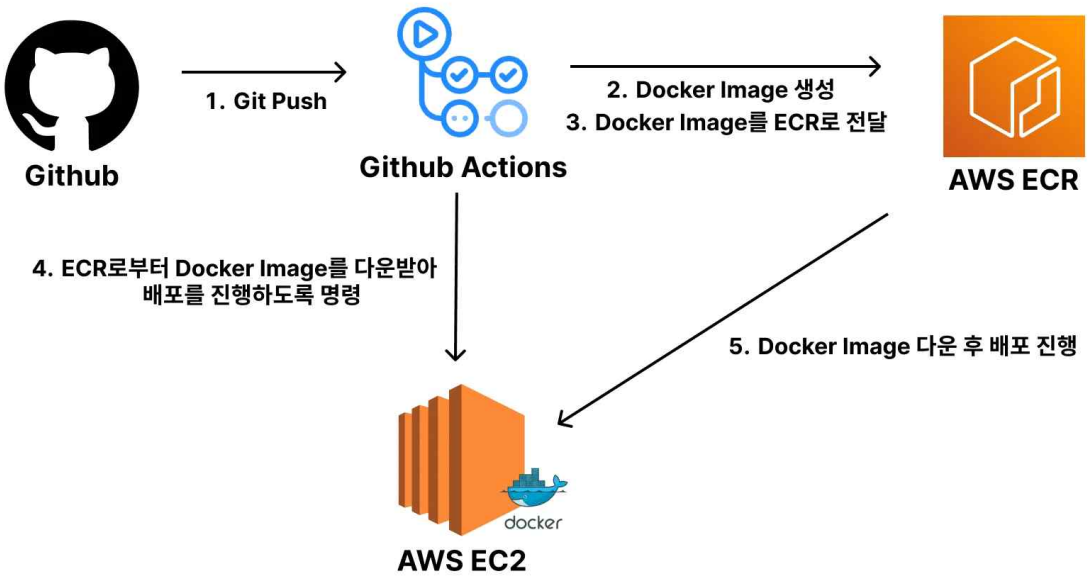
제 목	03장 Docker와 Nest.js CI/CD 적용하기	
상세내용	Docker + 백엔드(Nest.js) 프로젝트에 CI/CD 적용하기	

방법 4 - 컨테이너 기반의 프로젝트에서 많이 쓰는 CI/CD 구축 방법 (Docker)

✓ 전체적인 흐름



✓ 장점

- Docker 기반으로 서비스를 운영할 때, 가장 간단하게 구성할 수 있는 인프라 구조이다.

✓ 단점

- 무중단 배포를 구현하거나 여러 EC2 인스턴스에 배포를 해야 하는 상황이라면, 직접 Github Actions에 스크립트를 작성해서 구현해야 한다. 직접 구현을 해보면 알겠지만 생각보다 꽤 복잡하다.

✓ 이 방법은 언제 주로 쓰는 지

- 컨테이너 기반으로 인프라를 구성했을 때 이 방법을 많이 활용한다.
- 서버를 여러 대 운영하고 있지 않을 정도의 소규모 프로젝트 일 때 주로 활용한다.

[실습] EC2에 Docker 설치, ECR 셋팅하기

✓ 1. Ubuntu에서 Docker, Docker Compose 설치하기

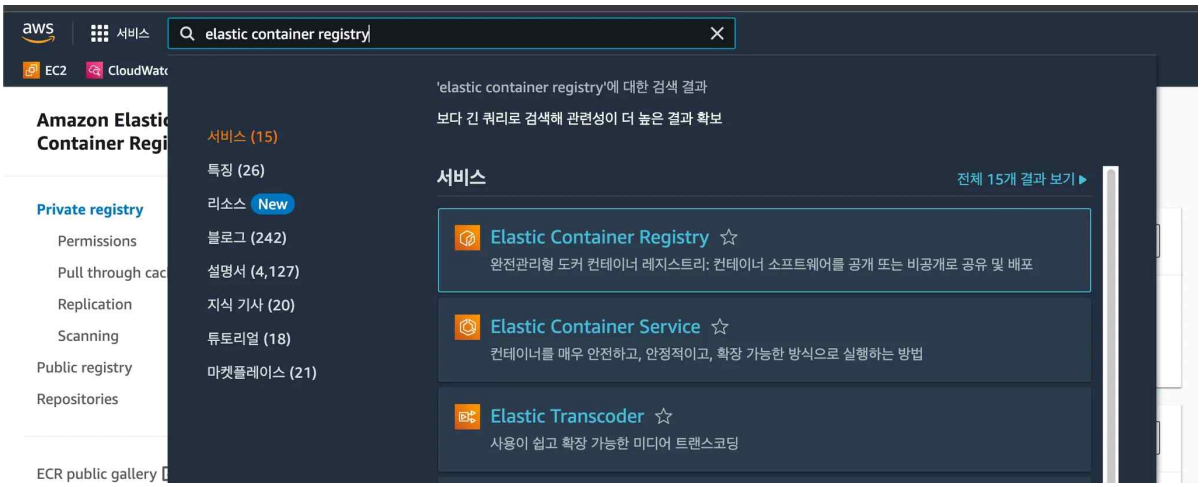
```
$ sudo apt-get update && \
        sudo apt-get install -y apt-transport-https ca-certificates curl softwar
e-properties-common && \
        curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-
key add - && \
        sudo apt-key fingerprint 0EBFCD88 && \
        sudo add-apt-repository "deb [arch=amd64] https://download.docker.co
m/linux/ubuntu $(lsb_release -cs) stable" && \
        sudo apt-get update && \
        sudo apt-get install -y docker-ce && \
        sudo usermod -aG docker ubuntu && \
        sudo curl -L "https://github.com/docker/compose/releases/download/
1.23.2/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-
compose && \
        sudo chmod +x /usr/local/bin/docker-compose && \
        sudo ln -s /usr/local/bin/docker-compose /usr/bin/docker-compose

# 잘 설치됐는 지 확인
$ docker -v # Docker 버전 확인
$ docker compose version # Docker Compose 버전 확인
```

✓ 2. Github Actions의 IAM에 권한 추가

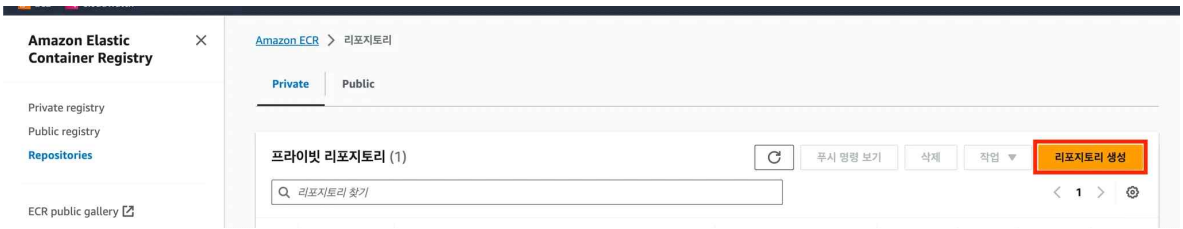
AmazonEC2ContainerRegistryFullAccess 권한 추가하기

✓ 3. ECR(Elastic Container Registry) 만들기

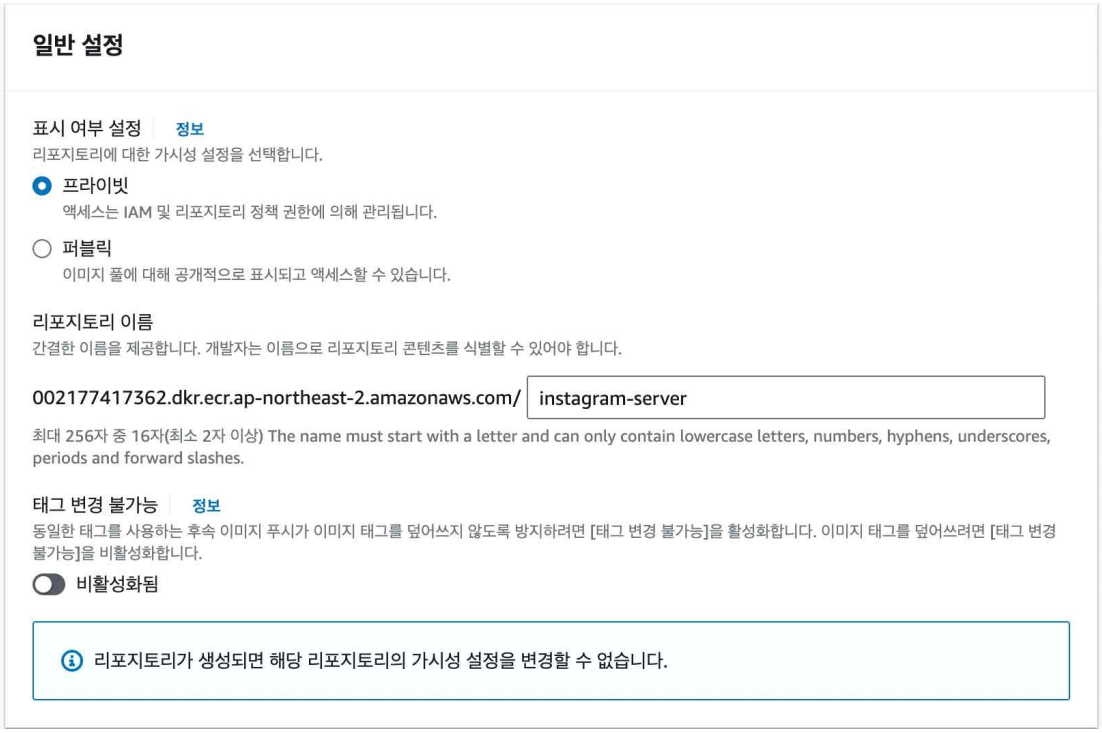
1. 리스트



2. 리스트



3. 리스트



[실습] 컨테이너 기반의 프로젝트에서 많이 쓰는 CI/CD 구축 방법

✓ 이전 실습했던 내용 정리

- 서버 종료
- 프로젝트 폴더 삭제

✓ 1. Docker 기반으로 프로젝트 수정하기

1. Dockerfile 작성하기

Dockerfile

```
FROM node:alpine

WORKDIR /usr/src/app

COPY . .

RUN npm install

RUN npm run build

EXPOSE 3000

CMD [ "node", "dist/main.js" ]
```

2. .dockerignore 파일 생성

컨테이너에 불필요한 파일들이 복사(COPY) 되는 걸 막아준다

```
node_modules
```

✓ 2. EC2가 Private ECR에 접근할 수 있게 셋팅하기

1. Amazon ECR Docker Credential Helper 설치하기

▶ 참조: https://github.com/awslabs/amazon-ecr-credential-helper?tab=readme-ov-file

```
# Ubuntu일 경우
$ sudo apt update
$ sudo apt install amazon-ecr-credential-helper
```

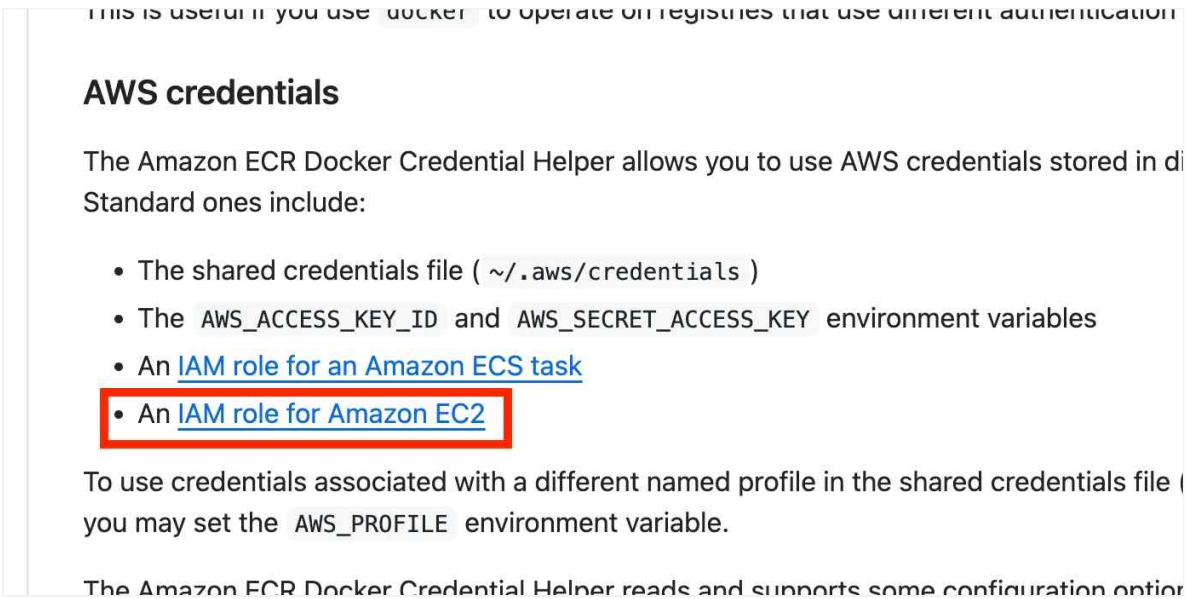
2. Configuration 설정하기

~ 경로에서 .docker라는 폴더 만들고, config.json 파일 만들어서 아래와 같이 작성해라.

~/.docker/config.json

```
{
        "credsStore": "ecr-login"
}
```

3. IAM Role을 활용해 EC2가 ECR에 접근할 수 있게 권한 부여하기



This is useful if you use `docker` to operate on registries that use different authentication

AWS credentials

The Amazon ECR Docker Credential Helper allows you to use AWS credentials stored in di
Standard ones include:

- The shared credentials file (`~/.aws/credentials`)
- The `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` environment variables
- An IAM role for an Amazon ECS task
- An IAM role for Amazon EC2

To use credentials associated with a different named profile in the shared credentials file (
you may set the `AWS_PROFILE` environment variable.

The Amazon ECR Docker Credential Helper reads and supports some configuration optior

✓ 3. Docker 기반 CI/CD 구축하

1. Github Actions 파일 작성하기

.github/workflows/deploy.yml

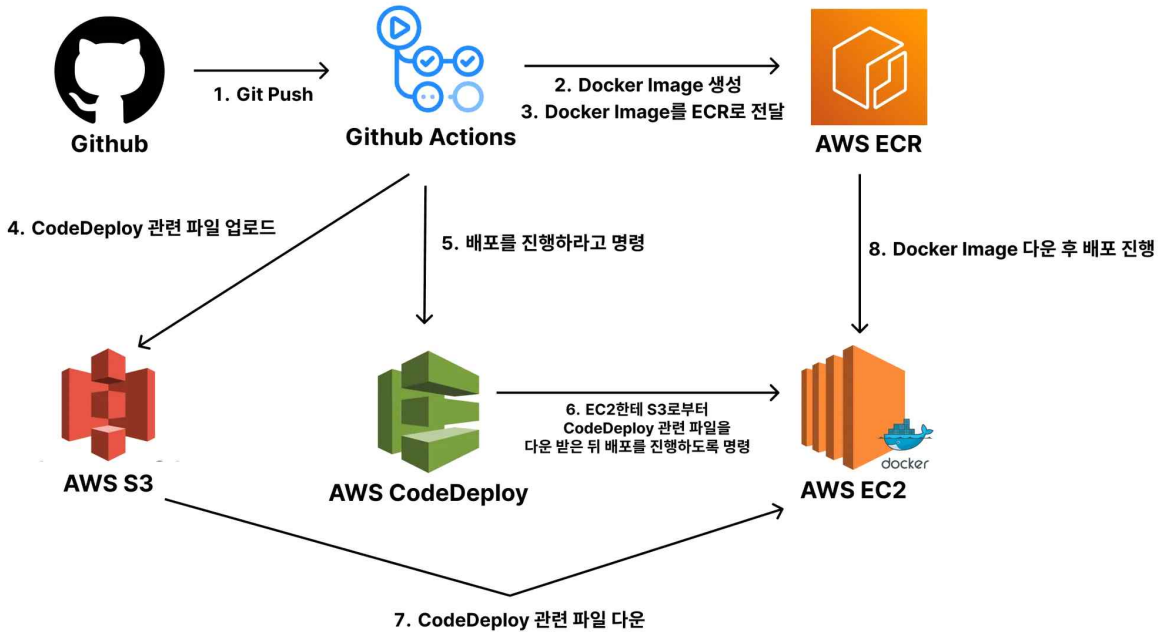
```
name: Deploy To EC2

on:
  push:
    branches:
      - main

jobs:
  deploy:
    runs-on: ubuntu-latest
    steps:
      - name: Github Repository 파일 불러오기
        uses: actions/checkout@v4

      - name: Node 설치
        uses: actions/setup-node@v4
        with:
          node-version: "20"

      - name: 의존성(라이브러리) 설치
        run: npm ci

      - name: .env 파일 만들기
        run: |
          touch .env
          echo '${{ secrets.ENV }}' >> .env

      - name: 테스트 코드 실행
        run: npm run test

      - name: AWS Resource에 접근할 수 있게 AWS credentials 설정
        uses: aws-actions/configure-aws-credentials@v4
        with:
          aws-region: ap-northeast-2
```

```
          aws-access-key-id: ${{ secrets.AWS_ACCESS_KEY_ID }}
          aws-secret-access-key: ${{ secrets.AWS_SECRET_ACCESS_KEY }}

      - name: ECR에 로그인하기
        id: login-ecr
        uses: aws-actions/amazon-ecr-login@v2

      - name: Docker 이미지 생성
        run: docker build -t instagram-server .

      - name: Docker 이미지에 Tag 붙이기
        run: docker tag instagram-server ${{ steps.login-ecr.outputs.registry }}/instagram-server:latest

      - name: ECR에 Docker 이미지 Push하기
        run: docker push ${{ steps.login-ecr.outputs.registry }}/instagram-server:latest

      - name: SSH로 EC2에 접속하기
        uses: appleboy/ssh-action@v1.0.3
        with:
          host: ${{ secrets.EC2_HOST }}
          username: ${{ secrets.EC2_USERNAME }}
          key: ${{ secrets.EC2_PRIVATE_KEY }}
          script_stop: true
          script: |
            docker stop instagram-server || true
            docker rm instagram-server || true
            docker pull ${{ steps.login-ecr.outputs.registry }}/instagram-server:latest
            docker run -d --name instagram-server -p 3000:3000 ${{ steps.login-ecr.outputs.registry }}/instagram-server:latest
```

2. CI/CD 과정이 잘 작동하는 지 확인하기

방법 5 - 컨테이너 기반 + 확장성을 고려한 프로젝트에서 많이 쓰는 CI/CD 구축 방법 (Docker, CodeDeploy)

✓ 전체적인 흐름



✓ 장점

- 컨테이너 기반의 서버가 여러 대이더라도 쉽게 자동 배포를 구축할 수 있다.
- 쉽게 무중단 배포를 적용시킬 수 있다.

✓ 단점

- CodeDeploy를 사용함으로써 인프라 구조가 복잡해졌다. 구조가 복잡해짐에 따라 관리 비용, 유지보수 비용, 난이도, 트러블 슈팅 어려움, 복잡도가 증가했다.

✓ 이 방법은 언제 주로 쓰는 지

- 컨테이너 기반의 서버를 여러 대 이상 구동해야 하거나 무중단 배포가 중요한 서비스일 때 주로 활용한다.

[실습] 컨테이너 기반 + 확장성을 고려한 프로젝트에서 많이 쓰는 CI/CD 구축 방법

✓ Github Actions 코드 수정하기

.github/workflows/deploy.yml

```
name: Deploy To EC2

on:
  push:
    branches:
      - main

jobs:
  deploy:
    runs-on: ubuntu-latest
    steps:
      - name: Github Repository 파일 불러오기
        uses: actions/checkout@v4

      - name: Node 설치
        uses: actions/setup-node@v4
        with:
          node-version: "20"

      - name: 의존성(라이브러리) 설치
        run: npm ci

      - name: .env 파일 만들기
        run: |
          touch .env
          echo '${{ secrets.ENV }}' >> .env

      - name: 테스트 코드 실행
        run: npm run test

      - name: AWS Resource에 접근할 수 있게 AWS credentials 설정
        uses: aws-actions/configure-aws-credentials@v4
        with:
```

```
          aws-region: ap-northeast-2
          aws-access-key-id: ${{ secrets.AWS_ACCESS_KEY_ID }}
          aws-secret-access-key: ${{ secrets.AWS_SECRET_ACCESS_KEY }}

      - name: ECR에 로그인하기
        id: login-ecr
        uses: aws-actions/amazon-ecr-login@v2

      - name: Docker 이미지 생성
        run: docker build -t instagram-server .

      - name: Docker 이미지에 Tag 붙이기
        run: docker tag instagram-server ${{ steps.login-ecr.outputs.registry }}/instagram-server:latest

      - name: ECR에 Docker 이미지 Push하기
        run: docker push ${{ steps.login-ecr.outputs.registry }}/instagram-server:latest

      - name: 압축하기
        run: tar -czvf $GITHUB_SHA.tar.gz appspec.yml scripts

      - name: S3에 프로젝트 폴더 업로드하기
        run: aws s3 cp --region ap-northeast-2 ./$GITHUB_SHA.tar.gz s3://instagram-server/$GITHUB_SHA.tar.gz

      - name: Code Deploy를 활용해 EC2에 프로젝트 코드 배포
        run: aws deploy create-deployment
          --application-name instagram-server
          --deployment-config-name CodeDeployDefault.AllAtOnce
          --deployment-group-name Production
          --s3-location  bucket=instagram-server,bundleType=tgz,key=$GITHUB_SHA.tar.gz
```

✅ appspec.yml, 스크립트 파일 수정하기

appspec.yml → 이전 작성했던 코드 그대로 유지

```
version: 0.0
os: linux

files:
  # CodeDeploy가 S3로부터 가져온 파일 중 destination으로 이동시킬 대상을 지정한다.
  # / 이라고 지정하면 S3로부터 가져온 전체 파일을 뜻한다.
  - source: /
    # CodeDeploy가 S3로부터 가져온 파일을 EC2의 어떤 경로에 저장할 지 지정한다.
    destination: /home/ubuntu/instagram-server

permissions:
  - object: /
    owner: ubuntu
    group: ubuntu

hooks:
  ApplicationStart:
    - location: scripts/start-server.sh
      timeout: 60
      runas: ubuntu
```

scripts/start-server.sh

```
#!/bin/bash

echo "--------------- 서버 배포 시작 -----------------"
docker stop instagram-server || true
docker rm instagram-server || true
docker pull {ECR Repository 주소}/instagram-server:latest
docker run -d --name instagram-server -p 3000:3000 {ECR Repository 주소}/instagram-server:latest
echo "--------------- 서버 배포 끝 -----------------"
```