불가피할 수 있다.

3. 수행 계획

3.1 역할 분담

이름	역할 분담
박준우	프론트엔드 개발 ● Figma를 활용한 UI 디자인 ● React를 사용한 IoT 기기 관리 페이지 개발 클라우드 개발 ● CDN 캐싱 서버 구현 ● IoT 기기와 클라우드 간의 MQTT 통신 구성 & 관리 로직 구현
송승우	백엔드 개발 ● API 설계 ● 클라우드 기반 Serverless 백엔드 서버 구성 ● 데이터베이스 구성 및 Schema 디자인
장민준	IoT 기기 개발 ● IoT 및 OTA 기술 구현을 위한 하드웨어 모듈 구성

	• 클라우드 - IoT 단말 간 MQTT & HTTPS 통신 및 펌웨어 다운로드 로직 구현 • IoT 기기 OTA 로직 구현

3.2 개발 일정

업무	5월				6월				7월				8월				9월			
	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
기획 및 설계																				
착수 보고서 작성																				
프론트엔드 개발																				
백엔드 개발																				
IoT 개발																				
클라우드 개발																				
중간보고서 작성																				
시스템 통합																				
테스트 및 디버깅																				
최종 테스트 및 배포																				
최종 보고서 작성																				
발표 준비																				

4. 참고 자료

[1] 임베디드 MCU 애플리케이션의 OTA 업데이트 설계 시 고려사항 - 벤자민 버클린 브라운, ANALOG DEVICES(https://www.eewebinar.com/adi/tech_view.asp?c=14&f=0&idx=426&page=3)
[2] 임베디드 IoT 디바이스를 위한 안전하고 강력한 over-the-air 소프트웨어 업데이트 - tQCS (https://ota.tqcs.io/4459249324-01.html)

-끝-