Porting Manual

Index

- Nginx

8. **Files ignored**

9. **etc) Settings or Tips**
    - How to apply temporary SSL to React, Spring Boot project

---

1. Stacks

1.1. Issue Management

1.2. SCM

1.3. Communities

1.4. Development Environment

1. **OS**

2. **IDE**

- IntelliJ IDEA 2022.1.4 (Ultimate Edition)

- Visual Studio Code 1.70.1

- JetBrains PyCharm Community Edition 2018.2.3

3. **Database**

- MongoDBCompass 1.33.1

- MySQL WorkBench 8.0

4. **Server**

Ubuntu 20.04 LTS

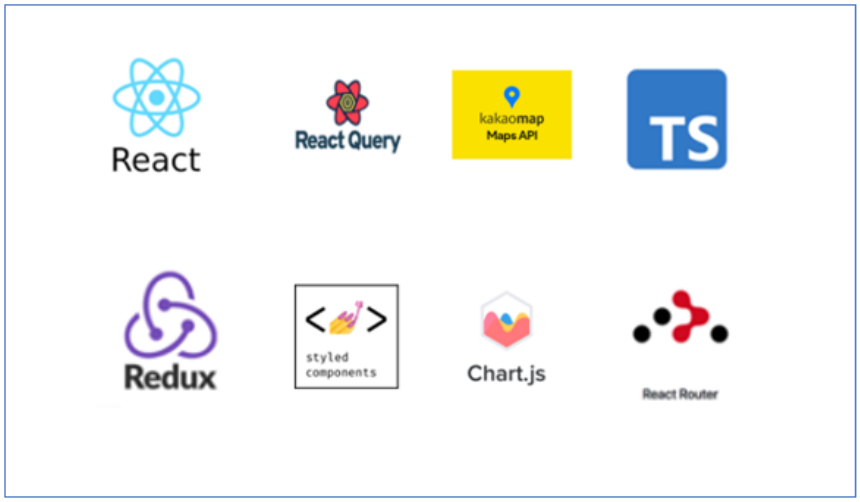
1.5. Details

1.5.1. Back-End

1. Java (Zulu 11.0.17-win64)
2. Spring Boot Gradle 7.5

3. Lombok 1.18.24
4. Swagger 3.0.0
5. JPA
6. JWT
7. Redis 3.2.100
8. Python 3.9.12
9. Flask 1.1.2
10. scikits learn 1.0.2
11. Pandas 1.4.2
12. Numpy 1.21.5
13. Kafka

1.5.2. Front-End



1. React 18.0.2
2. Typescript 4.8.4
3. React-Query 4.13.0
4. Styled-Components 5.3.6
5. Kakao Map API
6. axios 1.1.3
7. chart.js 3.9.1
8. d3.js 7.6.1
9. React-Router-Dom 6.4.2

1.5.3. etc

1. AWS EC2
2. Docker (20.10.17)
3. Nginx (1.18.0)
4. certBot
5. Jenkins

2. Build & Distribute

2.1. GateWay

- Spring Boot Version : 2.7.5
- Java Version : 11
- build.gradle

```
plugins {
  id 'org.springframework.boot' version '2.7.5'
  id 'io.spring.dependency-management' version '1.0.15.RELEASE'
  id 'java'
}

group = 'com.example'
version = '0.0.1-SNAPSHOT'
sourceCompatibility = '11'

repositories {
  mavenCentral()
}
ext {
  set('springCloudVersion', "2021.0.4")
}
dependencies {
  implementation 'org.springframework.boot:spring-boot-starter'
  testImplementation 'org.springframework.boot:spring-boot-starter-test'

  compileOnly 'org.projectlombok:lombok'
  developmentOnly 'org.springframework.boot:spring-boot-devtools'
  annotationProcessor 'org.projectlombok:lombok'

  // https://mvnrepository.com/artifact/org.springframework.cloud/spring-cloud-starter-gateway
  implementation group: 'org.springframework.cloud', name: 'spring-cloud-starter-gateway', version: '3.1.4'
  // https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-starter-actuator
  implementation group: 'org.springframework.boot', name: 'spring-boot-starter-actuator', version: '2.7.4'

  implementation 'org.springframework.boot:spring-boot-starter-web'
  implementation 'org.springframework.cloud:spring-cloud-starter-netflix-eureka-client'
  //swagger openapi
  implementation 'org.springdoc:springdoc-openapi-webflux-ui:1.6.12'
}
dependencyManagement {
  imports {
    mavenBom "org.springframework.cloud:spring-cloud-dependencies:${springCloudVersion}"
  }
```

```
}
tasks.named('test') {
  useJUnitPlatform()
}
```

- application.yml

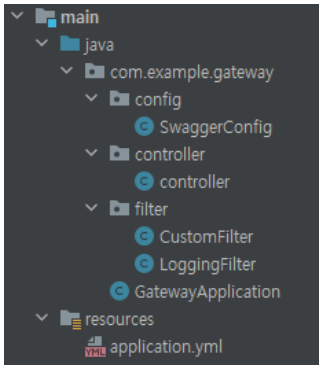
```
server:
  port: //server port

management:
  endpoints:
    web:
      exposure:
        include:
          - "gateway"
  endpoint:
    gateway:
      enabled: true

spring:
  application:
    name: apigateway-service
  cloud:
    gateway:
      routes:
        - id: openapi
          uri: lb://${spring.application.name}
          predicates:
            - Path=/v3/api-docs/**
          filters:
            - RewritePath=/v3/api-docs/(?<path>.*), /$\{path}/v3/api-docs
        - id: backend/anlz-service
          uri: lb://BACKEND-ANLZ
          predicates:
            - Path=/backend/anlz/**, /anlz/**
          filters:
            - RewritePath=/backend/anlz/(?<path>.*), /$\{path}
            - CustomFilter
            - name: LoggingFilter
              args:
                baseMessage: Spring Cloud Gateway Logging Filter
                preLogger: true
                postLogger: true
      default-filters:
        - DedupeResponseHeader=Access-Control-Allow-Origin Access-Control-Allow-Credentials
      globalcors:
        cors-configurations:
          '[/**]':
            allowedOrigins: '*'
            allow-credentials: false
            allowedHeaders: '*'
            allowedMethods:
              - PUT
              - GET
              - POST
              - DELETE
              - OPTIONS
  main:
    web-application-type: reactive

eureka:
  client:
    fetch-registry: true
    register-with-eureka: true
    service-url:
      defaultZone: http://{server domain:eureka port num}/eureka
  instance:
    instance-id: ${spring.application.name}:${spring.application.instance_id:${random.value}}
```

- Structure



- important point
  - Application.java
    - ADD @`EnableDiscoveryClient` Annotataion

```java
@SpringBootApplication
@EnableDiscoveryClient
public class GatewayApplication {

    public static void main(String[] args) { SpringApplication.run(GatewayApplication.class, args); }

}
```

  - SwaggerConfig.java
    - ADD Swagger config for Service Swagger

```java
@Configuration
public class SwaggerConfig {

    @Bean
    public CommandLineRunner openApiGroups(
            RouteDefinitionLocator locator,
            SwaggerUiConfigParameters swaggerUiParameters) {
        return args -> locator
                .getRouteDefinitions().collectList().block() List<RouteDefinition>
                .stream() Stream<RouteDefinition>
                .map(RouteDefinition::getId) Stream<String>
                .filter(id -> id.matches( regex: ".*-service"))
                .map(id -> id.replace( target: "-service",  replacement: ""))
                .forEach(swaggerUiParameters::addGroup);
    }
}
```

  - Filter
    - CustomFilter

```java
@Component
@Slf4j
public class CustomFilter extends AbstractGatewayFilterFactory<CustomFilter.Config> {
    public CustomFilter() { super(Config.class); }

    @Override
    public GatewayFilter apply(Config config) {
        // Custom Pre Filter
        return ((exchange, chain) -> {
            ServerHttpRequest request = exchange.getRequest(); // pre filter
            ServerHttpResponse response = exchange.getResponse(); // post filter
```

- LoggingFilter

```java
@Component
@Slf4j
public class LoggingFilter extends AbstractGatewayFilterFactory<LoggingFilter.Config> {
    public LoggingFilter() { super(Config.class); }

    @Override
    public GatewayFilter apply(Config config) {

        GatewayFilter filter = new OrderedGatewayFilter((exchange, chain) -> {
            ServerHttpRequest request = exchange.getRequest();
            ServerHttpResponse response = exchange.getResponse();
```

2.2. Eureka

- Spring Boot Version : 2.7.5
- Java Version : 11
- build.gradle

```
plugins {
  id 'org.springframework.boot' version '2.7.5'
  id 'io.spring.dependency-management' version '1.0.15.RELEASE'
  id 'java'
}

group = 'com.example'
version = '0.0.1-SNAPSHOT'
sourceCompatibility = '11'

repositories {
  mavenCentral()
}

ext {
  set('springCloudVersion', "2021.0.4")
}

dependencies {
  implementation 'org.springframework.cloud:spring-cloud-starter-netflix-eureka-server'
  testImplementation 'org.springframework.boot:spring-boot-starter-test'
}

dependencyManagement {
  imports {
    mavenBom "org.springframework.cloud:spring-cloud-dependencies:${springCloudVersion}"
  }
}

tasks.named('test') {
  useJUnitPlatform()
}
```

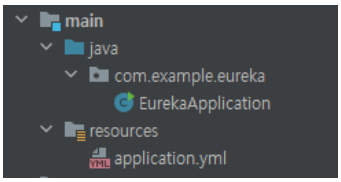
- application.yml

```
server:
  port: //eureka server port

spring:
  application:
    name: eurekaspring

eureka:
  client:
    register-with-eureka: false
    fetch-registry: false
```

- structure



2.3. Spring Boot(User)

2.4. Kafka

1. 설치(With Docker)
   a. zookeeper 설치
      - image pull

        ```
        docker pull wurstmeister/zookeeper
        ```

      - image start

        ```
        docker start --name zookeeper --network host -d -p 2181:2181 wurstmeister/zookeeper
        ```

   b. kafka 설치
      - image pull

        ```
        docker pull wurstmeister/kafka
        ```

      - image start

        ```
        docker start --name kafka --network host -d -p 9092:9092 wurstmeister/kafka
        ```

2. 사용 토픽 이름
   a. TOPIC_USER
   b. TOPIC_SIMUL
3. How To Use On Spring Boot?
   a. build.gradle

      ```
      // for apache kafka
      implementation 'org.springframework.kafka:spring-kafka'
      ```

   b. application.properties

```
### kafka
spring.kafka.bootstrap-servers={server domain}:9092
spring.kafka.consumer.group-id= testgroup
spring.kafka.consumer.auto-offset-reset= earliest
spring.kafka.consumer.key-deserializer= org.apache.kafka.common.serialization.StringDeserializer
spring.kafka.consumer.value-deserializer= org.apache.kafka.common.serialization.StringDeserializer
spring.kafka.producer.key-serializer= org.apache.kafka.common.serialization.StringSerializer
spring.kafka.producer.value-serializer= org.apache.kafka.common.serialization.StringSerializer
```

c. java file

i. KafkaController

```java
@RequiredArgsConstructor
@RestController
@RequestMapping("/user/kafka")
public class KafkaController {

    @Autowired
    private KafkaProducerService producer = new KafkaProducerServiceImpl();


    @PostMapping(value = "/message")
    public String sendMessage(@RequestParam("message") String message) {
        this.producer.sendMessage(message);
        return "success";
    }
}
```

ii. KafkaProducer

```java
@Service
@RequiredArgsConstructor
public class KafkaProducerServiceImpl implements KafkaProducerService{

    private String topicName = "TOPICNAME";

    @Autowired
    private KafkaTemplate<String, String> kafkaTemplate;

    @Override
    public void sendMessage(String message) {
        System.out.printf("Produce message : %s%n", message);
        this.kafkaTemplate.send(topicName, message);

    }
}
```

iii. KafkaConsumer

```java
@RequiredArgsConstructor
@Service
public class KafkaConsumerServiceImpl implements KafkaConsumerService{

    private final static String topicName = "TOPICNAME";

    @Autowired
    UserService userService;

    @KafkaListener(topics = topicName, groupId = ConsumerConfig.GROUP_ID_CONFIG)
    @Override
    public void consumer(String message) {
        System.out.printf("Consumed message : %s%n", message);
```

2.5. Nginx

- k7e205.p.ssafy.io Nginx config file

```
server {

        # SSL configuration
        #
        # listen 443 ssl default_server;
        # listen [::]:443 ssl default_server;
        #
        # Note: You should disable gzip for SSL traffic.
        # See: https://bugs.debian.org/773332
        #
        # Read up on ssl_ciphers to ensure a secure configuration.
        # See: https://bugs.debian.org/765782
        #
        # Self signed certs generated by the ssl-cert package
        # Don't use them in a production server!
        #
        # include snippets/snakeoil.conf;

        #root /var/www/html;
        root /jenkins/workspace/gitlab_frontend_react/frontend/build;

        # Add index.php to the list if you are using PHP
        index index.html index.htm index.nginx-debian.html;
        server_name k7e205.p.ssafy.io; # managed by Certbot


        location  / {
                # First attempt to serve request as file, then
                # as directory, then fall back to displaying a 404.
                try_files $uri $uri/ /index.html =404;
        }
        #location /user {
        #       return 301  http://localhost:8081/user$request_uri;
        #}

        location ~ /anlz {
                proxy_pass http://127.0.0.1:8081$request_uri;
        }

        # pass PHP scripts to FastCGI server
        #
        #location ~ \.php$ {
        #       include snippets/fastcgi-php.conf;
        #
        #       # With php-fpm (or other unix sockets):
        #       fastcgi_pass unix:/var/run/php/php7.4-fpm.sock;
        #       # With php-cgi (or other tcp sockets):
        #       fastcgi_pass 127.0.0.1:9000;
        #}

        # deny access to .htaccess files, if Apache's document root
        # concurs with nginx's one
        #
        #location ~ /\.ht {
        listen [::]:443 ssl ipv6only=on; # managed by Certbot
        listen 443 ssl; # managed by Certbot
        ssl_certificate /etc/letsencrypt/live/k7e205.p.ssafy.io/fullchain.pem; # managed by Certbot
        ssl_certificate_key /etc/letsencrypt/live/k7e205.p.ssafy.io/privkey.pem; # managed by Certbot
        include /etc/letsencrypt/options-ssl-nginx.conf; # managed by Certbot
```

```
        ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem; # managed by Certbot

}
server {
    if ($host = k7e205.p.ssafy.io) {
        return 301 https://$host$request_uri;
    } # managed by Certbot


        listen 80 ;
        listen [::]:80 ;
    server_name k7e205.p.ssafy.io;
    return 404; # managed by Certbot


}
```

- k7e2051.p.ssafy.io Nginx config file

```
server {

        # SSL configuration
        #
        # listen 443 ssl default_server;
        # listen [::]:443 ssl default_server;
        #
        # Note: You should disable gzip for SSL traffic.
        # See: https://bugs.debian.org/773332
        #
        # Read up on ssl_ciphers to ensure a secure configuration.
        # See: https://bugs.debian.org/765782
        #
        # Self signed certs generated by the ssl-cert package
        # Don't use them in a production server!
        #
        # include snippets/snakeoil.conf;

        root /var/www/html;

        # Add index.php to the list if you are using PHP
        index index.html index.htm index.nginx-debian.html;
    server_name k7e2051.p.ssafy.io; # managed by Certbot


        location / {
                # First attempt to serve request as file, then
                # as directory, then fall back to displaying a 404.
                try_files $uri $uri/ =404;
        }

        location ~ /user {
        #       return 301 http://localhost:8081$request_uri;
                proxy_pass http://127.0.0.1:8081$request_uri;
        }

        location ~ /simul {
                proxy_pass http://127.0.0.1:8081$request_uri;
        }
        #location /swagger{
        #       proxy_pass http://127.0.0.1:8081/swagger-ui.html;
        #}

        # pass PHP scripts to FastCGI server
        #
        #location ~ \.php$ {
        #       include snippets/fastcgi-php.conf;
        #
        #       # With php-fpm (or other unix sockets):
        #       fastcgi_pass unix:/var/run/php/php7.4-fpm.sock;
        #       # With php-cgi (or other tcp sockets):
        #       fastcgi_pass 127.0.0.1:9000;
        #}

        # deny access to .htaccess files, if Apache's document root
        # concurs with nginx's one
        #
        #location ~ /\.ht {
        #       deny all;
        #}


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

```
    listen 443 ssl; # managed by Certbot
    ssl_certificate /etc/letsencrypt/live/k7e2051.p.ssafy.io/fullchain.pem; # managed by Certbot
    ssl_certificate_key /etc/letsencrypt/live/k7e2051.p.ssafy.io/privkey.pem; # managed by Certbot
    include /etc/letsencrypt/options-ssl-nginx.conf; # managed by Certbot
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem; # managed by Certbot

}
server {
    if ($host = k7e2051.p.ssafy.io) {
        return 301 https://$host$request_uri;
    } # managed by Certbot


        listen 80 ;
        listen [::]:80 ;
    server_name k7e2051.p.ssafy.io;
    return 404; # managed by Certbot


}
```

2.6. JenKins

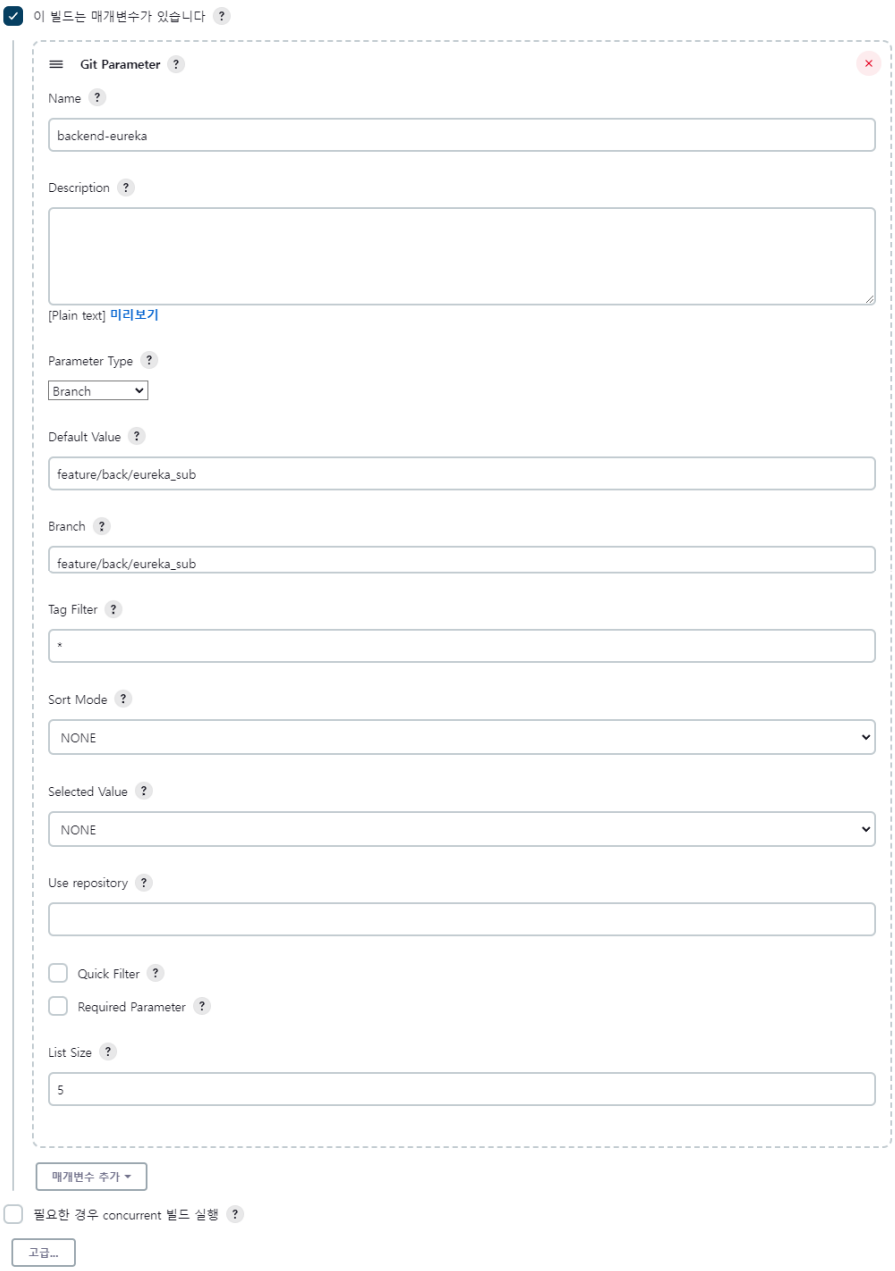
1. 설치(With Docker)
   - image pull

     ```
     docker pull jenkins/jenkins:lts
     ```

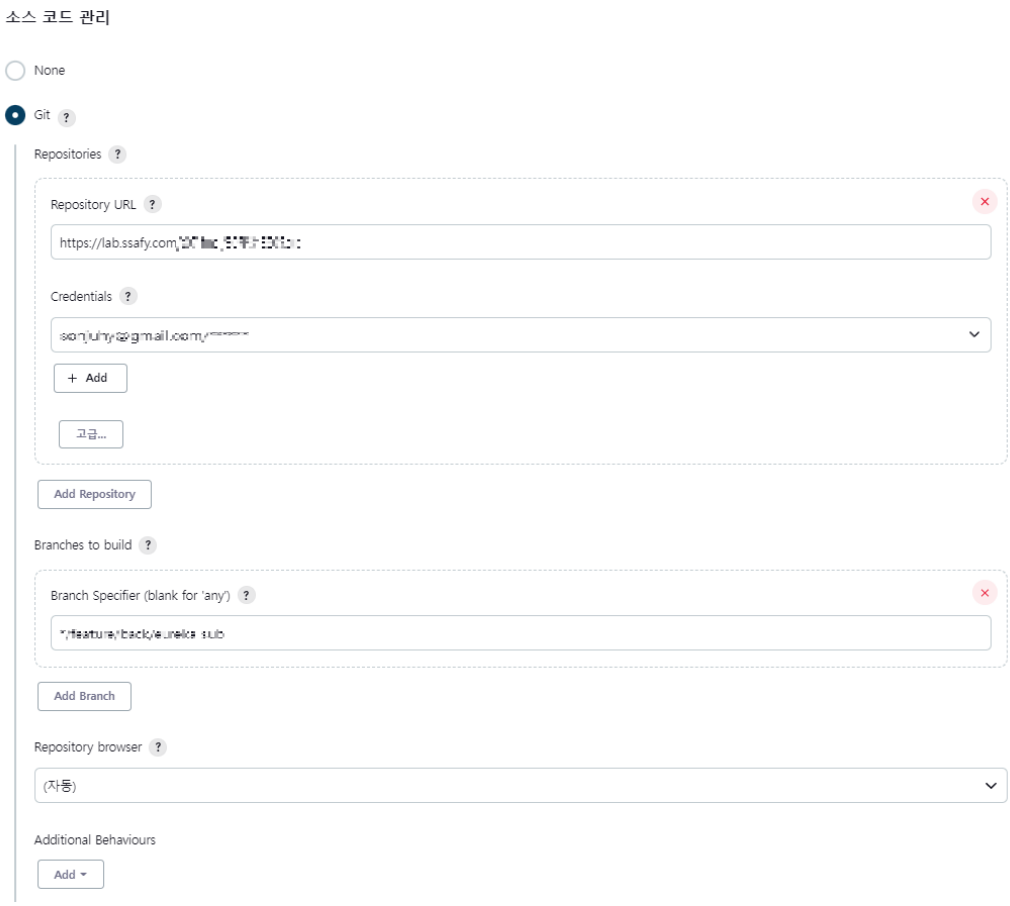
   - image start

     ```
     docker start —name jenkins —network host -p 9090:8080 -d jenkins/jenkins:lts
     ```

2. 플러그인
   a. Docker plugin
   b. Docker Commons Plugin
   c. Docker PipeLine
   d. Git Plugin
   e. Git client Plugin
   f. git Parameter Plug-in
   g. GitLab
   h. Gitlab API Plugin
   i. NodJS
3. 아이템 설정
   a. 단일 Spring Boot(gateway, eureka)
      - General
        - 이 빌드는 매개변수가 있습니다.

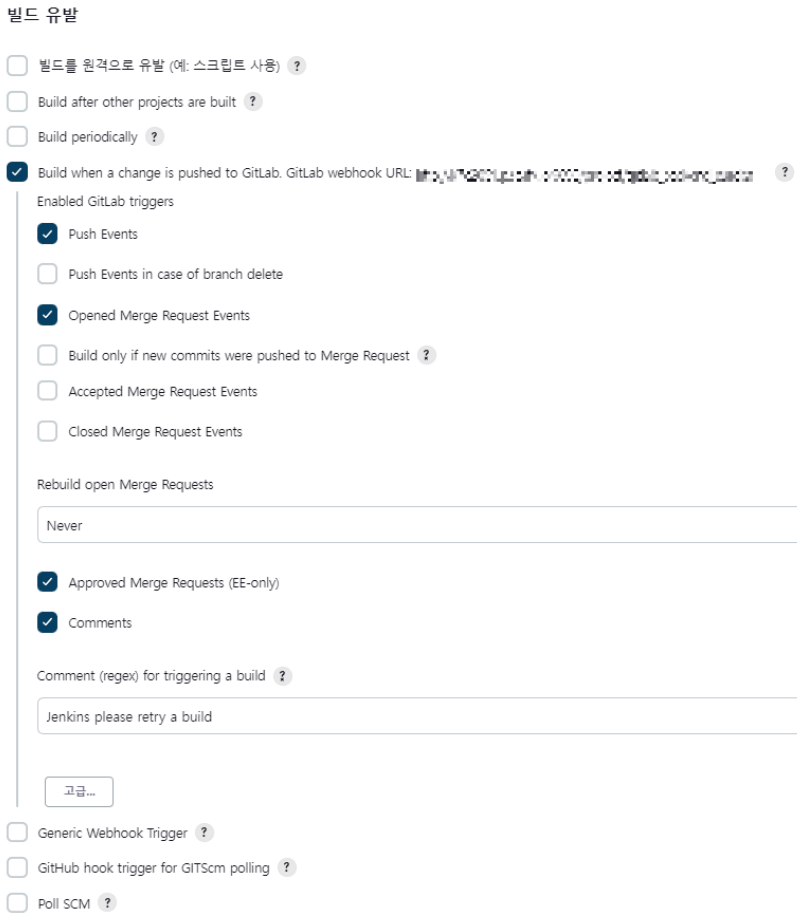

이 빌드는 매개변수가 있습니다
Git Parameter
Name
backend-eureka
Description
[Plain text] 미리보기
Parameter Type
Branch
Default Value
feature/back/eureka_sub
Branch
feature/back/eureka_sub
Tag Filter
*
Sort Mode
NONE
Selected Value
NONE
Use repository
Quick Filter
Required Parameter
List Size
5
매개변수 추가
필요한 경우 concurrent 빌드 실행
고급...

- 소스코드 관리
  - Git



- Repository URL : 프로젝트 Git Clone URL
- Credentials : Git 계정
- Bracnh Specifier : 빌드할 Repository

- 빌드 유발



- Build Steps

  - Command

```
cd {gradlew file path}
chmod 755 gradlew
./gradlew build
docker build -t {image name} .
if [ -z $(docker ps --filter 'name={container name}' -a -q)];
```

```
then
   docker run -d --name {container name} -p {host port}:{docker port} --network host {image name}
else
   docker stop {container name}
   docker rm $(docker ps --filter 'status=exited' -a -q)
   docker run -d --name s{container name} -p {host port}:{docker port} --network host {image name}
fi
```

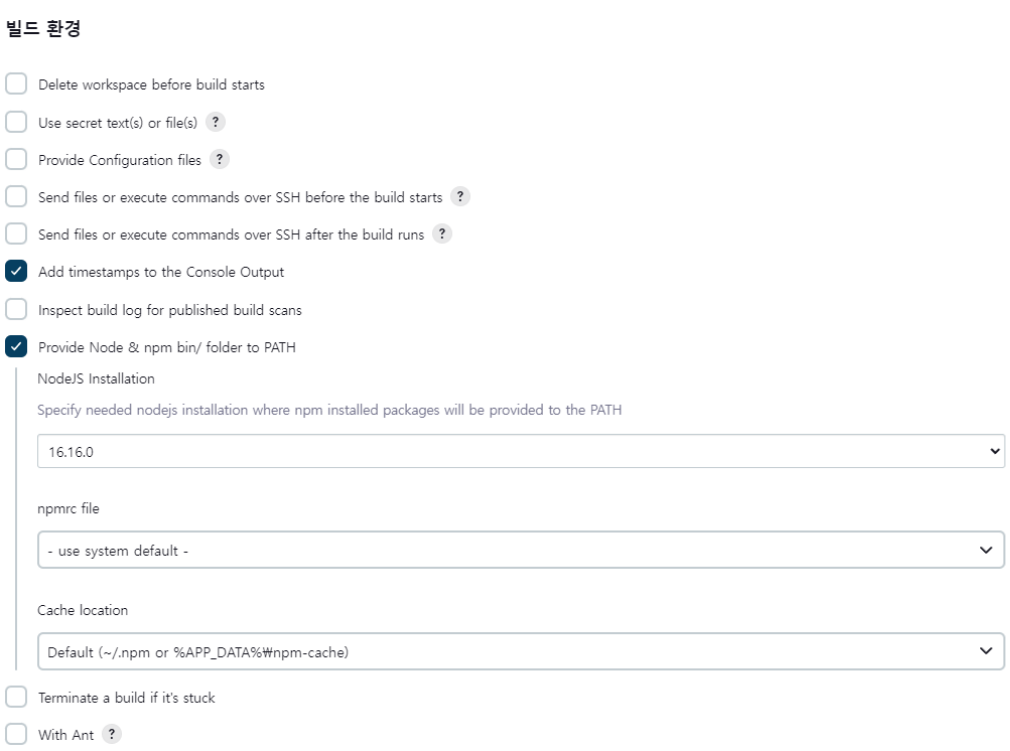
b. 복수 Spring Boot(user, anlz, simul)

- Build Steps
  - Command

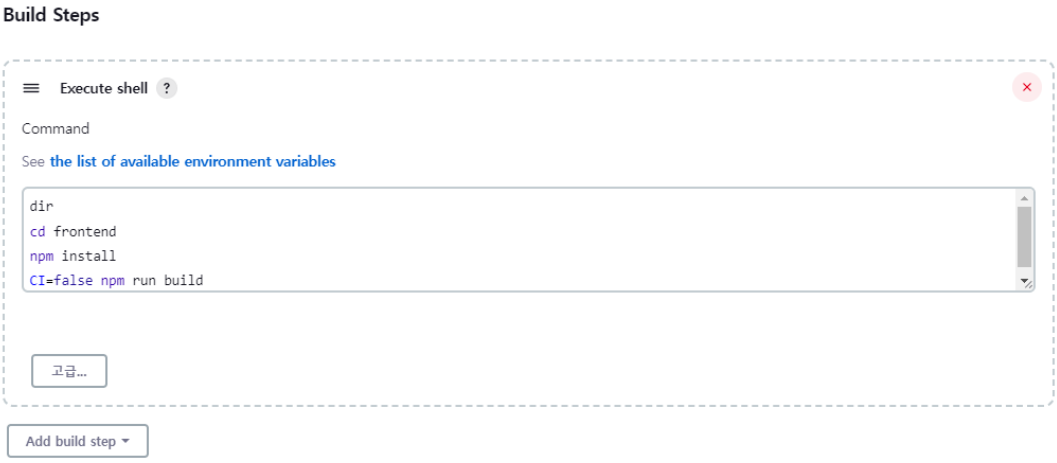
```
cd {gradlew file path}
chmod 755 gradlew
./gradlew build
docker build -t {image name} .
for i in 2 3
do
  if [ -z $(docker ps --filter 'name={container name}_'$i'' -a -q)];
  then
     docker run -d --name {container name} -p {host port}$i:{docker port}$i --network host {image name}
  else
     docker stop {container name}
     docker rm $(docker ps --filter 'status=exited_'$i'' -a -q)
     docker run -d --name s{container name} -p {host port}$i:{docker port}$i --network host {image name}
  fi
done
```

c. React

- 빌드 환경



- Build Steps



- Command

3. Deployment Command

3.1. React Storybook

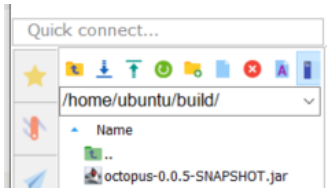
1. React App으로 이동(S07P31E205/frontend)

2. 아래 명령어로 실행

```
npm run storybook
```

3.2. Front & Back End Server

1. 빌드 파일 위치한 폴더 이동 (/home/ubuntu/build)



2. 아래 명령어로 실행

```
java -jar [Server File Name].jar
```

3.3. Nginx Web Server

1. 상태 확인

```
sudo systemctl status nginx
```

2. 프로세스 시작

```
sudo systemctl start nginx
```

3. 프로세스 종료

```
sudo systemctl stop nginx
```

4. 프로세스 재시작

```
sudo systemctl restart nginx
```

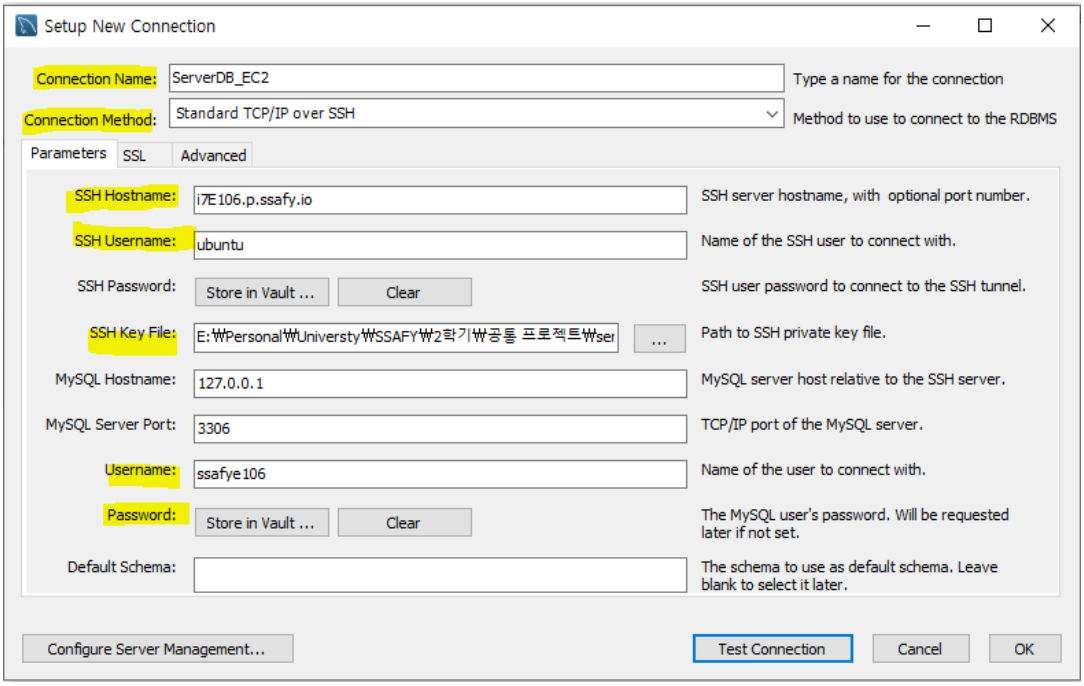
4. How to use the MySQL workbench

4.1. Standard TCP/IP over SSH Connection

1. MySQL Workbench 연결

2. 홈 화면에서 MySQL Connections 에서 + 추가

3. 아래 사진처럼 설정.



- Connection Name : 본인이 해당 커넥션이 어떤건지 알아보기 쉽게 설정
- Connection Method : Standard TCP/IP → Standard TCP/IP over SSH 로 변경
- SSH Hostname : DB 서버 도메인 명으로 설정
- SSH Username : ubuntu 로 설정
- SSH Key File : 공유 받은 pem 파일로 설정 (SSH Password는 설정 안함.)
- Username : 계정 ID 으로 설정
- Password : 계정 PW 로 설정

4. Test Connection으로 연결 확인

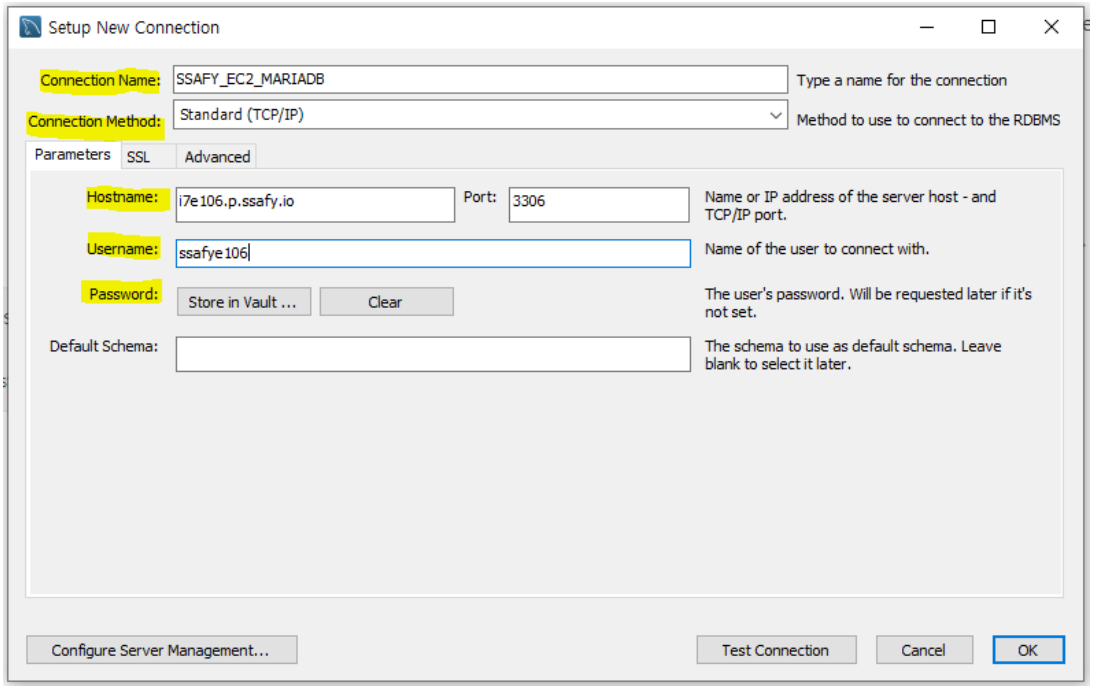
   ※ MariaDB ↔ MySQL 버전 호환 관련 Warning이 뜰 수 도 있는데 무시하고 연결.

5. 연결 테스트가 성공 - OK 누르고 사용

4.2. Standard TCP/IP 연결

1. 홈 화면에서 MySQL Connections 에서 + 추가

2. 아래 사진처럼 설정.



- Connection Name : 본인이 해당 커넥션이 어떤건지 알아보기 쉽게 설정
- Connection Method : Standard TCP/IP 그대로 사용
- Hostname : k7e205.p.ssafy.io 으로 설정 (도메인 이름)
- Username : 계정 ID 으로 설정
- Password : 계정 PW 으로 설정

3. Test Connection으로 연결 확인

   ※ MariaDB ↔ MySQL 버전 호환 관련 Warning이 뜰 수 도 있는데 무시하고 연결.

4. 연결 테스트가 성공하면 OK 누르고 사용

4.3. (공통)Spring Boot에서 연결

- **application.propertis에서 DB 연결 구문을 아래와 같이 수정**

```
spring.datasource.url=jdbc:mysql://[서버도메인]/[스키마 명]?serverTimezone=Asia/Seoul
spring.datasource.username=[계정 ID]
spring.datasource.password=[계정 PW]
```

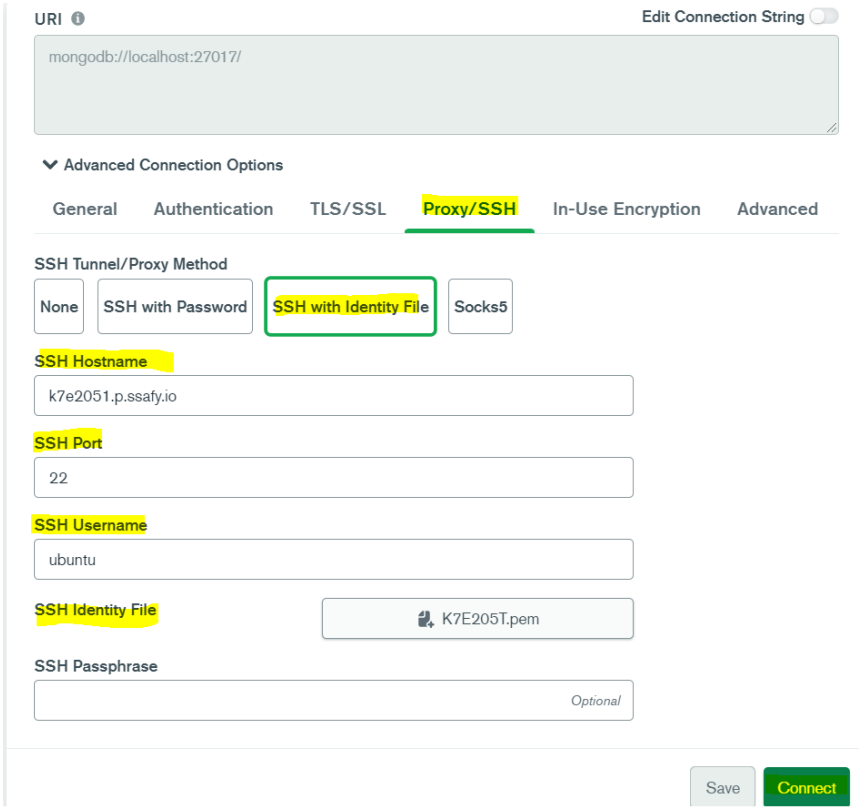
5. How to use the MongoDB Compass

5.1. SSH with Identity File

1. MongoDB Compass 연결

2. 홈 화면에서 New Connections 에서 + 추가

3. 아래 사진처럼 설정.



- Connection Option: SSH with Identity File 로 변경
- SSH Hostname : DB 서버 도메인 명으로 설정
- SSH Port : 22
- SSH Username : ubuntu 로 설정
- SSH Identity File : 공유 받은 pem 파일로 설정 (SSH Password는 설정 안함.)

4. Connect 로 연결 확인

5. 연결 테스트가 성공 - OK 누르고 사용

5.2. (공통)Spring Boot에서 연결

- **application.propertis에서 DB 연결 구문을 아래와 같이 수정**

```
implementation("org.springframework.boot:spring-boot-starter-data-mongodb")
```

5.3. Python에서 연결

```
import pymongo

connection = pymongo.MongoClient("mongodb://localhost:27017/")
db = connection["simulation"]
```

6. Nginx default

```
server {
  listen 80;
  server_name k7e205.p.ssafy.io;
  return 301 https://k7e205.p.ssafy.io$request_uri;

}
server {
  listen 443 ssl http2;
  server_name k7e205.p.ssafy.io;

  # ssl 인증서 적용하기
  ssl_certificate /etc/letsencrypt/live/i7e106.p.ssafy.io/fullchain.pem;
  ssl_certificate_key /etc/letsencrypt/live/i7e106.p.ssafy.io/privkey.pem;

  location / {
    # proxy 설정 (모든 요청 -> 8080으로 전송)
    proxy_pass http://localhost:8080;
    proxy_set_header Host $http_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;

    try_files $uri $uri/ /index.html =404;
  }
  location ~ /anlz {
    proxy_pass http://127.0.0.1:8081$request_uri;
  }
}
server {
    if ($host = k7e205.p.ssafy.io) {
        return 301 https://$host$request_uri;
    } # managed by Certbot


  listen 80;
  server_name k7e205.p.ssafy.io;
    return 404; # managed by Certbot


}
```

7. EC2 Settings

7.1. Docker

1. repository 최신 상태 업데이트 및 HTTP 패키지 설치

```
sudo apt-get update
```

```
sudo apt-get install -y ca-certificates \
    curl \
    software-properties-common \
    apt-transport-https \
    gnupg \
    lsb-release
```

2. GPG 키 및 저장소 추가

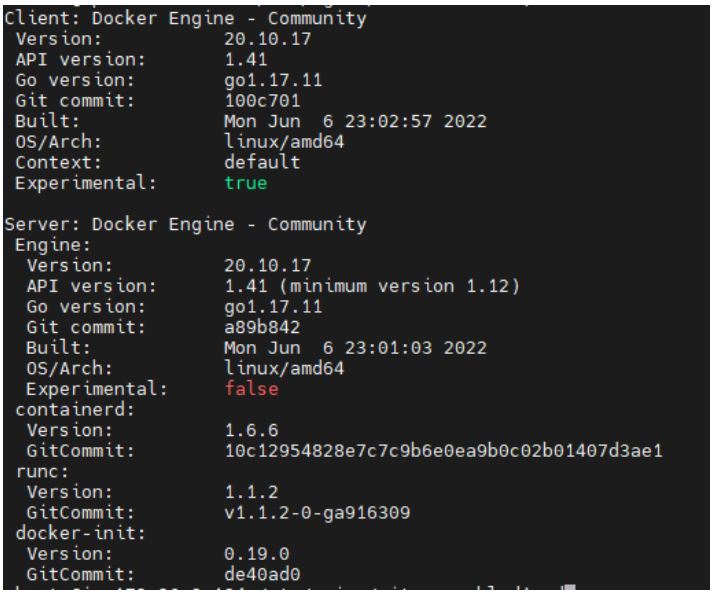
```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg
```

3. 도커 엔진 설치

```
sudo apt install docker-ce docker-ce-cli containerd.io
```

4. 도커 버전 확인

```
sudo docker version
```



명령어 실행 화면

7.2. Nginx

7.2.1. 설치

1. 아래 명령어로 설치

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

7.2.2. SSL 인증서 발급

1. 아래 명령어로 nginx certbot 툴 설치

```
sudo add-apt-repository ppa:certbot/certbot
```

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

2. **CertBot 실행 - SSL 인증서 발급**
   a. 아래 명령어 실행으로 CertBot 실행

```
sudo certbot --nginx -d 자신의도메인
```

   b. 이후 이메일과 약관 동의 하면 1,2 중 선택 하라고 한다. 각 내용은 다음과 같다.
      i. 1번 선택 : http 요청을 https로 리다이렉션 하지 않는다
      ii. 2번 선택 : http 요청을 https로 리다이렉션 한다.

   c. 선택 후 nginx가 재시작 되면서 https(SSL 인증서) 적용이 완료된다.

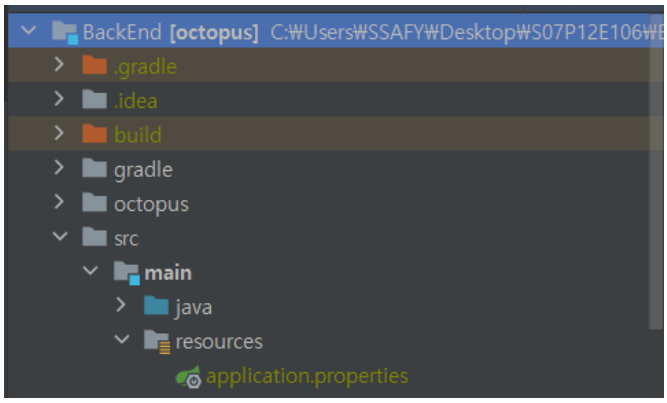
8. Files ignored

1. BackEnd - application.properties

※ DB 계정 등 보안 관련 정보 때문에 별도 관리

a. 파일 위치

- [BackEnd Folder] - src - main - resources - application.properties



b. 파일 내용

- DB(Schma) Auth Info

- JPA Setting
- Swagger

9. etc) Tips

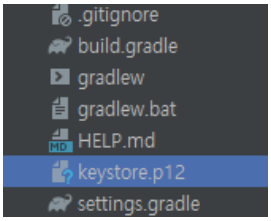
9.1. How to apply temporary SSL to React, Spring Boot project

9.1.1. Spring Boot (JAVA) SSL 적용

1. **Spring boot에 https 인증 임시로 localhost에 적용**
   a. 인증서 설치
      i. 실행할 프로젝트 폴더에 아래의 명령어 실행

```
keytool -genkey -alias octopus -storetype PKCS12 -keyalg RSA -keysize 2048 -keystore keystore.p12 -validity 3650
```

      ii. 아래 사진처럼 **keystore.p12**가 생성 되었는지 확인.



   b. 인증서 적용
      i. application.propertis 에 아래 내용 추가

```
## local ssl
server.ssl.enabled=true
server.ssl.key-store=keystore.p12
server.ssl.key-store-password: [본인이 설정한 패스워드]
server.ssl.key-store-type: PKCS12
server.ssl.key-alias: octopus
```

      ii. https://localhost:8080 (혹은 본인이 설정한 포트) 로 잘 작동하는지 확인.

9.1.2. React (JS) SSL 적용

1. 파일 추가
   a. 인증서 설치
      i. Windows 패키지 관리자인 Chocolatey(약칭 : Choco)를 아래와 같이 설치.
         1. cmd 창을 관리자 권한으로 실행.

```
@"%SystemRoot%\System32\WindowsPowerShell\v1.0\powershell.exe" -NoProfile -InputFormat None -ExecutionPolicy B
ypass -Command " [System.Net.ServicePointManager]::SecurityProtocol = 3072; iex ((New-Object System.Net.WebCli
ent).DownloadString('https://chocolatey.org/install.ps1'))" && SET "PATH=%PATH%;%ALLUSERSPROFILE%\chocolatey\b
in"
```

이 내용을 복사 붙여넣기 하여 실행하여 Choco를 설치.

ii. **관리자 권한으로 실행**한 cmd에 아래와 같이 입력하여 mkcert를 설치.

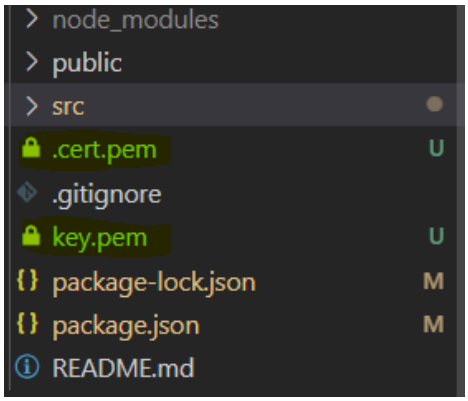
```
choco install mkcert
```

```
mkcert -install
```

iii. 이후 react가 설치된 폴더에 이동하여 마찬가지로 cmd창에 아래와 같이 입력하여 인증서를 설치.

```
mkcert -key-file ./key.pem -cert-file .cert.pem “localhost”
```

iv. 아래 사진처럼 파일 두개가 생성되어 있는걸 확인.



생성된 파일 두개 이름 : .cert.pem 그리고 key.pem

b. 인증서 적용

i. package.json 에서 scripts - start를 아래 줄로 변경

```
"start": "set HTTPS=true&&set SSL_CRT_FILE=.cert.pem&&set SSL_KEY_FILE=key.pem&&react-scripts start",
```

ii. 프로젝트 시작 후 적용 확인