

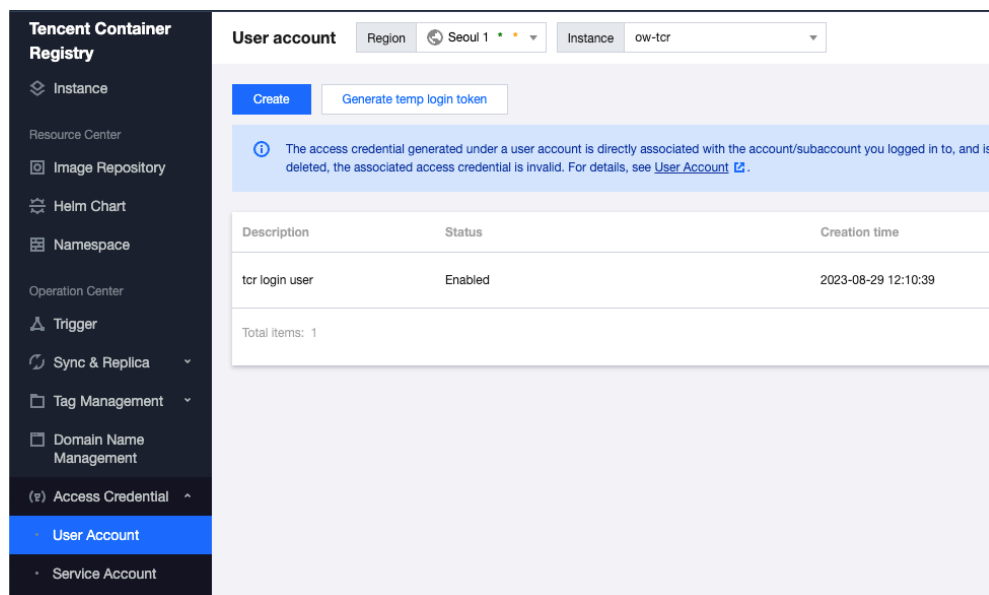


OW CI/CD

☀ 상태	작성 완료
📁 종류	개발
📁 프로젝트	OW
👥 참조	Ⓔ Emile Ⓕ Freddie
👤 작성자	👤 Pete
📅 작성일	@2023년 8월 30일

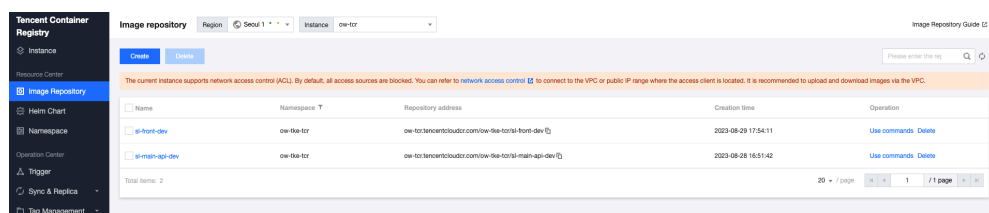
1. TCR 세팅

- Access Credential 탭에서 User Account 생성



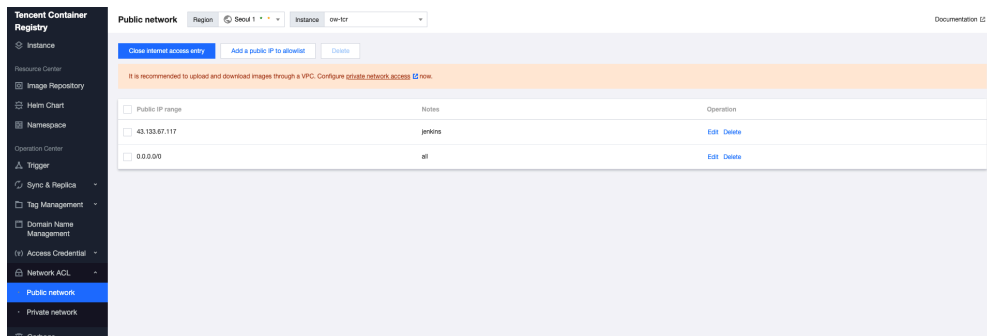
The screenshot shows the 'User account' page in the Tencent Container Registry console. The left sidebar contains navigation options: Instance, Resource Center, Image Repository, Helm Chart, Namespace, Operation Center, Trigger, Sync & Replica, Tag Management, Domain Name Management, Access Credential (expanded), User Account (selected), and Service Account. The main content area shows the 'User account' section for Region 'Seoul 1' and Instance 'ow-tcr'. It includes a 'Create' button and a 'Generate temp login token' button. A blue information box states: 'The access credential generated under a user account is directly associated with the account/subaccount you logged in to, and is deleted, the associated access credential is invalid. For details, see [User Account](#).' Below this is a table with columns 'Description', 'Status', and 'Creation time'. The table contains one entry: 'tcr login user' with status 'Enabled' and creation time '2023-08-29 12:10:39'. The total items are 1.

- Image Repository 생성



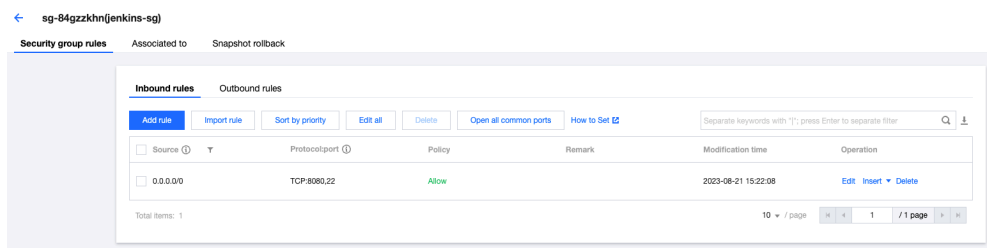
The screenshot shows the 'Image repository' page in the Tencent Container Registry console. The left sidebar is the same as the previous screenshot. The main content area shows the 'Image repository' section for Region 'Seoul 1' and Instance 'ow-tcr'. It includes 'Create' and 'Delete' buttons. A yellow warning box states: 'The current instance supports network access control (ACL). By default, all access sources are blocked. You can refer to [network access control](#) to connect to the VPC or public IP range where the access client is located. It is recommended to upload and download images via the VPC.' Below this is a table with columns 'Name', 'Namespace', 'Repository address', 'Creation time', and 'Operation'. The table contains two entries: 'se-front-dev' and 'se-main-api-dev', both with namespace 'ow-tcr' and repository address 'ow-tcr.tencentcloud.com/ow-tcr/se-front-dev' and 'ow-tcr.tencentcloud.com/ow-tcr/se-main-api-dev' respectively. The total items are 2.

- public network 설정




2. 젠킨스 설치

- 젠킨스 VM 인바운드 규칙에서 8080 포트 허용



- 도커 설치
- 젠킨스 이미지 PULL

Docker

 <https://hub.docker.com/r/jenkins/jenkins/tags>

docker pull jenkins/jenkins:latest

- 젠킨스 컨테이너 실행

```
docker run -d --name jenkins --restart=on-failure \
-p 8080:8080 \
-v /var/jenkins_home:/var/jenkins_home \
-v /var/run/docker.sock:/var/run/docker.sock \
-e TZ=Asia/Seoul \
-u root \
jenkins/jenkins:latest
```

- **-restart** → on-failure 옵션은 비정상 종료시 컨테이너를 재실행합니다.
- **p** → 외부 접속을 위해 호스트의 8080 포트를 바인딩 해주었습니다.

- **v** → 호스트의 `/var/jenkins` 디렉토리를 호스트 볼륨으로 설정하여 jenkins 컨테이너의 home 디렉토리에 마운트시켰습니다.
`docker.sock` 파일은 도커 데몬과 통신할 수 있는 소켓 파일입니다. `docker.sock` 파일을 컨테이너에 마운트시켜서 도커 명령을 실행할 수 있게 해줍니다. 이러한 방식을 `dood(docker out of docker)`라고 합니다.
- **e** → 젠킨스의 `timezone`을 KST 기준으로 설정해줍니다.
- **u** → 추후 권한 문제가 발생할 수 있기 때문에 `user` 옵션을 `root` 사용자로 주었습니다.
- 초기 비밀번호 조회
 초기 비밀번호는 `/var/jenkins_home/secrets/initialAdminPassword`에 저장 됨

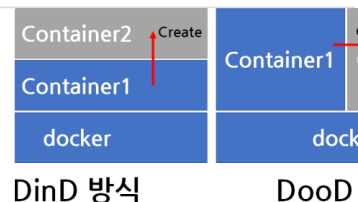
```
cat /var/jenkins_home/secrets/initialAdminPassword
//20598b4120f549c9bc171569a39defb0 // vote
//425a475c650947c3a93f4f08ba9afd69 // sl-space
```

- 젠킨스 컨테이너에 접속하여 docker CLI 설치

```
docker exec -it jenkins bash
apt-get update && apt-get install -y curl
curl -fsSL https://get.docker.com -o get-docker.sh
sh get-docker.sh
```

참고

도커 컨테이너 안에서 도커 실행하기(Docker in Docker, Docker Out of Docker)
 이번 시간은 도커를 사용해 컨테이너에서 컨테이너를 실행하는 방법을 알아봅니다. 도커 컨테이너 내에서 ...
<https://m.blog.naver.com/PostView.naver?isHttpsRedirect=true&blogId=isc0304&logNo=222274955992>



[CI/CD] Jenkins 설치하기 (Docker)

이번 포스트에서는 Docker를 이용하여 아무런 설정이 되어있지 않은 젠킨스를 설치하는 방법을 정리하겠습니다. 테스트 환경 OS: Ubuntu Server 22.04 LTS (HVM)
 성능: t2.micro(free tier) 만약, EC2에서 Jenkins를 설치하신다면 외부에서
https://seosh817.tistory.com/287#google_vignette



3. 젠킨스 파이프라인 세팅

- Credentials 생성

- TCR 접속을 위한 Credentials 생성

- password: git token

Update credentials

Scope ?
Global (Jenkins, nodes, items, all child items, etc) ▼

Username ?
200033049269

☐ Treat username as secret ?

Password ?
Concealed Change Password

ID ?
tencent-tcr

Description ?

- GitHub Credentials 생성

Update credentials

Scope ?
Global (Jenkins, nodes, items, all child items, etc) ▼

Username ?
chihunmanse

☐ Treat username as secret ?

Password ?
Concealed Change Password

ID ?
pete-git

Description ?

- 프로젝트 Repository 안에 젠킨스 스크립트 작성

```
node {
  def app
  def project = "sl-main-api"
  def builder = "${currentBuild.getBuildCauses()[0].shortDescription} / ${currentBuild}
  echo "PROJECT: ${project}"
  echo "BUILDER: ${builder}"
  echo "VERSION: ${env.VERSION}"

  // 추후 젠킨스 파이프라인 안에 scm 구성
  stage('Clone repository') {
    checkout scm
  }

  // 젠킨스 컨테이너 실행시 호스트의 /var/jenkins_home과 마운트 시켰기 때문에 env 파일
  stage('Build image') {
```

```

String currentDirectory = pwd()
sh "sed -i 's+VERSION=..*+VERSION=${env.VERSION}+g' /var/jenkins_home/env/"
sh "echo current directory = ${currentDirectory}"
sh "cp /var/jenkins_home/env/${project}/env.dev ${currentDirectory}/"
    // 이미지 빌드
app = docker.build("ow-tcr.tencentcloudcr.com/ow-tke-tcr/sl-main-api-dev", "--b
}

stage('Test image') {
    app.inside {
        sh 'echo "-----Tests passed-----"'
    }
}

stage('Push image') {
    // 젠킨스에 설정해둔 Credential을 통해 TCR에 env.VERSION을 태그로 이미지 푸시
    docker.withRegistry('http://ow-tcr.tencentcloudcr.com', 'tencent-tcr') {
        app.push("${env.VERSION}")
    }
}

    // update-manifest 파이프라인 실행
stage('Trigger ManifestUpdate') {
    echo "triggering update-manifest job"
    build job: 'update-manifest', parameters: [string(name: 'VERSION', value: env.VER
}
}

```

- 젠킨스 deploy 파이프라인 생성
 - 파라미터 설정
 - VERSION - String Parameter
 - SCM 파이프라인 설정
 - 프로젝트 Repository 연결
 - GitHub Credentials 연결
 - 브랜치 설정
 - Repository 안에 있는 젠킨스 스크립트 파일 설정

Pipeline

Definition

Pipeline script from SCM

SCM ?

Git

Repositories ?

Repository URL ?

https://github.com/ow-develop/ow-sl-main-api.git

Credentials ?

chihunmansej/*****

Add

그룹

Add Repository

Branches to build ?

Branch Specifier (blank for 'any') ?

*/dev

Script Path ?

Jenkinsfile_test

- manifest Repository 안에 젠킨스 스크립트 작성

```
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.TimeZone;

node {
    def app
    echo "BUILDER: ${env.BUILDER}"
    echo "VERSION: ${env.VERSION}"
    def TARGET = "sl-main-api-dev"

    // 추후 젠킨스 파이프라인 안에 scm 구성
    stage('Clone repository') {
        checkout scm
    }

    stage('Update GIT') {
        script {
            try {
                def date = new Date()
                def dateFormat = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss (z Z)")
                def time = TimeZone.getTimeZone("Asia/Seoul")
                dateFormat.setTimeZone(time)
            }
        }
    }
}
```

```

today = dateFormat.format(date)
echo today

// deploy repository의 deployment.yaml 파일의 이미지 태그(버전)값, hi:
withCredentials([usernamePassword(credentialsId: 'pete-git', passwordVaria
  sh "git config user.email chihunmanse@gmail.com"
  sh "git config user.name pete"
  sh "cat './${TARGET}/deployment.yaml'"
  sh "sed -i 's+${TARGET}:.+${TARGET}:${VERSION}+g' './${TARGET}/dep
  sh "cat './${TARGET}/deployment.yaml'"
  sh "echo 'TIMESTAMP: ${today}, VERSION: ${VERSION}, BUILD_NUMBER:
  sh "git add ."
  sh "git commit -m 'Deploy Complete [${TARGET}]: ${env.BUILD_NUMBER}
  // 아래 부분 때문에 credential의 이름이 이메일 주소가 아닌 ID가 되어
  sh "git push https://${GIT_USERNAME}:${GIT_PASSWORD}@github.com/o
}
} catch (err) {
  def errMsg = err.toString()
  echo errMsg
  throw err
}
}
}
}
}
}

```

- 젠킨스 update-manifest 파이프라인 생성
 - 파라미터 설정
 - VERSION - String Parameter
 - BUILDER - String Parameter
 - SCM 파이프라인 설정
 - manifest(deploy) Repository 연결
 - GitHub Credentials 연결
 - 브랜치 설정
 - Repository 안에 있는 젠킨스 스크립트 파일 설정
 - 해당 프로젝트 폴더 안의 스크립트 파일로 설정

Pipeline

Definition
Pipeline script from SCM

SCM ?
Git

Repositories ?

Repository URL ?
https://github.com/ow-develop/sl-deploy-tencent.git

Credentials ?
chihunmanse/*****

Add +

고급 ▾

Add Repository

Branches to build ?

Branch Specifier (blank for 'any') ?
*/master

Script Path ?
./sl-main-api-dev/Jenkinsfile_dev

4. ArgoCD 설치

- TKE 클러스터 API 서버 접속 설정

Cluster APIServer information

After the public network access and private network access are enabled, CLB and network fees are charged based on usage. For billing details, see [CLB Billing Rules](#).

Via internet ☐ Disabled

Private network access ☒ Enabled

Access IP 10.0.0.14

KubeConfig

- Jenkins VM이 접속할 수 있도록 설정 후 Private network 활성화
- KubeConfig 파일 복사
- 젠킨스 VM 접속 후 kubectl 설치
 - kubectl 바이너리 설치

```
curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/b
```

- kubectl 설치

```
sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl
```

- ~/.kube/config 파일에 TKE KubeConfig 내용 붙여넣기
- kubectl get node 등의 명령어로 연결 확인
- ArgoCD 설치

- 네임스페이스 생성

```
kubectl create namespace argocd
```

- ArgoCD 설치

```
kubectl apply -n argocd -f https://raw.githubusercontent.com/argoproj/argo-cd/stable
```

- argocd-server UI에 접속을 위해 **LoadBalancer Service** 로 수정

```
kubectl patch svc argocd-server -n argocd -p '{"spec": {"type": "LoadBalancer"}}'
```



```
kubectl get svc argocd-server -n argocd
```

- 초기 비밀번호 조회

```
//kubectl -n argocd get secret argocd-initial-admin-secret -o jsonpath="{.data.password}"
kubectl -n argocd get secret argocd-initial-admin-secret -o jsonpath="{.data.password}"
//iWyMA0e0aKsHvCsU
//hWEiwy1nER5zOY-w
//HZCjIAtkINViKFb5 // sl-space-argo-cd
```

- Jenkins VM 접속하여 kubectl을 통해 클러스터에 TCR 자격 증명 생성

```
kubectl create secret docker-registry ow-tcr \
--docker-server=ow-tcr.tencentcloudcr.com \
--docker-username=200033049269 \
--docker-password=eyJhbGciOiJSUzI1NiIsImtpZCI6IiJFVNzU6SldVSjpBRktSOjcyQk86
--docker-email=dev@otherworld.network
```

```
kubectl create secret docker-registry nestree-vote-tcr \
  --docker-server=nestree-vote-tcr.tencentcloudcr.com \
  --docker-username=200032194085 \
  --docker-password=eyJhbGciOiJSUzI1NiIsImtpZCI6IjZQQ046WE5MTDpZSVIzOkILQz
  --docker-email=nestree.vote@gmail.com
```

```
// sl-space
kubectl create secret docker-registry ow-tcr \
  --docker-server=ow-tcr.tencentcloudcr.com \
```

```
--docker-username=200032123199 \  
--docker-password=eyJhbGciOiJSUzI1NiIsImtpZCI6IiFVNzU6SldVSjpBRktSOjcyQk86  
--docker-email=dev@otherworld.network
```

참고


리눅스에 kubectl 설치 및 설정

시작하기 전에 클러스터의 마이너(minor) 버전 차이 내에 있는 kubectl 버전을 사용해야 한다. 예를 들어, v1.28 클라이언트는 v1.27, v1.28, v1.29의 컨트롤 플레인과 연동될 수 있다. 호환되는 최신 버전의 kubectl을 사용하면 예기치 않은 문제를 피할

 <https://kubernetes.io/ko/docs/tasks/tools/install-kubectl-linux/>

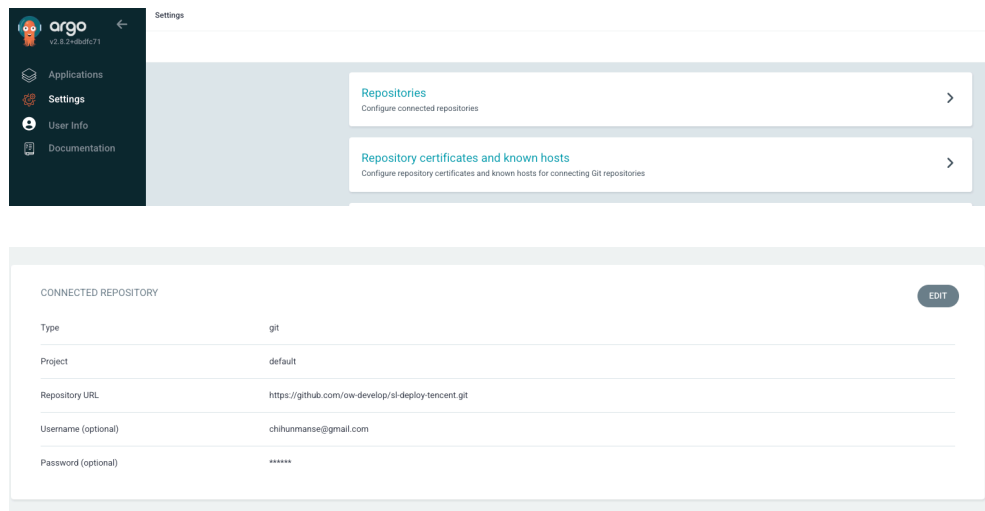
프라이빗 레지스트리에서 이미지 받아오기

이 페이지는 프라이빗 컨테이너 레지스트리나 리포지터리로부터 이미지를 받아오기 위해 시크릿(Secret)을 사용하는 파드를 생성하는 방법을 보여준다. 현재 많은 곳에 서 프라이빗 레지스트리가 사용되고 있다. 여기서는 예시 레지스트리로 Docker Hub

 <https://kubernetes.io/ko/docs/tasks/configure-pod-container/pull-image-private-registry/>

5. ArgoCD 어플리케이션 세팅

- manifest deploy Repository 연결



- 어플리케이션 생성
 - deploy repository의 프로젝트에 해당하는 path를 설정하여 생성

DEV-SL-MAIN-API

EDIT

PROJECT	default
ANNOTATIONS	
CLUSTER	in-cluster (https://kubernetes.default.svc)
NAMESPACE	default 
CREATED AT	08/29/2023 16:32:17 (20 hours ago)
REPO URL	https://github.com/ow-develop/si-deploy-tencent.git
TARGET REVISION	HEAD
PATH	./sl-main-api-dev

- 네임스페이스는 default로 생성
- deployment.yaml manifest 작성

```

apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: sl-main-api-dev
    name: sl-main-api-dev
spec:
  replicas: 1
  selector:
    matchLabels:
      app: sl-main-api-dev
  strategy: {}
  template:
    metadata:
      labels:
        app: sl-main-api-dev
    spec:
      # tcr 이미지 repository 경로 작성, tag 부분은 젠킨스 파이프라인에 의해 업데이트 됨
      containers:
        - image: ow-tcr.tencentcloudcr.com/ow-tke-tcr/sl-main-api-dev:0.0.13
          name: sl-main-api-dev
          resources: {}
          ports:
            - containerPort: 8001
      # 클러스터에 미리 생성해둔 자격 증명 사용
      imagePullSecrets:
        - name: ow-tcr
status: {}
---
apiVersion: v1
kind: Service
metadata:

```

```

name: sl-main-api-dev-service
spec:
  ports:
    - port: 80
      targetPort: 8001
  selector:
    app: sl-main-api-dev
  type: NodePort

```

CI/CD Flow

Jenkins VM

1. Jenkins에서 VERSION 파라미터 입력하여 deploy 파이프라인 실행
2. GitHub 프로젝트 Repository에 있는 Jenkins 스크립트 실행
 - a. 프로젝트 Repository 클론
 - b. /var/jenkins_home/env/\${project} 경로에 있는 env 파일의 VERSION 값을 파라미터 값으로 업데이트
 - c. /var/jenkins_home/env/\${project} 경로에 있는 env 파일을 현재 폴더 (프로젝트 루트 폴더) 로 복사
 - d. NODE_ENV = dev를 인자로 "ow-tcr.tencentcloudcr.com/ow-tke-tcr/sl-main-api-dev" 이미지 빌드
 - e. 파라미터 값을 이미지 태그로 지정하여 TCR에 이미지 푸시
 - f. update-manifest 파이프라인을 VERSION과 BUILDER 파라미터를 넣어 실행

```

node {
  def app
  def project = "sl-main-api"
  def builder = "${currentBuild.getBuildCauses()[0].shortDescription} / ${currentBuild}
  echo "PROJECT: ${project}"
  echo "BUILDER: ${builder}"
  echo "VERSION: ${env.VERSION}"

  stage('Clone repository') {
    checkout scm
  }

  stage('Build image') {
    String currentDirectory = pwd()
    sh "sed -i 's+VERSION=.*+VERSION=${env.VERSION}+g' /var/jenkins_home/env/"

```

```

sh "echo current directory = ${currentDirectory}"
sh "cp /var/jenkins_home/env/${project}/env.dev ${currentDirectory}/"
app = docker.build("ow-tcr.tencentcloudcr.com/ow-tke-tcr/sl-main-api-dev", "--b
}

stage('Test image') {
    app.inside {
        sh 'echo "-----Tests passed-----"'
    }
}

stage('Push image') {
    docker.withRegistry('http://ow-tcr.tencentcloudcr.com', 'tencent-tcr') {
        app.push("${env.VERSION}")
    }
}

stage('Trigger ManifestUpdate') {
    echo "triggering update-manifest job"
    build job: 'update-manifest', parameters: [string(name: 'VERSION', value: env.VER
}
}

```

3. Jenkins 서버에서 update-manifest 파이프라인 실행
4. GitHub Deploy Repository의 해당 프로젝트 폴더 안에 있는 Jenkins 스크립트 실행
 - a. Deploy Repository 클론
 - b. deployment.yaml 파일 안에 이미지 값인 "sl-main-api-dev:VERSION"의 VERSION 부분을 파라미터 값으로 업데이트
 - c. history.txt 파일 안에 로그 입력
 - d. 변경 사항 커밋 후 푸시

```

import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.TimeZone;

node {
    def app
    echo "BUILDER: ${env.BUILDER}"
    echo "VERSION: ${env.VERSION}"
    def TARGET = "sl-main-api-dev"

```

```

stage('Clone repository') {
    checkout scm
}

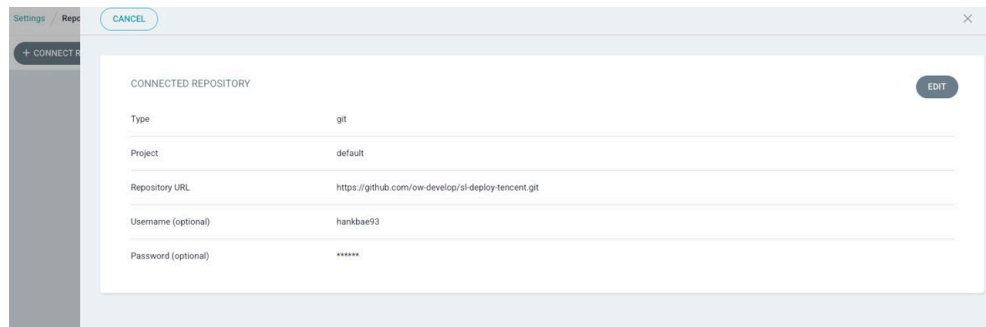
stage('Update GIT') {
    script {
        try {
            def date = new Date()
            def dateFormat = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss (z Z)")
            def time = TimeZone.getTimeZone("Asia/Seoul")
            dateFormat.setTimeZone(time)
            today = dateFormat.format(date)
            echo today

            withCredentials([usernamePassword(credentialsId: 'pete-git', passwordVariable: 'GIT_PASSWORD')]) {
                sh "git config user.email chihunmanse@gmail.com"
                sh "git config user.name pete"
                sh "cat './${TARGET}/deployment.yaml'"
                sh "sed -i 's+${TARGET}:+.*+${TARGET}:${VERSION}+g' './${TARGET}/deployment.yaml'"
                sh "cat './${TARGET}/deployment.yaml'"
                sh "echo 'TIMESTAMP: ${today}, VERSION: ${VERSION}, BUILD_NUMBER: ${BUILD_NUMBER}'"
                sh "git add ."
                sh "git commit -m 'Deploy Complete [${TARGET}]: ${env.BUILD_NUMBER}'"
                sh "git push https://${GIT_USERNAME}:${GIT_PASSWORD}@github.com:/${TARGET}"
            }
        } catch (err) {
            def errMsg = err.toString()
            echo errMsg
            throw err
        }
    }
}
}

```

ArgoCD In TKE

5. 젠킨스에서 빌드한 프로젝트에 해당하는 ArgoCD 어플리케이션 Sync 업데이트
6. 어플리케이션에 연동된 GitHub Repository와 Path 경로의 메니페스트 파일 변동 사항 추적



- username: git email (ex. alddl11@gmail.com)
- password: git token
- Argo 관리

7. deployment.yaml 파일 안에 image 버전 값이 업데이트 됐기 때문에 클러스터에 메니페스트 apply

a. TCR에서 업데이트된 image를 pull 받아와서 배포 실행

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: sl-main-api-dev
  name: sl-main-api-dev
spec:
  replicas: 1
  selector:
    matchLabels:
      app: sl-main-api-dev
  strategy: {}
  template:
    metadata:
      labels:
        app: sl-main-api-dev
    spec:
      containers:
        - image: ow-tcr.tencentcloudcr.com/ow-tke-tcr/sl-main-api-dev:0.0.13
          name: sl-main-api-dev
          resources: {}
          ports:
            - containerPort: 8001
      imagePullSecrets:
        - name: ow-tcr
status: {}
---
```

apiVersion: v1

```
kind: Service
metadata:
  name: sl-main-api-dev-service
spec:
  ports:
    - port: 80
      targetPort: 8001
  selector:
    app: sl-main-api-dev
  type: NodePort
```