# Aula 21 – Variáveis, Fatos e Templates com Jinja2

Imagine a rotina de configurar servidores. No início, talvez um ou dois, tudo feito manualmente. Mas e quando a demanda cresce, e você precisa gerenciar dezenas, centenas, ou até milhares de máquinas, cada uma com suas particularidades? A repetição de tarefas se torna um pesadelo, e um pequeno erro pode derrubar sistemas inteiros. É nesse cenário que a Infraestrutura como Código (IaC) brilha, transformando a maneira como interagimos com nossos ambientes.

No entanto, mesmo com a IaC, se cada configuração for estática e precisar de um arquivo diferente para cada pequena variação, acabamos voltando à complexidade. A verdadeira magia da automação reside na capacidade de criar configurações dinâmicas, que se adaptam a diferentes contextos sem exigir uma reescrita constante. É aqui que entram as variáveis, os fatos e os motores de template como o Jinja2. Eles são a chave para transformar playbooks estáticos em orquestrações inteligentes e flexíveis.

Nesta aula, vamos mergulhar fundo nesses conceitos essenciais. Você aprenderá a definir e gerenciar variáveis em diferentes níveis, a coletar informações valiosas dos seus servidores através dos fatos do Ansible, e a utilizar o poderoso motor de templates Jinja2 para construir arquivos de configuração que se moldam às necessidades de cada ambiente. Ao final, você será capaz de criar automações mais robustas, escaláveis e fáceis de manter, um passo fundamental para qualquer profissional de infraestrutura moderna.

Nosso percurso começará com a exploração das variáveis, passando pela inteligência dos fatos do Ansible, até culminar na arte de templating com Jinja2, tudo isso ilustrado com um exemplo prático de configuração de um virtual host do Nginx. Prepare-se para elevar o nível da sua automação!

# A Necessidade de Flexibilidade na Automação

**Por Que Variáveis?**

Variáveis são espaços reservados que você define em seu código e preenche com valores específicos no momento da execução.

Pense na sua rotina: você tem um conjunto de tarefas que precisa realizar, mas cada vez que as executa, há pequenos detalhes que mudam. Talvez o nome de um servidor, a porta de um serviço, ou o caminho de um arquivo. Se você tivesse que reescrever todo o script ou playbook para cada uma dessas pequenas alterações, a automação perderia grande parte de seu valor. A rigidez se tornaria um obstáculo, não uma solução.

É exatamente essa a dor que as variáveis vêm para curar. Elas são como espaços reservados, ou "placeholders", que você define em seu código e preenche com valores específicos no momento da execução. Em vez de ter um playbook para "servidor-web-producao" e outro para "servidor-web-desenvolvimento", você pode ter um único playbook que usa uma variável `ambiente` e se adapta conforme o valor que você fornece. Isso não apenas economiza tempo, mas também reduz drasticamente a chance de erros, pois a lógica central permanece a mesma.

**Reutilização**

Um único playbook serve múltiplos cenários

**Redução de Erros**

Lógica central consistente e testada

**Manutenção Simplificada**

Alterações em um único lugar

No universo do Ansible, as variáveis são a espinha dorsal da automação dinâmica. Elas permitem que você crie playbooks genéricos que podem ser reutilizados em diversos cenários, adaptando-se a diferentes hosts, grupos de hosts ou até mesmo a ambientes inteiros. Essa capacidade de abstração é o que transforma a Infraestrutura como Código de uma simples replicação de comandos em uma ferramenta poderosa para gerenciar complexidade.

# Desvendando as Variáveis no Ansible

## Onde Elas Moram?

Compreender o que são variáveis é o primeiro passo, mas saber onde e como defini-las no Ansible é crucial para construir automações eficientes. O Ansible oferece uma hierarquia rica e flexível para a definição de variáveis, permitindo que você as declare em diferentes níveis, desde o playbook individual até arquivos externos compartilhados por toda a sua infraestrutura. Essa flexibilidade é uma faca de dois gumes: poderosa quando bem utilizada, confusa se não entendida.

01

### Playbook

Variáveis específicas definidas diretamente no playbook

02

### Inventário

Variáveis em group_vars e host_vars para grupos e hosts

03

### Arquivos Externos

Carregamento via vars_files para compartilhamento

04

### Linha de Comando

Passagem direta com --extra-vars para sobrescrita

A forma mais simples de definir uma variável é diretamente dentro de um playbook. Isso é útil para valores que são específicos daquele playbook e não precisam ser compartilhados. No entanto, para configurações que se aplicam a grupos de servidores ou a servidores individuais, o inventário se torna o local ideal. Aqui, você pode ter variáveis que se aplicam a todos os hosts de um grupo (group_vars) ou a um host específico (host_vars), garantindo que cada máquina receba a configuração exata de que precisa.

Além disso, o Ansible permite carregar variáveis de arquivos externos (vars_files) ou passá-las diretamente pela linha de comando (--extra-vars). Essa diversidade de fontes é o que confere ao Ansible sua capacidade de adaptação a cenários complexos, desde um pequeno projeto até grandes ambientes corporativos. A chave é escolher o local certo para cada variável, pensando na sua abrangência e na sua sensibilidade.

```
# Exemplo de variável definida em um playbook
---
- name: Configurar servidor web
 hosts: webservers
 vars:
 http_port: 80
 max_clients: 200
 tasks:
 - name: Instalar Nginx
 ansible.builtin.apt:
 name: nginx
 state: present

 - name: Configurar porta HTTP
 ansible.builtin.lineinfile:
 path: /etc/nginx/nginx.conf
 regexp: '^listen\s+80;'
 line: "listen {{ http_port }};"
 notify: Restart Nginx
```

# Variáveis de Inventário

## Organizando sua Infraestrutura

Quando você começa a gerenciar mais do que alguns servidores, a ideia de definir variáveis diretamente em cada playbook se torna impraticável. É como tentar organizar uma biblioteca inteira colocando etiquetas em cada livro individualmente, em vez de categorizá-los por gênero ou autor. As variáveis de inventário do Ansible oferecem uma solução elegante para essa complexidade, permitindo que você estruture suas configurações de forma lógica e escalável.

### group_vars/

**Escopo:** Todos os membros de um grupo

**Uso:** Configurações compartilhadas

**Exemplo:** Versão do Nginx, diretório raiz padrão

- Garante consistência
- Facilita manutenção
- Uma mudança afeta todo o grupo

### host_vars/

**Escopo:** Host específico

**Uso:** Configurações exclusivas

**Exemplo:** Porta customizada, certificado SSL único

- Permite exceções
- Granularidade máxima
- Não quebra a regra geral

As variáveis de inventário são armazenadas em arquivos YAML dentro de diretórios especiais: group_vars e host_vars. O diretório group_vars contém arquivos com variáveis que se aplicam a todos os membros de um grupo específico definido no seu inventário. Por exemplo, você pode ter um arquivo group_vars/webservers.yml com variáveis como `nginx_version` ou `document_root`, que serão aplicadas a todos os servidores web. Isso garante consistência e facilita a manutenção, pois uma mudança em um único arquivo afeta todos os servidores daquele grupo.

Por outro lado, o diretório host_vars é usado para variáveis que são exclusivas de um host específico. Se um servidor web01.example.com tem uma configuração de porta diferente ou um certificado SSL único, você pode definir essas variáveis em host_vars/web01.example.com.yml. Essa granularidade permite que você lide com as exceções sem quebrar a regra geral definida nos group_vars. Essa abordagem, onde as configurações são versionadas junto com o código, é um pilar fundamental da metodologia **GitOps**, onde o repositório Git se torna a "fonte única da verdade" para o estado da sua infraestrutura.

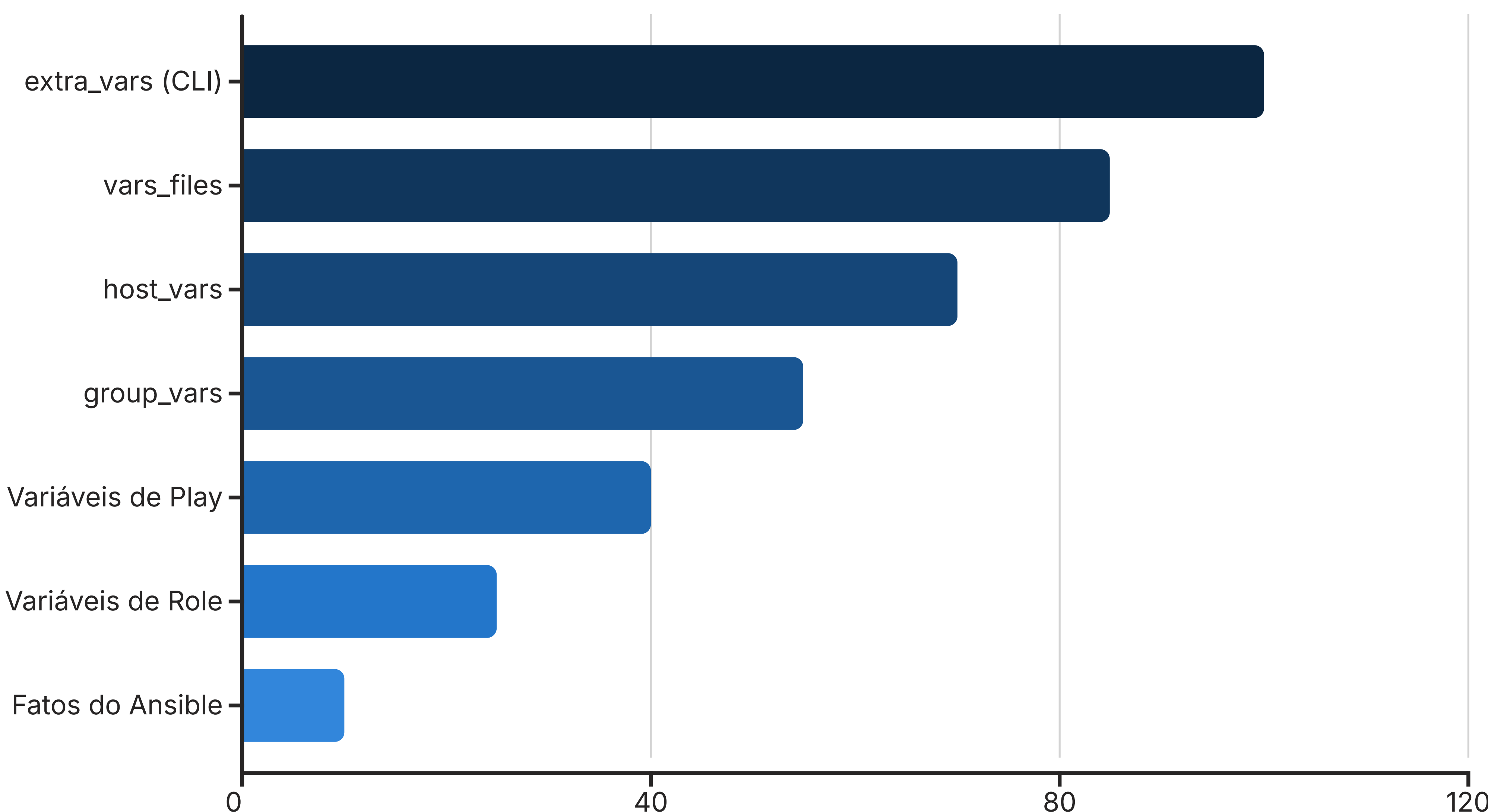
```
# Exemplo de group_vars/webservers.yml
---
nginx_port: 8080
nginx_root_dir: /var/www/html/default
nginx_log_level: info

# Exemplo de host_vars/web01.example.com.yml
---
nginx_port: 80
nginx_root_dir: /var/www/html/web01
```

# Variáveis Externas e Precedência

## A Ordem dos Fatores

A flexibilidade do Ansible em aceitar variáveis de diversas fontes é uma de suas maiores forças, mas também pode ser uma fonte de confusão se a ordem de precedência não for clara. Imagine que você tem uma variável `http_port` definida em um group_vars, em um host_vars, e também passada pela linha de comando. Qual valor o Ansible usará? Entender a "ordem dos fatores" é crucial para evitar surpresas e garantir que suas automações se comportem como esperado.

O Ansible resolve conflitos de variáveis seguindo uma hierarquia bem definida, onde algumas fontes têm prioridade sobre outras. Em termos gerais, variáveis mais específicas (como as definidas para um host individual) geralmente sobrescrevem variáveis mais genéricas (como as de grupo). Além disso, variáveis passadas na linha de comando (--extra-vars) ou definidas em arquivos específicos (vars_files) tendem a ter uma precedência muito alta, permitindo que você sobrescreva temporariamente valores para testes ou situações emergenciais.




Essa hierarquia é essencial para a robustez das suas automações. Ela permite que você estabeleça padrões em um nível mais alto (por exemplo, em group_vars) e, em seguida, faça ajustes finos em níveis mais baixos (em host_vars ou até mesmo em um playbook específico) sem precisar reescrever grandes blocos de código. Dominar essa precedência é um sinal de maturidade na utilização do Ansible, garantindo que você tenha controle total sobre o comportamento da sua infraestrutura.

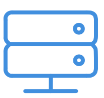
### Dica Prática

Use `ansible.builtin.debug` para imprimir valores de variáveis e entender qual fonte está sendo usada em caso de dúvida.

# Fatos do Ansible

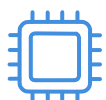
# O Conhecimento é Poder

Imagine que você é um médico e precisa diagnosticar um paciente. Você não começaria a prescrever medicamentos sem antes coletar informações vitais: temperatura, pressão arterial, histórico médico. Da mesma forma, para automatizar a configuração de um servidor de forma inteligente, você precisa conhecer o "estado de saúde" e as características desse servidor. É aqui que entram os "fatos" do Ansible.



### Sistema Operacional

Distribuição, versão, família (Debian, RedHat, etc.)



### Hardware

Memória RAM, número de CPUs, arquitetura do processador

### Rede

Endereços IP, interfaces de rede, gateway padrão

### Armazenamento

Discos disponíveis, pontos de montagem, espaço livre

Os fatos do Ansible, ou `ansible_facts`, são informações detalhadas que o Ansible coleta automaticamente sobre os managed nodes (os servidores que ele gerencia). Isso inclui dados sobre o sistema operacional (distribuição, versão), hardware (memória, CPUs), rede (endereços IP, interfaces), disco, e muito mais. Essa coleta de informações ocorre por padrão no início de cada execução de playbook, a menos que você a desabilite explicitamente.

Essa capacidade de autodescoberta é incrivelmente poderosa. Em vez de codificar manualmente o sistema operacional de cada servidor em suas variáveis, você pode simplesmente usar `ansible_facts['os_family']` para adaptar sua configuração se o servidor for baseado em Debian ou RedHat. Isso torna seus playbooks muito mais flexíveis e resilientes a mudanças no ambiente. É como ter um assistente que sempre sabe tudo sobre cada máquina antes de você começar a trabalhar nela.

# Explorando os ansible_facts

## Um Tesouro de Informações

Os ansible_facts são um verdadeiro tesouro de informações que podem ser acessadas e utilizadas em seus playbooks e templates Jinja2. Eles fornecem uma visão abrangente do ambiente de cada managed node, permitindo que você tome decisões de automação baseadas em dados reais e atualizados. Sem essa capacidade, a automação seria muito mais rígida e propensa a erros, exigindo que você mantivesse um inventário manual de características de cada servidor.

Para ter uma ideia da riqueza desses dados, você pode executar um playbook simples com o módulo `setup` ou o módulo `debug` para inspecionar todos os fatos coletados de um host. Você verá informações como `ansible_default_ipv4` (detalhes da interface de rede padrão), `ansible_memtotal_mb` (memória total em megabytes), `ansible_distribution` (nome da distribuição Linux), `ansible_processor_vcpus` (número de CPUs virtuais), entre muitos outros.

**ansible_hostname**

Nome do host do servidor

**ansible_distribution**

Distribuição Linux (Ubuntu, CentOS, etc.)

**ansible_default_ipv4**

Informações da interface de rede padrão

**ansible_memtotal_mb**

Memória total em megabytes

**ansible_processor_vcpus**

Número de CPUs virtuais

**ansible_mounts**

Pontos de montagem de disco

A utilização desses fatos é fundamental para criar playbooks verdadeiramente inteligentes. Por exemplo, você pode condicionar a instalação de um pacote específico ao sistema operacional (`when: ansible_facts['os_family'] == 'Debian'`). Ou, pode configurar um firewall para abrir uma porta apenas se o servidor tiver um determinado endereço IP. Essa capacidade de adaptação automática é um dos pilares da automação moderna e é um componente chave para a implementação de estratégias de **AIOps**, onde a inteligência artificial pode usar esses dados para otimizar operações e prever falhas.

```
# Exemplo de playbook para exibir fatos de um host
---
- name: Exibir fatos do servidor
 hosts: webservers
 tasks:
 - name: Coletar fatos (já é padrão, mas pode ser forçado)
 ansible.builtin.setup:

 - name: Imprimir alguns fatos importantes
 ansible.builtin.debug:
 msg: |
 Nome do Host: {{ ansible_facts['hostname'] }}
 Sistema Operacional: {{ ansible_facts['distribution'] }} {{ ansible_facts['distribution_version'] }}
 Endereço IP Padrão: {{ ansible_facts['default_ipv4']['address'] }}
 Memória Total (MB): {{ ansible_facts['memtotal_mb'] }}
```

# Quando os Fatos Não São Suficientes

## Custom Facts

Embora os ansible_facts forneçam uma vasta quantidade de informações sobre seus managed nodes, há momentos em que você precisa de dados muito específicos que não são coletados por padrão. Talvez você tenha uma aplicação legada que armazena sua versão em um arquivo customizado, ou precise de uma métrica de hardware muito particular que não está nos fatos padrão. Nesses casos, o Ansible oferece uma solução elegante: os "custom facts".

Custom facts permitem que você estenda a capacidade de coleta de informações do Ansible, adicionando seus próprios scripts ou programas que retornam dados no formato JSON. Esses scripts são executados nos managed nodes e seus resultados são incorporados aos ansible_facts padrão, tornando-os acessíveis da mesma forma que qualquer outro fato. É como ensinar seu assistente a coletar informações adicionais que são importantes para o seu trabalho específico.

**Localização**

Scripts devem estar em `/etc/ansible/facts.d/` no managed node

Para criar um custom fact, basta colocar um script executável (que pode ser em Python, Bash, ou qualquer outra linguagem) em um diretório específico no managed node, geralmente `/etc/ansible/facts.d/`. O script deve retornar uma saída JSON para o stdout. Na próxima execução do Ansible, esses dados serão coletados e estarão disponíveis sob a chave `ansible_local`. Essa funcionalidade é extremamente útil para integrar informações específicas da sua aplicação ou ambiente, garantindo que seus playbooks tenham todas as informações necessárias para tomar decisões inteligentes e personalizadas.

```
# Exemplo de um script de custom fact (meu_app_version.fact)
# Salvar em /etc/ansible/facts.d/meu_app_version.fact no managed node
#!/bin/bash
APP_VERSION="1.2.3"
echo "{\"meu_app\": {\"version\": \"$APP_VERSION\", \"status\": \"running\"}}"
```

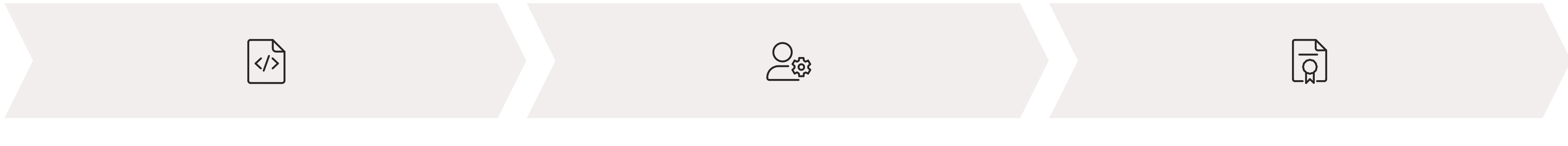
Depois de colocar este arquivo no servidor, ao executar um playbook, você poderá acessar:

{{ ansible_facts['ansible_local']['meu_app_version']['meu_app']['version'] }}

# Introdução ao Jinja2

# O Coração Dinâmico da Configuração

Até agora, aprendemos a definir variáveis e a coletar fatos. Mas como usamos essas informações para realmente criar arquivos de configuração que se adaptam dinamicamente? Inserir `{{ variavel }}` diretamente em um arquivo de texto não é suficiente; precisamos de um motor que interprete esses "placeholders" e gere o conteúdo final. É aqui que o Jinja2 entra em cena, atuando como o coração dinâmico da sua automação com Ansible.



**Template Jinja2**

Modelo com variáveis e lógica

**Motor Jinja2**

Processa e preenche valores

**Arquivo Final**

Configuração pronta para uso

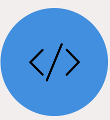
Jinja2 é um motor de templates moderno e poderoso para Python, e é a escolha padrão do Ansible para renderizar arquivos de configuração. Pense nele como um editor de texto inteligente que pode preencher formulários automaticamente. Você cria um "modelo" (o template Jinja2) com espaços para variáveis e lógica condicional, e o Jinja2 o preenche com os valores reais das suas variáveis e fatos, gerando um arquivo de configuração final e pronto para uso.

A beleza do Jinja2 reside em sua simplicidade e poder. Ele permite que você não apenas insira valores de variáveis, mas também execute lógica complexa dentro dos seus templates, como loops para criar várias entradas de configuração ou condicionais para incluir ou excluir blocos de texto. Essa capacidade de gerar conteúdo de forma programática é o que permite que um único template sirva para dezenas ou centenas de servidores, cada um com sua configuração personalizada.

# Sintaxe Essencial do Jinja2

## Para Ansible

Para começar a usar o Jinja2 de forma eficaz com o Ansible, é fundamental entender sua sintaxe básica. A boa notícia é que ela é bastante intuitiva e se baseia em apenas alguns marcadores principais. Dominar esses elementos permitirá que você crie templates complexos e flexíveis, transformando a maneira como você gerencia suas configurações.

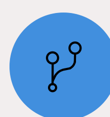


### {{ expressao }}

**Função:** Imprimir valores

Usado para exibir o valor de uma variável ou resultado de uma expressão. Exemplo: `{{ server_name }}`

Pode aplicar filtros: `{{ server_name | upper }}`



### {% declaracao %}

**Função:** Lógica de controle

Usado para loops (for) e condicionais (if/else). Exemplo: `{% if http_port is defined %}`

Permite construção dinâmica de conteúdo

### {# comentario #}

**Função:** Comentários

Ignorados pelo motor de template. Útil para documentação interna do template.

Não aparecem no arquivo final

A combinação desses elementos permite que você crie templates que são não apenas dinâmicos, mas também inteligentes, adaptando-se a diferentes cenários e dados de entrada. Essa flexibilidade é crucial para a segurança integrada (**DevSecOps**), permitindo que você template configurações de segurança, como regras de firewall ou permissões de arquivo, garantindo que elas sejam aplicadas de forma consistente em toda a sua infraestrutura.

```
# Exemplo de template Jinja2 (nginx.conf.j2)
server {
 listen {{ http_port }};
 server_name {{ server_name }};
 root {{ document_root }};

 {% if enable_ssl %}
 listen {{ https_port }} ssl;
 ssl_certificate /etc/nginx/ssl/{{ server_name }}.crt;
 ssl_certificate_key /etc/nginx/ssl/{{ server_name }}.key;
 {% endif %}

 location / {
 index index.html index.htm;
 }

 error_page 500 502 503 504 /50x.html;
 location = /50x.html {
 root /usr/share/nginx/html;
 }
}
```

# Criando Arquivos de Configuração Dinâmicos

## Com Jinja2

Agora que entendemos a sintaxe do Jinja2, o próximo passo é integrá-lo ao Ansible para gerar arquivos de configuração reais nos seus managed nodes. O módulo `ansible.builtin.template` é a ferramenta que faz essa ponte, pegando seu template Jinja2 (.j2) e as variáveis do Ansible, processando-os e, finalmente, copiando o arquivo resultante para o destino desejado no servidor remoto.

01

### Criar Template

Arquivo .j2 com variáveis e lógica Jinja2

02

### Definir Variáveis

Em playbook, inventário ou arquivos externos

03

### Usar Módulo Template

Especificar src (template) e dest (destino)

04

### Ansible Processa

Substitui variáveis e executa lógica

05

### Arquivo Copiado

Configuração final no servidor remoto

O processo é bastante direto: você cria um arquivo de template com a extensão .j2 (por exemplo, nginx.conf.j2) e o coloca em um diretório acessível ao seu playbook, geralmente dentro de uma estrutura de roles ou em um diretório templates/. Em seguida, no seu playbook, você usa o módulo template, especificando o caminho para o template de origem (src) e o caminho de destino no servidor remoto (dest).

Quando o Ansible executa essa tarefa, ele lê o template .j2, substitui todas as expressões `{{ ... }}` pelos valores das variáveis e fatos disponíveis, e executa toda a lógica `{% ... %}`. O resultado é um arquivo de texto puro, perfeitamente formatado e personalizado para aquele servidor específico, que é então copiado para o destino. Essa abordagem não só garante que cada servidor receba a configuração correta, mas também que essa configuração seja sempre consistente com o template original, facilitando auditorias e conformidade.

```
# Exemplo de uso do módulo template em um playbook
---
- name: Configurar Nginx virtual host
 hosts: webservers
 vars:
 http_port: 80
 server_name: example.com
 document_root: /var/www/html/example.com
 enable_ssl: true
 https_port: 443
 tasks:
 - name: Criar diretório para o site
 ansible.builtin.file:
 path: "{{ document_root }}"
 state: directory
 owner: www-data
 group: www-data
 mode: '0755'

 - name: Copiar template de virtual host do Nginx
 ansible.builtin.template:
 src: templates/nginx_vhost.conf.j2
 dest: /etc/nginx/sites-available/{{ server_name }}.conf
 owner: root
 group: root
 mode: '0644'
 notify: Restart Nginx

 - name: Habilitar virtual host
 ansible.builtin.file:
 src: /etc/nginx/sites-available/{{ server_name }}.conf
 dest: /etc/nginx/sites-enabled/{{ server_name }}.conf
 state: link
 notify: Restart Nginx
```

# Exemplo Prático: Configurando um Virtual Host do Nginx

## Parte 1: O Desafio

Vamos consolidar todo o conhecimento adquirido até agora com um exemplo prático e muito comum: a configuração de um virtual host para o servidor web Nginx. Imagine que você precisa configurar múltiplos sites em um único servidor, ou replicar a configuração de um site em vários servidores, mas cada um com seu próprio domínio, diretório raiz e talvez até portas diferentes. Fazer isso manualmente seria tedioso e propenso a erros.

### Objetivo do Exemplo

Criar um playbook Ansible que configure virtual hosts do Nginx de forma dinâmica, usando variáveis, fatos e templates Jinja2.

Nosso desafio é criar um playbook Ansible que configure um virtual host do Nginx de forma dinâmica. Para isso, precisaremos de variáveis para definir os parâmetros específicos de cada site, como o nome do domínio (`server_name`), o diretório onde os arquivos do site estarão (`document_root`), e a porta HTTP que o Nginx deve escutar. Essas variáveis podem ser definidas em group_vars (se aplicarem a um grupo de servidores web) ou host_vars (se forem específicas de um servidor).

**1**

**Template Único**

Um arquivo .j2 para todos os sites

**∞**

**Sites Ilimitados**

Adicione quantos quiser via variáveis

**0**

**Erros Manuais**

Configuração consistente e testada

A beleza dessa abordagem é que, uma vez que o template Jinja2 e o playbook Ansible estejam prontos, você pode adicionar novos sites ou configurar servidores adicionais com apenas algumas linhas de configuração no seu inventário, sem tocar no código do playbook. Isso é a essência da escalabilidade e da manutenção simplificada que a Infraestrutura como Código promete.

# Exemplo Prático: Configurando um Virtual Host do Nginx

## Parte 2: Definindo as Variáveis

Para o nosso exemplo do Nginx, vamos definir algumas variáveis essenciais que serão usadas para personalizar o virtual host. A ideia é que esses valores possam ser facilmente alterados para cada site ou servidor, sem modificar o template principal. Vamos considerar um cenário onde temos um grupo de servidores web e queremos configurar diferentes virtual hosts neles.

Podemos definir variáveis padrão em `group_vars/webservers.yml` que se aplicariam a todos os servidores no grupo webservers. Por exemplo, a porta HTTP padrão, ou um diretório raiz comum. No entanto, para cada site individual, precisaremos de variáveis mais específicas. Uma boa prática é criar uma lista de sites dentro das variáveis de um host ou grupo, onde cada item da lista representa um virtual host a ser configurado.

### group_vars/webservers.yml

```
---
nginx_default_http_port: 80
nginx_default_root: /var/www/html
```

Variáveis compartilhadas por todos os servidores web

### host_vars/meu_servidor_web.yml

```
---
sites:
  - name: site1.com
    document_root: "{{ nginx_default_root }}/site1"
    port: "{{ nginx_default_http_port }}"
    ssl_enabled: true
  - name: site2.org
    document_root: "{{ nginx_default_root }}/site2"
    port: 8080
    ssl_enabled: false
```

Lista de sites específicos deste servidor

Neste exemplo, `sites` é uma lista de dicionários, onde cada dicionário descreve um virtual host. Isso nos permite iterar sobre essa lista no template Jinja2 para gerar múltiplas configurações de virtual host a partir de um único template. Essa abordagem é extremamente poderosa para gerenciar ambientes complexos e é um exemplo claro de como o **GitOps** pode ser aplicado, onde a definição de cada site é um dado versionado no repositório.

# Exemplo Prático: O Template Jinja2 para Nginx

Com as variáveis definidas, o próximo passo é criar o template Jinja2 que usará essas variáveis para gerar o arquivo de configuração final do Nginx. Este template será o "molde" que o Ansible preencherá. Vamos criar um arquivo chamado `nginx_vhost.conf.j2` dentro do diretório templates/ da sua role ou playbook.

A estrutura do template refletirá a configuração padrão de um virtual host do Nginx, mas com os valores dinâmicos substituídos por marcadores Jinja2 (`{{ ... }}`). Além disso, podemos usar a lógica condicional (`{% if ... %}`) para incluir ou excluir blocos de configuração, como a seção SSL, dependendo do valor de uma variável. Isso demonstra o poder do Jinja2 em criar configurações adaptáveis.

```
# templates/nginx_vhost.conf.j2
{% for site in sites %}
server {
    listen {{ site.port }};
    server_name {{ site.name }} www.{{ site.name }};
    root {{ site.document_root }};
    index index.html index.htm;

    access_log /var/log/nginx/{{ site.name }}_access.log;
    error_log /var/log/nginx/{{ site.name }}_error.log;

    {% if site.ssl_enabled %}
    listen 443 ssl;
    ssl_certificate /etc/nginx/ssl/{{ site.name }}.crt;
    ssl_certificate_key /etc/nginx/ssl/{{ site.name }}.key;
    {% endif %}

    location / {
        try_files $uri $uri/ =404;
    }

    # Adicione outras configurações específicas do site aqui
}
{% endfor %}
```

1

### Loop for

Itera sobre a lista de sites

2

### Variáveis

Substitui valores dinâmicos

3

### Condicional if

Inclui SSL se habilitado

4

### Múltiplos Blocos

Gera um server{} por site

Neste template, o loop `{% for site in sites %}` é crucial. Ele permite que o Ansible itere sobre a lista de sites definida em host_vars e gere um bloco `server {}` completo para cada um, tudo a partir de um único arquivo de template. Isso exemplifica a eficiência e a escalabilidade que o Jinja2 e o Ansible juntos proporcionam.

# Exemplo Prático: O Playbook Ansible para Nginx

Finalmente, precisamos do playbook Ansible que orquestrará todo o processo: garantindo que o Nginx esteja instalado, que os diretórios necessários existam, e que o template Jinja2 seja processado e copiado para o local correto. Este playbook será o ponto de entrada para a nossa automação.

O playbook usará o módulo `ansible.builtin.apt` (para sistemas Debian/Ubuntu) ou `ansible.builtin.yum` (para sistemas RedHat/CentOS) para instalar o Nginx. Em seguida, ele criará os diretórios raiz para cada site e, o mais importante, usará o módulo `ansible.builtin.template` para gerar e copiar os arquivos de configuração do virtual host. Note que o `notify: Restart Nginx` é usado para garantir que o Nginx seja reiniciado apenas uma vez, após todas as configurações serem aplicadas, otimizando o processo.

```
# playbook.yml
---
- name: Configurar Nginx com virtual hosts dinâmicos
  hosts: webservers
  become: true
  vars_files:
    - group_vars/webservers.yml
  tasks:
    - name: Instalar Nginx
      ansible.builtin.apt:
        name: nginx
        state: present
        update_cache: yes

    - name: Criar diretórios raiz para cada site
      ansible.builtin.file:
        path: "{{ item.document_root }}"
        state: directory
        owner: www-data
        group: www-data
        mode: '0755'
      loop: "{{ sites }}"

    - name: Copiar template de virtual host do Nginx
      ansible.builtin.template:
        src: templates/nginx_vhost.conf.j2
        dest: /etc/nginx/sites-available/{{ item.name }}.conf
        owner: root
        group: root
        mode: '0644'
      loop: "{{ sites }}"
      notify: Restart Nginx

    - name: Habilitar virtual hosts (criar links simbólicos)
      ansible.builtin.file:
        src: /etc/nginx/sites-available/{{ item.name }}.conf
        dest: /etc/nginx/sites-enabled/{{ item.name }}.conf
        state: link
      loop: "{{ sites }}"
      notify: Restart Nginx

  handlers:
    - name: Restart Nginx
      ansible.builtin.service:
        name: nginx
        state: restarted
```

Este playbook é um exemplo completo de como variáveis, fatos (implícitos na detecção do OS para apt) e templates Jinja2 trabalham em conjunto para automatizar a configuração de infraestrutura de forma dinâmica e escalável.

# Testando e Validando a Configuração Dinâmica

Após a execução do playbook, a etapa de validação é crucial para garantir que tudo foi configurado corretamente e que os serviços estão operacionais. A automação é poderosa, mas a confiança em sua execução vem da capacidade de verificar seus resultados. Ignorar esta etapa é como assar um bolo sem provar antes de servir: você pode ter uma surpresa desagradável.

### Verificar Serviço

Confirmar que o Nginx está rodando

### Inspecionar Arquivos

Validar conteúdo dos arquivos de configuração

### Teste Funcional

Acessar sites e verificar resposta HTTP

Para o nosso exemplo do Nginx, a validação pode envolver algumas verificações simples, mas eficazes. Primeiro, você pode verificar se o serviço Nginx está em execução no servidor remoto. O módulo `ansible.builtin.service` pode ser usado para isso, ou até mesmo um comando `systemctl status nginx` via `ansible.builtin.command`. Em seguida, é essencial verificar se os arquivos de configuração foram criados corretamente e se seus conteúdos estão de acordo com o esperado, usando `ansible.builtin.command: cat /etc/nginx/sites-available/site1.com.conf`.

A validação mais importante, no entanto, é a funcional. Isso significa tentar acessar os sites configurados. Você pode usar o módulo `ansible.builtin.uri` ou um comando `curl` para verificar se os sites respondem na porta correta e retornam o conteúdo esperado. Essa abordagem de "testar o que foi construído" é um pilar do **DevSecOps**, onde a validação automatizada é integrada ao pipeline de entrega, garantindo que as configurações não apenas funcionem, mas também estejam seguras e em conformidade.

```
# Exemplo de tarefas de validação
---
- name: Validar configuração do Nginx
 hosts: webservers
 become: true
 tasks:
 - name: Verificar status do serviço Nginx
 ansible.builtin.service_facts:
 register: service_status

 - name: Reportar status do Nginx
 ansible.builtin.debug:
 msg: "Nginx está rodando: {{ service_status.ansible_facts.services['nginx.service'].state == 'running' }}"

 - name: Verificar acessibilidade dos sites via HTTP
 ansible.builtin.uri:
 url: "http://{{ item.name }}:{{ item.port }}"
 status_code: 200
 loop: "{{ sites }}"
 register: http_check
 failed_when: http_check.failed or http_check.status != 200
 ignore_errors: true
```

# Gerenciamento de Segredos com Variáveis

Nem todas as variáveis são criadas da mesma forma. Algumas, como nomes de domínio ou diretórios raiz, podem ser armazenadas em texto simples sem grandes preocupações. Outras, no entanto, são informações altamente sensíveis, como senhas de banco de dados, chaves de API, credenciais de acesso a serviços em nuvem ou certificados SSL. Armazenar esses "segredos" em texto simples em seus playbooks ou repositórios Git é uma falha de segurança grave e inaceitável.

## Ansible Vault

A solução robusta para gerenciamento de segredos

### Criptografia

Arquivos YAML protegidos

### Descriptografia

Em tempo de execução

### Segurança

Nunca exposto em texto

A segurança é um pilar fundamental em qualquer estratégia de automação, e o Ansible oferece uma solução robusta para o gerenciamento de segredos: o **Ansible Vault**. O Vault permite que você criptografe arquivos ou variáveis individuais, garantindo que informações sensíveis permaneçam protegidas mesmo que seu repositório seja comprometido. É como ter um cofre digital para suas informações mais valiosas, acessível apenas com a chave correta.

Ao usar o Ansible Vault, você criptografa os arquivos YAML que contêm seus segredos. Quando o Ansible precisa usar essas variáveis, ele solicita a senha do Vault (ou a obtém de um arquivo de senha seguro) para descriptografá-las em tempo de execução. Isso significa que seus segredos nunca são expostos em texto simples no disco ou no controle de versão. Integrar o Ansible Vault é uma prática essencial de **DevSecOps**, garantindo que a segurança seja incorporada desde o início do ciclo de vida da sua infraestrutura como código.

```
# Exemplo de uso do Ansible Vault

# 1. Criar um arquivo criptografado para segredos
ansible-vault create group_vars/all/secrets.yml

# 2. Editar um arquivo existente com o Vault
ansible-vault edit group_vars/all/secrets.yml

# Conteúdo de secrets.yml (criptografado)
# database_password: MinhaSenhaSuperSecreta123
# api_key: abcdef1234567890

# Ao executar o playbook, você precisará fornecer a senha do Vault:
ansible-playbook meu_playbook.yml --ask-vault-pass
```

# Boas Práticas e Tendências

## Em Variáveis e Templates

Dominar variáveis e templates é um passo gigante na automação com Ansible, mas adotar boas práticas e estar atento às tendências pode elevar ainda mais a qualidade e a segurança das suas automações. Não se trata apenas de fazer funcionar, mas de fazer funcionar bem, de forma sustentável e segura.

### Boas Práticas

- **Nomeclatura Clara**

  Use nomes descritivos e consistentes (ex: `nginx_http_port` em vez de `port`)

- **Modularidade**

  Organize variáveis em group_vars e host_vars de forma lógica

- **Documentação**

  Comente seus templates e arquivos de variáveis

- **Validação**

  Sempre valide a saída dos templates

- **Evitar Hardcoding**

  Use variáveis sempre que possível

### Tendências

#### GitOps como Padrão

Git é a única fonte da verdade. Variáveis e templates são a configuração desejada da infraestrutura.

#### DevSecOps

Gerenciamento de segredos com Vault, varredura de código IaC para vulnerabilidades.

#### AIOps

Fatos do Ansible alimentam IA para prever falhas e otimizar recursos.

# Desafios Comuns e Como Superá-los

Mesmo com um bom entendimento de variáveis, fatos e Jinja2, você inevitavelmente encontrará desafios. A automação, por mais poderosa que seja, não está isenta de complexidades. Reconhecer e saber como superar esses obstáculos é parte do processo de se tornar um especialista em Ansible.

1

### Precedência de Variáveis

**Problema:** Mesma variável definida em vários locais

**Solução:** Consulte a documentação oficial sobre precedência. Use `debug` para imprimir valores e identificar qual fonte está sendo usada.

2

### Erros de Sintaxe no Jinja2

**Problema:** Erro de digitação em `{{ }}` ou `{% %}`

**Solução:** Use ferramentas de linting como `ansible-lint`. Teste templates com dados de exemplo. Use `debug` para inspecionar variáveis.

3

### Depurar Fatos

**Problema:** Fato esperado não está presente ou tem formato diferente

**Solução:** Use módulo `setup` e `debug` para inspecionar todos os fatos disponíveis e entender sua estrutura.

### Dica de Ouro

A paciência e a metodologia são chaves para a depuração eficaz. Sempre teste em pequenos incrementos e valide cada etapa.

# Consolidação e Próximos Passos

Chegamos ao fim de uma jornada crucial no universo do Ansible. Vimos como as **variáveis** transformam playbooks estáticos em ferramentas dinâmicas e reutilizáveis, permitindo que você adapte configurações a diferentes ambientes e hosts. Exploramos os **fatos do Ansible**, esse tesouro de informações que seus managed nodes fornecem automaticamente, capacitando seus playbooks a tomar decisões inteligentes e contextuais. E, finalmente, desvendamos o poder do **Jinja2**, o motor de templates que dá vida a essas variáveis e fatos, gerando arquivos de configuração personalizados e complexos a partir de um único molde.

## Em prática

A capacidade de usar variáveis, fatos e templates é o que diferencia um automatizador básico de um especialista. Comece aplicando esses conceitos em pequenos projetos, como a configuração de um serviço simples ou a criação de um arquivo de texto dinâmico. A prática constante, a experimentação com diferentes níveis de precedência de variáveis e a criação de templates cada vez mais complexos solidificarão seu aprendizado e abrirão portas para automações mais sofisticadas e eficientes.

> "A verdadeira magia da automação reside na capacidade de criar configurações dinâmicas que se adaptam a diferentes contextos sem exigir uma reescrita constante."

# Autoavaliação

## Teste seus conhecimentos

**1** Qual das seguintes opções descreve melhor a principal função das variáveis no Ansible?

- a) Apenas armazenar senhas de forma segura.
- b) Permitir que playbooks sejam estáticos e imutáveis.
- c) Habilitar a criação de configurações dinâmicas e reutilizáveis.
- d) Exclusivamente para definir o sistema operacional dos managed nodes.

**2** Em qual diretório você normalmente definiria variáveis que se aplicam a um grupo específico de servidores (ex: webservers)?

- a) host_vars/
- b) playbook_vars/
- c) group_vars/
- d) ansible_facts/

**3** Qual é o principal motor de templates utilizado pelo Ansible para gerar arquivos de configuração dinâmicos?

- a) ERB
- b) Mustache
- c) Jinja2
- d) Handlebars

**4** Qual módulo Ansible é usado para coletar informações detalhadas (fatos) sobre os managed nodes?

- a) ansible.builtin.copy
- b) ansible.builtin.template
- c) ansible.builtin.setup
- d) ansible.builtin.debug

**5** Explique como a combinação de variáveis, fatos e templates Jinja2 contribui para a implementação de uma estratégia de Infraestrutura como Código (IaC) mais robusta e escalável, citando pelo menos um benefício prático.

---

## Gabarito

# 1

**Resposta: c)**

# 2

**Resposta: c)**

# 3

**Resposta: c)**

# 4

Resposta: c)

# Próxima Aula e Recursos Adicionais

> **Próxima Aula**
>
> Na **Aula 22 – Condicionais, Laços e Handlers**, aprofundaremos ainda mais a lógica de controle dos seus playbooks, aprendendo a executar tarefas apenas sob certas condições, a repetir ações para múltiplos itens e a gerenciar reinícios de serviços de forma inteligente.

## Recursos Adicionais

### Documentação Oficial do Ansible

Para detalhes técnicos e exemplos aprofundados sobre variáveis, fatos e módulos.

### Documentação do Jinja2

Para explorar todas as funcionalidades do motor de templates, filtros e funções avançadas.

### Artigos sobre GitOps

Para entender a aplicação de variáveis e templates no contexto de CI/CD e infraestrutura declarativa.

---

> **NOTA IMPORTANTE:** As informações regulatórias/legais/técnicas desta aula estão atualizadas até 2025. Consulte sempre fontes oficiais para verificar alterações.

**Parabéns por concluir a Aula 21!** Você agora possui as ferramentas essenciais para criar automações dinâmicas, escaláveis e profissionais com Ansible. Continue praticando e explorando novos desafios!