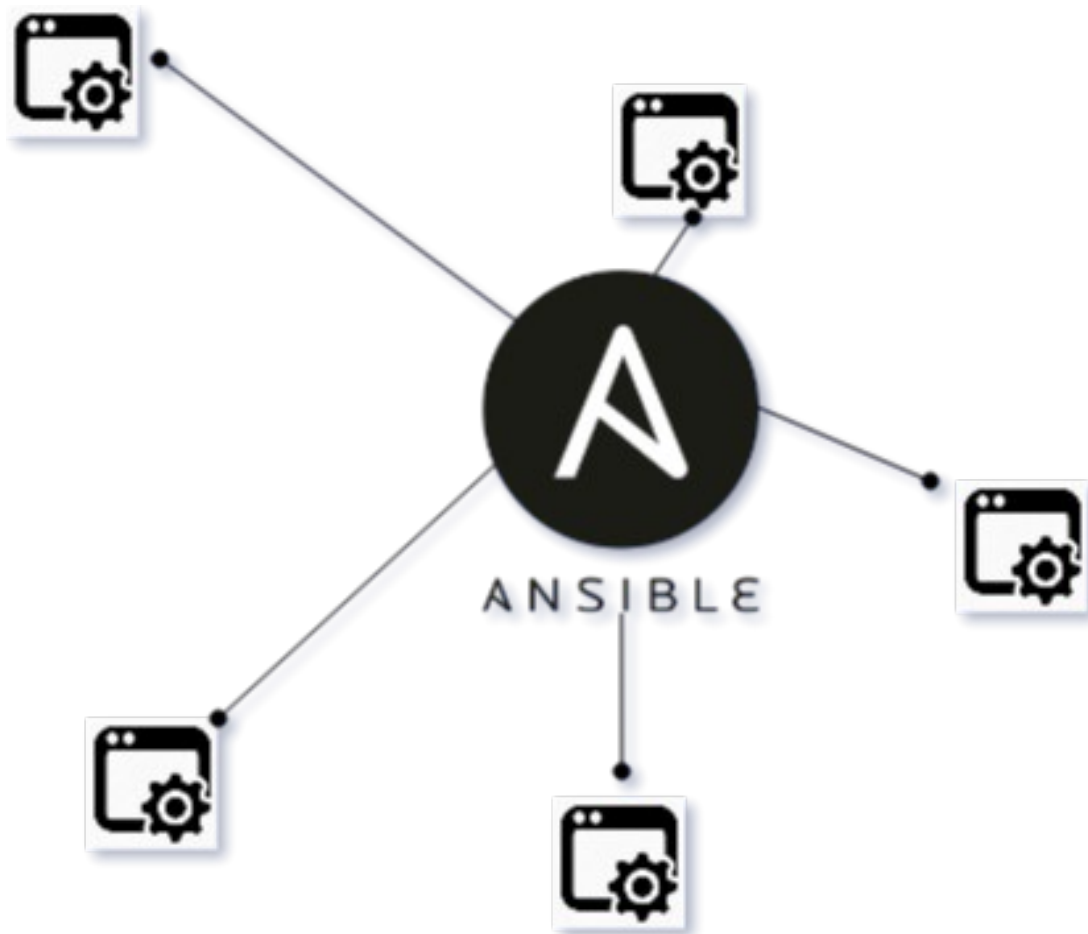


Ansible 과 AWX 를 활용한 애플리케이션 구축



이재원

목차

1. 개요

- 1) 고전적인 서버 인프라 구축
- 2) IaC 의 필요성
- 3) Ansible
- 4) AWX

2. 시스템 구성

- 1) Wordpress 구조
- 2) 시스템 구조
- 2) 네트워크 구조
- 3) SW 사양

3. Playbook 설명

- 1) roles 를 통한 playbook 구성
- 2) ansible.cfg
- 3) site.yaml
- 4) groupvars
- 5) Includes
- 6) templates
- 7) Tasks
- 8) Handler

4. AWX 설치

5. AWX 구성

- 1) Credential
- 2) Project
- 3) Inventory
- 4) Templates

6. 실행 결과

7. 결론

1. 개요

1) 고전적인 서버 인프라 구축

수동으로 일일이 서버에 애플리케이션을 배포하고 관리하는 방식을 생각해보자. 먼저 각 서버에 네트워크, 스토리지 등 필요한 리소스를 구성하는 과정이 필요할 것이다. 또한 서버 시스템 사양에 맞춰 애플리케이션 배포에 필요한 패키지를 설치하고 설정파일을 수정하는 과정도 필요하다.

이런 방식은 단순한 애플리케이션을 소수의 서버에 배포, 관리할 때는 문제가 없을 것이다. 하지만 구성이 복잡한 애플리케이션을 다수의 서버에 배포, 관리해야 한다면 다음과 같은 문제점이 발생할 수 있다.

- 일련의 업무를 수동으로 진행함에 따르는 인적 노력 및 시간
- 서버 시스템 사양에 따라 다른 패키지와 설정파일이 요구됨에 따른 복잡성
- 휴먼에러 등

2) IaC 의 필요성

IaC (Infra as Code)는 고전적인 서버 인프라 구축의 문제점을 보완하기 위해 등장하였다. IaC 는 시스템이 읽을 수 있는 인프라 정의 파일을 통해 서버의 인프라 구성을 마치 소프트웨어를 구성하는 것 처럼 처리하는 방식을 의미한다. 고전적인 서버 인프라 구축과 비교하여 IaC 가 가진 장점은 다음과 같다.

- 인적 노동 및 시간 절감 (비용 절감)
- 빠른 인프라 구축 가능
- 오류 및 보안 위반과 같은 위험 제거

IaC 를 구현할 수 있는 대표적인 툴은 Ansible, Chef, Fabric, Puppet, SaltStack 등이 있다.

3) Ansible



Ansible 은 애플리케이션을 원격 노드에 배포 및 관리하기 위해 사용되는 오픈소스 도구이다. Python 및 YAML 을 기반으로 만들어졌다. Python 모듈을 사용하여 애플리케이션 및 애플리케이션이 위치한 노드를 관리할 수 있으며, YAML 형식으로 리소스를 관리한다.

다른 IaC 툴과 비교하여 Ansible 은 다음과 같은 장점이 있다.

- SSH 기반으로 작동되며, 각 노드에 에이전트를 설치할 필요가 없다.
- Playbook 은 YAML 형식을 사용하여, 쉽게 배울수 있고 구조를 파악하기도 쉽다.
- 이미 존재하는 Python 모듈을 사용하며, 입력 항목이 정의되어 있으므로 직관적이다.

한편 Ansible 사용법은 기본적으로 AD-Hoc 방식과 Playbook 방식으로 구분할 수 있다.
AD-Hoc 은 터미널에서 커맨드 작성으로 작동되는 방식이며, 단일 task 만 수행할 수 있다.
반면 Playbook 은 AD-Hoc 으로 실행 가능한 task 들의 집합으로, YAML 형식으로 작성된 Playbook 을 작성하여 관리 및 수행한다.

4) AWX



AWX 는 RedHat 의 Ansible 관리 서비스인 Ansible Tower 의 오픈소스 버전이다.
REST API, 웹 서비스 및 웹 기반의 대시보드를 제공하여 Ansible 을 더 유용하게 사용할 수 있도록 하는 서비스이다.

AWX 는 데이터베이스로 PostgreSQL 을 사용하며, AWX 설치 시 AWX 및 데이터베이스를 구성하는 방법은 다음과 같다.

(1) AWX 와 데이터베이스를 같은 서버에 구성

- 장점: 원격에 접속할 수 있는 포트를 개방하지 않으므로 취약점 공격에 안전
- 단점: AWX 와 DB 가 같은 리소스를 사용하여, 리소스 손실 발생

(2) AWX 와 데이터베이스를 서로 다른 서버에 구성

- 장점: 리소스를 독립적으로 사용하여, 성능이 높음
- 단점: 원격 접속을 위한 포트가 개방되어 있어 보안관리가 필요함

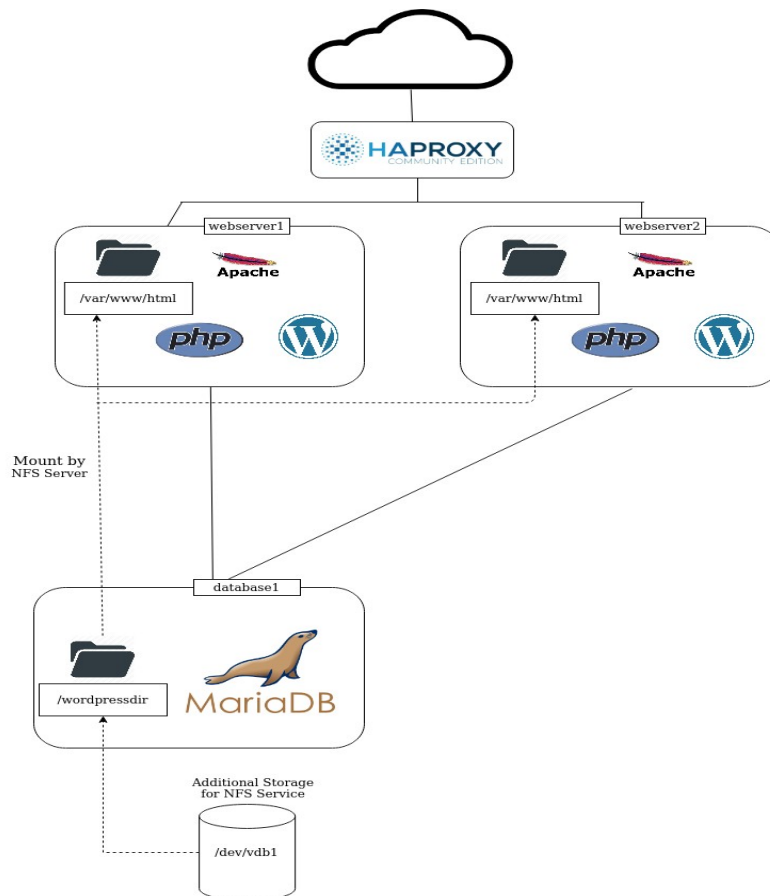
(3) AWX 클러스터 구성 (이중화)

- 장점:고가용성의 구축이 가능함
- 단점:고가용성을 위한 보다 많은 서버를 필요로 함

2. 시스템 구성

IaC 활용을 위해 Ansible playbook 및 AWX 로 간단한 애플리케이션을 구성 및 관리해보도록 하자.
Apache2, PHP 및 MariaDB 를 기반으로 WordPress 를 구성할 것이며, haproxy 를 활용하여 고가용성도 구축해 볼 것이다. 본 실습을 위한 시스템 구성은 다음과 같다.

1) Wordpress 구성



AWX 를 통한 서비스 배포 및 관리를 위해 위와 같은 서비스를 구성하였다.

(1) LoadBalancer (node1)

haproxy 를 통해 외부와 통신하며, 모든 webserver 들을 RoundRobin 방식으로 로드밸런싱 한다.

(2) Webserver (node2, node3)

Apache 및 PHP 를 통해 WordPress 를 서비스한다. 각 서비스는 node2, 3 에 위치하고 있으며, Ansible Playbook 을 통해 고가용성으로 구성한다.

html 디렉토리는 /var/www/html 이며, NFS 서버에서 제공하는 /wordpressdir 디렉토리와 마운트하여 사용한다.

(3) Database (node4)

MariaDB 를 통해 WordPress 에 데이터베이스서비스를 제공한다.

(4) NFS (node4)

Webserver 의 html 디렉토리를 위해 외부 디렉토리를 제공한다.

본래 별도의 스토리지 서버 구성이 필요하나, 실습 컴퓨터 리소스 문제로 Database1 이 위치한 node4 에 구성하였다.

추가 스토리지 /dev/vdb1 을 /wordpressdir 과 마운트하고, nfs-server 서비스로 webserver 와 통신할 수 있도록 설정하였다.

한편 각 서비스와 별도로 Ansible 및 AWX 구동을 위해 controller node 가 별도로 존재한다.

2) 시스템 구조

각 서비스를 위한 서버는 KVM 을 통해 VM 으로 구축하였다. 서버 구조 및 구성환경은 다음과 같다.

분류	호스트 네임	OS	VCPU (Core)	vMemory (MB)	NIC	서비스
Control Machine	controller	CentOS 7.8	2	2,048	NIC1: NAT NIC2: Private Network	1. Ansible 2. AWX
Node (Worker)	node1	Ubuntu 18.04*	1			RoadBalancer
	node2	CentOS 7.8				Webserver
	node3	CentOS 7.8				
	node4	CentOS 7.8				

* playbook 의 OS 타입별 조건문 테스트를 위해 node1 만 Ubuntu 환경으로 구성

3) 네트워크 구조

모든 서버는 (1) 패키지 설치 및 외부 서비스를 위한 NAT 네트워크, (2) SSH 및 내부 통신을 위한 Private Network 로 구성되어 있다. 각 네트워크별 세부구조는 다음과 같다.

분류	호스트 네임	NAT	Private Network
Control Machine	controller	Ipv4: 192.168.122.41/24 G.W: 192.168.122.1 DNS: 8.8.8.8, 4.4.8.8	Ipv4: 192.168.123.41/24
Node (Worker)	node1	Ipv4: 192.168.122.51/24 G.W: 192.168.122.1 DNS: 8.8.8.8, 4.4.8.8	Ipv4: 192.168.123.51/24
	node2	Ipv4: 192.168.122.52/24 G.W: 192.168.122.1 DNS: 8.8.8.8, 4.4.8.8	Ipv4: 192.168.123.52/24
	node3	Ipv4: 192.168.122.53/24 G.W: 192.168.122.1 DNS: 8.8.8.8, 4.4.8.8	Ipv4: 192.168.123.53/24
	node4	Ipv4: 192.168.122.54/24 G.W: 192.168.122.1 DNS: 8.8.8.8, 4.4.8.8	Ipv4: 192.168.123.54/24

4) SW 사양

서비스를 위해 설치되는 SW 들의 세부사양은 다음과 같다.

- (1) controller
 - Ansible 2.9.10
 - AWX 14.0.0
 - Docker 19.03.12
 - Docker-compose 1.26.2
 - Python 2.7.5
 - Python 3.6.8
- (2) RoadBalancer
 - haproxy 1.8.8
- (3) Webserver
 - httpd 2.4.6
 - php 7.4
- (4) Database
 - MariaDB 10.5.5
- (5) NFS
 - nfs-utils 1.3

3. Playbook 설명

1) roles 를 통한 playbook 구성

playbook 은 인프라를 선언하는 파일을 관리하여 인프라를 구축 및 관리하는 방식이며, yaml 형태로 관리한다. 본 서비스의 playbook 은 roles 방식으로 구성 및 관리하였다. roles 은 지정된 역할 및 파일 구조를 기반으로 var, task, template 및 handler 등을 관리하는 방법이다. 각 역할 및 파일 구조는 디렉토리로 구분하여 작동하며, 각 디렉토리 안에 playbook 파일을 작성하여 고유의 역할을 수행할 수 있다.

roles 는 (1) 수동 구성방식(사용자가 직접 roles 디렉토리를 생성) (2) 자동 구성방식(ansible-galaxy 사용) 으로 구분된다. ansible-galaxy 을 이용한 자동구성은 ansible-galaxy role init [roles name] 커맨드로 가능하다.

```
$ ansible-galaxy role init new_role
- Role new_role was created successfully
$ ls new_role/
defaults files handlers meta README.md tasks templates tests vars
```

roles 는 다음과 같은 계층구조 및 역할을 가진다.

```
inventory.yml
site.yml
ansible.cfg
group_vars/wp/
roles/
  roles_name/
    files/
    templates/
    tasks/
    vars/
    defaults/
    meta/
```

(1) inventory.yml

play 의 주체인 group 을 지정하며, 각 group 에 Node 들을 지정한다. 본 서비스는 AWX 를 통해 진행되고, AWX 는 자체적으로 inventory 를 관리하는 기능이 있기에 해당 파일을 작성하지 않는다.

(2) site.yml

지정한 group 에 role 을 매핑시킨다.

(3) ansible.cfg

Ansible 의 특정 설정을 조정하는 파일이다. 환경변수 지정, /etc/ansible/ansible.cfg 파일 설정방식 등 구성방식은 다양하며, 본 서비스는 playbook 의 현재 디렉토리에 생성하는 방법을 선택했다.

(4) group_vars/

playbook 에서 사용할 변수를 파일로 지정한다. 변수를 파일로 지정하여 관리하는 방식은 다양하나, 다수의 group 을 지정하여 관리하는 본 실습의 특성 상 그룹단위의 변수를 사용할 수 있는 group_vars 방식을 선택하였다.

group_vars/[group_name]/ 내 모든 변수파일은 group_name 을 공유하는 모든 play 경로지정 없이 사용가능하다. (그룹별 전역변수)

(5) roles/

playbook 에서 사용할 각 role 이 저장된 디렉토리이다.

(6) roles/roles_name/files/

현 roles 내에서 사용할 이미지, 음성 등 files 들을 저장한다. 해당 roles 에서 경로지정 없이 사용 가능하다. 본 실습에서는 file 을 사용할 일이 없으므로, 사용하지 않는다.

(7) roles/roles_name/templates/

현 roles 내에서 사용할 templates 파일을 저장한다. 해당 roles 에서 경로지정 없이 templates 이름만으로 사용 가능하다.

(8) roles/roles_name/tasks/

현 roles 내에서 사용할 task 파일을 저장한다. site.yml 파일 play 시 inventory 구성에 따라 자동으로 실행된다.

(9) roles/roles_name/vars/

현 roles 내에서 사용할 변수 파일을 저장한다. 본 서비스는 group_vars 사용하므로, 사용하지 않는다.

(10) roles/roles_name/defaults/

현 roles 내에서 사용할 변수 파일을 저장한다. 각 roles 에서 사용할 기본 변수를 지정하나, /vars/보단 우선순위가 떨어진다. 본 서비스는 group_vars 사용하므로, 사용하지 않는다.

(11) roles/roles_name/meta/

현 roles 내에서 사용할 메타데이터를 지정한다. 본 meta data 는 ansible 전용 repository 인 ansible-galaxy 에 사용자 및 play 관련 정보를 제공한다. 본 서비스는 단순히 서비스를 구현하는 것이 목적이므로 사용하지 않는다.

2) ansible.cfg

```
[defaults]
```

```
# ansible 의 인증 서비스 vault 를 이용하기 위해, 매 play 시 vault password 를 묻도록 지정하였다.
```

```
ask_vault_pass = True
```

```
[ssh_connection]
```

```
# 파이프 라이닝을 사용하면 실제 파일의 전송 없이 필수 모듈을 실행하여 모듈을 실행하는데 필요한 SSH 작업 수를 줄일 수 있다.
```

```
pipelining = True
```

3) site.yml

```
# 모든 roles 에는 sudo 사용을 할수 있는 권한상승 옵션 become: yes 지정
# inventory 의 wp 그룹을 roles 의 allnode 와 매핑
- hosts: wp
  become: yes
  roles:
    - allnode
# inventory 의 wp-proxy 그룹을 roles 의 haproxy 와 매핑
- hosts: wp-haproxy
  become: yes
  roles:
    - haproxy
# inventory 의 wp-nfs 그룹을 roles 의 nfs 와 매핑
- hosts: wp-nfs
  become: yes
  roles:
    - nfs
# inventory 의 wp-database 그룹을 roles 의 database 와 매핑
- hosts: wp-database
  become: yes
  roles:
    - database
# inventory 의 wp-webserver 그룹을 roles 의 webserver 와 매핑
- hosts: wp-webserver
  become: yes
  roles:
    - webserver
```

4) group_vars

inventory 에서 지정된 group 에서 전역변수로 사용할 변수를 지정한다. yaml 형태로 저장되며, 다음과 같이 변수로 사용할 수 있다.

```
> 변수파일 test.yml
...
test:
  user:
    name: lee
    passwd: dkagh1.
...

> 변수사용 방법
{{ test.user.name }}
{{ test.user.passwd }}
```

(1) group_vars/wp/database.yml

```

database:
  yum_repository:
# MariaDB 10.5.5 설치를 위한 repository 정보를 변수
  baseurl: https://ftp.harukasan.org/mariadb/yum/10.5/centos7-amd64
  gpgkey: https://ftp.harukasan.org/mariadb/yum/RPM-GPG-KEY-MariaDB
# MariaDB 의 conf 경로 및 db_port 를 자유 지정 하기 위해 변수화
  conf:
    conf_path: /etc/my.cnf.d/server.cnf
    db_port: 3306
# wp 용 database 유저와 database 이름을 변수로 지정
  user:
    wp_user: admin
    wp_database: wordpress_db

```

(2) group_vars/wp/db-pw.yml

database 에서 root 및 wp_user 의 패스워드를 지정하기 위한 변수이다. 패스워드와 같은 민감한 정보를 암호화 없이 변수로 지정하면, 취약점 공격시 패스워드가 노출될 위험이 있다. 노출을 방지하기 위해 ansible-vault 툴로 변수를 암호화하여 관리할 수 있다. 변수를 암호화하여 관리하는 절차는 다음과 같다.

a. 패스워드를 위한 변수파일 생성

```

# database root 유저 패스워드를 변수로 지정
dbroot_passwd: dkagh1.
# database wp 유저 패스워드를 변수로 지정
wp_passwd: dkagh1.

```

b. ansible-vault 로 해당 파일 암호화

```

# 원하는 변수파일을 암호화
$ ansible-vault encrypt test.yml
# vault 파일에서 사용할 패스워드 입력
New Vault password:
Confirm New Vault password:
Encryption successful

```

c. 암호화한 변수파일 확인

```

# 기존 파일이 해시방식으로 암호화되었음을 확인할 수 있다.
$ANSIBLE_VAULT;1.1;AES256
31313063336264633331656131306235393561656135663964616638643364336538373232656237
3635363162373136373763613335323732326130323962340a336539363130376162663437633333
38346433616362633234613865333538373130343930303731656562336161366633353765386332
6564643565336433620a633431373030313730643464373362396636376131343434623565336337
63303164633236623861303936303465636238623563613536626164386438633764373032626462
3232313531313761613337333864333933343230613438356139

```

(3) group_vars/wp/haproxy.yml

```
haproxy:
# 사용자가 자유롭게 haproxy 설정파일 경로를 지정할 수 있도록 변수화
  conf_path: /etc/haproxy/haproxy.cfg
# haproxy 단에서 사용할 포트 지정
  frontend:
    port: 80
  backend:
# backend 단의 이름 지정
    name: wp
# balance type 지정을 위해 변수화
    balance_type: roundrobin
# webserver 단의 포트 및 ip 를 지정하기 위해 변수화
  web1:
    ip: 192.168.123.52
    port: 80
  web2:
    ip: 192.168.123.53
    port: 80
```

(4) group_vars/wp/nfs.yml

```
nfs:
# webserver 의 /var/www/html 로 마운트할 디렉토리 경로 지정
  nfs_mountpoint: /wordpressdir
# {{ nfs.nfs_mountpoint }}에 마운트할 디바이스의 파티션 정보 지정
  partition:
    device: /dev/vdb
    part_number: 1
    size: 5GiB
    fstype: ext4
# nfs-server 에서 사용할 exports 파일 경로 및 nfs_option 지정
  exports:
    conf_path: /etc/exports
    nfs_option: rw,sync,no_root_squash
  wordpress:
# /wordpress 디렉토리로 다운받을 wordpress 아카이브 버전 정보
    wp-release: 5.3.4
    wp-lang: ko_KR
# exports 에 지정할 webserver ip 정보
    web_iprange: 192.168.123.0
    web_prefix: 24
# wp-config.php 에 지정할 database 정보
    db_name: wordpress_db
    db_user: admin
    db_password: dkagh1.
```

(5) group_vars/wp/webservice.yml

```
webservice:
# nfs-server 의 /wordpressdir 과 webserver 의 /var/www/html 마운트시 필요한 정보들
nfs:
  mount_path: /var/www/html
  nfs_source: 192.168.123.54:/wordpressdir
  fstype: nfs
# remi-release 버전을 자유지정하기 위한 변수
remi_ver: 7
# html 설정파일 경로와 webservice port 를 자유지정하기 위한 변수
conf:
  conf_path: /etc/httpd/conf/httpd.conf
  port: 80
```

5) includes

playbook 의 모든 tasks 는 include 모듈을 사용하여, 파일로 지정된 tasks 를 불러올 수 있다. 별도의 경로지정이 불필요한 group_vars 나 templates 와 달리, includes 로 tasks 를 불러오기 위해서 해당 tasks 가 위치한 경로를 지정해줘야 한다.

이번 서비스 구축에서는 자주 사용하거나, 코드가 지나치게 긴 tasks 를 별도의 파일로 생성하여 사용하였다. 하기 task 에서 변수에서 가져오는 항목은 붉은색으로 표기하였다.

(1) includes/apt-install.yml

```
# ubuntu 환경에서 사용할 apt 패키지매니저 관련 tasks
- name: install specific service
  apt:
    name: '{{ service_name }}'
    state: latest
```

(2) includes/yum-install.yml

```
# CentOS 환경에서 사용할 yum 패키지매니저 관련 tasks
- name: install specific service
  yum:
    name: '{{ service_name }}'
    state: latest
```

(3) includes/firewalld.yml

CentOS 환경에서 사용할 firewalld 관련 tasks

```
- name: allow port for specific service
  firewalld:
    service: '{{ service_name }}'
    permanent: yes
    state: enabled
    immediate: yes
```

(4) includes/partition.yml

nfs 서버에서 스토리지 파티셔닝 및 마운팅을 위해 사용할 tasks

lvm 방식으로 파티셔닝 진행하는 모듈

```
- name: Create a new primary partition for LVM
  parted:
    device: "{{ nfs['partition']['device'] }}"
    number: "{{ nfs['partition']['part_number'] }}"
    flags: [ lvm ]
    state: present
    part_start: "{{ nfs['partition']['size'] }}"
```

파티셔닝된 디바이스에 파일시스템을 생성하는 모듈

```
- name: Create a ext4 fs on /dev/vdb1
  filesystem:
    fstype: "{{ nfs['partition']['fstype'] }}"
    dev: "{{ nfs['partition']['device'] }}{{ nfs['partition']['part_number'] }}"
```

파티셔닝된 디바이스를 /wordpressdir 에 마운트하는 모듈

```
- name: mount /dev/vdb1 on /wordpressdir
  mount:
    path: "{{ nfs['nfs_mountpoint'] }}"
    src: "{{ nfs['partition']['device'] }}{{ nfs['partition']['part_number'] }}"
    fstype: ext4
    state: mounted
```

exports 파일 템플릿을 exports 설정 디렉토리로 복사하는 모듈

template 모듈에 대해서는 후술하도록 한다.

```
- template:
  src: templates/exports.j2
  dest: "{{ nfs['exports']['conf_path'] }}"
```

(5) includes/seboolean.yml

CentOS 환경에서 사용할 sebool 정책 관련 tasks

```
- name: Active seboolean for specific function
  seboolean:
    name: "{{ sebool_name }}"
    state: yes
    persistent: yes
```

(6) includes/service.yml

서비스 시작을 위해 사용할 tasks

```
- name: start specific service
  service:
    name: "{{ service_name }}"
    enabled: true
    state: started
```

6) templates

templates 는 /roles/[role_name]/templates/ 경로에 저장하여 관리하며, Jinja2 템플릿을 사용한다. 확장자 지정은 자유이나, 원본파일명 뒤에 j2 라는 접미사를 붙여 많이 사용한다. (ex) original_file.j2) 각 templates 는 tasks 작성시 해당하는 [role_name]에 한해 경로지정 없이 사용 가능하다.

하기 템플릿에서 변수에서 가져오는 항목은 붉은색으로 표기하였다.

(1) /roles/database/templates/server.cnf.j2

Mariadb 의 설정파일을 카피하기 위한 템플릿 파일이다.

[server]

[mysqld]

해당 node 의 ipv4 팩트변수를 가져와 bind-address 에 지정한다.

bind-address="{{ ansible_eth1.ipv4.address }}"

database.yml 파일에서 database 에서 사용할 포트 변수를 가져와 지정한다.

port="{{ database['conf']['db_port'] }}"

(2) /roles/haproxy/templates/haproxy-ubuntu.cfg.j2

haproxy 가 설치되는 호스트가 ubuntu 시스템일 경우 사용될 haproxy 설정파일 템플릿이다.

global

log /dev/log local0

log /dev/log local1 notice

chroot /var/lib/haproxy

stats socket /run/haproxy/admin.sock mode 660 level admin expose-fd listeners

stats timeout 30s

user haproxy

group haproxy

daemon

ca-base /etc/ssl/certs

crt-base /etc/ssl/private

ssl-default-bind-ciphers

ECDH+AESGCM:DH+AESGCM:ECDH+AES256:DH+AES256:ECDH+AES128:DH+AES:RSA+AESGCM:RSA+AES:!aNULL:!MD5:!DSS

ssl-default-bind-options no-ssl3

```
defaults
    log      global
    mode     http
    option   httplog
    option   dontlognull
    timeout connect 5000
    timeout client 50000
    timeout server 50000
    errorfile 400 /etc/haproxy/errors/400.http
    errorfile 403 /etc/haproxy/errors/403.http
    errorfile 408 /etc/haproxy/errors/408.http
    errorfile 500 /etc/haproxy/errors/500.http
    errorfile 502 /etc/haproxy/errors/502.http
    errorfile 503 /etc/haproxy/errors/503.http
    errorfile 504 /etc/haproxy/errors/504.http

frontend main
# bind 시킬 호스트 ip 정보를 변수로서 가져와 사용한다.
# {{ haproxy_public_ip }}: 호스트의 ip 를 정의한 팩트변수이며, group_vars 가 아닌 tasks 에서 정의할
# 예정이다.
bind {{ haproxy_public_ip }}:{{ haproxy['frontend']['port'] }}
acl url_static    path_beg    -i /static /images /javascript /stylesheets
acl url_static    path_end    -i .jpg .gif .png .css .js

# use_backend static 은 주석처리하여, 웹브라우저에서 php 가 표시될 수 있도록 한다.
#use_backend static      if url_static
default_backend          {{ haproxy['backend']['name'] }}

# 백엔드 앱 이름, 로드밸런싱 방식, 로드밸런싱할 웹서버 등을 변수로서 지정한다.
backend {{ haproxy['backend']['name'] }}
    balance    {{ haproxy['backend']['balance_type'] }}
    server web1 {{ haproxy['backend']['web1']['ip'] }}:{{ haproxy['backend']['web1']['port'] }} check
    server web2 {{ haproxy['backend']['web2']['ip'] }}:{{ haproxy['backend']['web2']['port'] }} check
```


(3) /roles/haproxy/templates/haproxy.cfg.j2

```
# haproxy 호스트 시스템이 CentOS 일 경우 사용될 haproxy 설정파일 템플릿이다.
# haproxy-ubuntu.cfg.j2 템플릿과 크게 다른점은 없다.
global
    log      127.0.0.1 local2
    chroot   /var/lib/haproxy
    pidfile  /var/run/haproxy.pid
    maxconn  4000
    user     haproxy
    group    haproxy
    daemon
    stats socket /var/lib/haproxy/stats
defaults
    mode                http
    log                  global
    option               httplog
    option               dontlognull
    option http-server-close
    option forwardfor    except 127.0.0.0/8
    option               redispatch
    retries              3
    timeout http-request 10s
    timeout queue        1m
    timeout connect      10s
    timeout client       1m
    timeout server       1m
    timeout http-keep-alive 10s
    timeout check        10s
    maxconn              3000

frontend main {{ haproxy_public_ip }}:{{ haproxy['frontend']['port'] }}
    acl url_static path_beg -i /static /images /javascript /stylesheets
    acl url_static path_end -i .jpg .gif .png .css .js

    #use_backend static      if url_static
    default_backend {{ haproxy['backend']['name'] }}

backend static
    balance roundrobin
    server static 127.0.0.1:4331 check

backend {{ haproxy['backend']['name'] }}
    balance {{ haproxy['backend']['balance_type'] }}
    server web1 {{ haproxy['backend']['web1']['ip'] }}:{{ haproxy['backend']['web1']['port'] }} check
    server web2 {{ haproxy['backend']['web2']['ip'] }}:{{ haproxy['backend']['web2']['port'] }} check
```

(4) /roles/nfs/templates/exports.j2

```
# nfs-server 에서 사용될 exports 설정파일에 대한 템플릿이다.
# group_vars 에서 마운트할 디렉토리, 클라이언트 ip 정보, nfs 옵션 등을 변수로서 호출하여 사용한다.
"{{ nfs['nfs_mountpoint'] }}" " {{ nfs['wordpress']['web_iprange'] }}" / " {{ nfs['wordpress']
['web_prefix'] }}" " {{ nfs['exports']['nfs_option'] }}" )
```

(5) /roles/nfs/templates/wp-config.php.j2

```
# wordpress 디렉토리에서 사용할 설정파일에 대한 템플릿이다.
# group_vars 에서 database 이름, database 유저명 및 비밀번호를 변수로서 호출하여 사용한다.

<?php
/** The name of the database for WordPress */
define( 'DB_NAME', " {{ nfs['wordpress']['db_name'] }}" );

/** MySQL database username */
define( 'DB_USER', " {{ nfs['wordpress']['db_user'] }}" );
# database 비밀번호는 보안을 위해 암호화된 db-pw.yml 에서 가져온다.
/** MySQL database password */
define( 'DB_PASSWORD', " {{ wp_passwd }}" );
# database host 정보는 해당 호스트가 고유하게 가지고 있는 팩트변수를 사용하여 지정한다.
# 본 실습은 nfs 서버-database 서버가 일치하기에 호스트 자신의 ip 를 사용했으나, 두 서버를 별도로 구성하여
# 사용하는 경우 팩트변수가 아닌, database 서버 ip 를 별도로 변수화하여 사용할 필요가 있다.
/** MySQL hostname */
define( 'DB_HOST', " {{ ansible_eth1.ipv4.address }}" );

/** Database Charset to use in creating database tables. */
define( 'DB_CHARSET', 'utf8' );

/** The Database Collate type. Don't change this if in doubt. */
define( 'DB_COLLATE', '' );

define( 'AUTH_KEY',      'put your unique phrase here' );
define( 'SECURE_AUTH_KEY', 'put your unique phrase here' );
define( 'LOGGED_IN_KEY',  'put your unique phrase here' );
define( 'NONCE_KEY',     'put your unique phrase here' );
define( 'AUTH_SALT',     'put your unique phrase here' );
define( 'SECURE_AUTH_SALT', 'put your unique phrase here' );
define( 'LOGGED_IN_SALT', 'put your unique phrase here' );
define( 'NONCE_SALT',    'put your unique phrase here' );

$table_prefix = 'wp_';

define( 'WP_DEBUG', false );

if ( ! defined( 'ABSPATH' ) ) {
```

```
    define( 'ABSPATH', dirname( __FILE__ ) . '/' );
}

require_once( ABSPATH . 'wp-settings.php' );
```

(6) /roles/webserver/templates/httpd.conf.j2

```
# httpd 에서 사용할 설정파일에 대한 템플릿이다.
# 기존 설정파일과 유사하며, Listen 대상 호스트 정보를 지정하기 위한 변수만 존재한다.

ServerRoot "/etc/httpd"

# {{ ansible_eth1.ipv4.address }}: 호스트 자신의 ip 를 호스트 고유의 팩트변수로 호출하여 사용했다.
# {{ webservice['conf']['port'] }}: 서비스할 포트는 group_vars 에서 호출하여 사용했다.
Listen {{ ansible_eth1.ipv4.address }}:{{ webservice['conf']['port'] }}

Include conf.modules.d/*.conf

User apache
Group apache

ServerAdmin root@localhost

<Directory />
    AllowOverride none
    Require all denied
</Directory>

DocumentRoot "/var/www/html"

<Directory "/var/www">
    AllowOverride None

    Require all granted
</Directory>

<Directory "/var/www/html">
    Options Indexes FollowSymLinks
    AllowOverride None
    Require all granted
</Directory>

<IfModule dir_module>
    DirectoryIndex index.html
</IfModule>

<Files ".ht*">
    Require all denied
</Files>
```

```
ErrorLog "logs/error_log"
```

```
LogLevel warn
```

```
<IfModule log_config_module>
```

```
LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-Agent}i\"" combined
```

```
LogFormat "%h %l %u %t \"%r\" %>s %b" common
```

```
<IfModule logio_module>
```

```
LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-Agent}i\" %I %O" combinedio
```

```
</IfModule>
```

```
CustomLog "logs/access_log" combined
```

```
</IfModule>
```

```
<IfModule alias_module>
```

```
ScriptAlias /cgi-bin/ "/var/www/cgi-bin/"
```

```
</IfModule>
```

```
<Directory "/var/www/cgi-bin">
```

```
AllowOverride None
```

```
Options None
```

```
Require all granted
```

```
</Directory>
```

```
<IfModule mime_module>
```

```
TypesConfig /etc/mime.types
```

```
AddType application/x-compress .Z
```

```
AddType application/x-gzip .gz .tgz
```

```
AddType text/html .shtml
```

```
AddOutputFilter INCLUDES .shtml
```

```
</IfModule>
```

```
AddDefaultCharset UTF-8
```

```
<IfModule mime_magic_module>
```

```
MIMEMagicFile conf/magic
```

```
</IfModule>
```

```
EnableSendfile on
```

```
IncludeOptional conf.d/*.conf
```

7) tasks

(1) roles/allnode/tasks/main.yml

모든 노드에 selinux 관련 python module 설치를 위한 패키지를 설치하는 task 이다. 이 task 를 통해 설치되는 패키지는 다음과 같다.

- A. epel-release: Python 모듈 패키지 설치를 위함
- B. libsemanage-python: playbook 에서 selinux 관련 모듈을 사용하기 위함

```
# 반복문 with_items 를 사용하여 필요한 패키지를 순차적으로 설치한다.
- name: install epel-release & libsemanage-python
  include: includes/yum-install.yml service_name={{ item }}
  with_items:
    - epel-release
    - libsemanage-python
# 시스템에 큰 영향을 미치지 않는 작업이므로, check 모드 설정 시에도 실제 설치가 이뤄질 수 있도록 한다.
check_mode: no
# selinux 기능은 RedHat 계열 시스템만 존재하므로, 조건문을 통해 RedHat 계열 패키지에만 해당 task 가
작동될 수 있도록 한다.
when: ansible_os_family == 'RedHat'
```

(2) roles/nfs/tasks/main.yml

node4 에 nfs 관련 패키지 설치 및 구성을 진행하는 task 이다.

```
# /tmp 디렉토리에 wordpress 아카이브파일을 다운로드 한다.
# async 에 해당 task 를 실행하는 시간을 지정하고, poll 값을 0 으로 두어 task 를 비동기화로 실행한다.
# 비동기화로 wordpress 의 다운로드가 진행되는 동안 다음 task 를 실행시킬 수 있다.
- name: Download wordpress
  get_url:
    url: "https://ko.wordpress.org/wordpress-{{ nfs['wordpress']['wp-release'] }}-{{ nfs['wordpress']['wp-lang'] }}.tar.gz"
    dest: "/tmp/wordpress-{{ nfs['wordpress']['wp-release'] }}-{{ nfs['wordpress']['wp-lang'] }}.tar.gz"
  async: 300
  poll: 0
  check_mode: no
# nfs-utils 패키지를 설치한다.
- name: install nfs-utils
  include: includes/yum-install.yml service_name=nfs-utils
  check_mode: no
# nfs 서비스를 위한 디렉토리를 생성한다.
- name: Create a directory for nfs service
  file:
    path: "{{ nfs['nfs_mountpoint'] }}"
```

```

state: directory
mode: '0775'
check_mode: no
# 추가 스토리지를 파티셔닝 후 이전 task 에서 생성한 디렉토리에 마운트한다.
# 단 해당 task 는 block 으로 묶고, /group_vars/wp/nfs.yaml 에 디바이스명에 해당하는 변수가 선언되었을
# 시에만 작동하도록 설정한다.
# 만약 디바이스명이 선언되지 않았다면 해당 task 는 작동하지 않는다.
- name: Attempt format partition and mount
  block:
    - name: format partition and mount if device is defined
      include: includes/partition.yml
      when: nfs['partition']['device'] is defined
# 다운받은 Wordpress 아카이브를 nfs 서비스를 위한 디렉토리에 언아카이브한다.
- name: Unarchive wordpress arch
  unarchive:
    src: "/tmp/wordpress-{{ nfs['wordpress']['wp-release'] }}-{{ nfs['wordpress']['wp-lang'] }}.tar.gz"
    dest: "{{ nfs['nfs_mountpoint'] }}"
    remote_src: yes
    owner: root
    group: root
# wordpress 에 database 설정을 위해 템플릿 wp-config.php.j2 를 wordpress 디렉토리로 복사한다.
# nfs 디렉토리 내용 변경시, 서비스를 재시작해야 반영되므로 notify 를 지정한다.
# notify 에 따라 템플릿에 변동사항 발생 시, handler 에 의해 nfs-service 가 재시작된다.
- name: template wp-config.php
  template:
    src: templates/wp-config.php.j2
    dest: "{{ nfs['nfs_mountpoint'] }}/wordpress/wp-config.php"
  notify: restart nfs-server
# nfs 서비스를 시작한다.
- name: start nfs service
  include: includes/service.yml service_name=nfs-server
# nfs 서비스를 위한 방화벽 설정을 진행한다.
- name: allow port for nfs, rpc-bind and mountd
  include: includes/firewalld.yml service_name={{ item }}
  with_items:
    - nfs
    - rpc-bind
    - mountd

```

(3) roles/database/tasks/main.yml

node4 에 mariadb 를 설치하고 wordpress 를 위한 user 및 database 를 생성하는 task 다.

```

# 원하는 버전의 mariadb 설치를 위해 repository 를 추가한다.
- name: Add yum_repository for mariadb
  yum_repository:

```

```
name: MariaDB
baseurl: "{{ database['yum_repository']['baseurl'] }}"
gpgkey: "{{ database['yum_repository']['gpgkey'] }}"
gpgcheck: 1
description: MariaDB
check_mode: no
```

MariaDB 패키지를 설치한다.

enablerepo 옵션으로 패키지를 불러오고자 하는 특정 레포지토리를 지정할 수 있다/

```
- name: Install MariaDB_server and client
yum:
  name: MariaDB-server,MariaDB-client
  enablerepo: MariaDB
  state: latest
  check_mode: no
```

ansible 에서 **database** 관련 모듈을 사용할 수 있도록 MySQL-python 모듈을 설치한다.

```
- name: Install MySQL-python
include: includes/yum-install.yml service_name="MySQL-python"
check_mode: no
```

bind-address 와 **port** 를 지정하기 위해 **server.cnf.j2** 템플릿을 **database** 설정 디렉토리로 복사한다.

설정파일 변경시 서비스를 재시작해야 반영되므로, notify 를 지정하여 **handler** 작동이 가능하도록 한다.

```
- name: Limit accessible hosts and Set port for db at init node
template:
  src: server.cnf.j2
  dest: "{{ database['conf']['conf_path'] }}"
  notify: restart mariadb
```

mariadb 서비스를 시작한다.

```
- name: Start mariadb
include: includes/service.yml service_name=mariadb
```

root 비밀번호 지정/익명유저 삭제/test db 삭제하는 과정을 진행한다.

```
- name: Set root password
mysql_user:
  login_user: root
  login_password: ""
  user: root
  password: "{{ dbroot_passwd }}"
  state: present
  ignore_errors: yes
- name: Delete anonymous user in DB
mysql_user:
  login_user: root
  login_password: "{{ dbroot_passwd }}"
  name: ""
  host_all: yes
  state: absent
```

```
- name: Delete test db in DB
mysql_db:
  login_user: root
  login_password: "{{ dbroot_passwd }}"
  name: test
  state: absent

# wordpress 서비스를 위한 database 및 user 를 생성한다.
- name: Create DB for wordpress
mysql_db:
  login_user: root
  login_password: "{{ dbroot_passwd }}"
  name: "{{ database['wp_database'] }}"
  state: present
- name: Create User for wordpress
mysql_user:
  login_user: root
  login_password: "{{ dbroot_passwd }}"
  name: "{{ database['user']['wp_user'] }}"
  password: "{{ wp_passwd }}"
  priv: "{{ database['wp_database'] }}.*:ALL,GRANT"
  host: '192.168.123.%'
  state: present

# database 서비스를 위한 방화벽을 설정한다.
# server.cnf.j2 에서 지정한 특정 포트를 개방하기 위해 port 옵션을 사용한다.
- name: Open port for database
firewalld:
  port: "{{ database['conf']['db_port'] }}/tcp"
  permanent: yes
  state: enabled
  immediate: yes

#database 와 webserver 통신이 가능하도록 특정 sebool 을 활성화시킨다.
- name: Active seboolean for mysql
include: includes/seboolean.yml sebool_name=mysql_connect_any
```


(4) roles/haproxy/tasks/main.yml

node1 에 haproxy 패키지를 설치하고, webserver 들을 설정파일에 추가하는 task 다.
task 를 묶는 block 모듈과 조건문 when 을 테스트 하기위해 OS 타입에 따라 block 를 구분하였다.

호스트의 팩트변수 ansible_distribution 이 CentOS 인 경우 실행되는 task

```
- name: Deploy to CentOS
  block:
    - name: Install haproxy
      yum:
        name: haproxy
        state: latest
    - name: Open http port
      firewallld:
        port: "{{ haproxy['frontend']['port'] }}/tcp"
        permanent: yes
        state: enabled
        immediate: yes
    - name: Active seboolean for httpd
      seboolean:
        name: haproxy_connect_any
        state: yes
        persistent: yes
```

set_fact 모듈로 호스트의 팩트변수를 지역변수로 선언할 수 있다.

host 의 ip 주소를 haproxy_public_ip 라는 변수로 선언한다.

```
- name: Set facts for haproxy public ip
  set_fact:
    haproxy_public_ip: "{{ ansible_eth0.ipv4.address }}"
```

haproxy 설정파일 템플릿을 haproxy 설정 디렉토리로 복사하는 task 이다.

haproxy 설정파일 변경시 haproxy 재시작되도록 notify 를 지정한다.

```
- name: Copy haproxy configuration
  template:
    src: haproxy.cfg.j2
    dest: /etc/haproxy/haproxy.cfg
  notify:
    - restart haproxy
  when: ansible_distribution == "CentOS"
```

호스트의 팩트변수 ansible_distribution 이 Ubuntu 인 경우 실행되는 task

```
- name: Deploy to Ubuntu
  block:
    - name: Install haproxy
      apt:
        name: haproxy
        state: latest
        update_cache: yes
    - name: Set facts for haproxy public ip
      set_fact:
```

```

haproxy_public_ip: "{{ ansible_ens3.ipv4.address }}"
# haproxy 설정파일 템플릿을 haproxy 설정 디렉토리로 복사하는 task 이다.
# 파일 형식이 호스트 OS 타입에 따라 차이가 있으므로 템플릿 선택시 주의하자.
# haproxy 설정파일 변경시 haproxy 재시작되도록 notify 를 지정한다.
- name: Copy haproxy configuration
  template:
    src: haproxy-ubuntu.cfg.j2
    dest: /etc/haproxy/haproxy.cfg
  notify:
    - Restart haproxy service
  when: ansible_distribution == "Ubuntu"

# haproxy 서비스를 시작한다.
- name: Start haproxy service
  service:
    name: haproxy
    enabled: true
    state: started

```

(5) roles/haproxy/tasks/main.yml

node2, node3 에 nfs-server 에서 제공하는 디렉토리를 웹서버 디렉토리와 마운트하고, 웹서비스를 시작하는 task 이다.

```

# mount type nfs 가 지원 가능하도록 nfs-utils 패키지를 설치한다.
- name: Install nfs-utils for mount
  include: includes/yum-install.yml service_name=nfs-utils
  check_mode: no

# nfs 서버에서 제공하는 디렉토리를 웹서버 디포트 디렉토리와 마운트한다.
- name: mount /wordpressdir on /var/www/html
  mount:
    path: "{{ webservice['nfs']['mount_path'] }}"
    src: "{{ webservice['nfs']['nfs_source'] }}"
    fstype: "{{ webservice['nfs']['fstype'] }}"
    state: mounted

# httpd 를 설치하고, 특정 버전의 php 를 설치하기 위해 remi-release 도 설치한다.
- name: Install httpd, remi-release-{{ webservice['remi_ver'] }}"
  include: includes/yum-install.yml service_name="{{ item }}"
  with_items:
    - httpd
    - https://rpms.remirepo.net/enterprise/remi-release-{{ webservice['remi_ver'] }}.rpm

# httpd 설정 템플릿을 httpd 설정 디렉토리로 복사한다.
# httpd 설정파일 변경시 서비스 재시작되도록 notify 를 지정한다.
- name: reset bind-address and port

```

```
template:
  src: httpd.conf.j2
  dest: "{{ webservice['conf']['conf_path'] }}"
  notify: restart httpd
# remi-release 패키지로 추가된 remi-php74 레포지토리를 활용하여 7.4 버전 php 패키지를 설치한다.
- name: Install php and php-mysql
  yum:
    name: php,php-mysql
    enablerepo: remi-php74
    state: latest

# apache 서비스를 위한 방화벽을 설정한다.
# httpd.conf.j2 에서 지정한 특정 포트를 개방하기 위해 port 옵션을 사용한다.
- name: Open port for httpd
  firewallld:
    port: "{{ webservice['conf']['port'] }}/tcp"
    permanent: yes
    state: enabled
    immediate: yes

# httpd 가 nfs 및 database 서비스를 이용하기 위해 특정 sebool 정책을 활성화한다.
- name: Active seboolean for httpd
  include: includes/seboolean.yml sebool_name="{{ item }}"
  with_items:
    - httpd_can_network_connect_db
    - httpd_use_nfs

# apache 서비스를 시작한다.
- name: Start httpd
  service:
    name: httpd
    state: started
    enabled: true
```

8) Handler

Handler 는 특정 task 의 변동사항이 발생했을 시 notify 값을 트리거로 작동되는 일종의 task 이다. 보통 특정 서비스의 설정파일 변경이 발생하였을 때 서비스를 재시작하는 용도로 많이 사용된다.

/roles/[role_name]/handlers/ 디렉토리에 YAML 형태로 저장되며, 해당하는 role 에 한해 경로지정 없이 사용 가능하다.

(1) roles/nfs/handlers/main.yml

```
# nfs 설정파일 exports 변동시 nfs-server 를 재시작하는 handler
- name: restart nfs-server
  service:
    name: nfs-server
    state: restarted
    enabled: true
```

(2) roles/database/handlers/main.yml

```
# MariaDB 설정파일 sever.cnf 변동시 mariadb 를 재시작하는 handler
- name: restart mariadb
  service:
    name: mariadb
    state: restarted
    enabled: true
```

(3) roles/haproxy/handlers/main.yml

```
# haproxy 설정파일 haproxy.cnf. 변동시 haproxy 를 재시작하는 handler
- name: restart haproxy
  service:
    name: haproxy
    enabled: true
    state: restarted
```

(4) roles/webserver/handlers/main.yml

```
# httpd 설정파일 httpd.conf 변동시 httpd 를 재시작하는 handler
- name: restart httpd
  service:
    name: httpd
    enabled: true
    state: restarted
```

4. AWX 설치

AWX 는 기본적으로 Ansible controller 에 설치된다. 앞서 설명한 바와 같이 database 는 PostgreSQL 을 사용하며, AWX 와 database 를 구성하는 다양한 방법이 존재한다. 본 서비스 구축에서는 컴퓨터 리소스의 문제로 AWX 와 PostgreSQL 을 같은 호스트에 설치하도록 한다.

한편 AWX 는 필요요소들을 container 로 구성하는 방식이며, OpenShift, Kubernetes, Docker-Compose 를 사용하여 설치 가능하다. 본 서비스 구축에서는 Docker-Compose 를 사용하여 설치하는 방식을 채택하였다.

설치에 앞서 AWX 14.0.0 기준 필요한 SW 는 다음과 같다. (향후 변동가능)

- Ansible 2.8 이상
- Docker 최신버전
- docker python 모듈
- Make
- Git 1.8.4 이상
- Python 3.6 이상
- Nodejs 10.x 버전 (선택사항)
- NPM 6.x 버전 (선택사항)

(1) 필요한 패키지 설치

```
# 파이썬 모듈 등 추가 패키지 설치를 위한 epel-release 설치
$ yum install epel-release

# ansible, make, git, nodejs, npm, python3, pip 설치
$ yum install ansible make git nodejs npm python3 python3-pip

# docker 설치를 위해 docker repository 등록
$ yum-config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo

# docker 및 구성요소 설치
$ yum install docker-ce docker-ce-cli containerd.io

# docker 서비스 활성화
$ systemctl enable --now docker

# sudo 없이 docker 실행이 가능하도록 현재 유저를 docker 그룹에 포함
$ sudo usermod -aG docker [username]

# pip 로 docker-python 패키지 설치
$ sudo pip3 install docker docker-compose selinux
```

(2) awx 패키지 설치

```
# awx 패키지는 github 의 공식 레포지토리에서 다운받을 수 있다. git 커맨드로 14.0.0 버전을 clone 하자.  
$ git clone -b 14.0.0 https://github.com/ansible/awx.git
```

(3) project data 디렉토리 설정

```
# AWX 는 project 라는 단위로 playbook 을 관리한다. 모든 playbook 은 project data 디렉토리에 저장하여  
사용한다.  
# AWX 설치에 앞서 project data 를 저장할 기본 디렉토리를 지정하자.  
$ cd awx/installer  
  
$ vi inventory  
...  
# project data 기본 디렉토리를 /var/lib/awx/projects 로 지정  
140 project_data_dir=/var/lib/awx/projects  
...
```

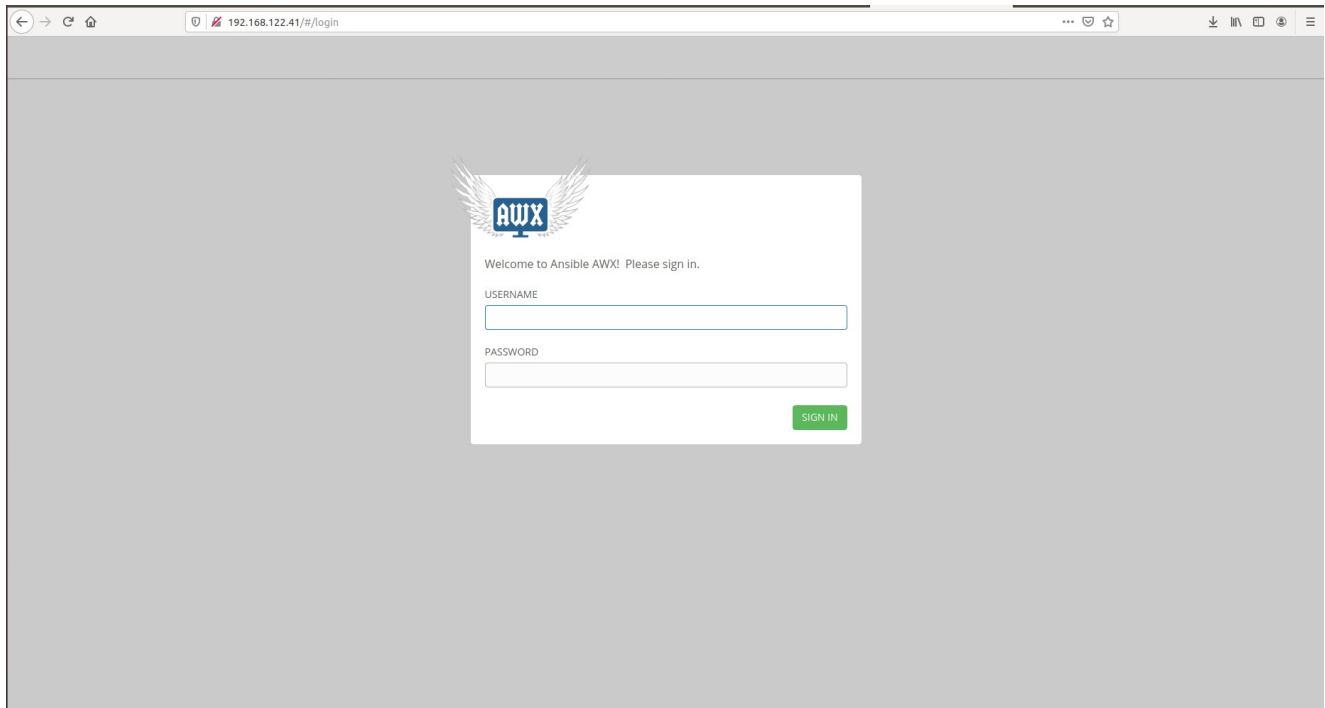
(4) ansible-playbook 로 awx 설치

```
$ ansible-playbook -i inventory installer.yml -b
```

설치 완료 후 docker ps 커맨드로 현재 구동되는 컨테이너를 확인하였을 때, awx_라는 접미사가 붙은 컨테이너들이 up 상태를 확인할 수 있다.

```
[student@controller playbook]$ docker ps  
CONTAINER ID   IMAGE                COMMAND                  CREATED        STATUS        PORTS                NAMES  
f9ed07844550   ansible/awx:14.0.0   "tini -- /usr/bin/la..." 4 days ago    Up 9 seconds   8052/tcp             awx_task  
5781eb5a59f5   ansible/awx:14.0.0   "tini -- /bin/sh -c ..." 4 days ago    Up 8 seconds   0.0.0.0:80->8052/tcp  awx_web  
7d58da8a92b7   redis                "docker-entrypoint.s..." 4 days ago    Up 9 seconds   6379/tcp             awx_redis  
2b96111ba7be   postgres:10          "docker-entrypoint.s..." 4 days ago    Up 8 seconds   5432/tcp             awx_postgres
```

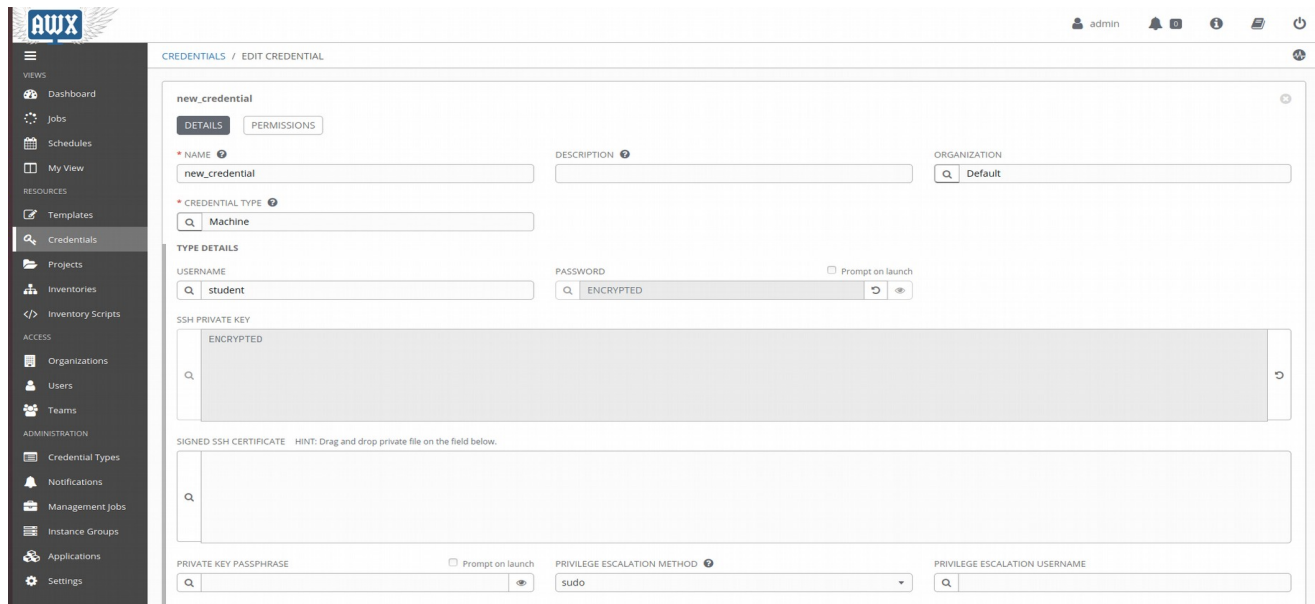
웹브라우저로 controller NAT 네트워크 아이피로 접속 시, AWX 대시보드 로그인 페이지를 확인할 수 있다. 디폴트 USERNAME: admin PASSWORD: password 이며, 설치 전 awx/installer/inventory 에서 변경 가능하다.



5. AWX 구성

AWX 는 인증, 인벤토리 등 play 에 필요한 다양한 리소스를 대시보드를 통해 관리할수 있다. 각 리소스를 설정하고 앞서 만든 playbook 을 play 하는 과정을 진행해보자.

1) Credential



AWX 의 Credential 은 play 에 필요한 인증관리 기능을 제공한다. Credential 을 통해 설정해야 할 인증정보는 다음과 같다.

(1) SSH key

기본적으로 ansible 은 SSH 기반으로 컨트롤러-노드의 통신이 이뤄지기에 ssh 키에 대한 인증설정이 필요하다. SSH key 를 등록하는 과정은 다음과 같다.

A. Credential 탭을 클릭한다.

B. 현재 등록된 Credential 정보가 표기된다. 정보 등록을 위해 우측 상단의 +버튼을 클릭한다.

C. DETAILS 탭에서 다음 정보를 입력한다.

- NAME: 생성할 Credential 의 이름 지정
- CREDENTIAL TYPE: Machine
- USERNAME: controller 에서 사용하는 username
- SSH PRIVATE KEY: controller 에서 사용하는 SSH private key 값
- PRIVILEGES ESCALATION METHOD: sudo (본 인증을 통해 지정할 권한상승 유형을 의미)

D. 우측 하단의 save 버튼 클릭하여 저장한다.

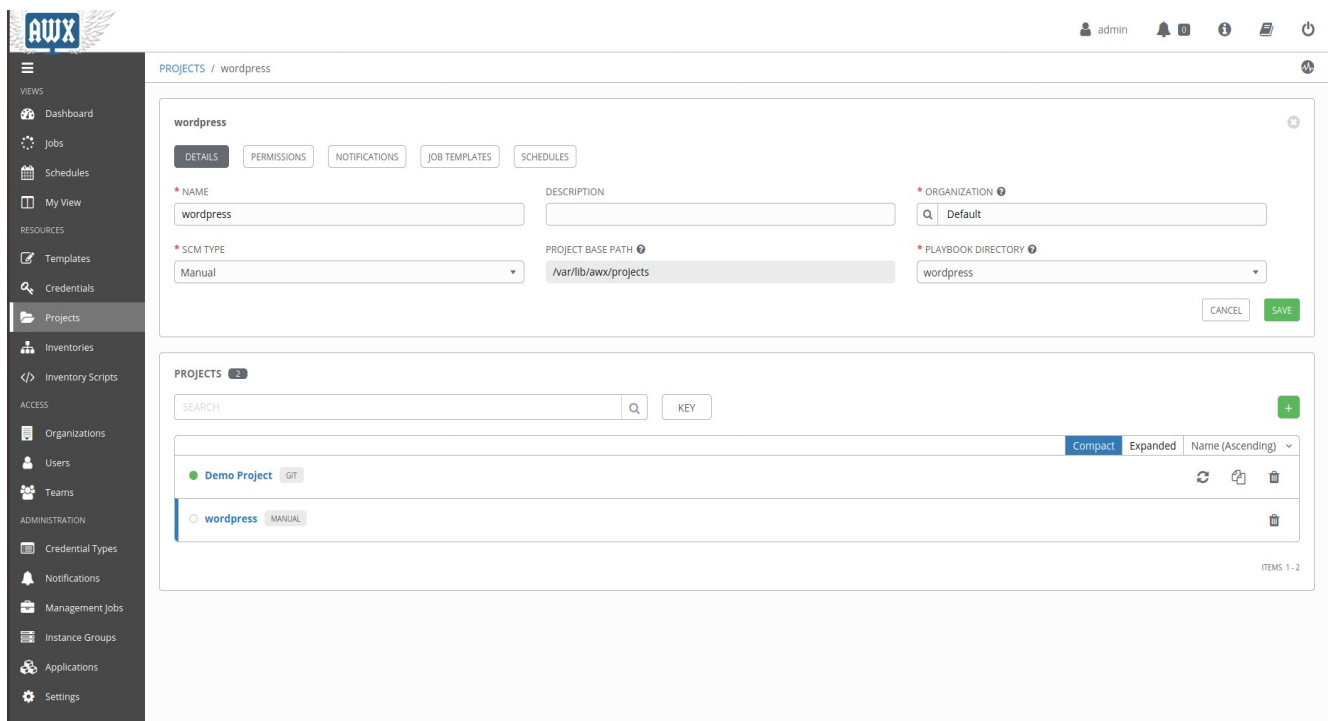
(2) Vault

본 서비스구축에서 database 와 관련된 root 패스워드, user 패스워드에 대한 변수는 ansible-vault 를 이용하여 암호화하였다. 해당 변수를 AWX 에서 사용하기 위해서 vault 에 대한 인증설정이 필요하다.

AWX 에서 vault 인증을 추가하는 방법은 다음과 같다.

- A. Credential 탭을 클릭한다.
- B. 현재 등록된 Credential 정보가 표기된다. 정보 등록을 위해 우측 상단의 +버튼을 클릭한다.
- C. DETAILS 탭에서 다음 정보를 입력한다.
 - NAME: 생성할 Credential 의 이름 지정
 - CREDENTIAL TYPE: Vault
 - VAULT PASSWORD: Vault 인증파일 생성시 지정했던 패스워드
- D. 우측 하단의 save 버튼 클릭하여 저장한다.

2) Project



Project 는 AWX 에서 사용할 playbook 의 모음이다. project 를 생성하는 방법은 다음과 같다.

A. project 생성에 앞서 playbook 을 관리할 디렉토리를 생성한다. 디렉토리 경로는 awx 설치 시 inventory 에 지정된 /var/lib/awx/projects/이다.

```
$ sudo mkdir /var/lib/awx/projects/awx-project
```

B. project 생성을 위해 AWX 대시보드에서 Project 탭을 클릭한다.

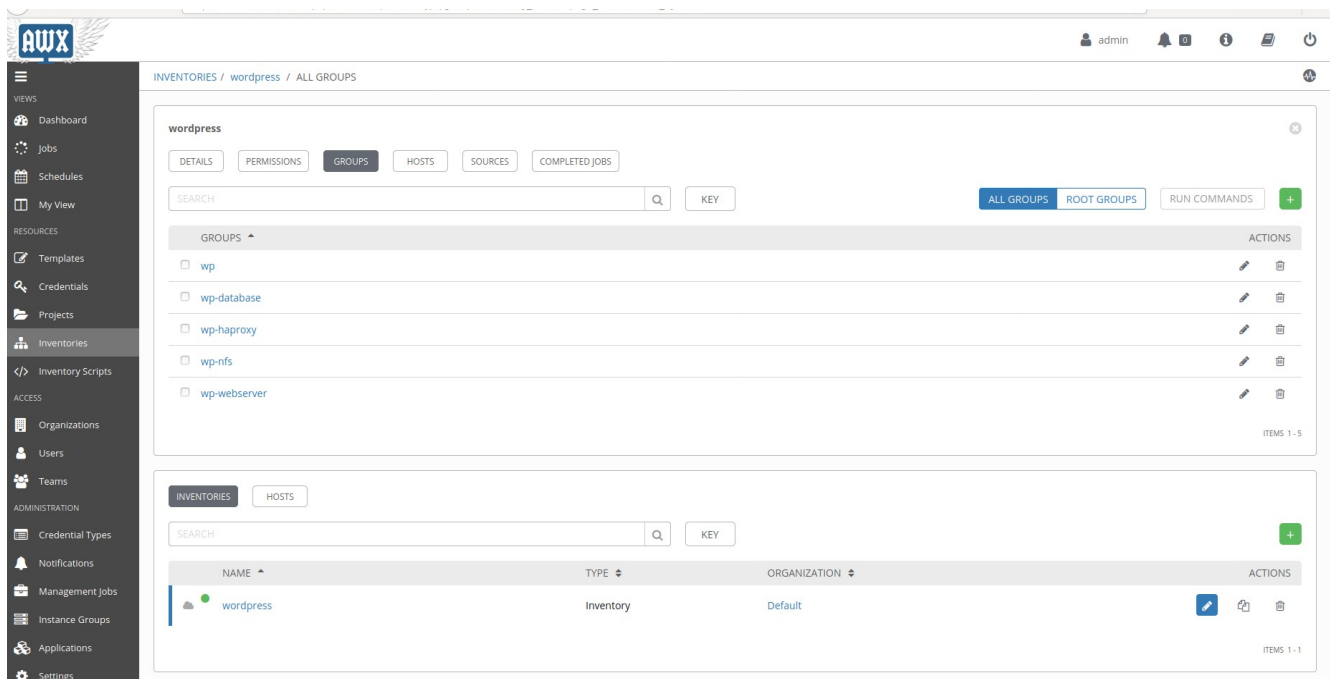
C. 현재 등록된 Project 정보가 표기된다. Project 등록을 위해 우측 상단의 +버튼을 클릭한다.

D. DETAILS 탭에서 다음 정보를 입력한다.

- NAME: 생성할 Project 의 이름 지정
- ORGANIZATION: Default
- SCM TYPE: Manual (Source Code Management Type 을 지정)
- PROJECT BASE PATH: 앞서 생성한 playbook 관리 디렉토리를 지정

E. 우측 하단의 save 버튼을 클릭하여 저장한다.

3) Inventory



AWX 로 playbook 관리할 경우 대시보드에서 inventory 생성 및 관리가 가능하며, 기존 방식처럼 인벤토리 파일을 별도 생성하여 관리할 필요가 사라진다. Inventory 를 생성하는 방법은 다음과 같다.

A. Inventory 탭을 클릭한다.

B. 현재 등록된 Inventory 정보가 표기된다. Inventory 등록을 위해 우측 상단의 +버튼을 클릭 후 [Inventory]를 선택한다.

C. DETAILS 탭에 다음 정보를 입력한다.

- NAME: 생성할 Inventory 의 이름
- ORGANIZATION: Default

D. save 버튼을 클릭하여 Inventory 를 저장한다.

E. 그룹을 추가하기 위해 상단의 GROUPS 를 클릭한다.

F. 상단의 +버튼을 클릭한다.

G. 모든 그룹을 포함시킬 wp 라는 이름의 그룹을 생성한다.

H. wp 그룹 탭에서 GROUPS 클릭 후 +버튼으로 하위 그룹을 생성할 수 있다.

I. wp 의 하위그룹 wp-database, wp-haproxy, wp-nfs, wp-webserver 를 생성한다.

J. 각 하위그룹의 탭으로 이동하여 HOSTS 버튼을 클릭 후, 그룹에 할당할 호스트를 생성해준다.
각 호스트의 변수란에 ansible_host: [host_ip] 형태로 모든 호스트를 매핑시켜준다.

4) Templates

The screenshot shows the 'NEW JOB TEMPLATE' form in the AWX interface. The form is organized into a grid of input fields and checkboxes. The left sidebar contains a navigation menu with categories like VIEWS, RESOURCES, ACCESS, and ADMINISTRATION. The 'TEMPLATES / CREATE JOB TEMPLATE' header is visible at the top of the main content area. The form fields include:

- NAME:** Text input field.
- DESCRIPTION:** Text input field.
- JOB TYPE:** Dropdown menu with 'Run' selected.
- INVENTORY:** Text input field with a search icon.
- PROJECT:** Text input field with a search icon.
- CREDENTIALS:** Text input field with a search icon.
- FORKS:** Text input field with a dropdown arrow.
- LIMIT:** Text input field with a dropdown arrow.
- VERBOSITY:** Text input field with a dropdown arrow.
- JOB TAGS:** Text input field with a dropdown arrow.
- SKIP TAGS:** Text input field with a dropdown arrow.
- LABELS:** Text input field.
- INSTANCE GROUPS:** Text input field with a search icon.
- JOB SLICING:** Text input field with a dropdown arrow.
- TIMEOUT:** Text input field with a dropdown arrow.
- SHOW CHANGES:** Toggle switch.
- OPTIONS:** Checkboxes for 'ENABLE PRIVILEGE ESCALATION', 'ENABLE PROVISIONING CALLBACKS', 'ENABLE WEBHOOK', 'ENABLE CONCURRENT JOBS', and 'ENABLE FACT CACHE'.
- EXTRA VARIABLES:** Section with 'YAML' and 'JSON' tabs and a text area for variables.

AWX 의 템플릿은 프로젝트 별 playbook 을 관리하고 실행 가능한 리소스이다. AWX 의 템플릿은 playbook 의 템플릿과 이름은 동일하나, 전혀 다른 형태의 리소스이다.

생성한 템플릿으로 지정한 playbook 을 실행 가능하며, 매번 플레이는 JOB 이라는 형태로 실행된다.

templates 을 이용하여 playbook 을 등록하는 과정은 다음과 같다.

A. Templates 탭을 클릭한다.

B. 현재 등록된 템플릿 정보가 표기된다. 템플릿 등록을 위해 우측 상단의 +버튼을 클릭 후 [Job Templates]를 선택한다.

C. DETAILS 탭에 다음 정보를 입력한다.

- NAME: 생성할 Templates 의 이름
- JOB TYPE^{*} : Run

- * 1. Run: 템플릿을 바로 실행하며 노드에 변경사항 발생.
- 2. Check: 변경 사항을 보고하나, 실제 노드에 반영되지는 않음)

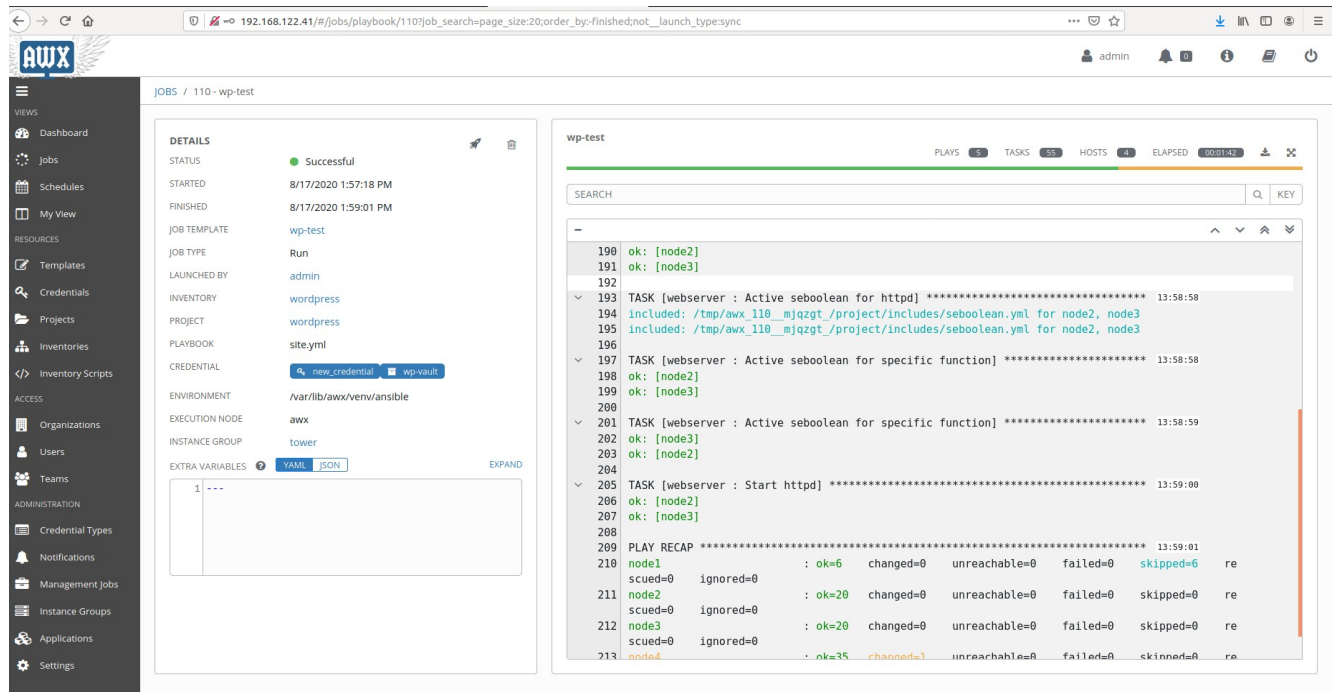
- INVENTORY: 실행대상 인벤토리를 선택
- PROJECT: 실행대상 프로젝트를 선택
- PLAYBOOK: 실행하고자 하는 playbook 선택
- CREDENTIAL:
 - 1. Machine 탭의 ssh-key
 - 2. Vault 탭의 vault 인증정보

D. save 버튼을 클릭하여 Templates 를 저장한다.

E. lunch 버튼을 클릭하여 Job 을 실행시킨다.

lunch 시 jobs 탭으로 자동이동되며, 우측 상태창에는 play 진행상황이 표시된다.
성공 혹은 실패여부에 따라 status 가 success 혹은 failed 로 표기된다.

6. 구성 결과



templates 의 런칭이 성공할 경우 DETAILS 의 STATUS 상태가 Successful 로 변경된다.

```
student@node1:~$ systemctl status haproxy
● haproxy.service - HAProxy Load Balancer
   Loaded: loaded (/lib/systemd/system/haproxy.service; enabled; vendor preset: enabled)
   Active: active (running) since Tue 2020-08-18 09:14:48 UTC; 46s ago
     Docs: man:haproxy(1)
           file:///usr/share/doc/haproxy/configuration.txt.gz
   Process: 944 ExecStartPre=/usr/sbin/haproxy -f $CONFIG -c -q $EXTRA_OPTS (code=exited, status=0/SUCCESS)
   Main PID: 1047 (haproxy)
     Tasks: 2 (limit: 2317)
   CGroup: /system.slice/haproxy.service
           └─1047 /usr/sbin/haproxy -Ws -f /etc/haproxy/haproxy.cfg -p /run/haproxy.pid
           └─1050 /usr/sbin/haproxy -Ws -f /etc/haproxy/haproxy.cfg -p /run/haproxy.pid

Aug 18 09:14:48 node1 systemd[1]: Starting HAProxy Load Balancer...
Aug 18 09:14:48 node1 haproxy[1047]: Proxy main started.
Aug 18 09:14:48 node1 haproxy[1047]: Proxy wp started.
Aug 18 09:14:48 node1 haproxy[1047]: Proxy wp started.
Aug 18 09:14:48 node1 systemd[1]: Started HAProxy Load Balancer.
```

```
student@node2:~$ systemctl status httpd
● httpd.service - The Apache HTTP Server
   Loaded: loaded (/usr/lib/systemd/system/httpd.service; enabled; vendor preset: disabled)
   Active: active (running) since Tue 2020-08-18 18:15:04 KST; 2min 43s ago
     Docs: man:httpd(8)
           man:apachectl(8)
   Main PID: 1242 (httpd)
   Status: "Total requests: 0; Current requests/sec: 0; Current traffic: 0 B/sec"
     CGroup: /system.slice/httpd.service
           └─1242 /usr/sbin/httpd -DFOREGROUND
           └─1490 /usr/sbin/httpd -DFOREGROUND
           └─1491 /usr/sbin/httpd -DFOREGROUND
           └─1493 /usr/sbin/httpd -DFOREGROUND
           └─1494 /usr/sbin/httpd -DFOREGROUND
           └─1495 /usr/sbin/httpd -DFOREGROUND

Aug 18 18:14:59 node2 systemd[1]: Starting The Apache HTTP Server...
Aug 18 18:15:04 node2 httpd[1242]: AH00558: httpd: Could not reliably determine the server's fully qualified domain name,
Aug 18 18:15:04 node2 systemd[1]: Started The Apache HTTP Server.
```

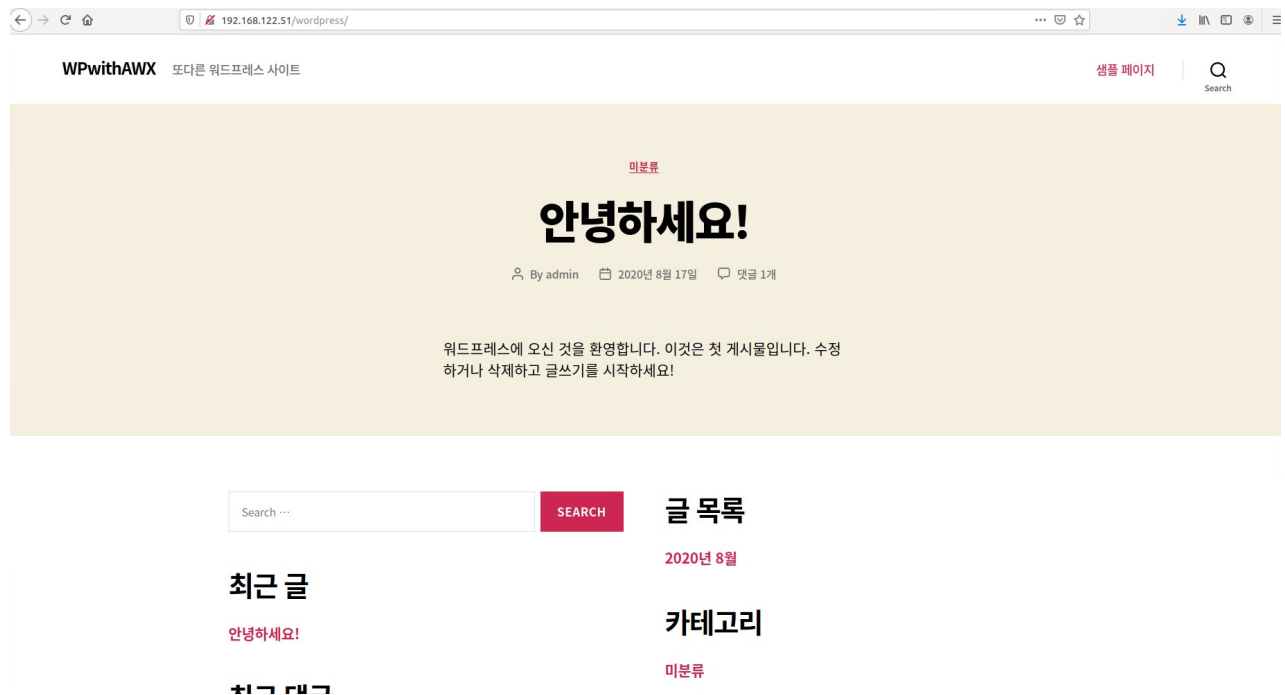
```
student@node3:~$ systemctl status httpd
● httpd.service - The Apache HTTP Server
   Loaded: loaded (/usr/lib/systemd/system/httpd.service; enabled; vendor preset: disabled)
   Active: active (running) since Tue 2020-08-18 18:15:02 KST; 5min ago
     Docs: man:httpd(8)
           man:apachectl(8)
   Main PID: 1185 (httpd)
   Status: "Total requests: 0; Current requests/sec: 0; Current traffic: 0 B/sec"
     CGroup: /system.slice/httpd.service
           └─1185 /usr/sbin/httpd -DFOREGROUND
           └─1575 /usr/sbin/httpd -DFOREGROUND
           └─1577 /usr/sbin/httpd -DFOREGROUND
           └─1578 /usr/sbin/httpd -DFOREGROUND
           └─1579 /usr/sbin/httpd -DFOREGROUND
           └─1580 /usr/sbin/httpd -DFOREGROUND

Aug 18 18:14:55 node3 systemd[1]: Starting The Apache HTTP Server...
Aug 18 18:15:00 node3 httpd[1185]: AH00558: httpd: Could not reliably determine the server's fully qualified domain name,
Aug 18 18:15:02 node3 systemd[1]: Started The Apache HTTP Server.
```

```
student@node4:~$ systemctl status mariadb
● mariadb.service - MariaDB 10.5.5 database server
   Loaded: loaded (/etc/systemd/system/mariadb.service; enabled; vendor preset: disabled)
   Drop-In: /etc/systemd/system/mariadb.service.d
           └─mariadb.conf.d/60-mysqld.conf
   Active: active (running) since Tue 2020-08-18 18:15:04 KST; 6min ago
     Docs: man:mariadb(8)
           https://mariadb.com/kb/en/library/systemd/
   Process: 1875 ExecStartPre=/bin/sh -c systemctl unset-environment _WOREP_START_POSITION (code=exited, status=0/SUCCESS)
   Status: "Taking your SQL requests now..."
     CGroup: /system.slice/mariadb.service
           └─1875 /usr/sbin/mariadb

Aug 18 18:15:04 node4 mariadb[1875]: 2020-08-18 18:15:03 0 [Note] InnoDB: 10.5.5 started; log sequence number 837663; transaction id 630
Aug 18 18:15:04 node4 mariadb[1875]: 2020-08-18 18:15:03 0 [Note] InnoDB: Loading buffer pool(s) from /var/lib/mysql/ib_buffer_pool
Aug 18 18:15:04 node4 mariadb[1875]: 2020-08-18 18:15:03 0 [Note] Plugin 'FEEDBACK' is disabled.
Aug 18 18:15:04 node4 mariadb[1875]: 2020-08-18 18:15:04 0 [Note] Server socket created on IP: 192.168.122.54
Aug 18 18:15:04 node4 mariadb[1875]: 2020-08-18 18:15:04 0 [Note] InnoDB: Buffer pool(s) load completed at 202018 18:15:04
Aug 18 18:15:04 node4 mariadb[1875]: 2020-08-18 18:15:04 0 [Note] Reading of all Master info entries succeeded
Aug 18 18:15:04 node4 mariadb[1875]: 2020-08-18 18:15:04 0 [Note] /usr/sbin/mariadb: ready for connections.
Aug 18 18:15:04 node4 mariadb[1875]: Version: '10.5.5-MariaDB' socket: '/var/lib/mysql/mysql.sock' port: 3306 MariaDB Server
Aug 18 18:15:04 node4 systemd[1]: Started MariaDB 10.5.5 database server.
```

ssh 로 각 node 에 접속 후 서비스 상태를 확인시 active 상태임을 확인할 수 있다.



웹브라우저로 node1(haproxy)의 ip 로 접근 시 Wordpress 사이트가 생성됨을 확인할 수 있다.

7. 결론

Ansible-playbook 과 AWX 를 활용하여 과거에는 엔지니어가 수동으로 구성해야 했던 인프라를 쉽게 구성 및 관리할 수 있었다. 수십에서 수백대의 서버에 대한 관리가 필요할 때 Ansible 은 유효한 툴이 될 수 있을것이라 느꼈다.

단 본 서비스 구축과정에서 경험한 어려움도 있었다. 먼저 templates, handler, group_vars 등 리소스 없이 task 만으로도 인프라 구성 및 관리하는 경우를 생각해보자. 구성이 가능할 수도 있지만 특정 기능을 구현하는데 한계가 있을 수 있고, 역등성의 문제가 발생할 가능성이 크다.

이와 같은 리소스를 구성하는 방식 중 하나로, 인프라 구성 및 관리가 가능한 task 를 만든 뒤 별도로 관리해야 하는 부분만 별도의 리소스로 분리하는 방식을 사용하였다. 하지만 이 방식은 task 가 하나만 존재할 때는 좋은 방법이었지만, task 를 여러개로 구성해야 할 때 중첩되는 리소스가 발생하고 의존성 문제로 task fail 이 발생하는 등의 문제를 겪었다.

이와 같은 문제를 겪지 않기 위해선 playbook 구성 이전에 구성, 관리하고자 하는 아키텍처를 이해하고 전체적인 to-do list 를 작성 후 필요한 inventory, roles, 리소스 등을 구성하는 거시적인 접근이 필요하다고 생각한다.