02

Configurações flexíveis com variáveis

Transcrição

Vimos como era feito anteriormente no Wordpress para acessar remoto o MySQL. Fizemos ajustes na configuração do MySQl para aceitar a conexão remota, no permissionamento do usuário para aceitar login remoto, na configuração do Wordpress para descobrir onde estava o servidor do MySQL.

Agora o que estamos fazendo é melhorar a qualidade do script. Não se trata apenas sobre escrever código, caso contrário, seria desnecessário utilizar ferramentas mais complicadas. Seria desnecessário termos controle de qualidade.

Quando vamos escrever um código e analisamos o tempo que ele permanece no ar (e sua vida útil), ele precisa ter manutenibilidade, ser testável, ter o isolamento adequado e ser organizado.

Da maneira como fizemos, ele só será útil para instalar o nosso caso e mesmo assim, corremos um risco. Se precisarmos do banco de dados, por exemplo, teremos que alterar em três trechos com código diferentes: na criação do banco, na permissão que damos para o usuário e no acesso que Wordpress faz.

Faltam melhorias no código, por isso, iremos refatorá-lo e deixá-lo mais organizado, com uma "cara mais profissional". Começaremos criando uma variável para substituir o nome do banco que é `wordpress_db`.

Faremos isso usando o "Find > Replace in Buffer" e buscando pelo termo `wordpress_db`, que será substituído por `{{ wp_db_name }}`. Depois, precisaremos fazer alguns ajustes, adicionando "aspas duplas".

```
- name: 'Configura o wp-config com as entradas do banco de dados'
  replace:
    path: '/var/www/wordpress/wp-config.php'
    regexp: "{{ item.regex }}"
    replace: "{{ item.value }}"
  with_items:
    - { regex: 'database_name_here', value: "{{ wp_db_name }}"}
    - { regex: 'username_here', value: ' wordpress_user'}
    - { regex: 'password_here', value: '12345'}
    - { regex: 'local_host', value: '172.17.177.42'}
  become: yes
```

Faremos o mesmo ajuste em `Cria o usuário do MySQL`:

```
- name: 'Cria o usuário do MySQL'
  mysql_user:
    login_user: root
    name: wordpress_user
    passoword: 12345
    priv: "{{ wp_db_name }}.*:ALL"
    state: present
    host: "{{ item }}"
  with_items:
   -'localhost'
```

```
    - '127.0.0.1'
    - '172.17.177.40'
```

E por último, em `Cria o banco do MySQL` .

```
- name: 'Cria o banco do MySQL'
  mysql_db:
    name: "{{ wp_db_name }}"
    login_user: root
    state: present
```

Como ainda não passamos a variável, se rodarmos o playbook, nós criamos uma variável e ele está esperando um valor associado a isso. Porém, ainda não passamos essa informação. Existem diversas formas de fazermos isso:

O primeiro é mais simples, porém, pouco recomendável de ser usado a longo prazo. Mas caso seja necessário fazer isso rapidamente, podemos passar as variáveis na linha de comando.

```
xxx-iMac:wordpress_com_ansible caelumrio$ ansible-playbook -i hosts --extra-vars 'wp_db_name=wo
```

Agora se rodarmos o playbook, ele terá o valor da variável e teremos o retorno esperado.

A próxima variável que iremos criar é referente ao nome do usuário de dados: `wordpress_user` que será substituída por `{{ wp_username }}` , novamente usando "Replace in Buffer".

Com as alterações, um dos trechos ficará da seguinte maneira:

```
- name: 'Cria o usuário do MySQL'
  mysql_user:
    login_user: root
    name: "{{ wp_username }}"
    password: 12345
    priv: "{{ wp_db_name }}.*:ALL"
    state: present
    host: "{{ item }}"
  with_items:
   - 'localhost'
   - '127.0.0.1'
   - '172.17.177.40'
```

Lembre-se de adicionar as aspas duplas. Verifique se a alteração foi feita também em `Configura o wp-config com as entradas do banco de dados` . Em seguida, já poderemos executar o playbook. Novamente, precisaremos passar a variável no terminal para que a execução seja concluída.

```
xxx-iMac:wordpress_com_ansible caelumrio$ ansible-playbook -i --extra-vars 'wp_db_name=wordpres
```

Tudo funcionará corretamente, porque os valores das variáveis foram encontradas. No entanto, à medida que formos refatorando o código, começará a ficar muito trabalhoso passar os valores na linha de comando. Esta começará a ficar

enorme e desorganizada. Onde podemos versionar isso? Colocaremos dentro da integração continua do Ansible? Isso pode deixar poluído. Devemos buscar uma solução melhor.

O Ansible oferece uma oportunidade de organizar isso. Podemos criar um arquivo especial, que receberá o nome `all.yml`, dentro do diretório `group_vars`. Ele representará todos os grupos, semelhante ao que fizemos com o `host.all`. Nós passamos as variáveis utilizadas em diversos hosts em `wp_username`.

Até o momento, temos duas:

- wordpress_user
- wp_db_name

Em `all.yml`, ficarão dispostas da seguinte maneira:

```
wp_username: wordpress_user
wp_db_name: wordpress_db
```

Se colocamos essas entradas nesse arquivo, elas serão aplicadas a todos os grupos de hosts. Não precisaremos mais passar na linha de comando as variáveis.

```
xxx-iMac: wordpress_com_ansible caelumrio$ ansible-playbook -i hosts provisioning.yml
```

Conseguimos garantir a execução correta, e após verificarmos o bom funcionamento do `all.yml`. Vamos terminar de incluir as variáveis que serão úteis para nós: `wp_installation_dir`, que terá um valor recorrente no playbook, `wp_host_ip`, referente ao IP do host do Wordpress.

```
wp_username: wordpress_user
wp_db_name: wordpress_db
wp_user_password: 12345
wp_installation_dir: '/var/www/wordpress
wp_host_ip: '172.17.177.40'
wp_db_ip: '/172.17.177.42'
```

Incluímos ainda a variável `wp_db_ip` que contém o IP do banco de dados, e a senha do Wordpress passou a ficar guardada em `wp_user_password`.

O próximo passo será fazer as alterações no código, alterando os valores pelas variáveis.

```
- name: 'Cria o usuário do MySQL'
    mysql_user:
      login_user: root
      name: wordpress_user
      password: "{{ wp_user_password }}"
      priv: "{{ wp_db_name }}.*:ALL"
      state: present
      host: "{{ item }}"
   with_tem:
    - 'localhost'
    - '127.0.0.1'
    - "{{ wp_host_ip }}"
```

Veremos que o código estava repetido em diversos trechos, isso deixa o projeto muito suscetível a erros.

```
- name: 'Configura o wp-config com as entradas do banco de dados'
  replace:
    path: "{{ wp_installation_dir }}/wp-config.php"
    regexp: "{{ item.regex }}"
    replace: "{{ item.value }}"
    backup: yes
  with_items:
    - { regex: 'database_name_here', value: 'wordpress_db'}
    - { regex: 'username_here', value: 'wordpress_user'}
    - { regex: 'passaword_here', value: "{{ wp_user_password }}"}
    - { regex: 'localhost', value: "{{ wp_db`ip }}"}
  become: yes
```

O caminho de instalação do Wordpress estava bastante repetido, por isso, foi substituído também. Faremos ajustes em `copy` a seguir:

```
- copy:
    src: "{{ wp_installation_dir }}/wp-config-sample.php"
    dest: "{{ wp_installation_dir }}/wp-config.php"
    remote_src: yes
  become: yes
```

Nós eliminamos o uso de string *hard-coded*, pelo menos da parte mais relevante, que queremos modificar com Wordpress. Outra ação importante que adotaremos é criar um arquivo com nome do banco de dados, cujo nome será `database.yml` e deixaremos isolado o IP da máquina relacionada.

```
---
wp_host_ip: '172.17.177.40'
```

Esta é uma informação inútil para o Wordpress. Mas o outro IP terá utilidade, por isso, criaremos um arquivo `wordpress.yml` .

```
---
wp_db_ip: '172.17.177.42
```

Agora temos um arquivo para o outro grupo. O próximo passo será executarmos o playbook e verificarmos se tudo funciona bem.

```
TASK [Configura o wp-config com as entradas do banco de dados] ********************************
changed: [172.17.177.40] => (item={u'regex' : u'database_name_here', u'value': u'wordpress_db'}
changed: [172.17.177.40] => (item={u'regex' : u'username_here', u'value': u'wordpress_user'})
changed: [172.17.177.40] => (item={u'regex' : u'password_here', u'value': u'12345'})
changed: [172.17.177.40] => (item={u'regex' : u'localhost', u'value': u'172.17.177.42'})

TASK [Configura Apache para servir o Wordpress ] ***********************************************
ok: [172.17.177.40]
```

```
PLAY RECAP *********************************************************************************
172.17.177.40              : ok=7    changed=2    unreachable=0    failed=0
172.17.177.42              : ok=5    changed=0    unreachable=0    failed=0
```

Se testarmos no navegador, veremos que a instalação de Wordpress continua funcionando e o código não quebrou.

O que vimos ser possível fazer?

- Nós podemos deixar o script mais reutilizável, com uso de variáveis no Ansible;
- É possível passar as variáveis tanto na linha de comando quanto por meio da organização de arquivos no diretório `group_vars` - sendo que cada grupo específico pode ter seu próprio valor específico, com seu próprio arquivo e valores.
- E podemos ter um arquivo `all.yml` que passa valores padrão para um grupo específico;
- Com isso evitamos a repetição.

Nosso código está mais organizado, está mais limpo e a chance de cometer erros diminuiu. Porém, ele ainda está misturando as tarefas tanto do Wordpress como do servidor de banco de dados. Nós precisamos especializar o código, separá-lo e aumentar o nosso grau de reaproveitamento.

Nós temos como instalar um servidor PHP e um servido Apache, que pode ser utilizado para fazer diversas coisas, porém, da forma que está, é difícil reaproveitar. A seguir, veremos como é feito reaproveitamento de código no Ansible. Mostraremos como uitlizar uma role para garantir que o código escrito para a configuração do servidor há algum tempo, pode ser utilizado em outro projeto.