SSAFY D201

SAI Porting Memual

목차

1 개발 세팅

1.1 C#

- 기본 세팅

목록	버전
Visual Stdio 2022(IDE)	17.13.6
NuGet Package Manager	6.13.2
.NET Framework	v4.7.2
C#	7.3

- 외부 라이브러리

이름	버전
CefSharp.Common	135.0.220
CefSharp.WinForms	135.0.220
chromiumembeddedframework.runtime.win-x64	135.0.220
chromiumembeddedframework.runtime.win-x86	135.0.220
Guna.Charts.WinForms	1.1.0
Guna.UI2.WinForms	2.0.4.7
Clipper2	1.4.0
jacobslusser.ScintillaNET	3.6.3
Microsoft.Web.WebView2	1.0.3240.44
System.Text.Json	9.0.4
Markdig	0.41.1

1.2 Frontend

- 기본 세팅

목록	버전
Visual studio code	1.100.2
Node.js	v22.13.1
pnpm	10.6.2
React	18.3.1

1.3 Backend

- 기본 세팅

목록	버전
IntelliJ IDEA (IDE)	2024.3.1.1
Java	17
Spring boot	3.4.5
Spring Dependency Management	1.1.7
Tomcat	10.1.40

2 EC2

2.1 ufw 방화벽 설정

- 필요한 포트 열기

```
ufw allow 22            #ssh
ufw allow 80            #http
ufw allow 443           #https
ufw allow 8083          #Jenkins
ufw allow 15432         #Postgres
ufw allow 7000          #redis
ufw allow 27555         #mongodb
```

3 Docker

3.1 Docker 설치

```
#패키지 업데이트
sudo apt-get update

#https 관련 패키지 설치
sudo apt install apt-transport-https ca-certificates curl software-
properties-common

#docker repository 접근을 위한 gpg 키 설정
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key
add -

#docker repository 등록
sudo add-apt-repository "deb [arch=amd64]
https://download.docker.com/linux/ubuntu focal stable"

#다시 패키지 업데이트
sudo apt-get update

#도커 설치
sudo apt-get install docker-ce docker-ce-cli containerd.io

#설치 확인
sudo docker –version

#도커로 hello-world 이미지로 테스트
sudo docker run hello-world

#도커 네트워크 생성
sudo docker network create fincatch
```

아래와 같은 메시지가 나오면 성공

```
latest: Pulling from library/hello-world
719385e32844: Pull complete
Digest:
sha256:926fac19d22aa2d60f1a276b66a20eb765fbeea2db5dbdaafeb456ad8ce81598
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working
correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker
Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs
the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which
sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/#다시 패키지 업데이트
```

3.2 Docker Compose 설치

```
#docker compose 설치
sudo curl -SL
https://github.com/docker/compose/releases/download/v2.20.0/docker-
compose-linux-x86_64 -o /usr/local/bin/docker-compose

#설치한 파일에 실행권한 추가
sudo chmod +x /usr/local/bin/docker-compose

#docker-compose 명령어 심볼릭 추가
sudo ln -s /usr/local/bin/docker-compose /usr/bin/docker-compose

#docker-compose 버전 확인
sudo docker-compose -v
```

4 Database

4.1 docker-compose.yml 작성

```
version: "3.8"

services:
  postgres:
    image: postgres:17.4
    container_name: postgres
    restart: unless-stopped
    networks:
      - sai
    ports:
      - "15432:5432"
    environment:
      POSTGRES_USER: {postgres-id}
      POSTGRES_PASSWORD: {postgres-password}
      POSTGRES_DB: sai
    volumes:
      - /home/ubuntu/postgres_data:/var/lib/postgresql/data
      - ./postgres-config:/docker-entrypoint-initdb.d

  redis:
    image: redis:latest
    container_name: redis
    restart: unless-stopped
    networks:
      - sai
    ports:
      - "7000:6379"
    command: ["redis-server", "--bind", "0.0.0.0", "--requirepass",
"{redis-password}"]
    volumes:
      - /home/ubuntu/redis:/data

  mongodb:
    image: mongo:7
    container_name: mongodb
    restart: unless-stopped
    command: ["mongod", "--port", "27017", "--bind_ip_all"]
    environment:
      MONGO_INITDB_ROOT_USERNAME: {mongodb-id}
      MONGO_INITDB_ROOT_PASSWORD: {mongodb-password}
    ports:
      - "27555:27017"
    volumes:
      - /home/ubuntu/mongo_data:/data/db
    networks:
      - sai

networks:
  sai:
    external: true
```

5 Jenkins

5.1 Jenkins 데이터 사전 설정

```
#jenkins volume 폴더 생성
mkdir /home/ubuntu/jenkins-data

#pull jenkins image & run container
sudo docker run -d \
        --name jenkins --user root \
        --network sai \
        -p 8083:8080 -p 60003:60000 \
        -v /home/ubuntu/jenkins-data:/var/jenkins_home \
        -v /var/run/docker.sock:/var/run/docker.sock \
        -e JENKINS_OPTS="--prefix=/jenkins --httpPort=8083" \
        -e JENKINS_URL="http://k12d201.p.ssafy.io:8083/jenkins" \
jenkins/jenkins:lts

# stop Jenkins container
sudo docker stop jenkins

# Jenkins 업데이터 센터 변경
cd /home/ubuntu/jenkins-data
mkdir update-center-rootCAs
wget https://cdn.jsdelivr.net/gh/lework/jenkins-update-center/rootCA/update-
center.crt -O ./update-center-rootCAs/update-center.crt
sudo sed -i 's#https://updates.jenkins.io/update-
center.json#https://raw.githubusercontent.com/lework/jenkins-update-
center/master/updates/tencent/update-
center.json#' ./hudson.model.UpdateCenter.xml
```

5.2 docker-compose.yml 작성

```
version: "3.8"

services:
  jenkins:
    image: jenkins/jenkins:latest
    container_name: jenkins
    restart: unless-stopped
    user: root
    volumes:
      - /home/ubuntu/jenkins-data:/var/jenkins_home
      - /var/run/docker.sock:/var/run/docker.sock
    networks:
      - sai
    environment:
      JENKINS_OPTS: "--prefix=/jenkins"  # Jenkins 접속 경로 설정
      JENKINS_URL: "http://k12d201.p.ssafy.io:8083/jenkins/"  # Jenkins 기본
        #JENKINS_URL: "https://k12d201.p.ssafy.io/jenkins/" #nginx 설정 이후
    entrypoint: ["/bin/sh", "-c", "apt update && apt install -y git docker.io
curl && exec /usr/bin/tini -- /usr/local/bin/jenkins.sh --prefix=/jenkins"]

networks:
  sai:
    external: true
```

5.3 Jenkins config.xml 주의 사항

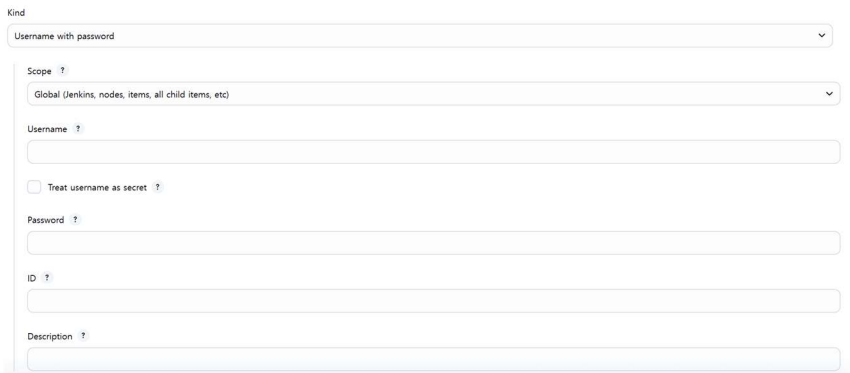
- 아래 명령어 작동 시 true 값이 나와야 함.

```
cat jenkins-data/config.xml | grep useSecurity
<useSecurity>true</useSecurity>
```

5.4 Jenkins 내부 설정

5.4.1 Credentials

- GITLAB (username & password) : Gitlab ID & Gitlab Password 각각 작성 후 Save



- DOCKER_NETWORK (secret text) : docker network 이름

- BACKEND_CONTAINER_NAME (secret text) : backend container 이름

- BACKEND_IMAGE_NAME (secret text) : backend image 이름

- FRONTEND_CONTAINER_NAME (secret text) : frontend container 이름

- FRONTEND_IMAGE_NAME (secret text) : frontend image 이름

- BACKEND_WEBHOOK_URL (secret text) : backend Build & Deploy 후 알림 전송 URL

- FRONTEND_WEBHOOK_URL (secret text) : frontend Build & Deploy 후 알림 전송 URL

- BACKEND_APPLICATION_YAML (secret file) : backend application-secret.yml 업로드

6 Nginx

6.1 docker-compose.yml 작성

```yaml
version: "3.8"

services:
  nginx:
    image: nginx:latest
    container_name: nginx
    restart: always
    networks:
      - sai
    ports:
      - "80:80"
      - "443:443"
    volumes:
      - /home/ubuntu/nginx-data/default.conf:/etc/nginx/conf.d/default.conf
      - /home/ubuntu/nginx-data/nginx.conf:/etc/nginx/nginx.conf
      - /etc/letsencrypt/:/etc/letsencrypt/
      - /etc/nginx/sites-available/:/etc/nginx/sites-available/
      - /etc/nginx/sites-enabled/:/etc/nginx/sites-enabled/

networks:
  sai:
    external: true
```

6.2 nginx.conf 작성

```
user nginx;
worker_processes auto;

error_log /var/log/nginx/error.log warn;
pid /var/run/nginx.pid;

events {
    worker_connections 1024;
}

http {
    include /etc/nginx/mime.types;
    default_type application/octet-stream;

    proxy_headers_hash_max_size 1024;
    proxy_headers_hash_bucket_size 128;

    log_format main '$remote_addr - $remote_user [$time_local] "$request" '
                    '$status $body_bytes_sent "$http_referer" '
                    '"$http_user_agent" "$http_x_forwarded_for"';

    access_log /var/log/nginx/access.log main;

    sendfile on;
    keepalive_timeout 65;

    include /etc/nginx/conf.d/*.conf;
}
```

6.3 default.conf 작성

```
resolver 127.0.0.11 valid=10s ipv6=off;
resolver_timeout 5s;

upstream jenkins {
        zone jenkins_zone 64k;
        server jenkins:8080 resolve;
}

upstream backend {
    zone backend_zone 64k;
    server backend:9091 resolve;
}

upstream frontend {
    zone frontend_zone 64k;
    server frontend:3000 resolve;
}

server {
        listen 80;
        listen [::]:80;
        server_name k12d201.p.ssafy.io;

        location / {
                return 301 https://k12d201.p.ssafy.io$request_uri;
        }
}

server {
        listen 443 ssl;
        listen [::]:443 ssl;
        server_name k12d201.p.ssafy.io;

        ssl_certificate
/etc/letsencrypt/live/k12d201.p.ssafy.io/fullchain.pem;
        ssl_certificate_key
/etc/letsencrypt/live/k12d201.p.ssafy.io/privkey.pem;
        include /etc/letsencrypt/options-ssl-nginx.conf;
        ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem;

        location / {
                proxy_pass http://frontend/;
                proxy_http_version 1.1;
                proxy_set_header Host $host;
                proxy_set_header X-Real-IP $remote_addr;
                proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
                proxy_set_header X-Forwarded-Proto https;
                proxy_set_header X-Forwarded-Host $host;
        }

        location /api/ {
                proxy_pass http://backend/api/;
                proxy_http_version 1.1;
                proxy_set_header Host $host;
```

```
                proxy_set_header X-Real-IP $remote_addr;
                proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
                proxy_set_header X-Forwarded-Proto https;
                proxy_set_header X-Forwarded-Host $host;
        }

        location /jenkins/ {
                proxy_pass http://jenkins;
                proxy_http_version 1.1;

                proxy_set_header Host $host;
                proxy_set_header X-Real-IP $remote_addr;
                proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
                proxy_set_header X-Forwarded-Proto https;
                proxy_set_header X-Forwarded-Host $host;
                proxy_set_header X-Forwarded-Prefix /jenkins;

                proxy_set_header X-Jenkins-Context-Path /jenkins;

                proxy_set_header Upgrade $http_upgrade;
                proxy_set_header Connection "Upgrade";
        }
}
```

7 Jenkins & Gitlab Webhooks 설정

7.1 Jenkins Pipeline 전략

- develop 브랜치에 push 이벤트가 발생했을 때 Webhook을 통해 Jenkins의 root 파이프라인을 실행하고, Webhook payload에 포함된 commit 정보를 분석하여 변경된 최상위 디렉토리(frontend/, backend/)를 기준으로 해당하는 하위 파이프라인(frontend 또는 backend)을 조건적으로 실행하는 구조로 Jenkins CI/CD를 설계

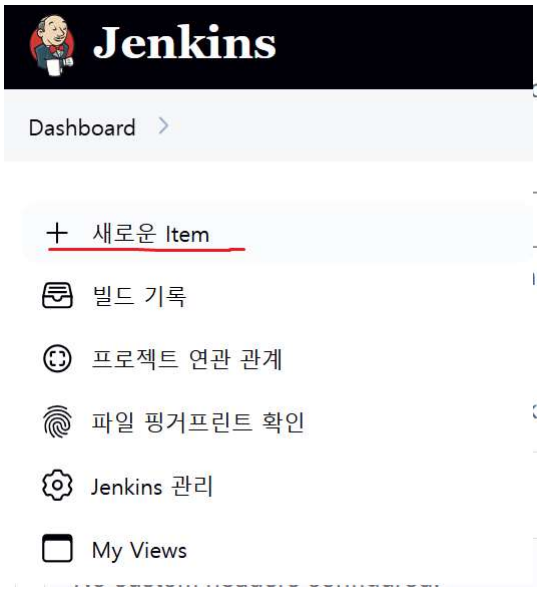
- root Jenkinsfile

```
pipeline {
    agent any

    options {
        disableConcurrentBuilds(abortPrevious: true)
        buildDiscarder(logRotator(numToKeepStr: '10'))
    }

    stages {
        stage('변경 디렉터리 탐색') {
            steps {
                script {
                    def since = env.GIT_PREVIOUS_SUCCESSFUL_COMMIT ?: 'HEAD~1'
                    def diff  = sh(
                        script: "git diff --name-only ${since}
${env.GIT_COMMIT}",
                        returnStdout: true
                    ).trim().split('\n')

                    changedTopDirs = diff.findAll { it.contains('/') }
                                         .collect  { it.tokenize('/')[0] }
                                         .unique()

                    echo "변경된 최상위 폴더: ${changedTopDirs}"
                }
            }
        }

        stage('Child Jobs Fan-out') {
            steps {
                script {
                    def fanout = [:]

                    if (changedTopDirs.contains('frontend')) {
                        fanout['frontend'] = {
                            build job: 'frontend',
                                  wait: false,
                                  propagate: false,
                                  parameters: [
                                      string(name: 'GIT_SHA', value:
env.GIT_COMMIT.take(7))
                                  ]
```

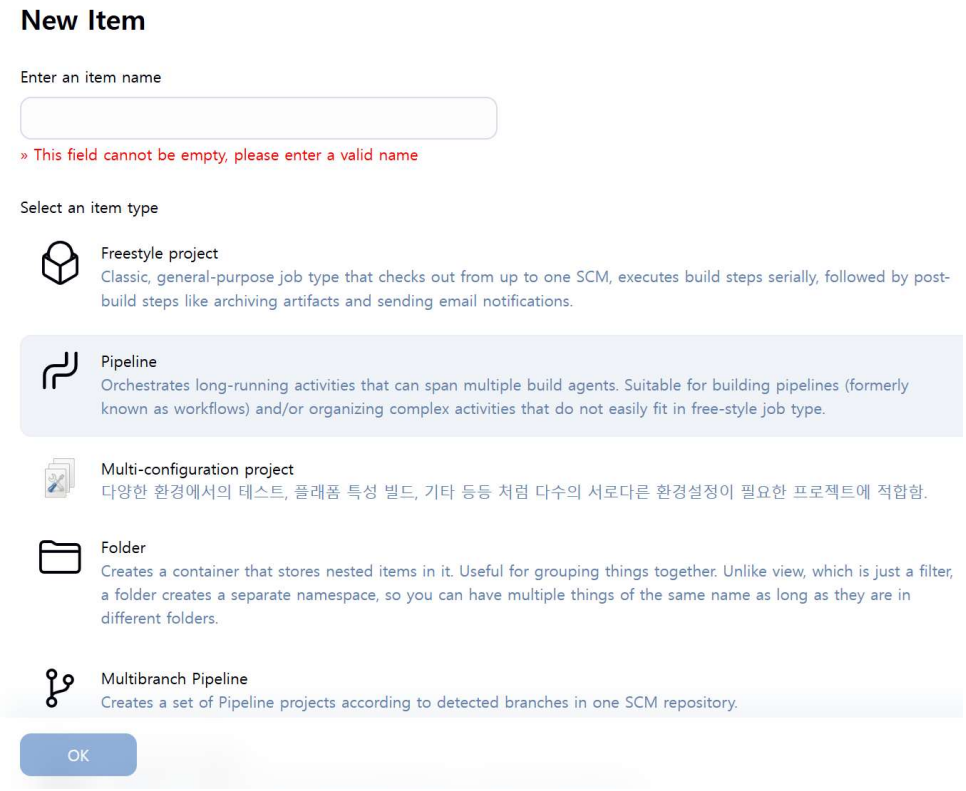
```
        }
                    }

                    if (changedTopDirs.contains('backend')) {
                        fanout['backend'] = {
                            build job: 'backend',
                                  wait: false,
                                  propagate: false,
                                  parameters: [
                                      string(name: 'GIT_SHA', value:
env.GIT_COMMIT.take(7))
                                  ]
                        }
                    }

                    if (fanout.isEmpty()) {
                        echo '변경된 디렉터리가 없습니다 - 빌드 종료'
                    } else {
                        parallel fanout
                    }
                }
            }
        }
    }

    post {
        success { echo 'Root 파이프라인 정상 종료 (Child Jobs 비동기 실행 중)' }
        failure { echo 'Root 파이프라인 실패 - 콘솔 로그 확인' }
    }
}
```

7.2 Jenkins Pipeline 생성

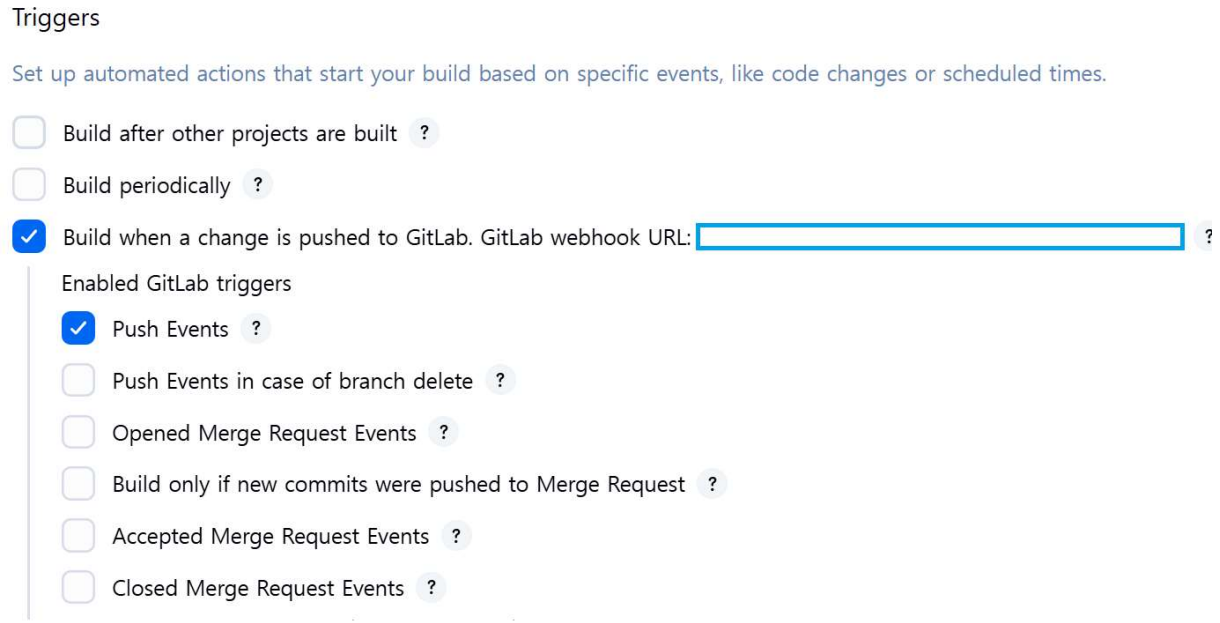
1. 새로운 item 클릭



2. Item 이름 작성 > Pipeline 선택 > OK 클릭 (root, backend, frontend 총 3개 생성)

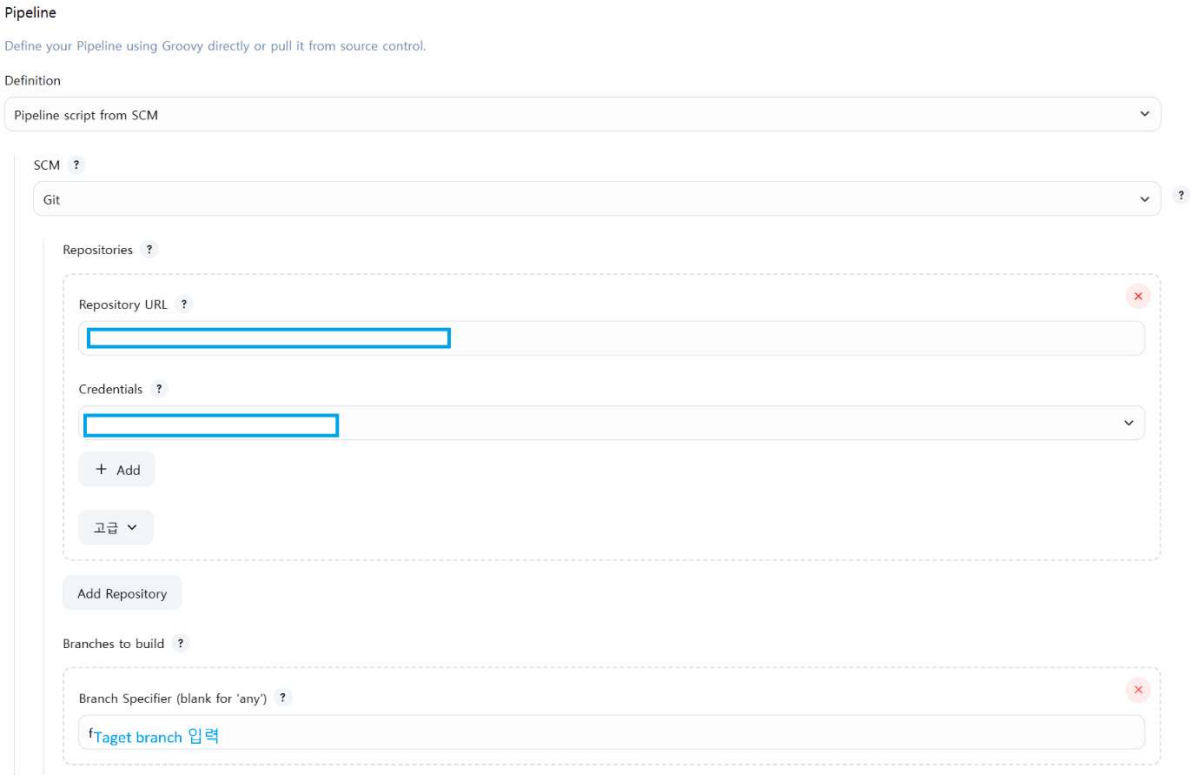


3. Triggers > Build when a change is pushed to Gitlab 클릭 > URL 복사 > Push Event 선택



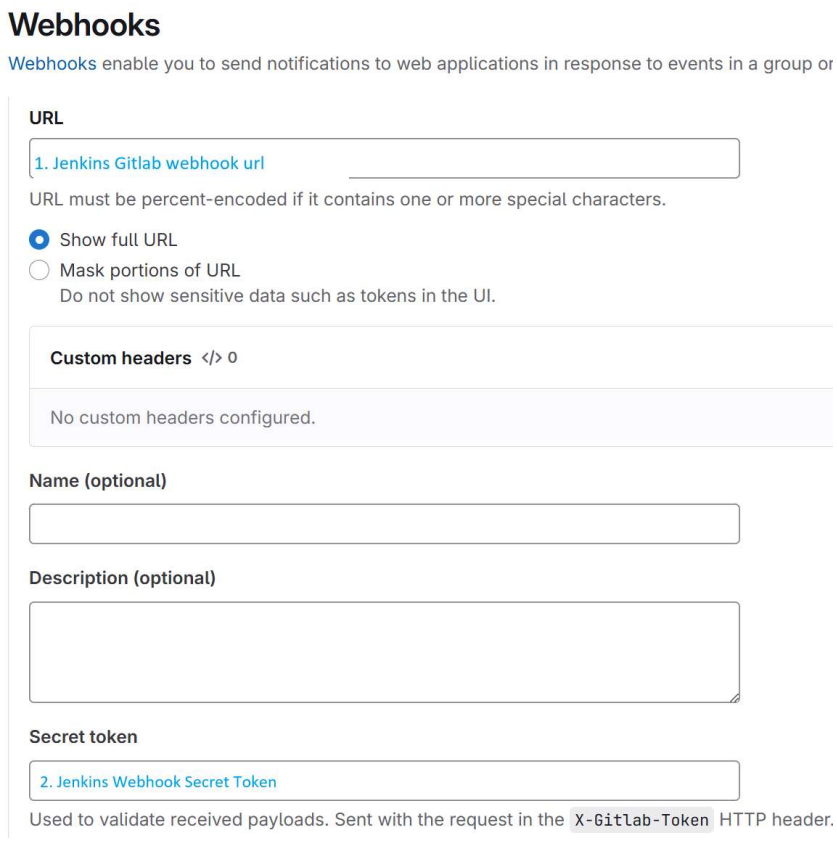
4. 고급 클릭 > Secret token Generate 후 복사

5. Pipeline Definition 설정 > Pipeline script from SCM

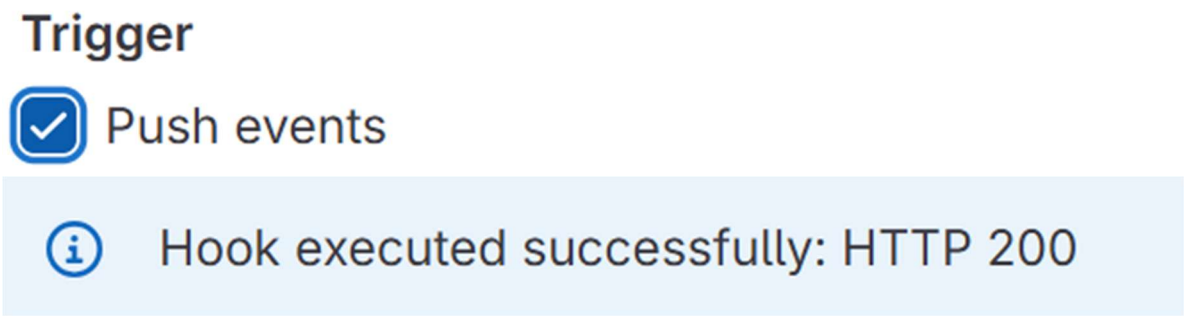


7.3 Gitlab Webhook 생성 및 Jenkins Pipeline 연결

1. Project > Settgins > Webhooks > Add new webhook



2. Push event 선택 > Add webhook > Send Test push evnet



- Root pipeline 만 webhook 연결

8 Backend Build & Deploy

8.1 build.gradle

```
plugins {
    id 'java'
    id 'org.springframework.boot' version '3.4.5'
    id 'io.spring.dependency-management' version '1.1.7'
}

group = 'com.sai.backend'
version = '0.0.1-SNAPSHOT'

java {
    toolchain {
        languageVersion = JavaLanguageVersion.of(17)
    }
}

configurations {
    compileOnly {
        extendsFrom annotationProcessor
    }
}

repositories {
    mavenCentral()
}

dependencies {
    //spring-boot develop
    implementation 'org.springframework.boot:spring-boot-starter-actuator'
    implementation 'org.springframework.boot:spring-boot-starter-thymeleaf'
    implementation 'org.springframework.boot:spring-boot-starter-web'
    implementation 'org.springframework.boot:spring-boot-starter-websocket'
    implementation 'org.springdoc:springdoc-openapi-starter-webmvc-ui:2.8.5'
    developmentOnly 'org.springframework.boot:spring-boot-devtools'
    implementation 'org.springframework.boot:spring-boot-starter-webflux'
    compileOnly 'org.projectlombok:lombok'
    annotationProcessor 'org.projectlombok:lombok'

    //security
    implementation 'org.springframework.boot:spring-boot-starter-security'
    implementation 'org.thymeleaf.extras:thymeleaf-extras-springsecurity6'
    implementation 'org.springframework.security:spring-security-messaging'

    //DB
    implementation 'org.springframework.boot:spring-boot-starter-data-jpa'
    implementation 'org.springframework.boot:spring-boot-starter-data-redis'
    implementation 'org.hibernate.validator:hibernate-validator:8.0.1.Final'
    implementation 'jakarta.validation:jakarta.validation-api:3.0.2'
    runtimeOnly 'org.postgresql:postgresql'
    implementation 'org.springframework.boot:spring-boot-starter-data-mongodb'
```

```
    // AWS
    implementation("software.amazon.awssdk:s3:2.31.33")
    implementation 'org.springframework.cloud:spring-cloud-starter-
aws:2.2.6.RELEASE'

    //openai
    implementation 'commons-io:commons-io:2.15.1'
    implementation 'org.springframework.boot:spring-boot-starter-webflux'
    implementation 'org.springframework.boot:spring-boot-starter-validation'

}

tasks.named('test') {
    useJUnitPlatform()
}
```

8.2 Jenkinfile

```
pipeline {
    agent any

    parameters {
        string(name: 'GIT_SHA', defaultValue: 'latest', description: 'Git
Commit Short SHA')
    }

    tools {
        jdk 'JDK17'
    }

    options {
        disableConcurrentBuilds()
        buildDiscarder(logRotator(numToKeepStr: '20'))
    }

    stages {
        stage('Checkout SCM') {
            steps {
                checkout scm
            }
        }

        stage('Load Credentials') {
            steps {
                withCredentials([file(credentialsId:
'BACKEND_APPLICATION_SECRET', variable: 'SECRET_FILE')]) {
                    sh 'cp "$SECRET_FILE" ./backend/src/main/resources/'
                }
            }
        }

        stage('Build with Gradle') {
            steps {
                dir('backend') {
                    sh 'chmod +x gradlew'
```

```
                    sh './gradlew clean build -x test'
                }
            }
        }

        stage('Build Docker Image') {
            steps {
                dir('backend') {
                    withCredentials([string(credentialsId: 'BACKEND_IMAGE_NAME',
variable: 'BACKEND_IMAGE_NAME')]) {
                        sh """
                            docker build -t ${BACKEND_IMAGE_NAME}:${GIT_SHA} .
                        """
                    }
                }
            }
        }

        stage('Deploy Backend') {
            steps {
                withCredentials([
                    string(credentialsId: 'BACKEND_IMAGE_NAME', variable:
'BACKEND_IMAGE_NAME'),
                    string(credentialsId: 'BACKEND_CONTAINER_NAME', variable:
'BACKEND_CONTAINER_NAME'),
                    string(credentialsId: 'DOCKER_NETWORK', variable:
'DOCKER_NETWORK')
                ]) {
                    sh """
                        docker stop ${BACKEND_CONTAINER_NAME} || true
                        docker rm ${BACKEND_CONTAINER_NAME} || true

                        docker run -d --name ${BACKEND_CONTAINER_NAME} --network
${DOCKER_NETWORK} \\
                            -p 9097:9091 \\
                            -e TZ=Asia/Seoul \\
                            -e JAVA_TOOL_OPTIONS="-Duser.timezone=Asia/Seoul" \\
                            -e SPRING_PROFILES_ACTIVE=prod \\
                            ${BACKEND_IMAGE_NAME}:${GIT_SHA}
                    """
                }
            }
        }
    }

    post {
        always {
           sh 'rm -f ./backend/src/main/resources/application-secret.yml'
        }
        success {
            script {
                withCredentials([string(credentialsId: 'BACKEND_WEBHOOK_URL',
variable: 'BACKEND_MM_WEBHOOK_URL')]) {
                    def jsonMessage = """{
                        "attachments": [{
                            "text": "**✅ Backend Build 성공**\\\\n- 상태:
```

```
                        "text": "**✅ Backend Build 성공**\\\\n- 상태:
SUCCESS\\\\n- [🔗 상세 정보](${env.BUILD_URL})",
                        "color": "#00FF00"
                    }]
                }"""

                sh """
                echo '${jsonMessage}' > mattermost_payload.json
                cat mattermost_payload.json
                curl -X POST -H "Content-Type: application/json" --data
@mattermost_payload.json '${BACKEND_MM_WEBHOOK_URL}'
                rm -f mattermost_payload.json
                """
            }
        }
    }
    failure {
        script {
            withCredentials([string(credentialsId: 'BACKEND_WEBHOOK_URL',
variable: 'BACKEND_MM_WEBHOOK_URL')]) {
                def jsonMessage = """{
                    "attachments": [{
                        "text": "**❌ Backend Build 실패**\\\\n- 상태:
FAILURE\\\\n- [🔗 상세 정보](${env.BUILD_URL}/console) ",
                        "color": "#FF0000"
                    }]
                }"""

                sh """
                echo '${jsonMessage}' > mattermost_payload.json
                cat mattermost_payload.json
                curl -X POST -H "Content-Type: application/json" --data
@mattermost_payload.json '${BACKEND_MM_WEBHOOK_URL}'
                rm -f mattermost_payload.json
                """
            }
        }
    }
}
```

8.3 Dockerfile

```
FROM openjdk:17
ARG JAR_FILE=build/libs/backend-0.0.1-SNAPSHOT.jar
ADD ${JAR_FILE} app.jar
EXPOSE 9091
ENTRYPOINT ["java", "-Duser.timezone=Asia/Seoul", "-jar", "/app.jar"]
```

8.4 application-secret.yml

```yaml
spring:
  datasource:
    url: jdbc:postgresql{postgresql_url:port}/{database_name}
    username: {postgres_id}
    password: {postgres_password}
    driver-class-name: org.postgresql.Driver

  data:
    redis:
      host: {redis_host}
      port: {redis_port}
      password: {redis_password}

    mongodb:
      uri: mongodb://{mongodb_id}:{mongodb_password}@{mongodb_url:port}/sai-
mongo?authSource=admin

cloud:
  aws:
    stack:
      auto: false
    s3:
      bucket: sai-comhaha-d201
    region:
      static: ap-northeast-2
    credentials:
      access-key: {AWS_IAM_ACCESS_KEY}
      secret-key: {AWS_IAM_SERECT_KEY}

openai:
  api-url: {OPENAI_API_URL}
  api-key: {OPENAI_API_KEY}
  model: gpt-4.1

notion:
  client:
    id: {NOTION_API_CLIENT_ID}
    secret: {NOTION_API_SECRET_KEY}
  redirect:
    uri: {NOTION_API_REDIRECT_URI}
  api:
    url: https://api.notion.com
```

9 Frontend Build & Deploy

9.1 Jenkinfile

```
pipeline {
    agent any

    parameters {
        string(name: 'GIT_SHA', defaultValue: 'latest', description: 'Git
Commit Short SHA')
    }

    options {
        disableConcurrentBuilds()
        buildDiscarder(logRotator(numToKeepStr: '20'))
    }

    stages {
        stage('Checkout SCM') {
            steps {
                script {
                    checkout scm
                }
            }
        }

        stage('Build Docker Image') {
            steps {
                script{
                    withCredentials([string(credentialsId:
'FRONTEND_IMAGE_NAME', variable: 'FRONTEND_IMAGE_NAME')]) {
                        dir('frontend') {
                           sh """
                                docker build -t
${FRONTEND_IMAGE_NAME}:${GIT_SHA} .
                            """
                        }
                    }
                }
            }
        }

        stage('Deploy Frontend') {
            steps {
                script {
                    withCredentials([
                        string(credentialsId: 'FRONTEND_IMAGE_NAME', variable:
'FRONTEND_IMAGE_NAME'),
                        string(credentialsId: 'FRONTEND_CONTAINER_NAME',
variable: 'FRONTEND_CONTAINER_NAME'),
                        string(credentialsId: 'DOCKER_NETWORK', variable:
'DOCKER_NETWORK')
                    ]) {
                        sh """
                            docker stop ${FRONTEND_CONTAINER_NAME} | true
```

```
                        docker rm ${FRONTEND_CONTAINER_NAME} | true

                        docker run -d --name ${FRONTEND_CONTAINER_NAME} \
                            --network ${DOCKER_NETWORK} -p 3001:3000 \
                        ${FRONTEND_IMAGE_NAME}:${GIT_SHA}
                    """
                }
            }
        }
    }

    post {
        success {
            script {
                withCredentials([string(credentialsId: 'FRONTEND_WEBHOOK_URL',
variable: 'FRONTEND_MM_WEBHOOK_URL')]) {
                    def jsonMessage = """{
                        "attachments": [{
                            "text": "**✅ Frontend Build 성공**\\\\n- 상태:
SUCCESS\\\\n- [🔗 상세 정보](${env.BUILD_URL})",
                            "color": "#00FF00"
                        }]
                    }"""

                    sh """
                    echo '${jsonMessage}' > mattermost_payload.json
                    cat mattermost_payload.json
                    curl -X POST -H "Content-Type: application/json" --data
@mattermost_payload.json '${FRONTEND_MM_WEBHOOK_URL}'
                    rm -f mattermost_payload.json
                    """
                }
            }
        }
        failure {
            script {
                withCredentials([string(credentialsId: 'FRONTEND_WEBHOOK_URL',
variable: 'FRONTEND_MM_WEBHOOK_URL')]) {
                    def jsonMessage = """{
                        "attachments": [{
                            "text": "**❌ Frontend Build 실패**\\\\n- 상태:
FAILURE\\\\n- [🔗 상세 정보](${env.BUILD_URL}/console) ",
                            "color": "#FF0000"
                        }]
                    }"""

                    sh """
                    echo '${jsonMessage}' > mattermost_payload.json
                    cat mattermost_payload.json
                    curl -X POST -H "Content-Type: application/json" --data
@mattermost_payload.json '${FRONTEND_MM_WEBHOOK_URL}'
                    rm -f mattermost_payload.json
                    """
                }
```

```
            }
        }
    }
}
```

9.2 Dockerfile

```
FROM node:22 AS builder
WORKDIR /app
COPY package*.json ./
RUN pnpm install
COPY . .
RUN pnpm run build
FROM nginx:latest
COPY nginx.conf /etc/nginx/conf.d/default.conf
COPY --from=builder /app/dist /usr/share/nginx/html
EXPOSE 3000
CMD ["nginx", "-g", "daemon off;"]
```

9.3 nginx.conf

```
server {
    listen 3000;
    server_name k12d108.p.ssafy.io;
    root /usr/share/nginx/html;
    index index.html;

    location / {
        try_files $uri /index.html;
    }
    error_page 404 /index.html;
}
```

10 C# Build & Deploy

- Inno setup 프로그램을 활용하여 설치프로그램 제작

10.1 Inno setup 설치

- https://jrsoftware.org/isdl.php 해당 사이트에서 최신 버전 innosetup-{version}.exe 파일 설치

Filename	Download Sites
innosetup-6.4.3.exe	US Netherlands

10.2 .iss 파일 작성

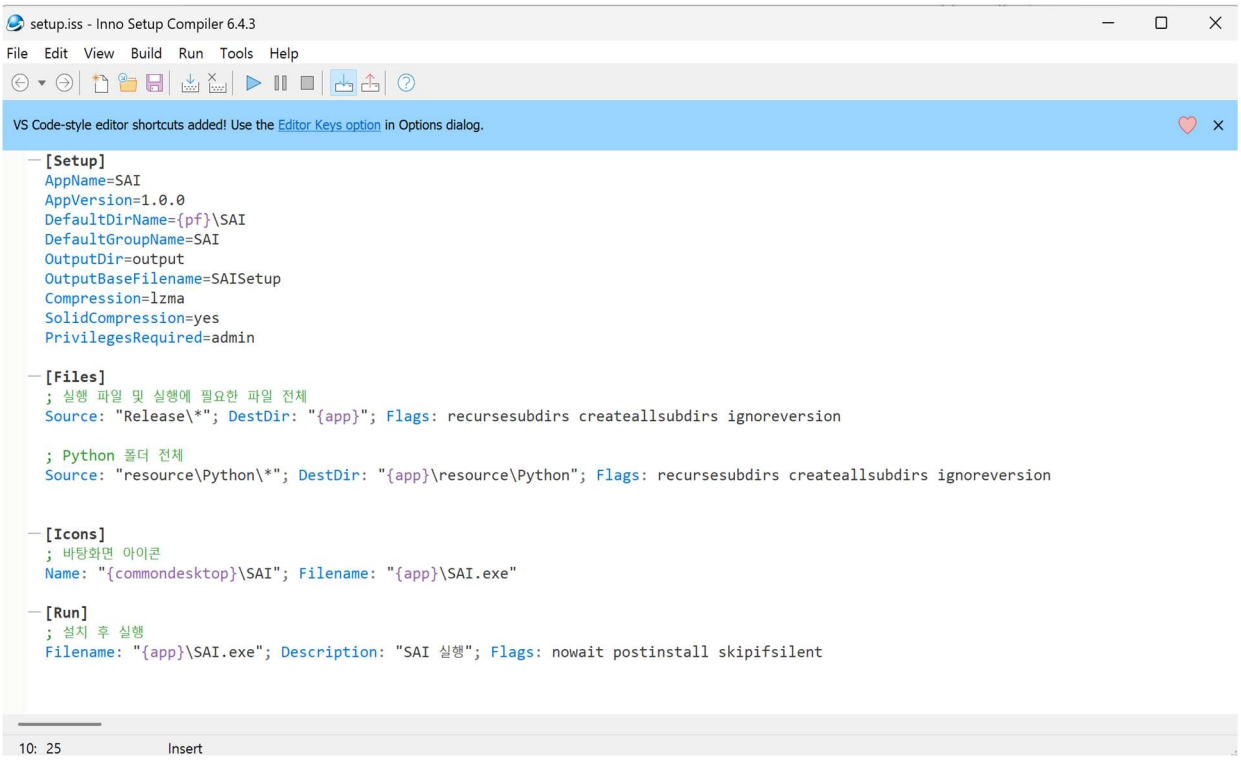
```
[Setup]
AppName=SAI
AppVersion=1.0.0
AppPublisher=SSAFY D201
AppCopyright=© 2025 SSAFY D201
DefaultDirName={pf}\SAI
DefaultGroupName=SAI
DisableDirPage=yes
OutputDir=output
OutputBaseFilename=SAISetup
Compression=lzma
SolidCompression=yes
PrivilegesRequired=admin
SetupIconFile=resource\icon\SAI.ico


[Files]
; 실행 파일 및 실행에 필요한 파일 전체
Source: "Release\*"; DestDir: "{app}"; Flags: recursesubdirs createallsubdirs
ignoreversion

; resource 폴더 전체
Source: "resource\*"; DestDir: "{app}\resource"; Flags: recursesubdirs
createallsubdirs ignoreversion


[Icons]
; 바탕화면 아이콘
Name: "{commondesktop}\SAI"; Filename: "{app}\SAI.exe"; IconFilename:
"{app}\resource\icon\SAI.ico"
Name: "{group}\SAI"; Filename: "{app}\SAI.exe"; IconFilename:
"{app}\resource\icon\SAI.ico"


[Run]
; 설치 후 실행
Filename: "{app}\SAI.exe"; Description: "SAI 실행"; Flags: nowait postinstall
skipifsilent
```

10.3 폴더 구조 설계

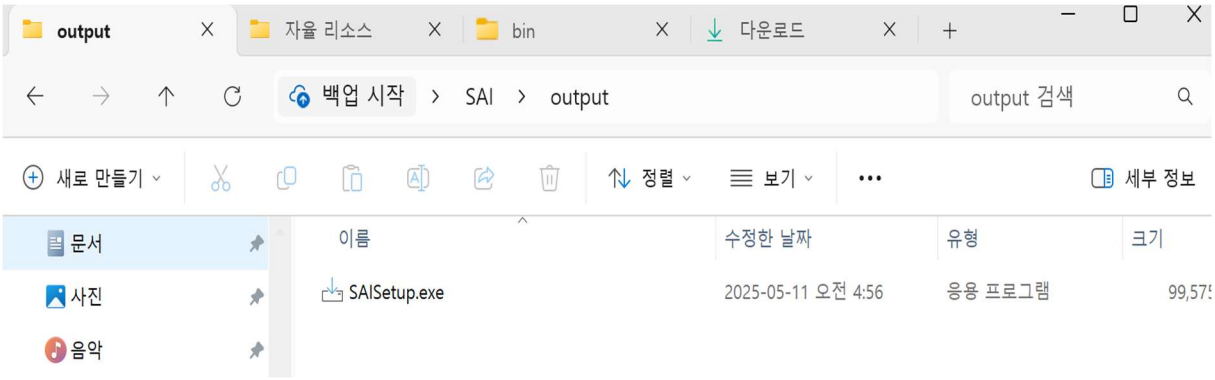
```
SAI/
 ├─── Release/
 │   ├─── SAI.exe
 │   ├─── *.dll
 │   ├─── SAI.exe.config
 ├─── resource/
 │   ├─── icon/
 │   └─── Python/
 ├─── setup.iss
```

10.4 SAI 설치 프로그램 생성

- Ctrl + F9로 Compile



- SAIsetup.exe 생성



10.5 SAIsetup.exe S3에 업로드