Ansible

→

MINICURSO

Bem-vindos à aula!

Para facilitar o acompanhamento da apresentação, inicie realizando o clone do repositório abaixo. Dessa forma, será possível ter acesso ao material desenvolvido para a parte prática do curso.

```
$ git clone https://github.com/higorsnt/ansible-workshop.git
```

Introdução

Lançado em 2012 e adquirido em 2015 pela Red Hat, o ansible é uma ferramenta de automação de provisionamento, gerenciamento de configurações, implantação de aplicações, orquestração entre outros processos.

Importância

Baseado em uma linguagem de configuração simples, fazendo-o ideal para equipes de DevOps e SRE que desejam automatizar tarefas repetitivas e complexas.

- **Automação de tarefas**: elimina as atividades manuais repetitivas para configurar servidores, implantar novas aplicações ou monitorar desempenho dos servidores;
- **Gerenciamento de configuração**: facilita a padronização e implantação das configurações entre os servidores;
- **Provisionamento de infraestrutura**: pode ser usado para implantar novos servidores e aplicativos de forma rápida e eficiente;
- **Simplicidade e facilidade de uso**: é relativamente simples de aprender e usar;
- **Gerenciamento de configuração sem agente**: não requer instalação de software em seus servidores;
- **Inventário dinâmico**: consegue descobrir dinamicamente servidores e seus recursos;
- **Módulos poderosos**: possui ampla biblioteca pré-construídos que automatizam tarefas comuns. Além disso, vários outros módulos desenvolvidos pela comunidade para as mais diversas funcionalidades;
- **Integração com outras ferramentas**: pode ser integrado a outra ferramentas como Jenkins e Nagios.

Conceitos

INVENTORY

É uma lista com os hosts que o Ansible irá gerenciar. O inventory pode ser definido em um arquivo YAML ou INI e é possível agrupar os hosts.

example.ini

```ini
mail.example.com

[webservers]
foo.example.com
bar.example.com

[dbservers]
one.example.com
two.example.com
three.example.com
```

example.yaml

```yaml
ungrouped:
  hosts:
    mail.example.com
webservers:
  hosts:
    foo.example.com
    bar.example.com
dbservers:
  hosts:
    one.example.com
    two.example.com
    three.example.com
```

MODULES

São pequenos programas executáveis do próprio Ansible que realizam tarefas específicas no host remoto. Existem módulos para manipulação de arquivos, instalação de pacotes, gerenciamento de usuários, execução de comandos, entre outros.

Conceitos

PLAYBOOKS

São arquivos YAML com as descrições das configurações, tarefas e procedimentos que serão executados. Um playbook pode conter um ou vários plays, que são as tarefas a serem executadas em hosts específicos.

TASKS

São as unidades básicas de trabalho de um playbook, sendo compostas por módulos Ansible a fim de executar ações específicas como a instalação de um pacote, iniciar serviços, executar comandos, etc.

VARIÁVEIS

Armazenam valores que podem ser referenciados ao longo de todo o playbook.

TAGS

Rótulos que podem ser atribuídos a playbooks e tasks. A utilização de tags pode ser útil para executar apenas partes específicas de um playbook

Conceitos

FACTS

São informações sobre o sistema em que o Ansible está sendo executado. Elas são coletadas automaticamente e podem ser usadas no playbook, facilitando a tomada de decisões condicionais com base no estado do sistema e no uso de condicionais.

HANDLERS

São tarefas executadas apenas quando eventos específicos ocorrem, como uma falha de tarefa ou uma mudança de estado. São usados principalmente para reiniciar serviços após uma alteração de configuração.

ROLES

É uma maneira de organizar e reutilizar Playbooks. Basicamente, é uma coleção de arquivos YAML estruturados contendo tarefas, variáveis, manipuladores e outros componentes importantes na automação.

Instalação

1. É necessário instalar o ***pipx***

```
$ python3 -m pip install --user pipx
$ python3 -m pipx ensurepath
```

1.1. Caso seja necessário instalar o **pip**:

```
$ curl -O https://bootstrap.pypa.io/get-pip.py
$ python3 get-pip.py
```

2. Com a instalação do pipx, agora podemos instalar o ansible:

```
$ pipx install --include-deps ansible
```

3. Caso seja necessário instalar algum módulo adicional, basta realizar o comando:

```
$ ansible-galaxy collection install community.docker
```

trocando o *community.docker* pela coleção que deseja adicionar

Principais Módulos

BREVE INTRODUÇÃO

A partir de agora, vamos ver alguns dos principais módulos, como e quando usá-los.

ansible.builtin.apt

Utilizado para realizar instalação de pacotes adicionais nos hosts baseados em Debian.

Para sistemas baseados em outras distribuições é necessário buscar o pacote do gerenciador específico, como o pacman do Archlinux.

example.yaml

```yaml
- name: Instalar o nginx # nome da task
  ansible.builtin.apt: # definindo o módulo que será usado
    name: nginx # lista com nome dos pacotes a serem instalados

- name: Instalar um pacote .deb
  ansible.builtin.apt:
    deb: /tmp/software.deb # também é possível realizar instalação via arquivos .deb
```

Documentação

ansible.builtin.copy

Utilizado para copiar arquivos ou diretórios da máquina local ou remota para um local do host. Metainformações do sistema de arquivos podem ser definidas, mesmo quando o arquivo ou diretório já existe no sistema de destino.

example.yaml

```yaml
- name: Copiar arquivo e definir dono e permissões # nome da task
  ansible.builtin.copy: # definindo o módulo que será usado
    src: /foo.conf # caminho local do arquivo a ser copiado
    dest: /etc/foo.conf # caminho absoluto onde o arquivo deve ser copiado
    owner: foo # nome do usuário que deve possuir o objeto
    group: foo # nome do grupo que deve possuir o objeto
    mode: '0644' # as permissões que o arquivo deve possuir
```

Documentação

ansible.builtin.file

Utilizado para realizar o gerenciamento de arquivos ou diretórios da máquina nos hosts.
Bastante útil para criar, alterar permissões, mover ou excluir arquivos e diretórios.
Esse módulo é útil para sistemas Unix, quando o host for Windows é necessário utilizar outro módulo.

example.yaml

```yaml
- name: Altera dono, grupo e permissões do arquivo # nome da task
  ansible.builtin.file: # definindo o módulo que será usado
    path: /etc/foo.conf # caminho do objeto alvo
    owner: foo # nome do usuário que deve possuir o objeto
    group: foo # nome do grupo que deve possuir o objeto
    mode: '0644' # as permissões que o arquivo deve possuir

- name: Cria um diretório, caso não exista
  ansible.builtin.file:
    path: /tmp/dir # caminho do objeto alvo
    state: 'directory' # indica que todos os os subdir serão criados, caso não existam
    # state pode assumir os valores: absent, directory, file, hard, link, touch
    mode: '0755'

- name: Apaga arquivo
  ansible.builtin.file:
    path: /etc/foo.conf # caminho do objeto alvo
    state: absent # os diretórios serão excluídos recursivamente
# state: file → retorna o estado atual do path
# state: hard → um hard link (cópia espelhada do arquivo original) será criado ou alterado
# state: link → um symlink (arquivo que aponta para outro arquivo ou diretório) será criado ou alterado
# state: touch → um arquivo vazio será criado se o arquivo não existir
```

Documentação

ansible.builtin.command

Utilizado para realizar comandos nos hosts.
Entretanto, tem limitações caso queira utilizar operações do tipo “*” (curinga), “>” (redirecionamento de saída), “<” (redirecionamento de entrada), “|” (pipe, permitindo que a saída de um comando seja usada como entrada para o próximo), sendo recomendado a utilização do módulo de ansible.builtin.shell.

Já em hosts Windows, o módulo a ser usado é o ansible.windows.win_shell.

example.yaml

```yaml
- name: Copiar arquivo e definir dono e permissões # nome da task
  ansible.builtin.command: # definindo o módulo que será usado
    cmd: cat /file.txt # comando a ser executado
  register: output # registra uma variável com o conteúdo do comando

- name: Listar arquivos de um diretório
  ansible.builtin.command:
    argv: # uma maneira de passar o comando como uma lista em vez de uma string
      - ls
      - -la
      - /tmp
  register: output
```

Documentação

ansible.builtin.debug

Utilizado para mostrar valores durante a execução e é útil para realizar debug de variáveis e expressões sem interromper a execução do playbook.

example.yaml

```yaml
- name: Mostrar mensagem na tela # nome da task
  ansible.builtin.debug: # definindo o módulo que será usado
    msg: "Hello {{ user }}!" # mensagem a ser mostrada com interpolação de variáveis

- name: Listar arquivos de um diretório
  ansible.builtin.command:
    cmd: cat /file.txt
  register: output

- name: Mostrar o valor da variável
  ansible.builtin.debug:
    var: output # mostrando o valor de uma variável
```

Documentação

ansible.builtin.import_playbook

Utilizado para importar um arquivo contendo uma lista de plays para ser executado.

Seu uso não é dentro de um play, mas sim em nível superior.

example.yaml

```yaml
- name: Incluindo um playbook
  ansible.builtin.import_playbook: task.yaml
```

task.yaml

```yaml
- name: Playbook que irá ser importado
  hosts: localhost
  gather_facts: false
  tasks:
    - name: Mostrar mensagem na tela # nome da task
      ansible.builtin.debug: # definindo o módulo que será usado
        msg: "Hello {{ user }}!" # mensagem a ser mostrada com interpolação de variáveis
```

Documentação

ansible.builtin.include_tasks

Utilizado para importar um arquivo contendo uma lista de tarefas para ser executada no playbook.

example.yaml

```yaml
- name: Incluindo um playbook
  hosts: localhost
  gather_facts: false

  tasks:
    - name: Inclue arquivo com tasks
      ansible.builtin.include_tasks:
        file: task.yaml # arquivo a ser importado
```

task.yaml

```yaml
- name: Mostrar mensagem na tela # nome da task
  ansible.builtin.debug: # definindo o módulo que será usado
    msg: "Hello {{ user }}!" # mensagem a ser mostrada com interpolação de variáveis
```

Documentação

ansible.builtin.git

Útil para gerenciar *git checkouts*.

example.yaml

```yaml
- name: Clona repositório # nome da task
  ansible.builtin.git: # definindo o módulo que será usado
    repo: https://github.com/ansible/ansible.git # link do repositório para clonar
    dest: /home/documents/ansible # caminho onde o repositório será clonado
    single_branch: true # clona apenas uma branch
    version: devel # nome da branch que será clonada
```

Documentação

Está tudo claro até agora?

Existem diversos outros módulos que podem ser consultados aqui.

Principais Filtros

BREVE INTRODUÇÃO

Agora, vamos ver alguns filtros que podem ser utilizados para facilitar na manipulação das informações e montagem de templates, comandos, arquivos, entre outras atividades.

Filtros

ansible.builtin.from_json

Converte uma string JSON em uma variável Ansible

example.yaml

```yaml
- name: Testes.
  tasks:
    - name: Debug filtro
      ansible.builtin.debug:
        msg: "{{ '{\"a\": true, \"b\": 54, \"c\": [1,2,3]}' | ansible.builtin.from_json }}"
```

ansible.builtin.difference

Realiza a comparação entre listas

example.yaml

```yaml
- name: Testes.
  vars:
    - list1: [1,2,3,4,5]
    - list2: [6,4,3,10]
  tasks:
    - name: Debug filtro
      ansible.builtin.debug:
        msg: "{{ list1 | ansible.builtin.difference(list2) }}"
        # output: [1,2,5] (os elementos que estão em list1, mas não está em list2)
```

Filtros

ansible.builtin.split

Realiza o processo de split em strings, transformando em uma lista.

example.yaml

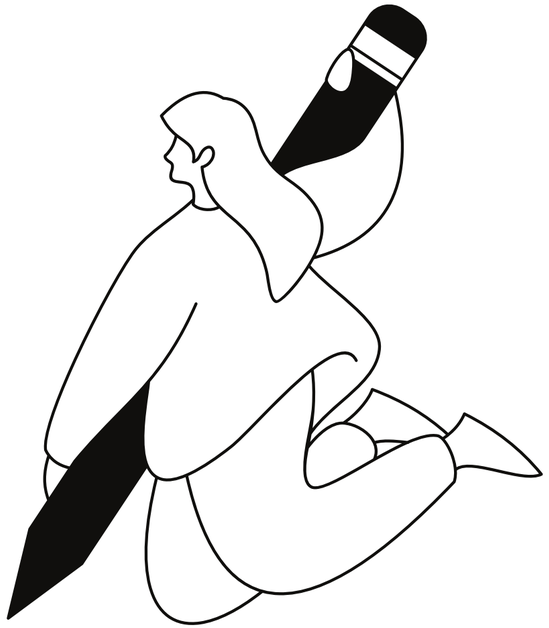
```yaml
- name: Testes.
  tasks:
    - name: Debug filtro
      ansible.builtin.debug:
        msg: "{{ 'maçã,mamão,banana' | ansible.builtin.split(',') }}"
        # output: ['maçã', 'mamão', 'banana']
```

ansible.builtin.realpath

Resolve links para retornar o caminho real de um determinado caminho.

example.yaml

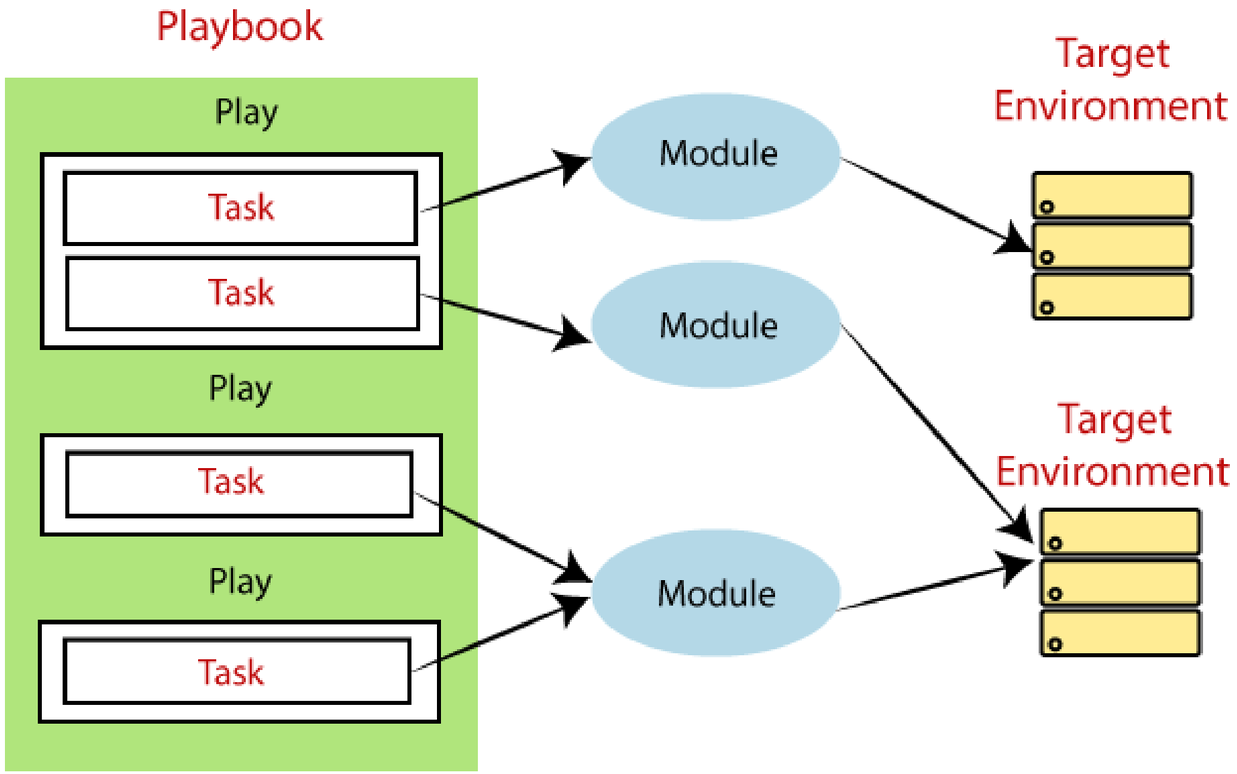
```yaml
- name: Testes.
  tasks:
    - name: Debug filtro
      ansible.builtin.debug:
        msg: "{{ 'task.yaml' | ansible.builtin.realpath }}"
        # output: /home/higor/Documentos/ansible-minicurso/task.yaml
```



Montando Playbooks

Agora que já introduzimos sobre algumas estruturas básicas de Ansible, vamos entender a estrutura de um Playbook e como montar.

Estrutura



fonte: https://www.javatpoint.com/ansible-playbooks

Cada Playbook é constituído por um ou mais plays.
Cada play é responsável por mapear um conjunto de atividades definidas para um conjunto de hosts.

Para executar um playbook o comando é:

```
$ ansible-playbook run <arquivo>.yml
```

Exemplo

example.yaml

```yaml
---
- name: Playbook de exemplo, para a pingar hosts
  hosts: localhost
  gather_facts: no
  tasks:
    - name: Pingar os hosts
      ansible.builtin.shell:
        cmd: ping -c 3 8.8.8.8
      register: ping

    - name: Mostrar o resultado
      ansible.builtin.debug:
        var: ping.stdout
```

L1: separador de arquivo
L2: definindo o nome do Play
L3: definindo os hosts os quais as tarefas serão executadas
L4: indica se o Ansible deve ou não coletar fatos sobre os hosts antes de executar as tarefas
L5: inicio da lista de tarefas
L6-L13: definição das tarefas, primeiro um ping ao servidor do Google e depois mostrar o resultado do ping

Loops

São usados para iterar sobre uma coleção de itens e operar alguma atividade sobre eles.

`iterar sobre listas`

example.yaml

```yaml
- name: Playbook de exemplo do uso de loops
  tasks:
    - name: Looping
      ansible.builtin.debug:
        msg: Hello {{ item }}!
      loop:
        - user1
        - user2
        - user3
```

Loops

iterar sobre dicionários

example.yaml

```yaml
- name: Playbook de exemplo do uso de loops
  hosts: localhost
  gather_facts: no
  vars:
    lista_presenca:
      - nome: "Carlos"
        presente: false
      - nome: "Maria"
        presente: true
      - nome: "Mario"
        presente: false
      - nome: "Vitória"
        presente: true
  tasks:
    - name: Looping
      ansible.builtin.debug:
        msg: "{{ item.nome }}"
      loop: "{{ lista_presenca }}"
```

Loops

É possível utilizar algumas funções para manipular os dados ou adicionar controle sobre cada execução do loop.

example.yaml

```yaml
---
- name: Playbook de exemplo do uso de loops com funções
  hosts: localhost
  gather_facts: no
  vars:
    frutas:
      maca: 5
      banana: 3
      laranja: 7
      melancia: 1
  tasks:
    - name: dict2items
      ansible.builtin.debug:
        msg: "Existem {{ item.value }} {{ item.key }}(s)"
      loop: "{{ frutas | dict2items }}" # transforma um dicionario em lista
      loop_control: # permite gerenciar o loop de diversas maneiras
        pause: 3 # pausa o loop por 3 segundos a cada execução
```

Blocks

São usados para agrupar tarefas, sendo bastante útil para aplicar operações ou condições a um conjunto de tarefas

example.yaml

```yaml
---
- name: Playbook de exemplo do uso de blocks
  hosts: localhost
  gather_facts: no
  become: yes # ativa privilégios e para executar é necessário passar a flag -K
  # ansible-playbook example.yaml -K
  tasks:
    - name: Configurar servidor web
      block:
        - name: Instalar Apache
          ansible.builtin.package: # módulo que ajuda no gerenciamento de pacotes
            name: apache2 # nome do pacote
            state: present # present para instalar e absent para remover um pacote
            use: apt # indica o gerenciador de pacote a ser usado
        - name: Iniciar o servidor Apache
          ansible.builtin.service: # módulo que auxilia no controle dos serviços do host
            name: apache2 # nome do serviço
            state: started # irá inicializar o serviço
            enabled: yes # indica que o serviço deve iniciar na inicialização
```

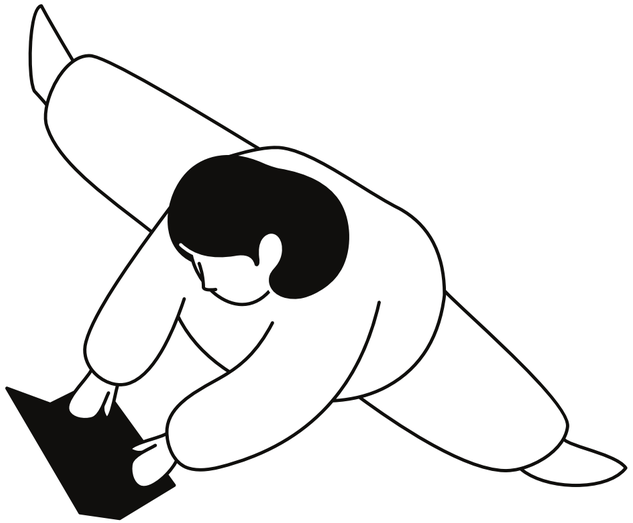
Condicional

São usadas para controlar o fluxo de execução de tarefas, tornando-as mais dinâmicas e adaptáveis para as mais diversas circunstâncias.
As condições em Ansible são implementadas utilizando a diretiva *when*.

example.yaml

```yaml
- name: Playbook de exemplo do uso de loops
  vars:
    lista_presenca:
      - nome: "Carlos"
        presente: false
      - nome: "Maria"
        presente: true
      - nome: "Mario"
        presente: false
      - nome: "Vitória"
        presente: true
  tasks:
    - name: Apenas os alunos presentes
      ansible.builtin.debug:
        msg: "{{ item.nome }}"
      when: item.presente
      loop: "{{ lista_presenca }}"
    - name: Apenas os alunos não presentes
      ansible.builtin.debug:
        msg: "{{ item.nome }}"
      when: not item.presente
      loop: "{{ lista_presenca }}"
```

DÚVIDAS?

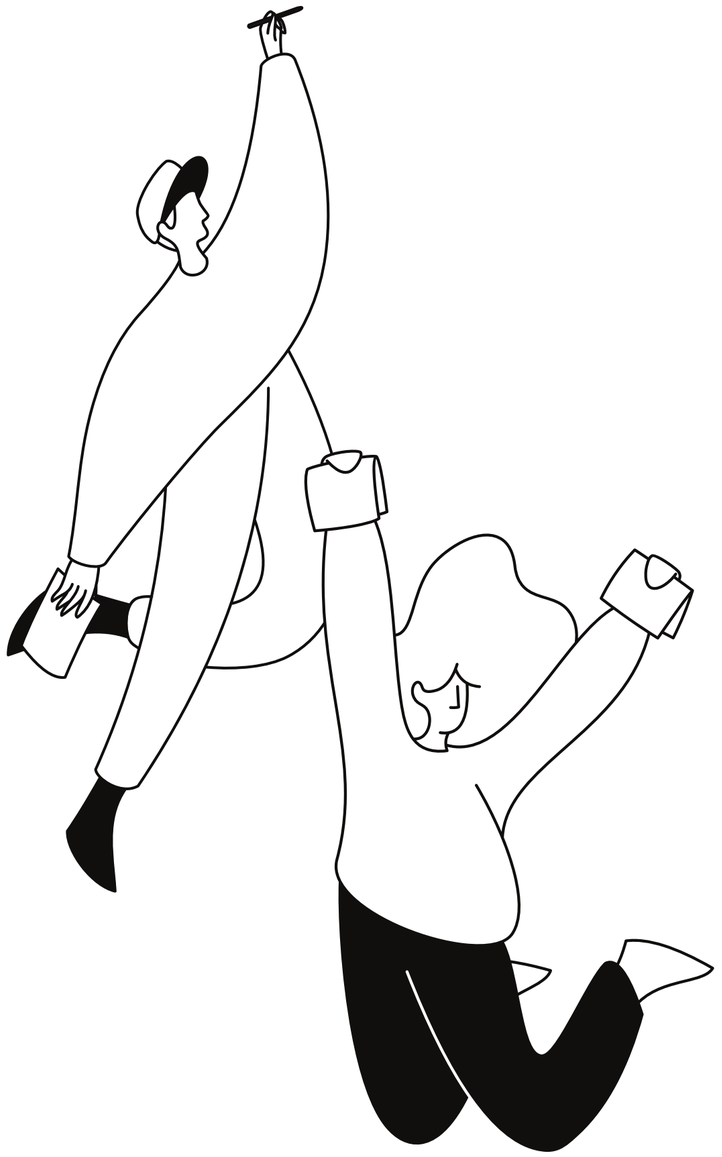


À SEGUIR

Prática

BORA PRATICAR?

Após tanto conceito e exemplos de código, para fixar o que foi visto sempre é bom por em prática.



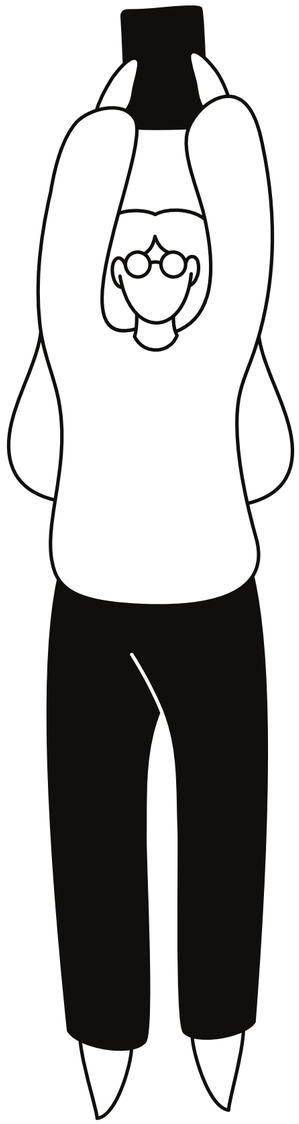
Atividade 1

No repositório clonado ao início da apresentação, abra a pasta ***atividades***. Entre os arquivos há o script.py que apenas realiza um print.

O objetivo é criar um playbook para executar esse script e depois ver o valor que foi impresso.

Dicas:

- Utilizar um dos módulos **ansible.builtin.shell**, **ansible.builtin.command** ou **ansible.builtin.script**
- Para imprimir o valor da saída utilizar o módulo **ansible.builtin.debug**



Atividade 1

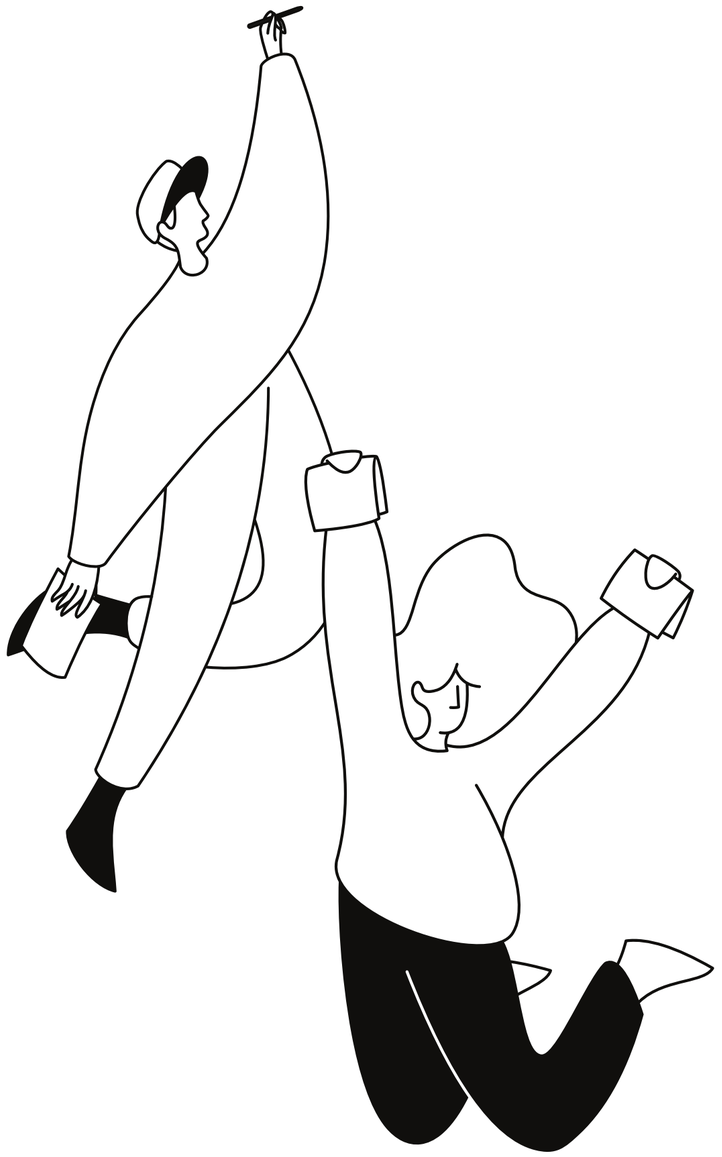
Solução:

script.py U | run-script.yaml U

atividades > soluções > run-script.yaml

```yaml
---
- name: Executar script python
  hosts: localhost
  tasks:
    - name: Executar script
      ansible.builtin.script:
        cmd: ../script.py
      register: output
      args:
        executable: python3
    # - ansible.builtin.command:
    #     cmd: python3 ../script.py
    #   register: output
    # - ansible.builtin.shell:
    #     cmd: python3 ../script.py
    #   register: output

    - ansible.builtin.debug:
        var: output
```



Atividade 2

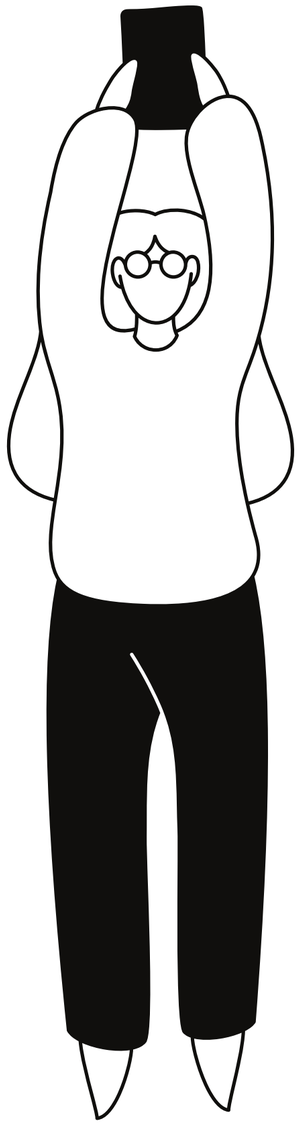
Imagine que há um projeto em Python e antes de executá-lo é preciso criar um ambiente virtual com as dependências do arquivo requirements.txt. E o ideal é automatizar essa geração para que seja facilmente replicável no ambiente de validação e de produção.

Faça um playbook que crie o virtual environment e realize a instalação das dependências.

Dicas:

- Utilize os módulos **ansible.builtin.file**, **ansible.builtin.command** e **ansible.builtin.pip**
- O comando para criar um ambiente virtual é

```
$ python3 -m venv <diretório>
```



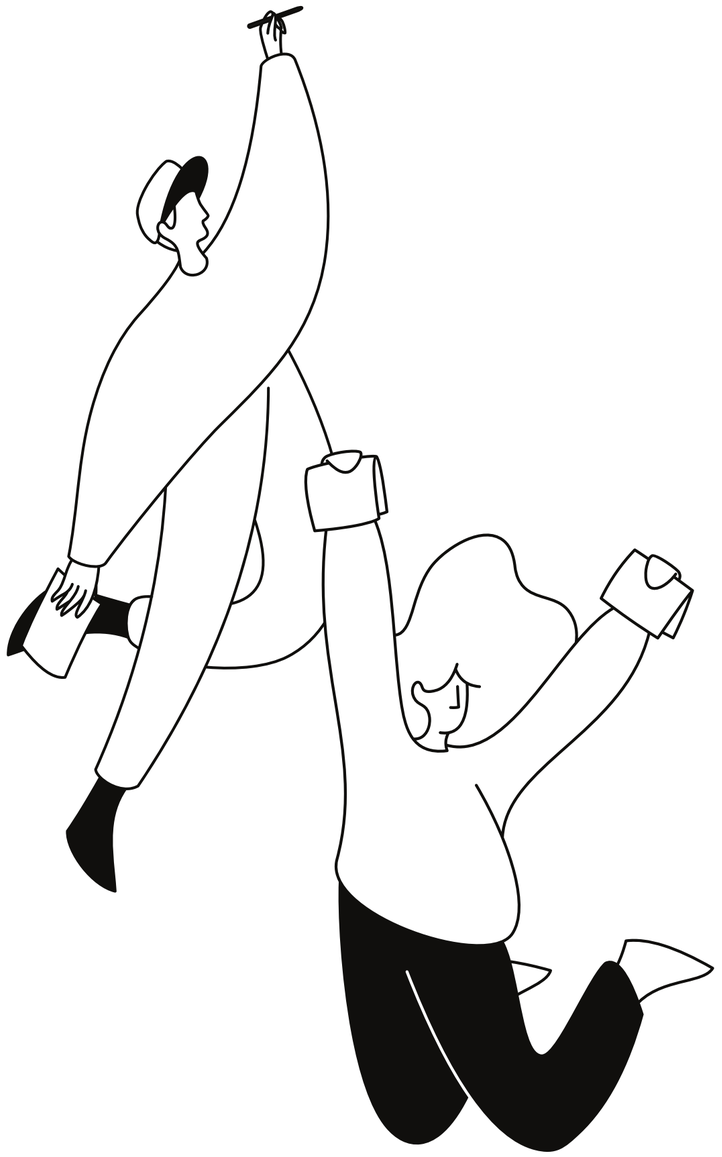
Atividade 2

Solução:

venv.yaml U

atividades > soluções > venv.yaml

```yaml
---
- name: Criar ambiente virtual Python
  hosts: localhost
  vars:
    venv_path: "{{ playbook_dir }}/venv"
  tasks:
    - name: Criar diretório para o ambiente virtual
      ansible.builtin.file:
        path: "{{ venv_path }}"
        state: directory
    - name: Criar ambiente virtual
      ansible.builtin.command: python3 -m venv {{ venv_path }}
    - name: Instalar pacotes Python usando pip no ambiente virtual
      ansible.builtin.pip:
        requirements: "{{ playbook_dir }}/../requirements.txt"
        virtualenv: "{{ venv_path }}"
```

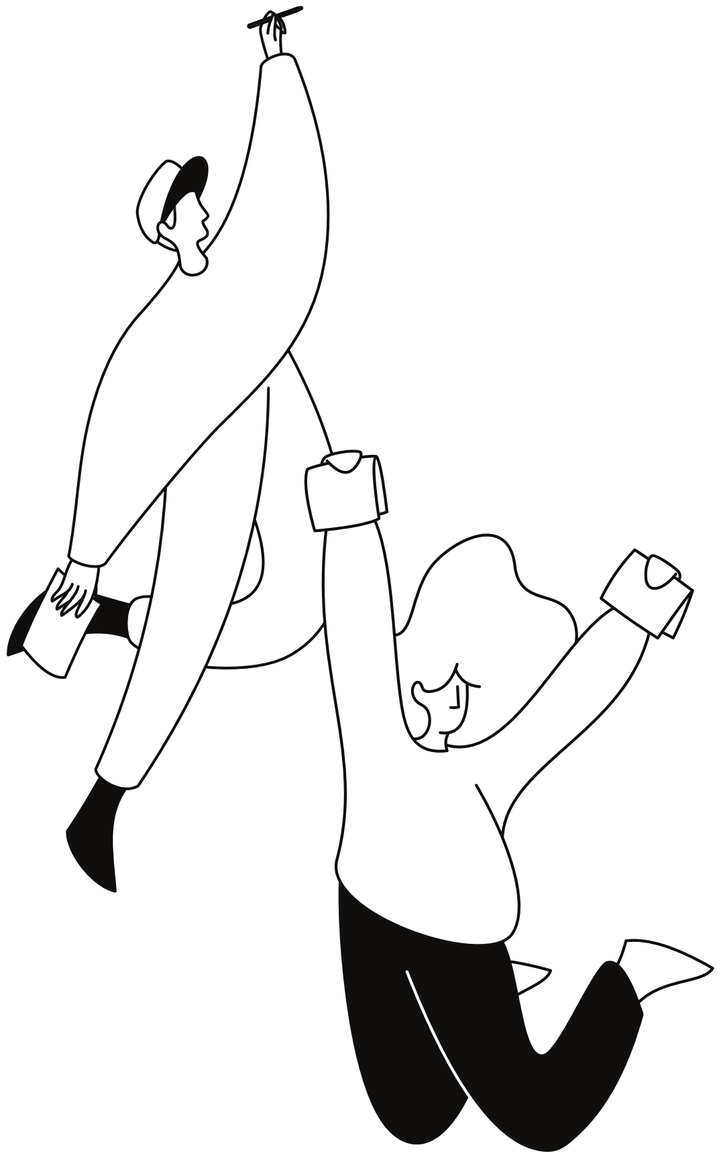


Atividade 3

Dentro do repositório clonado, há um projeto nomeado notification_service. Esse microsserviço tem a funcionalidade de receber informações sobre compras e enviar notificações aos usuários.

Sua atividade é criar um playbook que o execute, com o Kafka.
Note que já tem o Dockerfile e docker-compose que mostram o passo a passo de executar.

OBS.: Pode usar o docker-compose para iniciar o Kafka, mas o ideal é fazer a aplicação python inicializar via ansible.



Atividade 4

Dentro do repositório clonado, além do notification_service há outros dois microsserviços: order_processing_service e user_service. O primeiro é utilizado para processar compras e gerenciar produtos e o carrinho de compras, enquanto que o segundo gerencia os usuários e as lojas parceiras. Além disso, há o middleware que conecta todos os microsserviços.

Seu objetivo nessa atividade é montar um playbook conectando todas as 4 aplicações, montando assim o ecossistema de uma loja de e-commerce simples.
Essa atividade além de mostrar como subir microsserviços, mostra que pode ser usado para subir e conectar aplicações com as mais diversas linguagens de programação.

Obrigado pela participação!