

ECR & ECS

Container Service

❖ 가상화



AWS Container Service

- ❖ Django 애플리케이션을 Docker Image를 이용한 EC2 인스턴스에 배포
 - ✓ 프로젝트 생성
 - 가상 환경 생성
 - ◆ `python3 -m venv ./{your venv name}`
 - ◆ 가상 환경 활성화
 - Linux, Mac: `Source {your venv name}/bin/activate`
 - Windows: `{your venv name}/Scripts/activate`
 - Django 설치 및 실행
 - ◆ `(venv)$ pip install django djangorestframework`
 - ◆ `(venv)$ django-admin startproject apiserver`
 - ◆ `(venv)$ cd apiserver`
 - ◆ `(venv)$ python manage.py runserver`
 - ◆ 브라우저에 접속해서 `localhost:8000` 으로 접속
 - 라이브러리의 의존성을 파일에 내보내기
 - ◆ `pip freeze > requirements.txt`

AWS Container Service

❖ Django 애플리케이션을 Docker Image를 이용한 EC2 인스턴스에 배포

✓ 프로젝트 설정

- settings.py 파일 수정

```
ALLOWED_HOSTS = ['*']
```

```
# Application definition
```

```
INSTALLED_APPS = [  
    "django.contrib.admin",  
    "django.contrib.auth",  
    "django.contrib.contenttypes",  
    "django.contrib.sessions",  
    "django.contrib.messages",  
    "django.contrib.staticfiles",  
    "rest_framework",  
    "apiserver",]
```

AWS Container Service

❖ Django 애플리케이션을 Docker Image를 이용한 EC2 인스턴스에 배포

✓ 프로젝트 설정

- settings.py 파일 수정

Internationalization

<https://docs.djangoproject.com/en/4.1/topics/i18n/>

LANGUAGE_CODE = "en-us"

TIME_ZONE = "Asia/Seoul"

USE_I18N = True

USE_TZ = True

Static files (CSS, JavaScript, Images)

<https://docs.djangoproject.com/en/4.1/howto/static-files/>

STATIC_URL = "static/"

Default primary key field type

<https://docs.djangoproject.com/en/4.1/ref/settings/#default-auto-field>

DEFAULT_AUTO_FIELD = "django.db.models.BigAutoField"

AWS Container Service

❖ Django 애플리케이션을 Docker Image를 이용한 EC2 인스턴스에 배포

✓ 요청 처리

- views.py 파일을 생성하고 요청 처리 메서드 작성

```
from rest_framework.response import Response
from rest_framework.decorators import api_view
from rest_framework import status
```

```
@api_view(['GET'])
def index(request):
    data = {"result": "success", "data": [{"id": "itstudy", "name": "군계"},
                                         {"id": "ggangpae1", "name": "아담"}]}
    return Response(data, status=status.HTTP_200_OK)
```

AWS Container Service

❖ Django 애플리케이션을 Docker Image를 이용한 EC2 인스턴스에 배포

✓ 요청 처리

● urls.py 파일 수정

```
from django.contrib import admin
from django.urls import path
from .views import index
urlpatterns = [
    path("admin/", admin.site.urls),
    path("", index )
]
```

AWS Container Service

- ❖ Django 애플리케이션을 Docker Image를 이용한 EC2 인스턴스에 배포
 - ✓ 프로젝트 실행 및 확인
 - 프로젝트 실행: `python manage.py runserver`
 - 확인



AWS Container Service

❖ Django 애플리케이션을 Docker Image를 이용한 EC2 인스턴스에 배포

✓ Docker Image 생성 및 실행

● Dockerfile 작성

```
FROM --platform=linux/amd64 python:3.8-slim-buster as build
```

```
RUN apt-get update &
```

```
&& apt-get install -y --no-install-recommends &
```

```
postgresql-client &
```

```
&& rm -rf /var/lib/apt/lists/*
```

```
WORKDIR /usr/src/app
```

```
COPY requirements.txt ./
```

```
RUN pip install -r requirements.txt
```

```
COPY . .
```

```
EXPOSE 80
```

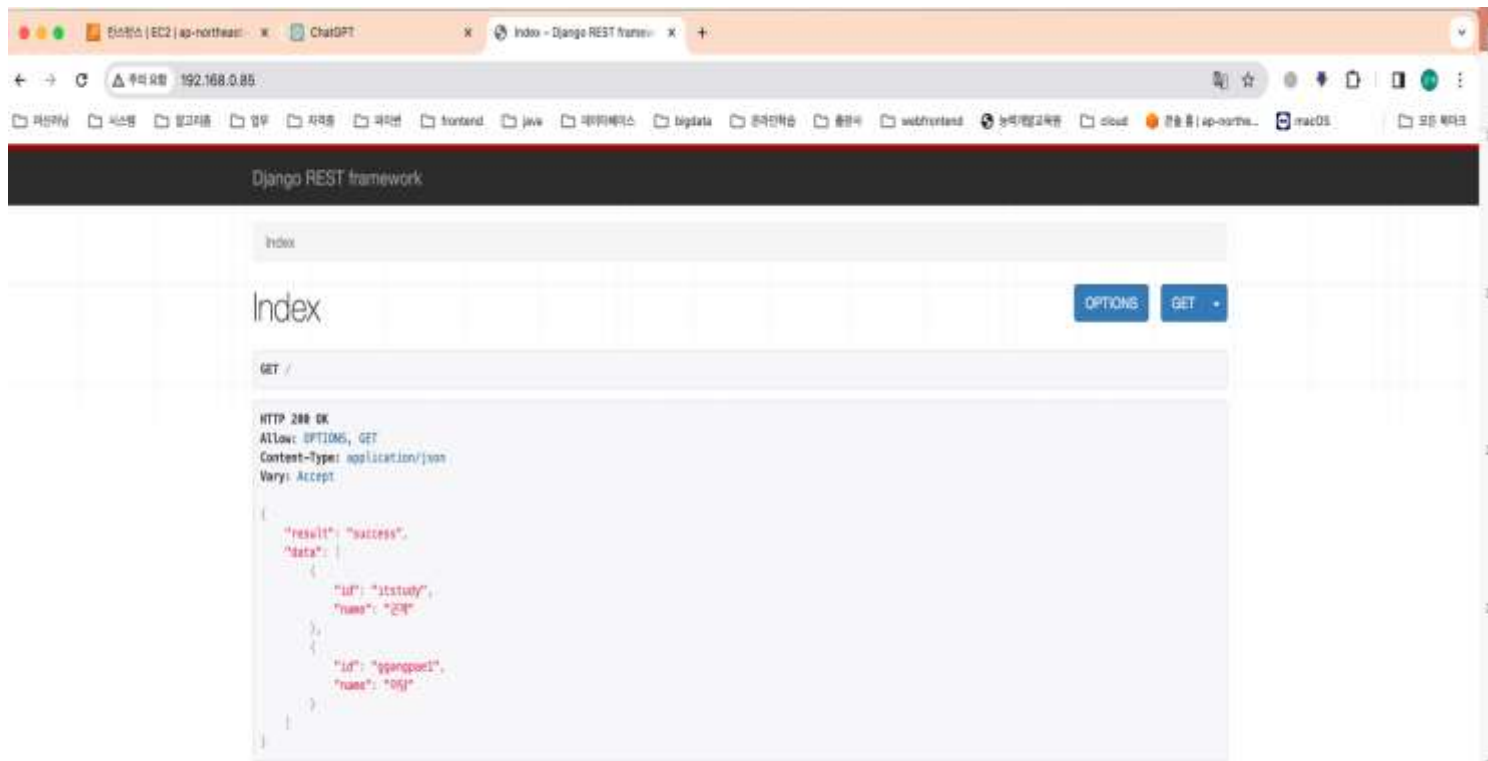
```
CMD ["python", "manage.py", "runserver", "0.0.0.0:80"]
```

AWS Container Service

- ❖ Django 애플리케이션을 Docker Image을 이용한 EC2 인스턴스에 배포
 - ✓ Docker Image 생성 및 실행
 - 이미지 빌드 후 실행
 - ◆ `docker build -t apiserver .`
 - ◆ `docker run --name apiserver -p 80:80 -d apiserver:latest`

AWS Container Service

- ❖ Django 애플리케이션을 Docker Image를 이용한 EC2 인스턴스에 배포
 - ✓ Docker Image 생성 및 실행
 - 확인



AWS Container Service

- ❖ Django 애플리케이션을 Docker Image를 이용한 EC2 인스턴스에 배포
 - ✓ Git Action을 활용한 Docker Hub에 이미지 배포
 - Git Hub에 업로드
 - Docker Hub에 Repository 생성
 - Docker Hub에서 Access Token 생성 – 오른쪽 상단의 아이디 옆을 클릭한 후 [Account Settings] – [Security] – [Access Token] – [New Access Token]

The screenshot shows the Docker Hub account settings page, specifically the 'Security' tab and 'Access Tokens' section. A 'New Access Token' button is located in the top right corner. The table below lists existing access tokens.

☐	Description	Source	Scope	Last Used	Created	
☐	DOCKERHUB	MANUAL	Read, Write, Delete	Nov 27, 2023 13:37:19	Nov 27, 2023 11:55:49	⋮
☐	sample	MANUAL	Read, Write, Delete	Oct 31, 2023 19:35:54	Oct 27, 2023 16:47:49	⋮
☐	1019	MANUAL	Read, Write, Delete	Oct 19, 2023 10:01:33	Oct 19, 2023 09:31:43	⋮
☐	1	MANUAL	Read, Write, Delete	Oct 17, 2023 08:13:53	Oct 17, 2023 07:43:33	⋮
☐	flaskweb	MANUAL	Read, Write, Delete	Oct 17, 2023 07:39:32	Oct 17, 2023 07:16:40	⋮

At the bottom of the table, there is a pagination bar showing '1' as the current page, with links for 2, 3, 4, and navigation arrows.

AWS Container Service

- ❖ Django 애플리케이션을 Docker Image를 이용한 EC2 인스턴스에 배포
 - ✓ Git Action을 활용한 Docker Hub에 이미지 배포
 - GIT HUB에서 Secret Key 등록: [Settings] – [Secrets and variable] – [Actions] – [New repository secret]
 - ◆ DOCKERHUB_USERNAME
 - ◆ DOCKERHUB_TOKEN



AWS Container Service

- ❖ Django 애플리케이션을 Docker Image를 이용한 EC2 인스턴스에 배포
 - ✓ Git Action을 활용한 Docker Hub에 이미지 배포
 - .github/workflows/ 디렉토리에 yaml 파일을 생성하고 push
name: django

on:
 push:
 branches: ["main"]
 pull_request:
 branches: ["main"]

AWS Container Service

❖ Django 애플리케이션을 Docker Image를 이용한 EC2 인스턴스에 배포

✓ Git Action을 활용한 Docker Hub에 이미지 배포

- .github/workflows/ 디렉토리에 yaml 파일을 생성하고 push

jobs:

build:

runs-on: ubuntu-latest

steps:

- name: Checkout

uses: actions/checkout@v3

- name: Set up Python 3.10

uses: actions/setup-python@v4

with:

python-version: '3.10'

- name: Install dependencies

run: |

python -m pip install --upgrade pip

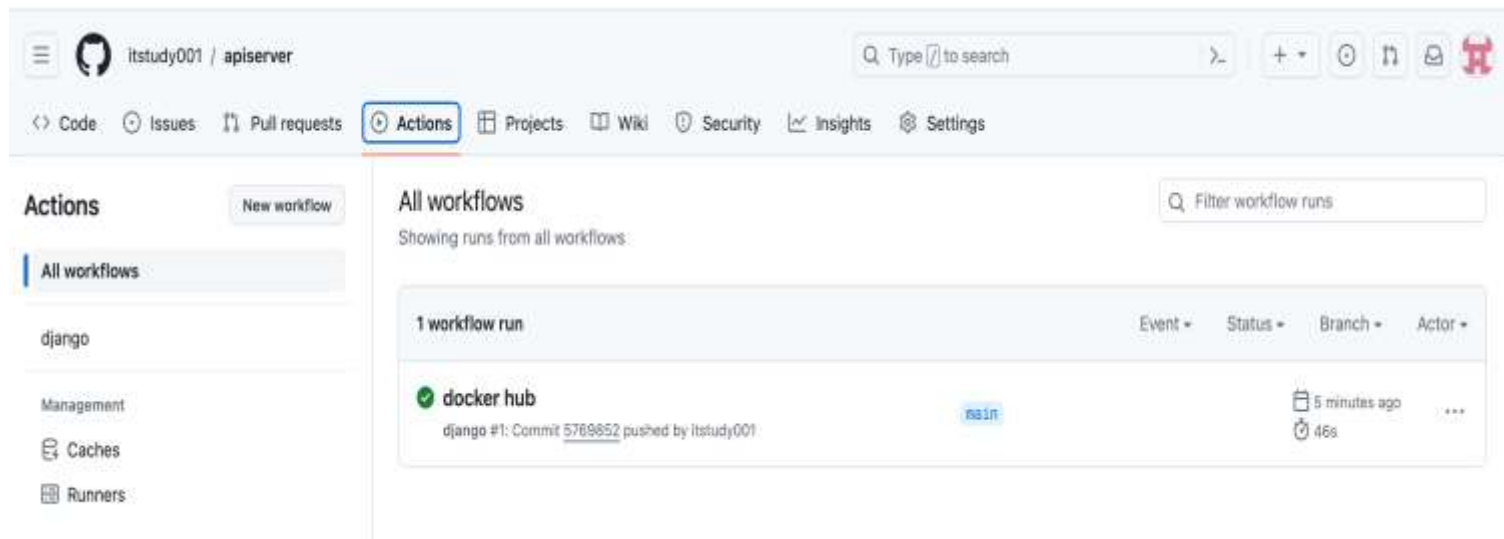
pip install -r requirements.txt

AWS Container Service

- ❖ Django 애플리케이션을 Docker Image를 이용한 EC2 인스턴스에 배포
 - ✓ Git Action을 활용한 Docker Hub에 이미지 배포
 - .github/workflows/ 디렉토리에 yaml 파일을 생성하고 push
 - name: Login to DockerHub
 - uses: docker/login-action@v1
 - with:
 - username: \${secrets.DOCKERHUB_USERNAME}
 - password: \${secrets.DOCKERHUB_TOKEN}
 - name: build and release to DockerHub
 - env:
 - NAME: \${secrets.DOCKERHUB_USERNAME}
 - REPO: apiserver
 - run: |
 - docker build -t \$REPO .
 - docker tag \$REPO:latest \$NAME/\$REPO:latest
 - docker push \$NAME/\$REPO:latest

AWS Container Service

- ❖ Django 애플리케이션을 Docker Image를 이용한 EC2 인스턴스에 배포
 - ✓ Git Action을 활용한 Docker Hub에 이미지 배포
 - 확인



AWS Container Service

- ❖ Django 애플리케이션을 Docker Image를 이용한 EC2 인스턴스에 배포
 - ✓ Git Action을 활용한 Docker Hub에 이미지 배포
 - 확인

The screenshot shows the Docker Hub interface for the repository 'ggangpae1 / apiserver'. The page includes a description section, a Docker commands section with a public view button, a tags section with a table of tags, and an automated builds section with an upgrade button.

ggangpae1 / apiserver

Description
This repository does not have a description

Last pushed: a few seconds ago

Docker commands [Public View](#)
To push a new tag to this repository:
`docker push gangpae1/apiserver:tagname`

Tags
This repository contains 1 tag(s).

Tag	OS	Type	Pulled	Pushed
latest		Image	---	a few seconds ago

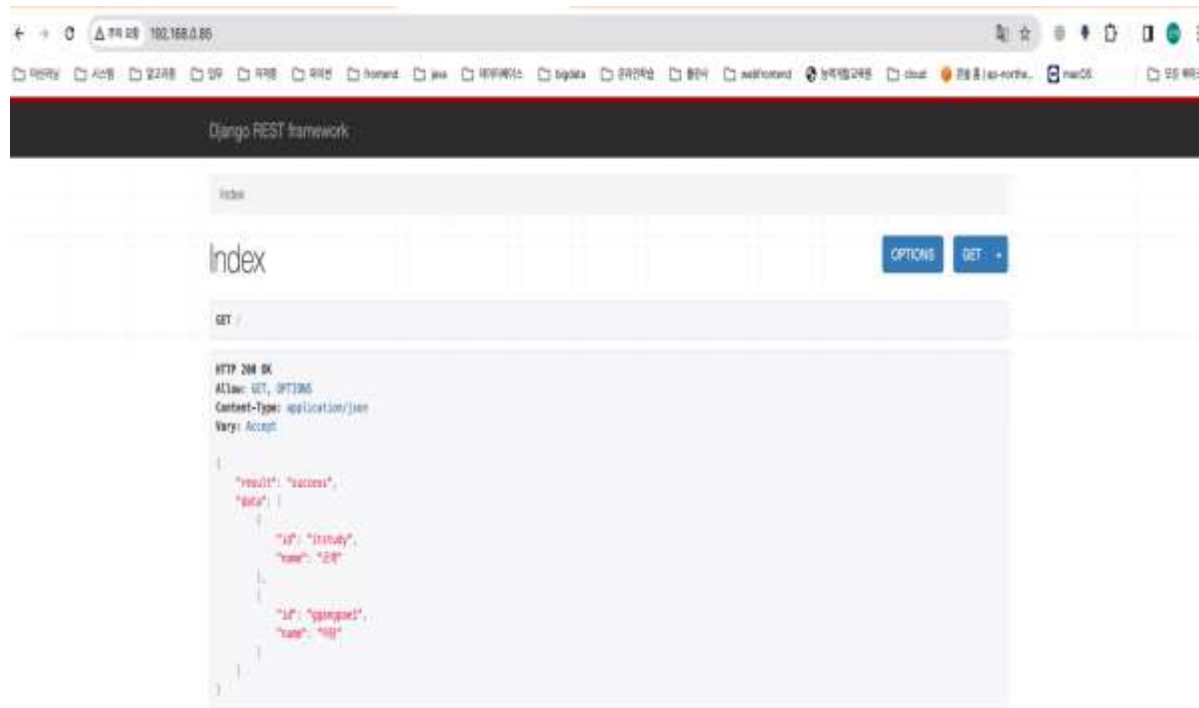
[See all](#)

Automated Builds
Manually pushing images to Hub? Connect your account to GitHub or Bitbucket to automatically build and tag new images whenever your code is updated, so you can focus your time on creating.
Available with Pro, Team and Business subscriptions. [Read more about automated builds](#) .

[Upgrade](#)

AWS Container Service

- ❖ Django 애플리케이션을 Docker Image를 이용한 EC2 인스턴스에 배포
 - ✓ Git Action을 활용한 Docker Hub에 이미지 배포
 - 확인
 - ◆ `docker run -dit -p 80:80 --name apiserver ggangpae1/apiserver`



AWS Container Service

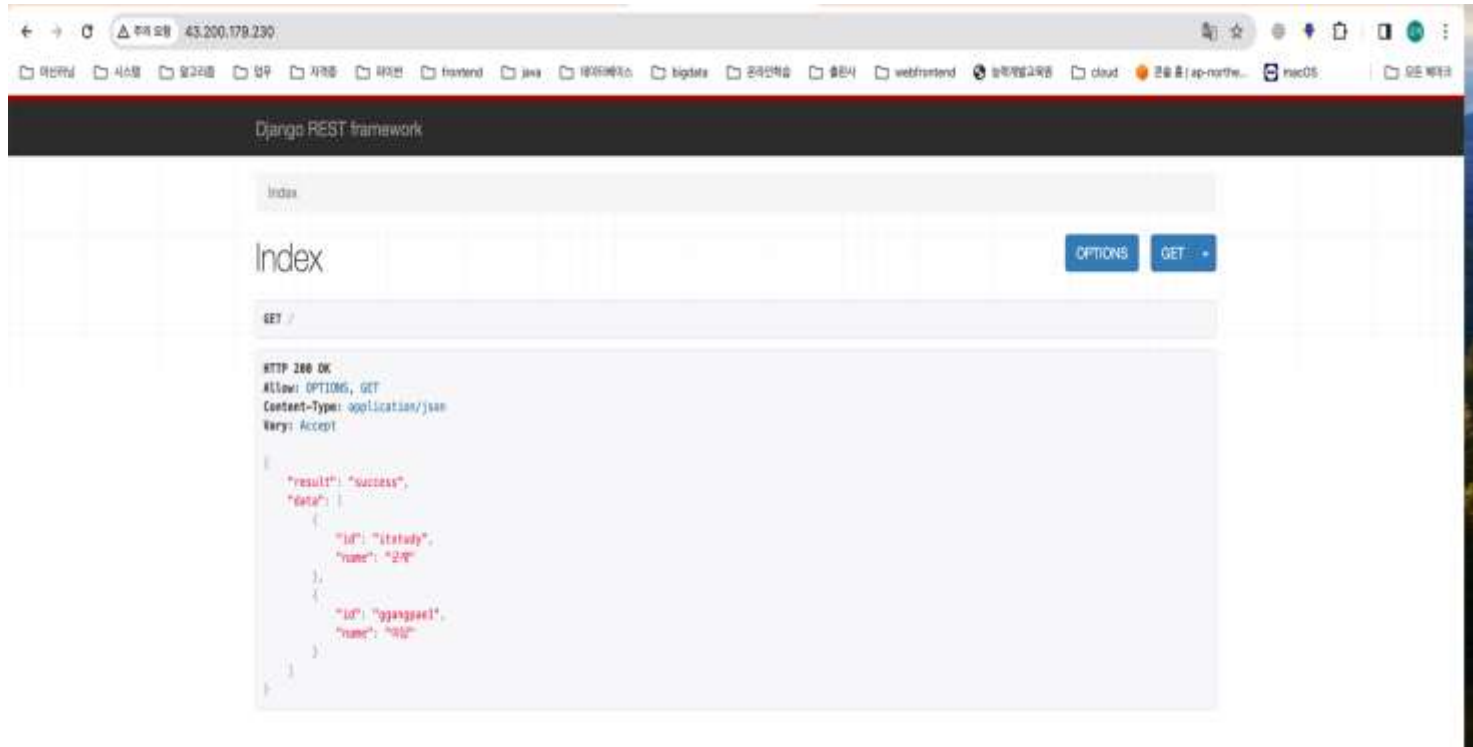
- ❖ Django 애플리케이션을 Docker Image를 이용한 EC2 인스턴스에 배포
 - ✓ EC2 인스턴스 작업
 - Docker 설치
 - ◆ 패키지 업데이트: `sudo apt-get update`
 - ◆ 패키지 설치: `sudo apt-get install apt-transport-https ca-certificates curl gnupg-agent software-properties-common`
 - ◆ Docker의 공식 GPG키를 추가
 - 경고 발생: `curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -`
 - 경고 발생하지 않음: `curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg`
 - ◆ Docker의 공식 apt 저장소를 추가
 - `sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"`
 - `echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null`

AWS Container Service

- ❖ Django 애플리케이션을 Docker Image를 이용한 EC2 인스턴스에 배포
 - ✓ EC2 인스턴스 작업
 - Docker 설치
 - ◆ 시스템 패키지 업데이트: `sudo apt-get update`
 - ◆ Docker 설치: `sudo apt-get install docker-ce docker-ce-cli containerd.io`
 - ◆ Docker 버전 확인: `sudo docker version`
 - ◆ Docker 서비스 등록 및 실행
 - `sudo systemctl enable docker`
 - `sudo systemctl start docker`

AWS Container Service

- ❖ Django 애플리케이션을 Docker Image를 이용한 EC2 배포
 - ✓ 이미지 다운로드 및 실행
 - `sudo docker run -p 80:80 -d --rm ggangpae1/apiserver`
 - ✓ 브라우저에서 퍼블릭 IP로 확인



AWS Container Service

❖ ECR

- ✓ Docker Hub와 비슷한 개념으로 Amazon Elastic Container Registry의 약자로 안전하고 확장 가능하고 신뢰할 수 있는 AWS 관리형 컨테이너 이미지 레지스트리 서비스
- ✓ Docker Hub와 동일하다고 볼 수 있지만 장점으로는 S3로 Docker Image를 관리하므로고가용성을 보장하고 AWS IAM 인증을 통해 이미지 push/pull에 대한 권한 관리가 가능
- ✓ Private 과 Public Repository 모두 가능

AWS Container Service

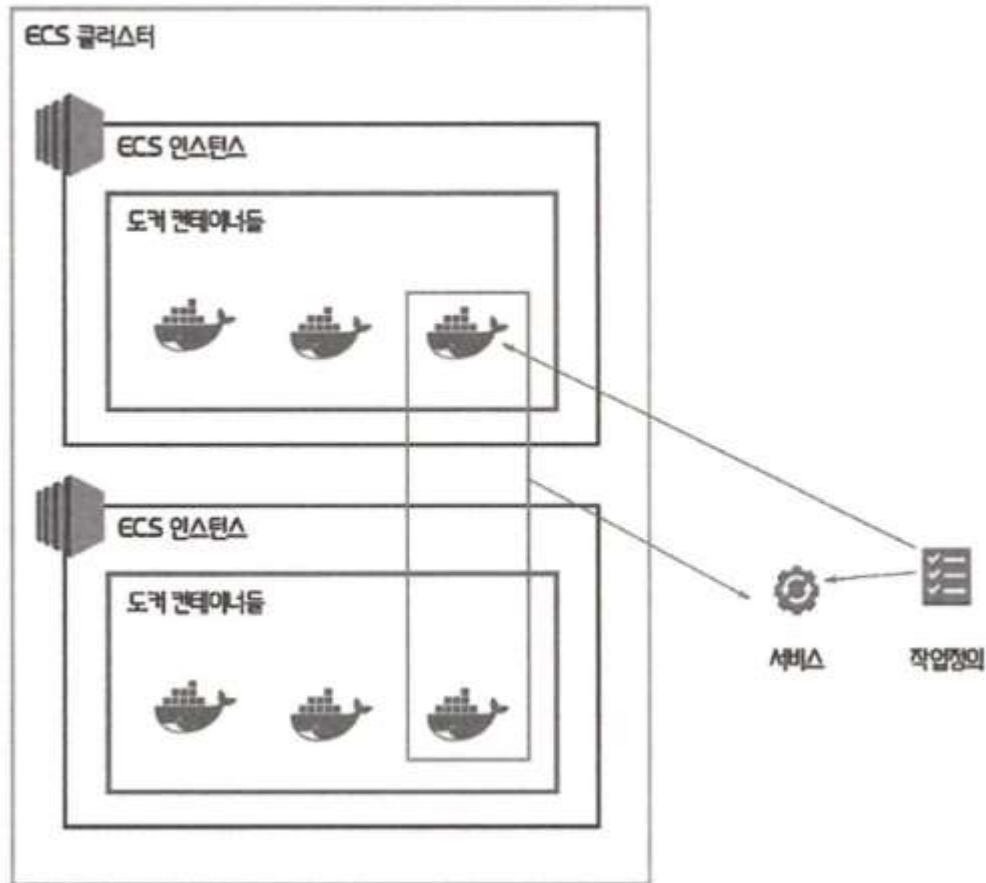
❖ ECS

✓ 개요

- Docker와 같은 컨테이너 가상화 도구를 처음 사용했을 때 컨테이너 기술의 장점 과 배포의 어려움을 동시에 느낌
- 운영 중인 Docker 컨테이너나 인스턴스가 몇 개 정도라면 오케스트레이션 도구까지는 필요 없겠지만 수십 개에서 수백 개가 필요하다면 어디에 어떤 컨테이너가 실행되고 있고 어떤 문제점이 있는지 모니터링하고 관리할 수 있는 도구가 필요한데 이러한 도구가 컨테이너 오케스트레이션(Container Orchestration)
- Docker에서 만든 스웜(Swarm), 구글의 노하우가 담긴 Kubernetes, 하시코프의 Nomad, AWS에서 제공하는 ECS, EKS 등 다양한 오케스트레이션 도구들이 있음
- AWS ECS(Elastic Container Service)는 AWS 컴퓨팅 서비스를 제공하는 오케스트레이션 서비스로 AWS 환경에 최적화되어 있음
- AWS Cloud 환경에서 사용하는 것을 전제로 하기 때문에 Docker의 모든 기능을 사용하기에는 일부 제약이 있으며 세세한 컨트롤이 어려움
- 구글에서 만든 Kubernetes는 오픈 소스로 개발되었기 때문에 사용자가 원하는 대로 변형하고 수정할 수 있지만 ECS는 Auto Scaling Task 기반 IAM 지원, Application Load Balancer 지원, Service Discovery, AWS Batch Release, Fargate 지원 등 AWS에 맞춤형 기능 등을 추가하며 최근 활용도가 더욱 높아지고 있음

AWS Container Service

- ❖ ECS
- ✓ 구조



AWS Container Service

❖ ECS

✓ 구성 요소

● 클러스터

- ◆ ECS의 가장 기본적인 단위는 클러스터
- ◆ 클러스터는 Docker 컨테이너를 실행할 수 있는 가상의 공간
- ◆ 클러스터는 프로젝트나 컨테이너의 성격에 따라서 나뉘질 수 있음
- ◆ ECS 클러스터는 기본적으로 EC2와 같은 컴퓨팅 자원을 기본적으로 포함하지 않은 논리적인 단위이므로 컴퓨팅 자원이 없는 빈 클러스터를 만드는 것도 가능
- ◆ EC2에 ECS-Client라는 서비스를 실행해서 특정 클러스터에 연결할 수 있는데 이렇게 클러스터에 연결된 EC2 인스턴스를 컨테이너 인스턴스라고 부름
- ◆ ECS-Client는 컨테이너 인스턴스의 자원을 모니터링 및 관리하고 클러스터로 요청된 컨테이너들을 적절하게 실행하는 역할을 수행

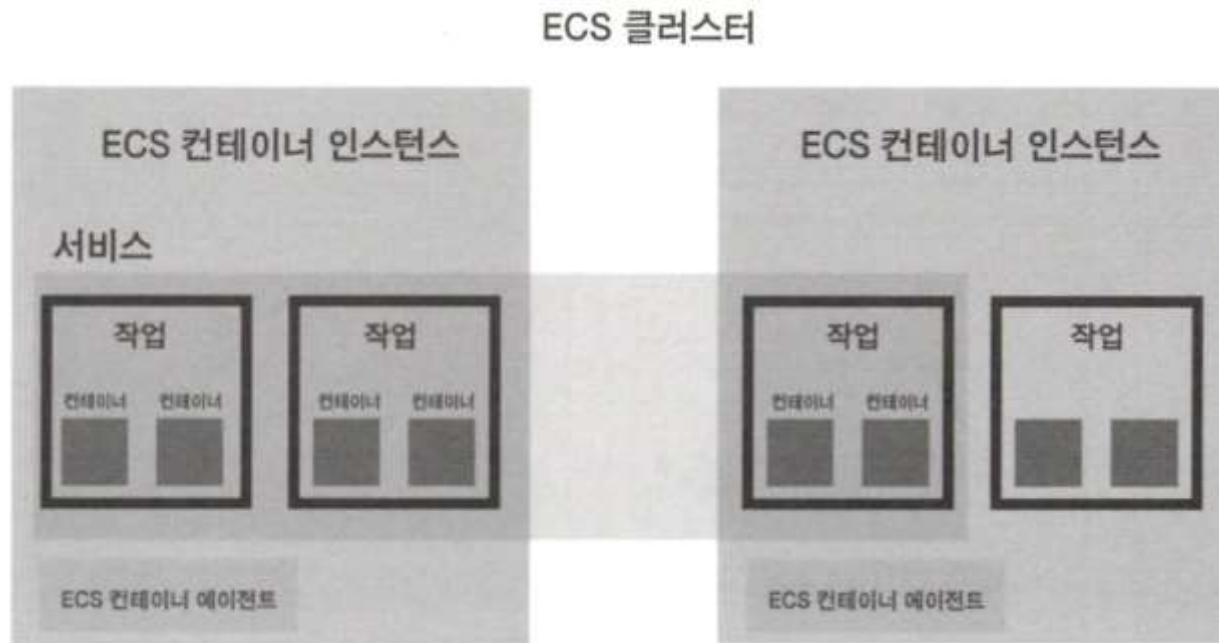
AWS Container Service

❖ ECS

✓ 구성 요소

● 클러스터

◆ ECS 클러스터 와 클러스터 인스턴스



AWS Container Service

❖ ECS

✓ 구성 요소

- 작업: pod
 - ◆ ECS에서 컨테이너를 실행하는 최소 단위는 작업
 - ◆ 클러스터 상에서 동작하는 1개 혹은 1개 이상의 컨테이너들을 지칭
 - ◆ 노드 서버를 위한 Docker 컨테이너와 데이터베이스를 위한 Docker 컨테이너가 필요하다면 각각 개별 작업으로 정의할 수도 있고 하나의 작업으로 정의할 수도 있음
 - ◆ 같은 하나의 작업으로 묶여 있다면 서로 네트워크 통신을 할 수 있음
- 작업 정의: docker-compose.yml
 - ◆ Docker 컨테이너를 실행하기 위해 정의한 도면
 - ◆ 컨테이너의 이미지, CPU/메모리 리소스 할당 설정, 포트 매핑, 볼륨 설정 같은 것들이 포함되며 기존 Docker 명령에서 가능했던 대부분 옵션이 설정 가능
 - ◆ 작업 정의를 바탕으로 작업 즉 Docker 컨테이너들을 실행할 수 있음

AWS Container Service

❖ ECS

✓ 구성 요소

- 서비스: deployment
 - ◆ 작업을 하나의 오케스트레이션 단위로 묶은 것이 서비스
 - ◆ 작업은 일반적으로 작업 개수로 정의하지만 서비스는 로드 밸런서 와 오토 스케일링을 연결하여 애플리케이션 단위로 조절
- 컨테이너 인스턴스
 - ◆ ECS는 컨테이너 배포를 EC2 인스턴스 기반에 올리도록 설계되어 있음
 - ◆ 컨테이너를 동작시킬 컴퓨터를 지칭하지만 파게이트 서비스는 컨테이너 인스턴스를 사용자가 직접 설정하거나 만들어줄 필요없이 AWS에서 관리 해주기 때문에 컨테이너 인스턴스를 필요로 하지 않음

AWS Container Service

❖ Fargate

- ✓ AWS 파게이트(Fargate)는 AWS에서 2017년에 발표한 서비스 – 서버리스 서비스
- ✓ 파게이트는 AWS의 매니지드 컨테이너 오케스트레이션 서비스인 ECS와 EKS를 기반으로 작동하는 서비스
- ✓ 특징은 Docker 컨테이너를 EC2 인스턴스 없이 독립적으로 실행할 수 있게 함
- ✓ EC2보다 컴퓨팅 성능을 더 세세하게 선택할 수 있으며 태스크 단위에서 IAM 롤이나 네트워크 인터페이스를 부여하는 것도 가능
- ✓ 람다 서비스보다는 저렴하지만 EC2보다는 비용이 비싸기 때문에 AWS 람다 와 EC2 기반의 ECS 컨테이너의 중간에 위치한 서비스라고 할 수 있음
- ✓ 컨테이너 오케스트레이션은 개발자 입장에서 애플리케이션 배포에 새로운 시대를 열
- ✓ AWS ECS, AWS EKS, Kubernetes, Docker Swarm, Apache Methos 와 같은 오케스트레이션을 통한 컨테이너 애플리케이션 배포는 빠르게 자리를 잡아가고 있음
- ✓ AWS에서 오케스트레이션 툴을 사용한다면 AWS의 EC2와 같은 컴퓨팅 자원을 묶어서 클러스터로 관리
- ✓ 컴퓨팅 자원을 관리하는 작업은 클러스터를 운영하는 데 있어서 가장 까다로운 작업 중 하나
- ✓ 파게이트가 비록 AWS 람다를 대체하기 위한 서비스는 아니지만 파게이트를 사용하면 AWS 람다와 같이 컴퓨팅 자원에 대한 추가적인 관리 없이 애플리케이션 실행이 가능
- ✓ 컨테이너를 기반으로 하기 때문에 람다에서 지원하는 언어들보다 훨씬 더 다양한 환경을 그대로 이식하는 것이 가능

AWS Container Service

❖ Fargate

- ✓ EC2 인스턴스보다 vCPU와 RAM을 세세하게 설정할 수 있으며 컨테이너에 IAM 롤을 지정하거나 ENI를 설정할 수도 있음
- ✓ 현재는 서울 리전에서도 파게이트 서비스를 사용할 수 있음
- ✓ ECS를 기반으로만 사용할 수 있음
- ✓ AWS 컨테이너 서비스



Amazon ECS



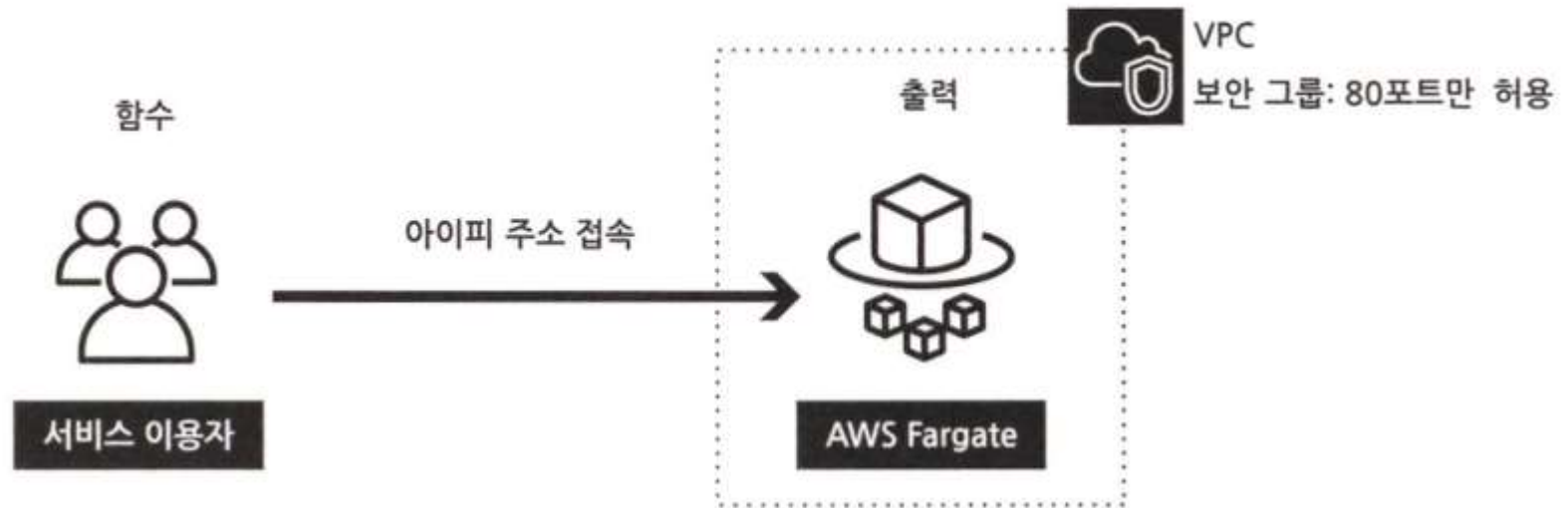
Amazon EKS



AWS Fargate

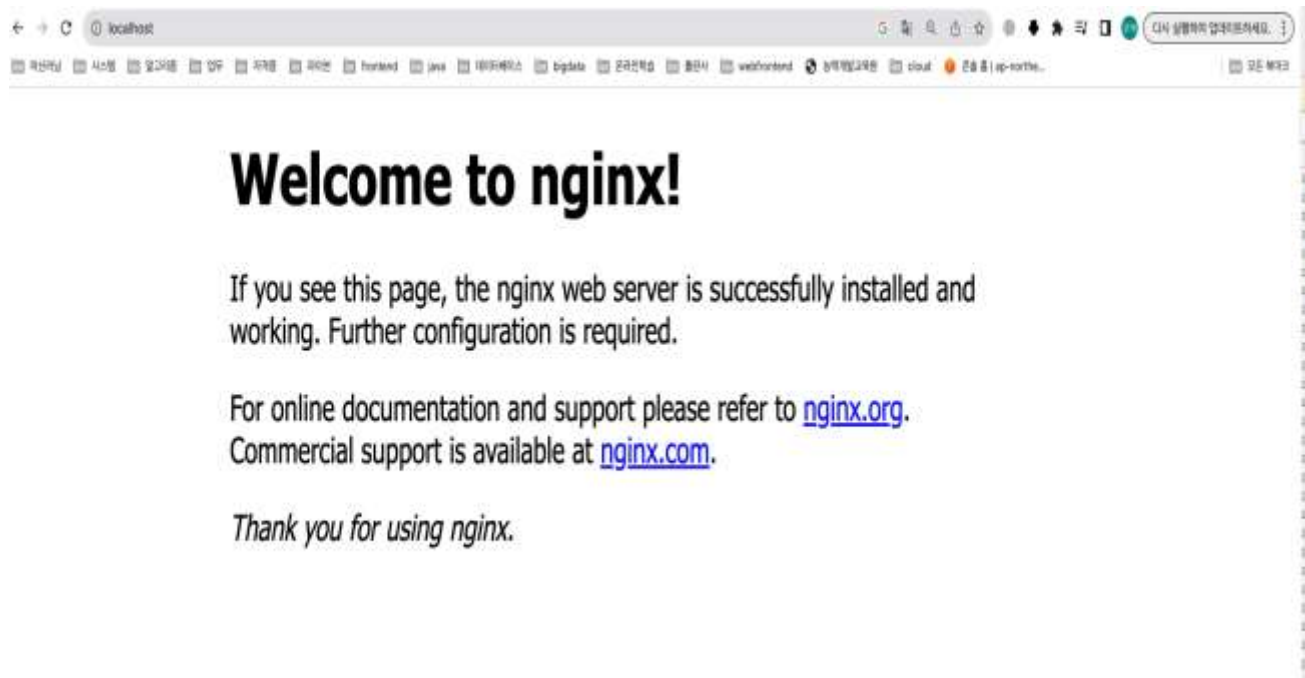
AWS Container Service

- ❖ ECS 컨테이너 서비스 구축
 - ✓ 아키텍처



AWS Container Service

- ❖ Nginx ECS 배포 – 태스크를 이용한 배포
 - ✓ nginx 이미지 확인
 - 컨테이너 생성: `docker run -it -p 80:80 nginx`
 - 브라우저에서 확인: localhost



AWS Container Service

- ❖ Nginx ECS 배포 – 태스크를 이용한 배포
 - ✓ 클러스터 생성



AWS Container Service

- ❖ Nginx ECS 배포 – 태스크를 이용한 배포
 - ✓ 클러스터 생성

Amazon Elastic Container Service

클러스터

네임스페이스 [선택](#)

태스크 정의

계정 설정 [선택](#)

AWS Copilot 설치 [🔗](#)

Amazon ECR [🔗](#)

리포지토리

AWS Batch [🔗](#)

설명서 [🔗](#)

제품 검색 [🔗](#)

구독 [🔗](#)

클러스터 생성 [정보](#)

Amazon ECS 클러스터는 태스크와 서비스를 함께 그룹화하고 용량과 공통적인 구성을 공유합니다. 모든 태스크, 서비스 및 용량은 클러스터에 속해야 합니다.

클러스터 구성

클러스터 이름

여기에 클러스터를 이름 입력합니다(예: DevCluster).

최대 255자까지 가능합니다. 유효한 문자는 문자(대문자 및 소문자), 숫자, 하이픈 및 밑줄입니다.

기본 네임스페이스 - 선택 사항

네임스페이스를 선택하여 애플리케이션을 구성하는 서비스 그룹을 지정합니다. 서비스 수준에서 이 값을 덮어쓸 수 있습니다.

[🔍 네임스페이스 지정](#)

▼ 인프라 [정보](#)

[서버리스](#)

클러스터는 2개의 용량 공급자를 통해 AWS Fargate(서버리스)에 대해 자동으로 구성됩니다. Amazon EC2 인스턴스를 추가하거나 ECS Anywhere를 사용하여 외부 인스턴스를 추가하세요.

☒ AWS Fargate(서버리스)

용량제입니다. 소켓, 메모리 또는 버스트 워크로드가 있거나 유지 관리 오버헤드가 있는 경우 사용합니다. 클러스터에는 기본적으로 Fargate 및 Fargate 스킴 용량 공급자가 있습니다.

☐ Amazon EC2 인스턴스

수동 구성. 일관된 리소스 요구가 있는 대규모 워크로드에 사용합니다.

☐ ECS Anywhere를 사용하는 외부 인스턴스

수동 구성. 데이터 센터 하이브리드를 추가하는 데 사용합니다.

AWS Container Service

- ❖ Nginx ECS 배포 – 태스크를 이용한 배포
 - ✓ 태스크 생성: 태스크 정의 클릭



AWS Container Service

- ❖ Nginx ECS 배포 – 태스크를 이용한 배포
 - ✓ 태스크 생성

태스크 정의 구성

태스크 정의 패밀리 [정보](#)
고유한 태스크 정의 패밀리 이름을 지정합니다.

최대 255개의 문자(대문자 및 소문자), 숫자, 하이픈 및 밑줄만 허용됩니다.

▼ 인프라 요구 사항

Specify the infrastructure requirements for the task definition.

시작 유형 [정보](#)
시작 유형을 선택하면 작업 정의 파라미터가 변경됩니다.

☒ **AWS Fargate**
컨테이너를 서버리스 컴퓨팅입니다.

☐ **Amazon EC2 인스턴스**
Amazon EC2 인스턴스를 사용하는 자체 관리형 인프라입니다.

OS, 아키텍처, 네트워크 모드
네트워크 모드는 작업에 사용되며 선택한 컴퓨팅 유형에 따라 달라집니다.

운영 체제/아키텍처 [정보](#)

네트워크 모드 [정보](#)

태스크 크기 [정보](#)
태스크를 위해 예약할 CPU 및 메모리의 양을 지정합니다.

CPU

메모리

AWS Container Service

- ❖ Nginx ECS 배포 – 태스크를 이용한 배포
 - ✓ 태스크 생성

컨테이너 - 1 정보

필수 컨테이너 제거

컨테이너 세부 정보

이름과 컨테이너 이미지를 지정하고 컨테이너를 필수로 표시할지 지정합니다. 각 태스크 정의에는 필수 컨테이너가 하나 이상 있어야 합니다.

이름

이미지 URI

필수 컨테이너

nginx

nginx

예

프라이빗 레지스트리 정보

Secrets Manager에 보안 인출 정보를 저장한 다음 보안 인출 정보를 사용하여 프라이빗 레지스트리의 이미지를 참조합니다.

☒ 프라이빗 레지스트리 인증

포트 매핑 정보

포트 매핑을 추가하여 컨테이너가 호스트의 포트에 액세스하여 트래픽을 송수신하도록 허용합니다. 포트 매핑 구성을 변경하면 연결된 서비스 연결 설정에 영향을 미칩니다.

컨테이너 포트

프로토콜

포트 이름

앱 프로토콜

제거

80

TCP

nginx-80-tcp

HTTP

제거

포트 매핑 추가

읽기 전용 루트 파일 시스템 정보

이 파라미터를 켜 경우 컨테이너에는 루트 파일 시스템에 대한 읽기 전용 액세스가 부여됩니다.

☐ 읽기 전용

리소스 할당 제한 - 조건부 정보

컨테이너 수준 CPU, GPU 및 메모리 제한은 태스크 수준 값과 다르며 컨테이너에 할당되는 리소스와 양을 정의합니다. 컨테이너가 하드 제한에 지정한 메모리를 초과하려고 시도하면 컨테이너가 종료됩니다.

CPU

GPU

메모리 하드 제한

메모리 소프트 제한

1

1

3

1

단위 vCPU

단위 기가바이트

단위 기가바이트

AWS Container Service

- ❖ Nginx ECS 배포 – 태스크를 이용한 배포
 - ✓ 클러스터에 태스크를 연결해서 배포

The screenshot displays the AWS Management Console interface for an ECS cluster named 'apiserver'. The top section, '클러스터 개요' (Cluster Overview), provides key details: ARN (arn:aws:ecs:ap-northeast-2:641022061021:cluster/apiserver), Status (Active), CloudWatch monitoring (Basic), and Registered container instances (0). Below this, a table lists services, tasks, and their states. The '서비스' (Services) tab is selected, showing a list of services. The table has columns for Service name, ARN, Status, Service type, Deployments and tasks, Last deployment, and Task definition. A message at the bottom states '서비스 없음' (No services) and '표시할 서비스가 없습니다.' (No services to display).

Service name	ARN	Status	Service type	Deployments and tasks	Last deployment	Task definition
서비스 없음 표시할 서비스가 없습니다.						

AWS Container Service

- ❖ Nginx ECS 배포 – 태스크를 이용한 배포
 - ✓ 클러스터에 태스크를 연결해서 배포

배포 구성

애플리케이션 유형 [정보](#)
실행할 애플리케이션 유형을 지정합니다.

☐ 서비스
중지 및 다시 시작할 수 있는 장기 실행 컴퓨팅 작업(예: 웹 애플리케이션)을 처리하는 태스크 그룹을 시작합니다.

☒ 태스크
실행하고 종료하는 독립 실행형 태스크(예: 배치 작업)를 시작합니다.

태스크 정의
기존 태스크 정의를 선택합니다. 새 태스크 정의를 생성하려면 [태스크 정의](#) 페이지로 이동합니다.

☐ 수동으로 개정 지정하기
선택한 작업 정의 패밀리와 가장 최근 개정 100개 중에서 선택하는 대신 수동으로 개정을 입력합니다.

패밀리 개정

nginxfamily ▼

2 (최신) ▼

원하는 태스크
시작할 태스크 수를 지정합니다.

1

작업 그룹
작업 그룹 이름이 같은 모든 작업은 분산 배치를 수행할 때 하나의 집합으로 간주됩니다.

AWS Container Service

- ❖ Nginx ECS 배포 – 태스크를 이용한 배포
 - ✓ 클러스터에 태스크를 연결해서 배포

▼ 네트워킹

VPC [정보](#)
사용할 가상 사설 클라우드를 선택합니다.

vpc-0c7db576b075e6a5d
기본값

서브넷
태스크 스케줄러가 배치 시 고려해야 하는 VPC 내 서브넷을 선택합니다.

서브넷 선택

subnet-0bcaf13c38a3170b4 ap-northeast-2d 172.31.48.0/20	subnet-0acde0836a0eab490 ap-northeast-2c 172.31.32.0/20	subnet-0fc7b211818954e26 ap-northeast-2a 172.31.0.0/20
subnet-0469bab0d1faaec8 ap-northeast-2b 172.31.16.0/20		

보안 그룹 [정보](#)
기존 보안 그룹을 선택하거나 새 보안 그룹을 생성합니다.

☒ 기존 보안 그룹 선택
☐ 새 보안 그룹 생성

보안 그룹 이름
기존 보안 그룹을 선택합니다.

보안 그룹 선택

sg-032e1bd0082b9339d
default

퍼블릭 IP [정보](#)
태스크의 탄력적 네트워크 인터페이스(ENI)에 퍼블릭 IP를 자동 지정할지 여부를 선택합니다.

☒ 켜짐

AWS Container Service

- ❖ Nginx ECS 배포 – 태스크를 이용한 배포
 - ✓ 클러스터에 태스크를 연결해서 배포
 - 보안 그룹의 인바운드 규칙에서 80 번 포트를 모든 IP에 개방

sg-0b21a49b19bc7ebe2 - default

작업 ▼

세부 정보

보안 그룹 이름 default	보안 그룹 ID sg-0b21a49b19bc7ebe2	설명 default VPC security group	VPC ID vpc-d664705a332efc6b3
소유자 753510247212	인바운드 규칙 수 1 권한 항목	아웃바운드 규칙 수 1 권한 항목	

인바운드 규칙 아웃바운드 규칙 태그

인바운드 규칙 (1/1)

🔍 보안 그룹 규칙 필터

🔄 태그 관리 인바운드 규칙 편집

< 1 > ⚙

☒	Name	보안 그룹 규칙 ID	IP 버전	유형	프로토콜	포트 범위	소스
☒	-	sgr-0cade712e3675553e	IPv4	HTTP	TCP	80	0.0.0.0/0

AWS Container Service

- ❖ Nginx ECS 배포 – 태스크를 이용한 배포
 - ✓ 브라우저에서 태스크의 퍼블릭 IP로 접속

Amazon Elastic Container Service > 클러스터 > apiserver > 태스크 > b3407358cb9848d78ea41422268e9342 > 구성

b3407358cb9848d78ea41422268e9342

구성 | 로그 | 네트워크 | 태그

작업 개요

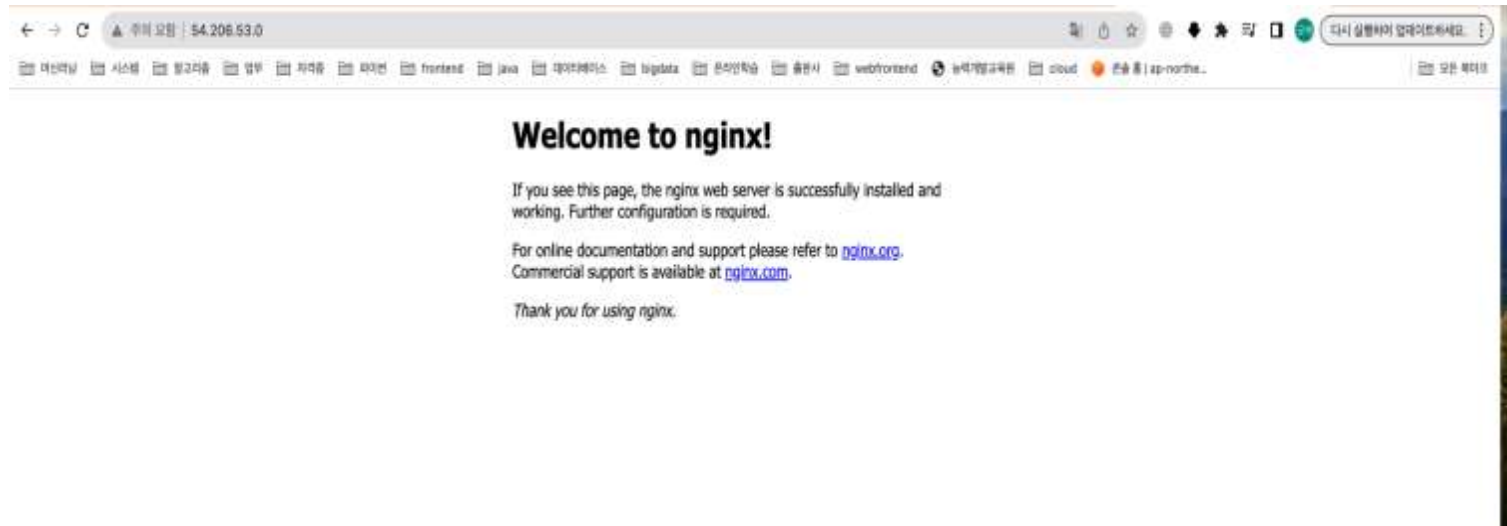
ARN arn:aws:ecs:ap-northeast-2:641022061021:task/apiserver/b3407358cb9848d78ea41422268e9342	마지막 상태 실행 중	종료하는 상태 실행 중	시작/생성 날짜 2023-12-17T01:28:48.590Z 2023-12-17T01:28:26.925Z
--	----------------	-----------------	--

구성

운영 체제/아키텍처 Linux/X86_64	용량 공급자 -	ENI ID eni-08e81ffdcac5e0a1	퍼블릭 IP 3.36.78.191 주소 열기
CPU 메모리 1 vCPU 3 GB	시작 유형 FARGATE	네트워크 모드 awsvpc	프라이빗 IP 172.31.4.170
플랫폼 버전 1.4.0	태스크 정의: 개칭 nginxfamily:2	서브넷 ID subnet-0fc7b211818954e26	MAC 주소 02:85:7f:d5:8e:e2
	작업 그룹 family:nginxfamily		

AWS Container Service

- ❖ Nginx ECS 배포 – 태스크를 이용한 배포
 - ✓ 브라우저에서 태스크의 퍼블릭 IP로 접속

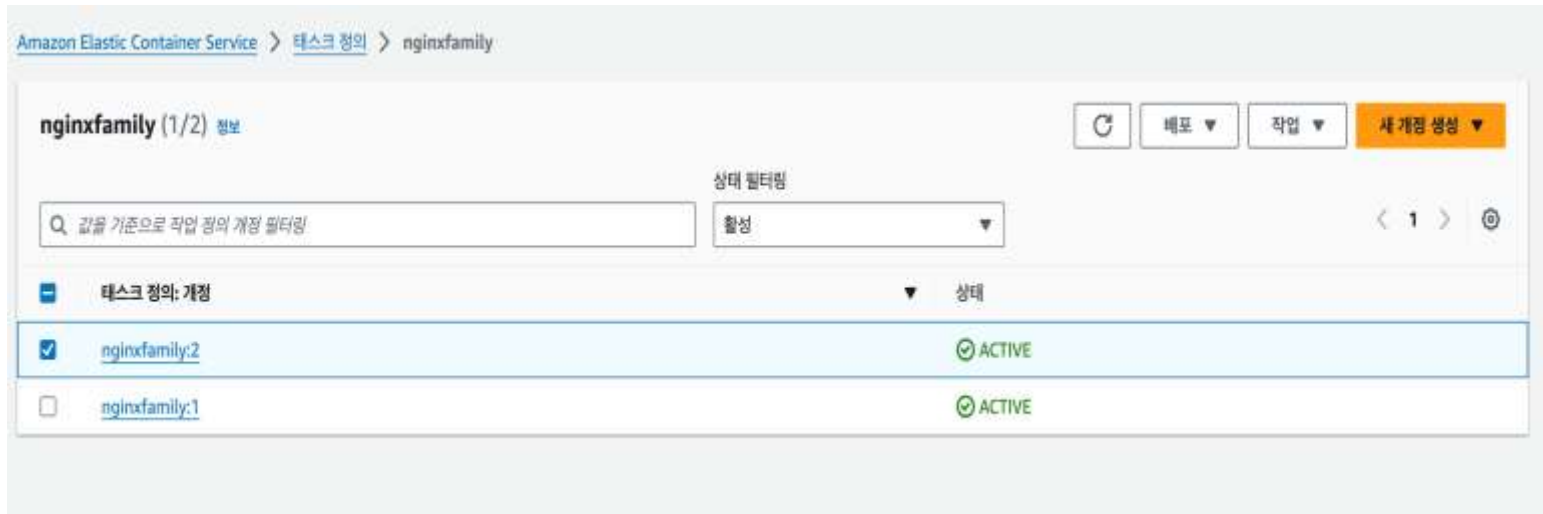


AWS Container Service

- ❖ Nginx ECS 배포 – 서비스를 이용한 배포
 - ✓ Service 사용
 - Task 대신 Service를 이용해서 배포를 하면 Load Balancer를 설정할 수 있음
 - Task 의 Public IP 대신 Load Balancer 의 Domain을 사용할 수 있음

AWS Container Service

- ❖ Nginx ECS 배포 – 서비스를 이용한 배포
 - ✓ 태스크를 새로 생성하거나 태스크 내에서 새 개정 생성



AWS Container Service

- ❖ Nginx ECS 배포 – 서비스를 이용한 배포
 - ✓ 클러스터에서 서비스 생성
 - 서비스 생성

AWS Container Service

- ❖ Nginx ECS 배포 – 서비스를 이용한 배포
 - ✓ 클러스터에서 서비스 생성
 - 서비스 생성

▼ 로드 밸런싱 - 선택 사항

로드 밸런서

로드 밸런서 유형 [정보](#)
서버리스에서 실행 중인 태스크에서 수신 트래픽을 분산하도록 로드 밸런서를 구성합니다.

Application Load Balancer ▼

Application Load Balancer

서버리스에서 실행 중인 태스크에서 수신 트래픽을 분산하도록 로드 밸런서를 선택할지 지정합니다.

☒ 새 로드 밸런서 생성
☐ 기존 로드 밸런서 사용

로드 밸런서 이름

로드 밸런서에 대한 고유한 이름을 지정합니다.

상태 검사 휴게 시간 [정보](#)

0

초

컨테이너

로드 밸런싱할 컨테이너 선택

itsstudydjango 80:80 ▼

리스너 [정보](#)
로드 밸런서가 연결 요청을 수신 대기할 포트 및 프로토콜을 지정합니다.

☒ 새 리스너 생성
☐ 기존 리스너 사용
기존 로드 밸런서를 선택해야 합니다.

포트

80

프로토콜

HTTP ▼

AWS Container Service

- ❖ Nginx ECS 배포 – 서비스를 이용한 배포
 - ✓ 클러스터에서 서비스 생성
 - 서비스 생성

대상 그룹 [정보](#)

새 대상 그룹을 생성할지, 아니면 로드 밸런서가 요청을 서비스의 태스크에 라우팅하는 데 사용할 기존 대상 그룹을 선택할지 지정합니다.

☒ 새 대상 그룹 생성

☐ 기존 대상 그룹 사용
기존 로드 밸런서를 선택해야 합니다.

대상 그룹 이름

itstudyloadbalance

프로토콜

HTTP

상태 확인 프로토콜

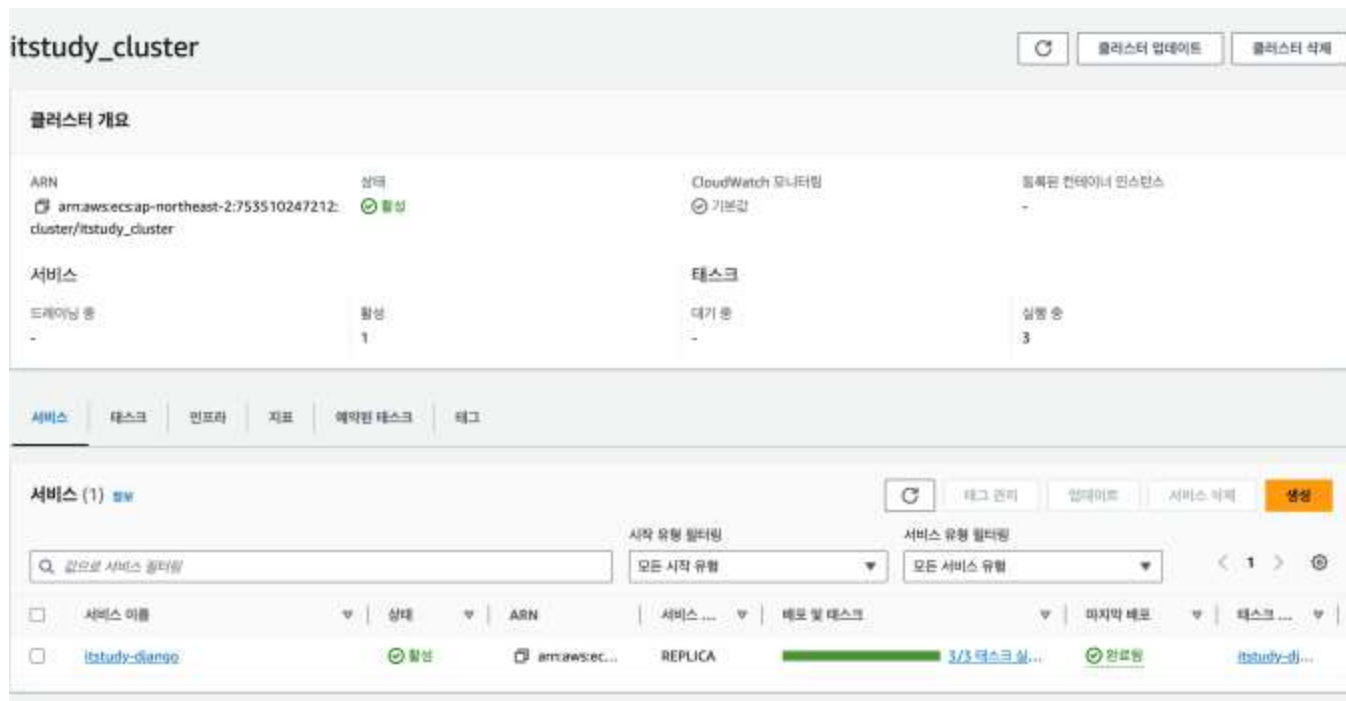
HTTP

상태 확인 경로 [정보](#)

/

AWS Container Service

- ❖ Nginx ECS 배포 – 서비스를 이용한 배포
 - ✓ 클러스터에서 서비스 생성
 - 서비스 생성



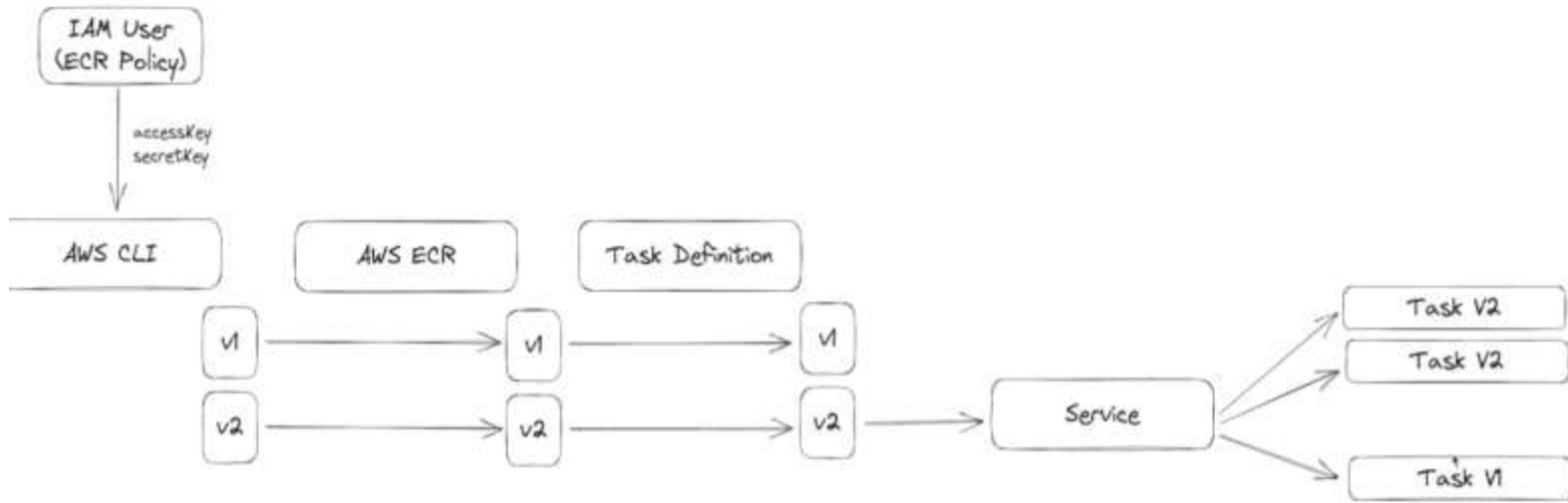
AWS Container Service

- ❖ Nginx ECS 배포 – 서비스를 이용한 배포
 - ✓ 클러스터에서 서비스 생성
 - Load Balancer 도메인으로 접속



AWS Container Service

❖ ECS 배포 전략



AWS Container Service

❖ ECS 배포 전략 – Green/Blue Strategy

태스크 (1/3)											
원하는 상태 필터링				시작 유형 필터링							
Q 속성 또는 값으로 태스크 필터링				실행 중				모든 시작 유형			
태스크	마지막 상태	원하는 상태	태스...	개정	상태	시작 위치	컨테이너 인스턴스	시작 유형	플랫폼		
78179...	실행 중	실행 중	itstudy-...	6	알 수 없음	13분 전	-	FARGATE	1.4.0		
96a1c...	실행 중	실행 중	itstudy-...	6	알 수 없음	12분 전	-	FARGATE	1.4.0		
aff502...	실행 중	실행 중	itstudy-...	6	알 수 없음	13분 전	-	FARGATE	1.4.0		

AWS Container Service

- ❖ ECS 배포 전략 – Green/Blue Strategy
 - ✓ 이전 태스크에서 새 개정 생성

The screenshot shows the AWS ECS console for a service named 'itstudy-django'. At the top, there are buttons for 'Refresh', 'Back', 'Tasks', and 'New Revision'. Below these is a search bar and a 'Status Filter' dropdown set to 'All'. The main table lists four tasks, all of which are 'ACTIVE'.

Task ID	Status
itstudy-django:7	ACTIVE
itstudy-django:6	ACTIVE
itstudy-django:5	ACTIVE
itstudy-django:4	ACTIVE

AWS Container Service

- ❖ ECS 배포 전략 – Green/Blue Strategy
 - ✓ 태스크에서 새 개정 생성

컨테이너 - 1 정보

필수 컨테이너 제거

컨테이너 세부 정보

이름과 컨테이너 이미지를 지정하고 컨테이너를 필수로 표시할지 지정합니다. 각 태스크 정의에는 필수 컨테이너가 하나 이상 있어야 합니다.

이름	이미지 URI	필수 컨테이너
itstudydjango	9bb18dae906bd736b7314e8f49bc99777352e11a26ec8	예 ▼

AWS Container Service

- ❖ ECS 배포 전략 – Green/Blue Strategy
 - ✓ 서비스 업데이트



AWS Container Service

- ❖ ECS 배포 전략 – Green/Blue Strategy
 - ✓ 서비스 업데이트

배포 구성

☐ 새 배포 강제 적용

태스크 정의

기존 태스크 정의를 선택합니다. 새 태스크 정의를 생성하려면 [태스크 정의](#) 페이지로 이동합니다.

☐ 수동으로 개정 지정하기

선택한 작업 정의 배열의 가장 최근 개정 100개 중에서 선택하는 대신 수동으로 개정을 입력합니다.

패밀리

개정

itstudy-django

8 (최신)

서비스 유형

REPLICA

원하는 태스크

시작할 태스크 수를 지정합니다.

3

최소 실행 작업 비율(%)

서비스 배포 중 허용되는 실행 중인 태스크의 최소 백분율을 지정합니다.

100

값(%)

최대 실행 작업 비율(%)

서비스 배포 중 허용되는 실행 중인 태스크의 최대 백분율을 지정합니다.

200

값(%)

▶ 배포 실패 감지

▶ 컴류팅 구성 (고급)

AWS Container Service

- ❖ ECS 배포 전략 – Green/Blue Strategy
 - ✓ 서비스 업데이트

태스크 (7)

실행 중인 상태 필터링

시작 유형 필터링

모든 시작 유형

태스크	마지막 상태	원하는 상태	태스크 정의	상태	시작 위치	컨테이너 인스턴스	시작 유형	클러스터
289da...	실행 중	실행 중	nginxfamily:4	① 알 수 없음	11초 전	-	FARGATE	1.4
551ea...	실행 중	실행 중	nginxfamily:4	① 알 수 없음	21초 전	-	FARGATE	1.4
72eef...	실행 중	실행 중	nginxfamily:3	① 알 수 없음	4분 전	-	FARGATE	1.4
b3407...	실행 중	실행 중	nginxfamily:2	① 알 수 없음	18분 전	-	FARGATE	1.4
b3721...	실행 중	실행 중	nginxfamily:3	① 알 수 없음	4분 전	-	FARGATE	1.4
ce1da...	실행 중	실행 중	nginxfamily:4	① 알 수 없음	17초 전	-	FARGATE	1.4
e9fd0...	실행 중	실행 중	nginxfamily:3	① 알 수 없음	4분 전	-	FARGATE	1.4

AWS Container Service

- ❖ ECS 배포 전략 – Green/Blue Strategy
 - ✓ 서비스 업데이트



태스크 (4)

속성 또는 값으로 태스크 필터링

원하는 상태 필터링: 실행 중

시작 유형 필터링: 모든 시작 유형

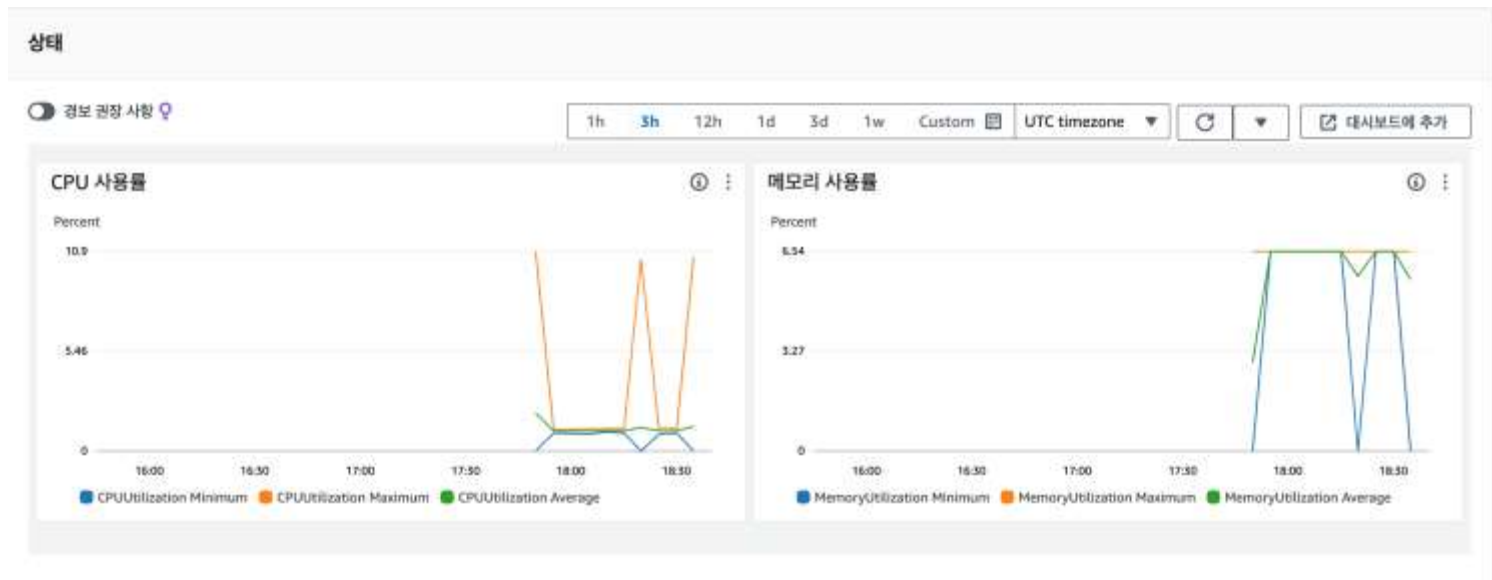
☐	태스크	마지막 상태	원하는 상태	태스크 정의	상태	시작 위치	컨테이너 인스턴스	시작 유형	플랫폼
☐	289da...	실행 중	실행 중	nginxfamily:4	알 수 없음	1분 전	-	FARGATE	1.4
☐	551ea...	실행 중	실행 중	nginxfamily:4	알 수 없음	2분 전	-	FARGATE	1.4
☐	b3407...	실행 중	실행 중	nginxfamily:2	알 수 없음	20분 전	-	FARGATE	1.4
☐	ce1da...	실행 중	실행 중	nginxfamily:4	알 수 없음	1분 전	-	FARGATE	1.4

AWS Container Service

❖ Auto Scaling

✓ 트리거 조건

- CPU 사용량
- 메모리 사용량
- 트래픽



AWS Container Service

❖ Auto Scaling

✓ 설정 내용

- 트리거 조건
- 확장 단위
- 쿨다운: scale out(수평 확장), scale in(수평 축소) 하는데 모니터링 하는 시간

AWS Container Service

❖ Auto Scaling

▼ 서비스 자동 크기 조정 - 선택 사항

CloudWatch 경보에 대응하여 지정한 범위 내에서 원하는 서비스 개수를 자동으로 늘리거나 줄입니다. 언제든지 서비스 Auto Scaling 구성을 수정하여 애플리케이션의 요구 사항을 충족할 수 있습니다.

☒ 서비스 자동 크기 조정 사용

서비스 자동 크기 조정을 구성하여 원하는 서비스 개수를 조정합니다.

최소 태스크 개수

서비스 자동 크기 조정이 원하는 서비스 개수를 조정할 수 있는 하한선입니다.

최대 태스크 개수

서비스 자동 크기 조정이 원하는 서비스 개수를 조정할 수 있는 상한선입니다.

AWS Container Service

❖ Auto Scaling

조정 정책

재거

조정 정책 유형

정보

대상 추적 또는 단계 크기 조정 정책을 생성합니다.

☐ 대상 추적

서비스가 실행하는 태스크의 수를 특정 지표의 대상 값에 따라 늘리거나 줄입니다.

☒ 단계 조정

경보 위반의 크기에 따라 달라지는 말련의 크기 조정(단계 조정이라고 함)에 따라 서비스가 실행하는 태스크의 수를 늘리거나 줄입니다.

정책 이름

itstudyautoscaling

Amazon ECS 서비스 경보

새 경보를 생성할지, 아니면 기존 경보를 선택할지 지정합니다.

☒ Amazon ECS 지표를 사용하여 새 경보를 생성합니다.

☐ 기존 경보 사용

경보 이름

cpuScaleout

Amazon ECS 서비스 지표

CPUUtilization

최대 255자까지 가능합니다. 유효한 문자는 문자(대문자 및 소문자), 숫자, 하이픈 및 밑줄입니다.

통계

평균

기간

1분

경보 조건

선택한 지표를 정의한 임계값과 비교하는 방법을 정의합니다.

☒ 초과
> 임계값

☐ 크거나 같음
>= 임계값

☐ 작거나 같음
<= 임계값

☐ 미만
< 임계값

AWS Container Service

❖ Auto Scaling

지표 비교 임계값		경보 시작을 위한 평가 기간	
50		1	
임계값은 0보다 크거나 같아야 합니다.		평가 기간은 1보다 크거나 같은 정수여야 합니다.	
크기 조정 작업			
단계 조정 정책에 대한 작업을 하나 이상 추가합니다.			
조정		지표 임계값	
작업	값	유형	하한 상한
추가 ▼	1	태스크 ▼	50 +infinity
			제거
새 스케일링 액션 추가			
휴지 기간			
크기 조정 작업 사이에 일시 중지할 초를 설정합니다.			
300			

AWS Container Service

❖ Auto Scaling

조정 정책

제거

조정 정책 유형 **정보**

대상 추적 또는 단계 크기 조정 정책을 생성합니다.

☐ 대상 추적
서비스가 실행하는 태스크의 수를 특정 지표의 대상 값에 따라 늘리거나 줄입니다.

☒ 단계 조정
경보 위반의 크기에 따라 달라지는 일련의 크기 조정(단계 조정이라고 함)에 따라 서비스가 실행하는 태스크의 수를 늘리거나 줄입니다.

정책 이름

cpuScaleIn

Amazon ECS 서비스 경보

새 경보를 생성할지, 아니면 기존 경보를 선택할지 지정합니다.

☒ Amazon ECS 지표를 사용하여 새 경보를 생성합니다.

☐ 기존 경보 사용

경보 이름

cpuScaleIn

Amazon ECS 서비스 지표

CPUUtilization ▼

최대 255자까지 가능합니다. 유효한 문자는 문자(대문자 및 소문자), 숫자, 하이픈 및 밑줄입니다.

AWS Container Service

❖ Auto Scaling

통계

기간

평균

1분

경보 조건

선택한 지표를 정의한 임계값과 비교하는 방법을 정의합니다.

☐ 초과
> 임계값

☐ 크거나 같음
>= 임계값

☐ 작거나 같음
<= 임계값

☒ 미만
< 임계값

지표 비교 임계값

20

임계값은 0보다 크거나 같아야 합니다.

경보 시작을 위한 평가 기간

1

평가 기간은 1보다 크거나 같은 정수여야 합니다.

크기 조정 작업

단계 조정 정책에 대한 작업을 하나 이상 추가합니다.

조정

작업

값

유형

지표 임계값

하한

상한

제거

1

태스크

-infinity

20

제거

새 스케일링 액션 추가

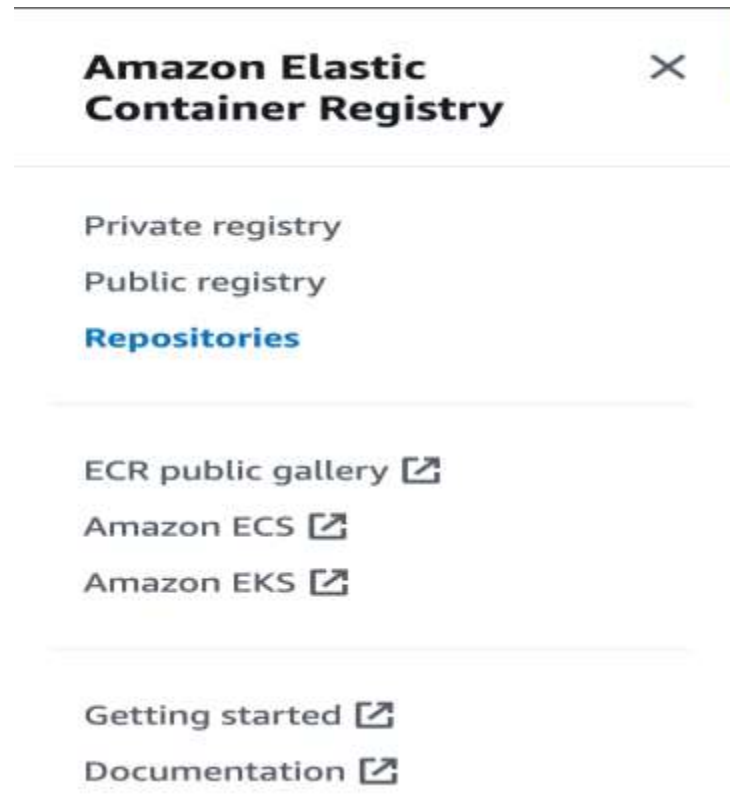
휴지 기간

크기 조정 작업 사이에 일시 중지할 초를 설정합니다.

300

AWS Container Service

- ❖ Django Application CI/CD Pipeline 설정
 - ✓ ECR(Elastic Container Registry) 서비스에 접속해서 Repository 생성



AWS Container Service

- ❖ Django Application CI/CD Pipeline 설정
 - ✓ ECR(Elastic Container Registry) 서비스에 접속해서 Repository 생성 - apiserver

Amazon ECR > 프라이빗 레지스트리 > 리포지토리 > 리포지토리 생성

리포지토리 생성

일반 설정

표시 여부 설정 [정보](#)
리포지토리에 대한 가시성 설정을 선택합니다.

☒ **프라이빗**
액세스는 IAM 및 리포지토리 정책 권한에 의해 관리됩니다.

☐ **퍼블릭**
이미지 물에 대해 공개적으로 표시되고 액세스할 수 있습니다.

리포지토리 이름
간결한 이름을 제공합니다. 개발자는 이름으로 리포지토리 콘텐츠를 식별할 수 있어야 합니다.

641022061021.dkr.ecr.ap-northeast-2.amazonaws.com/

최대 256자 중 9자(최소 2자 이상). The name must start with a letter and can only contain lowercase letters, numbers, hyphens, underscores, periods and forward slashes.

태그 변경 불가능 [정보](#)
동일한 태그를 사용하는 후속 이미지 푸시가 이미지 태그를 덮어쓰지 않도록 방지하려면 [태그 변경 불가능]을 활성화합니다. 이미지 태그를 덮어쓰려면 [태그 변경 불가능]을 비활성화합니다.

☐ 비활성화됨

[i](#) 리포지토리가 생성되면 해당 리포지토리의 가시성 설정을 변경할 수 없습니다.

AWS Container Service

- ❖ Django Application CI/CD Pipeline 설정
 - ✓ ECR(Elastic Container Registry) 서비스에 접속해서 Repository 생성 - apiserver

이미지 스캔 설정

 **사용 중단 경고**
리포지토리 수준의 ScanOnPush 구성은 더 이상 사용되지 않으며 레지스트리 수준 스캔 필터로 대체됩니다.

푸시할 때 스캔
리포지토리에 푸시된 후 각 이미지를 자동으로 스캔하려면 [푸시할 때 스캔]을 활성화합니다. 비활성화하는 경우 스캔 결과를 얻으려면 각 이미지 스캔을 수동으로 시작해야 합니다.

☐ 비활성화된

암호화 설정

KMS 암호화
기본 암호화 설정을 사용하는 대신 AWS Key Management Service(KMS)를 사용하여 이 리포지토리에 저장된 이미지를 암호화할 수 있습니다.

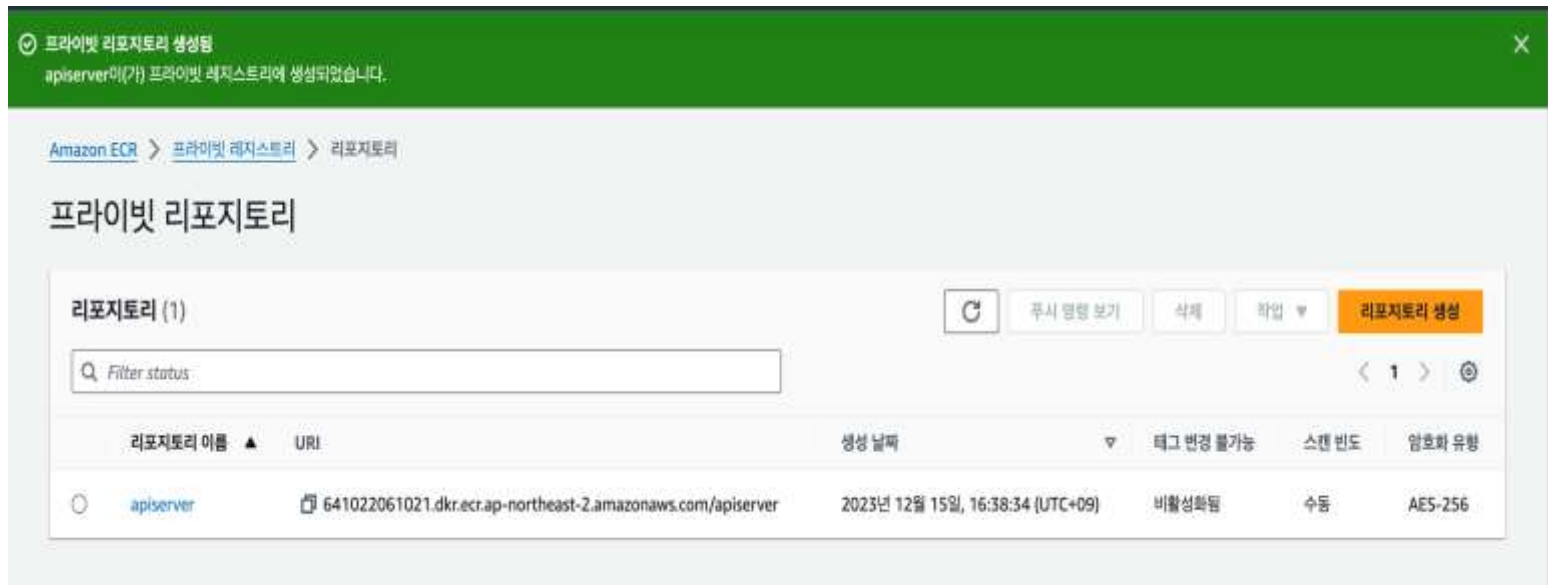
☐ 비활성화된

 리포지토리 생성 후 KMS 암호화 설정을 변경하거나 비활성화할 수 없습니다.

[취소](#) [리포지토리 생성](#)

AWS Container Service

- ❖ Django Application CI/CD Pipeline 설정
 - ✓ ECR(Elastic Container Registry) 서비스에 접속해서 Repository 생성 - apiserver



AWS Container Service

❖ Django Application CI/CD Pipeline 설정

✓ IAM 서비스에서 ECR에 이미지를 push할 수 있는 정책 생성

- 정책 생성(private_image_push) – 리전을 수정하고 숫자 와 이름은 ECR에서 가져와서 작성

◆ 정책 생성 클릭

The screenshot shows the AWS IAM console interface. On the left, the 'Identity and Access Management(IAM)' sidebar is visible with a search bar and navigation links. The main content area displays the 'Policy (1164)' page, which includes a search bar, a dropdown menu for '필터된 기준 유형' (Filtered criteria type), and a table of policies. The table has columns for '정책 이름' (Policy name), '유형' (Type), '다음과 같이 사용' (Used as follows), and '설명' (Description). The policies listed are:

정책 이름	유형	다음과 같이 사용	설명
AccessAnalyzerServiceRolePolicy	AWS 관리형	없음	Allow Access Analyzer to analyze resou...
AdministratorAccess	AWS 관리형 - 직무	없음	Provides full access to AWS services an...
AdministratorAccess-Ampify	AWS 관리형	없음	Grants account administrative permiss...
AdministratorAccess-AWS-EC2-Role	AWS 관리형	없음	Grants account administrative permiss...

AWS Container Service

❖ Django Application CI/CD Pipeline 설정

✓ IAM 서비스에서 ECR에 이미지를 push할 수 있는 정책 생성

- 정책 생성(private_image_push) – 리전을 수정하고 숫자 와 이름은 ECR에서 가져와서 작성

◆ JSON 클릭

권한 지정 정보

서비스, 작업, 리소스 및 조건을 선택하여 권한을 추가합니다. JSON 편집기를 사용하여 권한 설명문을 작성합니다.

정책 편집기 시각적 JSON 작업 ▼ 권한 지정

▼ 서비스 선택

서비스의 특정 리소스에 대해 수행할 수 있는 작업을 지정합니다.

서비스

서비스 선택

+ 권한 더 추가

취소 다음

AWS Container Service

❖ Django Application CI/CD Pipeline 설정

✓ ECR에 이미지를 push할 수 있는 정책 생성

- 정책 생성(private_image_push) – 리전을 수정하고 숫자 와 이름은 ECR에서 가져와서 작성

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ecr:CompleteLayerUpload",
        "ecr:UploadLayerPart",
        "ecr:InitiateLayerUpload",
        "ecr:BatchCheckLayerAvailability",
        "ecr:PutImage"
      ],
      "Resource": "arn:aws:ecr:ap-northeast-2:753510247212:repository/apiserver"
    }
  ],
}
```

AWS Container Service

❖ Django Application CI/CD Pipeline 설정

✓ ECR에 이미지를 push할 수 있는 정책 생성

- 정책 생성(private_image_push) – 리전을 수정하고 숫자 와 이름은 ECR에서 가져와서 작성

```
{
  "Effect": "Allow",
  "Action": "ecr:GetAuthorizationToken",
  "Resource": "*"
}
```

AWS Container Service

❖ Django Application CI/CD Pipeline 설정

✓ IAM에서 ECR에 이미지를 push할 수 있는 정책 생성

- 정책 생성(private_image_push) – 리전을 수정하고 숫자 와 이름은 ECR에서 가져와서 작성

정책 세부 정보

정책 이름
이 정책을 식별하는 데 사용되는 이름을 입력합니다.

private_image_push

최대 128자입니다. 영문자 및 '+', '@', '-' 문자를 사용하세요.

설명 - 선택 사항
이 정책에 대하여 간단한 설명을 추가합니다.

ECR Image Push

최대 1,000자입니다. 영문자 및 '+', '@', '-' 문자를 사용하세요.

이 정책에 정의된 권한 정보

이 정책 문서에 정의된 권한은 허용되거나 거부되는 작업을 지정합니다. IAM 자격 증명(사용자, 사용자 그룹 또는 역할)에 대한 권한을 정의하려면 여기에 정책을 연결합니다.

검색

허용(서비스 402개 중 1개) 나머지 서비스 401개 표시

서비스	액세스 수준	리소스	요청 조건
Elastic Container Registry	제한적: 읽기, 쓰기	Multiple	None

AWS Container Service

- ❖ Django Application CI/CD Pipeline 설정
 - ✓ OpenID Connect를 이용한 Git Hub AWS 로그인
 - IAM 에서 ID 제공업체를 클릭해서 추가

Adding the identity provider to AWS

To add the GitHub OIDC provider to IAM, see the [AWS documentation](#).

- For the provider URL: Use `https://token.actions.githubusercontent.com`
- For the "Audience": Use `sts.amazonaws.com` if you are using the [official action](#).

AWS Container Service

- ❖ Django Application CI/CD Pipeline 설정
 - ✓ OpenID Connect를 이용한 Git Hub AWS 로그인
 - IAM 에서 자격 증명 공급자를 클릭해서 추가
 - ◆ URL 확인: <https://docs.github.com/en/actions/deployment/security-hardening-your-deployments/configuring-openid-connect-in-amazon-web-services>

Adding the identity provider to AWS

To add the GitHub OIDC provider to IAM, see the [AWS documentation](#).

- For the provider URL: Use `https://token.actions.githubusercontent.com`
- For the "Audience": Use `sts.amazonaws.com` if you are using the [official action](#).

Configuring the role and trust policy

To configure the role and trust in IAM, see the AWS documentation "[Configure AWS Credentials for GitHub Actions](#)" and "[Configuring a role for GitHub OIDC identity provider](#)."

Note: AWS Identity and Access Management (IAM) recommends that users evaluate the IAM condition key, `token.actions.githubusercontent.com:sub`, in the trust policy of any role that trusts GitHub's OIDC identity provider (IdP). Evaluating this condition key in the role trust policy limits which GitHub actions are able to assume the role.

AWS Container Service

- ❖ Django Application CI/CD Pipeline 설정
 - ✓ OpenID Connect를 이용한 Git Hub AWS 로그인
 - IAM 에서 자격 증명 공급자를 클릭해서 추가

공급자 구성

공급자 유형 **정보**

☐ SAML

AWS 계정과 Shibboleth 또는 Active Directory Federation Services와 같은 SAML 2.0 호환 자격 증명 공급자 간에 신뢰를 설정합니다.

☒ OpenID Connect

AWS 계정과 Google 또는 Salesforce와 같은 자격 증명 공급자 서비스 간에 신뢰를 설정합니다.

공급자 URL

인증 요청에 대한 보안 OpenID Connect URL을 지정합니다.

URL 편집

최대 255자입니다. URL은 "https"로 시작해야 합니다.

⚠ AWS는 인증서 지문을 사용하여 idP의 서버 인증서를 확인하지 않고 신뢰할 수 있는 CA 라이브러리를 사용하여 이 OIDC ID 제 공업체(idP)와의 통신을 보호합니다. 레거시 지문은 구성에 남아 있지만 검증에 더 이상 필요하지 않습니다.

지문	발행자	CA 유효성
1b511abead59c6ce207077c0bf0e0043b1382612	DigiCert Inc	03/30/2021~03/30/2031

대상 **정보**

앱의 자격 증명 공급자가 발행한 클라이언트 ID를 지정합니다.

최대 255자입니다. 영숫자 또는 ~, / 문자를 사용하세요.

AWS Container Service

- ❖ Django Application CI/CD Pipeline 설정
 - ✓ OpenID Connect를 이용한 Git Hub AWS 로그인
 - 자격 증명 공급자에 새로운 역할 할당

The screenshot displays the AWS IAM console for the role `token.actions.githubusercontent.com`. The breadcrumb navigation shows `IAM > 자격 증명 공급자 > token.actions.githubusercontent.com`. The role name is `token.actions.githubusercontent.com` with a link to its details. Buttons for `역할 할당` (Attach) and `삭제` (Delete) are visible.

요약 (Summary)

공급자	공급자 유형	생성 시간	ARN
<code>token.actions.githubusercontent.com</code>	OpenID Connect	December 18, 2023, 10:17 (UTC+09:00)	<code>arnaws:iam::641022061021:oidc-provider/token.actions.githubusercontent.com</code>

대상 (1) (Groups (1))

클라우드 ID라고도 하는 대상은 OpenID Connect 공급자에 등록된 애플리케이션을 식별하는 값입니다.

대상
<code>sts.amazonaws.com</code>

지문 (1) (Policies (1))

서버 인증서 지문은 OpenID Connect 공급자가 키를 제공하는 도메인에서 사용하는 X.509 인증서의 16진수 인코딩 SHA-1 해시 값입니다.

최대 5개의 지문을 추가할 수 있습니다. 이렇게 하면 자격 증명 공급자가 인증서를 교체하는 경우 여러 지문을 유지할 수 있습니다.

경고: AWS는 인증서 지문을 사용하여 idP의 서버 인증서를 확인하지 않고 신뢰할 수 있는 CA 라이브러리를 사용하여 이 OIDC ID 제공업체(idP)와의 통신을 보호합니다. 레거시 지문은 구성에 남아 있지만 검증에 더 이상 필요하지 않습니다.

ARN: `arnaws:iam::641022061021:oidc-provider/token.actions.githubusercontent.com`

AWS Container Service

- ❖ Django Application CI/CD Pipeline 설정
 - ✓ OpenID Connect를 이용한 Git Hub AWS 로그인
 - 자격 증명 공급자에 새로운 역할 할당

신뢰할 수 있는 엔터티 선택 정보

신뢰할 수 있는 엔터티 유형

☐ AWS 서비스

EC2, Lambda 등의 AWS 서비스가 이 계정에서 작업을 수행하도록 허용합니다.

☐ AWS 계정

사용자 또는 서드 파티에 속한 다른 AWS 계정의 엔터티가 이 계정에서 작업을 수행하도록 허용합니다.

☒ 웹 자격 증명

지정된 외부 웹 자격 증명 공급자와 연동된 사용자가 이 역할을 맡아 이 계정에서 작업을 수행하도록 허용합니다.

☐ SAML 2.0 연동

기업 디렉터리에서 SAML 2.0과 연동된 사용자가 이 계정에서 작업을 수행할 수 있도록 허용합니다.

☐ 사용자 지정 신뢰 정책

다른 사용자가 이 계정에서 작업을 수행할 수 있도록 사용자 지정 신뢰 정책을 생성합니다.

AWS Container Service

- ❖ Django Application CI/CD Pipeline 설정
 - ✓ OpenID Connect를 이용한 Git Hub AWS 로그인
 - 자격 증명 공급자에 새로운 역할 할당

웹 자격 증명
지정된 외부 웹 자격 증명 공급자와 연동된 사용자가 이 역할에 맡겨 이 계정에서 작업을 수행하도록 허용합니다.

자격 증명 공급자
  [새로 생성](#)

Audience

GitHub 조직

최대 39입니다.

GitHub 리포지토리 - 선택 사항

최대 100입니다.

GitHub 브랜치 - 선택 사항

최대 255입니다.

AWS Container Service

- ❖ Django Application CI/CD Pipeline 설정
 - ✓ OpenID Connect를 이용한 Git Hub AWS 로그인
 - Role에 이전에 만든 Policy를 할당

권한 추가 정보

권한 정책 (885) 정보 

새 역할에 연결할 정책을 하나 이상 선택합니다.

필터링 기준 유형

🔍 djan X 모든 유형 ▼ 1 개 일치 < 1 > ⚙️

☐	정책 이름 	▲	유형 ▼	설명
☐	 privateDjangoECR		고객 관리형	-

▶ 권한 경계 설정 - 선택 사항

취소

AWS Container Service

- ❖ Django Application CI/CD Pipeline 설정
 - ✓ OpenID Connect를 이용한 Git Hub AWS 로그인
 - Role에 이전에 만든 Policy를 할당

이름 지정, 검토 및 생성

역할 세부 정보

역할 이름
이 역할을 식별하는 의미 있는 이름을 입력합니다.

apiserverrole

최대 64자입니다. 영숫자 및 '+', '@', '-' 문자를 사용하세요.

설명
이 역할에 대하여 간단한 설명을 추가합니다.

최대 1000자입니다. 영숫자 및 '+', '@', '-' 문자를 사용하세요.

AWS Container Service

❖ Django Application CI/CD Pipeline 설정

✓ OpenID Connect를 이용한 Git Hub AWS 로그인

- GitHub Action.yml 파일을 수정해서 역할을 가져올 수 있는지 확인

```
name: django
```

```
on:
```

```
  push:
```

```
    branches: [ "main" ]
```

```
  pull_request:
```

```
    branches: [ "main" ]
```

```
permissions:
```

```
  id-token: write # This is required for requesting the JWT
```

```
  contents: read # This is required for actions/checkout
```

AWS Container Service

❖ Django Application CI/CD Pipeline 설정

✓ OpenID Connect를 이용한 Git Hub AWS 로그인

- GitHub Action.yml 파일을 수정해서 역할을 가져올 수 있는지 확인

jobs:

build:

runs-on: ubuntu-latest

steps:

- name: Checkout

uses: actions/checkout@v4

- name: Set up Python 3.10

uses: actions/setup-python@v4

with:

python-version: '3.10'

- name: Install dependencies

run: |

python -m pip install --upgrade pip

pip install -r requirements.txt

- name: Build Docker Compose

run: docker-compose

AWS Container Service

- ❖ Django Application CI/CD Pipeline 설정
 - ✓ OpenID Connect를 이용한 Git Hub AWS 로그인
 - GitHub Action.yml 파일을 수정해서 역할을 가져올 수 있는지 확인
 - name: Configure AWS Credentials
 - uses: aws-actions/configure-aws-credentials@v4
 - with:
 - role-to-assume: [arn:aws:iam::641022061021:role/itstudyoidc](#)
 - role-session-name: sampleSessionName
 - aws-region: ap-northeast-2

AWS Container Service

❖ Django Application CI/CD Pipeline 설정

✓ Image Push

- GitHub Action.yml 파일을 수정

name: django

on:

push:

branches: ["main"]

permissions:

id-token: write

contents: read

env:

AWS_REGION: ap-northeast-2

ECR_REPOSITORY: itstudydjangoecs

AWS Container Service

❖ Django Application CI/CD Pipeline 설정

✓ Image Push

● GitHub Action.yml 파일을 수정

jobs:

build:

runs-on: ubuntu-latest

steps:

- name: Checkout

uses: actions/checkout@v4

- name: Set up Python 3.10

uses: actions/setup-python@v4

with:

python-version: '3.10'

- name: Install dependencies

run: |

python -m pip install --upgrade pip

pip install -r requirements.txt

AWS Container Service

❖ Django Application CI/CD Pipeline 설정

✓ Image Push

- GitHub Action.yml 파일을 수정

name: django

on:

push:

branches: ["main"]

permissions:

id-token: write

contents: read

env:

AWS_REGION: ap-northeast-2

ECR_REPOSITORY: itstudydjangoecs

AWS Container Service

❖ Django Application CI/CD Pipeline 설정

✓ Image Push

● GitHub Action.yml 파일을 수정

- name: Configure AWS Credentials

uses: aws-actions/configure-aws-credentials@v4

with:

role-to-assume: arn:aws:iam::641022061021:role/imagepush

role-session-name: sampleSessionName

aws-region: ap-northeast-2

- name: Login to Amazon ECR

id: login-ecr

uses: aws-actions/amazon-ecr-

login@62f4f872db3836360b72999f4b87f1ff13310f3a

AWS Container Service

❖ Django Application CI/CD Pipeline 설정

✓ Image Push

● GitHub Action.yml 파일을 수정

- name: Build, tag, and push image to Amazon ECR

id: build-image

env:

ECR_REGISTRY: \${ steps.login-ecr.outputs.registry }

IMAGE_TAG: \${ github.sha }

run: |

Build a docker container and

push it to ECR so that it can

be deployed to ECS.

docker build -t \$ECR_REGISTRY/\$ECR_REPOSITORY:\$IMAGE_TAG .

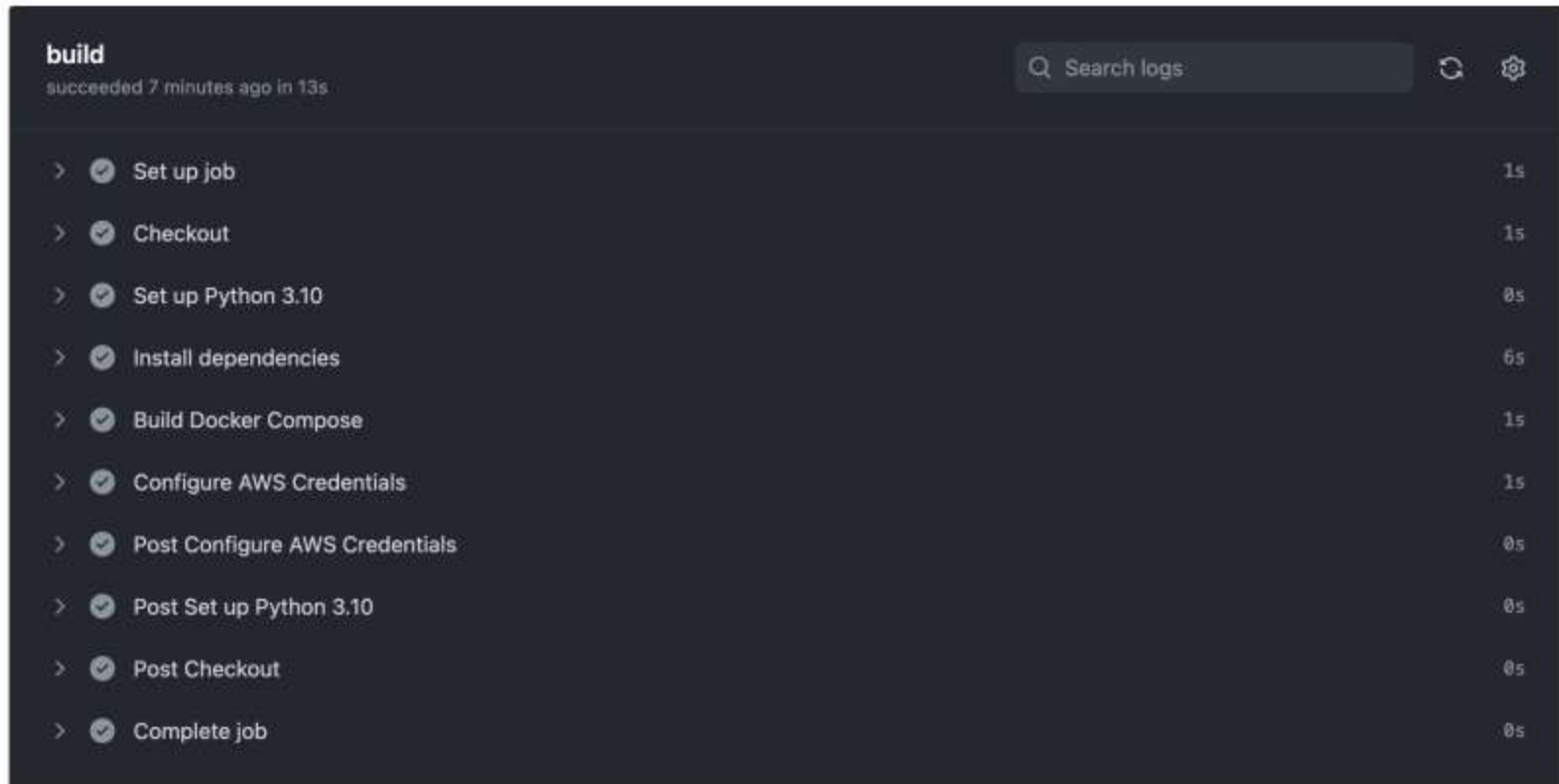
docker push \$ECR_REGISTRY/\$ECR_REPOSITORY:\$IMAGE_TAG

echo "image=\$ECR_REGISTRY/\$ECR_REPOSITORY:\$IMAGE_TAG" >>

\$GITHUB_OUTPUT

AWS Container Service

- ❖ Django Application CI/CD Pipeline 설정
 - ✓ OpenID Connect를 이용한 Git Hub AWS 로그인



AWS Container Service

- ❖ Django Application CI/CD Pipeline 설정
 - ✓ ECR Push And Task Definition 생성 및 Service Update
 - <https://docs.github.com/en/enterprise-cloud@latest/actions/deployment/deploying-to-your-cloud-provider/deploying-to-amazon-elastic-container-service> 참고
 - 사전 작업
 - ◆ ECR Repository 생성
 - ◆ ECS Cluster, Task, Service 생성
 - ◆ task-definition.json 파일 생성: ECS에서 생성한 task 의 JSON을 복사해서 프로젝트에 생성 – image 부분은 제거

AWS Container Service

- ❖ Django Application CI/CD Pipeline 설정
 - ✓ ECR Push And Task Definition 생성 및 Service Update
 - Git Action.yml 파일을 수정 – 환경 변수는 적절하게 수정

name: django

on:

push:

branches: ["main"]

pull_request:

branches: ["main"]

permissions:

id-token: write

contents: read

env:

AWS_REGION: ap-northeast-2

ECR_REPOSITORY: itstudydjangoecs

ECS_SERVICE: itstudyservice2

ECS_CLUSTER: deploytest

ECS_TASK_DEFINITION: ./task-definition.json

CONTAINER_NAME: itstudy1

AWS Container Service

- ❖ Django Application CI/CD Pipeline 설정
 - ✓ ECR Push And Task Definition 생성 및 Service Update
 - Git Action.yml 파일을 수정 – 환경 변수는 적절하게 수정

jobs:

build:

runs-on: ubuntu-latest

steps:

- name: Checkout
 - uses: actions/checkout@v4
- name: Set up Python 3.10
 - uses: actions/setup-python@v4
 - with:
 - python-version: '3.10'
- name: Install dependencies
 - run: |
 - python -m pip install --upgrade pip
 - pip install -r requirements.txt
- name: Build Docker Compose
 - run: docker-compose

AWS Container Service

❖ Django Application CI/CD Pipeline 설정

✓ ECR Push And Task Definition 생성 및 Service Update

● Git Action.yml 파일을 수정 – 환경 변수는 적절하게 수정

- name: Configure AWS Credentials

uses: aws-actions/configure-aws-credentials@v4

with:

role-to-assume: arn:aws:iam::641022061021:role/imagepush

role-session-name: sampleSessionName

aws-region: ap-northeast-2

- name: Login to Amazon ECR

id: login-ecr

uses: aws-actions/amazon-ecr-

login@62f4f872db3836360b72999f4b87f1ff13310f3a

AWS Container Service

❖ Django Application CI/CD Pipeline 설정

✓ ECR Push And Task Definition 생성 및 Service Update

● Git Action.yml 파일을 수정 – 환경 변수는 적절하게 수정

- name: Build, tag, and push image to Amazon ECR

id: build-image

env:

ECR_REGISTRY: \${ steps.login-ecr.outputs.registry }

IMAGE_TAG: \${ github.sha }

run: |

Build a docker container and

push it to ECR so that it can

be deployed to ECS.

docker build -t \$ECR_REGISTRY/\$ECR_REPOSITORY:\$IMAGE_TAG .

docker push \$ECR_REGISTRY/\$ECR_REPOSITORY:\$IMAGE_TAG

echo "image=\$ECR_REGISTRY/\$ECR_REPOSITORY:\$IMAGE_TAG" >>

\$GITHUB_OUTPUT

AWS Container Service

- ❖ Django Application CI/CD Pipeline 설정
 - ✓ ECR Push And Task Definition 생성 및 Service Update
 - Git Action.yml 파일을 수정 – 환경 변수는 적절하게 수정
 - name: Fill in the new image ID in the Amazon ECS task definition
 - id: task-def
 - uses: aws-actions/amazon-ecs-render-task-definition@v2
 - with:
 - task-definition: \${{ env.ECS_TASK_DEFINITION }}
 - container-name: \${{ env.CONTAINER_NAME }}
 - image: \${{ steps.build-image.outputs.image }}
 - name: Deploy Amazon ECS task definition
 - uses: aws-actions/amazon-ecs-deploy-task-definition@df9643053eda01f169e64a0e60233aacca83799a
 - with:
 - task-definition: \${{ steps.task-def.outputs.task-definition }}
 - service: \${{ env.ECS_SERVICE }}
 - cluster: \${{ env.ECS_CLUSTER }}
 - wait-for-service-stability: true

AWS Container Service

- ❖ Django Application CI/CD Pipeline 설정
 - ✓ ECR Push And Task Definition 생성 및 Service Update
 - Git Action.yml 파일을 수정 – 환경 변수는 적절하게 수정

```
> [x] Build Docker Compose
> [x] Configure AWS Credentials
v [x] Login to Amazon ECR

1 ▶ Run aws-actions/amazon-ecr-login@62f4f872db3836360b72999f4b87f1ff13310f3a
22 (node:1816) NOTE: The AWS SDK for JavaScript (v2) will be put into maintenance mode in 2023.
23
24 Please migrate your code to use AWS SDK for JavaScript (v3).
25 For more information, check the migration guide at https://a.co/7PzMCcy
26 (Use 'node --trace-warnings ...' to show where the warning was created)
27 Error: User: arn:aws:sts::641022061021:assumed-role/itstudyoicd/sampleSessionName is not authorized to perform:
    ecr:GetAuthorizationToken on resource: * because no identity-based policy allows the ecr:GetAuthorizationToken action

  Build, tag, and push image to Amazon ECR
  Fill in the new image ID in the Amazon ECS task definition
  Deploy Amazon ECS task definition
```

AWS Container Service

- ❖ Django Application CI/CD Pipeline 설정
 - ✓ ECR Push And Task Definition 생성 및 Service Update
 - 권한 부여 – IAM에서 이전에 만든 Role에 ECS 관련 권한 추가 후 다시 빌드

권한 정책 (3) 정보 시뮬레이션 제거 권한 추가 ▼

최대 10개의 관리형 정책을 연결할 수 있습니다.

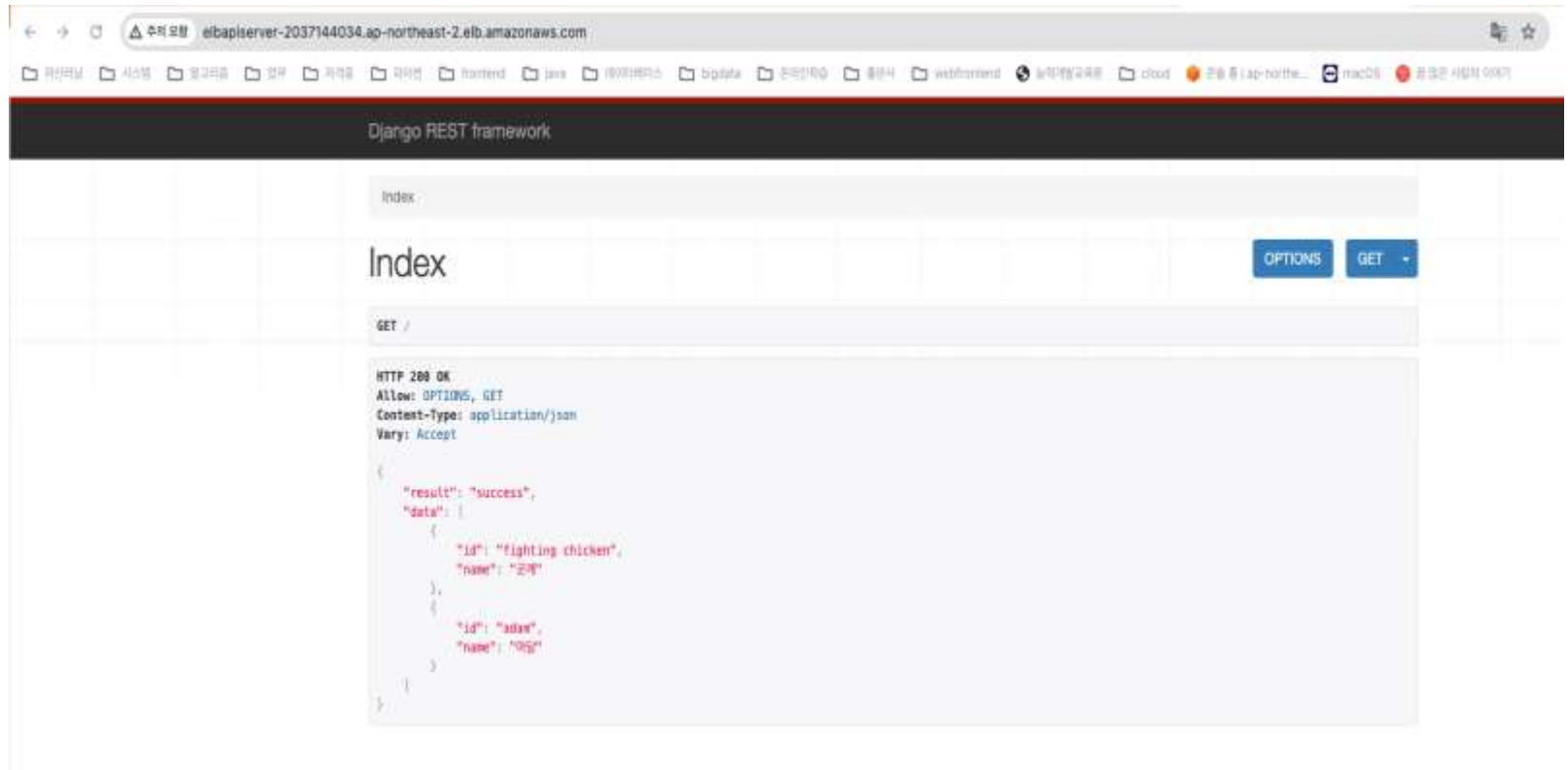
필터링 기준 유형

검색 모든 유형 ▼ < 1 > ⚙

☐	정책 이름 ↗	유형 ↗	연결된 엔터티 ↗
☐	AmazonECS_FullAccess	AWS 관리형	1
☐	AmazonECSTaskExecutionRole...	AWS 관리형	2
☐	privateDjangoECR	고객 관리형	2

AWS Container Service

- ❖ Django Application CI/CD Pipeline 설정
 - ✓ 내용을 수정하고 푸시

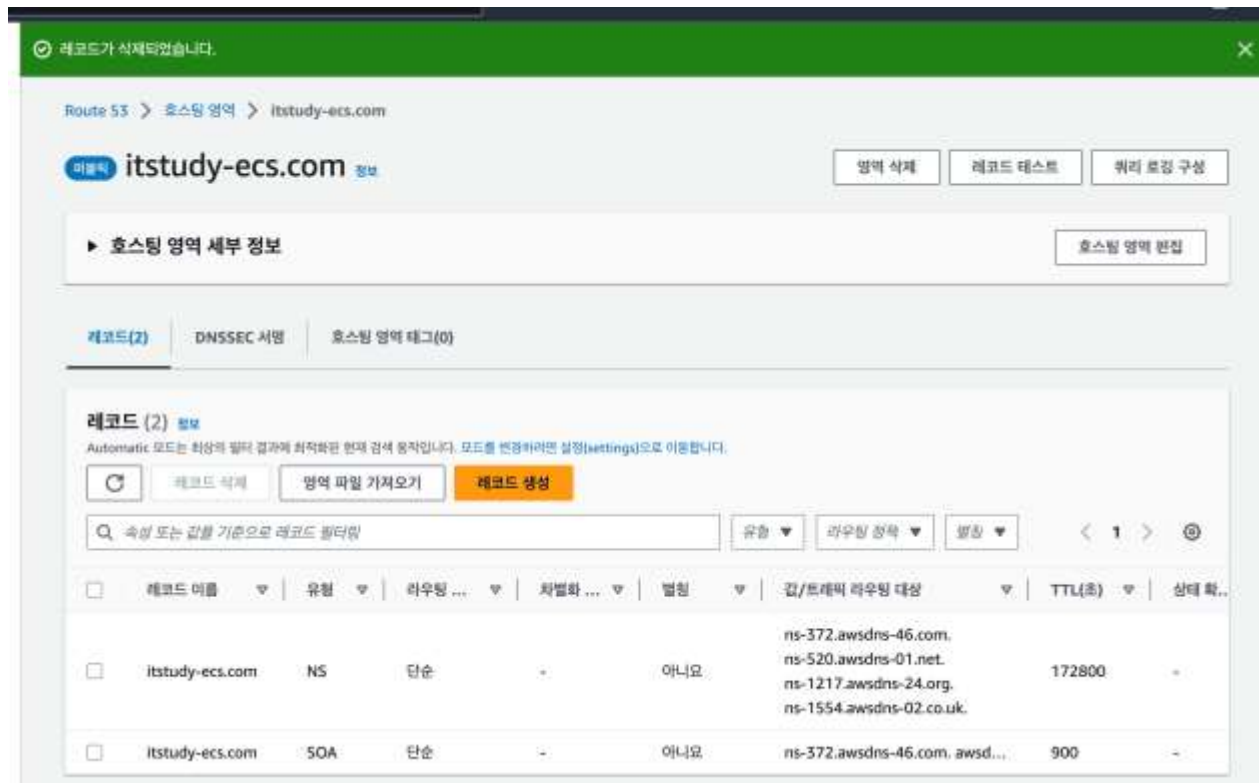


AWS Container Service

❖ 도메인 연결

✓ Domain 연결

- 호스팅 영역에서 호스팅 영역 선택



AWS Container Service

❖ 도메인 연결

✓ Domain 연결

- 레코드 생성 버튼을 누르고 정보 입력

레코드 생성 정보

빠른 레코드 생성 마법사로 전환

▼ 레코드 1 삭제

레코드 이름 정보 레코드 유형 정보

루트 도메인에 대한 레코드를 생성하려면 비워 두세요.

☒ 별칭

트래픽 라우팅 대상 정보

별칭 호스팅 영역 ID: ZWKZPGT148KDX

라우팅 정책 정보 대상 상태 평가 ☒ 예

다른 레코드 추가

취소 레코드 생성

AWS Container Service

❖ 도메인 연결

✓ Domain 연결

- 레코드 생성 버튼을 누르고 정보 입력

레코드 (3) 정보
Automatic 모드는 최상의 필터 결과에 최적화된 현재 검색 동작입니다. 모드를 변경하려면 설정(settings)으로 이동합니다.

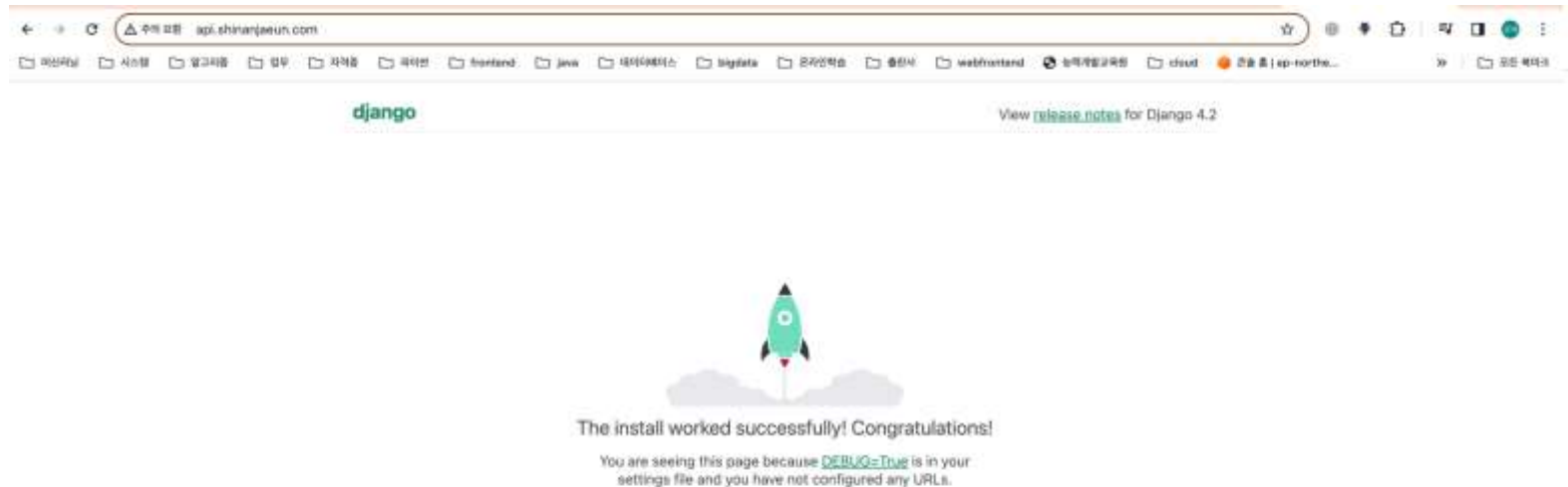
Q 속성 또는 값을 기준으로 레코드 필터링

유형 ▼ 라우팅 정책 ▼ 별칭 ▼ < 1 > ⓘ

☐	레코드 이름 ▼	유형 ▼	라우팅 ... ▼	차별화 ... ▼	별칭 ▼	값/트래픽 라우팅 대상 ▼	TTL(초) ▼	상태 확..
☐	itstudy-ecs.com	NS	단순	-	아니오	ns-372.awsdns-46.com, ns-520.awsdns-01.net, ns-1217.awsdns-24.org, ns-1554.awsdns-02.co.uk.	172800	-
☐	itstudy-ecs.com	SOA	단순	-	아니오	ns-372.awsdns-46.com, awsd...	900	-
☐	api.itstudy-ec...	A	단순	-	아니오	3.39.118.87	300	-

AWS Container Service

- ❖ 도메인 연결
 - ✓ 브라우저에서 확인



AWS Container Service

- ❖ HTTPS 적용
 - ✓ HTTPS 인증서 발급
 - AWS Certificate Manager 서비스에 접속



AWS Container Service

❖ HTTPS 적용

✓ HTTPS 인증서 발급

● 인증서 요청 클릭



AWS Container Service

- ❖ HTTPS 적용
 - ✓ HTTPS 인증서 발급
 - 인증서 요청을 위한 도메인 설정

퍼블릭 인증서 요청

도메인 이름

인증서에 대해 하나 이상의 도메인 이름을 제공합니다.

원전히 정규화된 도메인 이름 [정보](#)

api.shinajaeun.com

이 인증서에 다른 이름 추가

이 인증서에 이름을 추가할 수 있습니다. 예를 들어, 'www.example.com'에 대한 인증서를 요청하는 경우 고객어 두 이름 중 하나로 시어브에 접속할 수 있도록 'example.com'이라는 이름을 추가할 수 있습니다.

검증 방법 [정보](#)

도메인 소유권을 검증하기 위한 방법 선택

☒ DNS 검증 – 권장

인증서 요청에서 도메인에 대한 DNS 구성을 수정할 권한이 있는 경우 이 옵션을 선택합니다.

☐ 이메일 검증

인증서 요청에서 도메인에 대한 DNS 구성을 수정할 권한을 소유하지 않거나 획득할 수 없는 경우 이 옵션을 선택합니다.

AWS Container Service

❖ HTTPS 적용

✓ HTTPS 인증서 발급

- 인증서를 클릭하고 Route53에서 레코드 생성 클릭

8d320d9c-9ad2-4a64-a24b-08145e37e06f 삭제

인증서 상태

식별자: 8d320d9c-9ad2-4a64-a24b-08145e37e06f 상태: [검증 대기 중](#) [정보](#)

ARN: `arn:aws:acm:ap-northeast-2:641022061021:certificate/8d320d9c-9ad2-4a64-a24b-08145e37e06f`

유형: Amazon 발급

도메인 (1) Route 53에서 레코드 생성 CSV로 내보내기

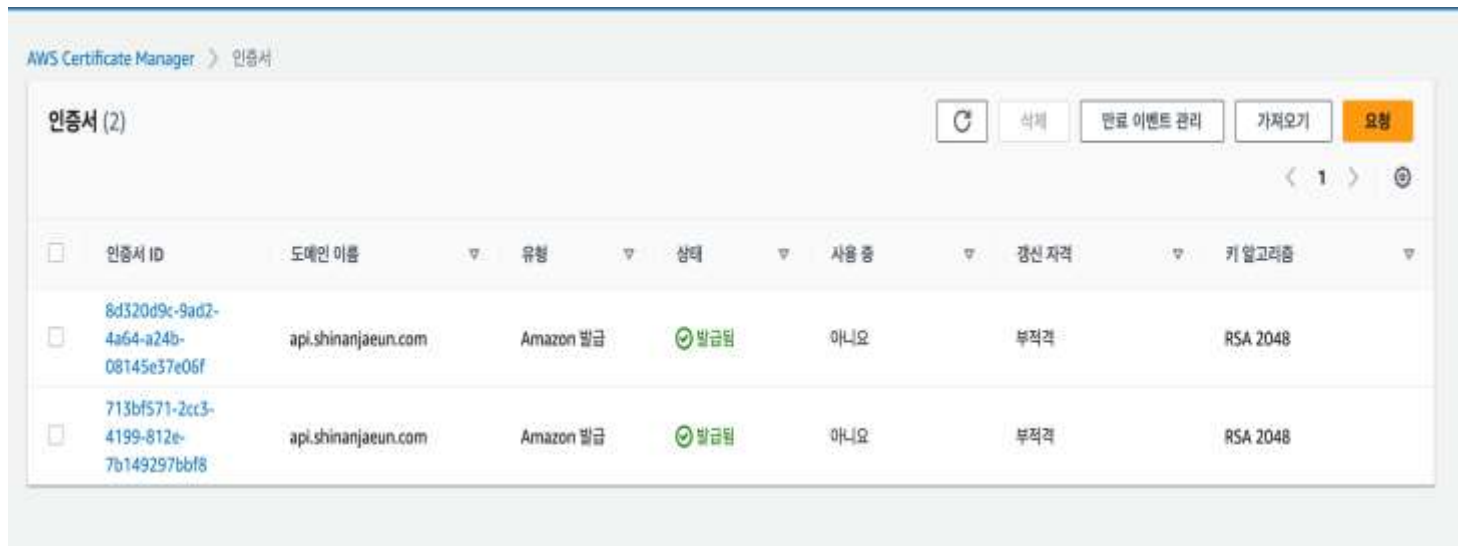
도메인	상태	경신 상태	유형	CNAME 이름	CNAME 값
api.shinanjaeun.com	검증 대기 중	-	CNAME	<code>_ad7d46b52cfa91db99d656e0b6ae318c.api.shinanjaeun.com.</code>	<code>_ab922b26f78c9d6240ff9b479d190908.mhbtspdn.acm-validations.aws.</code>

AWS Container Service

❖ HTTPS 적용

✓ HTTPS 인증서 발급

- 인증서를 클릭하고 Route53에서 레코드 생성 클릭

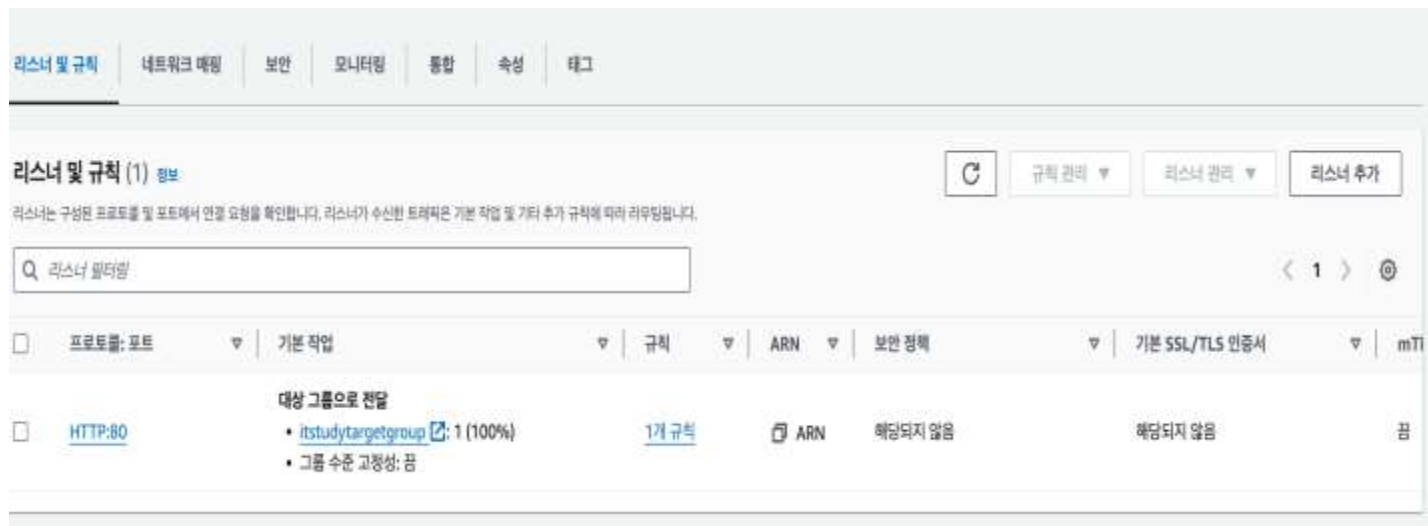


AWS Container Service

❖ HTTPS 적용

✓ ELB에 인증서 등록

- 로드 밸런서에 접속해서 [리스너 추가] 버튼 클릭



AWS Container Service

❖ HTTPS 적용

✓ ELB에 인증서 등록

● 로드 밸런서 설정 – 프로토콜을 HTTPS로 변경

리스너 세부 정보: HTTPS:443

리스너는 사용자가 구성된 프로토콜 및 포트를 사용하여 연결 요청을 확인합니다. 생성한 기본 작업 및 추가 규칙에 따라 Application Load Balancer가 요청을 등록된 대상으로 라우팅하는 방법이 결정됩니다.

리스너 구성

리스너는 프로토콜 및 포트 식별됩니다.

프로토콜	포트
클라이언트에서 로드 밸런서로 연결하기 위해 사용됩니다.	로드 밸런서가 연결을 위해 수신 대기하는 포트입니다.
HTTPS ▼	443 ↕ 1-65535

기본 작업 [정보](#)

다른 규칙이 적용되지 않는 경우 기본 작업이 사용됩니다. 이 리스너의 트래픽에 대해 기본 작업을 선택하세요.

인증 [정보](#)

☐ OpenID 또는 Amazon Cognito 사용

OpenID Connect(OIDC) 또는 Amazon Cognito를 사용한 인증을 포함합니다.

라우팅 액션

☒ 대상 그룹으로 전달

☐ URL로 리디렉션

☐ 고정 응답 반환

AWS Container Service

❖ HTTPS 적용

✓ ELB에 인증서 등록

● 로드 밸런서 설정 - 대상 그룹 선택

기본 작업 [정보](#)

다른 규칙이 적용되지 않는 경우 기본 작업이 사용됩니다. 이 리스너의 트래픽에 대해 기본 작업을 선택하세요.

인증 [정보](#)

☐ OpenID 또는 Amazon Cognito 사용

OpenID Connect(OIDC) 또는 Amazon Cognito를 사용한 인증을 포함합니다.

라우팅 액션

☒ 대상 그룹으로 전달

☐ URL로 리디렉션

☐ 고정 응답 반환

대상 그룹으로 전달 [정보](#)

대상 그룹을 선택하고 라우팅 가중치를 지정하거나 [대상 그룹을 생성](#)합니다.

대상 그룹

itstudytargetgroup

대상 유형: 인스턴스, IPv4

HTTP ▼



가중치

1

0-999

백분율

100%

대상 그룹 추가

최대 4개의 대상 그룹을 더 추가할 수 있습니다.

그룹 수준 고정성 [정보](#)

대상 그룹에 고정성이 설정되어 있는 경우, 해당 그룹에 라우팅된 요청은 세션 기간 동안 대상 그룹에 유지됩니다. 개별 대상 고정성은 대상 그룹의 구성 옵션입니다.

☐ 그룹 수준 고정 활성화

AWS Container Service

❖ HTTPS 적용

✓ ELB에 인증서 등록

● 로드 밸런서 설정 - 인증서 선택

보안 리스너 설정 정보

보안 정책 정보
로드 밸런서는 보안 정책이라고 하는 Secure Socket Layer(SSL) 협상 구성을 사용해 클라이언트와의 SSL 연결을 관리합니다. [보안 정책 비교](#)

보안 카테고리
모든 보안 정책 ▼

정책 이름
ELBSecurityPolicy-TLS13-1-2-2021-06 (권장) ▼

기본 SSL/TLS 서버 인증서
클라이언트가 SNI 프로토콜 없이 연결하거나 일치하는 인증서가 없는 경우에 사용되는 인증서입니다. 이 인증서를 AWS Certificate Manager(ACM), Amazon Identity and Access Management(IAM)에서 소싱하거나 인증서 가져오기를 실시할 수 있습니다. 이 인증서는 리스너 인증서 목록에 자동으로 추가됩니다.

인증서 소스

☒ ACM에서

☐ IAM에서

☐ 인증서 가져오기

인증서(ACM에서)
선택한 인증서는 이 로드 밸런서의 보안 리스너에 대한 기본 SSL/TLS 서버 인증서로 적용됩니다.

api.shinajaeun.com
8d320d9c-9ad2-4a64-a24b-0814...

↻

[새 ACM 인증서 요청](#)

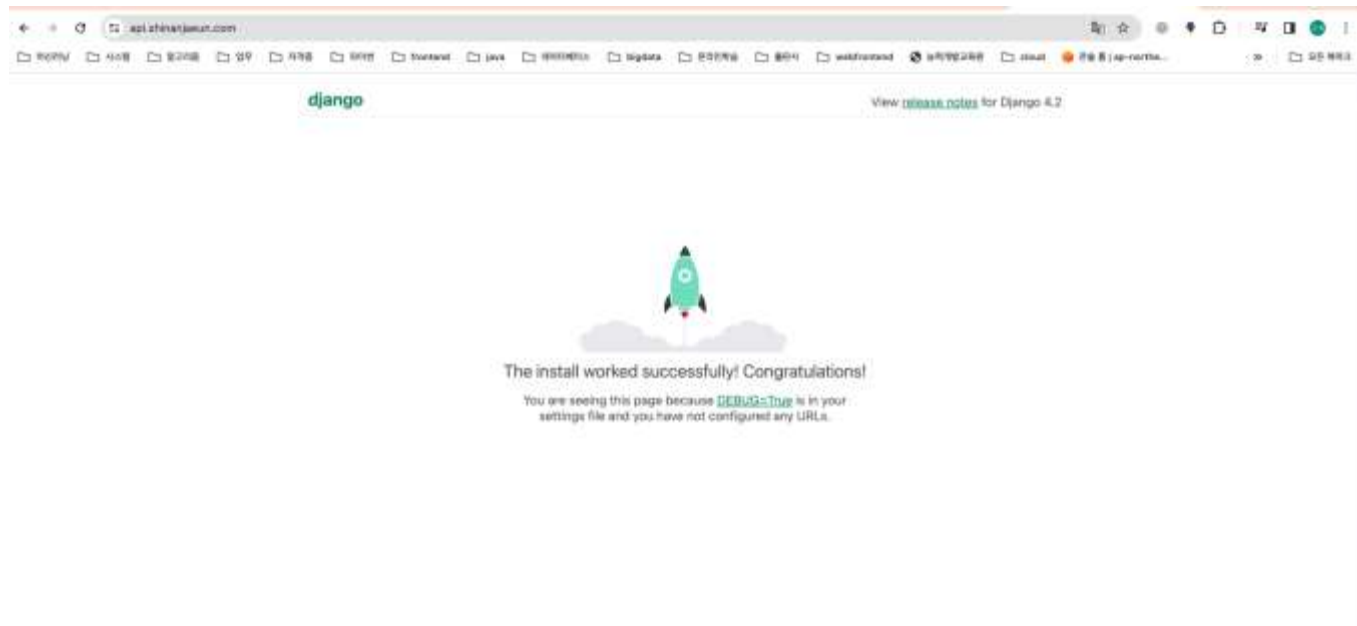
클라이언트 인증서 처리 정보
클라이언트 인증서는 인증된 요청을 원격 서버로 보내는 데 사용됩니다. [자세히 알아보기](#)

☐ 상호 인증(mTLS)
상호 전송 계층 보안(TLS) 인증은 양방향 피어 인증을 제공합니다. TLS를 통한 보안 계층을 추가하고 서비스에서 연결을 설정하는 클라이언트를 확인할 수 있도록 합니다.

AWS Container Service

❖ HTTPS 적용

- ✓ ELB에 인증서 등록
 - 확인 - 브라우저에서 https 로 접속



AWS Container Service

❖ Spring 애플리케이션을 Docker Image를 이용한 EC2 배포

✓ Spring Boot Application 작성

- Spring Boot Application 생성
- 프로젝트에 Controller 추가 - FrontController

```
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
```

```
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;
```

```
@RestController
```

```
public class FrontController {
```

```
    @GetMapping("/")
```

```
    public Map<String, Object> uploadFile() {
```

```
        Map<String, Object> data = new HashMap<String, Object>();
        data.put("result", "success");
```

AWS Container Service

❖ Spring 애플리케이션을 Docker Image를 이용한 EC2 배포

✓ Spring Boot Application 작성

● 프로젝트에 Controller 추가 - FrontController

```
@SuppressWarnings("rawtypes")
List <Map> list = new ArrayList<Map>();

Map<String, String> map1 = new HashMap<>();
map1.put("id", "itstudy");
map1.put("name", "군계");

Map<String, String> map2 = new HashMap<>();
map2.put("id", "itggangpae");
map2.put("name", "아담");

list.add(map1);
list.add(map2);

data.put("data", list);

return data;
}
```

AWS Container Service

- ❖ Spring 애플리케이션을 Docker Image을 이용한 EC2 배포
 - ✓ Spring Boot Application 작성
 - application.yml 파일에 서버 포트 설정

```
server:  
  port: 80
```

AWS Container Service

- ❖ Spring 애플리케이션을 Docker Image를 이용한 EC2 배포
 - ✓ Spring Boot Application 실행 확인



AWS Container Service

- ❖ Spring 애플리케이션을 Docker Image를 이용한 EC2 배포
 - ✓ 이미지 생성 및 확인
 - Spring Boot Application 의 루트 디렉토리에 Dockerfile을 작성
 - ◆ jar 로 생성

```
FROM amazoncorretto:17
CMD ["/mvnw", "clean", "package"]
ARG JAR_FILE=target/*.jar
COPY ./build/libs/*.jar app.jar
ENTRYPOINT ["java", "-jar", "app.jar"]
```
 - ◆ war 로 생성

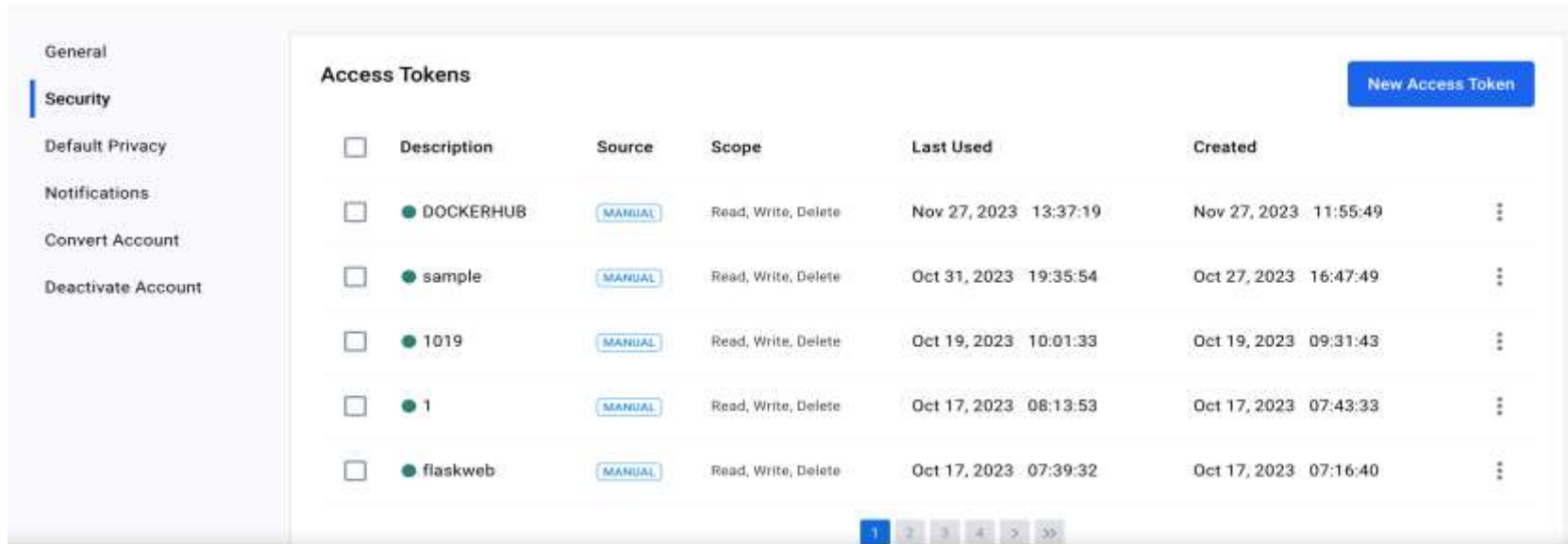
```
FROM amazoncorretto:17
CMD ["/mvnw", "clean", "package"]
ARG WAR_FILE=target/*.war
COPY ./build/libs/*.war app.war
ENTRYPOINT ["java", "-jar", "app.war"]
```

AWS Container Service

- ❖ Spring 애플리케이션을 Docker Image를 이용한 EC2 배포
 - ✓ 이미지 생성 및 확인
 - 터미널에서 build
`./gradlew clean build`
 - 이미지 빌드 및 생성
`docker build -f Dockerfile -t springapiserver:0.0.1 .`
 - 이미지 확인
`docker images`
 - 컨테이너 생성 및 실행
`docker run -d --name springapiserver -p 80:80 springapiserver:0.0.1`

AWS Container Service

- ❖ Spring 애플리케이션을 Docker Image를 이용한 EC2 배포
 - ✓ Git Action을 활용한 Docker Hub에 이미지 배포
 - Docker Hub에 Repository 생성
 - Docker Hub에서 Access Token 생성 – 오른쪽 상단의 아이디 옆을 클릭한 후 [Account Settings] – [Security] – [Access Token] – [New Access Token]



The screenshot shows the Docker Hub 'Security' settings page. On the left is a sidebar with navigation links: General, Security (selected), Default Privacy, Notifications, Convert Account, and Deactivate Account. The main content area is titled 'Access Tokens' and features a 'New Access Token' button in the top right corner. Below the title is a table listing existing access tokens. Each row includes a checkbox, a description with a green dot icon, a source (all are 'MANUAL'), a scope ('Read, Write, Delete'), a 'Last Used' timestamp, a 'Created' timestamp, and a three-dot menu icon.

☐	Description	Source	Scope	Last Used	Created	
☐	● DOCKERHUB	MANUAL	Read, Write, Delete	Nov 27, 2023 13:37:19	Nov 27, 2023 11:55:49	⋮
☐	● sample	MANUAL	Read, Write, Delete	Oct 31, 2023 19:35:54	Oct 27, 2023 16:47:49	⋮
☐	● 1019	MANUAL	Read, Write, Delete	Oct 19, 2023 10:01:33	Oct 19, 2023 09:31:43	⋮
☐	● 1	MANUAL	Read, Write, Delete	Oct 17, 2023 08:13:53	Oct 17, 2023 07:43:33	⋮
☐	● flaskweb	MANUAL	Read, Write, Delete	Oct 17, 2023 07:39:32	Oct 17, 2023 07:16:40	⋮

At the bottom of the table, there is a pagination bar showing '1' as the current page, followed by '2', '3', '4', and navigation arrows '>' and '>>'.

AWS Container Service

- ❖ Spring 애플리케이션을 Docker Image를 이용한 EC2 배포
 - ✓ Git Action을 활용한 Docker Hub에 이미지 배포
 - Git Hub에 자바 코드 업로드
 - GIT HUB의 Repository에 Secret Key 등록: [Settings] – [Secrets and variable] – [Actions] – [New repository secret]
 - ◆ DOCKERHUB_USERNAME
 - ◆ DOCKERHUB_TOKEN



AWS Container Service

❖ Spring 애플리케이션을 Docker Image를 이용한 EC2 배포

✓ Git Action을 활용한 Docker Hub에 이미지 배포

- .github/workflows/ 디렉토리에 yaml 파일을 생성하고 push
name: Java CI with Gradle

동작 조건 설정 : main 브랜치에 push 혹은 pull request가 발생할 경우 동작한다.
on:

```
push:  
  branches: [ "main" ]  
pull_request:  
  branches: [ "main" ]
```

```
permissions:  
  contents: read
```

AWS Container Service

- ❖ Spring 애플리케이션을 Docker Image를 이용한 EC2 배포
 - ✓ Git Action을 활용한 Docker Hub에 이미지 배포
 - .github/workflows/ 디렉토리에 yaml 파일을 생성하고 push
- ```
jobs:
 # Spring Boot 애플리케이션을 빌드하여 도커 허브에 푸시하는 과정
 build-docker-image:
 runs-on: ubuntu-latest
 steps:
 - uses: actions/checkout@v3
 # 1. Java 17 세팅
 - name: Set up JDK 17
 uses: actions/setup-java@v3
 with:
 java-version: '17'
 distribution: 'temurin'

 # 2. Spring Boot 애플리케이션 빌드
 - name: Build with Gradle
 uses: gradle/gradle-build-
action@67421db6bd0bf253fb4bd25b31ebb98943c375e1
 with:
 arguments: clean bootJar
```

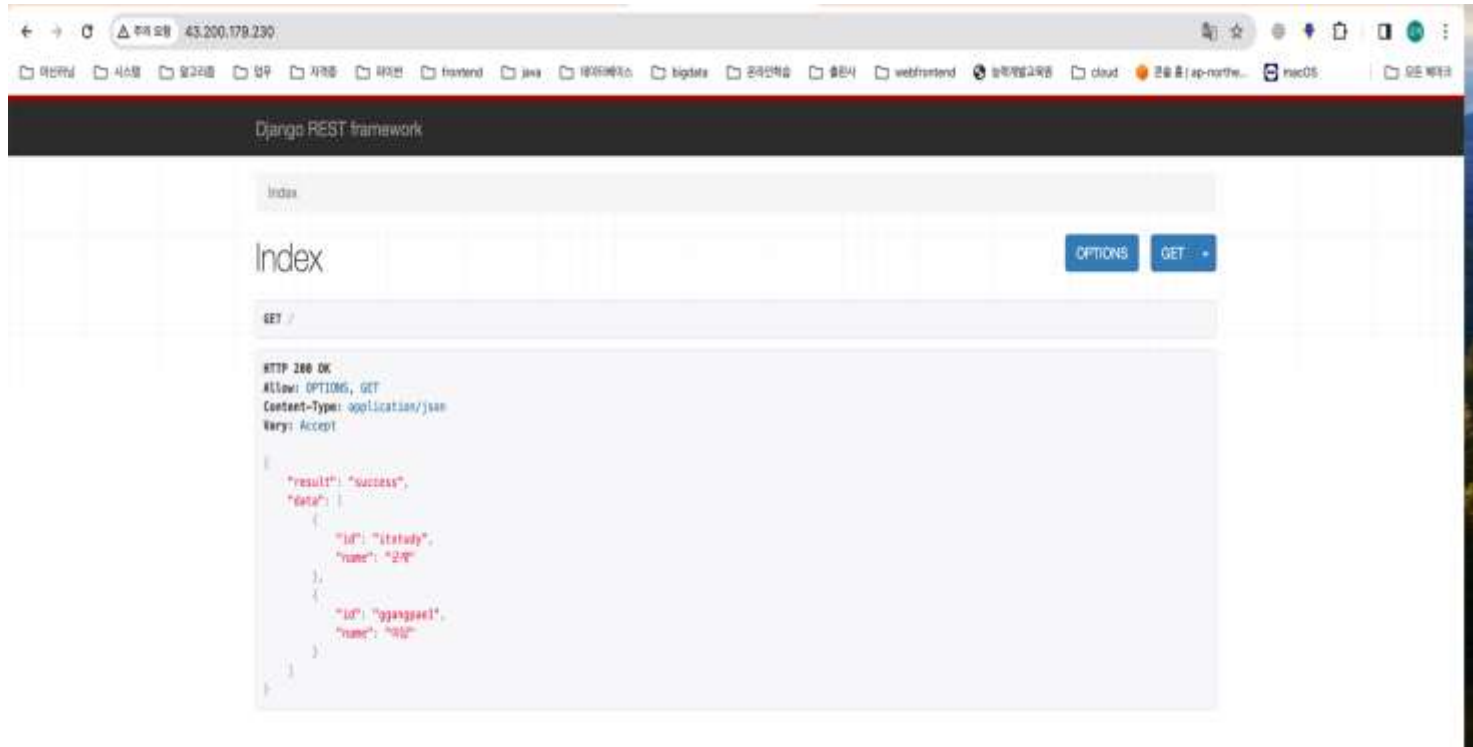
# AWS Container Service

- ❖ Spring 애플리케이션을 Docker Image를 이용한 EC2 배포
  - ✓ Git Action을 활용한 Docker Hub에 이미지 배포
    - .github/workflows/ 디렉토리에 yaml 파일을 생성하고 push

```
3. DockerHub 로그인
- name: Login to DockerHub
 uses: docker/login-action@v1
 with:
 username: ${secrets.DOCKERHUB_USERNAME}
 password: ${secrets.DOCKERHUB_TOKEN}
4. 이미지 업로드
- name: build and release to DockerHub
 env:
 NAME: ${secrets.DOCKERHUB_USERNAME}
 REPO: springapiserver
 run: |
 docker build -t $REPO .
 docker tag $REPO:latest $NAME/$REPO:latest
 docker push $NAME/$REPO:latest
```

# AWS Container Service

- ❖ Spring 애플리케이션을 Docker Image를 이용한 EC2 배포
  - ✓ 이미지 다운로드 및 실행
    - `sudo docker run -p 80:80 -d --rm ggangpae1/springapiserver`
  - ✓ 브라우저에서 퍼블릭 IP로 확인



# AWS Container Service

## ❖ Spring Boot Application CI/CD Pipeline 설정

### ✓ Docker-Compose 사용

- 프로젝트에 docker-compose.yml 파일을 생성하고 작성

```
version: "3"
```

```
services:
```

```
 web:
```

```
 build:
```

```
 context: .
```

```
 dockerfile: Dockerfile
```

```
 command: "java -jar app.jar"
```

```
 ports:
```

```
 - "80:80"
```

# AWS Container Service

- ❖ Spring Boot Application CI/CD Pipeline 설정

- ✓ Docker-Compose 사용

- 콘솔 명령으로 확인

- ```
docker-compose -f docker-compose.yml up --build
```

AWS Container Service

❖ Spring Boot Application CI/CD Pipeline 설정

✓ Docker-Compose 사용

- Git Hub Action을 위한 yml 파일을 수정해서 테스트

name: Java CI with Gradle

동작 조건 설정 : main 브랜치에 push 혹은 pull request가 발생할 경우 동작한다.
on:

push:

branches: ["main"]

pull_request:

branches: ["main"]

permissions:

contents: read

AWS Container Service

❖ Spring Boot Application CI/CD Pipeline 설정

✓ Docker-Compose 사용

- Git Hub Action을 위한 yml 파일을 수정해서 테스트

jobs:

Spring Boot 애플리케이션을 빌드하여 도커 허브에 푸시하는 과정

build-docker-image:

runs-on: ubuntu-latest

steps:

- uses: actions/checkout@v3

1. Java 17 세팅

- name: Set up JDK 17

uses: actions/setup-java@v3

with:

java-version: '17'

distribution: 'temurin'

2. Spring Boot 애플리케이션 빌드

- name: Build with Gradle

uses: gradle/gradle-build-

action@67421db6bd0bf253fb4bd25b31ebb98943c375e1

with:

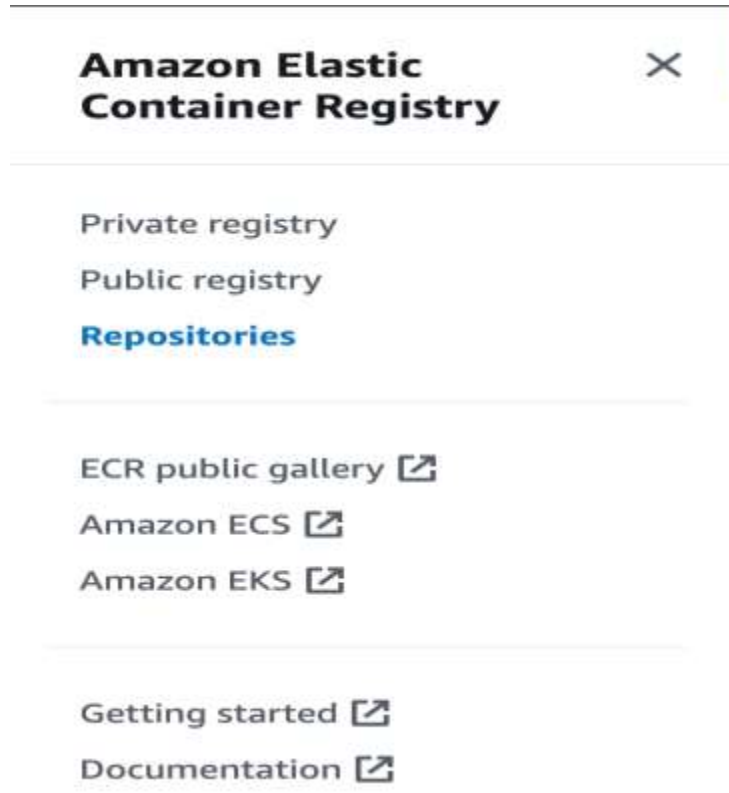
arguments: clean bootJar

AWS Container Service

- ❖ Spring Boot Application CI/CD Pipeline 설정
 - ✓ Docker-Compose 사용
 - Git Hub Action을 위한 yml 파일을 수정해서 테스트
 - name: Build Docker Compose
 - run: docker-compose

AWS Container Service

- ❖ Spring Boot Application CI/CD Pipeline 설정
 - ✓ ECR(Elastic Container Registry) 서비스에 접속해서 Repository 생성 - springapiserver



AWS Container Service

- ❖ Spring Boot Application CI/CD Pipeline 설정
 - ✓ ECR(Elastic Container Registry) 서비스에 접속해서 Repository 생성 - springapiserver

Amazon ECR > 프라이빗 레지스트리 > 리포지토리 > 리포지토리 생성

리포지토리 생성

일반 설정

표시 여부 설정 [정보](#)
리포지토리에 대한 가시성 설정을 선택합니다.

☒ **프라이빗**
액세스는 IAM 및 리포지토리 정책 권한에 의해 관리됩니다.

☐ **퍼블릭**
이미지 물에 대해 공개적으로 표시되고 액세스할 수 있습니다.

리포지토리 이름
간결한 이름을 제공합니다. 개발자는 이름으로 리포지토리 콘텐츠를 식별할 수 있어야 합니다.

641022061021.dkr.ecr.ap-northeast-2.amazonaws.com/

최대 256자 중 9자(최소 2자 이상). The name must start with a letter and can only contain lowercase letters, numbers, hyphens, underscores, periods and forward slashes.

태그 변경 불가능 [정보](#)
동일한 태그를 사용하는 후속 이미지 푸시가 이미지 태그를 덮어쓰지 않도록 방지하려면 [태그 변경 불가능]을 활성화합니다. 이미지 태그를 덮어쓰려면 [태그 변경 불가능]을 비활성화합니다.

☒ **비활성화됨**

[?](#) 리포지토리가 생성되면 해당 리포지토리의 가시성 설정을 변경할 수 없습니다.

AWS Container Service

- ❖ Spring Boot Application CI/CD Pipeline 설정
 - ✓ ECR(Elastic Container Registry) 서비스에 접속해서 Repository 생성 - springapiserver

이미지 스캔 설정

 **사용 중단 경고**
리포지토리 수준의 ScanOnPush 구성은 더 이상 사용되지 않으며 레지스트리 수준 스캔 필터로 대체됩니다.

푸시할 때 스캔
리포지토리에 푸시된 후 각 이미지를 자동으로 스캔하려면 [푸시할 때 스캔]을 활성화합니다. 비활성화하는 경우 스캔 결과를 얻으려면 각 이미지 스캔을 수동으로 시작해야 합니다.

☐ 비활성화된

암호화 설정

KMS 암호화
기본 암호화 설정을 사용하는 대신 AWS Key Management Service(KMS)를 사용하여 이 리포지토리에 저장된 이미지를 암호화할 수 있습니다.

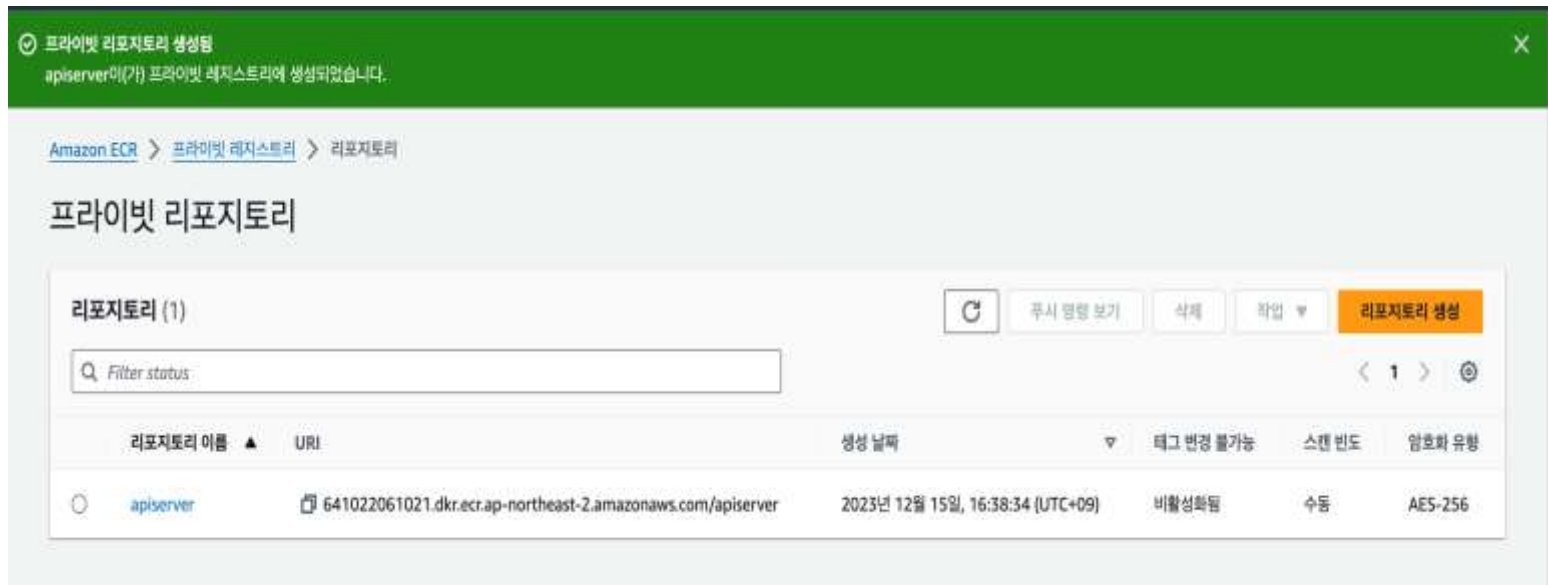
☐ 비활성화된

 리포지토리 생성 후 KMS 암호화 설정을 변경하거나 비활성화할 수 없습니다.

[취소](#) [리포지토리 생성](#)

AWS Container Service

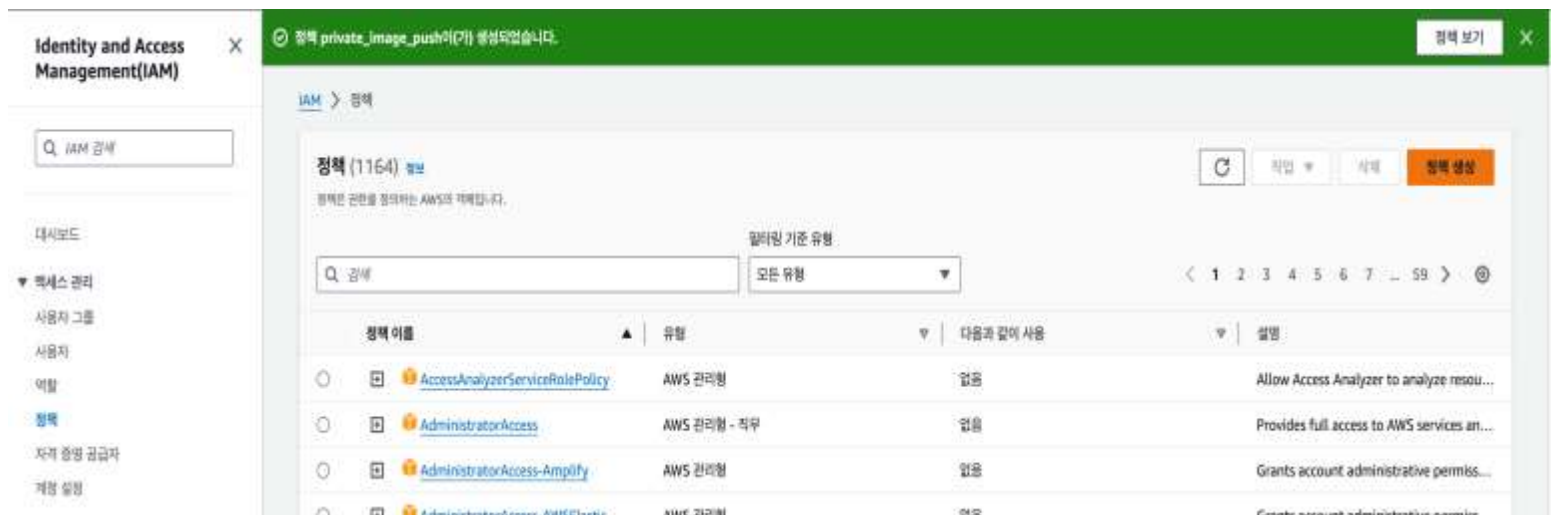
- ❖ Spring Boot Application CI/CD Pipeline 설정
 - ✓ ECR(Elastic Container Registry) 서비스에 접속해서 Repository 생성 - springapiserver



AWS Container Service

❖ Spring Boot Application CI/CD Pipeline 설정

- ✓ IAM에서 ECR에 이미지를 push할 수 있는 정책 생성(private_image_push): 리전을 수정하고 숫자와 이름은 ECR에서 가져와서 작성
 - 정책 생성 클릭



AWS Container Service

❖ Spring Boot Application CI/CD Pipeline 설정

- ✓ IAM에서 ECR에 이미지를 push할 수 있는 정책 생성(private_image_push): 리전을 수정하고 숫자와 이름은 ECR에서 가져와서 작성
 - JSON 클릭

권한 지정 정보

서비스, 작업, 리소스 및 조건을 선택하여 권한을 추가합니다. JSON 편집기를 사용하여 권한 설명문을 작성합니다.

정책 편집기

시각적 JSON 작업

▼ 서비스 선택

서비스의 특정 리소스에 대해 수행할 수 있는 작업을 지정합니다.

서비스

서비스 선택 ▼

+ 권한 더 추가

취소 다음

AWS Container Service

❖ Spring Boot Application CI/CD Pipeline 설정

- ✓ IAM에서 ECR에 이미지를 push할 수 있는 정책 생성(private_image_push): 리전을 수정하고 숫자와 이름은 ECR에서 가져와서 작성

- JSON 수정

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Effect": "Allow",  
      "Action": [  
        "ecr:CompleteLayerUpload",  
        "ecr:UploadLayerPart",  
        "ecr:InitiateLayerUpload",  
        "ecr:BatchCheckLayerAvailability",  
        "ecr:PutImage"  
      ],  
      "Resource": "arn:aws:ecr:ap-northeast-2:753510247212:repository/apiserver"  
    },  
  ],  
}
```

AWS Container Service

❖ Spring Boot Application CI/CD Pipeline 설정

- ✓ IAM에서 ECR에 이미지를 push할 수 있는 정책 생성(private_image_push): 리전을 수정하고 숫자와 이름은 ECR에서 가져와서 작성

- JSON 수정

```
{  
  "Effect": "Allow",  
  "Action": "ecr:GetAuthorizationToken",  
  "Resource": "*" }  
]  
}
```

AWS Container Service

❖ Spring Boot Application CI/CD Pipeline 설정

- ✓ IAM에서 ECR에 이미지를 push할 수 있는 정책 생성(private_image_push): 리전을 수정하고 숫자와 이름은 ECR에서 가져와서 작성

정책 세부 정보

정책 이름
이 정책을 식별하는 데 사용되는 이름을 입력합니다.

private_image_push

최대 128자입니다. 영문자 및 '+', '@', '_' 문자를 사용하세요.

설명 - 선택 사항
이 정책에 대하여 간단한 설명을 추가합니다.

ECR Image Push

최대 1,000자입니다. 영문자 및 '+', '@', '_' 문자를 사용하세요.

이 정책에 정의된 권한 정보

이 정책 문서에 정의된 권한은 허용되거나 거부되는 작업을 지정합니다. IAM 자격 증명(사용자, 사용자 그룹 또는 역할)에 대한 권한을 정의하려면 여기에 정책을 연결합니다.

Q 검색

허용(서비스 402개 중 1개) 나머지 서비스 401개 표시

서비스	액세스 수준	리소스	요청 조건
Elastic Container Registry	제한적: 읽기, 쓰기	Multiple	None

AWS Container Service

- ❖ Spring Boot Application CI/CD Pipeline 설정
 - ✓ OpenID Connect를 이용한 Git Hub AWS 로그인
 - IAM 에서 자격 증명 공급자를 클릭해서 추가

Adding the identity provider to AWS

To add the GitHub OIDC provider to IAM, see the [AWS documentation](#).

- For the provider URL: Use `https://token.actions.githubusercontent.com`
- For the "Audience": Use `sts.amazonaws.com` if you are using the [official action](#).

AWS Container Service

- ❖ Spring Boot Application CI/CD Pipeline 설정
 - ✓ OpenID Connect를 이용한 Git Hub AWS 로그인
 - IAM 에서 자격 증명 공급자를 클릭해서 추가
 - ◆ URL 확인: <https://docs.github.com/en/actions/deployment/security-hardening-your-deployments/configuring-openid-connect-in-amazon-web-services>

Adding the identity provider to AWS

To add the GitHub OIDC provider to IAM, see the [AWS documentation](#).

- For the provider URL: Use `https://token.actions.githubusercontent.com`
- For the "Audience": Use `sts.amazonaws.com` if you are using the [official action](#).

Configuring the role and trust policy

To configure the role and trust in IAM, see the AWS documentation "[Configure AWS Credentials for GitHub Actions](#)" and "[Configuring a role for GitHub OIDC identity provider](#)."

Note: AWS Identity and Access Management (IAM) recommends that users evaluate the IAM condition key, `token.actions.githubusercontent.com:sub`, in the trust policy of any role that trusts GitHub's OIDC identity provider (IdP). Evaluating this condition key in the role trust policy limits which GitHub actions are able to assume the role.

AWS Container Service

- ❖ Spring Boot Application CI/CD Pipeline 설정
 - ✓ OpenID Connect를 이용한 Git Hub AWS 로그인
 - IAM 에서 자격 증명 공급자를 클릭해서 추가

공급자 구성

공급자 유형 **정보**

☐ SAML

AWS 계정과 Shibboleth 또는 Active Directory Federation Services와 같은 SAML 2.0 호환 자격 증명 공급자 간에 신뢰를 설정합니다.

☒ OpenID Connect

AWS 계정과 Google 또는 Salesforce와 같은 자격 증명 공급자 서비스 간에 신뢰를 설정합니다.

공급자 URL

인증 요청에 대한 보안 OpenID Connect URL을 지정합니다.

URL 편집

최대 255자입니다. URL은 "https"로 시작해야 합니다.

⚠ AWS는 인증서 지문을 사용하여 idP의 서버 인증서를 확인하지 않고 신뢰할 수 있는 CA 라이브러리를 사용하여 이 OIDC ID 제 공업체(idP)와의 통신을 보호합니다. 레거시 지문은 구성에 남아 있지만 검증에 더 이상 필요하지 않습니다.

지문	발행자	CA 유효성
1b511abead59c6ce207077c0bf0e0043b1382612	DigiCert Inc	03/30/2021~03/30/2031

대상 **정보**

앱의 자격 증명 공급자가 발행한 클라이언트 ID를 지정합니다.

최대 255자입니다. 영숫자 또는 ~, / 문자를 사용하세요.

144

AWS Container Service

- ❖ Spring Boot Application CI/CD Pipeline 설정
 - ✓ OpenID Connect를 이용한 Git Hub AWS 로그인
 - 자격 증명 공급자에 새로운 역할 할당

The screenshot displays the AWS IAM console interface for the role `token.actions.githubusercontent.com`. The breadcrumb navigation shows `IAM > 자격 증명 공급자 > token.actions.githubusercontent.com`. The role name is `token.actions.githubusercontent.com` with a link to its details. Buttons for `역할 할당` (Attach) and `삭제` (Delete) are visible.

요약 (Summary) section:

공급자	공급자 유형	생성 시간	ARN
<code>token.actions.githubusercontent.com</code>	OpenID Connect	December 18, 2023, 10:17 (UTC+09:00)	<code>arnaws:iam::641022061021:oidc-provider/token.actions.githubusercontent.com</code>

대상 (1) (Groups (1)) section:

클라우드 ID라고도 하는 대상은 OpenID Connect 공급자에 등록된 애플리케이션을 식별하는 값입니다.

대상
<code>sts.amazonaws.com</code>

지문 (1) (Fingerprint (1)) section:

서버 인증서 지문은 OpenID Connect 공급자가 키를 제공하는 도메인에서 사용하는 X.509 인증서의 16진수 인코딩 SHA-1 해시 값입니다.

최대 5개의 지문을 추가할 수 있습니다. 이렇게 하면 자격 증명 공급자가 인증서를 교체하는 경우 여러 지문을 유지할 수 있습니다.

주의: AWS는 인증서 지문을 사용하여 IDP의 서버 인증서를 확인하지 않고 신뢰할 수 있는 CA 라이브러리를 사용하여 이 OIDC ID 제공업체(IDP)와의 통신을 보호합니다. 레거시 지문은 구성에 남아 있지만 검증에 더 이상 필요하지 않습니다.

ARN: `1b511abead59c6ce207077c0bf0e0043b1382612`

AWS Container Service

- ❖ Spring Boot Application CI/CD Pipeline 설정
 - ✓ OpenID Connect를 이용한 Git Hub AWS 로그인
 - 자격 증명 공급자에 새로운 역할 할당

신뢰할 수 있는 엔터티 선택 [정보](#)

신뢰할 수 있는 엔터티 유형

☐ AWS 서비스 EC2, Lambda 등의 AWS 서비스가 이 계정에서 작업을 수행하도록 허용합니다.	☐ AWS 계정 사용자 또는 서드 파티에 속한 다른 AWS 계정의 엔터티가 이 계정에서 작업을 수행하도록 허용합니다.
☒ 웹 자격 증명 지정된 외부 웹 자격 증명 공급자와 연동된 사용자가 이 역할을 맡아 이 계정에서 작업을 수행하도록 허용합니다.	☐ SAML 2.0 연동 기업 디렉터리에서 SAML 2.0과 연동된 사용자가 이 계정에서 작업을 수행할 수 있도록 허용합니다.
☐ 사용자 지정 신뢰 정책 다른 사용자가 이 계정에서 작업을 수행할 수 있도록 사용자 지정 신뢰 정책을 생성합니다.	

AWS Container Service

- ❖ Spring Boot Application CI/CD Pipeline 설정
 - ✓ OpenID Connect를 이용한 Git Hub AWS 로그인
 - 자격 증명 공급자에 새로운 역할 할당

웹 자격 증명
지정된 외부 웹 자격 증명 공급자와 연동된 사용자가 이 역할에 맡겨 이 계정에서 작업을 수행하도록 허용합니다.

자격 증명 공급자
  [새로 생성](#)

Audience

GitHub 조직

최대 39입니다.

GitHub 리포지토리 - 선택 사항

최대 100입니다.

GitHub 브랜치 - 선택 사항

최대 255입니다.

AWS Container Service

- ❖ Spring Boot Application CI/CD Pipeline 설정
 - ✓ OpenID Connect를 이용한 Git Hub AWS 로그인
 - Role에 이전에 만든 Policy를 할당

권한 추가 정보

권한 정책 (885) 정보 ↻

새 역할에 연결할 정책을 하나 이상 선택합니다.

필터링 기준 유형

🔍 djan X 모든 유형 ▼ 1 개 일치 < 1 > ⚙

☐	정책 이름 🔗	▲	유형 ▼	설명
☐	+ privateDjangoECR		고객 관리형	-

▶ 권한 경계 설정 - 선택 사항

취소 이전 다음

AWS Container Service

- ❖ Spring Boot Application CI/CD Pipeline 설정
 - ✓ OpenID Connect를 이용한 Git Hub AWS 로그인
 - Role에 이전에 만든 Policy를 할당

이름 지정, 검토 및 생성

역할 세부 정보

역할 이름
이 역할을 식별하는 의미 있는 이름을 입력합니다.

apiserverrole

최대 64자입니다. 영숫자 및 '*' , '@' , '-' 문자를 사용하세요.

설명
이 역할에 대하여 간단한 설명을 추가합니다.

|

최대 1000자입니다. 영숫자 및 '*' , '@' , '-' 문자를 사용하세요.

AWS Container Service

❖ Spring Boot Application CI/CD Pipeline 설정

✓ OpenID Connect를 이용한 Git Hub AWS 로그인

- Git Action.yml 파일을 수정해서 로그인이 되는지 확인: Role의 ARN 값으로 수정
name: Java CI with Gradle

동작 조건 설정 : main 브랜치에 push 혹은 pull request가 발생할 경우 동작한다.
on:

```
push:  
  branches: [ "main" ]  
pull_request:  
  branches: [ "main" ]
```

permissions:

```
id-token: write # This is required for requesting the JWT  
contents: read # This is required for actions/checkout
```

AWS Container Service

❖ Spring Boot Application CI/CD Pipeline 설정

✓ OpenID Connect를 이용한 Git Hub AWS 로그인

- Git Action.yml 파일을 수정해서 로그인이 되는지 확인

jobs:

Spring Boot 애플리케이션을 빌드하여 도커 허브에 푸시하는 과정

build-docker-image:

runs-on: ubuntu-latest

steps:

- uses: actions/checkout@v3

1. Java 17 세팅

- name: Set up JDK 17

uses: actions/setup-java@v3

with:

java-version: '17'

distribution: 'temurin'

2. Spring Boot 애플리케이션 빌드

- name: Build with Gradle

uses: gradle/gradle-build-

action@67421db6bd0bf253fb4bd25b31ebb98943c375e1

with:

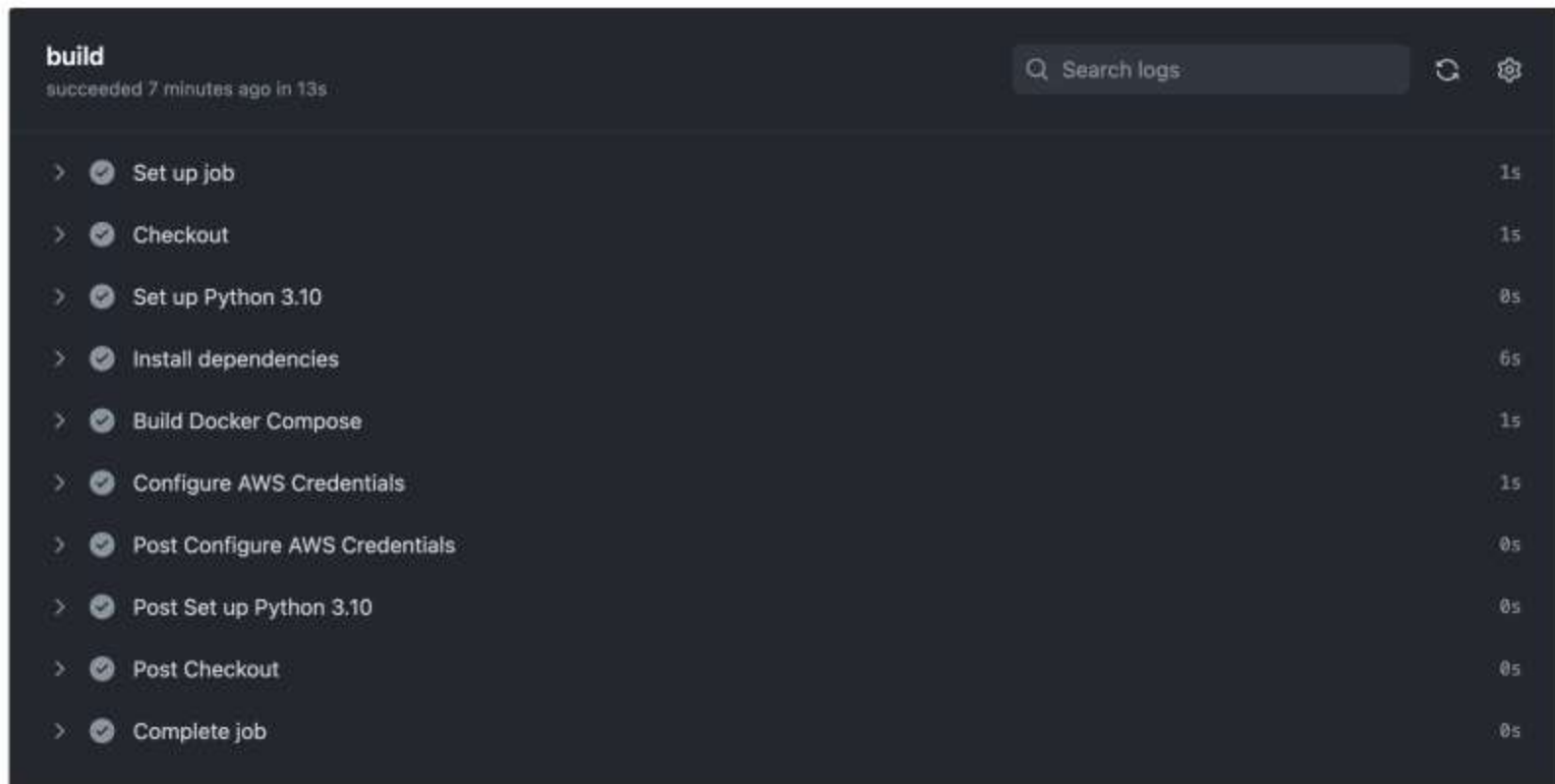
arguments: clean bootJar

AWS Container Service

- ❖ Spring Boot Application CI/CD Pipeline 설정
 - ✓ OpenID Connect를 이용한 Git Hub AWS 로그인
 - Git Action.yml 파일을 수정해서 로그인이 되는지 확인
 - name: Build Docker Compose
run: docker-compose
 - name: Configure AWS Credentials
uses: aws-actions/configure-aws-credentials@v4
with:
role-to-assume: arn:aws:iam::641022061021:role/itstudyoidc
role-session-name: sampleSessionName
aws-region: ap-northeast-2

AWS Container Service

- ❖ Spring Boot Application CI/CD Pipeline 설정
 - ✓ OpenID Connect를 이용한 Git Hub AWS 로그인



AWS Container Service

- ❖ Spring Boot Application CI/CD Pipeline 설정
 - ✓ ECR Push And Task Definition 생성 및 Service Update
 - <https://docs.github.com/en/enterprise-cloud@latest/actions/deployment/deploying-to-your-cloud-provider/deploying-to-amazon-elastic-container-service> 참고
 - 사전 작업
 - ◆ ECR Repository 생성
 - ◆ ECS Cluster, Task, Service 생성
 - ◆ task-definition.json 파일 생성: ECS에서 생성한 task 의 JSON을 복사해서 프로젝트에 생성 – image 부분은 제거

AWS Container Service

- ❖ Spring Boot Application CI/CD Pipeline 설정
 - ✓ ECR Push And Task Definition 생성 및 Service Update
 - Git Action.yml 파일을 수정 – 환경 변수는 적절하게 수정

name: django

on:

push:

branches: ["main"]

pull_request:

branches: ["main"]

permissions:

id-token: write

contents: read

env:

AWS_REGION: ap-northeast-2

ECR_REPOSITORY: itstudydjangoecs

ECS_SERVICE: itstudyservice2

ECS_CLUSTER: deploytest

ECS_TASK_DEFINITION: ./task-definition.json

CONTAINER_NAME: itstudy1

AWS Container Service

- ❖ Spring Boot Application CI/CD Pipeline 설정
 - ✓ ECR Push And Task Definition 생성 및 Service Update
 - Git Action.yml 파일을 수정 – 환경 변수는 적절하게 수정

jobs:

build-docker-image:

#ubuntu 환경에서 수행

runs-on: ubuntu-latest

steps:

#소스 코드 가져오기

- uses: actions/checkout@v3

#JDK 설치

- name: Set up JDK 17

uses: actions/setup-java@v3

with:

java-version: '17'

distribution: 'temurin'

AWS Container Service

❖ Spring Boot Application CI/CD Pipeline 설정

✓ ECR Push And Task Definition 생성 및 Service Update

- Git Action.yml 파일을 수정 – 환경 변수는 적절하게 수정

jobs:

Spring Boot 애플리케이션을 빌드하여 도커 허브에 푸시하는 과정

build-docker-image:

runs-on: ubuntu-latest

steps:

- uses: actions/checkout@v3

1. Java 17 세팅

- name: Set up JDK 17

uses: actions/setup-java@v3

with:

java-version: '17'

distribution: 'temurin'

2. Spring Boot 애플리케이션 빌드

- name: Build with Gradle

uses: gradle/gradle-build-

action@67421db6bd0bf253fb4bd25b31ebb98943c375e1

with:

arguments: clean bootJar

AWS Container Service

- ❖ Spring Boot Application CI/CD Pipeline 설정
 - ✓ ECR Push And Task Definition 생성 및 Service Update
 - Git Action.yml 파일을 수정 – 환경 변수는 적절하게 수정
 - name: Configure AWS Credentials
 - uses: aws-actions/configure-aws-credentials@v4
 - with:
 - role-to-assume: arn:aws:iam::641022061021:role/imagepush
 - role-session-name: sampleSessionName
 - aws-region: ap-northeast-2
 - name: Login to Amazon ECR
 - id: login-ecr
 - uses: aws-actions/amazon-ecr-login@62f4f872db3836360b72999f4b87f1ff13310f3a

AWS Container Service

❖ Spring Boot Application CI/CD Pipeline 설정

✓ ECR Push And Task Definition 생성 및 Service Update

● Git Action.yml 파일을 수정 – 환경 변수는 적절하게 수정

```
- name: Build, tag, and push image to Amazon ECR
  id: build-image
  env:
    ECR_REGISTRY: ${ steps.login-ecr.outputs.registry }
    IMAGE_TAG: ${ github.sha }
  run: |
    # Build a docker container and
    # push it to ECR so that it can
    # be deployed to ECS.
    docker build -t $ECR_REGISTRY/$ECR_REPOSITORY:$IMAGE_TAG .
    docker push $ECR_REGISTRY/$ECR_REPOSITORY:$IMAGE_TAG
    echo "image=$ECR_REGISTRY/$ECR_REPOSITORY:$IMAGE_TAG" >>
    $GITHUB_OUTPUT
```

AWS Container Service

- ❖ Spring Boot Application CI/CD Pipeline 설정
 - ✓ ECR Push And Task Definition 생성 및 Service Update
 - Git Action.yml 파일을 수정 – 환경 변수는 적절하게 수정
 - name: Fill in the new image ID in the Amazon ECS task definition
 - id: task-def
 - uses: aws-actions/amazon-ecs-render-task-definition@c804dfbdd57f713b6c079302a4c01db7017a36fc
 - with:
 - task-definition: \${{ env.ECS_TASK_DEFINITION }}
 - container-name: \${{ env.CONTAINER_NAME }}
 - image: \${{ steps.build-image.outputs.image }}
 - name: Deploy Amazon ECS task definition
 - uses: aws-actions/amazon-ecs-deploy-task-definition@v2
 - with:
 - task-definition: \${{ steps.task-def.outputs.task-definition }}
 - service: \${{ env.ECS_SERVICE }}
 - cluster: \${{ env.ECS_CLUSTER }}
 - wait-for-service-stability: true

AWS Container Service

- ❖ Spring Boot Application CI/CD Pipeline 설정
 - ✓ ECR Push And Task Definition 생성 및 Service Update
 - Git Action.yml 파일을 수정 – 환경 변수는 적절하게 수정

```
> [x] Build Docker Compose
> [x] Configure AWS Credentials
v [x] Login to Amazon ECR

1 ▶ Run aws-actions/amazon-ecr-login@62f4f872db3836360b72999f4b87f1ff13310f3a
22 (node:1816) NOTE: The AWS SDK for JavaScript (v2) will be put into maintenance mode in 2023.
23
24 Please migrate your code to use AWS SDK for JavaScript (v3).
25 For more information, check the migration guide at https://a.co/7PzMCcy
26 (Use 'node --trace-warnings ...' to show where the warning was created)
27 Error: User: arn:aws:sts::641022061021:assumed-role/itstudyoicd/sampleSessionName is not authorized to perform:
    ecr:GetAuthorizationToken on resource: * because no identity-based policy allows the ecr:GetAuthorizationToken action

  Build, tag, and push image to Amazon ECR
  Fill in the new image ID in the Amazon ECS task definition
  Deploy Amazon ECS task definition
```

AWS Container Service

- ❖ Spring Boot Application CI/CD Pipeline 설정
 - ✓ ECR Push And Task Definition 생성 및 Service Update
 - 권한 부여 – IAM에서 이전에 만든 Role에 ECS 관련 권한 추가 후 다시 빌드

권한 정책 (3) 정보 시뮬레이션 제거 권한 추가 ▼

최대 10개의 관리형 정책을 연결할 수 있습니다.

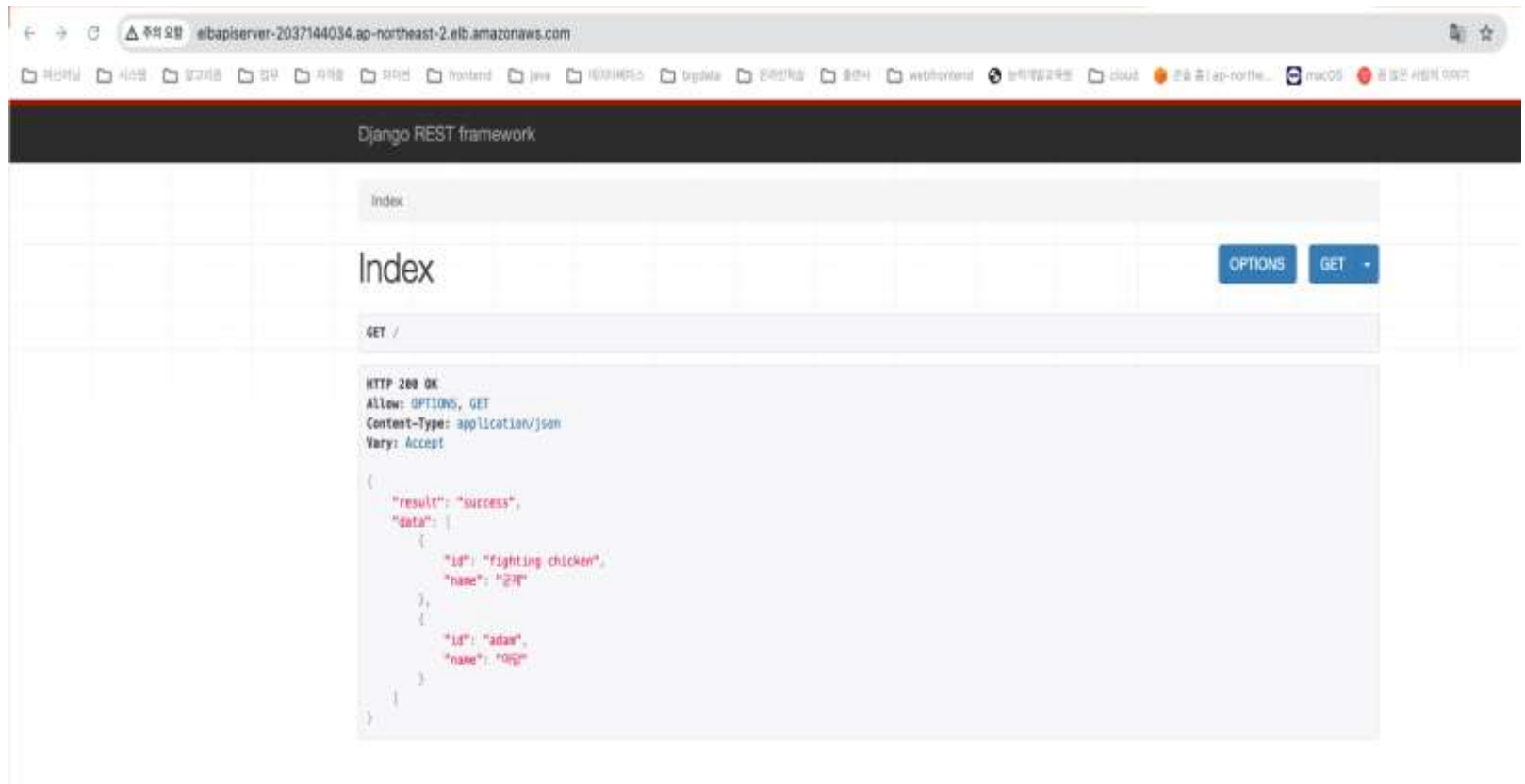
필터링 기준 유형

검색 모든 유형 ▼ < 1 > ⚙

☐	정책 이름 ↗	유형 ↗	연결된 엔터티 ↗
☐	AmazonECS_FullAccess	AWS 관리형	1
☐	AmazonECSTaskExecutionRole...	AWS 관리형	2
☐	privateDjangoECR	고객 관리형	2

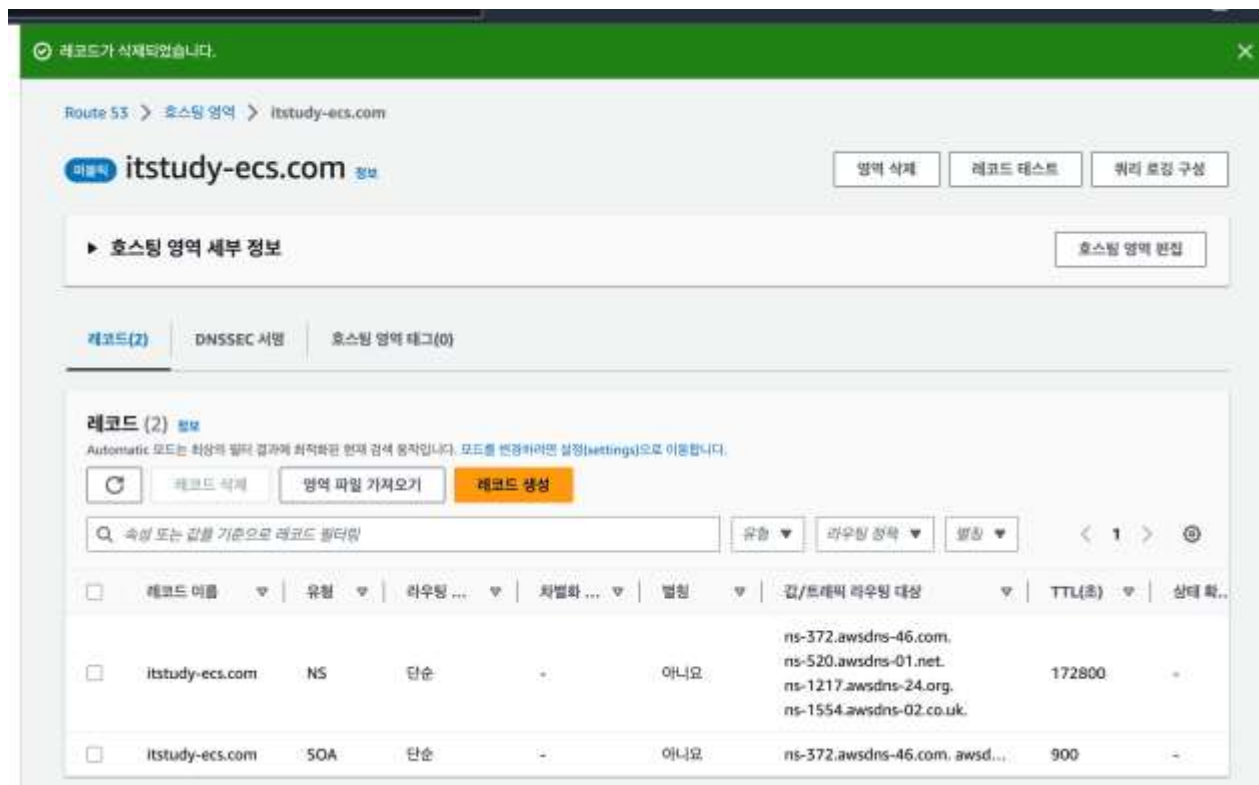
AWS Container Service

- ❖ Spring Boot Application CI/CD Pipeline 설정
 - ✓ 내용을 수정하고 푸시



AWS Container Service

- ❖ 도메인 연결
 - ✓ Domain 연결
 - 호스팅 영역에서 호스팅 영역 선택



AWS Container Service

❖ 도메인 연결

✓ Domain 연결

- 레코드 생성 버튼을 누르고 정보 입력

레코드 생성 정보

빠른 레코드 생성 마법사로 전환

▼ 레코드 1 삭제

레코드 이름 정보 레코드 유형 정보

루트 도메인에 대한 레코드를 생성하려면 비워 두세요.

☒ 별칭

트래픽 라우팅 대상 정보

별칭 호스팅 영역 ID: ZWKZPGT148KDX

라우팅 정책 정보 대상 상태 평가 ☒ 예

다른 레코드 추가

취소 레코드 생성

AWS Container Service

❖ 도메인 연결

✓ Domain 연결

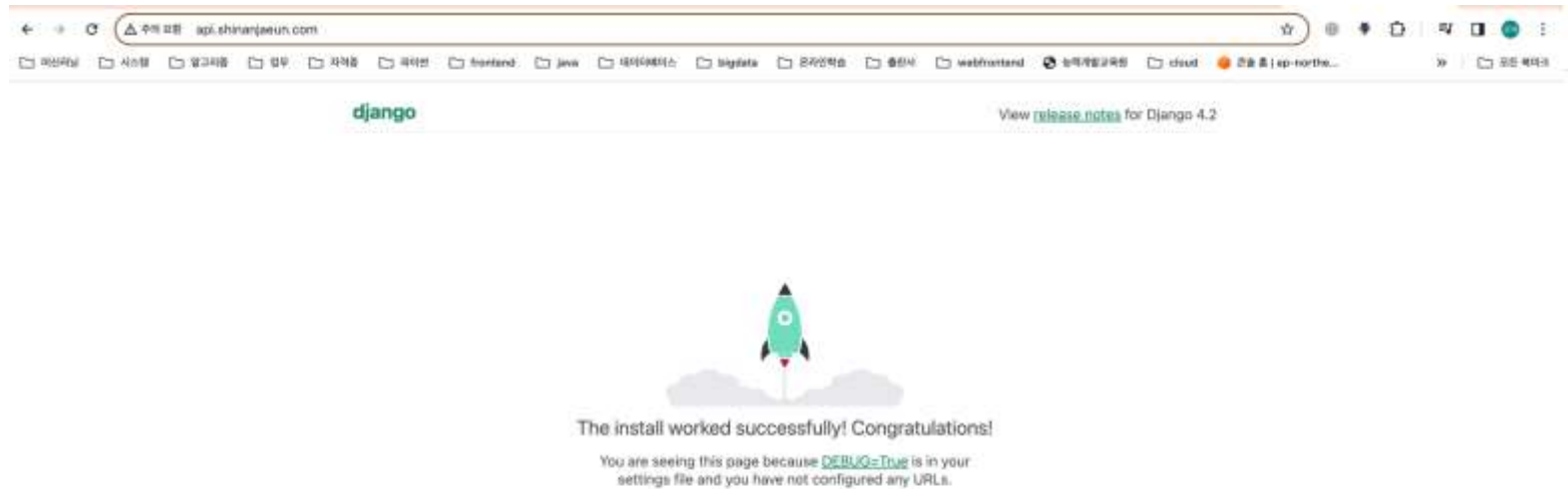
- 레코드 생성 버튼을 누르고 정보 입력

레코드 (3) 정보
Automatic 모드는 최상의 필터 결과에 최적화된 현재 검색 동작입니다. 모드를 변경하려면 설정(settings)으로 이동합니다.

☐	레코드 이름 ▼	유형 ▼	라우팅 ... ▼	차별화 ... ▼	별칭 ▼	값/트래픽 라우팅 대상 ▼	TTL(초) ▼	상태 확..
☐	itstudy-ecs.com	NS	단순	-	아니오	ns-372.awsdns-46.com, ns-520.awsdns-01.net, ns-1217.awsdns-24.org, ns-1554.awsdns-02.co.uk.	172800	-
☐	itstudy-ecs.com	SOA	단순	-	아니오	ns-372.awsdns-46.com, awsd...	900	-
☐	api.itstudy-ec...	A	단순	-	아니오	3.39.118.87	300	-

AWS Container Service

- ❖ 도메인 연결
 - ✓ 브라우저에서 확인



AWS Container Service

- ❖ HTTPS 적용
 - ✓ HTTPS 인증서 발급
 - AWS Certificate Manager 서비스에 접속



AWS Container Service

❖ HTTPS 적용

✓ HTTPS 인증서 발급

● 인증서 요청 클릭



AWS Container Service

❖ HTTPS 적용

✓ HTTPS 인증서 발급

● 인증서 요청을 위한 도메인 설정

퍼블릭 인증서 요청

도메인 이름

인증서에 대해 하나 이상의 도메인 이름을 제공합니다.

원전히 정규화된 도메인 이름 [정보](#)

api.shinajaeun.com

이 인증서에 다른 이름 추가

이 인증서에 이름을 추가할 수 있습니다. 예를 들어, 'www.example.com'에 대한 인증서를 요청하는 경우 고객어 두 이름 중 하나로 시어브에 접속할 수 있도록 'example.com'이라는 이름을 추가할 수 있습니다.

검증 방법 [정보](#)

도메인 소유권을 검증하기 위한 방법 선택

☒ DNS 검증 – 권장

인증서 요청에서 도메인에 대한 DNS 구성을 수정할 권한이 있는 경우 이 옵션을 선택합니다.

☐ 이메일 검증

인증서 요청에서 도메인에 대한 DNS 구성을 수정할 권한을 소유하지 않거나 획득할 수 없는 경우 이 옵션을 선택합니다.

AWS Container Service

❖ HTTPS 적용

✓ HTTPS 인증서 발급

- 인증서를 클릭하고 Route53에서 레코드 생성 클릭

8d320d9c-9ad2-4a64-a24b-08145e37e06f 삭제

인증서 상태

식별자: 8d320d9c-9ad2-4a64-a24b-08145e37e06f 상태: [검증 대기 중](#) [정보](#)

ARN: [arn:aws:acm:ap-northeast-2:641022061021:certificate/8d320d9c-9ad2-4a64-a24b-08145e37e06f](#)

유형: Amazon 발급

도메인 (1) Route 53에서 레코드 생성 CSV로 내보내기

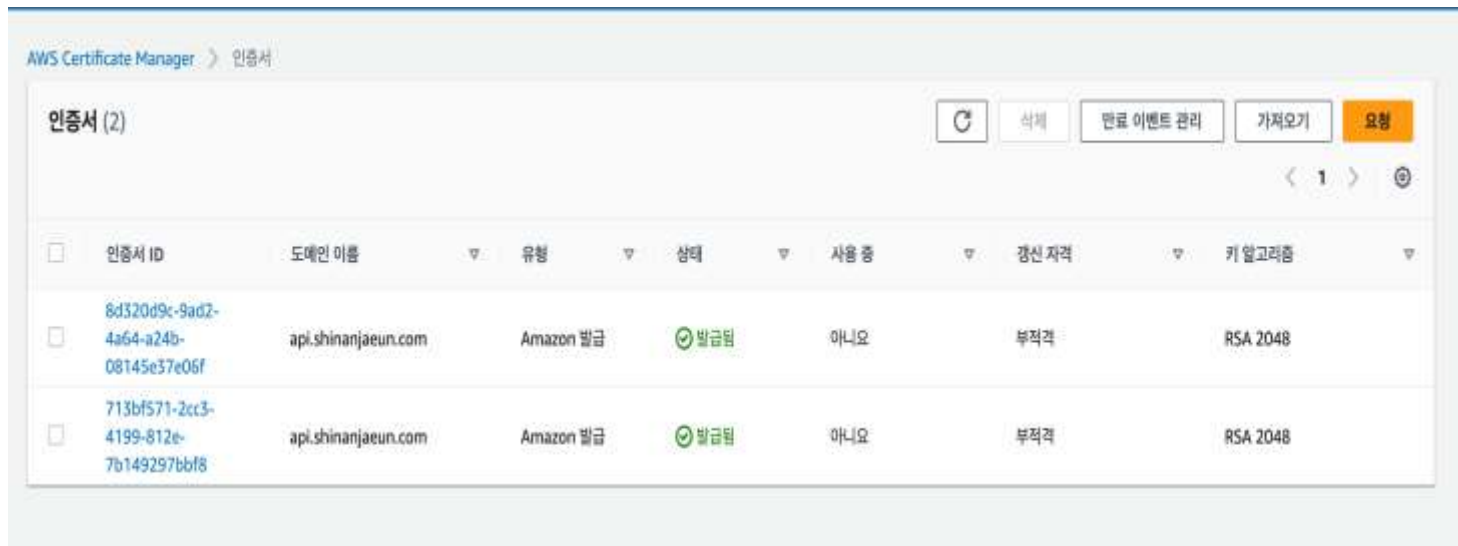
도메인	상태	경신 상태	유형	CNAME 이름	CNAME 값
api.shinanjaeun.com	검증 대기 중	-	CNAME	_ad7d46b52cfa91db99d656e0b6ae318c.api.shinanjaeun.com.	_ab922b26f78c9d6240ff9b479d190908.mhbtspdn.acm-validations.aws.

AWS Container Service

❖ HTTPS 적용

✓ HTTPS 인증서 발급

- 인증서를 클릭하고 Route53에서 레코드 생성 클릭

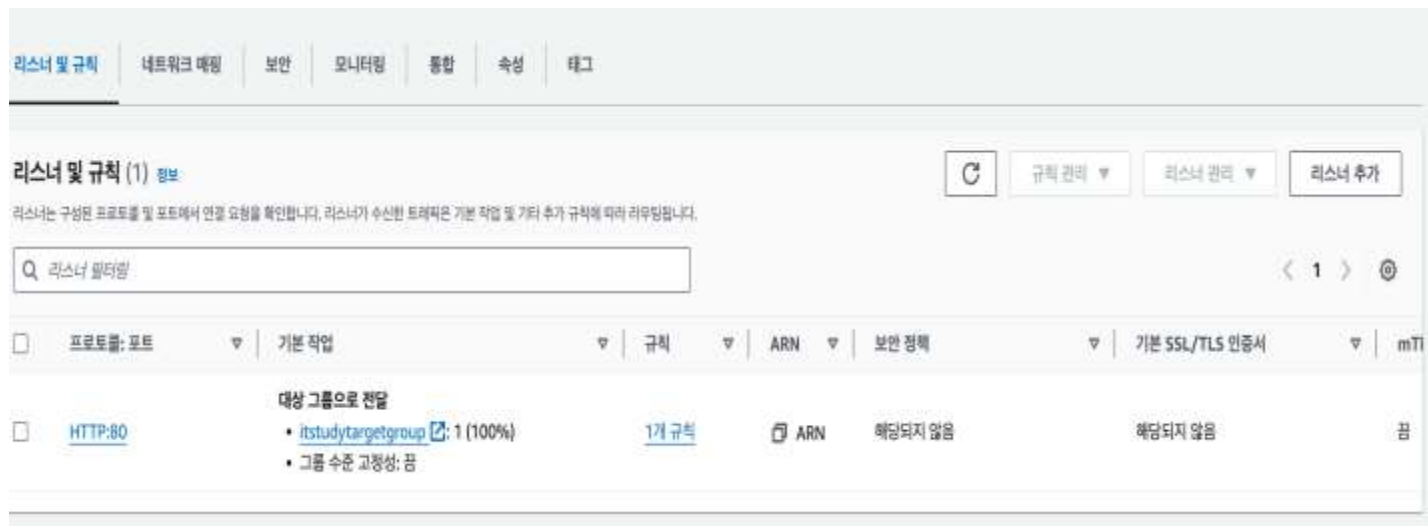


AWS Container Service

❖ HTTPS 적용

✓ ELB에 인증서 등록

- 로드 밸런서에 접속해서 [리스너 추가] 버튼 클릭



AWS Container Service

❖ HTTPS 적용

✓ ELB에 인증서 등록

● 로드 밸런서 설정 – 프로토콜을 HTTPS로 변경

리스너 세부 정보: HTTPS:443

리스너는 사용자가 구성된 프로토콜 및 포트를 사용하여 연결 요청을 확인합니다. 생성한 기본 작업 및 추가 규칙에 따라 Application Load Balancer가 요청을 등록된 대상으로 라우팅하는 방법이 결정됩니다.

리스너 구성

리스너는 프로토콜 및 포트 식별됩니다.

프로토콜	포트
클라이언트에서 로드 밸런서로 연결하기 위해 사용됩니다.	로드 밸런서가 연결을 위해 수신 대기하는 포트입니다.
HTTPS ▼	443 ↕
	1-65535

기본 작업 [정보](#)

다른 규칙이 적용되지 않는 경우 기본 작업이 사용됩니다. 이 리스너의 트래픽에 대해 기본 작업을 선택하세요.

인증 [정보](#)

☐ OpenID 또는 Amazon Cognito 사용

OpenID Connect(OIDC) 또는 Amazon Cognito를 사용한 인증을 포함합니다.

라우팅 액션

☒ 대상 그룹으로 전달

☐ URL로 리디렉션

☐ 고정 응답 반환

AWS Container Service

❖ HTTPS 적용

✓ ELB에 인증서 등록

● 로드 밸런서 설정 - 대상 그룹 선택

기본 작업 [정보](#)

다른 규칙이 적용되지 않는 경우 기본 작업이 사용됩니다. 이 리스너의 트래픽에 대해 기본 작업을 선택하세요.

인증 [정보](#)

☐ OpenID 또는 Amazon Cognito 사용

OpenID Connect(OIDC) 또는 Amazon Cognito를 사용한 인증을 포함합니다.

라우팅 액션

☒ 대상 그룹으로 전달

☐ URL로 리디렉션

☐ 고정 응답 반환

대상 그룹으로 전달 [정보](#)

대상 그룹을 선택하고 라우팅 가중치를 지정하거나 [대상 그룹을 생성](#)합니다.

대상 그룹

itstudytargetgroup

대상 유형: 인스턴스, IPv4

HTTP ▼



가중치

1

0-999

백분율

100%

대상 그룹 추가

최대 4개의 대상 그룹을 더 추가할 수 있습니다.

그룹 수준 고정성 [정보](#)

대상 그룹에 고정성이 설정되어 있는 경우, 해당 그룹에 라우팅된 요청은 세션 기간 동안 대상 그룹에 유지됩니다. 개별 대상 고정성은 대상 그룹의 구성 옵션입니다.

☐ 그룹 수준 고정 활성화

AWS Container Service

❖ HTTPS 적용

✓ ELB에 인증서 등록

● 로드 밸런서 설정 - 인증서 선택

보안 리스너 설정 정보

보안 정책 정보
로드 밸런서는 보안 정책이라고 하는 Secure Socket Layer(SSL) 협상 구성을 사용해 클라이언트와의 SSL 연결을 관리합니다. [보안 정책 비교](#)

보안 카테고리
모든 보안 정책 ▼

정책 이름
ELBSecurityPolicy-TLS13-1-2-2021-06 (권장) ▼

기본 SSL/TLS 서버 인증서
클라이언트가 SNI 프로토콜 없이 연결하거나 일치하는 인증서가 없는 경우에 사용되는 인증서입니다. 이 인증서를 AWS Certificate Manager(ACM), Amazon Identity and Access Management(IAM)에서 소싱하거나 인증서 가져오기를 실시할 수 있습니다. 이 인증서는 리스너 인증서 목록에 자동으로 추가됩니다.

인증서 소스

☒ ACM에서

☐ IAM에서

☐ 인증서 가져오기

인증서(ACM에서)
선택한 인증서는 이 로드 밸런서의 보안 리스너에 대한 기본 SSL/TLS 서버 인증서로 적용됩니다.

api.shinajaeun.com
8d320d9c-9ad2-4a64-a24b-0814...

↺

[새 ACM 인증서 요청](#)

클라이언트 인증서 처리 정보
클라이언트 인증서는 인증된 요청을 원격 서버로 보내는 데 사용됩니다. [자세히 알아보기](#)

☐ 상호 인증(mTLS)
상호 전송 계층 보안(TLS) 인증은 양방향 피어 인증을 제공합니다. TLS를 통한 보안 계층을 추가하고 서비스에서 연결을 설정하는 클라이언트를 확인할 수 있도록 합니다.