Smart-Factory
Server
최종보고서

학과	정보통신학과
학번	91714806
이름	허정운

0. 중간 발표 이전 요약 및 이후 변경된 점

1) Express 사용

초기에는 가장 인기있던 웹 프레임워크인 Express 를 통하여 공부와 개발을 했었습니다. 이유는 사용하는 사람이 많아 자료가 방대했기에 공부하기 수월하다고 판단하였기 때문입니다. 공부를 하다 보니, Express 의 개발팀이 Koa 라는 웹 프레임워크를 새로 만들었다는 것을 알게 되었습니다. 기존의 Express 의 경우 오픈소스의 소유권이 IBM 계열사로 이전되었으며, 유지보수가 되고 있긴 하지만 장기적으로 보면 문제가 생길 확률이 높습니다. 물론 워낙 유명하여 매우 많은 프로젝트에서 Express 를 사용하고 있지만, Koa 로 마이그레이션 하는 프로젝트도 늘어나는 추세입니다. Express 와의 큰 차이는 Koa 는 훨씬 가볍고, Node.js v7 의 async/await 기능을 아주 편하게 사용할 수 있다는 점입니다. 따라서 앞으로 많은 프로젝트들이 새 프레임워크인 Koa 로 개발 및 변경할 수 있다고 판단하여 Express 에서 Koa 로 변경하였습니다.

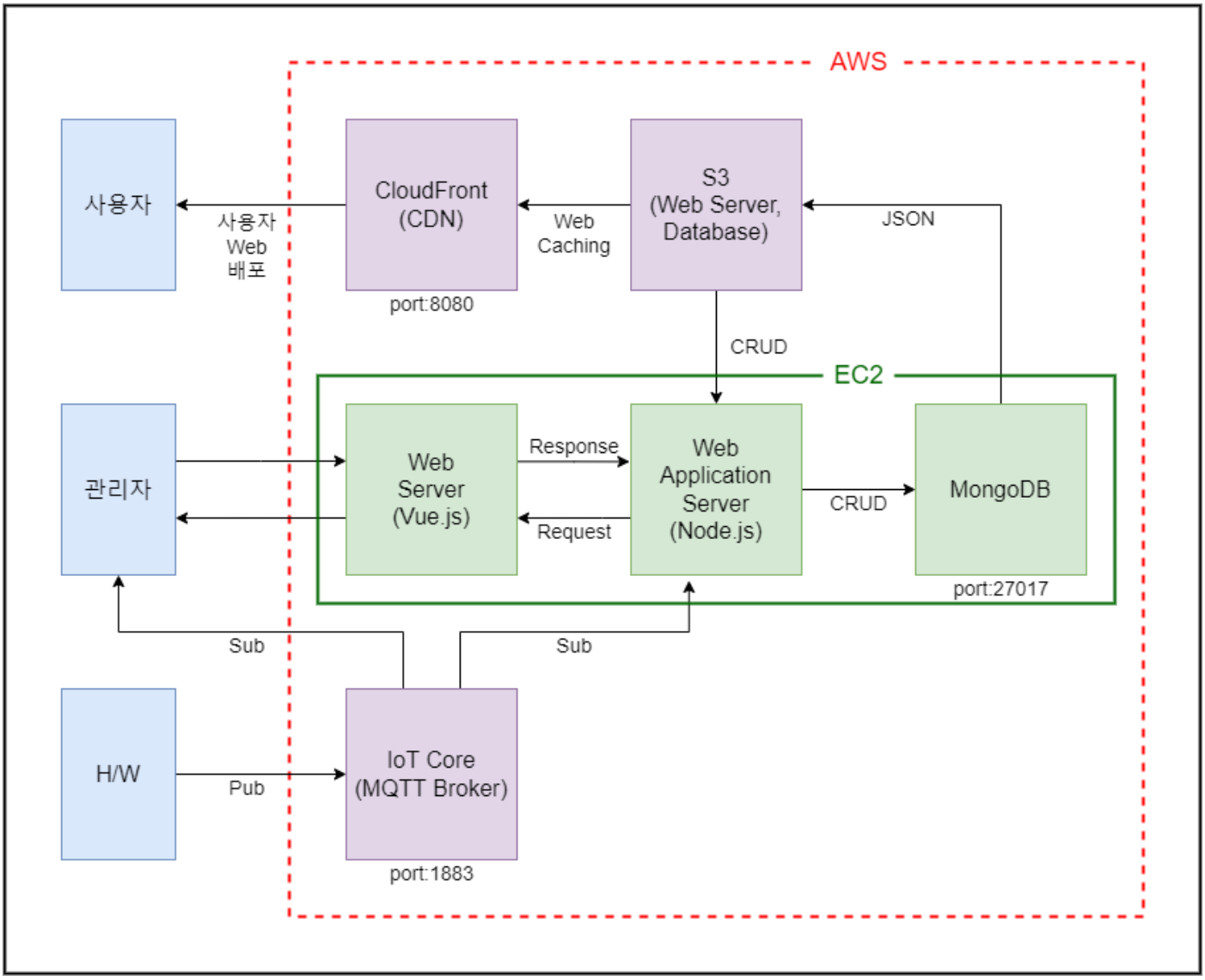
2) Raspberry Pi 에서 MQTT Broker Server 운용

Raspberry Pi 에서 MQTT 를 사용할 때는 Linux 이기 때문에 바로 사용하였지만, AWS 로 서버를 이전한 뒤에는 Linux 에서 Broker 서버를 운용하는 것 보다 AWS IoT Core 서비스를 이용하면 서버와 분리되어 있기 때문에 보안성도 얻을 수 있고, 서버 자원에도 이득이 있다는 것을 알게 되었습니다. 따라서 기존에 Linux 상에서 Mosquitto 를 통해 운용했던 Broker Server 를 IoT Core 에 있는 Broker Server 서비스로 이전하였습니다.

3) S3 와 CloudFront 를 통한 관리자/사용자에게 웹 페이지 배포

vue.js 로 개발된 웹 페이지를 빌드하여 배포가 가능한 형태로 변환하여 S3 에 저장하고, CloudFront 를 이용하여 배포하는 방식으로 설계하였습니다. 현재 사용자 페이지는 위와 같은 방식으로 저장하고 배포하였고 이용할 수 있지만, 관리자 페이지는 해당 파트를 담당했던 팀원이 빌드 에러를 잡지 못하여 EC2 instance 에 있는 Ubuntu 에 Web Application Server 와 관리자 Web Server 가 함께 운용되도록 변경하였습니다.

1. 개요 (전체 구조)



2. 기능

2.1. EC2 instance (Server port:3000)

EC2 instance는 가상 컴퓨터를 대여해주는 서비스이다. 여기에 Ubuntu 20.04 LTS를 설치하고 Node.js와 Koa로 개발한 Web Application Server를 구축했다. 그리고 CORS(Cross-Origin Resource Sharing) 통신으로 들어오고 나가는 데이터(CRUD:Create, Read, Update, Delete)를 MongoDB를 이용하여 S3에 저장할 수 있도록 하였다.

2.2. S3

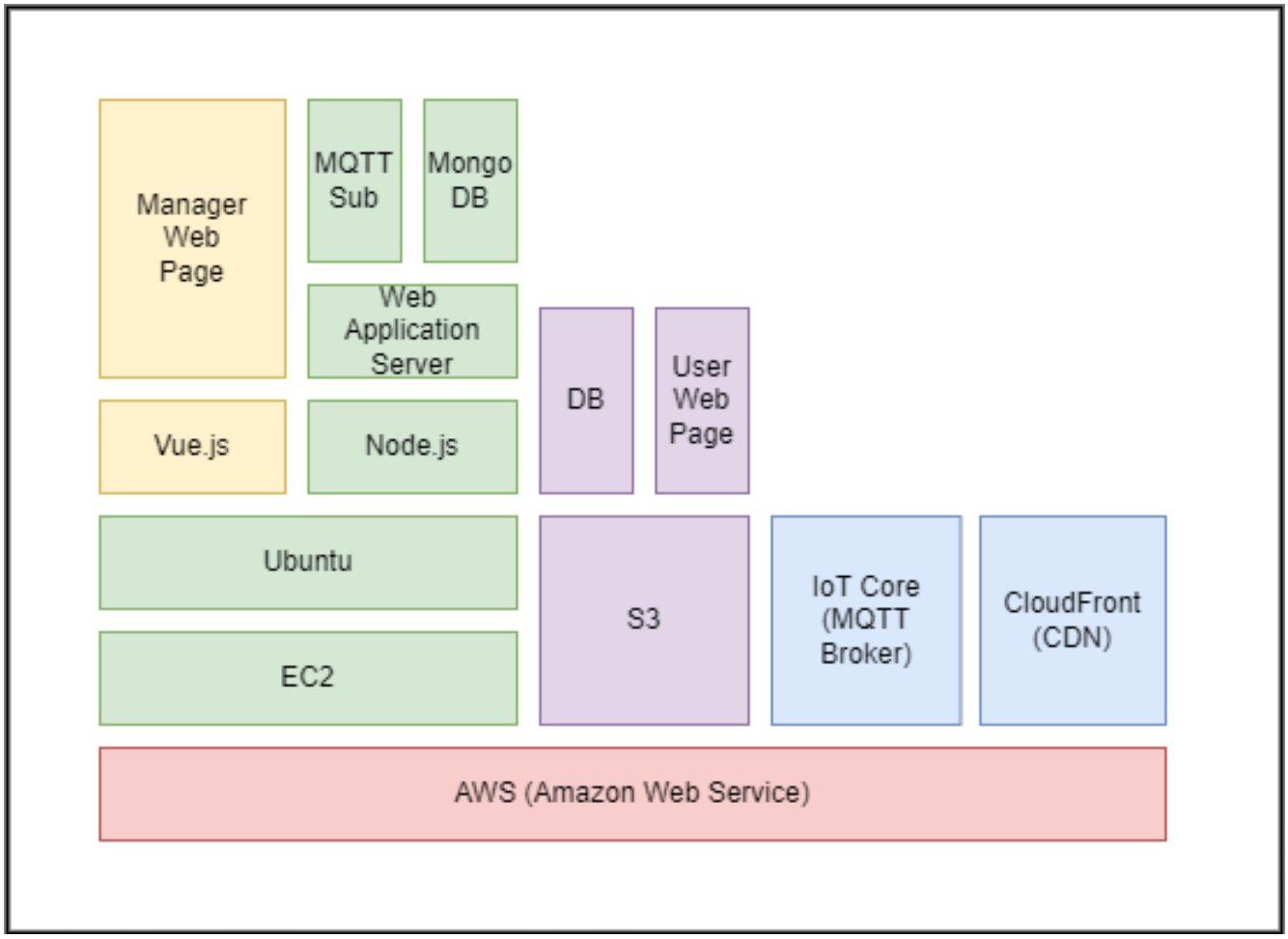
S3는 Storage를 대여해주는 서비스이다. 따라서 EC2와 같이 운영체제를 올릴 수는 없고 단순 파일만 올릴 수 있다. 여기에 Database의 Data를 저장하고, 사용자용 페이지 배포 파일을 담아두었다.

2.3. CloudFront (CDN : Content Delivery Network)

CloudFront는 S3에 있는 정적 웹 페이지를 배포할 수 있는 CDN 서비스이다. CDN이란 메인 서버(S3)에 있는 웹 페이지를 전세계에 위치한 Cache Server에 Content를 저장(캐싱)해 두었다가 사용자가 Content를 요청하면 가장 가까운 곳에 위치한 Cache Server가 응답해주는 기술이다. CloudFront를 쓰는 목적에는 CDN 서비스로 느린 응답속도와 다운로딩 타임을 극복하기 위함도 있지만 HTTPS를 무료로 이용할 수 있다는 점이 커서 사용하였다.

2.4. IoT Core (Broker port:1883)
IoT Core는 AWS에서 지원하는 MQTT 서비스이다. Broker Server로도 사용할 수 있고 Publish, Subscribe도 가능하다. IoT Core로 Broker Server를 만들고, Publish인 H/W에서 지정 Topic으로 보낸 데이터를 Subscribe 하고 있던 WAS에서 데이터를 parse하여 mongoDB를 통해 S3에 저장한다.

3. 설계



3.1. Web Application Server
AWS(Amazon Web Services) EC2 인스턴스에 Ubuntu를 올려서 사용하였고, Node.js와 Koa를 사용하여 HTTP Server와 MQTT Broker를 구현하였다.

3.2. Web Server
Amazon CloudFront 서비스를 이용하여 S3에 저장되어 있는 HTML, CSS, Vue.js로 작성된 Static Web Page를 Client에게 배포한다.

3.3. MQTT
H/W에서 Publish한 Topic을 Broker를 통해 IoT Core로 Subscribe하여 데이터를 받은 후에 JSON 포맷으로 변환하여 S3에 저장하고 Mongo DB로 관리한다.

4. 시험항목

분류	내용	결과
데이터 저장	1) 데이터를 Publish에서 지정 Topic으로 JSON 형태로 메시지 발행 2) 해당 Topic을 IoT Core에서 지원하는 Broker Server를 통해 Subscribe하고 있는 WAS에서 메시지를 수령 3) S3에 저장	데이터가 지정된 테이블에 입력됐는지 상호 비교
데이터의 호출 및 표시	1) 데이터를 Web 에서 get 요청 2) DB 가 담긴 S3 로부터 JSON 형태로 호출 3) Web으로 return	호출한 값과 Web 에 표시된 데이터가 일치하는지 확인
회원가입	1) client 가 회원가입 요청 시 데이터를 post 로 수령 2) 데이터를 쿼리문을 통해 DB 에 insert 3-1) 일치하는 데이터가 없을 경우, 성공 메시지를 return 3-2) key 혹은 id가 중복될 경우, 에러 메시지를 return	web 에서 입력한 데이터, 서버에서 받은 데이터, DB에 저장된 데이터가 일치하는지 확인
로그인	1) client 가 로그인 요청 시 web 에서 get 요청 2) 일치하는 데이터가 있는지 쿼리문을 통해 탐색 3-1) 일치하는 데이터가 있는 경우, 성공 메시지를 return 3-2) 일치하는 데이터가 없는 경우, 에러 메시지를 return 3-3) 비밀번호가 다를 경우, 에러 메시지를 return	요청한 아이디로 로그인 되는지 확인
MQTT	1) MQTT Broker(IoT Core)에서 발행되는 모든 메시지를 저장 2) Publish 에서 지정 Topic 으로 발행되는 메시지를 확인 3) Subscribe에서 메시지가 정상적으로 받아지는지 확인	Pub과 Sub, Broker를 확인하여 데이터가 모두 일치하는지 확인

5. 문제점

1) 높은 트래픽을 고려하지 않은 서버 구조

다중 사용자 및 높은 트래픽을 고려하지 않았기 때문에, Load-balancer와 Nginx 등 트래픽이 올라 갈 경우 트래픽을 분산할 수 있는 기술을 도입하지 않았다. 따라서 실제 서비스로 출시하려면 해당 사항들을 면밀히 고려하여 기능을 추가해야 한다.

2) 보안을 고려하지 않은 설계

아직 초기 개발 단계이기 때문에 데이터나 통신을 암호화하지 않았다. 따라서 보안에 상당히 취약한 상태이다. 물론 AWS를 이용하기 때문에 AWS에서 지원하는 기본적인 보안 정책이나 인증 방식을 일부 이용하고 있기 때문에 단순한 공격에는 안전하지만 아직 보안에 대해 지식이 거의 없기 때문에 공격이 들어온다면 대처하기가 어렵다고 판단된다.

7. 향후 계획

실제로 서비스를 한다고 했을 때 위의 문제점들을 보완하지 않는다면 서버가 멈추는 등의 문제가 발생할 수 있기 때문에 실제 서비스 시에 일어날 수 있는 다양한 문제점들을 고려해보고 구조를 변경하고 보완할 것이다. 또한, 미처 고려하지 못한 예외 상황은 언제든 일어날 수 있기 때문에 실제 서비스를 운영하게 된다면 로그 파일을 분석할 수 있도록 log를 저장하고 분석하며 대처할 것이다. AWS에 서버가 있기에 기본적인 보안이 적용되어 있겠지만 서버 구조 설계가 미흡할 수 있다는 점을 고려하여 서버 구조와 서버 보안, 통신 암호화, 데이터 암호화, AWS에서 지원하는 보안 서비스 등에 대해 공부하여 서버의 안정성과 보안성을 강화할 것이다.