# LAZARUS

FOR ETERNITY

# PORTING MANUAL

# 목 차

# I. 개발환경

## 1. 프로젝트 기술 스택

**Server :** AWS EC2 Ubuntu 20.04 LTS

**Visual Studio Code :** 1.78.2

**IntelliJ IDEA :** 2022.3.1 (Ultimate Edition) 17.0.5+1-b653.23 amd64

**JVM :** OpenJDK 11

**Spring Boot:** 2.7.11

**Gradle:** 7.6.1

**Node.js :** 18.16.0

**Vue.js:** 2.7.14

**MariaDB :** 10.11.2

**Redis :** 7.0.11

**MongoDB :** 6.0.5

**Kubernetes:** 1.26.2

**CRI-O:** 1.26.1

**Nginx Ingress Controller:** 1.6.4

**Jenkins :** 2.404

**ArgoCD:** 2.7.1

## 2. 설정 파일 목록과 프로젝트내 경로

**공통:**

- **Jenkinsfile :** /

### Frontend:

- **Dockerfile :** /front/admin-front

### MainServer:

- **Dockerfile :** /api-server/MainServer

### LogServer:

- **Dockerfile :** /api-server/LogServer

### SchedulerServer:

- **Dockerfile :** /api-server/SchedulerServer

## 3. Kubernetes Manifests

### jenkins.yaml:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: jenkins
spec:
  replicas: 1
  selector:
    matchLabels:
      app: jenkins
  template:
    metadata:
      labels:
        app: jenkins
    spec:
      securityContext:
        fsGroup: 0
        runAsUser: 0
      containers:
      - name: jenkins
        image: jenkins/jenkins:latest
        securityContext:
```

```yaml
          privileged: true
        env:
        - name: TZ
          value: Asia/Seoul
        ports:
        - containerPort: 8080
        - containerPort: 50000
        volumeMounts:
        - name: jenkins-home
          mountPath: /var/jenkins_home
      nodeSelector:
        node-role.kubernetes.io/control-plane: ""
      volumes:
      - name: jenkins-home
        persistentVolumeClaim:
          claimName: jenkins-pvc

---

apiVersion: v1
kind: Service
metadata:
  name: jenkins-svc
spec:
  selector:
    app: jenkins
  ports:
    - protocol: TCP
      port: 8080
      targetPort: 8080
      nodePort: 8093
  type: NodePort

---

apiVersion: v1
kind: Service
metadata:
  name: jenkins-jnlp
spec:
  selector:
    app: jenkins
  ports:
    - protocol: TCP
      port: 50000
      targetPort: 50000
      nodePort: 50000
  type: NodePort
```

### jenkins-pv.yaml:

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: jenkins-pv
spec:
  capacity:
    storage: 1Gi # 스토리지 용량 1GB
  volumeMode: Filesystem # 파일 시스템 형식
  accessModes: # 읽기/쓰기 옵션
  - ReadWriteOnce
    #  storageClassName: manual
  persistentVolumeReclaimPolicy: Retain
  hostPath:
    path: /var/jenkins_home # 스토리지를 연결할 Path

---

apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: jenkins-pvc
spec:
  accessModes: # AccessModes
  - ReadWriteOnce
  volumeMode: Filesystem # 파일  시스템 형식
  resources:
    requests:
      storage: 1Gi # 1GB 요청
      #  storageClassName: manual # 스토리지 클래스 명
```

### ingress-nginx.yaml:

```
curl -o ingress-nginx.yaml
https://raw.githubusercontent.com/kubernetes/ingress-nginx/controller-v1.6.4/deploy/static/provider/baremetal/deploy.yaml

# NodePort에 80, 443 포트 할당
vi ingress-nginx.yaml
```

```
# Service 부분에 nodePort 추가
~~~
ports:
  - appProtocol: http
    name: http
    port: 80
    protocol: TCP
    targetPort: http
    nodePort: 80
  - appProtocol: https
    name: https
    port: 443
    protocol: TCP
    targetPort: https
    nodePort: 443
~~~
```

### cluster-issuer.yaml:

```
apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
  name: letsencrypt-prod
spec:
  acme: # Automated Certificate Management Environment
    # 어떤 acme 서버를 사용할 지 지정 (아래 예제는 let's encrypt의 CA)
    server: https://acme-v02.api.letsencrypt.org/directory
    # 사용자 이메일 주소 기재
    email: 사용자이메일
    # 사용자의 개인키를 저장할 Secret 리소스 이름을 지정
    privateKeySecretRef:
```

```
      name: letsencrypt-prod
    # 도메인 주소에 대한 소유권 증명을 위한 방법 선택
    solvers:
    - http01: # http 요청을 통한 도메인 주소 소유권 증명 방법 사용
        ingress: # 이 때 사용할 ingress 컨트롤러 지정
          class: nginx # ingress 컨트롤러 타입 기재
```

### certificate.yaml:

```
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: my-certificate
  namespace: default
spec:
  secretName: my-certificate # TLS 키 이름을 지정합니다.
  duration: 2160h # 90d
  renewBefore: 360h # 15d
  issuerRef:
    name: letsencrypt-prod # cluster-issuer.yaml 의 이름과 일치해야 합니다.
    kind: ClusterIssuer
  dnsNames:
  - 서버도메인
```

### mariadb.yaml:

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: mariadb
spec:
  serviceName: mariadb
  replicas: 1
  selector:
    matchLabels:
      app: mariadb
  template:
    metadata:
      labels:
        app: mariadb
    spec:
      nodeSelector:
        node-role.kubernetes.io/control-plane: ""
      containers:
      - name: mariadb
        image: mariadb:latest
        ports:
        - containerPort: 3306
          name: mysql
        envFrom:
        - secretRef:
            name: mariadb-secret
        volumeMounts:
        - name: mariadb-data
          mountPath: /var/lib/mysql
  volumeClaimTemplates:
  - metadata:
      name: mariadb-data
    spec:
      accessModes: [ "ReadWriteOnce" ]
      resources:
        requests:
          storage: 10Gi

---

apiVersion: v1
kind: Service
metadata:
  name: mariadb-svc
spec:
  selector:
    app: mariadb
  ports:
    - protocol: TCP
      port: 3306
      targetPort: 3306
      nodePort: 3324
  type: NodePort
```

**mariadb-pv-0.yaml:**

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: mariadb-pv-0
  labels:
    app: mariadb
spec:
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: /home/ubuntu/data/mariadb
```

**redis.yaml:**

```yaml
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: redis
spec:
  serviceName: redis
  replicas: 1
  selector:
    matchLabels:
      app: redis
  template:
    metadata:
      labels:
        app: redis
    spec:
      nodeSelector:
        node-role.kubernetes.io/control-plane: ""
      containers:
      - name: redis
        image: redis:latest
        command: ["redis-server", "/redis-conf/redis.conf"]
        ports:
        - containerPort: 6379
          name: redis
        volumeMounts:
        - name: redis-data
```

```
          mountPath: /data
        - name: redis-conf
          mountPath: /redis-conf
      volumes:
      - name: redis-conf
        secret:
          secretName: redis-secret
  volumeClaimTemplates:
  - metadata:
      name: redis-data
    spec:
      accessModes: [ "ReadWriteOnce" ]
      resources:
        requests:
          storage: 1Gi

---

apiVersion: v1
kind: Service
metadata:
  name: redis-svc
spec:
  selector:
    app: redis
  ports:
    - protocol: TCP
      port: 6379
      targetPort: 6379
      nodePort: 6379
  type: NodePort
```

### redis-pv-0.yaml:

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: redis-pv-0
  labels:
    app: redis
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: /home/ubuntu/data/redis
```

### mongo.yaml:

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: mongodb
spec:
  serviceName: mongodb
  replicas: 1
  selector:
    matchLabels:
      app: mongodb
  template:
    metadata:
      labels:
        app: mongodb
    spec:
      nodeSelector:
        node-role.kubernetes.io/control-plane: ""
      containers:
      - name: mongodb
        image: mongo:latest
        ports:
        - containerPort: 27017
          name: mongodb
        envFrom:
        - secretRef:
            name: mongodb-secret
        volumeMounts:
        - name: mongodb-data
          mountPath: /data/db
  volumeClaimTemplates:
  - metadata:
      name: mongodb-data
    spec:
      accessModes: [ "ReadWriteOnce" ]
      resources:
        requests:
          storage: 100Gi

---

apiVersion: v1
kind: Service
metadata:
  name: mongodb-svc
spec:
  selector:
    app: mongodb
  ports:
    - protocol: TCP
      port: 27017
      targetPort: 27017
      nodePort: 3325
  type: NodePort
```

### mongo-pv-0.yaml:

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: mongodb-pv-0
  labels:
    app: mongodb
spec:
  capacity:
    storage: 100Gi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: /home/ubuntu/data/mongodb
```

### mongo-express.yaml:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mongo-express
  labels:
    app: mongo-express
spec:
  replicas: 1
  selector:
    matchLabels:
      app: mongo-express
  template:
    metadata:
      labels:
        app: mongo-express
    spec:
      containers:
      - name: mongo-express
        image: mongo-express:latest
        ports:
        - containerPort: 8081
        envFrom:
        - secretRef:
            name: me-secret

---

apiVersion: v1
kind: Service
metadata:
  name: mongo-express-svc
spec:
  selector:
    app: mongo-express
  ports:
    - protocol: TCP
      port: 8081
      targetPort: 8081
      nodePort: 8002
  type: NodePort
```

### admin-front.yaml:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: admin-front
  namespace: default
spec:
  replicas: 1
  revisionHistoryLimit: 2
  selector:
    matchLabels:
      app: admin-front
  template:
    metadata:
      labels:
        app: admin-front
    spec:
      containers:
      - name: admin-front
        image: docker.io/sadoruin/msa-admin-front:1
        ports:
        - containerPort: 80

---

apiVersion: v1
kind: Service
metadata:
  name: admin-front
  namespace: default
spec:
  selector:
    app: admin-front
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
  type: ClusterIP
```

### msa-mainserver.yaml:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: msa-mainserver
  namespace: default
spec:
  replicas: 2
  revisionHistoryLimit: 2
  selector:
    matchLabels:
      app: msa-mainserver
  template:
    metadata:
      labels:
        app: msa-mainserver
    spec:
      containers:
      - name: msa-mainserver
        image: docker.io/sadoruin/msa-mainserver:1
        ports:
        - containerPort: 8080
        volumeMounts:
        - name: secret-volume
          mountPath: /config
          readOnly: true
        env:
        - name: SPRING_CONFIG_LOCATION
          value: "file:/config/application-mainserver.yml"
      volumes:
      - name: secret-volume
        secret:
          secretName: mainserver-secret

---

apiVersion: v1
kind: Service
metadata:
  name: msa-mainserver
  namespace: default
spec:
  selector:
    app: msa-mainserver
  ports:
    - protocol: TCP
      port: 8080
      targetPort: 8080
  type: ClusterIP
```

### msa-logserver.yaml:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: msa-logserver
  namespace: default
spec:
  replicas: 2
  revisionHistoryLimit: 2
  selector:
    matchLabels:
      app: msa-logserver
  template:
    metadata:
      labels:
        app: msa-logserver
    spec:
      containers:
      - name: msa-logserver
        image: docker.io/sadoruin/msa-logserver:1
        ports:
        - containerPort: 8080
        volumeMounts:
        - name: secret-volume
          mountPath: /config
          readOnly: true
        env:
        - name: SPRING_CONFIG_LOCATION
          value: "file:/config/application-logserver.yml"
      volumes:
      - name: secret-volume
        secret:
          secretName: logserver-secret
```

```
---

apiVersion: v1
kind: Service
metadata:
  name: msa-logserver
  namespace: default
spec:
  selector:
    app: msa-logserver
  ports:
    - protocol: TCP
      port: 8080
      targetPort: 8080
  type: ClusterIP
```

**msa-schedulerserver.yaml:**

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: msa-schedulerserver
  namespace: default
spec:
  replicas: 1
  revisionHistoryLimit: 2
  selector:
    matchLabels:
      app: msa-schedulerserver
  template:
    metadata:
      labels:
        app: msa-schedulerserver
    spec:
      containers:
      - name: msa-schedulerserver
        image: docker.io/sadoruin/msa-schedulerserver:1
        ports:
        - containerPort: 8080
        volumeMounts:
        - name: secret-volume
          mountPath: /config
          readOnly: true
        env:
        - name: SPRING_CONFIG_LOCATION
          value: "file:/config/application-scheduler.yml"
      volumes:
      - name: secret-volume
        secret:
          secretName: scheduler-secret

---

apiVersion: v1
kind: Service
metadata:
  name: msa-schedulerserver
  namespace: default
spec:
  selector:
    app: msa-schedulerserver
  ports:
    - protocol: TCP
      port: 8080
      targetPort: 8080
  type: ClusterIP
```

## 4. 설정 파일 및 환경 변수 정보

**공통:**

- **Jenkinsfile :**

```
pipeline {
    agent any

    tools {
        nodejs "nodejs"
    }

    stages {
        stage('Project Build') {
            steps {
                script {
                    if(env.BRANCH_NAME == 'feature-front/admin') {
                        echo "Front Project Build Step"
                        dir('front/admin-front') {
                            withCredentials([file(credentialsId:
'env-file', variable: 'ENV_FILE')]) {
                                sh 'cp $ENV_FILE .env'
                                sh 'npm install'
                                sh 'npm run build'
                            }
                        }
                    } else if(env.BRANCH_NAME == 'api-server/member') {
                        echo "Main Server Project Build Step"
                        dir('api-server/MainServer') {
                            sh 'chmod +x gradlew'
                            sh './gradlew clean build -x test'
                        }
                    } else if(env.BRANCH_NAME == 'api-server/logserver') {
                        echo "Log Server Project Build Step"
                        dir('api-server/LogServer') {
                            sh 'chmod +x gradlew'
                            sh './gradlew clean build -x test'
                        }
                    } else if(env.BRANCH_NAME ==
'scheduler-server/schedulerServer') {
                        echo "Scheduler Server Project Build Step"
                        dir('scheduler-server/SchedulerServer') {
                            sh 'chmod +x gradlew'
                            sh './gradlew clean build -x test'
                        }
                    }
                }
            }
        }
        stage('Image Build') {
            environment {
                PATH = "/busybox:/kaniko:$PATH"
            }
            steps {
                script {
                    podTemplate(yaml: """
                      kind: Pod
                      metadata:
                        name: kaniko
                      spec:
                        containers:
                        - name: kaniko
                          image: gcr.io/kaniko-project/executor:debug
                          imagePullPolicy: Always
                          command:
                          - sleep
                          args:
                          - 99d
                          volumeMounts:
                          - name: shared-workspace
                            mountPath: /workspace
                          - name: docker-config
                            mountPath: /kaniko/.docker
                          tty: true
                        nodeSelector:
                          node-role.kubernetes.io/control-plane: ""
                        volumes:
                        - name: shared-workspace
                          hostPath:
                            path: ${WORKSPACE}
                            type: Directory
                        - name: docker-config
                          secret:
                            secretName: regcred
                            items:
                            - key: .dockerconfigjson
                              path: config.json
                        """) {
                        node(POD_LABEL) {
                            container(name: 'kaniko', shell: '/busybox/sh')
{
                                if(env.BRANCH_NAME ==
'feature-front/admin') {
                                    echo "Front Image Build Step"
                                    sh """#!/busybox/sh
                                    /kaniko/executor
--context=/workspace/front/admin-front
--dockerfile=/workspace/front/admin-front/Dockerfile
--destination=sadoruin/msa-admin-front:${env.BUILD_NUMBER}
```

```
                                    """
                                } else if(env.BRANCH_NAME ==
'api-server/member') {
                                    echo "Main Server Image Build Step"
                                    sh """#!/busybox/sh
                                    /kaniko/executor
--context=/workspace/api-server/MainServer
--dockerfile=/workspace/api-server/MainServer/Dockerfile
--destination=sadoruin/msa-mainserver:${env.BUILD_NUMBER}
                                    """
                                } else if(env.BRANCH_NAME ==
'api-server/logserver') {
                                    echo "Log Server Image Build Step"
                                    sh """#!/busybox/sh
                                    /kaniko/executor
--context=/workspace/api-server/LogServer
--dockerfile=/workspace/api-server/LogServer/Dockerfile
--destination=sadoruin/msa-logserver:${env.BUILD_NUMBER}
                                    """
                                } else if(env.BRANCH_NAME ==
'scheduler-server/schedulerServer') {
                                    echo "Scheduler Server Image Build
Step"

                                    sh """#!/busybox/sh
                                    /kaniko/executor
--context=/workspace/scheduler-server/SchedulerServer
--dockerfile=/workspace/scheduler-server/SchedulerServer/Dockerfile
--destination=sadoruin/msa-schedulerserver:${env.BUILD_NUMBER}
                                    """
                                }
                            }
                        }
                    }
                }
            }
        }
        stage('Deploy') {
            steps {
                script {
                    dir('/git') {
                        git branch: 'main',
                            credentialsId: 'gitlab-account',
                            url: '레포지토리 주소'
                        sh 'git config --global user.email "이메일"'
                        sh 'git config --global user.name "이름"'

                        if(env.BRANCH_NAME == 'feature-front/admin') {
                            echo "Front Deploy Step"
                            sh """
                                sed -i
's/msa-admin-front:\\([^:]*\\)/msa-admin-front:${env.BUILD_NUMBER}/g'
servers/admin-front.yaml
                                git add servers/admin-front.yaml
                                git commit -m 'Update msa-admin-front tag
to ${env.BUILD_NUMBER}'
                            """
                        } else if(env.BRANCH_NAME == 'api-server/member') {
                            echo "Main Server Deploy Step"
                            sh """
                                sed -i
's/msa-mainserver:\\([^:]*\\)/msa-mainserver:${env.BUILD_NUMBER}/g'
servers/msa-mainserver.yaml
                                git add servers/msa-mainserver.yaml
                                git commit -m 'Update msa-mainserver tag to
${env.BUILD_NUMBER}'
                            """
                        } else if(env.BRANCH_NAME ==
'api-server/logserver') {
                            echo "Log Server Deploy Step"
                            sh """
                                sed -i
's/msa-logserver:\\([^:]*\\)/msa-logserver:${env.BUILD_NUMBER}/g'
servers/msa-logserver.yaml
                                git add servers/msa-logserver.yaml
                                git commit -m 'Update msa-logserver tag to
${env.BUILD_NUMBER}'
                            """
                        } else if(env.BRANCH_NAME ==
'scheduler-server/schedulerServer') {
                            echo "Scheduler Server Deploy Step"
                            sh """
                                sed -i
's/msa-schedulerserver:\\([^:]*\\)/msa-schedulerserver:${env.BUILD_NUMBER}/
g' servers/msa-schedulerserver.yaml
```

```
                                git add servers/msa-schedulerserver.yaml
                                git commit -m 'Update msa-schedulerserver
tag to ${env.BUILD_NUMBER}'
                            """
                        }

                        withCredentials([usernamePassword(credentialsId:
'gitlab-account', passwordVariable: 'GIT_PASSWORD', usernameVariable:
'GIT_USERNAME')]) {
                            sh 'git remote set-url origin
https://$GIT_USERNAME:$GIT_PASSWORD@레포지토리주소'
                            sh 'git push origin main'
                        }
                    }

                }
            }
        }
    }
}
```

### Frontend:

- **Dockerfile :**

```
FROM nginx:stable-alpine
# Set timezone
RUN apk add --no-cache tzdata && \
    cp /usr/share/zoneinfo/Asia/Seoul /etc/localtime && \
    echo "Asia/Seoul" > /etc/timezone && \
    apk del tzdata
ADD ./dist /usr/share/nginx/html
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

- **.env :**

```
VUE_APP_SERVER_URL=https://서버도메인
```

### MainServer:

- **Dockerfile :**

```
FROM openjdk:11-jdk-slim
ARG JAR_FILE=build/libs/*.jar
COPY ${JAR_FILE} app.jar
EXPOSE 8080
ENTRYPOINT ["java", "-jar", "-Duser.timezone=Asia/Seoul", "app.jar"]
```

- **application-mainserver.yml :**

```
springdoc:
  version: v1.0.0
  api-docs:
    path: /api-docs
  default-consumes-media-type: application/json
  default-produces-media-type: application/json
  swagger-ui:
    operations-sorter: alpha
    tags-sorter: alpha
    path: /swagger-ui.html
    disable-swagger-default-url: true
    display-query-params-without-oauth2: true

spring:
  mail:
    verify-link: https://서버도메인/api/main/users/verify/request/
    host: smtp.gmail.com
    port: 587
    username: 구글 계정명
    password: 앱비밀번호
    properties:
      mail:
        smtp:
          starttls:
            enable: true
            required: true
          auth: true
          connection-timeout: 5000
          timeout: 5000
          write-timeout: 5000
  data:
    redis:
      host: redis-svc
      port: 6379
      password: 비밀번호

  datasource:
    url: jdbc:mariadb://mariadb-svc:3306/데이터베이스명
    username: 계정명
    password: 비밀번호
    driver-class-name: org.mariadb.jdbc.Driver
  jpa:
    hibernate:
      ddl-auto: none
    properties:
      hibernate:
        show_sql: true
        format_sql: true
        default_batch_fetch_size: 100
```

### LogServer:

- **Dockerfile :**

```
FROM openjdk:11-jdk-slim
ARG JAR_FILE=build/libs/*.jar
COPY ${JAR_FILE} app.jar
EXPOSE 8080
ENTRYPOINT ["java", "-jar", "-Duser.timezone=Asia/Seoul", "app.jar"]
```

- **application-logserver.yml :**

```
springdoc:
  version: v1.0.0
  api-docs:
    path: /api-docs
  default-consumes-media-type: application/json
```

```yaml
  default-produces-media-type: application/json
  swagger-ui:
    operations-sorter: alpha
    tags-sorter: alpha
    path: /swagger-ui.html
    disable-swagger-default-url: true
    display-query-params-without-oauth2: true

spring:
  data:
    mongodb:
      host: mongodb-svc
      port: 27017
      authentication-database: admin
      username: 계정명
      password: 비밀번호
      database: 데이터베이스명

  datasource:
    url: jdbc:mariadb://mariadb-svc:3306/데이터베이스명
    username: 계정명
    password: 비밀번호
    driver-class-name: org.mariadb.jdbc.Driver
  jpa:
    hibernate:
      ddl-auto: none
    properties:
      hibernate:
        show_sql: true
        format_sql: true
        default_batch_fetch_size: 100
```

### SchedulerServer:

- **Dockerfile :**

```
FROM openjdk:11-jdk-slim
ARG JAR_FILE=build/libs/*.jar
COPY ${JAR_FILE} app.jar
EXPOSE 8080
ENTRYPOINT ["java", "-jar", "-Duser.timezone=Asia/Seoul", "app.jar"]
```

- **application-scheduler.yml :**

```
spring:
  data:
    mongodb:
      host: mongodb-svc
      port: 27017
      authentication-database: admin
      username: 계정명
      password: 비밀번호
      database: 데이터베이스명

  datasource:
    url: jdbc:mariadb://mariadb-svc:3306/데이터베이스명
    username: 계정명
    password: 비밀번호
    driver-class-name: org.mariadb.jdbc.Driver
  jpa:
    hibernate:
      ddl-auto: none
    properties:
      hibernate:
        show_sql: true
        format_sql: true
        default_batch_fetch_size: 100
```

# II. 빌드 및 배포

## 1. Kubernetes 설치

- Ubuntu 방화벽 사용시 아래 포트들 허용

```
# Master Node
sudo ufw allow 80
sudo ufw allow 443
sudo ufw allow 179
sudo ufw allow 6443
sudo ufw allow 2379
sudo ufw allow 2380
sudo ufw allow 10250
sudo ufw allow 10257
sudo ufw allow 10259
sudo ufw allow 53

# Worker Node
sudo ufw allow 10250

# 나중에 외부에서 접속할 NodePort가 있다면 추가적으로 허용해준다.
```

- CRI-O 설치

```
# .conf 파일을 만들어 부팅 시 모듈을 로드한다

# Controller / Worker

cat <<EOF | sudo tee /etc/modules-load.d/crio.conf
overlay
br_netfilter
EOF

sudo modprobe overlay
sudo modprobe br_netfilter


# 요구되는 sysctl 파라미터 설정, 이 설정은 재부팅 간에도 유지된다.
# Controller / Worker

cat <<EOF | sudo tee /etc/sysctl.d/99-kubernetes-cri.conf
net.bridge.bridge-nf-call-iptables  = 1
net.ipv4.ip_forward                 = 1
net.bridge.bridge-nf-call-ip6tables = 1
EOF

sudo sysctl --system


# Controller / Worker

sudo -i
export OS=xUbuntu_20.04 # OS 버전
export VERSION=1.26     # cri-o 버전
echo "deb
https://download.opensuse.org/repositories/devel:/kubic:/libcontainers:/sta
ble/$OS/ /" | sudo tee
/etc/apt/sources.list.d/devel:kubic:libcontainers:stable.list

curl -L
"https://download.opensuse.org/repositories/devel:/kubic:/libcontainers:/st
able/$OS/Release.key" | apt-key add -

echo "deb
http://download.opensuse.org/repositories/devel:/kubic:/libcontainers:/stab
le:/cri-o:/$VERSION/$OS/ /" >
/etc/apt/sources.list.d/devel:kubic:libcontainers:stable:cri-o:$VERSION.lis
t

curl -L
https://download.opensuse.org/repositories/devel:kubic:libcontainers:stable
:cri-o:$VERSION/$OS/Release.key | apt-key add -



apt-get update

apt-get install cri-o cri-o-runc

sudo systemctl daemon-reload
sudo systemctl enable crio --now

# 상태확인(active가 running이면 성공)
sudo systemctl status crio


# CRI-O는 기본적으로 systemd cgroup 드라이버를 사용한다.
```

- kubeadm 설치

```
# curl 설치 (보통 설치되어 있으므로 pass)
sudo apt-get update
sudo apt-get install -y apt-transport-https ca-certificates curl


# 구글 클라우드의 공개 사이닝 키를 다운
sudo curl -fsSLo /usr/share/keyrings/kubernetes-archive-keyring.gpg
https://packages.cloud.google.com/apt/doc/apt-key.gpg

# 쿠버네티스 apt 리포지토리를 추가
echo "deb [signed-by=/usr/share/keyrings/kubernetes-archive-keyring.gpg]
https://apt.kubernetes.io/ kubernetes-xenial main" | sudo tee
/etc/apt/sources.list.d/kubernetes.list

# apt 패키지 색인을 업데이트하고, kubelet, kubeadm, kubectl을 설치하고 해당
버전을 고정
sudo apt-get update
sudo apt-get install -y kubelet=1.26.2-00 kubeadm=1.26.2-00
kubectl=1.26.2-00
sudo apt-mark hold kubelet kubeadm kubectl

# Kubernetes CRI-O 구성
sudo vi /etc/systemd/system/kubelet.service.d/10-kubeadm.conf

## 다음 내용 추가
[Service]
Environment="KUBELET_EXTRA_ARGS=--container-runtime=remote
--cgroup-driver=systemd --runtime-request-timeout=15m
--container-runtime-endpoint='unix:///var/run/crio/crio.sock'"
```

```
## 설정 적용을 위해 kubectl 재시작
sudo systemctl daemon-reload
sudo systemctl restart kubelet

# kubectl alias에 관한 shell 설정 추가
echo "alias k='kubectl'" >> ~/.bashrc
source ~/.bashrc
```

- 유용한 플러그인 kubectx, kubens 설치

```
sudo git clone https://github.com/ahmetb/kubectx /opt/kubectx
sudo ln -s /opt/kubectx/kubectx /usr/local/bin/kubectx
sudo ln -s /opt/kubectx/kubens /usr/local/bin/kubens
```

## 2. Kubernetes 세팅

- kubeadm으로 마스터 노드 초기화

```
# pod-network-cidr: 내부네트워크 범위
sudo kubeadm init --pod-network-cidr=192.168.0.0/16

# To start using your cluster, you need to run the following as a regular
user:
mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config

# Alternatively, if you are the root user, you can run:
export KUBECONFIG=/etc/kubernetes/admin.conf

# 마스터노드에 pod생성 허용
## Taints 확인
kubectl describe node <master-node-name> | grep Taints

## Taints가 존재할 경우 제거
kubectl taint node <master-node-name>
node-role.kubernetes.io/control-plane:NoSchedule-
```

- 워커노드 생성(다른 서버가 있을 경우)

```
# 마스터 노드에서 join 커맨드 출력
kubeadm token create --print-join-command

# 다른 기기에서 워커노드 생성
kubeadm join <control-plane-host>:<control-plane-port> --token <token>
--discovery-token-ca-cert-hash sha256:<hash>
```

- Calico CNI 설치

```
k apply -f
https://raw.githubusercontent.com/projectcalico/calico/v3.25.0/manifests/ca
lico.yaml
```

- NodePort 할당 가능 범위 변경

```
sudo vi /etc/kubernetes/manifests/kube-apiserver.yaml
```

```
# 다음 내용 추가
~~~
spec:
  containers:
  - command:
    - --service-node-port-range=1-65535
~~~

# kube-system 네임스페이스의 pod를 삭제하면 다시 생성하면서 수정내용 반영
```

- cert-manager 설치 및 ssl 적용

```
# cert-manager 설치
k apply -f
https://github.com/cert-manager/cert-manager/releases/download/v1.11.1/cert
```

```
-manager.yaml

# cluster-issuer.yaml 생성 및 적용 -> I-3 Kubernetes Manifests 참조
k apply -f cluster-issuer.yaml

# certificate.yaml 생성 및 적용 -> I-3 Kubernetes Manifests 참조
k apply -f certificate.yaml
```

- Nginx Ingress Controller 설치

```
curl -o ingress-nginx.yaml
https://raw.githubusercontent.com/kubernetes/ingress-nginx/controller-v1.6.
4/deploy/static/provider/baremetal/deploy.yaml

# manifest 수정
vi ingress-nginx.yaml
```

```
# Service 부분에 nodePort 추가
~~~
ports:
  - appProtocol: http
    name: http
    port: 80
    protocol: TCP
    targetPort: http
    nodePort: 80
  - appProtocol: https
    name: https
    port: 443
    protocol: TCP
    targetPort: https
    nodePort: 443
~~~
```

```
# ingress-nginx.yaml 적용
k apply -f ingress-nginx.yaml
```

- Ingress 리소스 생성 및 설정 : 서비스를 프록시 하기 위한 적절한 Ingress 설정

예시)

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: nginx-ingress
  annotations:
    kubernetes.io/ingress.class: nginx
    cert-manager.io/cluster-issuer: letsencrypt-prod
spec:
  tls:
  - hosts:
    - 서버도메인
    secretName: my-certificate
  rules:
  - host: 서버도메인
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: my-service
            port:
              number: 80
```

## 3. MariaDB 배포

- 각 API에 맞는 DB Manifest 생성 : **I-3 Kubernetes Manifests**의 mariadb 항목들 참조

- db 환경변수를 설정할 env 파일을 생성 후 secret 리소스 생성

```
vi mariadb.properties
```

```
MYSQL_ROOT_PASSWORD=root비밀번호
MYSQL_USER=계정명
MYSQL_PASSWORD=비밀번호
MYSQL_USER_HOST=%
MYSQL_DATABASE=데이터베이스명
```

```
k create secret generic mariadb-secret --from-env-file=mariadb.properties
```

- Manifest 적용

```
k apply -f mariadb-pv-0.yaml
k apply -f mariadb.yaml
```

### 4. Redis 배포

- 각 API에 맞는 DB Manifest 생성 : **I-3 Kubernetes Manifests**의 redis 항목들 참조

- redis 비밀번호를 설정할 config 파일을 생성 후 secret 리소스 생성

```
vi config
```

```
requirepass 비밀번호
```

```
k create secret generic redis-secret --from-env-file=config
```

- Manifest 적용

```
k apply -f redis-pv-0.yaml
k apply -f redis.yaml
```

### 5. MongoDB 배포

- 각 API에 맞는 DB Manifest 생성 : **I-3 Kubernetes Manifest**의 mongo 항목들 참조

- db 환경변수를 설정할 env 파일을 생성 후 secret 리소스 생성

```
vi mongo.properties
```

```
MONGO_INITDB_ROOT_USERNAME=exodia
MONGO_INITDB_ROOT_PASSWORD=dprwhemvkdldj
```

```
k create secret generic mongodb-secret --from-env-file=mongo.properties
```

- mongo express 환경변수를 설정할 env 파일을 생성 후 secret 리소스 생성

```
vi me.properties
```

```
ME_CONFIG_MONGODB_ADMINUSERNAME=exodia
ME_CONFIG_MONGODB_ADMINPASSWORD=dprwhemvkdldj
ME_CONFIG_BASICAUTH_USERNAME=exodia
ME_CONFIG_BASICAUTH_PASSWORD=dprwhemvkdldj
ME_CONFIG_MONGODB_SERVER=mongodb-svc
ME_CONFIG_MONGODB_PORT=27017
```

```
k create secret generic me-secret --from-env-file=me.properties
```

- Manifest 적용

```
k apply -f mongo-pv-0.yaml
k apply -f mongo.yaml
k apply -f mongo-express.yaml
```

## 6. SpringBoot 배포 사전준비

- Deployment 리소스로 배포시 application.yml 파일들을 적용시키기 위해 secret 리소스로 생성

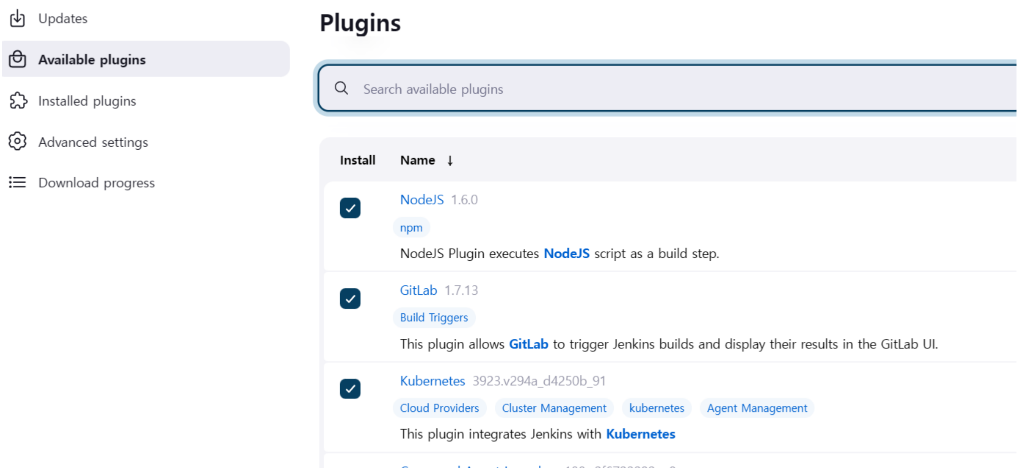
```
k create secret generic mainserver-secret
--from-file=application-mainserver.yml

k create secret generic logserver-secret
--from-file=application-logserver.yml

k create secret generic scheduler-secret
--from-file=application-scheduler.yml
```

## 7. Jenkins CI

- Plugin 설치 : Jenkins 관리 - Plugins에서 NodeJS, GitLab, Kubernetes 플러그인 설치

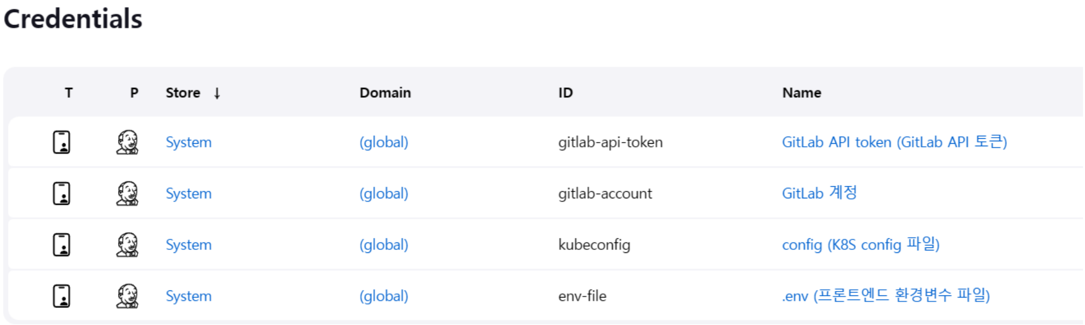


☑ NodeJS 1.6.0
npm
NodeJS Plugin executes **NodeJS** script as a build step.

☑ GitLab 1.7.13
Build Triggers
This plugin allows **GitLab** to trigger Jenkins builds and display their results in the GitLab UI.

☑ Kubernetes 3923.v294a_d4250b_91
Cloud Providers | Cluster Management | kubernetes | Agent Management
This plugin integrates Jenkins with **Kubernetes**

- Credential 설정 : Jenkins 관리 - Credentials



### Credentials

| T | P | Store ↓ | Domain | ID | Name |
|---|---|---|---|---|---|
|  |  | System | (global) | gitlab-api-token | GitLab API token (GitLab API 토큰) |
|  |  | System | (global) | gitlab-account | GitLab 계정 |
|  |  | System | (global) | kubeconfig | config (K8S config 파일) |
|  |  | System | (global) | env-file | .env (프론트엔드 환경변수 파일) |

1. gitlab-api-token : GitLab에서 액세스토큰을 발급받아서 등록
2. gitlab-account : 종류를 Username with password로 하고 GitLab의 아이디, 비밀번호 입력
3. kubeconfig : Kind를 Secret file로 하고 서버에 있는 ~/.kube/config 파일을 등록
4. env-file : Kind를 Secret file로 하고 vue.js 환경변수 파일 .env을 등록

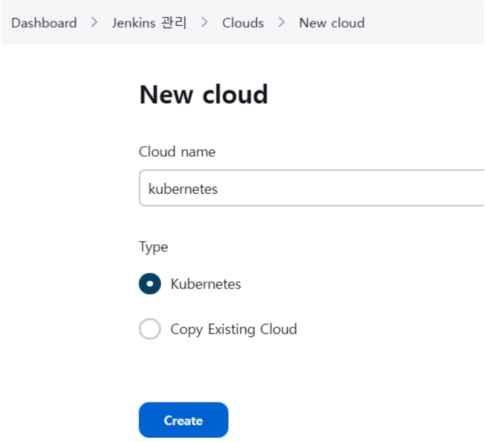
- Kubernetes에 Docker Hub Secret 등록

```
k create secret docker-registry [secret name] \
--docker-username="[Docker Hub 계정]" \
--docker-password="[Docker Hub 패스워드]" \
--docker-server=https://index.docker.io/v1/
```

- 스프링부트 프로젝트들의 build.gradle에 다음 내용 추가

```
jar{
    enabled = false
}
```

- Jenkins와 Kubernetes Cluster를 연동
  1. Jenkins 관리 - Clouds - New cloud
  2. Add a new cloud - Kubernetes



Dashboard > Jenkins 관리 > Clouds > New cloud

New cloud

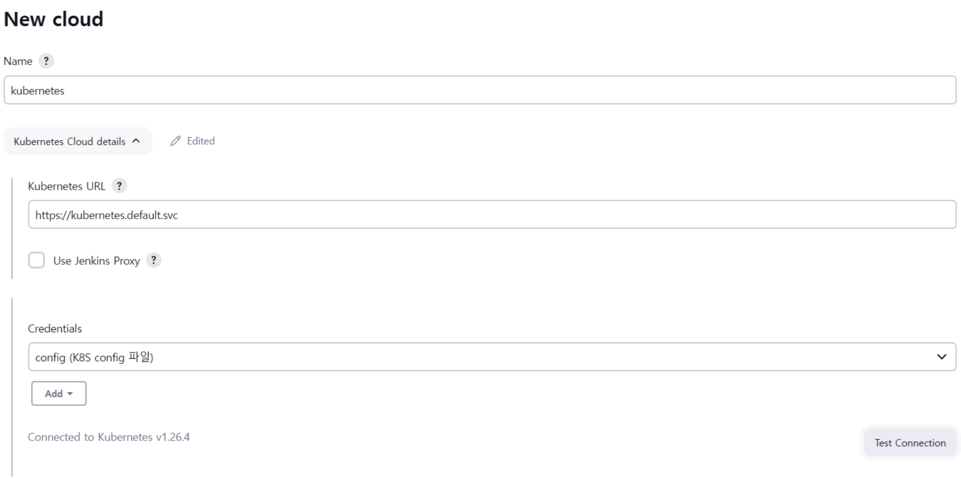
Cloud name: kubernetes

Type

● Kubernetes

○ Copy Existing Cloud

  3. 다음과 같이 설정



New cloud

Name ?: kubernetes

Kubernetes URL ?: https://kubernetes.default.svc

□ Use Jenkins Proxy ?

Credentials: config (K8S config 파일)

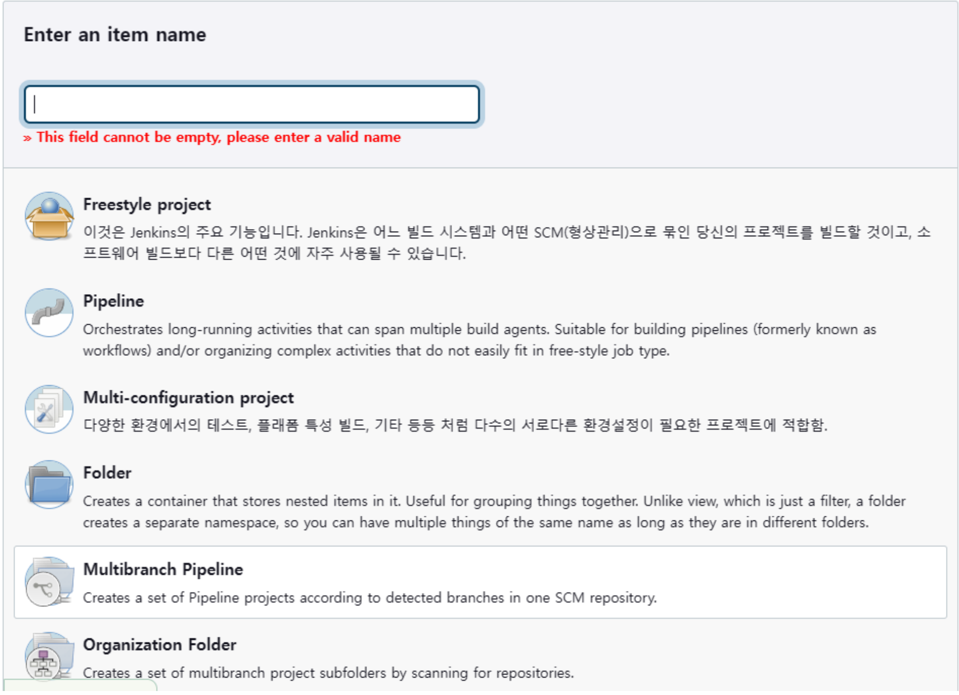
Connected to Kubernetes v1.26.4

- **Name** : 임의로 설정
- **Kubernetes URL** : https://kubernetes.default.svc
- **Credentials** : 등록한 Kubeconfig 파일

  4. Test Connection을 눌러서 연결 확인

- Multibranch Pipeline 생성

1. 새로운 Item - Multibranch Pipeline 선택



Enter an item name

» This field cannot be empty, please enter a valid name

**Freestyle project**
이것은 Jenkins의 주요 기능입니다. Jenkins은 어느 빌드 시스템과 어떤 SCM(형상관리)으로 묶인 당신의 프로젝트를 빌드할 것이고, 소프트웨어 빌드보다 다른 어떤 것에 자주 사용될 수 있습니다.

**Pipeline**
Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.

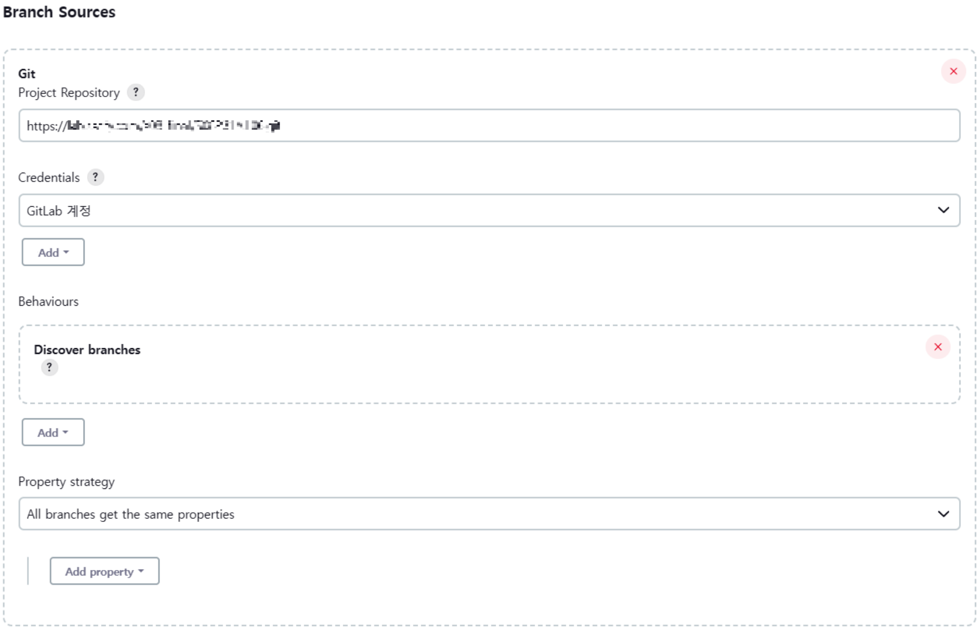
**Multi-configuration project**
다양한 환경에서의 테스트, 플래폼 특성 빌드, 기타 등등 처럼 다수의 서로다른 환경설정이 필요한 프로젝트에 적합함.

**Folder**
Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a separate namespace, so you can have multiple things of the same name as long as they are in different folders.

**Multibranch Pipeline**
Creates a set of Pipeline projects according to detected branches in one SCM repository.

**Organization Folder**
Creates a set of multibranch project subfolders by scanning for repositories.

2. Branch Sources에 Repository와 Credentials 등록



**Branch Sources**

Git

Project Repository: https://[redacted]

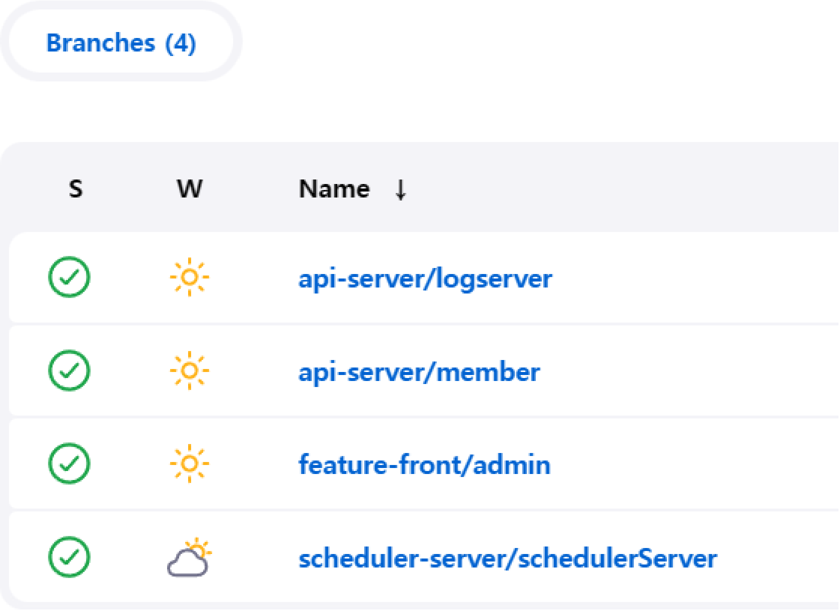
Credentials: GitLab 계정

Behaviours

Discover branches

Property strategy: All branches get the same properties

3. Jenkinsfile이 있는 브랜치는 다음과 같이 감지

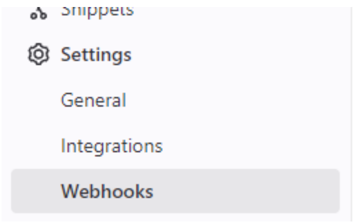


Branches (4)

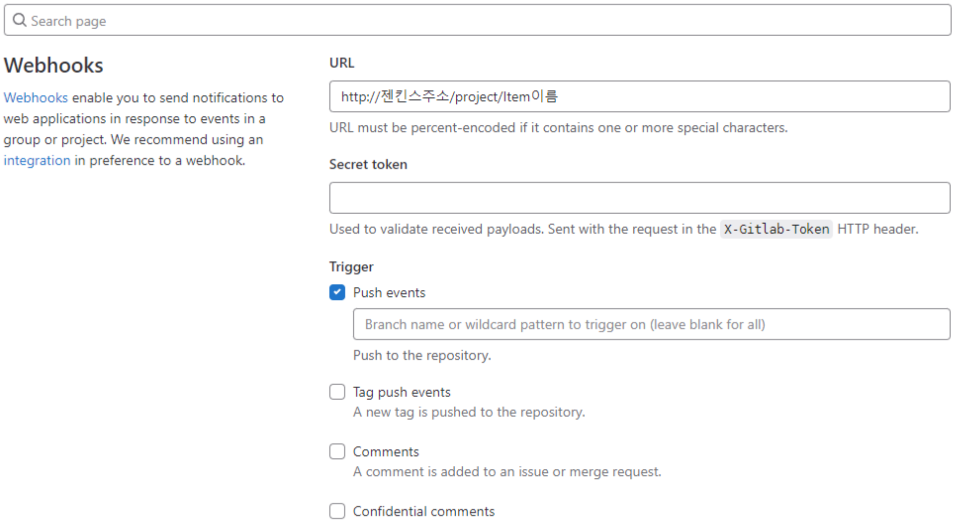
| S | W | Name ↓ |
|---|---|---|
| ✓ | ☀ | api-server/logserver |
| ✓ | ☀ | api-server/member |
| ✓ | ☀ | feature-front/admin |
| ✓ | ⛅ | scheduler-server/schedulerServer |

- GitLab Webhook 설정

1. GitLab 레포지토리 - Settings - Webhooks



2. 다음과 같이 설정



Search page

**Webhooks**

Webhooks enable you to send notifications to web applications in response to events in a group or project. We recommend using an integration in preference to a webhook.

URL

http://젠킨스주소/project/Item이름

URL must be percent-encoded if it contains one or more special characters.

Secret token

Used to validate received payloads. Sent with the request in the `X-Gitlab-Token` HTTP header.

Trigger

☑ Push events

Branch name or wildcard pattern to trigger on (leave blank for all)

Push to the repository.

□ Tag push events
A new tag is pushed to the repository.

□ Comments
A comment is added to an issue or merge request.

□ Confidential comments

## 8. Argo CD

- Argo 설치

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

- NodePort 설정

    1. 서버 설정 열기

```
kubectl edit svc argocd-server -n argocd
```

    2. type을 NodePort로 변경하고 https에 사용할 포트를 nodePort로 추가

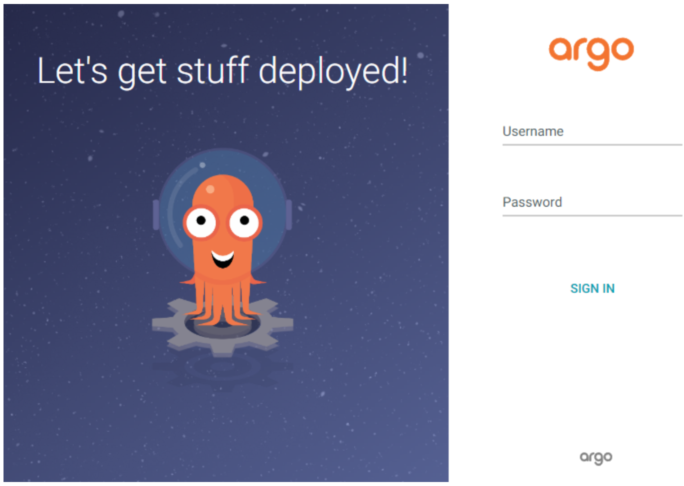
```
~~~
spec:
  ports:
  - name: http
    port: 80
    protocol: TCP
    targetPort: 8080
  - name: https
    nodePort: 사용할포트
    port: 443
    protocol: TCP
    targetPort: 8080
  selector:
    app.kubernetes.io/name: argocd-server
  sessionAffinity: None
  type: NodePort
~~~
```

- Argo CD 로그인

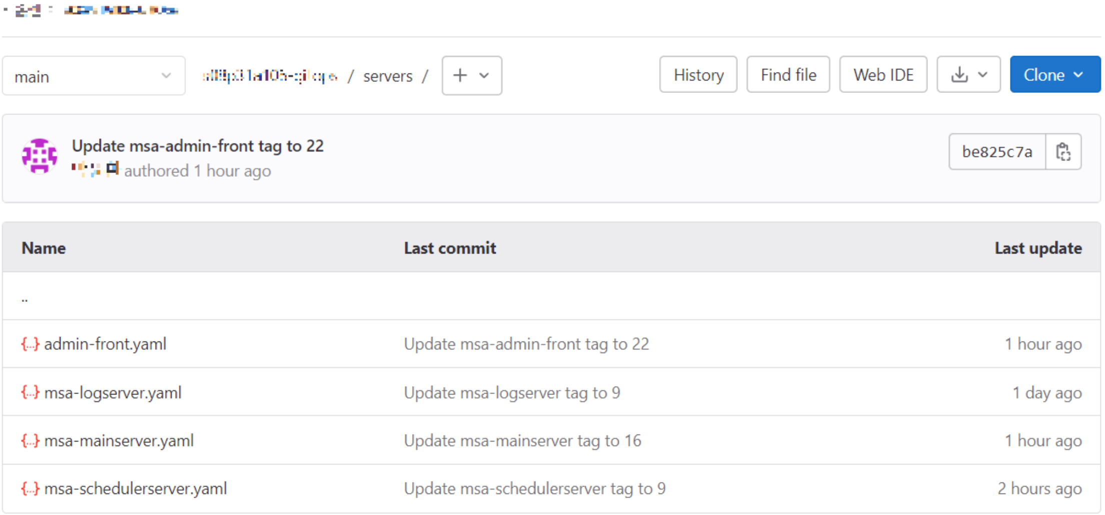
    1. 초기 비밀번호 복사

```
kubectl -n argocd get secret argocd-initial-admin-secret -o
jsonpath="{.data.password}" | base64 -d; echo
```

2. 설정한 포트로 접속해서 로그인 (username : admin)



- Manifest들을 푸시한 레포지토리 준비 (**I-3 Kubernetes Manifest** 참조)

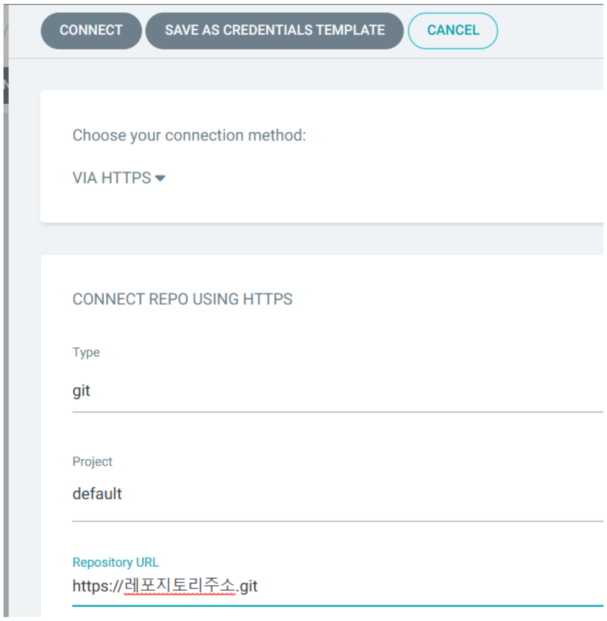


Update msa-admin-front tag to 22
[redacted] authored 1 hour ago

be825c7a

| Name | Last commit | Last update |
|---|---|---|
| .. | | |
| {..} admin-front.yaml | Update msa-admin-front tag to 22 | 1 hour ago |
| {..} msa-logserver.yaml | Update msa-logserver tag to 9 | 1 day ago |
| {..} msa-mainserver.yaml | Update msa-mainserver tag to 16 | 1 hour ago |
| {..} msa-schedulerserver.yaml | Update msa-schedulerserver tag to 9 | 2 hours ago |

- Settings - Repository - CONNECT REPO 설정



Choose your connection method:

VIA HTTPS ▾

CONNECT REPO USING HTTPS

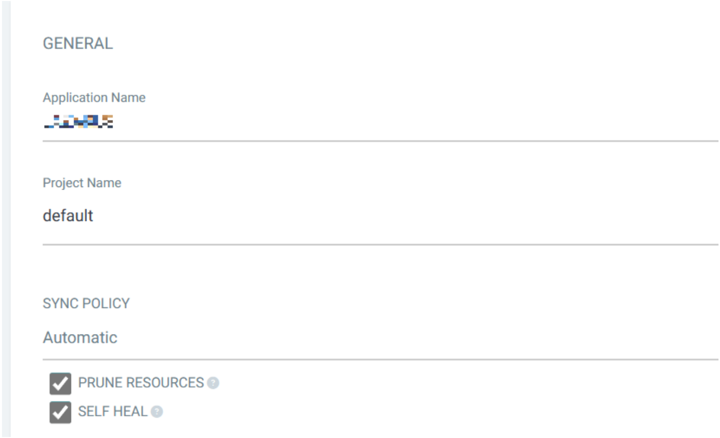
Type: git

Project: default

Repository URL: https://레포지토리주소.git

  - **Project** : 기본으로 default 선택가능
  - **Repository URL** : Manifest를 올린 레포지토리 주소
  - **Username** : 레포지토리 계정 아이디
  - **Password** : 레포지토리 계정 비밀번호

- Applications - NEW APP 설정



GENERAL

Application Name: [redacted]

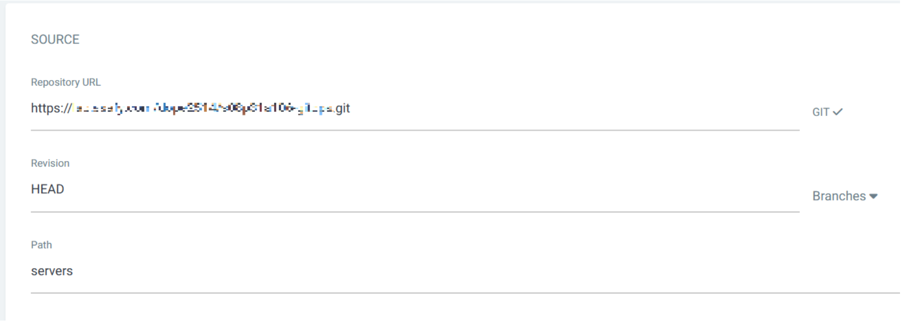
Project Name: default

SYNC POLICY: Automatic

☑ PRUNE RESOURCES

☑ SELF HEAL

  - **Application Name** : 임의로 설정
  - **Project Name** : 기본으로 default 선택 가능
  - **SYNC POLICY** : Automatic
  - **PRUNE RESOURCES** : 체크
  - **SELF HEAL** : 체크



SOURCE

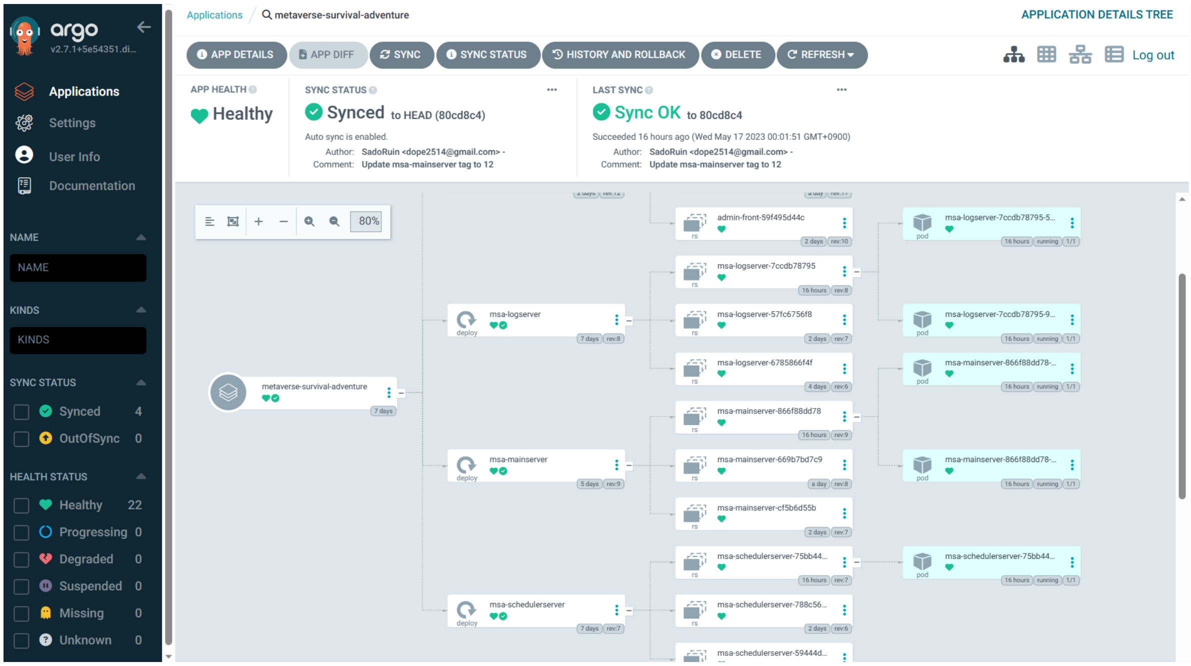
Repository URL: https://[redacted].git

Revision: HEAD

Path: servers

- **Repository URL** : 아까 설정한 주소 선택
- **Path** : 레포지토리내에 manifest들이 들어있는 디렉토리명

- 리소스 한눈에 확인가능 및 젠킨스 빌드 이후 자동으로 배포



APP HEALTH: Healthy

SYNC STATUS: Synced to HEAD (80cd8c4)

Auto sync is enabled.
Author: SadoRuin <dope2514@gmail.com> -
Comment: Update msa-mainserver tag to 12

LAST SYNC: Sync OK to 80cd8c4

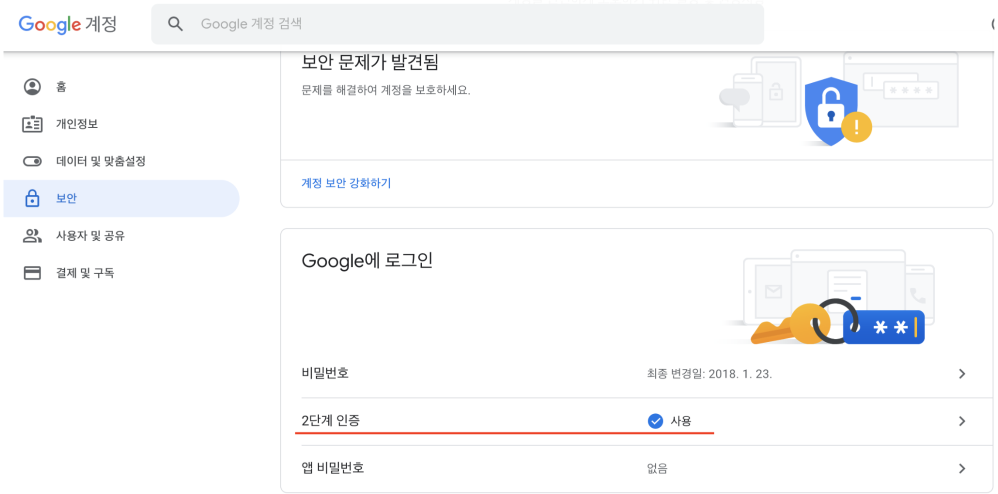
Succeeded 16 hours ago (Wed May 17 2023 00:01:51 GMT+0900)
Author: SadoRuin <dope2514@gmail.com> -
Comment: Update msa-mainserver tag to 12

# III. 외부 서비스

## 1. Google SMTP

- Google 계정 설정

  1. Google 계정 로그인
  2. Google 계정 관리 > 보안 > Google에 로그인 > 2단계 인증 설정



보안 문제가 발견됨

문제를 해결하여 계정을 보호하세요.

계정 보안 강화하기

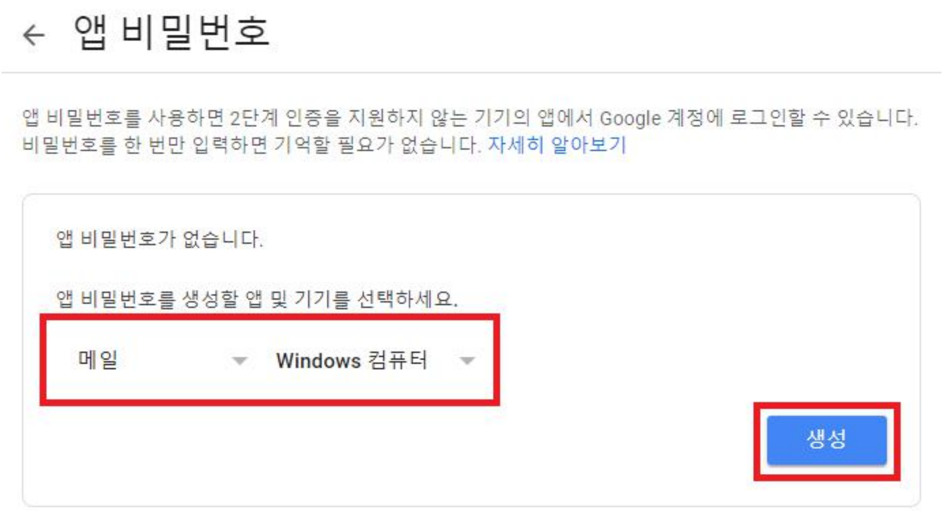
Google에 로그인

비밀번호 최종 변경일: 2018. 1. 23.

2단계 인증 ✓ 사용

앱 비밀번호 없음

  3. Google 계정 관리 > 보안 > Google에 로그인 > 앱 비밀번호 설정



← 앱 비밀번호

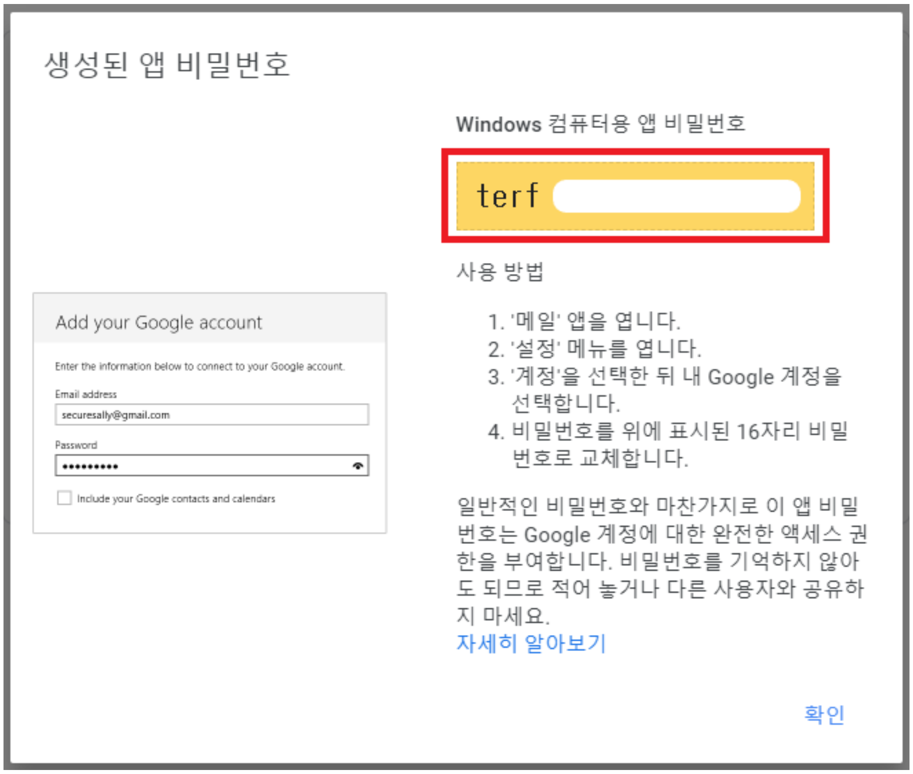
앱 비밀번호를 사용하면 2단계 인증을 지원하지 않는 기기의 앱에서 Google 계정에 로그인할 수 있습니다. 비밀번호를 한 번만 입력하면 기억할 필요가 없습니다. 자세히 알아보기

앱 비밀번호가 없습니다.

앱 비밀번호를 생성할 앱 및 기기를 선택하세요.

메일 ▾ Windows 컴퓨터 ▾

4. 생성된 앱 비밀번호를 복사해 놓는다



생성된 앱 비밀번호

Windows 컴퓨터용 앱 비밀번호

terf [redacted]

사용 방법

1. '메일' 앱을 엽니다.
2. '설정' 메뉴를 엽니다.
3. '계정'을 선택한 뒤 내 Google 계정을 선택합니다.
4. 비밀번호를 위에 표시된 16자리 비밀번호로 교체합니다.

일반적인 비밀번호와 마찬가지로 이 앱 비밀번호는 Google 계정에 대한 완전한 액세스 권한을 부여합니다. 비밀번호를 기억하지 않아도 되므로 적어 놓거나 다른 사용자와 공유하지 마세요.

자세히 알아보기

Add your Google account

Enter the information below to connect to your Google account.

Email address: securesally@gmail.com

Password: •••••••••

□ Include your Google contacts and calendars

- application.yml에 설정 추가 : **I-4 설정 파일 및 환경 변수 정보**의 MainServer 항목에서 application-mainserver.yml에 spring.mail 부분 참조