03

Reinstalando o servidor com um click

Transcrição

Anteriormente, nós separamos a máquina do banco de dados da máquina do Wordpress, ficando cada uma no seu ambiente. Depois, configuramos a máquina do banco de dados.

No entanto, a máquina do Wordpress ficou com a configuração antiga, do ambiente completo dentro dela e não está conectando com o banco de dados remoto. Por isso, nós vamos limpá-la, destruindo a já existente, então, construiremos uma do zero, com o script correto - que não instala o MySQL nela. Iremos configurar o Wordpress para conseguir acessar o novo MySQL.

Primeiramente, removeremos todos os comando relacionados à instalação do banco de dados da máquina do Wordpress. No grupo de hosts do Wordpress, removeremos os pacotes relacionados ao MySQL e à criação do banco de dados e do usuário.

```
tasks:
  - name: 'Instala pacotes de dependencia do sistema operacional'
    apt:
      name: "{{ item }}"
      state: latest
    become: yes
    with_items:
      - php5
      - apache2
      - libapache2-mod-php5
      - php5-gd
      - libssh2-php
      - php5-mcrypt
      - mysql-server-5.6
      - python-mysqldb
      - php5-mysql

  - name: 'Baixa o arquivo de instalação do Wordpress'
    get_url:
      url: 'http://wordpress.org/lastest.tar.gz'
      dest: '/tmp/wordpress.tar.gz'
```

No código, restou apenas as linhas relacionadas com a instalação de PHP, Apache e Wordpress. Nós poderíamos desinstalar o MySQL, da máquina virtual. Porém, nosso objetivo é demonstrar como podemos usar o Ansible. Logo, se minha máquina está provisionada corretamente, será fácil criá-la.

O próximo passo será destruir nossa máquina virtual e criar uma nova em seguida.

```
$ vagrant destroy -f wordpress
==> wordpress: Forcing shutdown of VM...
==> wordpress: Destroying VM and associated drives...

$ vagrant up wordpress
Bringing machine 'wordpress' up with 'virtualbox' provider...
==> wordpress: Importing base box 'ubunut/truty64'...
```

Vamos conferir se a máquina virtual foi criada usando o comando `vagrant status` .

```
$ vagrant status
Current machine states:

wordpress             running(virtualbox)
mysql                 running(virtualbox)
```

Teremos que configurar a nova máquina, rodando o playbook.

```
$ ansible-playbook -i hosts provisioning.yml

PLAY [database] ****************************************************************

TASK [Gathering Facts] *********************************************************
The authenticity of hosts '172.17.177.42 (172.17.177.42)' cant´be established.
ECDSA key fingerprint is SHA256:7jp5Cn61cGCGuG0fOwxCeqb6h+CA89IPooFYQQf5qY.
Are you sure you want to continue connecting (yes/no)? yes
ok: [172.17.177.42]

TASK [Instala pacotes de dependencia dos sistema operacional] ******************
ok: [172.17.177.42] => (item=[u'mysql-server-5.6', u'python-mysqldb'])
```

Ele novamente vai pedir para gravar a chave SSH, considerando que criamos uma máquina nova. Em seguida, ele vai iniciar a instalação de todos os pacotes, fará o download e vai configurar o Wordpress.

Após a configuração, teremos o seguinte retorno:

```
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'})

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

PLAY RECAP *********************************************************************
172.17.177.40              : ok=8   changed=7    unreachable=0    failed=0
172.17.177.42              : ok=4   changed=0    unreachable=0    failed=0
```

Conseguimos perceber que a máquina com o IP de final `40` foi a modificada. Além disso, o playbook instalou as dependências, baixou Wordpress, descompactou, modificou as entradas necessárias para acessar o banco de dados e iniciou o Apache com o arquivo de configuração novo do Wordpress. Quase sem novidades.

Agora vamos acessar o Wordpress. Se digitarmos o IP na barra de navegação, teremos um erro ao estabelecer a conexão com o banco de dados.

```
![error establishing a database connection](https://s3.amazonaws.com/caelum-online-public/746-ar
```

Por que isso aconteceu? Nós passamos o nome do banco, o usuário e a senha, mas não informamos o host onde roda o banco de dados. Para fazermos isso, ao trabalharmos com Wordpress, o que feito em seu arquivo de configuração é colocar isso em uma constante chamada `DB_HOST`. Descobriremos isso, usando primeiramente o comando `less`:

```
vagrant@vagrant-ubuntu-trusty-64:=~$ less /var/www/wordpress/

index.php              wp-admin/              wp-config-sample.php   wp-links-opml.php    wp-set
license.txt            wp-blog-header.php     wp-content/            wp-load.php          wp-sig
readme.html            wp-comment-post.php    wp-cron.php            wp-login.php         wp-tra
wp-activate.php        wp-config.php          wp-includes/           wp-mail.php          xmlrpc
```

Depois, digitaremos `wp-config`:

```
vagrant@vagrant-ubunut-trusty-64:~$ less /var/www/wordpress/wp-config.php
```

Ele coloca na constante `DB_HOST`, depois, escreve `localhost`.

```
/** MySQL database username */
define('DB_USER', 'wordpress_user');

/** MySQL database password */
define('DB_PASSWORD', '12345');

/** MySQL hostname */
define('DB_HOST', 'localhost');

/** Database Charset to use in creating database tables. */
define('DB_CHARSET', 'utf8');

/** Database Collate type. Don´t change this if in doubt. */
define('DB_COLLATE', '');
```

Nós precisamos alterar essa entrada para o IP da máquina onde roda o banco de dados. Faremos isso facilmente adicionando mais uma entrada de `regex` no `replace` do `Configura o wp-config com as entradas do banco de dados`.

```
- 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 }}"
    backup: yes
  with_items:
    - { regex: 'database_name_here', value: 'wordpress_db'}
    - { regex: 'username_here', value: 'wordpress_user'}
    - { regex: 'password_here', value: '12345'}
    - { regex: 'localhost', value: '171.17.177.42'}
  become: yes
```

O valor é o IP que colocamos na máquina de banco de dados. Agora sairemos da máquina virtual, e rodaremos o playbook. Veremos que o IP foi inserido na listagem.

```
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'})
```

Apesar do ajuste, se testarmos acessar o IP pelo navegador, ainda **não teremos sucesso** na conexão com o banco de dados. Por que não funcionou? Se conferirmos na máquina, o arquivo parece estar correto, e o IP também será exibido. O problema está que, além de precisarmos avisar ao Wordpress, a necessidade de se conectar a uma máquina remota, a instalação padrão do MySQL, não aceita duas coisas:

- não aceita a conexão de nenhum IP que não seja de localhost (`127001`);
- os usuários que criamos no MySQL, por padrão, quando colocamos algum tipo de permissão, é apenas para permissão local.

Teremos que modificar essas duas informações. Começaremos modificando o usuário, que é uma informação que temos mais acessível.

Quando criarmos um usuário do Wordpress, adicionaremos uma lista de hosts possíveis de serem conectarem com o MySQL, que é a própria máquina local. Ela será acessada por dois nomes: `localhost` e pelo IP `127.0.0.1` . Incluiremos também o IP da máquina que roda Wordpress, para definir que esta tenha permissão de utilizar o usuário quando se conectar com MySQL. Qual parâmetro vai receber o loop? O parâmetro `host` , lembrando que **devemos respeitar a indentação**.

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

Após verificarmos também a escrita correta do nome dos parâmetros, se rodarmos o playbook, veremos exibido a entrada padrão (`localhost`), que já estava inclusa e o retorno nos mostrará `127.0.0.1` e `172.17.177.40` .

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

Mas se testarmos no navegador, ainda não conseguiremos estabelecer a conexão. Desta vez, a explicação é porque o servidor do MySQL não aceita conexão remota, por isso, teremos que alterar o arquivo de configuração do mesmo. Nós adotaremos uma estratégia semelhante usada com o Apache, copiaremos o arquivo de instalação do MySQL que se encontra em `/etc/mysql/my.cnf` .

```
vagrant@vagrant-ubuntu-trusty-64:~$ cat /etc/mysql/my.cnf
```

Depois, copiaremos o arquivo `my.cnf` para o diretório `files` , em seguida, modificaremos a entrada `bind-address` que está fixa para `127.0.0.1` .

```
# Instead of skip-networking the default is now to listen only on
#localhost which is more compatible and is not less secure.
bind-address    = 0.0.0.0
#
```

Agora temos que informar ao Ansible a necessidade de fazer a cópia do arquivo. Para isto usaremos o `copy` dentro de `Configura MySQL para aceitar conexões remotas` .

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

- name: 'Configura MySQL para aceitar conexões remotas'
  copy:
    src: 'files/my.cnf'
    dest: '/etc/mysql/my.cnf'
  become: yes
```

Faremos isso como root, ao configurarmos `become` como `yes` , porque estamos modificando um arquivo administrativo do sistema operacional. Rodaremos a seguir.

```
$ ansible-playbook -i hosts provisioning.yml
```

No fim, teremos o seguinte retorno:

```
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=1    unreachable=0    failed=0
```

Mas se acessarmos o navegador, ainda não vai funcionar. O que faltou fazer, mas foi feito no Apache? Observe o código abaixo:

```
- name: 'Configura MySQL para aceitar conexões remotas'
  copy:
    src: 'files/my.cnf'
    dest: '/etc/mysql/my.cnf'
  become: yes
  notify:
   - restart mysql
```

No Apache, nós reiniciamos após escrever o arquivo de configuração. Não fizemos isso no MySQL, agora, adicionamos `restart mysql` . O próximo passo é incluir `handlers` com `restart mysql` .

```
- hosts: database
  handlers:
    - name: restart mysql
      service:
        name: mysql
        state: restart
      become: yes
  tasks:
```

Apesar dos ajustes, ele não vai conseguir copiar novamente o arquivo `my.cnf` , porque ele já existe. Para que ele possa chamar o handler, faremos uma pequena alteração no arquivo, incluindo `*` após `Basic Settings` .

```
[mysqld]
#
# * Basic Settings *
#
```

Desta forma, justificaremos a realização da cópia. Quando o playbook for executado, o `handler` vai ser executado.

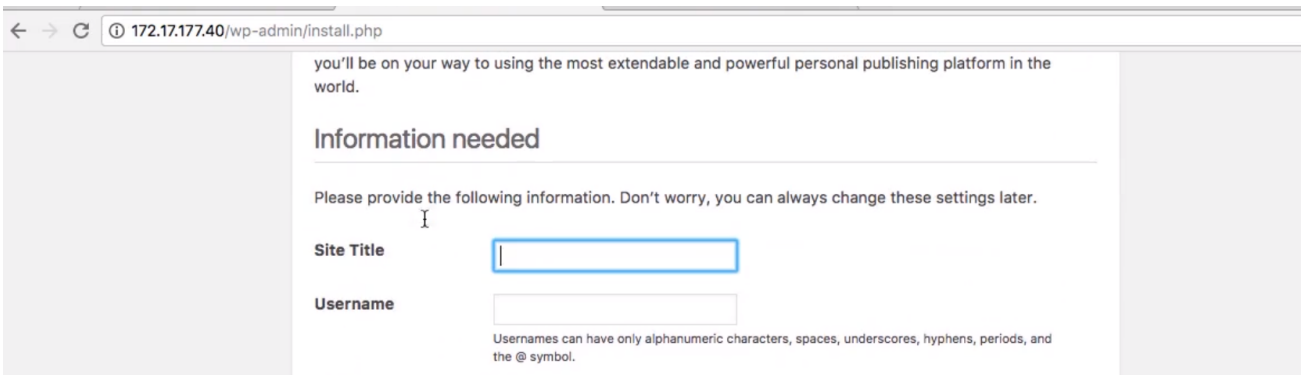
```
TASK [Configura MySQL para aceitar conexões remotas] ***************************************
changed: [172.17.177.42]

RUNNING HANDLER [ restart mysql] ***********************************************************
changed: [172.17.177.42]

PLAY [wordpress] ***************************************************************************

TASK [Gathering Facts] *********************************************************************
```

Perceba que o handler foi disparado, reiniciou com sucesso e, agora, esperamos conseguir nos conectarmos com o banco de dados. Se digitarmos o IP no navegador, veremos exibido a seguinte tela:



Revisando os passos seguidos para fazer a página do Wordpress funcionar:

- Separamos o playbook em dois grupos de `hosts`, que usam as duas máquina virtuais criadas anteriormente;
- Colocamos no inventário do Ansible os IPs e como eles se conectam nessas máquinas;
- Separamos as tarefas do MySQL, de forma que a máquina do Wordpress fique sem comandos do MySQL;
- E a máquina do MySQL fique sem comandos do Wordpress;
- Incluímos a informação de como o Wordpress conecta no servidor remoto, mudando o `replace`;
- Além disso, incluímos o arquivo de configuração novo `my.cnf`, liberando o MySQL para aceitar conexões remotas.
- Demos permissão para o usuário criado para o Wordpress de aceitar conexões de servidores remotos colocando a lista de IPs.

Temos o nosso ambiente separado - tanto o Wordpress, como o MySQL foram colocados cada qual em uma máquina diferente. E o próximo passo?

Para começarmos, o arquivo `provisioning.yml` tem diversas informações repetidas. Existem diversos caminhos, informações de usuários e nomes de banco de dados espalhados pelo playbook. Encontramos IPs de máquinas em diversos trechos, começaremos a organizar isso e a refatorar o código, colocando tais dados em variáveis. Assim, tornaremos nosso código mais limpo, reutilizável e, principalmente, mais seguro. Evitaremos que o próprio desenvolvedor cause problemas ao trabalhar com o projeto, por exemplo, criar bugs ao fazer alterações em uma parte e esquecer de fazer em outra.

Veremos como fazer essa refatoração com qualidade a seguir.