Parte 1 - Ansible core

Introdução

Bem-vindo ao nosso laboratório de Ansible-Core. Você deve ter recebido um e-mail nosso com dados de acesso ao laborátorio.

Abra esse e-mail e siga os passos a seguir.

Antes de acessar o laboratório verifique os itens abaixo:

- Notebook com acesso à Internet
- Navagador de Internet com Firefox ou Google Chrome
- 4 GB de memória RAM
- Linux: Centos ou RHEL (virtualizado ou físico)

Testando tudo

Agora você tem os dados do ambiente e notebook funcionando. Faça um teste e tente logar em cada servidor usando putty ou ssh (linux nativo)

Topologia do laboratório

Table 1. Tabela dos servidores e funcionalidades

Servidor	Função	S.O
serv001.betina.corp	Ansible engine e Servidor de aplicação	RHEL 7.4
serv002.betina.corp	Banco de Dados	RHEL 7.4

Table 2. Tabela de usuários e senhas

Usuário	Senha	Função
root	workshop2018	super usuário
ansible-core	redhat2018	usuário Ansible
redhat	redhat2018	usuário baixo privilégio

Let’s Rock

LAB01 - Instalação Ansible-Core

Primeiro passo é acessar o servidor Ansible-Engine e instale os pacotes relacionados ao Ansible-Core.

Instalação do Ansible Core passo - passo

```
yum  localinstall -y https://mirror.cedia.org.ec/epel/7/x86_64/Packages/e/epel-
release-7-11.noarch.rpm ①
yum install -y ansible  ②
```

① Instalação do repositório epel

② Instalação do Ansible Core

Configurando o usuário Ansible

Neste passo iremos criar um usuário chamado "ansible-core" em cada servidor servidor do laboratório . Este procedimento é necessário para evitar falhas de segurança e uso excessivo do usuário root

Comando para criação do usuário

```
adduser ansible-core
passwd  ansible-core ①
```

① Coloque a senha **'redhat2018'**

Procedimento para configuração do sudo para o usuário ansible-core para cada servidor

```
cat << EOF >/etc/sudoers.d/ansible-core ①
ansible-core ALL = (root) NOPASSWD:ALL
EOF
```

① Repita este procedimento em todos os servidores.

Criando e compartilhando chave SSH

```
ssh-keygen <<tecle enter>>
ssh-copy-id ansible-core@serv001.betina.corp
ssh-copy-id ansible-core@serv002.betina.corp
```

LAB02 - Configurando um inventário manualmente

Neste laborátorio iremos criar um inventário para nosso laboratório

Criando inventário

```
su ansible-core
vi inventario.ini
```

Exemplo de inventário para este laboratório

```
[all:vars]
 ansible_ssh_user=ansible-core
[web]
 web1 ansible_ssh_host=serv001.betina.corp
[banco]
 bd1 ansible_ssh_host=serv002.betina.corp
```

LAB03 - Módulos para execução de comandos

Utilizando o comando externo - uptime

```
ansible all -i inventario.ini -m command -a "uptime"
```

Instalando um pacote diretamente num grupo de hosts

```
ansible web -s -i inventario.ini -m yum -a "name=httpd state=present"
```

Inicializando serviço http via comando

```
ansible web -s -i inventario.ini -m service -a "name=httpd enabled=yes state=started"
```

Usando o módulo ping

```
ansible -i inventario.ini all -m ping ①
```

① É possível testar se todos os servidores registrados dentro do inventário estão funcionais a nível de rede

Validando o nível de privilégio do usuário ansible-core

```
ansible -i inventario.ini all -m command -a id -b ①
```

① O resultado da ação do comando Ansible terá como saída qual nível de privilégio

Desafio

1. Utilize o modulo ping para pingar todos os servidores
2. Instale o telnet apenas nos servidores web
3. Defina o Selinux para permissive:

Resposta do desafio

1. Utilize o modulo ping para pingar todos os servidores: ***ansible -i inventario.ini all -m ping***
2. Instale o telnet apenas nos servidores web: ***ansible web -s -i inventario.ini -m yum -a "name=telnet state=present"***
3. Defina o Selinux para permissive: ***ansible all -s -i inventario.ini -m command -a "setenforce permissive"***
4. Comando para listar todos os serviços: ***ansible all -i inventario.ini -m command -a "systemctl status"***

LAB04 - Construindo primeiro playbook

Criando o primeiro playbook

```
Loge com usuário ansible-core
su ansible-core
vi ~./primeiroplaybook.yaml ①
```

① Crie o arquivo utilizando vim que será utilizado como ferramenta para escrever os playbook

Utilize o modelo abaixo como padrão

```
---     ⑥
-
  name: Primeiro playbook
  hosts: web ①
  become: yes
  vars:
  remote_user: ansible-core ②

  tasks:
  - name: Instala a ferramenta net-tools ③
    yum: name=net-tools state=latest ④ ⑤
```

① Nome do grupo de hosts

② Usuário que irá realizar a operação

③ Nome da tarefa

④ Modulo yum sendo utilizado para instlalação do pacote net-tools na última versão

⑤ Nunca utilize TAB apenas espaço

⑥ Sempre inicie o seu script ansible com ---

Salve o seu playbook

Utilizando o vim salve todas as alterações do script ansible e execute a sequencia de comandos para salvar e sair do vim ':wq!'

Valide se seu playbook tem alguma erro

```
ansible-playbook -C -i inventario.ini  primeiroplaybook.yaml
```

Execute o playbook

```
ansible-playbook -i inventario.ini  primeiroplaybook.yaml
```

Utilizando loop

Utilizando o vim crie o segundo playbook com o nome *segundoplaybook.yaml*

```
---
-
  name: Segundo Playbook - trabalhando com loop
  hosts: web
  remote_user: ansible-core
  become: yes
  gather_facts: no
  vars:
   state: latest

  tasks:
  - name: Instalando Apache e PHP
    yum: name={{ item }} state={{ state }}
    with_items:
      - httpd
      - php
```

Execute o playbook

```
ansible-playbook -i inventario.ini  ~./segundoplaybook.yaml
```

LAB05 - Trabalhando com Handlers "Manipuladores"

O que são Handlers ? Qual é sua importância ?

Semelhante a uma tarefa, exceto que os handlers executam somente em resposta a uma tarefa configurada para notificar o handler na mudança de estado.

Exemplo de um playbook que utiliza handlers para gerenciar o serviço do Apache

```
---
-
 name: Trabalhando com Handlers
 hosts: web
 remote_user: ansible-core
 become: yes

 tasks:
   - name: Testando handlers do Apache
     yum: name={{ item }} state=installed
     with_items:
       - httpd
       - memcached
     notify: Restart Apache

   - template: src=templates/httpd.conf.j2 dest=/etc/httpd/conf/httpd.conf
     notify: Restart Apache

 handlers:
   - name: Restart Apache
     service: name=httpd state=restarted
```

Utilizando o vim crie o terceiro playbook utilizando o modelo acima e com o nome ***terceiroplaybook.yaml***

Execute o playbook

```
ansible-playbook -i inventario.ini  ~./terceiroplaybook.yaml
```

LAB06 - Trabalhando com TAGS

Por que devo usar Tags ?

Se você tiver um grande playbook, o uso de TAGs tornar-se útil para executar uma parte específica do playbook, sem executar todo o playbook.

```
---
-
 name: Trabalhando com tags
 hosts: web
 remote_user: ansible-core
 become: yes

 tasks:
   - name: instala httpd e memcached ou configura
     yum: name={{ item }} state=installed
     with_items:
       - httpd
       - memcached

     tags:
       - packages
       - template: src=templates/src.j2 dest=/etc/foo.conf

     tags:
       - configuration
```

Utilizando tags

Utilizando o vim crie o quarto playbook com o nome ***quartoplaybook.yaml***

Executando playbook com tags

Executando apenas a tag configuration

```
ansible-playbook -i inventario.ini  quartoplaybook.yaml --tags “configuration”
```

Executando apenas a tag notification

```
ansible-playbook -i inventario.ini  quartoplaybook.yaml --skip-tags "notification"
```

Executando tags padrão do Ansible

```
ansible-playbook example.yaml --tags “tagged” ①
ansible-playbook example.yaml --tags “untagged” ②
ansible-playbook example.yaml --tags “all” ②
```

① Será executada todas as tarefas que tenham uma tag amarrada

② Será executada todas as tarefas sem tag

③ Executa todas as tarefas independente da tag

Utilizando tags

Utilizando o vim crie o quinto playbook com o nome ***quintoplaybook.yaml***

Execute o playbook

```
ansible-playbook -i inventario.ini  ~./quintoplaybook.yaml
```

LAB07 - Trabalhando com condicional

Quando devo utilizar condicional ?

O uso de condicionais se da quando temos situações onde não sabemos exatamente qual sistema ou condicação exata que será encontrada.

Neste caso o condicacional consegue aplicar uma condicação para validar se o alvo condiz com contexto do playbook e se combinar, executar o restante do playbook.

```
---
-
 name: Trabalhando com Condocional
 hosts: web
 remote_user: ansible-core
 become: yes

 tasks:
 - name: install Apache
   yum: name=httpd state=removed
   when: ansible_os_family == "RedHat"
```

Valide com comando

```
sudo yum history list 12
```

LAB08 - Trabalhando com com saída de comandos

```
---
-
 name: Trabalhando com  saida de comandos
 hosts: web
 remote_user: ansible-core
 become: yes

- name: Saida do comando httpd
  shell: httpd -v|grep version|awk '{print $3}'|cut -f2 -d'/'
    register: result

- debug: var=result
```

Testando saída de comando

Utilizando o vim crie o sexto playbook com o nome ***sextoplaybook.yaml***

Execute o playbook

```
ansible-playbook -i inventario.ini  ~./sextoplaybook.yaml
```

LAB09 - Ignorando erros

```
---
-
  name: Ignorando errors
  hosts: web
  remote_user: ansible-core
  become: yes

  - name: ping host
    command: ping -c1 www.uolbbb.com.jp
    ignore_errors: yes

  - name: remove apache mesmo depois do uolbbb.com.jp nao pingar
    yum: name=httpd state=absent
```

Testando a função para ignorar erros

Utilizando o vim crie o setimo playbook com o nome ***setimoplaybook.yaml***

Execute o playbook

```
ansible-playbook -i inventario.ini  ~./setimoplaybook.yaml
```

Tratando mais erros

```
---
-
  name: Ignorando errors
  hosts: web
  remote_user: ansible-core
  become: yes

  tasks:
   - block:
   - debug: msg='i execute normally'
   - command: /bin/false
   - debug: msg='i never execute, cause ERROR!'
 rescue:
   - debug: msg='I caught an error'
   - command: /bin/false
   - debug: msg='I also never execute :-('
 always:
   - debug: msg="this always executes"
```

Testando a função para ignorar erros

```
Utilizando o vim crie o  setimo e meio playbook com o nome *_setimoplaybook-2.yaml_*
```

Execute o playbook

```
ansible-playbook -i inventario.ini  ~./setimoplaybook-2.yaml
```

LAB10 - Tratando arquivos

Imagine uma situação onde você precisa alterar uma única linha de um arquivo de configuração em mais de 100 servidores, complicado ?

```
---
-
  name: Tratando arquivos Selinux e HTTPD
  hosts: web
  remote_user: ansible-core
  become: yes

  tasks:
    - name: Tratando o arquivo de configuração selinux
      lineinfile: dest=/etc/selinux/config regexp=^SELINUX=  ①
      line=SELINUX=enforcing

    - name: Tratando o arquivo de configuração httpd
      lineinfile: dest=/etc/httpd/conf/httpd.conf regexp="^Listen " ②
      insertafter="^#Listen " line="Listen 8080"
```

① Abre o arquivo /etc/selinux/config e altera a linha para SELINUX=enforcing

② Abre o arquivo /etc/http/conf/httpd.conf e altera a linha para Listen 8080

Testando a função para de tratamento de arquivos

```
Utilizando o vim crie o  oitavo playbook com o nome *_oitavoplaybook-2.yaml_*
```

Execute o playbook

```
ansible-playbook -i inventario.ini  ~./oitavoplaybook-2.yaml
```

LAB11 - Trabalhando com variáveis

Ansible não é uma linguagem de programação, mas possui vários recursos de linguagem de programação, e uma das mais importantes é o uso variáveis.

Exemplo no uso de variáveis no Ansible

```
---
-
  name: Trabalhando com variaveis
  hosts: web
  remote_user: ansible-core
  become: yes

    - name: debug
     hosts: all

    tasks:
     - name: Show hostvars[inventory_hostname]
       debug: var=hostvars[inventory_hostname]

     - name: Show ansible_ssh_host variable in hostvars
       debug: var=hostvars[inventory_hostname].ansible_ssh_host

     - name: Show group_names
       debug: var=group_names

     - name: Show groups
       debug: var=groups
```

Testando a função para de tratamento de arquivos

Utilizando o vim crie o nono playbook com o nome ***nonoplaybook.yaml***

Execute o playbook

```
ansible-playbook -i inventario.ini  ~./nonoplaybook-2.yaml
```

LAB12 - Trabalhando com templates

Se você fez a programação na Web, provavelmente usou um sistema de modelo para gerar HTML. Caso não tenha, um modelo é apenas um arquivo de texto que possui sintaxe especial para especificar variáveis que devem ser substituídas por valores.

Se você já recebeu um email automatizado de uma empresa, provavelmente está usando um modelo de e-mail.

Ansible usa o mecanismo de modelo ***Jinja2*** para implementar modelos

```
---
-
  name: Trabalhando com template jinja2
  hosts: web
  remote_user: ansible-core
  become: yes
    vars: ④
    http_port: 80
    max_clients: 200
  remote_user: root

  tasks:
  - name: Valida que o Apache esteja na última versão
    yum: name=httpd state=latest ③

  - name: Substituia o arquivo de configuração httd.conf ②
    template: src=/template/httpd.j2 dest=/etc/http/httpd.conf ①
    notify:
    - restart apache

  - name: ensure apache is running (and enable it at boot)
    service: name=httpd state=started enabled=yes

  handlers:
    - name: restart apache
      service: name=httpd state=restarted ⑤
```

① Ansible copia arquivo /srv/httpd.j2 para /etc/httpd.conf

② Utilize as variaveis substituindo o arquivo de configuração /etc/http/httpd.conf

③ Valida que o pacote httpd na última versão

④ Variáveis que serão utilizada na substituição de vários parametros do arquivo de configuração "httpd.conf"

⑤ Este handlers garante que o serviço httpd será reinciado

Testando a função jinja2

Utilizando o vim crie o decimo playbook com o nome ***decimoplaybook.yaml***

Execute o playbook

```
ansible-playbook -i inventario.ini  ~./decimoplaybook-2.yaml
```

```
```