

# AWS ECR, ECS, ALB 설정

## AWS ECR(Docker repository) 설정 및 Docker image push

1. AWS Management Console 접속 > AWS 서비스 탭에서 **Elastic Container Service** 검색하여 클릭 > 왼쪽 탭에서 리포지토리 선택 후 **리포지토리 생성** 버튼 클릭

### 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\*

api-repo

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

리포지토리 URI 322749112518.dkr.ecr.ap-northeast-2.amazonaws.com/api-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](#)

2. 리포지토리 이름(**api-repo**) 입력 > **다음 단계** 버튼 클릭

```
aws ecr get-login --no-include-email --region ap-northeast-2 # 로컬 환경의
docker repository AWS ECR 인증 진행 > 인증이 진행 후 나오는 docker login ~ 복사 후
명령어 실행
docker tag api:0.1 322749112518.dkr.ecr.ap-northeast-2.amazonaws.com/api-
repo:latest # 도커 이미지(api:0.1)에 태그 부여, 본인 ECR(전 단계에서 만든 api-repo)
{ECR-URL:latest} 입력
docker push 322749112518.dkr.ecr.ap-northeast-2.amazonaws.com/api-
repo:latest # {ECR-URL}:latest 이미지를 ECR로 Push 진행
```

## AWS ECS 설정

### AWS ECS 클러스터 생성

해당 내용은 현재 과정에서 따로 진행하지 않습니다. 나중에 확인하실 참고용 자료로 남겨두며 앞으로 할 컨테이너 배포는 기존에 생성된 클러스터에서 진행하시면 됩니다.



1. Elastic Container Service > 클러스터 탭 클릭 > 클러스터 생성 버튼 클릭 > EC2 Linux + 네트워크 선택 > 다음 단계 클릭

## 클러스터 구성

---

클러스터 이름\*

ecs-cluster


☐ 빈 클러스터 생성

## 인스턴스 구성

---

프로비저닝 모델 ☒ 온디맨드 인스턴스

온디맨드 인스턴스를 사용하면 장기 약정이나 선결제 금액 없이 시간당 컴퓨팅 파워에 대한 요금을 지불할 수 있습니다.

☐ 스팟

Amazon EC2 스팟 인스턴스를 사용하면 온디맨드 가격에서 최대 90% 할인된 금액으로 예비 Amazon EC2 컴퓨팅 용량에 입찰할 수 있습니다. [자세히 알아보기](#)

EC2 인스턴스 유형\*

t2.micro


☐ Manually enter desired instance type

T2 무제한 활성화



인스턴스 개수\*

4



EC2 AMI ID\*

amzn-ami-2018.03.d-amazon-ecs-optimized [ami-02b0706448bd6fb5e]



### 2. 클러스터 - 구성 설정

- 클러스터 이름(ecs-cluster) 입력
- EC2 인스턴스 유형(t2.micro) 선택
  - 실습을 위해 요금이 부과되지 않는 free-tier EC2 인스턴스를 기반으로 클러스터링을 구성합니다.
- 인스턴스 개수(4) 입력

## 네트워킹

컨테이너 인스턴스가 사용할 VPC를 구성합니다. VPC는 AWS 클라우드의 격리된 부분으로서, Amazon EC2 인스턴스와 같은 AWS 객체로 채워집니다. 기존 VPC를 선택하거나 이 마법사로 새 VPC를 만들 수 있습니다.

**VPC** vpc-ec151b84 (172.31.... ↻ ⓘ

EC2 콘솔에서 [vpc-ec151b84](#)의 구조를 확인합니다.

**서브넷** ↻

subnet-ab9cf7e7  
(172.31.16.0/20) - ap-north-east-2c  
assign ipv6 on creation: Disabled

subnet-2d1a1d45  
(172.31.0.0/20) - ap-northeast-2a  
assign ipv6 on creation: Disabled

서브넷을 선택하십시오...

**보안 그룹** sg-040b4e27a660beefb... ↻ ⓘ

EC2 콘솔의 [sg-040b4e27a660beefb](#)에 대한 규칙입니다.

### 3. 클러스터 - 네트워크 설정

- 기본 VPC / 서브넷을 사용합니다.
  - 4주차에 진행한 AWS 네트워킹 수업 내용 참고
  - 필요할 경우 VPC / (Public, Private) Subnet을 만들어 적용합니다.
- 보안그룹 적용
  - 해당 클러스터에 구성되는 EC2 인스턴스 내엔 WAS 컨테이너가 구성됩니다. 해당 컨테이너들은 다이나믹 포트로 통신되기에 Host 머신인 EC2 인스턴스의 보안 그룹(방화벽)은 모든 포트 요청이 허용되어야 합니다.

### 4. 컨테이너 인스턴스 IAM 역할 > ecsInstanceRole 선택 후 생성 버튼 클릭

## AWS ECS 작업 정의 구성 생성 및 설정

### 1. Elastic Container Service > 왼쪽 작업 정의 > 새 작업 정의 생성 클릭

## 작업 정의 생성

작업 정의는 작업에 포함할 컨테이너와 그 컨테이너들이 상호 작용하는 방식을 지정합니다. 컨테이너가 사용할 데이터 볼륨을 지정할 수도 있습니다. [자세히 알아보기](#)

To learn about which parameters are supported for Windows Containers, please see the [ECS Documentation](#).

작업 정의 이름\*

api-task

i

작업 역할

없음

↻

인증된 AWS 서비스에 API 요청을 할 때 작업이 사용할 수 있는 IAM 역할 옵션입니다. [IAM 콘솔](#)에서 Amazon Elastic Container Service 작업 역할을 생성합니다. [↗](#)

네트워크 모드

브리지

i

<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.

### 2. 작업 정의 생성

- 작업 정의 이름(**api-task**) 입력
- 작업 역할(**없음**) 선택
- 네트워크 모드(**브리지**) 선택
  - Host 머신과 컨테이너의 포트를 직접 연결이 아닌 다이내믹 포트로 연결하기 위해 **브리지**로 네트워크 모드를 구성합니다.

### 작업 크기

작업 크기를 통해 작업의 고정된 크기를 지정할 수 있습니다. 작업 크기는 Fargate 시작 유형을 사용하는 작업에 대해 필요하며, EC2 시작 유형에 대해서는 선택 사항입니다. 작업 크기가 설정되면 컨테이너 수준 메모리 설정은 선택 사항입니다. 작업 크기는 Windows 컨테이너에서 지원되지 않습니다.

작업 메모리(MiB)

512

The amount of memory (in MiB) used by the task. It can be expressed as an integer using MiB, for example 1024, or as a string using GB, for example '1GB' or '1 gb'.

작업 CPU(유닛)

1024

The number of CPU units used by the task. It can be expressed as an integer using CPU units, for example 1024, or as a string using vCPUs, for example '1 vCPU' or '1 vcpu'.

컨테이너 메모리 예약에 대한 작업 메모리 최대 할당



컨테이너에 대한 작업 CPU 최대 할당



컨테이너 정의

컨테이너 추가

### 3. 작업 크기 설정

- 작업 메모리(**512**) 입력
- 작업 CPU(**1024** = 1vCPU) 입력
- 하단 **컨테이너 추가** 클릭

## 컨테이너 추가

## ▼ 표준

컨테이너 이름\*  ⓘ

이미지\*  ⓘ

사용자 지정 이미지 형식: [registry-url]/[namespace]/[image]:[tag]

프라이빗 레지스트리 인증\* ☐ ⓘ

메모리 제한(MB)\*\*

하드 제한 ▼	<input type="text" value="500"/>	✖	ⓘ
소프트 제한 ▼	<input type="text" value="256"/>	✖	

컨테이너에 MiB 단위로 하드 및/또는 소프트 메모리 제한을 정의합니다. 하드 및 소프트 제한은 작업 정의에서 각각 `memory` 및 `memoryReservation` 파라미터에 상응합니다.  
ECS는 웹 애플리케이션을 시작점으로 300-500MB를 권장합니다.

포트 매핑	호스트 포트	컨테이너 포트	프로토콜	ⓘ
	<input type="text"/>	<input type="text" value="8080"/>	<input type="text" value="tcp"/>	✖

[+ 포트 매핑 추가](#)

작업들에 애플리케이션 로드밸런서(ALB)를 사용한다면, 호스트 포트에 0을 입력하여 다이내믹 포트

## 4. 컨테이너 추가 설정

- 컨테이너 이름(**api-container**) 입력
- 이미지(**{ECR-API-REPO-URL}:latest**) 입력
  - 앞서 설정한 ECR - **api-repo**의 URL:latest를 입력합니다.
- 메모리 제한
  - 하드 제한(**500**) 입력
  - 소프트 제한(**256**) 입력
- 포트 매핑
  - 컨테이너 포트(**8080**) 입력
    - 앞서 해당 컨테이너는 **브리지** 네트워크 모드로 설정되어있습니다. 다이내믹 포트 매핑하기 위해 **호스트 포트**는 비워둡니다.

5. 하단 **생성** 클릭

## AWS ALB Target group - Container 생성 및 설정

1. AWS Management Console > EC2 > 왼쪽 로드밸런싱 탭에 있는 **대상 그룹** 클릭 > **대상 그룹** 생성 버튼 클릭

## 대상 그룹 생성



로드 밸런서는 지정된 프로토콜 및 포트를 사용하여 특정 대상 그룹의 대상으로 요청을 라우팅하며, 지정된 상태 검사 설정을 사용하여 대상에 상태 검사를 수행합니다.

대상 그룹 이름 ⓘ

프로토콜 ⓘ

포트 ⓘ

대상 유형 ⓘ

VPC ⓘ

## 상태 검사 설정

프로토콜 ⓘ

경로 ⓘ

## ▶ 고급 상태 검사 설정

취소 **생성**

## 2. 대상 그룹 생성 설정

- 대상 그룹 이름(**api-target-group**) 입력
- 포트(**3000**) 입력
  - 해당 대상 그룹에 묶일 컨테이너(**api-container**)는 WAS(Node.js)가 운영되는 환경입니다. 해당 컨테이너 내부(EXPORT)는 8080이며 로드밸런서로 들어온 요청(3000번 포트)을 다이나믹 포트로 연결되어있는 대상 그룹의 컨테이너들에게 요청을 분산합니다.

서비스
 리소스 그룹

전용 호스트
 이미지
 AMI
 번들 작업
 ELASTIC BLOCK STORE
 볼륨
 스냅샷
 네트워크 및 보안
 보안 그룹
 탄력적 IP
 배치 그룹
 키 페어
 네트워크 인터페이스
 로드 밸런싱
 로드밸런서
 대상 그룹
 AUTO SCALING
 시작 구성
 Auto Scaling 그룹

SYSTEMS MANAGER 서

로드 밸런서 생성
 작업

태그 및 속성별 필터 또는 키워드별 검색
 이름
 DNS 이름
 상태
 VPC ID
 web-container-lb
 web-container-lb-129975738...
 active
 vpc-ec151b84

로드 밸런서: web-container-lb
 설명
 리스너
 모니터링
 태그
 리스너는 구성된 프로토콜 및 포트를 사용하여 연결 요청 여부를 확인하고, 로드 밸런서는 리스너 규칙을 사용하여 요청을 대상으로 라우팅합니다. 리스너
 리스너 추가
 편집
 삭제
 리스너 ID
 보안 정책
 SSL 인증서
 규칙
 HTTP : 80
 해당 사항 없음
 해당 사항 없음
 기본값: 다음으로 전달 중: web-target-group
 규칙 보기/편집

### 3. 로드밸런서 탭 > web-container-lb 클릭 > 하단 리스너 탭 클릭 후 리스너 추가 버튼 클릭

### 4. web-container-lb 리스너 추가

- 프로토콜 : 포트(HTTP : 3000) 입력
  - api-target-group은 해당 ALB에 들어온 3000번대 요청을 컨테이너 분산합니다. 3000번 포트로 들어온 요청을 다이내믹 포트와 연결된 api-container \* N대로 요청을 전달합니다.
- 1. 전달 대상... 선택 > api-target-group 선택 > 저장

## AWS ECS 클러스터 내 서비스(컨테이너) 생성 및 설정

1. AWS Management Console > Elastic Container Service > 클러스터 > ecs-cluster 클릭
2. 하단 서비스 탭 내에 있는 생성 버튼 클릭



## 서비스 구성

서비스를 통해 클러스터에서 실행하고 유지 관리할 작업 정의의 사본 개수를 지정할 수 있습니다. Elastic Load Balancing 로드 밸런서를 옵션으로 사용하여 들어오는 트래픽을 서비스 내 컨테이너에 분산할 수 있습니다. Amazon ECS는 로드 밸런서를 통해 작업의 개수를 유지하고 작업 일정을 조정합니다. 서비스 Auto Scaling을 옵션으로 사용하여 서비스 내 작업의 개수를 조정할 수도 있습니다.

작업 정의	Family	api-task	값 입력
	Revision	1 (latest)	
클러스터	ecs-cluster		
서비스 이름	api		
서비스 유형*	<input checked="" type="radio"/> REPLICA <input type="radio"/> DAEMON		
작업 개수	2		
최소 정상 상태 백분율	50		
최대 백분율	200		

### 3. 서비스 구성 설정

- 작업 정의(api-task) 선택
- 작업 개수(2) 입력
  - 두 개의 컨테이너를 ecs-cluster 내에 생성합니다.
- 하단 다음 단계 버튼 클릭

## Elastic Load Balancing(선택 사항)

Elastic Load Balancing 로드 밸런서는 들어오는 트래픽을 서비스에서 실행 중인 작업에 두루 분산합니다. 기존 로드 밸런서를 선택하거나 [Amazon EC2 콘솔](#)에서 새 로드 밸런서를 생성합니다. 로드 밸런서 구성을 마쳤으면 **저장**을 선택하여 작업을 계속합니다.

### ELB 유형:

☐ 없음

해당 서비스에서는 로드 밸런서를 사용하지 않습니다.

☒ Application Load Balancer

컨테이너가 동적 호스트 포트 매핑을 사용하도록 허용합니다(컨테이너 인스턴스마다 다중 작업이 허용됨). 여러 서비스가 규칙 기반 라우팅 및 경로를 사용하여 단일 로드 밸런서에서 동일한 리스너 포트를 사용할 수 있습니다.

☐ Network Load Balancer

A Network Load Balancer functions at the fourth layer of the Open Systems Interconnection (OSI) model. After the load balancer receives a request, it selects a target from the target group for the default rule using a flow hash routing algorithm.

☐ Classic Load Balancing

정적 호스트 포트 매핑을 필요로 합니다(컨테이너 인스턴스당 1개의 작업만 허용됨). 규칙 기반 라우팅 및 경로가 지원되지 않습니다.

서비스의 IAM 역할 선택

ecsServiceRole



ELB 이름

web-container-lb



## 로드를 밸런싱할 컨테이너

컨테이너 선택

api-container:0:8080

ELB에 추가

### 4. Elastic Load Balancing 설정

- ELB 유형(Application Load Balancer) 선택
- 서비스의 IAM 역할 (ecsServiceRole) 선택
- ELB 이름(web-container-lb) 선택
- 컨테이너 선택(api-container:0:8080) 클릭 후 ELB에 추가 버튼 클릭

## api-container : 8080

제거 ✕

리스너 포트

3000:HT...



리스너 프로토콜

HTTP

대상 그룹 이름

api-target-group ▼



대상 그룹 프로토콜

HTTP



대상 유형

instance



경로 패턴

/

평가 순서

default

상태 확인 경로

/



서비스 생성 후 ELB 콘솔에 추가 상태 확인 옵션을 구성할 수 있습니다.

## 5. 대상 그룹 선택

- 중간에 위치한 대상 그룹 이름에서 앞서 생성한 대상 그룹인 `api-target-group`을 선택하면 나머지 값들이 설정한 값으로 적용됩니다.
- 하단 다음 단계 버튼 클릭
- Auto Scaling은 건너뛰니다.
- 서비스 검토 탭에서 앞서 설정한 정보들을 확인 후 하단 서비스 생성 버튼을 클릭합니다.

## AWS Codepipeline + Codebuild + ECS CI/CD 무중단 파이프라인 구축

## AWS Codepipeline 생성 및 설정

1. AWS Management Console > Codepipeline > 파이프라인 생성 버튼 클릭
2. 파이프라인 이름(`ecs-api-pipeline`) 입력 > 다음 단계 버튼 클릭

## 파이프라인 생성

1 단계: 이름

2 단계: 소스

3 단계: 빌드

4 단계: 배포

5 단계: 서비스 역할

6 단계: 검토

### 소스 위치



소스 코드가 저장된 위치를 지정합니다. 공급자를 선택한 후 그 공급자에 관한 연결 세부 정보를 입력합니다.

소스 공급자\*

GitHub

### GitHub에 연결

리포지토리 목록에서 리포지토리를 선택한 후 사용할 브랜치를 선택합니다. 최소한 해당 리포지토리에 대한 읽기 전용 액세스 권한이 있어야 합니다. [자세히 알아보기](#)

리포지토리\*

owen1025/Fastcampus-api-deploy

브랜치\*

container

#### Webhook을 사용하여 변경 감지

AWS CodePipeline이 Webhook을 만듭니다. 아래 옵션에서 옵트아웃할 수 있습니다.

#### ▶ 변경 탐지 옵션

\* 필수

취소

이전

다음 단계

### 3. 소스 설정

해당 과정을 통해 Github webhook이 자동으로 설정되며 설정한 리포지토리/브랜치에 Push 이벤트 발생 시 해당 Pipeline이 실행됩니다.

- 소스 공급자(Github) 선택 > Github 연결 버튼 클릭 후 로그인 or 인증
- 리포지토리({My\_Github\_Username}/Fastcampus-api-deploy) 선택
- 브랜치(container) 선택
- 다음 단계 클릭

## 파이프라인 생성

- 1 단계: 이름
- 2 단계: 소스
- 3 단계: 빌드**
- 4 단계: 배포
- 5 단계: 서비스 역할
- 6 단계: 검토

### 빌드

사용할 또는 이미 사용 중인 빌드 공급자를 선택합니다.

빌드 공급자\* AWS CodeBuild

#### AWS CodeBuild

AWS CodeBuild는 클라우드 환경에서 코드를 빌드 및 테스트할 수 있는 완전관리형 빌드 서비스입니다. CodeBuild는 규모가 지속적으로 조정됩니다. 분당 요금만 지불하면 됩니다. [자세히 알아보기](#)

#### 프로젝트 구성

- ☐ 기존 빌드 프로젝트 선택  
☒ 새 빌드 프로젝트 생성

프로젝트 이름\* 프로젝트 이름 입력

설명 + 설명 추가

## AWS Codedeploy 및 IAM 설정

### 1. 빌드 설정

AWS Codebuild를 통해 리포지토리 내에 Dockerfile을 기반으로 자동으로 Docker image 빌드를 수행합니다.

- 빌드 공급자(AWS CodeBuild) tjsxor
- 새 빌드 프로젝트 생성 선택
- 프로젝트 이름(api-build) 입력

## 환경: 빌드 방법

---

환경 이미지\* ☒ AWS CodeBuild에서 관리하는 이미지 사용  
☐ 도커 이미지 지정

운영 체제\*

실행 시간\*

버전\*

빌드 사양 ☒ 소스 코드 루트 디렉터리에서 `buildspec.yml` 사용  
☐ 빌드 명령 삽입

### 2. 환경: 빌드 방법 설정

- 환경 이미지(AWS CodeBuild에서 관리하는 이미지 사용) 선택
- 운영 체제(Ubuntu) 선택
- 실행 시간(Docker) 선택
- 버전(aws/codebuild/docker...) 선택

## AWS CodeBuild 서비스 역할

AWS CodeBuild가 사용자를 대신하여 종속 AWS 서비스를 호출할 수 있게 해주는 서비스 역할을 지정합니다. [자세히 알아보기](#).

- ☒ 계정에 서비스 역할 생성  
☐ 계정에서 기존 서비스 역할 선택

역할 이름\*

## VPC

VPC ID\*

## ▶ 고급

## 빌드 프로젝트 저장

**빌드 프로젝트 저장**

### 3. AWS CodeBuild 서비스 역할 설정

- 계정에 서비스 역할 생성 선택
- 역할 이름 > 기본값 사용
- 빌드 프로젝트 저장 버튼 클릭
- 해당 AWS Codebuild 아이템에 역할 부여를 위해 AWS Management Console > IAM으로 이동합니다.

The screenshot shows the AWS IAM console interface. On the left, there is a navigation menu with options like '대시보드', '그룹', '사용자', '역할', '정책', '자격 증명 공급자', '계정 설정', '자격 증명 보고서', and '암호화 키'. The main area displays a table of roles. The role 'code-build-api-build-service-role' is highlighted with a red box. Below it, 'code-build-web-test-build-service-role' is also visible. The table columns are '역할 이름', '설명', and '신뢰할 수 있는 개체'.

역할 이름	설명	신뢰할 수 있는 개체
<input type="checkbox"/> AWS-CodePipeline-Service		AWS 서비스: codepipeline
<input type="checkbox"/> AWSServiceRoleForAutoScaling	Default Service-Linked Role enables access to AWS Services and Resource...	AWS 서비스: autoscaling (서비스 연결 역할)
<input type="checkbox"/> AWSServiceRoleForECS		AWS 서비스: ecs (서비스 연결 역할)
<input type="checkbox"/> AWSServiceRoleForElasticLoadBala...	Allows ELB to call AWS services on your behalf.	AWS 서비스: elasticloadbalancing (서비스 연결 ...
<input type="checkbox"/> AWSServiceRoleForRDS	Allows Amazon RDS to manage AWS resources on your behalf	AWS 서비스: rds (서비스 연결 역할)
<input type="checkbox"/> AWSServiceRoleForSupport	Enables resource access for AWS to provide billing, administrative and supp...	AWS 서비스: support (서비스 연결 역할)
<input type="checkbox"/> AWSServiceRoleForTrustedAdvisor	Access for the AWS Trusted Advisor Service to help reduce cost, increase p...	AWS 서비스: trustedadvisor (서비스 연결 역할)
<input type="checkbox"/> code-build-api-build-service-role		AWS 서비스: codebuild
<input type="checkbox"/> code-build-web-test-build-service-role		AWS 서비스: codebuild
<input type="checkbox"/> code-deploy-ec2-role	Allows EC2 instances to call AWS services on your behalf.	AWS 서비스: ec2
<input type="checkbox"/> ecsInstanceRole		AWS 서비스: ec2
<input type="checkbox"/> ecsServiceRole		AWS 서비스: ecs

#### 4. AWS Codebuild에 역할 부여

- IAM 콘솔로 이동 후 왼쪽 탭에서 **역할** 탭을 클릭하면 앞서 설정한 Codebuild 역할로 자동 생성된 **code-build-api-build-service-role**를 클릭합니다.
- **요약** 탭 하단 **권한** 탭에서 **정책 연결** 버튼을 클릭합니다.

#### code-build-api-build-service-role에 권한 추가

##### 권한 연결

정책 생성

필터 정책

ec2container

8 결과 표시

	정책 이름	유형	로더 사용됨	설명
<input type="checkbox"/>	AmazonEC2ContainerRegistryFu...	AWS 관리형	없음	Provides administrative access to Amazon ECR resources
<input checked="" type="checkbox"/>	AmazonEC2ContainerRegistryP...	AWS 관리형	Permissions policy (1)	Provides full access to Amazon EC2 Container Registry repositories, but do...
<input type="checkbox"/>	AmazonEC2ContainerRegistryR...	AWS 관리형	없음	Provides read-only access to Amazon EC2 Container Registry repositories.
<input type="checkbox"/>	AmazonEC2ContainerServiceAut...	AWS 관리형	없음	Policy to enable Task Autoscaling for Amazon EC2 Container Service
<input type="checkbox"/>	AmazonEC2ContainerServiceEv...	AWS 관리형	없음	Policy to enable CloudWatch Events for EC2 Container Service
<input type="checkbox"/>	AmazonEC2ContainerServicefor...	AWS 관리형	Permissions policy (1)	Default policy for the Amazon EC2 Role for Amazon EC2 Container Service.
<input type="checkbox"/>	AmazonEC2ContainerServiceFul...	AWS 관리형	없음	Provides administrative access to Amazon ECS resources.
<input type="checkbox"/>	AmazonEC2ContainerServiceRole	AWS 관리형	Permissions policy (1)	Default policy for Amazon ECS service role.

취소

정책 연결

해당 사진처럼 **ec2container**로 검색하면 관련 권한 리스트가 검색됩니다. 그 중 **AmazonEC2ContainerRegistryPowerUser**를 클릭하고 하단 **정책 연결** 버튼을 클릭하여 권한을 부여합니다. 그 후 **AWS Codepipeline** 설정한 화면으로 이동합니다.

#### AWS Codedeploy(ECS) 설정



## 배포



인스턴스에 배포하는 방식을 선택합니다. 공급자를 선택한 후 그 공급자에 관한 구성 세부 정보를 입력합니다.

배포 공급자\*

Amazon ECS



### Amazon ECS ⓘ

기존 클러스터 중 하나를 선택하거나 Amazon ECS에서 새로 생성합니다.

클러스터 이름\*

ecs-cluster



기존 서비스 중 하나를 선택하거나 Amazon ECS에서 새로 생성합니다.

서비스 이름\*

api



이미지 정의 파일의 이름을 입력하십시오. Amazon ECS 서비스의 컨테이너 이름 및 이미지와 태그를 설명하는 JSON 파일입니다.

이미지 파일 이름

MyFilename.json

\* 필수

취소

이전

다음 단계

### 1. 배포 설정

- 배포 공급자(Amazon ECS) 선택
- 클러스터 이름(ecs-cluster) 선택
- 서비스 이름(api) 선택
- 이미지 파일 이름(MyFilename.json) 입력
- 하단 다음 단계 버튼 클릭

### 2. AWS 서비스 역할 설정

- AWS-CodePipeline-Service를 선택합니다. > 혹시 해당 역할이 없다면 오른쪽 역할 만들기 버튼을 클릭합니다.

## ▼ 세부 정보 숨기기

### 역할 요약 ?

**역할 설명** Provides read and write access to AWS services and resources.

**IAM 역할**

**정책 이름**

▶ 정책 문서 보기

- 다음 단계 버튼 클릭
- 파이프라인 설정 검토 이후 하단 파이프라인 생성 버튼을 클릭합니다.

## buildspec 설정 및 Github에 Push

### buildspec 생성 및 정의

포크하신 `Fastcampus-api-deploy` 디렉토리 내에 `buildspec.yml`를 생성하여 아래 내용을 추가합니다. 스크립트 안에 주석에 적힌 내용을 확인하신 후 적용하신 뒤에 저장합니다. 혹시 아래 코드를 복사하기 힘들다면 `buildspec.yml`로 접속하여 복사 후 수정합니다.

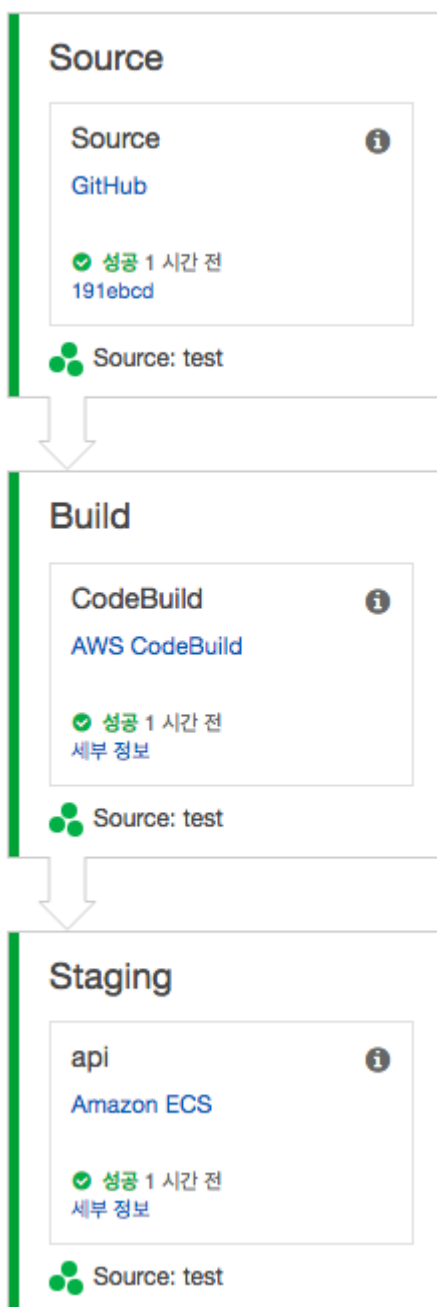
```
version: 0.2

phases:
  pre_build:
    commands:
      - echo Logging in to Amazon ECR...
      - aws --version
      - $(aws ecr get-login --region $AWS_DEFAULT_REGION --no-include-email)
      - REPOSITORY_URI=322749112518.dkr.ecr.ap-northeast-2.amazonaws.com/api-repo # web 이미지가 저장된 ECR 리포지토리 URL을 입력합니다. ex. REPOSITORY_URI={ECR-API-REPO-URL}
      - IMAGE_TAG=$(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":"api-container","imageUri":"%s"}'
$REPOSITORY_URI:$IMAGE_TAG > MyFilename.json
artifacts:
  files: MyFilename.json
```

AWS Codepipeline 테스트를 위해 Github에 Push하여 CI/CD 과정이 정상적으로 작동하는 지 확인합니다.

```
git add --all
git commit -m "ci/cd test"
git push
```



해당 화면과 같이 Source, Build, Staging 과정이 성공하면 정상적으로 작동하는 겁니다. 수고하셨습니다.