

MINICURSO



Bem-vindos!

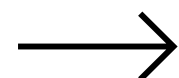
Para facilitar o acompanhamento da apresentação, inicie realizando o clone do repositório abaixo. Dessa forma, será possível ter acesso ao material desenvolvido para a parte prática do curso.

```
$ 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 x
example.ini
1 mail.example.com
2
3 [webservers]
4 foo.example.com
5 bar.example.com
6
7 [dbservers]
8 one.example.com
9 two.example.com
10 three.example.com
```

```
example.yaml x
example.yaml > ...
1 ungrouped:
2   hosts:
3     mail.example.com
4 webservers:
5   hosts:
6     foo.example.com
7     bar.example.com
8 dbservers:
9   hosts:
10    one.example.com
11    two.example.com
12    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 é o conjunto de 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 plays 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
```

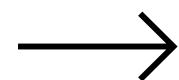
trocando o *community.docker* pela coleção que deseja adicionar





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 x

example.yaml > ...

```
1 - name: Instalar o nginx # nome da task
2   ansible.builtin.apt: # definindo o módulo que será usado
3     name: nginx # lista com nome dos pacotes a serem instalados
4
5 - name: Instalar um pacote .deb
6   ansible.builtin.apt:
7     deb: /tmp/software.deb # também é possível realizar instalação via arquivos .deb
```

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

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 x
example.yaml > ...
1  - name: Altera dono, grupo e permissões do arquivo # nome da task
2    ansible.builtin.file: # definindo o módulo que será usado
3      path: /etc/foo.conf # caminho do objeto alvo
4      owner: foo # nome do usuário que deve possuir o objeto
5      group: foo # nome do grupo que deve possuir o objeto
6      mode: '0644' # as permissões que o arquivo deve possuir
7
8  - name: Cria um diretório, caso não exista
9    ansible.builtin.file:
10     path: /tmp/dir # caminho do objeto alvo
11     state: 'directory' # indica que todos os os subdirs serão criados, caso não existam
12     # state pode assumir os valores: absent, directory, file, hard, link, touch
13     mode: '0755'
14
15  - name: Apaga arquivo
16    ansible.builtin.file:
17     path: /etc/foo.conf # caminho do objeto alvo
18     state: absent # os diretórios serão excluídos recursivamente
19     # state: file → retorna o estado atual do path
20     # state: hard → um hard link (cópia espelhada do arquivo original) será criado ou alterado
21     # state: link → um symlink (arquivo que aponta para outro arquivo ou diretório) será criado ou alterado
22     # 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 x
example.yaml > ...
1  - name: Copiar arquivo e definir dono e permissões # nome da task
2    ansible.builtin.command: # definindo o módulo que será usado
3      cmd: cat /file.txt # comando a ser executado
4      register: output # registra uma variável com o conteúdo do comando
5
6  - name: Listar arquivos de um diretório
7    ansible.builtin.command:
8      argv: # uma maneira de passar o comando como uma lista em vez de uma string
9        - ls
10       - -la
11       - /tmp
12     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 X
example.yaml > ...
1  - name: Mostrar mensagem na tela # nome da task
2    ansible.builtin.debug: # definindo o módulo que será usado
3      msg: "Hello {{ user }}!" # mensagem a ser mostrada com interpolação de variáveis
4
5  - name: Listar arquivos de um diretório
6    ansible.builtin.command:
7      cmd: cat /file.txt
8      register: output
9
10 - name: Mostrar o valor da variável
11   ansible.builtin.debug:
12     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 x
example.yaml > {} 0 > name
1 - name: Incluindo um playbook
2   ansible.builtin.import_playbook: task.yaml

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

ansible.builtin. include_tasks

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

```
example.yaml x
example.yaml
1  - name: Incluindo um playbook
2    hosts: localhost
3    gather_facts: false
4
5    tasks:
6      - name: Inclue arquivo com tasks
7        ansible.builtin.include_tasks:
8          file: task.yaml # arquivo a ser importado

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

[Documentação](#)

ansible.builtin.git

Útil para gerenciar *git checkouts*.

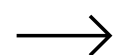
```
example.yaml x
example.yaml
1  - name: Clona repositório # nome da task
2    ansible.builtin.git: # definindo o módulo que será usado
3      repo: https://github.com/ansible/ansible.git # link do repositório para clonar
4      dest: /home/documents/ansible # caminho onde o repositório será clonado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

[Documentação](#)



Está tudo claro até agora?

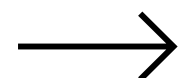
Existem diversos outros módulos que podem ser consultados aqui.





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`

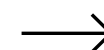
Converte uma string JSON em uma variável Ansible

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

`ansible.builtin.difference`

Realiza a comparação entre listas

```
example.yaml x
example.yaml
1  - name: Testes.
4    vars:
5      - list1: [1,2,3,4,5]
6      - list2: [6,4,3,10]
7    tasks:
8      - name: Debug filtro
9        ansible.builtin.debug:
10          msg: "{{ list1 | ansible.builtin.difference(list2) }}"
11          # 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 x
example.yaml
1  - name: Testes.
4    tasks:
5      - name: Debug filtro
6        ansible.builtin.debug:
7          msg: "{{ 'maçã,mamão,banana' | ansible.builtin.split(',') }}"
8          # output: ['maçã', 'mamão', 'banana']
```

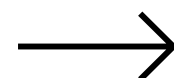
ansible.builtin.realpath

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

```
example.yaml x
example.yaml
1  - name: Testes.
4    tasks:
5      - name: Debug filtro
6        ansible.builtin.debug:
7          msg: "{{ 'task.yaml' | ansible.builtin.realpath }}"
8          # output: /home/higor/Documentos/ansible-minicurso/task.yaml
```

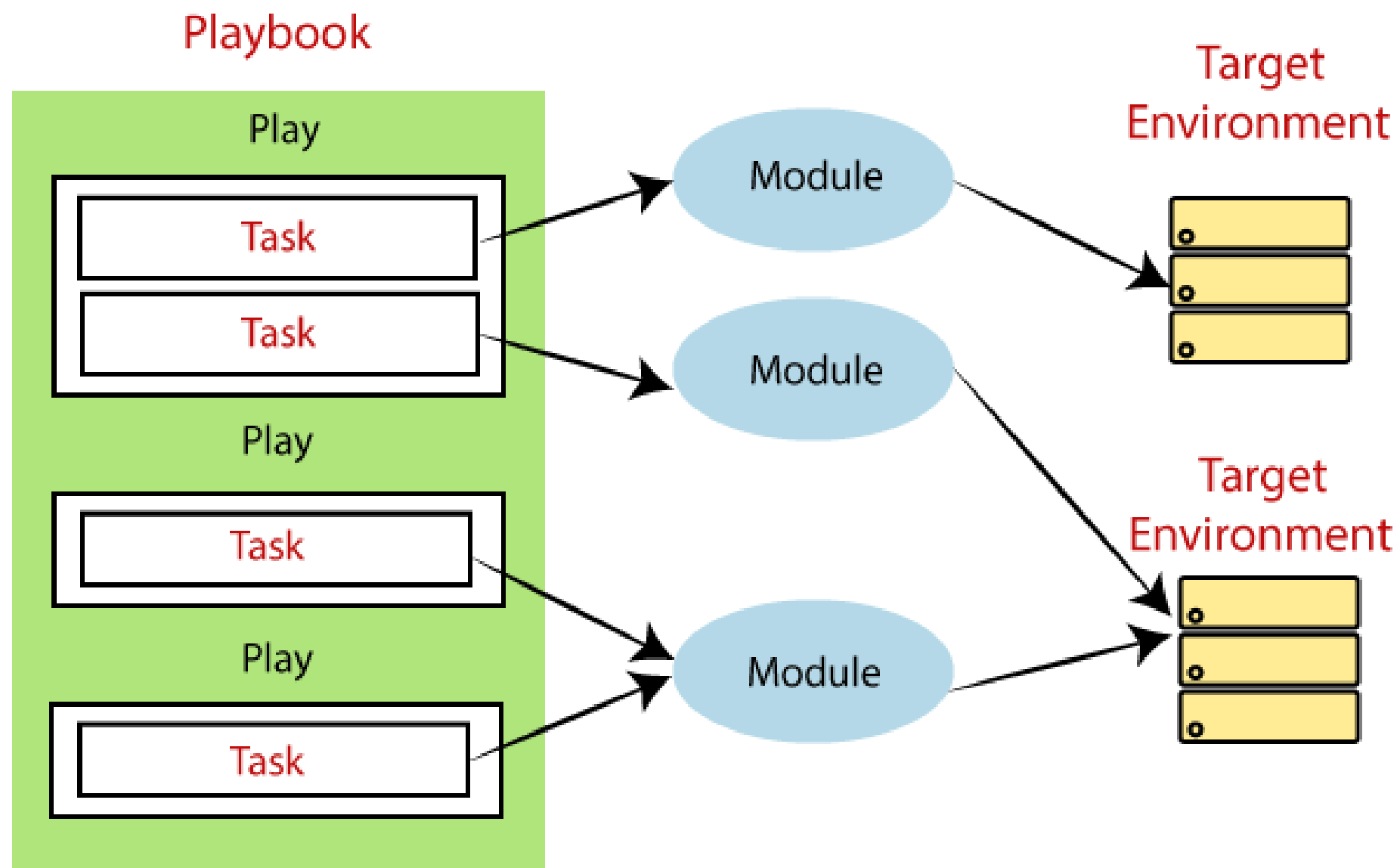


Montando Playbooks



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 <arquivo>.yaml
```

Exemplo

```
example.yaml x
example.yaml
1  —
2  - name: Playbook de exemplo, para a pingar hosts
3    hosts: localhost
4    gather_facts: no
5    tasks:
6      - name: Pingar os hosts
7        ansible.builtin.shell:
8          cmd: ping -c 3 8.8.8.8
9          register: ping
10
11     - name: Mostrar o resultado
12       ansible.builtin.debug:
13         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 x

example.yaml

```
2  - name: Playbook de exemplo do uso de loops
5      tasks:
6          - name: Looping
7              ansible.builtin.debug:
8                  msg: Hello {{ item }}!
9              loop:
10                 - user1
11                 - user2
12                 - user3
```

Loops

iterar sobre dicionários

```
example.yaml x
example.yaml
2  - name: Playbook de exemplo do uso de loops
3      hosts: localhost
4      gather_facts: no
5      vars:
6          lista_presenca:
7              - nome: "Carlos"
8                presente: false
9              - nome: "Maria"
10               presente: true
11              - nome: "Mario"
12                presente: false
13              - nome: "Vitória"
14                presente: true
15      tasks:
16          - name: Looping
17            ansible.builtin.debug:
18              msg: "{{ item.nome }}"
19            loop: "{{ lista_presenca }}"
```

Loops

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

```
example.yaml x
example.yaml
1  —
2  - name: Playbook de exemplo do uso de loops com funções
3    hosts: localhost
4    gather_facts: no
5    vars:
6      frutas:
7        maca: 5
8        banana: 3
9        laranja: 7
10       melancia: 1
11     tasks:
12       - name: dict2items
13         ansible.builtin.debug:
14           msg: "Existem {{ item.value }} {{ item.key }}(s)"
15         loop: "{{ frutas | dict2items }}" # transforma um dicionario em lista
16         loop_control: # permite gerenciar o loop de diversas maneiras
17         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 x
example.yaml
1  —
2  - name: Playbook de exemplo do uso de blocks
3    hosts: localhost
4    gather_facts: no
5    become: yes # ativa privilégios e para executar é necessário passar a flag -K
6    # ansible-playbook example.yaml -K
7    tasks:
8      - name: Configurar servidor web
9        block:
10          - name: Instalar Apache
11            ansible.builtin.package: # módulo que ajuda no gerenciamento de pacotes
12              name: apache2 # nome do pacote
13              state: present # present para instalar e absent para remover um pacote
14              use: apt # indica o gerenciador de pacote a ser usado
15          - name: Iniciar o servidor Apache
16            ansible.builtin.service: # módulo que auxilia no controle dos serviços do host
17              name: apache2 # nome do serviço
18              state: started # irá inicializar o serviço
19              enabled: yes # indica que o serviço deve iniciar na inicialização
```

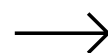
Condicional

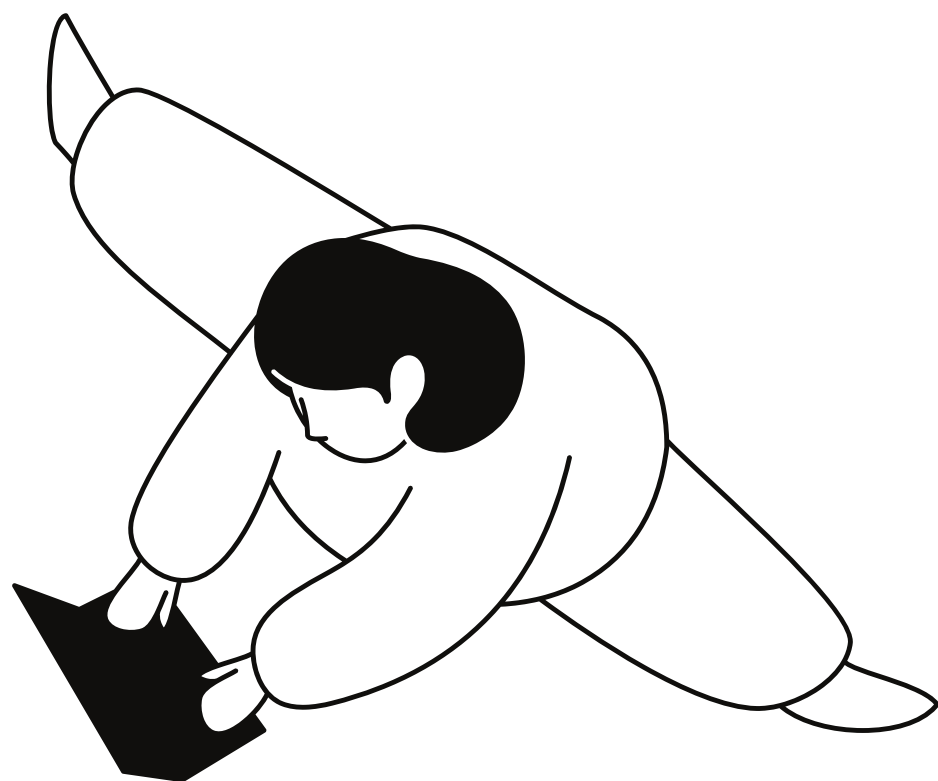
São usadas para controlar o fluxo de execução de tarefas, tornando-as mais dinâmicas e adaptáveis para as mais diversas circunstâncias. As condições em Ansible são implementadas utilizando a diretiva *when*.

```
example.yaml x
example.yaml
2  - name: Playbook de exemplo do uso de loops
5      vars:
6          lista_presenca:
7              - nome: "Carlos"
8                presente: false
9              - nome: "Maria"
10               presente: true
11              - nome: "Mario"
12               presente: false
13              - nome: "Vitória"
14               presente: true
15          tasks:
16              - name: Apenas os alunos presentes
17                ansible.builtin.debug:
18                  msg: "{{ item.nome }}"
19                  when: item.presente
20                  loop: "{{ lista_presenca }}"
21              - name: Apenas os alunos não presentes
22                ansible.builtin.debug:
23                  msg: "{{ item.nome }}"
24                  when: not item.presente
25                  loop: "{{ lista_presenca }}"
```



DÚVIDAS?

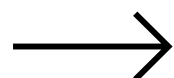




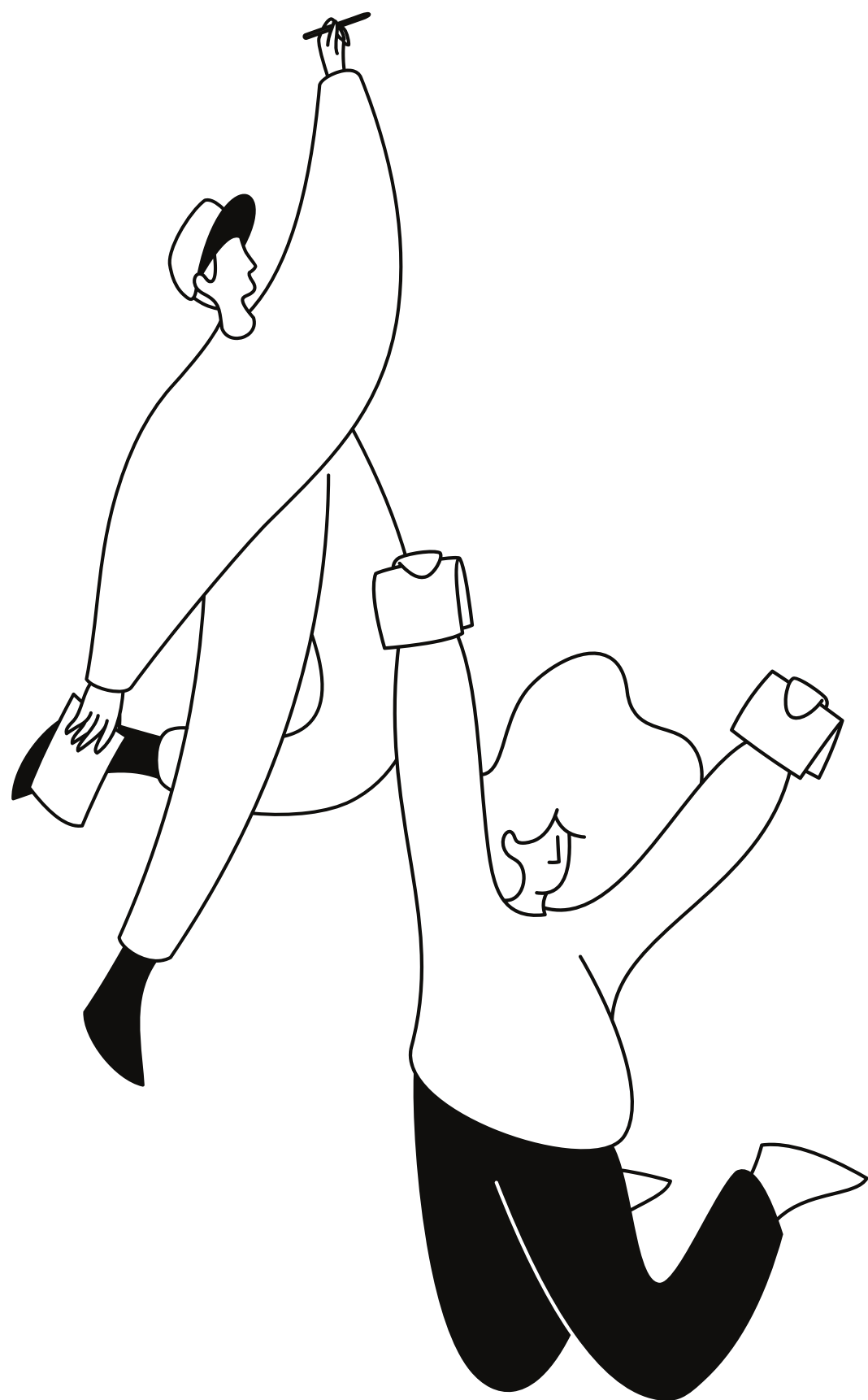
À SUIVIR

Prática

BORA PRATICAR?



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



Atividade 1

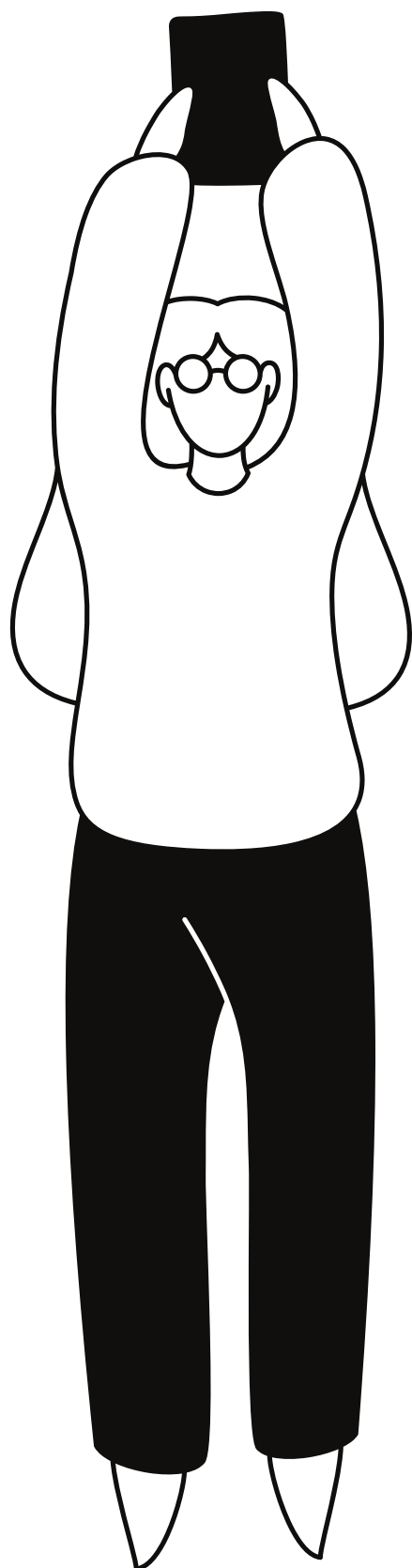
No repositório clonado ao início da apresentação, abra a pasta **atividades**. Entre os arquivos há o `script.py` que apenas realiza um `print`.

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`**

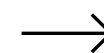


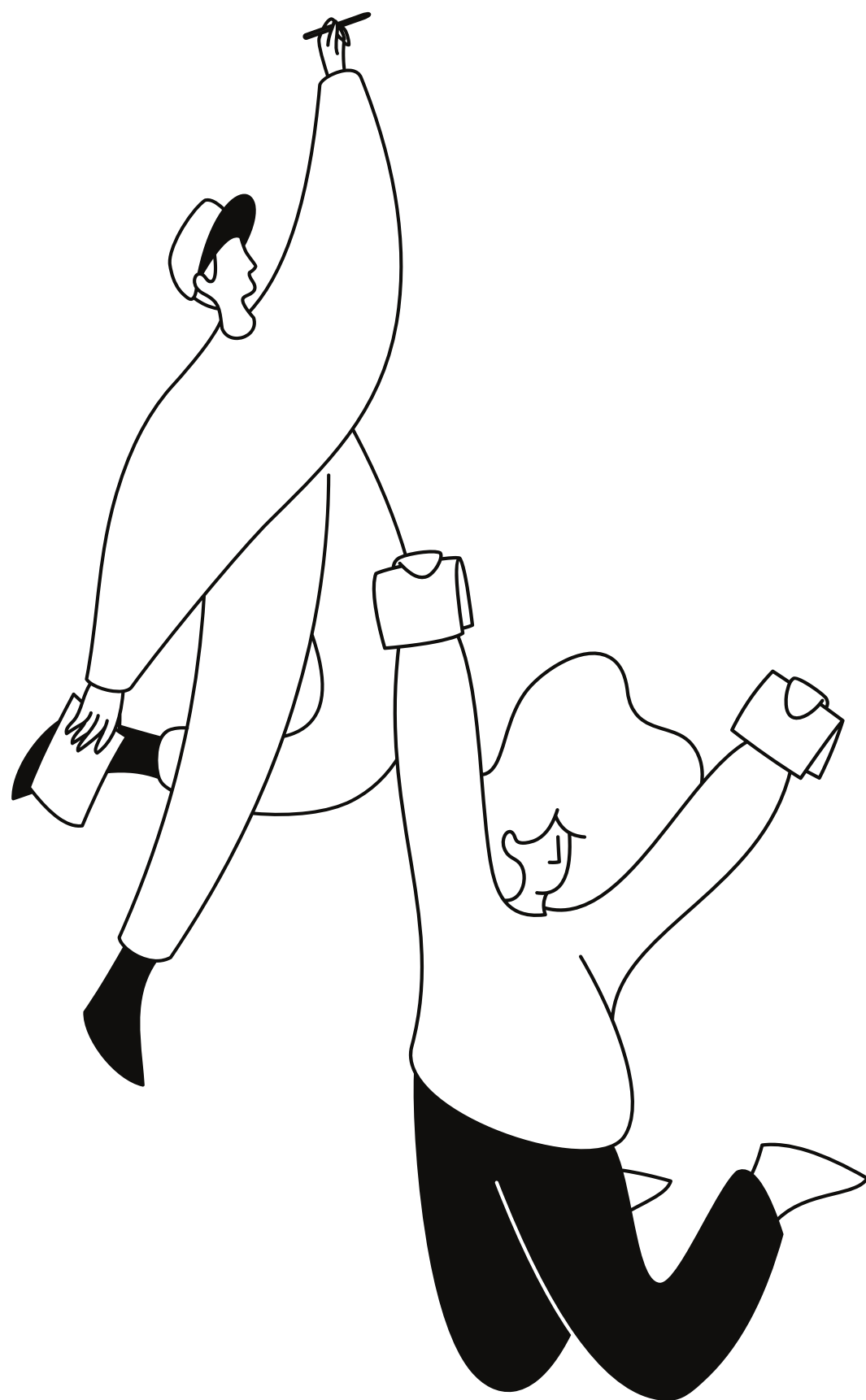


Atividade 1

Solução:

```
script.py U  run-script.yaml U x
atividades > soluções > run-script.yaml
1  —
2  - name: Executar script python
3    hosts: localhost
4    tasks:
5      - name: Executar script
6        ansible.builtin.script:
7          cmd: ../script.py
8          register: output
9          args:
10             executable: python3
11             # - ansible.builtin.command:
12             #   cmd: python3 ../script.py
13             #   register: output
14             # - ansible.builtin.shell:
15             #   cmd: python3 ../script.py
16             #   register: output
17
18     - ansible.builtin.debug:
19       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.

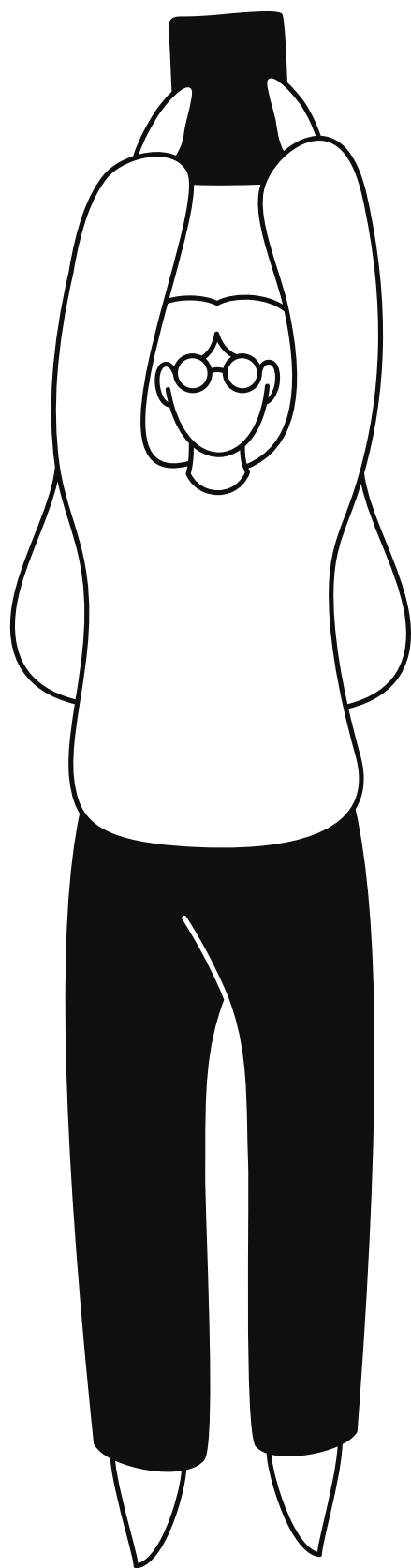
Faça um playbook que crie o virtual environment e realize a instalação das dependências.

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>
```



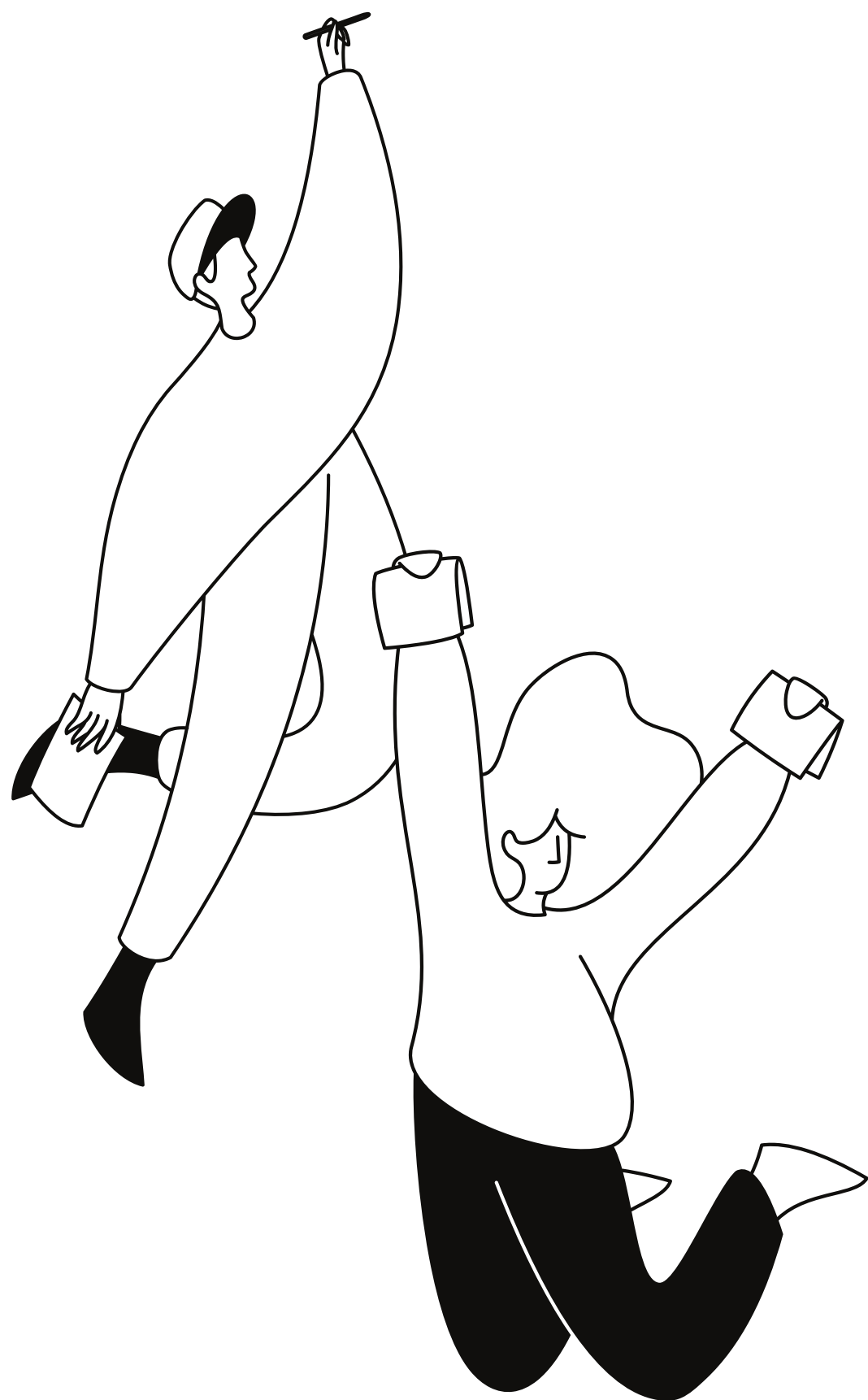


Atividade 2

Solução:

```
venv.yaml U X
atividades > soluções > venv.yaml
1  —
2  - name: Criar ambiente virtual Python
3    hosts: localhost
4    vars:
5      venv_path: "{{ playbook_dir }}/venv"
6    tasks:
7      - name: Criar diretório para o ambiente virtual
8        ansible.builtin.file:
9          path: "{{ venv_path }}"
10         state: directory
11      - name: Criar ambiente virtual
12        ansible.builtin.command: python3 -m venv {{ venv_path }}
13      - name: Instalar pacotes Python usando pip no ambiente virtual
14        ansible.builtin.pip:
15          requirements: "{{ playbook_dir }}/../requirements.txt"
16          virtualenv: "{{ venv_path }}"
```





Atividade 3

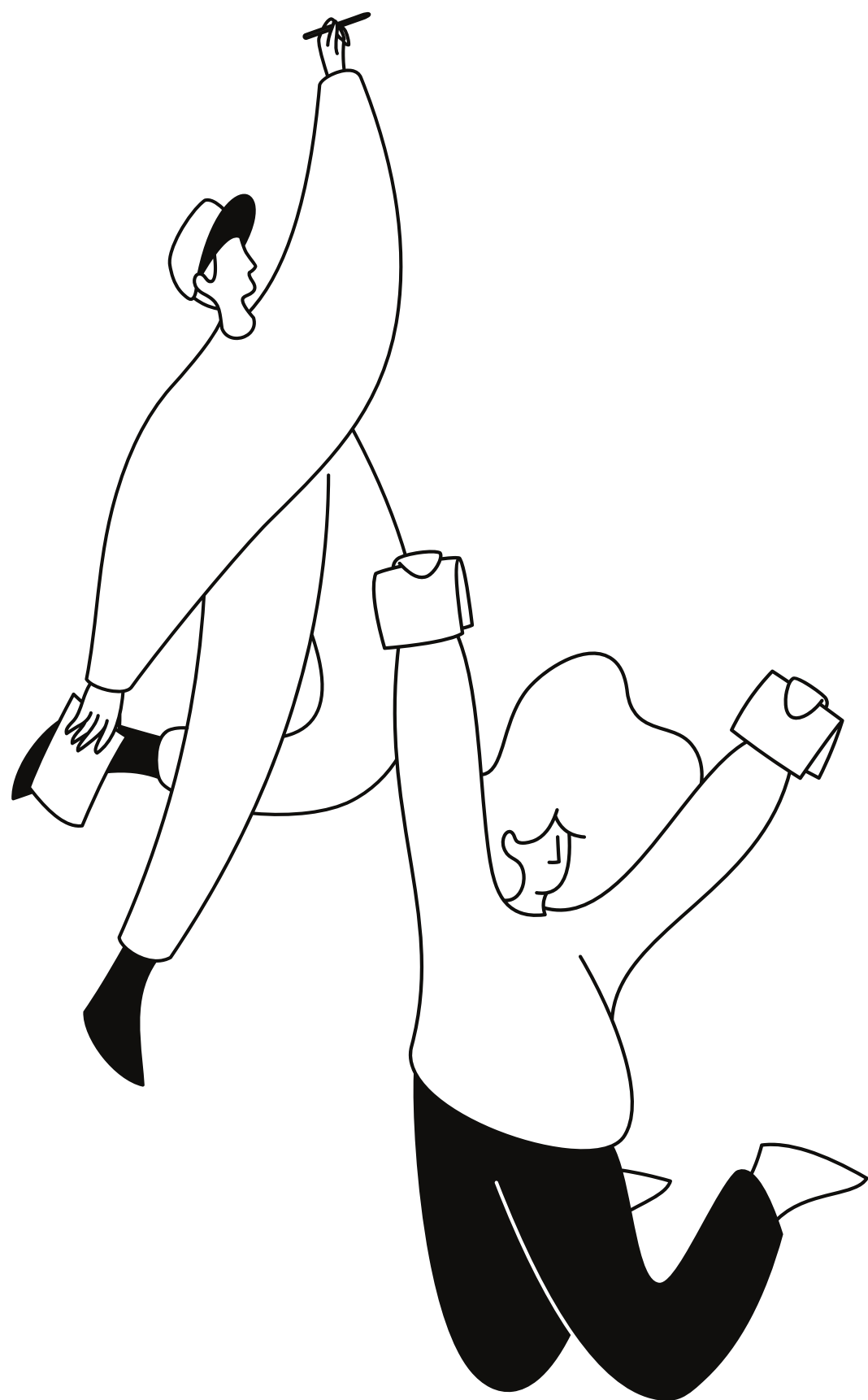
Dentro do repositório clonado, há um projeto nomeado `notification_service`. Esse microsserviço tem a funcionalidade de receber informações sobre compras e enviar notificações aos usuários.

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`.



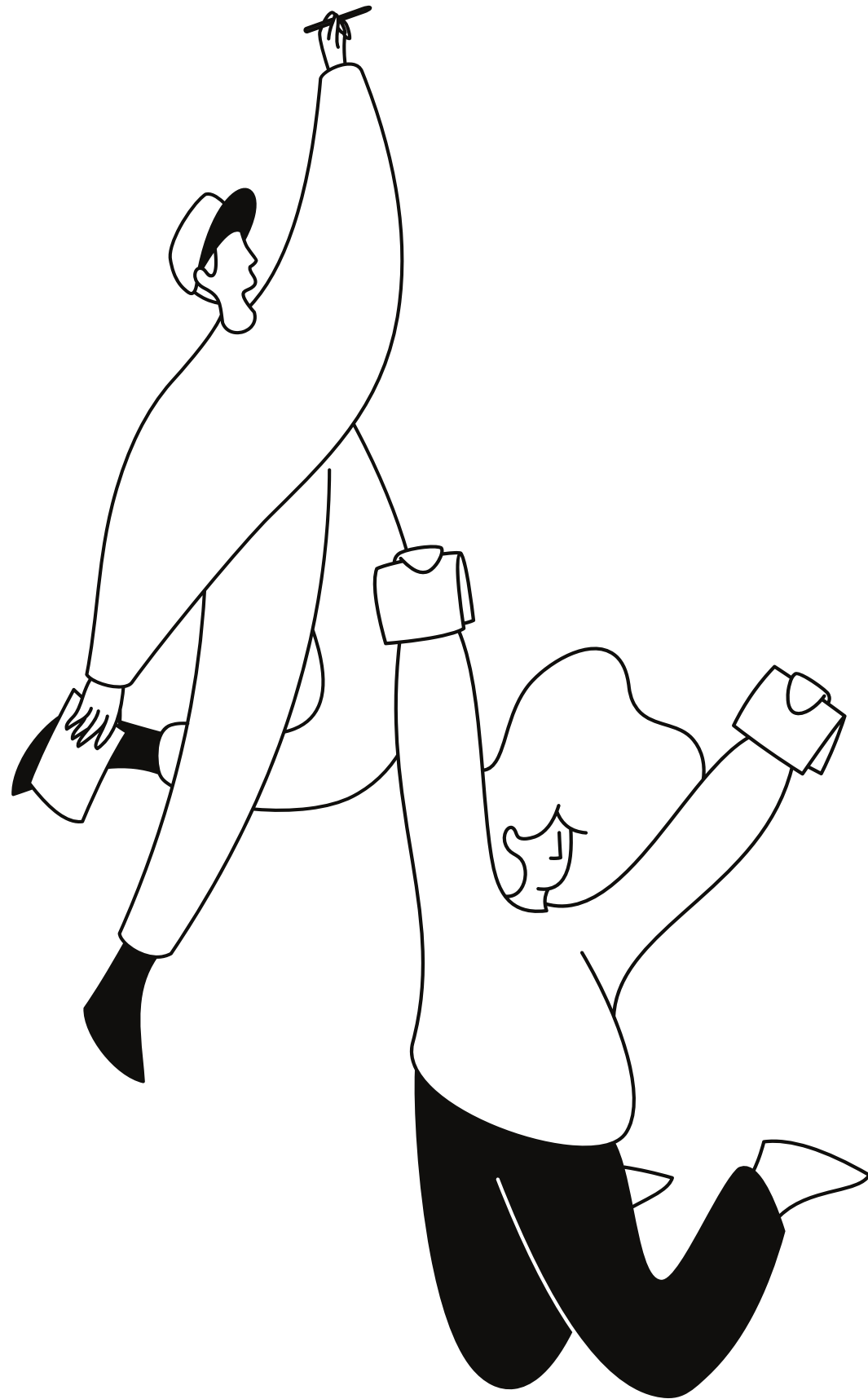


Atividade 4

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.

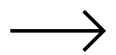




Atividade 4

Dicas:

- Observar os docker-compose e Dockerfile de cada repositório
- Para o notification-service:
 - é importante que a porta do Kafka esteja aberta antes da aplicação inicializar
 - altere os valores das variáveis de ambiente:
SMTP_LOGIN=mtessste1@gmail.com
SMTP_PASSWORD=behemkfrwvhpnfjy



**Obrigado pela
participação!**

