[AWS] 🏴‍☠️ 람다(Lambda) 개념 & 사용법

2023년 6월 14일 수요일 오후 2:28

AWS Lambda

AWS 람다(Lambda)는 서버리스 컴퓨팅 FaaS 상품이다.

서버리스란 개발자가 서버를 관리할 필요 없이 애플리케이션을 빌드하고 실행할 수 있도록 하는 클라우드 네이티브 개발 모델이다.

즉, 클라우드 제공업체가 서버 인프라에 대한 프로비저닝, 유지 관리 등을 대신 처리해주기 때문에, 개발자는 조금 더 비즈니스 로직 작성에만 집중할 수 있게 된다.

[WEB] 🌐 서버리스(ServerLess) 개념 💯 정리 (BaaS / FaaS)

inpa.tistory.com

서버리스 아키텍쳐(Serverless) 란? 서버리스(Serverless)는 직역하면 "서버가 없다"라는 뜻이 된다. 하지만 정말로 서버가 없는 것을 뜻하는게 아니다. 서버스를 하는데 있어 어찌되었든 저장소는 필

다시 본론으로 돌아와, AWS 람다는 서버를 프로비저닝하거나 관리할 필요 없이 코드를 실행하게 해주는 서버리스 컴퓨팅 서비스로서 모든 유형의 애플리케이션이나 백엔드 서비스에 대한 코드를 별도의 관리 없이 실행 가능하다.

사용자는 람다에다가 원하는 함수를 작성하고, 필요할 때 그 함수를 사용할 수 있다.

또한 다른 AWS 서비스들과 연동이 용이하다는 특징이 있다.

예를들어 이미지를 S3에서 읽어올 때, 람다 함수를 통해 필요한 크기로 Resize 하는 기능에도 사용 할 수 있다.

이처럼 다른 AWS 서비스에서 코드를 자동으로 트리거 하도록 설정하거나 웹 또는 모바일 앱에서 직접 코드를 호출할 수도 있다.

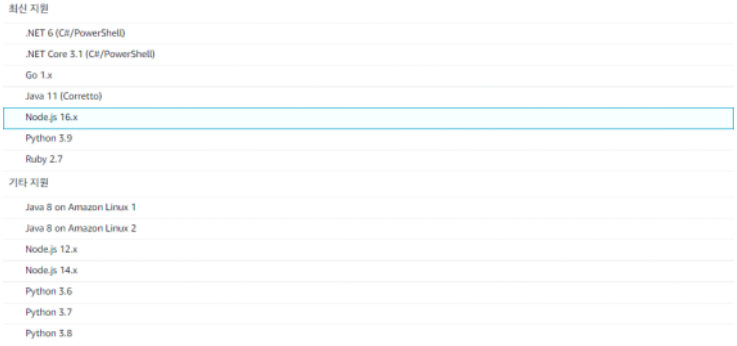
코드를 업로드만 하면 Lambda에서 높은 가용성으로 코드를 실행 및 확장하는 데 필요한 부분을 알아서 처리해주기도 한다.

Lambda의 스펙

람다 지원 언어

Lambda는 다음과 같은 파이썬, Node.js, 루비, 자바, C#, 파워셸, 구글 고로 작성된 런타임(Runtime)을 지원한다.

자신이 익숙한 프로그래밍 언어로 자유롭게 코딩이 가능하다.

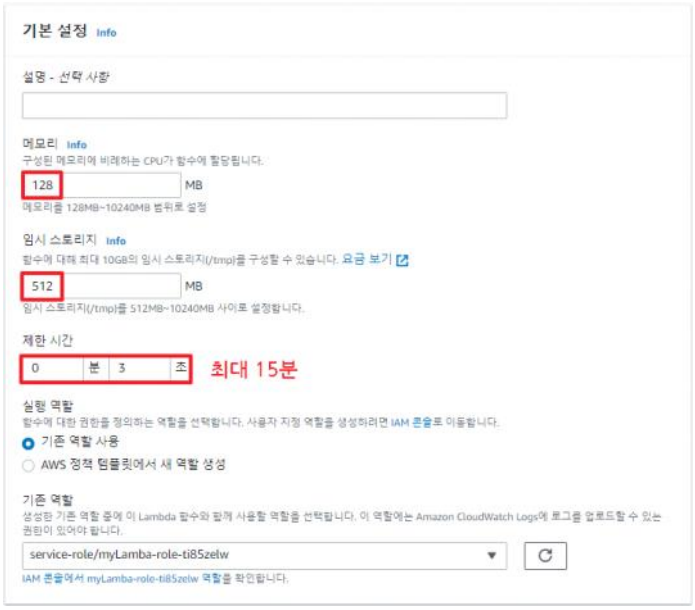


Lambda 실행 환경

또한 람다는 실행 환경 스펙 제한이 있다.

하드웨어 적으로도 비용 측면이나 공간 측면이나 업그레이드 할수 있는 메모리나 cpu 에 제한이 있듯이 람다에도 똑같이 적용된다라고 보면 된다.

메모리	CPU	MEMORY
128	Intel(R) Xeon(R) CPU E5-2666 v3 @ 2.90GHz (2)	3855600 kB
256	Intel(R) Xeon(R) CPU E5-2666 v3 @ 2.90GHz (2)	3855600 kB
512	Intel(R) Xeon(R) CPU E5-2686 v4 @ 2.30GHz (2)	4047688 kB
1024	Intel(R) Xeon(R) CPU E5-2686 v4 @ 2.30GHz (2)	4047688 kB
3008	Intel(R) Xeon(R) CPU E5-2686 v4 @ 2.30GHz (2)	4047688 kB



- CPU : EC2 처럼 정해져있는게 아니고, 메모리에 따라 CPU 사양이 자동으로 설정되는 형태 (위 그림 참조)
- 메모리 : 128MB ~ 10000MB (2020년 기준 10GB로 향상)
- 임시 저장용 디스크 공간 : 512MB
- 최대 실행 시간 : 900초 (람다 함수를 최대 15분 동안만 실행할 수 있다는 의미)
- 압축시 패키지 크기 : 50MB
- 압축을 풀었을 경우 패키지 크기 : 250MB

Info
여기서 패키지 크기는 함수를 실행하는 데 필요한 모든 코드를 일컫는다.
여기에는 함수가 가져올 수 있는 모든 의존성들(Node.js의 경우 node_modules/ 디렉토리)가 포함된다.
압축하지 않은 패키지의 경우 최대 250MB가 제한이며 압축된 경우 50MB 제한이 적용된다.

Lambda 장단점

Lambda 장점

1. 비용절감

필요할때만 함수가 호출되어 처리한다는 점에서 항상 서버를 켜두고 있지 않아도 되므로 비용을 절약 할 수 있다.

람다함수의 요청 수와 람다코드의 실행시간에 따라서 요금이 부과된다.

2. 인프라 운영관리 부담절감

성능이나 보안 등 서버 자체의 관리는 AWS가 알아서 해주기 때문에 서버를 관리할 필요가 없으니 운영관리에 대한 부담이 줄어든다.

예를들어 트래픽이 증가하면 알아서 자동으로 오토스케일링이 된다.

Tip
참고로 람다는 컨테이너 개념으로 스케일링 된다.
EC2의 오토 스케일링 같이 가상OS가 새로 생성되는게 아니라, OS 위에 독립적인 하나의 프로그램을 키는 방식이기 때문에 훨씬 가볍게 확장이 가능하다.
이벤트가 발생할 때 생성되고 더 이상 사용하지 않을 때는 제거된다.

3. 빠른 개발 배포

람다를 이용하게 되면 상당히 개발 및 배포에 대한 소요시간이 매우 짧아지게 된다.

AWS 자체에서 많은 기능을 제공해주고 있어서 API연동이 쉬우며, Serverless의 강점인 배포방법도 정말 쉬운 편이다.

그저 함수를 수정해주기만 하면 되기 때문이다.

4. 언제 Lambda를 쓰면 좋을까?

코드를 계속 실행시키기 보다는 특정한 시기에만 실행시키는 경우에 Lambda를 사용하면 유용하다.

- 서버 띄우지 않고 간단한 코드를 실행시키고 싶은 경우

- 특정 기간 또는 특정 주기로 코드를 실행시켜야 하는 경우
- 트리거가 실행될때만 코드를 실행시키고 싶은 경우

Lambda 단점

1. 리소스 제한

람다는 메모리(최대 10GB), 처리시간 (최대900초, 즉 15분)으로 제한되어 있다.

즉, 하나의 함수가 한번 호출될때 AWS에서는 최대 10GB의 메모리까지 사용이 가능하며, 처리시간은 최대 15분 이라는 말이다.

아무래도 직접 서버에서 돌리는 것보다 부족한 스펙이다.

2. Stateless (상태비저장)

람다는 함수가 호출되면 새로운 컨테이너를 띄우는 방식이기 때문에 별도의 **상태를 저장하지 않는다.**

이는 Lambda 함수가 이벤트에 의해 트리거 될 때마다 완전히 새로운 환경에서 호출된다는 것을 의미한다.

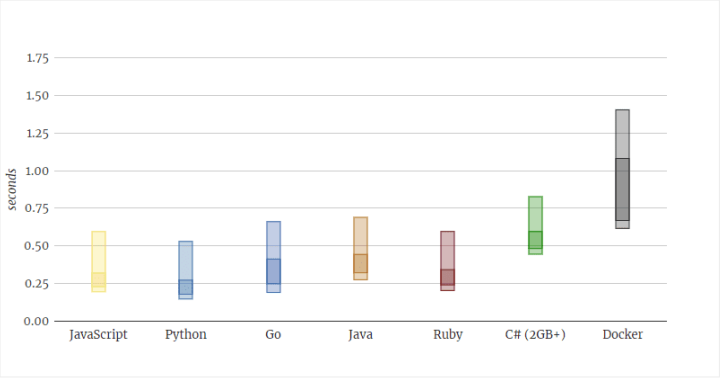
그래서 이전 이벤트의 실행 컨텍스트에 대한 액세스 권한이 없다보니, db connection을 유지하는 것 같은 기능은 수행하지 못한다.

3. ColdStart

람다는 리소스를 효율적으로 사용하기 위해서 오랫동안 사용하지 않고 있을 경우 잠시 컴퓨팅파워를 꺼두고 있는다.

그래서 다시 사용하려고 하면 람다 컨테이너를 띄우기 위해 서버가 켜지고 실행환경을 구성하기 위해 약간의 시간이 걸린다.

즉, 요청이 와서 람다함수를 실행하게 되면 어느정도 딜레이가 발생하게 된다.



Typical cold start durations per language

하지만 EC2 같은 경우 항상 가동된 상태에서 요청을 받을 준비가 되었기 때문에 딜레이가 없다.

이를 반댓말인 WarmStart라고도 불리운다.

즉, 서버리스의 최대 단점인 이 콜드 스타트 문제를 해결하는 것이 가장 큰 관건이라고 할 수 있다.

Info

[콜드스타트 해결방법]

1. 람다를 계속 호출해준다.
사실 항상 컨테이너가 준비되어 있게 하도록 람다를 지속적으로 호출해는게 가장 좋다.
하지만 호출이 될때마다 비용이 산정되는 방식이기 때문에 그만큼 비용이 더 들수도 있다.

2. 람다의 메모리를 늘려 스펙을 높인다.
메모리를 더 할당할 수록 컴퓨팅 스펙이 높아지기 때문에 처리속도가 빨라져서 딜레이가 줄어들 수도 있다.

3. 프로비저닝된 동시성 활성화
2019년 콜드스타트 문제를 해결하기 위해 나온 옵션으로 함수의 호출에 바로 응답할 수 있게 미리 준비하는 옵션이다.
이를 활성화 하면 미리 함수의 환경을 세팅해두기 딜레이가 줄어들지만, 추가적인 비용이 든다.

4. 동시성 제한

동시성이란 특정 시각에 함수가 제공하는 요청의 수이다.

함수가 호출되면 Lambda는 함수의 인스턴스를 할당하여 이벤트를 처리하고, 실행이 마치면, 다른요청을 처리할 수 있다.

람다는 각 리전별로 동시에 실행할 수 있는 람다함수의 개수를 최대 1000개로 제한하고 있다.

그래서 request 수 이를 넘어가게 되면 람다가 수행되지 않는 문제점이 발생할 수도 있다.

Info

[람다의 동시성]

동시성이란 특정시간에 람다함수가 처리할 수 있는 request의 수이다.

람다는 기본적으로 1개의 워커를 가지고 있는 컨테이너의 개념이다 그래서 reqeuest에 의해 람다함수가 호출이 되면, 하나의 컨테이너를 띄우고 작업을 처리한다. (vmware나 virtual box 처럼 일종의 가상머신으로 프로그램을 돌리는 것이라고 이해해도 된다)

만약 요청을 처리하는 와중에 다시 람다함수가 호출이 되면 또다른 컨테이너를 띄우기 때문에 동시성이 증가한다.

Info

[동시성을 해결하는 방법]

1. 예약된 동시성을 통해 동시성 개수 늘리기

함수에 대한 최대 동시 인스턴스를 보장하는 옵션이다.

예약된 동시성을 통해 한 함수가 동시성을 예약하면 다른 함수가 해당 동시성을 사용할 수 없게 된다.

예를들어 쇼핑몰에서 900명의 이용자가 상품을 구매한다고 했을때, 이 예약 처리를 람다로 하는데, 만일 다른 200명의 사용자가 게시글을 올리는 행위(람다로 처리한가고 가정)를 했을때 1000개 동시성 제약때문에 100명의 이용자의 구매 작업이 지연되어 서비스 이용에 치명적이게 될수 있다.

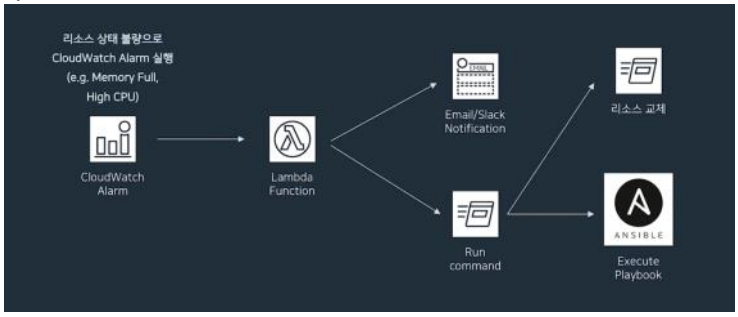
따라서 특정 람다함수에 대한 최대 동시 인스턴스 수를 보장해준다는 의미이다. (프로비저닝과는 다르게 예약된 동시성을 구성하는 데는 요금이 부과되지 않음)

2. Limit Increase 요청

하지만 서비스 이용객 특성상 아무리 최적화를 해도 어쩔 수 없이 최대동시성 수를 넘어갈 경우, AWS에 동시실행 수 늘려달라고 서비스 할당량 증가 요청을 할 수 있다.

Lambda 실무 사용 사례

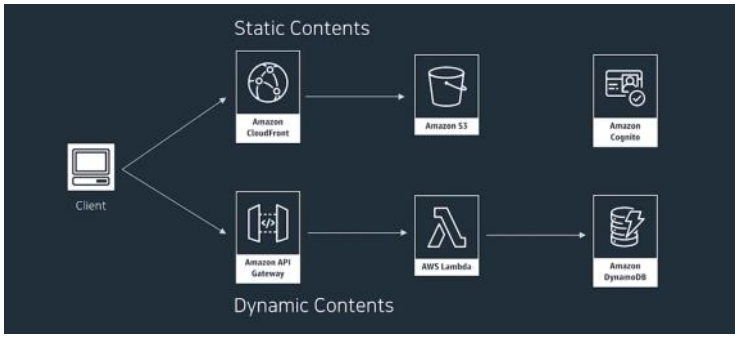
Operation Automation (시스템 운영 자동화)



1. CloudWatch Alarm을 모든 리소스에 걸어둔다.
2. Memory Full 이나 CPU가 높아져서 대응이 필요할 때, 즉 리소스 상태불량 일 때 CloudWatch Alarm이 실행되면 람다 함수 실행 된다
3. 람다 함수가 Email이나 Slack Notification으로 관련자들에게 알려준다
4. 람다 함수가 Ansible과 결합해서 Memory Full났을 때 로컬 인스턴스에서 메모리를 리셋해준다던지 리소스를 교체해준다던지 장애 발생 시 automatic하게 복구할수 있다.

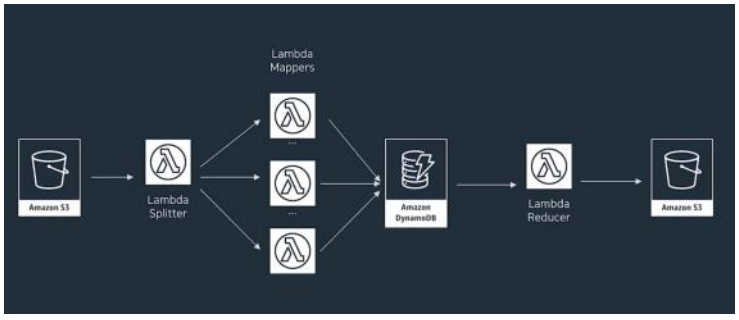
Web application

AWS Lambda를 다른 AWS 서비스와 결합하면, 확장성, 백업 또는 여러 데이터 센터 중복에 필요한 별도의 관리 작업 없이 개발자가 자동으로 확장 및 축소되고 여러 데이터 센터에 걸쳐 가용성이 높은 구성에서 실행되는 강력한 웹 애플리케이션을 구축할 수 있다.



5. 클라이언트가 접속을 했을 때 정적 컨텐츠(이미지 등)는 S3에 저장
6. S3 앞단에 CloudFront 달아서 전세계 어디서든 빠르게 서빙할 수 있음
7. 인증으로는 Cognito를 별도로 사용할 수 있음
8. Danamic Contents, 프로그램 작업이 필요한 경우에는 Lambda와 API Gateway로 서비스를 하고 백단에서는 DB로 DynamoDB를 운영할 수 있음

Serverless batch processing



9. S3에 어떤 Object 같은게 들어왔을 경우 Lambda Splitter가 Mapper에다가 작업을 분배를 하고 Mapper들은 작업이 끝난 후 DynamoDB에 저장
10. Lambda Reducer가 S3로 다시 아웃풋

EC2 와 람다의 분업

기본 서버로 EC2를 아예 사용하지 않을 수 있고, EC2의 보조적 용도로 사용할 수도 있다.

예를 들면, 일반적인 Request는 EC2로 하고, 구매와 같이 중요하면서도 특정상황에 따라 폭증할수도 있는 경우는 Lambda를 사용할 수도 있다.

비용대비 효과가 뛰어난 람다를 매출이 발생하는 부분에서 사용해준다면 매우 적절하다.

실시간 파일 처리

Amazon S3를 사용하여 업로드하는 즉시 데이터를 처리하도록 AWS Lambda를 트리거할 수 있다.

Lambda를 사용하여 실시간으로 이미지를 썸네일하고, 동영상을 트랜스코딩하고, 파일을 인덱싱하고, 로그를 처리하고, 콘텐츠를 검증하고, 데이터를 수집 및 필터링할 수 있다.

실시간 스트림 처리

AWS Lambda 및 Amazon Kinesis를 사용하여 애플리케이션 활동 추적, 트랜잭션 주문 처리, 클릭 스트림 분석, 데이터 정리, 지표 생성, 로그 필터링, 인덱싱, 소셜 미디어 분석, IoT 디바이스 데이터 텔레메트리 및 측정을 위한 실시간 스트리밍 데이터를 처리할 수 있음.

DynamoDB 와 연동 (추출, 변환, 로드)

AWS Lambda를 사용하여 DynamoDB 테이블의 모든 데이터 변경에 대한 데이터 검증, 필터링, 정렬 또는 기타 변환 작업을 수행하고 변환된 데이터를 다른 데이터 스토어로 로드할 수 있다.

IoT 백엔드

AWS Lambda 및 Amazon Kinesis를 사용하여 사물 인터넷(IoT) 디바이스 데이터 텔레메트리 및 분석을 위한 백엔드를 구축할 수 있다.

모바일 백엔드

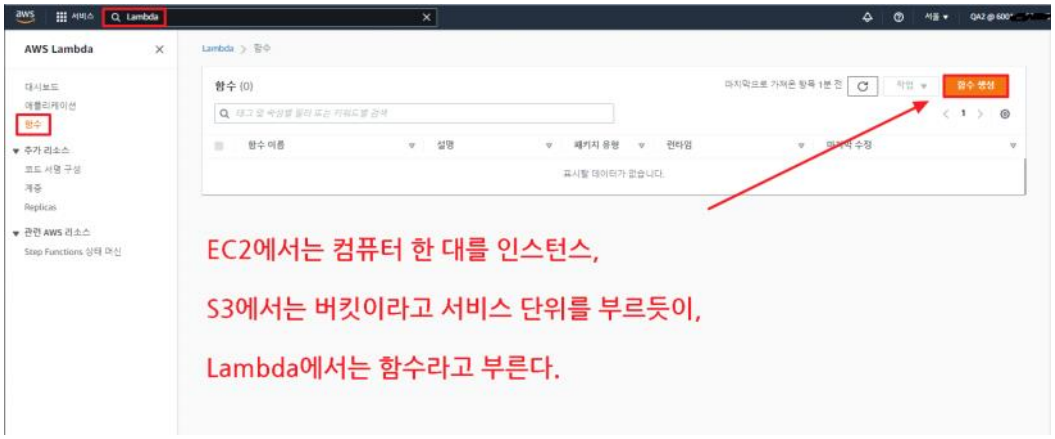
AWS Lambda 및 Amazon API Gateway를 사용하여 API 요청을 인증 및 처리하도록 백엔드를 구축할 수 있다.

람다(Lambda) 실전 사용해보기

람다 함수 생성

EC2에서는 컴퓨터 한 대를 인스턴스, S3에서는 버킷이라고 서비스 단위를 부르듯이, Lambda에서는 함수라고 부른다.

[함수 생성] 버튼을 눌러 람다함수를 만들어보자.



함수 생성을 누르면 상단에 4가지 옵션이 나온다.

- 새로 작성 : 처음부터 함수를 구현한다.
- 블루프린트 : 경우 AWS에서 제공하는 기능을 템플릿처럼 사용할 수 있는 기능이다. 샘플 코드 형식으로 커스터마이징 하여 사용할 수 있다.
- 컨테이너 이미지 : docker 컨테이너용
- 서버리스 앱 리포지토리 찾아보기 : 공유되는 간단한 서비스 아키텍처를 내 환경에다 가져다 사용할 수 있다.

런타임은 람다를 어느 프로그래밍언어로 작성할지에 대한 옵션이다.

노드, 파이썬 등 다양한 런타임 언어를 제공한다. 보통 노드를 많이 쓰는 편이다.

권한은 람다에 어떤 권한을 부여할지에 대한 옵션이다.

예를들어 람다함수 코드에 S3에 접근해서 파일을 가져오고 다시 저장하는 기능의 코드가 있다면, 람다에는 S3에 접근할수있는 권한을 가지고 있어야 한다.

이러한 서비스는 IAM에서 역할(Role)로 부여하는 식으로 사용할 수 있다.

[AWS] IAM 개념 원리 & IAM 계정 · 정책 생성하기
inpa.tistory.com
IAM (Identity and Access Management) 란? IAM은 사용자의 접근 권한을 관리 하는 서비스 이다. IAM을 통해서 회사 내 AWS를 사용하는 사람들에게 부서 마달 사용자별로 AWS에서 제공하는 서비스들, 서비스에

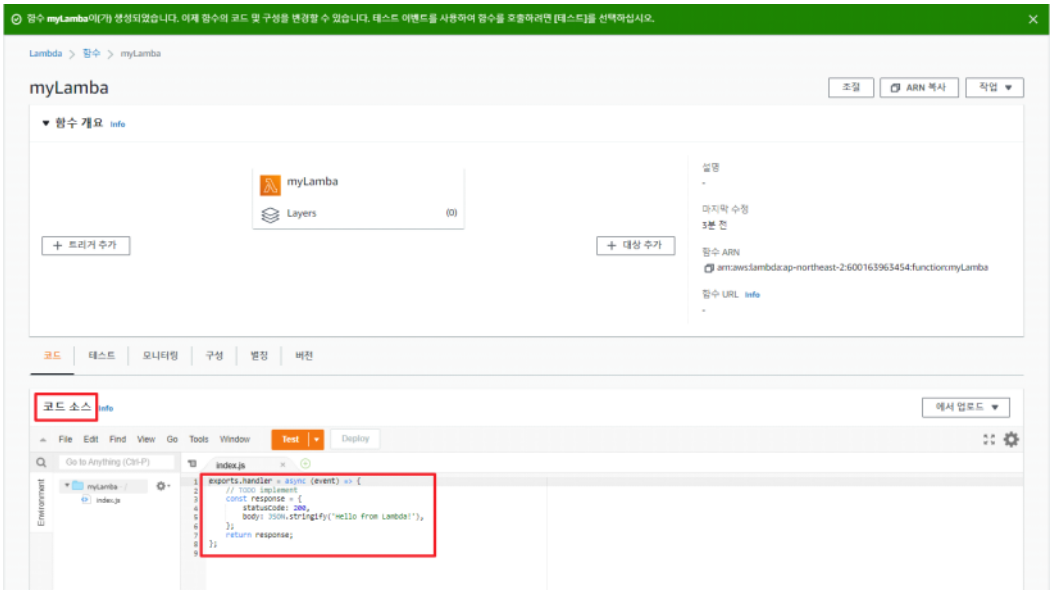
또한 밑에 "Lambda가 이름이 myLambdaFunction-role-ti85zel2이고 Amazon CloudWatch Logs에 로그를 업로드할 수 있는 권한이 포함된 실행 역할을 생성합니다"라는 문구는 람다 함수를 사용하면서 발생하는 로그파일들이 CloudWatch로 업로드될 수 있는 권한이 있다는 뜻이다.

람다 함수 코드 작성

람다 함수 메뉴로 들어오면 스크롤을 밑으로 내려, 함수코드로 가보면 샘플코드가 들어가있는걸 볼수 있다.

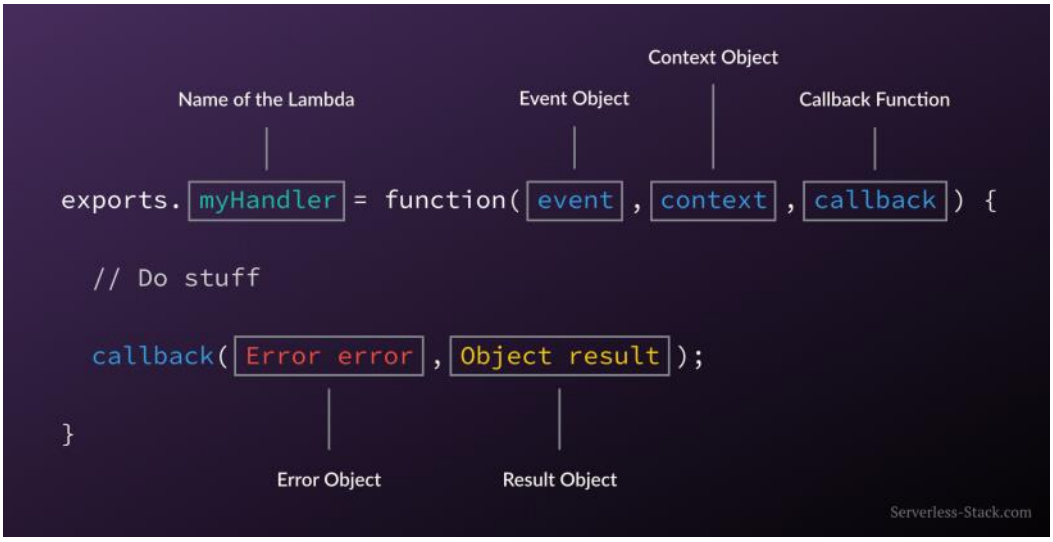
exports.handler라는 약속된 이름의 디폴트 함수가 있는데, 람다를 실행하면 이 exports.handler 메소드가 호출이 되어 코드가 실행되는 형식이다.

```
exports.handler = async (event) => {
    // TODO implement
    const response = {
        statusCode: 200,
        body: JSON.stringify('Hello from Lambda!'),
    };
    return response;
};
```



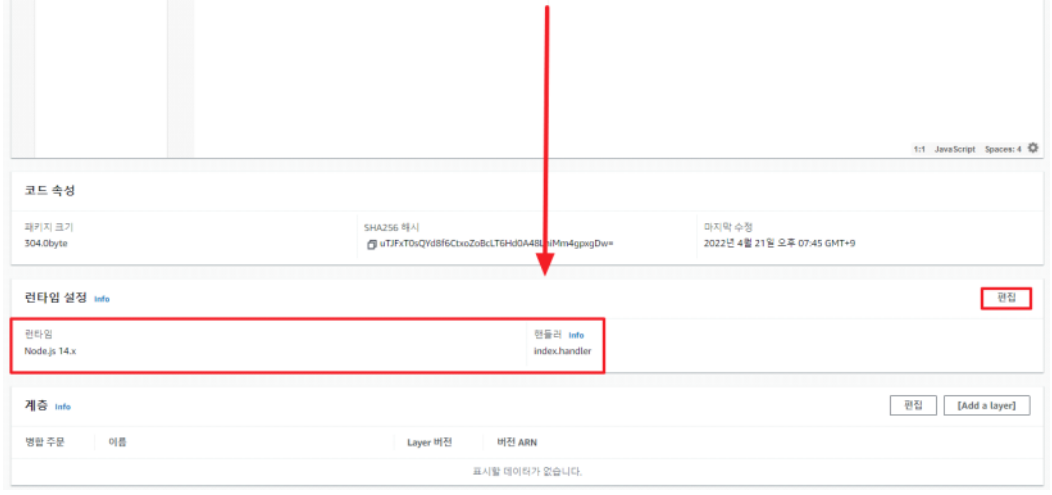
Lambda 함수 구성

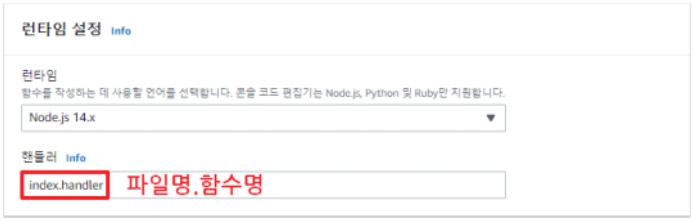
다음은 Lambda 함수(Node.js 버전)의 문법 구성이다.



- 여기서 myHandler는 Lambda 함수의 이름이 된다
- event 객체는 이 Lambda를 트리거 한 이벤트에 대한 모든 정보를 포함한다.
  예를들어 이벤트가 HTTP 요청인 경우, 해당 HTTP 요청에 대한 모든 정보가 들어 있다.
- context 객체는 Lambda 함수가 실행되는 런타임에 대한 정보를 포함한다.
- Lambda 함수 내부에서 모든 작업을 수행한 후에는 그에 대한 결과(또는 오류)와 함께 callback 함수를 호출하고 이를 AWS가 HTTP 요청에 대한 응답으로 처리한다.

만일 현재 디폴트로 적용되어있는 람다함수 진입점 index.js 파일과 handler 라는 함수명을 바꾸고 싶다면, 런타임 설정에서 편집을 하면 된다.

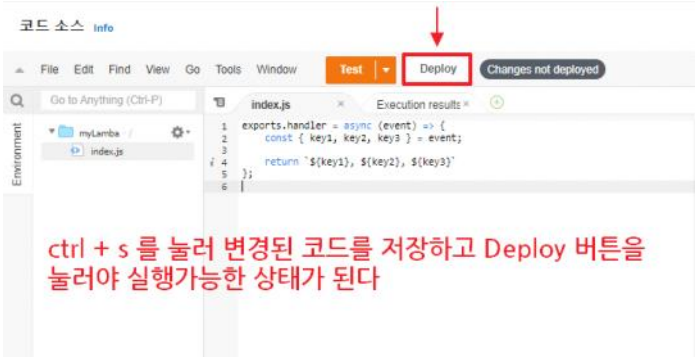


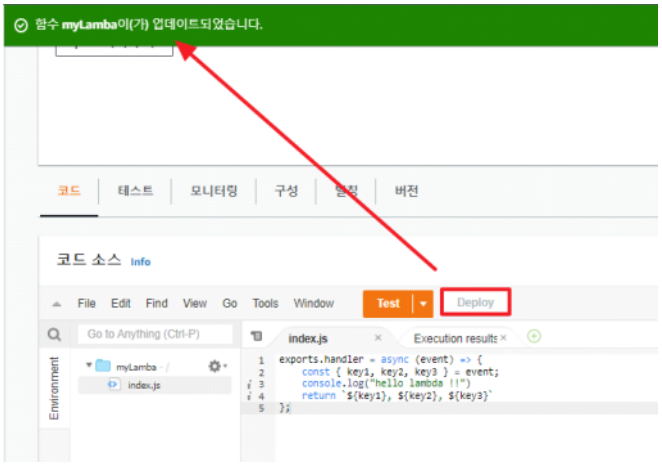


람다 함수 실행하기

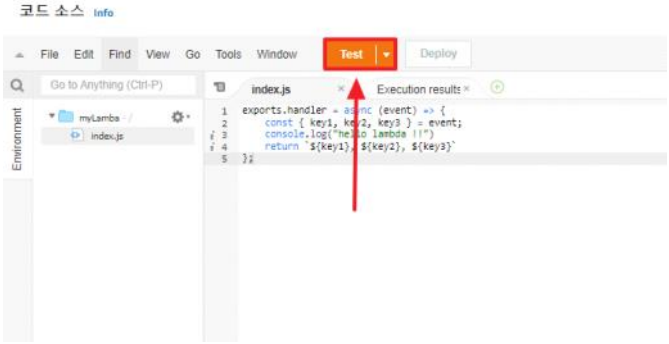
기본 코드를 다음과 같이 커스터마이징 해보자.

```
exports.handler = async (event) => {
    const { key1, key2, key3 } = event; // 이벤트에서 보내는 json 값을 구조분해로 파싱
    console.log("hello lambda !!")
    return `${key1}, ${key2}, ${key3}`
};
```



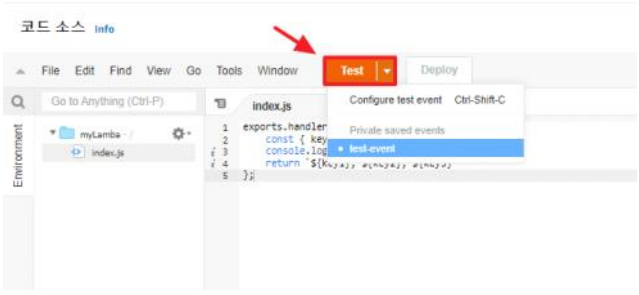


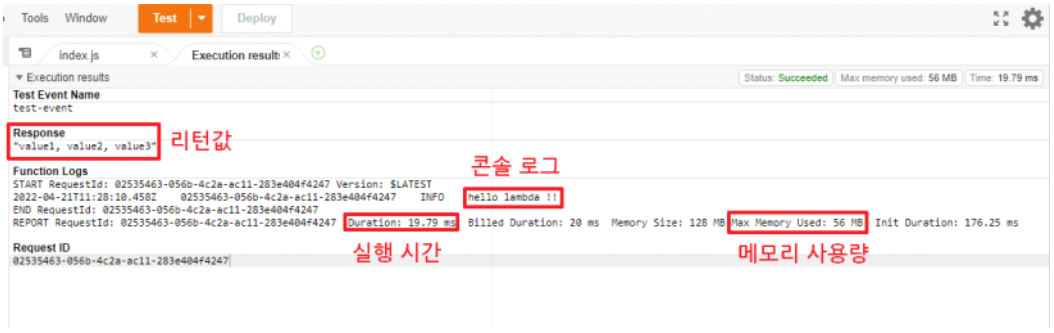
이제 함수를 실행하는데, 우선은 람다함수가 잘 동작하는지에 대한 확인 차원으로 테스트 실행을 해보자



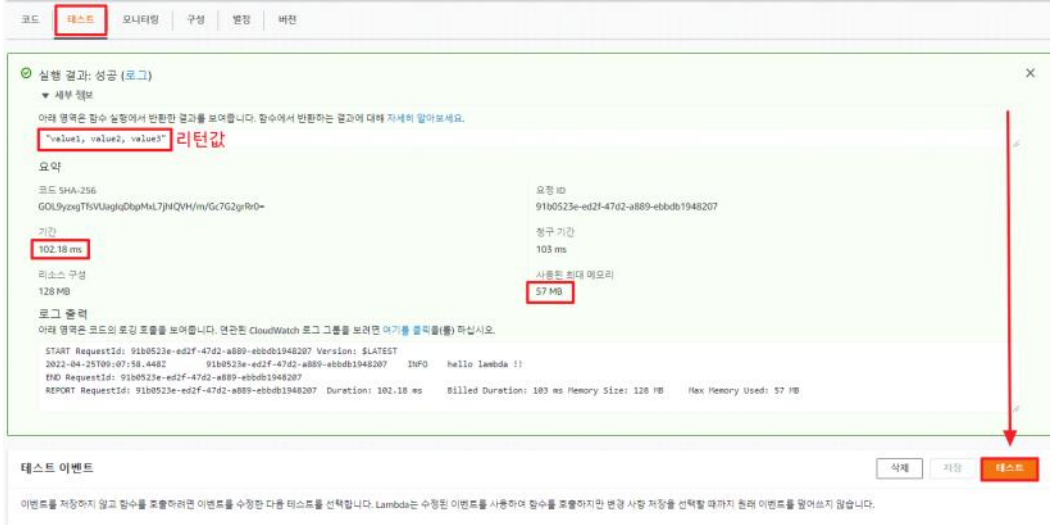
TEST 버튼을 누르면, 테스트 이벤트 구성 창이 나오는데, 람다함수에 보내고 싶은 값을 전달할 값을 세팅 할 수 있다.

템플릿을 고르고 Test 버튼을 누르면, 설정한 이벤트 JSON 값이 export.handler 함수의 event 파라미터로 들어가게 된다.





로그가 보기 어려우면, 에디터 상단 메뉴의 테스트 탭에서 가독성 좋게 볼수도 있다.



그러면 테스트 실행이 아닌 실무에서는 람다를 어떻게 실행할까?

대표적으로 람다 트리거 기능을 이용해서 자동으로 람다 함수를 실행토록 시나리오를 지정해줄 수 있다.

위에서 언급했듯이, 예를들어 S3에 이미지 파일이 올라오면 이를 트리거로 설정하여 람다함수를 실행하게 하여 이미지 압축을 한다던지 행동을 지정해줄 수 있다.

이외에도 API Gateway 서비스와 연동해서, 브라우저 에서 url(rest api)을 요청하면 바로 람다함수가 실행하도록 처리도 가능하다.

람다 트리거 설정법에 대한 방법은 다음 포스팅을 참고하길 바란다.

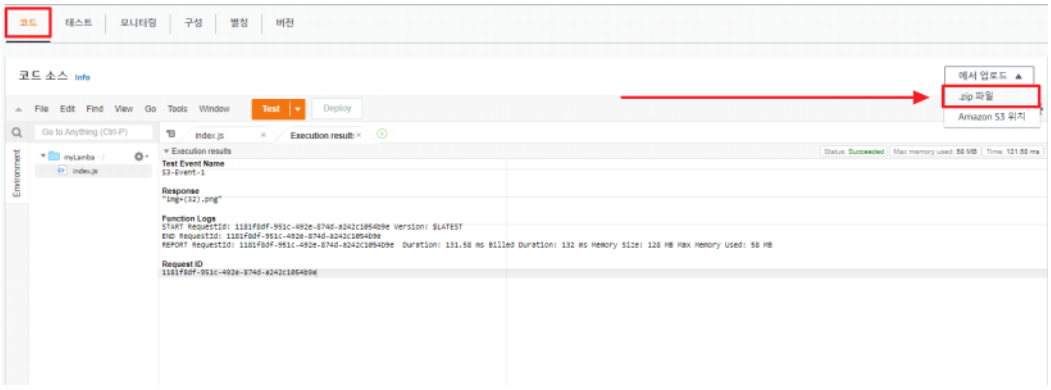
[AWS] 람다(Lambda) 실전 구축 연습하기 (트리거)

inpa.tistory.com

람다(Lambda) 사용 세팅 이번 강의에선 AWS 람다를 실제로 만들어보아 람다함수를 실행하는 방법과 디버깅 방법을 알아볼 것이다. 서버리스와 AWS Lambda에 대한 개념 원리는 다음 포스팅을 참고하길

외부 코드를 람다에 업로드 하기

만일 누가 만들어놓은 템플릿을 내 람다함수 코드에 업로드 하고 싶다면, 코드를 일일히 복붙하지말고 외부 파일을 zip으로 만든뒤에 코드 업로드를 하면 된다.



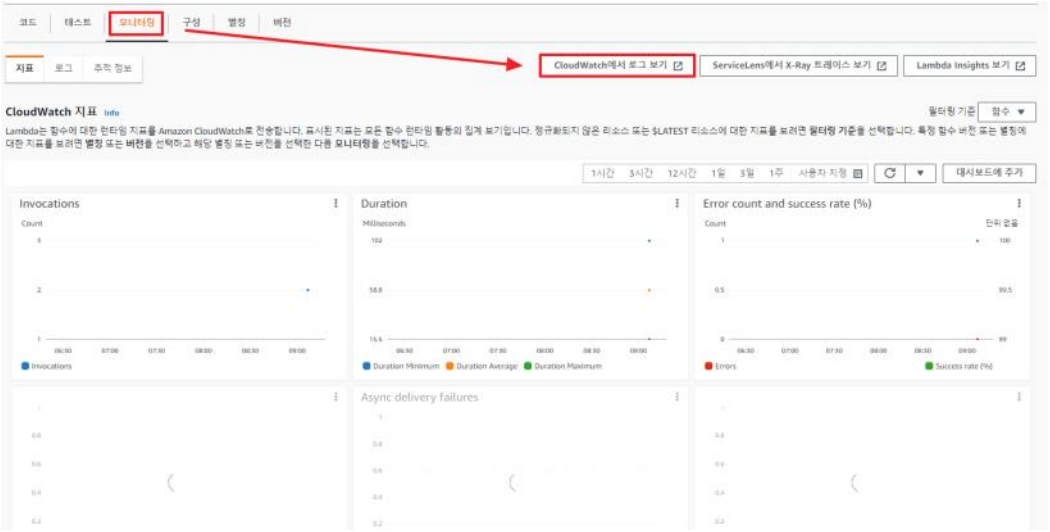
람다 디버깅 하기

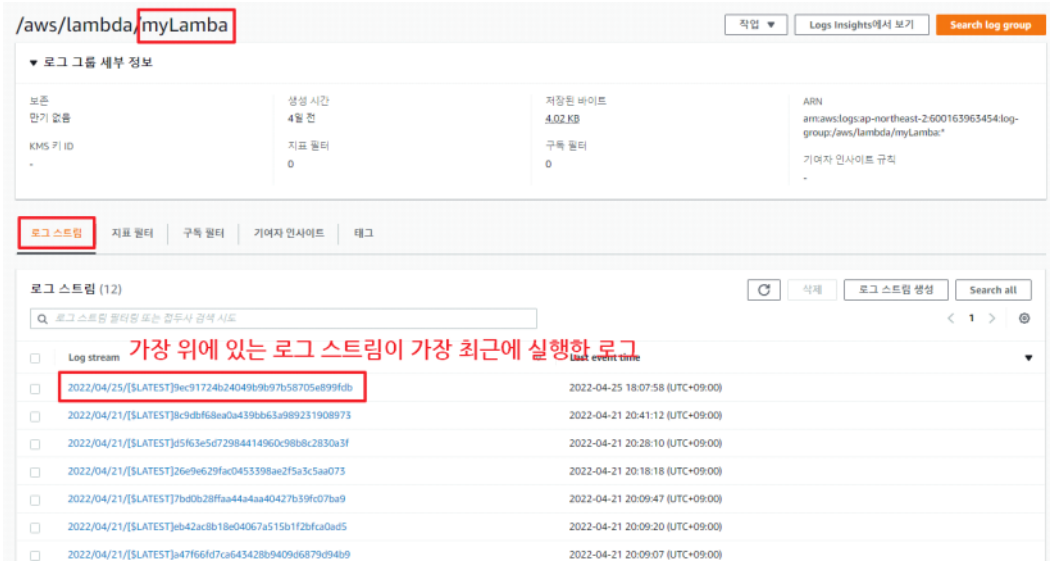
위에서 출력된 람다 로그는 클라우드 콘솔용 테스트 이벤트 로그이다.

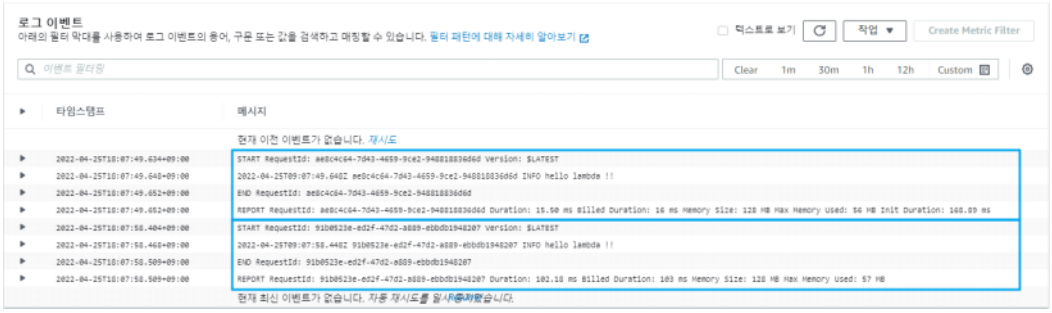
그래서 만일 다른 서비스와 연동한 람다함수 실제 실행 결과 로그를 보고 싶다면, AWS 서비스인 CloudWatch에서 lambda 함수에 대한 로그를 확인해야 된다.

상단 메뉴바에서 모니터링 탭에 들어가 CloudWatch 콘솔 메뉴로 이동한다.

람다 로그는 CloudWatch에서 로그 스트림 이라는 곳에 쌓여있다.







Tip

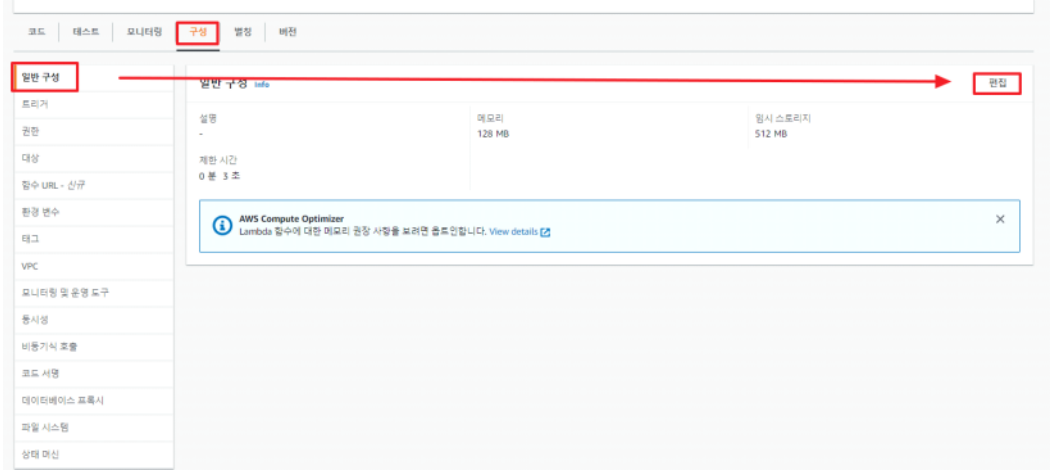
여기서 유의할 점은,

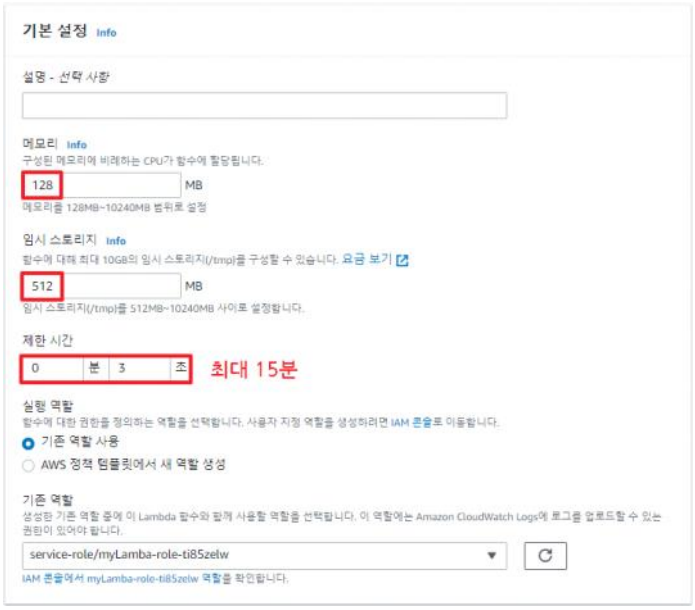
람다 코드를 수정을 하고 저장을 하면 새로운 로그스트림이 생기지만, 코드 수정을 안하고 같은 람다를 여러번 실행시키면 하나의 로그스트림 안에 로그가 위 사진 처럼 여러개 쌓인다.

람다 성능 세팅하기

EC2 컴퓨팅에도 스펙이 있던 것 처럼, 람다함수도 스펙이 존재한다.

람다 구성 메뉴에 일반 구성 편집에 들어가보면 128MB ~ 10240MB 범위로 설정 메모리의 컴퓨터를 빌릴 수 있다.





제한시간은 만일 람다함수 코드 실행이 3초를 넘으면 에러를 내뿜게 하는 기능이다.

람다 스펙상 최대 15분 까지 설정할 수 있다.

그리고 위 메뉴를 보면 따로 CPU 스펙 설정은 없는데, 람다는 메모리 설정에 비례해서 CPU가 자동으로 할당 되기 때문이다.

메모리	CPU	MEMORY
128	Intel(R) Xeon(R) CPU E5-2666 v3 @ 2.90GHz (2)	3855600 kB
256	Intel(R) Xeon(R) CPU E5-2666 v3 @ 2.90GHz (2)	3855600 kB
512	Intel(R) Xeon(R) CPU E5-2686 v4 @ 2.30GHz (2)	4047688 kB
1024	Intel(R) Xeon(R) CPU E5-2686 v4 @ 2.30GHz (2)	4047688 kB
3008	Intel(R) Xeon(R) CPU E5-2686 v4 @ 2.30GHz (2)	4047688 kB

메모리에 따른 CPU 성능 스펙