Docker로 Django 애플리케이션 구축하기

목차

Docker로 Django 애플리케이션 구축하기

Django는 Python으로 작성된 오픈 소스 웹 프레임워크입니다. 이 장에서는 Docker로 Django 애플리케이션을 구축하는 방법을 알아보겠습니다.

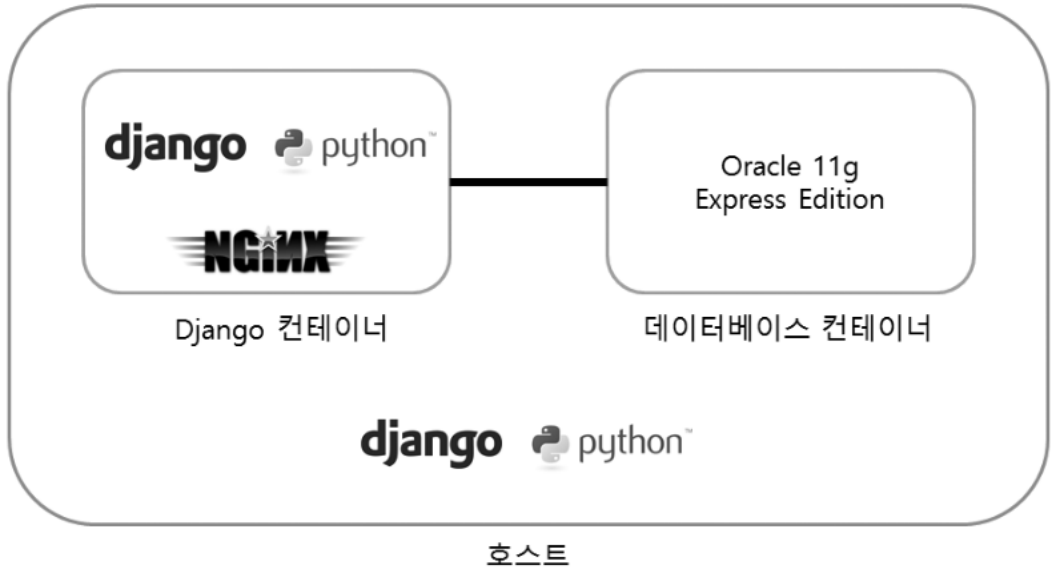
Docker 이미지를 만들기 전에 먼저 Django 개발 환경을 구축합니다.

- 리눅스 배포판별 패키지로 Python pip를 설치합니다.
- pip로 Django를 설치합니다.

Django 이미지와 데이터베이스 이미지 두개를 만듭니다.

- Django 이미지: 웹 서버로 사용할 Nginx를 설치합니다. 그리고 pip로 Gunicorn을 설치합니다.
- 데이터베이스 이미지: 앞에서 MySQL과 PostgreSQL을 사용해보았으니 이번에는 Oracle을 설치합니다. MySQL과 PostgreSQL을 설치하는 방법은 '16.2 MySQL 데이터베이스 Dockerfile 작성하기', '17.3 PostgreSQL 데이터베이스 Dockerfile 작성하기'를 참조하기 바랍니다.

Django 컨테이너에서 데이터베이스 컨테이너를 사용할 수 있도록 컨테이너를 생성할 때 docker run 명령의 --link 옵션으로 연결합니다.



1. Django 설치하기

Django 이미지를 생성하기 전에 먼저 Django 개발 환경을 구축해야 합니다. 각 리눅스 배포판의 패키지로 Python pip를 설치한 뒤 pip로 Django를 설치합니다.
다음 명령을 실행하여 pip와 Python 개발 패키지를 설치합니다.

우분투

```
$ sudo apt-get install python-pip python-dev
```

CentOS 6

```
$ sudo yum install http://dl.fedoraproject.org/pub/epel/6/x86_64/epel-release-6-8.noarch.rpm
$ sudo yum install python-pip python-devel gcc
```

CentOS 7

```
$ sudo yum install http://dl.fedoraproject.org/pub/epel/7/x86_64/e/epel-release-7-2.noarch.rpm
$ sudo yum install python-pip python-devel gcc
```

CentOS 7 EPEL 패키지 버전

CentOS 7용 EPEL 패키지는 버전이 빠르게 업데이트됩니다. rpm 파일을 받을 수 없을 때는 http://dl.fedoraproject.org/pub/epel/7/x86_64/e/에 접속하여 새 버전이 있는지 확인한 뒤 `yum` 명령으로 해당 버전을 설치합니다.

1. Django 설치하기

Python용 Oracle 라이브러리인 cx_Oracle을 사용하려면 Oracle 사이트에서 인스턴트 클라이언트 패키지를 받아서 설치해야 합니다. 웹 브라우저에서 다음 URL에 접속합니다.

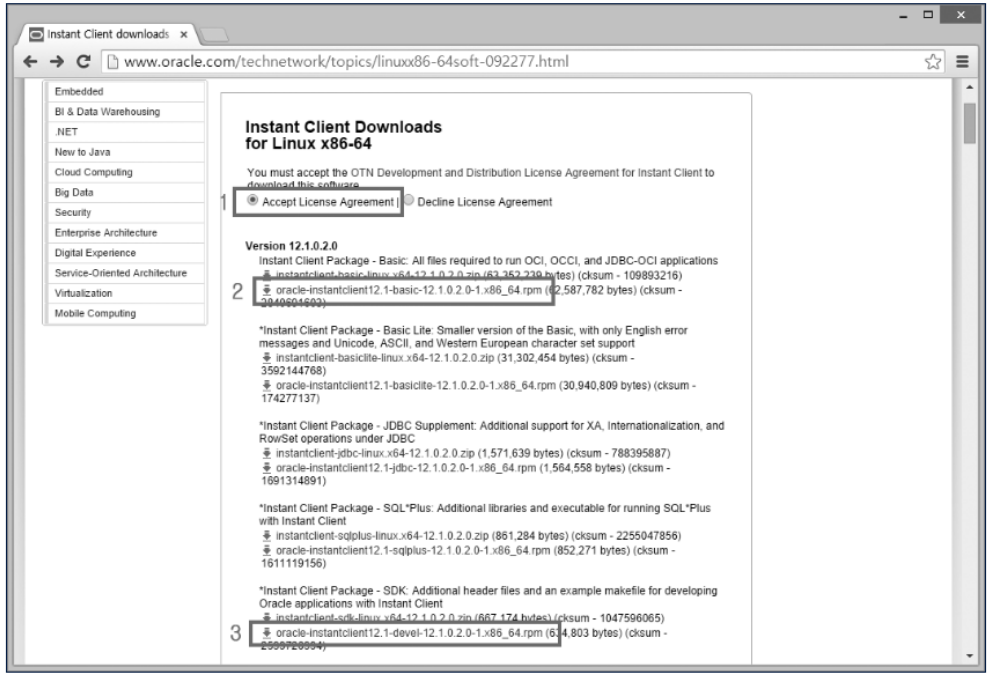


그림 18-2 Oracle 인스턴트 클라이언트 패키지 다운로드

Accept License Agreement를 클릭하고 다음 두 파일을 받습니다. 파일을 받으려면 Oracle 계정으로 로그인을 해야합니다. Oracle 계정이 없다면 먼저 가입을 합니다.

- oracle-instantclient12.1-basic-12.1.0.2.0-1.x86_64.rpm
- oracle-instantclient12.1-devel-12.1.0.2.0-1.x86_64.rpm

참고
SQL*Plus를 사용하려면 oracle-instantclient12.1-sqlplus-12.1.0.2.0-1.x86_64.rpm 파일을 받아서 설치합니다.

1. Django 설치하기

받은 rpm 파일을 Django를 설치할 리눅스 서버로 복사합니다. Windows에서는 WinSCP(http://www.winscp.net), Mac OS X에서는 sftp 명령을 사용하면 됩니다.
파일을 복사했으면 다음 명령을 실행하여 Oracle 인스턴트 클라이언트 패키지를 설치합니다. 우분투는 alien으로 rpm 패키지를 설치하면 됩니다. 단 alien으로는 의존 관계에 있는 패키지가 자동으로 설치되지 않으므로 libaio1 패키지를 따로 설치합니다.

우분투

```
$ sudo apt-get install alien libaio1
$ sudo alien -i oracle-instantclient12.1-basic-12.1.0.2.0-1.x86_64.rpm
$ sudo alien -i oracle-instantclient12.1-devel-12.1.0.2.0-1.x86_64.rpm
```

CentOS

```
$ sudo yum install oracle-instantclient12.1-basic-12.1.0.2.0-1.x86_64.rpm
$ sudo yum install oracle-instantclient12.1-devel-12.1.0.2.0-1.x86_64.rpm
```

alien

alien은 레드햇(.rpm), LSB, 스탬피드, 슬랙웨어 리눅스 배포판의 패키지를 데비안(우분투) 리눅스 패키지로 변환하여 설치해주는 도구입니다.

Oracle 인스턴트 클라이언트 동적 라이브러리를 사용할 수 있도록 설정합니다.

```
$ echo "/usr/lib/oracle/12.1/client64/lib" | sudo tee -a /etc/ld.so.conf.d/oracle.conf
$ sudo ldconfig
```

.bashrc 파일에 Oracle 환경 변수를 설정합니다.

```
~$ echo "export ORACLE_HOME=/usr/lib/oracle/12.1/client64" >> .bashrc
~$ source .bashrc
```

1. Django 설치하기

pip 명령으로 django, cx_Oracle 패키지를 설치합니다. 여기서 env 명령으로 root 권한에도 ORACLE_HOME을 설정합니다.

```
$ sudo env ORACLE_HOME=$ORACLE_HOME pip install django cx_Oracle
```

MySQL, PostgreSQL 사용하기

MySQL을 사용하려면 다음 패키지를 설치합니다.

우분투

```
$ sudo apt-get install mysql-client libmysqlclient-dev
```

CentOS

```
$ sudo yum install mysql mysql-devel
```

```
$ sudo pip install django MySQL-python
```

PostgreSQL을 사용하려면 다음 패키지를 설치합니다.

우분투

```
$ sudo apt-get install postgresql-client libpq-dev
```

CentOS

```
$ sudo yum install postgresql postgresql-devel
```

```
$ sudo pip install django psycopg2
```

2. Django Dockerfile 작성하기

Django 설치가 끝났으니 예제 Django 애플리케이션을 생성합니다.

```
~$ django-admin.py startproject exampleapp
```

exampleapp 디렉터리가 생성되었습니다. 앞에 설치한 Oracle 인스턴트 클라이언트 rpm 파일을 **exampleapp** 디렉터리 아래로 이동합니다. Django 이미지에도 Oracle 인스턴트 클라이언트를 설치해야 하므로 rpm 파일이 필요합니다.

```
$ mv oracle-instantclient12.1-basic-12.1.0.2.0-1.x86_64.rpm exampleapp/
$ mv oracle-instantclient12.1-devel-12.1.0.2.0-1.x86_64.rpm exampleapp/
```

2. Django Dockerfile 작성하기

exampleapp/exampleapp 디렉터리 아래에 있는 settings.py 파일을 열고 다음과 같이 수정합니다.

~/exampleapp/exampleapp/settings.py

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.oracle',
        'NAME': 'XE',
        'USER': 'system',
        'PASSWORD': os.getenv('DB_ENV_ORACLE_PASSWORD'),
        'HOST': os.getenv('ORACLE_HOST') or 'db',
        'PORT': '1521',
    }
}
```

- ENGINE: Oracle을 사용하기 위해 django.db.backends.oracle을 설정합니다.
- NAME: 데이터베이스 이름입니다. XE를 설정합니다.
- USER: system을 설정합니다.
- PASSWORD: 환경 변수의 DB_ENV_ORACLE_PASSWORD를 사용하도록 설정합니다. `docker run` 명령의 `--link` 옵션으로 컨테이너를 연결했을 때 연결한 컨테이너의 환경 변수는 `<별칭>_ENV_<환경 변수>` 형식입니다. 우리는 컨테이너를 연결할 때 별칭을 db로 하고, 데이터베이스 컨테이너에서 환경 변수는 ORACLE_PASSWORD를 사용할 것이기 때문에 DB_ENV_ORACLE _PASSWORD가 됩니다.
- HOST: or 연산자를 이용하여 개발 환경과 데이터베이스 컨테이너에서 사용할 데이터베이스 호스트를 각각 설정합니다. 개발을 할 때는 환경 변수의 ORACLE_HOST에 데이터베이스 컨테이너의 IP 주소를 설정합니다. 그리고 데이터베이스 컨테이너를 연결할 때는 별칭을 db로 할 것이므로 db로 설정합니다.
- PORT: 1521번을 설정합니다.

2. Django Dockerfile 작성하기

MySQL, PostgreSQL 사용하기

다음은 MySQL 사용 설정입니다.

- dockerbook/Chapter18/exampleapp_mysql/exampleapp/settings.py

~/exampleapp/exampleapp/settings.py

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'exampleapp',
        'USER': 'root',
        'PASSWORD': os.getenv('DB_ENV_MYSQL_ROOT_PASSWORD'),
        'HOST': os.getenv('MYSQL_HOST') or 'db',
        'PORT': '3306',
    }
}
```

다음은 PostgreSQL 사용 설정입니다.

- dockerbook/Chapter18/exampleapp_postgresql/exampleapp/settings.py

~/exampleapp/exampleapp/settings.py

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'NAME': 'exampleapp',
        'USER': 'postgres',
        'PASSWORD': os.getenv('DB_ENV_POSTGRESQL_PASSWORD'),
        'HOST': os.getenv('POSTGRESQL_HOST') or 'db',
        'PORT': '5432',
    }
}
```

2. Django Dockerfile 작성하기

다음 내용을 Dockerfile로 저장합니다.

~/exampleapp/Dockerfile

```
FROM centos:centos7

RUN yum install -y http://dl.fedoraproject.org/pub/epel/7/x86_64/e/epel-release-7-2.noarch.rpm
RUN yum install -y python-pip python-devel nginx gcc

RUN echo "daemon off;" >> /etc/nginx/nginx.conf
ADD exampleapp.conf /etc/nginx/conf.d/exampleapp.conf

WORKDIR /tmp
ADD oracle-instantclient12.1-basic-12.1.0.2.0-1.x86_64.rpm /tmp/
ADD oracle-instantclient12.1-devel-12.1.0.2.0-1.x86_64.rpm /tmp/
RUN yum install -y oracle-instantclient12.1-basic-12.1.0.2.0-1.x86_64.rpm
RUN yum install -y oracle-instantclient12.1-devel-12.1.0.2.0-1.x86_64.rpm
RUN rm -rf *.rpm

ENV ORACLE_HOME /usr/lib/oracle/12.1/client64
RUN echo "/usr/lib/oracle/12.1/client64/lib" > /etc/ld.so.conf.d/oracle.conf && ldconfig
RUN pip install django gunicorn cx_Oracle

ADD ./ /var/www/exampleapp
WORKDIR /var/www/exampleapp
RUN chmod +x entrypoint.sh
RUN rm -rf *.rpm

EXPOSE 80

ENTRYPOINT ./entrypoint.sh
```

지금까지 우분투 14.04를 기반으로 이미지를 구성해보았으니 이번에는 CentOS 7을 기반으로 하겠습니다.

- FROM으로 **centos:centos7**를 기반으로 이미지를 생성하도록 설정합니다.
- `yum install` 명령으로 EPEL 패키지를 설치하고, **python-pip**, **python-devel**, **nginx**, **gcc**를 설치합니다.
- nginx를 데몬이 아닌 foreground로 실행하도록 설정합니다. nginx를 데몬 상태로 실행하면 Docker 컨테이너가 바로 정지되므로 주의합니다.
- **/etc/nginx/conf.d** 디렉터리에 **exampleapp.conf** 파일을 추가합니다.
- **/tmp** 디렉터리에 Oracle 인스턴트 클라이언트 rpm 파일을 추가한 뒤 `yum install` 명령으로 설치합니다. 그리고 설치가 끝나면 rpm 파일을 삭제합니다.
- ENV로 ORACLE_HOME 환경 변수를 설정합니다. **/usr/lib/oracle/12.1/client64**는 Oracle 인스턴트 클라이언트가 설치된 경로입니다.
- **/etc/ld.so.conf.d** 디렉터리에 **oracle.conf** 파일을 생성하고, `ldconfig` 명령을 실행하여 Oracle 인스턴트 클라이언트의 동적 라이브러리를 사용할 수 있도록 설정합니다.
- `pip install` 명령으로 **django**, **gunicorn**, **cx_Oracle**을 설치합니다.
- Django 애플리케이션 디렉터리를 **/var/www/exampleapp** 디렉터리에 추가합니다.
- entrypoint.sh 파일을 실행할 수 있도록 권한을 설정합니다.
- EXPOSE에 **80**을 설정하여 **80**번 포트에 접속할 수 있도록 합니다.
- ENTRYPOINT에 ./entrypoint.sh 파일을 설정하여 컨테이너가 시작되었을 때 스크립트 파일을 실행합니다.

Django 애플리케이션 디렉터리(**exampleapp**)를 추가하기 전에 **oracle-instantclient12.1-basic-12.1.0.2.0-1.x86_64.rpm**, **oracle-instantclient12.1-devel-12.1.0.2.0-1.x86_64.rpm** 파일을 미리 **/tmp** 디렉터리에 추가하여 설치합니다. **exampleapp** 디렉터리를 추가한 뒤 rpm 파일을 설치하면 **exampleapp** 디렉터리의 내용이 바뀔 때마다 rpm 파일을 설치하게 되므로 이미지 생성 시간이 오래걸립니다. 자세한 내용은 '17.2 Rails Dockerfile 작성하기'의 'Docker 이미지 생성 시간 줄이기' 부분을 참조하기 바랍니다.

2. Django Dockerfile 작성하기

다음 내용을 Dockerfile로 저장합니다.

MySQL, PostgreSQL 사용하기

다음은 MySQL 사용 설정입니다.

- dockerbook/Chapter18/exampleapp_mysql/Dockerfile

~/exampleapp/Dockerfile

```
RUN yum install -y python-pip python-devel nginx gcc mysql-devel

RUN pip install django gunicorn MySQL-python
```

다음은 PostgreSQL 사용 설정입니다.

- dockerbook/Chapter18/exampleapp_postgresql/Dockerfile

~/exampleapp/Dockerfile

```
RUN yum install -y python-pip python-devel nginx gcc postgresql-devel

RUN pip install django gunicorn psycopg2
```

2. Django Dockerfile 작성하기

이제 Nginx 설정 파일을 작성합니다. 다음 내용을 exampleapp.conf 파일로 저장합니다.

~/exampleapp/exampleapp.conf

```
upstream gunicorn {
  server unix:/tmp/gunicorn.sock;
}

server {
  listen 80;
  server_name _;

  location /static/ {
    alias /var/www/exampleapp/static/;
  }

  location / {
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header Host $http_host;
    proxy_redirect off;

    if (!-f $request_filename) {
      proxy_pass http://gunicorn;
      break;
    }
  }
}
```

- **upstream** 항목에는 Gunicorn의 유닉스 소켓 **/tmp/gunicorn.sock**을 설정합니다.
- **server** 항목을 설정합니다.
  - **listen 80**을 설정하여 80번 포트를 사용합니다.
  - Nginx는 정적 파일(html, js, css 등)을 전송할 것이므로 Django 애플리케이션 디렉터리 아래의 **static** 디렉터리를 설정합니다(각자 상황에 맞게 수정합니다).
  - Nginx로 들어온 요청이 파일명이 아니면 앞에서 설정한 유닉스 소켓(**http://gunicorn**)으로 보냅니다. 이렇게 설정하면 정적 파일은 Nginx가 전송하고, 나머지 RESTful API는 Django가 처리하게 됩니다.

2. Django Dockerfile 작성하기

다음 내용을 entrypoint.sh로 저장합니다.

~/exampleapp/entrypoint.sh

```
#!/bin/bash

gunicorn exampleapp.wsgi -b unix:/tmp/gunicorn.sock -D
nginx
```

- `gunicorn`에 **exampleapp.wsgi**를 설정합니다. `<Django 애플리케이션 이름>.wsgi` 형식입니다.
  - `-b` 옵션을 사용하여 **/tmp/gunicorn.sock**에 유닉스 소켓을 생성합니다.
  - `-D` 옵션을 사용하여 데몬 모드로 실행합니다.
- 앞에서 **nginx.conf**에 **daemon off;**로 설정했으므로 Nginx 웹 서버를 foreground로 실행합니다. 여기서 Nginx를 foreground로 실행하지 않으면 `docker run -d`로 컨테이너를 생성해도 바로 정지되므로 주의합니다.

`docker build` 명령으로 이미지를 생성합니다.

```
~/exampleapp$ sudo docker build --tag django .
```

3. Oracle 데이터베이스 Dockerfile 작성하기

데이터베이스 이미지를 생성하기 전에 Oracle 사이트에서 Oracle 11g Express Edition 패키지를 받습니다. 웹 브라우저에서 다음 URL에 접속합니다.

- http://www.oracle.com/technetwork/database/database-technologies/express-edition/downloads/index.html

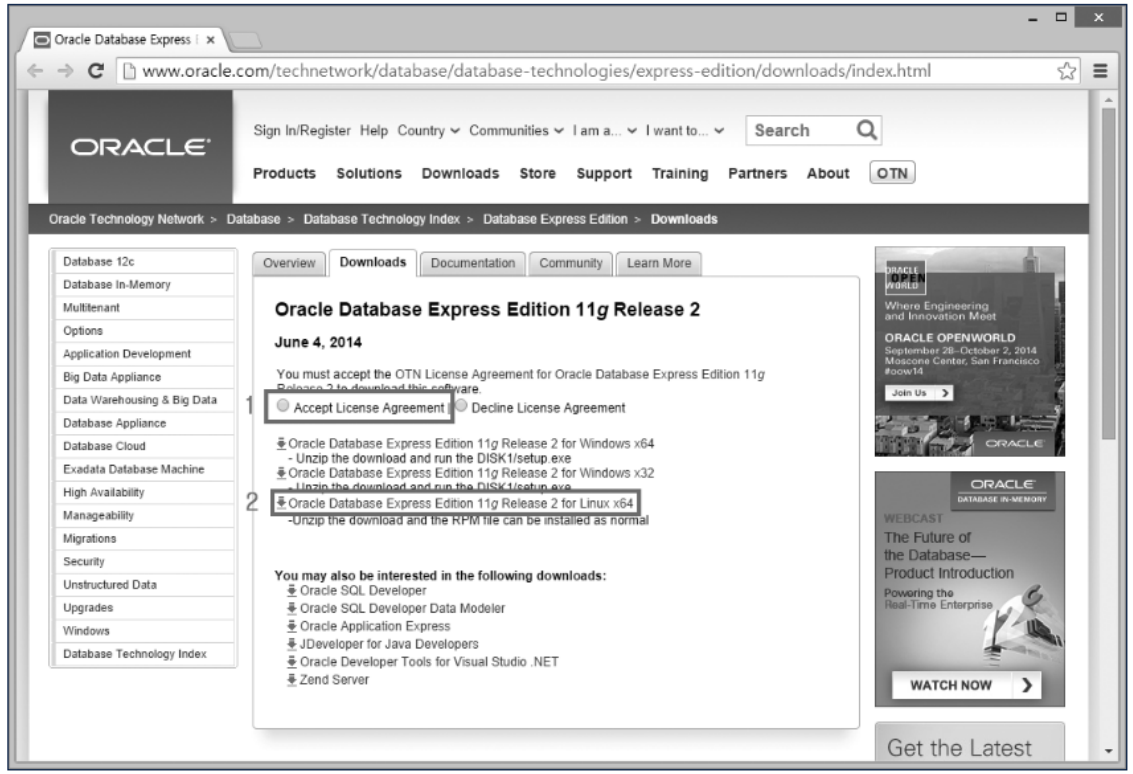


그림 18-3 Oracle 11g Express Edition 패키지 다운로드

Accept License Agreement를 클릭하고 Oracle Database Express Edition 11g Relase 2 for Linux x64를 받습니다. 파일을 받으려면 Oracle 계정으로 로그인을 해야합니다. Oracle 계정이 없다면 먼저 가입을 합니다.

oracle-xe-11.2.0-1.0.x86_64.rpm.zip 파일의 압축을 풀면 Disk1 디렉터리에 rpm 파일이 들어있습니다. 이 rpm 파일을 데이터베이스 이미지를 생성할 리눅스 서버로 복사합니다. Windows에서는 WinSCP(http://www.winscp.net), Mac OS X에서는 `sftp` 명령을 사용하면 됩니다.

3. Oracle 데이터베이스 Dockerfile 작성하기

이제 데이터베이스 이미지를 생성합니다. oracle 디렉터리를 생성하고 다음 내용을 Dockerfile로 저장합니다.

```
~$ mkdir oracle
~$ cd oracle
```

~/oracle/Dockerfile

```
FROM centos:centos7

RUN yum install -y net-tools bc openssh-server
RUN rm -rf /var/lock
RUN mkdir -p /var/lock/subsys

WORKDIR /tmp
ADD oracle-xe-11.2.0-1.0.x86_64.rpm /tmp/
RUN yum install -y oracle-xe-11.2.0-1.0.x86_64.rpm
RUN rm -rf oracle-xe-11.2.0-1.0.x86_64.rpm

WORKDIR /u01/app/oracle/product/11.2.0/xe/config/scripts
RUN sed -i "s/memory_target/#memory_target/g" init.ora
RUN sed -i "s/memory_target/#memory_target/g" initXETemp.ora
RUN echo -e "pga_aggregate_target=200540160\nsga_target=601620480" >> init.ora
RUN echo -e "pga_aggregate_target=200540160\nsga_target=601620480" >> initXETemp.ora

RUN sed -i "s/#PermitRootLogin/PermitRootLogin/g" /etc/ssh/sshd_config
RUN sshd-keygen

ADD entrypoint.sh /entrypoint.sh
RUN chmod +x /entrypoint.sh

EXPOSE 22
EXPOSE 1521
EXPOSE 8080

ENTRYPOINT /entrypoint.sh
```

- `yum install` 명령으로 net-tools, bc, openssh-server 패키지를 설치합니다.
- /var/lock 파일을 삭제하고 /var/lock/subsys 디렉터리를 생성합니다.
- /tmp 디렉터리에 oracle-xe-11.2.0-1.0.x86_64.rpm 파일을 추가하고 `yum install` 명령으로 설치합니다. 그리고 설치가 끝나면 rpm 파일을 삭제합니다.
- Oracle 설정 파일인 init.ora, initXETemp.ora 파일을 수정합니다.
  - memory_target은 #을 추가하여 사용하지 않도록 수정합니다.
  - pga_aggregate_target=200540160, nsga_target=601620480 두 줄을 추가합니다.
- SSH 서버를 설정합니다.
  - /etc/ssh/sshd_config의 PermitRootLogin 부분에서 #을 삭제하여 root로 로그인할 수 있도록 설정합니다.
  - `sshd-keygen` 명령으로 서버 키 쌍을 생성합니다.
- entrypoint.sh 파일을 추가한 뒤 실행할 수 있도록 권한을 설정합니다.
- EXPOSE에 22, 1521, 8080을 설정하여 22, 1521, 8080번 포트에 접속할 수 있도록 합니다.
- ENTRYPOINT에 /entrypoint.sh 파일을 설정하여 컨테이너가 시작되었을 때 스크립트 파일을 실행합니다.

3. Oracle 데이터베이스 Dockerfile 작성하기

다음 내용을 entrypoint.sh로 저장합니다.

~/oracle/entrypoint.sh

```
#!/bin/bash

if [ -z $ORACLE_PASSWORD ]; then
  exit 1
fi

echo "export ORACLE_HOME=/u01/app/oracle/product/11.2.0/xe" > /etc/profile.d/oracle.sh
echo "export ORACLE_SID=XE" >> /etc/profile.d/oracle.sh
echo "export PATH=$PATH:/u01/app/oracle/product/11.2.0/xe/bin" >> /etc/profile.d/oracle.sh
source /etc/profile.d/oracle.sh

printf 8080\\n1521\\n$ORACLE_PASSWORD\\n$ORACLE_PASSWORD\\nn | /etc/init.d/oracle-xe configure
echo "root:$ORACLE_PASSWORD" | chpasswd

/usr/sbin/sshd -D
```

- 환경 변수에 ORACLE_PASSWORD가 없으면 데이터베이스를 실행하지 않고 빠져나옵니다.
- /etc/profile.d/oracle.sh 파일에 ORACLE_HOME, ORACLE_SID, PATH 환경 변수를 설정하고, `source` 명령으로 설정값을 적용합니다.
  - /u01/app/oracle/product/11.2.0/xe는 Oracle 데이터베이스가 설치된 디렉터리입니다.
  - Oracle 11g Express Edition이므로 XE를 설정합니다.
  - `sqlplus` 등의 명령을 사용할 수 있도록 실행 파일 경로도 설정합니다.
- `printf` 명령을 사용하여 `/etc/init.d/oracle-xe configure`를 실행할 때 설정값을 넣습니다. 8080, 1521을 설정하고, 비밀번호는 환경 변수의 ORACLE_PASSWORD 값을 설정합니다.
  - `/etc/init.d/oracle-xe configure`를 실행하면 자동으로 Oracle 서비스가 실행됩니다.
- SSH로 접속할 수 있도록 root 계정의 비밀번호를 환경 변수의 ORACLE_PASSWORD 값으로 변경합니다.
- `/usr/sbin/sshd -D` 명령을 실행하여 SSH 서버를 foreground로 실행합니다.

`docker build` 명령으로 이미지를 생성합니다.

```
~/oracle$ sudo docker build --tag oracle .
```

4. Django와 데이터베이스 컨테이너 생성하기

Django와 데이터베이스 이미지 준비가 끝났으니 컨테이너를 생성합니다. 먼저 데이터베이스 컨테이너부터 생성합니다.

```
$ sudo docker run -d --name db -e ORACLE_PASSWORD=examplepassword oracle
```

- 데이터베이스 컨테이너를 생성할 때 `-e` 옵션을 사용하여 ORACLE_PASSWORD에 사용할 system 계정의 비밀번호를 설정합니다.

Django 애플리케이션 디렉터리로 이동한 뒤 Django 데이터베이스를 초기화하고 관리자 계정을 설정합니다.

```
~$ export ORACLE_HOST=$(sudo docker inspect -f "{{ .NetworkSettings.IPAddress }}" db)
~$ export DB_ENV_ORACLE_PASSWORD=examplepassword
~$ cd
~$ cd exampleapp
~/exampleapp$ ./manage.py syncdb
Operations to perform:
  Apply all migrations: admin, contenttypes, auth, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying sessions.0001_initial... OK

You have installed Django's auth system, and don't have any superusers defined.
Would you like to create one now? (yes/no): yes
Username (leave blank to use 'pyrasis'): admin
Email address: admin@example.com
Password: <비밀번호 입력>
Password (again): <비밀번호 입력>
Superuser created successfully.
```

- `export` 명령을 사용하여 환경 변수의 ORACLE_HOST에 db 컨테이너의 IP 주소를 설정합니다.
  - `docker inspect` 명령에서 `-f` 옵션을 사용하면 특정 항목만 출력할 수 있습니다. "{{ .NetworkSettings.IPAddress }}"는 컨테이너의 IP 주소입니다.
- `export` 명령을 사용하여 환경 변수의 DB_ENV_ORACLE_PASSWORD에 Oracle 데이터베이스 비밀번호를 설정합니다.
- `manage.py syncdb`를 실행하여 Django 데이터베이스를 초기화합니다. 관리자(superuser) 설정 부분이 나오면 `yes`를 입력하고 관리자 계정과 이메일, 비밀번호를 설정합니다.

MySQL, PostgreSQL 사용하기

다음은 MySQL 사용 설정입니다.

```
~$ export MYSQL_HOST=$(sudo docker inspect -f "{{ .NetworkSettings.IPAddress }}" db)
~$ export DB_ENV_MYSQL_ROOT_PASSWORD=examplepassword
~$ mysql -h $MYSQL_HOST -uroot -p$DB_ENV_MYSQLROOT_PASSWORD \
    -e "create database exampleapp"
~$ cd
~$ cd exampleapp
~/exampleapp$ ./manage.py syncdb
```

다음은 PostgreSQL 사용 설정입니다.

```
~$ export POSTGRESQL_HOST=$(sudo docker inspect -f "{{ .NetworkSettings.IPAddress }}" db)
~$ export DB_ENV_POSTGRESQL_PASSWORD=examplepassword
~$ PGPASSWORD=$DB_ENV_POSTGRESQL_PASSWORD psql -h $POSTGRESQL_HOST -U postgres \
    -c "CREATE DATABASE exampleapp WITH ENCODING 'UTF8' TEMPLATE template0"
~$ cd
~$ cd exampleapp
~/exampleapp$ ./manage.py syncdb
```

4. Django와 데이터베이스 컨테이너 생성하기

Django 컨테이너를 생성합니다.

```
$ sudo docker run -d --name example-django --link db:db -p 80:80 django
```

- Django 컨테이너를 생성할 때 `--link` 옵션을 사용하여 **db** 컨테이너를 **db** 별칭으로 연결합니다. 그리고 `-p` 옵션을 사용하여 외부에서 80번 포트에 접근할 수 있도록 설정합니다.

컨테이너 생성이 끝났으면 웹 브라우저를 실행하고 서버의 IP 주소나 도메인으로 접속합니다.