КУПЛЕНО НА SKLADCHIK.COM

## 5.2

Давайте попробуем установить зависимости — роли и коллекции — с помощью Ansible Galaxy.

Роль — набор тасков и служебных файлов, позволяющих установить или настроить конкретную утилиту, или приложение. Например, установка postgresql и его изначальная настройка.

Коллекция может включать служебные модули и роли для работы с экосистемой софта. Например, все операции с postgresql, а не только его установка.

Ansible Galaxy может нам в этом помочь, скачать роли и коллекции из официального репозитория Ansible Galaxy или даже с репозитория github.

Установка ролей через Ansible Galaxy производится с помощью команды ansible-galaxy install.

```
ansible-galaxy install rolespace.role - скачает и установит роль
ansible-galaxy collection install my_namespace.my_collection - скачает и установит коллекцию
```

Чтобы проапгрейдить коллекцию вы можете использовать команду **--upgrade**:

```
ansible-galaxy collection install my_namespace.my_collection --upgrade
```

Для установки нескольких ролей и коллекций, а также чтобы не вводить их вручную, мы можем создать файл зависимостей и объявить там все. Давайте сделаем файл для установки зависимостей нашего проекта.

```
roles:
- src: geerlingguy.postgresql

- name: green.mongodb
  src: https://github.com/UnderGreen/ansible-role-mongodb

collections:
- name: community.postgresql
```

Вы можете посмотреть его в [https://gitlab.slurm.io/edu/ansible_course/-/tree/master/4.other-apps-scenarios/ansible/requirements.yml](https://gitlab.slurm.io/edu/ansible_course/-/tree/master/4.other-apps-scenarios/ansible/requirements.yml)

И установим все с помощью команды:

```
ansible-galaxy install -r requirements.yml - установит роли
ansible-galaxy collection install -r requirements.yml - установит коллекции из файла
```

Посмотреть установленные роли/коллекции можно командой:

```
ansible-galaxy list - покажет список ролей
ansible-galaxy collection list - покажет список коллекций
```

# Использование чужих ролей и модулей Ansible Galaxy

Находимся в ранее созданной папке **C:\Users\User\vagrant** откуда запускали установку машин, и удаляем их.

```
vagrant destroy
```

Чистим всё:

```
rm -rf *
```

Копируем репозиторий с материалом в **C:\Users\User\vagrant**

```
cp -R C:\Users\User\ansible_course\4.other-apps-scenarios\* .
```

Текущая конфигурация установит наименование/система/IP-адрес.

| | | |
|---|---|---|
| controlnode | ubuntu/focal64 | 192.168.50.4 |
| server_postgres_master | bento/centos-7.5 | 192.168.51.2 |
| server_postgres_slave | bento/centos-7.5 | 192.168.51.3 |
| server_mongo_primary | bento/centos-7.5 | 192.168.52.2 |
| server_mongo_secondary | bento/centos-7.5 | 192.168.52.3 |
| server_mongo_arbiter | bento/centos-7.5 | 192.168.52.4 |
| server_docker_builder | bento/centos-7.5 | 192.168.53.2 |
| server_docker_runner | bento/centos-7.5 | 192.168.53.3 |

Сложим парные ключи ssh в папку **files**.

```
ssh-keygen

Enter file in which to save the key
```

Укажем полный путь до папки files и наименование ключа. Это наименование задано в раскладке по серверам в файле Vagrantfile. Если будете задавать другое наименование ключа, то надо сменить в файле Vagrantfile на это наименование.

```
C:\Users\User\vagrant\files\vagrant_test
```

Запустим только контролноду.

```
vagrant up controlnode
```

Можете воспользоваться **Vagrantfile-linux**, который раскладывает ssh ключи через ansible если у вас Linux машина.

Заходим в controlnode:

```
vagrant ssh controlnode
```

Можем ознакомится с хэлпом команды ansible-galaxy:

```
ansible-galaxy -h
```

Смотрим, что можем сделать с ролью:

```
ansible-galaxy role -h
```

Перейдем в расшаренную папку ansible. Создаем роль **test** и заходим в созданную папку **test** этой командой:

```
cd ansible/

ansible-galaxy role init test

cd test/
```

Смотрим содержимое. Как видим, тут всё разложено по папкам переменные, инвентари, таски и так далее.

```
ls -l

total 4
-rwxrwxrwx 1 vagrant vagrant 1328 Sep  2 08:57 README.md
drwxrwxrwx 1 vagrant vagrant    0 Sep  2 08:57 defaults
drwxrwxrwx 1 vagrant vagrant    0 Sep  2 08:57 files
drwxrwxrwx 1 vagrant vagrant    0 Sep  2 08:57 handlers
drwxrwxrwx 1 vagrant vagrant    0 Sep  2 08:57 meta
drwxrwxrwx 1 vagrant vagrant    0 Sep  2 08:57 tasks
drwxrwxrwx 1 vagrant vagrant    0 Sep  2 08:57 templates
drwxrwxrwx 1 vagrant vagrant    0 Sep  2 08:57 tests
drwxrwxrwx 1 vagrant vagrant    0 Sep  2 08:57 vars
```

Более наглядно можно увидеть через утилиту tree, но её нужно будет установить.

```
tree

.
├── README.md
├── defaults
│   └── main.yml
├── files
├── handlers
│   └── main.yml
├── meta
│   └── main.yml
├── tasks
│   └── main.yml
├── templates
├── tests
│   ├── inventory
│   └── test.yml
└── vars
    └── main.yml
```

Теперь давайте посмотрим, как роли могут нам пригодится.

Здесь можем обратить внимание на файл requirements.yml.

```
cd ..
cat requirements.yml

roles:
- src: geerlingguy.postgresql

- name: green.mongodb
  src: https://github.com/UnderGreen/ansible-role-mongodb

collections:
```

```
- name: community.postgresql
```

Устанавливаем роль на прямую из galaxy - **- src: geerlingguy.postgresql**

Устанавливаем роль из github ресурса - **- name: green.mongodb**

Устанавливаем коллекцию - **- name: community.postgresql**. Теперь нам не нужно предварительно устанавливать данную коллекцию перед запуском плейбука, как мы делали это раньше.

Установим роли и потом коллекцию.

```
ansible-galaxy install -r requirements.yml

...
- green.mongodb was installed successfully

ansible-galaxy collection install -r requirements.yml

...
Installing 'community.postgresql:1.4.0' to
'/home/vagrant/.ansible/collections/ansible_collections/community/postgresql'
```

Всё это устанавливается в домашнюю директорию того пользователя, от которого запускалась команда - ~/.ansible. Там лежат установленные роли и коллекции. Если по данному пути не было прав установить, то установит по пути /usr/share/, если не сможет, то /ent/ansible/. Либо просто отдаст ошибку.

5.3

# Установка PostgreSQL с помощью Ansible

Выйдем из контролноды и запустим установку двух машин для текущей практики.

```
exit

vagrant up server_postgres_master server_postgres_slave
```

Заходим обратно на контролноду и в расшаренную папку **ansible**.

```
vagrant ssh controlnode

cd /ansible
```

Теперь с помощью ролей и коллекций, которые мы скачали ранее, установим и настроим postgresql в комплектации master и одна replica. Для этого используем следующий playbook:  **postgres-playbook.yml**.

Обращаем внимания на два конфигурационных файла, которые были в видео.

```
cat postgres-playbook.yml
```

Обращаем внимание на состав файла **hosts.ini** для данного плейбука.

```
cat hosts
```

Запускаем плейбук **postgres-playbook.yml**. Предварительно скопируем папку ansible, иначе не сработает конфиг **ansible.cfg**.

```
cd ..
cp -R ansible internal_ansible
cd internal_ansible/
ansible-playbook postgres-playbook.yml

...
TASK [Include server to production]
*********************************************************************
ok: [postgres_master -> 127.0.0.1]
ok: [postgres_slave -> 127.0.0.1]
PLAY RECAP
*********************************************************************
*************
postgres_master            : ok=32   changed=16   unreachable=0    failed=0
skipped=2    rescued=0    ignored=0
postgres_slave             : ok=35   changed=19   unreachable=0    failed=0
skipped=2    rescued=0    ignored=0
```

Выходим из контрлноды. Останавливаем (**suspend**) или удаляем (**destroy**) машины.

```
vagrant suspend server_postgres_master server_postgres_slave
```

Что нужно использовать, чтобы следующие блоки текста не перезаписали друг друга в /.bashrc.

```
- name: Setup java environment
  blockinfile:
    dest: /home/{{ user }}/.bashrc
    block: |
      #Java path#
      JAVA_HOME={{ java_home }}/

- name: Setup apache environment
  blockinfile:
    dest: /home/{{ user }}/.bashrc
    block: |
      #Apachepath#
      APACHE_WEB_ROOT={{ apache_home }}/
```

5.4

# Установка Mongo Cluster с помощью Ansible

Выйдем из контролноды и запустим установку трёх машин для текущей практики.

```
exit
```

```
vagrant up server_mongo_primary server_mongo_secondary server_mongo_arbiter
```

Зайдём обратно на контролноду и в скопированную папку **internal_ansible**.

```
vagrant ssh controlnode

cd /internal_ansible
```

Воспользуемся готовой ролью для установки MongoDB в комплектации master, slave и arbiter, чтобы найти, какая нода сейчас мастер. Можно увидеть, что из-за переиспользования ролей, базовая установка MongoDB заняла совсем немного кода.

```
cat mongo-playbook.yml
```

Файлик **hosts** используется тот же самый, поэтому для этого упражнения мы подняли три виртуалки из Vagrantfile.

```
cat hosts
```

Запускаем плейбук **mongo-playbook.yml**.

```
ansible-playbook mongo-playbook.yml

...
TASK [Include server to production]
*********************************************************************
ok: [mongo_primary -> 127.0.0.1]
ok: [mongo_secondary -> 127.0.0.1]
ok: [mongo_arbiter -> 127.0.0.1]
PLAY RECAP
*********************************************************************
********************
mongo_arbiter              : ok=25   changed=10   unreachable=0    failed=0
skipped=27   rescued=0    ignored=0
mongo_primary              : ok=25   changed=11   unreachable=0    failed=0
skipped=27   rescued=0    ignored=0
mongo_secondary         : ok=25   changed=10   unreachable=0    failed=0    skipped=27
rescued=0    ignored=0
```

Выходим из контрлноды. Останавливаем (**suspend**) или удаляем (**destroy**) машины.

```
vagrant suspend server_mongo_primary server_mongo_secondary server_mongo_arbiter
```

5.5

# Собираем docker контейнеры с помощью Ansible

Выйдем из контролноды и запустим установку двух машин для текущей практики.

```
exit

vagrant up server_docker_builder server_docker_runner
```

Заходим обратно на контролноду и в скопированную папку **internal_ansible/Docker**.

```
vagrant ssh controlnode

cd internal_ansible/Docker/
```

Ansible умеет работать с Docker, и в этом шаге мы увидим, как собирать контейнеры и запускать их с помощью Docker. Для этого нужно  установить Docker и Docker SDK на машины, собирающие Docker контейнеры и запускающие контейнеры.

Playbook включает в себя подфайлы, позволяющие нам разбить наборы тасок на логические шаги.

```
cat docker-playbook.yml
```

Посмотрим на содержимое файла **container_assembly.yml** Он собирает один докер контейнер для каждого из докерфайлов и собирает архив для запуска в других системах.

```
cat container_assembly.yml
```

Второй файл запускает докер образы в докере на второй машине **container_load.yml**

```
cat container_load.yml
```

Запускаем плейбук **docker-playbook.yml**

```
ansible-playbook docker-playbook.yml

...
TASK [Copy files and build them]
*********************************************************************
TASK [Healthcheck]
*********************************************************************
********
ok: [runner_host]
PLAY RECAP
*********************************************************************
***************
build_host                 : ok=21   changed=16   unreachable=0    failed=0
skipped=0    rescued=0    ignored=0
runner_host                : ok=8    changed=5    unreachable=0    failed=0
skipped=1    rescued=0    ignored=0
```

Выходим из контрлноды. Заходим на раннер и проверяем контейнеры.

```
exit
vagrant ssh server_docker_runner
sudo docker ps

CONTAINER ID   IMAGE                 COMMAND                  CREATED          STATUS
PORTS      NAMES
ca9cc7f2a851   db_container:v1.0     "/bin/sh -c 'nc -l -…"   12 seconds ago   Up 11
seconds   8080/tcp   db_app
5288521862b6   web_container:v1.0    "/bin/sh -c 'nc -l -…"   26 seconds ago   Up 22
seconds   8080/tcp   web_app
```

Выходим и останавливаем **suspend** или удаляем **destroy** машины.

```
exit
vagrant suspend server_docker_builder server_docker_runner
```

5.8

Сопоставьте таски, необходимые для следующих действий:

Заменить блок авторизации для **ssh**;
Установить пакет **git**;
Склонировать репозиторий от этого пользователя;
Создать пользователя линукс для авторизации.

A)

```
- name: Task
  ansible.builtin.package:
    name: git
    state: present
```

B)

```
- name: Task
   ansible.builtin.lineinfile:
     path: "/etc/ssh/sshd_config"
     regexp: "PasswordAuthentication no"
     line: "PasswordAuthentication yes"
```

C)

```
- name: Task
  ansible.builtin.user:
    name: devuser
    shell: /bin/bash
    groups: developers
    append: yes
```

D)

```
- name: task
  git: repo=https://github.com/some/repo.git version=master     dest=/var/www/dev
  become: yes
  become_user: devuser
```

Какие ошибки есть в этом плейбуке?

```
---
- hosts: all
 vars:
   replica_password: "password"
   postgres_master_ip: "192.178.96.1"

 pre_tasks:
   - name: "Exclude server from production"
     delegate_to: 127.0.0.1
     script: '/home/vagrant/exclude_from_prod.py'
     register: command_result
     failed_when: "'error' in command_result.stdout"
```

```
    changed_when: "'success' in command_result.stdout"


post_tasks:
  - name: "Include server to production"
    delegate_to: 127.0.0.1
    script: '/home/vagrant/include_to_prod.py {{ansible_host}}'
    register: command_result
    failed_when: "'error' in command_result.stdout"
    changed_when: "'success' in command_result.stdout"


tasks:
  - name: Move data folder
    shell: mv data data-backup
    args:
      chdir: "{{ data_dir }}"

  - name: "Moving data"
    block:
      - name: Backup initial data from master
        shell: "PGPASSWORD={{ replica_password }} pg_basebackup -w -h {{
postgres_master_ip }} -U repluser -D {{postgresql_data_dir}} -P --xlog"
        become_user: "postgres"
        become: true

    always:
        ansible.builtin.file:
          path: "{{ postgresql_data_dir }}/../data-backup"
          state: absent

 roles:
   - green.mongodb
```

Какой будет результат запуска Jinja шаблона из плейбука:

```
---
- name: "Jinja run"
 hosts: localhost
 vars:
   addon_host: myhost.com
   addon_login: root
   params:
     block_name: main_block
     block_flags: pph, szh
     flags:
       - zsh
       - psh
       - ssh
       - aux

tasks:
- name: Copy service file
  ansible.builtin.template:
    src: config.txt.j2
    dest: ./config.txt
```

Config.txt.j2

```
flags = {% for item in params.flags -%}
   {{ item }},
{%- endfor %}

internal_template = '{%- raw %}
   {% for item in seq %}
       {{ item.name }} = {{item.param}}
   {% endfor %}
{%- endraw %}'


additional_flags = {%- for item in params.additional_flags | default([]) %}
{{ item }},
{%- else %}
 default
{%- endfor %}

{% set blocks %}
block_name = {{ params.block_name }}
block_flags = {{ params.block_flags }}
{% endset %}


{{ blocks }}


{% include './templates/addons.j2' %}
```

templates/addons.j2

```
addon_host = {{ addon_host }}
addon_login = {{ addon_login }}
```

В окно ответа, вставьте ваш ответ без пустых строк.

Что будет в файле в результате выполнения данного плейбука?

```
---
- name: "Replace strings"
  hosts: localhost
  vars:
    servers:
    - name: test1
      ip: 3.158.67.55
    - name: test2
      ip: 3.34.85.53
    - name: test3
```

```
    ip: 31.4.87.31
  other_servers:
  - name: test1
    ip: 31.11.33.21
  - name: test2
    ip: 31.11.54.33
  - name: test3
    ip: 55.11.44.33
tasks:
  - name: "Update postgresql.conf"
    lineinfile:
      path: "config.ini"
      regexp: "{{ item.regexp }}"
      line: "{{ item.line }}"
    with_items:
      - regexp: "frequency = 15m"
        line: "frequency = 1h"
      - regexp: "frequency = 5m"
        line: "frequency = 2h"

- name: "Update pg_hba.conf"
  blockinfile:
    path: "./config.ini"
    insertafter: "### hosts are here ###"
    block: |
      host replication repluser 127.0.0.1/32 md5
      host replication repluser 192.168.0.1/32 md5
      host main 192.168.10.1/32 md5
      host postgres 192.168.11.0/24 md5

- name: "Insert servers"
    blockinfile:
      path: "./config.ini"
      block: |
        host main {{item.name}} {{item.ip}}/32 md5
    loop: "{{ servers }}"

  - name: "Insert servers"
    blockinfile:
      path: "./config.ini"
      marker: "### other server {{ item.name }}"
      block: |
        host main {{item.name}} {{item.ip}}/32 md5
    loop: "{{ other_servers }}"
```

Оригинальный файл config.ini:

```
frequency = 15m
### hosts are here ###
```

5.9

## Создаем стенд

Текущий стенд состоит из 2-х узлов: `node-1.sXXXXXX`, `node-2.sXXXXXX` (XXXXXX - ваш номер студента). Работа стенда 6 часов. 2 попытки запуска стенда.

Что уже установлено на стенде?

- node-1: ansible2.9, python2.7\3.6, molecule, docker sdk, docker
- node-2: python2.7\3.6

Какой IP-адрес у них?

1 и 4 октет сети у всех одинаковый -**172.XX.XXX.6** (node-1) и **172.XX.XXX.7** (node-2). 2 и 3 узнать через команду `ip -a`. К примеру это **20.247**

```
ip a | grep eth0

inet 172.20.247.6/24 brd 172.20.247.255 scope global eth0
```

1. Над текстом нажмите кнопку "Создать стенд". Каждый стенд создан под определенную тему курса. Сейчас вы находитесь в теме **5. Другие приложения и специфические сценарии.** Этот стенд не подойдет под практические занятия из других пунктов курса.

Запуск обычно идёт до 10 минут, в редких случаях до 30 минут.

2. После создания стенда авторизуйтесь по SSH на adminbox с адресом **sbox.slurm.io** с помощью логина и пароля, находящихся в настройках профиля или над текстом по кнопке "**Доступы**".

3. Далее подключаетесь к первой ноде — контролнода. (XXXXXX - ваш номер студента)

```
ssh node-1.sXXXXXX
```

И повышаем себя до root пользователя.

394783

```
sudo -i
```

Можете приступать к выполнению практических заданий. Желаем успешно выполнить их!

## Мигрируем БД

Пришло время собрать навыки и знания в могучую кучку и проверить себя в настоящем деле.

В мире системного администрирования актуальна задача настройки баз данных. Выполняя настройку важно не поломать систему. Ansible не содержит в себе безусловной защиты, поэтому в некоторых случаях придется страховать себя самостоятельно.

Напишите плейбук, устанавливающий Postgres из роли и накатывающий миграции. Если миграции не установились, необходимо иметь блок, который восстанавливает исходное состояние.

Для установки PostgreSQL используйте следующую роль: https://github.com/geerlingguy/ansible-role-postgresql.

После выполнения роли ваша задача создать базу данных с именем namesdb и запустить миграции из блока ниже. Замените плейсхолдер %user% именем вашего пользователя.

```
create table employees

(
id serial not null
constraint employees_pk
primary key,
name text,
license_plate text
);

alter table employees owner to %user%;
create unique index employees_id_uindex
on employees (id);

INSERT INTO public.employees (id, name, license_plate) VALUES (1, 'John B.',
'DB124GR');
INSERT INTO public.employees (id, name, license_plate) VALUES (2, 'Jill J.',
'VG645PH');
INSERT INTO public.employees (id, name, license_plate) VALUES (3, 'Phil K.',
'GG322GG');
```

## Ненадежный скрипт

Исправьте плейбук с учетом следующих условий: есть ненадежный скрипт вывода сервера из ротации на продакшене. Обратитесь несколько, минимум десять раз для правильного кода ответа, а затем продолжайте выполнять операции (таски).

Скрипт. Лежит на **node-1** по пути **/root/setup.py**

```
#!/usr/bin/env python3

from random import randrange

res = randrange(10)
if randrange(10) > 6:
   print("success")
else:
   print("failure")
```

Обратите внимание, что успех равен ответу “success” в stdout.

```
---
- name: "Exclude hosts from prod"
  hosts: localhost
  gather_facts: true
  pre_tasks:
    - name: "Exclude server from production"
      delegate_to: 127.0.0.1
      script: './exclude_from_prod.py'
      register: command_result
      failed_when: "'failure' in command_result.stdout"
      changed_when: "'success' in command_result.stdout"

 tasks:
   - name: "Print os family"
     ansible.builtin.debug:
       msg: "OS family is {{ansible_os_family}} and python is
{{ansible_python_version}}"
```

# Изменяем данные

Напишите плейбук для изменения данных ниже по тексту.

Исходная конфигурация:

```
mode = default
frequency = 15m
prettify = false
#logs = min
### hosts are here ###
```

Требуемая конфигурация:

```
mode = default
frequency = 1h
prettify = minimal
logs = debug
### hosts are here ###
# BEGIN ANSIBLE MANAGED BLOCK
host replication repluser 127.0.0.1/32 md5
host replication repluser 192.168.0.1/32 md5
host main 192.168.10.1/32 md5
host postgres 192.168.11.0/24 md5
# END ANSIBLE MANAGED BLOCK
```

# Правка плейбука

Исправьте ошибки в плейбуке и запустите его.

```
---
- name: 'Initial install and configuration'
  become: true
  hosts: postgres
  vars:
    postgresql_users:
    - name: postgres
      password: password

 pref_tasks:
   - name: "Exclude server from production"
     delegate_to: 127.0.0.1
     script: '/home/vagrant/exclude_from_prod.py'
     register: command_result
     failed_when: "'error' in command_result.stdout"
     changed_when: "'success' in command_result.stdout"

 roles:
   - geerlingguy.postgresql

 tasks:
   - name: "Create a new database"
     community.postgresql.postgresql_db:
       name: newdb

   - name: "Runnin queries"
      block:

- name: Create a table test table
  community.postgresql.postgresql_table:
    name: test_table
    tablespace: new_tablespace

          - name: Insert query to test_table in db
            community.postgresql.postgresql_query:
              db: test_db
              query: INSERT INTO test_table (id, story) VALUES (2, 'my_long_story')
   rescue:
     - name: Insert query to test_table in db test_db
        community.postgresql.postgresql_query:
          db: newdb
          query: TRUNCATE test_table

- name: Remove a table test_table
  community.postgresql.postgresql_table:
    name: test_table
    tablespace: new_tablespace
    state: absent
```