EC2 서버에 프로젝트 배포

작성일시	@2022년 6월 19일 오전 11:15
파트	8장
자료	
작성자	
속성	

8.1 EC2에 프로젝트 Clone 받기

```
sudo yum install git

git --version
```

```
mkdir ~/app && mkdir ~/app/step1

cd ~/app/step1
```

```
git clone 복사한 주소

cd 프로젝트명

ll
```

1. **Github에서 프로젝트 주소를 복사를 진행**
2. **git clone 명령어로 현재 폴더에 프로젝트를 복사**
3. **해당 프로젝트로 이동(cd)**
4. **ll 명령어를 입력하면 프로젝트 안에 있는 파일들의 이름과 권한 정보가 같이 나옵니다.**

※ 만약 프로젝트 수정 후 **commit**을 하면 EC2에서도 **git pull** 을 통해 변경사항을 적용해주어야한다.

```
./gradlew test
```

- **bash: ./gradlew: Permission denied**

  → gradlew에 실행 권한이 없을 때 뜨는 오류

```
chmod +x ./gradlew
```

8.2배포 스크립트 만들기

쉘 스크립트와 Vim은 서로 다른 역할을 한다.

- **쉘 스크립트**
  - → .sh라는 파일의 확장자를 가진다.
  - → 리눅스에서 사용할 수 있는 스크립트 파일의 한 종류
  - → 명령어 하나하나 실행해야하는 작업을 한번에 가능하도록 한다.
- **Vim**
  - → 리눅스와 같이 GUI(마우스 컨트롤이 불가능한)가 아닌 환경에서 사용 가능한 편집 도구
  - → 가장 대중적이다.
  - → 자동화와 서버 환경에서 다양한 이점이 많다.
  - → 그 외.. > 이맥스(Emacs), 나노(NANO)

→ step1 디렉토리에 **deploy.sh** 라는 쉘 스크립트를 생성

```
vim ~/app/step1/deploy.sh
```

→ **deploy.sh 코드**

```
#!/bin/bash

#쉘에서는 타입 없이 변수를 저장하며 사용방법은 $변수명 입니다.
REPOSITORY=/home/ec2-user/app/step1  #프로젝트 디렉토리 주소를 변수에 저장
PROJECT_NAME=Springboot-webservice   #프로젝트 이름을 변수에 저장

cd $REPOSITORY/$PROJECT_NAME  #git clone을 받았던 디렉토리로 이동

echo "> Git Pull"

git pull  #master 브렉치의 최신 내용(Github)을 받습니다.

echo "> 프로젝트Build 시작"

./gradlew build  #프로젝트 내부의 gradlew로 build 수행

echo "> step1 디렉토리 이동"

cd $REPOSITORY

echo "> Build 파일복사"

#build의 결과물인 .jar파일을 복사하여 jar 파일을 모아둔 위치로 이동합니다.
cp $REPOSITORY/$PROJECT_NAME/build/libs/*.jar $REPOSITORY/

echo "> 현재 구동중인 애플리케이션pid 확인"

#기존에 수행중이던 스프링 부트 애플리케이션을 종료
#pgrab > process id만 추출
# -f > 프로세스 이름으로 찾는 옵션
CURRENT_PID=$(pgrep -f ${PROJECT_NAME}.*.jar)

echo "현재 구동 중인 애플리케이션pid: $CURRENT_PID"

#현재 구동 중인 프로세스가 있는지 없는지 process id값을 보고 있으면 해당 프로세스를 종료
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 어플리케이션 배포"

#새로 실행할 jar파일명을 찾는다.
JAR_NAME=$(ls -tr $REPOSITORY/ | grep jar | tail -n 1)

echo "> JAR Name: $JAR_NAME"

#찾은 jar 파일을  nohup으로 실행
nohup java -jar \
        -Dspring.config.location=classpath:/application.properties,/home/ec2-user/app/application-oauth.properties,/home/ec2-user/app/
        -Dspring.profiles.active=real \
        $REPOSITORY/$JAR_NAME 2>&1 &
```

nohup이란?

- nohup은 프로세스를 실행한 터미널의 세션 연결이 끊기더라도 프로세스를 계속해서 동작시키는 명령어이다.
- 기본적으로 터미널에서 세션의 logout이 발생하면, 해당 터미널에서 실행된 프로세스들에게 HUP 신호를 전달하여 종료시킨다.
- nohup은 프로세스들을 마치 데몬인 것처럼 동작시켜 이러한 HUP 시그널을 무시하도록한다.

```
chmod +x ./deploy.sh
```

※참고

- 퍼미션의 종류
  - r(읽기): 파일의 읽기 권한
  - w(쓰기): 파일의 쓰기 권한
  - x(실행): 파일의 실행 권한

```
./deploy.sh
```

```
vim nohub.out
```

nobub.out → 실행되는 애플리케이션의 출력되는 모든 내용을 가지고 있다.

외부 Security 파일 등록

- 로컬PC에서는 application-oauth.properties가 있어 괜찮았지만 해당 파일은 Github에 올리지 않아서 **서버에도 해당 설정을 넣어주어야합니다.**

따라서 EC2의 app 디렉토리에 application-oauth-properties라는 이름의 파일로 같은 내용을 넣어 만들어줍니다.

```
...
nohup java -jar \
```

```
      #스프링 설정 파일위치를 지정
      # > application-oauth-properties 위치로
      # > classpath(jar 안에 있는 resuorce 디렉토리를 기준으로경로 생성)
     -Dspring.config.location=classpath:/application.properties,/home/ec2-user/app/application-oauth.properties,/home/ec2-user/app/app
...
```

- 생성한 파일을 deploy.sh에 코드 추가

8.4 스프링 부트 프로젝트로 RDS 접근하기

- **Maria DB에서 스프링부트 프로젝트 실행을 위해서 필요한 작업**

1. 테이블 생성 - Maria DB에 직접 쿼리를 사용해 생성
2. 프로젝트 설정 - 자바 프로젝트가 Maria DB에 접근하기 위해 필요한 드라이버 설치
3. EC2 설정 - EC2 서버 내부에서 접속 정보 관리 설정

1. **RDS 테이블 생성**
   - JPA가 사용될 엔티티 테이블
     - 인텔리제이에서 프로젝트의 테스트코드 실행 시 로그로 생성되는 쿼리를 사용
   - 스프링 세션이 사용될 테이블
     - schema-mysql.sql에서 코드 복사

2. **프로젝트 설정**
   - build.gradle에
   - 하위 코드를 추가하여 Maria DB 드라이버 추가

```
implementation("org.mariadb.jdbc:mariadb-java-client")
```

   - src/main/resources/ 에 application-real.properties 파일 추가

```
#해당 프로필을 포함해서 실행
spring.profiles.include=oauth,real-db

#데이터베이스에 종속적이지 않다"를 쓰기위해 hibernate.dialect 를 쓴다.
#서로 다른 데이터베이스 문법, 타입 등을 알아서 처리해주므로
#개발자는 데이터베이스를 바꾸더라도 코드를 크게 바꾸지 않게 된다.
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL57Dialect
spring.jpa.properties.hibernate.dialect.storage_engine=innodb


spring.datasource.hikari.jdbc-url=jdbc:h2:mem:testdb;MODE=MYSQL
spring.datasource.hikari.username=sa

#스프링 세션 설정
spring.session.store-type=jdbc
```

3. **EC2 설정**
   - EC2 > app 디렉토리에 application-real-db.properties 파일 생성

```
#JPA로 테이블이 자동 생성되는 옵션을 None 설정
#이 설정을 안하면 자칫 테이블이 모두 새로 생설될 수 있다!
spring.jpa.hibernate.ddl-auto=none
```

```
spring.jpa.show_sql=false

spring.datasource.hikari.jdbc-url=jdbc:mariadb://k-springboot2-webservice.cjpxd5e3tbda.ap-northeast-2.rds.amazonaws.com:3306/
spring.datasource.hikari.username=rladudtjr871
spring.datasource.hikari.password=?
spring.datasource.hikari.driver-class-name=org.mariadb.jdbc.Driver
```

그리고 deploy.sh에 코드 추가

```
...
nohup java -jar \
      #스프링 설정 파일위치를 지정
      # > application-oauth-properties 위치로
      # > classpath(jar 안에 있는 resuorce 디렉토리를 기준으로경로 생성)
     -Dspring.config.location=classpath:/application.properties,/home/ec2-user/app/application-oauth.properties,/home/ec2-use
     #application-real.properties를 활성화
     #이렇게하면 application-real파일에 real-db를 포함한다는 옵션이 있어 real-db도 활성화 대상에 포함
     -Dspring.profiles.active=real \
...
```

```
#해당 명령어 실행시 html코드가 보여야 성공
curl localhost:9090
```

- **EC2에 자동으로 할당된 퍼블릭 DNS(도메인)을 통해 어디서나 EC2 서버에 접근 가능**

- **구글과 네이버에 EC2 주소를 등록하여 로그인이 사이트에서 동작하도록 하자.**

→ **서비스 URL**

- 로그인을 시도하는 서비스가 네이버에 등록된 서비스인지 판단하는 항목
- 네이버에서는 아직 지원되지 않아 하나만 등록 가능
- 즉 EC2의 주소를 등록하면 localhost가 안된다.
- 개발단계에서는 등록하지 않는 것을 추천

→ **Callback URL**

- 전체 주소를 등록(EC2 퍼블릭 DNS:9090/login/oauth2/code/naver)

현재 방식의 문제점

- 수동 실행되는 Test
  - 본인 코드가 다른 개발자 코드에 미치는 영향을 확인하려면 전체 테스트를 수행해야한다.
  - 작업 진행 시 수동으로 전체 테스트를 수행해야한다.
- 수동 Build

- 다른 사람과 자신의 브랜치가 합쳐졌을 때 이상 유무를 Build를 수행해야 알 수 있다.
- 이를 매번 개발자가 직접 실행해봐야 함.

9장에서 개선…