

# Serverless right now!

서버리스 아키텍쳐가 낯설다면 망설이지 말GO!

SEONGSPA

# 소개 말씀

안녕하세요. `seongspa` 입니다.

42 Seoul `m\_htkim` 멘토님의 Cloud: design and process 강의를 듣고 소프트웨어 아키텍쳐에 관심을 가지게 되었습니다.

AWS x 42 Adelaide에서 주관한 Cloud Quest에서 Seoul 리전 1등을 했습니다.

디지털유목민이 되고 싶은데요.

어찌보면 한 곳에 거처를 두지 않고 돌아다닌다는 점에서 서비스 아키텍쳐와 저는 떼려야 뗄 수 없을 것 같네요.

# 나누고 싶은 이야기

Part 1. Serverless X0, 서비스에 대해 알아가는 시간!

Part 2. Hands-on labs, 서비스 컴퓨팅의 핵심 요소인 AWS Lambda를 사용해봐요!

Part 3. Ask me anything, 서비스에 대해서 무엇이든 물어보세요!

# 누가 들으면 좋을까요?

서비스 초보자, 클라우드 입문자, 그리고 클라우드에 관심있는 사람이라면 누구나 대상으로 합니다.

서비스 기획단계에서 서비스 아키텍쳐를 검토하고 계신 분들께 도움이 될 수도(?) 있습니다.

숙련자분께는 내용이 뻔할 수 있어 뒤쪽에 다과를 섭취하면서 저를 귀엽게 봐주셨으면 합니다. (디펜스 환영!)

감사합니다.

# Part 1. Serverless XO

# 들어가기에 앞서 클라우드 서비스는 어떻게 사용하고 계신가요?

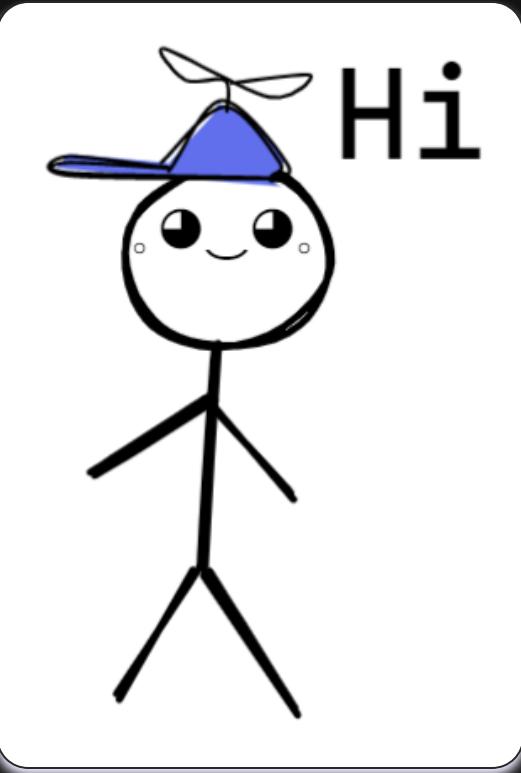
서버를 온프레미스로 구현하셨다고요?

클라우드 서비스 이용하신다고요?

서버가 없다고요?

아니 서버가 없을 수가 있나요?

이와 관련해서 궁금증이 있는 어떤 사람을 소개시켜드릴게요.



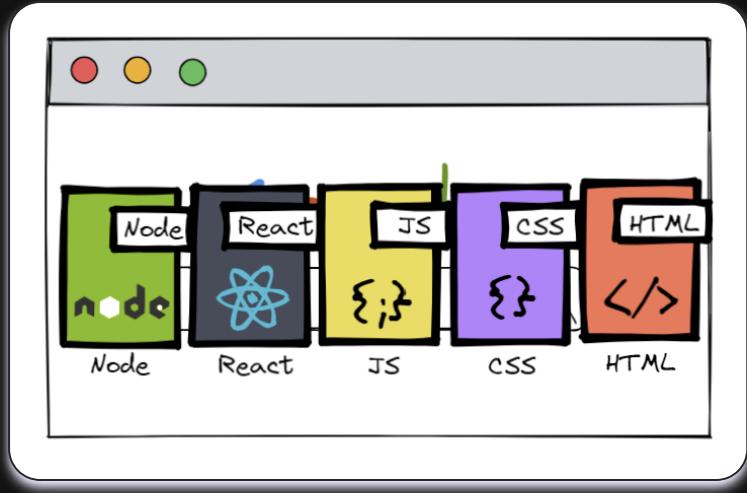
하이, 나 아기 카뎃.

번뜩이는 사이드 프로젝트 하나 가져왔어.

바로, 42 술랭이야.

클러스터 주변 맛집 지도 제작해보려고!

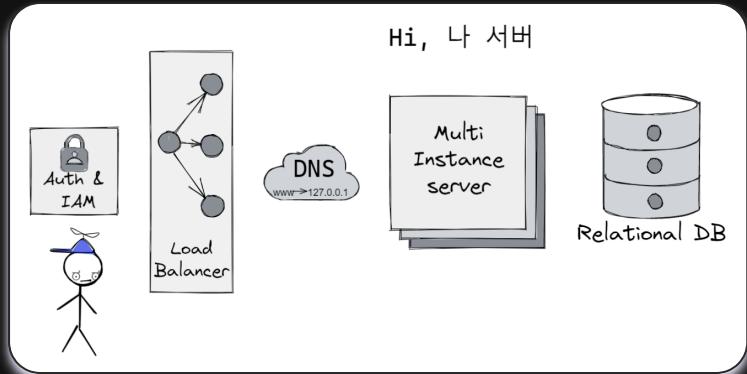
잘 부탁해.



오케이, 비지니스 로직 간단하고 🔥

프론트는 Node.js, React, JS, CSS, HTML이면

사이트 똑딱이겠네 ㅎㅎ



갑자기 백엔드 생각하니 골치가 아파졌다.

"서버를 제일 잘 관리하는 방법은  
서버가 없는 것이다."

– Werner Vogels, Amazon CTO

우리가 전달할 솔루션에 물리적인 서버 자체가 포함되어 있나요?

직접 서버를 만들어서 그 위에 모든 걸 구축하고 관리하는 것이 42 솔랭이 제공하고 싶은 비즈니스인가요?

그게 아니라면 필요한 서비스는 제공받으면 되는 것입니다.

## \*-a-a-S, etc.

: anything as a service

어떤 것을 서비스로 제공받는다는 의미인데 쉽게 말해 대여한다고 생각해보시죠.

IaaS

서버, 스토리지, 네트워크 등 인프라를 대여

PaaS

개발을 도와주는 플랫폼을 대여

SaaS

소프트웨어를 대여

BaaS <- serverless

백엔드 기능(데이터베이스, 소셜미디어 연동, 파일시스템 등)을 대여

FaaS <- serverless

함수를 대여

# BaaS

BaaS는 단일 웹페이지나 모바일 앱 기반의 서비스에서 필요한 백엔드 기능(로그인, 데이터 관리, 회원 관리 등)을 개발자가 직접 개발하지 않고 클라우드 공급자가 제공하는 서비스를 이용해 쉽고 안정적으로 구현하는 것입니다.

애플리케이션에 당연히 요구되지만 구현하기는 번거로운 데이터 저장소, 파일 저장소, SNS 연동, 위치 정보 검색, 푸쉬 알림과 같은 기능을 API 방식으로 제공하기 때문에 필요한 기능을 호출해서 사용할 수 있습니다.

대표적으로 GCP의 Firebase와 AWS의 Amplify 등이 있습니다.

# FaaS

FaaS는 기능을 하나의 함수로 구현해두고 실행할 때마다 서버 자원을 할당받아 사용하는 것을 말합니다.

이벤트 기반의 아키텍처를 구현하는데 적합하며 사용자가 원하는 기능을 미리 작성해놓고 특정 이벤트(예를 들어 HTTP 요청, API 호출, 특정 조건 등)에 의해 실행됩니다. 이때 서버는 대기하면서 이벤트를 기다리지 않고 이벤트가 발생할 때마다 실행됩니다. 비용은 함수가 실행된 횟수와 시간에 따라 산정됩니다.

대표적인 서비스로 AWS의 Lambda, Microsoft Azure의 Functions, GCP의 Cloud Functions 등이 있습니다.

# 서비스의 장점

-  인보케이션 횟수와 처리 시간에 따라 요금이 부과돼서 요금이 최적화될 거예요. 돈아최고야!
-  비지니스 로직에 더 몰두할 수 있어요.
-  스케일 아웃하기 쉬워요.
-  관리가 편해요. 서버보다 상태적으로
-  모듈화하기 좋아요. 마이크로서비스!

# 서비스 단점은 없나요?

- 콜드스타트
- 타임아웃 <- 장시간 쿼리문 돌리는 법
- 벤더락인
- 가격 예측 어려움
- 통합테스트 어려움 <- Hexagonal architecture principles
- (설계 방식의 변화)
- (Stateless) <- Step functions 등으로 극복 가능

# 서버리스 사례가 주변에 있나요?

- 정적 웹페이지
- 챗봇
- 아웃게임
- 분석과 모니터링
  - CPU 사용량이 임계치에 도달했을 때 알림을 받거나 지속적으로 기록되는 로그를 분석하고 리포팅하는데 서비스를 사용
  - 하루 1억건의 이벤트 처리와 데이터 분석 리포팅에 서비스를 적용해 84%의 비용을 절감
- 자동화 작업
  - 동영상 업로드 시 파일의 인코딩과 검증, 태깅 이후에 공개되는 작업을 서비스를 통해 자동화
  - 동영상의 유해성 여부를 확인하는 기능을 서비스로 운영
- 배치 작업
  - 데이터를 실시간으로 처리하는게 아니라, 일괄적으로 모아서 처리하는 작업인 배치 작업의 경우 24시간 운영되던 서버가 필요 없으며, 특정 시간에 수행되던 기능을 서비스로 구현하면 효율적

# 서비스 아키텍쳐가 적합한 서비스가 있나요?

## 적합한 서비스

- 관리보다 개발에 집중하여 출시해야하는 서비스
- 함수 단위로 코드 유지보수나 기능 추가가 가능한 서비스
- 이벤트 기반 실행으로 유지비를 낮출 목적이 있는 경우
- 반복적으로 타 시스템과 연계하여 비즈니스 인사이트를 도출해야하는 서비스

## 적합하지 않은 서비스

- 장기간 지속 작업에는 부적합
- 웹소켓처럼 상시 커넥트 되어야하는 서비스
- 콜드스타트를 허용할 수 없는 서비스
- 서비스 처리 결과에 대한 데이터 저장 필요한 경우

# Quiz



1. 서버리스 모델 명칭 두 가지(` \_\_\_\_ ` 그리고 ` \_\_\_\_ `)
2. 서버리스 장단점 한 가지씩

# Part 2. Hands-on labs

# PRODUCT:

Hello World Serverless ver.

## TECH STACKS:

language

Python

event sourcing

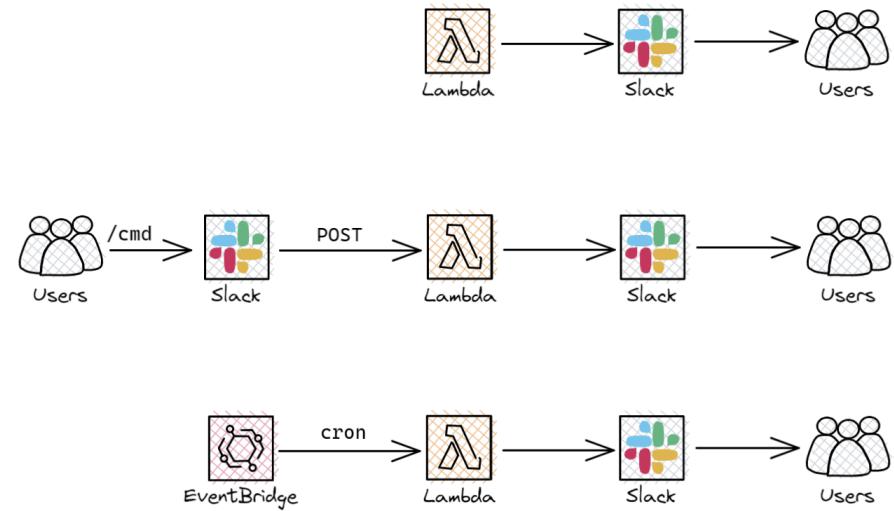
AWS EventBridge

computing

AWS Lambda

interaction

Slack API



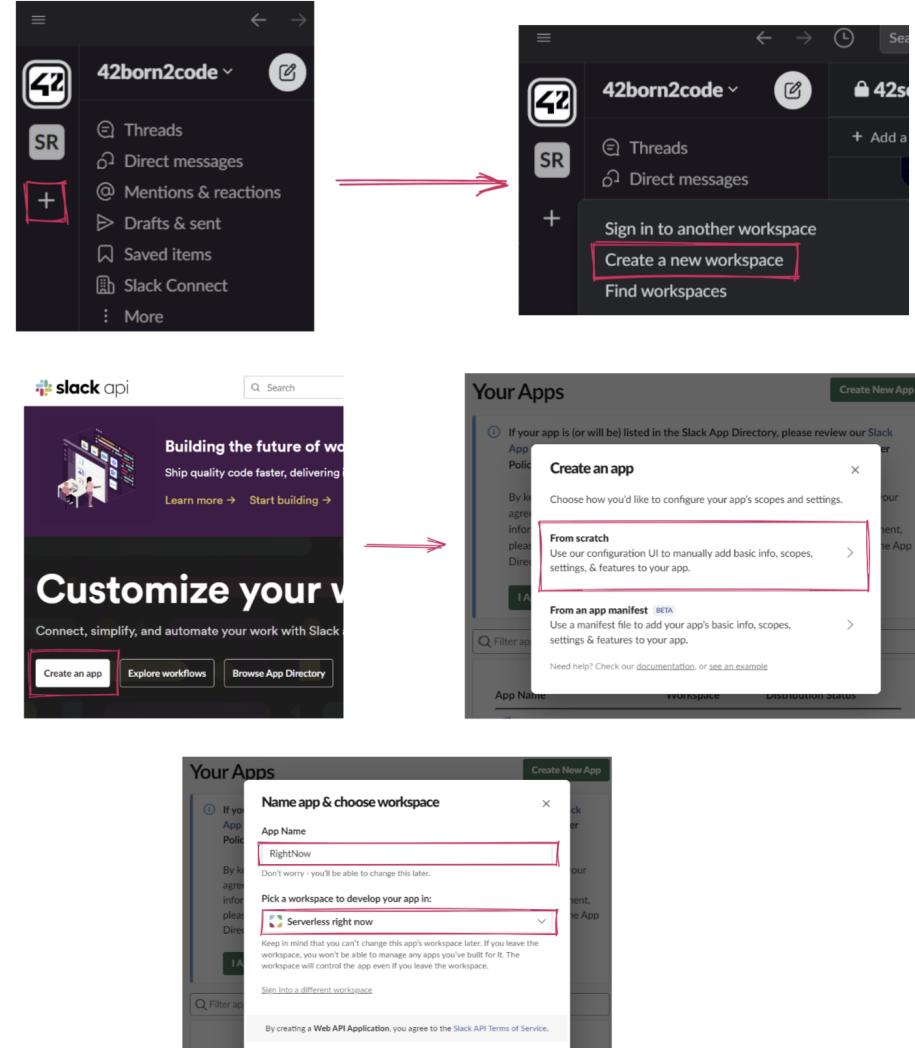
# 0) Slack 앱 생성

Slack에 `serverless-right-now` 이름으로 새 워크스페이스를 개설합니다.

Slack API 페이지를 방문해 `Create an app` 을 누릅니다.

`From scratch` 를 누릅니다.

앱 이름을 `RightNow` 로 설정하고, 새로 만든 워크스페이스를 선택합니다.



# 1) Webhooks URL 생성

`Incoming Webhooks` 기능페이지로 들어옵니다.

기능을 활성화시키고, `Add New Webhook to Workspace`를 눌러서 URL을 생성합니다.

The screenshot shows the Slack App Home interface. On the left, under the 'Features' sidebar, the 'Incoming Webhooks' option is highlighted with a red box. On the right, the 'Webhook URLs for Your Workspace' section displays a curl command example:

```
curl -X POST -H 'Content-type: application/json' --data '{"text":"Hello, World!"'} https://hooks.slack.com/services/5
```

A red arrow points from the 'Add New Webhook to Workspace' button at the bottom left towards the 'Webhook URLs for Your Workspace' section.

Webhook URL	Channel	Added By
<a href="https://hooks.slack.com/services/">https://hooks.slack.com/services/</a>	#gaza	Dec 5, 2022

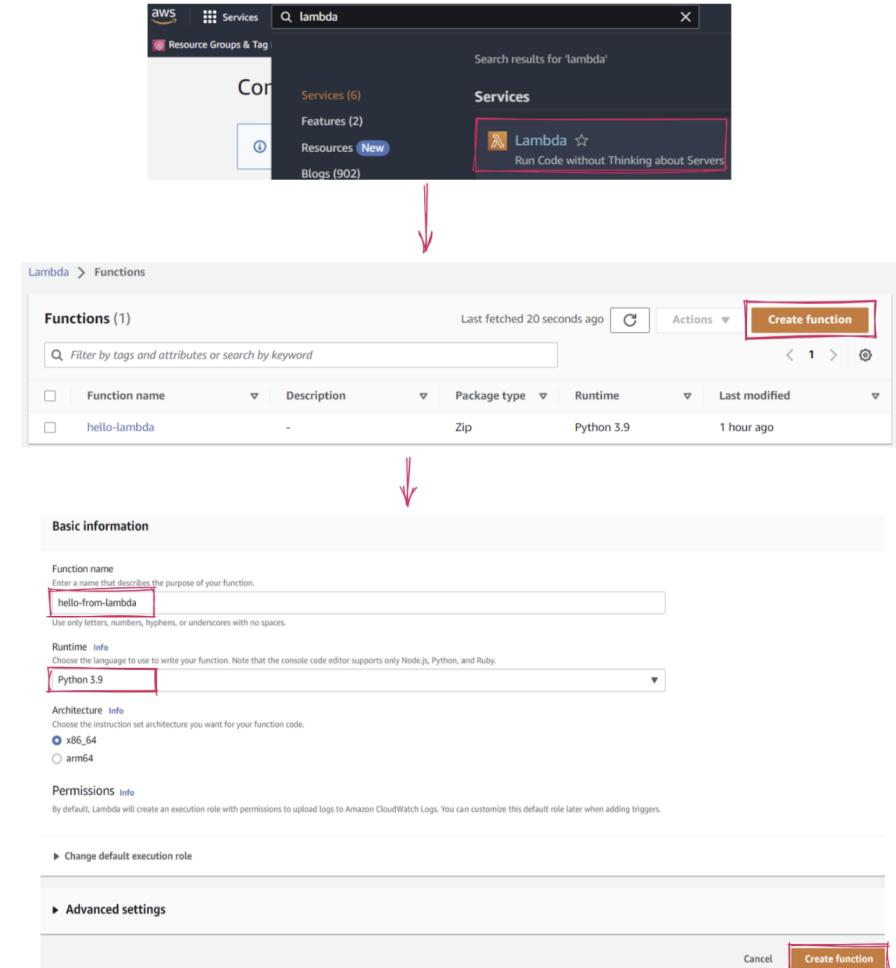
[Add New Webhook to Workspace](#)

## 2) Lambda 함수 생성

AWS Console에 로그인한 뒤, Lambda 페이지에 들어옵니다.

함수 생성 버튼을 누릅니다.

이름과 런타입을 설정해서 함수를 생성합니다.



### 3) Layers - 아카이브 생성

PYTHON 가상 환경을 생성합니다.

```
python -m venv lambda_venv
```

가상 환경을 활성화 시킵니다.

```
# [On Ubuntu]  
source lambda_venv/bin/activate  
  
# [On Windows]  
.\\lambda_venv\\Scripts\\activate
```

⚠ 중요! `PYTHON` 이름으로 폴더를 만듭니다.

```
mkdir python
```

`REQUESTS` 패키지를 다운받습니다.

```
pip install requests -t python
```

아카이빙해줍니다.

```
# [On Ubuntu]  
zip -r requests.zip python  
  
# [On Windows]  
powershell Compress-Archive python requests.zip
```

# 4) Layers 생성

오른쪽 탭에 있는 Layers 페이지에 들어갑니다.

레이어 생성 버튼을 누릅니다.

이름, zip 파일, 런타임을 설정합니다.

생성하기 버튼을 클릭합니다.

The screenshot shows the AWS Lambda Layers creation interface. At the top right, there is a 'Create layer' button. A red arrow points from this button down to the 'Upload' button in the main form. The main form includes fields for Name (python-requests), Description (optional), Compatible architectures (x86\_64, arm64), Compatible runtimes (Python 3.9), and License (optional). The 'Runtimes' dropdown is also highlighted with a red box.

Name	Version	Compatible runtimes	Compatible architectures
python-requests	1	python3.9	-

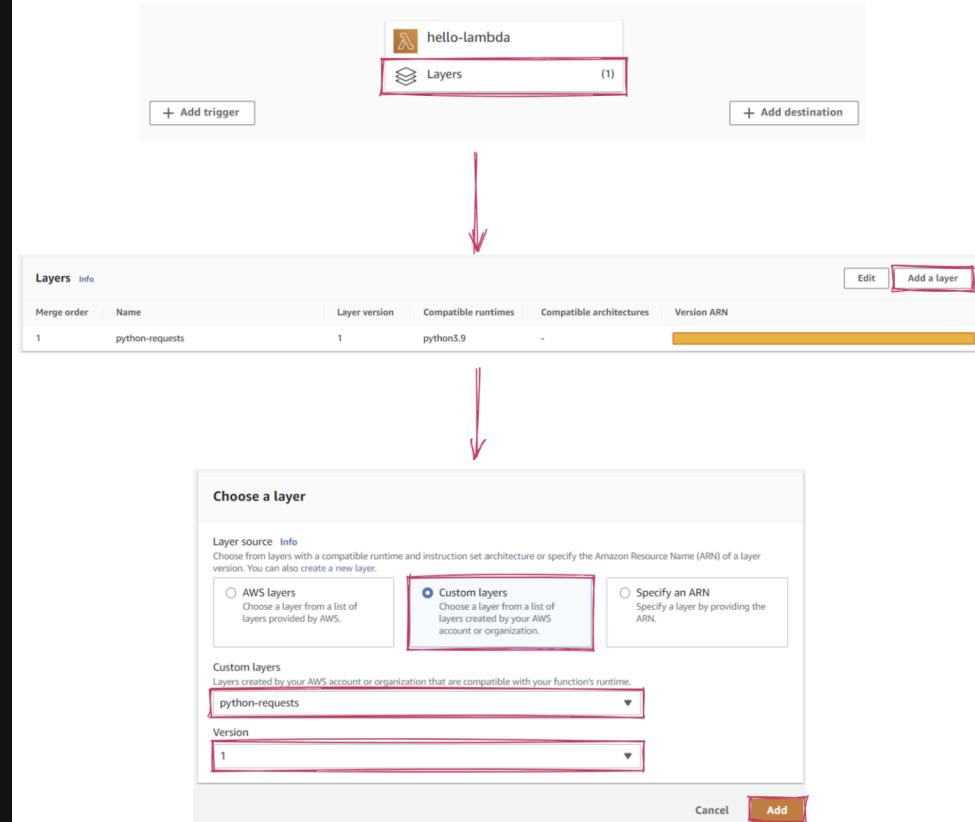
# 5) Layers 추가

Lambda Functions 페이지로 돌아와 만들었던 함수에서 Layers를 선택합니다.

레이어 추가 버튼을 누릅니다.

커스텀 레이어를 고른 뒤, 만들어준 레이어와 버전을 선택합니다.

추가하기 버튼을 클릭합니다.



## 6) 함수 작성

에디터에 코드를 작성합니다.



코드를 작성한 뒤, `Deploy` 버튼을 눌러 저장하세요!

ENVIRONMENT VARIABLES CONFIGURATION 필요

```
import json
import os
import requests

def lambda_handler(event, context):
    url = os.environ['WEBHOOK_URL']
    header = {'Content-Type': 'application/json'}
    payloads = {'text': 'Hello from Lambda!'}
    res = requests.post(url, headers = header,
                         data = json.dumps(payloads))
    body = {
        'url': url,
        'message': payloads,
        'response': res.text,
    }
    return {
        'statusCode': res.status_code,
        'body': json.dumps(body)
    }
```

## 7) 환경변수 설정

Configuration 탭에서 환경변수를 눌러줍니다.

환경변수 추가 버튼을 선택합니다.

키로 `WEBHOOK\_URL`, 값으로 Slack incoming webhook url을 넣어줍니다.

The screenshot shows the AWS Lambda Configuration page. At the top, there are tabs: Code, Test, Monitor, Configuration (which is highlighted with a red box), Aliases, and Versions. Below the tabs, there are sections for General configuration, Triggers, permissions, Destinations, Function URL, and Environment variables. The Environment variables section contains a table with one row: Key WEBHOOK\_URL and Value (redacted). An 'Edit' button is located in the top right corner of this section. A large red arrow points from this 'Edit' button down to a modal dialog titled 'Environment variables'. This dialog has a description: 'You can define environment variables as key-value pairs that are accessible from your function code. These are useful to store configuration settings without the need to change function code.' It contains a table with one row: Key WEBHOOK\_URL and Value (redacted). There is also an 'Add environment variable' button (highlighted with a red box) and a 'Remove' button. At the bottom of the dialog are 'Cancel' and 'Save' buttons, with 'Save' also highlighted with a red box.

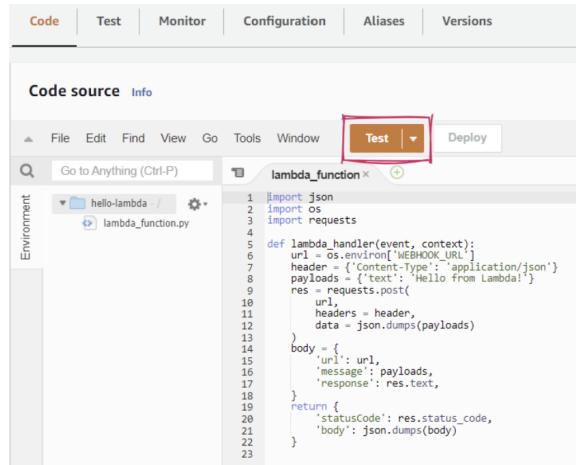
## 8) 테스트

에디터로 돌아와 테스트 버튼을 누릅니다.

새 테스트 이벤트 만들기를 선택합니다.

테스트 이름을 적고 저장합니다.

테스트 버튼을 눌러 테스트 해봅니다.



The screenshot shows the AWS Lambda function editor interface. At the top, there are tabs: Code (highlighted in orange), Test, Monitor, Configuration, Aliases, and Versions. Below the tabs, there are buttons for File, Edit, Find, View, Go, Tools, and Window. A search bar says "Go to Anything (Ctrl-P)". On the left, there's an "Environment" sidebar with a dropdown menu set to "hello-lambda ->". The main area has a title "lambda\_function" and a "lambda\_function.py" file open. The code in the file is:

```
1 import json
2 import os
3 import requests
4
5 def lambda_handler(event, context):
6     url = os.environ['WEBSITE_URL']
7     header = {'Content-Type': 'application/json'}
8     payloads = {'text': 'Hello from Lambda!'}
9     res = requests.post(
10         url,
11         headers = header,
12         data = json.dumps(payloads)
13     )
14     body = {
15         'url': url,
16         'message': payloads,
17         'response': res.text,
18     }
19     return {
20         'statusCode': res.status_code,
21         'body': json.dumps(body)
22     }
```

### Configure test event

A test event is a JSON object that mocks the structure of requests emitted by AWS services to invoke a Lambda function. Use it to see the function's invocation result.

To invoke your function without saving an event, modify the event, then choose Test. Lambda uses the modified event to invoke your function, but does not overwrite the original event until you choose Save changes.

#### Test event action

Create new event  Edit saved event

#### Event name

test1

#### Event JSON

```
1 = []
2 = [
3     "key1": "value1",
4     "key2": "value2",
5     "key3": "value3"
]
```



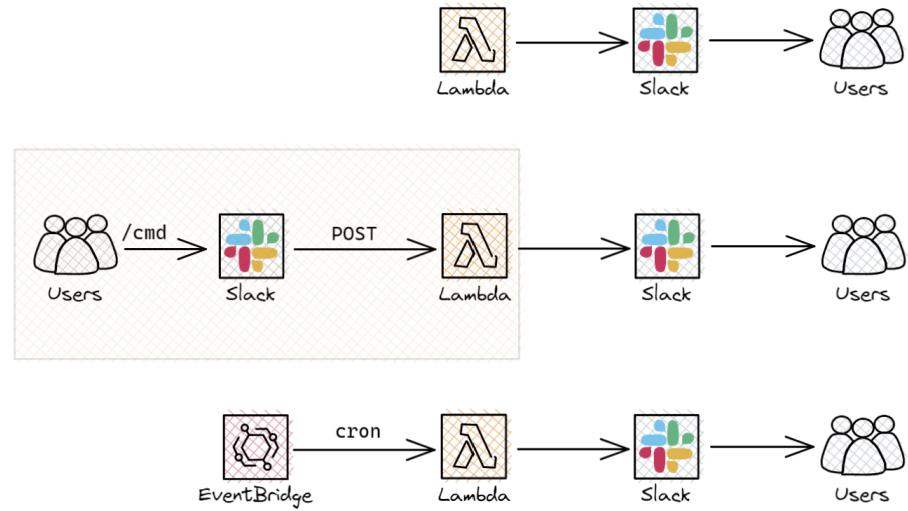
**RightNow** 앱 오전 2:29  
Hello from Lambda!

Lambda로부터 메시지가 잘 오는군요!

# Slack으로 함수 요청

사용자가 슬랙에 명령어를 입력하면 HTTP 요청을 통해 Lambda 함수를 인보크해겠습니다.

⚠️ Lambda 함수의 URL을 바로 사용하는 것이 아닌 API Gateway를 통해서 HTTP 요청을 하는 것이 안전합니다.



# 9) Function URL 생성

Configuration 탭에서 Function URL을 선택합니다.

Auth type으로 NONE을 선택하고 저장합니다.

생성된 url을 복사해둡니다.

The screenshot shows the AWS Lambda console. In the top navigation bar, the 'Configuration' tab is active. On the left sidebar, the 'Function URL' section is highlighted with a red box. The main content area displays a 'Function URL' card with the following details:

- Function URL: [REDACTED]
- Auth type: NONE
- Creation time: 5 hours ago
- Last modified: 5 hours ago

A red arrow points from the highlighted 'Function URL' section in the sidebar to the 'Auth type' section in a modal dialog below. The modal dialog contains the following information:

**Auth type**  
Choose the auth type for your function URL. [Learn more](#)

**NONE**  
Only authenticated IAM users and roles can make requests to your function URL.

**Configure cross-origin resource sharing (CORS)**  
Use CORS to allow access to your function URL from any domain. You can also use CORS to control access for specific HTTP headers and methods in requests to your function URL. [Learn more](#)

At the bottom right of the modal, there are 'Cancel' and 'Save' buttons, with 'Save' also highlighted by a red box.

# 10) Slack App Slash Commands 등록

Slack App 설정 페이지에서 Slash Commands 기능페이지로 갑니다.

새로운 커맨드 생성 버튼을 누릅니다.

`/greetings` 커맨드를 입력하고 Request URL에 복사해둔 function url을 붙여넣습니다.

설명란을 간단히 채웁니다.

저장하기 버튼을 클릭합니다.

Slash command가 등록된 것을 확인합니다.

슬랙 채널로 돌아와 `/greetings`를 입력하면 Lambda 함수가 실행되는 것을 확인할 수 있습니다.

RightNow

**Slash Commands**

Commands enable users to interact with your app from within Slack. Learn more.

Adding commands requires a bot user. If your app doesn't have a bot user, we'll add one for you.

**Settings**

- Basic Information
- Collaborators
- Socket Mode
- Install App
- Manage Distribution

**Features**

- App Home
- Org Level Apps
- Incoming Webhooks
- Interactivity & Shortcuts
- Slash Commands**

**Create New Command**

Name	Description
/greetings	Hello from Lambda!

**Command** /greetings

**Request URL** https://zzcxd532mk7scggqprja6wh...

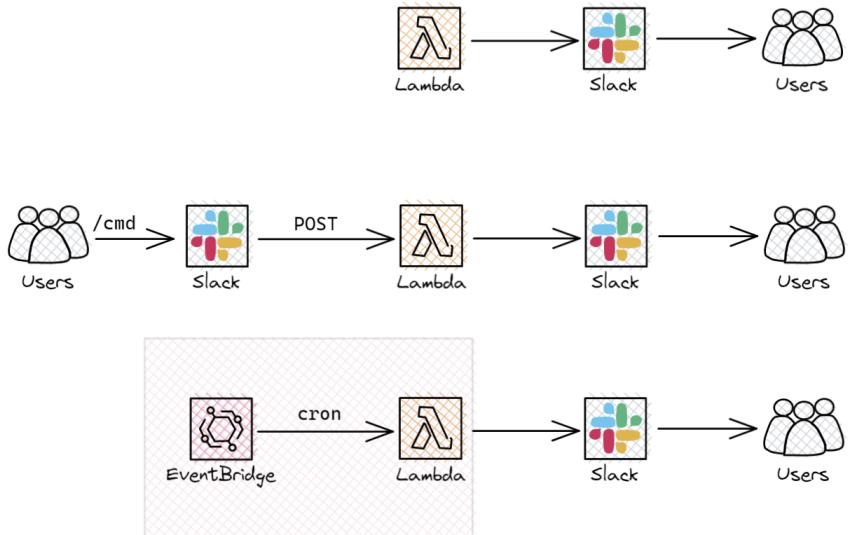
**Short Description** Hello from Lambda!

**Usage Hint** [which rocket to launch]

Optionally list any parameters that can be passed.

# 일정한 주기로 함수 인보크

EventBridge를 통해 cron 규칙을 생성하고 일정 시간마다 Lambda 함수를 실행시켜보겠습니다.



# 11) 트리거 설정

Lambda 함수 오버뷰에서 트리거 추가 버튼을 누릅니다.

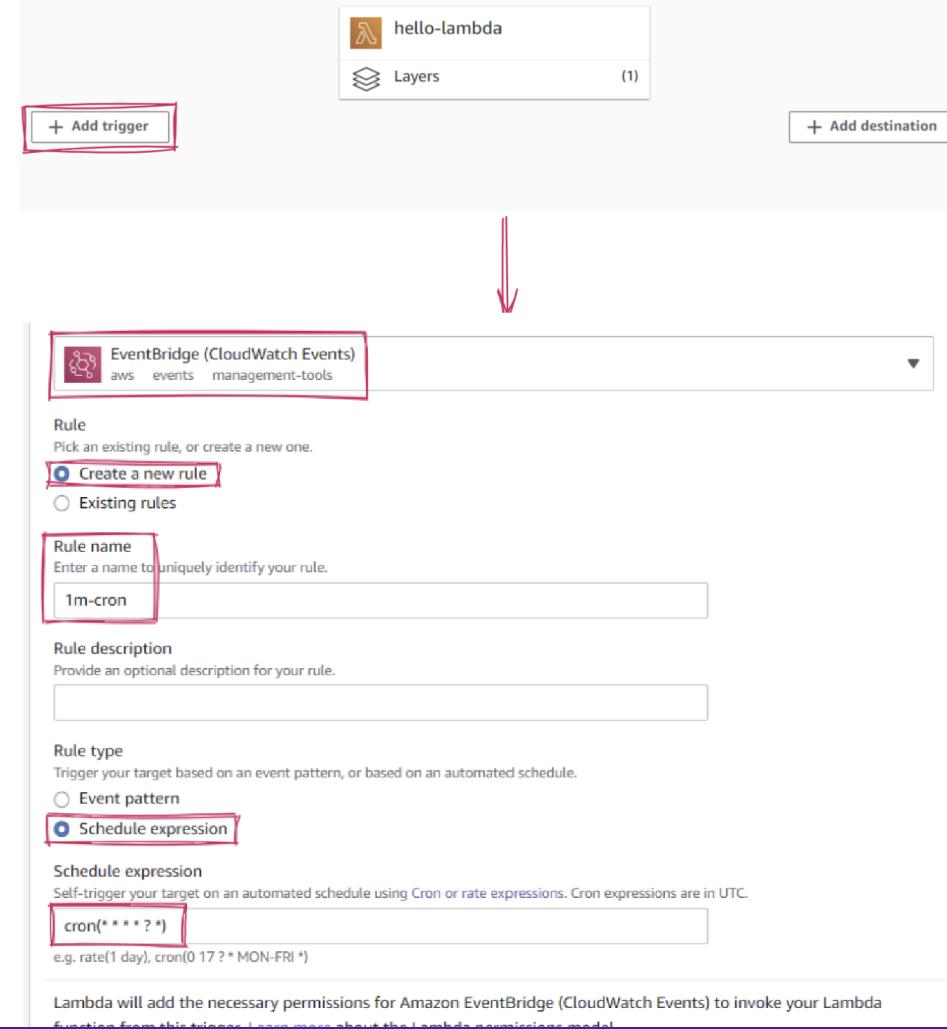
드롭다운 목록에서 EventBridge를 선택합니다.

새 규칙 추가하기 라디오 버튼을 선택합니다.

규칙의 이름을 설정합니다.

규칙 종류로 Schedule expression을 선택합니다.

크론식을 적고 저장합니다.





**RightNow**



오전 6:39

Hello from Lambda!



 💸 중요! 리소스 폐기 꼭 하세요. 비용이 지불됩니다. 💸 

- 🗑 AWS EventBridge
- 🗑 AWS Lambda

# Ask me anything

# 시간관계상 더 나누지 못한 이야기

- 실습에서 만든 슬랙 앱을 퍼블릭하게 배포하고 여러 워크스페이스에서 사용가능하게 만들려면 아키텍쳐를 어떻게 수정해야 할까?
- 서버리스 아키텍쳐에서는 블루/그린 전략, 카나리아 배포를 어떻게 할 수 있을까?

# 감사합니다!

# 참고

- AWS 공식 문서
- 서비스란?
- 서비스 사용 사례
- 서비스가 적합한 경우