04

Usando Defaults e dependências

Transcrição

Mostramos como modularizar melhor o nosso código Ansible, criando as nossas roles. A aplicação de três camadas está funcionando e nós conseguimos reaproveitar as partes de código que escrevemos - para escrevermos também o provisionamento de aplicações diferentes. Veremos a seguir como melhorar as roles, protegendo-as de forma de uso estranhas, que alguém pode esquecer de declarar uma variável ou outros lapsos do usuário. Podemos melhorar o comportamento padrão delas.

Também desejamos esclarecer qual é a dependência entre as roles. Começaremos apresentando como é possível declarar valores padrões para roles e garantir que elas tenham um comportamento que respeite o mínimo aceitável. Seguiremos com o nosso exemplo.

Se analisarmos a role do MySQL, passamos uma lista de hosts na quais o usuário vai ter permissão de acesso.

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

Neste trecho, nós usamos a variável `wp_host_ip` , e o que acontecerá se o usuário esquecer de passá-la? Vamos entender isso, analisando o arquivo `group_var/database2.yml` . Quando executarmos o Ansible, ele não vai associar o valor da variável `wp_host_ip` a uma máquina, porque faltou informar um host que seja parte do grupo `database2` .

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

Para rodarmos o playbook, executaremos o seguinte comando:

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

Ao executarmos o playbook, teremos um erro como retorno.

```
TASK [mysql: Cria o usuário do MySQL] ******************************************************
fatal: [172.17.177.42]: FAILED! => {"msg": "'wp_host_ip' is undefined"}
        to retry, use: --limit @/Users/marcoscropalato/wordpress_com_ansible/provisioning.retry
```

```
PLAY RECAP *********************************************************************
172.17.177.42              : ok=3    changed=0    unreachable=0    failed=1
```

Isto ocorreu porque o playbook está sendo executado contra `host_group2` (`database`). O que faremos na situação em que o usuário não passe o IP e assim permitir o acesso a um determinado usuário? Simplesmente, a role deixará de funcionar? Ela deve funcionar, para isto, a variável receberá valores default. A seguir, criaremos a pasta `defaults` e dentro, salvaremos o arquivo `main.yml` . A variável era uma string, mas agora, passará a ser uma lista, na qual definiremos alguns padrões:

```
---

wp_host_ip:
  - localhost
  - '127.0.0.1'
```

Depois, no playbook `main.yml - tasks` , `Cria o usuário do MySQL` recebe uma lista que iremos remover completamente, ficando apenas a variável. Após a alteração, o trecho ficará da seguinte maneira:

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

Agora que não existe um valor configurado para a variável, ele deve usar o valor default. Em seguida, rodaremos o playbook novamente e verificar se o Ansible encontra o valor.

```
TASK [mysql: Cria o usuário do MySQL] ******************************************
ok: [172.17.177.42]

TASK [mysql: Cria o usuário do MySQL] ******************************************
ok: [172.17.177.42] => (item=localhost)
ok: [172.17.177.42] => (item=127.0.0.1)
```

Desta vez, ele conseguiu encontrar o valor. Com este tipo de declaração padrões, nós facilitamos o uso de uma role, por exemplo, imagine em um caso no qual trabalhamos 150 variáveis diferentes - ainda que seja uma situação improvável. No entanto, às vezes, o projeto é bastante complexo e tem tantos casos de uso que ele pode precisar de mais opções. Deixamos a lista de padrões declarados dentro do arquivo `main.yml - defaults` e a role passou a se comportar conforme os valores padrões.

Se o usuário precisar de valores diferentes, deverá criar `group_vars` para o grupo ou para `all.yml` . Voltaremos ao arquivo para o nome padrão `database.yml` , depois, no nosso playbook, declaramos as roles (`webserver` e `wordpress`) que estão rodando nos hosts do grupo Wordpress. Qual o problema disso? Na primeira vez que criamos o host

relacionado, a role Wordpress depende do webserver instalado - ou seja, é necessário que já tenha ocorrido a instalação do Apache, PHP, para o funcionamento correto do Wordpress.

Se esquecermos de declarar a role de webserver, nós vamos quebrar o código e, talvez, não ficará explícito o porquê disso acontecer no momento da execução. O que faremos, então? Nós vamos garantir que Wordpress explicite essa dependência do webserver, criando o arquivo `main.yml` dentro do diretório `meta` e, em seguida, estabeleceremos sua lista de dependência:

```
---
dependencies:
  - webserver
  -
```

Por enquanto, a lista de dependência contém apenas um item. O próximo passo será rodar o playbook e verificaremos se as duas roles serão executadas.

```
TASK [webserver : Instala pacotes de dependencia do sistema operacional] **********
ok: [172.17.177.40] => (item=[u'php5' , u'apache2', u'libapache2-mod-php5', u'php5-gd', u'libssl

TASK [Baixa o arquivo de instalacao de Wordpress] ******************************************
ok: [172.17.177.40]
```

Tudo está funcionando corretamente, graças à dependência adicionada em `main.yml - wordpress/meta` . Neste caso, seria executado apenas `wordpress` e não será verificado se os pacotes do PHP e do Apache estavam instalados. Se a máquina não estiver no estado desejado, não conseguiremos prosseguir.

Nós falamos sobre os valores padrões que podemos estabelecer nas variáveis de uma role, falamos sobre como eles podem ser substituídos por `group_vars` e falamos sobre como determinar dependências entre roles. Faça os exercícios desta aula e continuaremos a seguir.