Ansible

Введение

Системы управления конфигурацией разрабатываются для администраторов и операционных отделов с целью ускорить процедуры управления большим количеством серверов. Они позволяют автоматически контролировать много разных систем из единого центра.

Хотя для систем Linux выпущено много популярных инструментов управления конфигурациями, в том числе Chef и Puppet, эти инструменты сложнее, чем требуется большинству людей.
Система Ansible — отличная альтернатива этим инструментам, поскольку имеет простую архитектуру, не требует установки на узлы специального программного обеспечения, использует SSH для выполнения задач автоматизации и файлы
YAML для определения деталей выделения ресурсов.

Ansible

Ansible — это программное решение для удаленного управления конфигурациями. Оно позволяет настраивать удаленные машины. Это означает, что с помощью Ansible вы можете удаленно предоставлять целый парк удаленных серверов, устанавливать и развертывать на них программное обеспечение и отслеживать их удаленно.

> ***Задача Ansible – выполнить ряд операций на удаленной машине. При этом машин может быть сразу несколько.***

Одной из уникальных особенностей Ansible является то, что он не устанавливает никакого программного обеспечения на управляемые машины. Управляет машинами удаленно через SSH. Чтобы управлять удаленным компьютером, вам просто нужно убедиться, что ваш открытый ключ SSH находится в файле авторизованных ключах этого компьютера.

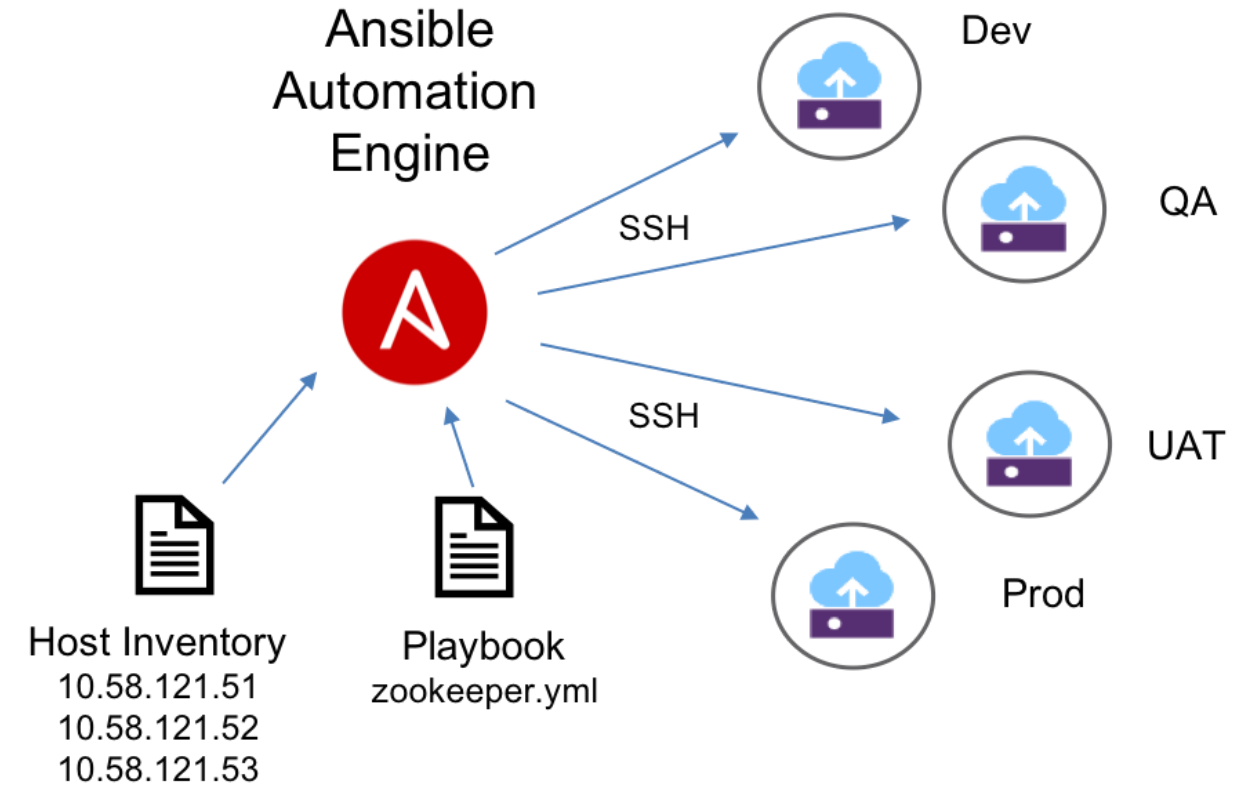
Примеры задач, которые поможет решить Ansible:
• подключение по SSH к устройствам
• параллельное подключение к устройствам по SSH (можно указывать, ко скольки устройствам подключаться одновременно)
• отправка команд на устройства
• удобный синтаксис описания устройств:
• можно разбивать устройства на группы и затем отправлять какие-то команды на всю группу
• поддержка шаблонов конфигураций с Jinja2

Ansible нужно устанавливать только на той машине, с которой будет выполняться управление устройствами.
Требования к управляющему хосту:
• поддержка Python 3 (тестировалось на 3.7)
• Windows не может быть управляющим хостом

Ansible берет на себя всю работу по приведению удаленных серверов в необходимое состояние. Администратору необходимо лишь описать, как достичь этого состояния с помощью так называемых сценариев — специальных файлов «**playbook**».

В них описывается желаемое состояние управляемой системы, например, необходимо наличие пакета, проверяет, соответствует ли удаленный компьютер описанию в плейбуке, и если это не так, приводит его в должный вид. Формат для **playbook** — **YAML.**

Основная идея **Ansible** – наличие одного или нескольких управляющих серверов, из которых вы можете отправлять команды или наборы последовательных инструкций (**playbooks**) на удаленные сервера, подключаясь к ним по **SSH**.



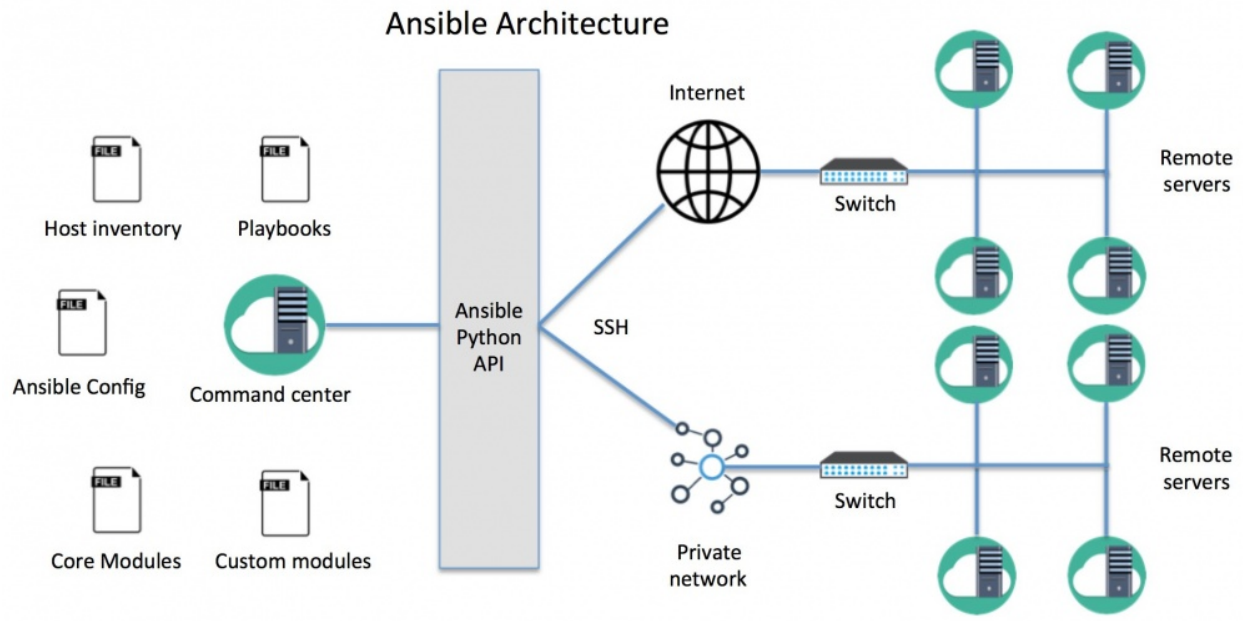


Схема работы Ansible

Файл **Host inventory** содержит информацию об обслуживаемых серверах, где команды будут исполнены. Файл конфигурации **Ansible** может быть полезен для указания настроек вашего окружения.

Наборы инструкций (**playbooks**) состоят из одной или более задач, которые описываются с помощью функциональность модуля ядра **Ansible** или сторонних модулей, которые могут потребоваться в специфических ситуациях. Сами по себе наборы инструкций — последовательные наборы команд, в которых могут быть проверки условий: если условие не выполняется, определенные команды могут пропускаться.

Также вы можете использовать **Ansible API** для запуска скриптов. Если скрипту-обертке (**wrapper**) может потребоваться запуск **playbook**, это можно сделать через **API**. Сами **playbooks** описываются декларативно в формате **YAML**. **Ansible** поддерживает сценарии развертывания новых облачных серверов и конфигурирования их на основании ролей. Часть работы может быть проведена в локальном режиме на управляющем сервере, а остальная — на созданном сервере после его первой загрузки.

Словарь терминов

В этом этой инструкции широко используются такие термины **Ansible**:

- **Control Machine** или **Node** — ведущая система, в которой установлен **Ansible** и откуда он может подключаться к **нодам** и выполнять на них команды.
- **Хост** — в **Ansible** хост — это удаленный компьютер, которому назначены отдельные переменные, и они далее группируются вместе. У каждого хоста есть выделенное имя или уникальный **IP-адрес**, чтобы сделать его идентификацию легкой и быстрой. Им также может быть присвоен простой номер порта, если вам не нужно обращаться к ним через соединение **ssh**.
- **Нода** или **Узел** — сервер, управляемый **Ansible**.
- **Файл инвентаря** — файл, который содержит информацию о серверах, которыми управляет **Ansible**, обычно находится в `/etc/ansible/hosts`.
- **Playbooks** — они написаны на языке программирования **YAML** с минимальным синтаксисом и обычно используются для автоматизации задач при необходимости.
- **Роль** — коллекция **плейбуков** и других файлов, которые имеют отношение к цели. Например, к установке web-сервера.
- **Play** — полный набор инструкций **Ansible**. В **play** может быть несколько **плейбуков** и **ролей**, включенных в один **плейбук**, который служит точкой входа.
- **Задача** — каждая инструкция, определенная в **книге игр**, называется **задачей**, которая будет выполняться в дальнейшем для выполнения действия.
- **Факты** — они выводятся из удаленных узлов автоматически при выполнении модулей на удаленных узлах.
- **Группа** — это комбинация хостов, которые назначены пулу, и переменные также могут совместно использоваться.
- **Инвентаризация** — инвентаризация является важным компонентом **ANSI** удаленного механизма, который описывает хосты, группы и так далее. С помощью **IP-адреса** или номера порта и так далее. Таким образом, вы можете определить все хосты в одном файле для быстрого доступа.

- **API** — это транспортная среда для различных облачных сервисов, как частных, так и общедоступных.
- **Модули** — с помощью **playbook** модули могут быть выполнены на удаленных узлах напрямую. Кроме того, его можно использовать для управления службами, ресурсами, пакетами, файлами или командами и так далее. Модули являются основными компонентами, которые помогают устанавливать пакеты, позволяют **API-интерфейсам** взаимодействовать друг с другом и планировать действия для системных файлов. В **Ansible** есть множество модулей, которые запрограммированы для автоматизации практически всего внутри инструмента.
- **Плагины** — это специальные части кода, которые помогают быстро писать код. Плагины автоматизируют задачи разработки и помогают максимально ускорить работу по развертыванию. **Ansible** оснащен различными удобными плагинами, которые можно использовать при необходимости, чтобы упростить вам задачу.
- **Оркестровка** — это общий термин, который часто используется в техническом мире. Почему это важно и в **Ansible**? Для разных программных продуктов значение оркестровки может быть различным. **Ansible** использует его в качестве дирижера для управления оркестром.

Подробнее про основные моменты.

Playbook

Все операции вы записываете в некий Ansible-скрипт. Действия выполняются последовательно. Логика работы похожа на обычный bash-скрипт, только на своих командах. Это позволяет отвязаться от дистрибутива, описав задачи на *промежуточном* языке. Такой Ansible-скрипт называется **playbook** (сценарий). Формат его очень простой и построен на yaml-разметке.

```
- name: "Первый шаг"
  hosts: localhost
  tasks:
  - ЗадачаA
  - ЗадачаB
```

Тут мы описали группу задач. Дали ей общее имя, *name: “*Первый шаг*”*. Указали на каких хостах выполнять, *hosts: localhost.*

Далее следует перечень задач, которые надо выполнить: *tasks: taskA, taskB*. Такая группа задач называется **play**. Это что-то типа блока задач, которые вы выделяете по какой-то своей логике. В одном playbook-е может быть несколько блоков (play-ев):

```
- name: "Первый шаг"
  hosts: localhost
  tasks:
  - ЗадачаA
  - ЗадачаB

- name: "Второй шаг"
```

```
  hosts: all
  tasks:
  - ЗадачаC
  - ЗадачаD
```

Заметьте, второй блок мы поставили для всех хостов (*hosts: all*). То есть в первом мы могли, к примеру, что-либо подготовить локально (создать виртуальные машины, подготовить файлы, залить их), а потом применить для всех машин во втором блоке.

Task

Task или задача - действие, которые необходимо выполнить.

Например, у нас есть задача “Установить Python3” на используемую для настройки ОС:

```
tasks:
 - name: Install Python 3
   apt: pkg=python3-minimal state=installed
```

Inventory

Inventory - это так называемый файл *инвентаризации*, в котором у нас хранится перечень машин, с которыми будет работать Ansible. Файл **hosts.**

Располагаться он может в вашей директории и иметь примерно такой вид:

```
[mail]
mail-01.serve.su
mail-02.myserver.lan ansible_host=10.10.10.6

[web]
web-01.myserver.lan ansible_host=10.10.10.12
```

Roles

Можно разбивать ваши задачи по разным playbook-файлам. К примеру вынести первичную настройку в common.yml, установку Apache в apache.yml, MySQL – в mysql.yml, PHP – в php.yml, потом все это включить в файл lamp.yml

```
- name: "LAMP"
  hosts: web
  tasks:
    - name: "Include common"
      include: common.yml
...
```

Роль – это как раз ваш LAMP в данном случае. Набор определенных типовых задач для конкретной машины. Ее роль.

Для того, чтобы определить роль, мы должны создать определенную файловую иерархию и разложить все по директориям. Вот пример такой иерархии для роли с именем “common”:

```
playbooks
├── roles
│   ├── common
│   │   ├── defaults
│   │   │   └── main.yml
│   │   ├── files
│   │   │   └── text.txt
│   │   └── tasks
│   │       └── main.yml
```

Смысл в том, чтобы разнести все раздельно по директориям.

- Переменные по умолчанию – в директорию **defaults**.
- Файлы, используемые в роли (к примеру которые надо загрузить на сервер) – в директорию **files**.

Таких определенных директорий много.

- Шаблоны – **templates**.
- Обработчики задач (вызываемые после успешного или неуспешного из завершения) – **handlers**.
- Прочие переменные – **vars**.
- Метаданные – **meta**.

И в каждой директории главный файл – **main.yml**. Но не обязательно создавать все эти директории сразу. Достаточно создать **roles/имя_роли/tasks/main.yml** и прописать в нем все задачи, характерные для данной роли. Далее в основном playbook-файле вы уже можете ссылаться на эти роли:

```
- name: "LAMP deployment"
  hosts: web
  become: true
  roles:
    - role: "common"
    - role: "php"
    - role: "apache"
    - role: "mysql"
```

Примерно так может выглядеть ваш playbook для установки LAMP. Теперь его можно “накатывать” сразу на кучу серверов в вашем hosts-файле!

Установка Ansible (Linux, Ubuntu 20.04)

Чтобы начать использовать Ansible для управления серверной инфраструктурой, необходимо предварительно установить программное обеспечение Ansible на компьютер, который будет выступать в качестве узла управления Ansible.

> Ansible требует наличие установленного интерпретатора Python (> 2.7).

Сначала обновите индекс пакетов системы с помощью следующей команды:

```
sudo apt-get update
sudo apt-get upgrade
```

После обновления вы можете установить программное обеспечение Ansible следующим образом из официального репозитория Ubuntu:

```
sudo apt install ansible
```

> Также есть вариант установки через пакетный менеджер python - pip: `pip install ansible`

Чтобы проверить правильность установки Ansible, введите команду для просмотра версии:

```
ansible --version
```

> Ansible на Windows возможно установить только через WSL. Прямой путь не предусмотрен.

Настройка кластера Vagrant

> Если вы хотите запустить виртуальные машины внутри другой виртуальной машины, то необходимо включить Вложенную виртуализацию: https://www.comss.ru/page.php?id=7726

> Если описанный выше способ не помог, то необходимо перенастроить гипервизор Hyper-V. Переходим Панель управления → Программы → Включение и отключение компонентов Windows. Снимаем галочки возле пунктов “Hyper-V” и “Платформа виртуальной машины”. Затем запускаем PowerShell от имени администратора и выполняем команду `bcdedit /set hypervisorlaunchtype off` и перезагрузите компьютер.

> Если ни один из вариантов не помог, то стоит воспользоваться WSL. Для гостевых ОС через WSL не требуется вложенная виртуализация.

> ***У условиях блокировок для работы Vagrant использовать прокси следующим образом:***

```
mkdir Vagrant
cd Vagrant/
vagrant init bento/ubuntu-20.04
export https_proxy='http://aeV7uu:kXayFC@193.32.155.205:8000/'
export VAGRANT_HTTP_PROXY=${http_proxy}
export VAGRANT_NO_PROXY="127.0.0.1"
vagrant up
```

Чтобы по-настоящему понять всю мощь Ansible, вам нужно несколько серверов для управления. В этом руководстве я буду использовать кластер Vagrant из 3 виртуальных машин, но для Ansible это всего лишь несколько хостов, которыми он должен управлять. Чтобы узнать больше о Vagrant, ознакомьтесь с руководством по Vagrant.

Сначала необходимо установить VirtualBox:

```
sudo apt install virtualbox virtualbox-ext-pack
```

Далее устанавливаем Vagrant:

```
curl -fsSL https://apt.releases.hashicorp.com/gpg | sudo apt-key add -

sudo apt-add-repository "deb [arch=amd64] https://apt.releases.hashicorp.com $(lsb_release -cs) main"

sudo apt-get update && sudo apt-get install vagrant
```

Чтобы убедиться, что установка прошла успешно, выполните следующую команду, которая распечатает версию Vagrant:

```
vagrant --version
```

Затем создадим папку проекта для работы с Vagrant, например, `vagrantvm`:

```
mkdir vagrantvm
cd vagrantvm
```

Создадим в этом каталоге следующей файл с именем Vagrantfile:

```
nano Vagrantfile
```

```
# -*- mode: ruby -*-

# vi: set ft=ruby :
```

```
hosts = {

  "conor" => "192.168.88.10",

  "mcgregor" => "192.168.88.11"

}

Vagrant.configure("2") do |config|

  config.vm.box = "bento/ubuntu-20.04"

 hosts.each do |name, ip|

    config.vm.define name do |machine|

      machine.vm.network :private_network, ip: ip

      machine.vm.provider "virtualbox" do |v|

        v.name = name

      end

    end

  end

end
```

Затем выполним команду `vagrant up`. Vagrant создаст 2 виртуальные машины, доступные как *conor* и *mcgregor*. Чтобы проверить, введите `vagrant status`. Вы должны увидеть:

```
Current machine states:

conor                     running (virtualbox)
mcgregor                  running (virtualbox)

This environment represents multiple VMs. The VMs are all listed
above with their current state. For more information about a specific
VM, run `vagrant status NAME`.
```

Чтобы убедиться, что вы можете использовать SSH на хостах кластера, введите:

`vagrant ssh-config > ~/.ssh/config`.

Теперь вы можете использовать SSH на любом из ваших виртуальных серверов, используя их имя хоста. Например: `vagrant ssh conor`.
Это позволит Ansible подключаться к хостам вашего кластера через SSH без каких-либо проблем с именами пользователей, паролями или ключами.

Таким образом, Vagrant нужен для удалённого и массового создания виртуальных машин, а Ansible для управления ими: настройки машин, установки и удаления на них ПО и т.д.

“Инвентаризация”

Теперь, когда у нас есть кластер, нам нужно рассказать об этом Ansible. Это делается с помощью инвентарного файла. Файл инвентаризации - это список имен хостов, организованных в группы с использованием формата файла INI. Поместите следующее в файл с именем 'hosts' в вашей рабочей директории проекта:

```
nano hosts
```

```
[first]

 conor


[second]

 mcgregor
```

Я поместил «conor» в группу «first», а остальные хосты (mcgregor) в группу «second». Эта организация позволит нам выполнять действия в отношении этих групп. Вы также можете выполнять действия на отдельных хостах и на всех хостах.

Команды Ansible

Пришло время взяться за дело. Простейший способ использования Ansible - запуск специальных команд. Специальные команды используют модули. Формат специальной команды:

```
ansible -i -m [-a , ... ]
```

Например, чтобы увидеть, все ли хосты в вашем инвентаре работают, вы можете использовать модуль ping (без аргументов):

```
ansible all -i hosts -m ping
```

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

Ansible имеет много модулей для всех распространенных задач системного администрирования, таких как управление файлами, управление пользователями и пакетами, а также множество необычных задач. Но если вы не можете найти то, что вам нужно, или просто чувствуете себя более комфортно с простыми командами оболочки, вы можете использовать модуль оболочки напрямую, включая каналы. Следующая команда извлекает внутренние и внешние IP-адреса всех хостов:

```
ansible all -i hosts -m shell -a 'ip addr | grep inet*'
```

```
conor | CHANGED | rc=0 >>
    inet 127.0.0.1/8 scope host lo
    inet6 ::1/128 scope host
    inet 10.0.2.15/24 brd 10.0.2.255 scope global dynamic eth0
    inet6 fe80::a00:27ff:feb1:285d/64 scope link
    inet 192.168.88.10/24 brd 192.168.88.255 scope global eth1
    inet6 fe80::a00:27ff:feb2:1a36/64 scope link
mcgregor | CHANGED | rc=0 >>
    inet 127.0.0.1/8 scope host lo
    inet6 ::1/128 scope host
    inet 10.0.2.15/24 brd 10.0.2.255 scope global dynamic eth0
    inet6 fe80::a00:27ff:feb1:285d/64 scope link
    inet 192.168.88.11/24 brd 192.168.88.255 scope global eth1
    inet6 fe80::a00:27ff:fe6a:dd32/64 scope link
```

Настройка Playbooks

Специальные команды хороши, когда вы хотите быстро что-то сделать на нескольких хостах, но настоящая сила Ansible в его Playbooks. Playbooks - это файлы YAML, в которых вы определяете наборы задач для достижения таких целей, как подготовка, настройка, развертывание и управление вашей инфраструктурой.

Давайте посмотрим на то, как выглядит типичный playbook, прежде чем мы перейдем к деталям.

```
---

- hosts: second

  tasks:

   - name: Install Nginx

     apt: name=nginx update_cache=true

     notify: Start Nginx
     become: yes
   - name: Install Python 3

     apt: name=python3.8
     become: yes

  handlers:
```

```
    - name: Start Nginx

      service: name=nginx state=started
```

В playbook есть раздел hosts, в котором вы указываете хосты из файла инвентаризации. В этом случае название группы "second". Затем есть раздел задач с двумя задачами, которые устанавливают Nginx и Python 3. Наконец, есть раздел обработчиков, где Nginx запускается после его установки.

> Опция become установите `yes` чтобы активировать повышение привилегий.

Вы запускаете playbook с помощью команды `ansible-playbook`.
Вам все еще нужно предоставить файл инвентаря и книгу воспроизведения, которую вы хотите запустить. Сохраните playbook в файл с именем "playbook.yml" в вашем рабочем каталоге. Давайте попробуем:

```
nano playbook.yml
```

Теперь запустим выполнение плейбука на всех хостах:

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

```
PLAY [second] ******************************************************************

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

TASK [Install Nginx] ***********************************************************
ok: [mcgregor]

TASK [Install Python 3] ********************************************************
ok: [mcgregor]

PLAY RECAP *********************************************************************
mcgregor                   : ok=3    changed=3    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

> `ansible all -a "df -h" -u root` - u root указывает на запуск от пользователя root

Ansible является идемпотентом, что означает, что если что-то уже находится в желаемом состоянии, то Ansible оставит это в покое. В выводе
`ansible-playbook`вы можете увидеть, какие задачи были успешными или неудачными, а какие были изменены.

Проверим всё ли корректно установилось. Подключимся к виртуальной машине “mcgregor” через ssh и проверим установленную версию python:

```
vagrant ssh mcgregor
python3 --version
```

```
exit
```

Либо проделаем это в интерактивном режиме без ssh:

```
ansible -i hosts -m shell -a 'python3 --version' mcgregor
```

```
mcgregor | CHANGED | rc=0 >>
Python 3.8.10
```

Таким образом всё настроено и установлено корректно.

Модули

Ansible имеет очень модульную и расширяемую архитектуру. Все его возможности организованы в модули. Есть основные модули и дополнительные модули. Каждый модуль представляет команду, и большинство принимает аргументы. Вы можете использовать модули непосредственно в специальных командах или в книгах воспроизведения. Вы можете прочитать обо всех модулях в документации.

Заключение

Таким образом, с помощью связки Ansible + Vagrant нам удолось автоматизировать поднятие и настройку кластера из двух виртуальных машин (с последующей установкой nginx и python).

Список использованных источников

1. https://code.tutsplus.com/ru/tutorials/automate-all-the-things-with-ansible-part-one--cms-25931
2. https://kuzevanov.ru/linux/ustanovka-i-nastrojka-ansible-v-ubuntu-20-04.html
3. https://habr.com/ru/post/305400/
4. https://www.8host.com/blog/kak-rabotat-s-ansible-prostaya-i-udobnaya-shpargalka/
5. https://docs.ansible.com/