**Azure CLI 설치**

[참고] <https://docs.microsoft.com/ko-kr/cli/azure/>

* **Azure CLI란?**

Azure CLI(명령줄 인터페이스)는 Azure에 연결하고 Azure 리소스에서 관리 명령을 실행하는 플랫폼 간 명령줄 도구입니다. 대화형 명령줄 프롬프트 또는 스크립트를 사용하여 터미널을 통해 명령을 실행하도록 허용합니다.

대화형 사용의 경우 먼저 Windows에서 cmd.exe와 같은 셸을 먼저 시작하거나 Linux 또는 macOS에서 Bash를 시작한 후 셸 프롬프트에서 명령을 실행합니다. 반복 작업을 자동화하려면 선택한 셸의 스크립트 구문을 사용하여 셸 스크립트로 CLI 명령을 어셈블한 후 스크립트를 실행합니다.

Linux, Mac 또는 Windows 컴퓨터에 로컬로 Azure CLI 설치할 수 있습니다. 또한 Azure Cloud Shell을 통해 브라우저에서 사용하거나 Docker 컨테이너 내부에서 실행할 수도 있습니다.

* **Linux에 Azure CLI 설치**

Azure CLI 팀은 모든 설치 명령을 한 번에 실행하는 스크립트를 관리합니다. 이 스크립트는 curl을 통해 다운로드하고 bash에 직접 파이프하여 CLI를 설치할 수 있습니다.

(명령어)

curl -sL https://aka.ms/InstallAzureCLIDeb | sudo bash

* **Azure CLI로 로그인 - 대화형으로 로그인**

로그인을 위한 Azure CLI의 기본 인증 방법은 웹 브라우저와 액세스 토큰을 사용하여 로그인합니다.

1. login 명령을 실행합니다.

(명령어)

az login

CLI는 기본 브라우저를 열 수 있으면 기본 브라우저를 열고 Azure 로그인 페이지를 로드합니다. 그렇지 않으면 <https://aka.ms/devicelogin>에서 브라우저 페이지를 열고 터미널에 표시된 권한 부여 코드를 입력합니다. 웹 브라우저를 사용할 수 없거나 웹 브라우저가 열리지 않으면 **az login --use-device-code** 와 함께 디바이스 코드 흐름을 사용합니다.

1. 브라우저에서 계정 자격 증명으로 로그인합니다.
2. [옵션] 로그인 상태를 점검하기 위하여 간단한 명령어를 실행해 봅니다.

(명령어)

az group list

**AKS를 사용하여 Kubernetes Cluster 만들기**

[참고] <https://docs.microsoft.com/ko-kr/azure/aks/>

[참고] <https://docs.microsoft.com/ko-kr/azure/aks/kubernetes-walkthrough-portal>

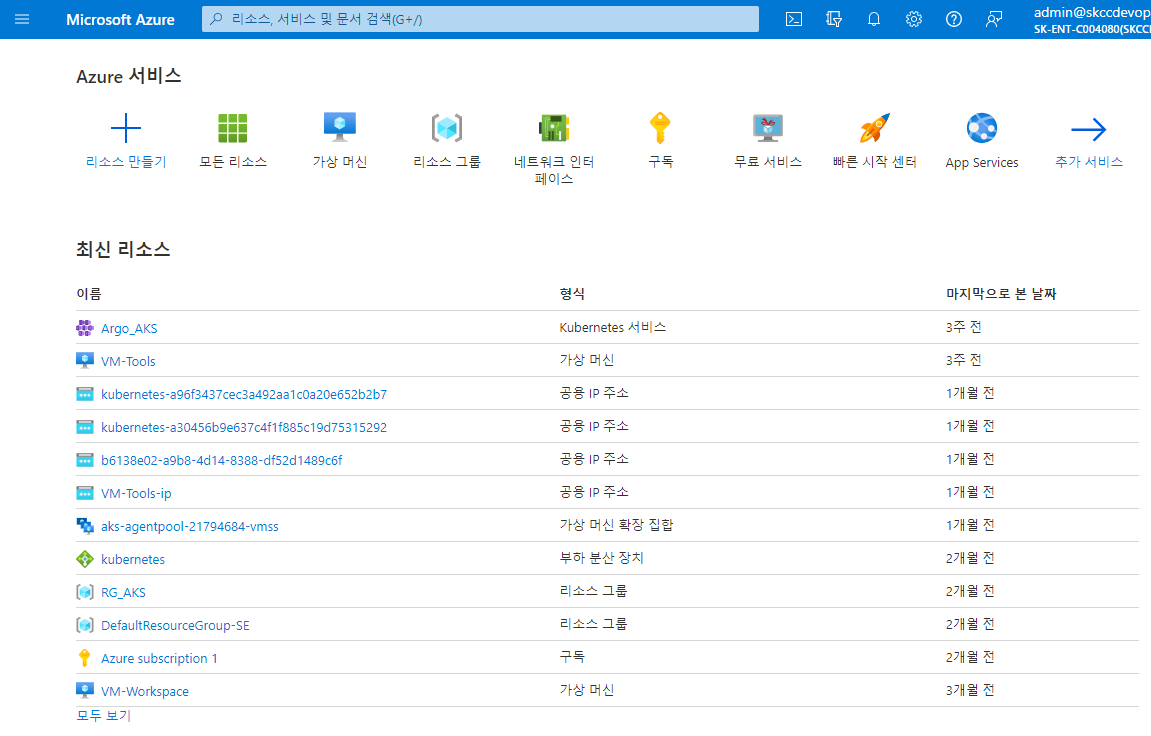
* **AKS란?**

AKS(Azure Kubernetes Service)는 운영 오버헤드를 Azure로 오프로드하여 Azure에서 관리되는 Kubernetes 클러스터 배포를 단순화합니다. 호스팅되는 Kubernetes 서비스인 Azure는 상태 모니터링 및 유지 관리 같은 중요 작업을 처리합니다. Kubernetes 마스터는 Azure에서 관리되므로 에이전트 노드만 관리하고 유지 관리합니다. 따라서 AKS는 무료입니다. 마스터가 아니라 클러스터 내의 에이전트 노드에 대해서만 지불합니다.

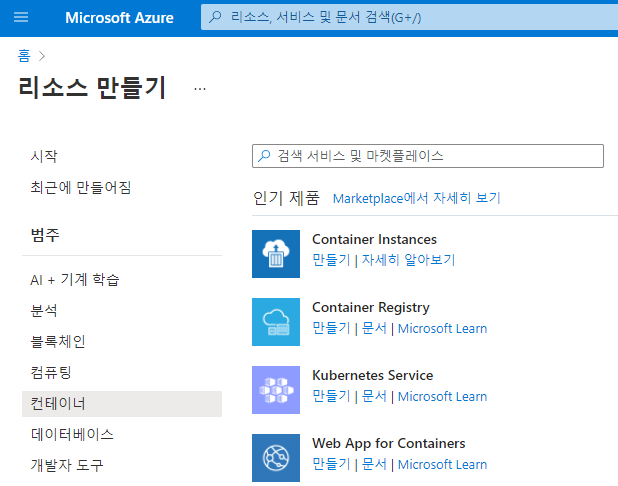
AKS(Azure Kubernetes Service)는 클러스터를 빠르게 배포하고 관리할 수 있는 관리형 Kubernetes 서비스입니다.

* **Azure Portal을 사용하여 AKS(Azure Kubernetes Service) 클러스터 배포**
* AKS 클러스터 만들기

1. Azure Portal 메뉴 또는 홈 페이지에서 리소스 만들기 를 선택합니다.



1. 컨테이너 > Kubernetes Service > 만들기를 선택합니다.



1. **Basics** 페이지에서 다음 옵션을 구성합니다.

* Project details > Subscription : 구독 선택(Azure subscription 1)
* Project details > Subscription > Resource group : 선택하거나 생성(RG\_AKS)
* 클러스터 세부 정보 > 사전 설정 구성 : ‘표준($$)’인지 확인
* 클러스터 세부 정보 > Kubernetes 클러스터 이름 : 이름 입력(Argo\_AKS)
* 클러스터 세부 정보 > 지역 : 한국 중부

[참고] 동일한 설정 내용에서도 계정에 따라, 지역에 따라

유효성 검사 단계 또는 실제 생성 시에 에러가 발생하기도 함

* 클러스터 세부 정보 > 가용성 영역 : 영역 1,2,3
* 클러스터 세부 정보 > kubernetes 버전 : 1.20.9(기본 값)
* 주 노드 풀 : 선택된 기본값 유지



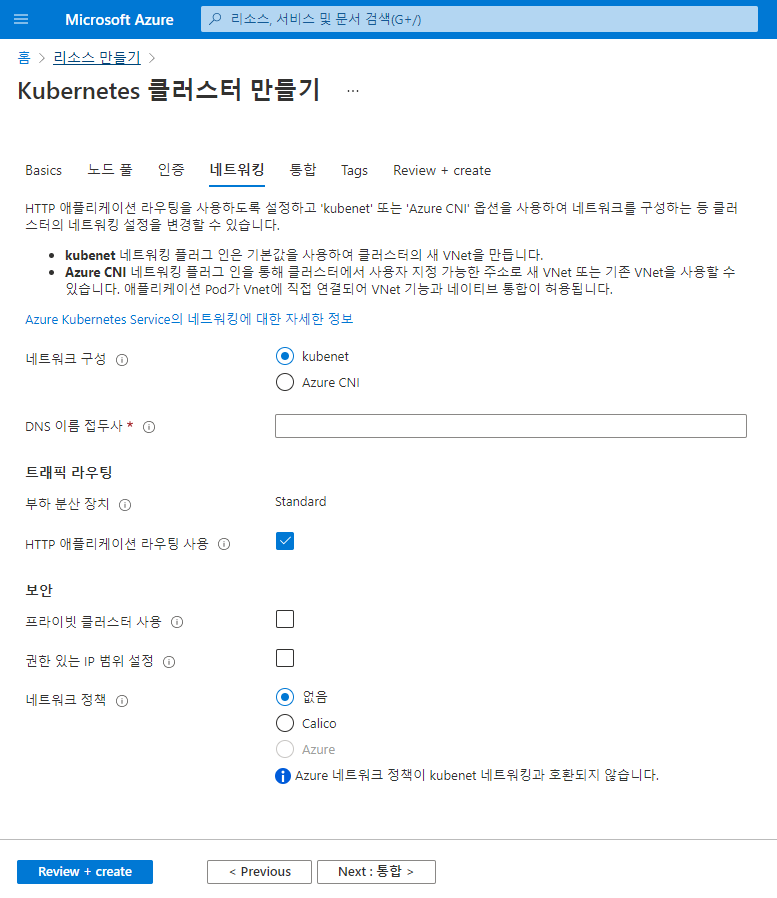
텍스트이(가) 표시된 사진

자동 생성된 설명

1. **노드 풀, 인증, 통합, Tags** 페이지의 기본값을 유지합니다.
2. **네트워킹** 페이지에서 다음 옵션을 구성합니다(AKS를 외부에 Open하기 위하여 설정함).

* HTTP 애플리케이션 라우팅 사용 : Check

1. “Review + create” 를 클릭한 후 유효성 검사가 완료되면 만들기 를 선택합니다.  
   AKS 클러스터를 만드는 데 몇 분이 걸립니다.



* AKS 클러스터에 연결

1. [az aks get-credentials](https://docs.microsoft.com/ko-kr/cli/azure/aks#az_aks_get_credentials) 명령을 사용하여 Kubernetes 클러스터에 연결하도록 kubectl을 구성합니다. 다음 명령은 자격 증명을 다운로드하고 해당 자격 증명을 사용하도록 Kubernetes CLI를 구성합니다.

(명령어]

az aks get-credentials --resource-group RG\_AKS --name Argo\_AKS

(실행결과]

Merged "Argo\_AKS" as current context in /home/azureuser/.kube/config

1. kubectl get을 사용해 클러스터에 대한 연결을 확인하여 클러스터 노드 목록을 반환합니다.

(명령어)

kubectl get nodes

(실행결과]

NAME STATUS ROLES AGE VERSION

aks-agentpool-21794684-vmss000000 Ready agent 2m47s v1.20.9

aks-agentpool-21794684-vmss000001 Ready agent 2m50s v1.20.9

aks-agentpool-21794684-vmss000002 Ready agent 2m47s v1.20.9

출력은 이전 단계에서 만든 단일 노드를 보여줍니다.

노드 상태가 준비 인지 확인합니다.

* **[참고] 클러스터 삭제**

(명령어)

az aks delete --resource-group RG\_AKS --name Argo\_AKS --no-wait

**Jenkins Pipeline을 Multiple Agents 방식으로 구성하기**

[참고] <https://www.jenkins.io/doc/book/pipeline/jenkinsfile/>

* **목적**

AKS(Azure Kubernetes Service) Cluster에서 /var/run/docker.sock 사용이 차단되어 있습니다. Cluster에서 동적으로 구현하는 Docker 기능을 사용 할 수 없어 관련 부분은 정적으로 구현하여야 합니다. Pipeline의 Stage 별로 별도의 Agents를 정의하여 구현합니다.

[참고] <https://docs.microsoft.com/ko-kr/azure/aks/cluster-configuration>

텍스트이(가) 표시된 사진

자동 생성된 설명

* **구현**
* 기존 Single Agent 방식의 Jenkins Pipeline Code

pipeline 범위의 agent를 정의하여 사용한다.

pipeline {

agent {

kubernetes {

label 'petclinic-cd'

yamlFile 'jenkins-agent-pod.yaml'

}

}

이하 생략

* Multiple Agents 방식의 Jenkins Pipeline Code

stage 범위의 agent를 각각 정의하여 사용한다.

pipeline {

agent none

생략

stages {

stage('Build') {

agent {

kubernetes {

label 'petclinic-cd'

yamlFile 'jenkins-agent-pod.yaml'

}

}

steps {

container('maven') {

sh 'mvn clean compile'

}

}

}

이하 생략

* [참고] Multiple Agents 방식의 Jenkins Pipeline Code - Full Source Code

pipeline {

agent none

environment {

REGISTRY\_URL = 'https://kktdevops.koreacentral.cloudapp.azure.com'

REGISTRY\_CREDENTIALS = 'credentials\_harbor'

APP\_IMAGE = null

IMAGE\_REPO = 'repo-spring-petclinic-rest'

IMAGE\_NAME = 'spring-petclinic-rest'

IMAGE\_TAG = "test\_${BUILD\_NUMBER}"

APP\_URL='http://azure-helm-petclinic.koreacentral.cloudapp.azure.com/petclinic'

APP\_PORT=80

PerfURL='azure-helm-petclinic.koreacentral.cloudapp.azure.com'

ArgoURL='20.200.241.185'

argocdAppPrefix='azure-helm-petclinic'

appWaitTimeout = 60

}

stages {

stage('Build') {

agent {

kubernetes {

label 'petclinic-cd'

yamlFile 'jenkins-agent-pod.yaml'

}

}

steps {

container('maven') {

sh 'mvn clean compile'

}

}

}

stage('Unit Test') {

agent {

kubernetes {

label 'petclinic-cd'

yamlFile 'jenkins-agent-pod.yaml'

}

}

steps {

container('maven') {

sh 'mvn test'

}

}

post {

always {

junit 'target/surefire-reports/\*.xml'

step([ $class: 'JacocoPublisher' ])

}

}

}

stage('Static Code Analysis') {

agent any

steps {

configFileProvider([configFile(fileId: 'maven-settings', variable: 'MAVEN\_SETTINGS')]) {

sh './mvnw sonar:sonar -s $MAVEN\_SETTINGS'

}

}

}

/\*

stage('Static Code Analysis') {

agent {

kubernetes {

label 'petclinic-cd'

yamlFile 'jenkins-agent-pod.yaml'

}

}

steps {

configFileProvider([configFile(fileId: 'maven-settings', variable: 'MAVEN\_SETTINGS')]) {

container('maven') {

sh 'mvn sonar:sonar -s $MAVEN\_SETTINGS'

}

}

}

}

\*/

stage('Package') {

agent {

kubernetes {

label 'petclinic-cd'

yamlFile 'jenkins-agent-pod.yaml'

}

}

steps {

container('maven') {

sh 'mvn clean package -DskipTests'

archiveArtifacts artifacts: 'target/\*.jar', fingerprint: true

}

}

}

stage('Build Docker image') {

agent any

steps {

script {

APP\_IMAGE = docker.build("${IMAGE\_REPO}/${IMAGE\_NAME}:${IMAGE\_TAG}")

}

}

}

stage('Push Docker image') {

agent any

steps {

script {

docker.withRegistry(REGISTRY\_URL, REGISTRY\_CREDENTIALS) {

APP\_IMAGE.push()

APP\_IMAGE.push('latest')

}

}

}

}

stage('Update manifest') {

agent any

steps {

sh """

git config --global user.name 'skccdevops03'

git config --global user.email 'skcc.devops03@sk.com'

git config --global credential.helper cache

git config --global push.default simple

"""

git url: 'https://github.com/skccdevops03/azure-helm-petclinic.git', credentialsId: 'credentials\_git', branch: 'main'

sh """

sed -i 's/tag:.\*/tag: "${IMAGE\_TAG}"/g' values.yaml

git add values.yaml

git commit -m 'Update Docker image tag: ${IMAGE\_TAG}'

git push origin main

"""

}

}

stage('Argo'){

agent {

kubernetes {

label 'petclinic-cd'

yamlFile 'jenkins-agent-pod.yaml'

}

}

steps {

withCredentials([usernamePassword(credentialsId: 'credentials\_argo', usernameVariable: 'ARGOCD\_USER', passwordVariable: 'ARGOCD\_PWD')]) {

container('argocd') {

sh """

yes | argocd login --insecure ${ArgoURL} --username ${ARGOCD\_USER} --password ${ARGOCD\_PWD}

argocd app sync ${argocdAppPrefix}

argocd app wait ${argocdAppPrefix} --timeout ${appWaitTimeout}

argocd logout ${ArgoURL}

sleep 10

"""

}

}

}

}

stage('PerforfmanceTest') {

agent {

kubernetes {

label 'petclinic-cd'

yamlFile 'jenkins-agent-pod.yaml'

}

}

steps {

withCredentials([usernamePassword(credentialsId: 'credentials\_git', usernameVariable: 'GIT\_USER', passwordVariable: 'GIT\_PWD')]) {

container('jmeter') {

sh """

git checkout origin/master

git config remote.origin.url https://${GIT\_USER}:${GIT\_PWD}@github.com/skccdevops03/spring-petclinic-rest.git

git config --global user.email 'skcc.devops03@sk.com'

git config --global user.name '${GIT\_USER}'

"""

sh """

JVM\_ARGS="-Xms1G -Xmx1G" && export JVM\_ARGS && /usr/local/jmeter/apache-jmeter-5.4.1/bin/jmeter.sh \

-n -f -t PerformanceTest/TS01\_TC01\_AWSPipeline.jmx -Jurl=${PerfURL} \

-l PerformanceTest/TestResult/Result\_${BUILD\_NUMBER}.jtl \

-e -o PerformanceTest/TestResult/Result\_html\_${BUILD\_NUMBER}

mv jmeter.log PerformanceTest/TestResult/Result\_html\_${BUILD\_NUMBER}/jmeter.log

"""

sh """

git add . && git commit -am 'Publish Jmeter result' && git push origin HEAD:master

"""

}

}

}

post {

always {

perfReport 'PerformanceTest/TestResult/Result\_${BUILD\_NUMBER}.jtl'

}

}

}

stage('API Test') {

agent {

kubernetes {

label 'petclinic-cd'

yamlFile 'jenkins-agent-pod.yaml'

}

}

steps {

container('newman') {

sh """

newman run api\_test.json \

--env-var 'baseUrl=${APP\_URL}' --env-var 'petTypeId=""'\

--reporters cli,junit \

--reporter-junit-export 'petclinic-report.xml'

"""

}

}

post {

always {

junit 'petclinic-report.xml'

}

}

}

} /\* stages \*/

} /\* pipeline \*/