Amazon Web Services 를 이용한
RESTful API 구축과 이를 활용한 고독사 예방 앱

2021년

안양대학교 스마트창의융합대학
컴퓨터공학전공
전 재 욱

지 도 교 수 강 민 섭

이 논문을 공학 학사 학위논문으로 제출함.

2021년 12월

안양대학교 스마트창의융합대학

컴퓨터공학과

전 재 욱

논 문 인 준 서

이 논문을 전재욱의 공학 학사 학위논문으로 인준함.

2021년 12월

주 심 박성순 인

부 심 이상홍 인

부 심 강민섭 인

차 례

요약 1

제 1장 서론 2

제 1절 연구배경 2

제 2절 연구목적 3

제 2장 관련 연구 4

제 1절 서버리스(Serverless) 아키텍처 4

제 2절 Amazon Web Service 와 FaaS 7

제 3절 Amazon RDS 8

제 4절 Amazon Lambda 11

제 5절 Amazon API Gateway 와 RESTful API 12

제 6절 모자일 앱에서 RESTful API 요청 13

제 3장 설계 14

제 1절 아키텍처 설계 14

제 2절 RDS를 사용한 데이터베이스 구축 14

제 3절 Lambda 를 사용한 서버리스 아키텍처 설계 15

제 4절 API Gateway 를 사용한 RESTful API 아키텍처 설계 16

제 5절 안드로이드 모바일 앱에서의 REST 요청 구성 17

제 4장 시스템의 구현 18

제 1절 서버리스 컴퓨팅 구현 18

제 2절 RESTful API 구현 23

제 3절 구현 결과 25

제 5장 기대효과 및 결론 27

제 1절 결론 27

제 2절 향후 과제 ····· 27
[참고 문헌] ····· 28

표 및 그림 차례

표 차례

표 1. 서버리스 아키텍처의 구현 방식 ····· 5
표 2. SQL 과 NoSQL 비교 ····· 10
표 3. 대표적인 Method 종류 ····· 13
표 4. CRUD 요청과 기능 ····· 15

그림 차례

그림 1. 전통적인 모놀리식 아키텍처(위)와 서버리스 아키텍처(아래) ····· 4
그림 2. BaaS 와 FaaS ····· 5
그림 3. 서버리스 아키텍처의 라이프 사이클(Faas) ····· 6
그림 4. Amazon Web Services에서 제공하는 서버리스 아키텍처 ····· 7
그림 5. Amazon Web Services에서 제공하는 DB 종류와 활용 사례 ····· 8
그림 6. NoSQL 기반의 DynamoDB 와 SQL 기반의 RDS ····· 8
그림 7. 관계형 데이터베이스에서 데이터가 저장되는 방식 ····· 9
그림 8. 비관계형 데이터베이스에서 데이터가 저장되는 방식 ····· 9
그림 9. AWS Lambda 를 사용한 서버리스 아키텍처 ····· 11
그림 10. RESTful API 의 동작 방식 ····· 12
그림 11. Amazon API Gateway 의 아키텍처 ····· 12
그림 12. 모바일 앱을 위한 서버리스 아키텍처 ····· 13
그림 13. 중앙 서비스 구성을 위한 AWS ····· 14
그림 14. Amazon RDS에서 제공하는 DB 종류 ····· 14
그림 15. HTTP Method와 URI 별 기능 실행도 ····· 16
그림 16. 클라이언트의 요청과 수행 결과 통신 방식 ····· 16

그림 17. JAVA에서 Http 통신을 진행하는 과정 ····· 17
그림 18. AWS 관리 콘솔 ····· 18
그림 19. RDS를 사용해 MySQL 데이터베이스 구축 ····· 18
그림 20. RDS 인스턴스 생성 ····· 19
그림 21. 외부 접속 허용 방법 ····· 19
그림 22. 생성한 인스턴스의 대시보드 ····· 20
그림 23. 테스트 스키마 생성 ····· 20
그림 24. Lambda 함수 생성 ····· 21
그림 25. Lambda 의 테스트 이벤트 기능 ····· 22
그림 26. Lambda 함수 테스트 결과 ····· 22
그림 27. CRUD 에 대한 Lambda 함수 ····· 22
그림 28. API Gateway 로 구현 가능한 API 종류 ····· 23
그림 29. 리소스 생성 ····· 23
그림 30. Lambda 함수에 대해 CRUD 매핑 ····· 24
그림 31. API 배포 ····· 24
그림 32. HTTP 요청을 전송하는 메서드 ····· 25
그림 33. HTTP POST 요청 예시 ····· 25
그림 34. RESTful API를 활용한 모바일 앱 예시 ····· 26

요 약

본 논문은 'IOT를 활용한 독거노인/장애인 돌봄 서비스 앱'으로 프로젝트를 진행하면서 고안된 논문이며, IOT에서 수집한 데이터를 효율적으로 관리하고 안정적으로 최종 사용자(End user)에게 제공하는 방안에 대해 고안하였다.

서버는 AWS(Amazon Web Service)를 사용해 적은 비용으로 별도의 관리 없이 운영이 가능한 FaaS 형태의 서버리스(Serverless) 아키텍처로 구성하였고 데이터 관리에는 AWS의 RDS(Relational Database Service)를 이용하여 MySql DBMS(Database Management System)를 사용하였다.

사용자가 DB(Database)와 상호작용하는 방식으론 RESTful API를 사용하였고 RESTful API 아키텍처 구성에는 AWS의 Lambda 와 API Gateway 서비스를 이용하였으며 최종 사용자에게 데이터를 시각화하고 기타 편리성을 제공하기 위해 안드로이드 기반의 모바일 앱을 개발하였다.

본 논문에서는 모바일 앱의 상세 개발에 관해서는 기술하지 않고 AWS를 사용한 서버리스 RESTful API 아키텍처 구성과 모바일 앱에서 미리 구성한 RESTful API를 사용하여 CRUD(Create, Read, Update, Delete) 요청을 통해 DB를 조작하는 방법에 관해서 기술한다.

제 1장 서 론

제 1절 연구배경

‘고독사’, 혹은 ‘고립사’라 불리는 죽음은 「고독사 예방 및 관리에 관한 법률」에 의하면 고독사는 “가족, 친척 등 주변 사람들과 단절된 채 홀로 사는 사람이 자살・병사 등으로 혼자 임종을 맞고, 시신이 일정한 시간이 흐른 뒤에 발견되는 죽음”을 의미하는데 그 대상은 주로 홀로 지내는 독거노인이나 장애인이다.

이러한 고독사를 예방하기 위한 정책은 이전부터 지속해서 시행됐으나 20년 초부터 확산한 코로나 19는 현장 중심의 예방정책을 무력하게 만들었다. 해마다 증가하는 독거노인 수에 맞춰 앞으로 고독사를 예방하기 위해서는 새로운 시스템이 필요하다고 판단하였다.

기존 예방정책은 대부분 등록된 독거노인의 거주지에 사회복지사가 일정 기간마다 방문하는 시스템이다. 하지만 코로나 19의 확산으로 이러한 현장 방문 방식은 어려운 실정이고 독거노인 수에 비해 사회복지사의 수 매우 부족한 수준이다. 따라서 직접 방문하지 않고도 한 번에 여러 대상의 상태를 확인할 수 있으며 위급 상황으로 판단되면 이를 알리고 조치를 취할 수 있는 언택트 방식의 안부 확인서비스를 기획하였다.

프로젝트의 결과물을 통해 비대면 안부확인서비스의 활성화와 사회적 보호가 절실한 독거노인 및 장애인을 위한 고독사 예방 정책의 보조적 기술로써 활용될 수 있다고 기대된다.

제 2절 연구목적

프로젝트는 각종 IoT 기기(문 열림 센서 등)을 통해 수집한 데이터를 **DB(Database)**에 적재하고 모바일 앱을 통해 이를 조작하는 시스템이지만 본 논문에서는 수집한 데이터를 효율적으로 관리하기 위한 **서버리스(Serverless)** 아키텍처와 안정적으로 데이터베이스를 조작할 수 있도록 **RESTful API** 아키텍처를 구성하는 방법 및 안드로이드 기반의 모바일 앱을 사용해 실제로 구축한 RESTful API 의 **CRUD** 요청을 보내고 이에 대한 동작 검증을 기술한다.

따라서 본 논문의 연구 목적은 다음과 같다.

AWS의 RDS 를 사용한 관계형 데이터베이스 관리 시스템의 구축을 진행하고 Lambda 서비스를 이용하여 REST CRUD(Create, Read, Update, Delete) 서버리스 요청 처리 시스템(FaaS) 구축 연구,

API Gateway 서비스를 이용하여 REST 요청에 필요한 URI(Uniform Resource Locator), 요청 방식(Method), 데이터 전송 형태(Representation of Resource)를 REST 규칙에 맞게 정의하고 이를 배포하는 RESTful API 구성 방안 연구,

안드로이드 기반의 모바일 앱을 통해 RESTful API를 사용하는 방법과 이를 통한 구성 검증 방법 연구,

최종적으로 Amazon Web Services를 이용하여 RESTful API 아키텍처 구조를 갖는 서버리스 컴퓨팅을 구축하고 이를 여러 플랫폼에서 사용할 수 있게 된다.

제 2장 관련 연구

제 1절 서버리스(Serverless) 아키텍처

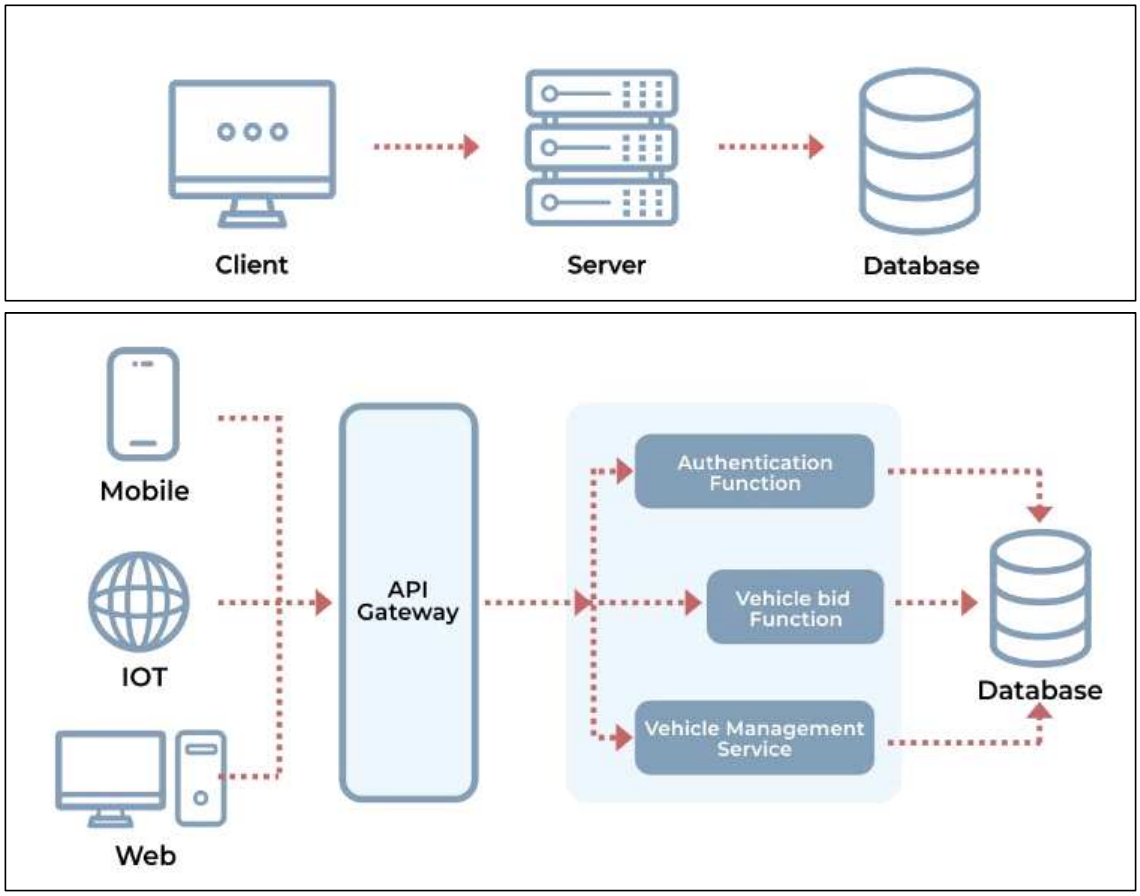

그림 1. 전통적인 모놀리식 아키텍처(위)와 서버리스 아키텍처(아래)

서버리스(Serverless)는 개발자가 서버를 관리할 필요 없이 애플리케이션을 빌드하고 실행할 수 있도록 하는 클라우드 네이티브 개발 모델이다.

서버리스 아키텍처에도 서버가 존재하긴 하지만, 애플리케이션 개발에서와 달리 추상화되어 있다. 클라우드 제공업체가 서버 인프라에 대한 프로비저닝, 유지 관리, 스케일링 등의 일상적인 작업을 처리하며, 개발자는 배포를 위해 코드를 컨테이너에 패키징하기만 하면 되기에 서버 시스템 관리가 필요 없다. 서버리스 애플리케이션은 배포되고 나면 필요에 따라 자동으로 스케일 업되거나 스케일

다운이 이루어지므로 서버리스 기능이 유휴 상태일 때는 아무런 비용도 들지 않기에 적은 비용으로 서비스 구축이 가능하다. 또한 런타임이 분리되어 있어 보안이 강력하고 서버 확장이 간편하여 확장성이 뛰어나다는 특징이 있다.

서버리스 아키텍처의 널리 알려진 두 가지 주요 구현 방식은 다음과 같다.

FaaS **(Fundation as a Service)**	무상태(Stateless) 함수가 서버 측 비즈니스 로직을 포함한다. 이 함수는 독립 컨테이너에서 실행되며 이벤트로 트리거 되며 AWS Lambda 또는 Azure Functions과 같은 클라우드 제공업체가 전부 관리한다.
BaaS **(Backend as Service)**	클라우드 서비스를 로직 처리 단계에서 사용한다. Baas를 사용하는 응용 프로그램은 일반적으로 SPA 또는 모바일 응용 프로그램과 같은 클라이언트 중심의 응용 프로그램이다.

표 1. 서버리스 아키텍처의 구현 방식

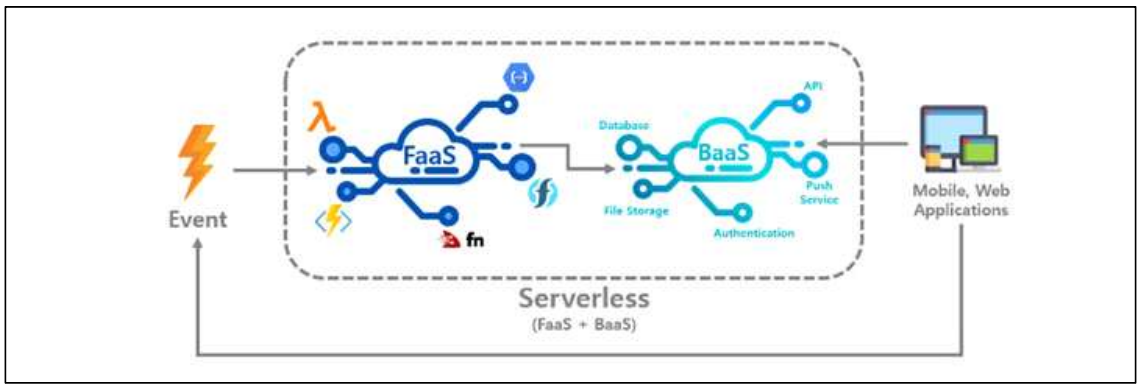

그림 2. BaaS 와 FaaS

요약하자면 BaaS는 어플리케이션 개발 시 요구되는 복잡한 백앤드(Back-End) 기능들을 사용자가 직접 개발하지 않고 클라우드 공급자가 제공하는 서비스를 이용해 쉽고 안정적으로 구현하는 방식이며 FaaS는 사용자가 쓸 기능을 함수 단위로 나누어 구현하고 이를 서비스하는 방식이다.

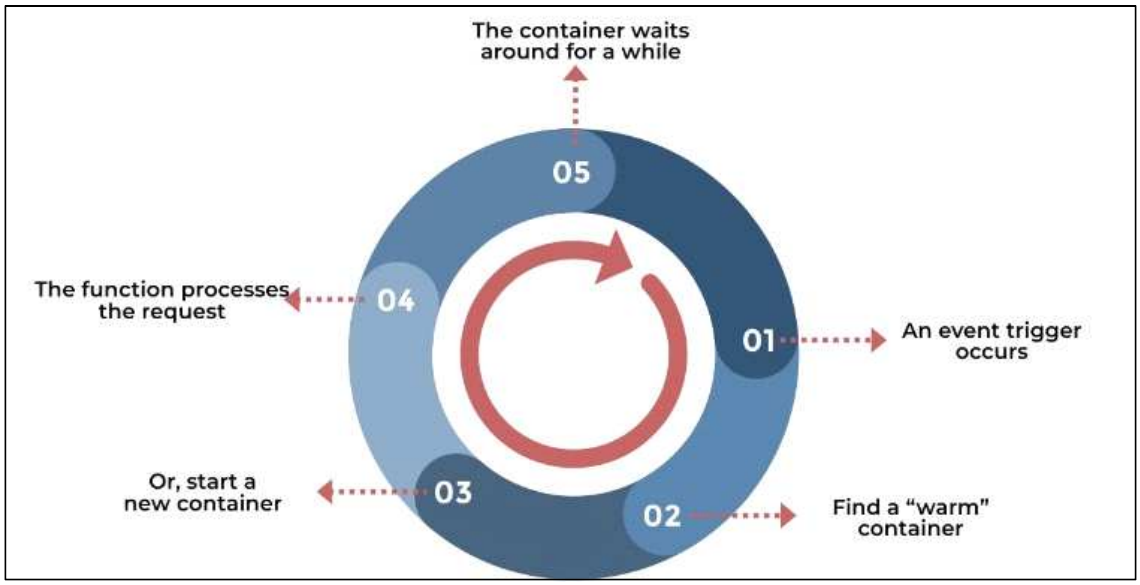

그림 3. 서버리스 아키텍처의 라이프 사이클(Faas)

FaaS 와 BaaS 형태 중 더 많이 사용되는 것은 FaaS 형태이다. FaaS 형태의 서버리스 아키텍처에서 개발자는 로직이 담긴 함수 구현만 신경쓰면 되기 때문에 인프라 관리에서 해방된다. FaaS 제공업체에서는 사용자가 원하는 로직을 구성해둔 함수에 대해 호출 요청을 할 수 있도록 API Gateway를 통해 처리하고 있어 특정 함수 호출 요청을 받으면 컨테이너를 실행하고 해당 런타임 내에서 정의해놓은 함수가 실행된 뒤 다시 컨테이너를 종료한다. 이러한 함수는 서버가 계속 대기하면서 사용자의 요청을 처리하는 것이 아니라, 이벤트가 있을 때마다 실행되기 때문에 첫 번째 이벤트의 처리 시 시작 시간이 필요하다.

앞서 서술한 특징 때문에 서버리스 아키텍처는 별도의 WAS(Web Application Server) 서버를 구성하지 않고도 독립적으로 동작하는 응용프로그램을 생성할 수 있기에 즉시 시작할 수 있는 비동기식, 무상태(Stateless) 애플리케이션이나 수신 데이터 스트림, 채팅 봇, 예정된 태스크, 비즈니스 로직에 적합하여 일반적으로 백엔드 API, 웹 애플리케이션, 비즈니스 프로세스 자동화, 서버리스 웹 사이트 등에 활용되고 있다.

제 2절 Amazon Web Services 와 FaaS 아키텍처

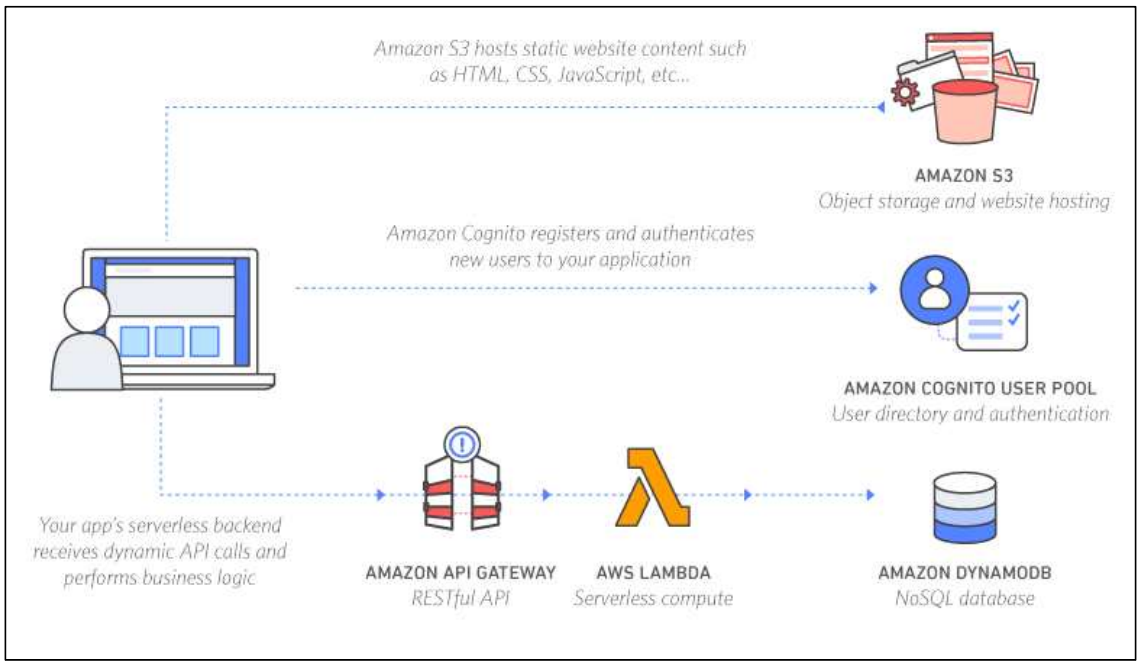

그림 4. Amazon Web Services에서 제공하는 서버리스 애플리케이션 아키텍처

대표적인 FaaS 는 Amazon Web Services 에서 제공하는 AWS Lambda 로 추가적인 서비스와 함께 사용하여 서버리스 아키텍처를 구성할 수 있다.

기본적으로 FaaS 형태의 아키텍처에서는 동일한 클라우드 제공업체의 API Gateway 서비스를 이용해 미리 지정해둔 함수를 실행하는 방식을 사용한다. API Gateway를 사용하여 RESTful API를 구축하면 특정 플랫폼에 국한되지 않는 시스템 구축이 가능하며 AWS Lambda를 통해 간편하고 안정적인 결과를 도출할 수 있을 뿐만 아니라 최종적으로는 데이터베이스 조작까지 가능하다.

Amazon 이외에도 마이크로소프트(Microsoft)의 Azure Functions, 구글(Google)의 Google Cloud Functions 등 서버리스 아키텍처를 구성할 수 있는 플랫폼이 다양하게 존재하지만 본 논문에서는 Amazon Web Services를 사용한다.

제 3절 Amazon RDS(Relational Database Service)

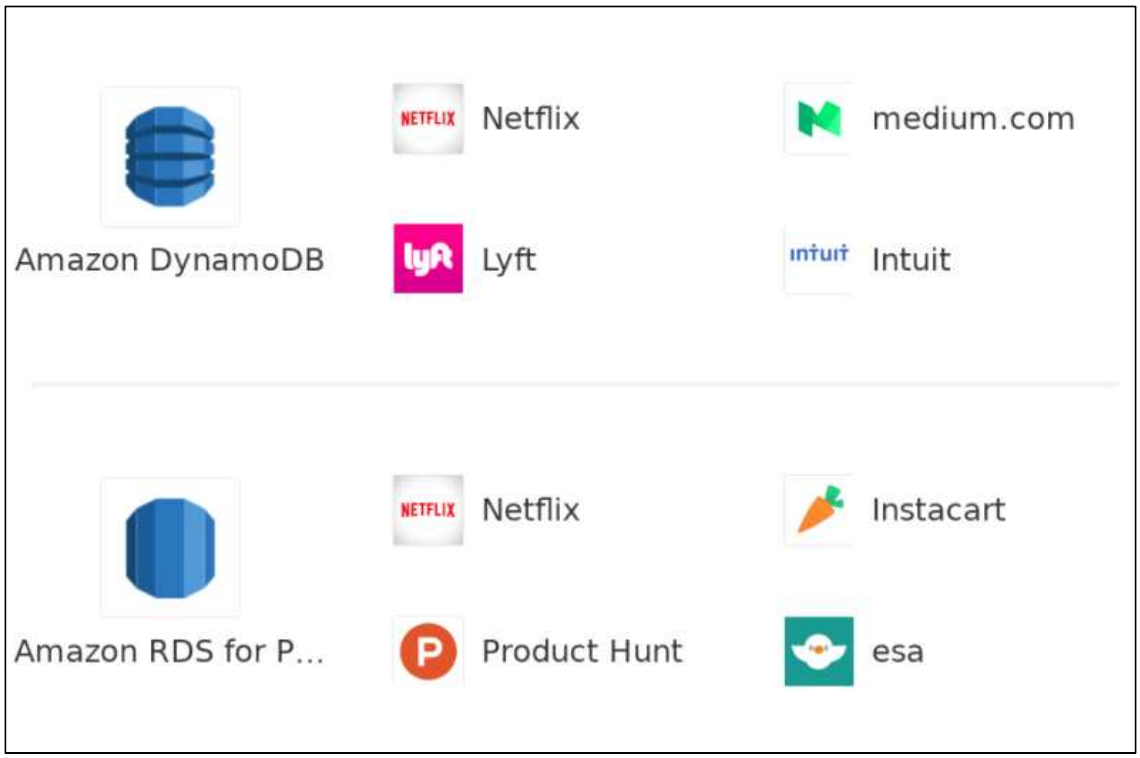

그림 5. Amazon Web Services에서 제공하는 DB 종류와 활용 사례

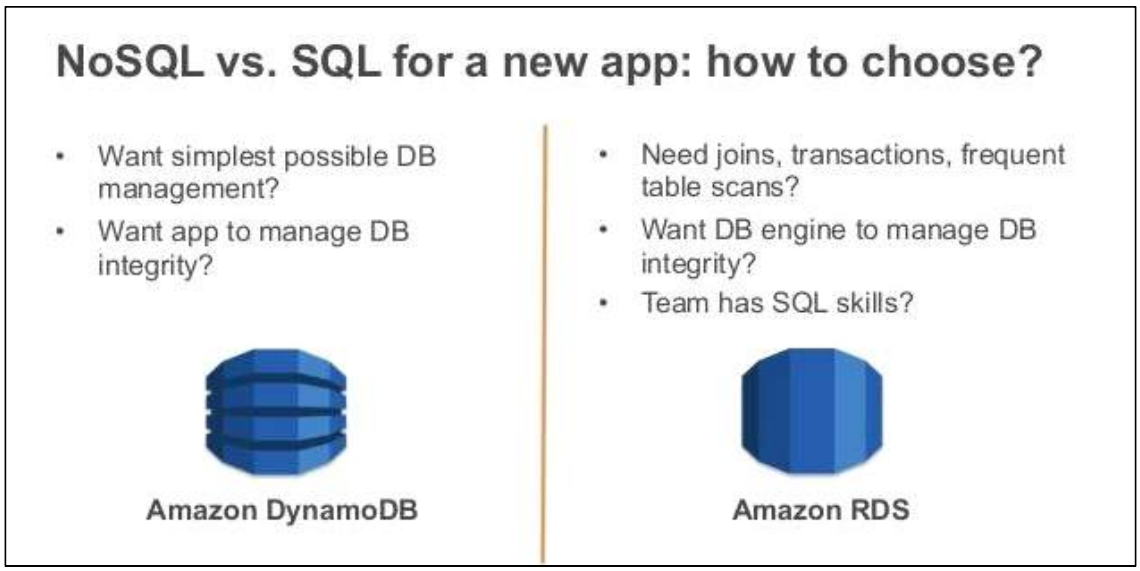

그림 6. NoSQL 기반의 DynamoDB 와 SQL 기반의 RDS

AWS에서 제공하는 데이터베이스 서비스의 종류는 [그림 5]와 같이 DynamoDB 와 RDS로 두 종류가 존재한다. DynamoDB 는 NoSQL(비관계형 데이터베이스) 기반의 서비스로 mongo DB, Redis 등이 이에 속한다. 반면 RDS는 SQL(관계형 데이터베이스) 기반의 서비스로 MySQL, PostgreSQL, MariaDB 등이 존재한다.

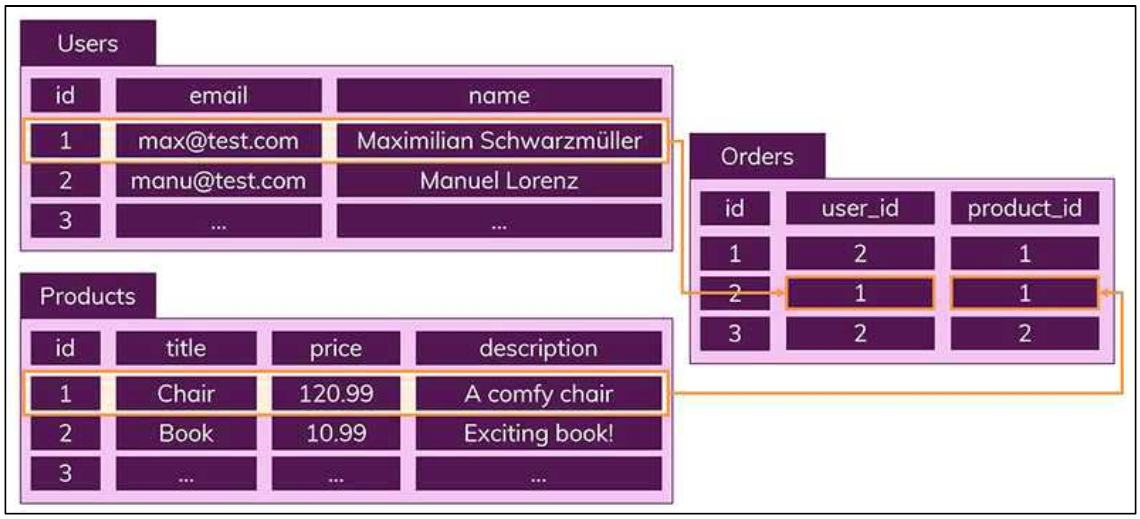

그림 7. 관계형 데이터베이스에서 데이터가 저장되는 방식

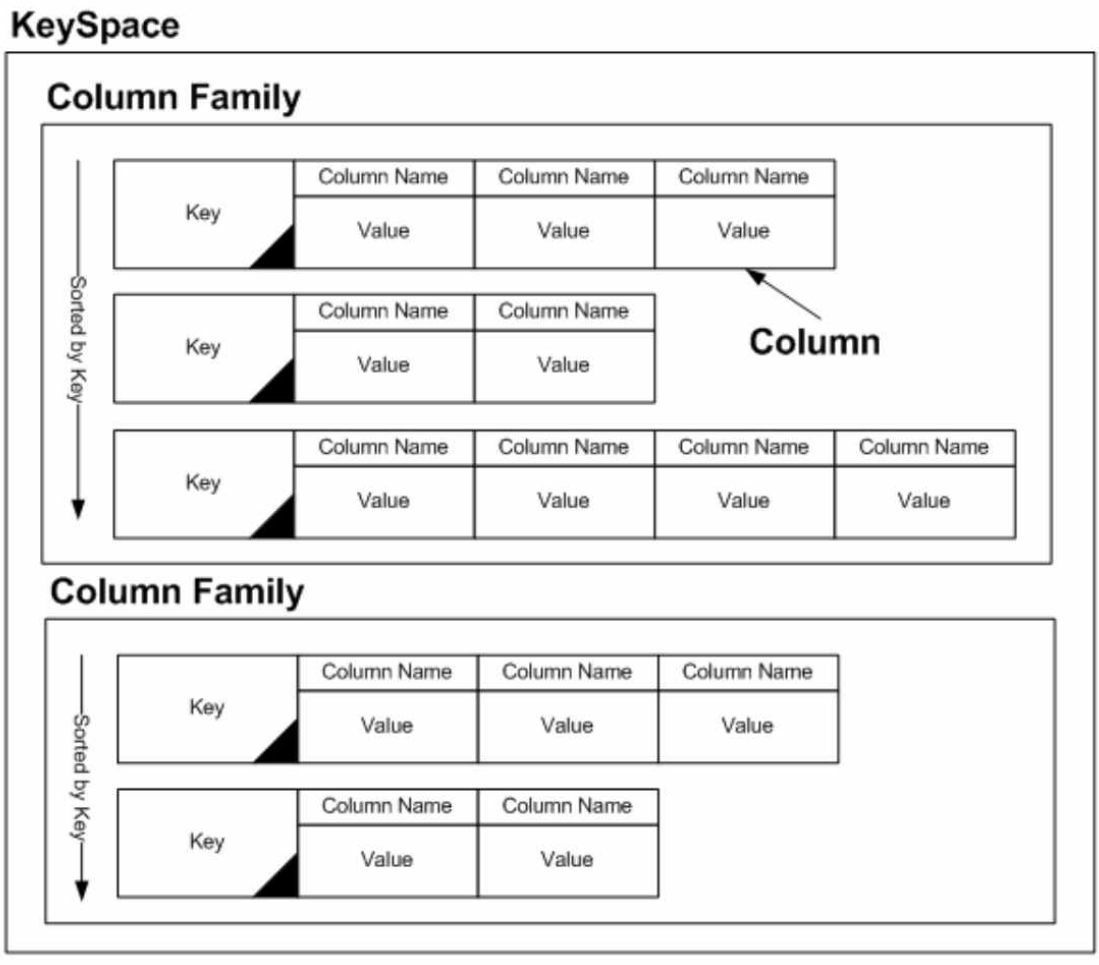

그림 8. 비관계형 데이터베이스에서 데이터가 저장되는 방식

SQL 데이터베이스는 정해진 데이터 스키마에 따라 테이블에 저장하고 관계를 통해 여러 테이블에 분산되는 방식이다. 데이터는 테이블에 레코드로 저장되는데, 각 테이블마다

명확하게 정의된 구조가 있으며 해당 구조는 필드의 이름과 데이터 유형으로 정의된다. 따라서 스키마를 준수하지 않은 레코드는 테이블에 추가할 수 없다. 또한 하나의 테이블에서 중복 없이 하나의 데이터만을 관리하기 때문에 다른 테이블에서 부정확한 데이터를 다룰 위험이 없어지는 장점이 있다.

NoSQL 데이터베이스는 레코드를 문서(Documents)라고 칭한다. NoSQL 은 SQL과 핵심적인 차이가 있는데, SQL은 정해진 스키마를 따르지 않으면 데이터 추가가 불가능한 반면 NoSQL에서는 다른 구조의 데이터를 같은 컬렉션에 추가할 수 있다. 문서는 Json(JavaScript Object Notation)과 비슷한 Key-Value 구성을 갖는데 데이터를 SQL처럼 여러 테이블에 나눠 담지 않고 동일한 컬렉션(Collection)에 넣는다. 따라서 여러 테이블에 조인(Join)할 필요 없이 모든 데이터를 담고 있는 문서를 작성하는 게 NoSQL 방식이며 조인(Join)을 잘 사용하지 않고 자주 변경되지 않는 데이터를 사용할 때 효율적이다.

DB 종류	장점	단점	비고
SQL (관계형 데이터베이스)	• 데이터 무결성 보장 • 명확한 스키마	• 스키마 수정이 어려워 유연성 낮음 • 쿼리에 대한 지식 필요	스키마가 명확하여 수정 여지가 없고 데이터가 자주 변경되는 경우에 효율적
NoSQL (비관계형 데이터베이스)	• 데이터 Read 속도 향상 • 스키마가 없어 뛰어난 유연성	• 데이터 중복 업데이트 필요 • 데이터 수정이 어려움	정확한 데이터 구조를 알 수 없거나 데이터 변경이 자주 없는 경우에 효율적

표 2. SQL 과 NoSQL 비교

본 논문을 기술하기 위한 연구 프로젝트에서는 데이터의 형식이 명확한 점과 스키마 설계 및 쿼리에 대한 배경 지식을 숙지하고 있던 점, 사용자 및 등록 기기의 프로필 데이터의 변경이 가능하다는 점을 고려하여 SQL 데이터베이스 구축이 가능한 AWS RDS 서비스를 사용하였다.

제 4절 Amazon Lambda

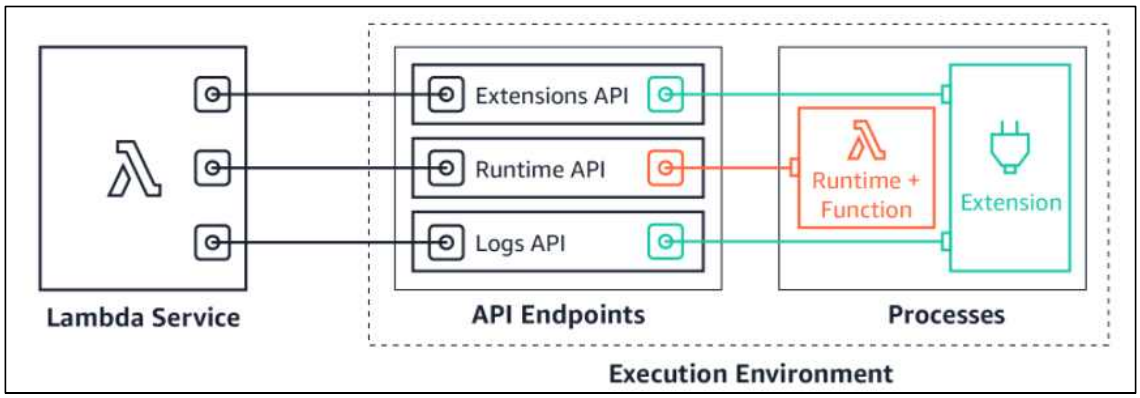

그림 9. AWS Lambda 를 사용한 서버리스 아키텍처

Amazon Web Services 의 AWS Lambda 서비스는 서버리스 아키텍처를 구성하는데 있어 가장 핵심요소로 서버를 프로비저닝 또는 관리하지 않고도 실제로 모든 유형의 애플리케이션 또는 백엔드 서비스에 대한 코드를 실행할 수 있는 이벤트 중심의 서버리스 컴퓨팅 서비스이다.

Lambda 서비스는 요청 수와 요청을 처리하는 함수 시간에 따라서 요금이 발생하기 때문에 서비스의 수요가 적을 때에는 그만큼 비용이 발생하지 않아 비용효율적인 서비스 운용이 가능하며 Lambda 가 지원하는 언어 중 하나로 코드를 공급하기만 하면 거의 모든 유형의 애플리케이션 또는 백엔드 서비스에 대한 코드를 실행할 수 있다. 또한 조건 없이 오토스케일링 되어 별도의 관리를 필요로 하지 않아 운영관리에 대한 부담이 줄어든다는 장점을 갖고 있다.

다만 AWS Lambda 는 최대 10GB의 메모리와 최대 900초의 처리시간으로 리소스가 제한된다는 점과 함수가 호출될 때 새로운 컨테이너를 생성하는 방식이기 때문의 데이터베이스 커넥션 등 별도의 상태를 저장하지 못하는 무상태(Stateless) 특성을 갖는다는 점, 장시간 사용하지 않고 실행하게 되면 딜레이가 생기는 콜드스타트(ColdStart)가 발생한다는 단점이 존재한다.

본 논문에서 진행할 프로젝트는 많은 리소스를 필요로 하지 않으며 게임과 같은 실시간 서비스를 요구하지 않기에 AWS Lambda 서비스의 사용이 적합하다.

제 5절 Amazon API Gateway 와 RESTful API

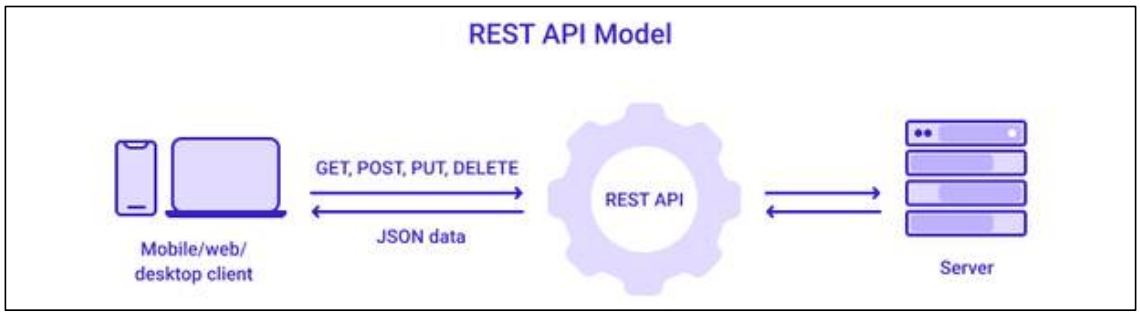

그림 10. RESTful API 의 동작 방식

RESTful API 란 REST(Representational State Transfer) 라고도 불리며 로이 필딩이 소개한 웹의 장점을 최대한 활용할 수 있는 아키텍처이다. REST 는 HTTP 통신에서 특정 자원에 대하여 CRUD(Create, Read, Update, Delete) 연산 수행을 위해 URI 로 요청을 보내는 방식으로 이러한 REST 아키텍처의 제약 조건을 준수하는 API(Application Programming Interface)를 RESTful API 라고 지칭한다.

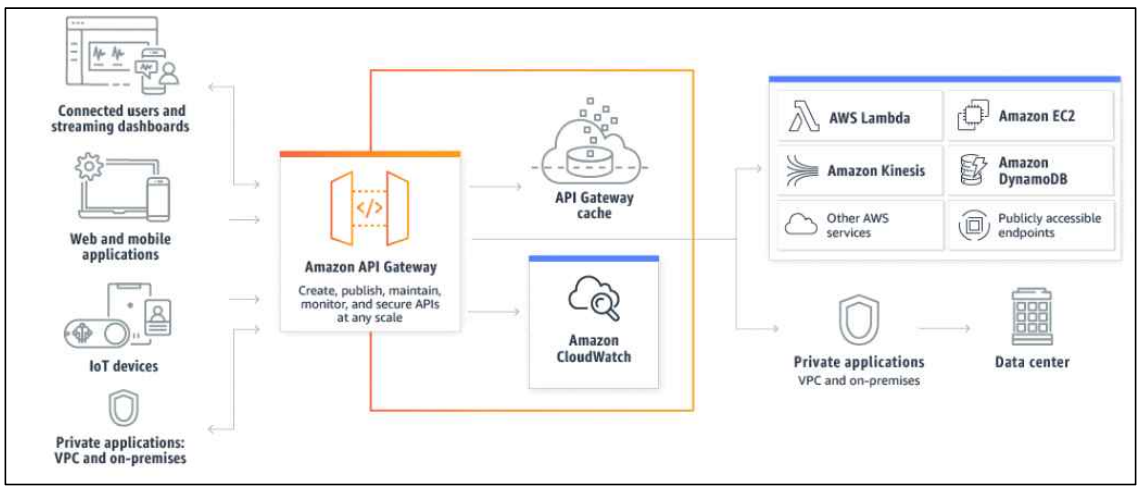

그림 11. Amazon API Gateway 의 아키텍처

AWS에서 RESTful API를 설계하기 위한 핵심요소는 Amazon API Gateway 이다. API Gateway 는 규모와 관계없이 REST 및 WebSocket API를 생성, 게시, 유지, 모니터링 및 보호하기 위한 서비스로 해당 서비스를 사용해 API 개발자는 AWS 또는 다른 웹 서비스를 비롯해 AWS 클라우드에 저장된 데이터에 액세스하는 API를 생성할 수 있다.

본 논문에서는 API Gateway 표준 HTTP 메서드(예: GET, POST, PUT, PATCH, DELETE)를 구현하여 클라이언트-서버의 REST 통신을 구성하고자 한다.

제 6절 모바일 앱에서 RESTful API 요청

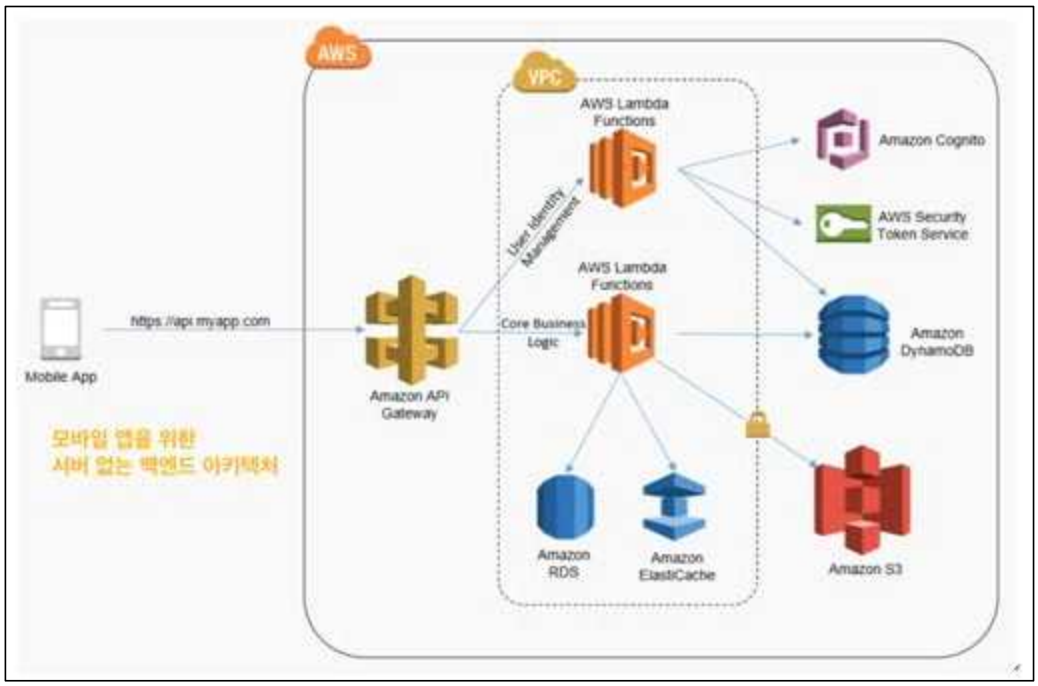

그림 12. 모바일 앱을 위한 서버리스 아키텍처

[그림 12] 는 모바일 앱에서 RESTful API 아키텍처로 구성된 서버리스 서비스에 접근하는 예시이다. RESTful API 아키텍처를 올바르게 준수하였다면 모바일 앱 이외에 웹 브라우저, IoT기기 등에서도 HTTP 프로토콜을 사용해 중앙 서비스에 접근할 수 있다.

METHOD	역할
POST	특정 리소스를 생성한다.
GET	리소스를 조회하고 해당 도큐먼트에 대한 자세한 정보를 가져온다.
PUT	리소스를 수정한다.
DELETE	리소스를 삭제한다.

표 3. 대표적인 Method 종류

HTTP 프로토콜을 이용해서 서버에 요청 타입을 함께 전송하기 위해서 메서드(Method)를 사용하며 RESTful API 에서는 해당 메서드에 따라 각기 다른 작업을 수행한다. 본 프로젝트에서는 이러한 메서드의 특징에 따라 REST 요청을 보낼 것이다.

제 3장 설계

제 1절 아키텍처 설계

그림 13. 중앙 서비스 구성을 위한 AWS

데이터를 관리하고 사용자의 요청을 처리하기 위한 중앙 서비스 구성에는 AWS(Amazon Web Services)를 사용하며 추가적으로 Lambda 서비스를 이용해 서버리스(Serverless) 아키텍처를, API Gateway 서비스를 이용해 RESTful API 아키텍처를 구성한다.

제 2절 RDS를 사용한 데이터베이스 구축

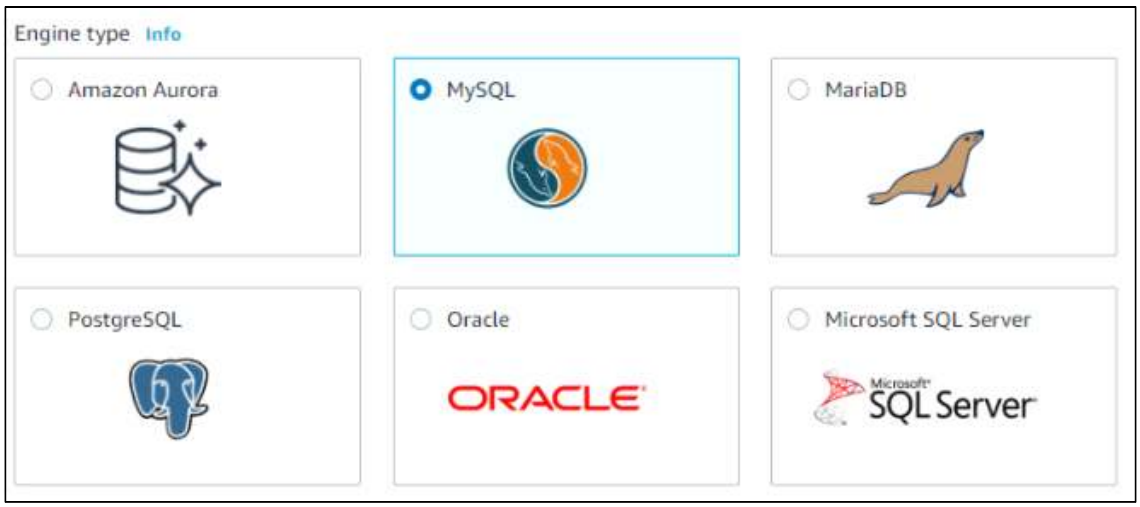

그림 14. Amazon RDS에서 제공하는 DB 종류

RDS(Relational Database Service)를 사용하여 IOT기기로 수집한 데이터를 관리할 수 있도록 데이터베이스를 구축한다. RDS는 관계형 데이터베이스로 PostgreSQL, MySQL, MariaDB 등의 DBMS(Database Management System)를 제공한다. 본 프로젝트에서는 간단하게 데이터 편집을 할 수 있도록 Workbench를 함께 제공하는 MySQL을 데이터베이스로 사용한다.

MySQL을 사용해 데이터베이스와 관리 시스템을 구성한 뒤 간단한 스키마를 하나 생성하고 해당 스키마에 테스트 데이터를 삽입하기로 한다.

제 3절 Lambda를 사용한 서버리스 아키텍처 설계

AWS에서 제공하는 FaaS 서비스인 Lambda를 사용해 서버리스 아키텍처를 설계한다. Lambda 함수를 작성하고 이를 호출하여 앞서 구성한 MySQL 데이터베이스를 조작한다. 함수 작성 언어로는 Python 언어를 사용하며 MySQL 데이터베이스와 커넥션을 생성하기 위해 pymysql 라이브러리를 사용한다. 다만 해당 라이브러리는 기본적으로 제공되는 것이 아니기에 Lambda에서 사용하고자 직접 Lambda 함수 경로에 pymysql 라이브러리를 추가한다.

Lambda 는 API Gateway 와 연동하여 RESTful API 구축에 사용될 예정이므로 CRUD(Create, Read, Update, Delete) 동작을 수행하는 테스트 함수 4개를 각각 작성한다.

요청 타입	기능
CREATE	데이터베이스에 새로운 데이터를 추가하고 결과를 반환
READ	데이터베이스에서 데이터를 조회하여 해당 데이터 정보를 반환
UPDATE	데이터베이스에서 데이터를 수정한 뒤 결과를 반환
DELETE	데이터베이스에서 데이터를 삭제한 뒤 결과를 반환

표 4. CRUD 요청과 기능

요청 결과는 미리 정의한 문자열을 반환하여 클라이언트에서 이를 비교, 확인한다.

제 4절 API Gateway를 사용한 RESTful API 아키텍처 설계

Resource	POST create	GET read	PUT update	DELETE delete
/dogs	create a new dog	list dogs	bulk update dogs	delete all dogs
/dogs/1234	error	show Bo	if exists update Bo if not error	delete Bo

그림 15. HTTP Method와 URI 별 기능 실행도

AWS에서 제공하는 API Gateway 서비스는 기본적으로 들어온 요청을 지정된 곳으로 리다이렉트(Redirect) 해주는 역할이지만 RESTful API 구성에 사용한다고 할 시 POST, GET, PUT, DELETE 로 요청된 리소스(Resource)를 일반적인 http 주소로 전달하는 것이 아니라 Lambda 함수가 실행되도록 하는 역할이 된다.

API Gateway 는 리소스를 생성하고 해당 리소스에 1개 이상의 Method를 등록할 수 있으며 클라이언트는 생성된 리소스 URI 에 HTTP Method를 정의하고 요청을 진행해 원하는 결과를 도출한다.

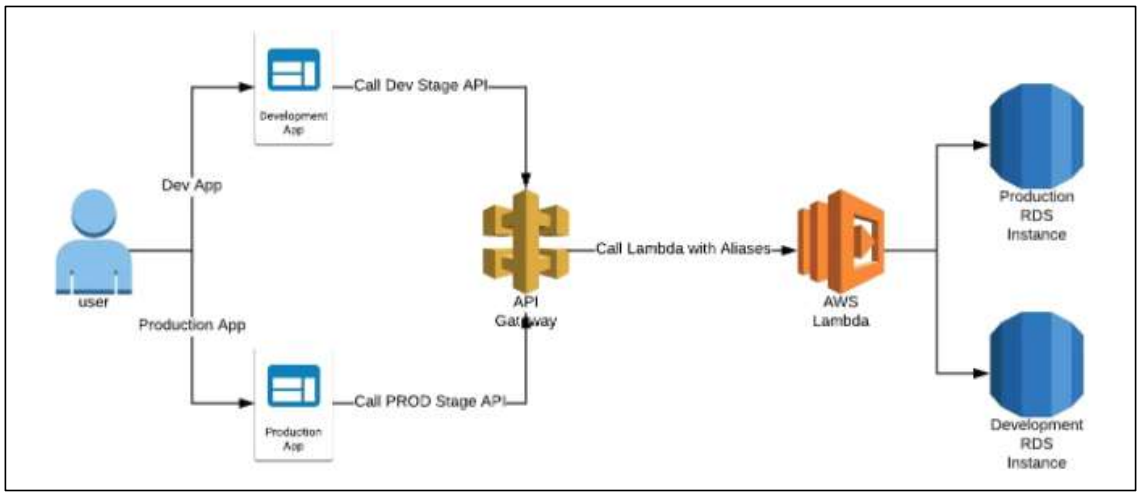

그림 16. 클라이언트의 요청과 수행 결과 통신 방식

따라서 클라이언트의 요청과 그 수행 결과는 API Gateway를 통해서만 통신한다.

제 5절 안드로이드 모바일 앱에서의 REST 요청 구성

앞서 구축한 서버리스 아키텍처와 RESTful 아키텍처를 검증하고자 모바일 앱을 통해 HTTP 프로토콜을 요청한다.

모바일 앱은 안드로이드 스튜디오(Android Studio)와 JDK를 사용해 JAVA 언어로 개발을 진행하며 Http 요청을 사용하기 위해 몇 가지 기능을 작성한다.

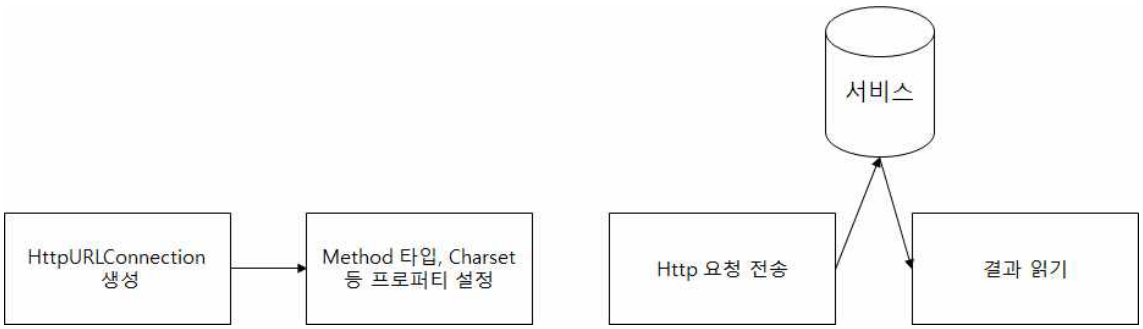

그림 17. JAVA에서 Http 통신을 진행하는 과정

Http 통신을 위해서 자바에서 기본적으로 제공하는 java.net.HttpURLConnection 모듈을 사용한다. 서비스에 기능 수행을 요청하는 방식은 리소스 URI 에 대해 HttpURLConnection을 생성하는 것부터 시작한다. 커넥션을 생성하면 해당 커넥션의 요청 Method 타입, CharSet 등 필요에 따라 적절히 프로퍼티를 설정한다. 다음으로 생성한 커넥션의 OutputStream을 통해 연결된 리소스에 Http 요청을 전송한다. 리소스에서는 기능을 수행한 뒤 그 결과를 다시 요청 클라이언트에게 반환하는데 이때 커넥션의 InputStream을 통해 결과를 받아올 수 있다.

제 4장 구현

제 1절 서버리스 컴퓨팅 구현

그림 18. AWS 관리 콘솔

Amazon Web Services에서 RDS와 Lambda를 사용하여 서버리스 컴퓨팅 구현을 진행한다. (계정 생성 등의 절차는 본 논문에서 다루지 않는다.)

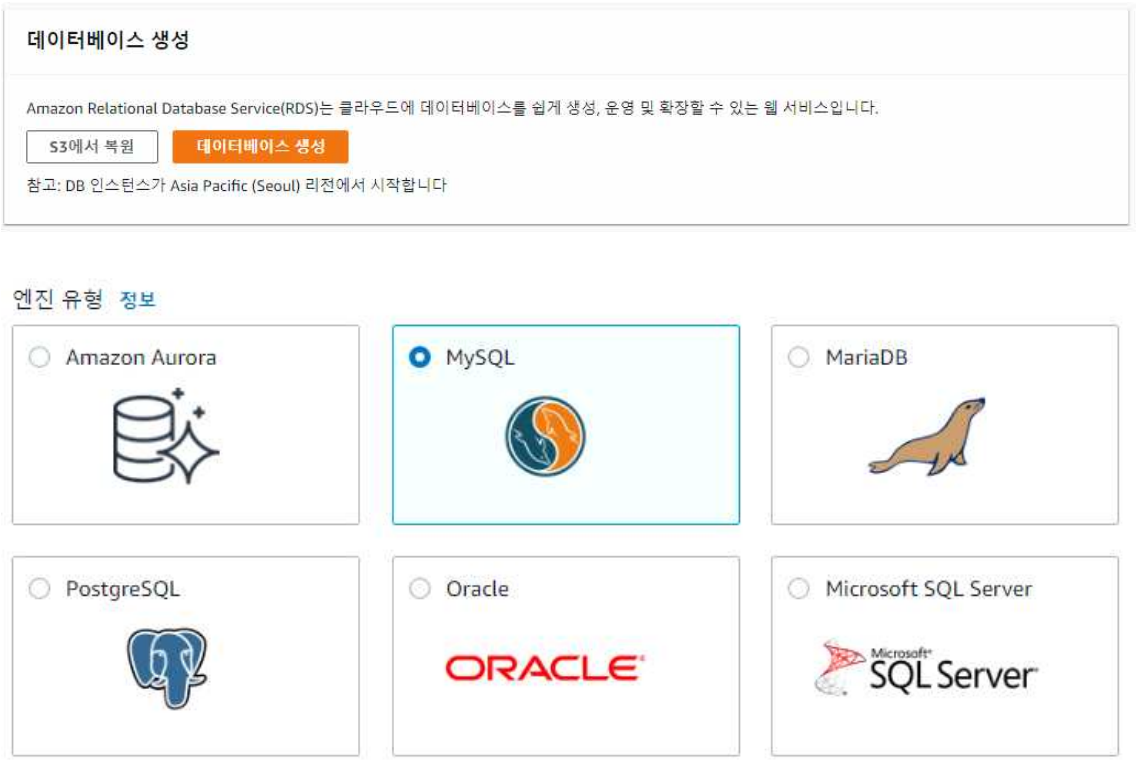

그림 19. RDS를 사용해 MySQL 데이터베이스 구축

RDS 의 데이터베이스 생성을 통해 인스턴스 생성이 가능하다. 앞서 설계한 대로 데이터베이스 엔진은 MySQL을 선택하였고 템플릿은 프리 티어(Free Tier)를 사용하였다.

그림 20. RDS 인스턴스 생성

인스턴스 생성을 진행하면 데이터베이스 목록에 해당 인스터스가 바로 추가 및 초기화된다. 상태가 사용 가능으로 표시된다면 이제 해당 DB 사용이 가능하다.

그림 21. 외부 접속 허용 방법

추가로 외부 접속을 허용하려면 생성 과정에서 퍼블릭 액세스 가능성을 '예'로 지정해야 하며 VPC 보안 그룹 설정에서 인바운드 규칙 편집을 통해 접속 허용할 IP를 추가해야 한다. MySQL Workbench를 사용해 생성한 데이터베이스에 접근하여 데이터를 편집하고 확인할 계획이므로 외부 접속을 허용시켜준다.

그림 22. 생성한 인스턴스의 대시보드

인스턴스가 정상적으로 활성화된 뒤 “연결 & 보안”에 대한 대시보드를 확인한다. 외부에서 해당 데이터베이스에 접근하려면 엔드포인트와 포트, 계정과 암호가 필요하다. 계정과 암호는 인스턴스 생성 시 설정했으며 엔드포인트와 포트는 [그림 22]처럼 대시보드에서 확인할 수 있다.

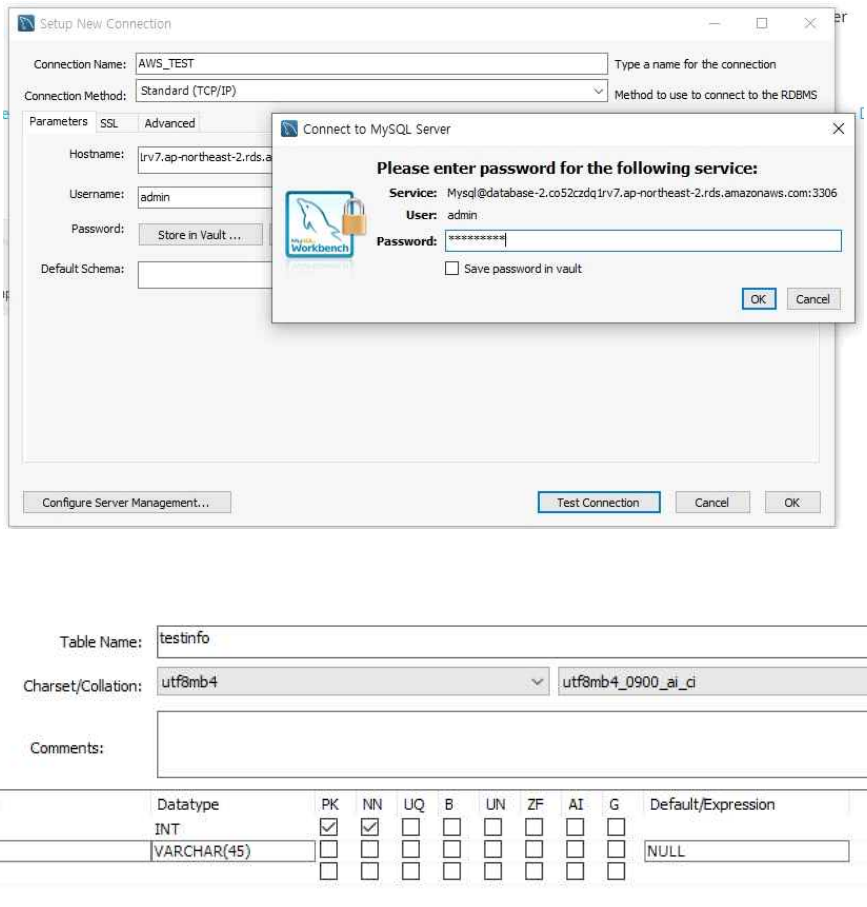

그림 23. 테스트 스키마 생성

앞서 수집한 인스터스 정보와 MySQL Workbench 를 사용하여 데이터베이스에 접속한 뒤 테스트용 스키마를 생성하여 데이터베이스 구성을 완료하였다.

그림 24. Lambda 함수 생성

다음으로 서버리스 컴퓨팅에서 가장 핵심 요소인 Amazon Lambda를 사용하여 생성한 데이터베이스 조작할 수 있도록 함수를 작성한다. 함수는 Python 언어를 사용하여 작성하기에 Python에서 MySQL DB에 접근할 수 있도록 pymysql 모듈을 업로드 해야한다. 따라서 별도로 모듈을 추출한 파일을 Lambda 의 업로드 기능을 통해 추가하고 CRUD 중 CREATE 요청을 수행할 함수를 작성한다.

```
conn = pymysql.connect(host=rds_host, port=port, user=name, password=password, db=db_name, connect_timeout=5)
```

추가된 pymysql 모듈을 import 한 뒤 데이터베이스 커넥션을 생성한다.

```
param = event["param"] # 파라미터 값
id = param["id"]
name = param["name"]
with conn.cursor() as cur:
        nowSql = f"INSERT INTO testinfo values('{id}','{name}')"
        cur.execute( nowSql )
        conn.commit()
```

CREATE 요청은 데이터 생성에 대한 동작을 수행하기에 클라이언트 쪽에서 추가할 데이터에 대한 정보를 함께 전달할 것이며 이를 데이터베이스에 추가하도록 한다.

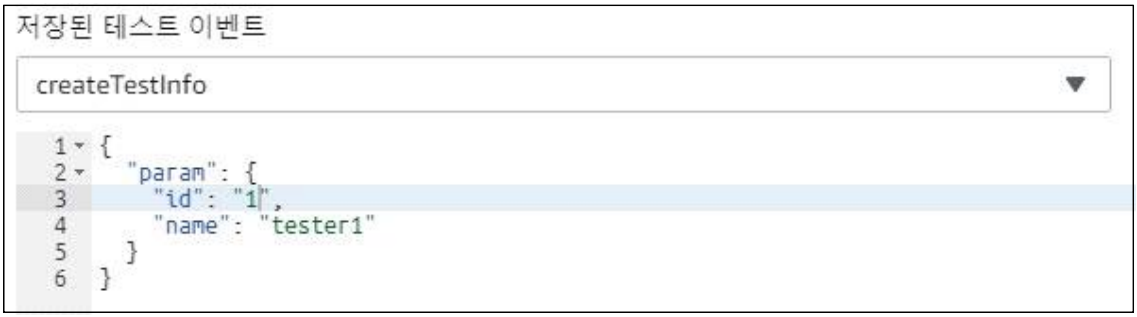

그림 25. Lambda 의 테스트 이벤트 기능

함수 작성을 완료하였다면 Deploy를 하여 최종 반영이 가능하다. 또한 Lambda 에서는 작성한 함수를 즉각 테스트할 수 있도록 테스트 이벤트 기능을 제공한다. 함수가 정상 작동을 확인하고자 테스트 이벤트를 생성하고 파라미터를 설정하여 테스트해본다.

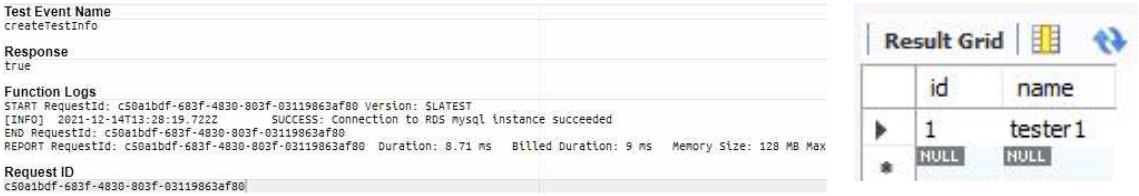

그림 26. Lambda 함수 테스트 결과

함수가 정상적으로 작동하여 true 값을 반환하였으며 MySQL Workbench를 통해 데이터베이스에서 직접 조회한 결과 의도한 대로 데이터 추가가 되어있는 걸 확인할 수 있다.

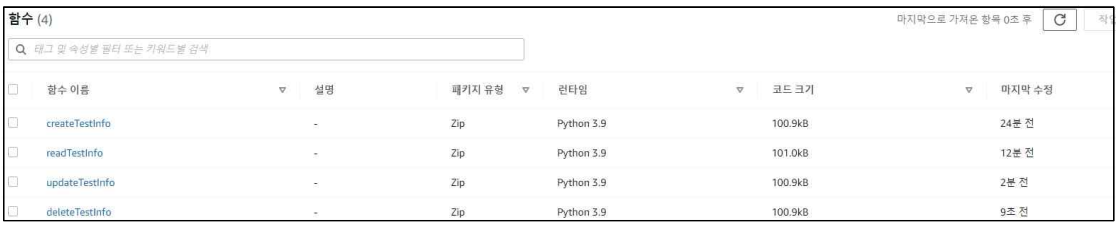

그림 27. CRUD 에 대한 Lambda 함수

같은 방식으로 READ, UPDATE, DELETE 에 대한 요청을 수행할 수 있는 Lambda 함수를 추가적으로 작성한다.

이로서 적은 비용과 별도의 관리 없이 서버 서비스를 제공할 수 있는 서버리스 컴퓨팅 구현을 완료하였다.

제 2절 RESTful API 구현

그림 28. API Gateway 로 구현 가능한 API 종류

API Gateway 서비스를 사용하여 다양한 API를 구현할 수 있으나 본 논문의 취지에 맞게 REST API 구성을 진행하도록 한다.

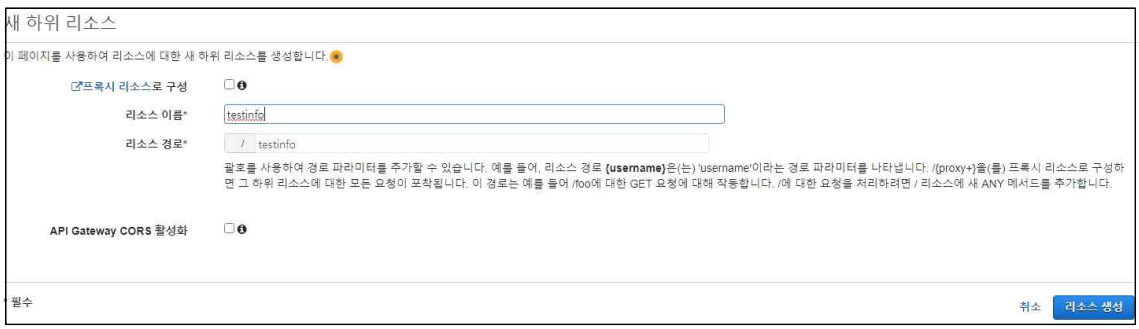

그림 29. 리소스 생성

먼저 클라이언트에서 요청을 보낼 수 있는 리소스를 생성한다. 리소스 경로는 곧 URI 의 요청경로로서 사용된다.

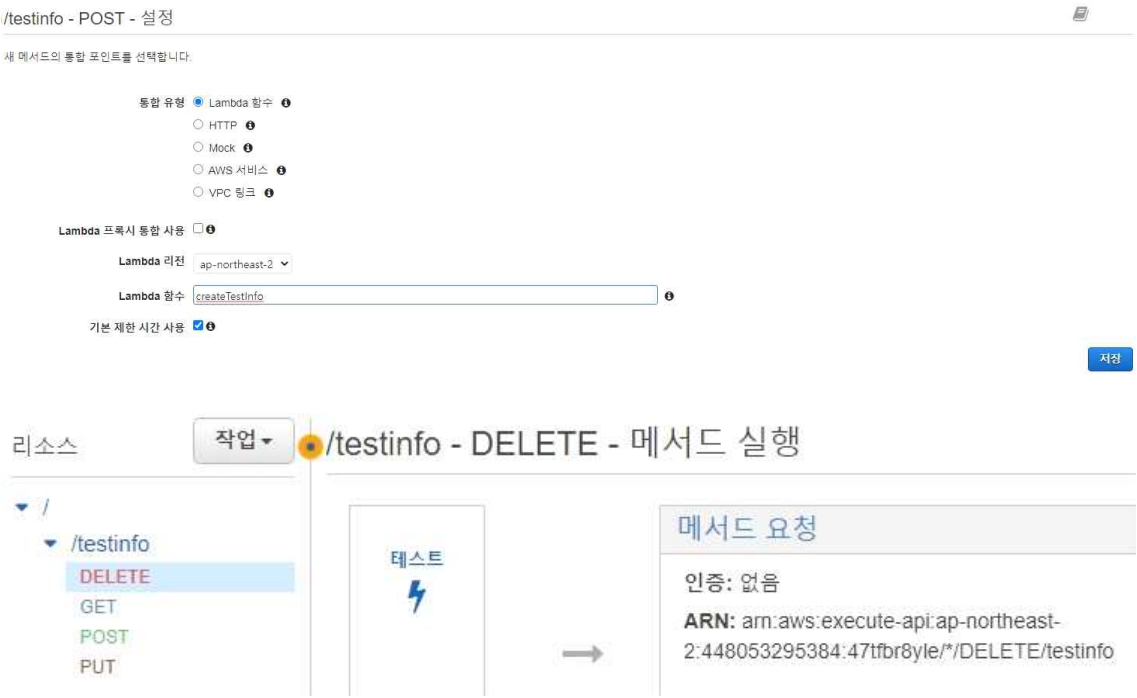

그림 30. Lambda 함수에 대해 CRUD 매핑

생성한 리소스에 POST, GET, PUT, DELETE 요청을 받았을 시 각각 CREATE, READ, UPDATE, DELETE 기능을 수행하도록 메서드를 생성하고 Lambda 함수 매핑을 진행한다.

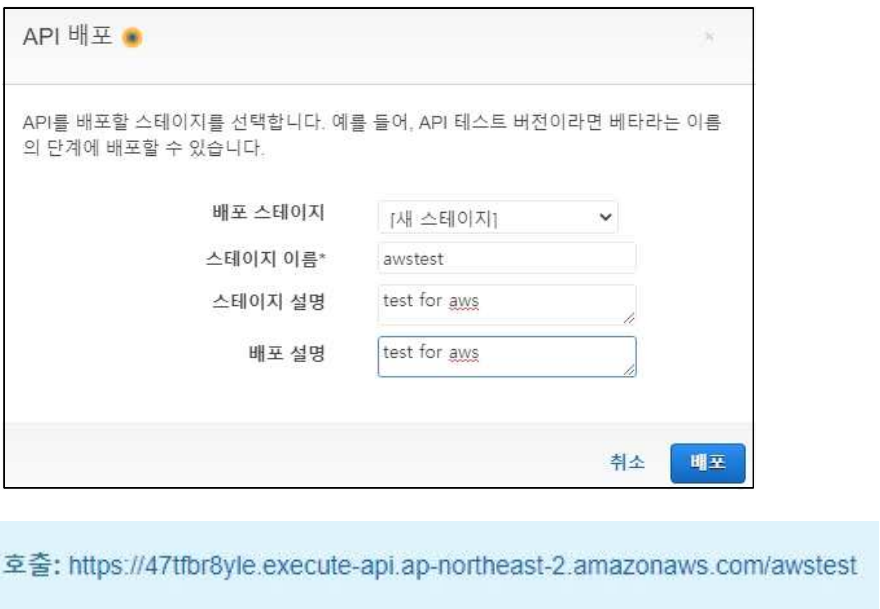

그림 31. API 배포

마지막으로 API 배포를 진행하여 RESTful API 아키텍처 구현을 완료한다.

제 3절 구현 결과

```java
public static String sendREST(String sendUrl, String methodType, String jsonValue) {
    String inputLine = null;
    StringBuffer outResult = new StringBuffer();
    try {
        URL url = new URL(sendUrl);
        HttpURLConnection conn = (HttpURLConnection) url.openConnection();

        conn.setDoOutput(true);
        conn.setRequestMethod(methodType);
        conn.setRequestProperty("Content-Type", "application/json");
        conn.setRequestProperty("Accept-Charset", "UTF-8");
        conn.setConnectTimeout(10000);
        conn.setReadTimeout(10000);

        OutputStream os = conn.getOutputStream();
        os.write(jsonValue.getBytes( charsetName: "UTF-8"));
        os.flush();

        BufferedReader in = new BufferedReader(new InputStreamReader(conn.getInputStream(), charsetName: "UTF-8"));
        while ((inputLine = in.readLine()) != null) {
            outResult.append(inputLine);
        }

        conn.disconnect();
    } catch (Exception e) {
        e.printStackTrace();
    }
    return outResult.toString();
}
```

URL 커넥션 생성

Method 타입 등의 프로퍼티 설정

json 형식의 파라미터와 함께 HTTP 요청 전송

수행 결과 읽기

그림 32. HTTP 요청을 전송하는 메서드

구현 결과를 확인하고자 간단한 모바일 앱을 개발하고 HTTP 통신 결과를 확인한다. JAVA를 사용해 HTTP 요청을 보낼 수 있는 메서드를 하나 생성하였다. 해당 메서드는 [그림 17]에서 서술한 것처럼 커넥션을 생성하고 메서드 타입을 설정한 뒤 요청을 전송한다.

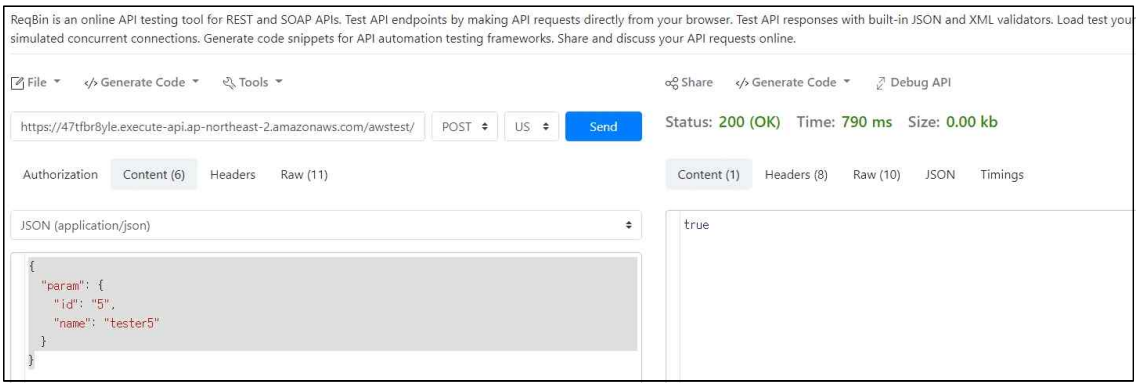

그림 33. HTTP POST 요청 예시

또한 검증을 위해서 모바일 앱뿐만 아니라 웹 브라우저를 사용하여 HTTP 요청을 전송할 수 있다.

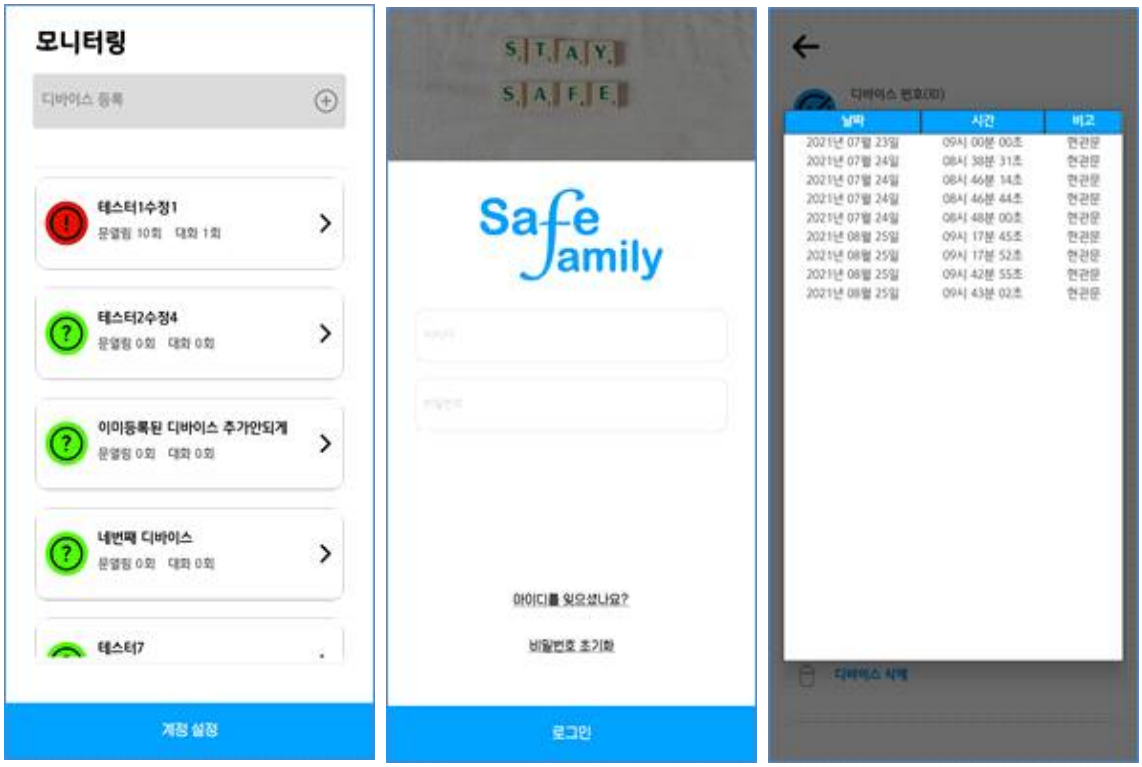

그림 34. RESTful API를 활용한 모바일 앱 예시

RESTful API를 사용하면 JDBC, ODBC 등의 라이브러리보다 데이터베이스를 조회, 조작하기에 있어 보안성과 확장성 효율적이며 서버 개발과 관리 매우 간편하다. 또한 JSON 형태의 반환 값은 가공하기에 간단하므로 계정 서비스, 실시간 로그 확인 서비스 기능을 빠르게 개발할 수 있다.

제 5장 기대효과 및 결론

제 1절 결론 및 향후과제

본 논문에서는 데이터베이스 구성에 있어 RDS를 사용한 관계형 데이터베이스를 구성하였다. 다만 서버리스 아키텍처에서는 DynamoDB 와 같은 NoSQL 형태의 데이터베이스를 더욱 선호한다. 서버리스 아키텍처의 장점은 빠르고 간편하게 서버 서비스를 만들 수 있다는 점인데 별도의 쿼리 지식이나 스키마 지식이 있어야 하는 SQL 형태보다는 누구나 사용할 수 있는 NoSQL이 이에 더욱 적합한 실정이다. 따라서 다음번에는 서버리스 아키텍처 구성에는 NoSQL 데이터베이스를 사용하여 구성할 예정이다.

제 2절 기대효과

본 논문을 통해 시스템 설계 과정에서 점차 사용 플랫폼이 늘어나고 있는 서버리스 아키텍처와 RESTful API 아키텍처를 고려해보고 필요할 시 구현 과정을 참고하여 개발할 시스템에 가장 적합한 아키텍처를 작성할 수 있을 것이라 기대한다.

참 고 문 헌

[1] “서버리스 컴퓨팅”. AWS. 2021년 12월 12일 접속, https://aws.amazon.com/ko/serverless/

[2] “서버리스 아키텍처란”. VELOPERT.LOG. 2021년 12월 12일 접속, https://velopert.com/3543

[3] “서버리스란?”. Red Hat. 2021년 12월 12일 접속,
https://www.redhat.com/ko/topics/cloud-native-apps/what-is-serverless

[4] “서버리스 심층분석”. AWS. 2021년 12월 12일 접속,
https://aws.amazon.com/ko/getting-started/deep-dive-serverless/

[5] “서버리스 아키텍처 파헤치기”. Github. 2021년 12월 12일 접속,
https://futurecreator.github.io/2019/03/14/serverless-architecture/

[6] “첫 번째 서버리스 웹 앱 구축”. AWS. 2021년 12월 12일 접속,.
https://aws.amazon.com/ko/serverless/build-a-web-app/

[7] “AWS Lambda란 무엇입니까?”. AWS. 2021년 12월 13일 접속,
https://docs.aws.amazon.com/ko_kr/lambda/latest/dg/welcome.html#features

[8] “Amazon API Gateway란 무엇입니까?”. AWS. 2021년 12월 13일 접속,
https://docs.aws.amazon.com/ko_kr/apigateway/latest/developerguide/welcome.html

[9] “What is a RESTful API and its methods”. Quora. 2021년 12월 13일 접속,
https://www.quora.com/What-is-a-RESTful-API-and-its-methods

[10] https://gun0912.tistory.com/63