포팅 매뉴얼

1. 사용 도구

- 이슈 관리: Notion, Jira
- 형상 관리: GitLab
- 커뮤니케이션: MatterMost, Discord
- 디자인: Figma
- CI/CD: Jenkins, Docker

2. 개발 도구

2.1. FrontEnd

2.1.1. 주요 개발 도구

1. **Next.js** (15.0.1) - React 기반 프레임워크로 SSR 및 SSG 지원.
2. **TypeScript** (5.x) - JavaScript에 타입을 추가하여 코드의 안정성 향상.
3. **ESLint** - 코드의 일관성 및 품질을 유지하는 린팅 도구.
    - `eslint-config-next` (15.0.1) - Next.js의 기본 ESLint 설정.
    - `eslint` (8.x) - 기본 ESLint 패키지.
4. **PostCSS** (8.x) - CSS 전처리기.
5. **TailwindCSS** (3.4.1) - 유틸리티 클래스 기반 CSS 프레임워크.

2.1.2. 주요 프론트엔드 라이브러리

1. **React** (18.3.1) - 사용자 인터페이스를 구성하기 위한 라이브러리.
2. **React DOM** (18.3.1) - React용 DOM 렌더링 라이브러리.
3. **Axios** (1.7.7) - HTTP 클라이언트로 API 요청에 사용.

4. **React DatePicker** (7.5.0) - 날짜 선택 컴포넌트.
5. **React Resizable** (3.0.5) - 드래그 앤 드롭으로 크기 조절 가능한 컴포넌트.
6. **React Select** (5.8.2) - 드롭다운 선택 UI 구현 라이브러리.

2.1.3. 아이콘 및 UI 구성

1. **FontAwesome Free Solid Icons** (6.6.0) - 다양한 무료 아이콘 세트.
2. **React FontAwesome** (0.2.2) - React에서 FontAwesome 아이콘 사용 지원.

2.1.4. 기타 유틸리티 및 도구

1. **BMP-JS** (0.1.0) - BMP 이미지 처리를 위한 라이브러리.

2.1.5. 타입 정의

- **@types** 패키지들:
  - `@types/bmp-js` (0.1.2) - BMP-JS를 위한 타입 정의.
  - `@types/node` (20.x) - Node.js의 타입 정의.
  - `@types/react` (18.x) - React의 타입 정의.
  - `@types/react-dom` (18.x) - React DOM의 타입 정의.
  - `@types/react-resizable` (3.0.8) - React Resizable의 타입 정의.

2.1.6. 빌드 및 최적화

1. **Turbopack** - Next.js에 포함된 고성능 번들러.
2. **TypeScript** (5.x) - JavaScript와 TypeScript 프로젝트를 통합적으로 관리.

이 포맷은 위의 데이터를 간결하게 정리하며 이해하기 쉽게 구성된 형태로, 노션이나 다른 문서 관리 툴에 적합합니다.

2.2. BackEnd

2.1.1. Integrated

- **개발 언어**: Java(21)

- **빌드 도구**: Gradle

플러그인

1. **Java** - 기본 Java 빌드 지원.
2. **Spring Boot** (3.3.4) - Spring Boot 기반 애플리케이션 개발 지원.
3. **Spring Dependency Management** (1.1.6) - Spring 프로젝트 종속성 관리.
4. **Asciidoctor** (3.3.2) - 문서화 도구.

프로젝트 정보

- **Group**: `com.semony`
- **Version**: `0.0.1-SNAPSHOT`
- **Java Toolchain**: Java 21

Repositories

- **Maven Central** - 패키지 의존성 관리.

Dependencies

1. **Implementation Dependencies**
   - `com.fasterxml.jackson.datatype:jackson-datatype-jsr310` (2.13.4) - Java 8 날짜 및 시간 지원을 위한 Jackson 모듈.
   - `com.fasterxml.jackson.core:jackson-databind` (2.15.0) - JSON 데이터 바인딩 라이브러리.
   - `com.h2database:h2` - H2 인메모리 데이터베이스.
   - `org.springframework.boot:spring-boot-starter-data-jpa` - Spring Data JPA 지원.
   - `org.springframework.boot:spring-boot-starter-web` - Spring Web 모듈.
2. **Compile Only**
   - `org.projectlombok:lombok` - Java 코드 간소화를 위한 어노테이션 제공.
3. **Development Only**
   - `org.springframework.boot:spring-boot-devtools` - 개발 중 자동 재시작 및 편리한 기능 제공.

4. **Runtime Only**
    - `org.postgresql:postgresql` - PostgreSQL 데이터베이스 드라이버.
5. **Annotation Processor**
    - `org.projectlombok:lombok` - Lombok을 위한 어노테이션 프로세서.
6. **Test Dependencies**
    - `org.springframework.boot:spring-boot-starter-test` - 테스트 관련 모듈.
    - `org.springframework.restdocs:spring-restdocs-mockmvc` - REST Docs 문서화 지원.
    - `org.junit.platform:junit-platform-launcher` - JUnit 플랫폼 런처.

테스트 및 문서화

1. **테스트 설정**
    - 테스트 출력 디렉토리: `build/generated-snippets`
    - 테스트 플랫폼: JUnit Platform
2. **Asciidoctor 문서화**
    - 입력 디렉토리: `build/generated-snippets`
    - 테스트 의존: `test` 작업 실행 후 Asciidoctor 작업 실행.

2.1.2. Maker

- **개발 언어**: Java(21)
- **빌드 도구**: Gradle

플러그인

1. **Java** - 기본 Java 빌드 지원.
2. **Spring Boot** (3.3.5) - Spring Boot 기반 애플리케이션 개발 지원.
3. **Spring Dependency Management** (1.1.6) - Spring 프로젝트 종속성 관리.

프로젝트 정보

- **Group**: `com.semony`
- **Version**: `0.0.1-SNAPSHOT`

- **Java Toolchain**: Java 21

Repositories

- **Maven Central** - 패키지 의존성 관리.

Dependencies

1. **Implementation Dependencies**
   - `org.springframework.boot:spring-boot-starter-data-jpa` - Spring Data JPA 지원.
   - `org.springframework.boot:spring-boot-starter-data-mongodb` - MongoDB 지원.
   - `org.springframework.boot:spring-boot-starter-web` - Spring Web 모듈.
   - `org.springframework.boot:spring-boot-starter-webflux` - Spring WebFlux 모듈.
   - `org.postgresql:postgresql` (42.7.3) - PostgreSQL 데이터베이스 드라이버.
   - `org.slf4j:slf4j-api` (2.0.9) - 로깅 API.
   - `ch.qos.logback:logback-classic` (1.4.12) - SLF4J를 위한 Logback 구현.
2. **Compile Only**
   - `org.projectlombok:lombok` - Java 코드 간소화를 위한 어노테이션 제공.
3. **Runtime Only**
   - `com.h2database:h2` - H2 인메모리 데이터베이스.
4. **Annotation Processor**
   - `org.projectlombok:lombok` - Lombok을 위한 어노테이션 프로세서.
5. **Test Dependencies**
   - `org.springframework.boot:spring-boot-starter-test` - 테스트 관련 모듈.
6. **Test Runtime Only**
   - `org.junit.platform:junit-platform-launcher` - JUnit 플랫폼 런처.

테스트 설정

- 테스트 플랫폼: JUnit Platform

2.1.3. Module

- **개발 언어**: Python
- **프레임워크**: FastAPI (0.115.4)

Dependencies

1. **FastAPI** (0.115.4) - Python 기반 웹 프레임워크로, 데이터 검증 및 OpenAPI 지원.
2. **Starlette** (0.41.2) - FastAPI의 기본 ASGI 툴킷.
3. **Uvicorn** (0.32.0) - ASGI 서버 구현체로, FastAPI 애플리케이션 실행에 사용.
4. **Pydantic** (2.9.2) - 데이터 모델링 및 검증 라이브러리.
    - **Pydantic Core** (2.23.4) - Pydantic의 핵심 데이터 검증 모듈.
5. **AnyIO** (4.6.2.post1) - 비동기 I/O 프레임워크로, 다양한 백엔드 지원.
6. **H11** (0.14.0) - HTTP/1.1용 프로토콜 구현.
7. **Sniffio** (1.3.1) - 비동기 라이브러리 간의 런타임 감지.
8. **Annotated Types** (0.7.0) - Python `typing`에 주석 기반 타입 지원.
9. **Typing Extensions** (4.12.2) - Python 버전에 상관없이 추가 타입 힌트를 제공.
10. **Click** (8.1.7) - 명령줄 인터페이스 생성 도구.
11. **Colorama** (0.4.6) - CLI 텍스트에 색상을 추가하는 라이브러리.
12. **IDNA** (3.10) - 국제화 도메인 이름 처리 라이브러리.

---

3. 서버 환경 설정 (Jenkins, Docker, Nginx)

3.1. 서버 별 패키지 설치

3.1.1. Main LightSale Instance

- 패키지 목록 업데이트

```
sudo apt update
sudo apt upgrade
```

- 도커 설치

```
// 사전에 설치된 도커 관련 패키지 삭제
for pkg in docker.io docker-doc docker-compose docker-compose

// 도커 apt 셋업
# Add Docker's official GPG key:
sudo apt-get update
sudo apt-get install ca-certificates curl
sudo install -m 0755 -d /etc/apt/keyrings
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg
sudo chmod a+r /etc/apt/keyrings/docker.asc

# Add the repository to Apt sources:
echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/
  $(. /etc/os-release && echo "$VERSION_CODENAME") stable" |
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
sudo apt-get update

// 도커 패키지 설치
sudo apt-get install docker-ce docker-ce-cli containerd.io do
```

- 젠킨스 이미지 설치 및 실행

```
docker pull jenkins/jenkins:lts-jdk21

docker run -d --name jenkins -p 8000:8080 -p 50000:50000 \
-v /var/jenkins_home:/var/jenkins_home \
-v /var/run/docker.sock:/var/run/docker.sock \
-u root
jenkins/jenkins:lts-jdk21
```

- Nginx 설치

```
sudo apt install nginx
```

- AWS 보안 그룹 설정

애플리케이션	프로토콜	포트 또는 범위 / 코드	제한:
Ping(ICMP)	ICMP		모든 IPv4 주소
SSH	TCP	22	모든 IPv4 주소 Lightsail 브라우저 SSH/RDP
HTTP	TCP	80	59.20.195.0 → 59.20.195.255
사용자 지정	TCP	88	모든 IPv4 주소
HTTPS	TCP	443	모든 IPv4 주소
사용자 지정	TCP	3000	모든 IPv4 주소

3.1.2. Maker EC2 Instance

- 패키지 목록 업데이트

```
sudo apt update
sudo apt upgrade
```

- 도커 설치

```
// 사전에 설치된 도커 관련 패키지 삭제
for pkg in docker.io docker-doc docker-compose docker-compose

// 도커 apt 셋업
# Add Docker's official GPG key:
sudo apt-get update
sudo apt-get install ca-certificates curl
sudo install -m 0755 -d /etc/apt/keyrings
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg
sudo chmod a+r /etc/apt/keyrings/docker.asc

# Add the repository to Apt sources:
echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/
  $(. /etc/os-release && echo "$VERSION_CODENAME") stable" |
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
sudo apt-get update
```

```
// 도커 패키지 설치
sudo apt-get install docker-ce docker-ce-cli containerd.io do
```

- 포트 개방

```
tcp        0      0 0.0.0.0:8080            0.0.0.0:*               LISTEN
```

3.1.3. Module#1 LightSale Instance

- 패키지 목록 업데이트

```
sudo apt update
sudo apt upgrade
```

- 도커 설치

```
// 사전에 설치된 도커 관련 패키지 삭제
for pkg in docker.io docker-doc docker-compose docker-compose

// 도커 apt 셋업
# Add Docker's official GPG key:
sudo apt-get update
sudo apt-get install ca-certificates curl
sudo install -m 0755 -d /etc/apt/keyrings
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg
sudo chmod a+r /etc/apt/keyrings/docker.asc

# Add the repository to Apt sources:
echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/
  $(. /etc/os-release && echo "$VERSION_CODENAME") stable" |
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
sudo apt-get update

// 도커 패키지 설치
sudo apt-get install docker-ce docker-ce-cli containerd.io do
```

- 보안 그룹 설정

애플리케이션	프로토콜	포트 또는 범위 / 코드	제한:	
Ping(ICMP)	ICMP		모든 IPv4 주소	
사용자 지정	TCP	22 → 23	모든 IPv4 주소	
사용자 지정	TCP	2049	모든 IPv4 주소	
사용자 지정	TCP	8080 → 8084	모든 IPv4 주소	

3.1.4. Module#2 LightSail Instance

// 3.1.3. 설정과 동일

3.2. Jenkins Pipeline

S	W	Name ↓	최근 성공	최근 실패	최근 소요 시간	
✓	☁	frontend-pipeline	5 hr 1 min #36	3 days 4 hr #34	4 min 48 sec	▷
✓	☀	integrated-pipeline	5 days 7 hr #93	10 days #74	45 sec	▷
✓	☁	maker-pipeline	5 days 23 hr #24	6 days 0 hr #23	56 sec	▷
✓	☀	module-pipeline	5 days 7 hr #23	10 days #8	41 sec	▷

총 4개의 파이프라인 구축

- Jenkins Plugin

```
GitLabv, Docker, Docker Pipeline, Generic Web Trigger 설치
```

- SSH키 호스트에 등록

```
// 컨테이너 내부 진입
docker exec -it jenkins /bin/bash
// ssh 키 생성
ssh-keygen
// ssh 키 복사
vi ~/.ssh/id_rsa.pub

// 각 호스트에 ssh키 등록
sudo vi ~/.ssh/authorized_keys
```

3.2.1. Integrated Pipeline

- JenkinsFile

```
pipeline {
    agent any

    tools {
        dockerTool 'Docker'
    }

    environment {
        repository = "tpwls1355/integrated"
        DOCKERHUB_CREDENTIALS = credentials('dockerhub-usernar
    }

    stages {
        stage('Checkout') {
            steps {
                git branch: 'integrated/develop',
                    credentialsId: 'gitlab-username-password'
                    url: 'https://lab.ssafy.com/s11-final/S11
            }
        }

        stage('Build') {
            steps {
                echo 'Building application...'
                dir('integrated') {
                    sh 'chmod +x ./gradlew'
                    sh './gradlew clean build -x test'
                }
            }
        }

        stage('Build Docker Image') {
            steps {
```

```
                echo 'Building Docker image...'
                dir('integrated') {
                    sh 'docker build -t integrated:latest .'
                }
            }
        }

        stage('Push Docker Image') {
            steps {
                script {
                    echo 'Pushing Docker image to Docker Hub.

                    // Docker Hub 로그인
                    withCredentials([usernamePassword(credent
                        sh 'docker login -u $DOCKERHUB_USERNA
                    }

                    // 이미지에 태그 추가
                    sh 'docker tag integrated:latest $reposit

                    // Docker Hub에 이미지 푸시
                    sh 'docker push $repository:latest'
                }
            }
        }

        stage('Deploy') {
            steps {
                echo 'Deploying application...'
                dir('integrated') {
                    script {
                        // SSH를 사용하여 외부 호스트에서 docker-c
                        echo 'Deploying application on extern
                        sh '''
                            ssh -t -t ubuntu@localhost "
                                cd /opt/semony
                                # 기존 컨테이너가 있다면 안전하게 중
                                if [ "$(docker ps -q -f name=
```

```
                                    echo 'Stopping existing c
                                    docker-compose -f integra
                                    echo 'Existing container
                                else
                                    echo 'No existing contain
                                fi
                                # 새로운 컨테이너 시작
                                docker image prune -f
                                echo 'Starting new container.
                                docker-compose -f integrated-
                                echo 'New container started s
                            "
                        '''
                    }
                }
            }
        }

        stage('Cleanup Dangling Images') {
            steps {
                script {
                    echo 'Cleaning up dangling Docker images.

                    // Dangling 이미지만 삭제
                    sh '''
                        docker images --filter "dangling=true
                            repo_tag=$(echo $image_info | awk
                            image_id=$(echo $image_info | awk

                            if [ "$repo_tag" = "tpwls1355/int
                                echo "Removing dangling image
                                docker rmi -f $image_id
                            fi
                        done
                    '''
                }
            }
        }
```

```
    }

    post {
        success {
            echo 'Pipeline succeeded!'
        }
        failure {
            echo 'Pipeline failed.'
        }
    }
}
```

3.2.2. Fronted Pipeline

- JenkinsFile

```
pipeline {
    agent any

    tools {
        dockerTool 'Docker'
        nodejs 'NodeJS'
    }

    environment {
        repository = "tpwls1355/semony-front"
        DOCKERHUB_CREDENTIALS = credentials('dockerhub-usernar
        SSH_HOST = "ubuntu@localhost"
        DEPLOY_PATH = "/opt/semony"
    }

    stages {
        stage('Checkout') {
            steps {
                git branch: 'front/develop',
                    credentialsId: 'gitlab-username-password'
                    url: 'https://lab.ssafy.com/s11-final/S11
```

```
            }
        }

        stage('Build Docker Image') {
            steps {
                echo 'Building Docker image for React app wit
                dir('semony_fe') {
                    sh 'docker build --cache-from semony-fron
                }
            }
        }

        stage('Push Docker Image') {
            steps {
                script {
                    echo 'Pushing Docker image to Docker Hub.

                    withCredentials([usernamePassword(credent
                        sh 'docker login -u $DOCKERHUB_USERNA
                    }
                    sh 'docker tag semony-front:latest $repos
                    sh 'docker push $repository:latest'
                }
            }
        }

        stage('Deploy') {
            steps {
                echo 'Deploying React app with conditional up
                dir('semony_fe') {
                    script {
                        sh '''
                            ssh -t -t $SSH_HOST "
                                cd $DEPLOY_PATH
                                docker-compose -f front-docke
                                docker-compose -f front-docke
                            "
                        '''
```

```
                    }
                }
            }
        }

        stage('Cleanup Dangling Images') {
            steps {
                script {
                    echo 'Cleaning up dangling Docker images.
                    sh 'docker image prune -f --filter "dangl
                }
            }
        }
    }

    post {
        success {
            echo 'Pipeline for React app succeeded!'
        }
        failure {
            echo 'Pipeline for React app failed.'
        }
    }
}
```

3.2.3. Maker Pipeline

- JenkinsFile

```
pipeline {
    agent any

    tools {
        dockerTool 'Docker'
    }

    environment {
        repository = "tpwls1355/maker"
```

```
        DOCKERHUB_CREDENTIALS = credentials('dockerhub-usernam
        SSH_HOST = "ubuntu@k11s109.p.ssafy.io"
        DEPLOY_PATH = "/opt/semony"
    }

    stages {
        stage('Checkout') {
            steps {
                git branch: 'maker/develop',
                    credentialsId: 'gitlab-username-password'
                    url: 'https://lab.ssafy.com/s11-final/S11
            }
        }

        stage('Prepare application.yml') {
            steps {
                dir('maker') {
                    script {
                        // 디렉터리가 없는 경우에만 생성
                        sh '''
                            if [ ! -d ./src/main/resources ];
                                mkdir -p ./src/main/resources
                            fi
                        '''

                        // Config File Provider 플러그인으로 app
                        configFileProvider([configFile(fileId
                            echo "application.yml 파일을 작업 공
                        }
                    }
                }
            }
        }


        stage('Build') {
            steps {
                echo 'Building application...'
```

```
                dir('maker') {
                    sh 'chmod +x ./gradlew'
                    sh './gradlew clean build -x test'
                }
            }
        }

        stage('Build Docker Image') {
            steps {
                echo 'Building Docker image...'
                dir('maker') {
                    sh 'docker build -t maker:latest .'
                }
            }
        }

        stage('Push Docker Image') {
            steps {
                script {
                    echo 'Pushing Docker image to Docker Hub.

                    // Docker Hub 로그인
                    withCredentials([usernamePassword(credent
                        sh 'docker login -u $DOCKERHUB_USERNA
                    }

                    // 이미지에 태그 추가
                    sh 'docker tag maker:latest $repository:l

                    // Docker Hub에 이미지 푸시
                    sh 'docker push $repository:latest'
                }
            }
        }

        stage('Deploy') {
            steps {
                echo 'Deploying React app with conditional up
```

```
                dir('maker') {
                    script {
                        sh '''
                            ssh -t -t $SSH_HOST "
                                cd $DEPLOY_PATH
                                docker-compose -f maker-docke
                                docker-compose -f maker-docke
                            "
                        '''
                    }
                }
            }
        }

        stage('Cleanup Dangling Images') {
            steps {
                script {
                    echo 'Cleaning up dangling Docker images.
                    sh 'docker image prune -f --filter "dangl
                }
            }
        }

    }

    post {
        success {
            echo 'Pipeline succeeded!'
        }
        failure {
            echo 'Pipeline failed.'
        }
    }
}
```

3.2.4. Module Pipeline

- JenkinsFile

```
pipeline {
    agent any

    tools {
        dockerTool 'Docker'
    }

    environment {
        repository = "tpwls1355/module"
        DOCKERHUB_CREDENTIALS = credentials('dockerhub-usernar
    }

    stages {
        stage('Checkout') {
            steps {
                git branch: 'module/develop',
                    credentialsId: 'gitlab-username-password'
                    url: 'https://lab.ssafy.com/s11-final/S11
            }
        }

        stage('Build Docker Image') {
            steps {
                echo 'Building Docker image for FastAPI modul
                dir('module') {
                    sh 'docker build -t module:latest .'
                }
            }
        }

        stage('Push Docker Image') {
            steps {
                script {
                    echo 'Pushing Docker image to Docker Hub.
                    withCredentials([usernamePassword(credent
                        sh 'docker login -u $DOCKERHUB_USERNA
                    }
                    sh 'docker tag module:latest $repository:
```

```
                    sh 'docker push $repository:latest'
                }
            }
        }

        stage('Deploy') {
            steps {
                echo 'Deploying FastAPI module on external ho
                dir('module') {
                    script {
                        sh '''
                            ssh -t -t ubuntu@3.36.105.208 "
                                cd /opt/module
                                echo 'Stopping existing conta
                                docker-compose -f module-dock
                                docker image prune -a -f  # 불
                                echo 'Starting new container.
                                docker-compose -f module-dock
                                echo 'New container started s
                            "
                        '''
                        sh '''
                            ssh -t -t ubuntu@3.36.54.152 "
                                cd /opt/module
                                echo 'Stopping existing conta
                                docker-compose -f module-dock
                                docker image prune -a -f  # 불
                                echo 'Starting new container.
                                docker-compose -f module-dock
                                echo 'New container started s
                            "
                        '''

                    }
                }
            }
        }
```

```
        stage('Cleanup Dangling Images') {
            steps {
                script {
                    echo 'Cleaning up dangling Docker images.
                    sh '''
                        docker images --filter "dangling=true
                            repo_tag=$(echo $image_info | awk
                            image_id=$(echo $image_info | awk

                            if [ "$repo_tag" = "tpwls1355/mod
                                echo "Removing dangling image
                                docker rmi -f $image_id
                            fi
                        done
                    '''
                }
            }
        }
    }

    post {
        success {
            echo 'Pipeline succeeded!'
        }
        failure {
            echo 'Pipeline failed.'
        }
    }
}
```

3.3. DockerFile, DockerCompose

3.3.1. Integrated

- DockerFile

```
# 1. Base Image로 OpenJDK 21 사용
FROM openjdk:21-jdk-slim

# 2. JAR 파일을 컨테이너로 복사
ARG JAR_FILE=build/libs/integrated-*-SNAPSHOT.jar
COPY ${JAR_FILE} integrated.jar

ENTRYPOINT ["java", "-jar", "/integrated.jar"]
```

- DockerCompose

```
version: '3.8'

services:
  integrated-server:                    # Spring Boot 애플리케이
    image: tpwls1355/integrated:latest              # Docker
    privileged: true
    ports:
      - "8090:8080"             # 호스트의 8080 포트를 컨테이너의 8
    environment:
      - SPRING_DATASOURCE_URL=jdbc:postgresql://db:5432/defec
      - SPRING_DATASOURCE_USERNAME=semony
      - SPRING_DATASOURCE_PASSWORD=s109
      - SPRING_JPA_HIBERNATE_DDL_AUTO=update
    depends_on:
      - db                      # db 서비스가 시작된 후 integrate
    volumes:
      - /mnt:/var/libs/data
  db:
    image: postgres:latest
    ports:
      - "5432:5432"
    environment:
      - POSTGRES_DB=defect_db
      - POSTGRES_USER=semony
      - POSTGRES_PASSWORD=s109
    volumes:
      - pgdata:/var/lib/postgresql/data
```

```
volumes:
  pgdata:
```

3.3.2. Maker

- DockerFile

```
# 1. 베이스 이미지 선택 (Python 3.9 버전 사용)
FROM python:3.9-slim

# 2. 작업 디렉토리 생성 및 설정
WORKDIR /app

# 3. 필요한 패키지 설치를 위한 요구사항 파일 복사
COPY requirements.txt .

# 4. 패키지 설치
RUN pip install --no-cache-dir -r requirements.txt

# 5. 애플리케이션 코드 복사
COPY . .

# 6. FastAPI 애플리케이션 실행 (uvicorn 사용)
CMD ["uvicorn", "main:app", "--host", "0.0.0.0"]
```

- DockerCompose

```
version: '3.8'

services:
  maker-server:
    image: tpwls1355/maker:latest
    ports:
      - "8080:8080"
    environment:
      - SPRING_DATA_MONGODB_URI=mongodb://semony:s109@db:2701
      - SPRING_DATASOURCE_URL=jdbc:postgresql://semony-s109.s
```

```
      - SPRING_DATASOURCE_USERNAME=semony
      - SPRING_DATASOURCE_PASSWORD=s109
      - MODULE_IP_MIW7_51=http://3.36.54.152:8081
      - MODULE_IP_MIW7_61=http://3.36.54.152:8082
      - MODULE_IP_EWIM1_36=http://3.36.54.152:8083
      - MODULE_IP_EWIM1_46=http://3.36.54.152:8084
      - MODULE_IP_MIW7_52=http://3.36.105.208:8081
      - MODULE_IP_MIW7_62=http://3.36.105.208:8082
      - MODULE_IP_EWIM2_36=http://3.36.105.208:8083
      - MODULE_IP_EWIM2_46=http://3.36.105.208:8084
    depends_on:
      - db

  db:
    image: mongo:latest
    ports:
      - "27017:27017"
    environment:
      - MONGO_INITDB_DATABASE=summary
      - MONGO_INITDB_ROOT_USERNAME=semony
      - MONGO_INITDB_ROOT_PASSWORD=s109
    volumes:
      - mongodbdata:/data/db

volumes:
  mongodbdata:
```

3.3.3. Module

- DockerFile

```
# 1. 베이스 이미지 선택 (Python 3.9 버전 사용)
FROM python:3.9-slim

# 2. 작업 디렉토리 생성 및 설정
WORKDIR /app

# 3. 필요한 패키지 설치를 위한 요구사항 파일 복사
```

```
COPY requirements.txt .

# 4. 패키지 설치
RUN pip install --no-cache-dir -r requirements.txt

# 5. 애플리케이션 코드 복사
COPY . .

# 6. FastAPI 애플리케이션 실행 (uvicorn 사용)
CMD ["uvicorn", "main:app", "--host", "0.0.0.0"]
```

3.3.3.1. Module#1

- DockerCompose

```
version: '3.8'

services:
  MIW7-51:
    image: tpwls1355/module:latest
    container_name: MIW7-51
    privileged: true
    volumes:
      - /nfs_shared:/var/lib/result
    ports:
      - "8081:8000"
    environment:
      DOCKER_TLS_CERTDIR: ""
    network_mode: bridge

  MIW7-61:
    image: tpwls1355/module:latest
    container_name: MIW7-61
    privileged: true
    volumes:
      - /nfs_shared:/var/lib/result
    ports:
```

```
      - "8082:8000"
    environment:
      DOCKER_TLS_CERTDIR: ""
    network_mode: bridge

  EWIM1-36:
    image: tpwls1355/module:latest
    container_name: EWIM1-36
    privileged: true
    volumes:
      - /nfs_shared:/var/lib/result
    ports:
      - "8083:8000"
    environment:
      DOCKER_TLS_CERTDIR: ""
    network_mode: bridge

  EWIM1-46:
    image: tpwls1355/module:latest
    container_name: EWIM1-46
    privileged: true
    volumes:
      - /nfs_shared:/var/lib/result
    ports:
      - "8084:8000"
    environment:
      DOCKER_TLS_CERTDIR: ""
    network_mode: bridge
```

3.3.3.2. Module#2

```
version: '3.8'

services:
  MIW7-52:
    image: tpwls1355/module:latest
    container_name: MIW7-52
    privileged: true
```

```
    volumes:
      - /nfs_shared:/var/lib/result
    ports:
      - "8081:8000"
    environment:
      DOCKER_TLS_CERTDIR: ""

  MIW7-62:
    image: tpwls1355/module:latest
    container_name: MIW7-62
    privileged: true
    volumes:
      - /nfs_shared:/var/lib/result
    ports:
      - "8082:8000"
    environment:
      DOCKER_TLS_CERTDIR: ""

  EWIM2-36:
    image: tpwls1355/module:latest
    container_name: EWIM2-36
    privileged: true
    volumes:
      - /nfs_shared:/var/lib/result
    ports:
      - "8083:8000"
    environment:
      DOCKER_TLS_CERTDIR: ""

  EWIM2-46:
    image: tpwls1355/module:latest
    container_name: EWIM2-46
    privileged: true
    volumes:
      - /nfs_shared:/var/lib/result
    ports:
      - "8084:8000"
```

```
    environment:
      DOCKER_TLS_CERTDIR: ""
```

3.4. Nginx 설정

- CertBot 발급

```
sudo apt update
sudo apt install certbot python3-certbot-nginx -y
sudo certbot --nginx -d semony-s109.site
```

- /etc/nginx/available-sites/default 설정 파일 수정

```
server {
    listen 80;
    server_name semony-s109.site;

    # HTTP에서 HTTPS로 리디렉션
    return 301 https://$host$request_uri;
}

server {
    listen 443 ssl;
    listen [::]:443 ssl;
    server_name semony-s109.site;

    ssl_certificate /etc/letsencrypt/live/semony-s109.site/fu
    ssl_certificate_key /etc/letsencrypt/live/semony-s109.sit
    include /etc/letsencrypt/options-ssl-nginx.conf;
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem;

    # HSTS 헤더 설정 (Mixed Content 방지)
    add_header Strict-Transport-Security "max-age=31536000; i

    # 프록시 설정
    location / {
        proxy_pass http://localhost:3000;
```

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

    location /api/ {
        proxy_pass http://localhost:8090;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forward
        proxy_set_header X-Forwarded-Proto $scheme;
    }
}
```

4. NFS 마운트 설정

❗ Module#1, Module#2에 각각 동일하게 설정합니다.

- 패키지 업데이트

```
sudo apt-get update
sudo apt-get upgrade
```

- NFS 서버 설치

```
sudo apt install -y nfs-kernel-server
```

- 공유 디렉토리 생성 및 설정

```
sudo mkdir -p /nfs_shared
sudo chown -R nobody:nogroup /nfs_shared
sudo chmod 777 /nfs_shared
```

- /etc/exports 설정

```
# /etc/exports: the access control list for filesystems which
#               to NFS clients.  See exports(5).
#
# Example for NFSv2 and NFSv3:
# /srv/homes       hostname1(rw,sync,no_subtree_check) hostna
#
# Example for NFSv4:
# /srv/nfs4        gss/krb5i(rw,sync,fsid=0,crossmnt,no_subtr
# /srv/nfs4/homes  gss/krb5i(rw,sync,no_subtree_check)
#

# 커버로스 설정을 위한 NFS4 루트 디렉토리 설정
/lib/data {NFS Client Private IP}(rw,sync,no_subtree_check,cr
/lib/data/module1 {NFS Client Private IP}(rw,sync,no_subtree_
```

- NFS 서버 시작 및 내보내기 활성화

```
sudo exportfs -a
sudo systemctl restart nfs-kernel-server
```

- Main Instance에서 NFS Client 설치

```
sudo apt update
sudo apt install -y nfs-common
```

5. 커버로스 설정

❗커버로스 환경에서는 FQDN을 사용합니다. /etc/hosts에 각 모듈 서버의 도메인 네임과 사설 IP 주소를 할당해주세요.

(Main, Module#1, Module#2)

- /etc/hosts

```
127.0.0.1 localhost
172.26.11.242 nfs-server.module1.com
172.26.8.150 nfs-server.module2.com
# The following lines are desirable for IPv6 capable hosts
::1 ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
ff02::3 ip6-allhosts
```

5.1. 커버로스 서버 설정 (Main LightSail Instance)

- 커버로스 인증 서버 설치

```
sudo apt update
sudo apt install krb5-kdc krb5-admin-server krb5-config
```

- 설치 시 Realm은 `SEMONY.COM`, 서버 호스트 이름은 `semony-s109.site`
- 미설정 시 /etc/krb5.conf 파일 다음과 같이 수정

```
[libdefaults]
    default_realm = SEMONY.COM
    dns_lookup_kdc = false
    dns_lookup_realm = false
    ticket_lifetime = 24h
    renew_lifetime = 7d
    forwardable = true

[realms]
    SEMONY.COM = {
        kdc = semony-s109.site
        admin_server = semony-s109.site
    }
```

```
[domain_realm]
    .semony.com = SEMONY.COM
    semony.com = SEMONY.COM
```

- 커버로스 데이터베이스 초기화

```
sudo krb5_newrealm
# 비밀번호: s109
```

- 커버로스 서버 시작

```
sudo systemctl enable krb5-kdc krb5-admin-server
sudo systemctl start krb5-kdc krb5-admin-server
```

- 커버로스 관리자 계정 생성

```
sudo kadmin.local -q "addprinc admin/admin"
# 비밀번호: s109
```

- NFS Client 계정 생성

```
sudo kadmin.local -q "addprinc integrated@SEMONY.COM"
# 비밀번호: s109
```

- NFS Server 계정 생성

```
sudo kadmin.local -q "addprinc -randkey nfs/nfs-server.module
sudo kadmin.local -q "ktadd -k /etc/krb5.keytab nfs/nfs-serve
sudo kadmin.local -q "addprinc -randkey nfs/nfs-server.module
sudo kadmin.local -q "ktadd -k /etc/krb5.keytab nfs/nfs-serve
```

- 생성된 keytab 파일을 NFS 서버로 복사 (사전에 ssh키를 등록해야 합니다.)

```
scp /etc/krb5.keytab ubuntu@nfs-server.module1.com:/etc/krb5.
scp /etc/krb5.keytab ubuntu@nfs-server.module2.com:/etc/krb5.
```

- NFS 클라이언트 티켓 발급

```
kinit integrated@SEMONY.COM
```

5.2. NFS 서버 커버로스 설정

- 커버로스 클라이언트 패키지 설치

```
sudo apt install krb5-user
```

- 설치 시 Realm은 `SEMONY.COM`, 서버 호스트 이름은 `semony-s109.site`
- 미설정 시 /etc/krb5.conf 파일 다음과 같이 수정

```
[libdefaults]
    default_realm = SEMONY.COM
    dns_lookup_kdc = false
    dns_lookup_realm = false
    ticket_lifetime = 24h
    renew_lifetime = 7d
    forwardable = true

[realms]
    SEMONY.COM = {
        kdc = semony-s109.site
        admin_server = semony-s109.site
    }

[domain_realm]
    .semony.com = SEMONY.COM
    semony.com = SEMONY.COM
```

5.3. 마운트 수행

- NFS 클라이언트 티켓 발급

```
kinit integrated@SEMONY.COM
```

- 마운트

```
sudo mount -t nfs4 -o sec=krb5 nfs-server.module1.com:/module
sudo mount -t nfs4 -o sec=krb5 nfs-server.module2.com:/module
```

❗ 커버로스 티켓 만료 시간은 기본 24h으로 설정되어 있습니다. 티켓이 만료되어도 연결은 유지되지만 만료된 티켓으로는 다시 마운트할 수 없습니다. `kinit` 명령으로 티켓을 재발급 받고 마운트해야 합니다.