Ansible과 Vagrant를 이용한 개발 환경 구축 #1
- 김용환 (http://knight76.tistory.com)

(이 글은 양이 많은 관계로 다음 주소의 글에 게재된 내용으로 축소되었다. 오타가 많고 정리가 안되었으나, 배경지식을 많이 할애한 관계로 공개한다.

http://knight76.tistory.com/entry/%EC%86%8C%ED%94%84%ED%8A%B8%EC%9B%A8%EC%96%B4-%EA%B3%B5%ED%95%99%EC%84%BC%ED%84%B0-%EA%B8%B0%EA%B3%A0-Vagrant%EC%99%80-Ansible%EC%9D%84-%EC%9D%B4%EC%9A%A9%ED%95%9C-%EA%B0%9C%EB%B0%9C-%ED%99%98%EA%B2%BD
)

Part 1 순서

1. 가상 머신 이용 배경

1.1. **가상화**

가상화는 컴퓨터에서 컴퓨터 리소스의 추상화를 일컫는 광범위한 용어이다. "물리적인 컴퓨터 리소스의 특징을 다른 시스템, 응용 프로그램, 최종 사용자들이 리소스와 상호 작용하는 방식으로부터 감추는 기술"로 정의할 수 있다. 기업은 서버 가상화를 통해 하나의 컴퓨터에서 동시에 1개 이상의 운영체제를 가동 시킬 수 있다. 대부분의 서버는 단지 용량의 10~15%만 사용하는데, 가상화는 이런 서버의 효용률(utilization rate)을 70% 그 이상으로 올릴 수 있다. 높은 수준의 효용률은 같은 분량의 업무 처리에서 요구하는 컴퓨터 수를 줄여준다. (출처 : http://ko.wikipedia.org/)

저자가 집중하는 가상화는 어플리케이션에 대해서 개발자나 테스터 중심으로 개발/테스트/재현 가능한 시나리오를 찾아줄 수 있는 환경이다. 즉 테스트 환경을 개발자/테스터와 함께 공유하고 개발 또는 테스팅을 로컬 환경에서 빨리 실행할 수 있도록 하는 것이 필요하다. 개발자 및 테스터는 테스팅 환경에 집중하기 때문에 성능 좋은 하드웨어를 갖춘 물리 장비를 여러 개의 가상화 장비로 분리하는 서버 가상화에 관한 얘기는 본 글에서는 하지 않도록 하겠다.

개발자의 입장에서는 개발자끼리 서로 환경을 공유할 수 있도록 하여 테스트를 진행할 수 있도록 하고, 다양한 컴포넌트들이 엮여 있는 환경에서도 하나의 가상머신으로 모을 수 있으니 개발 환경을 빠른 시간 내에 테스트할 수 있다. 게다가 CI (Continuous Integration)를 로컬 PC에서 진행할 수 있다. 가상화 환경에서 어플리케이션 또는 플랫폼과 DB를 설치 및 배포 한 후 CI 서버를 연동할 수 있는 환경이 될 수 있다. 특히, 원래 CI 에서는 DB를 삭제하고 새롭게 설치해서 완벽한 Integration 테스트를 진행하는 것을 권고한다. 그러나 실제 개발 환경에서는 DB를 로컬이 아닌 IDC 내의 장비로 활용했기 때문에 실질적인 CI를 진행할 수 없었다. 그러나 가상환경을 구축하게 되면 진정한 CI로 활용할 수 있다.

테스터의 입장에서는 미리 서버에 배포되어 있는 버전을 테스트하지 않고, 개발 중인 테스트 버전을 로컬환경에서 사전에 테스트함으로서 보다 높은 테스팅 품질 지표를 달성할 수 있도록 노력해볼 수 있다. 즉, 블랙 박스 테스트를 로컬환경에서 쉽게 진행할 수 있다. Jmeter나 SoapUI, WebDriver 를 이용해 테스트를 진행하고, 빠른 피드백을 개발팀에 전달할 수 있도록 해준다 .

1.2 기존 임베디드 / 스마트폰 개발 환경

임베디드 소프트웨어 개발자는 임베디드 개발 회사에 고용되어 해당 제품을 출시할 수 있도록 노력한다. 제품을 출시한 후에는 고객으로부터 오는 다양한 문제들과 VOC 이슈들을 해결해야 한다. 만약 테스트 환경이 구축되지 않는다면 발생되는 이슈, 문제들의 해결을 충분히할 수 없을 것이다. 테스트 환경이 구축된다 하더라도 로컬 PC 가 아닌 임베디드 기기 안에서 진행하면 많은 시간이 소요될 수 밖에 없다.

실제 안드로이드 단말을 사용하기 전에는 보드, 모니터, 셋탑박스, 핸드폰 단말에 이슈, 문제들을 재현할 수 있도록 테스트 환경 구축을 한다. 이러한 환경 구축을 위해 크로스 컴파일과 버닝(Burn, 이미지를 단말에 저장하는 말을 의미, CD/DVD를 굽는다 라는 표현과 같다고 보면 된다.) 시리얼 또는 패러릴 케이블을 이용하면, 최소 1시간에서 최대 3시간까지 소요되었다. <그림 1>과 같이 패러럴 포트를 이용해서 테스트를 진행하는 과정을 매번 했다. 로컬 PC에서 타겟 임베디드 보드에서 동작할 수 있는 바이너리 이미지를 타겟 보드로 전송하고, 그 디버그 내용을 받아서 테스트하는 과정이 일반적이었다. 따라서 많은 시간이 소요되곤 했다.

<그림 1> 패러럴 포트를 이용한 개발 환경 (출처 : http://www.ebay.com/)

개발 능력이 좋은 회사의 경우에는 이런 시간을 최소화하고, 생산 효율성을 높이기 위한 시도를 하였다. 또는 개발 플랫폼을 좋은 회사의 제품을 이용해 개발 및 테스트/디버깅 시간을 최소화 시킬 수 있었다. <그림 2>과 같이 OS 업체의 제품의 개발 도구를 이용한 방법이 있었다. MS 제품의 경우에는 Pocket Pc나 Windows Embedded 제품을 사용할 수 있었다. 그리고 <그림 3>과 같이 특정 하드웨어 업체의 에뮬레이터 툴을 이용하는 방법도 있다.

<그림 2> MS의 Visual studio를 이용한 pocket pc 테스트 환경 (출처 : http://msdn.microsoft.com/)

<그림 3> 하드웨어 업체, 삼성의 에뮬레이터 툴 (출처 : http://mobiforge.com/)

방송 셋탑박스의 경우는 방송스트리밍(MPEG-2, MPEG-4) 인코딩 데이터를 복사해 임베디드 시스템에서 재현을 할 수 있는 에뮬레이터를 개발한 경우가 있다. 그래서 core dump (메모리 영역 분석)나, 재현 가능 시나리오 및 바이너리 내용을 가지고 재현하면서 재현가능한 문제점을 해결하기도 했다. 마치 게임 에뮬레이터와 똑같은 원리라 할 수 있다. <그림 4>는 한 업체의 가상 머신 에뮬레이터 환경을 보여준다.

<그림 4> DTV Emulator (출처 : http://developer.lgappstv.com/)

하드웨어 업체나 전문 단말 소프트웨어업체의 에뮬레이터 환경은 가상화를 근간으로 하고 있다.
에뮬레이터가 완벽할수록 제품의 품질은 완벽해져 갔다. 저자가 근무했던 임베디드 미들웨어 업체에서는 이런 에뮬레이터를 보다 완벽하게 만들어 테스트/디버깅을 똑같이 재현할 수 있었다.

마찬가지로 <그림 5>의 구글의 안드로이드 에뮬레이터나 <그림 6>의 애플의 아이폰 에뮬레이터 등과 같이 App 개발자에게 개발환경을 제공하고 있다. 일부 기능을 제외하고는 많은 부분을 에뮬레이터에서 테스트/디버깅을 지원하고 있다. 특히 시스템 및 앱 로그 출력을 편하게 하여 높은 품질을 이룰 수 있도록 도와주고 있다.

<그림 5> 구글의 안드로이드 에뮬레이터 (출처 : http://www.akeric.com/)

참고로 구글 안드로이드 에뮬레이터 대신 성능 좋고 기능 좋은
Genymotion(http://www.genymotion.com/)이라는 에뮬레이터를 많이 사용하고 있다. non-commercial (기능 한정) 에게만 무료이니 http://widzard.tistory.com/23 를 참조해 사용하면 된다.

<그림 6> 애플의 iphone 개발환경 (출처 : http://www.hakbox.com/)

특히 바이너리를 서로 공유하여 문제나 이슈에 대한 재현을 쉽게 할 수 있도록 방법을 제공하고 있다. 그러나 안드로이드의 경우는 파편화나 UI 크기, 하드웨어 업체의 포팅 이슈가 있어서 단말기마다 테스트를 진행해야 하는 불편함이 존재하고 있다.

1.3 데스크탑 또는 서버 환경

오래된 역사를 가지고 있다는 MS 기반의 C++, Basic 또는 Delphi 데스크탑 어플리케이션의 경우는 작업 환경이 곧 개발 환경이기 때문에 쉽게 개발이 가능했다. 다만 운영체제에 의존성을 가지고 있기 때문에 다양한 운영체제에서 바이너리 실행을 할 수 있는 여건이 중요하다. 특히 MS 윈도우 운영체제 기반의 어플리케이션의 경우에는 윈도우 버전마다 조금씩 다른 변화가 있기 때문에 여러 MS 윈도우를 실행시킬 수 있는 환경을 두어야 했다. PC가 없는 환경이라면 VMware의 fusion, Oracle의 Virtual Box, parallels, funsion, boot camp와 같은 가상화 솔루션을 이용해서 한 대의 PC 에서 다양한 운영체제를 설치하여 어플리케이션을 테스트 했다.

이런 데스크탑 어플리케이션 개발의 불편함으로 한 때는 Java가 각광을 받기도 했다. MS 제품 뿐 아니라 리눅스와 같은 이질적인 운영체제에서도 실행 가능한 Java는 <그림 6> 처럼 한 번만 작성(설치)하면 어디서든 실행시킬 수 있다는 'Write once, run anywhere' 캐치 프레이즈로 이목을 끌기도 했다.

<그림 6> 윈도우 플랫폼의 Java의 전략 - Write once, run anywhere (출처 : http://dirkriehle.com/2011/06/30/the-java-ip-story/)

실제 많은 개발 업체들의 진행했던 예제를 들고 실제 어떻게 방향을 바꿀지 얘기하도록 한다.

Java라는 가상 머신(Virtual Machine) 상에서 동작하는 어플리케이션은 성능 이슈로 늘 문제가 있었다. 그러나 점점 PC 성능이 좋아지고, Java와 Servlet Container(resion, tomcat, jetty) 위에 동작하는 웹 서버(여기서 말하는 웹 서버는 Web/WAS를 총칭하는 의미)는 윈도우, 리눅스의 대부분의 운영체제에서도 충분히 개발/테스트/재현이 가능하게 되었다.

따라서 <그림 7>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.

이 방법은 2가지 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.

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 마땅히 할 수 있는 방법이 없다.

<그림 7> 일반적인 웹 서버의 개발 방식

이런 불편함을 제거하기 위해서 <그림 8>과 같이 하나의 가상 머신으로 통일화를 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 통일을 시킬 수 있고, 필요하다면 로컬 PC에서 충돌하지 않을 포트를 지정하여 서비스할 수 있다.

<그림 8> 가상머신 내에 웹 서버와 Database를 설치한 환경

이런 장점은 3가지를 제공한다. 위에서 이미 언급한 2문제를 해결할 뿐 아니라 더 좋은 장점들이 추가된다.

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 할지를 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 9>처럼 가상머신 위에 API 테스트 Layer (API 테스트베드) 를 두어 웹 서버의 환경을 두어 Acceptance Test 환경을 만들 수 있다. 이런 사례는 Facebook 나 Twitter의 개발자 센터와 같은 비슷한 모델로 구현이 가능하다 .

넷째, 사무실 IP에서 IDC로의 연결이 방화벽으로 막혀 있는 환경에서 유용하다. 전자 금융 거래법(일명 전금법)이 적용되거나, 개인정보를 가지고 테스트 해야 하는 상황에서는 <그림 7>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 JDK 8 위에서 동작하는 WAS를 테스트하기 위해서 JDK8을 설치했다가 테스트하고 다시 삭제해야 하는 경우가 있는데, 가상 머신을 이용해서 이런 번거로움을 해결할 수 있다.

<그림 9> 가상머신 위에 API Test Layer를 위치한 개발 환경

Practical한 개발자 환경은 <그림 9>에서 조금 더 진화한 <그림 10>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 10> <그림 9>에서 형상관리툴과 필요한 Utility가 포함된 개발 환경

가상머신이 존재하는 것은 간단한 작업을 하는 웹 서버를 테스트하기 위함보다는 여러 개의 웹 서버와 DB 또는 Nosql 이 존재하는 다양한 스토리지가 존재하는 테스트 환경을 테스트하는 경우가 더 많다. 이를 위해서는 가상 머신 내에서는 다양한 시스템을 설치하고 테스트 할 수 있는 환경을 <그림 11>과 같이 구성할 수 있을 것이다.

<그림 11> 연동가능한 개발 환경과 API Test Layer가 포함된 가상 머신

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

2. 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 것이다. 즉, virtual box를 실행시키려면 설정과 설치, 시작/중지/종료 등 이런 작업을 일일이 해야 하는 작업을 Vagrant를 이용하면 CLI(command line interface)기반에서 할 수 있다.

즉, <그림 12>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.

<그림 12> Oracle VM Virtual Box 화면

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

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

<그림 13>처럼 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.

<그림 13> Oracle의 Virtual Box 다운로드 화면 (출처 : https://www.virtualbox.org)

다음은 Vagrant 설치를 진행한다. <그림 14>와 같이 http://www.vagrantup.com/ 에 접속해서 하단 좌측 Download 버튼을 클릭한다.

<그림 14> vagrant 홈페이지 화면 (출처 : http://www.vagrantup.com/)

Download 화면(http://www.vagrantup.com/downloads.html)은 <그림 15>와 같다. 여기에서 Vagrant를 운영체제에 맞게 설치한다. 본 필자는 MAC OS X 를 사용하므로 MAC OS X 기준으로 설명하도록 하겠다.

<그림 15> vagrant 다운로드 화면 (출처 : http://www.vagrantup.com/downloads.html)

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
A `Vagrantfile` has been placed in this directory. You are now
ready to `vagrant up` your first virtual environment! Please read
the comments in the Vagrantfile as well as documentation on
`vagrantup.com` for more information on using Vagrant.

$ ls
Vagrantfile
```

Vagrantfile 파일은 다음과 같다. ruby 파일 템플릿이고, 주석과 프로퍼티 설명이 되어 있다.

```
# -*- mode: ruby -*-
# vi: set ft=ruby :

# 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|
  # All Vagrant configuration is done here. The most common configuration
  # options are documented and commented below. For a complete reference,
  # please see the online documentation at vagrantup.com.

  # Every Vagrant virtual environment requires a box to build off of.
  config.vm.box = "base"

  # Disable automatic box update checking. If you disable this, then
  # boxes will only be checked for updates when the user runs
  # `vagrant box outdated`. This is not recommended.
  # config.vm.box_check_update = false

  # Create a forwarded port mapping which allows access to a specific port
  # within the machine from a port on the host machine. In the example below,
  # accessing "localhost:8080" will access port 80 on the guest machine.
  # config.vm.network "forwarded_port", guest: 80, host: 8080

  # Create a private network, which allows host-only access to the machine
  # using a specific IP.
  # config.vm.network "private_network", ip: "192.168.33.10"
```

```
# Create a public network, which generally matched to bridged network.
# Bridged networks make the machine appear as another physical device on
# your network.
# config.vm.network "public_network"

# If true, then any SSH connections made will enable agent forwarding.
# Default value: false
# config.ssh.forward_agent = true

# Share an additional folder to the guest VM. The first argument is
# the path on the host to the actual folder. The second argument is
# the path on the guest to mount the folder. And the optional third
# argument is a set of non-required options.
# config.vm.synced_folder "../data", "/vagrant_data"

# Provider-specific configuration so you can fine-tune various
# backing providers for Vagrant. These expose provider-specific options.
# Example for VirtualBox:
#
# config.vm.provider "virtualbox" do |vb|
#   # Don't boot with headless mode
#   vb.gui = true
#
#   # Use VBoxManage to customize the VM. For example to change memory:
#   vb.customize ["modifyvm", :id, "--memory", "1024"]
# end
#
# View the documentation for the provider you're using for more
# information on available options.

# Enable provisioning with CFEngine. CFEngine Community packages are
# automatically installed. For example, configure the host as a
# policy server and optionally a policy file to run:
#
# config.vm.provision "cfengine" do |cf|
#   cf.am_policy_hub = true
#   # cf.run_file = "motd.cf"
# end
#
# You can also configure and bootstrap a client to an existing
# policy server:
#
# config.vm.provision "cfengine" do |cf|
#   cf.policy_server_address = "10.0.2.15"
# end
```

```
# Enable provisioning with Puppet stand alone.  Puppet manifests
# are contained in a directory path relative to this Vagrantfile.
# You will need to create the manifests directory and a manifest in
# the file default.pp in the manifests_path directory.
#
# config.vm.provision "puppet" do |puppet|
#   puppet.manifests_path = "manifests"
#   puppet.manifest_file  = "site.pp"
# end

# Enable provisioning with chef solo, specifying a cookbooks path, roles
# path, and data_bags path (all relative to this Vagrantfile), and adding
# some recipes and/or roles.
#
# config.vm.provision "chef_solo" do |chef|
#   chef.cookbooks_path = "../my-recipes/cookbooks"
#   chef.roles_path = "../my-recipes/roles"
#   chef.data_bags_path = "../my-recipes/data_bags"
#   chef.add_recipe "mysql"
#   chef.add_role "web"
#
#   # You may also specify custom JSON attributes:
#   chef.json = { :mysql_password => "foo" }
# end

# Enable provisioning with chef server, specifying the chef server URL,
# and the path to the validation key (relative to this Vagrantfile).
#
# The Opscode Platform uses HTTPS. Substitute your organization for
# ORGNAME in the URL and validation key.
#
# If you have your own Chef Server, use the appropriate URL, which may be
# HTTP instead of HTTPS depending on your configuration. Also change the
# validation key to validation.pem.
#
# config.vm.provision "chef_client" do |chef|
#   chef.chef_server_url = "https://api.opscode.com/organizations/ORGNAME"
#   chef.validation_key_path = "ORGNAME-validator.pem"
# end
#
# If you're using the Opscode platform, your validator client is
# ORGNAME-validator, replacing ORGNAME with your organization name.
#
# If you have your own Chef Server, the default validation client name is
# chef-validator, unless you changed the configuration.
#
#   chef.validation_client_name = "ORGNAME-validator"
```

```
end
```

주요 프로퍼티에 대한 설명을 진행한다. 자세한 내용은
http://docs.vagrantup.com/v2/vagrantfile/index.html 을 참조한다.

VAGRANTFILE_API_VERSION 은 Vagrantfile API/syntax 버전으로 상수 2 값을 지정해야 동작한다.

config.vm.box 의 값은 vagrant box add 명령어로 저장한 이름을 입력하면 된다. 만약 vagrant box add precise64 http://~ 라 하면 precise64가 config.vm.box의 값으로 지정하면 된다. 눈치 챘겠지만, 하나의 이름만 쓸 수 있는 것이 아니라, 여러 개의 이름으로 저장이 가능하다. config.vm.box 의 이름을 지정하지 않으면 base로 디폴트로 등록된다.

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

config.vm.network은 network에 관련된 설정을 수정할 수 있다. network 타인은 forwarded_port, private_network, public_network와 같은 식별자를 따로 두어 network 레벨의 설정을 진행할 수 있다. 특정 포트를 포워딩 하거나, DHCP, 고정 IP 등을 지정할 수 있다.

예를 들어 다음 설정의 내용은 가상 서버(guest)에서 8080으로 띄운 서버가 있다면 가상 머신을 실행한 OS(host, 앞으로 Host 라는 단어 사용)에서는 80 으로 바운드(bound) 하도록 한다는 내용이다.

```
config.vm.network "forwarded_port", guest: 80, host: 8080
```

그리고 vagrant 가상서버에 provisioning 툴을 CFEngine, VBoxManage, Puppet, Chef 중 하나로 쓸 수 있는 템플릿을 제공하고 있다.

저자는 ansible을 이용할 예정이니 때문에 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 16> 평상시 Virtual Box 상태

<그림 16>과 같이 평소 가상 머신의 상태는 “전원 꺼짐"으로 되어 있다. 사용자가 vagrant up 명령를 이용하면 <그림 17>과 같이 가상 머신의 상태는 “실행 중"으로 바뀐다.

<그림 17> vagrant up 명령 이후 가상 머신의 status가 “실행 중"으로 변경된 화면

vagrant up 명령 시 가상 머신이 실행되면서 관련 데몬들이 실행하고 로그가 출력된다.

```
$ vagrant up
Bringing machine 'default' up with 'virtualbox' provider...
==> default: Clearing any previously set forwarded ports...
==> default: Clearing any previously set network interfaces...
==> default: Preparing network interfaces based on configuration...
    default: Adapter 1: nat
    default: Adapter 2: hostonly
==> default: Forwarding ports...
    default: 9966 => 8888 (adapter 1)
==> default: Booting VM...
==> default: Waiting for machine to boot. This may take a few minutes...
    default: SSH address: 127.0.0.1:2222
```

```
    default: SSH username: vagrant
    default: SSH auth method: private key
==> default: Machine booted and ready!
==> default: Checking for guest additions in VM...
    default: The guest additions on this VM do not match the installed version of
    default: VirtualBox! In most cases this is fine, but in rare cases it can
    default: prevent things such as shared folders from working properly. If you see
    default: shared folder errors, please make sure the guest additions within the
    default: virtual machine match the version of VirtualBox you have installed on
    default: your host and reload your VM.
    default:
    default: Guest Additions Version: 4.2.0
    default: VirtualBox Version: 4.3
==> default: Setting hostname...
==> default: Configuring and enabling network interfaces...
==> default: Mounting shared folders...
    default: /vagrant => /development/work/idp-testbox
==> default: Machine already provisioned. Run `vagrant provision` or use the `--provision`
==> default: to force provisioning. Provisioners marked to run always will still run.
```

로그 상으로 확인했듯이 provision 된 것을 확인할 수 있다.
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
Bringing machine 'default' up with 'virtualbox' provider...
==> default: Clearing any previously set forwarded ports...
==> default: Clearing any previously set network interfaces...
==> default: Preparing network interfaces based on configuration...
    default: Adapter 1: nat
    default: Adapter 2: hostonly
==> default: Forwarding ports...
    default: 9966 => 8888 (adapter 1)
    default: 22 => 2222 (adapter 1)
==> default: Booting VM...
==> default: Waiting for machine to boot. This may take a few minutes...
    default: SSH address: 127.0.0.1:2222
    default: SSH username: vagrant
    default: SSH auth method: private key
==> default: Machine booted and ready!
==> default: Checking for guest additions in VM...
    default: The guest additions on this VM do not match the installed version of
    default: VirtualBox! In most cases this is fine, but in rare cases it can
    default: prevent things such as shared folders from working properly. If you see
```

```
    default: shared folder errors, please make sure the guest additions within the
    default: virtual machine match the version of VirtualBox you have installed on
    default: your host and reload your VM.
    default:
    default: Guest Additions Version: 4.2.0
    default: VirtualBox Version: 4.3
==> default: Setting hostname...
==> default: Configuring and enabling network interfaces...
==> default: Mounting shared folders...
    default: /vagrant => /development/work/idp-testbox
==> default: Machine not provisioning because `--no-provision` is specified.
```

2.3.3 vagrant suspend

가상머신을 잠깐 중지하려면 vagrant suspend 명령을 실행한다. <그림 18> 과 같이 가상 머신의 상태가 “저장 중"으로 바뀐다.

```
$ vagrant suspend
==> default: Saving VM state and suspending execution…
```

<그림 18> vagrant suspend 명령 이후 가상머신의 status가 “저장됨"으로 변경

2.3.3 vagrant resume

중지된 가상머신을 다시 재개하려면 vagrant resume을 실행하면 된다. <그림 17>과 같은 화면으로 출력한다.

```
$ vagrant resume
==> default: Resuming suspended VM...
==> default: Booting VM...
==> default: Waiting for machine to boot. This may take a few minutes...
    default: SSH address: 127.0.0.1:2222
    default: SSH username: vagrant
    default: SSH auth method: private key
    default: Warning: Connection refused. Retrying...
==> default: Machine booted and ready!
```

2.3.4 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 비교해볼때 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
```

2.3.5 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)

The VM is running. To stop this VM, you can run `vagrant halt` to
shut it down forcefully, or you can run `vagrant suspend` to simply
suspend the virtual machine. In either case, to restart it again,
simply run `vagrant up`.
```

2.3.6 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
Welcome to Ubuntu 12.04 LTS (GNU/Linux 3.2.0-23-generic x86_64)

 * Documentation:  https://help.ubuntu.com/
Welcome to your Vagrant-built virtual machine.
Last login: Wed Jul  9 12:48:04 2014 from 10.0.2.2


vagrant@web:~$ uname -a
Linux web 3.2.0-23-generic #36-Ubuntu SMP Tue Apr 10 20:39:51 UTC 2012 x86_64 x86_64
x86_64 GNU/Linux
```

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
Welcome to Ubuntu 12.04 LTS (GNU/Linux 3.2.0-23-generic x86_64)

 * Documentation:  https://help.ubuntu.com/
Welcome to your Vagrant-built virtual machine.
Last login: Wed Jul  9 13:25:58 2014 from 10.0.2.2
vagrant@web:~$
```

2.3.6 vagrant halt

vagrant halt 명령을 내려 가상머신을 종료한다. <그림 16>의 상태로 복귀하게 된다.

```
$ vagrant halt
```

```
==> default: Attempting graceful shutdown of VM…
```

2.3.7 vagrant package

vagrant package 를 통해서 패키징 하여 다른 개발자에게 전달할 수 있다.

```
$ vagrant package
==> default: Clearing any previously set forwarded ports...
==> default: Exporting VM...
==> default: Compressing package to: /development/work/idp-testbox/package.box

$ ls -al package.box
total 2154256
-rw-r--r--   1 knight  wheel  1102929920  7  9 22:38 package.box
```

가상 머신을 virtual box 가상머신 리스트에서 삭제를 할 수 있다.
<그림 19>와 같이 vagrant up을 해서 “vagrant-test_default_1401100903183_2186” 이란 이름을 가진 가상머신을 실행했다.

<그림 19> vagrant up 으로 가상머신이 “실행 중”인 화면.

2.3.7 vagrant destroy

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

```
$ vagrant destroy
    default: Are you sure you want to destroy the 'default' VM? [y/N] y
==> default: Forcing shutdown of VM...
==> default: Destroying VM and associated drives...
==> default: Running cleanup tasks for 'ansible' provisioner...
```

virtual box 가상머신 리스트 화면을 보면 “vagrant-test_default_1401100903183_2186” 이름을 가진 가상머신은 삭제되었다.

<그림 20> 실행 중이었던 가상머신이 삭제 된 화면

2.3.8 vagrant reload

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

```
$ vagrant reload
==> web: VM not created. Moving on...
==> testing: Attempting graceful shutdown of VM...
==> testing: Clearing any previously set forwarded ports…
...
```

2.3.9 가상 서버 이름 지정하기

Vagrantfile에 지금까지 name을 주지 않았다. 아래와 같이 config.vm.box와 config.vm.box_url 만 지정하면 <그림 21>과 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 21> 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 21>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 22> 예제 실행 후 가상 서버의 디폴트 이름이 web으로 변경됨

2.3.10 여러 가상 서버(Multiple VM) 을 통제 하기

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 23>과 같은 화면을 볼 수 있다. 두 개의 가상머신이 실행 중인 것을 확인할 수 있다.

```
$ vagrant up
Bringing machine 'web' up with 'virtualbox' provider...
Bringing machine 'mysql' up with 'virtualbox' provider...
==> web: Clearing any previously set forwarded ports...
==> web: Clearing any previously set network interfaces...
==> web: Preparing network interfaces based on configuration...
    web: Adapter 1: nat
    web: Adapter 2: hostonly
==> web: Forwarding ports...
    web: 22 => 2222 (adapter 1)
==> web: Booting VM...
==> web: Waiting for machine to boot. This may take a few minutes...
    web: SSH address: 127.0.0.1:2222
    web: SSH username: vagrant
    web: SSH auth method: private key
==> web: Machine booted and ready!
==> web: Checking for guest additions in VM...
    web: The guest additions on this VM do not match the installed version of
    web: VirtualBox! In most cases this is fine, but in rare cases it can
    web: prevent things such as shared folders from working properly. If you see
    web: shared folder errors, please make sure the guest additions within the
    web: virtual machine match the version of VirtualBox you have installed on
    web: your host and reload your VM.
    web:
    web: Guest Additions Version: 4.2.0
    web: VirtualBox Version: 4.3
==> web: Setting hostname...
==> web: Configuring and enabling network interfaces...
==> web: Mounting shared folders...
    web: /vagrant => /development/work/idp-testbox2
==> web: Machine already provisioned. Run `vagrant provision` or use the `--provision`
==> web: to force provisioning. Provisioners marked to run always will still run.
==> mysql: Clearing any previously set forwarded ports...
==> mysql: Fixed port collision for 22 => 2222. Now on port 2200.
==> mysql: Clearing any previously set network interfaces...
==> mysql: Preparing network interfaces based on configuration...
    mysql: Adapter 1: nat
    mysql: Adapter 2: hostonly
==> mysql: Forwarding ports...
    mysql: 22 => 2200 (adapter 1)
```

```
==> mysql: Booting VM...
==> mysql: Waiting for machine to boot. This may take a few minutes...
    mysql: SSH address: 127.0.0.1:2200
    mysql: SSH username: vagrant
    mysql: SSH auth method: private key
==> mysql: Machine booted and ready!
==> mysql: Checking for guest additions in VM...
    mysql: The guest additions on this VM do not match the installed version of
    mysql: VirtualBox! In most cases this is fine, but in rare cases it can
    mysql: prevent things such as shared folders from working properly. If you see
    mysql: shared folder errors, please make sure the guest additions within the
    mysql: virtual machine match the version of VirtualBox you have installed on
    mysql: your host and reload your VM.
    mysql:
    mysql: Guest Additions Version: 4.2.0
    mysql: VirtualBox Version: 4.3
==> mysql: Setting hostname...
==> mysql: Configuring and enabling network interfaces...
==> mysql: Mounting shared folders...
    mysql: /vagrant => /development/work/idp-testbox2
==> mysql: Machine already provisioned. Run `vagrant provision` or use the `--provision`
==> mysql: to force provisioning. Provisioners marked to run always will still run.
```

<그림 23> 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 글을 참고하면 힌트를 얻을 수 있을 것이다.
http://programmaticponderings.wordpress.com/2014/02/27/multi-vm-creation-using-vagrant-and-json/

3. 정리
지금까지 가상 머신을 활용한 배경을 이야기 했고, vagrant에 대한 설치, 소개를 모두 마쳤다. 다음 회에서는 Ansible을 활용한 방식을 설명할 예정이다.

참고 자료

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