

- Nginx로 스프링 프로젝트 무중단 배포하기
  - 무중단 배포란?
- 블루 그린(Blue-Green Deployment) 방식
- Nginx로 블루-그린 무중단 배포 방식
  - 1. 스프링 버전 1.0.0 무중단 배포
  - 2. 스프링 버전 1.0.1 무중단 배포
  - 3. 스프링 버전 1.0.2 무중단 배포
- 1. EC2 환경 Nginx 설치 후 실행하기
  - 1) EC2에 Nginx 설치
  - 2) Nginx 실행
  - 3) EC2 인스턴스 보안그룹에 80번 포트 추가
  - 4) 80 포트로 http 접속해보기
- 2. 스프링 프로젝트와 Nginx 연동하기
  - 1) OAuth 리다이렉션 주소 추가
  - 2) nginx 설정 파일에서 프록시 설정 변경하기
  - 3) Nginx 재시작하기
  - 스프링 Nginx 연동 완료
- 3. Profile API 만들기
  - 1) profile API 추가하기
  - 2) Spring Security API 접근권한 열기
  - 3) 테스트 코드 작성하기
  - 4) 브라우저에서 profile 조회하기
- 4. 무중단 배포에 필요한 프로필 생성하기
  - 1) real1, real2 프로필 생성하기
  - 2) Nginx 설정 수정하기
- 5. 무중단 배포 스크립트 작성하기
  - 1) EC2에 step3 디렉토리 생성
  - 2) appspec.yml 설정 변경
  - 3) 무중단 배포 스크립트 작성
- 6. 무중단 배포 테스트하기
  - 파일 및 인스턴스 정리
- Docker를 통한 배포

## Nginx로 스프링 프로젝트 무중단 배포하기

이전(~#3) 까지 프로젝트에서 git pull만 해주면 ec2 서버에 자동으로 배포해주는 환경까지 구축을 완료했다. 하지만 새로운 Jar가 실행되기 전까지 기존 Jar를 종료시켜 놓기 때문에 새로운 버전을 배포하는 동안 기존 운영되는 애플리케이션이 종료된다는 문제가 남았다. 실제 서비스하는 앱이라면 클라이언트에게 아주 안좋은 UX환경을 제공하게 된다.

### 무중단 배포란?

흔히 사용하는 앱에서는 해킹이나 서버에 큰 오류가 발생하지 않는 한 서비스가 정지되는 경우는 없다. 어떻게 서비스 중단 없이 새로운 버전을 계속 배포할 수 있는지 알아보자.

무중단 배포 방식은 여러가지가 있다. ([참고](#))

- 롤링(Rolling Update) 방식
- 블루 그린(Blue-Green Deployment) 방식
- 카나리(Canary Release) 방식

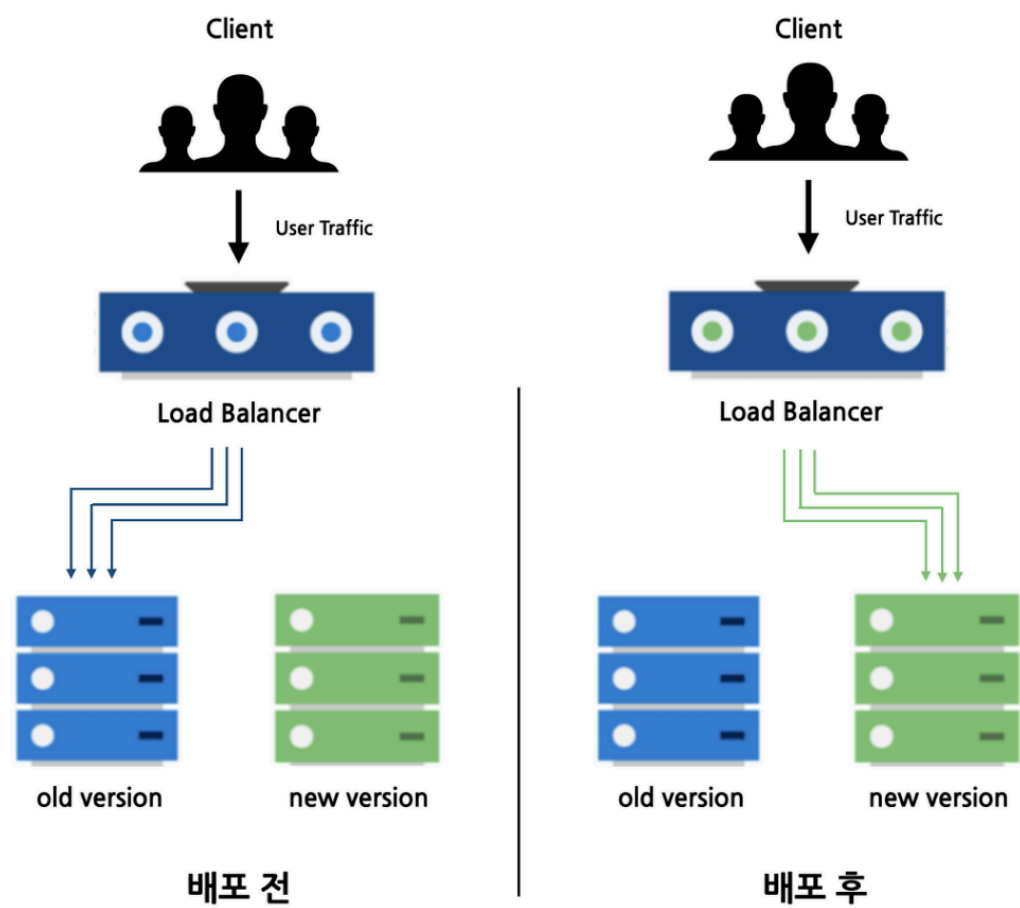
이러한 무중단 배포 방식에는 로드밸런싱을 사용하고있다. 로드밸런싱의 범주는 OSI 7계층 모델을 참조하여 L4와 L7 두 가지 범주로 나뉘어진다. L4는 일반적으로 TCP/UDP 연결/세션 수준에서만 작동하기 때문에 간단하다는 장점이 있다. L7는 L4와 마찬가지로 ip, port를 사용하여 로드밸런싱을 하는 것은 같으나 L7 프로토콜을 통해 헤더를 조작하여 상세한 커스텀을 할 수 있다는 장점이 있다.

- L4와 L7 로드밸런싱 비교

Nginx는 웹 서버, 리버스 프록시, 캐싱, 로드 밸런싱, 미디어 스트리밍 등을 위한 오픈소스 소프트웨어이다. Nginx는 저렴하고 사용법이 간단하기 때문에 많은 인기를 얻었다고 한다. 리버스 프록시란 Nginx가 외부의 요청을 받아 백엔드 서버로 요청을 전달하는 행위인데, 이를 이용해 L7 로드밸런싱 방식으로 무중단 배포 환경을 구축할 수 있다.

## 블루 그린(Blue-Green Deployment) 방식

블루 - 구 버전, 그린 - 신 버전이라 해서 붙여진 이름이다. 운영 환경에서 구 버전과 동일하게 신 버전을 배포하고 일제히 전환하여 모든 연결(트래픽)을 신 버전으로 전환하는 배포 방식이다.



블루 그린(Blue-Green Deployment) 방식

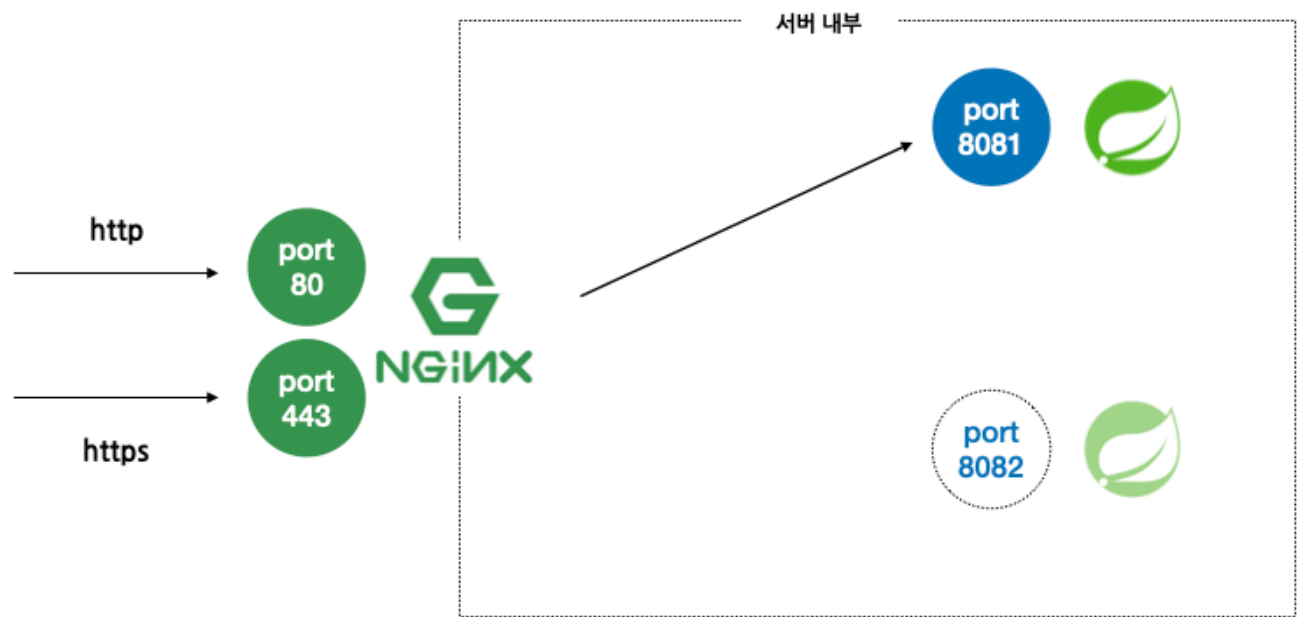
## Nginx로 블루-그린 무중단 배포 방식

EC2 프리티어 1대로 구성한 다음 Port를 2개로 나눠 블루-그린 배포 방식으로 진행할 것이다. 개인 프로젝트이기 때문에 비용 측면에서 이 방식이 효율적이다. 새로운 버전 또한 기존 EC2 인스턴스에 그대로 적용하면 되므로 배포를 위해 AWS EC2 인스턴스가 추가로 더 필요하지 않다. 구조는 **EC2 혹은 리눅스 서버에 Nginx 1대와 스프링 배포 파일 Jar 2대**를 사용하면 되므로 간단하다.

- Nginx에는 80(http), 443(https) 포트를 할당한다.
- 스프링 1에는 8081 포트를 할당한다.
- 스프링 2에는 8082 포트를 할당한다.

## 1. 스프링 버전 1.0.0 무중단 배포

1. 사용자가 서비스에 접속한다. (80 혹은 443 포트)
2. Nginx는 사용자의 요청을 받아 현재 연결된 스프링 1에 요청을 전달한다. (8081로 전달)
3. 스프링 2는 연결된 상태가 아니기 때문에 요청을 받지 못한다.

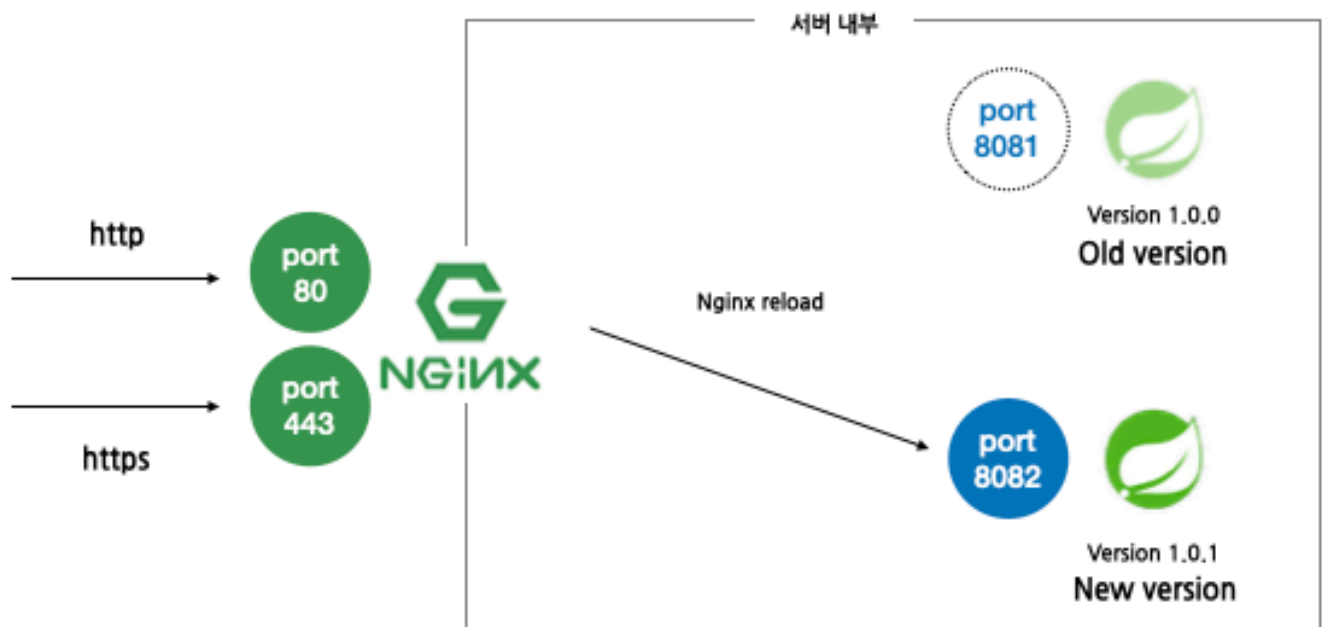


1. 스프링 버전 1.0.0 무중단 배포

## 2. 스프링 버전 1.0.1 무중단 배포

새로운 버전 1.0.1을 신규 배포하면, Nginx와 연결되지 않은 스프링 2로 배포한다.

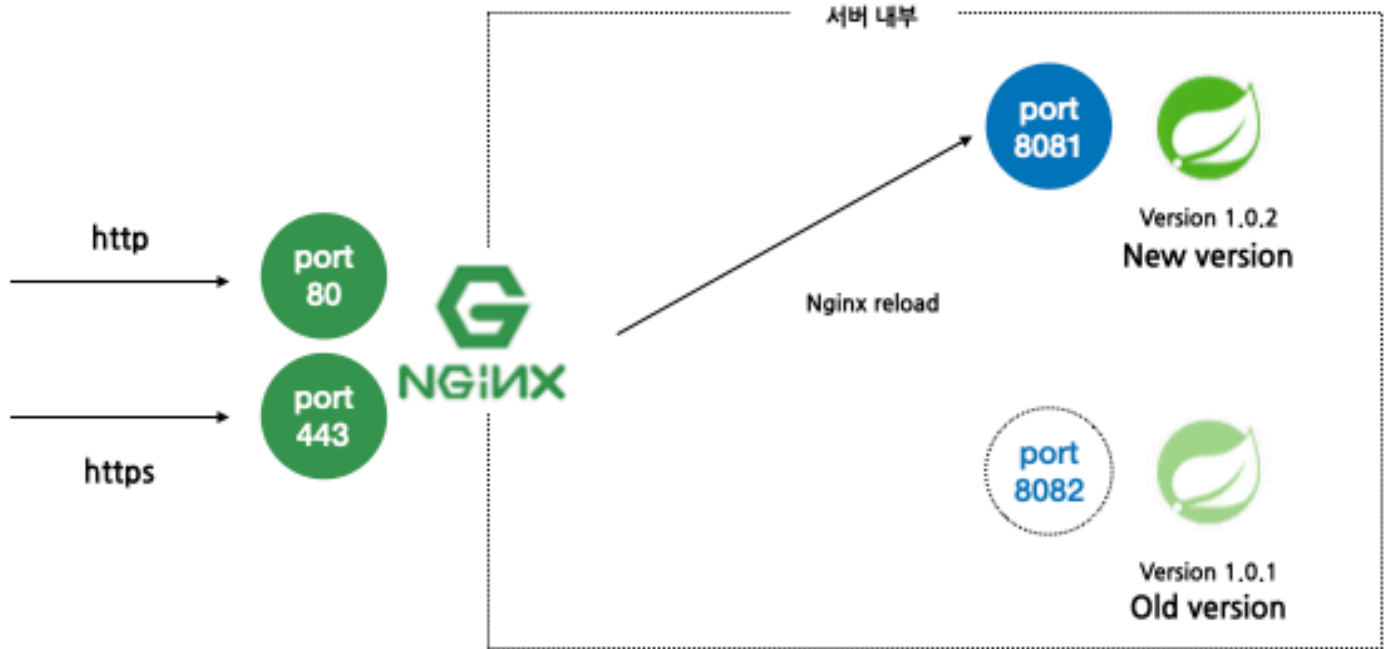
1. 배포하는 동안에도 서비스는 중단되지 않는다. (Nginx는 현재 스프링 1에 트래픽을 전송하고 있다.)
2. 배포가 끝나고 정상적으로 스프링 2가 구동 중인지 확인한다.
3. 스프링 2가 정상 구동 중이면 nginx reload 명령어를 통해 8081에서 8082로 연결을 변경한다.
4. nginx reload는 0.1초 내에 완료되기 때문에 클라이언트는 알아채지 못한다.



### 3. 스프링 버전 1.0.2 무중단 배포

이후 1.0.2 버전 배포가 필요하면 이번에는 스프링 1로 배포한다.

1. 현재 Nginx는 스프링 2와 연결되어 있다.
2. 스프링 1의 배포가 끝나고 나면 Nginx를 Reload하여 스프링 1로 연결을 변경한다.
3. 이후 클라이언트 요청이 오면 스프링 1로 전달한다.



3. 스프링 버전 1.0.2 무중단 배포

## 1. EC2 환경 Nginx 설치 후 실행하기

### 1) EC2에 Nginx 설치

Nginx를 통한 무중단 배포 과정을 모두 살펴봤으니 이제 직접 적용을 해보자.

- `sudo amazon-linux-extras install nginx1`

```
[ec2-user@spring-deploy-test ~]$ sudo amazon-linux-extras install nginx1
Installing nginx
Loaded plugins: extras_suggestions, langpacks, priorities, update-motd
Cleaning repos: amzn2-core amzn2extra-docker amzn2extra-kernel-5.10 amzn2extra-nginx1
17 metadata files removed
6 sqlite files removed
0 metadata files removed
Loaded plugins: extras_suggestions, langpacks, priorities, update-motd
```

EC2에 Nginx 설치

### 2) Nginx 실행

설치가 완료되었으면 Nginx를 실행한다.

- `sudo service nginx start`

```
[ec2-user@spring-deploy-test ~]$ sudo service nginx start
Redirecting to /bin/systemctl start nginx.service
```

Nginx 실행

### 3) EC2 인스턴스 보안그룹에 80번 포트 추가

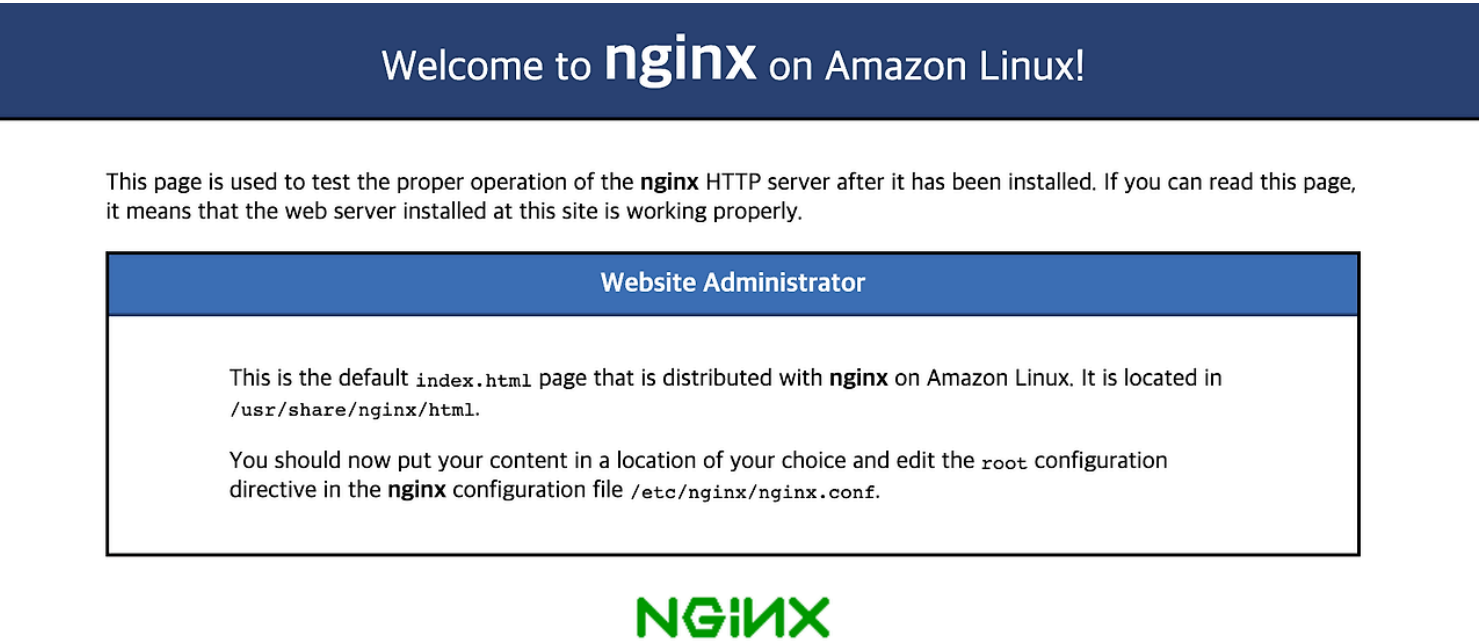
만약 http로 접속이 안된다면 EC2 보안그룹에 80포트를 열어줬는지 확인해보자.

보안 그룹 규칙 ID	유형 <a href="#">정보</a>	프로토콜 <a href="#">정보</a>
sgr-05b0788261a9a3dc9	HTTP ▼	TCP
포트 범위 <a href="#">정보</a>	소스 유형 <a href="#">정보</a>	소스 <a href="#">정보</a>
80	사용자 지정 ▼	<input type="text" value="0.0.0.0/0"/>
<a href="#">설명 - 선택 사항</a> <a href="#">정보</a>		

EC2 인스턴스 보안그룹에 80번 포트 추가

### 4) 80 포트로 http 접속해보기

서버를 구동시킨 후 8080포트 없이 http:EC2 주소로 접속해서 다음과 같은 페이지가 뜨면 정상적으로 동작하는 것이다.



80 포트로 http 접속해보기

## 2. 스프링 프로젝트와 Nginx 연동하기

### 1) OAuth 리다이렉션 주소 추가

8080이 아닌 80포트로 주소가 변경되니 구글, 네이버, 카카오 로그인에도 변경된 주소를 등록해줘야 한다.

## 승인된 리디렉션 URI

웹 서버의 요청에 사용

URI 1 \*

http://localhost:8080/login/oauth2/code/google

URI 2 \*

http://ec2-3-37-60-245.ap-northeast-2.compute.amazonaws.com:8080/login/

URI 3 \*

http://ec2-3-37-60-245.ap-northeast-2.compute.amazonaws.com/login/oauth

+ URI 추가

Note: It may take 5 minutes to a few hours for settings to take effect

저장

취소

OAuth 리다이렉션 주소 추가

## 2) nginx 설정 파일에서 프록시 설정 변경하기

Nginx를 설치하면 설정 파일이 자동으로 생성된다. 해당 설정 파일을 열어서 스프링 프로젝트에 접속할 수 있도록 프록시 설정을 변경해보자.

- `sudo vim /etc/nginx/nginx.conf`

설정 내용 중 server로 블록된 부분을 찾아 다음 설정을 추가하자. 수정을 완료한 후 `:wq` 명령어로 저장하고 종료한다.

```
# Load modular configuration files from the /etc/nginx/conf.d directory.
# See http://nginx.org/en/docs/nginx_core_module.html#include
# for more information.
include /etc/nginx/conf.d/*.conf;

server {
    listen      80;
    listen      [::]:80;
    server_name _;
    root        /usr/share/nginx/html;

    # Load configuration files for the default server block.
    include /etc/nginx/default.d/*.conf;

    location / {
        proxy_pass http://localhost:8080;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header Host $http_host;
    }

    error_page 404 /404.html;
    location = /404.html {
    }
}
```

nginx 설정 추가하기

### proxy\_pass

- 엔진엑스로 요청이 오면 http://localhost:8080로 전달한다.

proxy\_set\_header XXX

- 실제 요청 데이터를 header의 각 항목에 할당한다.
- X-Real-IP \$remote\_addr: Request Header의 X-Real-IP에 요청자의 IP를 저장한다.

3) Nginx 재시작하기

- `sudo service nginx restart`

스프링 Nginx 연동 완료

8080포트 없이 다음과 같이 접속이 성공하면 연동이 완료되었음을 알 수 있다.



스프링 Nginx 연동 완료

3. Profile API 만들기

1) profile API 추가하기

현재 실행 중인 ActiveProfile를 전부 가져와서 real, real1, real2에 포함되는 profile이 있다면 반환하는 Controller를 작성한다. 그러면 브라우저에서 현재 구동중인 profile을 확인할 수 있다.

```
@RestController
@RequiredArgsConstructor
public class ProfileController {
    private final Environment env;

    @GetMapping("/profile")
    public String profile() {
        List<String> profiles = Arrays.asList(env.getActiveProfiles());
        List<String> realProfiles = Arrays.asList("real", "real1", "real2");
        String defaultProfile = profiles.isEmpty() ? "default" : profiles.get(0);

        return profiles.stream().filter(realProfiles::contains).findAny().orElse(defaultProfile);
    }
}
```

2) Spring Security API 접근권한 열기

인증없이 "/profile"에 접근할 수 있게 SecurityConfig 파일에서 다음과 같이 설정을 추가해준다.

```
.antMatchers("/", "/css/**", "/images/**", "/js/**", "/h2-console/**", "/profile").permitAll()
```

### 3) 테스트 코드 작성하기

그리고 간단하게 API가 동작하는지 테스트 코드를 작성해준다. (JUnit4)

- 해당 API는 단순 MockEnvironment로 동작하기 때문에 @SpringBootTest가 필요없다. 그래서 만약 IntelliJ가 해당 테스트를 인지하지 못한다면 설정 → Gradle → Run test using: IntelliJ IDEA로 변경해준다.

```
public class ProfileControllerTest {

    @Test
    public void real_profile_조회(){
        // given
        String expectedProfile = "real";
        MockEnvironment env = new MockEnvironment();
        env.addActiveProfile(expectedProfile);
        env.addActiveProfile("oauth");
        env.addActiveProfile("real-db");

        ProfileController controller = new ProfileController(env);

        // when
        String profile = controller.profile();

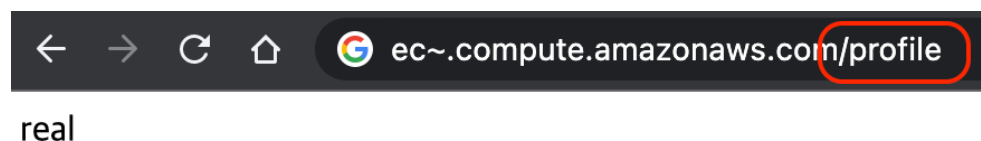
        // then
        assertEquals(profile, expectedProfile);
    }
}
```

### 4) 브라우저에서 profile 조회하기

이제 port로 배포된 스프링 Jar을 구분해줘야 하기 때문에 deploy.sh 파일에서 PID를 찾아주는 코드를 다음과 같이 변경하였다.

```
CURRENT_PID=$(lsof -ti tcp:8080)
```

그 다음 git pull을 하여 자동 배포가 이루어지고 /profile로 조회해보면 "real"이 반환되는 것을 확인할 수 있다.



/profile 조회

## 4. 무중단 배포에 필요한 프로필 생성하기

무중단 배포를 위해서는 포트 번호가 다른 두 스프링 부트가 필요하다. 그러기 위해서 spring profile을 2개 작성해주면 된다.



# 1) real1, real2 프로필 생성하기

참고로 Spring profile 설정 방식은 2.4부터 변경되었다.

- <https://github.com/spring-projects/spring-boot/wiki/Spring-Boot-Config-Data-Migration-Guide#profile-groups>

## 2.4 이전 버전

### application-real1.yml

```
spring:
  profiles:
    include: oauth, real-db
  jpa:
    database-platform: org.hibernate.dialect.MySQL5InnoDBDialect
  session:
    store-type: jdbc

server:
  port: 8081
```

### application-real2.yml

```
spring:
  profiles:
    include: oauth, real-db
  jpa:
    database-platform: org.hibernate.dialect.MySQL5InnoDBDialect
  session:
    store-type: jdbc

server:
  port: 8081
```

## 2.4 버전 이후

### application.yml

```
spring:
  profiles:
    group:
      "real1": "oauth, real-db"
      "real2": "oauth, real-db"
```

### application-real1.yml

```
spring:
  jpa:
    database-platform: org.hibernate.dialect.MySQL5InnoDBDialect
```

```
session:
  store-type: jdbc
```

```
server:
  port: 8081
```

## application-real2.yml

```
spring:
  jpa:
    database-platform: org.hibernate.dialect.MySQL5InnoDBDialect
  session:
    store-type: jdbc
```

```
server:
  port: 8082
```

## 2) Nginx 설정 수정하기

무중단 배포의 핵심은 Nginx 설정이다. 배포 때마다 프록시 설정이 순식간에 교체되도록 만들어야 한다. 엔진엑스 설정이 모여있는 /etc/nginx/conf.d/ 경로에 service-uri.inc라는 파일을 하나 생성한다.

- `sudo vim /etc/nginx/conf.d/service-uri.inc`

그리고 다음 코드를 입력한 후 `:wq` 명령어로 저장한다.

- `set $service_url http://127.0.0.1:8080;`

```
set $service_url http://127.0.0.1:8080;
```

service\_url 설정

그런 다음 nginx.conf 파일에 해당 설정을 사용할 수 있게 설정한다.

- `sudo vim /etc/nginx/nginx.conf`

```
server {
    listen      80;
    listen      [::]:80;
    server_name _;
    root        /usr/share/nginx/html;

    # Load configuration files for the default server block.
    include /etc/nginx/default.d/*.conf;
    include /etc/nginx/conf.d/service-url.inc;

    location / {
        proxy_pass $service_url;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header Host $http_host;
    }
}
```

nginx 설정 추가

Nginx를 재시작한 후 정상적으로 동작하는지 확인한다.

- `sudo nginx -s reload`

브라우저에서 직접 확인해봐도 되고 `$ curl localhost` 명령어로 확인해봐도 된다.

## 5. 무중단 배포 스크립트 작성하기

### 1) EC2에 step3 디렉토리 생성

먼저 step2와 중복되지 않기 위해 ec2에 step3 디렉토리를 생성한다.

- `mkdir ~/app/step3 && mkdir ~/app/step3/zip`

### 2) appspec.yml 설정 변경

무중단 배포는 앞으로 step3를 사용할 것이기 때문에 기존에 step2로 설정해둔 것을 변경해준다.

```
version: 0.0
os: linux
files:
  - source: / # CodeDeploy에서 전달해 준 파일 중 destination으로 이동시킬 대상 지정 ('/' 루트 파일은 전체파일을 의미)
    destination: /home/ec2-user/app/step3/zip/ # source에서 지정된 파일을 받을 위치
    overwrite: yes # 기존 파일 덮어쓰기 허용
```

### 3) 무중단 배포 스크립트 작성

무중단 배포를 위해 필요한 스크립트는 총 5개이다.

(스크립트는 컴파일 시점에서 오타를 잡을 수 없기 때문에 작성에 유의하자. 오타 때문에 런타임 시점에 에러가 잡혀 시간을 꽤나 잡아먹었다...)

- 1. **stop.sh** : 기존 Nginx에 연결되어 있지 않지만, 실행 중인 스프링을 종료해준다.
- 2. **start.sh** : 배포한 신규 버전 스프링 프로젝트를 stop.sh로 종료한 profile로 실행한다.
- 3. **health.sh** : 'start.sh'로 실행시킨 프로젝트가 정상적으로 동작하는지 체크해준다.
- 4. **switch.sh** : Nginx가 바라보는 스프링을 최신 버전으로 변경해준다.
- 5. **profile.sh** : 앞선 4개 스크립트 파일에서 공용으로 사용할 'profile'과 'port'를 동적으로 체크하는 로직을 작성한다.

위의 스크립트를 제대로 사용할 수 있도록 AWS CodeDeploy를 설정해주는 appspec.yml로 수정한다.

- Jar 파일 복사된 이후부터 차례로 앞선 스크립트들이 실행된다고 보면 된다. (stop.sh → start.sh → health.sh)

```
hooks:
  AfterInstall:
    - location: stop.sh # Nginx와 연결되어 있지 않은 스프링부트를 종료한다.
      timeout: 60
      runas: ec2-user
  ApplicationStart:
    - location: start.sh # Nginx와 연결되어 있지 않은 Port로 새 버전의 스프링부트를 시작한다.
      timeout: 60
      runas: ec2-user
  ValidateService:
    - location: health.sh # 새 스프링 부트가 정상적으로 실행됐는지 확인한다.
      timeout: 60
      runas: ec2-user
```

profile.sh

현재 쉬고 있는 profile을 캐치하여 그에 해당하는 port를 출력해주는 스크립트이다.

- **find\_idle\_profile()**: 쉬고 있는 profile을 찾는다.
  - 4xx, 5xx에러가 발생하면 real2로 현재 프로필을 변경한다.
  - IDLE\_PROFILE에 동작중이지 않은 profile을 저장해서 마지막에 echo로 출력한다.
- **find\_idle\_port()**: 쉬고 있는 profile의 port로 출력한다.
  - find\_idle\_profile()에서 출력한 IDLE\_PROFILE 값을 캐치해서 입력받는다.
  - 그리고 그에 따른 port 번호를 입력받아 echo로 출력한다.

```
#!/bin/bash

# 쉬고 있는 profile 찾기: real1이 사용 중이면 real2가 쉼. 반대로, real2가 사용 중이면 real1이 쉼.
function find_idle_profile() {
  RESPONSE_CODE=$(curl -s -o /dev/null -w "%{http_code}" http://localhost/profile) # Nginx와 연결되어 있는 스프링 부트가 정상
작동 중인지 확인

  if [ ${RESPONSE_CODE} -ge 400 ] # 400보다 크면(즉, 40x, 50x 에러 모두 포함)
  then
    CURRENT_PROFILE=real2
  else
    CURRENT_PROFILE=$(curl -s http://localhost/profile)
  fi

  # 연결되지 않은 profile 저장
  if [ ${CURRENT_PROFILE} == real1 ]
  then
    IDLE_PROFILE=real2
  else
    IDLE_PROFILE=real1
  fi

  echo "${IDLE_PROFILE}" # IDLE_PROFILE 출력. 스크립트는 값을 반환하는 기능이 없어서 마지막 줄 echo로 출력 후 그 값을 캐치하는
```

```
식으로 전송한다. ($(find_idle_profile))
}
```

```
# 쉬고 있는 profile의 port 찾기
function find_idle_port() {
    IDLE_PROFILE=$(find_idle_profile)

    if [ ${IDLE_PROFILE} == real1 ]
    then
        echo "8081"
    else
        echo "8082"
    fi
}
```

## stop.sh

profile.sh에서 선언한 함수를 불러와서 현재 Nginx와 연결되어 있지 않지만 실행 중인 스프링 부트를 종료시켜준다.

```
#!/bin/bash

ABSPATH=$(readlink -f $0)
ABSDIR=$(dirname $ABSPATH) # 현재 stop.sh에 속해 있는 경로
source ${ABSDIR}/profile.sh # 자바 import와 비슷한 기능. 해당 코드로 인해 profile.sh의 여러 함수를 사용할 수 있다.

IDLE_PORT=$(find_idle_port)

echo "> $IDLE_PORT 에서 구동 중인 애플리케이션 pid 확인"
IDLE_PID=$(lsof -ti tcp:${IDLE_PORT})

if[ -z ${IDLE_PID} ]
then
    echo "> 현재 구동 중인 애플리케이션이 없으므로 종료하지 않습니다."
else
    echo "> kill -15 $IDLE_PID"
    kill -15 ${IDLE_PID}
    sleep 5
fi
```

## start.sh

배포한 신규 버전 스프링 프로젝트를 stop.sh로 종료한 idle\_profile로 실행한다.

```
#!/bin/bash

ABSPATH=$(readlink -f $0)
ABSDIR=$(dirname $ABSPATH)
source ${ABSDIR}/profile.sh # 여기서도 profile.sh 함수를 사용

REPOSITORY=/home/ec2-user/app/step3
PROJECT_NAME=spring_deploy_test

echo "> Build 파일 복사"
echo "> cp $REPOSITORY/zip/*.jar $REPOSITORY/"

cp $REPOSITORY/zip/*.jar $REPOSITORY/

echo "> 새 애플리케이션 배포"
JAR_NAME=$(ls -tr $REPOSITORY/*.jar | tail -n 1)
```

```
echo "> JAR Name: $JAR_NAME"

echo "> $JAR_NAME 에 실행권한 추가"

chmod +x $JAR_NAME

echo "> $JAR_NAME 실행"

IDLE_PROFILE=$(find_idle_profile)

echo "> $JAR_NAME 를 profile=$IDLE_PROFILE로 실행합니다."
nohup java -jar W
    -Dspring.config.location=classpath:/application.yml,classpath:/application-$IDLE_PROFILE.yml,/home/ec2-user/app/application-
oauth.yml,/home/ec2-user/app/application-real-db.yml W
    -Dspring.profiles.active=$IDLE_PROFILE W
$JAR_NAME > $REPOSITORY/nohup.out 2>&1 &
```

health.sh

'start.sh'로 실행한 후 바로 해당 프로젝트가 정상적으로 동작하는지 Nginx와 연결되어 있지 않은 포트(IDLE\_PORT)로 스프링 부트가 정상적으로 동작하는지 체크한다.

- 정상적으로 동작하면("real" 문자열이 있는지 검증) 해당 스크립트를 종료한다.
- 만약 정상적으로 동작하지 않으면 최대 10번 재실행해보고 그래도 연결되지 않는다면 배포를 종료한다.
- Nginx 프록시 변경은 switch.sh에서 수행한다.

```
#!/bin/bash

ABSPATH=$(readlink -f $0)
ABSDIR=$(dirname $ABSPATH)
source ${ABSDIR}/profile.sh
source ${ABSDIR}/switch.sh

IDLE_PORT=$(find_idle_port)

echo "> Health Check Start"
echo "> IDLE_PORT: $IDLE_PORT"
echo "> curl -s http://localhost:$IDLE_PORT/profile"
sleep 10

for RETRY_COUNT in {1..10}
do
    RESPONSE=$(curl -s http://localhost:${IDLE_PORT}/profile)
    UP_COUNT=$(echo ${RESPONSE} | grep 'real' | wc -l)

    if [ ${UP_COUNT} -ge 1 ]
    then # $up_count >= 1 "real" 문자열이 있는지 검증
        echo "> Health check 성공"
        switch_proxy
        break
    else
        echo "> Health check의 응답을 알 수 없거나 혹은 실행 상태가 아닙니다."
        echo "> Health check: ${RESPONSE}"
    fi

    if [ ${RETRY_COUNT} -eq 10 ]
    then
        echo "> Health check 실패."
        echo "> Nginx에 연결하지 않고 배포를 종료합니다."
        exit 1
    fi
```

```
echo "> Health check 연결 실패. 재시도 ..."
sleep 10
done
```

## switch.sh

'health.sh'에서 정상적으로 동작이 확인되었다면 Nginx에 설정되어 있는 port번호를 바꿔준다. 현재 IDLE\_PORT가 새로운 버전을 배포할 port이기 때문에 해당 port를 service-url.inc파일에 넣어준다.

```
#!/bin/bash

ABSPATH=$(readlink -f $0)
ABSDIR=$(dirname $ABSPATH)
source ${ABSDIR}/profile.sh

function switch_proxy() {
    IDLE_PORT=$(find_idle_port)

    echo "> 전환할 port: $IDLE_PORT"
    echo "> port 전환"
    # 하나의 문장을 만들어 파이프라인으로 넘겨주기 위해 echo를 사용
    echo "set W$service_url http://127.0.0.1:${IDLE_PORT};" | sudo tee /etc/nginx/conf.d/service-url.inc


    echo "> Nginx Reload"
    sudo nginx -s reload
}
```

## 6. 무중단 배포 테스트하기

배포 테스트를 하기 전 마지막으로 Jar파일명 중복 방지를 위해 설정할 것이 하나 남았다. build.gradle파일에 가서 버전 이름을 다음과 같이 수정해준다.

```
version '1.0.2-SNAPSHOT-'+new Date().format("yyyyMMddHHmmss")
```

CodeDeploy에서 배포가 완료되었으면 상태를 확인해보자.

	배포 ID	상태	배포 유형	컴퓨팅 플랫폼	애플리케이션
	d-E72EE9TPF	✔ 성공	현재 위치	EC2/온프레미스	spring-deploy-test

배포 완료

EC2에서 다음과 같은 명령어로 CodeDeploy 로그로 잘 진행되는지 확인할 수 있다.

- `tail -f /opt/codedeploy-agent/deployment-root/deployment-logs/codedeploy-agent-deployments.log`



```
[2022-03-23 00:40:13.433] [d-E72EE9TPF]LifecycleEvent - AfterInstall
[2022-03-23 00:40:13.433] [d-E72EE9TPF]Script - stop.sh
[2022-03-23 00:40:13.454] [d-E72EE9TPF][stdout]> 8081 에서 구동 중인 애플리케이션 pid 확인
[2022-03-23 00:40:13.491] [d-E72EE9TPF][stdout]> 현재 구동 중인 애플리케이션이 없으므로 종료하지 않습니다.
[2022-03-23 00:40:14.478] [d-E72EE9TPF]LifecycleEvent - ApplicationStart
[2022-03-23 00:40:14.478] [d-E72EE9TPF]Script - start.sh
[2022-03-23 00:40:14.492] [d-E72EE9TPF][stdout]> Build 파일 복사
[2022-03-23 00:40:14.492] [d-E72EE9TPF][stdout]> cp /home/ec2-user/app/step3/zip/*.jar /home/ec2-user/app/step3/
[2022-03-23 00:40:14.515] [d-E72EE9TPF][stdout]> 새 애플리케이션 배포
[2022-03-23 00:40:14.518] [d-E72EE9TPF][stdout]> JAR Name: /home/ec2-user/app/step3/spring-deploy-test-1.0.3-SNAPSHOT-20220322
153904.jar
[2022-03-23 00:40:14.518] [d-E72EE9TPF][stdout]> /home/ec2-user/app/step3/spring-deploy-test-1.0.3-SNAPSHOT-20220322153904.jar
에 실행 권한 추가
[2022-03-23 00:40:14.519] [d-E72EE9TPF][stdout]> /home/ec2-user/app/step3/spring-deploy-test-1.0.3-SNAPSHOT-20220322153904.jar
실행
[2022-03-23 00:40:14.526] [d-E72EE9TPF][stdout]> /home/ec2-user/app/step3/spring-deploy-test-1.0.3-SNAPSHOT-20220322153904.jar
를 profile=real1로 실행합니다.
[2022-03-23 00:40:15.754] [d-E72EE9TPF]LifecycleEvent - ValidateService
[2022-03-23 00:40:15.755] [d-E72EE9TPF]Script - health.sh
[2022-03-23 00:40:15.878] [d-E72EE9TPF][stdout]> Health Check Start
[2022-03-23 00:40:15.878] [d-E72EE9TPF][stdout]> IDLE_PORT: 8081
[2022-03-23 00:40:15.878] [d-E72EE9TPF][stdout]> curl -s http://localhost:8081/profile
[2022-03-23 00:40:25.916] [d-E72EE9TPF][stdout]> Health check의 응답을 알 수 없거나 혹은 실행 상태가 아닙니다.
[2022-03-23 00:40:25.916] [d-E72EE9TPF][stdout]> Health check:
[2022-03-23 00:40:25.916] [d-E72EE9TPF][stdout]> Health check 연결 실패. 재시도 ...
[2022-03-23 00:40:36.230] [d-E72EE9TPF][stdout]> Health check 성공
[2022-03-23 00:40:36.251] [d-E72EE9TPF][stdout]> 전환할 port: 8081
[2022-03-23 00:40:36.251] [d-E72EE9TPF][stdout]> port 전환
[2022-03-23 00:40:36.275] [d-E72EE9TPF][stdout]set $service_url http://127.0.0.1:8081;
[2022-03-23 00:40:36.276] [d-E72EE9TPF][stdout]> Nginx Reload
```

CodeDeploy 로그 확인

스프링 부트 로고는 nohup.out파일에서 확인해 볼 수 있다.

- `cat ~/app/step3/nohup.out`

브라우저를 확인하면 다음과 같다.

## 스프링 부트로 시작하는 웹 서비스 ver.1.0.3

글 등록

Google Login

Naver Login

Kakao Login

게시글번호	제목	작성자	내용	최종수정일
-------	----	-----	----	-------

브라우저

여기서 한 번 더 배포하면 real2로 배포된다. 이 과정에서 브라우저 새로고침을 해도 전혀 중단이 없는 것을 확인할 수 있다. 새로운 버전 배포를 한 번 더 진행한 뒤에 다음과 같이 자바 애플리케이션 실행 여부를 확인해보자. 그러면 다음과 같이 현재 쉬고있는 8082번 포트에 연결하는 것을 볼 수 있다.



```

[2022-03-23 01:25:40.413] [d-ERMQYYUPF]LifecycleEvent - AfterInstall
[2022-03-23 01:25:40.414] [d-ERMQYYUPF]Script - stop.sh
[2022-03-23 01:25:40.451] [d-ERMQYYUPF][stdout]> 8082 에서 구동 중인 애플리케이션 pid 확인
[2022-03-23 01:25:40.488] [d-ERMQYYUPF][stdout]> 현재 구동 중인 애플리케이션이 없으므로 종료하지 않습니다.
[2022-03-23 01:25:41.675] [d-ERMQYYUPF]LifecycleEvent - ApplicationStart
[2022-03-23 01:25:41.676] [d-ERMQYYUPF]Script - start.sh
[2022-03-23 01:25:41.690] [d-ERMQYYUPF][stdout]> Build 파일 복사
[2022-03-23 01:25:41.690] [d-ERMQYYUPF][stdout]> cp /home/ec2-user/app/step3/zip/*.jar /home/ec2-user/app/step3/
[2022-03-23 01:25:41.728] [d-ERMQYYUPF][stdout]> 새 애플리케이션 배포
[2022-03-23 01:25:41.731] [d-ERMQYYUPF][stdout]> JAR Name: /home/ec2-user/app/step3/spring-deploy-test-1.0.4-SNAPSHOT-20220322162434.jar
[2022-03-23 01:25:41.731] [d-ERMQYYUPF][stdout]> /home/ec2-user/app/step3/spring-deploy-test-1.0.4-SNAPSHOT-20220322162434.jar
에 실행 권한 추가
[2022-03-23 01:25:41.732] [d-ERMQYYUPF][stdout]> /home/ec2-user/app/step3/spring-deploy-test-1.0.4-SNAPSHOT-20220322162434.jar
실행
[2022-03-23 01:25:41.768] [d-ERMQYYUPF][stdout]> /home/ec2-user/app/step3/spring-deploy-test-1.0.4-SNAPSHOT-20220322162434.jar
를 profile=real2로 실행합니다.
[2022-03-23 01:25:42.809] [d-ERMQYYUPF]LifecycleEvent - ValidateService
[2022-03-23 01:25:42.810] [d-ERMQYYUPF]Script - health.sh
[2022-03-23 01:25:42.966] [d-ERMQYYUPF][stdout]> Health Check Start
[2022-03-23 01:25:42.966] [d-ERMQYYUPF][stdout]> IDLE_PORT: 8082
[2022-03-23 01:25:42.966] [d-ERMQYYUPF][stdout]> curl -s http://localhost:8082/profile
[2022-03-23 01:25:53.010] [d-ERMQYYUPF][stdout]> Health check의 응답을 알 수 없거나 혹은 실행 상태가 아닙니다.
[2022-03-23 01:25:53.010] [d-ERMQYYUPF][stdout]> Health check:
[2022-03-23 01:25:53.010] [d-ERMQYYUPF][stdout]> Health check 연결 실패. 재시도 ...
[2022-03-23 01:26:03.262] [d-ERMQYYUPF][stdout]> Health check 성공
[2022-03-23 01:26:03.311] [d-ERMQYYUPF][stdout]> 전환할 port: 8082
[2022-03-23 01:26:03.311] [d-ERMQYYUPF][stdout]> port 전환
[2022-03-23 01:26:03.333] [d-ERMQYYUPF][stdout]set $service_url http://127.0.0.1:8082;
[2022-03-23 01:26:03.335] [d-ERMQYYUPF][stdout]> Nginx Reload

```

CodeDeploy 로그 확인

브라우저를 새로고침해도 종료없이 무중단 배포됨을 확인할 수 있다. 이렇게 무중단 배포까지 완료했다.

## 스프링 부트로 시작하는 웹 서비스 ver.1.0.4

글 등록

Google Login

Naver Login

Kakao Login

게시글번호	제목	작성자	내용	최종수정일
-------	----	-----	----	-------

현재 실행되고 있는 java 파일을 검색해보면 다음과 같이 두 개의 버전이 실행중임을 알 수 있다. 여기서 또 한 번 배포하면 1.0.3을 배포하고 있는 8081포트를 kill하고 해당 포트로 새로운 버전을 무중단 배포해줄 것이다.

- `ps -ef | grep java`

```

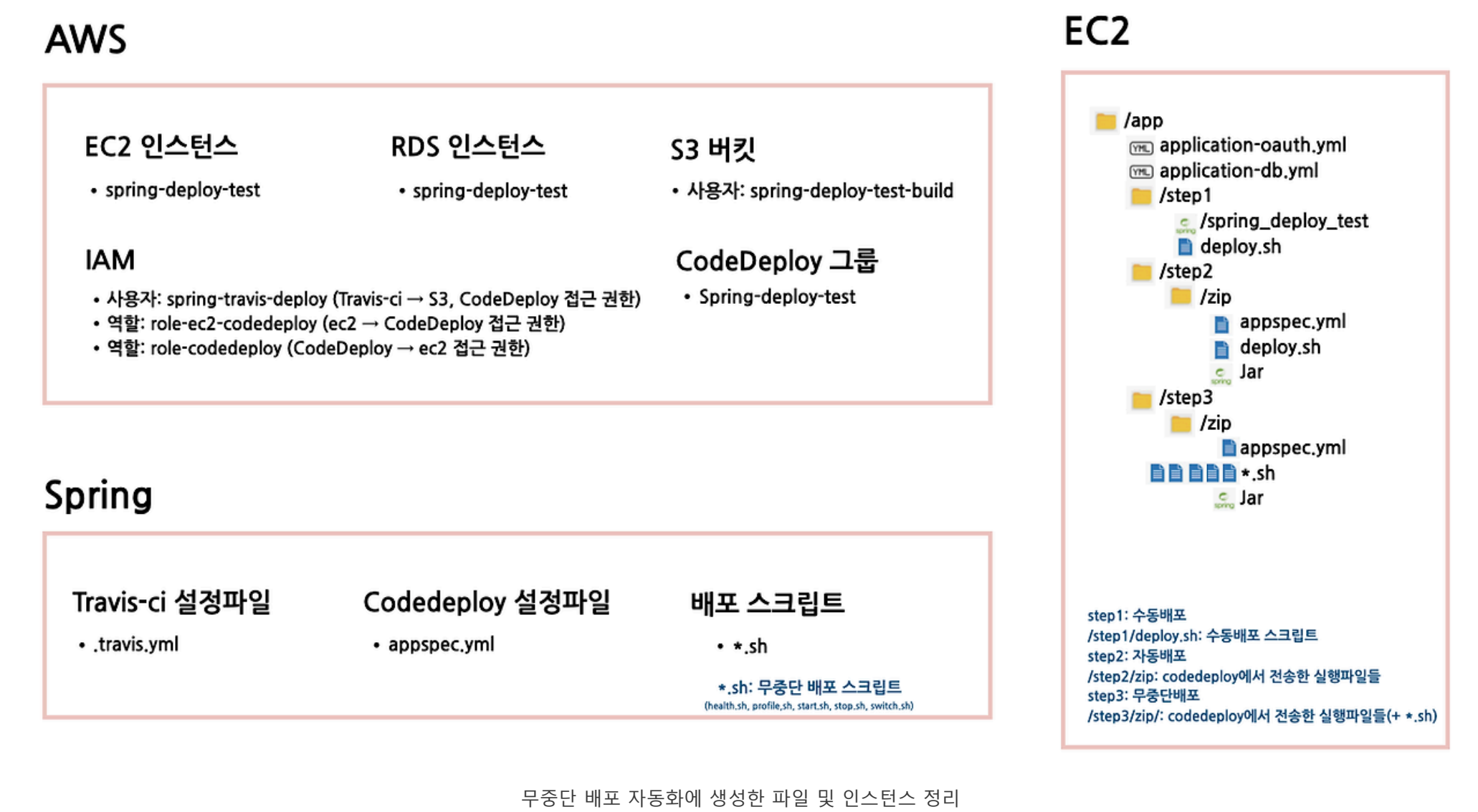
[ec2-user@spring-deploy-test step3]$ ps -ef | grep java
ec2-user 10254      1   0 01:25 ?        00:00:22 java -jar -Dspring.config.location=classpath:/application.yml,classpath:/application-real2.yml,/home/ec2-user/app/application-oauth.yml,/home/ec2-user/app/application-real-db.yml -Dspring.profiles.active=real2 /home/ec2-user/app/step3/spring-deploy-test-1.0.4-SNAPSHOT-20220322162434.jar
ec2-user 17948      1   0 00:40 ?        00:00:25 java -jar -Dspring.config.location=classpath:/application.yml,classpath:/application-real1.yml,/home/ec2-user/app/application-oauth.yml,/home/ec2-user/app/application-real-db.yml -Dspring.profiles.active=real1 /home/ec2-user/app/step3/spring-deploy-test-1.0.3-SNAPSHOT-20220322153904.jar

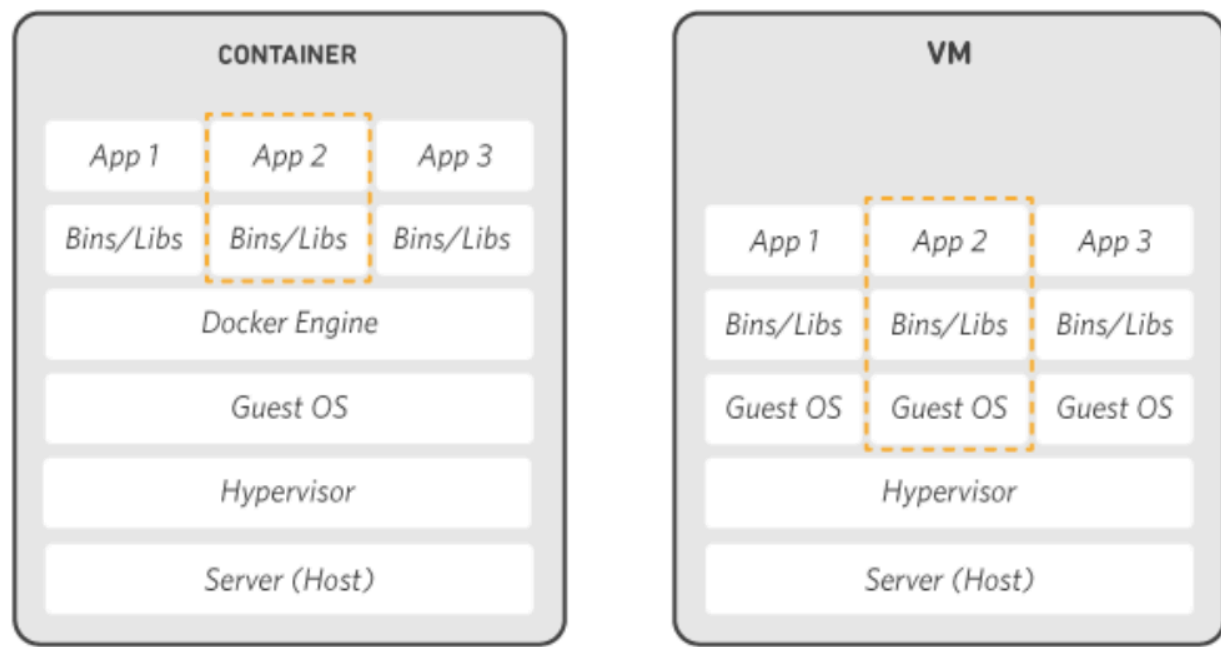
```

현재 실행중인 jar 파일들

## 파일 및 인스턴스 정리

현재까지 AWS 환경과 EC2(Linux), Spring 환경에 생성된 인스턴스 및 파일 구조는 다음과 같다.





컨테이너와 VM