# **Podding Manual**

## **1. 환경 설정**

### **실행 환경 및 사용 툴**

**1) 실행 환경**

1. **로컬 환경**

각종 IDE를 통한 작업과 문서 작성을 위한 개발환경은 Ubuntu 22.04 Jammy 위에서 이루어졌다. IntelliJ, Webstorm, VSCode, Jupyter Notebook, Git 등의 협업과 개발을 위한 툴을 Linux 버전으로 사용하였다.

1. **배포 환경**

배포환경에 사용한 AWS EC2 인스턴스 이미지는 Ubuntu 22.04 Jammy 을 사용하였다.

1. **Docker**

Hadoop, Jupyter Notebook Spring Boot, Nginx, Jenkins 어플리케이션을 담은 컨테이너는 다음과 같은 기본 이미지에서 빌드하였다.

|  |  |  |
| --- | --- | --- |
| 어플리케이션 | 이미지 | 비고 |
| Spring Boot | ubuntu:22.04 | Dockerfile 빌드 |
| Nginx | ubuntu:22.04 | Dockerfile 빌드 |
| Hadoop | ubuntu:22.04 | Dockerfile 빌드 |
| Jupyter Lab | dorowu/ubuntu-desktop-lxde-vnc | Dockerfile 빌드 |
| Jenkins | jenkins:lts-jdk11 | Dockerfile 빌드 |

표 1. 어플리케이션 별 사용 이미지

1. **Java**

Spring Boot JAR 아카이브 실행을 위해 JRE를 포함하는 JDK는 Zulu JDK를 사용하였다. Zulu-8.68.0.21-OpenJDK-8.0.342 버전을 JRE가 필요한 Spring Boot 컨테이너에 설치하여 개발와 배포를 진행하였다.

이와는 별개로, Jenkins의 컨테이너의 이미지는 JDK-11 버전을 제공한다.

1. **NodeJS**

Vue 어플리케이션 빌드를 위해 NodeJS 19.6.0 버전과 NPM 9.4.0 버전을 NginX 컨테이너에 설치하였다.

1. **Python**

PySpark 스크립트 개발과 Airflow DAG 개발을 위해 Python 3.9.5 버전을 사용하였다.

**2) 툴**

1. **IDE**

개발에 사용한 IDE와 버전정보는 다음과 같다.

|  |  |  |
| --- | --- | --- |
| 용도 | 버전 | 빌드 |
| Spring Boot 개발  Hive DB 관리(부) | JetBrains IntelliJ IDEA 2022.3.1 | #IU-223.8214.52 |
| Vue 개발(정) | Microsoft Visual Studio Code 1.75.1 |  |
| Vue 개발(부) | JetBrains WebStorm 2022.3.1 | #WS-223.8214.51 |
| Hive DB 관리(정) | JetBrains DataGrip 2022.3.2 | #DB-223.8214.62 |

표 2. 용도 별 사용 IDE

1. **Spring Boot**

Spring Boot 2.7.8 버전과 Spring Boot Data, Spring Security, Spring Boot Web, Spring Kafka Integration 등 각종 의존성을 추가하여 개발하였다. 멀티모듈 구조의 프로젝트이며 mata-core, mata-api-server, mata-mysql, mata-hive의 모듈을 가진다. 본 프로젝트 내의 pom.xml 파일을 참고하여 추가한 의존성을 확인할 수 있다.

1. **Maven**

의존성 관리를 위해 Maven 3.9.0 버전을 사용하였다. Spring Boot 애플리케이션과 같은 컨테이너 내에서 빌드가 이루어진다.

1. **Nginx**

Vue 빌드 후 생성된 정적파일을 제공하기 위해 Nginx 1.18.0 버전의 서버를 사용하였다. 프로젝트 내의 myapp.conf 설정파일을 통해 세부 설정을 확인할 수 있다.

1. **React**

웹 로그 수집 테스트를 위해 React 18.2.0 버전을 사용하였다. react-app 디렉토리의package.json 파일을 통해 세부 버전을 확인할 수 있다.

1. **Vue**

대시보드 웹서비스 개발을 위해 Vue 3.2.13 버전을 사용하였다. vue3 디렉토리의package-lock.json 파일을 통해 세부 버전을 확인할 수 있다.

1. **Jenkins**

GitLab과 연동을 통한 자동배포는 Jenkins 를 통해 이루어졌다. Jenkins 2.375.2 버전을 사용하여 이를 진행하였다.

1. **Docker Compose**

Jenkin 를 통해 컨테이너 오케스트레이션을 진행하기 위해서 docker-compose를 설치하였다. Docker Compose 2.15.1 버전을 사용하였다.

1. **Hadoop**

총 5개의 컨테이너로 이루어진 데이터 클러스터에 사용될 이미지엔 Hadoop이 기반된다. Hadoop 3.3.4 버전을 사용하였다.

1. **Zookeeper**

Hadoop 클러스터의 고가용성을 유지하기 위해 Zookeeper로 오케스트레이션을 진행하며 Zookeeper Failover Controller 서비스를 구동하였다. Zookeeper 3.7.1 버전을 사용하였다. Hadoop-Zookeeper 설정은 hadoop-cluster/lib/hadoop-3.3.4 디렉토리의 설정파일들인 \*-site.xml 내에서 확인할 수 있다.

1. **Hive**

Data warehouse 구축을 위해 Hadoop 클러스터상의 RDBMS인 Hive 3.1.3 버전을 사용하였다. Hive 설정과 Hiveserver2 설정은 hadoop-cluster/lib/apache-hive-3.1.3-bin 디렉토리에서 확인할 수 있다.

1. **Kafka**

실시간 데이터 수집을 위해 메시징 큐인 Kafka 3.4.0 버전을 사용하였다. Scala 2.12 버전임을 유의하여 설치하였다. Kafka의 설정을 hadoop-cluster/lib/kafka\_2.12-3.4.0 디렉토리에서 확인할 수 있다.

1. **Spark**

ETL 파이프라인 구축을 위해 Spark 3.3.2 버전을 사용하였다. Hadoop과 Integration을 위해 Hadoop 3.x 버전을 지원하는 Binary 파일을 사용하였다. Spark의 설정은 hadoop-cluster/lib/spark-3.3.2-bin-hadoop3 디렉토리에서 확인할 수 있다.

1. **Cassandra**

변환된 스트리밍 데이터를 적재하기 위해 Cassandra 4.0.8 버전을 사용하였다. Cassandra의 설정은 hadoop-cluster/lib/apache-cassandra-4.0.8 디렉토리에서 확인할 수 있다.

1. **RabbitMQ**

Airflow의 병렬적 작업을 위한 Celery Executer의 Job queue를 위해 RabbitMQ 3.10.9 버전을 사용하였다. V-host 등의 설정은 hadoop-cluster/run.sh 스크립트에서 확인할 수 있다.

1. **Airflow**

주기적인 ETL 배칭 작업 관리를 위해 Airflow 2.5.0 버전을 사용하였다. Airflow의 설정은 hadoop-cluster/lib/apache-airflow-2.5.0 디렉토리에서 확인할 수 있으며, 운영에 사용된 DAG 스크립트는 hadoop-cluster/lib/apache-airflow-2.5.0/dags 디렉토리에서 확인할 수있다. Airflow의 사용자 설정은 hadoop-cluster/run.sh 스크립트에서 확인할 수 있다.

1. **Ubuntu Desktop with VCN**

운영환경 모니터링과 Jupyter 개발환경을 위해 SSH 터널링을 통한 VCN 서버를 구동하였다. dorowu/ubuntu-desktop-lxde-vnc 이미지를 통해 구축하였으며, Tiger VNC 클라이언트를 통해 접근하였다.

1. **Jupyter Lab**

ETL 파이프라인 개발을 위해 Jupyter Lab 1.9 버전을 사용하였다.

1. **Redis**

클라이언트 토큰 검증의 성능을 향상시키기 위해 인메모리 데이터베이스인 Redis를 사용하였다. Spring Boot 어플리케이션과 동일한 컨테이너에서 실행된다.

### **환경변수**

**1)** **Development**

1. **Backend**

spring.profiles.active=dev

배포환경과 개발환경 분리를 위해 Spring Boot의 application.yml 프로필을 분리하였다. java –jar -Dspring.profiles.active=dev 옵션을 통해 프로필을 활성화시킬 수 있다.

1. **Frontend**

프로젝트 내의 .env 파일에서 Vue의 환경변수를 확인 가능하다.

VUE\_APP\_API\_HOST=http://ec2-3-38-85-143.ap-northeast-2.compute.amazonaws.com

API 서버 경로이다.

1. **Data cluster**

Dockerfile과 각종 설정 파일에 명시되어있다.

**2)** **Deployment**

1. **Backend**

spring.profiles.active=stage

배포환경과 개발환경 분리를 위해 Spring Boot의 application.yml 프로필을 분리하였다. java –jar -Dspring.profiles.active=stage 옵션을 통해 프로필을 활성화시킬 수 있다.

1. **Frontend**

VUE\_APP\_API\_HOST=http://ec2-3-38-85-143.ap-northeast-2.compute.amazonaws.com

API 서버 경로이다.

1. **Data cluster**

Dockerfile과 각종 설정 파일에 명시되어있다.

### **배포 환경**

**1) 컨테이너**

1. **Spring Boot**

spring\_app 컨테이너를 빌드하기 위한 Dockerfile은 bigdata-api 디렉토리에서 확인할 수 있다. Ubuntu 22.04 이미지에 Maven과 Redis를 설치하여 어플리케이션 빌드와 실행을 수행하도록 하였다.

1. **Nginx**

vue\_app 컨테이너를 빌드하기 위한 Dockerfile은 vue3 디렉토리에서 확인할 수 있다. Ubuntu 22.04 이미지에 Nginx와 NodeJS를 설치하여 spring\_app의 Spring Boot 어플리케이션에 대한 리버스 프록시, Vue 어플리케이션 빌드와 배포를 수행하도록 하였다.

1. **Jenkins**

jenkins\_container 컨테이너를 빌드하기 위한 Dockerfile은 Jenkins 디렉토리에서 확인할 수 있다. jenkins/jenkins:lts-jdk11 이미지에 Docker Engine과 Docker Compose를 설치하여 배포에 사용되는 spring\_app, vue\_app 컨테이너를 관리할 수 있도록 설정하였다.

1. **Data cluster node**

데이터 클러스터의 각 노드를 빌드하기 위한 Dockerfile은 hadoop-cluster/node 디렉토리에서 확인할 수 있다. Hadoop, Zookeeper, Hive 등의 툴을 설치하였으며 각종 설정파일을 추가하였다. 설정파일의 전체는 hadoop-cluster/lib 디렉토리에서 확인할 수 있으며, 노드 별 각 설정은 lib/\*/sbin 디렉토리 내 스크립트 파일에서 수행되도록 설정하였다.

1. **Data cluster client**

데이터 클러스터의 고립된 네트워크에 접근하여 작업을 수행하기 위한 client의 Dockerfile은 hadoop-cluster/client 디렉토리에서 확인할 수 있다. Jupyter Lab, Hadoop, Spark 등의 툴을 설치하였으며 각종 설정파일을 추가하였다.

**2) 네트워크**

hadoop-cluster/docker-compose.yml 설정파일 내에서 cluster-net 도커 네트워크를 생성하여 docker-compose로 올라간 컨테이너들을 접합시켰다. default driver를 사용하였고, 172.16.238.0/24 의 서브넷 범위를 가진다. 이 네트워크에는 node 컨테이너 5개, MySQL 기반 metastore 컨테이너 1개, client 컨테이너 1개, spring\_app 컨테이너, vue\_app 컨테이너, jenkins\_container 컨테이너의 총 10개의 컨테이너가 포함된다.

### **주요 계정 목록**

|  |  |  |  |
| --- | --- | --- | --- |
| 분류 | 접근 방법 | 계정명 | 비밀번호 |
| Jenkins | http://ec2-3-38-85-143.ap-northeast-2.compute.amazonaws.com:9090/ | admin | Cucumber52 |

표 3. 운영 시 주요 계정 목록

## **2. 배포 매뉴얼**

### **1) 호스트 머신 세팅**

1. **기본 유틸리티 설치**

호스트 머신에서 사용할 기본적인 유틸리티를 설치한다.

sudo apt install vim -y

sudo apt install curl –y

sudo apt install net-tools -y

코드 1. Jenkins 컨테이너의 실행 스크립트

1. **방화벽 설정**

호스트 머신에서 개방할 포트는 80, 443, 9090 포트이다. 각 포트의 사용 용도는 다음과 같다.

|  |  |
| --- | --- |
| 포트번호 | 용도 |
| 80 | HTTP |
| 443 | HTTPS |
| 9090 | Jenkins WEB UI |

표 4. Jenkins 컨테이너의 실행 스크립트

sudo ufw allow 80

sudo ufw allow 443

sudo ufw allow 9090

sudo ufw enable

코드 2. UFW 포트 개방 명령어

1. **Docker 설치**

호스트 머신에 Docker 엔진을 설치한다. 이는 Jenkins에 설치 할 Docker와 자원을 공유한다.

apt-get update

apt-get install ca-certificates curl gnupg lsb-release -y

mkdir -p /etc/apt/keyrings

curl -fsSL https://download.docker.com/linux/debian/gpg | gpg --dearmor -o /etc/apt/keyrings/docker.gpg

echo \

"deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux/debian \

$(lsb\_release -cs) stable" | tee /etc/apt/sources.list.d/docker.list > /dev/null

apt-get update

apt-get install docker-ce docker-ce-cli containerd.io docker-compose-plugin -y

코드 코드 3. Docker 설치 명령어

1. **호스트 설정**

컨테이너의 IP 주소를 hosts 파일에 매핑한다. /etc/hosts 파일을 관리자 권한으로 수정하여 매핑할 수 있다.

sudo vim /etc/hosts

## 다음의 내용 추가

172.16.238.2 master01

172.16.238.3 master02

172.16.238.4 slave01

172.16.238.5 slave02

172.16.238.6 slave03

172.16.238.7 metastore

172.16.238.8 client

코드 코드 3. Docker 설치 명령어

### **2) 데이터 클러스터 실행**

1. **스크립트 실행**

hadoop-cluster/run.sh 스크립트에는 컨테이너의 이미지 빌드, 설정파일 추가, 컨테이너 실행, SSH key 교환, 서비스 start-up 등 수행해야 할 명령이 담겨있다. 관리자 권한으로 run.sh 스크립트를 실행하면 간편하게 클러스터를 시작할 수 있다.

1. **Cassandra, Hive DB 스키마 및 테이블 생성**

hadoop-cluster/dbinit.sh 스크립트에는 Cassandra와 Hive DDL이 포함되어있다. 관리자권한으로 dbinit.sh 스크립트를 실행하면 간편하게 데이터베이스를 초기화할 수 있다.

1. **Client 컨테이너 VCN 연결**

데이터 클러스터에 접근하기 위한 VCN 연결은 다음과 같이 진행할 수 있다. 기본적으로 접근을 원하는 머신에는 VCN 클라이언트가 설치되어 있어야 한다.

ssh -L 62000:localhost:5900 -N ubuntu@ec2-3-38-85-143.ap-northeast-2.compute.amazonaws.com -i [키페어 위치]

코드 4. SSH 터널링 명령어

클라이언트 머신의 62000 포트에 원격 서버의 5900 포트가 터널링되어 localhost:62000 주소로 VCN 클라이언트를 접속할 수 있다.

1. **Spark 작업을 위한 HDFS 디렉토리 생성**

master01~slave03 혹은 client 컨테이너에서 HDFS에 접근할 수 있다. 다음과 같은 명령을 수행할 수 있다.

hadoop fs -mkdir -p /user/spark/event/log

코드 5. Spark event HDFS 디렉토리 생성 명령어

1. **Client 내 Jupyter Lab 서버 시작**

PySpark 파이프라인 스크립트 개발을 위해 다음과 같이 Jupyter Lab 서버를 시작할 수 있다.

jupyter lab --ip=0.0.0.0 --port=9090 --allow-root

코드 6. Jupyter Lab 서버 시작 명령어

hadoop-cluster/client/notebook 디렉토리가 client 컨테이너의 root/notebook 디렉토리에 바인딩되어 진행상황을 저장할 수 있다.

결론적으로 client 컨테이너에서는 다음과 같은 Web UI에 접근할 수 있다.

|  |  |
| --- | --- |
| 주소 | 용도 |
| master01:9870 | HDFS web UI |
| master01:8088 | YARN RM web UI |
| master01:8042 | YARN NM web UI |
| master02: 19888 | Job history server web UI |
| slave01:15672 | RabbitMQ web UI |
| slave01:5080 | Airflow web server UI |
| slave01:5555 | Flower web UI |
| master01:10002 | Hiveserver2 web UI |

표 5. 각종 툴의 web UI

### **3) Jenkins 컨테이너 실행**

1. **Jenkins 컨테이너 빌드 및 실행**

jenkins/Dockerfile 도커 스크립트를 통해 다음과 같이 이미지를 빌드한 후 실행한다. 아래 명령어는 Jenkins 디렉토리 내에서 실행되어야 한다.

sudo docker build -t jenkins/jenkins:custom .

sudo docker run -d -it -u root -v ./jenkins\_home:/var/jenkins\_home \

-v /var/run/docker.sock:/var/run/docker.sock -p 9090:8080 -p 50000:50000 \

--network hadoop-cluster\_cluster-net --name jenkins\_container jenkins/jenkins:custom

코드 7. Jenkins 컨테이너 실행 명령어

1. **Jenkins 컨테이너 내 SSH key 배포**

데이터 클러스터에 접근하기 위해서 SSH 키를 배포해주어야 한다. Jenkins/sbin 내의 스크립트를 통해 수행한다.

sudo docker exec -u 0 jenkins\_container /usr/sbin/sshd-keygen -A

sudo docker exec -u 0 jenkins\_container /etc/init.d/ssh start

sudo docker exec -u 0 jenkins\_container ssh-keygen -t rsa -P '' -f /root/.ssh/id\_rsa

sudo bash sbin/deploy-ssh-authorized-keys.sh

코드 8. Jenkins 컨테이너 키 배포 명령어

### **3) Jenkins 설정**

1. **Jenkins 웹 UI 진입**

실행 된 컨테이너는 http://[EC2 퍼블릭 도메인]:9090 으로 접근할 수 있다. 최초 실행 시 바인딩 한 Jenkins\_home 디렉토리 내 서버 로그에서 initial admin 키를 확인할 수 있다. 이를 통해 최초 계정을 생성, 기본 플러그인을 설치한다. 플러그인 설치 실패 시, 다음 단계에서 수동으로 설치할 수 있다. 플러그인 설치가 완료된 후 Jenkins가 종료된다면 컨테이너를 다시 start 한다.

1. **Jenkins 플러그인 설치**

기본적인 플러그인이 설치되지 않았을 시, 우측 상단에 Dependency 오류가 발생한다. 명시된 플러그인을 설치하면 이를 해결할 수 있다.

대시보드>Jenkins 관리>플러그인 관리 설정 메뉴를 통해 추가적인 플러그인을 설치할 수 있다. Docker 관련 플러그인과 GitLab 관련 플러그인을 설치한다.

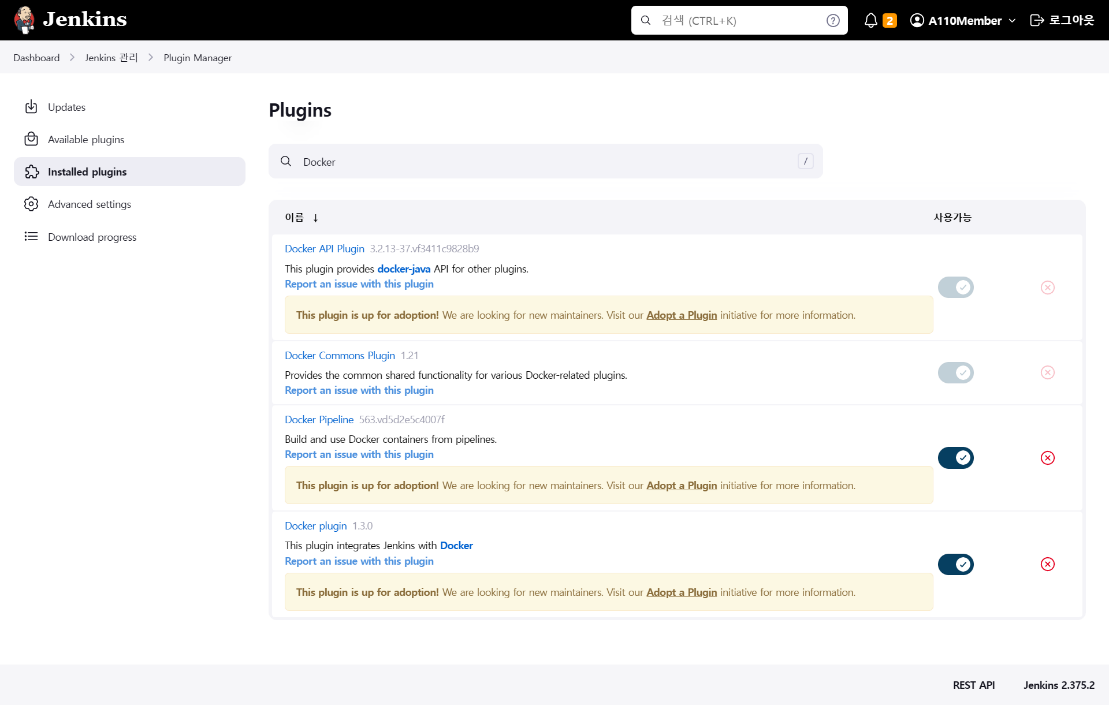


그림 1. Jenkins의 Docker 관련 플러그인

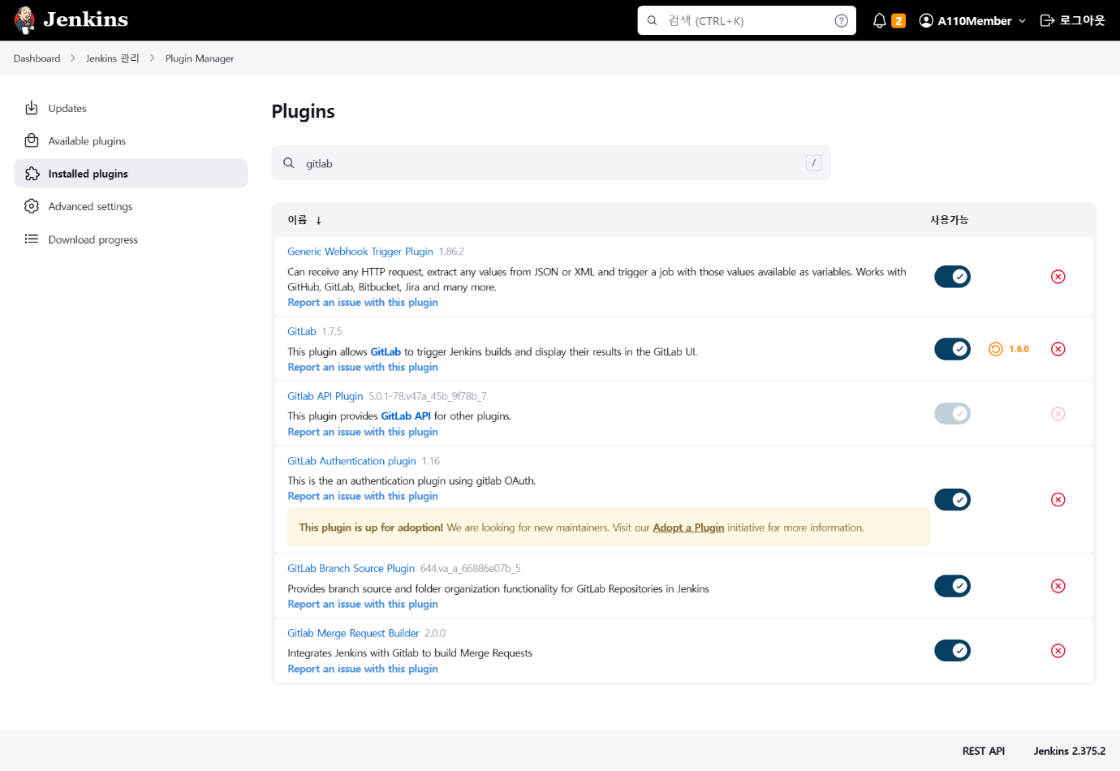


그림 2. Jenkins의 Gitlab 관련 플러그인

1. **Jenkins GitLab Credentials 추가**

대시보드>Jenkins 관리>Manage credentials 메뉴에서 Gitlab 연동 시의 인증 정보를 추가할 수 있다. Global 도메인에 인증 정보를 추가할 수 있다.

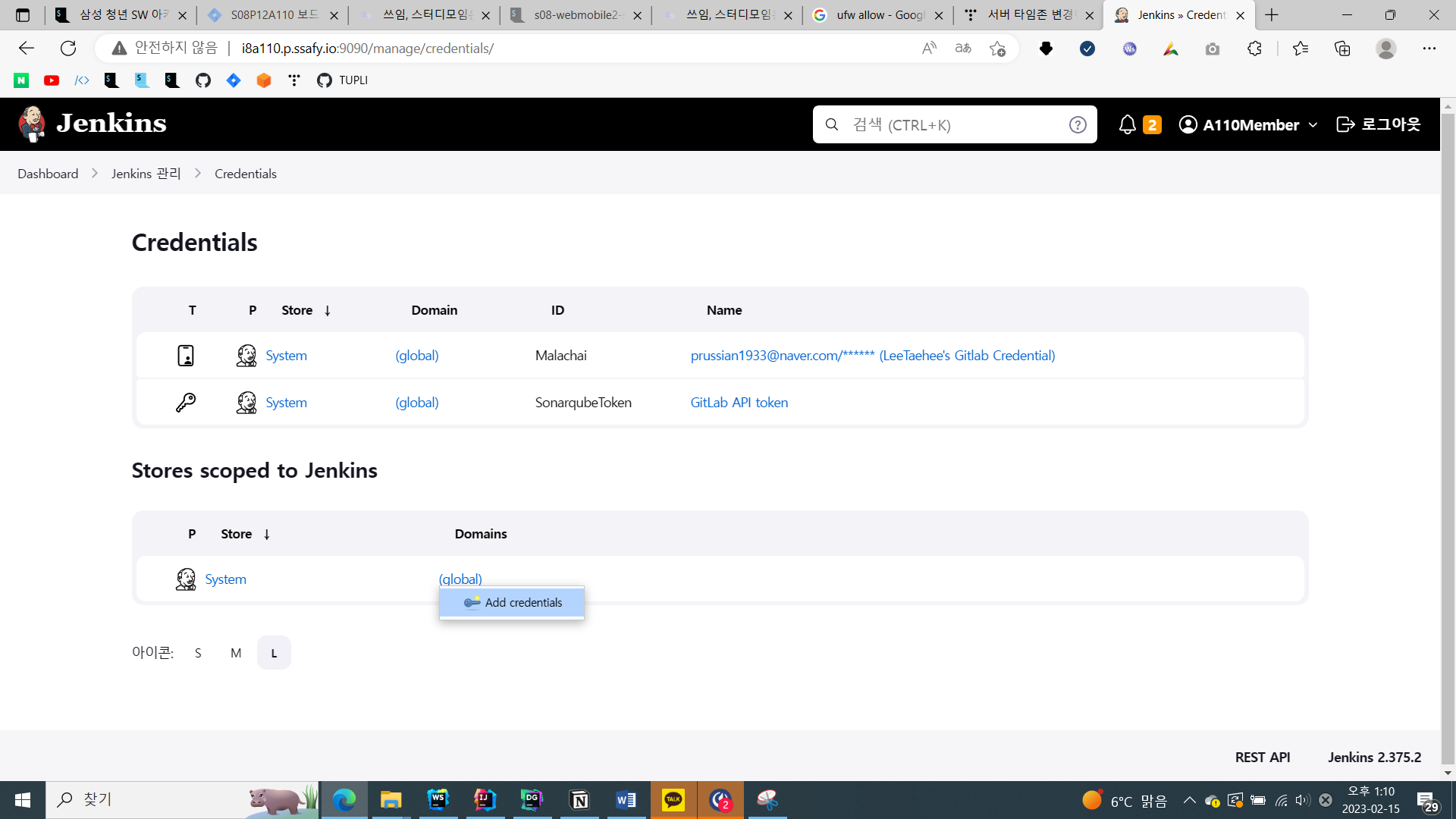


그림 3. Gitlab 인증 정보 추가

SSAFY의 Gitlab 인증방식은 Username with password 방식이기 때문에 알맞게 등록을 진행한다.

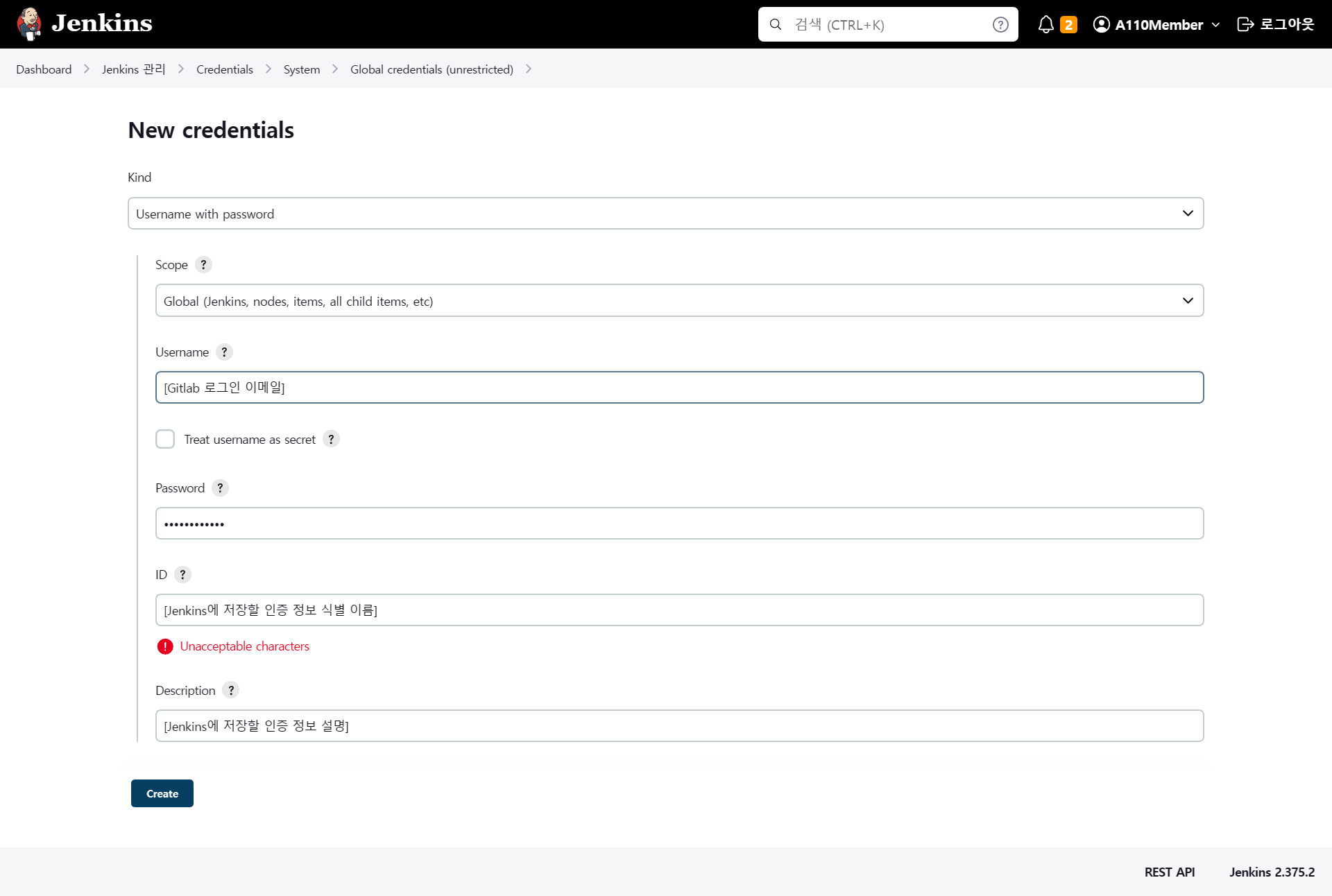


그림 4. Gitlab 인증 정보 추가

### **4) Jenkins 프로젝트 생성**

1. **프리스타일 프로젝트 생성**

대시보드>새로운 Item 메뉴에서 새로운 프리스타일 프로젝트를 생성할 수 있다.

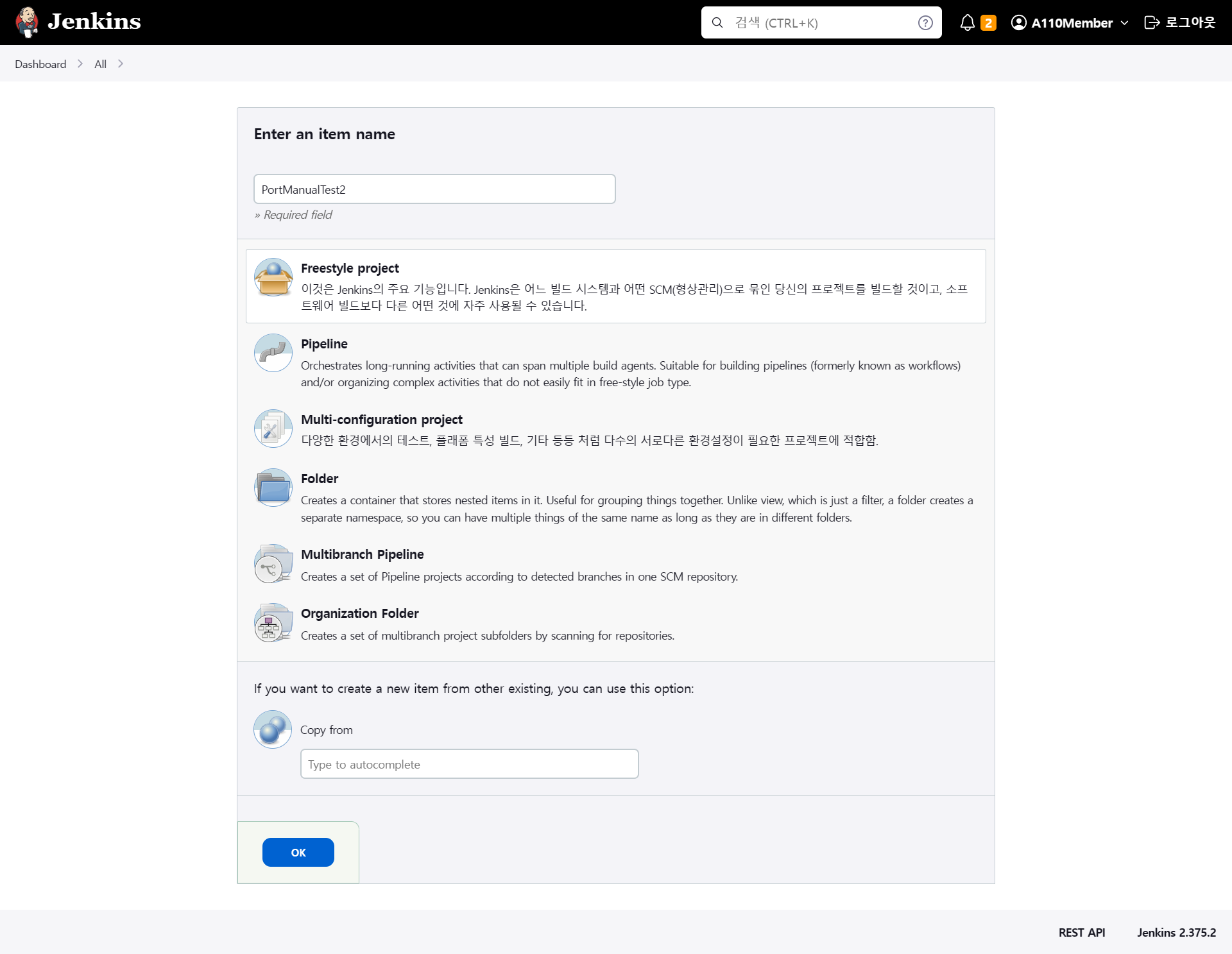


그림 5. Freestyle Project 생성

프로젝트 이름, 설명과 같은 기본적인 틀을 작성한 후 소스 코드 관리 메뉴로 이어 진행한다.

1. **Gitlab 연결**

Jenkins 에서 Gitlab 리포지토리를 자동으로 pull 받을 수 있도록 작업 리포지토리를 등록한다. 미리 등록한 Credential 을 통해 자동으로 접근을 확인한다.

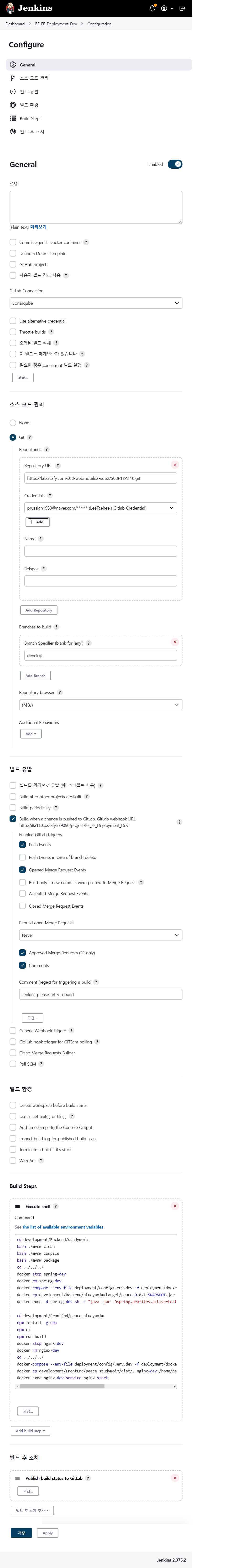
****

그림 6. Gitlab Repository 연결

1. **Gitlab Webhook 연결**

Gitlab 리포지토리에 push 이벤트가 발생하면 트리거를 전송할 수 있도록 Jenkins와 Gitlab Webhook를 연결해준다.

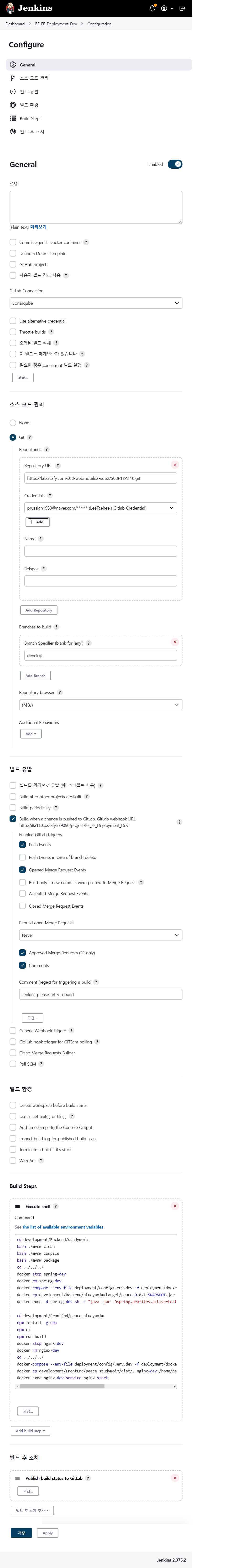
****

그림 7. Gitlab 이벤트 트리거 시 빌드 유발 행동 설정

Gitlab Repository의 Settings>Webhooks 메뉴에서 Webhook을 생성할 수 있다.

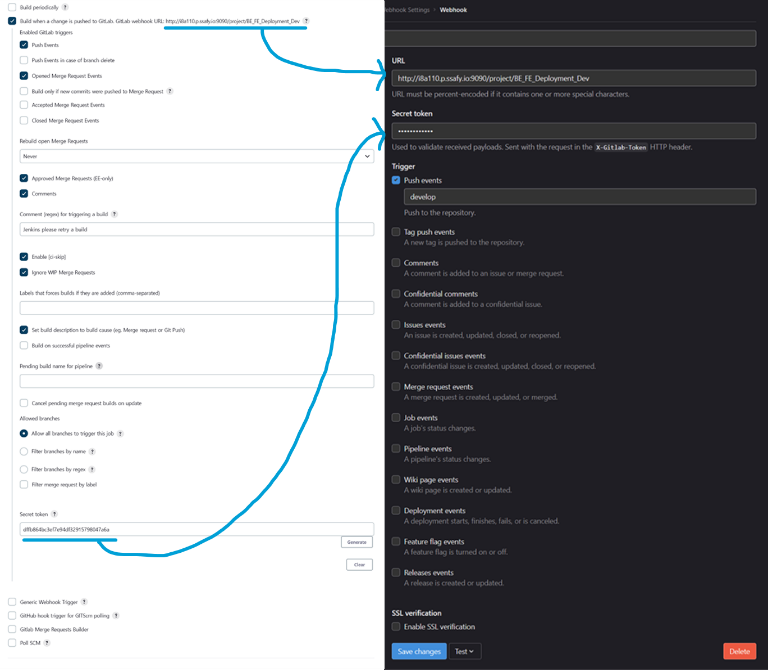


그림 8. Gitlab Webhook 생성 및 URL, Token 설정

1. **Execute shell 설정**

빌드 트리거 발생 시 자동으로 Spring Boot 어플리케이션과 Vue 어플리케이션 빌드, Nginx 배포를 수행하기 위해 쉘 스크립트를 작성한다.

Spring Boot의 로그 파일은 호스트 머신의 $HOME\_DIR/logs/spring/server.log 에서 조회할 수 있다.

cd bigdata-api

docker build -t java/maven:custom .

docker stop spring\_app

docker rm spring\_app

docker run -it -d --network hadoop-cluster\_cluster-net --name spring\_app java/maven:custom /bin/bash

docker cp . spring\_app:/home/bigdata-api

docker exec -w /home/bigdata-api spring\_app sh -c "mvn -Dspring.profiles.active=stage clean compile install package"

docker exec -w /home/bigdata-api -d spring\_app sh -c "java -jar -Dspring.profiles.active=stage mata-api-server/target/mata-api-server-0.0.1-SNAPSHOT.jar >> /home/bigdata-api/webserver.log"

docker exec spring\_app /etc/init.d/redis-server start

코드 9. Spring Boot 어플리케이션 배포를 위한 스크립트

cd vue3

docker build -t node/nginx:custom .

docker stop vue\_app

docker rm vue\_app

docker run -it -d -p 80:80 --network hadoop-cluster\_cluster-net --name vue\_app node/nginx:custom /bin/bash

docker cp . vue\_app:/home/vue3

docker exec -w /home/vue3 vue\_app sh -c "npm i"

docker exec -w /home/vue3 vue\_app sh -c "npm run build"

docker exec vue\_app /etc/init.d/nginx start

코드 10. Vue 어플리케이션 배포를 위한 스크립트

빌드 스크립트 실행 전 컨테이너가 구동중임을 가정하기 때문에, 최초 컨테이너를 미리 구동해야 한다. 이를 위해 필요한 명령은 다음과 같다.

sudo docker run –name spring\_app java/maven:custom

sudo docker run –name vue\_app node/nginx:custom

코드 11. Jenkins의 최초 빌드 유발 전 동작되어야 하는 컨테이너 구동

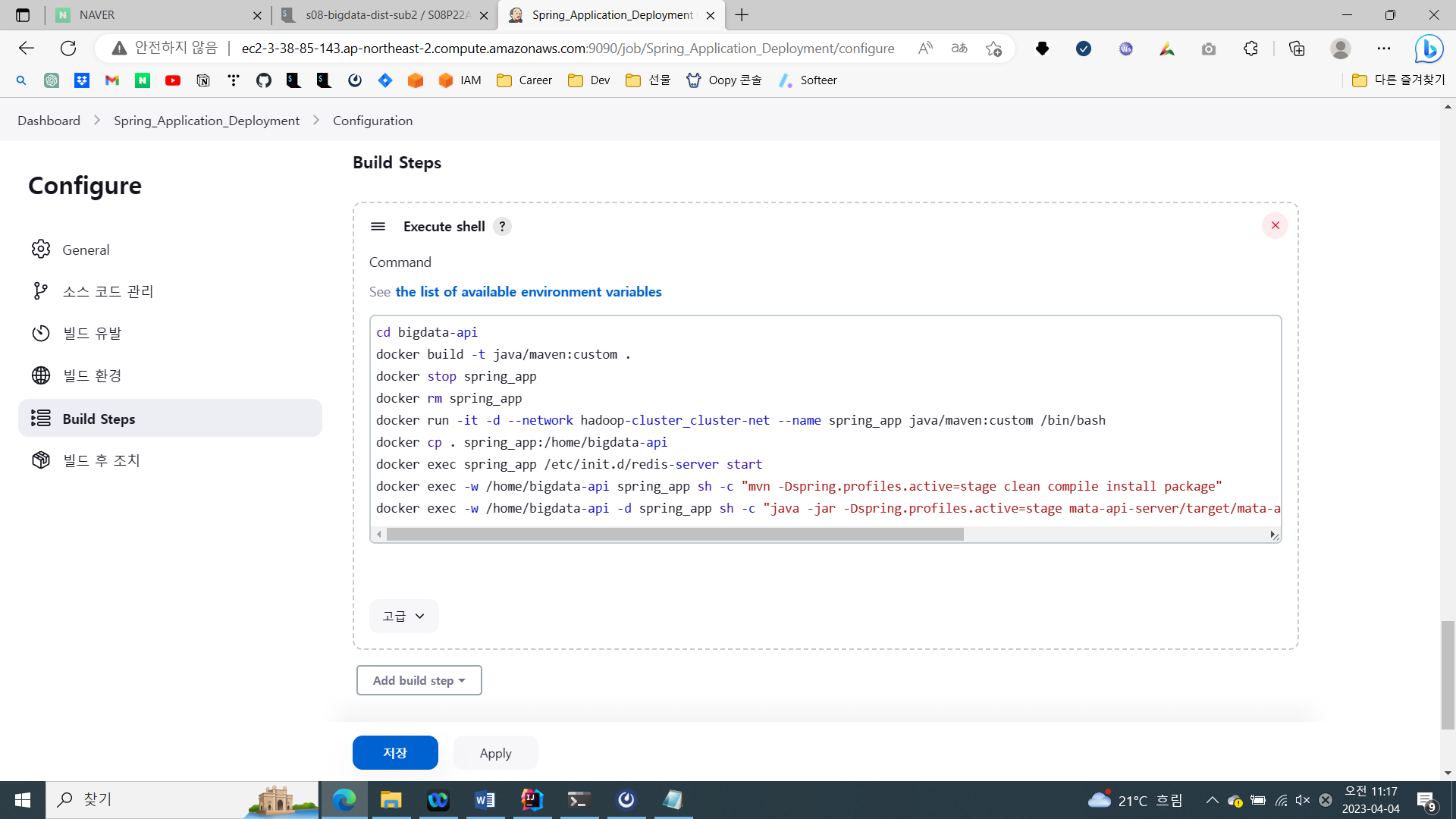


그림 9. Jenkins의 Spring Boot 어플리케이션 배포 스크립트

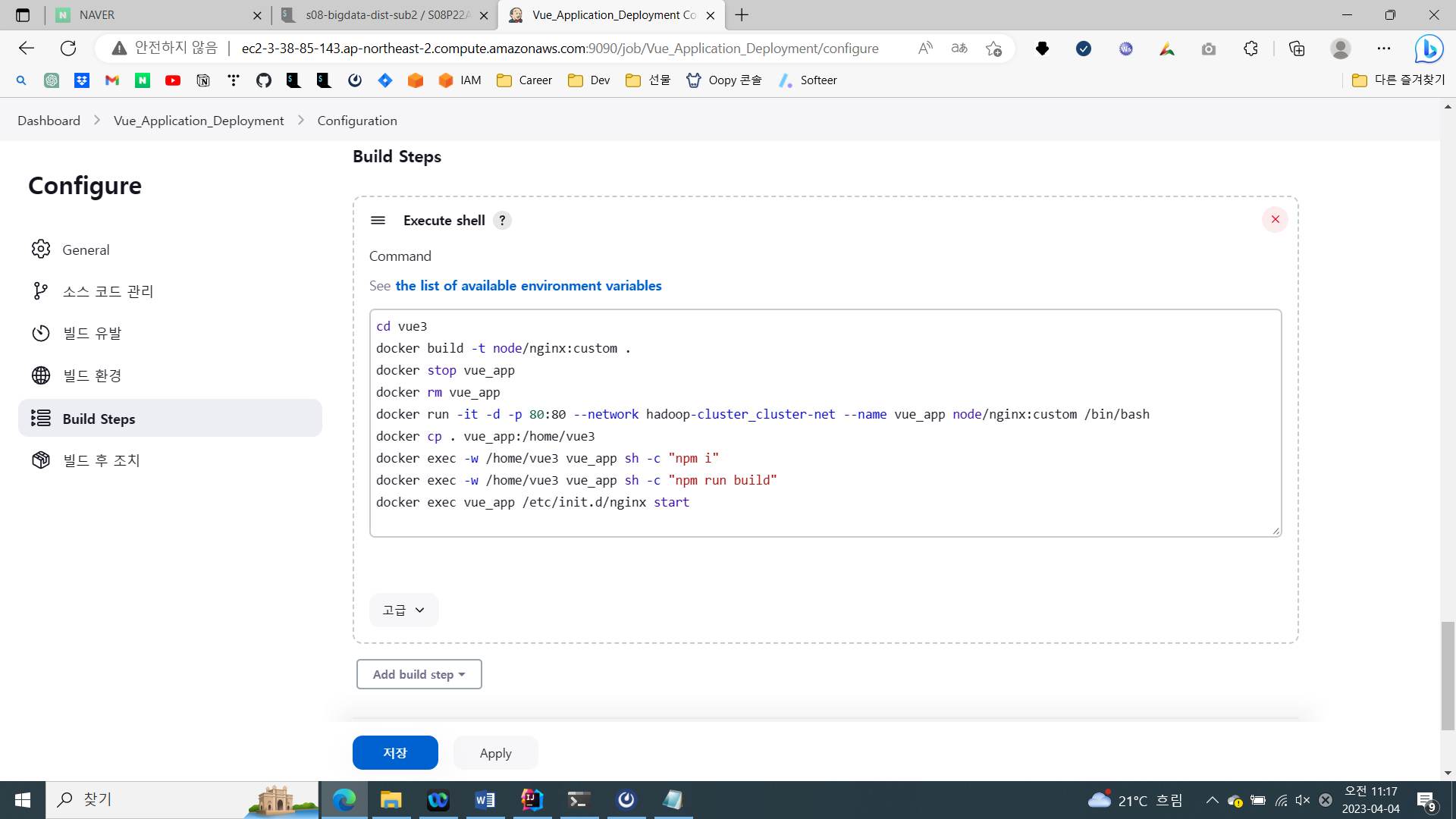


그림 10. Jenkins의 Vue 어플리케이션 배포 스크립트

빌드 수동 트리거 또는 Webhook 트리거 유발 후, 프리스타일 프로젝트의 히스토리 섹션에서 빌드 성공 유무를 확인할 수 있다.

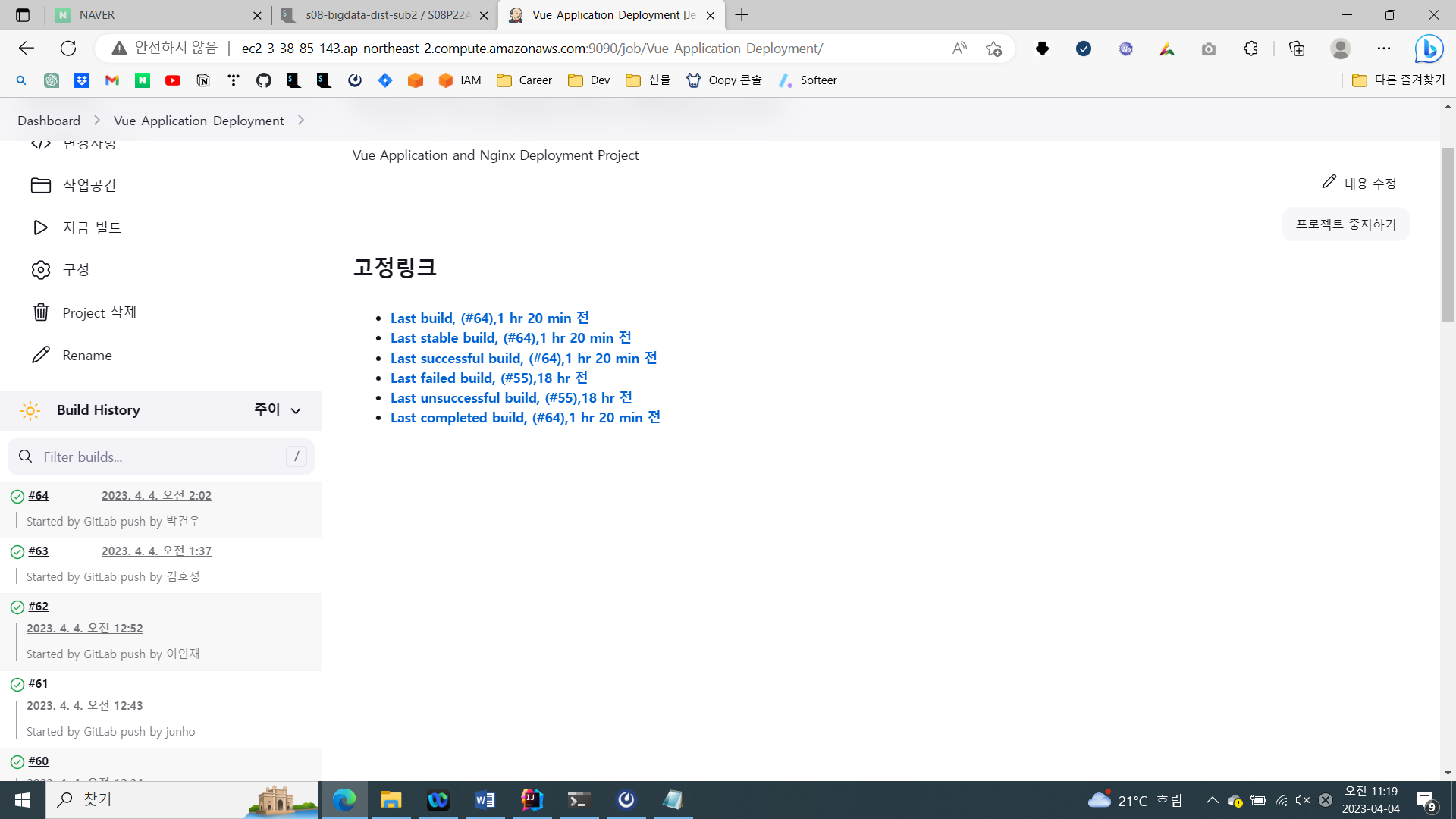
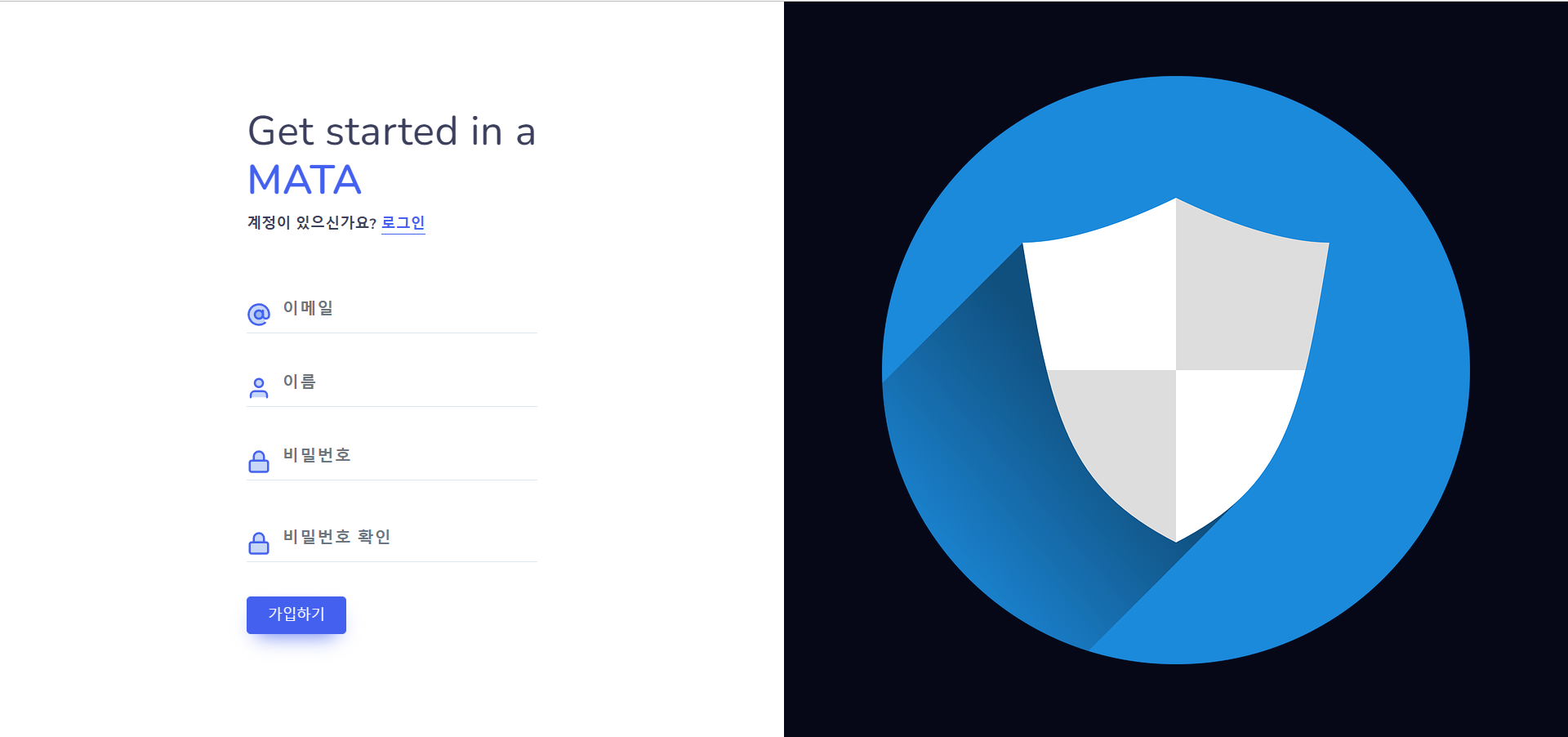
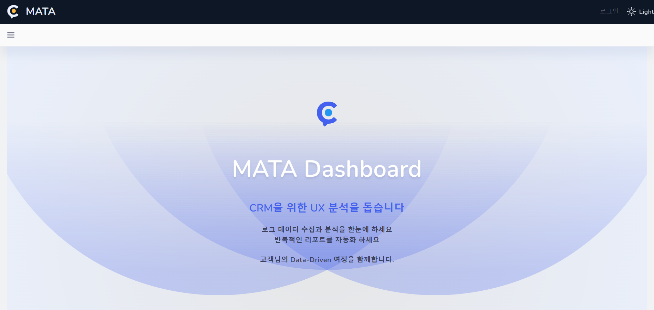


그림 11. 빌드 트리거 유발 후의 빌드 히스토리

## **3. 시연 시나리오**

### **1) 메인 페이지**

웹서비스에 로그 수집을 위한 코드를 주입하기 위해선 MATA 서비스에 가입해야 한다. 서비스 메인 페이지 우측 상단의 로그인 버튼을 눌러 회원가입을 진행한다.

  
그림 12. 회원가입 진행 페이지

이후 로그인하면 대시보드 화면으로 전환된다.

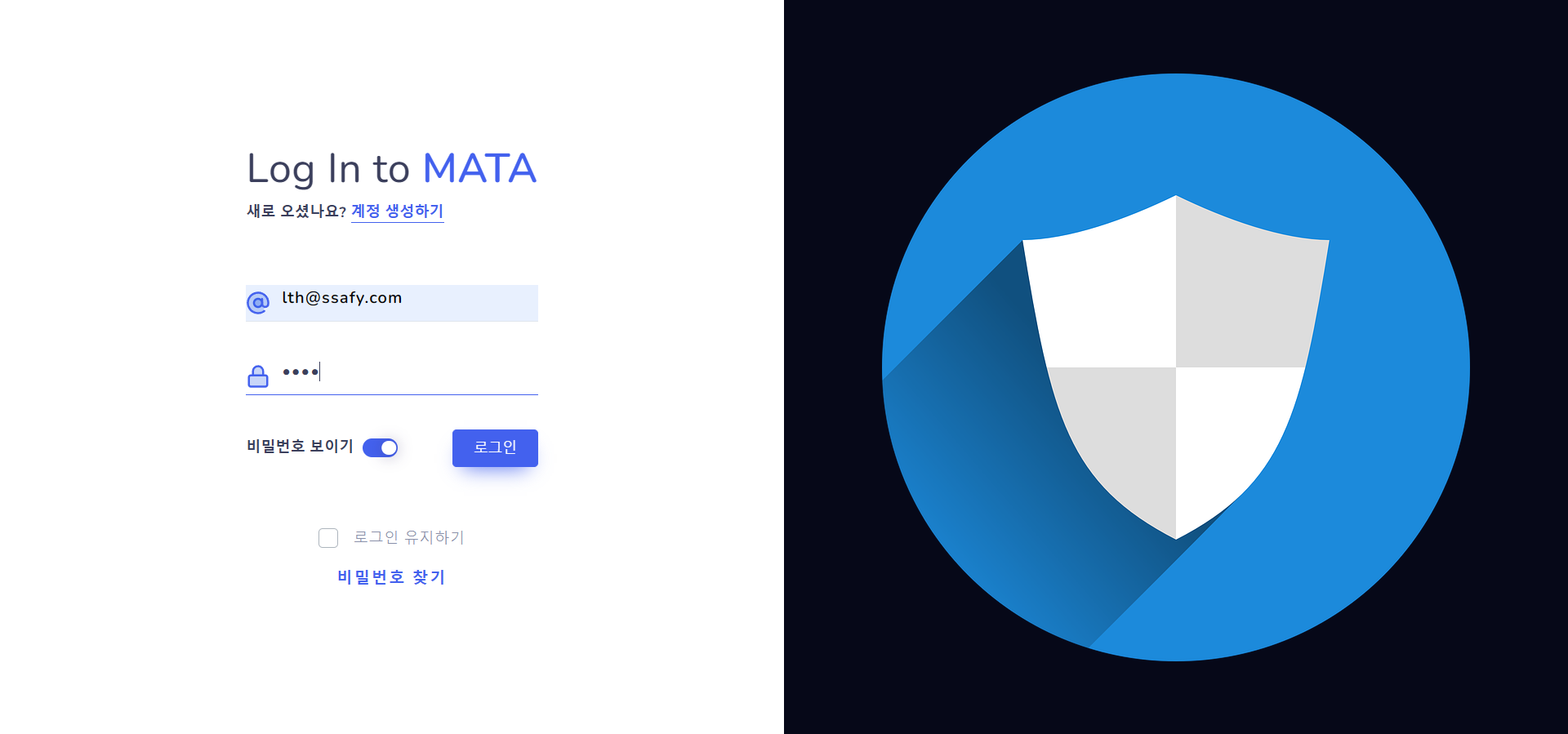


그림 13. 로그인 페이지

### **2) 시작 페이지**

로그인 진행 후 시작 페이지에서 코드를 주입하는 방법을 확인할 수 있다.



그림 14. 시작하기 페이지

좌측 사이드바에서 서비스를 생성한다. 서비스 이름과 서비스 웹사이트 경로, 사이트 목적을 기입할 수 있다.

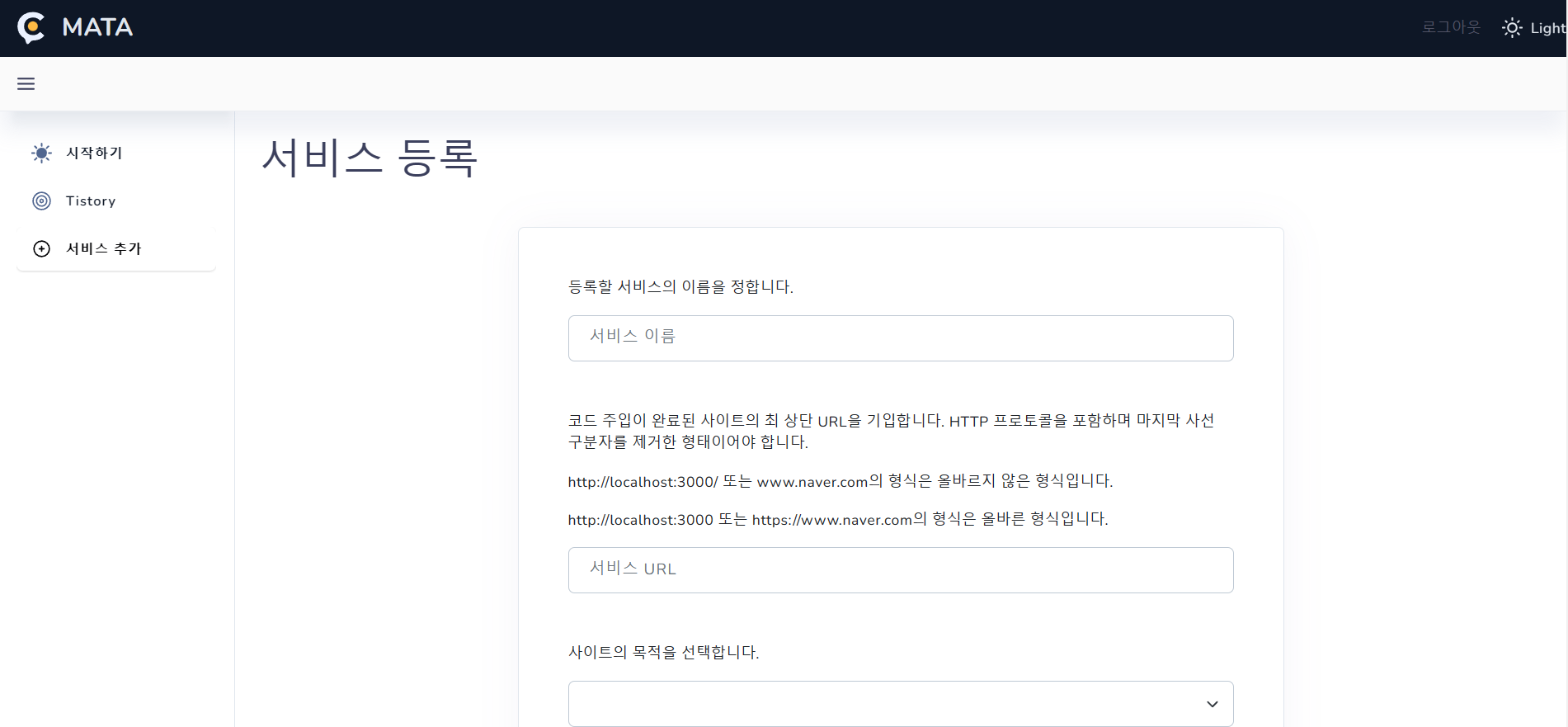


그림 15. 서비스 등록 페이지

서비스를 생성하면, 토큰 발급을 통해 TagManager 객체의 인자로 넘겨주어야 한다. 재발급 버튼을 클릭하여 서비스의 토큰을 발급받는다.

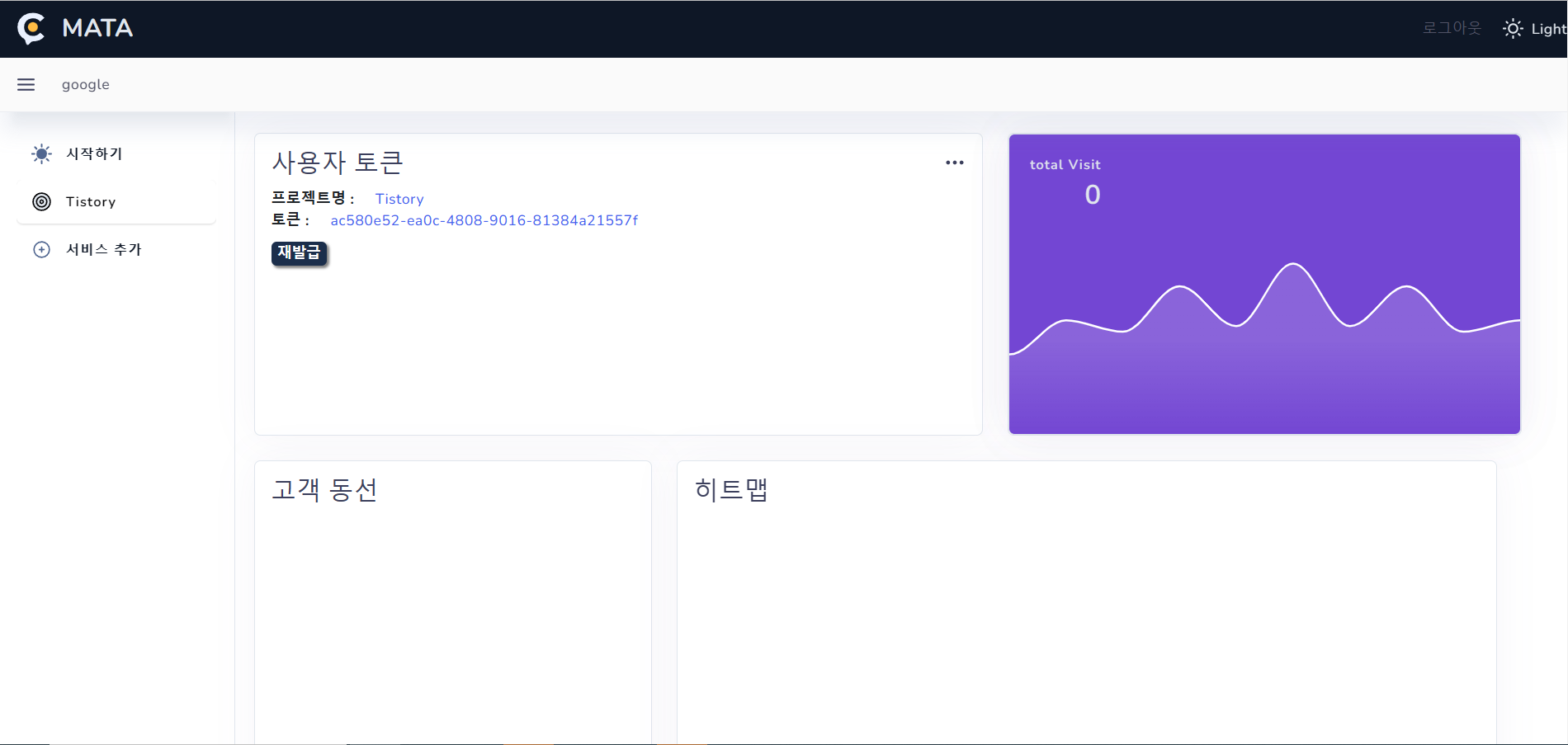


그림 16. 서비스 최초 페이지

웹 서버의 코드 내부에 토큰, 수집 할 이벤트, DOM Selector 쿼리를 입력하여 배포하면 데이터를 수집할 수 있다.

### **3) 서비스 페이지**

서비스를 등록하고 최소 5분의 시간이 지나면 체류시간, 유입 경로, 컴포넌트 통계를 확인할 수 있다.