SWING BACKEND구축형 환경설정 가이드

개정이력

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **버전** | **변경일** | **내용** | **작성자** |
| 1 | 1.0 | 2019-04-01 | 최초작성 | 김유경 |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

목차

[**1.** 개요 4](#_Toc5026805)

[**2. 시스템 구성** 5](#_Toc5026806)

[**3. 사전 설치 프로그램** 7](#_Toc5026807)

[**4. common-lib를 배포하기 위한 Maven repository 구축** 10](#_Toc5026808)

[**5. DISCOVERY-SERVER JENKINS 설정** 14](#_Toc5026809)

[**6. API-GATEWAY JENKINS 설정** 17](#_Toc5026810)

[**7. HOLIDAY-SERVICE JENKINS 설정** 20](#_Toc5026811)

[**8. SWINGAUTH-SERVICE JENKINS 설정** 23](#_Toc5026812)

[**9. SWING-SERVICE JENKINS 설정** 26](#_Toc5026813)

[**10. SCHEDULE-SERVICE JENKINS 설정** 29](#_Toc5026814)

[**11. 빌드** 32](#_Toc5026815)

# 개요

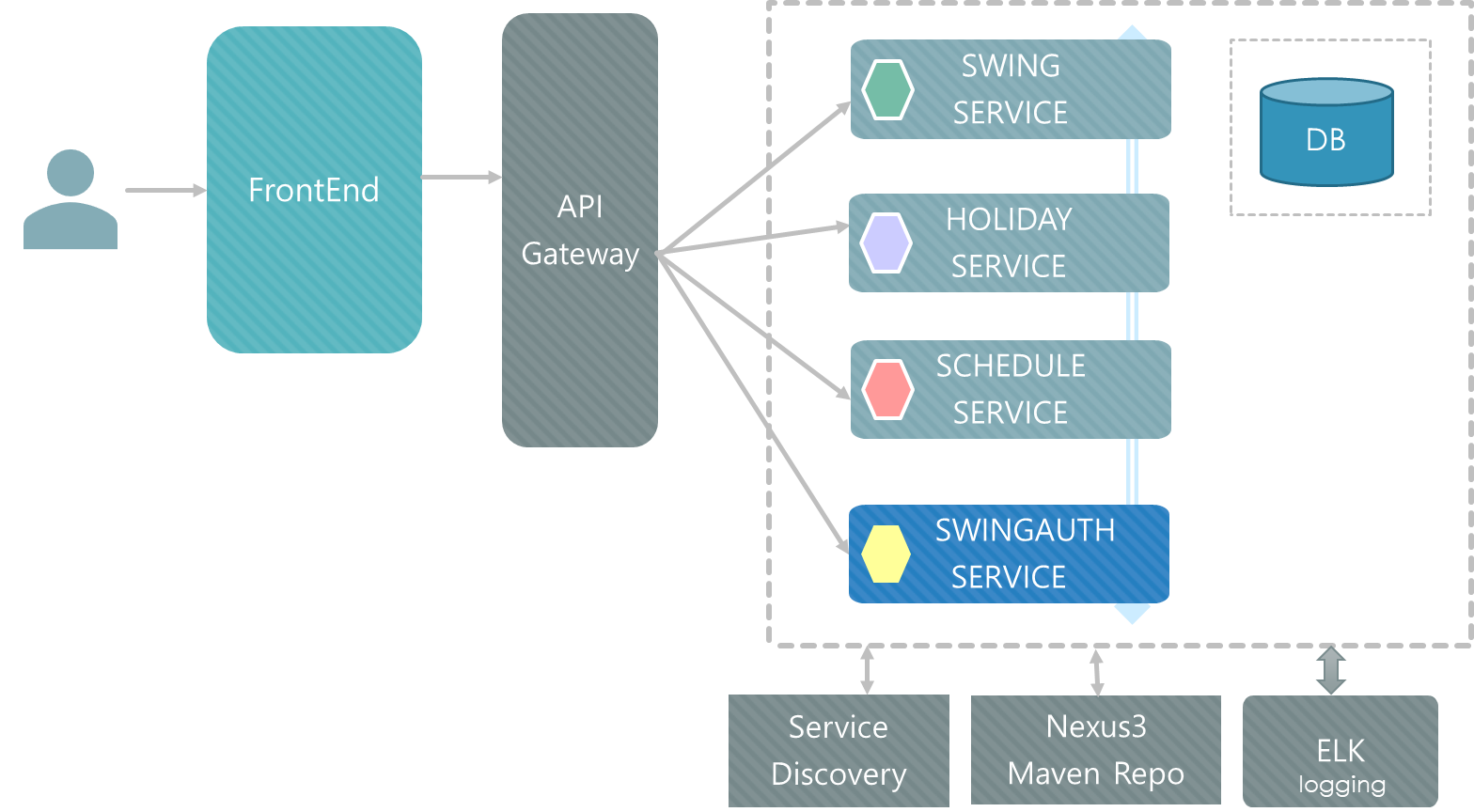
본 문서는 SWING BACKEND 구축형 환경 설정 가이드 문서로, 클라우드가 아닌 리눅스 환경에서 환경설정을 안내합니다.

리눅스 환경에서 GIT, JENKINS를 이용하여 배포하는 설정 가이드로, 사전에 GIT과 JENKINS는 설정되어 있다는 가정하에 내용을 기술합니다.

또한, ELK Stack의 설치에 관한 내용도 인터넷을 검색하면 많이 나와 있으니 이점 참고 하시고 구성하시기 바랍니다.

현재 아이피 등의 설정정보는 테스트서버의 구축과 본 문서에서 설명하고자 하는 방향이 같아, 테스트 서버를 기준으로 설명됩니다.

# 시스템 구성



BackEnd 서비스는 위와 같이 API Gateway를 통해서 이루어 집니다. API Gateway를 통해 요청된 서비스를 API Gateway가 확인하고, 필터링을 처리할 지 여부를 확인합니다. 요청된 헤더의 토큰정보로 인증을 처리후 요청 서비스를 호출합니다.

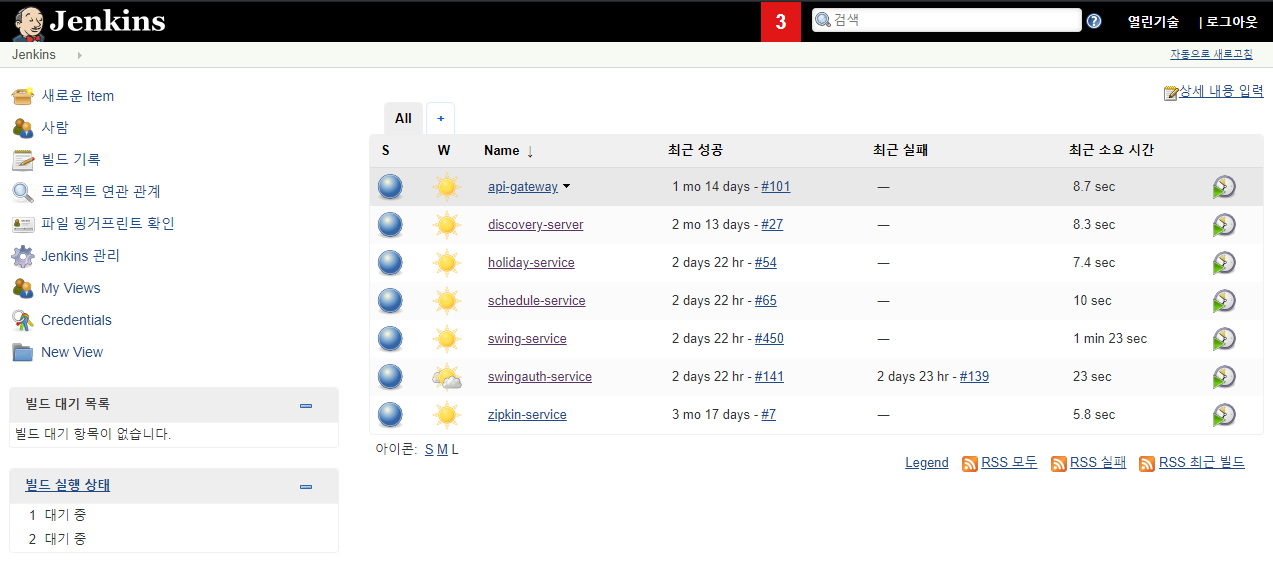
|  |  |  |  |
| --- | --- | --- | --- |
| 서비스명 | 아이피 | 포트 | 비고 |
| api-gateway | 10.10.10.162 | 8765 | 필터링 인증처리 |
| Backend 서비스의 가장 앞단에 위치하여, 서비스 요청에 대한 토큰인증을 SWINGAUTH-SERVICE에게 요청하여 인증처리를 담당하고 있다. | | |
| SPRING BOOT, SPRING CLOUD ZUUL (NETFLEX ZUUL) | | |
| service-discovery (discovery-server) | 10.10.10.162 | 8761 | SPRING EUREKA SERVER 서비스 검색 |
| MSA를 구성함에 있어서 각 모듈의 Lookup, Scale상태, 모듈의 Health 상태등을 관리하는 모듈로, 서비스명으로 호출이 들어오면 서비스가 어디에 있는지 요청이 가능한 상태인지 체크를 해서 해당 서버의 ip와 포트의 정보를 돌려주어 실행되도록 하는 역할. | | |
| SPRING BOOT, SPRING CLOUD EUREKA SERVER | | |
| swing-service | 10.10.10.162 | 8751 | SWING 업무 서비스 |
| SPRING BOOT, JPA | | |
| swingauth-service | 10.10.10.162 | 8750 | SWING 인증 서비스 |
| SPRING BOOT, JPA | | |
| holiday-service | 10.10.10.162 | 8753 | **휴일정보, 시스템팝업공지 정보를 제공** |
| SPRING BOOT, JPA | | |
| schedule-service | 10.10.10.162 | 8096 | SWING 스케줄링 서비스 |
| 계약자동처리, PM 및 점검 스케줄링 | | |
| SPRING BOOT, JPA, QUARTZ SCHEDULE, SPRING BATCH | | |
| nexux3 maven repo | 10.10.11.90 | 8081 | COMMON-LIB 메이븐 리포지토리 |
| **공통으로 사용되는 자바 클래스 라이브러리를 PRIVATE MAVEN REPOSITORY에 배포하여, 중복없이 해당 라이브러리를 공유하여 사용할 수 있도록** | | |
| elk stack | 10.10.10.163 | 5000 | 로그관리 |

*아이피정보는 가상으로 구성되었습니다. 해당 아이피가 같으면 같은 서버로 구성되는 의미입니다. 아이피가 다른 경우 다른 서버로 구축해야 하는 경우 입니다.*

# **사전 설치 프로그램**

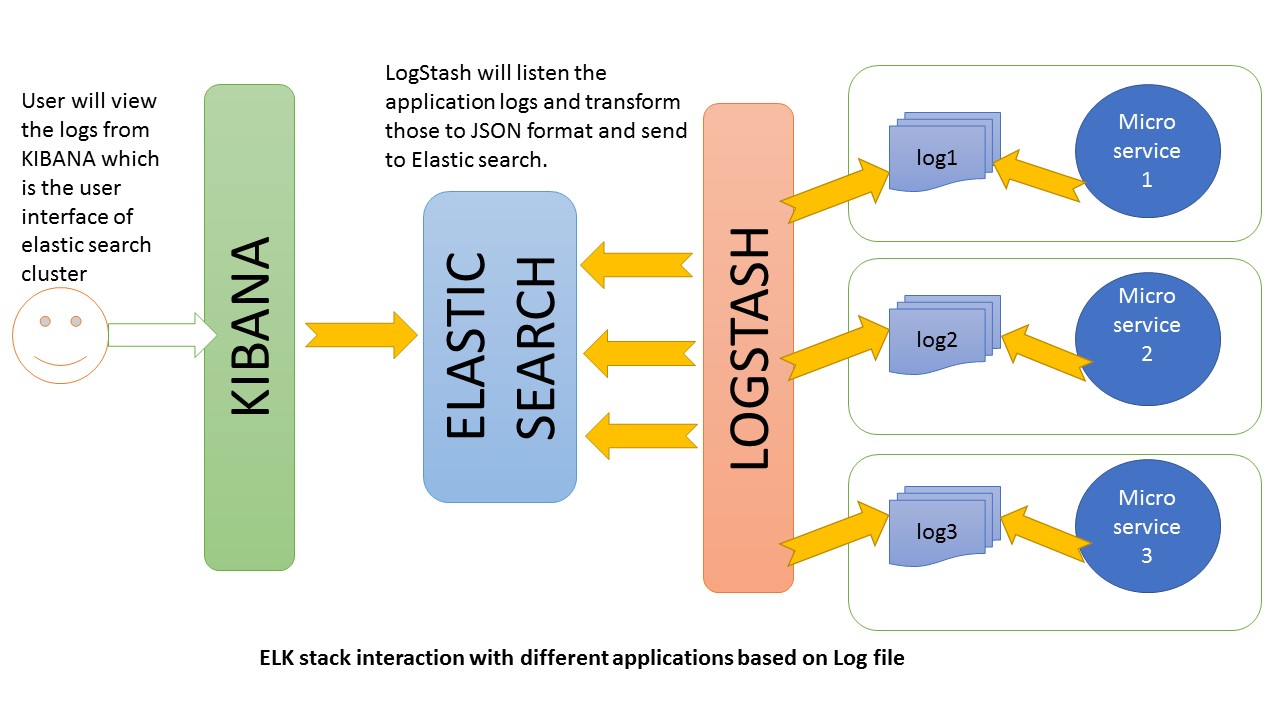
* 1. **MY SQL 5.7**
     1. db는 별도의 서버에 설치되어 있어야 합니다. 한 서버에 같이 구성하는 것은 보안 측면에서 좋지 않습니다. 되도록 다른 서버에 설치 하십시오.
     2. MY SQL 5.7버전으로 개발되었으므로, 5.7버전으로 설치 하십시오.
  2. **GIT**
     1. 개발시 yona git을 별도의 서버에 설치하여 개발되었습니다.
     2. github를 이용하거나, 다른 git을 설치하여 사용하셔도 됩니다. git은 소스코드의 형상 관리를 위해서 사용됩니다.
     3. 테스트서버의 <http://10.10.10.162:9100/>에 설치되어 사용중입니다. 자신의 계정을 사용하여 로그인하시면 됩니다. 각 서비스 명으로 소스가 배포되어 있습니다.
  3. **JENKINS (Continuous Integration)**
     1. 개발시 10.10.10.162:8080 번에 설치되어 테스트 서버의 jenkins로 운영중입니다. (관리자계정정보 : admin/yullin@7033)
     2. JENKINS는 다수의 개발자들이 하나의 프로그램을 개발할 때 버전 충돌을 방지하기 위해 각자 작업한 내용을 공유영역에 있는 저장소에 빈번히 업로드함으로써 지속적 통합이 가능하도록 해줍니다. 소스가 변경될 때마다 배포할 수도 있지만, 소스를 모두 배포후 수동으로 배포하여 테스트 하고 있습니다. (FrontEnd에 영향을 줄 수 있으므로)

아래 화면은 테스트 서버에서 운영되고 있는 JENKINS 설정 화면입니다.



* 1. **ELK STACK**
     1. Elastic Search, Log Stash, Kibana로 시스템에서 발생된 로그파일을 수집하여 Kibana에서 Visualize화 시켜 관리할 수 있습니다.
     2. ELK STACK을 한번에 배포할 수 있도록 Docker Image 파일이 많이 오픈되어 있다. 개발시 10.10.10.163서버에 Docker 이미지 파일을 다운받아 구축하였다. Docker이미지 파일로 생성시에 삭제 및 재구축이 용이하므로, 적절한 파일을 사용하여 구축후 알맞은 이미지를 만든 뒤, 해당 이미지를 보관하여 사용하는 것이 좋습니다.

아래 그림을 참고하여, ELK Stack 구조를 이해하시면 됩니다.

****

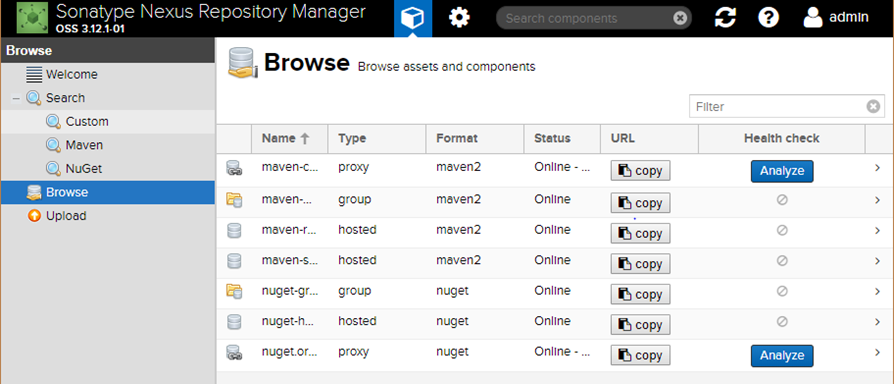
# **common-lib를 배포하기 위한 Maven repository 구축**

common-lib는 swing-service, swingauth-service, holiday-service등의 서비스에서 공통으로 사용되는 DTO, 유틸리티 등의 클래스를 라이브러리화 시킨 것입니다. common-lib를 내부 메이븐 리포지토리를 구축하여 배포되도록 구성하였습니다.

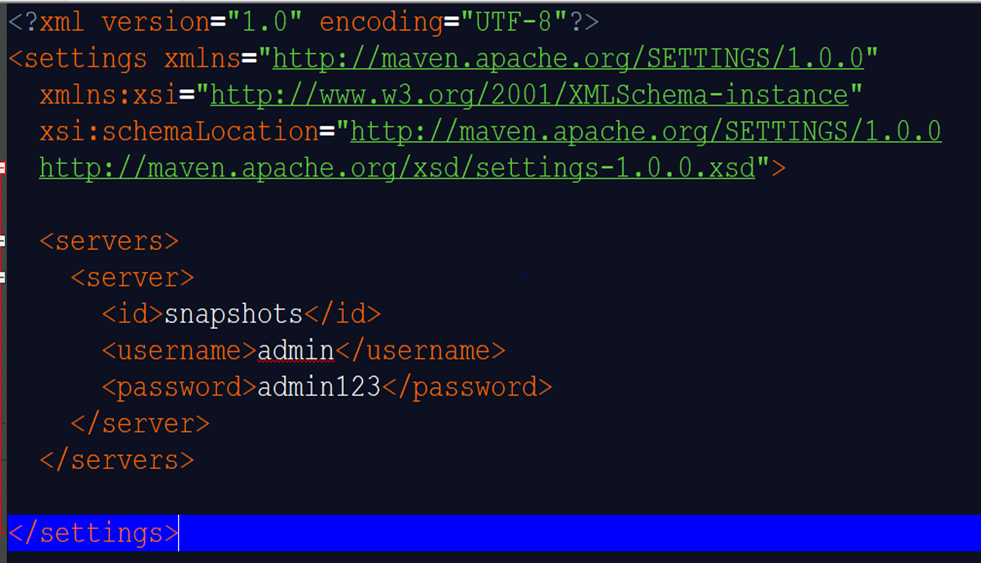
구축방법 및 배포 방법은 아래 내용을 참고 하시기 바랍니다.

* 1. Docker Image를 이용하여 설치 (해당 예제는 별도의 Maven Repository 서버에, Windows OS환경에서 구축한 예제입니다.)

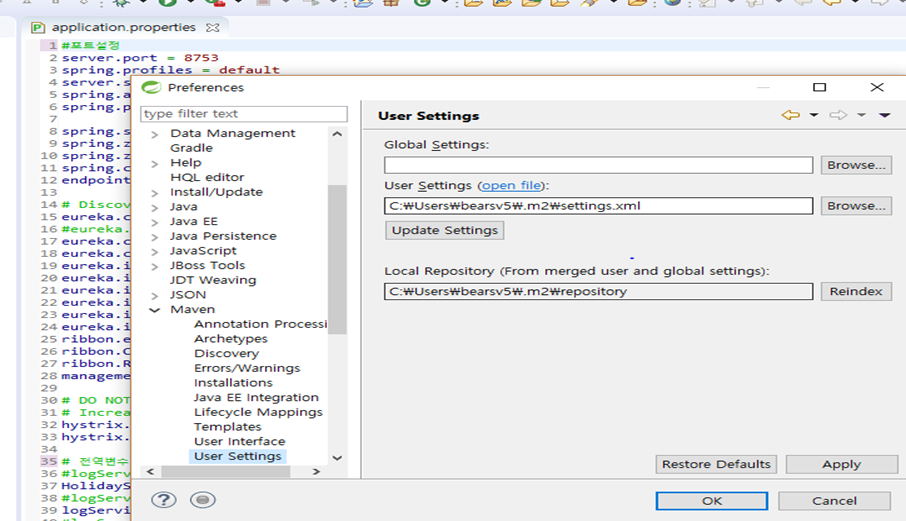
1. Sonatype Nexus3 Docker: sonatype/nexus3을 설치
2. 참고 https://github.com/sonatype/docker-nexus3
3. 설치방법 : $ docker run -d -p 8081:8081 --name nexus sonatype/nexus3
4. 확인 : http://localhost:8081 (아이디/암호 : admin/admin123)



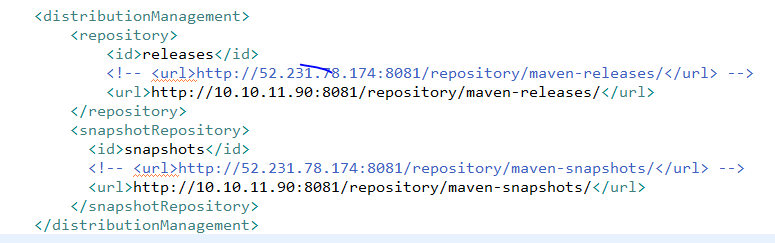
* 1. Spring common-lib를 maven repository에 올려보기 (해당 서버에 구축)
  2. maven 설정파일에 생성된 repository 정보 설정
  3. C:\Users\사용자\.m2\settings.xml



* 1. STS 설정 (개발 환경 이클립스)



* 1. COMMON-LIB의 pom.xml



현재 테스트용 Maven Repository는 10.10.11.90이므로, 위 설정파일에서 10.10.11.90으로 배포합니다. 배포방법은 아래와 같습니다.

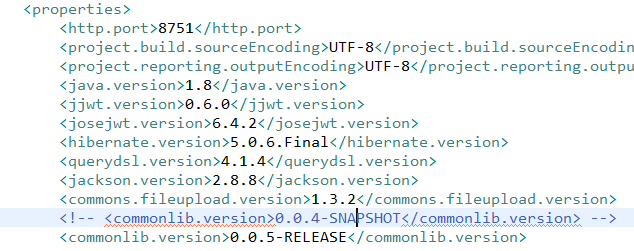
아래와 명령어로 nexus repository로 업로드 합니다. 아래는 Release 버전의 배포 예시입니다.

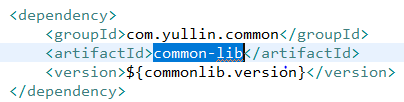
D:\workspace\swing-microservices\common-lib>mvn deploy -DrepositoryId=releases -Dversion=0.0.4-RELEASE

아래는 Snapshot 버전의 배포 예시입니다.

D:\workspace\swing-microservices\common-lib>mvn deploy -DrepositoryId=snapshots -Dversion=0.0.4-SNAPSHOT

* 1. private maven repository에 등록된 common-lib를 spring boot 프로젝트에서 아래와 같은 설정으로 사용할 수 있습니다.





# **DISCOVERY-SERVER JENKINS 설정**

* 1. 새로운 ITEM을 눌러 discovery-server를 Maven Project로 추가합니다.
  2. 아래 그림과 같이 각 설정정보를 입력합니다.

|  |  |
| --- | --- |
| 10개까지만 이전 이력을 저장하도록 설정되어 있습니다. | 사용하고 있는 Git 정보를 설정합니다. |
|  | pre Steps 은 배포전 처리할 작업입니다.  ***echo "fuser -k 8761/tcp || true" | at now***  현재 Running중인 포트번호 8761 을 Kill하는 명령어 입니다.  이 명령어를 하지 않는 경우 배포 오류가 발생합니다. |
| Execute Schell 설정으로 jar파일을 자동으로 배포하도록 합니다. 아래 명령어는 빌드 후 1분후에 파일을 배포하는 명령어 입니다.  **echo "java -jar /var/lib/jenkins/workspace/discovery-server/target/discovery-server-0.0.1-SNAPSHOT.jar" | at now + 1 minutes** | |

위와 같이 설정 후 저장 버튼을 눌러 설정을 저장합니다.

# **API-GATEWAY JENKINS 설정**

* 1. 새로운 ITEM을 눌러 api-gateway를 Maven Project로 추가합니다.
  2. 아래 그림과 같이 각 설정정보를 입력합니다.

|  |  |
| --- | --- |
| 10개까지만 이전 이력을 저장하도록 설정되어 있습니다. | 사용하고 있는 Git 정보를 설정합니다. |
|  | pre Steps 은 배포전 처리할 작업입니다.  ***echo "fuser -k 8765/tcp || true" | at now***  현재 Running중인 포트번호 8765 을 Kill하는 명령어 입니다.  이 명령어를 하지 않는 경우 배포 오류가 발생합니다. |
| Execute Schell 설정으로 jar파일을 자동으로 배포하도록 합니다. 아래 명령어는 빌드 후 1분후에 파일을 배포하는 명령어 입니다.  **echo "java -jar /var/lib/jenkins/workspace/api-gateway/target/api-gateway-0.0.1-SNAPSHOT.jar" | at now + 1 minutes** | |

위와 같이 설정 후 저장 버튼을 눌러 설정을 저장합니다.

# **HOLIDAY-SERVICE JENKINS 설정**

* 1. 새로운 ITEM을 눌러 holiday-service를 Maven Project로 추가합니다.
  2. 아래 그림과 같이 각 설정정보를 입력합니다.

|  |  |
| --- | --- |
| 10개까지만 이전 이력을 저장하도록 설정되어 있습니다. | 사용하고 있는 Git 정보를 설정합니다. |
|  | pre Steps 은 배포전 처리할 작업입니다.  ***echo "fuser -k 8753/tcp || true" | at now***  현재 Running중인 포트번호 8753 을 Kill하는 명령어 입니다.  이 명령어를 하지 않는 경우 배포 오류가 발생합니다. |
| Execute Schell 설정으로 jar파일을 자동으로 배포하도록 합니다. 아래 명령어는 빌드 후 1분후에 파일을 배포하는 명령어 입니다.  **echo "java -jar /var/lib/jenkins/workspace/holiday-service/target/holiday-service-0.0.1-SNAPSHOT.jar" | at now + 1 minutes** | |

위와 같이 설정 후 저장 버튼을 눌러 설정을 저장합니다.

# **SWINGAUTH-SERVICE JENKINS 설정**

* 1. 새로운 ITEM을 눌러 swingauth-service를 Maven Project로 추가합니다.
  2. 아래 그림과 같이 각 설정정보를 입력합니다.

|  |  |
| --- | --- |
| 10개까지만 이전 이력을 저장하도록 설정되어 있습니다. | 사용하고 있는 Git 정보를 설정합니다. |
|  | pre Steps 은 배포전 처리할 작업입니다.  ***echo "fuser -k 8750/tcp || true" | at now***  현재 Running중인 포트번호 8750 을 Kill하는 명령어 입니다.  이 명령어를 하지 않는 경우 배포 오류가 발생합니다. |
| Execute Schell 설정으로 jar파일을 자동으로 배포하도록 합니다. 아래 명령어는 빌드 후 1분후에 파일을 배포하는 명령어 입니다.  **echo "java -jar /var/lib/jenkins/workspace/swingauth-service/target/swingauth-service-0.0.2-SNAPSHOT.jar" | at now + 1 minutes** | |

위와 같이 설정 후 저장 버튼을 눌러 설정을 저장합니다.

# **SWING-SERVICE JENKINS 설정**

* 1. 새로운 ITEM을 눌러 swing-service를 Maven Project로 추가합니다.
  2. 아래 그림과 같이 각 설정정보를 입력합니다.

|  |  |
| --- | --- |
| 10개까지만 이전 이력을 저장하도록 설정되어 있습니다. | 사용하고 있는 Git 정보를 설정합니다. |
|  | pre Steps 은 배포전 처리할 작업입니다.  ***echo "fuser -k 8751/tcp || true" | at now***  현재 Running중인 포트번호 8751 을 Kill하는 명령어 입니다.  이 명령어를 하지 않는 경우 배포 오류가 발생합니다. |
| Execute Schell 설정으로 jar파일을 자동으로 배포하도록 합니다. 아래 명령어는 빌드 후 1분후에 파일을 배포하는 명령어 입니다.  **echo "java -XX:MaxMetaspaceSize=512m -XX:MetaspaceSize=256m -jar /var/lib/jenkins/workspace/swing-service/target/swing-service-0.0.1-SNAPSHOT.jar" | at now + 1 minutes**  메모리 이슈로 배포시 위와 같이 사이즈를 설정해 줄 수 있습니다. 각 배포 환경에 따라서 설정 하세요. | |

위와 같이 설정 후 저장 버튼을 눌러 설정을 저장합니다.

# **SCHEDULE-SERVICE JENKINS 설정**

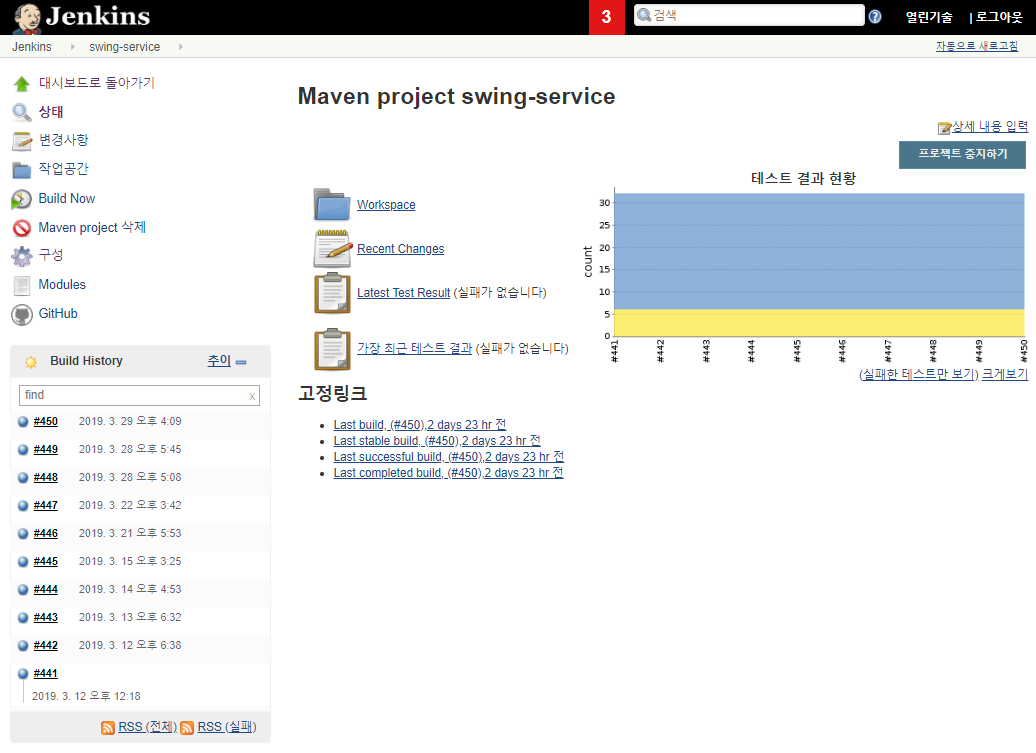
* 1. 새로운 ITEM을 눌러 schedule-service를 Maven Project로 추가합니다.
  2. 아래 그림과 같이 각 설정정보를 입력합니다.

|  |  |
| --- | --- |
| 10개까지만 이전 이력을 저장하도록 설정되어 있습니다. | 사용하고 있는 Git 정보를 설정합니다. |
|  | pre Steps 은 배포전 처리할 작업입니다.  ***echo "fuser -k 8096/tcp || true" | at now***  현재 Running중인 포트번호 8096 을 Kill하는 명령어 입니다.  이 명령어를 하지 않는 경우 배포 오류가 발생합니다. |
| Execute Schell 설정으로 jar파일을 자동으로 배포하도록 합니다. 아래 명령어는 빌드 후 1분후에 파일을 배포하는 명령어 입니다.  **echo "java -jar /var/lib/jenkins/workspace/schedule-service/target/schedule-service-0.0.1-SNAPSHOT.jar" | at now + 1 minutes** | |

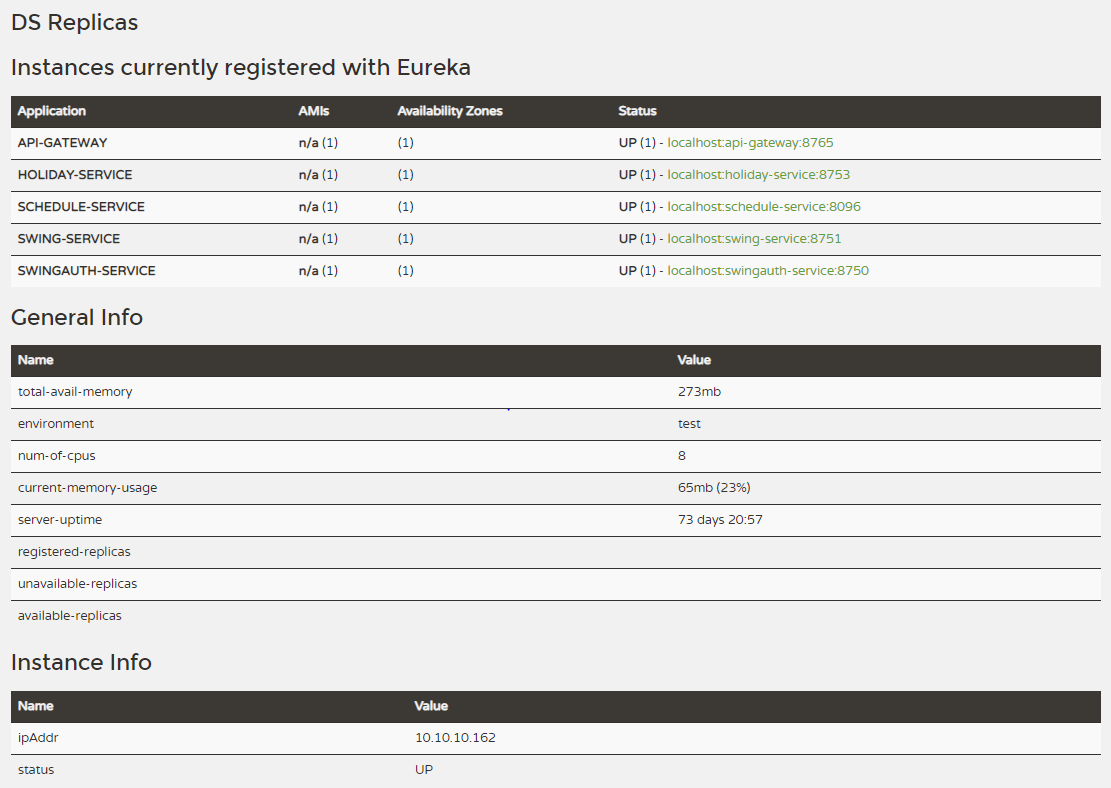
위와 같이 설정 후 저장 버튼을 눌러 설정을 저장합니다.

# **빌드**

* 1. 빌드는 아래와 같이 각 ITEM 설정 화면에서 ***”Build Now”*** 버튼을 눌러 진행합니다.



* 1. 빌드시 처리 상태를 확인하고자 하는 경우 Console out을 눌러 확인할 수 있습니다.
  2. 각 서비스는 discovery-server 에서 확인할 수 있습니다.



discovery-server는 위와 같이 각 서비스들의 상태를 확인할 수 있습니다. 이 정보는 각 서비스의 설정정보에 정의되어 있습니다.

* 1. Swagger를 통해서 데이터를 확인할 수 있습니다. 각 서비스의 URL/swagger-ui.html 로 조회해 볼 수 있습니다.

*아래는 swing-service의 swagger를 조회한 화면입니다. 상단의 그룹 정보를 눌러 업무별 서비스를 조회할 수 있습니다.*

