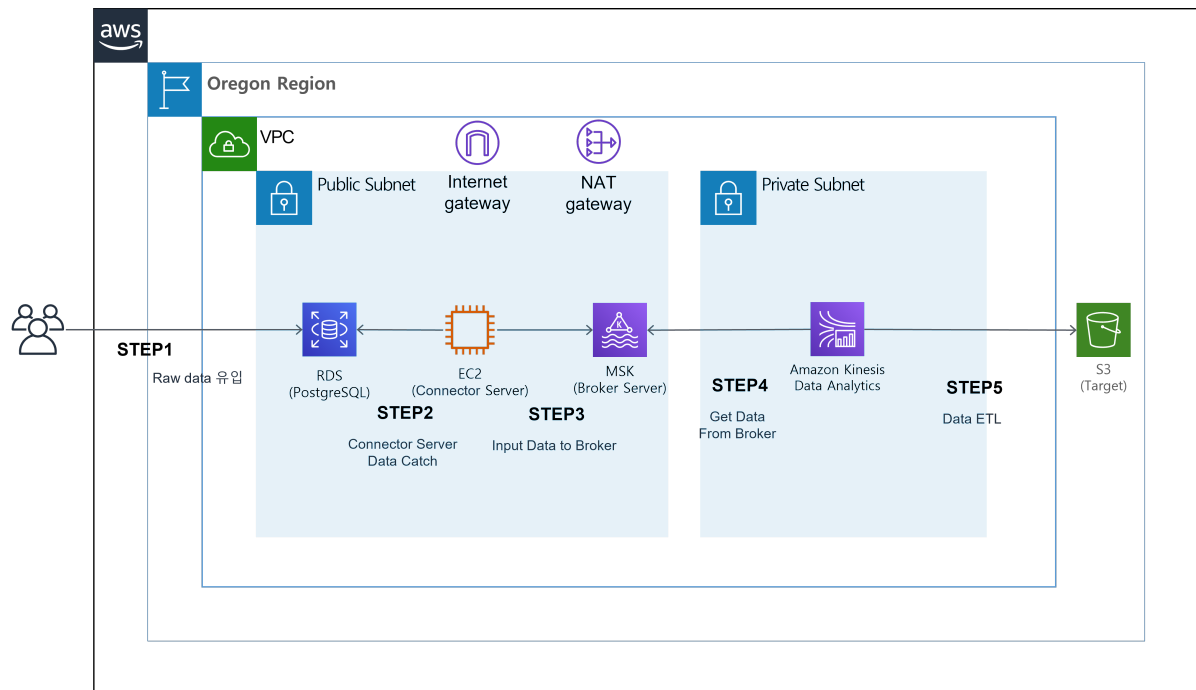


# AWS에서 Streaming Data를 ETL 작업 후 S3에 저장하기



DB에 실시간 들어오는 Data ( Row )를 Kafka Connector (debezium, jdbc ... etc)를 이용하여 MSK로 Ingest 및 KDA로 ETL 작업 후에 S3에 적재 시나리오

모든 구성은 oregon(us-west-2)에서 구성하였습니다.

Resource :

- RDS(MySQL)
- MSK
- Kinesis Data Analytics
- S3
- 네트워크 구성 (VPC, Public Subnet , Private Subnet, Routing Table , IGW, NATGW, Security Group)
- EC2
- iam role

## 1. 네트워크 구성

### 1. VPC

1. 이름 : {VPC\_NME}
2. IPv4 CIDR 블록 : 10.0.0.0/16

### 2. Subnet

1. Public Subnet1
  1. IPv4 CIDR 블록 : 10.0.1.0/24
  2. 가용 영역 : us-west-2a
2. Public Subnet2

1. IPv4 CIDR 블록 : 10.0.2.0/24
  2. 가용 영역 : us-west-2c
3. Private Subnet1
1. IPv4 CIDR 블록 : 10.0.3.0/24
  2. 가용 영역 : us-west-2a
4. Private Subnet1
1. IPv4 CIDR 블록 : 10.0.4.0/24
  2. 가용 영역 : us-west-2c

### 3. Internet Gateway (IGW)

1. VPC : 상단에서 생성한 VPC\_ID

### 4. Elastic IP (EIP)

1. 네트워크 경계 그룹 : us-west-2
2. 퍼블릭 IPv4 주소 풀 : Amazon IPv4 주소 풀

### 5. NAT Gateway (NATG)

1. 서브넷 : {Public Subnet1\_ID}
2. 연결 유형 : 퍼블릭
3. 탄력적 IP 주소 : 상단에서 생성한 EIP\_ID

### 6. Routing Table

#### 1. Public RT

1. VPC : 상단에서 생성한 VPC\_ID
2. 라우팅

대상	대상	상태	전파됨
VPC_CIDR (10.0.0.0/16)	local	활성	아니요
0.0.0.0/0	IGW_ID	활성	아니요

#### 3. 서브넷 연결

서브넷 ID	IPv4 CIDR	IPv6 CIDR
Public Subnet1_ID	Public Subnet1_CIDR (10.0.1.0/24)	-
Public Subnet2_ID	Public Subnet2_CIDR (10.0.2.0/24)	-

#### 2. Private RT

1. VPC : 상단에서 생성한 VPC\_ID
2. 라우팅

대상	대상	상태	전파됨
VPC_CIDR	local	활성	아니요
0.0.0.0/0	NATG_ID	활성	아니요

### 3. 서브넷 연결

서브넷 ID	IPv4 CIDR	IPv6 CIDR
Private Subnet1_ID	Private Subnet1_CIDR (10.0.3.0/24)	-
Private Subnet2_ID	Private Subnet2_CIDR (10.0.4.0/24)	-

### 7. Security Group (SG)

유형	프로토콜	포트 범위	소스	설명 - 선택 사항
모든 트래픽	전체	전체	10.0.0.0/16 (VPC CIDR)	VPC CIDR BLOCK
모든 트래픽	전체	전체	해당 {SG-ID}	refer to self for msk

## 2. Source 구성 ( RDS MySQL )

### 1. Subnet Group 생성

- 이름 : {YOUR\_Subnet\_Group\_Name}
- 설명 : Subnet Group for Data Source
- VPC : {Your\_VPC}

서브넷 그룹 세부 정보

이름

서브넷 그룹이 생성된 후에는 이름을 수정할 수 없습니다.

cjm-public-sg

1~255자로 구성되어야 합니다. 영숫자, 공백, 하이픈, 밑줄 및 마침표를 사용할 수 있습니다.

설명

cjm public subnet group

VPC

DB 서브넷 그룹에 사용할 서브넷에 해당하는 VPC 식별자를 선택합니다. 서브넷 그룹이 생성된 후에는 다른 VPC 식별자를 선택할 수 없습니다.

cjm-vpc

- 가용 영역 : us-west-2a, us-west-2c

- 서브넷 : {Your\_Subnet\_A}, {Your\_Subnet\_C}

## 서브넷 추가

### 가용 영역

추가할 서브넷이 포함된 가용 영역을 선택합니다.

가용 영역 선택

us-west-2a ✕

us-west-2c ✕

### 서브넷

추가할 서브넷을 선택합니다. 목록에는 선택한 가용 영역의 서브넷이 포함됩니다.

서브넷 선택

subnet- [REDACTED] (10.0.1.0/24) ✕

subnet- [REDACTED] (10.0.2.0/24) ✕

### 서브넷이 선택됨 (2)

가용 영역	서브넷 ID	CIDR 블록
us-west-2a	[REDACTED]	10.0.1.0/24
us-west-2c	[REDACTED]	10.0.2.0/24

## 2. 파라미터 그룹 생성

RDS의 기본 파라미터 그룹은 bin log을 읽을 수 없도록 설정이 되어 있습니다.  
따라서 bin log를 읽을 수 있도록 파라미터 그룹을 생성하고 RDS에 적용해야 합니다.  
최초 파라미터 그룹을 생성 후 수정 진행합니다.

- 파라미터 그룹 패밀리 : `mysql8.0`
- 그룹 이름 : `{Your_Parameter_Group_Name}`
- 설명 : `Your Data Source`

## 파라미터 그룹 세부 정보

### 파라미터 그룹 패밀리

이 DB 파라미터 그룹을 적용할 DB 패밀리

mysql8.0

### 그룹 이름

DB 파라미터 그룹의 식별자

kafka-mysql8

### 설명

DB 파라미터 그룹에 대한 설명

Your Data Source

[Binlog configuration properties](#) 다음 링크를 통해 설정값을 확인 및 수정합니다.  
생성 된 파라미터 그룹을 변경합니다.

이름	변경 전 값	변경 후 값
binlog_format	MIXED	ROW
binlog_row_image	full, minimal, noblob	full

### 3. RDS 생성

1. 데이터베이스 생성 방식 선택 : 표준 생성
2. 엔진 옵션 : MySQL
3. 버전 : MySQL.8.0.23


데이터베이스 생성 방식 선택정보


☒ 표준 생성  
가용성, 보안, 백업 및 유지 관리에 대한 옵션을 포함하여 모든 구성 옵션을 설정합니다.


☐ 손쉬운 생성  
권장 모범 사례 구성을 사용합니다. 일부 구성 옵션은 데이터베이스를 생성한 후 변경할 수 있습니다.


엔진 옵션


엔진 유형 정보


☐ Amazon Aurora  


☒ MySQL  


☐ MariaDB  


☐ PostgreSQL  


☐ Oracle  


☐ Microsoft SQL Server  


에디션

☒ MySQL Community

알려진 문제/제한 사항

알려진 문제/제한 사항 [🔗](#)를 검토하여 특정 데이터베이스 버전과 발생할 수 있는 호환성 문제를 확인하세요.

버전

MySQL 8.0.23 ▼

4. 템플릿 : 프로덕션
5. DB 인스턴스 식별자 : {Your\_DB\_ID}
6. 마스터 사용자 이름 : admin
7. 암호 : Bespin12!

## 템플릿

해당 사용 사례를 충족하는 샘플 템플릿을 선택하세요.



### 프로덕션

고가용성 및 빠르고 일관된 성능을 위해 기본값을 사용하세요.



### 개발/테스트

이 인스턴스는 프로덕션 환경 외부에서 개발 용도로 마련되었습니다.



### 프리 티어

RDS 프리 티어를 사용하여 새로운 애플리케이션을 개발하거나, 기존 애플리케이션을 테스트하거나 Amazon RDS에서 실무 경험을 쌓을 수 있습니다. [정보](#)

## 설정

### DB 인스턴스 식별자 정보

DB 인스턴스 이름을 입력하세요. 이름은 현재 AWS 리전에서 AWS 계정이 소유하는 모든 DB 인스턴스에 대해 고유해야 합니다.

kafka\_source\_db

DB 인스턴스 식별자는 대소문자를 구분하지 않지만 'mydbinstance'와 같이 모두 소문자로 저장됩니다. 제약: 1자~60자의 영숫자 또는 하이픈으로 구성되어야 합니다. 첫 번째 문자는 글자이어야 합니다. 하이픈 2개가 연속될 수 없습니다. 끝에 하이픈이 올 수 없습니다.

### ▼ 자격 증명 설정

#### 마스터 사용자 이름 정보

DB 인스턴스의 마스터 사용자에게 로그인 ID를 입력하세요.

admin

1~16자의 영숫자. 첫 번째 문자는 글자이어야 합니다.

#### ☐ 암호 자동 생성

Amazon RDS에서 사용자를 대신하여 암호를 생성하거나 사용자가 직접 암호를 지정할 수 있습니다.

#### 마스터 암호 정보

\*\*\*\*\*

제약 조건: 8자 이상의 인쇄 가능한 ASCII 문자. 다음은 포함할 수 없습니다. /(슬래시), '(작은따옴표)', "(큰따옴표) 및 @ (앳 기호).

#### 암호 확인 정보

\*\*\*\*\*

8. DB 인스턴스 클래스 : 버스터블 클래스(t 클래스 포함)/db.t3.micro2 (최소 사양)

## DB 인스턴스 클래스

### DB 인스턴스 클래스 정보

- ☐ 스탠다드 클래스(m 클래스 포함)
- ☐ 메모리 최적화 클래스(r 및 x 클래스 포함)
- ☒ 버스터블 클래스(t 클래스 포함)

db.t3.micro  
2 vCPUs 1 GiB RAM 네트워크: 2,085Mbps

☐ 이전 세대 클래스 포함

## 스토리지

### 스토리지 유형 정보

프로비저닝된 IOPS(SSD)

### 할당된 스토리지

100

GiB

최소: 100 GiB, 최대: 16,384 GiB

### 프로비저닝된 IOPS 정보

3000

IOPS

최소: 1,000 IOPS, 최대: 80,000 IOPS

📌 데이터베이스 워크로드 및 인스턴스 유형에 따라 실제 IOPS는 프로비저닝한 양과 다를 수 있습니다. 자세히 알아보기

### 스토리지 자동 조정 정보

애플리케이션의 필요에 따라 데이터베이스 스토리지의 동적 조정 지원을 제공합니다.

#### ☒ 스토리지 자동 조정 활성화

이 기능을 활성화하면 지정한 임계값 초과 시 스토리지를 늘릴 수 있습니다.

### 최대 스토리지 임계값 정보

데이터베이스를 지정한 임계값으로 자동 조정하면 요금이 부과됩니다.

1000

GiB

최소: 101 GiB, 최대: 16,384 GiB

9. 가용성 및 내구성 : 대기 인스턴스를 생성하지 마세요 .

10. VPC : {Subnet\_Group 과 동일한 VPC}

11. 서브넷 그룹 : {위에서 생성한 Subnet\_Group}

12. 퍼블릭 액세스 : 예

13. VPC 보안 그룹 : {Your\_Security\_Group}

14. 가용 영역 : us-west-2a

## 가용성 및 내구성

### 다중 AZ 배포 정보

- ☐ 대기 인스턴스 생성(생산 사용량에 권장)  
데이터 중복을 제공하고, I/O 중지를 없애고, 시스템 백업 중에 지연 시간 스파이크를 최소화하기 위해 다른 가용 영역(AZ)에 대기 인스턴스를 생성합니다.
- ☒ 대기 인스턴스를 생성하지 마세요.

## 연결



### Virtual Private Cloud(VPC) 정보

이 DB 인스턴스의 가상 네트워킹 환경을 정의하는 VPC.

cjm-vpc (vpc-02067642b982c2cf7)

해당 DB 서브넷 그룹이 있는 VPC만 나열됩니다.

❗ 데이터베이스를 생성한 후에는 VPC 선택을 변경할 수 없습니다.

### 서브넷 그룹 정보

선택한 VPC에서 DB 인스턴스가 어떤 서브넷과 IP 범위를 사용할 수 있는지를 정의하는 DB 서브넷 그룹.

cjm-public-sg

### 퍼블릭 액세스 정보

- ☒ 예  
VPC 외부의 Amazon EC2 인스턴스 및 디바이스는 데이터베이스에 연결할 수 있습니다. 데이터베이스에 연결할 수 있는 VPC 내부의 EC2 인스턴스 및 디바이스를 지정하는 하나 이상의 VPC 보안 그룹을 선택하세요.
- ☐ 아니요  
RDS는 데이터베이스에 퍼블릭 IP 주소를 할당하지 않습니다. VPC 내부의 Amazon EC2 인스턴스 및 디바이스만 데이터베이스에 연결할 수 있습니다.

### VPC 보안 그룹

데이터베이스에 대한 액세스를 허용할 VPC 보안 그룹을 선택합니다. 보안 그룹 규칙이 적절한 수신 트래픽을 허용하는지 확인합니다.

☒ 기존 항목 선택  
기존 VPC 보안 그룹 선택

☐ 새로 생성  
새 VPC 보안 그룹 생성

### 기존 VPC 보안 그룹

VPC 보안 그룹 선택

cjm-security\_group-public X

### 가용 영역 정보

us-west-2a

▶ 추가 구성

15. 데이터베이스 인증 : 암호 인증



## 데이터베이스 인증

### 데이터베이스 인증 옵션 정보

- ☒ 암호 인증  
데이터베이스 암호를 사용하여 인증합니다.
- ☐ 암호 및 IAM 데이터베이스 인증  
AWS IAM 사용자 및 역할을 통해 데이터베이스 암호와 사용자 자격 증명을 사용하여 인증합니다.
- ☐ 암호 및 Kerberos 인증  
권한이 부여된 사용자가 Kerberos 인증을 사용하여 이 DB 인스턴스에서 인증하도록 허용하려는 디렉터리를 선택합니다

### ▶ 추가 구성

데이터베이스 옵션, 암호화 활성화됨, 백업 활성화됨, 역추적 비활성화됨, 향상된 모니터링 활성화됨, 유지 관리, CloudWatch Logs, 삭제 보호 활성화됨

16. 초기 데이터베이스 : test

17. 파라미터 그룹 : {위에서 생성한 파라미터 그룹}

### ▼ 추가 구성

데이터베이스 옵션, 암호화 활성화됨, 백업 비활성화됨, 역추적 비활성화됨, 향상된 모니터링 활성화됨, 유지 관리, CloudWatch Logs, 삭제 보호 활성화됨

## 데이터베이스 옵션

### 초기 데이터베이스 이름 정보

test

데이터베이스 이름을 지정하지 않으면 Amazon RDS에서 데이터베이스를 생성하지 않습니다.

### DB 파라미터 그룹 정보

kafka-mysql8

### 옵션 그룹 정보

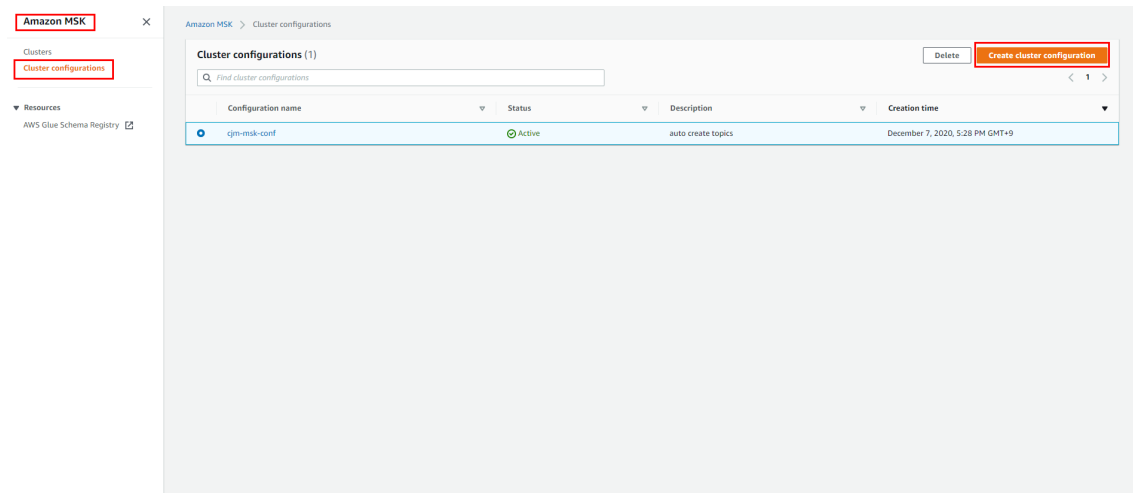
default:mysql-8-0

## 3. MSK

MSK는 Apache Kafka를 토대로 구현된 서비스이나 Apache Kafka에 비해 제약 사항이 많습니다. 하지만 Cloud Native한 workflow 구현을 위해 MSK로 Kafka cluster를 구성하였습니다.

### 1. MSK Configuration 생성

[msk config 생성](#) 해당 링크로 이동하여 `create cluster configuration`을 누릅니다.



config 구성은 다음과 같습니다.

1. Configuration name : `{Config_Name}`
2. Kafka version : `{To-Be_Kafka_Version}`
3. code :

```
auto.create.topics.enable=true
default.replication.factor=3
min.insync.replicas=2
num.io.threads=8
num.network.threads=5
num.partitions=1
num.replica.fetchers=2
replica.lag.time.max.ms=30000
socket.receive.buffer.bytes=102400
socket.request.max.bytes=104857600
socket.send.buffer.bytes=102400
unclean.leader.election.enable=true
zookeeper.session.timeout.ms=18000
delete.topic.enable=true
```

General

Configuration name

msk-conf

A unique name is required, but the name can't start with a hyphen. Valid characters: a-z, A-Z, 0-9, and - (hyphen)

Description - optional

Config for MSK

▼ Configuration properties for revision 1

Amazon MSK uses a default configuration to get you started quickly. [Learn more](#)

Apache Kafka version for code syntax - optional

Choose the Apache Kafka version. Configuration properties and values will appear in the section below.

2.2.1

↺

Copy from existing cluster

Code

```

1 auto.create.topics.enable=true
2 default.replication.factor=3
3 min.insync.replicas=2
4 num.io.threads=8
5 num.network.threads=5
6 num.partitions=1
7 num.replica.fetchers=2
8 replica.lag.time.max.ms=30000
9 socket.receive.buffer.bytes=102400
10 socket.request.max.bytes=104857600
11 socket.send.buffer.bytes=102400
12 unclean.leader.election.enable=true
13 zookeeper.session.timeout.ms=18000
14 delete.topic.enable=true

```

## 2. MSK Cluster 생성

1. Creation method : Custom create
2. Cluster name : {Your\_Kafka\_Cluster\_Name}
3. Apache Kafka version : {Same\_With\_Your\_Configuration}

### Creation method

Choose a method to apply settings to your cluster. You can change some of these settings after the cluster is created.

☐ Quick create  
Use recommended best-practice settings to create a starter cluster.

☒ Custom create  
Specify all settings, including security, availability, and custom configuration.

### Cluster name

Cluster name  
You can't change it after you create the cluster.

cjm-msk

The name must be unique and can have a maximum of 64 characters.

### Apache Kafka version

The Apache Kafka version that you want on the brokers. [Learn more](#)

Apache Kafka version

2.2.1

#### 4. Cluster configuration : Custom configuration

이미 Custom하게 생성을 해놨으므로 Custom 생성이 없다면 MSK default configuration 사용

#### 5. Cluster configuration : {Your\_MSK\_Configuration}

#### 6. Configuration revision : Revision 1

Configuration 업데이트 이력이 없다면 1을 선택하고, 이력이 있다면 최신 버전을 선택합니다  
최신일수록 숫자가 높습니다.

### Configuration

Configurations are settings and parameters that affect Apache Kafka brokers and default topic settings. You can use the default configuration that MSK provides or create a custom configuration. [Learn more](#)

☐ MSK default configuration  
Create a new configuration using MSK defaults to get you started quickly. You can apply another configuration after you create the cluster. [Learn more](#)

☒ Custom configuration  
Choose a custom configuration. [Learn more](#)

Cluster configuration

cjm-msk-conf

Create configuration

Configuration revision

Revision 4

#### 7. VPC : {Your\_VPC}

#### 8. Number of Zones : 2

2 이상 가능하면 3을 권장 합니다.

설정한 숫자만큼 HA 구성이 가능합니다.

9. Zone : {Available\_Zone\_Name}

10. Subnet : {Your\_Subnet}

### Networking

The Amazon VPC, Zones, and subnets where you want Amazon MSK to deploy your brokers. [Learn more](#)

**VPC**  
Defines the virtual networking environment for this cluster. You can't change this setting after you create the cluster. [Learn more](#)

(cjm-vpc) ↕ ↻ Create VPC

**Number of Zones**  
Specifies the number of isolated Zones in which brokers are distributed. You can't change this number after the cluster is created.

☐ 3 (recommended)

☒ 2

---

**First Zone**

Zone  

us-west-2a

Subnet  
Brokers will be hosted in this subnet.  
 (cjm-subnet-public-a) ↕

**Second Zone**

Zone  

us-west-2c

Subnet  
Brokers will be hosted in this subnet.  
 (cjm-subnet-public-c) ↕

11. Security groups : {Your\_SG}

12. Broker type : {Type\_Your\_Broker}

13. Number of brokers per Zone : 1

zone 생성할 갯수를 설정하는 것이므로 zone을 2로 설정하고 broker number르 1로 설정하면

총 2개가 생성됩니다.

## Security groups

A security group acts as a virtual firewall for your cluster to control incoming and outgoing traffic. Inbound rules control the incoming traffic to your cluster, and outbound rules control the outgoing traffic from your cluster. [Learn more](#)

### Security groups

Choose one or more security groups to assign to the cluster's ENIs.

- ☐ Default settings
- ☒ Custom settings

### Security groups

Choose security groups

(cjm-security\_group-public) X

## Brokers

Brokers are the EC2 instances on which Amazon MSK runs the Apache Kafka service. They store messages from producers and serve them to consumers. The performance of an MSK cluster depends on the number and type of brokers. For guidance on choosing the right number and type of brokers, see [right-size your cluster](#) in the documentation.

### Broker type

The type of a broker determines its compute, memory, and storage capabilities. For information to help you pick a type, see [Amazon MSK sizing and pricing](#).

kafka.m5.large

1000 recommended max partition count  
vCPU : 2 Memory (GiB): 8 Network bandwidth (Gbps): 10

### Number of brokers per Zone

Number of Apache Kafka brokers deployed in each Zone.

1

Minimum: 1. After you create the cluster, you can only increase the number of brokers per Zone.

**ⓘ Your cluster will have 2 total brokers, distributed evenly across your 2 Zones.**

14. EBS storage volume per broker : 1000

broker당 설정할 Storage 입니다.

MSK를 생성한 이후에 Storage에 대한 Auto Scaling을 적용할 수 있습니다.

15. Access control method : IAM access control

## Access control method

The method that you want Amazon MSK to use to authenticate clients and allow or deny actions. [Learn more](#)

### Access control method

- ☐ None  
No authentication is required for clients, and all actions are allowed.

☒ IAM access control

Amazon MSK uses IAM to authenticate and authorize clients.



#### Configure IAM

To use IAM access control, configure your client properties and provide IAM policies that allow or deny actions. [Learn more](#)

- ☐ SASL/SCRAM authentication  
SCRAM provides a username-and-password authentication method using the SASL framework. Amazon MSK uses AWS Secrets Manager to enable SASL/SCRAM. You can link secrets after you create a cluster.
- ☐ TLS client authentication through AWS certificate manager (ACM)  
Use private TLS certificates stored in ACM to authenticate the identity of clients that connect to an MSK cluster.

16. Encrypt data in transit : Enable encryption within the cluster

17. Encrypt data at rest : Use AWS managed CMK

## Encryption

Your data encryption options that you use to meet strict data-management requirements. [Learn more](#)

### Encrypt data in transit

Enable the Transport Layer Security (TLS) protocol to encrypt data as it travels between brokers within the cluster. Choose one or more encryption options for data communication between clients and brokers. [Learn more](#)

#### Within the cluster

☒ Enable encryption within the cluster

#### Between clients and brokers

☒ TLS encryption

Required for IAM, SASL/SCRAM and TLS access control methods.

☐ Plaintext

Plaintext traffic isn't possible with SASL/SCRAM or IAM access control methods.



#### TLS encryption enabled

Traffic between clients and brokers is TLS-encrypted when the access control method is IAM or SASL/SCRAM.

### Encrypt data at rest

Amazon MSK uses customer master keys (CMKs) to encrypt your data at rest. You can use AWS Key Management Services (KMS) to create and manage CMKs. [Learn more](#)

☒ Use AWS managed CMK

The AWS managed CMK (aws/kafka) is a CMK in your account that is created, managed, and used on your behalf by MSK.

☐ Use customer managed CMK

Customer managed CMKs are CMKs in your AWS account that you create, own, and manage.

18. Amazon CloudWatch metrics for this cluster : Basic monitoring

19. Open monitoring with Prometheus : enable

20. Broker log delivery : Deliver to Amazon Cloudwatch Logs , Deliver to Amazon S3 , Deliver to Amazon Kinesis Data Firehose

용도에 따라 Cloudwatch S3에 Log를 남길 수 있고  
추가 Service에 연결하려면 Firehose를 통해 연결 가능합니다.

## Monitoring

### Amazon CloudWatch metrics for this cluster

Enhanced metrics are available at an additional cost. [Learn more](#)

- ☒ **Basic monitoring**  
Includes basic cluster-level and broker-level monitoring. Available for free.
- ☐ **Enhanced broker-level monitoring**  
Also includes basic monitoring. Available at an additional cost.
- ☐ **Enhanced topic-level monitoring**  
Also includes basic and enhanced broker-level monitoring. Available at an additional cost.
- ☐ **Enhanced partition-level monitoring**  
Also includes basic, enhanced broker-level monitoring and topic-level monitoring. Available at an additional cost.

### Open monitoring with Prometheus

Prometheus is an open-source monitoring system for time-series metric data. You can also use tools that are compatible with Prometheus-formatted metrics. [Learn more](#)

- ☐ **Enable open monitoring with Prometheus**  
When you enable open monitoring with Prometheus, you can expose metrics using the JMX Exporter, the Node Exporter, or both. These metrics include cluster-level, broker-level, and topic-level information. Open monitoring is available for free but charges apply for the transfer of data across Zones.

### Broker log delivery

Broker logs enable you to troubleshoot your Apache Kafka applications and analyze communications with your MSK cluster. Amazon MSK doesn't charge for sending the logs. However, ingestion and storage charges apply based on the destination. [Learn more](#)

- ☒ **Deliver to Amazon CloudWatch Logs**  
Analyze, query, and set alarms on the logs.

#### Log group

To create a new log group, [visit Amazon CloudWatch Logs console](#)

Use the format `arn:aws:logs:[region]:[account-id]:log-group:[log-group-name]:*`

- ☒ **Deliver to Amazon S3**  
Store and retrieve raw logs in object storage.

#### Destination bucket in Amazon S3

To create a new bucket, [visit Amazon S3 console](#)

#### Prefix - optional

For example, if you specify `logs/2019-` and then write the file `date1.txt`, the Amazon S3 console shows a folder named `logs` with the object `2019-date1.txt`.

- ☐ **Deliver to Amazon Kinesis Data Firehose**  
Capture, transform, and deliver logs to Amazon Elasticsearch Service or other Kinesis Data Firehose destinations.

생성 완료되는 데 약 15분 정도 소요됩니다.

## 4. Kafka Connector 서버 구성

Connector Server를 위한 별도의 EC2가 필요하면 현 작업을 통해 구성되는 Connector로 Data Source를 가져와 Kafka Cluster에 전달 합니다.

[Kafka Tutorial](#) 해당 링크에 따라 Kafka를 설치합니다.

```
# Install Java:
sudo yum install -y java-1.8.0
```

```
# Create kafka dir:
mkdir kafka
```

```
# Move dir
cd kafka
```



```
# Get Kafka: Version에 맞는 Kafka를 Download 합니다.
wget https://archive.apache.org/dist/kafka/2.2.1/kafka_2.12-2.2.1.tgz

# Extract Kafka:
tar -xzf kafka_2.12-2.2.1.tgz

# 카프카 설치 완료
# ===

# Topic 생성
# msk 정보를 불러옵니다 zookeeper 및 broker ip 획득
aws kafka describe-cluster --cluster-arn "{ClusterArn}" --region

# mytest라는 topic을 생성합니다.
../bin/kafka-topics.sh --create --zookeeper "{ZookeeperConnectionString}" --
replication-factor 2 --partitions 1 --topic mytest

# 그 외에 알아두면 좋은 command
# Delete Topic:
../bin/kafka-topics.sh --delete --zookeeper "{ZookeeperConnectionString}" --
replication-factor 2 --partitions 1 --topic mytest

# Show Topic List:
../bin/kafka-topics.sh --list --zookeeper "{ZookeeperConnectionString}" --
replication-factor 2 --partitions 1 --topic AWSKafkaTutorialTopic

# consumer 그룹 확인
../bin/kafka-consumer-groups.sh --bootstrap-server {Bootstrap servers} --list

# consumer 상태와 오프셋
../bin/kafka-consumer-groups.sh --bootstrap-server {Bootstrap servers} --group
{consumer group id} --describe

# counsumer group 삭제
../bin/kafka-consumer-groups.sh --bootstrap-server {Bootstrap servers} --delete
--group {consumer group id}

# ===
# 카프카 connector 다운 및 설정
# 용도에 맞게 connector를 다운 로드 합니다.
# connector는 archive라는 이름으로 저장됩니다.

# s3-source 다운
wget https://api.hub.confluent.io/api/plugins/confluentinc/kafka-connect-s3-
source/versions/1.3.2/archive
# s3-sink 다운
wget https://api.hub.confluent.io/api/plugins/confluentinc/kafka-connect-
s3/versions/5.5.2/archive
# debezium 다운
wget https://api.hub.confluent.io/api/plugins/debezium/debezium-connector-
mysql/versions/1.3.1/archive
# jdbc sink&source 다운
wget https://api.hub.confluent.io/api/plugins/confluentinc/kafka-connect-
jdbc/versions/10.0.1/archive
# avro convter connector 다운
wget https://api.hub.confluent.io/api/plugins/confluentinc/kafka-connect-avro-
converter/versions/5.5.2/archive
```

```
# 압축해제
unzip archive

# plugin path dir 생성
mkdir -p plugins/kafka-connect-s3-sink
mkdir -p plugins/kafka-connect-s3-source
mkdir -p plugins/debezium
mkdir -p plugins/jdbc

# 각 connector를 plugin path로 이동
cp confluentinc-kafka-connect-s3-5.5.2/lib/* plugins/kafka-connect-s3-sink/
cp confluentinc-kafka-connect-s3-source-1.3.2/lib/* plugins/kafka-connect-s3-source/
cp debezium-debezium-connector-mysql-1.3.1/lib/* plugins/debezium/
cp confluentinc-kafka-connect-jdbc-10.0.1/lib/* plugins/jdbc/
cp confluentinc-kafka-connect-avro-converter-5.5.2/lib/* plugins/jdbc/

# connector 다운 완료

cd kafka_2.12-2.2.1/config
```

## connect 설정

```
vi connect.properties
```

```
# Kafka broker IP addresses to connect to
bootstrap.servers={MSK broker IP}

# Path to directory containing the connector jar and dependencies
plugin.path=/home/ec2-user/kafka/plugins/

# Converters to use to convert keys and values
key.converter=org.apache.kafka.connect.json.JsonConverter
value.converter=org.apache.kafka.connect.json.JsonConverter
key.converter.schemas.enable=false
value.converter.schemas.enable=false

# The internal converters Kafka Connect uses for storing offset and configuration data
internal.key.converter=org.apache.kafka.connect.json.JsonConverter
internal.value.converter=org.apache.kafka.connect.json.JsonConverter
internal.key.converter.schemas.enable=false
internal.value.converter.schemas.enable=false
offset.storage.file.filename=/tmp/connect.offsets

# connect internal topic names, auto-created if not exists
config.storage.topic=connect-configs
offset.storage.topic=connect-offsets
status.storage.topic=connect-status

# internal topic replication factors - auto 3x replication in Azure Storage
config.storage.replication.factor=2
offset.storage.replication.factor=2
status.storage.replication.factor=2
```

```
group.id=connect-cluster-group
```

### [debezium connector 설정](#)

```
vi debezium.json
```

```
{
  "name": "debezium",
  "config": {
    "connector.class": "io.debezium.connector.mysql.MySqlConnector",
    "tasks.max": "1",
    "database.hostname": "{RDS URL}",
    "database.port": "3306",
    "database.user": "admin",
    "database.password": "Bespın12!",
    "database.server.id": "1509343511",
    "database.server.name": "mytest",
    "database.whitelist": "test",
    "database.history.kafka.topic": "dbhistory.test",
    "database.history.kafka.bootstrap.servers": "{MSK broker IP}",
    "snapshot.mode": "schema_only_recovery",
    "transforms": "route",
    "transforms.route.type":
"org.apache.kafka.connect.transforms.RegexRouter",
    "transforms.route.regex": "([^.]*)\\.([^.]*)\\.([^.]*)",
    "transforms.route.replacement": "$3"
  }
}
```

[jdbc source connector 설정](#) : JDBC Connect 대상에서 Source를 받아옵니다

```
vi jdbc-source.json
```

```
{
  "name": "jdbc-source",
  "config": {
    "connector.class": "io.confluent.connect.jdbc.JdbcSourceConnector",
    "tasks.max": "1",
    "connection.url": "jdbc:mysql://{RDS URL}:3306/test",
    "mode": "timestamp",
    "connection.user" : "admin",
    "connection.password" : "Bespın12!",
    "table.whitelist" : "test.todo",
    "timestamp.column.name": "EVENT_TIME",
    "topic.prefix": "mytest."
  }
}
```

[jdbc sink connector 설정](#) : Target을 JDBC Connect 대상으로 연결합니다

```
vi jdbc-sink.json
```

```
{
  "name": "jdbc-sink",
  "config": {
    "connector.class": "io.confluent.connect.jdbc.JdbcSinkConnector",
    "tasks.max": "1",
    "topics": "mytest",
    "connection.url": "jdbc:mysql://{RDS_URL}:3306/{target DB}?
user=admin&password=Bespin12!",
    "transforms.unwrap.type": "io.debezium.transforms.UnwrapFromEnvelope",
    "auto.create": "true",
    "insert.mode": "upsert",
    "transforms": "unwrap",
    "pk.fields": "id",
    "pk.mode": "record_value"
  }
}
```

**s3 sink connector 설정** : Target이 S3가 됩니다

```
vi s3-sink.json
```

```
{
  "name": "s3-sink",
  "config": {
    "connector.class": "io.confluent.connect.s3.S3SinkConnector",
    "tasks.max": "1",
    "topics": "mytest.todo",
    "s3.region": "us-west-2",
    "s3.bucket.name": "{Bucket_Name}",
    "s3.compression.type": "gzip",
    "s3.part.size": "5242880",
    "flush.size": "1",
    "storage.class": "io.confluent.connect.s3.storage.S3Storage",
    "format.class": "io.confluent.connect.s3.format.json.JsonFormat",
    "schema.generator.class":
"io.confluent.connect.storage.hive.schema.DefaultSchemaGenerator",
    "partitioner.class":
"io.confluent.connect.storage.partitioners.TimeBasedPartitioner",
    "partition.duration.ms": "3600000",
    "path.format": "YYYY-MM-dd",
    "locale": "KR",
    "timezone": "UTC",
    "schema.compatibility": "NONE"
  }
}
```

**Connect와 Connector 설정**이 모두 완료 되었습니다. 이제 Connect를 실행시켜보겠습니다.

```
# Connect 정보가 있는 dir로 이동
cd ~/kafka/kafka_2.12-2.2.1/config/

# Connect 실행
../bin/connect-distributed.sh connect.properties
```

새 터미널을 실행 시킵니다. 방금 실행시킨 Connect에 Connector를 실행 시킬겁니다.

```
# Connector 정보가 있는 dir로 이동
cd ~/kafka/kafka_2.12-2.2.1/config/

# jdbc source connector를 등록할 예정입니다.
# jdbc source connector는 source db를 토픽으로 만들어 해당 토픽으로 생성하고 해당 토픽에 데이터를 넘깁니다.
# 토픽 이름 : {topic.prefix}{Table}
# 예 : mytest.todo
# 따라서 위의 jdbc-source.json 기준으로 mytest.todo 라는 토픽에 데이터가 전송됩니다.
# 하지만 msk는 topic을 자동으로 생성해주지 않습니다.(config를 변경해도 생성 x)
# 따라서 해당 토픽을 미리 생성합니다.
# 생성하지 않을 시 다음과 같은 에러 메시지가 발생합니다.
# [2020-12-15 04:23:02,187] WARN [Producer clientId=producer-4] Error while
fetching metadata with correlation id 142 :
{mytest.todo=INVALID_REPLICATION_FACTOR}
(org.apache.kafka.clients.NetworkClient:1031)

# 토픽 생성
../bin/kafka-topics.sh --create --zookeeper "z-1.cjm-cluster.kht7sx.c7.kafka.us-
west-2.amazonaws.com:2181,z-3.cjm-cluster.kht7sx.c7.kafka.us-west-
2.amazonaws.com:2181,z-2.cjm-cluster.kht7sx.c7.kafka.us-west-
2.amazonaws.com:2181" --replication-factor 2 --partitions 1 --topic mytest.todo

# connector 등록 (jdbc source)
curl -i -X POST -H "Accept:application/json" -H "Content-Type:application/json"
http://localhost:8083/connectors/ -d @jdbc-source.json

# connector가 등록되고 Table 정보가 전송됩니다.

# 그 외에 알면 좋은 command
# ===
# connector 삭제
curl -X DELETE localhost:8083/connectors/{connector name}

# connector list
curl http://localhost:8083/connector-plugins | python -m json.tool

# 등록된 connector list
curl http://localhost:8083/connectors | python -m json.tool
```

## 5. Application jar

다른 EC2 서버에 접속합니다. [aws에서 제공하는 코드](#)를 살짝 수정하여 jar 로 build할 예정입니다.

### 사전 작업

```
# git 설치
sudo yum install -y git

# 소스 코드 다운
git clone https://github.com/aws-samples/amazon-kinesis-data-analytics-java-
examples
```

```

# aws kinesis data analytics flink application을 만들기 위해서는 maven 및 java 11이
설치되어야 합니다.
# change the directory to /opt folder.
cd /opt

# install maven
wget https://downloads.apache.org/maven/maven-3/3.6.3/binaries/apache-maven-
3.6.3-bin.tar.gz

# unzip tar
sudo tar -xvzf apache-maven-3.6.3-bin.tar.gz

# Edit the /etc/environment file and add the following environment variable:
sudo nano /etc/environment
M2_HOME="/opt/apache-maven-3.6.3"

# After the modification, press Ctrl + O to save the changes and Ctrl + X to exit
nano.

# Update the mvn command:
sudo update-alternatives --install "/usr/bin/mvn" "mvn" "/opt/apache-maven-
3.6.3/bin/mvn" 0
sudo update-alternatives --set mvn /opt/apache-maven-3.6.3/bin/mvn

# Add Bash completion to mvn so that you can complete complex Maven commands by
hitting Tab multiple times.
sudo wget https://raw.githubusercontent.com/dimaj/maven-bash-
completion/master/bash_completion.bash --output-document
/etc/bash_completion.d/mvn

# Logout and login to the computer and check the Maven version using the
following command.
mvn --version

# Java install
cd ~/
sudo yum install -y java-11-openjdk-devel

```

## application 생성 Jar

```

# KafkaConnectors 프로젝트를 사용할 예정입니다.
cd ~/amazon-kinesis-data-analytics-java-
examples/KafkaConnectors/src/main/java/com/amazonaws/services/kinesisanalytics/

# code 수정
vi KafkaGettingStartedJob.java
package com.amazonaws.services.kinesisanalytics;

```

```

import com.amazonaws.services.kinesisanalytics.runtime.KinesisAnalyticsRuntime;
import org.apache.flink.api.common.serialization.SimpleStringSchema;
import org.apache.flink.streaming.api.datastream.DataStream;
import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
import org.apache.flink.streaming.connectors.kafka.FlinkKafkaConsumer;
import org.apache.flink.streaming.connectors.kafka.FlinkKafkaProducer;

```

```

import
org.apache.flink.streaming.connectors.kafka.internals.KeyedSerializationSchemawr
apper;
import org.apache.flink.streaming.util.serialization.KeyedSerializationSchema;
import
org.apache.flink.streaming.api.functions.sink.filesystem.StreamingFileSink;
import org.apache.flink.api.common.serialization.SimpleStringEncoder;
import org.apache.flink.api.common.functions.FlatMapFunction;
import org.apache.flink.api.java.tuple.Tuple2;
import org.apache.flink.streaming.api.windowing.time.Time;
import org.apache.flink.core.fs.Path;
import org.apache.flink.util.Collector;
import java.util.Properties;
import java.io.IOException;
import java.util.Map;
import java.util.Properties;

public class KafkaGettingStartedJob {

    private static final String region = "{region-name}";
    private static final String s3SinkPath = "s3a://{bucket-name}/data/";
    private static DataStream<String>
createKafkaSourceFromApplicationProperties(StreamExecutionEnvironment env)
throws IOException {
    Map<String, Properties> applicationProperties =
KinesisAnalyticsRuntime.getApplicationProperties();
    return env.addSource(new FlinkKafkaConsumer<>((String)
applicationProperties.get("kafkaSource").get("topic"),
        new SimpleStringSchema(),
applicationProperties.get("kafkaSource")));
    }

    private static StreamingFileSink<String> createS3SinkFromStaticConfig() {
        final StreamingFileSink<String> sink = StreamingFileSink
            .forRowFormat(new Path(s3SinkPath), new
SimpleStringEncoder<String>("UTF-8"))
            .build();
        return sink;
    }

    public static void main(String[] args) throws Exception {
        // set up the streaming execution environment
        final StreamExecutionEnvironment env =
StreamExecutionEnvironment.getExecutionEnvironment();
        DataStream<String> input =
createKafkaSourceFromApplicationProperties(env);

        // Add sink
input.flatMap(new Tokenizer()) // Tokenizer for generating words
    .keyBy(0) // Logically partition the stream for each word
    .timeWindow(Time.minutes(1)) // Tumbling window definition
    .sum(1) // Sum the number of words per partition
    .map(value -> value.f0 + " count: " + value.f1.toString() +
"\n")
    .addSink(createS3SinkFromStaticConfig());

        env.execute("Flink S3 Streaming Sink Job");
    }
}

```

```

    }

    public static final class Tokenizer
        implements FlatMapFunction<String, Tuple2<String, Integer>> {

        @Override
        public void flatMap(String value, Collector<Tuple2<String, Integer>>
out) {
            String[] tokens = value.toLowerCase().split("\\W+");
            for (String token : tokens) {
                if (token.length() > 0) {
                    out.collect(new Tuple2<>(token, 1));
                }
            }
        }
    }
}

```

```

# code 수정 후 프로젝트 dir로 이동
cd ~/amazon-kinesis-data-analytics-java-examples/KafkaConnectors/

# build jar
mvn package -Dflink.version=1.11.1

# jar 파일 확인
cd ~/amazon-kinesis-data-analytics-java-examples/KafkaConnectors/target/

# KafkaGettingStartedJob-1.0.jar 가 생성됩니다.
# 해당 jar를 s3 bucket upload합니다.
aws s3 cp KafkaGettingStartedJob-1.0.jar s3://{bucket-name}

```

## 6. IAM Role

### 역할 생성

IAM에서 좌측 **역할** 탭을 클릭하여 **역할 만들기**를 합니다.

신뢰할 수 있는 유형의 개체 선택 : AWS 서비스

사용 사례 선택 : Kinesis > Kinesis Analytics

다음 : 권한

연결해야 하는 정책 리스트

- AmazonEC2FullAccess (vpc 접근 정책)
- AmazonS3FullAccess (s3 data put 정책)
- AmazonKinesisFullAccess (kinesis 접근 정책)
- CloudWatchLogsFullAccess (log 접근 정책)
- AmazonMSKFullAccess (msk 접근 정책)

역할 이름 : `analytics-role`

## 7. Kinesis Data Analytics

KDA는 반드시 private subnet에 구성되어야 합니다.

MSK 콘솔에서 KDA Flink 생성





- 애플리케이션 이름 : `f1ink-msk`
- 액세스 권한 : Kinesis Data Analytics에서 위임할 수 있는 IAM 역할 중에서 선택
- IAM 역할 : `{위에서 Role}`
- 템플릿 : `개발`

## 애플리케이션 생성

애플리케이션이 생성되면 `구성` 을 선택합니다.

- Amazon S3 버킷 : 상단에서 생성한 bucket
- Amazon S3 객체의 경로 : `{jar file이 있는 경로 및 jar file 이름}`
- 속성 `그룹 추가`
- 그룹 ID : `KafkaSource`
- 키-값 페어:

키	값
<code>bootstrap.servers</code>	<code>{Broker - ip}</code>
<code>group.id</code>	<code>{consumer group id}</code>
<code>topic</code>	<code>{topic-name} ex) mytest.todo</code>

- vpc 연결 : 서브넷을 private으로 변경합니다.

`업데이트` 후 `실행`

실행 후에 생성 RDS에 데이터를 insert하면 S3에 `{bucket}/data/`에 저장됩니다.