

아르카나 포팅매뉴얼

목차

1. 프로젝트 개요
 2. 프로젝트 사용 도구
 3. 개발환경
 4. 외부 서비스
 5. 환경변수
 6. Nginx 설치 및 SSL 인증서 발급
 7. Jenkins 초기 설정 및 권한 부여
 8. Jenkins 설정
 9. Docker Compose 설치
 10. MySQL 및 Redis 설치
 11. 배포 전략
 12. SonarQube 설정
 13. API 문서화
 14. 추가 설치 및 설정 스크립트
 15. 기타 설정 및 명령어
-

1. 프로젝트 개요

- **프로젝트 이름:** 아르카나(Arcana)
- **설명:**
 - **장르:** 덱빌딩 로그라이크 게임
 - **게임 개요:**
 - 아르카나는 다양한 종족과의 상호작용과 전략적인 덱 구성을 통해 진행되는 덱 빌딩 로그라이크 게임입니다. 플레이어는 아르카나 행성에서 인류의 생존과 확장을 위해 카드를 모으고 전투를 치러나 가야 합니다.
- **주요 목표 및 문제 해결 방안:**

1. 덱 빌딩 및 전투 시스템 최적화

- 다양한 종족 카드를 수집하고 전략적으로 덱을 구성하여 전투를 진행합니다.

2. 로그라이크 기반의 무한한 재플레이성

- 무작위로 생성되는 맵과 이벤트를 통해 매번 새로운 게임 경험을 제공합니다.

3. AI 기반 적 행동 시스템

- Minimax 알고리즘을 활용하여 적의 행동 패턴을 최적화하고, 플레이어의 전략에 맞서는 AI 적을 구현합니다.

4. Redis 기반 캐싱 시스템

- 게임 데이터를 빠르게 접근 및 처리하여 실시간 플레이를 지원합니다.

5. Discord 및 커뮤니티 기능

- 커뮤니티 플랫폼과 연동하여 플레이어 간 소통을 강화합니다.

• 주요 기능:

- 절차적 생성 및 로그라이크 게임 진행
 - 덱 빌딩 및 전투 시스템
 - AI 기반 적 행동 시스템
 - Redis 기반 캐싱 시스템
 - Discord 및 커뮤니티 기능
-

2. 프로젝트 사용 도구

- 이슈 관리: Jira
 - 형상 관리: GitLab, Plastic SCM
 - 커뮤니케이션: Mattermost, Notion
 - 디자인: Figma
 - CI/CD: Jenkins, Docker, Docker Hub, Nginx
 - 정적 코드 분석 도구: SonarQube
 - 커뮤니티 연동: Discord
-

3. 개발환경

- Unity: 2022.3.50f1

- **프론트엔드:** React 18.3.1, Tailwind CSS 3.4.14, TypeScript 5.6.2
- **백엔드:** Java 17, Spring Boot 3.3.1
- **데이터베이스:** MySQL 8.0, Redis 7.0
- **인프라:** Ubuntu 20.04.6, Docker 27.3.1, Jenkins 2.482, Nginx 1.18.0
- **IDE:** IntelliJ IDEA, Visual Studio Code
- **빌드 도구:** Gradle
- **버전 관리:** Git

4. 외부 서비스

- **스토리지 서비스:** AWS S3

5. 환경변수

- `application.yml` 파일에 포함된 변수 및 설정:

```
spring:
  datasource:
    url: # 데이터베이스 URL
    username: # 데이터베이스 사용자명
    password: # 데이터베이스 비밀번호
    driver-class-name:
  jpa:
    hibernate:
      ddl-auto:
      show-sql:
    properties:
      hibernate:
        format_sql:
    database-platform:
  data:
    redis:
      host: # Redis 호스트
      port: # Redis 포트
  mail:
    host:
    port:
```

```

username: # 발신자 이메일 주소
password: # 발신자 이메일 앱 비밀번호
properties:
  mail:
    smtp:
      auth:
      starttls:
      enable:
jwt:
  secret: # JWT 시크릿 키
  accessTokenExpiration:
  refreshTokenExpiration:
cloud:
  aws:
    credentials:
      accessKey: # AWS 액세스 키
      secretKey: # AWS 시크릿 키
    region:
      static:
    s3:
      bucket:
logging:
  level:
    com.arcane.arcana:
    org.springframework.security:
    org.hibernate.SQL:

```

6. Nginx 설치 및 SSL 인증서 발급

- Nginx 설치

```

sudo apt-get update
sudo apt-get install nginx

```

- Let's Encrypt 설치 및 SSL 인증서 발급

```

sudo apt-get install letsencrypt
sudo systemctl stop nginx

```

```
sudo letsencrypt certonly --standalone -d k11d103.p.ssafy.io
sudo systemctl start nginx
```

- **Nginx 설정 파일 편집**

- **경로:** `/etc/nginx/sites-available/default`

```
# HTTP를 HTTPS로 리다이렉션하는 서버 블록
server {
    listen 80 default_server;
    listen [::]:80 default_server;
    server_name k11d103.p.ssafy.io;
    return 301 https://$host$request_uri;
}

# HTTPS 서버 블록
server {
    listen 443 ssl http2;
    listen [::]:443 ssl http2;
    server_name k11d103.p.ssafy.io;

    # SSL 설정
    ssl_certificate /etc/letsencrypt/live/k11d103.p.ssafy.io/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/k11d103.p.ssafy.io/privkey.pem;
    ssl_protocols TLSv1.2 TLSv1.3;
    ssl_ciphers HIGH:!aNULL:!MD5;

    # 기본 설정
    client_body_timeout 10s;
    client_header_timeout 10s;
    client_max_body_size 1m;

    # 백엔드 API 엔드포인트들
    location ~ ^/(user|game-data|token|support|subscribe|health|swagger-ui|v3/api-docs) {
        proxy_pass http://backend_servers;
    }
}
```

```

        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;

        # DDoS 방어 설정
        limit_req zone=ddos_limit burst=20 nodelay;
        limit_conn addr 10;
    }

    # Swagger UI의 정적 리소스를 위한 추가 설정
    location ~ ^/swagger-ui/.*\.(js|css|png|html)$ {
        proxy_pass http://backend_servers;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        expires 1d;
    }

    # 프론트엔드 요청 처리 (나머지 모든 요청)
    location / {
        proxy_pass http://frontend_servers;
        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;

        # WebSocket 지원을 위한 추가 설정 (필요한 경우)
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';
    }

    # 프론트엔드 정적 파일에 대한 캐싱 설정
    location ~* \.(jpg|jpeg|png|gif|ico|css|js|svg|woff|woff2|ttf|eot)$ {
        proxy_pass http://frontend_servers;

```

```

        expires 1d;
        access_log off;
        add_header Cache-Control "public";
    }
}

```

- **Nginx 업스트림 설정 파일**

- **경로:** `/etc/nginx/conf.d/upstream.conf`

```

upstream backend_servers {
    # server localhost:8081; # 블루 서버
    server localhost:8082; # 그린 서버
}

upstream frontend_servers {
    server localhost:3000;
}

```

- **Nginx 설정 테스트 및 재시작**

```

sudo nginx -t
sudo systemctl restart nginx

```

7. Jenkins 초기 설정 및 권한 부여

- **Jenkins 컨테이너 실행**

```

docker run -d -p 9090:8080 -p 50000:50000 \
    --name jenkins \
    -v /var/run/docker.sock:/var/run/docker.sock \
    -v /home/ubuntu/jenkins:/var/jenkins_home \
    jenkins/jenkins:jdk17

```

- **Jenkins 초기 비밀번호 확인**

```

sudo docker exec jenkins cat /var/jenkins_home/secrets/i
nitiaAdminPassword

```

- **Docker CLI 설치 (Jenkins 컨테이너 내)**

```
docker exec -it --user root jenkins bash
apt-get update
apt-get install -y docker.io
exit
docker restart jenkins
```

8. Jenkins 설정

- **플러그인 설치**

- **GitLab:** Version 1.9.5
- **Docker:** Version 1.7.0
- **SonarQube Scanner:** Version 2.17.2
- **Mattermost Notification:** Version 3.1.3
- **SSH Agent:** Version 376.v8933585c69d3
- **NodeJS:** Version 1.6.2

- **GitLab Access Token 발급**

- **토큰:** GitLab에서 개인 액세스 토큰을 생성하고 Jenkins Credentials에 저장합니다.

- **Jenkins 전역 설정**

- **GitLab Credentials:** Jenkins Credentials에 추가
- **SonarQube Server:** `http://k11d103.p.ssafy.io:9000`
- **SonarQube Token:** SonarQube에서 토큰을 생성하고 Jenkins Credentials에 저장합니다.

- **Jenkins 파이프라인 설정**

프론트엔드 파이프라인

```
pipeline {
    agent any

    environment {
```



```

    DOCKER_IMAGE = 'aldus0422/arcana:front'
    SSH_CREDENTIALS = 'ssh-key-id'
    SERVER_IP = 'k11d103.p.ssafy.io'
    SSH_USER = 'ubuntu'
}

stages {
    stage('Git Checkout') {
        steps {
            script {
                echo "=== Git 레포지토리 체크아웃 중 ==="

                checkout([$class: 'GitSCM', branches: [[name: '*/develop']], userRemoteConfigs: [[url: 'https://lab.ssafy.com/s11-final/S11P31D103.git', credentialsId: 'gitlab']]])

                sh 'ls -la'
            }
        }
    }

    stage('Install Dependencies') {
        steps {
            script {
                echo "=== 의존성 설치 중 ==="
                dir('frontend') {
                    nodejs(nodeJSInstallationName:
'NodeJS') {
                        sh 'npm install'
                    }
                }
            }
        }
    }

    stage('Build') {
        steps {
            script {

```

```

        echo "=== 빌드 중 ==="
        dir('frontend') {
            nodejs(nodeJSInstallationName:
'NodeJS') {
                sh 'npm run build'
            }
        }
    }
}

stage('Docker Build and Push') {
    steps {
        dir('frontend') {
            script {
                sh 'ls -la'
                echo "=== Docker 이미지 빌드 중 ==
="

                sh 'docker build --no-cache -t a
ldus0422/arcana:front .'

                echo "=== DockerHub로 이미지 푸시
중 ==="

                withCredentials([usernamePasswor
d(credentialsId: 'docker-credentials-id', passwordVariab
le: 'DOCKER_HUB_PASSWORD', usernameVariable: 'DOCKER_HUB
_USERNAME')])) {
                    sh 'echo $DOCKER_HUB_PASSWOR
D | docker login -u $DOCKER_HUB_USERNAME --password-stdi
n'

                    sh 'docker push aldus0422/ar
cana:front '
                }
            }
        }
    }
}

```

```

stage('Deploy to Server') {
    steps {
        script {
            echo "=== Docker 컨테이너 서버에 배포 중
===\"

            sshagent([SSH_CREDENTIALS]) {
                sh \"\"\"
                    ssh -o StrictHostKeyChecking
=no ${SSH_USER}@${SERVER_IP} \"
                                docker pull ${DOCKER_IMA
GE} &&
                                docker stop arcana-front
end || true &&
                                docker rm arcana-fronten
d || true &&
                                docker run -d -p 3000:80
--name arcana-frontend ${DOCKER_IMAGE}
                                \"
                                \"\"\"
                }
            }
        }
    }

    post {
        success {
            script {
                echo \"=== 빌드 및 배포 성공 ===\"
            }
        }
        failure {
            script {
                echo \"=== 빌드 또는 배포 실패 ===\"
            }
        }
    }
}
}

```

백엔드 파이프라인

```
pipeline {
    agent any

    environment {
        DOCKER_IMAGE = 'aldus0422/arcana:latest'
        SSH_CREDENTIALS = 'ssh-key-id'
        SERVER_IP = 'k11d103.p.ssafy.io'
        SSH_USER = 'ubuntu'
    }

    stages {
        stage('Git Checkout') {
            steps {
                script {
                    echo "=== Git 레포지토리 체크아웃 중 ==="

                    checkout([$class: 'GitSCM', branches: [[name: '*/develop']], userRemoteConfigs: [[url: 'https://lab.ssafy.com/s11-final/S11P31D103.git', credentialsId: 'gitlab']]])

                    sh 'ls -la'
                }
            }
        }

        stage('Set Permissions') {
            steps {
                echo "=== 파일 권한 설정 중 ==="
                sh 'chmod -R 777 backend'
            }
        }

        stage('Copy Secret Files') {
            steps {
                script {
                    echo "=== 시크릿 파일을 복사 중 ==="
```

```

        withCredentials([file(credentialsId:
'application.yml', variable: 'APP_YML')])) {
            sh 'cp $APP_YML backend/src/main/resources/application.yml'
        }
    }
}

stage('Build with Gradle') {
    steps {
        dir('backend') {
            script {
                echo "=== Gradle 빌드 ==="
                sh './gradlew clean build -x test --build-cache'
            }
        }
    }
}

stage('Docker Build and Push') {
    steps {
        dir('backend') {
            script {
                echo "=== Docker 이미지 빌드 중 ==="

                sh 'docker build -t aldus0422/arcanalatest .'

                echo "=== DockerHub로 이미지 푸시 중 ==="

                withCredentials([usernamePassword(credentialsId: 'docker-credentials-id', passwordVariable: 'DOCKER_HUB_PASSWORD', usernameVariable: 'DOCKER_HUB_USERNAME')])) {
                    sh 'echo $DOCKER_HUB_PASSWORD | docker login -u $DOCKER_HUB_USERNAME --password-stdin'
                }
            }
        }
    }
}

```

```

n'
                                sh 'docker push aldus0422/ar
cana:latest'
                                }
                                }
                                }
                                }
                                }

stage('Deploy to Inactive Environment') {
    steps {
        script {
            echo "=== 비활성화된 환경에 배포 중 ==="
            sshagent([SSH_CREDENTIALS]) {
                sh '''
                    ACTIVE_PORT=$(ssh -o StrictH
ostKeyChecking=no ${SSH_USER}@${SERVER_IP} "sudo grep -E
'^\\s*server localhost:808[12];' /etc/nginx/conf.d/upstr
eam.conf | grep -o '808[12]' | head -n1")
                    echo 현재 활성 환경 포트: ${ACT
IVE_PORT}

                    if [ "${ACTIVE_PORT}" = "808
1" ]; then

                        INACTIVE_SERVICE="spring
boot-green"

                        IMAGE_TAG="green"
                        INACTIVE_PORT="8082"
                        echo "비활성 환경: 그린 (포
트 8082)"

                        elif [ "${ACTIVE_PORT}" = "8
082" ]; then

                            INACTIVE_SERVICE="spring
boot-blue"

                            IMAGE_TAG="blue"
                            INACTIVE_PORT="8081"
                            echo "비활성 환경: 블루 (포
트 8081)"

```

```

else
    echo "활성화된 포트를 찾을 수 없습니다. 기본값으로 블루(8081)를 활성화합니다."
    INACTIVE_SERVICE="spring boot-blue"

    IMAGE_TAG="blue"
    INACTIVE_PORT="8081"
fi

echo "활성 포트: ${ACTIVE_PORT}, 비활성 포트: ${INACTIVE_PORT}"

ssh -o StrictHostKeyChecking=no ${SSH_USER}@${SERVER_IP} "docker-compose -f /home/ubuntu/arcana/backend/docker-compose.yml stop ${INACTIVE_SERVICE} && docker-compose -f /home/ubuntu/arcana/backend/docker-compose.yml rm -f ${INACTIVE_SERVICE}"

ssh -o StrictHostKeyChecking=no ${SSH_USER}@${SERVER_IP} "docker pull aldus0422/arcana:latest && docker tag aldus0422/arcana:latest aldus0422/arcana:${IMAGE_TAG} && cd /home/ubuntu/arcana/backend && docker-compose -f /home/ubuntu/arcana/backend/docker-compose.yml up -d --no-deps ${INACTIVE_SERVICE}"

ssh -o StrictHostKeyChecking=no ${SSH_USER}@${SERVER_IP} "docker-compose -f /home/ubuntu/arcana/backend/docker-compose.yml ps ${INACTIVE_SERVICE}"

...
}
}
}

stage('Wait for Stabilization') {
    steps {
        echo "=== 새로운 컨테이너 안정화 대기 중 ==="
        sleep time: 10, unit: 'SECONDS'
    }
}

```

```

    }

    stage('Health Check') {
        steps {
            script {
                echo "=== 헬스 체크 수행 중 ==="
                sshagent([SSH_CREDENTIALS]) {
                    sh '''
                        set +e
                        MAX_RETRIES=12
                        RETRY_INTERVAL=5

                        ACTIVE_PORT=$(ssh -o StrictHostKeyChecking=no ${SSH_USER}@${SERVER_IP} "sudo grep -E '^\\s*server localhost:808[12];' /etc/nginx/conf.d/upstream.conf | grep -o '808[12]' | head -n1")
                        if [ "${ACTIVE_PORT}" = "8081" ]; then

                            INACTIVE_PORT="8082"
                        else
                            INACTIVE_PORT="8081"
                        fi

                        HEALTH_CHECK_URL="http://${SERVER_IP}:${INACTIVE_PORT}/health"
                        echo "헬스 체크 URL: ${HEALTH_CHECK_URL}"

                        RETRIES=0

                        while [ $RETRIES -lt $MAX_RETRIES ]; do

                            STATUS=$(curl -s -o /dev/null -w '%{http_code}' ${HEALTH_CHECK_URL})
                            if [ "$STATUS" -eq 200 ] ; then

                                echo "헬스 체크 성공!"
                                exit 0

```



```

else
    echo "헬스 체크 실패,
    ${RETRY_INTERVAL}초 후 재시도... (${RETRIES}/${MAX_RETRIES})"

    RETRIES=$((RETRIES+
1))

    sleep $RETRY_INTERVAL

fi
done
echo "헬스 체크 실패: 최대 재시도
횟수 초과"

exit 1
'''
}
}
}

stage('Switch Nginx to New Environment') {
    steps {
        script {
            echo "=== Nginx를 새로운 환경으로 전환 중
===

            sshagent([SSH_CREDENTIALS]) {
                sh '''

                    ssh -T -o StrictHostKeyCheck
ing=no ${SSH_USER}@${SERVER_IP} << EOF
                    echo "현재 Nginx 설정:"
                    sudo cat /etc/nginx/con
f.d/upstream.conf

                    echo "스크립트 실행 중..."
                    sudo bash /usr/local/bi
n/switch_nginx.sh

                    echo "스크립트 실행 완료"
                    echo "변경 후 Nginx 설정:"
                    sudo cat /etc/nginx/con
f.d/upstream.conf

```

```

                                echo "Nginx 상태 확인:"
                                sudo systemctl --no-page
r status nginx
                                exit
                                EOF
                                ...
                                }
                                }
                                }
                                }

stage('SonarQube Analysis') {
    steps {
        dir('backend') {
            script {
                echo "=== SonarQube 분석 중 ==="
                withSonarQubeEnv('arcana') {
                    withCredentials([string(credentialsId: 'sonarQubeToken', variable: 'SONAR_TOKEN')])
                {
                    sh './gradlew sonarqube
-Dsonar.projectKey=arcana -Dsonar.projectName=arcana -Dsonar.host.url=http://k11d103.p.ssafy.io:9000 -Dsonar.login=$SONAR_TOKEN'
                }
            }
        }
    }
}

stage('Notify Mattermost') {
    steps {
        script {
            echo "=== Mattermost에 알림 전송 중 ==="

            mattermostSend color: '#32a852', message: """"빌드 성공 (${env.JOB_NAME}) #(${env.BUILD_NUMBE

```

```

R})) (<${env.BUILD_URL}|Open>)
See the (<${env.BUILD_URL}console|console>)"
    }
  }
}

post {
  always {
    script {
      echo "=== Cleanup 또는 에러 처리 중 ==="
    }
  }

  failure {
    script {
      echo "=== 빌드 실패, 알림 전송 ==="
      mattermostSend color: '#ff0000', message: ""빌드 실패 (${env.JOB_NAME}) #(${env.BUILD_NUMBER}) (<${env.BUILD_URL}|Open>) See the (<${env.BUILD_URL}console|console>)"
    }
  }
}
}

```

9. Docker Compose 설치

```

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

```

10. MySQL 및 Redis 설치

Redis 설치 스크립트

```
#!/bin/bash

# Redis 컨테이너 실행
docker run -d \
  -p 6379:6379 \
  -v /var/lib/redis-data:/data \
  --name redis \
  redis:7.0
```

MySQL 설치 스크립트

```
#!/bin/bash

# MySQL 컨테이너 실행
docker run -d \
  -p 3306:3306 \
  -e MYSQL_ROOT_PASSWORD=yourpassword \
  -v /var/lib/mysql-data:/var/lib/mysql \
  --name mysql \
  mysql:8.0
```

Docker 네트워크 연결

```
docker network connect backend_arcana-network redis
docker network connect backend_arcana-network mysql
```

11. 배포 전략

- **블루-그린 배포(Blue-Green Deployment)**
 - **설명:** 블루-그린 배포 방식을 사용하여 무중단 배포를 구현했습니다.
- **Nginx 업스트림 설정 스크립트**
 - **경로:** `/usr/local/bin/switch_nginx.sh`

```
#!/bin/bash
UPSTREAM_CONF="/etc/nginx/conf.d/upstream.conf"
```

```

BACKUP_CONF="/etc/nginx/conf.d/upstream.conf.bak"

# 현재 설정 백업
sudo cp $UPSTREAM_CONF $BACKUP_CONF

# 현재 활성화된 포트 확인
ACTIVE_PORT=$(grep -E '^s*server localhost:808[12];'
$UPSTREAM_CONF | grep -o '808[12]' | head -n1)

# 새로운 설정 생성 및 전환
if [ "$ACTIVE_PORT" == "8081" ]; then
    NEW_CONFIG="upstream backend_servers {
# server localhost:8081; # 블루 서버 비활성화
server localhost:8082; # 그린 서버 활성화
}"
elif [ "$ACTIVE_PORT" == "8082" ]; then
    NEW_CONFIG="upstream backend_servers {
server localhost:8081; # 블루 서버 활성화
# server localhost:8082; # 그린 서버 비활성화
}"
else
    NEW_CONFIG="upstream backend_servers {
server localhost:8081; # 블루 서버 활성화
# server localhost:8082; # 그린 서버 비활성화
}"
fi

# 새로운 설정을 파일에 쓰기
echo "$NEW_CONFIG" | sudo tee $UPSTREAM_CONF

# Nginx 설정 테스트 및 적용
if sudo nginx -t; then
    sudo systemctl reload nginx
else
    sudo cp $BACKUP_CONF $UPSTREAM_CONF
    sudo systemctl reload nginx
    exit 1
fi

```

12. SonarQube 설정

- SonarQube 컨테이너 실행

```
docker run -d --name sonarqube -e SONAR_ES_BOOTSTRAP_CHECKS_DISABLE=true -p 9000:9000 sonarqube:latest
```

- 토큰 생성 및 Jenkins 연동

- **SonarQube 토큰**: SonarQube에서 토큰을 생성하고 Jenkins Credentials에 `sonarQubeToken` 으로 저장합니다.

13. API 문서화

- Swagger 설정

```
// SwaggerConfig.java

import io.swagger.v3.oas.annotations.OpenAPIDefinition;
import io.swagger.v3.oas.annotations.servers.Server;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

import io.swagger.v3.oas.models.Components;
import io.swagger.v3.oas.models.OpenAPI;
import io.swagger.v3.oas.models.info.Info;
import io.swagger.v3.oas.models.security.SecurityRequirement;
import io.swagger.v3.oas.models.security.SecurityScheme;

@Configuration
@OpenAPIDefinition(
    servers = {
        @Server(url = "https://k11d103.p.ssafy.io", description = "Arcana HTTPS 서버"),
        @Server(url = "http://k11d103.p.ssafy.io", description = "Arcana HTTP 서버"),
        @Server(url = "http://localhost:8080", description = "Local 서버")
    }
)
```

```

    }
)
public class SwaggerConfig {

    @Bean
    public OpenAPI customOpenAPI() {
        return new OpenAPI()
            .components(new Components()
                .addSecuritySchemes("bearer-key",
                    new SecurityScheme()
                        .type(SecurityScheme.Type.HTTP)
                        .scheme("bearer")
                        .bearerFormat("JWT")))
            .info(new Info()
                .title("Arcana API Documentation")
                .version("1.0")
                .description("Arcana 프로젝트의 API 명세서입니다."))
            .addSecurityItem(new SecurityRequirement().addList("bearer-key"));
    }
}

```

- **Swagger UI 접근**

- <https://k11d103.p.ssafy.io/swagger-ui/index.html> 에서 API 문서 확인 가능

14. 추가 설치 및 설정 스크립트

Fail2Ban 설치 및 설정

```

sudo apt-get update
sudo apt-get install fail2ban -y

# 설정 파일 생성 및 편집
sudo nano /etc/fail2ban/jail.local

# 내용 추가
[nginx-req-limit]

```

```

enabled = true
filter = nginx-req-limit
action = iptables-multiport[name=ReqLimit, port="http,https", protocol=tcp]
logpath = /var/log/nginx/error.log
findtime = 600
bantime = 7200
maxretry = 10

# 필터 설정 파일 생성
sudo nano /etc/fail2ban/filter.d/nginx-req-limit.conf

# 내용 추가
[Definition]
failregex = limiting requests, excess:.* by zone

# Fail2Ban 재시작
sudo systemctl restart fail2ban

```

ModSecurity 웹 방화벽 설치 및 구성

```

sudo apt-get update
sudo apt-get install -y libnginx-mod-security

# ModSecurity 설정 파일 편집
sudo nano /etc/nginx/modsec/modsecurity.conf

# 설정 변경
SecRuleEngine On

# Nginx 설정 파일에 모듈 로드 추가
sudo nano /etc/nginx/nginx.conf

# 내용 추가
load_module modules/ngx_http_modsecurity_module.so;

# Nginx 재시작

```



```
sudo nginx -t
sudo systemctl restart nginx
```

15. 기타 설정 및 명령어

- SSH 설정

```
mkdir -p ~/.ssh
cd ~/.ssh
cp [현재 pem 파일의 위치] ~/.ssh/K11D103T.pem

# config 파일 생성 및 편집
vi config

# 내용 추가
Host ssafy
    HostName k11D103.p.ssafy.io
    User ubuntu
    IdentityFile ~/.ssh/K11D103T.pem

# 접속
ssh ssafy
```

- Dockerfile 작성

프론트엔드 Dockerfile

```
# frontend/Dockerfile

FROM nginx:stable-alpine

# 빌드 결과물을 Nginx 정적 파일 디렉토리로 복사
COPY dist /usr/share/nginx/html

# Nginx 포트 개방
EXPOSE 80
```

```
# Nginx 실행
CMD ["nginx", "-g", "daemon off;"]
```

백엔드 Dockerfile

```
# backend/Dockerfile

FROM openjdk:17-jdk-slim

# 작업 디렉토리 설정
WORKDIR /app

# 타임존 설정
RUN ln -sf /usr/share/zoneinfo/Asia/Seoul /etc/localtime

# 애플리케이션 JAR 파일 복사
COPY build/libs/arcana-0.0.1-SNAPSHOT.jar app.jar

# 애플리케이션 실행
ENTRYPOINT ["java", "-jar", "app.jar"]
```

- **Docker Compose 파일**

- **경로:** `backend/docker-compose.yml`

```
version: '3.8'
services:
  springboot-blue:
    image: aldus0422/arcana:blue
    container_name: springboot-blue
    ports:
      - "8081:8080"
    environment:
      - REDIS_HOST=redis
      - REDIS_PORT=6379
      - TZ=Asia/Seoul
    networks:
      - arcana-network
```

```
restart: unless-stopped

springboot-green:
  image: aldus0422/arcana:green
  container_name: springboot-green
  ports:
    - "8082:8080"
  environment:
    - REDIS_HOST=redis
    - REDIS_PORT=6379
    - TZ=Asia/Seoul
  networks:
    - arcana-network
  restart: unless-stopped

redis:
  image: redis:7.0
  container_name: redis
  ports:
    - "6379:6379"
  networks:
    - arcana-network
  restart: unless-stopped

networks:
  arcana-network:
    driver: bridge
```