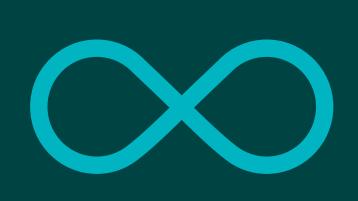


CURSO **525**

INFRAESTRUTURA ÁGIL COM PRÁTICAS DEVOPS







Automação para ambientes ágeis com Ansible

Características do Ansible

Anotações	

4LINU2	Objetivos da aula	2
1	O que é o Ansible	
2	Características	
3	Arquitetura	
4	Înstalação do Ansible	

Anotações		

O que é o Ansible?

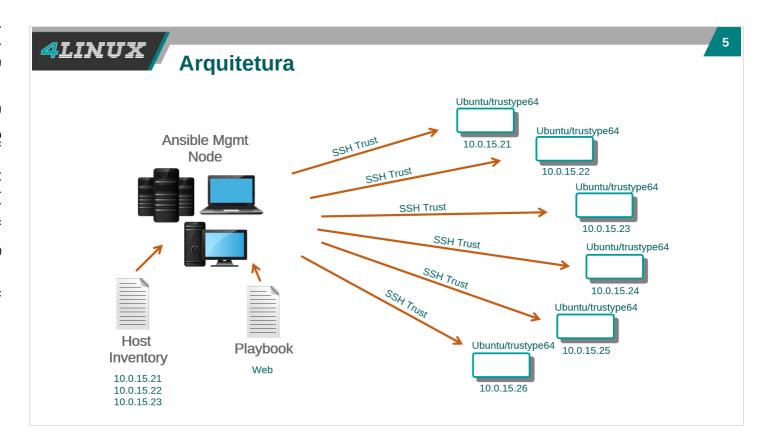
Ansible é uma ferramenta de automação, criada para gerenciar múltiplas máquinas de uma única vez, através de scripts numa linguagem simples chamada **YAML**.

Sua popularização atual se deve muito a sua documentação, considerada por muitos uma das mais detalhadas, além da curva de aprendizado ser mínima, permitindo que qualquer pessoa escreva suas próprias **playbooks**.



Anotações	

✓ Não necessita do uso de agents; ✓ Adaptável a qualquer situação.	✓ Arquitetura Master to Nodes;	✓ Linguagem de Configuração Simples
· · · · · · · · · · · · · · · · · · ·	✓ Não necessita do uso de agents;	✓ Adaptável a qualquer situação.
✓ Simples de realizar manutenção;	✓ Simples de realizar manutenção;	



O ansible possui uma arquitetura **MASTER TO NODES**, sendo necessário apenas que os nós possuam o Python instalado e uma conexão SSH com a máquina que deverá responder ao comando remoto, podendo ser realizada via usuário/senha ou até mesmo troca de chaves.

Uma das vantagens de se utilizar Ansible em ambientes Linux, é que na maioria das distribuições o Python já vem pré-instalado, o que não faz necessário nenhuma outra configuração no nó de destino, se não o Servidor SSH.

Instalação do Ansible

Executaremos a instalação do Ansible na máquina Automation.

- Acesse a pasta infraagil e conecte-se a máquina automation, utilizando o vagrant ssh:
 - # cd infraagil
 - # vagrant ssh automation
- Instale o repositório epel:
- # sudo yum install epel-release vim -y
- Instale o ansible:
- # sudo yum install ansible -y
- Verifique se o ansible foi instalado corretamente:
 - # ansible --version

Anotações		

Inventário

O arquivo de inventário do Ansible armazena uma lista de servidores gerenciados e acessados por ele. Por padrão, ele fica localizado em /etc/ansible/hosts.

Dentro do arquivo, podemos definir um host ou múltiplos, depois separá-los por grupos com o uso de "[]".

Gere o arquivo hosts utilizando o modelo ansible-hosts da pasta Aula 5.1:

- # cd /vagrant/4525/arquivos/Aula\ 5.1/
- # sudo cp ansible-hosts /etc/ansible/hosts
- # cat /etc/ansible/hosts
 [automation]
 automation.4labs.example

. . . .

1

automation.4labs.example

No arquivo de inventário, podemos especificar uma máquina colocando o hostname ou endereço ip, ou especificar um grupo utilizando as chaves.

Também podemos especificar variáveis pelo inventário e passar essas variáveis de acordo com o node. No exemplo abaixo, estamos passando a variável ansible_python_interpreter para apontar onde está localizado o interpretador do python, utilizado pelo ansible em cada servidor.

[ubuntu]

srv01.4labs.example ansible python interpreter=/usr/bin/python

[rhel]

srv02.4labs.example ansible_python_interpreter=/usr/bin/python3

srv03.4labs.example ansible_python_interpreter=/usr/bin/python3



===Ansible Hosts ===

[automation]

automation.4labs.example ansible_python_interpreter=/usr/bin/python

[compliance]

compliance.4labs.example ansible_python_interpreter=/usr/bin/python3

[container]

container.4labs.example ansible_python_interpreter=/usr/bin/python

[scm]

scm.4labs.example ansible_python_interpreter=/usr/bin/python

[log]

log.4labs.example ansible_python_interpreter=/usr/bin/python3

1

Instalação do Ansible

No arquivo ansible.cfg, podemos definir os parâmetros de configuração do ansible.

Edite o arquivo de configuração do ansible:

sudo vim /etc/ansible/ansible.cfg

roles_path = /etc/ansible/roles
host_key_checking = False
private_key_file = /root/.ssh/id_rsa

Podemos verificar agora a conexão, utilizando o módulo ping do ansible.

Execute o módulo ping do ansible para todo o inventário:
sudo ansible -m ping all

automation.4labs.example

O arquivo ansible.cfg é responsável por toda a configuração do ansible, nele definimos qual será a nossa chave de acesso privada a ser utilizada nas conexões via SSH para os nós. Definimos também que não será feita a checagem do fingerprint.

O comando **ansible -m ping all** faz com que o programa (ansible) utilize o módulo (-m) ping nos hosts (all).

O módulo ping do ansible executa a conexão SSH e retorna, caso a conexão seja realizada com sucesso ou não.

4LINUX Rec	apitulando		10
1 O que é	é o Ansible		
2 Caracte	erísticas		
3 Arquite	tura		
4 Instalaç	ão do Ansible		

Anotações	





Automação para ambientes ágeis com Ansible

Usando o Ansible Ad-hoc

Anotações	

4LINUX	Objetivos da aula	2
1	Módulos	
2	Comandos	
3	Facts	

Anotações		

Módulos

4LINUX

Módulos são utilizados para descrever as atividades que o Ansible executará na máquina de destino. Alguns desses módulos são:



Ansible possui milhares de módulos para as mais variáveis atividades, desde gerenciar os pacotes que serão instalados em uma máquina virtual, até o gerenciamento de conteiners. Alguns dos módulos mais usados no Ansible são:

apt: Gerencia a instalação de pacotes, utilizando o APTITUDE (debian/ubuntu);

yum: Gerencia a instalação de pacotes, utilizando o YUM (RedHat/CentOS/Fedora);

package: Gerencia a instalação de pacotes, utilizando o APTITUDE/YUM;

command: Executa um comando em node remoto;

shell: Executa um shell script dentro da máquina, após realizar as transferências;

service: Gerencia serviços em máquina remotas;

copy: Copia arquivo na máquina local para o node remoto;

git: Gerencia repositório do git;

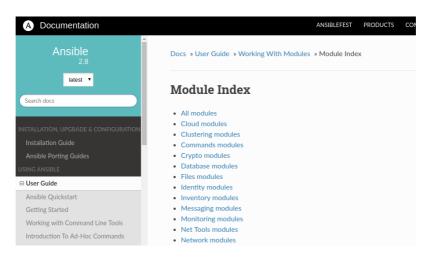
unarchive: Descompacta arquivos na máquina remota;mount: Gerencia os dispositivos montados na máquina;

template: Gerencia templates no ansible.

Módulos

4LINUX

A documentação de todos os módulos disponíveis do Ansible e seus parâmetros, podem ser visualizados em seu website https://docs.ansible.com/



A documentação do Ansible é completa, pesquisando por módulos temos uma lista de informações com todos seus parâmetros possíveis, bem como exemplos de playbooks (assunto que será visto posteriormente).

ansible-doc – Ferramenta de documentação de plugins do ansible.

- -I → Lista os plugins disponíveis;
- -s → Visualiza parte da documentação.

Comandos

Para executar um comando no ansible, precisamos informar qual módulo será utilizado, quais são os parâmetros, se houver, e quais máquinas no inventário receberão o comando.

Por padrão, o Ansible lê o arquivo de inventário localizado em /etc/ansible/hosts, porém é possível que seja informado outro arquivo de inventário, passando o parâmetro -i seguido do arquivo de inventário a ser utilizado.

Ex.:

ansible srv01 -m ping -i /files/inventario

4LINUX Comandos
Vamos realizar algumas tarefas rotineiras, utilizando o ansible.
<pre>1 # ansiblehelp</pre>
Leia o conteúdo de um arquivo no servidor compliance: # sudo ansible compliance -m shell -a "cat /etc/hosts"
Transfira um arquivo para o servidor log: # sudo ansible log -m copy -a "src=/etc/hosts dest=/tmp"
automation.4labs.example
Anotações

	Comandos Instale um aplicativo no servidor compliance:
1	<pre># sudo ansible compliance -m apt -a "name=cmatrix state=present update_cache=yes"</pre>
2	Reinicie um serviço no servidor automation: # sudo ansible automation -m service -a "name=crond state=restar" enabled=yes"
	automation.4labs.exan
Anota	ções



Facts

Facts são informações coletadas pelo ansible, quando o mesmo é executado em uma máquina remota.

Através dele podemos coletar diversas informações importantes, tais como armazenamento, configurações de redes, configuração do sistema, versão do sistema operacional, dentre outras.

Anotações		

<u>4</u> I	INUX Facts
	Para visualizar os dados trazidos pelo facts, podemos utilizar o módulo setup .
1	Verifique os facts do servidor log: # sudo ansible log -m setup
2	Colete os facts de todos os servidores: # sudo ansible all -m setup
3	Filtre uma seleção de facts: # sudo ansible all -m setup -a "filter=ansible_distribution"
	automation.4labs.example
Anota	ções

4LINUX	Recapitulando	11
1	Módulos	
2	Comandos	
3	Facts	

Anotações		





Automação para ambientes ágeis com Ansible

Introdução à Linguagem YAML

Anotações	

4LINU2	Objetivos da aula	2
1	O que é YAML	
2	Características	
3	Preparando o VIM	
4	Verificando a Sintaxe	

Anotações		

O que é YAML

O YAML (YAML Ain't Markup Language) é um formato de serialização de dados legível por humanos para todas as linguagens de programação.



É amplamente utilizada em ambientes DevOps, devido a sua facilidade de interpretação.

YAML é uma linguagem simples de ler e escrever, o ansible utiliza o formato nos arquivos de configuração dos Playbooks e Roles.

Exemplo de arquivo em YAML:

- - -

Funcionário da Dexter

- name: João Devops da Silva

job: Analista DevOps

skills:

- Administração de Sistema GNU/Linux
- Automação com Ansible
- Gerencia de Configuração com Puppet
- Testes de Infraestrutura com Inspec
- Versionamento com Git e Gogs

Características

- ✔ A estrutura do documento é composto por identação com espaços em branco, não é permitido o uso de caracteres de tabulação (TAB) para a identação;
- Comentários começam com o caractere cerquilha (#) e continuam até o final da linha;
- ✓ Os valores simples não levam aspas, mas podem ser incluídas aspas duplas (") ou aspas simples (').

Anotações	

Preparando o VIM

Podemos utilizar alguns recursos no VIM que nos ajudarão a escrever nossos arquivos no formato YAML, para isso vamos adicionar alguns parâmetros no arquivo

vimrc:

1

2

Acesse a pasta infraagil e conecte-se a máquina automation, utilizando o vagrant ssh:

- # cd infraagil
- # vagrant ssh automation

Edite o arquivo vimrc e adicione o conteúdo no final do arquivo:

- # sudo vim /etc/vimrc +\$
- " Recursos para o YAML
 set cursorline
 set cursorcolumn
 set number

automation.4labs.example

vimrc → Arquivo de parametrização do vim;

cursorline → Exibe uma linha, onde o cursor estiver posicionado;

cursor column → Exibe uma coluna, onde o cursor estiver posicionado;

number → Exibe o número das linhas.



Verificando a Sintaxe

Existe alguns recursos na internet, onde podemos digitar/colar o conteúdo do nosso arquivo YAML para que o mesmo seja validado. Isso nos ajuda inicialmente, a corrigir pequenos erros em nosso YAML.

YAML Lint

Paste in your YAML and click "Go" - we'll tell you if it's valid or it. Optimized for Ruby.

```
# Funcionário da Dexter

- name: João Devops da Silva
job: Analista DevOps
skills:
- Administração de Sistema GNU/Linux
- Automação com Ansible
- Gerencia de Configuração com Puppet
- Testes de Infraestrutura com Inspec
- Versionamento com Git e Gogs
```

http://www.yamllint.com/



https://jsonformatter.org/yaml-validator

notações	

4LINU2	Recapitulando	7
1	O que é YAML	
2	Características	
3	Preparando o VIM	
4	Verificando a Sintaxe	

Anotações	





Automação para ambientes ágeis com Ansible

Execuções em Playbooks

Anotações	

Anotações	

4LINUX O que são Playbooks

Playbooks são arquivos do Ansible em que você descreve as tarefas a serem executadas nos servidores.

Em um nível mais básico, as Playbooks podem ser usadas para gerenciar as configurações e realizar deploys nas máquinas remotas.

Anotações		



Escrevendo uma Playbook

Antes de escrever nossa primeira Playbook, vamos criar um diretório para organizar, guardar nossas playbooks e os exemplos.

- Alterne para o root e crie a estrutura de diretórios para testar as playbooks do ansible:
 - # sudo su -
 - # mkdir -p /etc/ansible/{playbooks,exemplos}
- Entre na pasta exemplos:
 - # cd /etc/ansible/exemplos

Anotações	



Escrevendo uma Playbook

Nossa primeira playbook será responsável por instalar e transferir o arquivo **resolv.conf** para todas as máquinas.

Gere o arquivo resolv.conf utilizando o modelo da pasta Aula 5.4:
cp /vagrant/4525/arquivos/Aula\ 5.4/resolv.conf .
cat resolv.conf

Atualizado via Ansible
nameserver 8.8.8.8

nameserver 8.8.4.4 search 4labs.example domain 4labs.example

Anotações		



Escrevendo uma Playbook

Vamos criar a nossa playbook.

Gere o arquivo primeiro.yml utilizando o modelo da pasta Aula 5.4:

- # cp /vagrant/4525/arquivos/Aula\ 5.4/primeiro.yml .
- # cat primeiro.yml
- # Minha primeira Playbook
- name: Minha primeira playbook

hosts: all tasks:

- name: Atualizar o resolv.conf

copy:

src: resolv.conf

dest: /etc/resolv.conf

Anotações	

<u>4</u> I	Escrevendo uma Playbook
	Agora, vamos executar nossa playbook:
1	Execute a playbook: # ansible-playbook primeiro.yml
	Utilizaremos o ansible ad-hoc para verificar a modificação do arquivo.
2	Utilize o módulo shell para verificar o arquivo modificado: # ansible all -m shell -a "cat /etc/resolv.conf"
Anota	ções

Anotações	

Trabalhando com Variáveis

Com variáveis, podemos permitir que nossas playbooks se comportem de forma diferente, dependendo do valor que for armazenado dentro dela.

Ao criar uma variável, você está permitindo que o usuário gerencie qual será a informação daquele campo, permitindo melhor reusabilidade da playbook.

Anotações	



Trabalhando com Variáveis

A maneira mais simples de declarar variáveis, é através de uma seção **vars** dentro da nossa playbook com os nomes e valores das variáveis.

vars:

- server_name: servidor01
- username: analistadevops
- conf_file: /opt/app/app.conf

Anotações	



1

Trabalhando com Variáveis

Vamos criar uma playbook para criação de usuários, utilizando variáveis.

Gere o arquivo users.yml utilizando o modelo da pasta Aula 5.4:

cp /vagrant/4525/arquivos/Aula\ 5.4/users.yml .

cat users.yml

Playbook para criar usuarios

- name: Criacao de usuarios nos servidores
hosts: all
vars:

- username: sysadmin

tasks:

- name: Criando os usuarios nos servidores

user:

name: "{{ username }}"

shell: /bin/bash

Anotações		

Agora, vamos executar nossa playbook com a variável. Execute a playbook:
Execute a playbook:
ansible-playbook users.yml
Utilizaremos o ansible ad-hoc para verificar a criação do usuário.
Utilize o módulo shell para verificar o usuário no passwd: # ansible all -m shell -a "getent passwd sysadmin"
bes des des des des des des des des des d

Trabalhando com Variáveis

A grande vantagem de se utilizar variáveis, é que podemos declará-las de forma externa à playbook.

- Execute a playbook:
 - # ansible-playbook users.yml -e username=developer

Utilizaremos o ansible ad-hoc para verificar a criação do usuário.

Utilize o módulo shell para verificar o usuário no passwd:
ansible all -m shell -a "getent passwd developer"

Anotações		

Trabalhando com Variáveis

Outra maneira de utilizar variáveis, é através de um arquivo externo sendo declarado na seção **vars_files**:

vars_files:

- config.yml

config.yml

server_name: servidor01
username: analistadevops

conf_file: /opt/app/app.conf

Anotações		

1

Trabalhando com Variáveis

Vamos criar o arquivo variaveis.yml.

Crie o arquivo:

vim variaveis.yml
username: analista

Gere o arquivo **users.yml** utilizando o modelo **users2.yml** da pasta Aula 5.4:

cp /vagrant/4525/arquivos/Aula\ 5.4/users2.yml users.yml

cat users.yml

2 - nam

- name: Criacao de usuarios nos servidores

hosts: all
vars_files:

- variaveis.yml

. . .

notações	

Trabalhando com Variáveis

Agora, vamos executar nossa playbook com o parâmetro vars_files.

1 Execute a playbook:

ansible-playbook users.yml

Utilizaremos o ansible ad-hoc para verificar a criação do usuário.

Utilize o módulo shell para verificar o usuário no passwd:
ansible all -m shell -a "getent passwd analista"

Anotações		

4LINUX Laços de Repetição

Laços de Repetição são estruturas criadas para que possamos executar uma mesma ação repetidas vezes, assim permitindo o reuso de código.

O Ansible permite que criemos laços de repetição, utilizando uma variável que receberá uma lista ou lista de loop, permitindo assim executar a mesma ação de acordo com a quantidade de itens que ela possui.

Anotações		

4LINUX Laços de Repetição

Vamos criar uma playbook para a instalação de vários programas:

Gere o arquivo loop.yml utilizando o modelo da pasta Aula 5.4:

cp /vagrant/4525/arquivos/Aula\ 5.4/loop.yml . # cat loop.yml name: Instalando uma lista de pacotes na Automation hosts: automation vars:

1

- packages:
 - wget
 - vim
 - tree

tasks:

- name: Instalando pacotes yum:

name: "{{ packages }}"

Anotações		

<u>4ī.</u>	INUX Laços de Repetição
	Agora, vamos executar nossa playbook com o loop e variáveis:
1	Execute a playbook: # ansible-playbook loop.yml
	Para verificar se os programas foram instalados, podemos listar os pacotes rpm:
2	Verifique se os pacotes foram instalados: # rpm -qa egrep "epel wget vim"
notac	cões

1

Laços de Repetição

Vamos criar uma segunda playbook, utilizando loops em variáveis:

```
Gere o arquivo loop2.yml utilizando o modelo da pasta Aula 5.4:
# cp /vagrant/4525/arquivos/Aula\ 5.4/loop2.yml .
# cat loop2.yml
- name: Instalando uma lista de pacotes nos servidores Ubuntu hosts: compliance log tasks:
- name: Instalando pacotes apt:
    name: "{{ item }}"
    update_cache: yes loop:
    - sl
    - cmatrix
    ncdu
```

automation.4labs.example

loop: Cria uma variável temporária chamada **item**, onde serão armazenados todos os valores que você passou para dentro do loop.

Vale lembrar que essa variável só existirá enquanto estiver executando aquele módulo. Após a conclusão, os valores serão perdidos. Caso necessite de uma lista reutilizável, use variáveis no lugar.

Em versões mais antigas do Ansible (anterior a 2.5), o parâmetro a ser utilizado era o **with_items** ao invés do **loop**, porém o mesmo foi descontinuado.

2

Anotações

Laços de Repetição

Agora, vamos executar nossa playbook com o loop:

- Execute a playbook:
 - # ansible-playbook loop2.yml

Para verificar se os programas foram instalados, podemos listar os pacotes com dpkg.

Verifique se os pacotes foram instalados:

- # ansible compliance -m shell -a "dpkg -l | egrep 'sl|cmatrix|ncdu'"
 - # ansible log -m shell -a "dpkg -l | egrep 'sl|cmatrix|ncdu'"

4LINUX Condicionais

Existem casos em que precisamos que um módulo só execute se uma determinada condição for verdadeira, como por exemplo instalar o epel-release apenas se o sistema operacional for um CentOS. Para isso, utilizamos Condicionais que servem para testar se uma determinada condição é verdadeira e pode executar uma ação.

O Ansible utiliza o parâmetro **when** para efetuar esse teste.

Anotações	

4LINUX Laços de Repetição

Criando uma playbook para instalação de programas, caso o sistema operacional seja Ubuntu.

```
Gere o arquivo when.yml utilizando o modelo da pasta Aula 5.4:
# cp /vagrant/4525/arquivos/Aula\ 5.4/when.yml .
# cat when.yml
- name: Instalando uma lista de pacotes nos servidores Ubuntu
  hosts: all
  tasks:

    name: Instalando pacotes

    apt:
      name: "{{ item }}"
    loop:
      - cmatrix
      - sl
      - ncdu
    when: ansible_distribution == "Ubuntu"
```

Anotações		



Condicional

Agora, vamos executar nossa playbook condicional.

Execute a playbook:

ansible-playbook when.yml

Note que ao executar a playbook, é feita uma varredura no inventário e os hosts que não trazem a condicional tratada no **when** como verdadeira, não executam a playbook e apresentam como **skipping** (pulando) no terminal.

```
TASK [Instalando pacotes] **********************
skipping: [scm.4labs.example] => (item=cmatrix)
          [scm.4labs.example] => (item=sl)
skipping:
skipping:
          [scm.4labs.example] => (item=ncdu)
skipping:
          [container.4labs.example] => (item=cmatrix)
skipping:
          [container.4labs.example] => (item=sl)
skipping: [container.4labs.example] => (item=ncdu)
          [automation.4labs.example] => (item=cmatrix)
skipping:
skipping: [automation.4labs.example] => (item=sl)
skipping: [automation.4labs.example] => (item=ncdu)
    [log.4labs.example] => (item=cmatrix)
[compliance.4labs.example] => (item=cmatrix)
    [log.4labs.example] => (item=sl)
    [compliance.4labs.example] => (item=sl)
    [log.4labs.example] => (item=ncdu)
```

Anotações

4LINUX Recapitulando	24
1 O que são Playbooks	
2 Escrevendo Playbooks	
3 Trabalhando com Variáveis	
4 Laços de Repetição	
5 Estrutura de Condicionais	

Anotações	