Ansible 사용 설명서

원본 의 내용을 한글로 번역해 봅니다.

> *역자노트:*
>
> - <원본> 이라고 표시된 부분은 해당 원본 링크를 그대로 연결한 것입니다.
> - 전체 문서 중 Quick Guid 정도는 완성 (전체 문서의 20%정도) 되었다고 보면 됩니다.

Ansible 소개

빨리 익히는 Ansible 동영상 <원본>

플레이북

플레이북:특정 주제 <원본>

모듈

배포 관련 상세 가이드 <원본>

Docker 이용하기

개발 정보

Ansible Galaxy <원본>

테스팅 전략 <원본>

YAML 문법 <원본>

Asible 소개

설치

```
$ pip install ansible
```

설치 결과는 ansible (2.0.1.0) 이고 python 2.7 및 3.5 모두 잘 설치되었습니다.

설치하기 참조

만약 최신버전 (2.2) 으로 설치하려면

```
$ pip install git+git://github.com/ansible/ansible.git@devel
```

라고 개발자 버전을 설치합니다. (현재 문서는 대부분 2.2의 내용을 언급하므로 위와 같이 설치하시기를 권고해 드립니다.)

> **역자노트:** 위에 내용은 시스템 파이썬이 아니라 VirtualEnv 환경에서의 설치입니다. 시스템 파이썬에 설치할 경우에는 `sudo pip ...` 라고 명령을 수행하십시오.

시작하기

원격 연결

- 원격 연결은 SSH key로 연결하는 것이 일반적
- 만약 암호를 물어보는 것이 필요하면 `--ask-pass` 옵션 사용
- 만약 sudo를 이용한 암호확인 이라면 `--ask-become-pass` 이용

> **노트** : 1.3 버전 이후부터는 파이썬의 OpenSSH 를 이용하여 하였고 1.2 부터는 `paramiko` 라는 파이썬 SSH 모듈을 이용

첫번째 명령

`/etc/ansible/hosts` 파일에 ansible 관리대상 호스트를 넣어줍니다.

```
192.168.2.2
192.168.2.14
192.168.2.141
```

이 파일을 `Inventory` 라고 하며 Inventory 상세 설명 에 잘 나와 있습니다.

> **노트** : 이미 SSH 키로 위의 관리 대상(현재는 그냥 호스트)을 연결하였다고 가정함. `--private-key` 옵션을 이용하여 연결할 수도 있음.

이제,

```
$ ansible all -m ping -u future
```

명령을 내려 볼 수 있습니다.

이것은 모든 Inventory (위에서 지정한 3개의 IP) 에 대해서 ping 이라는 모듈을 실행하는데 (`-m ping`) future 라는 사용자로 접속하는 것입니다. (`-u future`) 여기서 -u 옵션은 ping 이라는 모듈의 옵션입니다.

현재는 그 결과가

```
192.168.2.2 | UNREACHABLE! => {
    "changed": false,
    "msg": "SSH encountered an unknown error during the connection. We recommend you re-run the c
    "unreachable": true
}
192.168.2.14 | UNREACHABLE! => {
    "changed": false,
    "msg": "SSH encountered an unknown error during the connection. We recommend you re-run the c
    "unreachable": true
}
192.168.2.141 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
```

위와 같이 연결되는 것도 있고 안되는 것도 있습니다.

시스템의 ping 명령이 아닌 ssh 연결 위에 접속되는 것으로 보이는 군요. 세세히 지정하는 것은 추후에 살펴보도록 하겠습니다. 여기서는 SSH 키 교환으로 접속이 되는 것만 응답을 하는 것을 알 수 있습니다.

```
$ ansible all -a "/bin/echo hello" -u future
```

라고 하여 특정 명령을 수행하는 것도 가능한데 그 결과 역시

```
192.168.2.2 | UNREACHABLE! => {
    "changed": false,
    "msg": "SSH encountered an unknown error during the connection. We recommend you re-run the c
    "unreachable": true
}
192.168.2.14 | UNREACHABLE! => {
    "changed": false,
    "msg": "SSH encountered an unknown error during the connection. We recommend you re-run the c
    "unreachable": true
}
192.168.2.141 | SUCCESS | rc=0 >>
hello
```

라고 하여 192.168.2.141 만 응답을 하는 것을 알 수 있습니다.

관리 대상 목록 (Inventory)

Ansible 에서는 관리 대상 목록을 Inventory 라고 합니다. 이 대상 목록은 디폴트로 `/etc/ansible/hosts` 에 정의되어 있습니다. 하지만 `-i <path>` 옵션을 이용하여 명령행에서 지정할 수도 있습니다. 이것은 고정된 것 뿐만 아니라 동적 관리 대상 목록 (Dynamic Inventory)을 이용할 수 있습니다.

호스트 또는 그룹

`/etc/ansible/hosts` 파일 형식은 INI 파일 형식 처럼 사용할 수 있는데,

```
mail.example.com

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

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

와 같이 지정할 수 있습니다.

대괄호로 묶인 `[webservers]` 나 `[dbservers]` 는 그룹을 의미합니다. 디폴트로 모두 SSH 통신을 통하여 접속된다고 가정합니다. 만약 SSH 연결 포트가 디폴트인 22번이 아닌 경우 다음과 같이,

```
badwolf.example.com:5309
```

포트를 지정할 수 있습니다.

만약 alias 와 같이 지정하고 싶다면,

```
jumper ansible_port=5555 ansible_host=192.168.1.50
```

와 같이 지정하면 됩니다. 이것은 jumper를 접속하게 되면 192.168.1.50:5555 로 SSH 접속을 하게 됩니다.

다음과 같이 확장도 가능합니다.

```
[webservers]
www[01:50].example.com
[databases]
db-[a:f].example.com
```

`www[01:50].example.com` 은 `www01.example.com` 부터 `www50.example.com` 까지의 50개의 관리 대상을 의미하며, `db-[a:f].example.com` 은 `db-a.example.com` 부터 `db-f.example.com` 까지의 6개의 관리 대상을 의미합니다.

> **노트** : 만약 2.0 이전 버전이라면 `ansible_user` 대신 `ansible_ssh_user` , `ansible_host` 대신 `ansible_ssh_host` 또는 `ansible_port` 대신 `ansible_ssh_port` 라고 표현됩니다.

다음과 같이 각 대상 별 연결 방법을 달리 할 수 있습니다.

```
[targets]
localhost               ansible_connection=local
other1.example.com      ansible_connection=ssh        ansible_user=mpdehaan
other2.example.com      ansible_connection=ssh        ansible_user=mdehaan
```

호스트 변수

```
[atlanta]
host1 http_port=80 maxRequestsPerChild=808
host2 http_port=303 maxRequestsPerChild=909
```

`http_port`, `maxRequestsPerChild` 와 같이 호스트 변수를 지정하여 나중에 설명할 플레이북 에서 사용할 수 있습니다.

그룹 변수

```
[atlanta]
host1
host2

[atlanta:vars]
ntp_server=ntp.atlanta.example.com
proxy=proxy.atlanta.example.com
```

`atlanta` 그룹에서 공통으로 사용할 변수 **ntp_server**와 **proxy**를 지정하는 방법입니다.

그룹의 그룹, 그룹 변수

그룹 이름 다음에 `:children` 을 붙여 그룹의 하위 그룹을 표현할 수 있습니다. 또는 위에서 처럼 `:vars` 를 붙여 그룹 변수를 포현합니다. 약간 더 복잡하게 다음과 같이 표현할 수 있습니다.

```
[atlanta]
host1
host2

[raleigh]
host2
host3

[southeast:children]
atlanta
raleigh

[southeast:vars]
some_server=foo.southeast.example.com
halon_system_timeout=30
self_destruct_countdown=60
escape_pods=2

[usa:children]
southeast
northeast
southwest
northwest
```

특정 호스트 또는 특정 그룹의 데이터 분리

이전과 같이 특정 관리 대상 목록 파일에 변수 등의 데이터를 지정할 수도 있지만 이를 별도의 파일로 분리할 수 있습니다. 따로 떼어 놓을 변수 파일은 YAML 파일 형식입니다. (YAML 문법 참조 역자주: `키: 값` 을 지정시 꼭 콜론 다음에 공백을 하나 두도록 합니다)

```
/etc/ansible/hosts
```

위와 같이 관리 목록 대상 파일이 있고 `/etc/ansible/group_vars/그룹명` 의 파일을 지정하면 해당 그룹 변수를 지정합니다. 마찬가지로 `/etc/ansible/host_vars/호스트명` 의 파일을 지정하면 해당 호스트 변수를 지정합니다.

```
/etc/ansible/group_vars/raleigh # can optionally end in '.yml', '.yaml', or '.json'
/etc/ansible/group_vars/webservers
/etc/ansible/host_vars/foosball
```

예를 들어 위의 `/etc/ansible/group_vars/raleigh` 파일에

```
---
ntp_server: acme.example.org
database_server: storage.example.org
```

와 같이 지정하여 **ntp_server**와 **database_server** 그룹 변수를 지정하였습니다.

만약 위와 같은 경로에 해당 파일이 없는 경우 다음과 같은 선택적 대안도 존재할 수 있습니다. 예를 들어 `releigh` 그룹에 **db_settings** 및 **cluster_settings** 라는 변수가 이용된다면 다음과 같은 파일에 YAML 형식으로 해당 내용을 넣을 수 있습니다.

```
/etc/ansible/group_vars/raleigh/db_settings
/etc/ansible/group_vars/raleigh/cluster_settings
```

> **노트** : 이것은 특히 특정 변수 값이 매우 크거나 할 때 유용합니다. 버전 1.4 이후에서 지원합니다.

접속 관련 관리 대상 목록 패러미터

ansible 의 관리 대상 목록에 있는 호스트별 접속 정보를 위한 미리 설정된 패러미터는 다음과 같은 것들이 있습니다.

호스트 연결:

ansible_connection

호스트에 연결될 접속 형태를 의미합니다. 이것은 Ansible 연결 플러그인의 이름입니다. SSH 프로토콜 형식은 `smart` , `ssh` , `paramiko` 중에 하나가 될 수 있고 디폴트는 `smart` 입니다. SSH 연결이 아닌경우에는 아래에 별도로 기술됩니다.

> **노트** : 만약 2.0 이전 버전이라면 `ansible_user` 대신 `ansible_ssh_user` , `ansible_host` 대신 `ansible_ssh_host` 또는 `ansible_port` 대신 `ansible_ssh_port` 라고 표현됩니다.

SSH 연결:

ansible_host

만약 주어진 이름과 달리 접속할 호스트 명을 별도로 지정한다면 이 패러미터에 기술합니다.

ansible_port

만약 SSH 디폴트 포트인 22번이 아니라면 이 패러미터에 지정해 줍니다.

ansible_user

접속할 SSH 사용자 이름을 지정합니다.

ansible_ssh_pass

만약 SSH 접속시 암호를 묻는다면 이 암호를 지정해 줍니다.

> **노트** : 이것은 보안에 문제가 있을 수 있기 때문에 `--ask-pass` 를 이용하거나 SSH 키를 이용하기를 권고합니다.

ansible_ssh_private_key_file

ssh 연결에 사용할 개인키 파일을 지정합니다. 보통 `/home/user/.ssh/id_rsa` 와 같이 지정합니다.

ansible_ssh_common_args

이 패러미터는 sftp, scp 또는 ssh 명령에 추가되는 패러미터를 지정할 수 있습니다. 특정 호스트나 그룹에 ***ProxyCommand*** 를 설정하는 데 유용합니다.

ansible_sftp_extra_args

이 패러미터는 sftp 명령에 추가되는 패러미터를 지정할 수 있습니다.

ansible_scp_extra_args

이 패러미터는 scp 명령에 추가되는 패러미터를 지정할 수 있습니다.

ansible_ssh_extra_args

이 패러미터는 ssh 명령에 추가되는 패러미터를 지정할 수 있습니다.

ansible_ssh_pipelining

SSH 파이프라인을 이용할 것이가를 나타냅니다. 이것은 `ansible.cfg` 의 글로벌 설정에 있는 ***pipelining*** 설정에 우선합니다.

권한상승: 권한 상승 문서 참조

ansible_become

ansible_sudo 또는 ***ansible_su*** 명령과 동일합니다.

ansible_become_method

권한 상승 방법을 지정합니다.

ansible_become_user

ansiblesudouser 또는 ***ansiblesuuser*** 와 동일합니다.

> **노트** : 이것은 보안에 문제가 있을 수 있기 때문에 `--ask-become-pass` 를 이용하거나 SSH 키를 이용하기를 권고합니다.

원격 호스트 환경 관련 패러미터:

ansible_shell_type

타겟 시스템의 쉘을 지정합니다. ***ansibleshellexecutable*** 을 본쉘 호환이 되지 않은 것을 특별히 지정하지 않은한 이 패러미터는 사용하면 안됩니다. 디폴트로 sh 스타일의 문법을 이용한 명령 형식을 디폴트로 사용합니다. 만약 이 패러미터를 `csh` 또는 `fish` 로 지정하면 해당 쉘 명령 형식을 따릅니다.

ansible_python_interpreter

타겟 호스트의 파이썬 인터프리터 경로를 지정합니다.

ansible_*_interpreter

`ansible_python_interpreter` 와 유사하게 ruby, perl 등의 인터프리터 경로를 지정합니다. 이 경우 파이썬이 아닌 해당 인터프리터로 구성된 모듈을 구동시킵니다.

버전 2.1에 추가된 패러미터

ansible_shell_executable

이 패러미터는 `ansible.cfg` 글로벌 설정파일에 있는 `executable` 의 **/bin/sh** 대신 타겟 머신에서 제어되는 명령의 쉘을 지정합니다. **/bin/sh** 에서 지원되지 않는 경우에만 사용하기 바랍니다.

호스트 파일 사용 예

```
some_host         ansible_port=2222     ansible_user=manager
aws_host          ansible_ssh_private_key_file=/home/example/.ssh/aws.pem
freebsd_host      ansible_python_interpreter=/usr/local/bin/python
ruby_module_host  ansible_ruby_interpreter=/usr/bin/ruby.1.9.3
```

SSH 연결이 아닌 경우

Ansible은 SSH 연결 위에서 플레이북을 실행시키도록 되어 있지만 이 방식 이외의 연결도 가능합니다. ***ansible_connection=<connector>*** 패러미터에 의해 변경할 수 있습니다. 다음과 같은 비 SSH 연결이 가능합니다.

local

ansible을 수행하는 머신 자체에서 실행되는 경우입니다.

docker

이 연결은 로컬에 있는 docker 클라이언트를 이용하여 다커 컨테이너에 직접 playbook을 실행시킵니다. 이 커넥터에는 아래와 같은 패러미터가 지원됩니다.

ansible_host

연결할 다커 컨테이너의 이름을 지정합니다.

ansible_user

컨테이너에서 운영될 사용자를 지정합니다. 물론 해당 사용자는 컨테이너에서 미리 존재해야 합니다.

ansible_become

만약 `become_user` 가 `true` 로 설정된다면 컨테이너에서 동작합니다.

ansible_docker_extra_args

다커가 인식할 수 있는 추가적인 아규먼트를 문자열로 표현한 것입니다. 이 패러미터는 주로 원격 다커 데몬을 다루는 설정에 이용될 수 있습니다.

다음은 컨테이너를 배포하는 것에 대한 예제입니다.

```yaml
- name: create jenkins container
  docker:
    name: my_jenkins
    image: jenkins

- name: add container to inventory
  add_host:
    name: my_jenkins
    ansible_connection: docker
    ansible_docker_extra_args: "--tlsverify --tlscacert=/path/to/ca.pem --tlscert=/path/to/client
    ansible_user: jenkins
  changed_when: false

- name: create directory for ssh keys
  delegate_to: my_jenkins
  file:
    path: "/var/jenkins_home/.ssh/jupiter"
    state: directory
```

동적 Invntory

설정관리 시스템의 사용자는 종종 설정 관리 목록 (Inventory)을 다른 소스트웨어 시스템에 보관하기를 원할 수 있습니다. 클라우드 제공자, LDAP, Cobbler 또는 CMDB 소프트웨어 등과 같은 곳에 저장할 경우도 있습니다.

Ansible은 이런 모든 외부 저장소를 쉽게 연동할 수 있습니다. 이미 `contrib/inventory` 디렉터리에 이미 되어 있는 것도 있으나 아래에서는 EC2/Eucalyptus, Rackspace Cloud, 및 OpenStack 등과 연동되는 것을 확인해 보도록 하겠습니다.

Ansible Tower 또한 Inventory를 관리하는 DB를 가지고 있으며 REST API를 제공하고 웹으로 관리할 수 있도록 되어 있습니다. Inventory 편집기를 별도 가지고 있습니다. 설정관리에 대한 추적도 가능합니다. 플레이북을 실행하면서 어디에서 오류가 발생했는지 추적도 가능합니다.

예: Cobbler 외부 관리 대상 목록 스크립트

`/etc/ansible/cobbler.ini` 파일에 다음과 같이 지정할 수 있습니다.

```
[cobbler]

# Set Cobbler's hostname or IP address
host = http://127.0.0.1/cobbler_api

# API calls to Cobbler can be slow. For this reason, we cache the results of an API
# call. Set this to the path you want cache files to be written to. Two files
# will be written to this directory:
#   - ansible-cobbler.cache
#   - ansible-cobbler.index

cache_path = /tmp

# The number of seconds a cache file is considered valid. After this many
# seconds, a new API call will be made, and the cache file will be updated.

cache_max_age = 900
```

/etc/ansible/cobbler.py 스크립트를 바로 실행할 수 있습니다.

다음과 같은 식으로 실행될 수 있습니다.

```
cobbler profile add --name=webserver --distro=CentOS6-x86_64
cobbler profile edit --name=webserver --mgmt-classes="webserver" --ksmeta="a=2 b=3"
cobbler system edit --name=foo --dns-name="foo.example.com" --mgmt-classes="atlanta" --ksmeta="c=
cobbler system edit --name=bar --dns-name="bar.example.com" --mgmt-classes="atlanta" --ksmeta="c=
```

예: AWS EC2 외부 관리 대상 목록 스크립트

EC2 external inventory scritp 참조

> TODO : 자세한 내용은 원본 문서 페이지를 참조하시기 바랍니다.

패턴

Ansible에서 패턴이라는 것은 관리하기 위한 호스트를 결정하는 것입니다. 이것은 통신할 호스트들을 의미하지만 플레이북 관점에서는 실제 무언가 명령을 실행하고 로직을 처리하는 플레이하는 대상의 호스트(장비)를 의미합니다.

기본적인 실행 패턴입니다.

```
ansible <pattern_goes_here> -m <module_name> -a <arguments>
```

예,

```
$ ansible webservers -m service -a "name=httpd state=restarted"
```

패턴은 호스트 뿐만 아니라 그룹이 될 수도 있습니다.

만약 패턴에 다음과 같이,

```
all
*
```

라고 주었다면 이것은 관리 대상 목록 (Inventory)에 있는 모든 호스트를 의미합니다.

또는 호스트나 IP 주소를 직접 지정할 수도 있습니다.

```
one.example.com
one.example.com:two.example.com
192.168.1.50
192.168.1.*
```

아래와 같이 하나의 그룹 또는 여러개의 그룹을 지정할 수도 있습니다.

```
webservers
webservers:dbservers
```

만약 다음과 같이 지정한다면

```
webservers:! phoenix
```

이것은 `webservers` 그룹에는 있으나 `phoenix` 그룹에는 없는 호스트만을 의미합니다.

```
webservers:&staging
```

반면 위에 내용은 `webservers` 와 `staging` 그룹에 모두 포함된 호스트를 의미합니다.

여러 조합도 가능한데,

```
webservers:dbservers:&staging:!phoenix
```

위의 패턴은 `webservers` 또는 `dbservers` 그룹에 포함되는 모든 호스트 중에서 `staging` 그룹에 있으면서 `phoenix` 그룹에는 존재하지 않는 호스트를 의미합니다.

```
*.example.com
*.com
```

와일드카드 `*` 를 사용하여 호스트명을 지정할 수 있으면,

```
one*.com:dbservers
```

와일드카드를 사용한 호스트명 뿐만 아니라 그룹명을 같이 사용해도 됩니다.

그룹에 있는 호스트 중 일부만 선택할 수도 있습니다.

예를 들어,

```
[webservers]
cobweb
webbing
weber
```

`webservers` 그룹에 위와 같이 세개의 호스트가 있다면

```
webservers[0]       # == cobweb
webservers[-1]      # == weber
webservers[0:1]     # == webservers[0],webservers[1]
                    # == cobweb,webbing
webservers[1:]      # == webbing,weber
```

위와 같이 색인을 통하여 선택할 수도 있습니다.

```
~(web|db).*\.example\.com
```

심지어는 위와 같이 패턴에 정규식을 지정하는 것도 가능합니다.

애드-혹 명령어 소개

애드-혹 명령은 필요시 한번만 수행하고 이후 동일한 수행을 위하여 저장할 필요가 없는 명령입니다. 플레이북 실행에 앞서 간단히 ansible을 할 수 있는 기능을 살펴보는데 도움이 될 수 있습니다. 예를 들어 랩의 전체 휴가에 앞서 모든 호스트를 일괄적으로 끄는데 있어 플레이북을 작성하는 것 보다는 애드-혹 명령을 내리는 것이 더 간단합니다.

병렬화 및 쉘 명령어

```
$ ansible atlanta -a "/sbin/reboot" -f 10
```

위의 명령어는 간단히 atlanta 그룹에 있는 호스트에 `/sbin/reboot` 명령을 내리는데 `-f 10` 옵션으로 10개 까지 동시성을 제공한다는 것입니다. 여기서 동시성이라는 것은 동시에 10개씩의 호스트에 접속하여 명령을 내린다는 의미입니다.

위의 명령은 root 권한만 가능한데 연결은 특정 SSH 사용자로 연결하는 경우가 많으므로

```
$ ansible atlanta -a "/usr/bin/foo" -u username --become --ask-become-pass
```

위와 같이 할 수 있습니다.

```
$ ansible raleigh -m shell -a 'echo $TERM'
```

애드-혹 명령을 수행하는데 위에서 처럼 특정 모듈 `-m shell` 을 지정할 수 있습니다. (실은 `-m` 이 생략되면 디폴트 모듈인

`command` 모듈이 실행되며 해당 모듈의 수행 명령으로 `-a ...` 이 동작하게 됩니다.)

파일 전송

관리 대상 호스트의 파일에 대하여 파일 전송을 하려면

```
$ ansible atlanta -m copy -a "src=/etc/hosts dest=/tmp/hosts"
```

`copy` 모듈을 이용하면 됩니다.

```
$ ansible webservers -m file -a "dest=/srv/foo/a.txt mode=600"
$ ansible webservers -m file -a "dest=/srv/foo/b.txt mode=600 owner=mdehaan group=mdehaan"
```

`file` 모듈을 이용하여 파일 퍼미션이나 속성을 변경할 수 있습니다.

> **노트** : `file` 모듈에서 사용되는 옵션은 `copy` 모듈에서도 동일하게 적용됩니다.

```
$ ansible webservers -m file -a "dest=/path/to/c mode=755 owner=mdehaan group=mdehaan state=direc
```

`file` 모듈은 `mkdir -p` 명령과 같이 해당 폴더를 생성할 수 있습니다.

```
$ ansible webservers -m file -a "dest=/path/to/c state=absent"
```

반대로 해당 디렉터리를 삭제할 수 있습니다.

패키지 관리

레드햇 계열의 리눅스에 패키지 관리하는 yum 및 데비안,우분투 계열의 패키지 관리하는 apt (aptitude)를 사용하는 모듈을 이용할 수 있습니다.

```
$ ansible webservers -m yum -a "name=acme state=present"
```

`yum` 모듈로 **acme** 패키지를 설치하는데 패키지 update는 하지 않습니다.

```
$ ansible webservers -m yum -a "name=acme-1.5 state=present"
```

위와 같이 특정 버전을 지정할 수 있습니다.

```
$ ansible webservers -m yum -a "name=acme state=latest"
```

가장 최신버전의 패키지를 설치합니다.

```
$ ansible webservers -m yum -a "name=acme state=absent"
```

해당 패키지 `acme` 가 설치되어 있지 않은지 확인합니다.

사용자 및 그룹

```
$ ansible all -m user -a "name=foo password=<crypted password here>"
$ ansible all -m user -a "name=foo state=absent"
```

리눅스의 사용자와 그룹과 관련된 명령을 내릴 수 있습니다.

소스 버전 관리 시스템에서 배포

```
$ ansible webservers -m git -a "repo=git://foo.example.org/repo.git dest=/srv/myapp version=HEAD"
```

git 버전 관리를 이용하여 관리 대상 호스트에 해당 소스를 가져올 수 있습니다.

Ansible 모듈은 버전관리 시스템에서 해당 코드가 업데이트되었을 때 알 수 있는 변경 핸들러를 통해 알 수 있으므로 git에서 해당 프로그램이나 코드를 가져오고 아파치 서버를 재기동 하는 등의 일을 할 수 있습니다.

서비스 관리

```
$ ansible webservers -m service -a "name=httpd state=started"
```

`httpd` 서비스가 시작되었는가 확인하고 안되어 있으면 시작합니다.

```
$ ansible webservers -m service -a "name=httpd state=restarted"
```

유사하게 재기동 시킵니다.

```
$ ansible webservers -m service -a "name=httpd state= stopped"
```

해당 서비스를 멈춥니다.

시간 제한을 갖는 백그라운드 작업

오래 걸리는 작업은 일단 백그라운드로 동작하게 한 다음 일정 시간이 지나 그 상태를 조사할 수 있습니다. 예를 들어 `long_running_operation` 라는 작업을 비 동기적으로 백그라운드에서 실행시키고 3600초의 시간 (`-B 3600`)을 갖고 폴링하지 않는 (`-P 0`) 경우,

```
$ ansible all -B 3600 -P 0 -a "/usr/bin/long_running_operation --do-stuff"
```

위와 같이 설정할 수 있습니다.

만약 백그라운드로 실행만 시켰을 경우에는 해당 작업 ID를 가지고 다음과 같이,

```
$ ansible web1.example.com -m async_status -a "jid=488359678239.2844"
```

`async_status` 모듈을 이용하여 이후 상태를 조사할 수 있습니다.

그것이 아니면 폴링 방식이 디폴트이며,

```
$ ansible all -B 1800 -P 60 -a "/usr/bin/long_running_operation --do-stuff"
```

위 명령은 백그라운드로 최대 30분(1800초)를 돌고 나서 매 1분(60초) 마다 폴링을 하면서 그 결과를 확인합니다.

Fact 모음

fact는 플레이북 입장에서 해당 시스템에서 사용가능한 변수등을 나타내는 것입니다. 이것은 작업을 실행하는데 있어 조건 제어문을 수행하는 사용될 뿐만 아니라 해당 시스템의 애드-혹 정보를 구할 수도 있습니다.

```
$ ansible all -m setup
```

설정 파일

ansible 설정 파일은 다음과 같은 순서로 찾습니다.

- ANSIBLE_CONFIG (an environment variable)
- ansible.cfg (in the current directory)
- .ansible.cfg (in the home directory)
- /etc/ansible/ansible.cfg

가장 최근의 설정 구하기

`/etc/ansible` 디렉터리 안에 **.rpmnew** 와 같이 마지막 설정파일이 있을 수 있습니다.

환경 변수로 설정

`ANSIBLE_CONFIG` 환경변수로 설정할 수도 있습니다.

환경 설정 내용

디폴트 설정

`[default]` 부분에 있는 설정 내용입니다.

action_plugins

액션은 모듈 실행, 템플릿 실행 등과 같은 일을 하기 위한 일련의 코드입니다. 이런 액션 플러그인은 필요시 해당 코드가 있는 플러그인 폴더를 지정합니다.

```
action_plugins = ~/.ansible/plugins/action_plugins/:/usr/share/ansible_plugins/action_plugins
```

allow_world_readable_tmpfiles

> 버전 2.1에 추가됨

이 항목은 임시 파일이 관리 대상 머신에 생성되도록 하여 작업을 실패로 나타내는 대신 경고로 뜨도록 하는데 권한을 가지고 있지 않은 사용자로 작업할 때 유용합니다.

```
allow_world_readable_tmpfiles=True
```

ansible_managed

이 문자열은 다른 곳에서 사용될 때 치환될 수 있습니다.

```
{{ ansible_managed }}
```

와 같이 어디선가 사용되었고,

ansible 설정에,
`ini ansible_managed = Ansible managed: {file} modified on %Y-%m-%d %H:%M:%S by {uid} on {host}`
라고 되어 있다면 이 문자열이 치환됩니다.

ask_pass

이것은 플레이북이 실행될 때 암호를 물을지 안 물을지 나타내는 플래그입니다. 디폴트는 묻지 않습니다.

```
ask_pass=True
```

> 만약 보안을 위하여 모두 SSH 키로 접속한다면 이 항목은 False 로 놓아야 합니다.

ask_sudo_pass

Ansible이 플레이북을 통하여 관리 대상 장비에서 `sudo` 명령을 내렸고 이 경우 암호를 물어볼지 안 물어볼지를 나타내는 플래그입니다. 디폴트는 묻지 않습니다.

```
ask_sudo_pass=True
```

ask_vault_pass

Ansible에서 볼트(금고)는 암호 등의 알려지면 안되는 민감한 정보를 담고 있는 공간입니다. 이 볼트 암호를 물어볼지 안물어볼지 나타내는 플

래그입니다. 디폴트는 안물어봅니다.

```
ask_vault_pass=True
```

bin_ansible_callbacks

> 버전 1.8 이후

ansible 이 기동될 때 콜백 플러그인 들을 불러올지 아닐지를 나타내는 플래그입니다. ansible 명령에서 로그를 남기거나 결과를 알려주는 등의 역할을 수행한다면 이 플래그를 True로 놓습니다. (하지만 로드하는데 시간은 더 걸립니다) 하지만 `/usr/bin/ansible-playbook` 은 이 플래그에 상관없이 항상 콜백 플러그인을 로드합니다.

callback_plugins

특정 이벤트 발생 시 호출되거나 어떤 알림에 의한 수행 등을 하는 코드를 ansible에서는 콜백이라 합니다. 이런 콜백 플러그인이 있는 디렉터리 위치를 나타내는 항목입니다.

```
callback_plugins = ~/.ansible/plugins/callback:/usr/share/ansible/plugins/callback
```

stdout_callback

> 2.0에서 추가

이 설정은 ansible-playbook 에서 stdout 에 대한 기본 콜백을 변경합니다.

```
stdout_callback = skippy
```

callback_whitelist

> 2.0에서 추가

현재 바로 사용가능한 모든 콜백 플러그인이 이미 존재하지만 특별히 지정하지 않는한 사용할 수 없습니다. 이 설정은 사용가능한 콜백 플러그인을 지정하도록 합니다.

```
callback_whitelist = timer,mail
```

command_warnings

> 1.8에서 추가

아래와 같이

```
- name: usage of git that could be replaced with the git module
  shell: git update foo warn=yes
```

플레이북 설정에서 shell 명령으로 `git update foo` 명령을 내렸을 때 해당 명령 결과의 경고가 나타날 수 있는데 이것을 디폴트로 보

이게 할 것인지 아닌지를 나타내는 플래그 입니다.

하지만 위와 같이 개별 명령 뒤에 warn=yes 등으로 전체 설정을 덮어쓸 수 있습니다.

connection_plugins

연결 플러그인은 ansible에서 명령이나 파일을 전달하기위한 채널을 확장하는 사용되는 것입니다.

```
connection_plugins = ~/.ansible/plugins/connection_plugins/:/usr/share/ansible_plugins/connectior
```

deprecation_warnings

> 1.3에서 추가

`ansible-playbook` 명령의 결과에 예전 명령에 대한 경고 (deprecating warning)를 보이게 할지 아닐지를 나타내는 플래그입니다.

```
deprecation_warnings = True
```

display_args_to_stdout

> 2.1.0에서 추가

기본적으로 `ansible-playbook` 은 각각의 작업에 대한 헤더 정보를 stdout에 보여줍니다. `name:` 이나 `shell:` 과 같은 설정 헤더입니다. (YAML에서의 키 값이죠) 해당 명령을 수행할 때 그 다음 명령을 함께 stdout으로 출력하게 하거나 아니거나를 나타내는 플래그입니다.

```
display_args_to_stdout=False
```

display_skipped_hosts

```
display_skipped_hosts=True
```

error_on_undefined_vars

```
error_on_undefined_vars=False
```

만약 이 플래그가 false 이면 `{{ template_expression }}` 와 같이 확장될 변수가 존재하지 않아도 그대로 출력됩니다. 만약 True 이고 확장될 변수가 존재하지 않으면 오류가 발생합니다.

executable

sudo 명령을 한 다음 수행될 쉘을 나타냅니다.

```
executable = /bin/bash
```

filter_plugins

필터는 템플릿에 이용되는 특별한 함수입니다. 이런 필터 플러그인 폴더를 지정합니다.

```
filter_plugins = ~/.ansible/plugins/filter_plugins/:/usr/share/ansible_plugins/filter_plugins
```

force_color

TTY 가 없이도 수행될 때 컬러 모드를 켤지를 나타냅니다.

```
force_color = 1
```

force_handlers

> 1.9.1에서 추가

대상 호스트에서 수행 시 오류가 발생했더라도 알림 핸들러가 수행될지를 나타내는 플래그 입니다.

```
force_handlers = True
```

forks

이것은 관리 대상 호스트가 여러개가 있고 동일한 명령을 내리는데 동시에 얼마만큼의 호스트에 적용하는 가를 나타냅니다. 디폴트는 5 입니다. 일반적으로 50 내지 500 이상 까지 설정하는 경우도 있습니다. 디폴트 5는 너무 작게 잡아놓은 것입니다.

```
forks=5
```

gathering

버전 1.6 부터 원격 시스템에서 발견된 변수를 모으는 팩트에 관한 정책을 가리키는 항목입니다. ***implicit***가 디폴트이며 이것은 `gather_facts: False` 가 지정되어 있지 않는한 캐쉬를 무시하고 무조건 해당 정보를 모읍니다. ***smart*** 라는 값을 주면 해당 호스트의 팩트가 존재하지 않은 경우 가져오려고 하고 캐쉬에 있다면 그 정보를 이용합니다.

```
gathering = smart
```

> 2.1 부터는 아래와 같이 변경됨

```
gather_subset = all
```

- **all** : 모든 팩트를 모음 (디폴트)
- **network** : 네트워크 팩트를 모음
- **hardware** : 하드웨어 팩트를 모음
- **virtual** : 대상 머신에 호스트되고 있는 가상머신 팩트를 모음
- **ohai** : ohai 에서 팩트 모음
- **facter** : facter 에서 팩트 모음

```
# Don't gather hardware facts, facts from chef's ohai or puppet's facter
gather_subset = !hardware,!ohai,!facter
```

위와 같이 콤마로 목록을 줄 수도 있고 `!` 을 이용하여 제외하고 의미를 가질 수도 있습니다.

만약 아주 기본적인 것을 제외한 모든 팩트를 가져오지 않으려면,

```
gather_subset = !all
```

라고 하면 됩니다.

hash_behaviour

```
hash_behaviour=replace
```

hostfile

> 1.9 부터 사용되지 않음. 대신 ***inventory*** 항목 이용

host_key_checking

```
host_key_checking=True
```

inventory

관리 대상 목록을 지정하는 파일 및 스크립트 또는 디렉터리 등의 위치를 지정합니다.

```
inventory = /etc/ansible/hosts
```

jinja2_extensions

```
jinja2_extensions = jinja2.ext.do,jinja2.ext.i18n
```

library

Ansible이 디폴트로 모듈을 찾는 위치를 지정합니다.

```
library = /usr/share/ansible
```

콜론을 주어 여러 경로를 같이 줄 수 있습니다. 또한 `./library` 폴더를 같이 찾습니다.

local_tmp

> 2.1 이후

Ansible이 관리 대상 장비에서 명령을 수행하거나 모듈을 수행할 때, 수행할 모듈의 내용 등이 내려가서 해당 모듈을 수행합니다. 해당 명령이 종료될 때 내려갔던 모듈도 삭제됩니다. 이 항목은 이런 임시로 수행되는 폴더를 지정합니다.

```
local_tmp = $HOME/.ansible/tmp
```

Ansible은 이 해당 폴더 안에서 임의의 하위 디렉터리를 만들어 작업합니다.

log_path

```
log_path=/var/log/ansible.log
```

lookup_plugins

```
lookup_plugins = ~/.ansible/plugins/lookup_plugins/:/usr/share/ansible_plugins/lookup_plugins
```

module_set_locale

이 플래그가 활성화되면 원격 호스트에서 LANG, LC*MESSAGES, 및 LC*ALL 등이 설정됩니다.

> 2.1에서는 디폴트로 True 이고 2.2 에서는 디폴트로 False 입니다.

module_lang

모듈을 수행할 디폴트 LANG을 지정합니다.

```
module_lang = en_US.UTF-8
```

module_name

Ansible 에 `-m ...` 을 지정하지 않아도 수행할 디폴트 모듈을 나타냅니다.

```
module_name = command
```

nocolor

```
nocolor=0
```

nocows

`cowsay` 명령을 수행할 것인가를 나타내는 항목입니다.

```
nocows=0
```

pattern

```
hosts=*
```

poll_interval

```
poll_interval=15
```

private_key_file

```
private_key_file=/path/to/file.pem
```

remote_port

```
remote_port = 22
```

remote_tmp

```
remote_tmp = $HOME/.ansible/tmp
```

remote_user

```
remote_user = root
```

retry_files_enabled

Ansible에서 플레이북을 수행하다 실패하면 `.retry` 파일을 만듦니다. 이 파일을 만들 것인가 아닌가를 나타내는 플래그 입니다.

```
retry_files_enabled = False
```

retry_files_save_path

```
retry_files_save_path = ~/.ansible/retry-files
```

roles_path

`/etc/ansible/roles` 디폴트 역할 폴더 이외에 역할 폴더를 지정하는 플래그입니다.

```
roles_path = /opt/mysite/roles:/opt/othersite/roles
```

squash_actions

> 2.0 부터

```
squash_actions = apk,apt,dnf,package,pacman,pkgng,yum,zypper
```

strategy_plugins

```
strategy_plugins = ~/.ansible/plugins/strategy_plugins/:/usr/share/ansible_plugins/strategy_plugi
```

sudo_exe

```
sudo_exe=sudo
```

sudo_flags

```
sudo_flags=-H -S -n
```

sudo_user

```
sudo_user=root
```

system_warnings

```
system_warnings = True
```

timeout

SSH 타임아웃 시간(초) 입니다.

```
timeout = 10
```

transport

`ansible` 이나 ansible-playbook 에서 -c 을 지정하지 않는 한 사용할 전송 모듈입니다. 디폴트는 `smart` 인데 만약 `ssh` 로 충분하면 `ssh` 모듈로 하고 아니면 `paramiko` 전송 모듈을 사용합니다.

다른 전송 모듈로는 `local` , `chroot` , `jail` 등이 있습니다.

vars_plugins

```
vars_plugins = ~/.ansible/plugins/vars_plugins/:/usr/share/ansible_plugins/vars_plugins
```

vault*password*file

> 1.7 이후

```
vault_password_file = /path/to/vault_password_file
```

권한 상승 관련 설정

become

```
become=True
```

become_method

```
become_method=su
```

become_user

```
become_user=root
```

become_ask_pass

```
become_ask_pass=True
```

become_allow_same_user

대부분의 경우 동일한 사용자가 `sudo` 명령을 일정시간내에 반복하면 암호를 물어보지 않습니다. 그러나 SELinux 등에서 *sudo* 명령 설정에 따라 이런 기능이 불가능할 수도 있습니다. 이런 경우 이 플래그를 True로 설정하면 됩니다.

Paramiko 관련 설정

Paramiko 모듈은 Enterprise Linux 6 또는 그 이전 버전에 디폴트 SSH 연결을 위하여 사용되었습니다.
`[paramiko_connection]` 섹션에 다음과 같은 설정이 올 수 있습니다.

record*host*keys

최초 SSH 로 원격 호스트에 접속하면 host_key를 `~/.ssh/known_hosts 등`에 저장합니다. 이것은 보안 때문에 그 서버가 동일한 서버인지 매번 SSH 접속시 체크합니다. 하지만 너무 많은 수의 원격 호스트가 있다면 이 과정에서 많은 시간을 소요할 수 있으므로 이 항목 및 호스트 키 체크 항목을 False로 놓으면 더 빨라질 수 있습니다. (그만큼 보안은 더 떨어집니다)

```
record_host_keys=True
```

proxy_command

> 2.1 에서 추가

실제 접속에 앞서 proxy 에 접속하도록 하는 명령을 내릴 수 있습니다.

```
proxy_command = ssh -W "%h:%p" bastion
```

OpenSSH 관련 설정

`[ssh_connection]` 섹션 안에 설정되는 항목들 입니다.

ssh_args

```
ssh_args = -o ControlMaster=auto -o ControlPersist=60s
```

성능향상을 위해서 *ControlPersist*를 30분까지 지정하기도 합니다. 만약 이 항목이 설정되었다면 `control_path` 설정은 사용되지 않습니다.

control_path

ControlPath 소켓을 저장할 위치 입니다. 디폴트는

```
control_path=%(directory)s/ansible-ssh-%%h-%%p-%%r
```

입니다.

특정 시스템에서 일반적인 소켓 이름 길이 (108자)를 넘어서면 오류가 발생할 수 있으므로 이런 경우에는,

```
control_path = %(directory)s/%%h-%%r
```

와 같이 지정하기도 합니다.

Ansible 1.4 이후부터는 `-vvvv` 옵션을 통해서 Contol Path 파일이름까지 확인할 수 있습니다.

scp*if*ssh

원격 시스템에 따라서 SSH는 지원하지만 SFTP를 지원하지 않는 경우도 있습니다. 이런 경우에만 True로 설정하기 바랍니다.

pipelining

파이프라이닝을 활성화 시키면 실제 원격 서버에서 수행되는 모듈을 구동시키는데 SSH 명령이 많이 줄어듭니다. 하지만 **sudo:** 명령을 만난다면 `/etc/sudoers` 파일에서 requiretty 를 빼야 정상적으로 동작 합니다. 따라서 이 플래그는 sudo 명령 호환성을 위하여 디폴트로 `False` 로 되어 있지만 속도를 위해서는 활성화 시키고 requiretty 를 비활성화 시키시기 바랍니다. (가속 모드와는 별도입니다)

가속 모드 설정

`[accelerate]` 섹션에 설정되는 것으로 가속 모드와 관련된 것입니다. **가속**은 파이프라이닝을 사용할 수 없는 경우 수행 속도를 높일 수 있는 것이지만 가능하면 사용안하는 편이... (???)

accelerate_port

> 1.3 이후

```
accelerate_port = 5099
```

accelerate_timeout

> 1.4 이후

```
accelerate_timeout = 30
```

accelerate_connect_timeout

> 1.4 이후

```
accelerate_connect_timeout = 30
```

accelerate_daemon_timeout

> 1.6 이후

```
accelerate_daemon_timeout = 30
```

accelerate_multi_key

> 1.6 이후

```
accelerate_multi_key = yes
```

SELinux 관련 설정

special_context_filesystems

> 1.9 이후

```
special_context_filesystems = nfs,vboxsf,fuse,ramfs,myspecialfs
```

libvirt_lxc_noseclabel

> 2.1 이후

```
libvirt_lxc_noseclabel = True
```

Galaxy 설정

`[galaxy]' 세션에 정의되는 설정입니다.

server

디폴트 갤럭시 서버 대신 사용할 서버를 지정합니다.

ignore_certs

갤럭시 접속 시 TLS 인증서 확인을 하지 않습니다.

BSD 지원

원본 참조하시기를...

윈도우 지원

원본 참조하시기를...

네트워킹 (장비) 지원

네트워킹 장비 연동

Ansible 버전 2.1 이후부터 서로 다른 이기종 네트워크 장비를 지원합니다.

- Arista EOS
- Cisco NXOS
- Cisco IOS
- Cisco IOSXR
- Cumulus Linux
- Juniper JUNOS
- OpenSwitch

네트워크 자동화 설치

다음과 같은 설치가 필요합니다.

- 가장 최근 버전의 Ansible network release 설치
- 테스트용 네트워크 플레이북을 다운

네트워크 장치에도 사용가능한 모듈

대부분의 표준 Ansible 모듈은 Linux/Unix 또는 윈도우 머신과 동작하도록 되어 있기 때문에 네트워크 장치에는 사용할 수 없습니다만 `slurp`, `raw` 또는 `setup` 등과 같이 플랫폼을 타지 않는 모듈은 네트워크 장치에도 사용가능합니다.

네트워크 장치에 사용할 수 있는 모듈이 어떤 것들이 있는지 알기 위해서 네트워크 장치 이용가능 모듈 색인을 참조하십시오.

네트워크 장치 연결

모든 핵심 네트워킹 모듈은 `provider` 라는 인자를 갖는데, 이것은 어떻게 장치에 연결되는가를 나타내는 특성을 나타냅니다.

각 핵심 네트워크 모듈은 해당 운영체제 지원과 전송 기능을 제공합니다. 운영 체제는 이런 모듈과 일대일 대응을 하며 전송 기능은 운영 체제와 일대다 관계로 구성됩니다.

각 핵심 네트워크 모듈은 전송기능을 담당하는 다음과 같은 기본 인자를 갖습니다.

- host : 원격 장비의 호스트명 또는 IP주소
- port : 연결되는 포트 번호
- username : 연결에 사용되는 사용자 ID
- password : 연결 사용되는 사용자 암호
- transport : 어떤 전송 형태인지 지정
- authorize : 장비에서의 권한 상승 방법
- auth_pass : 권한 상승 시 필요에 따른 함호

개별 모듈은 위와 같은 인자에 대하여 디폴트 값을 가질 수 있습니다. 예를 들어 일반적인 전송 방법은 ***CLI*** 입니다. 어떤 모듈은 `EOS(eapi)` 또는 `NXOS(nxapi)` 전송 방법을 지원하는 반면 어떤 모듈은 `CLI` 만 지원하기도 합니다. 모든 인자는 각 모듈에서 자세히 설명합니다.

위와 같은 전송 인자를 설정하는 것은 개별적으로 가능하므로 각 모듈은 원하는 데로 서로 다른 전송 방법이나 인증 방식을 사용할 수 있습니다.

이와 같이 접속하는 방법의 한가지 단점은 모든 작업이 필요한 인자를 꼭 포함해야 한다는 것입니다. 이런 이유 때문에 `provider` 인자가 필요하게 되었습니다. `provider` 인자는 키워드 인자를 받아 접속 및 인증에 필요한 패러미터를 해당 작업으로 전달해 줍니다.

다음 두 개의 설정은 동일한 nxos_config 모듈을 사용하지만 모든 핵심 네트워크 모듈에도 적용됩니다.

```
---
nxos_config:
   src: config.j2
   host: "{{ inventory_hostname }}"
   username: "{{ ansible_ssh_user }}"
   password: "{{ ansible_ssh_pass }}"
   transport: cli
```

```
---
vars:
   cli:
   host: "{{ inventory_hostname }}"
   username: "{{ ansible_ssh_user }}"
   password: "{{ ansible_ssh_pass }} "
   transport: cli


nxos_config:
   src: config.j2
   provider: "{{ cli }}"
```

위의 두개의 예제는 동일하지만 인자가 우선권 및 디폴트 등을 지정할 수 있다는 것을 보여줍니다.

```
---
vars:
    cli:
    host: "{{ inventory_hostname }}"
    username: operator
    password: secret
    transport: cli

tasks:
- nxos_config:
   src: config.j2
   provider: "{{ cli }}"
   username: admin
   password: admin
```

위의 예에서는, cli provider에 operator/secret 라는 사용자 이름/암호가 있지만 아랫부분의 task 접속 사용자와 암호 admin/admin를 대신 이용할 것입니다.

이런 방식은 전송 방법을 기술하는 provider의 모든 인자에 동일하게 적용됩니다. 따라서 CLI 또는 NXAPI를 지원하는 단일 작업을 다음과 같이 가질 수 있습니다.

```
---
vars:
    cli:
    host: "{{ inventory_hostname }}"
    username: operator
    password: secret
    transport: cli

tasks:
  - nxos_config:
      src: config.j2
      provider: "{{ cli }}"
      transport: nxapi
```

하지만,

```
---
vars:
  conn:
  password: cisco_pass
  transport: cli

tasks:
- nxos_config:
  src: config.j2
  provider: "{{ conn }}"
```

위의 예제에서는 다음과 같은 오류가 발생할 겁니다.

```
"msg": "missing required arguments: username,host"
```

네트워킹 환경 변수

다음과 같은 환경 변수가 사용됩니다.

- ANSIBLE*NET*USERNAME : 사용자 ID
- ANSIBLE*NET*PASSWORD : 사용자 암호
- ANSIBLE*NET*SSH_KEYFILE : 키 파일
- ANSIBLE*NET*AUTHORIZE : 승인
- ANSIBLE*NET*AUTH_PASS : 승인 암호

변수는 다음과 같은 순서로 적용 됩니다. 가장 아래 부분이 가장 높은 우선 순위를 갖습니다.

1. 디폴트 값
2. 환경 변수
3. Provider
4. Task 인자

네트워킹 모듈에서 조건식

Ansible에서 플레이북의 명령이 수행될 때 조건식을 이용할 수 있습니다.

- eq : 같다
- neq : 같지 않다
- gt : 크다
- ge : 크거나 같다
- lt : 작다
- le : 작거나 같다
- contains : 포함됨

조건문은 원격 장치에서 수행된 명령의 결과를 비교합니다. 일단 작업이 명령 세트가 수행되다 ***waitfor*** 인자를 만나게 되면 이전 수행되던 작업이 플레이북으로 제어를 리턴하기 전 그 결과를 조사할 수 있습니다.

예를 들어,

```
---
- name: wait for interface to be admin enabled
  eos_command:
      commands:
          - show interface Ethernet4 | json
      waitfor:
          - "result[0].interfaces.Ethernet4.interfaceStatus eq connected"
```

위의 예제에서 `show interface Ethernet4 | json` 명령이 원격 장치에서 수행되고 나서 그 결과를 확인하는데, 만약 `(result[0].interfaces.Ethernet4.interfaceStatus)` 가 **connected** 가 될 때까지 이전 명령을 반복합니다. 이 과정은 재시도 횟수 만큼 반복합니다. (디폴트로 1초 간격으로 10번 재시도를 합니다)

command 모듈은 여러번 나올 수도 있습니다.

```
---
- name: wait for interfaces to be admin enabled
  eos_command:
      commands:
          - show interface Ethernet4 | json
          - show interface Ethernet5 | json
      waitfor:
          - "result[0].interfaces.Ethernet4.interfaceStatus eq connected"
          - "result[1].interfaces.Ethernet4.interfaceStatus eq connected"
```

위의 예제에서는 Ethernet4, 5를 확인(show interface) 하는 명령을 수행하고 그 결과를 조사합니다. waitfor 모듈의 결과는 항상 result로 나오며 앞 명령의 순서대로 [0], [1], ... 과 같은 순서로 지정하여 결과를 확인합니다.

플레이북 (Playbook)

플레이북은 Ansible의 핵심으로서 설정, 배포 및 지휘 언어로 동작해주게 하는 것입니다. 이것을 통하여 원격 시스템이 일련의 IT 작업을 수행하도록 하는 정책을 기술합니다.

만약 Ansible 모듈이 작업장의 공구이고 관리대상목록(Inventory)의 호스트들이 작업할 대상 이라면 플레이북은 실제 작업을 어떻게 수행하고 처리되는지를 나타내는 사용설명서와 같은 것입니다.

기본적으로 플레이북은 원격 머신을 설정하고 배포하는 등의 관리를 할 수 있습니다. 좀 더 고급스럽게는 다단계의 롤링 업데이트를 수행하고 다른 호스트에 수행할 행동을 지정하고 서버를 모니터링하여 로드 밸런싱을 하는 등의 고급 사용까지도 가능합니다.

비록 플레이북에 관한 많은 정보가 이 문서에서 제공된다 하더라도 모든 정보를 단번에 모두 알려고 하지 않아도 됩니다. 시간이 되는데로 원하는 기능을 하나씩 익혀가도 됩니다.

플레이북은 사람이 읽기 쉽게 설계되어 있으며 기본적인 텍스트로 구성되어 있습니다. 또한 플레이북을 구성하는 파일의 포함 방법 등을 비롯하여 아주 다양한 방법이 존재합니다.

이 책을 읽어가면서 예제 플레이북를 참고하시는게 많은 도움이 될 것입니다. 단지 어떻게 사용하는가 뿐만 아니라 다양한 배포 및 지휘 등에 대한 최선의 방법 등도 배울 수 있습니다.

플레이북 소개

플레이북에 관하여

플레이북은 단순히 Ansible의 애드-혹 명령을 사용하는 것 과는 완전히 다른 방식으로 강력한 기능을 제공합니다.

간단히 이야기 해서, 플레이북은 기존의 유사한 시스템이 아닌, 설정 관리의 기본 바탕을 제공하고 여러 머신의 배포 시스템으로 사용될 수 있으며 여러 다양한 응용을 할 수 있도록 되어 있습니다.

플레이북은 설정을 정의할 뿐만 아니라 일일이 지정하는 절차에 따라 수행되는 단계를 지휘할 수 있는데 관리 대상 장비를 특정 순서에 따라 앞에서 혹은 뒤에서 등의 다른 단계로 적용할 수 있습니다. 또한 작업을 동기 또는 비동기식으로 실행할 수 있습니다.

애드-혹 명령을 수행할 때 `/usr/bin/ansible` 명령을 실행했던 것과는 달리, 플레이북은 소스 버전관리 하듯이 설정을 밀어넣고 원격 시스템에서 해당 설정이 잘 적용되었는지 확인합니다.

플레이북에 관한 좀더 자세한 예제를 얻으려면 Ansible 예제 저장소를 참고합니다. 브라우저의 새로운 탭에서 이 예제 저장소를 열고 살펴보십시오.

플레이북 언어 예제

플레이북은 YAML 형식 (YAML 문법 참조)으로 표현됩니다.

각 플레이북은 하나 이상의 `plays` 목록으로 구성됩니다.

`play` 의 목적은 일련의 호스트를 어떤 잘 정의된 역할(`task` 라 불리우는)과 매핑되도록 하는 것입니다. 기본적으로 `task` 는 Ansible 모듈을 호출하는 것과 다름 없습니다. (모듈에 관하여참조)

플레이북에 여러 `play` 를 구성함으로써 여러 머신의 배포 뿐만 아니라 웹서버 그룹에 있는 모든 머신에 특정 작업을 수행하고 데이터베이스 서버 그룹에 다른 작업을 수행하는 지휘체계를 가집니다.

다음과 같이 하나의 플레이를 담고 있는 플레이북을 살펴보면,

```
---
- hosts: webservers
  vars:
    http_port: 80
    max_clients: 200
  remote_user: root
  tasks:
  - name: ensure apache is at the latest version
    yum: name=httpd state=latest
  - name: write the apache config file
    template: src=/srv/httpd.j2 dest=/etc/httpd.conf
    notify:
    - restart apache
  - name: ensure apache is running (and enable it at boot)
    service: name=httpd state=started enabled=yes
  handlers:
    - name: restart apache
      service: name=httpd state=restarted
```

다음은 동일한 내용이지만 모듈 안에 인자를 YAML의 `Key:Value` 형식으로 나누어 표시한 것입니다.

```yaml
---
- hosts: webservers
  vars:
    http_port: 80
    max_clients: 200
  remote_user: root
  tasks:
  - name: ensure apache is at the latest version
    yum:
      name: httpd
      state: latest
  - name: write the apache config file
    template:
      src: /srv/httpd.j2
      dest: /etc/httpd.conf
    notify:
    - restart apache
  - name: ensure apache is running
    service:
      name: httpd
      state: started
  handlers:
    - name: restart apache
      service:
        name: httpd
        state: restarted
```

플레이북은 또한 한개 이상의 플레이를 가질 수 있습니다.

```
---
- hosts: webservers
  remote_user: root

  tasks:
  - name: ensure apache is at the latest version
    yum: name=httpd state=latest
  - name: write the apache config file
    template: src=/srv/httpd.j2 dest=/etc/httpd.conf

- hosts: databases
  remote_user: root

  tasks:
  - name: ensure postgresql is at the latest version
    yum: name=postgresql state=latest
  - name: ensure that postgresql is started
    service: name=postgresql state=started
```

위의 예제에서는 우선 웹서버 관련 플레이를 진행한 다음 데이터베이스 작업을 수행하는 것을 보여줍니다.

다른 사용자로 로그인하고 필요 시 sudo 명령을 하는 등, 호스트 그룹 간에 다른 대상의 작업을 진행할 수 있습니다. 태스크와 마찬가지로 플레이도 위에서 아래로 순차적으로 작업이 진행됩니다.

기본기

호스트와 사용자

각 플레이에는 작업 대상 호스트가 어느 것인지 지정해야 합니다.

`hosts` 라인이 하나 이상의 호스트나 호스트 그룹으로 작업 대상을 나타냅니다. 표기 방식은 패턴에 나타난 것과 동일합니다. `remote_user` 내용은 원격 사용자 ID를 나타냅니다.

```
---
- hosts: webservers
  remote_user: root
```

> **노트** : 버전 1.4 이전까지는 `remote_user` 대신 ***user*** 라고 했지만 `user` 모듈과 혼동될 우려가 있어 이름이 바뀌었습니다.

`remote_user` 는 플레이 안에 있는 태스크에 나올 수도 있습니다.

```
---
- hosts: webservers
  remote_user: root
  tasks:
    - name: test connection
      ping:
      remote_user: yourname
```

> **노트** : 태스크 안에 `remote_user` 항목의 지원은 버전 1.4 이후입니다.

`become` 을 이용하여 권한 상승 (sudo, su등, Become 참조)도 지원합니다.

```
---
- hosts: webservers
  remote_user: yourname
  become: yes
```

또는 다음과 같이 특정 태스크에만 권한 상승을 할 수 있습니다.

```
---
- hosts: webservers
  remote_user: yourname
  tasks:
    - service: name=nginx state=started
      become: yes
      become_method: sudo
```

> **노트** : `became` 은 버전 1.9 이전에는 `sudo/su` 이었습니다.

또한 어떤 사용자로 로그인한 후 *root* 가 아닌 다른 사용자가 될 수 있습니다.

```
---
- hosts: webservers
  remote_user: yourname
  become: yes
  become_user: postgres
```

`become_method` 를 이용하여 권한 상승을 위한 방법을 제시할 수 있습니다.

```
---
- hosts: webservers
  remote_user: yourname
  become: yes
  become_method: su
```

만약 sudo 권한 상승을 하는데 암호가 필요하다면 `ansible-playbook` 을 실행하는데 `--ask-become-pass` 옵션을 지정하여 암호를 입력하도록 하기 바랍니다. (이전 방법인 `--ask-sudo-pass` 또는 `-K` 을 이용해도 됩니다.) 만약 플레이북이 실행되는데 있어 멈춘듯이 보인다면 아마도 이런 권한 상승을 위하여 사용자 입력을 대기하고 있는 것일 수 있습니다. 이런 경우에는 ***Ctrl-C***를 눌러 해당 프로그램을 멈춘 다음 적절한 암호를 지정해 주시기 바랍니다.

> ***!중요*** 만약 `become_user` 항목을 이용하여 root가 아닌 다른 사용자 권한을 얻을 때, 해당 모듈 인자는 `/tmp` 위치내의 임의 임시파일로 저장됩니다. 이 모듈은 해당 명령이 실행되지마자 바로 삭제됩니다. 이런 방식은 `bob` 이라는 사용자가 `timmy` 권한을 가질 때도 동일하게 나타납니다. 하지만 `bob` 사용자가 `root` 가 되거나 `bob` 또는 `root` 로 로그인 될 때에는 적용되지 않습니다. 만약 이런 데이터가 보안에 취약하다고 생각된다면 *become_user* 등과 같이 암호화되지 않은 암호를 전송하지 않도록 합니다. Ansible은 *password* 관련 인자의 내용을 로깅하지 않도록 하고 있습니다.

태스크 목록

각 플레이는 하나 이상의 태스크 목록을 포함합니다. 태스크는 순서대로 모든 작업 대상 호스트에 대해서 작업이 수행되는데 모든 호스트의 작업이 끝나고 나서 다음 태스크로 이동합니다. 플레이에서 수행되는 태스크는 모든 호스트의 태스크 명령들에 의해 모두 동일하게 수행되고 있음을 이해해야 합니다. 이것이 호스트 목록과 태스크를 매핑시키는 플레이의 목적입니다.

위에서 부터 아래로 차례대로 플레이북을 수행할 때 작업 실패가 발생한 호스트들은 전체 작업에서 제외됩니다. 만약 작업이 실패하면 플레이북 파일을 수정하고 다시 수행합니다.

각 태스크의 목표는 특정 인자를 지정하는 각각의 모듈을 실행하는 것입니다. 변수들은 이런 모듈 인자에 사용될 수 있습니다.

모듈은 `멱등성(idempotent)` 을 갖는데 만약 다시 수행되어도 원하는 상태가 되기를 바라는 만큼만 수행됩니다.

command 와 **shell** 모듈은 `chmod` 또는 `setsebool` 등과 같은 명령이 수행되는 것과 같은 명령을 그대로 수행합니다.

모든 태스크는 `name` 을 갖는데 플레이북이 실행되는 결과 중에 해당 이름이 나타납니다. 이 결과는 사람이 읽을 수 있는 형식이고 각 단계의 작업을 살펴보는데 유용합니다. 만약 `name` 이 지정되니 않는다면 결과에는 `action` 이라고만 나타날 것입니다.

이전 버전에는 `action: module options` 처럼 나타날 수 있지만 가능하면 `module: options` 와 같은 형식으로 사용하시기 바랍니다. 종종 오래된 플레이북을 살펴보면 이전 처럼 나타날 수 있지만 새로운 형시기 좀 더 읽기 좋습니다.

다음과 같이 위에 이야기한 태스크의 예가 있습니다.

```
tasks:
  - name: make sure apache is running
    service: name=httpd state=started
```

위의 예에서는 `service` 모듈에 대해서 ***key=value*** 방식으로 `name=httpd` 및 `state=started` 를 지정했습니다.

그러나,

```
tasks:
  - name: disable selinux
    command: /sbin/setenforce 0
```

위에서 처럼 `command` 와 `shelll` 모듈은 ***key=value*** 방식의 인자를 사용하지 않는 유일한 모듈입니다.

또한 `command` 와 `shelll` 모듈은 수행결과를 코드로 리턴하기 때문에 다음과 같이 성공인 경우, 또는 실패한 경우 다른 명령을 덧붙일 수 있습니다.

```
tasks:
  - name: run this command and ignore the result
    shell: /usr/bin/somecommand || /bin/true
```

또는,

```
tasks:
  - name: run this command and ignore the result
    shell: /usr/bin/somecommand
    ignore_errors: True
```

만약 모듈의 인자가 많아서 한줄을 넘어가는 경우에는 다음과 같이,

```
tasks:
  - name: Copy ansible inventory file to client
    copy: src=/etc/ansible/hosts dest=/etc/ansible/hosts
            owner=root group=root mode=0644
```

그 다음줄에 계속 인자를 사용하면 됩니다.

`vars` 섹션에 정의된 `vhost` 라는 변수를 태스크의 액션 라인에 사용하는 예 입니다.

```
tasks:
  - name: create a virtual host file for {{ vhost }}
    template: src=somefile.j2 dest=/etc/httpd/conf.d/{{ vhost }}
```

template 라는 곳에서도 변수를 사용하였습니다.

Action 축약

> 버전 0.8 이후

```
action: template src=templates/foo.j2 dest=/etc/foo.conf
```

0.8 이후에는

```
template: src=templates/foo.j2 dest=/etc/foo.conf
```

핸들러: 이벤트 발생시 수행

모듈은 멱등성을 갖도록 되어 있기 때문에 원격 시스템에서 변경이 이루어 졌을 때 전달 기능을 합니다. 플레이북은 이런 변경에 응답하기 위한 기본 핸들러를 가지고 있습니다.

플레이에 각 태스크 불럭의 마지막에서 액션을 트리거되고 다른 태스크에서도 알림이 떠도 한번만 핸들러가 동작합니다.

예를 들어, 웹서버의 설정 파일 변경되어 아파치 서버를 재기동시키라는 요구가 여러번 발생된다 하더라도 중간 중간 계속 아파치 서버가 재기동 될 필요는 없고 관련 작업이 모두 마친 다음 한번만 서비스 재기동을 하면 됩니다.

다음과 예는 특정 파일이 변경되었을 때 두 개의 서비스를 재시작하라는 것입니다.

```
- name: template configuration file
  template: src=template.j2 dest=/etc/foo.conf
  notify:
     - restart memcached
     - restart apache
```

위와 같이 `notify` 섹션에 있는 태스크를 **핸들러**라 합니다. 핸들러는 개별 `handlers` 섹션에 기술됩니다.

핸들러는 다른 일반 태스크와 동일한 태스크 목록이며 전체적으로 고유 이름을 갖고 핸들러 알림에 의해 발생합니다. 만약 핸들러에게 아무도 알려주지 않으면 핸들러는 수행되지 않습니다. 얼마나 많은 태스크에서 핸들러에게 알림을 하였던지 간에 모든 태스크가 끝나고 나서야 핸들러는 한번 수행됩니다.

다음은 이전 예제의 핸들러 입니다.

```
handlers:
    - name: restart memcached
      service: name=memcached state=restarted
    - name: restart apache
      service: name=apache state=restarted
```

Ansible 2.2 에서 핸들러는 더 일반 토픽을 *listen* 할 수 있으며 태스크는 이런 토픽에 알림을 보낼 수 있습니다.

```
handlers:
    - name: restart memcached
      service: name=memcached state=restarted
      listen: "restart web services"
    - name: restart apache
      service: name=apache state=restarted
      listen: "restart web services"

tasks:
    - name: restart everything
      command: echo "this task will restart the web services"
      notify: "restart web services"
```

위의 예제에서는 각각의 해당 이름이 아니라 `listen` 하고 있는 핸들러에 대해서 해당 태스크에서 알림을 보냄으로서 결과적으로 모든 핸들러가 수행되도록 하는 역할을 합니다.

> **노트**
>
> - 핸들러는 `notify` 에 호출된 순서나 listen 하는 순서가 아닌 항상 정의된 순서에 따라 수행됩니다.
> - 핸들러 이름과 `listen` 토픽은 전역 이름 영역입니다.
> - 만약 두 개의 핸들러가 동일 이름을 가지고 있다면 하나만 호출됩니다.
> - include 안에 정의된 핸들러는 호출될 수 없으나, 버전 2.1 이후 `static` 으로 선언된 외부 include에 정의된 핸들러는 불릴 수 있습니다.

핸들러에 대한 역할(Role)에 대한 것으로서,

- `pre_tasks` , `tasks` 및 `post_tasks` 섹션 에서 발생한 핸들러 호출은 발생한 섹션이 끝남과 동시에 자동으로 깨끗해집니다.
- `roles` 섹션에 발생한 핸들러는 `tasks` 섹션이 끝남과 따라 자동으로 깨끗이 되지만 다른 어떤 `tasks` 핸들러 앞에서는 해당되지 않습니다.

```
tasks:
   - shell: some tasks go here
   - meta: flush_handlers
   - shell: some other tasks
```

플레이북 실행

```
$ ansible-playbook playbook.yml -f 10
```

Ansible-Pull

설정을 관리 대상 장비에 밀어 넣고 작업하는 대신 설정 파일이 변경된 것을 찾아 적용할 필요가 있을 수 있습니다.

`ansible-pull` 은 작은 크기의 스크립트로서 git에 설정이 변경된 것을 체크아웃하여 `ansible-playbook` 을 실행합니다.

팁 및 트릭

플레이북이 실행되고 나서 나오는 결과 메시지를 확인하십시오. 어느 대상에 대하여 어떤 작업이 수행되었는지 나옵니다.

`--verbose` 플래그를 같이 이용하면 더 자세한 결과 메시지를 얻을 수 있습니다.

실제 플레이북 명령을 수행하기 전, 작업할 관리 대상 호스트 목록을 미리 구해보려면,

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

라고 하면 됩니다.

플레입북 Role 과 Include

소개

플레이북을 매우 큰 하나의 파일로 작성하는 것도 가능하지만 (초기에 플레이북을 배우면서는 이렇게 하기도 합니다) 점차 플레이북을 재사용하거나 계층적으로 관리할 필요도 생깁니다.

기본적으로는 플레이북의 내용을 여러 개로 나누어 단순히 포함하여 사용할 수 있습니다. 태스크 Incdlue 는 다른 파일에 있는 태스크를 가져올 수 있습니다. 핸들러 역시 외부에서 가져올 수 있기에 `handler` 섹션의 내용을 개벌 파일로 나눌 수 있습니다.

플레이북은 이처럼 안의 내용을 나누는 것 뿐만 아니라 다른 플레이북 파일을 불러올 수 있습니다. 이러면 플레이북이 실행되면서 다른 내용을 플레이북도 실행됩니다.

Ansible의 역할(Role)은 깨끗하고 재사용 가능한 추상화를 형성하는 파일들을 구성하는 것입니다.

우선 include 를 살펴본 다음 role을 살펴보도록 하겠습니다만 궁극적인 목적은 role을 잘 이해하는 것입니다. 플레이북을 더 잘 구성하면 할 수록 role을 잘 사용하게 될 것입니다.

태스크 가져오기(include) 및 재사용 의미

만약 플레이북의 플레이에 있는 일련의 태스크를 재사용한다고 가정해봅니다. ***include*** 를 이용하여 이 작업을 하면 됩니다. 가져온 태스크 목록은 특정 역할(role)을 수행하기에 알맞습니다. 다시한번 강조하면 플레이북은 관리 대상 시스템 그룹에 다양한 역할을 매핑하는 작업입니다.

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

- name: placeholder foo
  command: /bin/foo

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

위와 같은 작업이 `tasks/foo.yml` 에 있다고 하였을 때,

```
tasks:

  - include: tasks/foo.yml
```

위와 같이 `include` 를 이용하여 해당 태스크를 포함시킵니다.

include 할 때 변수도 포함해서 가져올 수 있습니다. (*parameterized include* 라고 합니다)

예를 들어, 다양한 wordpress 사이트를 배포한다고 할 때,

```
tasks:
  - include: wordpress.yml wp_user=timmy
  - include: wordpress.yml wp_user=alice
  - include: wordpress.yml wp_user=bob
```

위와 같이 wp_user 라는 변수를 지정하여 가져올 수 있습니다.

버전 1.0부터 시작하여 include에서의 패러미터를 다음과 같이 지정할 수도 있습니다.

```
tasks:

  - include: wordpress.yml
    vars:
        wp_user: timmy
        ssh_keys:
          - keys/one.txt
          - keys/two.txt
```

include에 작접 패러미터를 전달하던 아니면 `vars` 를 이용하던 상관없습니다. include 되는 파일에서는

```
{{ wp_user }}
```

와 같은 식으로 참조하면 됩니다.

아래의 예에서와 같이 handlers.yml 이라는 파일로 핸들러를 외부 파일로 빼 놓을 수 있습니다.

```
---
# this might be in a file like handlers/handlers.yml
- name: restart apache
  service: name=apache state=restarted
```

그리고 다음과 같이 include 합니다.

```
handlers:
  - include: handlers/handlers.yml
```

또한 다음과 같이 상위 플레이북에서 여러개의 하위 플레이북을 가져올 수 있습니다.

예를 들어,

```
- 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
```

플레이북에서 다른 플레이북을 include 하는 경우에는 변수 치환이 안된다는 것을 명심해야 한다.

> 주의
>
> `vars_files` 에서와 같이 include 파일에 조건을 주어 그 위치를 전달할 수 없습니다.

동적 혹은 정적 Include

Ansible 2.0 부터는 어떤 방식으로 include 시키는 방법이 약간 달라졌습니다. 이전 버전에서는 전처리기가 플레이북을 파싱하면서 단순 가져오기를 하였습니다. 이것은 그룹이나 호스트 변수와 같이 파싱 시에 알 수 없는 인벤토리 변수를 알 수 없다는 문제가 존재합니다.

Ansible 2.0에서는 ***동적(dynamic)*** include가 있어 실제 플레이가 실행되기 전 까지는 확장되지 않을 수 있습니다. 이런 변화는 다음의 예에서 처럼,

```
- include: foo.yml param={{item}}
  with_items:
  - 1
  - 2
  - 3
```

include 에 반복을 할 수 있습니다.

또한 동적 include 에서는 변수를 사용하는 것도 가능합니다.

```
- include: "{{inventory_hostname}}.yml"
```

하지만 동적 include는 다음과 같은 제약이 있음을 알아야 합니다.

- 동적 include에 존재하는 핸들러 이름을 이용하여 `notify` 할 수 없다.

- 동적 include에 있는 태스크를 `--start-at-task` 를 사용하여 수행할 수 없다.
- 동적 include에 있는 태그는 `-list-tags` 결과에 나오지 않는다.
- 동적 include에 있는 태스크는 `-list-tags` 결과에 나오지 않는다.

위의 문제를 해결하기 위하여 버전 2.1 이후에 ***정적(static)*** 옵션이 추가되었습니다.

```
- include: foo.yml
  static: <yes|no|true|false>
```

Ansible 2.1 이후 버전부터는 다음과 같은 조건을 만족하는 한 자동으로 동적(dynamic)이 아닌 정적(static) include를 합니다.

- include 가 loop을 이용하지 않음
- include 파일 이름이 변수를 사용하지 않음
- `static` 옵션이 `no` 인 경우
- ansible.cfg 옵션에서 static include가 비활성화 되어 있음

ansible.cfg 설정파일에 정적 include 를 위한 다음과 같은 두 가지 옵션이 있습니다.

- `task_includes_static` - 태스크 섹션에 있는 모든 include는 정적(static) 임
- `handler_includes_static` - 핸들러 섹션에 있는 모든 include는 정적(static) 임

Role

버전 1.2 이후.

태스크와 핸들러에 관하여 알았으므로 이제는 플레이북을 어떻게 잘 구성하는가를 알아낼 차례입니다. Role을 이용 하면 됩니다. Role은 파일 구조에 따라 vars_files, tasks 와 핸들러 등을 자동으로 로딩하는 것입니다. Role에 따른 내용을 그룹화 하는 것 또한 다른 사용자와 쉽게 공유하기 위함입니다.

다음과 같은 프로젝트 구조가 되어 있다고 합시다.

```
site.yml
webservers.yml
fooservers.yml
roles/
   common/
     files/
     templates/
     tasks/
     handlers/
     vars/
     defaults/
     meta/
   webservers/
     files/
     templates/
     tasks/
     handlers/
     vars/
     defaults/
     meta/
```

플레이북에는 다음과 같이:

```
---
- hosts: webservers
  roles:
     - common
     - webservers
```

할 수 있습니다.

다음과 같이 role `x` 에 대하여 작업이 진행됩니다.

- 만약 **roles/x/tasks/main.yml** 이 존재하면, 그곳에 있는 태스크 들이 플레이에 추가됩니다
- 만약 **roles/x/handlers/main.yml** 이 존재하면, 그곳에 있는 핸들러 들이 플레이에 추가됩니다
- 만약 **roles/x/vars/main.yml** 이 존재하면, 그곳에 있는 변수 들이 플레이에 추가됩니다
- 만약 **roles/x/defaults/main.yml** 이 존재하면, 그곳에 있는 변수 들이 플레이에 추가됩니다
- 만약 **roles/x/meta/main.yml** 이 존재하면, 그곳에 있는 role 의존성이 role 목록에 추가됩니다 (role 의존성은 아래에 따로 설명합니다)
- Role에 있는 다른 어떤 copy, script, template 또는 include task 들은 **role/x/{files,templates,tasks}/** 에 있는 것을 참조합니다.

버전 1.4 이후에는 role을 찾기위한 `roles_path` 설정 변수를 사용할 수 있습니다.

만약 해당 파일이 존재하지 않으면 해당 내용은 무시됩니다. 예를 들어 role을 위한 **vars/** 하위 디렉터리는 없을 수도 있습

니다.

다음과 같이 패러미터화 된 role 이 있을 수 있습니다.

```
---
- hosts: webservers
  roles:
    - common
    - { role: foo_app_instance, dir: '/opt/a',  app_port: 5000 }
    - { role: foo_app_instance, dir: '/opt/b',  app_port: 5001 }
```

자주 사용하지 않을 수 있지만 다음과 같이 조건적으로 사용할 수도 있습니다.

```
---
- hosts: webservers
  roles:
    - { role: some_role, when: "ansible_os_family == 'RedHat'" }
```

이것은 role 에 있는 모든 태스크에 조건적으로 적용됩니다. (자세한 것은 이후에)

마지막으로 태그를 지정하여 즉정 로그를 명시할 수 있습니다.

```
---
- hosts: webservers
  roles:
    - { role: foo, tags: ["bar", "baz"] }
```

role의 태스크에 있는 이런 tag는 role 안에 단순 정의된 tag를 우선합니다. 만약 만약 특정 role에 있는 여러 task를 부분적으로 사용할 필요가 생긴다면 해당 role을 더 작은 단위로 나눌 필요가 있습니다.

만약 플레이가 `tasks` 섹션을 그대로 가지고 있다면 이것은 role 이 먼저 적용되고 난 다음 실행됩니다.

만약 role 이전에 수행되어야 할 작업이 있다면,

```
---

- hosts: webservers

  pre_tasks:
    - shell: echo 'hello'

  roles:
    - { role: some_role }

  tasks:
    - shell: echo 'still busy'

  post_tasks:
    - shell: echo 'goodbye'
```

위에서 처럼 `pre_tasks` 를 이용합니다.

> **노트**
>
> 만약 태스크와 함께 태그를 사용하면 pre*tasks와 post*tasks에도 또한 태그가 있도록 합니다.

Role 디폴트 변수

버전 1.3 이후.

Role 디폴트 변수는 포함되거나 의존적인 Role에 디폴트 변수를 설정할 수 있도록 합니다. 디폴트를 생성하려면 role 디렉터리에 있는 ***defaults/main.yml*** 파일에 추가합니다. 이런 변수는 우선순위가 가장 낮으며 쉽게 다른 곳에 있는 동일 이름의 변수에 의해 덮어쓸 수 있습니다.

Role 의존성

버전 1.3 이후.

Role을 이용할 때 Role 의존성에 의해서 자동으로 다른 role을 수행하도록 하는 것입니다. 이 의존성은 ***meta/main.yml*** 파일에 포함됩니다. 이 파일은 role과 패러미터로 의존성을 가지는 rule 목록으로 구성됩니다.

```
---
dependencies:
  - { role: common, some_parameter: 3 }
  - { role: apache, apache_port: 80 }
  - { role: postgres, dbname: blarg, other_parameter: 12 }
```

또는 다음과 같이 role 경로를 직접 지정할 수도 있습니다.

```
---
dependencies:
   - { role: '/path/to/common/roles/foo', x: 1 }
```

Role 의존성은 버전관리나 tar 파일 (*galaxy*)을 이용해 설치될 수 있습니다. Role 의존성은 해당 Role을 포함하는 것이 실행되기 전에 재귀적으로 실행됩니다. 디폴트로 Role은 의존성으로 단 한번만 추가됩니다. 만약 또다른 role 목록이 의존성으로 나타난다면 다시 실행되지 않을 겁니다. 이런 방식은 `allow_duplicates: yes` 내용을 ***meta/main.yml file***에 넣어 줌으로써 여러번 실행될 수 있습니다. 예를 들어 `car` 라는 role이 `wheel` 이라는 rule 의존성을 갖도록 아래와 같이,

```
---
dependencies:
- { role: wheel, n: 1 }
- { role: wheel, n: 2 }
- { role: wheel, n: 3 }
- { role: wheel, n: 4 }
```

되어 있고 ***meta/main.yml*** 에 다음과 같이

```
---
allow_duplicates: yes
dependencies:
- { role: tire }
- { role: brake }
```

되어 있다면 그 실행 결과는,

```
tire(n=1)
brake(n=1)
wheel(n=1)
tire(n=2)
brake(n=2)
wheel(n=2)
...
car
```

와 같이 실행될 것입니다.

Role에 모듈 임베딩 및 플러그인

이 내용은 고급 사용자를 대상으로 합니다.

만약 커스텀 모듈을 개발(모듈 개발참조)하거나 플러그인을 개발(플러그인 개발참조)한다면 이것을 role의 일부로 배포하고 싶을 것입니다. 일반적으로 프로젝트로서의 Ansible은 핵심 모듈을 포함하여 잘 연동되어야 하는데 이런 부분이 어렵지 않도록 되어 있습니다.

AcmeWidgets 라는 회사에서 근무한다고 가정하고 내부 소프트웨어를 설정하는데 일익하는 내부 모듈을 개발하며, 같은 회사에 있는 다른 사람들이 당신이 개발한 모듈을 쉽게 이용하게 한다고 할때 모든 사람들에게 그 사용방법을 일일이 제공하지 않다고 됩니다.

role의 `tasks` 나 `handlers` 구조와는 별개로 `library` 라는 디렉터리를 추가합니다. 이 `library` 디렉터리에 필요한 모듈을 넣기만 하면 됩니다.

```
roles/
   my_custom_modules/
       library/
          module1
          module2
```

위와 같이 폴더 구성이 되어 있다면

다음과 같이,

```
- hosts: webservers
  roles:
    - my_custom_modules
    - some_other_role_using_my_custom_modules
    - yet_another_role_using_my_custom_modules
```

위와 같이 자신의 role을 미리 불러 놓으면 됩니다.

이와 같은 방식은 또한 테스트를 위하여 Ansible의 핵심 모듈을 직접 수정하여 테스트 해 보는데도 이용될 수 있습니다.

동일한 메카니즘으로 role에 플러그인이 사용될 수 있습니다.

```
roles/
   my_custom_filter/
       filter_plugins
          filter1
          filter2
```

Ansible Galaxy

Ansible Galaxy는 많은 사람들이 참여하는 커뮤니티로서 필요한 role을 찾고 다운로드하며 평가하고 리뷰하는 등의 일을 할 수 있는 사이트 이며 필요한 작업을 얻어 자신이 원하는 자동화를 실현하는 시작점으로 활용할 수 있습니다.

변수

자동화는 기본적으로 반복작업을 하는 것이지만 시스템에서는 항상 같은 상황일 수 없습니다. 어떤 시스템에서는 이렇게 설정하는 것도 다른 시스템에서는 약간 다르게 설정할 필요도 있습니다.

또한 실행에 따라 그 중간 결과를 보고 판단해야 할 수도 있습니다.

대부분 같지만 이렇게 부분 부분 다를 수 있는 것은 템플릿(template)을 이용하되 그 안에 변수로 다른 부분들을 설정합니다.

Ansible에서의 변수는 서로 다른 시스템을 유사하게 처리하는가를 의미합니다.

변수와 더불어 조건문과 반복문을 알아야 할 필요가 있습니다. **group_by** 모듈과 `when` 조건식 또한 변수를 이용하여 서로 다른 시스템을 활용합니다.

github를 포함하여 많은 예제를 살펴볼 것을 권해드립니다.

사용가능한 변수 이름

변수를 사용하기에 앞서 우선 사용가능한 변수 이름을 잘 알고 있는 것이 중요합니다. 영문자로 시작하고 영문자나 숫자 언더스코어로 구성된 변수명이 올바른 변수명입니다.

`foo_port` 또는 `foo5` 는 올바는 변수명이고 ***foo-port***, ***foo port***, ***12*** 는 모두 올바른 변수 이름이 아닙니다.

YAML 키:값 쌍으로 구성되어 있으며,

```
foo:
  field1: one
  field2: two
```

위와 같은 경우는 "foo" : { "field1" : "one", "field2" : "two" } 와 같은 딕셔너리로 구성되어 있습니다.

```
foo['field1']
foo.field1
```

위와 같이 두 가지 모두 접근 가능합니다. 모두 "one" 이라는 값을 가리킵니다. 하지만 `.` 로 접근하게 되면 파이썬 딕셔너리의 메서드나 속성 참조에 헷갈릴 수 있으므로 가능하면 대괄호와 표현하도록 합니다. 다음 아래에 나오는 것은 키워드로서 변수에 사용되지 않는 것들입니다.

add, append, asintegerratio, bitlength, capitalize, center, clear, conjugate, copy, count, decode, denominator, difference, differenceupdate, discard, encode, endswith, expandtabs, extend, find, format, fromhex, fromkeys, get, haskey, hex, imag, index, insert, intersection, intersectionupdate, isalnum, isalpha, isdecimal, isdigit, isdisjoint, isinteger, islower, isnumeric, isspace, issubset,

issuperset, istitle, isupper, items, iteritems, iterkeys, itervalues, join, keys, ljust, lower, lstrip, numerator, partition, pop, popitem, real, remove, replace, reverse, rfind, rindex, rjust, rpartition, rsplit, rstrip, setdefault, sort, split, splitlines, startswith, strip, swapcase, symmetricdifference, symmetricdifferenceupdate, title, translate, union, update, upper, values, viewitems, viewkeys, viewvalues, zfill

관리대상목록(인벤토리)에 정의된 변수

플레이북에 정의된 변수

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

위와 같이 플레이북에서 한글을 잘 사용할 수 있습니다.

Import 된 파일이나 Role에서 정의된 변수

외부의 파일 등에서 정의된 변수도 불러오기 하여 사용할 수 있습니다.

변수 사용: Jinja2

지금까지 변수를 선언하는 것을 알아보았다면 다음에는 변수를 어떻게 이용해야 될까요?

Ansible에서는 Jinja2라는 템플릿 시스템을 이용하여 플레이북에서 변수를 참조합니다. Jinja2 템플릿 시스템은 아주 많은 일을 할 수 있지만 아주 기본적인 것부터 확인해 봅니다.

예를 들어 간단한 템플릿으로...

```
My amp goes to {{ max_amp_value }}
```

중괄호 2개를 연속해서 이용해서 해당 변수와 치환합니다.

또한 다음과 같이,

```
template: src=foo.cfg.j2 dest={{ remote_install_path }}/foo.cfg
```

변수를 파일 경로에 대응하도록 할 수도 있습니다.

템플릿 내에서는 호스트 네임 스페이스안에 있는 모든 변수에 자동으로 접근 가능합니다. 그 밖에 다른 호스트에서 정의된 변수까지도 접근할 수 있습니다.

> **노트**
>
> Ansible 은 템플릿에서 Jinja2의 반복문과 조건문을 사용할 수 있지만 플레이북에서는 불가능합니다. 플레이북 자체는 순수 YAML 구분입니다. 자동 코드로 생성되거나 다른 Ansible 파일을 사용하는 다른 시스템에서 생성한 플레이북이 가능합니다.

Jinja2 필터

> **노트**
>
> 이 기능은 자주 사용되는 기능이 아닙니다.

Jinja2의 필터는 어떤 데이터를 다른 형태로 가공하는 변환 규칙의 방법입니다. Jinja2 내장 필터를 참고하시기 바랍니다.

잠시만요, YAML 좀 살펴보실께요

YAML에서 `{{ foo }}` 방식의 변수를 사용하려면 `"` 를 이용하시기 바랍니다.

```
- hosts: app_servers
  vars:
      app_path: {{ base_path }}/22
```

위의 방식은 동작하지 않을 것입니다.

```
- hosts: app_servers
  vars:
       app_path: "{{ base_path }}/22"
```

대신 이렇게 이용하시기 바랍니다.

Facts : 시스템 발견 정보

변수와는 달리 사용자가 아닌 시스템에서 발견된 정보를 가지고 변수처럼 사용할 수 있는 것이 바로 Fact 입니다.

Fact는 원격 시스템에서 발견된 정보에서 생성됩니다.

```
$ ansible hostname -m setup
```

위와 같이 실행되면 다음과 같이 생각외로 많은 정보 (변수, facts)를 리턴합니다.

> Ansible 1.4 버전에서 원격시스템, 우분투 12.04의 실행 결과 입니다.

```
"ansible_all_ipv4_addresses": [
    "REDACTED IP ADDRESS"
],
"ansible_all_ipv6_addresses": [
    "REDACTED IPV6 ADDRESS"
],
"ansible_architecture": "x86_64",
"ansible_bios_date": "09/20/2012",
"ansible_bios_version": "6.00",
"ansible_cmdline": {
    "BOOT_IMAGE": "/boot/vmlinuz-3.5.0-23-generic",
    "quiet": true,
    "ro": true,
    "root": "UUID=4195bff4-e157-4e41-8701-e93f0aec9e22",
    "splash": true
},
"ansible_date_time": {
    "date": "2013-10-02",
    "day": "02",
    "epoch": "1380756810",
    "hour": "19",
    "iso8601": "2013-10-02T23:33:30Z",
    "iso8601_micro": "2013-10-02T23:33:30.036070Z",
    "minute": "33",
    "month": "10",
    "second": "30",
    "time": "19:33:30",
    "tz": "EDT",
    "year": "2013"
},
"ansible_default_ipv4": {
    "address": "REDACTED",
    "alias": "eth0",
    "gateway": "REDACTED",
    "interface": "eth0",
    "macaddress": "REDACTED",
    "mtu": 1500,
    "netmask": "255.255.255.0",
    "network": "REDACTED",
    "type": "ether"
},
"ansible_default_ipv6": {},
"ansible_devices": {
    "fd0": {
        "holders": [],
        "host": "",
        "model": null,
        "partitions": {},
```

```
        "removable": "1",
        "rotational": "1",
        "scheduler_mode": "deadline",
        "sectors": "0",
        "sectorsize": "512",
        "size": "0.00 Bytes",
        "support_discard": "0",
        "vendor": null
    },
    "sda": {
        "holders": [],
        "host": "SCSI storage controller: LSI Logic / Symbios Logic 53c1030 PCI-X Fu
        "model": "VMware Virtual S",
        "partitions": {
            "sda1": {
                "sectors": "39843840",
                "sectorsize": 512,
                "size": "19.00 GB",
                "start": "2048"
            },
            "sda2": {
                "sectors": "2",
                "sectorsize": 512,
                "size": "1.00 KB",
                "start": "39847934"
            },
            "sda5": {
                "sectors": "2093056",
                "sectorsize": 512,
                "size": "1022.00 MB",
                "start": "39847936"
            }
        },
        "removable": "0",
        "rotational": "1",
        "scheduler_mode": "deadline",
        "sectors": "41943040",
        "sectorsize": "512",
        "size": "20.00 GB",
        "support_discard": "0",
        "vendor": "VMware,"
    },
    "sr0": {
        "holders": [],
        "host": "IDE interface: Intel Corporation 82371AB/EB/MB PIIX4 IDE (rev 01)",
        "model": "VMware IDE CDR10",
        "partitions": {},
        "removable": "1",
```

```
        "rotational": "1",
        "scheduler_mode": "deadline",
        "sectors": "2097151",
        "sectorsize": "512",
        "size": "1024.00 MB",
        "support_discard": "0",
        "vendor": "NECVMWar"
    }
},
"ansible_distribution": "Ubuntu",
"ansible_distribution_release": "precise",
"ansible_distribution_version": "12.04",
"ansible_domain": "",
"ansible_env": {
    "COLORTERM": "gnome-terminal",
    "DISPLAY": ":0",
    "HOME": "/home/mdehaan",
    "LANG": "C",
    "LESSCLOSE": "/usr/bin/lesspipe %s %s",
    "LESSOPEN": "| /usr/bin/lesspipe %s",
    "LOGNAME": "root",
    "LS_COLORS": "rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;35:do=01;35:bd=40;33;0
    "MAIL": "/var/mail/root",
    "OLDPWD": "/root/ansible/docsite",
    "PATH": "/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin",
    "PWD": "/root/ansible",
    "SHELL": "/bin/bash",
    "SHLVL": "1",
    "SUDO_COMMAND": "/bin/bash",
    "SUDO_GID": "1000",
    "SUDO_UID": "1000",
    "SUDO_USER": "mdehaan",
    "TERM": "xterm",
    "USER": "root",
    "USERNAME": "root",
    "XAUTHORITY": "/home/mdehaan/.Xauthority",
    "_": "/usr/local/bin/ansible"
},
"ansible_eth0": {
    "active": true,
    "device": "eth0",
    "ipv4": {
        "address": "REDACTED",
        "netmask": "255.255.255.0",
        "network": "REDACTED"
    },
    "ipv6": [
        {
```

```
            "address": "REDACTED",
            "prefix": "64",
            "scope": "link"
        }
    ],
    "macaddress": "REDACTED",
    "module": "e1000",
    "mtu": 1500,
    "type": "ether"
},
"ansible_form_factor": "Other",
"ansible_fqdn": "ubuntu2.example.com",
"ansible_hostname": "ubuntu2",
"ansible_interfaces": [
    "lo",
    "eth0"
],
"ansible_kernel": "3.5.0-23-generic",
"ansible_lo": {
    "active": true,
    "device": "lo",
    "ipv4": {
        "address": "127.0.0.1",
        "netmask": "255.0.0.0",
        "network": "127.0.0.0"
    },
    "ipv6": [
        {
            "address": "::1",
            "prefix": "128",
            "scope": "host"
        }
    ],
    "mtu": 16436,
    "type": "loopback"
},
"ansible_lsb": {
    "codename": "precise",
    "description": "Ubuntu 12.04.2 LTS",
    "id": "Ubuntu",
    "major_release": "12",
    "release": "12.04"
},
"ansible_machine": "x86_64",
"ansible_memfree_mb": 74,
"ansible_memtotal_mb": 991,
"ansible_mounts": [
    {
```

```
        "device": "/dev/sda1",
        "fstype": "ext4",
        "mount": "/",
        "options": "rw,errors=remount-ro",
        "size_available": 15032406016,
        "size_total": 20079898624
    }
],
"ansible_nodename": "ubuntu2.example.com",
"ansible_os_family": "Debian",
"ansible_pkg_mgr": "apt",
"ansible_processor": [
    "Intel(R) Core(TM) i7 CPU         860  @ 2.80GHz"
],
"ansible_processor_cores": 1,
"ansible_processor_count": 1,
"ansible_processor_threads_per_core": 1,
"ansible_processor_vcpus": 1,
"ansible_product_name": "VMware Virtual Platform",
"ansible_product_serial": "REDACTED",
"ansible_product_uuid": "REDACTED",
"ansible_product_version": "None",
"ansible_python_version": "2.7.3",
"ansible_selinux": false,
"ansible_ssh_host_key_dsa_public": "REDACTED KEY VALUE"
"ansible_ssh_host_key_ecdsa_public": "REDACTED KEY VALUE"
"ansible_ssh_host_key_rsa_public": "REDACTED KEY VALUE"
"ansible_swapfree_mb": 665,
"ansible_swaptotal_mb": 1021,
"ansible_system": "Linux",
"ansible_system_vendor": "VMware, Inc.",
"ansible_user_id": "root",
"ansible_userspace_architecture": "x86_64",
"ansible_userspace_bits": "64",
"ansible_virtualization_role": "guest",
"ansible_virtualization_type": "VMware"
```


위와 같은 결과에서 첫번째 하드디스크는

```
{{ ansible_devices.sda.model }}
```

와 같이 접근할 수 있습니다.

반면, 호스트이름은 (`hosta.b.com` 와 같이)

```
{{ ansible_nodename }}
```

와 같이 불러올 수 있고

```
{{ ansible_hostname }}
```

위와 같이 하여 해당 호스트명을 구해 올 수 있습니다.

Facts는 종종 조건문 혹은 템플릿에서 사용됩니다.

Facts는 또한 호스트를 동적으로 그룹화 하는데 사용되기도 합니다.

Facts 끄기

만약 이런 정보를 구해올 필요가 없거나 특별한 이유로 정보를 다 알거나 할 경우에는 이 기능을 끌 필요가 있습니다. 만약 실험적인 시스템을 테스트하거나 너무 많은 시스템을 다룰 때 이 기능을 끌 필요가 생깁니다.

```
- hosts: whatever
  gather_facts: no
```

플레이북의 플레이 안에 위와 같이 지정하면 됩니다.

로컬 Facts (Facts.d)

버전 1.3 이후.

Facts는 모통 setup 모듈에 의하여 원격 시스템의 정보를 구해오는 것입니다. 사용자는 개별 facts 모듈 API 가이드에 따라 작성할 수 있습니다. 그러나 이런 fact 모듈을 작성하지 않고 시스템 또는 데이터 정보를 지정할 수 있는 간단한 방법이 있을까요?

> **노트**
>
> **로컬** 이라는 의미가 원격시스템의 반대되는 로컬이라는 의미라기 보다는 원격 시스템 입장에서의 로컬 정보 설정으로 보아야 합니다.

만약 원격 관리 시스템이 `/etc/ansible/facts.d` 라는 디렉터리가 있고 그 아래에 `.fact` 로 끝나는 파일이 있는데 이 파일이 JSON, INI 또는 JSON을 리턴하는 실행파일이라면 이것은 Ansible에서 `로컬 fact` 라고 ㅁ합니다.

예를 들어, `/etc/ansible/facts.d/preferences.fact` 파일에

```
[general]
asdf=1
bar=2
```

라는 파일이 있다면

```
$ ansible <hostname> -m setup -a "filter=ansible_local"
```

라고 명령을 주면

```
"ansible_local": {
        "preferences": {
            "general": {
                "asdf" : "1",
                "bar"  : "2"
            }
        }
 }
```

라고 JSON 결과가 나옵니다. 또한 `template/playbook` 에서 이 데이터를 접근가능합니다.

```
{{ ansible_local.preferences.general.asdf }}
```

로컬 네임스페이스는 시스템 fact 또는 플레이북에서 정의된 변수를 덮어쓰는 것을 막습니다.

Ansible 버전

```
"ansible_version": {
    "full": "2.0.0.2",
    "major": 2,
    "minor": 0,
    "revision": 0,
    "string": "2.0.0.2"
}
```

Fact 캐싱

버전 1.8 이후.

한 서버에서 다른 서버에 있는 변수를 참조할 수 있습니다.

```
{{ hostvars['asdf.example.com']['ansible_os_family'] }}
```

만약 `Fact 캐싱` 이 활성화되어 있지 않다면 이런 역할을 하기 위하여 해당 `asdf.example.com` 호스트가 현재 플레이에 있으며 우선순위에 따라 미리 수행되어 있어야 합니다.

그렇지 않아도 가능하게 하기 위하여 플레이북이 실행되면서 각 호스트의 Fact가 캐싱되도록 합니다. 이것은 수동으로 설정에서 활성화시켜야 합니다. 예를 들어 수천개 이상 관리하는 매우 큰 시스템에서 이 기능을 활성화시키면 온갖 Fact를 저장하기 위하여 많은 자원을 소모할 것입니다.

Fact 캐싱이 활성화 되어 있으면 현재 `/usr/bin/ansible-playbook` 이 실행되고 있지 않아도 어떤 그룹에 있는 머신이 다른 그룹에 있는 머신의 변수를 참조할 수 있습니다.

캐싱 방법은 redis 또는 jsonfile 입니다.

Fact 캐싱을 활성화 하려면 `ansible.cfg` 파일을

```
[defaults]
gathering = smart
fact_caching = redis
fact_caching_timeout = 86400
# seconds
```

와 같이 지정합니다.

redis 서버를 이용하려면

```
$ sudo apt-get install redis
$ sudo service redis start
$ sudo pip install redis
```

와 같은 식으로 시스템에 redis 관련 패키지를 설치해야 합니다.

json 파일을 이용한 캐싱을 하려고 한다면, `ansible.cfg` 파일에서

```
[defaults]
gathering = smart
fact_caching = jsonfile
fact_caching_connection = /path/to/cachedir
fact_caching_timeout = 86400
# seconds
```

와 같이 지정합니다.

등록 변수

변수의 중요한 사용방법 중에 하나는 특정 명령어를 수행하고 난 결과를 변수로 저장하는 것입니다. 결과는 모듈마다 틀릴 수 있으므로 `-v` 옵션을 이용하여 플레이북을 기동시키면 좀 더 자세한 결과를 살펴볼 수 있습니다.

현재 실행되고 있는 작업의 값은 변수에 저장되고 나중에 사용될 수 있습니다.

```
- hosts: web_servers

  tasks:

     - shell: /usr/bin/foo
       register: foo_result
       ignore_errors: True

     - shell: /usr/bin/bar
       when: foo_result.rc == 5
```

등록 변수는 플레이북이 실행되는 내내 유효하며 이것은 `Fact` 도 동일합니다.

반복문에서 `register` 를 사용한다면 `results` 속성이 모듈 실행 결과의 목록을 담고 있습니다.

> **노트**
>
> 만약 작업이 실패하거나 건너뛰었다면 해당 변수는 실패 혹은 건너뛴 상태를 담고 있습니다.

좀 더 복잡한 변수 접근

네트워킹 정보처럼 제공된 fact는 중첩된 데이터 구조로 되어 있습니다.

```
{{ ansible_eth0["ipv4"]["address"] }}
```

또는

```
{{ ansible_eth0.ipv4.address }}
```

와 같이 해당 fact 정보를 참조할 수 있습니다.

매직 변수, 다른 호스트 정보 접근

아무런 변수를 지정하지 않았더라도 Ansible이 자동으로 설정하는 변수들이 있습니다. 이 중, `hostvars` , `group_names` , `groups` , 그리고 `environment` 가 있습니다. 이런 정의 변수 이름을 사용자가 사용하면 안 됩니다.

`hostvars` 는 fact를 포함하여 다른 호스트의 변수를 참조하는 변수입니다.

만약 데이터베이스 서버가 다른 노드에 있는 `fact` 의 값 또는 인벤토리 변수를 참조한다고 할 때, 템플릿 또는 action 라인에 다음과 같이,

```
{{ hostvars['test.example.com']['ansible_distribution'] }}
```

참조를 합니다.

`group_names` 는 현재 실행되고 있는 호스트가 속해있는 모든 그룹의 목록을 가지고 있습니다.

```
{% if 'webserver' in group_names %}
   # some part of a configuration file that only applies to webservers
{% endif %}
```

위와 같이 Jinja2 문법에 따른 템플릿을 구성할 수 있습니다.

`groups` 는 인벤토리에 있는 모드 그룹 (그리고 호스트) 를 담고 있는 목록 변수입니다.

```
{% for host in groups['app_servers'] %}
   # something that applies to all app servers.
{% endfor %}
```

위와 같이 모든 `app_servers` 그룹에 있는 모든 호스트를 열거할 수 있습니다.

한걸음 더 나아가 해당 그룹에 있는 모든 호스트의 IP 주소를 구하려면,

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

와 같이 하면 됩니다.

`inventory_hostname` 변수는 인벤토리 호스트 파일에 정의된 호스트이름을 담고 있습니다. 이것은 발견된 호스트 이름 (`ansible_hostname`)이 아니라 선언된 호스트이름을 이용할 때 유용합니다. 만약 긴 FQDN을 가지고 있다면 `inventory_hostname_short` 을 이용해 FQDN의 첫번째 부분 (a.b.c 에서 a) 만 참조할 수 있습니다.

`play_hosts` 는 현재 실행되고 있는 플레이에서 접속가능한 호스트 목록을 담고 있습니다. 이것은 해당 머신의 로드 밸런싱 등을 할 때 유용하게 사용될 수 있습니다.

`inventory_dir` 은 인벤토리 파일을 담고 있는 폴더를 알려줍니다. 반면 `inventory_file` 은 인벤토리 파일의 전체 경로를 알려줍니다.

`playbook_dir` 은 플레이북 파일의 베이스 디렉터리를 담고 있습니다.

`role_path` 는 현재 role의 경로명을 나타내는데 role 에 있을 때만 동작합니다.

마지막으로 `ansible_check_mode` (2.1에서 추가)는 Ansible이 --check 옵션으로 실행되었을 때 True가 되는 블리언 변수입니다.

변수 파일

플레이북을 버전관리한다는 생각은 아주 뛰어난 생각입니다만 플레이북은 누구나 볼 수 있지만 그 안에 있는 내용 일부는 누구나 볼 수 없게 하고 싶을 수 있습니다. 비슷하게 메인 플레이북과 별도로 다른 파일에 정보를 담고 있을 수 있습니다.

```
---

- hosts: all
  remote_user: root
  vars:
    favcolor: blue
  vars_files:
    - /vars/external_vars.yml

  tasks:

  - name: this is just a placeholder
    command: /bin/echo foo
```

`vars_files` 에 지정할 수 있는데, 그 내용은,

```
---
# in the above example, this would be vars/external_vars.yml
somevar: somevalue
password: magic
```

와 같습니다.

명령행에 변수 지정

`vars_prompt` 및 `vars_files` 과 더불어 Ansible 명령행에 다음과 같이 변수를 지정할 수 있습니다.

```
$ ansible-playbook release.yml --extra-vars "version=1.23.45 other_variable=foo"
```

1.2 이후 부터는 JSON 형식으로 `--extra-vars` 를 지정해도 됩니다.

```
--extra-vars '{"pacman":"mrs","ghosts":["inky","pinky","clyde","sue"]}'
```

> **노트**
>
> 값은 키=값 과 같은 문법이므로 문자열로 지정하는 반면 JSON 포맷은 불리언, 정수, 실수 등을 이용할 수 있습니다.

버전 1.3 이후로는 `@` 를 붙여 파일을 지정할 수 있습니다.

```
--extra-vars "@some_file.json"
```

변수 우선 순위 : 어느 곳에 변수를 지정해야 하는가?

우선 동일한 이름의 변수를 사용하지 않습니다. 그러나 동일한 이름으로 사용되는 변수가 우선순위에 의해 어디에 사용된 변수를 더 우선적으로 사용하는가가 있습니다. 그것이 변수 사용 우선순위 입니다.

버전 1.x 에서는 다음과 같이 변수 우선순위가 높아집니다.

- `role defaults` 에 정의된 변수
- 인벤토리에 정의된 변수
- 시스템에서 발견된 facts
- 기타 정의된 변수 (명령행 스위치, 플레이의 변수, 포함된 변수, role 등)
- 연결 변수 (`ansible_user` 등)
- 명령행에 지정한 `-e` 변수 (항상 최우선)

버전 2.x에서는 좀 더 상세히 우선순위가 나뉩니다.

- role 디폴트
- 인벤토리 변수
- 인벤토리 group_vars
- 인벤토리 host_vars
- 플레이북 group_vars
- 플레이북 host_vars
- 호스트 facts
- 플레이 변수
- 플레이 vars_files
- 등록 변수
- set_facts
- role과 포함된 변수
- block 변수 (블럭에 있는 태스크)
- 태스크 변수
- 명령행에 지정한 `-e` 변수 (항상 최우선)

또 다른 예로서 연결 변수는 다른 변수를 우선시 합니다.

```
ansible_ssh_user will override `-u <user>` and `remote_user: <user>`
```

변수 적용 범위

다음과 같은 3가지 적용 범위를 갖습니다.

- 글로벌 : 설정, 환경변수 명령줄 옵션 등으로 설정된 변수
- 플레이 : 각각의 플레이와 포함한 구조체, 변수, 포함 변수, role defaults 와 변수 등
- 호스트 : 호스트에 연관된 변수, facts, 등록 태스크 결과

변수 사용예

제일 먼저 그룹 변수의 사용예를 살펴보겠습니다.

`group_vars/all` 에 그룹 변수를 지정합니다.

```
---
# file: /etc/ansible/group_vars/all
# this is the site wide default
ntp_server: default-time.example.com
```

지역 정보는 `group_vars/region` 변수에 정의됩니다. 만약 만약 이 그룹이 `all` 그룹에도 있었다면 거기에 우선하여 설정됩니다.

```
---
# file: /etc/ansible/group_vars/boston
ntp_server: boston-time.example.com
```

만약 특별한 이유로 특정 호스트가 더 우선하여 해당 정보를 지정해야 한다면

```
---
# file: /etc/ansible/host_vars/xyz.boston.example.com
ntp_server: override.example.com
```

와 같이 설정합니다.

> 기억할 내용: 자식 그룹은 부모 그룹을 우선하고 호스트는 항상 그룹을 우선한다.

다음은 role 변수의 우선순위에 대한 것입니다.

```
---
# file: roles/x/defaults/main.yml
# if not overridden in inventory or as a parameter, this is the value that will be u
http_port: 80
```

`roles/x/defaults/main.yml` 에 정의되어 되어 있다면

```
---
# file: roles/x/vars/main.yml
# this will absolutely be used in this role
http_port: 80
```

이 변수가 더 우선 순위를 갖습니다.

role을 사용하다가 정의된 변수 대신 다른 것을 이용하려면,

```
roles:
   - { role: apache, http_port: 8080 }
```

이라고 합니다.

혹은,

```
roles:
   - { role: app_user, name: Ian    }
   - { role: app_user, name: Terry  }
   - { role: app_user, name: Graham }
   - { role: app_user, name: John   }
```

와 같이 사용할 수 있습니다.

Jinja2 필터

Jinja2 필터는 템플릿에서 데이터를 어떤 형태에서 다른 형태로 바꾸는 작업입니다. 이미 많은 내장 함수를 가지고 있습니다.

이런 필터 기능은 로컬 데이터를 다루는 Ansible의 컨트롤러에서 수행되는 것이며 타겟 머신의 태스크에서 이루어 지는 것이 아닙니다.

데이터 포맷을 위한 필터

어떤 결과를 다른 형태로 변경하는 것입니다. 디버깅을 할 때 유용한 경우가 있습니다.

```
{{ some_variable | to_json }}
{{ some_variable | to_yaml }}
```

또는 사람이 읽기 쉬운 형태로 출력합니다.

```
{{ some_variable | to_nice_json }}
{{ some_variable | to_nice_yaml }}
```

파이썬의 pprint 모듈처럼 indent를 줄 수도 있습니다. (버전 2.2 이후)

```
{{ some_variable | to_nice_json(indent=2) }}
{{ some_variable | to_nice_yaml(indent=8) }}
```

이제는 반대로 이미 포맷된 것에서 읽어 오는 경우도 있습니다.

```
{{ some_variable | from_json }}
{{ some_variable | from_yaml }}
```

예를 들어,

```
tasks:
  - shell: cat /some/path/to/file.json
    register: result

  - set_fact: myvar="{{ result.stdout | from_json }}"
```

어떤 변수가 정의되어야만 하도록 강제함

어떤 변수가 정의되어 있지 않다면 ansible의 ansible.cfg에 정의되어 있는 디폴트 행동을 따르게 되어 있는데 이를 끌 수 있습니다.

다음은 그 기능이 꺼지고 꼭 해당 변수가 존재하는지 체크하도록 하는 것입니다.

```
{{ variable | mandatory }}
```

만약 해당 변수가 정의되어 있지 않다면 템플릿이 동작하면서 오류가 발생합니다.

정의 되지 않은 변수의 디폴트 값 정의

Jinja2 는 `default` 필터를 제공하는데 해당 변수가 정의되지 않을 경우 디폴트 값을 갖게 됩니다.

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

정의되지 않은 변수와 패러미터의 생략

버전 1.8 부터 `omit` 이라는 특별 변수를 사용하여 변수나 모듈 패러미터를 생략하기위한 디폴트 필터를 사용할 수 있습니다.

```
- name: touch files with an optional mode
  file: dest={{item.path}} state=touch mode={{item.mode|default(omit)}}
  with_items:
    - path: /tmp/foo
    - path: /tmp/bar
    - path: /tmp/baz
      mode: "0444"
```

파일경로 `path` 가 3개 있는데 앞에 2개는 시스템에서 제공하는 umask에 따라 touch 되고 마지막 파일경로 `/tmp/baz` 는 시스템 디폴트 umask와 상관없이 0444 로 설정됩니다.

> **노트**
>
> 만약 `default(omit)` 필터 다음에 또다른 필터를 적용하려면 ***{{ foo | default(None) | some_filter or omit }}*** 와 같이 적용하면 될거라고 생각할 겁니다. 하지만 default 를 파이썬의 None에 적용하면 필터가 오류가 발생하므로 원하는 결과가 안나올 수 있습니다. 이런 경우에는 시행착오를 거쳐 해당 필터 `체이닝(chaining)` 테스트를 해 봐야 합니다.

List 필터

버전 1.8 이후.

해당 목록엣 제일 작은 값을 구하기 위하여,

```
{{ list1 | min }}
```

반대로 최대값을 구하려면,

```
{{ [3, 4, 2] | max }}
```

집합(SET) 이론 필터

버전 1.4 이후.

해당 목록에서 중복 항목을 제외한 set을 구하려면,

```
{{ list1 | unique }}
```

두 목록의 합집합을 구하려면,

```
{{ list1 | union(list2) }}
```

교집합을 구하려면,

```
{{ list1 | intersect(list2) }}
```

차집합을 구하려면,

```
{{ list1 | difference(list2) }}
```

합집합에서 교집합을 제외한 부분 (symmetric difference)을 구하려면,

```
{{ list1 | symmetric_difference(list2) }}
```

랜덤 숫자 필터

버전 1.6 이후.

어떤 목록에서 임의의 항목을 얻을 경우,

```
{{ ['a','b','c']|random }} => 'c'
```

0~n 까지의 임의의 숫자를 얻을 경우,

```
{{ 59 |random}} * * * * root /script/from/cron
```

0부터 100사이의 임의의 값을 구하는데 10 단위의 값을 구할 경우

```
{{ 100 |random(step=10) }}  => 70
```

1 부터 100 까지의 임의의 값을 구하는데 10 씩 증가하는 값을 구할 경우

```
{{ 100 |random(1, 10) }}    => 31
{{ 100 |random(start=1, step=10) }}    => 51
```

섞기 필터

버전 1.8 이후.

어떤 목록을 임의의 순서로 항목을 뒤섞을 경우,

```
{{ ['a','b','c']|shuffle }} => ['c','a','b']
{{ ['a','b','c']|shuffle }} => ['b','c','a']
```

Math

버전 1.9 이후.

log 값 구하기,

```
{{ myvar | log }}
```

10 base log 값 구하기,

```
{{ myvar | log(10) }}
```

해당 값의 제곱 또는 5제곱을 구할 때,

```
{{ myvar | pow(2) }}
{{ myvar | pow(5) }}
```

어떤 값의 제곱근, 5제곱근을 구할 때,

```
{{ myvar | root }}
{{ myvar | root(5) }}
```

IP 주소 필터

버전 1.9 이후.

해당 값이 IP 주소인가 조사.

```
{{ myvar | ipaddr }}
```

ip 버전 4 또는 6을 지정할 경우,

```
{{ myvar | ipv4 }}
{{ myvar | ipv6 }}
```

CIDR에서 특정 정보를 구할 경우,

```
{{ '192.0.2.1/24' | ipaddr('address') }}
```

더 자세한 ipaddr 기능은 Jinja2 ipaddr() 필터를 참조하십시오.

해쉬 필터

버전 1.9 이후.

특정 문자열의 sha1 해쉬 값을 구하려면,

```
{{ 'test1'|hash('sha1') }}
```

md5 해쉬는,

```
{{ 'test1'|hash('md5') }}
```

체크썸 문자열을 구하려면,

```
{{ 'test2'|checksum }}
```

기타 다른 해쉬 (플랫폼 의존적)

```
{{ 'test2'|hash('blowfish') }}
```

(random salt)를 이용한 sha512 해쉬인 경우,

```
{{ 'passwordsaresecret'|password_hash('sha512') }}
```

특정 sault를 이용한 sha256 해쉬를 구할 경우,

```
{{ 'secretpassword'|password_hash('sha256', 'mysecretsalt') }}
```

제공되는 해쉬 종류는 ansible을 동작시키는 마스터 시스템에 달려있고, 암호를 위해서는 `password_hash` 에서 지정한 해쉬 라이브러리를 따릅니다.

해쉬와 딕셔너리 조합

2.0 이후.

`combine` 필터는 해쉬를 병합시킵니다. 예를 들어, 다음에서

```
{{ {'a':1, 'b':2}|combine({'b':3}) }}
```

위의 결과는,

```
{'a':1, 'b':3}
```

가 됩니다.

또한 *`recursive=True`* 패러미터를 통하여 사전의 내포 항목까지 찾아들어가 병합시킵니다.

```
{{ {'a':{'foo':1, 'bar':2}, 'b':2}|combine({'a':{'bar':3, 'baz':4}}, recursive=True)
```

위의 결과는,

```
{'a':{'foo':1, 'bar':3, 'baz':4}, 'b':2}
```

또한 combine 필터는 여러 인자를 받을 수 있습니다.

```
{{ a|combine(b, c, d) }}
```

컨테이너에서 값 추출

2.1 이후.

`extract` 필터는 해쉬나 어레이 같은 컨테이너에서 일련의 값을 뽑아 매핑하는데 사용됩니다.

```
{{ [0,2]|map('extract', ['x','y','z'])|list }}
{{ ['x','y']|map('extract', {'x': 42, 'y': 31})|list }}
```

위의 결과는

```
['x', 'z']
[42, 31]
```

해당 필터는 또다른 인자를 취할 수 있는데,

```
{{ groups['x']|map('extract', hostvars, 'ec2_ip_address')|list }}
```

이것은 그룹 `x` 에서 호스트 목록을 얻은 다음 ***hostvars*** 에서 찾은 다음 *ec2_ip_address* 인 것을 찾습니다. 마지막 결과는 그룹 `x` 에 있는 호스트의 IP 주소 목록을 결과로 얻습니다.

이 필터의 세번째 인자는 목록이 될 수 있는데,

```
{{ ['a']|map('extract', b, ['x','y'])|list }}
```

이 결과는 ***b['a']['x']['y']*** 의 값을 갖는 목록을 리턴합니다.

코멘트 필터

버전 2.0 이후.

`comment` 필터는 해당 문자열을 코멘트 형식으로 출력합니다.

```
{{ "Plain style (default)" | comment }}
```

위의 결과는

```
#
# Plain style (default)
#
```

코멘트 스타일을 C(`//...`), C 블럭(`/*...*/`), Erlang(`%...`) 그리고 XML(`<!--...-->`) 와 같이 줄 수 있습니다.

```
{{ "C style" | comment('c') }}
{{ "C block style" | comment('cblock') }}
{{ "Erlang style" | comment('erlang') }}
{{ "XML style" | comment('xml') }}
```

그리고 코멘트 스타일을 별도로 지정할 수 있는데,

```
{{ "Custom style" | comment('plain', prefix='#######\n#', postfix='#\n#######\n   ##
```

위의 결과는

```
#######
#
# Custom style
#
#######
   ###
    #
```

예를 들어 `ansible.cfg` 파일에 `ansible_managed` 변수가 다음과 같이,

```
[defaults]

ansible_managed = This file is managed by Ansible.%n
  template: {file}
  date: %Y-%m-%d %H:%M:%S
  user: {uid}
  host: {host}
```

정의 되어 있었다면

```
{{ ansible_managed | comment }}
```

위와 같이 코멘트 필터를 적용한 결과는,

```
#
# This file is managed by Ansible.
#
# template: /home/ansible/env/dev/ansible_managed/roles/role1/templates/test.j2
# date: 2015-09-10 11:02:58
# user: ansible
# host: myhost
#
```

이 됩니다.

다른 유용한 필터들

쉘에서 따옴표를 추가하려면,

```
- shell: echo {{ string_value | quote }}
```

어떤 값이 true 면 아니면... 과 같은 구문을 적용하려면 (1.9 이후)

```
{{ (name == "John") | ternary('Mr','Ms') }}
```

목록을 하나의 문자열로 join

```
{{ list | join(" ") }}
```

파일의 경로명 `/etc/asdf/foo.txt` 에서 파일명 `foo.txt` 만 추출하려면

```
{{ path | basename }}
```

윈도우 시스템에서 파일명만 추출하려면, (2.0 이후)

```
{{ path | win_basename }}
```

윈도우 시스템에서 드라이브명만 추출하려면, (2.0이후)

```
{{ path | win_splitdrive }}
```

드라이브명의 첫 글자만 구하려면,

```
{{ path | win_splitdrive | first }}
```

드라이브명 없이 나머지 경로만 구하려면,

```
{{ path | win_splitdrive | last }}
```

파일 경로에서 디렉터리만 구하려면,

```
{{ path | dirname }}
```

윈도우 시스템에서 디렉터리만 구하려면 (2.0 이후)

```
{{ path | win_dirname }}
```

만약 틸더 `~` 문자를 가진 경로를 확장시키려면 (버전 1.5 이후)

```
{{ path | expanduser }}
```

링크의 실제 경로를 구하려면 (버전 1.8 이후)

```
{{ path | realpath }}
```

주어진 경로의 상대 경로를 구하려면 (버전 1.7 이후) `yaml {{ path | relpath('/etc') }}`

파일이름에서 확장자를 분리하려면 (버전 2.0 이후)

```
# with path == 'nginx.conf' the return would be ('nginx', '.conf')
{{ path | splitext }}
```

Base64 인코딩 문자열을 구하려면,

```
{{ encoded | b64decode }}
{{ decoded | b64encode }}
```

문자열에서 UUID를 생성하려면, (버전 1.9 이후)

```
{{ hostname | to_uuid }}
```

필요 시 cast 기능을 이용할 수 있습니다.

```
- debug: msg=test
  when: some_string_value | bool
```

버전 1.6 이후 ***regex_replace*** 필터를 이용하여 정규식 치환을 적용할 수 있습니다.

```
# convert "ansible" to "able"
{{ 'ansible' | regex_replace('^a.*i(.*)$', 'a\\1') }}

# convert "foobar" to "bar"
{{ 'foobar' | regex_replace('^f.*o(.*)$', '\\1') }}

# convert "localhost:80" to "localhost, 80" using named groups
{{ 'localhost:80' | regex_replace('^(?P<host>.+):(?P<port>\\d+)$', '\\g<host>, \\g<p
```

> **노트**
>
> 버전 2.0 이전에 YAML 인자에서 변수와 함께 `regex_replace` 필터가 사용되었다면 후방참조 (backreference) (예, `\\1`) 은 이스케잎 처리가 되어 `\\` 대신 `\\\\` 로 사용해야 합니다.

버전 2.0 이후.

정규식 안에서 특수 문자를 사용하려면 `regex_escape` 필터를 사용합니다.

```
# convert '^f.*o(.*)$' to '\^f\.\*o\(\.\*\)\$'
{{ '^f.*o(.*)$' | regex_escape() }}
```

복잡한 변수 목록에서 각 항목의 속성을 이용하려면 `map` 필터를 사용합니다.

```
# get a comma-separated list of the mount points (e.g. "/,/mnt/stuff") on a host
{{ ansible_mounts|map(attribute='mount')|join(',') }}
```

문자열에서 date 객체를 이용하려면 (2.2 이후)

```
# get amount of seconds between two dates, default date format is %Y-%d-%m %H:%M:%S
{{ (("2016-08-04 20:00:12"|to_datetime) - ("2015-10-06"|to_datetime('%Y-%d-%m'))).se
```

Jinja2 테스트 관련 필터

Jinja2의 테스트는 템플릿을 돌려 True 또는 False 결과를 리턴하는가를 보고 판단합니다. Jinja2의 내장 테스트 문서를 참조합니다. 테스트 하는 것은 필터를 이용하는 것과 동일하지만 C(`map()`) 또는 C(`select()`) 와 같이 목록에서 항목을 선택하는 것과 같은 목록 처리 필터에도 사용될 수 있습니다.

필터와 마찬가지로 테스트도 로컬 데이터를 테스트하는 Ansible 컨트롤러에서 수행되며, 원격 대상 태스크에서 수행되지는 않습니다.

문자열 테스트

서브 문자열 혹은 정규식 등으로 문자열을 매칭하기 위하여 `match` 또는 `search` 필터를 사용합니다.

```
vars:
  url: "http://example.com/users/foo/resources/bar"

tasks:
    - shell: "msg='matched pattern 1'"
      when: url | match("http://example.com/users/.*/resources/.*")

    - debug: "msg='matched pattern 2'"
      when: url | search("/users/.*/resources/.*")

    - debug: "msg='matched pattern 3'"
      when: url | search("/users/")
```

`match` 는 전체 문자열에 대한 매치를 하는 반면 `search` 는 부분 매치를 합니다.

버전 비교 터스트

버전 1.6 이후.

`ansible_distribution_version` 버전이 `12.04` 보다 같거나 큰 가를 비교하기 위하여 ***version_compare*** 필터를 사용합니다.

```
{{ ansible_distribution_version | version_compare('12.04', '>=') }}
```

`version_compare` 필터는 다음과 같은 비교 연산자를 사용할 수 있습니다.

```
<, lt, <=, le, >, gt, >=, ge, ==, =, eq, !=, <>, ne
```

그룹 포함 테스트

어떤 그룹이 다른 그룹의 서브셋 또는 수퍼셋 인가를 체크하기 위한 `issubset` , `issuperset` 필터가 있습니다.

```
vars:
    a: [1,2,3,4,5]
    b: [2,3]
tasks:
    - debug: msg="A includes B"
      when: a|issuperset(b)

    - debug: msg="B is included in A"
      when: b|issubset(a)
```

경로 테스트

```
- debug: msg="path is a directory"
  when: mypath|isdir

- debug: msg="path is a file"
  when: mypath|is_file

- debug: msg="path is a symlink"
  when: mypath|is_link

- debug: msg="path already exists"
  when: mypath|exists

- debug: msg="path is {{ (mypath|is_abs)|ternary('absolute','relative')}}"

- debug: msg="path is the same file as path2"
  when: mypath|samefile(path2)

- debug: msg="path is a mount"
  when: mypath|ismount
```

테스크 결과 테스트

```
tasks:

  - shell: /usr/bin/foo
    register: result
    ignore_errors: True

  - debug: msg="it failed"
    when: result|failed

  # in most cases you'll want a handler, but if you want to do something right now,
  - debug: msg="it changed"
    when: result|changed

  - debug: msg="it succeeded in Ansible >= 2.1"
    when: result|succeeded

  - debug: msg="it succeeded"
    when: result|success

  - debug: msg="it was skipped"
    when: result|skipped
```

> 노트
>
> 버전 2.1 이후 부터는 **success**, **failure**, **change**, 그리고 **skip** 을 결과에 맞게 사용할 수 있습니다.

조건식

종종 플레이 되는 것은 변수, fact (원격 시스템에 얻은 정보) 또는 이전 태스크 수행 결과 등에 따라 달라질 수 있습니다. 어떤 경우 변수의 값은 다른 변수에 의존합니다. Ansible의 실행 제어를 위한 다양한 방법이 존재합니다.

When 문장

때로는 특정 호스트에 대해서 작업을 건너 뛸 필요가 있을 수 있습니다. 만약 특정 패키지가 없다거나 해당 운영체제가 지원 버전보다 오래되었거나 아니면 파일시스템이 꽉 차서 비우는 등의 특정 작업을 수행해야 하는 등 때문에 건너 뛸 필요가 생깁니다.

이런 경우 `when` 문장을 사용하면 됩니다.

```yaml
tasks:
  - name: "shut down Debian flavored systems"
    command: /sbin/shutdown -t now
    when: ansible_os_family == "Debian"
    # note that Ansible facts and vars like ansible_os_family can be used
    # directly in conditionals without double curly braces
```

일련의 조건식을 주기 위하여 다음과 같이 괄호를 이용할 수도 있습니다.

```yaml
tasks:
  - name: "shut down CentOS 6 and Debian 7 systems"
    command: /sbin/shutdown -t now
    when: (ansible_distribution == "CentOS" and ansible_distribution_major_version =
          (ansible_distribution == "Debian" and ansible_distribution_major_version =
```

만약 여러 조건식이 모두 `and` 조건이라면 목록으로 기술할 수 있습니다.

```yaml
tasks:
  - name: "shut down CentOS 6 systems"
    command: /sbin/shutdown -t now
    when:
      - ansible_distribution == "CentOS"
      - ansible_distribution_major_version == "6"
```

when 문장에 Jinja2 필터를 사용할 수 있습니다.

```yaml
tasks:
  - command: /bin/false
    register: result
    ignore_errors: True

  - command: /bin/something
    when: result|failed

  # In older versions of ansible use |success, now both are valid but succeeded uses
  - command: /bin/something_else
    when: result|succeeded

  - command: /bin/still/something_else
    when: result|skipped
```

> 노트

> 버전 2.1부터 success, succeeded (fail/failed 등)작업이 업데이트 되었습니다.

`register` 문에는 좀 더 다른 의미를 갖습니다.

다시 이전 fact 를 상기하며,

```
$ ansible hostname.example.com -m setup
```

> *팁:* 때로는 문자열 변수에서 산술식 비교를 할 필요가 있습니다. 그런 경우에는 다음과 같이,

```
tasks:
  - shell: echo "only on Red Hat 6, derivatives, and later"
    when: ansible_os_family == "RedHat" and ansible_lsb.major_release|int >= 6
```

> **노트**
>
> 위의 예제는 관리 대상 호스트에서 `ansible_lsb.major_release` fact 를 구하기 위하여 `lsb_release` 패키지가 필요합니다.

```
vars:
  epic: true
```

위와 같이 플레이북이나 인벤토리에 정의된 변수는 모두 `when` 구문에서 사용가능합니다.

```
tasks:
    - shell: echo "This certainly is epic!"
      when: epic
```

또는,

```
tasks:
    - shell: echo "This certainly isn't epic!"
      when: not epic
```

만약 요구된 변수가 설정되지 않았다면 Jinja2의 ***defined*** 테스트를 이용하여 건너뛸 수 있습니다.

예를 들어,

```
tasks:
    - shell: echo "I've got '{{ foo }}' and am not afraid to use it!"
      when: foo is defined

    - fail: msg="Bailing out. this play requires 'bar'"
      when: bar is undefined
```

이 방법은 변수를 외부에서 불러와서 사용할 때 특히 유용합니다.

반복문과 조건식

`when` 구문을 `with_items` 과 같이 사용할 수 있습니다. `when` 구문은 for each 항목과 무관하게 동작함을 명심해야 합니다.

```
tasks:
    - command: echo {{ item }}
      with_items: [ 0, 2, 4, 6, 8, 10 ]
      when: item > 5
```

만약 선언되어 있지 않은 변수에 디폴트를 지정하고 싶다면,

```
- command: echo {{ item }}
  with_items: "{{ mylist|default([]) }}"
  when: item > 5
```

목록이 아니고 딕셔너리인 경우 `with_dict` 를 사용합니다.

```
- command: echo {{ item.key }}
  with_dict: "{{ mydict|default({}) }}"
  when: item.value > 5
```

커스텀 Facts에서 로딩

개발중인 모듈에서 쉽게 fact를 제공할 수 있습니다.

```
tasks:
    - name: gather site specific fact data
      action: site_facts
    - command: /usr/bin/thingy
      when: my_custom_fact_just_retrieved_from_the_remote_system == '1234'
```

role 과 include 에 `when` 구문 적용하기

만약 동일한 조건식을 여러 태스크에서 동일하게 사용한다면 다음과 같이 `include` 구문을 태스크에 조건적으로 첨부할 수 있습니다. 모든 태스크는 수행되지만 조건식은 각각의 태스크 혹은 모든 태스크에 적용됩니다.

```
- include: tasks/sometasks.yml
  when: "'reticulating splines' in output"
```

> **노트**
>
> 버전 2.0 이전에는 include 태스크에는 적용되었지만 플레이북에는 적용되지 않았었습니다.

role 에는,

```
- hosts: webservers
  roles:
     - { role: debian_stock_config, when: ansible_os_family == 'Debian' }
```

와 같이 사용가능합니다.

조건부 Import

> **노트**
>
> 이 방법은 자주사용하는 것은 아닙니다.

때로는 어떤 조건에 따라 플레이북에서 다르게 동작할 필요가 있습니다. 멀티 플랫폼이나 OS에서 동작하는 하나의 플레이북을 예를 들 수 있습니다.

CentOS와 Debian 에서 다른 이름으로 되어 있는 아파치 패키지를 예를 들 수 있는데 다음과 같이 플레이북을 이용할 수 있습니다.

```
---
- hosts: all
  remote_user: root
  vars_files:
    - "vars/common.yml"
    - [ "vars/{{ ansible_os_family }}.yml", "vars/os_defaults.yml" ]
  tasks:
  - name: make sure apache is running
    service: name={{ apache }} state=running
```

> **노트**

> `ansible_os_family` 변수는 ***vars_files*** 에 정의된 파일이름의 목록으로 해석됩니다.

CentOS를 위한 YAML은 다음 처럼,

```
---
# for vars/CentOS.yml
apache: httpd
somethingelse: 42
```

구성할 수 있습니다. 유사하게 Debian 은

```
---
# for vars/Debian.yml
apache: apache2
somethingelse: 42
```

구성될 수 있겠네요. (역자주)

이런 조건부 import 기능을 사용하려면 `factor` 또는 `ohai` 모듈이 플레이북 실행에 앞서 설치되어 있어야 하는데 다음과 같은 명령을 미리 실행해 놓을 수 있습니다.

```
# for facter
ansible -m yum -a "pkg=facter state=present"
ansible -m yum -a "pkg=ruby-json state=present"

# for ohai
ansible -m yum -a "pkg=ohai state=present"
```

위와 같이 구성하는 장점은 추적 포인트를 최소화 한다는 것입니다. 변수를 태스크에서 분리시켜 플레이북이 여러 중첩된 if 코드 등으로 뒤죽 박죽하는 것을 막을 수 있습니다.

변수 기반의 파일과 템플릿 선택

> **노트**
>
> 이 기능은 자주 사용하지 않습니다.

때로는 복사하려는 설정 파일이나 사용하려는 템플릿이 변수에 따라 다를 수 있습니다.

다음의 예는 CentOS와 Debian과 같이 서로 다른 시스템에 따라 템플릿에서 설정파일을 달리 사용할 수 있는 방법을 보여줍니다.

```
- name: template a file
  template: src={{ item }} dest=/etc/myapp/foo.conf
  with_first_found:
    - files:
       - {{ ansible_distribution }}.conf
       - default.conf
      paths:
       - search_location_one/somedir/
       - /opt/other_location/somedir/
```

Register 변수

종종 플레이북에서 실행한 명령의 결과를 변수로 저장했다가 나중에 재사용 할 필요가 있습니다. 이와 같은 방법은 특정 사이트의 fact나 인스턴스를 작성하지 않고도 쉽게 특정 프로그램이 존재하는가 확인할 수 있습니다.

`register` 키워드는 어느 변수에 실행 결과를 저장할 것인가를 나타냅니다. 결과 변수는 템플릿, `action` 라인 또는 `when` 구문 등에서 사용가능 합니다.

```
- name: test play
  hosts: all

  tasks:

      - shell: cat /etc/motd
        register: motd_contents

      - shell: echo "motd contains the word hi"
        when: motd_contents.stdout.find('hi') != -1
```

위의 결과에서 보듯이 등록된 변수 `motd_contents` 의 ***stdout*** 을 통하여 결과를 얻을 수 있습니다.

또는,

```
- name: registered variable usage as a with_items list
  hosts: all

  tasks:

      - name: retrieve the list of home directories
        command: ls /home
        register: home_dirs

      - name: add home dirs to the backup spooler
        file: path=/mnt/bkspool/{{ item }} src=/home/{{ item }} state=link
        with_items: "{{ home_dirs.stdout_lines }}"
        # same as with_items: "{{ home_dirs.stdout.split() }}"
```

`with_items` 를 이용하여 ***stdout*** 결과의 각 라인별로 작업을 할 수 있습니다.

위와 같이 등록된 변수의 출력 결과는 `stdout` 을 통하여 읽을 수 있고 또한 결과가 없는지 (emptiness) 확인할 수 있습니다.

```
- name: check registered variable for emptiness
  hosts: all

  tasks:

      - name: list contents of directory
        command: ls mydir
        register: contents

      - name: check contents for emptiness
        debug: msg="Directory is empty"
        when: contents.stdout == ""
```

반복문

표준 반복문

간단한 반복문은 다음과 같이 작성할 수 있습니다.

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

만약 변수 파일에 YAML 목록으로 정의하였거나 `vars` 섹션에 목록 변수를 정의 하였다면,

```
with_items: "{{ somelist }}"
```

와 같이 사용가능합니다.

위의 반복문을 풀어 본다면 다음과 동일합니다.

```
- name: add user testuser1
  user: name=testuser1 state=present groups=wheel
- name: add user testuser2
  user: name=testuser2 state=present groups=wheel
```

yum과 apt 모듈 역시 `with_items` 을 잘 이용할 수 있습니다. 만약 해쉬 목록을 가지고 있다면 다음과 같이 사용하면 됩니다.

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

만약 `when` 문장이 `with_items` 과 같이 사용된다면 (또는 다른 반복문에서도 동일하게) 각 항목에 대하여 `when` 구문 체크를 합니다.

중첩 반복문

```
- name: give users access to multiple databases
  mysql_user: name={{ item[0] }} priv={{ item[1] }}.*:ALL append_privs=yes password=
  with_nested:
    - [ 'alice', 'bob' ]
    - [ 'clientdb', 'employeedb', 'providerdb' ]
```

`with_nested` 항목에 이전에 사용한 변수를 사용할 수 있습니다.

```
- name: here, 'users' contains the above list of employees
  mysql_user: name={{ item[0] }} priv={{ item[1] }}.*:ALL append_privs=yes password=
  with_nested:
    - "{{ users }}"
    - [ 'clientdb', 'employeedb', 'providerdb' ]
```

해쉬에 대한 반복문

버전 1.5 이후.

다음과 같은 변수가 있다고 가정하면,

```
---
users:
  alice:
    name: Alice Appleworth
    telephone: 123-456-7890
  bob:
    name: Bob Bananarama
    telephone: 987-654-3210
```

이 변수를 이용한 반복문 활용 예 입니다.

```
tasks:
  - name: Print phone records
    debug: msg="User {{ item.key }} is {{ item.value.name }} ({{ item.value.telephor
    with_dict: "{{ users }}"
```

파일을 대상으로 하는 반복문

`with_file` 해당 파일의 내용을 목록으로 반복하는데 `item` 은 반복되는 항목을 가리킵니다.

```
---
- hosts: all

  tasks:

    # emit a debug message containing the content of each file.
    - debug:
        msg: "{{ item }}"
      with_file:
        - first_example_file
        - second_example_file
```

위에서 `first_example_file` 이 **hello**를 담고 있고 `second_example_file` 이 **world** 라는 문자열을 가진다면 이 결과는,

```
TASK [debug msg={{ item }}] ***************************************************
ok: [localhost] => (item=hello) => {
    "item": "hello",
    "msg": "hello"
}
ok: [localhost] => (item=world) => {
    "item": "world",
    "msg": "world"
}
```

파일목록에 대한 반복문

`with_fileglob` 은 어느 디렉터리에서 매칭되는 모든 파일 목록을 구해옵니다. (재귀적으로 탐색하지 않습니다.) 이것은 파이썬의 glob 라이브러리를 호출합니다.

```
---
- hosts: all

  tasks:

    # first ensure our target directory exists
    - file: dest=/etc/fooapp state=directory

    # copy each file over that matches the given pattern
    - copy: src={{ item }} dest=/etc/fooapp/ owner=root mode=600
      with_fileglob:
        - /playbooks/files/fooapp/*
```

> **노트**
>
> `with_fileglob` 이 role에서 상대 경로로 사용된다면 Ansible은 ***roles//files 디렉터리를 기준으로 상대경로로 인식할 겁니다.

데이터의 패러렐 셋의 반복

> **노트**
>
> 이 기능은 일상적으로 사용하는 것은 아니지만 기술해 봅니다.

다음과 같은 변수가 있고,

```
---
alpha: [ 'a', 'b', 'c', 'd' ]
numbers:  [ 1, 2, 3, 4 ]
```

`(a, 1)` and `(b, 2)` ... 등과 같이 조합하여 반복하고 싶다면, `with_together`을 사용합니다.

```
tasks:
    - debug: msg="{{ item.0 }} and {{ item.1 }}"
      with_together:
        - "{{ alpha }}"
        - "{{ numbers }}"
```

하위항목에 대한 반복문

때로는 사용자 목록을 만들어 해당 사용자에 대한 반복문을 돌면서 일련의 SSH 키로 로그인하는 등의 일을 할 필요가 있습니다.

어떻게 이런 일을 수행할까요? `group_vars/all` 파일에 있거나 `vars_files`에 의해 다음과 같이 정의되고 로드된다면,

```
---
users:
  - name: alice
    authorized:
      - /tmp/alice/onekey.pub
      - /tmp/alice/twokey.pub
    mysql:
        password: mysql-password
        hosts:
          - "%"
          - "127.0.0.1"
          - "::1"
          - "localhost"
        privs:
          - "*.*:SELECT"
          - "DB1.*:ALL"
  - name: bob
    authorized:
      - /tmp/bob/id_rsa.pub
    mysql:
        password: other-mysql-password
        hosts:
          - "db1"
        privs:
          - "*.*:SELECT"
          - "DB2.*:ALL"
```

아래와 같이 반복문을 사용할 수 있습니다.

```
- user: name={{ item.name }} state=present generate_ssh_key=yes
  with_items: "{{ users }}"

- authorized_key: "user={{ item.0.name }} key='{{ lookup('file', item.1) }}'"
  with_subelements:
     - "{{ users }}"
     - authorized
```

비슷하게 mysql 에 대해서는,

```
- name: Setup MySQL users
  mysql_user: name={{ item.0.name }} password={{ item.0.mysql.password }} host={{ it
  with_subelements:
    - "{{ users }}"
    - mysql.hosts
```

해쉬 (파이썬의 dictionary)의 목록에 대하여 하위 항목을 찾아가는 것은 해당 레코드의 키로 접근하는 방법입니다.

종종 해시 목록에서 하위 항목의 세번째 항목을 추가하거나 할 필요가 있는데 `skip_missing` 플래그를 추가하면 됩니다. 만약 이 플래그가 True 이면 주어진 하위키를 포함하지 않은 항목은 건너 뜁니다. 만약 해당 플래그가 False이거나 설정되지 않았다면 존재하지 않는 키를 접근하려면 오류가 발생합니다.

정수열에 대한 반복문

`with_sequence` 는 높임차순의 항목 목록을 생성합니다. start, end와 선택적 step을 지정할 수 있습니다.

인자는 key=value 와 같이 지정합니다.

정수값은 십진수, 16진수(예 0x3f8) 또는 8진수(0600) 등이 올 수 있습니다. 음수는 지원되지 않습니다. 다음과 같이 이용하면 됩니다.

```
---
- hosts: all

  tasks:

    # create groups
    - group: name=evens state=present
    - group: name=odds state=present

    # create some test users
    - user: name={{ item }} state=present groups=evens
      with_sequence: start=0 end=32 format=testuser%02x

    # create a series of directories with even numbers for some reason
    - file: dest=/var/stuff/{{ item }} state=directory
      with_sequence: start=4 end=16 stride=2

    # a simpler way to use the sequence plugin
    # create 4 groups
    - group: name=group{{ item }} state=present
      with_sequence: count=4
```

임의 선택

`random_choice` 기능은 임의로 어떤 항목을 선택할 수 있습니다.

```
- debug: msg={{ item }}
  with_random_choice:
     - "go through the door"
     - "drink from the goblet"
     - "press the red button"
     - "do nothing"
```

Do-Until 반복문

반복문 중에 다음과 같이 할 수 있습니다.

```
- action: shell /usr/bin/foo
  register: result
  until: result.stdout.find("all systems go") != -1
  retries: 5
  delay: 10
```

위의 예는 쉘 모듈을 재귀적으로 수행하는데 결과가 `all systems go` 라는 문자열이 결과에 나올 때까지 실행하는데 10초의 delay를 두고 5회까지 반복합니다. 디폴트는 3회까지 반복에 5초의 delay입니다.

첫번째 매칭되는 파일 구하기

이것은 반복이라기 보다는 선택입니다. 참조할 파일이름은 변수로 나타내며, 참조할 파일 목록 중에서 주어진 조건식에 맞는 첫번째 파일을 찾으려면 어떻게 할까요? 다음과 같이 하면 됩니다.

```
- name: INTERFACES | Create Ansible header for /etc/network/interfaces
  template: src={{ item }} dest=/etc/foo.conf
  with_first_found:
    - "{{ ansible_virtualization_type }}_foo.conf"
    - "default_foo.conf"
```

좀 더 긴 버전으로는,

```
- name: some configuration template
  template: src={{ item }} dest=/etc/file.cfg mode=0444 owner=root group=root
  with_first_found:
    - files:
       - "{{ inventory_hostname }}/etc/file.cfg"
      paths:
       - ../../../templates.overwrites
       - ../../../templates
    - files:
        - etc/file.cfg
      paths:
        - templates
```

프로그램 실행 결과에 대한 반복

> **노트**
>
> 자주 사용하는 기능이 아닙니다.

때로는 프로그램을 실행하고 그 결과를 줄 단위로 반복하여 작업할 필요가 있습니다. Ansible에서는 이런 작업을 하는데 있어 원격 관리 머신이 아니고 제어 머신에서 도는 결과입니다.

```
- name: Example of looping over a command result
  shell: /usr/bin/frobnicate {{ item }}
  with_lines: /usr/bin/frobnications_per_host --param {{ inventory_hostname }}
```

만약 명령을 원격에서 수행해야 한다면 다음과 같이 진행합니다.

```
- name: Example of looping over a REMOTE command result
  shell: /usr/bin/something
  register: command_result

- name: Do something with each result
  shell: /usr/bin/something_else --param {{ item }}
  with_items: "{{ command_result.stdout_lines }}"
```

색인으로 반복문 돌기

> **노트**
>
> 자주 사용하는 기능이 아닙니다.

```
- name: indexed loop demo
  debug: msg="at array position {{ item.0 }} there is a value {{ item.1 }}"
  with_indexed_items: "{{ some_list }}"
```

ini 파일에서 반복하기

다음과 같은 ini 파일이 있고,

```
[section1]
value1=section1/value1
value2=section1/value2

[section2]
value1=section2/value1
value2=section2/value2
```

`with_ini` 을 사용하여,

```
- debug: msg="{{ item }}"
  with_ini: value[1-2] section=section1 file=lookup.ini re=true
```

와 같이 사용하면 그 결과는,

```
{
      "changed": false,
      "msg": "All items completed",
      "results": [
          {
              "invocation": {
                  "module_args": "msg=\"section1/value1\"",
                  "module_name": "debug"
              },
              "item": "section1/value1",
              "msg": "section1/value1",
              "verbose_always": true
          },
          {
              "invocation": {
                  "module_args": "msg=\"section1/value2\"",
                  "module_name": "debug"
              },
              "item": "section1/value2",
              "msg": "section1/value2",
              "verbose_always": true
          }
      ]
  }
```


목록 Flattening

> **노트**
>
> 자주 사용하는 기능이 아닙니다.

다음과 같이 목록의 목록 등과 같이 목록이 중첩되어 있다면,

```
---
# file: roles/foo/vars/main.yml
packages_base:
  - [ 'foo-package', 'bar-package' ]
packages_apps:
  - [ ['one-package', 'two-package' ]]
  - [ ['red-package'], ['blue-package']]
```

모든 중첩된 목록도 모두 1차원의 목록으로만 하려면 `with_flattened` 을 이용합니다.

```
- name: flattened loop demo
  yum: name={{ item }} state=installed
  with_flattened:
     - "{{ packages_base }}"
     - "{{ packages_apps }}"
```

register를 이용한 반복문

`register` 를 이용한 결과는 `results` 속성에 들어있습니다.

```
- shell: echo "{{ item }}"
  with_items:
    - one
    - two
  register: echo
```

이것은 다음과 같이 반복문 없이 `register` 를 이용한 결과를 이용한 데이터 구조체와는 다릅니다.

```
{
    "changed": true,
    "msg": "All items completed",
    "results": [
        {
            "changed": true,
            "cmd": "echo \"one\" ",
            "delta": "0:00:00.003110",
            "end": "2013-12-19 12:00:05.187153",
            "invocation": {
                "module_args": "echo \"one\"",
                "module_name": "shell"
            },
            "item": "one",
            "rc": 0,
            "start": "2013-12-19 12:00:05.184043",
            "stderr": "",
            "stdout": "one"
        },
        {
            "changed": true,
            "cmd": "echo \"two\" ",
            "delta": "0:00:00.002920",
            "end": "2013-12-19 12:00:05.245502",
            "invocation": {
                "module_args": "echo \"two\"",
                "module_name": "shell"
            },
            "item": "two",
            "rc": 0,
            "start": "2013-12-19 12:00:05.242582",
            "stderr": "",
            "stdout": "two"
        }
    ]
}
```


이런 결과를 활용하기 위해서

```
- name: Fail if return code is not 0
  fail:
    msg: "The command ({{ item.cmd }}) did not have a 0 return code"
  when: item.rc != 0
  with_items: "{{ echo.results }}"
```

관리대상목록(Inventory) 반복

Inventory의 반복은 여러 방법으로 가능합니다. `play_hosts` 또는 `groups` 변수로 `with_items` 를 이용할 수 있습니다.

```
# show all the hosts in the inventory
- debug: msg={{ item }}
  with_items: "{{ groups['all'] }}"

# show all the hosts in the current play
- debug: msg={{ item }}
  with_items: play_hosts
```

다른 방법으로 `inventory_hostnames` 플러그인을 이용한 것이 있습니다.

```
# show all the hosts in the inventory
- debug: msg={{ item }}
  with_inventory_hostnames: all

# show all the hosts matching the pattern, ie all but the group www
- debug: msg={{ item }}
  with_inventory_hostnames: all:!www
```

반복문 제어

버전 2.0에서 (플레이북 include 가 아닌) `with_loops` 와 태스크 include를 사용할 수 있습니다. Ansible 2.1 이후부터는 `loop_control` 이 반복문에 사용되는 변수이름을 지정하는데 사용할 수 있습니다.

```
# main.yml
- include: inner.yml
  with_items:
    - 1
    - 2
    - 3
  loop_control:
    loop_var: outer_item

# inner.yml
- debug: msg="outer item={{ outer_item }} inner item={{ item }}"
  with_items:
    - a
    - b
    - c
```

> **노트**
>
> 만약 Ansible이 이미 정의된 변수를 현재 반복문에 사용된 것을 발견했다면 태스크에 오류가 발생할 것입니다.

복잡한 데이터 구조가 반복문에 사용되면 경우 c(label) 지시자를 이용할 수 있습니다.

```
- name: create servers
  digital_ocean: name={{item.name}} state=present ....
 with_items:
    - name: server1
      disks: 3gb
      ram: 15Gb
      netowrk:
        nic01: 100Gb
        nic02: 10Gb
        ...
  loop_control:
    label: "{{item.name}}"
```

이것은 디폴트 `{{item}}` 대신 `label` 필드를 보여줍니다.

반복문 제어를 위한 또다른 옵션은 c(pause)를 이용하여 반복문 사이에 얼마큼 (초 단위) 멈출지를 나타냅니다.

```
# main.yml
- name: create servers, pause 3s before creating next
  digital_ocean: name={{item}} state=present ....
 with_items:
    - server1
    - server2
 loop_control:
    pause: 3
```

2.0 에 추가된 반복문

Ansible 2.0 에는 `loop_control` 이 존재하지 않기 때문에 `set_fact` 를 이용합니다.

```yaml
# main.yml
- include: inner.yml
  with_items:
    - 1
    - 2
    - 3

# inner.yml
- set_fact:
    outer_item: "{{ item }}"

- debug:
    msg: "outer item={{ outer_item }} inner item={{ item }}"
  with_items:
    - a
    - b
    - c
```

자신의 반복문 사용

임의의 데이터 구조에 대해서 자신만의 반복문을 이용할 경우가 있는데 플러그인 개발을 참고하시기 바랍니다.

Blocks

2.0 에서 태스크의 논리적 그룹 및 플레이의 오류 처리를 위하여 block 이라는 것을 추가했습니다. 단일 태스크에 데이터나 지시자를 쉽게 지정할 수 있는 block 단계가 적용될 수 있습니다.

블럭(block) 예제.

```yaml
  tasks:
    - block:
        - yum: name={{ item }} state=installed
          with_items:
            - httpd
            - memcached
        - template: src=templates/src.j2 dest=/etc/foo.conf
        - service: name=bar state=started enabled=True

      when: ansible_distribution == 'CentOS'
      become: true
      become_user: root
```

위의 예제에서 각각 3개의 태스크 (yum, template, service)가 `when` 조건문 이후에 수행될 것입니다.

오류 처리

대부분의 프로그래밍 언어에서 지원하는 오류 예외처리와 `block` 이 관계되어 있습니다.

block 에러 처리 예제:

```
tasks:
 - block:
     - debug: msg='i execute normally'
     - command: /bin/false
     - debug: msg='i never execute, cause ERROR!'
   rescue:
     - debug: msg='I caught an error'
     - command: /bin/false
     - debug: msg='I also never execute :-('
   always:
     - debug: msg="this always executes"
```

`block` 문에 있는 태스크는 정상적으로 수행되는 것인데 만약 중간에 에러가 발생하면 `rescue` 섹션이 수행됩니다. `always` 섹션은 에러가 발생하던지 안하던지 상관없이 `block` 과 `rescue` 섹션 다음에 수행됩니다.

다른 예로 오류가 발생하였을 때 어떻게 핸들러를 수행하는가 입니다.

```
tasks:
 - block:
     - debug: msg='i execute normally'
       notify: run me even after an error
     - command: /bin/false
   rescue:
     - name: make sure all handlers run
       meta: flush_handlers
handlers:
  - name: run me even after an error
    debug: msg='this handler runs even on error'
```

Strategies

2.0에서 `strategy` 라는 플레이 제어 방식이 추가되었는데, 디폴트로 플레이는 `linear` 전략(strategy) 으로 수행됩니다. 다음 태스크로 넘어가기 전에 모든 호스트에 해당 태스크 작업을 마치는데 디폴트로 5개의 병렬화로 작업을 합니다.

`serial` 지시자는 호스트의 일정 부분에 대한 **batch** 작업을 하는데 다음 **batch** 작업이 시작하기 전에 작업을 마무리 합니다.

다음은 `free` 지시자가 있는데 각각의 호스트가 플레이별로 빨리 끝낼 수 있는데로 긑내는 방식입니다.

```
- hosts: all
  strategy: free
  tasks:
  ...
```

Strategy 플러그인

strategy는 사용자가 제공하거나 Ansible 코드로 새롭게 실행가능한 새로운 형태의 플러그인으로 구현되었습니다.

`debug` strategy 가 한 예입니다. 플레이북 디버거를 참고하십시오.

Best Practices

Content Organization

꼭 이렇게 하라는 것은 아니지만 role을 이용하는 것이 좋은 방법입니다.

디렉터리 구조

```
production                # inventory file for production servers
staging                   # inventory file for staging environment

group_vars/
   group1                 # here we assign variables to particular groups
   group2                 # ""
host_vars/
   hostname1              # if systems need specific variables, put them here
   hostname2              # ""

library/                  # if any custom modules, put them here (optional)
filter_plugins/           # if any custom filter plugins, put them here (optional)

site.yml                  # master playbook
webservers.yml            # playbook for webserver tier
dbservers.yml             # playbook for dbserver tier

roles/
    common/               # this hierarchy represents a "role"
        tasks/            #
            main.yml      #  <-- tasks file can include smaller files if warranted
        handlers/         #
            main.yml      #  <-- handlers file
        templates/        #  <-- files for use with the template resource
            ntp.conf.j2   #  <------- templates end in .j2
        files/            #
            bar.txt       #  <-- files for use with the copy resource
            foo.sh        #  <-- script files for use with the script resource
        vars/             #
            main.yml      #  <-- variables associated with this role
        defaults/         #
            main.yml      #  <-- default lower priority variables for this role
        meta/             #
            main.yml      #  <-- role dependencies

    webtier/              # same kind of structure as "common" was above, done for t
    monitoring/           # ""
    fooapp/               # ""
```

Cloud를 이용한 Dynamic Inventory 사용

동적 인벤토리를 이용한 관리대상 목록 관리를 클라우드 provider로 이용하는 것입니다.

Staging 과 Production의 차이점 이용

만약 정적 인벤토리를 이용한다면 환경 변수를 이용하여 관리하는 것이 보통이지만 동적 인벤토리에서는 그룹에 따라 작업하는 것이 가능합니다.

이런 스테이징이나 프로덕션 등의 목적에 따라 그룹을 나누거나 지역 (데이터 센터 등)에 따라 그룹을 나눕니다.

```
# file: production

[atlanta-webservers]
www-atl-1.example.com
www-atl-2.example.com

[boston-webservers]
www-bos-1.example.com
www-bos-2.example.com

[atlanta-dbservers]
db-atl-1.example.com
db-atl-2.example.com

[boston-dbservers]
db-bos-1.example.com

# webservers in all geos
[webservers:children]
atlanta-webservers
boston-webservers

# dbservers in all geos
[dbservers:children]
atlanta-dbservers
boston-dbservers

# everything in the atlanta geo
[atlanta:children]
atlanta-webservers
atlanta-dbservers

# everything in the boston geo
[boston:children]
boston-webservers
boston-dbservers
```

Group, Host 변수

그룹은 체계적으로 분류하는데 좋은 방법이지만 모든 그룹이 그렇지는 않습니다. 때로는 변수를 적용할 수도 있습니다.

```
---
# file: group_vars/atlanta
ntp: ntp-atlanta.example.com
backup: backup-atlanta.example.com
```

또는

```
---
# file: group_vars/webservers
apacheMaxRequestsPerChild: 3000
apacheMaxClients: 900
```

디폴트로는

```
---
# file: group_vars/all
ntp: ntp-boston.example.com
backup: backup-boston.example.com
```

각각의 호스트 변수는

```
---
# file: host_vars/db-bos-1.example.com
foo_agent_port: 86
bar_agent_port: 99
```

최상위 플레이북은 Rule에 의해 분리됨

전체 구조를 담고 있는 site.yml 은 다른 플레이북 등을 담고 있으므로 아주 간단한 것이 보통입니다.

```
---
# file: site.yml
- include: webservers.yml
- include: dbservers.yml
```

또한 같은 최상위 레벨에 있는 webservers.yml 같은 것은 해당 webservers 그룹이나 해당 그룹의 role 에 의해 작업이 이루어지는게 일반적입니다.

```
---
# file: webservers.yml
- hosts: webservers
  roles:
    - common
    - webtier
```

최상위에 site.yml 부터 시작하여 webservers.yml 등과 같이 각각의 역할 별 플레이북이 실행되도록 합니다.

때로는 명령줄에서 특정 플레이북만 선택적으로 실행가능합니다.

```
$ ansible-playbook site.yml --limit webservers
$ ansible-playbook webservers.yml
```

Role을 구성하는 태스크와 핸들러

다음은 role이 어떻게 동작하는지 설명하는 예제로써 NTP 를 필요에 따라 설정하는 것입니다.

```
---
# file: roles/common/tasks/main.yml

- name: be sure ntp is installed
  yum: name=ntp state=installed
  tags: ntp

- name: be sure ntp is configured
  template: src=ntp.conf.j2 dest=/etc/ntp.conf
  notify:
    - restart ntpd
  tags: ntp

- name: be sure ntpd is running and enabled
  service: name=ntpd state=started enabled=yes
  tags: ntp
```

다음은 핸들러 파일 예제입니다.

```
---
# file: roles/common/handlers/main.yml
- name: restart ntpd
  service: name=ntpd state=restarted
```

위의 구조에 따른 ansible-playbook 예제

전체 플레이 작업

```
$ ansible-playbook -i production site.yml
```

NTP 작업만 하려면

```
$ ansible-playbook -i production site.yml --tags ntp
```

webserver 만 작업하려면

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

Boston에 있는 webserver에 대해 작업하면

```
$ ansible-playbook -i production webservers.yml --limit boston
```

호스트 중에 앞, 뒤 10개씩의 호스트만 적용하려면

```
$ ansible-playbook -i production webservers.yml --limit boston[1-10]
$ ansible-playbook -i production webservers.yml --limit boston[11-20]
```

애드-혹 작업을 각각 하려면

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

버전 1.1 이후에 사용되는 유용한 명령행 옵션 등으로

```
# confirm what task names would be run if I ran this command and said "just ntp task
$ ansible-playbook -i production webservers.yml --tags ntp --list-tasks

# confirm what hostnames might be communicated with if I said "limit to boston"
$ ansible-playbook -i production webservers.yml --limit boston --list-hosts
```

배포 대 설정관리 구성

위와 같은 구성은 일반적인 설정관리 구성방법입니다. 하지만 다단계 배포를 할 때에는 단계를 넘어가거나 응용프로그램을 마무리하는 추가 플레이북이 필요합니다. 이런 경우 `site.yml` 에 `deploy_exampledotcom.yml` 와 같은 추가 플레이북을 추가합니다.

Staging 대 Production

Staging (또는 테스팅)과 Production 환경을 분리하는 좋은 방법은 Inventory 에서 분리하는 방법입니다. 이것을 ansible-playbook을 실행할 때 -i 옵션으로 서로 다른 인벤토리를 이용합니다.

Production에서 테스트하기 전에 각각의 Staging 환경에서 테스팅을 해 보는 것이 중요합니다. 작은 범위의 스테이징에서 테스트를 충분히 하는 것이 좋습니다.

Rolling 업데이트

`serial` 키워드를 이해한다. 만약 sebserver 팜을 업데이트하고 제어하려면 얼마나 많은 기계를 배치작업에서 한번에 할지 결정할 필요가 있습니다.

항상 State 고려

`state` 패러미터는 많은 모듈에서 선택사항입니다. `state=present` 또는 `state=absent` 이건간에 각각의 플레이북에서 패러미터로 남겨두는 것이 좋습니다.

Role 별 그룹

그룹은 아무리 강조해도 지나치지 않습니다. 각 그룹에 *webservers* 또는 *dbservers* 와 같이 의미를 부여해 줍니다.

이렇게 하여 각 플레이북에서 역할별 작업을 할 수 있습니다.

운영체제 등의 차이점 분류

서로 다른 두 운영 체제에 차이를 패러미터로 처리하려고 할 때, `group_by` 모듈을 잘 사용하면 좋습니다.

이것은 인벤토리 파일에 미리 정의되어 있지 않더라도 일정 조건에 따른 호스트의 그룹을 동적으로 지정할 수 있습니다.

```
---

# talk to all hosts just so we can learn about them
- hosts: all
  tasks:
     - group_by: key=os_{{ ansible_distribution }}

# now just on the CentOS hosts...

- hosts: os_CentOS
  gather_facts: False
  tasks:
     - # tasks that only happen on CentOS go here
```

위와 같이 함으로써 운영체제에 따라 동적으로 그룹을 나눌 수 있습니다.

만약 그룹에 특화된 설정이 필요하다면 다음과 같이 지정합니다.

```
---
# file: group_vars/all
asdf: 10

---
# file: group_vars/os_CentOS
asdf: 42
```

위의 예에서는 CentOS 운영체제인 경우에는 asdf 라는 변수가 42 값을 갖고 그 밖에는 10 값을 갖는 예 입니다.

다른 대안으로 만약 변수만 필요하다면,

```
- hosts: all
  tasks:
    - include_vars: "os_{{ ansible_distribution }}.yml"
    - debug: var=asdf
```

플레이북으로 모듈 번들링

만약 플레이북이 YAML파일의 상대 위치에 `./library` 디렉터리를 가지고 있다면 이 디렉터리는 ansible 모듈 경로에 자동으로 추가됩니다. 이런 방식으로 플레이북과 모듈을 함께 배포할 수 있는 좋은 방법이 될 수 있습니다.

공백 및 코멘트

가독성을 위해서 적절한 공백 및 코멘트 (`#` 로 시작하는 라인)를 잘 넣기를 권장합니다.

항상 태스크에 이름 지정

비록 태스크에 `name` 항목을 비워놓을 수 있지만 항상 적절한 이름을 지정하기 바랍니다. 이것은 플레이북이 실행될 때 해당 이름을 출력하기 때문에 여러모로 유용합니다.

간단하게 유지

가능하다면 복잡하지 않고 간단하게 합니다. 모든 사용가능한 Ansible 기능을 한데 사용하려고 하지 마십시오.예를 들어 `vars` , vars*files* `,` *vars*prompt 및 --extra-vars` 또는 외부 인벤토리 파일 등을 모두 사용할 수는 있지만 결국 우선순위에 의해 마지막 변수만 사용될 것이므로 필요한 것을 이용하는 것이 좋습니다. 만약 중복해서 사용했다면 간단하게 할 수 있는 방법이 존재할 것입니다.

버전 관리

가능한한 버전관리를 이용하기 바랍니다. 이렇게 함으로써 수정된 내용 등을 추적할 수 있습니다.

변수 및 Vaults

Ansible 설정등에 대한 간단한 관리에는 `grep` 과 같은 도구를 종종 사용하기도 합니다. 또한 `vault` 라는 것을 이용하면 이런 변수의 내용을 직접 확인할 수 없게 함으로써 좀 더 보안에 안정적일 수 있습니다. 플레이북이 실행될 때 암호화되지 않은 변수 뿐만 아니라 민감한 변수는 암호화된 파일에서 읽어 복호화하여 해당 내용을 확인합니다.

`group_vars` 하위 폴더에 그룹 이름을 딴 설정 파일을 이용하십시오. 아니면 대신 `vars` 와 `vault` 라는 이름의 두 파일을 만듭니다. `vars` 파일에는 민감한 변수를 포함한 모든 변수를 정의하고, `vault_` 라고 시작하는 이름의 파일에는 암호화가 필요한 민감한 변수를 복사합니다. 이런 `vault` 파일은 암호화 됩니다.

모듈에 관하여

Ansible은 이미 바로 플레이북에서 바로 수행할 수 있는 많은 모듈을 제공합니다.

사용자는 또한 자신의 모듈을 만들 수 있습니다. 이런 사용자 모듈은 서비스, 패키지, 파일 또는 시스템 명령어를 수행하는 등의 시스템 자원을 제어하는 모듈입니다.

소개

태스크 플러그인 또는 ***라이브러리 플러그인*** 으로 불리기도 하는 모듈은 ansible에서 플레이북 태스크를 실제 동작하는 주체입니다. 뿐만 아니라 `ansible` 명령에서 바로 실행 시켜 볼 수도 있습니다.

```
$ ansible webservers -m service -a "name=httpd state=started"
$ ansible webservers -m ping
$ ansible webservers -m command -a "/sbin/reboot -t now"
```

각각의 모듈은 인자를 제공하는데 `key=value` 와 같은 형식입니다. 인자를 받지 않는 모듈도 있지만 **command/shell** 모듈은 간단히 실행될 명령어를 인자로 받습니다.

플레이북에서는 다음과 같이 실행됩니다.

```
- name: reboot the servers
  action: command /sbin/reboot -t now
```

또는 줄여서,

```
- name: reboot the servers
  command: /sbin/reboot -t now
```

혹은 YAML을 이용하여,

```
- name: restart webserver
  service:
    name: httpd
    state: restarted
```

모두 같은 방법으로 모듈에 인자를 받아 실행합니다.

모든 모듈은 JSON 형식의 데이터를 리턴하는데 명령행이나 플레이북 모두에서 그 결과를 일일이 확인할 필요까지는 없습

니다. 만약 자신만의 모듈을 만든다면 적당한 형식의 리턴을 하면되고 특정 프로그래밍 언어를 따라야 하는 것은 아닙니다.

모듈은 *멱등성(idempotent)*을 유지하려고 애를 쓰는데 이는 시스템에 변화가 오지 않는한 시스템 자체를 변화시키지 않음을 의미합니다.

```
$ ansible-doc yum
```

위의 예에서 처럼 (예에서는 *yum* 모듈) 해당 모듈의 문서를 확인할 수 있습니다.

만약 해당 시스템에 설치된 모든 모듈 목록을 확인하려면,

```
$ ansible-doc -l
```

라고 합니다.

핵심 모듈

ansible 팀에서 직접 관리하고 유지보수하는 핵심모듈이 있습니다. 일반적으로 기타(*extras*) 저장소에 있는 모듈에 비해 높은 우선순위로 유지보수합니다. 핵심 모듈 저장소에 있습니다.

만약 핵심 모듈에 오류가 발견되면 해당 저장소의 이슈에 보고해 주기 바랍니다.

기타(*Extras*) 모듈

현재는 Ansible 모듈에 이런 기타 모듈도 같이 제공되지만 앞으로는 개별적으로 제공될 수도 있습니다. 이런 기타 모듈은 대부분 커뮤너티에 의해 유지보수 됩니다.

기타 모듈 중에서도 시간이 흐름에 따라 기본 모듈로 격상되는 것도 있습니다.

기타 모듈 저장소에 소스가 있습니다. 버그 리포트는 해당 저장소에서 하면 되고 다른 질문은 해당 그룹에 해 주시기 바랍니다.

공통 리턴 값

Ansible 모듈은 변수에 등록(*register*) 될 수 있는 데이터 구조에 리턴되거나 또는 ***ansible*** 명령에 의해 직접 실행되는 화면 결과를 통해 리턴값을 확인할 수 있습니다. 각 모듈은 (*ansible-doc* 에 의해 확인 가능한) 문서에 고유 리턴 값이 표기됩니다.

> **노트**
>
> 다음에 나오는 리턴 값 중 일부는 Ansible 자체에 의해 설정되는 것도 있습니다.

backup_file

backup=yes 등과 같이 백업을 설정하는 경우 위의 결과는 백업파일의 경로를 타나냅니다.

changed

해당 태스크가 변경되었는가를 나타내는 플래그 입니다.

failed

해당 태스크가 실패인지 아닌지를 나타내는 플래그 입니다.

invocation

모듈이 어떻게 호출되었는가를 나타냄

msg

사용자에게 전달되는 메시지 문자열

rc

몇몇 모듈은 명령행 유틸리티를 실행하거나 (raw, shell, command 등과 같은 모듈처럼) 직접 명령을 실행하는 경우 해당 실행된 결과의 리턴 코드를 담고 있습니다.

results

만약 이 키가 존재하면 태스크의 반복문이 실행된 것이며 반복 항목마다의 *결과*를 담고 있는 목록입니다.

skipped

만약 해당 태스크가 건너 뛰었는지를 나타내는 플래그

stderr

몇몇 모듈은 명령행 유틸리티나 명령을 바로 수행하기도 하는데 이런 경우 ***stderr***로 출력된 결과를 담고 있을 수 있습니다.

stderr_lines

위와 같이 stderr이 발생한 경우에 각 라인의 목록

stdout

명령을 수행하는 모듈인 경우 ***stdout*** 결과를 담고있는 문자열

stdout_lines

stdout이 존재하는 경우 해당 라인 목록

내부 사용

각 모듈이 사용하기 보다는 Ansible 자체에서 등록하는 변수입니다.

ansible_facts

해당 호스트에 할당된 fact에 추가될 디렉터리를 담고 있습니다.

exception

모듈에서 예외가 발생한 경우 해당 *traceback* 정보를 담고 있습니다. (-vvv) 와 같은 높은 메시지 출력 옵션에서 보여집니다.

warnings

사용자에게 보여질 경고 문자열 목록을 담고 있습니다.

모듈 색인

클라우드 모듈

클러스터링 모듈

커맨드 모듈

command (핵심모듈)

원격 노드에서 명령 실행

설명

`command` 모듈은 명령과 일련의 인자를 받아 해당 명령을 실행합니다. 주어진 명령은 해당 모드에서 수행됩니다. 해당 명령은 shell 이 아니라 바로 수행되므로 `$HOME` 과 같은 환경 변수 및 `<`, `>`, `|`, `;` 및 `&` 과 같은 쉘 동작은 하지 않을 것입니다. (이런 쉘 동작을 하려면 `shell` 모듈을 이용합니다)

옵션

패러미터	필수	디폴트	선택	비고
chdir	no			해당 명령을 실행하기 전 해당 디렉터리로 이동
creates	no			파일이름 또는 (2.0이후) glob 패턴으로 만약 존재한다면 아무런 동작 안 함
executable	no			명령을 실행하는데 사용할 shell 변경. 절대경로로 표현
free_form	**yes**			명령을 수행할 free form 지정. 실제 `free form` 패러미터 없음
removes	no			파일이름 또는 (2.0이후) glob 패턴으로 만약 존재한다면 아무런 동작 안 함
warn (1.8 이후)	no	True		만약 ansible.cfg에 명령 warnings가 on 되어 있더라도 이 패러미터가 no/false 로 되어 있다면 경고 안함

예제

```
# Example from Ansible Playbooks.
- command: /sbin/shutdown -t now

# Run the command if the specified file does not exist.
- command: /usr/bin/make_database.sh arg1 arg2 creates=/path/to/database

# You can also use the 'args' form to provide the options. This command
# will change the working directory to somedir/ and will only run when
# /path/to/database doesn't exist.
- command: /usr/bin/make_database.sh arg1 arg2
  args:
    chdir: somedir/
    creates: /path/to/database
```

expect (확장 모듈)

2.0에서 추가됨.

설명

어떤 명령을 실행하고 그 결과에서 특정 문자열이 올 때까지 기다립니다. 해당 명령은 shell 이 아니라 바로 수행되므로 `$HOME` 과 같은 환경 변수 및 `<`, `>`, `|`, `;` 및 `&` 과 같은 쉘 동작은 하지 않을 것입니다.

요구사항 (모듈이 실행되는 호스트)

- python >= 2.6
- pexpect >= 3.3

옵션

패러미터	필수	디폴트	선택	비고
chdir	no			해당 명령을 실행하기 전 해당 디렉터리로 이동
command	**yes**			수행할 명령
creates	no			파일이름 또는 (2.0이후) glob 패턴으로 만약 존재한다면 아무런 동작 안함
echo	no			응답 문자열을 출력할 것인가의 플래그
removes	no			파일이름 또는 (2.0이후) glob 패턴으로 만약 존재한다면 아무런 동작 안함
responses	**yes**			기대하는 문자열/정규식과 응답 문자열의 매핑. 만약 응답이 목록이면 매칭되는 것을 연속하여 매칭해봄. list 기능은 2.1 이후 추가됨
timeout	no	30		기대되는 문자열이 나타나기 기다는 시간(초)

예제

```
# Case insensitve password string match
- expect:
    command: passwd username
    responses:
      (?i)password: "MySekretPa$$word"

# Generic question with multiple different responses
- expect:
    command: /path/to/custom/command
    responses:
      Question:
        - response1
        - response2
        - response3
```

raw (핵심 모듈)

설명

모듈 하위시스템을 거치지 않고 바로 SSH 명령을 수행합니다. 이것은 다음과 같은 두 가지 경우에 필요합니다. 첫번째는 파이썬 2.4 이전 버전이 호스트에 설치되어 있어 `python-simplejson` 이 필요한 경우입니다. 또 다른 경우는 라우터 등과 같이 파이썬 인터프리터가 설치되어 있지 않은 시스템인 경우 입니다. 그 밖의 경우에는 `shell` 또는 `command` 모듈이 맞습니다. `raw` 의 인자는 원격 쉘에서 수행하는 것과 동일한 인자를 넣어줍니다. 표준 출력, 에러, 리턴 코드 등이 리턴됩니다. 이 모듈은 변경 핸들러를 달 수 없습니다. 이 모듈은 `script` 모듈처럼 원격 시스템에서 파이썬을 필요로 하지 않습니다.

옵션

패러미터	필수	디폴트	선택	비고
executable	no			수행할 명령의 shell 변경. 절대경로 필요. 만약 `become` 권한 상승을 한다면 디폴트 쉘 이용.
free_form	**yes**			명령을 수행할 free form 지정.

예제

```
# Bootstrap a legacy python 2.4 host
- raw: yum -y install python-simplejson

# Bootstrap a host without python2 installed
- raw: dnf install -y python2 python2-dnf libselinux-python

# Run a command that uses non-posix shell-isms (in this example /bin/sh
# doesn't handle redirection and wildcards together but bash does)
- raw: cat < /tmp/*txt
  args:
    executable: /bin/bash
```

script (핵심 모듈)

설명

해당 script와 공백 분리된 인자들로 실행됩니다. 이 로컬 스크립트는 원격 노드로 복사되고 실행됩니다. 주어진 스크립트는 원격 노드에서 쉘 환경에서 실행됩니다. 이 모듈은 `raw` 모듈 처럼 원격 시스템에 파이썬 인터프리터를 필요로 하지 않습니다.

옵션

패러미터	필수	디폴트	선택	비고
creates (1.5 이후)	no			생성할 파일이름, 만약 존재한다면 아무런 동작 안함
free_form	**yes**			선택 인자를 포함한 로컬 스크립트 경로
removes (1.5 이후)	no			삭제할 파일이름, 만약 존재한다면 아무런 동작 안함

예제

```
# Example from Ansible Playbooks
- script: /some/local/script.sh --some-arguments 1234

# Run a script that creates a file, but only if the file is not yet created
- script: /some/local/create_file.sh --some-arguments 1234 creates=/the/created/file

# Run a script that removes a file, but only if the file is not yet removed
- script: /some/local/remove_file.sh --some-arguments 1234 removes=/the/removed/file
```

shell (핵심 모듈)

설명

해당 명령과 공백 분리된 인자들로 실행됩니다. `command` 모듈과 거의 동일합니다만 원격 노드에서 ***/bin/sh*** 를 이용한 쉘에서 동작하는 것이 틀립니다.

옵션

패러미터	필수	디폴트	선택	비고
chdir	no			해당 명령을 실행하기 전 해당 디렉터리로 이동
creates	no			파일이름 만약 존재한다면 아무런 동작 안함
executable	no			명령을 실행하는데 사용할 shell 변경. 절대경로로 표현
free_form	**yes**			명령을 수행할 free form 지정. 실제 `free form` 패러미터 없음
removes	no			파일이름 만약 존재한다면 아무런 동작 안함
warn (1.8 이후)	no	True		만약 ansible.cfg에 명령 warnings가 on 되어 있더라도 이 패러미터가 no/false 로 되어 있다면 경고 안함

예제

```
# Execute the command in remote shell; stdout goes to the specified
# file on the remote.
- shell: somescript.sh >> somelog.txt

# Change the working directory to somedir/ before executing the command.
- shell: somescript.sh >> somelog.txt chdir=somedir/

# You can also use the 'args' form to provide the options. This command
# will change the working directory to somedir/ and will only run when
# somedir/somelog.txt doesn't exist.
- shell: somescript.sh >> somelog.txt
  args:
    chdir: somedir/
    creates: somelog.txt

# Run a command that uses non-posix shell-isms (in this example /bin/sh
# doesn't handle redirection and wildcards together but bash does)
- shell: cat < /tmp/*txt
  args:
    executable: /bin/bash
```

리턴 값

이름	설명	리턴 여부	리턴형	샘플
cmd	태스크에 의해 수행된 명령	항상	string	rabbitmqctl join_cluster rabbit@master
start	명령 시작 시각	항상	string	2016-02-25 09:18:26.429568
end	명령 종료 시각	항상	string	2016-02-25 09:18:26.755339
delta	명령 실행 시간	항상	string	0:00:00.325771
stdout	명령 표준 출력	항상	string	Clustering node rabbit@slave1 with rabbit@master ...
stdout_lines	표준 출력의 라인 목록	항상	list of string	["u'Clustering node rabbit@slave1 with rabbit@master ...'"]
stderr	명령 오류 출력	항상	string	ls: cannot access foo: No such file or directory
rc	명령 리턴 코드 (0은 OK)	항상	int	0
msg	변경됨	항상	boolean	True

데이터베이스 모듈

파일 모듈

Identity 모듈

Inventory 모듈

메시징 모듈

모니터링 모듈

네트워크 모듈

알림 모듈

패키징 모듈

원격관리 모듈

소스관리 모듈

저장 모듈

시스템 모듈

모듈명	핵심/확장	설명
`alternatives`	확장	일반 명령의 대체 프로그램 지정 (링크)
`at`	확장	at 명령을 통하여 특정 시각에 명령 수행
`authorized_keys`	**핵심**	SSH 인증 키를 추가하거나 삭제
`capabilities`	확장	Linux capabilities(7) 를 통한 권한 관리
`cron`	**핵심**	cron.d crontab 관리
`cronvar`	확장	crontab에 변수 관리
`crypttab`	확장	Linux 블럭 디바이스 암호화
`debconf`	확장	.deb 패키지 설정
`facter`	확장	원격 시스템에서 *facter* fact 발견 프로그램 실행
`filesystem`	확장	블락 디바이스에 파일 시스템 만들기
`firewalld`	확장	임의의 포트/서비스 등에 대한 firewalld 관리
`getent`	확장	UNIX *getent* 래퍼
`gluster_volume`	확장	FlusterFS 볼륨 관리
`group`	**핵심**	group 추가 또는 삭제
`host`	**핵심**	호스트명 관리

`hostname`	**핵심**	호스트명 관리
`iptables`	확장	시스템 iptables 변경
`kernel_blacklist`	확장	Blacklist 커널 모듈
`known_hosts`	확장	**known_hosts** 파일에 특정 호스트 추가 또는 삭제
`locale_gen`	확장	locale 추가 또는 삭제
`lvg`	확장	LVM 볼륨 그룹 설정
`lvol`	확장	LVM 로지컬 볼륨 설정
`make`	확장	Makefile 실행
`modprobe`	확장	커널 모듈 추가 또는 삭제
`mount`	**핵심**	mount 포인트 설정 및 활성화 제어
`ohai`	확장	*Ohai (Chef)* 에서 인벤토리 데이터 구하기
`open_iscsi`	확장	open-iscsi를 이용한 iscsi 타겟 관리
`osx_defaults`	확장	osx_defaults는 ansible 에서 Mac OS X 사용자가 defaults 를 읽고, 쓰고 삭제하는 등의 기능 수행
`pam_limits`	확장	Linux PAM limits 변경
`ping`	**핵심**	타겟 호스트에 접속하여 파이썬이 가능한가 확인하고 성공 시 *pong* 리턴
`puppet`	확장	puppet 실행
`seboolean`	**핵심**	SELinux 블리언 토글
`sefcontext`	확장	SELinux파일 컨텍스트 매핑 정의 관리
`selinux`	**핵심**	SELinux의 정책 및 상태 변경
`selinux_permissive`	확장	SELinux 정책에서 permissive 도메인 변경
`seport`	확장	SELinux 네트워크 포트 형태 정의 관리
`sevice`	**핵심**	services 관리
`setup`	**핵심**	원격 호스트에서 facts 모음
`solaris_zone`	확장	Solaris 존 관리
`svc`	확장	daemontools 서비스 관리

`sysctl`	**핵심**	sysctl.conf 에 있는 항목 관리
`systemd`	**핵심**	서비스 관리
`timezone`	확장	타임존 설정 관리
`ufw`	확장	UFW를 이용한 방화벽 관리
`user`	**핵심**	사용자 계정 관리
`zfs`	확장	zfs 관리

Univention 모듈

유틸리티 모듈

Helper

모듈명	핵심/확장/이전	설명
`accelerate`	*이전*	원격 노드에서 가속 모드 활성화
`fireball`	*이전*	원격 노드에서 fireball 모드 활성화
`meta`	**핵심**	Ansible *action* 수행

Logic

모듈명	핵심/확장	설명
`assert`	**핵심**	사용자 메시지로 실패 유발
`async_status`	**핵심**	비동기 태스크의 상태 구하기
`debug`	**핵심**	실행 시 상태 출력
`fail`	**핵심**	사용자 메시지로 실패
`include_role`	**핵심**	role 로드 및 실행
`include_vars`	**핵심**	(태스크에서 동적으로) 파일에서 변수 로드
`pause`	**핵심**	플레이북 실행을 잠시 멈춤
`set_fact`	**핵심**	태스크에서 호스트 fact 설정
`wait_for`	**핵심**	실행이 계속되기 전 특정 상태까지 기다림

웹 기반 환경 모듈

윈도우 모듈

Getting Started with Docker

Ansible 은 Docker 컨테이너를 지휘하기 위하여 다음과 같은 모듈을 제공합니다.

- `docker_service` : 단일 다커 데몬 또는 Swarm에서 컨테이너를 지휘하기 위하여 이미 존재하는 다커 compose 파일 이용. compose 버전 1,2를 지원
- `docker_container` : 컨테이너의 생성, 변경, 멈춤, 시작, 삭제 등의 컨테이너 라이프 사이클 관리
- `docker_image` : 다커 이미지를 생성, pull, push, tag, remove 등을 통한 이미지 관리
- `docker_image_facts` : 다커 호스트 이미지 캐쉬에서 하나 이상의 이미지를 조사하여 플레이북의 fact 정보로 이용할 수 있도록 함
- `docker_login` : 다커 허브 또는 다른 다커 저장소의 인증을 위한 다커 엔진 설정 파일 갱신
- `docker (동적 인벤토리)` : 하나 이상의 다커 호스트에 사용가능한 모든 컨테이너를 이용하여 인벤토리 생성

Ansible 2.1 이후부터 위와 같은 다커 모듈이 추가되었고 컨테이너 지휘가 가능합니다. 위의 모듈 외에도 다음과 같은 것을 작업하고 있습니다.

이미지를 생성하기 위하여 *Dockerfile*을 아직도 사용하고 계십니까? ansible-container를 확인해 보십시오. Ansible 플레이북으로 이미지를 생성하는 것이 가능합니다.

OpenShift에 있는 docker-compose 를 실행하기 위하여 ansible-container에 있는 `shipit` 명령어를 이용해 보십시오. 많은 수고를 하지 않아도 단순한 랩탑 응용프로그램을 클라우드에 확장 가능한 응용프로그램으로 전환 가능합니다.

기타 다커 관련 아이디어나 계획 등에 대해서는 해당 저장소에 있는 문서를 참고하십시오.

요구사항

docker를 이용하기 위해서는 docker-py 1.7.0 이후 버전이 설치되어 있어야 합니다.

```
$ pip install 'docker-py>=1.7.0'
```

`docker_service` 모듈은 docker-compose 모듈도 필요로 합니다.

```
$ pip install 'docker-compose>=1.7.0'
```

Docker API 연동

로컬 또는 원격 API 연결하여 패러미터를 전달하고 하는 것을 환경 변수로 할 수 있습니다. 우선 순위는 명령행에 주는 것이 우선이고 그 다음이 환경변수 입니다. 만약 명령행 인자와 환경변수 모두 발견되지 않으면 디폴트 값을 사용합니다.

패러미터

- `docker_host` : Docker API 에 접속하기 위하여 URL 또는 Unix socker 경로가 사용됩니다. 디폴트는 ***unix://var/run/docker.sock*** 입니다. 원격 서비스에 접속하기 위해서는 ***tcp://192.0.2.23:2376*** 와 같이 TCP 연결을 합니다. 만약 TLS를 이용한 암호화된 통신을 하려면 *tcp* 대신 *https* 를 이용합니다.
- `api_version` : Docker 호스트에서 작동하는 API의 버전인데 디폴트는 docker-py가 지원하는 API의 마지막 버전 입니다.
- `timeout` : API 를 호출하고 응답을 기다리는 타임아웃(초) 입니다. 디폴트는 60초 입니다.
- `tls` : TLS 암호화 통신으로 API 연결을 하는 플래그입니다. 디폴트는 False 입니다.
- `tls_verify` : Docker 호스트 서버를 인증하지 않할지 나타내는 플래그 입니다. 디폴트는 False 입니다.
- `cacert_path` : CA 인증서 파일의 경로입니다.
- `cert_path` : TLS 인증서 파일의 경로입니다.
- `key_path` : TLS 키파일의 경로입니다.
- `tls_hostname` : Docker 호스트 서버의 유효성을 검증하는 경우 서버 이름을 지정합니다. 디폴트는 *localhost* 입니다.
- `ssl_version` : SSL 버전입니다. docker-py에 의해 결정됩니다.

환경 변수

Docker API를 제어하는데 사용되는 환경변수 입니다.

- `DOCKER_HOST` : Docker API 에 접속하기 위하여 URL 또는 Unix socker 경로가 사용됩니다.
- `DOCKER_API_VERSION` : Docker 호스트에서 작동하는 API의 버전인데 디폴트는 docker-py가 지원하는 API의 마지막 버전 입니다.
- `DOCKER_TIMEOUT` : API 를 호출하고 응답을 기다리는 타임아웃(초) 입니다.
- `DOCKER_CERT_PATH` : CA 인증서 파일의 경로입니다.
- `DOCKER_SSL_VERSION` : SSL 버전입니다.
- `DOCKER_TLS` : TLS 암호화 통신으로 API 연결을 하는 플래그입니다.
- `DOCKER_TLS_VERIFY` : Docker 호스트 서버를 인증하지 않할지 나타내는 플래그 입니다.

동적 인벤토리 스크립트

인벤토리 스크립트는 하나 이상의 Docker API 서비스를 이용하여 수집된 동적 인벤토리를 생성합니다. 이것은 정적 파일이 아닌 API를 수행하는 시간에 생성되므로 동적 개념입니다. 각각의 다커 호스트가 제공하는 컨테이너를 보고 정보를 얻어옵니다. 접속할 API 스크립트는 환경 변수 또는 설정 파일에 정의됩니다.

Groups

이 스크립트는 다음과 같은 호스트 그룹을 생성합니다.

- 컨테이너 ID
- 컨테이너 이름
- 컨테이너 short ID
- 이미지 이름 (*image_<이미지이름>*)
- docker_host

- running
- stopped

예제

```
# Connect to the Docker API on localhost port 4243 and format the JSON output
DOCKER_HOST=tcp://localhost:4243 ./docker.py --pretty

# Any container's ssh port exposed on 0.0.0.0 will be mapped to
# another IP address (where Ansible will attempt to connect via SSH)
DOCKER_DEFAULT_IP=192.0.2.5 ./docker.py --pretty

# Run as input to a playbook:
ansible-playbook -i ~/projects/ansible/contrib/inventory/docker.py docker_inventory_
```

```
# Simple playbook to invoke with the above example:

    - name: Test docker_inventory
      hosts: all
      connection: local
      gather_facts: no
      tasks:
        - debug: msg="Container - {{ inventory_hostname }}"
```

개발 정보

모듈 개발

Ansible 모듈은 재사용 가능하고 Ansible API, ansible 또는 ansible-playbook 등을 아용하여 단독 실행가능합니다. 이 모듈은 종료되기 전 표준 출력(*stdout*)으로 JSON 문자열을 출력하여 ansible에 결과를 리턴합니다. 그런데 모듈의 인자는 앞으로 설명할 방법에 따라 설명됩니다.

모듈은 `ANSIBLE_LIBRARY` 또는 명령줄의 `--module-path` 인자에 지정된 위치에 어떤 언어로도 상관없이 있을 수 있습니다.

디폴트로 Ansible의 소스트리에 모든 핵심 및 확장 모듈이 포함되어 있지만 추가 모듈이 다른 곳에 추가될 수도 있습니다.

최상위 플레이북과 더불어 ansible이 실행되는 `./library` 디렉터리 또한 자동으로 모듈을 찾는 디렉터리가 됩니다.

흥미로운 Ansible 모듈을 개발한다면 modules-extra project에 제안해 보시기 바랍니다. 또한 핵심 모듈을 위한 저장소도 있습니다. **Extras** 모듈 중에서 어던 것들은 **핵심** 모듈이 되기도 하지만 이런 모듈 사이의 차이는 없습니다.

튜토리얼

시스템의 시간을 구하거나 설정하는 매우 기본적인 모듈에서 시작해 봅시다. 초보자로서 현제 시각을 출력하는 모듈을 만드는 겁니다.

이곳에서 파이썬 언어를 이용하지만 어떤 언어라도 상관없습니다. 단지 파일 입출력 및 결과를 표준출력으로 리턴하면 됩니다. bash, C++ clojure, Ruby 등 어떤 언어도 좋습니다.

모든 핵심 모듈이 그렇듯이 아주 간단한 방법으로 모듈을 만들 수 있지만 우선은 어려운 방법을 설명하겠습니다. 이렇게 하는 이유는 파이썬이 아닌 다른 언어에서도 작성하는 것을 돕기 위함입니다. 그 다음에 쉬운 방법을 살펴보겠습니다.

`command` 모듈을 사용하면 되므로 실제로 사용할 이유는 없고 튜토리얼 용 입니다.

실제 Ansible의 모듈을 살펴보는 것이 어떻게 모듈을 만들 것이가 하는 좋은 방법입니다. 하지만 Ansible 소스 트리에서 `service` 나 `yum` 대신 `async_wrapper` 같은 모듈은 살펴보지 마십시오. 너무 ansible 내부 적인 구동과 관련이 많이 되어 있습니다.

좋습니다. 예제를 시작하지요. `timetest.py` 라는 파이썬 프로그램을 작성합니다.

```
#!/usr/bin/python

import datetime
import json

date = str(datetime.datetime.now())
print json.dumps({
    "time" : date
})
```

모듈 테스트

Ansible 소스를 체크아웃 하여 실행하면,

```
$ git clone git://github.com/ansible/ansible.git --recursive
$ source ansible/hacking/env-setup
```

위에 작성한 모듈을 테스트 해 봅니다.

```
$ ansible/hacking/test-module -m ./timetest.py
```

그러면 결과는 다음처럼

```
{'time': '2012-03-14 22:13:48.539183'}
```

나옵니다.

입력 읽기

이제는 현재 시각을 설정하도록 해 보겠습니다. 해당 모듈에 *time=* 과 같은 형식으로 입력되게 해 보겠습니다.

Ansible은 내부적으로 인자를 인자 파일에 저장합니다. 따라서 파일을 읽고 파싱하도록 합니다. 인자 파일은 단순히 문자열로 구성되어 있어 어떤 형식의 문자열도 좋습니다. 하지만 일반적인 방법인 *key=value*를 사용하겠습니다.

다음과 같이 인자가 되어 있습니다.

```
time time="March 14 22:10"
```

만약 인자 없이 모듈을 호출하면 앞에서처럼 현재 시각을 리턴합니다.

> **노트:** 실제로는 `command` 모듈을 이용하면 되기에 이런 모듈을 작성하는 사람은 없겠지만 그래도 좋은 튜토리얼 입니다.

이제 코드를 살펴보겠습니다. 설명을 위하여 최대한 코멘트를 많이 달았습니다.

```
#!/usr/bin/python

# import some python modules that we'll use.  These are all
# available in Python's core

import datetime
import sys
import json
import os
import shlex

# read the argument string from the arguments file
args_file = sys.argv[1]
args_data = file(args_file).read()

# For this module, we're going to do key=value style arguments.
# Modules can choose to receive json instead by adding the string:
#   WANT_JSON
# Somewhere in the file.
# Modules can also take free-form arguments instead of key-value or json
```

```
# but this is not recommended.

arguments = shlex.split(args_data)
for arg in arguments:

    # ignore any arguments without an equals in it
    if "=" in arg:

        (key, value) = arg.split("=")

        # if setting the time, the key 'time'
        # will contain the value we want to set the time to

        if key == "time":

            # now we'll affect the change.  Many modules
            # will strive to be 'idempotent', meaning they
            # will only make changes when the desired state
            # expressed to the module does not match
            # the current state.  Look at 'service'
            # or 'yum' in the main git tree for an example
            # of how that might look.

            rc = os.system("date -s \"%s\"" % value)

            # always handle all possible errors
            #
            # when returning a failure, include 'failed'
            # in the return data, and explain the failure
            # in 'msg'.  Both of these conventions are
            # required however additional keys and values
            # can be added.

            if rc != 0:
                print json.dumps({
                    "failed" : True,
                    "msg"    : "failed setting the time"
                })
                sys.exit(1)

            # when things do not fail, we do not
            # have any restrictions on what kinds of
            # data are returned, but it's always a
            # good idea to include whether or not
            # a change was made, as that will allow
            # notifiers to be used in playbooks.

            date = str(datetime.datetime.now())
            print json.dumps({
                "time" : date,
                "changed" : True
            })
            sys.exit(0)

# if no parameters are sent, the module may or
# may not error out, this one will just
```

```
# return the time

date = str(datetime.datetime.now())
print json.dumps({
    "time" : date
})
```

다시 테스트를 해 보면

```
$ ansible/hacking/test-module -m ./timetest.py -a "time=\"March 14 12:23\""
```

결과는,

```
{"changed": true, "time": "2012-03-14 12:23:00.000307"}
```

와 같이 나옵니다.

바이너리 모듈 입력

Ansible 2.2 이후부터는 바이너리 입력을 지원합니다. Ansible은 바이너리 모듈 임을 알게되면 `argv[1]` 에 파일이름 대신 JSON으로 전달된 인자를 바로 파싱합니다. 이것은 다음의 `ping` 모듈에서 받아 들이는 인자와 같은 형태를 갖습니다.

```
{
    "data": "pong",
    "_ansible_verbosity": 4,
    "_ansible_diff": false,
    "_ansible_debug": false,
    "_ansible_check_mode": false,
    "_ansible_no_log": false
}
```

'Facts' 제공 모듈

ansible 에서 제공하는 원격 호스트에서 facts를 구하는 setup 모듈은 플레이북과 템플릿에서 사용되는 대상 시스템의 많은 변수를 구해옵니다. 그럼에도 불구하고 시스템 모듈을 수정하지 않고 자신만의 facts를 추가하는 것이 가능합니다. 이렇게 하기위하여 *ansible_facts* 라는 키를 갖는 결과를 리턴하는 모듈을 만듭니다.

```
{
    "changed" : True,
    "rc" : 5,
    "ansible_facts" : {
        "leptons" : 5000,
        "colors" : {
            "red"   : "FF0000",
            "white" : "FFFFFF"
        }
    }
}
```

이런 `facts` 는 플레이북에서 해당 모듈이 호출되고 나서 사용가능합니다. 비록 Ansible에서도 핵심 facts의 선택를 향상시키기 위하여 열더라도, 각 플레이북의 상단에 `site_facts` 라고 불리우는 모듈을 포함하도록 하여 사설 정의한 facts를 가져오도록 하는 것이 좋습니다.

공통 모듈 반복 재사용

미리 언급되었듯이 만약 파이썬으로 모듈을 작성한다면 이전보다 쉬운 방법이 있다고 하였습니다. 모듈은 여전히 하나의 파일로 전송되지만 인자 파일은 더이상 필요하지 않으므로 코드의 양이 적어지고 수행시 더 빠르게 진행됩니다.

`group` 과 `user` 라는 모듈이 이런 방법을 보여주는 좋은 예 입니다.

`ansible.module_utils.basic` 라는 파이썬 모듈을 이용합니다.

```
from ansible.module_utils.basic import AnsibleModule
if __name__ == '__main__':
    main()
```

> **노트:** Ansible 2.1.0 이전 버전에는 단순히 `ansible.module_utils.basic` 만 import 해서는 동작하지 않아고 다음과 같이
>
> `from ansible.module_utils.basic import *`
>
> 모든 하위 모듈을 가져와야 했습니다.

그리고 실제 모듈 클래스는 다음과 같이 작성됩니다.

```
def main():
    module = AnsibleModule(
        argument_spec = dict(
            state     = dict(default='present', choices=['present', 'absent']),
            name      = dict(required=True),
            enabled   = dict(required=True, type='bool'),
            something = dict(aliases=['whatever'])
        )
    )
```

`AnsibleModule` 은 인자를 파싱하고 입력을 체크하며 결과를 처리하는 등의 공통 코드의 많은 부분을 제공합니다.

성공적인 리턴은 다음과 같이 구성됩니다.

```
module.exit_json(changed=True, something_else=12345)
```

반면 실패는 다음과 같이 간단합니다. (`msg` 는 오류를 설명하는 문자열입니다)

```
module.fail_json(msg="Something fatal happened")
```

또한 `module.sha1(path)()` 와 같은 모듈 클래스의 유용한 함수가 있습니다. 그 구현 내용이 궁금하다면 `lib/ansible/module_utils/basic.py` 해당 소스를 확인하십시오.

이전 방법의 모듈 테스트와 마찬가지로 `hacking/test-module` 스크립트를 이용하십시오. 매직과도 같은 이 방법이 Ansible 이외에 모듈이 동작하는지 확인하는 유일한 방법입니다.

만약 Ansible의 핵심 모듈로 등록하려면 `AnsibleModule` 를 이용하기를 권고합니다.

체크 모드

버전 1.1 이후.

모듈은 선택적으로 체크 모드를 제공할 수 있습니다. 만약 사용자가 체크모드에서 Ansible을 구동시키면 해당 모듈은 변화가 발생하는지 예상하게 됩니다.

체크 모드를 지원하는 모듈인 경우 `AnsibleModule` 클래스의 인스턴스를 생성할 때 `supports_check_mode=True` 라는 인자를 추가해야 합니다.

다음 예와 같습니다.

```
module = AnsibleModule(
    argument_spec = dict(...),
    supports_check_mode=True
)

if module.check_mode:
    # Check if any changes would be made but don't actually make those changes
    module.exit_json(changed=check_if_system_state_would_be_changed())
```

위와 같이 해당 모듈의 ***check_Mode*** 플래그로 체크 모드인지 아닌지 확인가능합니다.

빠지기 쉬운 함정

해당 모듈에서는 절대,

```
print "some status message"
```

와 같이 표준 출력으로 메시지를 출력하지 마십시오.

모듈은 반드시 정해진 형식의 JSON 결과를 표준 출력으로 출력해야 하기 때문에 디버깅을 위하여 위와 같이 명령을 내린다면 결과의 JSON 파싱에 실패하게 될 것입니다. 또한 표준 오류 (stderr) 로도 출력하지 마십시오. 표준 출력과 표준 오류는 결과가 병합되기 때문입니다.

만약 모듈이 stderr로 리턴하거나 오류가 발생할 JSON을 리턴한다면 그 실제 결과는 Ansible에서 보이지만 해당 명령은 성공하지 못할 겁니다.

파일에 바로 출력하지 마십시오. 임시 파일을 이용하고 **ansibile.module_utils.basic**에 있는 *atomic_move* 함수를 이용하여 임시 파일을 올바른 위치로 이동시키십시오. 이렇게 하여야만 데이터가 깨지지 않고 파일에 저장된 데이터가 올바름을 보장하게 됩니다.

다른 모듈에서 작업된 것을 가져다가 재사용하는 모듈을 지양하십시오. 이것은 코드의 중복 비슷한 코드의 다양한 버전 및 유지보수에 어려움을 가져올 것입니다. 유사 모듈을 다시 작성하는 것 보다는 플레이와 Role 을 사용하시기 바랍니다.

`캐쉬` 생성을 피하십시오. Ansible은 중앙 서버를 통해 동작되도록 하지 않았기 때문에 서로 다른 권한이나 옵션 또는 파일 위치 등에서 실행될 수 있습니다. 만약 중앙 통제를 원한다면 최상위 Ansible (Ansible Tower와 같은)에서 실행하십시오 . 이런 해결책을 모듈로 찾지 마십시오.

모듈 개발을 할 때에는 항상 `hacking/test-module` 스크립트를 잘 이용하십시오.

규약/권고안

다음가 같은 가이드라인이 존재합니다.

- 만약 모듈이 객체를 참조하면 해당 객체의 인자는 가능할 때마다 ***이름*** 또는 이름의 가명으로 호출됩니다.
- 만약 설치했던 특정 facts를 리턴하는 회사 모듈을 가지고 있다면 이 모듈의 이름은 *site_facts* 처럼 사용합니다.
- *불리언* 상태를 받는 모듈은 `yes` , `no` , `true` , `false` 를 받아야 합니다.
- 가능한한 의존성을 최소화 합니다. 만약 의존성이 있다면 최상위 모듈의 문서에 그것을 기술해야 합니다. 그리고 *import* 가 실패하면 그에 따른 적절한 오류 메시지를 JSON 결과로 리턴해야 합니다.
- 모듈은 모든 실행 코드가 단일 파일로 존재하는 형태로 되어 있어야 합니다. 원격에서 하나의 파일이 전달되어 실행되는 형식이기 때문입니다.
- 만약 RPM에 모듈을 패키징 한다면 제어 머신의 `/usr/share/ansible` 에 해당 모듈을 넣도록 합니다. (해당 위치는 정하기 나름입니다)
- 모듈의 출력은 정해진 JSON으로만 지정됩니다. 출력의 최상위 결과는 중첩가능한 해쉬(사전) 형식입니다. 단순 스칼라 값 혹은 목록의 결과는 허용되지 않습니다. 간단한 해쉬(사전)에 스칼라 값이나 목록을 넣도록 합니다.
- 모듈이 실행되다 시패한 경우에는 리턴 해쉬 값에 해당 ***msg*** 에 실패 원인을 기술하는 `failed` 키가 포함되도록 합니다. 단순 *traceback* (*stacktrace*) 를 출력하는 모듈은 *빈약(poor)* 한 모듈입니다. 물론 JSON이 아닌 *traceback* 형식을 리턴하더라도 ansible을 해당 오류를 출력하기는 합니다. 파이썬으로 작성된 보통 Ansible 모듈을 이용한다면 `fail_json` 을 호출할 때 자동으로 `failed` 항목이 포함됩니다.
- 모듈의 리턴 코드는 실제로 그렇게 중요하지는 않지만 장차 사용될 경우를 고려하여 `0 = 성공` 을 의미하고 `0이 아닌값 = 실패` 를 위미 하도록 합니다.
- 많은 관리 대상 호스트가 동시에 결과를 보내주기 때문에 적당한 크기의 출력을 하도록 합니다. 너무 많은 결과를 리턴하면 그만큼 처리하는데 시간이 걸립니다.

모듈 문서화

CORE 모듈을 포함한 모든 Ansible 모듈은 `DOCUMENTATION` 이라는 글로벌 문자열을 가집니다. 이 문자열은 아래에 기술된 것과 같은 YAML 형식의 문자열 이어야 합니다.

예제

DOCUMENTATION 예제 와 같이 작성하는데,

```
#!/usr/bin/python
# Copyright header....

DOCUMENTATION = '''
---
module: modulename
short_description: This is a sentence describing the module
# ... snip ...
'''
```

와 같은 식입니다.

만약 인자가 `C(True)/C(False)` 와 `C(Yes)/C(No)` 를 받는다면 문서는 `C(True)/C(False)` 를 사용합니다.

`description` 과 `notes` 필드는 특별한 매크로 형식을 지원합니다.

URL, 모듈, 이텔릭체 그리고 고정너비의 의미로 각각 `U(), M(), I(), C()` 과 같은 포맷 함수들을 이용합니다. 파일과 옵션 이름으로는 `C()` , 패러미터는 `I()` , 그리고 모듈이름은 `M(module)` 과 같이 사용합니다.

콜론, 따옴표 등과 같이 YAML로 표기하기 힘든 문자들을 포함한 예제는 다음과 같이 `EXAMPLES` 문자열 안에 표현합니다.

```
EXAMPLES = '''
- action: modulename opt1=arg1 opt2=arg2
'''
```

`EXAMPLES` 섹션은 DOCUMENTATION 섹션과 같이 새로 추가되는 모든 모듈에서 기술해 주는 것이 좋습니다.

`RETURN` 섹션은 모듈이 리턴하는 것을 기술하는 부분으로 각각의 리턴 값을 기술하는데 `description` 은 해당 값의 설명을, `returned` 는 어떤 상황에서 리턴되는 가를, `type` 은 해당 값 형태를, `sample` 은 값 예시 등을 나타냅니다.

예를 들어 ***copy*** 모듈을 예로 보면 다음과 같습니다.

```
RETURN = '''
dest:
    description: destination file/path
    returned: success
    type: string
    sample: "/path/to/file.txt"
src:
    description: source file used for the copy on the target machine
    returned: changed
    type: string
    sample: "/home/httpd/.ansible/tmp/ansible-tmp-1423796390.97-147729857856000/source"
md5sum:
    description: md5 checksum of the file after running copy
    returned: when supported
    type: string
    sample: "2a5aeecc61dc98c4d780b14b330e3282"
...
'''
```

building 및 테스트

완성된 모듈을 *library* 디렉터리에 넣어 `make_webdocs` 명령을 수행합니다. 새로운 *docsite/* 폴더가 생긴 것이 보이고 그 안에 *modules.html* 파일이 만들어 집니다.

> **팁!** 만약 YAML 문법 오류가 발생하면 YAML link 를 이용하여 해결하도록 합니다.
>
> **팁!** `ANSIBLE_KEEP_REMOTE_FILES=1` 이라고 환경변수를 설정하여 원격 머신에서 해당 모듈을 실행하고 그 결과를 삭제하지 않음으로 해서 직접 원격에서 테스트할 수 있습니다.

Ansible 모듈 디버깅

> **팁!** `hacking/test-module` 스크립트를 이용하여 테스트를 할 경우에는 많은 정보를 제공해 주지만 직접 원격 호스트에서 해당 모듈이 실행되고 있고 이 경우 디버깅을 하려고 하면 어려울 수 있습니다. 이런 경우에 이용합니다.

Ansible 2.1.0 이후 모듈은 단일파일 대신 여러 파일이 하나의 zip 파일로 묶여 전송되고 해제되어 실행됩니다.

다음과 같이 `ANSIBLE_KEEP_REMOTE_FILES` 를 이용하여 실행하십시오.

```
$ ANSIBLE_KEEP_REMOTE_FILES=1 ansible localhost -m ping -a 'data=debugging_session' -vvv
<127.0.0.1> ESTABLISH LOCAL CONNECTION FOR USER: badger
<127.0.0.1> EXEC /bin/sh -c '( umask 77 && mkdir -p "` echo $HOME/.ansible/tmp/ansible-tmp-14614347
<127.0.0.1> PUT /var/tmp/tmpjdbJ1w TO /home/badger/.ansible/tmp/ansible-tmp-1461434734.35-235318071
<127.0.0.1> EXEC /bin/sh -c 'LANG=en_US.UTF-8 LC_ALL=en_US.UTF-8 LC_MESSAGES=en_US.UTF-8 /usr/bin/p
localhost | SUCCESS => {
    "changed": false,
    "invocation": {
        "module_args": {
            "data": "debugging_session"
        },
        "module_name": "ping"
    },
    "ping": "debugging_session"
}
```

일반적인 경우 모듈은 원격 서버에서 수행되고 종료되면 삭제되는데 이를 `ANSIBLE_KEEP_REMOTE_FILE` 을 `1` 로 설정함으로써 원격에서 수행되고 난 뒤에 삭제되지 않도록 합니다. 또한 Ansible에 `-vvv` 옵션을 주어 더 많은 메시지를 출력하도록 합니다. 그러면 원격에서 실행되는 모듈의 임시 파일 이름을 출력합니다.

만약 wrapper 파일을 조사하려고 한다면 크고 base64 인코딩 된 문자열을 변환하는 작은 파이썬 코드를 보여줍니다. 해당 문자열은 실행되는 모듈을 포함합니다.

```
$ python /home/badger/.ansible/tmp/ansible-tmp-1461434734.35-235318071810595/ping explode
Module expanded into:
/home/badger/.ansible/tmp/ansible-tmp-1461434734.35-235318071810595/debug_dir
```

만약 *debug_dir* 아래의 구조를 확인해 보면,

```
├── ansible_module_ping.py
├── args
└── ansible
    ├── __init__.py
    └── module_utils
        ├── basic.py
        └── __init__.py
```

와 같이 구성되어 있음을 볼 수 있습니다.

- `ansible_module_ping.py` 는 모듈 그 자체 코드입니다. *ansiblemodule* 이라는 것을 앞에 붙여 다른 모듈과 헷갈리지 않도록 합니다. 해당 모듈을 수정하여 어떤 결과가 나오는지 확인하여 디버깅하도록 합니다.
- `args` 파일은 JSON 문자열의 입력되는 값입니다. 해당 해쉬는 모듈 인자를 포함하여 Ansible 자체에서 전달해주는 다양한 인자를 모두 포함합니다. 만약 모듈에 입력되는 값을 변경하고 싶다면 해당 입력 JSON 파일을 수정하여 실행해 봅니다.
- `ansible` 디렉터리는 모듈에 의해 사용되는 `ansible.module_utils` 모듈 디렉터리를 포함합니다.

모듈 패스

만약 ansible이 모듈을 가져오는데 실패하면 `ANSIBLE_LIBRARY` 환경 변수에 있는지 확인합니다.

```
ANSIBLE_LIBRARY=~/ansible-modules-core:~/ansible-modules-extras
```

과 같이 지정할 수 있습니다.

작업 모듈을 Ansible 에 넣기

최소한의 의존성을 갖고 잘 만들어진 모듈이 Ansible에 포함되지만 종종 파이썬으로 구현되고 AnsibleModule 일반 코드를 사용하며 변함없는 인자를 이용하기도 합니다.

모듈 체크리스

원본 참조

플러그인 개발

플러그인은 Ansible의 핵심 기능에 더해서 코드를 덧붙이는 것을 의미합니다. Ansible은 이미 여러 플러그인을 포함하고 있으며 자신만의 플러그인을 쉽게 추가할 수 있습니다.

다음과 같은 플러그인 형태가 있습니다.

- `Callback` 플러그인은 디스플레이 또는 로깅 목적의 Ansible 이벤트를 변경할 경우입니다.
- `Connection` 플러그인은 인벤토리 호스트와 어떻게 통신하는를 정의하는 것입니다.
- `Lookup` 플러그인 외부 소스에서 데이터를 가져오는 플러그인 입니다.
- `Vars` 플러그인은 Ansible이 실행될 때 인벤토리, 플레이북 또는 명령행에서 받지 않은 추가 변수와 데이터를 밀어 넣을 때 사용됩니다.

Callback 플러그인

콜백 플러그인은 Ansible이 이벤트를 발생할 때 새로운 행동을 취하게 하는 플러그인입니다.

콜백 플러그인 예제

해당 플러그인 예제는 이미 ansible 안에 있는 것을 예제로 확인할 수 있습니다.

log_plays 콜백은 어떻게 플레이북 이벤트를 파일로 출력할 수 있게 하는 좋은 예제입니다. 그리고 mail 콜백은 플레이북이 종료되었을 때 이메일로 전달하는 콜백입니다.

또한 osx_say 콜백은 동료 맥 사용자들에게 플레이 이벤트를 전달하는 역할을 하는 콜백입니다.

콜백 플러그인 설정

콜백을 활성화 하려면 *ansible.cfg* 에서 설정한 콜백 디렉터리에 해당 콜백을 넣으면 됩니다.

플러그인은 파일의 알파벳 순서로 로드됩니다.

*whitelist*를 이용하여 수행될 콜백을 *ansible.cfg*에 지정할 수 있습니다.

```
callback_whitelist = timer, mail, myplugin
```

콜백 플러그인 개발

`CallbackBase` 라는 베이스 클래스를 상속받아 콜백 플러그인을 개발합니다.

```
from ansible.plugins.callback import CallbackBase
from ansible import constants as C

class CallbackModule(CallbackBase):
```

그런 다음 *덮어쓰기 (override)*를 이용하여 원하는 메소드의 기능을 수행합니다. 버전 2.0 이후부터는 *v2* 라고 시작하는 이름의 메소드를 덮어씁니다. CallbackBase 클래스에서 `_` 로 시작하지 않는 메소드를 참고 하십시오.

다음은 Ansible의 *timer* 플러그인이 구현된 것을 보여줍니다.

```
# Make coding more python3-ish
from __future__ import (absolute_import, division, print_function)
__metaclass__ = type

from datetime import datetime

from ansible.plugins.callback import CallbackBase


class CallbackModule(CallbackBase):
    """
    This callback module tells you how long your plays ran for.
    """
    CALLBACK_VERSION = 2.0
    CALLBACK_TYPE = 'aggregate'
    CALLBACK_NAME = 'timer'
    CALLBACK_NEEDS_WHITELIST = True

    def __init__(self):

        super(CallbackModule, self).__init__()

        self.start_time = datetime.now()

    def days_hours_minutes_seconds(self, runtime):
        minutes = (runtime.seconds // 60) % 60
        r_seconds = runtime.seconds - (minutes * 60)
        return runtime.days, runtime.seconds // 3600, minutes, r_seconds

    def playbook_on_stats(self, stats):
        self.v2_playbook_on_stats(stats)

    def v2_playbook_on_stats(self, stats):
        end_time = datetime.now()
        runtime = end_time - self.start_time
        self._display.display("Playbook run took %s days, %s hours, %s minutes, %s seconds" % (self
```

`CALLBACK_VERSION` 과 `CALLBACK_NAME` 클래스 변수가 꼭 필요합니다. 만약 콜백 플러그인이 표준 출력으로 출력할 필요가 있다면 `CALLBACK_TYPE = stdout` 라고 정의하고 ansible.cfg 설정파일에 디폴트 표준출력을 설정해야 합니다.

```
stdout_callback = mycallbackplugin
```

Connection 플러그인

디폴트로 `paramiko SSH` 와 표준 `ssh` , `local` 연결 뿐만 아니라 `chroot` 와 `jail` 과 같은 연결 설정이 있습니다. 이와 같은 연결 플러그인은 플레이북 또는 */usr/bin/ansible*에서 원격 머신과 통신하여 연결하는 것을 결정합니다. 그런데 SNMP, 메시지 버스, 전송구 등과 같은 다른 전송 방법을 사용하기를 원하십니까? 이미 있는 모듈 중에 하나를 복사하여 적절히 기능을 채운 다음 연결 플러그인 디렉터리에 넣어주기만 하면 됩니다. 연결을 위한 `smart` 라는 값은 해당 시스템의 상황에 따라 **paramiko** 또는 **openssh** 를 선택하거나 *ControlPersist* 를 제공하는 ***OpenSSH*** 라면 `ssh` 를 선택하도록 해 줍니다.

연결 플러그인 소스를 참고 하십시오.

Loopkup 플러그인

with_fileglob 와 **with_items** 와 같은 기능을 지원하는 언어는 lookup 플러그인으로 구현가능합니다.

lookup 플러그인 소스를 참고하십시오.

Vars 플러그인

`vars` 플러그인을 통하여 ***host_vars*** 와 ***group_vars*** 같은 것을 구성할 수 있습니다. 또한 인벤토리, 플레이북 또는 명령행에 지정하지 않은 추가 변수 데이터를 껴 넣을 수 있습니다.

vars_plugin 소스를 참조하십시오.

Filter 플러그인

만약 Jinja2 템플릿에서 제공하는 (*to_yaml* 과 *to_json*과 같은) 필터 이외의 필터를 원한다면 필터 플러그인을 작성할 수 있습니다. 대부분의 경우 플레이북에서 필요한 새로운 필터를 만들려고 하는 것이 있다면 `core.py` 에 넣을 의향이 있습니다.

더 자세한 것은 필터 플러그인 소스를 참조하십시오.

플러그인 배포

플러그인은 (ansible 배포 시 설치되는) 파이썬의 *site_packages* 와 **/usr/share/ansible/plugins** 아래에 다음과 같은 이름으로 각각 저장됩니다.

```
* action
* lookup
* callback
* connection
* filter
* strategy
* cache
* test
* shell
```

이 경로를 변경하기 위하여 ansible 설정 파일을 수정합니다.

추가하여 최상위 플레이북의 상대 경로의 하위 디렉터리에 위와 같은 이름의 하위 폴더에 저장됩니다.

동적 인벤토리 소스 개발

동적 인벤토리에 설명되어 있듯이 ansible은 클라우드 소스를 포함한 여러 동적 소스에서 관리대상목록인 인벤토리 정보를 구할 수 있습니다.

그러면 어떻게 작성할까요?

간단합니다! 적당한 인자를 제공받아 올바른 형식의 JSON으로 출력하는 스크립트 혹은 프로그램만 작성하면 됩니다. 특정 프로그램 언어의 제약은 없습니다.

스크립트 규칙

외부 노드의 스크립트가 `--list` 인자로만 호출되었을 때 스크립트는 결과는 *stdout*으로 일련의 JSON 형식의 해시/사전입니다. 각 그룹은 각 호스트/IP 해시/사전 구조의 목록이고 그룹 변수 또는 단순 호스트/IP 목록 들이 올 수 있습니다.

```
{
    "databases"   : {
        "hosts"   : [ "host1.example.com", "host2.example.com" ],
        "vars"    : {
            "a"   : true
        }
    },
    "webservers"  : [ "host2.example.com", "host3.example.com" ],
    "atlanta"     : {
        "hosts"   : [ "host1.example.com", "host4.example.com", "host5.example.com" ],
        "vars"    : {
            "b"   : false
        },
        "children": [ "marietta", "5points" ]
    },
    "marietta"    : [ "host6.example.com" ],
    "5points"     : [ "host7.example.com" ]
}
```

위 예제에서 webservers, marietta 및 5points 와 같이, 1.0 이전 버전에서 각 그룹은 단순 호스트이름/IP 주소의 목록만 있었습니다.

`--host <hostname>` 인자와 함께 호출되면 (위의 예에서 `<hostname>` 과 같은) 그 스크립트는 빈 JSON 또는 템플릿과 플레이북에서 사용가능한 변수가 보일 겁니다.
`json { "favcolor" : "red", "ntpserver" : "wolf.example.com", "monitoring" : "pack.example.com" }`

외부 인벤토리 스크립트 튜닝

버전 1.3 이후.

고정 인벤토리 스크립트 시스템은 모든 버전의 Ansible에서 동작을 잘 하지만 `--host` 옵션에서는 원격 호스트에 대해 비싼 API를 아용하는 경우 그렇지 않습니다. 버전 1.3 이후 만약 인벤토리 스크립트가 최상위에 **_meta** 라는 항목이 있으면 하나의 인벤토리 스크립트 호출에 모든 호스트 변수가 리턴될 수 있습니다. ***hostvars*** 의 내용을 포함하는 메타 항목일 때, 인벤토리 스크립트는 `--host` 로 불려지지 않습니다. 이 결과는 아주 많은 관리를 위한 호스트가 있을 때 성능 향상을 가져다 주고 클라이언트 영역에서 인벤토리 스크립트를 캐싱하기 쉽게 해 줍니다.

이런 최상위 JSON 사전은,

```
{

    # results of inventory script as above go here
    # ...

    "_meta" : {
       "hostvars" : {
          "moocow.example.com"     : { "asdf" : 1234 },
          "llama.example.com"      : { "asdf" : 5678 },
       }
    }

}
```

과 같습니다.

Python API

우선 본 API는 처음부터 외부에 이용할 목적으로 작성되지 않았음을 알아두시기 바랍니다.우선은 Ansible의 명령행 툴로 제공될 예정이었습니다.

다음의 문서는 API를 직접 사용할 경우 필요한 문서이지만 Ansible team에서 공식적으로 지원하는 것이 아님을 알아두십시오.

API 관점에서 바라볼 때 Ansible을 이용하는 여러 가지 방법이 존재합니다. 여러 노드를 제어하고, 다양한 파이썬 이벤트에 응답하기 위하여 그리고 또는 외부데이터에서 인벤토리 자료를 얻는 등을 하는데 API를 이용할 수 있습니다. 이 문서는 플레이북 API 및 실행에 관한 기본적인 차원의 문서입니다.

만약 파이썬이 아닌 다른 프로그래밍 언어로 비동기 이벤트 트리거링 또는 접근제어 및 로깅 등을 위하여 Ansible을 이용하려 한다면 Ansible Tower을 이용하기 바랍니다. 그것은 REST API 등이 잘 정의되어 있어 다른 언어로도 쉽게 이용할 수 있습니다.

Ansible은 자체 API로 작정되었기에 그 기능을 원하는데로 이용할 수 있습니다.

파이썬 API는 매우 강격하여 다른 모든 ansible CLI 툴이 구현되었는가 확인할 수 있습니다. 버전 2.0에서 대부분의 핵심 API가 재 작성되었습니다.

> 노트
>
> Ansible은 다른 프로세스를 생성(forking) 하도록 되어 있기에 자체로는 쓰레드 안전하지 않습니다.

Python API 2.0

비록 2.0이 좀 더 시작하기에 어려워 보일 수는 있어도 개별적이고 읽기 쉬운 클래스로 구성되어 있습니다.

```
#!/usr/bin/env python

import json
from collections import namedtuple
from ansible.parsing.dataloader import DataLoader
from ansible.vars import VariableManager
from ansible.inventory import Inventory
from ansible.playbook.play import Play
from ansible.executor.task_queue_manager import TaskQueueManager
```

```
from ansible.plugins.callback import CallbackBase

class ResultCallback(CallbackBase):
    """A sample callback plugin used for performing an action as results come in

    If you want to collect all results into a single object for processing at
    the end of the execution, look into utilizing the ``json`` callback plugin
    or writing your own custom callback plugin
    """
    def v2_runner_on_ok(self, result, **kwargs):
        """Print a json representation of the result

        This method could store the result in an instance attribute for retrieval later
        """
        host = result._host
        print json.dumps({host.name: result._result}, indent=4)

Options = namedtuple('Options', ['connection', 'module_path', 'forks', 'become', 'become_method', '
# initialize needed objects
variable_manager = VariableManager()
loader = DataLoader()
options = Options(connection='local', module_path='/path/to/mymodules', forks=100, become=None, bec
passwords = dict(vault_pass='secret')

# Instantiate our ResultCallback for handling results as they come in
results_callback = ResultCallback()

# create inventory and pass to var manager
inventory = Inventory(loader=loader, variable_manager=variable_manager, host_list='localhost')
variable_manager.set_inventory(inventory)

# create play with tasks
play_source =  dict(
        name = "Ansible Play",
        hosts = 'localhost',
        gather_facts = 'no',
        tasks = [
            dict(action=dict(module='shell', args='ls'), register='shell_out'),
            dict(action=dict(module='debug', args=dict(msg='{{shell_out.stdout}}')))
         ]
    )
play = Play().load(play_source, variable_manager=variable_manager, loader=loader)

# actually run it
tqm = None
try:
    tqm = TaskQueueManager(
              inventory=inventory,
              variable_manager=variable_manager,
              loader=loader,
              options=options,
              passwords=passwords,
              stdout_callback=results_callback,  # Use our custom callback instead of the ``default
          )
    result = tqm.run(play)
finally:
```

```
    if tqm is not None:
        tqm.cleanup()
```

Python API 2.0 이전

이전 버전은 아주 간단합니다.

```
import ansible.runner

runner = ansible.runner.Runner(
   module_name='ping',
   module_args='',
   pattern='web*',
   forks=10
)
datastructure = runner.run()
```

run 메소드는 호스트 마다 결과를 리턴하는데 컨택(contacted) 되는지 안되는지에 따라 그룹지어집니다. 리턴 형식은 모듈에 따라 다릅니다. 모듈에 관하여에 설명되어 있습니다.

```
{
    "dark" : {
       "web1.example.com" : "failure message"
    },
    "contacted" : {
       "web2.example.com" : 1
    }
}
```

모듈 결과는 JSON 형식이면 어떤 것이든 올 수 있기 때문에 이를 응용한 응용프로그램이나 스크립트의 프레임워크로써 동작할 수 있습니다.

자세한 API 예제

다음 스크립트는 모든 호스트에 대한 *uptime*을 출력합니다.

```
#!/usr/bin/python

import ansible.runner
import sys

# construct the ansible runner and execute on all hosts
results = ansible.runner.Runner(
    pattern='*', forks=10,
    module_name='command', module_args='/usr/bin/uptime',
).run()

if results is None:
   print "No hosts found"
   sys.exit(1)

print "UP ***********"
for (hostname, result) in results['contacted'].items():
    if not 'failed' in result:
        print "%s >>> %s" % (hostname, result['stdout'])

print "FAILED *******"
for (hostname, result) in results['contacted'].items():
    if 'failed' in result:
        print "%s >>> %s" % (hostname, result['msg'])

print "DOWN *********"
for (hostname, result) in results['dark'].items():
    print "%s >>> %s" % (hostname, result)
```

아마 더 고급 프로그래머는 `ansible` 명령행 툴(`lib/ansible/cli/`)를 구현하는데 API를 사용하는 것 처럼 ansible 자체의 소스를 읽고 파악할 수 있을 것입니다.

Ansible 핵심 엔진 개발

비록 Ansible 핵심 엔진의 많은 부분이 플레이북 명령이나 설정을 통해 바뀔 수 있는 플러그인으로 구성되어 있지만 여전히 모듈화 되어 있지 않은 부분도 있습니다.

모듈 원본 참조

Helping Testing PRs 참조

Release 참조