7/31 2일차

내용 Remind

최근 트렌드에선 자동화가 필수이다.

예전 IT의 문제점들을 해결하기 위해 복잡한 시스템들을 편하게 관리하기 위해

지금은 자동화 도구가 없으면 구성 자체가 힘들다

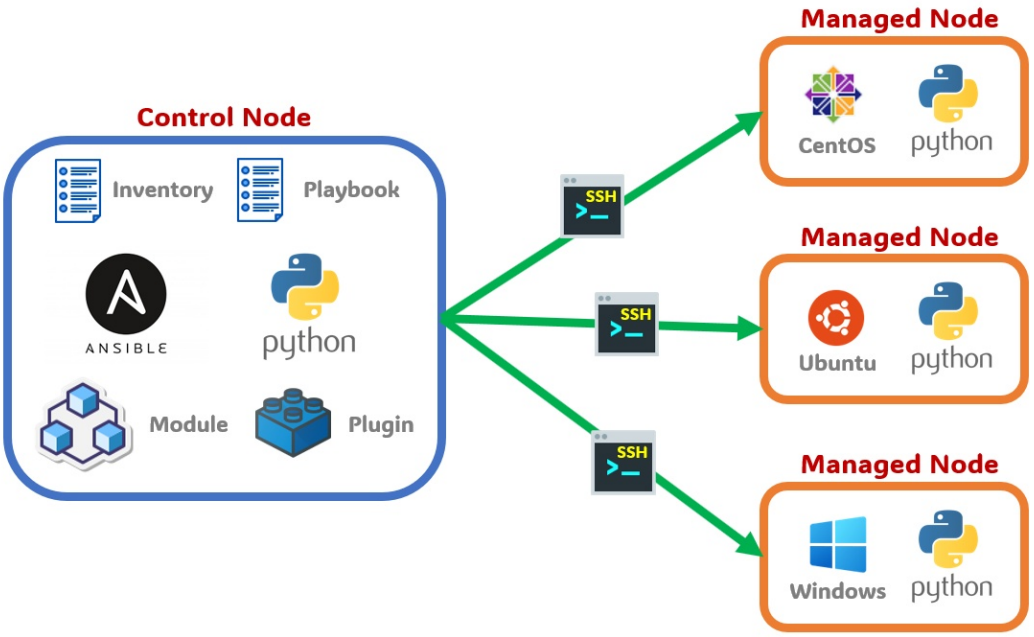
Ansible 같은 자동화 도구들로 cloud native 환경에서 마이크로 서비스 아키텍처 구성

젠킨스 argocd로 ci cd 구성

terraform은 프로비저닝

만들어진 것들에게 이렇게 동작해라 명령을 넣어주는게 ansible

Ansible 구조



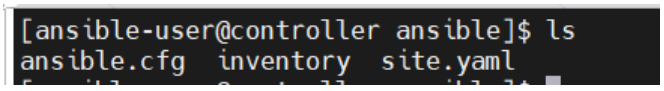
모든 node에는 python이 깔려있어야 하며 ssh로 접근을 한다. ssh로 접근을 하기 위해 no password 옵션을 활성화 해줘야 하며 rsa 키 파일 생성과 공유키를 먼저 전달하는 과정이 선행되어야 한다.

Ansible 사용 절차

- 환경 구성

1. OS 설치
2. 네트워크 설정
3. Ansible 패키지 설치 (epel repo)

- 실제 사용



1. ansible.cfg

```
[defaults]
inventory = ./inventory
remote_user = ansible-user
ask_pass = false

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

2. inventory (/etc/hosts 파일도 작성)

```
servera
serverb
```

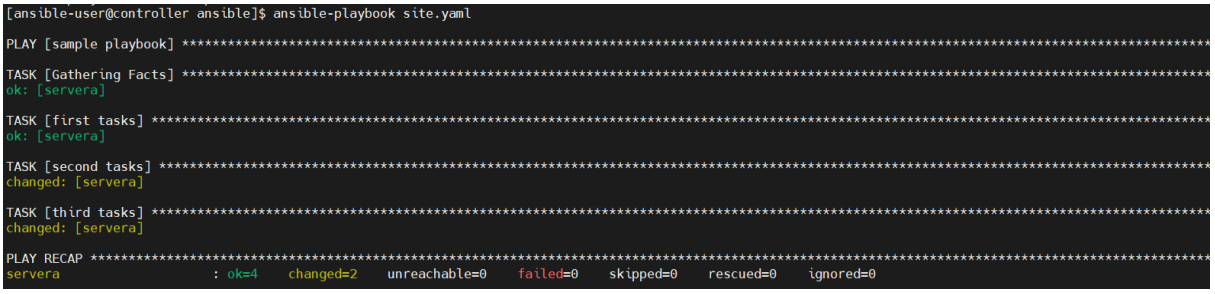
3. playbook 파일 작성

```
- name: sample playbook
  hosts: servera
  tasks:
  - name: first task
    yum: # 설치 모듈
      name: httpd
      state: latest # 최신을 설치하라
      #state: absent # 삭제
      #state: present  # 없으면 설치
  - name: second task
```

```
    service: # 데몬을 올린다. 직접하려면 systemctl 명령어 사용
      name: httpd
      state: started
  - name: third task


  - name: sample playbook
  hosts: servera
  tasks:
  - name: first tasks
    yum:
      name: httpd
      state: latest
  - name: second tasks
    service:
      name: httpd
      state: started
  - name: third tasks
    service:
      name: firewalld
      state: stopped
```

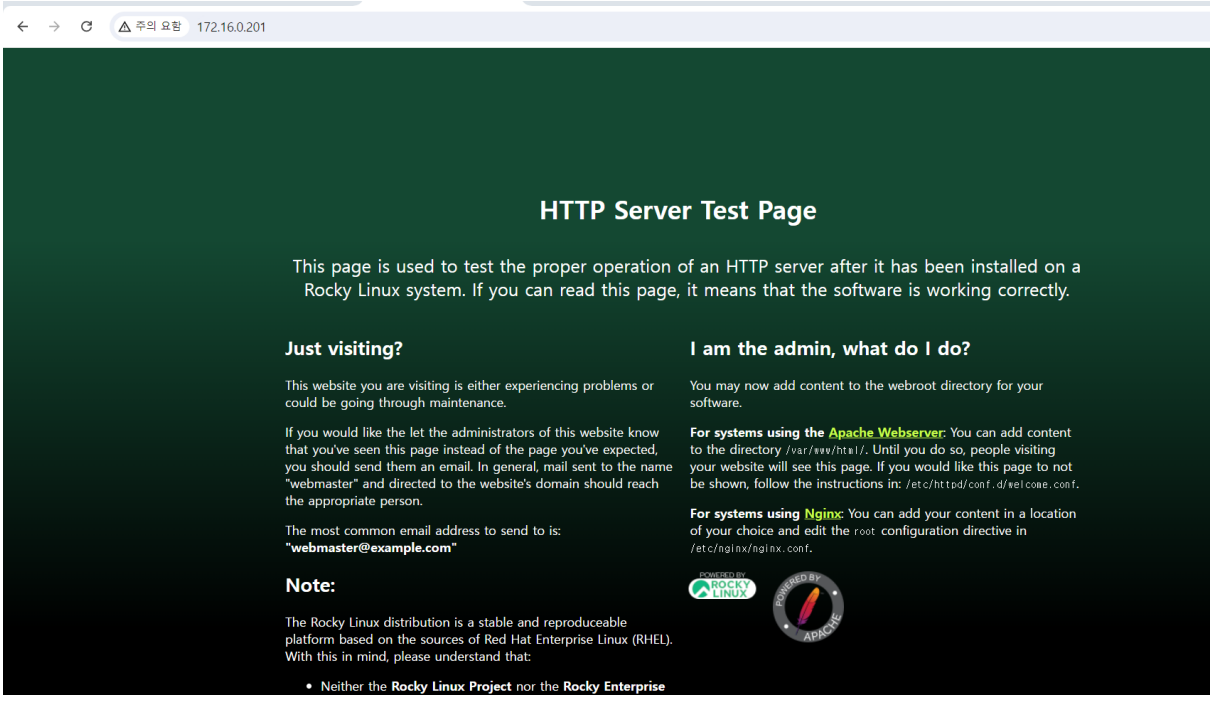
httpd yum으로 다운받고 http systemctl로 실행시키고 방화벽 내린다.

- 실행 결과



어제 실행됐던 것과 동일하면 ok 변경된 부분만 changed

- http 접속 결과



> 💡 Ansible을 잘 사용하기 위해선 모듈을 잘 알고 그 모듈 아래에 들어가는 요소를 알아야 한다. 모듈들은 1000개 가량 있고 핵심 모듈은 수십 개이다. 현업에서 ansible 사용하면 가이드가 다 있다.

playbook inventory 파일은 지속적으로 관리해야 한다.

- 작업 결과 리포트

상세한 작업의 경과를 출력하기 위해서는 -v 옵션을 사용할 수 있습니다. v문자의 개수에 따라 수준을 조절할 수 있습니다.

-v : 간단한 작업 결과를 표시

-vv : 작업 결과와 구성 내용을 표시

-vvv : 관리 호스트에 연결하기 위한 추가 정보도 표시

-vvvv : 스크립트를 사용했다면 관련 내용도 함께 표시

앤서블은 플레이북을 실행하기 전에 ***--syntax-check*** 옵션으로 해당 구문의 유효성을 미리 검사하는 기능을 지원합니다. 구문에 문제가 있을 경우 오류를 보고 합니다. 또한 비슷한 기능으로 ***--check*** 옵션을 통해 실제 작업을 처리하지 않지만 작업에 대한 결과가 어떻게 보고 되는지를 확인할 수 있는 기능을 지원합니다.

다중 플레이

다중 플레이를 사용하면 다른 호스트 그룹에서 실행할 수 있다.

즉 호스트 작업 대상이 다르게 해서 실행.

```
- name: first play
  hosts: dbservers
  tasks:
    - name: first task
      service:
        name: mariadb
        enabled: true
- name: second play
  hosts: webservers
  tasks:
    - name: first task
      service:
        name: httpd
        enabled: true
```

YAML 문법

yaml은 사람이 읽고 쓰기 쉽도록 개발되었습니다. 처음 사용하는 사용자도 불편함 없이 금방 익혀서 사용할 수 있도록 문법적인 제약이 강하지 않지만 몇가지 규칙을 알고 있어야 합니다. 먼저 yaml에서 주석을 사용할수 있습니다. 주석은 해시 기호(#) 를 제일 왼쪽에 입력합니다. 중간에 해시 기호가 나올 경우 해시 앞에 공백을 두어야 합니다.

```
# this is comment
data # also this is comment
```

yaml은 dictionary 와 list 데이터 형식을 사용합니다. 키-값 페어 데이터는 dictionary 데이터 형으로 인라인 형식의 중괄호를 사용합니다.

```
name: testservice
serviceport: 80

{name: testservice, serviceport: 80}
```

대시 기호를 사용한 list 데이터는 대괄호를 사용하여 인라인 형식으로 입력할 수 있습니다.

```
hosts:
  - servera
  - serverb
  - serverc

hosts: [servera, serverb, serverc]
```

또한 플레이북 내에서 속기 형식을 입력할 수 있습니다. 동일한 수준의 데이터를 같은 행에 입력하는 방식이라서 들여쓰기가 많을수록 번거로운 작업을 피할 수가 있지만, 익숙하지 않은 사용자는 가독성이 떨어질 수 있습니다.

```
tasks:
  - name: httpd service start
    service: name:httpd enabled=true state=started
```

실습

- webserver.yaml

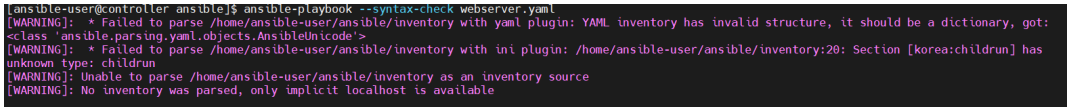
```
# webserver.yaml
---
- name: Install and Start Apache web server
  hosts: webservers
  vars:
    http_port: 80
    max_clients: 200
    remote_user: root
  tasks:
    - name: Ensure Apache is at the latest version
      yum:
        name: httpd
        state: latest

    - name: Write the Apache config file
      copy:
        src: index.html
        dest: /var/www/html/index.html

    - name: Ensure Apache is running
      service:
        name: httpd
        state: started
        enabled: yes
```

실행 방법

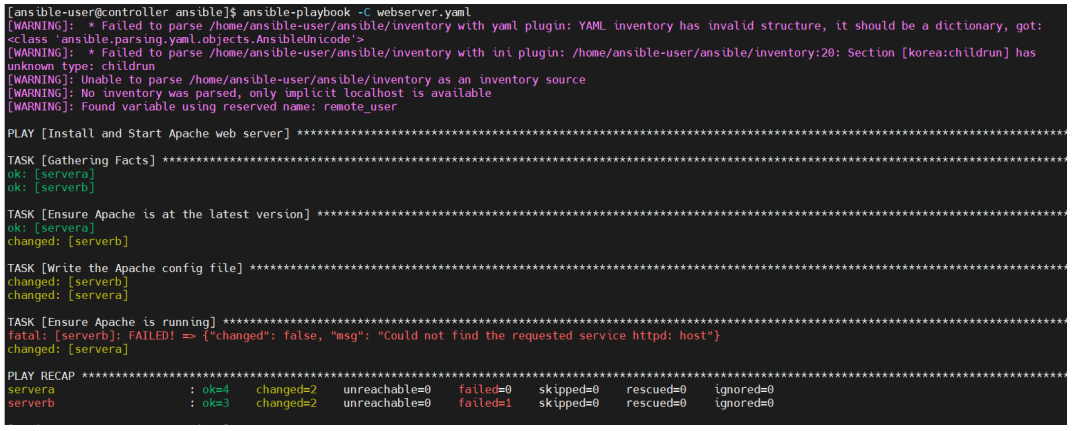
```
ansible-playbook --syntax-check webserver.yaml              # yaml 파일 문법 체크
ansible-playbook -C webserver.yaml                          # 실행전 변경사항 체크
```

```
ansible-playbook -v webserver.yaml                    # 작업 결과 표시
ansible-playbook  webserver.yaml                      # 일반 실행
```

- ansible-playbook --syntax-check webserver.yaml

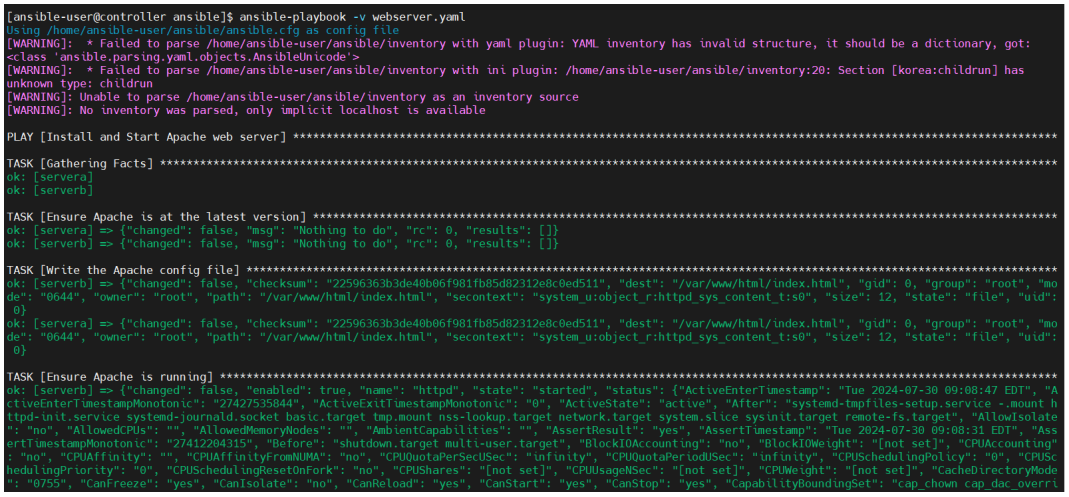


- ansible-playbook -C webserver.yaml



실행 전에 변경사항만 체크하고 실제 설치는 하지 않기에 올리려고 할 때 오류 발생

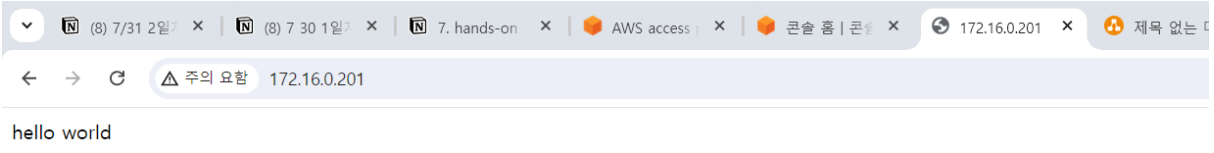
- ansible-playbook -v webserver.yaml



- ansible-playbook webserver.yaml

```
[ansible-user@controller ansible]$ ansible-playbook webserver.yaml
[WARNING]:  * Failed to parse /home/ansible-user/ansible/inventory with yaml plugin: YAML inventory has invalid structure, it should be a dictionary, got:
<class 'ansible.parsing.yaml.objects.AnsibleUnicode'>
[WARNING]:  * Failed to parse /home/ansible-user/ansible/inventory with ini plugin: /home/ansible-user/ansible/inventory:20: Section [korea:childrun] has
unknown type: childrun
[WARNING]: Unable to parse /home/ansible-user/ansible/inventory as an inventory source
[WARNING]: No inventory was parsed, only implicit localhost is available

PLAY [Install and Start Apache web server] *************************************************************************************************************

TASK [Gathering Facts] *********************************************************************************************************************************
ok: [servera]
ok: [serverb]

TASK [Ensure Apache is at the latest version] **********************************************************************************************************
ok: [servera]
changed: [serverb]

TASK [Write the Apache config file] ********************************************************************************************************************
changed: [servera]
changed: [serverb]

TASK [Ensure Apache is running] ************************************************************************************************************************
changed: [servera]
changed: [serverb]

PLAY RECAP *********************************************************************************************************************************************
servera                    : ok=4    changed=2    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
serverb                    : ok=4    changed=3    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

결과



```
[root@serverb ~]# netstat -ntlp
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      930/sshd
tcp6       0      0 :::80                   :::*                    LISTEN      6019/httpd
tcp6       0      0 :::22                   :::*                    LISTEN      930/sshd
[root@serverb ~]# exit
logout
Connection to serverb closed.
[root@controller ~]# ssh servera
Activate the web console with: systemctl enable --now cockpit.socket

Last login: Tue Jul 30 07:50:23 2024 from 172.16.0.200
[root@servera ~]# netstat -ntlp
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      941/sshd
tcp6       0      0 :::22                   :::*                    LISTEN      941/sshd
tcp6       0      0 :::80                   :::*                    LISTEN      5512/httpd
[root@servera ~]#
```

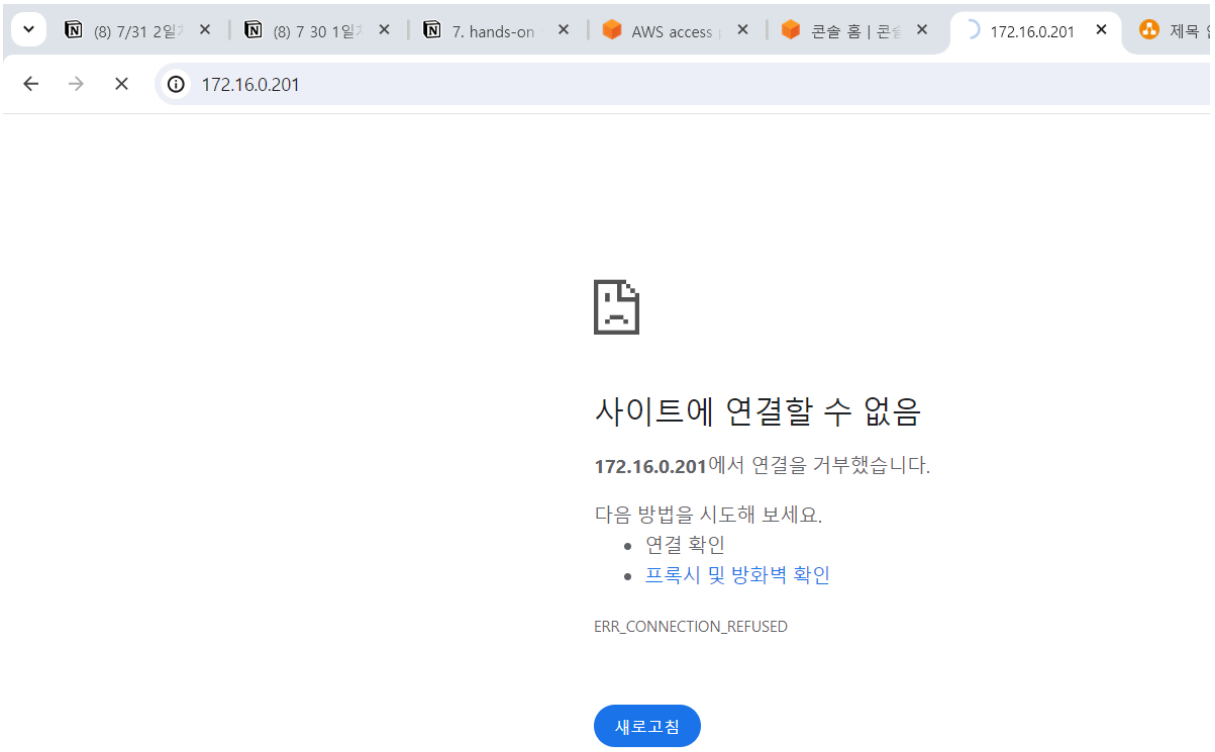
삭제

- webserver-delete.yaml

```yaml
# webserver-delete.yaml
---
- name: Delete Apache webserver
  hosts: webservers
  tasks:
    - name: HTTPD is deleted.
      yum:
        name: httpd
        state: absent
```

- 삭제 후

```
[ansible-user@controller ansible]$ ansible-playbook webserver-delete.yaml
[WARNING]:  * Failed to parse /home/ansible-user/ansible/inventory with yaml plugin: YAML inventory has invalid structure, it should be a dictionary, got:
<class 'ansible.parsing.yaml.objects.AnsibleUnicode'>
[WARNING]:  * Failed to parse /home/ansible-user/ansible/inventory with ini plugin: /home/ansible-user/ansible/inventory:20: Section [korea:childrun] has
unknown type: childrun
[WARNING]: Unable to parse /home/ansible-user/ansible/inventory as an inventory source
[WARNING]: No inventory was parsed, only implicit localhost is available

PLAY [Delete Apache webserver] ************************************************************************************************************************************************************

TASK [Gathering Facts] ********************************************************************************************************************************************************************
ok: [servera]
ok: [serverb]

TASK [HTTPD is deleted.] ******************************************************************************************************************************************************************
changed: [servera]
changed: [serverb]

PLAY RECAP ********************************************************************************************************************************************************************************
servera                    : ok=2    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
serverb                    : ok=2    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```



변수 관리

Ansible은 플레이에서 재사용할 수 있는 값을 저장하는데 사용하는 변수를 지원합니다. 관리가 간편해지고 오류의 발생을 줄이는데 기여할 수 있습니다. 변수는 여러 위치에 정의할 수 있습니다.

예를 들어 특정 호스트 그룹 또는 개별 호스트에만 영향을 미치는 변수를 설정할 수 있고 일부 변수는 시스템 구성에 따라 Ansible에서 설정할 수 있는 변수도 있으며 플레이북 내부에서 플레이북의 한 가지 플레이 또는 해당 플레이의 한 가지 작업에만 영향을 미치는 다른 변수를 설정할 수 있습니다.

또한 --extra-vars 또는 -e 옵션을 사용해서 ***ansible-playbook*** 명령어의 인자로 전달할 수 있습니다. 이렇게 서로 다른 위치에서 사용되는 변수에 동일한 이름을 사용해서 변수를 설정한다면 위치에 따라 정해져 있는 우선순위에 따라 사용할 값이 결정됩니다.

변수의 위치마다 정해진 우선순위는 다음 문서에서 확인할 수 있습니다.

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

- 우선순위

1. command line values (for example, `-u my_user`, these are not variables)
2. role defaults (as defined in Role directory structure) [1]
3. inventory file or script group vars [2]
4. inventory group_vars/all [3]
5. playbook group_vars/all [3]
6. inventory group_vars/* [3]
7. playbook group_vars/* [3]
8. inventory file or script host vars [2]
9. inventory host_vars/* [3]
10. playbook host_vars/* [3]
11. host facts / cached set_facts [4]
12. play vars
13. play vars_prompt
14. play vars_files
15. role vars (as defined in Role directory structure)
16. block vars (only for tasks in block)
17. task vars (only for the task)
18. include_vars
19. set_facts / registered vars
20. role (and include_role) params
21. include params
22. extra vars (for example, `-e "user=my_user"`)(always win precedence)

💡 scope가 클 수록 우선순위가 작아진다.

group보다 host가 우선순위가 더 높고 playbook이 host보다 높다.

inventory는 전체 집합이기에 playbook보다 낮다.

플레이북 변수

플레이를 작성할 때 자신만의 변수를 정의한 다음, 작업에서 해당 값을 호출할 수 있습니다. vars라는 키워드는 플레이에서 변수를 선언할 때 사용합니다.

💡 tasks는 마지막에 적고 hosts를 첫 번째로 vars를 그 사이에 가능한 위에 적는다 가독성을 위해 문법적으로 순서가 상관은 없으나 가독성을 위함

```
- hosts: all
  vars:
    user: user01
    home: /home/user01
```

플레이북 파일에 직접 변수를 입력하지 않고 분리된 파일에서 변수를 관리할 수도 있습니다. vars_files 키워드는 외부의 변수 파일을 호출해 변수를 읽어 들이는 키워드 입니다.

- user.yaml

```
# user.yaml
user: user01
home: /home/user01
```

```
- hosts: all
  vars_files:
    - user.yaml
```

플레이에서 변수를 사용하고자 할 경우 이중 중괄호를 사용해 변수명을 호출 하여 사용할 수 있습니다.

```
vars:
  user: user01

tasks:
  - name: create user {{ user }}
    user:
      name: "{{ user }}"
```

이때 한 가지 조심해야 할 점은 변수가 값을 시작하는 첫 번째 요소로 입력할 경우에 따옴표를 반드시 붙여 줘야 합니다. 만약 따옴표가 없을 경우 문법적인 오류가 발생하고 에러 메세지가 출력됩니다.

인벤토리 변수

호스트에 직접 적용되는 인벤토리 변수는 특정 호스트에 적용되는 호스트 변수와 그룹 호스트로 나뉩니다. 그룹에는 중첩 그룹 그리고 모든 호스트에 적용되는 그룹 변수도 포함됩니다. 호스트 변수가 그룹 변수보다 우선하지만, 플레이북에서 정의한 변수가 이 두 변수보다 우선합니다.

다음은 인벤토리에 호스트 변수를 선언하는 예시입니다.

```
[webservers]
web01.test.com    user=user01
```

그리고 이어서 그룹 변수를 지정하는 예시입니다. 일반 그룹과 중첩 그룹 모두 지정할 수 있습니다. 또한

```
[servers1]
test1.example.com
test2.example.com

[servers2]
test3.example.com
test4.example.com

[servers:children]
servers1
servers2

[server1:vars]
user=user01

[servers:vars]
user=user02
```

인벤토리 파일 역시 변수만 따로 분리하여 외부에 위치 시킬 수 있습니다. 이때 작업 디렉토리 내부에 group_vars 및 host_vars 라는 두 디렉토리를 생성하여 정의합니다. 이 디렉토리 내부에서는 호스트 이름 및 그룹 이름과 동일한 파일로 명명하여 변수만 따로 분리 하여 관리합니다. servers 그룹에 대한 변수를 정의 하는 경우 다음처럼 구성합니다.

- 전체 구조

```
ansible_project
├── ansible.cfg
├── group_vars
│ ├── seoul
│ └── busan
├── host_vars
│ ├── servera.example.com
│ ├── serverb.example.com
│ ├── serverc.example.com
├── inventory
└── playbook.yml
```

실습

- group_vars/seoul

```
# group_vars/seoul
package: httpd
```

- group_vars/busan

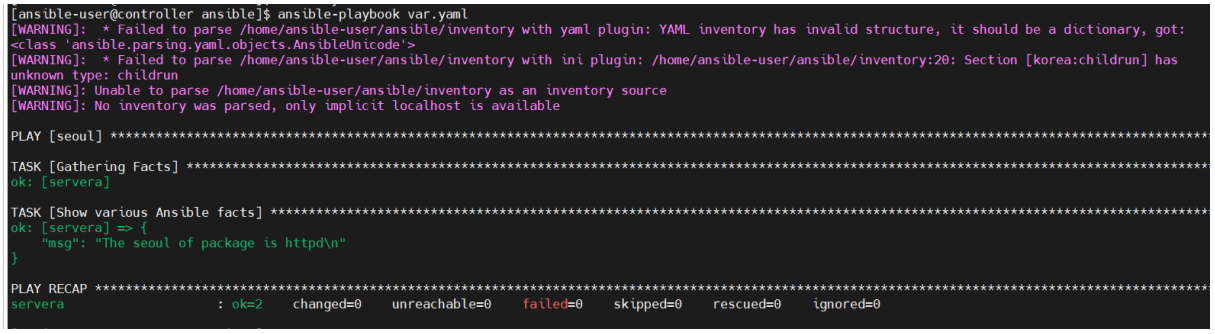
```
# group_vars/busan
package: apache
```

- var.yaml

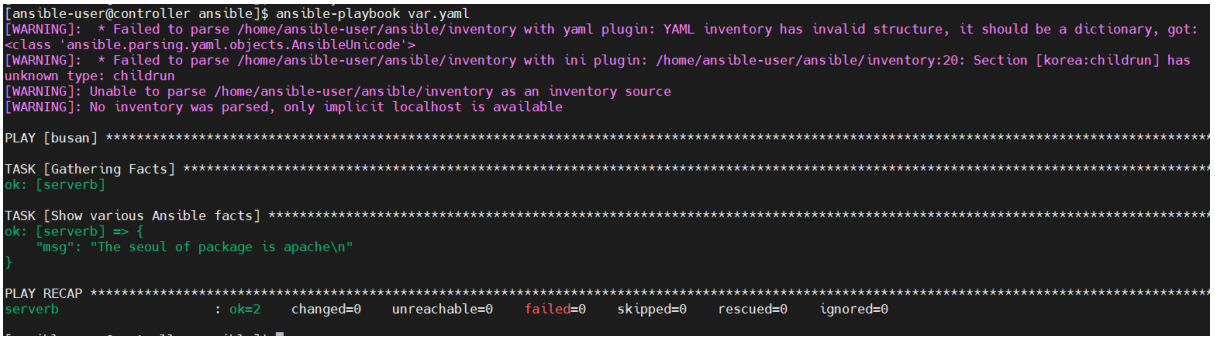
```
# var.yaml
---
- hosts: seoul
  tasks:
  - name: Show various Ansible facts
    debug:
      msg: >
        The seoul of package is {{ package }}
```

- 결과

1. 그룹이 서울일 때



2. 그룹이 부산일 때



플레이북 파일과 인벤토리 파일에 지정한 변수는 플레이를 실행하는 명령어를 입력하는 시점에 재지정할 수 있도록 -e 옵션을 지원합니다.

변수 파일 암호화

Ansible에는 암호 또는 API 키 등의 중요한 데이터에 대한 액세스 권한이 필요할 수 있습니다. 일반적으로 이 정보는 인벤토리 변수 또는 분리된 파일에서 일반 텍스트로 저장됩니다.

그러나 파일에 대한 액세스 권한이 있는 일반 사용자나 파일을 저장하는 버전 제어시스템에 이 중요한 데이터에 대한 액세스 권한이 있는 경우 보안 상의 위험이 있습니다. Ansible과 함께 포함되어 있는 Ansible Vault를 사용하여 Ansible에서 사용하는 모든 데이터 파일을 암호화하고 암호를 해독할 수 있습니다. Ansible Vault를 사용하려면 ansible-vault 라는 명령줄 툴을 사용하여 파일을 생성하고, 편집하고, 암호화하고, 암호 해독하고, 확인할 수 있습니다.

Ansible Vault는 Ansible에서 사용하는 모든 데이터 파일을 암호화할 수 있습니다. 여기에는 인벤토리 변수,플레이북에 포함된 변수 파일, 플레이북 실행 시 인수로 전달된 변수 파일 또는 Ansible 역할(role)에서 정의된 변수가 포함될 수 있습니다.

새로 암호화된 파일을 만들려면 ***ansible-vault create*** 명령을 사용합니다. 이 명령을 통해 새 Vault 암호에 대한 메시지가 나타나며 기본 편집기 vi 를 사용하여 파일이 열립니다. EDITOR 환경 변수를 내보내 다른 기본 편집기를 지정할 수 있습니다.

표준입력을 통해 암호를 입력하지 않고 암호가 저장된 파일을 인자로 전달하기 위해서 ***--vault-password-file*** 라는 옵션을 지원합니다. 예를 들어 vault-pass라는 파일에 암호가 저장되어 있다면 다음과 같이 입력합니다.

```
ansible-vault create --vault-password-file=vault-pass site.yaml
```

암호화된 파일을 확인하기 위해서 ***ansible-vault view*** 라는 명령어를 사용할 수 있습니다. 암호화된 기존 파일을 편집할 때는 ***ansible-vault edit*** 명령어를 사용할 수 있습니다. 또는 기존의 파일을 새롭게 암호화 할수 있습니다.

ansible-vault encrypt 명령어에 인자를 일반 파일로 지정하면 암호를 입력하여 암호화 할 수 있습니다. 그리고 거꾸로 암호화된 파일을 일반 파일로 변경할 경우 ***ansible-vault decrypt*** 명령어를 사용합니다. 마지막으로 암호를 변경할 경우 ***ansible-vault rekey*** 명령어를 사용할 수 있습니다.

플레이북에서 암호화된 파일에 접근하려면 마찬가지로 암호가 필요합니다. ***ansible-playbook*** 명령어로 실행할 때 앤서블은 암호화된 파일에 접근하기 위한 암호를 전달하지 않으면 에러가 발생합니다. 암호를 전달할 때는 3가지 방법이 있습니다.

- 프롬프트 입력
- 암호파일
- 환경변수 설정 (ANSIBLE_VAULT_PASSWORD)

명령어를 입력할 때 프롬프트에 직접 입력하거나 혹은 암호가 저장된 파일을 인자로 전달할 수 있습니다. 혹은 환경변수를 설정해 놓으면 자동으로 암호가 입력되어 동작합니다. 암호파일을 인자로 전달하기 위해서는 앞의 예제와 마찬가지로 ***--vault-password-file*** 옵션을 사용합니다. 그리고 환경변수는 ANSIBLE_VAULT_PASSWORD를 사용합니다.

실습

- valut pass 파일 활용해서 암호 파일 생성

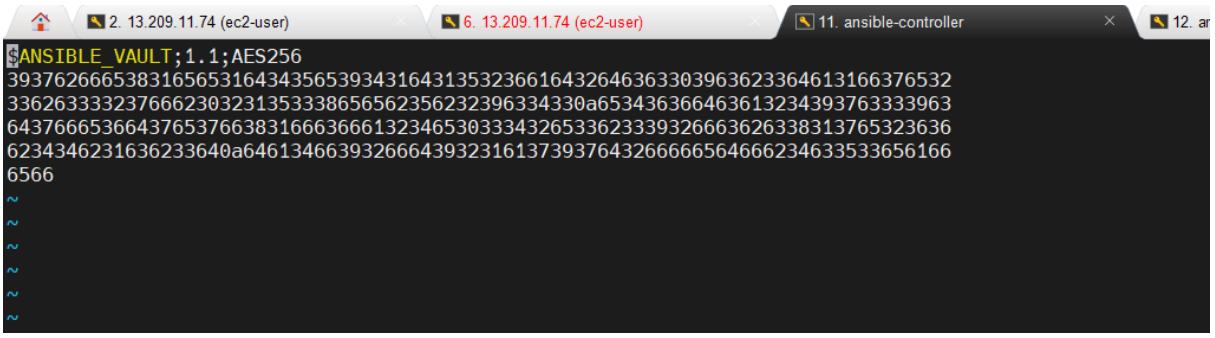
```
ansible-vault create --vault-password-file=vault-pass site.yaml
```

vim 에디터로 빈 파일 열리고 내용 적고 wq

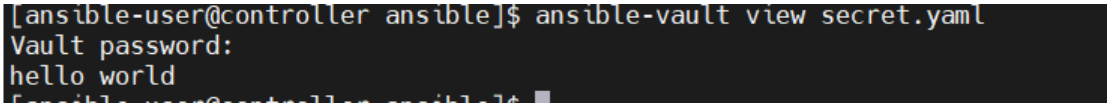
- 암호화된 파일 확인

```
ansible-vault view site.yaml
```

그냥 vi editor로 확인 시



view로 확인 시

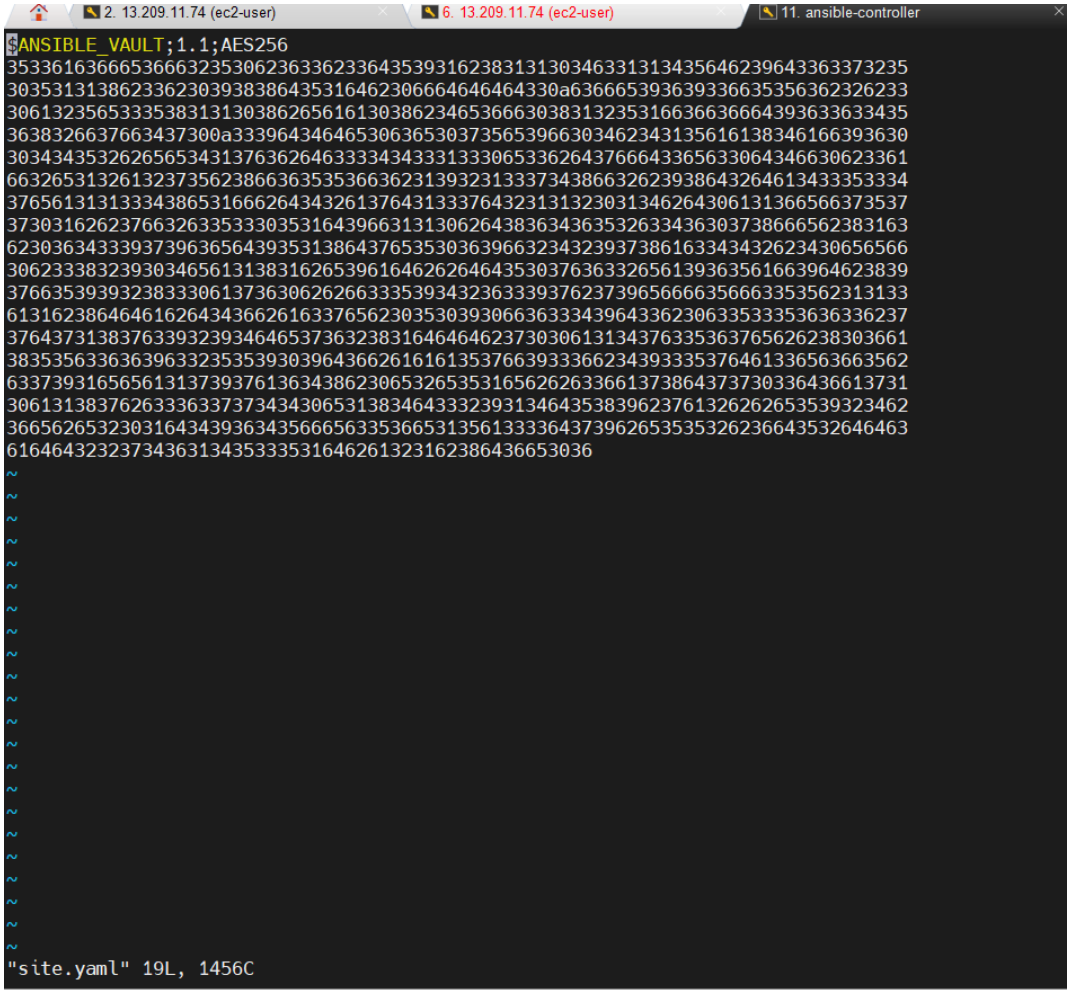


- 파일 편집

```
ansible-vault edit site.yaml
```

- 기존 파일 암호화

```
ansible-vault encrypt webserver.yaml
ansible-vault encrypt --vault-password-file=vault-pass site.yaml
```



암호화 된 것 확인 가능

- 암호 변경

```
ansible-vault rekey site.yaml
```

- 암호화된 파일을 복호화

```
ansible-vault decrypt webserver.yaml
```

- 플레이북 실행에 암호화된 파일 사용

```
echo 'Test123!' > vault-pass

ansible-vault encrypt --vault-password-file=vault-pass group_vars/seoul
ansible-playbook var.yaml   # 에러발생
```

```
ansible-playbook --vault-password-file=vault-pass var.yaml
ansible-vault decrypt --vault-password-file=vault-pass group_vars/seoul
```

팩트 변수

ansible facts는 관리 호스트에서 자동으로 검색한 호스트 별 정보가 포함되어 있는 변수 입니다. 팩트에는 다음의 내용이 포함됩니다.

- 호스트 이름
- 커널 버전
- 네트워크 인터페이스 이름
- 네트워크 인터페이스 IP 주소
- 운영 체제 버전
- CPU 개수
- 사용 가능한 메모리
- 스토리지 장치의 크기 및 여유 공간

시스템에서 고유한 사용자 지정 팩트를 생성할 수도 있습니다. 팩트는 관리 대상 호스트의 상태를 검색하고 해당 상태에 따라 수행할 조치를 결정하는 편리한 방법입니다. 예를 들면 다음과 같습니다.

- 플레이는 특정 서비스의 상태와 같이 수집된 팩트 값에 따라 조건부 작업을 사용하여 서버를 재시작할 수 있습니다.
- 팩트에서 보고하는 사용 가능한 메모리에 따라 플레이에서 MySQL 구성 파일을 사용자 지정할 수 있습니다.
- 구성 파일에 사용된 IPv4 주소는 팩트 값에 따라 설정할 수 있습니다.

일반적으로 모든 플레이는 첫 번째 작업을 수행하기 전에 setup 모듈을 자동으로 실행하여 팩트를 수집합니다.그러면 Ansible 2.3 이상에서는 Gathering Facts 작업으로 결과에 보고됩니다. 기본적으로 setup 모듈은 자동으로 실행되기 때문에 플레이에서 따로 정의할 필요가 없습니다.

팩트 변수는 상위 수준의 필드 ansible_facts 로 분류합니다. 예를 들어 시스템의 호스트 명을 확인할 때 ***ansible_facts.hostname*** 으로 검색할 수 있습니다. 혹은 ***ansible_facts['hostname']***으로 입력할 수도 있습니다. 그리고 플레이북에서 사용할 때는 이중 중괄호를 입력하여 변수 명을 입력합니다. 예를 들어 호스트 명과 IP를 확인하는 플레이북은 다음처럼 작성합니다.

```
- hosts: all
  tasks:
  - debug:
      msg: >
        Hostname is {{ ansible_facts.hostname }}
        and IP is {{ ansible_facts.default_ipv4.address }}
```

팩트 변수는 보통 자동으로 실행되지만 부하를 줄이고 플레이 속도를 높이고자 실행되지 않도록 비활성화 할 수 있습니다. 플레이 속성에 gather_facts 를 no로 설정하는 것입니다.

```
---
- name: disable gather facts
  hosts: all
  gather_facts: no
```

```
  tasks:
...(생략)...
```

자동으로 실행되지 않도록 비활성화 하더라도 setup 모듈을 tasks에 정의해서 수동으로 입력해서 실행하게 만들수 있습니다.

```
tasks:
  - name: manually gather facts
    setup:
```

setup 모듈에는 gather_subset 하위 속성이 있습니다. 예를 들어 전체 팩트 정보를 가져오는 대신 하드웨어 정보만 수집하려면 다음처럼 입력합니다.

```
tasks:
  - name: hardware facts collect
    setup:
      gather_subset:
        - hardware
```

혹은 느낌표(!)를 이름 앞에 붙여서 특정 정보만 제외할 수 있습니다.

```
tasks:
  - name: hardware facts collect
    setup:
      gather_subset:
        - !hardware
```

팩트는 사용자 지정 팩트를 사용하여 관리 호스트의 특정 값을 임의로 정의할 수 있습니다. 플레이는 사용자 지정 팩트를 사용하여 구성 파일을 채우거나 조건부로 작업을 실행할 수 있습니다. 사용자 지정 팩트는 각 관리 호스트에 로컬로 저장됩니다. 이러한 팩트는 관리 호스트에서 setup 모듈이 실행될 때 수집되는 표준 팩트 목록으로 통합됩니다. INI 또는 JSON 파일에서 사용자 지정 팩트를 정적으로 정의하거나 플레이를 실행할 때 동적으로 생성할 수 있습니다. 동적 사용자 지정 팩트는 JSON 출력을 생성하는 실행 가능한 스크립트를 통해 수집됩니다. 기본적으로 setup 모듈은 각 관리 호스트의 /etc/ansible/facts.d 디렉터리에 있는 파일 및 스크립트를 가져옵니다. 각 파일 또는 스크립트는 이름이 .fact 로 끝나야 사용할 수 있습니다. 동적 사용자 지정 팩트 스크립트는 JSON 형식의 팩트를 출력하고 실행 가능해야 합니다. 다음의 정적 사용자 지정 팩트 파일 예는 INI 형식으로 작성됩니다. INI 형식의 사용자 지정 팩트 파일에는 섹션에서 정의한 최상위 수준과 정의할 팩트에 대한 키-값 쌍이 차례로 포함되어 있습니다.

```
[packages]
web_package = httpd
db_package = mariadb-server

[users]
user1 = user01
user2 = user02
```

실습

- servera의 팩트 변수를 가지고 온다.

```
Ansible -m setup servera
```

매직 변수

매직 변수는 Ansible에 의해 자동으로 설정되는 변수로써 특정 관리 호스트의 고유 정보를 습득합니다.

hostvars	관리 호스트의 변수가 포함되며, 다른 관리 호스트 변수에 대한 값을 가져오는 데 사용할 수 있습니다. 해당 호스트에 대해 아직 수집되지 않은 경우 관리 호스트의 팩트는 포함되지 않습니다.
group_names	현재 관리 호스트가 속한 모든 그룹을 나열합니다.
groups	인벤토리의 모든 그룹 및 호스트를 나열합니다.
inventory_hostname	인벤토리에 구성된 현재 관리 대상 호스트의 호스트 이름이 포함됩니다. 이 호스트 이름은 다양한 이유로 팩트에서 보고하는 호스트 이름과 다를 수 있습니다.

```
# magic-var.yaml
---
- hosts: seoul
  become: true
  tasks:
    - name: Show ansible_facts
      debug:
        var: hostvars['servera.example.com']['ansible_all_ipv4_addresses']
```

```
# magic-var2.yaml
---
- hosts: webservers
  gather_facts: yes
  tasks:
    - name: Here we are checking ansible_hostname
      debug:
        msg: "hostname is {{ ansible_hostname }}"
    - name: Here we are checking inventory_hostname
      debug:
        msg: "hostname is {{ inventory_hostname }}"
```

작업 제어

사용자는 특정 작업을 반복한다거나 혹은 특정 조건에 따라 작업을 수행할지 여부를 결정하도록 플레이에서 작업을 제어할 수 있습니다. 이를 통해 좀 더 다양한 환경에서 사용할 수 있는 고도화 된 플레이북을 생성할 수 있습니다.

반복문

반복문을 사용하면 동일한 모듈을 사용하는 작업을 여러 개 작성하지 않아도 됩니다. 반복문을 구성하여 목록의 각 항목, 목록에 있는 각 파일의 콘텐츠, 생성된 숫자 시퀀스 또는 더 복잡한 구조를 사용하여 작업을 반복할 수 있습니다. 일련의 항목에 대해 작업을 반복하려면 loop 키워드를 사용하면 됩니다. 이때 loop 키워드로 구성한 항목들은 item 이라는 변수로 반복 치환됩니다.

```
- name: mail server check
  service:
    name: postfix
    state: started
- name: mail server check2
  service:
    name: dovecot
    state: started
```

위의 예시에서 반복적으로 동일한 모듈을 사용하는데 이름만 변경해서 사용하고 반복문을 작성한다면 다음 처럼 구성할 수 있습니다.

```
- name: mail server check
  service:
    name: "{{ item }}"
    state: started
  loop:
    - postfix
    - dovecot
```

loop문에서도 변수를 사용할 수 있습니다.

```
vars:
  mail_services:
    - postfix
    - dovecot
tasks:
  - name: mail server check
    service:
      name: "{{ item }}"
      state: started
    loop: "{{ mail_services }}"
```

또한 각 항목은 사전 데이터형으로 구성할 수도 있습니다.

```
- user:
    name: "{{ item['name'] }}"
    state: present
    groups: "{{ item['groups'] }}"
  loop:
    - name: user01
      groups: wheel
    - name: user02
      groups: root
```

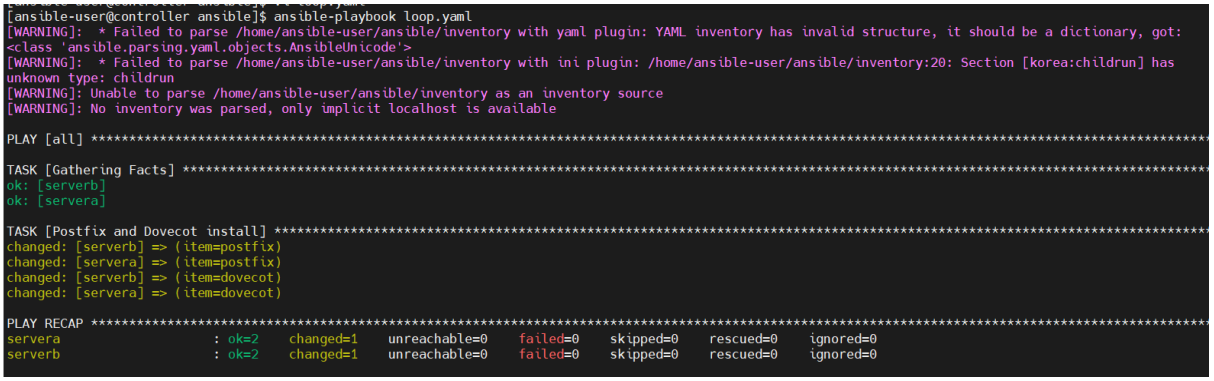
리스트 항목 개수만큼 반복

반복문 실습

1. **단순 반복문**

- loop.yaml

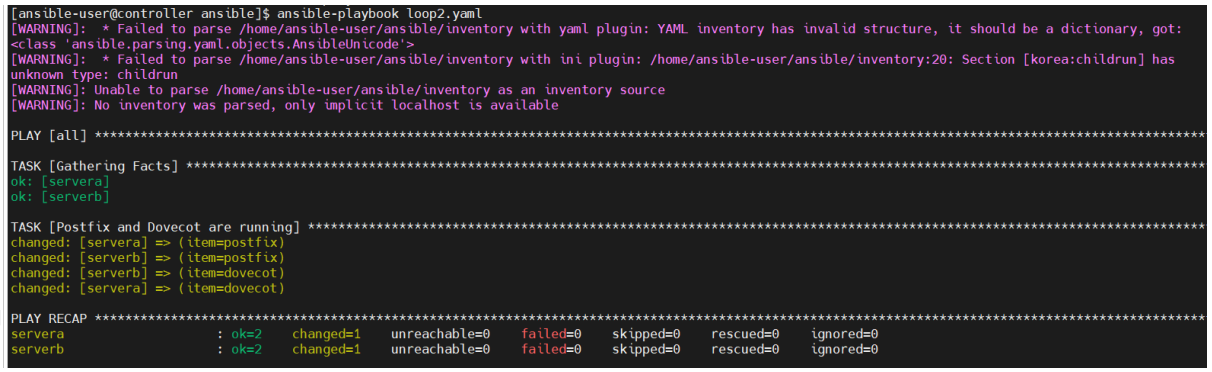
```
# loop.yaml
---
- hosts: all
  tasks:
    - name: Postfix and Dovecot install
      yum:
        name: "{{ item }}"
        state: latest
      loop:
        - postfix
        - dovecot
```

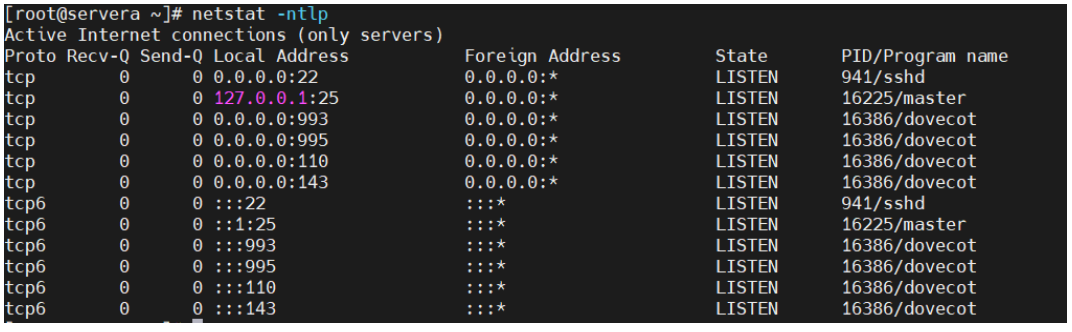


- loop2.yaml

```yaml
# loop2.yaml
---
- hosts: all
  vars:
    mail_services:
      - postfix
      - dovecot
  tasks:
    - name: Postfix and Dovecot are running
      service:
        name: "{{ item }}"
        state: started
      loop: "{{ mail_services }}"
```

- 결과 확인





```
[root@serverb ~]# netstat -ntlp
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 0.0.0.0:993             0.0.0.0:*               LISTEN      11112/dovecot
tcp        0      0 0.0.0.0:995             0.0.0.0:*               LISTEN      11112/dovecot
tcp        0      0 0.0.0.0:110             0.0.0.0:*               LISTEN      11112/dovecot
tcp        0      0 0.0.0.0:143             0.0.0.0:*               LISTEN      11112/dovecot
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      930/sshd
tcp        0      0 127.0.0.1:25            0.0.0.0:*               LISTEN      10950/master
tcp6       0      0 :::993                  :::*                    LISTEN      11112/dovecot
tcp6       0      0 :::995                  :::*                    LISTEN      11112/dovecot
tcp6       0      0 :::110                  :::*                    LISTEN      11112/dovecot
tcp6       0      0 :::143                  :::*                    LISTEN      11112/dovecot
tcp6       0      0 :::22                   :::*                    LISTEN      930/sshd
tcp6       0      0 ::1:25                  :::*                    LISTEN      10950/master
```

2. **사전 목록에 대한 반복문**

- loop3.yaml

```yaml
# loop3.yaml
---
- hosts: all
  tasks:
    - name: Users exist and are in the correct groups
      user:
        name: "{{ item['name'] }}"
        state: present
        groups: "{{ item['groups'] }}"
      loop:
        - name: jane
          groups: wheel
        - name: joe
          groups: root
```

- 결과

```
[ansible-user@controller ansible]$ ansible-playbook loop3.yaml
[WARNING]:  * Failed to parse /home/ansible-user/ansible/inventory with yaml plugin: YAML inventory has invalid structure, it should be a dictionary, got:
<class 'ansible.parsing.yaml.objects.AnsibleUnicode'>
[WARNING]:  * Failed to parse /home/ansible-user/ansible/inventory with ini plugin: /home/ansible-user/ansible/inventory:20: Section [korea:childrun] has
unknown type: childrun
[WARNING]: Unable to parse /home/ansible-user/ansible/inventory as an inventory source
[WARNING]: No inventory was parsed, only implicit localhost is available

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

TASK [Gathering Facts] *********************************************************************************************************************************
ok: [servera]
ok: [serverb]

TASK [Users exist and are in the correct groups] ********************************************************************************************************
changed: [serverb] => (item={'name': 'jane', 'groups': 'wheel'})
changed: [servera] => (item={'name': 'jane', 'groups': 'wheel'})
changed: [servera] => (item={'name': 'joe', 'groups': 'root'})
changed: [serverb] => (item={'name': 'joe', 'groups': 'root'})

PLAY RECAP *********************************************************************************************************************************************
servera                    : ok=2    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
serverb                    : ok=2    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

조건문

Ansible은 조건문을 사용하여 특정 조건이 충족될 때 작업 또는 플레이를 실행할 수 있습니다. 예를 들면 조건문을 사용하여 Ansible에서 서비스를 설치하거나 구성하기 전에 관리 호스트에서 사용 가능한 메모리를 확인할 수 있습니다.

조건문을 사용하면 관리 호스트 간의 차이점을 구분하고 충족된 조건을 기반으로 관리 호스트에 기능 역할을 할당할 수 있습니다. 플레이북 변수, 등록된 변수 및 Ansible 팩트는 모두 조건문을 사용하여 테스트할 수 있습니다. 문자열, 숫자 데이터 및 부울 값을 비교하는 연산자를 사용할 수 있습니다.

조건문은 **when** 이라는 키워드를 사용합니다. 이 항목은 체크할 조건을 값으로 사용합니다. 조건이 참으로 판단되면 작업이 실행되고 거짓이면 작업은 건너 뜁니다. 참과 거짓을 판단하기 위해서는 연산자가 필요합니다. 다음은 연산자 목록들입니다.

연산자	의미	예시
==	같음을 나타냄	ansible_facts['machine'] == "x86_64" max_memory == 512
<	보다 작음	min_memory < 128
>	보다 큼	min_memory > 256
<=	작거나 같음	min_memory <= 256
>=	크거나 같음	min_memory >= 512
!=	같지 않음	min_memory != 512
is defined	변수가 있음	min_memory is defined
in not defined	변수가 없음	min_memory is not defined
not	거짓일 때 실행	not memory_available
in	목록의 값에 포함	ansible_facts['distribution'] in supported_distros

마지막 항목에 대한 예시는 다음과 같은 예시로 활용합니다.

```
---
- hosts: all
  vars:
    supported_distros:
      - RedHat
      - Fedora
      - Centos
  tasks:
    - yum:
        name: http
        state: present
      when: ansible_facts['distribution'] in supported_distros
```

```
when: ansible_facts['distribution'] == "RedHat" or ansible_facts['distribution']
```

레드햇 계열이면 실행해라 yum 서비스

apt 서비스도 제공을 하긴 한다.

when이라는 키워드는 -를 붙이지 않아도 된다.

```
when: >
  ansible_facts['distribution_version'] == "9.0" and
  ansible_facts['kernel']  == "5.14.0-70.13.1.el9_0.x86_64"
```

and 연산자는 목록 데이터 형으로 지원하기도 합니다. 위의 and 예시문은 다음처럼 사용할 수 도 있습니다.

```
when:
  - ansible_facts['distribution_version'] == "9.0"
  - ansible_facts['kernel']  == "5.14.0-70.13.1.el9_0.x86_64"
```

또한 사용자는 반복문과 조건문을 결합하여 보다 다양한 플레이북을 작성할 수 있습니다. 즉 when 문과 loop 문을 결합하는 것입니다.

```
- yum:
    name: mariadb-server
    state: latest
  loop: "{{ ansible_facts['mounts] }}"
  when: item['mount'] == "/" and item['size_available'] > 300000000
```

Register 변수

조건을 설정할 때 많이 사용하는 방법 중 하나가 register 변수를 활용하는 것입니다. register 는 어떤 모듈이 실행되며 발생되는 이벤트를 캡쳐한 특별한 변수 입니다. 이 변수를 debug 같은 모듈로 호출해서 내용을 살펴보거나 혹은 변수의 값을 통해 판단하는 when 조건을 설정할 수 있습니다. 다음은 debug 모듈로 register 변수의 내용을 살펴보는 예시입니다

모듈마다 register로 들어가는 내용이 다르다.

```
- tasks:
    - name: postfix status
      command: /usr/bin/systemctl is-active postfix
      register: result
    - name: debugging
      debug:
        var: result
        verbosity: 2
```

이제 특정한 모듈의 결과 값으로 후속 작업의 실행 유무를 제어 할 수 있습니다.

```
- tasks:
    - name: postfix status
```

```
      command: /usr/bin/systemctl is-active postfix
      register: result
    - name: debugging
      service:
        name: httpd
        state: restarted
      when: result.rc == 0
```

return code가 0이라면 즉 정상적으로 끝났다면 httpd를 재시작한다.

register 실습

1. **실습1**

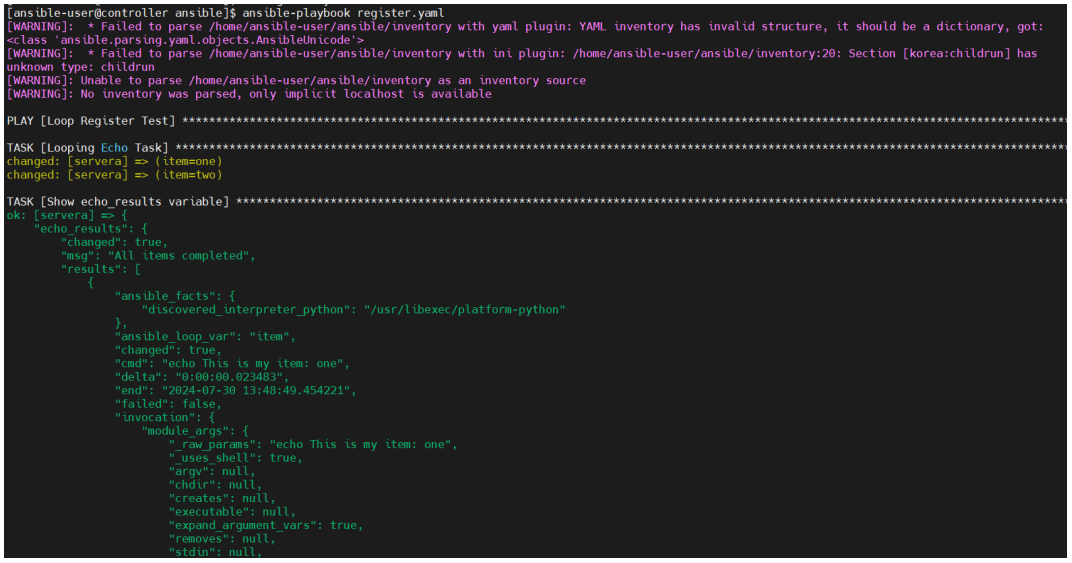
- register.yaml

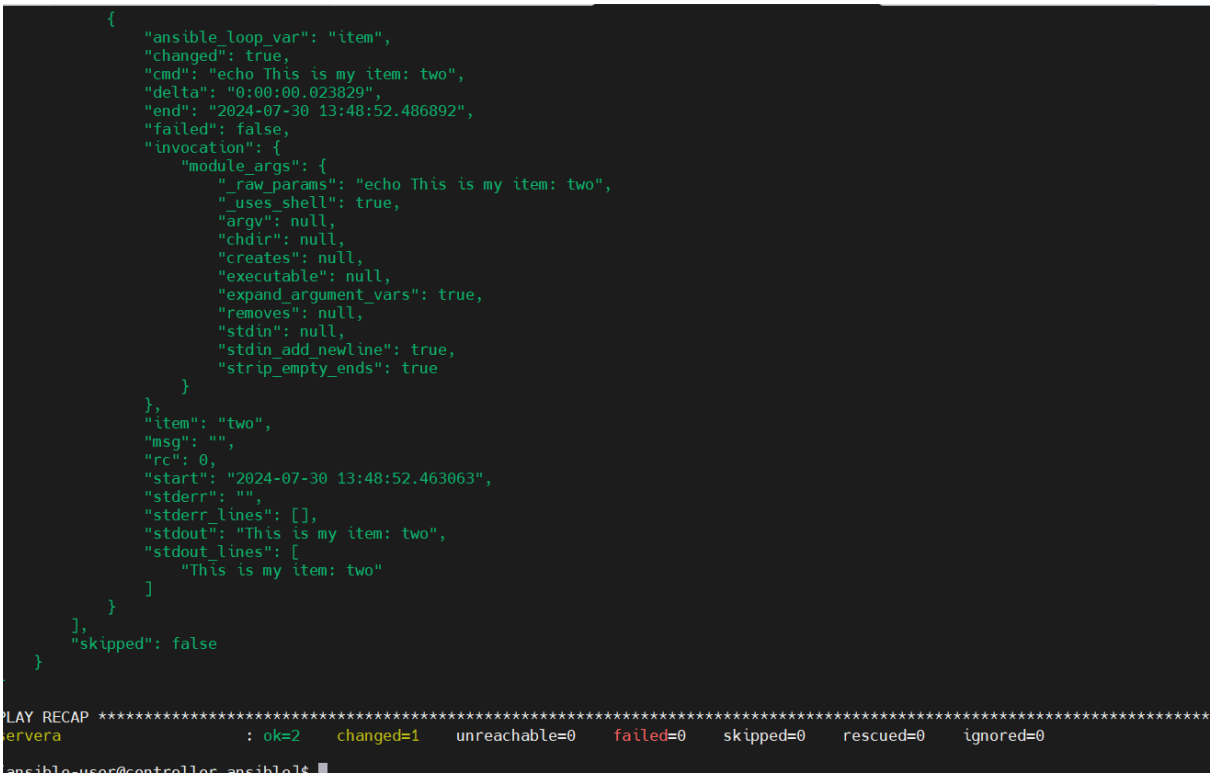
```
# register.yaml
---
- name: Loop Register Test
  gather_facts: no
  hosts: seoul
  tasks:
    - name: Looping Echo Task
      shell: "echo This is my item: {{ item }}"
      loop:
        - one
        - two
      register: echo_results  # echo_results 변수가 등록됩니다.

    - name: Show echo_results variable
      debug:
        var: echo_results     # echo_results 변수의 콘텐츠가 화면에 표시됩니다.
```

💡 debug는 화면에 출력하는 옵션이다.

one,two로 하여 반복문 2번 실행 그 결과를 echo_results에 담은 다음 debug 옵션으로 출력

- 결과





2. **실습2**

- register2.yaml

```
# register2.yaml
---
- name: Loop Register Test
  gather_facts: no
  hosts: seoul
  tasks:
```

```
    - name: Looping Echo Task
      ansible.builtin.shell: "echo This is my item: {{ item }}"
      loop:
        - one
        - two
      register: echo_results

    - name: Show stdout from the previous task.
      ansible.builtin.debug:
        msg: "STDOUT from previous task: {{ item['stdout'] }}"
      loop: "{{ echo_results['results'] }}"
```

- 결과

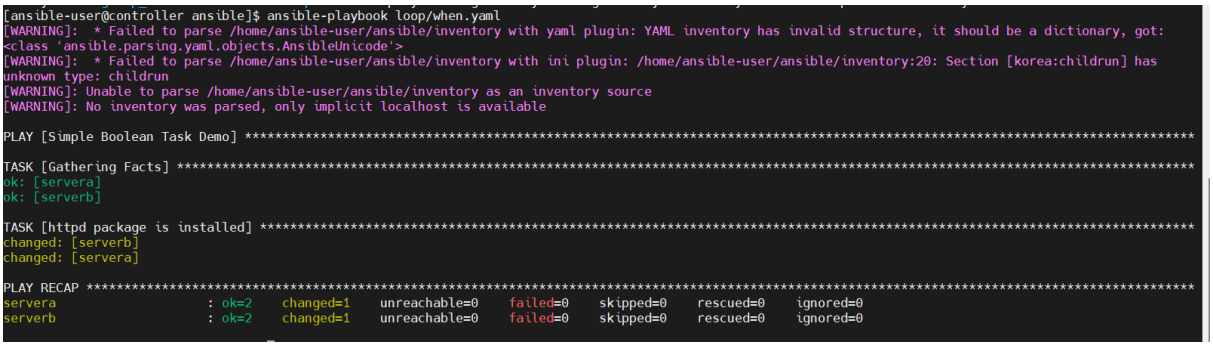
조건문 실습

1. **실습 1**

```
# when.yaml
---
- name: Simple Boolean Task Demo
  hosts: all
  vars:
    run_my_task: true
  tasks:
    - name: httpd package is installed
      yum:
        name: httpd
      when: run_my_task
```

플레이북 변수를 사용 true이니 yum httpd 진행

- 결과



2. **실습2**

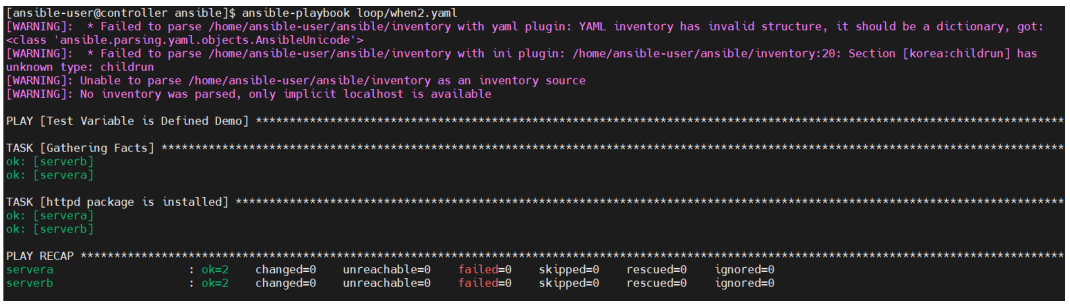
- when2.yaml

```
# when2.yaml
---
- name: Test Variable is Defined Demo
  hosts: all
  vars:
    my_service: httpd
  tasks:
    - name: "{{ my_service }} package is installed"
      yum:
        name: "{{ my_service }}"
      when: my_service is defined
```

플레이북 변수로 httpd 선언하고 when 조건문으로 선언됐는지 체킹한다.

선언 돼있으면 yum httpd 실행

- 결과



3. **실습3**

- ansible_facts.yaml

```
# ansible_facts.yaml
- name: Demonstrate the "in" keyword
  hosts: all
  gather_facts: yes
  vars:
    supported_distros:
      - RedHat
      - Fedora
      - Rocky
  tasks:
    - name: Install httpd using dnf, where supported
      ansible.builtin.dnf:
        name: httpd
        state: present
      when: ansible_facts['distribution'] in supported_distros
```

dnf를 지원하는 os만 ansible_facts로 걸러내서 진행

gather_facts는 fact 변수 수집 여부를 정한다.

dnf 대신 yum도 활용 가능

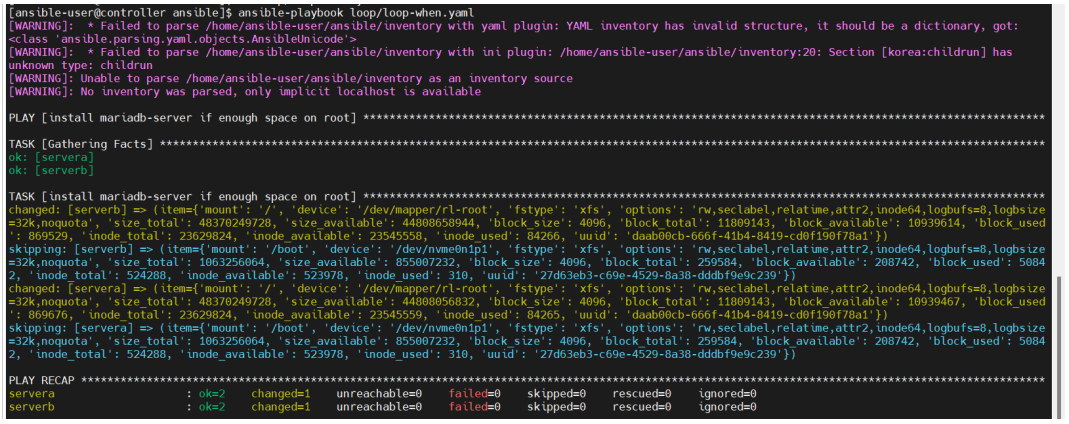
```
# ansible_facts.yaml
---
- name: Demonstrate the "in" keyword
  hosts: all
  gather_facts: yes
  vars:
    supported_distros:
      - RedHat
      - Fedora
      - Rocky
  tasks:
    - name: Install httpd using yum, where supported
      yum:
        name: httpd
        state: present
      when: ansible_facts['distribution'] in supported_distros
```

조건문과 반복문 결합 실습

```yaml
# loop-when.yaml
---
- name: install mariadb-server if enough space on root
  hosts: all
  gather_facts: yes
  tasks:
    - name: install mariadb-server if enough space on root
      yum:
        name: mariadb-server
        state: latest
      loop: "{{ ansible_facts['mounts'] }}"
      when: item['mount'] == '/' and item['size_available'] > 300000000
```

마운트 위치가 root이고 가용 사이즈가 300000000인 것만 loop 돌면서 보여준다

- 결과



조건문과 반복문, register 결합 실습

```yaml
# loop-when-register.yaml
---
- name: Restart HTTPD if Postfix is Running
  hosts: all
  tasks:
    - name: Get Postfix server status
      command: /usr/bin/systemctl is-active postfix # Postfix가 실행 중인지 확인
      register: result

    - name: Restart Apache HTTPD based on Postfix status
      service:
        name: httpd
```

```
        state: restarted
      when: result.rc == 0   # 결과가 0(성공)일때
```

변경된 작업의 후속 작업 - Handler

Ansible 모듈은 멱등이 되도록 설계되어 있습니다. 즉, 플레이북을 여러 번 실행해도 결과는 항상 동일합니다. 플레이 및 해당 작업은 여러 번 실행할 수 있지만 관리 호스트를 원하는 상태로 만드는 데 필요한 경우에만 변경됩니다. 이미 원하는 상태대로 되어 있다면 변경은 발생하지 않고 그대로 둡니다.

그리고 경우에 따라 어떤 작업에서 시스템이 변경되는 경우 추가 작업을 실행해야 할 수 있습니다. 예를 들어 서비스의 구성 파일을 변경하면 변경된 구성이 적용되도록 서비스를 다시 로드해야 할 수 있습니다.

핸들러(handler)는 **다른 작업에서 트리거한 알림에 반응하는 작업**입니다. 작업은 관리 호스트에서 작업이 변경될 때만 핸들러에 알림을 통해서 통지합니다. 각 핸들러는 플레이의 작업 블록 다음에 해당 이름으로 트리거됩니다. **작업에서 이름을 사용하여 핸들러에 알리지 않으면 핸들러가 실행되지 않습니다**. 하나 이상의 작업에서 **핸들러에 알리면 플레이의 다른 모든 작업이 완료된 후 핸들러가 한 번 실행**됩니다. 핸들러는 작업이므로 관리자가 다른 작업에 사용하려는 핸들러에 같은 모듈을 사용할 수 있습니다.일반적으로 핸들러는 호스트를 재부팅하고 서비스를 다시 시작하는 데 사용됩니다.

핸들러는 notify 문을 사용하여 명시적으로 호출된 경우에만 트리거되는 비활성 작업으로 간주할 수 있습니다. 다음 코드 조각은 구성 파일이 업데이트 되고 이를 알리는 경우에만 restart apache 핸들러에서 apache 웹 서버를 재시작하는 방법을 보여줍니다.

```
- tasks:
    - name: notify tasks
      template:
        src: /var/lib/templates/demo.conf
        dest: /etc/httpd/conf.d/demo-httpd.conf
      notify:
        - restart apache
handlers:
  - name: restart apache
    service:
      name: httpd
      state: restarted
```

restart apache 핸들러는 notify 구문의 이름과 동일하게 작성하여 해당 작업으로 인해 시스템이 변경이 되는 상황에 트리거 됩니다. **하나의 작업에서 해당 notify 섹션에 있는 핸들러를 두 개 이상 호출할 수 있습니다**. Ansible은 **notify 문을 배열로 취급**하고 핸들러 이름을 모두 호출합니다.

```
- tasks:
    - name: notify tasks
      template:
        src: /var/lib/templates/demo.conf
        dest: /etc/httpd/conf.d/demo-httpd.conf
      notify:
```

```
        - restart mysql
        - restart apache
handlers:
  - name: restart mysql
    service:
      name: mariadb
      state: restarted
  - name: restart apache
    service:
      name: httpd
      state: restarted
```

핸들러를 올바르게 사용하기 위해서는 다음의 내용을 파악하고 있어야 합니다.

- 핸들러는 항상 플레이의 handlers 섹션에서 지정한 순서대로 실행됩니다. 작업의 notify 문에 의해 나열된 순서나 작업이 핸들러에 알리는 순서대로 실행되지 않습니다.
- 핸들러는 일반적으로 플레이에서 다른 모든 작업이 완료된 후에 실행됩니다. 플레이북 중 tasks 부분의 작업에서 호출한 핸들러는 tasks 아래에 있는 모든 작업이 처리될 때까지 실행되지 않습니다.
- 핸들러 이름은 플레이별 네임스페이스에 있습니다. 두 핸들러에 같은 이름이 잘못 지정되면 하나의 핸들러만 실행됩니다.
- 둘 이상의 작업에서 하나의 핸들러에 알리는 경우 해당 핸들러가 한 번 실행됩니다. 핸들러에 알리는 작업이 없는 경우 핸들러가 실행되지 않습니다.
- notify 문이 포함된 작업에서 changed 결과를 보고하지 않으면 핸들러에 알리지 않습니다.

작업의 결과를 사용자가 임의로 changed 로 보고 하게 만들 수 있습니다. 조건문에서 사용된 when 키워드를 응용하여 ***changed_when*** 속성으로 조건을 설정하여 특정 조건에서 알림이 발생하도록 구성합니다. 다음 예시는 명령어를 실행하고 결과에 대한 메세지를 통해 핸들러 실행의 유무를 판단하는 식입니다.

```
tasks:
  - shell:
      cmd: /usr/local/bin/upgrade-database
    register: command_result
    changed_when: "'Success' in command_result.stdout"
    notify:
      - restart_database
```

혹은 강제로 false 값을 입력하여 항상 changed를 보고 하지 않도록 만들 수도 있습니다.

```
- name: validate config
  command: httpd -t
  changed_when: false
```


shell,command,raw 모듈들은 멱등이 되지 않는다. ansible은 멱등성이 중요하기에 이런 모듈들은 가급적 사용하지 않는 것이 좋다

핸들러 실습

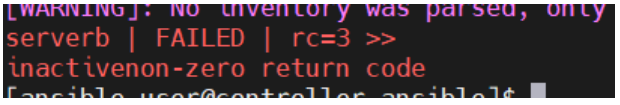
- notify와 handler 1

```
# handler.yaml
---
- name: Handlers Example
  hosts: serverb
  gather_facts: false
  tasks:
    - name: Install httpd latest version
      yum:
        name: httpd
        state: latest
      become: true
      notify: restart_httpd  # 해당 테스크의 changed 상태가 발견 되었을때 notify가 트리거 ㅂ

  handlers:
    - name: restart_httpd
      become: true
      service:
        name: httpd
        state: started
```

- 확인

```
ansible serverb -m shell -a 'systemctl is-active httpd'
```

이미 httpd가 설치되어 있으므로 해당 오류 발생




```
yum remove httpd를 serverb에서 실행 후 다시시도
```

```
[ansible-user@controller ansible]$ ansible-playbook handler/handler.yaml
[WARNING]:  * Failed to parse /home/ansible-user/ansible/inventory with yaml plugin: YAML inventory has invalid structure, it should be a dictionary, got:
<class 'ansible.parsing.yaml.objects.AnsibleUnicode'>
[WARNING]:  * Failed to parse /home/ansible-user/ansible/inventory with ini plugin: /home/ansible-user/ansible/inventory:20: Section [korea:childrun] has
unknown type: childrun
[WARNING]: Unable to parse /home/ansible-user/ansible/inventory as an inventory source
[WARNING]: No inventory was parsed, only implicit localhost is available

PLAY [Handlers Example] ********************************************************************************************************************************

TASK [Install httpd latest version] ********************************************************************************************************************
changed: [serverb]

RUNNING HANDLER [restart_httpd] ************************************************************************************************************************
changed: [serverb]

PLAY RECAP *********************************************************************************************************************************************
serverb                    : ok=2    changed=2    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0

[ansible-user@controller ansible]$ ansible serverb -m shell -a 'systemctl is-active httpd'
[WARNING]:  * Failed to parse /home/ansible-user/ansible/inventory with yaml plugin: YAML inventory has invalid structure, it should be a dictionary, got:
<class 'ansible.parsing.yaml.objects.AnsibleUnicode'>
[WARNING]:  * Failed to parse /home/ansible-user/ansible/inventory with ini plugin: /home/ansible-user/ansible/inventory:20: Section [korea:childrun] has
unknown type: childrun
[WARNING]: Unable to parse /home/ansible-user/ansible/inventory as an inventory source
[WARNING]: No inventory was parsed, only implicit localhost is available
serverb | CHANGED | rc=0 >>
active
```

- notify와 handler 2

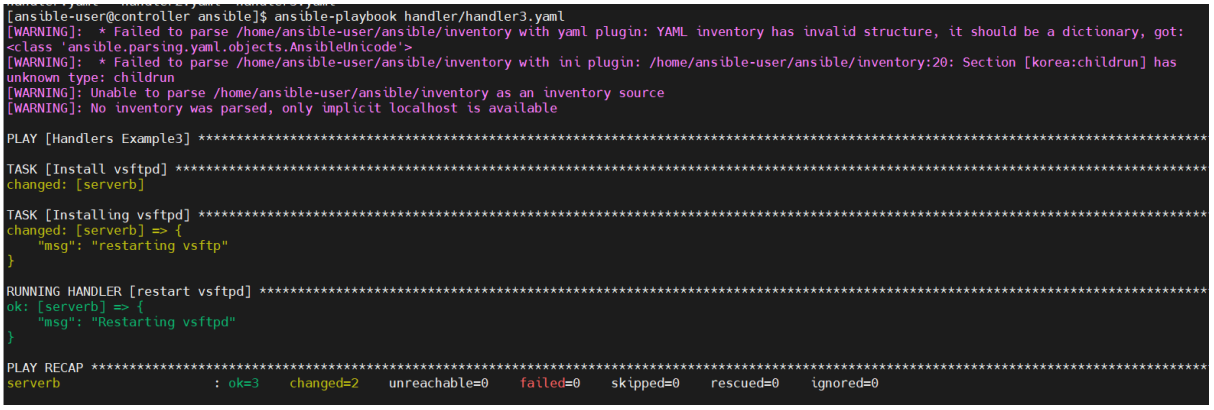
```yaml
# handler2.yaml
---
- name: Handlers Example2
  hosts: serverb
  gather_facts: true
  become: true
  tasks:
    - name: print message-1
      debug:
        msg: "First Message"
      changed_when: true
      notify: run_handler

    - name: print message-2
      debug:
        msg: "Second Message"
      changed_when: true
      notify: run_handler

  handlers:
    - name: run_handler  # 두번의 notify를 받았지만 한번만 실행
      debug:
        msg: "Today's date and time: {{ '%d-%m-%Y %H:%M:%S' | strftime }}"
```

- 결과

- notify와 handler 3

```yaml
# handler3.yaml
---
- name: Handlers Example3
  hosts: serverb
  gather_facts: false
  become: true
  vars:
    pkg: vsftpd
  tasks:
    - name: Install vsftpd
      yum:
        name: "{{ pkg }}"
        state: latest

    - name: Installing vsftpd
      debug:
        msg: "restarting vsftp"
      changed_when: true                # notify를 전달하기 위해 강제로 changed 보고
      notify: restart vsftpd

  handlers:
    - name: "restart {{ pkg | default('vsftpd')}}"
      debug:
        msg: "Restarting {{ pkg | default('vsftpd')}}"
```

- 결과



작업 오류 제어

Ansible은 각 작업의 반환 코드를 평가하여 작업의 성공 여부를 판단합니다. 일반적으로 작업이 실패하면 Ansible은 바로 이후의 모든 작업을 건너뜁니다. 하지만 작업이 실패해도 플레이를 계속 실행할 수 있습니다. 예를 들면, 특정 작업이 실패할 것으로 예상하고 몇 가지 다른 작업을 조건부로 실행하여 복구하려고 할 수 있습니다. 다양한 Ansible 기능을 사용하여 작업 오류를 관리할 수 있습니다.

ignore_errors 속성은 작업이 실패하더라도 중단하지 않고 계속 플레이를 진행하게 합니다.

```
- yum:
    name: http
    state: latest
  ignore_errors: yes
```

특정 작업에서 에러가 발생하면 플레이가 중단되어 이전에 알림을 받았던 핸들러는 모두 동작하지 않게 됩니다. 이때 플레이는 중단되더라도 알림을 받은 핸들러는 모두 동작하도록 구성하는 ***force_handlers*** 속성이 있습니다.

```
- hosts: all
  force_handlers: yes
  tasks:
....(생략)....
```

때로는 원하는 결과가 나오지 않더라도 작업에서 정상적으로 넘어가는 것 처럼 보이는 경우가 있습니다. 이때는 ***failed_when*** 속성을 지정하여 작업은 성공되더라도 에러가 발생되게 강제화 할 수 있습니다. 예를 들면 스크립트를 실행하는데 스크립트 동작은 완료 되었지만 실패로 간주되는 상황이 발생 할 수 있습니다.

```
tasks:
  - name: run script
    shell: /usr/local/bin/create_users.sh
```

```
    register: command_result
    failed_when: "'password missing' in command_result.stdout"
```

혹은 fail 모듈을 통해 오류를 생성하고 실패 메세지를 입력할 수도 있습니다.

```
tasks:
  - name: run script
    shell: /usr/local/bin/create_users.sh
    register: command_result
    ignore_errors: yes
  - name: fail message
    fail:
      msg: "Passowrd missing"
    when: "'password missing' in command_result.stdout"
```

플레이북에서 block을 사용하여 작업을 논리적인 그룹으로 만들 수 있으며 여러 작업에 조건을 적용하는 단일 ***when*** 속성을 지정할 수 있습니다.

```
tasks:
  - name: install and configuration versionlock
    block:
    - name: package install
        yum:
          name: python3-dnf-plugin-versionlock
          state: present
    - name: config lock
        lineinfile:
          dest: /etc/yum/pluginconf.d/versionlock.list
          line: tzdata-2016j-1
          state: present
     when: ansible_distribution == "RedHat"
```

그리고 block 절 문제가 발생하면 조치를 위한 작업도 추가할 수 있습니다. ***block*** 절과 함께 사용하는 ***rescue*** 절을 사용하여 오류에 대한 작업 내용을 정의 하거나 혹은 ***always*** 절을 통해 실패 유무와는 상관없이 동일하게 실행하는 작업 내용도 정의 할 수 있습니다.

```
tasks:
  - name: Upgrade DB
    block:
      - name: upgrade the database
        shell:
          cmd: /usr/local/lib/upgrade-database
    rescue:
      - name: revert the database upgrade
        shell:
          cmd: /usr/local/lib/revert-database
```

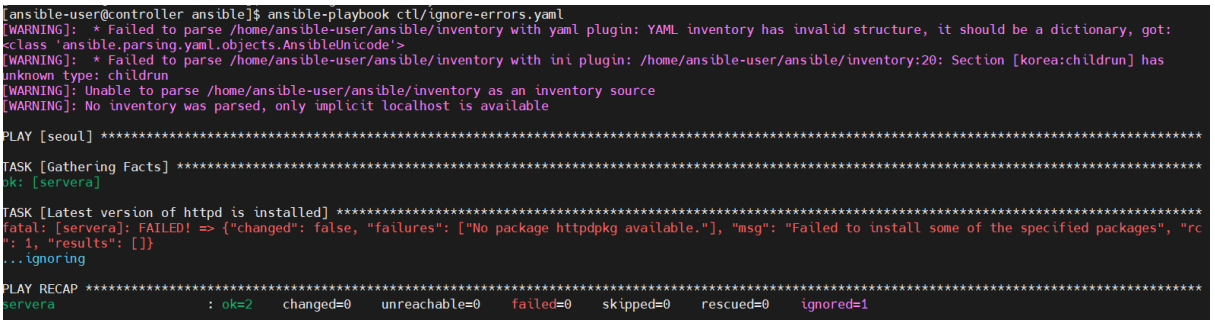
```
    always:
      - name: always restart the database
        service:
          name: mariadb
          state: restarted
```

작업 오류 제어 실습

- 실습 1 ignore-error

```
# ignore_errors.yaml
- hosts: seoul
  force_handlers: yes
  tasks:
    - name: Latest version of httpd is installed
      yum:
        name: httpdpkg
        state: latest
      ignore_errors: yes
```

- 결과



에러 발생하면 무시하고 플레이 계속 진행

- 실습 2 force_handlers

```
# force_handlers.yaml
- hosts: seoul
  force_handlers: yes
  tasks:
    - name: a task to notify handler
      command: /bin/true
      notify: restart mariadb
```

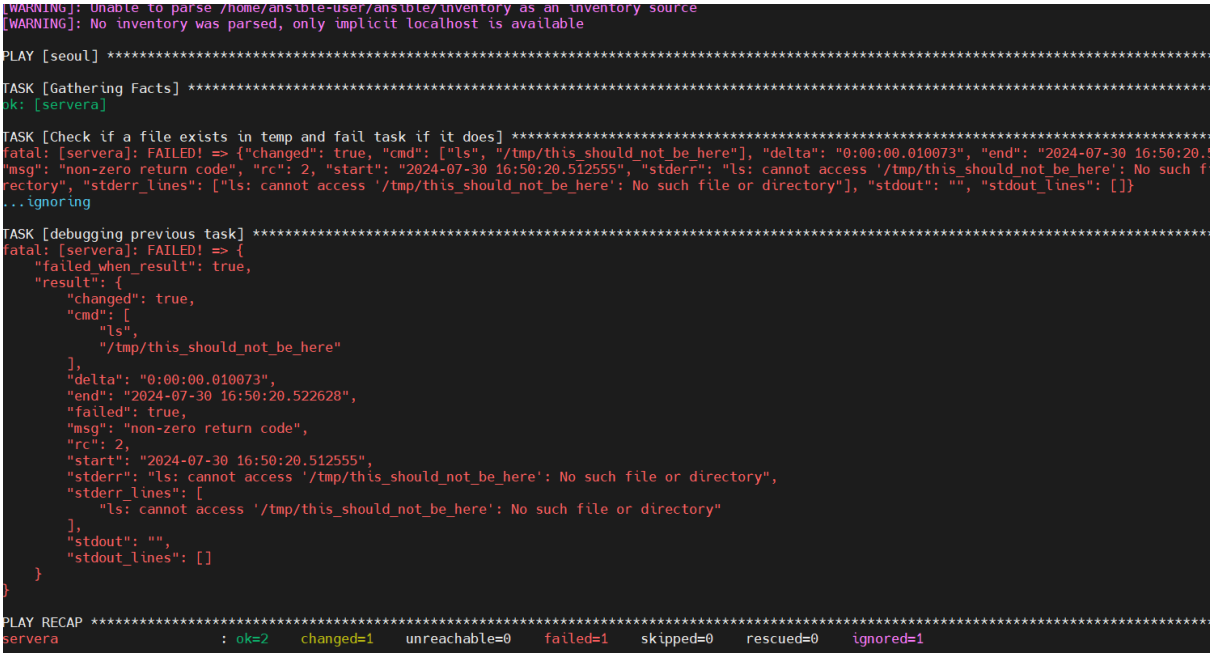
```
    - name: a failed task if the package doesn't exist
      yum:
        name: httpdpkg
        state: latest

  handlers:
    - name: restart mariadb
      service:
        name: mariadb
        state: restarted
```

에러 발생해서 플레이가 중단되어도 핸들러 동작하게끔

- 실습3 failed when

```
# failed_when.yaml
- hosts: seoul
  tasks:
    - name: Check if a file exists in temp and fail task if it does
      command: ls /tmp/this_should_not_be_here
      register: result
      ignore_errors: yes
    - name: debugging previous task
      debug:
        var: result
      failed_when: result.rc == 2
```

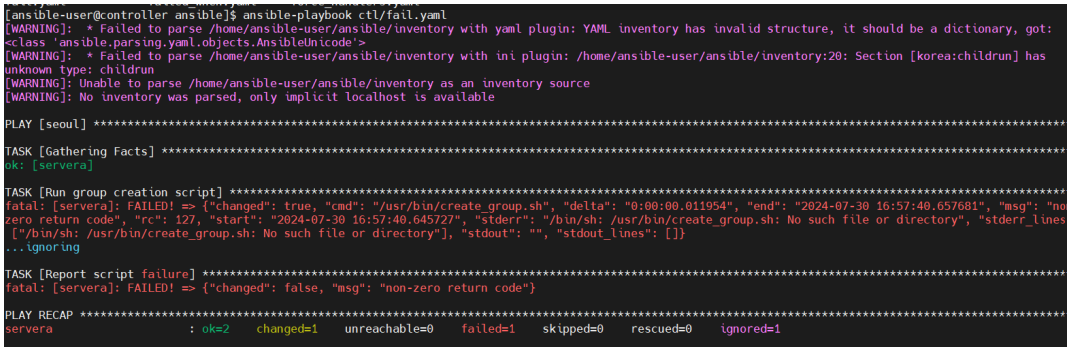
- 결과



- 실습4 fail

```
# fail.yaml
- hosts: seoul
  tasks:
    - name: Run group creation script
      shell: /usr/bin/create_group.sh
      register: result
      ignore_errors: yes

    - name: Report script failure
      fail:
        msg: "non-zero return code"
      when: result.rc == 127
```

- 결과

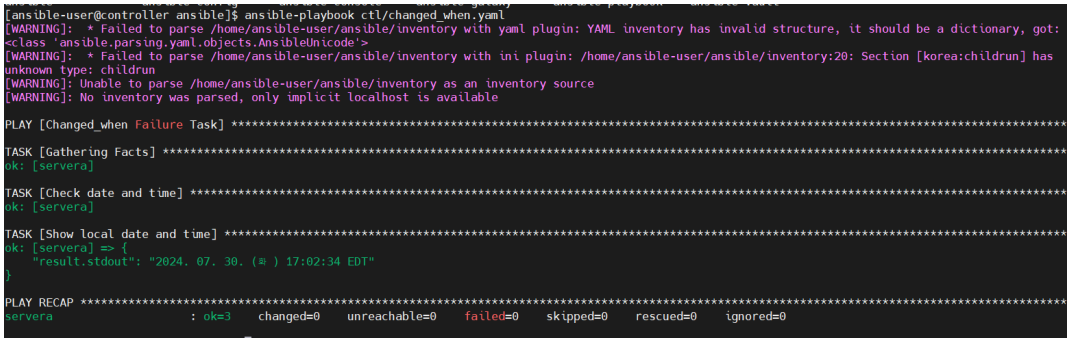


- 실습5 changed when

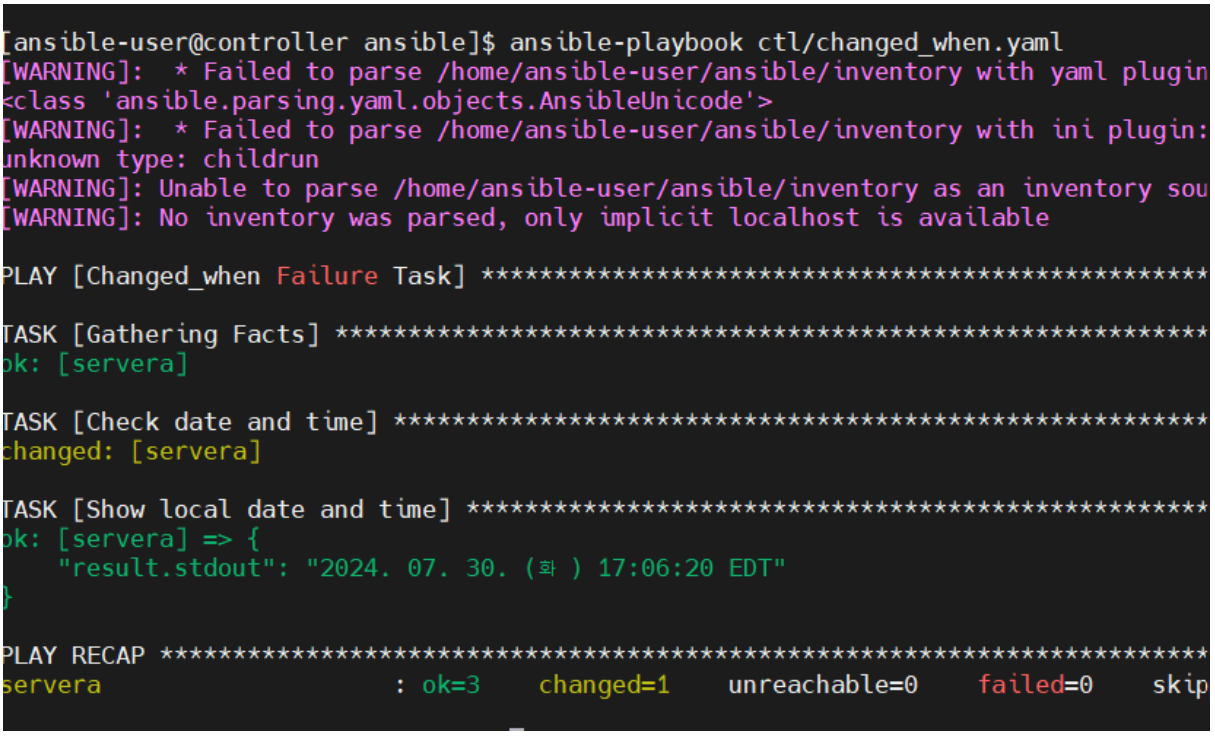
```yaml
# changed_when.yaml
- name: Changed_when Failure Task
  hosts: seoul
  tasks:
    - name: Check date and time
      command: date
      register: result
      changed_when: false # 이 부분 추가
    - name: Show local date and time
      debug:
        var: result.stdout
```

변경사항 보고 x false로 지정한다.

- 보고 안한 결과



- 보고한 결과

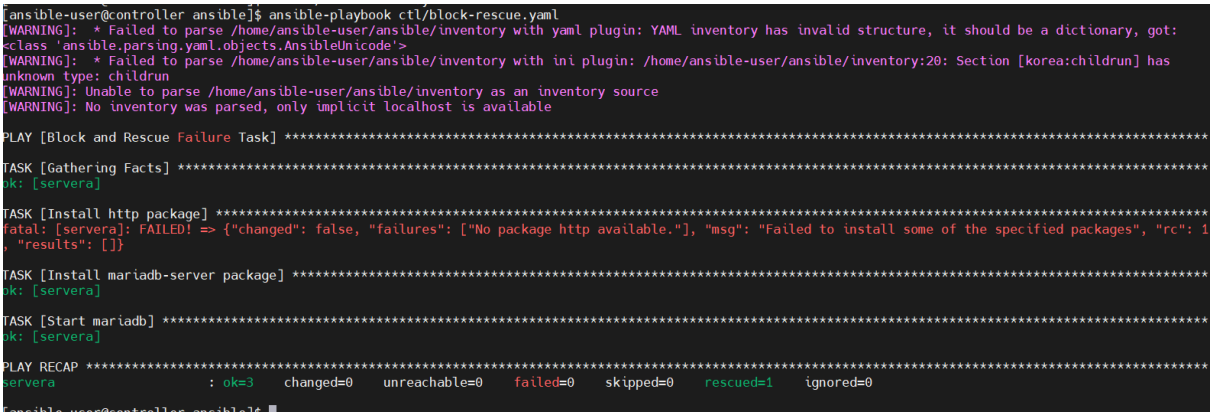


changed=1 확인 가능하다.

- 실습 7 block , rescue, always

```
# block_rescue.yaml
- name: Block and Rescue Failure Task
  hosts: seoul
  vars:
    http_package: http
    db_package: mariadb-server
    db_service: mariadb
  tasks:
    - name: Check http and db server installed
      block:
        - name: Install {{ http_package }} package
          yum:
            name: "{{ http_package }}"
            state: present
      rescue:
        - name: Install {{ db_package }} package
          yum:
            name: "{{ db_package }}"
            state: present
      always:
        - name: Start {{ db_service }}
          service:
            name: "{{ db_service }}"
            state: started
```

- 결과



block문이 실패하자 rescue문 실행하고 always는 항상 결과와 상관없이 실행하는 것을 확인 가능하다.