4525 – Curso de Infraestrutura Ágil com Práticas DevOps

Desafio - Gerência de Configuração com Puppet

Neste desafio iremos utilizar os conhecimentos que tivemos na aula **Gerência de Configuração com Puppet** para desenvolver o módulo de instalação do memcached e garantir que um container do Docker esteja sempre rodando. Este desafio será divido em duas parte:

1. Vamos desenvolver um módulo do puppet para garantir a configuração do serviço memcached instalado na máquina, aprendendo a utilizar templates e herança de classes.

2. Desenvolver um módulo do puppet responsável por garantir que um container do Docker sempre esteja rodando.

Parte 1. Trabalhando com serviços, templates e heranças do Puppet

Na primeira parte do nosso desafio será criar um módulo do puppet responsável pelo serviço de Memcached dentro da Infraestrutura.

1. Crie um container do Docker para realizar a homologação deste modulo. Crie um container com o nome e hostname **puppet-hom** utilizando a imagem **homolog-template** que foi criada na aula de **Administrando e Gerênciando o Docker**, que já possui o agente do puppet instalado para poder homologar o modulo do puppet.

Dica: Caso você não possua a imagem homolog-template, você pode procurar no repositório do docker hub imagens que possui o agente do puppet instalado.

$ docker run -it --name puppet-hom --hostname puppet-hom homolog-template /bin/bash

2. Dentro do diretório /etc/puppet/modules/ realize a criação do módulo do puppet atráves do próprio comando do Puppet com o nome **puppet-memcached.**

$ cd /etc/puppet/modules/
$ puppet module generate puppet-memcached
$ cd puppet-memcached

3. Dentro do módulo do puppet, crie o arquivo de template para a configuração do memcached. Por padrão o serviço do memcached está configurado para escutar no endereço da interface de loopback (127.0.0.1) na máquina, porém para ser utilizado na Infraestrutura por outras máquinas deve ser colocado o endereço ip da máquina na qual o memcached será instalado. Sua missão é descobrir qual variável do facter possui as informações do endereço ip da máquina, configurando essa variável dentro do arquivo de template do módulo.

Dentro dos arquivos de template do Puppet você pode utilizar as variavéis das classes ou dos facters utilizando a syntaxe: <%= @var_name %>

templates/exemplo.erb

```
hostname = <%= @fqdn >
domain = <%= @domain >
```

No momento em que o agente do puppet realiza a configuração do template ele irá trocar o contéudo das variaveis pelo valor que trazido pelo facter instalado na máquina.

OBS: Caso o diretório de templates não exista, ele deve ser criado.

Segue o arquivo de configuração padrão do serviço memcached , não esqueça de torna o arquivo dinâmico alterando a linha **-l 127.0.0.1**.

`templates/memcached.erb`

```
# memcached template
-d
logfile /var/log/memcached.log
-m 1024
-p 11211
-u memcache
-c 1024
-l 127.0.0.1
# end of memcached template
```

```
$ vim templates/memcached.erb
# memcached template
-d
logfile /var/log/memcached.log
-m 1024
-p 11211
-u memcache
-c 1024
-l <%= @ipaddress %>
# end of memcached template
```

4. Vamos agora criar os manifests do nosso módulo para instalar e configurar o memcached de forma automática utilizando o puppet. No arquivo principal do Módulo **init.pp** vamos incluir outros arquivos que serão criados dentro do diretório manifests, utilizando dessa forma você consegue seperar as funções em arquivos menores para uma melhor manutenção e gerenciamento dos manifests.

4.1 Dentro do arquivo principal da classe vamos incluir os outros arquivos que fazem parte do projeto.

Dentro do arquivo **mainfests/init.pp** insira a estrutura a baixo e realize a configurações

descritas:

```
# Arquivo principal do módulo
class puppet-memcached {
     include puppet-memcached::install
     include puppet-memcached::service
     include puppet-memcached::files
}
```

4.2 No arquivo **files.pp** vamos colocar as tarefas que são responsável pelos arquivos de configuração.

Dentro do arquivo **mainfests/files.pp** insira a estrutura a baixo e realize a configurações descritas:

```
# Arquivo responsável pelo arquivos do serviço
class puppet-memcached::files inherits puppet-memcached {
     # Código AQUI
}
```

Nesse arquivo o seu desafio e escrever o mainfest do puppet para preencher os requisitos a baixo:

- Garantir o arquivo de configuração: **/etc/memcached.conf**
- O Dono e grupo do arquivo: **root**
- A permissão do arquivo de ser: **644**
- Deve ser utilizar o template criado no exercício anterior para esse arquivo de configuração.

  Segue a syntax para utilizar o template dentro do recurso (resource type) **file**:

  ```
  file {'</path/to/file>':
  ```

```
        content => template('<module_name>/<file_name>'),
    }
```

- Ele deve possui os metaparàmetros para garantir que pacote seja instalado antes da configuração do arquivo (metaparâmetro: **require**) e que serviço seja notificado toda vez que houver uma mudança nesse arquivo (metaparâmetro: **notify**).

```
$ vim manfiests/files.pp
class puppet-memcached::files inherits puppet-memcached {
   file {'/etc/memcached.conf':
     owner => root,
     group => root,
     mode => 0644,
     content => template('puppet-memcached/memcached.erb'),
     require => Package['memcached'],
     notify => Service['memcached'],
   }
}
```

4.3. No arquivo **service.pp** vamos colocar os resouces para gerenciar os serviços do serviço.

Dentro do arquivo **mainfests/service.pp** insira a estrutura a baixo e realize a configurações descritas:

```
# Arquivo responsável pelo gerenciamento de serviço do memcached
class puppet-memcached::service inherits puppet-memcached {
    # Código AQUI
}
```

Nesse arquivo o seu desafio e escrever o mainfest do puppet para preencher os requisitos abaixo:

- Garantir que serviço memcached esteja rodando na máquina
- O Recurso do serviço deve ser uma depêndencia do recurso do pacote utilizando o metaparâmetro: **require**.

```
$ vim manifests/service.pp
# manifests/service.pp
class puppet-memcached::service inherits puppet-memcached {

  service {'memcached':
     ensure => running,
     hasrestart => true,
     hasstatus => true,
     require => Package['memcached'],
  }
}
```

4.4. No arquivo **install.pp** vamos configurar recursos (resouces) que são responsável pela instalação dos pacotes

Dentro do arquivo mainfests/install.pp insira a estrutura a baixo e realize a configurações descritas:

```
# Arquivo responsável pelo instalação dos pacotes do serviço
class puppet-memcached::install inherits puppet-memcached {
     # Código AQUI
}
```

- Nesse arquivo o seu desafio e garantir a instalação do pacote memcached.

```
# manifests/install.pp
class puppet-memcached::install inherits puppet-memcached {

  package {'memcached':
    ensure => installed,
  }
}
```

Podemos perceber que os arquivos criado, todos possui uma herança da classe principal do módulo, como podemos ver na linha abaixo:

```
class puppet-memcached::install inherits puppet-memcached
```

sendo que a classe *puppet-memcached::install* é uma classe filha do puppet-memcached, que herda as informações da classe mãe reutilizando o código já configurado na classe mãe e segmentando os arquivos melhorando o gerenciamento das informações.

Para testar o nosso modulo utilize o comando do puppet

$ puppet apply -e “include puppet-memcached”

Lembrando que esse formato é o modo autonômo ou serverless.

Garantir que o container do Docker esteja sempre rodando no servidor

1. Dentro da máquina do Docker realize as seguintes configurações para preparar o nosso ambiente:

- Realize a instalação do agente do puppet na máquina Docker
- Configure o agente do puppet para utilizar o ambiente de homologação, e configure a opção master para utilizar o nome puppet.
- Altere a linha igual no exemplo a baixo dento do arquivo: **/etc/hosts:** para garantir que o nome *puppet*, que foi configurado como o servidor master, resolva para o endereço 192.168.200.50.

192.168.200.50 devops.dexter.com.br dexter **puppet**

```
# Instalação do agente do puppet
$ wget https://apt.puppetlabs.com/puppetlabs-release-precise.deb
$ dpkg -i puppetlabs-release-precise.deb
$ apt-update && apt-get install puppet

# Configurando a máquina Docker
$ vim /etc/puppet/puppet.conf
environment=homolog
master=puppet

# Adcionando o nome puppet para mapeamento internon do Linux.
$ vim /etc/hosts
192.168.200.50        devops.dexter.com.br devops puppet
```

2. Agora na máquina DevOps, crie o módulo do Puppet de nome: **docker-engine** dento do diretório **/etc/puppet/environments/homolog/modules/.** Utilizando o comando do

puppet.

$ cd /etc/puppet/environments/homolog/modules/

$ puppet module generate docker-engine

3. Dentro do manifest principal do modulo realiza as configurações descrita a baixo:

```
class docker-engine ($name, $ensure = "running") {
     # A FAZER
}
```

Esse manifests possui uma classe parametrizadas, com dois valores.

- **$name:** O nome do container a ser utilizado
- **$ensure:** Se classe deve garantir o container rodando ou parado. Valor padrão e “running”.

Dentro dessa manifests você deve escrever as configurações para iniciar o container do docker ou parar o container do docker de acordo com a variável $ensure passada.

O manifest dessa classe deve preencher o seguintes requisitos:

- Se a classe for chamada com o valor *ensure => 'running'* realizar o o comando docker start no container especificado, você pode utilizar o **if** para soluciona esse problema
- Se a classe for chamada com o valor *ensure => 'stopped'* realizar o comando docker stop no container especificado. você pode utilizar o metaparâmetro **onlyif** e utilizar o comando **docker inspect** para validar se o container existe para chegar a essa solução.

- O comando docker start e docker stop só pode ser rodado caso o container exista, você pode utilizar o metaparâmetro **onlyif** e utilizar o comando **docker inspect** para validar se o container existe para chegar a essa solução.
- Configure a **PATH** do resouce type (recurso) **Exec**, para evitar que o agente do puppet não encontre o comando.

```
$ vim /etc/puppet/environments/homolog/modules/docker-engine/

class docker-engine ($name, $ensure = "running") {

  # Start o container se o comando for executado com sucesso
  if ($ensure == 'running') {
     exec {'ensure_running':
        path => '/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin',
        command => "docker start ${name}",
        onlyif => "docker inspect ${name}",
        timeout => 300,
     }
  }

  # Caso parametro for passado com stopped
  if ($ensure == 'stopped') {
     exec {'ensure_stopped':
        path => '/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin',
        command => "docker stop ${name}",
       onlyif => "docker inspect ${name}",
       timeout => 300,
     }
   }
}
```

Para testar o nosso módulo realize a seguinte configuração

no arquivo: **/etc/puppet/environemnts/homolog/manifests/site.pp**

```
...
node 'docker.dexter.com.br' {
  # Ensure container is running on the master
  class {'docker-engine':
    name => 'puppet-hom',
    ensure => 'running'
  }
}
...
```

OBS: O container puppet-hom, foi criado na primeira parte desse desafio.

Na máquina docker execute o agente do puppet manualmente atráves do comando:

```
$ puppet agent -t
```

Após a configuração da tarefa acima o seu desafio estará completo.