02

# Provisionando um MySQL

### Transcrição

Temos uma máquina virtual que é executada no Ubuntu, instala o Java, instala o Tomcat, configura o Tomcat com a minha aplicação web. Coloco um ip na minha máquina e eu consigo acessar o navegador. Inclusive tem um banco de dados rolando em memória.

Em memória? Não era em memória que gostaríamos que isso ocorresse, certo? Pelo menos no meu ambiente de produção isso não é interessante, neste caso gostaríamos de utilizar o MySQL.

O primeiro passo é a gente instalar o MySQL e configurá-lo, certo? Então vamos fazer isso. Na nossa configuração, a gente vai criar um banco de dados pro nosso Music Jungle. Primeiro passo, criar o banco: instalar o banco, e criar o banco. Vamos lá?

Como é que eu crio o banco MySQL? Num ambiente Linux, é bem comum, no Ubuntu, que eu faça um `apt-get : apt-get install mysql-server` . E com isso ele instala o mysql-server pra mim.

Legal, o apt-get instalou o mysql-server? Instalou. Agora eu preciso criar o meu banco. Pra eu criar o meu banco, eu vou com o `mysqladmin` , passo o meu usuário, que é `root` , e dou um `create` + nome de banco de dados, `musicjungle` . Com isso, eu tenho um espaço onde eu posso criar minhas tabelas e tudo mais o que eu precisar. Então:

```
mysqladmin -uroot create musicjungle;
```

Com isso eu crio o meu banco de dados. São dois passos, `aptget install` do `mysql-server` e `mysqladmin -uroot create musicjungle` . Eu executo esses dois passos e ele instala o MySQL pra mim, cria o banco de dados e eu posso até testar. Eu dou um:

```
mysql -uroot musicjungle
```

Conseguimos acessar o banco de dados do Music Jungle.

Contudo, estamos aqui para automatizar o processo de criação das máquina e configuração dos ambientes. Não devemos fazer execuções manualmente, pois elas são mais frágeis e sujeita a erros, uma vez que o projeto pode passar por vários desenvolvedores.

Acessaremos o Puppet para realizar as configurações. Primeira coisa, lá onde a gente falava que a gente queria ter o package do `openjdk` e o package do `tomcat` instalado, manda instalar o `mysql-server` ! Eu quero ter os 3 pacotes instalados. Manda instalar, ele que se vire, certo? Ele se vira.

Se eu tenho o package do `mysql-server` instalado, assim como no caso do Tomcat, o MySQL é aquele caso em que a gente está interessado em ter certeza de que ele está rodando. Eu preciso que ele esteja rodando. Então, o que é que eu vou definir? Um service.

O service, a gente já conhece. O service vai ter o nome de `mysql` . Ele depende, lá no final (require) do `mysql-server` , isso é, se ele nem conseguiu instalar o `mysql-server` , nem adianta tentar ver se o serviço está lá. O que mais? Quero ter certeza de que ele esteja rodando. Está rodando? Não está rodando? Está rodando ou não está rodando? Se não está

rodando, nem continua, porque eu não instalei uma parte do meu serviço. O que mais, `enable => true` pra ter certeza de deixar ativo o serviço, `hasstatus` e `hasrestart`. Todos eles têm isso, o MySQL e o Tomcat, então eu não preciso me preocupar.

```
package { ["openjdk-7-jre", "tomcat7", "mysql-server"]:
    ensure => installed,
    require => Exec["apt-update"]
}

service { "mysql":
    ensure => running,
    enable => true,
    hasstatus => true,
    hasrestart => true,
    require => Package["mysql-server"]
}
```

Agora que eu tenho o MySQL rodando, isto é, se eu der um `require` no service `mysql`, eu tenho certeza de que ele está rodando. Agora o que eu quero fazer mesmo?

Criar o banco? Como é que eu crio o banco? `mysqladmin` etc. etc. Eu vou executar alguma coisa no *bash*, certo? Se eu vou executar – a gente já até viu, o primeiro de todos os comandos, `exec`. Eu quero executar um comando. Eu tenho que defini-lo.

`exec`, vou chamar isso de `musicjungle` (podia chamar de musicjungledatabase, createdatabase, o que a gente achar que faz sentido), lembre-se, é o nome da minha tarefa, esse é o nome do `exec`. O nome é `musicjungle`. Qual é o comando que eu vou executar mesmo? Lembra: o *path* inteiro, `mysqladmin -uroot create musicjungle`.

Mas o path inteiro, Guilherme, não está no diretório `/usr/bin`? Está no `/usr/bin`! Então, eu podia escrever `/usr/bin/mysqladmin`, ou eu posso separar no próximo atributo. `path => "/usr/bin"`.

E separando nesse próximo atributo, no path, quer dizer que esse comando vai ser executado dentro desse diretório. São duas maneiras, cada uma tem uma pequena diferença. Ou está executando dentro desse diretório, ou em outro diretório.

```
exec { "musicjungle":
    command => "mysqladmin -uroot create musicjungle",
    path => "/usr/bin",
    require => Service["mysql"]
}
```

Então, eu estou executando o `mysqladmin -uroot create musicjungle` depois de ter certeza de que o serviço está rodando perfeitamente. A gente está dando `apt-get install`, a gente está garantindo que o serviço está rodando, e a gente está rodando o `mysqladmin` pra criar o banco.

Lembra que eu já tinha dado, na minha máquina, o `apt-get install`? O que acontece se eu dou `apt-get install` duas vezes no mesmo pacote? Não faz nada! Já está instalado, não tem diferença. Mas o que acontece se eu crio o mesmo banco duas vezes? O que acontece se eu mandar o `mysqladmin` criar o mesmo banco duas vezes?

Se eu executo esse script agora, teremos um pequeno problema: o install do `apt-get install` ele nem tenta, porque já está instalado o MySQL. Não tenta instalar de novo. Mas ele fala o seguinte: *criar o banco de dados deu erro*. Por quê? Porque o MySQL reclama se o banco de dados já existe.

Eu gostaria de executar essa tarefa, executar a tarefa `musicjungle` (que é equivalente a `mysqladmin -uroot create musicjungle` ) a não ser que o banco exista.

Como é que eu descubro se esse banco de dados já existe, mesmo? Se o banco de dados já existe é porque eu consigo me conectar a ele. Como é que eu me conecto? `mysql -uroot musicjungle` . Se esse comando for um sucesso, é porque o banco de dados já existe. Então, se esse comando for um sucesso, não cria de novo, não! Não tenta criar, não seja louco!

O que eu faço com esse nosso `exec` ? Eu falo:

- Exec, execute esse comando **somente se** (unless) esse `mysql -uroot musicjungle` não deu um sucesso. Quer dizer, se ele deu um sucesso, esse comando, não executa! Não quero criar o banco de novo, o banco já existe. Por favor, não o crie de novo.

```
exec { "musicjungle":
    command => "mysqladmin -uroot create musicjungle",
    unless => "mysql -u root musicjungle",
    path => "/usr/bin",
    require => Service["mysql"]
}
```

O `unless` faz com que eu possa executar esse script diversas vezes e o resultado seja o mesmo. Se o banco já existe, deixa o banco lá. Se o banco não existe, ele cria. Não é que o resultado é bem o mesmo, mas eu posso executar esse script diversas vezes, sem ter um efeito colateral indesejado. Isso é o que a gente chama quando o script é **idempotente**. Eu posso executá-lo diversas vezes e eu vou ter o resultado que eu estou esperando ali no final, que é o banco de dados estar criado, não vai dar pau em nenhuma delas.

Agora, se eu executar o `puppet apply` , joinha. Ele nem chega a executar o `create` , porque ele tentou conectar com o banco, e o banco já existia. É claro, se o banco não existisse, e o MySQL não estivesse instalado, ele ia ter que fazer tudo isso.

Aí é um ponto interessante: **não seja mão de vaca**. A gente não precisa mais ser mão de vaca agora. Por que não? Porque, antes, pra você criar uma máquina de produção, ou uma máquina de aceitação, seja lá qual for a máquina que você queria criar, você tinha que executar um monte de comando na mão. Era trabalhoso. Agora não é mais. Você quer criar uma máquina igual a uma máquina de produção, uma máquina igual a uma máquina de homologação, é só você dar o `vagrant up` e você tem a sua *virtual machine*.

Agora você dá um `sudo puppet apply` e você já tem tudo instalado. Não é mais caro criar uma máquina virtual com um ambiente igual ao de produção. E depois a gente vai ver como jogar isso no cloud, claro, como jogar isso em outros lugares.

Não é mais caro criar essa máquina que nem a produção ou homologação. Então, não seja mão de vaca. O que você faz? Você faz um `vagrant destroy` , destrói tudo, e dá um `vagrant up` . E depois do `vagrant up` , dá um `puppet apply` . Aplica o seu script do Puppet, e você tem uma máquina igualzinha à outra. Não seja mais mão de vaca, você está com um problema, você testou umas coisas num ambiente igual ao de produção, aqui local, e viu que era isso que você queria ou não era isso que você queria, não tenha dó: destrua! `vagrant destroy` e `vagrant up` . Não tenha dó! Não seja mão de

vaca. Quando você der um `vagrant up` e você der um `puppet apply`, o seu MySQL vai estar lá rodando, o seu Tomcat vai estar lá rodando.

De novo, **não seja mão de vaca** se você precisar de novo. Não seja mão de vaca, não economize vagrants destroys e vagrants ups. Porque a qualquer hora você tem um ambiente igual ao que você tinha antes. Esse é o poder do vagrant com o Puppet. Dei o `destroy`, dei o `up`, dei o `apply` do Puppet, e agora eu tenho uma máquina igualzinha a antes. Igualzinha à que eu tinha cinco minutos atrás.

Então, **tãn dãn**, o que eu faço? Acesso lá o Music Jungle, e eu tenho o meu serviço. Acesso-o, cadastro meu usuário e fico feliz e contente. Hum... Fico **quase** feliz e contente. Por quê? Porque quando eu chegar lá no meu MySQL, eu não estou ainda adicionando nada no meu MySQL. O MySQL até está lá, mas não tem nada na minha tabela do banco. Porque eu instalei o MySQL, é verdade, e eu vi o poder do `vagrant up`, do `vagrant destroy` e do `puppet apply`. Eu consigo criar a mesma máquina com todo o ambiente de configuração quantas vezes eu quiser. Mas eu ainda não falei pra minha aplicação web que é pra ela usar o MySQL.

De alguma maneira, eu tenho que fazer com que a minha aplicação web seja capaz de entender quando ela está em um ambiente de desenvolvimento, quando ela está em um ambiente de produção. E é isso que a gente vai fazer daqui a pouco.

Fazendo o código, o que é que a gente faz? Definir o pacote, criou o serviço do MySQL e falou:

- Olha, por favor, faça o `mysqladmin`, crie o banco de dados que eu estou desejando.

  Só que aí, quando eu o executo, eu vejo que ele reclama:

- Não, não, não, o banco de dados já existe.

Então eu vou lá e coloco um `unless`. Aí eu executo o `puppet apply` e ele funciona! Eu fico feliz e contente! Não sou mão de vaca! Não sou mão de vaca com vocês! `vagrant destroy`, `vagrant up`! Faço um ssh na minha máquina depois do `vagrant up` e dou um `sudo puppet apply`.

Quando eu dou um `sudo puppet apply`, ele instala tudo, tudo, tudo. Inclusive o banco. E aí eu posso dar um `mysql -uroot` e conecto no banco `musicjungle`. Ele está lá. Vou ao navegador, o navegador está lá. Consigo até me cadastrar. Só que, lembra, ele ainda não está usando o MySQL. Eu aprendi a adicionar novos pacotes, aprendi a executar qualquer tarefa, inclusive o `unless`, e aprendi o poder do `vagrant destroy`, do `vagrant up` e do `puppet apply`. Eu crio o mesmo ambiente quantas vezes eu quiser.

Agora... Como é que eu falo pra minha aplicação web que ela está num ambiente de desenvolvimento ou de produção? Como é que eu crio ambientes diferentes de configuração? Como é que eu crio configurações de ambientes diferentes? É o que a gente vai ver daqui a pouco. Primeiro, vamos lá pros exercícios!