포팅메뉴얼

Ddukdoc 프로젝트 포팅 메뉴얼

1. **기본 배포 방식**: EC2 인스턴스에서 Git Clone을 통해 소스 코드를 가져와 수동으로 배포.
2. **CI/CD 포함 배포 방식**: 개발 및 운영에 사용하였던 Jenkins를 활용한 자동화된 배포 파이프라인.

Version 및 Setting

Backend + Infra

- Springboot : 3.4.3
- Gradle : 8.12.1
- Java : 17 (LTS)
- Docker : 28.0.1
- Docker Compose : 1.29.2
- Nginx : 1.27.4
- Nodejs : 18.20.7
- Redis(개발용) : 17.4.2
- Jenkins : 2.50.1
- ElasticSearch : 8.17.4

Frontend

- React : 19
- Vite : 6 기반의 초고속 번들링 및 개발 환경.
- TailwindCSS : 3.4 + Prettier 플러그인으로 스타일 자동 정렬.
- Storybook : 8.6 도입으로 컴포넌트 개발 및 테스트 용이.
- MSW : 2.x 활용으로 API Mocking 가능.
- Zustand : 5.x 도입으로 간단하고 직관적인 전역 상태 관리.

KAKAO 디벨로퍼 설정

Redirect URI

```
https://j12b108.p.ssafy.io/oauth/kakao/callback
https://ddukdoc.shop/oauth/kakao/callback
http://localhost:8080/api/oauth/kakao/login
https://j12b108.p.ssafy.io/api/oauth/kakao/login
https://ddukdoc.shop/api/oauth/kakao/login
```

SSAFY OpenAPI 설정

도메인 : `https://j12b108.p.ssafy.io`

Redirect URI

```
http://localhost:8080/api/oauth/ssafy/login
https://j12b108.p.ssafy.io/api/oauth/ssafy/login
https://ddukdoc.shop/api/oauth/ssafy/login
```

1. 기본 배포 방식 (Git Clone 기반)

이 섹션에서는 EC2 인스턴스에 접속하여 소스 코드를 클론하고, 수동으로 환경을 설정하여 애플리케이션을 배포하는 과정을 설명합니다.

1.1. 사전 준비

1.1.1. 도메인 및 DNS 설정

- **도메인**: `ddukdoc.shop` (Gabia에서 구매)
- **DNS 설정**:
  - EC2 인스턴스의 퍼블릭 IP와 도메인을 연결.
  - Gabia DNS 관리 콘솔에서 설정.
  - **A 레코드**:

        - 호스트: `@`
        - 값: EC2 퍼블릭 IP (예: `172.26.5.85`)
    - **CNAME 레코드**:
        - 호스트: `www`
        - 값: `ddukdoc.shop`

DNS 설정 사진 첨부

DNS 설정 레코드 수정

타입	호스트	값/위치	TTL	우선 순위	서비스
A	@	43.203.179.207	600		DNS 설정
A	www	43.203.179.207	600		DNS 설정
A	*	43.203.179.207	86400		DNS 설정

1.1.2. EC2 인스턴스 설정

- **운영 체제**: Ubuntu (버전은 제공되지 않았으므로 확인 필요, 예: 22.04 LTS)
- **보안 그룹 및 방화벽 설정**:
    - 아래 포트들을 열어 외부 접근 허용.
    - 방화벽 관리: `ufw` 사용.
    - **명령어**:

```
sudo ufw allow [포트]
sudo ufw enable
sudo ufw status numbered
```

    - **포트 목록 및 용도**:
        - `22`: SSH 접속
        - `80`: HTTP (Nginx)
        - `443`: HTTPS (Nginx)
        - `8080`: 운영 환경 Spring Boot - Blue
        - `8081`: 운영 환경 Spring Boot - Green
        - `8085`: 개발 환경 Spring Boot - Blue

- `8086`: 개발 환경 Spring Boot - Green
- `6379`: Redis 데이터베이스
- `3306`: MySQL/MariaDB 데이터베이스
- `9090`: Jenkins 웹 인터페이스
- `50000`: Jenkins 에이전트 통신
- `3000`: 스마트 컨트랙트 통신 Express.js 서버
- `5044`, `5045`: Logstash 입력
- `9600`: Logstash API
- `51000`: Logstash TCP/UDP
- `9200`: Elasticsearch HTTP
- `9300`: Elasticsearch 노드 간 통신
- `5601`: Kibana 웹 인터페이스

1.1.3. 필수 소프트웨어 설치

EC2 인스턴스에 접속하여 아래 소프트웨어를 설치합니다.

- **Git**:

```
sudo apt update
sudo apt install -y git
git --version
```

- **Docker**:

```
sudo apt update
sudo apt install -y apt-transport-https ca-certificates curl software-pro
perties-common
curl -fsSL <https://download.docker.com/linux/ubuntu/gpg> | sudo gpg
--dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg
echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/ke
yrings/docker-archive-keyring.gpg] <https://download.docker.com/linu
x/ubuntu> $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/d
ocker.list > /dev/null
sudo apt update
```

```
sudo apt install -y docker-ce
docker --version
```

- **Docker Compose**:

```
sudo curl -L "<https://github.com/docker/compose/releases/latest/download/docker-compose-$>(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
sudo chmod +x /usr/local/bin/docker-compose
docker-compose --version
```

1.2. 소스 코드 클론

프로젝트 소스 코드를 GitHub/GitLab에서 클론합니다.

```
git clone [Repository_URL]
cd [Repository_Name]
```

누락 확인: Repository URL이 제공되지 않았습니다. 실제 URL (예: `https://github.com/username/ddukdoc.git`)을 명시해야 합니다.

1.3. 환경 설정

1.3.1. 디렉토리 구조 생성

필요한 디렉토리를 생성합니다.

```
mkdir -p /home/ubuntu/nginx/conf
mkdir -p /home/ubuntu/nginx/html/dev
mkdir -p /home/ubuntu/nginx/html/prod
mkdir -p /home/ubuntu/nginx/ssl
mkdir -p /home/ubuntu/nginx_landing/landing
mkdir -p /home/ubuntu/certbot/conf
mkdir -p /home/ubuntu/certbot/www
mkdir -p /home/ubuntu/logs
```

1.3.2. Docker 네트워크 생성

컨테이너 간 통신을 위해 네트워크를 생성합니다.

```
docker network create app-network
```

1.3.3. Docker Compose 파일 작성

`/home/ubuntu/` 경로에 아래 Docker Compose 파일들을 작성합니다.

- **docker-compose.yml** (Jenkins, Nginx, Certbot):

```
version: '3'
services:
  jenkins:
    image: jenkins/jenkins:jdk17
    container_name: jenkins
    ports:
      - "9090:8080"
      - "50000:50000"
    volumes:
      - jenkins_home:/var/jenkins_home
      - /var/run/docker.sock:/var/run/docker.sock
      - /home/ubuntu:/home/ubuntu
    restart: unless-stopped
    networks:
      - app-network

  nginx:
    image: nginx:latest
    container_name: nginx
    ports:
      - "80:80"
      - "443:443"
    volumes:
      - ./nginx/conf:/etc/nginx/conf.d
      - ./nginx/html:/usr/share/nginx/html
      - ./nginx/ssl:/etc/nginx/ssl
      - ./certbot/conf:/etc/letsencrypt
      - ./certbot/www:/var/www/certbot
      - /home/ubuntu/nginx_landing/landing:/var/www/landing
    restart: unless-stopped
```

```
    command: "/bin/sh -c 'while :; do sleep 6h & wait $${!}; nginx -s relo
ad; done & nginx -g \\"daemon off;\\"'"
    networks:
      - app-network

  certbot:
    image: certbot/certbot
    container_name: certbot
    volumes:
      - ./certbot/conf:/etc/letsencrypt
      - ./certbot/www:/var/www/certbot
    restart: unless-stopped
    entrypoint: "/bin/sh -c 'trap exit TERM; while :; do certbot renew; sle
ep 12h & wait $${!}; done;'"
    networks:
      - app-network

volumes:
  jenkins_home:

networks:
  app-network:
    external: true
```

- **docker-compose-prod.yml** (운영 환경):

```
version: '3'
services:
  backend-prod-blue:
    image: ddukdoc-backend:production-blue
    container_name: backend-prod-blue
    ports:
      - "8080:8080"
    environment:
      - SERVER_PORT=8080
      - SPRING_PROFILES_ACTIVE=prod
      - TZ=Asia/Seoul
    volumes:
```

```
      - log-volume:/logs
    restart: unless-stopped
    networks:
      - app-network
    healthcheck:
      test: ["CMD", "curl", "-f", "<http://localhost:8080/api/actuator/healt
h>"]
      interval: 30s
      timeout: 10s
      retries: 3
      start_period: 40s

  backend-prod-green:
    image: ddukdoc-backend:production-green
    container_name: backend-prod-green
    ports:
      - "8081:8080"
    environment:
      - SERVER_PORT=8080
      - SPRING_PROFILES_ACTIVE=prod
      - TZ=Asia/Seoul
    volumes:
      - log-volume:/logs
    restart: unless-stopped
    networks:
      - app-network
    healthcheck:
      test: ["CMD", "curl", "-f", "<http://localhost:8080/api/actuator/healt
h>"]
      interval: 30s
      timeout: 10s
      retries: 3
      start_period: 40s

  blockchain-api:
    image: blockchain-api:latest
    container_name: blockchain-api
    ports:
```

```
      - "3000:3000"
    environment:
      - NODE_ENV=production
    restart: unless-stopped
    networks:
      - app-network

volumes:
  log-volume:
    driver: local
    driver_opts:
      type: none
      o: bind
      device: /home/ubuntu/logs

networks:
  app-network:
    external: true
```

- **docker-compose-dev.yml** (개발 환경):

```
version: '3'
services:
  backend-dev-blue:
    image: ddukdoc-backend:development-blue
    container_name: backend-dev-blue
    ports:
      - "8085:8085"
    environment:
      - SERVER_PORT=8085
      - SPRING_PROFILES_ACTIVE=dev
      - TZ=Asia/Seoul
    volumes:
      - log-volume:/logs
    restart: unless-stopped
    networks:
      - app-network
    healthcheck:
```

```
      test: ["CMD", "curl", "-f", "<http://localhost:8085/api/actuator/healt
h>"]
      interval: 30s
      timeout: 10s
      retries: 3
      start_period: 40s

  backend-dev-green:
    image: ddukdoc-backend:development-green
    container_name: backend-dev-green
    ports:
      - "8086:8085"
    environment:
      - SERVER_PORT=8085
      - SPRING_PROFILES_ACTIVE=dev
      - TZ=Asia/Seoul
    volumes:
      - log-volume:/logs
    restart: unless-stopped
    networks:
      - app-network
    healthcheck:
      test: ["CMD", "curl", "-f", "<http://localhost:8085/api/actuator/healt
h>"]
      interval: 30s
      timeout: 10s
      retries: 3
      start_period: 40s

  redis-dev:
    image: redis:latest
    container_name: redis-dev
    command: redis-server --requirepass b108ddukdoc
    ports:
      - "6379:6379"
    volumes:
      - redis-data:/data
    restart: unless-stopped
```

```
    networks:
      - app-network

volumes:
  redis-data:
  log-volume:
    driver: local
    driver_opts:
      type: none
      o: bind
      device: /home/ubuntu/logs

networks:
  app-network:
    external: true
```

1.3.4. Nginx 설정

Nginx 설정 파일을 작성하여 트래픽을 관리합니다.

- **/home/ubuntu/nginx/conf/default.conf** (운영 환경):

```
server {
    listen 80;
    server_name j12b108.p.ssafy.io;

    location /.well-known/acme-challenge/ {
        root /var/www/certbot;
    }

    location / {
        return 301 https://$host$request_uri;
    }
}

upstream backend_prod {
  server backend-prod-green:8080;
}
```

```
server {
    listen 443 ssl;
    server_name j12b108.p.ssafy.io;

    ssl_certificate /etc/letsencrypt/live/j12b108.p.ssafy.io/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/j12b108.p.ssafy.io/privkey.pe
m;

    ssl_protocols TLSv1.2 TLSv1.3;
    ssl_prefer_server_ciphers on;
    ssl_ciphers 'ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES
128-GCM-SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RS
A-AES256-GCM-SHA384';

    client_max_body_size 100M;

    location /blockchain {
      proxy_pass <http://blockchain-api:3000/blockchain>;
      proxy_set_header Host $host;
      proxy_set_header X-Real-IP $remote_addr;
      proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
      proxy_set_header X-Forwarded-Proto $scheme;
    }

    location /landing {
        alias /var/www/landing/;
        index index.html;
        try_files $uri $uri/ /index.html =404;
    }

    location / {
        root /usr/share/nginx/html/prod;
        index index.html index.htm;
        try_files $uri $uri/ /index.html;
    }

    location /api {
        proxy_pass http://backend_prod/api;
```

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

        proxy_connect_timeout 300s;
        proxy_send_timeout 300s;
        proxy_read_timeout 300s;
    }
}
```

- **/home/ubuntu/nginx/conf/dev.conf** (개발 환경):

```
server {
    listen 80;
    server_name ddukdoc.shop;

    location /.well-known/acme-challenge/ {
        root /var/www/certbot;
    }

    location / {
        return 301 https://$host$request_uri;
    }
}

upstream backend_dev {
  server backend-dev-blue:8085;
}

server {
    listen 443 ssl;
    server_name ddukdoc.shop;

    ssl_certificate /etc/letsencrypt/live/ddukdoc.shop/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/ddukdoc.shop/privkey.pem;

    ssl_protocols TLSv1.2 TLSv1.3;
```

```
    ssl_prefer_server_ciphers on;
    ssl_ciphers 'ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES
128-GCM-SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RS
A-AES256-GCM-SHA384';

    client_max_body_size 100M;

    location /api {
        proxy_pass http://backend_dev/api;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;

        proxy_connect_timeout 300s;
        proxy_send_timeout 300s;
        proxy_read_timeout 300s;
    }

    location / {
        root /usr/share/nginx/html/dev;
        index index.html index.htm;
        try_files $uri $uri/ /index.html;
    }

    location /swagger-ui {
        proxy_pass http://backend_dev/swagger-ui;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }

    location /v3/api-docs {
      proxy_pass http://backend_dev/v3/api-docs;
      proxy_set_header Host $host;
      proxy_set_header X-Real-IP $remote_addr;
      proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
```

```
        proxy_set_header X-Forwarded-Proto $scheme;
    }
}
```

1.3.5. HTTPS 설정 (Let's Encrypt)

HTTPS를 위해 Let's Encrypt 인증서를 발급하고, Docker 컨테이너에 마운트합니다.

1. **Certbot 설치 (호스트)**:

```
sudo apt install -y certbot python3-certbot-nginx
```

2. **인증서 발급**:

```
sudo certbot certonly --standalone -d ddukdoc.shop -d j12b108.p.ssafy.io
```

   - 인증서 경로:
     - `/etc/letsencrypt/live/ddukdoc.shop/`
     - `/etc/letsencrypt/live/j12b108.p.ssafy.io/`
   - 파일:
     - `fullchain.pem`
     - `privkey.pem`

3. **인증서 복사**:

```
sudo cp -r /etc/letsencrypt /home/ubuntu/certbot/conf/
sudo chown -R ubuntu:ubuntu /home/ubuntu/certbot
```

4. **Docker Compose에서 마운트**:
   - `docker-compose.yml`의 `nginx` 서비스에서 `/home/ubuntu/certbot/conf:/etc/letsencrypt` 볼륨 마운트.
5. **인증서 갱신**:
   - `docker-compose.yml`의 `certbot` 서비스가 12시간마다 `certbot renew` 실행.

1.3.6. Redis 설정

개발 환경 Redis에 비밀번호 설정:

```
docker exec -it redis-dev redis-cli
CONFIG SET requirepass "b108ddukdoc"
CONFIG REWRITE
```

1.4. 애플리케이션 빌드 및 배포

누락 확인: 빌드 및 배포에 필요한 `Dockerfile`이 제공되지 않았습니다. 아래는 가정된 명령어입니다.

1.4.1. 백엔드 (Spring Boot)

1. **빌드**:

   ```
   cd backend
   ./gradlew clean build -x test
   ```

2. **Docker 이미지 빌드**:

   ```
   docker build -t ddukdoc-backend:development-blue .
   docker build -t ddukdoc-backend:development-green .
   docker build -t ddukdoc-backend:production-blue .
   docker build -t ddukdoc-backend:production-green .
   ```

3. **컨테이너 실행**:

   ```
   docker-compose -f /home/ubuntu/docker-compose-dev.yml up -d
   docker-compose -f /home/ubuntu/docker-compose-prod.yml up -d
   ```

1.4.2. 블록체인 API (Express.js)

1. **빌드**:

   ```
   cd contracts
   npm install
   ```

2. **Docker 이미지 빌드**:

```
docker build -t blockchain-api:latest .
```

3. **컨테이너 실행**:
   - `docker-compose-prod.yml`에서 `blockchain-api` 서비스로 실행.

1.4.3. 프론트엔드 (React)

1. **빌드**:

```
cd frontend
npm install
npm run build
```

2. **배포**:

```
cp -r dist/* /home/ubuntu/nginx/html/dev/
cp -r dist/* /home/ubuntu/nginx/html/prod/
```

3. **Nginx 재시작**:

```
docker exec nginx nginx -s reload
```

1.5. ELK 스택 설정

로그 관리를 위해 ELK 스택을 설정합니다.

1. **소스 클론**:

```
git clone <https://github.com/deviantony/docker-elk.git>
cd docker-elk
```

2. **환경 변수 설정**:
   - `.env` 파일 수정:

```
ELASTIC_PASSWORD=Edduk108
LOGSTASH_INTERNAL_PASSWORD=Ldduk108
KIBANA_SYSTEM_PASSWORD=Kdduk108
```

3. **Logstash 포트 설정**:
   - `docker-compose.yml` 수정:

```
logstash:
  build:
    context: logstash/
    args:
      ELASTIC_VERSION: ${ELASTIC_VERSION}
  volumes:
    - ./logstash/config/logstash.yml:/usr/share/logstash/config/logst
ash.yml:ro,Z
    - ./logstash/pipeline:/usr/share/logstash/pipeline:ro,Z
  ports:
    - 5044:5044
    - 51000:50000/tcp
    - 51000:50000/udp
```

4. **ELK 실행**:

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

1.6. 검증

- **컨테이너 상태**:

```
docker ps
```

- **서비스 접근**:
  - 개발 환경: `https://ddukdoc.shop`
  - 운영 환경: `https://j12b108.p.ssafy.io`
  - Jenkins: `http://[EC2_IP]:9090`
  - Kibana: `http://[EC2_IP]:5601`

---

2. CI/CD를 포함한 배포 방식

이 섹션에서는 Jenkins를 활용한 자동화된 CI/CD 파이프라인을 설정하고 배포하는 과정을 설명합니다.

2.1. Jenkins 설정

2.1.1. Jenkins 초기 설정

1. **컨테이너 실행**:
   - `docker-compose.yml`에서 `jenkins` 서비스로 실행.
2. **초기 비밀번호 확인**:

```
docker exec jenkins cat /var/jenkins_home/secrets/initialAdminPassword
```

3. **웹 인터페이스 접속**:
   - `http://[EC2_IP]:9090`에 접속.
   - 초기 비밀번호 입력 후 관리자 계정 생성.
4. **Jenkins 내부 Docker 설치**:

```
docker exec -it -u root jenkins bash
apt update
apt install -y apt-transport-https ca-certificates curl software-properties-common
curl -fsSL <https://download.docker.com/linux/debian/gpg> | apt-key add -
add-apt-repository "deb [arch=amd64] <https://download.docker.com/linux/debian> $(lsb_release -cs) stable"
apt update
apt install -y docker-ce
exit
```

5. **DooD (Docker outside of Docker) 설정**:
   - `/var/run/docker.sock` 권한 문제 해결:

```
docker exec -it -u root jenkins bash
groupadd docker
groupmod -g 944 docker
```

```
usermod -aG docker jenkins
exit
```

- **호스트 GID 확인**:

```
ls -l /var/run/docker.sock
getent group docker
```

2.1.2. Jenkins 플러그인 설치

- **플러그인**:
  - Generic Webhook Trigger
  - Gitlab
  - Gitlab API
  - Gitlab Authentication
  - Mattermost Notification
  - Docker Pipeline
  - NodeJS Plugin
  - SonarQube Scanner for Jenkins
- **설치**:
  - `Manage Jenkins` > `Manage Plugins` > `Available` 탭에서 설치.

2.1.3. 글로벌 도구 설정

1. **Node.js**:
   - `Manage Jenkins` > `Global Tool Configuration`
   - `NodeJS installations`:
     - Name: `NodeJS 22.14`
     - Version: `NodeJS 22.14.0`
     - Install automatically
2. **SonarQube Scanner**:
   - `SonarQube Scanner installations`:

    - Name: `sonarqube`
    - Version: `SonarQube Scanner 7.1.0.4889`
    - Install automatically

2.1.4. Jenkins Item 설정

- **Item**:
    - `dev` : 개발 환경 배포
    - `prod` : 운영 환경 배포
- **GitLab 연결**:
    - `Source Code Management`에서 GitLab 설정.
    - Repository URL: [GitLab_Repository_URL]
    - Branch: `dev` (개발), `main` (운영)

누락 확인: GitLab Repository URL이 제공되지 않았습니다. 실제 URL을 명시해야 합니다.

- **빌드 매개변수** (String Parameter):
    - **dev Item**:
        - `DB_URL` : `jdbc:mariadb://stg-yswa-kr-practice-db-master.mariadb.database.azure.com:3306/s12p22b108?serverTimezone=UTC&useUnicode=true&characterEncoding=utf8`
        - `DB_USERNAME` : `S12P22B108`
        - `DB_PASSWORD` : `dduk108doc`
        - `SPRING_PROFILE` : `dev`
        - `DEPLOY_PATH` : `/home/ubuntu/nginx/html/dev`
        - `DEPLOY_ENV` : `development`
        - `URL` : `https://ddukdoc.shop`
    - **prod Item**:
        - `DB_URL` : `jdbc:mariadb://rds-mariadb-stg.ssafyapp.com:3306/dukd0c8`
        - `DB_USERNAME` : `dukd0c8`
        - `DB_PASSWORD` : `duk!Rspa19`

- `REDIS_HOST` : `redis-stg.ssafyapp.com`
- `URL` : `https://j12b108.p.ssafy.io`
- `SPRING_PROFILE` : `prod`
- `DEPLOY_PATH` : `/home/ubuntu/nginx/html/prod`
- `DEPLOY_ENV` : `production`

2.1.5. Credentials 설정

- `Manage Jenkins` > `Manage Credentials`에서 Secret File 등록:

1. **APPLICATION-SECRET** : 실제 값으로 대체 필요 요소는 다음과 같이 표현 {{key}}

```
spring:
  data:
    redis:
      password: b108ddukdoc
      port: 6379
  jwt:
    secret: ddukDocDdukdocddukDocDdukdocddukDocDdukdocddukDocDdukdocddukDocDdukdocDDUKDDoCddUkDDOcDDUKDDoCddUkDDOcDDUKDDoCddUkDDOcDDUKDDoCddUkDDOcDDUKDDoCddUkDDOcDDUKDDoCddUkDDOc
  oauth:
    kakao:
      client-id: {{KAKAO 개발자 센터 생성 key}}
    ssafy:
      client-id: {{SSAFY 개발자 센터 생성 key}}
      client-secret: {{SSAFY 개발자 센터 생성 key}}
  encryption:
    kek: {{충분한 길이의 암호화}}
  ssafy:
    open-api:
      key: {{SSAFY 개발자 센터 제공 key}}
  cloud:
    aws:
      credentials:
        accessKey: {{S3 사용위해 ssafy에서 제공받은 key}}
        secretKey: {{S3 사용위해 ssafy에서 제공받은 key}}
```

```
    region:
      static: ap-northeast-2
    stack:
      auto: false
    s3:
      bucket: ssafyapp-stg-project-prv-data
    kms:
      key: {{AWS KMS 사용위해 생성한 key}}
      credentials:
        accessKey: {{AWS KMS 사용위해 생성한 IAM 계정의 access key}}
        secretKey: {{AWS KMS 사용위해 생성한 IAM 계정의 access key}}
  blockchain:
    private-key: {{METAMASK 개인 key}}
    address: {{BLOCKCHAIN 배포 주소}}
    contractAddress: {{SMARTCONTRACT 배포 주소}}
    baseurl: "<https://j12b108.p.ssafy.io/blockchain/tokens/>"
```

2. **frontend-env-file**:

```
VITE_API_MOCKING=enabled
VITE_NODE_ENV=master
VITE_USE_MSW=false
VITE_API_URL=${URL}
VITE_CONTRACT=/api/contract
VITE_RETURN=/return
VITE_KAKAO_CLIENT_ID=3a14a618b0ba1ad22e03d90f026a9ce5
VITE_KAKAO_REDIRECT_URI=${URL}/api/oauth/kakao/login
VITE_SSAFY_CLIENT_ID=b68dd054-ab9f-435e-801b-a2434d0f2e4a
VITE_SSAFY_REDIRECT_URI=${URL}/api/oauth/ssafy/login
VITE_KAKAO_API=9113ce838b366d7d9b38490e7781ee28
VITE_SSAFY_CONTRACT=/api/ssafy/contract
```

3. **sonarqube-token**:
    - SonarQube에서 생성한 사용자 토큰.
4. **blockchain-api-env**:

```
INFURA_URL=https://polygon-mainnet.infura.io/v3/e4bde91de46e44049faf5b0fe8ceddf9
CONTRACT_ADDRESS=0x3a276cd278a021523d964012b2f23d060fd425bf
OWNER_ADDRESS=0x5942922B40f897d105ce2D0512DE1cC37E7e6c00
PRIVATE_KEY=0xb57abd66e6421638e17aec78e74d6019fe1a4a8668dd5cb68ae060aa52e0582a
```

2.2. CI/CD 파이프라인

- **Jenkinsfile**:
  - 프로젝트 루트에 `Jenkinsfile` 배치 (제공된 파일 사용).
  - 단계:
    - 변경 사항 확인 (프론트엔드, 백엔드, 블록체인 API)
    - 빌드 및 Docker 이미지 생성
    - SonarQube 분석 (개발 환경)
    - 배포 (Blue-Green)
    - 헬스체크
    - 트래픽 전환
- **Webhook**:
  - GitLab에서 Webhook 설정:
    - URL: `http://[EC2_IP]:9090/generic-webhook-trigger/invoke`
    - Trigger: Push events, Merge request events

2.3. Blue-Green 배포

- **개발 환경**:
  - `backend-dev-blue` (8085) ↔ `backend-dev-green` (8086)
  - `dev.conf`에서 `upstream backend_dev` 업데이트.
- **운영 환경**:
  - `backend-prod-blue` (8080) ↔ `backend-prod-green` (8081)

- `default.conf`에서 `upstream backend_prod` 업데이트.

2.4. 모니터링 및 알림

- **Mattermost**:
  - 빌드 성공/실패 알림.
  - Endpoint: `https://meeting.ssafy.com/hooks/pmu7f349wb8y5q1djoar94k8mc`
  - Channel: `78077804f0d7f41a4976e15a024145e8`
- **ELK**:
  - Kibana: `http://[EC2_IP]:5601`

3. SMART CONTRACT 코드

REMIX에서 POLYGON 메인 네트워크에 배포

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract DocumentRegistry {

    struct Document {
        string name;
        string uri;
        string hash;
        uint256 timestamp;
        address owner;
    }

    mapping(string => Document) private documents;
    string[] private documentNames;

    event DocumentRegistered(string name, string hash, string uri);
    event DocumentDeleted(string name);

    function registerDocument(string memory name, string memory hash, strin
```

```
        require(bytes(documents[name].name).length == 0, "Document already e

        documents[name] = Document({
            name: name,
            uri: uri,
            hash: hash,
            timestamp: block.timestamp,
            owner: msg.sender
        });

        documentNames.push(name);

        emit DocumentRegistered(name, hash, uri);
    }

    function getDocument(string memory name) public view returns (string mer
        Document memory doc = documents[name];
        require(bytes(doc.name).length > 0, "Document not found");

        return (doc.uri, doc.hash, doc.timestamp);
    }

    function getAllDocuments() public view returns (Document[] memory) {
        Document[] memory result = new Document[](documentNames.length);

        for (uint i = 0; i < documentNames.length; i++) {
            result[i] = documents[documentNames[i]];
        }

        return result;
    }

    function deleteDocument(string memory name) public {
        Document memory doc = documents[name];
        require(doc.owner == msg.sender, "Only owner can delete");

        delete documents[name];
```

```
        // remove from documentNames
        for (uint i = 0; i < documentNames.length; i++) {
            if (keccak256(abi.encodePacked(documentNames[i])) == keccak256(a
                documentNames[i] = documentNames[documentNames.length - 1];
                documentNames.pop();
                break;
            }
        }

        emit DocumentDeleted(name);
    }
}
```