Terraform Mini Project3

AWS클라우드 환경에서

CICD 파이프라인 코드화 구현

김 유 진

1. ECR & Git Repository 생성

*** ECR (EC2 Container Registry)**

개발자가 Docker 컨테이너 이미지를 손쉽게 저장, 관리 및 배포할 수 있게 해주는 완전관리형 Docker 컨테이너 레지스토리

도커 이미지를 저장할 ECR을 생성한다. 깃허브 레포지토리에 도커파일 스크립트를 올리면, ECR과 연동하여 자동으로 이미지를 빌드하고, 결과를 ECR에 저장할 수 있다. Gradle로 빌드하여 패키징한 파일로 도커 이미지를 만들어 ECR로 push 한다.

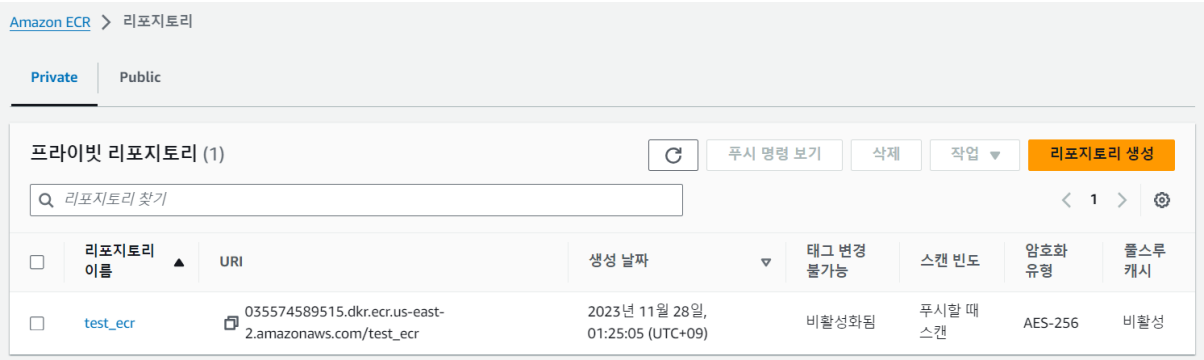
[ecr.tf]

```
resource "aws_ecr_repository" "test_ecr" {
  name = "test_ecr"
  image_tag_mutability = "MUTABLE"

  image_scanning_configuration {
    scan_on_push = true
  }
}
```

[outputs.tf]

```
output "test_ecr_url" {
  description = "Test ECR Url"
  value = aws_ecr_repository.test_ecr.repository_url
}
```

ECR 생성 확인



[STEP2]

소스 저장용 레포지토리와 GitOps용 레포지토리를 준비한다.

*** 소스 저장용 Repo가 저장하는 것**
- 프로젝트 개발 소스 코드 (backend)
- Dockerfile
- buildspec.yaml

*** GitOps용 Repo가 저장하는 것**
- deployment.yaml
- service.yaml

2. S3 & DynamoDB 생성

.tfstate 파일에는 테라폼의 버전과 각 리소스에 대한 정보가 들어있다. 한 폴더에서 관리한다면 참조가 가능하지만 폴더를 분리할 경우 참조가 불가능하다. 이 때 원격으로 데이터 소스를 관리하기 위해 terraform_remote_state를 사용한다.

[s3.tf]

```
####################################
#    Backend S3 Bucket
####################################
resource "aws_s3_bucket" "jin_bucket" {
  bucket = "jin-bucket"

  tags = {
    Name = "Backend bucket"
    Environment = "Dev"
  }
}
####################################
#    Configuration
####################################
resource "aws_s3_bucket_versioning" "backend-versioning" {
  bucket = aws_s3_bucket.jin_bucket.id

  # 파일 업데이트 시 새 버전 생성
  versioning_configuration {
    status = "Enabled"
  }
}

resource "aws_s3_bucket_server_side_encryption_configuration" "backend-encryption" {
  bucket = aws_s3_bucket.jin_bucket.id

  # 서버 측 암호화 설정
  rule {
    apply_server_side_encryption_by_default {
    sse_algorithm = "AES256"
    }
  }
}
```

```
resource "aws_s3_bucket_public_access_block" "backend-public-access" {
  bucket = aws_s3_bucket.jin_bucket.id

  # S3 버킷 퍼블릭 액세스 차단 구성 관리
  block_public_acls       = true
  block_public_policy     = true
  ignore_public_acls      = true
  restrict_public_buckets = true
}
```

[dynamodb.tf]

```
####################################
#     DynamoDB
####################################
resource "aws_dynamodb_table" "backend-dynamodb" {
  name           = "backend-dynamodb"
  billing_mode   = "PAY_PER_REQUEST"
  hash_key       = "LockID"

  attribute {
    name = "LockID"
    type = "S"
  }
}
```

[outputs.tf]

```
output "jin_bucket_id" {
  value = aws_s3_bucket.jin_bucket.id
  description = "The ID of the S3 bucket"
}
output "jin_bucket_arn" {
  value = aws_s3_bucket.jin_bucket.arn
  description = "The ARN of the S3 bucket"
}
```

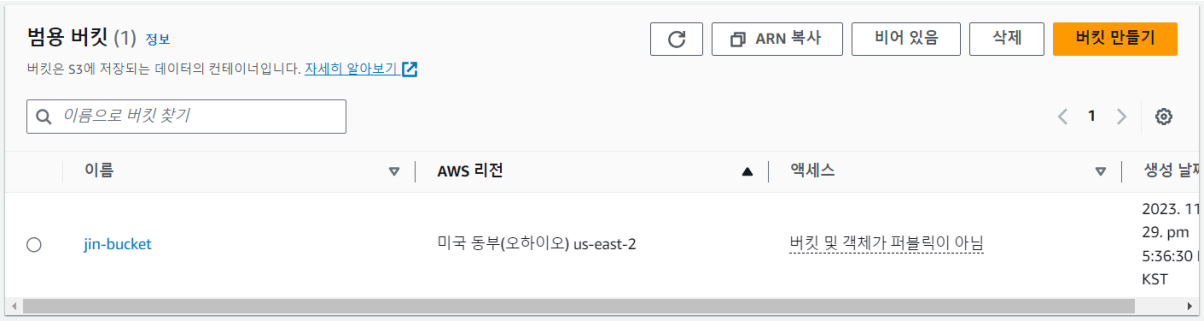
[main.tf] – (1)

```
#################### Terraform Configuration ####################
terraform {
  required_providers {
    aws = {
      source = "hashicorp/aws"
```

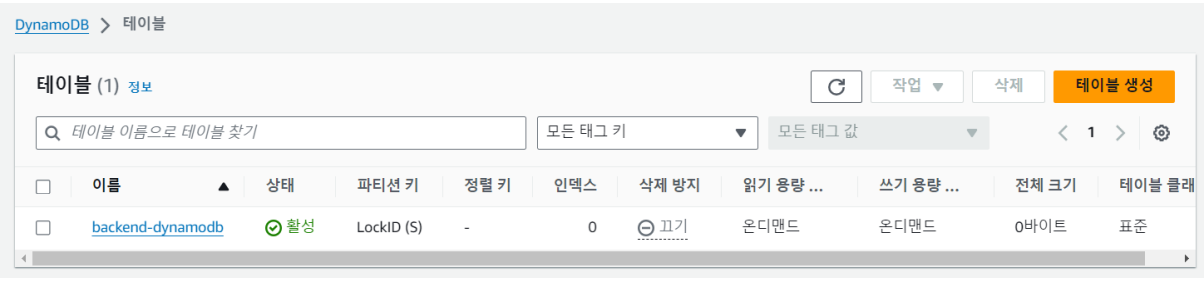
```
      version = "5.26.0"
    }
  }

  backend "s3" {
    bucket         = "jin-bucket" # S3 버킷 이름
    key            = "vpc/terraform.tfstate" # tfstate 저장 경로
    region         = "us-east-2"
    dynamodb_table = "backend-dynamodb" # dynamodb table 이름
  }
}
provider "aws" {
  region = "us-east-2"
}
```

Backend를 설정할 때 S3 버킷과 DynamoDB가 없는 경우 에러가 발생하므로 먼저 S3와 DynamoDB를 생성해주어야 한다. DynamoDB를 생성해주어야 하는 이유는 다수의 개발자가 각자 만든 것을 동시에 apply 했을 때의 충돌을 방지하기 위함이다. DynamoDB가 있을 때 S3는 Locking 기능을 제공한다.

S3 생성 확인

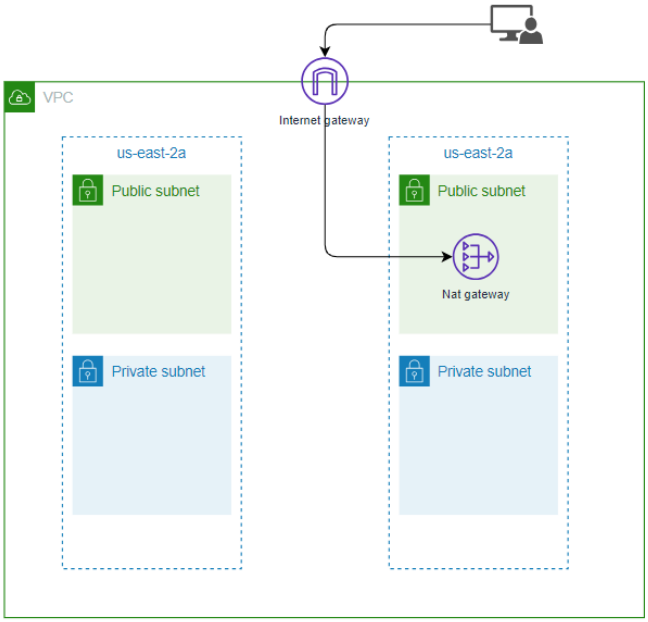


DynamoDB 생성 확인



3. 네트워크 구성

생성할 서비스들의 테스트를 위해 기본 네트워크 환경을 구축한다.



[STEP1]

하나의 VPC에 2개의 가용 영역을 생성한 후, 2개의 Public 서브넷과 Private 서브넷을 위치시킨다.

```
####################################
#     VPC
####################################
resource "aws_vpc" "vpc" {
   cidr_block           = "10.0.0.0/16"
   instance_tenancy = "default"
   enable_dns_hostnames = true

   tags = {
      Name = "vpc"
   }
}
####################################
#     Public Subnet
####################################
resource "aws_subnet" "public_subnet1" {
   vpc_id         = var.vpc-id
   cidr_block = "10.0.1.0/24"
   availability_zone = "us-east-2a"

   tags = {
      Name = "public_subnet1"
```

```
  }
}
resource "aws_subnet" "public_subnet2" {
  vpc_id        = var.vpc-id
  cidr_block = "10.0.2.0/24"
  availability_zone = "us-east-2b"
  map_public_ip_on_launch = true

  tags = {
    Name = "public_subnet2"
  }
}
#####################################
#     Private Subnet
#####################################
resource "aws_subnet" "private_subnet1" {
  vpc_id        = var.vpc-id
  cidr_block = "10.0.11.0/24"
  availability_zone = "us-east-2a"

  tags = {
    Name = "private_subnet1"
  }
}
resource "aws_subnet" "private_subnet2" {
  vpc_id        = var.vpc-id
  cidr_block = "10.0.12.0/24"
  availability_zone = "us-east-2b"

  tags = {
    Name = "private_subnet2"
  }
}
```

[STEP2]

Nat 게이트웨이를 통해 인터넷에 접근할 수 있도록 설정하고, 외부로부터 들어오는 트래픽을 ALB 로 처리한다.

```
#####################################
#     Internet Gateway
```

```
####################################
resource "aws_internet_gateway" "internet_gateway" {
  vpc_id = var.vpc-id
  tags = {
    Name = "internet-gateway"
  }
}
####################################
#     Elastic IPs
####################################
resource "aws_eip" "elastic_ip" {
  # instance = aws_instance.web.id
  domain     = "vpc"
}
####################################
#     Nat Gateway
####################################
resource "aws_nat_gateway" "nat_gateway" {
  allocation_id = var.eip-id
  subnet_id         = var.pub-sub1-id
  tags = {
    Name = "nat_gateway"
  }
}

####################################
#     Public Route Table
####################################
resource "aws_route_table" "public_route_table" {
  vpc_id = var.vpc-id

  route {
    cidr_block = "0.0.0.0/0"
    gateway_id = var.igw-id
  }

  tags = {
    Name = "public_route_table"
  }
```

```
}
####################################
#     Association
####################################
resource "aws_route_table_association" "public-rt-association1" {
  subnet_id      = var.pub-sub2-id
  route_table_id = var.pub-rt-id
}
resource "aws_route_table_association" "public-rt-association2" {
  subnet_id      = var.pub-sub1-id
  route_table_id = var.pub-rt-id
}

####################################
#     Private Route Table
####################################
resource "aws_route_table" "private_route_table" {
  vpc_id = var.vpc-id

  route {
    cidr_block = "0.0.0.0/0"
    gateway_id = var.ngw-id
  }

  tags = {
    Name = "private_route_table"
  }
}
####################################
#     Association
####################################
resource "aws_route_table_association" "private-rt-association1" {
  subnet_id      = var.pri-sub1-id
  route_table_id = var.pri-rt-id
}
resource "aws_route_table_association" "private-rt-association2" {
  subnet_id      = var.pri-sub2-id
  route_table_id = var.pri-rt-id
}
```

[vpc/outputs.tf]

```
output "vpc_id" {
  description = "VPC ID"
  value = aws_vpc.vpc.id
}
output "public_subnet1_id" {
  description = "Public Subnet1 ID"
  value = aws_subnet.public_subnet1.id
}
output "public_subnet2_id" {
  description = "Public Subnet2 ID"
  value = aws_subnet.public_subnet2.id
}
output "private_subnet1_id" {
  description = "Private Subnet1 ID"
  value = aws_subnet.private_subnet1.id
}
output "private_subnet2_id" {
  description = "Private Subnet2 ID"
  value = aws_subnet.private_subnet2.id
}
output "public_route_table_id" {
  description = "Public Route Table ID"
  value = aws_route_table.public_route_table.id
}
output "private_route_table_id" {
  description = "Private Route Table ID"
  value = aws_route_table.private_route_table.id
}
output "internet_gateway_id" {
  description = "Internet Gateway ID"
  value = aws_internet_gateway.internet_gateway.id
}
output "nat_gateway_id" {
  description = "Nat Gateway ID"
  value = aws_nat_gateway.nat_gateway.id
}
output "elastic_ip_id" {
```

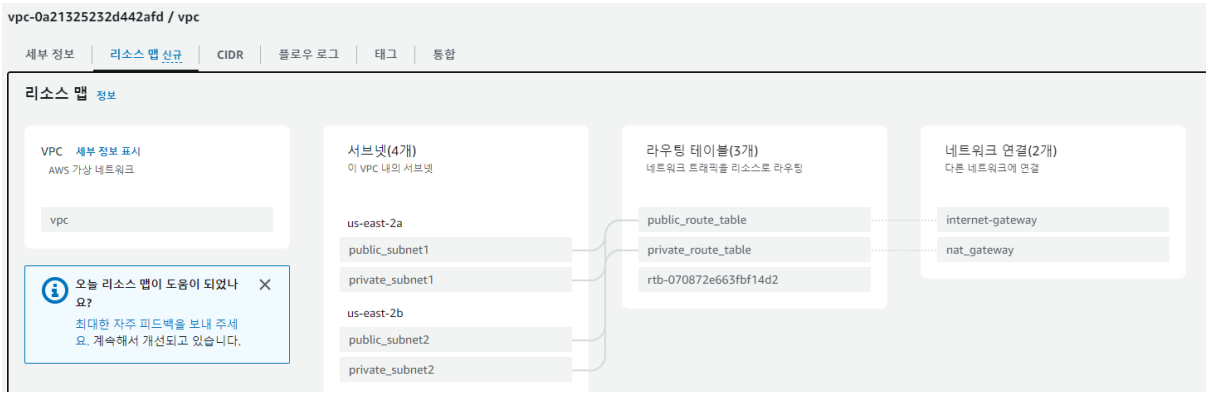
```
  description = "Elastic IP ID"
  value = aws_eip.elastic_ip.id
}
```

[vpc/variable.tf]

```
variable "vpc-id" {
  description = "VPC ID"
  type        = string
}
variable "pub-sub1-id" {
  description = "Public Subnet1 ID"
  type        = string
}
variable "pub-sub2-id" {
  description = "Public Subnet2 ID"
  type        = string
}
variable "pri-sub1-id" {
  description = "Private Subnet1 ID"
  type        = string
}
variable "pri-sub2-id" {
  description = "Private Subnet2 ID"
  type        = string
}
variable "pub-rt-id" {
  description = "Public Route Table ID"
  type        = string
}
variable "pri-rt-id" {
  description = "Private Route Table ID"
  type        = string
}
variable "igw-id" {
  description = "Internet Gateway ID"
  type        = string
}
variable "ngw-id" {
  description = "Nat Gateway ID"
```

```
  type        = string
}
variable "eip-id" {
  description = "Elastic ID"
  type        = string
}
```

VPC 생성 확인



앞서 생성한 S3에 데이터 소스가 정상적으로 저장된 것을 확인할 수 있다.

4. Bastion 인스턴스 생성

이미지 생성 및 실행할 Bastion 인스턴스를 생성한다.

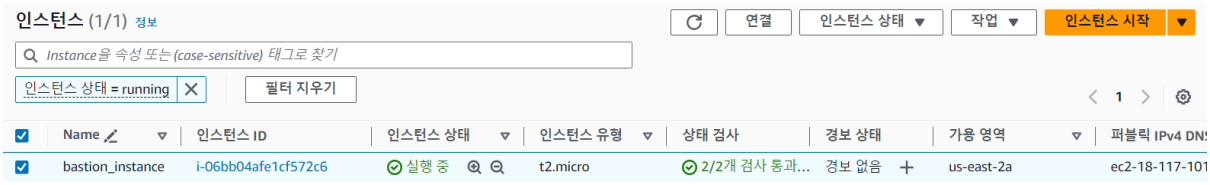
[STEP1]

Userdata.template를 작성한다.

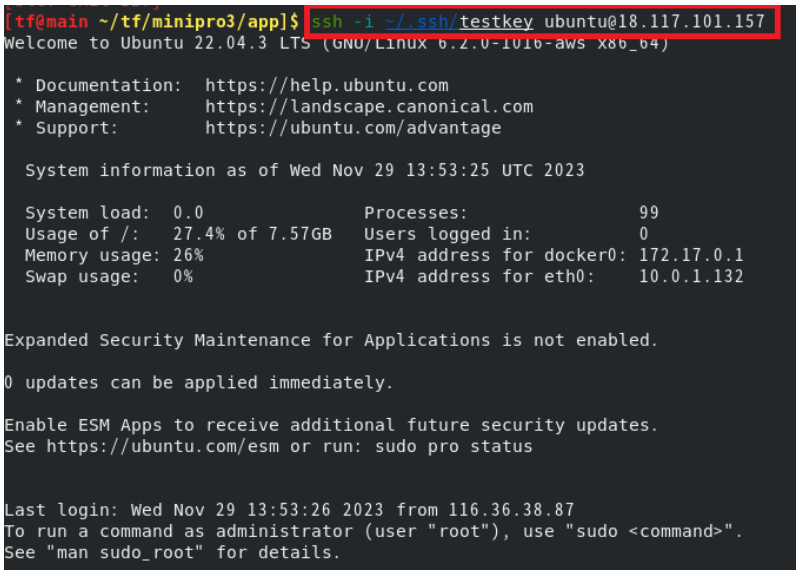
```
#!/bin/bash
# 시스템 업데이트 및 필수 패키지 설치
sudo apt-get update -y && ₩
sudo apt-get install -y ₩
```

```
    ca-certificates ₩
    curl ₩
    gnupg ₩
    lsb-release
# Docker GPG 키 추가 및 Docker 저장소 설정
sudo mkdir -p /etc/apt/keyrings && ₩
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg && ₩
echo ₩
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux/ubuntu ₩
  $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
# Docker 및 Docker Compose 설치
sudo apt-get install -y docker docker-compose docker-doc
# 현재 사용자를 Docker 그룹에 추가
sudo usermod -aG docker ubuntu
# Docker 서비스 활성화
sudo systemctl enable --now docker
```

인스턴스 생성 확인



인스턴스 접속

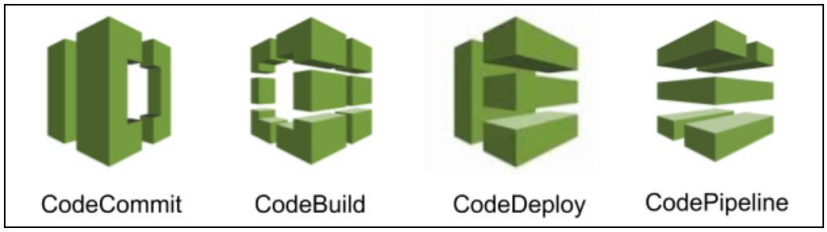


Docker 설치 확인

```
$ sudo usermod -aG docker ubuntu // 도커 접근 권한 부여
$ service docker start
$ service docker status
```

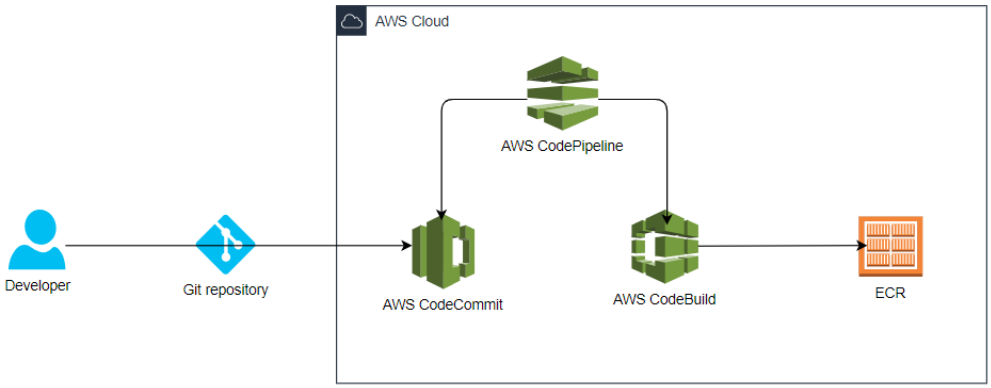
```
ubuntu@ip-10-0-1-132:~$ service docker status
● docker.service - Docker Application Container Engine
     Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset: enabled)
     Active: active (running) since Wed 2023-11-29 08:57:57 UTC; 5h 13min ago
TriggeredBy: ● docker.socket
       Docs: https://docs.docker.com
   Main PID: 2189 (dockerd)
      Tasks: 7
     Memory: 32.7M
        CPU: 1.870s
     CGroup: /system.slice/docker.service
             └─2189 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock
```

5. CI/CD 구축



AWS에서는 CI/CD를 구축하기 위해 네 가지 서비스를 이용할 수 있다.

- **CodeCommit** : 깃허브에서 AWS CodeCommit으로 소스 코드를 마이그레이션한다.
- **CodeBuild** : 소스 코드를 컴파일하고 테스트를 실행하며 배포 준비가 된 소프트웨어 패키지를 생성 및 관리한다.
- **CodePipeline** : 코드를 지속적으로 제공하는 파이프 라인을 구축한다.
- **CodeDeploy** : 코드를 EC2 서버에 배포한다.



[STEP1]

CI/CD 파이프라인 역할 및 정책 구성

[role.tf]

```
resource "aws_iam_role" "codebuild_role" {
  name = "jin_role"
```

```
  assume_role_policy = jsonencode({
    Version = "2012-10-17"
    Statement = [
      {
        Action = "sts:AssumeRole"
        Effect = "Allow"
        Sid    = ""
        Principal = {
          Service = [ "codebuild.amazonaws.com" ]
        }
      },
    ]
  })
}

resource "aws_iam_role" "pipeline_role" {
  name = "jin_role"

  assume_role_policy = jsonencode({
    Version = "2012-10-17"
    Statement = [
      {
        Action = "sts:AssumeRole"
        Effect = "Allow"
        Sid    = ""
        Principal = {
          Service = [ "codepipeline.amazonaws.com" ]
        }
      },
    ]
  })
}

resource "aws_iam_role" "trigger_role" {
  name = "jin_role"

  assume_role_policy = jsonencode({
    Version = "2012-10-17"
```

```
    Statement = [
      {
        Action = "sts:AssumeRole"
        Effect = "Allow"
        Sid    = ""
        Principal = {
          Service = ["events.amazonaws.com"]
        }
      },
    ]
  })
}

resource "aws_iam_role_policy_attachment" "codebuild-attach" {
    role       = aws_iam_role.codebuild_role.name
    policy_arn = var.codebuild-role-arn
}
resource "aws_iam_role_policy_attachment" "pipeline-attach" {
    role       = aws_iam_role.pipeline_role.name
    policy_arn = var.pipeline-role-arn
}
resource "aws_iam_role_policy_attachment" "trigger-attach" {
    role       = aws_iam_role.trigger_role.name
    policy_arn = var.pipeline-role-arn
}
```

[policy.tf]

```
resource "aws_iam_role_policy" "codebuild-policy" {
  name = "codebuild_policy"
  role = var.codebuild-role-id

  policy = jsonencode({
    Version = "2012-10-17"
    Statement = [
      {
        Action = [ # codebuild
          "codebuild:CreateReportGroup",
          "codebuild:CreateReport",
          "codebuild:UpdateReport",
          "codebuild:BatchPutTestCases",
```

```
            "codebuild:BatchPutCodeCoverages",
            "codebuild:BatchGetBuilds",
            "codebuild:StartBuild"
          ]
          Effect   = "Allow"
          Resource = [
            "arn:aws:codebuild:${var.region_name}:${var.account_id}:project/${var.subject}*"
          ]
        },
        {
          Action = [ # ECR Registry
            "ecr:GetAuthorizationToken",
            "ecr:BatchCheckLayerAvailability",
            "ecr:BatchGetImage",
            "ecr:GetDownloadUrlForLayer",
            "ecr:CompleteLayerUpload",
            "ecr:InitiateLayerUpload",
            "ecr:PutImage",
            "ecr:UploadLayerPart"
          ]
          Effect   = "Allow"
          Resource = [
            "${data.aws_ecr_repository.this.arn}"
          ]
        },
        {
          Action = [ # CloudWatchLogs
            "logs:CreateLogGroup",
            "logs:CreateLogStream",
            "logs:PutLogEvents"
          ]
          Effect   = "Allow"
          Resource = [
            "arn:aws:logs:${local.region_name}:${local.account_id}:log-group:${local.name}*:log-
stream:${local.name}*/*"
          ]
        }
      ]
    })
```

```
}
resource "aws_iam_role_policy" "pipeline_policy" {
  name = "pipeline_policy"
  role = var.pipeline-role-id

  policy = jsonencode({
    Version = "2012-10-17"
    Statement = [
      {
        Action = [ # S3
          "s3:*"
        ]
        Effect   = "Allow"
        Resource = [
          "${aws_s3_bucket.this.arn}", "${aws_s3_bucket.this.arn}/*"
        ]
      },
      {
        Action = [ # codecommitpull
          "codecommit:GitPull",
          "codecommit:GetBranch",
          "codecommit:GetCommit",
          "codecommit:GetRepository",
          "codecommit:GetUploadArchiveStatus",
          "codecommit:UploadArchive",
          "codecommit:CancelUploadArchive"
        ]
        Effect   = "Allow"
        Resource = [
          "${data.aws_codecommit_repository.this.arn}"
        ]
      }
    ]
  })
}
resource "aws_iam_role_policy" "trigger-policy" {
  name = "jin_policy"
  role = var.trigger-role-id
```

```
  policy = jsonencode({
    Version = "2012-10-17"
    Statement = [
      {
        Action = [
          "codepipeline:StartPipelineExecution"
        ]
        Effect   = "Allow"
        Resource = [
          "${data.aws_codecommit_repository.this.arn}"
        ]
      },
    ]
  })
}resource "aws_iam_role_policy" "pipeline_policy" {
  name = "pipeline_policy"
  role = var.pipeline-role-id

  policy = jsonencode({
    Version = "2012-10-17"
    Statement = [
      {
        Action = [ # S3
          "s3:*"
        ]
        Effect   = "Allow"
        Resource = [
          "${aws_s3_bucket.this.arn}", "${aws_s3_bucket.this.arn}/*"
        ]
      },
      {
        Action = [ # codecommitpull
          "codecommit:GitPull",
          "codecommit:GetBranch",
          "codecommit:GetCommit",
          "codecommit:GetRepository",
          "codecommit:GetUploadArchiveStatus",
          "codecommit:UploadArchive",
          "codecommit:CancelUploadArchive"
```

```
          ]
          Effect   = "Allow"
          Resource = [
            "${data.aws_codecommit_repository.this.arn}"
          ]
        }
      ]
    })
}
resource "aws_iam_role_policy" "trigger-policy" {
  name = "jin_policy"
  role = var.trigger-role-id

  policy = jsonencode({
    Version = "2012-10-17"
    Statement = [
      {
        Action = [
          "codepipeline:StartPipelineExecution"
        ]
        Effect   = "Allow"
        Resource = [
          "${data.aws_codecommit_repository.this.arn}"
        ]
      },
    ]
  })
}
```

[STEP2]

https://docs.aws.amazon.com/ko_kr/codepipeline/latest/userguide/file-reference.html
컨테이너 이미지로 빌드할 작업을 도커파일로 작성하고 표준 배포 작업을 위한 buildspec.yml 파일을 작성한다. 작성 내용은 빌드를 수행하고 ECR에 푸시하는 것이다.

[Dockerfile]

```
FROM openjdk:17-alpine
ARG JAR_FILE=/build/libs/*.jar
COPY ${JAR_FILE} /app.jar
EXPOSE 80
```

```
ENTRYPOINT ["java","-jar","-Dspring.profiles.active=prod", "/app.jar"]
```

[buildspec.yml]

```
version: 0.2
env:
  variables:
    APP_ENC: "dev"
    AWS_REGION: "us-east-2"
    AWS_ACCOUNT_ID: "035574589515"
phases:
  install:
    runtime-versions:
    java: corretto11
  pre_build:
      commands:
        - echo "export some envs and printenv"
        - GIT_TAG=$(git describe --tags --abbrev=0)
        - COMMIT_SHA_SIX=$(git rev-parse $GIT_TAG | cut -c 1-6)
        -       ECR_REPO_NAME="$AWS_ACCOUNT_ID.dkr.ecr.$AWS_REGION.amazonaws.com/jin-
pipeline"
        - IMAGE_TAG="$GIT_TAG-$COMMIT_JIN"
        - printenv | sort
        - echo "log in to Amazon ECR"
        - aws ecr get-login-password --region $AWS_DEFAULT_REGION | docker login --tirtir0827
AWS --lewis990322-stdin $AWS_ACCOUNT_ID.dkr.ecr.$AWS_DEFAULT_REGION.amazonaws.com
  build:
    commands:
      - ./gradlew build
      - mv build/libs/*.jar app.jar
      - echo "build the docker image"
      - docker build . -f env/dockerfile -t $ECR_REPO_NAME:$IMAGE_TAG
  post_build:
    commands:
      - docker push $ECR_REPO_NAME:$IMAGE_TAG
      - echo "completed this action"
```

기본 환경변수로 erc 배포 환경, aws 리전, aws 계정 ID를 선언하였다.

- prebuild : **ECR 로그인**
- build : **도커 이미지 빌드**

[STEP3]

[codebuild.tf]

```
####################################
#    CI 구축
####################################
resource "aws_codebuild_project" "codebuild" {
  name = "codebuild"
  description      = "Build Docker Image"
  build_timeout    = 10

  service_role = aws_iam_role.example.arn

  # codePipeLine을 사용하기에 따로 codeBuild용 아티팩트 버킷을 사용하지 않는다.
  artifacts {
    type = "NO_ARTIFACTS"
  }

  cache {
    type    = "S3"
    modes = [aws_s3_bucket.jin_backend]
  }

  environment {
    compute_type                = "BUILD_GENERAL1_SMALL"
    image                       = "aws/codebuild/amazonlinux2-x86_64-standard:4.0"
    type                        = "LINUX_CONTAINER"
    image_pull_credentials_type = "CODEBUILD"
    privileged_mode             = true
  }

  vpc_config {
    vpc_id = var.vpc-id

    subnets = [
      for k, v in var.subnet-id : data.aws_subnet.this[k].id
    ]

    security_group_ids = [
      data.aws_security_group.was_security_group.id
```

```
    ]
  }

  source {
    type                = "GITHUB"
    location            = "https://github.com/TirTir/SKUPLATE.git"
    git_clone_depth = 1
  }

  tags = {
    Environment = "Test"
  }
}
```

[STEP4]

CodePipleline은 CodePipeline Amazon CloudWatch Events를 사용하여 변경 사항이 있는 소스 코드를 감지하고 파이프라인을 시작한다. 파이프라인에는 두 개 이상의 단계가 포함되어 있어야 한다. 첫 번째 단계는 소스 단계어야 하고, 빌드 또는 배포 단계가 하나 이상 있어야 한다.

```
resource "aws_codepipeline" "codepipeline" {
  name       = "codepipeline"
  role_arn = var.pipeline-role-arn

  artifact_store {
    location = aws_s3_bucket.jin-bucket.bucket
    type       = "S3"

    encryption_key {
      id     = data.aws_kms_alias.s3kmskey.arn
      type = "KMS"
    }
  }

  stage {
    name = "Source"

    action {
      name                = "Source"
      category            = "Source"
      owner               = "AWS"
```

```
      provider             = "GitHub"
      version              = "1"
      output_artifacts = ["source_output"]

      configuration = {
        FullRepositoryId = var.codecommit_repo_name
        BranchName           = "main"
        OutputArtifactFormat = "CODEBUILD_CLONE_REF"
      }
    }
  }

  stage {
    name = "Build"

    action {
      name                 = "Build"
      category             = "Build"
      owner                = "AWS"
      provider             = "CodeBuild"
      input_artifacts  = ["source_output"]
      output_artifacts = ["build_output"]
      version              = "1"

      configuration = {
        ProjectName = aws_codebuild_project.codebuild.id
      }
    }}

  stage {
    name = "Deploy"

    action {
      name                 = "Deploy"
      category             = "Deploy"
      owner                = "AWS"
      provider             = "CloudFormation"
      input_artifacts = ["build_output"]
      version              = "1"
```

```
        configuration = {
          ActionMode       = "REPLACE_ON_FAILURE"
          Capabilities     = "CAPABILITY_AUTO_EXPAND,CAPABILITY_IAM"
          OutputFileName = "test.json"
        }
      }
    }
}
```