1) 스프링 프로젝트 실제 배포 자동화하기

모든 파일(*.zip) → 실제 배포 파일(jar, appspec.yml, 배포 스크립트)로 구성해서 배포

1. 스프링 프로젝트에 배포 스크립트 작성(/scripts/deploy.sh)
2. .travis.yml 파일 설정 변경 (before-deploy : 배포 파일 구성 변경)
3. appspec.yml 파일에 설정 추가 (permissions, hooks)
4. Git push 실제 배포 과정 테스트 : ver1.0.0 → ver1.0.1
5. EC2에서 CodeDeploy 로그 확인해보기

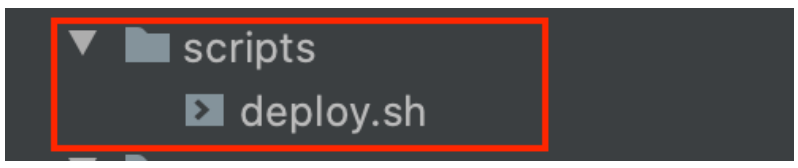
스프링 프로젝트 실제 배포 자동화하기

스프링 프로젝트 실제 배포 자동화하기

앞의 과정으로 Travis CI, S3, CodeDeploy 연동까지 구현되었다. 연동하는 과정만 빠르게 체크하기 위해서 EC2에 보낸 파일은 스프링 프로젝트 전체를 묶어서 보냈다. 이제 그 파일을 실제 배포에 필요한 파일들로만 구성해서 보내주면 된다.

실제 배포 파일 (jar + 기타 설정 파일)을 보내서 배포한다. 그 후 실제로 변경사항을 Git으로 푸시하면 그대로 변경사항이 반영되어 자동 배포되는 과정을 확인할 수 있다.

1. 스프링 프로젝트에 배포 스크립트 생성



deploy.sh

먼저 step2 환경에서 실행될 deploy.sh를 생성한다. scripts 디렉토리를 생성해서 여기에 스크립트를 작성한다.

```
#!/bin/bash

REPOSITORY=/home/ec2-user/app/step2
PROJECT_NAME=spring_deploy_test

echo "> Build 파일 복사"

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

echo "> 현재 구동 중인 애플리케이션 pid 확인"
```

```
CURRENT_PID=$(pgrep -fl spring_deploy_test | grep jar | awk '{print $1}')

echo "현재 구동 중인 애플리케이션 pid : $CURRENT_PID"

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

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"

nohup java -jar ₩
  -Dspring.config.location=classpath:/application.yml,classpath:/application-real.yml,/home/ec2-user/app/application-oauth.ym
l,/home/ec2-user/app/application-real-db.yml ₩
  -Dspring.profiles.active=real ₩
  $JAR_NAME > $REPOSITORY/nohup.out 2>&1 &
```

CURRENT_PID

- 현재 수행 중인 스프링 부트 애플리케이션의 프로세스 ID를 찾는다.
- 실행 중이면 종료하기 위해서이다.
- 프로젝트 이름으로만 검색하면 다른 프로그램이 있을 수 있어 jar 프로세스를 찾은 뒤 ID를 찾는다.

chmod +x $JAR_NAME

- Jar 파일은 실행 권한이 없는 상태이다.
- nohup으로 실행할 수 있게 실행 권한을 부여한다.

$JAR_NAME > $REPOSITORY/nohup.out 2>&1 &

- nohup 실행 시 CodeDeploy는 무한 대기한다.
- 이 이슈를 해결하기 위해 nohup.out 파일을 표준 입출력용으로 별도로 사용한다.
- 이렇게 하지 않으면 nohup.out파일이 생기지 않고, **CodeDeploy 로그에 표준 입출력이 출력**된다.
- nohup이 끝나기 전까지 CodeDeploy도 끝나지 않으니 꼭 이렇게 해야 한다.

2. .travis.yml 파일 설정 변경

현재는 프로젝트의 모든 파일을 zip 파일로 만드는데, 실제로 필요한 파일들은 **(Jar, appspec.yml, 배포를 위한 스크립트)**들이다. 이 외 나머지는 배포에 필요하지 않으니 포함하지 않겠다. 그래서 .travis.yml 파일의 **before_deploy를 수정**한다.

```
language: java
jdk:
  - openjdk11

branches:
  only:
    - main

# Travis CI 서버의 Home
cache:
  directories:
    - '$HOME/.m2/repository'
    - '$HOME/.gradle'

script: "./gradlew clean build"

# CI 실행 완료 시 메일로 알림
notifications:
  email:
    recipients:
      - jong9712@naver.com

########################## 새롭게 추가한 코드 ###################################

# deploy 전에 수행 : jar+(기타 설정 파일) .zip으로 압축
before_deploy:
  - mkdir -p before-deploy # zip에 포함시킬 파일들을 담을 디렉토리 생성
  - cp scripts/*.sh before-deploy/ # scripts파일 추가
  - cp appspec.yml before-deploy/ # appsepc.yml 추가
  - cp build/libs/*.jar before-deploy/ # build/libs 목록 추가
  - cd before-deploy && zip -r before-deploy * # before-deploy로 이동 후 전체 압축
  - cd ../ && mkdir -p deploy # 상위 디렉토리 이동 후 deploy 디렉토리 생성
  - mv before-deploy/before-deploy.zip deploy/spring_deploy_test.zip # deploy로 zip파일 이동

########################################################################################

## (S3로 파일업로드 | CodeDeploy 배포) 외부 서비스와 연동될 행위들을 선언
deploy:
  # S3로 파일업로드
  - provider: s3
    access_key_id: $AWS_ACCESS_KEY # Travis repo settings에 설정된 값
    secret_access_key: $AWS_SECRET_KEY # Travis repos settings에 설정된 값
    bucket: spring-deploy-test-build # s3 버킷
    region: ap-northeast-2
    skip_cleanup: true
    acl: private # zip 파일 접근을 private으로
    local_dir: deploy # before_deploy에서 생성한 디렉토리
    wait-until-deployed: true
    on:
      all_branches: true # master말고 다른 모든 브런치 허용
  # CodeDeploy 배포
  - provider: codedeploy
    access_key_id: $AWS_ACCESS_KEY # Travis repo settings에 설정된 값
    secret_access_key: $AWS_SECRET_KEY # Travis repos settings에 설정된 값
    bucket: spring-deploy-test-build # s3 버킷
    key: spring_deploy_test.zip # 빌드 파일 압축해서 전달
    bundle_type: zip # 압축 확장자
    application: spring-deploy-test # 웹 콘솔에서 등록한 CodeDelpoy 애플리케이션 이름
    deployment_group: spring-deploy-test-group # 웹 콘솔에서 등록한 CodeDelpoy 배포 그룹 이름
    region: ap-northeast-2
    wait-until-deployed: true
    on:
      all_branches: true # master말고 다른 모든 브런치 허용
```

Travis CI는 S3로 특정 파일만 업로드가 안된다.

- 디렉토리 단위로만 업로드할 수 있기 때문에 before-deploy 디렉토리는 항상 생성한다.

before-deploy에는 zip 파일에 포함시킬 파일들을 저장한다.

zip -r 명령어를 통해 before-deploy 디렉토리 전체 파일을 압축한다.

3. appspec.yml 파일에 설정 추가

마지막으로 CodeDeploy의 명령을 담당할 **appspec.yml 파일을 수정**한다.

```
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/step2/zip/ # source에서 지정된 파일을 받을 위치
    overwrite: yes # 기존 파일 덮어쓰기 허용

######## 새롭게 추가한 코드 ########

permissions:
  - object: /
    pattern: "**"
    owner: ec2-user
    group: ec2-user

hooks:
  ApplicationStart:
    - location: deploy.sh
      timeout: 60
      runas: ec2-user

################################
```

permissions

- CodeDeploy에서 EC2 서버로 넘겨준 파일들을 모두 ec2-user 권한을 갖도록 한다.

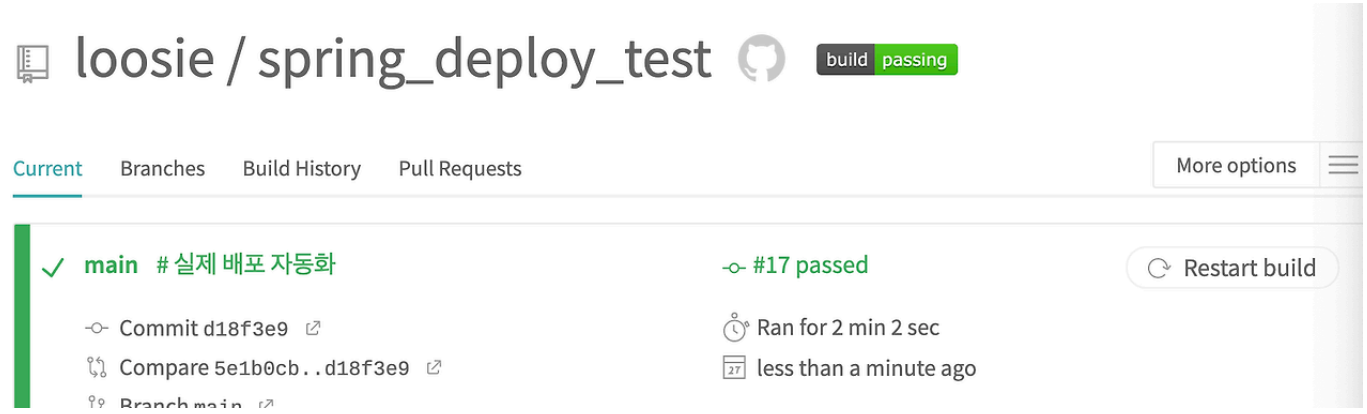
hooks

- CodeDeploy 배포 단계에서 실행할 명령어를 지정한다.
- ApplicationStart라는 단계에서 deploy.sh를 ec2-user 권한으로 실행하게 한다.
- timeout: 60으로 스크립트 실행 60초 이상 수행되면 실패가 된다. (무한정 대기할 수 없으므로 시간 제한을 둔다.)

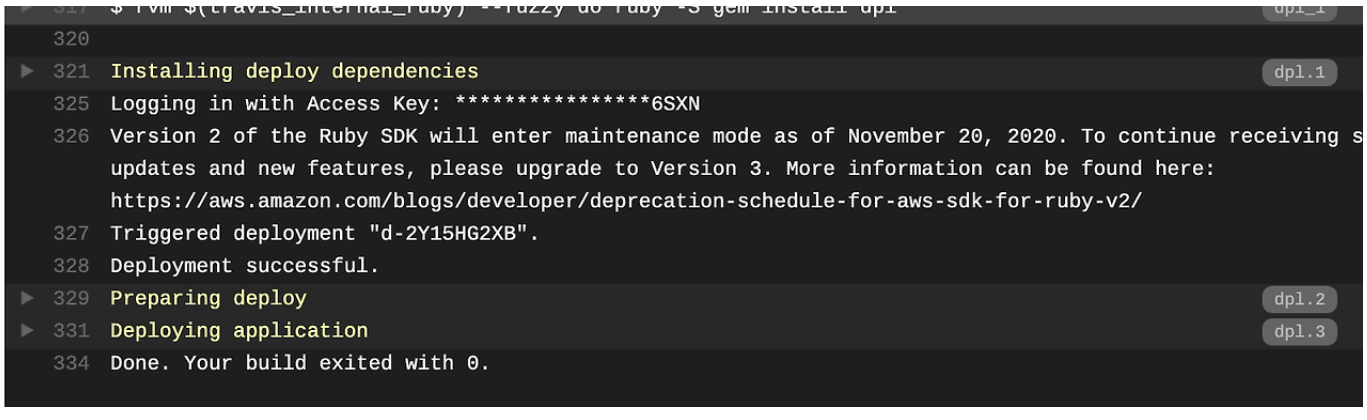
4. Git push 실제 배포 과정 테스트

Travis CI 성공 메시지 확인

모든 설정이 완료되었으니 깃헙으로 commit과 push를 진행한다. **Travis CI에서 다음과 같이 성공 메시지를 확인**한다.



Travs CI 빌드 성공



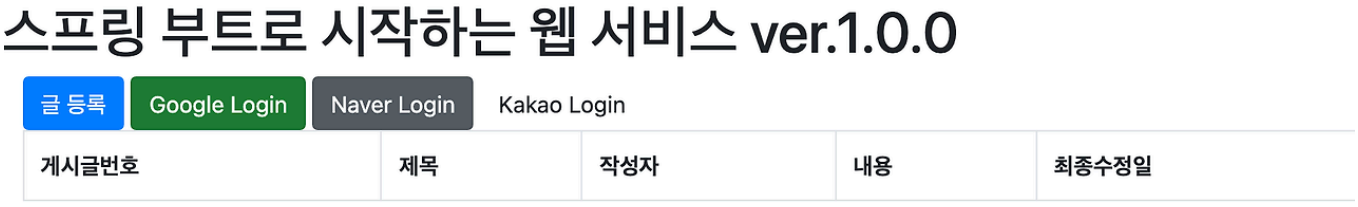
Travis CI 빌드 성공 로그 마지막 화면

CodeDeploy 배포 확인

그런 다음 **CodeDeploy**에서도 배포가 성공한 것을 확인한다.

CodeDeploy 배포 확인

웹 브라우저에서 EC2 도메인을 입력해서 확인해 본다.



웹 브라우저

실제 배포 과정 진행해보기

마지막으로 실제 배포하듯이 진행해보자. **build.gradle에서 프로젝트 버전을 변경**한다.

```
version '1.0.1-SNAPSHOT'
```

간단하게나마 변경된 내용을 알 수 있게 src/main/resources/templates/index.mustache 내용에 다음과 같이 Ver.1.0.1 텍스트를 추가한다.

```
<h1>스프링 부트로 시작하는 웹 서비스 ver.1.0.1</h1>
```

그리고 깃헙으로 commit과 push를 한다. 그럼 다음과 같이 변경된 코드가 배포된 것을 확인할 수 있다.

ver1.0.1 배포

혹시나 변경이 안된다면 ec2 hostname과 project name이 일치하는지 확인해보자.

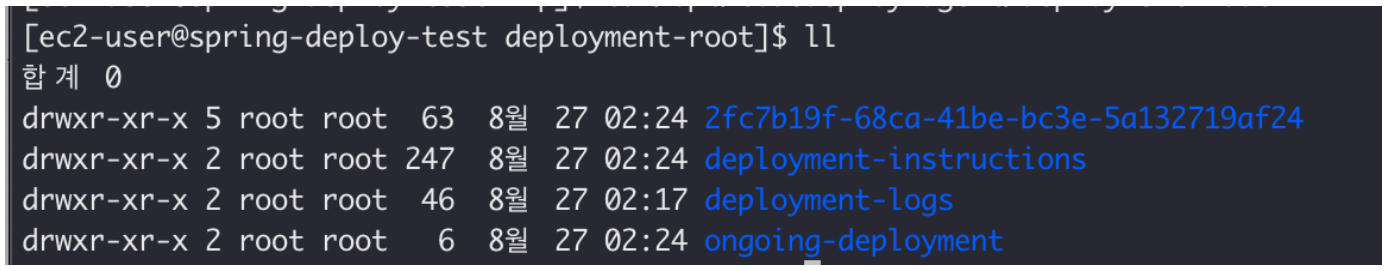
5. EC2에서 CodeDeploy 로그 확인해보기

CodeDeploy와 같이 AWS가 지원하는 서비스에는 오류가 발생했을 때 로그 찾는 방법을 모르면 오류를 해결하기가 어렵다. 그래서 배포가 실패하면 어느 로그를 봐야 할지 간단하게 소개해보려고 한다.

CodeDeploy에 관한 대부분 내용은 **/opt/codedeploy-agent/deployment-root**에 있다.

- 해당 디렉토리 이동 : cd /opt/codedeploy-agent/deployment-root

이동한 뒤 목록을 확인해 보면 다음과 같은 내용을 확인할 수 있다.



목록 확인

최상단의 영문과 대시(-)가 있는 디렉토리 명은 CodeDeploy ID이다.

/opt/codedeploy-agent/deployment-root/deployment-logs/codedeploy-agent-deployments.log

- CodeDeploy 로그 파일이다.
- CodeDeploy로 이루어지는 배포 내용 중 표준 입/출력 내용은 모두 여기에 담겨 있다.
- 작성한 echo 내용도 모두 표기된다.

파일 및 인스턴스 정리

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

AWS

EC2 인스턴스
- spring-deploy-test

RDS 인스턴스
- spring-deploy-test

S3 버킷
- 사용자: spring-deploy-test-build

IAM
- 사용자: spring-travis-deploy (Travis-ci → S3, CodeDeploy 접근 권한)
- 역할: role-ec2-codedeploy (ec2 → CodeDeploy 접근 권한)
- 역할: role-codedeploy (CodeDeploy → ec2 접근 권한)

CodeDeploy 그룹
- Spring-deploy-test

Spring

Travis-ci 설정파일
- .travis.yml

Codedeploy 설정파일
- appspec.yml

배포 스크립트
- deploy.sh

deploy.sh: 자동배포 스크립트

EC2

```
/app
    application-oauth.yml
    application-db.yml
    /step1
        /spring_deploy_test
        deploy.sh
    /step2
        /zip
            appspec.yml
            deploy.sh
            jar
```

step1: 수동배포
/step1/deploy.sh: 수동배포 스크립트
step2: 자동배포
/step2/zip: codedeploy에서 전송한 실행파일들

하지만 문제가 하나 남았다. 배포하는 동안 스프링 부트 프로젝트는 종료 상태가 되어 서비스를 이용할 수 없다는 것이다. 계속 서비스가 유지될 수 있는 서비스 중단 없는 배포 방법 즉, 무중단 배포를 NginX를 통해 진행하면 된다.