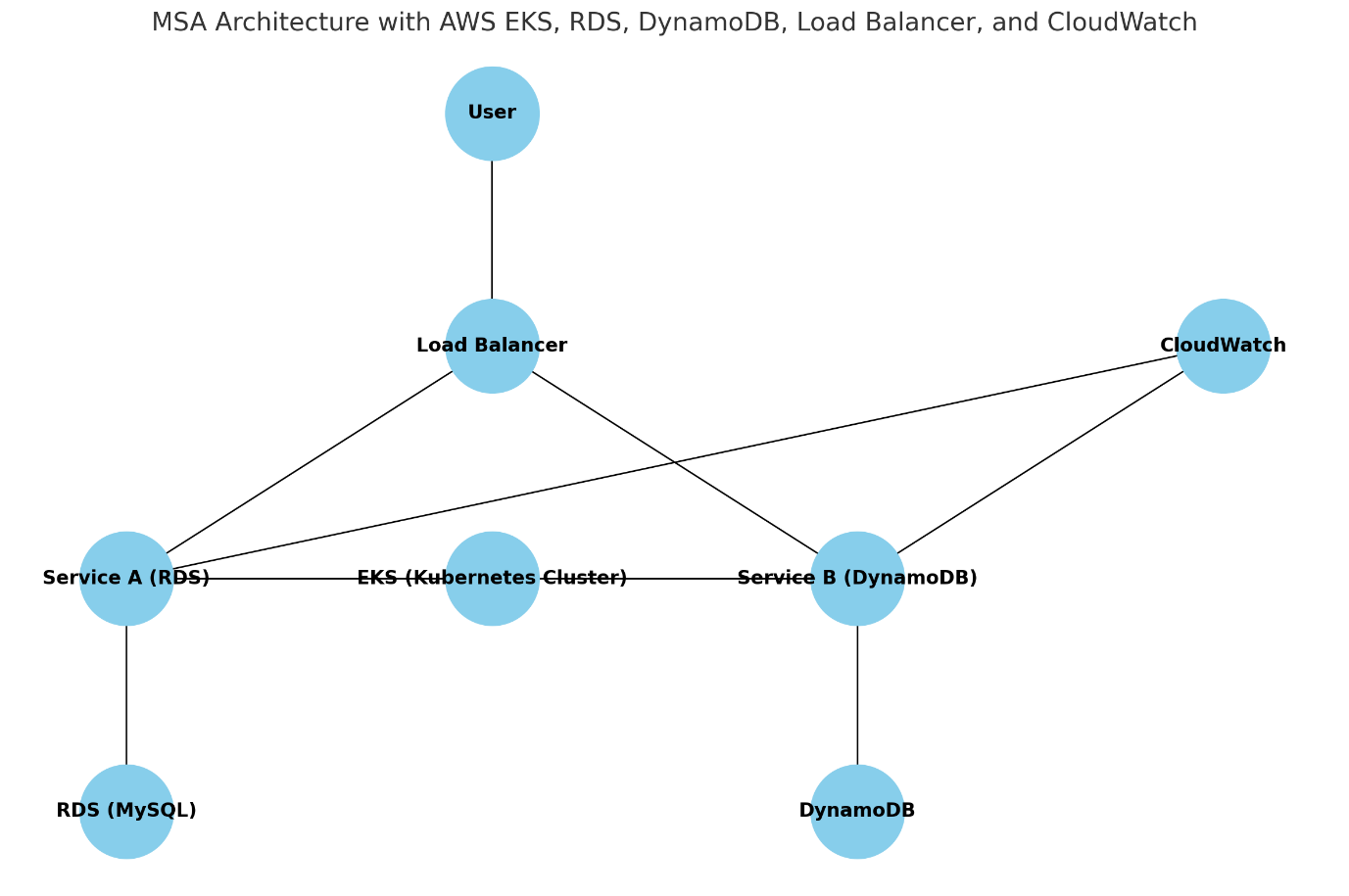
**AWS 클라우드 컴퓨팅 분산 저장 기술 실습**

**1. 개요**

본 실습에서는 AWS EKS(Elastic Kubernetes Service)를 활용하여 마이크로서비스 아키텍처(MSA) 환경을 구축하고, 각 마이크로서비스가 개별 데이터베이스를 가지도록 구현한다. 또한 지속적 통합(CI) 및 지속적 전달(CD) 파이프라인을 구축하고, 모니터링 및 로깅을 설정한다.

**실습 목표**

* AWS EKS 클러스터 구축
* RDS 및 DynamoDB를 활용한 분산 저장 설계
* 샘플 마이크로서비스 애플리케이션 배포 및 데이터 저장/조회 실습
* CloudWatch를 통한 모니터링 및 로깅 설정



**AWS ECR(Amazon Elastic Container Registry)**

**1. 개요**

Amazon ECR은 AWS에서 제공하는 완전관리형 Docker 컨테이너 이미지 저장소이다. 사용자는 자체 Docker 이미지를 빌드하고 이를 Amazon ECR에 저장하여 Amazon ECS, Amazon EKS, AWS Lambda 등 다양한 AWS 서비스에서 사용할 수 있다. ECR은 고가용성, 보안, 확장성을 보장하며, AWS IAM과 연동하여 세밀한 접근 제어가 가능하다.

**2. ECR 구성 요소**

| **구성 요소** | **설명** |
| --- | --- |
| **리포지토리(Repository)** | 컨테이너 이미지를 저장하는 논리적 공간이다. Git의 저장소처럼 이미지 버전을 관리할 수 있다. |
| **이미지(Image)** | docker build 명령으로 생성한 결과물이다. 특정 태그(tag)를 통해 다양한 버전으로 관리된다. |
| **URI** | 이미지 업로드 및 다운로드에 사용하는 고유 주소이다. 형식은 계정번호.dkr.ecr.리전.amazonaws.com/리포지토리명이다. |

**3. ECR 사용 흐름**

**3.1 Docker 이미지 생성**

docker build -t myapp .

**3.2 ECR 리포지토리 생성**

aws ecr create-repository --repository-name myapp

**3.3 ECR 로그인 (인증 토큰 기반)**

aws ecr get-login-password \

| docker login --username AWS \

--password-stdin 123456789012.dkr.ecr.us-east-1.amazonaws.com

**3.4 이미지 태깅**

docker tag myapp:latest 123456789012.dkr.ecr.us-east-1.amazonaws.com/myapp:latest

**3.5 이미지 푸시**

docker push 123456789012.dkr.ecr.us-east-1.amazonaws.com/myapp:latest

**3.6 AWS 서비스에서 이미지 사용**

Amazon ECS, EKS, Lambda 등의 서비스에서 위 URI를 통해 이미지를 가져와 컨테이너를 실행할 수 있다.

**4. 보안 및 권한 관리**

**4.1 IAM 정책 예시 (Push/Pull 허용)**

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow",

"Action": [

"ecr:GetAuthorizationToken",

"ecr:BatchCheckLayerAvailability",

"ecr:GetDownloadUrlForLayer",

"ecr:BatchGetImage",

"ecr:PutImage",

"ecr:InitiateLayerUpload",

"ecr:UploadLayerPart",

"ecr:CompleteLayerUpload"

],

"Resource": "\*"

}

]

}

**4.2 리포지토리 정책 설정 (예: 읽기 전용 공개)**

{

"Version": "2008-10-17",

"Statement": [

{

"Sid": "AllowPull",

"Effect": "Allow",

"Principal": "\*",

"Action": [

"ecr:GetDownloadUrlForLayer",

"ecr:BatchGetImage",

"ecr:BatchCheckLayerAvailability"

]

}

]

}

**5. ECR Public vs Private**

| **구분** | **설명** |
| --- | --- |
| **Private ECR** | 기본값이며, AWS 계정 소유자만 접근 가능하다. |
| **Public ECR** | 누구나 Pull할 수 있도록 공개된 이미지 저장소이다. Amazon Linux 등의 공식 이미지가 등록되어 있다. |

**6. 이미지 스캔 기능**

ECR은 이미지 푸시 시 자동으로 취약점 스캔을 수행할 수 있다. 이는 Amazon Inspector를 통해 작동하며, 활성화 설정이 필요하다.

**6.1 스캔 활성화 리포지토리 생성 예시**

aws ecr create-repository \

--repository-name myapp \

--image-scanning-configuration scanOnPush=true

**6.2 스캔 결과 확인**

aws ecr describe-image-scan-findings \

--repository-name myapp \

--image-id imageTag=latest

**7. CloudFormation 예시 (ECR 리포지토리 생성)**

Resources:

MyECRRepository:

Type: AWS::ECR::Repository

Properties:

RepositoryName: myapp

ImageScanningConfiguration:

scanOnPush: true

RepositoryPolicyText:

Version: "2008-10-17"

Statement:

- Sid: AllowPull

Effect: Allow

Principal: "\*"

Action:

- ecr:GetDownloadUrlForLayer

- ecr:BatchGetImage

- ecr:BatchCheckLayerAvailability

**8. 요약**

* ECR은 AWS의 컨테이너 이미지 저장소 서비스이다.
* Docker CLI를 통해 이미지를 생성하고, 태깅 후 푸시하면 저장된다.
* AWS IAM과 연동하여 세밀한 권한 설정이 가능하다.
* 이미지 취약점 스캔, 퍼블릭/프라이빗 리포지토리 기능도 지원한다.
* Amazon ECS, EKS, Lambda 등과 쉽게 통합된다.

**9. 참고 명령어 정리**

# 리포지토리 생성

aws ecr create-repository --repository-name myapp

# 로그인

aws ecr get-login-password | docker login --username AWS --password-stdin [ECR-URI]

# 이미지 태깅

docker tag myapp:latest [ECR-URI]/myapp:latest

# 이미지 푸시

docker push [ECR-URI]/myapp:latest

# 이미지 스캔 결과 확인

aws ecr describe-image-scan-findings --repository-name myapp --image-id imageTag=latest

**AWS EKS(Amazon Elastic Kubernetes Service)**

**1. 개요**

Amazon EKS는 AWS에서 제공하는 완전관리형 Kubernetes 서비스이다. 사용자 대신 Kubernetes 클러스터의 제어 플레인(Control Plane)을 관리해주며, 고가용성과 보안, 확장성을 제공한다. EKS를 사용하면 표준 Kubernetes API를 그대로 사용하면서 AWS 인프라의 이점을 활용할 수 있다.

**2. EKS 구성 요소**

| **구성 요소** | **설명** |
| --- | --- |
| **클러스터(Cluster)** | 컨트롤 플레인과 워커 노드로 구성된다. |
| **노드 그룹(Node Group)** | EKS 클러스터에 연결된 EC2 인스턴스 집합이다. 관리형 또는 자가 관리형으로 운영할 수 있다. |
| **Pod** | Kubernetes의 기본 실행 단위이며, 컨테이너 하나 또는 여러 개를 포함할 수 있다. |
| **노드(Node)** | EKS 클러스터에서 Pod를 실행하는 실제 서버(EC2 인스턴스 또는 Fargate)이다. |
| **Kubelet** | 각 노드에서 실행되며, Pod의 실행과 상태를 관리한다. |

**3. EKS 클러스터 생성 절차**

**3.1 EKS 클러스터 생성 (CLI)**

# 클러스터 생성

aws eks create-cluster \

--name my-cluster \

--region us-east-1 \

--kubernetes-version 1.29 \

--role-arn arn:aws:iam::123456789012:role/EKSClusterRole \

--resources-vpc-config subnetIds=subnet-xxx,securityGroupIds=sg-xxx

**3.2 관리형 노드 그룹 생성**

aws eks create-nodegroup \

--cluster-name my-cluster \

--nodegroup-name my-nodes \

--subnets subnet-xxx \

--node-role arn:aws:iam::123456789012:role/EKSNodeRole \

--scaling-config minSize=1,maxSize=3,desiredSize=2

**3.3 kubectl 연결 설정**

aws eks update-kubeconfig --region us-east-1 --name my-cluster

**4. EKS의 노드 옵션**

| **노드 종류** | **설명** |
| --- | --- |
| **관리형 노드 그룹(Managed Node Group)** | AWS가 EC2 인스턴스 생성 및 라이프사이클을 관리한다. |
| **자가 관리형 노드(Self-managed Node)** | 사용자가 직접 EC2 인스턴스를 구성하고 클러스터에 조인시킨다. |
| **Fargate 노드** | 서버리스 환경에서 Pod를 실행한다. EC2 인스턴스 관리가 필요 없다. |

**5. EKS 보안 설정**

* IAM 역할 기반 접근 제어(IAM-RBAC 연동)
* aws-auth ConfigMap을 수정하여 사용자/역할을 클러스터에 등록
* Security Group과 VPC 서브넷을 통한 네트워크 제어 가능

**예시: aws-auth ConfigMap 수정**

kubectl edit configmap aws-auth -n kube-system

mapRoles:

- rolearn: arn:aws:iam::123456789012:role/EKSNodeRole

username: system:node:{{EC2PrivateDNSName}}

groups:

- system:bootstrappers

- system:nodes

**6. EKS와 연동되는 주요 도구**

| **도구** | **설명** |
| --- | --- |
| **kubectl** | Kubernetes 클러스터 제어 CLI 도구 |
| **eksctl** | EKS 클러스터를 빠르게 설정하는 CLI 툴 (eksctl.io) |
| **Helm** | Kubernetes 패키지 매니저로, 애플리케이션 배포를 템플릿화함 |
| **k9s** | TUI 기반 Kubernetes 클러스터 모니터링 도구 |

**7. 애플리케이션 배포 예시**

**7.1 간단한 NGINX 배포**

kubectl create deployment nginx --image=nginx

**7.2 서비스 노출**

kubectl expose deployment nginx --port=80 --type=LoadBalancer

**7.3 서비스 확인**

kubectl get svc

**8. CloudFormation 예시**

Resources:

MyEKSCluster:

Type: AWS::EKS::Cluster

Properties:

Name: my-cluster

RoleArn: arn:aws:iam::123456789012:role/EKSClusterRole

ResourcesVpcConfig:

SubnetIds:

- subnet-abc

- subnet-def

SecurityGroupIds:

- sg-123

**9. 요약**

* Amazon EKS는 완전관리형 Kubernetes 서비스이다.
* EC2, Fargate 기반 노드에서 Kubernetes Pod를 실행할 수 있다.
* IAM, VPC, Security Group 등 AWS 인프라와 긴밀하게 통합된다.
* eksctl, kubectl, Helm 등의 도구와 함께 사용하는 것이 일반적이다.

**10. 참고 명령어 정리**

# 클러스터 생성

aws eks create-cluster --name my-cluster --role-arn ... --resources-vpc-config ...

# 노드 그룹 생성

aws eks create-nodegroup --cluster-name my-cluster --nodegroup-name my-nodes ...

# kubeconfig 설정

aws eks update-kubeconfig --region us-east-1 --name my-cluster

# nginx 배포

kubectl create deployment nginx --image=nginx

# 서비스 노출

kubectl expose deployment nginx --port=80 --type=LoadBalancer

**2. 실습 환경 준비**

**필요한 AWS 서비스**

* AWS EKS (Kubernetes 클러스터)
* AWS RDS (MySQL 또는 PostgreSQL)
* AWS DynamoDB (NoSQL 데이터베이스)
* AWS CloudWatch

**사전 요구 사항**

* AWS 계정 및 IAM 권한 설정
* AWS CLI 및 kubectl 설치
* eksctl을 사용한 EKS 클러스터 생성
* 아래 workshop에서 미리 cloud9 환경 설정과 EKS 설치 필요함

https://catalog.us-east-1.prod.workshops.aws/workshops/46236689-b414-4db8-b5fc-8d2954f2d94a/ko-KR/eks/10-install

**3. EKS 클러스터 생성**

**1) eksctl을 사용한 클러스터 생성 : 리전은 버지니아북부(us-east-1)로 사용**

먼저, AWS CLI가 정상적으로 설치되었는지 확인합니다.(cloud9에서는 불필요)

aws --version

cloud9에서 아래 명령 실행

touch create-eks-in-cloud9-vpc.sh

에디터로 create-eks-in-cloud9-vpc.sh 파일 아래 내용으로 수정

|  |
| --- |
| #!/bin/bash  set -e  CLUSTER\_NAME="my-eks-cluster"  NODEGROUP\_NAME="my-node-group"  REGION="us-east-1"  INSTANCE\_TYPE="t3.medium"  NODES=2  # 1. Cloud9 인스턴스 ID  INSTANCE\_ID=$(curl -s http://169.254.169.254/latest/meta-data/instance-id)  VPC\_ID=$(aws ec2 describe-instances \  --instance-ids "$INSTANCE\_ID" \  --query "Reservations[0].Instances[0].VpcId" \  --region "$REGION" \  --output text)  echo "📡 VPC: $VPC\_ID"  # 2. 퍼블릭 서브넷 중 EKS 지원 AZ만 필터링  ALLOWED\_ZONES="us-east-1a us-east-1b us-east-1c us-east-1d us-east-1f"  echo "🔍 EKS 지원 AZ: $ALLOWED\_ZONES"  SUBNET\_IDS=$(aws ec2 describe-subnets \  --filters "Name=vpc-id,Values=$VPC\_ID" "Name=map-public-ip-on-launch,Values=true" \  --query "Subnets[?AvailabilityZone=='us-east-1a' || AvailabilityZone=='us-east-1b' || AvailabilityZone=='us-east-1c' || AvailabilityZone=='us-east-1d' || AvailabilityZone=='us-east-1f'].SubnetId" \  --region "$REGION" \  --output text)  SUBNET\_COUNT=$(echo "$SUBNET\_IDS" | wc -w)  if [ "$SUBNET\_COUNT" -lt 2 ]; then  echo "❌ EKS가 지원하는 AZ에서 퍼블릭 서브넷이 2개 이상 필요합니다."  exit 1  fi  SUBNETS\_CSV=$(echo $SUBNET\_IDS | sed 's/ /,/g')  echo "✅ 사용할 서브넷 (EKS 지원 AZ): $SUBNETS\_CSV"  # 3. EKS 클러스터 생성  eksctl create cluster \  --name "$CLUSTER\_NAME" \  --region "$REGION" \  --nodegroup-name "$NODEGROUP\_NAME" \  --node-type "$INSTANCE\_TYPE" \  --nodes "$NODES" \  --vpc-public-subnets "$SUBNETS\_CSV"  echo "🎉 클러스터 생성 요청 완료!" |

아래 명령 실행

chmod +x create-eks-in-cloud9-vpc.sh

이제 스크립트를 사용해 EKS 클러스터를 생성합니다.(약 20~30분 이상 소요)

./create-eks-in-cloud9-vpc.sh

실행 화면

|  |
| --- |
| ec2-user:~/environment $ ./create-eks-in-cloud9-vpc.sh  📡 VPC: vpc-0576468abe5380cc5  🔍 EKS 지원 AZ: us-east-1a us-east-1b us-east-1c us-east-1d us-east-1f  ✅ 사용할 서브넷 (EKS 지원 AZ): subnet-098780a150cad141a,subnet-0f0516db92d02d85f,subnet-0a87c57722f28438d,subnet-0c3ed0e0187c18af5,subnet-0121828e6a581c088  2025-03-23 17:10:59 [ℹ] eksctl version 0.205.0  2025-03-23 17:10:59 [ℹ] using region us-east-1  2025-03-23 17:10:59 [✔] using existing VPC (vpc-0576468abe5380cc5) and subnets (private:map[] public:map[us-east-1a:{subnet-0f0516db92d02d85f us-east-1a 172.31.16.0/20 0 } us-east-1b:{subnet-0c3ed0e0187c18af5 us-east-1b 172.31.32.0/20 0 } us-east-1c:{subnet-0a87c57722f28438d us-east-1c 172.31.0.0/20 0 } us-east-1d:{subnet-0121828e6a581c088 us-east-1d 172.31.80.0/20 0 } us-east-1f:{subnet-098780a150cad141a us-east-1f 172.31.64.0/20 0 }])  2025-03-23 17:10:59 [!] custom VPC/subnets will be used; if resulting cluster doesn't function as expected, make sure to review the configuration of VPC/subnets  2025-03-23 17:10:59 [ℹ] nodegroup "my-node-group" will use "" [AmazonLinux2/1.30]  2025-03-23 17:10:59 [ℹ] using Kubernetes version 1.30  2025-03-23 17:10:59 [ℹ] creating EKS cluster "my-eks-cluster" in "us-east-1" region with managed nodes  2025-03-23 17:10:59 [ℹ] will create 2 separate CloudFormation stacks for cluster itself and the initial managed nodegroup  2025-03-23 17:10:59 [ℹ] if you encounter any issues, check CloudFormation console or try 'eksctl utils describe-stacks --region=us-east-1 --cluster=my-eks-cluster'  2025-03-23 17:10:59 [ℹ] Kubernetes API endpoint access will use default of {publicAccess=true, privateAccess=false} for cluster "my-eks-cluster" in "us-east-1"  2025-03-23 17:10:59 [ℹ] CloudWatch logging will not be enabled for cluster "my-eks-cluster" in "us-east-1"  2025-03-23 17:10:59 [ℹ] you can enable it with 'eksctl utils update-cluster-logging --enable-types={SPECIFY-YOUR-LOG-TYPES-HERE (e.g. all)} --region=us-east-1 --cluster=my-eks-cluster'  2025-03-23 17:10:59 [ℹ] default addons metrics-server, vpc-cni, kube-proxy, coredns were not specified, will install them as EKS addons  2025-03-23 17:10:59 [ℹ]  2 sequential tasks: { create cluster control plane "my-eks-cluster",  2 sequential sub-tasks: {  2 sequential sub-tasks: {  1 task: { create addons },  wait for control plane to become ready,  },  create managed nodegroup "my-node-group",  }  }  2025-03-23 17:10:59 [ℹ] building cluster stack "eksctl-my-eks-cluster-cluster"  2025-03-23 17:11:00 [ℹ] deploying stack "eksctl-my-eks-cluster-cluster"  2025-03-23 17:11:30 [ℹ] waiting for CloudFormation stack "eksctl-my-eks-cluster-cluster"  2025-03-23 17:12:00 [ℹ] waiting for CloudFormation stack "eksctl-my-eks-cluster-cluster"  2025-03-23 17:13:00 [ℹ] waiting for CloudFormation stack "eksctl-my-eks-cluster-cluster"  2025-03-23 17:14:00 [ℹ] waiting for CloudFormation stack "eksctl-my-eks-cluster-cluster"  2025-03-23 17:15:00 [ℹ] waiting for CloudFormation stack "eksctl-my-eks-cluster-cluster"  2025-03-23 17:16:00 [ℹ] waiting for CloudFormation stack "eksctl-my-eks-cluster-cluster"  2025-03-23 17:17:00 [ℹ] waiting for CloudFormation stack "eksctl-my-eks-cluster-cluster"  2025-03-23 17:18:00 [ℹ] waiting for CloudFormation stack "eksctl-my-eks-cluster-cluster"  2025-03-23 17:19:00 [ℹ] waiting for CloudFormation stack "eksctl-my-eks-cluster-cluster"  2025-03-23 17:19:01 [ℹ] creating addon: metrics-server  2025-03-23 17:19:01 [ℹ] successfully created addon: metrics-server  2025-03-23 17:19:02 [!] recommended policies were found for "vpc-cni" addon, but since OIDC is disabled on the cluster, eksctl cannot configure the requested permissions; the recommended way to provide IAM permissions for "vpc-cni" addon is via pod identity associations; after addon creation is completed, add all recommended policies to the config file, under `addon.PodIdentityAssociations`, and run `eksctl update addon`  2025-03-23 17:19:02 [ℹ] creating addon: vpc-cni  2025-03-23 17:19:02 [ℹ] successfully created addon: vpc-cni  2025-03-23 17:19:02 [ℹ] creating addon: kube-proxy  2025-03-23 17:19:03 [ℹ] successfully created addon: kube-proxy  2025-03-23 17:19:03 [ℹ] creating addon: coredns  2025-03-23 17:19:03 [ℹ] successfully created addon: coredns  2025-03-23 17:21:04 [ℹ] building managed nodegroup stack "eksctl-my-eks-cluster-nodegroup-my-node-group"  2025-03-23 17:21:04 [ℹ] deploying stack "eksctl-my-eks-cluster-nodegroup-my-node-group"  2025-03-23 17:21:04 [ℹ] waiting for CloudFormation stack "eksctl-my-eks-cluster-nodegroup-my-node-group"  2025-03-23 17:21:34 [ℹ] waiting for CloudFormation stack "eksctl-my-eks-cluster-nodegroup-my-node-group"  2025-03-23 17:22:26 [ℹ] waiting for CloudFormation stack "eksctl-my-eks-cluster-nodegroup-my-node-group" |

너무 오래 걸리면(20분이상) 다른 터미널에서 인스턴스 생성을 확인하고 다른 터미널 창에서 계속 진행한다

[참고] 오류 발생시 클러스터 삭제 명령(시간 수분 소요)

eksctl delete cluster --name my-eks-cluster --region us-east-1

**2) kubectl 및 클러스터 연결 확인**

EKS 클러스터가 정상적으로 생성되었는지 확인하기 위해 다음 명령어를 실행합니다.

*# 생성된 클러스터에 접근 가능 하도록 ~/.kube/config 파일을 수정(생략해도됨)*

aws eks --region us-east-1 update-kubeconfig --name my-eks-cluster

kubectl get nodes

아래와 같이 노드가 정상적으로 출력되면 클러스터가 준비된 것입니다.

(cloud9 EC2 인스턴스와 같은 네트워크(VPC)를 사용하는 노드가 생성된다)

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

**4. 데이터베이스 구성 (RDS 및 DynamoDB)**

**1) RDS(MySQL) 생성**

**강사 제공 “AWS RDS 사용 실습” 내용으로 미리 완료**

.

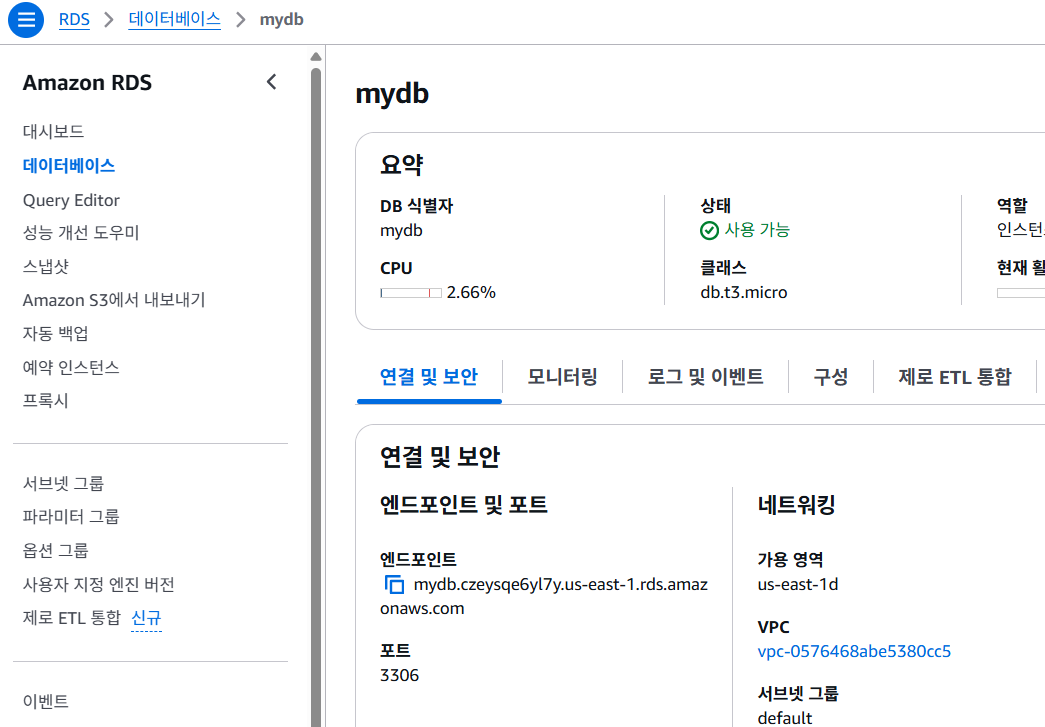
AWS 콘솔에서 RDS에 DB가 생성된 결과를 확인한다

텍스트, 폰트, 라인, 번호이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

mydb를 클릭해서 연결 및 보안안에 들어있는 엔드포인트 주소를 복사해온다

(예: mydb.czeysqe6yl7y.us-east-1.rds.amazonaws.com )



**2) DynamoDB 테이블 생성**

DynamoDB 테이블을 생성하려면 다음 명령어를 실행합니다.

aws dynamodb create-table --table-name MyDynamoTable --attribute-definitions AttributeName=ID,AttributeType=S --key-schema AttributeName=ID,KeyType=HASH --provisioned-throughput ReadCapacityUnits=5,WriteCapacityUnits=5

화면에 긴 텍스트가 출력되면 ‘q’를 입력하고 Enter를 쳐서 빠져나온다

AWS 콘솔에서 DynamoDB에 DB가 생성된 결과를 확인한다

텍스트, 라인, 폰트, 번호이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

**5. 마이크로서비스 애플리케이션 구축 및 배포**

**1) Flask 애플리케이션 생성**

먼저 프로젝트 디렉토리를 생성합니다.

mkdir msa-project && cd msa-project

Flask 애플리케이션을 위한 디렉토리를 생성하고 필요한 파일을 만듭니다.

mkdir service && cd service

touch app.py

**app.py** 파일을 클릭하여 편집기에서 다음과 같이 작성합니다.

(앞에서 복사한 RDS의 엔드포인트 주소를 host 변수의 값으로 넣는다)

(소스 편집기에서 따옴표와 같은 파이썬 소스 코드의 문법 오류를 확인하고 진행한다)

|  |
| --- |
| from flask import Flask, request, jsonify  import boto3  import pymysql  app = Flask(\_\_name\_\_)  # DB 접속 설정  db\_settings = {  "host": "mydb.czeysqe6yl7y.us-east-1.rds.amazonaws.com",  "port": 3306,  "user": "admin",  "password": "password1234",  "database": "testdb",  "charset": "utf8mb4"  }  # 초기화: 데이터베이스 및 테이블 생성  def init\_db():  try:  # DB 없을 경우 생성 (초기 접속은 mysql DB 사용)  conn = pymysql.connect(  host=db\_settings['host'],  port=db\_settings['port'],  user=db\_settings['user'],  password=db\_settings['password'],  database='mysql',  charset='utf8mb4',  autocommit=True  )  with conn.cursor() as cursor:  cursor.execute("CREATE DATABASE IF NOT EXISTS testdb;")  conn.close()  # 테이블 생성  conn = pymysql.connect(\*\*db\_settings)  with conn.cursor() as cursor:  cursor.execute("""  CREATE TABLE IF NOT EXISTS my\_table (  id INT PRIMARY KEY,  name VARCHAR(100),  age INT  );  """)  conn.close()  print("✅ RDS 초기화 완료")  except Exception as e:  print("❌ RDS 초기화 실패:", e)  # RDS 연결 (매 요청마다 열고 닫는 방식이 안정적)  def get\_db\_connection():  return pymysql.connect(\*\*db\_settings)  # DynamoDB 리소스  dynamodb = boto3.resource('dynamodb', region\_name='us-east-1')  table = dynamodb.Table('MyDynamoTable')  @app.route('/')  def home():  return "<h2>Flask 서버가 실행 중입니다. 😊<br>사용 가능한 엔드포인트: /store/dynamodb, /fetch/dynamodb, /store/rds, /fetch/rds</h2>"  # RDS에 저장  @app.route('/store/rds', methods=['POST'])  def store\_rds():  data = request.json  conn = get\_db\_connection()  try:  with conn.cursor() as cursor:  sql = "INSERT INTO my\_table (id, name, age) VALUES (%s, %s, %s)"  cursor.execute(sql, (data['id'], data['name'], data['age']))  conn.commit()  return jsonify({"message": "Data stored in RDS"}), 200  except Exception as e:  return jsonify({"error": str(e)}), 500  finally:  conn.close()  # RDS에서 조회  @app.route('/fetch/rds', methods=['GET'])  def fetch\_rds():  id = request.args.get('id')  conn = get\_db\_connection()  try:  with conn.cursor() as cursor:  cursor.execute("SELECT id, name, age FROM my\_table WHERE id = %s", (id,))  result = cursor.fetchone()  if result:  return jsonify({"id": result[0], "name": result[1], "age": result[2]}), 200  else:  return jsonify({"error": "No data found"}), 404  except Exception as e:  return jsonify({"error": str(e)}), 500  finally:  conn.close()  # DynamoDB에 저장  @app.route('/store/dynamodb', methods=['POST'])  def store\_dynamodb():  data = request.json  try:  table.put\_item(Item=data)  return jsonify({"message": "Data stored in DynamoDB"}), 200  except Exception as e:  return jsonify({"error": str(e)}), 500  # DynamoDB에서 조회  @app.route('/fetch/dynamodb', methods=['GET'])  def fetch\_dynamodb():  id = request.args.get('id')  try:  response = table.get\_item(Key={'ID': id})  return jsonify(response.get('Item', {})), 200  except Exception as e:  return jsonify({"error": str(e)}), 500  if \_\_name\_\_ == '\_\_main\_\_':  init\_db()  app.run(host='0.0.0.0', port=5000) |

**2) Docker 빌드 및 배포**

AWS ECR에 컨테이너를 빌드하고 푸시하는 과정을 포함합니다.

아래 명령 실행

touch Dockerfile requirements.txt deploy\_ecr.sh

**Dockerfile 작성 : 파일명은 Dockerfile**

|  |
| --- |
| FROM python:3.8  WORKDIR /app  COPY app.py requirements.txt ./  RUN pip install -r requirements.txt  CMD ["python", "app.py"] |

**requirement.txt 파일 작성**

|  |
| --- |
| flask  boto3  pymysql  gunicorn |

**deploy\_ecr.sh 파일 작성 : ECR 리포지토리 생성 및 도커 빌드 및 푸시 스크립트**

|  |
| --- |
| #!/bin/bash  # AWS ECR 리포지토리 생성  ecr\_repo=$(aws ecr create-repository --repository-name msa-service --query 'repository.repositoryUri' --output text)  echo "ECR Repository Created: $ecr\_repo"  # Docker 이미지 빌드 및 태깅  docker build -t msa-service .  docker tag msa-service:latest $ecr\_repo:latest  # AWS ECR 로그인 및 푸시  aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin $ecr\_repo  docker push $ecr\_repo:latest  echo "Docker image pushed to ECR: $ecr\_repo" |

**아래 명령 수행**

chmod +x deploy\_ecr.sh

./deploy\_ecr.sh

몇 분 기다리면 아래와 같은 출력을 얻는다

스크린샷, 텍스트, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

마지막줄에서 “891377038690.dkr.ecr.us-east-1.amazonaws.com” 부분을 복사하여

아래 deployment.yaml 파일 작성시에 넣는다

**아래 명령 수행**

touch deployment.yaml

**3) Kubernetes 배포 실행**

**deployment.yaml 파일 작성**

|  |
| --- |
| apiVersion: apps/v1  kind: Deployment  metadata:  name: msa-service  spec:  replicas: 2  selector:  matchLabels:  app: msa-service  template:  metadata:  labels:  app: msa-service  spec:  containers:  - name: msa-service  image: 891377038690.dkr.ecr.us-east-1.amazonaws.com/msa-service:latest  - containerPort: 80  protocol: TCP  ---  apiVersion: v1  kind: Service  metadata:  name: msa-service  spec:  type: LoadBalancer # 외부 접근이 가능하도록 설정  selector:  app: msa-service  ports:  - protocol: TCP  port: 80 # 외부에서 접근할 포트  targetPort: 80 # 컨테이너 내부 Flask 서비스 포트 |

**배포 실행**

kubectl apply -f deployment.yaml

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

이제 Docker 컨테이너를 빌드하고 AWS EKS에 배포가 완료되었습니다.

(잘못 생성시는 kubectl delete -f deployment.yaml 명령을 실행하여 삭제한다)

**다음 명령어로 만들어진 Deployment, ReplicaSet, Pod의 리스트를 확인합니다.**

kubectl get deployment,replicaset,pods

<출력화면>

텍스트, 스크린샷, 소프트웨어, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

**\* 아래와 같이 오류시 원인을 찾는 방법**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

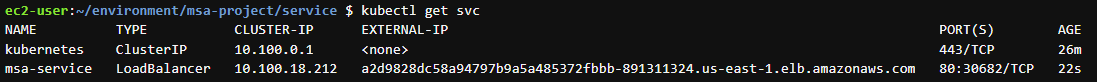
: kubectl logs pod/msa-service-68cd98f4dc를 실행한다

**다음 명령어로 Kubernetes Object의 리스트와 짧은 이름, ApiVersion 등을 확인할 수 있습니다.**

kubectl api-resources

**현재 배포된 서비스 목록 확인**

kubectl get svc

****

**EXTERNAL-IP 부분을 복사해 놓는다**

a2d9828dc58a94797b9a5a485372fbbb-891311324.us-east-1.elb.amazonaws.com

**EKS 워커 노드 IAM 역할에 권한 추가하기(DynamoDB 접근 권한 허용)**

**EKS 클러스터의 Role 이름 찾기**

aws iam list-roles \

--query "Roles[?contains(RoleName, 'eksctl-my-eks-cluster-nodegroup')].RoleName" \

--output text

<출력>

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

eksctl-my-eks-cluster-nodegroup-my-NodeInstanceRole-UzAOhoeJppIa 를 복사한다음

아래 명령에 넣어 실행한다

|  |
| --- |
| aws iam attach-role-policy --role-name eksctl-my-eks-cluster-nodegroup-my-NodeInstanceRole-UzAOhoeJppIa --policy-arn arn:aws:iam::aws:policy/AmazonDynamoDBFullAccess |

**6. RDS와 DynamoDB 데이터 읽고 쓰기 동작 테스트**

**EKS LoadBalancer 서비스의 외부 주소(URL) 를 자동으로 추출해 SERVICE\_URL 환경 변수에 저장**

|  |
| --- |
| export SERVICE\_URL=$(kubectl get svc msa-service \  -o jsonpath='{.status.loadBalancer.ingress[0].hostname}')  echo $SERVICE\_URL |

**[RDS 데이터 저장]**

|  |
| --- |
| curl -X POST http://$SERVICE\_URL/store/rds \  -H "Content-Type: application/json" \  -d '{"id": 1, "name": "Alice", "age": 30}' |

**[RDS 데이터 읽기]**

curl $SERVICE\_URL/fetch/rds?id=1

**[RDS 데이터 변경]**

|  |
| --- |
| curl -X PUT http://$SERVICE\_URL/update/rds \  -H "Content-Type: application/json" \  -d '{"id": 1, "name": "Tom", "age": 40}' |

|  |
| --- |
| curl -X PUT http://$SERVICE\_URL/update/rds \  -H "Content-Type: application/json" \  -d '{"id": 1, "name": "Alice", "age": 30}' |

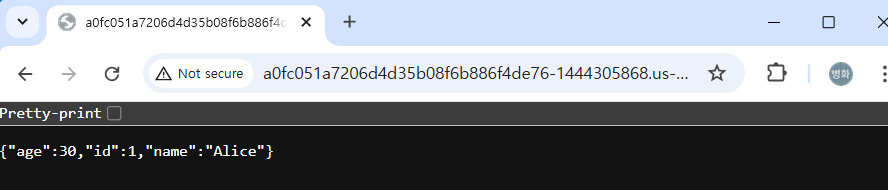
**[RDS 데이터 삭제]**

curl -X DELETE $SERVICE\_URL/delete/rds?id=1

[**외부 웹브라우저에서 RDS 데이터 읽기]**

앞에서 EXTERNAL-IP 부분을 복사해 놓은걸 사용한다

http://a0fc051a7206d4d35b08f6b886f4de76-1444305868.us-east-1.elb.amazonaws.com/fetch/rds?id=1



**[DynamoDB 데이터 저장]**

|  |
| --- |
| curl -X POST $SERVICE\_URL/store/dynamodb \  -H "Content-Type: application/json" \  -d '{"ID": "123", "name": "Alice", "age": 30}' |

**[DynamoDB데이터 읽기]**

curl $SERVICE\_URL/fetch/dynamodb?id=123

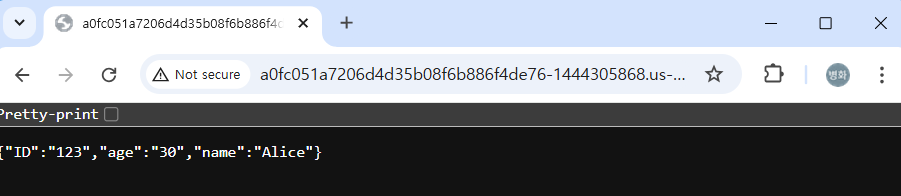
**<출력 결과>**

**텍스트, 스크린샷, 폰트, 라인이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.**

**[외부 웹브라우저에서 DynamoDB 데이터 읽기]**

http://a0fc051a7206d4d35b08f6b886f4de76-1444305868.us-east-1.elb.amazonaws.com/fetch/dynamodb?id=123

****

**[DynamoDB 데이터 변경]**

|  |
| --- |
| curl -X PUT $SERVICE\_URL/update/dynamodb \  -H "Content-Type: application/json" \  -d '{"ID": "123", "name": "John", "age": 40}' |

**변경 확인**

curl $SERVICE\_URL/fetch/dynamodb?id=123

**[DynamoDB 데이터 삭제]**

curl -X DELETE http://$SERVICE\_URL/delete/dynamodb?id=123

**삭제 확인**

curl $SERVICE\_URL/fetch/dynamodb?id=123

**텍스트, 스크린샷, 폰트, 라인이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.**

**7. 모니터링 및 로깅 실습 (CloudWatch )**

**✅ 목표**

* EKS에서 실행 중인 마이크로 서비스를 실시간으로 모니터링하고 로그를 수집.
* AWS CloudWatch를 이용해 **중앙 집중형 관측 환경**을 구축한다.

**🔹 1. CloudWatch 로그 수집 설정**

**1.1 EKS 클러스터에 로그 수집 활성화**

eksctl utils update-cluster-logging \

--region=us-east-1 \

--cluster=my-eks-cluster \

--enable-types=all \

--approve

약간 기다리면 완료

**1.2 CloudWatch 로그 그룹 확인**

aws logs describe-log-groups

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

/aws/eks/<클러스터명>/cluster 이름의 로그 그룹이 생성되어 있는 걸 알 수 있다

**✅ 로그 확인 방법**

EKS 클러스터의 **로그는 AWS CloudWatch Logs**로 전송됩니다. 따라서 확인은 CloudWatch에서 진행합니다.

**1. CloudWatch 콘솔에서 확인**

1. AWS 콘솔 접속 → CloudWatch → "로그 그룹(Log groups)" 메뉴로 이동
2. 로그 그룹 이름 중 다음 형식 확인:

/aws/eks/<클러스터 이름>/cluster

(실습에서의 예: /aws/eks/my-eks-cluster/cluster)

텍스트, 스크린샷, 번호, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

1. 로그 그룹 안에 들어가면 여러 로그 스트림이 있다.
   * kube-apiserver-...
   * audit-...
   * controllerManager-...
   * scheduler-...

텍스트, 스크린샷, 번호, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

1. 각 로그 스트림을 클릭하면 실시간으로 수집된 로그를 확인할 수 있다.

**텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.**

**2. CloudWatch 로그 그룹 확인**

아래 명령어로 생성된 로그 그룹을 확인할 수 있습니다.

aws logs describe-log-groups --log-group-name-prefix /aws/eks/my-eks-cluster --region us-east-1

**3. 로그 스트림 이름 확인 방법**

aws logs describe-log-streams \

--log-group-name /aws/eks/my-eks-cluster/cluster \

--region us-east-1 \

--order-by LastEventTime \

--descending \

--limit 5

**설명:**

* --order-by LastEventTime: 최근 로그가 있는 스트림부터 보여줌
* --descending: 최신 로그가 위로
* --limit: 최대 몇 개까지 보여줄지 (원하면 늘려도 됨)

**4. 로그 스트림 내용 확인 (예: kube-apiserver)**

aws logs get-log-events \

--log-group-name /aws/eks/my-eks-cluster/cluster \

--log-stream-name kube-apiserver-audit-2bf5efba5b4fd9f57cd1e22943cee6b4\

--region us-east-1 \

--limit 20

**5. 로그 설정 상태 확인 (eksctl 사용)**

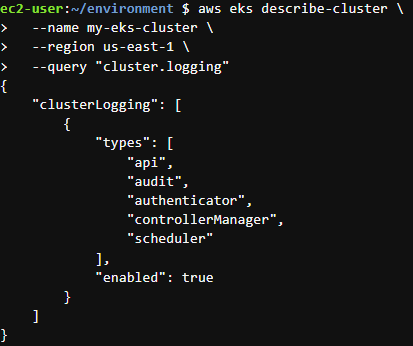
aws eks describe-cluster \

--name my-eks-cluster \

--region us-east-1 \

--query "cluster.logging"

현재 어떤 로그 타입이 활성화되어 있는지 보여줍니다.



**💡 지원되는 로그 타입**

* api
* audit
* authenticator
* controllerManager
* scheduler

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

**✅ EKS 컨트롤 플레인 로그 그룹 제거 방법**

1. **로그 수집 비활성화**

eksctl utils update-cluster-logging \

--region=us-east-1 \

--cluster=my-eks-cluster \

--disable-types=all \

--approve

1. **CloudWatch 로그 그룹 삭제**

aws logs delete-log-group --log-group-name /aws/eks/my-eks-cluster/cluster

**1.3 Flask 앱에서 로그 출력**

app.py에서 print() 또는 logging을 통해 로그를 출력하면 자동으로 CloudWatch에 수집됩니다.

|  |
| --- |
| import logging  logging.basicConfig(level=logging.INFO)  logging.info("Flask service started") |

**🔍 코드 설명**

1. **import logging**
   * 파이썬의 기본 로깅 모듈을 불러옵니다.
2. **logging.basicConfig(level=logging.INFO)**
   * 로깅 시스템의 기본 설정을 정의합니다.
   * level=logging.INFO는 "INFO 이상"의 로그만 출력하겠다는 뜻입니다.
     + 예: INFO, WARNING, ERROR, CRITICAL은 출력됨
     + DEBUG는 출력되지 않음
3. **logging.info("Flask service started")**
   * "Flask service started"라는 로그 메시지를 **INFO 레벨**로 출력합니다.

**💡 CloudWatch로 로그가 자동 수집되는 이유**

Flask 앱에서 print() 또는 logging을 통해 로그를 출력하면 **자동으로 CloudWatch에 수집됩니다**.

**✅ 조건:**

* Flask 앱이 **Amazon EC2 (또는 EKS, ECS)** 환경에서 실행되고 있어야 합니다.
* 그 인스턴스/컨테이너에는 **CloudWatch Logs Agent** 또는 **CloudWatch Container Insights** 설정이 되어 있어야 합니다.
* 보통 EKS에서는 **aws-for-fluent-bit DaemonSet**이 로그를 수집합니다.
* EC2에서는 /var/log/messages, /var/log/syslog, 또는 사용자 앱 로그 경로를 CloudWatch로 전송하도록 설정되어야 합니다.

**🔁 정리:**

* Flask 앱이 stdout/stderr 또는 logging 모듈로 로그를 출력하면,
* 이 출력은 기본적으로 터미널(콘솔)에 찍히지만,
* EC2나 EKS 환경에서는 **로그 수집 에이전트**가 그 출력을 캡처해서 CloudWatch Logs로 전송합니다.

**✅ 실제 사용 팁**

* 로그에 시간, 로그레벨, 파일명 등을 포함하고 싶다면 포맷을 지정한다:

logging.basicConfig(

level=logging.INFO,

format='%(asctime)s [%(levelname)s] %(message)s' )

**[ App 로깅 실습 ]**

로깅 지원 앱 소스 작성 : app-logging.py

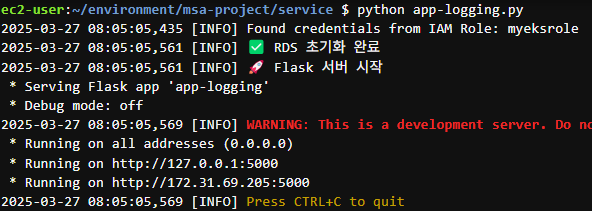
|  |
| --- |
| from flask import Flask, request, jsonify  import boto3  import pymysql  import logging  # 로깅 설정  logging.basicConfig(  level=logging.INFO,  format='%(asctime)s [%(levelname)s] %(message)s'  )  app = Flask(\_\_name\_\_)  # DB 접속 설정  db\_settings = {  "host": "mydb.czeysqe6yl7y.us-east-1.rds.amazonaws.com",  "port": 3306,  "user": "admin",  "password": "password1234",  "database": "testdb",  "charset": "utf8mb4"  }  # 초기화: 데이터베이스 및 테이블 생성  def init\_db():  try:  conn = pymysql.connect(  host=db\_settings['host'],  port=db\_settings['port'],  user=db\_settings['user'],  password=db\_settings['password'],  database='mysql',  charset='utf8mb4',  autocommit=True  )  with conn.cursor() as cursor:  cursor.execute("CREATE DATABASE IF NOT EXISTS testdb;")  conn.close()  conn = pymysql.connect(\*\*db\_settings)  with conn.cursor() as cursor:  cursor.execute("""  CREATE TABLE IF NOT EXISTS my\_table (  id INT PRIMARY KEY,  name VARCHAR(100),  age INT  );  """)  conn.close()  logging.info("✅ RDS 초기화 완료")  except Exception as e:  logging.error(f"❌ RDS 초기화 실패: {e}")  def get\_db\_connection():  return pymysql.connect(\*\*db\_settings)  dynamodb = boto3.resource('dynamodb', region\_name='us-east-1')  table = dynamodb.Table('MyDynamoTable')  @app.route('/')  def home():  logging.info("홈('/') 요청 수신")  return "<h2>Flask 서버가 실행 중입니다. 😊<br>사용 가능한 엔드포인트: /store/dynamodb, /fetch/dynamodb, /store/rds, /fetch/rds</h2>"  @app.route('/store/rds', methods=['POST'])  def store\_rds():  data = request.json  logging.info(f"[RDS 저장 요청] 데이터: {data}")  conn = get\_db\_connection()  try:  with conn.cursor() as cursor:  sql = "INSERT INTO my\_table (id, name, age) VALUES (%s, %s, %s)"  cursor.execute(sql, (data['id'], data['name'], data['age']))  conn.commit()  logging.info("✅ RDS 저장 성공")  return jsonify({"message": "Data stored in RDS"}), 200  except Exception as e:  logging.error(f"❌ RDS 저장 실패: {e}")  return jsonify({"error": str(e)}), 500  finally:  conn.close()  @app.route('/fetch/rds', methods=['GET'])  def fetch\_rds():  id = request.args.get('id')  logging.info(f"[RDS 조회 요청] ID: {id}")  conn = get\_db\_connection()  try:  with conn.cursor() as cursor:  cursor.execute("SELECT id, name, age FROM my\_table WHERE id = %s", (id,))  result = cursor.fetchone()  if result:  logging.info(f"✅ 조회 성공: {result}")  return jsonify({"id": result[0], "name": result[1], "age": result[2]}), 200  else:  logging.warning("⚠️ 조회 결과 없음")  return jsonify({"error": "No data found"}), 404  except Exception as e:  logging.error(f"❌ RDS 조회 실패: {e}")  return jsonify({"error": str(e)}), 500  finally:  conn.close()  @app.route('/update/rds', methods=['PUT'])  def update\_rds():  data = request.json  logging.info(f"[RDS 수정 요청] 데이터: {data}")  conn = get\_db\_connection()  try:  with conn.cursor() as cursor:  sql = "UPDATE my\_table SET name = %s, age = %s WHERE id = %s"  cursor.execute(sql, (data['name'], data['age'], data['id']))  conn.commit()  if cursor.rowcount == 0:  logging.warning("⚠️ 수정 대상 없음")  return jsonify({"error": "No record found to update"}), 404  logging.info("✅ RDS 수정 성공")  return jsonify({"message": "Data updated in RDS"}), 200  except Exception as e:  logging.error(f"❌ RDS 수정 실패: {e}")  return jsonify({"error": str(e)}), 500  finally:  conn.close()  @app.route('/delete/rds', methods=['DELETE'])  def delete\_rds():  id = request.args.get('id')  logging.info(f"[RDS 삭제 요청] ID: {id}")  conn = get\_db\_connection()  try:  with conn.cursor() as cursor:  sql = "DELETE FROM my\_table WHERE id = %s"  cursor.execute(sql, (id,))  conn.commit()  if cursor.rowcount == 0:  logging.warning("⚠️ 삭제 대상 없음")  return jsonify({"error": "No record found to delete"}), 404  logging.info("✅ RDS 삭제 성공")  return jsonify({"message": "Data deleted from RDS"}), 200  except Exception as e:  logging.error(f"❌ RDS 삭제 실패: {e}")  return jsonify({"error": str(e)}), 500  finally:  conn.close()  @app.route('/store/dynamodb', methods=['POST'])  def store\_dynamodb():  data = request.json  logging.info(f"[DynamoDB 저장 요청] 데이터: {data}")  try:  table.put\_item(Item=data)  logging.info("✅ DynamoDB 저장 성공")  return jsonify({"message": "Data stored in DynamoDB"}), 200  except Exception as e:  logging.error(f"❌ DynamoDB 저장 실패: {e}")  return jsonify({"error": str(e)}), 500  @app.route('/fetch/dynamodb', methods=['GET'])  def fetch\_dynamodb():  id = request.args.get('id')  logging.info(f"[DynamoDB 조회 요청] ID: {id}")  try:  response = table.get\_item(Key={'ID': id})  item = response.get('Item', {})  if item:  logging.info(f"✅ 조회 결과: {item}")  else:  logging.warning("⚠️ 조회 결과 없음")  return jsonify(item), 200  except Exception as e:  logging.error(f"❌ DynamoDB 조회 실패: {e}")  return jsonify({"error": str(e)}), 500  @app.route('/update/dynamodb', methods=['PUT'])  def update\_dynamodb():  data = request.json  logging.info(f"[DynamoDB 수정 요청] 데이터: {data}")  try:  table.update\_item(  Key={'ID': data['ID']},  UpdateExpression="SET #n = :name, age = :age",  ExpressionAttributeNames={'#n': 'name'},  ExpressionAttributeValues={  ':name': data['name'],  ':age': data['age']  }  )  logging.info("✅ DynamoDB 수정 성공")  return jsonify({"message": "Data updated in DynamoDB"}), 200  except Exception as e:  logging.error(f"❌ DynamoDB 수정 실패: {e}")  return jsonify({"error": str(e)}), 500  @app.route('/delete/dynamodb', methods=['DELETE'])  def delete\_dynamodb():  id = request.args.get('id')  logging.info(f"[DynamoDB 삭제 요청] ID: {id}")  try:  response = table.delete\_item(  Key={'ID': id},  ReturnValues='ALL\_OLD'  )  if 'Attributes' in response:  logging.info("✅ DynamoDB 삭제 성공")  return jsonify({"message": "Data deleted from DynamoDB"}), 200  else:  logging.warning("⚠️ 삭제 대상 없음")  return jsonify({"error": "No record found to delete"}), 404  except Exception as e:  logging.error(f"❌ DynamoDB 삭제 실패: {e}")  return jsonify({"error": str(e)}), 500  if \_\_name\_\_ == '\_\_main\_\_':  init\_db()  logging.info("🚀 Flask 서버 시작")  app.run(host='0.0.0.0', port=5000) |

**쿠버네티스로 배포 전에 먼저 앱서버를 로컬에서 실행시켜 로깅 메시지를 확인해본다**

Cloud9 터미널에서 소스가 들어 있는 경로 안에서 아래 명령 실행

python app-logging.py

<출력화면>



잘못된 RDS 접근 명령을 수행하면 [WARNING]이나 [ERROR] 메시지도 볼 수 있다

(예시: 모두 동일한 값으로 수정으로 시도하거나 동일한 ID로 저장을 시도할 경우)

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

<출력화면>

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

잘 실행되었으니 앱서버를 CTRL+C로 종료시키고 Kubernetes로 배포하여 CloudWatch로 로깅을 확인해보자

**먼저 Kubernetes 리소스(Deployment와 Service)를 삭제한다**

kubectl delete -f deployment.yaml

**ECR 리포지토리를 삭제한다**

aws ecr delete-repository --repository-name msa-service –force

Dockerfile을 아래와 같이 수정한다 :

|  |
| --- |
| FROM python:3.8  WORKDIR /app  COPY app-logging.py requirements.txt ./  RUN pip install -r requirements.txt  CMD ["python", "app-logging.py"] |

**컨테이너 생성 및 배포**

./deploy\_ecr.sh

kubectl apply -f deployment.yaml

**다음 명령어로 만들어진 Deployment, ReplicaSet, Pod의 리스트를 확인합니다.**

kubectl get deployment,replicaset,pods

<출력화면>

텍스트, 스크린샷, 소프트웨어, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

pod name을 복사해서 아래와 같이 실행한다

kubectl logs pod/msa-service-596cf6b865-qjtkn

다음과 같이 log 메시지를 확인할 수 있다

<출력화면>

**텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.**

Kubernetes 노드(EC2)의 파일 시스템에 임시로 저장된 로그를 읽어와서 출력한다

pod의 로그 저장 경로 : /var/log/pods/

Container의 로그 저장 경로 : /var/log/containers/

**pod 로그를 CloudWatch로 보내려면?**

**- Fluent Bit 설치 (AWS 권장 로그 수집기)**

<설치명령>

|  |
| --- |
| # 1. Helm 설치  curl -fsSL https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3 | bash  # 2. Helm 정상 설치 확인  helm version  # 3. 필요한 Helm 리포지터리 등록 및 업데이트  helm repo add aws-observability https://aws.github.io/eks-charts  helm repo update  # 4. Fluent Bit 설치 (EKS 파드 로그를 CloudWatch로 전송)  helm upgrade --install aws-for-fluent-bit aws-observability/aws-for-fluent-bit \  --namespace amazon-cloudwatch \  --create-namespace \  --set cloudWatch.region=us-east-1 \  --set cloudWatch.logGroupName=/aws/eks/fluentbit-cloudwatch/logs \  --set cloudWatch.logStreamPrefix=fluentbit |

**💡 Helm이란?**

* Helm은 Kubernetes의 **패키지 매니저**입니다.
* yum, apt, pip 같은 역할을 Kubernetes에서 합니다.
* 우리가 설치하려는 **Fluent Bit** 같은 구성 요소를 Helm chart를 통해 설치할 수 있습니다.

**설치 확인 명령**

helm list -A

**혹시 잘못 설치되었거나 다시 설치하려면 아래 삭제 명령을 수행한다**

helm uninstall aws-for-fluent-bit -n amazon-cloudwatch

**AWS 콘솔 🡪 CloudWatch 🡪 로그 🡪 로그 그룹**에서 아래와 같이

/aws/eks/fluentbit-cloudwatch/logs 로그 그룹이 생성된 걸 확인할 수 있다

텍스트, 폰트, 라인, 번호이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

**CloudWatch 로그 그룹이 잘못 생성시애는 아래 로그 그룹 삭제 명령을 수행한다**

aws logs delete-log-group --log-group-name /aws/eks/fluentbit-cloudwatch/logs

EKS 노드 IAM 역할에 CloudWatchLogsFullAccess 정책을 자동으로 붙이는 **전체 자동화 스크립트 : Cloud9 터미널에서 아래 명령을 복사하여 바로 실행해도 된다**

|  |
| --- |
| #!/bin/bash  # EKS 클러스터 이름  CLUSTER\_NAME="my-eks-cluster"  echo "🔍 EKS 클러스터 '$CLUSTER\_NAME'의 노드 IAM 역할 추적 중..."  # 1. 노드 인스턴스의 인스턴스 프로파일 이름 추출  INSTANCE\_PROFILE\_ARN=$(aws ec2 describe-instances \  --filters "Name=tag:eks:cluster-name,Values=$CLUSTER\_NAME" \  --query "Reservations[\*].Instances[\*].IamInstanceProfile.Arn" \  --output text | head -n 1)  if [ -z "$INSTANCE\_PROFILE\_ARN" ]; then  echo "❌ 인스턴스 프로파일을 찾을 수 없습니다."  exit 1  fi  INSTANCE\_PROFILE\_NAME=$(basename "$INSTANCE\_PROFILE\_ARN")  echo "✅ 인스턴스 프로파일 이름: $INSTANCE\_PROFILE\_NAME"  # 2. 인스턴스 프로파일에서 실제 IAM 역할 이름 추출  ROLE\_NAME=$(aws iam get-instance-profile \  --instance-profile-name "$INSTANCE\_PROFILE\_NAME" \  --query 'InstanceProfile.Roles[0].RoleName' \  --output text)  if [ -z "$ROLE\_NAME" ]; then  echo "❌ IAM 역할 이름을 찾을 수 없습니다."  exit 1  fi  echo "✅ 실제 IAM 역할 이름: $ROLE\_NAME"  # 3. IAM 역할에 정책 연결  echo "🔗 CloudWatchLogsFullAccess 정책 연결 중..."  aws iam attach-role-policy \  --role-name "$ROLE\_NAME" \  --policy-arn arn:aws:iam::aws:policy/CloudWatchLogsFullAccess  echo "🎉 완료: $ROLE\_NAME 역할에 CloudWatchLogsFullAccess 권한 부여됨!" |

**Cloud9에서 RDS 데이터 접근 명령어 수행**

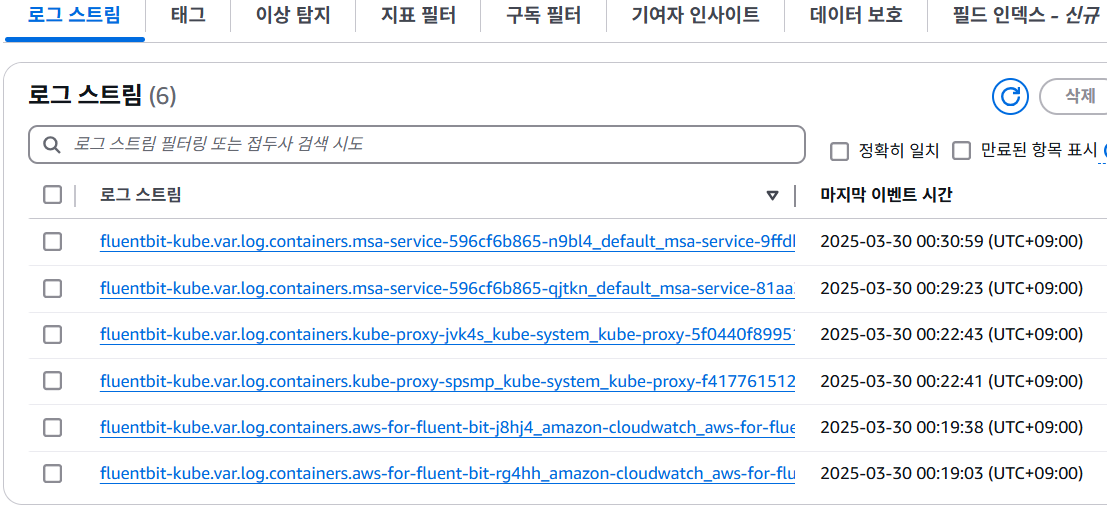
|  |
| --- |
| export SERVICE\_URL=$(kubectl get svc msa-service \  -o jsonpath='{.status.loadBalancer.ingress[0].hostname}')  echo $SERVICE\_URL  curl -X POST http://$SERVICE\_URL/store/rds \  -H "Content-Type: application/json" \  -d '{"id": 1, "name": "Alice", "age": 30}'  curl $SERVICE\_URL/fetch/rds?id=1 |

CloudWatch에 가서 우측상단에서 로깅할 시간을 현지 시간대(한국시간)로 설정해준다

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

**CloudWatch 🡪 로그 🡪 로그 그룹**에서 /aws/eks/fluentbit-cloudwatch/logs 을 클릭한다



목록의 맨 마지막이 가장 나중 로그이다

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

{"id": 1, "name": "Alice", "age": 30}' 값이 이미 저장된 상태서 아래 같은 데이터로 쓰기 명령을 내리면 오류가 난다

|  |
| --- |
| curl -X POST http://$SERVICE\_URL/store/rds \  -H "Content-Type: application/json" \  -d '{"id": 1, "name": "Alice", "age": 30}' |

{"error":"(1062, \"Duplicate entry '1' for key 'my\_table.PRIMARY'\")"}

이 때의 로그 내용은 아래와 같이 나온다

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

curl $SERVICE\_URL/fetch/rds?id=5

명령 수행 후 로그는 아래와 같다: {"error":"No data found"}

텍스트, 스크린샷, 폰트, 문서이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

RDS 데이터 변경 명령을 수행한다

|  |
| --- |
| curl -X PUT http://$SERVICE\_URL/update/rds \  -H "Content-Type: application/json" \  -d '{"id": 1, "name": "Tom", "age": 40}' |

이 때의 로그 내용은 아래와 같이 나온다

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

**✅ 에러 로그는 바로 보이는데, 성공 로그는 느리게 보이는 이유**

**🔥 에러 로그 (logging.error(...) 또는 stderr)는 즉시 CloudWatch에 보임**

* Fluent Bit는 기본적으로 **stderr** 로그를 더 우선적으로 전송합니다.
* 에러는 보통 고유한 메시지(Exception, stack trace 등)가 많아 중복이 되지 않고, **즉시 전송**됨

**😐 일반 로그 (logging.info(...) → stdout)는 지연되거나 무시될 수 있음**

* stdout 로그는 Fluent Bit에서 **batching**, **buffering**, **deduplication** 대상이 됩니다
* 같은 메시지를 반복해서 출력하면 “중복”으로 인식되어 **일부 생략**되거나 **늦게 전송**됩니다
* stdout은 너무 많이 쌓이면 throttling(속도 제한)될 수도 있음

텍스트, 영수증, 폰트, 스크린샷이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

**🔍 예시: 같은 요청 여러 번 반복한 경우**

logging.info("✅ 조회 성공: (1, 'Alice', 30)")

이걸 계속 반복하면:

* Fluent Bit은 "같은 메시지 또 나왔네" → 버퍼에 쌓거나 무시
* 반면, 에러는 매번 다르거나 중요하다고 인식 → 바로 전송

**✅ 개선 방법**

**🔁 로그 메시지를 고유하게 만들기**

import datetime

logging.info(f"✅ 조회 성공: {result} at {datetime.datetime.now()}")

이렇게 하면 **CloudWatch에 항상 새로운 로그 이벤트로 인식**되어 잘 보입니다.

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

**🔧 실제 운영에서는…**

* stdout 로그도 반드시 남겨야 하는 경우 → **log level, 태그, timestamp** 필수
* JSON 포맷으로 structured logging 구성 시 더 안정적으로 CloudWatch로 전송 가능

**✅ cloud9에서 CloudWatch 로그 보는 명령**

**1. 가장 최근 로그 스트림 확인**

aws logs describe-log-streams \

--log-group-name /aws/eks/fluentbit-cloudwatch/logs \

--order-by LastEventTime --descending \

--limit 1

아래와 같이 가장 마지막의 로그 스트림 이름을 알 수 있다. 이 이름을 복사해놓는다

"logStreamName": "**fluentbit-kube.var.log.containers.msa-service-596cf6b865-n9bl4\_default\_msa-service-9ffdb6ea739cddce12f569aaca3514e06298a7b6fed22f604efaed414a7180a6.log**",

**2. 해당 로그 스트림의 최근 이벤트 보기**

aws logs get-log-events \

--log-group-name /aws/eks/fluentbit-cloudwatch/logs \

--log-stream-name **fluentbit-kube.var.log.containers.msa-service-596cf6b865-n9bl4\_default\_msa-service-9ffdb6ea739cddce12f569aaca3514e06298a7b6fed22f604efaed414a7180a6.log** \

--limit 10

최신 로그가 10개 출력된다 (AWS 콘솔에서 보는 것과 동일한 결과)

**실습 완료 후 모든 로그 그룹 삭제 (과금 방지)**

aws logs delete-log-group --log-group-name /aws/eks/fluentbit-cloudwatch/logs

로그 그룹 /aws/eks/my-eks-cluster/cluster 삭제는 25 Page 참조

로그 그룹 모두 삭제 후 CloudWatch 로그 그룹 확인

텍스트, 폰트, 스크린샷이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

이제 CloudWatch 기반의 로깅 및 모니터링 실습 구성이 완료되었습니다.  
배포된 마이크로서비스의 성능, 상태, 로그를 확인할 수 있는 환경이 갖춰졌다.