

# Rumeet 포팅메뉴얼

본 문서는 안드로이드로 개발된 Rumeet 를 사용하기 위한 가이드를 안드로이드와 분산 처리 환경의 리소스 제한으로 인해 Virtual 환경을 구축하는 과정, Spring Boot, FastApi, Hadoop, Spark, RabbitMQ, Kafka 를 배포하는 과정에 대해 서술하고 있습니다.

## 1. 안드로이드 배포

### 1.1. AOS 개발 환경

안드로이드

- Android Studio Dolphin (2021.3.1 Patch 1)
- targetSDK 33
- minSDK 24
- Kotlin

### 1.2. Android 기술 스택

안드로이드 Gradle 7.0 이상부터 지원하는 Version Catalog 활용

```
[versions]

kotlin = "1.7.10"

# androidx
androidx-core = "1.7.0"
androidx-splash = "1.0.0"
androidx-appcompat = "1.5.1"
androidx-test-junit = "1.1.4"
androidx-test-espresso = "3.5.0"
androidx_navigation_version = "2.5.0"
androidx_fragment_version = "1.5.0"
androidx_lifecycle_extension_version = "2.2.0"
navigation_safe_args_version = "2.4.2"
databinding = "7.3.1"
datastore_version = "1.0.0"

# android
android-material = "1.7.0"
android-library = "7.3.0"
android-application = "7.3.0"

# junit
junit = "4.13.2"

# hilt
hilt = "2.44"

# retrofit
retrofit = "2.9.0"
okhttp3 = "4.10.0"

# kakao
kakao_sdk_version = "2.11.2"

# gson
gson_version = "2.9.1"

# custom view library
circleindicator = "2.1.6"
shawnlin013_number_picker_version = "2.4.13"
skydoves_balloon_version = "1.4.6"
circle_image_view_version = "3.1.0"

# google
google_auth_version = "20.4.1"
google_service_version = "4.3.15"
firebase_version = "31.2.3"
firebase_cloud_storage_version = "20.1.0"
firebase_crashlytics_version = "18.3.5"
firebase_crashlytics_gradle_version = "2.9.4"

# glide
glide_version = "4.13.0"
```

```

glide_transformation_version = "4.3.0"

# rabbitmq
rabbitmq_version = "5.13.1"

[libraries]

# Firebase (Firebase cloud message & Analytics & Crashlytics)
google_firebase = { module = "com.google.firebase:firebase-bom", version.ref = "firebase_version" }
google_firebase_analytics = { module = "com.google.firebase:firebase-analytics-ktx" }
google_firebase_storage = { module = "com.google.firebase:firebase-storage-ktx", version.ref = "firebase_cloud_storage_version" }
firebase_messaging = { module = "com.google.firebase:firebase-messaging-ktx", version.ref = "firebase_version" }
firebase_bom = { module = "com.google.firebase:firebase-bom", version.ref = "firebase_version" }
firebase_analytics = { module = "com.google.firebase:firebase-analytics-ktx" }
firebase_crashlytics = { module = "com.google.firebase:firebase-crashlytics-ktx", version.ref = "firebase_crashlytics_version" }

# android core (안드로이드 기본 제공 라이브러리)
androidx_core = { module = "androidx.core:core-ktx", version.ref = "androidx-core" }
androidx_appcompat = { module = "androidx.appcompat:appcompat", version.ref = "androidx-appcompat" }
android_material = { module = "com.google.android.material:material", version.ref = "android-material" }

# junit (테스트 도구)
junit = { module = "junit:junit", version.ref = "junit" }
android_test_junit = { module = "androidx.test.ext:junit", version.ref = "androidx-test-junit" }
android_test_espresso = { module = "androidx.test.espresso:espresso-core", version.ref = "androidx-test-espresso" }

# hilt (DI)
hilt = { module = "com.google.dagger:hilt-android", version.ref = "hilt" }
hilt_compiler = { module = "com.google.dagger:hilt-compiler", version.ref = "hilt" }

# Kakao (카카오 로그인)
kakao_sdk = { module = "com.kakao.sdk:v2-all", version.ref = "kakao_sdk_version" }

# Jetpack Databinding (모델과 뷰 사이의 데이터 바인딩)
databinding = { module = "androidx.databinding:databinding-runtime", version.ref = "databinding" }
databinding_compiler = { module = "com.android.databinding:compiler", version.ref = "databinding" }

# Jetpack Navigation (프래그먼트의 유연함을 더욱 편리하게 적용)
androidx_navigation_fragment_ktx = { module = "androidx.navigation:navigation-fragment-ktx", version.ref = "androidx_navigation_version" }
androidx_navigation_ui_ktx = { module = "androidx.navigation:navigation-ui-ktx", version.ref = "androidx_navigation_version" }
androidx_navigation_common = { module = "androidx.navigation:navigation-common", version.ref = "androidx_navigation_version" }
androidx_navigation_dynamic_features_fragment = { module = "androidx.navigation:navigation-dynamic-features-fragment", version.ref = "androidx_navigation_version" }
androidx_navigation_testing = { module = "androidx.navigation:navigation-testing", version.ref = "androidx_navigation_version" }
androidx_fragment_ktx = { module = "androidx.fragment:fragment-ktx", version.ref = "androidx_fragment_version" }

# Jetpack ViewModel (생명주기에 따른 관리)
androidx_lifecycle_extensions = { module = "androidx.lifecycle:lifecycle-extensions", version.ref = "androidx_lifecycle_extension_version" }

# Splash Screen (안드로이드 버전 상승에 따른 스플래시 스크린 API)
androidx_core_splashscreen = { module = "androidx.core:core-splashscreen", version.ref = "androidx-splash" }

# Retrofit (Http 통신 라이브러리)
retrofit = { module = "com.squareup.retrofit2:retrofit", version.ref = "retrofit" }
retrofit_scalars_converter = { module = "com.squareup.retrofit2:converter-scalars", version.ref = "retrofit" }
retrofit_gson_converter = { module = "com.squareup.retrofit2:converter-gson", version.ref = "retrofit" }
retrofit_rxjava = { module = "com.squareup.retrofit2:adapter-rxjava2", version.ref = "retrofit" }

# OkHttp3 (인터셉터, 로깅 등 네트워크 통신간 유틸 기능)
okhttp3 = { module = "com.squareup.okhttp3:okhttp", version.ref = "okhttp3" }
okhttp3_interceptor = { module = "com.squareup.okhttp3:logging-interceptor", version.ref = "okhttp3" }
okhttp3_urlconnection = { module = "com.squareup.okhttp3:okhttp-urlconnection", version.ref = "okhttp3" }
okhttp3_logging = { module = "com.squareup.okhttp3:logging-interceptor", version.ref = "okhttp3" }

# CircleIndicator (ViewPager의 인디케이터)
circleindicator = { module = "me.relex:circleindicator", version.ref = "circleindicator" }

# NumberPicker (커스텀 Number Picker 라이브러리)
shawnlin013-numberpicker = { module = "io.github.ShawnLin013:number-picker", version.ref = "shawnlin013_number_picker_version" }

# CircleImageView (원형 이미지)
circleimageview = { module = "de.hdodenhof:circleimageview", version.ref="circle_image_view_version" }

# Gson (String을 객체로 변환)
google_gson = { module = "com.google.code.gson:gson", version.ref = "gson_version" }

# Glide (이미지 프로세스 라이브러리)
glide = { module = "com.github.bumptech.glide:glide", version.ref = "glide_version" }
glide_compiler = { module = "com.github.bumptech.glide:compiler", version.ref = "glide_version" }
glide_transformation = { module = "jp.wasabeef:glide-transformations", version.ref = "glide_transformation_version" }

# Datastore (Flow 기반 비동기 로컬 저장소)
datastore_preferences = { module = "androidx.datastore:datastore-preferences", version.ref = "datastore-version" }
datastore_preferences_core = { module = "androidx.datastore:datastore-preferences-core", version.ref = "datastore-version" }

# RabbitMQ (메세지 브로커)
rabbitmq_client = { module = "com.rabbitmq:amqp-client", version.ref = "rabbitmq_version" }

```

```
# bundles는 여러 라이브러리를 한가지로 묶기 위함
[bundles]
androidx = ["androidx_core", "androidx_appcompat", "androidx-navigation-fragment-ktx", "androidx-navigation-ui-ktx", "androidx-navigation-fragment-ktx", "androidx-lifecycle-extensions", "androidx-core-splashscreen", "datastore-preferences", "datastore-preferences"]
navigation = ["androidx-navigation-fragment-ktx", "androidx-navigation-ui-ktx", "androidx-navigation-common", "androidx-navigation-dynamic"]
retrofit-bundles = ["retrofit", "retrofit-gson-converter", "retrofit-rxjava", "retrofit-scalars-converter"]
okhttp3-bundles = ["okhttp3", "okhttp3-interceptor", "okhttp3-urlconnection", "okhttp3-logging"]

# 안드로이드 플러그인 (프로젝트 단위의 gradle에 적용)
[plugins]
kotlin = { id = "org.jetbrains.kotlin.android", version.ref = "kotlin" }
android_library = { id = "com.android.library", version.ref = "android-library" }
android_application = { id = "com.android.application", version.ref = "android-application" }
hilt = { id = "com.google.dagger.hilt.android", version.ref = "hilt" }
navigation_safe_args_plugin = { id = "androidx.navigation.safeargs", version.ref = "navigation_safe_args_version" }
google_service = { id = "com.google.gms.google-services", version.ref = "google_service_version" }
firebase_crashlytics = { id = "com.google.firebase.crashlytics", version.ref = "firebase_crashlytics_gradle_version" }
```

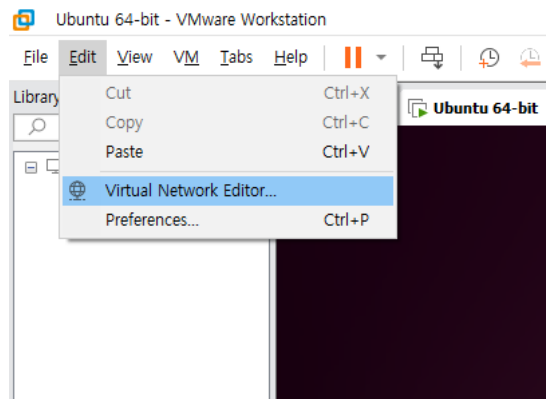
## 2. Virtual Machine 환경 구축 과정

### 2.1 과정

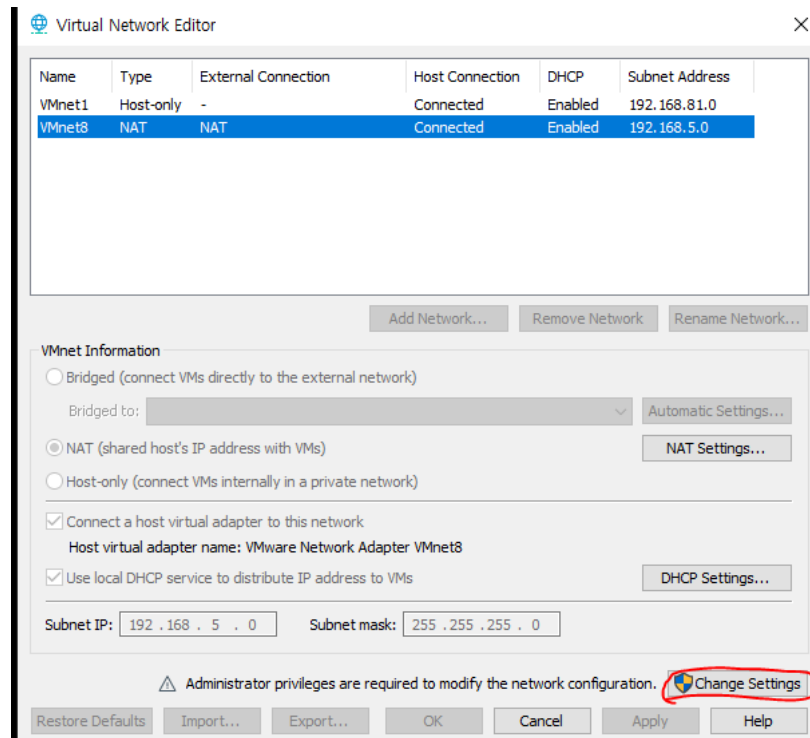
1. VMware Workstation 17 Pro 를 설치한다.
2. Ubuntu 22.04.2 LTS 의 iso를 받아 VMware에 설치한다.
3. 설치 후 다음 과정을 따른다.

#### ▼ 네트워크 설정

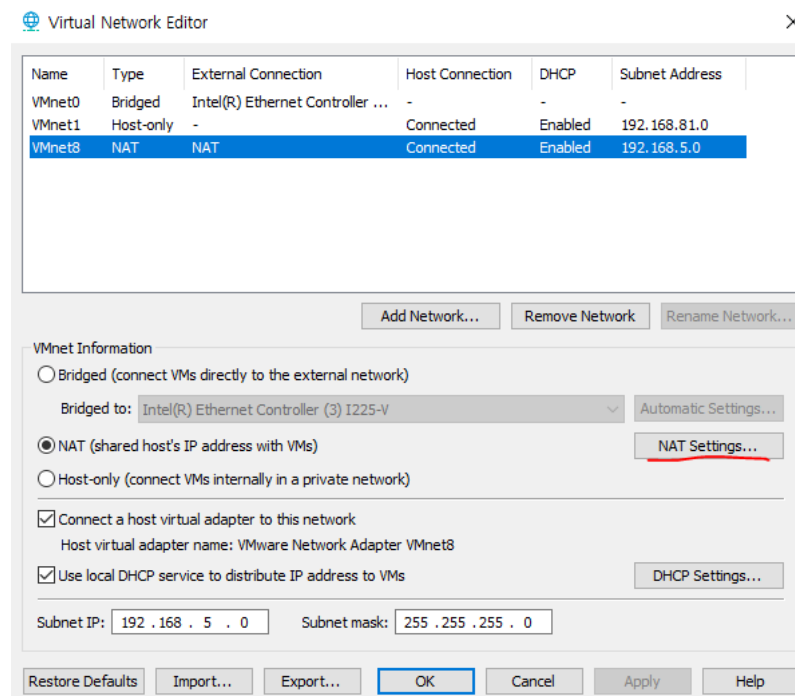
1. Edit - Virtual Network Editor 선택



2. VMnet8 - Change Settings



### 3. VMnet8 - NAT Settings...



### 4. Add...

NAT Settings

Network: vmnet8  
Subnet IP: 192.168.5.0  
Subnet mask: 255.255.255.0  
Gateway IP: 192.168.5.2

Port Forwarding

Host Port	Type	Virtual Machine IP Address	Description
22	TCP	192.168.5.128:22	ssh
9870	TCP	192.168.5.128:9870	hdfshttp
9866	TCP	192.168.5.128:9866	하둡

Add... Remove Properties

Advanced

☒ Allow active FTP  
☒ Allow any Organizationally Unique Identifier  
UDP timeout (in seconds): 30  
Config port: 0  
☐ Enable IPv6  
IPv6 prefix: fd15:4ba5:5a2b::1008::/64  
DNS Settings... NetBIOS Settings...

OK Cancel Help

5. 다음을 확인하여 적고 OK를 누른다.

Map Incoming Port

Host port:

Type: ☒ TCP ☐ UDP

Virtual machine IP address:

Virtual machine port:

Description:

OK Cancel Help

- Host port : 외부 접속 포트
- Type : TCP 고정
- Virtual machine IP address : 실제 인스턴스의 IP를 적는다.(192.168.5.128)
- Virtual machine port : 인스턴스에서 외부로 연결 하고 싶은 포트를 적는다.
- Description : 선택사항(설명 기재)

6. Rumeet 메뉴얼 (H-port, ip, V-port, dec)

- 22, 192.168.5.128, 22, ssh
- 9870, 192.168.5.128, 9870, hdfshttp
- 9866, 192.168.5.128, 9866, 하둡
- 9864, 192.168.5.128, 9864, 데이터노드
- 4040, 192.168.5.128, 4040, 스파크
- 8080, 192.168.5.128, 8080, 스파크
- 8000, 192.168.5.128, 8000, fastAPI
- 8001, 192.168.5.128, 8001, fast
- 8002, 192.168.5.128, 8002, fast

## ▼ Open SSH Server

```
sudo apt-get update
sudo apt-get install openssh-server

sudo vi /etc/ssh/sshd_config
- PermitRootLogin yes

sudo service ssh restart
```

## ▼ Hadoop

### 1. Hadoop 설치

```
sudo apt update
sudo apt install openjdk-17-jdk
wget https://downloads.apache.org/hadoop/common/hadoop-3.3.3/hadoop-3.3.3.tar.gz
tar -xzf hadoop-3.3.3.tar.gz
sudo mv hadoop-3.3.3 /usr/local/hadoop
```

### 2. 환경 변수 설정

```
vi ~/.bashrc

맨 밑에
export HADOOP_HOME=/usr/local/hadoop
export PATH=$PATH:$HADOOP_HOME/bin:$HADOOP_HOME/sbin
추가 후
source ~/.bashrc
```

### 3. 파일 수정

- cd \$HADOOP\_HOME/etc/hadoop/

#### 1. hadoop-env.sh

```
vi hadoop-env.sh

export JAVA_HOME=/usr/lib/jvm/java-17-openjdk-amd64
```

#### 2. core-site.xml

```
<configuration>
  <property>
    <name>fs.defaultFS</name>
    <value>hdfs://0.0.0.0:9000</value>
  </property>
</configuration>
```

#### 3. yarn-site.xml

```
<configuration>

<!-- Site specific YARN configuration properties -->
  <property>
    <name>yarn.nodemanager.aux-services</name>
    <value>mapreduce_shuffle</value>
  </property>
  <property>
    <name>yarn.nodemanager.aux-services.mapreduce.shuffle.class</name>
    <value>org.apache.hadoop.mapred.ShuffleHandler</value>
  </property>

</configuration>
```

#### 4. mapred.xml

```
<configuration>
  <property>
    <name>mapreduce.framework.name</name>
    <value>yarn</value>
  </property>
</configuration>
```

#### 5. `hdfs namenode -format`

#### 6. 키 생성 및 배포

```
ssh-keygen -t rsa
ssh-copy-id rumeet@localhost
sudo usermod -aG hadoop rumeet
```

#### 7. 실행

```
$HADOOP_HOME/sbin/start-all.sh

혹시 실행 안될시,

unset HDFS_NAMENODE_USER
unset HDFS_DATANODE_USER
unset HDFS_SECONDARYNAMENODE_USER
unset YARN_RESOURCEMANAGER_USER
unset YARN_NODEMANAGER_USER

export HDFS_NAMENODE_USER=rumeet
export HDFS_DATANODE_USER=rumeet
export HDFS_SECONDARYNAMENODE_USER=rumeet
export YARN_RESOURCEMANAGER_USER=rumeet
export YARN_NODEMANAGER_USER=rumeet
```

### ▼ Spark

#### 1. Spark 설치

```
wget https://downloads.apache.org/spark/spark-3.3.2/spark-3.3.2-bin-hadoop3.tgz

tar xvf spark-3.3.2-bin-hadoop3.tgz
```

#### 2. 환경 변수 설정

```
vi ~/.bashrc

export SPARK_HOME=/spark-3.3.2-bin-hadoop3
export PATH=$PATH:$SPARK_HOME/bin:$SPARK_HOME/sbin

source ~/.bashrc
```

#### 3. 초기 설정

```
cd $SPARK_HOME/conf/
cp spark-env.sh.template spark-env.sh
vi spark-env.sh

SPARK_MASTER_HOST=192.168.5.128
SPARK_LOCAL_IP=192.168.5.128
SPARK_WORKER_CORES=6
SPARK_WORKER_MEMORY=8g
```

#### 4. 실행

```
$SPARK_HOME/sbin/start-master.sh
$SPARK_HOME/sbin/start-worker.sh spark://localhost:7077
```

## 3. Spring Boot 배포 과정

### 3.1. 개발환경 및 배포 환경

개발 환경

- 'org.springframework.boot' version '3.0.4'
- 빌드 도구 : Gradle
- IDE : IntelliJ IDEA 2022.3.1

배포 환경

- 배포 서버 : Ubuntu 20.04.4 LTS (AWS)
- Docker : 20.10.23
- Gradle : Gradle 8.1-rc-2

### 3.2. API 서버활성을 위한 배포과정

- 기본적으로 CI/CD툴의 젠킨스를 이용한다.
- 위 내용은 9번(젠킨스를 통한 자동배포) 항목에서 자세히 설명한다.

## 4. FAST API 배포 과정

### 4.1. 개발환경 및 배포 환경

개발 환경

- 빌드 도구 : Python3.9
- IDE : PyCharm 2022.1.3

배포 환경

- 배포 서버 : uvicorn
- 라이브러리 : PySpark
- 포트
  - 8001 : PySpark 서버 (be/fast)
    - 구동  

```
nohup uvicorn main:app --reload --host=0.0.0.0 --port=8001 &
```
  - 8002 : 구글 맵 폴리라인 서버 (be/poly)
    - 구동  

```
nohup uvicorn a:app --reload --host=0.0.0.0 --port=8002 &
```

### 4.2. API 서버활성을 위한 배포

▼ 세팅

1. Python3를 설치한다

```
apt-get update
apt-get install python3
apt-get install pip
```

2. pip를 활용하여 FastApi와 Uvicorn을 다운한다



```
pip install fastapi
pip install uvicorn
```

3. 다음을 실행한다. (깃 클론 가정)

```
mkdir ~/fast
mv main.py ~/fast
nohup uvicorn main:app --reload --host=0.0.0.0 --port=8001 &

mkdir ~/img_server
mv a.py ~/img_server
nohup uvicorn a:app --reload --host=0.0.0.0 --port=8002 &
```

## 5. Hadoop

### 5.1. 개발환경 및 배포 환경

개발 환경

- 버전 : 3.3.3
- StandAlone (Name Node : 1, Data Node : 1)

배포 환경

- 배포 서버 : Ubuntu 22.0.4

### 5.2. Hadoop Cluster 구축을 위한 과정

- 커맨드

```
$HADOOP_HOME/bin/hdfs dfs -mkdir -p hdfs://localhost:9000/user/spark/output
$HADOOP_HOME/bin/hdfs dfs -mkdir -p hdfs://localhost:9000/user/spark/output/1km
$HADOOP_HOME/bin/hdfs dfs -mkdir -p hdfs://localhost:9000/user/spark/output/2km
$HADOOP_HOME/bin/hdfs dfs -mkdir -p hdfs://localhost:9000/user/spark/output/3km
$HADOOP_HOME/bin/hdfs dfs -mkdir -p hdfs://localhost:9000/user/spark/output/5km
$HADOOP_HOME/bin/hdfs dfs -chmod -R 777 /user/spark/output
```

## 6. Spark

### 6.1. 개발환경 및 배포 환경

개발 환경

- 버전 : 3.3.2
- StandAlone (Name Node : 1, Data Node : 1)

배포 환경

- 배포 서버 : Ubuntu 22.0.4

## 7. RabbitMQ

### 7.1. 배포 환경

배포 환경

- Docker Container

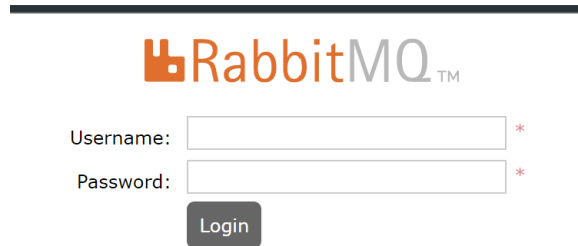
### 7.2. 서버활성을 위한 배포과정

- ▼ 과정

1. docker container 실행

```
docker run -d --name rabbitmq -p 5672:5672 -p 15672:15672 --restart=unless-stopped rabbitmq:management
```

2. <http://j8d204.p.ssafy.io:15672> Username : guest, Password : guest



The image shows the RabbitMQ Management UI login page. It features the RabbitMQ logo at the top. Below the logo, there are two input fields: 'Username:' and 'Password:'. Both fields have a red asterisk to their right, indicating they are required. Below the password field is a 'Login' button.

3. Exchange 구성

<b>chat.exchange</b>	topic	<b>D</b>
<b>friend.user.exchange</b>	direct	<b>D</b>
<b>game.exchange</b>	topic	<b>D</b>
<b>rec.exchange</b>	direct	<b>D</b>
<b>running.exchange</b>	topic	<b>D</b>
<b>user.exchange</b>	topic	<b>D</b>

a. game.exchange

Exchange: game.exchange

► Overview

▼ Bindings

This exchange

↓

To	Routing key	Arguments	
cancel.queue	cancel		Unbind
end.queue	end		Unbind
friend.queue	friend		Unbind
matching.queue	matching		Unbind

b. chat.exchange

Exchange: chat.exchange

► Overview

▼ Bindings

This exchange

↓

To	Routing key	Arguments	
chat.queue	room.*		Unbind

## 8. Kafka

### 8.1. 배포 환경

배포 환경

- Ubuntu 20.0.4

### 8.2. 개발환경 및 배포 환경

```
wget https://dlcdn.apache.org/kafka/3.3.2/kafka_2.12-3.3.2.tgz
tar xvf kafka_2.12-3.3.2.tgz
```

## 9. 젠킨스를 통한 자동배포

### 9.1. 초기 세팅

도커를 통하여 젠킨스를 설치한다.

- <https://dongle94.github.io/docker/docker-ubuntu-install/>
- 해당 사이트를 참고하여 도커 설치한다.
- 설치 후 다음 명령어를 통하여 젠킨스를 설치한다.

도커에 Ansible을 설치한다.

- 명령어를 통하여 Ansible을 설치한다.

```
docker run -itd --name ansible-server -p 20022:22 -e container=docker --tmpfs /run --tmpfs /tmp -v /sys/fs/cgroup:/sys/fs/cgroup:ro -v
```

젠킨스 플러그인을 설치한다.

- Ansible
- GitLab
- Publish Over SSH

젠킨스에 Ansible 컨테이너를 연결한다.

**Publish over SSH**

Jenkins SSH Key ?

Passphrase ?

Concealed

Path to key ?

Key ?

☐ Disable exec ?

SSH Servers

SSH Server Name ?

ansible-server

Hostname ?

172.26.5.131

Username ?

root

Remote Directory ?

.

## ▼ 젠킨스 설정

- JDK

### JDK

#### JDK installations

List of JDK installations on this system

Add JDK

JDK Name

jdk11.0.18

JAVA\_HOME

/opt/java/openjdk

☐ Install automatically ?

- Gradle

Add Gradle

Gradle name ?

gradle

☒ Install automatically ?

Install from Gradle.org

Version

Gradle 8.1-rc-2

Add Installer ▾

## GitLab Webhook 등록

- 젠킨스 프로젝트 설정에서 Build Triggers에 webhook을 등록한다.
- Push Events을 체크한다.

### Build Triggers

- ☐ Build after other projects are built ?
- ☐ Build periodically ?
- ☒ Build when a change is pushed to GitLab. GitLab webhook URL: <http://i8d111.p.ssafy.io:8080/project/Back-End-Project> ?

#### Enabled GitLab triggers

- ☒ Push Events
- ☐ Push Events in case of branch delete
- ☐ Opened Merge Request Events
- ☐ Build only if new commits were pushed to Merge Request ?
- ☐ Accepted Merge Request Events
- ☐ Closed Merge Request Events

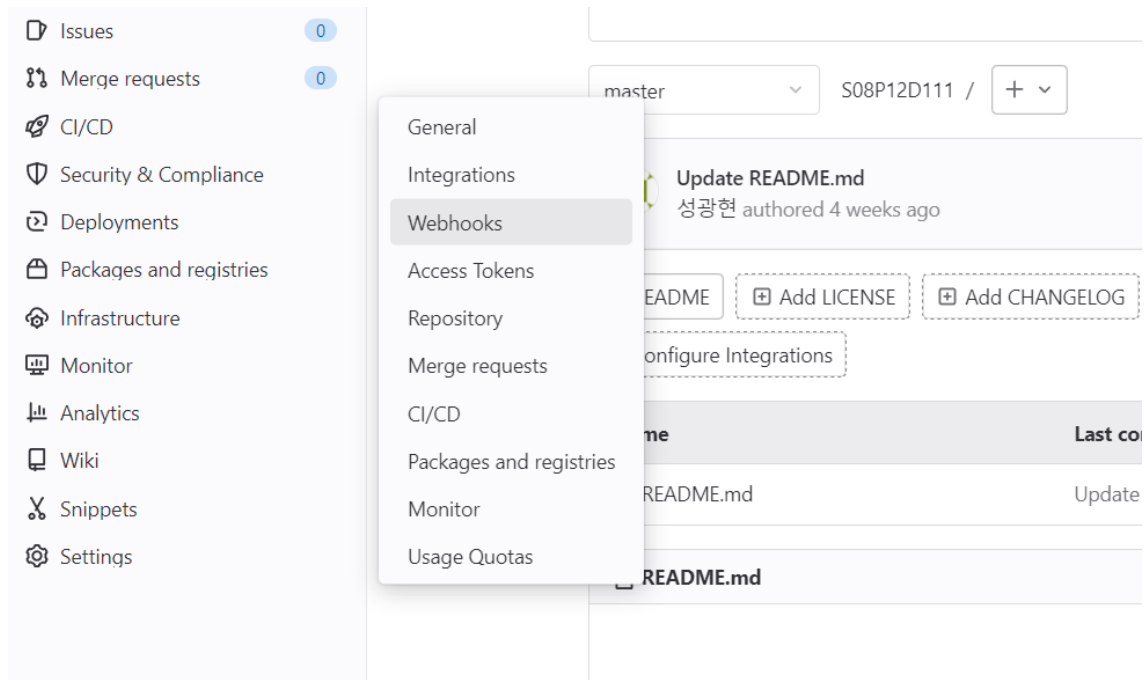
Rebuild open Merge Requests

- Secret Key를 발급한다.

Secret token ?

124c03 [REDACTED] 6bb2

- <https://lab.ssafy.com/s08-bigdata-recom-sub2/S08P22D204>에서 Settings의 Webhooks를 누른다.



- Webhooks 에 Jenkins URL을 적고 Push events에 be/develop 그리고 Jenkins에서 발급받은 Secret token을 적는다.

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**

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

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

A comment is added to a confidential issue.

☐ Issues events

An issue is created, updated, closed, or reopened.

- Add Webhooks 를 눌러 GitLab에 등록한다.

### SSL verification

- ☒ Enable SSL verification

Add webhook

## 6.2. Ansible 설정

### ▼ 파일 목록

- Infra의 Ansible 서버

```
[root@b2b8a7f38df1 ~]# ls
anaconda-ks.cfg  anaconda-post.log  Dockerfile  hosts  original-ks.cfg  playbook.yml
[root@b2b8a7f38df1 ~]#
```

- 사용 네트워크 및 컨테이너

```
"Containers": {
  "16216e7f2622fe74414ea939e24211091f4ac426ae0b359a6f9195fb19335318": {
    "Name": "rabbitmq",
    "EndpointID": "d3e2ae33fc66617800f61bacaf7193d266ad3d42d568c22edaa5d75c389",
    "MacAddress": "02:42:ac:11:00:04",
    "IPv4Address": "172.17.0.4/16",
    "IPv6Address": ""
  },
  "1e9929d48a457b7d7259adde7dae53695b5b29454a4b10b554bc2385b7bc98d3": {
    "Name": "rumeet",
    "EndpointID": "64f36bb21c6d066147ab2b173ebb49ab97c2b6b2e43dfbb4e603ebc0121",
    "MacAddress": "02:42:ac:11:00:05",
    "IPv4Address": "172.17.0.5/16",
    "IPv6Address": ""
  },
  "b2b8a7f38df15b5f249bca69711be8a55028fb98968735892d26b0d62457d9bc": {
    "Name": "ansible-server",
    "EndpointID": "e047fce5bec3d0062bea089e1ebe2ebf15c6910aa060178249949daf78",
    "MacAddress": "02:42:ac:11:00:03",
    "IPv4Address": "172.17.0.3/16",
    "IPv6Address": ""
  },
  "d65afb15b38ed0a2d518a89b7ba216b352ca5540c9dc53aa6f08143f550ce47a": {
    "Name": "jenkins-server",
    "EndpointID": "6ec75ebfe8f43f057137ae8b76d3a4b4ad8be9f0143cf688bea0cbfc4bd",
    "MacAddress": "02:42:ac:11:00:02",
    "IPv4Address": "172.17.0.2/16",
    "IPv6Address": ""
  }
}
```

- Dockerfile

```
FROM openjdk:17-slim

COPY rumeet-0.0.1-SNAPSHOT.jar rumeet.jar

CMD ["java", "-jar", "rumeet.jar"]
```

- playbook.yml

```
- hosts: all
  tasks:
    - name: stop current running container
      command: docker stop rumeet
      ignore_errors: yes

    - name: remove stopped container
      command: docker rm rumeet
      ignore_errors: yes

    - name: remove the docker image from the ansible server
      command: docker rmi rumeet
      ignore_errors: yes

    - name: create a docker image with deployed waf file
```

```

command: docker build -t rumeet .
args:
  chdir: /root

- name: create a container using ckd-project-ansible image
  command: docker run -d --name rumeet -p 80:80 -m 8g rumeet

```

- 1행 hosts에 명시된 모든 도커 컨테이너에 적용
- 2행 작업 실행
  - 1번 작업 rumeet의 컨테이너 종료 (에러 무시)
  - 2번 작업 현재 rumeet의 이름을 가진 컨테이너 삭제 (에러 무시)
  - 3번 작업 rumeet의 이름을 가진 이미지 삭제 (에러 무시)
  - 4번 작업 Dockerfile의 설정에 따라 rumeet라는 이름으로 이미지 생성
  - 5번 작업 rumeet의 이미지를 백그라운드로 80포트를 사용하여 컨테이너 실행
- hosts

```
172.17.0.3
```

### 6.3. 백엔드 서버 배포

- 젠킨스 파이프라인

```

pipeline {
  agent any
  tools {
    gradle 'gradle'
  }
  stages {
    stage('clone') {
      steps {
        git branch: 'be/develop', credentialsId: 'gch03944', url: 'https://lab.ssafy.com/s08-bigdata-recom-sub2/S08P22D204
      }
    }
    stage('build'){
      steps {
        sh '''
          ./gradlew clean build'''
      }
    }
    stage('p') {
      steps {
        sshPublisher(publishers: [sshPublisherDesc(configName: 'ansible', transfers: [sshTransfer(cleanRemote: false, exclu
      ]
    }
  }
}

```