DevOps 구축 BOOTCAMP

5주차 목표

Docker 명령어 및 Dockerfile 작성 방법 이해

AWS ECS를 활용한 EC2 인스턴스를 컨테이너 구동 환경을 위한 클러스터링

AWS ALB를 활용하여 다이나믹 포트 매핑을 통한 컨테이너 트래픽 분산

Docker 실습

- 실행되고 있는 docker container 프로세스 확인

Docker 실습

1. ec2-user@ip-172-31-25-131:~ (bash)

a1@1ui-MacBook-Air:~$

docker stop web1

- 실행되고 있는 docker container 프로세스 종료
  - web1
    - docker run 시 name 옵션으로 지정한 컨테이너 이름으로 종료 가능
    - 혹은 container id 입력

Docker 실습

1. ec2-user@ip-172-31-25-131:~ (bash)

a1@1ui-MacBook-Air:~$

docker rm web1

docker rmi web:0.1

- rm / rmi : 컨테이너 / 이미지 삭제시 사용
- 컨테이너/이미지 이름 혹은 컨테이너/이미지 ID로 삭제
- 컨테이너 삭제시 실행 중이면 삭제 안됨. stop 명령어로 종료 후 삭제

Docker 실습

1. ec2-user@ip-172-31-25-131:~ (bash)

a1@1ui-MacBook-Air:~$

docker run -d -p 8000:80 --name web1 web:latest

- run : docker image를 기반으로 컨테이너 생성/실행
- -d : 데몬으로 실행
- -p : 포트 매핑(HOST에 8000포트로 들어온 요청을 컨테이너 내 80 포트와 연결)
- —name : 컨테이너 이름 설정
- web:latest : 해당 이미지를 기반으로 컨테이너 실행

Docker 실습

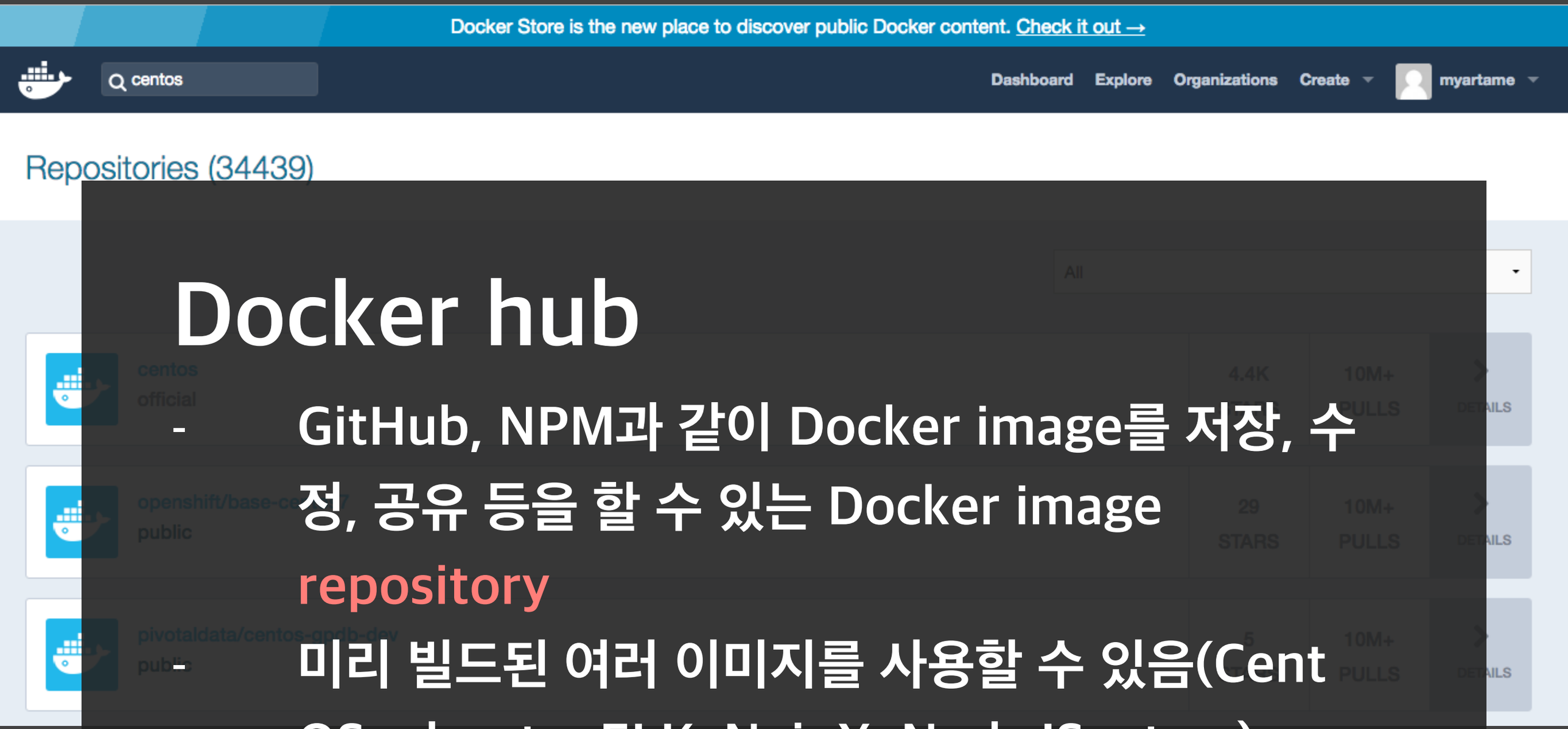
1. ec2-user@ip-172-31-25-131:~ (bash)

a1@1ui-MacBook-Air:~$

docker pull centos:latest

- pull : 외부 레포지토리에서 이미지 다운로드, 따로 명시된 레포지토리가 없다면 dockerhub(https://hub.docker.com)에서 다운로드
- centos : 다운로드 할 이미지 이름
- latest : 다운로드 할 이미지의 버전, latest로 명시하면 가장 최근에 업데이트 된 이미지를 가져옴

Docker hub



Docker hub

- GitHub, NPM과 같이 Docker image를 저장, 수정, 공유 등을 할 수 있는 Docker image repository
- 미리 빌드된 여러 이미지를 사용할 수 있음(CentOS, ubuntu, ELK, NginX, NodeJS, etc…)

Docker 실습

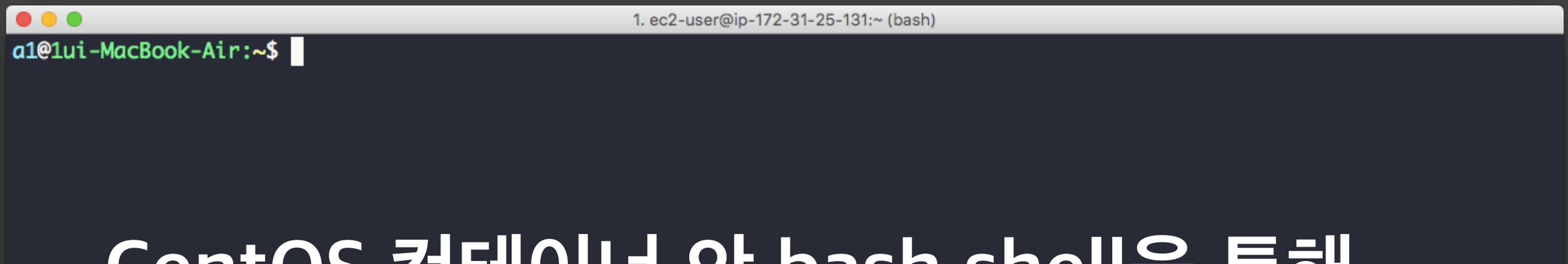
1. ec2-user@ip-172-31-25-131:~ (bash)

```
a1@1ui-MacBook-Air:~$
```

docker run -it --name cent centos:latest /bin/bash

- -it : -i, -t 옵션으로 실행
  - -i(interactive), -t(Pseudo-tty) : Bash shell에 입력 및 출력 가능
- /bin/bash : centos 이미지 내에 /bin/bash 실행

Docker 실습



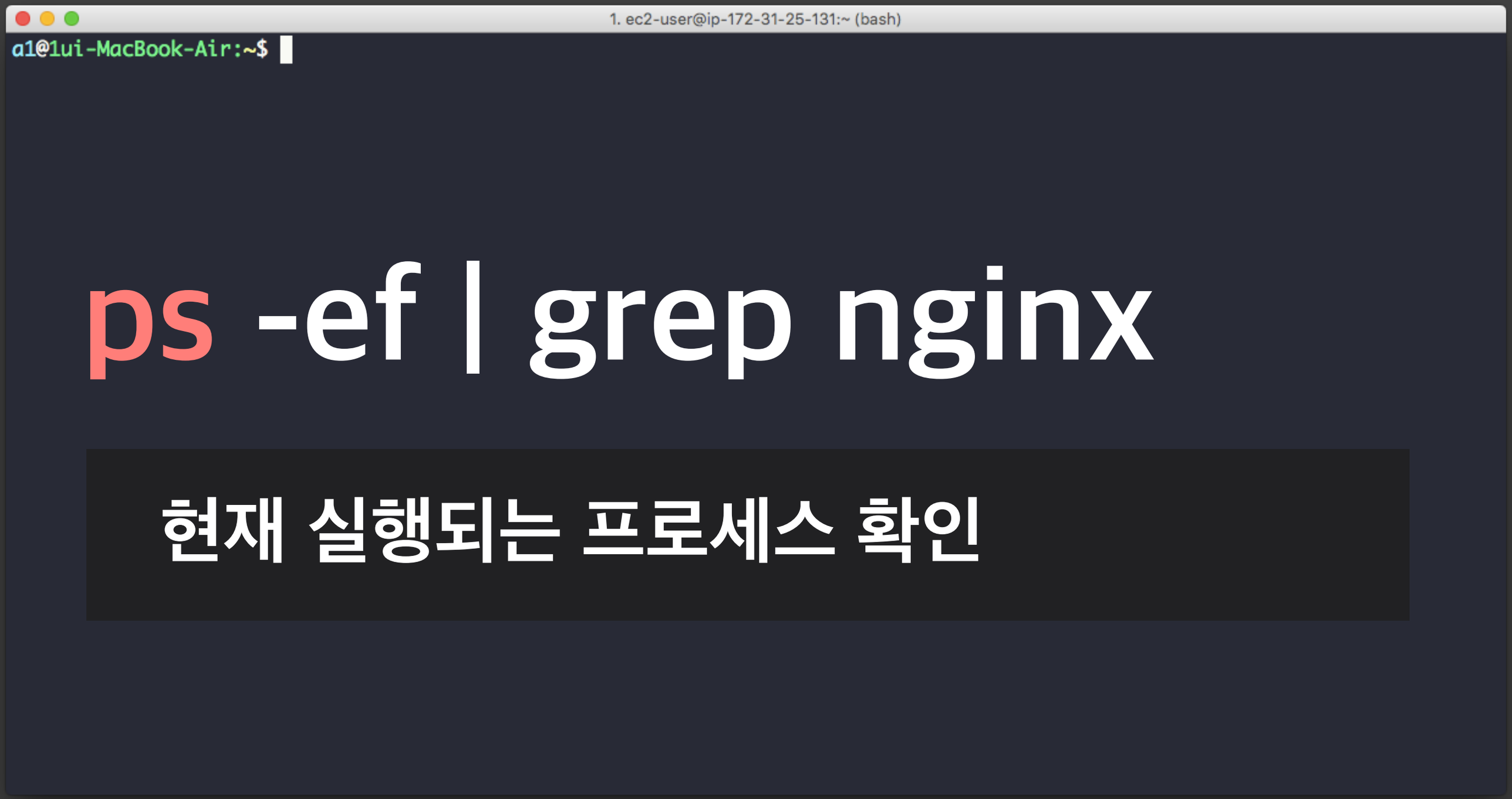
CentOS 컨테이너 안 bash shell을 통해

yum install -y epel-release

yum install -y nginx

nginx

Docker 실습



현재 실행되는 프로세스 확인

Docker 실습

1. ec2-user@ip-172-31-25-131:~ (bash)

a1@1ui-MacBook-Air:~$

ps -ef | grep nginx

현재 실행되는 프로세스 중 nginx 이름을 가지고 있는 프로세스 검색

- ps 뿐만 아니라 리스트 형태로 출력하는 대부분의 명령어(ls, tail 등)

Docker 실습

1. ec2-user@ip-172-31-25-131:~ (bash)

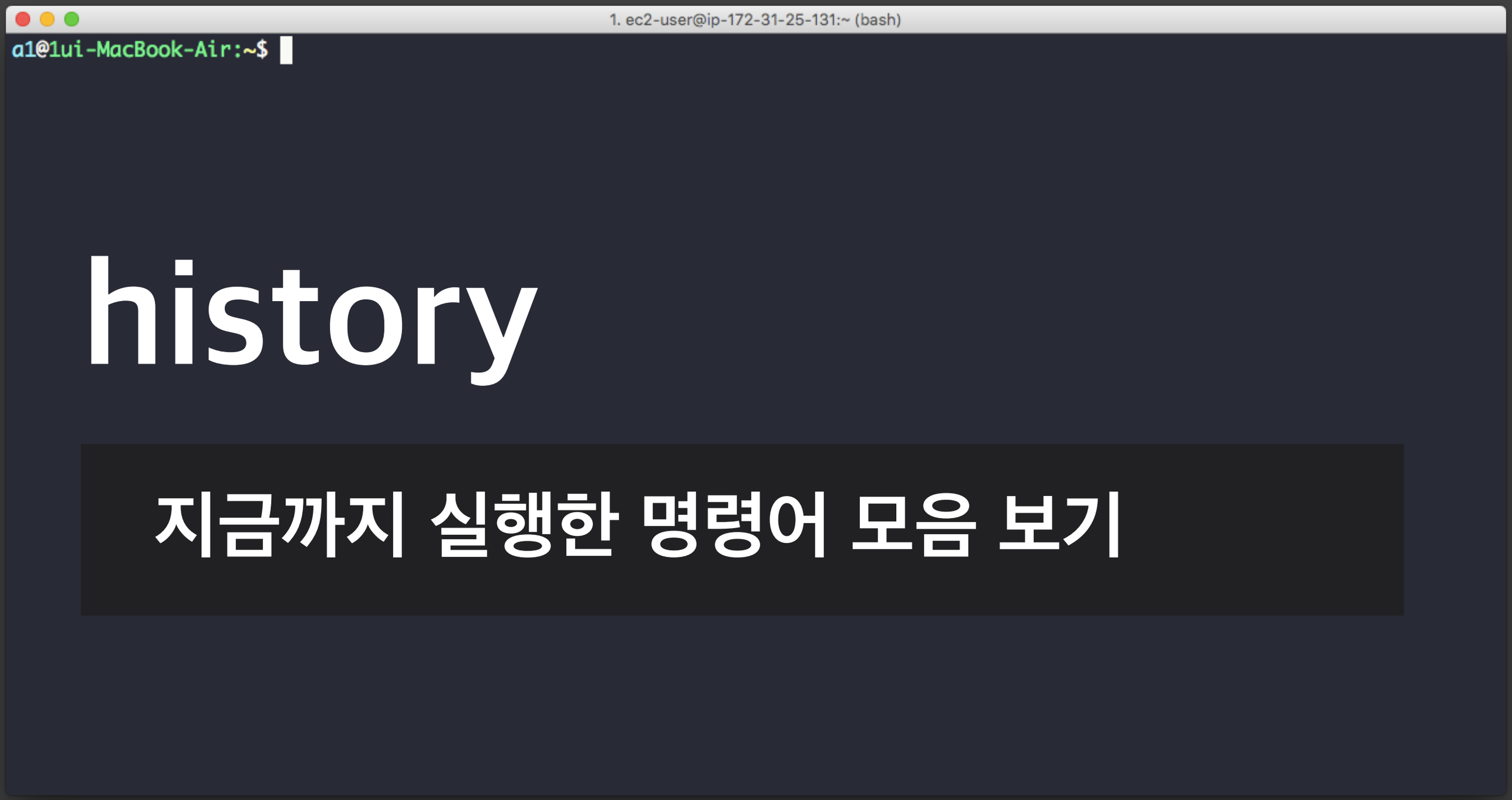
a1@1ui-MacBook-Air:~$

yum install -y nmap

nmap localhost | grep http

현재 열려있는 포트 확인

Docker 실습



지금까지 실행한 명령어 모음 보기

Dockerfile 작성하기

앞서 Cent OS에 작업한 명령어들을 기반으로 작성

1. 해당 컨테이너에서 어떤 OS를 기반으로 실행될 것인지
2. 필요한 소프트웨어를 해당 OS에 맞는 package manager 등을 통해 다운로드
3. 소스 코드, 설정 파일 다운로드, 적용
4. 컨테이너 내에서 소프트웨어 구동 및 Port listen

Docker 실습

```
FROM centos:latest

RUN yum install -y epel-release
RUN yum install -y nginx

COPY ./ /usr/share/nginx/html/Fastcampus-web-deploy

EXPOSE 80

CMD ["nginx", "-g", "daemon off;"]
```

Docker 실습

```
FROM centos:latest

RUN yum install -y epel-release
RUN yum install -y nginx

COPY ./ /usr/share/nginx/html/fastcampus-web-deploy

EXPOSE 80

CMD ["nginx", "-g", "daemon off;"]
```

- 컨테이너 내에서 기반으로 할 OS(정확히는 docker image)
- FROM {image_name:tag}
- tag에 latest로 명시하면 가장 최근에 업데이트 된 이미지 다운로드

Docker 실습

```
FROM centos:latest

RUN yum install -y epel-release
RUN yum install -y nginx

COPY ./ /usr/share/nginx/html/Fastcampus-web-deploy

EXPOSE 80

CMD ["nginx", "-g", "daemon off;"]
```

컨테이너 내에서 명령어 실행

- RUN {EXEC commend}
- RUN : 뒤에 오는 명령어 실행, 기반이 되는 컨테이너 OS의 체계에 맞춰 실행됨

컨테이너 외부에 있는 파일을 내부로 복사

- COPY {external path / internal path} : 디렉토리 / 파일 복사
- ./ : Dockerfile과 같은 경로에 있는 모든 파일
- 소스 코드, 설정 파일 등

```
FROM centos:latest

RUN yum install -y epel-release
RUN yum install -y nginx

COPY ./ /usr/share/nginx/html/Fastcampus-web-deploy

EXPOSE 80

CMD ["nginx", "-g", "daemon off;"]
```

= RUN yum install -y git

FROM centos:latest

RUN WORKDIR /usr/share/nginx/html

RUN git clone {git-repository-url.git}

COPY ./ /usr/share/nginx/html/Fastcampus-web-deploy

EXPOSE 80

CMD ["nginx", "-g", "daemon off;"]

FRO

RUN

RUN

COPY ./ /usr/share/nginx/html/Fastcampus-web-deploy

EXPOSE 80

CMD ["nginx", "-g", "daemon off;"]

WORKDIR /usr/share/nginx/html 경로 설정

- RUN으로 명령어 실행시 기본 Path는 /인 root directory에서 실행됨
- NginX root directory에 소스 코드를 배포하기 위해 /usr/share/nginx/html 경로로 작업

Docker 실습

EXPOSE

- HOST와 연결될 포트 번호
- Nginx는 80포트로 통신하기에 외부(host)와 연결하기 위해 80포트 설정
- HOST : Docker가 구동되고 있는 환경, 물리 머신, VM, 컨테이너 안이 될 수도 있음

EXPOSE 80

CMD ["nginx", "-g", "daemon off;"]

Docker 실습

FROM centos:latest

RUN yum install -y epel-release

RUN yum install -y nginx

CMD

- 컨테이너가 start(run) / restart 될 때 명령어 혹은 스크립트를 실행
- Dockerfile내에서 한 번만 사용 가능
  - RUN ~~과의 차이점

CMD ["nginx", "-g", "daemon off;"]

Docker 실습

FROM centos:latest

RUN yum install -y epel-release

RUN yum install -y nginx

CMD ["nginx", "-g", "daemon off;”]

- CMD [“실행 파일”, “매개 변수1”, “매개 변수 2”]
- Nginx를 “daemon off” 매개 변수를 통해 foreground로 실행

COPY ./ /usr/share/nginx/html/Fastcampus-web-deploy

EXPOSE 80

CMD ["nginx", "-g", "daemon off;"]

Foreground VS Background

Foreground

- 기본적으로 모든 프로세스는 foreground로 실행
- 키보드로부터 입력을 받아 결과를 직접 스크린으로 출력
- 한 커맨드가 실행되는 동안 다른 커맨드를 실행할 수 없음(yum, vim, etc… 실습 때를 생각해봅시다)

Background

- 키보드와 연결되지 않는 실행 방식
- 여러 프로세스를 실행할 수 있는 장점이 있음
- 프로세스 실행 시 명령어 뒤에 ‘&’를 붙히거나 Background로 실행되는 프로세스를 관리하는 daemon에 등록하여 실행

Foreground VS Background

컨테이너 내에선 Foreground로 실행

- 정확히는 컨테이너 내에서 실행할 프로세스들은 foreground로 해당 컨테이너는 '-d' 옵션을 통해 background로 실행
- 컨테이너를 Background로 실행하면 docker daemon에 등록되며 컨테이너 내에서 background list에 등록된 프로세스가 있다면 충돌이 발생되어 컨테이너 내에선 foreground로 실행하며 프로세스 관리는 docker host에게 전담

Dockerfile 작성하기

Docker container

Running processes

Project code & Configuration

Process

OS

Dockerfile 작성하기

Docker image

Running processes

Project code & Configuration

Process

1. 어떤 이미지/OS를 기반으로 할 것인지

Dockerfile 작성하기

Docker image

Running processes

Project code & Configuration

2. 컨테이너 내에서 구동시킬 소프트웨어의 다운로드 방식 결정 및 다운로드 진행

Dockerfile 작성하기

Docker image

Running processes

3. 해당 컨테이너에서 필요한 소스 코드 및 환경설정 파일을 적용시킬 방법(복사, Git으로 다운로드 등)

OS

Dockerfile 작성하기

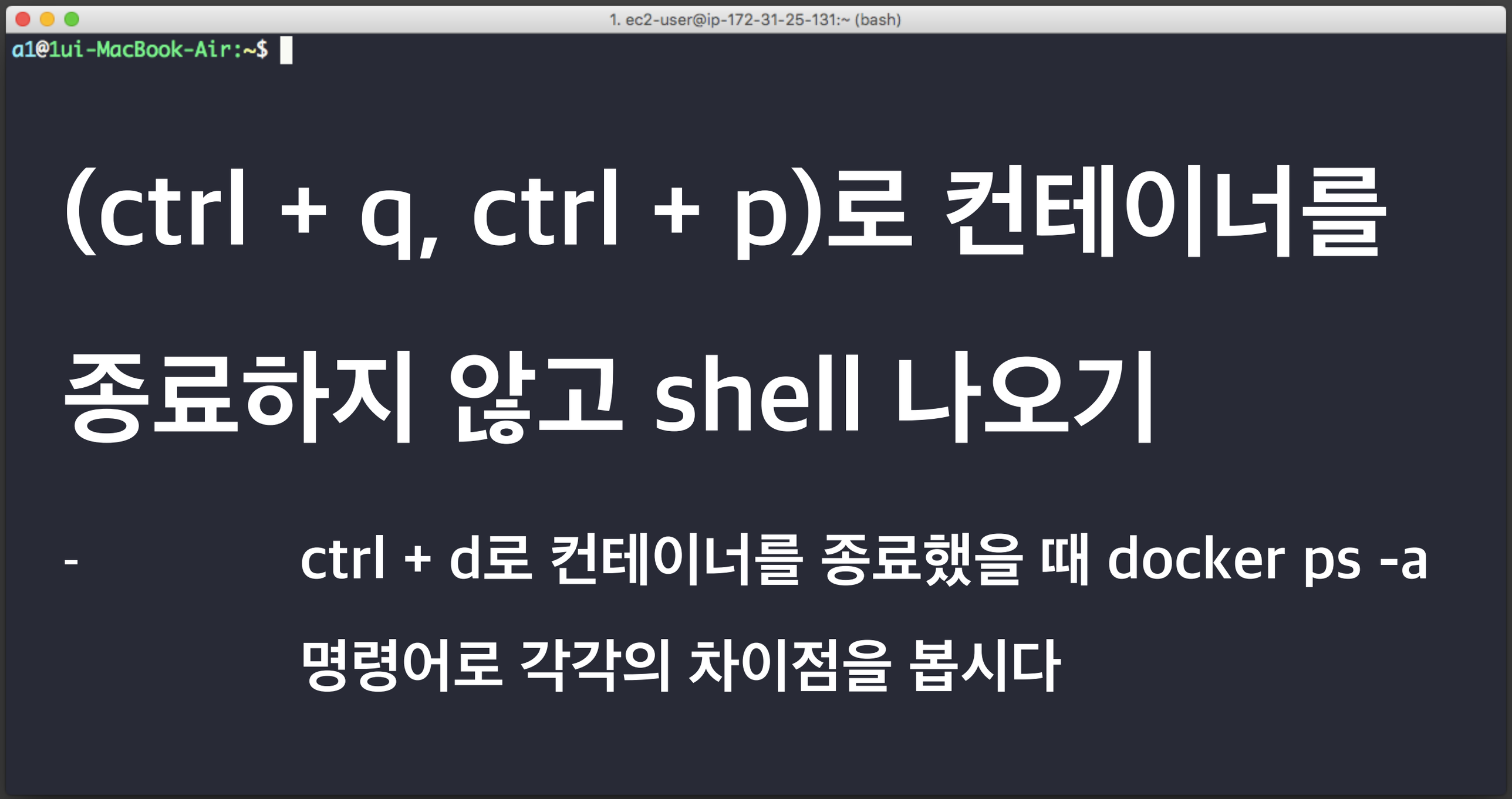
Docker image

4. 컨테이너 내에서 소프트웨어들을 foreground로 구동

Process

OS

Docker 실습



- ctrl + d로 컨테이너를 종료했을 때 docker ps -a 명령어로 각각의 차이점을 봅시다

Docker 실습

1. ec2-user@ip-172-31-25-131:~ (bash)

a1@1ui-MacBook-Air:~$

docker attach {container_name or id}

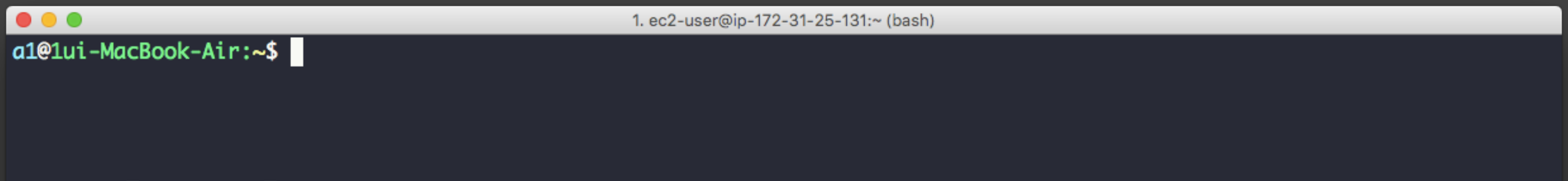
= docker exec -it {container_name or id} /bin/bash

- attach : 컨테이너 안으로 접속, bash shell로 접속
- exec : 컨테이너 외부에서 해당 컨테이너 안에서 명령어 실행시 사용하는 옵션

docker exec -it {container_name or id} /bin/bash

- docker run ‘-d’ 옵션으로 background로 컨테이너 실행시 bash shell이 아닌 CMD에 명시된 프로세스가 컨테이너 내에서 foreground로 실행되고 있어 attach로 shell 실행이 안됨
- Container가 background로 실행 중일 때 컨테이너 내에 접속하기 위해 exec 옵션을 통해 bash shell 실행

Docker 실습



docker stop cent

docker rm cent

- docker container를 삭제하기 전 종료를 먼저 실행하거나 ‘-f’ 옵션으로 강제 삭제 가능

Docker 실습

1. ec2-user@ip-172-31-25-131:~ (bash)

a1@1ui-MacBook-Air:~$

docker rm cent

- stop 상태가 아닌 컨테이너를 삭제하면 에러 발생, rm -f 옵션으로 강제종료를 하거나 컨테이너 stop 이후 종료

docker run -it --name cent centos:latest /bin/bash

- WAS 테스트를 위해 새로운 centos를 기반으로 하는 컨테이너를 실행합니다

Docker 실습 - WAS(NodeJS)

```
curl --silent --location https://rpm.nodesource.com/setup_8.x | bash -

yum install -y nodejs

yum install -y git

git clone -b v1 https://github.com/owen1025/Fastcampus-api-deploy.git

cd Fastcampus-api-deploy/

npm install -g pm2

npm install

pm2 start bin/www --name WAS
```

Docker 실습 - WAS(NodeJS)

```
curl --silent --location https://rpm.nodesource.com/setup_8.x | bash -

yum install -y nodejs
```

Node.js/NPM 코드 다운로드 및 yum을 통해 설치

```
npm install

pm2 start bin/www --name WAS
```

Docker 실습 - WAS(NodeJS)

```
curl --silent --location https://rpm.nodesource.com/setup_8.x | bash -

yum install -y nodejs

yum install -y git

git clone -b v1 https://github.com/owen1025/Fastcampus-api-deploy.git

cd Fastcampus-api-deploy/

npm install pm2

npm install

pm2 start bin/www --name WAS
```

Git 설치 및 git 레포지토리에서 api 프로젝트 코드 다운로드

Docker 실습 - WAS(NodeJS)

1. pm2(node.js process manager) 설치
2. npm을 통해 api 프로젝트에서 필요한 module 다운로드
3. pm2로 nodejs background로 실행

```
npm install -g pm2

npm install

pm2 start bin/www --name WAS
```

Docker 실습 - WAS(NodeJS)

curl --silent --location https://rpm.nodesource.com/setup_8.x | bash -

yum install -y nodejs

yum install -y git

git clone https://github.com/fastcampus-devops/Fastcampus-api-deploy.git

cd Fastcampus-api-deploy/

pm2로 nodejs background로 실행

=> node를 통해 foreground로 실행

npm install -g pm2

npm install

pm2 start bin/www --name WAS

Docker 실습 - WAS(NodeJS)

```
curl --silent --location https://rpm.nodesource.com/setup_8.x | bash -

yum install -y nodejs

cd Fastcampus-api-deploy/

npm install

node bin/www
```

Dockerfile 작성 - WAS(NodeJS)

1. ec2-user@ip-172-31-25-131:~ (bash)

a1@1ui-MacBook-Air:~$

로컬 환경에서 수행하기

git clone -b container https://github.com/owen1025/Fastcampus-api-deploy.git

해당 프로젝트에서 Dockerfile 열어보기

Dockerfile 작성 - WAS(NodeJS)

1. ec2-user@ip-172-31-25-131:~ (bash)

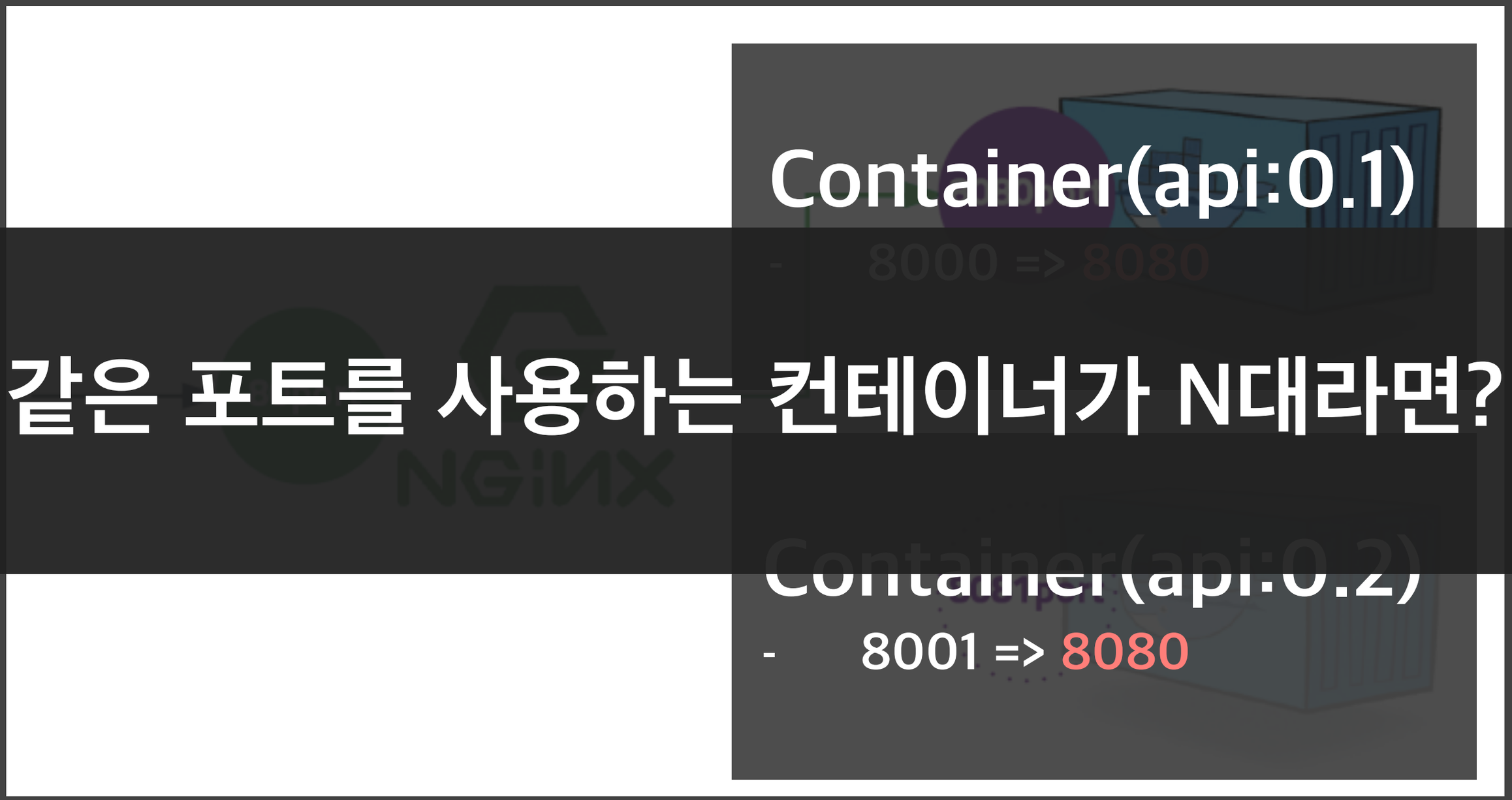
a1@1ui-MacBook-Air:~$

주석 처리 된 Dockerfile 작성을 끝낸 후

```
docker build -t api:0.1 .
docker run -d -p 3000:8080
—name api1 api:0.1
```

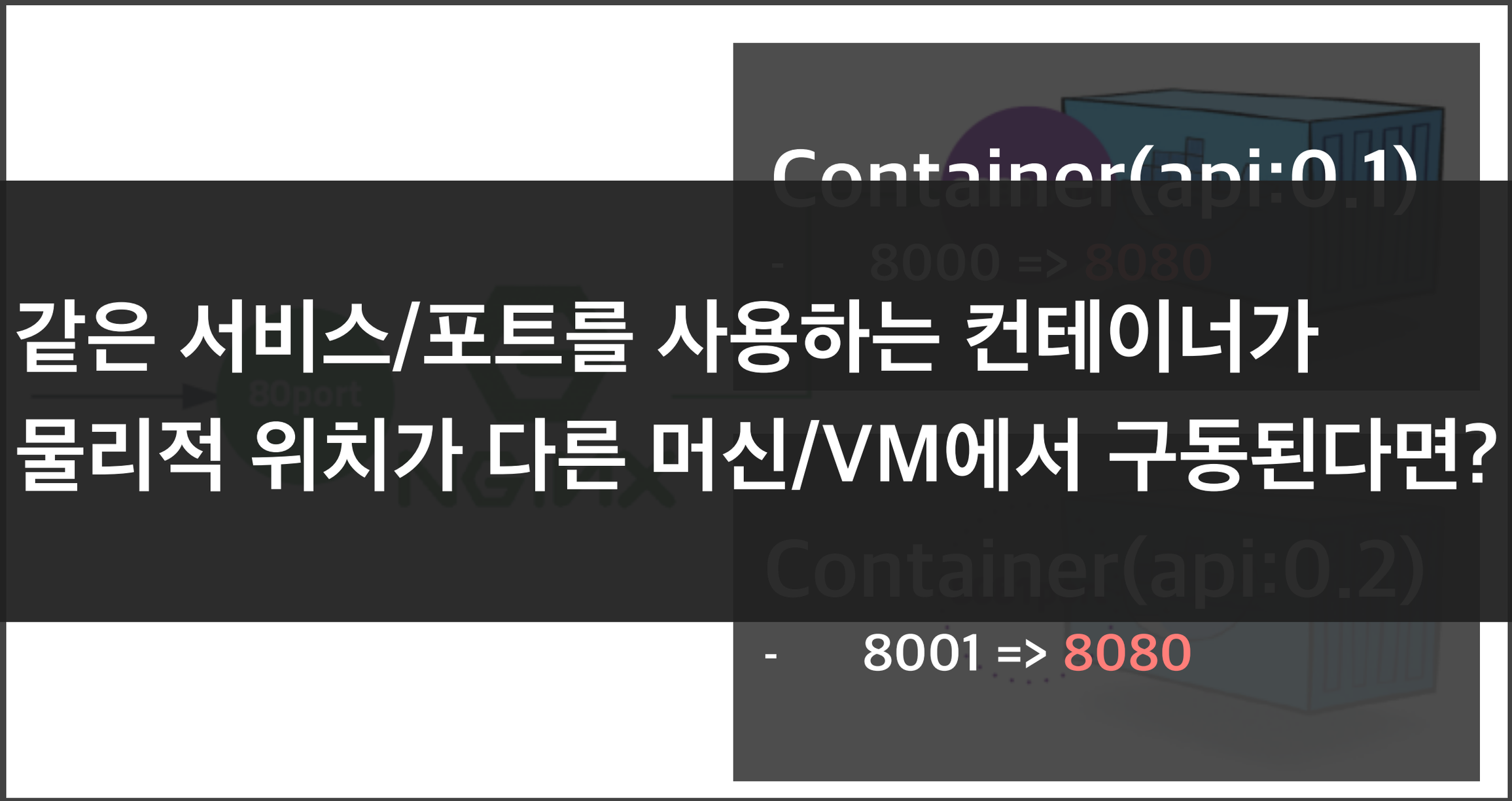
컨테이너 구성

컨테이너 구성



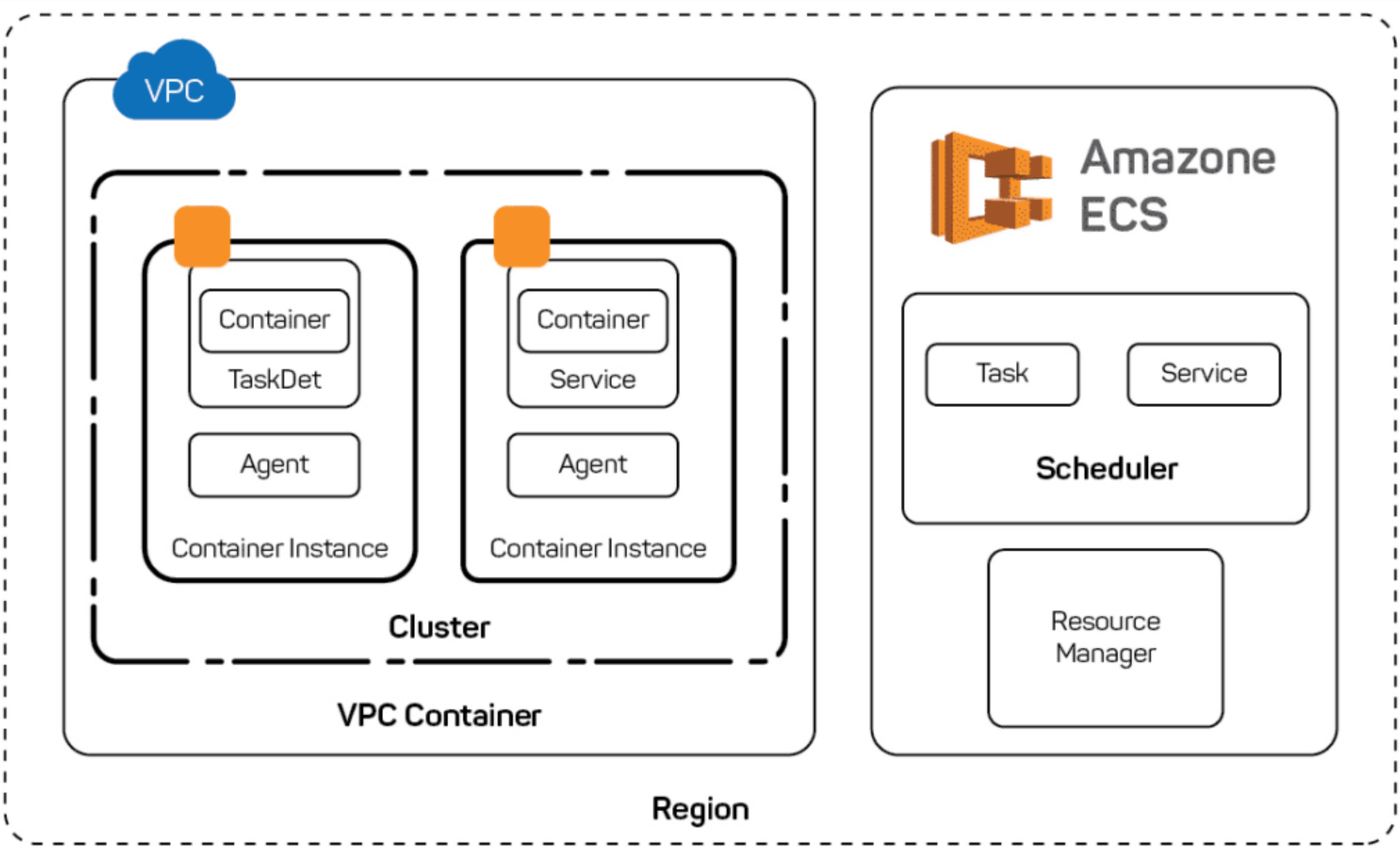
같은 포트를 사용하는 컨테이너가 N대라면?

컨테이너 구성



같은 서비스/포트를 사용하는 컨테이너가
물리적 위치가 다른 머신/VM에서 구동된다면?

Container orchestration - AWS ECS



클러스터링

- 여러 개의 서버를 하나의 서버처럼 사용. 클러스터에 새로운 서버를 추가할 수도 있고 제거 가능. 작게는 몇 개 안 되는 서버부터 많게는 수천 대의 서버를 하나의 클러스터로 만들 수 있음
- AWS ECS는 AWS EC2 인스턴스를 기반으로 클러스터링을 위한 서버로 구축되며 Auto scaling group을 통해 자동 확장, 제거 가능

스케쥴링

서비스 디스커버리

Container orchestration - AWS ECS

클러스터링

스케쥴링

- 컨테이너를 적당한 서버에 배포해 주는 작업입니다. 여러 대의 서버(클러스터링 된 여러 대의 EC2 인스턴스 중 하나) 중 가장 할일 없는 서버에 배포하거나 그냥 차례대로 배포 또는 아예 랜덤하게 배포하는 전략으로 실행 가능

서비스 디스커버리

Container orchestration - AWS ECS

클러스터링
스케쥴링
서비스 디스커버리

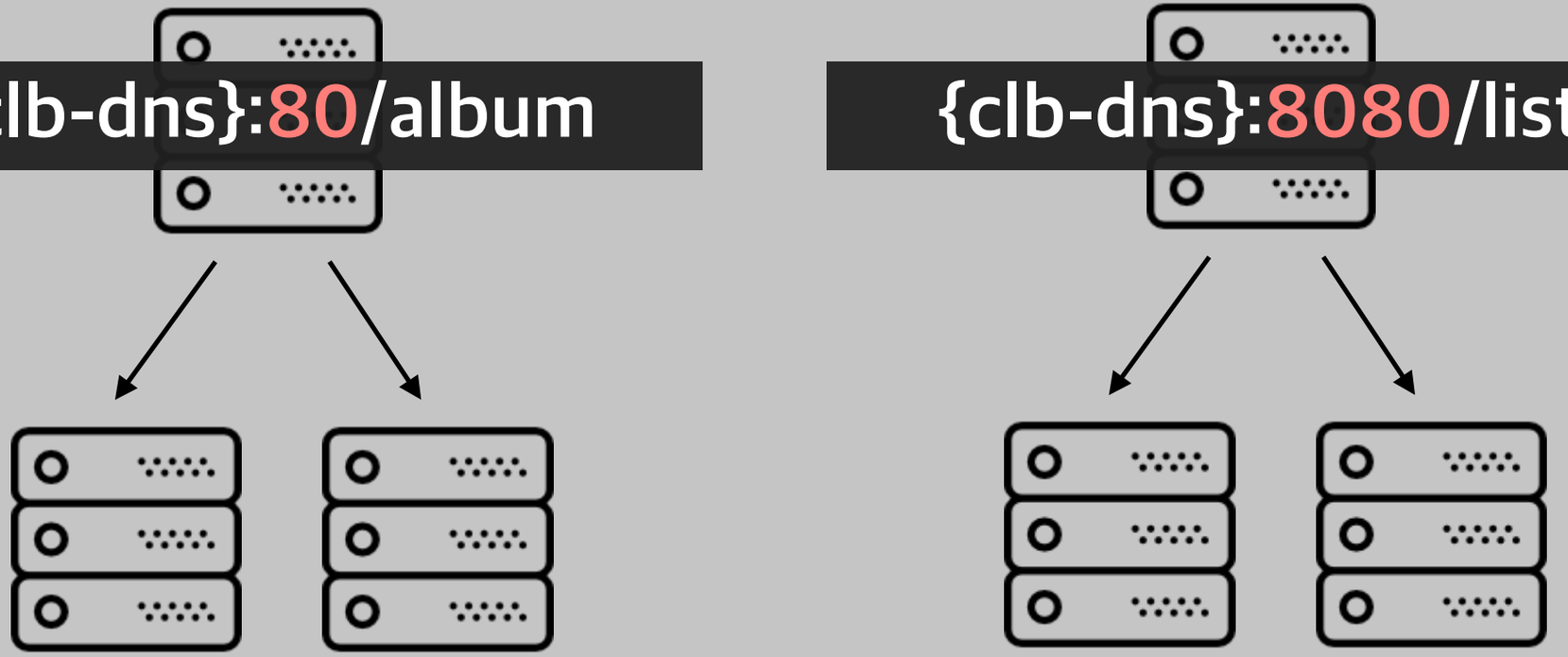
- 클러스터 환경에서 컨테이너는 어느 서버에 생성 될 지(=> 스케쥴링 알고리즘/정책에 기반) 알 수 없고 다른 서버로 이동할 수도 있음
- 여러 서비스로 배포된 컨테이너를 key/value 형태로 서비스를 찾아주는 기능
- 같은 기능의 서비스(컨테이너)가 같은 포트를 사용한다 해도 논리적 그룹을 찾아내어 요청 전달 가능

AWS ALB(Application load balancer)

Classic load balancer

{clb-dns}:80/album

{clb-dns}:8080/list

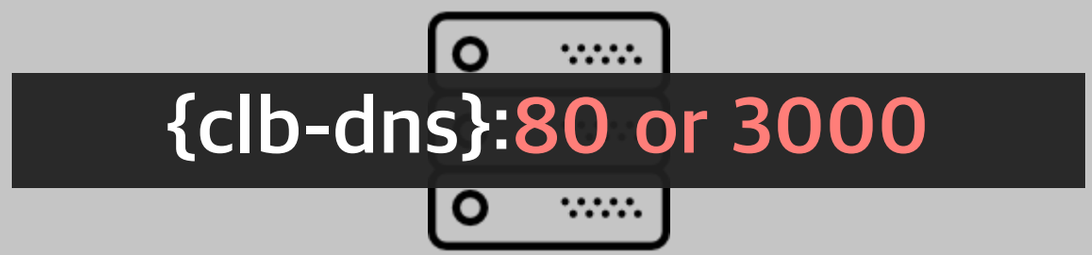


Web server(EC2)

WAS(EC2)

AWS ALB(Application load balancer)

Application load balancer



Target group(http:80)
=> web-container(0:80)

Target group(http:3000)
=> was-container(0:8080)

AWS ALB(Application load balancer)

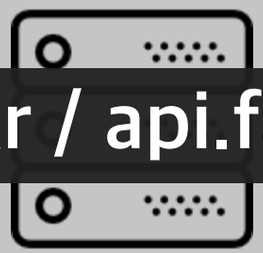
Application load balancer

Target group(http:80)
=> web-container(0:80)

Target group(http:3000)
=> was-container(0:8080)

AWS ALB(Application load balancer)

Application load balancer



fastcampus.co.kr / api.fastcampus.co.kr

fastcampus.co.kr

api.fastcampus.co.kr

Target group(http:80)
=> web-container(0:80)

Target group(http:3000)
=> was-container(0:8080)

Elastic Container Registry

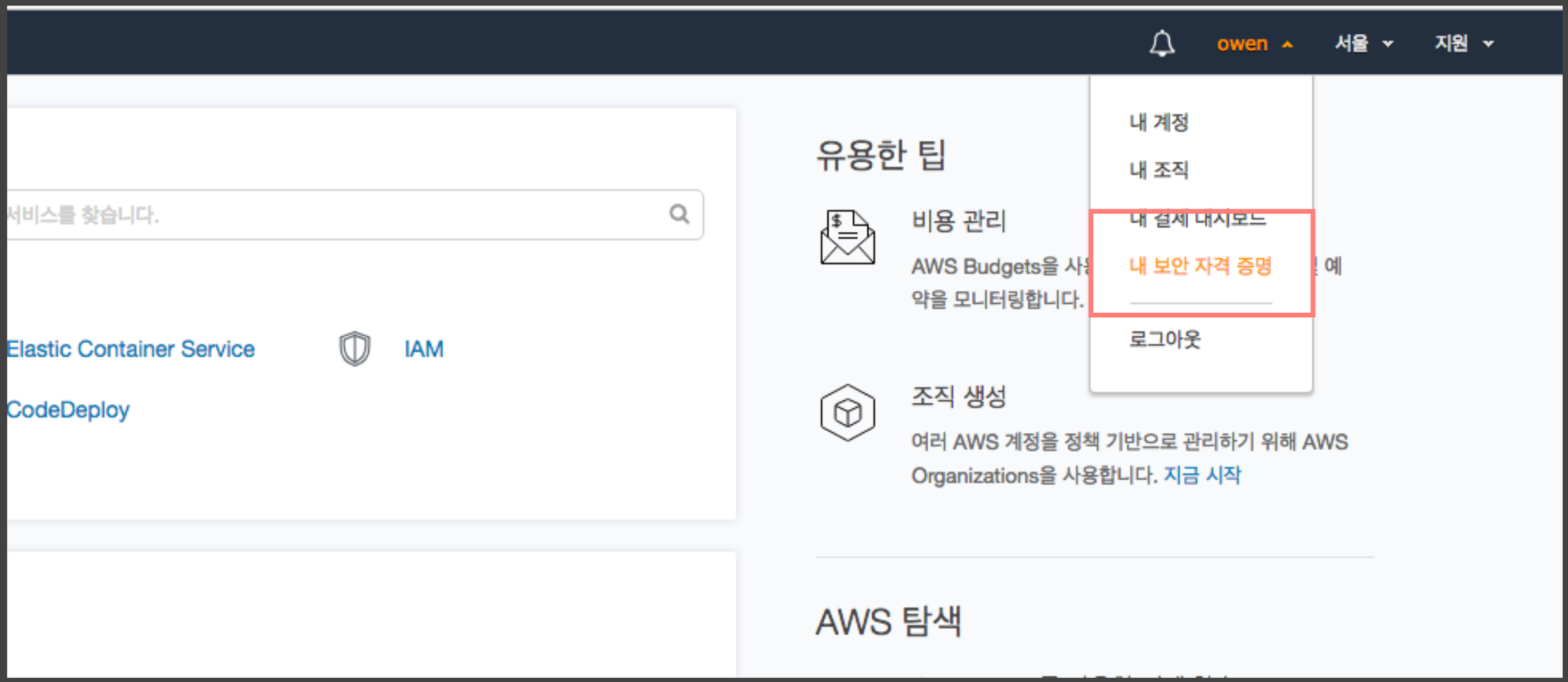
- Docker 컨테이너 이미지를 손쉽게 저장, 관리 및 배포할 수 있게 해주는 완전관리형 Docker 컨테이너 레지스트리
- AWS ECS(Elastic container service)와 결합하여 EC2 인스턴스에 컨테이너 배포까지 통합 지원

AWS CLI

AWS 명령줄 인터페이스

- bash shell처럼 명령어를 통해 AWS 서비스를 관리하는 통합 도구
- Mac os
  - sudo pip install awscli
- Windows : https://aws.amazon.com/ko/cli/ 에 접속하여 64/32bit 중 맞는 걸로 설치

AWS CLI를 사용하기 위해 액세스 키 발급



AWS CLI를 사용하기 위해 액세스 키 발급

보안 자격 증명

이 페이지를 사용하여 AWS 계정의 자격 증명을 관리합니다. AWS Identity and Access Management(IAM) 사용자의 자격 증명을 관리하려면 IAM 콘솔을 사용하십시오.

AWS 자격 증명 유형과 사용 방법에 대해 자세히 알아보려면 AWS 일반 참조의 AWS 보안 자격 증명을 참조하십시오.

+ 비밀번호

+ 멀티 팩터 인증(MFA)

− 액세스 키(액세스 키 ID 및 보안 액세스 키)

액세스 키를 사용하여 AWS 서비스에 대한 프로그래밍 방식의 요청에 서명합니다. 액세스 키를 사용하여 요청에 서명하는 방법을 알아보려면 서명 설명서를 참조하십시오. 보호를 위해 액세스 키를 안전하게 보관하고 다른 사람과 공유하지 마십시오. 또한 90일마다 액세스 키를 교체하는 것이 좋습니다.

참고: 한 번에 최대 두 개의 액세스 키(활성 또는 비활성)를 보유할 수 있습니다.

생성 완료	삭제됨	액세스 키 ID	마지막 사용	마지막으로 사용한 리전	마지막으로 사용한 서비스	상태	작업
8월 2일 2018		AKIAIMXMD3KGKSEDIALQ	2018-08-02 17:21 UTC+0900	ap-northeast-2	ecr	활성	비활성화 | 삭제
8월 2일 2018	8월 2일 2018	AKIAJ7ZLD24EJ4ICV56Q	해당 사항 없음	해당 사항 없음	해당 사항 없음	삭제됨	

새 액세스 키 만들기

중요한 변경 사항 - AWS 보안 액세스 키 관리

AWS CLI

1. ec2-user@ip-172-31-25-131:~ (bash)

a1@1ui-MacBook-Air:~$

cat rootkey.csv

- AWSAccessKeyId
- AWSSecretKey

aws configure

- aws cli 설정

AWS CLI

1. ec2-user@ip-172-31-25-131:~ (bash)

a1@1ui-MacBook-Air:~$

AWS Access Key ID [None]: {rootkey.csv -> AWSAccessKeyId}

AWS Secret Access Key [None]: {AWSSecretKey}

Default region name [ap-northeast-2]: ap-northeast-2

Default output format [json]: json

AWS ECR

AWS console > ECS 검색 > 리포지토리 탭 클릭 > 리포지토리 생성 선택

Elastic Container Registry 시작하기

단계 1: 리포지토리 구성

단계 2: Docker 이미지 빌드, 태그 지정 및 푸시

리포지토리 구성

이 마법사가 Elastic Container Registry에 리포지토리를 생성하는 절차를 단계별로 안내합니다. 자세히 알아보기

리포지토리 이름* web-repo

네임스페이스는 선택 사항이고, 슬래시를 사용하여 리포지토리 이름에 포함할 수 있습니다(예: namespace/repo)

리포지토리 URI 322749112518.dkr.ecr.ap-northeast-2.amazonaws.com/web-repo

권한

기본적으로 리포지토리 소유자인 사용자만 이 리포지토리에 액세스할 수 있습니다. 이 마법사를 완료한 후 다른 사용자에게 콘솔에서 이 리포지토리에 액세스할 수 있는 권한을 부여할 수 있습니다.

*필수

취소 다음 단계

AWS CLI

- 로컬에서 작동하는 Docker client 인증

ecr get-login 명령어를 통해 나온 Docker client 인증 정보를 입력하여 인증 진행

AWS CLI

1. ec2-user@ip-172-31-25-131:~ (bash)

a1@1ui-MacBook-Air:~$

docker tag web:0.2 {ecr-web-repo-url}:latest

- 방금 생성된 web-repo 이름을 가진 ECR에 push를 위해 태그 설정

docker push {ecr-web-repo-url}:latest

- ECR(web-repo)에 web:0.2 이미지 push

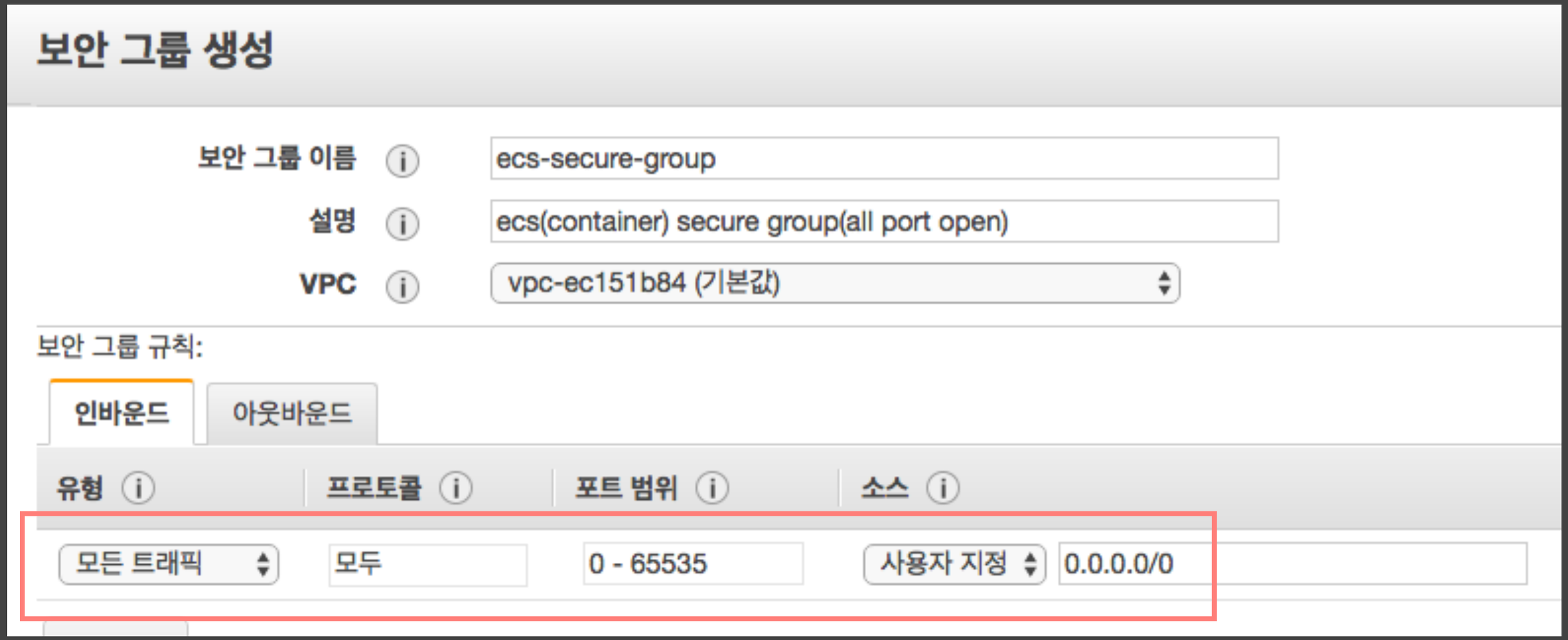
클러스터

- AWS EC2 instance를 하나의 서버처럼 구성하기 위해 클러스터링 진행
- 몇 대의 인스턴스를 생성할 것인지 (Auto scaling group 적용 가능) 결정. 지시한 인스턴스 별로 ECS-agent 설치
  - 컨테이너 배포 스케쥴링 및 서비스 디스커버리 실행

작업 정의

- 애플리케이션을 구성하는 컨테이너를 설명하는 JSON 형식의 텍스트 파일(최대 10개)
- 어떤 Docker image를 기반으로 구성할 것인지
- 컨테이너에서 사용할 포트 및 네트워크 설정
- 컨테이너가 사용할 서버 리소스(CPU, RAM 등) 설정

AWS ECS - 클러스터



ECS 클러스터에 구성될 EC2 보안그룹 생성

- ECS 클러스터는 여러 대의 EC2가 클러스터링 되어 있으며, 스케쥴링에 의해 컨테이너가 구성됩니다.
- 스케쥴러가 임의로 컨테이너의 포트를 부여(ex. 35531 -> 8080)하기에 각 EC2의 보안 그룹엔 모든 포트를 허용합니다.
- 추후에 작업 정의를 통해 사용할 컨테이너 포트의 대역대를 설정할 수 있습니다.

AWS ECS - 작업 정의

작업 정의 생성

작업 정의는 작업에 포함할 컨테이너와 그 컨테이너들이 상호 작용하는 방식을 지정합니다. 컨테이너가 사용할 데이터 볼륨을 지정할 수도 있습니다. 자세

To learn about which parameters are supported for Windows Containers, please see the ECS Documentation.,

작업 정의 이름*: web-task

작업 역할: 없음

인증된 AWS 서비스에 API 요청을 할 때 작업이 사용할 수 있는 IAM 역할 옵션입니다. IAM 콘솔에서 Amazon Elastic Container Service 작업 역할을 생성합니다.

네트워크 모드: 브리지

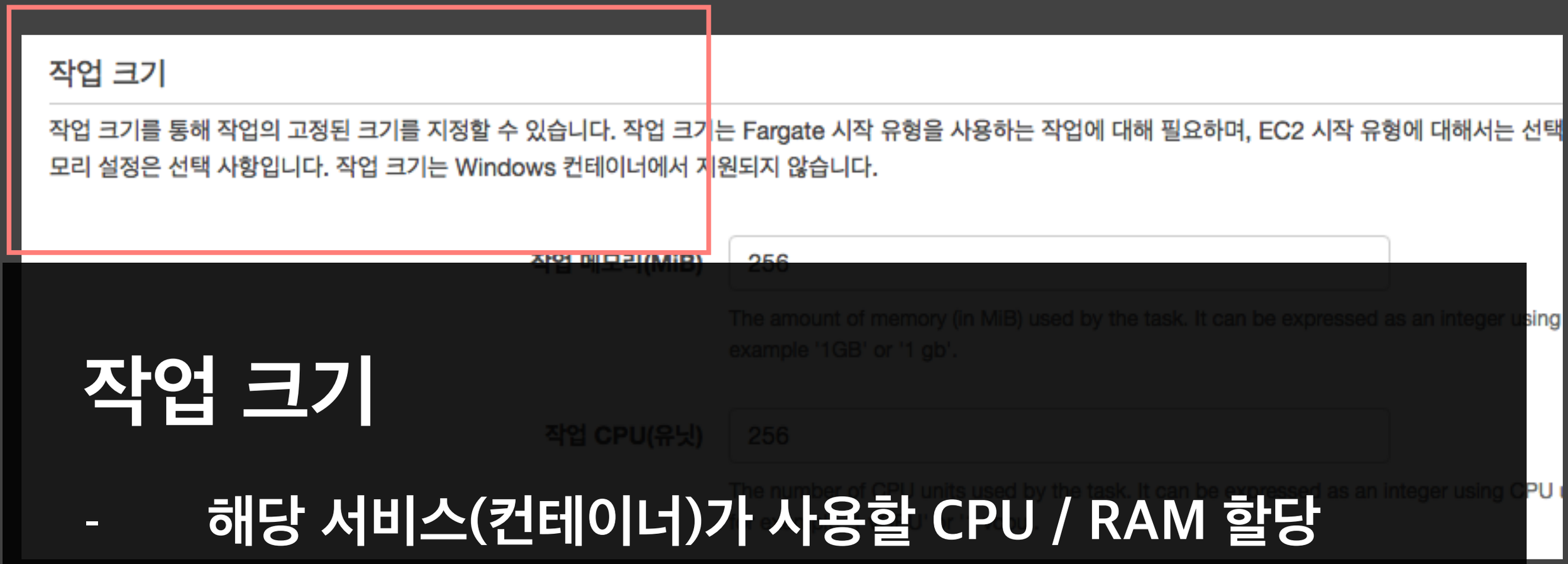
<default>을(를) 선택할 경우 ECS는 Docker의 기본 네트워킹 모드를 사용하여 컨테이너를 시작합니다. Docker의 기본 네트워킹 모드는 Windows의 Linux 브리지(Bridge) 및 NAT 브리지(Bridge)입니다. <default>은(는) Windows에서 유일하게 지원되는 모드입니다.

AWS ECS - 작업 정의

Docker(ECS) 네트워크 모드

- 브리지 : docker0 bridge 모드, dynamic port mapping
- 호스트 : Docker daemon이 운영되는 Host 머신의 네트워크 인터페이스를 그대로 사용
- aws vpc : EC2 instance에 적용하는 탄력적 네트워크 인터페이스를 컨테이너에 적용(elastic dns / ip)

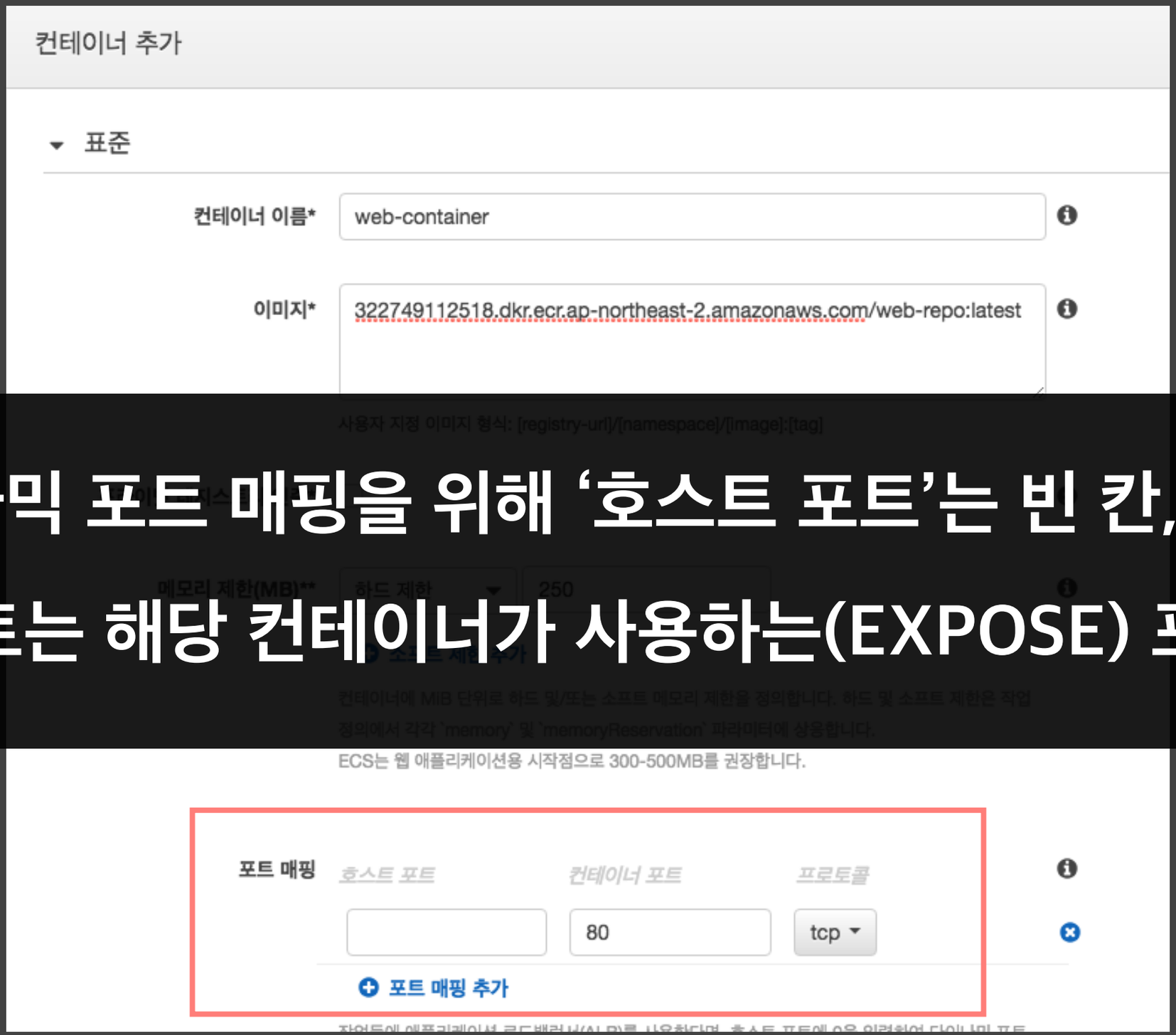
AWS ECS - 작업 정의



작업 크기

- 해당 서비스(컨테이너)가 사용할 CPU / RAM 할당
- VM보다 유동적이고 효율적으로 제공 가능
- 컨테이너 안에 구동되는 프로세스들에 맞춰 최적화 작업

AWS ECS - 작업 정의



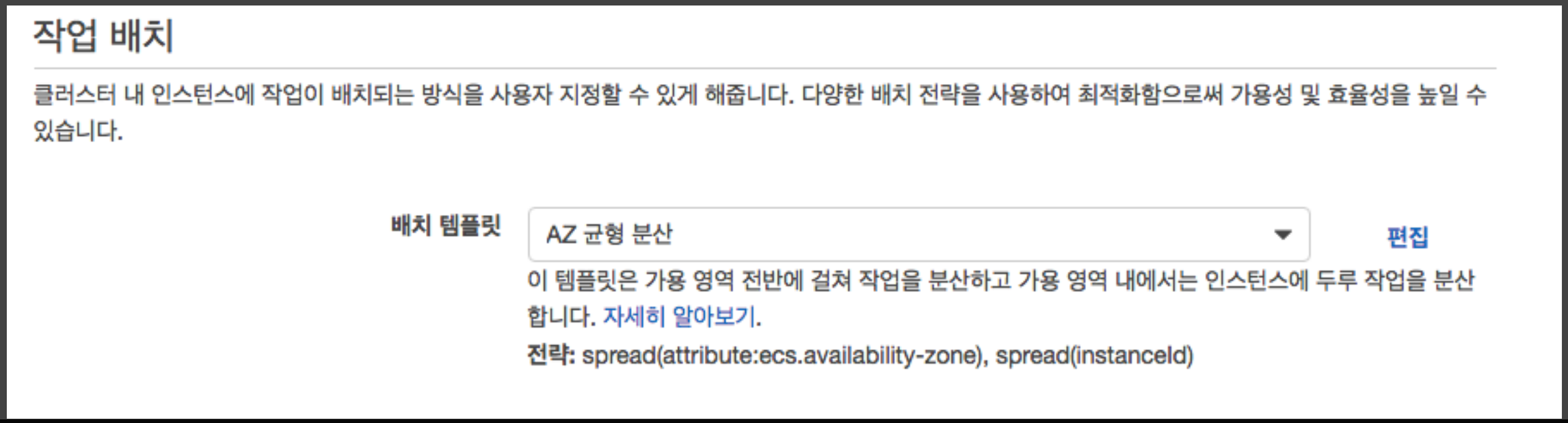
다이나믹 포트 매핑을 위해 '호스트 포트'는 빈 칸, 컨테이너 포트는 해당 컨테이너가 사용하는(EXPOSE) 포트 입력

AWS ECS - 서비스 생성

서비스 유형

- REPLICA : 부하 분산을 위해 해당 갯수만큼의 운영이 가능한 컨테이너 생성
- DAEMON : 고가용성을 위해 예비 컨테이너의 생성
- RDS의 multi-AZ와 read replica의 차이와 동일합니다

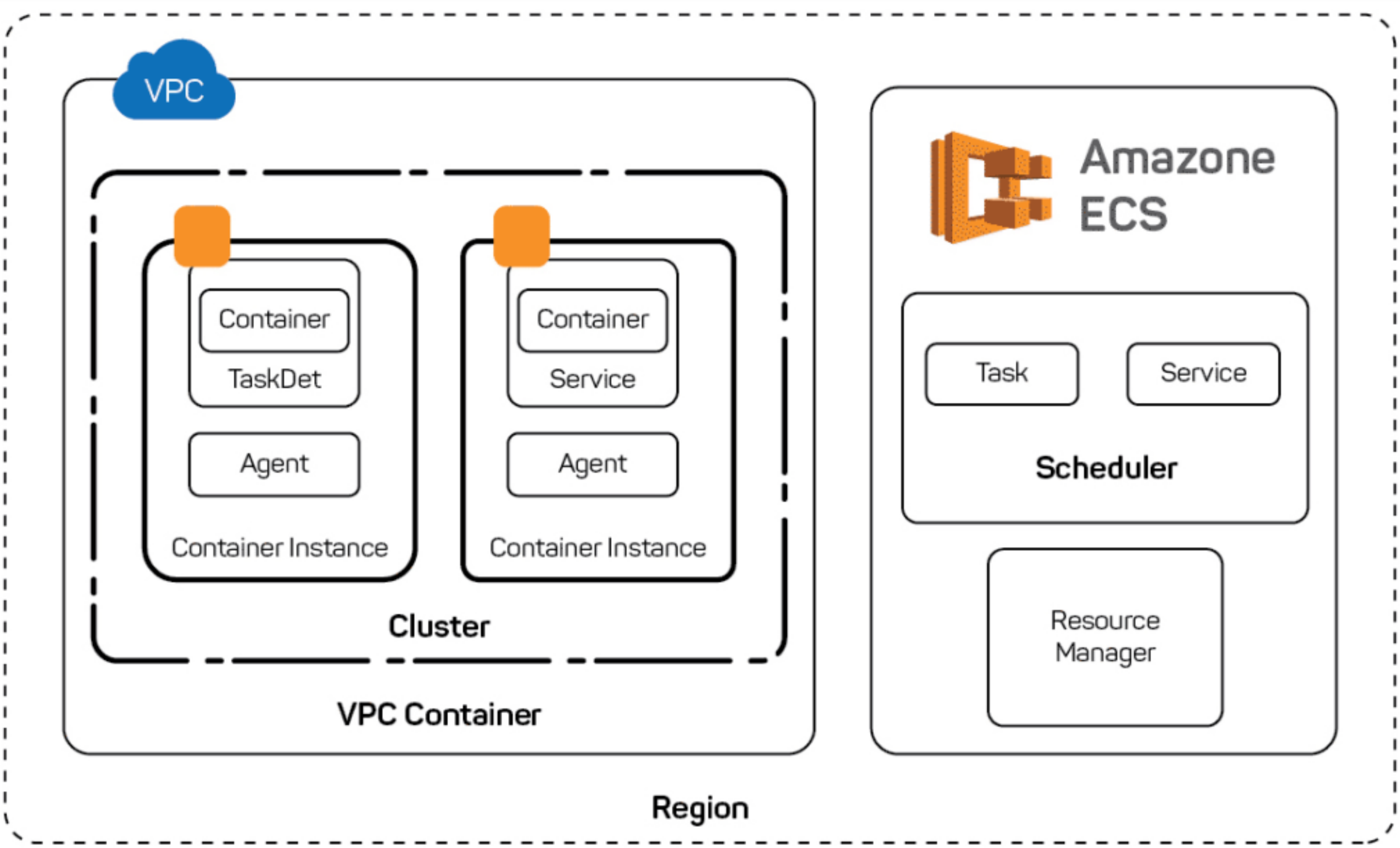
AWS ECS - 서비스 생성



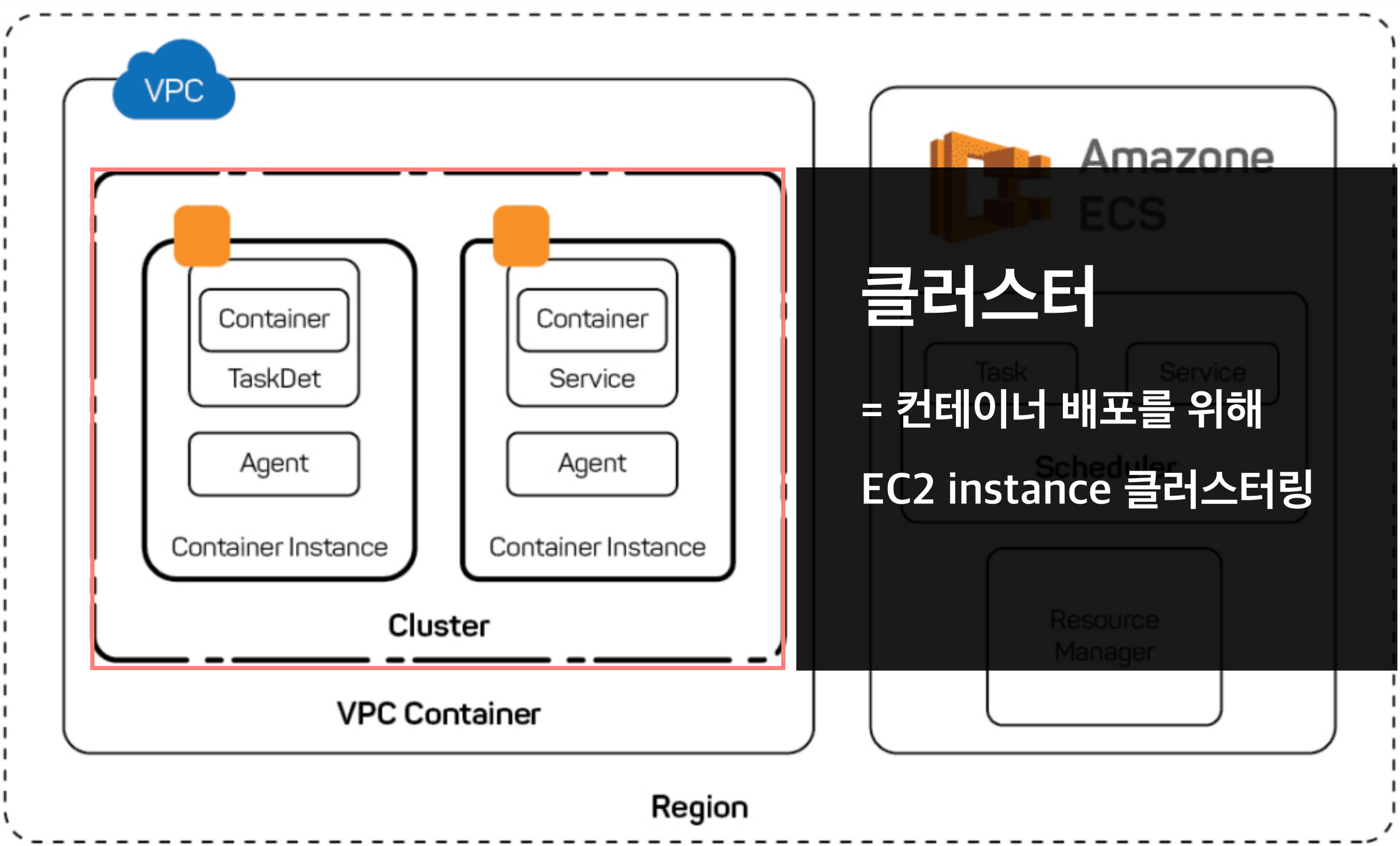
작업 배치

- AZ 균형 분산 : 클러스터링 된 EC2 인스턴스에 고르게 배포
- AZ 균형 빈팩 : 클러스터링 된 EC2 인스턴스 중 가장 적은 컨테이너 갯수와 메모리 사용량을 종합적으로 고려하여 인스턴스에 배포
- 호스팅 당 작업 한 개 : 하나의 인스턴스에 하나의 서비스(컨테이너) 배포
- 빈팩 : 가장 적은 메모리를 사용하는 인스턴스에 배포

Container orchestration - AWS ECS



Container orchestration - AWS ECS



클러스터

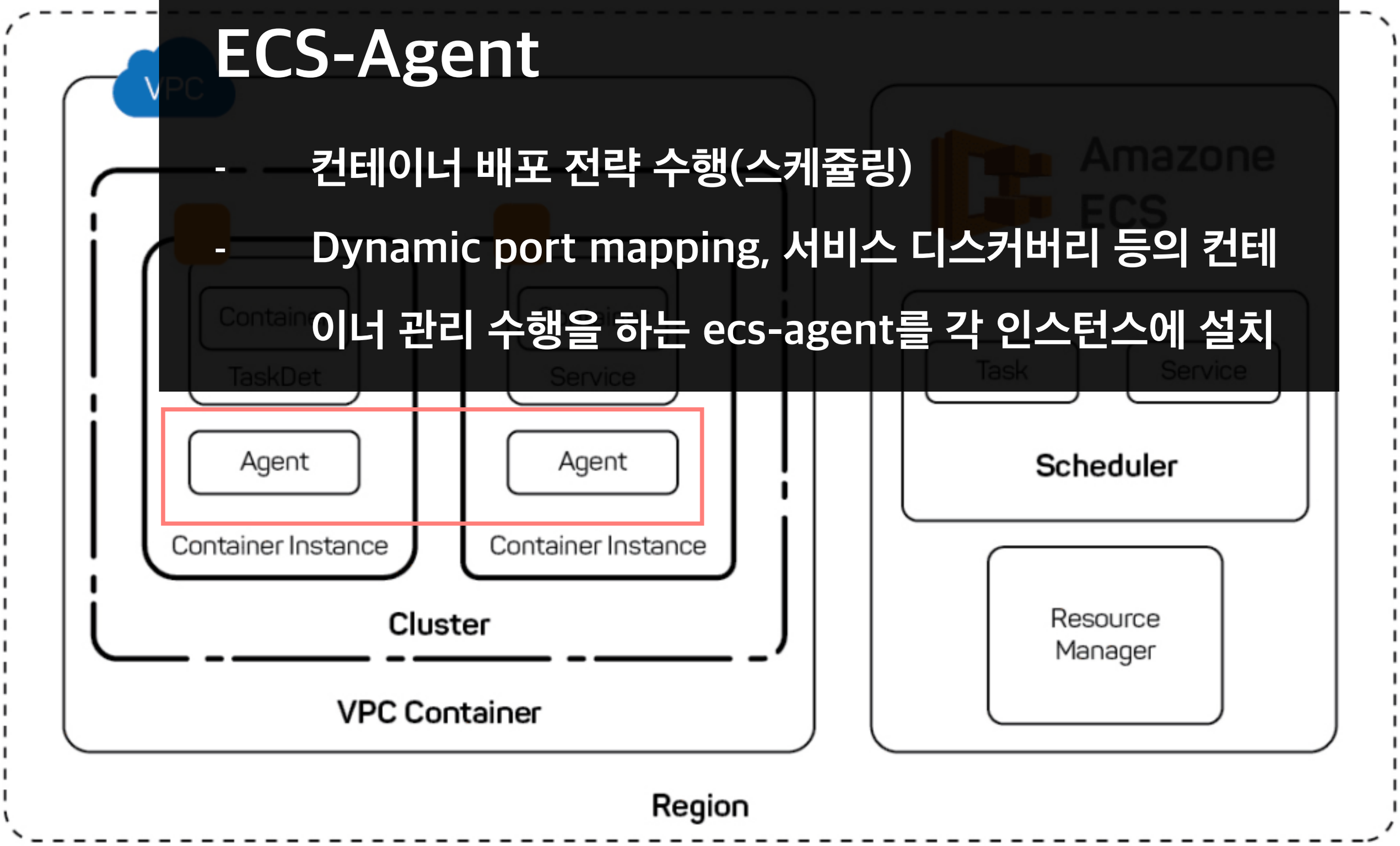
= 컨테이너 배포를 위해

EC2 instance 클러스터링

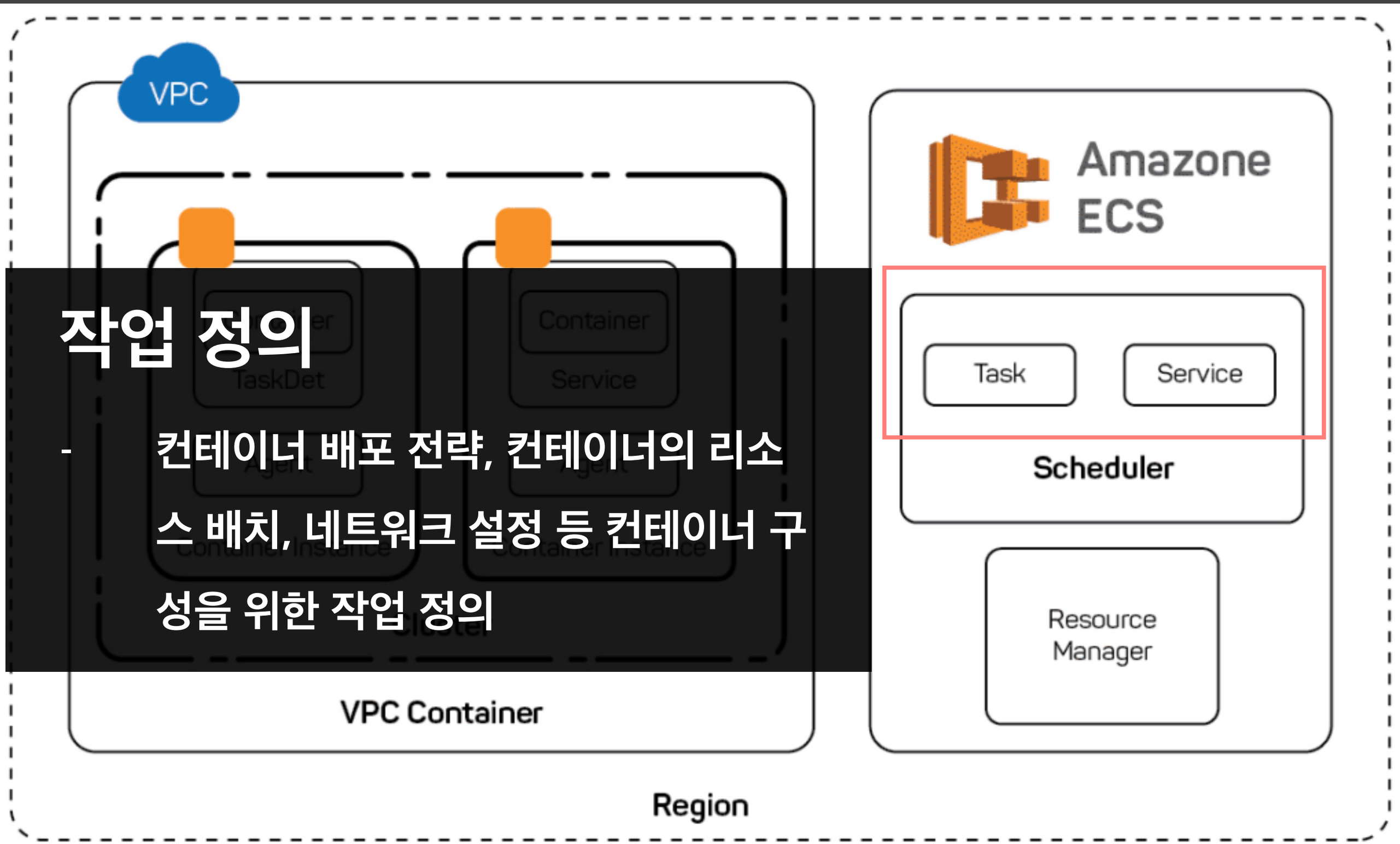
Container orchestration - AWS ECS

ECS-Agent

- 컨테이너 배포 전략 수행(스케쥴링)
- Dynamic port mapping, 서비스 디스커버리 등의 컨테이너 관리 수행을 하는 ecs-agent를 각 인스턴스에 설치



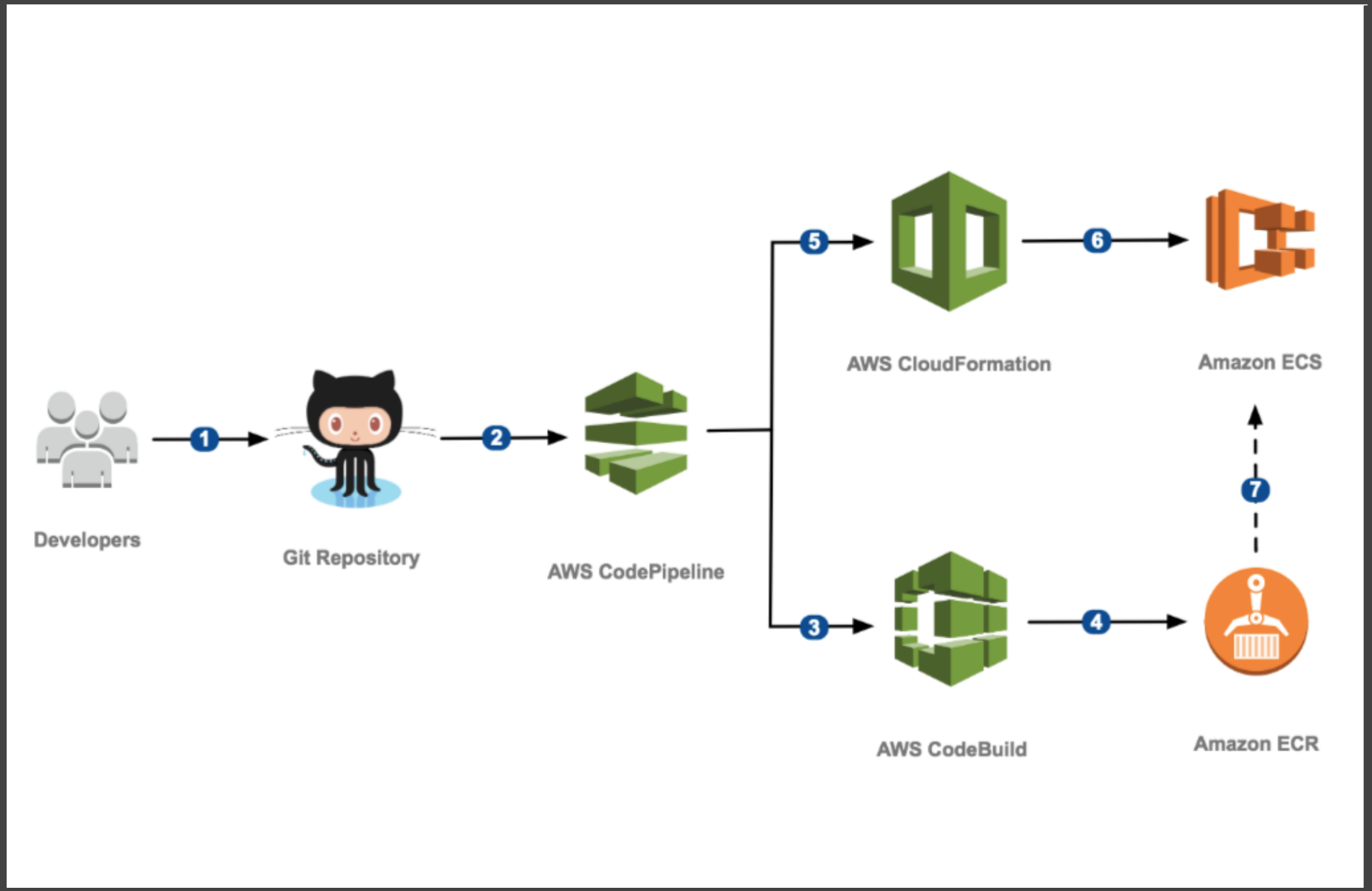
Container orchestration - AWS ECS



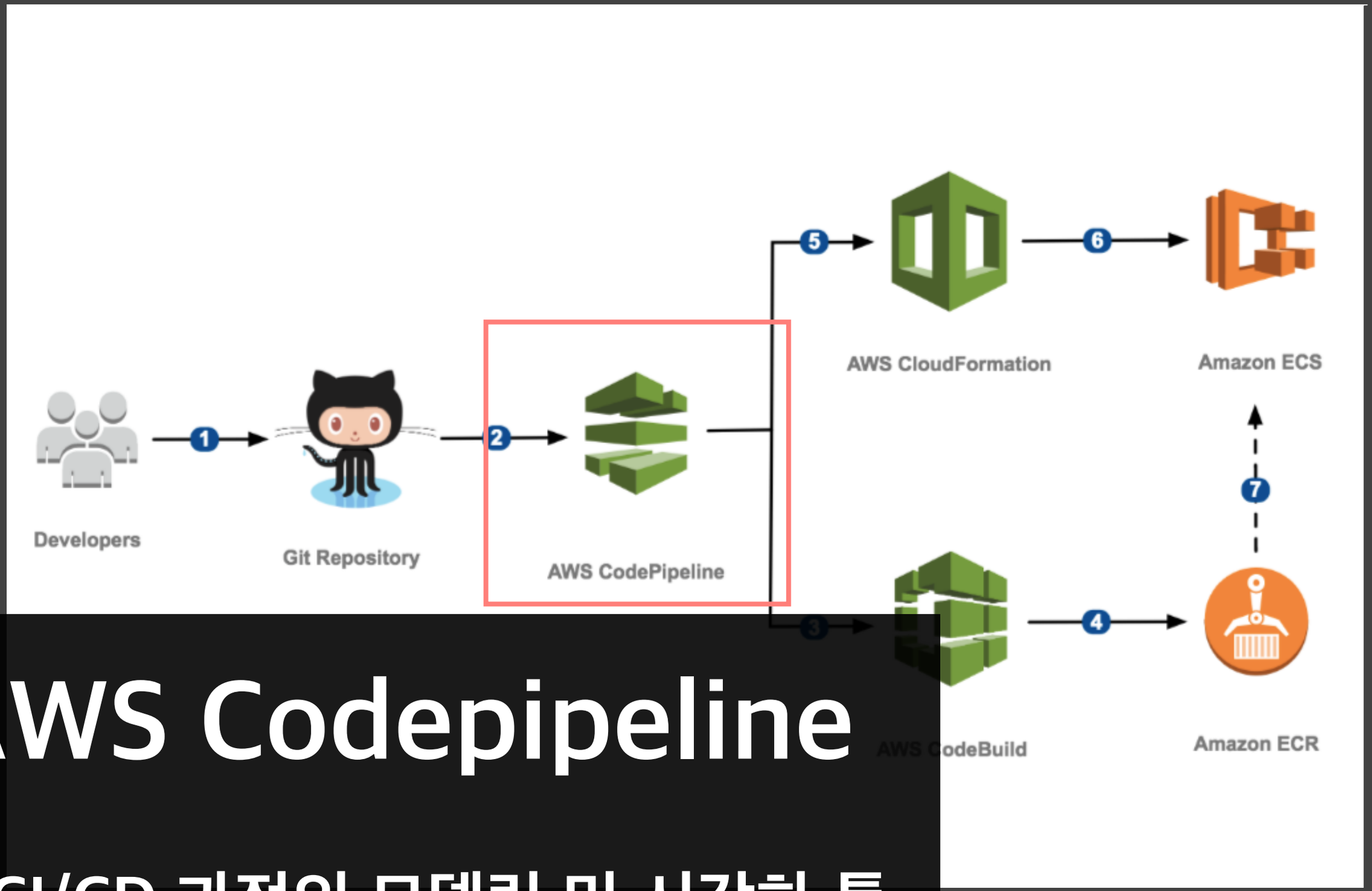
작업 정의

- 컨테이너 배포 전략, 컨테이너의 리소스 배치, 네트워크 설정 등 컨테이너 구성을 위한 작업 정의

AWS Codepipeline + CodeBuild + CodeDeploy



AWS Codepipeline + CodeBuild + CodeDeploy



AWS Codepipeline

= CI/CD 과정의 모델링 및 시각화 툴

AWS CodeBuild

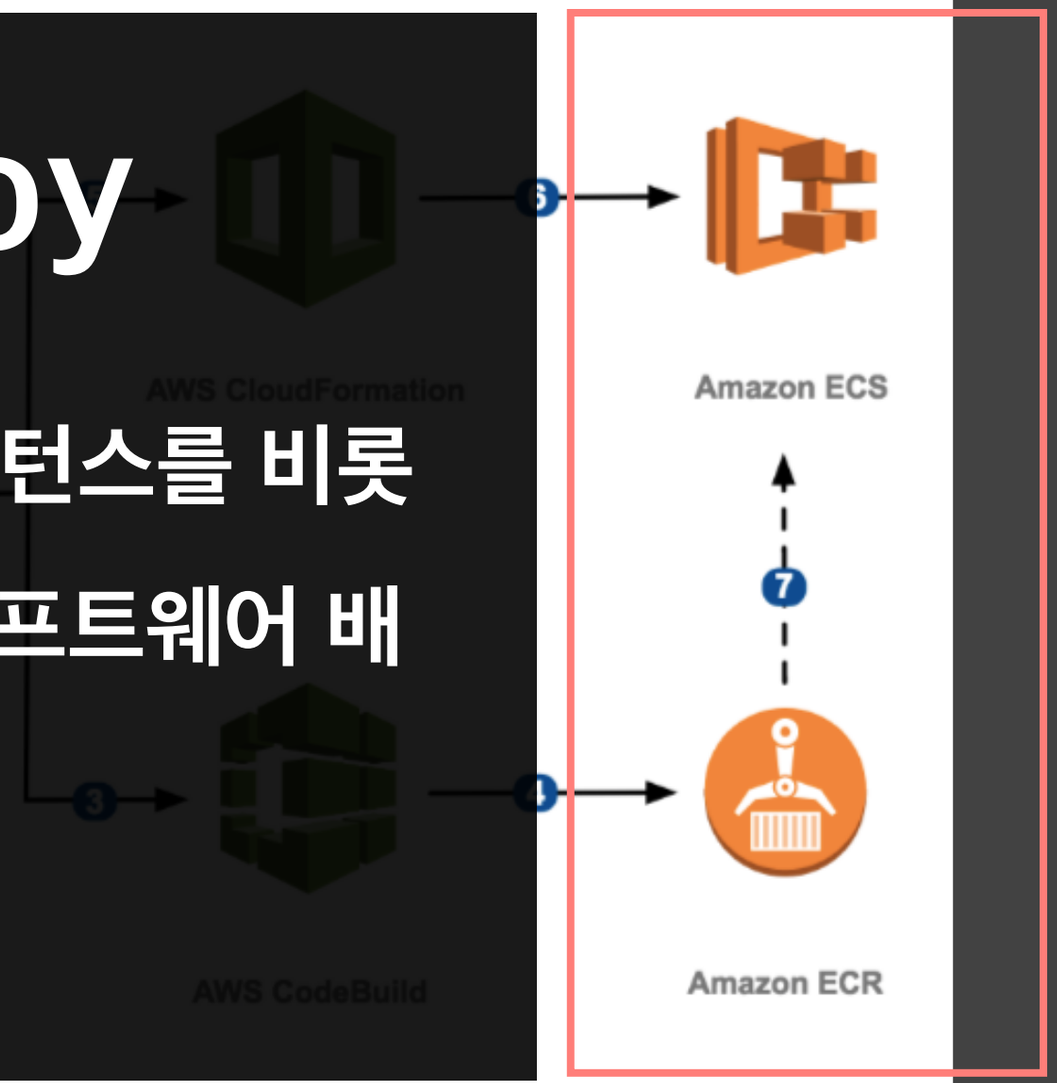
= 완전관리형 빌드 서비스,

코드를 컴파일하고, 테스트하며, 패키징하기 위한 빌드 스크립트를 실행

AWS Codepipeline + CodeBuild + CodeDeploy

AWS CodeDeploy

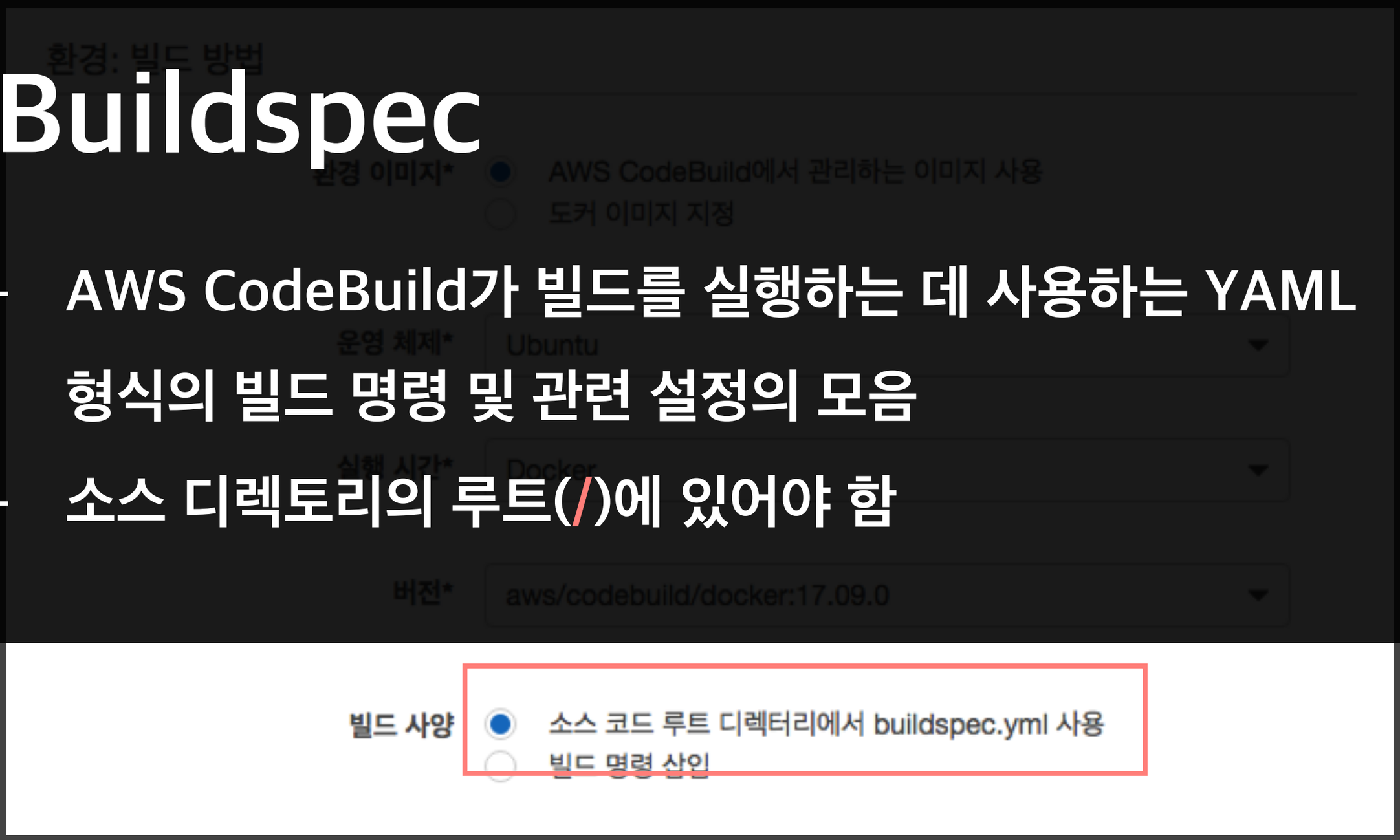
= Amazon EC2, Lambda 등 인스턴스를 비롯한 다양한 컴퓨팅 서비스에 대한 소프트웨어 배포를 자동화하는 서비스



buildspec.yml

Buildspec

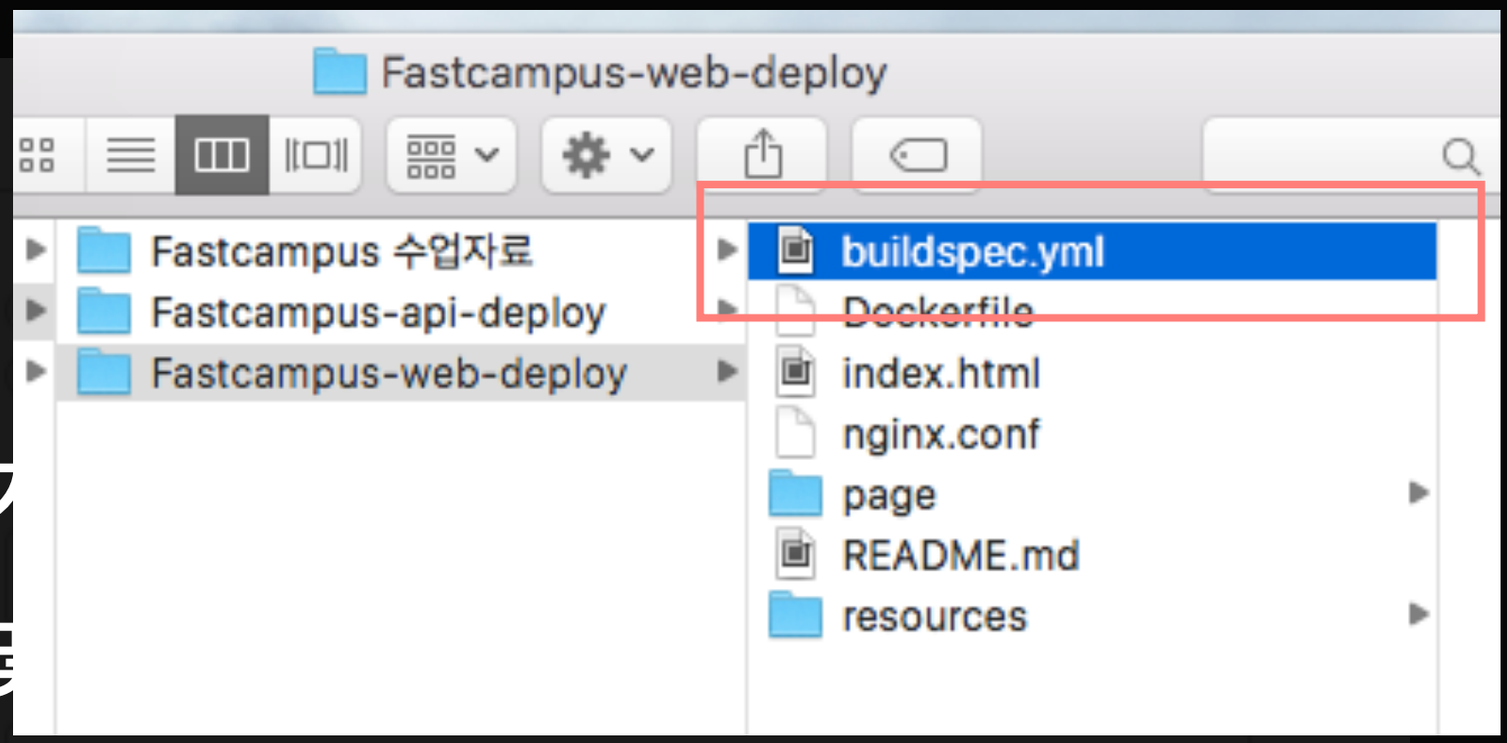
- AWS CodeBuild가 빌드를 실행하는 데 사용하는 YAML 형식의 빌드 명령 및 관련 설정의 모음
- 소스 디렉토리의 루트(/)에 있어야 함

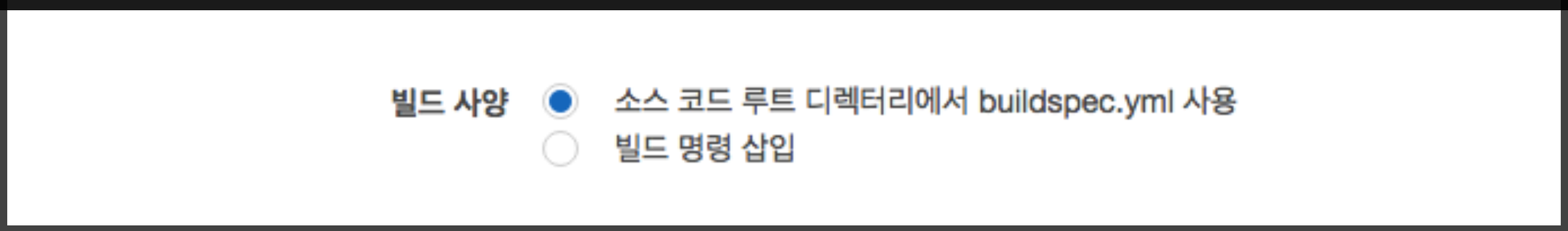


buildspec.yml

Buildspec

- AWS CodeBuild가 형식의 빌드 명령 및
- 소스 디렉토리의 루트(/)에 있어야 함





buildspec.yml

```
version: 0.2

phases:
  pre_build:
    commands:
      - echo Logging in to Amazon ECR...
      - aws --version
      - $(aws ecr get-login --region $AWS_DEFAULT_REGION --no-include-email)
      - REPOSITORY_URI=322749112518.dkr.ecr.ap-northeast-2.amazonaws.com/web-repo
      - IMAGE_TAG=$(echo $CODEBUILD_RESOLVED_SOURCE_VERSION | cut -c 1-7)
  build:
    commands:
      - echo Build started on `date`
      - echo Building the Docker image...
      - docker build -t $REPOSITORY_URI:latest .
      - docker tag $REPOSITORY_URI:latest $REPOSITORY_URI:$IMAGE_TAG
  post_build:
    commands:
      - echo Build completed on `date`
      - echo Pushing the Docker images...
      - docker push $REPOSITORY_URI:latest
      - docker push $REPOSITORY_URI:$IMAGE_TAG
      - echo Writing image definitions file...
      - printf '[{"name":"web-container","imageUri":"%s"}]' $REPOSITORY_URI:$IMAGE_TAG > MyFilename.json
artifacts:
    files: MyFilename.json
```

buildspec.yml

```
version: 0.2

phases:
  pre_build:
    commands:
      - echo Logging in to Amazon ECR...
      - aws --version
      - $(aws ecr get-login --region $AWS_DEFAULT_REGION --no-include-email)
      - REPOSITORY_URI=322749112518.dkr.ecr.ap-northeast-2.amazonaws.com/web-repo
      - IMAGE_TAG=$(echo $CODEBUILD_RESOLVED_SOURCE_VERSION | cut -c 1-7)
  build:
    commands:
      - echo Build started on `date`
      - echo Building the Docker image...
      - docker build -t $REPOSITORY_URI:latest .
      - docker tag $REPOSITORY_URI:latest $REPOSITORY_URI:$IMAGE_TAG
  post_build:
    commands:
      - echo Build completed on `date`
      - echo Pushing the Docker images...
      - docker push $REPOSITORY_URI:latest
      - docker push $REPOSITORY_URI:$IMAGE_TAG
      - echo Writing image definitions file...
      - printf '[{"name":"web-container","imageUri":"%s"}]' $REPOSITORY_URI:$IMAGE_TAG > MyFilename.json
artifacts:
  files: MyFilename.json
```

pre_build

- 빌드 수행 하기 전 설정 파일 적용 등의 역할 수행
- 현 실습은 Docker image 빌드 과정으로 AWS ECR 인증, 이미지 태그 설정 등의 작업 수행

buildspec.yml

```
version: 0.2

phases:
  build:
    commands:
      - echo Build started on `date`
      - echo Building the Docker image...
      - docker build -t $REPOSITORY_URI:latest .
      - docker tag $REPOSITORY_URI:latest $REPOSITORY_URI:$IMAGE_TAG
  post_build:
    commands:
      - echo Build completed on `date`
      - echo Pushing the Docker images...
      - docker push $REPOSITORY_URI:latest
      - docker push $REPOSITORY_URI:$IMAGE_TAG
      - echo Writing image definitions file...
      - printf '[{"name":"container_name","imageUri":"%s"}]' $REPOSITORY_URI:$IMAGE_TAG > MyFilename.json
artifacts:
    files: MyFilename.json
```

build

- 빌드 단계 수행(ex. docker build -t …)
- Docker build를 수행하는 명령어가 적힌 스크립트가 작성되있지만 Maven build 등 기타 Build tool 사용 가능

buildspec.yml

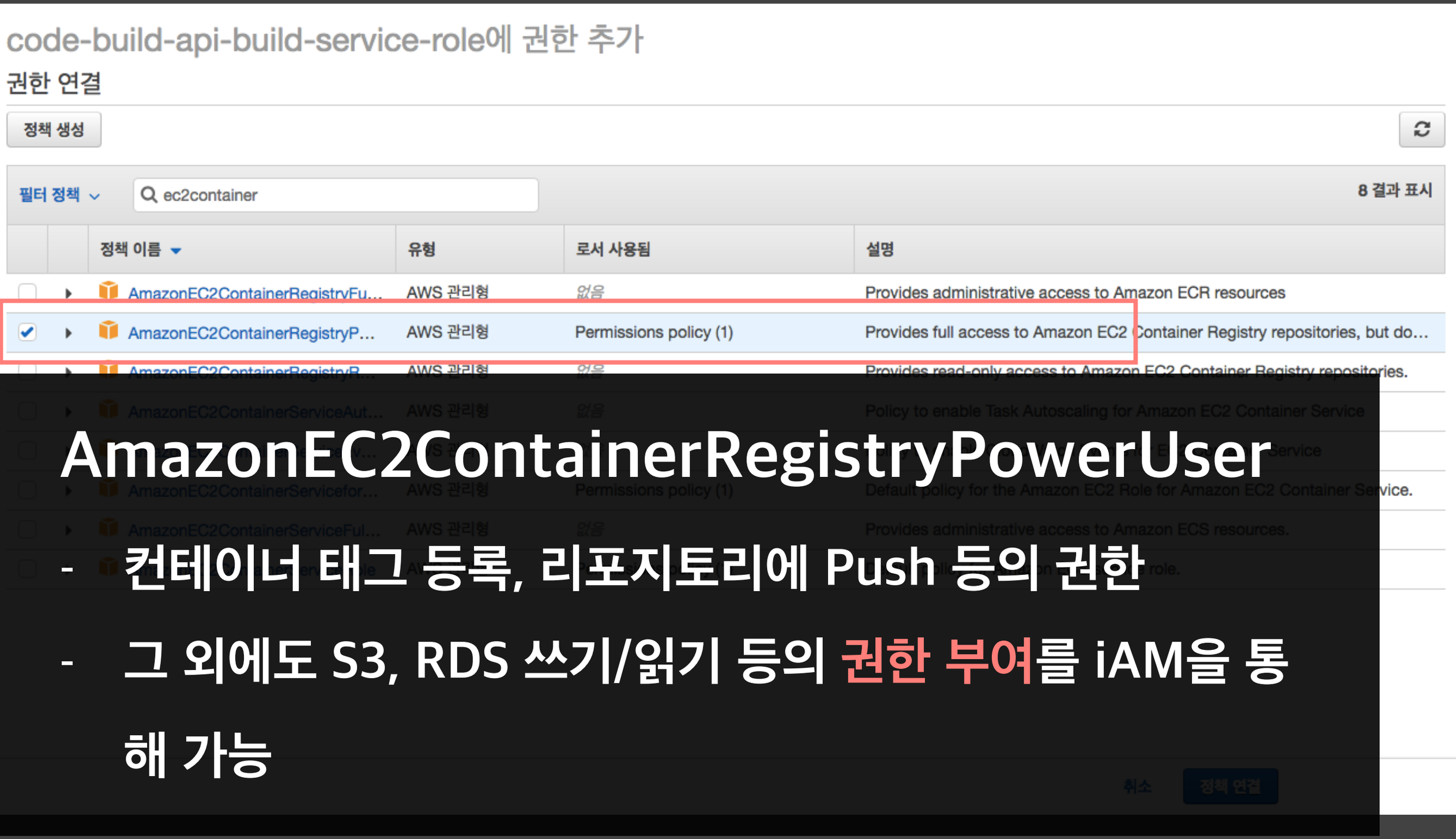
```
version: 0.2

phases:
  post_build:
    commands:
      - echo Build completed on `date`
      - echo Pushing the Docker images...
      - docker push $REPOSITORY_URI:latest
      - docker push $REPOSITORY_URI:$IMAGE_TAG
      - echo Writing image definitions file...
      - printf '[{"name":"web-container","imageUri":"%s"}]' $REPOSITORY_URI:$IMAGE_TAG > MyFilename.json
artifacts:
    files: MyFilename.json
```

post_build

- 빌드가 끝난 후 진행해야하는 작업 수행
- ECR로 Push하여 최신 버전의 Docker image(:latest)를 기반으로 하는 ECS 작업 정의 구성 실행

CodeBuild iAM



AmazonEC2ContainerRegistryPowerUser

- 컨테이너 태그 등록, 리포지토리에 Push 등의 권한
- 그 외에도 S3, RDS 쓰기/읽기 등의 권한 부여를 iAM을 통해 가능

배포 정의

어떤 컨테이너에 배포할 것인지 어떤 리포지토리(Docker)를 기반으로 할 것인지에 대한 json 파일 명칭

buildspec.yml

```
post_build:
    commands:
      - printf '[{"name":"web-
container","imageUri":"%s"}]' $REPOSITORY_URI:
$IMAGE_TAG > MyFilename.json
```

빌드 후 단계(post_build)

- 컨테이너 이름, 리포지토리 URL:TAG의 정보를 *.json 파일에 저장