공학
트렌드

소프트웨어공학센터 경영지원TF팀

Vagrant와 Ansible을 이용한 개발 환경 구축
Part 1 : Vagrant의 이해

2014. 8. 12. [제102호]

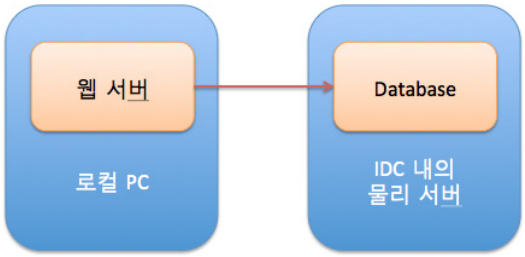
Contents

정보통신산업진흥원 부설
nipa 소프트웨어공학센터

Ⅰ. 가상 머신 이용 배경

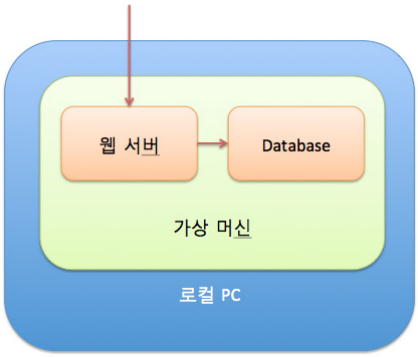
일반적인 서버 개발환경은 〈그림 1〉처럼 DB는 IDC 내의 서버를 활용하고, 웹 서버는 로컬 환경을 이용하는 단순한 웹 서버 개발 환경을 사용하고 있었다. 따라서 개발자가 테스트하는 동안 테스터는 개발 버전을 테스트할 수 없고 개발 서버로 배포될 때까지 기다려야 했다.

그림 1_일반적인 웹 서버의 개발 방식



이 방법은 두 가지 이슈가 있다. 첫 번째, DB를 초기화 해 다시 셋팅하고 DAO 유닛 테스트를 통해서 CI (Continuous Integration)를 진행하는 과정을 할 수 없다. 그 이유는 DB는 모든 이들이 함께 쓰는 장비이기 때문이다. 그래서 이 부분은 진행을 할 수 없다. DB 데이터를 모든 개발자가 공유할 수는 있지만, 모든 테스트를 다 진행할 수 없는 형태이다.

그림 2_가상머신 내에 웹 서버와 Database를 설치한 환경



두 번째, 테스터는 개발되고 있는 버전이 완료되었다는 테스트 시작을 알려주기 전까지는 기다려야 한다. 애자일 개발 방법으로 진행할 때, 2주 단위로 구현하고 테스트하는 일정이 존재하면 개발자가 개발 서버에 따로 배포해 줄 때까지 기다리는 것 말고는 마땅

히 할 수 있는 방법이 없다. 이런 불편함을 제거하기 위해서 〈그림 2〉와 같이 하나의 가상 머신으로 통일화 할 수 있다. 웹 서버와 Database를 함께 설치하면 하나의 개발 서버를 구축할 수 있고, 이 구축된 가상 머신은 개발자와 테스터 간에 공유될 수 있을 것이다. 또한 설정을 통해서 웹 서버에 접속할 포트 역시 하나의 포트로 통일 시킬 수 있고, 필요하다면 로컬 PC에서 충돌하지 않을 포트를 지정하여 서비스 할 수 있다. 이런 방법은 다음의 장점들을 제공한다.

첫째, CI 철학에 맞게 가상머신에서 DB 초기화를 진행할 수 있다. 따라서 DAO 테스트를 완벽하게 진행할 수 있다. (물론, staging 환경에 맞게 production 레벨에서는 테스트 코드가 동작되지 못하게 진행해야 한다.)

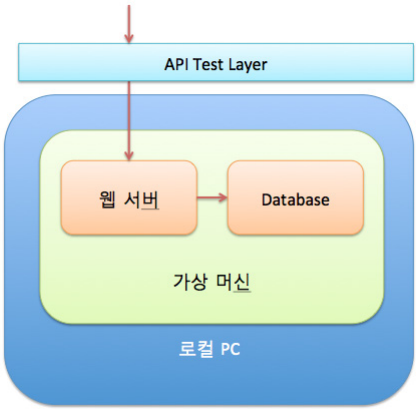
둘째, 테스터는 가상머신만 받을 수 있으면 언제든지 테스트를 진행할 수 있다. 테스터가 개발자로부터 어느 버전을 테스트 할지 미리 알고 있는 상태라면, 해당 소스의 버전을 다운로드 해 테스트를 병행으로 진행하면서 스크럼 팀의 빠른 스피드의 개발을 진행시킬 수 있도록 도와준다.

셋째, 개발자 – 테스터뿐 아니라, 연동이 필요한 다른 개발자에게 도움을 줄 수 있다. 즉, 일종의 개발 환경 킷을 제공하는 효과를 가질 수 있다. 〈그림 3〉또는 〈그림 4〉와 같이 가상머신 위에 API 테스트 Layer(API 테스트베드)를 두어 웹 서버의 환경을 두어 Acceptance Test 환경을 만들 수 있다. 이런 사례는 Facebook 이나 Twitter의 개발자 센터와 같은 비슷한 모델로 구현이 가능하다.

넷째, 사무실 IP에서 IDC로의 연결이 방화벽으로 막혀 있는 환경에서 유용하다. 전자금융 거래법(일명 전금법)이 적용되거나, 개인정보를 가지고 테스트해야 하는 상황에서는 〈그림 1〉과 같은 방식은 더 이상 개발이 불가능해진다. 즉, 로컬 개발 환경에서 Production이 아닌 개발 단계라 할지라도 IDC의 서버나 DB에 연결하는 방식이 어려워진다. 보안이 더욱 대두되는 상황이기 때문이다.

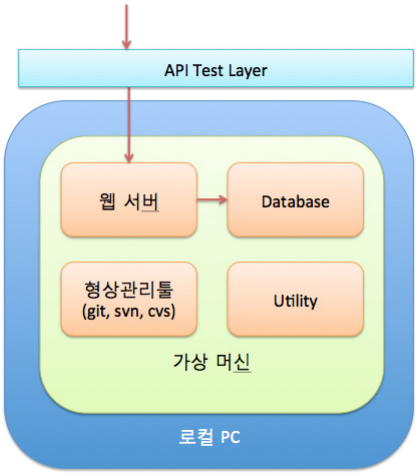
다섯째, 로컬 환경에서 소프트웨어 테스트를 위해 새로 설치하고 삭제하는 것이 번거로울 수 있다. 예를 들어 새로 출시된 JDK8 위에서 동작하는 WAS를 테스트하기 위해서 JDK8을 설치했다가 테스트하고 다시 삭제해야 하는 경우가 있는데, 가상머신을 이용해서 이런 번거로움을 해결할 수 있다.

그림 3_가상머신 위에 API Test Layer를 위치한 개발 환경



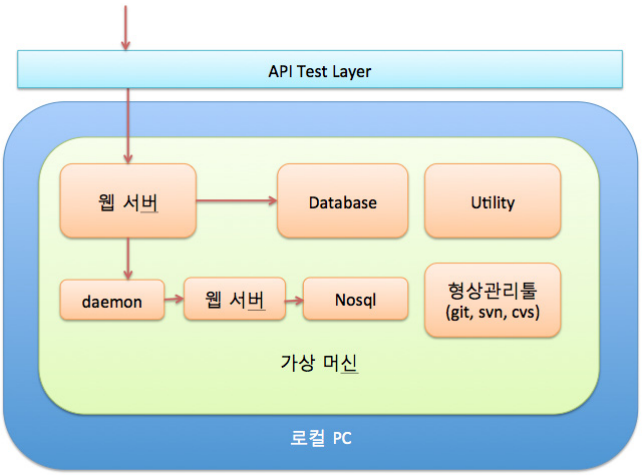
Practical한 개발자 환경은 〈그림 3〉에서 조금 더 진화한 〈그림 4〉의 모습이다. 개발자 환경 기반 위에 다양한 유틸리티를 포함한 그림이 될 것이다. 일반 유틸리티부터 개발자나 테스터를 위한 소스 배포 툴, 재시작 스크립트, DB 쪽 스크립트를 포함한 그림이 될 것이다.

그림 4_그림 10에서 형상관리툴과 필요한 Utility가 포함된 개발 환경



가상머신이 존재하는 것은 간단한 작업을 하는 웹 서버를 테스트하기 위함보다는 여러 개의 웹 서버와 DB 또는 Nosql이 존재하는 다양한 스토리지가 존재하는 테스트 환경을 테스트하는 경우가 더 많다. 이를 위해서는 가상 머신 내에서는 다양한 시스템을 설치하고 테스트 할 수 있는 환경을 〈그림 5〉와 같이 구성할 수 있을 것이다.

그림 5_연동 가능한 개발 환경과 API Test Layer가 포함된 가상 머신



이렇게 만들 가상 머신은 Vagrant로 테스트 환경 구현이 가능하다. 그리고 배포는 shell script, puppet, chef, ansible과 같은 자동화 툴로 가상머신을 만들 수 있는 배포 기능을 추가할 수 있다. 유지보수가 가능한 가상머신을 계속 만들 수 있다.

II. Vagrant의 소개와 설치

2.1 Vagrant 소개

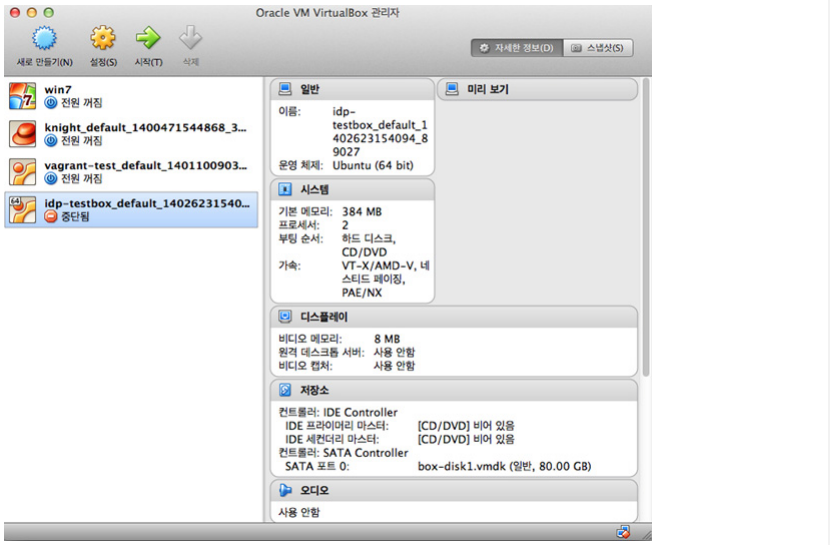
Vagrant는 가상 머신 관리 도구이다. 즉, 가상 머신 플랫폼 (Hypervisor) 이 먼저 설치되어야 하고 그 가상 머신 플랫폼을 관리하는 CLI로 관리하는 도구가 Vagrant이다.

Vagrant를 쓰는 이유는 자동화가 편하다는 점 때문이다. 즉, virtual box를 실행시키기 위한 설정과 설치, 시작/중지/종료 등 이런 작업을 일일이 해야 하는 작업을 Vagrant를 이용하면 CLI(command line interface)기반에서 할 수 있다.

즉, 〈그림 6〉의 예처럼 Mac OS 에서 Windows 7이나 Ubuntu 리눅스를 실행시키려면 Oracle VM Virtual Box 에서 이미지를 다운로드 해서 이미지의 설정을 정하고 시작/중지/종료 버튼을 눌러야 한다. 이 과정은 사람이 수동으로 작업해야 한다.

그러나 Vagrant는 이를 CLI상에서 쉽게 해결 할 수 있다. 예를 들어 시작은 vagrant up, 종료는 vagrant halt 이다. 자세한 설명은 뒤에서 하도록 한다.

그림 6_Oracle VM Virtual Box 화면



Vagrant는 가상 서버를 구현하는 좋은 툴로서, 지금까지 얘기한 모든 것이 가능하도록 지원한다. 우선 간단하게 소개하자면 다음과 같다.

- 라이선스 : MIT License
- 개발 언어 : Ruby
- 메인 개발자 : Mitchell Hashimoto http://mitchellh.com/
- 홈페이지 : http://www.vagrantup.com/
- 사용 할 수 있는 이미지 목록 : http://www.vagrantbox.es/
- 버전
- 1.6.3 (2014.7.7 현재)
- 1.0.0 (2012.3.7)
- 문서 : http://docs.vagrantup.com/v2/

2.2 Vagrant 설치

Virtualbox download url(https://www.virtualbox.org/wiki/Downloads)에 접속해서 가장 먼저 가상 머신 플랫폼(Hypervisor)인 Oracle의 Virtual Box 를 설치한다. Oracle Virtual Box는 GPL License 이기 때문에 사용하는 데는 별다른 이슈가 없다. 저자는 Mac OS X 사용자이므로 OS X hosts 용을 다운로드 했다[1]. 다음은 Vagrant 설치를 진행한다. http://www.vagrantup.com/에 접속해서 하단 좌측 Download 버튼을 클릭한다. Vagrant download 화면(http:

1) 본 필자는 MAC OS X 를 사용하므로 MAC OS X 기준으로 설명하도록 하겠다.

//www.vagrantup.com/downloads.html)에서 Vagrant를 운영체제에 맞게 설치한다.

설치한 이후에는 Vagrant 의 버전을 확인한다.

```
$ vagrant -v
Vagrant 1.6.3
```

2.3 Vagrant 실행

2.3.1 가상 머신 이미지 다운로드 및 vagrant init (초기화)

먼저 프로젝트 디렉토리를 생성하고 생성한 디렉토리로 접근한다.

```
$ mkdir idp-testbox
$ cd idp-testbox
```

Vagrant 이미지를 다운로드 해서 local 에 저장한다. 아래 예는 precise(ubuntu 12.04) 64비트 버전을 다운로드 한 예시이다. precise64는 Vagrant 안에서 사용되는 이름 또는 별칭이다. OS가 설치된 가상 머신 이미지를 box로 생각 하면 된다.

```
$ vagrant box add precise64 http://files.vagrantup.com/precise64.box
```

참고로 저장된 vagrant 가상 머신 이미지는 Mac의 경우 ~/.vagrant.d/boxes 디렉토리에 저장된다. ~/.vagrant 디렉토리에는 가상 머신에 대한 메타 정보가 포함되어 있다.

vagrant init 명령어를 사용하여 box를 초기화한다. 그래서 vagrant 메타 파일인 Vagrantfile을 생성하도록 한다.

```
$ vagrant init
$ ls
Vagrantfile
```

Vagrantfile 파일을 ansible로 provision할 수 있도록 수정한다. 관련 없는 내용과 주석을 제외해서 Vagrantfile 파일을 아래와 같이 수정한다.

```
# Vagrantfile API/syntax version. Don't touch unless you know what you're doing!
VAGRANTFILE_API_VERSION = "2"

Vagrant.configure(VAGRANTFILE_API_VERSION) do |config|
    config.vm.box = "precise64"
    config.vm.box_url = "http://files.vagrantup.com/precise64.box"

    config.vm.hostname = "web"
    config.vm.network "private_network", ip: "192.168.1.50"
    config.vm.network :forwarded_port, guest: 9966, host: 8888

    config.vm.provision "ansible" do |ansible|
        ansible.playbook = "playbook.yml"
    end
end
```

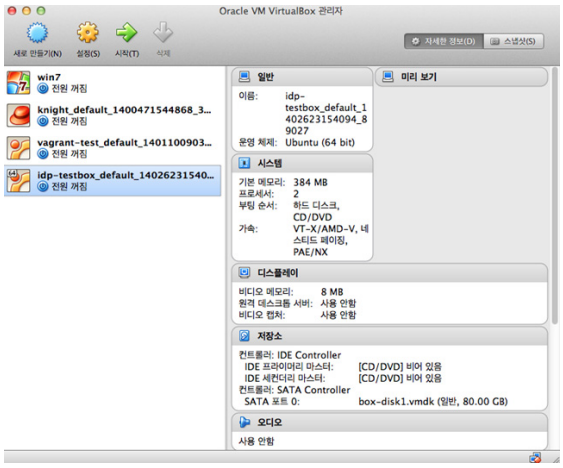
config.vm.box에 precise64로 선택하고 config.vm.box_url을 이미지를 다운로드 할 수 있는 url로 지정했다. 가상 머신의 host 이름을 web이라 지칭했다. config.vm.network 프로퍼티를 이용해 private_network ip를 사용하도록 했다. 그리고 가상서버에서 9966포트로 띄운 서버를 Host 서버에서 8888포트로 포트 포워딩 방식을 사용한다. 예를 들어 가상서버에서는 http://192.168.1.50:9966/view 라는 주소를 가진 웹 주소를 호스트서버에서는 http://127.0.0.1:8888/view 라는 주소로 접근할 수 있음을 의미한다.

config.vm.provision 은 가상서버로 어떤 툴로 provision(배포)할 것인지를 지정하는 것이다. config.vm.provision "ansible"의 의미는 Ansible을 이용해서 하겠다는 의미를 가진다. ansible.playbook = "playbook.yml"은 ansible provisioning 메타 파일로 playbook.yml 파일을 사용하겠다는 의미를 가진다. ansible에서 사용하는 스크립트를 playbook 이라고 하며 yml 파일은 yaml 이라는 표준 스펙을 지킨다. 자세한 얘기는 다음 장에서 설명할 예정이다. 예제로 playbook.yml을 다음과 같이 저장한다. 간단하게 설명하면 vagrant 계정으로 'ls –al /home' 결과를 화면에 출력하라는 샘플 정보이다.

```
- hosts: all
  user: vagrant
  tasks:
  - name : test
     action:  command ls -al /home
     register: vagrant
  - debug: var=vagrant.stdout_lines
```

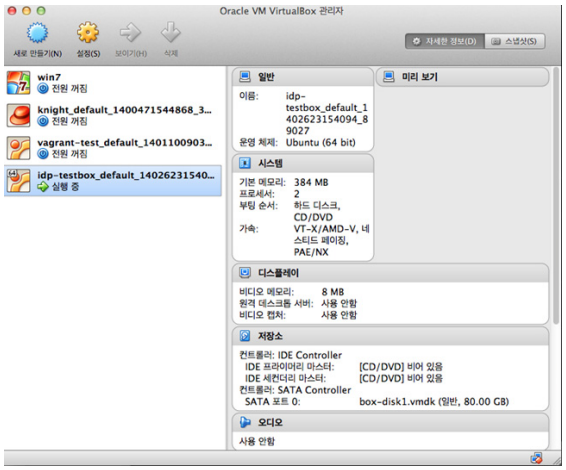
2.3.2 vagrant up

그림 7_평상시 Virtual Box 상태



〈그림 7〉과 같이 평소 가상 머신의 상태는 "전원 꺼짐"으로 되어 있다. 사용자가 vagrant up 명령을 이용하면 〈그림 8〉과 같이 가상 머신의 상태는 "실행 중"으로 바뀐다.

그림 8_vagrant up 명령 이후 가상 머신의 status가 “실행 중"으로 변경된 화면



vagrant up 명령을 내린다.

```
$ vagrant up
```

상황에 따라서는 provision 하지 않은 가상 머신 시작을 하고 싶을 때는 —no-provision 을 명령어 뒤에 붙여 사용한다. provision 되지 않은 상태임을 확인할 수 있다.

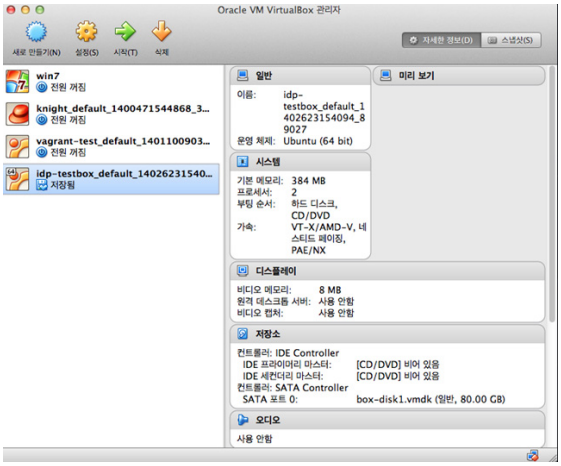
```
$ vagrant up --no-provision
```

2.3.3 vagrant suspend

가상머신을 잠깐 중지하려면 vagrant suspend 명령을 실행한다. 〈그림 9〉 와 같이 가상머신의 상태가 “저장됨"으로 바뀐다.

```
$ vagrant suspend
```

그림 9_vagrant suspend 명령 이후 가상머신의 status가 “저장됨"으로 변경



2.3.4 vagrant resume

중지된 가상머신을 다시 재개하려면 vagrant resume을 실행하면 된다. 〈그림 8〉과 같은 "실행 중" 인 화면으로 변경된다.

```
$ vagrant resume
```

2.3.5 vagrant provision

위에서 언급했던 playbook.yml 을 기반으로 provisioning을 한다. vagrant provioning 명령을 하면 다음과 같이 가상 머신 서버로 들어가 ls –al /home 결과를 화면에 출력한다.

```
$ vagrant provision
==> default: Running provisioner: ansible...

PLAY [all] ********************************************************************

GATHERING FACTS ***************************************************************
ok: [default]

TASK: [test] ******************************************************************
changed: [default]

TASK: [debug var=vagrant.stdout_lines] ****************************************
ok: [default] => {
    "vagrant.stdout_lines": [
        "total 12",
        "drwxr-xr-x  3 root    root    4096 Sep 14  2012 .",
        "drwxr-xr-x 25 root    root    4096 Jun 27 06:00 ..",
        "drwxr-xr-x  7 vagrant vagrant 4096 Jun 30 05:50 vagrant"
    ]
}

PLAY RECAP ********************************************************************
default                    : ok=3    changed=1    unreachable=0    failed=0
```

만약 provision을 ansible이 아닌 shell 기반으로 수정한다. Vagrantfile을 아래 예제의 볼드체로 표시한 부분(가상 머신에서 test.sh 을 실행) 하는 부분으로 수정한다.

```
$ cat Vagrantfile
# Vagrantfile API/syntax version. Don't touch unless you know what you're doing!
VAGRANTFILE_API_VERSION = "2"

Vagrant.configure(VAGRANTFILE_API_VERSION) do |config|
    config.vm.box = "precise64"
    config.vm.box_url = "http://files.vagrantup.com/precise64.box"

    #config.vm.network :public_network
    config.vm.hostname = "web"
    config.vm.network "private_network", ip: "192.168.1.50"
    config.vm.network :forwarded_port, guest: 9966, host: 8888

    config.vm.provision :shell, :path => "test.sh"

end
```

test.sh 파일은 간단히 /home 디렉토리 출력하는 것이다.

```
$ cat test.sh
#!/bin/sh

ls -al /home
```

그 결과는 ansible로 이용했던 것과 결과가 같다. shell 결과 ansible 결과를 비교해볼 때 고급스러운 개념이 있다는 것을 확인할 수 있다.

```
$ vagrant provision
==> default: Running provisioner: shell...
    default: Running:
/var/folders/nx/lkzmd37d6fj3sg9kg5flt3yr0000gp/T/vagrant-shell20140710-58428-
mkmu26.sh
==> default: stdin: is not a tty
==> default: total 12
==> default: drwxr-xr-x  3 root    root    4096 Sep 14  2012 .
==> default: drwxr-xr-x 25 root    root    4096 Jun 27 06:00 ..
==> default: drwxr-xr-x  7 vagrant vagrant 4096 Jun 30 05:50 vagrant
```

2.3.6 vagrant status

가상 머신 status 정보를 보려면 vagrant status 명령을 사용한다. 아래와 같이 영어로 실행 중을 의미하는 running 이 출력되는 것을 볼 수 있다.

```
$ vagrant status
Current machine states:
default                   running (virtualbox)
```

2.3.7 vagrant ssh

vagrant ssh 를 이용하여 가상 머신 서버로 접근이 가능한지 테스트한다. ssh 로 가상 서버로 접근한다. 위에서 config.vm.hostname으로 지정된 호스트명 web을 확인할 수 있으며, 기본 계정은 vagrant이다.

```
$ vagrant ssh
vagrant@web:~$
```

ssh 포트가 열려 있으니 ssh 연결하는 방법은 당연히 존재한다. RSA 알고리즘의 ssh key인 id_rsa(개인 키), id_rsa.pub 키를 $HOME/.ssh에 생성하도록 한다.

```
$ ssh-keygen -t rsa
```

host의 ssh 키를 가상서버에 복사한다. vagrant 설정에서 지정한 config.vm.network "private_network" ip로 지정한 ip를 입력한다.

```
$ ssh-copy-id vagrant@192.168.1.50
```

이후 ssh 연결 시 정상적으로 접근한 것으로 나온다.

```
$ ssh vagrant@192.168.1.50
vagrant@web:~$
```

2.3.8 vagrant halt

vagrant halt 명령을 내려 가상머신을 종료한다. 〈그림 7〉의 상태로 복귀하게 된다.

```
$ vagrant halt
```

2.3.9 vagrant package

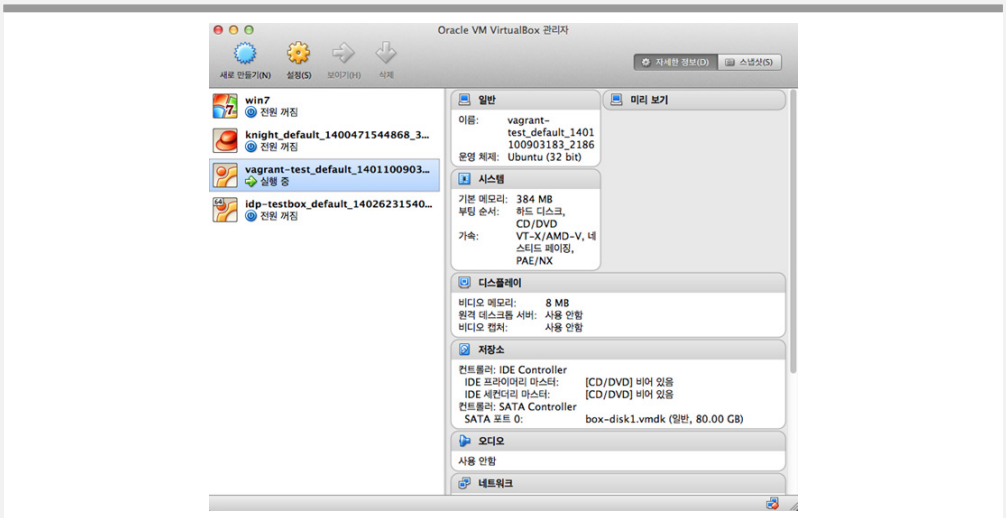
vagrant package를 통해서 패키징하여 다른 개발자에게 전달할 수 있다.

```
$ vagrant package
$ ls package.box
package.box
```

가상 머신을 virtual box 가상머신 리스트에서 삭제를 할 수 있다.

〈그림 10〉과 같이 vagrant up을 해서 "vagrant-test_default_1401100903183_2186"이란 이름을 가진 가상머신을 실행했다.

그림 10_vagrant up 으로 가상머신이 "실행 중" 인 화면



2.3.10 vagrant destroy

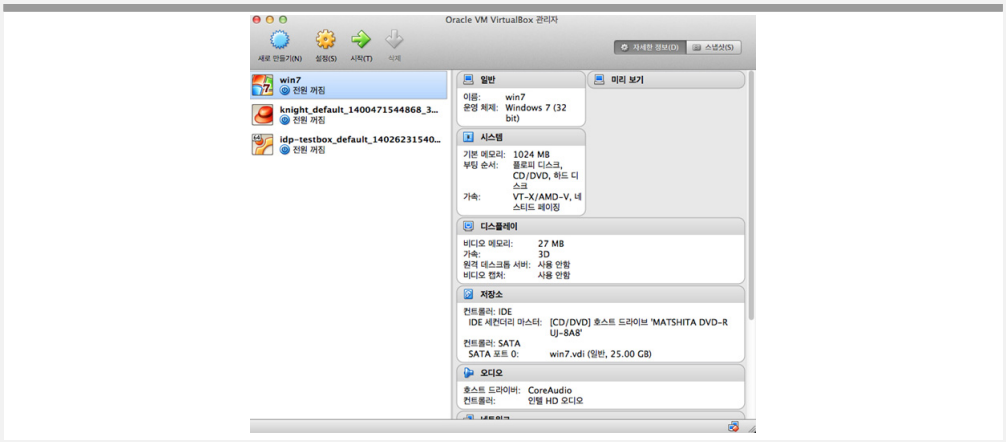
이 때 vagrant destroy 명령을 내리면 다시 한 번 destroy를 확인한다. y를 입력하고 엔터를 입력한다.

```
$ vagrant destroy
```

〈그림 11〉과 같이 virtual box 가상머신 리스트 화면을 보면 "vagrant-test_default_14011

00903183_2186"이름을 가진 가상머신은 삭제되었다.

그림 11_실행 중이었던 가상머신이 삭제 된 화면



2.3.11 vagrant reload

Vagrantfile 수정 이후 반영을 하려면 vagrant reload 명령을 실행하면 설정 파일을 읽고 반영한다. 가상머신도 종료했다가 다시 실행한다.

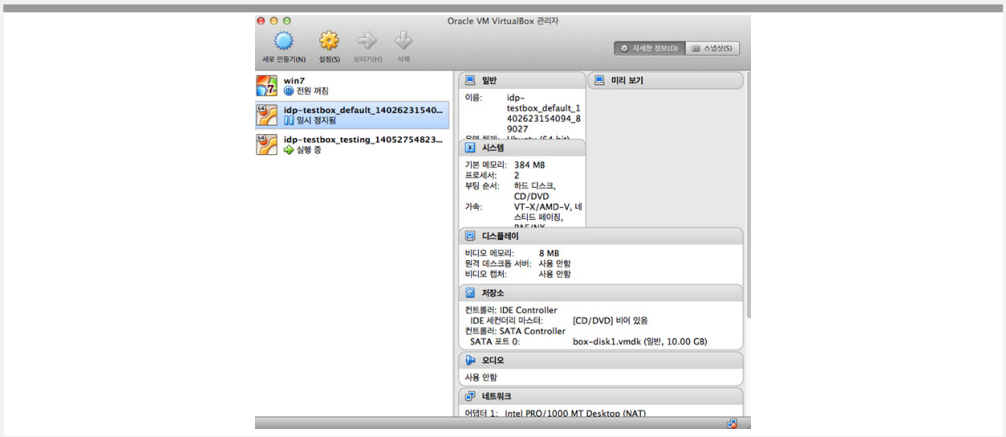
```
$ vagrant reload
```

2.3.12 가상 서버 이름 지정하기

Vagrantfile에 지금까지 name을 주지 않았다. 아래와 같이 config.vm.box와 config.vm.box_url 만 지정하면 〈그림 12〉와 같이 '디렉토리명_testing_1405275482326_94624'의 명을 가지게 된다.

```
Vagrant.configure(VAGRANTFILE_API_VERSION) do |config|
    config.vm.box = "precise64"
    config.vm.box_url = "http://files.vagrantup.com/precise64.box"
...
```

그림 12_이름을 따로 주지 않았을 때 나오는 가상서버 디폴트 이름

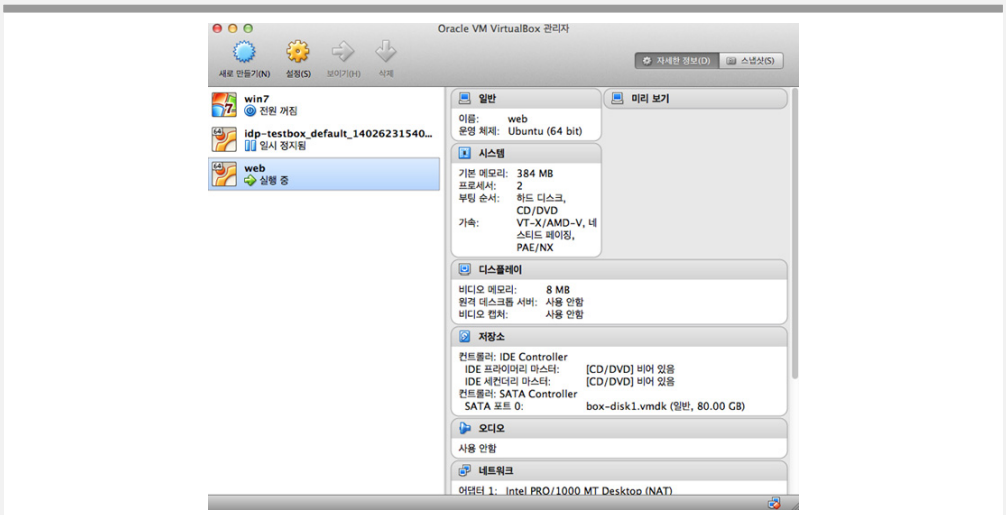


virtual box의 화면에서 이름을 직접 수정하지 않고 Vagrantfile의 내용을 수정하여 이름을 변경할 수 있다. 아래 예제는 〈그림 13〉의 디폴트 가상 서버 이름을 'web'으로 바꾸는 예제이다.

```
Vagrant.configure(VAGRANTFILE_API_VERSION) do |config|
    config.vm.box = "precise64"
    config.vm.box_url = "http://files.vagrantup.com/precise64.box"

    config.vm.provider :virtualbox do |vb|
      vb.name = "web"
    end
..
end
```

그림 13_예제 실행 후 가상 서버의 디폴트 이름이 web으로 변경됨



2.3.13 여러 가상 서버(Multiple VM) 을 통제하기

다른 디렉토리를 하나 생성한 후, Vagrantfile 파일을 아래와 같이 수정한다. 가상 머신 2대를 동시에 실행/종료 또는 하나만 실행이 가능하다.

```
$ cat Vagrantfile
# Vagrantfile API/syntax version. Don't touch unless you know what you're doing!
VAGRANTFILE_API_VERSION = "2"

Vagrant.configure(VAGRANTFILE_API_VERSION) do |config|
    config.vm.box = "precise64"
    config.vm.box_url = "http://files.vagrantup.com/precise64.box"
    config.vm.provision :shell, :path => "test.sh"

    config.vm.define :web do |web_config|
        web_config.vm.box = "web"
        web_config.vm.define "foohost" do |foohost|
        end
        web_config.vm.hostname = "web"
        web_config.vm.network "private_network", ip: "192.168.1.50"
        web_config.vm.provider :virtualbox do |vb|
            vb.name = "web"
        end
```

```
    end

    config.vm.define :mysql do |mysql_config|
        mysql_config.vm.box = "mysql"
        mysql_config.vm.define "foohost" do |foohost|
        end
        mysql_config.vm.hostname = "mysql"
        mysql_config.vm.network "private_network", ip: "192.168.1.51"
        mysql_config.vm.provider :virtualbox do |vb|
            vb.name = "mysql"
        end
    end

end
```

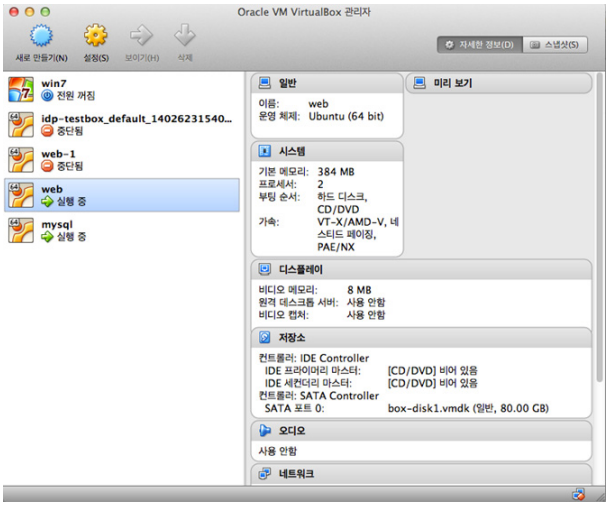
config.vm.box_url 으로 정의한 "http://files.vagrantup.com/precise64.box" vagrant 이미지를 다운로드 해 web과 mysql이라는 이름으로 각각 virtual box 이미지의 이름을 변경한다. 가상 머신 이름(config.vm.define)과 프로바이더(config.vm.provider)의 이름을 정의할 수 있다. 프로바이더(config.vm.provider)를 잘 활용하면 cpu나 memory 설정을 구체적으로 지정할 수 있다.

```
config.vm.provider "virtualbox" do |v|
  v.memory = 1024
  v.cpus = 2
end
```

vagrant up 명령을 실행하면 web과 mysql이라는 이름의 가상서버를 아래와 같이 실행한다. 〈그림 14〉와 같이 두 개의 가상머신이 실행 중인 것을 확인할 수 있다.

```
$ vagrant up
```

그림 14_가상 머신 2개를 동시에 실행시킨 이후 화면



만약 각 가상머신 별로 실행을 따로 하려면 vagrant up web 또는 vagrant up mysql 을 사용한다. 참고로 json 파일로 여러 개의 가상 머신을 생성할 수 있다. 아래 블로그 글

을 참고하면 힌트를 얻을 수 있을 것이다. http://programmaticponderings.wordpress.com/2014/02/27/multi-vm-creation-using-vagrant-and-json/

지금까지 가상 머신을 활용한 배경을 이야기 했고, vagrant에 대한 설치, 소개를 모두 마쳤다. Part 2에서는 Ansible을 활용한 방식을 상세히 살펴보도록 하겠다.

참고 자료

1. https://docs.vagrantup.com
2. https://www.virtualbox.org
3. http://stackoverflow.com/questions/17845637/vagrant-default-name