6. 서버리스 컴퓨팅 아키텍처 설계

Cloud_서비스/세일즈 툴킷

Exported on 06/19/2023

Table of Contents

서버리스 컴퓨팅 적용 영역과 아키텍처 예시를 통해 아키텍처 고려사항을 기술하고, 서버리스 컴퓨팅 논리 아키텍처를 통해 연계하여 사용할 수 있는 다른 서비스들을 선택할 수 있는 정보를 제공하고, 실제 환경에서 구현될 수 있는 가상의 시나리오를 기반으로 기존 환경에서 서버리스 컴퓨팅으로 설계하는 내용을 기술하여 실제 설계 시 참고할 수 있는 자료를 제공한다.

1 서버리스 컴퓨팅 아키텍처 설계 시 고려사항

1.1 적용 영역 선정

1.1.1 신규 소규모 서비스

빠르게 출시해야 하는 소규모의 웹사이트나 모바일 어플리케이션은 높은 개발 생산성을 가진 서버리스 컴퓨팅을 이용해 구축하면 효율적이다. 또한 신규 서비스들은 수요 예측이 어려운데 서버리스 컴퓨팅의 자동 확장 기능으로 안정적인 서비스를 구현할 수 있다.
하지만 대규모 아키텍처는 함수들 관리가 복잡할 것이고 이를 통합 모니터링 하거나 디버깅하기 어렵다. 또한 동시 실행 수나 실행시간의 제한으로 인해 다른 소프트웨어 구축 방안들보다 더 많은 관리가 필요할 수 있다.

1.1.2 Add on

기존 시스템에 기능을 추가할 때 서버리스 컴퓨팅의 도입할 때, 새로운 기능에 서버리스 컴퓨팅의 트리거가 될 서비스만 연결하면 해당 기능이 변경되어도 기존 시스템 패치 없이 함수만 변경하면 되어 보다 빠른 대응이 가능해지며, 기존 시스템에서 자주 변경되는 기능들도 이와 같이 리팩토링 할 수 있다.
하지만 기존 시스템을 전체 마이그레이션 하는 것은 추천하지 않는다. 그럼에도 불구하고 진행해야 한다면 특정 기능을 점진적으로 교체하여 단계적으로 마이그레이션하는 스트랭글러 패턴을 이용해야 한다.

1.1.3 이벤트 드리븐 마이그레이션

기존 아키텍처를 이벤트 드리븐 아키텍처로 마이그레이션 할 때 서버리스 컴퓨팅을 사용할 수 있다. 예를 들어 Web socket의 경우 클라이언트가 많아지면 많은 서버가 필요한데 이벤트 드리븐 의 경우 이벤트가 발생할 때만 서버리스 컴퓨팅이 동작하고 발생 수만큼 자동확장하여 효율 적으로 처리할 수 있게 된다.

1.1.4 Proxy API 서버

폐쇠망에서 외부의 데이터를 가져올 때 서버리스 컴퓨팅를 이용해 Proxy API서버로 사용할 수 있다.

1.1.5 배치 처리

타이머와 서버리스 컴퓨팅을 이용해 배치 작업을 효과적으로 사용할 수 있는데, 대부분의 배치는 자주 실행하지 않고 일정한 간격을 가지고 실행되기 때문에 실행할 때만 비용이 발생하는 서버리스 컴퓨팅이 비용 효율적이다. 단, 실행 제한시간을 고려하여 오래 걸리는 작업이라면 비동기 호출로 수행하거나 함수를 잘게 쪼개 사용해야 한다.

1.1.6 스트림 데이터 실시간 처리

대량 스트림 데이터를 수집한 후 서버리스 컴퓨팅을 활용해 분석하게 되면 수집할 때만 가동되어 비용 효율적이고, 실시간으로 분석을 수행할 수 있다. 사용한 만큼 비용이 발생하는 서버리스 컴퓨팅의 비용체계로 인해 동일한 비용으로 빠른 분석을 하려면 스트림 데이터를 병렬로 처리해야 한다.

1.2 적용 아키텍처 선정

서버리스 컴퓨팅 중 가장 많이 사용하는 AWS Lambda를 이용하여 비즈니스 요구사항을 만족시키기 위한 아키텍처들을 작성하여 이와 유사한 사례에 도입하기 위한 정보들을 제공한다. AWS 서비스들의 자세한 내용은 본문에는 포함하지 않으니 기존에 정리된 TEC의 Cloud 나 AWS 문서를 참고한다.

1.2.1 파일처리 아키텍처

파일에 대한 이벤트를 감지하여 서버리스 컴퓨팅이 이를 처리하는 아키텍처로 이미지나 동영상 같은 콘텐츠성 파일을 변환하거나 데이터 파일을 분석하는 용도로 사용할 수 있다. 파일처리를 서버리스 컴퓨팅으로 분리하면 파일 처리에 따른 부하를 줄일 수 있어 효과적이다.

1.2.1.1 AWS 서비스 구성

항목	AWS 서비스	특징
Input	S3	객체 스토리지 서비스 파일의 변화 시 Lambda 호출 가능
	EC2 + SNS	가상머신과 SNS 서비스
Function	Lambda	서버리스 컴퓨팅 서비스로 API Gateway와 연결
Output	S3	객체 스토리지 서비스
	RDS	관계형 데이터베이스 서비스
	EC2 DB	가상머신에 DB를 설치하여 사용
	DynamoDB	NoSQL 서비스
	Redshift	Datawarehouse 서비스

1.2.1.2 AWS 아키텍처 구성도

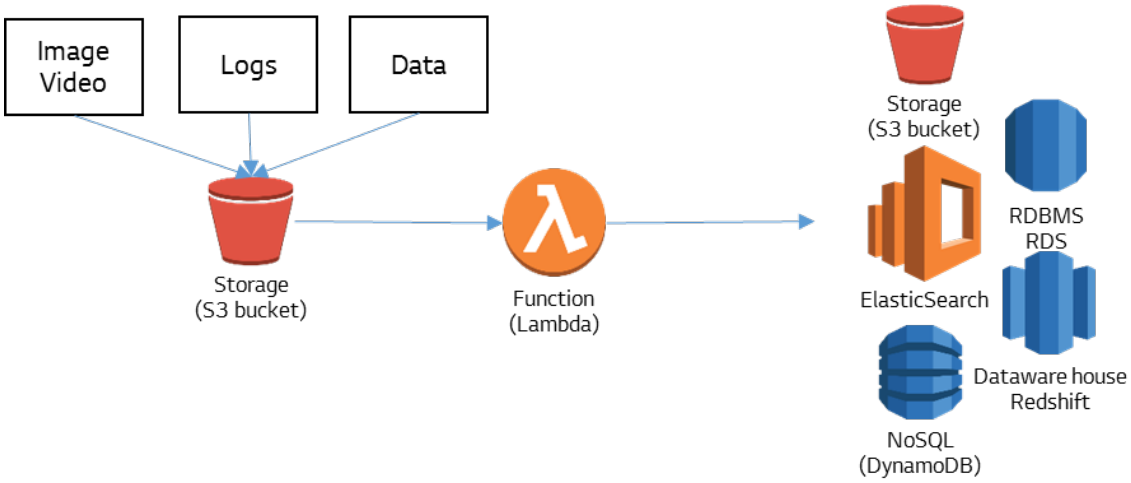

그림. 파일처리 AWS 아키텍

1.2.1.3 타 업체 사례

회사명	회사소개	사용사례
Finra	미국 금융산업규제기관	하루 최대 50조건의 데이터 검증 시 서버리스컴퓨팅을 활용하여 비용 효율성이 2배 증가하고 생산성은 3대 증대됨
Square Enix	게임회사	게임 내에서 이미지 캡쳐 시 서버 부하가 많이 발생하여 이미지 변환 작업을 서버리스 컴퓨팅으로 변환하여 이미지 변환 시간이 몇 시간에서 10초 이하로 단축되고 1/20의 비용 절감
Benchling	생명공학	파일 스토리지에 게놈 데이터를 업로드하면 서버리스 컴퓨팅이 이를 분석하는 용도로 처리하여 비용절감 효과와 검색시간이 90% 절감
Zapproved	eDiscovery (전자증거) 솔루션 회사	서버리스 컴퓨팅으로 OCR 이미지 처리, 시간당 수천만개의 로그 처리, WAF Rule 처리, Serverless Back-end로 활용하여 처리시간을 몇 시간에서 20초 이하로 단축하고 1/10 비용 절감
Vogue (PhotoVogue)	온라인 사진 플랫폼	이미지 자동 변환 (PNG, GIF, JPEG, TIFF 등 자동 변환) 으로 활용하여 비용 절감

회사명	회사소개	사용사례
The Seattle Times	뉴스 미디어	각 플랫폼 별 (PC, Phone, Tablet) 이미지 크기 조정으로 기존 대비 변환 속도 향상

1.2.2 실시간 스트림 데이터 처리 아키텍처

Pipe-Filter 아키텍처 패턴으로 연속적인 input 데이터에 일련의 변형을 주거나 필요한 부분만 추출하는 아키텍처이다. ETL(Extraction, Transformation, Loading), IoT 장비 센서 데이터 수집 및 실시간 처리, 비디오 스트리밍이나 웹사이트에서 고객의 클릭 이벤트, 소셜미디어의 RRS FEED*, IT 서비스의 로그들, 위치 같은 스트림 데이터를 수집하여 실시간 처리하는 용도로 사용할 수 있다.

*) RSS FEED: 뉴스나 블로그 사이트에서 자주 업데이트되는 콘텐츠를 XML 기반으로 제공하는 표준 통신 포멧

1.2.2.1 AWS 서비스 구성

항목	AWS 서비스	특징
Input	Kinesis	데이터, 비디오 스트림 데이터 수집, 처리 및 분석 서비스
	DynamoDB	NoSQL, 스트림 기능
Filter	Lambda	목적에 맞게 여러 기능으로 분산 가능
Output	Redshift	DataWarehouse 서비스
	S3	객체 스토리지 서비스
	DynamoDB	NoSQL 서비스
	RDS	RDBMS 관리형 서비스
	ElasticSearch	분산형 검색 및 분석 엔진
	EC2	가상머신

1.2.2.2 AWS 아키텍처 구성도

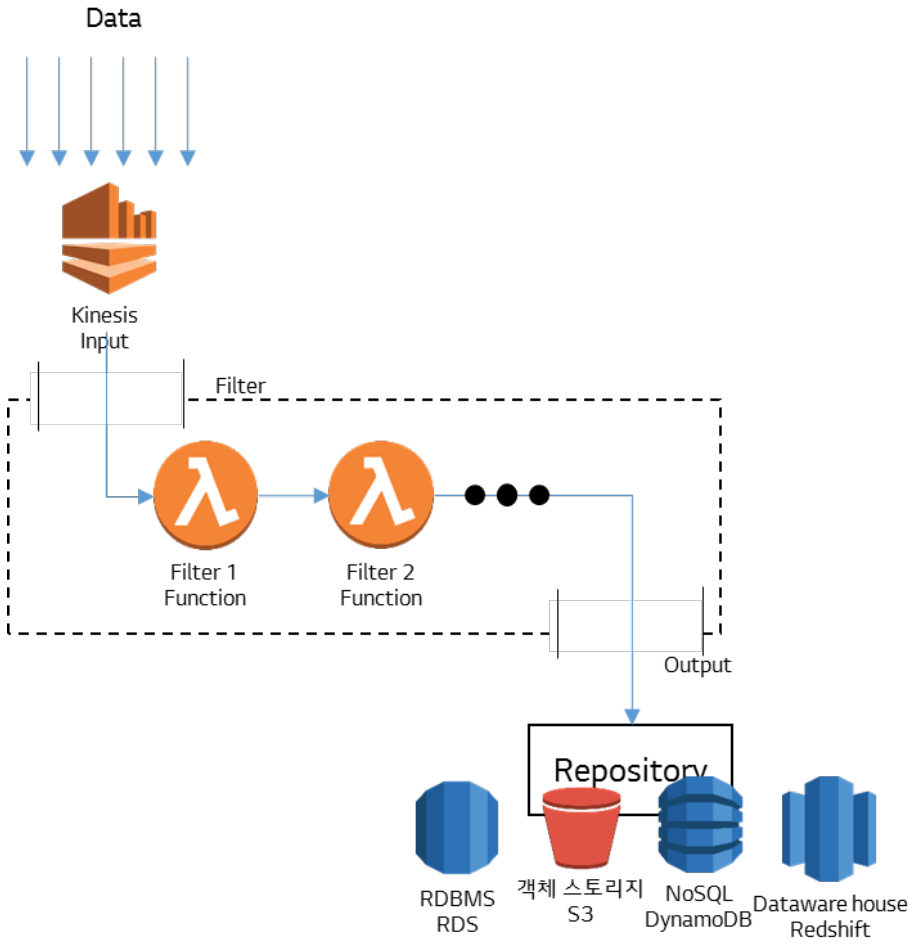

그림. 실시간 스트림 데이터 처리 AWS 아키텍처

1.2.2.3 타 업체 사례

회사명	회사소개	사용사례
Thomson Reuters	금융,법률,지적재산,과학 등 전문정보 제공	스트림 데이터 실시간 처리로 생산성 향상, 많은 스트림 데이터를 손실없이 저장하여 안정성 확보
Bustle	여성전용 뉴스, 엔터테이먼트, 패션 웹사이트	스트림 데이터 실시간 처리로 84% 비용 절감 효과, 유지보수 인력이 절반으로 줄임

회사명	회사소개	사용사례
MLB Advanced Media	미국 메이저리그 야구	야구 경기 데이터 실시간 처리, 레이더 시스템으로 공의 위치 초당 2,000회 샘플링, 선수 위치 초당 30회 샘플링샘플링 된 데이터를 가공 (투구, 가속도, 최고 도루속도 등의 지표 생성)할 때 서버리스 컴퓨팅 사용 (하루 최대 15 경기)하여 광범위한 IT 리소스 투자비 감소

1.2.3 서버리스 백엔드 아키텍처

서버리스 컴퓨팅으로 RESTful API 를 구성하여 웹사이트 백엔드, 모바일 백엔드, IoT 백엔드, 기존 시스템에 기능 추가, Proxy API 서버 등 다양한 백엔드로 활용할 수 있다.

1.2.3.1 AWS 서비스 구성

Layer	AWS 서비스	특징
Presentation	Route53	DNS 서비스
	CloudFront	CDN 서비스 DDoS 방지 서비스인 Shield Standard(무료)와 자동으로 통합되어 Layer3/4 공격을 보호함 SSL/TLS 서비스인 ACM을 무료로 사용
	S3	객체 스토리지 서비스, 99.99% 이상의 가용성 보장 HTML, css, js 파일을 이용해 정적 웹 호스팅으로 사용 가능 CloudFront를 통해서만 S3에 접근할 수 있게 처리하여 보안성 강화
	EC2 + ELB	가상머신과 로드밸런서
	IoT	AWS IoT 서비스들
Business	API Gateway	API 서비스로 HTTP 호출 CORS[Cross Origin Resource Sharing] 기능 제공 최대 수십만개의 동시 API 처리

Layer	AWS 서비스	특징
	Lambda	서버리스 컴퓨팅 서비스로 API Gateway와 연결
Data	DynamoDB	NoSQL 서비스
	RDS	관계형 데이터베이스 서비스
	EC2 DB	가상머신에 DB를 설치하여 사용

1.2.3.2 AWS 서비스 구성

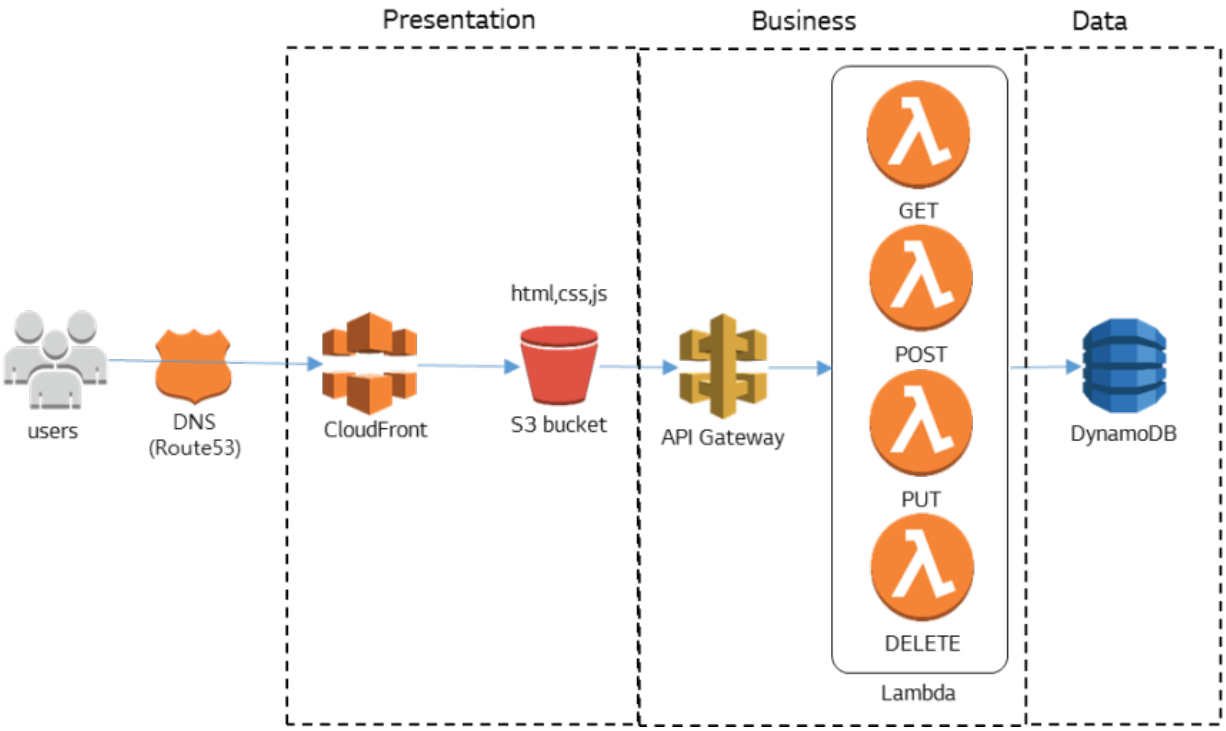

그림. 서버리스3-Tier어플리케이션 AWS아키텍처

1.2.3.3 타 업체 사례

회사명	회사소개	사용사례
Autodesk	CAD 소프트웨어 회사	전 세계 지사별 AWS Account 관리를 서버리스 컴퓨팅을 활용하여 신규 Account 생성에 2주에서 10분으로 절감
guardian News and Media	뉴스 미디어	신문을 제대로 배달하기 위한 구독자 목록 확인에 많은 시간이 소요되거나 오류 발생하던 것을 워크플로우 서비스로 서버리스 컴퓨팅을 순차적으로 처리하여 구독자 목록을 정상적으로 처리하고 주당 처리 시간의 80% 감소
Financial Engines	투자 자문	50개의 IPO 서버를 4개의 Lambda 로 변경하여 비용절감
Alt/S	데이터 분석	서버리스 마이크로 서비스 아키텍처로 구축하여 서버 관리 비용 절감
VidRoll	광고	동영상 광고 트랜스코딩, 실시간 광고 입찰을 위한 서버리스 백엔드를 구축하여 비용 절감, 생산성 증대, 유지보수 인력 최소화
Localytics	웹앱, 모바일 앱 분석	수많은 고객사의 데이터를 분석하기 위해 고객이 생기면 별도 서비스를 위한 인프라를 프로비저닝하는 대신 서버리스 컴퓨팅을 활용하여 생산성 향상, 추가 마이크로서비스의 프로비저닝 시간을 절감

1.2.4 타이머 기반 아키텍처

타이머를 이용하여 일정 시간마다 이루어져야 하는 비즈니스 요구사항을 충족시킬 수 있으며 운영 자동화나 배치 작업, 주기적인 CI/CD 등에 주로 사용된다.

1.2.4.1 AWS 서비스 구성

분류	AWS 서비스	특징
Timer	CloudWatch Event rule	일정 시간 또는 지정된 시간에 이벤트를 호출
Function	Lambda	서버리스 컴퓨팅 서비스로 API Gateway와 연결

분류	AWS 서비스	특징
Output	EC2 + AMI	가상머신, 가상머신 이미지
	SES	메일 서비스

1.2.4.2 AWS 아키텍처 구성도

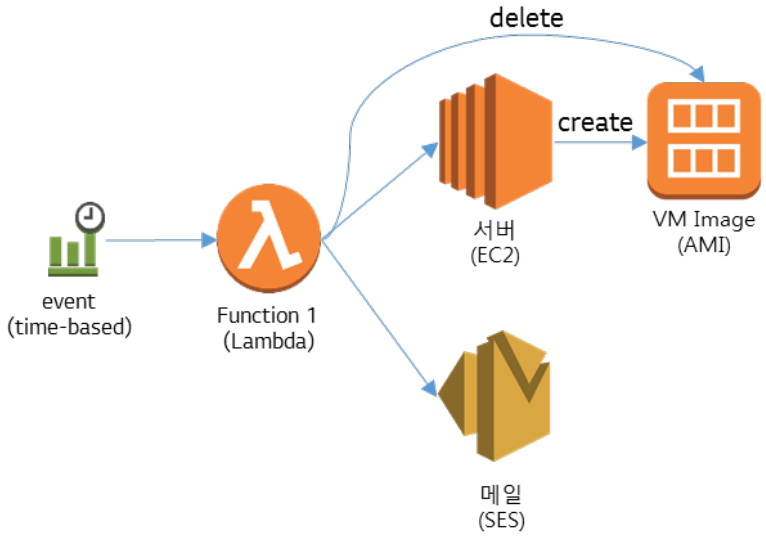

그림. 타이머 기반 서버리스 AWS 아키텍처

1.3 아키텍처 설계

1.3.1 Lambda 사이징 설계

Lambda에서 성능을 선택할 수 있는 것은 메모리로 이에 따라 CPU 성능이 자동으로 적용된다. 메모리는 128MB 부터 3008MB 까지 64MB 단위로 구분되며 메모리가 256MB인 Lambda의 CPU 성능은 128MB인 Lambda의 CPU보다 2배 높은 성능이 적용된다.
가격 정책이 메모리-GB 초 이기에 높은 메모리로 짧은 시간을 실행하는 것이 적은 메모리로 많은 시간을 실행하는 것보다 저렴하니 사전에 적합한 메모리 사이즈를 선택할 수 있게 테스트를 수행해야 한다.

1.3.2 인터페이스 설계

서버리스 컴퓨팅은 단독으로 수행할 수 없고 호출하는 매개체, 즉 이벤트 트리거가 필요하며, CSP다양한 서비스와 쉽게 연계하여 사용할 수 있게 되어 있어 비즈니스 요구사항에 맞게 이벤트 트리거를 사용해야 한다.

1.3.3 스토리지 설계

서버리스 컴퓨팅은 휘발성 스토리지를 가지고 있어 함수 수행 시에 사용할 수 있으나 함수가 종료되면 사라지기에 필요에 맞게 별도의 저장소에 처리 결과를 보관해야 한다. 별도의 저장소는 서버리스 컴퓨팅 전용이 아니며 기존에 사용하던 서비스들로서 상황에 맞게 서비스를 선택해야 한다.

1.3.4 네크워크 설계

함수가 다른 시스템의 데이터를 가져오거나 결과 값을 저장할 때 사용할 수 있는 서비스들의 네트워크는 외부망과 내부망으로 구분할 수 있다. 내부망에 함수가 위치하면 제약사항이 더 많아지니 꼭 필요한 경우가 아니라면 외부망에 위치해야 한다.

1.3.5 보안 설계

이벤트 트리거 서비스들이 함수를 실행하기 위해서는 인증 절차가 필요하며, 함수에서 다른 서비스와 연결할 때에도 인증 또는 권한이 있어야 수행할 수 있다. 모든 것을 수행할 수 있는 권한을 부여하면 보안상 안전하지 못하므로 반드시 필요한 권한만 적용해야 한다.

1.3.6 서비스 별 제한 검토

서버리스 컴퓨팅 뿐 아니라 호출 서비스들도 대부분 PaaS 서비스로 제한사항이 있는데, 예를 들어 API Gateway의 경우 초당 처리할 수 있는 요청 수에 제한이 있어 Lambda가 아무리 확장하더라도 사전에 앞 단에서 처리가 안되는 상황이 발생할 수 있다. 이런 상황에 대비하여 사전에 시스템에서 사용할 모든 제한 값을 파악하여 확장하거나 별도의 방안을 고려해야 하는데, 특정 제한 값은 별도 요청을 해도 확장할 수 없으니 반드시 사전에 확인해두어야 한다.

1.3.7 동시 실행 수

서버리스 컴퓨팅은 동시에 실행할 수 있는 수가 존재하며 동시실행 수도 자동으로 확장하지만 시간이 지연되어 원활한 서비스를 제공하지 못할 수도 있으니 이를 고려하여 미리 동시 실행 수 값을 늘리는 방안을 검토해야 한다.

1.3.8 콜드스타트 지연시간

콜드스타트로 인해 발생하는 지연시간이 발생하며 이를 회피하는 방안을 고려해야 한다.

2 서버리스 컴퓨팅 아키텍처 구성

2.1 개요

서버리스 컴퓨팅의 논리 아키텍처는 이벤트 트리거, 처리, 저장의 3개 layer 로 구성되어 있다. 이벤트 트리거 layer 는 서버리스 컴퓨팅의 구현체인 함수를 호출하는 매개체를 의미하며, 이벤트를 발생시키는 주체로는 http, 메시지 큐 등이 있으며 목차 [6.2.2 이벤트 트리거]에 상세히 설명되어 있다.

이벤트 처리 layer 는 서버리스 컴퓨팅의 구현체인 함수가 위치하는 곳을 의미하며, 함수 단독 수행이나 여러 함수가 순차적으로 수행할 수 있으며, 여러 함수 순차 수행할 경우는 효율적인 수행과 관리를 위해서 워크플로우 서비스를 활용하는 것이 좋다.

서버리스 컴퓨팅은 stateless한 특성이 있으며 처리 과정이나 결과를 저장하지 않는다. 이벤트 저장 layer 는 stateless한 특성을 보완하는 계층으로 함수의 처리 과정이나 결과를 저장하는 persistent가 존재하는 계층이다.

본 장에서는 각 Layer에 속하는 서비스들을 소개하되 서비스의 상세 정보는 포함하지 않는다.

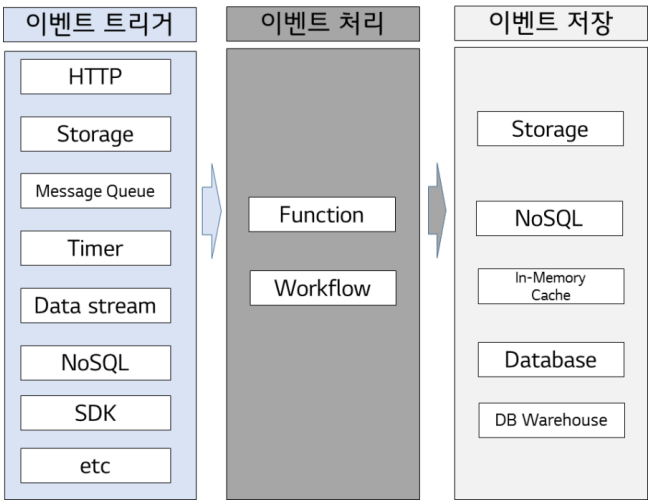

그림. 서버리스 컴퓨팅 논리 아키텍처 예시

2.2 이벤트 트리거 계층

이벤트 트리거로 사용이 가능한 서비스들은 서비리스 컴퓨팅에서만 사용이 가능한 전용서비스들은 아니며 일반적으로 클라우드 PaaS에서 제공되는 서비스들을 이벤트 트리거로 사용 할 수 있다.

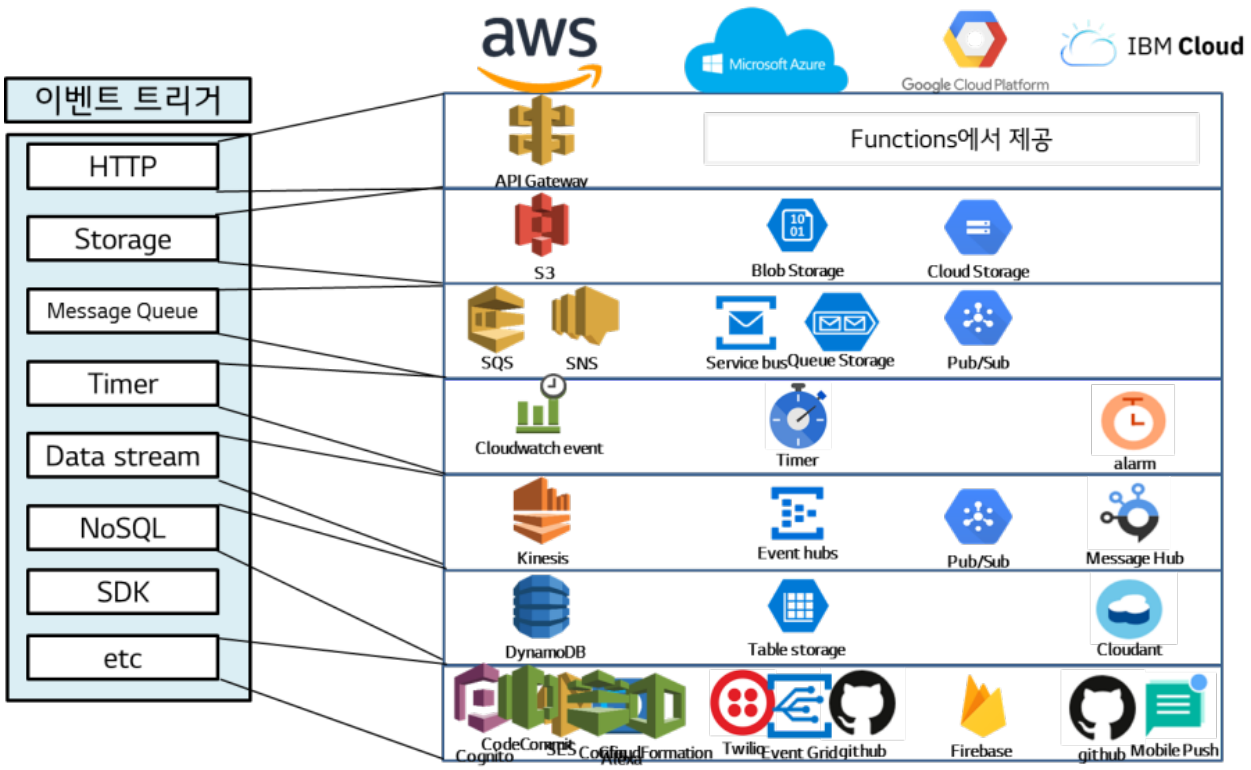

그림. CSP 별 이벤트 트리거 서비스

2.2.1 HTTP

GET/POST/PUT/DELETE 등의 HTTP메서드로 서버리스 컴퓨팅 함수를 트리거할 때 사용하는 방식으로 HTTP를 통해 타 클라우드 서비스나 On-premise 서비스와도 통신이 가능하며 이 경우 인증키를 사용해야 한다. AWS를 제외한 나머지 CSP의 서버리스 컴퓨팅들은 HTTP 트리거를 기본 제공한다.

공급사	트리거	설명
AWS	API Gateway	API 관리형 서비스로 비용 발생
Azure	-	기본 제공
Google	-	기본 제공
IBM	-	기본 제공

2.2.2 Storage

객체 스토리지의 파일 변화를 감지하여 서버리스 컴퓨팅 함수를 트리거할 때 사용하는 방식으로 이미지 변환이나 비디오 인코딩, 로그 파일 분석 용도로 활용할 때 사용한다.

공급사	트리거	설명
AWS	S3	객체 스토리지 서비스. Prefix, 파일 확장자 등의 세부 사항 설정이 가능
Azure	Blob Storage	객체 스토리지 서비스 파일이 생기거나 삭제될 때 함수를 호출 함수 실행이 5번 실패하면 webjobs-blobtrigger-poison 이라는 메시지 큐에 메시지를 추가. 이를 이용해 별도의 실패 처리해야 함 Azure Function 1.x, 2.x 지원
	외부파일	3rd party 파일 SaaS 공급자(Dropbox 또는 Google Drive등)의 파일 변환에 함수를 호출 Azure Function 1.x 지원
Google	Google Cloud Storage	Storage의 기존 객체 변견 알림으로 트리거 Google Pub/Sub 서비스와 연계해서도 사용가능
IBM	-	-

2.2.3 Message Queue

메시지 큐에 메시지가 게시되면 서버리스 컴퓨팅 함수를 트리거할 때 사용하는 방식으로 순차적으로 처리되지는 않지만 서버리스 컴퓨팅에서 호출한 메시지를 완료처리하거나 메시지 만료기간이 지날 때까지 재시도하기에 안정성이 중요한 기능에서 사용한다.

공급사	트리거	설명
AWS	SQS	메시지 큐 서비스 메시지 수신 시 트리거, 순차적 처리 보장 안함 순차적으로 처리할 수 있는 FIFO 기능은 현재 한정된 Region (US East (N. Virginia), US East (Ohio), US West (Oregon), and EU (Ireland) Regions)에만 제공되며 Seoul Region에는 제공 안함.

공급사	트리거	설명
	SNS	Pub/Sub* 서비스 메시지(topic) 게시 시 호출 AWS 계정 간 호출 가능하여 하나의 계정으로 여러 계정 관리에 활용가능
Azure	Queue Storage	메시지 큐 서비스 메시지 수신 시 트리거, 순차적 처리 보장 안함 5번 실패한 경우 [queue-name]-poison 큐에 메시지 추가 Azure Function 1.x, 2.x 지원
	Service bus	FIFO 메시지, Pub/Sub 서비스 메시지 수신 시 트리거 Azure Function 1.x, 2.x 지원
Google	Pub/Sub	메세지큐, Pub/Sub, 스트림 데이터 전달 서비스 Topic 수신 시 트리거
IBM	-	-

Pub/Sub (Publish/Subscribe) : 발신자의 메시지를 정해진 범주에 따라 구독을 신청한 수신자에게 전달하는 비동기 메시지 서비스.

2.2.4 Timer

일정 주기로 서버리스 컴퓨팅 함수를 트리거할 때 사용하는 방식으로, 예를 들어 매일 5시나 월-금으로 1시간마다 1회씩 함수를 트리거할 수 있다.

공급사	**트리거**	**설명**
AWS	CloudWatch Event	타이머 서비스 일정 주기 트리거로 사용. Cron 식으로 수행
Azure	Timer	타이머 서비스 일정 주기 트리거로 사용. Cron 식으로 수행 Azure Function 1.x, 2.x 지원
Google	-	-
IBM	/whisk.system/alarms	/whisk.system/alarms 패키지로 지정된 시간에 트리거

2.2.5 Data Stream

스트림 데이터 수집 서비스에 데이터가 생성되면 서버리스 컴퓨팅 함수를 트리거하는 방식으로 로그나 사용자의 웹사이트 클릭 등을 실시간으로 분석하여 대응할 수 있다.

공급사	트리거	설명
AWS	Kinesis	실시간 스트림 데이터 서비스 데이터 수집 시 트리거
Azure	Event Hubs	실시간 스트림 데이터 서비스 데이터 수집 시 트리거 Azure Function 1.x, 2.x 지원
Google	Pub/Sub	메세지큐, Pub/Sub, 스트림 데이터 전달 서비스 데이터 수집 시 트리거
IBM	Message Hub	실시간 스트림 데이터 서비스

2.2.6 NoSQL

NoSQL 에 데이터 입력/출력 시 서버리스 컴퓨팅 함수를 트리거하는 방식으로 데이터 변화를 감지하여 실시간으로 처리할 때 사용한다.

공급사	트리거	설명
AWS	DynamoDB	NoSQL 서비스 스트림 기능을 활성화해야 동작 데이터 변화 시 트리거
Azure	Table Storage	NoSQL 서비스 데이터 변화 시 트리거 단일 region의 대용량 처리 시 유용 Azure Function 1.x, 2.x 지원
	CosmosDB	NoSQL 서비스 데이터 변화 시 트리거 멀티 region에 효율적인 높은 처리량에 유용 Azure Function 1.x, 2.x 지원
Google	-	-
IBM	Cloudant	NoSQL 서비스 데이터 변환 시 트리거

2.2.7 직접호출

CSP별로 서버리스 컴퓨팅 함수를 직접 호출할 수 있다.

2.3 이벤트 처리 계층

이벤트 트리거로 인해 데이터를 처리할 수 있는 서버리스 컴퓨팅 함수들이 호출되고 미리 작성된 코드로 트리거에서 보낸 이벤트 소스를 파싱하여 데이터를 처리한다.

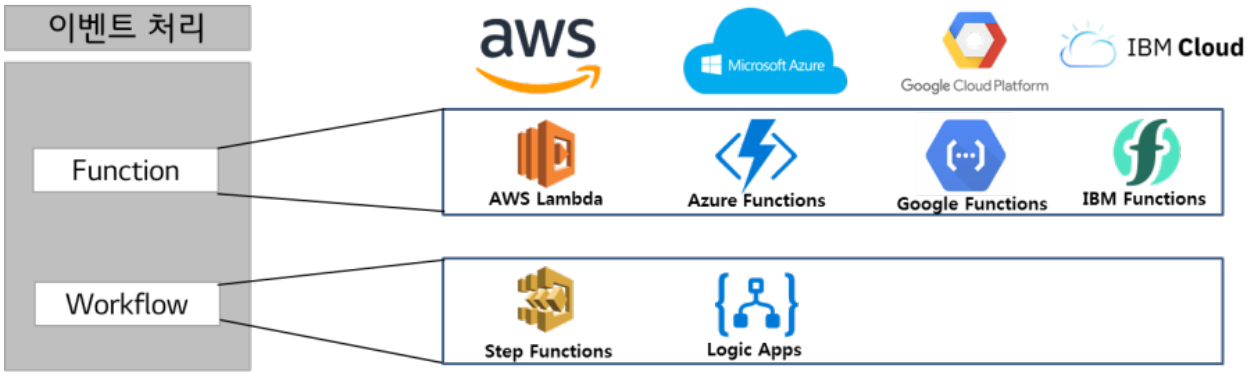

그림. CSP 별 이벤트 처리 서비스

2.3.1 구현 단위

분류	설명
함수 기능 단위	함수는 최대 구동 제한시간이 존재하여 작은 기능으로 코드 작성 필요 하나의 함수는 하나의 기능만 수행하도록 작성
함수 동작 시간	함수 동작 시간은 기본 3초 미만, 최대 동작 시간은 콜드스타트를 고려하여 10초 정도로 구성
함수 파일 단위	하나의 함수는 여러 파일로 구성이 가능 여러 라이브러리들을 추가하여 zip으로 압축 후 클라우드 상에 업로드
분류	
함수 기능 단위	**함수는 최대 구동 제한시간이 존재하여 작은 기능으로 코드 작성 필요 하나의수는 하나의 기능만 수행하도록 작성**
함수 동작 시간	**함수 동작 시간은 기본 3초 미만, 최대 동작 시간은 콜드스타트를 고려하여 10초 정도로 구성**

함수 파일 단위	**하나의 함수는 여러 파일로 구성이 가능** **여러 라이브러리들을 추가하여 zip으로 압축 후 클라우드 상에 업로드**

2.3.2 워크플로우 서비스

여러 함수들을 비즈니스 워크플로우에 맞게 설정할 경우가 있는데, 서버리스 컴퓨팅 앞 단에서 직접 구현하거나 CSP에서 제공하는 서비스를 이용할 수 있다.
아래는 비즈니스 워크플로우에 맞춰 서버리스 컴퓨팅을 각 단계 별로 실행할 수 있게 통합하는 서비스들에 대한 정보이다.

공급사	트리거	설명
AWS	Step Functions	여러 AWS 서비스들을 워크플로로 연결할 수 있는 서비스 AWS Lambda 지원 Lambda를 상황에 맞게 분기 처리할 수 있게 연결
Azure	Logic Apps	앱, 데이터, 시스템 및 서비스를 통합하는 서비스로 on-premise와 cloud 서비스 간 연결이나 Office 365 와도 통합 가능 Azure Functions 지원 Functions를 각 단계 별로 실행할 수 있게 연결
Google	-	-
IBM	-	-

2.4 이벤트 저장 계층

서버리스 컴퓨팅은 호출 시 컨테이너가 할당되어 로직을 수행하고 종료되면 할당되었던 컨테이너를 반환하기에 데이터를 저장할 수 없어 필요 시 별도의 저장소에 처리 결과를 보관해야한다. 별도의 저장소는 서버리스 컴퓨팅 전용이 아니며 기존에 사용하던 서비스들로 상황에 맞게 선택하여 사용할 수 있다.

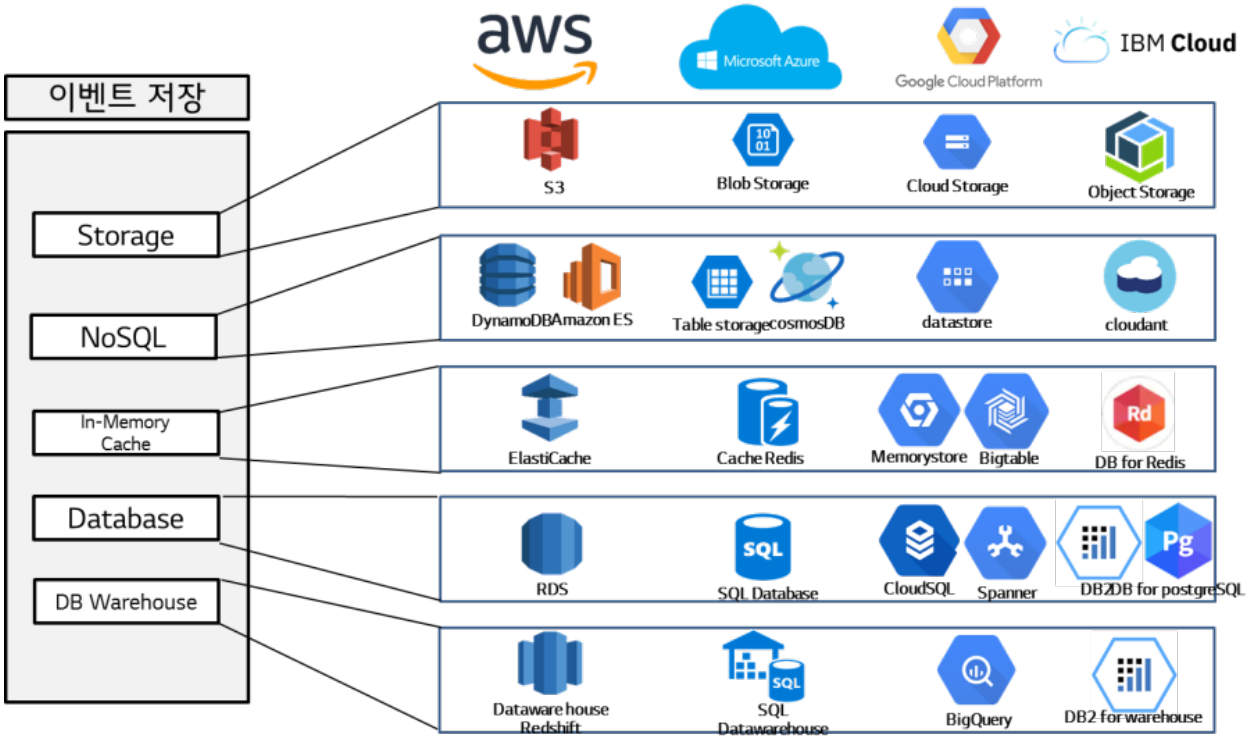

그림. CSP 별 이벤트 저장 서비스

2.4.1 Storage

파일 형태의 결과 값은 비용이 다른 서비스에 비해 저렴하고 안정성이 보장되는 객체 스토리지 서비스에 보관하는 것이 효율적이다.

공급사	트리거	설명
AWS	S3	객체 스토리지 서비스. Prefix, 파일 확장자 등의 세부 사항 설정이 가능
	EC2 + EBS	가상머신의 스토리지에 결과값을 보관 안정화를 위해 이중화 필요 내부망의 서버에 접근해야 한다면 Lambda를 VPC 내부에 위치해야 함
Azure	Blob Storage	객체 스토리지 서비스
	VM + Storage Disk	가상머신의 스토리지에 결과값을 보관 안정화를 위해 이중화 필요
Google	Google Cloud Storage	객체 스토리지 서비스

공급사	트리거	설명
IBM	IBM Object Storage	객체 스토리지 서비스

2.4.2 NoSQL

처리 결과를 NoSQL 에 저장하는 방식으로 데이터 보관 시 사용한다.

공급사	트리거	설명
AWS	DynamoDB	NoSQL 서비스, 확장성 및 고가용성 제공
	ElasticSearch	NoSQL 서비스 ElasticSearch(ES) 를 가상서버에 설치하거나 AWS 에서 PaaS로 제공하는 ES를 사용
Azure	Table Storage	NoSQL 서비스, 단일 region의 대용량 처리 시 유용
	CosmosDB	NoSQL 서비스, 멀티 region에 효율적인 높은 처리량에 유용
Google	Datastore	NoSQL 서비스, 확장성 및 고가용성 제공
	Bigtable	NoSQL 서비스, 대규모 확장형 서비스
IBM	Cloudant	NoSQL 서비스

2.4.3 In-Memory Cache

In-Memory Cache 에 데이터를 캐싱하여 빠르게 조회할 때 사용한다.

공급사	트리거	설명
AWS	ElastiCache	In-memory 서비스 Redis와 memcached로 캐시 용도로 사용
Azure	Azure Redis Cache	In-memory 서비스. Redis 지원
Google	Memorystore for Redis	In-memory 서비스, Redis 지원
IBM	Database for Redis	In-memory 서비스, Redis 지원

2.4.4 Database

처리 결과를 DB에 저장하는 방식으로 데이터 보관 시 사용한다.

공급사	트리거	설명
AWS	RDS	RDBMS 관리형서비스로 Oracle, MS-SQL, MySQL, MariaDB, AuroraDB를 지원 대부분의 DB는 가상 네트워크 내부망에 위치하며 Lambda도 동일하게 내부망에 위치해야 통신 가능 AuroraDB는 특정 region에서 serverlesss 로 제공
	EC2 + EBS	가상머신에 RDBMS를 설치하여 사용
Azure	SQL Database	SQL DB 관리형 서비스로 Azure-SQL, MySQL, PostgreSQL 지원
	VM + Storage Disk	가상머신의 RDBMS를 설치하여 사용
Google	Cloud SQL	RDBMS 서비스로 PostgreSQL, MySQL 지원
	Cloud Spanner	RDBMS와 NoSQL의 장점을 결합한 일관성을 갖추고 수평확장이 가능한 DB
IBM	Db2 on cloud	RDBMS 서비스
	DB for postgreSQL	RDBMS 서비스로 PostgreSQL 지원

2.4.5 Datawarehouse

페타데이터 규모의 데이터를 통합 관리할 때 사용하며 함수에서 처리 전 원본 데이터를 저장하거나 가공된 데이터를 저장하여 사용한다.

공급사	트리거	설명
AWS	Redshift	PostgreSQL 기반의 Data warehouse 관리형 서비스
Azure	SQL Data Warehouse	Data warehouse 관리형 서비스
Google	Bigquery	머신러닝이 내장된 Data warehouse 관리형 서비스

공급사	트리거	설명
IBM	Db2 Warehouse on Cloud	Db2 Data warehouse 관리형 서비스

3 서버리스 컴퓨팅 아키텍처 설계

가상의 시나리오를 바탕으로 서버리스 컴퓨팅 아키텍처를 설계한다.

3.1 요구사항 분석

요구사항	As-Is
서버 추가로 인해 많은 비용 발생	관리하고 있는 IoT 장비가 많아질수록 고사양의 서버가 추가로 필요하여 많은 비용이 발생
IoT 기기의 사진촬영 시 변환작업에 높은 부하 발생	IoT 기기에서 사진을 촬영하고 이미지 파일을 가상서버 스토리지에 저장하는데 사이즈가 크고 이미지는 워터마크를 처리하고 썸네일 처리를 진행하는데 이 과정에서 서버에 높은 부하가 발생
특정 기능 안정성 보장 필요	요청이 오면 시간이 걸리더라도 반드시 처리가 되어야 하는 기능이 알 수 없는 이유로 요청 자체가 사라진다는 불만사항 발생
IoT의 데이터 실시간 분석 필요	IoT에서 발생하는 데이터를 수집해서 사용량이 적은 시간대에 배치 잡으로 진행하기에 하루 늦은 분석 데이터를 사용, 실시간 분석 및 대응을 요구
유휴자원 비용증가	접속량이 적은 시간대에도 고사양의 서버가 동작하여 많은 비용이 발생
관리 비용 상승	가상서버에 운영체재 패치, 보안 패치 등의 여러가지 관리 비용 발생
성능 오버헤드	불규칙한 트래픽으로 오버 사이즈의 서버를 사용하고 스토리지 크기도 크게 사용하여 많은 비용 지출
빠른 개발 요청	새로운 버전을 빠르게 요청함
불예측적인 대용량 트래픽 처리시 불안정	Autoscaling 은 불예측적인 대용량 트래픽을 처리하기에 적합하지 않음 Autoscaling 은 지정된 시간동안 특정 메트릭이 임계치 이상 사용되었을 때 자동으로 서버를 추가하는 Scale out이 동작 Scale out 된 서버가 정상적으로 올라오는데 수분의 시간이 소요, 미들웨어가 구동되는데도 수분의 시간이 소요되므로 총 수분~수십분 이상이 필요하며, 구동되기 전까지의 서비스가 느려지거나 오류가 발생

3.2 As-Is 아키텍처 분석

As-Is는 AWS에 3티어로 구성된 IoT장비를 등록하고 모니터링을 수행하는 시스템으로 관리하고 있는 IoT장비가 증가함에 따라 고사양의 장비가 필요하게 되었고, 많은 서버들이 추가적으로 필요하여 이를 유지하기에는 많은 비용이 발생하는 경우로 고객은 IoT장비 증가에 능동적으로 대응하고 비용이 저렴한 신규 아키텍처를 원하고 있어 이를 서버리스 컴퓨팅을 적용하여 To-Be아키텍처로 재설계하여 기존 고려사항을 반영한 상세 설계를 하도록 한다.

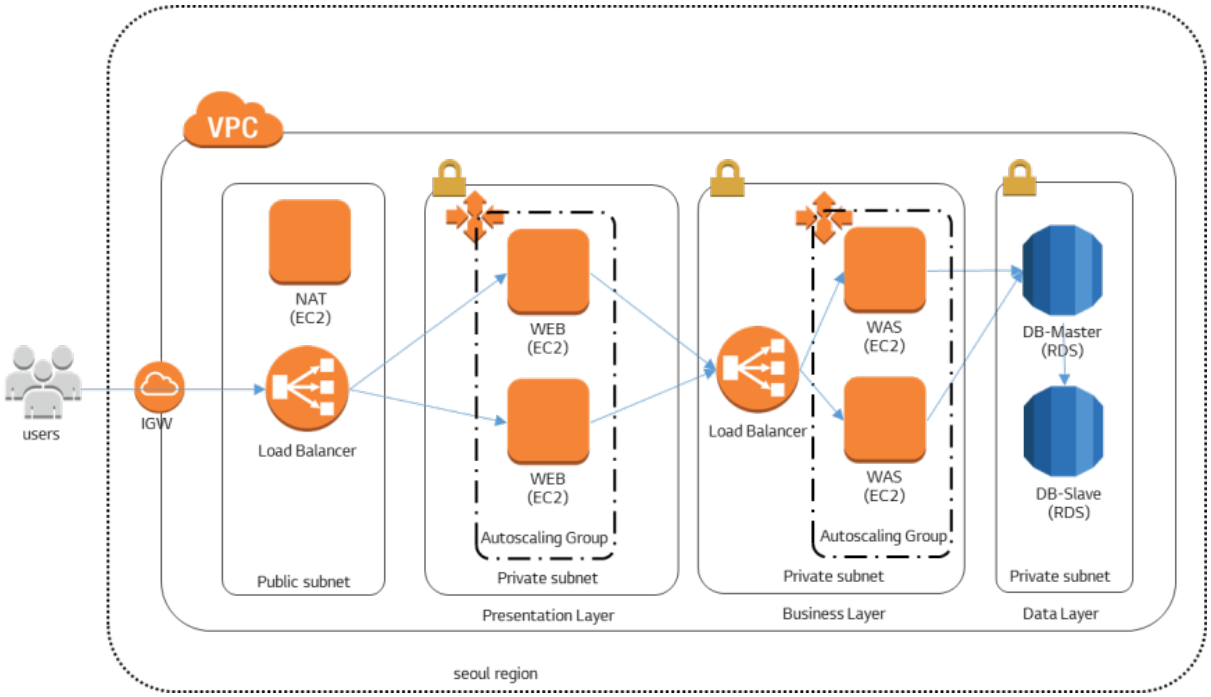

그림. As-Is 웹사이트 아키텍처

3.3 To-Be 아키텍처 구성

To-Be아키텍처에는 사용자가 많아도 부하가 크지 않은 Presentation Layer는 아키텍처 상으로 큰 변화없이 구성되어 있다. Business Layer는 부하에 의해 서버가 가장 많이 사용되어 As-Is에서 가장 큰 비용을 차지한 기능을 API Gateway와 Lambda를 이용해 비즈니스 로직을 수행하고, 파일 처리로 인한 부하를 분산하기 위해 S3 와 Lambda를 이용해 파일 처리를 수행하고, 안정성을 위해 메시지를 완료하거나 만료기간이 종료되기까지 계속 수행하는 SQS와 Lambda를 사용하고 실시간 분석을 위해 Web에서 발생하는 데이터를 실시간으로 수집하는 Kinesis와 Lambda를 연결해서 최종적으로 아래와 같은 아키텍처를 구상하였다.

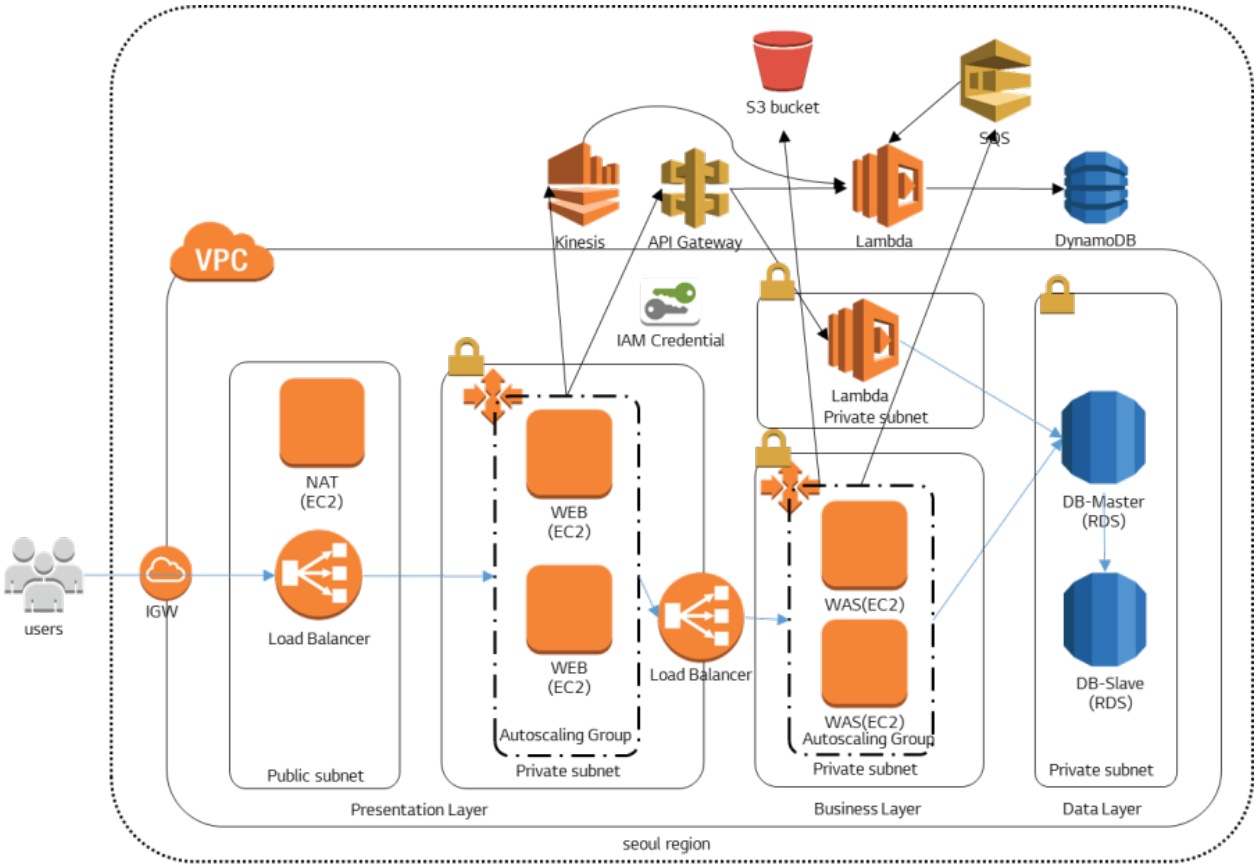

그림. To-Be 웹사이트 아키텍처

3.4 To-Be 아키텍처 상세 설계

3.4.1 인터페이스

Business Layer의 WAS 에서 발생하는 이슈사항을 해소하기 위해 특정 기능들만 서버리스 아키텍처를 적용하기로 한다. 요구 사항 중 서버 추가로 인해 많은 비용이 발생하는 것은 부하가 많이 발생하는 영역을 따로 분리하고 이를 호출하기 위해 REST API를 이용하고, 파일 처리는 S3를 이용하고, 안정성이 보장되어야 하는 것은 비동기 메시지 큐 서비스를 이용하고, 실시간 데이터 분석을 위한 데이터 수집은 Kinesis를 이용한다.

분류	서비스	설명
HTTP	API Gateway	API Gateway는 RESTFul* API 를 생성, 구성, 호스팅하는 관리형 서비스 클라우드/퍼블릭/프라이빗과 연결하는 서비스 API Gateway 29초의 timeout 존재 타 시스템과 Lambda를 이어주는 게이트웨이 역할

분류	서비스	설명
파일	S3	S3는 내구성 및 고가용성이 보장된 객체 스토리지 서비스 파일이 생성되면 이를 감지하여 Lambda를 호출하고 Lambda에서 이를 내용을 처리
메시지큐	SQS	SQS는 내구성과 확장성, 안정성이 우수한 Message Queue 서비스 Message 폴링 순서 보장 못함, (순서 보장하는 FIFO는 Seoul regio지원 안함) 안정성 보장을 위해 Message가 성공 또는 보관기간이 만료될 때까지 재시도 안정성이 필요한 기능을 Lambda로 수행할 때 사용
스트림데이터	Kinesis	Kinesis는 실시간으로 스트림 데이터를 수집하는 서비스 데이터 순서 처리 보장 데이터 처리 실패 시 데이터 보관기간까지 재시도, 처리 성공까지 다른 레코드 처리 안함 실시간 스트림 데이터를 수집하고 Lambda로 처리하여 실시간 분석이 가능

3.4.2 스토리지

3.4.2.1 임시 스토리시

Lambda는 임시 저장소인 /tmp 를 512MB까지 사용할 수 있고 함수가 종료되고 할당된 컨테이너를 반납하면 저장소 내용은 삭제된다. 하지만 Warm Start 상태일 경우 데이터가 그대로 보존되는데 데이터가 보관되는 것을 보장할 수 없으니 임시 저장소를 연속적으로 사용하는 로직은 작성하지 않아야 한다.
아래 예시는 Lambda의 임시저장소를 활용하는 예시로 임시 저장소 파일 목록 확인과 쓰기, 읽기로 되어 있고 쓰기는 /tmp 폴더에 존재하는 파일 수로 파일명을 생성한다. 3차까지 1~2분 텀으로 실행할 때는 동일한 컨테이너에서 실행하여 파일 갯수가 늘어난 것을 확인할 수 있으나 약 1시간 뒤에 실행한 4차에서는 새로운 컨테이너가 할당되어 파일이 존재하지 않는 것을 확인할 수 있다.

```
// Lambda 임시저장소 처리 예시
var fs = require("fs");

exports.handler = (event, context, callback) => {
var path = "/tmp";
// list
var files = fs.readdirSync(path);
console.log("list:" + files);

var newFileNm = path + "/jk" + files.length + ".txt";

// write
console.log("new_write: " + newFileNm);
var writeStream = fs.createWriteStream(newFileNm);
writeStream.write("hello write");
writeStream.end();

// read
fs.readFile(newFileNm, 'utf-8', readData);
};

function readData(err, data){
if (err != null)
console.log("err: " + err);
else
console.log("read: " + data);
}

// 결과 1차
2018-10-01T02:34:38.558Z 8acfbc5e-c522-11e8-85fc-6ff14ffb22b3list:
2018-10-01T02:34:38.559Z 8acfbc5e-c522-11e8-85fc-6ff14ffb22b3new_write: /tmp/jk0.txt
2018-10-01T02:34:38.636Z 8acfbc5e-c522-11e8-85fc-6ff14ffb22b3read: hello write

// 결과 2차
2018-10-01T02:35:42.976Z b154937a-c522-11e8-b291-b1596881d45flist:jk0.txt
2018-10-01T02:35:42.976Z b154937a-c522-11e8-b291-b1596881d45fnew_write: /tmp/jk1.txt
2018-10-01T02:35:42.996Z b154937a-c522-11e8-b291-b1596881d45fread: hello write

// 결과 3차
2018-10-01T02:35:54.341Z b81d7be7-c522-11e8-b879-9b22a9f38068list:jk0.txt,jk1.txt
2018-10-01T02:35:54.341Z b81d7be7-c522-11e8-b879-9b22a9f38068new_write: /tmp/jk2.txt
2018-10-01T02:35:54.356Z b81d7be7-c522-11e8-b879-9b22a9f38068read: hello write

// 결과 4차
2018-10-01T03:33:14.467Z ba70e88b-c52a-11e8-bfc3-e5d7821c567d list:
2018-10-01T03:33:14.467Z ba70e88b-c52a-11e8-bfc3-e5d7821c567dnew_write: /tmp/jk0.txt
2018-10-01T03:33:14.544Z ba70e88b-c52a-11e8-bfc3-e5d7821c567dread: hello write
```

3.4.2.2 외부 스토리지

Lambda는 요청이 발생하면 내부적으로 컨테이너를 할당받아 코드를 수행하고 종료되면 일정 시간이 지난 후 할당받은 컨테이너를 반납하여 데이터를 보관해야 한다면 별도의 저장소를 이용해야 한다. 외부 스토리지 서비스는 목차 [6.2.3 데이터 저장]에 상세히 설명되어 있다.

3.4.2.3 To-Be 스토리지 설계

파일을 업로드하고 바로 변환하는 처리가 필요할 때는 S3 객체 스토리지에 이벤트 등록을 하여 Lambda를 연결하면 파일 변화를 감지해 Lambda가 수행된다. 변화가 감지된 파일을 S3에서 불러와 임시 스토리지에 보관하여 함수 수행 중에 사용하고 처리가 완료된 파일을 S3에 저장하는 식으로 활용할 수 있다.
또한 데이터베이스는 인터넷을 통해 연결할 수 있는 DynamoDB나 VPC 내부망에 있는 RDS등과 연결할 수 있다.
DynamoDB 는 사양을 선택하는 것은 아니지만 Table별로 초당 읽고 쓰는 Capacity가 각각 존재하며 사전에 이를 파악하여 설정해두어야 한다.
DynamoDB의 읽기는 최근에 쓰기가 된 데이터를 가져오는지 보장할 수 없는 Strongly Consistent Read와 가장 최근 데이터까지 가져오는지 보장하는 Eventually Consistent Read로 구분할 수 있다.
읽기 Capacity 1은 초당 Strongly Consistenct 2회나 Eventually Consistent Read 1회 수행한 것을 의미하고 읽기 데이터가 4KB가 넘을 경우 그 만큼의 Capacity 가 필요하게 된다.
쓰기 Capacity 1은 초당 1KB 쓰기 1회를 의미한다.
사전에 이를 고려하여 예약 값을 지정하거나Capacity Autoscaling을 지정해서 사용할 수 있다.

3.4.3 네트워크

3.4.3.1 외부망

인터넷에 오픈된 AWS서비스들이나 AWS외 서비스들이 이에 포함되며 Lambda는 AWS 서비스들에 접근할 때 IAM role을 이용한다. AWS외 서비스들은 HTTP와 API Key를 이용해 접근한다. Lambda에 API 통신을 위한 Key가 필요한 경우 환경변수에 KMS로 API-Key를 암호화해서 이용해야 한다.

3.4.3.2 내부망

내부망에 있는 가상머신이나 DB를 Lambda와 연결하려면 가상네트워크 서비스인 VPC를 이용해야 한다. VPC의 내부 서브넷(Private subnet)에 Lambda를 위치하며 이를 VPC Lambda라 한다. VPC Lambda가 외부망과 연결하려면 외부 서브넷(Public subnet)에는 NAT(가상머신 NAT나 NAT Gateway 서비스를 이용)가 필요하다. VPC내에 있는 Lambda는 실행 시 VPC의 가상 네트워크 카드인 ENI[Elastic Network Interfaces]가 자동으로 생성되어 연결되어 Lambda별로 개별 IP가 생성되고 이로 인한 패널티(지연시간)가 발생한다. IP 할당으로 인해 서브넷의 크기 만큼만 Lambda가 자동 확장하게되니 사전에 충분한 크기의 서브넷을 준비해야 한다. (서브넷 예, 10.0.0.0/24 -> 256개 – AWS 예약IP 5개 = 총 251개의 IP 할당 가능)

VPC Lambda를 이용한다면, 아래 그림의 ① 처럼 Lambda 전용 내부망(Private subnet)을 생성해서 다른 서비스들과 IP를 공유하지 않고 주소가 할당되도록 해야 한다.
또한, VPC에 위치할 수 없는 DynamoDB 같은 서비스와 VPC Lambda를 연결할 때는 VPC Endpoint나 NAT를 이용할 수 있다. 아래 그림의 ②와 같이 VPC Endpoint는 AWS 망에 위치해서 빠르고 안정적으로 접근할 수 있으나 VPC Endpoint 비용이 발생하고, ③ 처럼 NAT를 이용하게 되면 인터넷망을 거쳐 접속하기에 VPC Endpoint 통신에 비해 느리다.
VPC Lambda는 서브넷만큼 확장할 수 있다는 점과, ENI패널티 등으로 인해 내부망과 통신할 용도가 아니라면 고려할 필요없다.

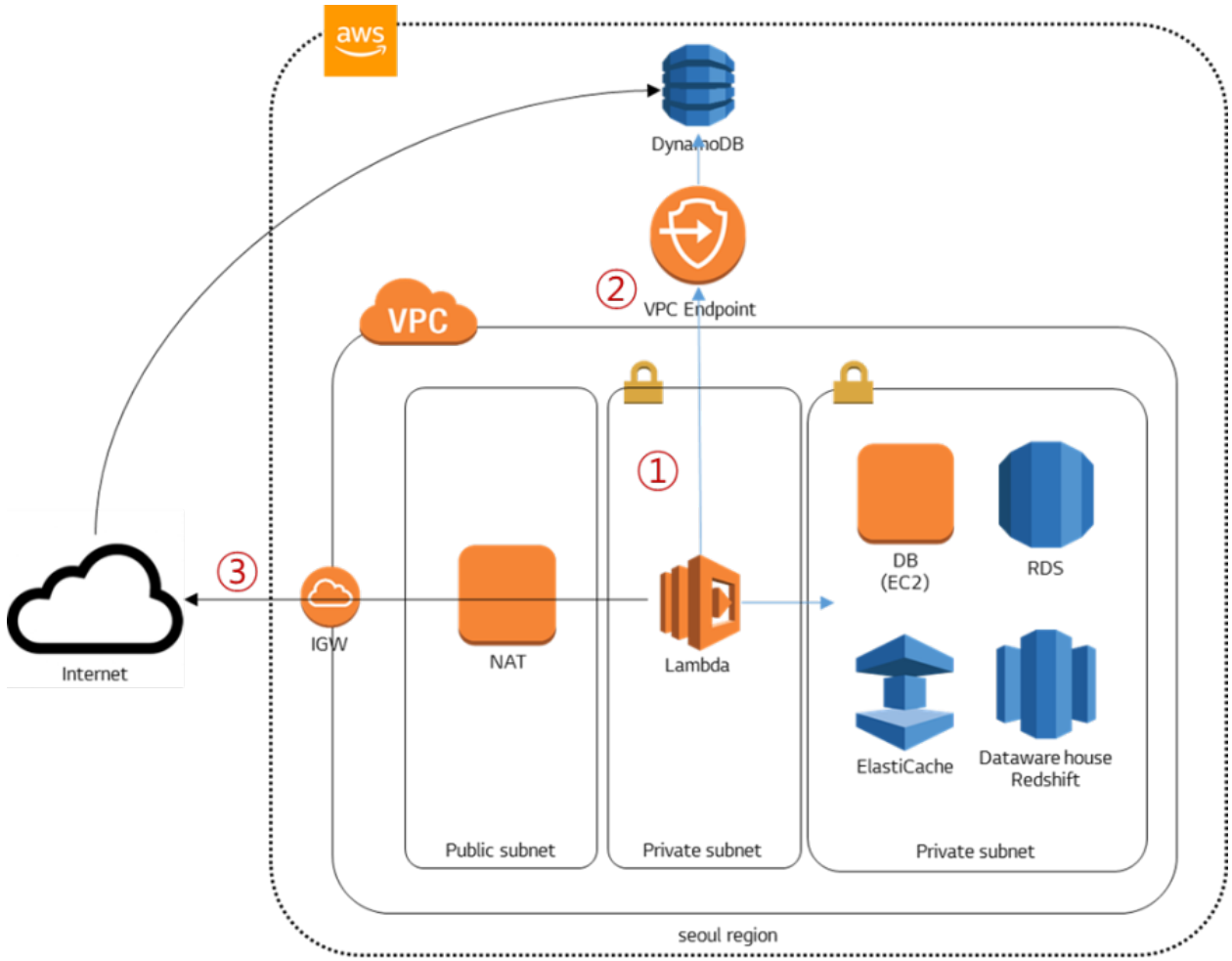

그림. 내부망 Lambda 아키텍처

3.4.3.3 To-Be 네트워크 설계

Lambda와 DynamoDB를 이용할 때는 바로 연결되고, VPC 내부망에 있는 데이터베이스와 Lambda를 연결하려면 VPC에 Lambda가 위치해야 하며, VPC Lambda에서 DynamoDB에 연결하기 위해서는 NAT나 VPC-Endpoint를 이용한다.

3.4.4 보안

Lambda를 수행하기 위한 호출 서비스들을 실행하기 위해서는 AWS IAM의 인증절차가 필요하다.

3.4.4.1 Credential로 인증 시

AccessKey와 SecretKey로 이루어진 AWS IAM Credential을 이용할 수 있으나, 유출 시 보안에 문제가 생기기에 되도록 다른 방안을 고려해야 한다.

3.4.4.2 AWS 계정 내에서 인증 시

호출할 서비스에 호출서비스에 필요한 IAM Role(권한)을 부여한다. 예를 들어 가상머신인 EC2에서 SQS 메시지 큐를 생성하여 Lambda를 수행해야 할 때 EC2에 SQS 메시지 큐 생성 권한을 가지고 있어야 한다.

3.4.4.3 AWS 타 계정에서 인증 시

AWS 계정 간에는 임시보안자격증명을 생성하는 STS Security Token Service 의 AssumeRole 을 설정하여 호출 서비스를 실행한다. 임시보안자격증명은 기간이 한정되어 있고 자격 증명이 만료되면 리소스에 대한 접근을 허용하지 않는다.

3.4.4.4 타 시스템에서 인증 시

타 시스템에서는 STS로 SAML2.0로 연동하는 Enterprise identity federation이나 Amazon, Google, Facebook등의 타사 자격증명 공급자와 연동하는 Web identity federation 을 통해 인증받을 수 있다.
단, 모바일 어플리케이션은 사용자 고유의 자격증명을 생성하여 AWS서비스에 접근할 수 있는 AWS Cognito를 사용하는 것을 권장한다.

3.4.4.5 API Gateway 인증 방안

Api Gateway 권한에 리소스 정책

API Gateway의 IAM 정책을 통해 접근을 허용할 수 있는데, 특정 IP나 CIDR block에서만 API Gateway에 요청하거나 AWS VPC에서만 접근할 수 있도록 설정할 수 있다.

IAM credential

API Gateway 인증 키를 IAM Credential로 사용하는 방법이다.

도메인

특정 도메인에서의 요청만 허용하는 방식으로 API Gateway에 CORS 설정이 되어야 한다.

Cognito User pool

AWS Cognito를 이용해 인증된 사용자 모바일에서의 요청을 인증하는 방법이다.

3.4.4.6 To-Be 보안 설계

Layer	As-Is	To-Be
Presentation	Security Group으로 설정 Load Balancer는 모두 오픈 (0.0.0.0/0) Web 서버는 LB에서만 접근 허용	As-Is와 동일

Layer	As-Is	To-Be
Business	Security Group으로 설정 Load Balancer는 Web서버에서만 접근 허용 Was서버는 LB에서만 접근 허용	API Gateway 호출 시 AWS IAM을 API Key로 설정 Web서버에서 API Gateway 호출 시 API Key 사용 Web서버에서 Kinesis 를 수행할 수 있는 IAM 역할 부여 WAS서버에서 SQS 메시지를 생성하는 IAM 역할 부여 VPC Lambda를 이용하여 내부망 DB와 연결하고 Security Group을 사용
Data	Security Group으로 설정 WAS서버에서만 DB에 접근 허용	AWS 계정내 인증으로 IAM Role으로 설정 Lambda에서 DynamoDB 간에 권한 설정

3.4.5 서비스 제한요소

PaaS 서비스들의 제한 값을 확인하고 미리 확장하거나 별도의 방안을 마련해야 한다.

3.4.5.1 제한값 목록

중요한 제한값만 정리하며 자세한 내용은 사이트에서 확인할 수 있다.

서비스	내용	제한값	증가여부
API Gateway	Region당 초당 요청 수	10,000	Y
API Gateway	Region당 API 수	600	N
API Gateway	리소스 정책 최대 길이	8092	Y
API Gateway	Region당 APIキ 최대 수	500	Y
API Gateway	API당 Lambda권한 부여자* 최대 수	10	Y
API Gateway	통합 시간 제한	50ms ~ 29s	N
Lambda	메모리 크기	128MB ~ 3008MB	N
Lambda	임시 스토리지 크기	512MB	N
Lambda	요청 당 최대 실행 시간	15 mins	N
Lambda	Region당 동시 실행 수	1,000	Y

서비스	내용	제한값	증가여부
DynamoDB	Table 크기	제한없음	-
DynamoDB	계정 당 Table 수	256	Y
DynamoDB	Table당 Read Capacity	N.Virginia Region 40,000, 그외 10,000	Y
DynamoDB	Region당 Read Capacity	N.Virginia Region 80,000, 그외 20,000	Y
DynamoDB	Table당 Write Capacity	N.Virginia Region 40,000, 그외 10,000	Y
DynamoDB	Region당Write Capacity	N.Virginia Region 80,000, 그외 20,000	Y

*) Lambda 권한 부여자는 API Gateway가 호출될 때 Lambda가 Oauth나 SAML 같은 보유자 토큰 인증을 할 때 사용

[1] AWS Limits: https://docs.aws.amazon.com/ko_kr/general/latest/gr/aws_service_limits.html

3.4.5.2 제한 값 상향신청

제한 값을 요청하면 짧게는 1시간 미만, 길게는 48시간의 시간이 소요되기도 하고 제한 값 요청을 거부당할 수도 있으며 제한 값이 올라가는 중에 PaaS 서비스가 중단되는 경우도 있으니 서비스 오픈 일주일 전에는 미리 제한 값을 상향 신청해야 한다.

제한 값 상향 신청은 AWS 에 로그인 한 후 AWS Support Center에서 Case를 생성한 후 Service Limit Increse로 해당 서비스를 선택하여 요청하면 된다.

3.4.5.3 동시실행 제한

Lambda는 기본적으로 한 계정에서 Region 당 최대 1,000 개의 Lambda를 동시에 실행할 수 있고 그 이상 실행하면 **자동으로 1분마다 동시 실행 수를 500 ~ 3,000 으로 추가 확장**하게 되며 증가되는 수는 Region 마다 다르며 서울 region은 500이다.
동시 실행 수를 초과하면 호출 서비스가 자동으로 재시도를 처리하지만 자동 확장하기 까지 시간이 소요되어 서비스가 지연되니 이를 고려하여 동시에 1,000 건 이상의 함수가 자주 호출된다면 미리 기본 동시 실행수를 확장해야 한다.
또한 함수 별로 동시실행 수 제한 값 지정이 가능하여 무한정 늘어나 많은 비용이 발생하는 것을 막을 수 있다.
아래는 Lambda가 동시 실행 수를 초과하면 호출 방식에 따라 재시도하는 방식에 대한 설명이다.

호출방식	설명
동기	성공할 때까지 재시도 Lambda가 429 오류를 리턴하기에 호출단에서 처리도 가능

호출방식	설명
비동기	최대 6시간동안 자동으로 재시도
폴기반	메시지 보관기간까지 재시도 처리
스트림폴기반	데이터 만료될까지 성공할 때 까지 재시도

3.4.6 콜드스타트

3.4.6.1 콜드스타트 지연시간이 짧은 개발 언어 선택

Java나 C#보다는 Node.js나 Python 같은 언어가 콜드스타트로 인한 지연시간이 적다. 기존에 사용하던 Java를 서버리스컴퓨팅에 적용했다가 콜드스타트 이슈로 인해 Node.js나 Python으로 마이그레이션한 사례가 있으니 제약사항이 없는 한 Node.js나 Python 같은 언어를 사용해야 한다.

3.4.6.2 Pre-warming

콜드스타트로 인한 지연시간을 방지하려면 일정 간격으로 Lambda를 호출해서 웜스타트 상태를 유지해야 한다.
아래 그림은 AWS Lambda에서 웜스타트 하는 방식 중 하나이다. 타이머에 의해 5분에 1회 웜스타트 함수를 호출하여 나머지 함수들을 호출하는 방식으로 함수들은 전체 로직이 수행하지 않고 heartbit check 하는 ping 처리 후 종료되어야 비용을 절감할 수 있다.

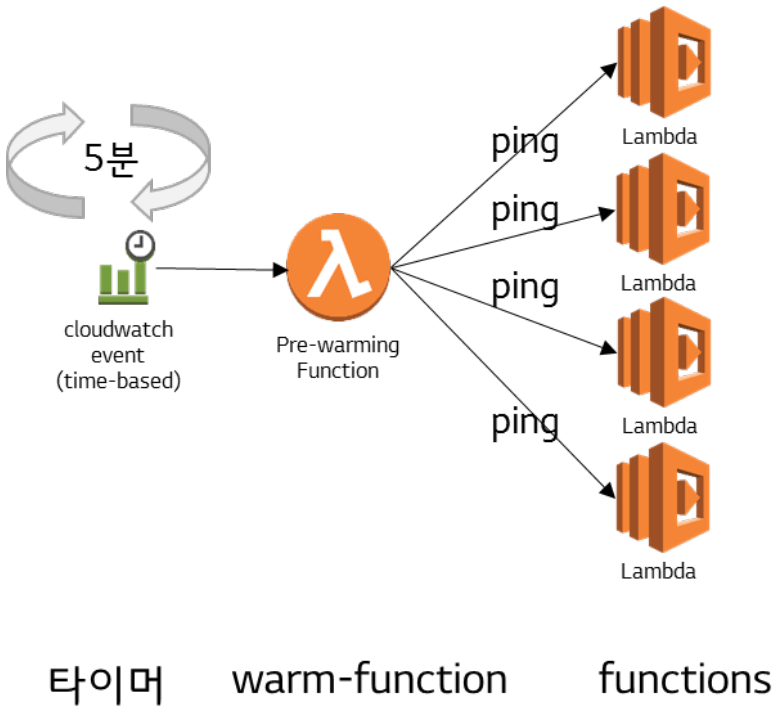

그림. AWS Lambda pre-warming

위와 같이 웜스타트 함수를 구현할 경우 AWS 기준으로 월 예상 비용은 아래와 같다.

- AWS Seoul Region기준
- 각 함수들의 메모리는 256MB
- Warm Function 함수 1회 실행 시간 1초, 나머지 4개 함수들 ping 실행시간 1초로 ping을 위해 총 5초 발생
- 5분에 1회 Warm Function 함수 호출
- 실행횟수 : 시간 당 실행 수 * 하루 24시간 * 월 31일 = (60 / 5) * 24 * 31 = 8,928 회
- 실행시간 : 8,928 * 5초 = 44,640 초
- CloudWatch Event스케줄러 비용 : 0.008928 USD
- AWS Lambda 컴퓨팅 비용 : 0.186037 USD
- AWS Lambda 요청 비용: 0.001786 USD
- 총 : 0.196751 USD

1 USD가 1,130원일 때 월간 230원으로 서버리스 컴퓨팅의 콜드스타트를 방지할 수 있다. 함수들이 많거나 높은 메모리를 사용하거나 ping 에 소요되는 시간이 긴 경우 조금 더 많은 비용이 발생할 수 있다. 하지만 이 방법은 **하나의 함수에 동시 실행 수가 동일하게 유지될 경우에만 적용할 수 있다.** 평소 100개의 요청이 동시에 실행되는 함수에 웜스타트를 적용한 상황에서 150개의 요청이 실행된다면 100개는 웜스타트로 수행되고 나머지 50개는 콜드스타트로 실행하기에 콜드스타트를 100% 피할 수는 없다.

3.4.7 Timezone

서버리스 컴퓨팅은 Timezone이 UTC로 되어 있어 국내시간과 +9 시간의 시차가 발생한다. 국내 시간을 사용하려면 날짜 형식 라이브러리인 moment.js 와 함수의 환경변수를 이용해서 설정해야 한다.
아래 예시는 AWS Lambda를 이용했고 현재 시간을 출력하면 UTC기반인 Lambda의 실행시간보다 +9 시간으로 출력된 것을 확인할 수 있다.

// 1. moment.js 추가,

https://momentjs.com/downloads/moment.js

// 2. 환경변수에 Key: TZ / Values: Asia/Seoul 추가

// 3. 실행 파일 추가, index.js

```
const moment = require('moment');
    exports.handler = (event, context, callback) => {
    console.log(moment().format());
};
```

// 결과값
// 수행시간 lambda실행id moment().format()의 결과값
2018-10-01T04:52:24.169Z c96a377d-c535-11e8-b0be-2b98da915f97 Asia/Seoul: 2018-10-01T13:52:24+09:00

Pre-warming(see page 37)
Timezone(see page 38)