Различные способы подключения

Скачиваем репозиторий. К примеру в домашнюю папку `C:\Users\User\`

```
git clone git@gitlab.slurm.io:edu/ansible_course.git
```

Создаём папку **vagrant** также в `C:\Users\User\`, куда будем копировать материал для работы.

```
mkdir vagrant
```

Копируем туда материал текущей темы и переходим в неё.

```
cp C:\Users\Aleksgray\ansible_course\3.other-systems-and-stacks\2.Connection\*
.\vagrant\

cd vagrant
```

Запустим установку машин. Текущая конфигурация раскладывает ключи ssh через shell. Текущая конфигурация установит наименование/система/IP-адрес.

controlnode	ubuntu/focal64	192.168.50.4
server	ubuntu/focal64	192.168.51.2
server_no_python	ubuntu/focal64	192.168.51.3
server_centos	bento/centos-7.5	192.168.51.4
server_centos	bento/centos-7.5	192.168.51.5

Сначала создадим папку **files** и туда сложим парные ключи ssh.

```
mkdir files
ssh-keygen

Enter file in which to save the key
```

Если вы счастливый обладатель Windows 7/8 то у вас отсутствует `ssh-keygen`. В данном случае можно установить клиент Git для Windows, добавить в переменные среды -> переменная Path значение: `C:\Programs Files\Git\usr\bin`, после этого в Powershell будет доступен `ssh-keygen`

Укажем полный путь до папки files и наименование ключа. Это наименование задано в раскладке их 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
```

Теперь можно запускать установку виртуальных машин.

```
vagrant up
```

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

Заходим в controlnode

```
vagrant ssh controlnode
```

Можете взглянуть код в файле Vagrantfile в поле объекта **controlnode**.

```
    controlnode.vm.provision "shell", inline: <<-SHELL
      sed -i 's/ChallengeResponseAuthentication no/ChallengeResponseAuthentication
yes/#g' /etc/ssh/sshd_config
      service ssh restart
      curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
      sudo add-apt-repository "deb [arch=amd64]
https://download.docker.com/linux/ubuntu bionic stable"
      sudo apt update && sudo apt --assume-yes install ansible docker-ce docker-
compose sshpass
      chmod 600 /home/vagrant/.ssh/vagrant_test
      chmod 644 /home/vagrant/.ssh/vagrant_test.pub
    SHELL
```

Там прописаны команды устанавливающие заранее программы: ansible, docker-ce, docker-compose, sshpass.

Нижеследующая строка раздает всем объектам публичный ключ внутри config.

```
config.vm.provision "file", source: "files/vagrant_test.pub", destination: "/home/vag
rant/.ssh/"
```

Нижеследующая строка объявлена для объекта **controlnode**, поэтому только туда скопируется приватный ключ.

```
controlnode.vm.provision "file", source: "files/vagrant_test", destination: "/home/va
grant/.ssh/"
```

Можно вызвать документацию ansible, которая поставляется вместе с версией ansible community. Её нет в версии core.

```
ansible-doc
```

К примеру вызвать доку по плагину connection (-t connection) и вывести всё в список (-l).

```
ansible-doc -t connection -l
```

Теперь приступим к проверкам пингов контейнера докер и одного из инстанса. В файле Vagrantfile можно увидеть код в SHELL для controlnode, где был прописан код для установки необходимого ПО, можете ознакомиться.

Заходим на controlnode и запускаем контейнер докер.

```
cd ansible/
sudo docker-compose up -d
```

Проверим

```
sudo docker ps

CONTAINER
ID   IMAGE      COMMAND                  CREATED         STATUS          PORTS     NA
MES
79677ed3e33e   python:3   "python3 /code/test.…"   5 minutes ago   Up 18
seconds             ansible_test_1
```

Раскомментируем `host_key_checking = False` в файле **/etc/ansible/ansible.cfg.** Можете ознакомиться с настройками в файле hosts.ini. Так как запускать команду ansible будем от рута, то **ansible.cfg** лежащий в

домашней папки от vagrant работать не будет. А запускаем от рута, для доступа к докер сервису.

Проверяем пинги через команду ansible и вызов модуля пинг ключом (**-m**).

```
sudo ansible all -m ping -i hosts.ini

remote | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}

pythonlocal | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
```

Теперь пробуем пинги вызвать через playbook

```
sudo ansible-playbook playbook.yml -i hosts.ini

PLAY [all]
*********************************************************************************
************
TASK [Gathering Facts]
*********************************************************************************
ok: [remote]
ok: [pythonlocal]

TASK [ping host]
*********************************************************************************
******
ok: [remote]
ok: [pythonlocal]

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

4.3.2

Ansible.cfg и ключи ssh

Скопируем необходимые файлы для текущей практики в папку **3.other-systems-and-stacks\2.Connection\ansible**

```
cd .\ansible\
cp C:\Users\User\ansible_course\3.other-systems-and-stacks\3.Ssh\* .
```

Заходим в контролноду и в папку ansible.

```
vagrant ssh controlnode

cd ansible/
```

Сначала будем оперировать настройками в файлах **hosts.ini** и **/etc/ansible/ansible.cfg**.

Ранее уже раскомментировали `host_key_checking = False` в файле **/etc/ansible/ansible.cfg.** В hosts.ini только IP-адреса серверов.

Запускаем команду ansible.

```
ansible centos -m ping -i hosts.ini

192.168.51.3 | UNREACHABLE! => {
    "changed": false,
    "msg": "Failed to connect to the host via ssh: Warning: Permanently added
'192.168.51.3' (ECDSA) to the list of known hosts.\r\nvagrant@192.168.51.3:
Permission denied (publickey,password).",
    "unreachable": true
}
192.168.51.5 | UNREACHABLE! => {
    "changed": false,
    "msg": "Failed to connect to the host via ssh: Warning: Permanently added
'192.168.51.5' (ECDSA) to the list of known hosts.\r\nvagrant@192.168.51.5:
Permission denied (publickey,password).",
    "unreachable": true
}
```

Добавляем в файл **hosts.ini** в блоке **centos** после каждого IP-адреса строку `- ansible_private_key_file=/home/vagrant/.ssh/vagrant_test`. Теперь доступ есть.

```
ansible centos -m ping -i hosts.ini

192.168.51.3 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python"
[defaults]
    },
    "changed": false,
    "ping": "pong"
}
192.168.51.5 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": false,
    "ping": "pong"
}
```

Теперь оперируем также с **hosts.ini** и локальным **./ansible.cfg**. Сначала копируем папку ansible смонтированную от windows и именуем в internal_ansible.

```
cd ..
cp -R ansible internal_ansible
```

Даём права 775 на всю папку, иначе не будет работать файл ansible.cfg.

```
chmod -R 775 internal_ansible
```

Переходим в папку **internal_ansible**. Смотрим файл ansible.cfg. Тут объявляем путь до инвентаря, теперь можем не указывать в команде запуска ansible ключ **-i**. Можем убрать пути до ключа каждого хоста в файле hosts.ini, т.к. мы тут это также объявили. Отключаем проверку ключа **host_key_checking**.

```
cd internal_ansible
cat ansible.cfg

[defaults]
inventory=./hosts.ini
private_key_file=/home/vagrant/.ssh/vagrant_test
host_key_checking=false
```

Проверяем пинги.

```
ansible centos -m ping

192.168.51.3 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": false,
    "ping": "pong"
}
192.168.51.5 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": false,
    "ping": "pong"
}
```

4.5.2

Переменные в Ansible

Копируем необходимые для текущей практики файлы в папку **3.other-systems-and-stacks\2.Connection\ansible**

```
cp -R C:\Users\User\ansible_course\3.other-systems-and-stacks\5.Vars .
```

Заходим в контролноду и перекопируем материал в папку internal_ansible.

```
vagrant ssh controlnode
cp -R ansible/5.Vars internal_ansible/
cd internal_ansible/5.Vars
```

Смотрим файл hosts.ini. Можем объявить переменную только одному узлу. Либо одинарный или массив.

```
[centos]
192.168.51.3 test=13123 testvar=["abc","222"]
```

Также объявить какой-то группы, к примеру centos.

```
[centos]
192.168.51.3

[centos:vars]
test=13123
testvar=["abc","222"]
```

Всем группам через all.

```
[all:vars]
test=13123
testvar=["abc","222"]

[centos]
192.168.51.3
192.168.51.5

[ubuntu]
192.168.51.4
```

Смотрим содержание плейбука.

```
cat playbook.yml
```

Запускаем его. Можете раскомментировать таск **name: Print the package facts,** чтобы получить полный вывод фактов о сервере.

```
ansible-playbook playbook.yml
...
TASK [Test vars]
*******************************
ok: [192.168.51.4] => {
    "changed": false,
    "msg": "Variable is set to test {'year': '2021', 'month': '08', 'weekday':
'Friday', 'weekday_number': '5', 'weeknumber': '34', 'day': '27', 'hour': '18',
'minute': '33', 'second': '02', 'epoch': '1630089182', 'date': '2021-08-27', 'time':
'18:33:02', 'iso8601_micro': '2021-08-27T18:33:02.199560Z', 'iso8601': '2021-08-
27T18:33:02Z', 'iso8601_basic': '20210827T183302199377', 'iso8601_basic_short':
'20210827T183302', 'tz': 'UTC', 'tz_offset': '+0000'}"
}
...
PLAY RECAP
*********************************************************************************
***********************************
192.168.51.4               : ok=2    changed=0    unreachable=0    failed=0
skipped=2    rescued=0    ignored=0
```

Как видите 2 таска было пропущено, т.к. переменная version: 7.0 не соотвествует последним двум таскам. Видим время, взятое из фактов сервера.

Теперь давайте попробуем переопределим переменную **ansible_date_time:**. Заходим в плейбук и раскомментируем эту строчку. Запускаем.

```
ansible-playbook playbook.yml
...
TASK [Test vars]
*********************************************************************************
*******************************
ok: [192.168.51.4] => {
```

```
    "changed": false,
    "msg": "Variable is set to test notime"
}
...
192.168.51.4               : ok=2    changed=0    unreachable=0    failed=0
skipped=2    rescued=0    ignored=0
```

Как видим, мы действительно переопредилили переменную.

Теперь мы можем ещё переопределить переменную указанную в плейбуке, при запуске команды ansible-playbook через ключ **-e**.

```
ansible-playbook playbook.yml -e "ansible_date_time=newtime"
...
TASK [Test vars]
**********************************************************************
**************
ok: [192.168.51.4] => {
    "changed": false,
    "msg": "Variable is set to test newtime"
}
...
192.168.51.4               : ok=2    changed=0    unreachable=0    failed=0
skipped=2    rescued=0    ignored=0
```

Также можем ещё переопределить **testvar**.

```
ansible-playbook playbook.yml -e "ansible_date_time=newtime, test_var=newtest"
...
ok: [192.168.51.4]

TASK [Test vars]
**********************************************************************
************
ok: [192.168.51.4] => {
    "changed": false,
    "msg": "Variable is set to newtest newtime,"
}
...
192.168.51.4               : ok=2    changed=0    unreachable=0    failed=0
skipped=2    rescued=0    ignored=0
```

4.6.2

Популярные конструкции jinja

Переменные

Если вы хотите использовать значения раннее определенных переменных в Jinja, вы просто указываете имя переменной в фигурных скобках, например `{{ myvar }}`.

Для переменных в словарях мы пишем значения через точку, например {{ person.name }}. Правилом хорошего тона является ставить отступы перед двойными фигурными скобками и после двойных фигурных скобок.

Если переменной не существует то Jinja просто не выведет ничего. По умолчанию переменные кастуются к самому универсальному типу, то есть если переменной нет, будет использована пустая строка для вывода, что и даст нам “ничего” в

шаблоне.
ВНИМАНИЕ! Для установки переменной воспользуйтесь командой {% set имя переменной=значение %}

For loop

Для вывода значений в цикле, либо отображения всех элементов списка мы можем использовать конструкцию for.

В общем виде выглядит это следующим образом

`{% for item in inventory %}` - начало конструкции, inventory это список

`{{ item.name }}:{{ item.value }}` - указываем имена вложенных объектов через точку, если у вас список из примитивных типов, например строк, item выведет значение

`{% endfor %}` - обязательно закрываем for

ВНИМАНИЕ! При скармливании строки в for, он разберет ее на элементы, например строка “name” породит четыре элемента в цикле -> n a m e
ВНИМАНИЕ! Для вывода индекса в цикле можно использовать конструкцию `{{ loop.index }}`, индекс начинается с 0.

If

Для условного вывода блоков в Jinja вы можете воспользоваться конструкцией if.

`{% if showConfig %}` - showConfig это переменная типа boolean

Here is config

`{% else %}` - опциональный параметр, если if не сработал, имеем шанс вывести вторую ветку

Here is no config

`{% endif %}` - как и с for - закрываем конструкцию

В if можно так же использовать конструкцию in, для проверки, что ключ есть в словаре

```
{% if 'priority' in data %}
  Priority: {{ data['priority'] }}
{% endif %}
```

Для определения что переменная определена есть конструкция is defined, и is not defined в противоположном случае.

```
{% if data is defined %}
  // data goes wshhhhhh
{% endif %}
```

Условия в if можно комбинировать с помощью конструкций булевой алгебры - and и or

And - if сработает если выполнятся ОБА условия в and

```
{% if data is defined and showConfig  %} - showConfig должен быть true И data должна быть определена
  // data goes wshhhhhh
{% endif %}
```

Or - if сработает если выполнится ХОТЯ БЫ ОДНО условие в or

```
{% if data is defined or showConfig  %} - showConfig должен быть true, ИЛИ data должна быть определена
  // data goes wshhhhhh
{% endif %}
```

Эти выражения можно объединить в скобочки для проверок выражений в правильном порядке.

```
{% if (data is defined or showConfig) and if 'priority' in config  %} - showConfig должен быть true, ИЛИ data должна быть определена И ПРИ ЭТОМ priroity в config, без этого у нас читалось бы как data должна быть определена ИЛИ showConfig И priority.
  // data goes wshhhhhh
{% endif %}
```

Default

Одним из атрибутов Jinja в Ansible являются так называемые фильтры - специальные функции, применяемые к переменным. Фильтры применяются к переменным через служебный символ | и возвращают результат уже обработанных переменных.

Одним из часто используемых фильтров в шаблонах является фильтр default

`{{ first_name|default("there") }}` **- если first_name не определен, то выведется there.**

`{{ first_name|default("there", true) }}` **- если first_name не определен ИЛИ определен как пустая строка то выведется there**

Теоретически default можно заменить if c is not defined, но не нужно.

Replace

Фильтр replace служит для замены подстроки в строковом значении, например:

`{{ department|replace("Personnel", "HR") }}` - в переменной department у нас лежит Personnel Department Personnel, на выходе получится HR Department HR

Если мы хотим ограничить использование HR, например, заменив только первый Personnel, мы можем добавить аргумент, указывающий сколько раз нам надо заменить подстроку, например:

`{{ department|replace("Personnel", "HR", 1) }}` - в переменной department у нас лежит Personnel Department Personnel, на выходе получится HR Department Personnel

|
Символ служит не только для применения фильтров но и для цепочки фильтров, например:

`Hello {{name | default("family name") | replace("family","second")}}!` - у нас нет name, мы подставляем family name и потом заменяем family на second, получая Hello second name!

Больше информации можно найти по следующим ссылкам

https://jinja.palletsprojects.com/en/3.0.x/ - официальная документация, пропускаем питоновский булшит и прям оно

https://docs.ansible.com/ansible/2.3/playbooks_filters.html - фильтры jinja в Ansible

https://cryptic-cliffs-32040.herokuapp.com/ - парсер для тестирования ваших переменных и шаблонов

https://regex101.com/ - парсер регулярных выражений

4.7.2

Ubuntu + CentOS

Находясь в папке ansible на локальном вашем компьютере, копируем материал данной темы.

```
cp -R C:\Users\User\ansible_course\3.other-systems-and-stacks\6.UbuntuAndCentos .
```

Заходим на контролноду и перекопируем материал в папку internal_ansible

```
vagrant ssh controlnode

cp -R ansible/6.UbuntuAndCentos internal_ansible/

cd internal_ansible/6.UbuntuAndCentos
```

Устанавливаем модуль community.mysql

```
ansible-galaxy collection install community.mysql
```

```
Process install dependency map
Starting collection install process
Installing 'community.mysql:2.1.1' to
'/home/vagrant/.ansible/collections/ansible_collections/community/mysql'
```

Запускаем плейбук на установку LEMP на два сервера CentOS/Ubuntu.

```
ansible-playbook playbook.yml

PLAY RECAP
*********************************************************************************
*****************
centos_server              : ok=24   changed=19   unreachable=0    failed=0
skipped=4    rescued=0    ignored=0
ubuntu_server              : ok=21   changed=14   unreachable=0    failed=0
skipped=4    rescued=0    ignored=0
```

Теперь можем проверить доступность наших сайтов. Заходим по IP-адресу Ubuntu http://192.168.51.4 и Centos http://192.168.51.3. Проверяем доступность php страниц. Сначала в локальный ваш hosts файл вводите `192.168.51.3 php-test.com` и заходим http://php-test.com / http://php-test.com/test-connection.php

4.8

Python

Находясь в папке ansible на локальном вашем компьютере, копируем материал данной темы.

```
cp -R C:\Users\User\ansible_course\3.other-systems-and-stacks\7.Python .
```

Заходим на контролноду и перекопируем материал в папку internal_ansible.

```
vagrant ssh controlnode

cp -R ansible/7.Python internal_ansible/

cd internal_ansible/7.Python
```

Запускаем плейбук.

```
ansible-playbook playbook.yml

PLAY RECAP
*********************************************************************************
*********************
ubuntu_no_python_server    : ok=14   changed=9    unreachable=0    failed=0
skipped=1    rescued=0    ignored=0
```

Проверяем доступность сервиса. http://192.168.51.2

4.11

Каким будет результат выполнения следующего Jinja шаблона?

*Напишите результат этого шаблона Jinja в виде

```
Storage:
   dbPath:
...
storage:
  dbPath: {{ mongodb_storage_dbpath | default("./") }}
  directoryPerDB: {{ mongodb_storage_dirperdb | to_nice_json }}
  engine: {{ mongodb_storage_engine }}
  {% if mongodb_storage_engine == 'mmapv1' -%}
  mmapv1:
    quota:
      enforced: {{ mongodb_storage_quota_enforced | to_nice_json }}
      maxFilesPerDB: {{ mongodb_storage_quota_maxfiles }}
    smallFiles: {{ mongodb_storage_smallfiles | to_nice_json }}
  {% endif -%}
  {% if mongodb_storage_engine == 'wiredTiger' -%}
  wiredTiger:
    engineConfig:
      {% if mongodb_wiredtiger_cache_size is defined -%}
      cacheSizeGB: {{ mongodb_wiredtiger_cache_size }}
      {% endif -%}
      directoryForIndexes: {{ mongodb_wiredtiger_directory_for_indexes | to_nice_json
}}
  {%- endif %}
```

Для переменных

```
mongodb_storage_dirperdb: true
mongodb_storage_engine: "wiredTiger"
mongodb_wiredtiger_cache_size: "512G"
mongodb_wiredtiger_directory_for_indexes: "/usr/share/mongo/indexes"
```


Напишите Jinja шаблон генерирующий следующий код

```
#---- Server 01 ------#
Space_name = ranger
Software_list = ["ntp", "nginx", "http"]

#---- Server 02 ------#
Space_name = ranger
Software_list = ["postgres", "mysql"]

#---- Server 03 ------#
Space_name = hunter
Software_list = ["webapp", "loader"]
Calls = {"name": "arg1", "values": ["run_main", "run_schedule"]}
```

Для переменных

```
servers:
- sname: ranger
  software:
  - ntp
  - nginx
  - http
- sname: ranger
  software:
  - postgres
  - mysql
- sname: hunter
  software:
  - webapp
  - loader
  calls:
   name: arg1
   values:
   - run_main
   - run_schedule
```

Обратите внимание: эта задача потребует объявления переменной для обсчета цифр. В версиях до 2.9* достаточно написать **{% set variable = 0 %}** и использовать в цикле. Однако с новых версий придется объявлять namespace, следующего вида **variable = namespace(value=0)** и обращаться по ключу в духе **variable.value**, иначе инкременты внутри циклов не заработают из-за изменений в областях видимости с новой версией Ansible.

* В данной задаче мы используем версию 2.9

Сопоставьте имена тасков с выполнением команд в следующем плейбуке:

```
tasks:
 - name: "Task 1"
   ansible.builtin.debug:
     msg: "System is {{ansible_os_family}}"

 - name: "Task 2"
   ansible.builtin.template:
     src: "conf.j2"
     dest: "config/conf.conf"

- name: "Task 3"
  ansible.builtin.copy:
    src: "/srv/myfiles/task.conf"
    dest: "/etc/task.conf"
    owner: ansible
    group: ansible
    mode: '0644'
    checksum: "6e642bb8dd5c2e027bf21dd923337cbb4214f827"

- name: "Task 4"
  ansible.builtin.service:
    name: nginx
    state: "reloaded"
```

Каким будет результат выполнения следующего Jinja шаблона?

*Напишите результат этого шаблона Jinja в виде

Имя = значение

Шаблон:

```
{% for key in agent.names %}
{{key}} = {{agent.vals[loop.index0]}}
{% endfor %}
policy = {{network_policy | upper}}
procedures = {{ agent.names | zip(agent.vals) | list | to_json }}
```

Для переменных

```
network_policy: strict
agent:
   names:
   - host
   - port
   - group
   - gate
   vals:
   - agent.web01.internal
   - 8081
   - net
   - disabled
```

4.12

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
node-2: python2.7

Какой IP-адрес у них?

1 и 4-ый октет сети у всех одинаковый -**172.XX.XXX.6** (node-1)
и **172.XX.XXX.7** (node-2). 2 и 3-ий 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 - **4.Другие операционные системы**, **стеки и Python**, 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.

```
sudo -i
```

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

Чиним playbook

Время приступить к реальной работе автоматизатора — искать ошибки в плейбуке. Перед вами плейбук, загружающий питоновское приложение и запускающий его вместе с Mysql. В нем ошибки, которые надо найти и исправить. К сожалению, это часто встречающаяся задача, поэтому важно уделить ей внимание.

За основу берите материал, который ранее вы рассматривали уже в теме "Python". https://gitlab.slurm.io/edu/ansible_course/-/tree/master/3.other-systems-and-stacks/7.Python

Найдите ошибки в playbook и запустите его. Подумайте, что самое трудное в поиске ошибок и почему? Что может быть причиной этих ошибок?

Все файлы расположены на node-1, в папке /etc/ansible. Работаем под рутом `sudo -i`

Наш вариант верного плейбука, для сравнения.

```
---
```

```
- name: "Pip venv"
 hosts: "centos_python"
 become: false
 vars:
   app_name: "hello"
   mysql_password: "password"
   ansible_pythonr_interpreter: /usr/bin/python3
 tasks:
 - name: "Copy python files"
   ansible.builtin.copy:
     src: files/app
     dest: /apps/{{app_name}}
     owner: vagrant
     group: vagrant
     mode: 0644

- name: Install virtualenv via pip
  pip:
    name: virtualenv
    executable: pip3

- name: Install requirements
  pip:
    requirements: /apps/{{app_name}}/app/requirements.txt
    virtualenv: /apps/{{app_name}}/venv
    virtualenv_command: /usr/bin/python3 -m venv

- name: Copy service file
  ansible.builtin.template:
    src: service.service.j2
    dest: /etc/systemd/system/{{app_name}}.service
    owner: vagrant
    group: vagrant
    mode: 0644

- name: "Add the MySQL database"
  community.mysql.mysql_db:
    name: service_db
    state: present
    login_user: root
    login_password: "password"
    check_implicit_admin: true

- name: "Start service"
  service:
    name: "{{ app_name }}"
    state: started
```

Таски можно добавлять/исправлять, удалять нельзя, все что внутри этого плея, должно быть исполнено на стенде

Пишем playbook

После поиска ошибок в плейбуке, вы готовы написать свой. Это поможет попрактиковаться в создании плейбуков и глубже понять процессы Ansible. И подарит незабываемые минуты отладки.

Напишите плейбук, устанавливающий python3 и ansible на тестовый стенд. Сгенерируйте файл hosts для этого ansible из текстового файла вида

127.0.0.1 = production01

10.65.190.11 = production02 (адреса стендов)

67.99.120.110 = production03

Положите playbook.yml к установленному ansible и запустите playbook удаленного ansible из вашего локального. Ansible запускает ansible.

```
---
- name: "Test run"
  hosts: "localhost"
  gather_facts: true

  tasks:
  - name: "ping host"
    ansible.builtin.ping:

  - name: "Print OS"
    ansible.builtin.debug:
      msg: "Host OS is {{ansible_os_family}} {{ansible_distribution_major_version}}"
...
```