`

서버 구축 기술 가이드

㈜SSTC

사용 기술 가이드

2023. 10. 20



목 차

[1. 개요 3](#_Toc151049963)

[1.1. 기술 가이드 소개 3](#_Toc151049964)

[1.2. 사용 기술 3](#_Toc151049965)

[1.2.1. EKS 3](#_Toc151049966)

[1.2.2. Docker 4](#_Toc151049967)

[1.2.3. Ingress 4](#_Toc151049968)

[1.2.4. Horizontal Pod Autoscaler(HPA) 5](#_Toc151049969)

[1.2.5. Cluster AutoScaler(Karpenter) 5](#_Toc151049970)

[2. 구성도 5](#_Toc151049971)

[3. 기술 구현 6](#_Toc151049972)

[3.1. Dockerfile구성 및 이미지 배포 6](#_Toc151049973)

[3.1.1. Docker 작업 폴더 생성 및 Dockerfile 작성 6](#_Toc151049974)

[3.2. EKS 웹 서버 구성 및 배포 10](#_Toc151049975)

[3.2.1. User, Admin 웹 서버 배포 yaml 작성 10](#_Toc151049976)

[3.2.2. User,Admin 서버 외부 연결용 서비스 생성 14](#_Toc151049977)

[3.2.3. 웹 서버 deploymet, service 배포 14](#_Toc151049978)

[3.3. 웹 서버 Ingress 연결 구성 및 배포 15](#_Toc151049979)

[3.3.1. Ingress Contorller 역할 생성 및 연결 15](#_Toc151049980)

[ingress controller에 IRSA 설정 15](#_Toc151049981)

[3.3.2. Helm 차트로 aws-load-balancer-controller 설치 16](#_Toc151049982)

[3.3.3. User, Admin 서비스에 대한 Ingress.yaml 작성 16](#_Toc151049983)

[3.3.4. Ingress 배포 상태 확인 17](#_Toc151049984)

[3.4. Horizontal Pod Autoscaling(HPA) 설정 및 배포 18](#_Toc151049985)

[3.4.1. Metric-server 설치 및 확인 18](#_Toc151049986)

[3.4.2. User-Server 에 대해 HPA.yaml 파일 작성 18](#_Toc151049987)

[3.4.3. Admin-Server에 대한 Hpa.yaml 작성 19](#_Toc151049988)

[3.4.4. HPA 배포 및 상태 확인 20](#_Toc151049989)

[3.5. Karpenter 설정 및 배포 20](#_Toc151049990)

[3.5.1. Karpenter 설치 전 필요한 환경 변수 지정 20](#_Toc151049991)

[3.5.2. IAM 역할,정책 생성 및 적용 21](#_Toc151049992)

[3.5.3. Node Group 및 Security Group 연결 26](#_Toc151049993)

[3.5.4. Karpenter 설치 및 provisioner 배포 27](#_Toc151049994)

[4. 구현 테스트 31](#_Toc151049995)

[4.1. 웹 서버 배포 상태 및 Ingress 경로 테스트 31](#_Toc151049996)

[4.1.1. Deployment, SVC, Ingress 정상 동작 확인 31](#_Toc151049997)

[4.1.2. ALB 주소에 접속하여 웹 서버 접근 테스트 31](#_Toc151049998)

[4.2. HPA - Pod AutoScaling 테스트 32](#_Toc151049999)

[4.2.1. Metric Server 및 HPA 정상 동작 확인 32](#_Toc151050000)

[4.2.2. Web-server의 Cpu, Memory 임계치 넘은 경우 Scale out 확인 33](#_Toc151050001)

[4.2.3. 임계치 밑으로 내려 갔을 경우 Scale in 확인 34](#_Toc151050002)

[4.3. Karpenter - Cluster AutoScaling 테스트 34](#_Toc151050003)

[4.3.1. Node 장애 발생 상황 34](#_Toc151050004)

[4.3.2. Node 리소스 부족으로 Pod pending 상황 35](#_Toc151050005)

# 개요

## 기술 가이드 소개

본 모니터링 구축 기술 가이드는 “2팀(SSTC)\_쇼핑몰\_마이그레이션\_및\_고도화.dox”에서 구성 된 AWS 인프라 환경 및 EKS 클러스터 구축 환경에서 필요한 웹 서버를 구축하기 위한 설정 및 AWS 연동 방안에 대해 기술한다.

## 사용 기술

### EKS

* EKS는 사용자가 쿠버네티스를 더 쉽게 사용하고 관리할 수 있도록 도와주는 AWS 서비스입니다. 쿠버네티스 마스터 관리 및 노드를 프로비저닝하고 구성하는 복잡한 작업을 AWS가 대신 수행해 줍니다. 사용자는 애플리케이션을 배포하고 관리하기 위해 필요한 쿠버네티스 API를 사용할 수 있으며, AWS는 백그라운드에서 클러스터의 인프라스트럭처를 관리합니다.
* 기술 사용 이점
  + EKS는 관리형 쿠버네티스 서비스로, 쿠버네티스 마스터를 AWS가 관리하므로 사용자는 클러스터의 관리에 집중할 필요가 없습니다
  + EKS는 다양한 AWS 서비스와 통합되어 풍부한 확장성과 기능을 제공합니다
  + 쿠버네티스 서비스를 사용하기 때문에, 여러 서비스를 컨테이너화하여 다양한 방식으로 배포 및 운영하는 것에 용이합니다.

### Docker

* Docker는 애플리케이션을 컨테이너화하여 여러 환경에서 실행할 수 있게 하는 이시식성이 뛰어난 기술입니다. 컨테이너는 소프트웨어를 패키지로 묶어 필요한 모든 것을 포함하는 독립적인 실행 단위로 만듭니다. Docker는 이러한 컨테이너화된 애플리케이션을 생성, 배포, 및 실행하기 위한 도구와 플랫폼을 제공합니다.
* 기술 사용 이점
  + 이미지를 사용하여 애플리케이션 및 라이브러리를 패키징합니다. 이미지는 버전 관리되며, 버전 간에 쉽게 전환할 수 있습니다. 이는 환경을 일관성 있게 유지하고 변경 사항을 관리하는 데 도움이 됩니다.
  + 애플리케이션을 작은 독립적인 서비스로 나누는 데 용이하며, 마이크로서비스 아키텍처를 구현하는 데 도움이 됩니다.

### Ingress

* Ingress는 Kubernetes에서 사용되는 API 오브젝트 중 하나로, 클러스터 내부의 서비스에 대한 외부 HTTP 및 HTTPS 라우팅 규칙을 정의합니다. Ingress를 사용하면 클러스터 외부에서 액세스되는 애플리케이션에 대한 URL 경로 기반 라우팅 및 로드 밸런싱을 설정할 수 있습니다.
* 기술 사용 이점
  + 경로 기반 라우팅을 통해 이용자 유형에따라 목적별로 분리한 각 웹 서버로 라우팅이 가능합니다 .
  + 다수의 서비스에 대하여 로드밸런싱이 가능해 각 pod의 부하를 줄일 수 있습니다.
  + Yaml 파일을 통해 설정을 편리하게 변경하고 관리 할 수 있으며 annotation을 활용해 다양한 기능을 이용할 수 있습니다.

### Horizontal Pod Autoscaler(HPA)

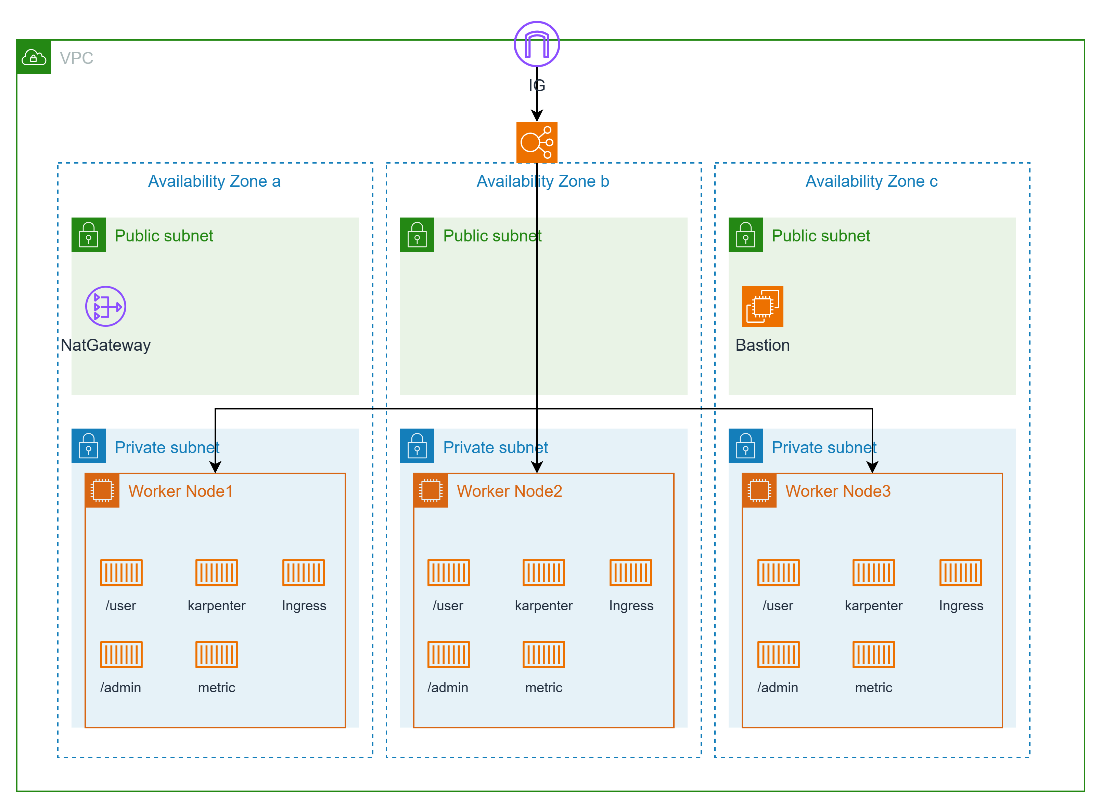
* HPA는 클러스터에서 실행 중인 파드의 수를 동적으로 조절하여 애플리케이션의 부하에 대응할 수 있게 하는 기능을 제공합니다. 이는 자동으로 확장하거나 축소하여 애플리케이션의 성능을 최적화하고 리소스 사용량을 효과적으로 관리할 수 있게 합니다.
* 기술 사용 이점
  + 지정된 지표를 기반으로 현재 파드의 리소스 사용량을 지속적으로 모니터링합니다. 이를 통해 애플리케이션의 부하 상태를 판단할 수 있습니다
  + 설정된 지표를 기반으로 HPA는 파드의 수를 동적으로 확장하거나 축소합니다. 임계치를 넘으면 파드 수를 증가시켜 추가 리소스를 할당하고, 사용률이 낮아지면 파드 수를 감소시켜 리소스 효율적으로 사용할 수 있습니다.
  + 최소 및 최대 파드 수, 타깃 지표 값을 조절할 수 있는 여러 가지 구성 옵션을 제공합니다

### Cluster AutoScaler(Karpenter)

* Karpenter는 Kubernetes에서 실행되는 프로비저닝 도구 중 하나입니다. 다양한 클라우드 환경에서 인프라스트럭처를 프로비저닝하고 관리합니다. 주로 Kubernetes 클러스터에서 필요한 노드(인스턴스)를 자동으로 생성하고 관리하는 데 사용됩니다.
* 기술 사용 이점
  + Karpenter는 클러스터에서 필요한 노드를 자동으로 프로비저닝합니다. 이를 통해 클러스터에 새로운 워크로드가 발생하거나 리소스가 부족할 때 자동으로 확장될 수 있습니다.
  + AWS AutoScaling Group에의해 동작하는 다른 CA서비스와 다르게 클러스터 내부에서 동작하기 때문에 훨씬 빠른속도로 장애에 대처 할 수 있습니다.

# 구성도

기존 온프레미스 서버가 하나로 이루어져 장애복구 및 서비스 기능저하 이슈가 있는점을 감안해 서비스를 분리하여 이미지화 하였습니다. EKS 클러스터가 구성 된 환경에 워커 노드 내부에 User/Admin Web Pod가 구성되어 웹 서버가 동작됩니다. Nodeport, Ingress를 통해 각 Pod에 경로 기반 라우팅이 됩니다. 각 Pod들은 HPA의 CPU와 Memory 임계치를 통해 수평적으로 확장되어 서비스를 안정적으로 유지할 수 있습니다. 클러스터의 경우 Karpenter를 사용해 AutoScaling 환경을 구축하였습니다. Node의 장애 상황이나 Node 리소스 부족으로 인한 pod의 pending 상황에 대하여 동적으로 확장합니다. Karpenter로 인해 기존 AutoScaling 보다 훨씬 빠르게 동작하여 장애상황에 빠르게 대처할 수 있습니다.



# 기술 구현

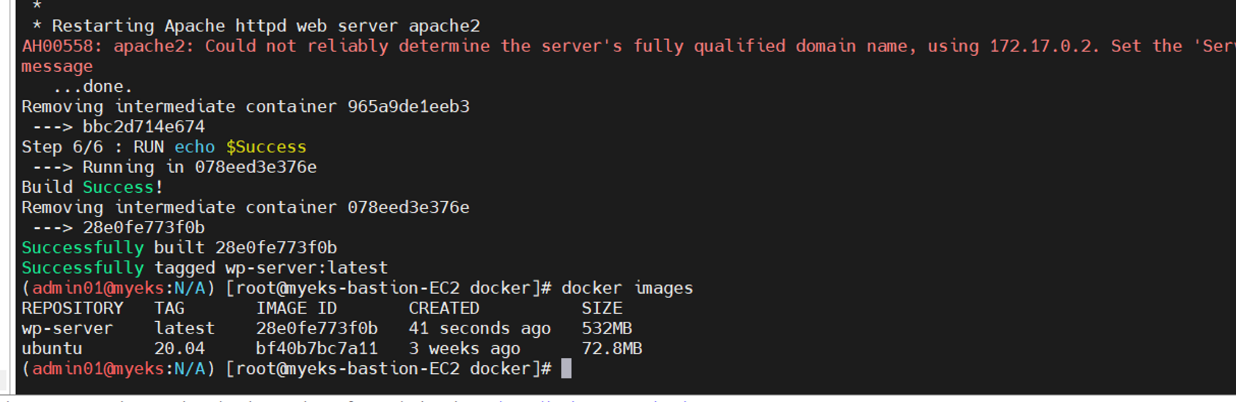
## Dockerfile구성 및 이미지 배포

### Docker 작업 폴더 생성 및 Dockerfile 작성

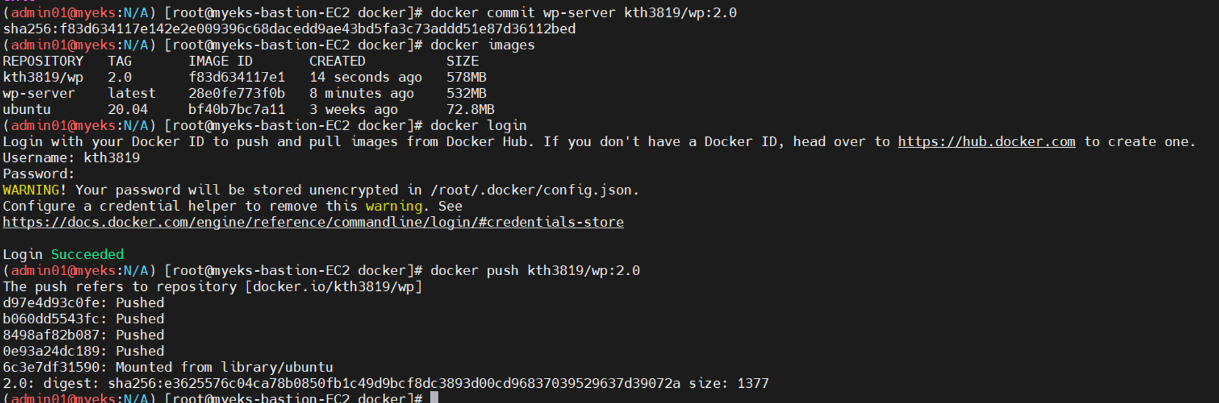
|  |
| --- |
| #Docker 작업 폴더 생성  mkdir docker  cd docker  #Dockerfile 작성  Vim Dockerfile  #Dockerfile 내용  FROM ubuntu:20.04    ENV Success "Build Success!"    RUN apt-get update && apt install -yq tzdata \  && ln -fs /usr/share/zoneinfo/America/New\_York /etc/localtime \  && dpkg-reconfigure -f noninteractive tzdata    # 패키지 설치  RUN apt-get install -y \  apache2 \  mariadb-client \  php7.4 libapache2-mod-php7.4 php7.4-mysql \  wget \  vim \  net-tools \  curl \  zip    COPY wwwroot.zip ./  #Wordpress 설치 및 권한 설정  RUN unzip wwwroot.zip \  && cp -r wwwroot/\* /var/www/html/ \  && chown -R www-data:www-data /var/www/html \  && find /var/www/html/ -type d -exec chmod 755 {} \; \  && find /var/www/html/ -type f -exec chmod 644 {} \; \  && rm /var/www/html/index.html \  && service apache2 restart  && echo $Success    CMD ["apachectl", "-D", "FOREGROUND"]    EXPOSE 80 |

|  |
| --- |
| #이미지 빌드  docker build . -t wp-server  #이미지 확인  docker images  #이미지 레포지토리에 업로드  docker push kth3819/wp |

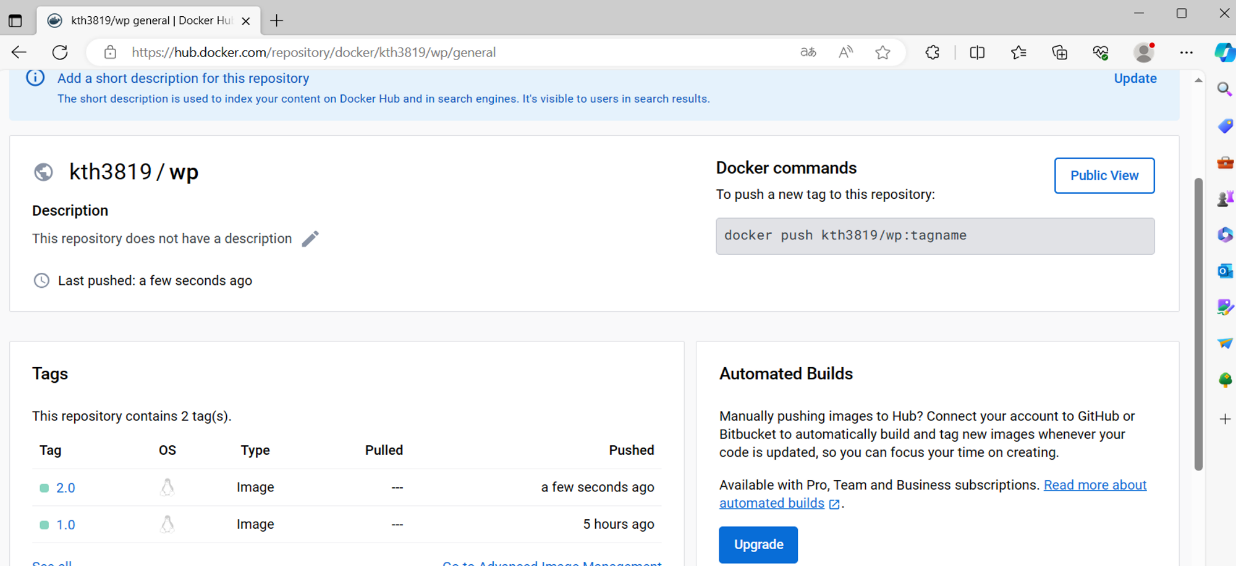
Docker image build



Docker image 레포지토리에 push



결과 확인



## EKS 웹 서버 구성 및 배포

### User, Admin 웹 서버 배포 yaml 작성

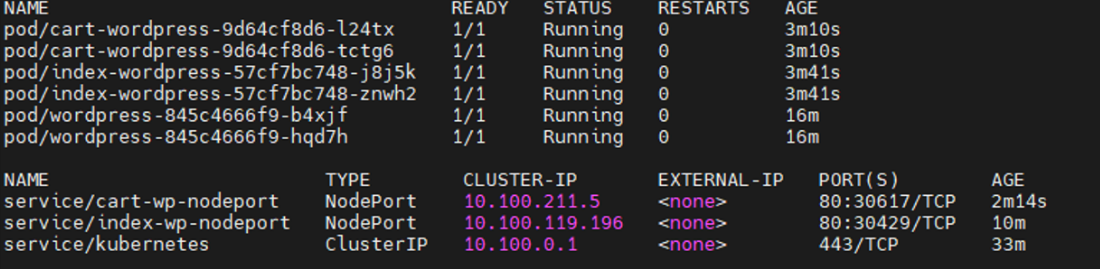
|  |
| --- |
| #user-deployment.yaml  apiVersion: apps/v1  kind: Deployment  metadata:  name: user-wordpress  labels:  app: user-wordpress  spec:  replicas: 1  selector:  matchLabels:  app: user-wordpress  tier: frontend  strategy:  type: RollingUpdate  rollingUpdate:  maxSurge: 1  maxUnavailable: 1  template:  metadata:  labels:  app: user-wordpress  tier: frontend  spec:  containers:  - image: kth3819/wp:3.2  name: user-wordpress  ports:  - containerPort: 80  name: user-wordpress  resources:  requests:  memory: "512Mi"  cpu: "500m"  limits:  memory: "512Mi"  cpu: "500m"  livenessProbe:  httpGet:  port: 80  path: /  initialDelaySeconds: 30  periodSeconds: 15  readinessProbe:  httpGet:  path: /  port: 80  initialDelaySeconds: 30  periodSeconds: 15  #admin-deployment.yaml  apiVersion: apps/v1  kind: Deployment  metadata:  name: admin-wordpress  labels:  app: admin-wordpress  spec:  replicas: 1  selector:  matchLabels:  app: admin-wordpress  tier: frontend  strategy:  type: RollingUpdate  rollingUpdate:  maxSurge: 1  maxUnavailable: 1  template:  metadata:  labels:  app: admin-wordpress  tier: frontend  spec:  containers:  - image: kth3819/wp:3.2  name: admin-wordpress  ports:  - containerPort: 80  name: admin-wordpress  resources:  requests:  memory: "512Mi"  cpu: "500m"  limits:  memory: "512Mi"  cpu: "500m"  livenessProbe:  httpGet:  port: 80  path: /wp-admin/  initialDelaySeconds: 30  periodSeconds: 15  readinessProbe:  httpGet:  path: /wp-admin/  port: 80  initialDelaySeconds: 30  periodSeconds: 15 |

### User,Admin 서버 외부 연결용 서비스 생성

|  |
| --- |
| #user-nodeport.yaml  apiVersion: v1  kind: Service  metadata:  name: user-nodeport  spec:  selector:  app: user-wordpress  ports:  - port: 80  targetPort: 80  protocol: TCP  type: NodePort    #admin-nodeport.yaml  apiVersion: v1  kind: Service  metadata:  name: admin-nodeport  spec:  selector:  app: admin-wordpress  ports:  - port: 80  targetPort: 80  protocol: TCP  type: NodePort |

### 웹 서버 deploymet, service 배포

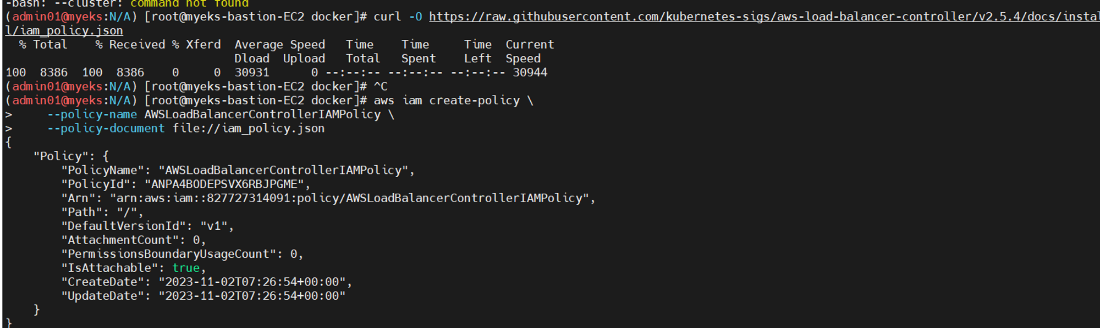
|  |
| --- |
| #user-deployment, service 배포  kubectl apply -f user-deployment.yaml  kubectl apply -f user-nodeport.yaml  #admin-deployment, service 배포  kubectl apply -f admin-deployment.yaml  kubectl apply -f admin-nodeport.yaml |



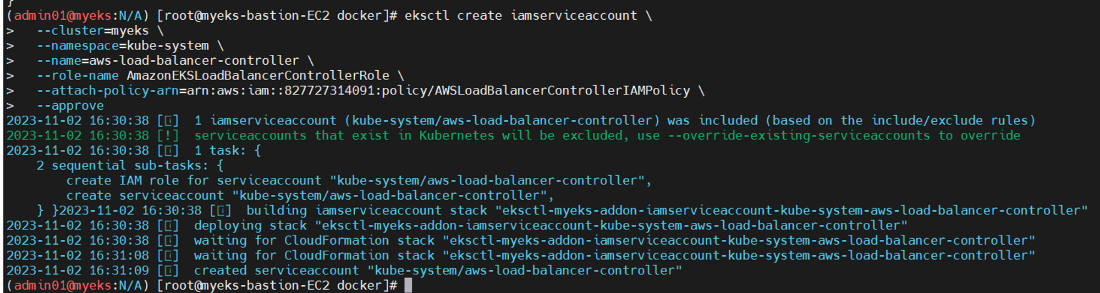
## 웹 서버 Ingress 연결 구성 및 배포

### Ingress Contorller 역할 생성 및 연결

|  |
| --- |
| #EKS 클러스터 이름 지정하고 진행    EKS\_CLUSTER\_NAME="myeks"  eksctl utils associate-iam-oidc-provider --cluster ${EKS\_CLUSTER\_NAME} --approve |



### ingress controller에 IRSA 설정



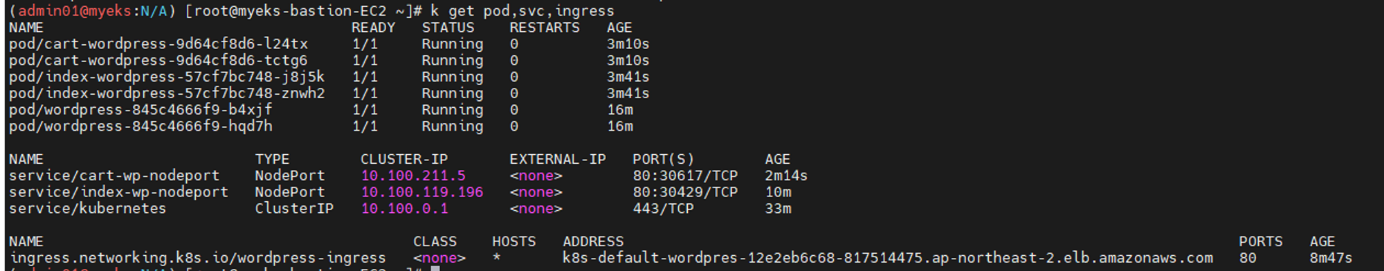
### Helm 차트로 aws-load-balancer-controller 설치

|  |
| --- |
| #helm repo 설정  helm repo add eks https://aws.github.io/eks-charts  helm repo update eks    #Load Balancer controller 설치  helm install aws-load-balancer-controller eks/aws-load-balancer-controller \  -n kube-system \  --set clusterName=myeks \  --set serviceAccount.create=false \  --set serviceAccount.name=aws-load-balancer-controller |

### User, Admin 서비스에 대한 Ingress.yaml 작성

|  |
| --- |
| #wordpress-ingresss.yaml  apiVersion: networking.k8s.io/v1  kind: Ingress  metadata:  name: wordpress-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: user-nodeport  port:  number: 80  - path: /wp-admin  pathType: Prefix  backend:  service:  name: admin-nodeport  port:  number: 80 |

### Ingress 배포 상태 확인



## Horizontal Pod Autoscaling(HPA) 설정 및 배포

### Metric-server 설치 및 확인

|  |
| --- |
| #hpa(metric-server) 배포  kubectl apply -f https://github.com/kubernetes-sigs/metrics-server/releases/latest/download/components.yaml |



### User-Server 에 대해 HPA.yaml 파일 작성

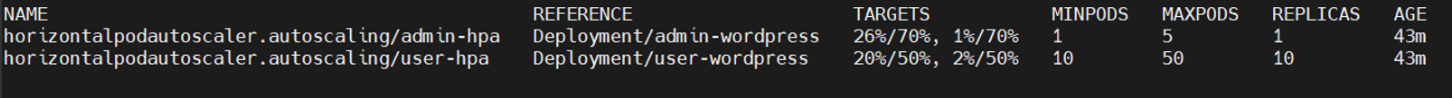
|  |
| --- |
| #hpa설정파일 user-hpa.yaml  apiVersion: autoscaling/v2  kind: HorizontalPodAutoscaler  metadata:  name: user-hpa  spec:  scaleTargetRef:  apiVersion: apps/v1  kind: Deployment  name: user-wordpress  minReplicas: 2  maxReplicas: 10  metrics:  - type: Resource  resource:  name: cpu  target:  type: Utilization  averageUtilization: 50  - type: Resource  resource:  name: memory  target:  type: Utilization  averageUtilization: 50 |

### Admin-Server에 대한 Hpa.yaml 작성

|  |
| --- |
| #hpa설정파일 admin-hpa.yaml  apiVersion: autoscaling/v2  kind: HorizontalPodAutoscaler  metadata:  name: admin-hpa  spec:  scaleTargetRef:  apiVersion: apps/v1  kind: Deployment  name: admin-wordpress  minReplicas: 1  maxReplicas: 5  metrics:  - type: Resource  resource:  name: cpu  target:  type: Utilization  averageUtilization: 70  - type: Resource  resource:  name: memory  target:  type: Utilization  averageUtilization: 70 |

### HPA 배포 및 상태 확인

|  |
| --- |
| #hpa 설정 배포  kubectl apply –f user-hpa.yml  kubectl apply –f admin-hpa.yaml  #hpa 배포 상태 확인  kubectl get hpa |

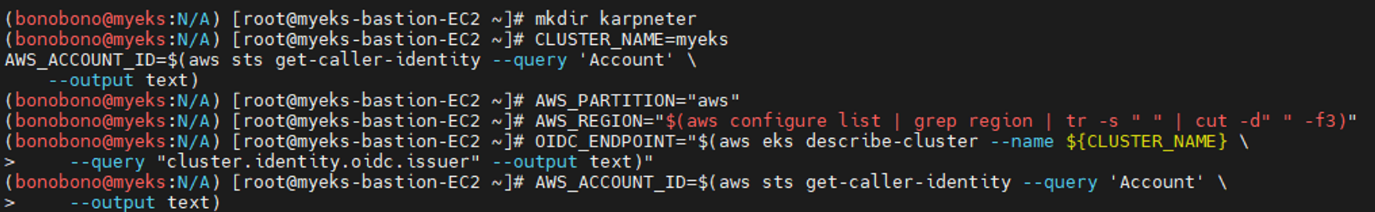


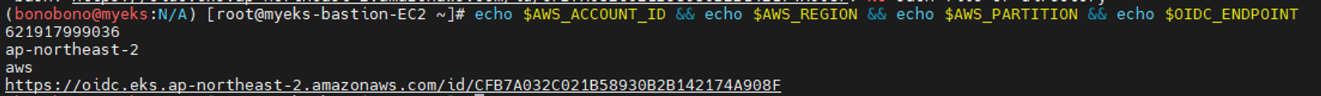
## Karpenter 설정 및 배포

### Karpenter 설치 전 필요한 환경 변수 지정

|  |
| --- |
| #환경변수 지정  CLUSTER\_NAME=myeks  AWS\_PARTITION="aws"  AWS\_REGION="$(aws configure list | grep region | tr -s " " | cut -d" " -f3)"  OIDC\_ENDPOINT="$(aws eks describe-cluster --name ${CLUSTER\_NAME} \  --query "cluster.identity.oidc.issuer" --output text)"  AWS\_ACCOUNT\_ID=$(aws sts get-caller-identity --query 'Account' \  --output text)  #환경변수 확인  echo $AWS\_PARTITION && echo $AWS\_REGION && echo $OIDC\_ENDPOINT && echo $AWS\_ACCOUNT\_ID |

환경 변수 지정 및 확인

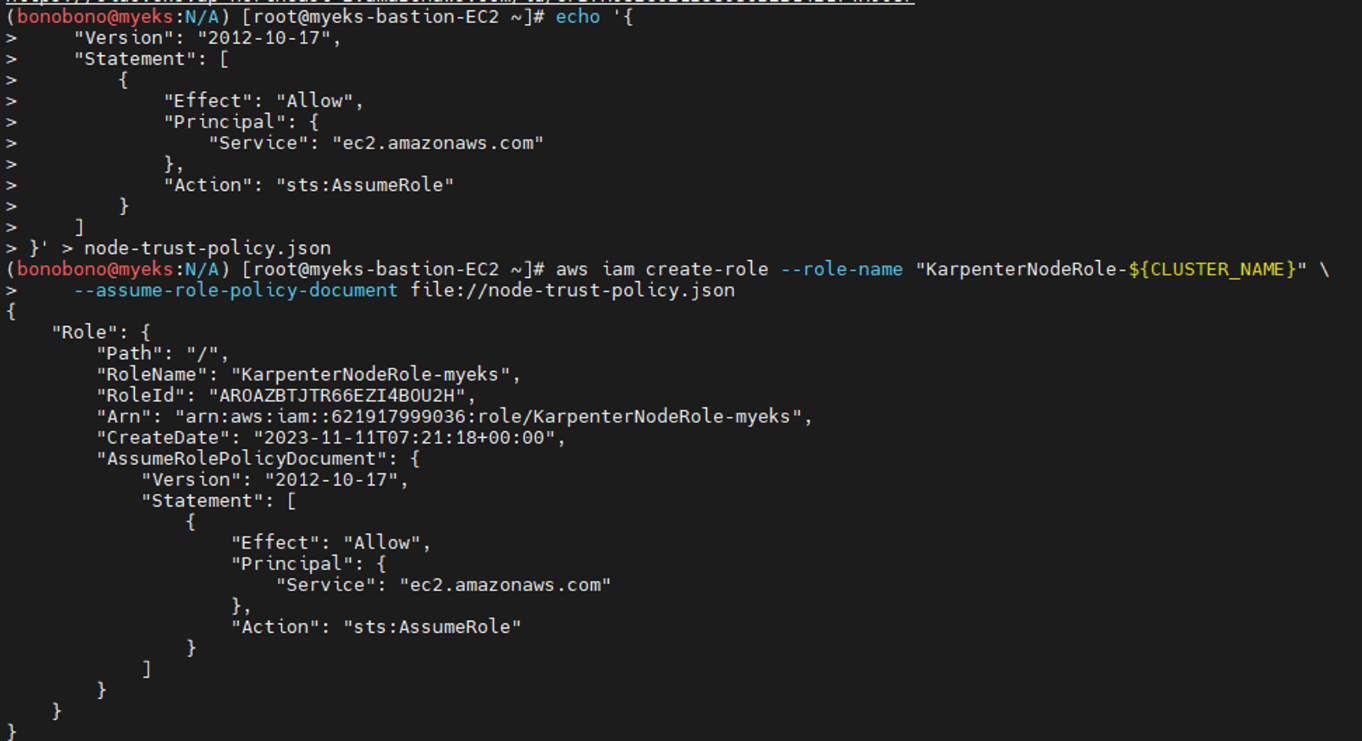




### IAM 역할,정책 생성 및 적용

|  |
| --- |
| #IAM 역할,정책 생성 및 적용  #node-trust-policy.json  echo '{  "Version": "2012-10-17",  "Statement": [  {  "Effect": "Allow",  "Principal": {  "Service": "ec2.amazonaws.com"  },  "Action": "sts:AssumeRole"  }  ]  }' > node-trust-policy.json  #역할 생성  aws iam create-role --role-name "KarpenterNodeRole-${CLUSTER\_NAME}" \  --assume-role-policy-document file://node-trust-policy.json  #역할에 정책 부여  aws iam attach-role-policy --role-name "KarpenterNodeRole-${CLUSTER\_NAME}" \  --policy-arn arn:${AWS\_PARTITION}:iam::aws:policy/AmazonEKSWorkerNodePolicy    aws iam attach-role-policy --role-name "KarpenterNodeRole-${CLUSTER\_NAME}" \  --policy-arn arn:${AWS\_PARTITION}:iam::aws:policy/AmazonEKS\_CNI\_Policy    aws iam attach-role-policy --role-name "KarpenterNodeRole-${CLUSTER\_NAME}" \  --policy-arn arn:${AWS\_PARTITION}:iam::aws:policy/AmazonEC2ContainerRegistryReadOnly    aws iam attach-role-policy --role-name "KarpenterNodeRole-${CLUSTER\_NAME}" \  --policy-arn arn:${AWS\_PARTITION}:iam::aws:policy/AmazonSSMManagedInstanceCore    #IAM Policy에 정책을 추가 완료한 후 EC2 Instance Profile 생성  aws iam create-instance-profile \  --instance-profile-name "KarpenterNodeInstanceProfile-${CLUSTER\_NAME}"    aws iam add-role-to-instance-profile \  --instance-profile-name "KarpenterNodeInstanceProfile-${CLUSTER\_NAME}" \  --role-name "KarpenterNodeRole-${CLUSTER\_NAME}"    #EKS controller가 사용할 정책,역할 생성  cat << EOF > controller-trust-policy.json  {  "Version": "2012-10-17",  "Statement": [  {  "Effect": "Allow",  "Principal": {  "Federated": "arn:${AWS\_PARTITION}:iam::${AWS\_ACCOUNT\_ID}:oidc-provider/${OIDC\_ENDPOINT#\*//}"  },  "Action": "sts:AssumeRoleWithWebIdentity",  "Condition": {  "StringEquals": {  "${OIDC\_ENDPOINT#\*//}:aud": "sts.amazonaws.com",  "${OIDC\_ENDPOINT#\*//}:sub": "system:serviceaccount:karpenter:karpenter"  }  }  }  ]  }  EOF    aws iam create-role --role-name KarpenterControllerRole-${CLUSTER\_NAME} \  --assume-role-policy-document file://controller-trust-policy.json    cat << EOF > controller-policy.json  {  "Statement": [  {  "Action": [  "ssm:GetParameter",  "ec2:DescribeImages",  "ec2:RunInstances",  "ec2:DescribeSubnets",  "ec2:DescribeSecurityGroups",  "ec2:DescribeLaunchTemplates",  "ec2:DescribeInstances",  "ec2:DescribeInstanceTypes",  "ec2:DescribeInstanceTypeOfferings",  "ec2:DescribeAvailabilityZones",  "ec2:DeleteLaunchTemplate",  "ec2:CreateTags",  "ec2:CreateLaunchTemplate",  "ec2:CreateFleet",  "ec2:DescribeSpotPriceHistory",  "pricing:GetProducts"  ],  "Effect": "Allow",  "Resource": "\*",  "Sid": "Karpenter"  },  {  "Action": "ec2:TerminateInstances",  "Condition": {  "StringLike": {  "ec2:ResourceTag/karpenter.sh/provisioner-name": "\*"  }  },  "Effect": "Allow",  "Resource": "\*",  "Sid": "ConditionalEC2Termination"  },  {  "Effect": "Allow",  "Action": "iam:PassRole",  "Resource": "arn:${AWS\_PARTITION}:iam::${AWS\_ACCOUNT\_ID}:role/KarpenterNodeRole-${CLUSTER\_NAME}",  "Sid": "PassNodeIAMRole"  },  {  "Effect": "Allow",  "Action": "eks:DescribeCluster",  "Resource": "arn:${AWS\_PARTITION}:eks:${AWS\_REGION}:${AWS\_ACCOUNT\_ID}:cluster/${CLUSTER\_NAME}",  "Sid": "EKSClusterEndpointLookup"  }  ],  "Version": "2012-10-17"  }  EOF    aws iam put-role-policy --role-name KarpenterControllerRole-${CLUSTER\_NAME} \  --policy-name KarpenterControllerPolicy-${CLUSTER\_NAME} \  --policy-document file://controller-policy.json |

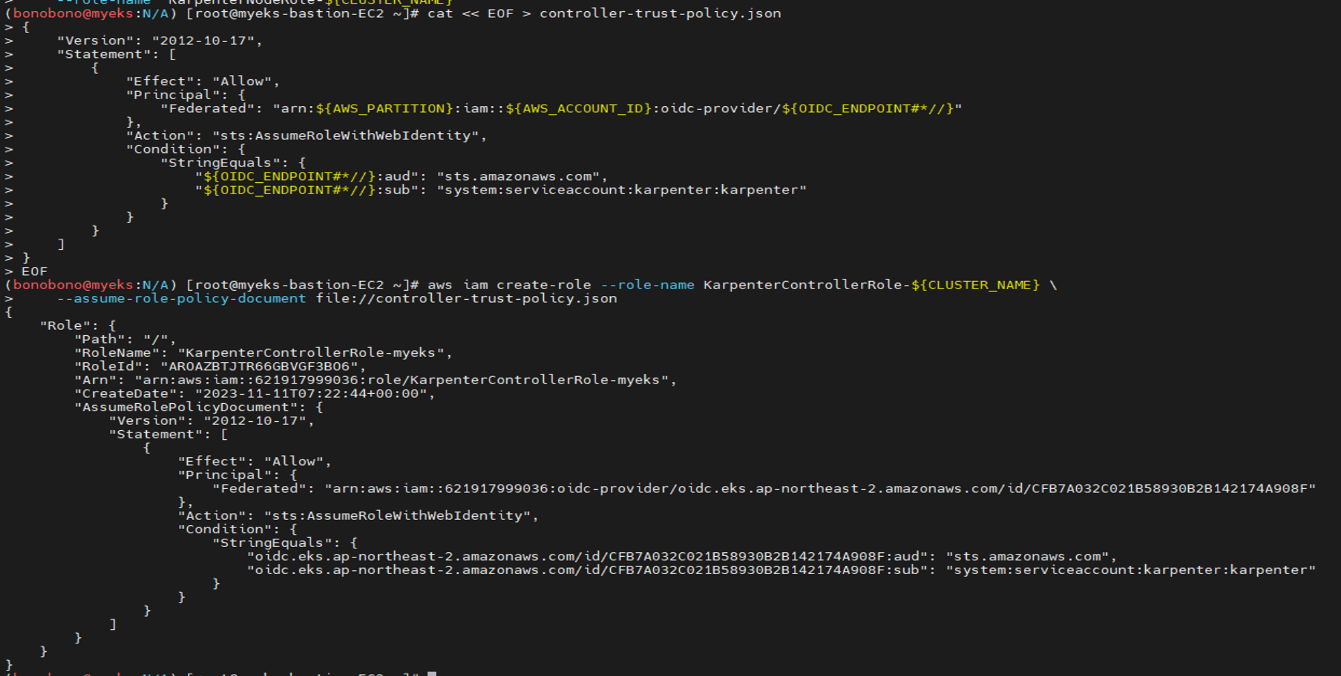
KarpenterNodeRole 역할 생성 및 정책 연

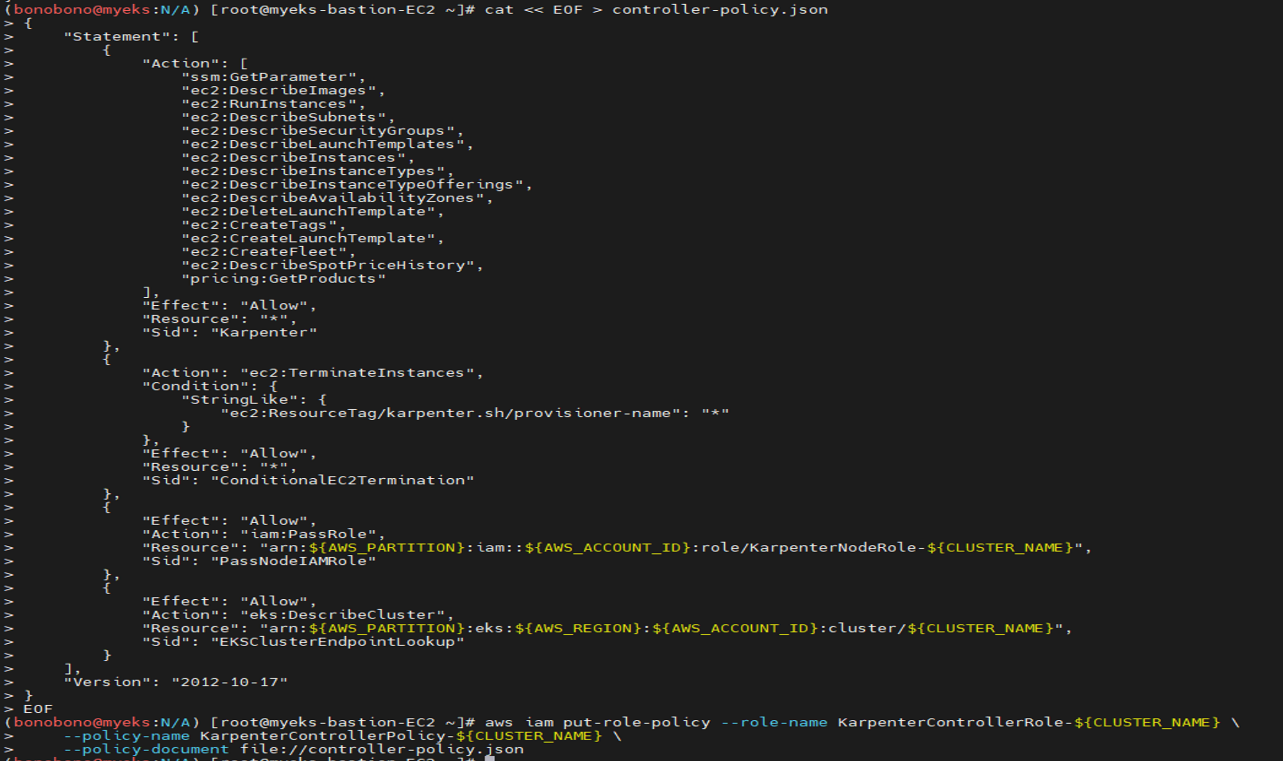


Iam instance-profile 생성



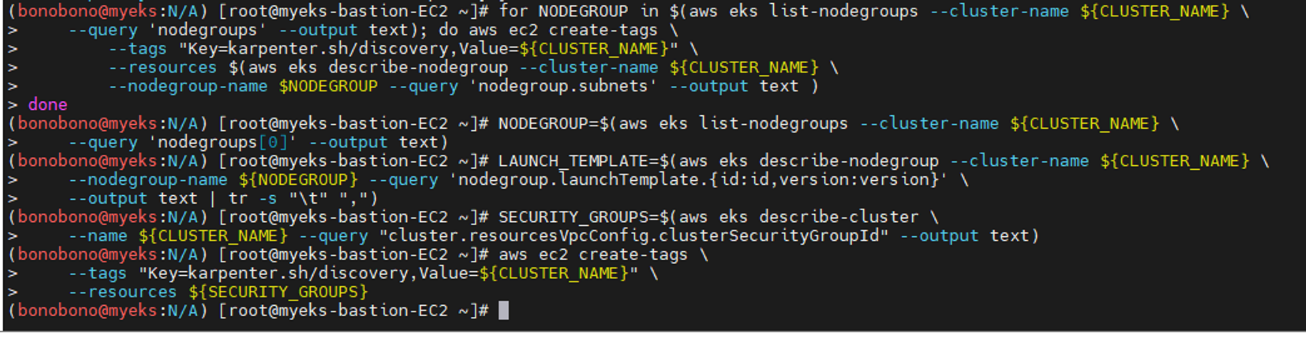
KarpenterControllerRole 생성 및 정책 연결





### Node Group 및 Security Group 연결

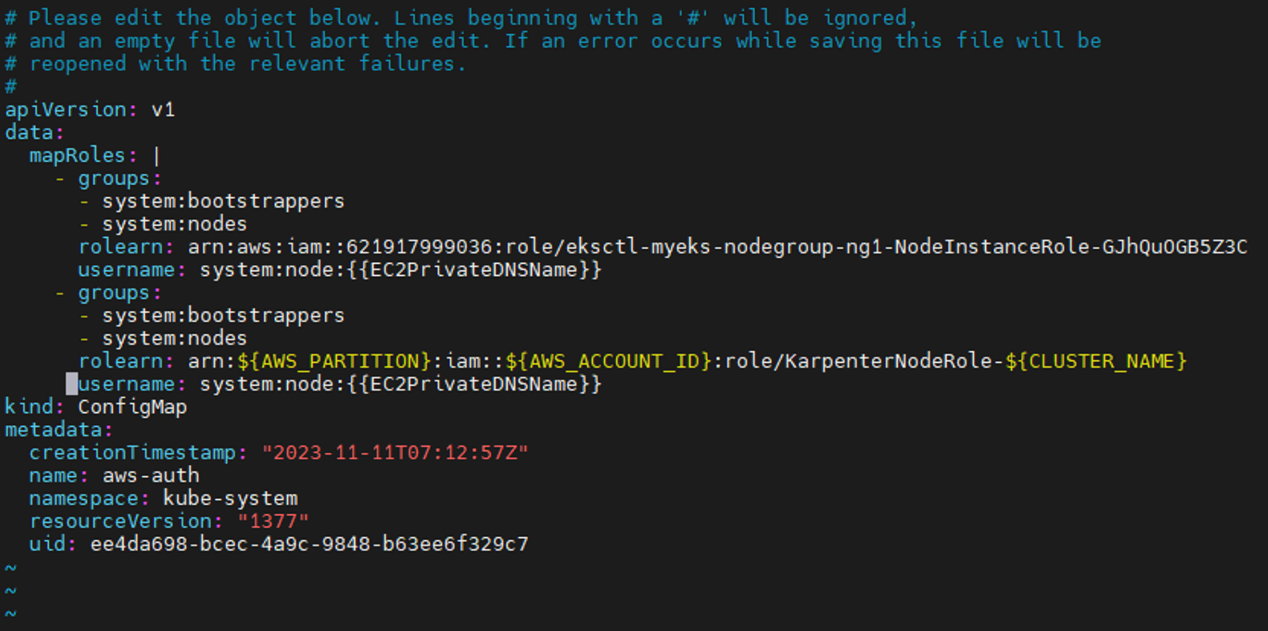
|  |
| --- |
| #VPC Subnet에서 Karpenter가 사용할 Subnet을 discovery할 수 있도록 태그 삽입  for NODEGROUP in $(aws eks list-nodegroups --cluster-name ${CLUSTER\_NAME} \  --query 'nodegroups' --output text); do aws ec2 create-tags \  --tags "Key=karpenter.sh/discovery,Value=${CLUSTER\_NAME}" \  --resources $(aws eks describe-nodegroup --cluster-name ${CLUSTER\_NAME} \  --nodegroup-name $NODEGROUP --query 'nodegroup.subnets' --output text )  done    #Security Group  NODEGROUP=$(aws eks list-nodegroups --cluster-name ${CLUSTER\_NAME} \  --query 'nodegroups[0]' --output text)    LAUNCH\_TEMPLATE=$(aws eks describe-nodegroup --cluster-name ${CLUSTER\_NAME} \  --nodegroup-name ${NODEGROUP} --query 'nodegroup.launchTemplate.{id:id,version:version}' \  --output text | tr -s "\t" ",")    SECURITY\_GROUPS=$(aws eks describe-cluster \  --name ${CLUSTER\_NAME} --query "cluster.resourcesVpcConfig.clusterSecurityGroupId" --output text)    #ec2 태그 생성  aws ec2 create-tags \  --tags "Key=karpenter.sh/discovery,Value=${CLUSTER\_NAME}" \  --resources ${SECURITY\_GROUPS} |



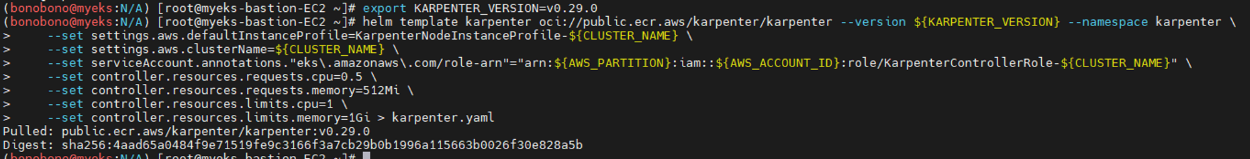
### Karpenter 설치 및 provisioner 배포

|  |
| --- |
| #kubectl cli를 이용하여 kube-system 네임스페이스에 있는 aws-auth를 수정  kubectl edit configmap aws-auth -n kube-system  아래 내용을 추가  - groups:  - system:bootstrappers  - system:nodes  rolearn: arn:aws:iam::621917999036:role/KarpenterNodeRole-myeks  username: system:node:{{EC2PrivateDNSName}}    #karpenter 버전 입력  export KARPENTER\_VERSION=v0.29.0  #helm으로 설치  helm template karpenter oci://public.ecr.aws/karpenter/karpenter --version ${KARPENTER\_VERSION} --namespace karpenter \  --set settings.aws.defaultInstanceProfile=KarpenterNodeInstanceProfile-${CLUSTER\_NAME} \  --set settings.aws.clusterName=${CLUSTER\_NAME} \  --set serviceAccount.annotations."eks\.amazonaws\.com/role-arn"="arn:${AWS\_PARTITION}:iam::${AWS\_ACCOUNT\_ID}:role/KarpenterControllerRole-${CLUSTER\_NAME}" \  --set controller.resources.requests.cpu=0.5 \  --set controller.resources.requests.memory=512Mi \  --set controller.resources.limits.cpu=1 \  --set controller.resources.limits.memory=1Gi > karpenter.yaml    #Karpenter.yaml에서 node affinity 부분에 아래 내용 추가  - matchExpressions:  - key: eks.amazonaws.com/nodegroup  operator: In  values:  - ${NODEGROUP}    #Karpenter 배포  kubectl create namespace karpenter  kubectl create -f \  https://raw.githubusercontent.com/aws/karpenter/${KARPENTER\_VERSION}/pkg/apis/crds/karpenter.sh\_provisioners.yaml  kubectl create -f \  https://raw.githubusercontent.com/aws/karpenter/${KARPENTER\_VERSION}/pkg/apis/crds/karpenter.k8s.aws\_awsnodetemplates.yaml  kubectl create -f \  https://raw.githubusercontent.com/aws/karpenter/${KARPENTER\_VERSION}/pkg/apis/crds/karpenter.sh\_machines.yaml  kubectl apply -f karpenter.yaml |

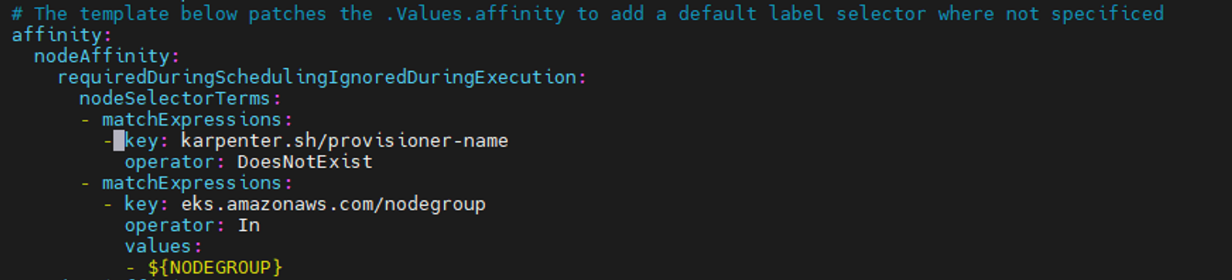
Configmap 수정 - Karpenter로 Node 관리 가능하도록



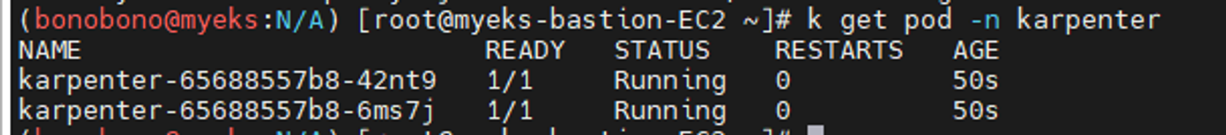
Helm으로 Karpenter 설치



Karpenter.yaml 수정

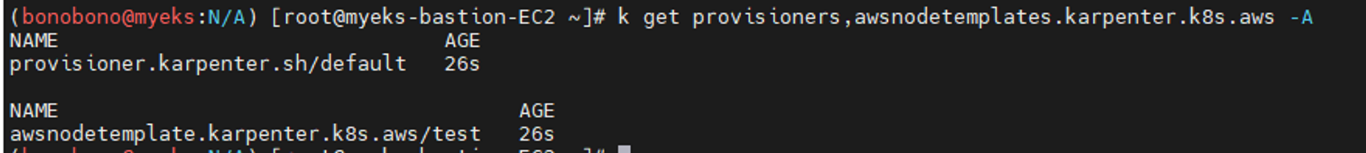


Karpenter pod 배포 상태 확인



|  |
| --- |
| cat <<EOF | kubectl apply -f -  apiVersion: karpenter.sh/v1alpha5  kind: Provisioner  metadata:  name: default  spec:  requirements:  - key: karpenter.k8s.aws/instance-category  operator: In  values: [c, m, r]  - key: karpenter.k8s.aws/instance-generation  operator: Gt  values: ["2"]  labels:  node\_role: test  providerRef:  name: default  ttlSecondsAfterEmpty: 30  ---  apiVersion: karpenter.k8s.aws/v1alpha1  kind: AWSNodeTemplate  metadata:  name: default  spec:  subnetSelector:  karpenter.sh/discovery: "${CLUSTER\_NAME}"  securityGroupSelector:  karpenter.sh/discovery: "${CLUSTER\_NAME}"  EOF |

Provsioners 및 awsnodetemplate 동작 상태 확인

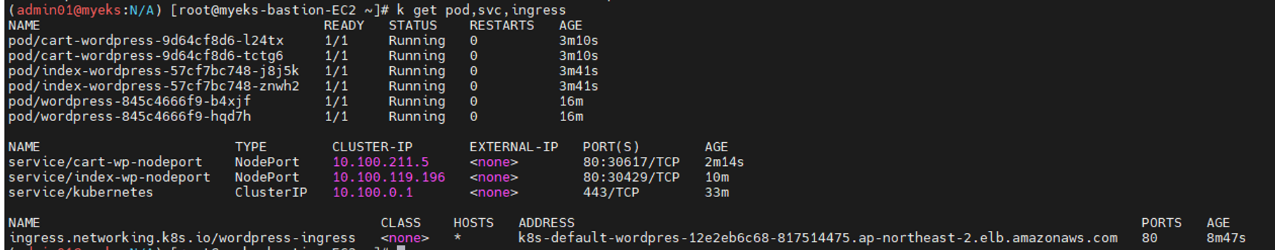


# 구현 테스트

## 웹 서버 배포 상태 및 Ingress 경로 테스트

### Deployment, SVC, Ingress 정상 동작 확인

ALB와 웹 서버의 정상 동작을 확인합니다



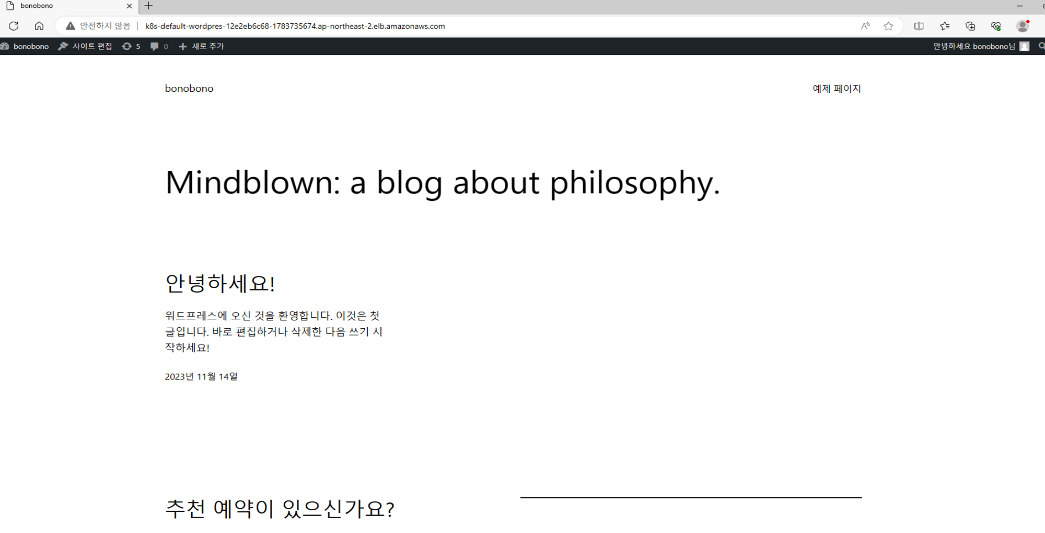
### ALB 주소에 접속하여 웹 서버 접근 테스트

브라우저에서 확인

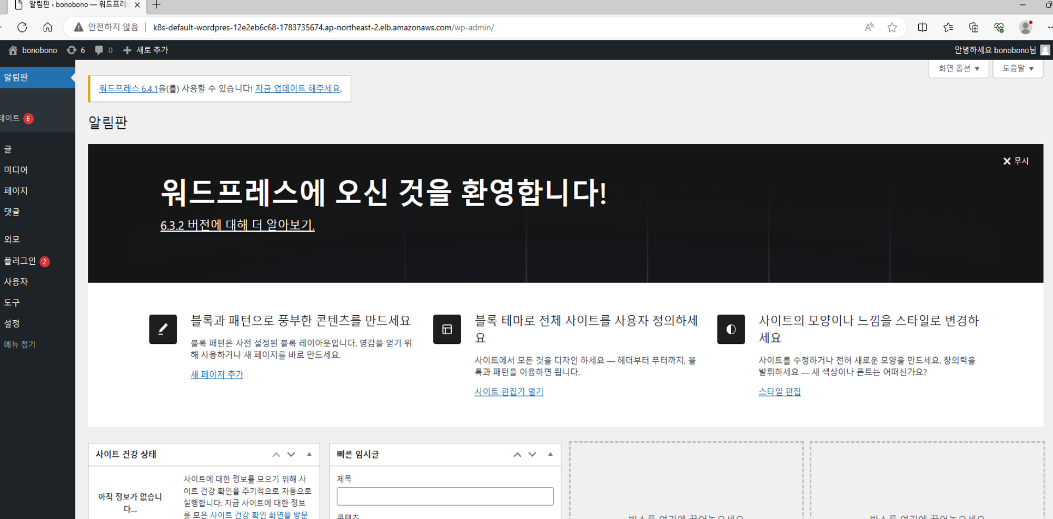


Ingress path에 따른 동작 확인

/user



/wp-admin



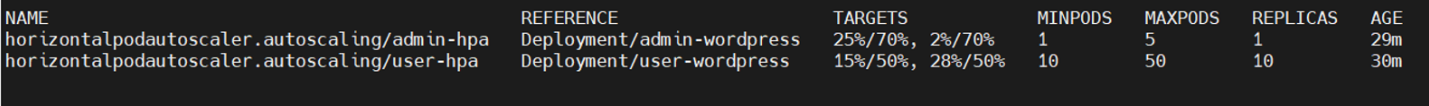
## HPA - Pod AutoScaling 테스트

### Metric Server 및 HPA 정상 동작 확인

Metric Server 동작 확인



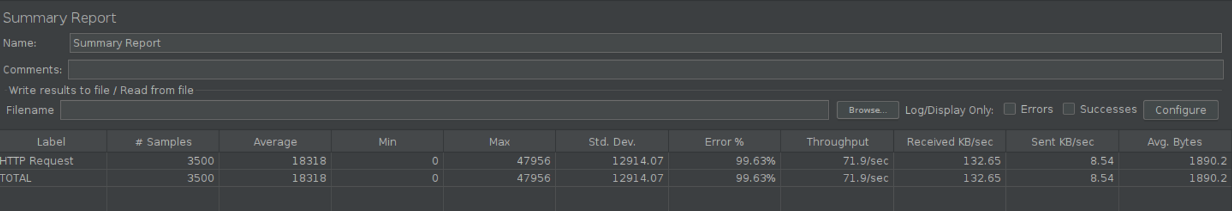
HPA 동작 확인

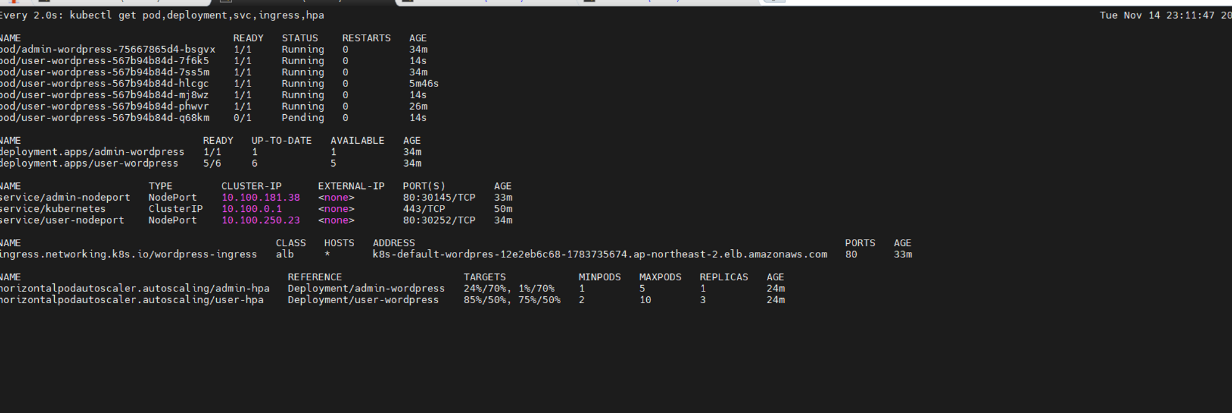


### Web-server의 Cpu, Memory 임계치 넘은 경우 Scale out 확인

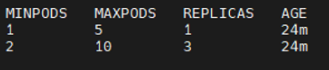
Jmeter를 통해 부하를 줘서 Pod의 임계치 초과



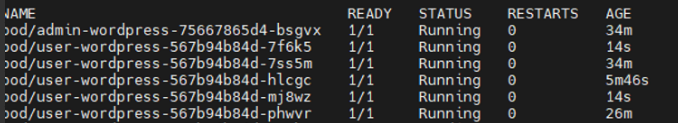




Min 값은 2지만 임계치를 초과해 Replica가 추가 됐다

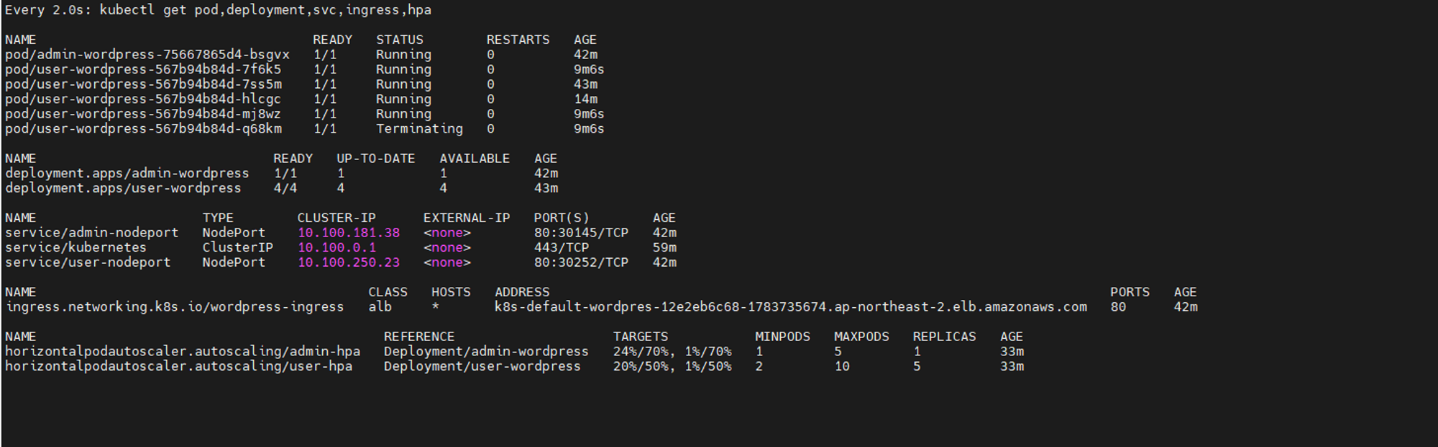


Pod 수 증가



### 임계치 밑으로 내려 갔을 경우 Scale in 확인

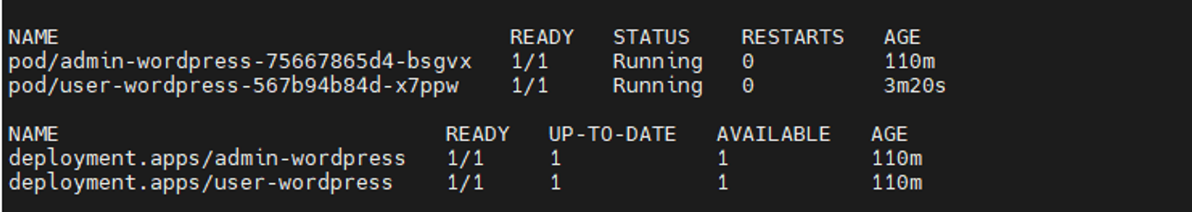
증가한 pod가 임계치 밑으로 내려가자 다시 줄어든 것을 확인

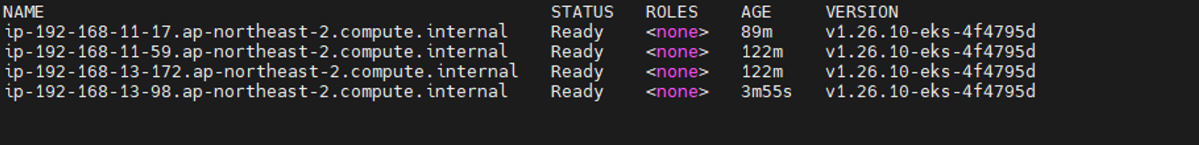


## Karpenter - Cluster AutoScaling 테스트

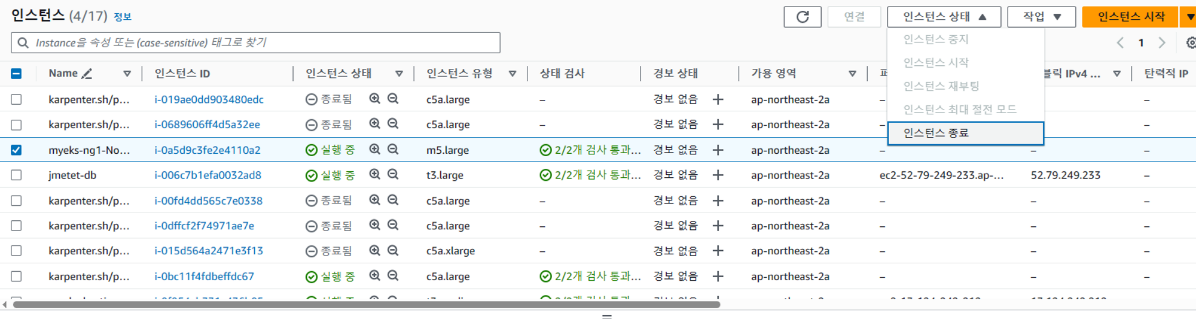
### Node 장애 발생 상황

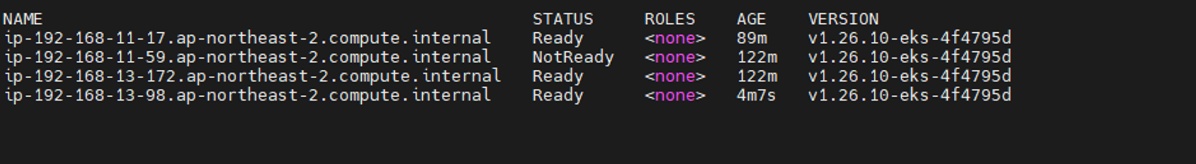
정상 상태



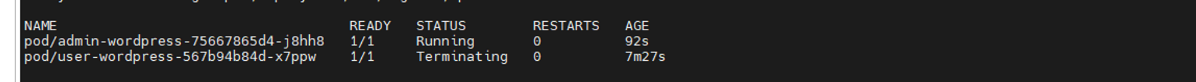


Node 삭제 시

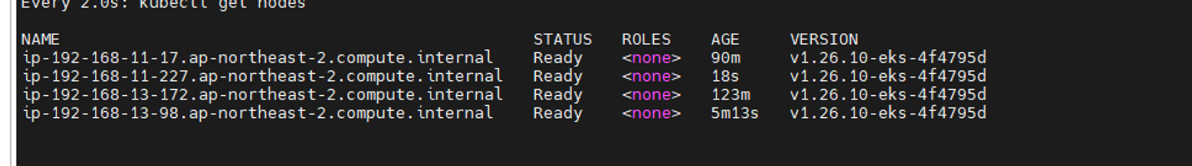




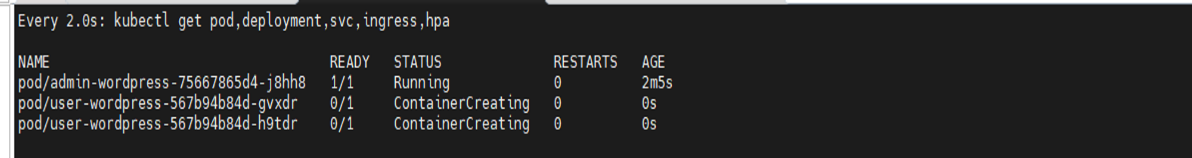
Pod도 종료 되는 상황



Node 재생성 약 45초

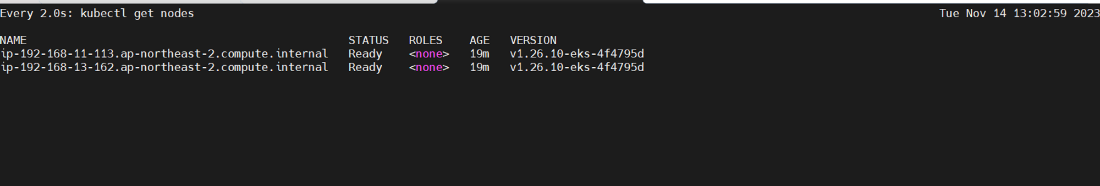


Pod 정상적으로 배포 및 동작

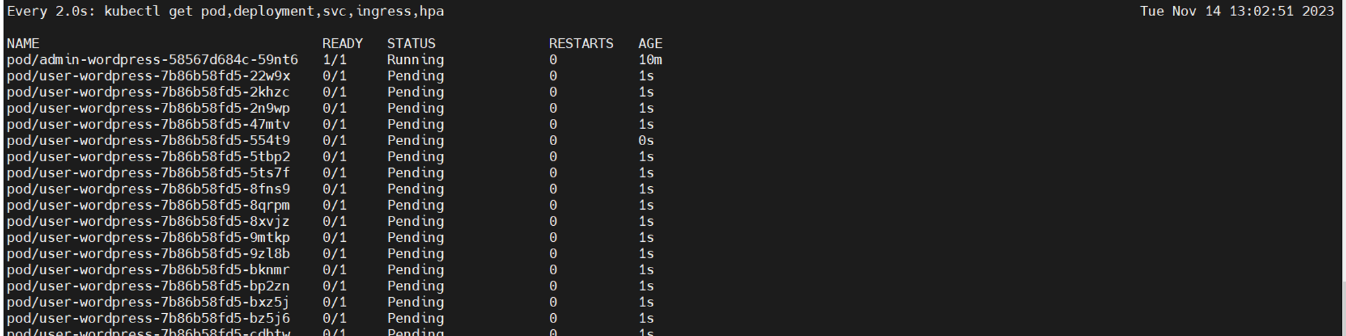


### Node 리소스 부족으로 Pod pending 상황

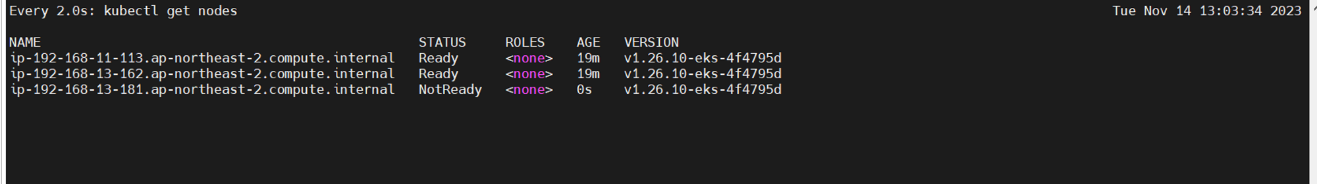
현재 Node 상태



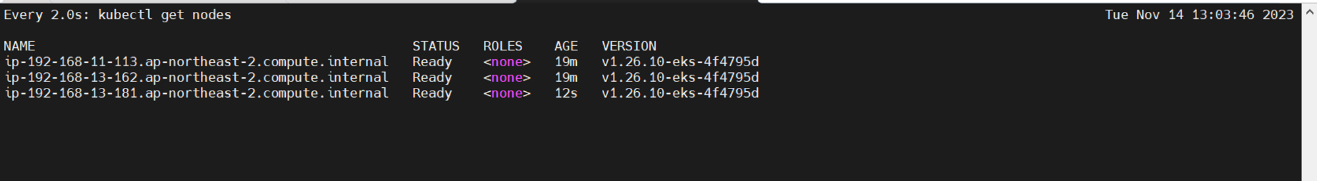
Pod 수 증가, Node 리소스 부족으로 Pending



35초 경과 새로운 노드 생성



45초 경과 Node Ready 상태



Pending 후 1분 뒤 모든 Pod 정상 배포

