Docker Remote API 사용하기

목차

Docker Remote API 사용하기

Docker는 데몬과 클라이언트 간의 통신을 할 때 로컬에서는 유닉스 소켓을 사용하고, 원격에서는 TCP 소켓을 사용합니다. 여기에 HTTP REST 형식으로 API가 구현되어 있습니다. 따라서 API가 특정 언어에 종속되어 있지 않고, 다양한 언어에서 사용할 수 있습니다.
먼저 기존의 Docker 데몬을 정지하고 TCP 소켓으로 다시 실행시켜 API를 테스트해보겠습니다.

```
$ sudo service docker stop
$ sudo docker -d -H tcp://0.0.0.0:4243
```

`docker -d -H tcp://0.0.0.0:4243` 형식입니다. `-d` 옵션을 사용하여 데몬 모드로 실행하고, `-H` 옵션을 사용하여 접속을 받을 IP 주소와 포트 번호를 설정합니다. Docker 데몬의 기본 포트 번호는 **4243**입니다.

명령을 실행하면 Docker 데몬이 foreground로 실행됩니다. 따라서 테스트를 위해 새 터미널을 실행합니다.

Docker Remote API는 HTTP REST 형식이므로 `curl` 명령으로 손쉽게 사용할 수 있습니다. 먼저 **nginx** 이미지를 받습니다.

```
$ curl -X POST http://127.0.0.1:4243/images/create?fromImage=nginx:latest
```

- `-X` : POST 메서드를 사용하도록 설정합니다.
- 이미지를 받는 Remote API는 **/images/create?fromImage=<이미지 이름>:<태그>**입니다.

Docker Remote API 사용하기

받은 **nginx** 이미지로 컨테이너를 생성합니다.

```
$ curl -X POST -H "Content-Type: application/json" \
    -d '{ "Image":"nginx:latest", "ExposedPorts":"80/tcp" }' \
    http://127.0.0.1:4243/containers/create
{"Id":"386147f1c71ca7ee6a7013dbd2fff5d3091e41268885f954046be964b9ade39e","Warnings":null}
```

- `-X` : POST 메서드를 사용하도록 설정합니다.
- `-H` : Content-Type을 **application/json**으로 사용하도록 설정합니다.
- `-d` : 컨테이너를 생성하기 위해 설정 데이터를 보냅니다. 여기서는 **Image**에 **nginx:latest**를 설정하였고, 호스트에 포트를 연결하기 위해 **ExposedPorts**에 **80/tcp**를 설정하였습니다.
- 컨테이너를 생성하는 Remote API는 **/containers/create**입니다.

컨테이너가 생성되면 컨테이너 ID가 출력됩니다.

이제 생성된 컨테이너를 시작합니다.

```
$ curl -X POST -H "Content-Type: application/json" \
    -d '{ "PortBindings": {"80/tcp": [{ "HostPort": "80"}]} }' \
    http://127.0.0.1:4243/containers/386147f1c71c/start
```

- `-X` : POST 메서드를 사용하도록 설정합니다.
- `-H` : Content-Type을 **application/json**으로 사용하도록 설정합니다.
- `-d` : 컨테이너의 80번 포트를 외부에 노출합니다. 여기서는 **PortBindings**에 **{“80/tcp”: [{ “HostPort”: “80”}]}**를 설정하였습니다.
- 컨테이너를 시작하는 Remote API는 **/containers/<컨테이너 ID>/start**입니다. 컨테이너 ID는 앞에서 컨테이너를 생성할 때 출력된 값을 사용합니다(12자리).

Docker Remote API 사용하기

컨테이너 목록을 출력합니다.

```
$ curl http://127.0.0.1:4243/containers/json
[{"Command":"nginx","Created":1409246048,"Id":"386147f1c71ca7ee6a7013dbd2fff5d3091e41268885f954046be964b9ade39
e","Image":"nginx:1","Names":["/angry_bartik"],"Ports":[{"IP":"0.0.0.0","PrivatePort":80,"PublicPort":80,"Typ
e":"tcp"}],"Status":"Up 6 seconds"}
```

컨테이너 목록을 출력하는 Remote API는 GET 메서드에 **/containers/json**입니다. 정지된 컨테이너까지 출력하려면 **/containers/json?all=1**처럼 **all=1**을 붙여주면 됩니다.

이처럼 HTTP 메서드를 이용하여 Docker 데몬을 제어할 수 있습니다. HTTP REST 형식으로 된 Remote API를 그대로 사용하여 구현하는 일은 드물 것이라 생각됩니다. 또한, Remote API는 빠르게 업그레이드되고 있기 때문에 전체 API는 책에 담지 않았습니다. Remote API 목록은 다음 URL을 참조하기 바랍니다.

1. Docker Remote API Python 라이브러리 사용하기

앞에서 HTTP 메서드를 이용하여 Remote API를 사용해보았습니다. 이 HTTP REST 형식의 Remote API를 그대로 사용해도 되지만, 이미 각 언어별로 래핑한 Remote API 클라이언트 라이브러리가 나와 있습니다. 이 책에서는 Docker에서 공식적으로 배포하는 Python 라이브러리를 소개하겠습니다.

- 언어별 라이브러리: https://docs.docker.com/reference/api/remote_api_client_libraries
- Python 라이브러리: https://github.com/docker/docker-py

예제 파일은 저의 GitHub 저장소에서 받을 수 있습니다.

- https://github.com/pyrasis/dockerbook

Docker Remote API Python 라이브러리는 pip를 이용하여 설치할 수 있습니다. 먼저 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
```

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
```

이제 `pip`로 docker-py 패키지를 설치합니다.

```
$ sudo pip install docker-py
```

1. Docker Remote API Python 라이브러리 사용하기

1. 컨테이너 생성 및 시작하기

다음 내용을 docker-run.py로 저장합니다. nginx 이미지를 받고 컨테이너로 실행하는 예제입니다

docker-run.py

```
import docker

c = docker.Client(base_url='unix://var/run/docker.sock')
c.pull(repository='nginx', tag='latest')
container_id = c.create_container(
  image='nginx:latest',
  ports=[80],
  volumes=['/data'],
  name='hello'
)
c.start(
  container_id,
  port_bindings={80: ('0.0.0.0', 80)},
  binds={'/data': {'bind': '/data', 'ro': False}}
)
```

- **docker.Client** 클래스를 생성합니다. **base_url**에는 Docker 데몬의 유닉스 소켓 경로를 설정합니다.
- **docker.Client** 클래스로 생성한 인스턴스 **c**로 **pull** 함수를 실행합니다. 이미지 이름과 태그를 설정합니다.
- **c.create_container** 함수로 컨테이너를 생성합니다. 이미지 이름, 호스트에 연결할 포트, 호스트와 연결할 디렉터리, 컨테이너 이름을 설정합니다.
- **c.start** 함수로 컨테이너를 시작합니다. **c.create_container** 함수를 실행하고 받은 컨테이너 객체를 사용하고, 컨테이너 포트 번호와 외부에 노출할 포트 번호, 호스트 디렉터리와 연결할 컨테이너 디렉터리를 설정합니다.

1. Docker Remote API Python 라이브러리 사용하기

1. 컨테이너 생성 및 시작하기

이제 docker-run.py 파일을 실행합니다.

```
$ sudo python docker-run.py
```

컨테이너 목록을 출력하면 Python 라이브러리로 생성한 컨테이너가 표시됩니다.

```
$ sudo docker ps
CONTAINER ID  IMAGE     COMMAND  CREATED     STATUS       PORTS               NAMES
fda50393ec9f  nginx:1   "nginx"  1 sec...    Up 1 sec...  0.0.0.0:80->80/tcp  hello
```

docker.Client 클래스: Docker 명령을 실행할 기본 객체를 생성합니다.

- base_url: Docker 데몬 주소와 포트 번호입니다.
  - 유닉스 소켓: **unix://**
  - 일반 HTTP 프로토콜: **tcp://**, 포트 번호 4243
  - 인증서를 사용한 HTTPS 프로토콜(TLS): **https://**, 포트 번호 2376
- version: Docker Remote API의 특정 버전을 사용합니다. Docker 실행 파일의 버전과는 다릅니다. https://docs.docker.com/reference/api/docker_remote_api/
- timeout: 접속 대기 시간이며 초 단위입니다.

```
c = docker.Client(base_url='unix://var/run/docker.sock',
                  version='1.12',
                  timeout=10)
```

1. Docker Remote API Python 라이브러리 사용하기

1. 컨테이너 생성 및 시작하기

pull 함수: 저장소에서 이미지를 받아옵니다.

- repository: 받을 이미지 이름이며 '**<Docker Hub 사용자 계정>/<이미지 이름>**' 또는 '**<이미지 이름>**' 형식입니다.
- tag: 이미지의 태그입니다.
- stream: 결과 값을 받을 때 HTTP 1.1 Chunked transfer encoding을 사용합니다.

```
c.pull(repository, tag=None, stream=False)
```

1. Docker Remote API Python 라이브러리 사용하기

1. 컨테이너 생성 및 시작하기

create_container 함수: 컨테이너를 생성합니다.

- image: 컨테이너로 생성할 이미지 이름이며 '<Docker Hub 사용자 계정>/<이미지 이름>' 또는 '<이미지 이름>' 형식입니다.
- command: 컨테이너가 시작했을 때 실행할 명령입니다. 예) *'/bin/bash'* 또는 *['node', 'app.js']*
- hostname: 컨테이너의 호스트 이름입니다.
- user: command, entrypoint에 설정한 명령을 컨테이너 안의 특정 사용자로 실행합니다. 사용자명이나 UID를 설정할 수 있습니다.
- detach: 데몬 모드로 실행합니다. `docker run` 명령의 `-d` 옵션입니다.
- stdin_open: 기본 입력(stdin)을 연결합니다. `docker run` 명령의 `-i` 옵션입니다.
- tty: pseudo tty를 할당합니다. `docker run` 명령의 `-t` 옵션입니다.
- mem_limit: 메모리 한계를 설정합니다. 예) *'100000b', '1000k', '128m', '1g'*
- ports: 호스트에 연결할 포트입니다.
  - 예) *[80, 443]*
  - 예) *[(100, 'udp'), 200]*
- environment: 환경 변수입니다. 예) *['Hello=1', 'Workd=2']* 또는 *{'Hello': '1', 'World': '2'}*
- volumes: 호스트와 연결할 디렉터리입니다(데이터 볼륨). 예) *['/data', '/www']*
- network_disabled: 네트워크를 비활성화합니다.
- name: 컨테이너 이름입니다.
- entrypoint: 컨테이너가 시작했을 때 실행할 명령입니다. command와 entrypoint의 차이점은 '7.6 ENTRYPOINT'를 참조하기 바랍니다. 예) *'/bin/bash'* 또는 *['node', 'app.js']*
- cpu_shares: CPU 자원을 할당할 때 가중치 설정이며 숫자 형식입니다..
- working_dir: command, entrypoint에 설정한 명령이 실행될 디렉터리입니다.
- memswap_limit: 메모리 스왑 한계 설정이며 숫자 형식입니다.

```
c.create_container(image, command=None, hostname=None, user=None,
                   detach=False, stdin_open=False, tty=False, mem_limit=0,
                   ports=None, environment=None, dns=None, volumes=None,
                   network_disabled=False, name=None, entrypoint=None,
                   cpu_shares=None, working_dir=None, memswap_limit=0)
```

1. Docker Remote API Python 라이브러리 사용하기

1. 컨테이너 생성 및 시작하기

start 함수: 컨테이너를 시작합니다.

- container: 시작할 컨테이너의 객체 또는 이름입니다.
- binds: 컨테이너의 디렉터리를 호스트의 특정 디렉터리에 연결합니다. **create_container** 함수에서 **volumes**를 설정해야 합니다. **ro**는 읽기 전용 옵션입니다.
  - 예) *{'/data': {'bind': '/data', 'ro': False}, '/www': {'bind': '/www', 'ro': False}}*
- port_bindings: 호스트에 연결된 포트를 외부에 노출합니다.
  - 예) *{80: ('0.0.0.0', 80), 443: ('0.0.0.0', 443)}*
  - 예) *{80: ('0.0.0.0',), 443: ('0.0.0.0',)}*
  - 예) *{'100/udp': 100, 200: None}*
  - 예) *{80: 80, 443: None}*
- lxc_conf: LXC 드라이버를 사용했을 때의 옵션 설정입니다.
- publish_all_ports: 호스트에 연결된 모든 포트를 외부에 노출합니다. `docker run` 명령의 `-P` 옵션입니다.
- links: 컨테이너를 연결하는 옵션입니다. 예) *{'db': 'db', 'hello': 'hello'}*
- privileged: 호스트 커널의 기능을 모두 사용할 수 있도록 설정합니다.
- dns: 추가 DNS 서버 설정입니다. 예) *['168.126.63.1', '192.168.0.100']*
- dns_search: 추가 검색 도메인 설정입니다. 예) *['hello.com', 'world.com']*
- volumes_from: 데이터 볼륨 컨테이너 설정입니다. 예) *['hello-volume', 'world-volume']*
- network_mode: 네트워크 모드 설정입니다. 예) *'bridge', 'none', 'host'*
- restart_policy: 컨테이너가 종료되었을 때 자동으로 재시작할지 설정합니다.
  - 종료되었을 때 항상 재시작: *{'MaximumRetryCount': 0, 'Name': 'always'}*
  - 종료되었을 때 10번까지 재시도: *{'MaximumRetryCount': 10, 'Name': 'on-failure'}*

```
c.start(container, binds=None, port_bindings=None, lxc_conf=None,
        publish_all_ports=False, links=None, privileged=False, dns=None,
        dns_search=None, volumes_from=None, network_mode=None, restart_policy=None)
```

1. Docker Remote API Python 라이브러리 사용하기

2. 이미지 생성하기

다음 내용을 docker-run.py로 저장합니다. nginx 이미지를 받고 컨테이너로 실행하는 예제입니다.

docker-build.py

```
import docker
import io

script = io.StringIO(u'\n'.join([
  'FROM ubuntu:14.04',
  'MAINTAINER Foo Bar <foo@bar.com>',
  'RUN apt-get update',
  'RUN apt-get install -y nginx',
  'RUN echo "\\ndaemon off;" >> /etc/nginx/nginx.conf',
  'RUN chown -R www-data:www-data /var/lib/nginx',
  'VOLUME ["/data", "/etc/nginx/site-enabled", "/var/log/nginx"]',
  'WORKDIR /etc/nginx',
  'CMD ["nginx"]',
  'EXPOSE 80',
  'EXPOSE 443'
]))

c = docker.Client(base_url='unix://var/run/docker.sock')
c.build(tag='hello:0.1', quiet=True, fileobj=script, rm=True)
```

- **io.StringIO** 함수를 사용하여 Dockerfile의 내용을 생성합니다.
- **docker.Client** 클래스를 생성합니다. **base_url**에는 Docker 데몬의 유닉스 소켓 경로를 설정합니다.
- **docker.Client** 클래스로 생성한 인스턴스 **c**로 **build** 함수를 실행합니다. 이미지 이름과 태그를 설정합니다. 이미지를 생성하면서 출력되는 결과를 표시하지 않도록 **quiet=True**로 설정하고, 이미지 생성을 완료했을 때 임시 이미지를 삭제하도록 **rm=True**로 설정합니다. **fileobj**에는 앞에서 생성한 **script**를 설정합니다.

1. Docker Remote API Python 라이브러리 사용하기

2. 이미지 생성하기

이제 docker-build.py 파일을 실행합니다.

```
$ sudo python docker-build.py
```

이미지 목록을 출력하면 Python 라이브러리로 생성한 이미지가 표시됩니다.

```
$ sudo docker ps
REPOSITORY    TAG      IMAGE ID       CREATED              VIRTUAL SIZE
hello         0.1      8da15999194f   About an hour ago    263.8 MB
ubuntu        14.04    c4ff7513909d   2 weeks ago          225.4 MB
```

1. Docker Remote API Python 라이브러리 사용하기

2. 이미지 생성하기

build 함수: 이미지를 생성합니다.

- path: Dockerfile의 경로입니다. 디렉터리 경로뿐만 아니라 **http://**, **https://**, **git://**을 사용할 수 있습니다.
- tag: 이미지 이름 및 태그입니다.
- quiet: 이미지를 생성하면서 출력되는 결과를 표시하지 않습니다.
- fileobj: Docker 파일을 객체 형식으로 설정합니다. **fileobj**를 설정하면 **path**는 무시됩니다.
- nocache: 빌드 결과를 캐시하지 않습니다.
- rm: 이미지 생성을 완료했을 때 임시 이미지를 삭제합니다.
- stream: 결과 값을 받을 때 HTTP 1.1 Chunked transfer encoding을 사용합니다.
- timeout: 접속 대기 시간이며 초 단위입니다.
- custom_context: tar, tar.gz 등의 파일을 읽어서 사용할 때 **True**로 설정합니다. 파일 내용은 **fileobj**에 설정해야 합니다.
- encoding: **custom_context**가 압축이 되어 있으면 압축 형식을 지정합니다. 예) *'gzip'*

```
c.build(path=None, tag=None, quiet=False, fileobj=None, nocache=False,
        rm=False, stream=False, timeout=None,
        custom_context=False, encoding=None)
```

1. Docker Remote API Python 라이브러리 사용하기

2. 이미지 생성하기

다음은 현재 디렉터리에 Dockerfile이 있을 때 이미지를 생성하는 예제입니다.

docker-build-local.py

```
import docker

c = docker.Client(base_url='unix://var/run/docker.sock')
c.build(path='.', tag='hello:0.1', quiet=True, rm=True)
```

다음은 Dockerfile 및 필요한 파일을 hello.tar.gz로 압축했을 때 이미지를 생성하는 예제입니다.

docker-build-gzip.py

```
import docker

script = open('./hello.tar.gz', 'r')

c = docker.Client(base_url='unix://var/run/docker.sock')
c.build(tag='hello:0.1', quiet=True, fileobj=script,
        rm=True, custom_context=True, encoding='gzip')
```

1. Docker Remote API Python 라이브러리 사용하기

2. 이미지 생성하기

다음은 GitHub 또는 웹에서 Dockerfile을 받아서 이미지를 생성하는 예제입니다. 단 **http://**로 파일을 받을 때는 tar, tar.gz 파일은 사용할 수 없습니다.

docker-build-remote.py

```
import docker

c = docker.Client(base_url='unix://var/run/docker.sock')
c.build(
  path='github.com/pyrasis/dind.git',      # GitHub
  #path='http://example.com/Dockerfile',  # http
  tag='hello:0.1', quiet=True, rm=True
)
```

1. Docker Remote API Python 라이브러리 사용하기

3. 컨테이너 목록 출력하기

다음 내용을 docker-ps.py로 저장합니다. 컨테이너 목록을 출력하는 예제입니다.

docker-ps.py

```
import docker

c = docker.Client(base_url='unix://var/run/docker.sock')
print c.containers(all=True)
```

- **docker.Client** 클래스를 생성합니다. **base_url에**는 Docker 데몬의 유닉스 소켓 경로를 설정합니다.
- **docker.Client** 클래스로 생성한 인스턴스 **c**로 **containers** 함수를 실행합니다. 모든 컨테이너 목록을 출력하도록 **all=True**로 설정합니다.

docker-ps.py 파일을 실행하면 다음과 같이 JSON 형태로 컨테이너 목록이 출력됩니다.

```
$ sudo python docker-ps.py
[{u'Status': u'Up 11 seconds', u'Created': 1409387349, u'Image': u'nginx:latest', u'Ports': [{u'IP': u'0.0.0.0',
u'Type': u'tcp', u'PublicPort': 80, u'PrivatePort': 80}], u'Command': u'nginx', u'Names': [u'/hello'], u'Id': u'6
c70cda2b562c764c8d3cb605bdd2734fe32baf5c5ca6a7076180bf49297566e'}]
```

containers 함수: 컨테이너 목록을 출력합니다.

- quiet: 컨테이너 ID만 출력합니다.
- all: 정지된 컨테이너를 포함해서 모든 컨테이너를 출력합니다.
- trunc: 긴 출력 결과를 일부만 표시합니다.
- latest: 태그가 **latest**인 것만 출력합니다.
- since: 특정 컨테이너 이후에 생성된 컨테이너를 출력합니다.
- before: 특정 컨테이너 이전에 생성된 컨테이너를 출력합니다.
- limit: 컨테이너를 출력할 때 최대 개수입니다. -1은 모든 컨테이너를 출력합니다.

```
c.containers(quiet=False, all=False, trunc=True, latest=False, since=None,
             before=None, limit=-1)
```

1. Docker Remote API Python 라이브러리 사용하기

4. 이미지 목록 출력하기

다음 내용을 docker-images.py로 저장합니다. 이미지 목록을 출력하는 예제입니다.

docker-images.py

```
import docker

c = docker.Client(base_url='unix://var/run/docker.sock')
print c.images()
```

docker-images.py 파일을 실행하면 다음과 같이 JSON 형태로 이미지 목록이 출력됩니다.

```
$ sudo python docker-images.py
[{u'Created': 1407814247, u'VirtualSize': 225406429, u'ParentId': u'cc58e55aa5a53b572f3b9009eb07e50989553b95a1545
a27dcec830939892dba', u'RepoTags': [u'ubuntu:14.04'], u'Id': u'c4ff7513909dedf4ddf3a450aea68cd817c42e698ebccf5475
5973576525c416', u'Size': 0}, {u'Created': 1405934538, u'VirtualSize': 499135815, u'ParentId': u'5b9d57417804c431
880c93e1adcc41afe6bf6451309680 3f03840117688f921a', u'RepoTags': [u'nginx:latest'], u'Id': u'61e8f94e1d65cf3f2f409
c70ecbc4401a9c5db83e9cfbf82c4e595e44a890376', u'Size': 0}]
```

images 함수: 이미지 목록을 출력합니다.

- name: 특정 이름을 가진 이미지만 출력합니다.
- quiet: 이미지 ID만 출력합니다.
- all: 임시 이미지를 포함한 모든 이미지를 출력합니다.

```
c.images(name=None, quiet=False, all=False)
```

1. Docker Remote API Python 라이브러리 사용하기

5. 기타 예제 및 함수

attach 함수: 컨테이너에 연결합니다.

- container: 연결할 컨테이너의 객체 또는 이름입니다.
- stdout: 표준 출력(stdout)을 연결합니다.
- stderr: 표준 에러(stderr)를 연결합니다.
- stream: 결과 값을 받을 때 HTTP 1.1 Chunked transfer encoding을 사용합니다.
- logs: 로그를 출력합니다.

```
c.attach(container, stdout=True, stderr=True, stream=False, logs=False)
```

Python으로 입출력을 구현하려면 dockerpty를 사용하면 됩니다.

- https://github.com/d11wtq/dockerpty

```
~$ git clone https://github.com/d11wtq/dockerpty.git
~$ cd dockerpty
~/dockerpty$ sudo python setup.py install
```

다음은 dockerpty를 사용하여 터미널에서 입출력을 하는 예제입니다.

docker-attach.py

```
import docker
import dockerpty

c = docker.Client(base_url='unix://var/run/docker.sock')
container_id = c.create_container(
  image='ubuntu:14.04',
  stdin_open=True,
  tty=True,
  command='/bin/bash'
)

dockerpty.start(c, container_id)
```

1. Docker Remote API Python 라이브러리 사용하기

5. 기타 예제 및 함수: commit

commit 함수: 컨테이너를 이미지로 저장합니다.

- container: 커밋 기능을 사용할 컨테이너의 객체 또는 이름입니다.
- repository: 생성할 이미지 이름이며 '<Docker Hub 사용자 계정>/<이미지 이름>' 또는 '<이미지 이름>' 형식입니다.
- tag: 생성할 이미지의 태그입니다.
- message: 커밋 로그 메시지입니다.
- author: 이미지를 생성한 사람의 정보입니다. 예) *'Hong, Gildong gd@yuldo.com'*
- conf: 이미지를 생성할 때 필요한 설정 값입니다.

```
c.commit(container, repository=None, tag=None, message=None, author=None,
        conf=None)
```

docker-commit.py

```
import docker

c = docker.Client(base_url='unix://var/run/docker.sock')
c.pull(repository='nginx', tag='latest')
container_id = c.create_container(
  image='nginx:latest',
  ports=[80],
  volumes=['/data']
)
c.commit(
  container_id,
  repository='hello',
  tag='0.1',
  message='example message',
  author='Hong, Gildong <gd@yuldo.com>',
  conf={
    'Hostname':'',
    'User':'',
    'Memory':0,
    'MemorySwap':0,
    'AttachStdin':False,
    'AttachStdout':False,
    'AttachStderr':True,
    'PortSpecs':None,
    'Tty':False,
    'OpenStdin':False,
    'StdinOnce':False,
    'Env':None,
    'Cmd':[
      '/bin/bash'
    ],
    'Volumes':{
      '/data':{}
    },
    'WorkingDir':'',
    'DisableNetwork':False,
    'ExposedPorts':{
      '80/tcp':{}
    }
  }
)
```

1. Docker Remote API Python 라이브러리 사용하기

5. 기타 예제 및 함수: copy

copy 함수: 컨테이너에서 파일을 가져옵니다.

- container: 파일을 가져올 컨테이너의 객체 또는 이름입니다.
- resource: 컨테이너 안의 파일 경로입니다.
- 리턴값은 urllib3.response.HTTPResponse이며 .data에 tar 형식으로 데이터가 저장되어 있습니다.

```
c.copy(container, resource)
```

docker-cp.py

```
import docker
import tarfile
import io

c = docker.Client(base_url='unix://var/run/docker.sock')
container_id = c.create_container(
  image='nginx:latest',
  ports=[80],
  volumes=['/data']
)
c.start(container_id)
response = c.copy(container_id, '/etc/nginx.conf')
t = tarfile.open(fileobj=io.BytesIO(response.data))
t.extractall();
```

1. Docker Remote API Python 라이브러리 사용하기

5. 기타 예제 및 함수: diff

diff 함수: 이미지와 컨테이너 간의 바뀐 부분을 출력합니다.

- container: 비교할 컨테이너의 객체 또는 이름입니다.

```
c.diff(container)
```

docker-diff.py

```
import docker

c = docker.Client(base_url='unix://var/run/docker.sock')
container_id = c.create_container(
  image='nginx:latest',
  ports=[80],
  volumes=['/data']
)
c.start(container_id)
print c.diff(container_id)
```

1. Docker Remote API Python 라이브러리 사용하기

5. 기타 예제 및 함수: export

export 함수: 컨테이너를 tar 파일로 저장합니다.

- container: tar 파일로 저장할 컨테이너의 객체 또는 이름입니다.

```
c.export(container)
```

docker-export.py

```
import docker

c = docker.Client(base_url='unix://var/run/docker.sock')
container_id = c.create_container(
  image='nginx:latest',
  ports=[80],
  volumes=['/data']
)
c.start(container_id)
response = c.export(container_id)
with open('./nginx.tar', 'wb') as f:
  f.write(response.data)
```

1. Docker Remote API Python 라이브러리 사용하기

5. 기타 예제 및 함수: history

history 함수: 이미지의 히스토리를 출력합니다.

- image: 히스토리를 출력할 이미지 이름이며 '<이미지 이름>:<태그>' 또는 '<이미지 이름>' 입니다.

```
c.history(image)
```

docker-history.py

```
import docker

c = docker.Client(base_url='unix://var/run/docker.sock')
print c.history('nginx:latest')
```

import_image 함수: tar 파일을 사용하여 이미지를 생성합니다.

- src: tar 파일의 경로입니다.
- repository: 생성할 이미지 이름이며 '<Docker Hub 사용자 계정>/<이미지 이름>' 또는 '<이미지 이름> 형식입니다.
- tag: 생성할 이미지의 태그입니다.
- image: 기존에 있는 이미지를 가져옵니다. 예) *'nginx:latest'*

1. Docker Remote API Python 라이브러리 사용하기

5. 기타 예제 및 함수: import_image, info

import_image 함수: tar 파일을 사용하여 이미지를 생성합니다.

- src: tar 파일의 경로입니다.
- repository: 생성할 이미지 이름이며 '<Docker Hub 사용자 계정>/<이미지 이름>' 또는 '<이미지 이름> 형식입니다.
- tag: 생성할 이미지의 태그입니다.
- image: 기존에 있는 이미지를 가져옵니다. 예) *'nginx:latest'*

```
c.import_image(src=None, repository=None, tag=None, image=None)
```

docker-import.py

```
import docker

c = docker.Client(base_url='unix://var/run/docker.sock')
c.import_image(src='./nginx.tar', repository='hello', tag='0.1')
```

info 함수: 현재 Docker의 컨테이너, 이미지 개수, Execution Driver, Storage Driver, 커널 버전 등의 정보를 출력합니다.

```
c.info()
```

docker-info.py

```
import docker

c = docker.Client(base_url='unix://var/run/docker.sock')
print c.info()
```

1. Docker Remote API Python 라이브러리 사용하기

5. 기타 예제 및 함수: inspect_container

inspect_container 함수: 컨테이너의 세부 정보를 출력합니다.

- container: 세부 정보를 출력할 컨테이너의 객체 또는 이름입니다.

```
c.inspect_container(container)
```

docker-inspect-container.py

```
import docker

c = docker.Client(base_url='unix://var/run/docker.sock')
container_id = c.create_container(
  image='nginx:latest',
  ports=[80],
  volumes=['/data']
)
c.start(container_id)
print c.inspect_container(container_id)
```

inspect_image 함수: 이미지의 세부 정보를 출력합니다.

1. Docker Remote API Python 라이브러리 사용하기

5. 기타 예제 및 함수: inspect_image, kill

inspect_image 함수: 이미지의 세부 정보를 출력합니다.

- 세부 정보를 출력할 이미지 ID입니다.

```
c.inspect_image(image_id)
```

docker-inspect-image.py

```
import docker

c = docker.Client(base_url='unix://var/run/docker.sock')
print c.inspect_image('nginx:latest')
```

kill 함수: 컨테이너를 강제 종료합니다.

- container: 종료할 컨테이너의 객체 또는 이름입니다.
- signal: 컨테이너에 특정 시그널을 보냅니다. 예) *'KILL'*

```
c.kill(container, signal=None)
```

docker-kill.py

```
import docker

c = docker.Client(base_url='unix://var/run/docker.sock')
container_id = c.create_container(
  image='nginx:latest',
  ports=[80],
  volumes=['/data']
)
c.start(container_id)
c.kill(container_id)
```

1. Docker Remote API Python 라이브러리 사용하기

5. 기타 예제 및 함수: login

login 함수: Docker Hub에 로그인합니다.

- username: Docker Hub 계정입니다.
- password: Docker Hub 계정의 비밀번호입니다.
- email: Docker Hub에 가입할 때 사용한 이메일 주소입니다.
- registry: 레지스트리 서버 주소입니다. 기본 값은 Docker Hub입니다.

```
c.login(username, password=None, email=None, registry=None)
```

docker-login.py

```
import docker

c = docker.Client(base_url='unix://var/run/docker.sock')
c.login(username='exampleuser',
        password='examplepassword',
        email='exampleuser@example.com')
```

1. Docker Remote API Python 라이브러리 사용하기

5. 기타 예제 및 함수: logs

logs 함수: 컨테이너의 표준 출력, 표준 에러를 출력합니다.

- container: 로그를 출력할 컨테이너의 객체 또는 이름입니다.
- stdout: 표준 출력을 출력합니다.
- stderr: 표준 에러를 출력합니다.
- stream: 결과 값을 받을 때 HTTP 1.1 Chunked transfer encoding을 사용합니다.
- timestamps: 시간값을 출력합니다.

```
c.logs(container, stdout=True, stderr=True, stream=False, timestamps=False)
```

docker-logs.py

```
import docker
import time

c = docker.Client(base_url='unix://var/run/docker.sock')
c.pull('ubuntu', tag='14.04')
container_id = c.create_container(
  image='ubuntu:14.04',
  command='/bin/bash -c "while sleep 1; do echo 1; done"',
)
c.start(container_id)
time.sleep(5)
print c.logs(container_id)
```

1. Docker Remote API Python 라이브러리 사용하기

5. 기타 예제 및 함수: port

port 함수: 컨테이너에서 호스트에 연결되고 외부에 노출된 포트를 조회합니다.

- container: 조회할 컨테이너의 객체 또는 이름입니다.
- private_port: 조회할 포트 번호입니다. 예) *80*

```
c.port(container, private_port)
```

docker-port.py

```
import docker

c = docker.Client(base_url='unix://var/run/docker.sock')
container_id = c.create_container(
  image='nginx:latest',
  ports=[80],
  volumes=['/data']
)
c.start(
  container_id,
  port_bindings={80: ('0.0.0.0', 80)}
)
print c.port(container_id, 80)
```

1. Docker Remote API Python 라이브러리 사용하기

5. 기타 예제 및 함수: push

push 함수: 이미지를 저장소에 올립니다.

- repository: 이미지 이름이며 '<Docker Hub 사용자 계정>/<이미지 이름>' 형식입니다.
- tag: 이미지의 태그입니다.
- stream: 결과 값을 받을 때 HTTP 1.1 Chunked transfer encoding을 사용합니다.

```
c.push(repository, tag=None, stream=False)
```

docker-push.py

```
import docker

c = docker.Client(base_url='unix://var/run/docker.sock')
#c.push(repository='localhost:5000/hello', tag='0.1') # 개인 저장소

c.login(username='exampleuser',
        password='examplepassword',
        email='exampleuser@example.com')
c.push(repository='exampleuser/hello', tag='0.1')     # Docker Hub
```

1. Docker Remote API Python 라이브러리 사용하기

5. 기타 예제 및 함수: remove_container, remove_image

remove_container 함수: 컨테이너를 삭제합니다.

- container: 삭제할 컨테이너의 객체 또는 이름입니다.
- v: 컨테이너와 연결된 데이터 볼륨을 함께 삭제합니다.
- link: `docker run` 명령의 `--link` 옵션으로 컨테이너를 연결했을 때 연결 상태만 삭제합니다. 예) *'web/db'*

```
c.remove_container(container, v=False, link=False)
```

docker-remove-container.py

```
import docker

c = docker.Client(base_url='unix://var/run/docker.sock')
c.remove_container('hello-nginx')
```

remove_image 함수: 이미지를 삭제합니다.

- image: 삭제할 이미지 이름입니다. 예) *'nginx:latest'*

```
c.remove_image(image)
```

```
import docker

c = docker.Client(base_url='unix://var/run/docker.sock')
c.remove_image('nginx:latest')
```

1. Docker Remote API Python 라이브러리 사용하기

5. 기타 예제 및 함수: restart, search

restart 함수: 컨테이너를 재시작합니다.

- container: 재시작할 컨테이너의 객체 또는 이름입니다.
- timeout: 컨테이너가 종료될 때까지 지정한 시간 동안 대기합니다. 초 단위입니다.

```
c.restart(container, timeout=10)
```

docker-restart.py

```
import docker

c = docker.Client(base_url='unix://var/run/docker.sock')
c.restart('hello-nginx')
```

search 함수: Docker Hub에서 이미지를 검색합니다.

- term: 검색할 단어입니다.

```
c.search(term)
```

docker-search.py

```
import docker

c = docker.Client(base_url='unix://var/run/docker.sock')
print c.search('nginx')
```

1. Docker Remote API Python 라이브러리 사용하기

5. 기타 예제 및 함수: stop, tag

stop 함수: 컨테이너를 정지합니다.

- container: 정지할 컨테이너의 객체 또는 이름입니다.
- timeout: 컨테이너가 종료될 때까지 지정한 시간 동안 대기합니다. 초 단위입니다.

```
c.stop(container, timeout=10)
```

docker-stop.py

```
import docker

c = docker.Client(base_url='unix://var/run/docker.sock')
c.stop('hello-nginx')
```

tag 함수: 이미지에 태그를 설정합니다.

- image: 이미지 이름입니다. 예) *'hello:0.1'*
- repository: 레지스트리 서버 주소와 이미지 이름입니다. 예) *'192.168.0.10:5000/hello'*
- tag: 생성할 이미지의 태그입니다.
- force: 강제로 태그를 설정합니다.

```
c.tag(image, repository, tag=None, force=False)
```

docker-tag.py

```
import docker

c = docker.Client(base_url='unix://var/run/docker.sock')
#c.tag('hello:0.1', 'localhost:5000/hello', '0.1')  # 개인 저장소
c.tag('hello:0.1', 'exampleuser/hello', '0.1')      # Docker Hub
```

1. Docker Remote API Python 라이브러리 사용하기

5. 기타 예제 및 함수: top, version

top 함수: 컨테이너에서 현재 실행되고 있는 프로세스 목록을 출력합니다.

- container: 프로세스 목록을 출력할 컨테이너의 객체 또는 이름입니다.

```
c.top(container)
```

docker-top.py

```
import docker

c = docker.Client(base_url='unix://var/run/docker.sock')
print c.top('hello-nginx')
```

version 함수: Docker 버전 정보를 출력합니다.

```
c.version()
```

docker-version.py

```
import docker

c = docker.Client(base_url='unix://var/run/docker.sock')
print c.version()
```

1. Docker Remote API Python 라이브러리 사용하기

5. 기타 예제 및 함수: wait

wait 함수: 컨테이너가 정지될 때까지 대기합니다. 컨테이너가 정지되면 Exit Code를 출력합니다.

- container: 대기할 컨테이너의 객체 또는 이름입니다.

```
c.wait(container)
```

docker-wait.py

```
import docker

c = docker.Client(base_url='unix://var/run/docker.sock')
container_id = c.create_container(
  image='nginx:latest',
  ports=[80],
  volumes=['/data']
)
c.start(container_id)
print c.wait(container_id)
```

2. Docker Remote API Python 라이브러리로 HTTPS 통신하기

기본적으로 Docker 데몬은 ID와 패스워드를 이용한 로그인 기능이 없습니다. 그래서 Docker는 인증서를 이용한 HTTPS 통신을 제공합니다. 클라이언트 측에서는 인증서를 가지고 있어야 서버와 통신할 수 있습니다.

1. 인증서 생성하기

기본적으로 Docker 데몬은 ID와 패스워드를 이용한 로그인 기능이 없습니다. 그래서 Docker는 인증서를 이용한 HTTPS 통신을 제공합니다. 클라이언트 측에서는 인증서를 가지고 있어야 서버와 통신할 수 있습니다.

/etc/hosts

```
127.0.0.1       localhost
127.0.1.1       ubuntu
127.0.0.1       docker.example.com

# The following lines are desirable for IPv6 capable hosts
::1     localhost ip6-localhost ip6-loopback
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
```

127.0.0.1을 **docker.example.com**으로 설정합니다. 실제로 구입한 도메인을 사용해도 되고, 구입하지 않았더라도 사용할 수 있습니다. 이 책에서는 **docker.example.com**을 기준으로 설명하겠습니다.

2. Docker Remote API Python 라이브러리로 HTTPS 통신하기

1. 인증서 생성하기

인증서 파일을 생성할 차례입니다. 다음 명령들은 dockerbook/Chapter14/cert.sh 파일에 /Chapter14/cert.sh정리해놓았습니다. cert.sh 파일을 그대로 실행해도 됩니다. 단 비밀번호는 직접 입력해야 합니다.

CA(Certificate Authority) 인증서 파일을 생성합니다. 비밀번호 설정 부분이 나오면 사용할 비밀번호를 입력합니다. 그리고 **ca.pem** 파일을 생성할 때는 **ca-key.pem** 파일에 설정한 비밀번호를 입력합니다.

- Country Name: 국가 코드입니다. KO를 입력합니다.
- State or Province Name: 주 또는 도입니다. 자신의 상황에 맞게 입력합니다.
- Locality Name: 도시입니다. 자신의 상황에 맞게 입력합니다.
- Organization Name: 회사 이름을 입력합니다.
- Organizational Unit Name: 조직 이름을 입력합니다.
- Common Name: Docker 데몬을 실행하는 서버의 도메인입니다. 이 부분을 정확하게 입력하지 않으면 인증서를 생성해도 정상적으로 접속할 수 없습니다. /etc/hosts 파일에 설정한대로 docker.example.com를 입력합니다.
- Email Address: 이메일 주소입니다.

```
$ echo 01 > ca.srl
$ openssl genrsa -des3 -out ca-key.pem 2048
Generating RSA private key, 2048 bit long modulus
.....................................................................................+++
...............+++
e is 65537 (0x10001)
Enter pass phrase for ca-key.pem:<사용할 비밀번호 입력>
Verifying - Enter pass phrase for ca-key.pem:<사용할 비밀번호 입력>
$ openssl req -new -x509 -days 365 -key ca-key.pem -out ca.pem
Enter pass phrase for ca-key.pem:<ca-key.pem의 비밀번호 입력>
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:KO
State or Province Name (full name) [Some-State]:
Locality Name (eg, city) []:Seoul
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Example Company
Organizational Unit Name (eg, section) []:Example Company
Common Name (e.g. server FQDN or YOUR name) []:docker.example.com
Email Address []:exampleuser@example.com
```

2. Docker Remote API Python 라이브러리로 HTTPS 통신하기

1. 인증서 생성하기

서버 키, 인증서 서명 요청(Certificate signing request) 파일을 생성합니다. 비밀번호 설정 부분이 나오면 사용할 비밀번호를 입력합니다. 그리고 **server.csr** 파일을 생성할 때는 **server-key.pem** 파일에 설정한 비밀번호를 입력합니다.

- '/CN=docker.example.com' 부분은 반드시 앞에서 Common Name에 설정한 도메인을 입력합니다.

```
$ openssl genrsa -des3 -out server-key.pem 2048
Generating RSA private key, 2048 bit long modulus
.....................................................+++
..............+++
e is 65537 (0x10001)
Enter pass phrase for server-key.pem:<사용할 비밀번호 입력>
Verifying - Enter pass phrase for server-key.pem:<사용할 비밀번호 입력>
$ openssl req -subj '/CN=docker.example.com' -new -key server-key.pem \
-out server.csr
Enter pass phrase for server-key.pem:<server-key.pem의 비밀번호 입력>
```

서버 인증서 파일을 생성합니다. 비밀번호 입력 부분에는 ca-key.pem 파일에 설정한 비밀번호를 입력합니다.

```
$ openssl x509 -req -days 365 -in server.csr -CA ca.pem -CAkey ca-key.pem \
    -out server-cert.pem
Signature ok
subject=/CN=docker.example.com
Getting CA Private Key
Enter pass phrase for ca-key.pem:<ca-key.pem의 비밀번호 입력>
```

2. Docker Remote API Python 라이브러리로 HTTPS 통신하기

1. 인증서 생성하기

클라이언트 키, 인증서 서명 요청 파일을 생성합니다. 비밀번호 설정 부분이 나오면 사용할 비밀번호를 입력합니다. 그리고 client.csr 파일을 생성할 때는 key.pem 파일에 설정한 비밀번호를 입력합니다.

```
$ openssl genrsa -des3 -out key.pem 2048
Generating RSA private key, 2048 bit long modulus
.....................................................................+++
.....................................................+++
e is 65537 (0x10001)
Enter pass phrase for key.pem:<사용할 비밀번호 입력>
Verifying - Enter pass phrase for key.pem:<사용할 비밀번호 입력>
$ openssl req -subj '/CN=client' -new -key key.pem -out client.csr
Enter pass phrase for key.pem:<key.pem의 비밀번호 입력>
```

인증서 파일을 사용하여 접속을 허용하도록 설정 파일을 생성하고, 클라이언트 인증서 파일을 생성합니다. 비밀번호 입력 부분에는 ca-key.pem 파일에 설정한 비밀번호를 입력합니다.

```
$ echo extendedKeyUsage = clientAuth > extfile.cnf
$ openssl x509 -req -days 365 -in client.csr -CA ca.pem -CAkey ca-key.pem \
    -out cert.pem -extfile extfile.cnf
Signature ok
subject=/CN=client
Getting CA Private Key
Enter pass phrase for ca-key.pem:<ca-key.pem의 비밀번호 입력>
```

서버용 server-key.pem 파일, 클라이언트용 key.pem 파일의 비밀번호를 제거합니다. 비밀번호 입력 부분이 나오면 기존에 설정한 비밀번호를 입력합니다.

```
$ openssl rsa -in server-key.pem -out server-key.pem
Enter pass phrase for server-key.pem:<server-key.pem의 비밀번호 입력>
writing RSA key
$ openssl rsa -in key.pem -out key.pem
Enter pass phrase for key.pem:<key.pem의 비밀번호 입력>
writing RSA key
```

인증서 생성이 끝났습니다. 이제 인증서를 이용하여 Docker 데몬을 실행합니다.

2. Docker Remote API Python 라이브러리로 HTTPS 통신하기

1. 인증서 생성하기

```
$ sudo service docker stop
$ sudo docker -d --tlsverify --tlscacert=ca.pem --tlscert=server-cert.pem \
    --tlskey=server-key.pem -H tcp://0.0.0.0:2376
```

- `-d` : Docker를 데몬으로 실행합니다.
- `--tlsverify` : 인증서로 접속을 제어합니다.
- `--tlscacert` : 앞에서 생성한 ca.pem 파일을 설정합니다.
- `--tlscert` : 앞에서 생성한 server-cert.pem 파일을 설정합니다.
- `--tlskey` : 앞에서 생성한 server-key.pem 파일을 설정합니다.
- `-H` : 접속을 받을 IP 주소와 포트 번호입니다. tcp://0.0.0.0:2376를 설정합니다. 인증서를 이용한 데몬의 기본 포트 번호는 2376입니다.

명령을 실행하면 Docker 데몬이 foreground로 실행됩니다. 따라서 테스트를 위해 새 터미널을 실행합니다.

2. Docker Remote API Python 라이브러리로 HTTPS 통신하기

1. 인증서 생성하기

현재 리눅스 계정의 홈 디렉터리(/home/<사용자 계정>)에 .docker 디렉터리를 생성하고, 그 아래에 ca.pem, cert.pem, key.pem 파일을 복사합니다.

```
$ mkdir ~/.docker
$ cp ca.pem ~/.docker/ca.pem
$ cp cert.pem ~/.docker/cert.pem
$ cp key.pem ~/.docker/key.pem
```

다른 컴퓨터에서 Docker 데몬에 접속하려면 .docker 디렉터리만 복사해서 사용하면 됩니다.

이제 Docker 데몬에 접속하여 명령을 실행해보겠습니다.

```
$ sudo docker --tlsverify -H docker.example.com:2376 info
Containers: 1
Images: 3
Storage Driver: aufs
 Root Dir: /var/lib/docker/aufs
 Dirs: 5
Execution Driver: native-0.2
Kernel Version: 3.13.0-24-generic
Operating System: Ubuntu 14.04 LTS
WARNING: No swap limit support
```

`docker --tlsverify -H <Docker 데몬 도메인>:2376` 형식입니다. `-H` 옵션에는 앞에서 인증서를 생성할 때 입력했던 도메인을 설정합니다. 이 도메인이 맞지 않으면 Docker 명령을 실행할 수 없습니다. 또한, .docker 디렉터리에 인증서 파일이 없거나 맞지 않으면 Docker 명령을 실행할 수 없습니다.

2. Docker Remote API Python 라이브러리로 HTTPS 통신하기

2. Python 라이브러리 사용하기

이제 Python 라이브러리로 인증서를 사용한 Docker 데몬(TLS)에 접속해보겠습니다.
다음 내용을 docker-run-tls.py로 저장합니다. 인증서를 사용한 Docker 데몬에 접속하여 nginx 이미지를 받고 컨테이너로 실행하는 예제입니다.

docker-run-tls.py

```python
import docker
from os.path import expanduser
home = expanduser('~')

tls_config = docker.tls.TLSConfig(
  client_cert=(home + '/.docker/cert.pem', home + '/.docker/key.pem'),
  ca_cert=home + '/.docker/ca.pem',
  verify=True
)
c = docker.Client(base_url='https://docker.example.com:2376', tls=tls_config)
c.pull(repository='nginx', tag='latest')
container_id = c.create_container(
  image='nginx:latest',
  ports=[80],
  volumes=['/data'],
  name='hello'
)
c.start(
  container_id,
  port_bindings={80: ('0.0.0.0', 80)},
  binds={'/data': {'bind': '/data', 'ro': False}}
)
```

- docker.tls.TLSConfig 클래스를 생성합니다. client_cert에는 cert.pem, key.pem 파일의 경로를 설정하고, ca_cert에는 ca.pem 파일의 경로를 설정합니다. 반드시 절대 경로로 설정해야 하고, expanduser 함수를 이용하여 현재 리눅스 계정의 홈 디렉터리를 구한 뒤 .docker 디렉터리를 설정합니다. 그리고 인증서를 이용하여 접속하도록 verify=True로 설정합니다.
- docker.Client 클래스를 생성합니다. base_url에는 https://docker.example.com:2376과 같이 https 프로토콜에 인증서를 생성할 때 입력했던 도메인을 설정합니다. tls에는 앞에서 생성한 tls_config를 설정합니다.
- docker.Client 클래스로 생성한 인스턴스 c로 pull 함수를 실행합니다. 이미지 이름과 태그를 설정합니다.
- c.create_container 함수로 컨테이너를 생성합니다. 이미지 이름, 호스트에 연결할 포트, 호스트와 연결할 디렉터리, 컨테이너 이름을 설정합니다.
- c.start 함수로 컨테이너를 시작합니다. c.create_container 함수를 실행하고 받은 컨테이너 객체를 사용하고, 컨테이너 포트 번호와 외부에 노출할 포트 번호, 호스트 디렉터리와 연결할 컨테이너 디렉터리를 설정합니다.

2. Docker Remote API Python 라이브러리로 HTTPS 통신하기

2. Python 라이브러리 사용하기

이제 Python 라이브러리로 인증서를 사용한 Docker 데몬(TLS)에 접속해보겠습니다.
다음 내용을 docker-run-tls.py로 저장합니다. 인증서를 사용한 Docker 데몬에 접속하여 nginx 이미지를 받고 컨테이너로 실행하는 예제입니다.

이제 docker-run-tls.py 파일을 실행합니다.

```
$ sudo python docker-run-tls.py
```

컨테이너 목록을 출력하면 Python 라이브러리로 생성한 컨테이너가 표시됩니다.

```
$ sudo docker --tlsverify -H docker.example.com:2376 ps
CONTAINER ID  IMAGE     COMMAND  CREATED     STATUS       PORTS               NAMES
18e2d04f8753  nginx:1   "nginx"  1 sec...    Up 1 sec...  0.0.0.0:80->80/tcp  hello
```

`-H` 옵션에는 docker.example.com:2376과 같이 반드시 인증서를 생성할 때 입력한 도메인을 설정합니다.