# Ansible_Syntax

# Variable & Facts

- 변수의 종류 와 사용법
  - ✓ 개요
    - 앤서블은 변수를 사용하여 사용자, 설치하고자 하는 패키지, 재시작 할 서비스, 생성 또는 삭제 할 파일명 등 시스템 작업 시 사용되는 다양한 값을 저장할 수 있음
    - 변수를 활용하면 플레이북을 재사용할 수 있으며 사용자로부터 받은 값도 쉽게 적용할 수 있음
    - 앤서블에서 사용되는 변수는 그룹 변수, 호스트 변수, 플레이 변수, 추가 변수가 있으며 플레이 결과를 저장하기 위한 작업 변수도 있음
  - ✓ 그룹 변수
    - 그룹 변수는 인벤토리에 정의된 호스트 그룹에 적용하는 변수를 의미
    - 인벤토리에 선언해야 하고 선언하고자 하는 그룹명과 함께 :vars라는 문자열을 추가해 변수를 선언한다는 것을 알려줌

# Variable & Facts

- ❖ 변수의 종류 와 사용법
  - ✓ 그룹 변수
    - ● 실습
      - ◆ 실행: ansible-playbook create-user.yml

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

TASK [Gathering Facts]
*****************************************************************

ok: [master]
ok: [worker1]
ok: [worker2]
TASK [Create User ansible]
***********************************************************

changed: [worker2]
changed: [worker1]
changed: [master]
PLAY RECAP
**********************************************************************************
master                     : ok=2    changed=1    unreachable=0    failed=0
skipped=0    rescued=0    ignored=0
worker1                    : ok=2    changed=1    unreachable=0    failed=0
skipped=0    rescued=0    ignored=0
worker2                    : ok=2    changed=1    unreachable=0    failed=0
skipped=0    rescued=0    ignored=0
```

# Variable & Facts

- ❖ 변수의 종류 와 사용법
  - ✓ 그룹 변수
    - ● 실습
      - ◆ 관리 노드에서 확인: ll /home

```
total 16
drwxr-xr-x  4 root    root    4096 Mar 31 23:05 ./
drwxr-xr-x 23 root    root    4096 Feb 10 02:02 ../
drwxr-x---  5 adam    adam    4096 Mar 31 07:46 adam/
drwxr-x---  2 ansible ansible 4096 Mar 31 23:05 ansible/
```

# Variable & Facts

- ❖ 변수의 종류 와 사용법
  - ✓ 호스트 변수
    - 해당 호스트에서만 사용할 수 있는 변수
    - 호스트 옆에 선언

```
[itstudy]
master
worker1 user=ansible1
worker2

[itstudy:vars]
user=ansible
```

# Variable & Facts

- ❖ 변수의 종류 와 사용법
  - ✓ 플레이 변수
    - ● 플레이북 내에서 선언되는 변수

```
---
- hosts: all
  vars:
    user: ansible2
  tasks:
  - name: Create User {{ user }}
    ansible.builtin.user:
      name: "{{ user }}"
      state: present
```

# Variable & Facts

- 변수의 종류 와 사용법
  - ✓ 추가 변수
    - 외부에서 ansible-playbook를 실행할 때 함께 파라미터로 넘겨주는 변수
    - 추가 변수는 우선순위가 가장 높음

      ansible-playbook -e user=ansible5 create-user4.yml
    - 외부 파일에 변수를 생성하고 읽어와서 실행하는 것도 가능
      - vars 디렉토리에 users.yml 파일 생성

        user: ansible4
      - vars 디렉토리에 users.yml 파일 생성

        user: ansible4
      - playbook

```
---
- hosts: all
  vars_files:
    - vars/users.yml
  tasks:
  - name: Create User {{ user }}
    ansible.builtin.user:
      name: "{{ user }}"
      state: present
```

# Variable & Facts

- 변수의 종류 와 사용법
  - ✓ 추가 변수
    - 외부에서 ansible-playbook를 실행할 때 함께 파라미터로 넘겨주는 변수
    - 추가 변수는 우선순위가 가장 높음

      ansible-playbook -e user=ansible5 create-user4.yml
    - 외부 파일에 변수를 생성하고 읽어와서 실행하는 것도 가능
      - vars 디렉토리에 users.yml 파일 생성

        user: ansible4
      - vars 디렉토리에 users.yml 파일 생성

        user: ansible4
      - playbook

```
---
- hosts: all
  vars_files:
    - vars/users.yml
  tasks:
  - name: Create User {{ user }}
    ansible.builtin.user:
      name: "{{ user }}"
      state: present
```

# Variable & Facts

- 변수의 종류 와 사용법
  - ✓ 작업 변수
    - 작업 변수는 플레이북의 태스크 수행 결과를 저장한 것
    - 특정 작업 수행 후 그 결과를 후속 작업에서 사용할 때 주로 사용
    - 클라우드 시스템에 VM을 생성한다고 가정하면 이를 위해서는 네트워크나 운영체제 이미지와 같은 가상 자원이 필요하며 가상 자원을 조회하고 조회된 결과를 가지고 VM 을 생성할 때는 작업 변수를 사용하면 편리

# Variable & Facts

- ❖ 변수의 종류 와 사용법
  - ✓ 작업 변수
    - ● 실습: 유저를 생성한 결과를 저장해서 debug 모듈을 이용해서 출력
      - ◆ Playbook: create-user-result.yaml

```
---
- hosts: all
  tasks:
  - name: Create User {{ user }}
    ansible.builtin.user:
      name: "{{ user }}"
      state: present
    register: result
    become: yes
  - ansible.builtin.debug:
      var: result
```

# Variable & Facts

- ❖ 변수의 종류 와 사용법
  - ✓ 작업 변수
    - ● 실습: 유저를 생성한 결과를 저장해서 debug 모듈을 이용해서 출력
      - ◆ 실행: ansible-playbook -e user=ansibleresult create-user-result.yml

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

TASK [Gathering Facts]
    ********************************************************************
ok: [worker2]
ok: [master]
ok: [worker1]

TASK [Create User ansibleresult]
    ***********************************************************
changed: [master]
changed: [worker1]
changed: [worker2]
```

# Variable & Facts

- ❖ 패스워드를 안전하게 보관하기 위한 Ansible Vault
  - ✓ 개요
    - 앤서블을 사용할 때 패스워드나 API 키 등 중요한 데이터에 대한 액세스 권한이 필요할 수 있는데 이런 정보들은 인벤토리 변수나 일반 앤서블 플레이북에 텍스트로 저
    - 앤서블 파일에 접근 권한이 있는 사용자라면 모두 파일 내용을 볼 수 있는데 이런 파일은 외부로 유출될 수 있다는 보안상의 위험을 야기하게 됨
    - 앤서블은 사용되는 모든 데이터 파일을 암호화하고 암호화된 파일의 내용을 해독할 수 있는 Ansible Vault라는 기능을 제공

# Variable & Facts

- 패스워드를 안전하게 보관하기 위한 Ansible Vault
  - ✓ 암호화된 파일 만들기
    - 디렉토리 생성: my-ansible
    - 생성한 디렉토리로 프롬프트 이동
    - 파일을 확인할 때 사용할 비밀번호를 입력해서 생성: ansible-vault create mysecret.yml

      user: itstudy

      password: wnddkd
    - 파일의 접근 권한 확인: ll mysecret.yml

      -rw------- 1 adam adam 419 Mar 31 23:39 mysecret.yml
    - 파일 내용 확인: cat mysecret.yml

      $ANSIBLE_VAULT;1.1;AES256
      34363332393337633761336162633331343864363866323365376435316463316339363231373636
      62313538323231663664346435313231613366346239313 80a34623261653430356
      4616566333438
      38666339653132333731626132613637303966386163373461376130353265323064303732643766
      34306134613733646 60a30613333333439363766326535331613938353634616530393362303065
      33623130626234393838303030323864643065666233646431346335363762353561

# Variable & Facts

- ❖ 패스워드를 안전하게 보관하기 위한 Ansible Vault
  - ✓ 암호화된 파일 만들기
    - ● 파일 원래 내용 확인: ansible-vault view  mysecret.yml

      Vault password:
      user: itstudy
      password: wnddkd

# Variable & Facts

- ❖ 패스워드를 안전하게 보관하기 위한 Ansible Vault
  - ✓ 파일을 이용한 암호화된 파일 만들기
    - 비밀번호를 저장할 파일 생성: vault-pass
    - 파일 생성: ansible-vault create --vault-pass-file ./vault-pass mysecret1.yml
    - 파일 확인: ansible-vault view --vault-pass-file ./vault-pass mysecret1.yml

# Variable & Facts

- 패스워드를 안전하게 보관하기 위한 Ansible Vault
  - ✓ 기존 파일 암호화
    - ansible-vault encrypt 명령어를 이용하면 기존 파일을 암호화하고 해당 파일을 다시 복호화할 수 있음
    - 암호화: ansible-vault encrypt create-user.yml

      New Vault password:
      Confirm New Vault password:
      Encryption successful
    - 확인: nano create-user.yml

      $ANSIBLE_VAULT;1.1;AES256
      34616432396137626238613832313163333232663531653963623035623465666435363664343066
      61356661363563646638313234323663366661366162333300a6561626464666661356664316633326339653962326661356638643539643165356233653930636564353764323934643132633539313035663238383032306406261643231326233336334646561353939653663363565303866313339646462346630663337353434386534646434653063633231633161616361636132656561333386161316131

# Variable & Facts

- 패스워드를 안전하게 보관하기 위한 Ansible Vault
  - ✓ 기존 파일 암호화
    - 복호화: ansible-vault decrypt create-user.yml

      Vault password:
      Decryption successful
    - 확인: nano create-user.yml

```
---
- hosts: all
  tasks:
  - name: Create User {{ user }}
    ansible.builtin.user:
      name: "{{ user }}"
      state: present
    become: yes
```

# Variable & Facts

- 패스워드를 안전하게 보관하기 위한 Ansible Vault
  - ✓ 암호화된 playbook 실행
    - 암호화된 파일을 vars 디렉토리로 이동: mkdir vars && mv mysecret.yml vars/
    - 암호화된 파일 확인: ansible-vault view vars/mysecret.yml

      user: itstudy

      password: wnddkd
    - 유저 생성을 위한 playbook 파일: create-user.yml

```
---
- hosts: all
  vars_files:
    - vars/mysecret.yml
  tasks:
  - name: Create User {{ user }}
    ansible.builtin.user:
      name: "{{ user }}"
      state: present
    become: yes
```

    - 에러가 발생하는 실행: ansible-playbook create-user.yml

      ERROR! Attempting to decrypt but no vault secrets found

# Variable & Facts

- 패스워드를 안전하게 보관하기 위한 Ansible Vault
  - ✓ 암호화된 playbook 실행
    - 정상 실행: ansible-playbook --vault-id @prompt create-user.yml

```
PLAY [all]
*********************************************************************************
TASK [Gathering Facts]
*********************************************************************
ok: [worker1]
ok: [worker2]
ok: [master]
TASK [Create User itstudy]
*************************************************************
changed: [worker2]
changed: [master]
changed: [worker1]
PLAY RECAP
*********************************************************************************
master                     : ok=2    changed=1    unreachable=0    failed=0
skipped=0    rescued=0    ignored=0
worker1                    : ok=2    changed=1    unreachable=0    failed=0
skipped=0    rescued=0    ignored=0
worker2                    : ok=2    changed=1    unreachable=0    failed=0
skipped=0    rescued=0    ignored=0
```

# Variable & Facts

- Facts
  - ✓ 개요
    - Facts는 앤서블이 관리 호스트에서 자동으로 검색한 변수
    - 팩트에는 플레이, 조건문, 반복문 또는 관리 호스트에서 수집한 값에 의존하는 기타 명령문의 일반 변수처 럼 사용 가능한 호스트별 정보가 포함되어 있음
    - 관리 호스트에서 수집된 일부 팩트에는 다음 내용들이 포함될 수 있음
      - 호스트 이름
      - 커널 버전
      - 네트워크 인터페이스 이름
      - 운영체제 버전
      - CPU 개수
      - 사용 가능한 메모리
      - 스토리지 장치의 크기 및 여유 공간
    - 팩트에 의해 수집된 변수 값을 이용하여 서비스 상태를 확인하고 나면 다음 작업 진행 여부를 판단할 수 있음

# Variable & Facts

- Facts
  - ✓ 팩트 사용
    - 모든 Fact 출력: facts.yml

```
---
- hosts: all

  tasks:
  - name: Print all facts
    ansible.builtin.debug:
      var: ansible_facts
```

# Variable & Facts

- ❖ Facts
  - ✓ 팩트 사용
    - ● 특정 Fact 출력: facts1.yml

```
---
- hosts: all

  tasks:
  - name: Print facts
    ansible.builtin.debug:
      msg: >
        The default IPv4 address of {{ ansible_facts.fqdn }}
        is {{ ansible_facts.default_ipv4.address }}
```

# Variable & Facts

- Facts
  - ✓ 변수로 사용 가능한 facts

| 팩트 | ansible_facts.* 표기법 |
|---|---|
| 호스트명 | ansible_facts.hostname |
| 도메인 기반 호스트명 | ansible_facts.fqdn |
| 기본 IPv4 주소 | ansible_facts.defatult_ipv4.address |
| 네트워크 인터페이스 이름 목록 | ansible_facts.interfaces |
| /dev/vda1 디스크 파티션 크기 | ansible_facts.device.vda.partitions.vda1.size |
| DNS 서버 목록 | ansible_facts.dns.nameservers |
| 현재 실행중인 커널 버전 | ansible_facts.kernel |
| 운영체제 종류 | ansible_facts.distribution |

# Variable & Facts

- ❖ Facts
  - ✓ 변수로 사용 가능한 facts
    - ● 특정 Fact 출력: facts2.yml

```
---
- hosts: all

  tasks:
  - name: Print facts
    ansible.builtin.debug:
      msg: >
        This node's host name is {{ ansible_hostname }}
        and the ip is {{ ansible_default_ipv4.address }}
```

# Variable & Facts

- Facts
  - ✓ 사용자 지정 Fact
    - /etc/ansible/facts.d 디렉토리를 생성하고 my-custom.fact 파일을 생성하고 작성

```
[packages]
web_package = httpd
db_package = mariadb-server
[users]
userl = ansible
user2 = terraform
```

    - facts3.yml 파일을 작성

```
---
- hosts: localhost

  tasks:
  - name: Print local facts
    ansible.builtin.debug:
      var: ansible_local
```

# Variable & Facts

- Facts
  - ✓ 사용자 지정 Fact
    - 실행

```
PLAY [localhost]
*********************************************************************

TASK [Gathering Facts]
*********************************************************************
ok: [localhost]
```

# Variable & Facts

- Facts
  - ✓ 사용자 지정 Fact
    - 실행

```
TASK [Print local facts]
*********************************************************************
ok: [localhost] => {
    "ansible_local": {
        "my-custom": {
            "packages": {
                "db_package": "mariadb-server",
                "web_package": "httpd"
            },
            "users": {
                "user2": "terraform",
                "userl": "ansible"
            }
        }
    }
}

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

# 제어문

- ❖ 반복문 - loop
  - ✓ 단순 반복문
    - ● 반복문을 사용하지 않는 check-service.yml

```
---
- hosts: all
  tasks:
    - name: Check sshd state
      ansible.builtin.service:
        name: ssh
        state: started
      become: yes
    - name: Check rsyslog state
      ansible.builtin.service:
        name: rsyslog
        state: started
      become: yes
```

# 제어문

- ❖ 반복문 - loop
  - ✓ 단순 반복문
    - ● 반복문을 사용하는 check-service.yml

```
---
- hosts: all
  tasks:
    - name: Check sshd and rsyslog state
      ansible.builtin.service:
        name: "{{ item }}"
        state: started
      loop:
        - ssh
        - rsyslog
```

# 제어문

- ❖ 반복문 - loop
  - ✓ 단순 반복문
    - ● 변수에 저장해서 반복문을 사용하는 check-service.yml

```
---
- hosts: all
  vars:
    services:
      - ssh
      - rsyslog

  tasks:
    - name: Check sshd and rsyslog state
      ansible.builtin.service:
        name: "{{ item }}"
        state: started
      loop: "{{ services }}"
```

# 제어문

- ❖ 반복문 - loop
  - ✓ 사전 목록에 의한 반복문
    - ● 여러 개의 변수를 이용하는 반복문을 사용하는 make-file.yml

```
---
- hosts: all

  tasks:
    - name: Create files
      ansible.builtin.file:
        path: "{{ item['log-path'] }}"
        mode: "{{ item['log-mode'] }}"
        state: touch
      loop:
        - log-path: /var/log/test1.log
          log-mode: '0644'
        - log-path: /var/log/test2.log
          log-mode: '0600'
      become: yes
```

# 제어문

- ❖ 반복문 - loop
  - ✓ 반복문과 Register 변수 사용
    - Register 변수는 반복 실행되는 작업의 출력을 캡처할 수 있는데 작업들이 모두 잘 수행되었는지 확인할 수 있으며 이를 통해 반복 실행되는 이 값을 이용하여 다음 작업을 수행할 수 있음
    - 작업의 출력을 캡쳐해서 출력하는 loop_register.yml

```
---
- hosts: localhost
  tasks:
    - name: Loop echo test
      ansible.builtin.shell: "echo 'I can speak {{ item }}'"
      loop:
        - Korean
        - English
      register: result

    - name: Show result
      ansible.builtin.debug:
        var: result
```

# 제어문

- ❖ 반복문 - loop
  - ✓ 반복문과 Register 변수 사용
    - ● 작업의 출력을 캡쳐해서 출력하는 loop_register.yml

```
---
- hosts: localhost
  tasks:
    - name: Loop echo test
      ansible.builtin.shell: "echo 'I can speak {{ item }}'"
      loop:
        - Korean
        - English
      register: result

    - name: Show result
      ansible.builtin.debug:
        msg: "Stdout: {{ item.stdout }}"
      loop: "{{ result.results }}"
```

# 제어문

- ❖ 조건문 – when
  - ✓ 조건 작업 구문
    - ● 조건을 확인하기 위한 when_task.yml

```
---
- hosts: localhost
  vars:
    run_my_task: true

  tasks:
    - name: echo message
      ansible.builtin.shell: "echo test"
      when: run_my_task
```

# 제어문

- 조건문 – when
  - ✓ 조건 연산자
    - 연산 예시

| | |
|---|---|
| ansible_facts['machine'] == "x86_64" | ansible_facts['machine']의 값이 "x86_64"와 같으면 true |
| max_memory == 512 | max_memory 값이 512와 같으면 true |
| min_memory < 128 | min_memory 값이 128보다 작으면 true |
| min_memory > 256 | min_memory 값이 256보다 크면 true |
| min_memory <= 256 | min_memory 값이 256보다 작거나 같으면 true |
| min_memory >= 512 | min_memory 값이 512보다 크거나 같으면 true |
| min_memory != 512 | min_memory 값이 512와 같지 않으면 true |
| min_memory is defined | min_memory라는 변수가 있으면 true |
| min_memory is not defined | min_memory라는 변수가 없으면 true |
| memory_available | memory 값이 true이면 true. 이때 해당 값이 1이거나 True 또는 yes면 true |
| not memory_available | memory 값이 false면 true. 이때 해당 값이 0이거나 false 또는 no이면 true |
| ansible_facts['distribution'] in supported_distros | Ansible_facts['distribution']의 값이 supported_distros라는 변수에 있으면 true |

# 제어문

- ❖ 조건문 – when
  - ✓ 조건 연산자
    - ● 운영체제를 확인하는 check-os.yml

```
---
- hosts: all
  vars:
    supported_distros:
      - RedHat
      - CentOS

  tasks:
    - name: Print supported os
      ansible.builtin.debug:
        msg: "This {{ ansible_facts['distribution'] }} need to use dnf"
      when: ansible_facts['distribution'] in supported_distros
```

# 제어문

- ❖ 조건문 – when
  - ✓ 조건 연산자
    - ● 여러 개의 조건을 연결하는 운영체제를 확인하는 check-os.yml

```
---
- hosts: all


  tasks:
    - name: Print os type
      ansible.builtin.debug:
        msg: "OS Type: {{ ansible_facts['distribution'] }}"
      when: ansible_facts['distribution'] == "CentOS" or ansible_facts['distribution']
     =>
```

# 제어문

- ❖ 조건문 – when
  - ✓ 조건 연산자
    - ● 여러 개의 조건을 연결하는 운영체제를 확인하는 check-os.yml

```
---
- hosts: all


  tasks:
    - name: Print os type
      ansible.builtin.debug:
        msg: "OS Type: {{ ansible_facts['distribution'] }}"
      when: ansible_facts['distribution'] == "CentOS" or ansible_facts['distribution']
     =>
```

# 제어문

- ❖ 조건문 – when
  - ✓ 조건 연산자
    - ● 여러 개의 조건을 연결하는 운영체제를 확인하는 check-os.yml

```
---
- hosts: all


  tasks:
    - name: Print os type
      ansible.builtin.debug:
        msg: >
            OS Type: {{ ansible_facts['distribution'] }}
            OS Version: {{ ansible_facts['distribution_version'] }}
      when: ansible_facts['distribution'] == "CentOS" and
     ansible_facts['distribution_ve>
```

# 제어문

- ❖ 조건문 – when
  - ✓ 조건 연산자
    - ● 여러 개의 조건을 연결하는 운영체제를 확인하는 check-os.yml

```
---

- hosts: all

  tasks:
    - name: Print os type
      ansible.builtin.debug:
        msg: >
            OS Type: {{ ansible_facts['distribution'] }}
            OS Version: {{ ansible_facts['distribution_version'] }}
      when:
        - ansible_facts['distribution'] == "CentOS"
        - ansible_facts['distribution_version'] == "8"
```

# 제어문

- ❖ 조건문 – when
  - ✓ 조건 연산자
    - ● 여러 개의 조건을 연결하는 운영체제를 확인하는 check-os.yml

```
---
- hosts: all
  tasks:
    - name: Print os type
      ansible.builtin.debug:
        msg: >
            OS Type: {{ ansible_facts['distribution'] }}
            OS Version: {{ ansible_facts['distribution_version'] }}
      when: >
          ( ansible_facts['distribution'] == "CentOS" and
            ansible_facts['distribution_version'] == "8" )
          or
          ( ansible_facts['distribution'] == "Ubuntu" and
            ansible_facts['distribution_version'] == "20.04" )
```

# 제어문

- ❖ 조건문 – when
  - ✓ 반복문 과 조건문 같이 사용
    - check-mount.yml

```
---
- hosts: all

  tasks:
    - name: Print Root Directory Size
      ansible.builtin.debug:
        msg: "Directory {{ item.mount }} size is {{ item.size_available }}"
      loop: "{{ ansible_facts['mounts'] }}"
      when: item['mount'] == "/" and item['size_available'] > 300000000
```

# 제어문

- ❖ 조건문 – when
  - ✓ 반복문 과 조건문 같이 사용
    - check-mount.yml

```
---
- hosts: all

  tasks:
    - name: Print Root Directory Size
      ansible.builtin.debug:
        msg: "Directory {{ item.mount }} size is {{ item.size_available }}"
      loop: "{{ ansible_facts['mounts'] }}"
      when: item['mount'] == "/" and item['size_available'] > 300000000
```

# 제어문

- ❖ 조건문 – when
  - ✓ register 키워드를 이용한 작업 변수 사용
    - register-when.yml

```
---
- hosts: all

  tasks:
    - name: Get rsyslog service status
      ansible.builtin.command: systemctl is-active rsyslog
      register: result

    - name: Print rsyslog status
      ansible.builtin.debug:
        msg: "Rsyslog status is {{ result.stdout }}"
      when: result.stdout == "active"
```

# 핸들러 및 작업 실패 처리

- 개요
  - ✓ 앤서블 모듈은 멱등이 가능하도록 설계되어 있음
  - ✓ 플레이북을 여러 번 실행해도 결과는 항상 동일
  - ✓ 플레이 및 해당 작업은 여러 번 실행할 수 있지만 해당 호스트는 원하는 상태로 만드는 데 필요한 경우에만 변경됨
  - ✓ 한 작업에서 시스템을 변경해야 하는 경우 추가 작업을 실행해야 할 수도 있는데 예를 들어 서비스의 구성 파일을 변경하려면 변경 내용이 적용되도록 서비스를 다시 로드해야 하는데 이때 핸들러는 다른 작업에서 트리거한 알림에 응답하는 작업이며 해당 호스트에서 작업이 변경될 때만 핸들러에 통지

# 핸들러 및 작업 실패 처리

- ❖ 앤서블 핸들러
  - ✓ 개요
    - 앤서블에서 핸들러를 사용하려면 notify 문을 사용하여 명시적으로 호출된 경우에만 사용할 수 있음
    - 핸들러를 정의할 때는 같은 이름으로 여러 개의 핸들러를 정의하기보다는 각각의 고유한 이름을 사용하여 정의하는 것이 좋음

# 핸들러 및 작업 실패 처리

- 앤서블 핸들러
  - ✓ master 노드의 rsyslog 서비스를 재시작하고 print msg라는 핸들러를 호출해 "rsyslog is restarted"라는 메시지를 출력
    - handler-sample.yml

```
---
- hosts: master
  tasks:
    - name: restart rsyslog
      ansible.builtin.service:
        name: "rsyslog"
        state: restarted
      notify:
        - print msg
      become: yes
  handlers:
    - name: print msg
      ansible.builtin.debug:
        msg: "rsyslog is restarted"
```

# 핸들러 및 작업 실패 처리

- 앤서블 핸들러
  - ✓ master 노드의 rsyslog 서비스를 재시작하고 print msg라는 핸들러를 호출해 "rsyslog is restarted"라는 메시지를 출력
    - 실행: ansible-playbook handler-sample.yml

```
PLAY [master]
**********************************************************************

TASK [Gathering Facts]
**********************************************************************
ok: [master]
TASK [restart rsyslog]
**********************************************************************
changed: [master]
RUNNING HANDLER [print msg]
**********************************************************************
ok: [master] => {
    "msg": "rsyslog is restarted"
}
PLAY RECAP
**********************************************************************
master                     : ok=3    changed=1    unreachable=0    failed=0
skipped=0    rescued=0    ignored=0
```

# 핸들러 및 작업 실패 처리

- 작업 실패 무시
  - ✓ 개요
    - 앤서블은 플레이 시 각 작업의 반환 코드를 평가하여 작업의 성공 여부를 판단
    - 작업이 실패하면 앤서블은 이후의 모든 작업을 건너뜀
    - 작업이 실패해도 플레이를 계속 실행할 수 있는데 이는 ignore_errors라는 키워드로 구현할 수 있음

# 핸들러 및 작업 실패 처리

- ❖ 작업 실패 무시
  - ✓ ubuntu에 없는 apache3 설치
    - ● 플레이북: ignore-example.yml

```
---
- hosts: master

  tasks:
    - name: Install apache3
      ansible.builtin.dnf:
        name: apache3
        state: latest
      ignore_errors: yes

    - name: Print msg
      ansible.builtin.debug:
        msg: "Before task is ignored"
```

# 핸들러 및 작업 실패 처리

- ❖ 작업 실패 무시
  - ✓ ubuntu에 없는 apache3 설치
    - ● 실행

```
PLAY [master]
*********************************************************************************
TASK [Gathering Facts]
*********************************************************************
ok: [master]
TASK [Install apache3]
*********************************************************************
fatal: [master]: FAILED! => {"ansible_facts": {"pkg_mgr": "apt"}, "changed": false,
"msg": ["Could not detect which major revision of dnf is in use, which is required
to determine module backend.", "You should manually specify use_backend to
tell the module whether to use the dnf4 or dnf5 backend})"]}
...ignoring
TASK [Print msg]
*********************************************************************************
ok: [master] => {
    "msg": "Before task is ignored"
}
PLAY RECAP
*********************************************************************************
master                     : ok=3    changed=0    unreachable=0    failed=0
skipped=0    rescued=0    ignored=1
```

# 핸들러 및 작업 실패 처리

- ❖ 작업 실패 후 핸들러 실행
  - ✓ 개요
    - 앤서블은 일반적으로 작업이 실패하고 해당 호스트에서 플레이가 중단되면 이전 작업에서 알림을 받은 모든 핸들러가 실행되지 않음
    - 플레이북에 force_handlers: yes 키워드를 설정하면 이후 작업이 실패하여 플레이가 중단되어도 알림을 받은 핸들러가 호출

# 핸들러 및 작업 실패 처리

- ❖ 작업 실패 후 핸들러 실행
  - ✓ 작업 실패 후 핸들러 실행
    - ● 플레이북: force-handler.yml

```
---
- hosts: master
  force_handlers: yes

  tasks:
    - name: restart rsyslog
      ansible.builtin.service:
        name: "rsyslog"
        state: restarted
      notify:
        - print msg
      become: yes
    - name: install apache3
      ansible.builtin.dnf:
        name: "apache3"
        state: lastest
  handlers:
    - name: print msg
      ansible.builtin.debug:
        msg: "rsyslog is restarted"
```

# 핸들러 및 작업 실패 처리

- ❖ 작업 실패 후 핸들러 실행
  - ✓ 작업 실패 후 핸들러 실행
    - ● 실행

```
PLAY [master]
*********************************************************************************

TASK [Gathering Facts]
*********************************************************************
ok: [master]
TASK [restart rsyslog]
*********************************************************************
changed: [master]
TASK [install apache3]
*********************************************************************
fatal: [master]: FAILED! => {"ansible_facts": {"pkg_mgr": "apt"}, "changed": false,
"msg": ["Could not detect which major revision of dnf is in use, which is required
to determine module backend.", "You should manually specify use_backend to
tell the module whether to use the dnf4 or dnf5 backend})"]}
RUNNING HANDLER [print msg]
*************************************************************
ok: [master] => {
    "msg": "rsyslog is restarted"
}
```

# 핸들러 및 작업 실패 처리

- ❖ 작업 실패 후 조건 지정: failed_when
  - ✓ 작업 실패
    - master node에 Shell Script 파일 생성 – adduser-script.sh

```
#!/bin/bash
# 사용자 계정 및 패스워드가 입력되었는지 확인
if [[ -n $1 ]] && [[ -n $2 ]]
then
  UserList=($1)
  Password=($2)
  # for문을 이용하여 사용자 계정 생성
  for (( i=0; i < ${#UserList[@]}; i++ ))
  do
    # if문을 사용하여 사용자 계정이 있는지 확인
    if [[ $(cat /etc/passwd | grep ${UserList[$i]} | wc -l) == 0 ]]
    then
      # 사용자 생성 및 패스워드 설정
      useradd ${UserList[$i]}
      echo ${Password[$i]} | passwd ${UserList[$i]} --stdin
    else
      # 사용자가 있다고 메시지를 보여줌.
      echo "this user ${UserList[$i]} is existing."
    fi
  done
```

# 핸들러 및 작업 실패 처리

- ❖ 작업 실패 후 조건 지정: failed_when
  - ✓ 작업 실패
    - master node에 Shell Script 파일 생성 – adduser-script.sh

```
else
  # 사용자 계정과 패스워드를 입력하라는 메시지를 보여줌.
  echo -e 'Please input user id and password.₩nUsage: adduser-script.sh "user01 user02" "pw01 pw02"'
fi
```

    - master node에 Shell Script 파일 권한 수정: chmod +x adduser-script.sh

# 핸들러 및 작업 실패 처리

- ❖ 작업 실패 후 조건 지정: failed_when
  - ✓ 작업 실패
    - control node에 playbook 생성: failed-when.yml

```
---
- hosts: master

  tasks:
    - name: Run user add script
      ansible.builtin.shell: /root/adduser-script.sh
      register: command_result
      failed_when: "'Please input user id and password' in command_result.stdout"

    - name: Print msg
      ansible.builtin.debug:
        msg: "This task is next task"
```

# 핸들러 및 작업 실패 처리

- ❖ 작업 실패 후 조건 지정: failed_when
  - ✓ 작업 실패
    - ● 실행

```
PLAY [master]
*********************************************************************

TASK [Gathering Facts]
*********************************************************************
ok: [worker1]

TASK [Run user add script]
*********************************************************************
fatal: [worker1]: FAILED! => {"changed": true, "cmd": "~/adduser-script.sh",
"delta": "0:00:00.002906", "end": "2026-04-01 23:22:11.702061",
"failed_when_result": true, "msg": "non-zero return code", "rc": 1, "start": "2026-
04-01 23:22:11.699155", "stderr": "", "stderr_lines": [], "stdout": "Please input user
id and password.₩nUsage: adduser-script.sh ₩"user01 user02₩" ₩"pass01
pass02₩"", "stdout_lines": ["Please input user id and password.", "Usage: adduser-
script.sh ₩"user01 user02₩" ₩"pass01 pass02₩""]}

PLAY RECAP
*********************************************************************
worker1                    : ok=1    changed=0    unreachable=0    failed=1
skipped=0    rescued=0    ignored=0
```

# 핸들러 및 작업 실패 처리

- ❖ 작업 실패 후 조건 지정: failed_when
  - ✓ 작업 실패
    - control node에 playbook 수정: failed-when.yml

```
---

- hosts: master

  tasks:
    - name: Run user add script
      ansible.builtin.shell: ~/adduser-script.sh
      register: command_result
      ignore_errors: yes

    - name: Report script failure
      ansible.builtin.fail:
        msg: "{{ command_result.stdout }}"
      when: "'Please input user id and password' in command_result.stdout"
```

# 핸들러 및 작업 실패 처리

- ❖ 작업 실패 후 조건 지정: failed_when
  - ✓ 작업 실패
    - ● 실행

```
PLAY [worker1]
*********************************************************************************

TASK [Gathering Facts]
*********************************************************************
ok: [worker1]
TASK [Run user add script]
*************************************************************
fatal: [worker1]: FAILED! => {"changed": true, "cmd": "~/adduser-script.sh",
"delta": "0:00:00.002671", "end": "2026-04-01 23:34:29.561157", "msg": "non-zero
return code", "rc": 1, "start": "2026-04-01 23:34:29.558486", "stderr": "",
"stderr_lines": [], "stdout": "Please input user id and password.₩nUsage: adduser-
script.sh ₩"user01 user02₩" ₩"pass01 pass02₩"", "stdout_lines": ["Please input
user id and password.", "Usage: adduser-script.sh ₩"user01 user02₩" ₩"pass01
pass02₩""]}
...ignoring
TASK [Report script failure]
*************************************************************
fatal: [worker1]: FAILED! => {"changed": false, "msg": "Please input user id and
password.₩nUsage: adduser-script.sh ₩"user01 user02₩" ₩"pass01 pass02₩""}
```