Ansible과 Vagrant를 이용한 개발 환경 구축 #2
- 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 2 순서

- Vagrant/Ansible을 이용한 개발 환경 구축 예제

1. 배경

Part 1에서 Oracle Virtual Box와 Vagrant를 설치했고 간단한 provision(배포) 를 설명했다.

오픈 소스 배포 툴 (propvision tool)이며 환경 설정 툴 (configuration management tool)이고 remote 서버에 접근후 명령 실행 툴 (ad-hoc task execution) 은 puppet, chef, ansible, salt 가 유명하다. 여기에서는 salt를 제외한 나머지 puppet, chef를 언급하고 주제인 ansible을 설명할 예정이다.

puppet과 chef, ansible, salt는 오픈 소스 시스템 설정 관리 소프트웨어(도구)로서, 각각 고유한 설정 명세 언어를 이용하여 환경 설정 파일 또는 스크립트를 배포할 수 있는 툴이다. 효율적인 개발환경 및 서버 관리를 지원하여 서버 환경의 인프라를 최적화할 수 있다.

저자는 국내의 한 포털업체서 1000대가 넘는 서버에 오픈 소스 web/was/app을 설치하고, 환경 설정 파일, 스크립트, 유틸리티를 배포하는 작업을 맡아서 진행했다. 이를 바탕으로 Web/WAS/APP 배포 서버를 개발했었다. 당시에는 쓸만한 유틸리티가 전무한 상태이기 때문에 직접 만들어서 사용했다. 이렇게 만든 provision 툴을 이용하여 Infra Engineer로부터 서버를 받자마자 바로 설치와 배포를 완료시켜 web/was 서버를 실행할 수 있도록 했다. Concurrent 한 환경으로 개발하여 수십대의 서버를 순식간에 설치하여 서비스가 되도록 하였다. 최적화된 인프라를 지탱하는 솔루션으로 in-house 용으로 개발하여 사용하였다.

그러나 이런 툴이 없는 IT 회사나 조직에서는 provision 을 하지 못하고 열악한 방식으로 구현하였다. 서버에 접속해서 소스를 다운받고 컴파일 한 후 서버 재시작하는 구조로 진행했다. 그리고 web 또는 was 환경 설정 파일은 ftp로 복사하는 형태로 진행했다. 그리고 이렇게 한 대의 서버를 구축하고 난 후, 해당 서버의 로그를 제외한 바이너리와 설정파일, 스크립트들을 모두 한번에 복사해서 다른 서버로 파일 복사하는 경우가 있다. 만약 서버 셋팅이 20대라면 20대에 일일이 접속하여 설치/설정을 진행하다 보니, 시간이 많이 소비되었다. 한국 뿐 아니라 전세계적으로 아직까지도 이렇게 작업하고 있는 현실이다.

참고로 아마존 AWS 서비스에서도 초기 버전에는 provision 툴이 없었다. 그러나 Cloud Formation(http://aws.amazon.com/ko/cloudformation/)이라는 툴을 공개하여 provision이 쉽게 되도록 지원하고 있다.

<그림 1> AWS CloudFormation 툴 (출처 : http://aws.amazon.com/ko/cloudformation/)

최근 전세계적으로 Devops 개발자들의 노력으로 말미암아 오픈소스 라이브러리에서 오픈 소소 툴로 승화되고 있다. In-house 용 플랫폼으로 쓰던 제품들이 일부 오픈소스 및 상용화하면서 provision 툴에 대한 마인드가 변화되었다.

Devops에 대한 소개

제임스 어쿼트(James Urquhart)는 클라우드에서 돌아가는 요즘 애플리케이션에 여전히 회복력과 내구성이 필요하며, 모니터링이 필요하고, 크게 변하는 부하에 대응할 필요가 있음을 설명하면서 이를 확실히 했다. 그러나 그는 예전에는 IT/기반 체계 운영에서 제공된 이런 기능들이 이제는 애플리케이션의 일부가 되어야하며, 특히 "서비스로서의 플랫폼(Platform as a service)" 환경이 되어야 한다고

언급했다. 운영은 사라지지 않으며, 개발의 일부가 된다. 빅 데이터, 웹 성능 최적화, 애플리케이션 미들웨어, 어마어마하게 분산된 환경에서의 내구성을 이해하고 있는 최고의 개발자 따위를 마음에 그리기보다, 개발 팀에 있는 운영 전문가가 필요하다. 기반 체계는 사라지는 것이 아니며, 코드로 옮겨간다. 그리고 기반 체계, 시스템 관리자 및 기업의 IT 부서를 담당한 사람들은 발전해서 그들의 기반 체계를 유지할 수 있는 코드를 작성할 수 있다. 고립되기 보다는, 애플리케이션을 만든 개발자와 협동하고 협력할 필요가 있다. 이런 움직임이 비공식적으로 알려진 "개발운영(DevOps)"이다.

출처 : http://www.hanbit.co.kr/network/view.html?bi_id=1831

현재까지 오픈 소스 Provision 툴로는 Puppet, Chef가 가장 유명하고 많은 개발자들이 쓰고 있었다. 그러나 최근에는 Salt 나 Ansible이라는 툴이 떠오르고 있는 상황이다. Debian 운영체제 안에서 이 4개 오픈 소스 툴의 다운로드 수를 <그림 2>에서 확인할 수 있다. puppet이 가장 많이 사용하고 있으며, 그 뒤로 chef, salt, ansible의 순서를 보여주고 있다. (출처 : http://redmonk.com/sograd y/2013/12/06/configuration-management-2013/)

<그림 2> Debian에서의 puppet, chef, salt, ansible 설치 횟수 (출처 : http://redmonk.com/)

<그림 3> Debian에서의 puppet을 제외한 chef, salt, ansible 설치 횟수 (출처 : http://redmonk.com/)

한편 해커 뉴스 커뮤니티에서 언급되고 있는 수로 따지면 현재 ansible이 puppet이나 chef, salt보다 더 높은 인기를 누리고 있음을 <그림 4>에서 확인할 수 있다.

<그림 4> 해커 뉴스 커뮤니티에서 주목하고 있는 ansible

Github프로젝트에서 git 저장소에 대한 관심을 star 마킹을 하여 체크할 수 있는데, <그림 5>와 같이 네 개의 오픈 소스 provision툴 중에서는 ansible이 가장 인기가 많은 상태이다.

<그림 5> puppet, chef, ansible, salt GitHub 저장소에서의 star 마킹 횟수

또한 GitHub은 public git 저장소를 자기 계정의 git 저장소로 복사(fork)할 수 있는 기능을 제공하는데, ansible이 <그림 6> 과 같이 비교 표본 중 가장 많은 fork 숫자를 포함하고 있다.

<그림 6> puppet, chef, ansible, salt GitHub 저장소에서의 fork 횟수

puppet과 chef 라는 유명한 툴이 있음에도 불구하고 ansible은 빠르게 Devops개발자들에게 최근에 많은 호감을 가지고 점점 인기가 높아지고 있다.

구글 트렌드에 따르면, <그림 7>과 같이 2013년 초부터 급격히 ansible에 대한 관심이 생긴 것을 확인할 수 있다.

<그림 7> 구글 트렌드 결과 (출처 : http://www.google.com/trends)

많은 개발자들이 slashdot.org에서 다양한 의견을 펼친 것이 있다. 아래 링크를 살펴보면 재미있을 것이다.
http://tech.slashdot.org/story/13/11/22/0239230/review-puppet-vs-chef-vs-ansible-vs-salt

2. ansible의 소개와 설치

2.1 ansible 선택 이유

저자는 agent기반과 rsh/ssh기반으로 provision 방식을 개발/활용하니, 상황에 따라 맞는 형태가 있음을 알게 되었다. agent 기반이 확장성이나 추후 대규모 provision를 할 경우 매우 효과적인 측면이 있다. 또한 agent를 provision뿐 아니라 다양한 방식으로 활용이 되는 장점이 있다. 대신 provision 서버와 통신하는 부분이 고도화된다. 따라서 빠르고 간단한 provision을 할 수 없다.

반면, rsh/ssh provision 방식은 빠른 provision이 가능하다. 그러나 수천 대의 서버를 통제하려면 rsh/ssh 기반은 한계가 있다.

저자는 주관적으로 puppet과 chef와 ansible의 비교해 보았다. 일부 내용은 주관적이며, 관련 자료는 Part 2 문서의 부록을 참고한다.

이름	개발 언어	정의	agent 인스톨 여부	통신 방법	공유 리파지토리	학습 시간	시스템 복잡성	github 활동성

puppet	ruby	DSL (독자)	필요	http ssl	존재	중	중	중
chef	ruby	DSL (ruby기반)	필요	STOMP / rest	존재	상	상	중
ansible	python	yaml	불필요	ssh/json	존재하지 않음	하	하	상

가장 많이 사용하고 있는 puppet이나 chef 대신 ansible을 선택하게 된 배경은 다음의 조건이었다. salt라는 provision 툴도 선택 가능성 안에 있지만, ansible이 좀 더 community 활동성이 좋아서 이를 선택했다.

- ➢ Mac/Linux 환경를 지원해야 한다. (윈도우 환경은 지원하지 않아도 됨)
- ➢ Agent 기반이 아니고 ssh 기반이어야 한다. (추후 상용 환경에서 사용할 때 Agent 기반이면 방화벽 이슈, agent 데몬 관리라는 불편한 점이 존재한다.)
- ➢ 쉽게 사용할 수 있어야 한다. Learning curve가 높지 않아야 한다.
- ➢ DSL 대신 최대한 Standard을 사용해야 한다.
- ➢ 테스트 환경(vagrant)에서 사용할 수 있어야 하고 서버 배포 환경에서도 사용할 수 있어야 한다.
- ➢ 설정 내용은 읽기 쉬워야 한다. 유지보수가 편해야 하며 인수인계가 쉽고 가능해야 한다.
- ➢ 자동 배포 환경이 쉬워야 한다.
- ➢ 병렬 실행이 되어야 한다.
- ➢ 앞으로 많이 개발 가능성이 높은 오픈소스(github 활동성) 여야 한다.
- ➢ 멱등성(idempotence)이 보장되어야 한다.

* 멱등성(idempotence) 이란?
연산을 여러 번 적용하더라도 결과가 달라지지 않는 성질을 멱등성(idempotence) 이라 한다. puppet, chef, ansible 등은 모두 이런 특성을 가지고 있다.
쉽게 말하면, rest api의 경우 get, head,put, delete 메소드는 멱등성을 가지고 있다. 그러나 post는 상태를 변하시키기 때문에 멱등성이 없다. 서버의 어떤 변화를 주지 않기 때문이다.

* ansible 툴에서의 멱등성이란?
여러번 ansible 툴을 사용하더라도 동일한 결과값을 나올 수 있도록 제공되는 형태여야 한다. 즉 매번 다른 결과가 나오거나 에러가 나온다면 비멱등성(non-idempotent) 하다고 할 수 있다. ansible 툴의 거의 대부분의 모듈은 멱등성을 제공한다. 또한 멱등성을 제공하기 위해서는 조건절을 제공하고 있다.

예를 들자면, 처음 ansible 스크립트를 실행 후 다시 실행을 했다. 상황에 따라서는 파일이 append가 될 수 있다. 그러나 멱등성의 원칙은 언제나 실행은 해도 결과가 동일하게 나온다.

또한 파일/디렉토리를 생성또는 삭제하는 ‘create:’, ‘remove:’ 같은 ansible 모듈을 실행할 때 ‘when:’ 조건절을 이용할 수 있다. 상태가 변경되면 ‘changed_when:’ 또는 ‘failed_when:’을 사용할 수 있다.

대부분의 ansible 모듈이 멱등성(idempotence)를 보장한다라는 의미는 상태(status)를 파악할 수 있다는 의미를 가진다.

원래 제공한 ansible 모듈은 대부분 멱등성를 제공하나 개발자/시스템 운영자가 만든 ansible 스크립트는 멱등성를 제공하지 못할 수 있다. 따라서 멱등성이 깨지지 않도록 주의해야 한다.

2.2 ansible 소개

ansible 이라는 회사(ansible inc) 가 ansible이라는 오픈소스(https://github.com/ansible/ansible)를 기반으로 ansible tower(http://www.ansible.com/home) 를 판매하고 있다. ansible 오픈 소스는 무료이나 ansible tower는 유료이다. 여기서 얘기하는 내용은 오픈 소스 버전이며, ansible tower는 유료이므로 이 글의 작성 범위에서 벗어나기 때문에 언급하지 않는다. ansible tower는 ansible 을 쉽게 사용할 수 있는 UI 버전과 LDAP을 지원하고 있다.

ansible provision 툴은 현재 Atlassian, Twitter, VeriSign, EA, Evernote, Nasa, Gopro, rackspace, juniper, mapr 등 해외 유명업체에서 사용중이다.

<그림 8> ansible 홈페이지 (출처 : http://www.ansible.com)

ansible은 application과 시스템을 쉽게 배포할 수 있도록 하는 간단한 IT 자동화 플랫폼이다. application을 배포, 업데이트할 때마다 매번 서버에 접속해서 들어가서 스크립트를 실행하지 않아도 자동화된 형태의 언어로 agent 없는 리모트 환경에 ssh를 이용하여 접근하여 작업할 수 있다.

현재 ansible은 오픈소스 (GNU GENERAL PUBLIC 라이선스)이며, <그림 9>과 같이 https://github.com/ansible/ansible 에서 소스를 제공하고 있다. 특히 793명의 contributer가 있는 것은 가장 활발히 움직이고 있음을 의미한다. puppet이나 chef에서 비해서 많은 개발자들이 참여하고 있다. 현재 Github 프로젝트 중 Python top 10 프로젝트 중 하나 이다.

<그림 9> ansible github 페이지 (https://github.com/ansible/ansible)

현재 (2014.7.23) 버전은 1.5.3 이며 조만간에 1.6.0이 출시될 예정이다.

ansible은 ssh(디폴트)로 리모트 서버로 연결후, “ansible module”(이하 ansible 모듈)이라 불리는 작은 프로그램을 전송하여 실행한다. 실행이 완료후에는 해당 ansible 모듈은은 삭제한다. ssh로 연결만 된다면 어떤 서버에도 접근하여 라이브러리 모듈을 저장할 수 있다.

ansible 모듈은 JSON으로 리턴만 하면 사용할 수 있기 때문에 ruby, bash, ptyhon 등 어떤 언어에서라도 사용될 수 있다. 그리고 플러그인을 제공하여 확장이 가능하고 callback, python api, 추가 기능을 구현할 수 있는 큰 특징들이 있다. 최근에 이런 plugin이 확장되면서 좋은 기능이 추가되고 있다.

ansible은 YAML 이라는 아주 간단한 포맷의 언어를 사용하고 있다. 쉬운 영문을 기반으로 한 YAML 을 이용하여 자동화 job을 기술할 수 있도록 한다. YAML에 대한 설명은 다음 장에서 진행한다.

ansible은 YAML 기반으로 된 playbook이라는 설정 파일을 자동화된 job을 정의한다. 설치부터 복사까지 모든 작업은 이 playbook에서 정의할 수 있다. 그리고 inventory 라는 파일을 이용하여 서버 설정을 할 수 있다.

<그림 10> ansible 아키텍처 (출처 : http://terry.im/wiki/terry/Ansible.html)

2.3 ansible 설치

ansible은 agent가 필요없기 때문에 서버에만 설치하면 된다. 서버를 ansible에서는 control machine이라 부른다. 설치 방법은 소스, python, yum, apt, freebsd, mac osx에서 쉽게 설치할 수 있다.
http://docs.ansible.com/intro_installation.html#installing-the-control-machine

저자는 Mac OSX 사용자이므로 homebrew를 이용하여 ansible을 설치했다.

```
$ brew update
$ brew install ansible
```

OS X Mavericks버전에서는 pip 설치시에는 컴파일러 이슈가 발생할 수 있다. 이를 해결하기 위해서는 아래와 같이 사용하여 ansible을 설치한다.

```
$ sudo CFLAGS=-Qunused-arguments CPPFLAGS=-Qunused-arguments pip install ansible
```

만약, 그래도 clang: error: unknown argument: '-mno-fused-madd' [-Wunused-command-line-argument-hard-error-in-future] 이라는 에러 문구가 표시하며 해결이 되지 않는다면 다음을 실행하여 ansible을 설치한다. XCode 5.1.1. 부터는 unknown argument는 컴파일 에러가 나도록 바뀐 듯 하다.

```
$ ARCHFLAGS=-Wno-error=unused-command-line-argument-hard-error-in-future pip install ansible
```

참고로 ansible를 windows내 설치을 공식적으로 지원하지는 않지만, 개인 블로그가 cygwin을 활용하여 windows에서도 ansible을 사용할 수 있도록 가이드를 작성한 것이 있으니 참고한다.
https://servercheck.in/blog/running-ansible-within-windows

다음은 ansible 의 리모트 호스트(remote host), managed node는 특별히 할 것은 없다. ansible서버에서 해당 node로 ssh 연결이 쉽게 될 수 있도록 해당 node에서 ssh 인증키를 생성하고 인증 키를 ansible 서버에서 갖고 있도록 하면 된다.

그리고 node 에서는 python 2.4 이상의 버전이 설치되어 있으면 된다. 만약 2.4 버전 미만의 python버전이 설치되어 있으면, 2.4 이상으로 업그레이드를 하거나 python-simplejson을 반드시 설치해야 한다.

```
$ pip install simplejson
```

simplejson으로 눈치 챘겠지만 json 형태로 데이터가 전달된다. 에러 또는 디버그시에 json형태로 결과를 리턴받는다. 아래 예제는 저자의 생각 없는 copy & paste를 통해서 발생한 에러 정보를 json으로 보여주고 있는 것을 보여주고 있다.

```
TASK: [mysql | start mysql] ***************************************************
failed: [192.168.1.50] => {"failed": true, "item": ""}
msg: service not found: tomcat
```

```
TASK: [mysql | shell /usr/bin/mysql -u root --password= -e "CREATE DATABASE IF NOT EXISTS hello; USE hello; CREATE TABLE person (id INT PRIMARY KEY AUTO_INCREMENT,name VARCHAR(50));"] ***
failed: [192.168.1.50] => {"changed": true, "cmd": "/usr/bin/mysql -u root --password= -e \"CREATE DATABASE IF NOT EXISTS hello; USE hello; CREATE TABLE person (id INT PRIMARY KEY AUTO_INCREMENT,name VARCHAR(50));\" ", "delta": "0:00:00.007686", "end": "2014-07-28 06:58:35.883732", "item": "", "rc": 1, "start": "2014-07-28 06:58:35.876046"}
stderr: ERROR 1050 (42S01) at line 1: Table 'person' already exists
```

3. ansible의 yaml과 playbook 소개

3.1 yaml 소개

위키(http://ko.wikipedia.org/wiki/YAML)에 따르면, yaml 은 xml, c, python, perl에서 정의된 email 양식에서 개념을 얻어 만들어진 '사람이 쉽게 읽을 수 있는' 데이터 직렬화 양식이다. yaml 이라는 이름은 'YAML ain't Markup Language' (YAML은 마크업 언어가 아니다.) 라는 재귀적인 이름에서 유래되었으나, 'Yet Another Markup Language' (또 다른 마크업 언어) 이 현재 공식적인 약자이다. yaml 은 마크업보다는 데이터 중심, 즉 serialization format 이라는 의미를 강조하고 있다.

발음은 '야믈' 이라 발음한다. 마치 camel과 음율을 맞추는 의도를 가지고 있다. 공식 싸이트는 http://yaml.org/ 이다.

<그림 11> yaml 공식 싸이트 http://yaml.org

yaml은 2001년에 1.0 (http://www.yaml.org/spec/history/2001-05-26.html)이 퍼블리싱되었다. 2009년에는 1.2(http://yaml.org/spec/1.2/spec.html#id2759572)가 퍼블리싱 되었는데. 이 때 json간의 관계를 설명이 되어 있다. 1.2의 변경은 yaml을 json과의 호환을 위해서이다. json과 yaml은 사람이 읽기 편한 데이터 교환 포맷을 목표로하고 있다. json은 간결함과 범용성에 초점을 둔다면, yaml은 json의 장점을 포함하고 임의의 고유 데이터 구조에 대한 직렬화(serialization)을 지원한다.

yaml은 모든 데이터를 리스트, 해쉬, 스칼라 데이터의 조합으로 적절히 표현할 수 있다. 문법이 상대적으로 쉽고 가독성이 좋다. 주요 특징은 다음과 같다.

- YAML 문자열은 UTF-8 또는 UTF-16과 같이 출력 가능한 유니코드 문자집합을 이용한다.
- 공백 문자를 이용한 들여쓰기로 구조체를 구분한다. 그러나 탭문자를 들여쓰기에 사용하지 않는다.
- 리스트 요소는 여러 줄에 쓸 때에는 하이픈(**-**)으로 시작하는 한 줄에 하나의 요소를 표현하며, 한 줄에 모아 쓸 때에는 대괄호(**[]**)를 이용하며 쉼표로 각 요소를 구분한다.
- 해쉬는 콜론 기호를 이용해서 **키:값**의 형태로 한 줄에 하나를 표현하거나, 한 줄에 모아 쓸 때에는 중괄호(**{}**)를 이용하며 쉼표로 각 요소를 구분한다.
- 간단한 값(스칼라 값)은 보통 아무 표시를 하지 않으나 따옴표("")나 작은 따옴표(*'')를 이용해 둘러쌀 수 있다.*
- 따옴표 안에서 특수 문자는 C언어 스타일(역슬래쉬키와 함께쓰이는 제어문자 예. \n)로 표시한다.
- 블록 값은 보존(|) 또는 접기(>)의 선택 지시자로 나눈다.
- 하나의 스트림에 있는 여러 개의 문서는 하이픈 3개(---)로 나누며, 마침표 세개(...)로 스트림의 끝을 나타낸다.
- 반복되는 노드는 기본적으로 &를 통해 나타내며, * 문자 이후의 내용을 참조한다.
- 주석은 #으로 표시하며, 한 줄이 끝날 때까지 유효하다.
- 노드들은 타입과 느낌표로 시작해 URI 주소를 지시하는 태그를 통해 라벨이 붙는다.

xml 예제를 yaml으로 변경한 사례를 통해서 yaml을 이해한다. xml 예제는 다음과 같다.

```
<recipe>
  <title>Macaroni and Cheese</title>
  <description>My favorite comfort food.</description>
  <author>Brian Genisio</author>
  <timeToPrepare>30 Minutes</timeToPrepare>
  <ingredients>
```

```
    <ingredient>
      <name>Cheese</name>
      <quantity>3</quantity>
      <units>cups</units>
    </ingredient>
    <ingredient>
      <name>Macaroni</name>
      <quantity>16</quantity>
      <units>oz</units>
    </ingredient>
  </ingredients>
</recipe>
```

이를 yaml로 바꾸면 다음과 같다. 특별한 설명을 하지 않아도 attribute의 key와 value값을 콜론(:) 단위로 정의하고 list에 대해서는 하이픈(-)으로 표시한다.

```
Recipe:
  Title:  Macaroni and Cheese
  Description:  My favorite comfort food.
  Author:  Brian Genisio
  TimeToPrepare: 30 Minutes
  Ingredients:
    -
      Name:     Cheese
      Quantity: 3
      Units:    cups
    -
      Name:     Macaroni
      Quantity: 16
      Units:    oz
```

그리고, json 예제를 yaml으로 변경한 사례를 통해서 yaml을 이해한다.
json 예제는 다음과 같다.

```
'[
    "apple",
    {
        "bar": ["baz", "kwa", 3.0, 5]
    }
]'
```

이를 yaml 예제로 바꾼다면 아래와 같다.

```
- apple
- bar:
    - baz
    - kwa
```

```
- 3.0
- 5
```

ansible뿐 아니라 대부분의 소프트웨어에서는 yaml 설정파일의 확장자는 yml 로 정의해서 사용하고 있다.

참고로 독자에게는 생소해 보일 수 있는 yaml은 travis CI, redmine, google app engine, cassandra, google-code-prettify외 많은 오픈소스및 솔루션 에서 설정 파일을 정의하는데 사용하고 있다.

3.2 jinja2 소개

yaml은 정적이다. ansible에서는 이를 동적으로 표현할 수 있는 무엇인가가 필요했다. jinja2 (http://jinja.pocoo.org/docs/) 라는 python 라이브러리를 이용해서 yaml 문법의 단점을 보완했다. 일종의 템플릿 언어(java의 freemarker, rythm)와 같다.

'ntp server 1.1.1.1' 로 표현했던 것을 변수 선언부에 'ntp_variable = 2.2.2.2'를 선언하고 설정에는 'ntp {{ntp_variable}}'로 표현가능하다.

jinja2 파일은 사실상 템플릿 엔진을 사용해서 만들 파일이기 때문에 확장자는 .j2 로 지정한다. 그래서 ansible예제에서 종종 보이는 템플릿(template) 디렉토리 밑의 파일들이 .j2로 되어 있는 것을 볼 수 있다.

예를 들어 mysql 의 client.cnf.j2 파일을 아래 예제와 같이 저장할 수 있다 .만약 utf8 변수가 true 이면 default-character-set 의 값을 utf8로 바꾸라는 정보를 추가할 수 있다.

```
[client]
{% if utf8 %}
# apply utf8
default-character-set = utf8
{% endif %}
```

단조로울 수 있는 환경 배포가 python 모듈 jinja2를 통해서 쉽게 배포 환경을 구성할 수 있다.

3.3 playbook 소개

playbook은 ansible의 환경 설정, 배포를 가능케 하는 언어이다. 리모트 서버에 접속하여 무엇가를 실행시키는 정책을 기술한다. 사람이 쉽게 읽을 수 있는 yaml 문법을 채용하여 정책을 기술한다. playbook은 정책을 기술하는데, ansible module을 활용한다.

playbook의 기본 레벨은 환경설정이나 배포에 유용하게 사용할 수 있다. 좀 더 고급 단계에서는 특정 서버에 무엇인가를 전달하거나, 로드 밸런서나 모니터링하는 복잡한 환경에서 사용할 수 있다.

각 playbook은 하나 또는 하나 이상의 ‘play’를 둔다. play의 목적은 여러 호스트들에 잘 정의된 ‘role’과 ‘task’를 매핑하는 역할을 한다. <그림 12>는 playbook의 구성요소를 간단하게 설명하고 있다.

<그림 12> ansible model (출처 : http://www.jedelman.com/home/ansible-for-networking)

‘role’은 file들을 include 하거나 하나로 합칠 수 있는(combile) 아이디어를 기반으로 만들어졌다. 재사용 가능한 추상화를 가능케 한다. ‘task’는 ansible 모듈의 호출을 의미한다.

role의 개념을 예제를 통해서 설명한다.

tasks/foo.yml 이라는 task를 정의한 파일이다.

```
---
# possibly saved as tasks/foo.yml

- name: placeholder foo
  command: /bin/foo

- name: placeholder bar
  command: /bin/bar
```

playbook 안에 task를 정의한 tasks/foo.yml을 include 하는 것이 가능하다.

```
tasks:
```

```
  - include: tasks/foo.yml
```

다음은 playbook 안에는 아래와 같은 형태로 개발하는데. include 를 활용하여 복잡하지 않도록 할 수 있다.

```
- name: this is a play at the top level of a file
  hosts: all
  remote_user: root

  tasks:
  - name: say hi
    tags: foo
    shell: echo "hi..."

- include: load_balancers.yml
- include: webservers.yml
- include: dbservers.yml
```

role를 좀 더 관리를 편하게 하기 위해 사용할 수 있다. host inventory 파일에 정의한 서버 그룹별로 각각 나우어 provision할 수 있도록 할 수 있다. 예를 들어 inventory 파일에서 dbservers와 webservers로 나눈 경우라면 role의 개념을 잘 이용하여 설치 정보를 디렉토리로 나눌 수 있다. common, dbservers, webservers 이렇게 디렉토리를 나누어서 db를 설치할 때는 common 디렉토리의 yml 파일과 dbservers의 yml을 읽어 설치하게 하고 web 서버를 설치할 때는 common 디렉토리의 yml 파일과 webservers 디렉토리의 yml을 설치하게 할 수 있다.

변수 선언은 vars 문 안에서 정의한다. http_port는 80으로 초기화한다.

```
- hosts: webservers
  vars:
    http_port: 80
```

미리 정의된 base_path 변수의 값의 app 디렉토리 스트링 값을 app_path에 저장한다.

```
- hosts: app_servers
  vars:
      app_path: "{{ base_path }}/app"
```

그것 외에 변수의 디폴트값을 넣는 다거나, for 문을 동작시킬 수 있다.

```
{{ some_variable | default(5) }}

{% for host in groups['app_servers'] %}
   {{ hostvars[host]['ansible_eth0']['ipv4']['address'] }}
```

```
{% endfor %}
```

command 바깥에서 변수를 받을 수 있다.

```
$ ansible-playbook release.yml --extra-vars "host=127.0.0.1 user=vagrant"

playbook 파일
---
- hosts: '{{ hosts }}'
  remote_user: '{{ user }}'
```

그리고 조건문을 제공한다. when, false_when 등 다양한 조건문을 지원한다. ansible 노드가 Debian 이면 운영체제를 바로 셧다운하라 지정한다.

```
tasks:
  - name: "shutdown Debian flavored systems"
    command: /sbin/shutdown -t now
    when: ansible_os_family == "Debian"
```

마지막으로 루프(loop) 를 제공한다. wheel 그룹으로 testuser1, testuser2계정을 추가하는 명령어를 루프를 통해서 진행할 수 있다.

```
- name: add several users
  user: name={{ item }} state=present groups=wheel
  with_items:
     - testuser1
     - testuser2
```

여러 파라미터를 두어 group별로 지정을 할 수 있다.

```
- name: add several users
  user: name={{ item.name }} state=present groups={{ item.groups }}
  with_items:
    - { name: 'testuser1', groups: 'wheel' }
    - { name: 'testuser2', groups: 'root' }
```

3.3 inventory 소개

inventory 파일은 리모트 서버에 대한 meta 데이터를 기술 하는 파일이다. ansible에서는 inventory 파일에는 yaml은 적용하지 않았다.

기본 파일은 /etc/ansible/hosts을 읽게 하거나, 따로 inventory 파일을 만들고 따로 옵션을 주어 동작하게 할 수 있다.

다음은 hosts 파일 예제이다. [,] 안에 있는 이름이 그룹 이름이고, 호스트명(host)은 해당 그룹에 속한다. webservers와 dbservers로 나누어서 작업을 할 수 있도록 그룹핑이 되어 있다.

```
[webservers]
foo.example.com
bar.example.com

[dbservers]
one.example.com
two.example.com
three.example.com
```

만약 고정 ip를 가지고 있고 hosts 파일 안에 들어가 있지 않는 서버가 있다면 아래와 같이 설정파일을 만들수 있다. 192.168.1.50 ip를 가진 서버에 22번 ssh 포트로 vagrant 계정으로 접근할 수 있도록 한다. 이는 테스트 환경을 만들 때 매우 유용하다.

```
192.168.1.50 ansible_ssh_user=vagrant ansible_ssh_port=22
```

3.4 명령어 소개

ansible-playbook와 ansible 명령어가 있다.

ansible-playbook 명령어는 playbook yaml 파일을 실행한다.
아래 예제는 /etc/ansible/hosts inventory 파일을 사용하지 않고 따로 정의된 hosts inventory 파일을 기반으로 playbook 으로 정의한 webservers.yml 을 실행시키라는 명령어이다.

```
$ ansible-playbook -i hosts webservers.yml
```

리모트 호스트에 접근하여 특정 명령어를 실행할 수 있다. ansible에서는 ad-hoc task 실행이라 한다. ansible-playbook 이 아닌 ansible 명령어를 이용해야 한다.
아래 예제는 따로 정의된 production inventory 파일에 저장된 모든 호스트에서 대해서 재부팅을 명령하는

```
$ ansible boston -i production -m command -a '/sbin/reboot'
```

만약 에러가 발생하면 verbose 옵션을 주어 자세한 내용을 얻을 수 있다.

```
$ ansible-playbook -i hosts playbook.yml  -vvvv

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

```
GATHERING FACTS ***************************************************************
<192.168.1.50> ESTABLISH CONNECTION FOR USER: vagrant
<192.168.1.50> REMOTE_MODULE setup
<192.168.1.50> EXEC ['ssh', '-C', '-tt', '-vvv', '-o', 'ControlMaster=auto', '-o',
'ControlPersist=60s', '-o', 'ControlPath=/Users/knight/.ansible/cp/ansible-ssh-%h-%p-%r', '-o',
'Port=22', '-o', 'KbdInteractiveAuthentication=no', '-o',
'PreferredAuthentications=gssapi-with-mic,gssapi-keyex,hostbased,publickey', '-o',
'PasswordAuthentication=no', '-o', 'User=vagrant', '-o', 'ConnectTimeout=10', '192.168.1.50',
"/bin/sh -c 'mkdir -p $HOME/.ansible/tmp/ansible-tmp-1406200040.69-215131276335574 &&
chmod a+rx $HOME/.ansible/tmp/ansible-tmp-1406200040.69-215131276335574 && echo
$HOME/.ansible/tmp/ansible-tmp-1406200040.69-215131276335574'"]
fatal: [192.168.1.50] => SSH encountered an unknown error. The output was:
...
```

4. vagrant/ansible 테스트 시나리오

4.1 가상 서버 준비

virtualbox/vagrant와 ansible을 간단하게 연동하는 예제를 진행한다. 리눅스 버전 중 하나인 ubuntu의 12.04 버전이며 64비트 버전인 precise64를 기반으로 예제를 준비했다.
가상 서버(192.168.1.50)가 실행 중인 상태에서 Vagrant 파일, hosts 파일, playbook.yml을 추가하였다.

이해도를 높이기 위해서 vagrant 작업은 두번 진행한다. 먼저 Vagrant 파일은 아래와 같이 생성한다.

```
$ mkdir -p idp-testbox
$ cd idp-testbox
$ 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.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
```

hosts 파일은 다음과 설정한다.

```
192.168.1.50 ansible_ssh_user=vagrant ansible_ssh_port=22
```

playbook.yml 파일은 다음과 같다. /user/local 디렉토리 리스트를 하고 ansible로 그 결과를 로그로 찍도록 한다.

```
$ cat playbook.yml
- hosts: all
  user: vagrant
  tasks:
  - name : test
    action:  command ls -al /usr/local
    register: vagrant
  - debug: var=vagrant.stdout_lines
```

vagrant up 명령을 실행하면. provision 까지 진행하면 성공하는 결과 화면이 보인다. <그림 13>과 같이 virtualbox 리스트 화면을 확인할 수 있다.

```
$ vagrant up
Bringing machine 'default' up with 'virtualbox' provider...
==> default: Importing base box 'precise64'...
==> default: Matching MAC address for NAT networking...
==> default: Setting the name of the VM: idp-testbox_default_1406257067416_3765
==> 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: Warning: Connection timeout. Retrying...
==> 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: 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 40",
        "drwxr-xr-x 10 root root 4096 Sep 14  2012 .",
        "drwxr-xr-x 10 root root 4096 Sep 14  2012 ..",
        "drwxr-xr-x  2 root root 4096 Sep 14  2012 bin",
        "drwxr-xr-x  2 root root 4096 Sep 14  2012 etc",
        "drwxr-xr-x  2 root root 4096 Sep 14  2012 games",
        "drwxr-xr-x  2 root root 4096 Sep 14  2012 include",
        "drwxr-xr-x  3 root root 4096 Sep 14  2012 lib",
        "lrwxrwxrwx  1 root root    9 Sep 14  2012 man -> share/man",
        "drwxr-xr-x  2 root root 4096 Sep 14  2012 sbin",
        "drwxr-xr-x  6 root root 4096 Sep 14  2012 share",
        "drwxr-xr-x  2 root root 4096 Sep 14  2012 src"
    ]
}

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

<그림 13> vagrant up 실행 후 virtualbox 화면
이제는 web 이라는 virtualbox의 gui 이름을 주도록 한다. Vagrant 파일을 아래와 같이 수정한다.
config.vm.provider의 이름을 web으로 변경하도록 한다.

```
$ 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.hostname = "web"

   config.vm.network "private_network", ip: "192.168.1.50"
   config.vm.network :forwarded_port, guest: 9966, host: 3333
```

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

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

end
```

vagrant reload 명령을 내려서 다시 가상 서버를 실행한다. <그림 14>와 같이 virtualbox에 web 이라는 이름을 변경했다. vagrant up 하면서 provision 한 부분때문에 다시 진행하지는 않는다.

```
$ vagrant reload
==> default: Attempting graceful shutdown of VM...
==> default: Setting the name of the VM: web
==> 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 => 3333 (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 already provisioned. Run `vagrant provision` or use the `--provision`
==> default: to force provisioning. Provisioners marked to run always will still run.
```

<그림 14> vagrant 설정의 provider 이름을 변경하고, vagrant reload 실행한 virtualbox 화면

가장 먼저 ssh인증서를 복사해서 로컬 장비에서 가상서버로 접근할 수 있도록 한다.

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

그리고, playbook을 실행하였다. 다음의 provision 결과를 얻을 수 있다.

```
$ ansible-playbook -i hosts playbook.yml
```

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

GATHERING FACTS ***************************************************************
ok: [192.168.1.50]

TASK: [test] ******************************************************************
changed: [192.168.1.50]

TASK: [debug var=vagrant.stdout_lines] ****************************************
ok: [192.168.1.50] => {
    "vagrant.stdout_lines": [
        "total 40",
        "drwxr-xr-x 10 root root 4096 Sep 14  2012 .",
        "drwxr-xr-x 10 root root 4096 Sep 14  2012 ..",
        "drwxr-xr-x  2 root root 4096 Sep 14  2012 bin",
        "drwxr-xr-x  2 root root 4096 Sep 14  2012 etc",
        "drwxr-xr-x  2 root root 4096 Sep 14  2012 games",
        "drwxr-xr-x  2 root root 4096 Sep 14  2012 include",
        "drwxr-xr-x  3 root root 4096 Sep 14  2012 lib",
        "lrwxrwxrwx  1 root root    9 Sep 14  2012 man -> share/man",
        "drwxr-xr-x  2 root root 4096 Sep 14  2012 sbin",
        "drwxr-xr-x  6 root root 4096 Sep 14  2012 share",
        "drwxr-xr-x  2 root root 4096 Sep 14  2012 src"
    ]
}

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

4.2 가상 서버에 소프트웨어/WAS/DB 설치

https://bitbucket.org/gerrytan/jpaminimal (jpaminimal 예제)로 공개한 소스를 가지고 테스트를 진행한다. (자세한 소스 설명은 http://gerrydevstory.com/2014/03/03/creating-a-minimal-spring-mvc-jpa-hibernate-and-mysql-project/을 참조한다.)
이 예제는 tomcat 위에서 동작하는 was는 JPA/Hibernate 를 이용하여 mysql db를 연결하고 간단한 web ui를 보여주는 웹 어플리케이션이다.

setup.yml 파일을 만들어 아래의 예제와 같이 설치할 수 있다. 모든 host inventory에 정의된 host 파일에 vagrant user로 접근 후, sudo 권한으로 git 패키지외 3종류의 소프트웨어를 설치하고 mysql 5.5 서버를 설치한 후 mysql 데몬을 시작하도록 한다.

```
---
```

```
- hosts: all
  user: vagrant
  sudo: True

  tasks:

  - name: common install
    apt: pkg={{ item }} state=latest update_cache=yes
    tags: packages
    with_items:
    - maven
    - ant
    - zip
    - git

  - name: MySQL install
    apt: pkg=mysql-server-5.5 state=present update_cache=yes
    tags: mysql

  - name: MySQL start
    service: name=mysql state=started
    tags: mysql
```

설치 소프트웨어가 많아질 수록 라인 수는 점점 커지고 길어질 수 있다. 따라서 적절히 디렉토리를 나누고 설정 파일을 분할할 필요가 있다. ansible에서 정리한 best practice (http://docs.ansible.com/playbooks_best_practices.html)와 role모델 (http://docs.ansible.com/playbooks_roles.html) 정보를 기반으로 구성하였다.

<그림 15>의 디렉토리 구조와 파일 형태를 최대한 활용했다. 이렇게 사용하면 분할된 role으로 나뉘다 보니 복잡성은 사라지고 유지보수 관리가 편해지는 장점이 있다.

<그림 15> ansible의 role 관점의 디렉토리 구조 정보 (출처 : http://docs.ansible.com/playbooks_roles.html)

저자는 <그림 16>과 같이 디렉토리 구조를 사용했다.

<그림 16> playbooks 예제 디렉토리 구조

설치 기본 파일은 playbooks디렉토리 밑에 setup.yml 로 한다. 아래는 playbooks/setup.yml 파일이다. vagrant user로 sudo 권한으로 common, java, compile, mysql, tomcat role 순서대로 설치하도록 한다.

```
---
- hosts: all
  user: vagrant
  sudo: True
  roles:
    - common
    - java
    - compile
    - mysql
    - tomcat
```

common role은 가상서버에서 사용할 공통 유틸리티를 설치하는 내용을 정의한다. common 디렉토의 main.yml 파일을 찾는다.

common/vars/main.yml 파일이다. 유틸리티 zip, git, maven, ant 를 정의한다.

```
---
packages:
  - zip
  - git
  - maven
  - ant
```

common/tasks/main.yml 파일에서는 packages.yml 파일을 include하고 common/vars/main.yml에서 packages 값을 전달한다.

```
---
- include: packages.yml tags=packages
```

common/tasks/packages.yml 파일은 apt-get으로 zip, git, maven, ant를 설치하라고 한다.

```
---
- name: install packages
  apt: pkg={{ item }} state=latest update_cache=yes
  with_items: packages
```

다음은 oracle의 jdk 7을 설치하도록 하는 java/tasks/main.yml 파일이다. oracle jdk를 바로 다운이 쉽지 않다. apt-get으로 다운받을 수 있도록 준비하고 다운받도록 한다. (이런 방법이 항상 통할 수 없다. 가장 좋은 방법은 미리 oracle jdk를 다운받은 후 가상서버로 복사하는 것일 수 있다.)

```
---
- name: ensuring add-apt-repository is installed
  apt: pkg=python-software-properties state=latest

- name: adding webupd8 ppa for java7 installer
  apt_repository: repo=ppa:webupd8team/java

- name: updating apt cache
  apt: update_cache=yes

- name: installing java7
  shell: echo debconf shared/accepted-oracle-license-v1-1 select true |
         debconf-set-selections &&
         echo debconf shared/accepted-oracle-license-v1-1 seen true |
         debconf-set-selections &&
```

```
apt-get -y install oracle-java7-installer
```

지금까지 공통 유틸리티와 oracle java7 설치를 완료했다. 이제는 소스를 clone 하고 컴파일을 한다.
compile/vars/main.yml 파일이다. jpaminimal 예제의 git url 정보와 clone 되고 난 후 저장할 가상서버 위치를 변수로 지정한다.

```
---
git_clone_url: https://bitbucket.org/gerrytan/jpaminimal.git
git_location: /app/git
```

compile/tasks/main.yml 파일로서, 가상 서버의 clone된 파일을 모두 지우고 clone을 진행한 후, maven 빌드를 진행한 후 디렉토리 permission을 vagrant.vagrant로 수정한다.

```
---
- name : delete git cloned directory
  shell: rm -rf {{ git_location }}

- name : git clone
  git: repo={{ git_clone_url }} dest={{ git_location }} update=no

- name : mvn compile
  command: chdir={{ git_location}} mvn package

- name: change permission
  file: path=/app/git owner=vagrant group=vagrant state=directory recurse=yes
```

다음은 mysql 설치를 진행한다.
mysql/vars/main.yml 파일이다. utf8로 설정했고 mysqld 설정으로 bind_adress를 지정한다.

```
---

utf8: true
mysqld:
  bind_address: 127.0.0.1
```

mysql/tasks/main.yml 파일이다. /usr/bin/mysqld 파일이 있으면 mysql을 설치하지 않도록 했다. 그리고 templates/mysqld.conf.j2 (jinja2 파일)을 가성 서버에 mysqld.conf 파일로 복사하도록 하였다. 이렇게 하여 변수나 이벤트시 설정 파일을 동적으로 변경시킬 수 있도록 하였다. (예, utf8 변수 선언) was에서 사용할 논리DB의 table을 매번 삭제하고 재생성하도록 한 후 mysqld 데몬을 재실행하도록 하였다.

```
---

  - stat: path=/usr/bin/mysqld
```

```
    register: mysqld_exist

  - name: mysql 5.5 installation
    apt: pkg=mysql-server-5.5 state=present update_cache=yes
    when: mysqld_exist.stat.exists == false

  - name: copy conf.d/mysqld.cnf
    action: template src=mysqld.cnf.j2 dest=/etc/mysql/conf.d/mysqld.cnf
    notify: restart mysql

  - shell: /usr/bin/mysql -u root --password= -e "DROP TABLE IF EXISTS hello ; CREATE
DATABASE IF NOT EXISTS hello; USE hello; CREATE TABLE person (id INT PRIMARY
KEY AUTO_INCREMENT,name VARCHAR(50));"
    ignore_errors: True
    when: mysqld_exist.stat.exists == true

  - name: start mysql
    service: name=mysql state=restarted enabled=yes
    sudo: True
```

mysql/template/mysql.conf.j2 파일이다. jinja2의 스크립트가 사용되었다. utf8 변수가 정의되어 있다면 언어 설정에 utf8을 설정 값을 저장하도록 한다.

```
[mysqld]
{% if utf8 %}
character-set-server = utf8
collation-server = utf8_unicode_ci
{% endif %}
```

mysql/handlers/handlers.yml 파일은 tasks.yml 에서 정의한 copy conf.d/mysqld.cnf 이름을 가진 task가 완료되면 handler로 notify 할 수 있다. mysql 데몬을 재시작하도록 한다.

```
---
  - name: restart mysql
    action: service name=mysql state=restarted enabled=yes
```

마지막은 tomcat 설치 부분이다. tomcat/vars/main.yml 에는 가상서버에 tomcat이 저장할 위치를 지정한다.

```
---
tomcat_dir: /app/tomcat
```

tomcat/tasks/main.yml 파일은 톰캣 설치에 대한 정보를 언급하고 있다. 웹에서 다운받아서 tomcat_dir에 저장하도록 한 후, compile된 war 파일을 webapps디렉토리에 복사한다. 그리고 tomcat 실행 파일(tomcat-initscript.sh)과 tomcat 설정 파일 (server.xml)을 가상서버에 잘 복사하도록 한다. 그리고 service 데몬으로 tomcat을 실행한다. 마지막으로는 톰캣이 제대로 실행되어 8080 포트가 alive한지 wait한다. 만약 정상이면 play를 종료한다.

```
---
- name: check if tomcat path exists
  stat: path=/app/src

- name: if tomcat dir exists
  stat: path=/app/tomcat

- name: download Tomcat
  get_url:
url=http://archive.apache.org/dist/tomcat/tomcat-7/v7.0.54/bin/apache-tomcat-7.0.54.tar.gz
dest=/app/src/apache-tomcat-7.0.54.tar.gz

- name: extract archive tomcat
  command: chdir=/app /bin/tar xvf /app/src/apache-tomcat-7.0.54.tar.gz -C /app/
creates=/app/apache-tomcat-7.0.54

- name: symlink install directory
  file: src=/app/apache-tomcat-7.0.54 path=/app/tomcat state=link

- name: change ownership of Tomcat installation
  file: path=/app/tomcat owner=vagrant group=vagrant state=directory recurse=yes

- name: configure Tomcat server
  template: src=server.xml dest={{ tomcat_dir }}/conf/
  notify: restart tomcat

- name: copy war to webapps
  command: cp {{ git_location }}/target/hello-1.0.war {{ tomcat_dir }}/webapps

- name: install Tomcat init script
  copy: src=tomcat-initscript.sh dest=/etc/init.d/tomcat mode=0755

- name: start Tomcat
  service: name=tomcat state=restarted enabled=yes
  register: result

- debug: var=result
```

```
- name: wait for tomcat to start
  wait_for: port=8080
```

tomcat/handlers/handlers.yml 은 다음과 같다. tomcat task에서 정의한 설정 파일 복사 task가 완료되면 아래 이벤트가 notify 되어 재시작을 하도록 한다 .

```
---
- name: restart tomcat
  service: name=tomcat state=restarted
```

tomcat/templates/server.xml 은 사용자 정의된 형태로 저장한다. 데모에서는 특별히 바꾼 것은 없다. 그리고 service 데몬으로 쓰기 위해서 tomcat/files/tomcat-initscript.sh 파일은 다음과 같다. 이 파일은 https://gist.github.com/valotas/1000094 파일을 그대로 활용했다. tomcat 구동 스크립트는 init.d 기준으로 하지 않으나, tomcat을 service 명령어를 통해서 사용할 수 있는 좋은 예제라서 참고하였다.

```
#!/bin/bash
#
# chkconfig: 345 99 28
# description: Starts/Stops Apache Tomcat
#
# Tomcat 7 start/stop/status script
# Forked from: https://gist.github.com/valotas/1000094
# @author: Miglen Evlogiev <bash@miglen.com>
#
# Release updates:
# Updated method for gathering pid of the current proccess
# Added usage of CATALINA_BASE
# Added coloring and additional status
# Added check for existence of the tomcat user
#


#CATALINA_HOME is the location of the bin files of Tomcat
export CATALINA_HOME=/app/tomcat

#CATALINA_BASE is the location of the configuration files of this instance of Tomcat
export CATALINA_BASE=/app/tomcat

#TOMCAT_USER is the default user of tomcat
export TOMCAT_USER=vagrant

#TOMCAT_USAGE is the message if this script is called without any options
TOMCAT_USAGE="Usage: $0
{\e[00;32mstart\e[00m|\e[00;31mstop\e[00m|\e[00;32mstatus\e[00m|\e[00;31mrestart\e[00m}"
```

```
export JAVA_OPTS="-Dname=tomcat"

#SHUTDOWN_WAIT is wait time in seconds for java proccess to stop
SHUTDOWN_WAIT=20

tomcat_pid() {
        echo `ps -fe | grep $CATALINA_BASE | grep -v grep | tr -s " "|cut -d" " -f2`
}

start() {
  pid=$(tomcat_pid)
  if [ -n "$pid" ]
  then
    echo -e "\e[00;31mTomcat is already running (pid: $pid)\e[00m"
  else
    # Start tomcat
    echo -e "\e[00;32mStarting tomcat\e[00m"
    #ulimit -n 100000
    #umask 007
    #/bin/su -p -s /bin/sh tomcat
        if [ `user_exists $TOMCAT_USER` = "1" ]
        then
                su $TOMCAT_USER -c $CATALINA_HOME/bin/startup.sh
        else
                sh $CATALINA_HOME/bin/startup.sh
        fi
        status
  fi
  return 0
}

status(){
          pid=$(tomcat_pid)
          if [ -n "$pid" ]; then echo -e "\e[00;32mTomcat is running with pid: $pid\e[00m"
          else echo -e "\e[00;31mTomcat is not running\e[00m"
          fi
}

stop() {
  pid=$(tomcat_pid)
  if [ -n "$pid" ]
  then
    echo -e "\e[00;31mStoping Tomcat\e[00m"
    #/bin/su -p -s /bin/sh tomcat
        sh $CATALINA_HOME/bin/shutdown.sh
```

```
    let kwait=$SHUTDOWN_WAIT
    count=0;
    until [ `ps -p $pid | grep -c $pid` = '0' ] || [ $count -gt $kwait ]
    do
      echo -n -e "\n\e[00;31mwaiting for processes to exit\e[00m";
      sleep 1
      let count=$count+1;
    done

    if [ $count -gt $kwait ]; then
      echo -n -e "\n\e[00;31mkilling processes which didn't stop after $SHUTDOWN_WAIT
seconds\e[00m"
      kill -9 $pid
    fi
  else
    echo -e "\e[00;31mTomcat is not running\e[00m"
  fi

  return 0
}

user_exists(){
        if id -u $1 >/dev/null 2>&1; then
        echo "1"
        else
                echo "0"
        fi
}

case $1 in

        start)
          start
        ;;

        stop)
          stop
        ;;

        restart)
          stop
          start
        ;;

        status)
                status
```

```
        ;;

        *)
                echo -e $TOMCAT_USAGE
        ;;
esac
exit 0
```

playbook 결과는 로그를 확인하면 된다. 특별히 fail난 부분이 없다.
이제 가상서버의 was 서버에 접근한다. host inventory 파일에서 정한 대로 이용한다.
http://192.168.1.50:8080/hello-1.0/ 을 웹으로 접속하면 <그림 17>화면과 같이 정상적으로 작동하는지 확인이 가능하다.

<그림 17> jpaminimal 예제의 웹 UI

jpaminimal 예제는 https://github.com/knight1128/vagrant-ansible-testbox 를 참조하면 된다.

4.3 실제 개발 환경

4.2의 jpaminimal 예제의 경우는 워낙 간단해서 가상 환경이 필요하지 않을 수 있다.

그러나 저자의 경우는 화이트 박스 테스트 케이스가 없는 Legacy 환경에서는 반드시 필요했다. 일반 유틸리티에 java, mysql, tomcat, nodejs 소프트웨어를 설치하고 java was 서버 한대, java 프로세스 6대, node js프로세스 6대, mysql 1 대를 설치도 4.2에서 언급한 방식으로 쉽게 개발 환경을 구성할 수 있다. 이 개발환경을 그대로 개발자끼리 공유해서 환경 구성 시간을 최대한 줄일 수 있었다. (설치 등등)

또한 vagrant/ansible 가상 서버 테스트 환경을 바탕으로 API 테스트를 진행할 수 있다. jmeter나 soapui를 활용할 수 있다. <그림 18>의 jmeter 를 구성하여 테스트 환경을 구축할 수 있다.

<그림 18> jmeter 개발 환경

만약 UI가 있는 경우라면, acceptance 테스트 또는 funtional 테스트가 가능하다. HtmlUnit, Selenium과 같은 툴로 UI 를 테스트하며 명확한 테스트를 진행할 수 있을 것이다.

그리고, QA나 연동 개발이 필요한 개발자에게는 가상 서버 테스트 환경 기반의 UI admin 을 따로 둘 수 있다. 가상 서버로 api 호출과 응답을 체크하는 admin 서버를 만들 수 있다. 좀 더 고급스럽게 개발한다면 facebook의 개발자 센터와 같은 형태도 가능하다.

5. 결론

지금까지 vagrant/ansible을 활용한 개발자 환경을 구성해보았다. 가상 머신을 통제하는 vagrant와 환경 구성 툴인 ansible을 이용한다면 개발자간, 또는 개발자-QA, 개발자-관리자간 원활한 소통을 이어줄 수 있을 것이다.

ansible은 vagrant 뿐 아니라 docker와도 연동 가능하기 때문에 다른 개발 환경을 구축할 수 있다. 그리고 ansible의 고급 기능은 production 레벨에서 서버 배포 환경 및 서버 모니터링 연동도 쓰일 수 있는 훌륭한 상용 배포 툴로 사용가능하다. 최근에 트렌디하게 사용하고 있는 Continuous Deployment를 구현할 수 있다.

그리고 ansible을 push 방식으로 쓰기로 하지만, pull 방식으로 쓸 수 있다. 즉 상용 서버에서 git repository로 cronjob으로 소스 checkout을 받아 서버 설정의 현행화를 할 수 있다.

ansible은 계속 발전되고 있기 때문에 앞으로 배워두면 많은 도움이 될 것이라 생각된다.

참고

1. http://redmonk.com/sogrady/2013/12/06/configuration-management-2013/
2. http://en.wikipedia.org
3. http://ko.wikipedia.org

부록

1. puppet 소개
puppet은 ruby언어로 개발되었고 2005년 puppet labs(http://puppetlabs.com/)에서 발표하였다. 초기 버전은 GPL 라이선스였으나, 2.7.0 버전 후에는 Apache 라이선스 2.0 으로 변경했다.
대표적 적용 회사로서 twitter, redhat, salesforce, starbucks, at&t 에서 채택되어 사용 중이다. <그림 19>은 puppet 홈페이지이다.

<그림 19> puppet 홈페이지 (출처 : http://puppetlabs.com/)

enterprise 버전은 유료이나, 오픈 소스 버전(https://github.com/puppetlabs/puppet)은 무료로 사용할 수 있다. 제약 조건은 <그림 20>를 참조한다. 중요한 관리 부분은 puppet 오픈 소스 부분에서 사용 가능하다. 그러나 지원은 받을 수 없다.

<그림 20> puppet의 상용 버전과 오픈 소스 버전 차이 (출처 :http://puppetlabs.com/puppet/enterprise-vs-open-source)

puppet은 server와 agent 간의 통신 체계를 근간으로 되어 있다. 따라서 server, agent 각각 설치한다. 그리고 통신은 SSL 로 통신하도록 되어 있다. puppet server에는 설정 내용을 담고 있고 puppet agent를 통해서 provision된다. puppet agent가 설치된 서버를 node라 불린다.

puppet agent가 server로 계속 주기적으로 요청하는 클라이언트-서버 아키텍처를 지원하기 때문에 puppet agent가 대량의 서버에 설치될 경우 puppet 서버에 부하를 발생할 수 있는 단점이 있다. 그리고 puppet server에서 특정 서버만 테스트하려면 운영의 노하우가 필요하다.

<그림 21>은 puppet 아키텍처를 표현한 그림이다.

<그림 21> puppet 아키텍처 (출처 : http://www.slashroot.in)

설정 파일은 독자적인 DSL 기반(manifest)을 정의하여 시스템 관리를 한다. 그리고 다양한 OS 환경에서도 추상화된 레이어를 제공하여 단일 manifest에서 실행할 수 있도록 하였다. 따라서 리눅스 버전마다 다른 yum, apt-get 같은 설치 방법을 최소화하였다. 리소스 의존성을 가지게 하였고 LDAP을 지원한다.

puppet에 대해서 궁금한 독자는 아래 싸이트를 참조하면 된다.
http://docs.puppetlabs.com/
http://puppetcookbook.com/

2. chef 소개
chef는 puppet과 마찬가지로 ruby언어로 개발되었다. Opscode 사에 의해서 2009년 출시되었다. 이후 Chef라는 회사명으로 변경되었다. <그림 22>은 chef 홈페이지이다. chef는 facebook, splunk 등과 같은 쟁쟁한 회사에서 많이 사용하고 있다. 최근에는 Amazon의 aws 서비스에서 적용되어 chef를 사용할 수 있다.

<그림 22> chef 홈페이지 (출처 : http://www.getchef.com/)

ruby 언어 기반으로 DSL 방식(recipes, cookbooks) 으로 시스템 설정을 정의한다. ruby 언어를 잘 아는 사람에게는 좋은 장점이 된다. chef는 cookbook을 정의한다. 즉 cookbook은 요리책으로서 형상 관리할 대상을 어떻게 요리할 지 작성한다. cookbook 안에 recipe가 존재하며, recipe는 하나의 인프라를 관리한다. 이런 모델링의 내용이 chef의 가장 중요한 부분이다. 리소스와 서버간의 연동관계를 잘 정의되어 있는 장점이 있는 반면, 배우는 입장에서는 학습 시간이 많이 필요하다는 단점이 있다. 자세한 모델링에 대한 내용은 <그림 23>을 참조한다.

<그림 23> chef 사용 모델 (출처 : http://magazine.rubyist.net/?0035-ChefInDECOLOG)

puppet과 마찬가지로 chef도 클라이언트-서버 아키텍처 기반으로 되어 있다. <그림 24>에서 보듯이 특이한 점은 chef server를 knife로 통제하는 chef development machine (chef workstation) 이 존재한다. chef node는 배포될 서버들이다.

<그림 24> chef 간략 아키텍처 (출처 : http://tasting.solita.fi/chef)

chef 서버는 다른 시스템과 연동을 위한 rest api를 제공하고 있다. 노드 및 cookbook를 각 node에 배포하며 통제하는 역할을 한다. chef indexer라는 시스템이 있는데, chef환경에서 검색이 필요한 작업을 책임진다. <그림 25>과 같은 구성 아키텍처를 통해 확인할 수 있다.
특히 CouchDB를 설치해서 chef 서버와 연동하도록 해야 하며 또한 chef indexer안에는 rabbitmq가 있어 이를 미리 설치해야 한다. 통신은 ruby 의 message 연동 방식인 STOMP(http://stomp.github.io/) 를 사용하도록 되어 있다.

<그림 25> chef 구성 아키텍처
(출처 : https://www.ibm.com/developerworks/community/blogs/9e635b49-09e9-4c23-8999-a4d461aeace2/entry/215?lang=en)

puppet과 마찬가지로 chef는 open source(https://github.com/opscode/chef) 와 enterprise 버전(http://www.getchef.com/chef/)으로 나뉜다. 기본적인 기능은 open source만으로 충분하다. <그림 26>는 오픈소스 버전과 상용 버전의 차이를 설명한다.

<그림 26> chef의 puppet의 상용 버전과 오픈 소스 버전 차이 (출처 : http://www.getchef.com/chef/)

참고자료는 여기서 확인한다.
http://docs.getchef.com/
https://wiki.opscode.com/display/chef/Home

3. docker 소개
요즘에는 vagrant 대신 가상 머신으로 docker를 쓰는 경우가 있다. docker (가상서버는 아니지만 경량화된 컨테이너, 사용자는 마치 가상서버처럼 느낌)에 ansible를 붙여서 사용 가능하다.

<그림 27>과 같이 docker는 hypervisor 위에 guest os가 올라간 형태의 가상머신이 아닌 docker enginer위에 binary container 이다. 따라서 가상머신과 기능이 같고 속도가 훨신 빠르다.

<그림 27> (출처 : https://www.docker.com)

이런 장점 때문에 최근에 docker가 인기를 얻고 있으며 다양한 docker linux image들이 배포되고 있다. 최근에 빠른 관심이 보이는 부분이이라 관심을 가져볼 만 하다. ansible과 docker 연동 사례들이 늘고 있으니 참조하면 좋을 것 같다.

참고자료는 다음과 같다.
http://docs.ansible.com/docker_module.html
http://thenewstack.io/are-docker-users-migrating-to-ansible-and-away-from-puppet-and-chef/
https://github.com/cove/docker-ansible