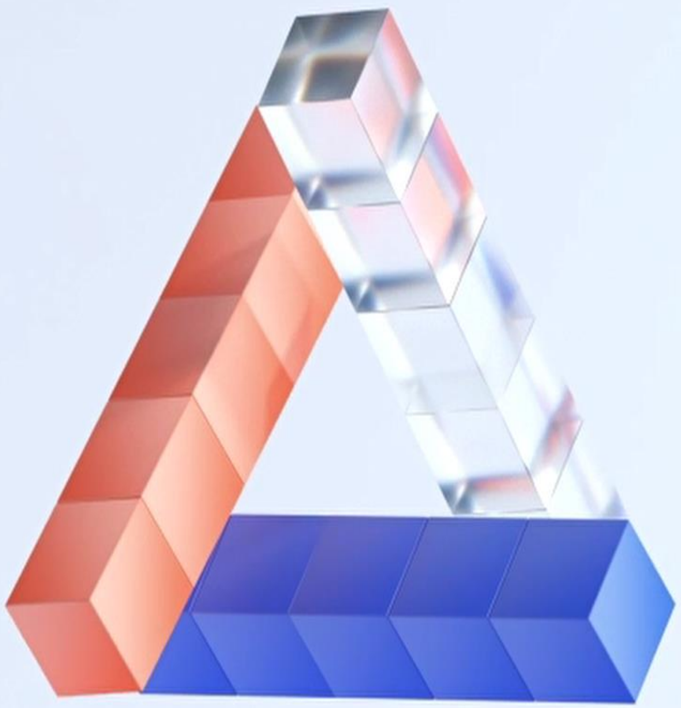
DEVOCEAN

TANGO 비용최적화 운영

EC2/RDS Auto Stop/Start

DEVOCEAN



1. TANGO on AWS

2. Cloud 비용 최적화 사례, Off time 최적화

   - EC2, RDS는 되고

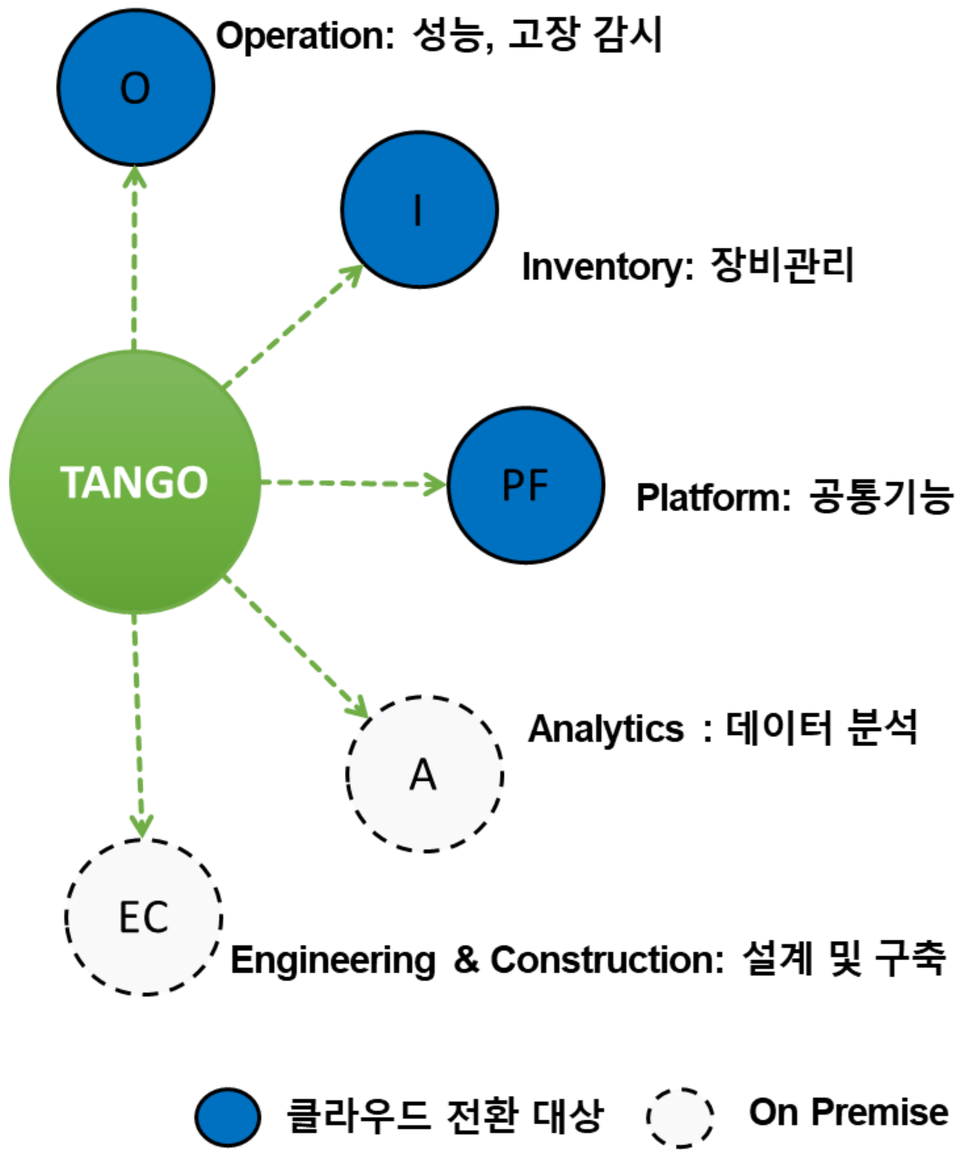
   - MSK, DMS는 안되고

3. Cloud 비용 최적화 사례, 그 외

4. Cloud 운영 방식 – 자동화/IaC 그리고 시각화

1. TANGO on AWS

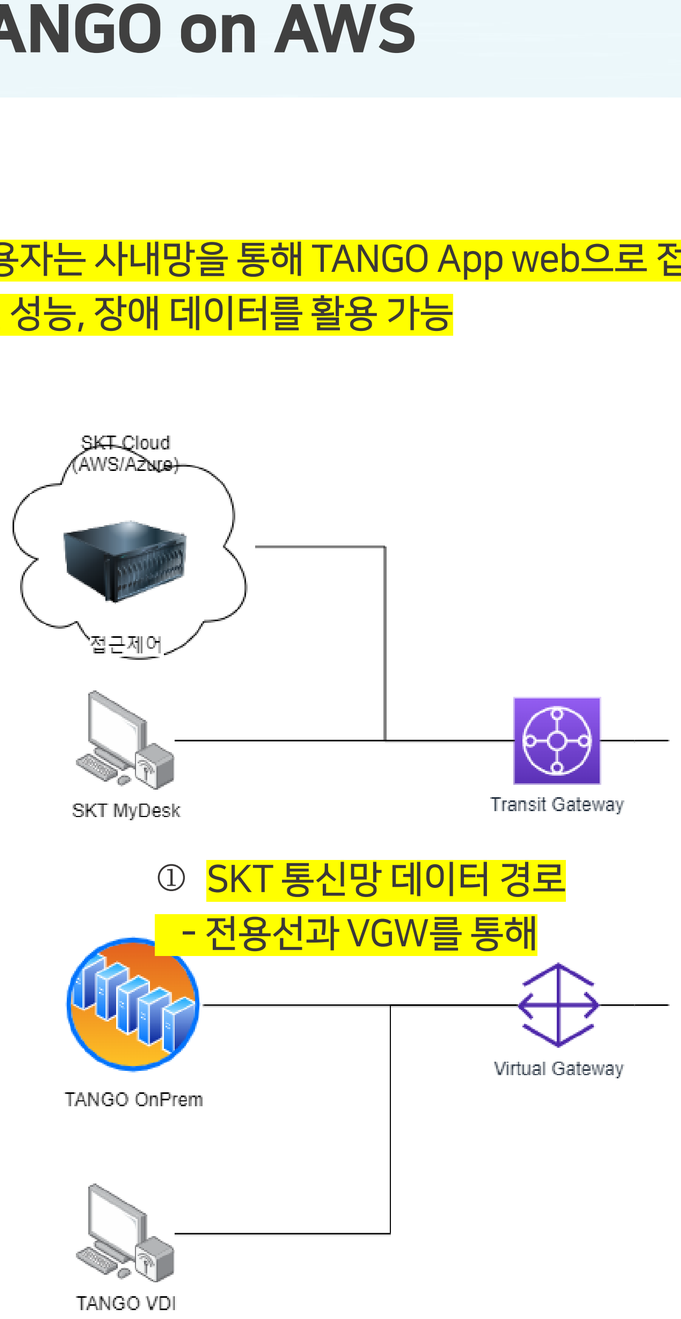
- **TANGO 란? (T-Advanced Network Generation OSS)**
  - SKT 망 관리를 지원해 주는 통합 시스템(OSS)
  - 주요기능: 장비 관리, 감시, 분석, 망 설계 구축
  - 구성: O, I, PF, A, EC의 5가지 기능으로 구성



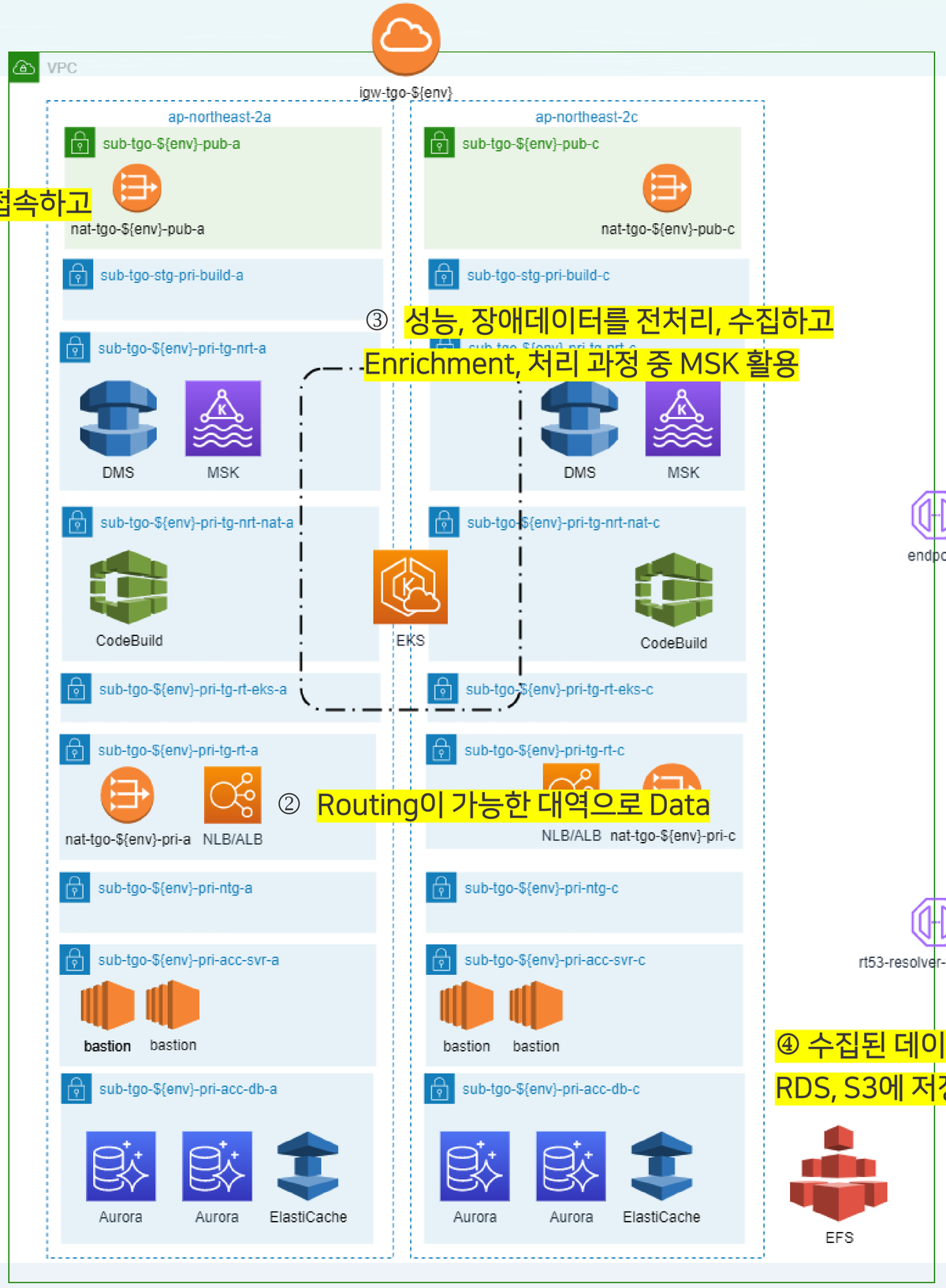
- **퍼블릭 클라우드 전환 추진 일정**
  - 오픈소스 DB 전환 방안 설계 (‘21년)
  - 5G 망 감시 시스템 (‘22년)
  - 3G, LTE, IP 망 감시시스템 (‘23년)
  - 장비관리 및 공통기능 (‘24년)

1. TANGO on AWS

⑤ 사용자는 사내망을 통해 TANGO App web으로 접속하고 저장된 성능, 장애 데이터를 활용 가능



⑥ 개발자는 bastion 서버, CodeCommit, SSM, CodeBuild, Lambda 등을 활용 배포와 운영관리



1. TANGO on AWS

1. TANGO on AWS

Dev/Stg/Prd 목적으로 AWS 계정을 구분하여 활용
계정별로 미사용 시간대에 리소스 최적화를 고민

Dev account

신규 기능 개발 및 테스트
필요 인프라 개별 생성

Stg account

Prd에 상응하는 인프라와
데이터 처리를 구성
EC2/RDS/EKS/MSK/S3 …
AWS 서비스 업그레이드 검증
AWS CI/CD 활용 개발 및 테스트

Prd account

24시간 운영 서비스
EC2/RDS/EKS/MSK
/Cloudwatch …
Stg 검증 이미지 기반 Prd배포

Stop/Start EC2, RDS

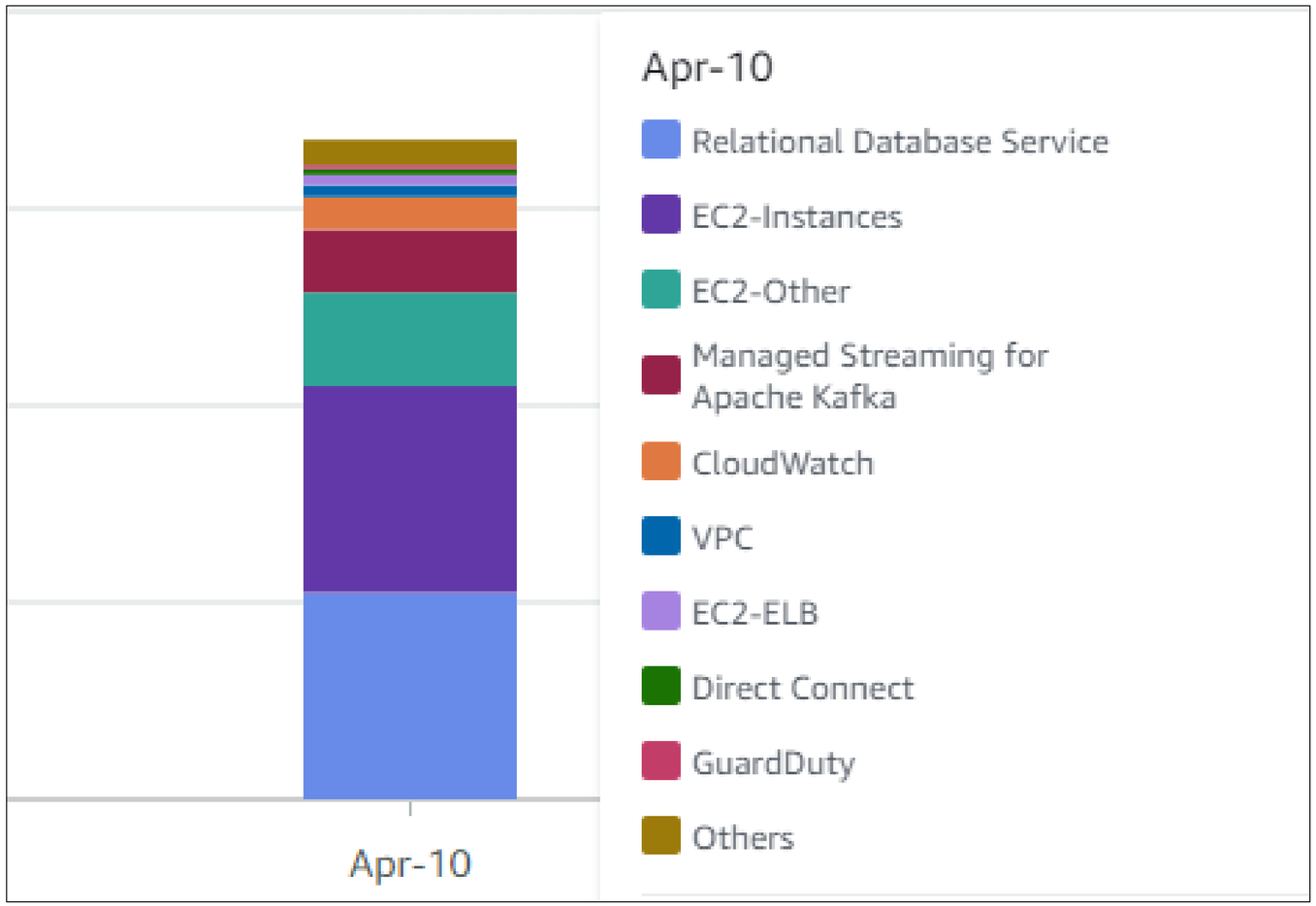
Modify MSK, DMS

EKS Cost 분석

EKS Down Scaler

2. Cloud 비용 최적화 사례, Off time 최적화

4/10 Stg 계정의 비용을 확인하면 RDS, EC2, MSK, CloudWatch 순으로 비용 발생



2. Cloud 비용 최적화 사례, Off time 최적화

Dev/Stg AWS 계정은 Off time이 존재하고 이 시간에 자동으로 EC2, RDS 리소스 Stop/Start 적용
- 평일 7~22시, 주말과 공휴일 미사용

- 개 요

1. Dev/Stg 계정에 비용 절감을 위해 미사용 시간에 RDS와 EC2 on/off 수행
2. AWS에서 제공해 주는 플렛폼 형태의 솔루션도 가능
   Instance Scheduler on AWS(https://docs.aws.amazon.com/solutions/latest/instance-scheduler-on-aws/architecture-overview.html)
3. 람다를 구성하고 일정 주기 마다 호출하여 tag 기반 rule에 의거 EC2와 RDS를 filter후 on/off를 수행하는 로직
4. Instance Scheduler on AWS는 dynamoDB를 활용하고 해당 lambda를 주기적으로 (최소 5분~최대 1시간) 호출
5. 플렛폼화 하여 여러 인스턴스를 다양한 주기로 관리하려고 하면 Instance Scheduler on AWS도 고려해 볼만함

2. Cloud 비용 최적화 사례, Off time 최적화

Dev/Stg AWS 계정은 Off time이 존재하고 이 시간에 자동으로 EC2, RDS 리소스 Stop/Start 적용

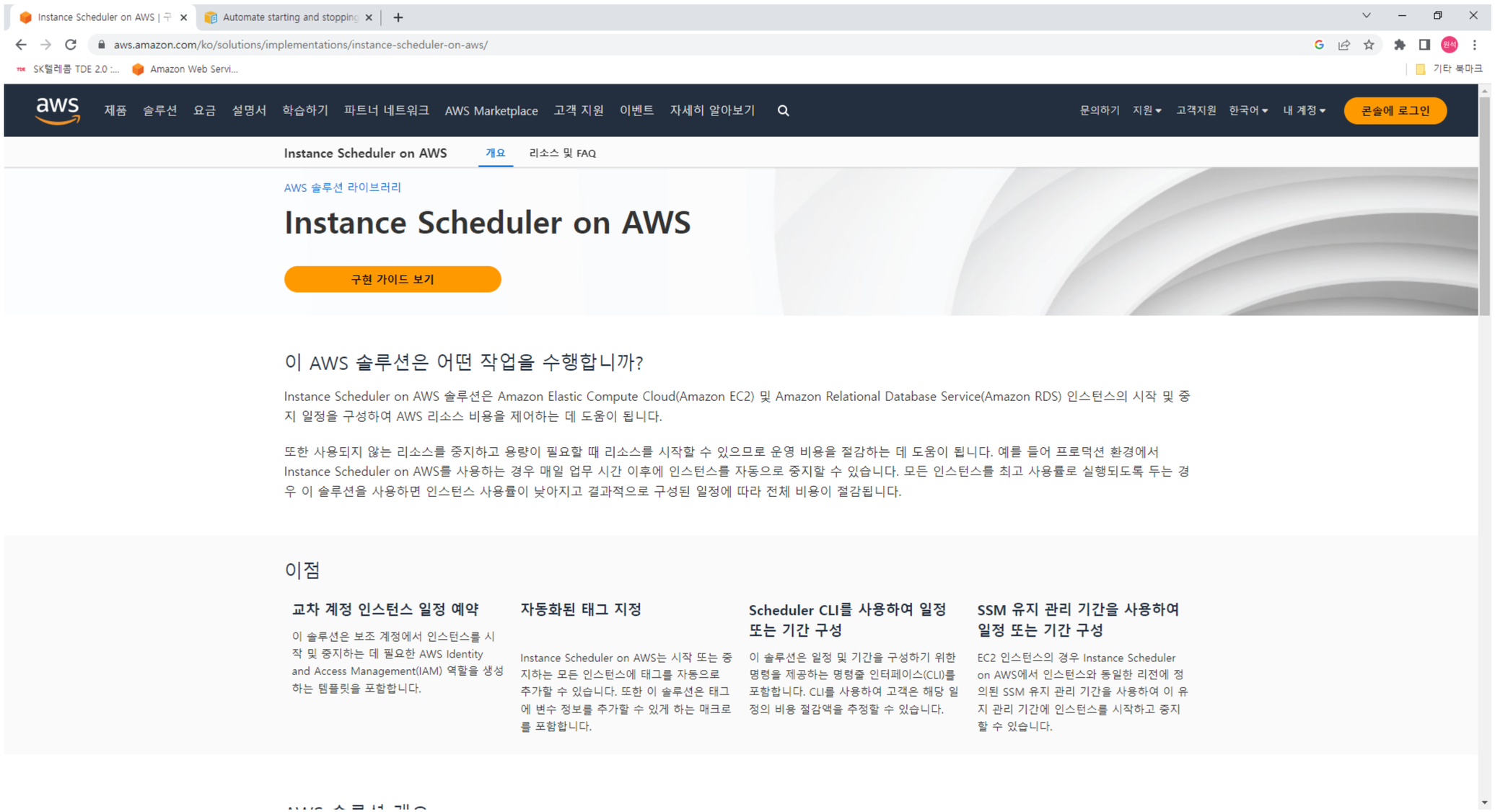
- 평일 7~22시, 주말과 공휴일 미사용

- **주의사항**

1. 자동으로 on/off가 적용되는 만큼 상용 계정에는 적용하지 말 것!
2. Tag 기반으로 대상을 선정하기 때문에 Tag 할당/편집에 대한 권한관리 필요
3. Lambda에도 권한이 부여되는데 최소한의 권한 부여
4. 인프라가 자동 복구 시, Application에서도 자동복구 될 수 있게 사전 구성 필수
    - Off time에 수행되는 Business 로직이 있다면, 시간대 변경.ex) DB Procedure 동작

2. Cloud 비용 최적화 사례, Off time 최적화

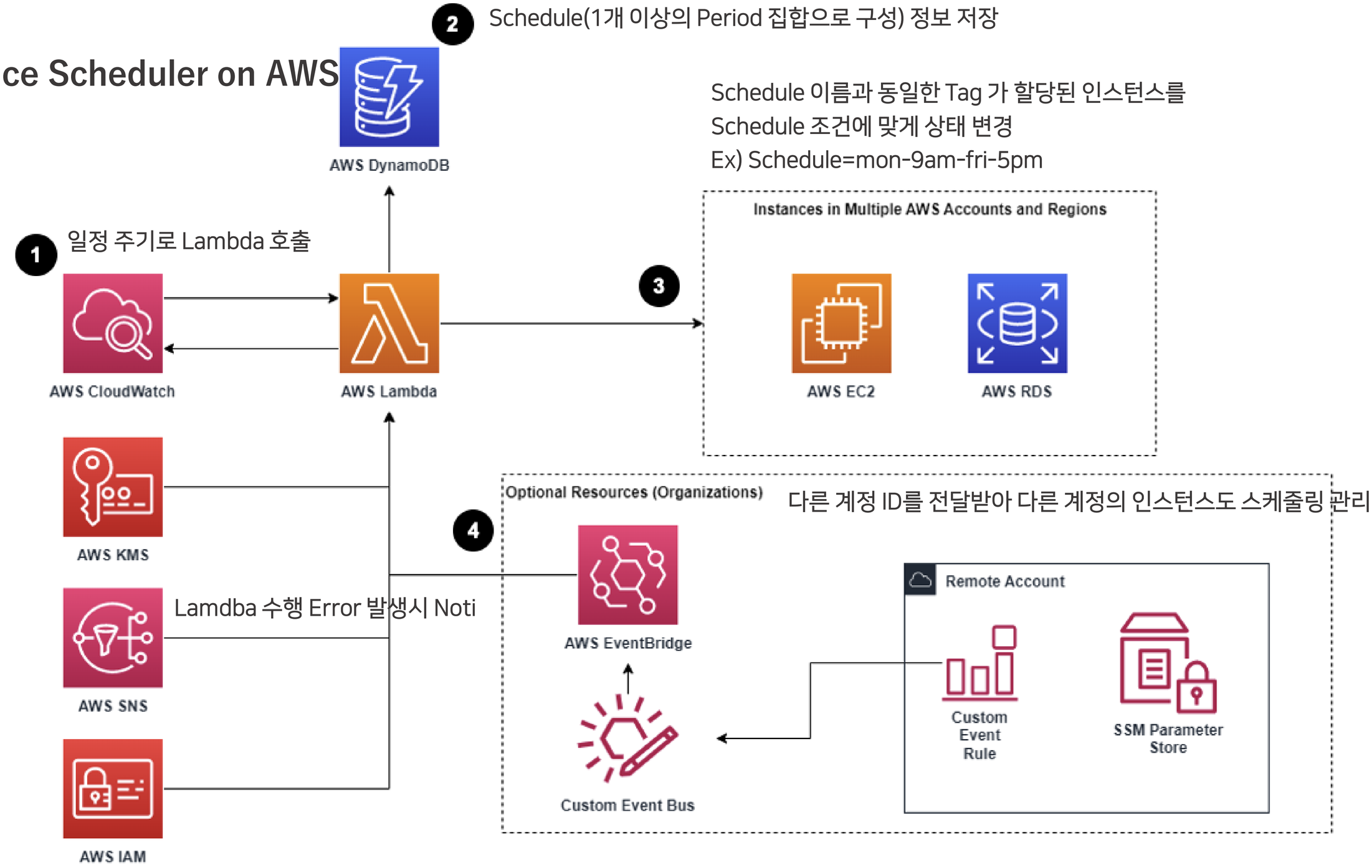
Instance Scheduler on AWS



Ref : https://aws.amazon.com/ko/solutions/implementations/instance-scheduler-on-aws/

2. Cloud 비용 최적화 사례, Off time 최적화

Instance Scheduler on AWS



Ref : https://docs.aws.amazon.com/solutions/latest/instance-scheduler-on-aws/architecture-overview.html

2. Cloud 비용 최적화 사례, Off time 최적화

Instance Scheduler on AWS DynamoDB에 Schedule과 Period

Period

```
{
   "Periods": [
      {
         "Name": "weekdays",
         "Endtime": "18:00",
         "Type": "period",
         "Weekdays": [
            "mon-fri"
         ],
         "Begintime": "09:00"
      },
      {
         "Name": "weekends",
         "Type": "period",
         "Weekdays": [
            "sat-sun"
         ],
         "Description": "Days in weekend"
      }
   ]
}
```

Schedule

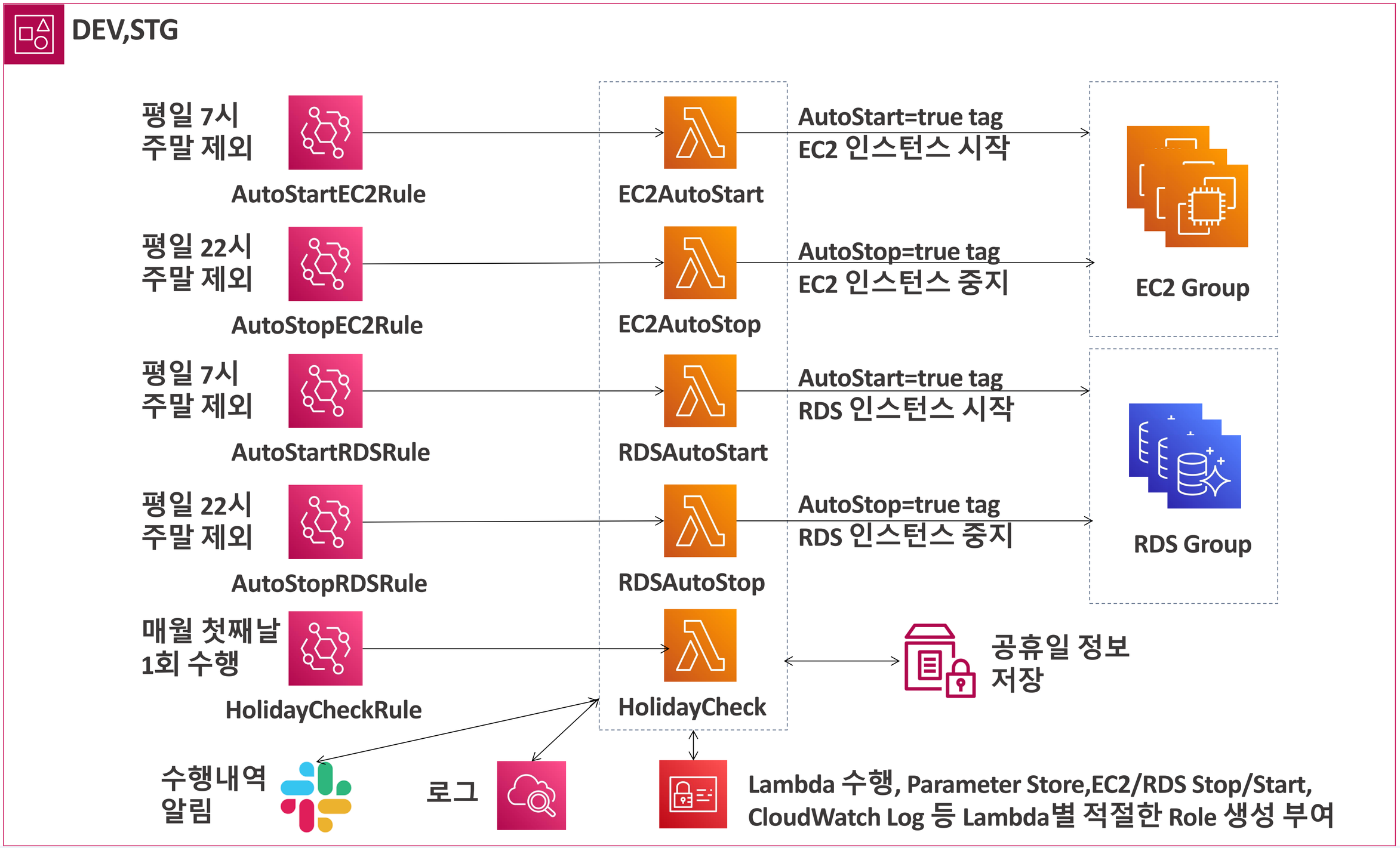
```
{
   "Schedules": [
      {
         "Timezone": "KST",
         "Type": "schedule",
         "Periods": [
            "weekdays",
            "weekends"
         ],
         "Name": "office-hours"
      }
   ]
}
```

2. Cloud 비용 최적화 사례, Off time 최적화

RDS start/stop, EC2 start/stop 하는 자체 Lambda 구성

1. [Simple] 복잡한 스케줄이 아닌, 평일 7-22시 On, 주말/공휴일 Off 스케줄만 있으면 됨
2. [Cost] 스케줄을 Event Bridge로 대신하고, 별도의 DB 구성 안함
3. [Operation] 자동화 이지만 동작에 대한 가시성 확보를 위해 On/Off 내역 Slack Notification 구성
4. [Custom] 공휴일에도 Off 할 수 있게 휴일 정보 Paramter Store에 저장하고 활용

2. Cloud 비용 최적화 사례, Off time 최적화



2. Cloud 비용 최적화 사례, Off time 최적화

EC2 / RDS stop 제한사항

EC2

- EIP로 고정되지 않은 Public IP는 변경됨
- Private IP는 변경되지 않음
- EBS볼륨 데이터는 유지되지만 RAM 데이터는 사라짐
- Stopping → stopped 상태 변경
- Auto Scaling 인스턴스라면 종료 후 대체하는 인스턴스가 다시 생김
- Advanced 설정에 Stop protection이 설정이 Enable 된 인스턴스는 종료 불가

RDS

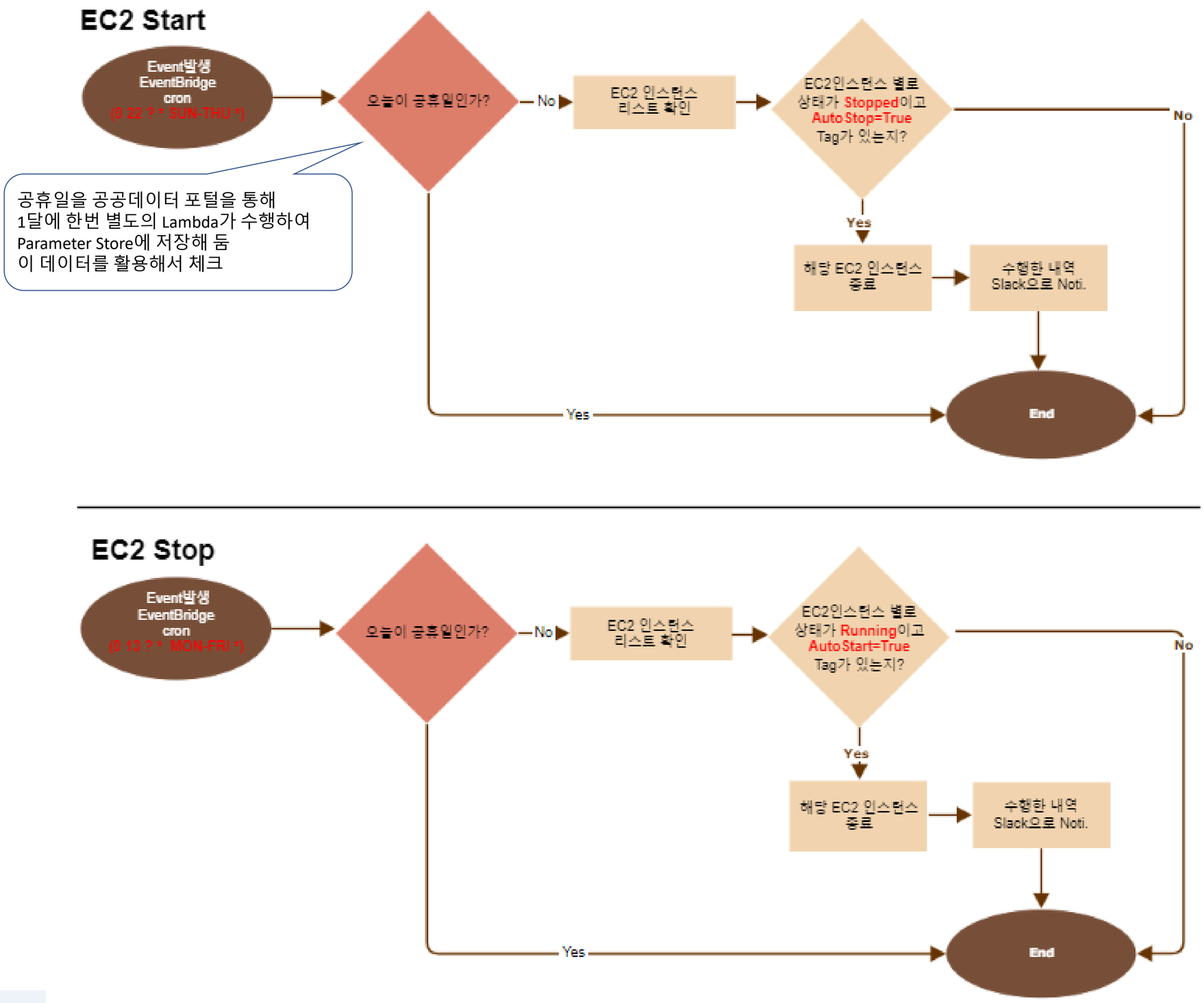
- Read Replica가 없어야 Stop 가능
- Read Replica는 Delete만 가능(stop 불가)
- Stop 된 후 그대로 두면 7일 후 자동으로 재시작

Aurora

- 개별 DB Instance 단위로는 stop 불가
- Cluster 단위별로만 Stop 가능
- Stop 된 후 그대로 두면 7일 후 자동으로 재시작

2. Cloud 비용 최적화 사례, Off time 최적화

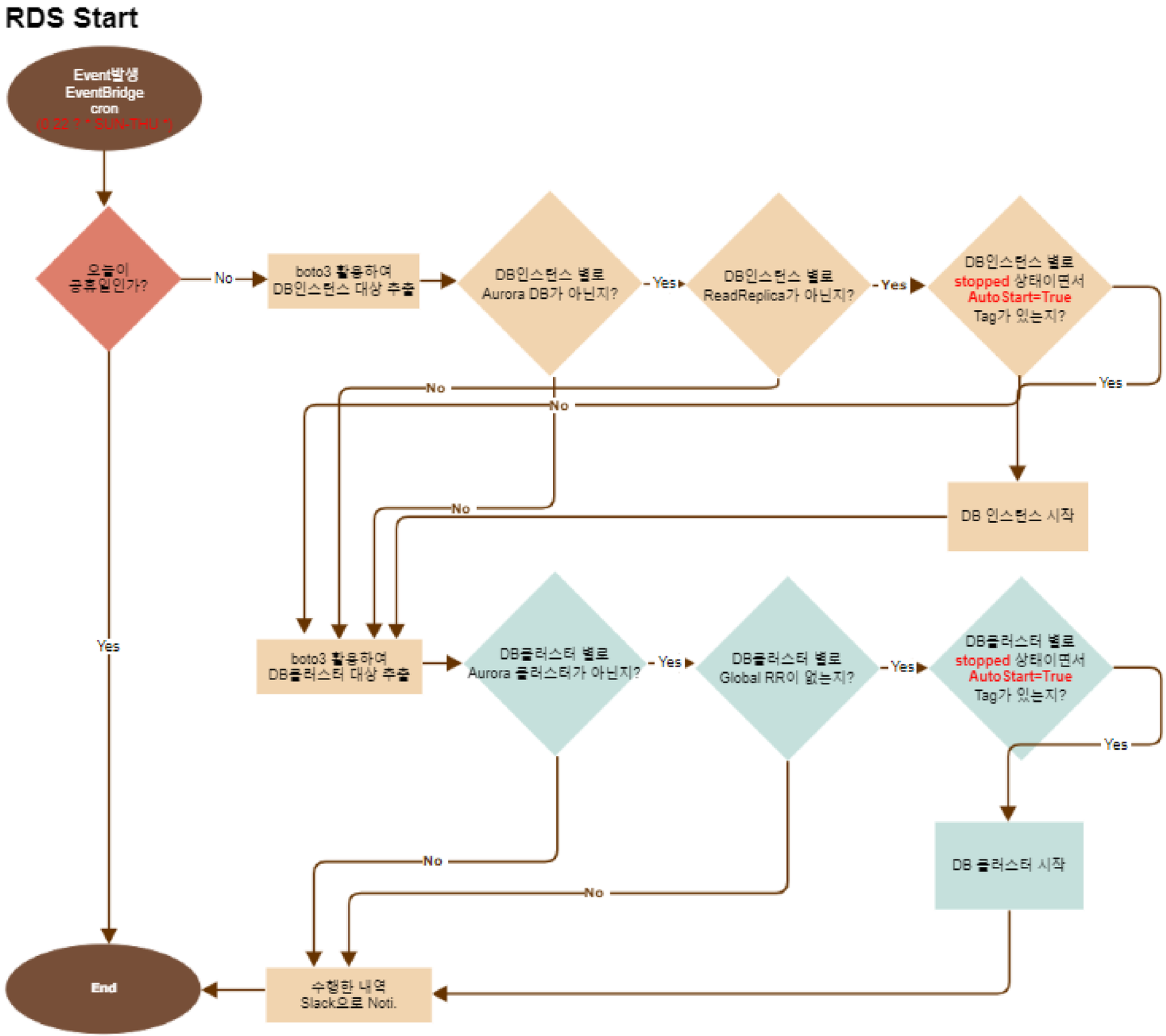
EC2 Stop/Start Lambda 순서도



2. Cloud 비용 최적화 사례, Off time 최적화

RDS Start

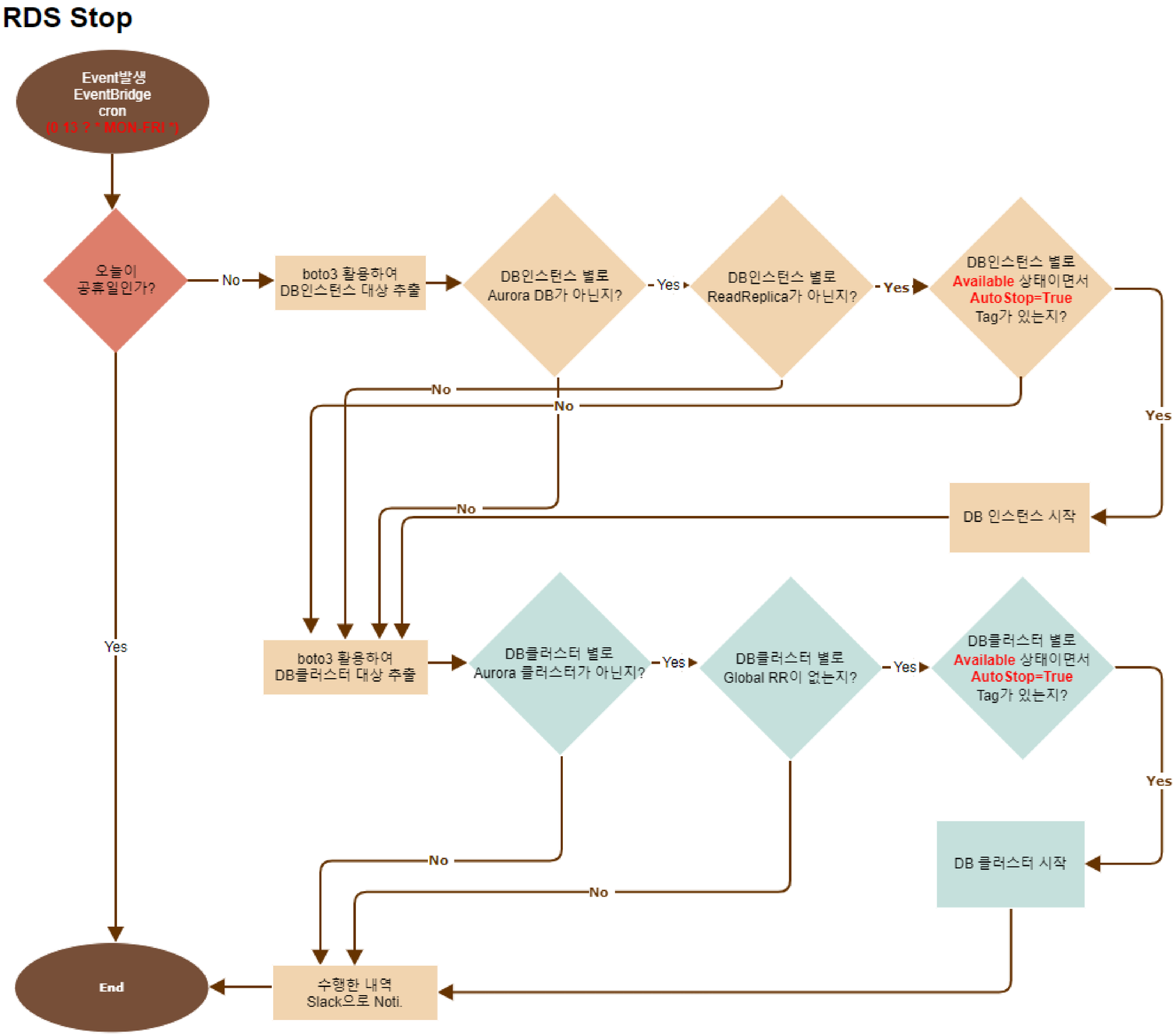
Lambda 순서도



2. Cloud 비용 최적화 사례, Off time 최적화

RDS Stop

Lambda 순서도



2. Cloud 비용 최적화 사례, Off time 최적화

Stop 지원이 안되는 리소스는 Modify로 수동 관리

MSK

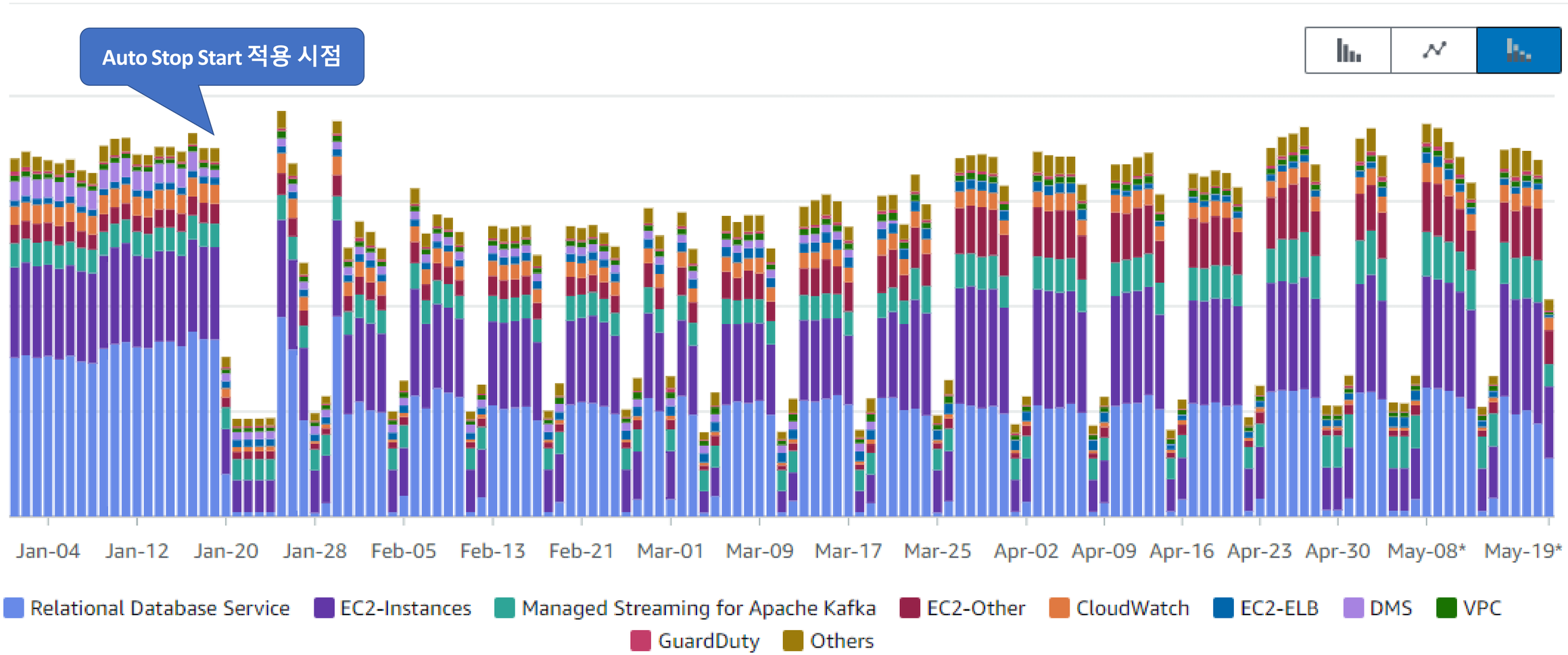
- TANGO 계정에서 EC2, RDS 다음 3번째로 많은 비용 차지
- MSK는 Stop이 지원되지 않음, 삭제나 변경만 가능
- MSK 내 인스턴스(Broker) Scale-in 기능 출시 준비 단계('23년)
- Broker당 topic별 분산처리 목적 Partition 생성하는데 Partition 개수에 따른 최소 Broker type으로 변경

DMS

- DMS는 On-prem에 있는 Oracle DB에서 Cloud상에 RDS로 데이터 전송 시 활용
- 개발 검증 시에만 일회성으로 필요하고 실시간으로 활용하지 않음
- 평상시 최소 Type으로 변경, 개발 검증 시 필요 type으로 변경하여 활용 중

2. Cloud 비용 최적화 사례, Off time 최적화

STG 비용 추세 변화



3. Cloud 비용 최적화 사례, 그 외

비용 절감효과의 차이는 있었지만, 유효했던 비용 최적화 추가 사례

- **MSK 리소스 최적화**
  - 네트워크 bandwidth, latency 기반 리소스 선정 계산
- **EC2, EKS, RDS Graviton 전환**
  - Managed 서비스는 쉽게 전환, EC2,EKS는 그 위에 App, Package ARM 형태로 전환 중
- **RDS 설정 최적화**
  - I/O가 높은 비중을 차지하는 Tango의 경우 I/O-Optimized Aurora 설정
- **S3 스토리지 디렉토리 구조 단순화**
  - Data 통계 경로를 날짜, 시스템 별 → 날짜
- **Datadog에서 Cloudwatch 수집 주기 조정**
  - 10분 -> 30분

4. Cloud 운영 방식

자동화/IaC 그리고 시각화

- **Cloud의 모든 리소스에 대한 변경관리는 Code 기반 관리**
  - IaC(CloudFormation) 기반 EC2, RDS, EKS 등 구성, tag 포함
  - 앞서 언급했던 lambda등 모든 코드는 CF로 구성됨
  - 지속적인 통합을 위해 AWS Code Commit으로 관리, AWS Code Build/Lambda 로 수행
- 자동화의 관리를 통해 이를 시각화 하기 위해 노력 중
  - Off Time 적용대상 알림, EKS 비용 최적화 대상 알림, 서비스별 비용 리포트 등

10:00 Account : tango-stg
33 DB Cluster were stopped.
0 DB Cluster tag were not setting for AutoStop.
0 DB Cluster cannot stop becuase of Read-Replica.

```
rds-tgo-stg-fm-alm1-mysql
rds-tgo-stg-fm-alm2-mysql
rds-tgo-stg-fm-alm3-mysql
rds-tgo-stg-fm-cfr1-mysql
rds-tgo-stg-fm-cfr2-mysql
rds-tgo-stg-fm-cfr3-mysql
rds-tgo-stg-fm-cfr4-mysql
rds-tgo-stg-fm-comm-mysql
rds-tgo-stg-fm-eqp-mysql
rds-tgo-stg-fm-falt-mysql
rds-tgo-stg-fm-if-mysql
rds-tgo-stg-fm-stat1-mysql
rds-tgo-stg-fm-stat2-mysql
rds-tgo-stg-fm-stc-mysql
rds-tgo-stg-fm-tree-mysql
rds-tgo-stg-oc-mysql
```

비용 incoming-webhook 앱 오전 7:00
5/18(Thursday) tango-stg 비용

Service	Δ% Last 7d
Amazon Elastic Compute Cloud - Compute	-3%
Amazon Relational Database Service	4%
EC2 - Other	7%
Amazon Managed Streaming for Apache Kafka	1%
AmazonCloudWatch	8%
Amazon Elastic Load Balancing	-9%
Amazon Virtual Private Cloud	-1%
Amazon Simple Storage Service	-60%
Amazon GuardDuty	1%
AWS Direct Connect	1%
Other	-11%
Total	-1%

비용 incoming-webhook 앱 오전 9:07
[stg, 20230518] EKS Pod별 가이드 점검
- 세부내용: s3://s3-tgo-stg-eks-mgmt/'eks_resource/20230518/5_eks_container_check.csv'

namespace	kind	requests cpu	requests memory gi	pod count	workload count	check define resource	check hpa	check hpa working	check hostpath	check strategy	check probe	check safe to evict	check emptyDir sizeLimit	check working	check rt
airflow	Deployment	1.55	2.3125	1.0	1	0	-1	-1	0	0	-1	0	-1	0	0
airflow	no_owner	0.3	0.75	1.0	1	0	0	-1	0	0	0	0	-1	0	0
ns-tgo-etl	no_owner	0.4	0.1875	1.0	1	0	0	-1	0	0	0	0	0	0	0
ns-tgo-fm	CronJob	33.95	57.2109375	138.0	138	0	0	-138	0	0	0	0	0	0	-9
ns-tgo-fm	Deployment	49.84	617.236328125	314.0	170	-2	-149	-170	0	0	-3	0	0	0	-75
ns-tgo-fm	Job	21.45	119.8203125	154.0	154	0	0	-154	0	0	0	0	0	0	-78
ns-tgo-fm	Pod	2.1	34.671875	21.0	21	0	0	-21	0	0	-21	0	-21	0	0
ns-tgo-fm	StatefulSet	2.55	124.0234375	6.0	4	0	0	-4	0	0	0	0	0	0	0
ns-tgo-oc	CronJob	1.3	2.0625	10.0	10	0	0	-10	0	0	0	0	0	0	0
ns-tgo-oc	Deployment	93.424	341.1953125	117.0	57	0	-57	-57	0	0	0	0	-20	0	-9
ns-tgo-oc	StatefulSet	49.357	283.5625	204.0	83	0	0	-83	0	0	0	0	-41	0	0
ns-tgo-pf	Deployment	13.1	31.1171875	23.0	12	0	-12	-12	0	0	0	0	0	0	0
ns-tgo-pf	StatefulSet	0.5	0.5	4.0	4	-4	0	-4	0	0	0	0	-2	0	0
ns-tgo-pm	CronJob	7.72	35.5	15.0	15	-1	0	-15	0	0	0	0	-4	-1	0
ns-tgo-pm	Deployment	77.6	377.0078125	36.0	23	0	-20	-23	0	0	-13	0	-9	-5	-1
ns-tgo-pm	Pod	11.0	24.0	4.0	4	0	0	-4	0	0	-4	0	-4	0	0
ns-tgo-pm	StatefulSet	0.45	9.0	9.0	1	0	0	-1	0	0	0	0	0	0	0