Cloud Infra, Kubernetes

컨테이너 오케스트레이션에 관하여…

개발자가 꼭 알아야하는 기술

#AWS #Devops #Github-Actions
#CI/CD #Docker #k8s

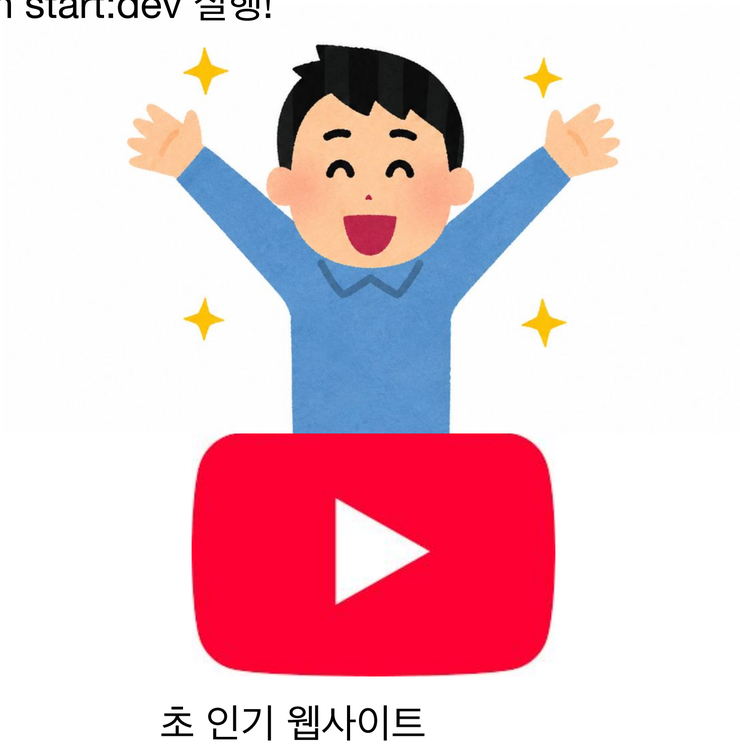
10201 구본무

m161awm.kr

데브옵스/클라우드 인프라를 모르고 그냥 배포만 했을 때

드디어 다 만들었다! git push… 푸시 완료! 이제 서버에서 받고 실행하면!
(패키지 설치만 10분걸림) git clone… npm run start:dev 실행!

저희 사이트 운영합니다 ~



초 인기 웹사이트

데브옵스/클라우드 인프라를 모르고 그냥 배포만 했을 때

초 인기 웹사이트

데브옵스/클라우드 인프라를 모르고 그냥 배포만 했을 때

초 인기 웹사이트

데브옵스/클라우드 인프라를 모르고 그냥 배포만 했을 때

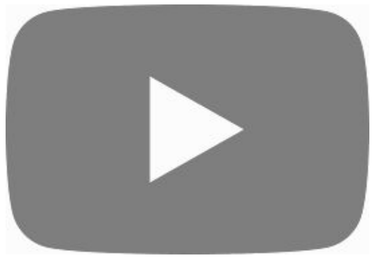
초 인기 웹사이트

데브옵스/클라우드 인프라를 모르고 그냥 배포만 했을 때

초 인기 웹사이트

그런거 들어갈리 없어!

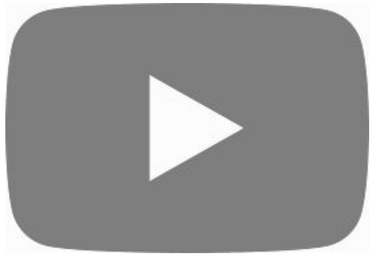
데브옵스/클라우드 인프라를 모르고 그냥 배포만 했을 때



초 인기 웹사이트(였던것)

그런거 들어갈리 없어!

데브옵스/클라우드 인프라를 모르고 그냥 배포만 했을 때



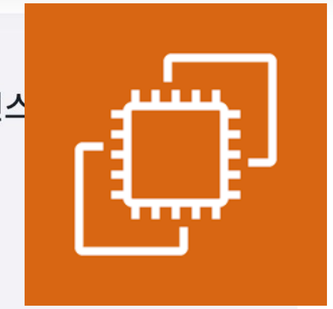
초 인기 웹사이트(였던것)

AWS를 사용했다면?

aws AWS Summit | AWS 살펴보기 제품 솔루션 요금 리소스 검색 콘솔 로그인 계정 생성

클라우드 인프라를 알고있었다면

Amazon EC2 개요 기능 요금 인스턴스 유형 FAQ 시작하기 리소스



나여~ EC2

Amazon EC2 C6g 인스턴스
컴퓨팅 최적화

Amazon EC2 C8gd 인스턴스
컴퓨팅 최적화

Amazon EC2 C8i 인스턴스
컴퓨팅 최적화

Amazon EC2 C5a 인스턴스
컴퓨팅 최적화

Amazon EC2 C8i-flex 인스턴스
컴퓨팅 최적화

Amazon EC2 M7gd 인스턴스
범용

알맞는 서버를 직접 빌릴 수 있음

월 0원부터 5000만원까지 다양하다! 거의 900개에 가까운 인스턴스가 존재함

클라우드 인프라를 알고있었다면

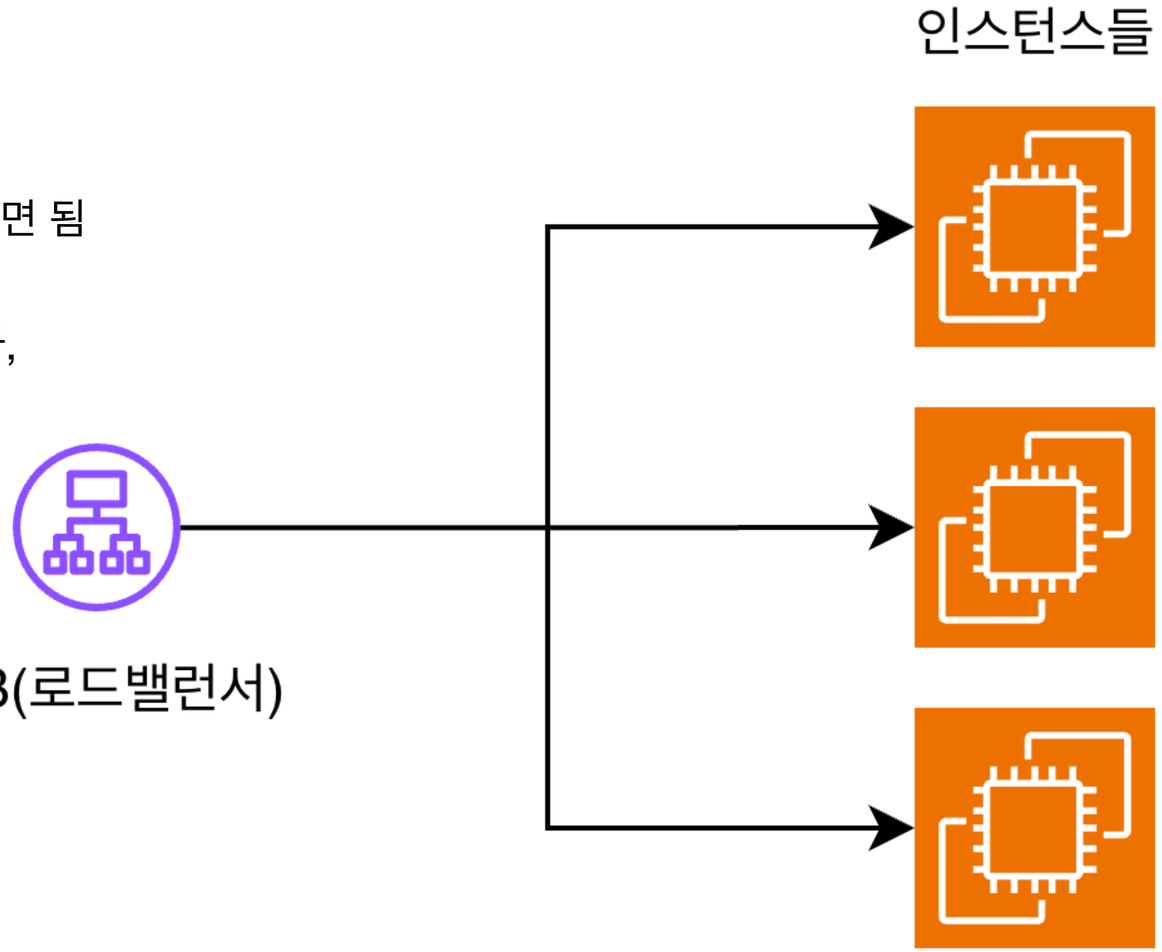
근데 얘도 터지면 어떡함??

클라우드 인프라를 알고있었다면

근데 얘도 터지면 어떡함??

인스턴스를 많이 만들고
EC2들 대상그룹 묶어주고 로드밸런서에 넣어주면 됨

터지면 다른 인스턴스들로 보내는것 뿐만 아니라,
트래픽을 분산시켜줌

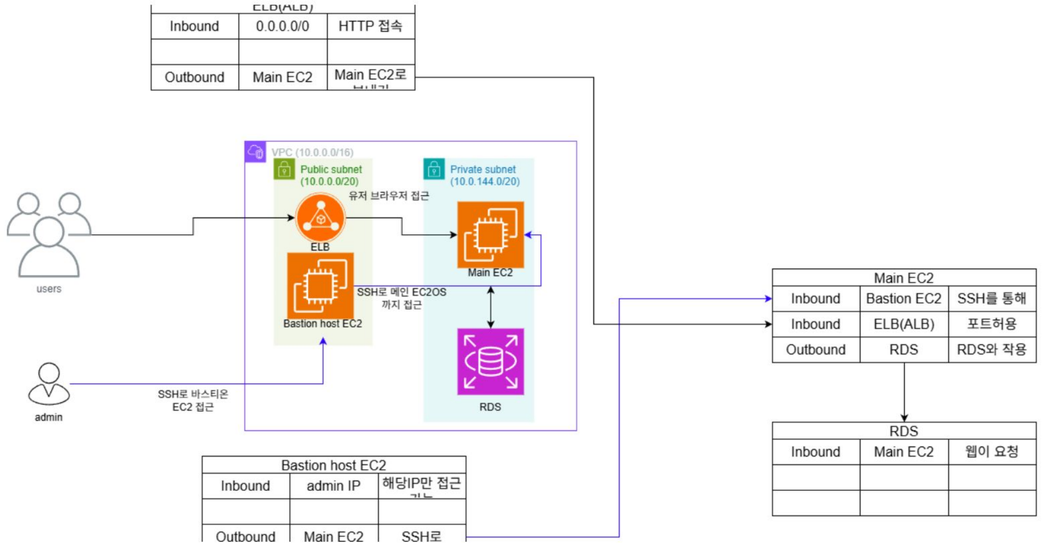


클라우드 인프라를 알고있었다면

근데 얘도 터지면 어떡함??

인스턴스를 많이 만들고
EC2들 대상그룹 묶어주고 로드밸런서에 넣어주면 됨

터지면 다른 인스턴스들로 보내는것 뿐만 아니라,
트래픽을 분산시켜줌



AWS 안전한 인프라 구축하기

기존까진 그냥 EC2를 만든 후 AWS가 자동으로 VPC만들고, RDS를 붙혀오는 방식이였다. 근데 이런식이면 누가 클라우드 엔지니어라고 인정을 하겠나 그래서 내가 할 수 있는 한 직접 클라우드 인프라를 구축해볼것이다. 첫번째로 VPC를 만드는거다, VPC는 스위...

aws

2026년 4월 23일 . 0개의 댓글 · ♥ 1

자세한 내용은 블로그 참조

AWS를 배워야하는이유

일반 개발자도 AWS를 필수로 알아야할 시대가 옴
채용 공고 사이트에서 개발자 공고를 확인해본다면

상세요강

백엔드 개발

- 서버 사이드 로직 개발
  - Java/Spring Boot 기반 서버 애플리케이션 개발
- 데이터 모델링 및 쿼리 최적화
  - MySQL 스키마 설계, 쿼리 최적화 및 인덱스 관리
- 웹 UI/UX 구현
  - HTML, CSS, JavaScript
- 클라우드 및 인프라 활용
  - AWS 클라우드 환경에서 서비스 운영
- 협업 및 버전 관리
  - Git/GitHub를 활용한 버전 관리 및 협업

신입 ~ 경력 4년

Click

백앤드(인프라) 경력 채용공고(솔루션 개발)

- 자격 요건
  - 백엔드 개발 실무 경험 3년 이상 (Spring Boot 또는 FastAPI 필수)
  - REST API 및 DB(PostgreSQL, MongoDB) 설계 실무 역량
  - CI/CD 구축·운영 경험 (Jenkins, ArgoCD, Helm 등)
  - 클라우드 환경에서의 서비스 배포 및 인프라 운영 경험
  - 서비스 최적화(캐시 전략, 로깅, !
- 우대 사항
  - CSAP 보안 인증 심사 준비 및 운
  - Pandas, Numpy 등 통계 관련
  - Kafka, Redis 등 분산 시스템 및
  - 데이터 처리 파이프라인 개선 경

백엔드개발자-node.js/nestjs (신입~7년)

상세요강

자격요건

- 경력: 신입 ~ 7년 이하
- typescript를 활용한 개발 작업에 능숙하신 분
- NestJS 를 활용한 개발 경험이 있으신 분
- AWS와 github 등을 활용한 CI / CD 의 구성이 가능하신 분
- 서비스 아키텍처 설계 및 기술 의사결정 주도
- 장애 대응 및 트러블슈팅 경험 (로그 분석, 성능 병목 해결 등)
- 데이터베이스 설계 및 성능 최적화 경험

AWS를 배워야하는이유

일반 개발자도 AWS를 필수로 알아야할 시대가 옴
채용 공고 사이트에서 개발자 공고를 확인해본다면

상세요강

백엔드 개발

- 서버 사이드 로직 개발
  - Java/Spring Boot 기반 서버 애플리케이션 개발
- 데이터 모델링 및 쿼리 최적화
  - MySQL 스키마 설계, 쿼리 최적화 및 인덱스 관리
- 웹 UI/UX 구현
  - HTML, CSS, JavaScript
- 클라우드 및 인프라 활용
  - AWS 클라우드 환경에서 서비스 운영
- 협업 및 버전 관리
  - Git/GitHub를 활용한 버전 관리 및 협업

신입 ~ 경력 4년

Click

분명히 백엔드 개발자를 뽑는건데?

백앤드(인프라) 경력 채용공고(솔루션 개발)

- 자격 요건
  - 백엔드 개발 실무 경험 3년 이상 (Spring Boot 또는 FastAPI 필수)
  - REST API 및 DB(PostgreSQL, MongoDB) 설계 실무 역량
  - CI/CD 구축·운영 경험 (Jenkins, ArgoCD, Helm 등)
  - 클라우드 환경에서의 서비스 배포 및 인프라 운영 경험
  - 서비스 최적화(캐시 전략, 로깅, !
- 우대 사항
  - CSAP 보안 인증 심사 준비 및 운
  - Pandas, Numpy 등 통계 관련
  - Kafka, Redis 등 분산 시스템 및
  - 데이터 처리 파이프라인 개선 경

백엔드개발자-node.js/nestjs (신입~7년)

상세요강

자격요건

- 경력: 신입 ~ 7년 이하
- typescript를 활용한 개발 작업에 능숙하신 분
- NestJS 를 활용한 개발 경험이 있으신 분
- AWS와 github 등을 활용한 CI / CD 의 구성이 가능하신 분
- 서비스 아키텍처 설계 및 기술 의사결정 주도
- 장애 대응 및 트러블슈팅 경험 (로그 분석, 성능 병목 해결 등)
- 데이터베이스 설계 및 성능 최적화 경험

AWS를 배워야하는이유

일반 개발자도 AWS를 필수로 알아야할 시대가 옴
채용 공고 사이트에서 개발자 공고를 확인해본다면

상세요강

백엔드 개발

- 서버 사이드 로직 개발
  - Java/Spring Boot 기반 서버 애플리케이션 개발
- 데이터 모델링 및 쿼리 최적화
  - MySQL 스키마 설계, 쿼리 최적화 및 인덱스 관리
- 웹 UI/UX 구현
  - HTML, CSS, JavaScript
- 클라우드 및 인프라 활용
  - AWS 클라우드 환경에서 서비스 운영
- 협업 및 버전 관리
  - Git/GitHub를 활용한 버전 관리 및 협업

신입 ~ 경력 4년

Click

분명히 백엔드 개발자를 뽑는건데?

백앤드(인프라) 경력 채용공고(솔루션 개발)

- 자격 요건
  - 백엔드 개발 실무 경험 3년 이상 (Spring Boot 또는 FastAPI 필수)
  - REST API 및 DB(PostgreSQL, MongoDB) 설계 실무 역량
  - CI/CD 구축·운영 경험 (Jenkins, ArgoCD, Helm 등)
  - 클라우드 환경에서의 서비스 배포 및 인프라 운영 경험
  - 서비스 최적화(캐시 전략, 로깅, !
- 우대 사항
  - CSAP 보안 인증 심사 준비 및 운
  - Pandas, Numpy 등 통계 관련
  - Kafka, Redis 등 분산 시스템 및
  - 데이터 처리 파이프라인 개선 경

백엔드개발자-node.js/nestjs (신입~7년)

상세요강

자격요건

- 경력: 신입 ~ 7년 이하
- typescript를 활용한 개발 작업에 능숙하신 분
- NestJS 를 활용한 개발 경험이 있으신 분
- AWS와 github 등을 활용한 CI / CD 의 구성이 가능하신 분
- 서비스 아키텍처 설계 및 기술 의사결정 주도
- 장애 대응 및 트러블슈팅 경험 (로그 분석, 성능 병목 해결 등)
- 데이터베이스 설계 및 성능 최적화 경험

예전에는 백엔드 프레임워크 하나만 제대로 알고있어도 뽑혔는데 현재는 AWS도 필수다

AWS를 배워야하는이유

일반 개발자도 AWS를 필수로 알아야할 시대가 옴
채용 공고 사이트에서 개발자 공고를 확인해본다면

자신이 개발자라면 AWS를 어느정도 할 줄 알아야함.

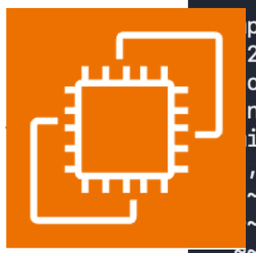
현재 IT분야 취업은 회사에서 직접 가르쳐서 정직원으로 채용하는게 줄어듦
=> 회사는 괴물신입을 원한다....

분명히 백엔드 개발자를 뽑는건데?

예전에는 백엔드 프레임 워크 하나만 제대로 알고있어도 뽑혔는데 현재는 AWS는 필수다

생길 문제 : 배포는 누가함?

인스턴스 만들때마다 계속 명령어 작성



admin — ec2-user@ip-10-0-0-14:~ — ssh -i ~/Downloads/instagramKey.pem ec2-user@13.210.118.244 — 120×30

```
  Verifying        : perl-Git-2.50.1-1.amzn2023.0.1.noarch                                    6/8
  Verifying        : perl-TermReadKey-2.38-9.amzn2023.0.3.x86_64                              7/8
  Verifying        : perl-lib-0.65-477.amzn2023.0.8.x86_64                                    8/8

Installed:
  git-2.50.1-1.amzn2023.0.1.x86_64                  git-core-2.50.1-1.amzn2023.0.1.x86_64
  git-core-doc-2.50.1-1.amzn2023.0.1.noarch         perl-Error-1:0.17030-2.amzn2023.0.1.noarch
  perl-File-Find-1.37-477.amzn2023.0.8.noarch       perl-Git-2.50.1-1.amzn2023.0.1.noarch
  perl-TermReadKey-2.38-9.amzn2023.0.3.x86_64       perl-lib-0.65-477.amzn2023.0.8.x86_64

plete!
2-user@ip-10-0-0-14 ~]$ exit
out
nection to 13.210.118.244 closed.
in@m161awmui-MacBookPro ~ % ssh -i "~/Downloads/instagramKey.pem" ec2-user@13.210.118.244
,     #_
~\_  ####_        Amazon Linux 2023
~  \_#####\
~~     \###|
~~       \#/ ___   https://aws.amazon.com/linux/amazon-linux-2023
 ~~       V~' '->
  ~~~         /
    ~~._.   _/
       _/ _/
     _/m/'
Last login: Sun Jun  7 11:35:21 2026 from 180.228.198.89
[ec2-user@ip-10-0-0-14 ~]$ git clone https://개쩌는웹사이트
Cloning into '개쩌는웹사이트'...
fatal: unable to access 'https://개쩌는웹사이트/': Could not resolve host: 개쩌는웹사이트
[ec2-user@ip-10-0-0-14 ~]$
```

생길 문제 : 배포는 누가함?

인스턴스 만들때마다 계속 작성 <= 3개가 아니라 30개면?

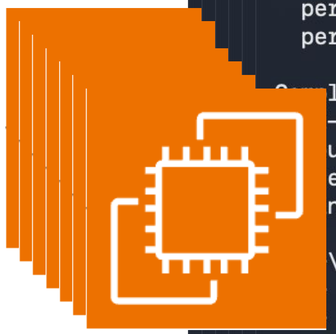
생길 문제 : 배포는 누가함? + 만약에 코드가 커밋되었다면?

또 30개의 인스턴스에서 git pull 해야함

생길 문제 : 배포는 누가함? + 만약에 코드가 커밋되었다면?

또 30개의 인스턴스에서 git pull 해야함

24시간동안 배포만 하게될거임



```
admin — ec2-user@ip-10-0-0-14:~ — ssh -i ~/Downloads/instagramKey.pem ec2-user@13.210.118.244 — 120×30

  Verifying        : perl-Git-2.50.1-1.amzn2023.0.1.noarch                                    6/8
  Verifying        : perl-TermReadKey-2.38-9.amzn2023.0.3.x86_64                              7/8
  Verifying        : perl-lib-0.65-477.amzn2023.0.8.x86_64                                    8/8

Installed:
  git-2.50.1-1.amzn2023.0.1.x86_64                    git-core-2.50.1-1.amzn2023.0.1.x86_64
  git-core-doc-2.50.1-1.amzn2023.0.1.noarch           perl-Error-1:0.17030-2.amzn2023.0.1.noarch
  perl-File-Find-1.37-477.amzn2023.0.8.noarch         perl-Git-2.50.1-1.amzn2023.0.1.noarch
  perl-TermReadKey-2.38-9.amzn2023.0.3.x86_64         perl-lib-0.65-477.amzn2023.0.8.x86_64

lete!
-user@ip-10-0-0-14 ~]$ exit
ut
ection to 13.210.118.244 closed.
n@m161awmui-MacBookPro ~ % ssh -i "~/Downloads/instagramKey.pem" ec2-user@13.210.118.244
      #_
\_  ####_        Amazon Linux 2023
  \_#####\
      \###|
  ~~       \#/ ___   https://aws.amazon.com/linux/amazon-linux-2023
   ~~       V~' '->
    ~~~         /
      ~~._.   _/
         _/ _/
       _/m/'
Last login: Sun Jun  7 11:35:21 2026 from 180.228.198.89
[ec2-user@ip-10-0-0-14 ~]$ git clone https://개쩌는웹사이트
Cloning into '개쩌는웹사이트'...
fatal: unable to access 'https://개쩌는웹사이트/': Could not resolve host: 개쩌는웹사이트
[ec2-user@ip-10-0-0-14 ~]$
```

수동 배포

서버마다 접속 · git clone · 재시작

→

자동화된 운영

GitHub Push · CI/CD · Docker Image · 배포 자동화

DevOps

코드가 바뀌면, 배포는 자동으로 흐르게 만든다

CI/CD 파이프라인

CI/CD란?

- CI/CD는 개발자가 코드를 돌리면
  자동으로 빌드, 테스트, 배포까지 진행하는 자동화 파이프라인

Code | Blame

```yaml
name: CI/CD Deploy to EC2

on:
  push:
    branches:
      - main

jobs:
  ci:
    runs-on: ubuntu-latest
    steps:
      - name: 코드 가져오기
        uses: actions/checkout@v4

      - name: Node.js 설치
        uses: actions/setup-node@v4
        with:
          node-version: '20'
          cache: 'npm'

      - name: 패키지 설치
        run: npm ci

      - name: 테스트 실행
        run: npm test

      - name: 빌드 실행
        run: npm run build

  cd:
    runs-on: ubuntu-latest
    needs: ci

    steps:
      - name: EC2 배포
        uses: appleboy/ssh-action@v1.0.3
        with:
          host: ${{ secrets.EC2_HOST }}
          username: ${{ secrets.EC2_USER }}
          key: ${{ secrets.EC2_KEY }}
          script: |
            cd /home/ec2-user/app
            git pull origin main
            npm ci
            npm run build
            pm2 restart nest-app
```

Node js 배포 기준

해당 워크플로는 Github Actions

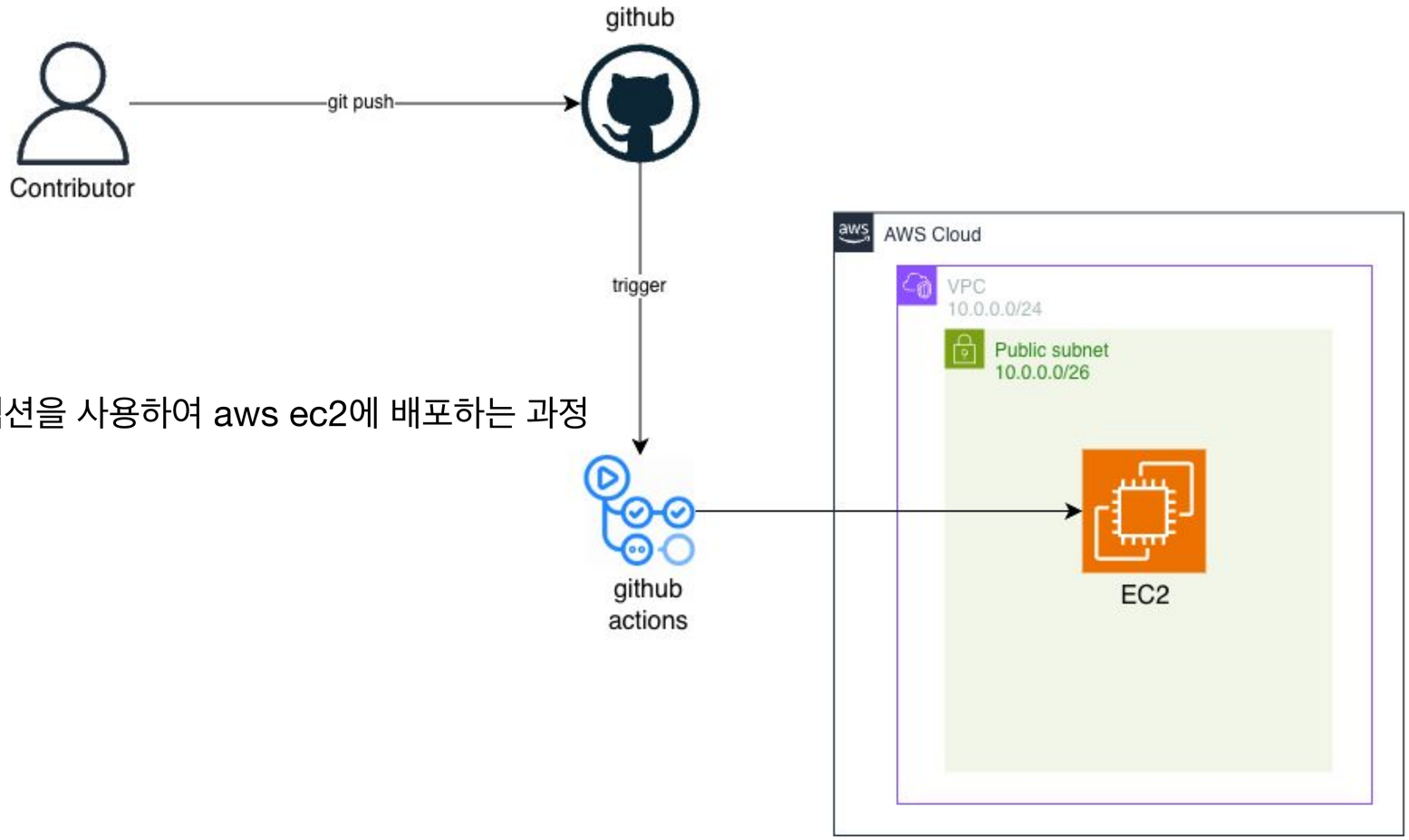
Jenkins

Github actions

CI/CD 파이프라인

깃허브 엑션즈를 활용하여 EC2에 지속적 배포

깃허브 액션을 사용하여 aws ec2에 배포하는 과정

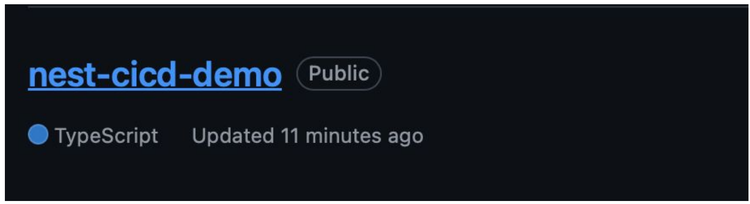


CI/CD 파이프라인

깃허브 액션을 사용하여 aws ec2에 배포하기 (실습)

```
host: ${{ secrets.EC2_HOST }}
username: ${{ secrets.EC2_USER }}
key: ${{ secrets.EC2_SSH_KEY }}
port: ${{ secrets.EC2_PORT }}
```

해당 파이프라인의 내용을 살펴보면 배포 부분의 환경 변수를 만들어줘야 EC2까지 무사히 배포가 된다는걸 알 수 있음



자세한 내용은 해당 레포에 README 부분을 확인

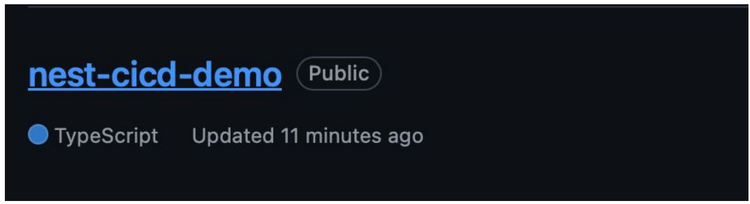
CI/CD 파이프라인

깃허브 액션을 사용하여 aws ec2에 배포하기 (실습)

```
host: ${{ secrets.EC2_HOST }}
username: ${{ secrets.EC2_USER }}
key: ${{ secrets.EC2_SSH_KEY }}
port: ${{ secrets.EC2_PORT }}
```

해당 파이프라인의 내용을 살펴보면 배포 부분의 환경 변수를 만들어줘야 EC2까지 무사히 배포가 된다는걸 알 수 있음

깃허브 내에서 환경변수를 지정하지 못하면 러너는 EC2에 ssh를 하지못하고 배포도 불가능해짐



자세한 내용은 해당 레포에 README 부분을 확인

CI/CD 파이프라인

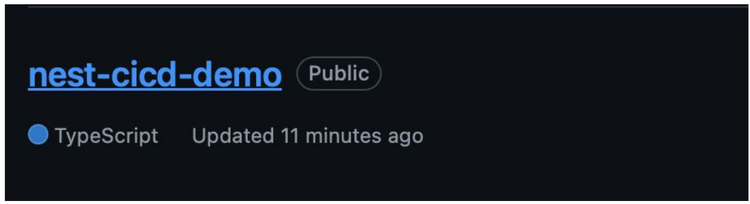
깃허브 액션을 사용하여 aws ec2에 배포하기 (실습)

```
host: ${{ secrets.EC2_HOST }}
username: ${{ secrets.EC2_USER }}
key: ${{ secrets.EC2_SSH_KEY }}
port: ${{ secrets.EC2_PORT }}
```

해당 파이프라인의 내용을 살펴보면 배포 부분의 환경 변수를 만들어줘야 EC2까지 무사히 배포가 된다는걸 알 수 있음

깃허브 내에서 환경변수를 지정하지 못하면 러너는 EC2에 ssh를 하지못하고 배포도 불가능해짐

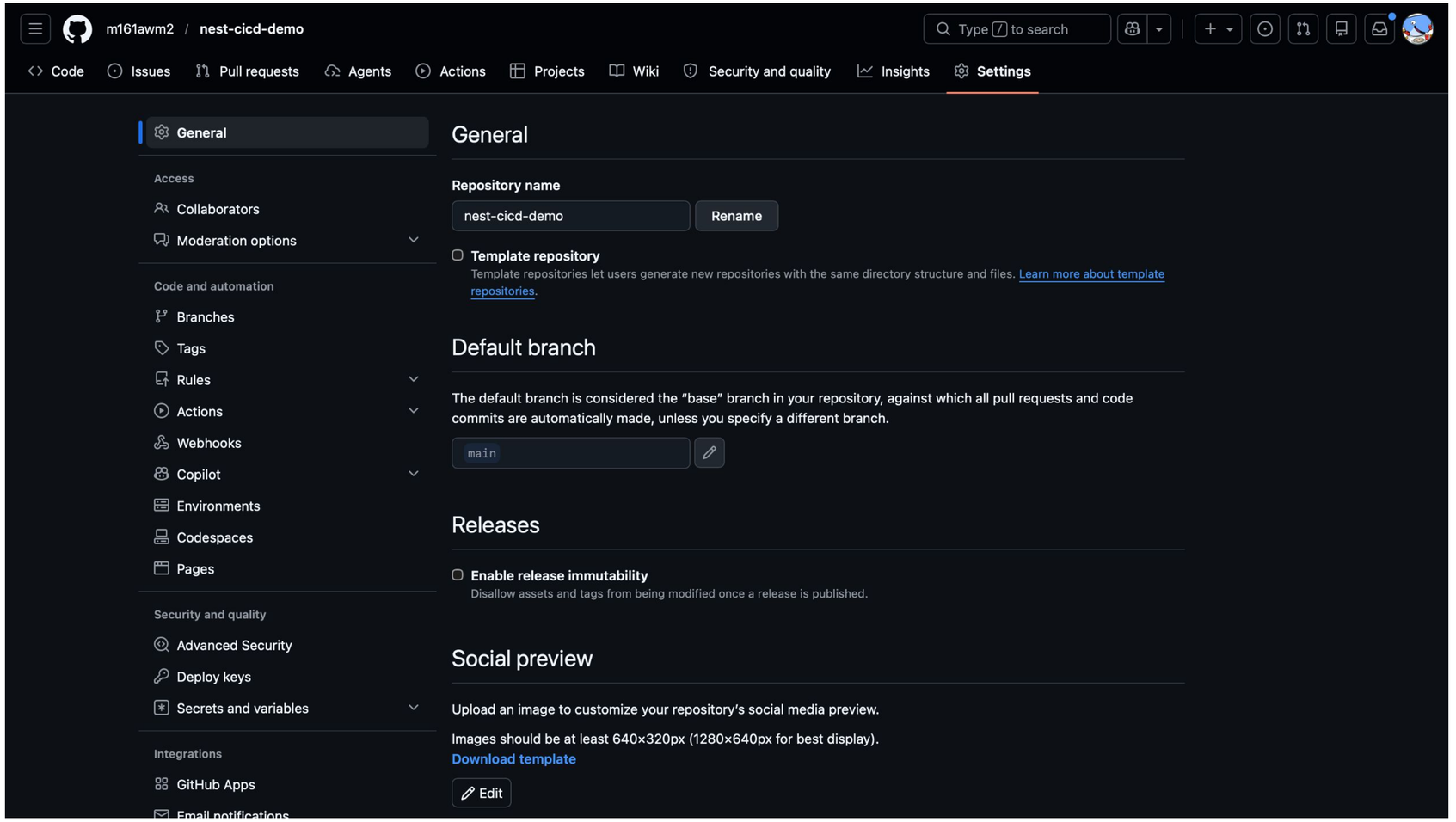
환경 변수를 지정해보자!



자세한 내용은 해당 레포에 README 부분을 확인

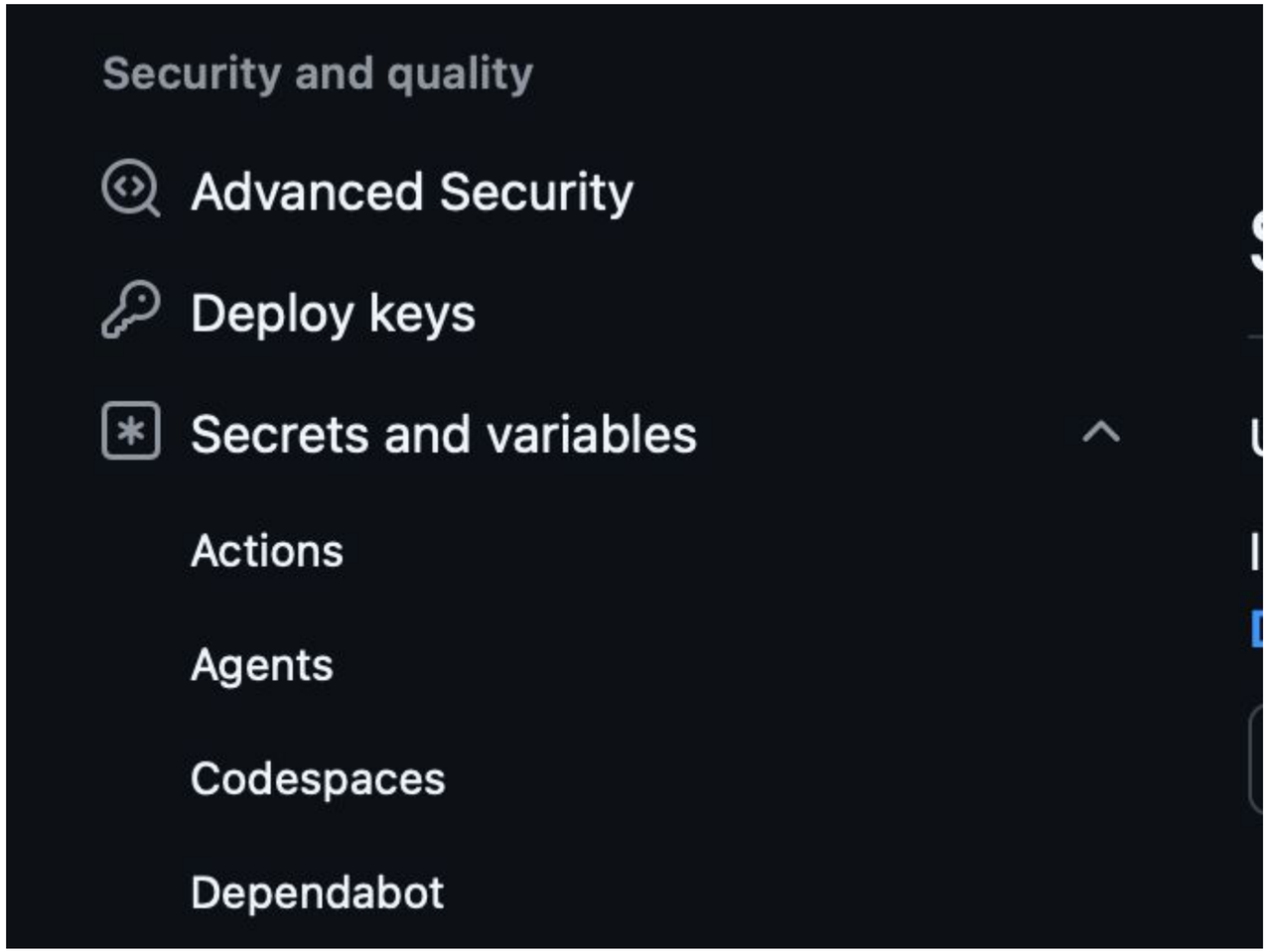
CI/CD 파이프라인

깃허브 액션을 사용하여 aws ec2에 배포하기 (실습)



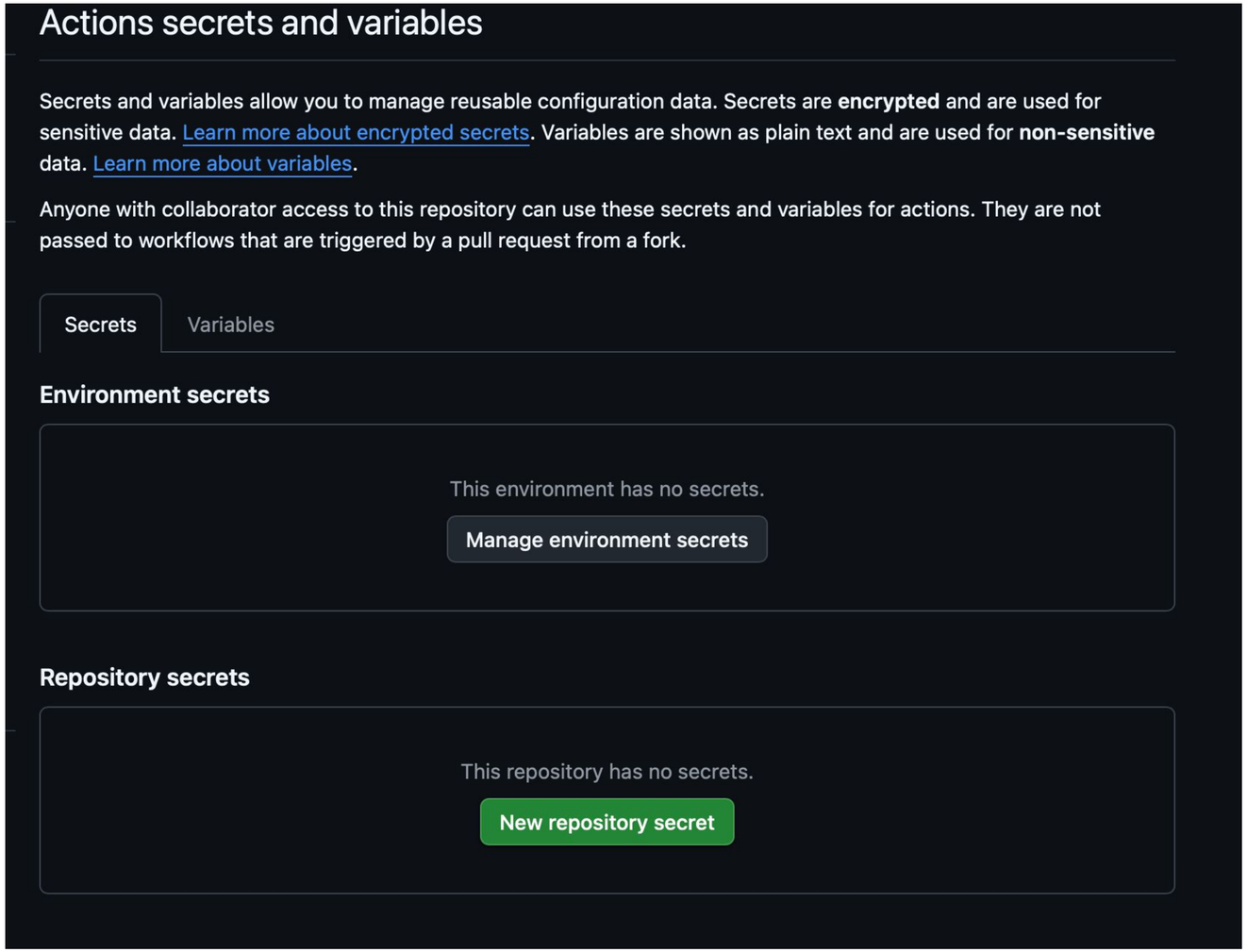
깃허브 레포의 Settings 탭에 들어가면 왼쪽에 사이드바가 보인다.

깃허브 액션을 사용하여 aws ec2에 배포하기 (실습)



그중 **Secrets and variables**안에 있는 **Actions**를 눌러보자

깃허브 액션을 사용하여 aws ec2에 배포하기 (실습)

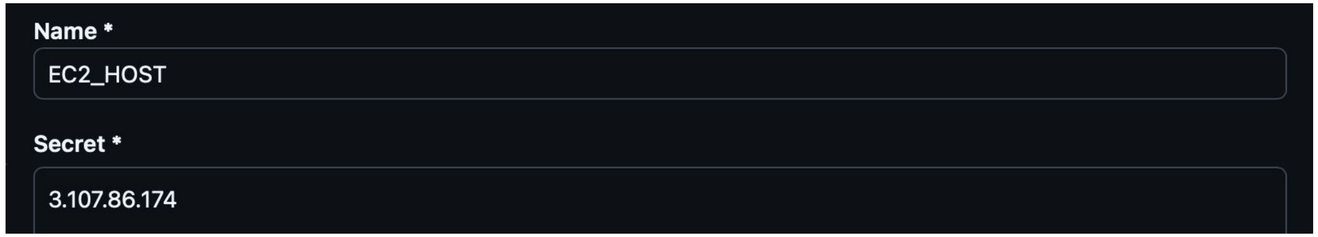


New repository secret 클릭

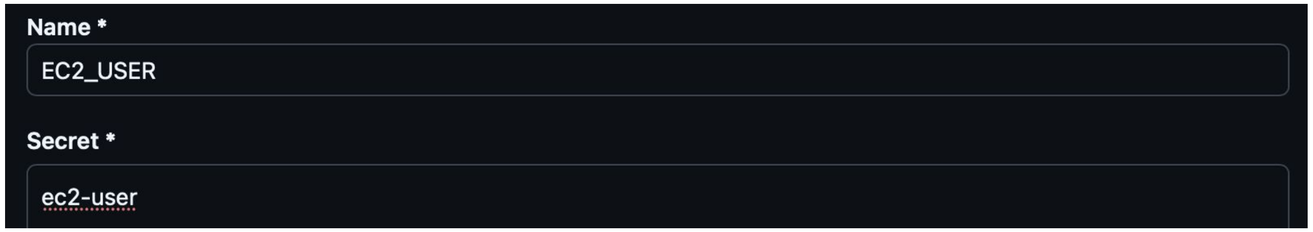
CI/CD 파이프라인

깃허브 액션을 사용하여 aws ec2에 배포하기 (실습)

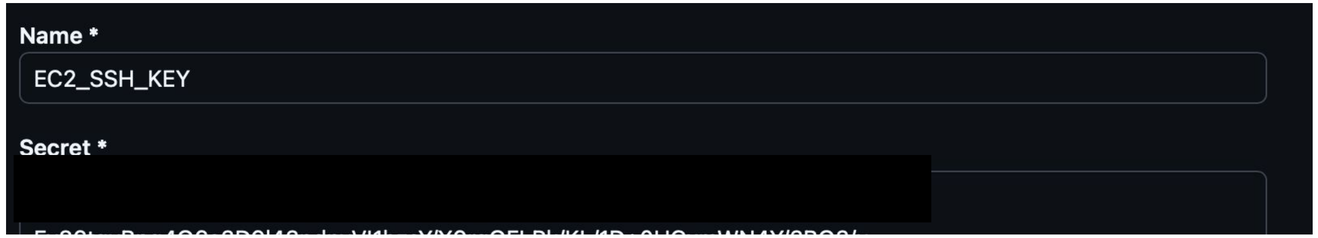
EC2_HOST에는 EC2의 퍼블릭 IP / EIP



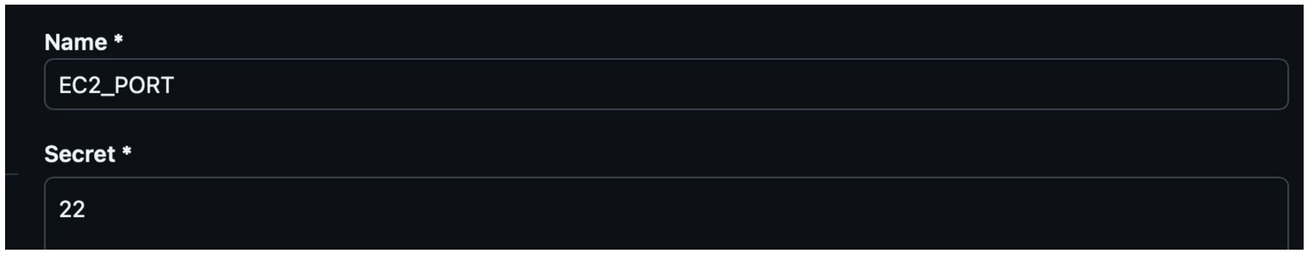
EC2_USER에는 ec2유저의 이름을



EC2_SSH_KEY에는 키페어 내용 전체



EC2_PORT에는 SSH포트번호



깃허브 액션을 사용하여 aws ec2에 배포하기 (실습)

```
[ec2-user@ip-10-0-0-45 ~]$ git clone https://github.com/m161awm2/nest-cicd-demo
Cloning into 'nest-cicd-demo'...
remote: Enumerating objects: 36, done.
remote: Counting objects: 100% (36/36), done.
remote: Compressing objects: 100% (27/27), done.
remote: Total 36 (delta 4), reused 36 (delta 4), pack-reused 0 (from 0)
Receiving objects: 100% (36/36), 94.71 KiB | 13.53 MiB/s, done.
Resolving deltas: 100% (4/4), done.
[ec2-user@ip-10-0-0-45 ~]$
```

깃허브 액션을 사용하여 aws ec2에 배포하기 (실습)

```
[3:48:58 PM] Starting compilation in watch mode...

src/app.controller.ts:1:33 - error TS2307: Cannot find module '@nestjs/common' or its corresponding type declarations.

1 import { Controller, Get } from '@nestjs/common';
                              ~~~~~~~~~~~~~~~~~

src/app.module.ts:1:24 - error TS2307: Cannot find module '@nestjs/common' or its corresponding type declarations.

1 import { Module } from '@nestjs/common';
                         ~~~~~~~~~~~~~~~~

src/app.module.ts:2:30 - error TS2307: Cannot find module '@nestjs/config' or its corresponding type declarations.

2 import { ConfigModule } from '@nestjs/config';
                               ~~~~~~~~~~~~~~~~

src/app.module.ts:3:35 - error TS2307: Cannot find module '@nestjs/serve-static' or its corresponding type declarations.

3 import { ServeStaticModule } from '@nestjs/serve-static';
                                    ~~~~~~~~~~~~~~~~~~~~~~

src/app.service.ts:1:28 - error TS2307: Cannot find module '@nestjs/common' or its corresponding type declarations.

1 import { Injectable } from '@nestjs/common';
                             ~~~~~~~~~~~~~~~~

src/main.ts:1:29 - error TS2307: Cannot find module '@nestjs/core' or its corresponding type declarations.

1 import { NestFactory } from '@nestjs/core';
                              ~~~~~~~~~~~~~

[3:49:00 PM] Found 6 errors. Watching for file changes.
```

그냥 바로 실행하려하니 패키지를 설치하지 않아서 오류가 나는 모습

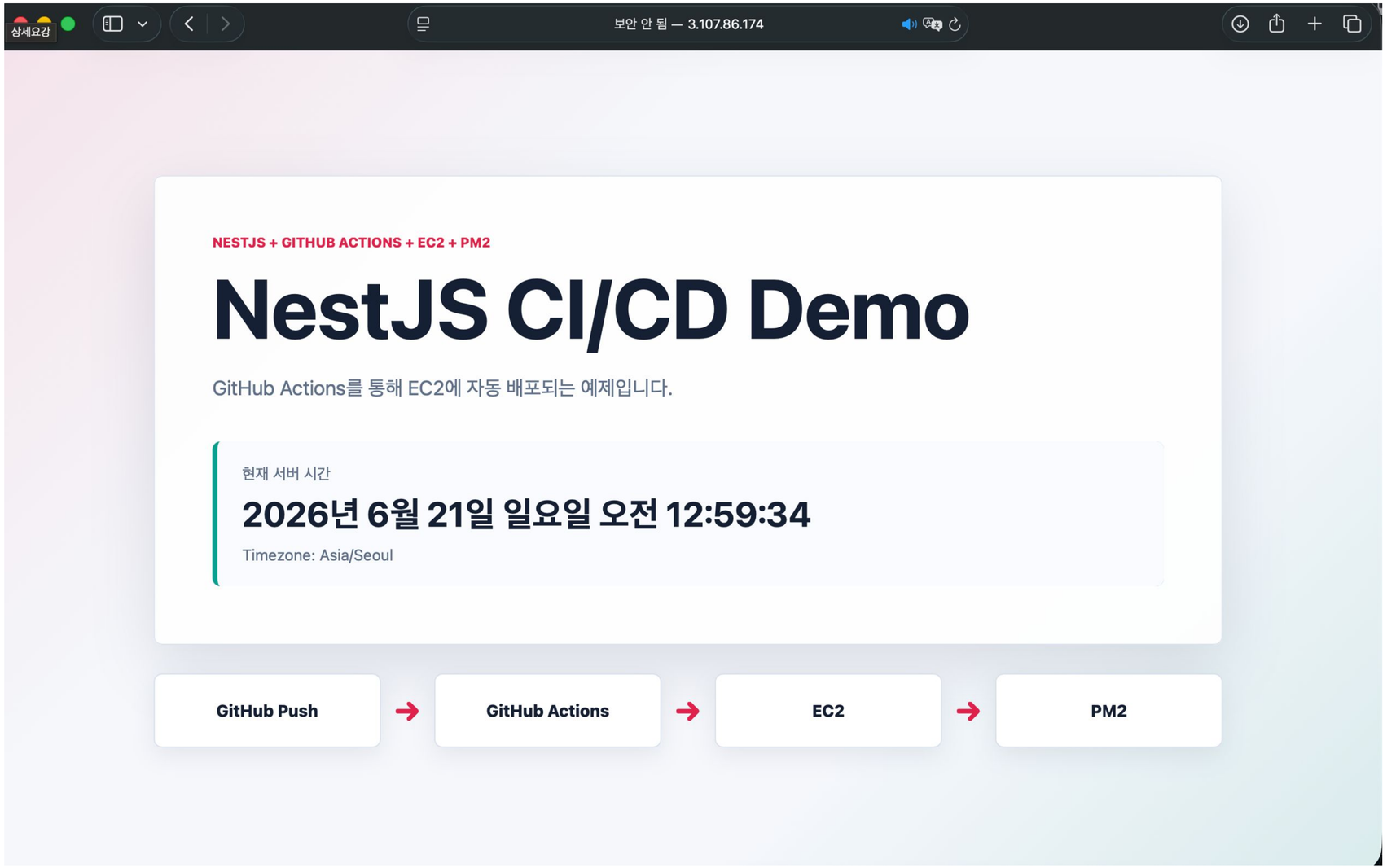
깃허브 액션을 사용하여 aws ec2에 배포하기 (실습)

```
added 1 package, and audited 695 packages in 3s

151 packages are looking for funding
  run `npm fund` for details

24 vulnerabilities (18 moderate, 6 high)

To address issues that do not require attention, run:
  npm audit fix

To address all issues (including breaking changes), run:
  npm audit fix --force

Run `npm audit` for details.
[ec2-user@ip-10-0-0-45 nest-cicd-demo]$
```

패키지를 설치해주고 재실행

CI/CD 파이프라인

깃허브 액션을 사용하여 aws ec2에 배포하기 (실습)



패키지 설치후 실행해보니 EC2서버가 잘 작동하는걸 알 수 있다.

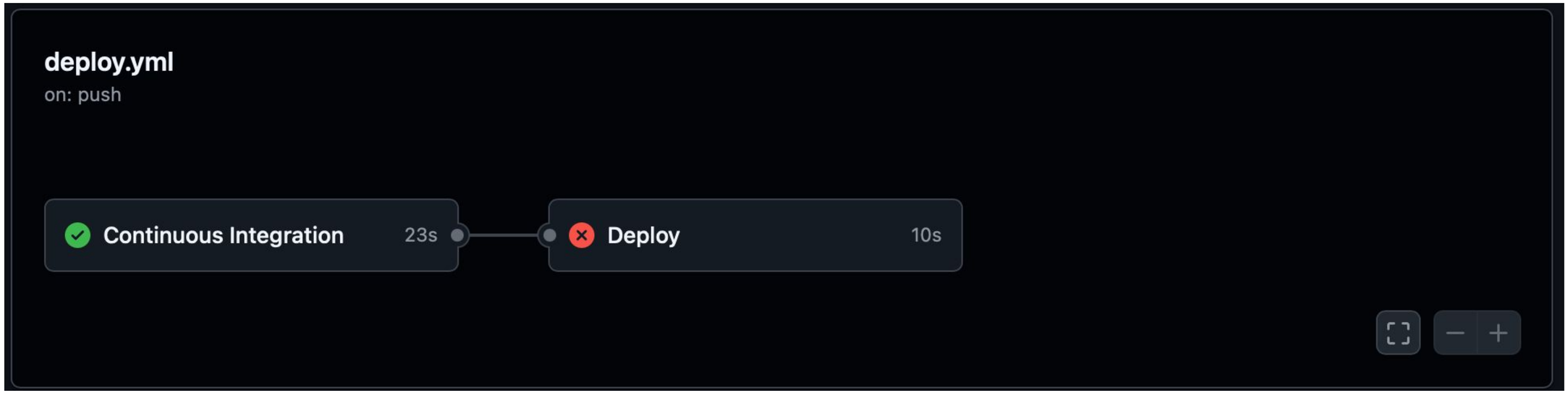
깃허브 액션을 사용하여 aws ec2에 배포하기 (실습)

```
admin@m161awmui-MacBookPro nest-cicd-demo % git add .
admin@m161awmui-MacBookPro nest-cicd-demo % git commit -m "actions Test Push"
[main 23002d1] actions Test Push
 3 files changed, 66 insertions(+), 54 deletions(-)
admin@m161awmui-MacBookPro nest-cicd-demo % git push
Enumerating objects: 15, done.
Counting objects: 100% (15/15), done.
Delta compression using up to 10 threads
Compressing objects: 100% (6/6), done.
Writing objects: 100% (8/8), 1.04 KiB | 1.04 MiB/s, done.
Total 8 (delta 4), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (4/4), completed with 4 local objects.
To https://github.com/m161awm2/nest-cicd-demo
   c77af6e..23002d1  main -> main
admin@m161awmui-MacBookPro nest-cicd-demo %
```

로컬에서 코드 에디터로 일정영역을 수정해보고 푸시

CI/CD 파이프라인

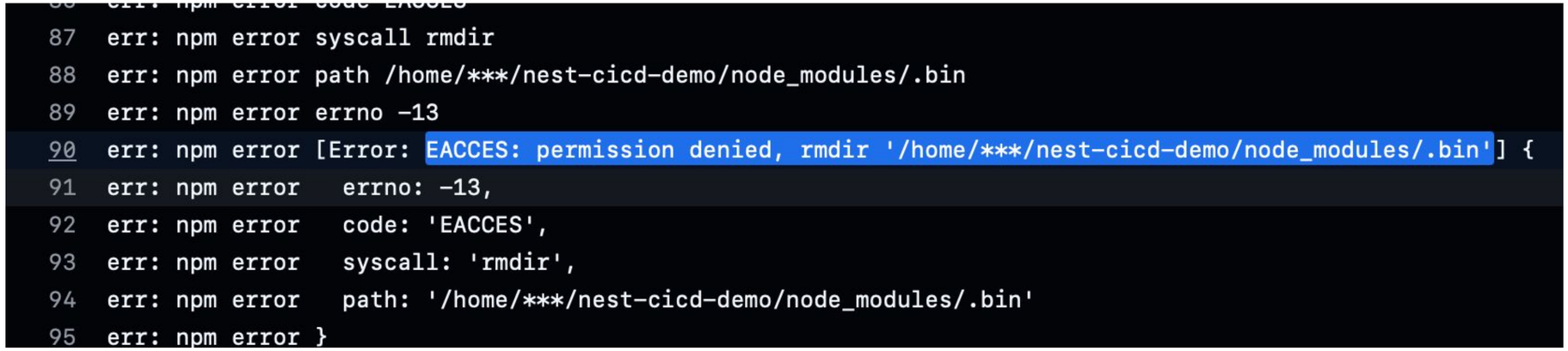
깃허브 액션을 사용하여 aws ec2에 배포하기 (실습)



액션을 확인해보니 CI는 성공했지만 CD는 실패하였다

CI/CD 파이프라인

깃허브 액션을 사용하여 aws ec2에 배포하기 (실습)



원인을 분석해보니 권한문제, sudo npm 명령어로 실행해서 node_moudules 폴더의 권한이 root로 잡혀있어서 액션으로 들어오는 유저는 일반 유저기 때문에 CD쪽에 문제가 발생한걸 알 수 있음

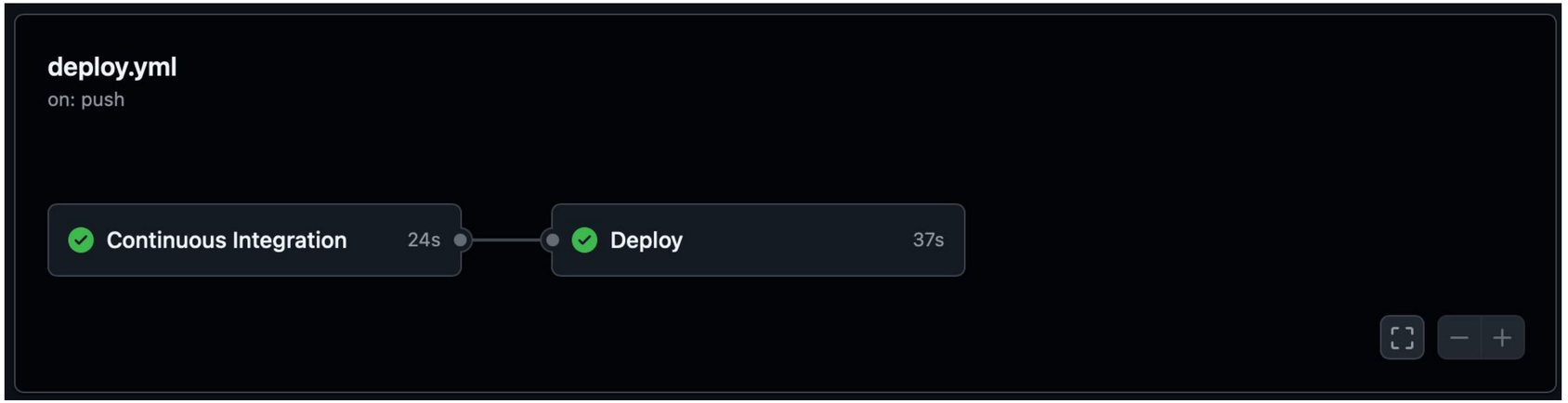
깃허브 액션을 사용하여 aws ec2에 배포하기 (실습)

```
[ec2-user@ip-10-0-0-45 nest-cicd-demo]$ sudo chown -R ec2-user:ec2-user /home/ec2-user/nest-cicd-demo
[ec2-user@ip-10-0-0-45 nest-cicd-demo]$
```

그렇게 때문에 change own 명령어로 이 디렉토리의 주인을 일반 유저(ec2-user)로 넘겨주고 다시 푸시를 해보겠다.

CI/CD 파이프라인

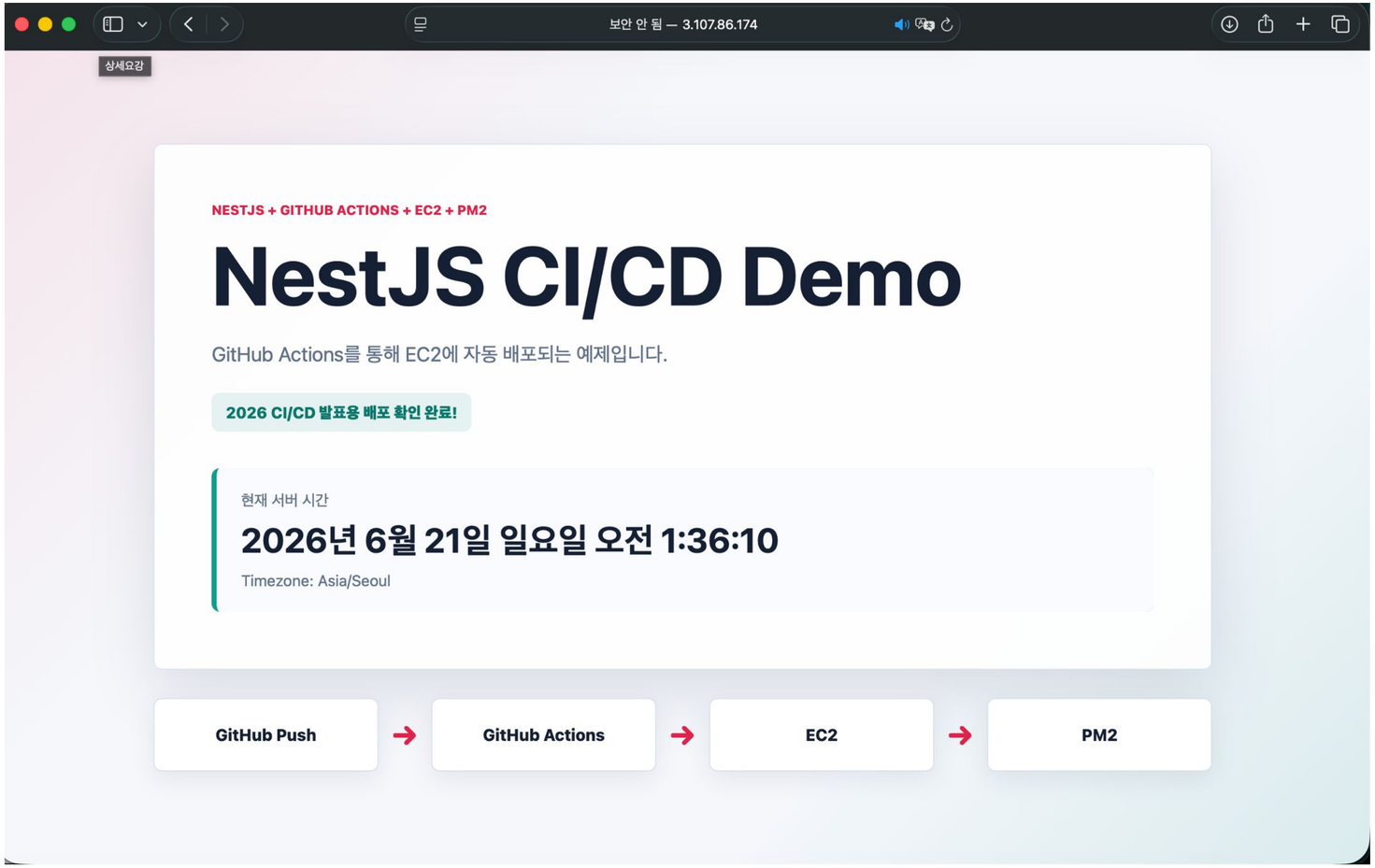
깃허브 액션을 사용하여 aws ec2에 배포하기 (실습)



파이프라인이 정상적으로 작동된걸 알 수 있음

CI/CD 파이프라인

깃허브 액션을 사용하여 aws ec2에 배포하기 (실습)



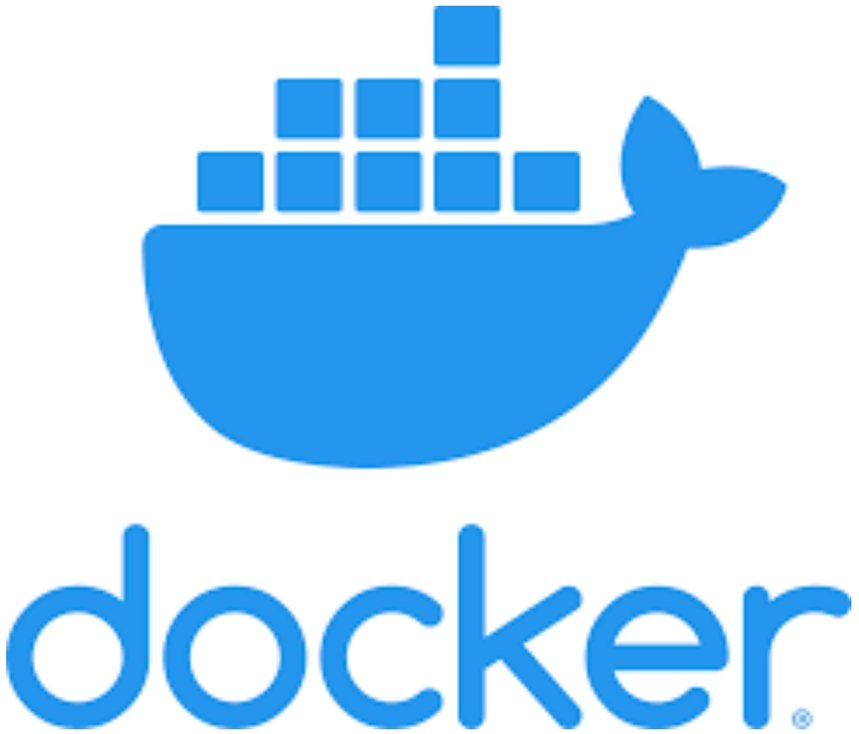
결론적으로 배포가 성공적으로 완료된걸 알 수 있다.

깃허브 액션을 사용하여 aws ec2에 배포하기 (실습)

```
[3:48:58 PM] Starting compilation in watch mode...

src/app.controller.ts:1:33 - error TS2307: Cannot find module '@nestjs/common' or its corresponding type declarations.

1 import { Controller, Get } from '@nestjs/common';
                               ~~~~~~~~~~~~~~~~~

src/app.module.ts:1:24 - error TS2307: Cannot find module '@nestjs/common' or its corresponding type declarations.

1 import { Module } from '@nestjs/common';
                         ~~~~~~~~~~~~~~~~

src/app.module.ts:2:30 - error TS2307: Cannot find module '@nestjs/config' or its corresponding type declarations.

2 import { ConfigModule } from '@nestjs/config';
                               ~~~~~~~~~~~~~~~~

src/app.module.ts:3:35 - error TS2307: Cannot find module '@nestjs/serve-static' or its corresponding type declarations.

3 import { ServeStaticModule } from '@nestjs/serve-static';
                                    ~~~~~~~~~~~~~~~~~~~~~~

src/app.service.ts:1:28 - error TS2307: Cannot find module '@nestjs/common' or its corresponding type declarations.

1 import { Injectable } from '@nestjs/common';
                             ~~~~~~~~~~~~~~~~~

src/main.ts:1:29 - error TS2307: Cannot find module '@nestjs/core' or its corresponding type declarations.

1 import { NestFactory } from '@nestjs/core';
                              ~~~~~~~~~~~~~~

[3:49:00 PM] Found 6 errors. Watching for file changes.
```

그래도 문제점이 있음, 인스턴스가 늘어난다면 계속 패키지를 깔아줘야하고 설치 경로가 꼬일수도 있음

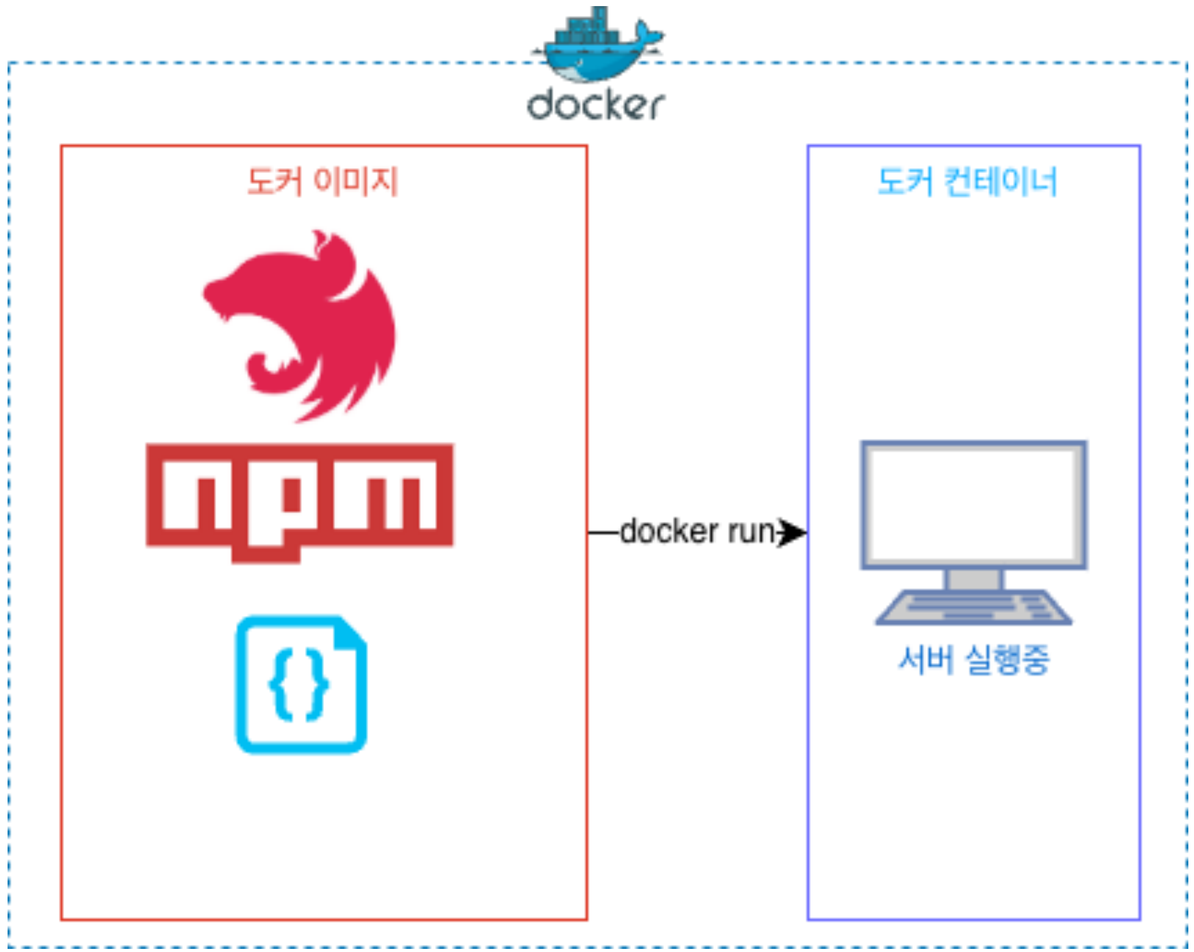
도커란?



이런 문제를 해결하기 위해 도커가 존재한다.

도커란?

소스코드나 패키지들을 한번에 묶어 컨테이너로 만들어주는 개발 도구

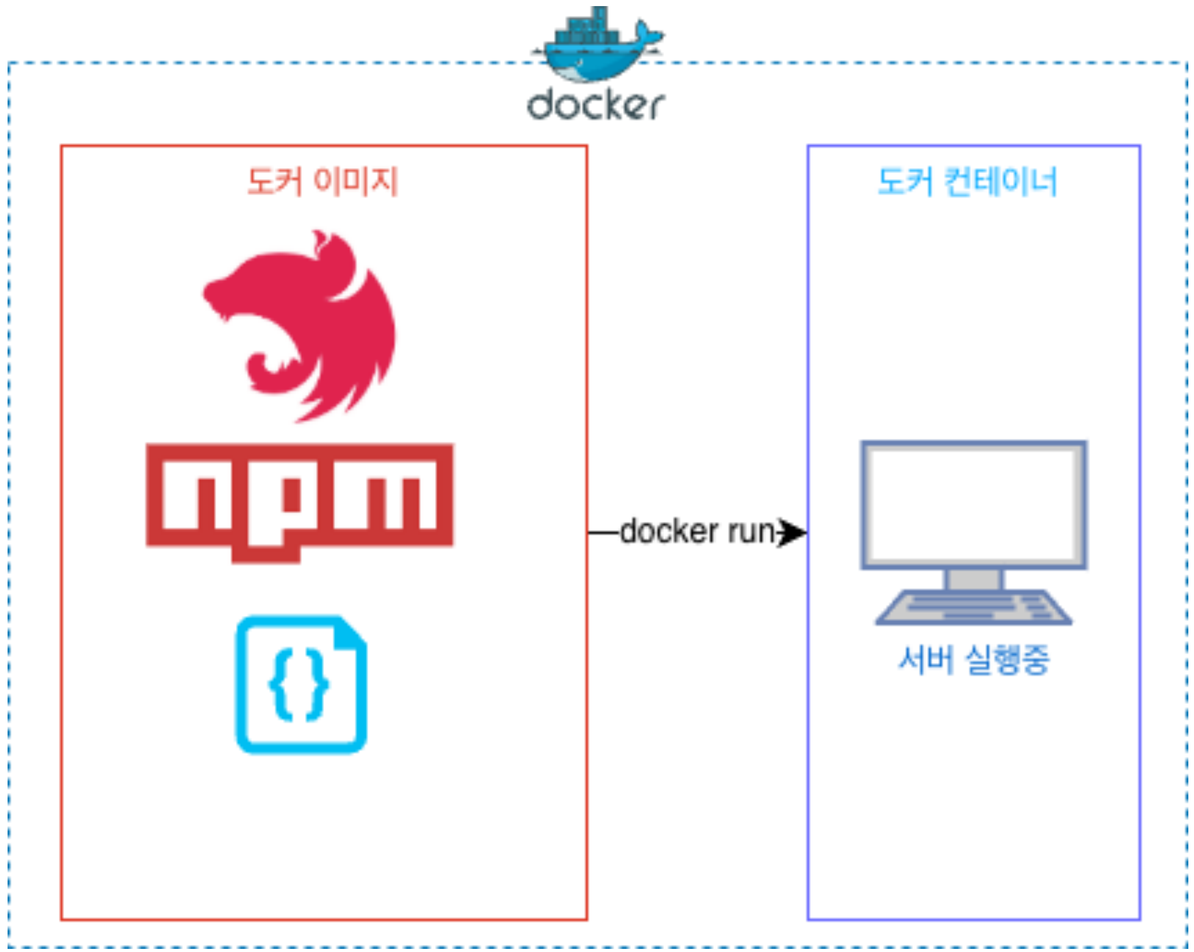


CI/CD 파이프라인

도커란?

소스코드나 패키지들을 한번에 묶어 컨테이너로 만들어주는 개발 도구

컨테이너라는 환경에서 실행시켜주기 때문에 호스트의 OS 영향을 받지않음

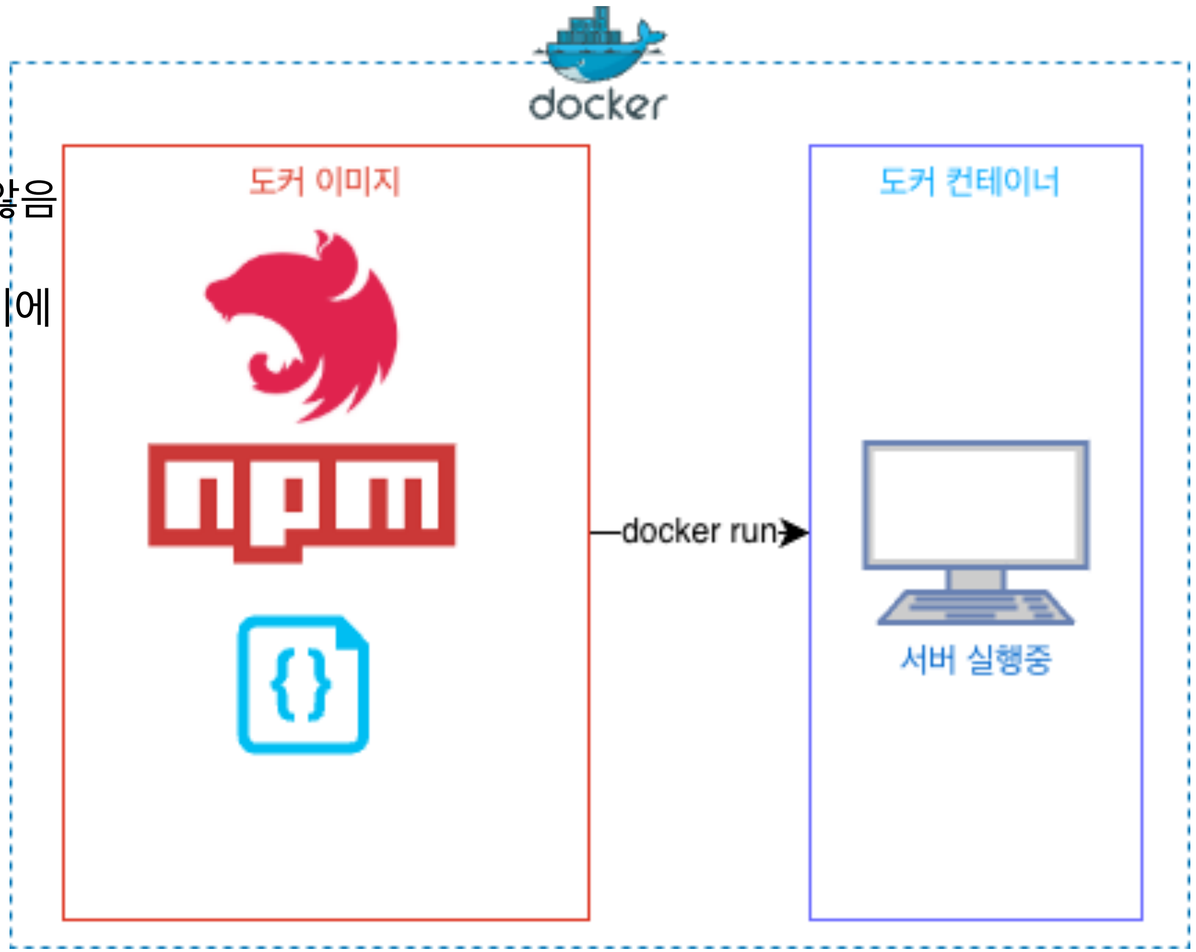


도커란?

소스코드나 패키지들을 한번에 묶어 컨테이너로 만들어주는 개발 도구

컨테이너라는 환경에서 실행시켜주기 때문에 호스트의 OS 영향을 거의 받지않음

패키지를 설치하지 않고 도커만 깔고 이미지를 빌드하고 실행만 시켜주면 되기에 배포에 매우 편리함



도커가 없었을때 개발자들 (도커가 매우매우 중요한 이유)

개발자 1

개발 다 했습니다! Git clone 하시고 패키지 12개 다운로드 하시고 실행하시면 됩니다 ~

개발자 2

도커가 없었을때 개발자들 (도커가 매우매우 중요한 이유)

개발자 1
OS: Window

개발 다 했습니다! Git clone 하시고 패키지 12개 다운로드 하시고 실행하시면 됩니다 ~

개발자 2
OS: MacOS

(오류남)

도커가 없었을때 개발자들 (도커가 매우매우 중요한 이유)

개발자 1

에휴… Rocky Linux 10 기준으로 다시 만들고 VM 다운로드 링크 드리겠습니다

개발자 2

도커가 없었을때 개발자들 (도커가 매우매우 중요한 이유)

개발자 1

에휴… Rocky Linux 10 기준으로 다시 만들고 VM 다운로드 링크 드리겠습니다

개발자 2

(프로그램 하나 실행하는데 CPU 80% 차지, 비용 폭탄)

도커가 없었을때 개발자들 (도커가 매우매우 중요한 이유)

개발자 1

에휴… Rocky Linux 10 기준으로 다시 만들고 VM 다운로드 링크 드리겠습니다

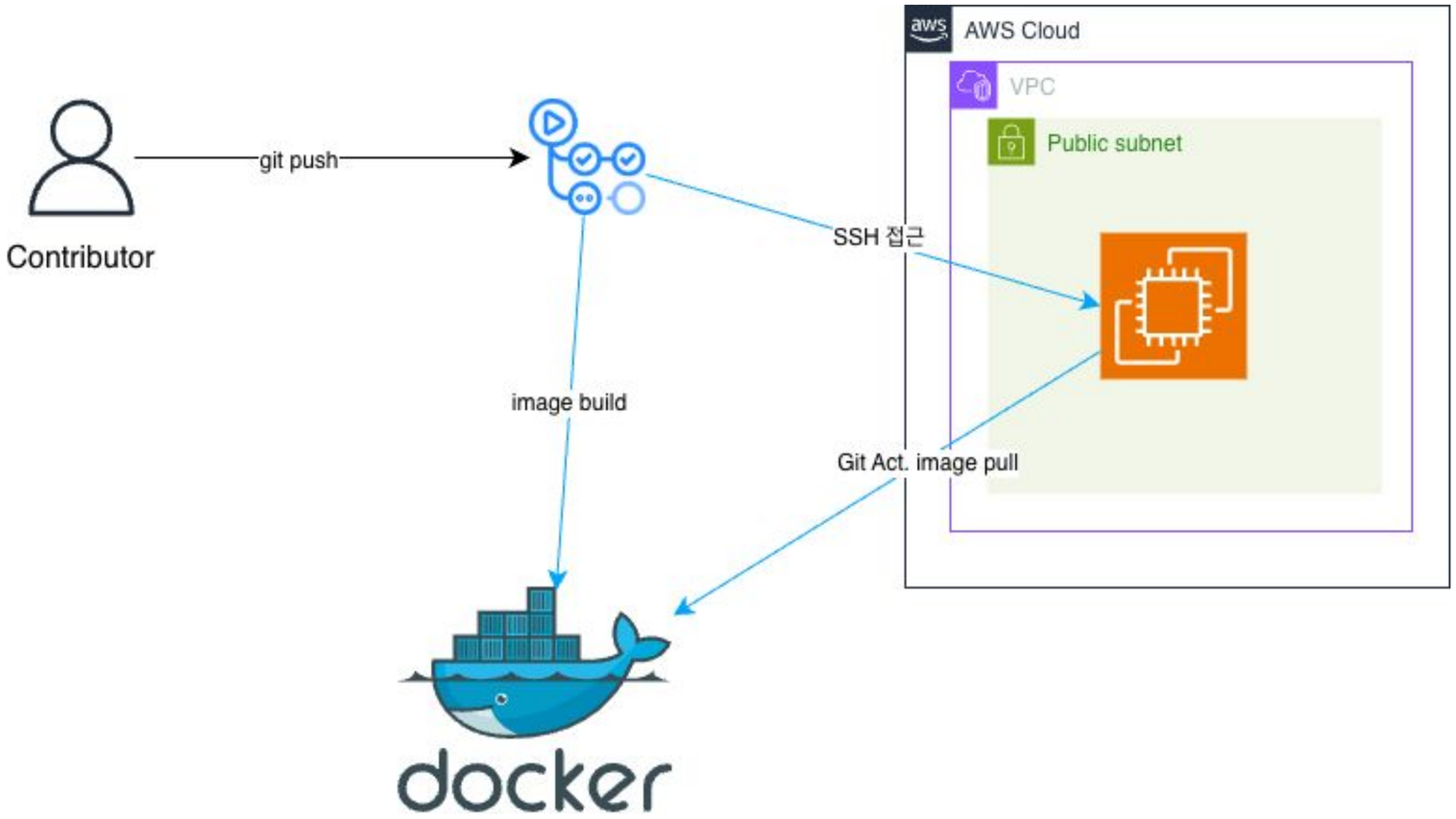
도커는 클라우드, 컨테이너 오케스트레이션을 안배워도 **1000% 필수 무조건 알아야함**.

개발자 2

(프로그램 하나 실행하는데 GPU 80% 차지, 비용 폭탄)

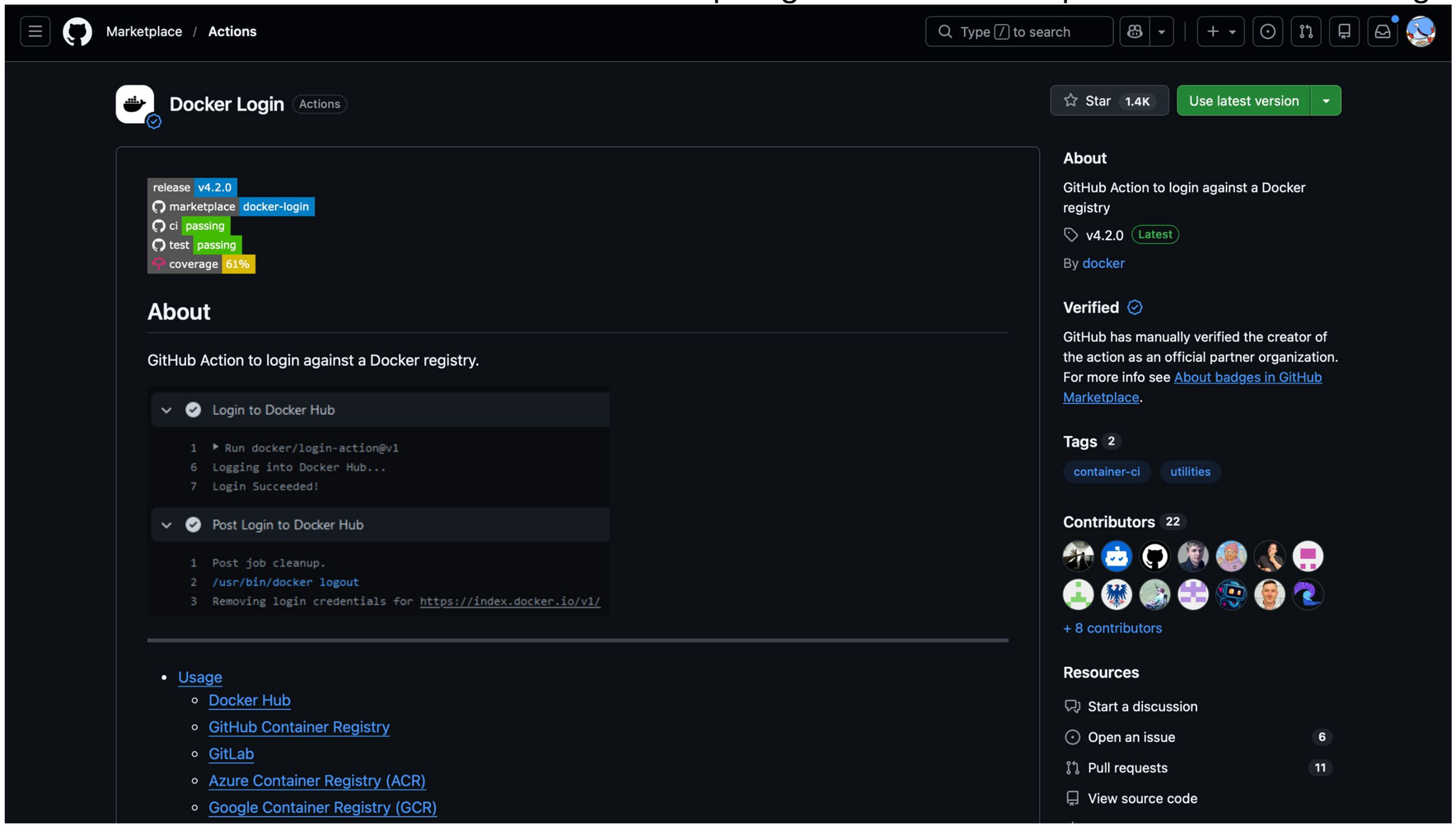
CI/CD 파이프라인

깃허브 액션을 사용하여 도커를 활용한 배포방식



CI/CD 파이프라인

깃허브 액션을 사용하여 도커를 활용한 배포방식 https://github.com/marketplace/actions/docker-login



해당 액션을 사용하여 EC2 내부에서 도커로 로그인 할 수 있도록 사용해줄것이다.

CI/CD 파이프라인

깃허브 액션을 사용하여 도커를 활용한 배포방식

CI

```yaml
name: Deploy NestJS to EC2

on:
  push:
    branches:
      - main

jobs:
  ci:
    name: Continuous Integration
    runs-on: ubuntu-latest

    steps:
      - name: Checkout repository
        uses: actions/checkout@v4

      - name: Setup Node.js
        uses: actions/setup-node@v3
        with:
          node-version: 20

      - name: Install dependencies and run tests
        run: |
          npm ci
          npm run build
          npm test
```

CD

```yaml
  deploy:
    name: Deploy to EC2
    runs-on: ubuntu-latest
    needs: ci

    steps:
      - name: Checkout repository
        uses: actions/checkout@v4

      - name: Build Docker image
        run: |
          docker build -t your-dockerhub-username/nest-cicd-demo:latest .

      - name: Log in to Docker Hub
        uses: docker/login-action@v2
        with:
          username: ${{ secrets.DOCKER_USERNAME }}
          password: ${{ secrets.DOCKER_PASSWORD }}

      - name: Push Docker image to Docker Hub
        run: |
          docker push your-dockerhub-username/nest-cicd-demo:latest

      - name: SSH and deploy to EC2
        uses: appleboy/ssh-action@v0.1.6
        with:
          host: ${{ secrets.EC2_HOST }}
          username: ${{ secrets.EC2_USER }}
          key: ${{ secrets.EC2_SSH_KEY }}
          script: |
            docker pull your-dockerhub-username/nest-cicd-demo:latest
            docker stop nest-cicd-demo || true
            docker rm nest-cicd-demo || true
            docker run -d --name nest-cicd-demo -p 80:3000 your-dockerhub-username/
nest-cicd-demo:latest
```

CI부분은 다른게 별로 없지만 CD부분에서 큰 차이를 볼 수 있음

CI/CD 파이프라인

깃허브 액션을 사용하여 도커를 활용한 배포방식

```
username: ${{ secrets.DOCKER_USERNAME }}
password: ${{ secrets.DOCKER_PASSWORD }}
```

도커 유저네임 + 토큰을 가져와야함

Personal access tokens

You can use a personal access token instead of a password for Docker CLI authentication. Create multiple tokens, control their scope, and delete tokens at any time. Learn more

Generate new token

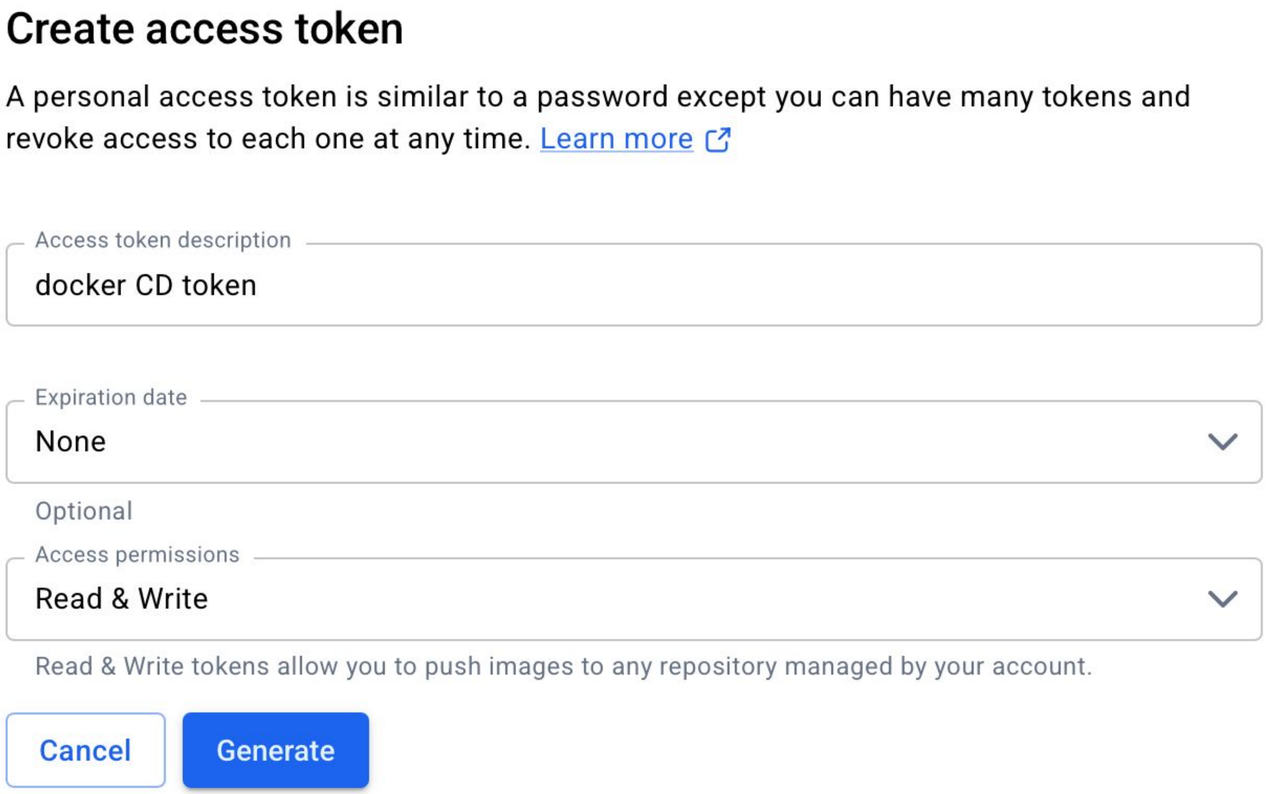
Description	Scope	Status	Source	Created	Last used	Expiration date
Generated through d…	Read, Write, Delete	Active	Auto-generated	Jun 22, 2026 at 00:08:23	Jun 22, 2026 at 00:13:11	Never
Generated through d…	Read, Write, Delete	Active	Auto-generated	Mar 05, 2026 at 23:38:10	Mar 05, 2026 at 23:38:12	Never

Rows per page: 10 1–2 of 2

깃허브 액션을 사용하여 도커를 활용한 배포방식

```
username: ${{ secrets.DOCKER_USERNAME }}
password: ${{ secrets.DOCKER_PASSWORD }}
```

도커 유저네임 + 토큰을 가져와야함



깃허브 액션을 사용하여 도커를 활용한 배포방식

```
username: ${{ secrets.DOCKER_USERNAME }}
password: ${{ secrets.DOCKER_PASSWORD }}
```

도커 유저네임 + 토큰을 가져와야함

To use the access token from your Docker CLI client:

1. Run

```
$  docker login -u m161awm
```

Copy

2. At the password prompt, enter the personal access token.

토큰 있는곳

Copy

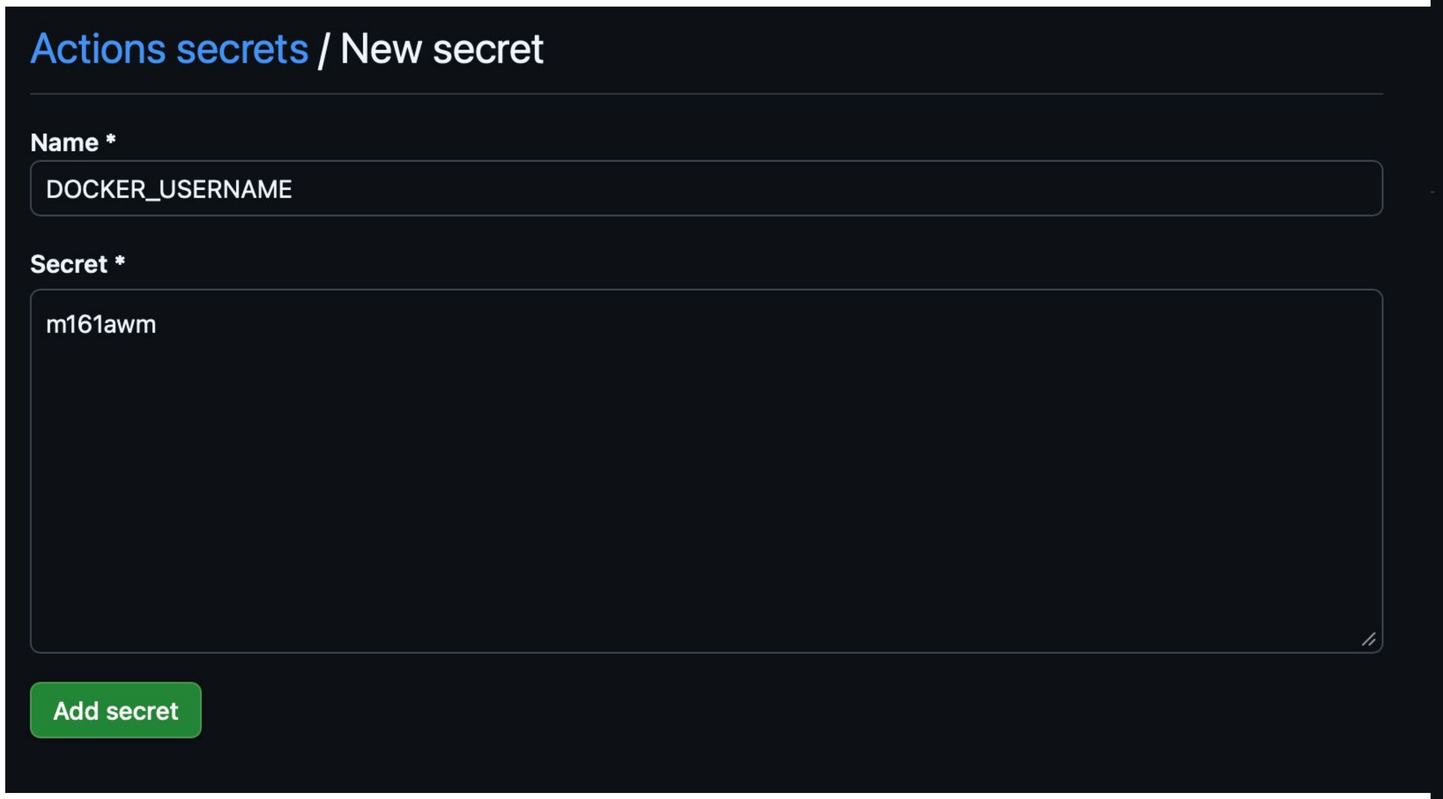
Back to access tokens

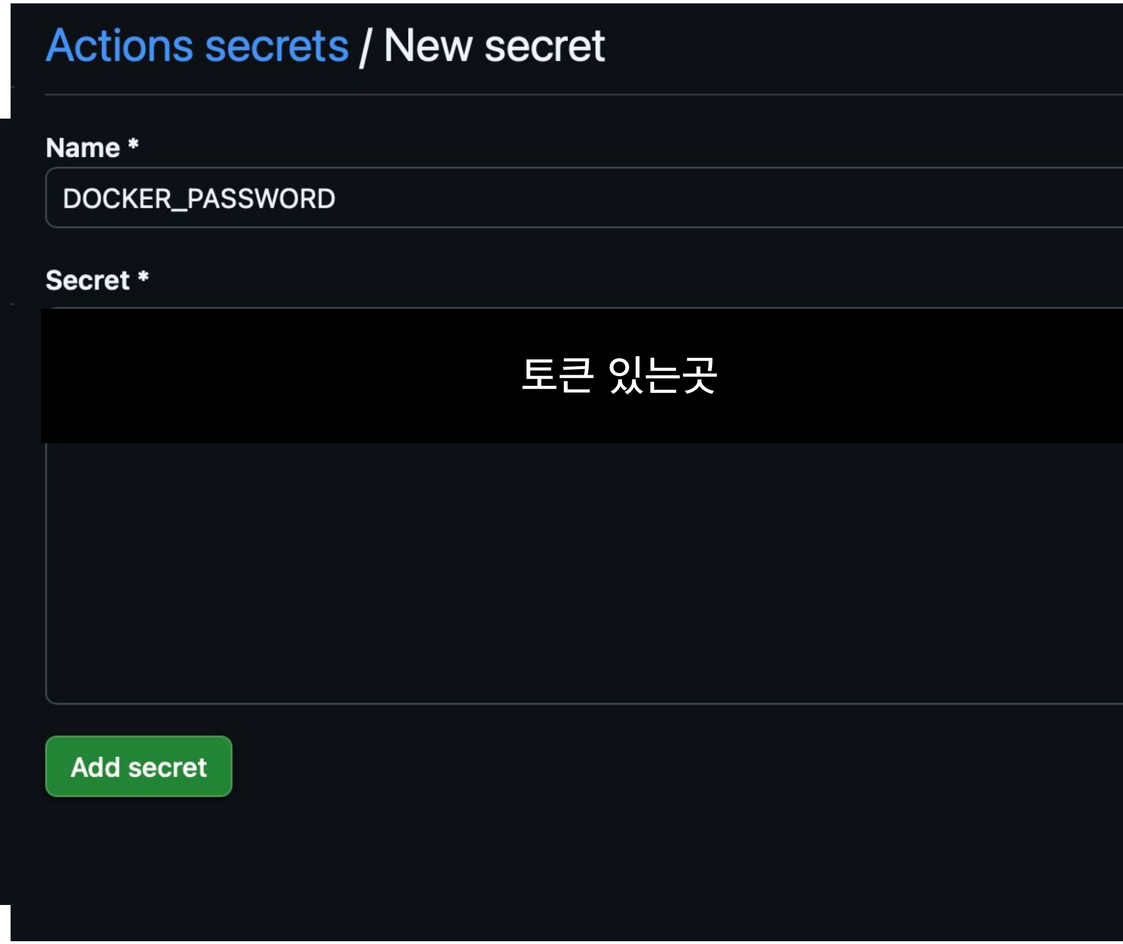
CI/CD 파이프라인

깃허브 액션을 사용하여 도커를 활용한 배포방식

```
username: ${{ secrets.DOCKER_USERNAME }}
password: ${{ secrets.DOCKER_PASSWORD }}
```

도커 유저네임 + 토큰을 가져와야함



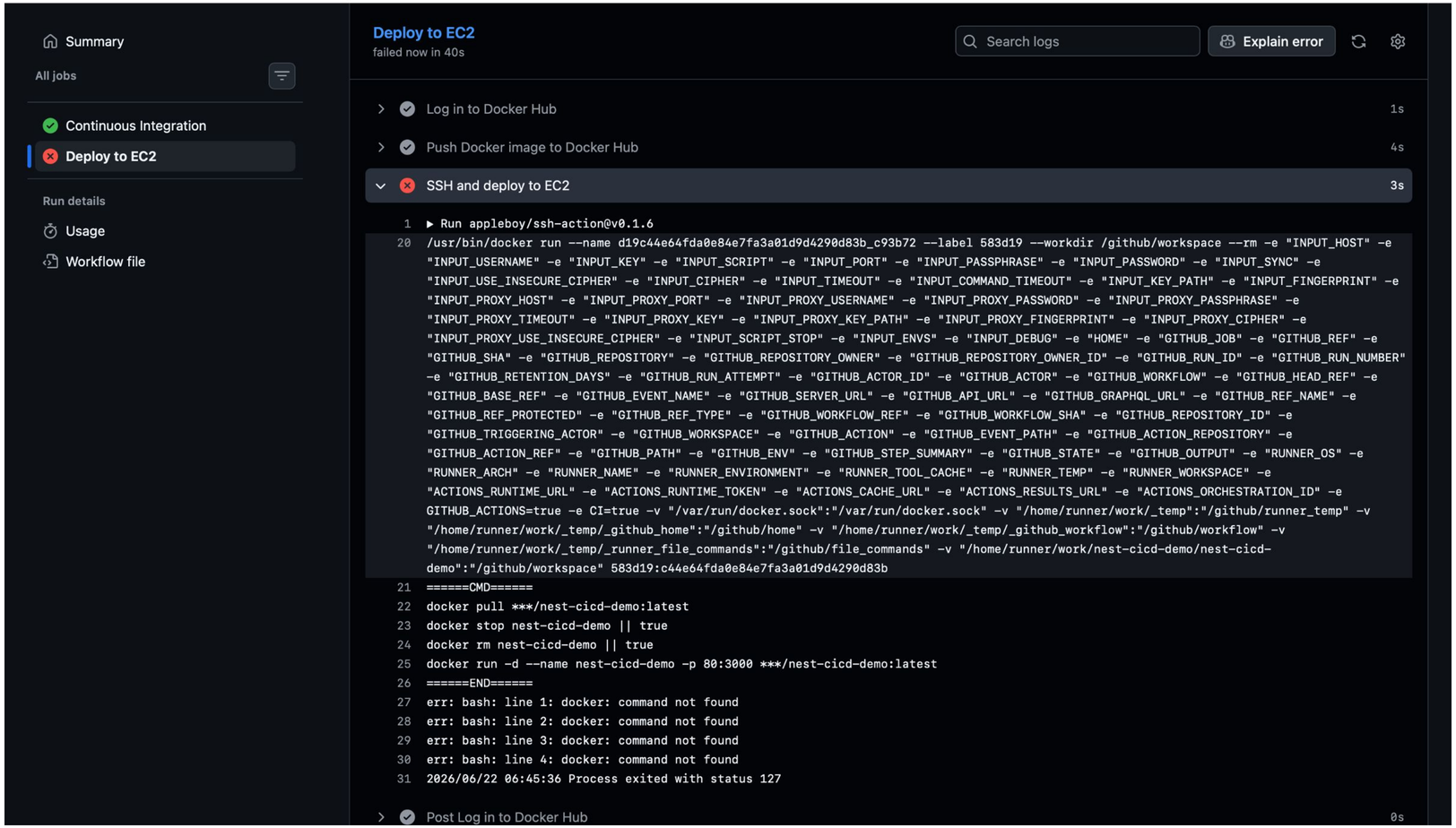


깃허브 액션을 사용하여 도커를 활용한 배포방식

```
admin@m161awmui-MacBookPro nest-cicd-demo % git add .
admin@m161awmui-MacBookPro nest-cicd-demo % git commit -m "changed name"
[main 0614b0e] changed name
 1 file changed, 4 insertions(+), 8 deletions(-)
admin@m161awmui-MacBookPro nest-cicd-demo % git push
Enumerating objects: 9, done.
Counting objects: 100% (9/9), done.
Delta compression using up to 10 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (5/5), 425 bytes | 425.00 KiB/s, done.
Total 5 (delta 2), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/m161awm2/nest-cicd-demo
   341dc3f..0614b0e  main -> main
admin@m161awmui-MacBookPro nest-cicd-demo %
```

CI/CD 파이프라인

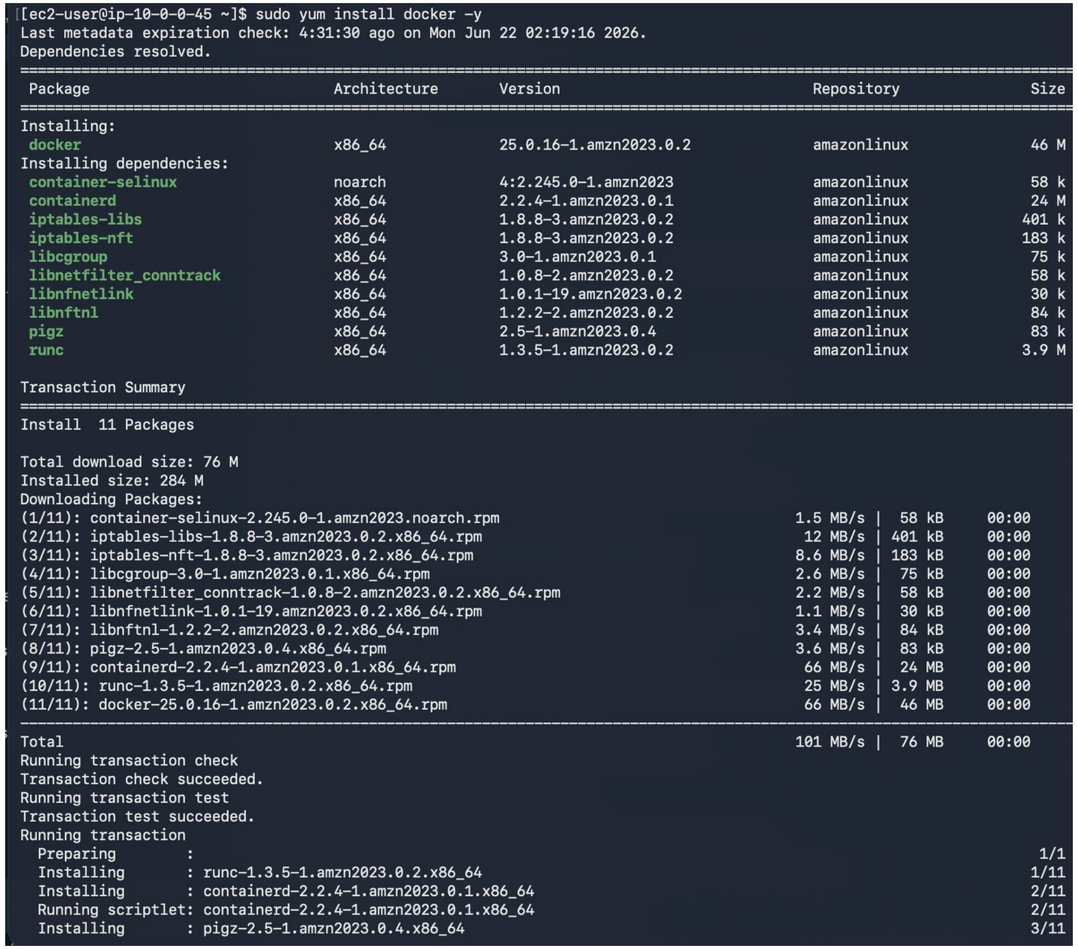
깃허브 액션을 사용하여 도커를 활용한 배포방식



배포 부분에 오류가 발생하였다 원인을 보니깐 docker: command not found.. 도커가 EC2에 안깔려있는가보다.

CI/CD 파이프라인

깃허브 액션을 사용하여 도커를 활용한 배포방식



```
[ec2-user@ip-10-0-0-45 ~]$ sudo yum install docker -y
Last metadata expiration check: 4:31:30 ago on Mon Jun 22 02:19:16 2026.
Dependencies resolved.
================================================================================
 Package                 Architecture  Version                    Repository  Size
================================================================================
Installing:
 docker                  x86_64        25.0.16-1.amzn2023.0.2     amazonlinux  46 M
Installing dependencies:
 container-selinux       noarch        4:2.245.0-1.amzn2023       amazonlinux  58 k
 containerd              x86_64        2.2.4-1.amzn2023.0.1       amazonlinux  24 M
 iptables-libs           x86_64        1.8.8-3.amzn2023.0.2       amazonlinux 401 k
 iptables-nft            x86_64        1.8.8-3.amzn2023.0.2       amazonlinux 183 k
 libcgroup               x86_64        3.0-1.amzn2023.0.1         amazonlinux  75 k
 libnetfilter_conntrack  x86_64        1.0.8-2.amzn2023.0.2       amazonlinux  58 k
 libnfnetlink            x86_64        1.0.1-19.amzn2023.0.2      amazonlinux  30 k
 libnftnl                x86_64        1.2.2-2.amzn2023.0.2       amazonlinux  84 k
 pigz                    x86_64        2.5-1.amzn2023.0.4         amazonlinux  83 k
 runc                    x86_64        1.3.5-1.amzn2023.0.2       amazonlinux 3.9 M

Transaction Summary
================================================================================
Install  11 Packages

Total download size: 76 M
Installed size: 284 M
Downloading Packages:
(1/11): container-selinux-2.245.0-1.amzn2023.noarch.rpm      1.5 MB/s |  58 kB     00:00
(2/11): iptables-libs-1.8.8-3.amzn2023.0.2.x86_64.rpm         12 MB/s | 401 kB     00:00
(3/11): iptables-nft-1.8.8-3.amzn2023.0.2.x86_64.rpm         8.6 MB/s | 183 kB     00:00
(4/11): libcgroup-3.0-1.amzn2023.0.1.x86_64.rpm              2.6 MB/s |  75 kB     00:00
(5/11): libnetfilter_conntrack-1.0.8-2.amzn2023.0.2.x86_64.rpm 2.2 MB/s |  58 kB     00:00
(6/11): libnfnetlink-1.0.1-19.amzn2023.0.2.x86_64.rpm        1.1 MB/s |  30 kB     00:00
(7/11): libnftnl-1.2.2-2.amzn2023.0.2.x86_64.rpm             3.4 MB/s |  84 kB     00:00
(8/11): pigz-2.5-1.amzn2023.0.4.x86_64.rpm                   3.6 MB/s |  83 kB     00:00
(9/11): containerd-2.2.4-1.amzn2023.0.1.x86_64.rpm            66 MB/s |  24 MB     00:00
(10/11): runc-1.3.5-1.amzn2023.0.2.x86_64.rpm                 25 MB/s | 3.9 MB     00:00
(11/11): docker-25.0.16-1.amzn2023.0.2.x86_64.rpm             66 MB/s |  46 MB     00:00
--------------------------------------------------------------------------------
Total                                                        101 MB/s |  76 MB     00:00
Running transaction check
Transaction check succeeded.
Running transaction test
Transaction test succeeded.
Running transaction
  Preparing        :                                                        1/1
  Installing       : runc-1.3.5-1.amzn2023.0.2.x86_64                      1/11
  Installing       : containerd-2.2.4-1.amzn2023.0.1.x86_64                2/11
  Running scriptlet: containerd-2.2.4-1.amzn2023.0.1.x86_64                2/11
  Installing       : pigz-2.5-1.amzn2023.0.4.x86_64                        3/11
```

sudo yum install docker -y ec2에서 도커를 쭉 깔아주겠다.

CI/CD 파이프라인

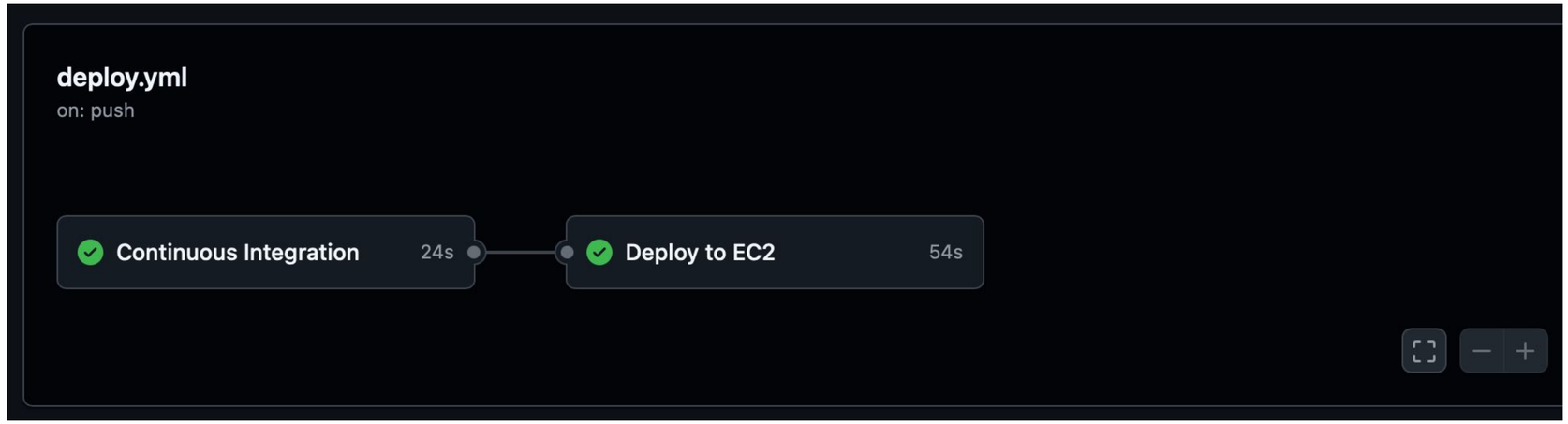
깃허브 액션을 사용하여 도커를 활용한 배포방식

```
[ec2-user@ip-10-0-0-45 nest-cicd-demo]$ cd ..
[ec2-user@ip-10-0-0-45 ~]$ sudo systemctl start docker
[ec2-user@ip-10-0-0-45 ~]$ sudo systemctl enable docker
Created symlink /etc/systemd/system/multi-user.target.wants/docker.service → /usr/lib/systemd/system/docker.service.
[ec2-user@ip-10-0-0-45 ~]$ sudo usermod -aG docker $USER
[ec2-user@ip-10-0-0-45 ~]$ newgrp docker
[ec2-user@ip-10-0-0-45 ~]$
```

시스템 컨트롤을 사용하여 도커 켜놓는 상태를 계속 유지시켜야한다.
방금 추가한 Docker 그룹 권한을 바로 적용해준다. 로그아웃했다가 다시 접속하지 않고도, 방금 추가한 Docker 그룹 권한을 바로 적용한다.

CI/CD 파이프라인

깃허브 액션을 사용하여 도커를 활용한 배포방식



CD가 완료되었다. 접근해보자

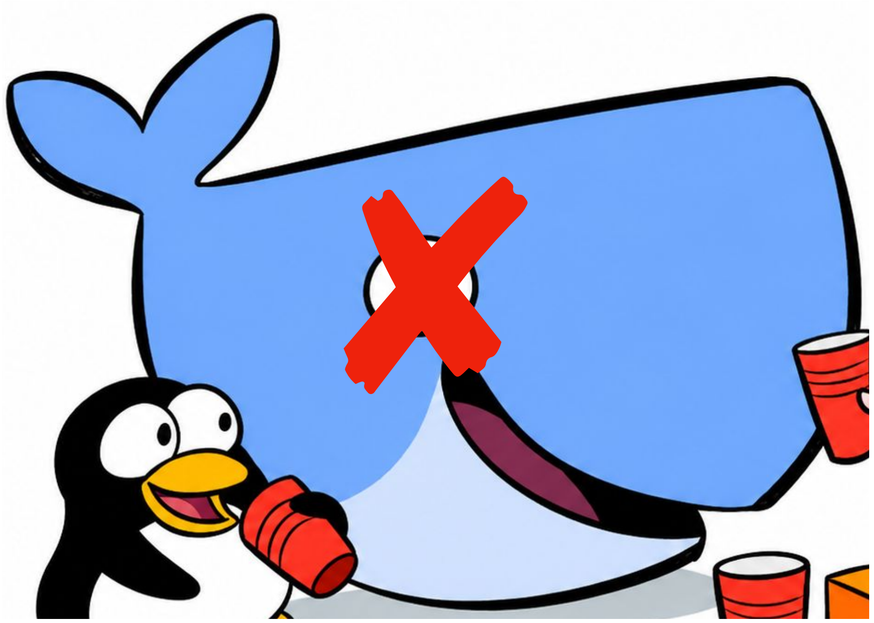
CI/CD 파이프라인

깃허브 액션을 사용하여 도커를 활용한 배포방식

배포가 성공적으로 일어난걸 알 수 있다.

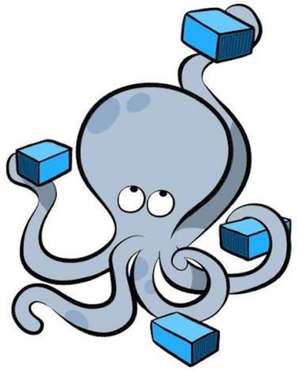
kubernetes란?

컨테이너에 사람이 많이 들어오면 도커 컨테이너가 죽을수도 있음



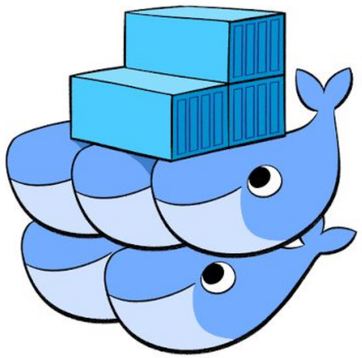
kubernetes란?

컨테이너에 사람이 많이 들어오면 도커 컨테이너가 죽을수도 있음



Docker compose

Mesos

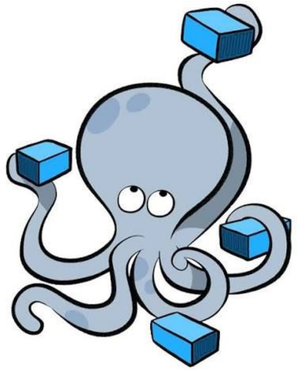


Docker swarm

컨테이너 오케스트레이션 도구들

Nomad

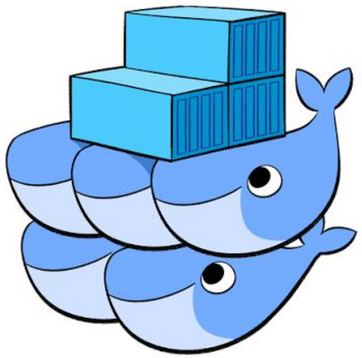
kubernetes란?



Docker compose

구글의 쿠버네티스

Mesos



Docker swarm

컨테이너 오케스트레이션 도구들

Nomad

단점 : k8s는 정말 어렵다…

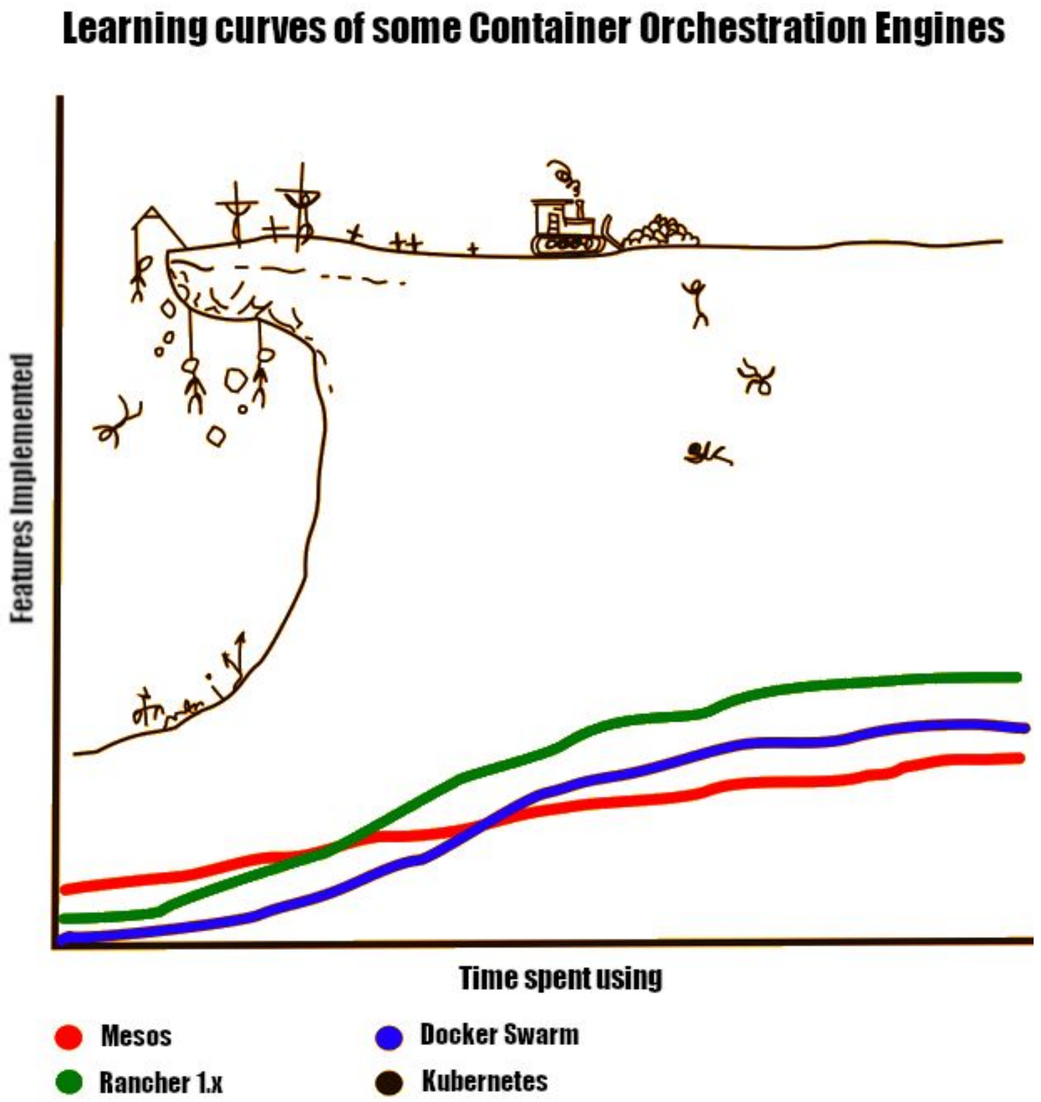
가장 어려운 it 도구 top10

AI 모드 | 전체 | 이미지 | 동영상 | 쇼핑 | 뉴스 | 짧은 동영상 | 더보기 | 도구

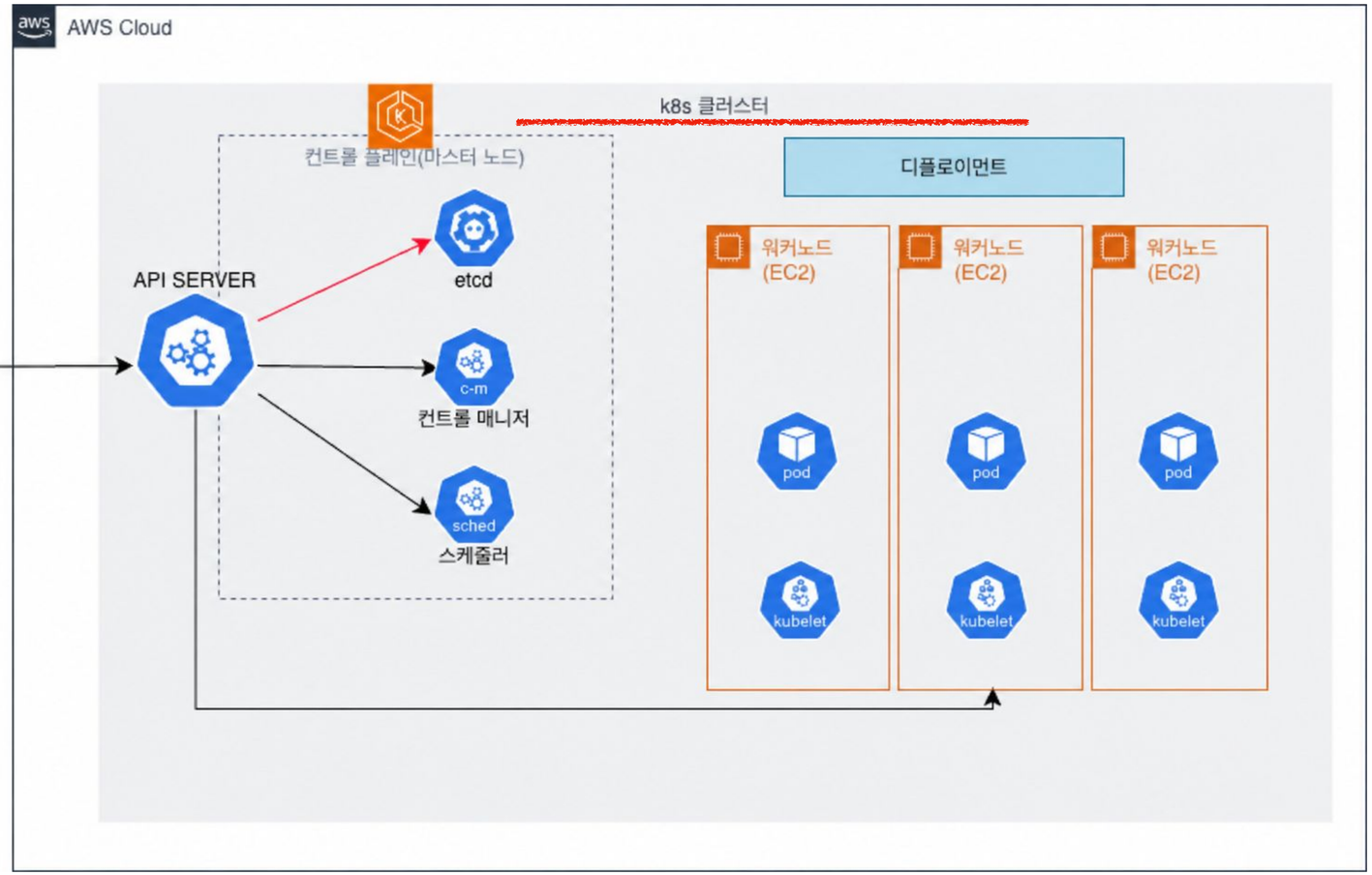
AI 개요

가파른 학습 곡선(Learning Curve)과 복잡한 아키텍처로 인해 IT 전문가들 사이에서도 습득하기 가장 어렵다고 평가받는 도구 및 기술 TOP 10입니다.

1. **말레볼제(Malbolge):** 고의적으로 복잡하게 설계된 난해한 프로그래밍 언어로, 암호학적 구조와 코드 자체 변조 방식을 사용하여 해독이 거의 불가능합니다. www.cwn.kr
2. **하스켈(Haskell):** 순수 함수형 프로그래밍 언어로, 모나드(Monad)와 같은 고도의 추상적 수학 개념을 이해해야만 다룰 수 있어 입문자에게 진입장벽이 높습니다. letspl.me +1
3. **C++:** 객체지향과 절차적 프로그래밍을 모두 지원하지만, 방대한 기능과 메모리 직접 관리(포인터) 등으로 인해 마스터하기 가장 복잡한 언어 중 하나로 꼽힙니다. letspl.me
4. **프롤로그(Prolog):** 논리형 프로그래밍 언어로, 일반적인 절차적 사고방식을 버리고 규칙과 사실을 기반으로 한 선언적 추론 개념을 익혀야 합니다. letspl.me +1
5. **쿠버네티스(Kubernetes):** 컨테이너 오케스트레이션 도구로, 방대한 YAML 설정 파일, 클러스터 네트워킹, 복잡한 마이크로서비스 아키텍처를 모두 이해해야 구축 및 운영이 가능합니다.
6. **APL:** 특수 기호 기반의 고도로 함축된 프로그래밍 도구로, 전용 키보드가 필요할 정도의 직관적이지 않은 문법을 가집니다.
7. **비욘드컴페어(Beyond Compare):** 단순한 파일 비교 툴을 넘어 코드 병합과 구조적 차이점을 완벽히 제어하려면 심도 있는 도메인 지식이 필요합니다.

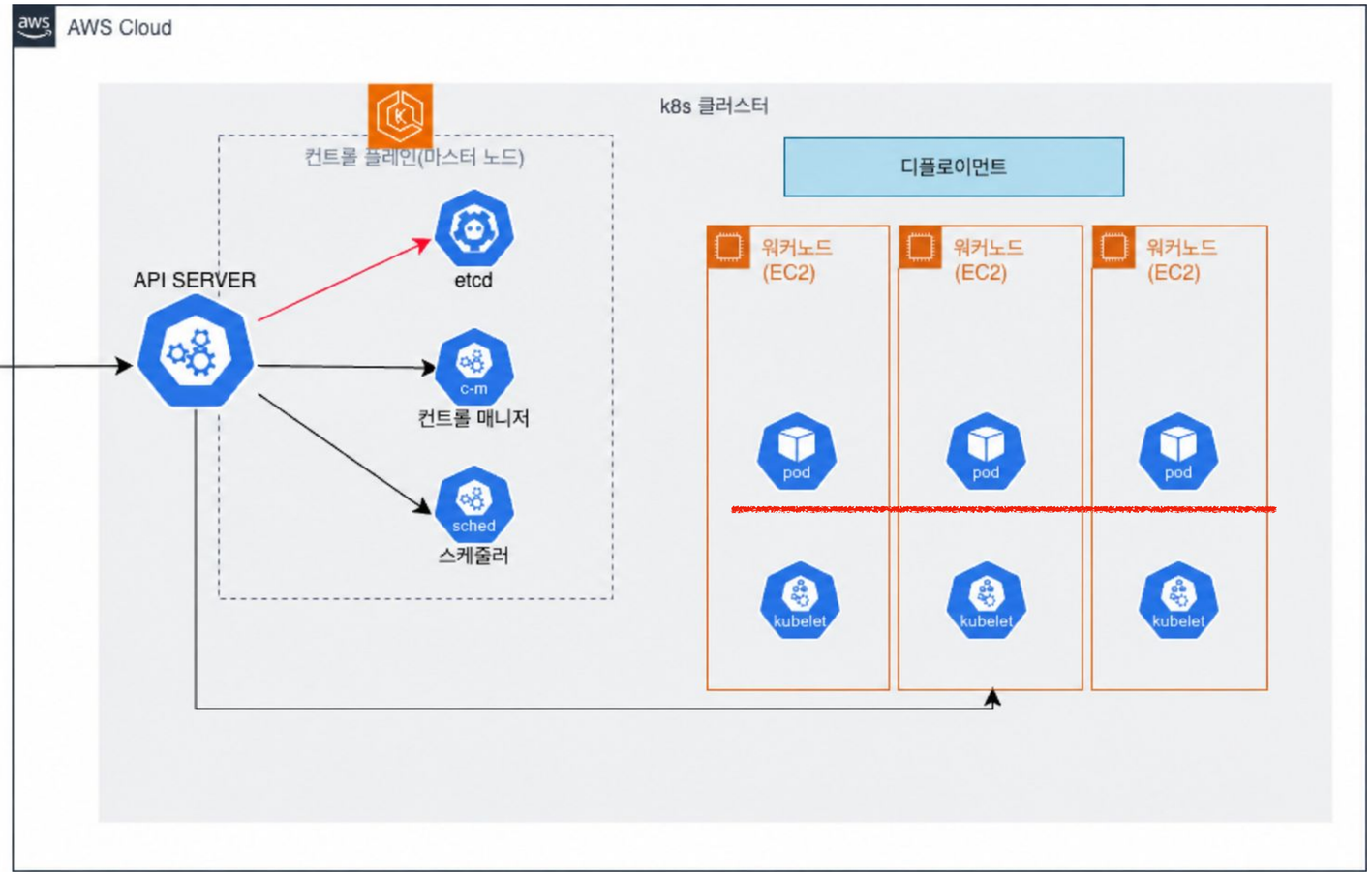


k8s의 구성요소와 배포 과정



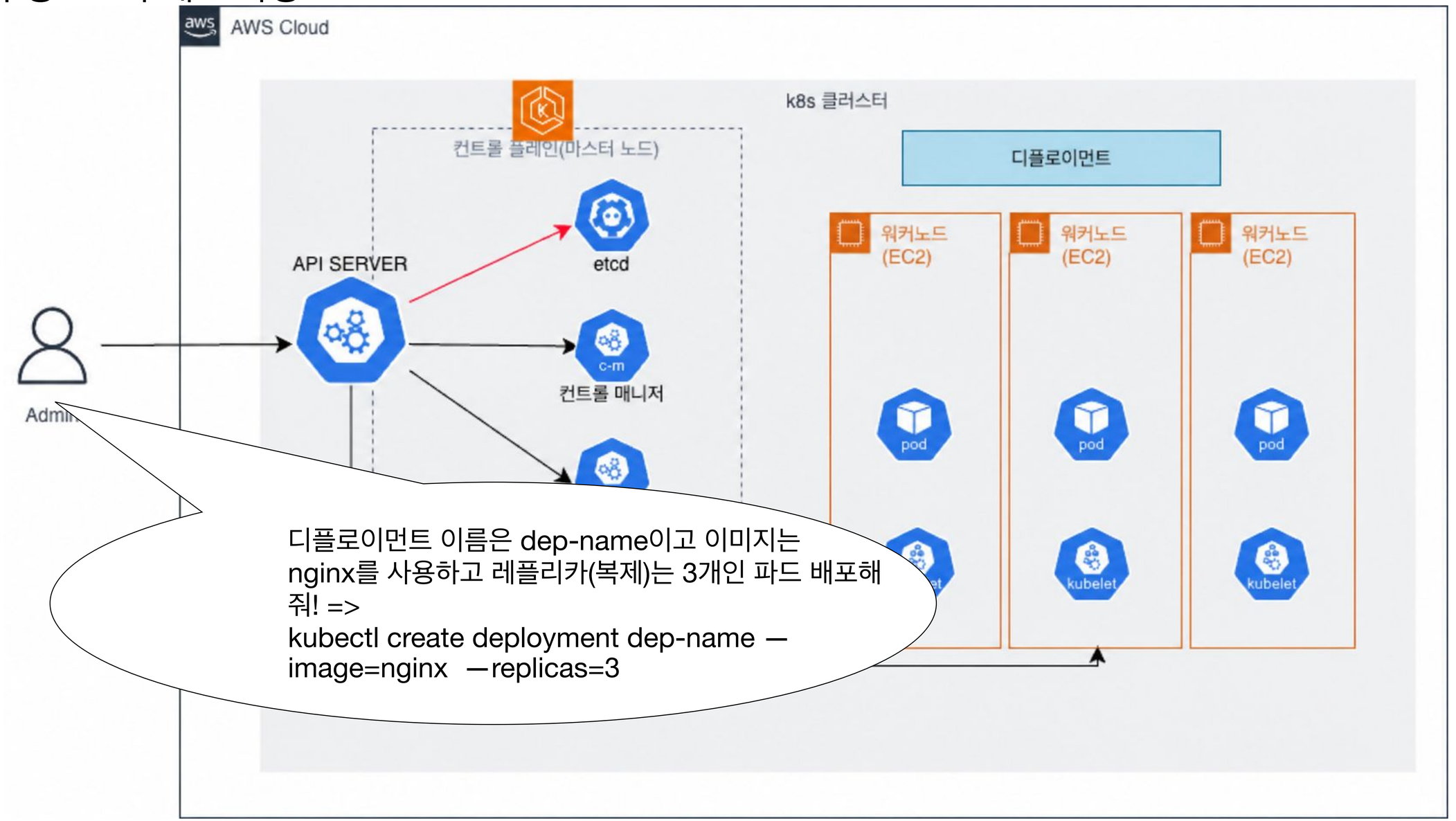
쿠버네티스 클러스터는 컨테이너 환경을 배포하기 위해 만들어짐

k8s의 구성요소와 배포 과정

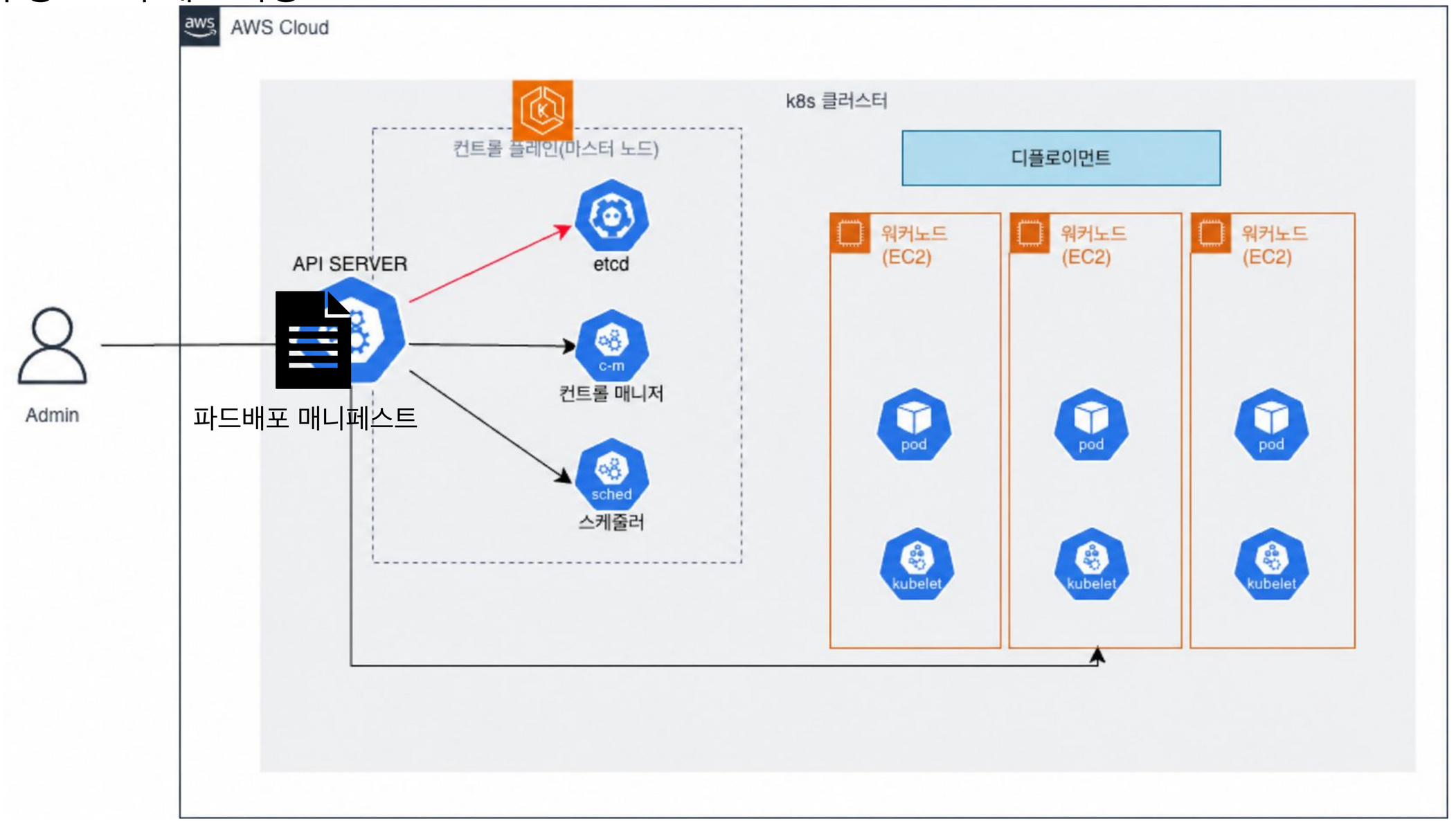


파드 : 배포가 가능한 컨테이너, EKS기준으로 EC2 인스턴스에 배포되며 컨테이너 인프라 환경에선 해당 인스턴스를 워커 노드라고 부름.

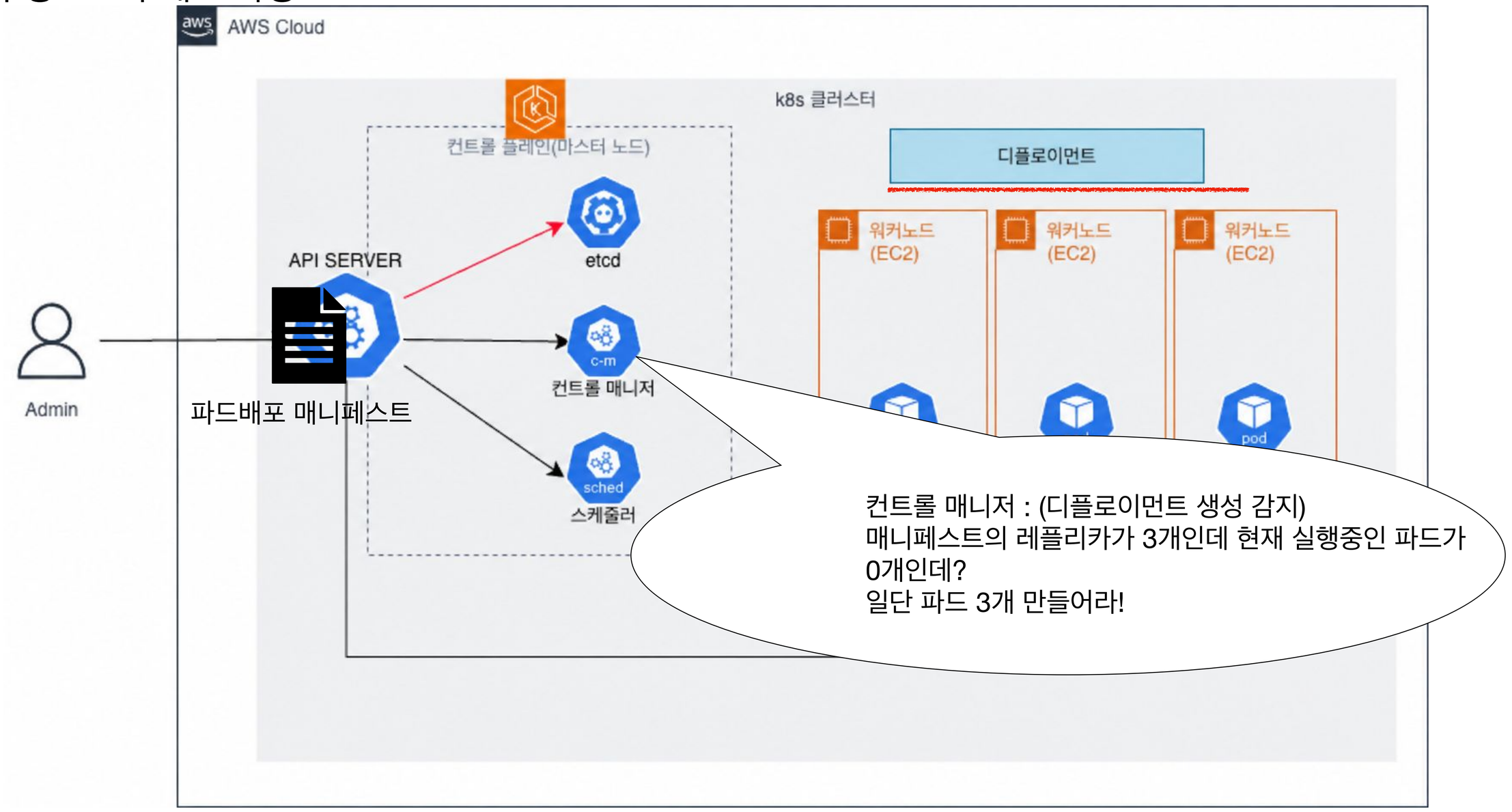
k8s의 구성요소와 배포 과정



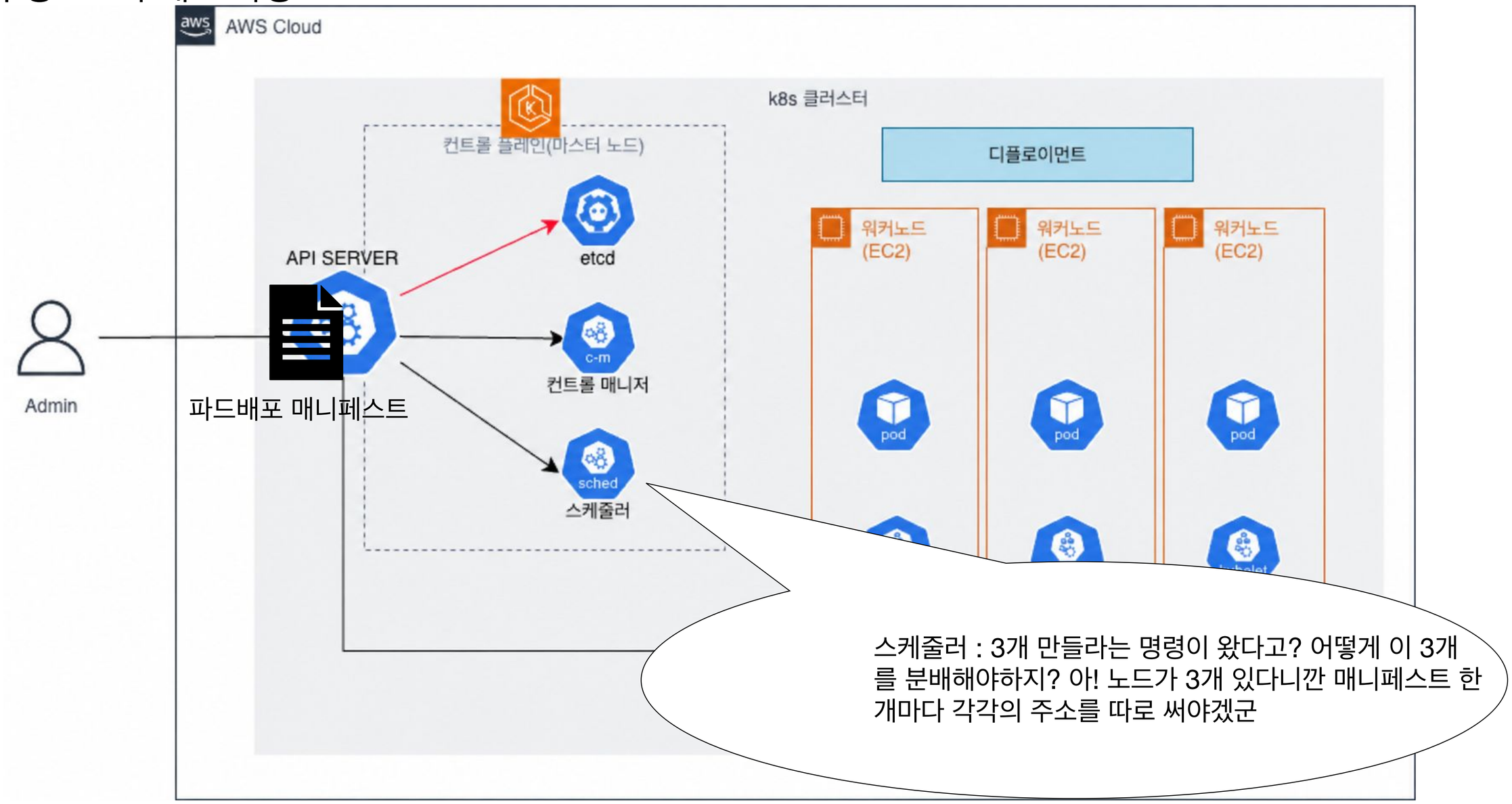
k8s의 구성요소와 배포 과정



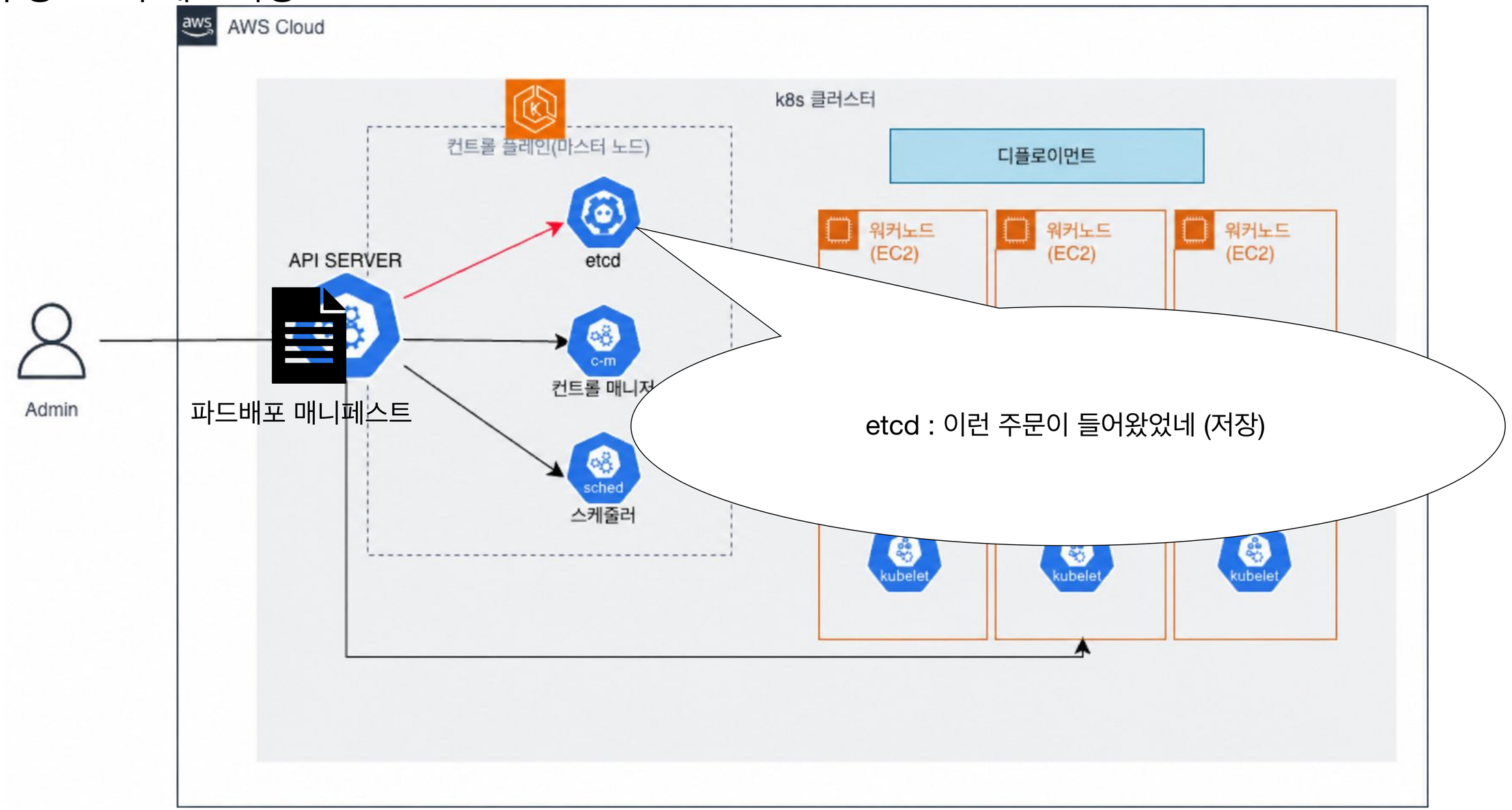
k8s의 구성요소와 배포 과정



k8s의 구성요소와 배포 과정

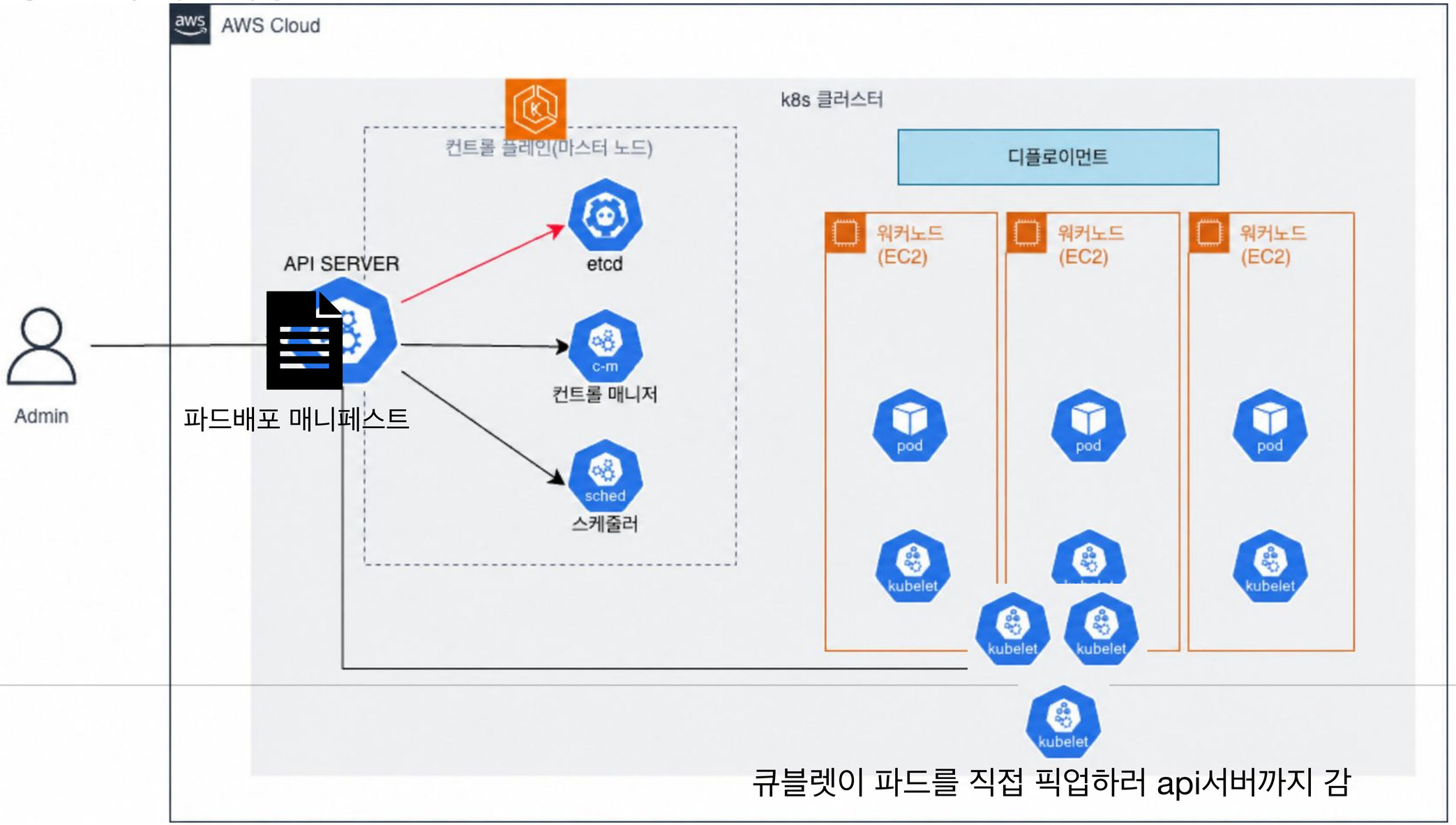


k8s의 구성요소와 배포 과정



kubernetes란?

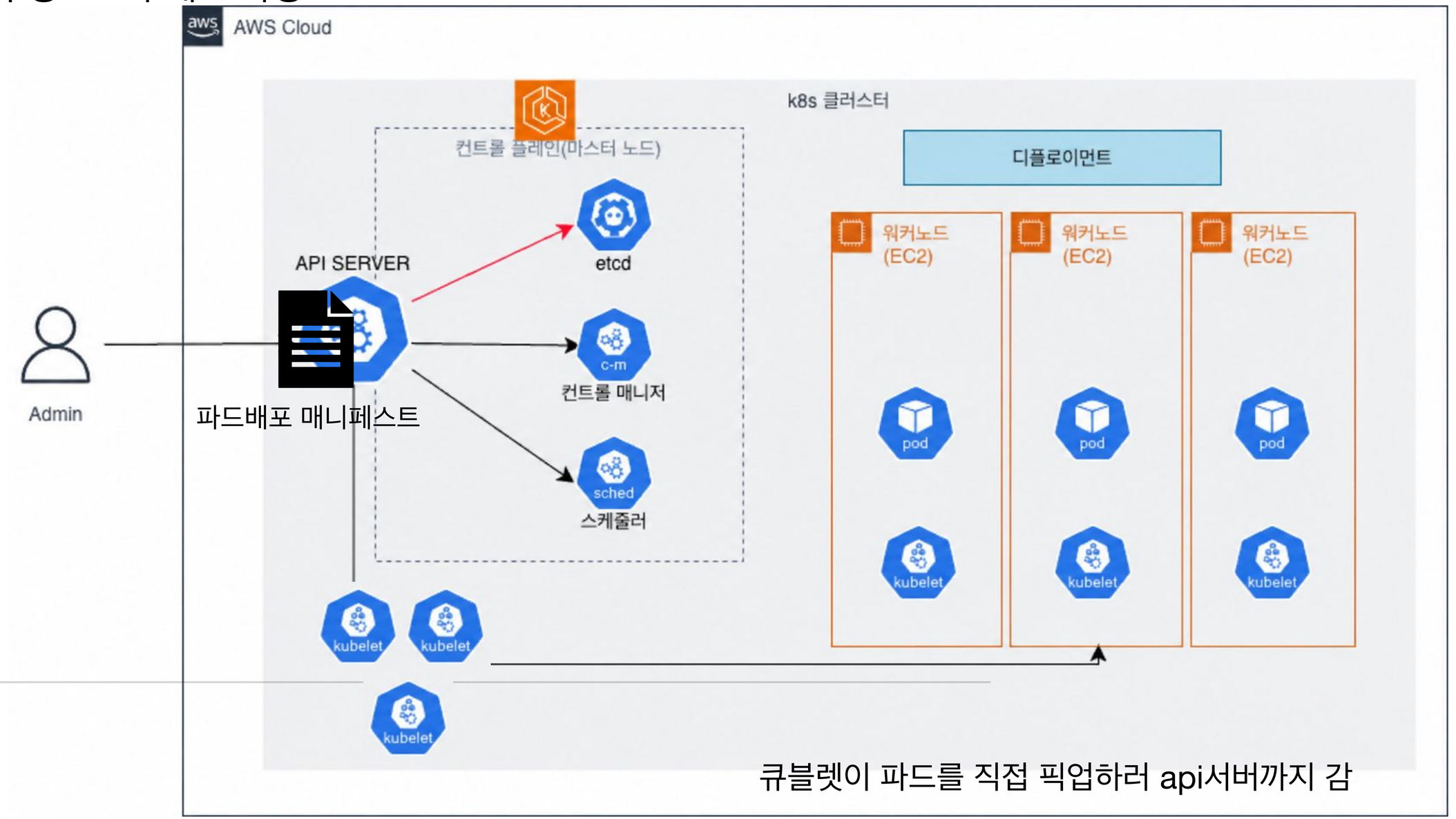
k8s의 구성요소와 배포 과정



큐블렛이 파드를 직접 픽업하러 api서버까지 감

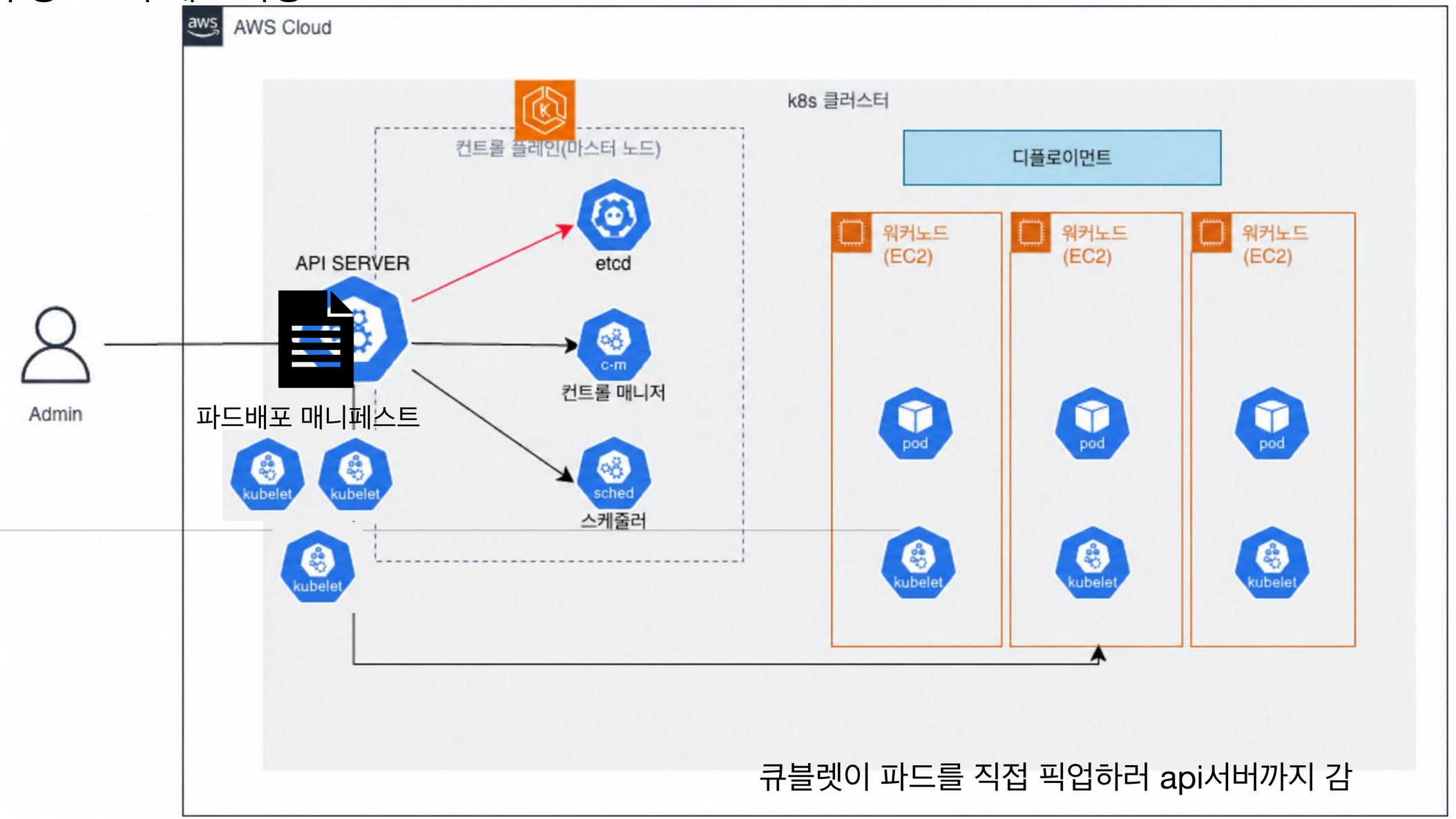
kubernetes란?

k8s의 구성요소와 배포 과정



kubernetes란?

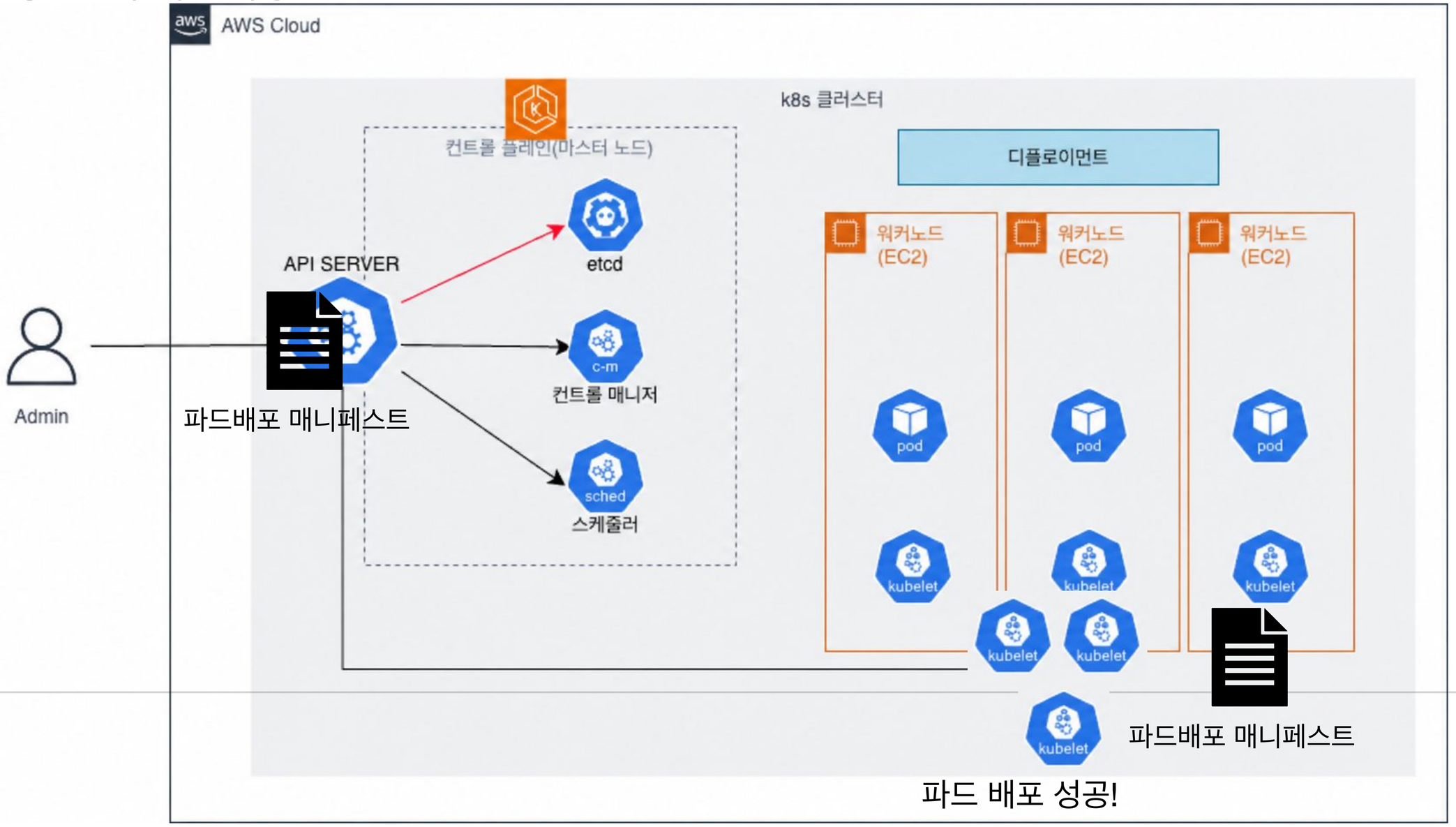
k8s의 구성요소와 배포 과정



큐블렛이 파드를 직접 픽업하러 api서버까지 감

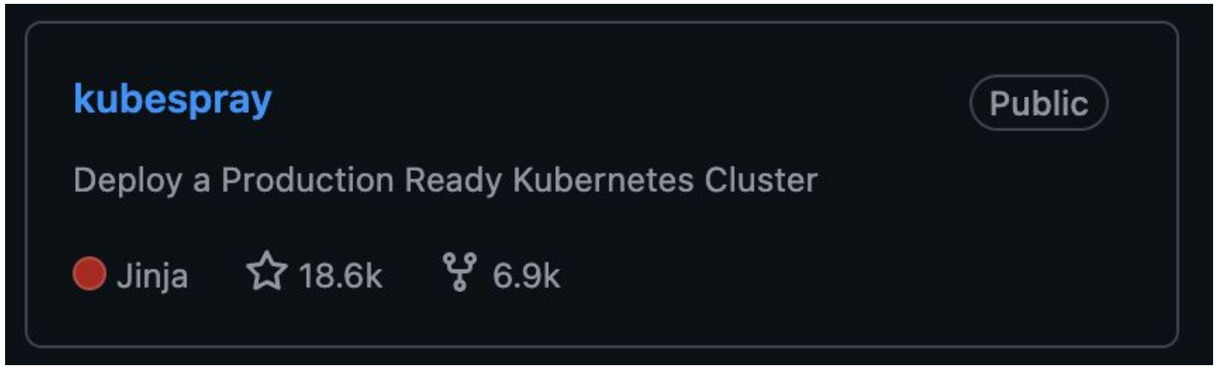
kubernetes란?

k8s의 구성요소와 배포 과정



k8s 사용해보기

직접 k8s를 설치하는 방법

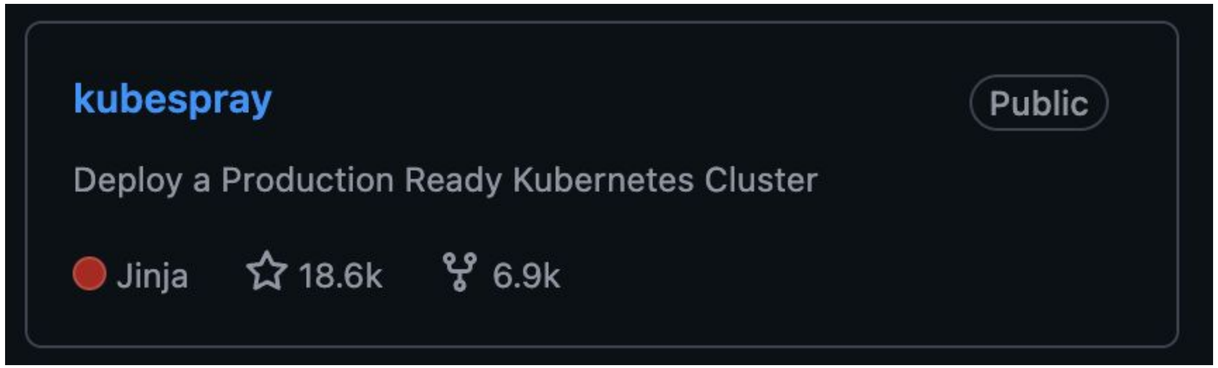


https://github.com/kubernetes-sigs/kubespray

k8s 사용해보기

채감상 30분 이상 걸림 + 메모리 엄청 잡아먹는다..

직접 k8s를 설치하는 방법



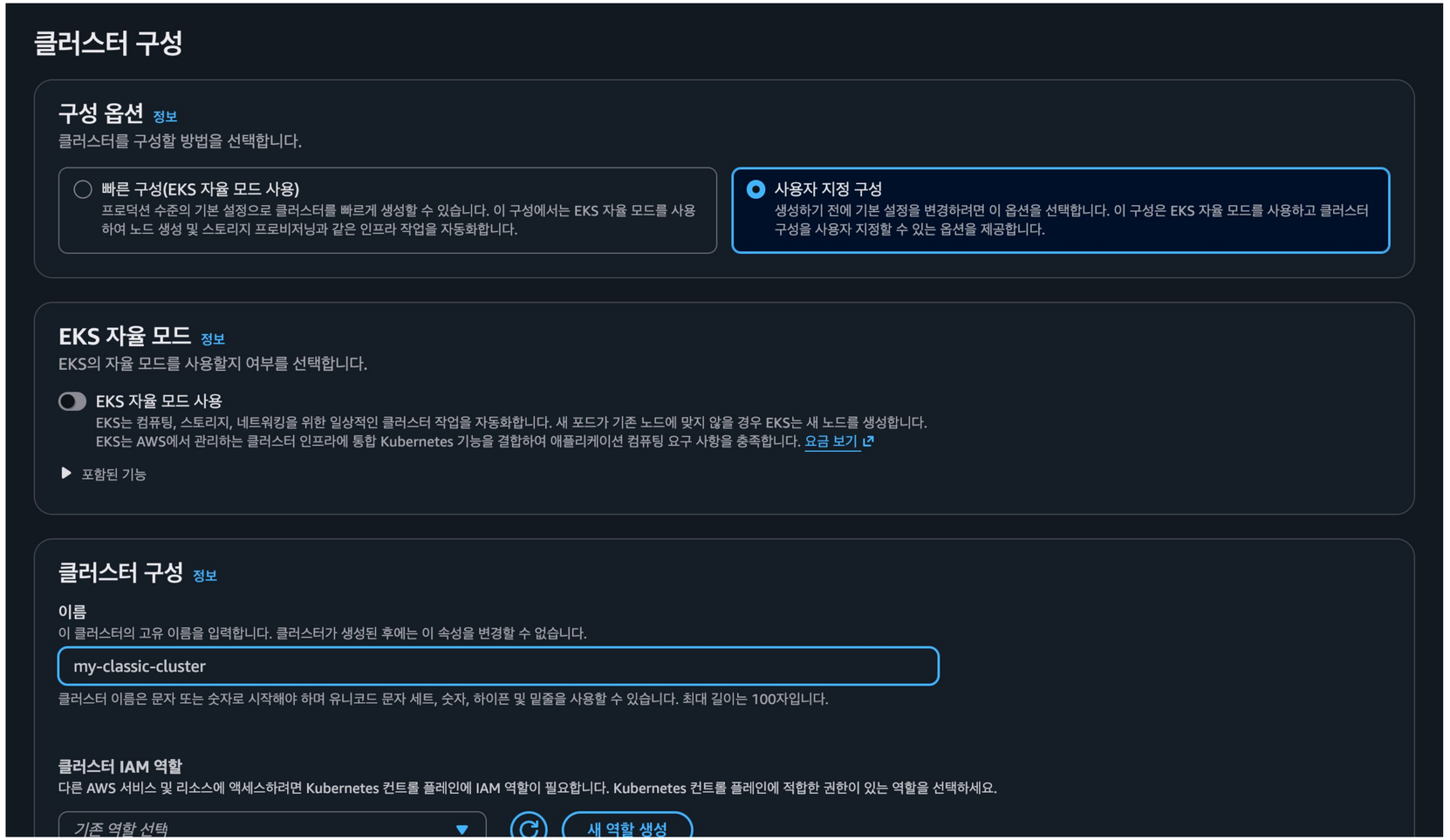
https://github.com/kubernetes-sigs/kubespray

k8s 사용해보기

AWS에서는 쿠버네티스가 미리 깔려있다!

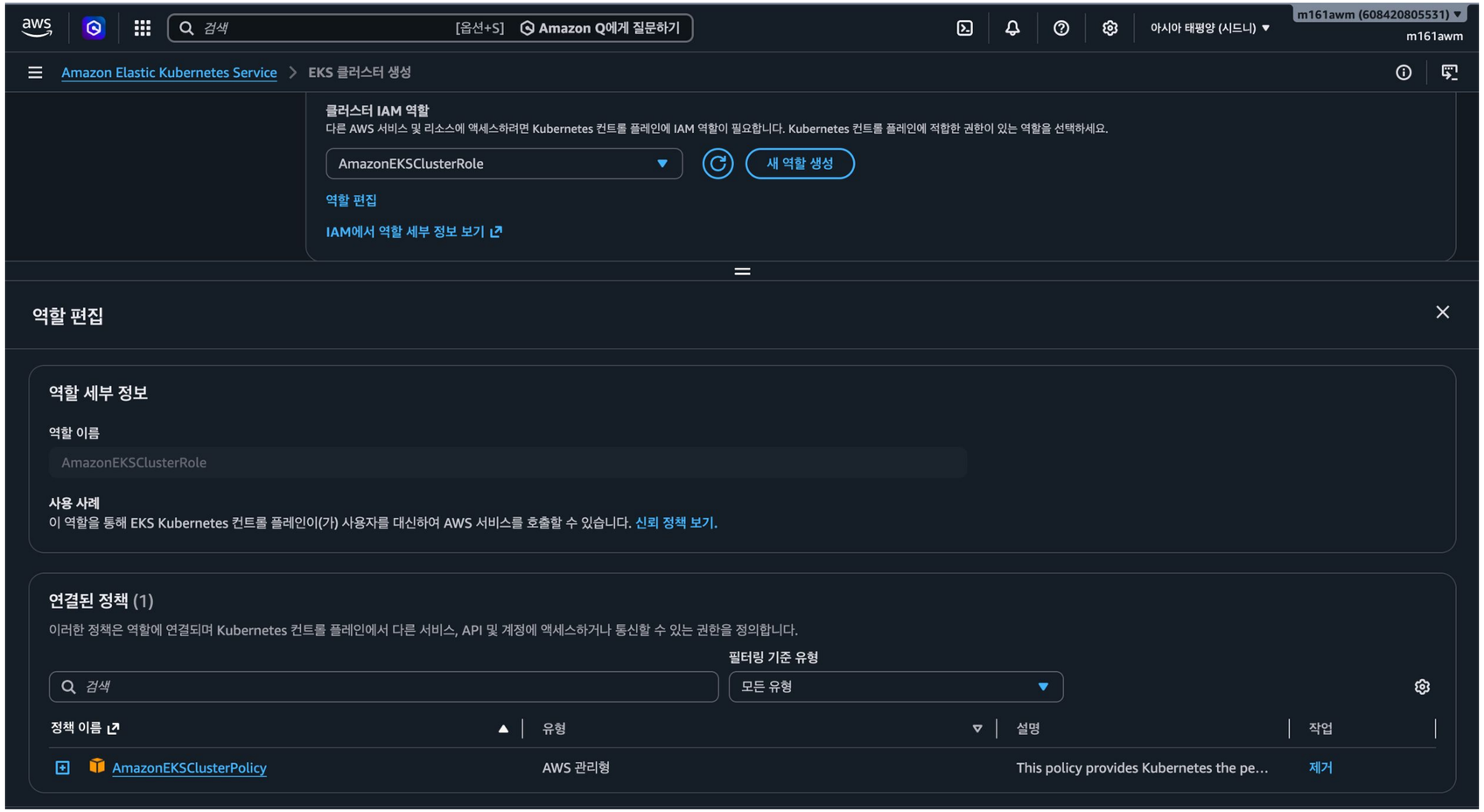
AWS의 elastic kubernetes service사용해보기

k8s 사용해보기



EKS에 들어가 클러스터를 구성해주겠다.
자율모드를 사용하면 즉시 클러스터를 제작할수 있지만 실습을 위해 사용자 지정모드로...

k8s 사용해보기



공식 표준 권한인 해당 정책을 마스터노드(컨트롤 플레인)에 붙이기.

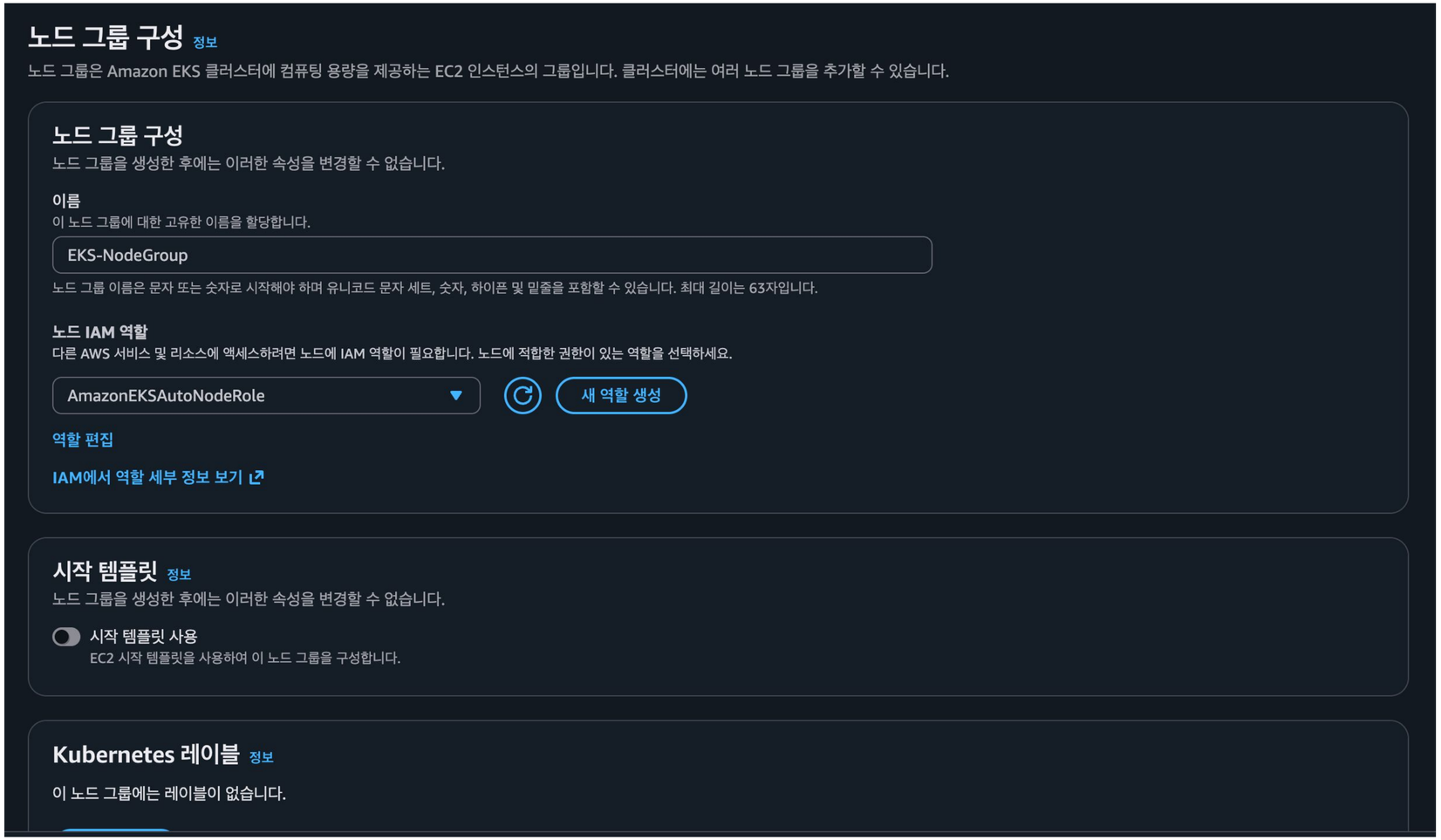
k8s 사용해보기

넘어가면 쿠버네티스 노드를 어디에 놓을거냐고 알려줘야한다. az(가용영역)2개의 프라이빗 서브넷을 골라주는게 정석

k8s 사용해보기

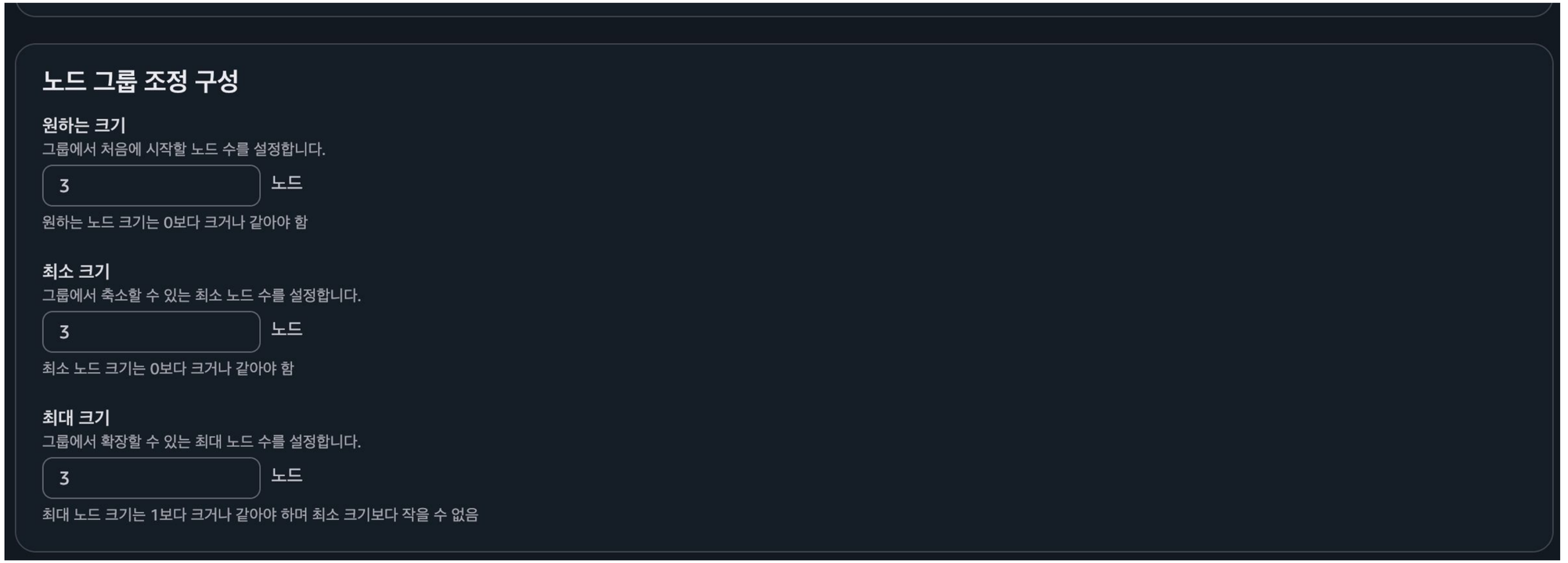
이후 별로 중요한게 없으니 계속 다음으로 넘어가주고 클러스터를 생성하자.

k8s 사용해보기



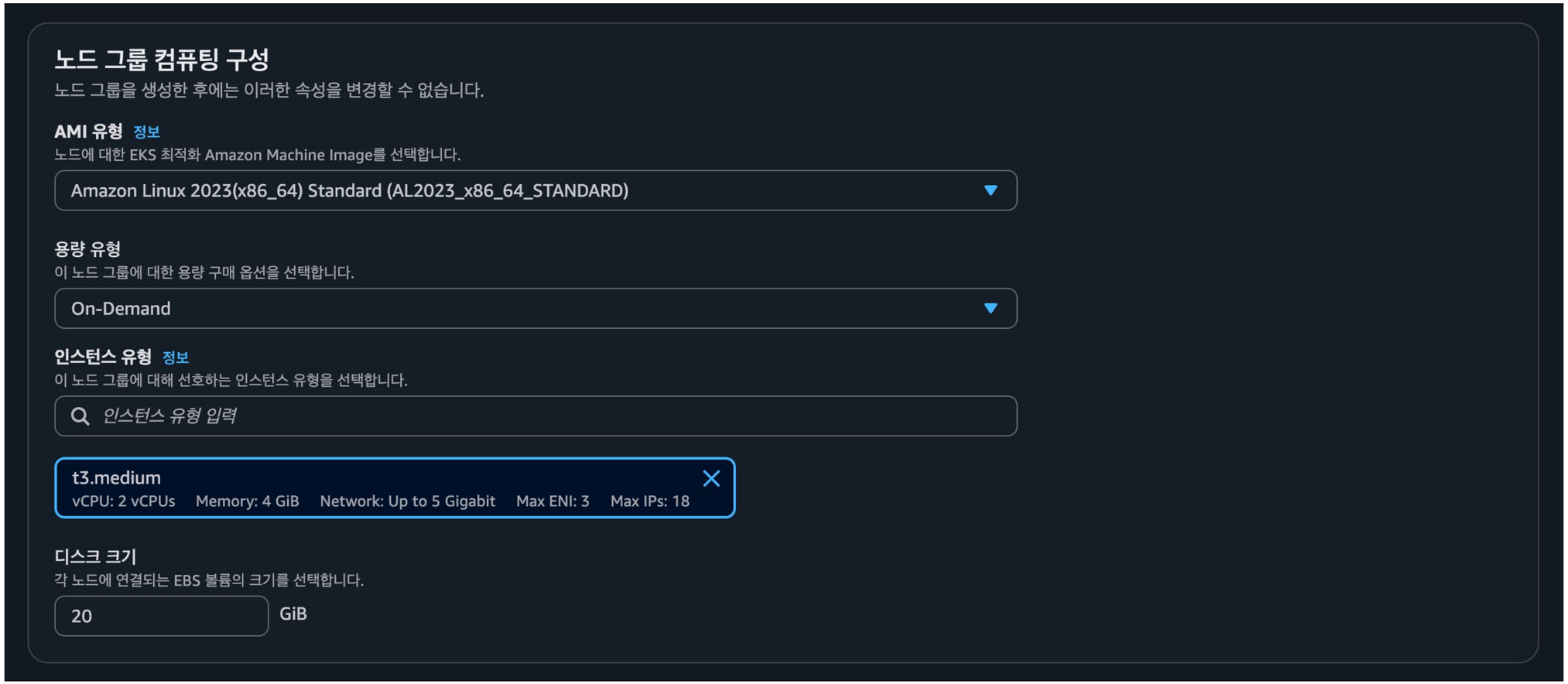
자신의 EKS 클러스터로 들어가 노드그룹(EKS 워커 노드)을 만들어주겠다

k8s 사용해보기



자신의 EKS 클러스터로 들어가 노드그룹(EKS 워커 노드)을 만들어주겠다

k8s 사용해보기



자신의 EKS 클러스터로 들어가 노드그룹(EKS 워커 노드)을 만들어주겠다

k8s 사용해보기

```
admin@m161awmui-MacBookPro ~ % brew install kubectl
==> Auto-updating Homebrew...
Adjust how often this is run with `$HOMEBREW_AUTO_UPDATE_SECS` or disable with
`$HOMEBREW_NO_AUTO_UPDATE=1`. Hide these hints with `$HOMEBREW_NO_ENV_HINTS=1` (see `man brew`).
==> Auto-updated Homebrew!
Updated 2 taps (homebrew/core and homebrew/cask).
==> New Formulae
==> Downloading https://formulae.brew.sh/api/formula.jws.json
bbrew: TUI for managing Homebrew, Flatpak, and Mac App Store packages
bomctl: Format-agnostic SBOM tooling for the stages between SBOM generation and analysis
kotofetch: Small, configurable CLI that displays Japanese quotes in the terminal
sdl3_sound: Abstract soundfile decoder
zero: Terminal coding agent you own
==> New Casks
==> Downloading https://formulae.brew.sh/api/cask.jws.json
font-caacupe-one
font-scoutie-sans
lumide: Agent-native code editor
muesli: Local-first dictation and meeting transcription
sonarqube-cli: Code quality and security for terminal workflows, scripts, and AI agents
zush: AI-powered file renamer and organiser
```

물론 aws에 내장되어있는 CloudShell에서 큐브컨트롤을 쉽게 사용 할 수 있지만 다음 파트를 위해 로컬에서 직접 큐브컨트롤을 깔겠음

k8s 사용해보기

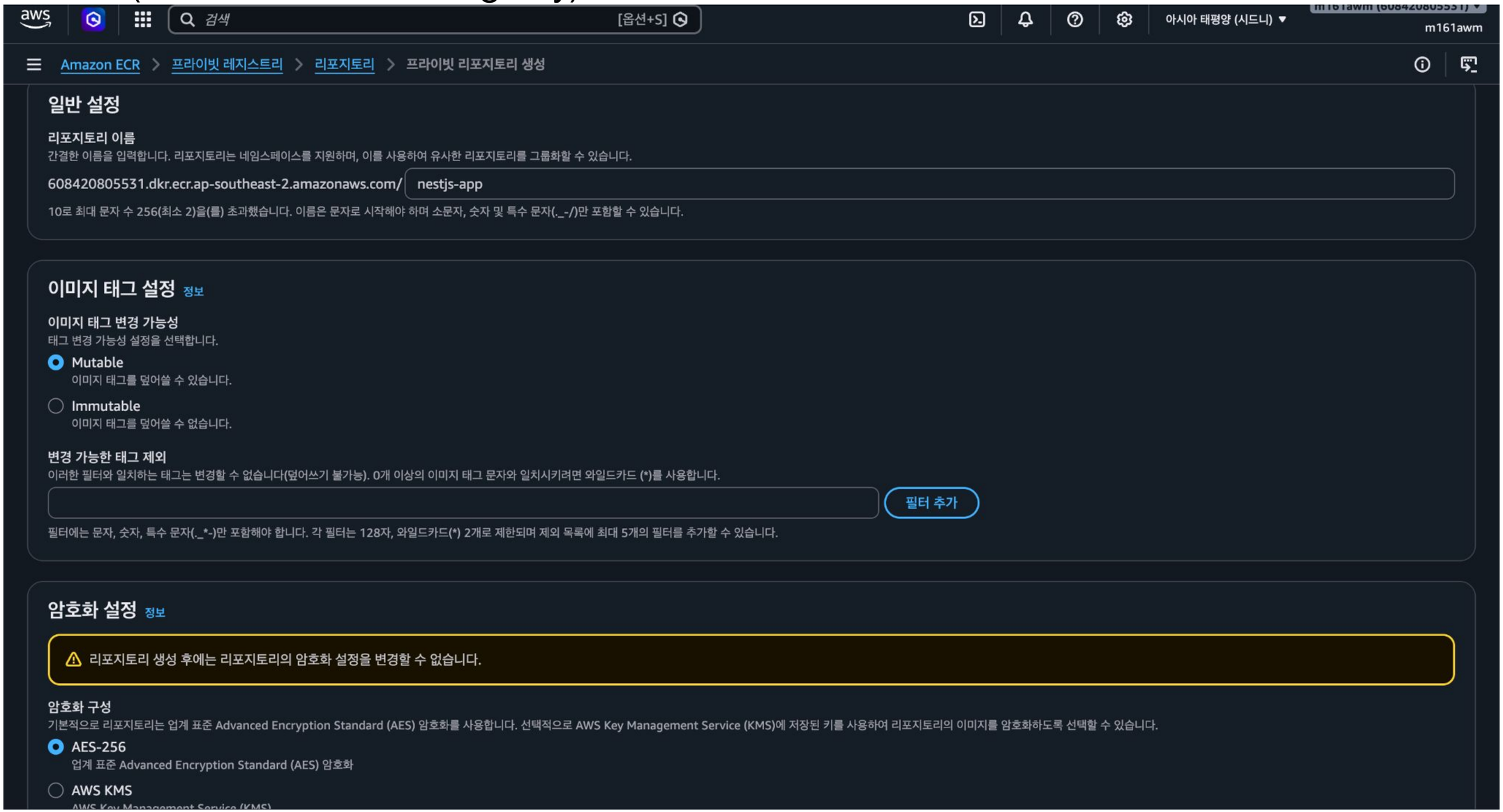
```
admin@m161awmui-MacBookPro ~ % kubectl get nodes
NAME                                            STATUS   ROLES    AGE     VERSION
ip-10-0-129-245.ap-southeast-2.compute.internal   Ready    <none>   8m44s   v1.36.2-eks-7d6f6ec
ip-10-0-150-84.ap-southeast-2.compute.internal    Ready    <none>   8m40s   v1.36.2-eks-7d6f6ec
admin@m161awmui-MacBookPro ~ %
```

Kubectl get nodes로 워커 노드가 잘 설정되었는지 확인해주겠다

ECR(Elastic Container Registry)

K8s의 파드를 제작하려면 도커의 이미지 빌드가 필요함
그래서 AWS에서 이미지를 빌드하고 사용 할 수 있는 ECR을 사용해보겠다

ECR(Elastic Container Registry)



ECR(Elastic Container Registry)

```
admin@m161awmui-MacBookPro ~ % aws ecr describe-repositories \
  --repository-names nestjs-app \
  --region ap-southeast-2
{
    "repositories": [
        {
            "repositoryArn": "arn:aws:ecr:ap-southeast-2:608420805531:repository/nestjs-app",
            "registryId": "608420805531",
            "repositoryName": "nestjs-app",
            "repositoryUri": "608420805531.dkr.ecr.ap-southeast-2.amazonaws.com/nestjs-app",
            "createdAt": "2026-07-06T13:50:24.767000+09:00",
            "imageTagMutability": "MUTABLE",
            "imageScanningConfiguration": {
                "scanOnPush": false
            },
            "encryptionConfiguration": {
                "encryptionType": "AES256"
            }
        }
    ]
}
admin@m161awmui-MacBookPro ~ %
```

이미지 저장 주소를 알아내기, 파드를 배포할때 해당 주소를 사용함

ECR(Elastic Container Registry)

```
admin@m161awmui-MacBookPro ~ % aws ecr describe-repositories \
  --repository-names nestjs-app \
  --region ap-southeast-2
{
    "repositories": [
        {
            "repositoryArn": "arn:aws:ecr:ap-southeast-2:608420805531:repository/nestjs-app",
            "registryId": "608420805531",
            "repositoryName": "nestjs-app",
            "repositoryUri": "608420805531.dkr.ecr.ap-southeast-2.amazonaws.com/nestjs-app",
            "createdAt": "2026-07-06T13:50:24.767000+09:00",
            "imageTagMutability": "MUTABLE",
            "imageScanningConfiguration": {
                "scanOnPush": false
            },
            "encryptionConfiguration": {
                "encryptionType": "AES256"
            }
        }
    ]
}
admin@m161awmui-MacBookPro ~ %
```

이미지 저장 주소를 알아내기, 파드를 배포할때 해당 주소를 사용함

ECR(Elastic Container Registry)

```
admin@m161awmui-MacBookPro ~ % aws ecr describe-repositories \
  --repository-names nestjs-app \
  --region ap-southeast-2
{
    "repositories": [
        {
            "repositoryArn": "arn:aws:ecr:ap-southeast-2:608420805531:repository/nestjs-app",
            "registryId": "608420805531",
            "repositoryName": "nestjs-app",
            "repositoryUri": "608420805531.dkr.ecr.ap-southeast-2.amazonaws.com/nestjs-app",
            "createdAt": "2026-07-06T13:50:24.767000+09:00",
            "imageTagMutability": "MUTABLE",
            "imageScanningConfiguration": {
                "scanOnPush": false
            },
            "encryptionConfiguration": {
                "encryptionType": "AES256"
            }
        }
    ]
}
admin@m161awmui-MacBookPro ~ %
```

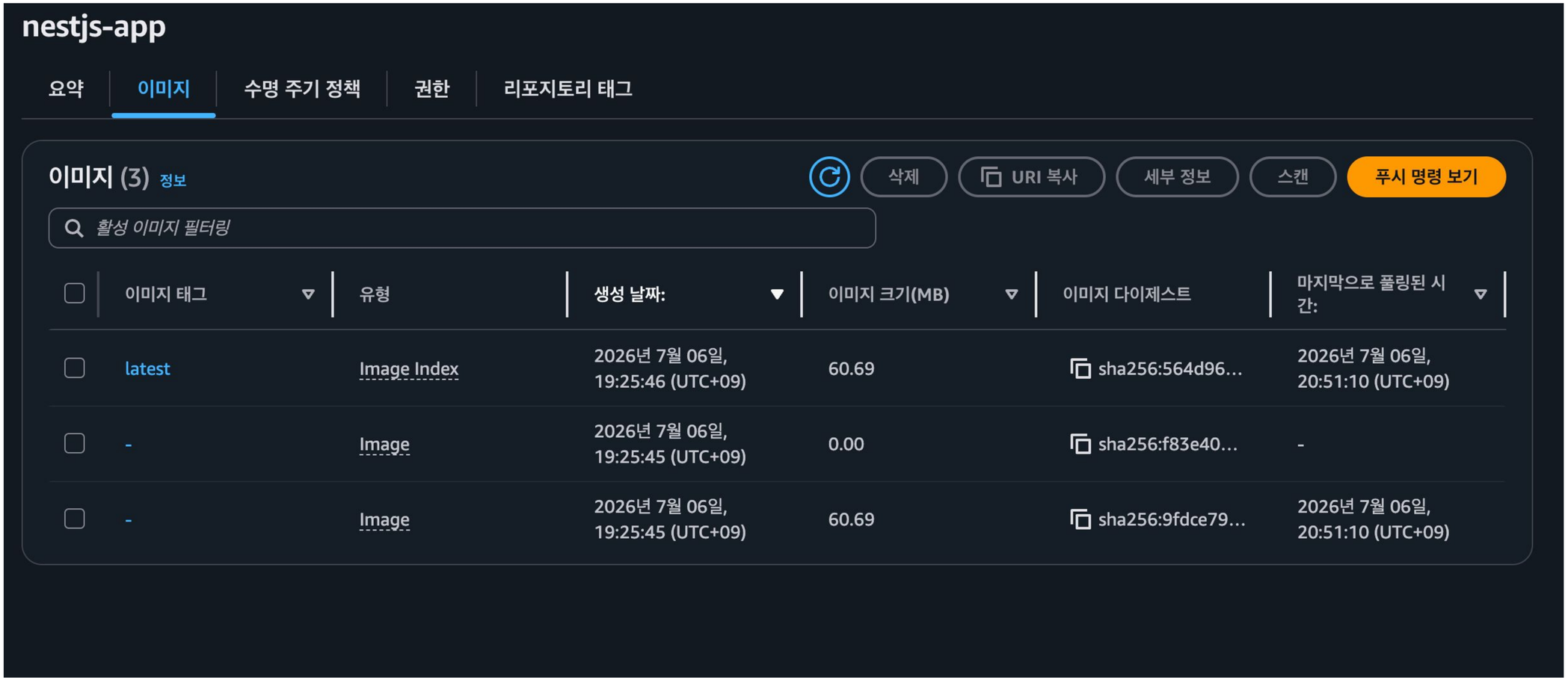
이미지 저장 주소를 알아내기, 파드를 배포할때 해당 주소를 사용함

ECR(Elastic Container Registry)

```
admin@m161awmui-MacBookPro howTo-argoCD % docker buildx build \
  --platform linux/amd64 \
  -t 608420805531.dkr.ecr.ap-southeast-2.amazonaws.com/nestjs-app:latest \
  --load .
[+] Building 19.3s (12/15)                                                  docker:desktop-linux
 => [internal] load build definition from Dockerfile                                        0.0s
 => => transferring dockerfile: 662B                                                        0.0s
 => [internal] load metadata for docker.io/library/node:22-alpine                           3.6s
 => [auth] library/node:pull token for registry-1.docker.io                                 0.0s
 => [internal] load .dockerignore                                                           0.0s
 => => transferring context: 144B                                                           0.0s
 => [deps 1/4] FROM docker.io/library/node:22-alpine@sha256:16e22a550f3863206a3f701448c45f7912c6896a6  2.1s
 => => resolve docker.io/library/node:22-alpine@sha256:16e22a550f3863206a3f701448c45f7912c6896a62de43  0.0s
 => => sha256:d10ff586f6bdbdd6000a70a914e56bf05bedc78de3311d08a953643982d84e23 446B / 446B  0.2s
 => => sha256:1d4d101ceaad39fb56dcac0bb6cc291e6cdec3b711786f0c82b050b5b5b85690 1.26MB / 1.26MB  0.5s
 => => sha256:3f4c164634d251a8b256151a9e1c1d3e5c5b21f1974f00a213875e66e3fa0801 52.31MB / 52.31MB  1.5s
 => => sha256:55afa1ecc21d2bb5e5045f32dafee56272ffd89860bac26f6c32123439af26a4 3.85MB / 3.85MB  1.5s
 => => extracting sha256:55afa1ecc21d2bb5e5045f32dafee56272ffd89860bac26f6c32123439af26a4   0.1s
 => => extracting sha256:3f4c164634d251a8b256151a9e1c1d3e5c5b21f1974f00a213875e66e3fa0801   0.4s
 => => extracting sha256:1d4d101ceaad39fb56dcac0bb6cc291e6cdec3b711786f0c82b050b5b5b85690   0.0s
 => => extracting sha256:d10ff586f6bdbdd6000a70a914e56bf05bedc78de3311d08a953643982d84e23   0.0s
 => [internal] load build context                                                           0.0s
 => => transferring context: 382.41kB                                                       0.0s
 => [deps 2/4] WORKDIR /app                                                                 0.1s
 => [deps 3/4] COPY package*.json ./                                                        0.0s
 => [runner 4/6] RUN npm ci --omit=dev && npm cache clean --force                           4.5s
 => [deps 4/4] RUN npm ci                                                                   8.9s
 => [builder 3/5] COPY --from=deps /app/node_modules ./node_modules                         0.7s
 => [builder 4/5] COPY . .                                                                  0.1s
 => [builder 5/5] RUN npm run build                                                         3.4s
 => => # > nest-cicd-demo@0.0.1 build
 => => # > nest build
```

도커 데스크탑이 열려있는 상태에서 로컬에서 이미지를 빌드한다

ECR(Elastic Container Registry)



도커 푸시로 ECR 레포지토리에 이미지를 박아주면 된다.

파드를 배포하고 서비스를 사용해보기

```
admin@m161awmui-MacBookPro ~ % kubectl create deployment nestjs-dep --replicas=3 --image=608420805531.dkr.ecr.ap-southeast-2.amazonaws.com/nestjs-app:latest --dry-run=client -o yaml > deployment.yaml
```

```
admin@m161awmui-MacBookPro ~ % kubectl apply -f deployment.yaml
deployment.apps/nestjs-dep created
admin@m161awmui-MacBookPro ~ %
```

파드를 배포하고 서비스를 사용해보기

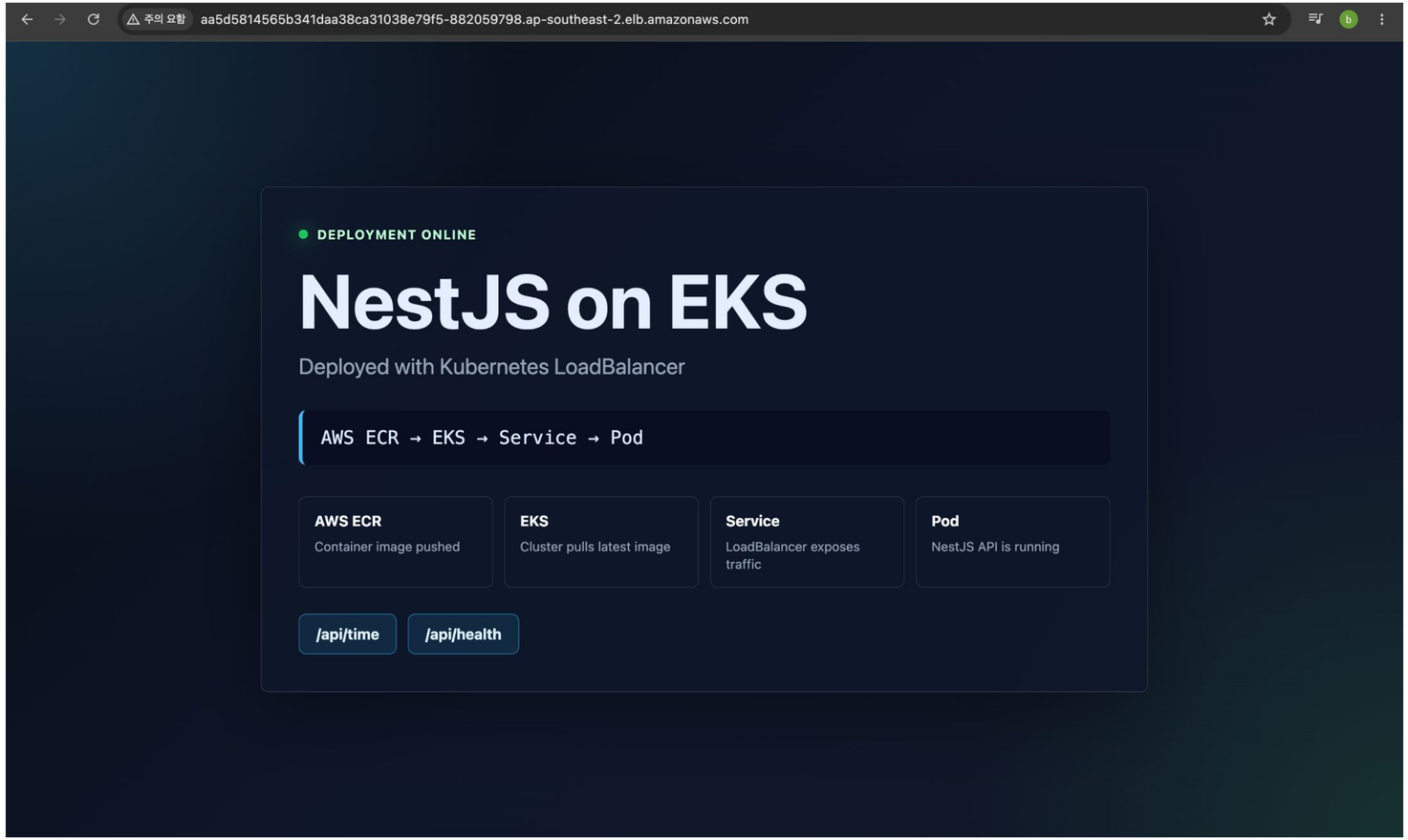
```
[admin@m161awmui-MacBookPro ~ % kubectl scale deployment nestjs-dep --replicas=2                                    ]
deployment.apps/nestjs-dep scaled
[admin@m161awmui-MacBookPro ~ % kubectl get pods -o wide                                                            ]
NAME                          READY   STATUS    RESTARTS   AGE   IP            NODE
         NOMINATED NODE   READINESS GATES
nestjs-dep-7784dd568f-fpncw   1/1     Running   0          11h   10.0.155.6    ip-10-0-150-84.ap-southeast-2.compute.int
ernal    <none>           <none>
nestjs-dep-7784dd568f-pr79g   1/1     Running   0          11h   10.0.142.38   ip-10-0-129-245.ap-southeast-2.compute.in
ternal   <none>           <none>
admin@m161awmui-MacBookPro ~ %
```

파드를 배포하고 서비스를 사용해보기

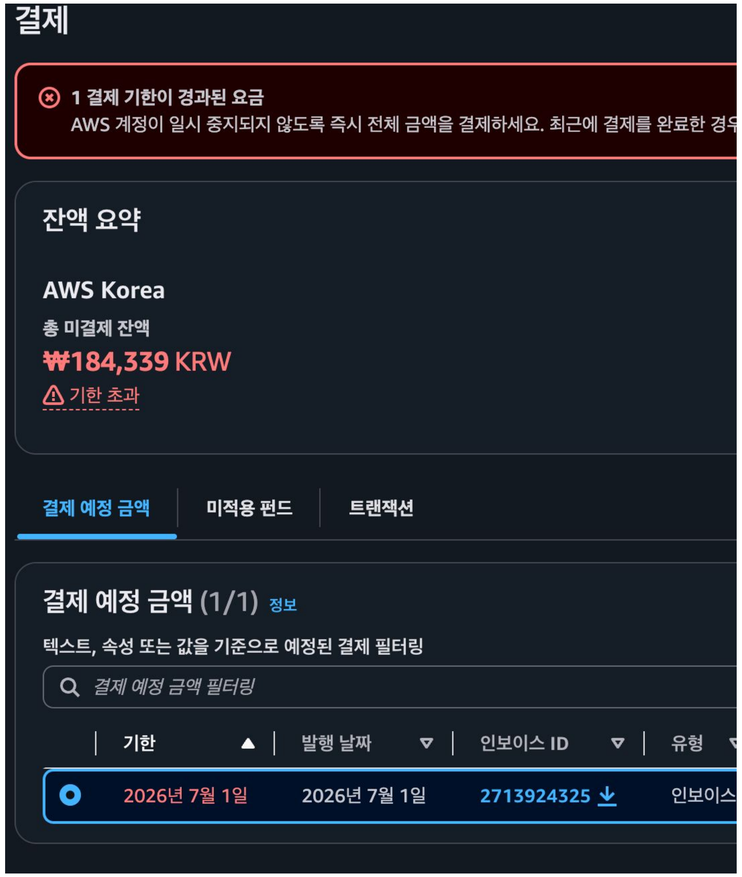
```
admin@m161awmui-MacBookPro ~ % kubectl get pods -o wide
NAME                          READY   STATUS    RESTARTS   AGE   IP            NODE
        NOMINATED NODE   READINESS GATES
nestjs-dep-7784dd568f-fpncw   1/1     Running   0          11h   10.0.155.6    ip-10-0-150-84.ap-southeast-2.compute.int
ernal    <none>           <none>
nestjs-dep-7784dd568f-pr79g   1/1     Running   0          11h   10.0.142.38   ip-10-0-129-245.ap-southeast-2.compute.in
ternal   <none>           <none>
admin@m161awmui-MacBookPro ~ % kubectl expose deployment nestjs-dep \
  --name=nestjs-service \
  --type=LoadBalancer \
  --port=80 \
  --target-port=3000
service/nestjs-service exposed
admin@m161awmui-MacBookPro ~ % kubectl get svc
NAME             TYPE           CLUSTER-IP       EXTERNAL-IP
       PORT(S)        AGE
kubernetes       ClusterIP      172.20.0.1       <none>
       443/TCP        27h
nestjs-service   LoadBalancer   172.20.252.231   aa5d5814565b341daa38ca31038e79f5-882059798.ap-southeast-2.elb.amazonaws
.com   80:30720/TCP   7s
admin@m161awmui-MacBookPro ~ %
```

서비스를 만들어 사용자가 고정된 IP로 로드밸런서가 두개의 파드중 하나로 보내줌

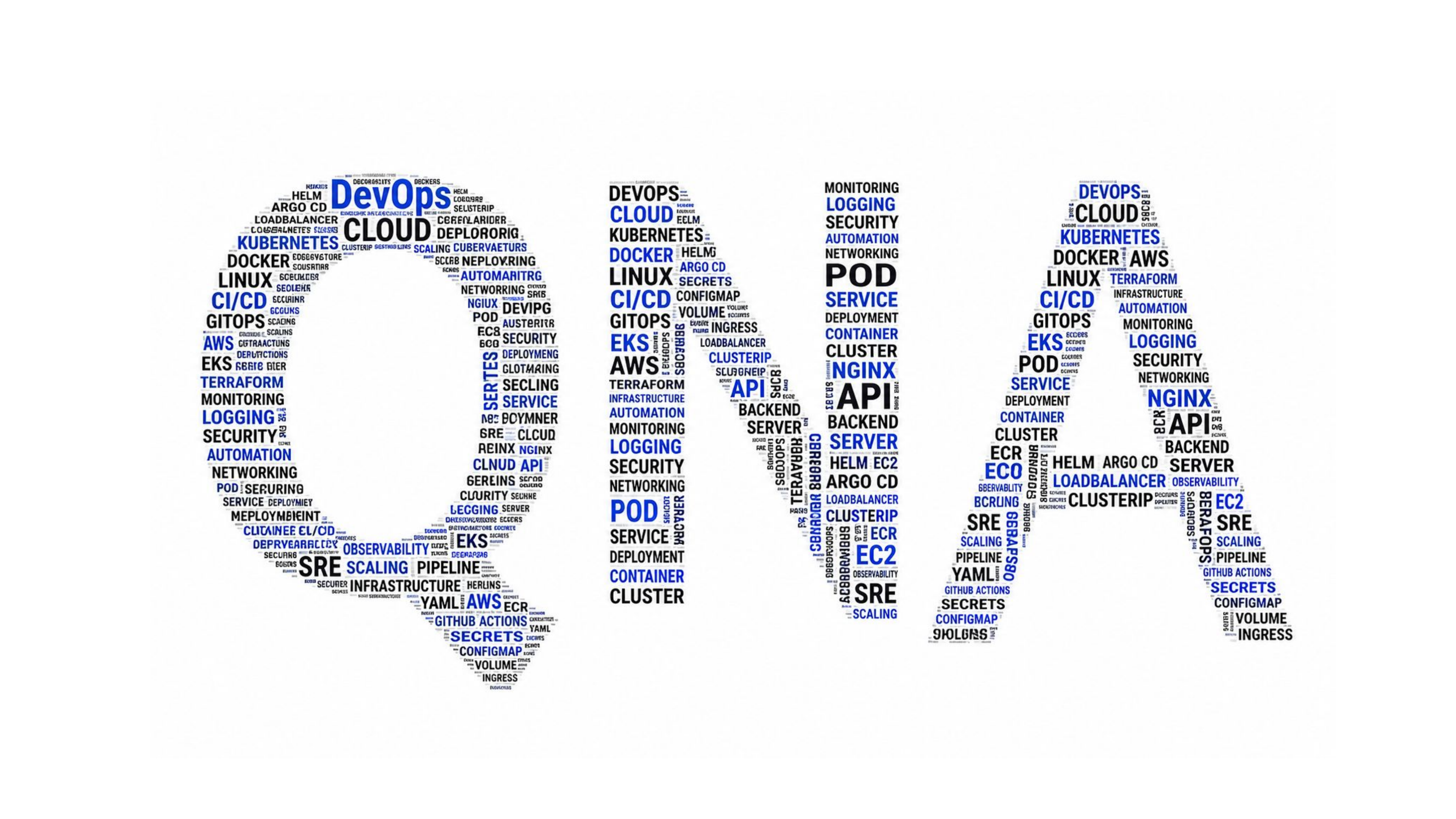
파드를 배포하고 서비스를 사용해보기



로컬에서 로드밸런서의 dns주소로 들어갔을때 배포가 성공적으로 완료된걸 알 수 있다.



AWS는 잘못만지면 나처럼 체불당할수도 있음.
실습이 끝나면 무조건 리소스를 삭제해줘야한다...


DevOps
CLOUD
KUBERNETES
DOCKER
LINUX
CI/CD
GITOPS
AWS
EKS
TERRAFORM
MONITORING
LOGGING
SECURITY
AUTOMATION
NETWORKING
POD
SERVICE
DEPLOYMENT
CONTAINER
CLUSTER
ECR
HELM
ARGO CD
LOADBALANCER
CLUSTERIP
OBSERVABILITY
SRE
SCALING
PIPELINE
YAML
GITHUB ACTIONS
SECRETS
CONFIGMAP
VOLUME
INGRESS
NGINX
API
BACKEND
SERVER
EC2

들어주셔서 감사합니다

10201 구본무