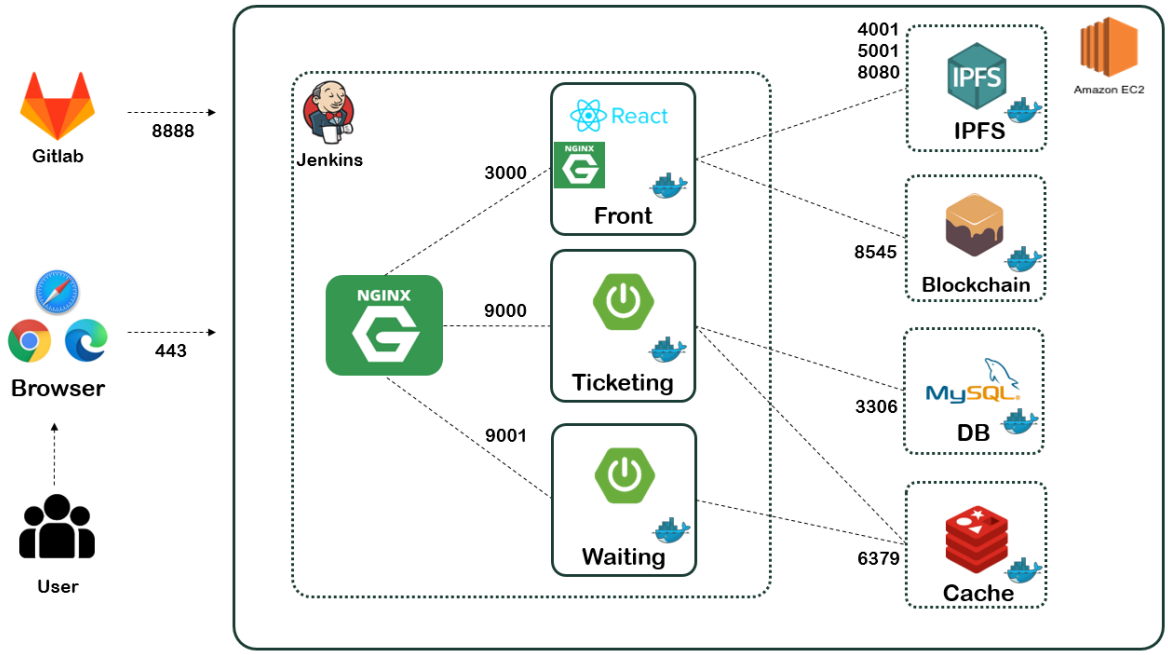
🎪NFT 티켓 서비스 “ 똑켓 “ 포팅 메뉴얼🎪

🖐목차

아키텍쳐


Gitlab
8888
Browser
443
User
Jenkins
NGINX
3000
9000
9001
React
NGINX
Front
Ticketing
Waiting
4001
5001
8080
IPFS
IPFS
Amazon EC2
8545
Blockchain
3306
MySQL
DB
6379
Cache

EC2

Docker version : Docker version 23.0.1, build a5ee5b1

Installed Packages

- net-tools
- nginx
- certbot

Allowed Ports

Port Number	Usage
22	SSH
80	HTTP
443	HTTPS
8888	Jenkins
3000	FrontEnd
9000	Spring Boot
9001	Spring Boot
3306	MySQL
6379 - 6381	Redis
26379 - 26381	Redis Sentinel
4001,5001,8080	IPFS
9094 - 9096	IPFS-CLUSTER

```
sudo ufw enable

sudo ufw allow [port number]

sudo ufw enable
```

```
sudo apt-get update -y
sudo apt-get install     ca-certificates     curl     gnupg     lsb-release

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg
echo   "deb [arch=amd64 signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/ubuntu \
$(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

sudo apt-get update

sudo apt-get install docker-ce docker-ce-cli containerd.io -y

sudo usermod -aG docker $USER

sudo curl -L "https://github.com/docker/compose/releases/download/1.29.2/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/doc

sudo chmod +x /usr/local/bin/docker-compose

docker-compose --version
```

Dockerfile & docker-compose 로 jenkins 설치 및 Docker In Docker 설정

```
sudo mkdir -p /home/ubuntu/jenkins

cd /home/ubuntu/jenkins && sudo vi Dockerfile
```

```
# Dockerfile

FROM jenkins/jenkins:lts

USER root

# docker 설치
RUN apt-get update && \
 apt-get -y install apt-transport-https \
 ca-certificates \
 curl \
 gnupg2 \
 zip \
 unzip \
 software-properties-common && \
curl -fsSL https://download.docker.com/linux/$(. /etc/os-release; echo "$ID")/gpg > /tmp/dkey; apt-key add /tmp/dkey && \
 add-apt-repository \
 "deb [arch=amd64] https://download.docker.com/linux/$(. /etc/os-release; echo "$ID") \
 $(lsb_release -cs) \
 stable" && \
 apt-get update && \
 apt-get -y install docker-ce

 # docker-compose 설치
 RUN curl -L "https://github.com/docker/compose/releases/download/1.28.5/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/doc
    chmod +x /usr/local/bin/docker-compose && \
    ln -s /usr/local/bin/docker-compose /usr/bin/docker-compose
```

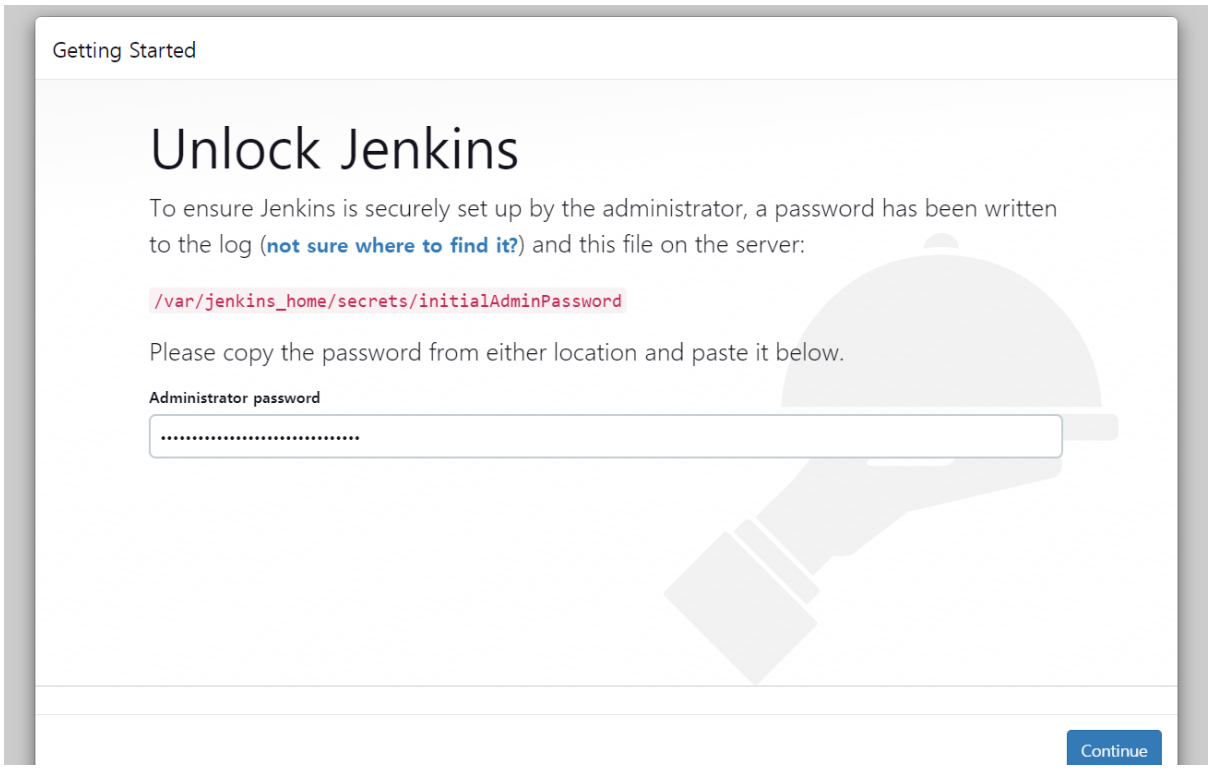
```
# docker-compose.yml

version: '3.7'

services:
  jenkins:
    build:
      context: .
      dockerfile: Dockerfile
    container_name: 'jenkins-container'
    restart: always
    user: root
    ports:
      - '8888:8080'
      - '50000:50000'
    volumes:
      - '/home/ubuntu/jenkins:/var/jenkins_home'
      - '/var/run/docker.sock:/var/run/docker.sock'
```

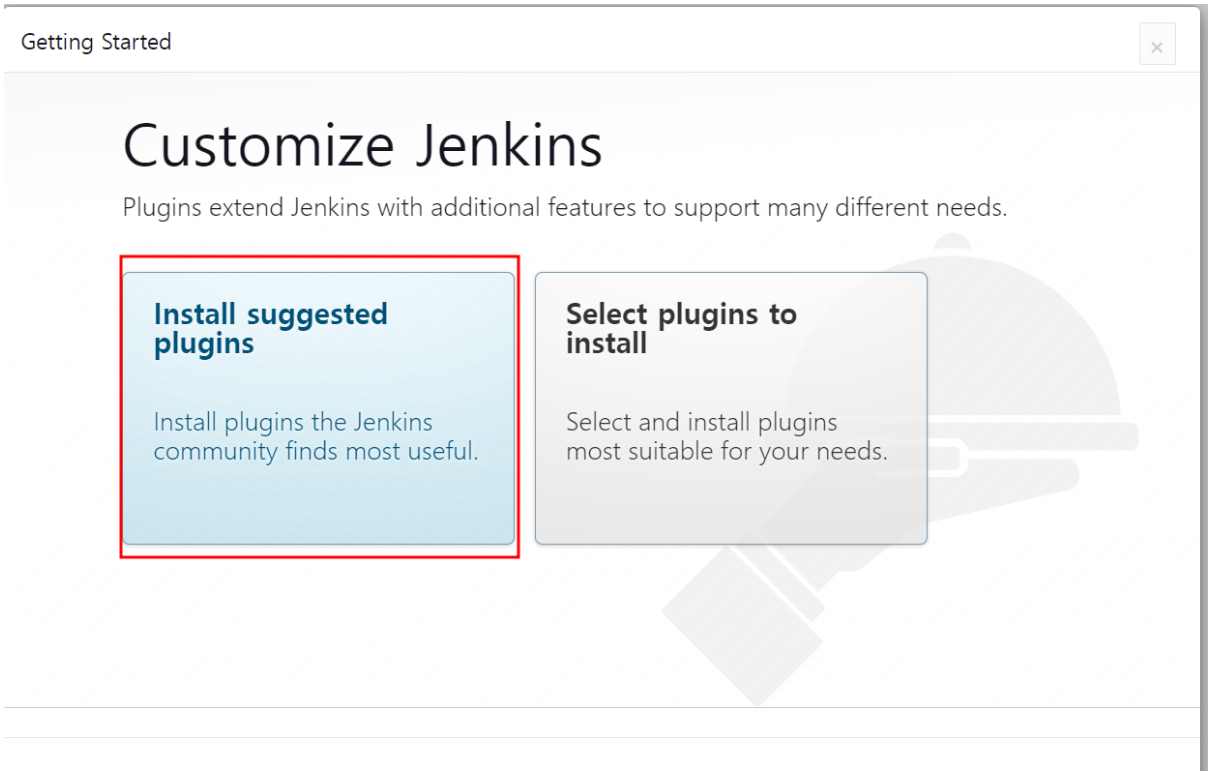
```
docker-compose up -d # 로 실행
```

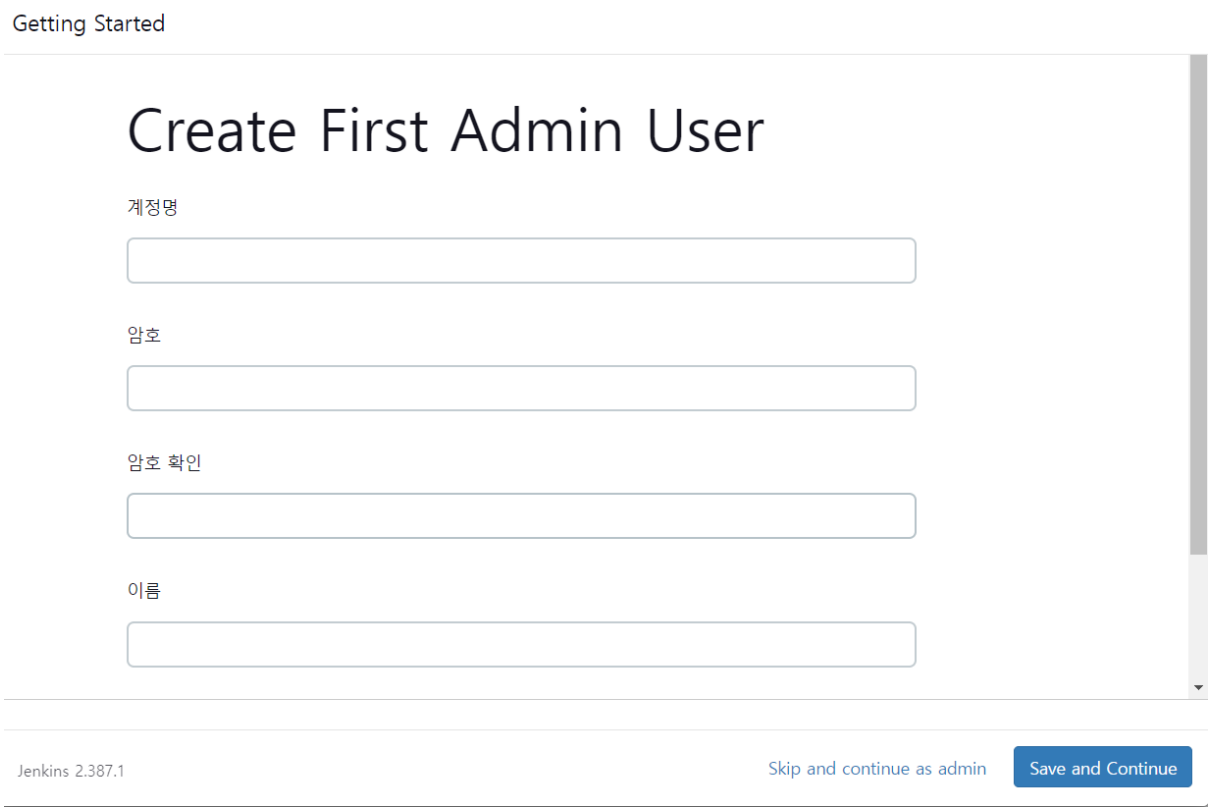
[public -ip]:8888 로 jenkins 접속

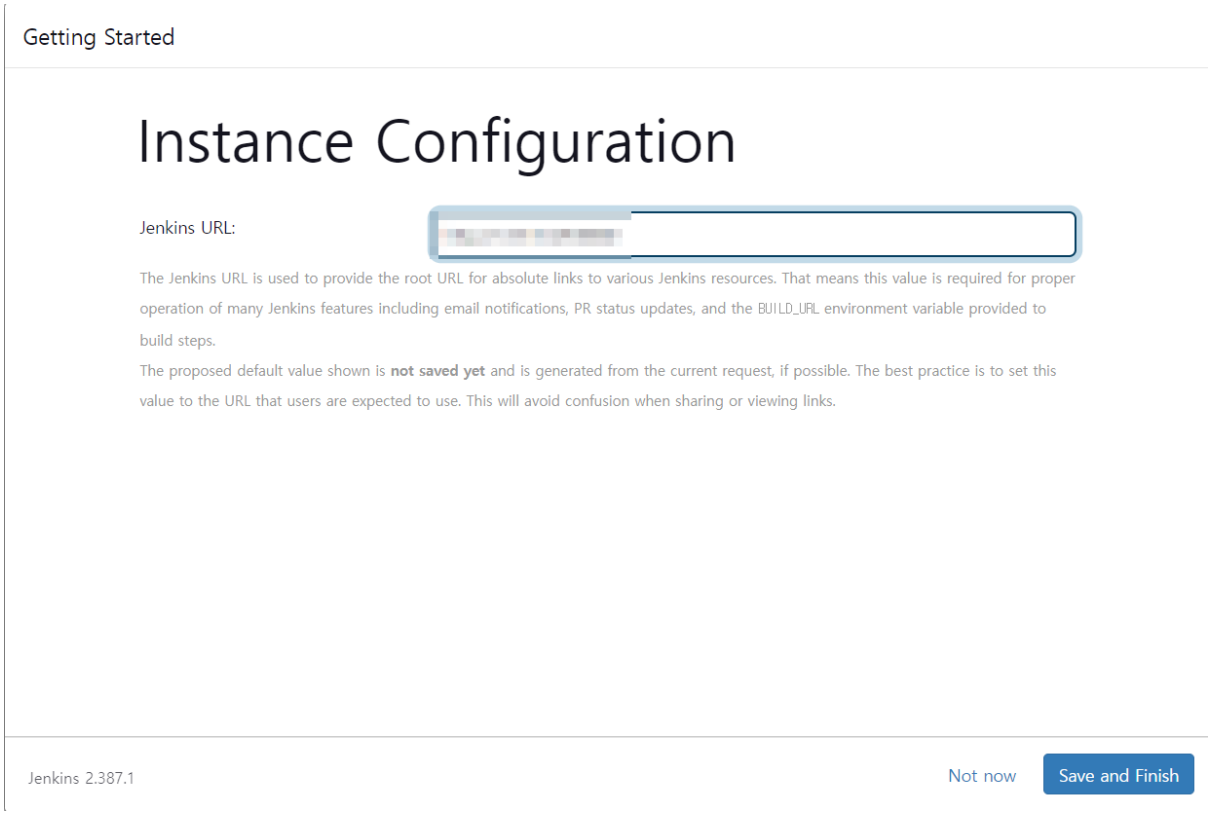
```
# jenkins-container 로그에서 비밀번호 확인
docker logs -f jenkins-container
```

```
*************************************************************
*************************************************************
*************************************************************

Jenkins initial setup is required. An admin user has been created and a password generated.
Please use the following password to proceed to installation:

[redacted]  <= 복사

This may also be found at: /var/jenkins_home/secrets/initialAdminPassword

*************************************************************
*************************************************************
*************************************************************

2023-04-04 01:50:31.923+0000 [id=29]    INFO    jenkins.InitReactorRunner$1#onAttained: Completed initialization
2023-04-04 01:50:31.971+0000 [id=22]    INFO    hudson.lifecycle.Lifecycle#onReady: Jenkins is fully up and running
2023-04-04 01:50:32.868+0000 [id=42]    INFO    h.m.DownloadService$Downloadable#load: Obtained the updated data file for hudson.tasks.Ma
ven.MavenInstaller
2023-04-04 01:50:32.869+0000 [id=42]    INFO    hudson.util.Retrier#start: Performed the action check updates server successfully at the
attempt #1
2023-04-04 01:56:09.206+0000 [id=69]    INFO    h.TcpSlaveAgentListener$ConnectionHandler#run: Connection #2 from /80.66.77.235:43684 fai
led: null
```

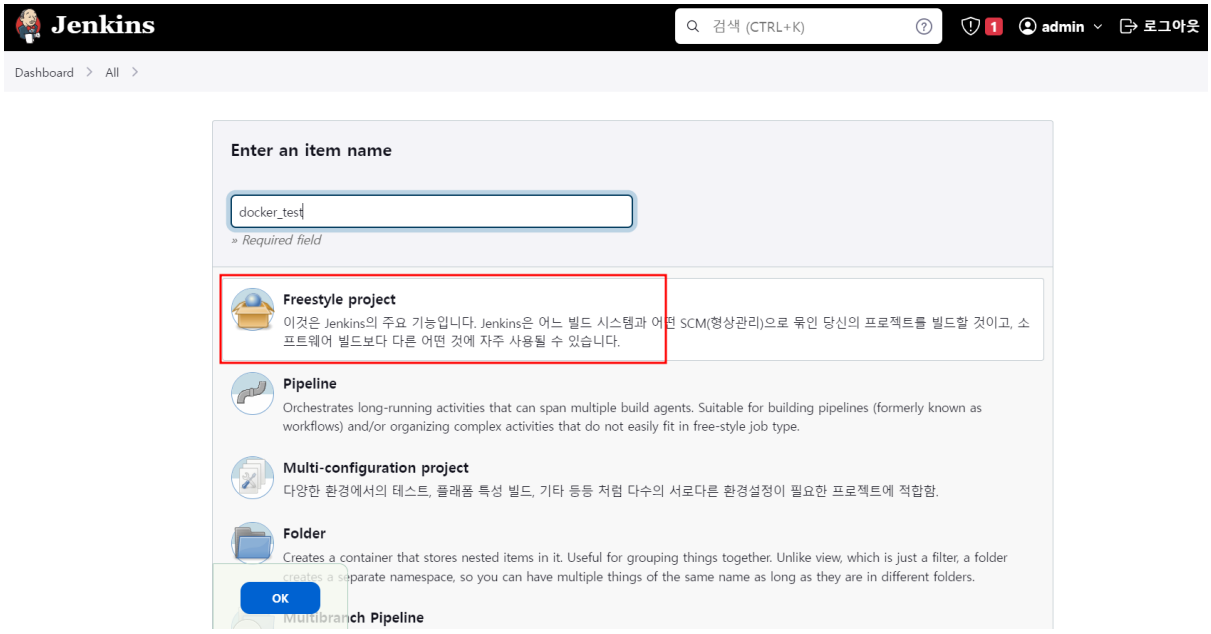


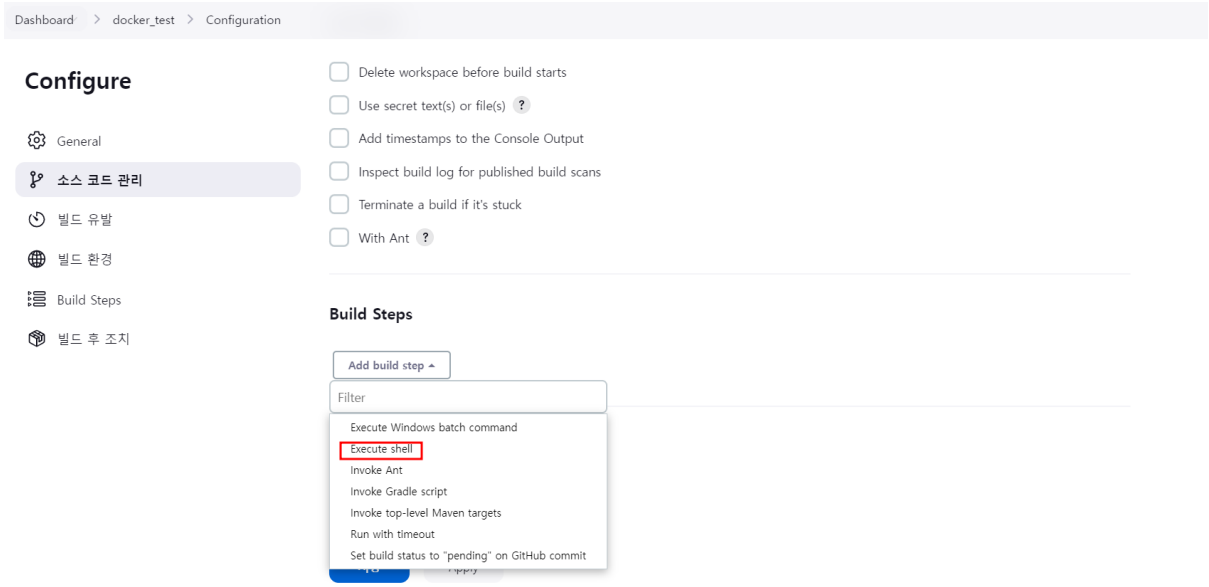

Getting Started
Customize Jenkins
Plugins extend Jenkins with additional features to support many different needs.
Install suggested plugins
Install plugins the Jenkins community finds most useful.
Select plugins to install
Select and install plugins most suitable for your needs.

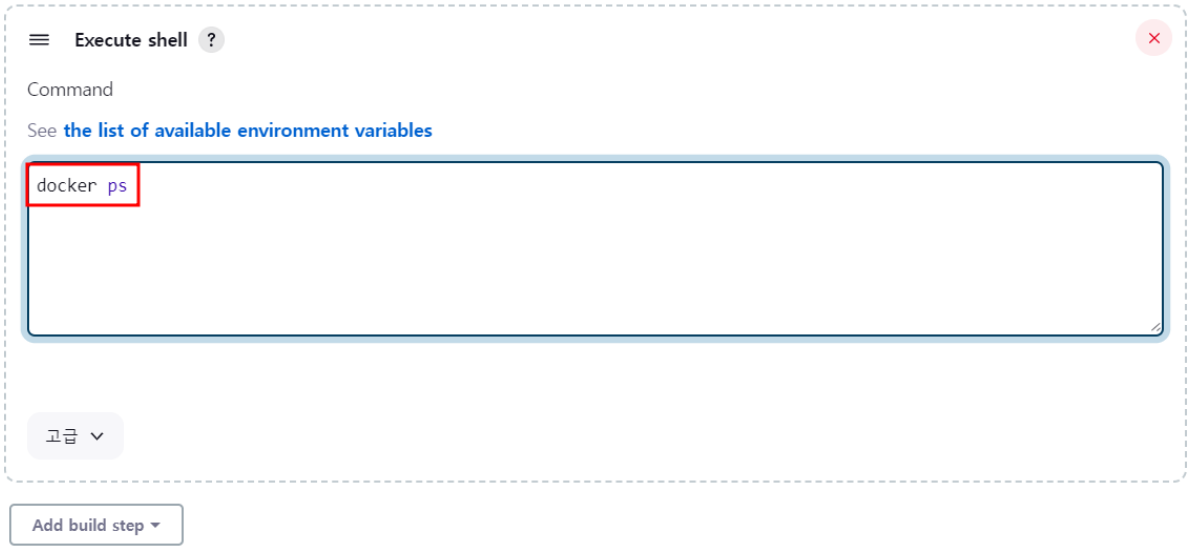

Getting Started
Create First Admin User
계정명
암호
암호 확인
이름
Jenkins 2.387.1
Skip and continue as admin
Save and Continue

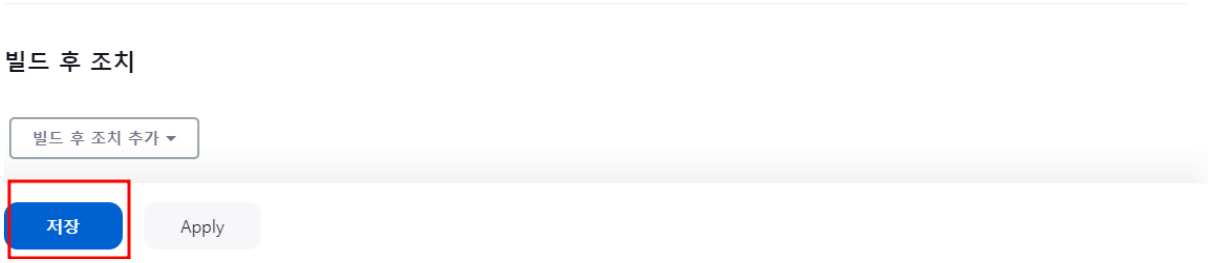


Jenkins ↔ EC2 docker.sock 연결 테스트









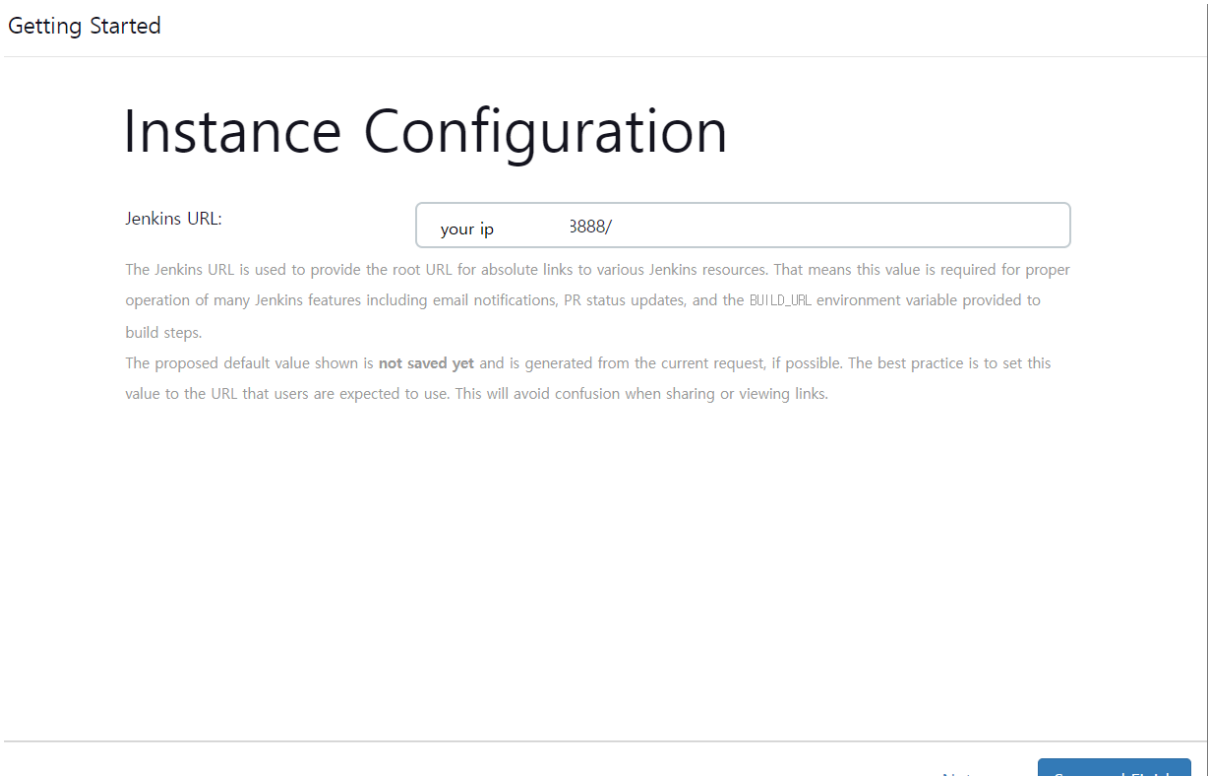
지금 빌드 후 다음 내용 Console Output 확인

콘솔 출력

```
Started by user admin
Running as SYSTEM
Building in workspace /var/jenkins_home/workspace/docker_test
[docker_test] $ /bin/sh -xe /tmp/jenkins17535912800685067018.sh
+ docker ps
CONTAINER ID   IMAGE             COMMAND                  CREATED          STATUS          PORTS
NAMES
5089ef98a246   jenkins_jenkins   "/usr/bin/tini -- /u…"   18 minutes ago   Up 18 minutes   0.0.0.0:50000->50000/tcp, :::50000-
>50000/tcp, 0.0.0.0:8888->8080/tcp, :::8888->8080/tcp   jenkins-container
Finished: SUCCESS
```

Jenkins Settings

> Ver. **Jenkins 2.387.1**



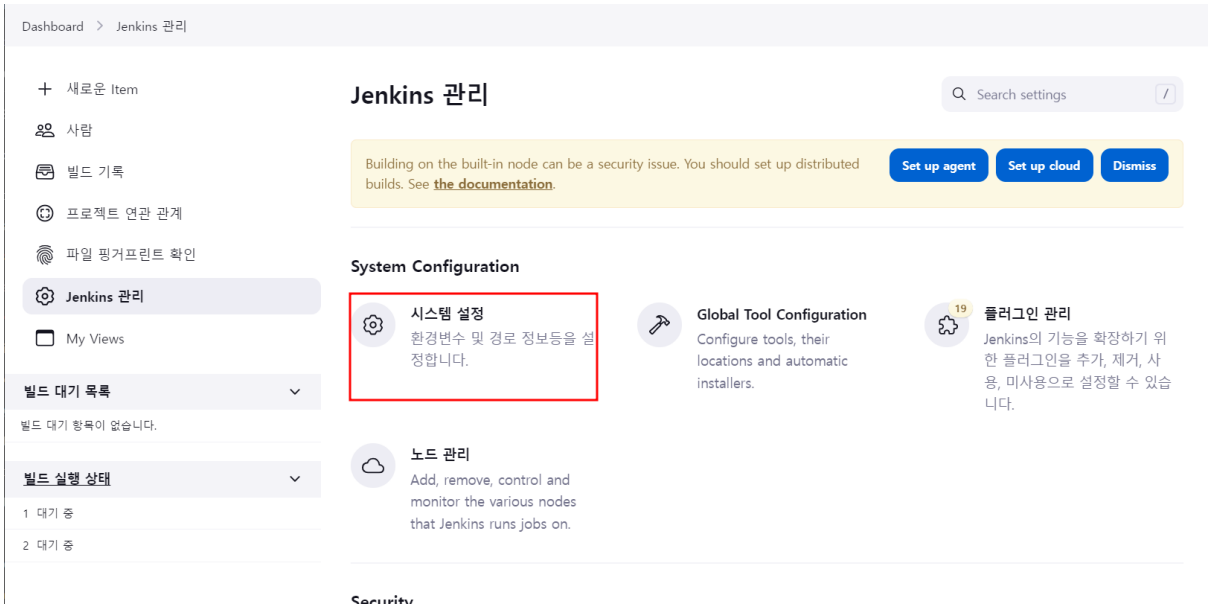
🔧초기 설정

주요 Plugins

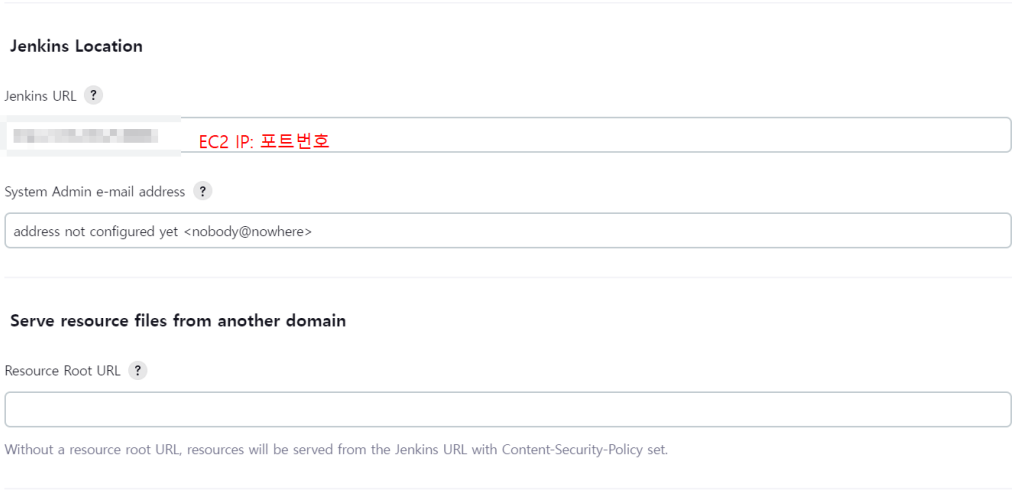
- Docker
- Gitlab
- Gradle
- Pipeline

System Configuration

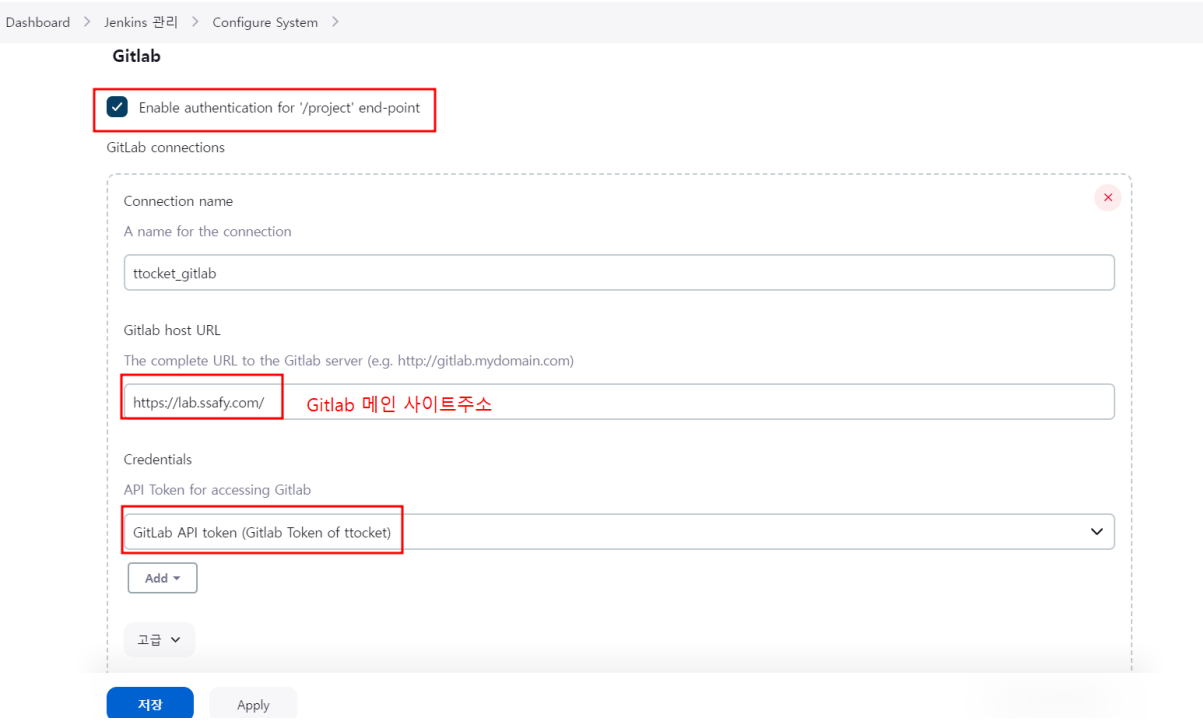
- Dashboard → Jenkins 관리



- Jenkins Location

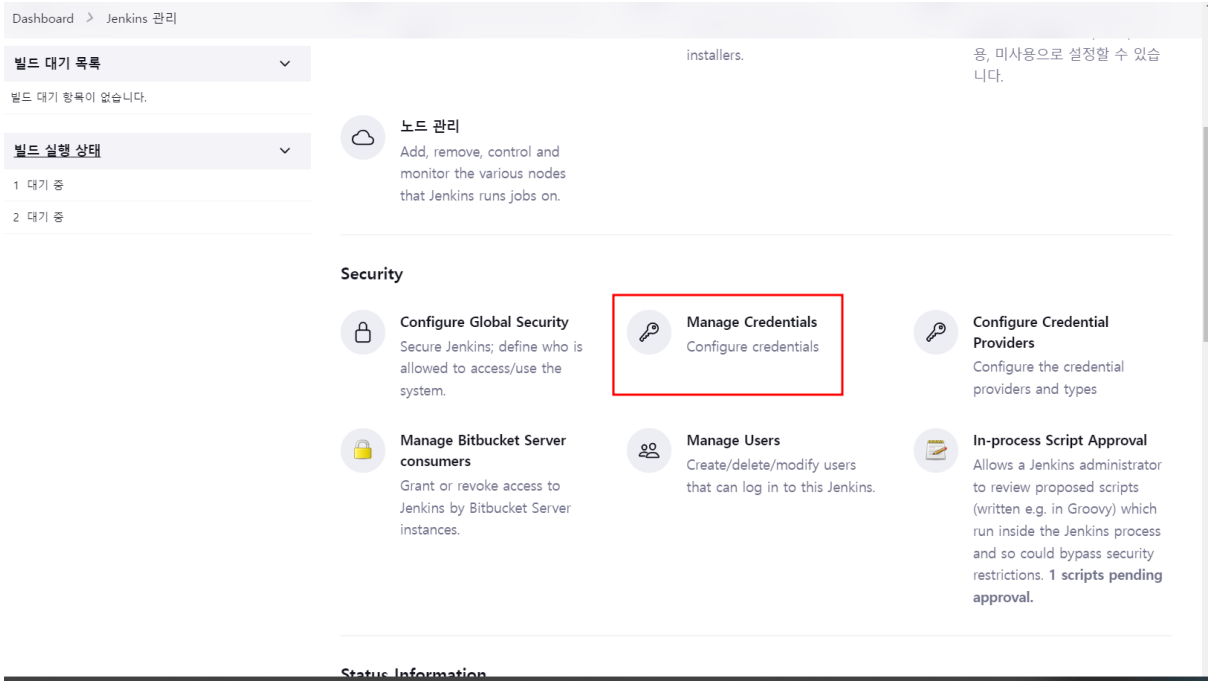


- Gitlab API token 발급

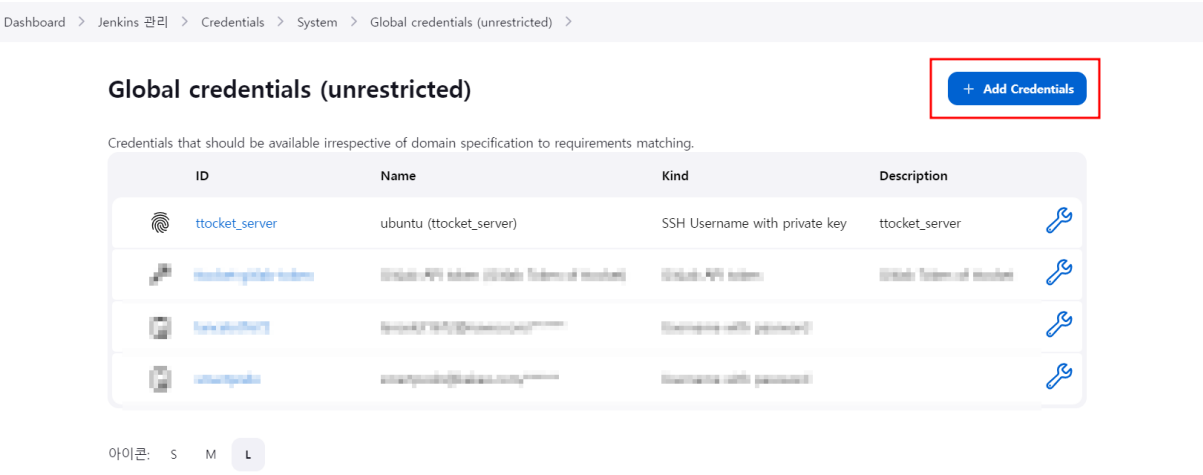


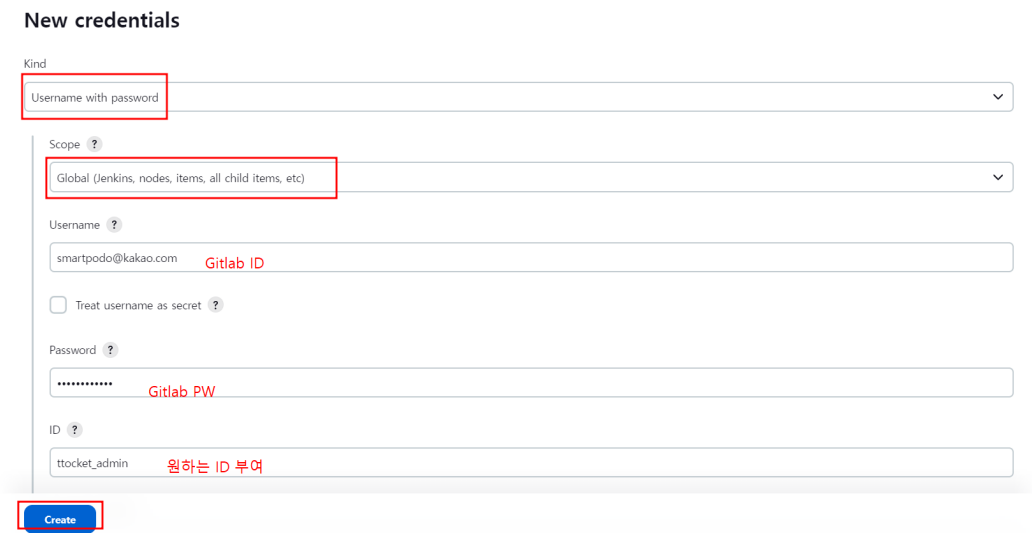
Manage Credentials

- Dashboard → Jenkins 관리



- Credentials → System → Global credentials





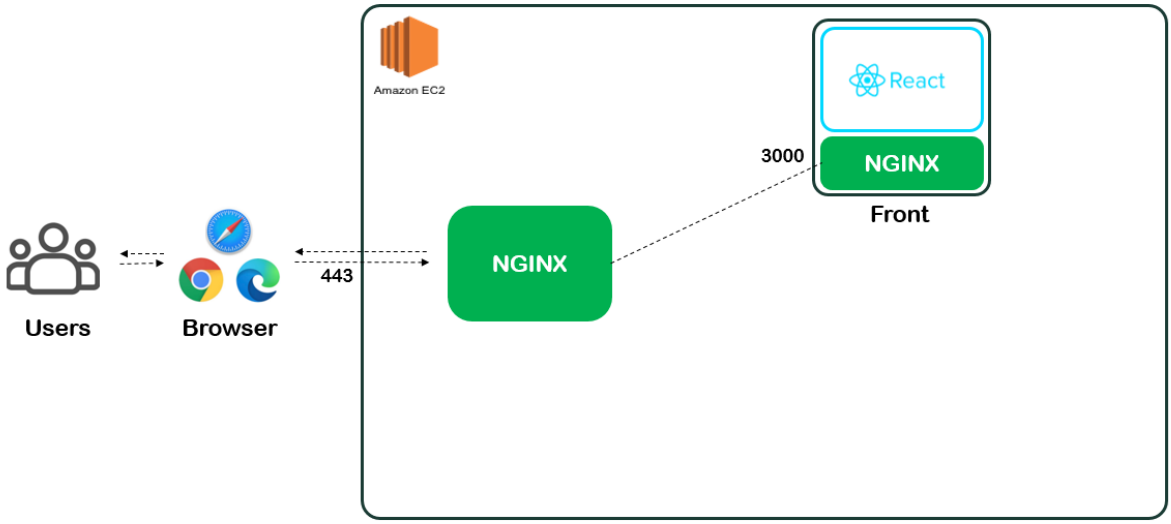
Jenkins 내에 Docker 설치 (Docker In Docker)

💡 Jenkins 파이프라인 내에서 docker 를 사용해야 하는 경우가 있기 때문에 EC2 ↔ Jenkins Container 내에 있는 Docker 를 연결하여 설정해야할 필요가 생겼습니다.

```
# jenkins-container 안에 bash shell 접속
docker exec -it jenkins-container bash

# Jenkins Container 내에 docker 설치

apt-get update && \
 apt-get -y install apt-transport-https \
 ca-certificates \
 curl \
 gnupg2 \
 zip \
 unzip \
 software-properties-common && \
curl -fsSL https://download.docker.com/linux/$(. /etc/os-release; echo "$ID")/gpg > /tmp/dkey; apt-key add /tmp/dkey && \
```

```
 add-apt-repository \
 "deb [arch=amd64] https://download.docker.com/linux/$(. /etc/os-release; echo "$ID") \
 $(lsb_release -cs) \
 stable" && \
 apt-get update && \
 apt-get -y install docker-ce


# docker-compose 설치

curl -L "https://github.com/docker/compose/releases/download/1.28.5/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-c
    chmod +x /usr/local/bin/docker-compose && \
    ln -s /usr/local/bin/docker-compose /usr/bin/docker-compose
```

NGINX



EC2 환경에 동작하는 NGINX와 React-container에서 동작하는 NGINX로 구성되어 있습니다. 위와 같은 설계에 대한 이유는 Front 포팅 메뉴얼을 참고해주세요.

- EC2로 모든 요청을 받아 처리하는 NGINX
- React 빌드 파일을 동작하는 NGINX

EC2 내에 존재하는 NGINX

```
# /etc/nginx/conf.d/default.conf

upstream frontend
{
  server localhost:3000;
}
server {
    listen 80;
```

```
    server_name j8b210.p.ssafy.io;
    return 301 https://$server_name$request_uri;

}
server {
    listen       443 ssl;
    server_name  j8b210.p.ssafy.io;

    ssl_certificate /etc/letsencrypt/live/j8b210.p.ssafy.io/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/j8b210.p.ssafy.io/privkey.pem;

    location  / {
        proxy_pass http://frontend;
    }

    location /ttocket {
        proxy_pass http://localhost:8080;

        # CORS Settings
        add_header 'Access-Control-Allow-Origin' '*' always;
        add_header 'Access-Control-Allow-Methods' 'GET, POST, PUT, DELECT';
        add_header 'Content-Type' 'application/json' always;
    }
    location /wait {
        # 웹소켓 연결을 위한 nginx 설정
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
        proxy_http_version 1.1;

        proxy_pass_request_headers on;  # 2. 요청된 헤더를 프록시하는 서버로 전달
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarder-For $proxy_add_x_forwarded_for;
        proxy_set_header Host $http_host;

        add_header 'Access-Control-Allow-Origin' '*' always;
        add_header 'Access-Control-Allow-Methods' '*';

        proxy_pass http://localhost:9001;
    }

    location = /ganache {
        proxy_pass http://localhost:8545;
    }
    location /webui {
        proxy_pass http://localhost:5001;
    }
    location /ipfs {
        proxy_pass http://localhost:5001;
    }
    location /api {
        proxy_pass http://localhost:5001;
    }
}
```

react-container 내에 존재하는 NGINX

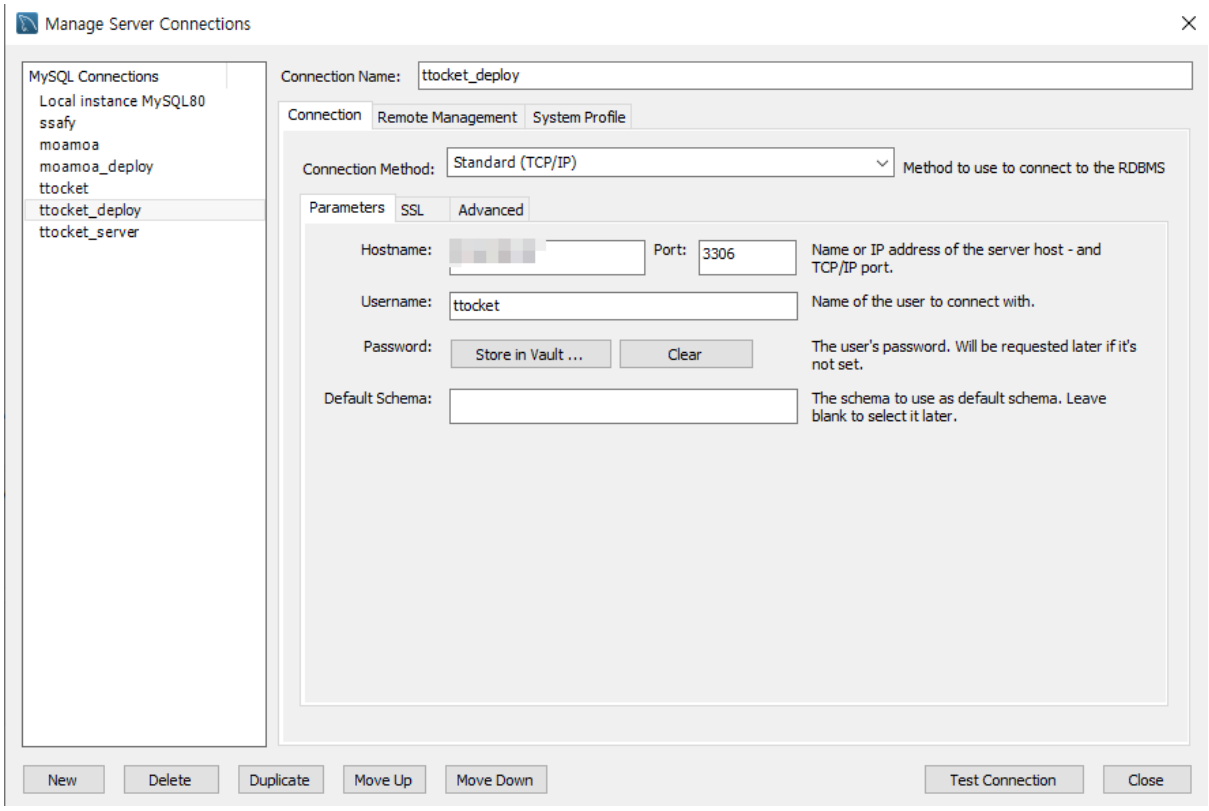
```
# /etc/nginx/conf.d/react.conf

server {
    listen 3000;

    location / {
        # React build 시 해당 경로에 빌드된
        root /usr/share/nginx/html;

        index index.html index.htm;

        # React Routing 할 때 페이지간 이동을 할 수 있게 하기 위한 설정
        try_files $uri $uri/ /index.html;
    }
}
```

MySQL

- mysql user list

username	password	port
root		3306
ttocket		3306

Workbench 에서 접속




```
# EC2 안 에서 volume 을 마운트 할 폴더 생성
sudo mkdir -p /var/lib/mysql

# 3306 포트 안에서 mysql-container 라는 이름으로 /var/lib/mysql 안에 volume 설정
sudo docker run --name mysql-container -p 3306:3306 -v /var/lib/mysql:/var/lib/mysql -e MYSQL_ROOT_PASSWORD="비밀번호", -d mysql

# mysql-container 안에 존재하는 bash 접속
docker exec -it mysql-container bash

# mysql cli 설정 root 로 접속
mysql -u root -p

# mysql table로 변경
use mysql;

# 사용자 정보 조회
SELECT HOST,USER FROM user;

# username ttocket 에 password !23Qwe 라는 사용자 생성
```

```
CREATE USER 'ttocket'@'%' identified by '비밀번호';

# ttocket 사용자에게 ttocket 스키마에 대한 모든 권한 부여
GRANT ALL PRIVILEGES ON TTOCKET.* TO 'ttocket'@'%';
```

Redis

Redis 하나만 올렸을 때

- **EC2 ubuntu 에서**

```
# Redis 설정파일 및 Dockerfile 들을 놓을 폴더 생성
sudo mkdir redis

# Redis 설정파일들 생성
sudo touch redis.conf && sudo touch Dockerfile
```

```
# Dockerfile

FROM redis:latest
COPY redis.conf /usr/local/etc/redis/redis.conf
CMD ["redis-server", "/usr/local/etc/redis/redis.conf"]
```

```
# redis.conf
# 비밀번호 설정
requirepass = "비밀번호"
```

```
# Dockerfile 기반으로 이미지 생성
docker build -t {이미지 이름} .

# 생성된 이미지 6379 포트로 설정하고 redis.conf에 적힌 password 로 Authentication 후 실행
docker run --name redis-container -p 6379:6379 -d {이미지 이름} redis-server /usr/local/etc/redis/redis.conf --requirepass "비밀번호"

# Redis 컨테이너에 비밀번호로 redis-cli 접속
docker exec -it redis-container redis-cli -a "비밀번호"
```

Redis Sentinel 구축

Ports

- Master - 6379
  - Slaves- 6380, 6381
  - Sentinels - 26379 - 26381

```
# docker-compose.yml

version: '3'

services:
  redis-master:
    container_name: redis-master
    image: 'bitnami/redis:latest'
    environment:
      - REDIS_REPLICATION_MODE=master
      - REDIS_PASSWORD="master 비밀번호"
    networks:
```

```
      - redis-network
    ports:
      - '6379:6379'

  redis-slave-1:
    container_name: redis-slave-1
    image: 'bitnami/redis:latest'
    environment:
      - REDIS_REPLICATION_MODE=slave
      - REDIS_MASTER_HOST=redis-master
      - REDIS_MASTER_PASSWORD="비밀번호"
      - REDIS_PASSWORD="slave 비밀번호"
    ports:
      - '6380:6379'
    depends_on:
      - redis-master
    networks:
      - redis-network

  redis-slave-2:
    container_name: redis-slave-2
    image: 'bitnami/redis:latest'
    environment:
      - REDIS_REPLICATION_MODE=slave
      - REDIS_MASTER_HOST=redis-master
      - REDIS_MASTER_PASSWORD="master 비밀번호"
      - REDIS_PASSWORD="slave 비밀번호"
    ports:
      - '6381:6379'
    depends_on:
      - redis-master
    networks:
      - redis-network
  redis-sentinel-1:
    container_name: redis-sentinel-1
    image: 'bitnami/redis-sentinel:latest'
    environment:
      - REDIS_MASTER_PASSWORD="master 비밀번호"
      - REDIS_SENTINEL_DOWN_AFTER_MILLISECONDS=3000
      - REDIS_MASTER_HOST=redis-master
      - REDIS_MASTER_PORT_NUMBER=6379
      - REDIS_MASTER_SET=mymaster
      - REDIS_SENTINEL_QUORUM=2
    depends_on:
      - redis-master
      - redis-slave-1
      - redis-slave-2
    ports:
      - '26379:26379'
    networks:
      - redis-network

  redis-sentinel-2:
    container_name: redis-sentinel-2
    image: 'bitnami/redis-sentinel:latest'
    environment:
      - REDIS_MASTER_PASSWORD="master 비밀번호"
      - REDIS_SENTINEL_DOWN_AFTER_MILLISECONDS=3000
      - REDIS_MASTER_HOST=redis-master
      - REDIS_MASTER_PORT_NUMBER=6379
      - REDIS_MASTER_SET=mymaster
      - REDIS_SENTINEL_QUORUM=2
    depends_on:
      - redis-master
      - redis-slave-1
      - redis-slave-2
    ports:
      - '26380:26379'
    networks:
      - redis-network

  redis-sentinel-3:
    container_name: redis-sentinel-3
    image: 'bitnami/redis-sentinel:latest'
    environment:
      - REDIS_MASTER_PASSWORD="master 비밀번호"
      - REDIS_SENTINEL_DOWN_AFTER_MILLISECONDS=3000
      - REDIS_MASTER_HOST=redis-master
      - REDIS_MASTER_PORT_NUMBER=6379
      - REDIS_MASTER_SET=mymaster
      - REDIS_SENTINEL_QUORUM=2
    depends_on:
      - redis-master
      - redis-slave-1
      - redis-slave-2
    ports:
      - '26381:26379'
```

```
    networks:
      - redis-network
networks:
  redis-network:
          external: true
```

❗Redis - Spring Connection timeout 에러 발생

> Redis Sentinel들을 구축한 상태에서 Spring 설정들도 맞췄는데 Spring 에서 Redis를 연결할 수 없었다…

```
io.netty.channel.ConnectTimeoutException: connection timed out: /172.21.0.2:6379
        at io.netty.channel.nio.AbstractNioChannel$AbstractNioUnsafe$1.run(AbstractNioChannel.java:261) ~[netty-transport-4.1.89.Final
        at io.netty.util.concurrent.PromiseTask.runTask(PromiseTask.java:98) ~[netty-common-4.1.89.Final.jar!/:4.1.89.Final]
```

```
docker compose up -d

# Redis 와 Redis Sentinels 상태 확인

docker exec -it redis-master redis-cli -a "비밀번호"
> 127.0.0.1:6379 info

# Sentinel들 로그 확인

docker logs -f redis-sentinel-1

docker exec -it redis-sentinel-1 redis-cli -p 26379 -a "비밀번호"

> 127.0.0.1:26379: sentinel masters

# 이 외에 redis-master 를 내려봐서 redis-slave 가 master 로 승격되는 것 확인
```

💡 Redis 는 redis-network에 연결되어 있고, Spring은 따로 네트워크 설정을 하지 않아서 배포된 spring boot 에서 172.21.0.2:6379 를 찾을 수 없는 문제가 있다. 즉 , Spring과 Redis의 네트워크가 ip 매핑을 할 수 없었다.

Solutions

```
# Spring 과 Redis 를 같은 네트워크로 묶어준다.
docker network connect redis-network spring-container
```

IPFS

IPFS란??

Inter**P**lanetary **F**ile **S**ystem

분산형 파일 시스템에 데이터를 저장하고 인터넷으로 공유하기 위한 프로토콜이다. 우리가 흔히 아는 토렌트(Torrent) 등 P2P 방식으로 대용량 파일과 데이터를 공유하기 위해 사용한다.

docker image

- ipfs/go-ipfs:latest
- ipfs/ipfs-cluster:latest

IPFS Port

4001	다른 노드와 통신
5001	API 서버
8080	게이트웨이 서버

IPFS-CLUSTER

9094	HTTP API 엔드 포인트
9095	IPFS 프록시 엔드 포인트
9096	클러스터 노드 간 통신에 사용되는 클러스터들

Docker를 이용한 IPFS 클러스터 구축

각 ipfs-cluster는 ipfs 노드 하나씩을 바라보고 있으며 docker-compose를 통해 클러스터를 구축한다. docker-compose.yml 파일을 아래와 같이 작성한다.

▼ docker-compose.yml

```
version: '3.4'

services:
  ipfs0:
    container_name: ipfs0
    image: ipfs/go-ipfs:latest
    ports:
      - "4001:4001" # ipfs swarm - expose if needed/wanted
      - "5001:5001" # ipfs api - expose if needed/wanted
      - "8080:8080" # ipfs gateway - expose if needed/wanted
    volumes:
      - ./compose/ipfs0:/data/ipfs

  cluster0:
    container_name: cluster0
    image: ipfs/ipfs-cluster:latest
    depends_on:
      - ipfs0
    environment:
      CLUSTER_PEERNAME: cluster0
      CLUSTER_SECRET: ${CLUSTER_SECRET} # From shell variable if set
      CLUSTER_IPFSHTTP_NODEMULTIADDRESS: /dns4/ipfs0/tcp/5001
      CLUSTER_CRDT_TRUSTEDPEERS: '*' # Trust all peers in Cluster
      CLUSTER_RESTAPI_HTTPLISTENMULTIADDRESS: /ip4/0.0.0.0/tcp/9094 # Expose API
      CLUSTER_MONITORPINGINTERVAL: 2s # Speed up peer discovery
    ports:
          - "127.0.0.1:9094:9094"
          # - "9096:9096" # Cluster IPFS Proxy endpoint
    volumes:
      - ./compose/cluster0:/data/ipfs-cluster
  ipfs1:
    container_name: ipfs1
    image: ipfs/go-ipfs:latest
    volumes:
      - ./compose/ipfs1:/data/ipfs

  cluster1:
    container_name: cluster1
    image: ipfs/ipfs-cluster:latest
    depends_on:
      - ipfs1
    environment:
      CLUSTER_PEERNAME: cluster1
      CLUSTER_SECRET: ${CLUSTER_SECRET}
      CLUSTER_IPFSHTTP_NODEMULTIADDRESS: /dns4/ipfs1/tcp/5001
      CLUSTER_CRDT_TRUSTEDPEERS: '*'
      CLUSTER_MONITORPINGINTERVAL: 2s # Speed up peer discovery
    volumes:
      - ./compose/cluster1:/data/ipfs-cluster

  ipfs2:
    container_name: ipfs2
```

```
    image: ipfs/go-ipfs:latest
    volumes:
      - ./compose/ipfs2:/data/ipfs

  cluster2:
    container_name: cluster2
    image: ipfs/ipfs-cluster:latest
    depends_on:
      - ipfs2
    environment:
      CLUSTER_PEERNAME: cluster2
      CLUSTER_SECRET: ${CLUSTER_SECRET}
      CLUSTER_IPFSHTTP_NODEMULTIADDRESS: /dns4/ipfs2/tcp/5001
      CLUSTER_CRDT_TRUSTEDPEERS: '*'
      CLUSTER_MONITORPINGINTERVAL: 2s # Speed up peer discovery
    volumes:
      - ./compose/cluster2:/data/ipfs-cluster
```

Docker를 통해 IPFS를 구축하기전, 다음 명령어를 실행하여 Secret Key를 생성하여 네트워크 피어에 비밀키를 공유하는 피어와 통신을 하게해야합니다.

```
od  -vN 32 -An -tx1 /dev/urandom | tr -d ' \n' && echo ""
>> 74b914e9a54a699489b12a7e15293a19a2129897ad7277f79fd18c65f66c60b9
```

`~/.bashrc` 에 `CLUSTER_SECRET` 값을 저장

```
sudo vi ~/.bashrc

# 밑 부분을 추가
export CLUSTER_SECRET="74b914e9a54a699489b12a7e15293a19a2129897ad7277f79fd18c65f66c60b9"

source ~/.bashrc
```

출력 값을 docker-compose 파일의 CLUSTER_SECRET 부분의 값으로 매핑한다.

```
environment:
  CLUSTER_PEERNAME: cluster0
  CLUSTER_SECRET: ${CLUSTER_SECRET}
```

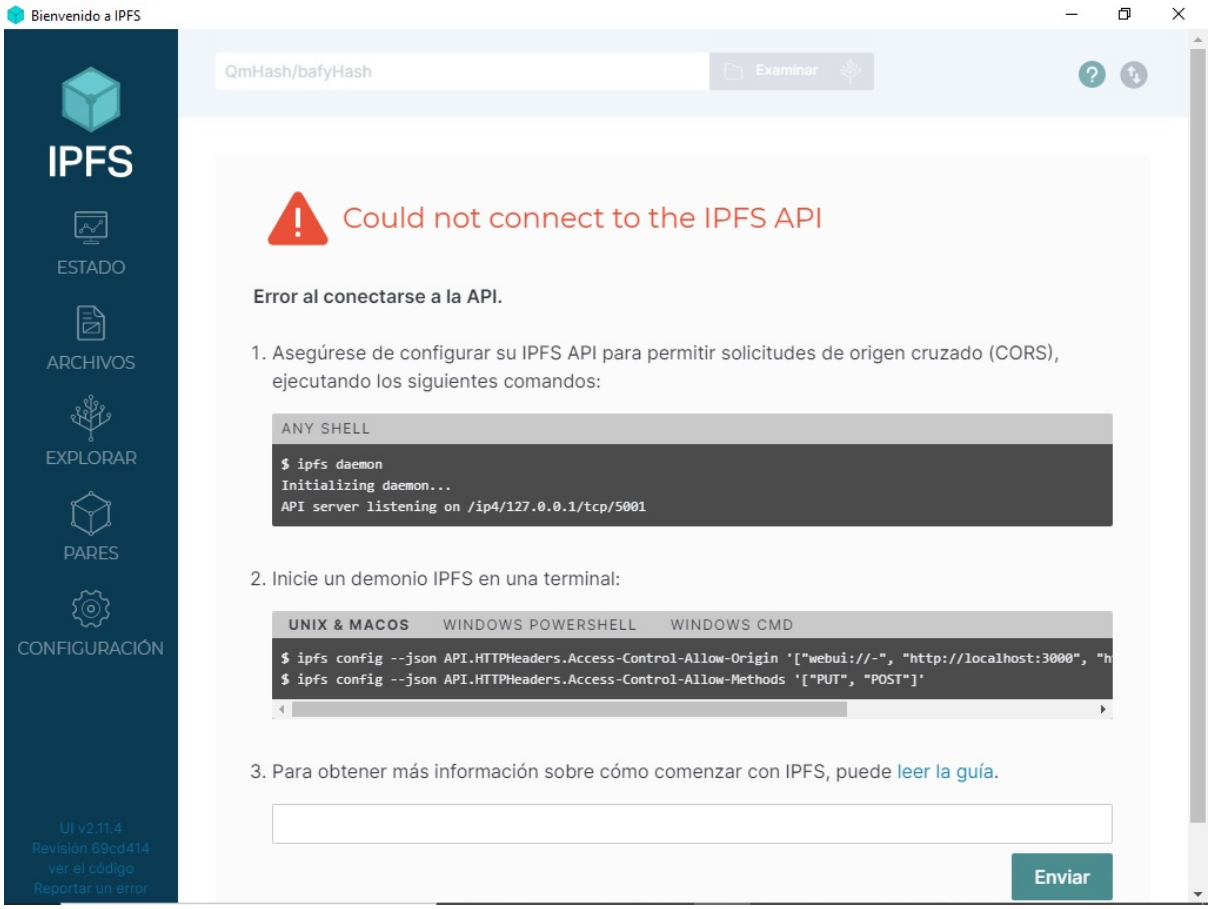
컨테이너 실행

```
docker-compose up -d
```

컨테이너가 잘 실행되었다면 아래와 같이 IPFS 3개, IPFS-CLUSTER 3개가 동작 중인 것을 확인할 수 있다.

```
6e28f29637ba   ipfs/ipfs-cluster:latest          "/sbin/tini -- /usr/…"   8 days ago      Up 7 days              9094-9096/tcp
                                                                                                                 cluster2
704c4d5ad152   ipfs/ipfs-cluster:latest          "/sbin/tini -- /usr/…"   8 days ago      Up 7 days              127.0.0.1:9094->9094/tcp, 9095-9096/tcp
                                                                                                                 cluster0
aacc8ed3aecc   ipfs/ipfs-cluster:latest          "/sbin/tini -- /usr/…"   8 days ago      Up 7 days              9094-9096/tcp
                                                                                                                 cluster1
db334ab0dfd1   ipfs/go-ipfs:latest               "/sbin/tini -- /usr/…"   8 days ago      Up 7 days (healthy)    0.0.0.0:4001->4001/tcp, :::4001->4001/tcp, 0.0.0.0:5001->5001/tcp, :::5001->5
001/tcp, 4001/udp, 0.0.0.0:8080->8080/tcp, :::8080->8080/tcp, 8081/tcp                                           ipfs0
7afc2828339f   ipfs/go-ipfs:latest               "/sbin/tini -- /usr/…"   8 days ago      Up 7 days (healthy)    4001/tcp, 5001/tcp, 8080-8081/tcp, 4001/udp
                                                                                                                 ipfs1
e719b30b9942   ipfs/go-ipfs:latest               "/sbin/tini -- /usr/…"   8 days ago      Up 7 days (healthy)    4001/tcp, 5001/tcp, 8080-8081/tcp, 4001/udp
                                                                                                                 ipfs2
```

https://[www.your-domain.com]:5001/webui 로 접속한다면 IPFS의 대쉬보드가 나타나게 되는데 CORS 설정을 추가적으로 해줘야 한다.



IPFS의 설정 파일은 Docker 컨테이너 구축 당시 연결했던 Volume 밑에 존재한다.

```
vi ./compose/ipfs0/config
vi ./compose/ipfs1/config
vi ./compose/ipfs2/config
```

설정 파일의 API 부분과 Gateway 부분을 아래와 같이 CORS 설정을 해준다.

```
"API": {
    "HTTPHeaders": {
      "Access-Control-Allow-Methods": [
        "PUT",
        "GET",
        "POST"
      ],
      "Access-Control-Allow-Origin": [
        "*"
      ]
    }
}
```

```
"Gateway": {
    "APICommands": [],
    "HTTPHeaders": {
      "Access-Control-Allow-Headers": [
        "X-Requested-With",
        "Range",
        "User-Agent"
      ],
      "Access-Control-Allow-Methods": [
        "*"
      ],
      "Access-Control-Allow-Origin": [
        "*"
      ]
    },
    "NoDNSLink": false,
    "NoFetch": false,
    "PathPrefixes": [],
    "PublicGateways": null,
    "RootRedirect": ""
},
```

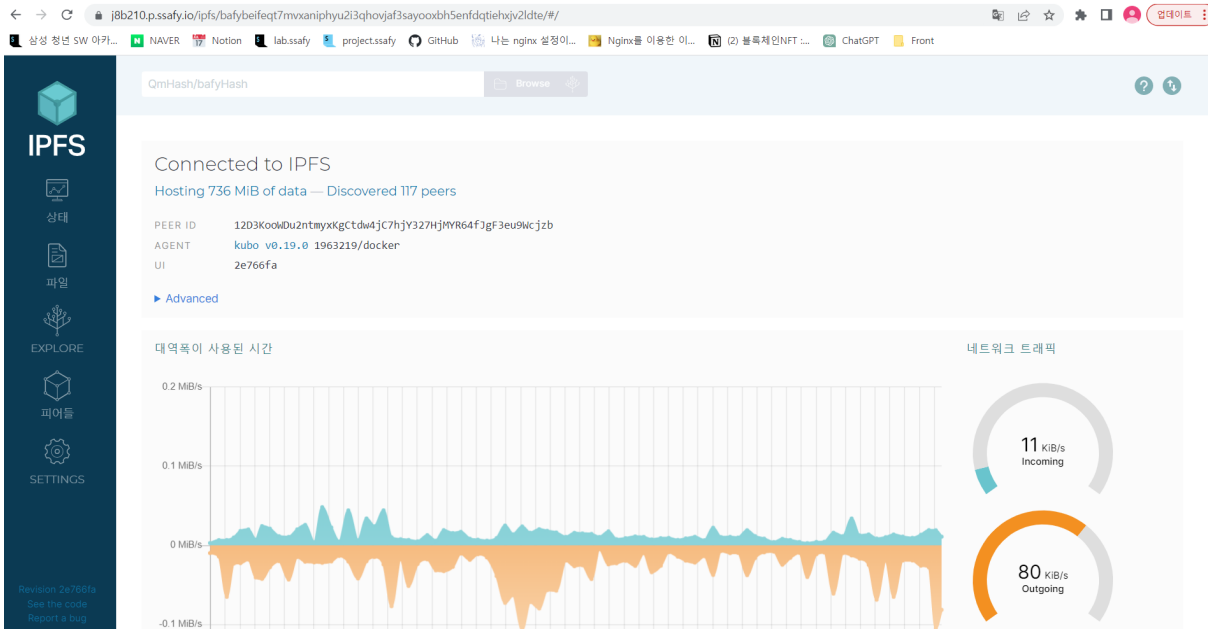
nginx 설정

nginx의 location 설정에도 다음과 같은 내용을 추가한다.

```
location /webui {
    proxy_pass http://localhost:5001;
}
location /ipfs {
    proxy_pass http://localhost:5001;
}
location /api {
    proxy_pass http://localhost:5001;
}
```

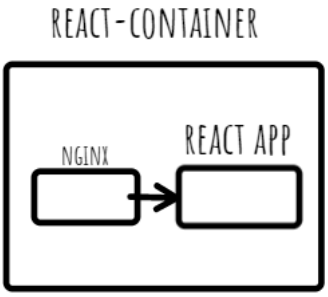
설정 파일의 수정이 완료되었다면 IPFS Container를 다시 실행시켜준다.

아래와 같이 Connected to IPFS 라고 안내가 뜨게 된다면 설정이 완료된 것이다.


j8b210.p.ssafy.io/ipfs/bafybeifeqt7mvxaniphyu2i3qhovjaf3sayooxbh5enfdqtiehxjv2ldte/#/
업데이트
삼성 청년 SW 아카...
NAVER
Notion
lab.ssafy
project.ssafy
GitHub
나는 nginx 설정이...
Nginx를 이용한 이...
(2) 블록체인NFT :...
ChatGPT
Front
QmHash/bafyHash
Browse
IPFS
상태
파일
EXPLORE
피어들
SETTINGS
Revision 2e766fa
See the code
Report a bug
Connected to IPFS
Hosting 736 MiB of data — Discovered 117 peers
PEER ID 12D3KooWDu2ntmyxKgCtdw4jC7hjY327HjMYR64fJgF3eu9Wcjzb
AGENT kubo v0.19.0 1963219/docker
UI 2e766fa
Advanced
대역폭이 사용된 시간
0.2 MiB/s
0.1 MiB/s
0 MiB/s
-0.1 MiB/s
네트워크 트래픽
11 KiB/s
Incoming
80 KiB/s
Outgoing

React

react-container 설정

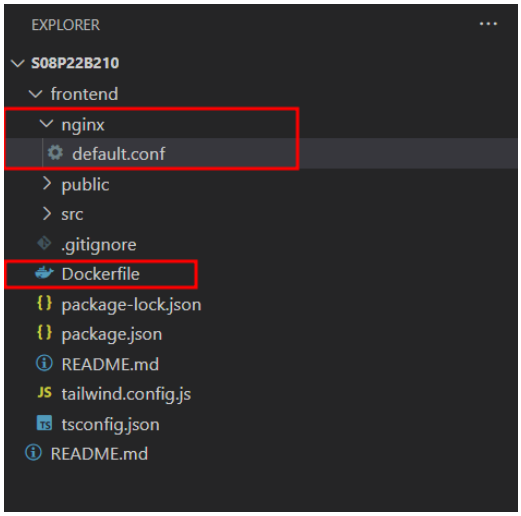


react-container 내에 React app 과 NGINX 를 두어서 EC2 내에 NGINX와 비슷하게 로드밸런싱 역할을 진행하게 설정해주었습니다. → EC2 내의 NGINX 렌더링 보다 한 컨테이너 안에 담는게 더 렌더링 속도도 빠른 이유도 존재

수동배포

로컬에서

- **nginx/default.conf** 파일과 **Dockerfile** 을 생성



```
# Dockerfile

FROM node:alpine as builder

WORKDIR /app

COPY ./package.json ./

RUN npm install

COPY . .

RUN npm run build

FROM nginx

EXPOSE 3000

COPY ./nginx/default.conf /etc/nginx/conf.d/react.conf

COPY --from=builder /app/build /usr/share/nginx/html
```

```
# nginx/default.conf

server {
    listen 3000;

    location / {

        root /usr/share/nginx/html;

        index index.html index.htm;

        # React Routing 할 때 페이지간 이동을 할 수 있게 하기 위한 설정
        try_files $uri $uri/ /index.html;
    }
}
```

- Local → Docker Hub → EC2

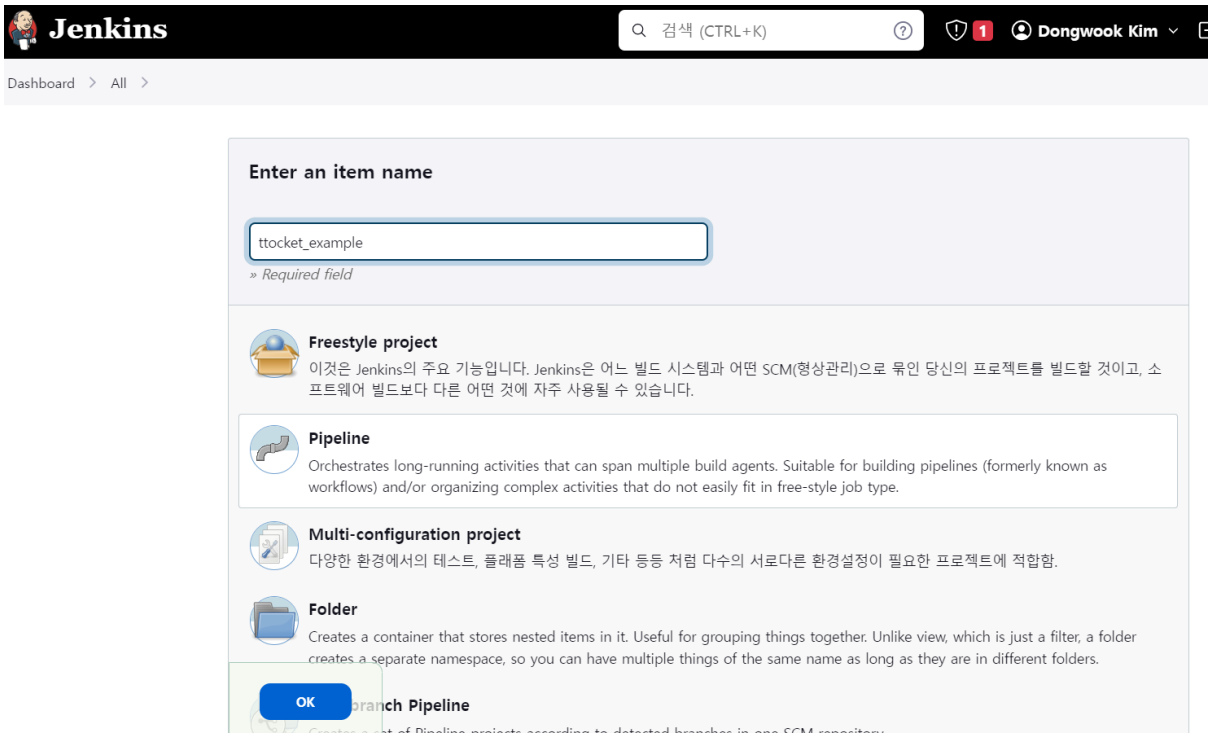
```
# 수동 배포 과정

docker build -t { image name } .

docker login


# 로컬에서 테스트

docker run --name {container name} -d -p 3000:3000 {image name}

docker logs -f { container name }


# Docker hub 에 push
docker push { image name }
```

EC2에서

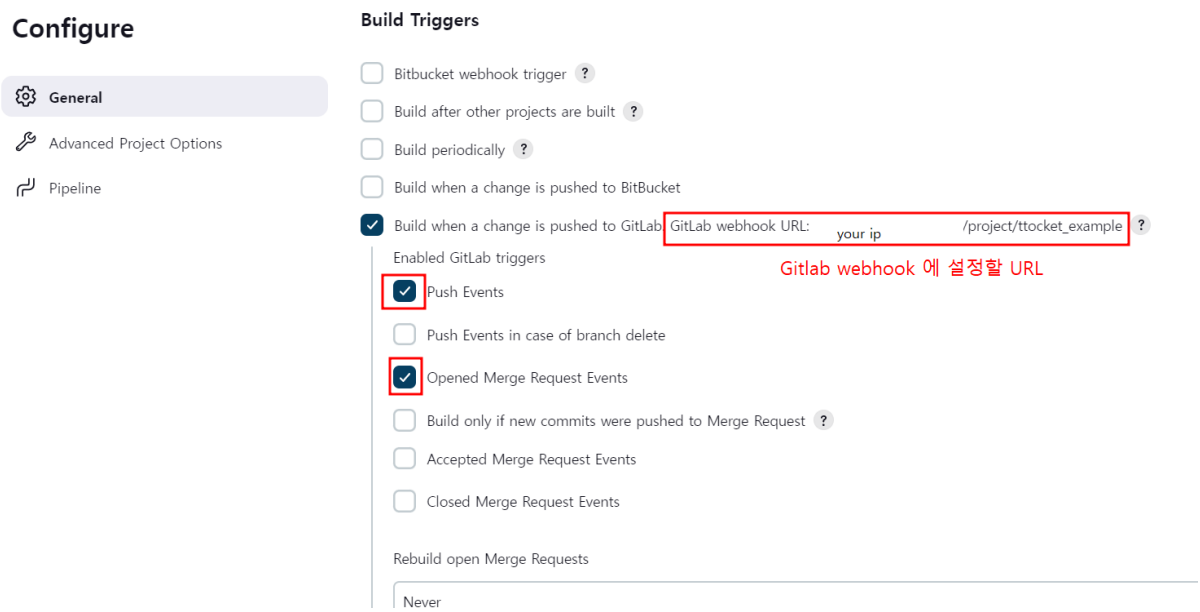
```
docker run --name {container name} -d -p 3000:3000 {image name}

# 로그 확인
docker logs -f { container name }
```

React 자동배포

- Dashboard → 새로운 item → Pipeline 체크

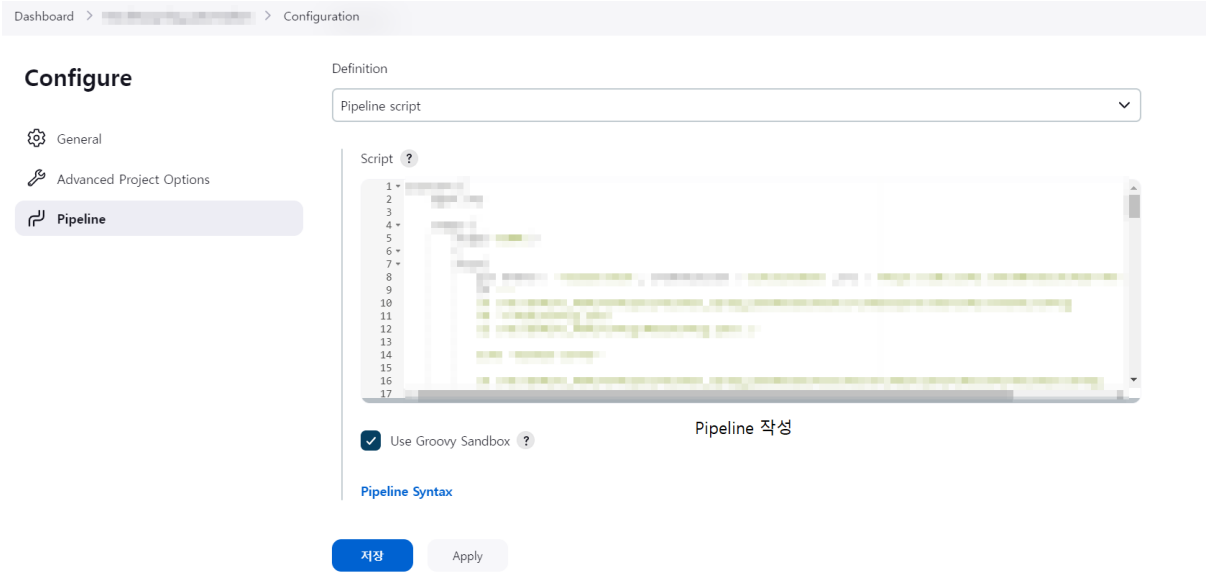


- Build Triggers - Webhook 설정을 해주기 위함



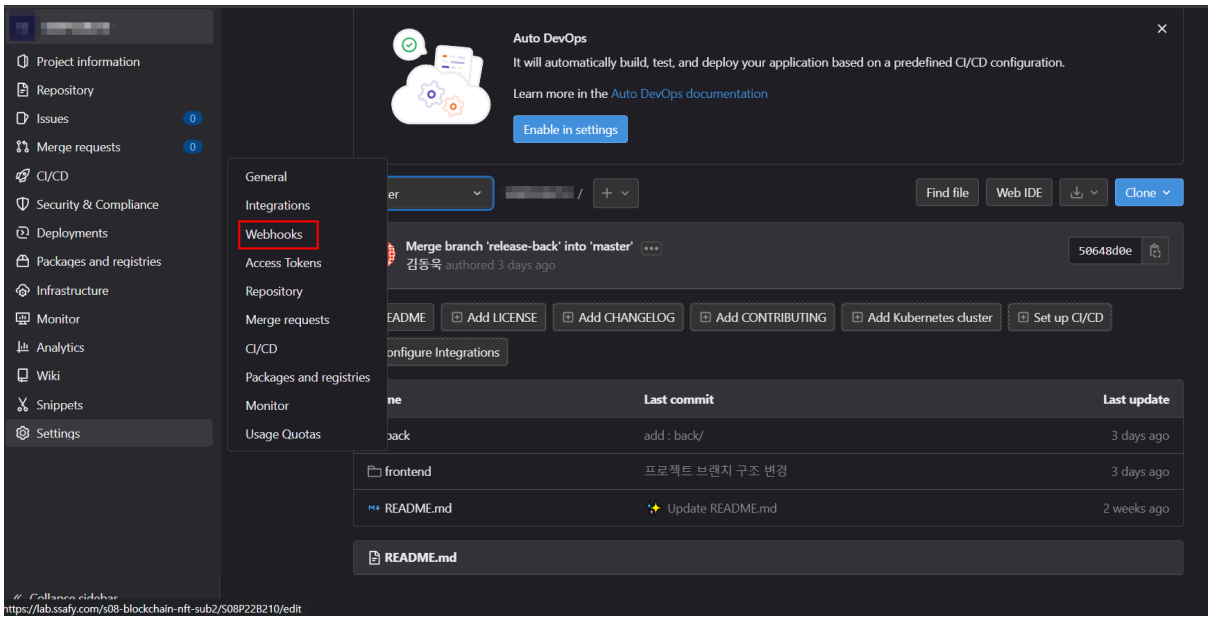
- 하단에 **고급** 체크 후 **Secret Token Generate**

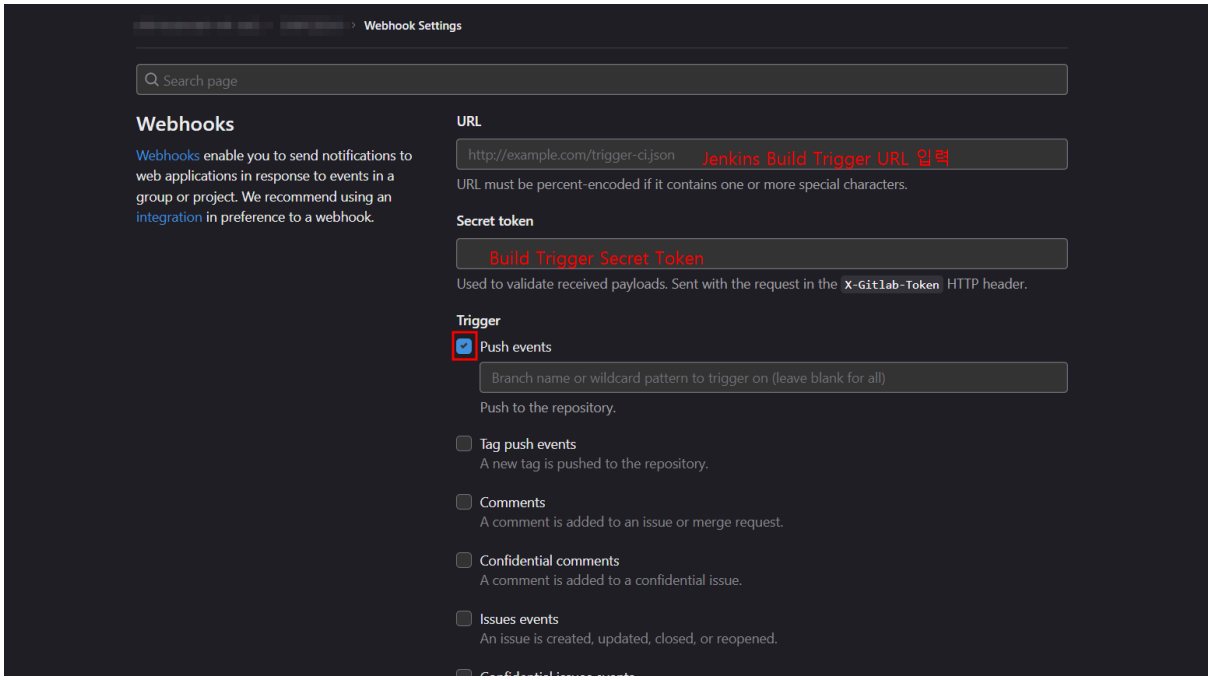
- Pipeline



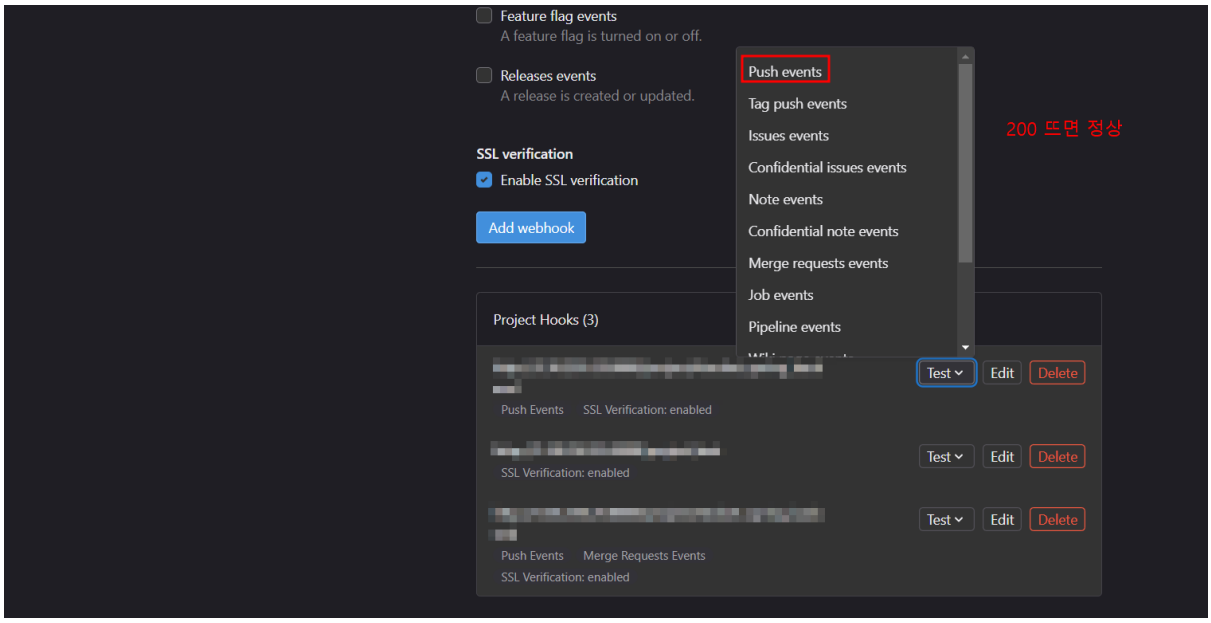
Gitlab Webhook 설정

- Gitlab Repository → Webhooks





- 생성한 webhook 하단에서 test → 200 OK 뜨면 사용

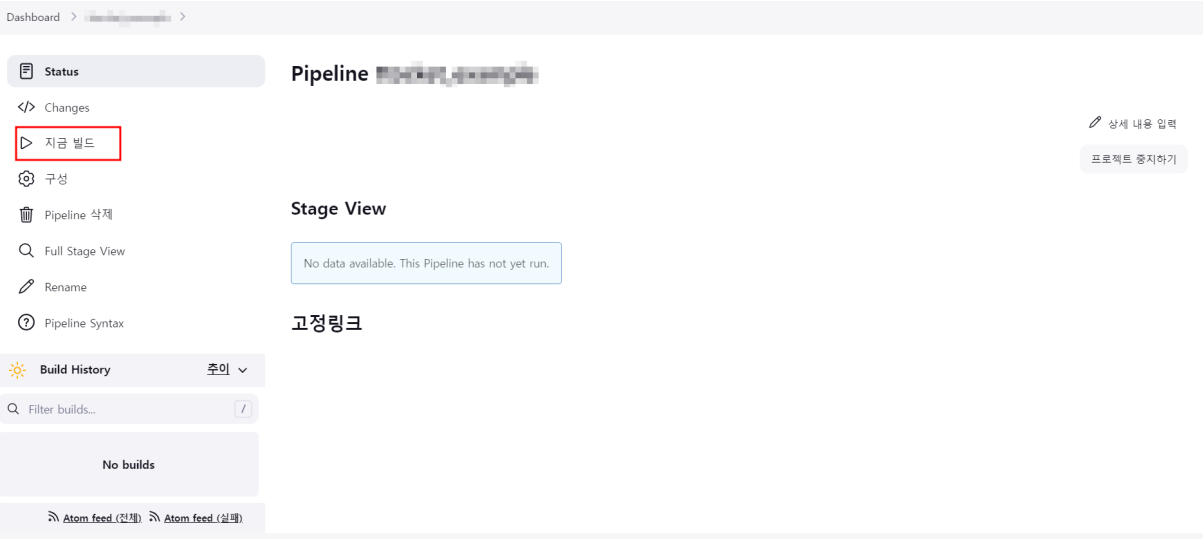


ttocket_react_automation

```
pipeline {
    agent any

    stages {
        stage('CLONE')
        {
        steps{
            git branch : 'release-front', credentialsId : 'lancelot1672' ,url : 'https://lab.ssafy.com/s08-blockchain-nft-sub2/S08P22B
        }
```

```
        }

         stage('DOCKER BUILD')
        {
            steps
            {
            sh '''
            echo 'docker build dir ' ${PWD}
            cd frontend

            docker stop react-container || true
            docker rm react-container || true


            docker rmi ttocket/react-front-end || true
             echo 'Remove if react-container exist'

            docker build -t ttocket/react-front-end .
            '''
            }

        }

        stage('DEPLOY')
        {
            steps
            {
            sh '''

            docker run --name react-container -d -p 3000:3000 ttocket/react-front-end
            echo 'Success'
            '''
            }

        }
    }
}
```

- Build now 로 테스트



Spring Boot

- 참고
  - https://velog.io/@mooh2jj/springboot-jar파일-AWS-EC2에-Docker로-배포하기

수동 배포

로컬에서

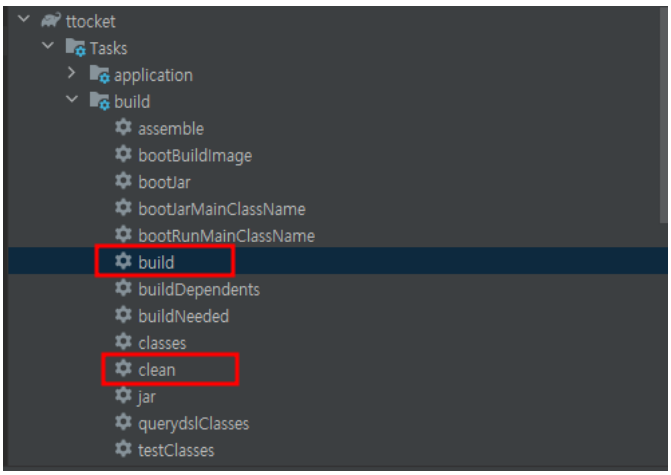
- Spring Boot 어플리케이션에서 Dockerfile 파일 생성

```
# Dockerfile

FROM openjdk:11-jdk

ARG JAR_FILE=build/libs/*.jar

# Docker 컨테이너에서 8080 포트 열기
EXPOSE 8080

COPY ${JAR_FILE} app.jar
ENTRYPOINT ["java","-jar","/app.jar"]
```

  - `build.gradle` 에 다음과 같은 설정 추가

```
// build.gradle
// Spring boot 2.5 버전 이후에 jar 파일 2개 생성 방지 옵션
jar
{
enabled = false
}
```

  - `build clean` 후 다시 `build`



- docker desktop 에서 로그인 혹은 CLI에서 로그인

```
docker login -username={도커hub ID}
```

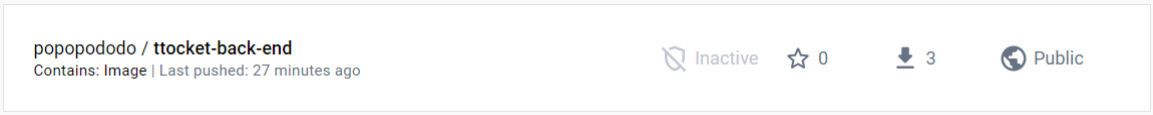
```
# 생성된 Dockerfile을 기반으로 docker build
docker build -t {dockerhub username}/{image name}:{tag} .

# image를 username으로 생성하지 못 해서 생성된 이미지 이름을 변경해야할 때

docker tag {source name} {target name}

# 생성된 image 컨테이너화

docker run -d -p 5000:8080 {image name}
```

- 실행한 컨테이너가 잘 실행되는지 로그 확인 & Postman으로 test

```
docker logs -f {container name}
```

- 잘 실행되면 docker hub 에 image push

```
docker push {image name}
```



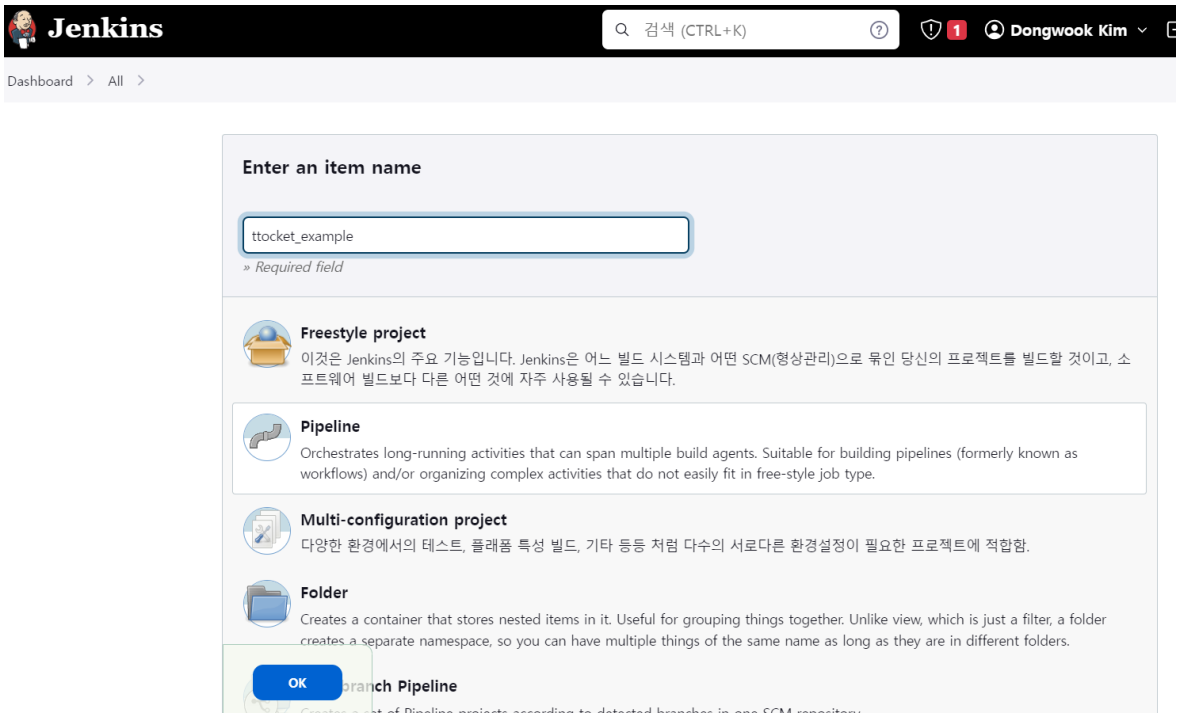
EC2에서

```
# Docker hub 에서 이미지 pull 후 실행
docker run -d -p 8080:8080 {image name}

# 실행 후 로그 확인

docker logs -f {image name}
```

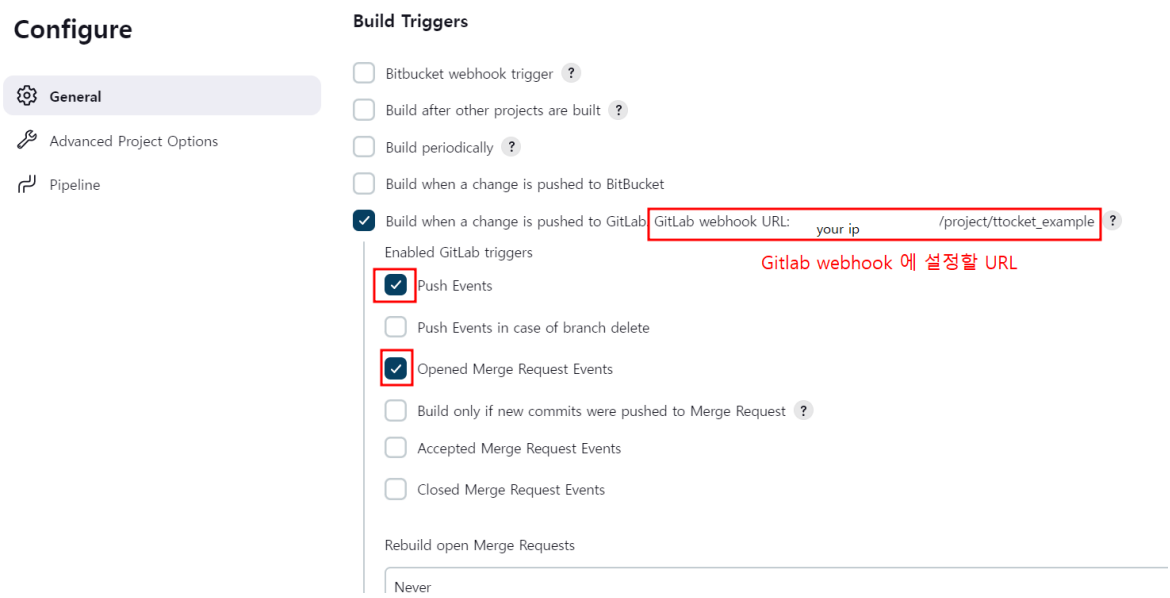
```
performance_start_time, performance_title, user_id)
    values
        (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
Hibernate:
    insert
    into
        performance
        (performance_desc, performance_end_time, performance_etc, performance_location, performance_max_seats, performance_poster, performance_price,
performance_start_time, performance_title, user_id)
    values
        (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
2023-03-21 01:54:49.193 TRACE 1 --- [           main] o.h.type.descriptor.sql.BasicBinder      : binding parameter [1] as [VARCHAR] - [다시봤을 땐 이
기고 싶었는데]
2023-03-21 01:54:49.194 TRACE 1 --- [           main] o.h.type.descriptor.sql.BasicBinder      : binding parameter [2] as [TIMESTAMP] - [2023-03-21T0
1:54:49.183447]
2023-03-21 01:54:49.194 TRACE 1 --- [           main] o.h.type.descriptor.sql.BasicBinder      : binding parameter [3] as [VARCHAR] - [주도권도 다 뺏
기고 허둥거렸어]
2023-03-21 01:54:49.194 TRACE 1 --- [           main] o.h.type.descriptor.sql.BasicBinder      : binding parameter [4] as [VARCHAR] - [그런 순간도 갖
고 싶었어]
2023-03-21 01:54:49.195 TRACE 1 --- [           main] o.h.type.descriptor.sql.BasicBinder      : binding parameter [5] as [INTEGER] - [10]
2023-03-21 01:54:49.195 TRACE 1 --- [           main] o.h.type.descriptor.sql.BasicBinder      : binding parameter [6] as [VARCHAR] - [한동안 안 보였
을 땐 기다려졌고]
2023-03-21 01:54:49.195 TRACE 1 --- [           main] o.h.type.descriptor.sql.BasicBinder      : binding parameter [7] as [DOUBLE] - [34000.0]
2023-03-21 01:54:49.195 TRACE 1 --- [           main] o.h.type.descriptor.sql.BasicBinder      : binding parameter [8] as [TIMESTAMP] - [2023-03-21T0
1:54:49.183436]
2023-03-21 01:54:49.196 TRACE 1 --- [           main] o.h.type.descriptor.sql.BasicBinder      : binding parameter [9] as [VARCHAR] - [처음엔 호기심
이었고]
2023-03-21 01:54:49.196 TRACE 1 --- [           main] o.h.type.descriptor.sql.BasicBinder      : binding parameter [10] as [VARCHAR] - [바둑을 두면서
]
ubuntu@ip-172-26-14-105:~$
```

Jenkins 를 활용한 CI / CD 자동화 (완료!)

- Dashboard → 새로운 item → Pipeline 체크

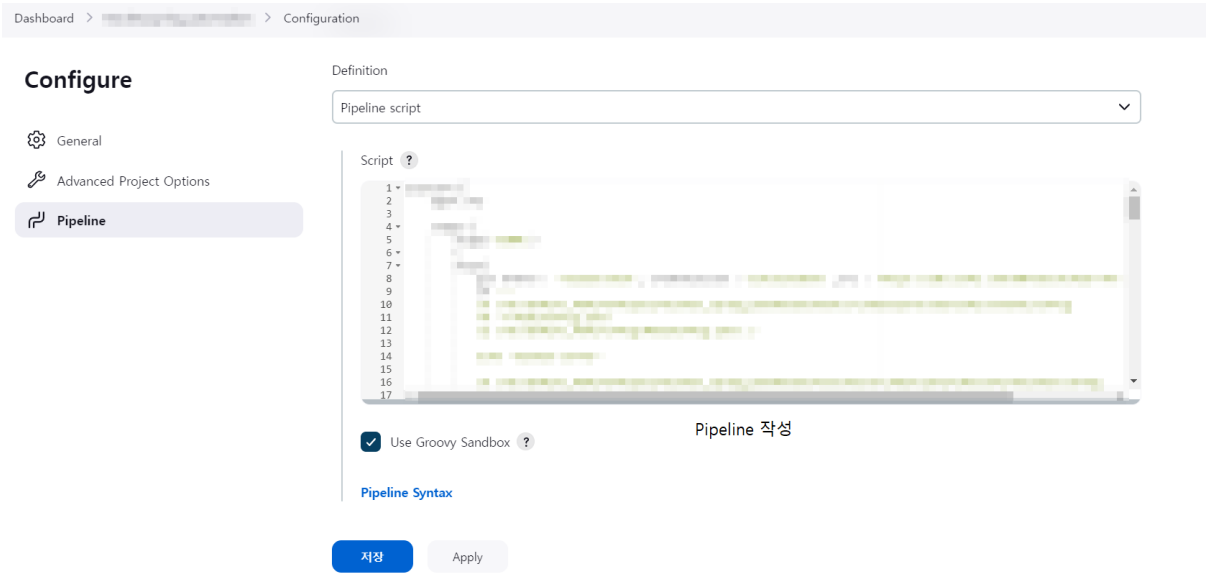


- Build Triggers - Webhook 설정을 해주기 위함



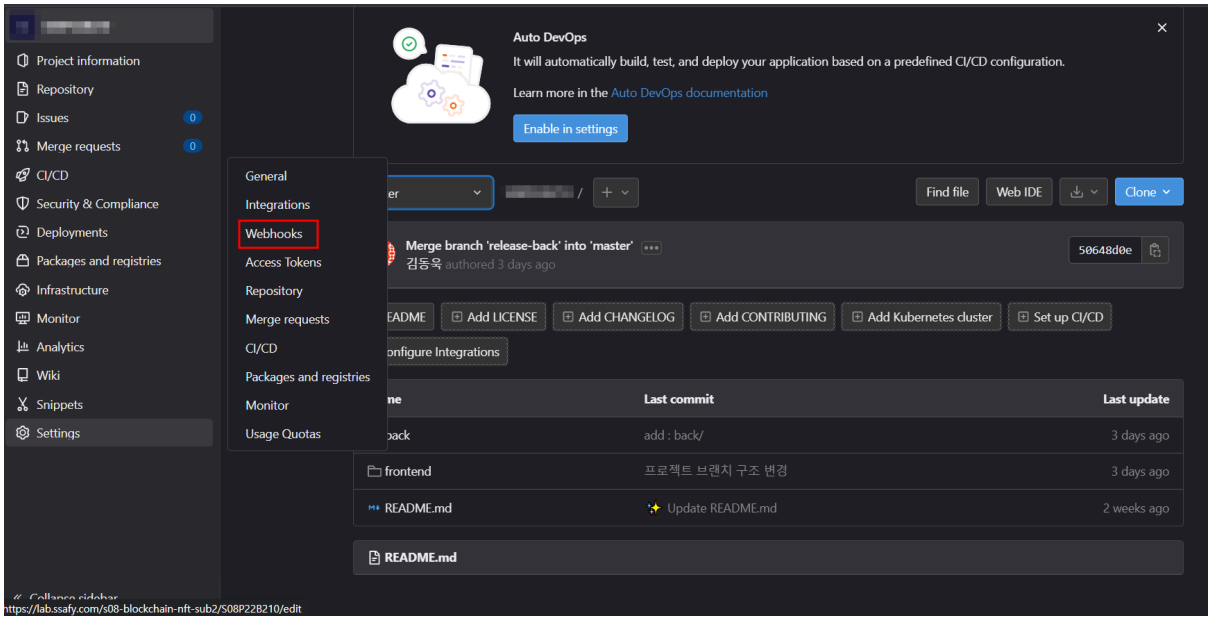
- 하단에 **고급** 체크 후 **Secret Token Generate**

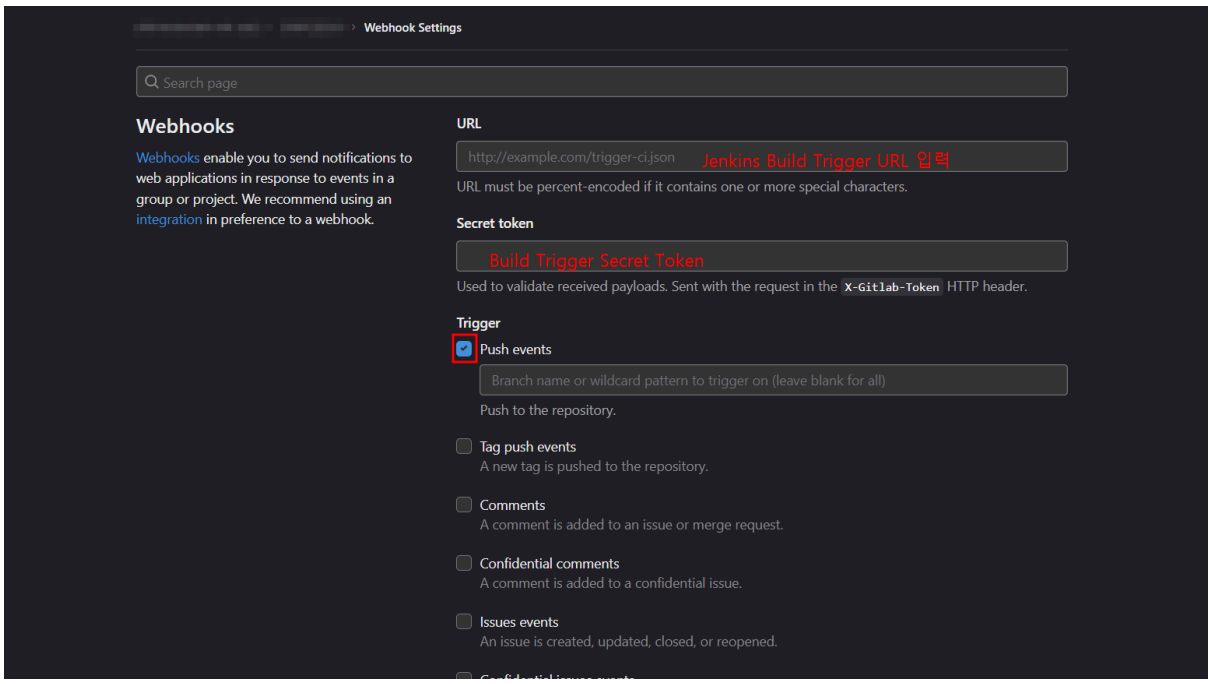
- Pipeline



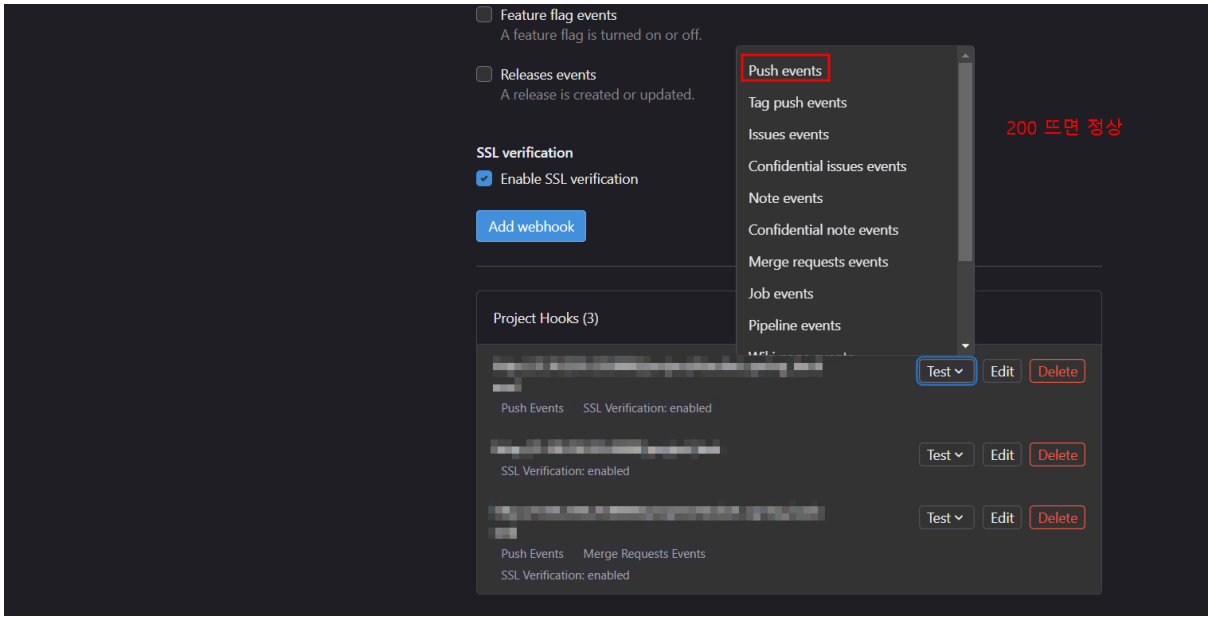
Gitlab Webhook 설정

- Gitlab Repository → Webhooks





- 생성한 webhook 하단에서 test → 200 OK 뜨면 사용

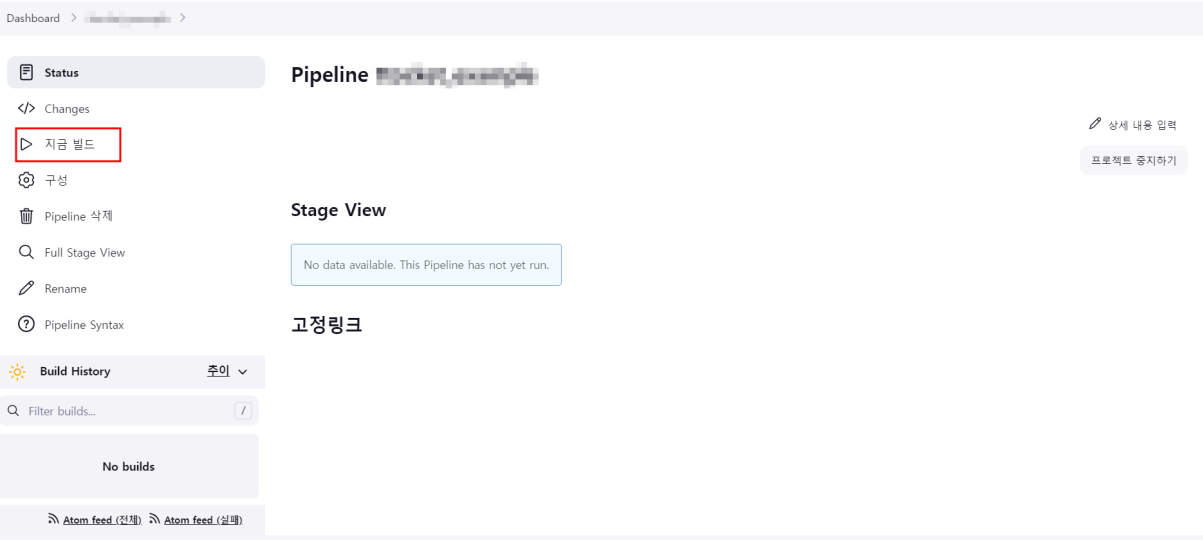


ttocket_spring_automation

```
pipeline {
    agent any

    stages {
        stage('CLONE')
        {
        steps{
            git branch : 'release-back', credentialsId : 'lancelot1672' ,url : 'https://lab.ssafy.com/s08-blockchain-nft-sub2/S08P22B2
            sh '''
            cd /var/jenkins_home/workspace/ttocket_spring_automation/back/src/main/java/com/ssafy/ttocket/config
            rm -r RedisConfig.java
            cp /var/jenkins_home/config/RedisConfig.java ./

            echo 'ttocket-server'

            cd /var/jenkins_home/workspace/ttocket_spring_automation/tttocket/src/main/java/com/ssafy/tttocket/config
            rm -r RedisConfig.java
            cp /var/jenkins_home/config/RedisConfig1.java ./
            mv RedisConfig1.java RedisConfig.java
            '''
        }

        }

        stage('SPRING BUILD')
        {
            steps
            {
            sh '''
            echo 'Build dir ' ${PWD}
            cd /var/jenkins_home/workspace/ttocket_spring_automation/back
            chmod +x gradlew
            ./gradlew clean build

            '''
            }

        }
        stage('TTTOCKET BUILD')
        {
            steps
            {
            sh '''
            echo 'Build dir ' ${PWD}
            cd /var/jenkins_home/workspace/ttocket_spring_automation/tttocket
            chmod +x gradlew
```

```
            ./gradlew clean build

            '''
            }

        }


         stage('SPRING DOCKER BUILD')
        {
            steps
            {
            sh '''
            echo 'docker build dir ' ${PWD}
            cd /var/jenkins_home/workspace/ttocket_spring_automation/back

            docker stop spring-container || true
            docker rm spring-container || true


            docker rmi ttocket/spring-back-end || true
             echo 'Remove if spring-container exist'

            docker build -t ttocket/spring-back-end .
            '''
            }

        }


         stage('TTTOCKET DOCKER BUILD')
        {
            steps
            {
            sh '''
            echo 'docker build dir ' ${PWD}
            cd /var/jenkins_home/workspace/ttocket_spring_automation/tttocket

            docker stop ttocket-container || true
            docker rm ttocket-container || true


            docker rmi ttocket/spring-ttocket-back-end || true
             echo 'Remove if ttocket-container exist'

            docker build -t ttocket/spring-ttocket-back-end .
            '''
            }

        }

        stage('SPRING DEPLOY')
        {
            steps
            {
            sh '''

            docker run --name spring-container --network redis-network -d -p 9000:8080 ttocket/spring-back-end
            echo 'Success'
            '''
            }

        }

        stage('TTTOCKET DEPLOY')
        {
            steps
            {
            sh '''

            docker run --name ttocket-container --network redis-network -d -p 9001:8081 ttocket/spring-ttocket-back-end
            echo 'Success'
            '''
            }

        }

    }
}
```

- Build now 로 테스트



회고 및 향후 진행

- Jenkins GUI 에서 pipeline script 를 직접 수정하는게 테스트하기 편한 것 같다.. → 프로젝트 최종 산출물에 SCM 사용
- Pipeline 의 자동화 어디까지 진행할 수 있을까? → Script 내의 변수를 설정해서 docker image 이름이나 Tag 관리도 한다면 편할텐데

BlockChain

Ganache Network

> Ganache 란 테스트 목적으로 로컬에 설치해서 사용할 수 있는 일종의 간이 블록체인 서버이다. 블록체인 네트워크와 연결할 필요가 없으므로, Contract들을 손쉽게 배포 및 테스트 해볼 수 있다.

- Ganache Network EC2 내에 배포

```
docker run --name ganache-container -d -p 8545:8545 trufflesuite/ganache-cli -l 8000000 -g 0 --networkId 1 --chainId 1337 --hostname 0
```

- Truffle , ERC721 install

```
npm install truffle
npm install @openzeppelin/contracts
```

- 스마트 컨트랙트 컴파일
  - 컴파일 하면 ABI 생성

```
truffle compile
```

- truffle-config.js 파일

```
module.exports = {
  networks: {
    development: {
     host: ip,     // Localhost (default: none)
     port: 8545,            // Standard Ethereum port (default: none)
     network_id: "*",       // Any network (default: none)
    },
  },
  mocha: {
    // timeout: 100000
  },
  compilers: {
    solc: {
      version: "0.8.19",
      settings: {
       optimizer: {
         enabled: true,
         runs: 200
       },
       viaIR: true
      }
    }
  },
};
```

- truffle migration 파일

```
const Ticket = artifacts.require("Ticket");
module.exports = async function (deployer) {
  await deployer.deploy(Ticket);
};
```

- 스마트 컨트랙트 배포

- —compile-all : 전부다 컴파일
- —network development : development 네트워크에 배포
- 하면 배포결과 Contract Address 생성

```
truffle migrate --compile-all --network development
```