포팅 매뉴얼

기술 스택 및 버전

Front-End

- Typescript 5.2.2
- React 18.2.0
- Recoil 0.7.7
- Vite 5.1.6
- Node.js 20.10.0
- NPM 10.3.0

Back-End

- Java 17
- Spring Boot 3.2.3
  - dependency-management 1.1.4
- Spring Data JPA 3.2.3
- Spring Security 6.2.2
- Spring Batch 5
- Querydsl 5.0.0

- Jjwt 0.9.1
- OkHttp 4.12.0
- gson 2.10.1
- Spring Webflux 3.2.3
- Spring Data Redis 3.2.3

DB

- MariaDB 11.3.2
- Redis 7.2.4

Infra

- Ubuntu 20.04.6
- Nginx 1.18.0
- Docker 25.0.4
- Jenkins 2.449

환경 변수

Front-End

.env

```
VITE_CLIENT_ID=[구글 client-id]
VITE_BASE_URL=[서버 API 요청 도메인]
VITE_REDIRECT_URL=[소셜 로그인 시 필요한 redirect url]
```

DockerFile

```
# nginx 이미지 사용
FROM nginx:latest

# root에 /app 폴더 생성
RUN mkdir /app

# work dir 고정
WORKDIR /app

# work dir에 dist 폴더 생성
RUN mkdir ./dist

# host pc의 현재 경로의 dist 폴더를 work dir의 build 폴더로 복사
ADD ./dist ./dist
# nginx의 default.conf 삭제
RUN rm /etc/nginx/conf.d/default.conf

# host pc의 nginx.conf를 아래 경로에 복사
COPY ./nginx.conf /etc/nginx/conf.d

# 80 포트 개방
EXPOSE 80

# container 실행 시 자동으로 실행할 command. nginx 시작함
CMD ["nginx", "-g", "daemon off;"]
```

nginx.conf

```
server {
    listen 80;
    location / {
        root    /app/dist;
        index   index.html;
        try_files $uri $uri/ /index.html;
    }
}
```

Back-End

application.yml

```
spring:
  profiles:
    active: deploy

  application:
    name: funco
```

application-deploy.yml

```
spring:
  config:
    activate:
      on-profile: deploy

  mvc:
    servlet:
      path: /api
```

```
datasource:
  driver-class-name: org.mariadb.jdbc.Driver # Database를 mari
  url: jdbc:mariadb://[호스트 도메인]:[포트 번호]/[db 이름] # maria
  username: [db username] # mariadb 접속 시 입력할 username 정보
  password: [db password] # mariadb 접속 시 입력할 password 정보

#  h2:
#    console:
#      enabled: true  # H2 Console을 사용할지 여부 (H2 Console은
#      path: /h2-console  # H2 Console의 Path

jpa:
  generate-ddl: true
  hibernate:
    ddl-auto: update
  # show-sql: true
  defer-datasource-initialization: false
  # properties:
  #   hibernate:
  #     format_sql: true

data:
  redis:
    host: [호스트 도메인]
    port: [포트 번호]
    password: [레디스 비밀번호]

jwt:
  token:
    secret-key: [설정한 secret-key]
    refresh-secret-key: [설정한 refresh-secret-key]
oauth:
  google:
    client-id: [설정한 client-id]
    client-secret-id: [설정한 client-secret-id]
```

```
      redirect-uri: [설정한 redirect url]
      scope: profile,email
```

Dockerfile

```
FROM docker
COPY --from=docker/buildx-bin:latest /buildx /usr/libexec/docker

FROM openjdk:17-jdk

ADD ./build/libs/funco-0.0.1-SNAPSHOT.jar /app.jar
ENTRYPOINT ["java", "-jar", "/app.jar"]
```

Infra

인증서 및 리버스 프록시 설정(Nginx)

```
# /etc/nginx/sites-enabled/default
# Default server configuration
#
server {
    listen 80 default_server;
    listen [::]:80 default_server;

    root /var/www/html;

    index index.html index.htm index.nginx-debian.html;

    server_name _;
```

```
    location / {
        try_files $uri $uri/ =404;
    }

}


server {

    root /var/www/html;

    index index.html index.htm index.nginx-debian.html;
    server_name j10a404.p.ssafy.io; # managed by Certbot

    location / {

        proxy_pass http://127.0.0.1:3000;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_
        proxy_set_header X-Forwarded-Proto $scheme;
    }

    location /api {
        # First attempt to serve request as file, then
        # as directory, then fall back to displaying a 404.
        # try_files $uri $uri/ =404;
        proxy_pass http://127.0.0.1:8081;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_
        proxy_set_header X-Forwarded-Proto $scheme;
    }

    listen [::]:443 ssl ipv6only=on; # managed by Certbot
    listen 443 ssl; # managed by Certbot
```

```
    ssl_certificate /etc/letsencrypt/live/j10a404.p.ssafy.io/ful
    ssl_certificate_key /etc/letsencrypt/live/j10a404.p.ssafy.io
    include /etc/letsencrypt/options-ssl-nginx.conf; # managed b
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem; # managed by

}
server {
    if ($host = j10a404.p.ssafy.io) {
        return 301 https://$host$request_uri;
    } # managed by Certbot


    listen 80 ;
    listen [::]:80 ;
    server_name j10a404.p.ssafy.io;
    return 404; # managed by Certbot

}
```

인프라 구축하기

개요

- Nginx는 Host OS에 설치
- Jenkins, MariaDB, Redis는 도커 컨테이너에서 구동
- Nginx에 인증서를 적용하여 웹 브라우저에서 요청 시 기본 https로 접근
- Nginx에 리버스 프록시를 적용하여 주소 접근 위치에 따라 각각 화면과 서버로 리다이렉트
- Jenkins 파이프라인 설정을 통해 CI/CD 구축

활용 포트

포트 번호	활용
22	EC2 기본 포트
80	http 접근 시 Nginx 기본 포트
443	https 접근 시 Nginx 기본 포트
3000	서비스 화면
3306	MariaDB
6379	Redis
8080	Jenkins
8081	API 서버

우분투 서버 세팅하기

우분투 서버 시간을 한국 표준시로 변경

```
sudo timedatectl set-timezone Asia/Seoul
```

미러 서버를 카카오 서버로 변경

- 기본 서버는 *.ubuntu.com이라는 해외 서버
- 국내망을 이용할 수 있는 카카오 미러서버를 사용
    - 패키지 갱신/다운로드 속도를 개선한다.

```
sudo sed -i 's/ap-northeast-2.ec2.archive.ubuntu.com/mirror.kaka
```

패키지 목록 업데이트

- 패키지를 받는 미러 서버가 변경되었으므로 업데이트 진행

```
sudo apt-get -y update && sudo apt-get -y upgrade
```

swap 영역 할당

- 메모리 점유율이 높은 작업이 진행될 때, 우선순위가 낮은 작업이 중단되는 현상을 방지
- 아래 순서대로 커맨드 실행
  - 용량 확인

```
free -h
```

  - 스왑 영역 할당(예: 4GB)

```
sudo fallocate -l 4G /swapfile
```

  - swapfile 권한 수정

```
sudo chmod 600 /swapfile
```

  - swapfile 생성

```
sudo mkswap /swapfile
```

  - swapfile 활성화

```
sudo swapon /swapfile
```

  - 시스템이 재부팅 되어도 swap 유지할 수 있도록 설정

```
sudo echo '/swapfile none swap sw 0 0' | sudo tee -a /etc/fst
```

- swap 영역 할당 확인

```
free -h
```

Docker

Docker 설치 전 필요한 패키지 설치

```
sudo apt-get -y install apt-transport-https ca-certificates curl
```

Docker에 대한 GPC Key 인증 진행

- OK가 뜨면 정상적으로 등록되었다는 뜻

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo a
```

Docker 레포지토리 등록

- 해당 레포지토리에 이미지를 등록
- 이미지를 내려받아서 만든 도커 컨테이너가 작동할 환경에 맞추어 ARM64 혹은 AMD64 계열로 레포지토리를 등록
- Ubuntu는 AMD64 계열의 운영체제이기 때문에 AMD64 계열로 등록
- AMD64 계열

```
sudo add-apt-repository "deb [arch=amd64] https://download.docke
```

- ARM64 계열

```
sudo add-apt-repository "deb [arch=arm64] https://download.docke
```

Docker 패키지 설치

- 설치 전 패키지 리스트 갱신

```
sudo apt-get -y update
```

- apt-get을 이용하여 Docker 설치
  - docker-ce : Docker Community Edition
  - docker-ce-cli : Docker Community Edition의 CLI 환경에서 추가로 설치해야 하는 패키지
  - containerd.io : Docker 컨테이너 런타임

Docker 일반 유저에게 권한 부여

- Docker는 기본적으로 항상 root로 실행되어 sudo로 명령어를 입력
- 사용자를 docker 그룹에 추가하여 sudo를 생략해도 명령어를 사용할 수 있도록 허용

```
sudo usermod -aG docker ubuntu
```

- 이후 사용자 세션 로그아웃 및 재로그인

```
sudo service docker restart
```

```
exit
```

Docker 컨테이너 활용하기

- 실행되고 있는 컨테이너 목록 조회

```
docker ps -a
```

- 컨테이너 로그 조회

```
docker logs 컨테이너명
```

NginX

NginX 설치 및 삭제

- NginX 설치

```
sudo apt-get -y install nginx
```

- NginX 삭제
  - nginx, nginx-full, nginx-common 모두 삭제

```
sudo apt-get -y remove --purge nginx nginx-full nginx-common
```

SSL 설정 (CertBot)

- CertBot 다운로드

```
sudo snap install --classic certbot
```

- 인증서 발급

```
sudo certbot --nginx -d [도메인 Host 주소]
```

- 인증서가 발급되면 fullchain.pem과 priavkey.pem 생성
  - 다음 경로에 생성

```
/etc/letsencrypt/live/[도메인 Host 주소]/fullchain.pem
/etc/letsencrypt/live/[도메인 Host 주소]/privkey.pem
```

  - 인증서 발급 시 경로를 알려줌

NginX 리버스 프록시 설정

- 다음 커맨드 입력해서 VIM에 접속

```
sudo vim /etc/nginx/sites-enabled/default
```

- **인증서 및 리버스 프록시 설정(Nginx)** 내용 참고하여 입력

NginX 서비스 재시작

- 설정 적용하기 위해 재시작 필요

```
sudo service nginx restart
```

MariaDB

MariaDB 이미지 받기

```
sudo docker pull mariadb:11.3.2
```

MariaDB 컨테이너 실행

```
docker run -d -p 3306:3306 -e MYSQL_ROOT_PASSWORD=비밀번호 -v /va
```

MariaDB 종료

```
docker stop mariadb
docker rm mariadb
```

MariaDB OS 재시작 후 자동실행 설정

- docker-mariadb.service 등록
  - VIM 실행

```
sudo vim /etc/systemd/system/docker-mariadb.service
```

  - 내용 입력

```
[Unit]
Description=docker-mariadb
Wants=docker.service
After=docker.service

[Service]
RemainAfterExit=yes
ExecStart=/usr/bin/docker start mariadb
ExecStop=/usr/bin/docker stop mariadb

[Install]
WantedBy=multi-user.target
```

- docker 서비스 활성화 및 시작

```
sudo systemctl enable docker
```

```
sudo systemctl start docker
```

- docker-mariadb 서비스 활성화 및 시작

```
sudo systemctl enable docker-mariadb.service
```

```
sudo systemctl start docker-mariadb.service
```

MariaDB 접속

- 기본 계정은 root

```
sudo mariadb -u root -p
```

Redis

Redis 이미지 받기

```
sudo docker pull redis:7.2.4
```

Redis 컨테이너 실행

```
docker run -d -p 6379:6379 -v /home/ubuntu/redis/data:/data -v /
```

Redis 접속 및 assword 설정

- 컨테이너 접속

```
docker exec -it redis /bin/bash
```

- redis-cli 실행

```
redis-cli
```

- redis password
  - password 설정 정보 확인

    ```
    config get requirepass
    ```

  - password 설정

    ```
    config set requirepass 패스워드
    ```

  - password 인증

    ```
    AUTH 패스워드
    ```

Jenkins

Jenkins 설치

Jenkins 이미지 받기

```
docker pull jenkins/jenkins:jdk17
```

Jenkins 컨테이너 실행

```
docker run -d --env JENKINS_OPTS=--httpPort=8080 -v /etc/localt
```

Jenkins 컨테이너 종료

```
docker stop jenkins
docker rm jenkins
```

실행 확인

```
netstat -nltp
```

Jenkins OS 재시작 후 자동실행 설정

docker-jenkins.service 등록

- VIM 실행

```
sudo vim /etc/systemd/system/docker-jenkins.service
```

- 내용 입력

```
[Unit]
Description=docker-jenkins
Wants=docker.service
After=docker.service
```

```
[Service]
RemainAfterExit=yes
ExecStart=/usr/bin/docker start jenkins
ExecStop=/usr/bin/docker stop jenkins

[Install]
WantedBy=multi-user.target
```

docker 서비스 활성화 및 시작

```
sudo systemctl enable docker
```

```
sudo systemctl start docker
```

docker-jenkins 서비스 활성화 및 시작

```
sudo systemctl enable docker-jenkins.service
```

```
sudo systemctl start docker-jenkins.service
```

Jenkins 접속 및 초기 설정

8080포트로 매핑하였으므로, 아래 주소로 접속

```
http://[서버 도메인]:8080
```

Jenkins 잠금을 풀기 위해 비밀번호 확인 후 입력

```
docker exec -it jenkins /bin/bash
```

```
cd /var/jenkins_home/secrets
```

```
# 경로 접속 후 초기 비밀번호 확인
cat initialAdminPassword
```

기본 플러그인 설치

- Install suggested plugins
- 네트워크 환경 문제로 잘 안 될 수 있으나 여러 번 시도하면 잘 됨

관리자 계정 설정

- Instance Configuration
  - Jenkins URL 확인 후 Save and Finish
- Start using Jenkins

빌드 및 배포 : CI/CD 파이프라인 구축

개요

- 깃랩 특정 브랜치에 코드 변경이 감지되면, 파이프라인이 작동하여 빌드와 배포를 수행
- 작업 순서대로 작성

플러그인 설치

```
# ssh 커맨드 입력에 사용
SSH Agent

# docker 이미지 생성에 사용
Docker
Docker Commons
Docker Pipeline
Docker API

# 웹훅을 통해 브랜치 merge request 이벤트 발생시 Jenkins 자동 빌드에 사용
Generic Webhook Trigger

# 타사 레포지토리 이용시 사용 (GitLab, Github 등)
GitLab
GitLab API
GitLab Authentication
GitHub Authentication

# Node.js 빌드시 사용
NodeJS
```

Jenkins- GitLab연동

Credential 등록

- Jenkins 관리 - Manage Credentials 클릭
- Stores scoped to Jenkins - Domains - (global) - Add credentials 클릭
- GitLab 계정 Credential 등록

    - Username : Gitlab 계정 아이디 입력
    - Password : Gitlab 계정 비밀번호 입력(API 토큰 발행한다면 토큰 입력)
    - ID : Credential에 대한 별칭
- GitLab 프로젝트(레포지터리) API Token 등록
    - Kind : Gitlab API token 선택
    - API tokens : Gitlab 계정 토큰 입력
    - ID : Credential에 대한 별칭

GitLab 커넥션 추가

- Jenkins 관리 - System Configuration - System 클릭
- Gitlab의 **Enable authentication for '/project' end-point** 체크
    - Connection name : Gitlab 커넥션 이름 지정
    - Gitlab host URL : Gitlab 시스템의 Host 주소 입력
    - Credentials : 조금 전 등록한 **Jenkins Credential (API Token)**을 선택
    - 이후, **Test Connection**을 눌러 Success가 뜨면 **저장** 클릭
        - 아니라면 입력한 정보를 다시 확인

파이프라인 설정 시 Jenkins Webhook Integeration 설정

- Jenkins 파이프라인 설정
    - Pipeline 아이템에 다음과 같은 설정 추가
        - General - Build Triggers
            - Build when a change is pushed to Gitlab 체크
            - Push Events 체크
            - Opened Merge Request Events 체크
            - Approved Merge Request (EE-only) 체크

        - Comments 체크
    - 고급 - Generate 클릭
        - 발행된 Secret token 복사해두고 **저장** 클릭
- GitLab Repository
    - Settings - Webhooks 클릭
        - URL : Jenkins의 Item URL 입력

```
http://[Jenkins Host]:[Jenkins Port]/project/[파이프라인 아이
```

        - Secret token : Jenkins의 Gitlab trigger 고급 설정 중 Secret token Generate 버튼을 이용해 만든 토큰 입력
        - Trigger : Push events 체크, merge request가 되면 Jenkins 이벤트가 발동하게 할 브랜치 입력
        - SSL verification의 **Enable SSL verification** 체크
            - 이후, **Add webhook** 클릭

Jenkins-DockerHub 연동

Jenkins

- Jenkins 관리 - Security - Manage Credentials 클릭
- Stores scoped to Jenkins - Domains - (global) - Add Credentials
- Credential 정보
    - Kind : Username with password
    - Username : DockerHub에서 사용하는 계정 아이디 입력
    - Password : DockerHub에서 사용하는 Access Token 입력

- ID : Jenkins 내부에서 사용하는 Credential 별칭 입력

DockerHub

- 레포지토리 생성
- Access Token 발급

Jenkins-Ubuntu 연동

Jenkins

- Jenkins 관리 - Security - Manage Credentials 클릭
- Stores scoped to Jenkins - Domains - (global) - Add Credentials
- Credential 정보
  - Kind : SSH Username with private key
  - ID : Jenkins에서 Credential에 지정할 별칭
  - Username : SSH 원격 서버 호스트에서 사용하는 계정명
  - Private Key
    - Enter directly 체크 후 Add 클릭
    - AWS *.pem 키의 내용을 메모장으로 읽어 복사 후 Key에 붙여넣기

Jenkins Pipeline 추가

아이템 추가

- 새로운 Item 추가
- 아이템 이름 지정
- Pipeline → OK

GitLab 연동 설정

- Configure - General - GitLab Connection
- Build when a change is pushed to GitLab 체크

Jenkins Credential

환경 변수 파일 등록

- Jenkins 관리 - Manage Credentials 클릭
- Stores scoped to Jenkins - Domains - (global) - Add credentials 클릭
  - Kind : Secret file
  - File 클릭 후 환경 설정 파일을 업로드
    - .env
    - application-deploy.yml
  - ID : 파이프라인에서 사용할 별칭
  - Description : 파일 설명

프론트엔드 추가 설정

Node.js 추가

- Jenkins 관리 - System Configuration - Tools 클릭

- Tools - NodeJS installations - Add NodeJS 클릭
- Jenkins 컨테이너의 Node.js 빌드환경 설정
  - Name : Node.js 환경에 대한 이름
  - Version : 빌드하려는 Node.js 버전 선택

환경 변수 설정하기

- Jenkins 관리 - System Configuration - System 클릭
- Global Properties
  - Environment variables 체크
  - **CI, false** 환경변수 추가
  - 빌드 시 경고를 에러로 인식하는 문제를 방지하기 위함

백엔드 파이프라인 스크립트

변수명은 등록한 내용에 맞춰서 작성

```
pipeline {
    agent any

    environment {

        imageName = "ert4263/be-release"
        registryCredential = 'ert4263-docker'
        dockerImage = ''

        releaseServerAccount = 'ubuntu'
        releaseServerUri = 'j10a404.p.ssafy.io'
        releasePort = '8081'
```

```
}

stages {
    stage('Git Clone') {
        steps {
            git branch: 'BE-Release', credentialsId:'gitlab
        }
    }
    stage('Add yml'){
        steps{
            dir('./Backend/funco'){
                withCredentials([file(credentialsId: 'applic
                    sh 'cp ${application} src/main/resources
                }
            }
        }
    }
    stage('BE-Build') {
        steps {
            dir("./Backend/funco") {
                sh "chmod +x ./gradlew"
                sh "./gradlew clean bootJar"
            }
        }
    }
    stage('Image Build & DockerHub Push') {
        steps {
            dir('./Backend/funco') {
                script {
                    docker.withRegistry('', registryCredenti
                        sh "docker buildx create --use --nam
                        sh "docker buildx build --platform l
                        sh "docker buildx build --platform l
                    }
```

```
                }
            }
        }
    }

    stage('Before Service Stop') {
        steps {
            sshagent(credentials: ['ubuntu-ssh']) {
                sh '''
                if test "`ssh -o StrictHostKeyChecking=no $r
                ssh -o StrictHostKeyChecking=no $releaseServ
                ssh -o StrictHostKeyChecking=no $releaseServ
                ssh -o StrictHostKeyChecking=no $releaseServ
                fi
                '''
            }
        }
    }
    stage('DockerHub Pull') {
        steps {
            sshagent(credentials: ['ubuntu-ssh']) {
                sh "ssh -o StrictHostKeyChecking=no $release
            }
        }
    }
    stage('Service Start') {
        steps {
            sshagent(credentials: ['ubuntu-ssh']) {
                script {
                    docker.withRegistry('', registryCredenti
                        // 현재 -e spring profile은 지워둔 상태
                        sh '''
                            ssh -o StrictHostKeyChecking=no
                            '''
                    }
                }
```

```
                }
            }
        }
    }
}
```

프론트엔드 파이프라인 스크립트

변수명은 등록한 내용에 맞춰서 작성

```
pipeline {
    agent any
    tools {nodejs "nodejs"}
    environment {

        imageName = "ert4263/fe-release"
        registryCredential = 'ert4263-docker'
        dockerImage = ''

        releaseServerAccount = 'ubuntu'
        releaseServerUri = 'j10a404.p.ssafy.io'
        releasePort = '3000'


    }

    stages {
        stage('Git Clone') {
            steps {
                git branch: 'FE-Release', credentialsId:'gitlab
```

```
            }
        }
        stage('Add Env') {
        steps {
            dir('./Frontend') {
                withCredentials([file(credentialsId: 'applicatio
                   sh 'cp ${env} .env'
                    }
                }
            }
        }
        stage('Node Build') {
            steps {
                dir("./Frontend") {
                    sh "npm install"
                    sh "npm run build"
                }
            }
        }
        stage('Image Build & DockerHub Push') {
            steps {
                dir('./Frontend'){
                    script {
                        docker.withRegistry('', registryCredenti
                            sh "docker buildx create --use --nam
                            sh "docker buildx build --platform l
                            sh "docker buildx build --platform l
                        }
                    }
                }
            }
        }


        stage('Before Service Stop') {
            steps {
```

```
                sshagent(credentials: ['ubuntu-ssh']) {
                    sh '''
                    if test "`ssh -o StrictHostKeyChecking=no $r
                    ssh -o StrictHostKeyChecking=no $releaseServ
                    ssh -o StrictHostKeyChecking=no $releaseServ
                    ssh -o StrictHostKeyChecking=no $releaseServ
                    fi
                    '''
                }
            }
        }
        stage('DockerHub Pull') {
            steps {
                sshagent(credentials: ['ubuntu-ssh']) {
                    sh "ssh -o StrictHostKeyChecking=no $release
                }
            }
        }
        stage('Service Start') {
            steps {
                sshagent(credentials: ['ubuntu-ssh']) {
                    script {
                        docker.withRegistry('', registryCredenti

                            sh '''
                                ssh -o StrictHostKeyChecking=no
                                '''

                        }
                    }
                }
            }
        }
    }
}
```

외부 API 설정

구글 소셜 로그인

Google Cloud 설정

앱 정보 등록하기

- 프로젝트 생성 후 API & Services로 이동
- OAuth 동의 화면 → 앱 만들기
- 앱 정보 등록
    - 앱 이름
    - 사용자 정의 이메일
    - 앱 도메인
    - 승인된 도메인
    - 개발자 연락처 정보
- 활용한 API 범위
    - 이메일 주소 확인
        - auth/userinfo/email
    - 개인정보 확인

        - auth/userinfo/profile
    - 개인정보 연결
        - openid
- 앱 생성 완료

앱 생성 이후 사용자 인증 정보

- 클라이언트 ID, 클라이언트 보안 비밀번호 확인 가능
- 필요한 정보들을 기재
    - 승인된 자바스크립트 원본
        - 도메인
    - 승인된 리디렉션 URI
        - 구글에서 인가 코드를 받을 URI

업비트 API

UpbitAPI-Server

다음 요청을 수행하며, 따로 Credential은 불필요

코인 리스트

https://api.upbit.com/v1/market/all

websocket

wss://api.upbit.com/websocket/v1

현재 가격

https://api.upbit.com/v1/ticker?markets=KRW-BTC,KRW-ETH

캔들 1분 봉 200개

https://api.upbit.com/v1/candles/minutes/1?market=KRW-BTC&count=200