

AWS Lambda 를 활용한 웹 기반 Serverless 개발환경 구축 사례

2023.6.28

클라우드퀘스트 주식회사
이종훈 대표 / alexlee@cloudquest.net

클라우드퀘스트 주식회사

2013 설립

2022 법인 전환

클라우드 컨설팅 및 아키텍처 수립

projectoom.io 서비스 개발 운영

대표자 소개

이종훈 대표

1999 LG CNS / ERP / 아키텍처 그룹

2008 삼성 SDS / OSP

2011 Amadeus Singapore

2013 클라우드퀘스트 설립

2023 **projectroom.io** 런칭

LG계열사, 삼성전자, 대한항공, 건강보험공단
의료급여 프로젝트, 삼성 Account, AWS and GCP

“쉽고 빠르게 웹 시스템을 구현할 수 있는 Serverless 클라우드 기반의 통합 웹 개발환경”

다양한 언어를 지원하는 개발 프레임워크

Lambda, S3 기반의 개발 및 테스트 환경

Git / Git Flow 를 통한 협업 지원

개발 지원 및 컨설팅



웹 개발에 필요한 기술 요소를 Serverless Resource 의 대응

Server 기반

ESB on EC2
S3, EFS

ECS, EKS,
EC2

RDS, Redis,
DynamoDB

ALB

기술 요소

Frontend
(html, js, css, image)

Backend
코드 실행 엔진

데이터 저장소
(DB, Cache, 3rd party Service)

통신 프로토콜
(JSON over http, WebSocket)

Serverless

S3, EFS

ECS, EKS,
Lambda

RDS, Redis,
DynamoDB

API Gateway
ALB

Lambda 중심의 플랫폼 구성

Pros.

다양한 프로그래밍 언어 런타임

저렴한 운영 유지 비용

쉬운 시작

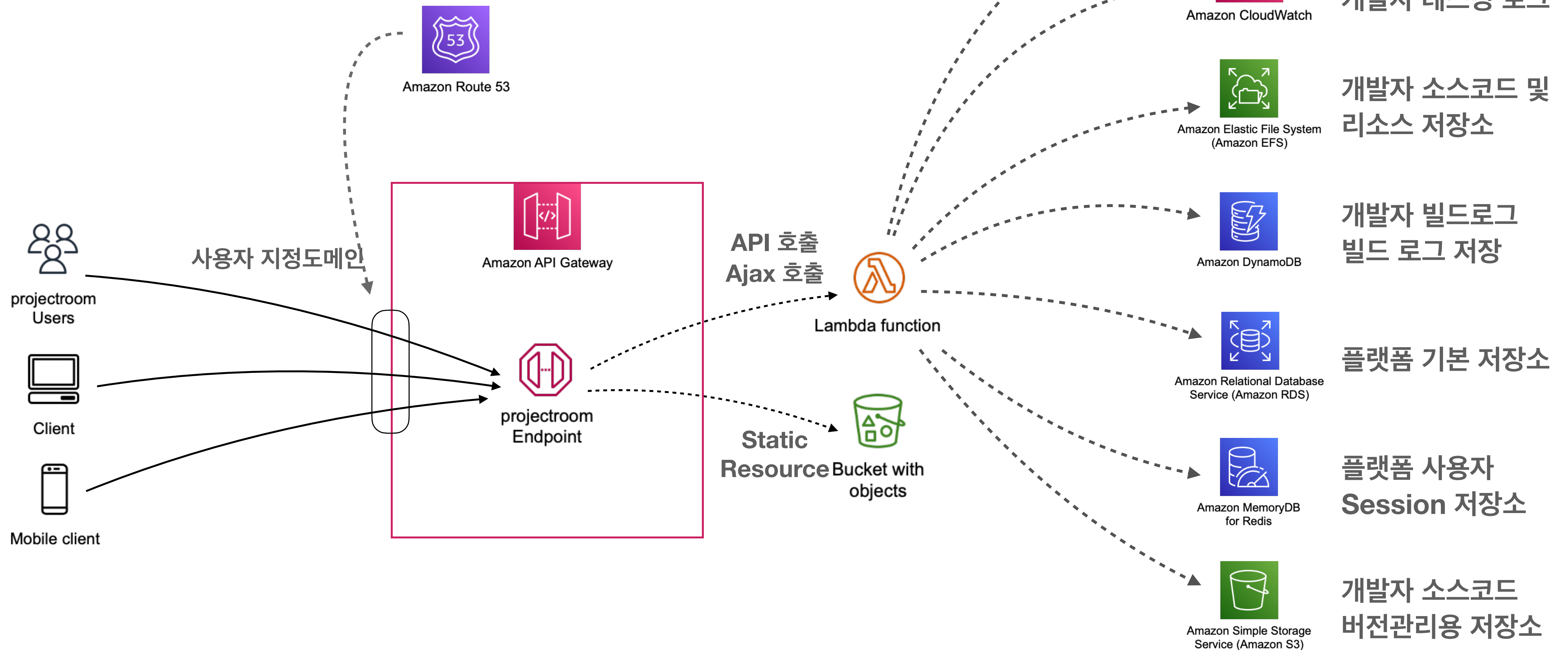
Cons.

단순한 프로그래밍 구조

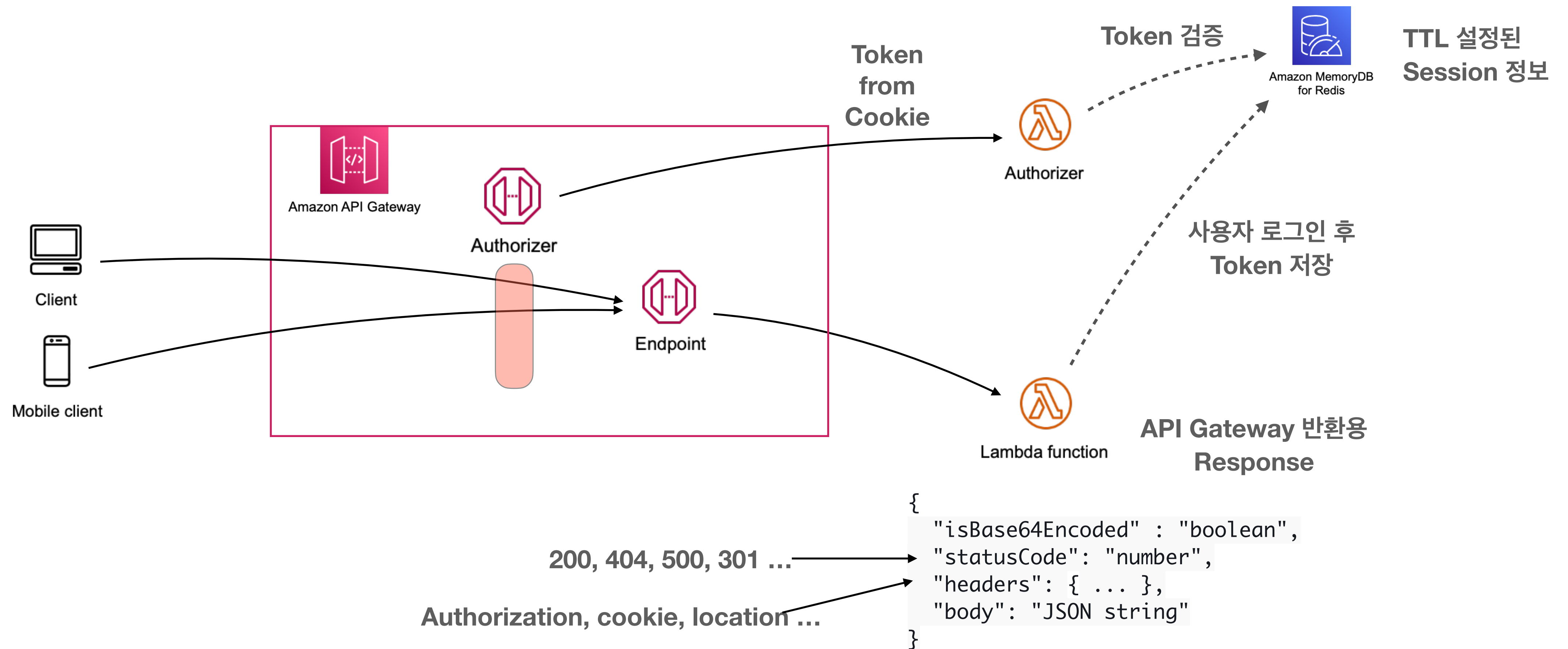
지속성있는 리소스 관리의 어려움

다양한 Serverless 기술 적용이 필요

API Gateway 를 Frontend 로, Lambda 와 S3 를 Backend로 구성
사용자 지정도메인으로 여러 Backend를 동일 도메인으로 구성해 CORS Rule 충족
플랫폼의 필요와 개발자의 필요에 따라 다양한 솔루션을 조합하여 구성



REST API : Lambda 로 연결된 JSON in/out 의 API 구현
Authorizer : 사용자 접근제어 구현



URL 이 고정되지 않고 특정 리소스 아래 주소가 계속 변하는 모든 경우를 수용하는 패턴

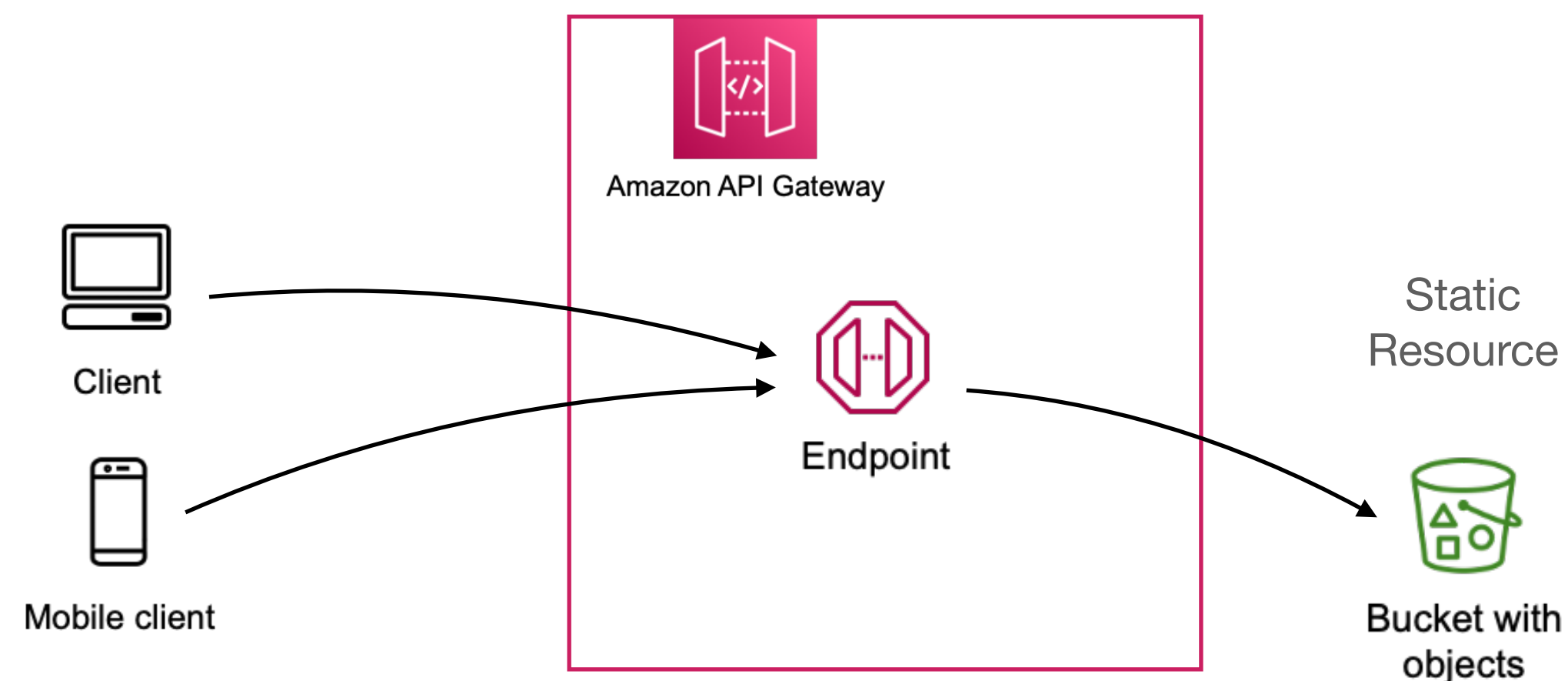
S3에 있는 static 파일의 위치 : [S3버킷이름]/**static**/global/css/all.css

[S3버킷이름]/**static**/main/js/custom.js

[S3버킷이름]/**static**/main/image/logo.png

모든 static 리소스의 최상위 루트

루트 아래에서 파일이 업로드 되는대로 만들어지는 다양한 하위 Path



새 하위 리소스

이 페이지를 사용하여 리소스에 대한 새 하위 리소스를 생성합니다.

프록시 리소스로 구성



리소스 이름*

proxy

리소스 경로*

/static/ {proxy+}

괄호를 사용하여 경로 파라미터를 추가
소스로 구성하면 그 하위 리소스에 대한
스에 새 ANY 메서드를 추가합니다.

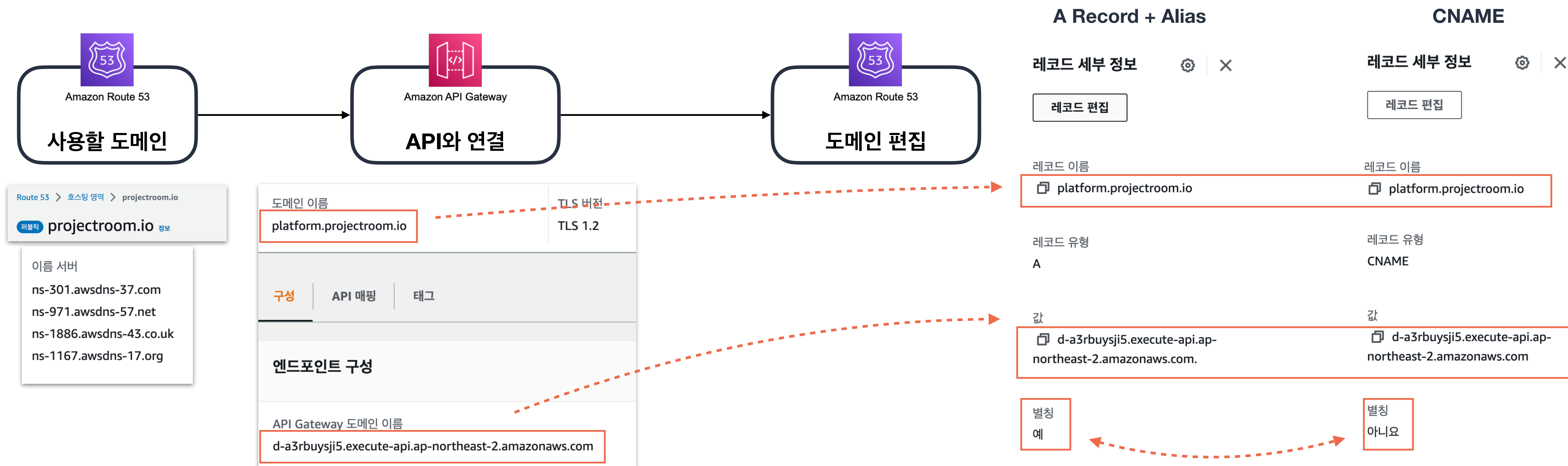
배포된 API 를 대상으로 사용자 지정도메인을 지정해 사용

와일드카드 도메인 적용 가능

AWS Certificate Manager 의 SSL 인증서 발급 및 적용 가능 (무료)

Route53 등록 도메인 (또는 호스팅 영역 등록 도메인) + 동일계정의 API : A Record 별칭 또는 CNAME 연결가능

타사 등록 도메인 또는 타계정 API : CNAME 연결가능



API Gateway 의 사용자 지정 도메인은 여러 API 매핑이 가능
와일드카드 도메인

도메인 이름

platform.projectroom.io

TLS 버전

TLS 1.2

상태

가용

구성

API 매핑

태그

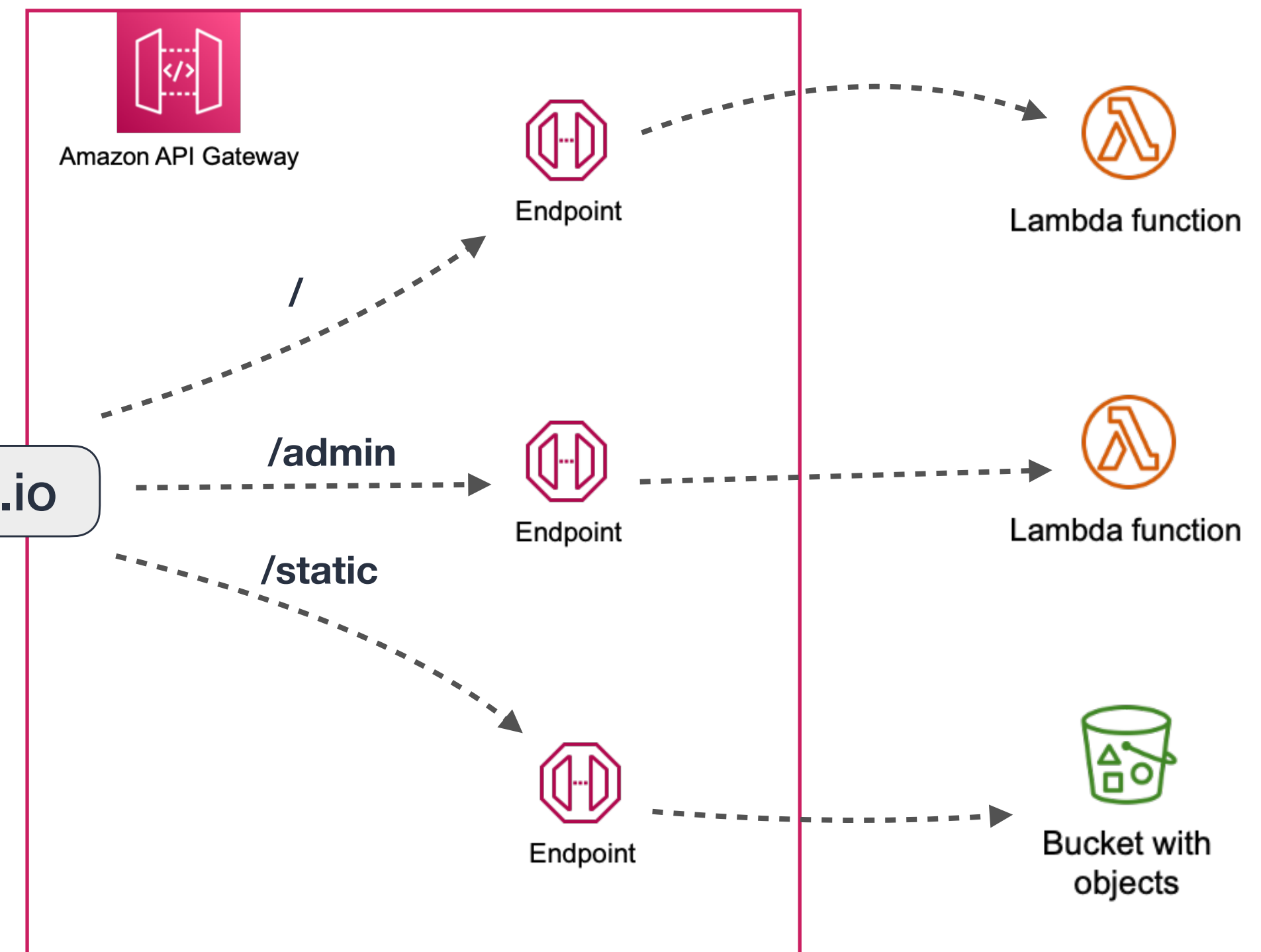
API 매핑

도메인 이름에서 API 스테이지로 경로 매핑

platform

API	스테이지	경로	기본 엔드포인트
projectroom-serverless-platform	prd	(none)	비활성화됨
projectroom-serverless-admin	dev	admin	활성화됨

platform.projectroom.io



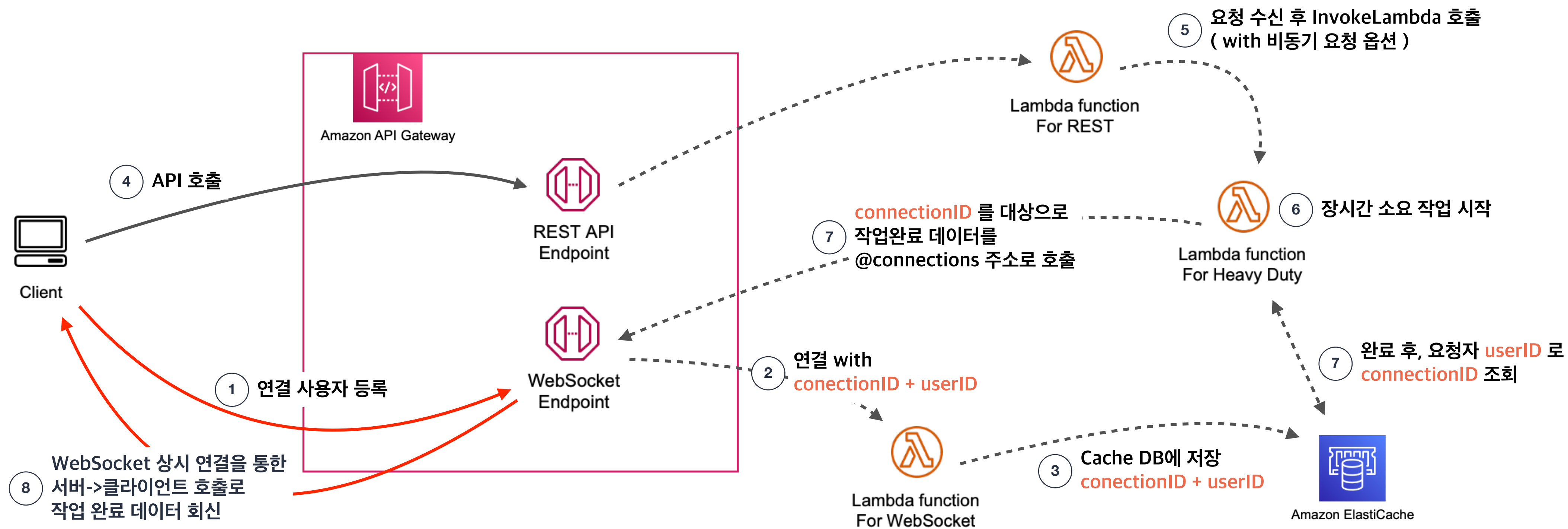
WebSocket API 를 사용하여 Lambda 의 비동기 통신의 결과 회신에 사용

- API Gateway 의 최대 응답 대기 시간 : 30초 (연장 불가)
- 긴 작업시간이 필요한 모든 요청에서, “작업 수행 완료” 를 화면에 리포트 하는 일이 필요

WebSocket URL: `wss://kmwrb6odp4.execute-api.ap-northeast-2.amazonaws.com/prd`
연결 URL: `https://kmwrb6odp4.execute-api.ap-northeast-2.amazonaws.com/prd/@connections`

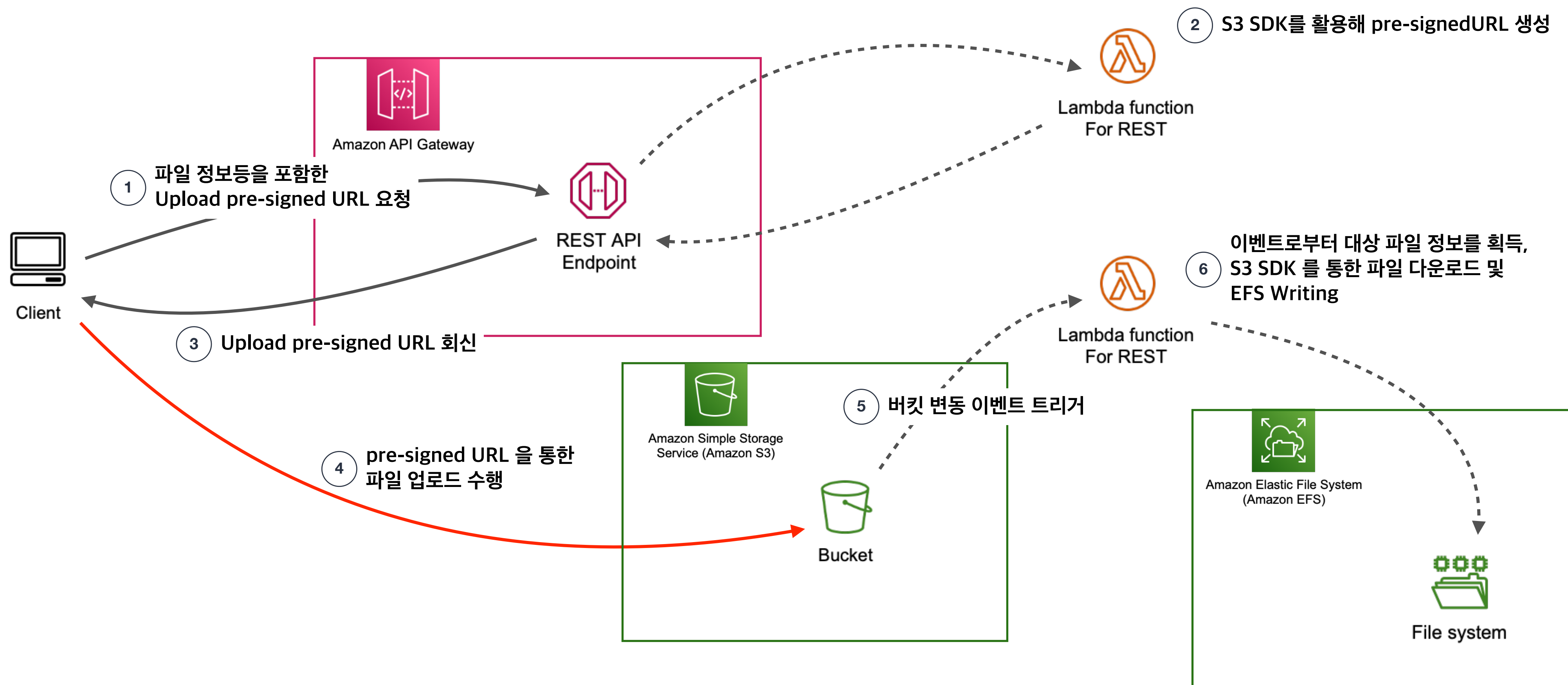
WebSocket 연결을 위해 화면에서 API Gateway 를 호출하는 주소 (화면 -> 서버 메시징은 이 연결을 통해 가능)

WebSocket 응답을 위해 서버측에서 호출하는 주소 (Signed URL 호출 필요)



pre-signed URL을 사용한 파일 업로드 다운로드

- Multipart -> Lambda -> S3 방식은 Serverless 에 적합하지 않음
- 업로드 / 다운로드시 pre-signed URL 을 발급하여 Lambda 를 경유하지 않고 파일을 직접 업로드 다운로드
- S3 업로드시 버킷을 대상으로 이벤트 트리거 발생 => EFS파일 배치 연계 가능



Lambda 의 자동 로그

- Lambda 스스로가 남기는 로그, 또는 Console 로 출력을 잡은 모든 로그가 CloudWatch 로 수집됨
- Tomcat 로그 등과 같이 파일로 남은 로그를 크롤링 해서 CloudWatch 로 적재하는 것도 가능

Lambda 의 수동 로그

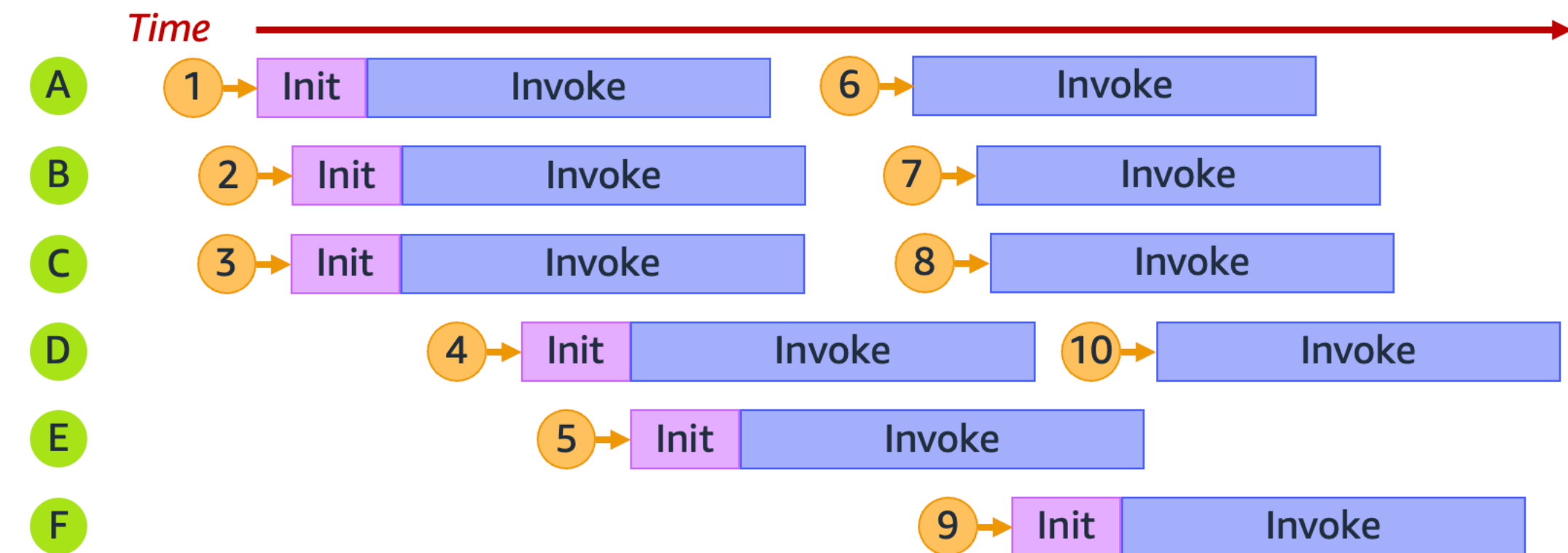
- **로그 그룹**을 생성하고, 업무적 구별을 목적으로 **로그 스트림**을 만들어서 로그를 적재 할 수 있음
- ex) 프로젝트 룸 운영용 Lambda 의 기본 로그와 분리된 “프로젝트 별 작업 로그를 적재”

로그 보존 기간

- 저장된 데이터 기준으로 매월 청구 되기 때문에, 필요 없는 로그를 적당히 지워가면서 관리 필요
- 매월 5G 프리티어 / 수집 GB당 0.76 USD / 아카이브 GB당 0.0314 USD

계정 내 모든 Lambda 가 공유하는 수행 인스턴스의 총량

- 전달된 모든 수행 요청은 각각 1개의 인스턴스에서 수행됨
- 계정 별 지정 쿼터 1,000 개 동시성 보장 (증설 요청 가능)
- Cold Boot (init)시간을 줄이기 위해 프로비전된 동시성 운용 가능



기존 서버의 동시처리

- SpringBoot, Node 등 에서는 서버인스턴스 1개가 상주하며 여러 요청을 처리하는 방식을 사용 (ex. Process > Thread , Event Loop)
- 서버 인스턴스에 적재한 각종 자원을 모든 요청이 공유하도록 구성하여 요청 처리 부하를 최소화

Lambda 의 동시처리

- 모든 요청 수행이 하나의 서버 인스턴스를 사용하므로, 자원 공유를 통한 부하 감소에 어려움 (ex. DB Connection Pool)
- 요청간 정보 공유 및 유지가 필요하면, DB, DynamoDB, Redis, S3 등의 여러 스토리지를 활용하는 설계필요

Lambda 특성

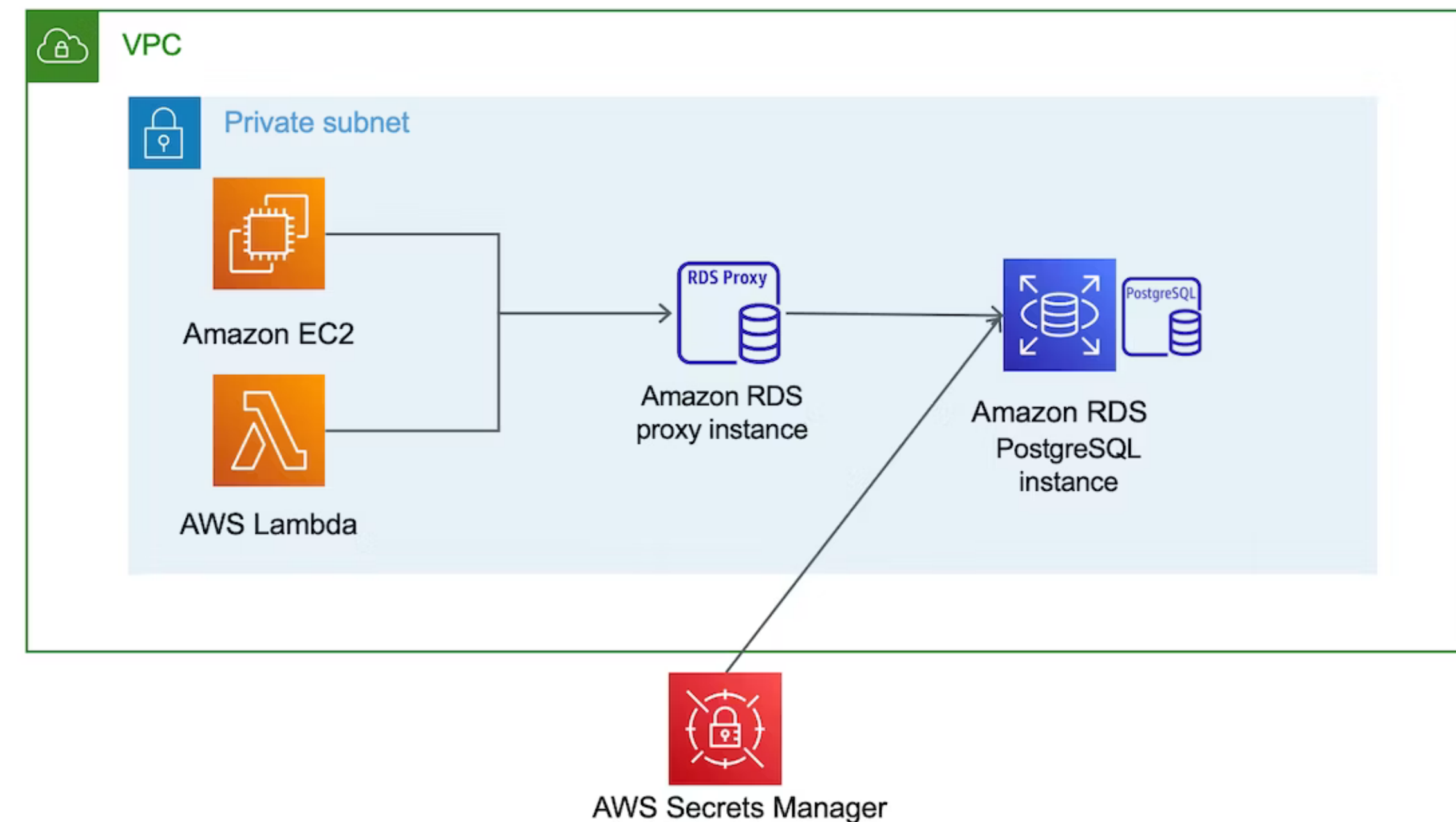
- 요청 발생시 인스턴스 배정 후 처리 시작하고, 추가 요청이 없으면 지정된 시간만큼 유지 후 인스턴스가 사라짐
- 인스턴스 내에 공유 자원을 설정 해도 요청#1과 요청#2가 이 자원을 공유 사용하는 것을 보장하지 않음

DB Connection 의 경우

- 인스턴스 내에 설정한 DB Connection Pool이 사실상 무의미 하므로, 요청 별로 connection 을 생성 / 해제 하며 사용
-> 심각한 성능 저하 초래
- 인스턴스 외부에 DB Connection Pool의 기능을 수행할 별도의 서비스 운용이 불가피함

RDS Proxy

- DB Connection Pool 기능 제공
- VPC 건너의 RDS와도 연결이 가능
- DB와 똑같은 접속 방법
- RDS DB의 vCpu 당 0.018 USD/Hour -> 추가 비용 필요
- Secret Manager 적용 필수 -> 추가 비용 필요



RDS Proxy 의 인증

Application 에서는 DB와 동일 하게 접속 하므로, DB ID Password 인증을 수행할 수 있으나 실제 DB가 아니므로 인증을 위한 별도의 서비스로 Secrets Manager 를 활용하며, 이 정보는 RDS Proxy 가 DB 인스턴스와 연결을 유지하는 데에도 사용됨

Secret Manager 비용

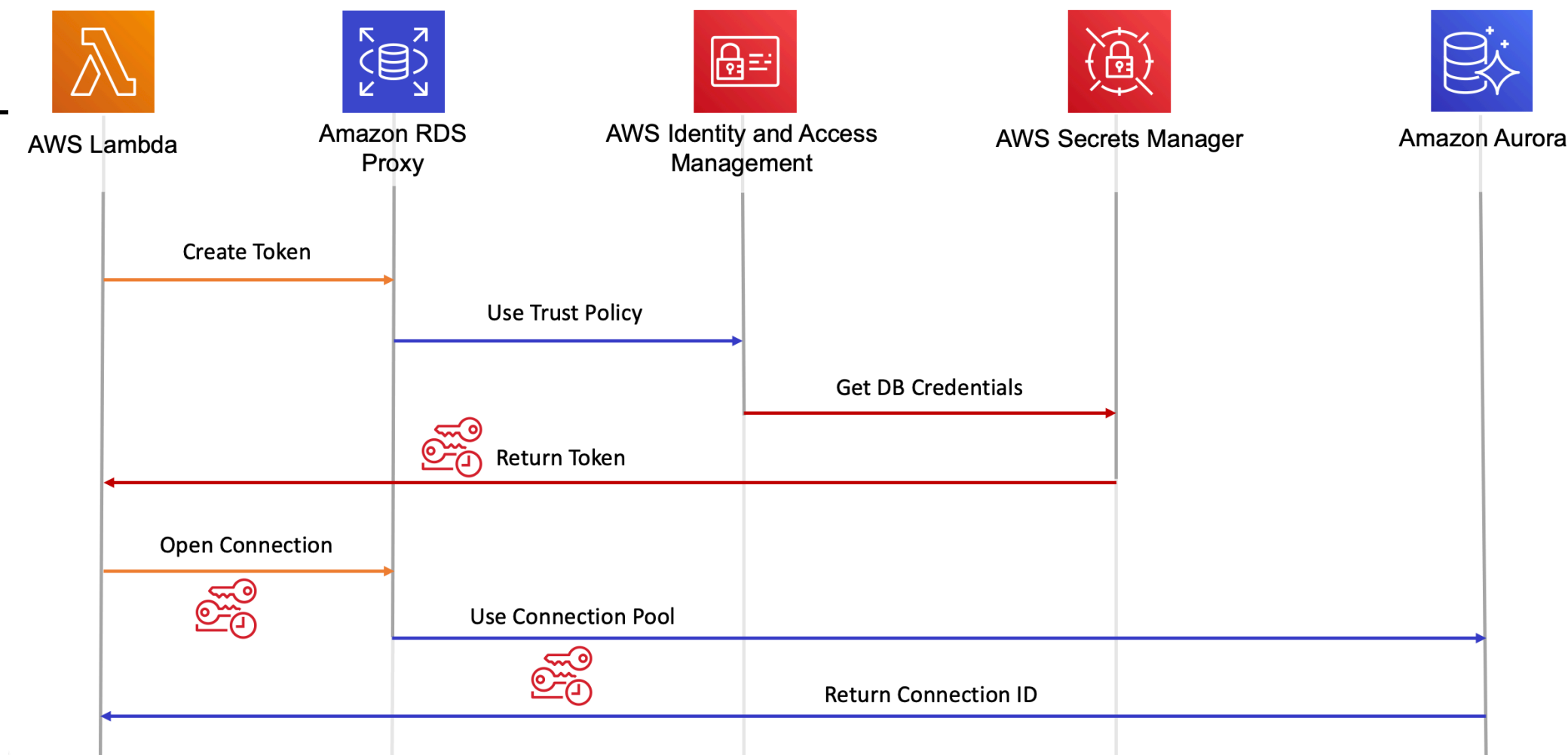
계정의 보유 수와 인증 요청 총 숫자로 비용 청구 -> 비싸다
IAM 인증을 활용해 인증 요청 비용을 줄일 수 있음

```
RdsUtilities utilities = rdsClient.utilities();
```






```
GenerateAuthenticationTokenRequest tokenRequest = GenerateAuthenticationTokenRequest.builder()
    .credentialsProvider(EnvironmentVariableCredentialsProvider.create())
    .username(System.getenv( name: "db_user"))
    .port( i: 3306)
    .hostname(System.getenv( name: "db_host"))
    .build();
```

```
String token = utilities.generateAuthenticationToken(tokenRequest);
```

```
DataSourceBuilder<CustomSimpleDriverDataSource> dataSourceBuilder = DataSourceBuilder.create().type(
    dataSourceBuilder.driverClassName(System.getenv( name: "db_driver_class_name"));
dataSourceBuilder.url(System.getenv( name: "db_url"));
dataSourceBuilder.username(System.getenv( name: "db_user"));
dataSourceBuilder.password(token);
```



IAM 인증 적용 이전

 Asia Pacific (Seoul)	USD 186.58	
 AWS Secrets Manager APN2-AWSSecretsManager-Secrets	USD 93.44	
 \$0.40 per Secret	233.599 Secrets	USD 93.44
 AWS Secrets Manager APN2-AWSSecretsManagerAPIRequests	USD 93.14	
 \$0.05 per 10000 API Requests	18,628,957 API Requests	USD 93.14

18,628,957 API Requests

IAM 인증 적용 이후

[-] Asia Pacific (Seoul)	USD 79.24	
[-] AWS Secrets Manager APN2-AWSSecretsManager-Secrets	USD 54.23	
\$0.40 per Secret	135.586 Secrets	USD 54.23
[-] AWS Secrets Manager APN2-AWSSecretsManagerAPIRequests	USD 25.01	
\$0.05 per 10000 API Requests	5,001,411 API Requests	USD 25.01

5,001,411 API Requests

1. Command line tool (feat. Git)

- Git (w/ ssh) binaries for AWS Lambda (<https://github.com/lambci/git-lambda-layer>)
- Git 커맨드를 Lambda 에서 사용 가능
- Lambda 구현 언어의 외부 Command 프로세스 구동 명령어를 이용
(Node.js executeSync(), Java 의 Process 클래스 등)

Layer selection
 Select an existing AWS-vended layer or layer in your account, or provide a layer that has been shared with you

☐ Select from list of runtime compatible layers
☒ Provide a layer version ARN

Provide a layer version ARN
 Layer version ARN [Info](#)
 Provide the ARN of a layer to add to your function.

2. External Libraries

- Lambda 계층 만들기 및 공유(https://docs.aws.amazon.com/ko_kr/lambda/latest/dg/configuration-layers.html)
- 배포 시간과 본 함수 패키지 크기(함수, Layer 각각 250M 제한)를 줄일 수 있음
- 최대 5개 까지 추가 가능

각 Lambda 런타임에 대한 계층 경로	
런타임	경로
Node.js	nodejs/node_modules
	nodejs/node14/node_modules (NODE_PATH)
Python	python
	python/lib/python3.9/site-packages (사이트 디렉터리)
Java	java/lib (CLASSPATH)
Ruby	ruby/gems/2.7.0 (GEM_PATH)
	ruby/lib (RUBYLIB)
모든 런타임	bin (PATH)
	lib (LD_LIBRARY_PATH)

1. 컨테이너 이미지 배포

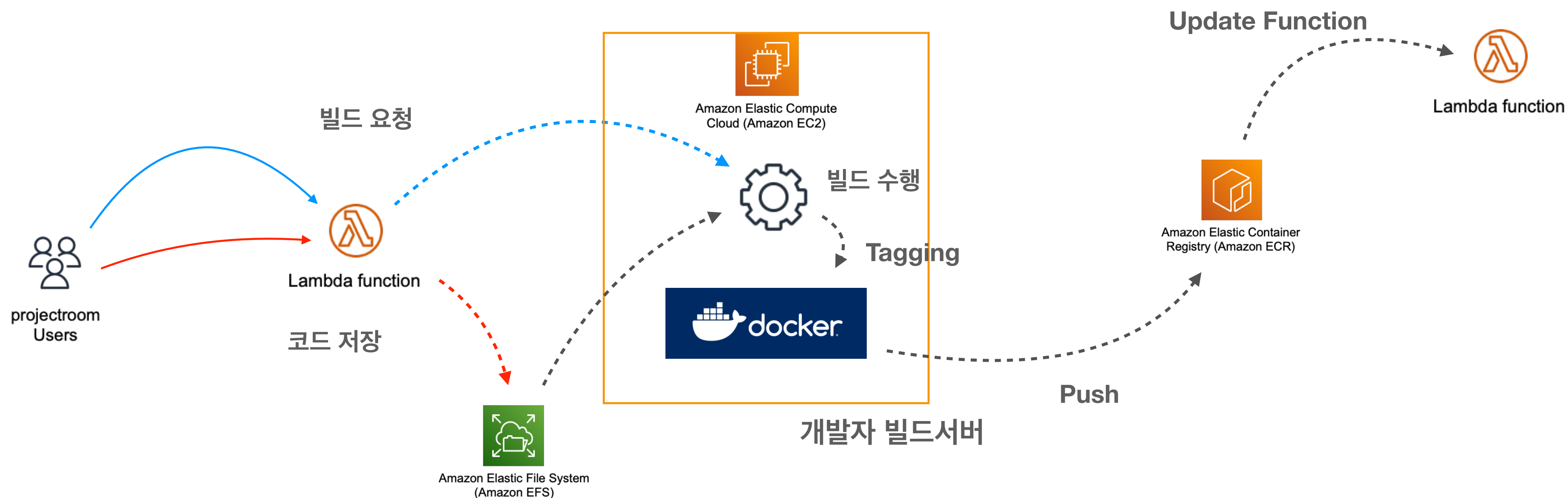
- Docker 이미지 매니페스트 V2, 스키마 2 / Open Container Initiative(OCI) 컨테이너 지원 등을 지원
- 최대 10G의 비 압축 이미지 지원
- Lambda 컨테이너 안에서 다양한 유틸리티의 설치 사용이 가능
- Lambda 실행 모드에서 사용자와 권한 바뀔때 주의 (\$USER_HOME 이 읽기 전용)

```
FROM public.ecr.aws/lambda/nodejs:18
```

```
# Assumes your function is named "app.js", and then  
COPY app.js package.json ${LAMBDA_TASK_ROOT}/
```

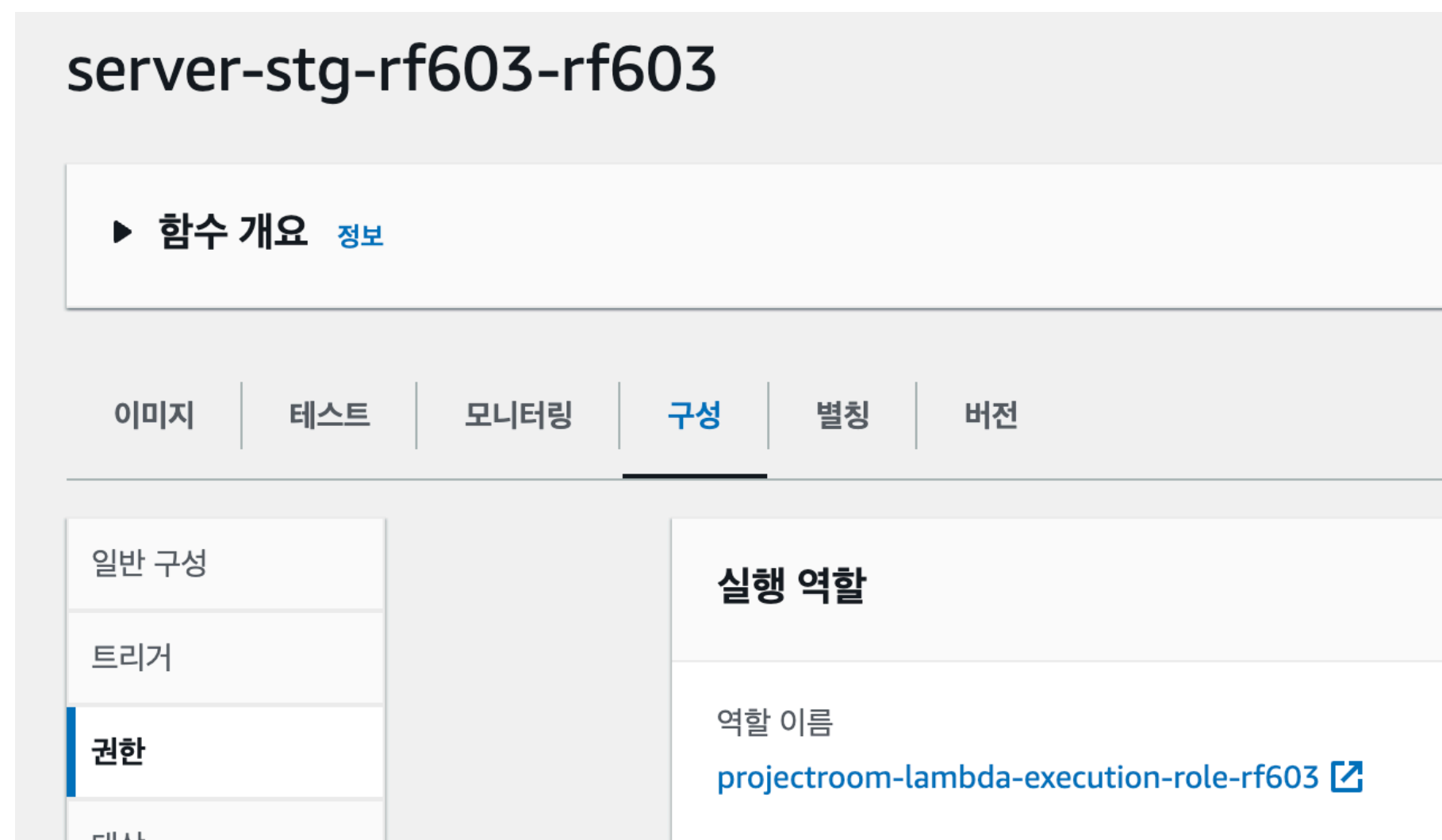
```
# Install NPM dependencies for function  
RUN npm install
```

```
# Set the CMD to your handler (could also be done at  
CMD [ "app.handler" ]
```



Role Based : assumeRole 정책적용

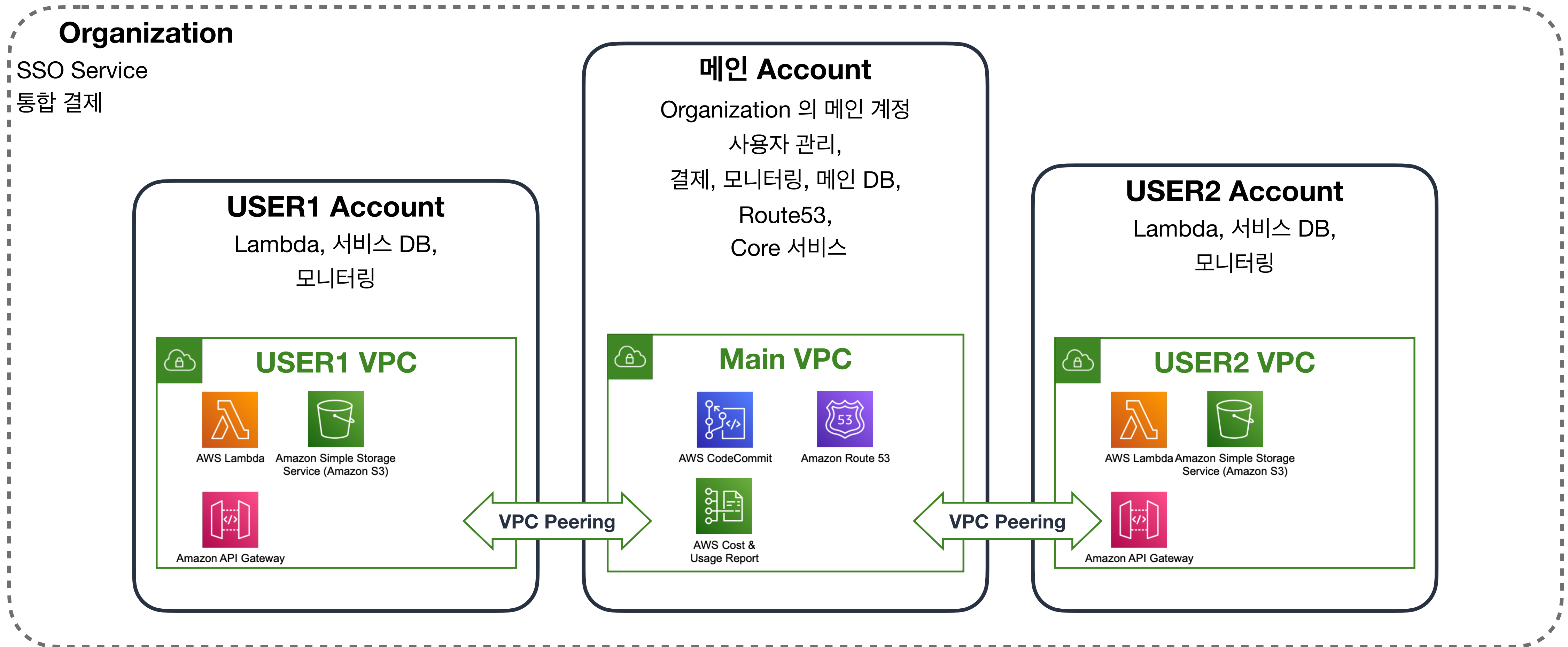
- Lambda 에 1개의 Role 을 적용할 수 있으며, Lambda 실행시 해당 Role 에서 생성한 임의의 Key, Secret, Token 생성하여 환경 변수에 적용
- Lambda 코드 내부에서 작동하는 AWS SDK는 따로 코드에 인증 정보를 명시하지 않아도 해당 Role 이 지정한 범위의 동작을 구현할 수 있음
- 권한 적용을 통한 자원 접근제어의
 - 특정 S3의 버킷 아래 디렉토리의 독점적 접근 지정
 - DynamoDB 테이블에 대한 독점적 접근 지정
 - RDS 사용자 계정 및 데이터 베이스에 대한 독점적 접근 지정



```
{
  "Sid": "VisualEditor1",
  "Effect": "Allow",
  "Action": [
    "s3:PutObject",
    "s3:GetObject",
    "rds-db:connect",
    "dynamodb:PutItem",
    "dynamodb>DeleteItem",
    "s3:GetObjectAttributes",
    "dynamodb:GetItem",
    "dynamodb:UpdateItem",
    "s3>DeleteObject",
    "s3:GetObjectVersion"
  ],
  "Resource": [
    "arn:aws:rds-db:ap-northeast-2:613210570360:dbuser:prx-0ca249faa1aee90f2/user_rf603",
    "arn:aws:s3:::projectroom-project-file-storage-user1/storage-dev-rf603/*",
    "arn:aws:dynamodb:ap-northeast-2:613210570360:table/project-simple-data-rf603"
  ]
}
```

API Gateway, Lambda 를 비롯한 AWS 서비스 자원의 Quota 가 존재

- “User 별 Region 당 000” 방식으로 카운트
- Quota 를 늘리는 방법 ==> **하나의 Organization 아래 여러 User 를 운용하는 것**



서비스 요건에 유연하게 대응하는 아키텍처 : 빌드서버 구축의 사례

빌드 서버 요건	EC2 + Auto Scaling Group	ECS Fargate Lambda with Machine Image	Code Build
반응성 : 빠른 시작	GOOD Spring Boot Servlet	BAD Provisioning or Cold/Warm Start	BAD Provisioning
빌드 라이브러리 캐시 저장소 : 반복 수행시 빌드 속도 향상	GOOD EBS on EC2	BAD Temp Storage EFS	BAD Temp Storage
추가적인 응용구성 DB / Redis / WebSocket / CloudWatch	GOOD Based on Application F/W	GOOD Based on Application F/W	BAD CLI is not enough
가용성 / 확장성	GOOD EC2 Auto Scaling Group	GOOD Auto Scaling	GOOD Auto provisioning by Request
운영 비용	BAD 상시 유지 필요	GOOD 필요할 때 만들고, 쓴 만큼 과금	GOOD 필요할 때 만들고, 쓴 만큼 과금

QnA