.net app deploy HOL

프로젝트	CA LAB
중분류	공통
생성일	@June 19, 2024 7:33 PM
작성자	daredevil
작업자	daredevil
상태	작업중
업데이트	@June 20, 2024 6:56 PM

참고 소스파일 다운로드

GIT 설치

GIT 설치 가이드

샘플코드 다운로드

```
git clone https://github.com/MicrosoftDocs/Virtualization-Document
```

도커 설치

도커 설치 가이드

Dockerfile 생성

```
FROM mcr.microsoft.com/dotnet/core/sdk:2.1 AS build-env
WORKDIR /app

COPY *.csproj ./
RUN dotnet restore

COPY . ./
RUN dotnet publish -c Release -o out

FROM mcr.microsoft.com/dotnet/core/aspnet:2.1
WORKDIR /app
COPY --from=build-env /app/out .
ENTRYPOINT ["dotnet", "asp-net-getting-started.dll"]
```

Dockerfile 설명

```
FROM mcr.microsoft.com/dotnet/core/sdk:2.1 AS build-env
WORKDIR /app
```

첫 번째 줄은 컨테이너를 빌드하는 데 사용할 기본 이미지를 선언합니다. 로컬 시스템에 이 이미지가 없는 경우 docker는 자동으로 이미지를 가져오려고 시도합니다. mcr.microsoft.com/dotnet/core/sdk:2.1은 .NET 코어 2.1 SDK가 설치된 상태의 패키지로 제공되므로 버전 2.1을 대상으로 하는 ASP .NET 코어 프로젝트를 빌드하는 작업은 사용자에게 달려 있습니다. 다음 명령은 컨테이너의 작업 디렉터리를 /app로 변경하므로 이 명령 뒤에 나오는 모든 명령은 이 컨텍스트에서 실행됩니다.

```
COPY *.csproj ./
RUN dotnet restore
```

다음으로, 이 지침은 .csproj 파일을 빌드 환경 컨테이너의 /app 디렉터리로 복사합니다. 이 파일을 복사하면 .NET이 이 파일을 읽은 다음 프로젝트에 필요한 모든 종속 요소와 도구를 가져옵니다.

```
COPY . ./
RUN dotnet publish -c Release -o out
```

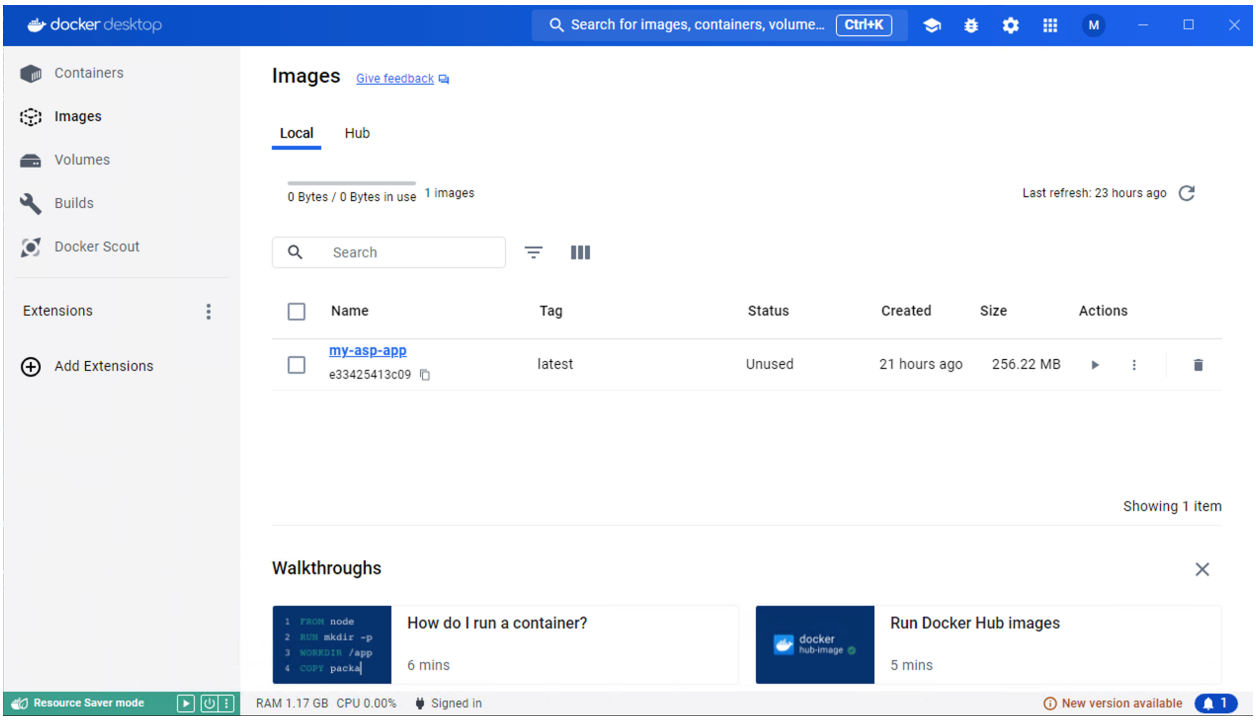
.NET이 모든 종속성을 빌드 환경 컨테이너로 가져오면 다음 명령은 모든 프로젝트 소스 파일을 컨테이너에 복사합니다. 그런 다음 .NET에 릴리스 구성으로 애플리케이션을 게시하고 출력 경로를 설정합니다.

```
FROM mcr.microsoft.com/dotnet/core/aspnet:2.1
WORKDIR /app
COPY --from=build-env /app/out .
ENTRYPOINT ["dotnet", "asp-net-getting-started.dll"]
```

애플리케이션이 ASP.NET이므로 이 런타임이 포함된 이미지를 지정합니다. 그런 다음 임시 컨테이너의 출력 디렉토리에 있는 모든 파일을 최종 컨테이너로 복사합니다. 컨테이너가 시작될 때 새 앱을 진입점으로 사용하여 실행되도록 컨테이너를 구성합니다.

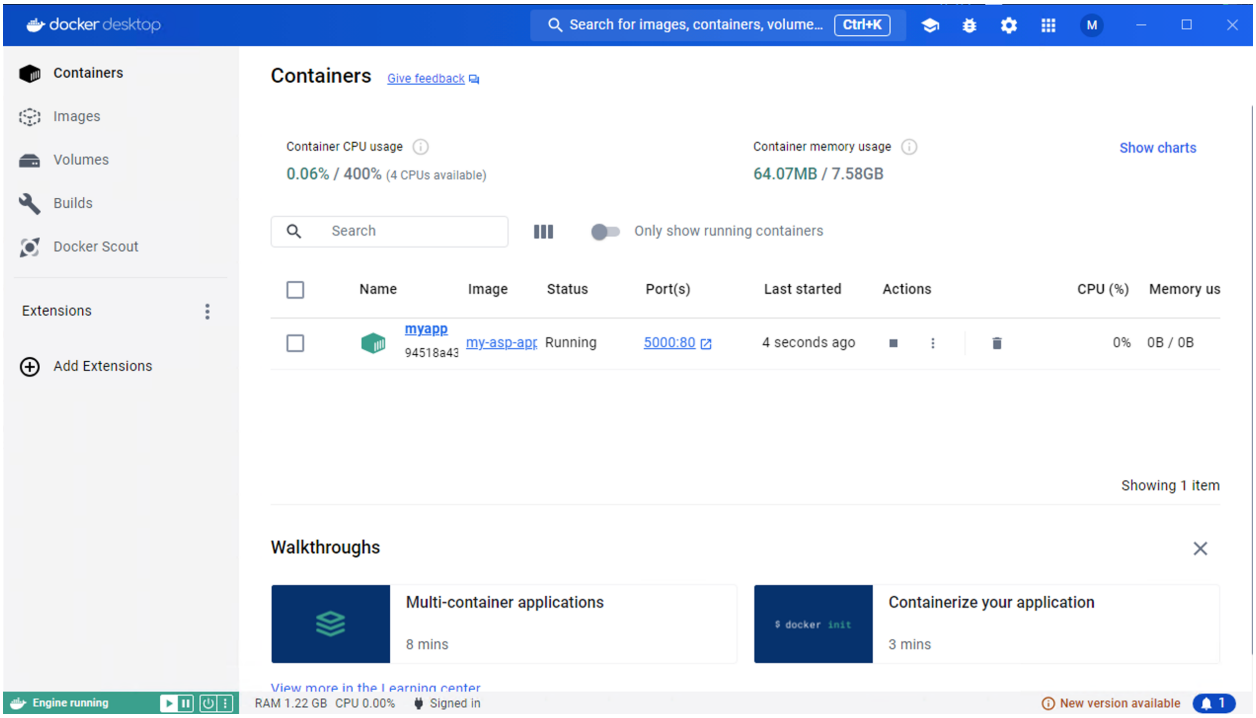
이미지 생성

```
docker build -t my-asp-app .
```

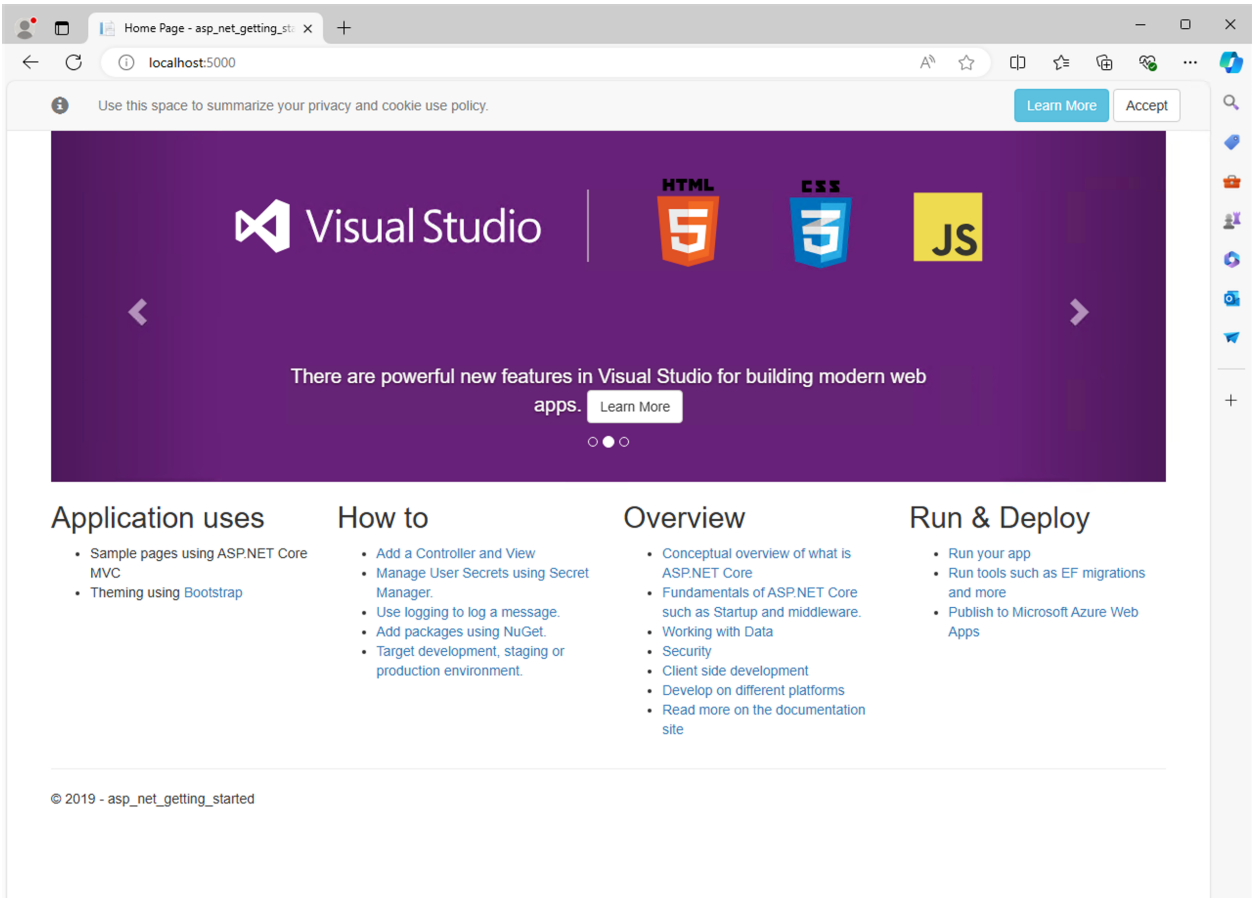


컨테이너 실행

```
docker run -d -p 5000:80 --name myapp my-asp-app
```



컨테이너 실행 확인



choco 설치

The Package Manager for Windows

choco 설치 가이드

AWS CLI 설치 및 구성

AWS 명령줄 인터페이스(AWS CLI)는 Amazon Web Services의 모든 부분과 상호 작용하기 위한 일관된 인터페이스를 제공하는 통합 도구

AWS CLI 설치 가이드

```
choco install awscli
```

AWS CLI를 구성

AWS CLI 구성 가이드

```
aws configure
```

ECR 로그인

```
aws ecr get-login-password --region eu-central-1 | docker login
--username AWS --password-stdin 585748631694.dkr.ecr.eu-central
-1.amazonaws.com
```

ECR 리포지토리 생성

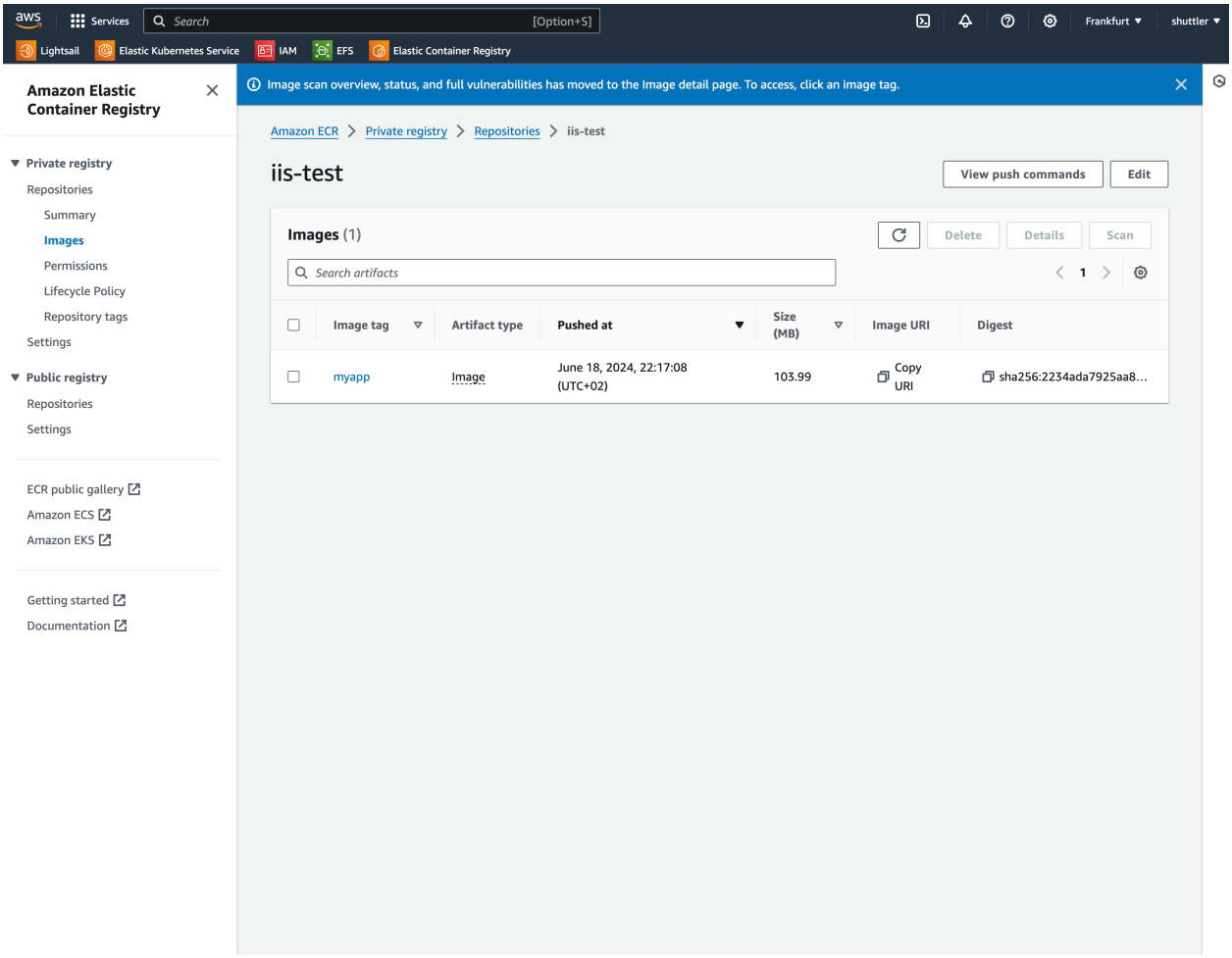
```
aws ecr create-repository --repository-name iis-test --region e
u-central-1
```

Docker 이미지 태그

```
docker tag my-asp-app:latest 585748631694.dkr.ecr.eu-central-1.
amazonaws.com/iis-test:myapp
```

1. **Docker 이미지 푸시**:

```
docker push 585748631694.dkr.ecr.eu-central-1.amazonaws.com/iis
-test:myapp
```



Test Cluster Creation

```
$Region = "eu-central-1"
$ClusterName = "iis-test-ingress"
$NodeCount = 3
$ServerSpec = "t3.medium"
$NodeGroupName = "iis-test-ingress-ng-1"

# Create eks Cluster$eksConfigContent = @"
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: $ClusterName
```

```
  region: $Region

managedNodeGroups:
  - name: $NodeGroupName
    instanceType: $ServerSpec
    desiredCapacity: $NodeCount
    privateNetworking: true
"@

# Write the configuration to a file$eksConfigPath = ".\eks-conf
ig.yaml"
$eksConfigContent | Out-File -FilePath $eksConfigPath -Encoding
utf8

# Create the cluster
& eksctl create cluster -f $eksConfigPath

# Remove the configuration fileRemove-Item $eksConfigPath
```

클러스터 생성 코드 설명

1. 변수 정의 먼저, 스크립트 전체에서 사용될 여러 변수를 정의합니다.

```
$Region = "eu-central-1"# The AWS region where the EKS cluster
will be created$ClusterName = "iis-ingress"# The name of the EK
S cluster$NodeCount = 3# The number of nodes in the node group
$ServerSpec = "t3.medium"# The instance type for the nodes$Node
GroupName = "iis-ingress-ng-1"# The name of the node group
```

1. EKS 클러스터 구성 생성 다음으로 정의된 변수를 사용하여 EKS 클러스터 구성 콘텐츠를 생성합니다. 이 구성은 eksctl클러스터를 정의하기 위한 도구 형식을 따릅니다.

```
$eksConfigContent = @"
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig
```

```
metadata:
  name: $ClusterName
  region: $Region

managedNodeGroups:
  - name: $NodeGroupName
    instanceType: $ServerSpec
    desiredCapacity: $NodeCount
    privateNetworking: true
"@
```

1. 파일에 구성 쓰기 EKS 구성 콘텐츠를 eks-config.yaml에 저장합니다. 이 파일은 eksctl클러스터를 생성하는 데 사용됩니다 .

```
$eksConfigPath = ".\eks-config.yaml"# Define the file path for the configuration$eksConfigContent | Out-File -FilePath $eksConfigPath -Encoding utf8# Write the content to the file
```

1. 클러스터 생성 구성 파일이 준비되었으므로 이제 eksctl create cluster명령과 구성 파일을 사용하여 EKS 클러스터를 생성할 수 있습니다.

```
& eksctl create cluster -f $eksConfigPath
```

1. 구성 파일 제거 마지막으로 클러스터 생성 후에는 더 이상 필요하지 않으므로 구성 파일을 제거하여 정리합니다.

```
Remove-Item $eksConfigPath# Delete the configuration file
```

IAM Policy & IAM OIDC Creation

```
$Region = "eu-central-1"
$ClusterName = "iis-test-ingress"
$AccountId = (aws sts get-caller-identity --query Account --output text)

# Download IAM policy for ALB Ingress ControllerInvoke-WebReque
```

```
st -Uri "https://raw.githubusercontent.com/kubernetes-sigs/aws-
load-balancer-controller/v2.7.2/docs/install/iam_policy.json" -
OutFile "iam_policy.json"

# Create IAM policy$PolicyArn = (aws iam create-policy --policy
-name AWSLoadBalancerControllerIAMPolicy --policy-document fil
e://iam_policy.json).Policy.Arn

# Create IAM OIDC provider for the cluster$oidc_id = $(aws eks
describe-cluster --name $ClusterName --query "cluster.identity.
oidc.issuer" --output text | Out-String | Split-Path -Leaf)
if (-not (aws iam list-open-id-connect-providers | Select-Strin
g $oidc_id)) {
    eksctl utils associate-iam-oidc-provider --cluster $Cluster
Name --approve
}
```

코드의 목적

1. **IAM 정책 다운로드**: ALB Ingress Controller에 필요한 IAM 정책을 다운로드합니다.
2. **IAM 정책 생성**: 다운로드한 정책을 사용하여 새로운 IAM 정책을 생성합니다.
3. **IAM OIDC 제공자 생성**: EKS 클러스터에 OIDC 제공자를 설정하여 IAM 역할과 Kubernetes 서비스 계정을 연결합니다.

상세 설명

1. **변수 설정**

```
$Region = "eu-central-1"
$ClusterName = "iis-test-ingress"
$AccountId = (aws sts get-caller-identity --query Account --
output text)
```

- `Region`: AWS 리전 설정 (예: `eu-central-1`).
- `ClusterName`: EKS 클러스터 이름 설정 (예: `iis-ingress`).
- `AccountId`: AWS 계정 ID를 가져옵니다.

2. **IAM 정책 다운로드**

```
Invoke-WebRequest -Uri "https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-controller/v2.7.2/docs/install/iam_policy.json" -OutFile "iam_policy.json"
```

- ALB Ingress Controller에 필요한 IAM 정책을 GitHub에서 다운로드하여 `iam_policy.json` 파일로 저장합니다.

3. **IAM 정책 생성**

```
$PolicyArn = (aws iam create-policy --policy-name AWSLoadBalancerControllerIAMPolicy --policy-document file://iam_policy.json).Policy.Arn
```

- 다운로드한 정책 파일을 사용하여 새로운 IAM 정책을 생성하고, 생성된 정책의 ARN을 변수 `$PolicyArn`에 저장합니다.

4. **IAM OIDC 제공자 생성**

```
$oidc_id = $(aws eks describe-cluster --name $ClusterName --query "cluster.identity.oidc.issuer" --output text | Out-String | Split-Path -Leaf)
if (-not (aws iam list-open-id-connect-providers | Select-String $oidc_id)) {
    eksctl utils associate-iam-oidc-provider --cluster $ClusterName --approve
}
```

- 클러스터의 OIDC 제공자 ID를 가져와 `$oidc_id` 변수에 저장합니다.
- OIDC 제공자가 존재하지 않는 경우, `eksctl` 명령어를 사용하여 클러스터에 OIDC 제공자를 설정합니다.

IAM Role Creation

아래의 json파일에서 Principal.Federated의 Value를 정보에 맞게 수정해야하고, Condition.StringEquals의 필드 키값을 oidc.eks... 값을 키에다가 설정해주어야함. 뒤에 :aud, :sub 바꾸면 안되는 것 주의

```
//load-balancer-role-trust-policy.json{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Federated": "arn:aws:iam::585748631694:oidc-pr
ovider/oidc.eks.eu-central-1.amazonaws.com/id/46F2917928CB2CF19
56FD1EE9449B48B"
            },
            "Action": "sts:AssumeRoleWithWebIdentity",
            "Condition": {
                "StringEquals": {
                    "oidc.eks.eu-central-1.amazonaws.com/id/46F
2917928CB2CF1956FD1EE9449B48B:aud": "sts.amazonaws.com",
                    "oidc.eks.eu-central-1.amazonaws.com/id/46F
2917928CB2CF1956FD1EE9449B48B:sub": "system:serviceaccount:kube
-system:aws-load-balancer-controller"
                }
            }
        }
    ]
}
```

코드의 목적

AWS EKS 클러스터에서 ALB Ingress Controller를 실행하기 위한 IAM 역할 신뢰 정책(trust policy)을 정의합니다. 이 정책은 특정 IAM 역할이 OIDC(OpenID Connect) 제공자를 통해 인증된 주체(예: Kubernetes 서비스 계정)에게 역할을 맡길 수 있도록 허용합니다.

1. **IAM 역할 신뢰 정책 정의**: ALB Ingress Controller가 EKS 클러스터에서 제대로 작동할 수 있도록 필요한 IAM 역할을 설정합니다.
2. **OIDC 제공자와 연계**: OIDC 제공자를 통해 인증된 Kubernetes 서비스 계정에게 IAM 역할을 위임할 수 있도록 설정합니다.

- Effect: 이 정책의 효과를 나타냅니다. Allow는 특정 작업을 허용함을 의미합니다.
- Principal: 역할을 맡을 수 있는 주체를 정의합니다.

    - Federated: OIDC 제공자의 ARN을 지정합니다. 이 ARN은 사용자의 AWS 계정 ID와 EKS 클러스터의 OIDC 제공자 ID로 구성됩니다.
- Action: 허용된 작업을 정의합니다.
    - sts:AssumeRoleWithWebIdentity: 웹 ID를 통해 역할을 맡는 작업을 허용합니다.
- Condition: 추가 조건을 정의합니다.
    - StringEquals: 주어진 문자열과 동일한 조건을 검사합니다.
        - oidc.eks.region-code.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE:aud: OIDC 제공자의 audience가 sts.amazonaws.com이어야 합니다.
        - oidc.eks.region-code.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE:sub: 주체가 system:serviceaccount:kube-system:aws-load-balancer-controller 여야 합니다.

```
# aws-load-balancer-controller-service-account.yamlapiVersion:
v1
kind: ServiceAccount
metadata:
  labels:
    app.kubernetes.io/component: controller
    app.kubernetes.io/name: aws-load-balancer-controller
  name: aws-load-balancer-controller
  namespace: kube-system
  annotations:
    eks.amazonaws.com/role-arn: arn:aws:iam::585748631694:role/
AmazonEKSLoadBalancerControllerRole
```

코드의 목적

AWS EKS 클러스터에서 AWS Load Balancer Controller를 실행하기 위한 Kubernetes 서비스 계정을 정의합니다. 이 서비스 계정은 특정 IAM 역할을 사용할 수 있도록 설정되어 있습니다.

1. **Kubernetes 서비스 계정 생성**: AWS Load Balancer Controller를 실행하기 위한 서비스 계정을 생성합니다.
2. **IAM 역할 연결**: 생성된 서비스 계정이 특정 IAM 역할을 사용할 수 있도록 설정합니다.

- name: 서비스 계정의 이름을 정의합니다.
- namespace: 서비스 계정이 속할 네임스페이스를 정의합니다.
- annotations: 서비스 계정에 대한 추가 정보를 정의합니다. 여기서는 특정 IAM 역할을 사용할 수 있도록 설정합니다.

```
# Create role
aws iam create-role --role-name AmazonEKSLoadBalancerController
Role --assume-role-policy-document file://"load-balancer-role-t
rust-policy.json"

# Attach role
aws iam attach-role-policy --policy-arn arn:aws:iam::5857486316
94:policy/AWSLoadBalancerControllerIAMPolicy --role-name Amazon
EKSLoadBalancerControllerRole

# Service Account 수동 생성
kubectl apply -f aws-load-balancer-controller-service-account.y
aml
```

Cert-Manager

```
kubectl apply --validate=false -f https://github.com/jetstack/c
ert-manager/releases/download/v1.13.5/cert-manager.yaml
```

AWS Load Balancer Controller Installation and Configuration

```
Invoke-WebRequest -Uri "https://github.com/kubernetes-sigs/aws-
load-balancer-controller/releases/download/v2.7.2/v2_7_2_full.y
aml" -OutFile "v2_7_2_full.yaml"
```

```


---
apiVersion: v1 # 해당 부분 지워주기
kind: ServiceAccount # 해당 부분 지워주기metadata:# 해당 부분 지워주기
```

```
labels:# 해당 부분 지워주기app.kubernetes.io/component: controller# 해당 부분 지워주기app.kubernetes.io/name: aws-load-balancer-controller# 해당 부분 지워주기name: aws-load-balancer-controller# 해당 부분 지워주기namespace: kube-system# 해당 부분 지워주기---
...
---
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app.kubernetes.io/component: controller
    app.kubernetes.io/name: aws-load-balancer-controller
  name: aws-load-balancer-controller
  namespace: kube-system
spec:
  replicas: 1
  selector:
    matchLabels:
      app.kubernetes.io/component: controller
      app.kubernetes.io/name: aws-load-balancer-controller
  template:
    metadata:
      labels:
        app.kubernetes.io/component: controller
        app.kubernetes.io/name: aws-load-balancer-controller
    spec:
      containers:
      - args:
        - --cluster-name=iis-joker-ingress# 이 부분에서는 args부분에서 cluster name 수정
        - --ingress-class=alb
        - --aws-vpc-id=vpc-054296c296a3d1e3d# vpcid 기입
        - --aws-region=eu-central-1# region 기입
        image: public.ecr.aws/eks/aws-load-balancer-controller:v2.7.2
```

```
# Apply the manifest
kubectl apply -f .\v2_7_2_full.yaml

# Download IngressClass and IngressClassParams manifest
Invoke-WebRequest -Uri "https://github.com/kubernetes-sigs/aws-
load-balancer-controller/releases/download/v2.7.2/v2_7_2_ingcla
ss.yaml" -OutFile "v2_7_2_ingclass.yaml"

# Apply IngressClass and IngressClassParams manifest
kubectl apply -f .\v2_7_2_ingclass.yaml

# Check if it is applied well
kubectl get deployment -n kube-system aws-load-balancer-control
ler
```

```
# DotNet Example
kubectl apply -f dotnet-deployment.yaml
```

```
# dotnet-deployment.yamlapiVersion: apps/v1
kind: Deployment
metadata:
  name: my-asp-app-deployment
  labels:
    app: my-asp-app
spec:
  replicas: 2
  selector:
    matchLabels:
      app: my-asp-app
  template:
    metadata:
      labels:
        app: my-asp-app
    spec:
      containers:
```

```
      - name: my-asp-app
        image: 585748631694.dkr.ecr.eu-central-1.amazonaws.com/
iis-test:myapp
        ports:
        - containerPort: 80
---
apiVersion: v1
kind: Service
metadata:
  name: my-asp-app-service
  labels:
    app: my-asp-app
spec:
  type: LoadBalancer
  ports:
  - port: 80
    targetPort: 80
  selector:
    app: my-asp-app
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: my-asp-app-ingress
  annotations:
    alb.ingress.kubernetes.io/scheme: internet-facing
    alb.ingress.kubernetes.io/target-type: ip
spec:
  ingressClassName: alb
  rules:
  - http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: my-asp-app-service
```

```
            port:
              number: 80
```

Deployment

Deployment는 Kubernetes에서 애플리케이션의 배포와 관리를 자동화하는 객체입니다. Deployment를 사용하면 애플리케이션의 복제본을 관리하고, 상태를 모니터링할 수 있습니다.

주요 기능

- **애플리케이션 가용성 보장**: 애플리케이션 인스턴스의 원하는 상태를 보장합니다.
- **스케일링**: 수동 또는 자동으로 애플리케이션의 인스턴스 수를 조정할 수 있습니다.

Service

Service는 Kubernetes 클러스터 내에서 네트워크 서비스의 추상화를 제공합니다. Service는 Pod 집합에 대한 안정적인 네트워크 엔드포인트를 정의하며, 클러스터 내부 및 외부에서 접근할 수 있습니다.

주요 기능

- **로드 밸런싱**: 여러 Pod에 트래픽을 분산시킵니다.
- **서비스 발견**: DNS를 통해 서비스 이름으로 Pod에 접근할 수 있습니다.
- **유형**: ClusterIP, NodePort, LoadBalancer, ExternalName 등 다양한 유형이 있습니다.

Ingress

Ingress는 클러스터 외부에서 내부 서비스로의 HTTP 및 HTTPS 경로를 설정하는 객체입니다. Ingress를 사용하면 하나의 IP 주소에 여러 서비스를 노출할 수 있으며, 로드 밸런싱, SSL/TLS 종료, 이름 기반의 가상 호스팅을 지원합니다.

주요 기능

- **로드 밸런싱**: HTTP 및 HTTPS 트래픽을 여러 서비스에 분산시킵니다.
- **SSL/TLS**: SSL/TLS 인증서를 사용하여 보안을 설정할 수 있습니다.
- **호스트 및 경로 기반 라우팅**: 특정 호스트 및 경로에 따라 트래픽을 라우팅합니다.

Sample Deployment Test

```
# 2048 Game Example
kubectl apply -f https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-controller/v2.7.2/docs/examples/2048/2048_full.yaml

# check ingress

kubectl get ingress/ingress-2048 -n game-2048
```