Docker로 Ruby on Rails 애플리케이션 구축하기

목차

Docker로 Ruby on Rails 애플리케이션 구축하기

Ruby on Rails는 Ruby로 작성된 오픈 소스 웹 프레임워크입니다. 이 장에서는 Docker로 Ruby on Rails 애플리케이션을 구축하는 방법을 알아보겠습니다.

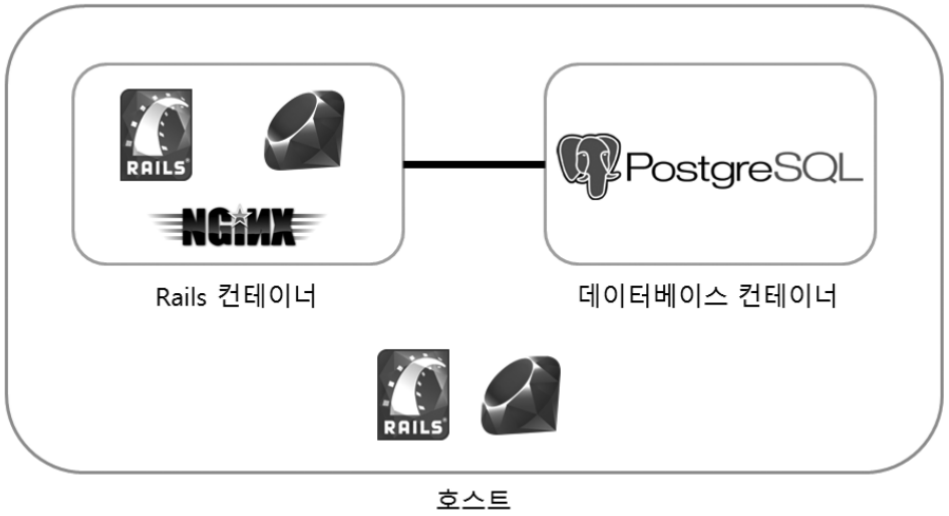
Docker 이미지를 만들기 전에 먼저 Ruby on Rails 개발 환경을 구축합니다.

- Git과 필요한 패키지를 설치합니다.
- rbenv를 설치합니다.
- rbenv로 Ruby를 설치합니다.
- gem으로 Rails, Unicorn를 설치합니다.

Rails 이미지와 데이터베이스 이미지 두 개를 만듭니다.

- Rails 이미지: 웹 서버로 사용할 Nginx를 설치합니다. 그리고 rbenv로 Ruby를 설치하고 필요한 gem(Rails, Unicorn 등)을 설치합니다.
- 데이터베이스 이미지: PostgreSQL을 설치합니다. MySQL을 설치하는 방법은 '16.2 MySQL 데이터베이스 Dockerfile 작성하기'를 참조하기 바랍니다.

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



1. Ruby와 Rails 설치하기

Rails 이미지를 생성하기 전에 먼저 Ruby on Rails 개발 환경을 구축해야 합니다. 각 리눅스 배포판의 패키지를 이용하지 않고 rbenv으로 Ruby를 설치하겠습니다.

먼저 필요한 패키지를 설치합니다.

우분투

```
$ sudo apt-get install autoconf bison build-essential libssl-dev libyaml-dev libreadline6-dev zlib1g-dev libncurses5-dev git
```

CentOS

```
$ sudo yum install gcc openssl-devel libyaml-devel readline-devel zlib-devel git
```

다음 명령을 실행하여 rbenv를 설치합니다.

```
$ git clone https://github.com/sstephenson/rbenv.git ~/.rbenv
```

/home/<사용자 계정>/.rbenv 디렉터리에 rbenv가 설치됩니다. `rbenv` 명령을 사용할 수 있도록 다음 명령을 실행합니다.

```
$ echo 'export PATH="$HOME/.rbenv/bin:$PATH"' >> ~/.bashrc
$ echo 'eval "$(rbenv init -)"' >> ~/.bashrc
$ source .bashrc
```

1. Ruby와 Rails 설치하기

Rails 이미지를 생성하기 전에 먼저 Ruby on Rails 개발 환경을 구축해야 합니다. 각 리눅스 배포판의 패키지를 이용하지 않고 rbenv으로 Ruby를 설치하겠습니다.

`rbenv install` 명령을 사용할 수 있도록 ruby-build를 설치합니다. 그리고 gem을 설치했을 때 매번 `rbenv rehash` 명령을 입력하지 않도록 rbenv-gem-rehash를 설치합니다.

```
~$ mkdir ~/.rbenv/plugins
~$ cd ~/.rbenv/plugins
~/.rbenv/plugins$ git clone https://github.com/sstephenson/ruby-build.git
~/.rbenv/plugins$ git clone https://github.com/sstephenson/rbenv-gem-rehash.git
```

이제 사용자 계정 디렉터리로 이동한 뒤 `rbenv install` 명령으로 Ruby 2.1.3 버전을 설치합니다. 그리고 `rbenv local` 명령으로 Ruby를 현재 사용자 계정에서만 사용할 수 있도록 설정합니다.

```
~/.rbenv/plugins$ cd
~$ rbenv install 2.1.3
~$ rbenv local 2.1.3
```

설치할 수 있는 Ruby 버전 확인하기

다음 명령을 실행하면 설치할 수 있는 Ruby 버전이 출력됩니다. 각자 상황에 맞게 버전을 선택합니다.

```
$ rbenv install -l
```

1. Ruby와 Rails 설치하기

Rails 이미지를 생성하기 전에 먼저 Ruby on Rails 개발 환경을 구축해야 합니다. 각 리눅스 배포판의 패키지를 이용하지 않고 rbenv으로 Ruby를 설치하겠습니다.

`gem` 명령으로 Rails, Unicorn를 설치합니다.

```
$ gem install rails unicorn
```

데이터베이스로 PostgreSQL을 사용하기로 했으므로 PostgreSQL gem을 위한 패키지를 설치합니다.

우분투

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

CentOS

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

MySQL 사용하기

우분투

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

CentOS

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

1. Ruby와 Rails 설치하기

Rails 이미지를 생성하기 전에 먼저 Ruby on Rails 개발 환경을 구축해야 합니다. 각 리눅스 배포판의 패키지를 이용하지 않고 rbenv으로 Ruby를 설치하겠습니다.

일부 gem은 JavaScript로 되어있으므로 Node.js도 설치합니다.

우분투

```
$ sudo apt-get install nodejs
```

CentOS 6

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

CentOS 7

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

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.

2. Rails Dockerfile 작성하기

Ruby와 Rails 설치가 끝났으니 예제 Rails 애플리케이션을 생성합니다.

```
~$ rails new exampleapp --database=postgresql
```

exampleapp 디렉터리가 생성되었습니다. **exampleapp/config** 디렉터리 아래에 있는 **database.yml** 파일을 열고 다음과 같이 수정합니다.

~/exampleapp/config/database.yml

```
default: &default
  adapter: postgresql
  encoding: unicode
  # For details on connection pooling, see rails configuration guide
  # http://guides.rubyonrails.org/configuring.html#database-pooling
  pool: 5
  template: template0
  username: postgres
  password: <%= ENV['DB_ENV_POSTGRESQL_PASSWORD'] %>
  host: <%= ENV['POSTGRESQL_HOST'] || 'db' %>
```

- template: **template0**를 설정합니다. PostgreSQL은 데이터베이스를 생성할 때 기존 데이터베이스를 복제합니다. **template0**는 인코딩을 새로 설정할 수 있습니다. 이 부분을 설정하지 않으면 UTF-8 에러가 발생합니다.
- username: **postgres**를 설정합니다.
- password: 환경 변수의 **DB_ENV_POSTGRESQL_PASSWORD**를 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 **POSTGRESQL_PASSWORD**를 사용할 것이기 때문에 **DB_ENV_POSTGRESQL _PASSWORD**가 됩니다.
- host: || 연산자를 이용하여 개발 환경과 데이터베이스 컨테이너에서 사용할 데이터베이스 호스트를 각각 설정합니다. 개발을 할 때는 환경 변수의 **POSTGRESQL_HOST**에 데이터베이스 컨테이너의 IP 주소를 설정합니다. 그리고 데이터베이스 컨테이너를 연결할 때는 별칭을 **db**로 할 것이므로 **db**를 설정합니다.

2. Rails Dockerfile 작성하기

Ruby와 Rails 설치가 끝났으니 예제 Rails 애플리케이션을 생성합니다.

MySQL 사용하기

```
~$ rails new exampleapp --database=mysql
```

- dockerbook/Chapter17/exampleapp_mysql/config/database.yml

~/exampleapp/config/database.yml

```
default: &default
  adapter: mysql2
  encoding: utf8
  pool: 5
  username: root
  password: <%= ENV['DB_ENV_MYSQL_ROOT_PASSWORD'] %>
  host: <%= ENV['MYSQL_HOST'] || 'db' %>
```

Unicorn을 사용하기 위해 exampleapp 디렉터리 아래에 있는 Gemfile을 열고 **gem 'unicorn'** 부분의 주석을 해제합니다.

- dockerbook/Chapter17/exampleapp/Gemfile

~/exampleapp/Gemfile

```
gem 'unicorn'
```

2. Rails Dockerfile 작성하기

다음 내용을 Dockerfile로 저장합니다.

~/exampleapp/Dockerfile

```
FROM ubuntu:14.04

RUN apt-get update
RUN apt-get install -y autoconf bison build-essential \
    libssl-dev libyaml-dev libreadline6-dev zlib1g-dev libncurses5-dev git
RUN apt-get install -y nginx nodejs curl libpq-dev

RUN git clone https://github.com/sstephenson/rbenv.git /root/.rbenv
RUN git clone https://github.com/sstephenson/ruby-build.git \
    /root/.rbenv/plugins/ruby-build
ENV PATH /root/.rbenv/bin:/root/.rbenv/shims:$PATH

ENV CONFIGURE_OPTS --disable-install-doc
RUN rbenv install 2.1.3
RUN rbenv global 2.1.3
RUN rbenv init -

RUN echo 'gem: --no-rdoc --no-ri' >> /root/.gemrc
RUN gem install bundler
RUN rbenv rehash

RUN echo "daemon off;" >> /etc/nginx/nginx.conf
RUN rm -rf /etc/nginx/sites-enabled/default
ADD exampleapp.conf /etc/nginx/sites-enabled/exampleapp.conf

WORKDIR /tmp
ADD Gemfile Gemfile
ADD Gemfile.lock Gemfile.lock
RUN bundle install

ADD ./ /var/www/exampleapp
WORKDIR /var/www/exampleapp
RUN chmod +x entrypoint.sh

EXPOSE 80

ENTRYPOINT ./entrypoint.sh
```

여기서는 우분투 14.04에 apt-get으로 필요한 패키지를 설치하도록 구성하였습니다.

- FROM으로 **ubuntu:14.04**를 기반으로 이미지를 생성하도록 설정합니다.
- `apt-get update`로 패키지 목록을 최신 상태로 업데이트한 뒤 Git과 Ruby 설치에 필요한 패키지를 설치합니다. 그리고 **nginx**, **nodejs**, **libpq-dev** 패키지도 설치합니다.
  - **nodejs**는 JavaScript로 작성된 gem을 실행하기 위해 필요합니다.
  - **libpq-dev**는 PostgreSQL gem을 설치하기 위해 필요합니다.
- `git` 명령으로 rbenv를 **/root/.rbenv** 디렉터리에 받습니다.
- `git` 명령으로 ruby-build를 **/root/.rbenv/plugins/ruby-build** 디렉터리에 받습니다.
- ENV로 환경 변수 **PATH**에 rbenv 경로를 추가합니다.
- `rbenv` 명령으로 Ruby를 설치합니다.
  - ENV로 환경 변수 **CONFIGURE_OPTS**에 **-disable-install-doc** 옵션을 추가하여 문서를 설치하지 않습니다.
  - `rbenv install` 명령으로 Ruby 2.1.3 버전을 설치합니다.
  - `rbenv global` 명령으로 Ruby 2.1.3 버전을 모든 계정에서 사용하도록 설정합니다.
  - `rbenv init -` 명령을 실행하여 rbenv를 초기화합니다.
- `bundler`를 설치합니다.
  - **.gemrc** 파일에 **-no-rdoc**, **-no-ri** 옵션을 추가하여 gem을 설치할 때 문서와 ri(문서 도구)를 설치하지 않습니다.
  - gem 명령으로 bundler를 설치하고 `rbenv rehash` 명령을 실행합니다.
- Nginx를 데몬이 아닌 foreground로 실행하도록 설정합니다. Nginx를 데몬 상태로 실행하면 Docker 컨테이너가 바로 정지되므로 주의합니다.
- **/etc/nginx/sites-enabled** 디렉터리에 있는 nginx 기본 설정 파일(default)을 삭제하고, **exampleapp.conf** 파일을 추가합니다.
- **/tmp** 디렉터리에 Gemfile, Gemfile.lock 파일을 추가한 뒤 `bundle install` 명령으로 gem 파일을 설치합니다.
- Rails 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.

2. Rails Dockerfile 작성하기

다음 내용을 Dockerfile로 저장합니다.

Docker 이미지 생성 시간 줄이기

다음과 같이 Rails 애플리케이션 디렉터리(**exampleapp**)를 추가한 뒤 `bundle install` 명령을 실행하면 Rails 애플리케이션의 .rb 파일이나 기타 파일이 바뀔 때마다 gem 파일을 다시 설치하게 됩니다. Dockerfile은 ADD로 추가했던 디렉터리 안의 파일이 바뀌면 그 뒤에 오는 명령을 다시 실행하기 때문입니다.

```
ADD ./ /var/www/exampleapp
WORKDIR /var/www/exampleapp
RUN bash -l -c "bundle install"
```

다음과 같이 Rails 애플리케이션 디렉터리를 추가하기 전에 **Gemfile**, **Gemfile.lock** 파일만 따로 추가하여 `bundle install` 명령을 실행하면 .rb 파일 등이 바뀌어도 gem 파일을 다시 설치하지 않습니다.

```
WORKDIR /tmp
ADD Gemfile Gemfile
ADD Gemfile.lock Gemfile.lock
RUN bundle install

ADD ./ /var/www/exampleapp
```

Dockerfile의 캐시 기능을 활용하기 위해 자주 변경되지 않고, 시간이 오래 걸리는 부분은 따로 빼서 위로 올립니다.

MySQL 사용하기

- dockerbook/Chapter17/exampleapp_mysql/Dockerfile

~/exampleapp/Dockerfile

```
RUN apt-get install -y nginx nodejs curl libmysqlclient-dev
```

2. Rails Dockerfile 작성하기

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

~/exampleapp/exampleapp.conf

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

server {
  listen 80;
  server_name _;
  root /var/www/exampleapp/pubic;

  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://unicorn;
      break;
    }
  }
}
```

- upstream 항목에는 Unicorn의 유닉스 소켓 /tmp/unicorn.sock을 설정합니다.
- server 항목을 설정합니다.
  - listen 80을 설정하여 80번 포트를 사용합니다.
  - Nginx는 정적 파일(html, js, css 등)을 전송할 것이므로 Rails 애플리케이션 디렉터리아래의 public 디렉터리를 설정합니다.
  - Nginx로 들어온 요청이 파일명이 아니면 앞에서 설정한 유닉스 소켓(http://unicorn)으로 보냅니다. 이렇게 설정하면 정적 파일은 Nginx가 전송하고, 나머지 RESTful API는 Rails가 처리하게 됩니다.

2. Rails Dockerfile 작성하기

다음 내용을 entrypoint.sh로 저장합니다.

~/exampleapp/entrypoint.sh

```
#!/bin/bash

RAILS_ENV=${RAILS_ENV:-"development"}

bundle exec unicorn -D -c unicorn.rb
nginx
```

- 환경 변수 RAILS_ENV는 `docker run` 명령의 `-e` 옵션으로 설정한 값이 있으면 그 값을 사용하고 없으면 development를 사용합니다.
- `unicorn`에 `-D` 옵션을 사용하여 데몬 모드로 실행하고, `-c` 옵션을 사용하여 설정 파일로 unicorn.rb 파일을 사용합니다.
- 앞에서 nginx.conf에 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 rails .
```

3. PostgreSQL 데이터베이스 Dockerfile 작성하기

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

```
~$ mkdir postgresql
~$ cd postgresql
```

~/postgresql/Dockerfile

```
FROM ubuntu:14.04

RUN apt-get update
RUN apt-get install -y postgresql-9.3

WORKDIR /etc/postgresql/9.3/main
RUN sed -i "s/#listen_addresses = 'localhost'/listen_addresses = '*'/g" postgresql.conf
RUN echo "host all all 0.0.0.0/0 password" >> pg_hba.conf

EXPOSE 5432

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

ENTRYPOINT /entrypoint.sh
```

- `apt-get update`로 패키지 목록을 최신 상태로 업데이트한 뒤 postgresql-9.3 패키지를 설치합니다.
- `sed`로 /etc/postgresql/9.3/main 디렉터리의 postgresql.conf 파일 내용을 수정합니다. #listen_addresses = 'localhost' 부분을 listen_addresses = '*'으로 수정합니다. 이 부분을 수정하지 않으면 외부에서 PostgreSQL에 접속할 수 없습니다.
- pg_hba.conf 파일에 host all all 0.0.0.0/0 password를 추가하여 외부에서 접속했을 때는 비밀번호로 인증하도록 설정합니다.
- entrypoint.sh 파일을 추가한 뒤 실행할 수 있도록 권한을 설정합니다.
- EXPOSE에 5432을 설정하여 5432번 포트에 접속할 수 있도록 합니다.
- ENTRYPOINT에 /entrypoint.sh 파일을 설정하여 컨테이너가 시작되었을 때 스크립트 파일을 실행합니다.

3. PostgreSQL 데이터베이스 Dockerfile 작성하기

다음 내용을 entrypoint.sh로 저장합니다.

~/postgresql/entrypoint.sh

```
#!/bin/bash

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

POSTGRESQL_BIN=/usr/lib/postgresql/9.3/bin/postgres
POSTGRESQL_CONFIG_FILE=/etc/postgresql/9.3/main/postgresql.conf

POSTGRESQL_SINGLE="sudo -u postgres $POSTGRESQL_BIN --single --config-file=$POSTGRESQL_CONFIG_FILE"
$POSTGRESQL_SINGLE <<< "ALTER USER postgres PASSWORD '$POSTGRESQL_PASSWORD';" > /dev/null

exec sudo -u postgres $POSTGRESQL_BIN --config-file=$POSTGRESQL_CONFIG_FILE
```

- 환경 변수에 POSTGRESQL_PASSWORD가 없으면 데이터베이스를 실행하지 않고 빠져나옵니다.
- `postgres`를 싱글 모드로 실행한 뒤 postgres 계정의 비밀번호를 설정합니다. 비밀번호는 환경 변수의 POSTGRESQL_PASSWORD를 사용합니다. Dockerfile에서 비밀번호를 설정하지 않고 이곳에서 비밀번호를 설정하는 이유는 `docker run` 명령의 `-e` 옵션으로 비밀번호를 설정하기 위해서입니다. `-e` 옵션으로 설정한 환경 변수 값은 CMD, ENTRYPOINT에서만 사용할 수 있습니다.
- `postgres`를 실행합니다. Nginx와 마찬가지로 PostgreSQL도 foreground로 실행합니다.

`docker build` 명령으로 이미지를 생성합니다.

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

4. Rails와 데이터베이스 컨테이너 생성하기

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

```
$ sudo docker run -d --name db -e POSTGRESQL_PASSWORD=examplepassword postgresql
```

- 데이터베이스 컨테이너를 생성할 때 `-e` 옵션을 사용하여 POSTGRESQL_PASSWORD에 사용할 postgres 계정의 비밀번호를 설정합니다.

Rails 애플리케이션 디렉터리로 이동한 뒤 Rails 데이터베이스를 초기화합니다.

```
~$ export POSTGRESQL_HOST=$(sudo docker inspect -f "{{ .NetworkSettings.IPAddress }}" db)
~$ export DB_ENV_POSTGRESQL_PASSWORD=examplepassword
~$ export RAILS_ENV=development
~$ cd
~$ cd exampleapp
~/exampleapp$ rake db:create
```

- `export` 명령을 사용하여 환경 변수의 POSTGRESQL_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_POSTGRESQL_PASSWORD에 PostgreSQL데이터베이스 비밀번호를 설정합니다.
- `export` 명령을 사용하여 환경 변수의 RAILS_ENV에 development를 설정합니다(각자 상황에 따라 production, test를 설정합니다).
- `rake db:create`를 실행하여 Rails 데이터베이스를 초기화합니다.

MySQL 사용하기

```
~$ export MYSQL_HOST=$(sudo docker inspect -f "{{ .NetworkSettings.IPAddress }}" db)
~$ export DB_ENV_MYSQL_ROOT_PASSWORD=examplepassword
~$ export RAILS_ENV=development
~$ cd
~$ cd exampleapp
~/exampleapp$ rake db:create
```

4. Rails와 데이터베이스 컨테이너 생성하기

Rails 컨테이너를 생성합니다.

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

- Rails 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.