Builders Online - DevOps

AWS CDK Hands-on Lab

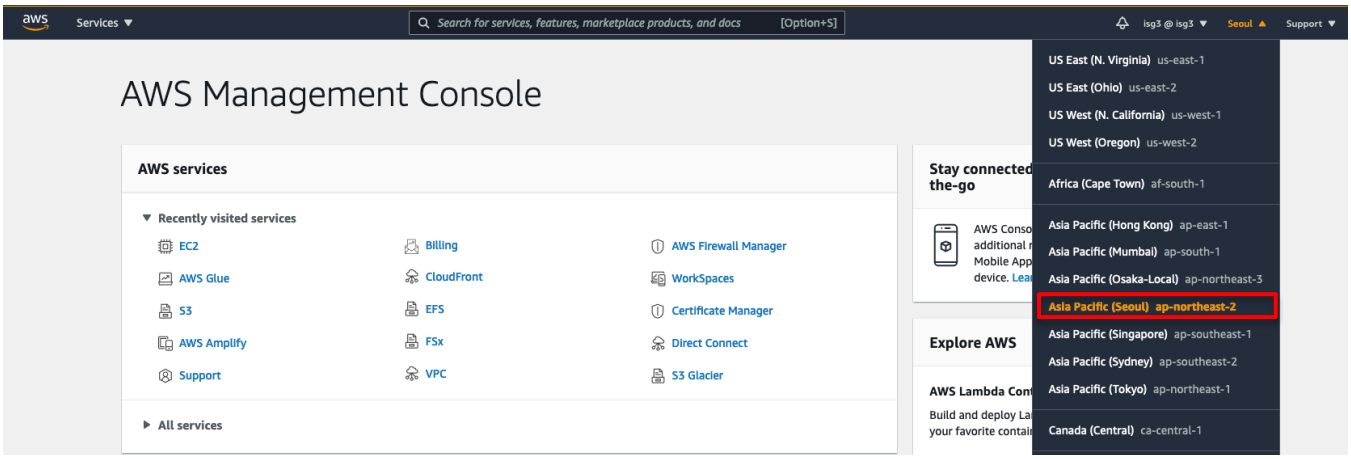
June 23, 2022

Table of Contents

1. 사전 준비 사항

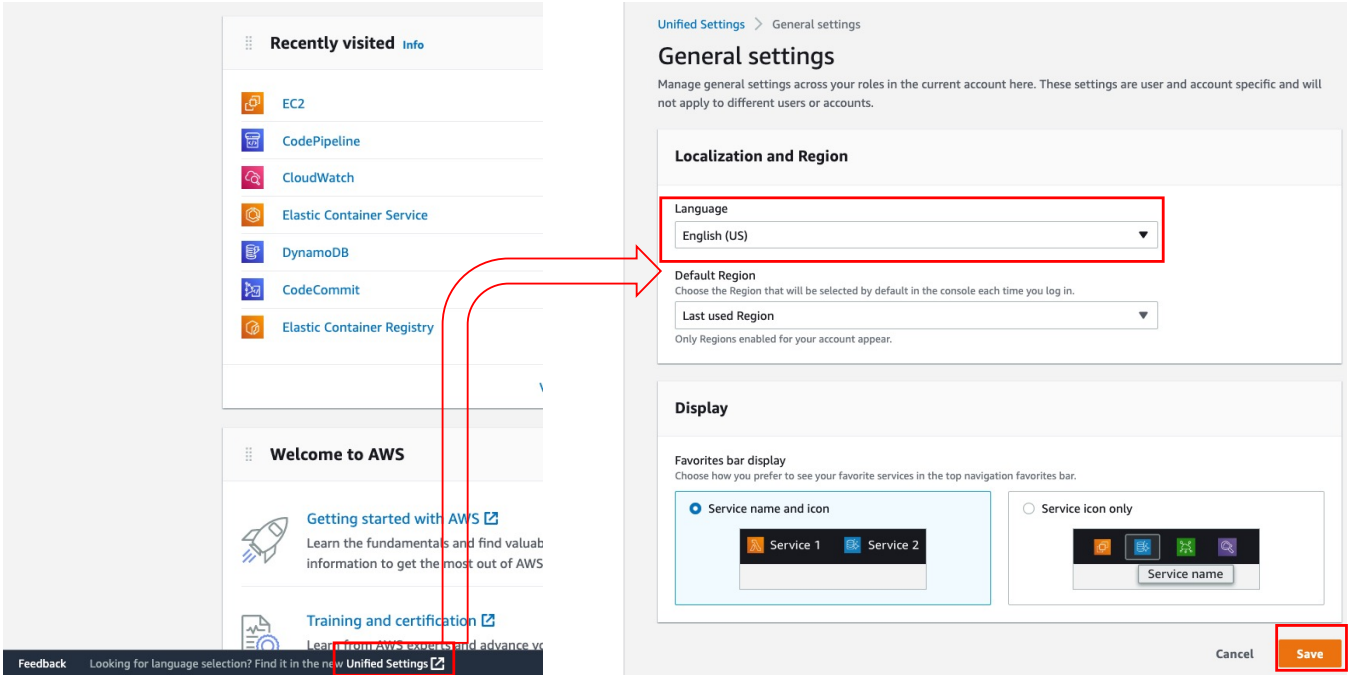
1-1. 실습 리전

본 실습은 **서울 리전**(ap-northeast-2)에서 진행합니다.



1-2. 실습 언어

본 실습 문서는 영어 AWS 관리 콘솔을 기준으로 작성되었습니다. 관리 콘솔 아래쪽에 있는 언어 선택 메뉴를 통해 원하시는 언어로 전환할 수 있습니다.



2. 실습 환경 설정

실습에 참여하시는 여러분의 환경에 맞게 아래 2 가지 방법 중에 하나의 방법으로 실습 환경을 설정한 후 진행하세요.

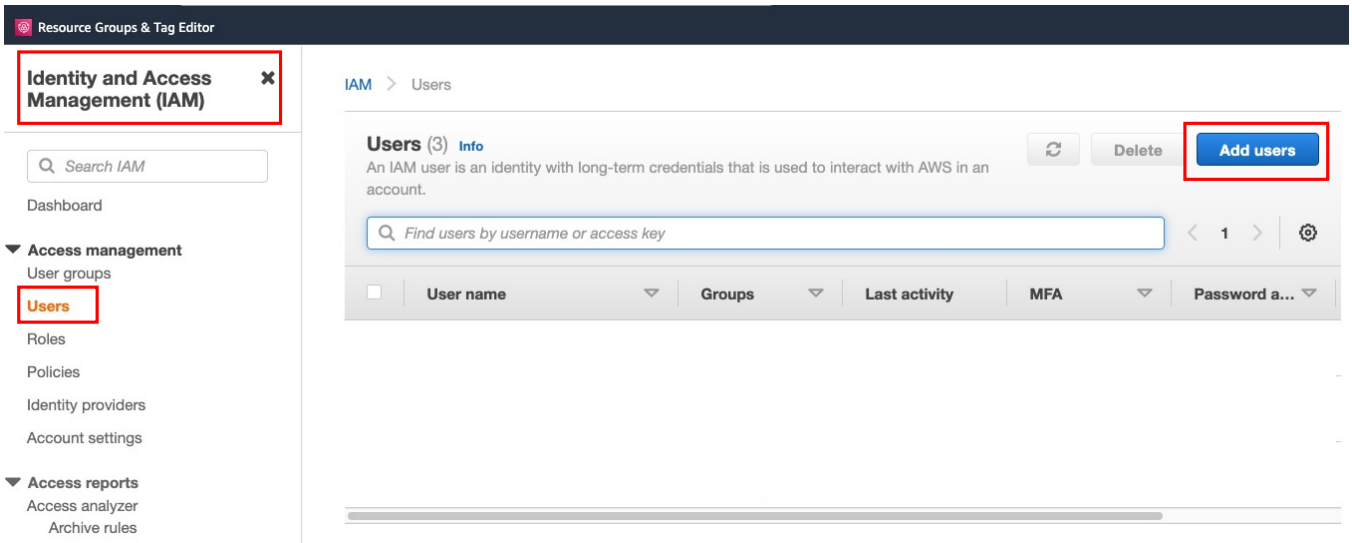
Case1: 자신의 로컬 개발 PC 에서 직접 실습하는 경우(만약 자신의 로컬 개발 PC 가 맥이나 리눅스 계열이라면 권장 드립니다.)
Case2: AWS Cloud9 으로 실습하는 경우(만약 자신의 로컬 개발 PC 가 윈도우즈라면 이 방식을 적극 권장 드립니다.)

2-1. Case1: 자신의 로컬 개발 PC 에서 직접 실습하는 경우

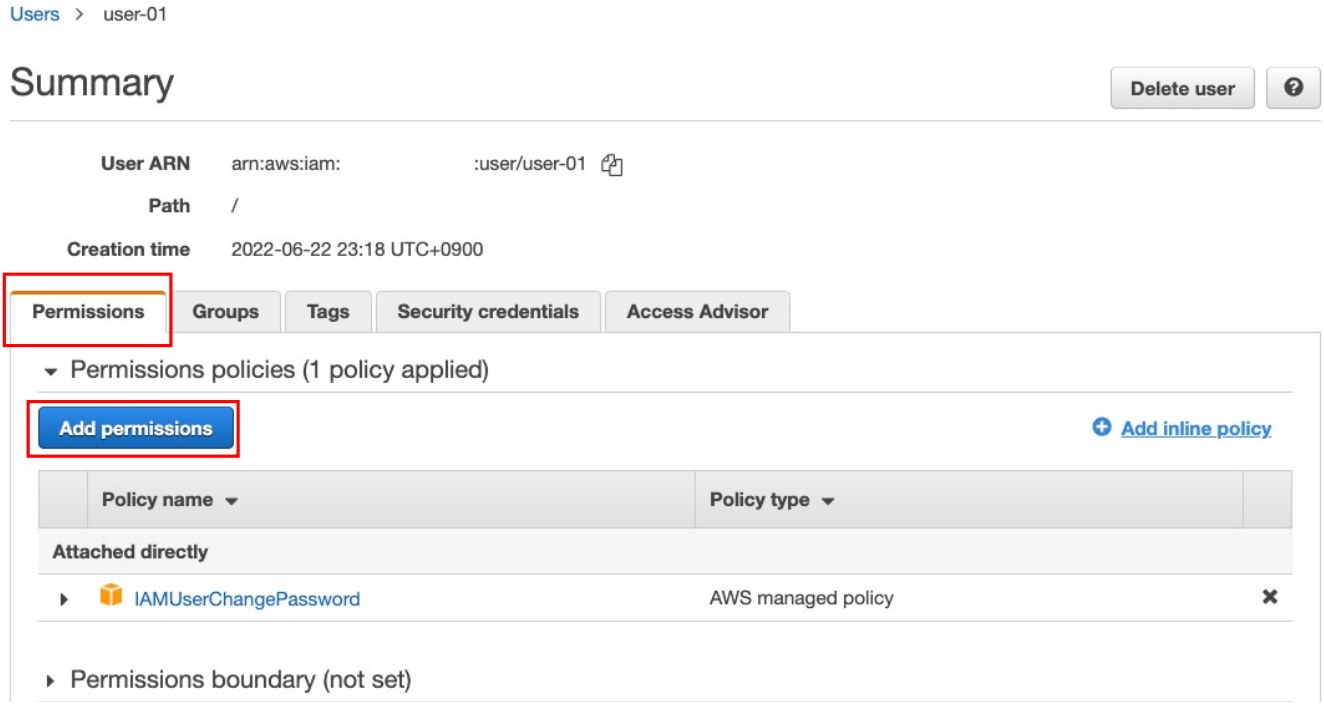
본 실습을 위해서 AWS IAM User 로 등록이 되어 있어야 하며, IAM User 에 대한 Credential 이 생성되어 개발자 PC 에 "aws configure –profile [your-profile]"로 설정이 되어 있어야 합니다. 만약 이미 설정이 되었다면 3 AWS CDK 환경 설정으로 넘어가세요.

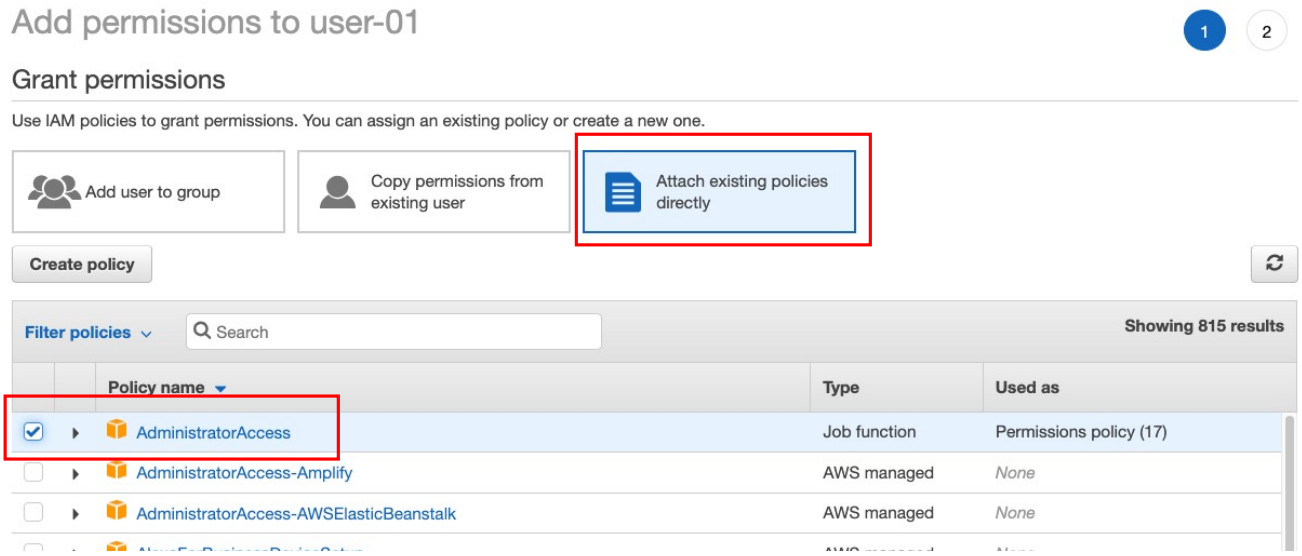
2-1-1. IAM User 생성

AWSWS 관리 콘솔(http://console.aws.amazon.com)에 접속 후 IAM→Users→Add users 를 통하여 신규 IAM 사용자를 등록해주세요. Programmatic access, AWS Management Console access 가 모두 가능해야 합니다.



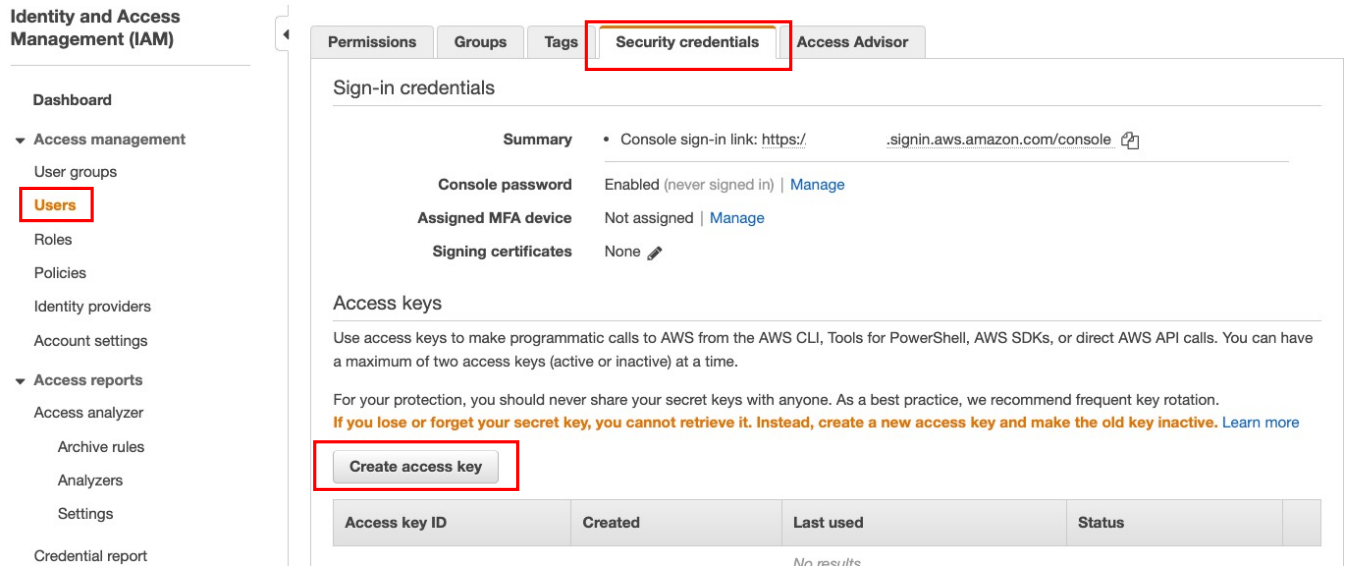
다음과 같이 신규로 생성한 IAM User 에게 오직 실습 편의를 위해서 "" 권한을 부여해주세요. 실제 권장 사항은 아닙니다





2-1-2. 사용자 Credential 생성

자신의 IAM User 를 선택 후에 "Security credentials"로 이동하여, credential 을 생성합니다. "Access key ID"와 "Secret access key"를 잘 메모해두세요(또는 csv 파일은 다운로드하세요). 오직 지금 한번만 확인 가능한 값입니다.



2-1-3. 개발 PC 에 설정

"AWS CLI"가 설치되어 있다고 가정고 진행합니다. 만약 설치되어 있지 않으시다면 https://cdkworkshop.com/15-prerequisites/100-awscli.html 안내를 통해서 설치하세요.

다음 명령어를 입력하여 자신의 로컬 개발 PC 에 credential 을 설정해주세요. 가급적 profile 을 명시하는 것이 좋은 습관입니다.

명령어: `aws configure --profile [your-profile]`

```
└▪ aws configure --profile cdk-work
AWS Access Key ID [None]: sssss
AWS Secret Access Key [None]: xxxxxx
Default region name [None]: ap-northeast-2
Default output format [None]: json
```

2-1-4. 계정 설정 확인하기

다음과 같은 명령어를 통하여 자신 계정과 iam user 가 맞는지 확인합니다..

명령어: `aws sts get-caller-identity --profile [your-profile]`

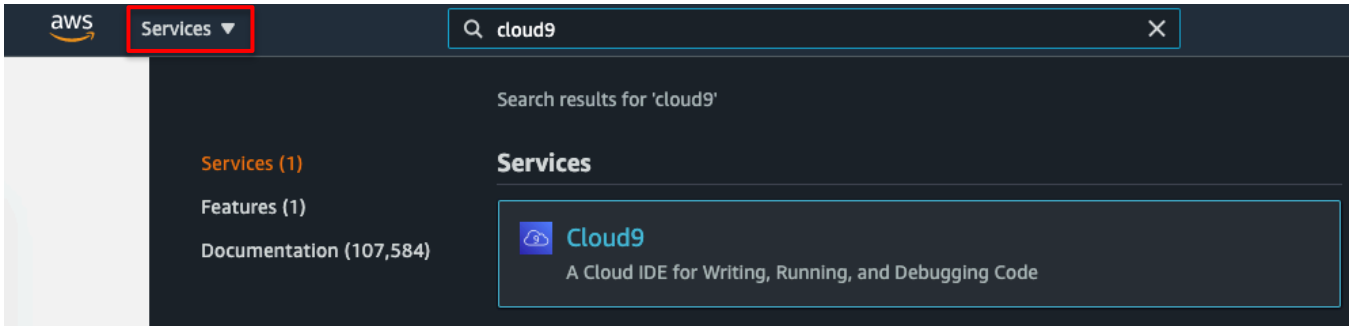
```
└■ aws sts get-caller-identity --profile cdk-demo
{
    "UserId": "AIDA257JKJDEGHNAFVEPC",
    "Account": "  your-account  ",
    "Arn": "arn:aws:iam:  your-account  :user/  your-user
}
```

2-2. Case2: AWS Cloud9 으로 실습하는 경우

AWS Cloud9 은 브라우저만으로도 코드를 작성, 실행 및 디버깅할 수 있는 IDE 입니다. 코드 편집기, 디버거 및 터미널이 포함되어 있으며 많이 사용되는 프로그래밍 언어를 위한 필수 도구가 사전에 패키징되어 제공되므로, 새로운 프로젝트를 시작하기 위해 파일을 설치하거나 개발 머신을 구성할 필요가 없다는 특징을 가지고 있습니다.

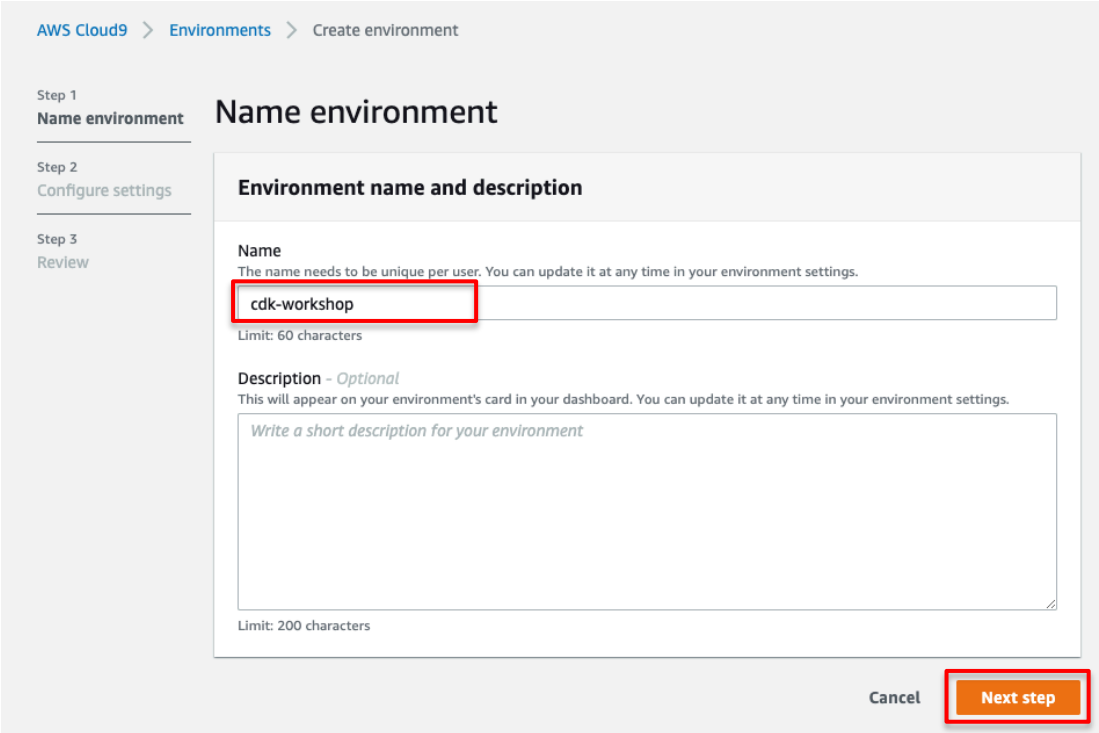
2-2-1. AWS Cloud9 생성하기

AWS 콘솔 좌측 상단에 있는 Services 메뉴를 클릭한 후, 검색창에 Cloud9 을 입력한 후, 아래 화면과 같은 결과가 나오면 서비스를 선택합니다.



Create environment 버튼을 누르고 화면과 같이 진행합니다.

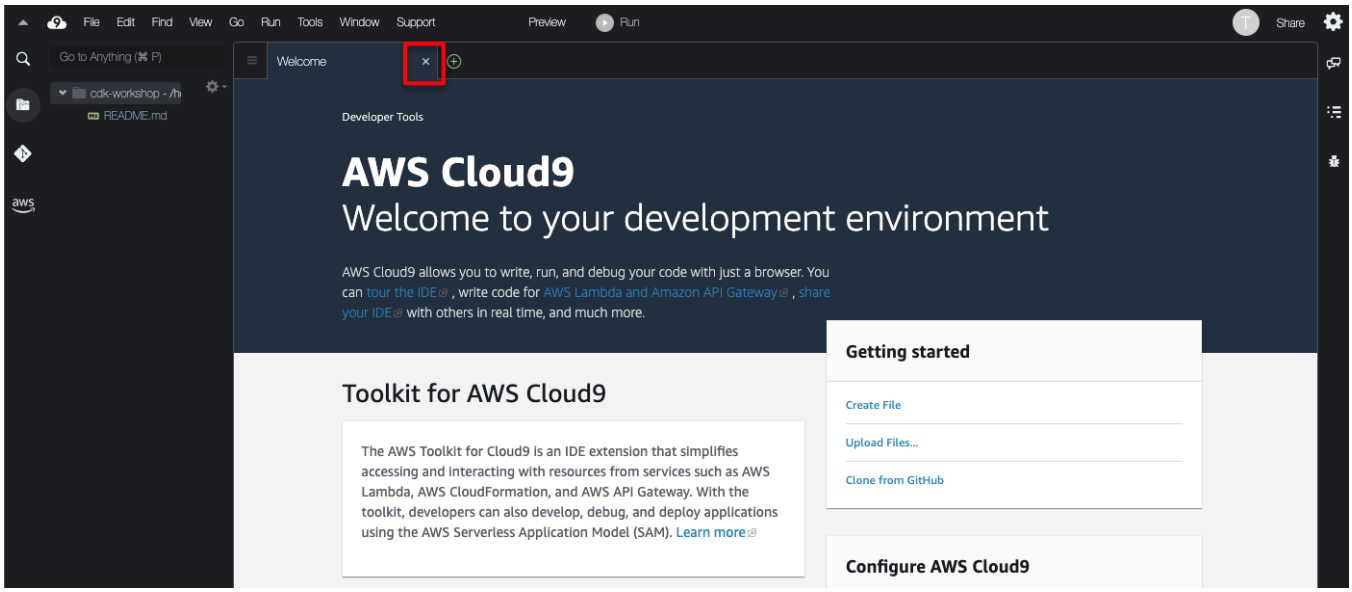
Name 에 **cdk-workshop** 을 입력하고 Next step 으로 갑니다.



다음 페이지의 모든 설정을 기본 값으로 둔 후, 다음 단계로 이동합니다. 기본 값은 아래를 참고합니다.

Environment type	Create a new EC2 instance for environment(direct access)
Instance type	t2.micro (1 GiB RAM + 1 vCPU)
Platform	Amazon Linux 2 (recommended)

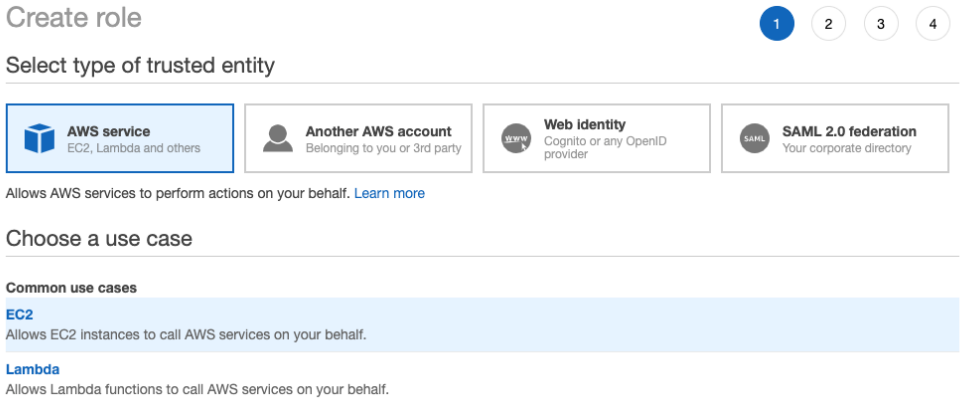
마지막 페이지에서 설정 값을 확인한 후, Create environment 버튼을 클릭하여 Cloud9 실습 환경을 구축합니다.



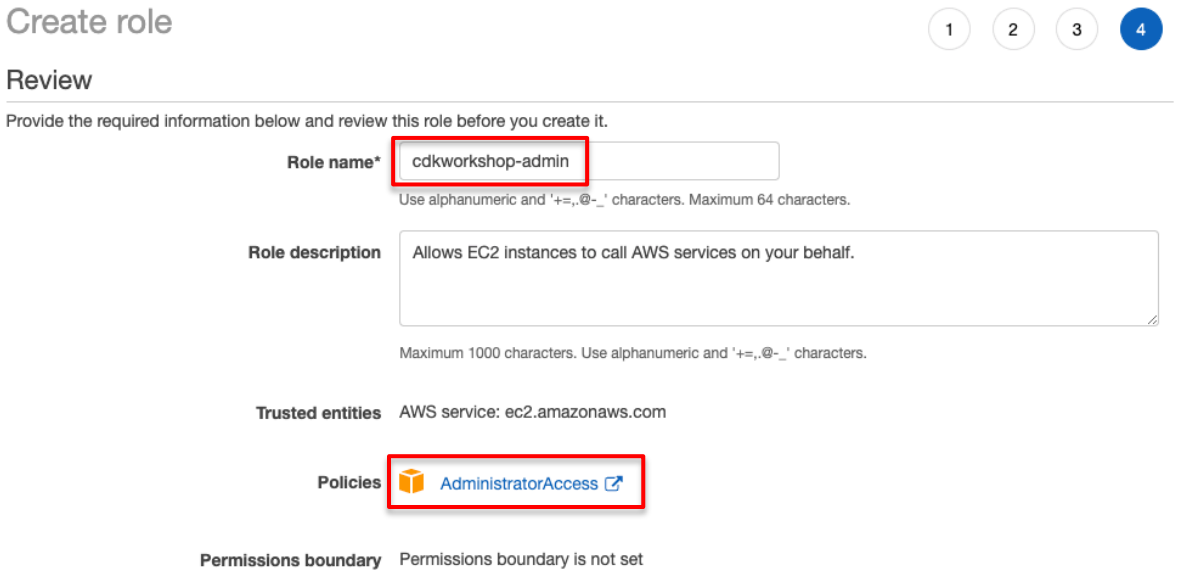
2-2-2. Cloud9 에 IAM Role 부여하기

본 실습에서는 Cloud9 실습 환경에 IAM Role 을 부여하기 위해, 여기(https://console.aws.amazon.com/iam/home#/roles$new?step=type&commonUseCase=EC2%2BEC2&selectedUseCase=EC2&policies=arn:aws:iam::aws:policy%2FAdministratorAccess)를 클릭합니다.

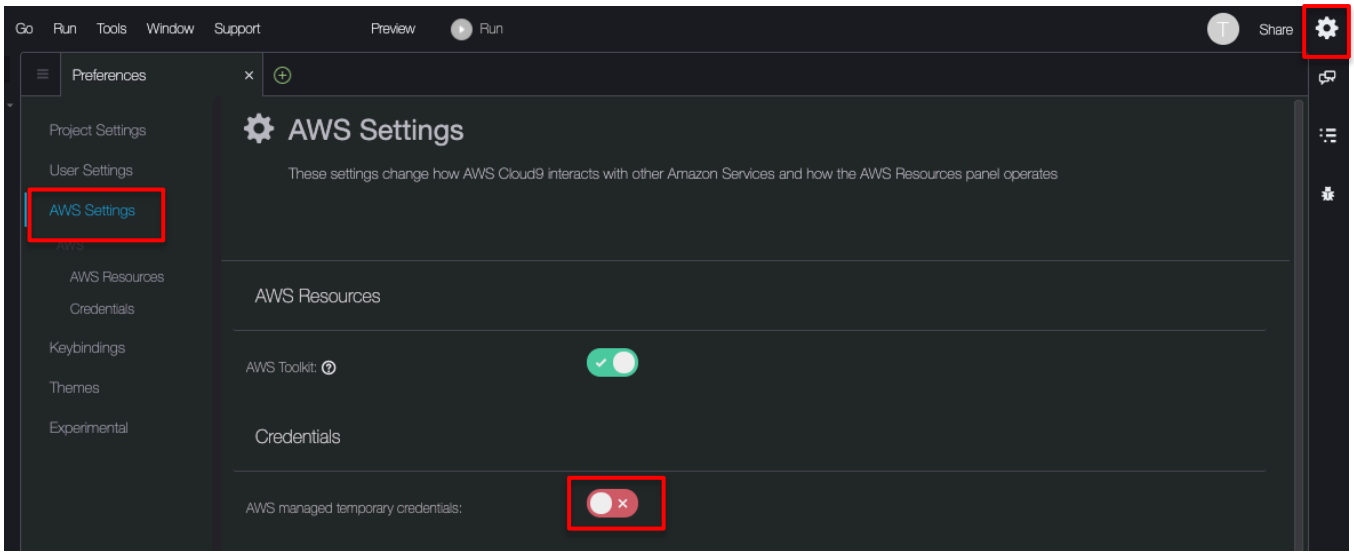
- **AWS Service** 및 **EC2** 가 선택된 것을 확인하고 다음 단계로 이동합니다.



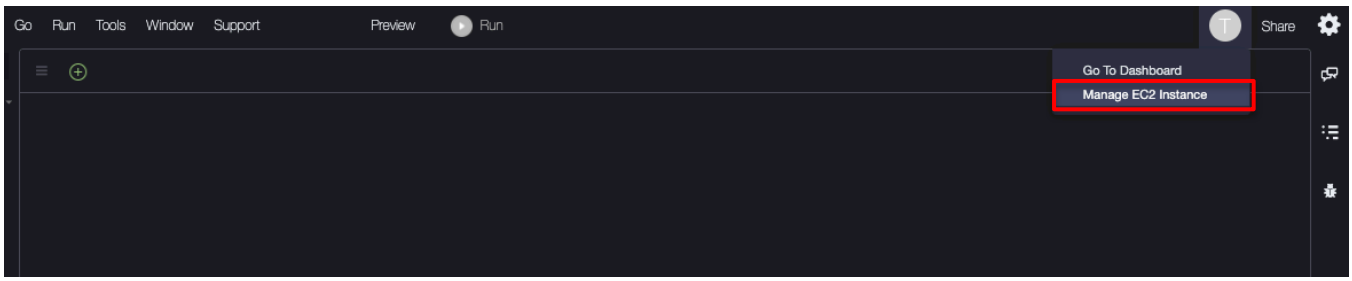
- **AdministratorAccess** 정책이 선택된 것을 확인하고 다음 단계로 이동합니다.
- 태그 추가 단계는 넘어갑니다. 마지막 단계에서 Role name 에 **cdkworkshop-admin** 을 입력한 후, Create role 을 클릭합니다.



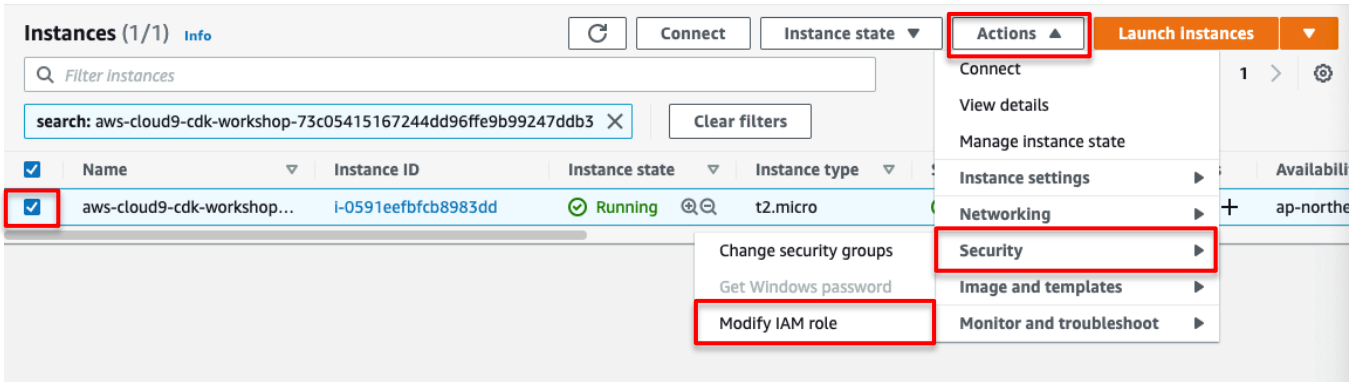
다시 AWS Cloud9 환경으로 돌아와 오른쪽 상단에 있는 **톱니바퀴 모양**을 클릭한 후, 사이드 바에서 **AWS Settings 메뉴**를 선택합니다. Credentials 에서 AWS managed temporary credentials 를 **비활성화**합니다. 설정이 완료되면 창을 닫습니다.



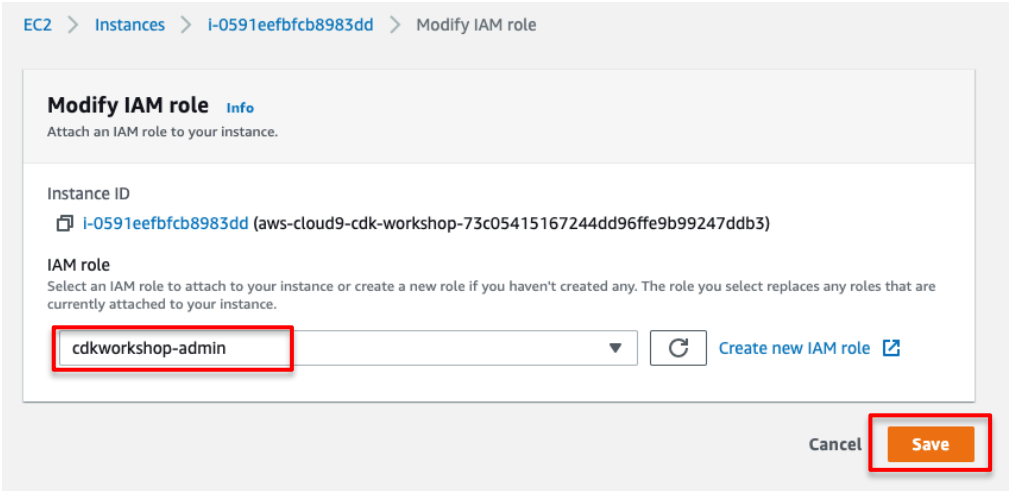
오른쪽 상단 영문 이니셜 아이콘에서 Manage EC2 Instance 를 클릭합니다.



Amazon EC2 화면으로 이동한 후, **AWS Cloud9 인스턴스를 선택**하고, **Actions** 버튼에서 **Security** 메뉴를 선택한 후, **Modify IAM role** 을 클릭합니다.

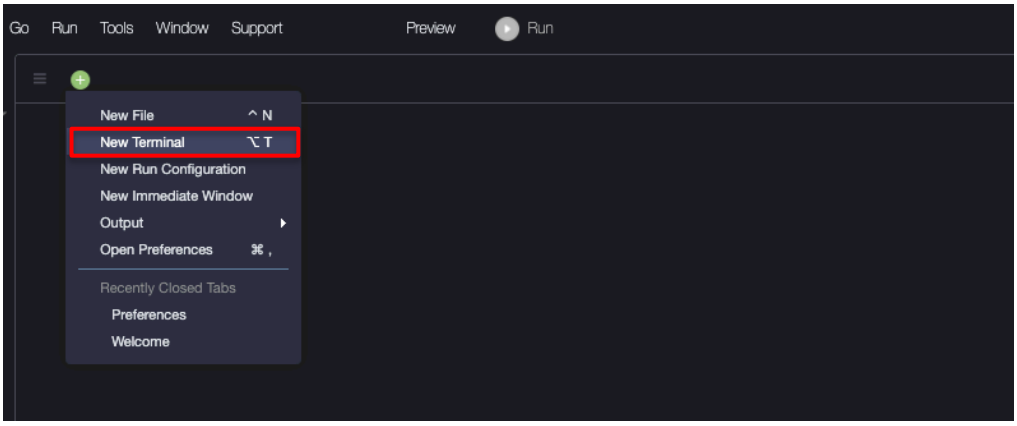


아래의 화면처럼 AWS Cloud9 EC2 인스턴스의 IAM role 을 방금 생성한 cdkworkshop-admin 으로 변경합니다.



2-2-3. 계정 설정 확인하기

다시 Cloud9 실습 환경으로 돌아와서 왼쪽 상단 + 버튼을 클릭한 후, **New Terminal** 을 선택하여 터미널 창을 실행합니다.



터미널 창에 아래의 명령어를 입력합니다. 결과 값이 출력되면 Cloud9 에 IAM role 을 성공적으로 부여한 것입니다.

명령어: `aws sts get-caller-identity`

```
environment $ aws sts get-caller-identity
{
    "Account": "[redacted]",
    "UserId": "AROA257JKJDEAFFHQB3WC:i-03da8a0554c8daa29",
    "Arn": "arn:aws:sts::[redacted]:assumed-role/cdkworkshop-admin/i-03da8a0554c8daa29"
}
```

추가적으로 AWS Region 을 설정합니다.

설정 명령어: `export AWS_REGION=ap-northeast-2 && aws configure set default.region ${AWS_REGION}`

확인 명령어: `aws configure get default.region`

3. AWS CDK 환경 설정

AWS CDK 기반으로 개발하기 위해서 아래와 같은 패키지들이 사전에 설저되어 있어야 합니다.

3-1. jq

JSON 형식의 데이터를 다루는 커맨드라인 유틸리티인 jq 를 설치합니다.

명령어: `sudo yum install -y jq`

3-2. node

다음 링크 가이드에 따라 설치하세요. https://cdkworkshop.com/15-prerequisites/300-nodejs.html

Cloud9 인 경우 이미 설치되어 있습니다.

3-3. cdk cli

다음 링크 가이드에 따라 설치하세요. https://cdkworkshop.com/15-prerequisites/500-toolkit.html

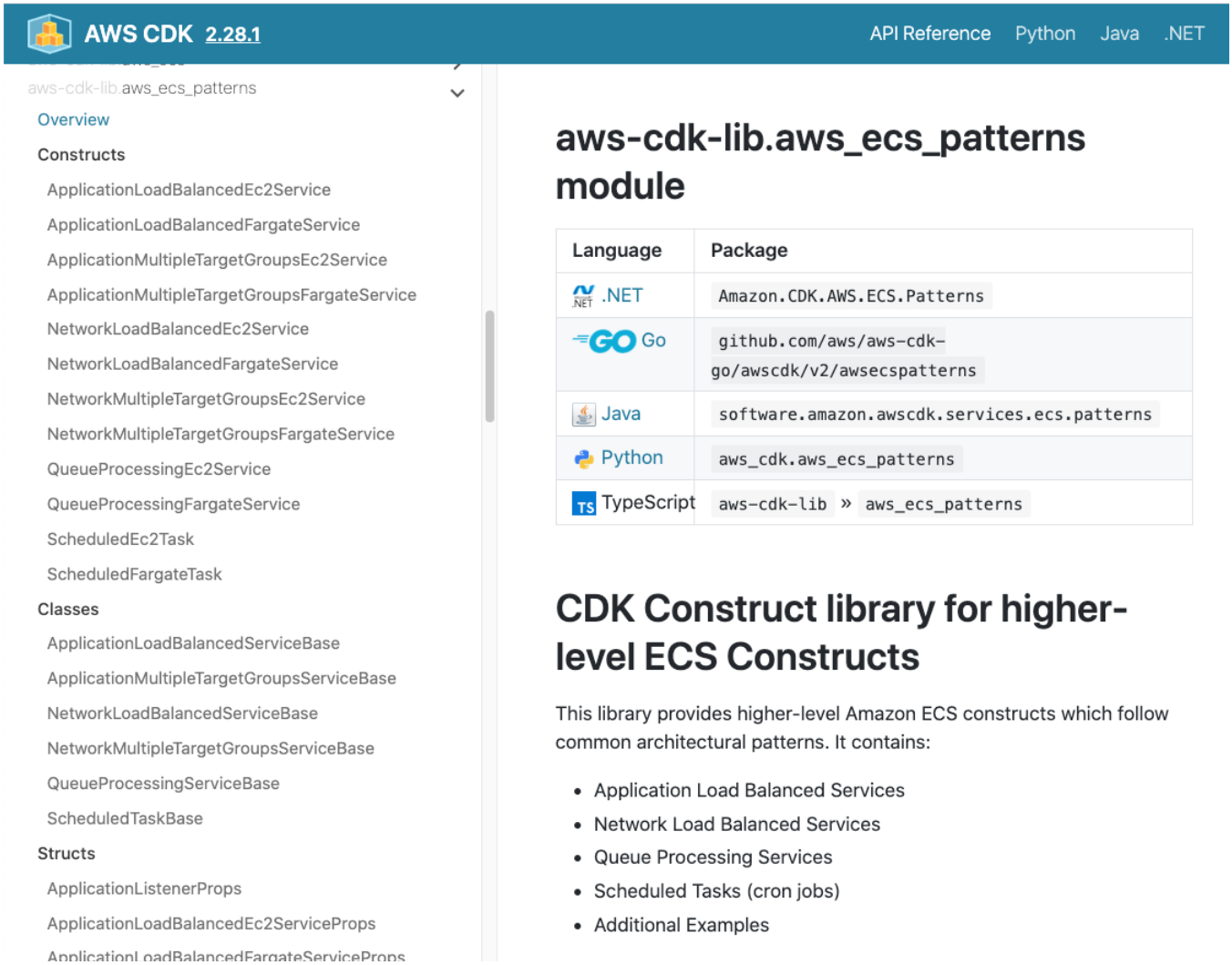
본 실습에서는 "npx"를 이용하여 직접 aws-cdk 를 사용할 예정이기 땜문에 굳이 설치할 필요는 없습니다.

Cloud9 인 경우 이미 설치되어 있습니다.

4. AWS CDK 처음 시작하기

aws ecs patterns 패키지에 있는 모듈을 활용하여 아주 간단하게 VPC + ALB + ECS Cluster + ECS Service + ECS Task 를 생성합니다.

aws ecs patterns:
https://docs.aws.amazon.com/cdk/api/v2/docs/aws-cdk-lib.aws_ecs_patterns-readme.html



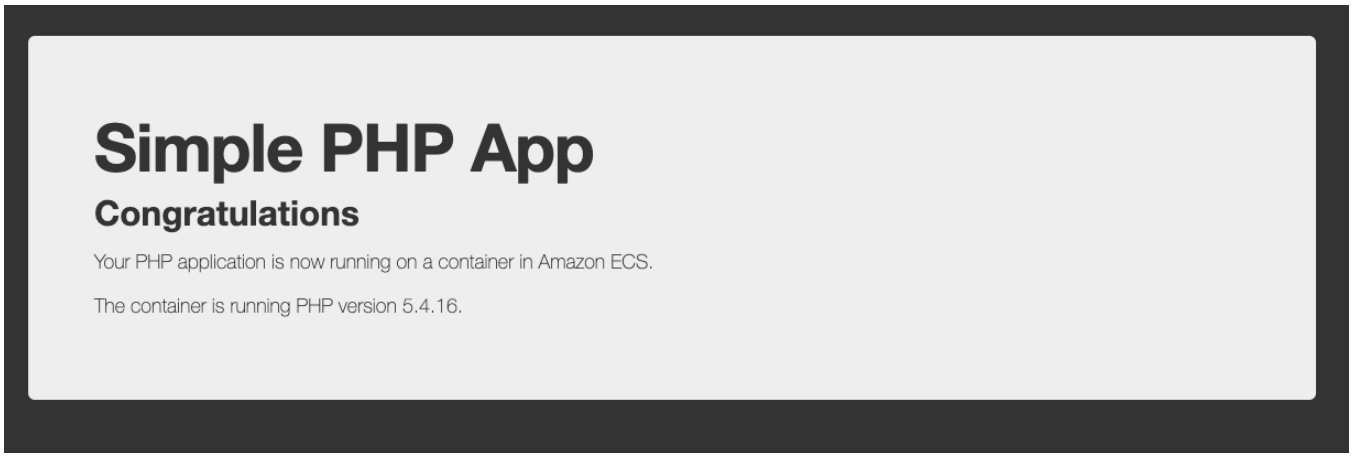
실습 준비 절차:

```
mkdir ecs-cdk && cd ecs-cdk
npx aws-cdk@[1.160.0 or 2.28.1] init app --language=typescript
```

실습 진행 절차:

이후는 실습 진행자의 안내를 받습니다.

a. 디렉토리 정리
b. 배포 타겟 설정(account + region)
c. import 추가
d. 코드 구현
e. cdk list
f. cdk bootstrap aws://[your-account]/[your-region]
g. cdk deploy Xxxx –profile [your-profile]
h. AWS CloudFormation 확인
i. ALB DNS 주소 확인
j. 브라우저 접속하여 확인



예시 소스 코드 – Version1

```
import * as cdk from '@aws-cdk/core';
import * as ecs from '@aws-cdk/aws-ecs';
import * as ecsPatterns from '@aws-cdk/aws-ecs-patterns';

export class EcsCdkStack extends cdk.Stack {
    constructor(scope: cdk.Construct, id: string, props?: cdk.StackProps) {
        super(scope, id, props);

        const loadBalancedFargateService = new
ecsPatterns.ApplicationLoadBalancedFargateService(this, 'Service', {
            memoryLimitMiB: 1024,
            cpu: 512,
            taskImageOptions: {
                image: ecs.ContainerImage.fromRegistry("amazon/amazon-ecs-sample"),
            },
        });
    }
}
```

예시 소스 코드 – Version2

```
import * as cdk from 'aws-cdk-lib';
import { Construct } from 'constructs';
import * as ecs from 'aws-cdk-lib/aws-ecs';
import * as ecsPatterns from 'aws-cdk-lib/aws-ecs-patterns';

export class EcsCdkStack extends cdk.Stack {
  constructor(scope: Construct, id: string, props?: cdk.StackProps) {
    super(scope, id, props);

    const loadBalancedFargateService = new
ecsPatterns.ApplicationLoadBalancedFargateService(this, 'Service', {
        memoryLimitMiB: 1024,
        cpu: 512,
        taskImageOptions: {
          image: ecs.ContainerImage.fromRegistry("amazon/amazon-ecs-sample"),
        },
      });
  }
}
```

5. AWS CDK 로 ECS DevOps 실전 적용하기

본 실습은 소수의 DevOps Engineer 가 동시에 다수의 서비스 개발에 참여하여 MSA/IaC 기반으로 전체 Infrastructure 를 구성 및 배포하기 위한 Best Practice 를 AWS CDK 기반으로 실습합니다. 본 실습을 통해서 완성된 CDK Project 를 활용하면 Container 기반의 DevOps(Culture+Practice+Tool) 역량을 빠르게 끌어 올릴 수 있습니다.

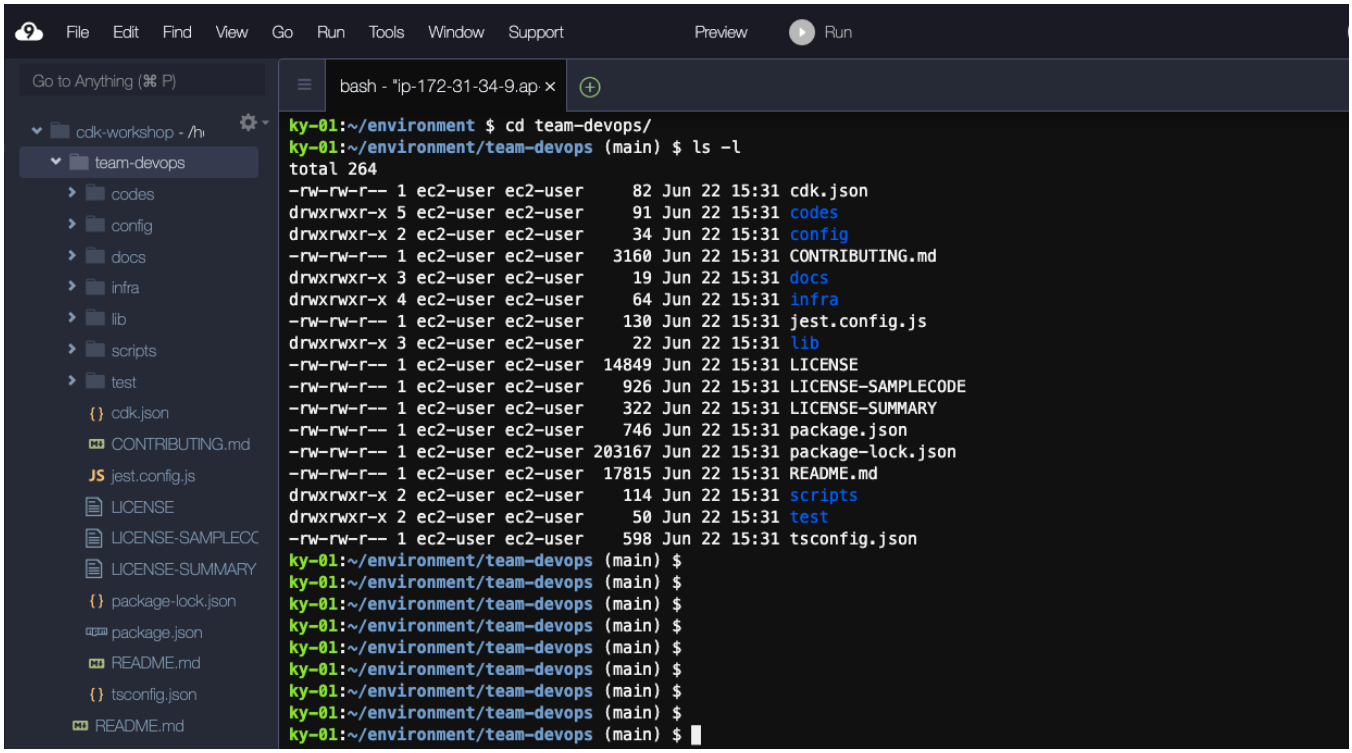
5-1. 소스 코드 준비(DevOps Team)

여기[https://github.com/aws-samples/aws-ecs-devops-using-aws-cdk]를 클릭하여 본 실습을 위한 기본 소스 코드를 확인하고, 다음과 같이 실습용 코드를 다운로드합니다.

실습 코드 다운 받기:

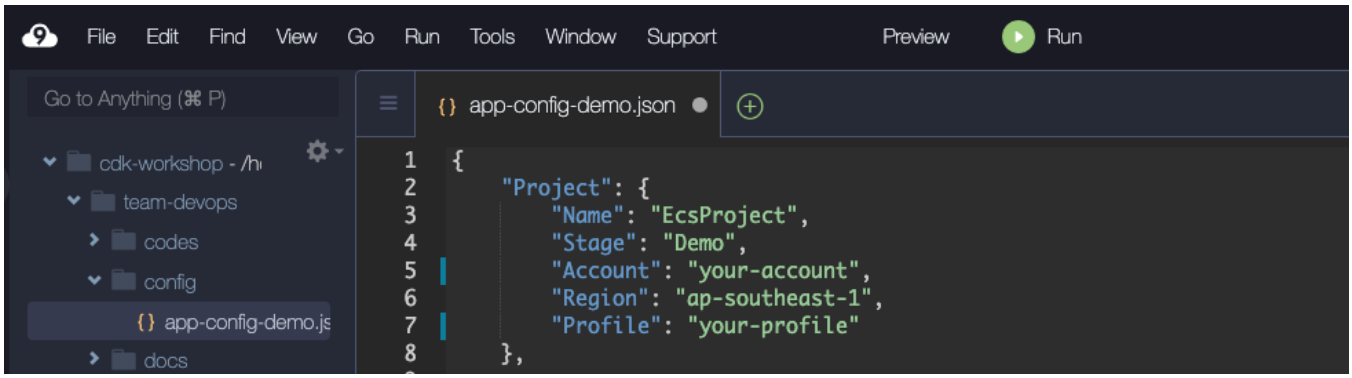
```
git clone https://github.com/aws-samples/aws-ecs-devops-using-aws-cdk.git team-devops
```

"team-devops" 폴더가 새로 생성된 것을 확인할 수 있습니다. 다음 작업을 위하여 clone 한 폴더를 IDE 를 통하여 Open 합니다.



5-2. 배포 Target 설정(DevOps Team)

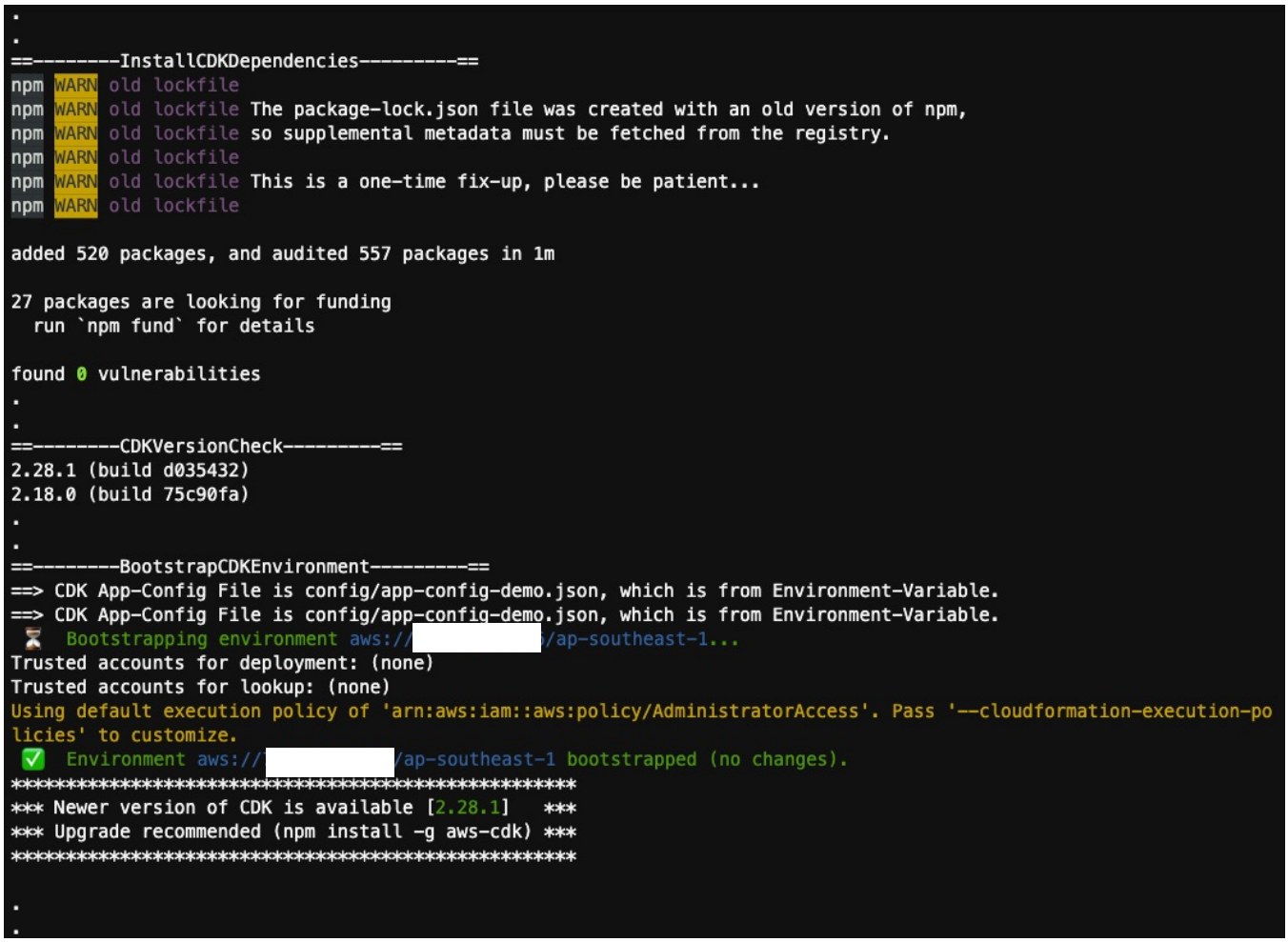
config/app-config-demo.json 파일을 열어 배포 Target 을 지정합니다. 여기서 Target 은 account(aws sts get-caller-identity), region(ap-northeast-2)그리고 profile(default 프로파일을 사용 중인 경우나 Cloud9 을 사용 중인 경우 ""로 입력)을 뜻합니다.



끝으로 다음 명령어를 [team-devops] 폴더에서 실행하여 CDK 프로젝트 초기 셋업을 시작합니다.

```
export APP_CONFIG=config/app-config-demo.json
sh scripts/setup_initial.sh config/app-config-demo.json
```

정상적으로 실행되면 다음과 유사하 화면이 보일 것입니다.

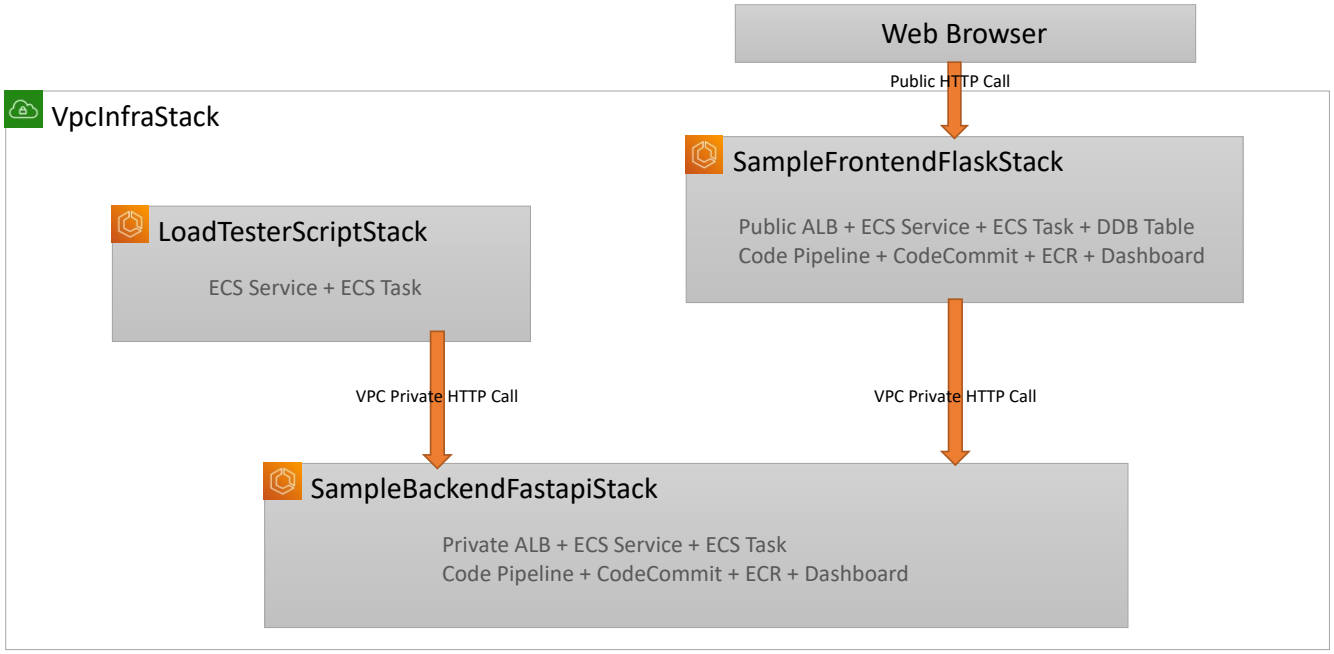


여기서 app-config-demo.json 파일이 매우 중요합니다. 여러분이 다양한 배포 타켓에 대응하기 위해서 여러 벌의 app-config-demo.json 을 갖게 될 것이고, 환경 변수를 이용하여 어떤 배포 타겟을 사용할지를 설정하시면 됩니다. 이렇게 함으로써 우리는 코드 수정없이 다양한 배포 타겟을 대응할 수 있게 됩니다. 예를 들어 아래와 같은 json 파일들을 관리하게 됩니다.

- app-config-demo.json
- app-config-test.json
- app-config-dev.json
- app-config-prod.json

5-3. Infrastructure 배포하기(DevOps Team)

DevOps 팀은 공통 인프라와 각 서비스팀의 인프라 배포 역할을 맡고 있습니다. 현재 이를 위해 다음과 같은 스택으로 구성되어 있습니다.



DevOps 팀은 기본 공통으로 사용되는 VPC 와 ECS Cluster 를 사전에 구성해 놓습니다. 이후 DevOps 팀은 서비스(backend/frontend) 개발팀과 미팅을 통해서기본적인 요구사항을 접수 후, `app-config-demo.json` 파일에 각 서비스 개발팀이 요구하는 스펙에 맞춰 초기 인프라를 배포합니다.

공통 리소스에 대한 설정은 다음과 같습니다.

```
"VpcInfra": {
    "Name": "VpcInfraStack",

    "VPCName": "CommonVPC",
    "VPCMaxAzs": 3,
    "VPCCIDR": "10.0.0.0/16",
    "NATGatewayCount": 1,

    "ECSClusterName": "MainCluster"
},
```

backend 팀의 요구 사항에 대한 설정은 다음과 같습니다.

```
"SampleBackendFastapi": {
    "Name": "SampleBackendFastapiStack",
    "InfraVersion": "'1.0.0'",
    "DockerImageType": "HUB",
    "DockerImageType-Desc": "HUB or ECR or LOCAL",

    "PortNumber": 80,
    "InternetFacing": false,

    "AppPath": "codes/sample-backend-fastapi",
    "DesiredTasks": 1,
    "Cpu": 256,
    "Memory": 512,

    "AutoScalingEnable": false,
    "AutoScalingMinCapacity": 1,
    "AutoScalingMaxCapacity": 2,
    "AutoScalingTargetInvocation": 50,

    "TableName": "LogTable",

    "AlarmThreshold": 200,
    "SubscriptionEmails": ["kwonyul@amazon.com"]
},
```

frontend 팀의 요구 사항에 대한 설정은 다음과 같습니다.

```
"SampleFrontendFlask": {
    "Name": "SampleFrontendFlaskStack",
    "InfraVersion": "'1.0.0'",
    "DockerImageType": "HUB",
    "DockerImageType-Desc": "HUB or ECR or LOCAL"

    "PortNumber": 80,
    "InternetFacing": true,

    "TargetStack": "SampleBackendFastapiStack",

    "AppPath": "codes/sample-frontend-flask",
    "DesiredTasks": 1,
    "Cpu": 256,
    "Memory": 512,

    "AutoScalingEnable": false,
    "AutoScalingMinCapacity": 1,
    "AutoScalingMaxCapacity": 2,
    "AutoScalingTargetInvocation": 50,

    "AlarmThreshold": 200,
    "SubscriptionEmails": ["kwonyul@amazon.com"]
},
```

추가적으로 backend 를 로드테스트하기 위한 구성 설정은 다음과 같습니다.

```
"LoadTesterScript": {
    "Name": "LoadTesterScriptStack",

    "TargetStack": "SampleBackendFastapiStack",

    "AppPath": "codes/load-tester-script",
    "DesiredTasks": 1,

    "Environment": {
        "RequestCount": 10,
        "SleepPeriodInSec": 1
    }
}
```

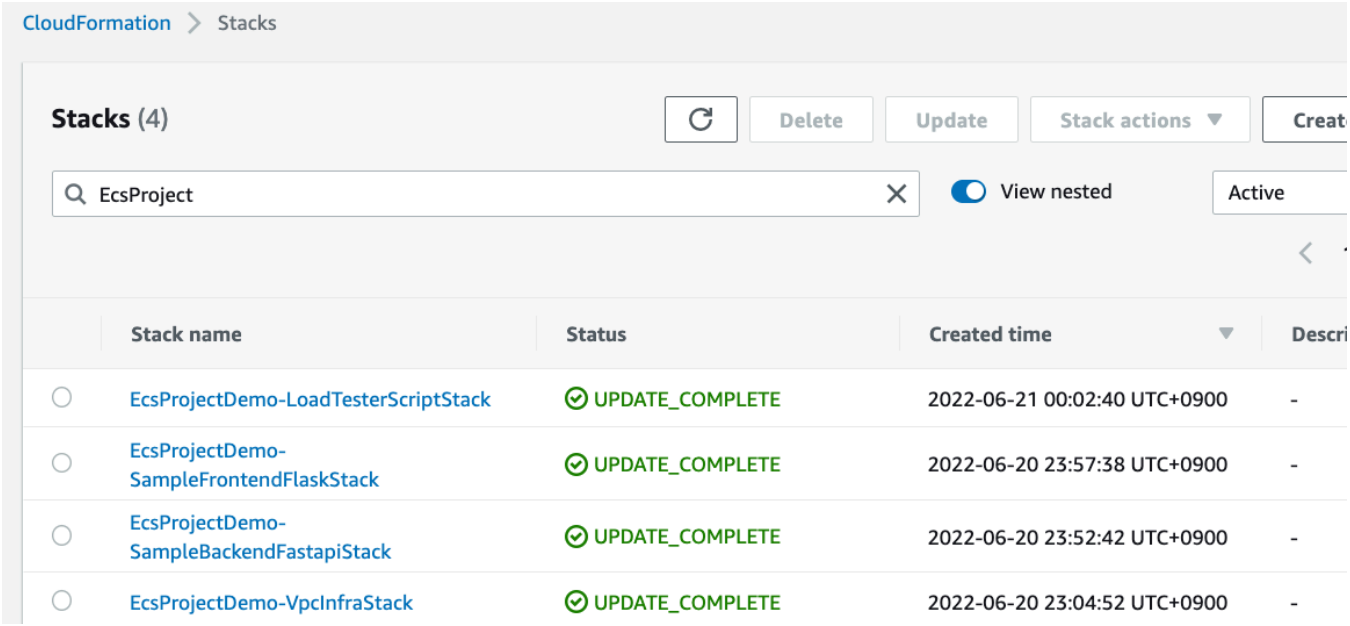
다음 명령어를 통해서 모든 리소스들을 한번에 모두 배포합니다.

```
sh scripts/deploy_stacks.sh config/app-config-demo.json
```

이 스크립트의 핵심 코드는 다음과 같습니다.

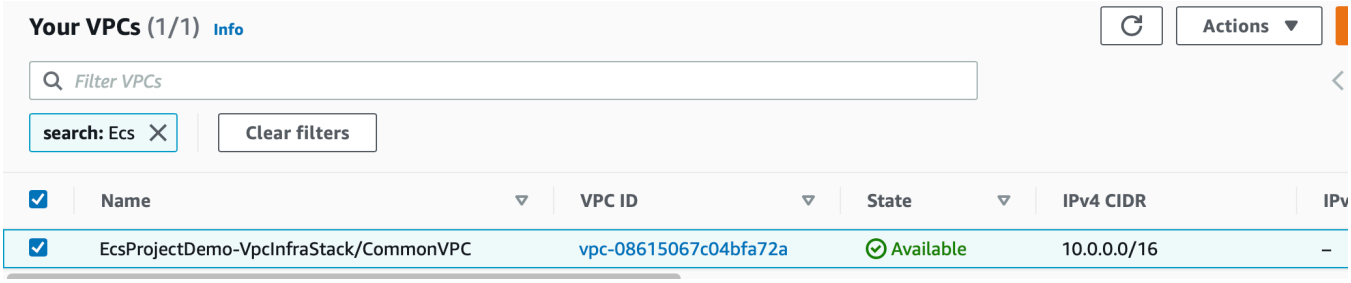
```
echo ==--------CDKVersionCheck---------==
alias cdk-local="./node_modules/.bin/cdk"

echo ==--------DeployStacksStepByStep---------==
cdk-local deploy *-VpcInfraStack --require-approval never
cdk-local deploy *-SampleBackendFastapiStack --require-approval never
cdk-local deploy *-SampleFrontendFlaskStack --require-approval never
cdk-local deploy *-LoadTesterScriptStack --require-approval never
```

CloudFormation 을 통해서 이들의 배포 상태를 확인할 수 있습니다.

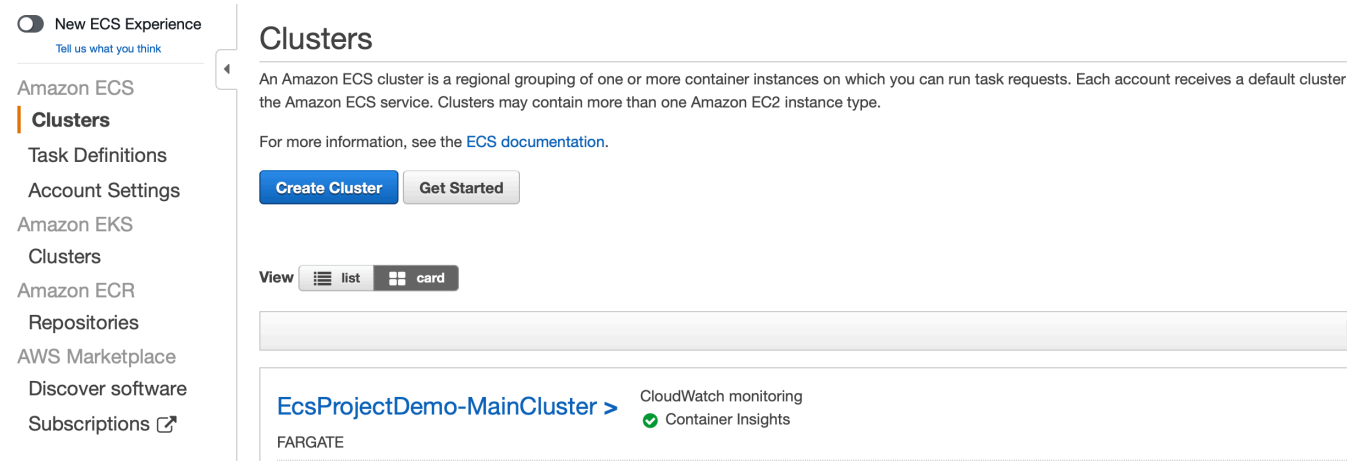


5-3-1. VpcInfraStack 배포 확인

VpcInfraStack 배포가 완료되면 VPC 가 배포된 것을 VPC 서비스 화면에서 확인할 수 있습니다.

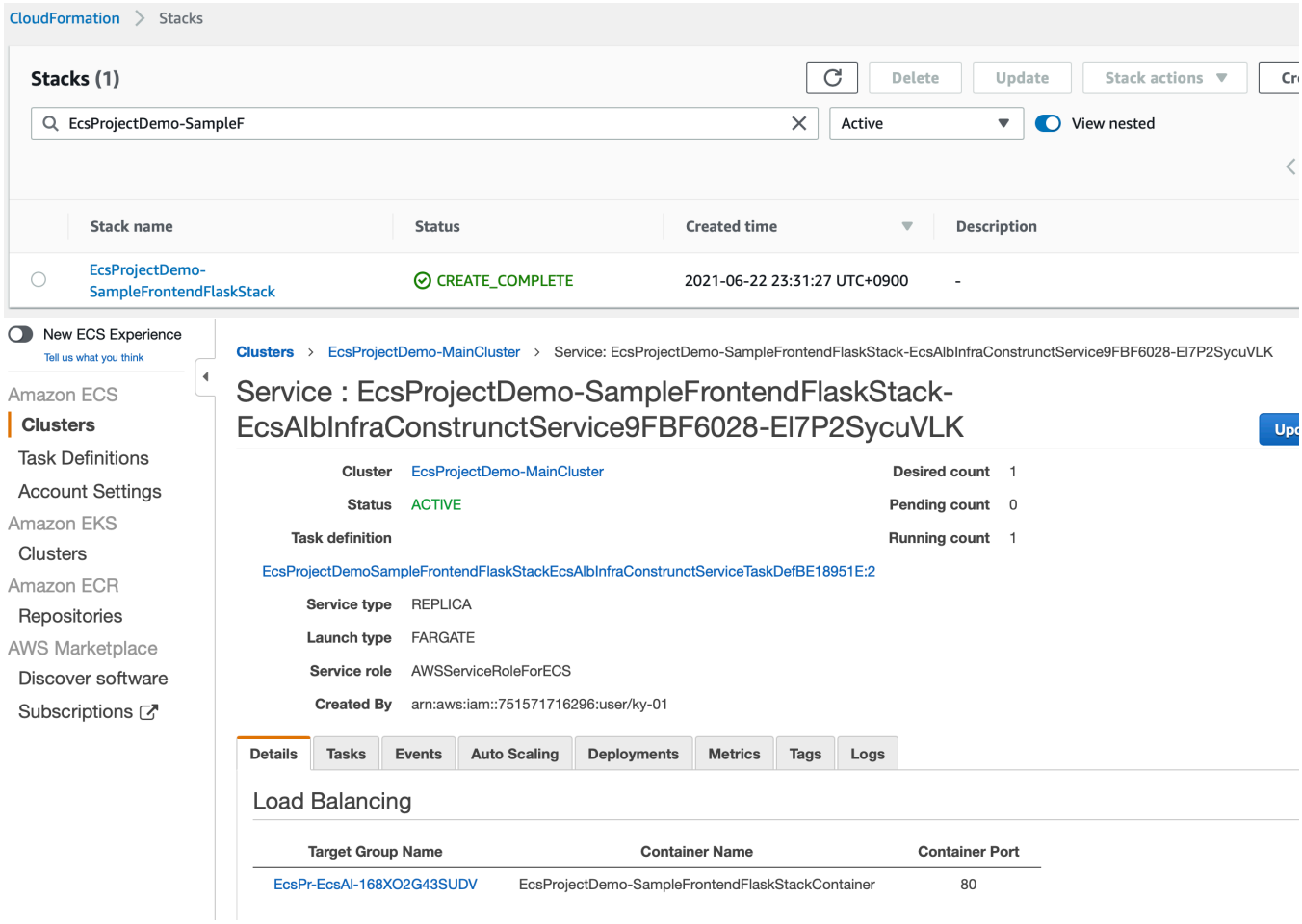


VpcInfraStack 배포가 완료되면 ECS Cluster 가 배포된 것을 ECS 서비스 화면에서 확인할 수 있습니다.



5-3-2. SampleFrontendFlaskStack 배포 확인

배포가 완료되면 ALB 와 ECS Service/Task 가 배포된 것을 CloudFormation 이나 각 서비스 화면에서 확인할 수 있습니다.

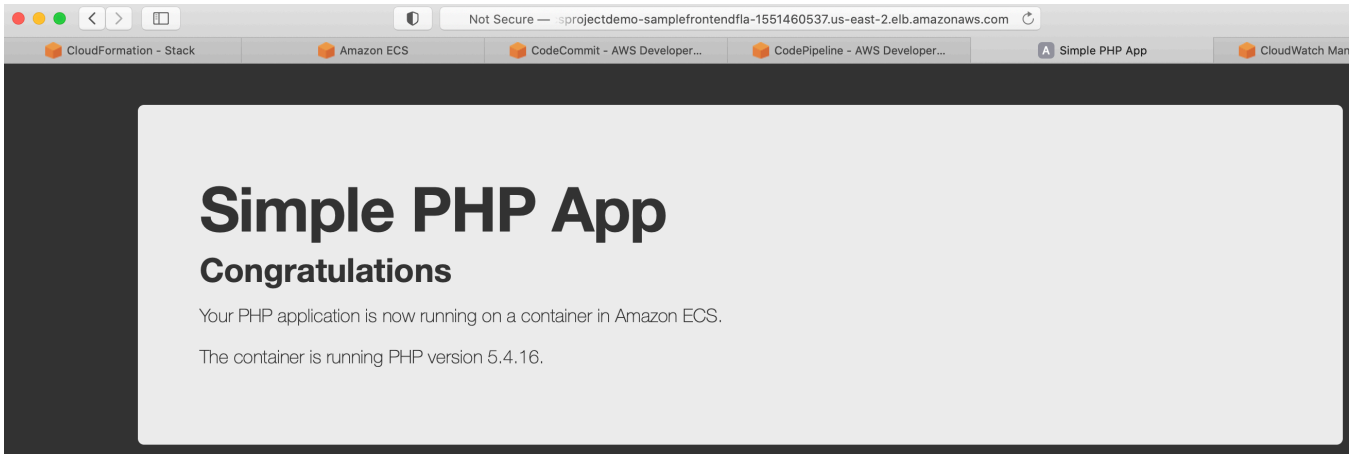


ALB 의 DNS 주소를 웹브라우저를 통하여 정상적으로 open 되는지 확인합니다. ALB 의 주소는 다음 그림과 같이 CDK 배포 과정에 출력됩니다.

```
✅  EcsProjectDemo-SampleFrontendFlaskStack

Outputs:
EcsProjectDemo-SampleFrontendFlaskStack.EcsAlbCicdConstructCodeCommitName916C195C = ecsprojectdemo-samplefrontendflaskstack-repo
EcsProjectDemo-SampleFrontendFlaskStack.EcsAlbInfraConstructServiceLoadBalancerDNSF445CBCD = EcsProjectDemo-SampleFrontendFla-1551460537.us-east-2.elb.amazonaws.com
EcsProjectDemo-SampleFrontendFlaskStack.EcsAlbInfraConstructServiceServiceURL290953F6 = http://EcsProjectDemo-SampleFrontendFla-1551460537.us-east-2.elb.amazonaws.com
```

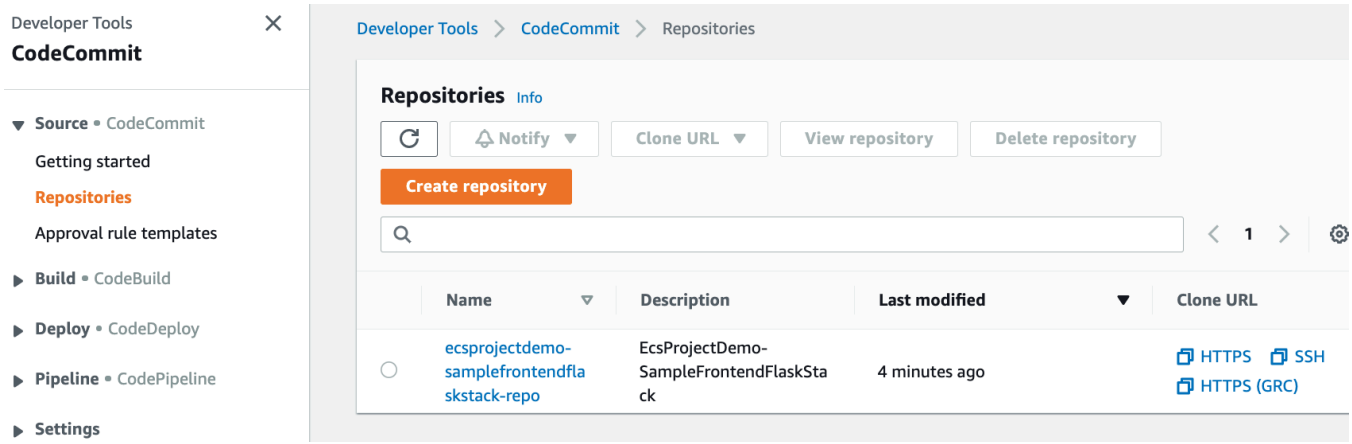
본 초기 웹 페이지는 PHP Sample Page 로서 github 를 통해서 제공되는 초기 화면일 뿐이며 향후 스텝에 CodePipeline 을 통하여 정식 ECR 에 있는 우리만의 Docker Image 가 배포될 예정입니다. 아직

서비스팀에 의해서 정상 Push 되지 않은 상태이기 때문에 최초 배포시에 임시방편으로 지정한 것 뿐입니다.

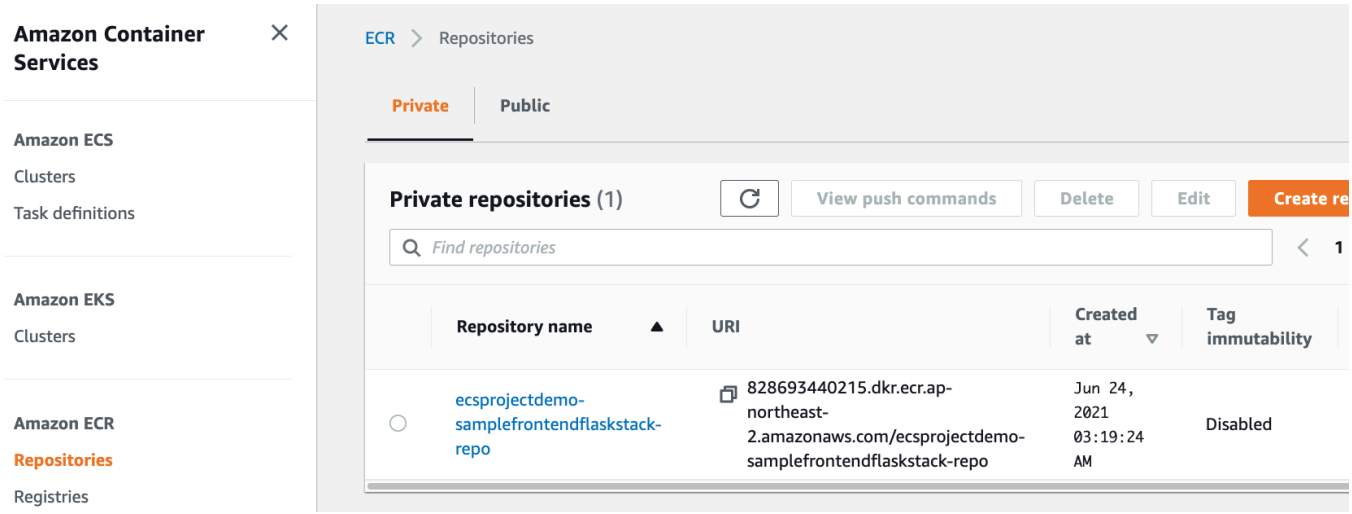


DevOps 팀은 최종적으로 codecommit 주소(ecsprojectdemo-samplefrontendflaskstack-repo) 및 소스 위치 directory(codes/sample-frontend-flask)를 Service 팀에게 전달합니다.

새로 생성된 CodeCommit



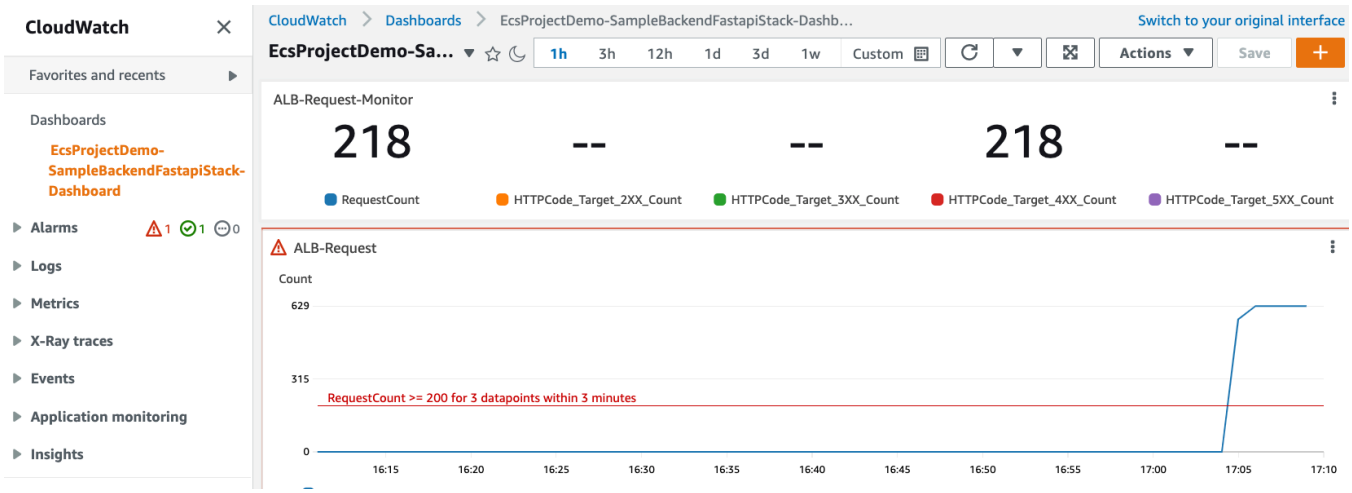
새로 생성된 ECR Repository



5-3-3. SampleBackendFastapiStack 및 LoadTesterScriptStack 배포 확인

Backend API Service 이기 때문에 Private 으로 배포되고 이때문에 Web Browser 를 통하여 육안으로 직접 확인이 불가하며 대신 Test 용 ECS Task 를 동일 VPC 에 배포하여 AB(Apache Bench) Tool 을 이용하여 원격 자동 부하 테스트합니다. 또한 이때 CloudMap 을 이용하여 Private DNS 로 해당 서비스를 ServiceDiscovery 하여 접속합니다.

최종적으로 CloudWatch 를 통하여 실시간 현황을 확인할 수 있습니다. 하지만 아직 서비스팀을 통해서 서비스 로직이 배포되지 않았기 때문에 사전에 정의된 Test 용 Request URL 이 구현되어 있지 않아서 2XX 가 아닌 4XX 의 에러들만 감지되고 있습니다.



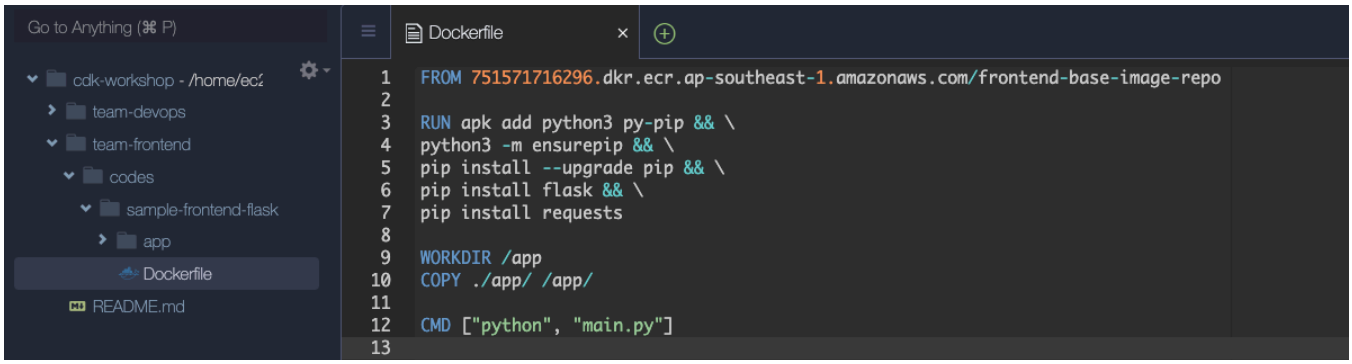
DevOps 팀은 최종적으로 codecommit 주소(ecsprojectdemo-samplebackendfastapistack-repo) 및 소스 위치 directory(codes/sample-backend-fastapi
)를 Service 팀에게 전달합니다.

5-4. Web Frontend Application 배포하기(Service Team)

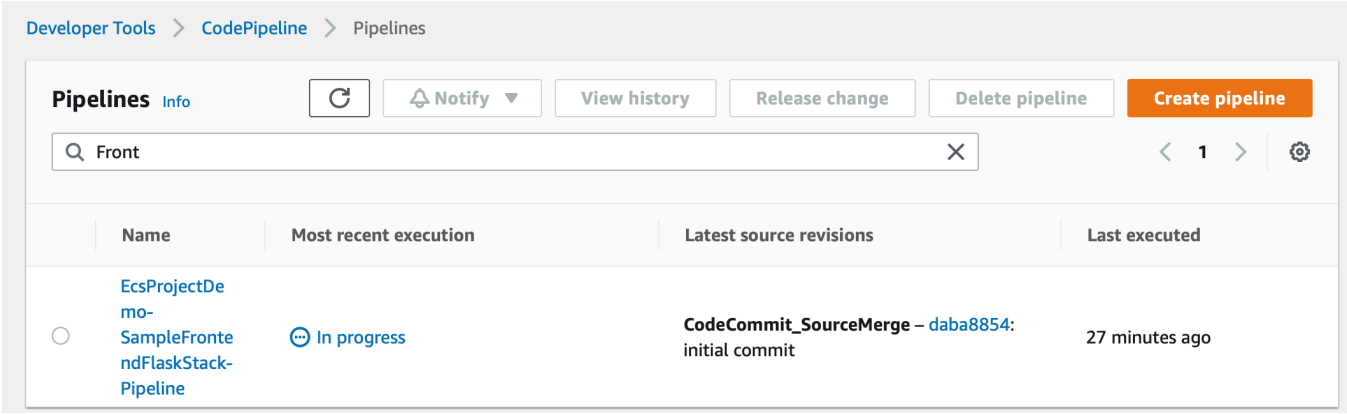
Frontend 팀은 이제 사전에 전달받은 codecommit 주소로 clone 후, 사전에 전달받은 directory(codes/sample-frontend-flask)에 service 로직 코드를 구현하여 준비합니다. 본 실습에서는 편의 상 codes 디렉토리에 미리 준비해두었으며, 실제는 codes 하위의 각 비지니스 로직 구현은 각 서비스 개발팀이 자신의 Repository 에서 구현해야 합니다.

```
git clone https://git-codecommit.ap-southeast-
1.amazonaws.com/v1/repos/ecsprojectdemo-samplefrontendflaskstack-repo team-
frontend

cd team-frontend
mkdir codes
cp -r ../team-devops/codes/sample-frontend-flask ./codes/

git add .
git commit -m "initial commit"
git push origin master
```

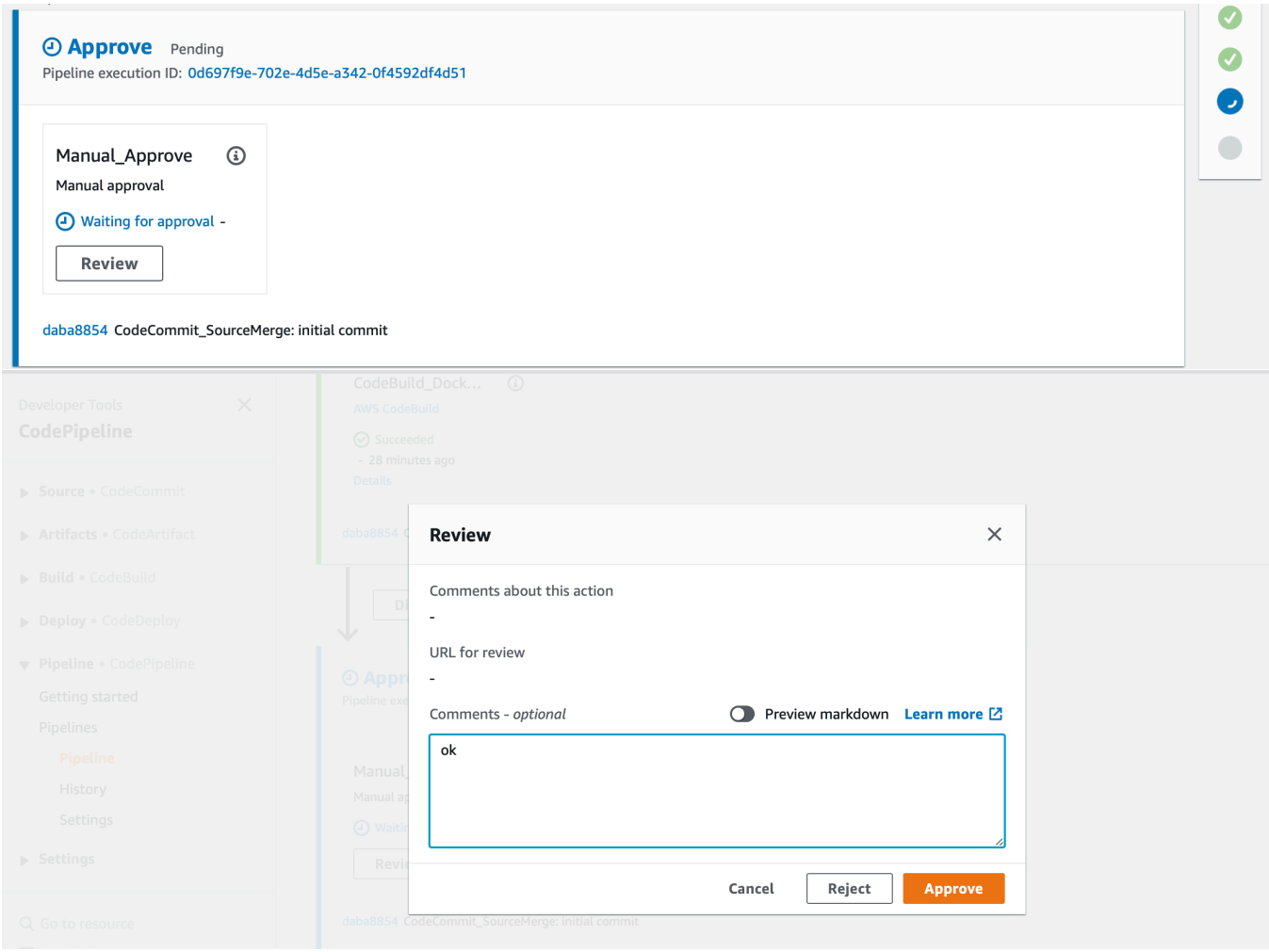
directory 구조는 아래와 같습니다.



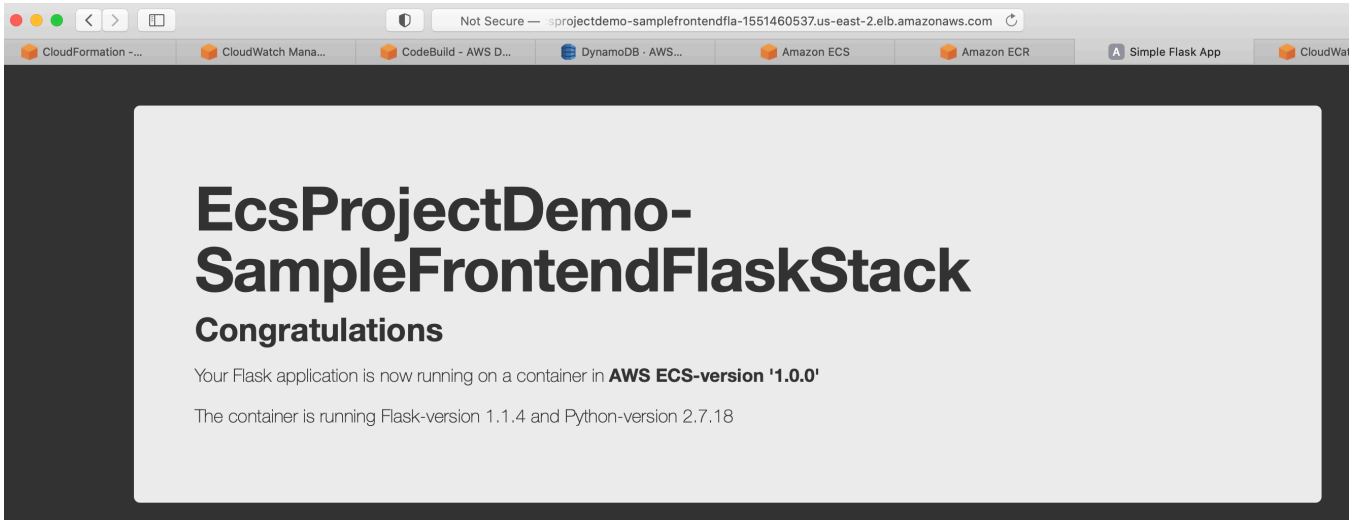
CodeCommit 의 master branch 로 push 이벤트 발생 시에 배포 Pipeline 이 시작되도록 구현되어 있습니다. 이제 CodePipeline 으로 이동하여 소스가 자동 배포되는지 확인합니다.



다음과 같이 Approve 상태에서 Review 버튼을 클릭해야지 다음 스텝으로 진행되어 최종 배포됩니다.



최종 배포 완료 후에는 다음과 같이 Flask 로 구현된 웹 페이지를 확인할 수 있습니다.

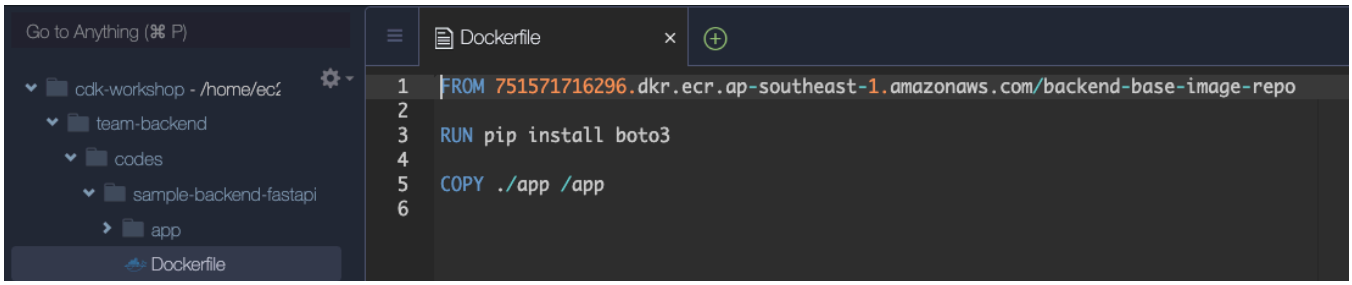


5-5. API Backend Application 배포하기(Service Team)

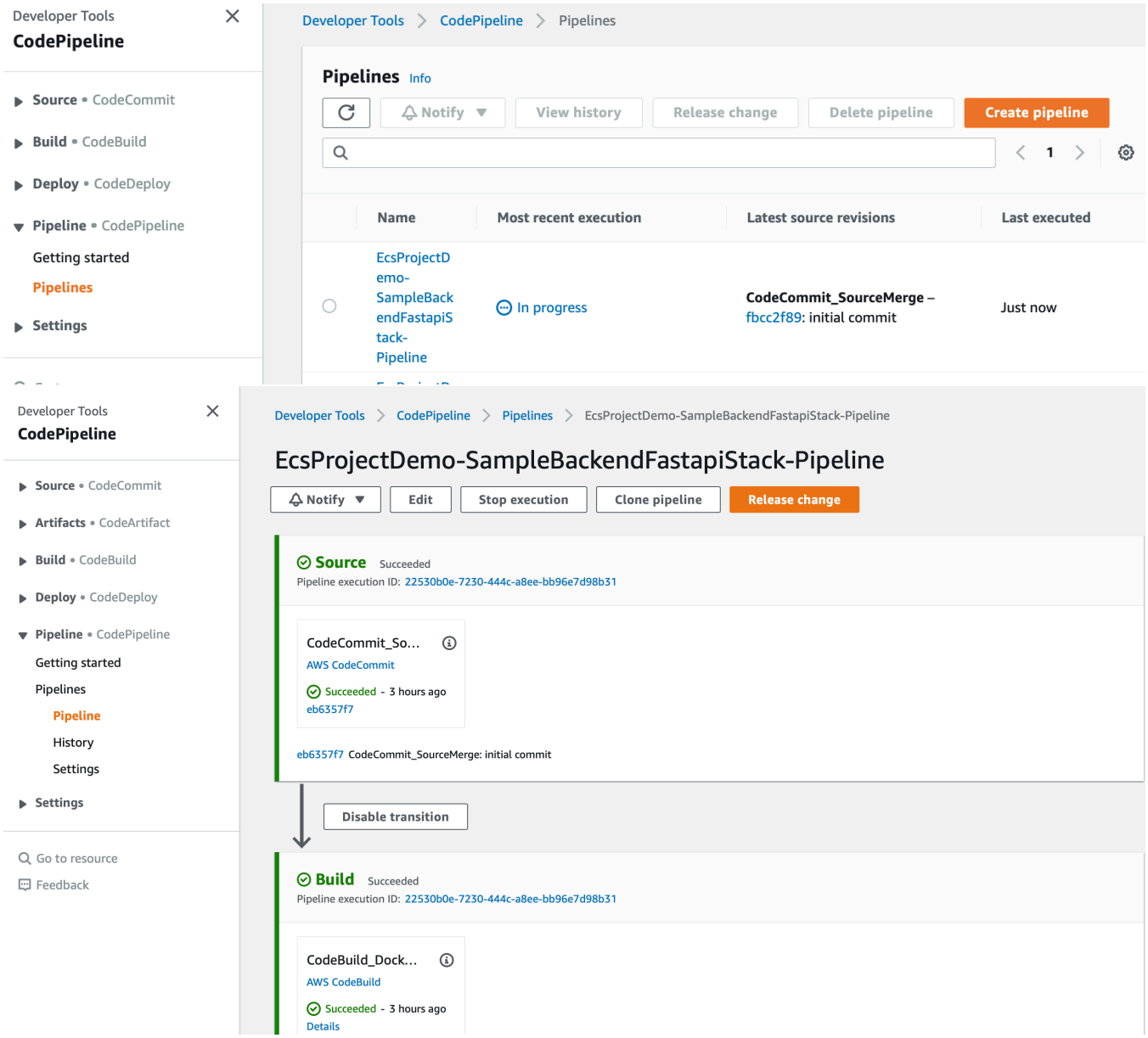
Backend 팀은 동일하게 CodeCommit ecsprojectdemo-samplebackendfastapistack-repo 를 clone 하고, sample-backend-fastapi 를 복사 후, 최종 push 합니다. 최종적으로 CodePipeline 이 트리거되어 배포가 자동화되고 Accept 과정을 거쳐 최종 배포된 것을 확인할 수 있습니다.

```
git clone https://git-codecommit.ap-southeast-1.amazonaws.com/v1/repos/ecsprojectdemo-samplebackendfastapistack-repo team-backend

cd team-backend
mkdir codes
cp -r ../team-devops/codes/sample-backend-fastapi ./codes/

git add .
git commit -m "initial commit"
git push origin master
```

최종 directory 모습



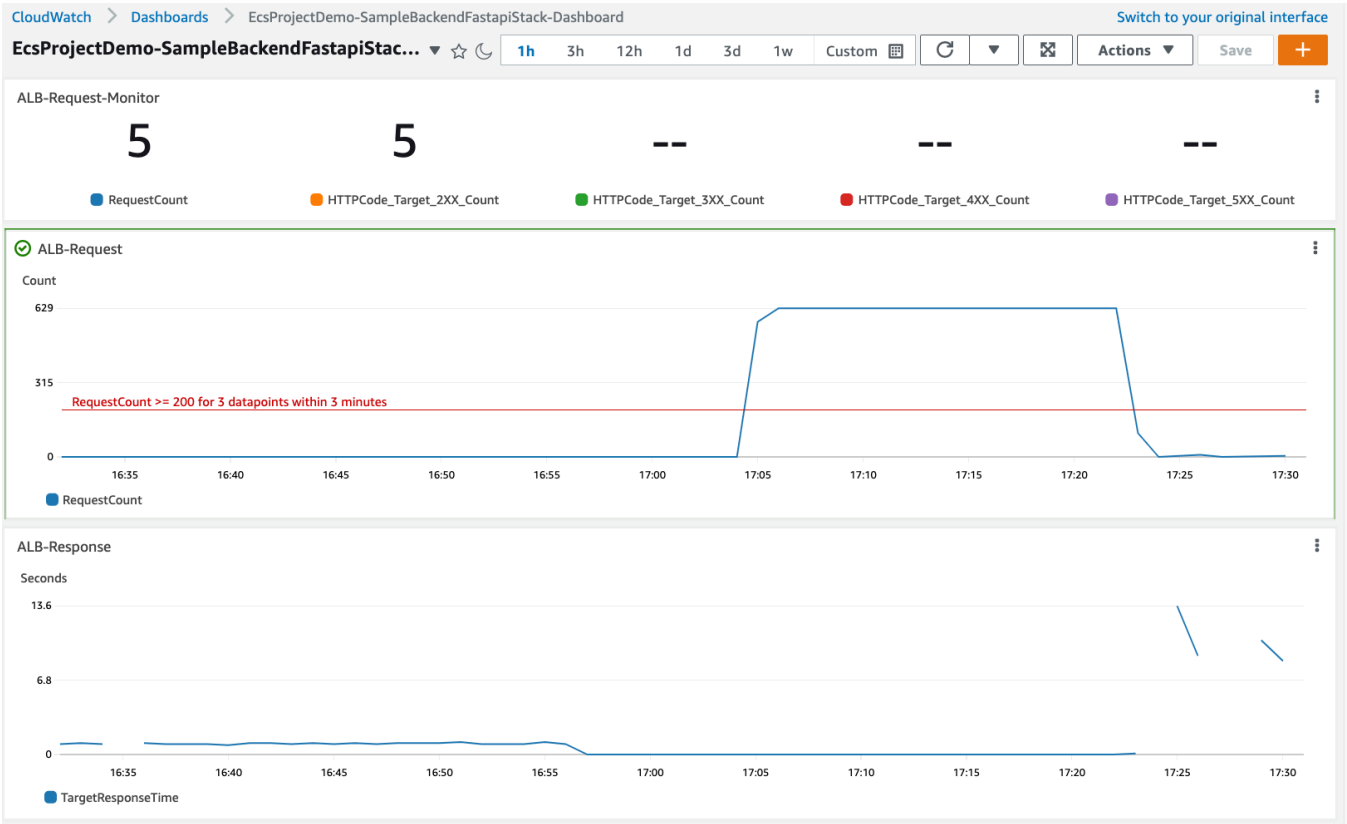
Pipelie 배포 모습



최종 배포되어 CloudWatch Dashboard 를 통하여 실시간 모니터링되는 화면은 다음과 같습니다. 전체 Request 갯수와 2XX Request 가 동일하여 이슈가 없음을 확인할 수 있습니다.

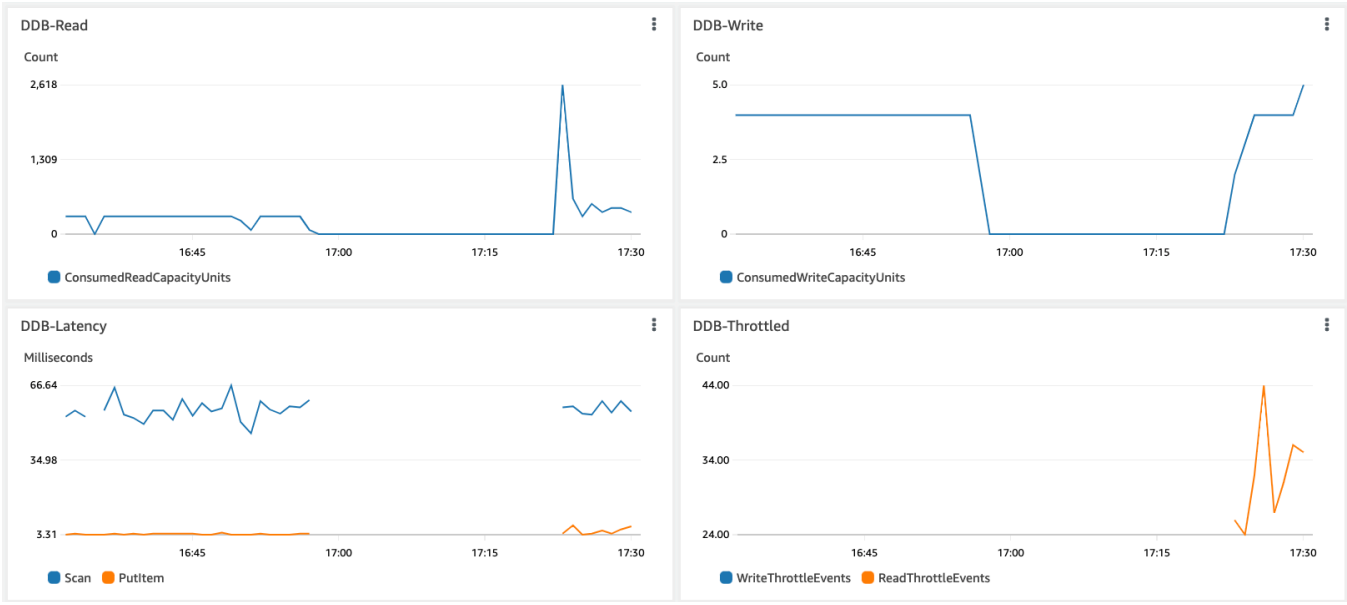
ALB Metric

특이한 현상이 하나 발견됩니다. RequestCount 의 숫자가 현저하게 작아졌고, ResponseTime 도 매우 길어졌습니다. 이유는 DDB Table 의 메트릭에서 찾을 수 있습니다.



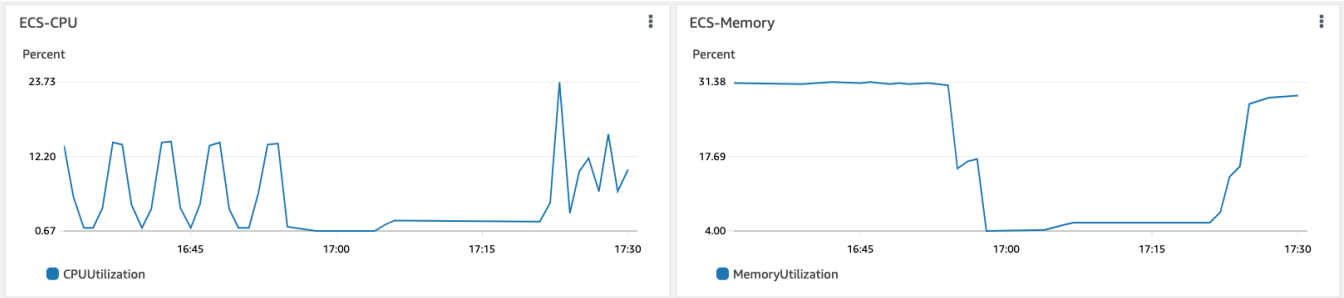
DynamoDB Metric

내부적으로 DDB 의 Read Capacity 가 낮아서 Throttle 이 발생하고 있음을 확인할 수 있습니다.



ECS Service Metric

CPU 와 Memory 의 Utilization 이 10% 이하로 현재 여유 가동되고 있음을 확인할 수 있습니다.



5-6. Infrastructure 업데이트하기(DevOps Team)

DevOps 팀은 운영 중에 인플라에 대한 변경 사항이 분명히 발생할 수 있습니다. 기존 처럼 cdk deploy 를 통해서 원하는 스택을 업데이트 배포하면 되지만 조심할 사항이 있습니다. 초기에 임시방편으로 DockerHub 에서 가져오도록 되어 있었기 때문에 Container 의 주소가 이제는 ECR 로 변경을 해줘야 합니다.

Backend/Frontend 동일하게 "DockerImageType"을 아래와 같이 변경해주세요.

- 변경 전

```
"DockerImageType": "HUB",
```

- 변경 후

```
"DockerImageType": "ECR",
```

6. 트러블슈팅

6-1. CICD 빌드 실패

CICD 의 빌드 과정 중에 docker pull 이 되지 않아서 배포가 실패되는 경우를 목격할 수 있습니다. 이는 DockerHub 의 pull limitation 에 의해서 발생합니다. 이런 경우는 ECR 에 base-image 를 미리 준비해 놓고 이를 참조할 수 있도록 해야 합니다.

ECR Repo 만들고 base-image push 하기 위해서 다음 명령어를 실행하세요.

- backend

```
sh scripts/create_base_image_in_ecr.sh config/app-config-demo.json backend-base-image-repo tiangolo/uvicorn-gunicorn-fastapi:python3.7
```

- frontend

```
sh scripts/create_base_image_in_ecr.sh config/app-config-demo.json frontend-base-image-repo alpine:3.10
```

각 서비스팀의 Dockerfile 의 "FROM"을 ECR Repo 로 변경하세요.

- backend

```
FROM [your-account].dkr.ecr.ap-southeast-1.amazonaws.com/backend-base-image-repo
```

-frontend

```
FROM [your-account].dkr.ecr.ap-southeast-1.amazonaws.com/frontend-base-image-repo
```

6-2. DDB Read Capacity

DDB 의 Read Capacity 가 기본값으로 설정되어 있어서 backend 가 제대로 응답을 받지 못 하고 있습니다. DDB Table 의 Throttle 이 제거될 수 있도록 DDB Table 의 Capacity 를 올려야 합니다.

7. 도전 과제

- CICD Pipeline 에 대한 트리거 브랜치가 "master"로 하드코딩 되어 있습니다. 이것을 파라미터로 추출하여 외부에서 변경 가능하도록 해보세요.
- 기존에 직접 만들어 놓은 ECS 서비스에 CICD 만 붙여 보세요.(EcsCicdStack 활용)
- ECS Service 에 AutoScaling 을 구현하고, Scaling 될 수 있도록 Load Test 강도를 높여 보세요.(ECS Service 의 autoScaleTaskCount() 메소드 활용)

8. 정리하기

최종적으로 다음과 같은 사전에 준비된 script 파일을 실행하여 리소스를 한번에 정리합니다. 참고로 DynamoDB, CodeCommit 그리고 ECR 과 같은 저장소들은 CDK 를 통해서 자동 삭제되지 않을 수 있으므로 수동 삭제하셔야 합니다.

```
sh scripts/destroy_stacks.sh config/app-config-demo.json
```

물론 다시 새롭게 모든 스택을 한번에 배포하려면 다음과 같이 script 파일을 실행시킬 수 있습니다.

```
sh scripts/deploy_stacks.sh config/app-config-demo.json
```