ANSIBLE

앤서블 101

목차

앤서블 101

목차

목차

소개

과정

담당자 및 강사

소개

01

앤서블에 관심이 있는 기초 과정

02

앤서블 도구에 대해서 전반적으로 학습을 원하는 엔지니어 및 서비스 개발자

강의일정

강의 일정은 아래와 같이 진행이 될 예정 입니다.

DAY 1:

- 랩 설명 및 활용
- 앤서블 소개 및 기본 문법
- 앤서블 명령어 및 인벤토리
- 모듈 활용 및 기능 구현

DAY 2:

- 모듈 활용 및 기능 구현
- 태스크 구성
- 앤서블 변수

DUSTBOX

강의일정

DAY 3:

- 조건문 및 레지스터
- 블록 및 핸들러 구성
- 시스템 관리

DAY 4/5

- 예외처리
- 마지막 슬라이드(roles)
- 플레이북 1
- 플레이북 2

강사

강사

강사

이름: 최국현

메일: tang@dustbox.kr, 회신은 다른 메일 주소로 드리고 있습니다. :)

사이트: tang.dustbox.kr

언제든지 질문 및 요청 환영 입니다.

LAB

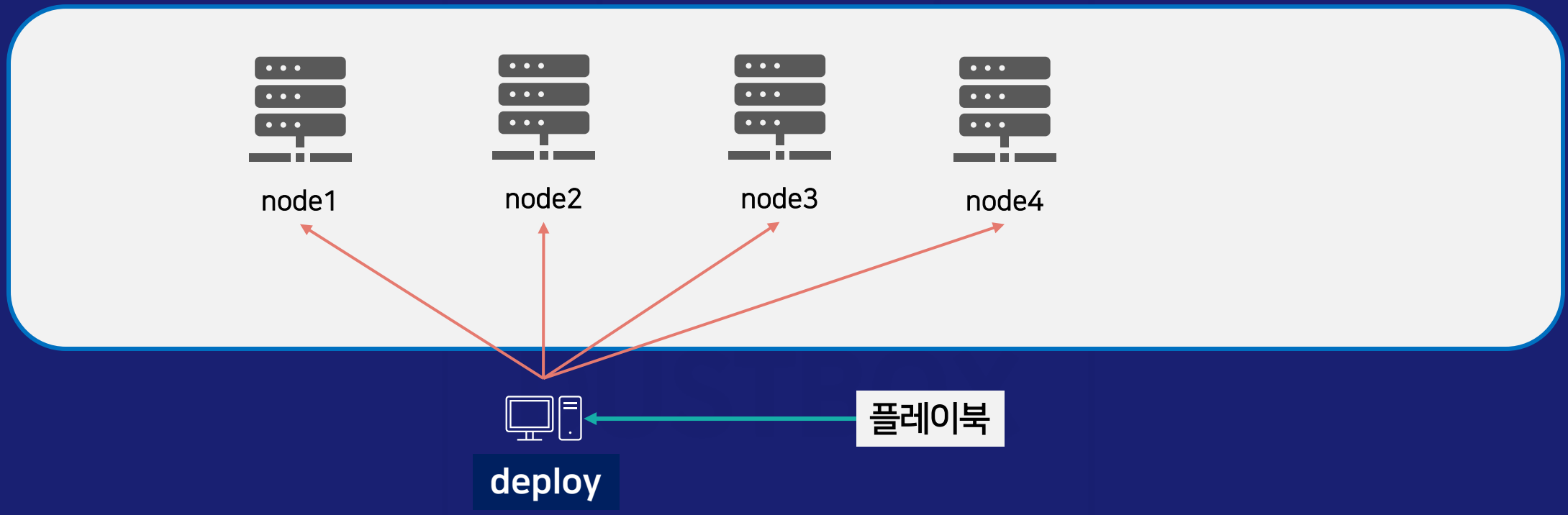
랩 및 교육대상

앤서블

이번 앤서블 교육은 다음 대상으로 제작 및 구성.

1. 앤서블 처음 사용하는 사용자
2. 실무까지는 아니어도 어떠한 방식으로 동작하는지 궁금하신 사용자
3. 간단하게 YAML 형태로 동작 학습
4. 전체적인 용어 및 기능 학습

랩 다이어그램



오픈스택 랩 정보

- **웹 주소**: https://vlab.dustbox.kr
- **사용자 계정**: ans1~20
- **비밀번호**: ansible
- **도메인**: ans-training

위 정보는 강의 진행 시 접근이 가능하며, 이후에는 접근이 불가능 합니다.

소개

앤서블 기능

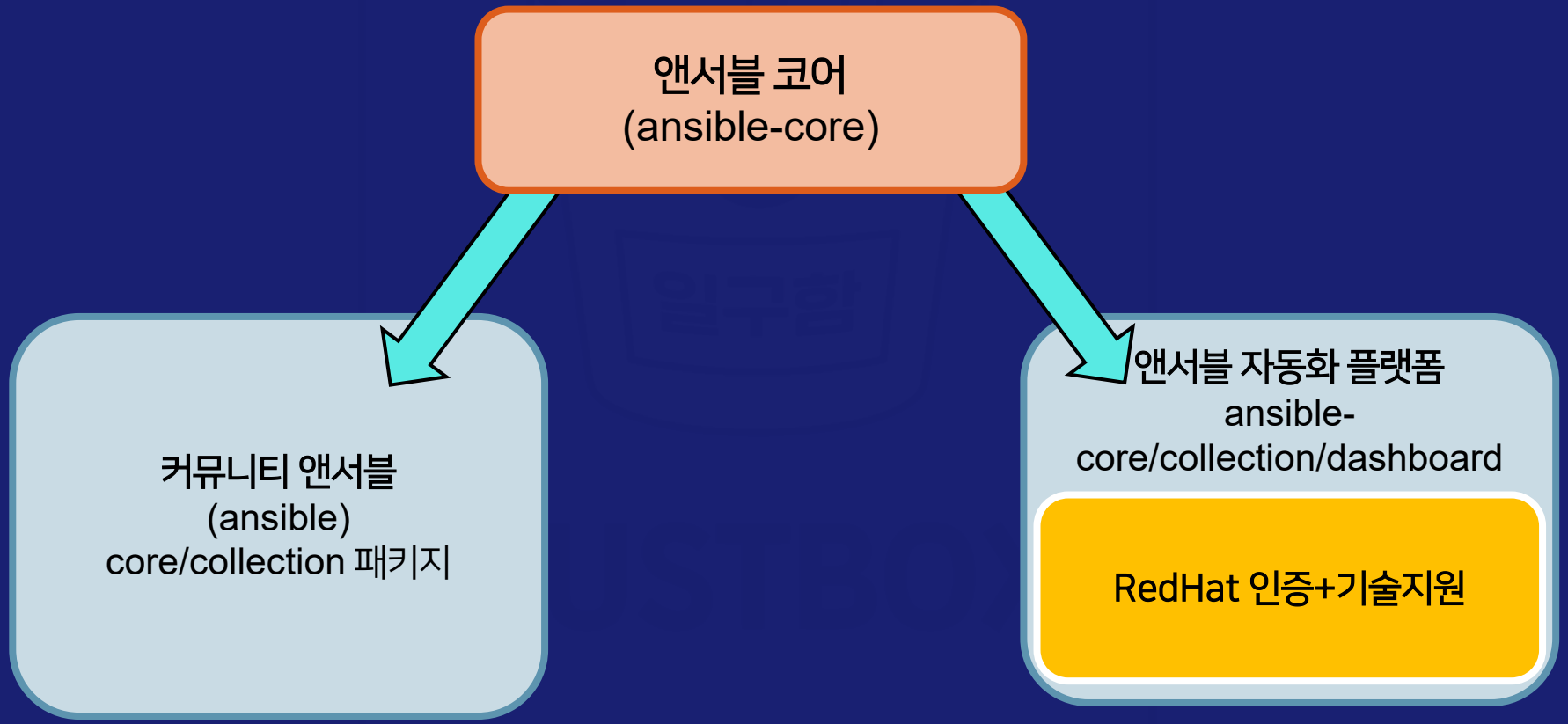
앤서블

Ansible은 원래 Ansible, Inc.에서 개발되었으며, 2015년에 Red Hat에 인수. 현재 Ansible은 두 가지 릴리즈로 나뉘어 유지.

1. **Ansible Core**: CLI 명령어와 기본 모듈 등 핵심 기능만 포함된 최소 단위 배포판 (ansible-core 패키지).
2. **Ansible (Community Distribution)**: Core에 더해 다양한 공식/비공식 컬렉션을 포함한 전체 배포판 (ansible 패키지).

참고로, 과거에는 Ansible을 "Ansible Engine"이라고 부르기도 했으나, 현재는 해당 명칭은 더 이상 공식적으로 사용되지 않는다. 기업용으로는 Red Hat의 Ansible Automation Platform이 별도 제공.

앤서블


앤서블 코어
(ansible-core)
커뮤니티 앤서블
(ansible)
core/collection 패키지
앤서블 자동화 플랫폼
ansible-
core/collection/dashboard
RedHat 인증+기술지원

앤서블

코어는 앤서블 핵심 모듈로 구성이 되어 있으며, 그 이외 확장으로 `collection, roles`로 통해서 확장이 가능하다. 현재 앤서블은 **앤서블 코어**로 통합 및 배포가 되고 있다.

각 확장은 제 3자 제공 혹은 커뮤니티로 나누어져 있고, 주요 코어 모듈 혹은 확장 모듈 경우 Redhat기여 혹은 제공을 하고 있음. 코드상 차이는 크게 없음.

1. 기본적으로 많이 사용하는 모듈(제공) `ANSIBLE CORE`에서는 `POSIX/BUILT-IN`이다.
2. 그 이외 나머지 기능들은 `collection(community, vender)`로 확장.

앤서블

비교는 다음과 같다.

항목	Ansible Core	Ansible	AAP
개발 주체	Red Hat (오픈소스)	커뮤니티 + Red Hat	Red Hat (상용 제품)
배포 패키지 이름	ansible-core	ansible	AAP (유료 구독 기반)
포함 요소	CLI 도구, 기본 모듈	Core + Collections + Plugin 등	Core + 인증된 컬렉션 + Controller 등
컬렉션 관리	수동(필요시 추가)	커뮤니티 컬렉션 포함	인증된 컬렉션 (Automation Hub)

앤서블

비교는 다음과 같다.

항목	Ansible Core	Ansible	AAP
라이선스	GPLv3	GPLv3	Red Hat 구독 라이선스
지원 방식	커뮤니티	커뮤니티	상용 기술지원 포함
GUI 및 자동화 도구	없음	없음	Ansible Controller (이전 Tower) 등 포함
목적/용도	최소 기능만 필요한 사용자용	일반 사용자를 위한 기능 완비 배포판	기업용 자동화 및 대규모 관리

앤서블 프로그램

또한 앤서블 코어는 두 가지 릴리즈 방식이 있다.

ansible-core

- 앤서블 코어는 앤서블 인터프리터 + 코어 모듈
- 바이너리 프로그램

ansible-project

- 앤서블 코어 + 추가적인 컬렉션 구성

앤서블 프로그램

ansible-navigator

- 앤서블 통합 도구. 해당 명령어로 플레이북 관리 및 실행이 가능하다. 다만, 현재 레드햇 배포판 제외하고 다른 모든 리눅스 배포판에서는 PIP를 통해서 설치해야 한다.
- 네비게이터를 사용하기 위해서는 컨테이너 런타임이 설치가 필요하다.
- 포드만 혹은 도커 기반으로 실행

ansible-builder

ansible-builder는 **Ansible 실행 환경(Execution Environment, EE)** 을 **자동으로 빌드**해주는 도구. 쉽게 말해, **Ansible이 돌아가는 맞춤형 컨테이너 이미지**를 만드는 도구.

앤서블 프레임 워크 도구

앤서블 AAP(Ansible Automate Platform)

Ansible Automation Platform (AAP)는 Ansible을 기반으로 **모든 인프라, 애플리케이션, 네트워크 및 클라우드 환경의 자동화**를 가능하게 하는 통합 플랫폼 도구.

AAP의 핵심 기술은 다음과 같다.

- **YAML 기반 선언형 자동화** (Playbook 중심)
- Ansible Navigator를 통한 CLI 기반 실행 및 디버깅
- Automation Controller(구 Ansible Tower)를 통한 시각화, RBAC, 워크플로우 관리
- **Ansible Content Collections 및 Galaxy 역할(Role) 기반 재사용성과 표준화 확장성**
- **IBM왓슨을 통한 플레이북 작성 및 구성**

앤서블 프레임 워크 도구

앤서블 타워(AAP 커뮤니티 버전)

앤서블 타워는 두 가지 버전으로 구성이 되어있다. 첫 번째는 **앤서블 AWX 업 스트리밍 버전**이 있으며 레드햇에서 제공하는 서브스크립션 버전.

1. 제일 큰 차이점은 버전 픽스가 되지 않는 롤링 업데이트 버전이다.
2. 기술지원이 되지 않음.

타워의 큰 특징은 웹 대시보드에서 API기반으로 **앤서블 호출, 구성 및 관리가 가능**하며 **사용자 단위로 작업 할당 및 추적**이 가능하다.

앤서블 프레임 워크 도구

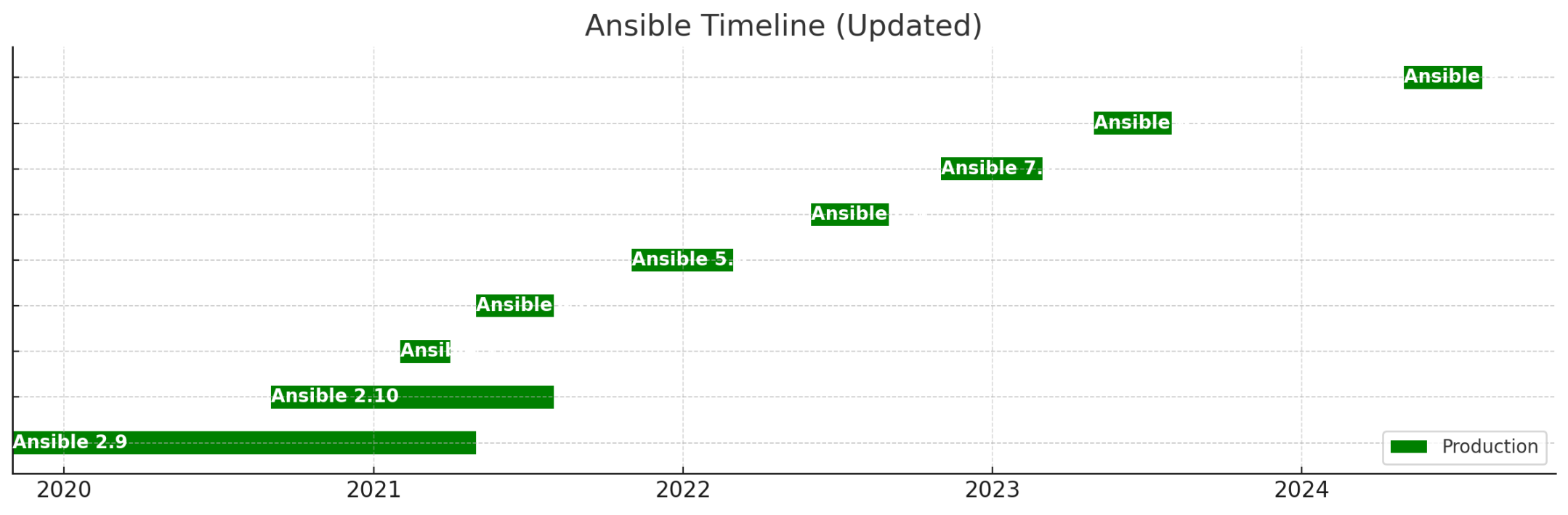
앤서블 Galaxy

Ansible Galaxy는 Ansible 자동화 개발자가 만든 **역할(Role)** 및 **컬렉션(Collection)**을 공유하고 배포할 수 있는 **공식 콘텐츠 허브.** 사용자는 Galaxy를 통해 다음을 할 수 있다.

- HTTP/HTTPS 또는 Git 기반으로 콘텐츠 검색 및 다운로드
- 내부망/외부망에 관계없이 **사내 Galaxy 서버(Private Automation Hub)** 구성 가능
- 모듈, 플러그인, 역할, 플레이북 등을 포함한 **전체 자동화 콘텐츠 패키지화(Collection)** 지원

이를 통해 조직은 **자체 자동화 표준을 정립**하고, **인증된 콘텐츠를 재사용 및 버전 관리** 가능.

앤서블 EOL


Ansible Timeline (Updated)
Ansible
Ansible
Ansible 7.
Ansible
Ansible 5.
Ansible
Ansi
Ansible 2.10
Ansible 2.9
Production
2020
2021
2022
2023
2024

앤서블 타워

항목	AWX (오픈소스)	Ansible Tower (Red Hat 상용)
라이선스	Apache License 2.0	Red Hat 서브스크립션 기반
업데이트 방식	롤링 릴리즈 (버전 고정 없음)	버전 고정, 안정성 확보
기술 지원	공식 기술 지원 없음 (커뮤니티 기반)	Red Hat에서 기술 지원 제공
보안 및 인증	기본 제공	보안 인증(QA), 보안 패치, CVE 대응 포함
상용 연계 기능	없음	인증된 Collections, AAP 연계 기능 포함
주요 기능	웹 UI, REST API, RBAC, 스케줄링	동일 + 보안 검증, 조직 내 정책 통합 가능

앤서블 에디터 도구

앤서블 사용하기 전에 준비를 해야 될 부분은 다음과 같다. YAML 작성 시 사용할 에디터.

아무거나 좋다! 정말로! 나머지는 필요한 도구 및 재료이다.

- `vi, vim, NeoVim, nano`
- `YAML`
- `ansible.cfg`
- `ansible, ansible-playbook, ansible-navigator, ansible-builder`

앤서블 랩 환경

리눅스 콘솔에서 작성 시 사용하는 대표적인 에디터는 **vi/vim**에디터가 있다.

거의 대다수 리눅스는 기본적으로 vi는 설치가 되어 있으나, **vim**설치가 되어 있지 않는 경우가 있다. 이러한 이유로 **vim**설치 이외 몇 가지 기능을 추가적으로 구성 및 활성화 해야 한다.

사용하기 어려운 경우에는 **nano**를 사용하여도 된다.

오픈스택 랩 환경에서는 외부에서 접근이 되지 않기 때문에, 윈도우 환경의 에디터 사용은 불가능하다.

앤서블 준비

에디터

접근

SSH 키 및 비밀번호 접근

앤서블은 기본적으로 두 가지 접근 방식을 제공한다.

1. SSH 비공개/공개키 접근 방법
2. 사용자 아이디 및 비밀번호 접근

ssh키 사용이 어려운 경우, 비밀번호 접근으로 사용이 가능하다.

SSH 키 배포

SSH 키는 두 번째 네트워크를 통해서 공개키를 배포한다. 혹은, 아래 슬라이드와 같이 파일을 작성 후, 공개키 배포가 가능합니다.

```
deploy# ssh-keygen -t rsa -N '' -f ~/.ssh/id_rsa
deploy# ssh-copy-id root@192.168.90.X
```

SSH 키 배포

ssh 키 기반으로 사용하는 경우, 아래와 같이 YAML 작성 후, 배포가 가능하다.

```
deploy# vim sendsshkey.yaml
---
- hosts: all
  ansible_user: root
  ansible_password: centos
  tasks:
  - authorized_key:
      user: root
      key: "{{ lookup('file', '/home/' + lookup('env', 'USER') +
'{{ k8s_public_rsa_locate }}') }}"
      key: "{{ lookup('file', '/root/' + '.ssh/id_rsa.pub') }}"
…
```

에디터 환경

모든 YAML은 아래와 같은 조건으로 작성이 되어 있음.

- 들여쓰기 2칸
- 탭은 빈 공간(space)으로 전환
- 자동 들여쓰기 활성화

vim editor

vi/vim를 위해서 기능 강화.

```
deploy# touch .vimrc
deploy# dnf install vim vim-enhanced neovim yamllint -y
```

레드햇 계열 배포판을 사용하는 경우, 아래와 같이 설치.

```
deploy# dnf install vim-ansible
```

vim + ale

Windows

```
> curl.exe https://webi.ms/vim-ale | powershell
```

Linux

```
# curl -sS https://webi.sh/vim-ale | sh
```

MacOS

```
# curl -sS https://webi.sh/vim-ale | sh
```

.vimrc

`.vimrc` 파일을 다음과 같이 설정을 변경한다.

```
# vi ~/.vimrc
set ts=2
set sts=2
set sw=2
set expandtab
set number
syntax on
filetype indent plugin on
```

nano

나노 에디터를 사용하는 경우, 아래 명령어로 나노 에디터 설정 및 구성이 가능하다. 혹은, 수동으로 구성 시, 아래와 같이 `~/.nanorc` 파일을 생성한다.

```
# dnf install nano wget curl
# curl https://raw.githubusercontent.com/scopatz/nanorc/master/install.sh | sh
```

.nanorc

나노 파일은 다음과 같이 설정한다.

```
syntax "YAML" "\.ya?ml$"
header "^(---|\.\.\.)" "%YAML"
color magenta "^\s*[\$A-Za-z0-9_-]+\:"
color brightmagenta "^\s*@[\$A-Za-z0-9_-]+\:"
color white ":\s.+$"
icolor brightcyan " (y|yes|n|no|true|false|on|off)$"
color brightred " [[:digit:]]+(\.[[:digit:]]+)?"
color red "\[" "\]" ":\s+[>]" "^\s*- "
color green "(^| )!!(binary|bool|float|int|map|null|omap|seq|set|str) "
```

.nanorc

앞에 이어서 계속…

```
color brightwhite "#.*$"
color ,red ":\w.+$"
color ,red ":'.+$"
color ,red ":".+$"
color ,red "\s+$"
color ,red "['\"][^['\"]]*$"
color yellow "['\"].*['\"]"
color brightgreen ":( |$)"
set tabsize 2
set tabstpotospacs
```

SSH KEY

키 배포

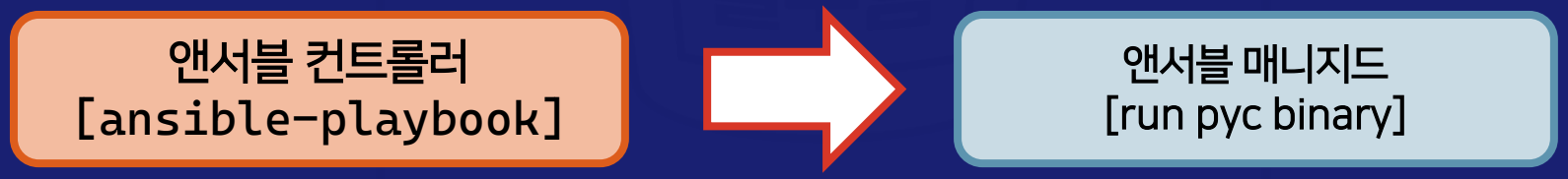
비밀번호 접근

앤서블 사용자 구성

키 배포

앤서블은 앞에서 잠깐 언급 하였지만 SSH를 사용해서 `managed node`에 접근. 배포를 하기 위해서는 다음과 같은 명령어를 통해서 생성 및 진행 해야 한다.

가급적이면, 앤서블 사용자는 root 계정이 아닌, 일반 계정에서 실행을 권장.



앤서블 사용자 구성

계정 생성 후, 비공개 키 생성 및 배포는 다음과 같이 한다.

```
# adduser ansible
# echo ansible | passwd --stdin ansible
# usermod -aG wheel
# vi /etc/sudoers.d/ansible
%wheel  ALL=(ALL)       ALL
%wheel  ALL=(ALL)       NOPASSWD: ALL

# ssh-keygen -t rsa -N '<PASSPHASE>' -f '<SSH_KEY_DIRECTORY>'
# sshpass -p <PASSWORD> ssh-copy-id <USER>@<HOST>
```

비공개키 생성

`ssh private`키 생성에 관련된 앤서블 모듈은 아직 없기 때문에 생성을 원하는 경우에는 `command, shell`와 같은 모듈을 통해서 생성해야 한다.

혹은, 커뮤니티 모듈인 `ansible-galaxy collection install community.crypto`를 사용해야 한다.

```
- name: Generate an OpenSSH rsa
  community.crypto.openssh_keypair:
    path: /tmp/id_ssh_rsa
    size: 2048
```

플레이북 기반 키 배포

혹은 `authorized_keys` 기능을 사용해서 `ssh`공개키 배포가 가능하다.

아래 예제 코드가 있다. 이 코드 내용은 앤서블 웹 사이트에서 확인이 가능하기 때문에, 자세한 내용은 아래 링크에서 확인이 가능하다. (앞에서 잠깐 설명하였던 코드와 동일하다)

https://docs.ansible.com/ansible/latest/collections/ansible/posix/authorized_key_module.html

플레이북 기반 키 배포

authroized_key 예제(sendsshkey.yaml)

```
- hosts: all
  tasks:
  - authorized_key:
      user: testuser
      key: "{{ lookup('file', '/home/' + lookup('env', 'USER') +
'/.ssh/id_rsa.pub') }}"
```

SSH 비밀번호 접근

만약, 특정한 이유로 보안키로 서버나 혹은 네트워크 장비에 접근이 안되는 경우, 다음과 같은 방법으로 원격 서버 혹은 장비에 접근이 가능하다. 아래 코드는 플레이 북에서 작성한다.

```
become: false
become_password: rocky
```

아래 코드는 인벤토리에 설정하는 내용이다.

```
localhost ansible_user=ansible ansible_password=ansible
ansible_become_password=ansible
```

앤서블 명령어

ansible

ansible-playbook

ansible-navigator

ansible-builder

ansible ad-hoc

앤서블에서 모듈 기반으로 사용하는 명령어. 일반적으로 **ad-hoc**처리시 많이 사용한다.

보통 쉘 스크립트를 앤서블 모듈 기반으로 처리하고 싶은 경우 많이 사용한다. 하지만, **systemd**로 들어 오면서 점점 쉘 스크립트를 사용하는 상황이 작아지면서 **ansible명령어**는 사용하는 영역은 작아지고 있다.

```
# ansible -m shell -a 'cmd="ls -l"'
```

ansible-doc

앤서블 문서를 확인하는 방법은 두 가지가 있다.

1. **앤서블 온라인 문서**
2. **앤서블 문서 명령어**

앤서블은 모듈에 대한 맨-페이지를 제공하지 않기 때문에, 모듈 사용 방법을 확인하기 위해서는 **`ansible-doc`** 명령어를 통해서 확인해야 한다.

```
# ansible-doc -l
# ansibel-doc shell
```

ansible-playbook/navigator

앤서블 플레이북을 하나 이상 실행 시, 해당 명령어를 사용한다. 앤서블은 보통 하나 이상의 YAML기반으로 구성이 되어 있다.

이를 실행해주는 프로그램은 **`ansible-playbook`** 명령어. 이를 보통 **실행자(launcher)**라고 부르며, **`YAML`**및 **`Jinja2`**로 구성이 되어 있는 구성 파일을 메모리에 불러와서 컴파일 후 작업을 수행 및 실행한다.

`ansible-navigator`는 앤서블 런타임을 컨테이너 형태로 패키징이 되어 있는, 일종의 런타임이다.

`ansible-navigator`는 **`Ansible Automation Platform(AAP)`**에서 제공되는 도구로, **`Ansible`** 런타임을 컨테이너 기반으로 실행할 수 있게 해준다.

이 런타임 컨테이너에는 필요한 라이브러리와 실행 환경이 포함되어 있으며, 일반적으로 **루트리스 컨테이너 기술**(Podman 등)을 활용해 실행된다.

ansible-playbook/navigator

2024년 기준, **ansible-navigator**는 **RHEL** 및 **AAP** 구독 환경에서 지원되며, **Fedora**나 **CentOS**에서는 공식적으로 RPM 패키지가 제공되지 않는다. (2024년 기준)

```
# pip3 install 'ansible-navigator[ansible-core]'
# ansible-playbook -i hosts main.yaml
# ansible-navigator run main.yaml
```

ansible-tower 설명

앤서블 타워는 웹 기반으로 **플레이북/인벤토리/사용자를** 관리 할 수 있는 도구이다. 대규모로 사용하는 개인 및 업체를 위한 도구이다. 앤서블 타워는 현재 두 가지 버전으로 구성되어 있다.

1. 앤서블 오토메이션 플랫폼(이전이름 ansible tower(레드햇 판매 버전))
2. 앤서블 AWX(커뮤니티 버전)

두 개의 제일 큰 차이점은, 버전 갱신 방식이 **롤링 업데이트(rolling update), 릴리즈 업데이트(version release update)**차이가 있다. 이 과정에서는 앤서블 **AWX**에서 다루지는 않는다.

ansible-builder 설명

ansible-builder는 **ansible-navigator**에서 사용할 이미지를 구성 및 빌드한다. 기본적으로 **ansible-navigator**는 모든 모듈을 가지고 있기 때문에, 동작 시, 느린 부분이 있다. 이러한 문제를 해결하기 위해서 **ansible-builder**로 컨테이너 이미지를 빌드한다.

기능	설명
의존성 자동 수집	**execution-environment.yml** 파일을 읽어서 Python 패키지, OS 패키지, Ansible Collection을 자동으로 포함
컨테이너 빌드 자동화	Podman 또는 Docker를 이용해 EE(Execution Environment) 이미지 생성
AWX/Automation Controller	생성된 이미지를 AWX나 Ansible Automation Platform에서 실행 환경으로 바로 사용 가능
커스텀 이미지 기반 확장	`base_image` 설정을 통해 기존 EE 이미지를 확장 가능 (예: Red Hat 공식 EE 이미지 기반)
버전 고정 관리	특정 버전의 ansible-core, collection 등을 고정하여 일관된 환경 보장

ansible-builder 동작 순서

execution-environment.yml
- 기본 설정(version, base/builder 이미지)
- dependencies 섹션 → 아래 3개 파일 참조
- 추가 빌드 단계(additional_build_steps)

requirements.txt
Python 패키지
예: openstacksdk

bindep.txt
시스템 패키지
예: gcc, libffi

requirements.yml
Ansible 컬렉션
예: openstack.cloud

ansible-builder build
Podman/Docker 기반으로
이미지 빌드

Ansible Navigator / AWX / Controller 등에서 실행환경으로 사용.
ansible-navigator run ... -eei ee-openstack

DUSTBOX

ansible-builder structure

디렉터리 구조는 다음과 같이 생성 및 구성한다.

```
~/ee-build/
├── execution-environment.yaml
├── requirements.txt
├── bindep.txt
└── requirements.yaml
# ansible-builder build -t ee-openstack:latest --container-runtime podman
# ansible-builder build -f /path/to/execution-environment.yml -t ee-
openstack:latest
```

ansible-builder 구성

예를 들어서 오픈스택 운영을 위한 이미지를 만들기 위해서 다음과 같이 구성이 가능하다.

```
# vi execution-environment.yaml
version: 3
images:
  base_image:
    name: quay.io/ansible/ansible-runner:stable-2.15
  builder_image:
    name: quay.io/ansible/ansible-builder:latest

dependencies:
  python: requirements.txt
  system: bindep.txt
  galaxy: requirements.yml
```

ansible-builder 구성

위의 내용 계속 이어서…

```
additional_build_steps:
  prepend:
    - RUN echo "Starting custom build for OpenStack EE"
    - RUN dnf install -y git && dnf clean all
  append:
    - RUN echo "OpenStack Ansible Execution Environment build complete!"
```

ansible-builder 구성

requirements.txt 파일은 다음과 같이 작성한다.

```
# vi requirements.txt
openstacksdk≥1.0.0
python-openstackclient≥6.0.0
ansible-core≥2.15
netaddr
cryptography
requests
```

ansible-builder 구성

bindep.txt는 OS 패키지를 구성한다.

```
# vi bindep.txt
openstacksdk≥1.0.0
python-openstackclient≥6.0.0
ansible-core≥2.15
netaddr
cryptography
requests
```

ansible-builder 구성

requirements.yaml 앤서블 컬렉션 구성.

```
# vi requirements.yaml
---
collections:
  - name: ansible.posix
  - name: community.general
  - name: openstack.cloud
  - name: ansible.netcommon
```

ansible-builder 구성

위의 리소스가 구성이 완료가 되면, 최종적으로 아래와 같이 이미지 빌드가 가능하다.

```
# ansible-builder build -t ee-openstack:latest --container-runtime podman
# podman images | grep ee-openstack
# ansible-navigator run site.yml -m stdout -eei ee-openstack:latest
# ansible-playbook -i inventory site.yml -e
ansible_python_interpreter=/usr/bin/python3
```

앤서블 문법

문법 및 간단한 인벤토리

표준 데이터 파일

현재 오픈소스는 데이터 형식에 대해서 표준화를 진행하고 있다. 현재, 오픈소스에서 다음과 같이 자원 표준화를 진행하고 있다.

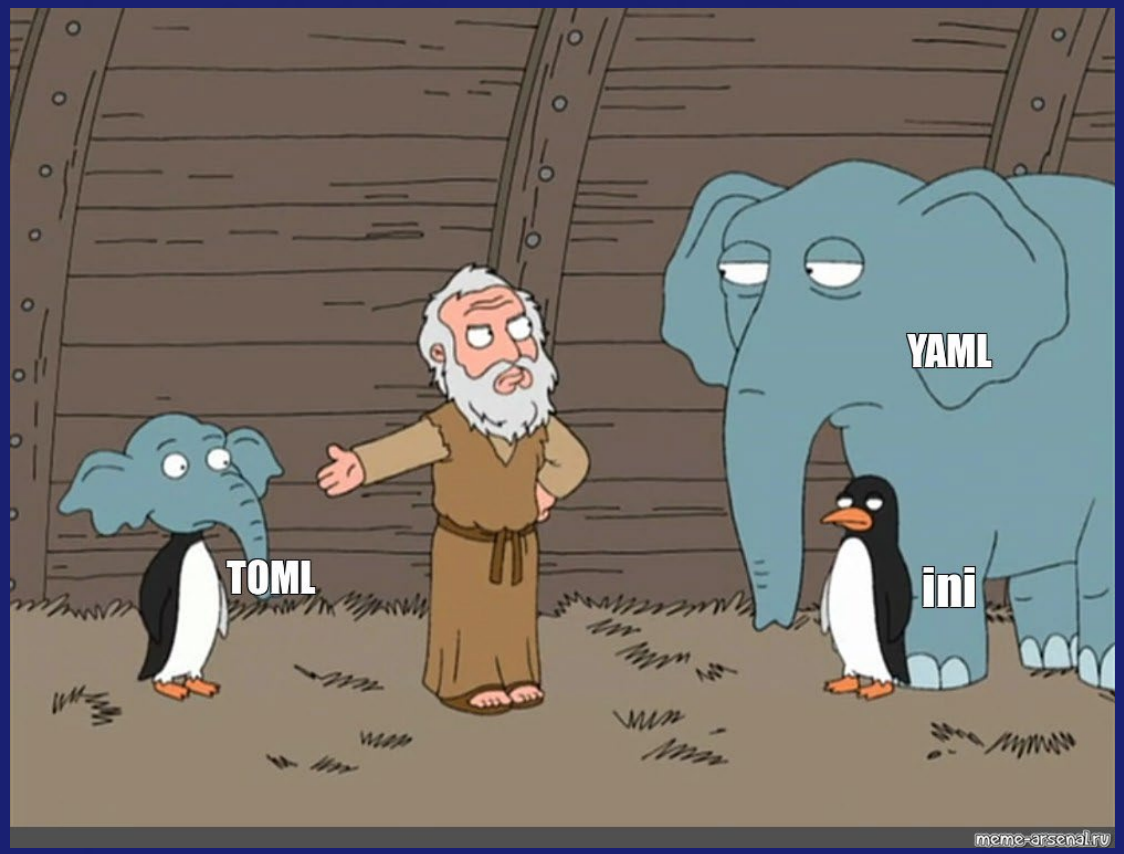
1. TOML
2. YAML
3. JSON

앤서블은 TOML/YAML/JSON를 전부 사용하고 있다. 자동화를 위한 **작업 데이터 셋**은 YAML으로 작성이 되며, 앤서블 내부적으로 데이터 핸들링은 JSON으로 데이터를 다룬다.

인벤토리와 같은 설정형식은 기존 INI에서 TOML으로 변환하고 있다. TOML를 **앤서블 코어 2.8** 이후부터 지원한

```
https://docs.ansible.com/ansible/latest/collections/ansible/builtin/toml_inventory.html
```

YAML/JSON/TOML



YAML

YAML 조건

앤서블에서 사용하는 문법을 작성하기 위해서는 다음과 같은 조건을 만족해야 한다.

1. 최소 한 칸 이상의 띄어쓰기(권장은 2칸)
2. 탭 사용시 반드시 빈 공간으로 전환
3. 블록 구별은 -(대시)로 반드시 처리

YAML?

```
- name: simple playbook
  hosts: all
  become: false
  tasks:
    - name:
      module
      args1:
      args2:
```

YAML??

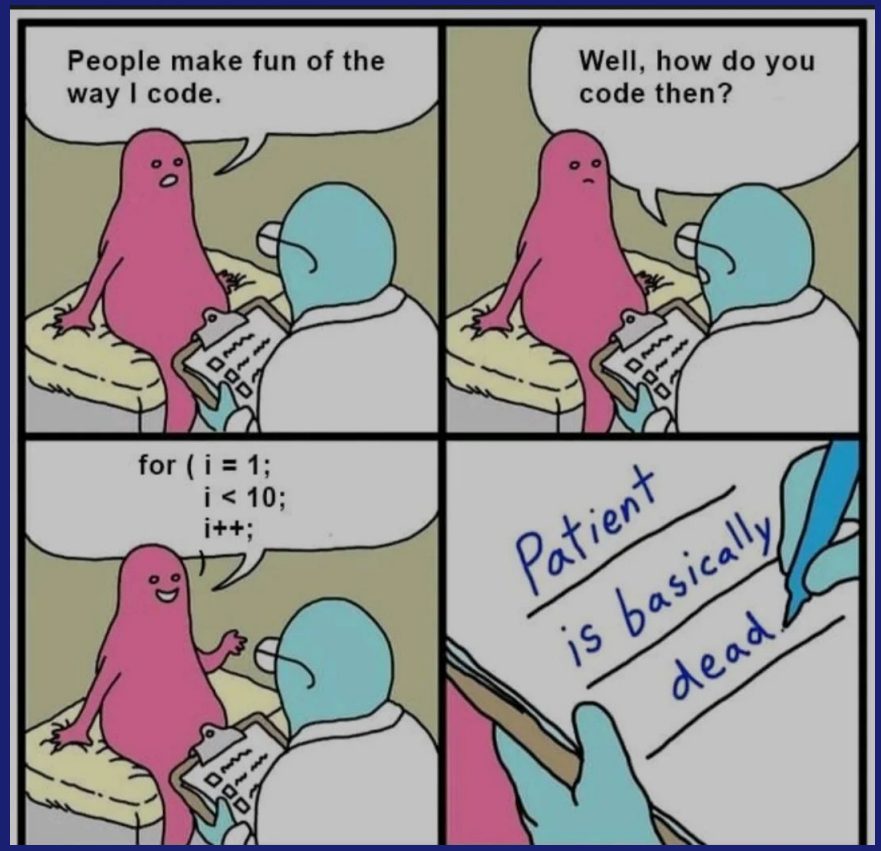
YAML를 형식(format)이 정해져 있다. 다만, 문제가 **들여쓰기(ident)**가 조금은 명확하지 않는 부분이 있다. 예를 들어서 리스트 혹은 딕셔너리는 선언 기호인 - 경우에는 다음과 같은 문제가(?) 있다.

```
tasks:
- test
    parameter
```

```
tasks:
  - test
      parameter
```

위의 두 개는 들여쓰기가 다르지만, 결과적으로 같은 **파싱(parsing)**결과가 나온다. 결국 사람마다 다른 형식으로 작성하기 때문에, 적절하게 작업자끼리 형식에 대한 합의가 필요하다.

STYLE DEAL WITH DEATH



그러면..YAML?

기본 앤서블 YAML 문법

앤서블 블록 구별을 보통 -로 구별한다. 시작 블록은 보통 다음과 같은 형식으로 많이 사용한다.

```
- name: <keyword>
  <module>
```

기본 앤서블 YAML 문법

그래서 YAML 상단에는 다음과 같은 형태로 키워드 명령어를 사용한다.

```
- name:
  hosts: all
  become: true
```

작업이름

대상서버 이름

앤서블 내장 키워드

전역 키워드 선언

전역 키워드는 **hosts:, tasks:** 사이에 작성한다. 일반적으로 많이 사용하는 명령어는 아래와 같다. 이 이외 키워드 명령어는 **보통 ansible.cfg**이나 혹은 매뉴얼을 통해서 확인이 가능하다. 일반적으로 많이 사용하는 전역 키워드는 아래와 같다.

```
- hosts:
  vars:
    - var1
    - var2
  become:
  remote_user:
  task
```

보통 해당 영역에 변수를 선언한다.

tasks:

모든 작업이 시작되는 구간. tasks 구간에는 여러 **모듈(module)**이 모여서 하나의 작업 **워크플로우(workflow)**를 구성한다.

여러 개의 워크 플로우가 구성이 되면 이걸 플레이북 혹은 **플레이북 작업(playbook tasking)**이라고 부른다.

```
# 공식적으로 틀린 문법; 하지만, 이 방식을 많이 사용한다.
   tasks:
   - name:
# 일반적인 문법
   tasks:
   - name:
```

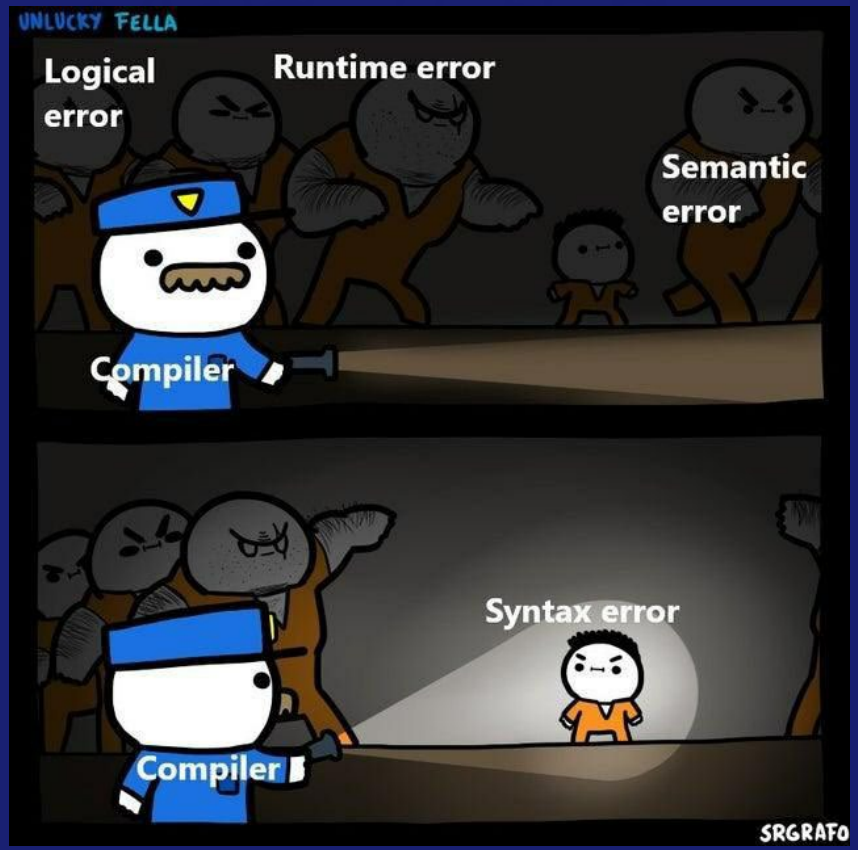
- name

YAML에서 작업 및 혹은 플레이북 작성 시, 각각 모듈 혹은 플레이북에 `name:` 키워드를 사용하여 작성 및 구성을 권장한다.

```
- name: this is the first module task
  ping:
```

위와 같은 방법으로 명시한 모듈이나 플레이북에 어떠한 목적으로 동작하는지 간단하게 적는다. 이전에는 CJK를 지원하지 않았지만, 지금은 문제없이 지원하고 있다.

YAML


UNLUCKY FELLA
Logical error
Runtime error
Semantic error
Compiler
Syntax error
Compiler
SRGRAFO

연습문제

아래와 같은 조건으로 인벤토리 및 간단한 `INI, YAML`형태로 `inventory-test.ini, test-payload.yaml`를 작성한다.

1. 인벤토리 파일에 두 개의 그룹을 생성한다.
   - webserver, database 그룹을 생성한다.
   - webserver 그룹에는 nodea, nodeb를 추가, database 그룹에는 nodec, noded를 추가한다.
2. `test-payload.yaml` 파일에는 다음과 같이 내용을 추가한다.
   - `the test payload`라는 이름으로 작업(task)를 생성한다.
   - 위의 작업 이름에 모듈은 아래처럼 작성한다.

```
debug:
  msg: "Hello the first payload"
```

연습문제

정답 :)

```
# vi inventory-test.ini
[webserver]
nodea ansible_host=192.168.100.10
nodeb ansible_host=192.168.100.11

[database]
nodec ansible_host=192.168.100.12
noded ansible_host=192.168.100.13
```

연습문제

정답 :)

```
# vi test-payload.yaml
---
- name: Simple test payload play
  hosts: webserver,database
  gather_facts: false

  tasks:
    - name: the test payload
      ansible.builtin.debug:
        msg: "hello from {{ inventory_hostname }}"
```

기본 기능 및 모듈

기본 기능

모듈 사용

ad-hoc

단일 실행

ad-hoc

앤서블은 YAML형태 말고 애드 훅(ad-hoc)**방식**이 있다. 이 방식은 마치 쉘 스크립트 실행하는 방식과 비슷하게 **모듈+인자 형태**로 구성이 되어 있다.

애드훅은 쉘 스크립트에서 같이 사용하거나 혹은 몇몇 쉘 스크립트 기능을 표준화 모듈 기반으로 사용하기 위해서 사용한다.

아래는 간단한 **ad-hoc**사용 방식이다.

```
# ansible -i <INVENTORY|HOSTGROUP> -m <module> -a "arg1=<value>
arg2=<value> <OPTIONS>
```

ad-hoc

간단하게 테스트나 혹은 사용시에 보통 다음과 같이 명령줄에서 실행한다.

```
# ansible -i hosts -m copy -a "src=/etc/hosts dest=/tmp" node1.example.com
# ansible -m ansible.builtin.debug -a "msg=hello" localhost
# ansible -m setup localhost
# ansible web -m command -a "uptime"
# ansible all -m ping
# ansible all -m service -a "name=sshd state=restarted" -b
# ansible web -m yum -a "name=httpd state=present" -b
# ansible db -m apt -a "name=mariadb-server state=present update_cache=yes"
-b
```

adhoc -> YAML

애드훅은 YAML으로 다음과 같이 대입된다.

```
# ansible all -m ping

# vi ping.yaml
- hosts: all
  gather_facts: no
  tasks:
    - name: 연결 확인
      ansible.builtin.ping:
```

adhoc -> YAML

애드훅은 YAML으로 다음과 같이 대입된다.

```
# ansible web -m package -a "name=httpd state=present" -b

# vi install_pkg.yaml
- hosts: web
  become: yes
  tasks:
    - name: httpd 설치
      ansible.builtin.package:
        name: httpd
        state: present
```

adhoc -> YAML

애드훅은 YAML으로 다음과 같이 대입된다.

```
# ansible all -m service -a "name=nginx state=restarted" -b

# vi restart_nginx.yaml
- hosts: all
  become: yes
  tasks:
    - name: nginx 재시작
      ansible.builtin.service:
        name: nginx
        state: restarted
```

ad-hoc shell script

```
$ vi adhoc_replace_shell_functions.sh
#!/bin/bash
if $(ansible localhost, -m ping)
then
  ansible localhost, -m copy -a "dest=/var/www/html/index.html,
content='This is World'"
else
  ansible localhost, -m copy -a "dest=/root/README.md, content='This file
is wrong'"
fi
```

ADHOC 연습문제

파일명은 `adhoc-exam.sh`으로 정한다. `node4.example.com` 대상으로 아래 작업을 수행한다.

1. `Hello World`라는 `default.html` 파일을 `/var/www/html/`에 생성.
   - `copy:` 모듈에서 `content:`를 사용해서 위의 내용 생성.
2. 웹 서버 패키지가 설치가 안되어 있으면 `yum:` 모듈을 사용해서 설치.
3. 방화벽에 `http, https` 서비스가 등록이 안되어 있으면 `http, https` 서비스에 등록. 모듈은 `firewalld`를 사용.
4. 문제가 없으면 `uri:` 모듈을 통해서 웹 페이지 접근 및 접속이 잘 되는지 확인.
5. 동작이 안되는 부분이 있으면 올바르게 동작하도록 수정.

정답 :)

아래와 작성 후 실행한다.

```
#!/usr/bin/env bash
# adhoc-exam.sh - simple ad-hoc steps for node4.example.com

set -e
H=node4.example.com
INV="${INVENTORY:-/etc/ansible/hosts}"

# 1) httpd 설치 및 기동
ansible "$H" -i "$INV" -b -m yum -a "name=httpd state=present"
ansible "$H" -i "$INV" -b -m service -a "name=httpd state=started
enabled=yes"
```

정답 :)

위의 내용 계속 이어서…

```
# 2) Hello World 페이지 생성 (copy: content:)
ansible "$H" -I "$INV" -b -m copy \
  -a "dest=/var/www/html/default.html content='Hello World' mode=0644"

# 3) 방화벽 서비스 등록 (firewalld)
ansible "$H" -I "$INV" -b -m service -a "name=firewalld state=started
enabled=yes"
ansible "$H" -I "$INV" -b -m firewalld -a "service=http  permanent=yes
state=enabled immediate=yes"
ansible "$H" -I "$INV" -b -m firewalld -a "service=https permanent=yes
state=enabled immediate=yes"
```

정답 :)

위의 내용 계속 이어서…

```
# 4) 접속 확인 (uri)
ansible "$H" -i "$INV" -b -m uri -a "url=http://127.0.0.1/default.html
status_code=200 return_content=yes"
echo "OK: http://$H/default.html 에서 Hello World!"

# chmod o+x adhoc-exam.sh
# adhoc-exam.sh
```

앤서블 인벤토리

추가 내용

앤서블 인벤토리

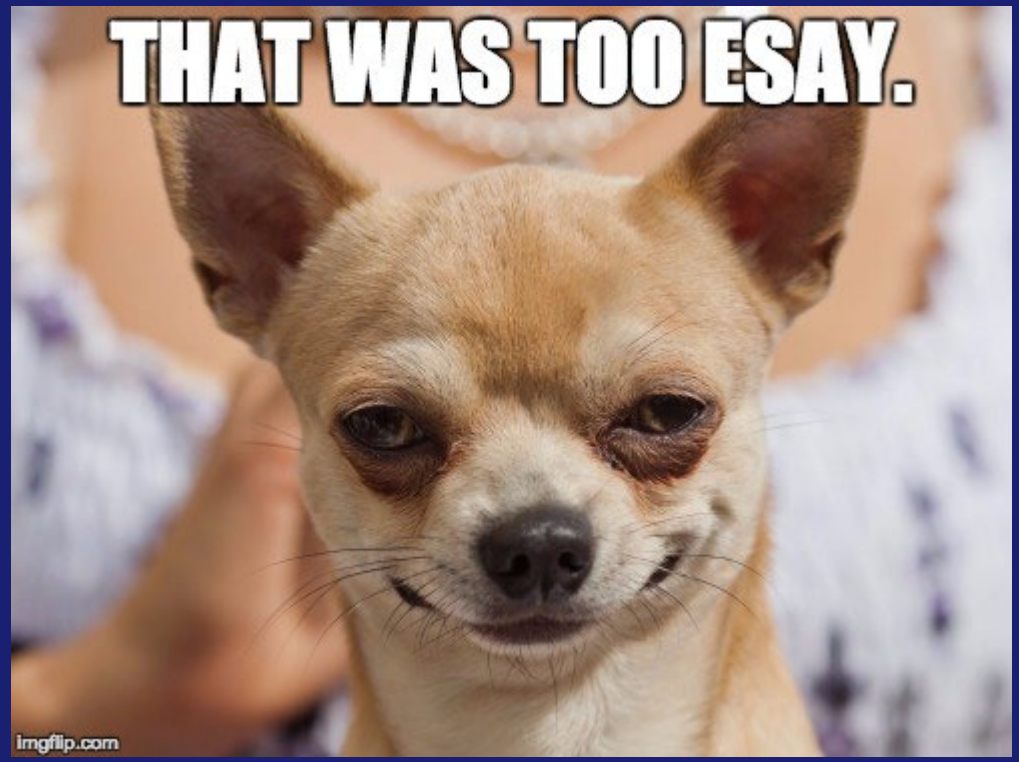
앞에서 간단하게 이야기 하였지만, 앤서블 인벤토리는 총 3가지 형식으로 지원한다.

1. TOML: Tom's Obvious Markup Language
2. INI: Initialization
3. YAML: YAML Ain't Markup Language

현재 ansible-core에서는 공식적으로 INI, YAML를 지원하며, TOML형식은 지원하고 있으나, 공식은 아니다.

형식	지원 상태	설명
INI	공식 지원	전통적인 `/etc/ansible/hosts` 형식 (`[group]` 등)
YAML	공식 지원	구조적이고 가독성이 높아 현대 앤서블의 기본 형식
TOML	**비공식/실험적**	일부 사용자 도구나 변환 스크립트로 가능하지만 `ansible-inventory`나 `ansible-core`에서는 기본 파서가 없음

왜 하필 INI?



Inventory changed

현재 사용하는 인벤토리는 "INI"형식을 사용하고 있다. 레드햇 및 앤서블 커뮤니티에서는 현재 "YAML"형태의 인벤토리를 사용을 권장하고 있다.

https://docs.ansible.com/ansible/latest/user_guide/intro_inventory.html

위의 부분은 변경이 되었다. 앤서블은 다음과 같은 순서로 인벤토리 파일을 확인한다. "inventory" 플러그인 설정이 안되어 있는 경우, 기본 값은 "auto"이며, 나머지는 아래와 같은 순서대로 파싱을 시도한다.

1. `host_list`
2. `script`
3. `auto`
4. `yaml`
5. `ini`
6. `toml`

YAML 인벤토리

서버 접근 시, 사용하는 **hosts**키워드는 다음과 같은 미리 예약된 옵션 혹은 값이 있다. 일반적으로 많이 사용하는 값은 아래와 같다.

1. **localhost:** 127.0.0.1와 같은 자기 자신 루프 백(loopback)
2. **all** : 인벤토리에(inventory)등록된 모든 호스트
3. **[group]:** 특정 그룹에만 적용하는 명령어 키워드.그룹명은 반드시 중괄호와 같이 사용한다.

YAML INVENTORY(INI 형식)

```
[PoC:children]
web
db
[web]
servera.node.example.com
serverb.node.example.com
[db]
serverc.node.example.com
serverd.node.example.com
[web:vars]
hostname=test.example.com
manager=tang
```

YAML INVENTORY(YAML 형식)

```
PoC:
  children:
    web:
      hosts:
        servera.node.example.com: {}
        serverb.node.example.com: {}
      vars:
        hostname: test.example.com
        manager: tang
    db:
      hosts:
        serverc.node.example.com: {}
        serverd.node.example.com: {}
```

TOML INVENTORY(TOML 형식)

```
[PoC]
children = [ "test", "db" ]
vars = {hostname=test.example.com, manager=tang }
[web.hosts]
servera.node.example.com = {}
serverb.node.example.com = {}
[db.hosts]
serverc.node.example.com = {}
serverd.node.example.com = {}
[web.vars]
hostname=test.example.com
manager=tang
```

인벤토리 확인

앤서블에서 인벤토리가 올바르게 구성이 되었는지 아래와 같이 명령어로 렌더링 및 확인이 가능하다.

```
# ansible-inventory -i <INVENTORY_FILE> --graph
# ansible-inventory -i <INVENTORY_FILE> --list group
# ansible-inventory -i <INVENTORY_FILE> --list host
# ansible-inventory -i <INVENTORY_FILE> --list host --vars
```

인벤토리 형식 비교

구분	INI 형식	YAML 형식	TOML 형식

형식 예시	`[web]`
	`server1`
	`server2`
		`yaml`
		`web:`
		`hosts:`
		`server1: {}`
		`server2: {}`
			`toml`
			`[web.hosts]`
			`server1 = {}`
			`server2 = {}`

Ansible 공식 지원 여부	완전 지원 (기본 형식)	완전 지원 (추천 형식)	비공식 또는 실험적 지원 (ansible-core 직접 지원 X)
가독성	보통 (단순하지만 변수 계층 표현이 제한됨)	매우 높음 (계층 구조 명확)	높음 (단, Python 생태계에서는 다소 생소)
변수 관리 편의성	제한적 (`[group:vars]`만 가능)	뛰어남 (`vars`, `children`, `hosts` 자유롭게 표현)	보통 (명확하지만 사용 예제 적음)

인벤토리 형식 비교

구분	INI 형식	YAML 형식	TOML 형식
대규모 인벤토리 확장성	중간 (단순하지만 중첩 불가)	높음 (계층적 구조로 복잡한 환경 표현 용이)	중간 (구조화 가능하지만 YAML만큼은 아님)
호환성 / 도구 지원	높음 (가장 오래된 형식, 대부분 도구 호환)	높음 (현대적 Ansible 및 AWX/AAP 추천)	낮음 (거의 사용되지 않음)
AWX / AAP 사용 시 추천도	지원은 하지만 YAML로 전환 권장	**강력히 추천 (표준화 및 유지보수 용이)**	사용 비권장
초보자 친화도	높음 (간단한 문법)	보통 (들여쓰기 주의 필요)	낮음 (활용 예제 부족)
현재 트렌드	구형 Playbook 또는 레거시 환경에서 여전히 사용	**신규 프로젝트 대부분 YAML 채택**	거의 사용되지 않음

INI 예제

INI 인벤토리 구성 및 구조는 다음과 같다.

```
[webservers]
web1 ansible_host=192.168.1.10
web2 ansible_host=192.168.1.11

[db]
db1 ansible_host=192.168.1.20
```

INI 예제

위의 내용 계속 이어서…

```
mail.example.com
[webservers]
foo.example.com
bar.example.com
[webservers:vars]
http_port=8080
http_name=www.example.com
[dbservers]
one.example.com
two.example.com
three.example.com
```

YAML 예제

YAML 형식은 INI 보다 길이가 길어진다. 개인적으로 INI 형식을 계속 유지를 추천.

```
all:
  children:
    webservers:
      hosts:
        web1:
          ansible_host: 192.168.1.10
        web2:
          ansible_host: 192.168.1.11
    db:
      hosts:
        db1:
          ansible_host: 192.168.1.20
```

YAML 예제

```yaml
all:
  hosts:
    mail.example.com:
  children:
    webservers:
      hosts:
        foo.example.com:
        bar.example.com:
      vars:
        http_port: 8080
        http_name: www.example.com
```

인벤토리 그룹 분류

앤서블 인벤토리는 다음과 같은 형식을 가지고 있다. 그룹 이름은 **대괄호로 구별 및 분류**한다.

```
[그룹 이름]
<호스트>

[그룹 이름:children]
<그룹이름>
```

인벤토리 와일드 카드

인벤토리에는 와일드 카드 방식이 사용이 가능하다. 이와 같은 방식으로 와일드 카드 선언이 가능하다.

```
192.168.10.*
192.168.[10-20].[1-100]
*.example.com
```

인벤토리 변수

```
[test]
node1.example.com package=httpd


[db]
10.10.5.2 package=mariadb


[was]
was1.example.com package=wildfly
```

인벤토리 오버라이드 변수

인벤토리에서 몇몇 앤서블 변수는 오버라이드가 가능하다. 아래는 특정 서버에 접근 시, 사용하는 리모트 유저 정보를 변경한다.

```
[test]
node1.example.com package=httpd ansible_user=slack ansible_port=8899

[db]
10.10.5.2 package=mariadb

[was]
was1.example.com package=wildfly
```

인벤토리 정보

인벤토리는 위의 내용을 기준으로 다음과 같은 내용을 가지고 있다.

- 호스트 이름
- 아이피 주소
- 호스트에서 사용하는 변수
- 앤서블 오버라이드 변수

인벤토리 옵션

인벤토리 파일 기반으로 실행 시 다음과 같이 실행한다.

```
# ansible-playbook -i hosts <PLAYBOOK>
```

ansible.cfg

인벤토리 파일은 일반적으로 **inventory**라는 이름으로 구성함. 영구적으로 고정을 원하는 경우 **ansible.cfg** 에서 변경이 가능함.

```
# cat ansible.cfg
[defaults]
inventory = <INVENTORY_FILE>
host_key_checking = False
remote_user = <SSH_USER>
remote_port = <SSH_PORT>
[privilege_escalation]
become=true
```

연습문제

ansible.cfg에는 다음과 같은 내용이 구성이 되어야 한다. 디렉터리를 **inventory_lab**으로 생성 후 진행한다.

1. inventory_cloud라는 파일을 기본 인벤토리 파일로 불러와야 한다.
2. ssh 접근 포트는 8822번을 사용한다.
   - 위의 설정은 노드 2/3번에 각각한다.
   - /etc/sshd/sshd_config.d/
   - 02-port.conf
3. 접근 시 사용하는 사용자는 ansible, 사용자의 비밀번호는 ansible이다.
   - 리모트 접근 사용자는 ansible로 한다.
   - ansible.cfg
   - 각 노드에는 해당 사용자가 구성이 되어 있어야 됨.

연습문제

4. sudo접근이 가능하도록 wheel그룹 설정을 각 노드에 올바르게 한다.
   - /etc/sudoers.d/nopasswd
   - grep -I wheel /etc/sudoers
5. inventory_cloud라는 설정 파일에는 다음과 같은 서버를 등록한다. 등록되는 그룹의 이름은 noexist_server으로 명시한다.
   - 10.10.10.1
   - 10.20.30.5
   - 192.168.90.*

연습문제

`inventory_cloud`에는 **다음과 같이 설정 및 구성한다.**

1. web, db라는 두개의 그룹이 존재하고 각각 그룹에 서버 하나씩 할당한다.
2. web에는 web.lab.example.com, db에는 db.lab.example.com할당
3. 할당된 노드에는 nodename이라는 이름의 변수에 web.example.com, db.example.com라는 호스트 이름을 할당
4. together라는 그룹에 web, db가 동시에 사용이 가능하도록 한다

정답 :)

예제는 다음과 같다.

```
# vi ansible.cfg
[defaults]
inventory = inventory_cloud
ansible_ssh_port = 8877
ansible_ssh_user = ansible
ansible_ssh_pass = ansible
```

정답 :)

예제는 다음과 같다.

```
# vi inventory_cloud
[noexist_group]
10.10.10.1
10.20.30.5
192.168.90.*
```

정답 :)

예제는 다음과 같다.

```
# vi inventory_cloud
[web]
web.lab.example.com
[db]
db.lab.example.com
[together:children]
web
db
# ansible -i inventory_cloud --list-hosts
```

모듈

모듈 활용

MODULE

앤서블 모듈은 확장이 가능하며 확장 기능을 제공하는 프로그램. 현재 앤서블 모듈은 두 가지로 나누어서 기능 제공 하고 있음

1. `ansible.core.*`
2. `ansible.posix.*`
3. `ansible.builtin`

앤서블에서 제공하는 핵심 기능. 일반적인 핵심 기능은 `POSIX` 모듈에서 제공한다. 여기에 **`copy`, `file`, `fetch`**와 같은 자주 사용하는 기능이 포함이 되어 있다.

현재 2023년도 부터 앤서블은 앤서블 모듈 **도메인(네임스페이스, FQCN(Fully Qualified Collection Name))** 권장하고 있다. Ansible 2.9까지는 기존 방식, 2.10이후 부터는 FQCN으로 권장. 2.10/11부터는 단일 이름은 호환성으로 제공하고 있다.

linux(rhel)-system-roles(roles)

앤서블에서 제공하는 **확장 기능**. 리눅스 배포판에 상관없이 사용이 가능하다.

일반적으로 리눅스 배포판에는 핵심 기능만 **역할(role)**로 제공하며, 추가 기능은 `linux-system-roles`에서 추가로 내려받기가 가능하다.

아래 링크에서 확인이 가능하다.

https://galaxy.ansible.com/ui/repo/published/fedora/linux_system_roles/content/

사용자가 원하는 경우 직접 자신만의 roles를 생성하여 사용이 가능하다.

ansible.collection

앤서블 확장 기능. 코어 기능에서 확장된 기능이며, 이걸 보통 컬렉션(collection)이라고 부른다.

컬렉션에는 밴더사에서 제공하는 모듈 기능도 포함이 되어 있다. 이전 앤서블 엔진 버전에서는 컬렉션 개념이 없었지만, 앤서블 코어로 넘어 오면서 추가가 되었다.

컬렉션은 `FQCN(fully-qualified collection name)`라고 부름. 마치 도메인 시스템의 `FQDN(fully-qualified domain name)`과 비슷한 이름.

예를 들어서 대표적으로 많이 사용하는 **`Linux System Role`**도 컬렉션에 포함 되어 있다.

컬렉션은 `ansible-galaxy`통해서 검색 및 설치가 가능하다. 하지만, 버전에 따라서 갤럭시에서 컬렉션 검색이 되지 않기 때문에 직접 웹 사이트에서 직접 검색을 권장한다.

ansible.community

커뮤니티는 검증이 되지 않는 사용자 혹은 특정 그룹에서 만들어서 배포하는 **Role** 혹은 **Module**이다.

커뮤니티 컬렉션에서 사용하는 많은 모듈은 레드햇 혹은 파트너사에서 검증 후, 정식 모듈로 포함이 되는 경우도 있다.

```
# ansible-galaxy collection list
# ansible-galaxy collection install community.general
```

ansible.builtin

앤서블 코어에서 같이 배포하는 기본 모듈이다. 모든 앤서블 버전에서 동일하게 사용이 가능하며, 호출 이름은 **`ansible.builtin.*`**으로 시작한다.

```
- name: copy the httpd.conf
  ansible.builtin.copy:
    src: httpd.conf
    dest: /tmp/httpd.conf
```

모듈 명령어 목록 출력(참고용)

```
- hosts: localhost
  vars:
    ansible_doc: /usr/share/doc/ansible-core
    modules_all: "{{ (out.stdout|fro
    modules_str: |
      {% for i in modules_all %}
      {% set arr = i.split('.') %}
      - {collection: {{ arr[:-1]|join('.')|d('ansible.builtin', true) }},
module: {{ arr[-1] }}}
      {% endfor %}
```

앤서블에서 Jinja2를 사용해서 템플릿팅 작업을 수행한다.
이러한 이유로 중괄호 "}"가 많이 보인다.

모듈 명령어 목록 출력(참고용)

```
    modules_grp: "{{ modules_str|from_yaml|groupby('collection') }}"

    collections_list: "{{ modules_grp|map('first') }}"

    modules_lists: "{{ modules_grp|map('last')|map('map',
attribute='module') }}"

    collections: "{{ dict(collections_list|zip(modules_lists)) }}"
```

앤서블에서 Jinja2를 사용해서 템플릿팅 작업을 수행한다.
이러한 이유로 중괄호 "}"가 많이 보인다.

모듈 명령어 목록 출력(참고용)

```
tasks:
  - command: "{{ ansible_doc }} -t module -l -j"
    register: out
  - debug:
      var: out.stdout_lines
    when: false
  - debug:
      var: out.stdout
    when: false
```

모듈 명령어 목록 출력(참고용)

```
- debug:
    var: collections.keys()|list|to_yaml
  when: true
- debug:
    var: collections['ansible.builtin']|to_yaml
  when: true
```

모듈 명령어 목록 출력(참고용)

```
- debug:
    var: modules_all
  when: false
- debug:
    var: modules_str
  when: false
```

모듈 명령어 목록 출력(참고용)

```
- debug:
    var: collections.keys()|list|to_yaml
  when: true
- debug:
    var: collections['ansible.builtin']|to_yaml
  when: true
```

모듈 명령어 목록 출력(참고용)

```
- debug:
    var: collections['community.digitalocean']|to_yaml
  when: true
- debug:
    var: collections['community.general']|to_yaml
  when: true
```

모듈 명령어 목록 출력(참고용)

제일 쉬운 방법은 명령어로 앤서블에서 사용 가능한 모듈 정보 확인이 가능하다.

```
# ansible-doc -l
# ansible-doc --list
# ansible-navigator doc --list
```

위의 플레이북 방식은 너무 복잡하니 권장하지 않는다.

모듈 및 문서 위치

앤서블 모듈 설치 위치

1. /lib/python3.8/site-packages/ansible_collections
2. /usr/lib/python3.9/site-packages/ansible/plugins/
3. /usr/lib/python3.9/site-packages/ansible/modules/

앤서블 표준 컬렉션 모듈 목록

https://docs.ansible.com/ansible/latest/collections/community/general/index.html#plugin-index

앤서블 문서 위치

```
# man -k ansible
```

모듈(copy)

제일 많이 사용하는 모듈 copy기반으로 기능을 구현하면 다음과 같다.

```
- name: copy an issue file to remote server
  copy:
    src: /etc/hostname
    dest: /tmp/hostname.bak
    remote_src: yes
```

모듈(copy)

옵션	의미
`src`	로컬에 있는 파일 경로 (컨트롤 노드 기준)
`dest`	복사 대상 서버의 파일 경로
`owner`	파일 소유자
`group`	파일 그룹
`mode`	파일 권한 (`'0644'` 형태로 문자열로 입력)
`backup`	기존 파일 백업 여부 (`yes` 지정 시 `<filename>.~timestamp~` 형태로 백업됨)
`content`	파일을 직접 작성 (파일 없이 즉석 생성 가능)

모듈(패키지)

패키지 설치시 사용하는 모듈. 패키지 관련 모듈은 운영체제 및 배포판 별로 다르게 제공한다. 일반적으로 레드햇 계열은 다음과 같은 패키지 관리자 제공한다.

dnf/yum

레드햇 계열 배포판에서 사용하는 네트워크 패키지 관리자. 패키지 설치 및 삭제 자동으로 의존성부분을 확인한다. 다만, 온라인 상태가 아닌 경우, 올바르게 동작하지 않는다.

zypper

수세에서 사용하는 패키지 매니저. 수세는 기본적으로 RPM를 사용하지만, 레드햇과 다른 네트워크 패키지 매니저를 사용한다.

모듈(패키지)

RPM

단일 패키지 관리는 RPM명령어를 통해서 사용한다. 레드햇 계열은 RPM기반으로 관리가 되며, 패키지에 문제가 있을 때, 해당 명령어로 장애처리가 가능하다.

모듈(패키지 관리자)

```yaml
- name: yum로 Nginx 설치
  ansible.builtin.yum:
    name: nginx
    state: present
- name: dnf로 Nginx 설치
  ansible.builtin.dnf:
    name: nginx
    state: present
```

모듈(패키지 관리자)

```
- name: zypper로 Nginx 설치
  ansible.builtin.zypper:
    name: nginx
    state: present
- name: apt로 Nginx 설치
  ansible.builtin.apt:
    name: nginx
    state: latest
    update_cache: yes
```

모듈(패키지(dnf,yum))

둘의 옵션은 거의 동일하다. 자주 사용하는 옵션은 다음과 같다.

옵션	의미
`name`	패키지 이름 (리스트도 가능)
`state`	`present, latest, absent`
`enablerepo / disablerepo`	특정 리포지토리 활성/비활성
`update_cache`	메타데이터 캐시 갱신

모듈(패키지(dnf,yum,apt,zypper))

공통적인 옵션은 전부 다 비슷하게 공유한다. 상세한 설정 및 옵션이 필요한 경우 doc를 참조.

옵션	의미
`name`	패키지 이름 (리스트도 가능)
`state`	`present, latest, absent`
`enablerepo / disablerepo`	특정 리포지토리 활성/비활성
`update_cache`	메타데이터 캐시 갱신

모듈(저수준 패키지)

앤서블에서는 **저-수준 패키지** 관리자 명령어인, `rpm`에 대해서는 직접적인 관리 모듈을 제공하지 않는다. 관리하기 위해서는 다음 명령어로 관리가 가능하다.

```
- name: remove the httpd RPM package
  shell: rpm -e httpd
```

모듈(저수준 패키지)

앤서블에서는 **저-수준 패키지** 관리자 명령어인, `rpm/deb`는 이전에 지원하지 않았지만, 현재는 별도로 지원하고 있다.

```
- name: remove the httpd RPM package
  ansible.builtin.shell: rpm -e httpd
- name: 로컬 .deb 패키지 설치
  ansible.builtin.deb:
    name: /tmp/custom-agent_1.0.2_amd64.deb
    state: present
- name: 로컬 RPM 설치
  ansible.builtin.rpm:
    name: /tmp/custom-agent-1.0.2-1.x86_64.rpm
    state: present
```

모듈(파일)

```yaml
- file:
    path: /tmp/test
    owner: foo
    group: foo
    mode: 0644
- ansible.builtin.file:
    path: /tmp/test
    owner: foo
    group: foo
    mode: 0644
```

모듈(파일)

`"file:"` 디렉터리/파일/링크와 같은 자원을 생성하거나 혹은 소유권 및 퍼미션 변경 시 사용하는 명령어. 자주 사용하는 옵션만 나열하였다.

옵션	설명
`path`	변경이 될 디렉터리 및 파일 대상.
`perm`	디렉터리 및 파일의 퍼미션.
`owner`	파일 및 디렉터리의 소유자.
`group`	파일 및 디렉터리의 그룹 소유자.
`mode`	심볼릭 및 혹은 8진수 기반으로 퍼미션 설정.
`state`	링크/디렉터리 혹은 비어 있는 파일(touch)를 생성하는 옵션.

모듈(디버깅)

debug 모듈은 사용자 혹은 시스템 변수 출력이나 혹은 모듈의 특정 정보를 확인 시 많이 사용하는 모듈이다.

```
- debug:
    msg: "{{ username }}"
- debug:
    msg: "{{ ansible_facts['distribution'] }}"
- debug:
    msg: "Hello Ansible World!"
```

모듈(명령어 실행)

관리가 되고 있는 노드에 직접 명령어를 실행한다. 앤서블에서 두 개의 모듈을 지원하는데, 기능은 비슷하지만, 동작 방식이 조금 다르다.

shell

사용자가 사용하는 쉘 기반으로 명령어를 실행한다. 기존에 사용하던 쉘 변수를 그대로 사용이 가능하기 때문에 스크립트 실행 시, 많이 사용한다.

command

간단하게 명령어를 실행 시 사용한다. 위의 모듈 shell과 다르게 쉘 환경 변수를 사용하지 않기 때문에, 간단한 명령어 실행 시 사용한다. 또한, 명령어는 쉘에서 실행이 안되며, 파이썬 모듈을 통해서 명령어 실행한다.

모듈(서비스)

현재 리눅스는 2가지 **초기화 관리자(initiator program)**를 가지고 있다. 이전의 System V의 스크립트 기반, 두 번째는 systemd기반으로 관리 및 운영하는 서비스 관리자.

1. /bin/init
2. /bin/systemd

최근 리눅스는 systemd기반으로 사용하고 있기 때문에, systemd모듈을 사용한다. System V 계열은 service 모듈 사용을 권장한다.

```
- systemd:
    name: httpd.service
    state: restarted
- service:
    name: httpd
    state: restarted
```

모듈(서비스)

기본적인 옵션은 systemd, servic나 거의 동일하다. 추가 기능이 필요한 경우 ansible-doc를 참조.

옵션명	설명	기본값 / 비고
`name`	제어할 서비스 이름	(필수) 예: `nginx`, `sshd`, `httpd`
`state`	서비스 상태 지정	`started`, `stopped`, `restarted`, `reloaded`, `absent`
`enabled`	부팅 시 자동 시작 여부	`yes/no`
`sleep`	서비스 재시작 전 대기 시간(초)	재시작 시 잠깐 대기하도록 설정 가능
`daemon_reload`	systemd 데몬 재로딩	`no` (기본)
`pattern`	프로세스 패턴 문자열로 서비스 존재 확인	PID 파일이 없을 때 유용

모듈(서비스)

기본적인 옵션은 systemd, service나 거의 동일하다. 추가 기능이 필요한 경우 ansible-doc를 참조.

옵션명	설명	기본값 / 비고
`masked`	서비스 마스킹 여부 (systemd 전용)	`yes` / `no`
`use`	사용할 서비스 관리 시스템 지정	예: `systemd`, `sysvinit`, `upstart`
`runlevel`	sysvinit 기반 OS에서만 사용	예: `3`
`force`	강제 재시작 또는 중지	일반적으로 `no`, 특정 시스템만 지원

모듈(사용자)

사용자 추가 모듈이다. 관리자 계정 이외, 사용자를 추가하기 위해서는 이 모듈로 사용자를 추가한다.

배포판 상관없이 사용이 가능하며, 다만 윈도우 서버 경우에는 사용하는 모듈이 다르다.

```
# user.yaml
- name: ansible 사용자 생성
  ansible.builtin.user:
    name: ansible
    comment: "Automation user"
    shell: /bin/bash
    groups: wheel
    append: yes
    state: present
```

모듈(사용자)

사용자 비밀번호 변경.

```
# passwd.yaml
- name: ansible 계정 생성 및 암호 지정
  ansible.builtin.user:
    name: ansible
    password: "{{ 'ansible123' | password_hash('sha512') }}"
    state: present
```

모듈(사용자)

사용자 제거.

```
# adduser.yaml
- name: 계정 및 홈 디렉터리 삭제
  ansible.builtin.user:
    name: tempuser
    state: absent
    remove: yes
```

모듈(템플릿)

같은 양식에 내용만 변경하는 경우 템플릿(template)를 사용한다. 템플릿은 Jinja2 형식을 사용하며, 확장자는 보통 j2, jin2라고 많이 사용한다.

```
# vi /templates/nginx.conf.j2
server {
    listen 80;
    server_name {{ hostname }};
    root /var/www/{{ web_dir }};
}
```

모듈(템플릿)

템플릿을 플레이북에 적용은 다음과 같이 한다. 템플릿에서 사용할 변수를 선언한다.

```
# vi nginx-template.yaml
- name: Nginx 설정 파일 배포
  hosts: web
  vars:
    hostname: example.com
    web_dir: html
```

모듈(템플릿)

템플릿을 호출하여 변수를 적용한다.

```
tasks:
  - name: 템플릿 파일을 복사하여 Nginx 설정 생성
    ansible.builtin.template:
      src: templates/nginx.conf.j2
      dest: /etc/nginx/conf.d/default.conf
      owner: root
      group: root
      mode: '0644'
```

모듈(템플릿)

옵션명	설명	기본값 / 비고
src	로컬(컨트롤 노드)에 있는 Jinja2 템플릿 파일 경로	(필수) `.j2` 확장자 권장
dest	원격 서버에 템플릿을 복사할 경로	(필수) 절대경로 사용
owner	복사된 파일의 소유자	지정하지 않으면 기본 SSH 사용자
group	복사된 파일의 그룹	지정하지 않으면 기본 SSH 사용자 그룹
mode	퍼미션 (권한) 설정	예: `'0644'` 또는 `u=rw,g=r,o=r`
force	대상 파일이 이미 있을 때 덮어쓸지 여부	`yes` (기본)
backup	덮어쓰기 전 백업 파일 생성 여부	`no` (기본), `yes` 시 `~timestamp~` 파일 생성

모듈(템플릿)

옵션명	설명	기본값 / 비고
validate	복사 전 파일의 유효성 검증 명령	예: `'nginx -t -c %s'` → 실패 시 적용 안 함
variable_start_string	Jinja2 변수 시작 구분자 변경	기본값: `{{`
variable_end_string	Jinja2 변수 끝 구분자 변경	기본값: `}}`
block_start_string	Jinja2 제어문(`{% ... %}`) 시작 구분자 변경	기본값: `{%`
block_end_string	제어문 끝 구분자 변경	기본값: `%}`
trim_blocks	템플릿 렌더링 시 개행 처리 제어	`yes` (기본) → 블록 끝 개행 제거
lstrip_blocks	블록 앞의 공백 제거 여부	`no` (기본), `yes` 시 깔끔한 포맷

모듈(템플릿)

옵션명	설명	기본값 / 비고
newline_sequence	개행 문자 지정	`\n`(기본), Windows용은 `\r\n`
follow	심볼릭 링크를 따라갈지 여부	`no`
selevel / serole / setype / seuser	SELinux 관련 속성 설정	SELinux 사용 환경에서 필요 시만 지정
dest_is_directory	`dest`가 디렉터리일 때 파일 이름 유지 여부	`no` (기본)
unsafe_writes	파일 쓰기 시 원자적(atomic) 쓰기 비활성화	`no` (기본), NFS 환경에서 필요할 때 `yes`

모듈(fetch)

fetch는 원격 서버에 있는 파일을 로컬(control)으로 복사하는 기능이다. **원격 -> 로컬** 방향으로 복사한다.

```
# vi fetch.yaml
- name: 원격 로그 파일 수집
  hosts: all
  tasks:
    - name: messages 로그를 로컬로 가져오기
      ansible.builtin.fetch:
        src: /var/log/messages
        dest: /tmp/logs/
        flat: no
```

모듈(lineinfile)

특정 파일 한 줄을 **삽입/추가** 혹은 수정하는 모듈이다. 보통 키-값 형태로 구성된 설정 변경에 유용하다.

```
# lineinfile.yaml
- name: SSH Root 로그인 허용
  ansible.builtin.lineinfile:
    path: /etc/ssh/sshd_config
    regexp: '^PermitRootLogin'
    line: 'PermitRootLogin yes'
    backup: yes
```

모듈(blockinfile)

1. blockinfile은 지정한 파일 안에 여러 줄의 텍스트 블록을 삽입하거나 업데이트
2. 앤서블이 관리하는 영역을 marker로 구분해 자동으로 업데이트
3. 설정 블록 추가, 주석 블록 삽입 등에 자주 사용

```
# vi blockinfile.yaml
- name: Apache VirtualHost 블록 추가
  ansible.builtin.blockinfile:
    path: /etc/httpd/conf/httpd.conf
    marker: "# {mark} ANSIBLE MANAGED BLOCK"
    block: |
      <VirtualHost *:80>
          ServerName example.com
          DocumentRoot /var/www/html
      </VirtualHost>
```

모듈(unarchive)

unarchive는 tar나 zip 형태의 압축 파일을 자동으로 해제하는 모듈이다. 원격 서버에 파일이 이미 있을 경우 remote_src: yes를 사용하며, 애플리케이션 배포나 설정 파일 패키지 설치 시 자주 활용된다.

```
# vi unarchive.yaml
- name: 웹 애플리케이션 압축 해제
  ansible.builtin.unarchive:
    src: /tmp/webapp.tar.gz
    dest: /var/www/html/
    remote_src: yes
```

옵션	의미
src	압축 파일 경로 (로컬 또는 원격)
dest	압축 해제할 대상 경로
remote_src	원격 파일 여부 (yes: 이미 대상 서버에 있음)
extra_opts	압축 해제 시 추가 옵션 지정 가능

MODULE DOCUMENT

모듈에 대한 자세한 옵션을 보기 위해서는 다음과 같은 명령어로 실행한다. 사용 가능한 모듈 목록을 확인하기 위해서는 아래 명령어로 목록 확인이 가능하다.

```
# ansible-doc <MODULE NAME>
# ansible-doc -l
```

연습문제

설치한 앤서블에서 간단하게 다음 모듈을 실행해본다. 테스트 용도로 만든 인벤토리 파일을 사용해서 작업을 수행한다. 이 작업은 node1/2번에서 수행한다.

1. 'file'모듈을 사용하여 "/tmp/test"라는 파일을 생성한다.
    - "path" 옵션을 사용한다.
    - 사용자 및 그룹을 "ansible"으로 변경한다.
2. 사용자 "test1", "test2"를 생성한다.
    - 암호는 test으로 통일한다.
    - 그룹은 adm, wheel에 가입한다.
3. 패키지를 설치한다.
    - httpd, vsftp패키지를 설치한다.
    - squid패키지를 설치한다.

연습문제

위의 내용 계속 이어서…

4. 서비스를 동작하도록 구성한다.
    - httpd/vsftpd/squid 서비스를 시작한다.
5. 설치된 패키지 및 사용자 제거한다.
    - 위에서 설치한 패키지를 전부 제거한다.
    - 위에서 생성한 사용자를 전부 제거한다.
6. 'ansible-doc'명령어를 통해서 'dnf'명령어에 대한 도움말을 확인한다.
    - 도움말을 확인하여 간단하게 "httpd"패키지를 설치한다.

정답 :)

위의 내용을 아래와 같이 작성한다.

```
---
- name: Ansible Basic Practice
  hosts: node1,node2
  become: yes
  tasks:
    # 1. /tmp/test 파일 생성
    - name: Create /tmp/test file
      ansible.builtin.file:
        path: /tmp/test
        state: touch
        owner: ansible
        group: ansible
        mode: '0644'
```

정답 :)

위의 내용 계속 이어서…

```
    # 2. 사용자 생성
    - name: Create users test1 and test2
      ansible.builtin.user:
        name: "{{ item }}"
        password: "{{ 'test' | password_hash('sha512') }}"
        groups: adm,wheel
        state: present
      loop:
        - test1
        - test2
```

정답 :)

위의 내용 계속 이어서…

```
    # 3. 패키지 설치
    - name: Install web and ftp packages
      ansible.builtin.dnf:
        name:
          - httpd
          - vsftpd
          - squid
        state: present
      register: package_result
```

정답 :)

위의 내용 계속 이어서…

```
    # 4. 서비스 시작
    - name: Start httpd/vsftpd/squid services
      ansible.builtin.service:
        name: "{{ item }}"
        state: started
        enabled: yes
      loop:
        - httpd
        - vsftpd
        - squid
```

2025-12-26

정답 :)

위의 내용 계속 이어서…

```
    # 5-1. 설치한 패키지 제거
    - name: Remove installed packages
      ansible.builtin.dnf:
        name:
          - httpd
          - vsftpd
          - squid
        state: absent
```

정답 :)

위의 내용 계속 이어서…

```
    # 5-2. 생성한 사용자 제거
    - name: Remove created users
      ansible.builtin.user:
        name: "{{ item }}"
        state: absent
        remove: yes
      loop:
        - test1
        - test2
```

정답 :)

위의 내용 계속 이어서…

```
    # 6. ansible-doc 확인 (도움말 확인은 설명으로 대체)
    - name: Display dnf module info (example)
      ansible.builtin.debug:
        msg:
          - "명령어 예시: ansible-doc dnf"
          - "도움말을 확인한 뒤 아래와 같이 httpd 설치 가능"
          - "ansible node1 -m dnf -a 'name=httpd state=present'"
```

앤서블 태스크

tasks(pre_tasks, post_tasks)

role

delegate_to

설명

앤서블에서 사용하는 작업 관리 방법에 대해서 전체적으로 알아본다. 크게 두 가지 형태로 작업을 구별 및 구분하여 사용/분리가 가능하다.

tasks

앤서블 작업이 수행이 되는 영역. 모든 작업은 Top-Down형태로 수행이 된다. 기본작업은 모듈기반으로 각각 작업이 수행된다.

roles

역할은 작업의 독립적인 형태이며, 오브젝트 개념으로 보면, 각각 함수(function)들이 모인, 클래스(class)와 같은 개념적인 역할을 구성한다. 이 부분은 뒤에서 더 다루도록 한다.

역할(roles)

다중 작업은 **roles**과 함께 사용이 가능하다. **tasks:**보다는 **roles:**키워드와 함께 더 많이 사용한다.

이 부분에 대해서는 뒤에서 더 다루도록 한다.

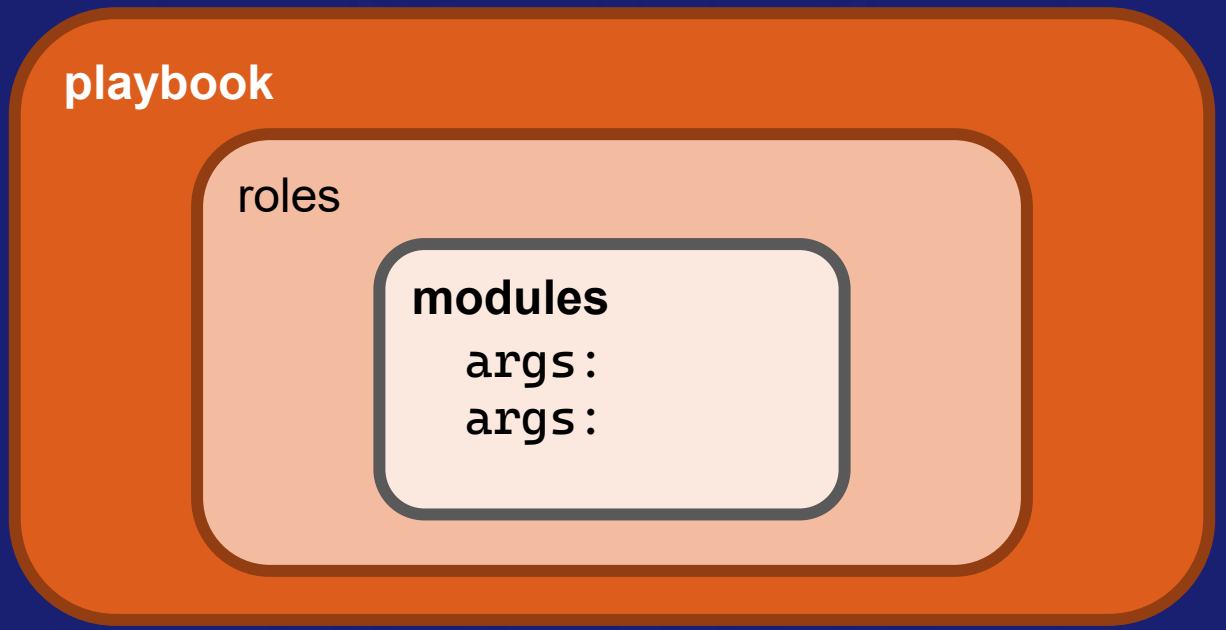
```
pre_tasks:
roles:
  - { role: vsftpd } 혹은 vsftpd
post_tasks:
```

모든 작업이 수행 전 사용

tasks가 role로 변경. 한 개 이상의 role 조합으로 사용이 가능하다.

마지막에 실행이 되는 작업

작업/역할



pre/main/post

앤서블 태스크는 기본적은 **tasks기본으로 동작**한다. 하지만, 여러가지 이유로 **전/후 작업 처리**가 필요한 경우 `pre_tasks, task, post_tasks` 총3개의 작업으로 나누어진다.

`pre_tasks:`

본 작업(tasks)가 실행 되기전에 특정 작업을 수행한다.

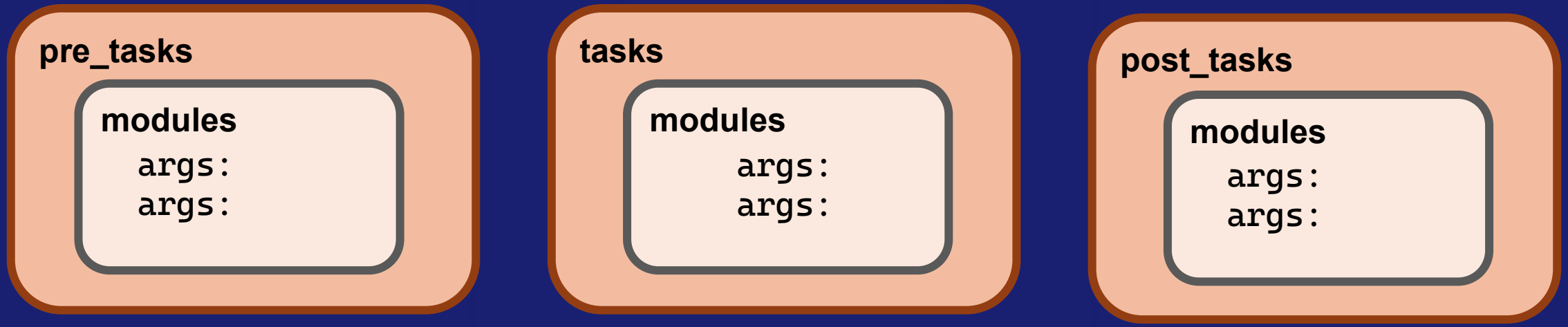
`tasks:`

본 작업이 수행이 되는 영역. 일반적으로 제일 많이 사용하는 작업 지시자다.

`post_tasks:`

본 작업이 완료 후, 후속 작업이 필요한 경우 사용한다.

pre/main/post



pre/main/post

본 작업이 발생하기 전에 작업한다. **pre_tasks** 보통 **tasks(혹은** role)과 수행이 되며 단독으로 사용하지 않는다.

```
pre_tasks:
  - name: updates all of packages
    yum:
      name: "*"
      state: latest
```

pre/main/post

보통 `pre_tasks` 이후에 본 작업이 수행이 되며, 단독 실행이 가능하다.

```yaml
tasks:
  - name: install a vsftpd package
    yum:
      name: vsftpd
      state: present
```

pre/main/post

`task`작업이 수행이 된 다음에 후 처리 작업을 보통 `post_tasks`에서 수행한다. 역시 `post_tasks`만 단독으로 사용하는 경우는 거의 없다.

```
post_tasks:
  - name: start and enable the vsftpd service
    service:
       name: vsftpd
       state: started
       enabled: true
```

pre/main/post

```
pre_tasks:
  - name: updates all of packages
    yum:
      name: *
      state: present
tasks:
  - name: install a vsftpd package
    yum:
      name: vsftpd
      state: present
```

첫번째 작업 수행

주 작업 수행

pre/main/post

```
post_tasks:
  - name: start and enable the vsftpd service
    service:
      name: vsftpd
      state: started
      enabled: yes
```

후속 작업 수행

pre/main/post

```
- hosts: web
  pre_tasks:
  - name: this is the hello message from ansible
    debug:
      msg: "This message shows from pre tasks"
  roles:
  - { role: vsftpd }
  post_tasks:
  - name: this is the by message from ansible
    debug:
      msg: "This message shows from post tasks"
```

tasks가 role로 변경.
한 개 이상의 role 조합으로 사용이 가능하다.

pre/main/post + tasks

혹은 roles와 같이 사용하기도 한다.

```
- hosts: web
  pre_tasks:
  - name: this is the hello message from ansible
    debug:
      msg: "This message shows from pre tasks"
  roles:
  - { role: vsftpd }
  tasks:
  - debug:
      msg: "This is Tasks session. This part will run after run to "roles".
```

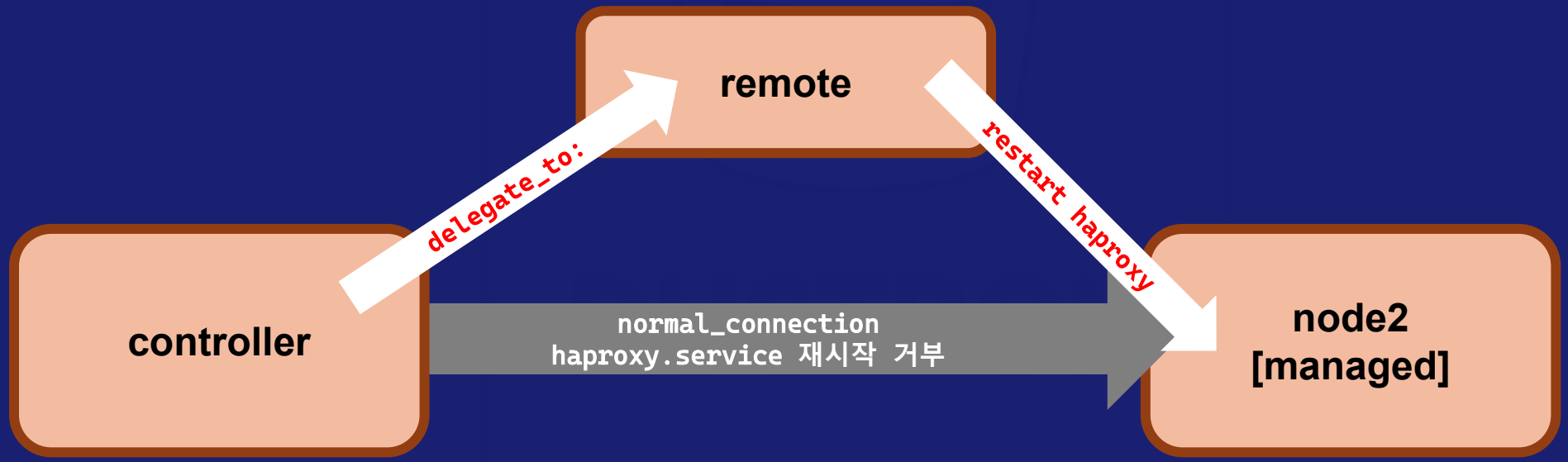
pre/main/post + tasks

위의 내용 계속 이어서…

```
post_tasks:
  - name: this is the by message from ansible
    debug:
      msg: "This message shows from post tasks"
```

delegate_to

작업 수행 시, "hosts:"에 명시된 서버 그룹 혹은 서버에 직접 접근이 아니라, 한번 더 다른 서버에 걸쳐서 작업을 수행한다. 이를 "delegate"라고 하며 호출 시 사용하는 지시자는 'delegate_to'를 사용한다.



delegate_to

```
# vi delegate_to.yaml
- hosts:
    - node1
    - node2
  tasks:
  - name: restart nginx
    service:
      name: nginx
      state: restart
    delegate_to: node5.example.com
```

연습문제

앤서블 작업을 다음 조건으로 구성한다.

pre, post 및 tasks를 사용하여, 전후 작업을 구성한다.

1. 선 작업은 "we are going to install the httpd package"메시지 출력.
    - /var/www/html/index.html
    - 만약, 위의 디렉터리 및 파일이 없는 경우, 'file'로 생성 및 구성.
2. 주 작업에서는 아파치 패키지를 최신 패키지로 설치.
3. 후 작업에서는 "the package has been installed"라는 메시지를 출력.
    - debug, msg.

연습문제

모든 노드의 호스트 이름 설정한다.

1. node[1-4].example.com으로 올바르게 호스트 이름(FQDN)를 설정한다.
    - 'hostname'모듈을 사용해서 모든 서버에 적절하게 FQDN으로 도메인 및 호스트 네임 구성한다.

대행서버(delegate)서버는 node1가 대행, 다음과 같은 작업을 수행한다.

1. 대행 서버는 반드시 node1이 구성한다.
2. node1에서 다른 노드에 문제 없이 접근이 가능해야 한다.
3. 각 노드에 설치가 된 squid패키지를 설치한다.

답안 :)

```
---
- name: Pre and Post Tasks Example
  hosts: all
  become: yes
  pre_tasks:
    - name: Display starting message
      ansible.builtin.debug:
        msg: "We are going to install the httpd package"
```

답안 :)

```
- name: Ensure /var/www/html directory exists
  ansible.builtin.file:
    path: /var/www/html
    state: directory
    mode: '0755'
```

답안 :)

```
- name: Ensure /var/www/html/index.html file exists
  ansible.builtin.file:
    path: /var/www/html/index.html
    state: touch
    mode: '0644'
```

답안 :)

```yaml
tasks:
  - name: Install the latest httpd package
    ansible.builtin.dnf:
      name: httpd
      state: latest
post_tasks:
  - name: Display completion message
    ansible.builtin.debug:
      msg: "The package has been installed"
```

답안 :)

```
---
- name: Set hostnames for all nodes
  hosts: all
  become: yes
  tasks:
    - name: Set FQDN as nodeX.example.com
      ansible.builtin.hostname:
        name: "{{ inventory_hostname }}.example.com"
```

답안 :)

```yaml
- name: Delegate tasks to node1
  hosts: node1
  become: yes
  tasks:
    - name: Check SSH connectivity to all other nodes
      ansible.builtin.ping:
      delegate_to: "{{ item }}"
      loop:
        - node2
        - node3
        - node4
```

답안 :)

```
- name: Install squid package on all nodes from node1
  ansible.builtin.dnf:
    name: squid
    state: present
  delegate_to: "{{ item }}"
  loop:
    - node1
    - node2
    - node3
    - node4
```

앤서블 변수

변수선언 및 구성

앤서블 시스템 변수

VAR

앤서블도 다른 프로그램 언어처럼 변수를 사용한다.

다만, 우리가 알고 있는 변수와는 다른 방법으로 폭 넓게 지원을 하고 있다. 맨 처음 앤서블을 접근하는 사용자가 어려운 부분 중 하나가 바로 변수이다.

아래는 앤서블에서 지원하는 변수 범위이다.

1. `inventory`
2. `group_vars`
3. `host_vars`
4. `role vars`
5. `앤서블 시스템 변수(ansible_facts)`

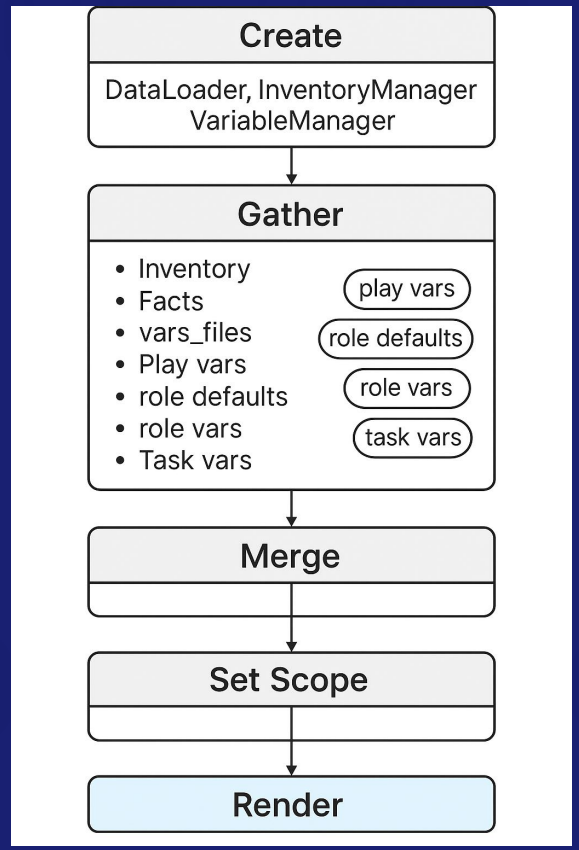
VAR IN INVENTORY

var inventory는 다음처럼 보통 선언한다. 인벤토리 파일에 아래와 변수를 선언이 가능. **var1**은 변수 이름이며, **=**변수 대입 연산자 **hello**는 영문 소문자로 변수 값 할당.

앤서블에서 사용하는 모든 변수는 **public type**으로 구성이 되어 있다. 앤서블은 private변수는 없으며 **앤서블 런타임** 동작이 완료가 되면 변수는 그 즉시 heap/stack **메모리**에서 제거가 된다.

```
[server]
test.lab.example.com var1=hello var2=world ansible_host=10.10.1.1
```

앤서블 변수 랜더링


Create
DataLoader, InventoryManager
VariableManager
Gather
Inventory
Facts
vars_files
Play vars
role defaults
role vars
Task vars
play vars
role defaults
role vars
task vars
Merge
Set Scope
Render

일반변수

일반 변수는 보통 플레이북이 다음처럼 작성 및 구성한다.

```
# vi variable1.yaml
---
- name: this is first vars
  hosts: localhost
  vars:
    division: middleware
  tasks:
  - name: Hello world
    command: echo "Hello world"
  - debug:
        msg: "{{ division }}"
```

전역 키워드 선언

앞에서 사용하였던 슬라이드.

```
- hosts:
  vars:
    - var1
    - var2
  become:
  remote_user:
  task
```

보통 해당 영역에 변수를 선언한다.

리스트

앤서블 리스트는 매우 간단하게 선언한다. 리스트의 이름 그리고, 리스트가 가져가는 값을 선언한다.

```
# vi variable2.yaml
---
- name: this is first list
  hosts: localhost
  vars:
    fruits_list:
      - apple
      - mango
      - pineapple
  tasks:
  - name: shows your fruits
    command: echo "{{ fruits_list }}"
```

리스트

리스트는 변수 앞에 "-"대시가 붙는다. 또한 리스트는 맨 앞에 사용할 리스트의 이름이 붙는다.

```
# vi variable3.yaml
---
- name: this is first list
  hosts: localhost
  vars:
    region:
      - northeast
      - southeast
      - midwest
  tasks:
  - name: echo the the first element value
    command: echo "{{ region[0] }}"
```

딕셔너리

딕셔러너리는 리스트와 비슷하지만, 키쌍(keypair)형태로 구성이 되어 있다. 딕셔너리는 이름 및 값으로 구성이 되어 있으며, **이름/값** 구분은 **:(콜론)**으로 되어 있다.

```
# variables4.yaml
---
- name: this is first dict
  hosts: localhost
  vars:
    dict:
      firstname: choi
      lastname: gookhyun
  tasks:
  - name: shows your name
    command: echo "{{ dict.fistname }} {{ dict.lastname }}"
```

연습문제

다음과 같이 리스트 및 사전을 생성한다. 호스트는 **`localhost`**로 구성.

1. **리스트 이름은 `company`로 생성.**
    - 리스트 자원은 "삼성", "현대", "기아", "하이닉스", "SKT"구성.
    - 구성된 내용을 'debug'모듈로 화면에 출력.

연습문제

2. 사전 이름은 userinfo로 생성.
    - 사용자 이름(username)에 사용자 값(testuser1) 할당.
    - 사용자 비밀번호(upassword)에 비밀번호 값(testuserpasswd)할당.
    - 사용자 쉘(ushell)에 쉘 값(/bin/zsh) 할당.
    - 사용자 홈 디렉터리(uhome)에 디렉터리 값(/home/testuser1) 할당.
    - 구성된 내용을 'debug'모듈로 화면에 출력.
3. 올바르게 결과값이 출력이 되는지 실행 및 확인 필요.

연습문제

```
---
- name: 리스트와 사전 생성 및 출력
  hosts: localhost
  gather_facts: no
```

연습문제

```
vars:
  # 리스트 변수 생성
  company:
    - 삼성
    - 현대
    - 기아
    - 하이닉스
    - SKT
```

연습문제

```
# 사전 변수 생성
userinfo:
  username: testuser1
  upassword: testuserpasswd
  ushell: /bin/zsh
  uhome: /home/testuser1
```

연습문제

```
tasks:
  - name: company 리스트 출력
    ansible.builtin.debug:
      var: company
  - name: userinfo 사전 출력
    ansible.builtin.debug:
      var: userinfo
```

그룹변수

설명

GROUP_VARS 설명

앤서블 변수에서 제일 많이 사용하는 변수는 `group_vars`라는 변수이다. 이 변수는 보통 그룹 명 기반으로 할당 및 동작한다.

1. **파일**
2. **디렉터리 + 파일**

단순하게 사용 시, 보통 파일 기반으로 구성하지만, 많은 변수 및 변수 선언 분리가 필요한 경우 2번 **디렉터리+파일** 형태를 많이 사용한다.

GROUP_VARS 구성 방식

group_vars는 다음처럼 디렉터리나 혹은 파일을 생성 후 구성이 가능하다.

```
# mkdir group_vars
# mkdir server
# touch server
# cd server
# vim hostname
node1: node1.example.com
```

GROUP_VARS 선언 방식

디렉터리 혹은 파일은 아래와 같은 조건으로 생성을 고려해야 한다.

1. 그룹변수는 아래와 같이 디렉터리나 혹은 파일 형태로 생성할 수 있다.
2. 단순한 변수 선언은 파일, 복잡하고 많은 양의 변수 선언은 디렉터리 기반으로 구성한다.

example_group_vars_dir/

그룹이름으로 디렉터리를 만든 다음에, 임의의 파일 이름으로 변수 선언.

example_group_vars_file

디렉터리를 만들지 않고, 해당 그룹 이름으로 파일을 생성 후, 변수 선언.

GROUP_VARS 선언 예제

아래와 같이 생성한다.

```
# mkdir group_vars_example
# cd group_vars_example
# mkdir group_vars
# vim inventory
[node1]
192.168.100.10 nodename=node1.example.com
# mkdir -p group_vars/node1
```

GROUP_VARS 선언 예제

인벤토리 node1에 할당이 되어 있는 서버는 192.168.100.10이기 때문에 192.168.100.10서버에는 위의 변수를 할당 받는다.

```
# vim all
httpd_package: httpd
ftp_package: vsftpd
```

GROUP_VARS 선언 예제

실행 할 플레이북을 아래와 같이 생성한다.

```
# vi site.yaml
---
- name: Install and configure web/ftp using group_vars
  hosts: node1
  become: true
  vars:
    web_root: /var/www/html
```

GROUP_VARS 선언 예제

실행 할 플레이북을 아래와 같이 생성한다.

```
tasks:
  - name: Install web and ftp packages
    package:
      name:
        - "{{ httpd_package }}"
        - "{{ ftp_package }}"
      state: present
```

GROUP_VARS 선언 예제

실행 할 플레이북을 아래와 같이 생성한다.

```
    - name: Enable & start services
      service:
        name: "{{ item }}"
        state: started
        enabled: true
      loop:
        - "{{ httpd_package }}"
        - "{{ ftp_package }}"
```

GROUP_VARS 선언 예제

실행 할 플레이북을 아래와 같이 생성한다.

```
    - name: Create a simple index.html (uses inventory var 'nodename')
      copy:
        dest: "{{ web_root }}/index.html"
        mode: "0644"
        content: |
          Hello from {{ nodename | default(inventory_hostname) }}
          (served by {{ httpd_package }})
```

GROUP_VARS 선언 예제

실행 할 플레이북을 아래와 같이 생성한다.

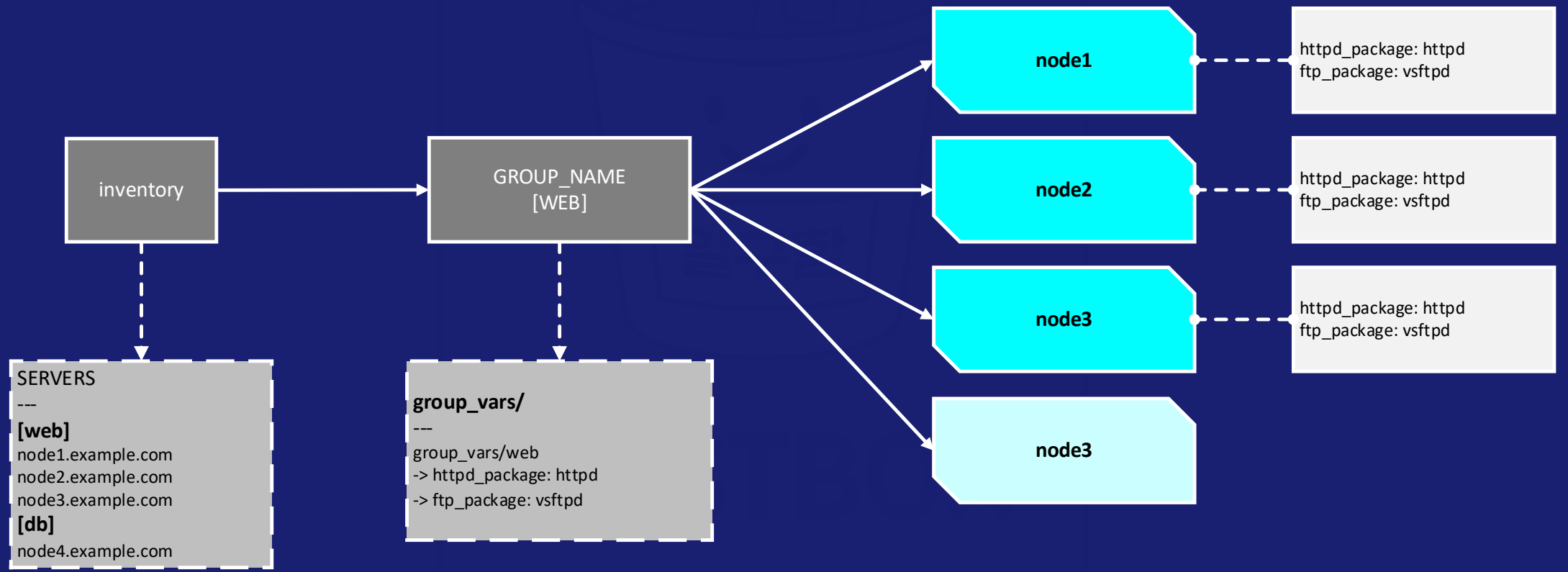
```
    - name: Set FTP banner
      lineinfile:
        path: /etc/vsftpd/vsftpd.conf
        regexp: '^ftpd_banner='
        line: 'ftpd_banner=Welcome to {{ nodename |
default(inventory_hostname) }} FTP'
        create: false
      notify: Restart FTP
```

GROUP_VARS 선언 예제

실행 할 플레이북을 아래와 같이 생성한다.

```
  handlers:
    - name: Restart FTP
      service:
        name: "{{ ftp_package }}"
        state: restarted
# cd group_vars_example
# ansible-playbook -i inventory site.yml
```

GROUP_VARS 할당 방식



호스트 변수

설명

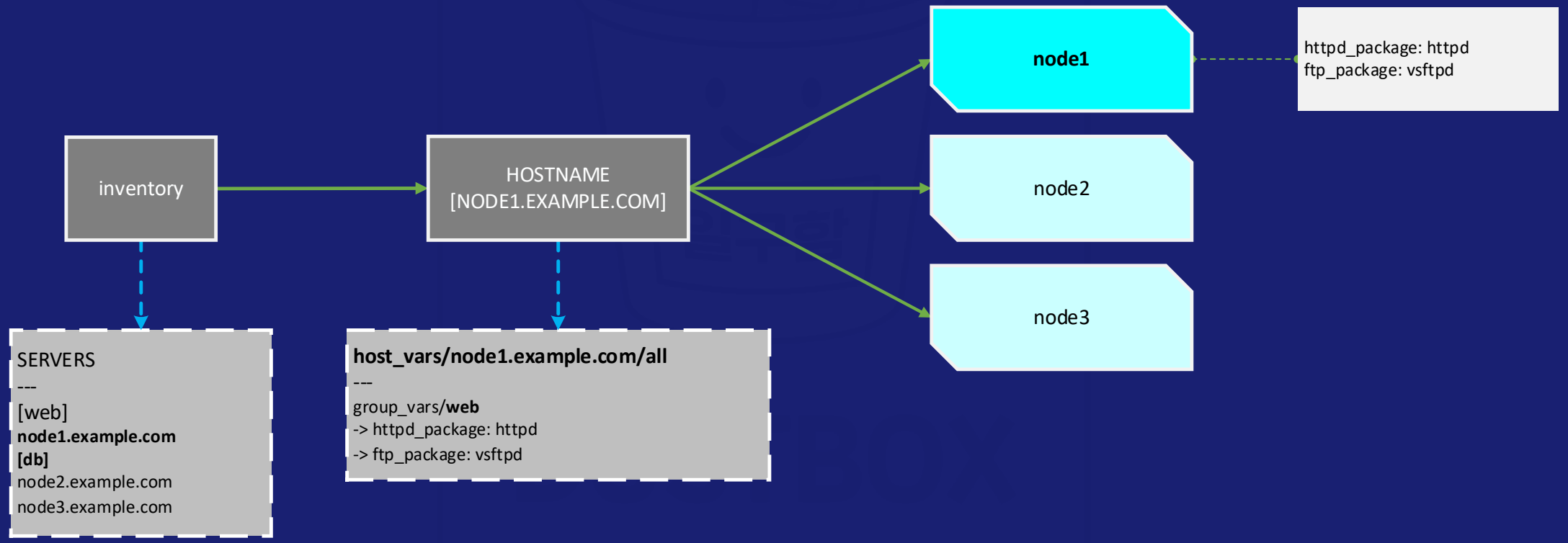
HOST_VARS 설명

호스트 변수도 그룹과 동일하게, 두 가지 형태로 구현 할 수 있다.

1. 파일
2. 디렉터리 + 파일

보통은 2번 **디렉터리 + 파일** 형태를 많이 사용한다. 혹은, 인벤토리를 통해서 특정 호스트에 대해서 변수 선언이 가능하다.

HOST_VARS 할당 방식



HOST_VARS 선언 방식

host_vars는 다음처럼 구성하고 선언한다.

```
# mkdir host_vars
# mkdir node1.example.com
# touch node2.example.com
# cd node1.example.com
# vi node1.example.com/value
nodename: node2.example.com
# vi node2.example.com
nodename: node2.example.com
```

HOST_VARS 선언 방식

192.168.90.13에 서버만 **echoecho**라는 변수에 "This is node1.example.com"이라는 변수를 할당하고 싶으면 다음처럼 작성한다.

```
# mkdir -p inventory/host_vars/192.168.90.13/
# vim inventory/host_vars/192.168.90.13/all
echoecho: "This is node1.example.com"
```

HOST_VARS 선언 예제

아래와 같이 생성한다.

```
# mkdir host_vars_example
# cd host_vars_example
# mkdir host_vars
# vim inventory
[node1]
192.168.100.10 nodename=node1.example.com
# mkdir -p host_vars/node1
```

HOST_VARS 선언 예제

인벤토리 node1에 할당이 되어 있는 서버는 192.168.100.10이기 때문에 192.168.100.10서버에는 위의 변수를 할당 받는다.

```
# vim all
httpd_package: httpd
ftp_package: vsftpd
```

HOST_VARS 선언 예제

실행 할 플레이북을 아래와 같이 생성한다.

```
# vi site.yaml
---
- name: Install and configure web/ftp using group_vars
  hosts: node1
  become: true
  vars:
    web_root: /var/www/html
```

HOST_VARS 선언 예제

실행 할 플레이북을 아래와 같이 생성한다.

```
  tasks:
    - name: Install web and ftp packages
      package:
        name:
          - "{{ httpd_package }}"
          - "{{ ftp_package }}"
        state: present
```

HOST_VARS 선언 예제

실행 할 플레이북을 아래와 같이 생성한다.

```
    - name: Enable & start services
      service:
        name: "{{ item }}"
        state: started
        enabled: true
      loop:
        - "{{ httpd_package }}"
        - "{{ ftp_package }}"
```

HOST_VARS 선언 예제

실행 할 플레이북을 아래와 같이 생성한다.

```
    - name: Create a simple index.html (uses inventory var 'nodename')
      copy:
        dest: "{{ web_root }}/index.html"
        mode: "0644"
        content: |
          Hello from {{ nodename | default(inventory_hostname) }}
          (served by {{ httpd_package }})
```

HOST_VARS 선언 예제

실행 할 플레이북을 아래와 같이 생성한다.

```
    - name: Set FTP banner
      lineinfile:
        path: /etc/vsftpd/vsftpd.conf
        regexp: '^ftpd_banner='
        line: 'ftpd_banner=Welcome to {{ nodename |
default(inventory_hostname) }} FTP'
        create: false
      notify: Restart FTP
```

HOST_VARS 선언 예제

실행 할 플레이북을 아래와 같이 생성한다.

```
  handlers:
    - name: Restart FTP
      service:
        name: "{{ ftp_package }}"
        state: restarted
# cd host_vars_example
# ansible-playbook -i inventory site.yml
```

대화모드 변수

설명

대화모드

변수를 대화형 형식으로 처리가 가능하다. 자주 사용하지는 않지만 아래와 같은 이유로 많이 사용한다.

1. 유연성 (Flexibility) 및 재사용성
2. 보안 (Security) 강화
3. 사용자 확인 및 상호작용 (Confirmation & Interaction)

대화모드

대화형 모드의 시작은 변수 구성으로 시작한다. 하지만, 조금 특이하다. 다른 언어와 다르게 변수 선언 시 인터프리터를 명시를 해야 한다. 대화형 모드 시작 모듈은 `prompt`으로 호출한다. `private`은 사용자가 입력한 내용을 화면에 출력 여부이다.

```
vars_prompt:
  - name: username
    prompt: Put down your name
    private: no

  - name: password
    prompt: Put your password
```

```
https://docs.ansible.com/ansible/latest/playbook_guide/playbooks_prompts.html
```

대화모드

만약, 기본값을 선언한 상태에서 사용하고 싶으면 다음처럼 **기본값**을 설정한다.

```
vars_prompt:
  - name: nic_name
    prompt: put the NIC card name
    default: "ens4"
```

대화모드

입력된 값을 해시 혹은 암호화 하기 위해서 다음처럼 옵션을 구성한다.

```yaml
vars_prompt:
  - name: new_password
    prompt: enter new password
    private: yes
    encrypt: sha512_crypt
    confirm: yes
    salt_size: 10
```

대화모드

특수문자를 입력 받기 위해서는 다음과 같은 옵션을 사용한다.

```yaml
vars_prompt:
  - name: new_password
    prompt: enter new password
    private: yes
    encrypt: sha512_crypt
    confirm: yes
    salt_size: 10
    unsafe: yes
```

비밀번호

혹은 다음과 같이 플레이북을 사용하여 비밀번호 설정이 가능하다.

```
- name: Creating users "{{ uusername }}" without admin access
  user:
    name: "{{ uusername }}"
    password: "{{ upassword | password_hash('sha512') }}"
  when:  assigned_role == "no"
```

적용 예

다음과 같이 활용이 가능하다.

```
---
- name: Create local user with prompted password (sha512_crypt)
  hosts: all
  become: true

  # 슬라이드1: vars_prompt (encrypt, confirm, salt_size, private, unsafe)
  vars_prompt:
    - name: uusername
      prompt: "Enter username to create"
      private: no
```

적용 예

위의 내용 계속 이어서…

```
    - name: assigned_role
      prompt: "Is this an admin user? (yes/no)"
      private: no

    - name: upassword
      prompt: "Enter new password"
      private: yes
      confirm: yes
      encrypt: sha512_crypt
      salt_size: 10
      unsafe: yes
```

적용 예

위의 내용 계속 이어서…

```
  tasks:
    - name: "Creating user {{ uusername }} without admin access"
      user:
        name: "{{ uusername }}"
        password: "{{ upassword }}"
        state: present
        create_home: true
      when: assigned_role | lower == "no"
```

적용 예

위의 내용 계속 이어서…

```
    - name: "Creating admin user {{ uusername }} (wheel/sudo)"
      user:
        name: "{{ uusername }}"
        password: "{{ upassword }}"
        state: present
        create_home: true
        groups: "{{ 'wheel' if ansible_facts['os_family']=='RedHat' else
'sudo' }}"
        append: true
      when: assigned_role | lower == "yes"
# ansible-playbook -i inventory create-user-with-password.yaml
```

암호화 지원 목록

지원하는 암호화 라이브러리는 아래와 같다. 필요에 따라서 선택해서 사용한다.

1. des_crypt - DES Crypt
2. bsdi_crypt - BSDi Crypt
3. bigcrypt - BigCrypt
4. crypt16 - Crypt16
5. md5_crypt - MD5 Crypt
6. bcrypt - BCrypt
7. sha1_crypt - SHA-1 Crypt

암호화 지원 목록

8. sun_md5_crypt - Sun MD5 Crypt
9. sha256_crypt - SHA-256 Crypt
10. sha512_crypt - SHA-512 Crypt
11. apr_md5_crypt - Apache's MD5-Crypt variant
12. phpass - PHPass' Portable Hash
13. pbkdf2_digest - Generic PBKDF2 Hashes
14. cta_pbkdf2_sha1 - Cryptacular's PBKDF2 hash
15. dlitz_pbkdf2_sha1 - Dwayne Litzenberger's PBKDF2 hash
16. scram - SCRAM Hash
17. bsd_nthash - FreeBSD's MCF-compatible nthash encoding

연습문제

앤서블 플레이 북으로 사용자를 생성한다.

1. user1를 대화형으로 생성한다. 비밀번호는 ansible로 한다.
2. user2는 비 대화형으로 생성한다. 비밀번호는 ansible로 한다.
3. 모든 노드(node1/2/3/4/5)에 동일하게 구성한다.

:)

아래와 같이 작성한다.

```
---
- name: Create users on all nodes
  hosts: node1,node2,node3,node4,node5
  become: yes
  # user1은 대화형으로 비밀번호를 입력받음
  vars_prompt:
    - name: user1_password
      prompt: "Enter password for user1"
      private: yes   # 입력 시 화면에 표시되지 않음
  # user2는 비대화형으로 지정
  vars:
    user2_password: "ansible"
```

:)

아래와 같이 작성한다.

```
  tasks:
    # user1 생성
    - name: Create user1 interactively
      ansible.builtin.user:
        name: user1
        password: "{{ user1_password | password_hash('sha512') }}"
        state: present
      tags: user1
```

:)

아래와 같이 작성한다.

```
    # user2 생성
    - name: Create user2 non-interactively
      ansible.builtin.user:
        name: user2
        password: "{{ user2_password | password_hash('sha512') }}"
        state: present
      tags: user2
```

앤서블 시스템 변수

키워드 변수

시스템 변수

시스템 변수의 대다수는 또한, 키워드 명령어와 관련이 있다. 이러한 이유로 어떠한 부분은 앤서블 설정 파일에서 변경이 되며, 혹은 어떠한 변수는 YAML에서 변수 형태로 값이 변경이 된다.

모든 시스템 변수는 별도로 값을 설정하지 않으면, 하드코드 수준에서 기본값이 정해져 있기 때문에, 기본 코드 값으로 **시스템 변수는 동작**하게 된다.

```
https://docs.ansible.com/ansible/latest/reference_appendices/special_variables.html
```

변수 구분

구분	Facts (사실, 대상 시스템 정보)	Magic Variables (실행 환경 정보)	Special Variables (Ansible 구성 정보)
변수 종류	시스템 변수 (System Variables)의 주요 부분	시스템 변수 (System Variables)	시스템 변수 (System Variables)
정보 출처	**대상 호스트** (운영체제, 하드웨어)	**Ansible 엔진 및 인벤토리 구조**	**Ansible 설정 파일 및 실행 경로**
수집/정의 시점	플레이북 실행 초기 `setup` 모듈 실행 시	플레이북 실행 시작 시점에 자동 정의	플레이북 실행 시작 시점에 자동 정의
주요 용도	호스트의 상태 기반 조건문(`when`), 설정 값(`template`) 적용	인벤토리 접근(`hostvars`), 실행 흐름 제어, 디버깅	설정 파일 경로 참조, 역할(Role) 경로 관리
대표적인 예시	`ansible_os_family`, `ansible_default_ipv4.address`, `ansible_hostname`	`inventory_hostname`, `hostvars`, `groups`, `play_hosts`, `ansible_user`	`ansible_config_file`, `role_path`, `playbook_dir`
재정의 가능 여부	**읽기전용**	**가능**	**위치에 따라서 가능**

매직변수(hosts)

`hosts`는 **앤서블 키워드**이며, 싱글 플레이북 혹은 멀티 플레이 북에서 하나 혹은 여러 개 사용이 가능하다. `hosts`에는 하나 이상의 값 할당이 가능하다. 하지만, 이는 **시스템 변수**에 포함이 되지 않는다.

```
- name: this is the first playbook
  hosts: control.example.com

- name: this is the second playbook
  hosts: manage.example.com
```

매직변수(remote_user)

remote_user는 앤서블에서 관리 노드에 접근 시에, 사용하는 계정이다. 명령어로 비교하면 다음과 같다.

```
- name: this is the first playbook
  hosts: remote.example.com
  remote_user: ansible
  become: true
# ssh ansible@remote.example.com
```

매직변수(ansible_connection)

보통 인벤토리에 선언하는 시스템 변수(키워드)이다. 인벤토리에 선언된 이름과 실제로 접속이 되는 서버 이름이 다른 경우 아래와 같이 선언한다. `hostname, IP Address`상관 없다.

```
inventory_file
---
192.168.90.11 ansible_connection=local
```

매직변수(ansible_user)

인벤토리 서버들에 접근 시 보통 사용하는 사용자는 **remote_user**:를 통해서 접근한다. 만약 다른 사용자로 접근을 해야 되는 경우 아래와 같이 선언을 한다. **ansible_user → remote_user**를 대신한다. 동작이 되는 영역은 인벤토리 파일에서만 동작한다.

inventory

```
---
192.168.90.11 ansible_host=node1.example.com ansible_user=testuser
ansible_connection=localhost
internal.prod.db1.example.com ansible_host=node1.example.com
ansible_user=deployer
internal.prod.db2.example.com ansible_host=node2.example.com
ansible_user=oracle
internal.prod.db3.example.com ansible_host=node3.example.com
ansible_user=semanager
```

매직변수(ansible_port)

앤서블 리모트 서버에 접근 시 사용하는 포트 번호가 다른 경우 다음처럼 임시로 포트번호 변경이 가능하다. 앤서블 설정 파일에서는 `remote_port:`라는 옵션을 통해서 변경이 가능하다.

inventory

```
---
192.168.90.11 ansible_port=8080
```

ansible.cfg

```
---
[defaults]
remote_port=8080
```

매직변수(ansible_ssh_pass)

앤서블 서버에 접근 시, ssh 공개키가 아닌 비밀번호로 접근해야 되는 경우 아래와 같은 옵션으로 접근이 가능하다.
비밀번호는 가급적이면 변수, 그리고 **앤서블 볼트**를 통해서 암호화를 권장한다.

```
inventory
---
192.168.90.11 ansible_user=userpass ansible_ssh_pass=centos
ansible_port=8899 ansible_host=node1.example.com
```

확장 팩트

팩트는 사용자가 별도로 시스템에 값을 추가적으로 설정이 가능하다. 예를 들어서 소유권 및 용도에 대한 펙트 추가가 필요한 경우, 아래처럼 펙트를 작성한다.

위의 정보 기반으로 검색 시, 다음과 같은 명령어로 확인이 가능하다.

```
# vi /etc/ansible/facts.d/test.fact
[deployment]
mgmt_name=choigookhyun
mgmt_type=k8s_master
mgmt_division=infra
# ansible <hostname> -m ansible.builtin.setup -a "filter=ansible_local"
```

오래된 키워드 변수

앤서블 사이트에 보면 종종 오랫동안 사용한 키워드 변수 혹은 명령어를 변경 혹은 제거를 한다. 한번씩 꼭 확인한다.

Note

Ansible 2.0 has deprecated the "ssh" from `ansible_ssh_user`, `ansible_ssh_host`, and `ansible_ssh_port` to become `ansible_user`, `ansible_host`, and `ansible_port`. If you are using a version of Ansible prior to 2.0, you should continue using the older style variables (`ansible_ssh_*`). These shorter variables are ignored, without warning, in older versions of Ansible.

ansible_facts 변수

앤서블 **팩트(Fact)**는 앞에서 다루었던, 앤서블 시스템 변수 혹은 사용자 정의 변수이다. 이 내용은 이미 앞서 **"변수"**에서 다루었다.

앤서블 시스템 변수는 앤서블 인벤토리에 등록된 서버에 접근하여 각각 서버의 정보를 setup모듈을 통해서 수집한다. 사용자가 사용자화를 통해서 별도로 설정 및 구성이 가능하다.

앤서블 명령어로 확인이 가능하다.

```
# ansible localhost, -m ansible.builitin.setup
```

확장 팩트를 생성하기 위해서 아래 슬라이드와 같이 생성한다.

ansible_facts 변수

이러한 앤서블 매직 변수는 다음 소스코드에서 확인이 가능하다. 긴 내용이기 때문에 여유 시간이 있을 때, 전체적인 키워드 명령어를 한번 훑어본다.

https://github.com/ansible/ansible/blob/c600ab81ee/lib/ansible/playbook/play_context.py#L46-L55

앤서블에서 제일 활발하게 호출하여 사용하는 변수는 따로 있다. 바로 **ansible_facts**라는 앤서블 내부 변수이다. 이 변수는 **ansible.builtin.setup** 모듈을 통해서 호스트의 정보를 플레이북 시작 전에 수집 후, 앤서블 데이터를 컴파일 한다. 버전에 따라서 다르지만, **ansible.builtin.gather_facts** 모듈을 통해서도 수집이 가능하다.

모든 데이터는 **리스트, 딕셔너리 형태**로 관리 노드에 전달이 되며, 이 정보 기반으로 데이터를 생성 및 컴파일을 진행한다.

ansible_facts

사용하는 방법은 두 가지가 있다.

첫 번째 변수 호출 방법은 JSON형태로 부르며, 두번째 호출 방법은 이전 앤서블 템플릿 변수 호출 방식이다.

두번째 호출 방법은 아래처럼 속성 접근이 가능하다.

```
{{ ansible_facts["eth0"]["ipv4"]["address"] }}
{{ ansible_facts.eth0.ipv4.address }}
```

ansible_facts

변수를 사용하기전에 테스트 및 필요한 도구. 앤서블 네비게이터를 사용하는 경우, 별도로 아래 패키지를 사용하지 않아도 된다. 아래는 `setup` 모듈을 통해서 시스템 정보를 수집 및 확인한다.

특정한 팩트 정보를 수집하기 위해서, 아래와 같이 필터를 걸어서 확인이 가능하다.

```
# ansible localhost, -m setup
# yum install python3-jmespath
# ansible localhost -m setup -a 'filter=ansible_dist*'
```

앤서블 팩트(NIC/DISK)

```
# vi info_hardware.yaml
- hosts: all
  tasks:
  - name: Shows NIC interface name
    debug:
      msg: "{{ ansible_facts.default_ipv4.interface }}"
  - name: Shows NIC IP Address
    debug:
      msg: "{{ ansible_facts.default_ipv4.address }}"
```

앤서블 팩트(NIC/DISK)

```
  - name: Shows Block device size

    debug:

      msg: "{{ ansible_facts.mounts | json_query('[?mount ==
'/'].size_available') }}"
```

ansible_ssh_{{ user, password }}

`SSH Fingerprint Issue`, 이를 해결하기 위해서 **`ansible.cfg`**에 다음처럼 추가한다. 이전에 이 부분을 해결하기 위해서 **`~./ssh/config`**를 통해서 해결 하였다.

위의 설정을 넣어주면 Fingerprint **경고**를 무시하고 진행한다. 보통 오류가 발생하면, 아래와 같이 화면에 출력이 된다.

```
[defaults]
host_key_checking = False

{"msg": "Using a SSH password instead of a key is not possible because Host Key checking is enabled and sshpass does not support this.  Please add this host's fingerprint to your known_hosts file to manage this host."}
```

ansible_ssh_{{ user, password }}

만약, 아이디 및 비밀번호를 가지고 로그인이 필요한 경우, 다음처럼 `ansible.cfg`에 구성이 가능하다.

```
[defaults]
ansible_user = root
ansible_pass = centos
remote_user = root
```

ansible_ssh_{{ user, password }}

ansible_ssh_* 키워드는 사용은 가능하나, 다음 키워드로 대체하여 사용하는 것을 권장한다. 버전이 업그레이드 되면서 많은 키워드 명령어는 제거 혹은 변경이 되기 때문에, 릴리즈 시, 해당 부분을 꼭 확인해야 한다.

```
ansible_ssh_user → ansible_user
ansible_ssh_host → ansible_host
ansible_ssh_port → ansible_port
```

비밀번호 기반으로 접근 설정

ansible.cfg 설정 기반으로 ssh, sudo 접근이 가능하다.

```
[defaults]
become_password_file=sudopass.txt
connection_password_file=sshpass.txt
ask_pass=ansible
remote_user=ansible
remote_port=22
host_key_checking=false
[privilege_escalation]
become_ask_pass=false
become=true
become_user=root
become_method=sudo
```

비밀번호 기반으로 접근 설정

앞에 내용 계속 이어서…

```
[privilege_escalation]
become_ask_pass=false
become=true
become_user=root
become_method=sudo
```

비밀번호 기반으로 접근 설정

비밀번호 파일 기반으로 접근 시, 아래와 같이 구성이 가능하다.

```
# echo "ansible" > sshpass.txt
# echo "ansible" > sudopass.txt
```

EXTRA-VARS

확장변수는 `--extra-vars` 옵션이 있다. 이 옵션을 통해서 기존에 구성된 변수 값 변경이 가능하다. 임시로 아래와 같이 변수를 플레이북에 선언한다.

```
vars:
  config_dir: "/etc/httpd/conf.d/"
  package_name: httpd
```

EXTRA-VARS

아래 명령어 통해서 위의 변수 값에 대해서 오버라이드(override)가 가능하다.

```
# ansible-playbook -e config_dir="/etc/vsftpd/" -e package_name=vsftpd
main.yaml
vars:
  config_dir: "/etc/vsftpd"
  package_name: vsftpd
```

간단한 변수우선 순위

모든 변수에는 우선 순위가 있다. 이를 통해서 기본값 및 동작 방법을 적절하게 구성 및 구현이 가능하다.

1. command line values (for example, -u my_user, these are not variables)
2. inventory file or script group vars
3. role defaults (defined in role/defaults/main.yml)
4. inventory group_vars/all
5. inventory group_vars/*
6. playbook group_vars/*

간단한 변수우선 순위

7. inventory file or script host vars
8. inventory host_vars/*
9. playbook host_vars/*

변수 우선순위에 대한 자세한 정보는 아래의 링크에서 확인이 가능하다.

https://docs.ansible.com/ansible/latest/user_guide/playbooks_variables.html#list-variables

연습문제

앞에서 학습한 내용을 가지고 간단하게 앤서블 기반으로 플레이북 만들어 본다.

1. **ansible adhoc 기반으로 node4에서 node5로 파일을 보낸다.**
    - copy모듈로 hello.txt라는 파일을 생성하여 node2의 /tmp/hello.txt에 저장한다.
    - 메시지 내용은 "Hello World"
    - **ansible inventory 파일 생성한 후 node2에 파일을 보낸다.**
    - inventory에는 호스트 그룹은 web, db로 구성한다.
2. web에는 control.example.com db는 manage.example.com 호스트 이름 사용. 실제 접근 시, node3/4로 접근하게 한다.
    - copy.yaml를 만들어서 /tmp/issue파일을 원격 서버의 /etc/issue에 보낸다.
    - /tmp/issue파일에는 "Hello an ansible remote"라는 메시지가 포함이 되어 있어야 한다.

연습문제

ad-hoc: node1에서 node2로 파일 전송(내용: “Hello World”)

```
# node1(컨트롤 노드)에서 실행
ansible all -i node2, -m copy -a 'content="Hello World" dest=/tmp/hello.txt
mode=0644'
```

연습문제

인벤토리: web/db 그룹과 실제 접속 노드 매핑.

```
[web]
control.example.com ansible_host=node3

[db]
manage.example.com  ansible_host=node4

# (선택) 공통 ssh 사용자/키가 있다면 이렇게 그룹/전역 변수로 추가
#[all:vars]
#ansible_user=ec2-user
#ansible_ssh_private_key_file=~/.ssh/id_rsa
```

연습문제

플레이북: /tmp/issue → 원격 /etc/issue로 배포 (copy.yaml)

```
---
- name: Distribute /tmp/issue to remote /etc/issue
  hosts: web,db
  gather_facts: false
  tasks:
    - name: (컨트롤러) /tmp/issue 파일 생성 - "Hello an ansible remote"
      ansible.builtin.copy:
        dest: /tmp/issue
        content: "Hello an ansible remote\n"
        mode: '0644'
      delegate_to: localhost
      run_once: true
```

연습문제

플레이북: /tmp/issue → 원격 /etc/issue로 배포 (copy.yaml)

```
    - name: (원격) /tmp/issue → /etc/issue 복사
      ansible.builtin.copy:
        src: /tmp/issue
        dest: /etc/issue
        owner: root
        group: root
        mode: '0644'
        backup: true
```

확장 명령어

Structuring Directives(Re-use and Organization)

확장 명령어 유형

확장 명령어는 다음과 같이 정리가 된다.

명령어	레벨	처리 방식	용도 및 특징
`import_playbook`	Playbook	정적	현재 플레이북의 실행 범위를 다른 **전체 플레이북 파일**로 확장. 플레이북 파싱 시점에 컴파일.
`import_tasks`	Task	정적	현재 플레이의 `tasks` 목록을 다른 **외부 작업 목록 파일**로 확장(Ansible 2.4+에서 권장)
`include_tasks`	Task	동적	런타임에 **외부 작업 목록 파일**. `loop`나 `when`과 같은 반복/조건문을 **가져온 작업 전체에 직접 적용 가능**
`include_role`	Task	동적	런타임에 역할을 호출하며, `loop`나 `when`을 사용하여 **조건부로 역할 실행**을 제어가능.

확장 명령어 유형

위의 내용 계속 이어서…

명령어	레벨	처리 방식	용도 및 특징
`include`	Play/Task (구형)	동적	(구버전에서 사용, 현재는 `*_tasks`나 `*_role`로 분리 권장) `playbook`, `tasks`, `vars` 등 다양한 레벨에서 동적 확장 처리
`roles`	Play	정적	플레이 레벨에서 **실행할 역할 호출**(가장 기본적인 역할 호출 방법)
`block`	Task	정적	오류 처리(`rescue`, `always`) 또는 조건부 적용(`when`)을 위해 **여러 작업(tasks)을 그룹화**

import/include

import

이 키워드는 **`tasks`** 이외 **`role, playbook`**와 같은 추가적인 기능들이 더 있다. **`import`** 키워드는 정적으로 동작한다. 즉, 실행 시 모든 내용을 메모리에 불러온 후, 플레이북 작업을 수행한다. 동작 방식은 **정적**으로 동작한다.

import/include

include

`include_` 키워드는 `tasks, role`를 지원하면 `playbook` 영역은 지원하지 않는다.

`include, import`의 다른 부분은 `include`는 해당 지시자를 만나면, 그 순간 파일을 메모리에 불러오며, **동적**으로 동작한다.

아래는 앤서블에서 지원하는 **정적/동적 파일 불러오기** 기능이다. 보통 이를 통해서 파일 기능 분류를 하기도 한다.

import/include

1. import_tasks
2. import_playbook
3. import_role
4. include_tasks
5. include_role

include_playbook은 더 이상 지원하지 않는다.

https://github.com/ansible/ansible/issues/76684

include_tasks

include_ 문법은 위의 **import_**와 기능적으로는 동일하다.

다만, include_는 동적으로 파일을 불러오며, 앤서블 인터프리터가 파싱 하면서 include_ 문법을 만나면, 해당 파일을 메모리에 불러온다.

import/include_tasks 예제

파일 이름은 `ext-include.yaml`으로 작성한다.

```
# vi ext-include.yaml
- name: this play2
  debug:
    msg: "play2"
- name: this play3
  debug:
    sldkjslkjewoie: hehehe
```

include_tasks 예제

위에서 만든 파일을 실험용 플레이북에 적용한다. 문제가 없으면 오류 없이 실행이 된다.

```
- hosts: localhost
  tasks:
    - name: this play1
      debug:
        msg: "play1"
    - name: include
      include_tasks: ext-include.yaml

# ansible-playbook -i hosts includes.yaml --syntanx-check
# ansible-inventory -i hosts --list
# ansible-playbook --syntax-check includes.yaml
```

import_tasks 예제

위에서 만든 내용을 다시 **import_tasks**로 변경한다. 실제로 차이점은 눈으로 확인은 거의 불가능하다.

```
- hosts: localhost
  tasks:
    - name: this play1
      debug:
        msg: "play1"
    - name: include
      import_tasks: ext-include.yaml

# ansible-playbook -i hosts includes.yaml --syntanx-check
# ansible-inventory -i hosts --list
# ansible-playbook --syntax-check includes.yaml
```

import_playbook

플레이북 `tasks:` 영역까지 참조해서 불러온다. 보통 한 개 이상의 플레이북을 시작 시 불러올 때 사용한다.

첫 번째 파일 이름은 `import-playbook1.yaml`으로 작성한다.

```
# vi import-playbook1.yaml
- name: playbook1
  hosts: localhost
  tasks:
  - debug:
      msg: "this is playbook1.yaml file"
```

import_playbook

두 번째 파일 이름은 `import-playbook2.yaml`으로 작성한다.

```
# vi import-playbook2.yaml
- name: playbook2
  hosts: localhost
  tasks:
  - debug:
      msg: "this is playbook2.yaml file"
```

import_playbook

위의 두 개 파일을 import-main.yaml으로 통합 및 작성한다.

```
# vi import-main.yaml
- name: plays all playbooks
  hosts: localhost

- import_playbook: import-playbook1.yaml
- import_playbook: import-playbook2.yaml

# ansible-playbook -i localhost, import-main.yaml
# ansible-playbook --syntax-check import-main.yaml
```

조건문/레지스터

condition

register

when_*

조건문

when/failed_when

assert/superset/tests

when

when이라는 조건문을 사용한다. when, module, roles에서 함께 많이 사용한다. 그 이외 조건에서 사용이 가능하면 언제든지 붙여서 사용이 가능하다.

when명령어는 일반 언어에서 if 명령어와 동일하다. 다만, if 명령어처럼 복잡하게 조건식이 들어가지 않으며, 기본적으로 **반환 값(return count)**가 **참(true)**이면 동작한다.

다만, 특정 상황에서는 when 명령어가 전부 적용되지 않는다. 아래는 쉘 스크립트와 비교한 내용이다.

```
## bash
$ rc=true
if [[ $rc -eq 0 ]] ; then
  echo "run a module"
fi
```

when

앞에 내용 계속 이어서…

```
## ansible
- name: check to the httpd package installed
   shell: rpm -qa httpd
   register: res_httpd_pkg
- name: remove a httpd package
  package:
    name: httpd
    state: absent
  when: res_httpd_pkg
```

when

간단하게 when 명령어를 사용하기 위해서 아래와 같이 간단하게 작성한다.

```
# vi when.yaml
- hosts: node1.example.com
  vars:
    httpd_conf_update: true
  tasks:
  - copy:
      src: httpd.conf
      dest: /etc/httpd/conf/httpd.conf
    when: httpd_conf_update
```

when

```
vars_file:
  - vars.yaml
tasks:
copy:
  src: httpd.conf
  dest: /etc/httpd/conf/httpd.conf
when: httpd_conf_update  == "yes"
```

외부 변수파일을 불러옴

```
$ vi vars.yaml
httpd_conf_update: yes
```

"httpd_conf_update"의 변수가 "yes"로 되어 있으면 참으로 판단하여 수행이 됨.

when+facts

`when` 조건에는 `ansible_facts`와 함께 사용이 가능하다.

먼저, `ansible_facts`에 대해서 잠깐 설명하자면, 앤서블에서 `setup` 모듈이 동작하면서 원격 서버들의 정보를 수집하는데 이를 `facts`라고 부른다.

앞 슬라이드에서 변수 설명하면서 다루었던 부분이다.

when+facts

ansible_facts를 그냥 네이티브로 사용하기도 하는데 보통은 when을 통해서 조건 비교를 많이 한다.

```
when: (ansible_facts['os_family'] == "Debian") or
(ansible_facts['os_family'] == "CentOS")
```

위의 조건을 아래처럼 적용이 가능하다.

```
# vi whenos.yaml
- hosts: localhost
  tasks:
  - debug:
      msg: "{{ ansible_facts['os_family'] }}"
    when: ansible_facts['os_family'] == "Debian" or "CentOS"
# ansible-playbook -i localhost, whenos.yaml
```

when

위에서 잠깐 확인 하였지만, 조건문을 사용할 때 `or, and` 같은 확장 조건문 사용이 가능하다.

```
and: true + true = true
or: true + false = true
```

`os_family`가 `Debian`이나 혹은 `CentOS`라는 문자열을 가지고 있으면 참이다.

when multi-condition

여러 개의 조건에 대해서 확인이 필요한 경우는 다음처럼 조건문을 다중으로 사용이 가능하다. 기존 내용에 업데이트 한다.

```
# vi whenos.yaml
- name: shutdown system
  shutdown:
    msg: "The {{ inventory_hostname }} is going to shutdown by Ansible"
  when:
    - ansible_facts['distribution'] == "CentOS"
    - ansible_facts['distribution']['major_version'] == "8"
# ansible-playbook -i localhost, whenos.yaml
```

when multi-condition

좀 더 많은 조건식 사용하기.

```
# vi whenos.yaml
- hosts: localhost
  tasks:
  - debug:
      msg: "Shutdown OS"
    when:
      - ansible_facts['distribution'] == "CentOS"
      - ansible_distribution_version == "8"
      - ansible_distribution_release == "Stream"
```

when multi-condition

앞에 내용 계속 이어서…

```
 - debug:
      msg: "No Shutdown OS"
    when:
      - ansible_facts['distribution'] ≠ "CentOS"
      - ansible_distribution_version ≠ "8"
      - ansible_distribution_release ≠ "Stream"
# ansible-playbook -i localhost, whenos.yaml
```

failed_when

falied_when은 when과 반대로 실패 시 참인 조건문이다. 사용은 아래와 같이 사용한다.

```
# vi failed_when.yaml
- name: 소프트웨어 설치 (경고 시 실패)
  ansible.builtin.command: echo "INSTALL SUCCESSFUL" ; echo "WARNING: This
version is old" >&2  # Exit Code 0이지만 stderr에 경고를 출력
  register: install_status
  failed_when: "'WARNING:' in install_status.stderr"

- name: 2차 설치 진행
  ansible.builtin.debug:
    msg: "설치에 경고가 없었으므로 다음 단계를 진행합니다."
  when: install_status is succeeded
# ansible-playbook -i localhost, failed_when.yaml
```

changed_when

changed_when은 when과 반대로 변경 시 참인 조건문이다. 사용은 아래와 같이 사용한다.

```
# vi changed_when.yaml
- name: 사용자 생성 및 변경 상태 보고
  ansible.builtin.command: /usr/bin/true
  register: user_creation_result
  changed_when: "'User already exists' not in user_creation_result.stdout"

- name: 디버그 출력
  ansible.builtin.debug:
    msg: "사용자 생성 Task가 시스템을 변경했습니다."
  when: user_creation_result is changed
# ansible-playbook -i localhost, changed_when.yaml
```

버전 조건문 설명

앤서블을 사용하다 보면, 자주는 아니지만, 버전 비교가 종종 필요하다. 앤서블 2.11이후부터 다양 버전 형식에 대한 조건문이 추가가 되었다. 버전 비교 시 사용한다.

보통 사용하는 방법은 아래와 같다.

```
installedVersion is version(softwareVersion, '≤')
```

버전(loose)

loose

loose 버전은 버전 레이블링 형식이 우리가 아는 상식(?)내에서 자유스럽게 보통 표현을 한다. 보통 형식이 없이 작성하는 경우 loose를 사용한다. 버전은 보통 아래와 같은 형식으로 작성한다.

- `1996.07.12`
- `3.2.pl0`
- `3.1.1.6`
- `2g6`

버전(loose)

```
# vi loose.yaml
- hosts: localhost
  vars:
    installedVersion: 1996.07.12
    softwareVersion: 1996.07.12
  tasks:
  - debug:
      msg: "{{ installedVersion is version(softwareVersion, '≤') }}"
# ansible-playbook -i localhost, loose.yaml
```

버전(strict)

strict

보통 3자리로 구별이 된다. 일반적으로 많이 사용하는 릴리즈 방식.

- `V1.1`
- `V3`
- `V2.4.0`

버전(strict)

```
# vi strict.yaml
- hosts: localhost
  vars:
    installedVersion: 1.1
    softwareVersion: 2.4.0
  tasks:
  - debug:
      msg: "{{ installedVersion is version(softwareVersion, '≤') }}"
# ansible-playbook -i localhost, strict.yaml
```

일반 조건문(기호 혹은 문자)

일반 조건문은 다음처럼 지원한다.

```
<, lt, ≤, le, >, gt, ≥, ge, ==, =, eq, ≠, <>, ne
```

위의 조건문을 적용하면 아래처럼 사용이 가능하다.

```
{{ your_version_var is version('4.1.9-rc.1+build.100', operator='lt',
version_type='semver') }}
```

조건문 확인

```
- name: Check if the variable 'admin_password' is defined and not empty
  assert:
    that:
      - admin_password is defined
      - admin_password | length > 0
    # 실패 시 사용자에게 보여줄 메시지 정의
    fail_msg: "ERROR: The 'admin_password' variable is missing or empty."
```

assert는 값이 비어 있는지 확인한다. 비어 있으면 거짓

조건문 모듈

특정 변수에 따라서 메시지 혹은 참 조건을 만들기 위해서 위에서 사용한 **assert**라는 명령어 사용이 가능하다.
that이라는 옵션을 사용하면, **if** 문법에서 조건을 **and** 연산자로 처리하는 것과 비슷하게 동작한다.

```
# vi assert_short.yaml
- name: Ansible Assert 간단 예제
  hosts: localhost
  gather_facts: true # OS 확인을 위해 facts 수집
  vars:
    # 예시 변수
    command_stdout: "result: foo success"
    target_value: 50
```

조건문 모듈

앞에 내용 계속 이어서...

```
  tasks:
    - name: 1. OS가 RedHat이 아닐 때만 실행
      assert:
        that:
          - ansible_os_family ≠ 'RedHat'
        fail_msg: "RedHat 계열 OS에서는 실행 불가"
```

조건문 모듈

앞에 내용 계속 이어서...

```
    - name: 2. 명령어 결과 확인 및 숫자 범위 검증
      assert:
        that:
          - "'foo' in command_stdout"
          - target_value ≥ 0 and target_value ≤ 100
        msg: "모든 조건 충족 (Value: {{ target_value }})"
        fail_msg: "검증 실패: 결과에 'foo'가 없거나 범위(0-100)를 벗어남."
# ansible-playbook -i localhost, assert_short.yml
```


SOOOOO BORED
zzZ

앤서블 tests

앤서블에서 제공하는 **subset** 명령어는 다음처럼 수정 및 변경이 되었다.

```
issubset      → subset
issuperset    → superset
```

subset

말 그대로 변수가 하위에 포함이 되었는지 확인 시 사용하는 명령어. 보통 배열 비교 시 많이 사용한다.

superset

특정 배열의 값이 다른 배열에 포함이 되었는지 확인한다. subset과 다른 부분은 superset의 비교 대상 배열에 모든 값이 포함이 되어 있으면, **"참"**이다.

test(superset/subset)

A영역에 B가 포함이 되어 있는지(혹은 B가 A영역에 포함이 되어 있는지) 검증 및 확인한다.

```
# vi short_superset.yml
- name: 라이브러리 검증
  hosts: localhost
  vars:
    # 시스템에 있어야 하는 최소 필수 목록 (작은 집합)
    required: [ 'numpy', 'pandas' ]
    # 현재 설치된 목록 (큰 집합이어야 함)
    installed: [ 'numpy', 'scipy', 'pandas', 'matplotlib' ]
  tasks:
    - name: 필수 라이브러리 설치 확인
      debug:
        # installed (설치됨)이 required (필수)를 포함(superset)하는지 확인
        msg: "필수 라이브러리 검증 결과: {{ installed is superset(required) }}"
      when: installed is superset(required)
# ansible-playbook -i localhost, short_superset.yml
```

연습문제

ansible_facts['os_family'] 값이 "RedHat"일 때만 "This is RedHat system"을 출력하고, 그 외의 경우 "This is not RedHat"을 출력하는 플레이북을 작성.

```
---
- name: When 조건문 예제
  hosts: localhost
  gather_facts: true

  tasks:
    - name: Print when RedHat
      ansible.builtin.debug:
        msg: "This is RedHat system"
      when: ansible_facts['os_family'] == "RedHat"
```

연습문제

ansible_facts['os_family'] 값이 "RedHat"일 때만 "This is RedHat system"을 출력하고, 그 외의 경우 "This is not RedHat"을 출력하는 플레이북을 작성.

```
- name: Print when not RedHat
    ansible.builtin.debug:
      msg: "This is not RedHat"
    when: ansible_facts['os_family'] ≠ "RedHat"
```

연습문제

ping 명령의 결과에 "1 received" 문자열이 없으면 실패 처리하고, 있으면 "Ping success" 메시지를 출력하는 플레이북을 작성하시오.

```
---
- name: failed_when 예제
  hosts: localhost
  gather_facts: false

  tasks:
    - name: Run ping test
      ansible.builtin.command: ping -c 1 8.8.8.8
      register: ping_result
      failed_when: "'1 received' not in ping_result.stdout"
```

연습문제

ping 명령의 결과에 "1 received" 문자열이 없으면 실패 처리하고, 있으면 "Ping success" 메시지를 출력하는 플레이북을 작성하시오.

```
    - name: Print success
      ansible.builtin.debug:
        msg: "Ping success"
```

연습문제

변수 my_port의 값이 1~65535 범위 내에 있을 때만 통과되도록 하고, 아니면 "Invalid port number"라는 오류를 발생시키시오.

```
---
- name: assert 예제
  hosts: localhost
  gather_facts: false
  vars:
    my_port: 8080
```

연습문제

변수 my_port의 값이 1~65535 범위 내에 있을 때만 통과되도록 하고, 아니면 "Invalid port number"라는 오류를 발생시키시오.

```
tasks:
  - name: Check port range
    ansible.builtin.assert:
      that:
        - my_port ≥ 1
        - my_port ≤ 65535
      fail_msg: "Invalid port number"
      success_msg: "Port {{ my_port }} is valid"
```

레지스터

런타임 변수 관리

레지스터 설명

레지스터는 앤서블 모듈에서 실행된 결과를 메모리에 저장하는 기능이다. 레지스터 명령어는 **register**, 사용법은 다음과 같다.

레지스터는 모듈 실행 후 출력된 표준 출력 및 오류 내용을 JSON형태로 메모리에 저장한다. 사용자가 원하는 내용을 호출 및 출력해서 사용이 가능하다.

일반적인 언어에서는 **echo, print**와 같은 키워드를 앤서블에서는 **debug**모듈을 통해서 출력 및 확인이 가능하다.

```
- <MODULE>
    <ARGS>:
    <ARGS>:
  register: <REGISTER_NAME>
```

레지스터 설명

TASK A:

when: false
register: result

Thinking the var is undefined...

레지스터

간단하게 레지스터를 작성한다.

```
# vi register1.yaml
- hosts: localhost
  tasks:
  - name: list content of directory
    command: ls /tmp
    register: contents
  - name: check contents for emptiness
    debug:
      msg: "Directory is empty"
    when: contents.stdout == ""
# ansible-playbook -i localhost, register1.yaml
```

레지스터

두 번째 레지스터 활용은 다음과 같다.

```
# vi register2.yaml
- hosts: localhost
  vars:
    result: true
  tasks:
  - name: register a variable
    command: /bin/false
    register: result
    ignore_errors: true
```

레지스터

두 번째 레지스터 활용은 다음과 같다.

```
  - name: the result is going to failed
    command: /bin/somedothing
    when: result is failed
# ansible-playbook -i localhost, register2.yaml
```

오류가 발생하여도 작업 수행

레지스터

또한 이와 같은 방식으로 사용이 가능하다.

```
vars:
  result: true
tasks:
- name: when the result is true
  command: /bin/true
  register: result
  when: result is succeeded
- name: when the result is true
  command: /bin/somedothing_else
  register: result
  when: result is skipped
```

레지스터

레지스터 내용을 출력하고 싶은 경우 다음처럼 `debug:` 모듈을 사용하면 된다.

```
- debug:
    msg: "{{ VARIABLE_NAME }}"
- debug:
    var: "{{ VARIABLE_NAME | type_debug }}"
```

https://docs.ansible.com/ansible/latest/collections/ansible/builtin/type_debug_filter.html#examples

레지스터 반환 값

- 성공은 `succeeded`로 표현.
- 실패는 `failed`라고 표현.
- 무시는 `skipped`.

위의 조건들은 `when` 명령어에서 사용이 가능하다.

```
success ← succeeded
failed ← failed
fail ← failed
skip ← skipped
```

Boolean

최근에 앤서블 커뮤니티에서 위의 내용에 대해서 투표를 시작하였다. 결과는 다음과 같다.

1. 결과는 대다수가 "true"를 선호를 하고 있다.
2. "on", "0", "yes"와 같은 표현식 가급적이면 사용하지 않는다.

레지스터에서 값을 다룰 때, 위와 같은 **리턴 변수(return value)**에 대해서 확인을 꼭 해야 한다. 앤서블에서 불린 값을 사용하기 위해서는 다음과 같이 사용한다. 위의 모든 내용은 참을 나타낸다.

```
yes / on / 0 / true
```

Boolean

앤서블에서 **불린 값(Boolean value)** 을 사용하기 위해서는 다음과 같이 사용한다. 위의 모든 내용은 참을 나타낸다. 레지스터는 보통 'true', 'false'나 혹은 '0', '1'으로 반환한다.

```
yes / on / 1 / true

반대는

no / off / 0 / false
```

Boolean

이걸 컨디션으로 사용하기 위해서는 다음처럼 한다.

```
vars:
  test1: true
  test2: yes

- name: Boolean true
  shell: echo "This is true"
  when: test1 or test2 | bool
```

Boolean

혹은 부정형으로 위의 내용을 앤서블에서 적용할 수 있다.

```
vars:
  test1: true
  test2: yes

- name: Boolean false
  shell: echo "This is true"
  when: not test1
```

연습문제

다음과 같은 조건으로 플레이북을 구성한다.

1. 리눅스 배포판이 데비안이면 apt를 통해서 apache2패키지 설치, 레드햇 계열이면 dnf를 통해서 httpd를 통해서 설치한다.
2. 서비스가 올바르게 구성이 되면 웹 서비스를 시작한다. 올바르게 시작하지 않으면 "apache is not well"이라는 메시지를 출력한다.
3. 방화벽에 웹 서비스를 http 및 https 프로토콜을 등록한다.

연습문제

정답 :)

```
# file: web_install_firewall.yml
---
- name: Install Apache and open firewall per distro
  hosts: webservers
  become: true
  gather_facts: true

  vars:
    deb_pkg: apache2
    deb_svc: apache2
    rhel_pkg: httpd
    rhel_svc: httpd
```

연습문제

정답 :)

```
tasks:
   # --- Install (Debian/Ubuntu 계열) ---
   - name: Install Apache on Debian family (apt)
     ansible.builtin.apt:
       name: "{{ deb_pkg }}"
       state: present
       update_cache: yes
     when: ansible_facts.os_family == "Debian"
```

연습문제

정답 :)

```
    # --- Install (RHEL/Rocky/Alma 계열) ---
    - name: Install Apache on RedHat family (dnf)
      ansible.builtin.dnf:
        name: "{{ rhel_pkg }}"
        state: present
      when: ansible_facts.os_family == "RedHat"
```

연습문제

정답 :)

```
  - name: Start web service (Debian)
    block:
      - name: Enable & start apache2
        ansible.builtin.service:
          name: "{{ deb_svc }}"
          enabled: true
          state: started
    rescue:
      - name: Print failure message (Debian)
        ansible.builtin.debug:
          msg: "apache is not well"
    when: ansible_facts.os_family == "Debian"
```

연습문제

정답 :)

```
  - name: Start web service (RedHat)
    block:
      - name: Enable & start httpd
        ansible.builtin.service:
          name: "{{ rhel_svc }}"
          enabled: true
          state: started
    rescue:
      - name: Print failure message (RedHat)
        ansible.builtin.debug:
          msg: "apache is not well"
    when: ansible_facts.os_family == "RedHat"
```

연습문제

정답 :)

```
    # --- Firewall rules ---
    # RHEL: firewalld 서비스 등록
    - name: Open HTTP/HTTPS on firewalld (RHEL family)
      ansible.posix.firewalld:
        service: "{{ item }}"
        permanent: true
        state: enabled
        immediate: true
      loop:
        - http
        - https
      when: ansible_facts.os_family == "RedHat"
```

연습문제

정답 :)

```
    # Debian: UFW로 포트 허용 (UFW가 없으면 설치)
    - name: Ensure ufw installed (Debian family)
      ansible.builtin.apt:
        name: ufw
        state: present
        update_cache: yes
      when: ansible_facts.os_family == "Debian"
```

연습문제

정답 :)

```
    - name: Allow HTTP/HTTPS via ufw (Debian family)
      community.general.ufw:
        rule: allow
        port: "{{ item }}"
        proto: tcp
      loop:
        - "80"
        - "443"
      when: ansible_facts.os_family == "Debian"
```

루프

반복

loop

앤서블 루프 명령어 키워드는 다음과 같다. 다만, 현재는 `loop` 명령어로 통합을 하고 있다. 그렇다고 해서 기존에 사용하던, with_ **명령어** 사용이 불가능한 것은 아니다.

현재 앤서블 커뮤니티는 loop 명령어 기반으로 사용을 권장하고 있으나, 여전히 `with_*`사용은 가능하다.

명령어	설명
`loop`	앤서블에서 기본으로 사용하는 반복문 키워드.
`with_*`	with_* loop와 동일하지만, 기능별로 키워드가 나누어져 있으며, 성능은 loop에 비해서는 낮은 편.
`until`	loop와 함께 사용하는 키워드 조건문.

loop

loop키워드는 앤서블 2.5에서 추가가 되었다.

하지만, 이 명령어는 'with_*'명령어를 전부 대체하지는 못하고 있다. 앤서블 매뉴얼에서는 가급적이면 'loop'명령어를 사용해서 처리한다. 루프의 유일한 단점은, 유틸리티 모듈과 함께 사용해야 효과적으로 활용이 가능하다.

1. We added loop in Ansible 2.5. It is not yet a full replacement for with_<lookup>, but we recommend it for most use cases.
2. We have not deprecated the use of with_<lookup> - that syntax will still be valid for the foreseeable future.
3. We are looking to improve loop syntax - watch this page and the changelog for updates.

loop/with 비교

ㅋ`기존 구문 (with_)	최신 권장 구문 (loop)	설명
`with_list`	`loop:`	가장 단순한 목록(List) 반복작업 명령어
`with_items`	`loop: "{{ items }}"`	`loop`와 `flatten(levels=1)` 필터를 함께 사용하여 대체; (단, 단순 리스트 반복 시에는 `flatten` 없이 `loop`만으로도 대체 가능)
`with_flattened`	`loop: "{{ items flatten }}"`	
`with_sequence`	`loop: "{{ range(start, end+1, stride) | list }}"`	
`with_subelements`	`loop: "{{ users subelements('key') }}"`	
`with_nested`	`loop: "{{ list1| product(list2) }}"`	

loop

with_키워드는 아직은 구형 명령어(deprecated)으로 전환이 되지 않지만, 이는 호환성 유지를 위해서 존재한다.

loop vs with_

https://github.com/ansible/ansible/tree/devel/changelogs

with_items

일반적인 **with_items**명령어. **loop**명령어의 대다수 기능은 **with_items**에서 사용이 가능하다.

```
# with_items.yaml
- hosts: localhost
  tasks:
  - debug:
      msg: "{{ item }}"
    with_items:
      - 1
      - [2,3]
      - 4
- hosts: localhost
  tasks:
  - debug:
      msg: "{{ item }}"
```

with_items

이전에 `with_items:`에서 `dict`를 다음과 같이 선언이 가능했다. 현재는 권장하지 않는다.

```
# vi with_items_dict.yaml
- name: Create users
  user:
    name: "{{ item.name }}"
    state: present
  with_items:
    - { name: alice }
    - { name: bob }
# vi with_items_list.yaml
- name: Touch files
  file:
    path: "/tmp/{{ item }}"
    state: touch
  with_items: [file1, file2, file3]
```

with_fileglob

특정 파일을 루프로 불러오는 경우 다음과 같이 구현 및 구성이 가능하다. 루프에서 위와 동일하게 구현 및 사용하기 위해서 플러그인(flatten)를 사용해야 한다.

```
# vi with_fileglob.yaml
- hosts: localhost
  tasks:
  - debug:
      msg: "{{ item }}"
    with_fileglob: '*.yaml'
- hosts: localhost
  tasks:
  - debug:
      msg: "{{ item }}"
    loop: "{{ lookup('fileglob', '*.txt', wantlist=True) }}"
    loop: "{{ lookup('fileglob', '*.txt', wantlist=True) | flatten }}"
```

with_dict

딕셔너리로 구성된 배열을 다루는 경우, 아래와 같이 구성이 가능하다.

```
# vi with_dict.yaml
- hosts: localhost
  vars:
    dictionary:
      dict1:
        name: keyname1
        value: keyvalue1
      dict2:
        name: keyname2
        value: keyvalue2
```

with_dict/loop

앞에 내용 계속 이어서…

```
  tasks:
  - name: with dict
    ansible.builtin.debug:
      msg: "{{ item.key }} - {{ item.value.name }}"
    with_dict: "{{ dictionary }}"

  - name: loop dict
    ansible.builtin.debug:
      msg: "{{ item.key }} - {{ item.value.name }}"
    loop: "{{ dictionary|dict2items }}"
```

with_sequence

순차적으로 루프를 돌리기 위해서는 아래와 같이 구성이 가능하다.

```
# vi with_sequence.yaml
- hosts: localhost
  tasks:
  - name: with_sequence
    ansible.builtin.debug:
      msg: "{{ item }}"
    with_sequence:
      start=0
      end=4
      stride=2
      format=testuser%02x
```

with_sequence

위의 내용을 `loop:`로 구성이 되면 아래와 같다.

```
  - name: with_sequence → loop
    ansible.builtin.debug:
      msg: "{{ 'testuser%02x' | format(item) }}"
    loop: "{{ range(0, 4 + 1, 2)|list }}"
```

loop

루프는 하나 이상의 값을 반복적으로 모듈에 전달할 때 사용한다.

```
- name: add users
  user:
    name: "{{ item }}"
    state: present
    groups: "wheel"
  loop:
    - user1
    - user2
```

loop with list

혹은 이걸 리스트를 통해서 전달 받을 수 있다.

```
# vi loop1.yaml
- hosts: localhost
  vars:
    userlist:
      - user1
      - user2
    groups: "wheel"
  tasks:
    - debug:
        msg: "{{ item }}"
      loop: "{{ userlist }}"
```

loop with dict

호~~~은~~ 딕셔너리는 다음처럼 핸들링이 가능하다. x 1

```
# vi loop2.yaml
- hosts: localhost
  tasks:
  - name: add users
    user:
      name: "{{ item.name }}"
      state: present
      groups: "{{ item.groups }}"
    loop:
      - { name: 'admuser', groups: 'wheel' }
      - { name: 'normaluser', groups: 'adm' }
```

loop with dict

호~~~은~~ 딕셔너리는 다음처럼 핸들링이 가능하다. x 2

```yaml
- name: add users
  user:
    name: "{{ item.username }}"
    state: present
    groups: "{{ item.groups }}"
  loop: "{{ user_data | dict2items }}"
  vars:
    user_data:
      username: toor
      groups: wheel
```

loop with register

루프는 레지스터와 함께 사용이 가능하다.

```
- name: register with loop
  shell: "echo {{ item }}"
  loop:
    - "one"
    - "two"
  register: echo
- name: dumped the echo register
  debug:
    msg: "{{ echo }}"
```

loop

```
- name: Fail if return code is not 0
  fail:
    msg: "The command ({{ item.cmd }}) is not return 0"
  when: item.rc ≠ 0
  loop: "{{ echo.results }}"
```

loop

```
- name: In loop and looping the result
  file:
    path: "/tmp/{{ item }}"
    state: touch
  loop:
    - one
    - two
```

complex loop

이러한 방식으로 잘 사용하지는 않지만, 다음과 같은 방식으로 리스트 사용하기도 한다. 다만, 두 번째 리스트 `product` 이름으로 리스트를 선언한다.

```
  - name: mysql user access
    mariadb_user:
      name: "{{ item[0] }}"
      priv: "{{ item[1] }}.*:ALL"
      append_privs: yes
      password: "marriadb"
    loop: - "{{ ['user1', 'user2' ] | product(['sample_db1', 'sample_db2',
'sample_db3']) | list }}"
```


SENORGIF.COM

연습문제

loop를 사용하여 다음 내용을 출력한다.

1. apple, pineapple, orange, sugar, star를 화면에 출력한다.
2. name=zenix, company=LF, number=122-3923를 화면에 출력한다.

연습문제

정답 :)

```
---
- name: Loop 예제 - JSON 리스트로 과일 출력
  hosts: localhost
  gather_facts: false

  tasks:
    - name: 과일 리스트 출력
      ansible.builtin.debug:
        msg: "과일 이름: {{ item }}"
      loop: ["apple", "pineapple", "orange", "sugar", "star"]
```

연습문제

정답 :)

```
    - name: 사용자 정보(사전) 출력
      ansible.builtin.debug:
        msg: "name={{ item.name }}, company={{ item.company }},
number={{ item.number }}"
      loop:
        - { name: "zenix", company: "LF", number: "122-3923" }
```

디버깅

런타임 디버깅

디버깅 설명

앤서블은 기본적으로 디버깅 기능이 거의 없다. 에디터 혹은 런타임 프로그램인 **ansible-playbook**에서 메모리 상주 및 컴파일 수행해야 코드 상태 확인이 가능하다.

결론은, 별도의 특별한 디버깅 기능은 **ansible-core**에서는 제공하지 않는다. 보통 많이 사용하는 디버깅 옵션은 아래와 같이 많이 사용한다.

--tags: 특정 플레이만 실행을 원하는 경우, 모듈 작성시 태그를 구성 후 실행한다.

--skip-tags: 특정 플레이 태그만 제외하고 실행을 원하는 경우 이 옵션을 사용한다.

디버깅 코드

디버깅을 위해서 아래와 같이 코드 작성 및 실행이 가능하다.

```
# vi tags.yaml
- hosts: localhost
  tasks:
    - name: tag1
      debug:
        msg: "tag1 is run"
      tags: tag1
    - name: tag2
      debug:
        msg: "tag2 is run"
      tags: tag2
```

디버깅 명령어

실행은 다음과 같이 한다. 여기에서 디버깅을 위한 옵션은 아래와 같다.

`--start-at-task:` 특정 플레이(작업)에서 실행을 원하는 경우 이 옵션을 사용한다.

`--step:` **`--start-at-task`**를 사용하는 경우, 특정 플레이 위치부터 실행하기 때문에, 중간에 멈추고 싶은 경우 이 옵션을 같이 사용한다.

```
# ansible-playbook -i localhost, --tags=tag1 tags.yaml
# ansible-playbook -i localhost, --skip-tags=tag1 tags.yaml
# ansible-playbook -i localhost, --start-at-task=tag1 --step
```

디버깅

혹은 실행 시, 다음과 같이 옵션을 주고 실행 하여도 된다. 문법 확인 및 간접 실행은 다음과 같이 수행이 가능하다.

```
# ansible-playbook -vvv debug.yaml
# ansible-playbook --syntax-check
# ansible-playbook -C
```

디버깅

시스템 관리

블록장치

네트워크

2025-12-26

네트워크

모든 리눅스 배포판은 더 이상, **ifcfg**형태의 파일 형태의 네트워크 설정을 더 이상 지원하지 않는다.

- 레드햇 계열은 버전 8버전 이후부터는 더 이상, **ifcfg-rh**를 지원하지 않는다.
- 데비안(우분투 포함) 및 수세 리눅스 경우에도 더 이상 **ifcfg-suse, ifcfg-deb**를 지원하지 않는다.

현재 대다수 리눅스 배포판은 네트워크를 관리 아래 프로그램 기반으로 관리한다. 보통 기본 값은 **NetworkManager** 혹은 Netplan를 사용한다.

1. **NetworkManager**
2. **systemd-networkd**

앞으로 모든 리눅스는 systemd-networkd로 통합이 될 예정이다.

네트워크 매니저 설치

네트워크 매니저를 구성하기 위해서 아래와 같이 패키지를 구성한다. 다만, 최근 리눅스 배포판은 아래 작업이 별도로 필요하지 않는다. 앤서블에서 다음과 같이 사용이 가능하다.

```
---
- name: 네트워크 매니저 도구 설치 (Red Hat 계열 및 SUSE 계열 대응)
  hosts: node1.example.com
  become: yes
  tasks:
    - name: OS 계열 정보 수집
      ansible.builtin.setup:
        gather_subset:
          - os_family
```

네트워크 매니저 설치

앞에 내용 계속 이어서…

```
    - name: Red Hat 계열 - 패키지 설치
      when: ansible_facts['os_family'] == 'RedHat'
      package:
        name:
          - NetworkManager-libnm
          - "{{ 'libsemanage-python3' if
ansible_facts['distribution_major_version'] is version('8', '≥') else
'libsemanage-python' }}"
          - "{{ 'policycoreutils-python-utils' if
ansible_facts['distribution_major_version'] is version('8', '≥') else
'policycoreutils-python' }}"
        state: present
```

네트워크 매니저 설치

앞에 내용 계속 이어서…

```
    - name: SUSE 계열 - 패키지 설치
      when: ansible_facts['os_family'] == 'Suse'
      package:
        name:
          - libnm0
          - NetworkManager-applet
          - python3-policycoreutils
          - python3-libsemanage
        state: present
```

네트워크 매니저 아이피 설정

아이피 설정은 다음과 같이 가능하다.

```
# vi nm_configure.yaml
 - name: nmcil ipv4 Addr, ipv4 GW and ipv4 DNS
   nmcli:
     type: ethernet
     conn_name: '{{ item.conn_name }}'
     ip4: '{{ item.ip4 }}'
     gw4: '{{ item.gw4 }}'
     dns4: '{{ item.dns }}'
     state: present
   with_items:
     - '{{ nmcli_ethernet }}'
```

systemd-networkd 아이피 설정

systemd 기반으로 네트워크 설정은 다음과 같이 한다. 템플릿 기반으로 아이피 및 네트워크 구성이 되어야 한다.

```
# vi systemd-networkd.jinja2
[Match]
Name="{{ name }}"
[Network]
Address="{{ address }}"
Gateway="{{ gateway }}"
```

systemd-networkd

systemd에 적용할 설정 파일을 생성한다.

```
# vi systemd-networkd.yaml
- hosts: localhost
  vars:
    name: eth0
    address: 10.10.10.1
    gateway: 10.10.10.254
```

systemd-networkd

위의 내용 계속 이어서…

```
  tasks:
  - name: create systemd-networkd
    template:
      src: systemd-networkd.jinja2
      dest: /etc/systemd/network/eth0.network
```

LINUX BRIDGE NETWORK

앤서블에서 리눅스 브릿지를 간단하게 생성 및 구성한다. 다만, 인터페이스 카드는 별도로 없기 때문에 dummy 장치를 통해서 생성 및 구성한다.

```
# vi bridge.yaml
---
- name: Create virtual interface fake-eth0 and bridge fake-br0
  hosts: all
  become: true
  gather_facts: false
  vars:
    dummy_if: "fake-eth0"
    bridge_if: "fake-br0"
    # bridge_ip: "192.168.200.10/24"
```

LINUX BRIDGE NETWORK

위의 내용 계속 이어서…

```
  tasks:
    - name: Ensure 'ip' command is available
      ansible.builtin.command: "which ip"
      register: ip_cmd
      changed_when: false
      failed_when: ip_cmd.rc ≠ 0
      tags: sanity
```

LINUX BRIDGE NETWORK

위의 내용 계속 이어서…

```
  - name: Load dummy kernel module (if available)
    ansible.builtin.command: "modprobe dummy"
    changed_when: false
    failed_when: false
    tags: setup
  - name: Check if dummy interface exists
    ansible.builtin.shell: "ip link show {{ dummy_if }}"
    register: check_dummy
    changed_when: false
    failed_when: false
    tags: interface
```

LINUX BRIDGE NETWORK

위의 내용 계속 이어서…

```
  - name: Create bridge {{ bridge_if }} (if missing)
    ansible.builtin.command: "ip link add name {{ bridge_if }} type bridge"
    when: check_bridge.rc ≠ 0
    register: create_bridge
    tags: bridge
  - name: Ensure dummy interface is down before enslaving
    ansible.builtin.command: "ip link set {{ dummy_if }} down"
    ignore_errors: true
    changed_when: false
    tags: interface
```

LINUX BRIDGE NETWORK

위의 내용 계속 이어서…

```
  - name: Add {{ dummy_if }} to bridge {{ bridge_if }} (if not already member)
    ansible.builtin.shell: |
      # check if already enslaved
      ip -o link show {{ dummy_if }} | awk -F': ' '{print $2}' | grep -q "{{ bridge_if }}" || (
        ip link set {{ dummy_if }} master {{ bridge_if }}
      )
    args:
      executable: /bin/bash
    register: enslave_result
    changed_when: "'master' in enslave_result.stdout or enslave_result.rc == 0 and enslave_result.stdout == ''"
    failed_when: false
    tags: bridge
```

LINUX BRIDGE NETWORK

위의 내용 계속 이어서…

```
- name: Bring up bridge interface {{ bridge_if }}
  ansible.builtin.command: "ip link set {{ bridge_if }} up"
  tags: bridge
- name: Bring up member interface {{ dummy_if }}
  ansible.builtin.command: "ip link set {{ dummy_if }} up"
  tags: interface
- name: (Optional) Assign IP to bridge (uncomment and set var above if needed)
  ansible.builtin.command: "ip addr add {{ bridge_ip }} dev {{ bridge_if }}"
  when: bridge_ip is defined
  ignore_errors: true
  tags: bridge
```

LINUX BRIDGE NETWORK

위의 내용 계속 이어서…

```
# ansible-playbook -i localhost, -c local create-fake-bridge.yml
```

NETPLAN

NETPLAN 모듈도 앤서블에서 지원하고 있다. NETPLAN를 사용하는 경우 다음과 같이 구성이 가능하다.

```
# vi /etc/netplan/ens33.yaml
network:
  version: 2
  renderer: networkd  # 또는 NetworkManager
  ethernets:
    ens33:
      dhcp4: no
      addresses:
        - 192.168.10.100/24
      gateway4: 192.168.10.1
      nameservers:
        addresses: [8.8.8.8, 1.1.1.1]
```

NETPLAN

NETPLAN 모듈도 앤서블에서 지원하고 있다. NETPLAN를 사용하는 경우 다음과 같이 구성이 가능하다.

```
netplan_bridge_playbook/
├── inventory
├── playbook.yaml
└── templates/
    └── netplan-bridge.yaml.j2
```

NETPLAN

NETPLAN은 YAML로 관리 및 구성이 되기 때문에 템플릿으로 생성해야 한다.

```
# vi templates/netplan-bridge.yaml.j2
network:
  version: 2
  renderer: networkd
  ethernets:
    {{ iface_name }}:
      dhcp4: no
```

NETPLAN

NETPLAN은 YAML로 관리 및 구성이 되기 때문에 템플릿으로 생성해야 한다.

```
bridges:
  {{ bridge_name }}:
    interfaces: [{{ iface_name }}]
    addresses: [{{ bridge_ip }}/{{ bridge_cidr }}]
    gateway4: {{ bridge_gw }}
    nameservers:
      addresses: {{ dns_servers }}
```

NETPLAN

플레이북을 아래와 같이 구성 및 생성한다.

```
# vi playbook.yaml
---
- name: Netplan 기반 리눅스 브릿지 구성
  hosts: all
  become: true
  vars:
    iface_name: fake-eth0
    bridge_name: fake-br0
    bridge_ip: 192.168.50.10
    bridge_cidr: 24
    bridge_gw: 192.168.50.1
    dns_servers: [8.8.8.8, 1.1.1.1]
```

NETPLAN

NETPLAN은 YAML로 관리 및 구성이 되기 때문에 템플릿으로 생성해야 한다.

```
tasks:
  - name: 가상 인터페이스 생성 (dummy)
    command: ip link add {{ iface_name }} type dummy
    args:
      warn: false
    ignore_errors: yes
  - name: netplan 구성 파일 생성
    template:
      src: netplan-bridge.yaml.j2
      dest: /etc/netplan/99-bridge.yaml
      owner: root
      group: root
      mode: '0644'
```

NETPLAN

NETPLAN은 YAML로 관리 및 구성이 되기 때문에 템플릿으로 생성해야 한다.

```
- name: netplan 구성 적용
  command: netplan apply

- name: 브릿지 상태 확인
  command: ip a show {{ bridge_name }}
  register: br_state

- name: 결과 출력
  debug:
    msg: "{{ br_state.stdout_lines }}"
```

NETPLAN

실행 및 확인.

```
# ansible-playbook -i inventory playbook.yaml
# ip link show fake-br0
# bridge link
# ip addr show fake-br0
3: fake-br0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 ...
    inet 192.168.50.10/24 brd 192.168.50.255 scope global fake-br0
```

TEAMD

현재 대다수 리눅스 배포판은 LinuxBridge 기반 혹은 Teamd 기반으로 본딩(bonding) 시스템을 구성한다.
Teamd 기반으로 본딩을 구성 시 다음과 같이 구성한다.

```
# vi teamd-setup.yaml
---
- name: Configure teamd interface
  hosts: all
  become: true
  vars:
    team_name: team0
    team_mode: activebackup
```

TEAMD

위의 내용 계속 이어서…

```
    team_ports:
      - eth1
      - eth2
    team_ip: 192.168.10.100
    team_prefix: 24
    team_gateway: 192.168.10.1
```

TEAMD

위의 내용 계속 이어서…

```
  tasks:
    - name: Install teamd package
      ansible.builtin.package:
        name: teamd
        state: present
```

TEAMD

위의 내용 계속 이어서…

```
    - name: Configure team interface
      community.general.nmcli:
        conn_name: "{{ team_name }}"
        ifname: "{{ team_name }}"
        type: team
        ip4: "{{ team_ip }}/{{ team_prefix }}"
        gw4: "{{ team_gateway }}"
        team:
          runner:
            name: "{{ team_mode }}"
        state: up
```

TEAMD

위의 내용 계속 이어서…

```
    - name: Add slave interfaces to team
      community.general.nmcli:
        conn_name: "{{ item }}"
        ifname: "{{ item }}"
        type: ethernet
        master: "{{ team_name }}"
        state: up
      loop: "{{ team_ports }}"
```

TEAMD

실행은 다음과 같이 한다.

```
# ansible-playbook -i inventory.ini teamd-setup.yaml
```

파티션 구성 설명

앤서블 기반으로 파티션은 여러가지 방법이 있다. 보통 다음과 같은 방법을 권장한다.

1. parted
2. sfdisk(shell)

하지만, 멱등성을 위해서는 무조건 1번 앤서블 모듈 기반으로 구성을 권장한다.

LVM2 파티션 구성

LVM2 파티션 구성은 아래처럼 실행이 가능하다.

```
- name: LVM 파티션 생성(1기가)
  parted:
    device: /dev/sdb
    number: 1
    flags: [ lvm ]
    state: present
    part_end: 1GiB
```

일반 파티션 구성

일반 파티션 구성은 아래처럼 실행이 가능하다.

```
- name: 1기가 파티션 생성
  parted:
    device: /dev/sdb
    number: 1
    state: present
    part_end: 1GiB
```

EXT4 파일 시스템 구성

ext4 파일 시스템 구성은 다음과 같이 가능하다.

```
- name: 파일 시스템 생성(ext4)
  community.general.filesystem:
    fstype: ext4
    dev: /dev/sdb1
    opts: -cc
```

XFS 파일 시스템 구성

xfs 파일 시스템 구성은 다음과 같이 가능하다.

```
- name: 파일 시스템 생성(xfs), UUID 생성
  community.general.filesystem:
    fstype: xfs
    dev: /dev/sdb1
    uuid: generate
```

파일 시스템 UUID 재구성

UUID 재구성.

```
- name: 파일 시스템 재생성(ext4), UUID 재구성
  community.general.filesystem:
    fstype: ext4
    dev: /dev/sdb1
    uuid: random
```

마운트 구성(systemd)

systemd 기반으로 구성하는 경우 아래와 같이 구성한다. 먼저, 템플릿 파일을 생성한다.

```
# vi data.mount.j2
[Unit]
Description=Mount point for Data Disk

[Mount]
What=/dev/disk/by-uuid/{{ disk_uuid }}
Where=/mnt/data
Type=ext4
Options=defaults

[Install]
WantedBy=multi-user.target
```

마운트 구성(systemd)

systemd 기반으로 구성하는 경우 아래와 같이 구성한다. **Hosts, vars** 부분은 코드에 제외.

```
# vi systemd-mount.yaml
- name: 1. Copy the custom mount unit file
  ansible.builtin.template:
    src: data.mount.j2
    dest: /etc/systemd/system/mnt-data.mount
    owner: root
    group: root
    mode: '0644'
  notify: daemon-reload
```

마운트 구성(systemd)

앞에 내용 계속 이어서...

```
- name: 2. Enable and start the mount unit for permanent mounting
  ansible.builtin.systemd_service:
    name: mnt-data.mount
    state: started # 즉시 마운트
    enabled: true  # 부팅 시 자동 마운트 (영구 설정)
# ansible-playbook systemd-mount.yaml
```

마운트 구성(일반 방식)

디스크 레이블 기반으로 마운트.

```
# vi general-mount.yaml
- name: Mount up device by label
  ansible.posix.mount:
    path: /srv/disk
    src: LABEL=SOME_LABEL
    fstype: ext4
    state: present
# ansible-playbook general-mount.yaml
```

마운트 구성(일반 방식)

NFS 형식에 부팅은 지원하지 않고 자동 마운트 기능 끄기.

```
- name: Mount NFS volumes with noauto according to boot option
  ansible.posix.mount:
    src: 192.168.1.100:/nfs/ssd/shared_data
    path: /mnt/shared_data
    opts: rw,sync,hard
    boot: false
    state: mounted
    fstype: nfs
```

마운트 구성(일반 방식)

일반적인 NFS 파일 시스템 마운트.

```
- name: Mount an NFS volume
  ansible.posix.mount:
    src: 192.168.1.100:/nfs/ssd/shared_data
    path: /mnt/shared_data
    opts: rw,sync,hard
    state: mounted
    fstype: nfs
```

마운트 구성

디스크 레이블 기반으로 마운트.

```
- name: Mount NFS volumes with noauto according to boot option
  ansible.posix.mount:
    src: 192.168.1.100:/nfs/ssd/shared_data
    path: /mnt/shared_data
    opts: rw,sync,hard
    boot: false
    state: mounted
    fstype: nfs
```

연습문제

두 번째 인터페이스에 보조 NIC 정적 IP를 아래와 같은 조건으로 구성한다.

1. 인터페이스 eth1에 정적 IP 192.168.50.10/24, 게이트웨이 192.168.50.1, DNS 1.1.1.1로 설정한다.
2. 연결 이름은 eth1-static, 부팅 시 자동 연결.
3. 적용 후 연결이 활성 상태여야 한다.

```
- name: Mount NFS volumes with noauto according to boot option
  ansible.posix.mount:
    src: 192.168.1.100:/nfs/ssd/shared_data
    path: /mnt/shared_data
    opts: rw,sync,hard
    boot: false
    state: mounted
    fstype: nfs
```

연습문제

답안 :)

```
---
- name: Configure eth1 with static IP via NetworkManager
  hosts: all
  become: true
  vars:
    ifname: eth1
    con_name: eth1-static
    ip4: 192.168.50.10/24
    gw4: 192.168.50.1
    dns1: 1.1.1.1
```

연습문제

답안 :)

```
tasks:
  - name: Ensure static connection exists/updated
    community.general.nmcli:
      conn_name: "{{ con_name }}"
      ifname: "{{ ifname }}"
      type: ethernet
      autoconnect: yes
      ip4: "{{ ip4 }}"
      gw4: "{{ gw4 }}"
      dns4:
        - "{{ dns1 }}"
      state: present
```

연습문제

답안 :)

```
    - name: Bring up the connection
      community.general.nmcli:
        conn_name: "{{ con_name }}"
        state: up
```

예외처리

오류

핸들러

예외처리

앤서블에서는 예외 처리 조건이 은근히 다양하게 있다. 앤서블 자주 사용하는 오류 핸들링 명령어 및 사용방법에 대해서 학습한다.

보통 많이 사용하는 예외처리는 아래와 같다.

1. `until`
2. `block/resuce/always`
3. `failed_*`
4. `ignore_*`
5. `run_once`

블록

BLOCK과 동료들?

until

만약 특정 모듈에서 특정조건을 충족할 때까지 조건 대기를 하기 위해서는 **until**이라는 추가 조건을 사용한다.

```
- shell: /bin/noexistcmd
  register: result
  until: result.stdout.find("I am okay") != -1
  retries: 2
  delay: 5
```

위의 조건은 **"I am okay"**라는 문자열이 레지스터 **result**에 있으면 참이 되기 때문에 != -1 성립이 된다. 하지만 위의 명령어가 없기 때문에 올바르지 않는 값을 반환한다.

최대 2번 그리고 **5초 동안 반복**하게 된다.

블록 구성(block)

앤서블에서 많이 사용하는 블록 핸들링 오류 처리이다. 블록 자체만으로 사용이 가능하지만, 블록은 같이 사용하는 내부 키워드가 별도로 존재한다.

```
block
rescue
always
+
any_errors_fatal
```

블록(block)

키워드	구분	역할	실행 시점
`block`	오류 그룹	오류를 감지하고 처리할 태스크 그룹 정의	일반적인 태스크 실행 순서
`rescue`	오류 처리	`block` 내의 태스크에서 **오류(fatal error)**가 발생했을 때 실행될 복구(Recovery) 태스크 그룹 정의.	`block` 내에서 오류 발생 **직후**에 실행
`always`	최종 정리	`block` 또는 `rescue` 태스크의 **성공/실패 여부와 관계없이** 항상 실행될 태스크 그룹을 정의.	`block`과 `rescue` 실행 완료 **후**에 실행
`any_errors_fatal`	오류 정책	Play 수준 설정. 기본적으로 Ansible은 Playbook을 실행할 때 하나의 호스트에서 오류가 발생해도 다른 호스트에서는 계속 진행. 이 설정을 `true`로 지정하면, **단 하나의 호스트라도 오류가 발생하면 전체 Play 실행을 즉시 중단**시키고 치명적인(fatal) 오류로 간주.	오류 발생 **즉시** 적용되어 Play 전체의 실행 흐름을 제어

블록의 기능

여러 모듈을 마치 하나의 클래스처럼 처리한다.

```
block:
  - yum:
    name: "{{ item }}"
    state: present
    loop:
    - httpd
    - vsftpd
  - dnf:
    name: vsftpd
    state: absent
```

블록(block)

앤서블에서 많이 사용하는 블록 핸들링 오류 처리이다. 여러 명령어를 마치 네임스페이스처럼 구성해서 사용이 필요한 경우 아래 블록처럼 구성한다.

```
- name: start the block session
  block:
    - yum:
        name: httpd
        state: present
      register: result_package
```

블록(rescue)

block부분에서 오류가 발생하면 rescue바로 설정된 작업을 차례대로 수행한다. 보통은 block에서 중지된 작업에 대해서 rollback작업을 수행한다.

```
- name: start the block session
  block:
    - yum:
        name: httpd
        state: present
      register: result_package
  rescue:
  - yum:
      name: httpd
      state: absent
```

블록(always)

`always:`는 필수(mandatory)키워드는 아니다. `block:, rescue:` 수행이 되면 무조건 `always:` 명시된 작업들은 수행이 된다. 코드 구성은 아래와 같이 된다.

```
- name: start the block session
  block:
    - yum:
        name: httpd
        state: present
      register: result_package
  rescue:
    - yum:
        name: httpd
        state: absent
  always:
    - debug:
        msg: "Always runs it"
```

블록(always)

혹은 다음과 같이 활용이 가능하다.

```
always:
  - debug:
      msg: "the httpd package installed"
    when: result_package.rc
  - debug:
      msg: "the httpd package removed"
    failed_when: result_package.rc ≠ 1
```

블록(any_errors_fatal)

오류가 발생하면, 현재 배치(작업순서)에 걸려있는 모든 호스트에 동일하게 오류 작업을 수행 후, 종료한다. 오류가 발생하면 해당 작업은 중지가 된다.

```
tasks:
  - block:
    - include_tasks: mytasks.yaml
    - debug:
        msg: "It's error"
      any_errors_fatal: true
```

예외처리 키워드

이외 예외처리사 사용하는 키워드 명령어.

1. `ignore_unreachable`
2. `ignore_errors`
3. `failed_when`
4. `changed_when`

ignore_unreachable

서버는 동작하나, 네트워크나 혹은 관리 서버의 워크로드 인하여 접근이 되지 않는 경우 `ignore_unreachable` 라는 키워드를 통해서 접근 오류가 발생하면 무시하고 정상 수행이 가능하다.

```
- hosts: server_ok, server_error
  tasks:
  - command: /bin/true
    ignore_unreachable: true
  - command: echo "동작 했습니다."
```

ignore_errors

종류에 상관 없이 모듈에 오류가 발생하면 중지하지 않고 다음 작업으로 진행한다.

```
- hosts: node1, node2
  tasks:
  - command: /bin/noexist_cmd
    ignore_errors: yes
  - command: echo "동작 했습니다."
```

Rejecting modules

특정 모듈을 사용을 하지 않기 위해서 avoid(무시)목록을 만들 수 있다. `/etc/ansible.cfg`의 `[defaults]`에서 `plugin_filter_cfg`에 리스트 파일을 명시한다.

```
[defaults]
plugin_filter_cfg = plugin_filters.yaml
```

Rejecting modules

```
# vi plugin_filters.yaml
---
filter_version: '1.0'
module_rejectlist:
  - docker
  - easy_install
```

run_once

특정 모듈만 한번 실행하는 명령어. 보통 쉘 같은 스크립트에 적용한다. 반복적으로 계속 실행되는 부분을 방지한다.
예를 들어서 메일 발송을 한번만 해야 되는 경우, 메일 발송 모듈에 다음과 같이 적용이 가능하다.

```
- hosts: node2.example.com
  vars:
    mail_recipient: root@example.com
    body: 'Hacking So hot'
  tasks:
    - name: Send summary mail
      local_action:
        module: community.general.mail
        subject: "해킹 당했습니다."
        to: "{{ mail_recipient }}"
        body: "{{ mail_body }}"
      run_once: true
```

연습문제

http://127.0.0.1:8080/health 엔드포인트가 200을 반환할 때까지 최대 5회, 3초 간격으로 재시도하라. 실패 시 플레이를 실패 처리하라.

```
- hosts: localhost
  gather_facts: false
  tasks:
    - name: Wait until /health returns 200
      uri:
        url: http://127.0.0.1:8080/health
        return_content: false
      register: health
      until: health.status == 200
      retries: 5
      delay: 3
      failed_when: health.status ≠ 200
```

연습문제

until + 실패 무시.

```
- hosts: localhost
  gather_facts: false
  tasks:
    - name: Try health check but do not fail the play
      uri:
        url: http://127.0.0.1:8080/health
      register: health
      until: health.status == 200
      retries: 5
      delay: 3
      failed_when: false
      ignore_errors: true
```

연습문제

htop 설치가 실패하면 대체 리포지토리를 추가하고 다시 설치해라. 마지막엔 성공/실패와 무관하게 “정리 작업” 메시지를 남겨라.

```
- hosts: localhost
  become: true
  gather_facts: false
  tasks:
    - block:
        - name: Install htop
          package:
            name: htop
            state: present
```

연습문제

htop 설치가 실패하면 대체 리포지토리를 추가하고 다시 설치해라. 마지막엔 성공/실패와 무관하게 “정리 작업” 메시지를 남겨라.

```
    rescue:
      - name: Add fallback repo
        copy:
          dest: /etc/yum.repos.d/fallback.repo
          content: |
            [fallback]
            name=Fallback
            baseurl=http://mirror.example.com/fallback/$basearch
            enabled=1
            gpgcheck=0
```

연습문제

htop 설치가 실패하면 대체 레포지토리 추가하고 다시 설치해라. 마지막엔 성공/실패와 무관하게 “정리 작업” 메시지를 남겨라.

```
    - name: Retry install htop
        package:
          name: htop
          state: present
      always:
        - name: Cleanup message
          debug:
            msg: "정리 작업 완료(성패와 무관)."
```

연습문제

/usr/bin/mycheck가 0으로 종료되지만 출력에 "DEGRADED"가 들어가면 태스크 실패로 간주하라.

```
- hosts: localhost
  gather_facts: false
  tasks:
    - name: Run check command
      command: /usr/bin/mycheck
      register: out
      changed_when: false
      failed_when: "'DEGRADED' in out.stdout"
```

연습문제

df -h / 결과에서 사용률이 90% 이상이거나, 명령이 비정상 종료(rc != 0)이면 실패 처리하라.

```
- hosts: localhost
  gather_facts: false
  tasks:
    - name: Check root fs usage
      shell: "df -P / | awk 'NR==2 {print $5}' | tr -d %"
      register: disk
      changed_when: false
      failed_when: disk.rc ≠ 0 or (disk.stdout|int) ≥ 90
```

핸들러

HANDLER

handler, notify

핸들러(handler)는 특정한 이벤트가 발생하면 "goto"명령어처럼 특정 블록으로 넘어가서 작업 수행 후 다시 본래 작업 위치로 돌아 온다. 문법은 다음처럼 작성한다.

handler, notify

```
- name: install a httpd package
  yum:
    name: httpd
    state: latesed
- name: enabled and start the httpd service
  service:
    name: httpd
    state: started
```

handler, notify

```
- name: a copy the index.html file into /var/www/html
  copy:
    src: index.html
    dest: /var/www/html/index.html
  notify:
    - restart httpd srv
- handlers:
  - name: restart httpd srv
    service:
      name: httpd
      state: restarted
```

handler, notify 설명

```
- name: a copy the index.html file into /var/www/html
  copy:
    src: index.html
    dest: /var/www/html/index.html
  notify:
    - restart httpd srv

- handlers:
  - name: restart httpd srv
    service:
      name: httpd
      state: restarted
```

작업이 문제 없이 수행이 되면 "notify"에 명시된 이름을 호출한다. 작업은 handlers에 명시가 되어있다.

"notify"에 명시된 이름은 "restart httpd srv" 해당 이름에 명시된 모듈을 실행한다.

연습문제

앤서블 플레이북 핸들러 기반으로 아래와 같이 작업 구성 및 핸들러를 통하여 아래 작업이 수행이 되어야 한다.

1. vsftpd서비스를 구성 후, 서비스를 시작한다.
2. 올바르게 시작되지 않는 경우 "can't start to vsftpd"라는 메시지를 출력한다.
3. 올바르게 설치가 된 경우에는, 서비스 시작 및 "start vsftpd service"를 출력한다.

연습문제

```
# vi vsftpd.yml
---
- name: Install & manage vsftpd with handler messages
  hosts: ftp_servers
  become: true
  vars:
    vsftpd_pkg: vsftpd
    vsftpd_service: vsftpd
```

연습문제

```
tasks:
  - name: Ensure vsftpd is installed
    ansible.builtin.package:
      name: "{{ vsftpd_pkg }}"
      state: present
    notify: "start vsftpd via handler"
  - name: Flush handlers so service starts in this run
    ansible.builtin.meta: flush_handlers
```

연습문제

```
handlers:
  - name: start vsftpd via handler
    listen: "start vsftpd via handler"
    block:
      - name: Enable & start vsftpd service
        ansible.builtin.service:
          name: "{{ vsftpd_service }}"
          enabled: true
          state: started
        register: vsftpd_start
```

연습문제

```
  - name: Print success message
      ansible.builtin.debug:
        msg: "start vsftpd service"
    rescue:
      - name: Print failure message
        ansible.builtin.debug:
          msg: "can't start to vsftpd"
      - name: Mark play as failed when vsftpd cannot start
        ansible.builtin.fail:
          msg: "can't start to vsftpd"
```

마지막 슬라이드; _roles

역할

역할설명

역할(Role)은 앤서블에서 **코드 재사용성과 관리 효율성을 극대화하는 기능**이다. 여러 개의 역할이 모여 하나의 플레이북을 구성하며, 각 역할은 특정 기능 단위를 담당한다.

역할을 설계할 때는 다음 사항을 고려해야 한다.

1. 기능적으로 중복되는 역할이 없는가?
2. 각 역할이 독립적으로 동작할 수 있는가?
3. 멱등성을 해치는 모듈을 사용하지 않았는가?
4. 변수화를 통해 재사용이 용이한가?
5. 역할 간 의존성이 명확하게 정의되어 있는가 (meta/main.yml)
6. 표준 역할 디렉터리 구조를 유지하고 있는가
7. 역할 단위 테스트 및 검증이 가능한가 (--check, molecule test)

생성할 역할 디렉터리

roles에 제공 혹은 필요한 디렉터리는 다음과 같다. 아래 구성대로 마지막 역할 랩을 진행한다. 모든 디렉터리에는 최고 한 개의 **main.yml** 혹은 **main.yaml** 파일이 존재해야 한다.

```
roles/httpd/
    tasks/: 기본 디렉터리. 이 디렉터리가 없으면 올바르게 tasks를 불러오지 못한다.
    files/: copy같은 명령어로 복사 시 사용하는 디렉터리.
    templates/: template모듈로 템플릿 기반으로 파일 생성시 사용.
    vars/: 변수 디렉터리
    defaults/: 기본 값을 정의하는 디렉터리
    meta/: 메타에는 roles의 설명 그리고 의존성 같은 부분에 대해서 구성.
```

생성할 역할 디렉터리

기본적으로 동작에 필요한 디렉터리는 tasks/만 있어도 된다. 나머지 특정 디렉터리에 대해서는 앤서블 사이트를 참조하여 필요한 경우 그때그때 참조해서 만들면 된다.

다만, 앤서블 갤럭시를 통해서 배포하는 경우 반드시 "meta/"에 패키지 정보를 꼭 작성해주어야 한다.

역할 구성 준비

roles는 기본적으로 디렉터리 기반의 프레임워크를 가지고 있다.디렉터리 생성 방법은 두 가지 방법이 있다. 아래는 간단하게 **역할**에서 사용하는 **네임 스페이스 디렉터리**를 생성하는 방법이다.

```
# ansible-galaxy init roles/httpd

# mkdir -p roles/httpd/tasks
# touch roles/httpd/tasks/main.yaml
# mkdir -p roles/httpd/tasks/
```

테스크 구성

```
# vi roles/httpd/tasks/main.yaml
- name: install a httpd package
  yum:
    name: httpd
    state: latest
- name: template a httpd configuration file
  template:
    src: httpd.confi.j2
    dest: /etc/httpd/conf/httpd.conf
```

템플릿 구성

역할에서 템플릿을 배포 및 사용하는 경우, 아래와 같이 템플릿 디렉터리를 생성한다. 모든 자원에 대해서는 각각 디렉터리를 가지고 있다. 나머지 부분은 httpd.conf.j2파일을 직접 열어서 확인한다.

```
# mkdir -p roles/httpd/templates/
# touch roles/httpd/templates/httpd.conf.j2
# grep -Ev "^#|^$|*#" /etc/httpd/conf/httpd.conf > touch
roles/httpd/templates/httpd.conf.j2
```

파일 구성

플레이북 동작 시, 배포가 필요한 고정적인 내용의 설정파일 혹은 문서가 있는 경우, 파일을 통해서 배포가 가능하다.

```
# mkdir -p roles/httpd/files/
# touch roles/httpd/files/README.md
# cat <<EOF> roles/httpd/files/README.md
Hello This is New HTTPD Server!
EOF
```

변수 구성

변수는 roles에 사용하는 변수를 선언한다. 이 변수는 두 가지 방법으로 제공한다.

1. **defaults/**

2. **vars/**

역할에서는 **defaults/**에서 먼저 고정 변수를 읽어오고, 그 다음으로 **vars/**에서 유동 변수를 읽어온다. 서로 변수를 덮어쓰기가 가능하기 때문에, 적절하게 분리 및 운영이 가능하도록 한다.

```
# mkdir -p roles/httpd/defaults/
# mkdir -p roles/httpd/vars/
# touch roles/httpd/defaults/main.yaml
# touch roles/httpd/vars/main.yaml
# vi roles/httpd/defaults/main.yaml
hostname: node5.example.com
# vi roles/httpd/vars/main.yaml
deployment: webserver
```

변수 구성

일반 변수 디렉터리는 역할에서 다음과 같이 구성한다.

```
# mkdir roles/hostname/vars
# vim roles/hostname/vars/main.yaml
node1: node1.example.com
```

기본 변수 값 디렉터리는 역할에서 다음과 같이 구성한다.

```
# mkdir roles/hostname/defaults
# vim roles/hostname/defaults/main.yaml
node1: nodex.example.com
```

include/import_role

역할을 이전에 학습하였던 include_roles, import_roles로 확장 관리가 가능하다. 아래는 두 가지 방식으로 역할 디렉터리를 불러올 수 있다.

```
# vi deploy-web.yaml
- name: include roles from role directory
  include_role:
    name: testRole
    tasks_from: verify_httpd_srv
- name: include roles from role directory
- include_role: "{ name: role1, tasks_from: verify_httpd_srv }"
- include_role: "{ name: role2 }"
- include_role: "{ name: role3}"
```

include/import_role

```
- name: include roles from role directory
  include_role:
    name: testRole
    tasks_from: verify_httpd_srv
```

roles/디렉터리에서 testRole/라는 이름을 가지고 있는 디렉터리를 불러온다.
ex: roles/testRole/tasks/main.yml

tasks디렉터리에 있는 "verify_httpd_srv.yaml"파일을 불러온다
ex: roles/testRole/tasks/verify_httpd_srv.yml

VAR IN ROLE

role에서 변수 작성 및 구성은 다음과 형태로 한다.

1. `roles/<ROLE NAME>/vars`
2. `roles/<ROLE NAME>/defaults`
3. `main.yaml` 생성
4. 변수 값 선언

role에서 `vars/` 혹은 `defaults/` 밑에 반드시 최소 하나의 `main.yaml`이 존재해야 한다.

ROLE with VAR

플레이 북에서 "role"를 호출하여 사용 시, 일시적으로 기존 변수 값을 "override"가 필요한 경우가 있다. 기존의 변수 값은 변경을 원하는 경우, 아래처럼 YAML, 인라인 형태로 사용이 가능하다.

```
# vim var_roles/hostname.yaml
- hosts: localhost
  roles:
  - { role: hostname, node1: node1-1.lab.example.com }
# vim var_roles/hostname.yaml
- hosts: localhost
  roles:
  - role: hostname
    vars:
       node1: 'node1-1.lab.example.com'
```

role default vs vars

role에서 사용하는 **defaults**와 **vars**에 대한 오해가 있어서 정리가 필요하다. 정리된 내용은 아래 링크에서 확인이 가능하며, **default, vars**에 대한 정리는 다음과 같다.

https://docs.ansible.com/ansible/latest/user_guide/playbooks_variables.html#variable-precedence-where-should-i-put-a-variable

VARS 디렉터리

In general, Ansible gives precedence to variables that were defined more recently, more actively, and with more explicit scope. Variables in the defaults folder inside a role are easily overridden. Anything in the vars directory of the role overrides previous versions of that variable in the namespace. Host and/or inventory variables override role defaults, but **explicit includes such as the vars directory or an include_vars task** override inventory variables.

ROLE 랩 시나리오

ROLES 활용 및 학습을 위해서 다음과 같은 시나리오 기반으로 MariaDB, HAProxy roles를 구성 및 생성한다.

1. MariaDB: 실시간 동기화 복제
2. HAProxy: **agent-check**로 “up”인 노드만 **쓰기 엔드포인트**로 사용 (쓰기 안전)

디렉터리 구조

ROLE 랩 디렉터리 구조는 다음과 같다.

```
galera-ha-simple/
├── inventory.ini
├── site.yml
├── group_vars/
│   └── all.yml
├── roles/
│   ├── galera_simple/
│   │   ├── tasks/main.yml
│   │   ├── templates/99-galera.cnf.j2
│   │   └── handlers/main.yml
```

디렉터리 구조

위의 내용 계속 이어서...

```
galera-ha-simple/
|    └── haproxy_simple/
|        ├── tasks/main.yml
|        ├── templates/haproxy.cfg.j2
|        └── handlers/main.yml
```

ROLE 랩 인벤토리

다음과 같은 내용으로 인벤토리를 구성 및 생성한다.

```
[haproxy]
haproxy1 ansible_host=192.168.100.10

[galera]
db1 ansible_host=192.168.100.13 server_id=1
db2 ansible_host=192.168.100.14 server_id=2
db3 ansible_host=192.168.100.15 server_id=3
```

메인 플레이북 구성

메인 플레이북은 다음과 같이 구성한다.

```
# vi site.yml
- hosts: galera
  become: yes
  roles: [galera_simple]

- hosts: haproxy
  become: yes
  roles: [haproxy_simple]
```

ROLE 랩 변수 구성

roles를 구성하면 다음과 같이 디렉터리가 구성이 된다.

```
# group_vars/all.yml
mariadb_root_password: "ChangeMe-Root!"
galera_cluster_name: "my-galera"
galera_nodes: "{{ groups['galera'] | map('extract', hostvars,
'ansible_host') | list }}"
mariadb_port: 3306
```

ROLE 랩 GALERA 템플릿

roles를 구성하면 다음과 같이 디렉터리가 구성이 된다.

```
# vi roles/galera_simple/templates/99-galera.cnf.j2
[mysqld]
bind-address = 0.0.0.0
port = {{ mariadb_port }}
server_id = {{ hostvars[inventory_hostname].server_id }}
binlog_format = ROW
default_storage_engine = InnoDB
innodb_autoinc_lock_mode = 2
```

ROLE 랩 GALERA 템플릿

roles를 구성하면 다음과 같이 디렉터리가 구성이 된다.

```
wsrep_on = ON
wsrep_provider = /usr/lib64/galera/libgalera_smm.so
wsrep_cluster_name = {{ galera_cluster_name }}
wsrep_cluster_address = "gcomm://{{ galera_nodes | join(',') }}"
wsrep_node_address = {{ hostvars[inventory_hostname].ansible_host }}
wsrep_node_name = {{ inventory_hostname }}

wsrep_sst_method = rsync
```

ROLE 랩 GALERA/MARIADB 설치

데이터 베이스/갈래라 설치를 진행한다.

```
# vi roles/galera_simple/tasks/main.yml
- name: Install MariaDB Galera
  package:
    name:
      - MariaDB-server
      - MariaDB-client
      - galera-4
      - python3-PyMySQL
      - rsync
    state: present
```

ROLE 랩 GALERA/MARIADB 설치

데이터 베이스/갈래라 설치를 진행한다.

```
- name: Drop minimal galera config
  template:
    src: 99-galera.cnf.j2
    dest: /etc/my.cnf.d/99-galera.cnf
    mode: '0644'
  notify: Restart MariaDB

- name: Enable MariaDB service (stopped for bootstrap)
  service:
    name: mariadb
    enabled: yes
    state: stopped
```

ROLE 랩 GALERA/MARIADB 설치

데이터 베이스/갈래라 설치를 진행한다.

```
# vi roles/galera_simple/handlers/main.yml
- name: Restart MariaDB
  service:
    name: mariadb
    state: restarted
```

ROLE 랩 HAPROXY 설정파일

HAProxy 설정 파일을 구성.

```
# vi roles/haproxy_simple/templates/haproxy.cfg.j2
global
  log /dev/log local0
  maxconn 20480
  daemon
defaults
  log global
  mode tcp
  option tcplog
  timeout connect 5s
  timeout client 60s
  timeout server 60s
```

ROLE 랩 HAPROXY 설정파일

HAProxy 설정 파일을 구성.

```
frontend mysql
  bind 0.0.0.0:3306
  default_backend galera_nodes

backend galera_nodes
  balance roundrobin
  option mysql-check user root post-41
  default-server inter 2s fall 3 rise 2 maxconn 2000
{% for h in groups['galera'] %}
  server {{ h }} {{ hostvars[h].ansible_host }}:{{ mariadb_port }} check
{% endfor %}
```

ROLE 랩 HAPROXY 설치

HAProxy 설치 및 설정 파일 배포한다.

```
# vi roles/haproxy_simple/tasks/main.yml
- name: Install haproxy
  package:
    name: haproxy
    state: present

- name: Render haproxy.cfg
  template:
    src: haproxy.cfg.j2
    dest: /etc/haproxy/haproxy.cfg
    mode: '0644'
  notify: Restart HAProxy
```

ROLE 랩 HAPROXY 설치

HAProxy 설치 및 설정 파일 배포한다.

```
- name: Enable and start haproxy
  service:
    name: haproxy
    enabled: yes
    state: started
```

ROLE 랩 변수 구성

HAProxy 재시작 핸들러를 구성한다.

```
# vi roles/haproxy_simple/handlers/main.yml
- name: Restart HAProxy
  service:
    name: haproxy
    state: restarted
```

컨테이너

podman

개요

앤서블로 포드만 사용하기 위해서 collection 설치가 필요하다. 설치를 위해서 다음과 같이 실행하여 모듈을 추가한다. 인벤토리가 필요하면 같이 생성한다.

예제로 생성하는 조건은 다음과 같다.

1. Nginx container image
2. container-volume
3. container-network

```
# ansible-galaxy collection install containers.podman
# vi inventory
[containers]
node1 ansible_host=192.168.10.11
```

플레이북 변수

플레이북이 사용할 변수를 아래와 같이 생성 및 구성한다.

```
# mkdir vars
# vi vars/podman.yaml
app_name: my-nginx
app_image: docker.io/library/nginx:1.27
app_port: "8080:80"
app_net: appnet
app_root: /srv/nginx
```

포드만 자원 생성

앞에서 생성한 변수와 컬렉션을 선택한다. `collections:` 항목은 굳이 작성하지 않아도 동작한다.

```
# vi nginx-podman.yaml
---
- name: Run nginx on Podman with systemd
  hosts: containers
  become: true
  vars_files:
    - vars/podman.yml
  collections:
    - containers.podman
```

포드만 자원 생성

포드만 설치가 되어 있지 않는 경우, 설치 후 진행한다.

```
tasks:
  - name: Ensure Podman installed (RHEL/Rocky/Alma 계열)
    package:
      name:
        - podman
        - podman-plugins
      state: present
  - name: Create app content dir
    file:
      path: "{{ app_root }}"
      state: directory
      mode: "0755"
```

포드만 자원 생성

컨테이너가 사용할 웹 파일, 그리고 네트워크를 생성한다.

```
  - name: Place a simple index.html
      copy:
        dest: "{{ app_root }}/index.html"
        content: |
          <!doctype html><h1>Hello, {{ app_name }} (Podman+Ansible)</h1>

    - name: Create Podman network
      podman_network:
        name: "{{ app_net }}"
        state: present
```

포드만 자원 생성

포드만에서 사용할 이미지를 다운로드 한다.

```
    - name: Pull image
      podman_image:
        name: "{{ app_image }}"
        state: present
```

포드만 자원 생성

Nginx 컨테이너를 실행하기 위해서 컨테이너가 사용할 자원을 생성 및 구성한다.

```
    - name: Run container
      podman_container:
        name: "{{ app_name }}"
        image: "{{ app_image }}"
        state: started
        restart_policy: always
        networks:
          - name: "{{ app_net }}"
        published_ports:
          - "{{ app_port }}"      # host:container
```

포드만 자원 생성

컨테이너에서 사용할 **네트워크/볼륨/상태정보**를 설정한다.

```
        volume:
          - "{{ app_root }}:/usr/share/nginx/html:Z"  # SELinux 대비 :Z
        healthcheck:
          test: ["CMD-SHELL", "curl -fsS http://localhost/ || exit 1"]
          interval: 30s
          timeout: 3s
          retries: 3
          start_period: 5s
```

포드만 자원 생성

마지막으로 해당 컨테이너를 시스템 서비스로 구성하기 위해서 systemd의 서비스 자원으로 생성한다.

```
    - name: Generate systemd unit for the container
      podman_generate_systemd:
        name: "{{ app_name }}"
        dest: /etc/systemd/system
        restart_policy: always
        new: true
    - name: Reload systemd & enable service
      systemd:
        daemon_reload: true
        name: "container-{{ app_name }}.service"
        enabled: true
        state: started
```

실행

위의 내용 계속 이어서…

```
# ansible-playbook -i hosts nginx-podman.yml
# curl http://<node1 IP>:8080
Hello, my-nginx (Podman+Ansible)
```

가상머신

libvirtd

개요

리눅스에서 기본적으로 제공하는 libvirt 기반으로 인스턴스 생성한다. 일반적으로 앤서블에서 순서을 통해서 구성이 된다.

1. 이미지 다운로드
2. 디스크 준비
3. 네트워크 확인
4. VM 생성/부팅 확인

위 순서대로 가상머신 혹은 인스턴스를 생성한다.

변수 선언

컨테이너 생성 시, 사용할 가상머신을 선언한다.

```
# vi libvirt_cirros.yml
---
- name: Create a CirrOS VM on libvirt
  hosts: localhost
  connection: local
  become: true
  vars:
    vm_name: cirros-01
    vm_vcpus: 1
    vm_memory_mb: 256
```

변수 선언

위의 내용 계속 이어서…

```
    vm_disk_size_gb: 2
    vm_network: default
    images_dir: /var/lib/libvirt/images
    cirros_version: "0.6.2"
    cirros_base: "cirros-{{ cirros_version }}-x86_64-disk.qcow2"
    cirros_url: "https://download.cirros-
cloud.net/{{ cirros_version }}/{{ cirros_base }}"
    cirros_base_path: "{{ images_dir }}/{{ cirros_base }}"
    vm_disk_path: "{{ images_dir }}/{{ vm_name }}.qcow2"
```

전 처리 작업 수행

전 작업을 선언 및 수행한다. 여기서는 **pre_tasks:** 통해서 패키지 설치 작업을 수행한다.

```
  pre_tasks:
    - name: Install Virtualization Host group package (RHEL/Rocky/Alma)
      ansible.builtin.command:
        cmd: dnf groupinstall -y "Virtualization Host"
      args:
        creates: /usr/libexec/qemu-kvm
      when: ansible_facts['os_family'] == "RedHat"
```

전 처리 작업 수행

레드햇 계열이 아닌 경우 아래처럼 처리한다.

```
- name: Install libvirt/kvm packages (Debian/Ubuntu)
    package:
      name:
        - libvirt-daemon
        - libvirt-daemon-system
        - qemu-kvm
        - virtinst
        - libvirt-clients
        - python3-libvirt
        - libguestfs-tools
      state: present
    when: ansible_facts['os_family'] == "Debian"
```

전 처리 작업 수행

가상머신 생성을 위해서 libvirtd 서비스를 활성화 한다.

```
    - name: Enable and start libvirtd
      service:
        name: "{{ 'libvirtd' if ansible_facts['os_family'] == 'RedHat' else
'libvirtd' }}"
        enabled: true
        state: started
```

본 작업

가상 머신 생성을 위해서 본 작업을 `tasks:` 통해서 수행한다.

```
tasks:
  - name: Ensure default libvirt network is active & autostart
    community.libvirt.virt_net:
      name: "{{ vm_network }}"
      state: active
      autostart: true
```

본 작업

이미지 디렉터리는 기본적으로 생성이 된다. 검증용도로 추가.

```
    - name: Ensure images directory exists
      file:
        path: "{{ images_dir }}"
        state: directory
        owner: root
        group: root
        mode: "0755"
```

본 작업

가상머신 생성을 위해서 외부에서 이미지를 다운로드 한다.

```
    - name: Download CirrOS base image (if not exists or changed)
      get_url:
        url: "{{ cirros_url }}"
        dest: "{{ cirros_base_path }}"
        mode: "0644"
        force: false
        timeout: 60
```

본 작업

가상머신 선언 및 생성한다.

```
- name: Define & start CirrOS VM
    community.libvirt.virt:
      name: "{{ vm_name }}"
      state: running
      autostart: true
      memory_mb: "{{ vm_memory_mb }}"
      vcpus: "{{ vm_vcpus }}"
      cpu: host-model
```

본 작업

생성 시, 사용할 디스크 크기 및 네트워크를 구성한다.

```
        disks:
          - name: vda
            type: disk
            device: disk
            capacity: "{{ vm_disk_size_gb }}G"
            format: qcow2
            src: "{{ vm_disk_path }}"
            cache: none
        networks:
          - network: "{{ vm_network }}"
            type: network
            model: virtio
```

본 작업

가상머신에서 사용할 그래픽 카드 및 부팅 디스크 선언한다.

```
        graphics:
          type: vnc
          listen: 0.0.0.0
          autoport: true
        boot:
          devices:
            - hd
```

본 작업

최종적으로 생성된 가상머신을 정보를 가져온 후, 화면에 출력한다.

```
    - name: Show VM info
      community.libvirt.virt:
        command: info
        name: "{{ vm_name }}"
      register: vm_info

    - name: Debug VM info
      debug:
        var: vm_info
```

실행

아래 명령어로 libvirt 기반으로 가상머신을 생성한다.

```
# ansible-galaxy collection install community.libvirt
# ansible-playbook -K -i 'localhost,' libvirt_cirros.yml
```

플레이북 1

최종연습

2025-12-26

PLAYBOOK

앞에서 배운 기본적인 지식을 통해서 간단한 플레이북을 작성해본다. 현재 우리가 가지고 있는 서버는 총 6+대의 서버를 가지고 있다.

- 앤서블 컨트롤러 노드
- 1/2/3/4/5/6 노드

각각 서버를 인벤토리에 등록 후 텍스트 파일을 생성 및 전달을 한다. 이를 구성하기 위해서 다음과 같이 디렉터리를 생성한다.

디렉터리 구성

다음과 같이 플레이북을 생성 및 구성한다.

```
# mkdir playbook1
# cd playbook1
# touch inventory
# touch ansible.cfg
# mkdir roles
```

플레이북 디렉터리

```
# tree -L 2 .
.
├── ansible.cfg
├── inventory
└── roles
1 directory, 2 files
```

PLAYBOOK

```
# vim inventory
[node1]
192.168.90.11 → node1.example.com
[node2]
192.168.90.12 → node2.example.com
```

PLAYBOOK

```
# echo "Hello an automation World" > welcome.html
# vim copy.yaml
- hosts: node1
  tasks:
  - name: install a httpd package on {{ inventory_hostname }}
    yum:
      name: httpd
      state: latest
```

PLAYBOOK

올바르게 동작하면 추가로 다음과 같이 기능을 추가한다.

```
- name: copy the welcome.html to {{ inventory_hostname }}
  copy:
    content: "Hello an automation world"
    dest: /var/www/html/welcome.html
```

PLAYBOOK

PLAYBOOK

웹 서버가 설치가 안되어 있으면 아래와 같이 웹 서버 패키지도 같이 설치를 진행한다.

```
- name: install a httpd package on {{ inventory_hostname }}
  yum:
    name: httpd
    state: latest
- name: enable and start the service
  service:
    name: httpd
    state: started
    enabled: yes
```

PLAYBOOK

웹 페이지 배포 플레이북을 생성한다.

```
- name: copy the welcome.html to {{ inventory_hostname }}
  copy:
    src: welcome.html
    dest: /var/www/html/welcome.html
 - name: enable and start the service
   service:
     name: httpd
     state: started
     enabled: yes
```

플레이북 2

최종연습

2025-12-26

PLAYBOOK 2

지금까지 학습한 내용을 가지고 간단하게 플레이북 생성. **플레이북** 2에서는 인벤토리 변수를 사용해서 기능을 더 추가 해보도록 하겠다.

가상 머신(node1/2/3/4) **플레이북**를 통해서 패키지 및 "welcome.html"파일은 배포 하였지만, 호스트 이름을 아직 수정하지 않았다. 호스트 이름을 변수에 다음처럼 선언한다.

INVENTORY

인벤토리를 다음처럼 수정을 한다.

```
[node1]
10.10.1.1 nodename=node1.example.com

[node2]
10.10.1.2 nodename=node2.example.com
```

PLAYBOOK

그리고 추가적으로 플레이북 파일을 더 생성한다. 메인 파일에 "호스트 이름 변경"에 대한 부분을 통합한다.

```
# touch hostname.yaml
# vim hostname.yaml
- name: set up hostname to {{ inventory_hostname }}
  hostname:
    name: "{{ nodename }}"
# vim main.yaml
  - name: setup hostname
    import_tasks: hostname.yaml
```

플레이북 + 역할

다중 플레이북 구성

역할 설명

이쯤이면 슬슬 혼동이 오는 부분, 바로 roles vs playbook이다.

roles에는 tasks, files, templates같은 기능들이 모여서 하나의 기능을 구현하여 이걸 호스트에 제공하는 기능.
앞서 이야기 하였지만, 일종의 함수와 같은 역할을 한다.

역할 설명

플레이북에서 하나 이상의 **역할(roles)** 모아서 특정 작업에 대한 프로세스를 구현하고 있는 파일.

예를 들어서 "webserver.yaml", "dbserver.yaml"이런 식으로 좀 더 추상적인 작업의 워크 플로우를 role기반으로 구성하고 있다. 플레이북은 항상 다음과 같은 내용을 가지고 있다.

1. 어떤 호스트에서 어떠한 작업을 진행할 것인가?
2. 어떠한 역할을 가지고 작업을 할 것인가?
3. YAML에 어떠한 작업(task)나 혹은 역할이 선언이 되었는가?

_PLAYBOOK

통합

2025-12-26

include_playbook

앞에서 만든 플레이북을 하나로 통합한다.

```
- include_playbook: prepare_os.yaml
- include_playbook: webserver.yaml
- include_playbook: dbserver.yam
```

import_playbook

`import_playbook`은, 이미 작성한 플레이북 파일을 메모리에 적재하는 키워드 명령어. 미리 메모리에 적재를 하기 때문에 실행 전 문법 검사가 선행되기 때문에, 문법에 문제가 있는 경우 실행이 되지 않는다.

include_playbook

`include_playbook`은 앤서블 인터프리터가 해당문법(include)를 만나면 그 순간 이벤트가 발생하여 명시된 파일을 불러 옴. 장점은 빠르지만, 단점은 불러오는 순간에 문법 검사를 하기 때문에 문법에 문제가 있는 경우 중간에 종료가 됨.

include_playbook

플레이북에 기능을 추가한다. 운영체제의 모든 패키지를 최신으로 업데이트 한다.

```
# vi roles/prepare_os/tasks/main.yaml
- name: update all of thing
  yum:
    name: *
    state: latest
```

include_playbook

```
# vi roles/webserver/tasks/main.yaml
- name: install a httpd package
  yum:
    name: httpd
    state: latest
- name: start and enable the httpd service
  service:
    name: httpd
    enabled: yes
    state: started
```

include_playbook

```
# vi roles/webserver/tasks/main.yaml
- name: install a mariadb
  yum:
    name: "{{ item }}"
    state: latest
  loop:
    - mariadb
    - mariadb-server
```

include_playbook

```
- name: start and enable the mariadb service
  service:
    name: mariadb-server
    enabled: yes
    state: started
```

include_playbook

```
# vi prepare_os.yaml
- name: Updated to all node hosts recently package
  hosts: all
  roles:
    - { role: prepare_os }
```

include_playbook

```
# vi webserver.yaml
- name: Updated to all node hosts recently package
  hosts: webserver
  roles:
    - { role: webserver }
```

include_playbook

```
# vi dbserver.yaml
- name: install and configuration the database server
  hosts: dbserver
  roles:
    - { role: dbserver }
```

include_playbook

```
# vi allinone_playbooks.yaml
- include_playbook: prepare_os.yaml
- include_playbook: webserver.yaml
- include_playbook: dbserver.yam
```