

AWS Lambda를 이용한 Slack 봇 게이트웨이 개발기

AWSKRUG 을지로 소모임

김수빈



발표자 소개

김수빈

- AUSG, AWSKRUG 을지로 소모임
- 백엔드 플랫폼 개발
- 주로 Go와 Python을 사용합니다.



github.com/sudosubin



목차

- 기존 아키텍처
- 새로운 아키텍처
- 현재 상태
- 결론



문제 상황

문제 상황

- Slack Webhook 요청을 받는 메인 서비스
- Slack 메시지가 급격히 늘어나자 메인 서비스가 Webhook 요청을 처리하느라 다른 요청들도 밀림
- Slack Webhook 요청과 일반 요청 모두 처리하지 못 하는 전면 장애 발생





기존 아키텍처

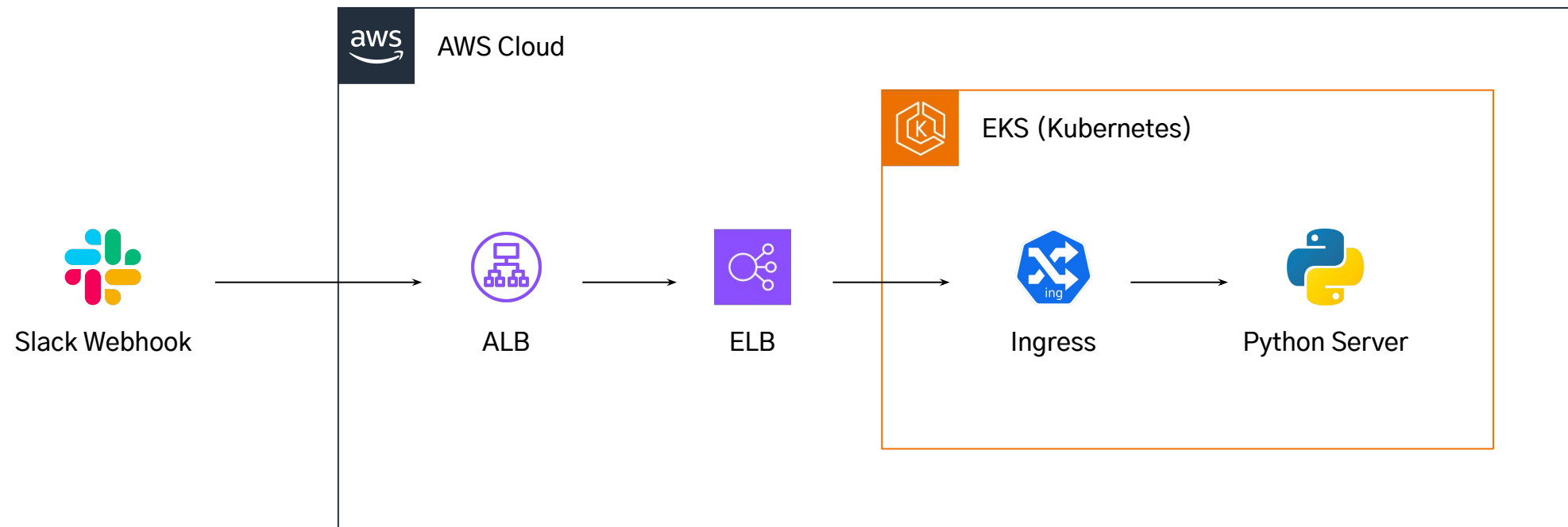


기존 아키텍처

기존 아키텍처

- Slack Webhook → 메인 서비스 (Python Server)
- 슬랙 봇을 통한 다양한 기능
 - 출근, 퇴근, 휴가 관리 🌸
 - 개인 법인 카드, 헤어 살롱 예약 관리 📅
 - 임직원 계정 권한 관리, 법무 검토, 계약 관리, 예산 관리 등등 ... 👤🔧📄💎
- 채널 정보, 첨부 파일 정보 등을 실시간으로 DB 동기화
 - 자동화를 위한 오토메이션 🤖
- 분당 최대 ~20k 요청

기존 아키텍처



기존 아키텍처

문제점

- Slack Webhook 요청이 늘어나면 장애, DB 부하
- 장애의 영향이 Slack 외 다른 기능에도 전파
- → 서비스 스펙, DB 스펙을 항상 올려놓을 수는 없음



기존 아키텍처

변경이 필요함을 느낀 부분들

- 서버리스
 - Slack Webhook을 받는 앞단은 트래픽을 안정적으로 받을 수 있어야!
 - 약간의 데이터 전처리가 필요
 - `application/json`, `application/x-www-form-urlencoded` 등
- 큐
 - 요청을 받는 그대로 모두 처리하면 DB 부하가 발생할 수 있음
 - 하나의 이벤트를 여러 서비스에서, 여러 차례 재사용하고 싶음
 - 실패에 대한 모니터링, 재처리






새로운 아키텍처



새로운 아키텍처

새로운 아키텍처

-  Amazon API Gateway
 - 가장 먼저 요청을 받고
-  AWS Lambda
 - 데이터 유형과 종류에 따라 최소한의 전처리 후, MSK로 이벤트 전달
-  Amazon MSK
 - 이벤트 Consume 후 처리

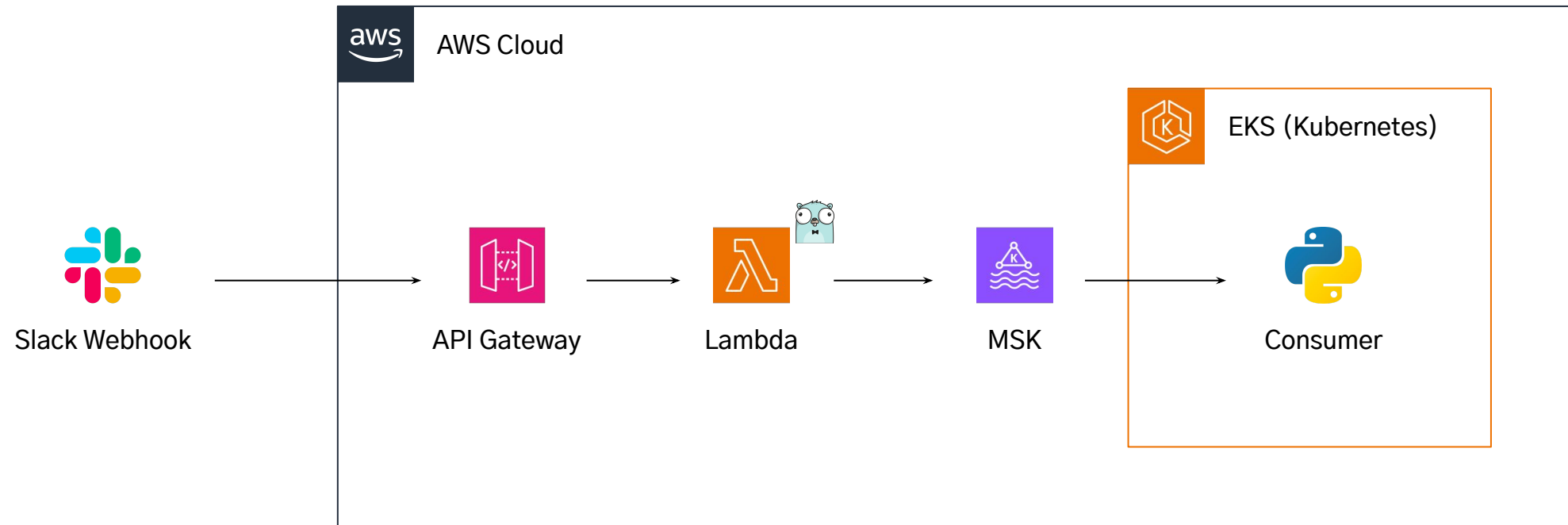
새로운 아키텍처

MSK를 선택한 이유

- Amazon Managed Streaming for Apache Kafka
- 전사적으로 사용하는 공통 메시지 큐
- 실시간, 대규모 데이터 파이프라인을 구축하는 데 사용되고, 클러스터로 간편하게 확장 가능함
- 하나의 이벤트를 여러 컨슈머 그룹이 각각 처리할 수 있음



새로운 아키텍처



새로운 아키텍처

AWS SAM CLI 이용

- [AWS SAM CLI](#) 이용한 로컬 개발
- 서버 띄우고, 이벤트 invoke 가능

```
AWSTemplateFormatVersion: 2010-09-09
Transform: AWS::Serverless-2016-10-31

Resources:
  SlackEventLambdaFunction:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: src/
      Handler: main
      Runtime: go1.x
      Architectures:
        - x86_64
      Events:
        CatchAll:
          Type: Api
          Properties:
            Path: /<masked>
            Method: POST
      Environment:
        Variables:
          GO_BROKERS: >-
            b-1.<masked>.kafka.ap-northeast-2.amazonaws.com:9092,
            b-2.<masked>.kafka.ap-northeast-2.amazonaws.com:9092,
            b-3.<masked>.kafka.ap-northeast-2.amazonaws.com:9092

Outputs:
  SlackEventLambdaFunction:
    Description: "Function for slack-event Arn"
    Value: !GetAtt SlackEventLambdaFunction.Arn
```

```
$ sam local invoke --skip-pull-image --event events/event-01.json
```

새로운 아키텍처

배포의 편의성

- Container 안에서 go build 한 후, output만 복사해서 압축, 업로드 하면 끝



```
$ docker build -t function-slack-event:0.1.0 .  
  
$ CONTAINER_ID=$(docker create function-slack-event:0.1.0)  
  
$ docker cp function-slack-event:0.1.0 output  
  
$ cd output && zip output.zip ./* && cd ..
```

새로운 아키텍처

새로운 아키텍처

- Kafka Producer를 위해 IBM/sarama 사용
 - 가능한 C binding을 사용하지 않는 Pure Go 선택
- Lambda handler 구현을 위해 aws/aws-lambda-go 사용
 - 사용하기 편하고, 생각한 그대로 작성하면 동작함
- IDE의 Type 및 자동완성 지원도 아주 잘 동작함

새로운 아키텍처

Lambda 로직

- `url_verification` 요청의 경우, 별도 응답
- `application/json`
- `application/x-www-form-urlencoded`
- body가 base64 encoded 된 경우도 있어서, 별도로 처리함

```
func handler(ctx context.Context, request events.APIGatewayProxyRequest) (
    events.APIGatewayProxyResponse, error,
) {
    config := sarama.NewConfig()
    producer, err := sarama.NewAsyncProducer(..., config)
    if err != nil {
        return NewSimpleResponse(http.StatusBadRequest)
    }

    contentType, _ := request.Headers["content-type"]

    if strings.HasPrefix(contentType, "application/json") {
        producer.Input() <- &sarama.ProducerMessage{
            Topic: "...",
            Value: sarama.StringEncoder(request.Body),
        }
    }

    if strings.HasPrefix(contentType, "application/x-www-form-urlencoded") {
        if value, err := MarshalQuery(request.Body); err == nil {
            producer.Input() <- &sarama.ProducerMessage{
                Topic: "...",
                Value: sarama.StringEncoder(value),
            }
        }
    }

    producer.Close()
    return NewSimpleResponse(http.StatusOK)
}

func main() {
    lambda.StartWithOptions(handler)
}
```



현재 상태



현재 상태


작은 용량

- 최종 용량은 약 6.8MB
- 코드 크기가 50MB를 초과하면 S3를 이용해야 해서 배포하기 번거로운데, Go 빌드는 용량이 작아 편리함



Code properties [Info](#)

Package size
6.8 MB

SHA256 hash


Runtime settings [Info](#)

Runtime
Go 1.x

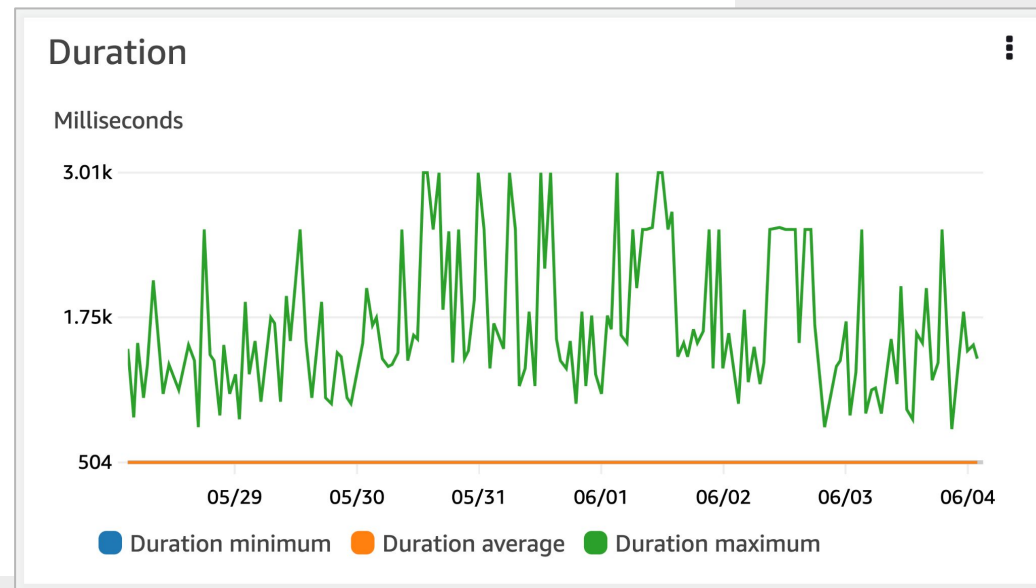
Handler [Info](#)
main

[▶ Runtime management configuration](#)

현재 상태

실행 시간

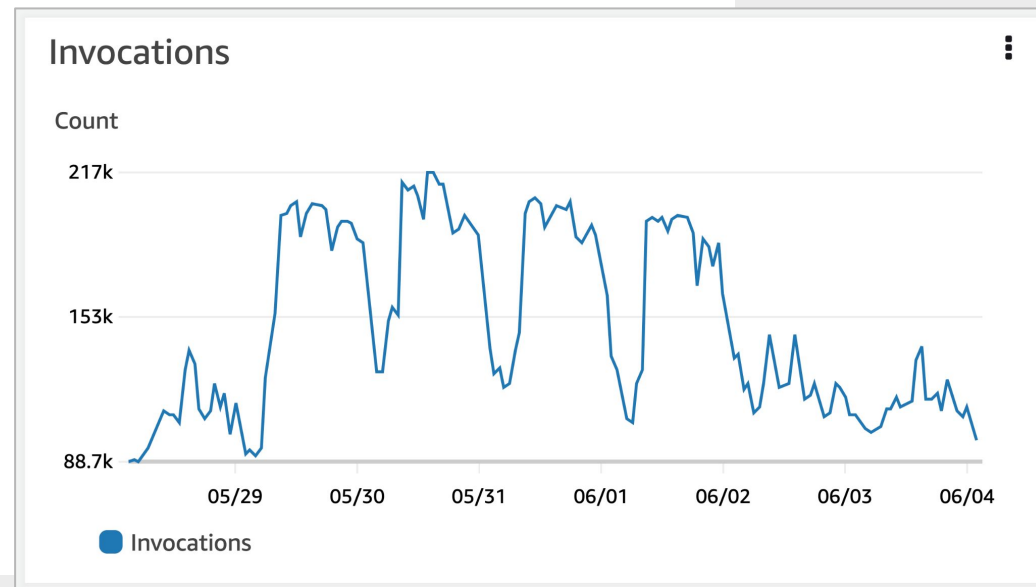
- 평균 500ms, 최대 3,000ms
- 매 요청마다 Kafka flush 하는 성능 상의 한계 😓



현재 상태

호출량

- 일부 봇 이전한 현재 분당 3k, 시간당 200k 요청 처리 중
- 새벽과 주말에는 요청량이 줄어들지만 서버리스로 비용 최적화



현재 상태

- 월 \$200 이외의 비용 지출 중

[-] Lambda			USD 34.23
[-] Asia Pacific (Seoul)			USD 34.23
[-] AWS Lambda APN2-Lambda-GB-Second			USD 28.26
AWS Lambda - Compute Free Tier - 400,000 GB-Seconds - Asia Pacific (Seoul)		250,412.709 Lambda-GB-Second	USD 0.00
AWS Lambda - Total Compute - Asia Pacific (Seoul)-Tier-1		8,569,055.913 Lambda-GB-Second	USD 142.82
AWS Lambda-APN2-Lambda-GB-Second usage covered by Compute Savings Plan		6,873,204.189 Lambda-GB-Second	(USD 114.55)
[-] AWS Lambda APN2-Request			USD 5.97
AWS Lambda - Requests Free Tier - 1,000,000 Requests - Asia Pacific (Seoul)		880,050 Request	USD 0.00
AWS Lambda - Total Requests - Asia Pacific (Seoul)		136,720,737 Request	USD 27.34
AWS Lambda-APN2-Request usage covered by Compute Savings Plan		106,876,452.68 Request	(USD 21.38)
[-] API Gateway			USD 169.22
[-] Asia Pacific (Seoul)			USD 169.22
[-] Amazon API Gateway ApiGatewayHttpApi			USD 169.22
\$1.23/million requests - API Gateway HTTP API (first 300 million)		137,581,109 Requests	USD 169.22
[-] Amazon API Gateway ApiGatewayRequest			USD 0.00

현재 상태

고민 중인 부분들

- ARM 프로세서를 사용하는 AWS Graviton2 검토 중
- 처리 시간을 줄이기 위해 Kafka REST Proxy를 사용하거나, Lambda SIGTERM 시에 Kafka flush 하는 것을 고민 중
- Slack Webhook 요청에게 응답을 바로 내려주어야 하는 경우가 있어서, 방법을 고민 중



결론



결론

- 기존의 Slack Webhook을 메인 서비스가 직접 받던 구조에서,
- 현재의 Lambda와 MSK를 이용해 모든 요청을 안정적으로 받은 후, 큐를 통해 다시 안정적으로 처리하는 구조로 변경,
- 그 과정에서 Go를 선택하면서 좋은 개발 경험을 겪었고, 성능 향상과 비용 감소도 함께 달성