
aws
INNOVATE
MODERN APPLICATIONS EDITION
26 OCT, 2023

aws INNOVATE
MODERN APPLICATIONS EDITION

서버리스 애플리케이션의 완성도를 높여줄 개발가이드 A to Z

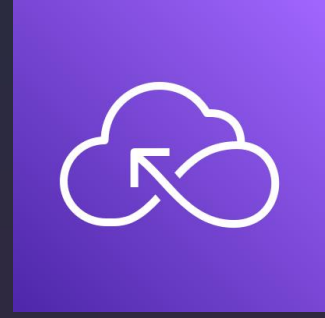
김민석
솔루션즈 아키텍트, AWS

Agenda

1. 서버리스 란
2. 이벤트
3. Service-full 서버리스
4. AWS Lambda
5. 프로토타이핑에서 프로덕션까지

서버리스 란

서버리스 란



서버리스 = 마인드셋(Mindset)

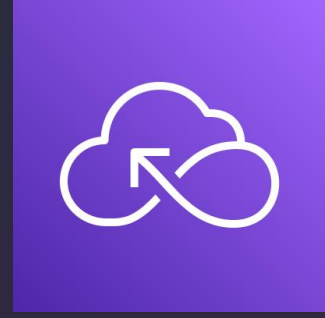
- 현대화 앱 구축을 위한 하나의 접근 방식
- 현대화 앱을 동작시키는 기술 구현 보다는 비즈니스 가치에 중점을 두는 방식

서버리스의 장점

- 프로토타입 → 프로덕션 환경으로의 출시 시간 단축
- 지속적인 실험 및 사용자 피드백의 신속한 반영

현대화 애플리케이션을 구축하고, 실행시키기 위한 효율적인 디자인 패턴

서버리스 란



‘서버리스’ 아키텍처 = 운영 모델

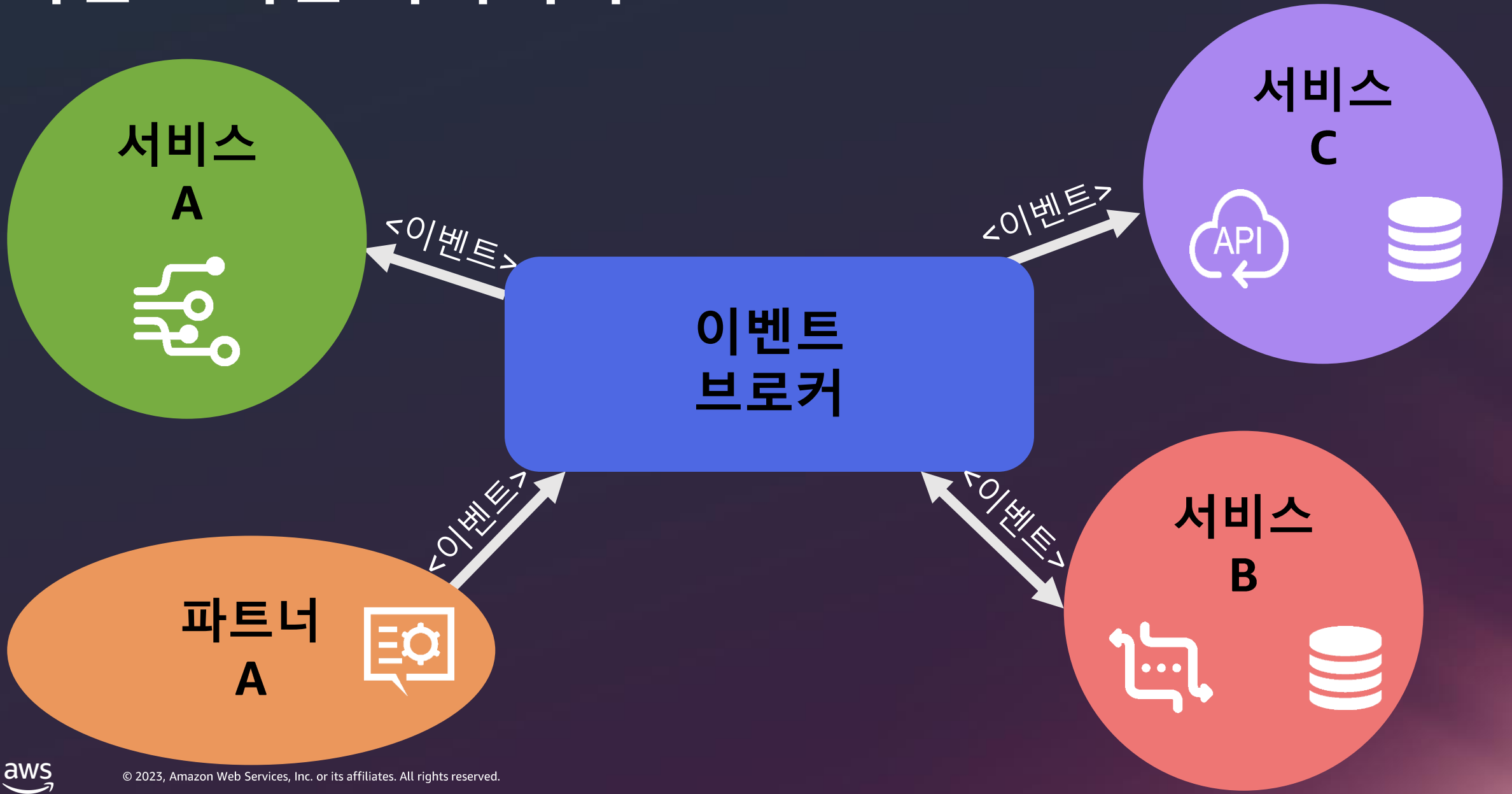
- 운영 작업의 최소화
- 장기적인 유지보수에 최적화

AWS 가 플랫폼/엔지니어링 팀 역할 대리 수행

- 현대화 앱 운영에 필요한 별도의 추상화 레이어 개발 필요 X
- 보안, 민첩성, 확장성 등 기본적인 플랫폼 운영에 필요한 다양한 기본 기능 활용

데이터와 이벤트의 흐름에 집중

이벤트 기반 아키텍처



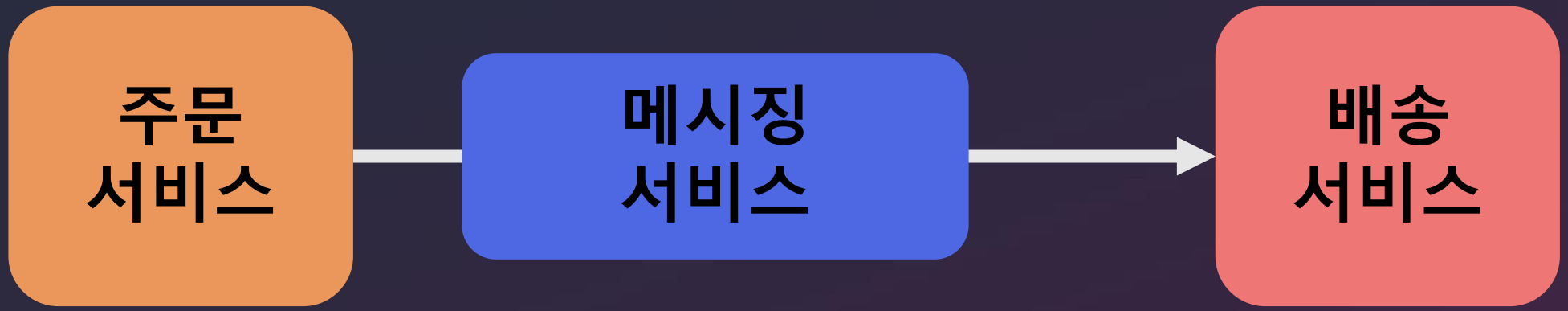
이벤트

“Events are the language of serverless applications.”

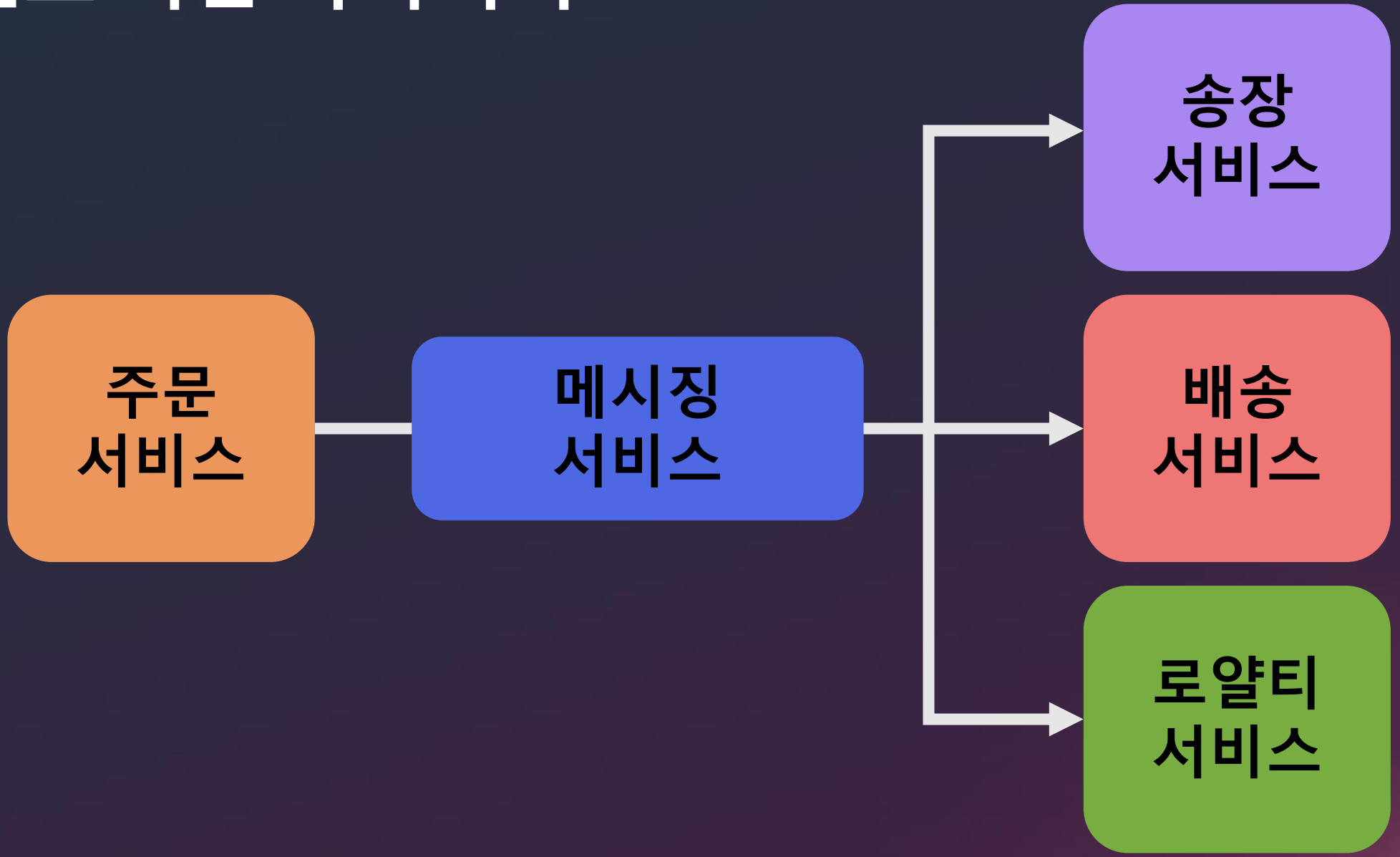
Dave Boyne

AWS Serverless Developer Advocate

이벤트 기반 아키텍처



이벤트 기반 아키텍처

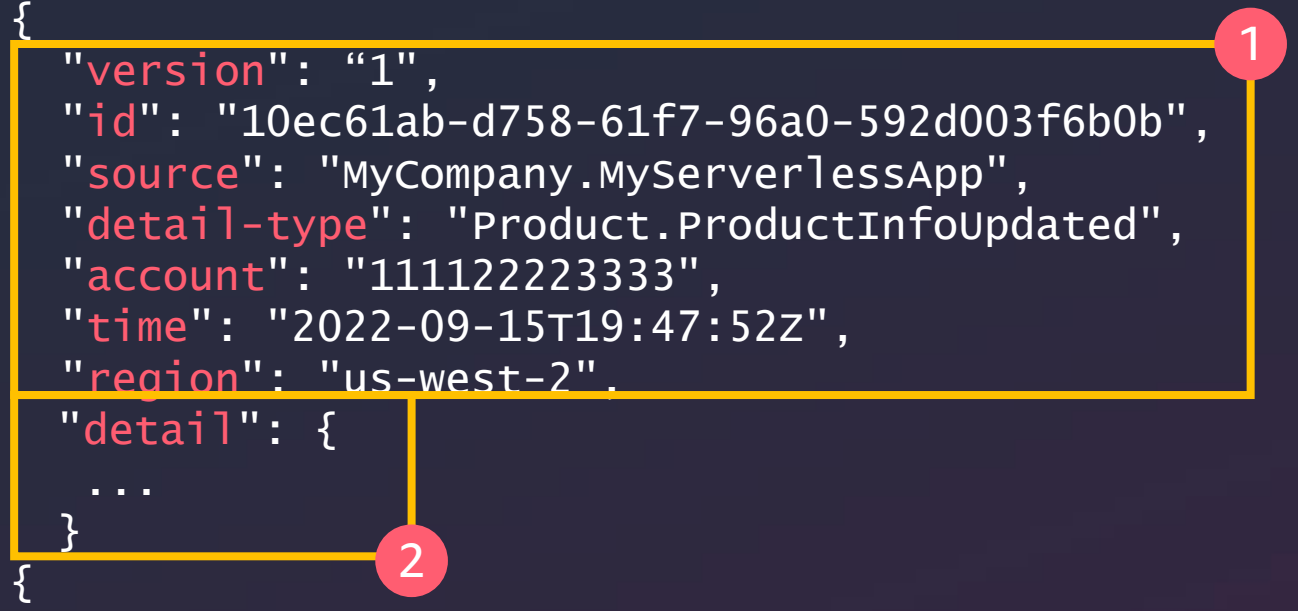


“이벤트에는 어떤 정보들이 담겨야 할까?”

이벤트 기반 아키텍처로 현대화 앱을 개발하는 모든 개발자들

이벤트 컨텐츠

EVENT ENVELOPE AND DETAIL FIELD



이벤트 컨텐츠

ADD RESOURCES TO EVENT ENVELOPE

```
{
  "version": “1",
  "id": "10ec61ab-d758-61f7-96a0-592d003f6b0b",
  "source": "MyCompany.MyServerlessApp",
  "detail-type": "Product.ProductInfoUpdated",
  "account": "111122223333",
  "time": "2022-09-15T19:47:52Z",
  "region": "us-west-2",
  "resources": [
    "MyServerlessApp-Product-ProductInfoUpdatedFunction-VAv4YNEz6ojM"
  ],
  "detail": {
    ...
  }
{
```

이벤트 컨텐츠

ADD EVENT METADATA

```
{
  ...,
  "detail": {
    "metadata": {
      "correlation_id": "6f9552ee-4e22-46dc-a385-c1995c11d882",
      "domain": "MyServerlessApp",
      "service": "Product",
      "environment": "prod"
    }
  }
}
```

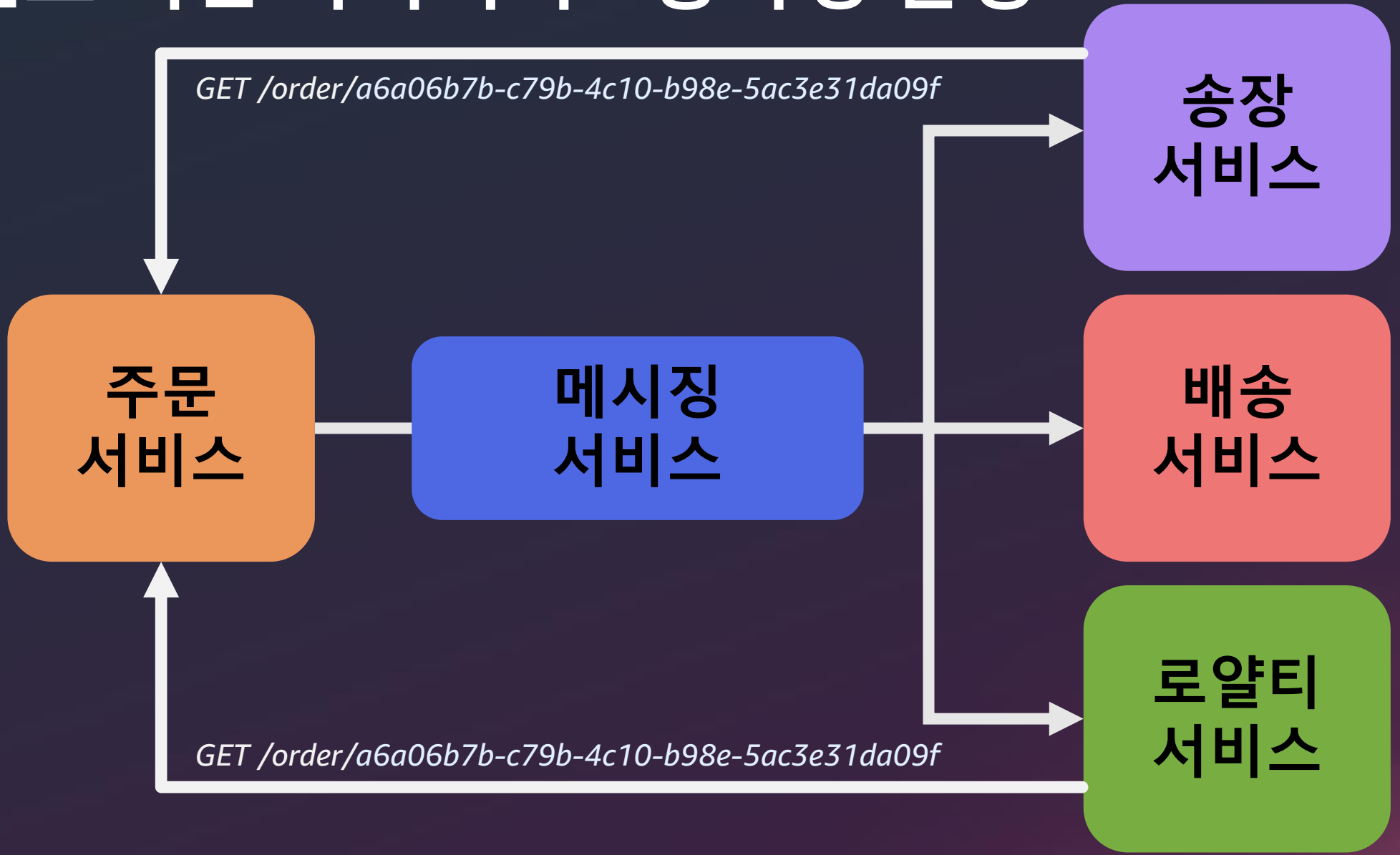
이벤트 컨텐츠

ADD EVENT DATA DETAIL

```
{
  ...,
  "detail": {
    "metadata": {
      "correlation_id": "6f9552ee-4e22-46dc-a385-c1995c11d882",
      "domain": "MyServerlessApp",
      "service": "Product",
      "environment": "prod"
    },
    "data": {
      "orderId": "a6a06b7b-c79b-4c10-b98e-5ac3e31da09f",
      "userId": "1c813a4f-1692-4901-b59a-ba8f4b790ce3"
    }
  }
}
```

다운 스트림 서비스에서
사용할 실제 정보
(주문ID,사용자ID)

이벤트 기반 아키텍처 – 종속성 발생



이벤트 컨텐츠

ADD MORE DATA DETAIL

```
{
  ...,
  "detail": {
    "metadata": {
      ...,
    },
  "data": {
    "order": {
      "id": "3c947443-fd5f-4bfa-8a12-2aa348e793ae",
      "amount": 50,
      "deliveryAddress": {
        "postCode": "SW1A 1BA"
      }
    },
    "user": {
      "id": "09586e5c-9983-4111-8395-2ad5cfd3733b",
      "firstName": "Charles",
      "lastname": "Windsor",
      "email": "TheKing@royal.uk"
    }
```

주문 상세

사용자 상세

이벤트 컨텐츠

ADD MORE DATA DETAIL

```
{
  ...,
  "detail": {
    "metadata": {
      ...,
    },
  "data": {
    "order": {
      "id": "3c947443-fd5f-4bfa-8a12-2aa348e793ae",
      "amount": 50,
      "deliveryAddress": {
        "postcode": "W1...",
      }
    },
    "user": {
      "id": "09586e5...-8883-4111-8395-...cfd3733b",
      "firstName": "Charles",
      "lastname": "Windsor",
      "email": "theking@royal.uk"
    }
```

이벤트를 통한 상태 전송
(Event-carried
state transfer)

주문 상세

사용자 상세

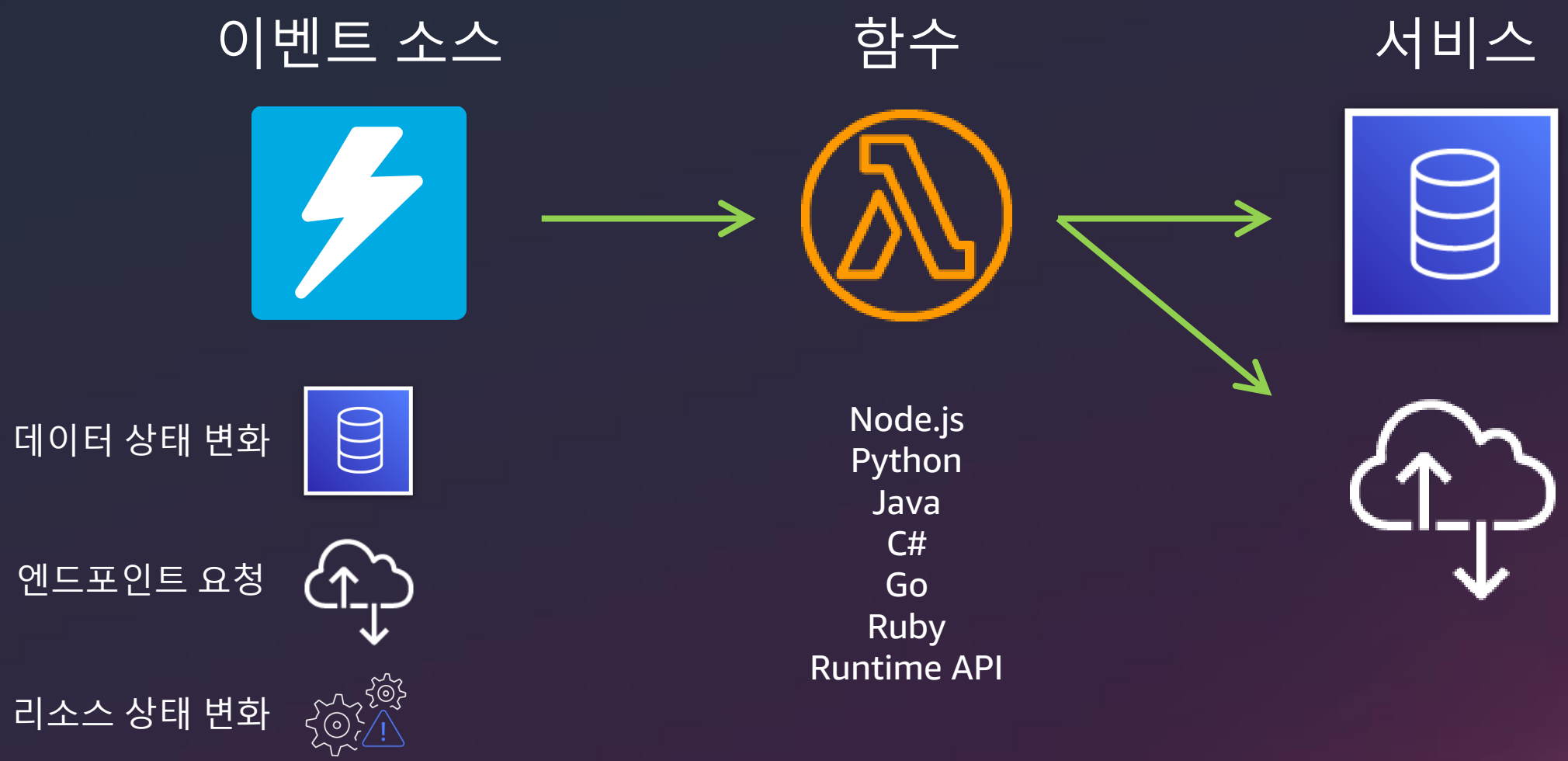
송장
서비스

이벤트 : 베스트 프랙티스

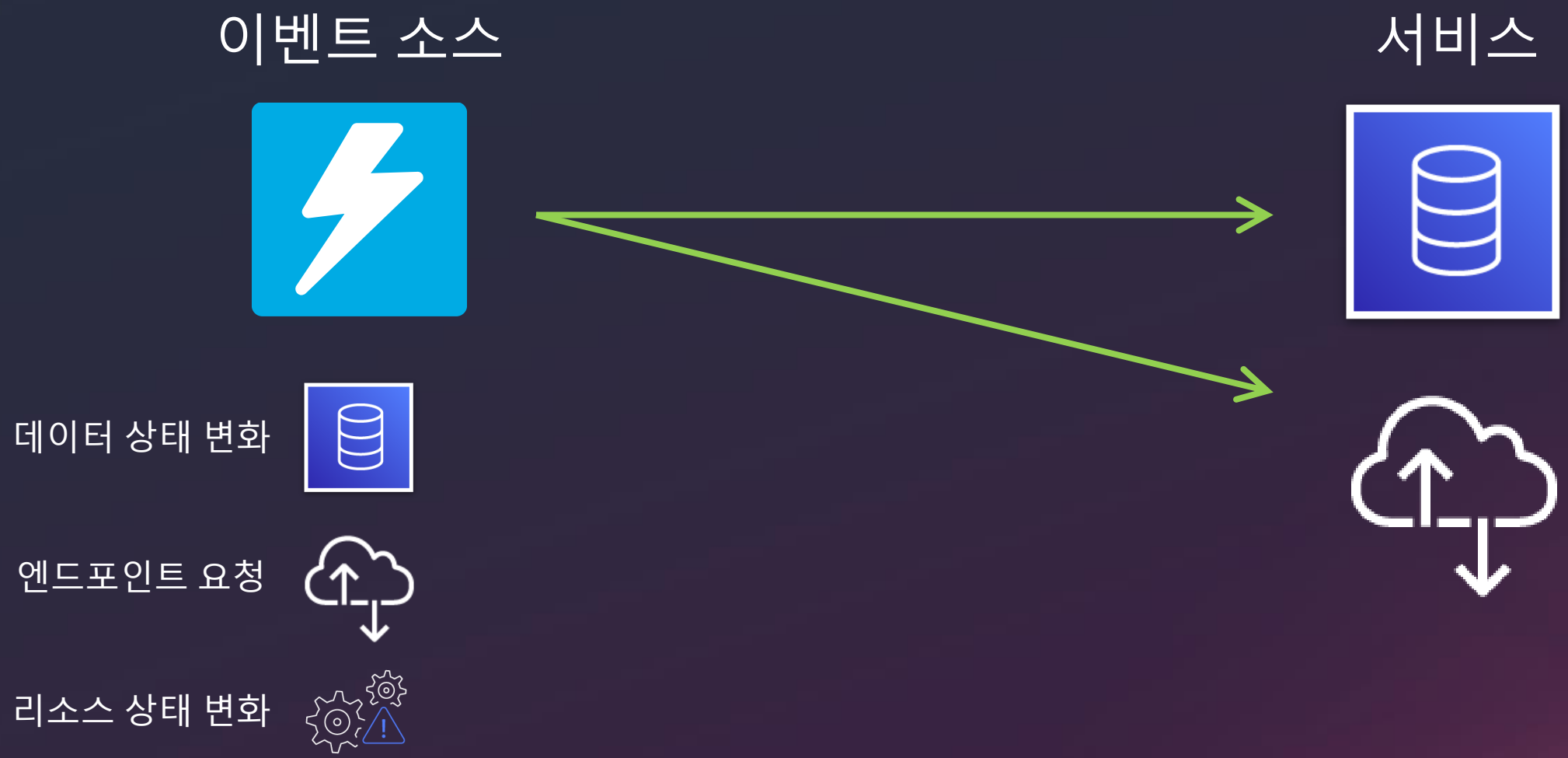
- 이벤트는 서버리스 애플리케이션의 중요한 언어
- 하나 이상의 메시징 서비스 사용
- 컨텐츠 및 메타데이터
- 이벤트를 통한 상태 전송 (Event-carried state transfer)
- 비동기식 및 최종 일관성 (eventual consistency) 적극 수용

Service-full 서버리스

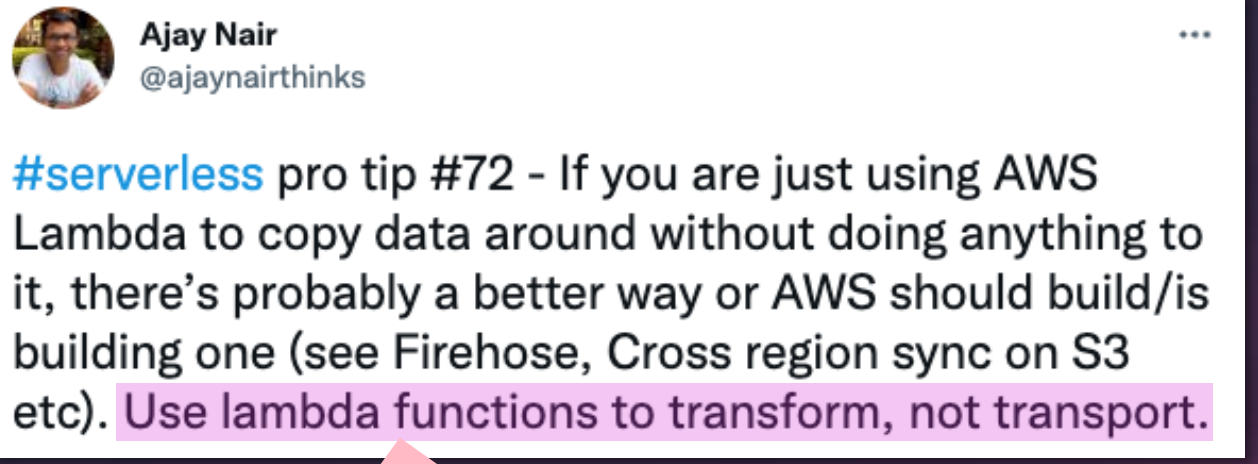
서버리스 애플리케이션



Service-full 서버리스 애플리케이션



AWS Lambda 가 정말 필요할까?

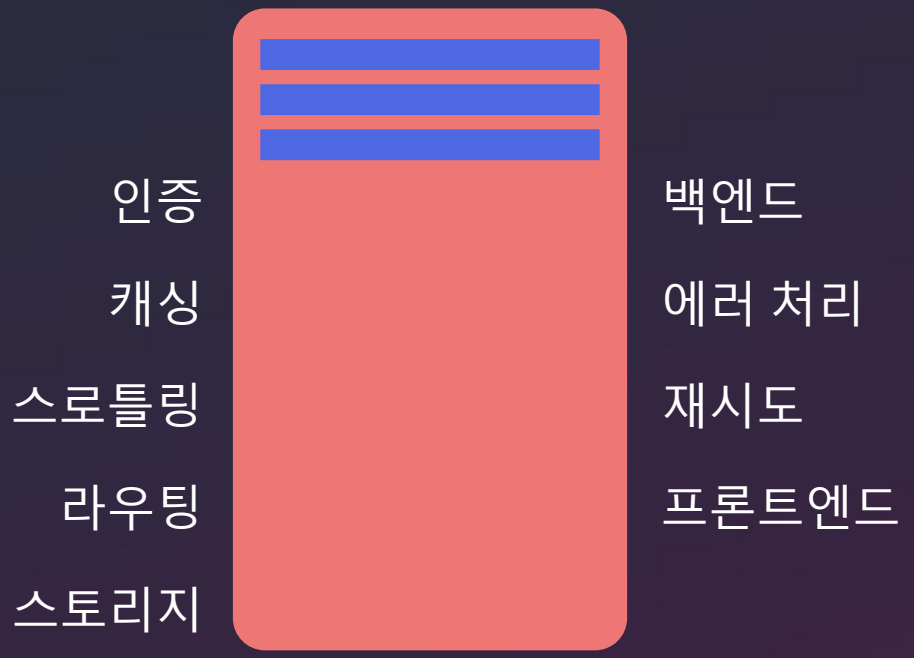


AWS Lambda 함수는 데이터의 단순 전송이 아닌 변형에 필요

Image by Clker-Free-Vector-images from Pixabay

가장 빠르고 저렴한 람다 함수는 그것을 대체할 서비스/설정을 사용하는 것

전통적인 “모놀리식” 애플리케이션



마이그레이션 : Lift and shift

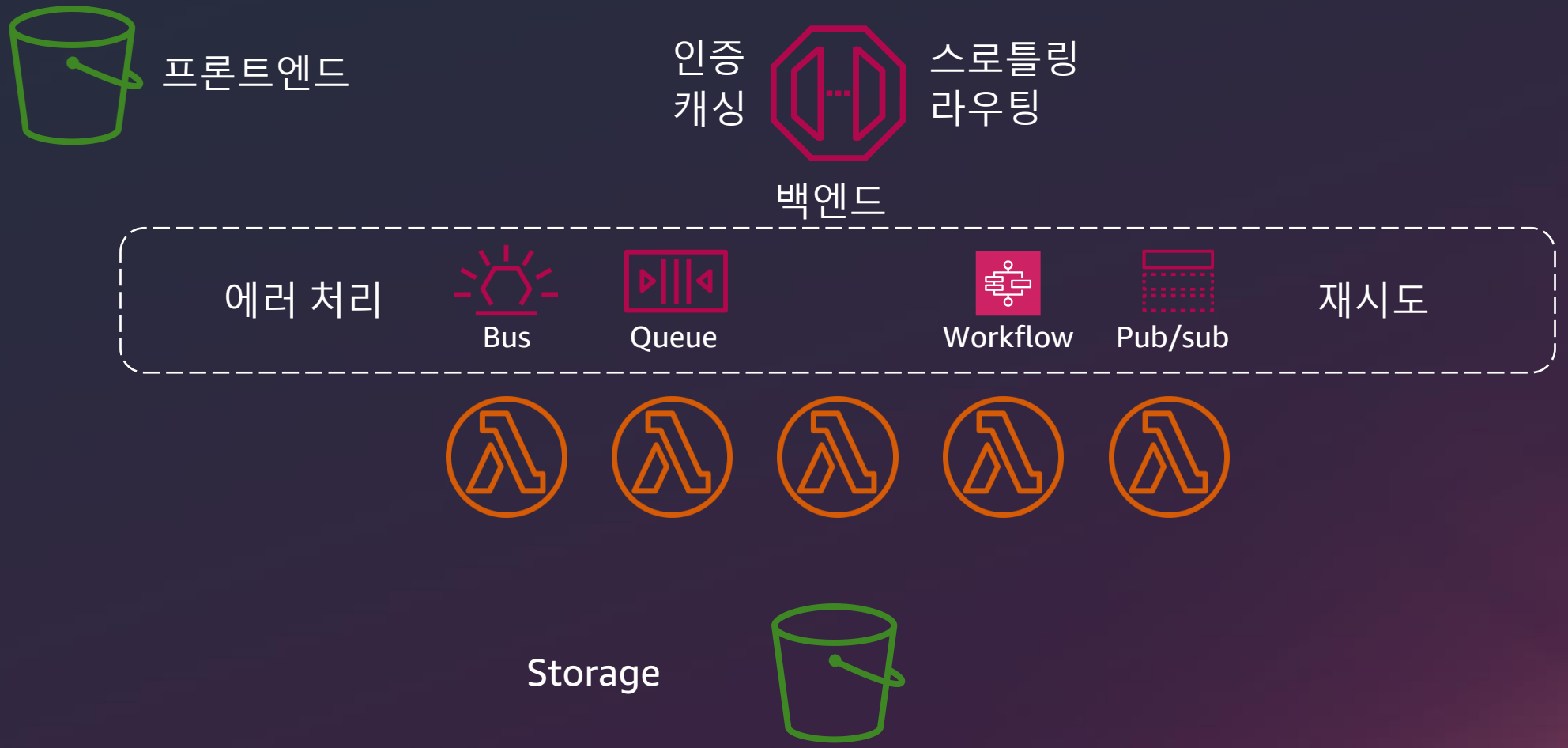
인증
캐싱
스로틀링
라우팅

백엔드
에러 처리
재시도
프론트엔드

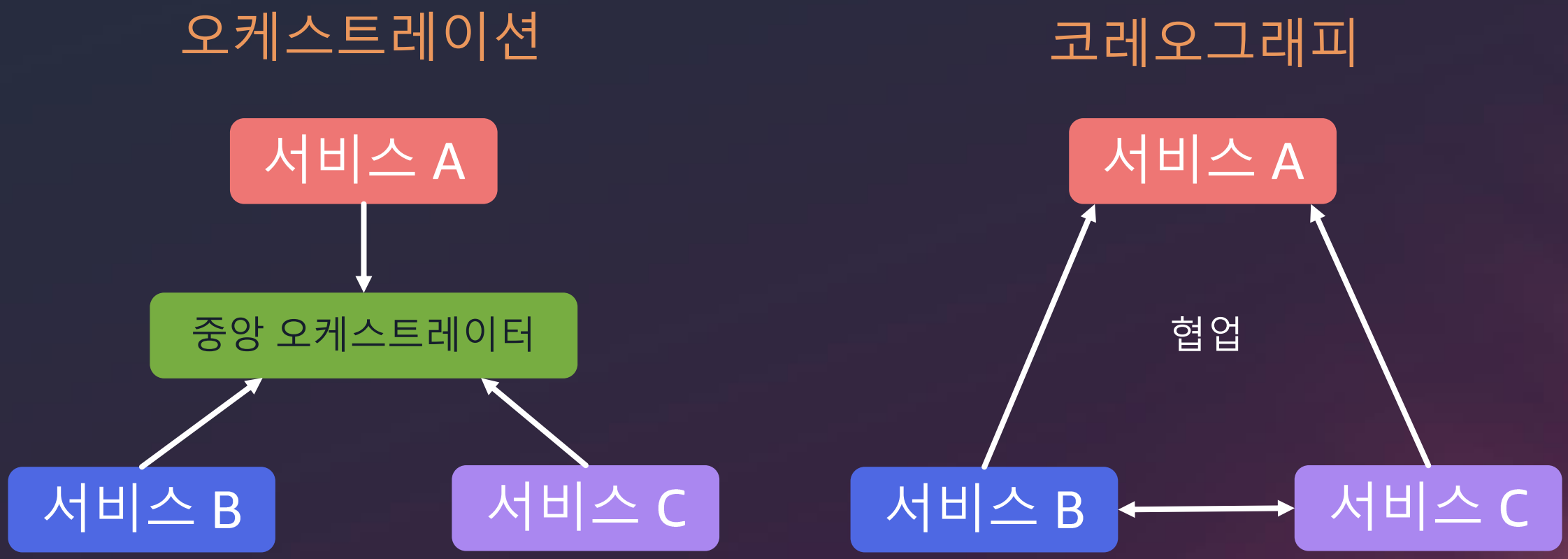
스토리지

마이그레이션 : Re-architect

매니지드 서비스 활용



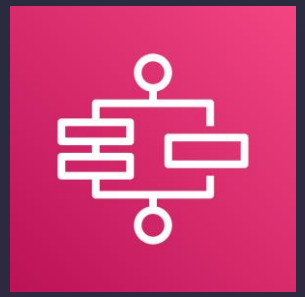
오케스트레이션 와 코레오그래피



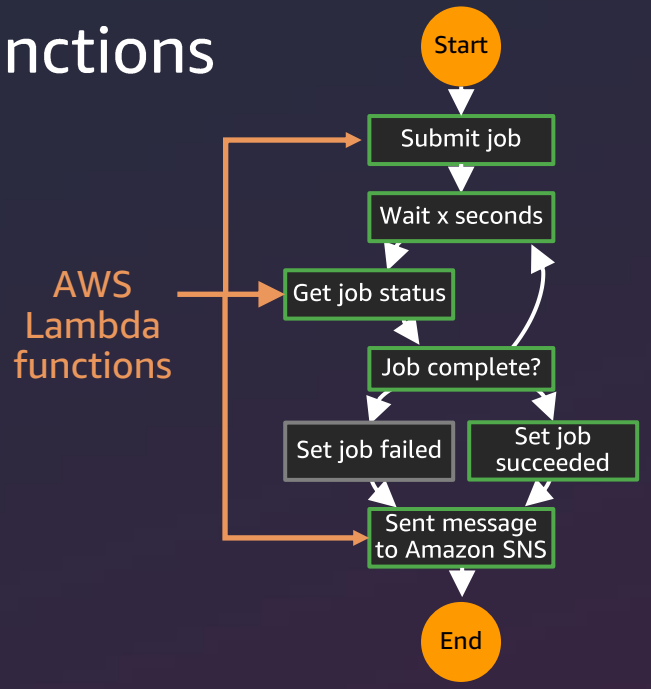
활용할 수 있는 서버리스 서비스

오케스트레이션

AWS Step Functions

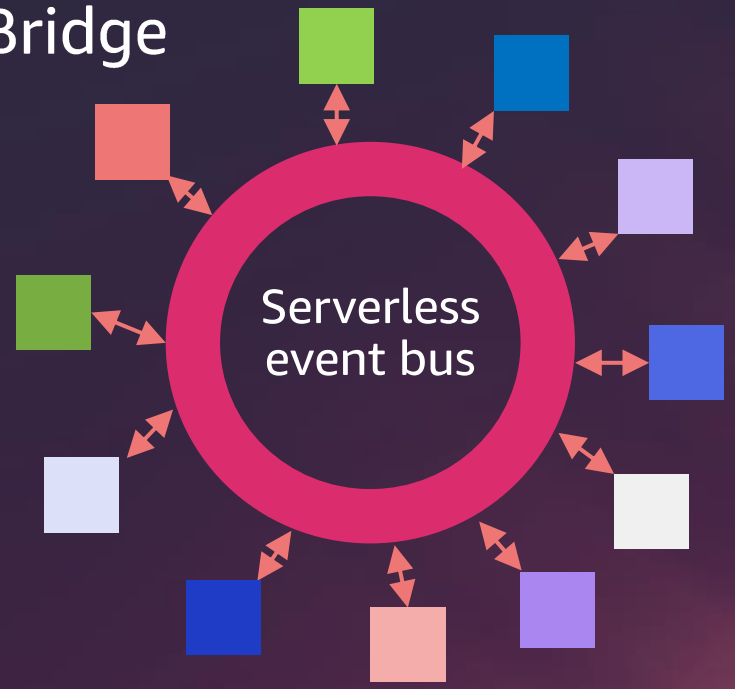


AWS Lambda functions



코레오그래피

Amazon EventBridge



Service-full 서버리스 : 베스트 프랙티스

- 최대한 서비스간 통합 기능 사용
- 코딩 = 유지보수, 최대한 설정/구성으로 대체
- AWS Lambda 는 단순 데이터 전송이 아닌 변환으로 사용
- 단일 모놀리식 서비스 및 AWS Lambda 함수 지양
- AWS Step Function 을 사용한 워크플로우 오케스트레이션
- Amazon EventBridge 를 사용한 이벤트 코레오그래피

AWS Lambda

AWS Lambda 실행 모델

동기

요청/응답

Application Load Balancer

Amazon API Gateway

AWS Lambda function URL

request event

response

AWS Lambda function

비동기

이벤트

Amazon EventBridge

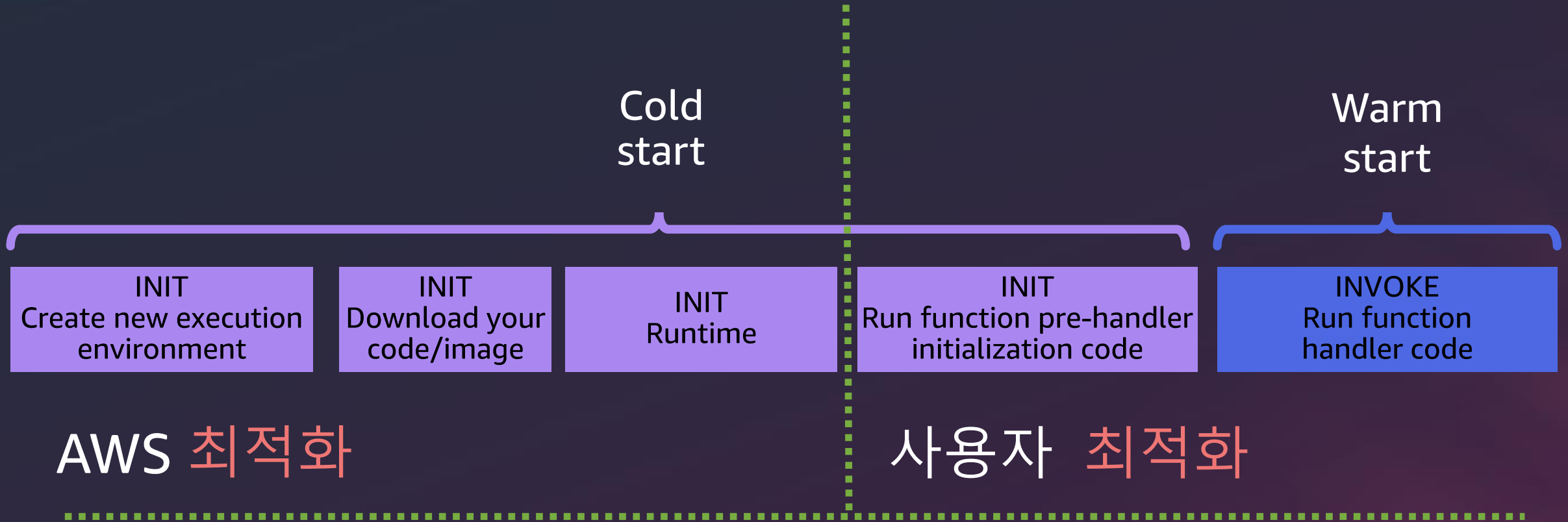
Amazon S3

event

event queue

AWS Lambda function

AWS Lambda 초기화 과정 및 최적화 포인트



AWS Lambda Cold Start

AWS Lambda Cold Start 가 발생하는 경우:

- 사용자
  - 스케일 아웃
  - 최초 프로비전드 동시성구성
  - AWS Lambda 함수 코드/설정 업데이트
- AWS
  - 실행 환경 새로 고침
  - 리소스 장애
  - 가용 영역 간 요청 리밸런싱

얼마나 오래 걸릴까?

- 100 ms 이하 ~ 1 초 이상
- 프로덕션 환경에서의 빈도는 1% 이하

pre-handler INIT 코드

```
Import sdk
Import http-lib
Import cheese-sandwich

Pre-handler-secret-getter()
Pre-handler-db-connect()

Function myhandler(event, context) {
....
```

Pre-handler INIT 코드 : 베스트 프랙티스

불필요한 로딩 지양

- 필요한 것만 Import
- 필요한 라이브러리/SDK 모듈만 사용
- Minify/uglify 를 통한 코드 최소화
- 배포 패키지 크기 최소화
- “모놀리식” 함수 지양

라이브러리 지연 초기화(Lazy-initialize)

- 단일 AWS Lambda 에서 복수 개의 함수를 사용하는 경우에 유용

커넥션 연결

- 커넥션의 재연결 로직은 핸들러 함수에 포함
- AWS SDK 의 Keep-alive 설정

실행 환경 재사용에 따른 상태 처리

- 후속 호출을 위한 데이터 보관
- 민감한 데이터는 보관 X

프로비전드 동시성

- 코드 변경 없이, Lambda 함수 단위로 적용

디펜던시 최적화

워크로드에 필요한 라이브러리만 사용

```
// const AWS = require('aws-sdk’)
const DynamoDB = require('aws-sdk/clients/dynamodb’) // 125ms 절약


// const AWSXRay = require('aws-xray-sdk’)
const AWSXRay = require('aws-xray-sdk-core’) // 5ms 절약


// const AWS = AWSXRay.captureAWS(require('aws-sdk’))
const dynamodb = new DynamoDB.DocumentClient()
AWSXRay.captureAWSClient(dynamodb.service) // 140ms 절약
```

라이브러리 지연 초기화(Lazy-initialize)

```
import boto3
S3_client = None
ddb_client = None

def get_objects_handler(event, context):
    if not s3_client:
        s3_client = boto3.client("s3")
    # business logic


def get_items_handler(event, context):
    if not ddb_client:
        ddb_client = boto3.client(”dynamodb")
    # business logic
```

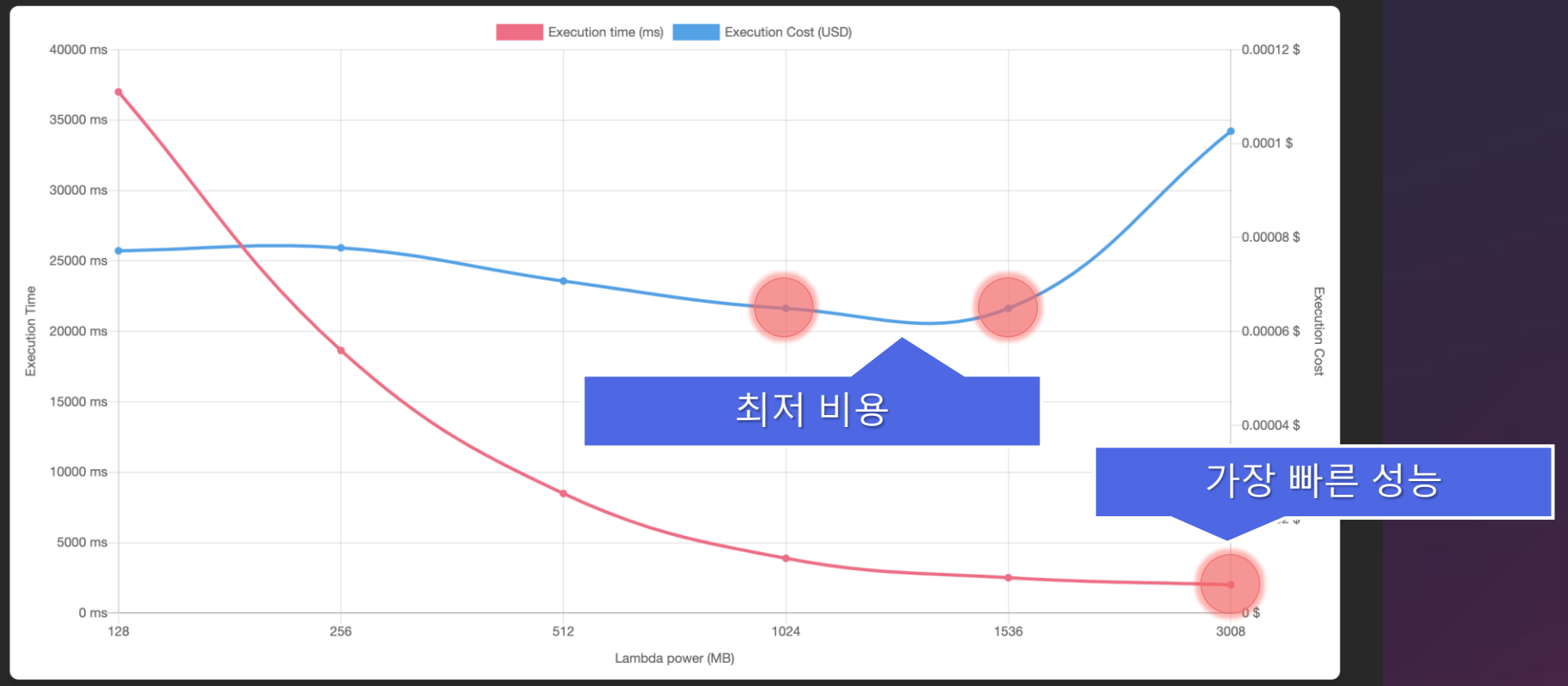
AWS Lambda 함수 메모리 = “파워”

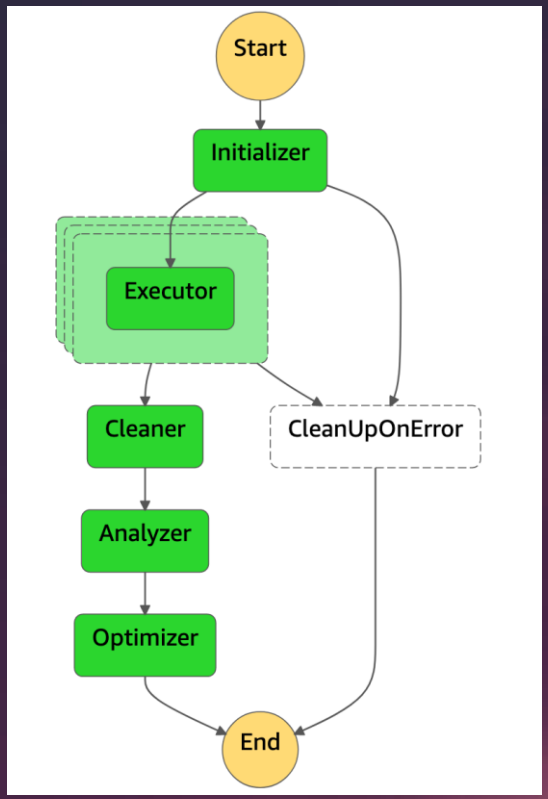
AWS Lambda 는 메모리 할당만 선택 가능
  128 MB ~ 10 GB 사이 (1 MB 단위)

AWS Lambda 는 메모리 사이즈에 비례하여 자원 할당:
  CPU 파워
  네트워크 대역폭

코드가 메모리/CPU/네트워크에 의한 성능 저하가 있다면 메모리를 추가 할당을 통해 성능 및 비용 절감 가능

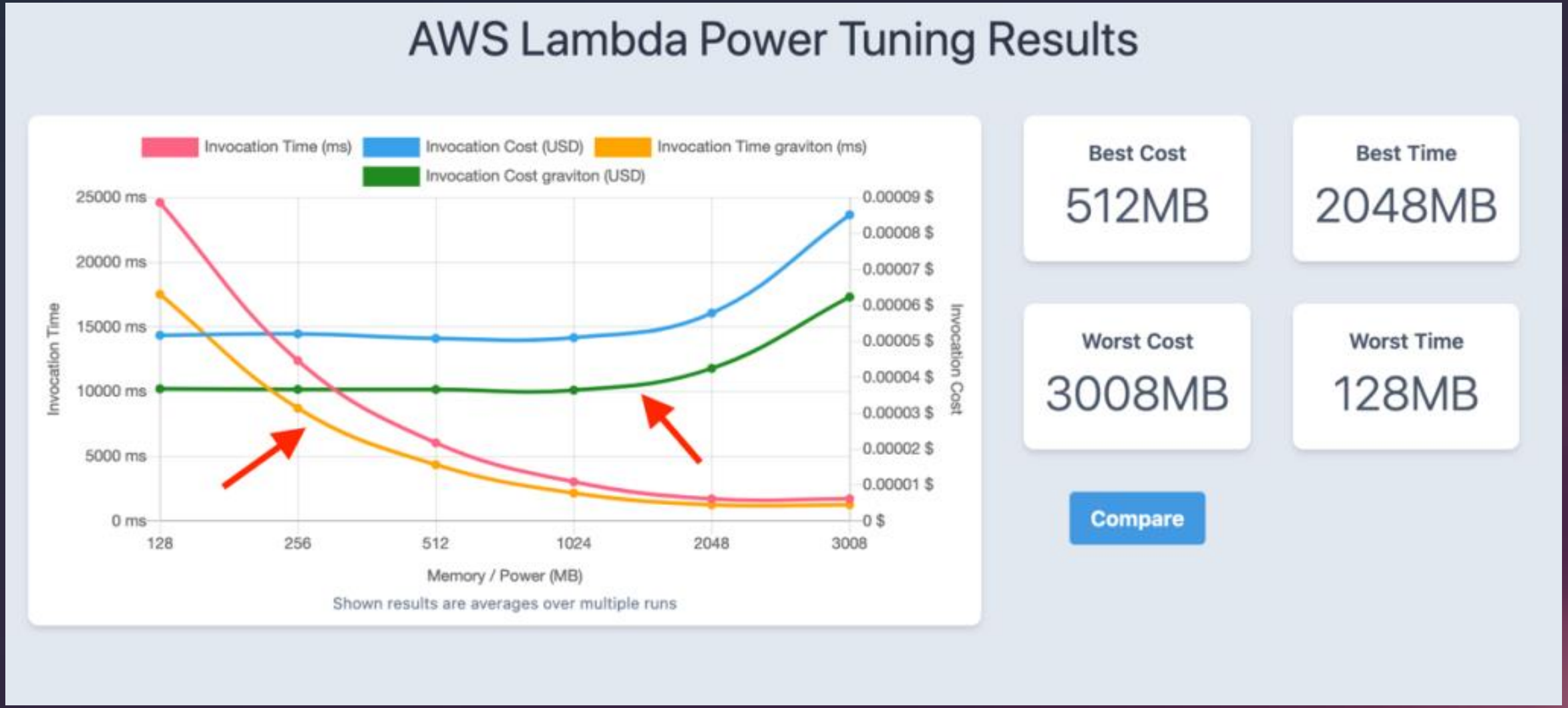
AWS Lambda Power Tuning





https://github.com/alexcasalboni/aws-lambda-power-tuning

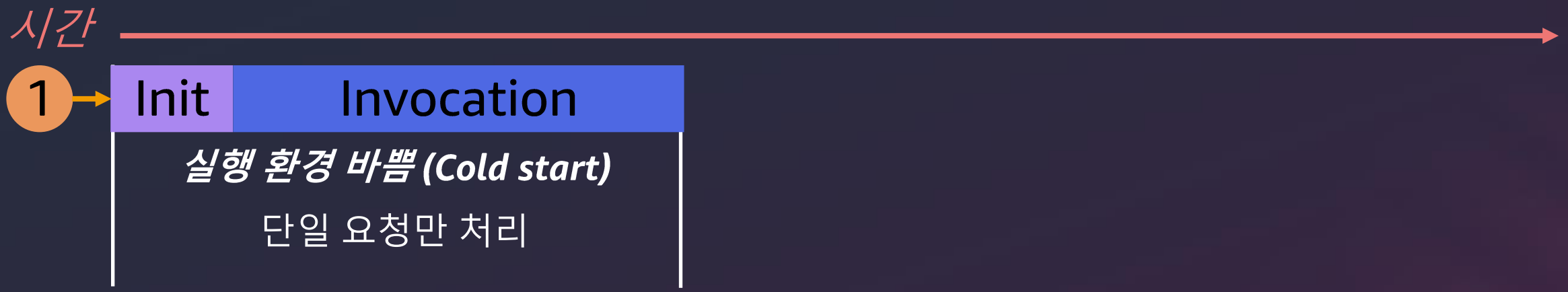
AWS Lambda function on Graviton2



https://aws.amazon.com/blogs/compute/optimizing-your-aws-lambda-costs-part-1/

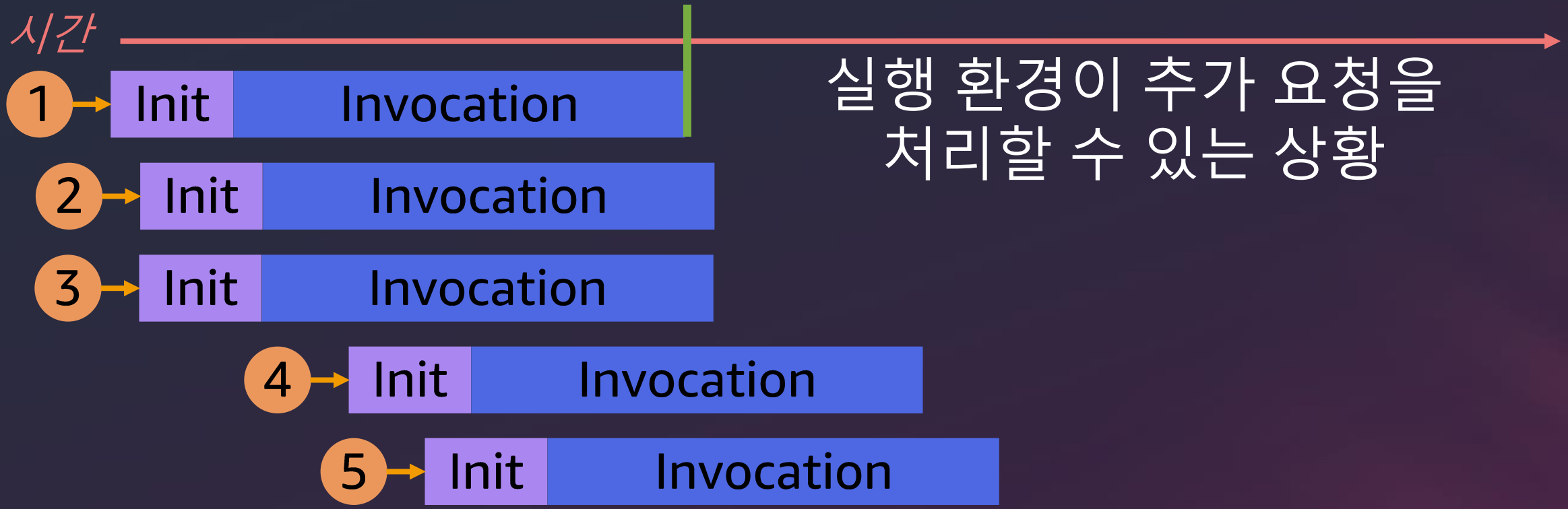
AWS Lambda 동시성 이해하기

AWS Lambda 동시성 이해하기

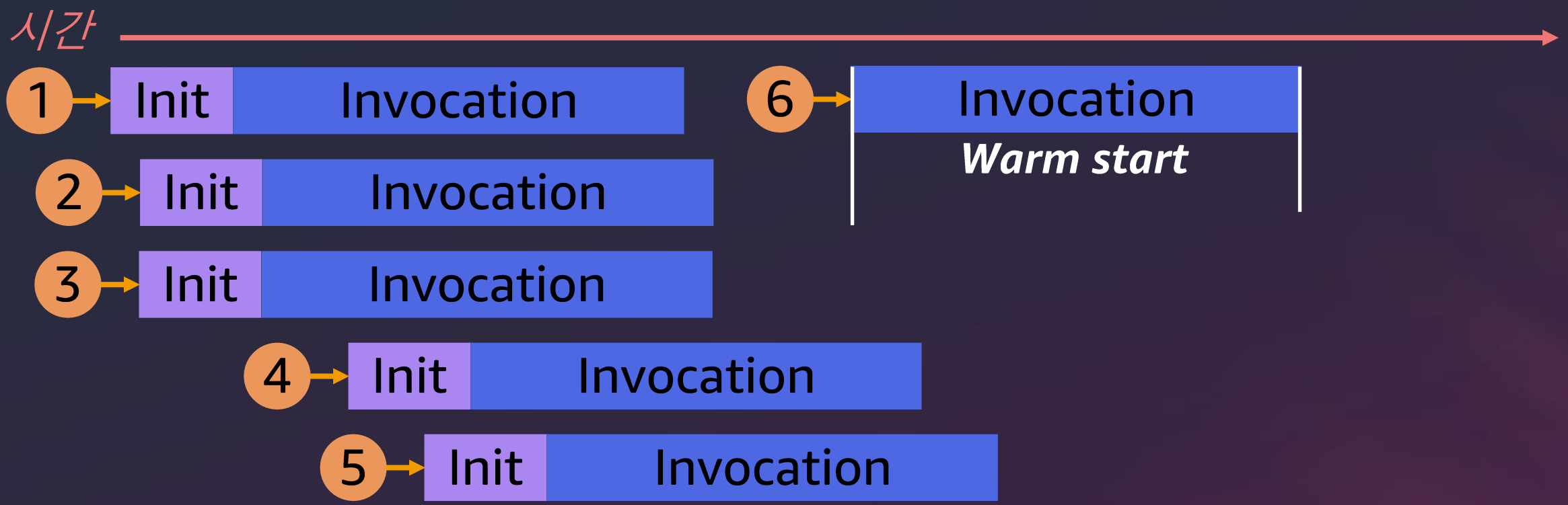


AWS Lambda 동시성 이해하기

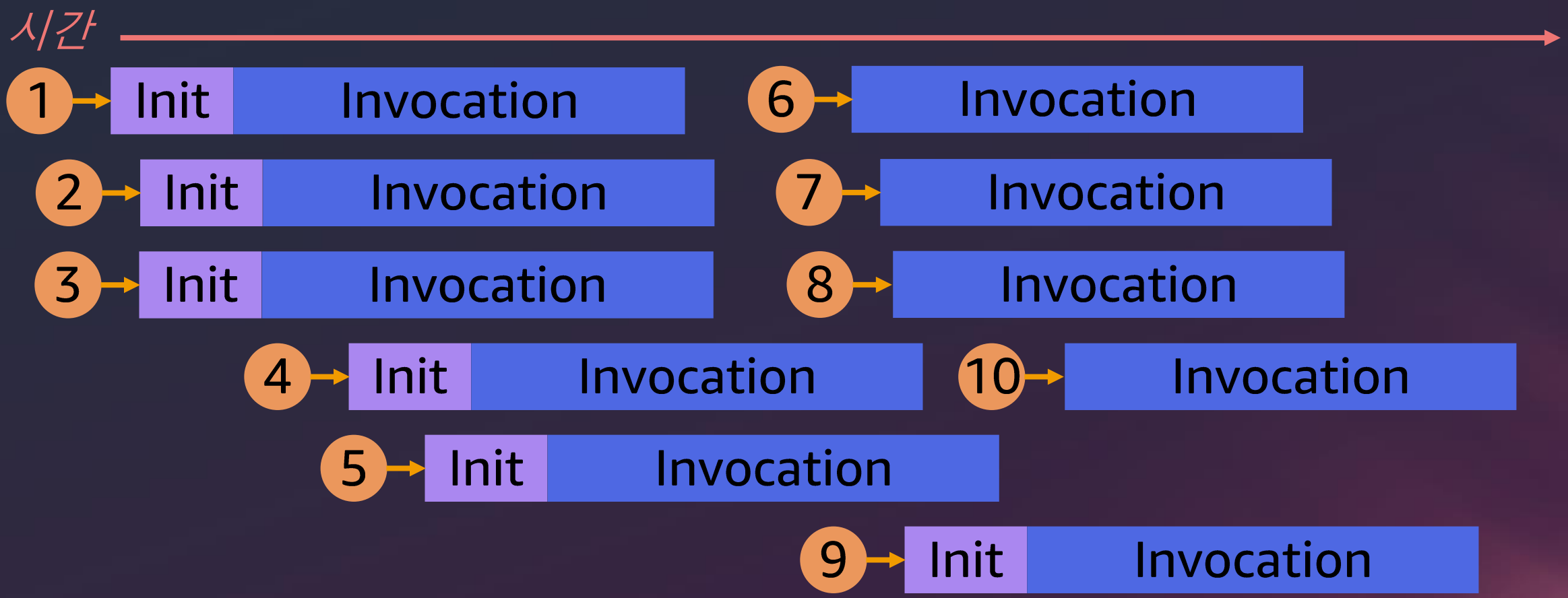
AWS Lambda 동시성 이해하기



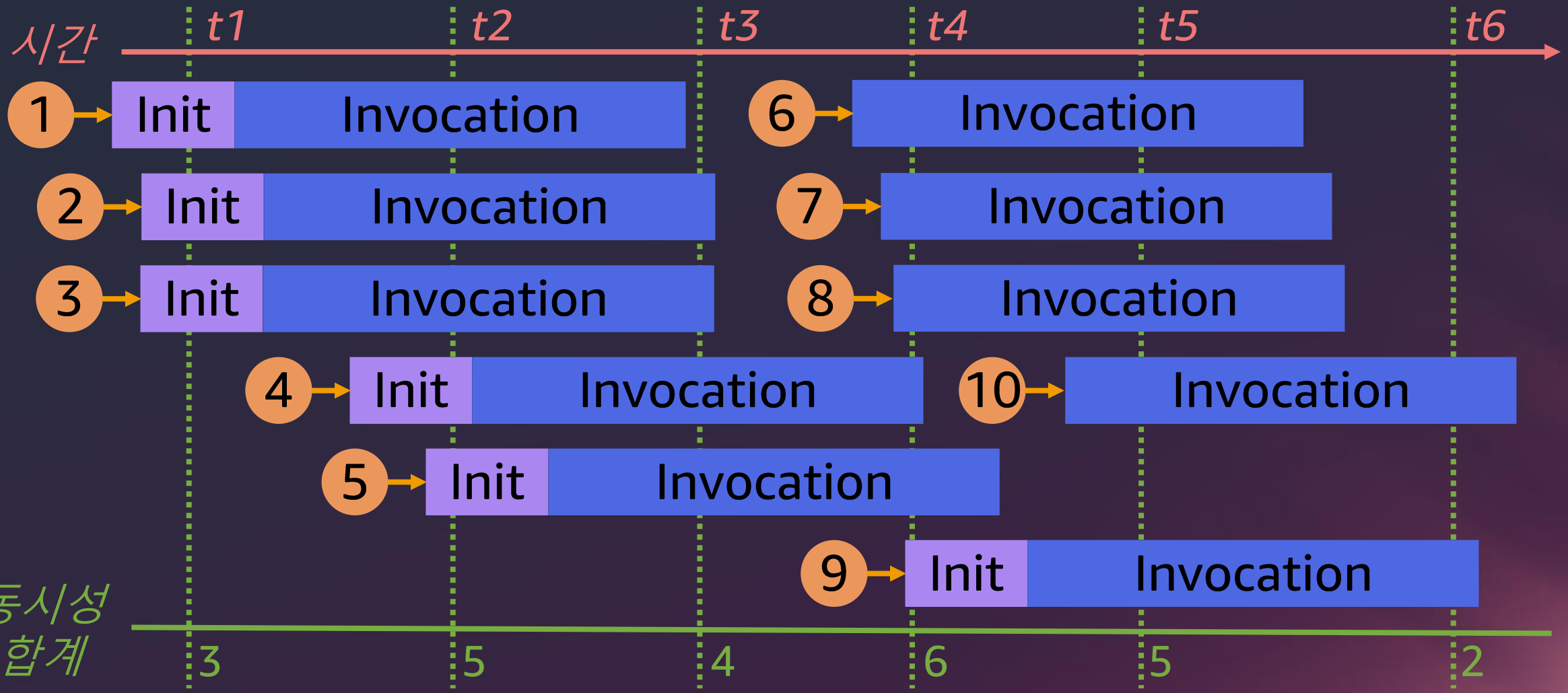
AWS Lambda 동시성 이해하기



AWS Lambda 동시성 이해하기



AWS Lambda 동시성 이해하기

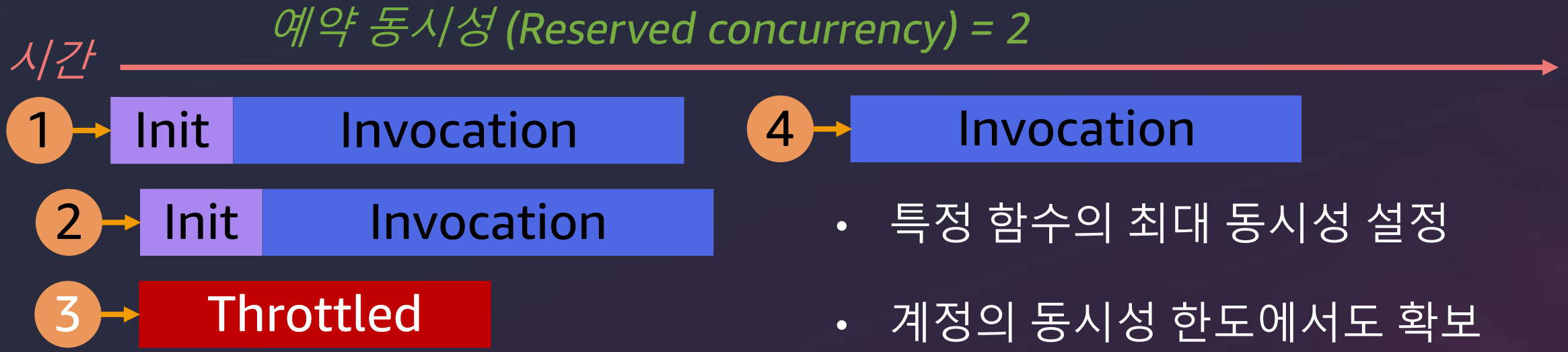


초당 트랜잭션 처리량 이해하기

초당 트랜잭션 처리량 = 10

초당 트랜잭션 처리량 = **20**

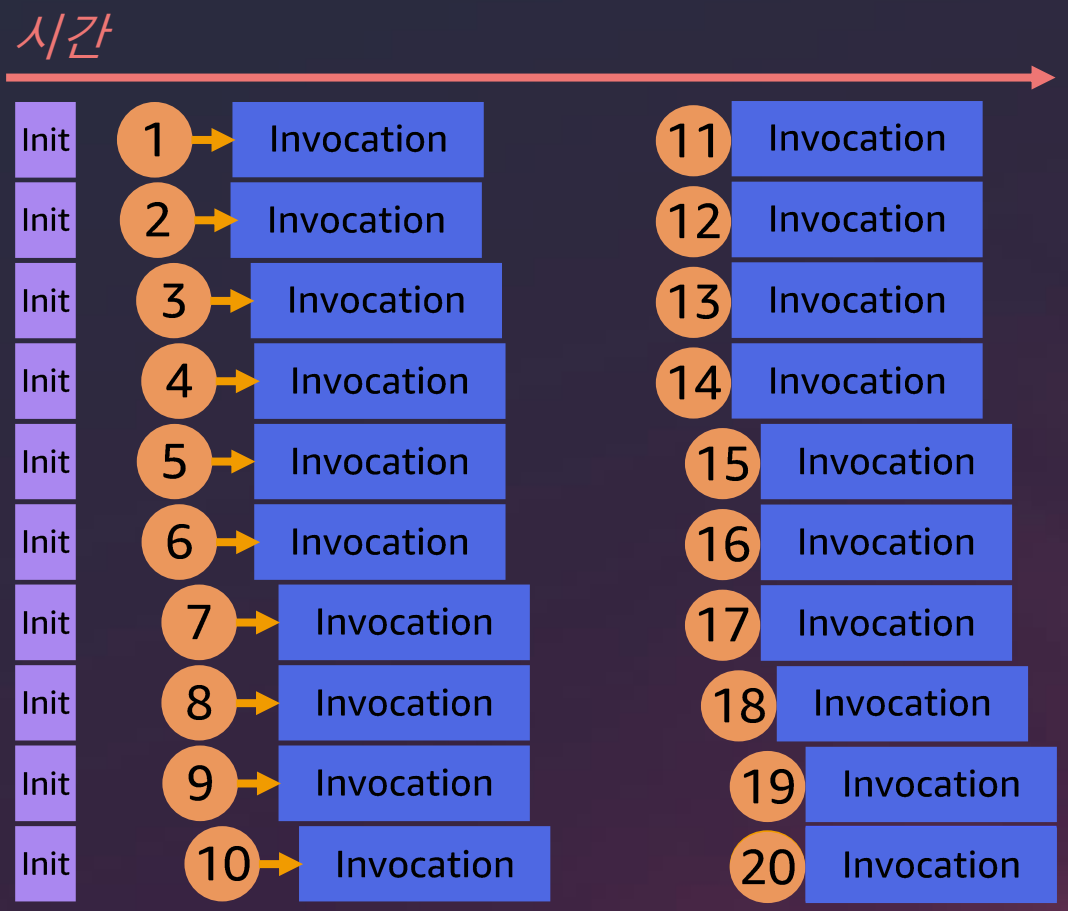
예약 동시성 이해 하기



- 특정 함수의 최대 동시성 설정
- 계정의 동시성 한도에서도 확보
- 다운 스트림 리소스 보호를 위해 활용 가능
- 예약 동시성을 0 으로 설정하여 비활성화 가능

프로비전드 동시성 이해 하기

- 최소 실행 환경 수 확보
- Cold-start 의 영향을 줄이기 위한 사전 예열
- 프로비전드 동시성 이상의 요청들은 표준 동시성 사용
- Application AutoScaling을 통해 자동 확장 가능



AWS Lambda : 베스트 프랙티스

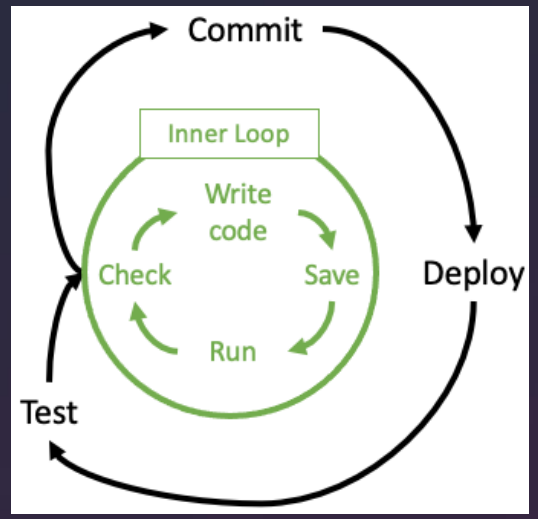
- AWS Lambda 호출 모델 이해
- 콜드 스타트 최적화
  - 불필요한 경우에는 로딩 X , 공유 라이브러리에 대한 지연 초기화(Lazy-initialize)
  - 연결 설정
  - 실행 환경 재사용에 따른 상태 공유
- ARM/AWS Graviton2 도입 검토
- 추가 메모리 = 추가 CPU 및 I/O (메모리 할당량에 비례)
- 예약 동시성 및 프로비전드 동시성 이해

프로토타이핑에서 프로덕션까지

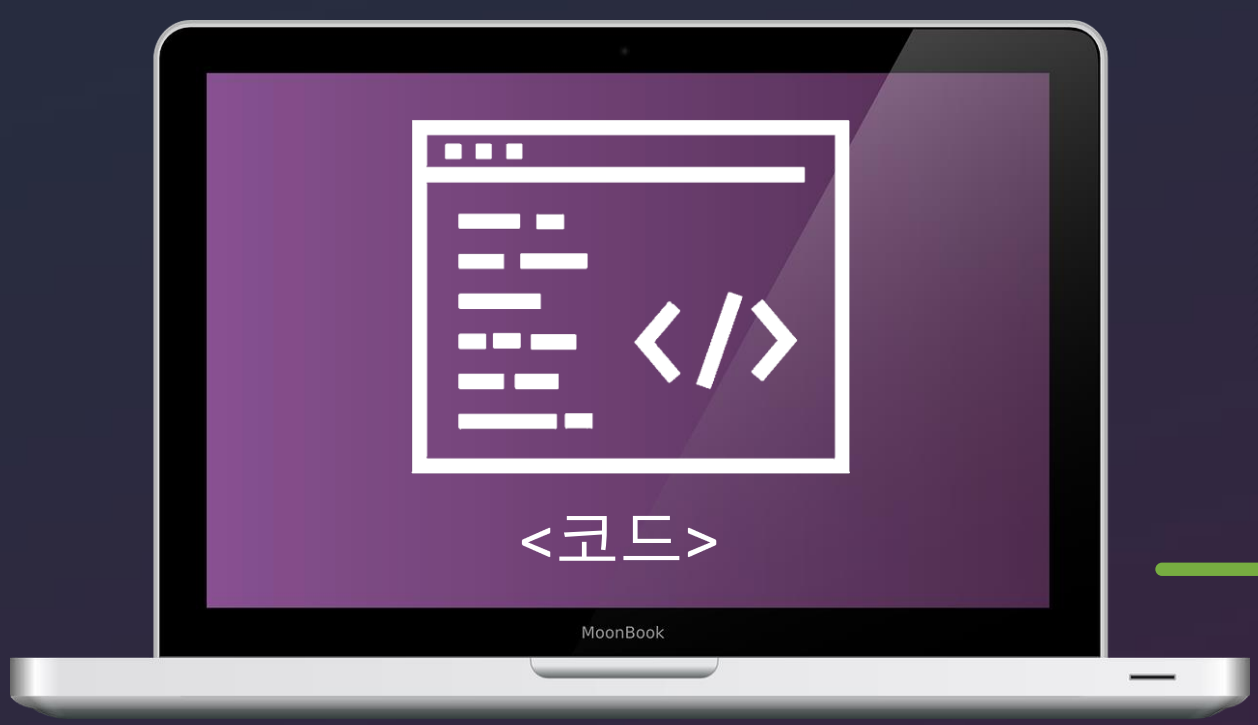
지금까지의 개발자들은...

개발자들은 일반적으로 메인 브랜치에 코드를 커밋하기 전까지
아래의 워크플로우를 따릅니다.

1. 코드 작성
2. 코드 저장
3. 코드 실행
4. 결과 확인



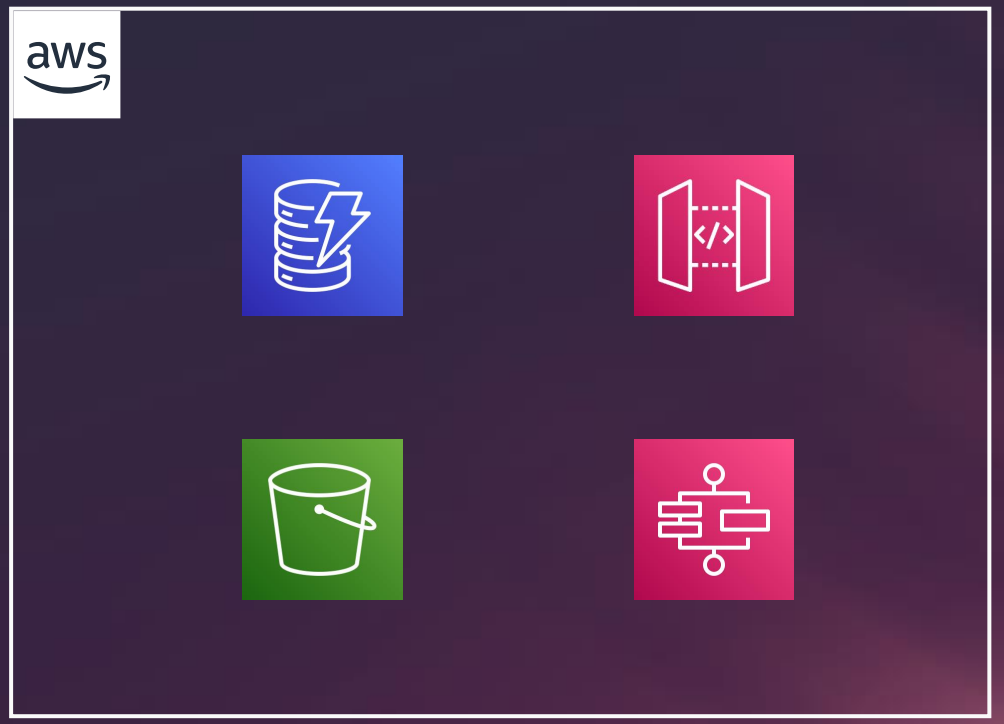
클라우드 네이티브 애플리케이션은 다릅니다.



<테스트>

최소한의 에뮬레이션

실제 클라우드 환경에서 최대한 빨리 코드를 실행



- AWS Lambda 함수는 로컬에서 개발/테스트
- 실제 서비스와 커뮤니케이션
- 통합 테스트는 클라우드에서 실행
- 테스트의 인프라스트럭처 정확도 향상

Serverless Application Model

AWS SAM 구성

AWS SAM transform

서버리스 리소스 및
이벤트 소스 매핑을
표현하는 템플릿 제공

서버리스
애플리케이션에 대한
Infrastructure as a code
(IaC) 제공

AWS SAM CLI

서버리스
애플리케이션의 로컬
및 클라우드 개발,
디버깅, 빌드, 패키징,
파이프라인 생성을
위한 툴링 제공

AWS SAM Accelerate

sam build

코드의 변경된
부분만 빌드

- 병렬 빌드
- 증분 빌드
- 임시 레이어

sam sync

코드/워크플로우/API
변경사항을 클라우드와
신속하게 동기화

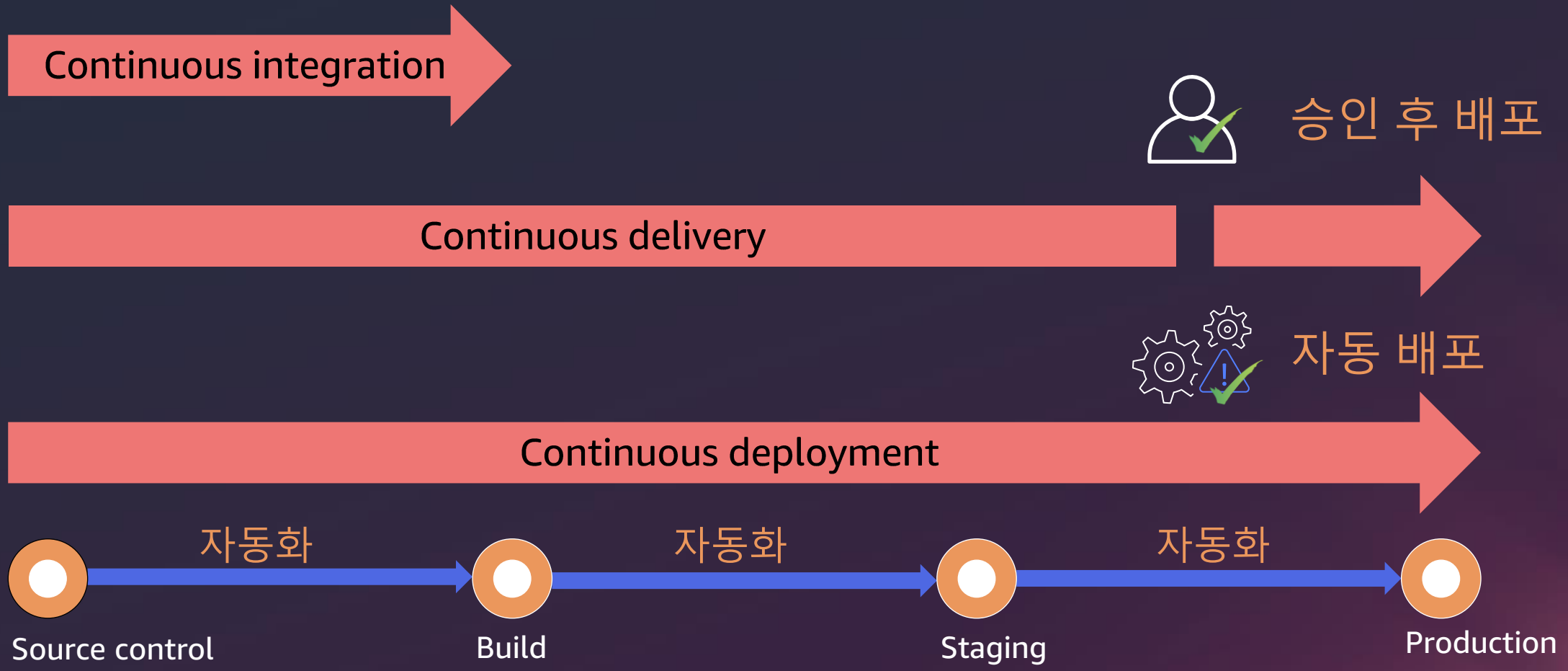
- 변경된 파일 모니터링

sam logs

로컬 터미널에서
애플리케이션 로그 추적

- 로그 필터
- 추적

CI/CD 파이프라인

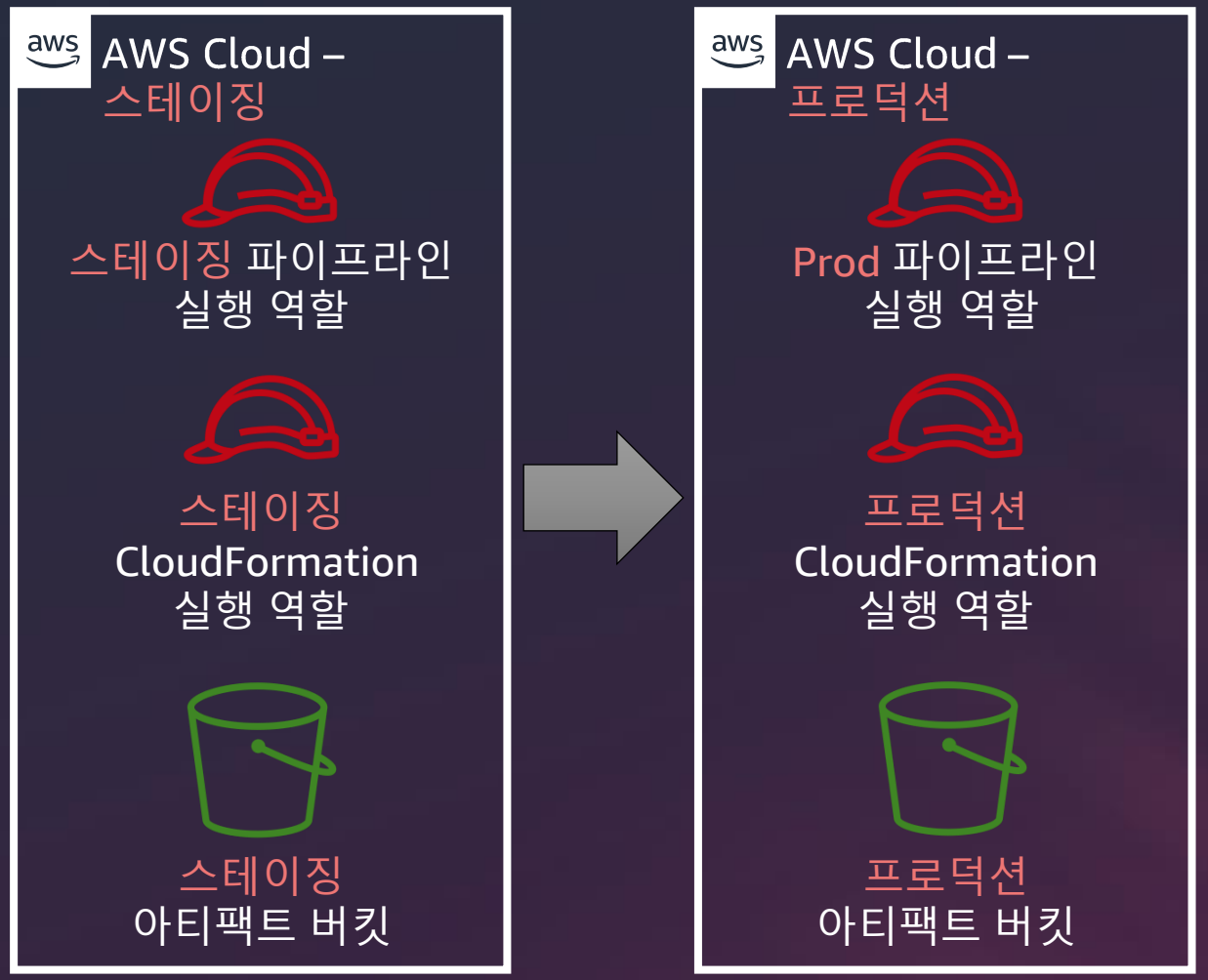


AWS SAM 파이프라인

```
sam pipeline init --bootstrap
```

코드 저장소의 애플리케이션 아티팩트를 AWS 배포에 필요한 AWS 리소스 및 권한 생성

IAM 혹은 OIDC 인증



GitHub

GitLab

AWS CodePipeline

Jenkins

로그의 CloudWatch 지표화

Amazon CloudWatch embedded metric format 활용

Event payload

```
message = {
  "PriceInCart": 100,
  "QuantityInCart": 2,
  "ProductId": "a23390f3",
  "CategoryId": "bca4cec1",
  "Environment": "prod",
  "UserId": "31ba3930",
  "CartId": "58dd189f",
"LogLevel": "INFO",
  "Timestamp": "2019-12-11 12:44:40.300473",
  "Message": "Added 2 items 'a23390f3' to
cart '58dd189f'"
}
```

Log entry

```
[...]
  "_aws": {
  "functionVersion": "$LATEST",
    "Timestamp": 1576064416496,
    "CloudWatchMetrics": [{
      "Namespace": "ecommerce-cart",
      "Dimensions": [
        ["Environment", "CategoryId"]
      ],
      "Metrics": [
        {"Name": "PriceInCart", "Unit": "None"},
        {"Name": "QuantityInCart", "Unit": "None"}
      ]}]},
 "Environment": "prod",
 "CategoryId": "bca4cec1"
 "PriceInCart": 100,
 "QuantityInCart": "2"
}
```

로그의 CloudWatch 지표화

Amazon CloudWatch embedded metric format 활용

Event payload

```
message = {
  "PriceInCart": 100,
  "QuantityInCart": 2,
  "ProductId":
  "CategoryId"
  "Environment
  "UserId": "3
  "CartId": "5
"LogLevel": "I
  "Timestamp":
  "Message": "
cart '58dd189f
}
```

Log entry

```
[...]
  "_aws": {
  "functionVersion": "$LATEST",
  "                  416496,
  "CloudWatchMetrics": [{
    "Namespace": "eCommerce-cart",
    "Dimensions": [
      ["Environment", "CategoryId"]
    ],
    "Metrics": [
      {"Name": "PriceInCart", "Unit": "None"},
      {"Name": "QuantityInCart", "Unit": "None"}
    ]}]},
  "Environment": "prod",
  "CategoryId": "bca4cec1"
  "PriceInCart": 100,
  "QuantityInCart": "2"
}
```

오픈소스 클라이언트 라이브러리 제공

- Node.js
- Python
- Java
- C#

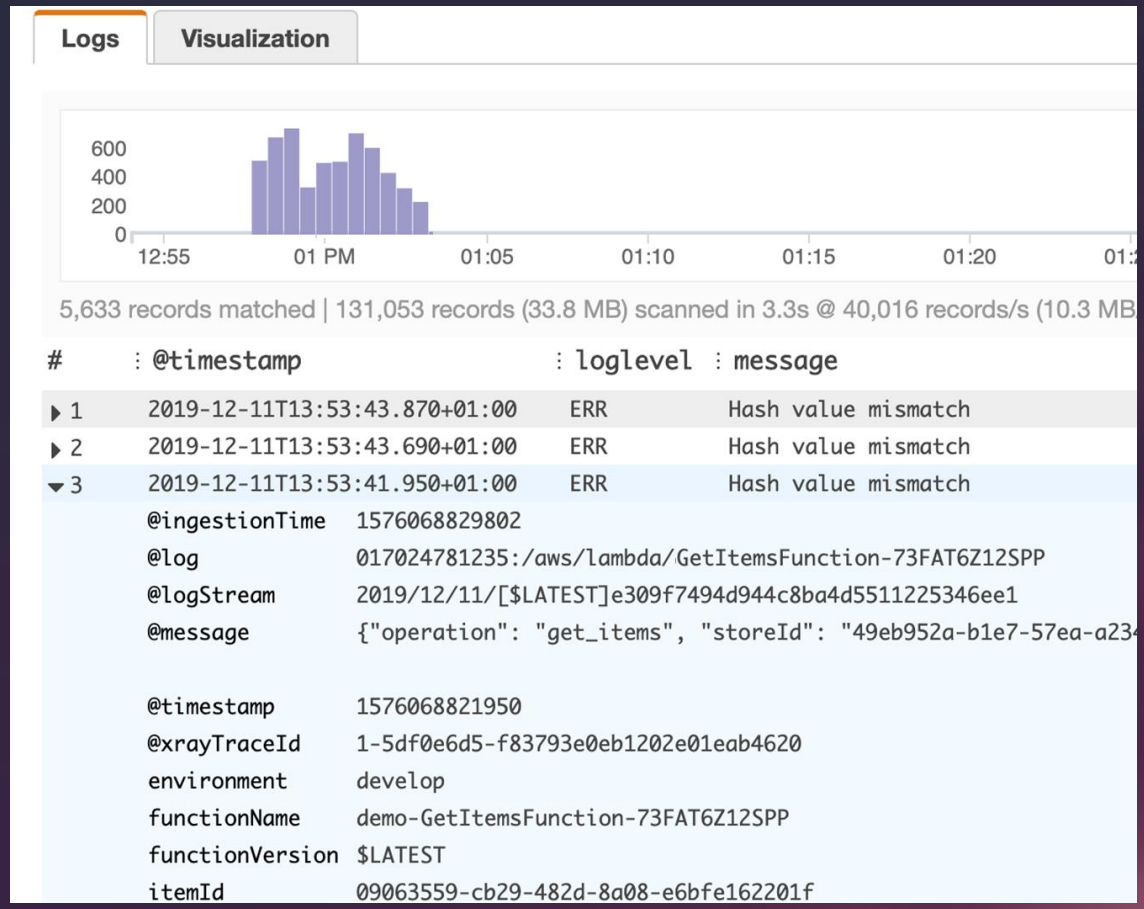
https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/CloudWatch_Embedded_Metric_Format_Libraries.html

로그 쿼리

Amazon CloudWatch Logs Insights

- 상호작용형 로그 검색 및 분석
- 구조화된 로그 분석
- 유연한 쿼리 언어 제공
- 최대 20개의 로그 그룹 쿼리
- 쿼리문 저장

```
fields Timestamp, LogLevel, Message
| filter LogLevel == "ERR"
| sort @timestamp desc
| limit 10
```



https://serverlessland.com/snippets?type=CloudWatch+Logs+Insights

AWS Lambda Powertools

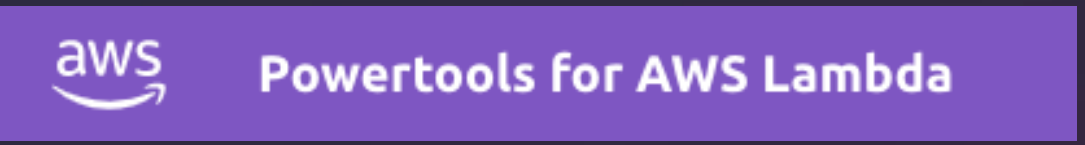
Logging: 구조화된 JSON으로 출력

Tracing: AWS X-Ray 로 추적 전송

Metrics: Embedded metric 포맷의 커스텀 메트릭

Utilities: 파라미터, 멱등성, SQS 처리 등

(언어별로 상이)



Python Java Typescript/JavaScript .NET

프로토타이핑에서 프로덕션까지 : 베스트 프랙티스

- 로컬에서 서비스 에뮬레이션 지양
- 비즈니스 로직은 로컬에서 나머지 테스트는 전부 클라우드에서
- AWS SAM 을 활용한 코드형 인프라 관리
- AWS SAM 을 활용하여 로컬과 클라우드 사이의 정합성 유지
- AWS Lambda Powertools 사용 해보기
- 로그로부터 애플리케이션 메트릭 추출하여 모니터링에 활용하기

앱 현대화 리소스 허브

AWS가 제공하는 앱 현대화에 관한 다양한 자료들을 통해 더욱 심층적으로 학습해 보세요!

현대화 여정을 계획하고 실행하는데 유용한 AWS의 다양한 자료들을
앱 현대화 리소스 허브에서 확인해 보세요.

- AWS기반 현대적 애플리케이션 구축 자료
- 클라우드 현대화의 비즈니스 가치
- 이벤트 드리븐 아키텍처 소개 및 가이드
- 풀스택 웹 앱 및 모바일 앱 개발 가이드
- 총 소유 비용 비교: 서버리스 기술과 서버 기반 기술 비교

https://bit.ly/modern-apps-aws

리소스 허브 방문하기

AWS 교육 및 자격증

AWS 스킬 빌더, 600개 이상의 무료 디지털 콘텐츠를 만나보세요!

30개 이상의 AWS 솔루션에 대한 디지털 셀프 학습 계획 및 Ramp-Up 가이드 등 다양한 학습 리소스를 제공하여 여러분에게 가장 필요한 클라우드 기술과 서비스에 집중해서 학습하실 수 있습니다.

- 여러분의 진도에 맞춰 원하는 목표를 달성하세요.
- 학습 계획에 따라 여러분의 지식과 기술을 발전시키세요.
- AWS 자격증으로 여러분의 클라우드 역량을 증명하시기 바랍니다.

https://aws.amazon.com/ko/training/digital/

클라우드 기술 습득하기

AWS 파트너와 참여

AWS 파트너와 함께 클라우드로의 여정을 가속화하고 AWS가 제공하는 모든 서비스를 최대한 활용해 보세요.

- **혁신** – 조직을 위한 혁신적이고 비용 효율적인 확장 가능한 클라우드 솔루션 및 기능으로 최첨단 기술 변화에 보조를 맞출 수 있습니다.
- **전문성** – 전략적이고 경험이 풍부한 전문가들이 귀사의 비즈니스 서장을 지원하는 획기적이고 적절하며 신뢰할 수 있는 솔루션을 제고합니다.
- **글로벌 범위** – 소프트웨어, 하드웨어, 및 서비스 전반에 걸쳐 신뢰할 수 있는 AWS 파트너의 글로벌 커뮤니티를 만나볼 수 있습니다.

Connect with an AWS Partner

적합한 AWS 파트너 찾아보기

AWS Innovate – 앱 현대화 특집에 참석해 주셔서 감사합니다.

저희가 준비한 강연, 어떻게 보셨나요?
더 나은 세미나를 위하여 설문을 꼭 작성해 주세요!


aws-korea-marketing@amazon.com


twitter.com/AWSKorea


facebook.com/amazonwebservices.ko

youtube.com/AWSKorea


linkedin.com/company/amazon-web-services


twitch.tv/aws

Thank you!