



**Continuous Deployment to  
Amazon ECS  
(Limited use only)**

© 2017 Amazon Web Services, Inc. or its affiliates. All rights reserved.

This work may not be reproduced or redistributed, in whole or in part, without prior written permission from Amazon Web Services, Inc. Commercial copying, lending, or selling is prohibited.

Corrections or feedback on the lab guide, please email me at: [piljoong@amazon.com](mailto:piljoong@amazon.com).

All trademarks are the property of their owners.

# Index

실습 1. Amazon ECS 로 애플리케이션 배포

실습 2. Amazon ECS 로 애플리케이션 Blue/Green 배포

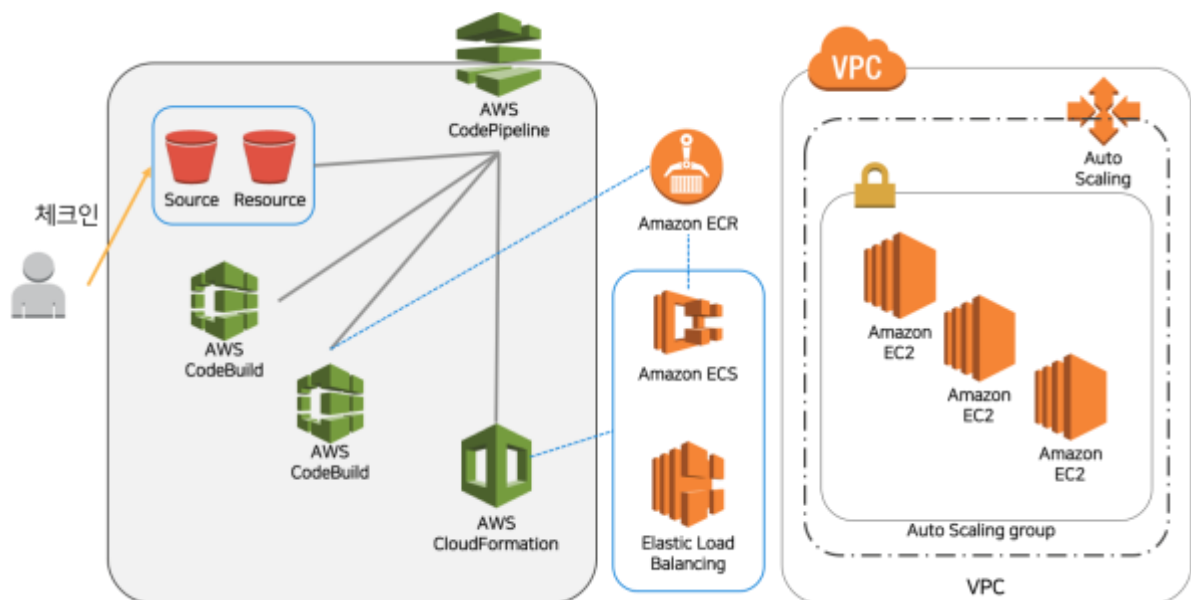
## 실습 1

# Amazon ECS 로 애플리케이션 배포

### Overview

이 실습은 AWS 개발자 도구를 활용하여 컨테이너 기반의 애플리케이션을 Amazon ECS 클러스터에 배포 자동화하는 방법을 익히기 위함입니다. 이번 실습에서는 Amazon CodePipeline 으로 배포 프로세스를 구성하고, Amazon CodeCommit (**본 실습에서는 S3 를 사용**) 에 저장되어 있는 소스코드를 Amazon CodeBuild 를 통해 컴파일 및 컨테이너 이미지 를 생성 한 뒤, Amazon CloudFormation 으로 Amazon ECS 에 배포 해볼 것 입니다. 데브옵스는 조직의 문화와 사용하는 도구들, 방법론에 따라 다양한 방법이 있기에, 이 실습에서 다루는 내용은 그 중 한가지 예제를 보여주기 위함으로 알아주시기 바랍니다.

실습 1 에서는 CodePipeline, CodeCommit, CodeBuild, CloudFormation 을 설정하여 Golang 으로 작성된 웹 애플리케이션 배포를 자동화해 볼 것 입니다.



<그림 1. 실습 1 아키텍처>

### Objectives

이 실습을 완료하면 다음을 할 수 있습니다.

- 배포 자동화를 위해 개발자 도구를 활용할 수 있습니다.
- Amazon ECS 를 통해 클러스터를 구축할 수 있습니다.
- AWS CodePipeline 을 설정할 수 있으며, 자동화 프로세스를 가시적으로 구성할 수 있습니다.
- AWS CodeCommit 을 설정할 수 있으며, 저장소로 활용 가능합니다.
- AWS CodeBuild 를 설정할 수 있으며, 코드 빌드는 물론 테스트와 같은 특정 환경에서 특정 작업을 진행할 수 있습니다.
- AWS CloudFormation 템플릿을 배포 자동화에 활용할 수 있습니다.
- 배포 자동화 파이프라인에서 S3 의 다양한 활용을 알 수 있습니다.

## Prerequisites

---

Lab 을 완료하기 위해 다음이 필요합니다:

- Wi-Fi 사용기 가능한 노트북 컴퓨터 (Mac OS X, Linux, Windows)
- Windows 사용자는 Administrator 권한이 필요
- Windows 사용자는 Putty 필요
- Chrome, Firefox, Safari 등의 인터넷 브라우저

## Duration

---

60 분

## Task 1: 배포 자동화 구성을 위한 리소스 및 환경 구성

### Overview

---

이번 Task에서는 배포 자동화 구성을 위해 필요한 리소스 및 환경을 구성합니다.

참고) 원활한 lab 진행을 위해 필요로 하는 최소한의 설정으로만 진행합니다. 실제 서비스를 위한 환경에서는 접근 제어와 같은 부분에 대해서 더 세심하게 설정하여야 합니다.

## Task 1.1: Artifact 저장을 위한 S3 버킷 생성

### Overview

---

이번 섹션에서는 Artifact 저장을 위한 S3 버킷을 생성합니다. 생성된 S3 버킷은 배포 자동화에 필요한 스크립트 뿐만 아니라 CloudFormation 템플릿 등을 저장하는데 사용 됩니다. S3 버킷을 배포 자동화에 필요한 리소스를 위한 저장소로 활용하면 보다 안전하고 유연하게 자동화 환경을 구성할 수 있으며 효율적입니다.

1.1.1 AWS 관리 콘솔에 로그인 후 **Services** → **Storage** → **S3** 를 선택 합니다.

1.1.2 Create bucket 을 클릭 합니다.

1.1.3 다음과 같이 설정합니다.

- Bucket name: ecsdeploy-{my\_name} (또는 원하는 버킷 이름 설정)
- Region: US East (N. Virginia, 미국 동부-버지니아 북부)

1.1.4 **Next** 를 클릭 합니다.

1.1.5 Versioning 을 클릭한 뒤 Enable versioning 을 선택한 뒤 Save 를 클릭 합니다.

1.1.6 **Next** 를 2 번 클릭하여 **Create bucket** 을 클릭하여 버킷을 생성합니다.

## Task 1.2: VPC 로 네트워크 환경 구성

### Overview

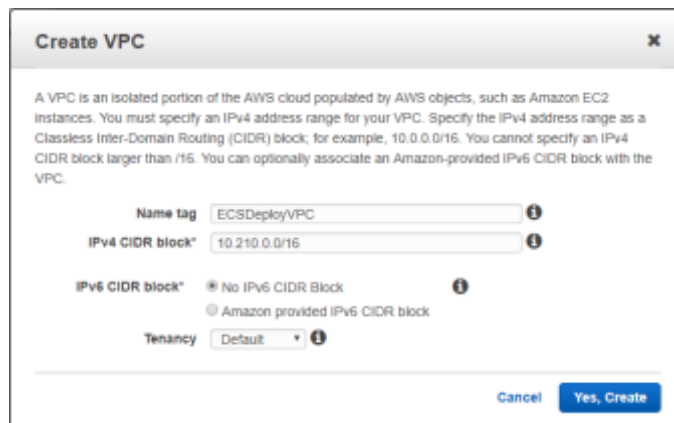
이번 섹션에서는 배포된 시스템이 동작할 네트워크 환경을 구성하기 위해 VPC 를 구성합니다.

1.2.1 AWS 관리 콘솔에 로그인 후 Services → Networking & Content Delivery → VPC 을 선택 합니다.

1.2.2 왼쪽 패널의 **Your VPCs** 를 클릭한 뒤 **Create VPC** 를 클릭 합니다.

1.2.3 다음과 같이 설정합니다:

- Name tag: ECSDeployVPC
- IPv4 CIDR block: 10.210.0.0/16



1.2.4 **Yes, Create** 를 클릭하여 생성합니다.

1.2.5 왼쪽 패널의 Internet Gateways 를 클릭합니다.

1.2.6 Create Internet Gateway 를 클릭 한뒤 다음과 같이 설정합니다:

- Name tag: ECSDeployIGW



1.2.7 **Attach to VPC** 를 클릭한 뒤 바로 전에 만든 **ECSDeployVPC** 에 Attach 합니다.

1.2.8 왼쪽 패널의 **Subnets** 를 클릭한 뒤 **Create Subnet** 을 클릭합니다.

1.2.9 다음과 같이 설정하여 Subnet 하나를 생성합니다:



- Name tag: ECSDeploySubnetA
- VPC: (바로 전에 만든) ECSDeployVPC
- Availability Zone: us-east-1a
- IPv4 CIDR block: 10.210.10.0/24

**Create Subnet**

Use the CIDR format to specify your subnet's IP address block (e.g., 10.0.0.0/24). Note that block sizes must be between a /16 netmask and /28 netmask. Also, note that a subnet can be the same size as your VPC. An IPv6 CIDR block must be a /64 CIDR block.

Name tag:

VPC:

VPC CIDRs

CIDR	Status	Status Reason
10.210.0.0/16	associated	

Availability Zone:

IPv4 CIDR block:

[Cancel](#) [Yes, Create](#)

1.2.10 **Yes, Create** 를 클릭하여 생성합니다.

1.2.11 같은 방법으로 아래 내용으로 두번째 Subnet 을 생성합니다:

- Name tag: ECSDeploySubnetB
- VPC: (바로 전에 만든) ECSDeployVPC
- Availability Zone: us-east-1b
- IPv4 CIDR block: 10.210.20.0/24

**Create Subnet**

Use the CIDR format to specify your subnet's IP address block (e.g., 10.0.0.0/24). Note that block sizes must be between a /16 netmask and /28 netmask. Also, note that a subnet can be the same size as your VPC. An IPv6 CIDR block must be a /64 CIDR block.

Name tag:

VPC:

VPC CIDRs

CIDR	Status	Status Reason
10.210.0.0/16	associated	

Availability Zone:

IPv4 CIDR block:

[Cancel](#) [Yes, Create](#)

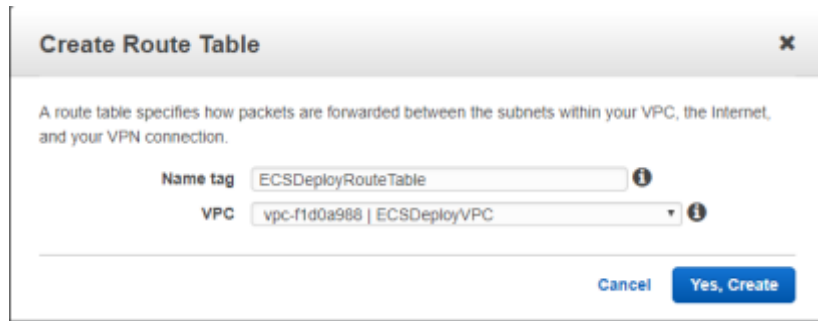
1.2.12 **Yes, Create** 를 클릭하여 두번째 Subnet 을 생성합니다.

1.2.13 왼쪽 패널의 Route Tables 를 클릭한 뒤 Create Route Table 을 클릭합니다.

1.2.14 다음과 같이 설정합니다:

- Name tag: ECSDeployRouteTable

- VPC: (바로 전에 만든) ECSDeployVPC



**Create Route Table**

A route table specifies how packets are forwarded between the subnets within your VPC, the Internet, and your VPN connection.

Name tag: ECSDeployRouteTable

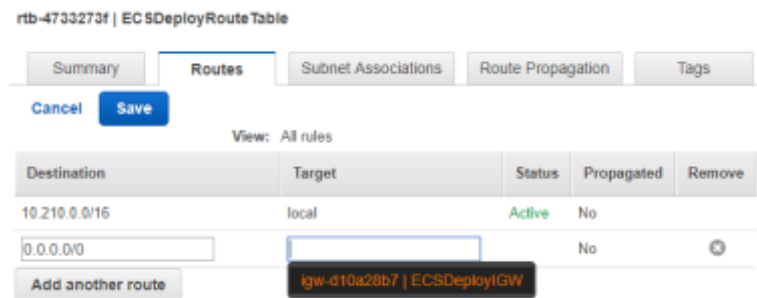
VPC: vpc-f1d0a988 | ECSDeployVPC

Buttons: Cancel, Yes, Create

1.2.15 **Yes, Create** 를 클릭하여 생성합니다.

1.2.16 아래 패널의 **Routes** 를 클릭한 뒤 **Edit** 를 누른 후 **Add another route** 를 눌러 다음을 추가 합니다:

- Destination: 0.0.0.0/0
- Target: (이전에 만든) ECSDeployIGW



rtb-4733273f | ECSDeployRouteTable

Summary Routes Subnet Associations Route Propagation Tags

Cancel Save

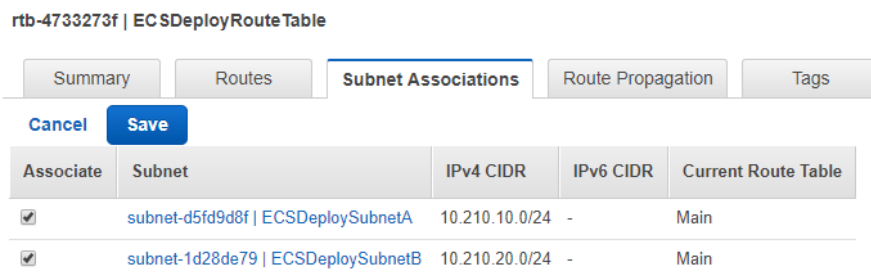
View: All rules

Destination	Target	Status	Propagated	Remove
10.210.0.0/16	local	Active	No	
0.0.0.0/0	igw-d10a28b7   ECSDeployIGW		No	

Add another route

1.2.17 **Save** 를 눌러 변경사항을 저장합니다.

1.2.18 **Subnet Associations** 을 클릭한 뒤 **Edit** 를 클릭하여 생성한 **Subnet** 둘 모두 선택한 뒤 **Save** 를 클릭하여 저장합니다.



rtb-4733273f | ECSDeployRouteTable

Summary Routes Subnet Associations Route Propagation Tags

Cancel Save

Associate	Subnet	IPv4 CIDR	IPv6 CIDR	Current Route Table
<input checked="" type="checkbox"/>	subnet-d5fd9d8f   ECSDeploySubnetA	10.210.10.0/24	-	Main
<input checked="" type="checkbox"/>	subnet-1d28de79   ECSDeploySubnetB	10.210.20.0/24	-	Main

## Task 1.3: CodePipeline 설정

### Overview

---

이번 섹션에서는 배포 자동화 파이프라인을 가시적으로 구성 및 관리할 수 있는 CodePipeline 을 설정합니다.

- 1.3.1 AWS 관리 콘솔에 로그인 후 Services → Developer Tools → CodePipeline 을 선택 합니다.
- 1.3.2 Create pipeline 을 클릭합니다.
- 1.3.3 Pipeline name 에 **ECSDeployPipeline** 을 입력한 뒤 **Next step** 을 클릭합니다.
- 1.3.4 Source provider 에서 Amazon S3 를 선택한 뒤 Amazon S3 location 에 Task 1.1 에서 생성한 S3 버킷 위치와 app.zip 이라는 파일명을 추가하여 위치를 지정해 준 뒤 Next step 을 클릭 합니다:
  - Amazon S3 location: s3://ecsdeploy-{my\_name}/app.zip
- 1.3.5 Build provider 에서 **AWS CodeBuild** 를 선택합니다.
- 1.3.6 **Create a new build project** 를 선택하여 새로운 빌드 프로젝트를 만들어 줍니다.
  - Project name: ECSDeployAppBuilder
  - Environment image: Use an image managed by AWS CodeBuild
  - Operating system: Ubuntu
  - Runtime: golang
  - Version: aws/codebuild/golang:1.7.3
  - Build specification: Use the buildspect.yml in the source code root directory
  - Create a service role in your account
- 1.3.7 **Save build project** 를 클릭하여 빌드 프로젝트를 생성합니다.
- 1.3.8 **Next step** 을 클릭하여 계속 진행합니다.
- 1.3.9 Deployment provider 에서는 **No Deployment** 를 선택한 뒤 **Next step** 을 클릭하여 넘어갑니다. 배포 관련 내용은 추후 다시 설정합니다.
- 1.3.10 AWS Service Role 을 생성하기 위해 **Create role** 을 클릭합니다.

1.3.11 Role 생성화면에서 IAM Role 에서 **Create a new IAM Role** 을 선택하고,  
Role Name 에 **ECSDeployPipeline-ServiceRole** 으로 설정한 뒤 **Allow** 를  
클릭합니다.

1.3.12 **Next step** 을 클릭하여 계속 Review 페이지로 넘어가 입력한 내용을 확인 한  
뒤 **Create pipeline** 을 클릭하여 파이프라인을 생성합니다.

참고) AWS CodePipeline 화면으로 넘어왔을 때 Source 단계에서 Failed 로  
표시되는건 정상적인 경우입니다. 만약 실패 없이 진행하고자 한다면 미리 app.zip  
파일을 S3 에 업로드 하거나, Source 스테이지에서 S3 의 변경 감지를 수동으로  
설정하면 됩니다.

## Task 1.4: Amazon ECS 클러스터 및 ECR 리포지토리 구성

### Overview

---

이번 섹션에서는 생성할 컨테이너 이미지가 저장될 Amazon ECR 리포지토리와 Amazon ECS 클러스터를 구성합니다.

1.4.1 AWS 관리 콘솔에 로그인 후 Services → Compute → EC2 Container Service 을 선택 합니다.

1.4.2 **Create Cluster** 를 클릭한 뒤 다음과 같이 설정 합니다.

- Cluster name: ECSDeployCluster
- Create an empty cluster 선택

1.4.3 **Create** 를 클릭하여 생성합니다.

1.4.4 생성된 빈 클러스터에서 사용될 Task 및 Service 들은 추후 CloudFormation 을 통해서 관리됩니다.

1.4.5 왼쪽 패널의 Repositories 를 클릭한 뒤 Create repository 를 클릭합니다.

1.4.6 Repository name 에 **ecsdeploy/app** 을 입력한 뒤 **Next step** 을 클릭한 뒤 **Done** 을 클릭하여 완료합니다.

## Task 2: 배포 자동화를 위한 파이프라인 수정 및 리소스 구성

### Overview

---

이번 Task에서는 배포 자동화 파이프라인을 수정하고 필요한 리소스를 구성합니다. (다이어그램 및 기타 정보 추가 될 예정)

참고) 원활한 lab 진행을 위해 필요로 하는 최소한의 설정으로만 진행합니다. 실제 서비스를 위한 환경에서는 접근 제어와 같은 부분에 대해서 더 세심하게 설정하여야 합니다.

## Task 2.1: 로드밸런서 구성

### Overview

---

이번 섹션에서는 서비스의 엔드포인트로 사용될 로드밸런서를 구성합니다. 서비스에 사용할 컨테이너는 여기서 생성할 로드밸런서에 등록되어 서비스되어집니다.

2.1.1 로드밸런서 구성 전 Security Group 을 먼저 생성합니다.

2.1.2 AWS 관리 콘솔에 로그인 후 **Services** → **Compute** → **EC2** 를 선택합니다.

2.1.3 왼쪽 패널의 Security Groups 을 클릭한 뒤 Create Security Group 을 클릭합니다.

2.1.4 다음 내용으로 설정합니다:

- Security group name: ECSDeployLB-SG
- Description: ECSDeploy LoadBalancer SG
- VPC: (이전에 생성한) ECSDeployVPC
- Inbound:
  - Custom TCP Rule/TCP/80/Anywhere
  - Custom TCP Rule/TCP/8080/Anywhere

2.1.5 **Create** 를 클릭하여 생성합니다.

2.1.6 왼쪽 패널의 **Load Balancers** 를 클릭합니다.

2.1.7 Create Load Balancer 를 클릭합니다.

2.1.8 **Application Load Balancer** 를 선택한 상태에서 **Continue** 를 클릭한 뒤 다음 내용으로 설정합니다:

- Name: ECSDeployLB
- Scheme: internet-facing
- IP address type: ipv4
- Listeners: HTTP/80
- Availability Zones:
  - VPC: (이전에 생성한) ECSDeployVPC
  - 표시되는 두개의 AZ (us-east-1a, us-east-1b) 모두 선택

2.1.9 Next: Configure Security Settings 을 클릭한 뒤 Next: Configure Security Groups 을 클릭합니다.

2.1.10 Select an existing security group 을 선택한 뒤 이전에 생성한 ECSDeployLB-SG 를 선택한 뒤 Next: Configure Routing 을 클릭합니다.

2.1.11 다음 내용으로 Target Group 및 Health checks 를 설정합니다:

- Target group:
  - Target group: New target group
  - Name: ECSDeploy-DefaultTG
  - Protocol: HTTP
  - Port: 80
- Health checks:
  - Protocol: HTTP
  - Path: /
  - Advanced health check settings
    - Port: traffic port
    - Healthy threshold: 2
    - Unhealthy threshold: 2
    - Timeout: 5
    - Interval: 10
    - Success codes: 200-299

2.1.12 **Next: Register Targets** 을 클릭한 뒤 **Next: Review** 를 클릭하고 설정값을 다시 확인한 뒤 **Create** 를 눌러 생성합니다.



## Task 2.2: 오토스케일링 설정

### Overview

---

이번 섹션에서는 오토스케일링을 설정하여 ECS 클러스터에 인스턴스를 추가합니다. 클러스터에 인스턴스를 추가하기 위해 오토스케일링을 활용하면 최소 및 최대의 인스턴스 개수를 지정할 수 있을 뿐만 아니라 부하가 늘어남에 따라 요구되는 컴퓨팅 리소스를 추가하도록 구성할 수 있습니다.

- 2.2.1 오토스케일링 그룹을 통해 구동되는 인스턴스는 Amazon ECS 컨테이너 에이전트를 통해 Amazon ECS API 작업을 호출합니다. 이를 위해 컨테이너 인스턴스에 적절한 IAM 정책과 역할이 필요합니다. 이를 위해 IAM 에서 인스턴스 Role 을 먼저 생성합니다.
- 2.2.2 AWS 관리 콘솔에 로그인 후 **Services → Security, Identity & Compliance → IAM** 을 클릭합니다.
- 2.2.3 왼쪽 패널의 **Roles** 를 클릭한 뒤 **Create role** 을 클릭합니다.
- 2.2.4 Select type of trusted entity 에서 **AWS Service** 박스를 선택한 뒤 Choose the service that will use this role 에서 **EC2** 를 선택합니다. 다시 하단의 Select your use case 에서 **EC2** 를 선택한 뒤 **Next: Permissions** 을 클릭합니다.
- 2.2.5 Attach permissions policies 에 나오는 리스트 중에서 **AmazonEC2ContainerServiceforEC2Role** 을 선택한 뒤 **Next: Review** 를 클릭합니다.
- 2.2.6 Role name 에 **ECSDeployEC2-Role** 을 입력한 뒤 **Create role** 을 클릭합니다.
- 2.2.7 오토스케일링 설정을 위해 **Services → Compute → EC2** 를 선택합니다.
- 2.2.8 Launch Configuration 에서 사용될 Security Group 을 생성하기 위해 왼쪽 패널의 **Security Groups** 을 클릭합니다.
- 2.2.9 다음과 같이 설정합니다:
  - Security group name: ECSDeployASLC-SG
  - Description: ECSDeploy ASLC SG
  - VPC: (이전에 생성한) ECSDeployVPC
  - Inbound:
  - All traffic/All/0 – 65535/ ECSDeployLB-SG

2.2.10 **Create** 를 클릭하여 생성합니다.

2.2.11 왼쪽 패널의 **Launch Configurations** 을 클릭한 뒤 Create Auto Scaling Group 을 클릭한 후 Create launch configuration 을 클릭합니다.

2.2.12 Community AMIs 을 선택한 뒤 검색어 2017 amazon ecs optimized 를 입력하여 나오는 이미지 중 amzn-ami-2017.03.d-amazon-ecs-optimized – ami-04351e12 를 Select 합니다.

2.2.13 t2.large 인스턴스 타입을 선택한 뒤 Next: Configure details 를 클릭합니다.

2.2.14 다음과 같이 설정합니다:

- Name: ECSDeployASLC
- IAM Role: (바로 전에 생성한) ECSDeployEC2-Role
- Advanced Details:
  - User data:
 

```
#!/bin/bash
echo ECS_CLUSTER=ECSDeployCluster > /etc/ecs/ecs.config
```
  - IP Address Type: Assign a public IP address to any instances.

2.2.15 Next: Add Storage 를 클릭한 뒤 Next: Configure Security Group 을 클릭합니다.

2.2.16 Select an existing security group 을 선택한 뒤 (이전에 생성한) ECSDeployASLC-SG 을 선택한 뒤 Review 를 클릭합니다.

2.2.17 입력 내용을 다시 한번 확인한 뒤 **Create launch configuration** 을 클릭합니다.

2.2.18 생성된 Launch Configuration 으로 Auto Scaling Group 을 생성하기 위해 **Create an Auto Scaling group using this launch configuration** 을 클릭합니다.

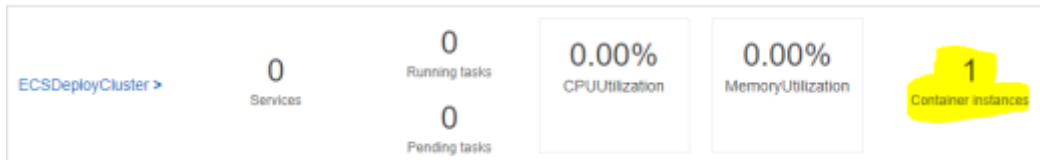
2.2.19 다음 내용으로 설정합니다:

- Group name: ECSDeployASG
- Group size: 1
- Network: (이전에 만든) ECSDeployVPC
- Subnet: (이전에 만든 두 Subnet 모두) ECSDeploySubnetA, ECSDeploySubnetB
- Advanced Details:
  - Health Check Grace Period: 300 → 30 seconds

2.2.20 Next: Configure scaling polices 를 클릭한 뒤 Review 를 클릭합니다.

2.2.21 입력한 내용을 다시 한번 확인한 뒤 **Create Auto Scaling group** 을 클릭합니다.

2.2.22 오토스케일링이 구성되면 ECS Cluster 에 컨테이너 인스턴스가 1 개 등록됩니다.



## Task 2.3: 배포 자동화를 위한 파이프라인 설정

### Overview

이번 섹션에서는 파이프라인을 설정하여 배포 자동화를 위한 단계 및 수행 액션을 설정합니다.

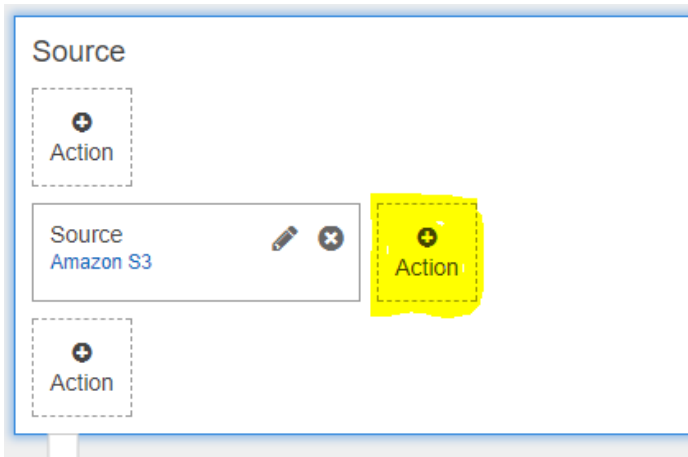
2.3.1 AWS 관리 콘솔에 로그인 후 Services → Developer Tools → CodePipeline 을 선택합니다.

2.3.2 이전에 생성한 파이프라인 **ECSDeployPipeline** 을 선택한 뒤 **Edit** 를 클릭합니다.

2.3.3 Source 스테이지 오른쪽 연필 모양의 수정 버튼을 클릭합니다.



2.3.4 기존에 생성되어있던 Source 액션 오른쪽에 + **Action** 을 클릭하여 액션을 추가합니다.



2.3.5 다음과 같이 설정합니다.

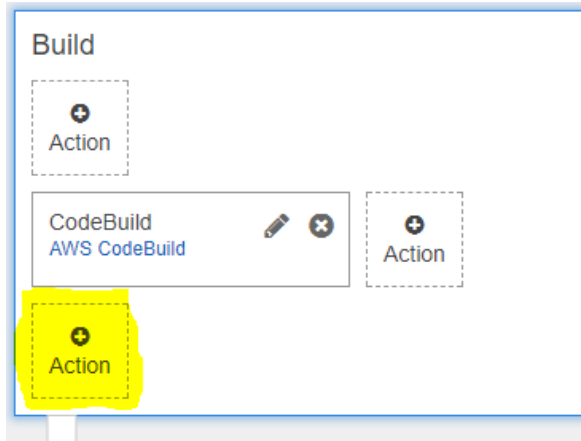
- Action category: Source
- Action name: Resource
- Source provider: Amazon S3
- Amazon S3 location: s3://ecsdeploy-{my\_name}/resource.zip
- Advanced -> Pipeline execution method: I will setup CloudWatch events or run my pipeline manually
- Output artifact #1: Resource

2.3.6 **Add action** 을 클릭하여 액션을 추가합니다.

2.3.7 Save pipeline changes 를 클릭하여 변경사항을 저장합니다.

2.3.8 다시 **Edit** 를 눌러 수정화면으로 들어갑니다.

2.3.9 Build 스테이지의 **수정 버튼**을 클릭한 뒤 기존에 생성한 **CodeBuild 밑에 Action** 을 추가합니다.



2.3.10 다음과 같이 설정합니다:

- Action category: Build
- Action name: Dockerize
- Build provider: AWS CodeBuild
- Create a new build project
- Project name: ECSDeployDockerizer
- Environment image: Use an image managed by AWS CodeBuild
- Operating system: Ubuntu
- Runtime: Docker
- Version: aws/codebuild/docker:1.12.1
- Build specification: Use the buildspect.yml in the source code root directory
- Create a service role in your account
- Role name: code-build-ECSDeployDockerizer-service-role
- Advanced -> Environment variables:
  - Name: REPOSITORY\_URI
  - Value: (이전에 생성한) ECR 리포지토리 URI

2.3.11 **Save build project** 를 클릭하여 빌드 프로젝트를 저장합니다.

2.3.12 추가적으로 Input 및 Output artifact 를 다음과 같이 설정합니다:

- Input artifacts #1: MyAppBuild
- Output artifact #1: BuildInfo

2.3.13 Add action 을 클릭하여 액션을 추가한 뒤 Save pipeline changes 를 클릭합니다.

2.3.14 생성한 IAM Role 을 수정하기 위해 Services → Security, Identity & Compliance → IAM 을 클릭합니다.

2.3.15 왼쪽 패널의 Roles 를 클릭한 뒤 Filter 에 code-build-ECSDeployDockerizer-service-role 를 입력하여 나오는 Role 을 클릭합니다.

2.3.16 하단의 Add inline policy 를 클릭합니다.

2.3.17 Custom Policy 를 선택한 뒤 Select 를 클릭합니다.

2.3.18 다음 내용으로 설정합니다:

- Policy Name: ECSDeployDockerizer-Policy

- Policy Document

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Resource": [
        "*"
      ],
      "Action": [
        "ecr:GetAuthorizationToken"
      ]
    },
    {
      "Effect": "Allow",
      "Resource": [
        "arn:aws:ecr:us-east-1:[ACCOUNT_ID]:repository/*"
      ],
      "Action": [
        "ecr:BatchGetImage",
        "ecr:BatchCheckLayerAvailability",
        "ecr:CompleteLayerUpload",
        "ecr:GetDownloadUrlForLayer",
        "ecr:PutImage",
        "ecr:InitiateLayerUpload",
        "ecr:UploadLayerPart"
      ]
    }
  ]
}
```

2.3.19 **Apply Policy** 를 클릭하여 추가합니다.

2.3.20 Deploy 할 때 필요한 역할을 추가하기 위해 **Create role** 을 클릭합니다.

2.3.21 Select type of trusted entity 에서 AWS Service 박스를 선택한 뒤 Choose the service that will use this role 에서 AWS Cloudformation 을 선택합니다. 다시 하단에서 Select your use case 에서 CloudFormation 을 선택합니다.

2.3.22 Next: Permissions 을 클릭한 뒤 Role name 에 ECSDeployCFNRole 를 입력한 뒤 Create role 을 클릭합니다.

2.3.23 생성한 ECSDeployCFNRole 을 검색하여 선택한 뒤 Inline Policies 를 추가합니다.

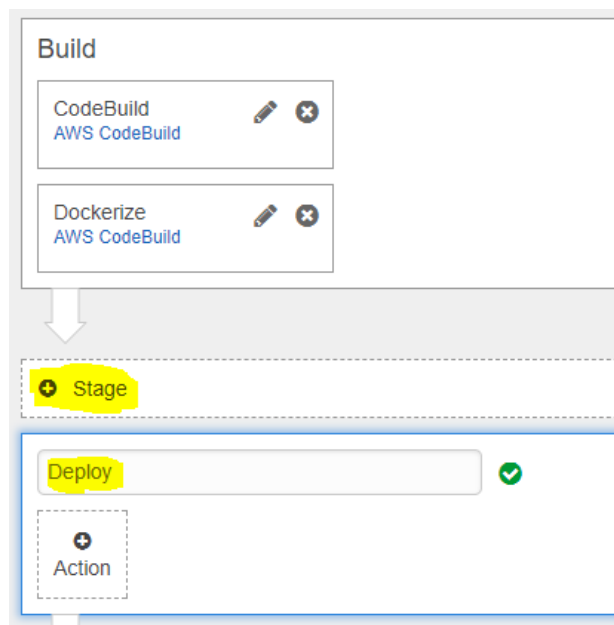
2.3.24 Custom Policy 을 선택한 뒤 다음 내용으로 설정합니다:

- Policy Name: ECSDeployCFNRole-Policy
- Policy Document:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Resource": [
        "*"
      ],
      "Action": [
        "iam:*",
        "ecr:*",
        "ecs:*",
        "ec2:*",
        "elasticloadbalancing:*"
      ]
    }
  ]
}
```

2.3.25 다시 파이프라인에서 **Edit** 를 클릭하여 실제 애플리케이션 배포를 위한 액션을 추가합니다

2.3.26 Build 스테이지 밑에 **Stage +** 를 클릭하여 스테이지를 추가한 뒤 **Deploy** 라고 스테이지 이름을 입력합니다.



2.3.27 **Action** 추가 버튼을 누른 뒤 다음과 같이 설정합니다:

- Action category: Deploy



- Action name: DeployEnvA
- Deployment provider: AWS CloudFormation
- Action mode: Create or update a stack
- Stack name: ECSDeployEnvA
- Template: Resource::cfn/service.yml
- Template configuration: Resource::config/single.conf
- Capabilities: CAPABILITY\_NAMED\_IAM
- Role name: ECSDeployCFNRole
- Advanced:

- Parameter overrides:

```
{
  "Tag": { "Fn::GetParam": [ "BuildInfo", "build.json", "tag" ] }
}
```

- Input artifacts #1: Resource, BuildInfo

2.3.28 Save pipeline changes 를 클릭하여 변경사항을 저장합니다.

## Task 3: 배포 수행

### Overview

---

이번 Task에서는 구성된 배포 자동화를 통해 배포를 직접 수행해 봅니다.

참고) 원활한 lab 진행을 위해 필요로 하는 최소한의 설정으로만 진행합니다. 실제 서비스를 위한 환경에서는 접근 제어와 같은 부분에 대해서 더 세심하게 설정하여야 합니다.

## Task 3.1: 환경 설정 파일 수정

### Overview

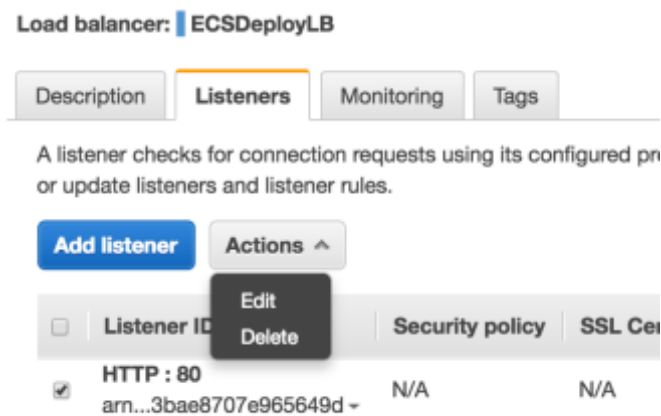
이번 섹션에서는 환경 설정 파일을 수정하여 애플리케이션 배포를 준비합니다. 배포는 CloudFormation 을 통해 진행되며 생성된 스택이 로드밸런서의 Listener 와 ECS 의 Task, Service 등을 관리하게 됩니다. 이를 위해 로드밸런서 생성 시 추가하였던 Listener 를 삭제하여 CloudFormation 스택이 추가 및 삭제할 수 있도록 합니다

**참고) Listener 를 삭제하지 않으면, CloudFormation 스택이 관리할 Listener 를 생성하는 중 충돌로 인하여 스택 생성 실패가 일어납니다.**

3.1.1 관리자 콘솔에서 Services → Compute → EC2 를 클릭 합니다.

3.1.2 왼쪽 패널의 **Load Balancers** 를 클릭한 뒤 이전에 생성한 **ECSDeployLB** 를 선택합니다.

3.1.3 Listeners 에서 등록되어있는 **HTTP : 80** 을 선택한 뒤 **Actions → Delete** 를 클릭합니다.



3.1.4 애플리케이션 및 리소스 파일을 다운로드 받습니다.

- <http://bit.ly/2wcAcGf>

3.1.5 resources/config/single.conf 파일을 열어 수정합니다.

- VpcId: 이전에 생성한 VPC ID
- Repository: ECR 리포지토리
- Cluster: ECS 클러스터명
- LoadBalancerArn: 로드밸런서 ARN

- DesiredCount: 인스턴스 갯수
- EnvType: 환경 타입
- Port: 서비스 포트

## Task 3.2: 애플리케이션 배포

### Overview

---

이번 섹션에서는 환경 설정 파일 및 애플리케이션을 S3 에 업로드하여 실제 배포를 수행합니다.

3.2.1 resources 폴더 내의 두 폴더 (config, cfn) 를 resource.zip 으로 압축합니다

**주의) 폴더 내의 두 폴더를 선택한뒤 압축을 하여야 합니다. resources 폴더를 압축하는게 아닙니다.**

3.2.2 build 폴더, buildspec.yml, main.go 파일을 선택하여 **app.zip** 으로 압축합니다.

3.2.3 생성해둔 버킷에 두 파일 (resource.zip, app.zip) 을 업로드 합니다.

3.2.4 CodePipeline 에서 변경사항을 감지하여 배포 작업을 수행하는지 확인합니다.

3.2.5 CloudFormation 을 통해 배포가 완료되면 로드밸런서의 DNS 를 통해 웹 페이지를 접속하면 배포된 애플리케이션을 확인할 수 있습니다.

## Task 3.3: 애플리케이션 수정 및 배포

### Overview

---

이번 섹션에서는 실제 애플리케이션을 수정한 뒤 다시 배포하여 정상적으로 배포되는지 확인 해봅니다.

- 3.3.1 다운로드 받은 애플리케이션 및 리소스 파일 중 **main.go** 파일을 에디터에서 불러옵니다
- 3.3.2 `const version string = "0.0.1 (AWS Korea)"` 를 **"0.0.2 (AWS Korea)"** 로 수정합니다.
- 3.3.3 build 폴더, buildspec.yml, main.go 파일을 선택하여 **app.zip** 으로 압축합니다.
- 3.3.4 생성해둔 버킷에 app.zip 파일을 업로드 합니다.
- 3.3.5 CodePipeline 에서 변경사항을 감지하여 배포 작업을 수행하는지 확인합니다.
- 3.3.6 CloudFormation 을 통해 배포가 완료되면 로드밸런서의 DNS 를 통해 웹 페이지를 접속하면 배포된 애플리케이션을 확인할 수 있습니다

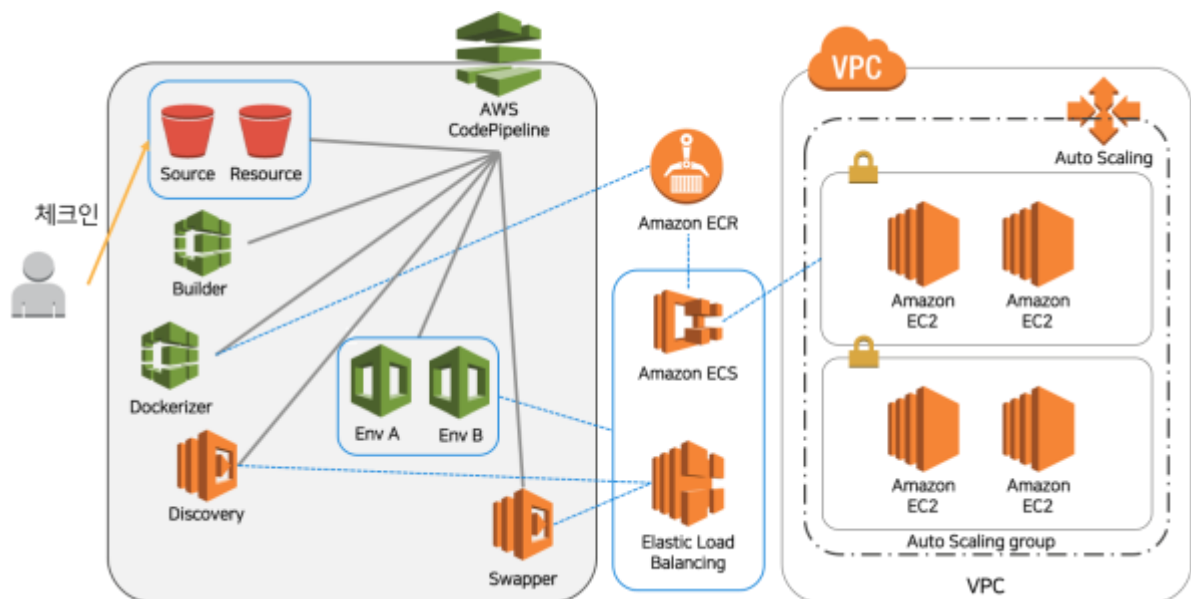
## 실습 2

# Amazon ECS 로 애플리케이션 Blue/Green 배포

### Overview

이 실습은 AWS 가 제공하는 컨테이너 클러스터, 개발자 도구 등을 활용하여 컨테이너로 구성된 애플리케이션을 배포함에 있어 Blue/Green 배포 방식으로 배포하는 방법을 알아보고 실습해 볼 것 입니다.

실습 1 에서 구성한 환경을 기반으로 진행하기 때문에, 실습 1 을 완료하여야 실습 2 를 진행할 수 있습니다 (또는 제공된 CloudFormation 템플릿을 통해 실습 1 의 환경을 구성한 뒤 진행하실 수 있습니다).



<그림 1. 실습 2 아키텍처>

### Objectives

이 실습을 완료하면 다음을 할 수 있습니다.

- AWS 에서 Blue/Green 배포 방법을 어떻게 적용할 수 있는지 알 수 있습니다.

- 컨테이너 클러스터를 대상으로 Blue/Green 배포 방법을 적용할 수 있습니다.
- AWS Lambda 를 배포 자동화에 활용할 수 있습니다.
- AWS CodePipeline 의 파이프라인 실행을 제어할 수 있습니다.

## Prerequisites

---

Lab 을 완료하기 위해 다음이 필요합니다:

- Wi-Fi 사용기 가능한 노트북 컴퓨터 (Mac OS X, Linux, Windows)
- Windows 사용자는 Administrator 권한이 필요
- Windows 사용자는 Putty 필요
- Chrome, Firefox, Safari 등의 인터넷 브라우저

## Duration

---

40 분



## Task 1: Discovery 구성

### Overview

---

컨테이너에 두개의 환경을 구성한 뒤 Blue/Green 배포를 하기 위해선 현재 어떤 환경이 Production 환경인지 아닌지 파악할 필요가 있습니다. 이를 위해 AWS Lambda 를 통해 Discovery 를 구성합니다. Discovery 는 로드밸런서에 연결된 Listener 의 정보를 가져오고 그 중 포트 정보를 기반으로 Production 인지 아닌지 판단합니다 (80: Production, 8080: Staging).

## Task 1.1: Discovery Lambda 함수 생성

### Overview

---

Discovery Lambda 함수를 생성하고 기존의 파이프라인에 추가합니다.

1.1.1 관리자 콘솔에서 **Services → Compute → Lambda** 를 클릭 합니다.

1.1.2 **Create function** 을 클릭합니다.

1.1.3 **Author from scratch** 를 선택하고 다음 내용으로 함수를 설정합니다.

- Name: ECSDeployDiscovery
- Role: Create a custom role 선택한 뒤 뜨는 새 창에서 아래 내용으로 설정합니다.
  - IAM Role: Create a new IAM Role
  - Role name: ECSDeployLambda-Role
  - Policy Document: Edit 를 클릭한 뒤 아래 내용 입력

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup", "logs:CreateLogStream", "logs:PutLogEvents"
      ],
      "Resource": "arn:aws:logs:*:*:*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "elasticloadbalancing:Describe*", "elasticloadbalancing:ModifyRule",
        "codepipeline:AcknowledgeJob", "codepipeline:GetJobDetails",
        "codepipeline:PollForJobs", "codepipeline:PutJobFailureResult",
        "codepipeline:PutJobSuccessResult",
        "s3:PutObject", "s3:PutObjectAcl", "s3:GetObject", "s3:GetObjectAcl",
        "s3:ListBucket", "s3:ListObjects"
      ],
      "Resource": [ "*" ]
    }
  ]
}
```

- Allow 클릭합니다.

1.1.4 **Create function** 을 클릭합니다.

1.1.5 함수가 생성되면 아래 내용으로 함수를 설정합니다:

- Runtime: Python 3.6
- Handler: index.lambda\_handler
- Lambda function code: <http://bit.ly/2gpRybW>
- Basic settings:
  - Timeout: 10sec

1.1.6 상단의 **Save** 를 클릭하여 함수를 저장합니다.

## Task 1.2: 파이프라인 수정

### Overview

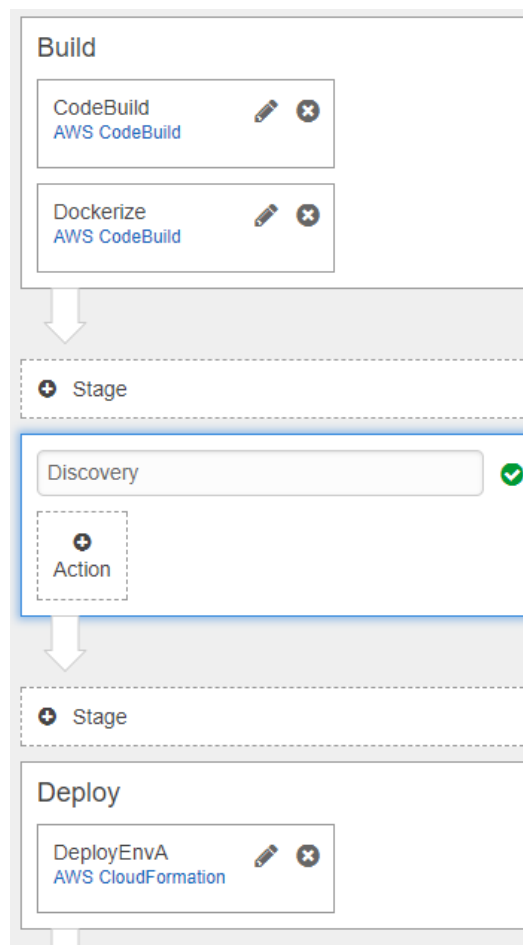
---

파이프라인을 수정해서 Discovery 부분을 추가합니다.

1.2.1 생성해둔 파이프라인 **ECSDeployPipeline** 페이지로 이동합니다.

1.2.2 **Edit** 를 클릭합니다.

1.2.3 Build 스테이지와 Deploy 스테이지 사이에 **Stage** 를 추가한 뒤 **Discovery** 라고 이름을 입력합니다.



1.2.4 **Action 추가 버튼**을 클릭한 뒤 다음 내용으로 설정합니다:

- Action category: Invoke
- Action name: Discovery
- Provider: AWS Lambda
- Function name: ECSDeployDiscovery
- User parameters: ECSDeployLB

- Input artifacts: BuildInfo, Resource
- Output artifacts: DiscoveryInfo

1.2.5 **Add action** 을 클릭하여 액션을 추가합니다.

1.2.6 **Save pipeline changes** 를 클릭하여 파이프라인을 저장합니다.

## Task 2: 추가 환경 구성

### Overview

---

Blue/Green 에는 Blue 와 Green 두개의 환경이 필요합니다. 그렇기때문에 추가로 하나의 환경을 더 구성합니다.

## Task 2.1: 기존 환경 A 수정

### Overview

---

기존에 파이프라인에 구성되어 있는 환경은 단일 환경을 위함 입니다. Blue/Green 배포는 두개의 환경을 사용하므로, 두개의 환경에 맞도록 기존 환경에 대한 설정을 수정합니다.

2.1.1 파이프라인 ECSDeployPipeline 에서 Edit 를 클릭합니다.

2.1.2 Deploy 스테이지의 수정 버튼을 클릭한 뒤 DeployEnvA 의 수정 버튼을 클릭합니다.

2.1.3 다음 부분을 수정합니다:

- Template configuration: Resource::config/env\_a.conf
- Advanced:

- Parameter overrides:

```
{  
  "Tag": { "Fn::GetParam": [ "DiscoveryInfo", "env_info.json", "EnvA" ] }  
}
```

- Input artifacts: Resource, DiscoveryInfo

2.1.4 **Update** 를 클릭하여 액션을 추가 합니다.

## Task 2.2: 새로운 환경 B 추가

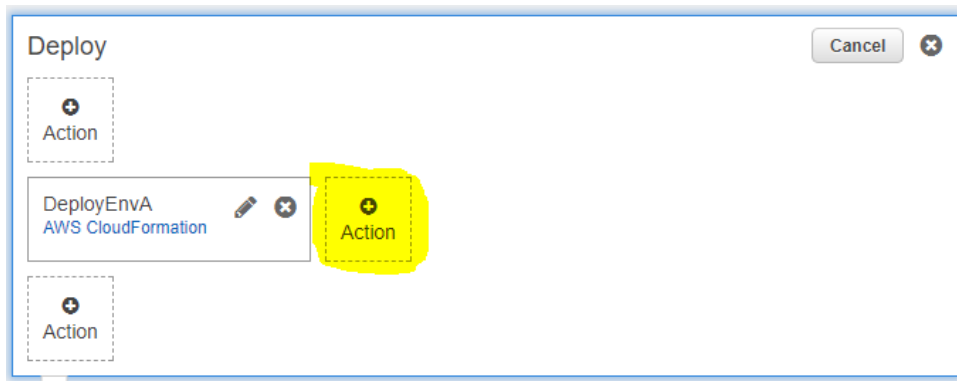
### Overview

---

새로운 환경 B 를 추가 구성합니다.

2.2.1 파이프라인 ECSDeployPipeline 에서 Edit 를 클릭합니다.

2.2.2 Deploy 스테이지의 수정 버튼을 클릭한 뒤 DeployEnvA 옆에 Action 추가 버튼을 누릅니다.



2.2.3 다음과 같이 설정합니다:

- Action category: Deploy
- Action name: DeployEnvB
- Deployment provider: AWS CloudFormation
- Action mode: Create or update a stack
- Stack name: ECSDeployEnvB
- Template: Resource::cfn/service.yml
- Template configuration: Resource::config/env\_b.conf
- Capabilities: CAPABILITY\_NAMED\_IAM
- Role name: ECSDeployCFN-Role
- Advanced:

- Parameter overrides:

```
{
  "Tag": { "Fn::GetParam": [ "DiscoveryInfo", "env_info.json", "EnvB" ] }
}
```

- Input artifacts: Resource, DiscoveryInfo

2.2.4 **Add action** 을 클릭하여 액션을 추가 합니다.



## Task 3: 수동 승인 기능 추가

### Overview

---

일반적으로 배포 수행 시 변경사항이 올바르게 작동하는지 검증을 수행합니다. 코드 수정 후 리포지토리에 체크인 한 뒤, 빌드 및 테스트가 수행되고 스테이징 환경에 배포 됩니다. 이때 기대한 대로 작동하는지 검증이 필요하며, 검증 프로세스를 거친 뒤 승인이 나면 실제 프로덕션으로 배포하게 하는 것이 일반적입니다. 이를 위해 수동으로 승인할 수 있는 기능을 추가합니다.

## Task 3.1: 수동 승인 추가

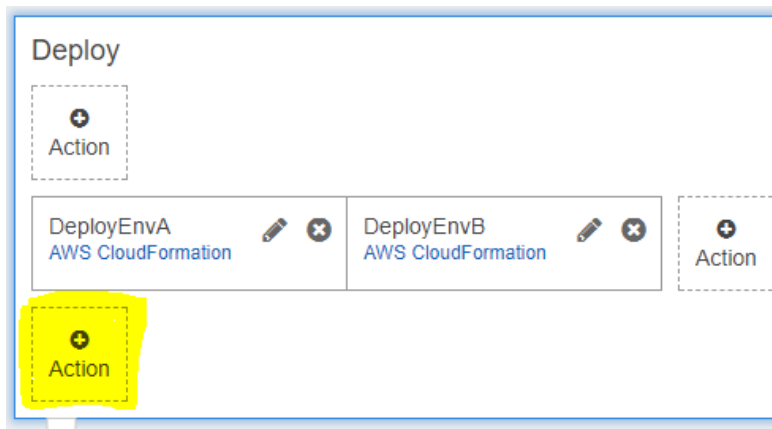
### Overview

---

파이프라인을 수정하여 수동 승인을 추가합니다. 보통 이메일이나 프로젝트 관리 시스템과 연동하여 승인 요청 알림을 받을 수 있게 구성하지만, 이 실습에서는 단순히 승인만 기다리도록 설정합니다.

3.1.1 파이프라인 ECSDeployPipeline 에서 Edit 를 클릭합니다.

3.1.2 Deploy 스테이지의 수정 버튼을 클릭한 뒤 두개의 환경 밑에 Action 추가 버튼을 클릭합니다.



3.1.3 다음과 같이 설정합니다:

- Action category: Approval
- Action name: SwapApproval
- Approval type: Manual approval
- Comments: Do you approve of this deployment?

3.1.4 **Add action** 을 클릭하여 추가합니다.

3.1.5 **Save pipeline changes** 를 클릭하여 파이프라인을 저장합니다.

## Task 4: Blue/Green 환경 Swap 기능 추가

### Overview

---

코드를 수정한 뒤 리파지토리에 체크인 하게 되면 파이프라인의 각 단계에 따라 배포가 진행됩니다. 배포가 진행이 되면 코드로부터 빌드를 만들게 되고, 만들어진 빌드가 스테이징 환경에 배포되어 실제 수정된 애플리케이션이 동작하는 것을 확인할 수 있습니다. 이후 검증을 진행하고, 검증이 통과되면 현재 이전 애플리케이션이 동작하는 프로덕션과 새로운 애플리케이션이 배포된 스테이징 환경을 Swap 해주어야 합니다. Lambda 함수를 통해 Swap 을 진행하여, 실제 다운타임 없이 새로운 버전으로 배포하는 Blue/Green 배포를 실습합니다.

## Task 4.1: Lambda 함수 추가

### Overview

---

스테이징 환경과 프로덕션 환경 Swap 을 위해 Lambda 함수를 추가합니다.

4.1.1 관리자 콘솔에서 **Services → Compute → Lambda** 를 클릭 합니다.

4.1.2 **Create function** 을 클릭합니다.

4.1.3 **Author from scratch** 를 선택하고 다음 내용으로 함수를 설정합니다.

- Name: ECSDeploySwapper
- Role: Choose an existing role
- Existing role: ECSDeployLambda-Role

4.1.4 **Create function** 을 클릭합니다.

4.1.5 함수가 생성되면 아래 내용으로 함수를 설정합니다:

- Runtime: Python 3.6
- Handler: index.lambda\_handler
- Lambda function code: <http://bit.ly/2gvEiX1>
- Basic settings:
  - Timeout: 10sec

4.1.6 상단의 **Save** 를 클릭하여 함수를 저장합니다.

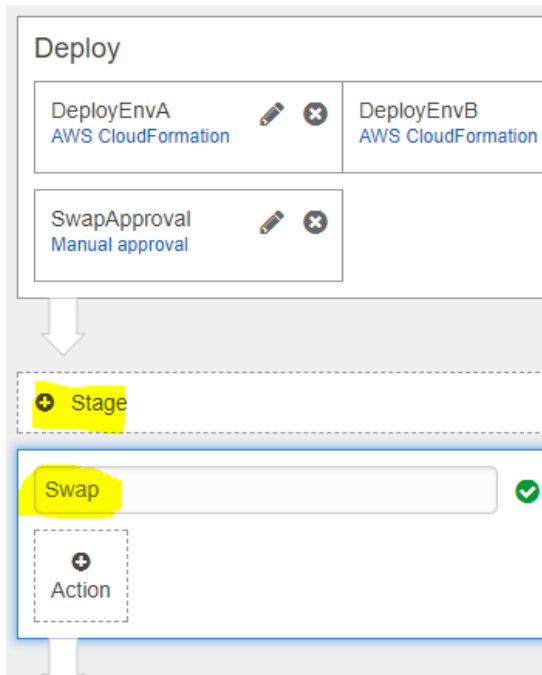
## Task 4.2: 파이프라인 수정

### Overview

Swap Lambda 함수가 실행될 수 있도록 배포 자동화 파이프라인을 수정합니다.

4.2.1 파이프라인 ECSDeployPipeline 에서 Edit 를 클릭합니다.

4.2.2 Deploy 스테이지 밑 Stage 추가 버튼을 클릭한 뒤 Swap 이라고 이름을 입력합니다.



4.2.3 **Action 추가 버튼**을 클릭한 뒤 다음과 같이 설정합니다:

- Action category: Invoke
- Action name: SwapEnv
- Provider: AWS Lambda
- Function name: ECSDeploySwapper
- User parameters: ECSDeployLB

4.2.4 **Add action** 을 클릭하여 추가합니다.

4.2.5 Save pipeline changes 를 클릭하여 파이프라인을

4.2.6 **Add action** 을 클릭하여 추가합니다.

## Task 5: 애플리케이션 배포 테스트

### Overview

---

구성한 환경이 기대한 대로 작동하는지 테스트합니다.

## Task 5.1: 환경 설정 파일 수정 및 애플리케이션 수정

### Overview

---

준비되어있는 2 개의 환경에 대한 환경 설정 파일을 수정하고, 애플리케이션을 수정합니다.

5.1.1 resources/config/env\_a.conf 파일을 열어 수정합니다.

- VpcId: 이전에 생성한 VPC ID
- Repository: ECR 리포지토리
- Cluster: ECS 클러스터명
- LoadBalancerArn: 로드밸런서 ARN
- DesiredCount: 인스턴스 갯수
- EnvType: **EnvA**
- Port: **80**

5.1.2 resources/config/env\_b.conf 파일을 열어 수정합니다.

- VpcId: 이전에 생성한 VPC ID
- Repository: ECR 리포지토리
- Cluster: ECS 클러스터명
- LoadBalancerArn: 로드밸런서 ARN
- DesiredCount: 인스턴스 갯수
- EnvType: **EnvB**
- Port: **8080**

5.1.3 main.go 파일을 열어 수정합니다:

- const version string = "0.0.2 (AWS Korea)" → "Blue/Green 0.0.1 (AWS Korea)"

## Task 5.2: 환경 파일 및 애플리케이션 배포

### Overview

---

수정한 환경 설정 파일과 애플리케이션을 S3 를 통해 배포합니다. 배포한 뒤 파이프라인을 통해 적용 과정을 확인하고, 최종적으로 로드밸런서의 DNS 를 통해 접속하여 기대한 결과가 나오는지 확인합니다.

5.2.1 resources 폴더 내의 두 폴더 (config, cfn) 을 resource.zip 으로 압축합니다

**주의) 폴더 내의 두 폴더를 선택한뒤 압축을 하여야 합니다. resources 폴더를 압축하는게 아닙니다.**

5.2.2 build 폴더, buildspect.yml, main.go 파일을 선택하여 app.zip 으로 압축합니다.

5.2.3 생성해둔 버킷에 두 파일 (resource.zip, app.zip) 을 업로드 합니다.

5.2.4 CodePipeline 에서 변경사항을 감지하여 배포 작업을 수행하는지 확인합니다.

5.2.5 CloudFormation 을 통해 배포가 완료되면 로드밸런서의 DNS 를 통해 웹 페이지를 접속하면 배포된 애플리케이션을 확인할 수 있습니다.



## Task 5.3: 변경사항 확인 및 승인

### Overview

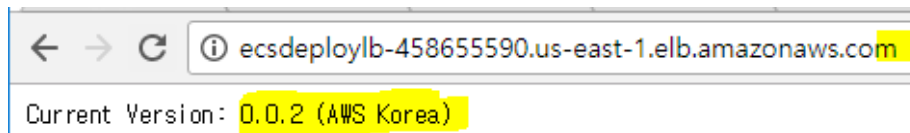
이전 섹션에서 환경 파일과 애플리케이션을 수정하여 배포하였습니다. 이제 제대로 적용되었는지 확인을 합니다. 제대로 적용되었다면 승인을 한 뒤 Production 에 변경사항이 적용되는지 확인합니다.

5.3.1 파이프라인에서 변경사항을 감지하여 배포 작업을 수행하는지 확인합니다.

5.3.2 승인 이전 단계까지 모두 정상적으로 수행되었다면 **로드밸런서의 DNS 를 통해 애플리케이션에 접속합니다.**

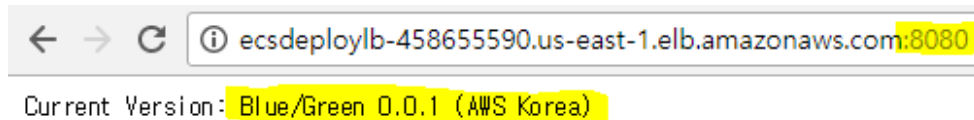
5.3.3 **80 번 포트로 접속** 시 그 이전의 애플리케이션의 결과가 출력됩니다.

- Current Version: 0.0.2 (AWS Korea)



5.3.4 **8080 번 포트로 접속** 시 새로 변경한 애플리케이션의 결과가 출력됩니다.

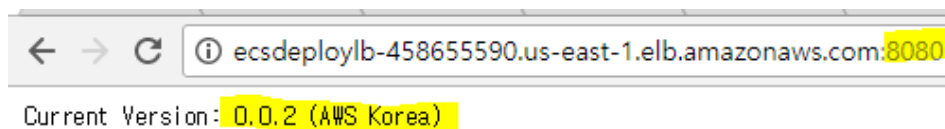
- Current Version: Blue/Green 0.0.1 (AWS Korea)



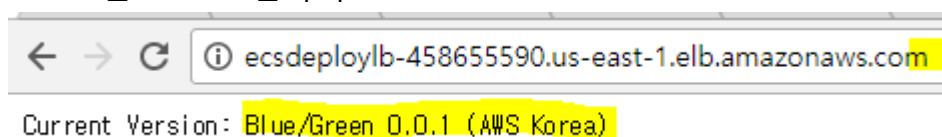
5.3.5 Review 를 클릭한 뒤 Comments 입력과 Approve 버튼을 클릭하여 SwapEnv 액션을 실행시킵니다.

5.3.6 액션이 완료된 뒤 **80 번 포트와 8080 번 포트에 접속**하여, 정상적으로 환경이 교체되었는지 확인합니다.

- 8080 번 포트로 접속시:



- 80 번 포트로 접속시



주의) 모든 실습이 완료된 뒤 사용하셨던 리소스를 삭제하여야지만 추후 추가적인 비용이 발생하지 않습니다.