Практическое занятие 4: Ansible

Практически все действия в этом упражнении будем выполнять на Ansible Control Machine (ACM).

0. Установка на Ubuntu

! В классе уже выполнено.

Устанавливаем по инструкции
```
sudo apt update
sudo apt install software-properties-common -y
sudo add-apt-repository --yes --update ppa:ansible/ansible
sudo apt install ansible -y
```

Проверяем как установилось, обратите внимание на вспомогательную информацию: где находится конфиг, где бинарник, какая версия питона используется:
```
ansible --version
```

На всех машинах убедитесь, что разрешен вход по паролю:
```
sudo nano /etc/ssh/sshd_config
```
* Найти (или добавить) строчку `PasswordAuthentication yes`

Немного о документации.

Помощь из командной строки:
```
ansible-doc -l
```

Список всех модулей очень большой, можно ввести для примера /yum - и посмотреть справку по отдельному модулю детально, с опциями и параметрами.
Чтобы перейти в конец, используйте *SHIFT+g*
```
ansible-doc yum
```

Ну и в браузере: [Module Index — Ansible Documentation]

Когда копируете что-то в YML, не забудьте проверить форматирование (отступы)!

1. Inventory и модуль Ping

Каждое упражнение будет в своей папке.

```
mkdir –p ansible_project/exercise1
cd ansible_project
cd exercise1
```

С помощью Ansible можно управлять несколькими хостами, информация о них должна быть в инвентори-файле. Если файл не указан, будет использоваться файл по-умолчанию: /etc/ansible/hosts

```
nano inventory
-------
appserver ansible_host=appserver ansible_user=vagrant ansible_password=vagrant
web01 ansible_host=web01-5       ansible_user=vagrant ansible_password=vagrant
=======
```

Опционально: чтоб не светить паролями подготовим private key-файл

```
cat nano project-key.pem
chmod 400 project-key.pem
```

Файл может иметь любое имя, оно будет указано в параметре для playbook

для ssh использовать *ansible_ssh_private_key_file=project-key.pem*

Тестируем adhoc-командами (т.е. без playbook)
```
ansible -i inventory -m ping web01
```

* Результат: в первый раз мы получим ошибку, т.к. будет выполнен интерактивный запрос на подтверждение подключения к новому хосту. От этой проверки нужно избавиться.

Изменим конфиг
```
sudo nano /etc/ansible/ansible.cfg
```

В конфиге изменить или добавить следующий параметр:
```
  host_key_checking = False
```

Проверка связи:
```
ansible -i inventory -m ping web01
ansible -i inventory -m ping web02
ansible -i inventory -m ping db01
```

Запускать одну и ту же команду можно для группы серверов, а также для групп, состоящих из групп.

Добавим в inventory следующие строки:
```
nano inventory
-------
...
[websrvgrp]
web01
web02

[dbsrvgrp]
db01

[datacenter:children]
websrvgrp
dbsrvgrp
=======
```

Проверка связи сразу для всех серверов из группы:
```
ansible -i inventory -m ping websrvgrp
ansible -i inventory -m ping datacenter
```

И то же для всех серверов сразу (можно использовать `all` или `'*'`):
```
ansible -i inventory -m ping all
```

А так можно задать параметр для всей группы:
```
nano inventory
-------
...
[datacenter:vars]
ansible_user=<user>
ansible_ssh_private_key_file=project-key.pem
=======
```

2. AdHoc-команды

Возвращаемся в /home/user/ansible_project
```
cd ..
```

Копируем первое упражнение для второго
```
cp -r exercise1/ exercise2
cd exercise2
```
Проверочка
```
ansible -i inventory -m ping all
```

Используем модуль apt, чтобы поставить Апач на web01
```
ansible -i inventory -m apt -a "name=apache2 state=present" web01
```
* Результат: ошибка: ***you need to be root***

Для повышения прав до суперпользователя существует параметр *--become*
```
ansible -i inventory -m apt -a "name=apache2 state=present" web01 --
become
```
* Результат: ***changed***

При выполнении еще раз результат будет ***changed:false***
```
ansible -i inventory -m apt -a "name=apache2 state=present" web01 --
become
```

Конфигурационные команды идемпотентны по своей природе, т.е. если они уже в нужном состоянии, ансибл поймет, что команду не надо выполнять.

Еще один пример идемпотентной команды:
```
ansible -i inventory -m service -a "name=apache2 state=started enabled=yes" web01 --become
```
* ***Changed, started***
```
ansible -i inventory -m service -a "name=apache2 state=started enabled=yes" web01 --become
```
* ***Changed: false***

Но есть и другие модули, когда изменения будут вноситься даже при повторном выполнении – когда ансибл не может угадать статус.

Создайте новый файл
```
nano index.html
-------
Some text
=======
```

Используем модуль copy для копирования файла на веб-сервера
```
ansible -i inventory -m copy -a "src=index.html dest=/var/www/html/index.html" web01 --
become
```
* Результат: файл скопирован

Но это не идемпотентная команда и при повторном выполнении будет повторное копирование, если, конечно, в целевой папке не окажется файл с тем же именем и с той же контрольной суммой.

Повторное копирование того же файла
```
ansible -i inventory -m copy -a "src=index.html dest=/var/www/html/index.html" web01 --
become
```
* Результат: файл не скопирован, контрольная сумма совпала с существующим файлом.

Измените файл
```
nano index.html
-------
Another text
=======
```

Повторное копирование измененного файла

```
ansible -i inventory -m copy -a \
  "src=index.html dest=/var/www/html/index.html" web01 --become
```
* Результат: файл скопирован

Теперь можно зайти в браузер и посмотреть на веб-страничку с текстом.

3. Playbook

Начало упражнения
```
cd /home/user/ansible_project
cp -r exercise2/ exercise3
cd exercise3
ansible -i inventory -m ping all
```

Создаем файл playbook
```
nano web_db.yml (имя выбирайте по вкусу)
```

```
-------
---
- name: Setup WebServer
  hosts: websrvgrp
  become: yes
  tasks:
    - name: Install Apache apache2
      apt:
        name: apache2
        state: present
    - name: Start & Enable apache2
      service:
        name: apache2
        state: started
        enabled: yes

- name: Setup DBServer
  hosts: dbsrvgrp
  become: yes
  tasks:
    - name: Install MySQL service
      yum:
        name: mariadb-server
        state: present
    - name: Start & Enable MySQL
      service:
        name: mariadb
        state: started
        enabled: yes
=======
```

Проверим синтаксис файла:
```
ansible-playbook -i inventory web_db.yml --syntax-check
```

Удалим ранее установленный httpd
```
ansible -i inventory -m apt -a "name= apache2 state=absent" --become web01
```

Теперь запустим плейбук
```
ansible-playbook -i inventory web_db.yml
```

Добавим задачу с копированием файла. Добавьте в первую секцию tasks
```
nano web_db.yml
-------
    - copy:
        src: index.html
        dest: /var/www/html/index.html
========
```

Теперь запустим плейбук
```
ansible-playbook -i inventory web_db.yml
```

Без названия плохо, внесем изменение в первой секцию tasks
```
nano web_db.yml
--------
    - Deploy page file
      copy:
        src: index.html
        dest: /var/www/html/index.html
========
```

Теперь запустим плейбук
```
ansible-playbook -i inventory web_db.yml
```

В случае ошибки сообщение об ошибке старается угадать где она произошла, но на самом деле ошибка может быть не там, где показывает транслятор.

Посмотрите на шаблоны использования модуля ***copy*** в документации. Скопируем один из них, чтобы добавить бекап
```
nano web_db.yml
-------
    - name: Deploy page file
      copy:
        src: index.html
        dest: /var/www/html/index.html
        owner: root
        group: root
        mode: '0644'
        backup: yes
========
```

Для тестирования нового модуля можно использовать вначале ключ -C, чтобы не применять изменения
```
ansible-playbook -i inventory web_db.yml -C
```

Финальный вариант с дополнительной группировкой в виде блоков:
```
-------
---
- name: Setup webserver
  hosts: websrvgrp
  become: yes
  tasks:
    - name: Provision services
      block:
        - name: Install Apache apache2
          apt:
            name: apache2
            state: present
        - name: Start & Enable apache2
          service:
            name: apache2
            state: started
            enabled: yes
    - name: Copy user data
      block:
        - name: Deploy page file
          copy:
            src: index.html
            dest: /var/www/html/index.html
            owner: root
            group: root
            mode: '0644'
            backup: yes

- name:
  hosts: dbsrvgrp
  become: yes
  tasks:
```

```
    - name: Install MySQL service
      yum:
        name: mariadb-server
        state: present
    - name: Start & Enable MySQL
      service:
        name: mariadb
        state: started
        enabled: yes
=======
```

Для проверки можно зайти на web02 по ssh и посмотреть изменения:
`ls /var/www/html`
* Результат: появился бекап-файл

4. Модули

Мы попробуем определить нужный модуль, использовать его, получить ошибку и решить ее.

```
cd /home/user/ansible_project
cp -r exercise3/ exercise4
cd exercise4
ansible -i inventory -m ping all
```

Для работы приложения vprofile (то, что в домашней работе) надо создать в MySQL пользователя и базу данных, а также выдать привилегии. Этим и займемся.

Из документации модуля mysql_db (работа с БД) копируем пример:
`nano db.yml`
```
-------
- name: Setup DBServer
  hosts: dbsrvgrp
  become: yes
  tasks:
    - name: Install MySQL service
      yum:
        name: mariadb-server
        state: present
    - name: Start & Enable MySQL
      service:
        name: mariadb
        state: started
        enabled: yes
    - name: Create a new db 'accounts'
      mysql_db:
        name: accounts
        state: present
=======
```

Запуск:
`ansible-playbook -i inventory db.yml`
* Результат: ошибка "Нужен PyMySQL".

В описании *Requirements* к модулю это было указано и требуется поставить этот модуль Питона. Далее необходимо понять куда это требуется поставить, т.к. при развертывании на виртуалках эта ошибка придет с “удаленной” машины, а для облака, например, это надо будет установить локально.

Иногда сложность бывает в определении необходимых зависимостей. Тогда можно залогиниться на удаленную машину и найти эти зависимости через менеджер пакетов.

Теперь давайте удаленно поставим это все на db01. В случае удаленной машины можно использовать команду:

```
ssh -i project-key.pem centos@<ip-addr-db01>
```

Но у нас Vagrant, поэтому достаточно:
```
vagrant ssh db01
```

```
sudo -i
```

Поиск модуля чувствителен к регистру, поэтому используем опцию ***-i***
```
yum search python | grep -i mysql
```
* Результат: должны найти MySQL-python.x86_64

```
nano db.yml
-------
- name: Setup DBServer
  hosts: dbsrvgrp
  become: yes
  tasks:
    - name: Install MySQL service
      yum:
        name: mariadb-server
        state: present
    - name: Install Python MySQL
      yum:
        name: MySQL-python
        state: present
    - name: Start & Enable MySQL
      service:
        name: mariadb
        state: started
        enabled: yes
    - name: Create a new db 'accounts'
      mysql_db:
        name: accounts
        state: present
=======
```

```
ansible-playbook -i inventory db.yml # OK
```

Следующий шаг, создаем пользователя
Находим модуль mysql_user

```
nano db.yml
-------
- name: Setup DBServer
  hosts: dbsrvgrp
  become: yes
  tasks:
- name: Setup DBServer
  hosts: dbsrvgrp
  become: yes
  tasks:
    - name: Install MySQL service
      yum:
        name: mariadb-server
        state: present
    - name: Install Python MySQL
      yum:
        name: MySQL-python
        state: present
    - name: Start & Enable MySQL
      service:
        name: mariadb
        state: started
        enabled: yes
    - name: Create a new db 'accounts'
      mysql_db:
        name: accounts
        state: present
    - name: Create db user 'admin'
```

```
      mysql_user:
        name: admin
        password: 12345
        priv: '*.*:ALL'
        state: present
=======
```

Запускаем, игнорируем предупреждение.
```
ansible-playbook -i inventory db.yml
```

5. Настройка конфигурации

Зачем менять конфигурацию? Ну, например, сменить порт SSH по-умолчанию 22 на что-нибудь другое. Также, если ансибл делает не то, что вы хотите – значит вопрос в конфигурации.

Конфигурация задается иерархией файлов, начиная с глобального: `/etc/ansible/ansible.cfg`
Файл конфигурации разделен на секции.
Подробности здесь: https://docs.ansible.com/ansible/2.3/intro_configuration.html

```
cd /home/user/ansible_project
cp -r exercise4/ exercise5
cd exercise5
ansible -i inventory -m ping all
```

Создаем более приоритетный файл конфигурации, проектный:
```
nano ansible.cfg
-------
[defaults]
host_key_checking = False
inventory = ./inventory
forks = 5
log_path = ./ansible.log

[privilege_escalation]
become=True
become_method=sudo
become_ask_pass=False
=======
```

Запуск
```
ansible-playbook db.yml
```

Посмотреть лог
```
cat /var/log/ansible.log
```

Обратите внимание на создаваемый локально лог. Если хотите, можно использовать общий лог, например, ***/var/log/ansible.log***
Проблема с общим логом - то, что он создается пользователем ***ubuntu*** (или ***vagrant***), а путь этот доступен только руту. То есть, если очень надо - создайте его руками и дайте привилегии:
```
      sudo touch /var/log/ansible.log
      sudo chown ubuntu:ubuntu /var/log/ansible.log
```

Другой уровень журналирования можно получить, если использовать опцию ***verbose***: -v, -vv, -vvv или -vvvv .
```
ansible-playbook db.yml -vvv
```

6. Переменные

С переменными в Ansible все как в привычных скриптовых языках.

```
cd /home/user/ansible_project
cp -r exercise5/ exercise6
cd exercise6
```
nano
```
nano db.yml
```
```
-------
- name: Setup DBServer
  hosts: dbsrvgrp
  become: yes

  vars:
    dbname: groups
    dbuser: devops
    dbpass: lesson234

  tasks:
    - name: Install MySQL service
      yum:
        name: mariadb-server
        state: present
    - name: Install Python MySQL
      yum:
        name: MySQL-python
        state: present
    - name: Start & Enable MySQL
      service:
        name: mariadb
        state: started
        enabled: yes
    - name: Create a new db 'accounts'
      mysql_db:
        name: "{{dbname}}"
        state: present
    - name: Create db user 'admin'
      mysql_user:
        name: "{{dbuser}}"
        password: "{{dbpass}}"
        priv: '*.*:ALL'
        state: present
=======
```

Устанавливаем изменения
```
ansible-playbook -i inventory db.yml
```

Но как их увидеть?

Чтоб вывести в *stdout* текушие значения переменных, используем коллекцию ***debug***:
```
-------
  tasks:
        - debug:
            var: dbname

        - debug:
            msg: "Value of dbuser is {{dbuser}}"
=======
```

Запуск
```
ansible-playbook -i inventory db.yml
```

7. Inventory-based переменные (для хостов и групп)

```
cd /home/user/ansible_project
cp -r exercise6/ exercise7
cd exercise7
```

Закомментируем все переменные в db.yml:

```
-------
#  vars:
#    dbname: groups
#       dbuser: devops
#       dbpass: lesson234
=======
```

При запуске
```
ansible-playbook -i inventory db.yml
```
* получается ошибка «*VARS not defined*»

Создаем в проекте папку под список переменных:
```
mkdir group_vars
cd group_vars
```

Переменные, описанные в файле ***all*** распространяют эффект на все хосты из Inventory
```
nano all
-------
dbuser: testadmin
dbpass: admin123
dbname: redhat
=======
```

Запуск (должно пройти без ошибок)
```
cd ..
ansible-playbook -i inventory db.yml
```

8. Про иерархию наследования переменных

```
cd /home/user/ansible_project
cp -r exercise7/ exercise8
cd exercise8
rm -rf db.yml index.html group_vars/all
```

Сделаем новый плейбук, который создаст для нас юзера на всех хостах
```
nano vars_precedence.yml
-------
- name: Understand predecence of vars
  hosts: all
  become: yes
  vars:
    USRNM: playuser
    COMM: value from playbook
  tasks:
    - name: create a user
          user:
            name: "{{USRNM}}"
               comment: "{{COMM}}"
          register: USROUT   # Используем для вывода
        - debug:
            var: USROUT
=======
```

Запуск
```
ansible-playbook vars_precedence.yml
```

Если переменная – это JSON, можно выводить отдельные поля:
```
-------
       - debug:
           var: USROUT.name
       - debug:
           var: USROUT.comment
=======
```

Запуск
```
ansible-playbook vars_precedence.yml
```
* Результат: выведены обе переменные для каждого хоста

Теперь объявим те же переменные на другом уровне
```
nano group_vars/all
-------
USRNM: globaluser
COMM: Value from group_vars/all
=======
```

Теперь переменные получают значение в разных местах
```
ansible-playbook vars_precedence.yml
```
* Результат: приоритет получили значения из плейбука

Теперь закомментим их в плейбуке
```
ansible-playbook vars_precedence.yml
```
* Результат: остались значения из ***all***

Отдельно для websrvgrp
```
nano group_vars/websrvgrp
-------
USRNM: websrvgrpuser
COMM: Value from group_vars/websrvgrp
=======
```

Проверяем
```
ansible-playbook vars_precedence.yml
```
* Результат: значения из группы более приоритетны

```
mkdir host_vars
nano host_vars/web02
-------
USRNM: web02user
COMM: Value from host_vars/web02
=======
```

Проверяем
```
ansible-playbook vars_precedence.yml
```
* Результат: значения на уровне хоста еще главней

Итого: playbook => host_vars => group_vars => all
https://docs.ansible.com/ansible/latest/user_guide/playbooks_variables.html#variable-precedence-where-should-i-put-a-variable

А еще главней - командная строка!
* Раскомментируем объявление переменных в плейбуке
# А также подставляем в командную строку
```
ansible-playbook -e USRNM=cliuser -e COMM=cli vars_precedence.yml
```
* Результат: победа за командной строкой.

9. Переменные-факты: модуль *setup*

Модуль setup запускается по-умолчанию каждый раз в любом плейбуке и нужен, чтобы собирать «факты» об окружении. Это переменные, значения которых используются другими модулями. Но не всеми.

```
cd /home/user/ansible_project
```

```
cp -r exercise8/ exercise9
cd exercise9
```

Модуль setup запускается по-умолчанию каждый раз в любом плейбуке, но можно запустить его и отдельно:

```
ansible -m setup web01
```

Можно и не запускать *setup*, если вы не используете *fact vars*. Для этого в плейбуке перед *tasks* нужно будет добавить:

```
gather_facts: False
```

Проверяем

```
ansible-playbook vars_precedence.yml
```

* Результат: отличается тем, что факты не собирались.

Вывод этих переменных на экран

```
nano print_facts.yml
-------
- name: Learning fact vars
  hosts: all
  tasks:
    - name: Print OS names
          debug:
            var:ansible_distribution
=======
```

Проверяем вывод

```
ansible-playbook print_facts.yml
```

Давайте поднимем web03, теперь на Убунту

* Потребуется добавить *private ip* в Inventory и в websrvgrp

Проверяем

```
ansible-playbook print_fact.yml
```

* Результат: ошибка «*web03 unreachable*»

Попробуем найти ошибку, получив подробный лог

```
ansible-playbook print_fact.yml -vvv
```

* В выводе видно, что используется user ***centos***, надо поменять на адекватного ситуации ***ubuntu***

В Inventory добавить (подставьте ip):

```
web03 ansible_host=<ip> ansible_user=ubuntu
```

Проверяем

```
ansible-playbook print_fact.yml
```

* Результат: Работает!

Добавим вывод свободной памяти (факт *ansible_memory_mb.real.free*)

```
nano print_fact.yml
--------
- name: Learning fact vars
  hosts: all
  tasks:
    - name: Print OS names
          debug:
            var: ansible_distribution
    - name: Print Memory info
          debug:
            var: ansible_memory_mb
    - name: Print Free Memory Details
          debug:
            var: ansible_memory_mb.real.free
=======
```

```
ansible-playbook print_fact.yml
```

Можно вывести и список. Например, *ansible_processor*

```
nano print_fact.yml
-------
- name: Learning fact vars
  hosts: all
  tasks:
    - name: Print OS names
          debug:
            var: ansible_distribution
    - name: Print Free Memory Details
          debug:
            var: ansible_memory_mb.real.free
    - name: Print Processor info
          debug:
            var: ansible_processor
=======
```

```
ansible-playbook print_fact.yml
```

Чтобы вывести только название (третий элемент списка) используются квадратные скобки:

```
    - name: Print Processor details
        debug:
          var: ansible_processor[2]
```

10. Использование фактов

В этом упражнении мы используем переменные-факты для ветвления на основе конкетного дистрибутива Linux. В документации можно посмотреть на работу с более сложными предикатами. *https://docs.ansible.com/ansible/latest/user_guide/playbooks_conditionals.html*

```
cd /home/user/ansible_project
cp -r exercise9/ exercise10
cd exercise10
ls
rm -rf group_vars hostvars print_fact.yml vars_precedence.yml
```
Должны остаться *ansible.cfg, inventory* и *project-key.pem*

В плейбуке будем включать сервис NTP (синхронизация времени).

```
nano provisioning.yml
-------
- name: Provisioning server
  hosts: all
  become: yes
  tasks:
    - name: Install NTP
      yum:
        name: ntp
        state: present
      when: ansible_distribution == "CentOS"

       - name: Install NTP on Ubuntu
      apt:
        name: ntp
        state: present
      when: ansible_distribution == "Ubuntu"
=======
```

```
ansible-playbook provisioning.yml
```

Запустим и добавим в автозагрузку сервис
Нам понадобится факт *ansible_os_family.* В зависимости от этого пропишем разные имена сервисов в соответствующих дистрибутивах RedHat/Debian.

```
nano provisioning.yml
-------
...
  tasks:
    ...
    - name: Start and enable NTP on Centos(RedHat)
      service:
        name: ntpd
        state: started
        enabled: yes
      when: ansible_os_family == "RedHat"

    - name: Start and enable NTP on Ubuntu(Debian)
      service:
        name: ntp
        state: started
        enabled: yes
      when: ansible_os_family == "Debian"
=======
```

```
ansible-playbook provisioning.yml
```

Если количество сервисов для установки больше одного, можно использовать циклы (проход по списку значений), объявляемых через связь loop-item.
https://docs.ansible.com/ansible/latest/user_guide/playbooks_loops.html

В документации по циклам стоит обратить внимание:
* на with_items
* на встроенные модули: ansible.builtin.user
* на списки словарей и форму item.subname

```
nano provisioning.yml
-------
- name: Provisioning server
  hosts: all
  become: yes
  tasks:
    - name: Install services
      yum:
        name: "{{item}}"
        state: present
      when: ansible_distribution == "CentOS"
      loop: # Список значений
        - ntp
        - wget
        - git
        - zip
        - unzip

    - name: Install services on Ubuntu
      apt:
        name: "{{item}}"
        state: present
        update_cache: yes # apt update сначала
      when: ansible_distribution == "Ubuntu"
      loop: # Список значений
        - ntp
        - wget
        - git
        - zip
        - unzip
=======
```

```
ansible-playbook provisioning.yml
```

Добавим пользователя и группу

```
nano provisioning.yml
-------
...
  tasks:
    ...
       - name: Add group
         group:
           name: devops
           state: present

       - name: Add users
         user: # Можно указать шелл, домашнюю директорию и т.п.
           name: "{{item}}"
           state: present
           groups: devops
         loop:
           - user1
           - user2
           - user3
           - user4
=======
```

```
ansible-playbook provisioning.yml
```

Циклы можно сделать по спискам из переменных.

```
nano group_vars/all
-------
usernames:
 - user11
 - user12
 - user13
 - user14
 - user15
=======

nano provisioning.yml
-------
...
         loop: "{{usernames}}"
=======

ansible-playbook provisioning.yml
```

11. Управление файлами

Посмотреть на список модулей для работы с файлами:
https://docs.ansible.com/ansible/2.9/modules/list_of_files_modules.html

```
cd /home/user/ansible_project
cp -r exercise10/ exercise11
cd exercise11
```

Простое упражнение: выставить сообщение, которое будет отображаться при подключении к удаленным машинам. Для этого необходимо записать нужный текст в файл ***/etc/motd***

```
nano provisioning.yml
-------
...
    - name: Banner file /etc/motd
      copy:
        content: "This {{ansible_distribution}} is managed by Ansible.\n" # Текст в файл
        dest: /etc/motd
=======

ansible-playbook provisioning.yml
```

* Результат: Created

Вторая часть упражнения – изменение конфигурации NTP. Будем менять список пула серверов службы времени, с которыми нужно синхронизироваться нашим машинам.

Если зайти на ***web01*** (он под *Centos*) и посмотреть файл конфигурации NTP, там есть строки с именами NTP-серверов. Скопируем на локаль или в буфер обмена весь файл:

```
vagrant ssh web01
sudo -i
cat /etc/ntp.conf
logout
```

Возвращаемся обратно на виртуалку Ansible Control Machine.

Вставьте скопированное содержимое файла конфигурации в файл проекта:

```
nano ntp_redhat.conf
```

Однако замените имена серверов на те, что рекомендуют для России. Например, можно погуглить «NTP server in Russia», скопировать и подставить найденные имена серверов.

Теперь повторите те же действия для файла конфигурации, взятого с *Ubuntu* и создайте файл проекта ntp_debian.conf.

```
nano ntp_debian.conf
```

Файлы для разных дистрибутивов похожи, но не одинаковы. Поэтому вместо ***Copy*** будем использовать модуль ***Template***. Он позволяет использовать динамические данные, т.е. подстановку. Файлы шаблонов имеют расширение .j2 – это формат библиотеки Jinger 2.

```
mkdir templates
mv ntp_* templates/
cd templates
mv ntp_debian.conf ntp_debian.conf.j2
mv ntp_redhat.conf ntp_redhat.conf.j2
cd ..

nano provisioning.yml
-------
...
    - name: Deploy NTP conf file for RedHat
      template:
        src: templates/ntp_redhat.conf.j2
        dest: /etc/ntp.conf
      when: ansible_os_family == "RedHat"

    - name: Deploy NTP conf file for Debian
      template:
        src: templates/ntp_debian.conf.j2
        dest: /etc/ntp.conf
      when: ansible_os_family == "Debian"

    - name: Restart NTP on Centos(RedHat)
      service:
        name: ntpd
        state: restarted
        enabled: yes
      when: ansible_os_family == "RedHat"

    - name: Restart NTP on Ubuntu(Debian)
      service:
        name: ntp
        state: restarted
        enabled: yes
      when: ansible_os_family == "Debian"
=======
```

Подставьте в значения переменных найденные выше имена серверов.

```
nano group_vars/all
-------
...
ntp0: …
ntp1: …
ntp2: …
ntp3: …
=======

nano templates/ntp_debian.conf.j2
-------
pool {{ntp0}} iburst
pool {{ntp1}} iburst
pool {{ntp2}} iburst
pool {{ntp3}} iburst
=======
```

```
nano templates/ntp_redhat.conf.j2
-------
server {{ntp0}} iburst
server {{ntp1}} iburst
server {{ntp2}} iburst
server {{ntp3}} iburst
=======
```

Теперь у нас есть набор переменных, два шаблона и две задачи, которые это добро используют

Перед запуском нового функционала рекомендую проверить синтаксис и посмотреть как оно будет работать (ключ -C).

```
ansible-playbook provisioning.yml --syntax-check
ansible-playbook provisioning.yml -C
```

Запуск

```
ansible-playbook provisioning.yml
```

Модуль ***File*** позволяет менять свойства файла, создавать ссылки и папки.

Для примера можно создать папку и выдать права.

```
nano provisioning.yml
-------
...
    - name: Dir for dev data
      file:
        path: /opt/devdata
        state: directory
        mode: 0775
=======
```

```
ansible-playbook provisioning.yml
```

* Результат: папка создана.

12. Handlers

В упражнении с созданием папки есть одна проблема. Она в том, что каждый при повторном запуске сервисы перезапускаются. Было бы здорово добавить проверку, что бы перезапуск происходил только при изменении настроек сервиса. Для таких случаев есть обработчики, Handlers.

https://docs.ansible.com/ansible/latest/user_guide/playbooks_handlers.html

Параметр **notify** определяет какой обработчик (или несколько) использовать, если что-то изменилось. Но обработчик вызывается лишь один раз в конце. Перезапуск сервисов - это обычный пример использования.

```
nano provisioning.yml
-------
...
    - name: Deploy NTP conf file for RedHat
          ...
          notify:
            - Restart NTP on RedHat # нужно точное имя задачи из секции handlers

    - name: Deploy NTP conf file for Debian
          ...
          notify:
            - Restart NTP on Debian # нужно точное имя задачи из секции handlers

  handlers:
    - name: Restart NTP on RedHat
          ...

    - name: Restart NTP on Debian
          ...
=======
```

`ansible-playbook provisioning.yml` # По нулям, правда?

* Внесите простое изменение в шаблоны (.j2).

`ansible-playbook provisioning.yml`
* Результат: Сервисы перезапустились.

Итого, мы использовали переменные, условия, группы, контент, шаблоны с динамической подстановкой, обработчики, модуль ***file***.

13. Теги

Задачи из playbook можно выполнить отдельно или начиная с какой-то.

Посмотреть список задач
```
ansible-playbook provisioning.yml --list-tasks
```

Выполнить playbook, начиная с конкретной задачи
```
ansible-playbook provisioning.yml --start-at-task "Deploy NTP conf file
for RedHat"
```

Выполнить playbook, в интерактиве спрашивая выполнять ли каждую следующую задачу:
```
ansible-playbook provisioning.yml --step
```

Теги – это еще один способ группировать задачи или блоки внутри файла, чтобы выполнить отдельную группу задач.

```
cd /home/user/ansible_project
cp -r exercise12/ exercise13
cd exercise13
```

Для каждой задачи в provisioning.yml добавим какой-нибудь тег. Например:

```
      tags:
        - packages

      tags:
        - services

      tags:
        - system

      tags:
        - conf
```

Посмотреть список тегов
```
ansible-playbook provisioning.yml --list-tags
```

Выполнить из playbook только задачи с тегом **packages**:
```
ansible-playbook provisioning.yml --tags packages
```

Выполнить из playbook все задачи, кроме задач с тегом **packages**:
```
ansible-playbook provisioning.yml --skip-tags packages
```

14. Роли

Установить nginx можно очень просто:

```
---
- name: webserver setup
  hosts: all
  tasks:
    - name: ensure nginx is at the latest version
      apt:
        name: nginx
        state: latest
    - name: start nginx
      service:
        name: nginx
        state: started
        enabled: yes
...
```

Но возможностей по конфигурированию такого веб-сервера довольно много и учитывая разные варианты развертывания можно придумать много вариантов рецептов. В этом помогают роли. Роли в Ansible позволяют навести порядок и структурировать развертывание приложений на предприятии. Базовый репозиторий опубликованных сообществом ролей находится здесь: *https://galaxy.ansible.com*

Скопировать к себе код роли можно утилитой ***ansible-galaxy***:

```
ansible-galaxy install geerlingguy.nginx
```

Использовать роль можно так (пример без особого смысла):

```
nano web.yml
---
- name: Provisioning server
  hosts: all
  become: yes
  roles:
    - role: geerlingguy.nginx
...
```

```
ansible-playbook web.yml
```

Куда это скачивается:

```
###sudo -i
cd ~/.ansible/roles
ls
cd geerlingguy.nginx
ls
```

Здесь можно поучиться.
Задание: разобраться со структурой папок и файлов в роли и с вложенностью ролей. Здесь поможет документация: *https://docs.ansible.com/ansible/latest/user_guide/playbooks_reuse_roles.html*

Ansible Vault

0. Настройка редактора Ansible Vault

Чтобы сохранить вашу конфиденциальную информацию, такую как пароли или секретные ключи, вам понадобится защищенное хранилище, Vault.

Прежде чем начать, установите системный редактор по умолчанию на Ansible Control Machine. Заменим **vim** вашим любимым редактором, например, **nano**. Для Bash это делается так:

```
# Проверить установлен ли редактор по-умолчанию
cat ~/.bashrc | grep EDITOR

# Добавить в настройку редактор по-умолчанию
echo "export EDITOR=nano" >> ~/.bashrc
source ~/.bashrc

# Чтобы применить настройки нужно перезайти в систему
exit
vagrant ssh acm
echo $EDITOR
```

1. Использование Ansible Vault

Для управления зашифрованным содержимым в Ansible используется утилита ansible-vault.
С её помощью можно создавать, редактировать, просматривать и расшифровывать зашифрованные файлы или части playbook (об этом позже).

Продолжаем серию упражнений из предыдущего модуля.

```
cd /home/user/ansible_project
mkdir exercise14
cd exercise14

# Создать зашифрованный файл.
ansible-vault create create_users.yml
```

Вам будет предложено ввести и подтвердить безопасный пароль, после чего Ansible откроет окно редактирования файла.

Для шифрования существующих файлов используется команда **ansible-vault encrypt**.

```
echo "SecurePassword" > passwords.txt

# Зашифровать существующий файл
ansible-vault encrypt passwords.txt
```

Это заменит незашифрованный файл зашифрованным.

```
# Так содержимое файла больше просмотреть не получится
cat passwords.txt
```

2. Работа с зашифрованными файлами

Вот так можно посмотреть что в зашифрованном файле на самом деле лежит
```
ansible-vault view passwords.txt
```

Для редактирования зашифрованного файла используйте команду **ansible-vault edit**.
```
ansible-vault edit create_users.yml
```

Вас попросят ввести пароль для файла.

Вы всегда можете обновить пароль шифрования с помощью команды **ansible-vault rekey**.
```
ansible-vault rekey create_users.yml
```

Введите старый пароль и новый пароль для установки при появлении запроса. После обновления файл будет доступен с использованием нового пароля.

Просмотр Ansible зашифрованного файла командой **ansible-vault view**.
```
ansible-vault view create_users.yml
```

Если файл больше не нужно шифровать, вы можете расшифровать его командой **decrypt**.
```
ansible-vault decrypt create_users.yml
```

Теперь, после расшифровки, вы сможете увидеть фактическое содержимое файла.
```
cat create_users.yml
```

3. Шифрование чувствительных переменных

В мире автоматизации с совместной работой вам не обязательно шифровать файлы целиком, а лишь конфиденциальные данные в них, такие как пароли базы данных, ключи от интерфейсов, учетные данные пользователя и т. д.

```
cd ..
cp -r exercise13/ exercise15
cd exercise15
ls
rm -rf ansible.log group_vars templates
```

Создадим зашифрованную переменную
```
ansible-vault encrypt_string MyStrongPassw@rd
```

Скопируем вывод
и определим другие переменные и ссылку на зашифрованную в другом файле переменную.
```
nano vars/plain.yml
-------
       db_user: devopspoweruser
       db_port: 3306
       db_pass: <hash>
=======
```

Создать новый playbook.
```
nano learn_vault.yml
-------
- name: Create users
  hosts: localhost
  tasks:
```

```
    - name: Include vars
      include_vars:
        dir: vars
    - name: Show vars
      debug:
        msg: "user {{db_user}}, port {{db_port}}, pass {{db_pass}}"
=======
```

```
ansible-playbook --connection=local learn_vault.yml --ask-vault-pass
```

* Результат: в выводе отобразились все переменные

Создадим файл зашифрованных переменных vault.yml

```
nano vars/vault.yml
-------
       vault_db_pass: MyStrongPassw@rd
=======
```

```
ansible-vault encrypt vars/vault.yml
```

Теперь это все зашифровано.

```
cat vars/vault.yml
```

Уберем зашифрованную переменную из plain.yml

```
nano vars/plain.yml
-------
       db_pass: "{{ vault_db_pass }}"
=======
```

Протестировать

```
ansible-playbook --connection=local learn_vault.yml --ask-vault-pass
```

* Результат: в выводе отобразились все переменные

4. После того, как конфиденциальные данные зашифрованы, давайте использовать их в Ansible playbook

Возьмем нашу конфигурацию и inventory из прошлого семинара

```
cd ..
cp -r exercise13/ exercise16
cd exercise16
```

Создадим playbook для работы с MySQL: создать базу данных, создать пользователя, создать таблицу, заполнить таблицу данными.

Для этого потребуется модуль, которого нет в оициальной документации.

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

```
nano makedb.yml
-------
- name: Deploy database with user data
  hosts: dbsrvgrp
  tasks:
    - name: Include vars
      include_vars:
        dir: vars

    - name: Create database and user
      tags:
        ddl
```

```
      block:
        - name: Create database
          mysql_db:
            name: "{{db_name}}"
            state: present

        - name: Create user
          no_log: yes
          mysql_user:
            name: "{{db_user}}"
            password: "{{db_user_pass}}"
            priv: '*.*:ALL'
            state: present

    - name: Create db tables
      tags:
        ddl
      community.mysql.mysql_query:
        login_db: "{{db_name}}"
        login_user: "{{db_user}}"
        login_password: "{{db_user_pass}}"
        query:
          CREATE TABLE IF NOT EXISTS {{base_table_name}} ( c1 INT, c2 VARCHAR(100) )
======
```

Проверить переменные
```
nano vars/plain.yml
```

Протестировать
```
ansible-playbook makedb.yml --syntax-check
ansible-playbook makedb.yml -C
ansible-playbook makedb.yml
```

В относительно новых версиях Ansible, 2.4 и выше, вы можете использовать параметр **vault-id**
```
ansible-playbook --vault-id=@prompt  makedb.yml
```

Чтоб не вводить пароль каждый раз, его можно записать в файл паролей.
```
echo "lesson1234" > .ansible-vault-pass
ansible-playbook --vault-password-file=.ansible-vault-pass  makedb.yml
```

Создать файл паролей. Имя может быть любым, но мы в-общем случае не хотим его светить или публиковать.
```
echo 'MyStrongVaultP@ssword' > .ansible_vault_pass
```

Еще можно установить переменную среды **ANSIBLE_VAULT_PASSWORD_FILE** с путем к файлу пароля.
Так вы сможете настроить файл пароля по-умолчанию.
```
export ANSIBLE_VAULT_PASSWORD_FILE=./.ansible_vault_pass
```

Этого же можно достичь, используя локальный файл конфигурации ansible.cfg.
```
nano ansible.cfg
-------
    [defaults]
       vault_password_file = ./.ansible-vault-pass
=======

ansible-playbook makedb.yml
```

Добавим загрузку файла данных.
```
nano makedb.yml
-------
...
    - name: Deploy userdata
      tags:
        dml
```

```
      block:
        - name: Create userdata dir
          file:
            path: "{{user_data_path}}"
            state: directory
            mode: 0775

        - name: Deploy a data file
          copy:
            src:  files/initial_data.csv
            dest: "{{user_data_path}}/{{base_table_name}}.csv"

        - name: Delete all from db tables
          community.mysql.mysql_query:
            login_db: "{{db_name}}"
            login_user: "{{db_user}}"
            login_password: "{{db_user_pass}}"
            query:
              TRUNCATE TABLE {{base_table_name}}

        - name: Insert values from file
          shell:
            mysqlimport --ignore-lines=1 --fields-terminated-by='\t' \
                        --columns='c1,c2' --local -u root --password=  \
                        "{{db_name}}" "{{user_data_path}}/{{base_table_name}}.csv"
=======
```

Тут требуются переменные: **user_data_path, base_table_name, db_name, db_user, db_user_pass**

```
nano vars/plain.yml
```

И файл данных

```
mkdir files
nano files/initial_data.csv
```

```
-------
       c1     c2
       098    текст
=======
```

```
ansible-playbook makedb.yml --skip-tags=ddl
```

5. Шифрование файла данных

Файл данных тоже зачастую нужно шифровать при пересылке. Можно воспользоваться тем же паролем для шифрования:

```
ansible-vault encrypt files/initial_data.csv
```

Для интереса добавим туда строчку

```
ansible-vault edit files/initial_data.csv
```

Протестировать

```
ansible-playbook makedb.yml
```

6. vault-id, использование нескольких файлов паролей

Для прошлого упражнения сделаем файл данных зашифрованным другим паролем.

```
cd ..
cp -r exercise16/ exercise17
cd exercise17
```

Уберите пароль по-умолчанию (иначе он будет подставляться везде)

```
nano ansible.cfg
```

Создать новый файл паролей
```
echo 'MyStrongVaultP@ssword' > myvault
```

Создадим новый файл данных
```
ansible-vault create --vault-password-file=myvault files/private_data.csv
```
а можно иначе:
```
ansible-vault create --vault-id=data@myvault files/private_data.csv
```

vault-id – можно трактовать как имя секрета в хранилище.

Теперь, чтоб playbook работал, нужны два пароля.

Подставить новый файл данных
```
nano makedb.yml
```

Протестировать
```
ansible-playbook --vault-password-file=.ansible-vault-pass --vault-
password-file=myvault makedb.yml
```

Зайти на db01 и посмотреть на файл данных – там он уже расшифрован
```
cat /opt/userdata/t1.csv
```

Чтобы использовать автоподстановку нескольких файлов паролей нужно править конфиг:

```
nano ansible.cfg
-------
       [defaults]
       vault_password_file = ./.ansible-vault-pass
       vault_identity_list = data@myvault
=======
```

Тогда для выполнения будет достаточно простой команды:
```
ansible-playbook makedb.yml
```