Apostila Puppet Básico

http://instruct.com.br/apostila-puppet

Sumário

1 Licença de uso desta obra

A OBRA (CONFORME DEFINIDA ABAIXO) É DISPONIBILIZADA DE ACORDO COM OS TERMOS DESTA LICENÇA PÚBLICA CREATIVE COMMONS ("CCPL" OU "LICENÇA"). A OBRA É PROTEGIDA POR DIREITO AUTORAL E/OU OUTRAS LEIS APLICÁVEIS. QUALQUER USO DA OBRA QUE NÃO O AUTORIZADO SOB ESTA LICENÇA OU PELA LEGISLAÇÃO AUTORAL É PROIBIDO.

AO EXERCER QUAISQUER DOS DIREITOS À OBRA AQUI CONCEDIDOS, VOCÊ ACEITA E CONCORDA FICAR OBRIGADO NOS TERMOS DESTA LICENÇA. O LICENCIANTE CONCEDE A VOCÊ OS DIREITOS AQUI CONTIDOS EM CONTRAPARTIDA A SUA ACEITAÇÃO DESTES TERMOS E CONDIÇÕES.

1. **Definições**
   a. **"Obra Derivada"** significa uma Obra baseada na Obra ou na Obra e outras Obras pré-existentes, tal qual uma tradução, adaptação, arranjo musical ou outras alterações de uma Obra literária, artística ou científica, ou fonograma ou performance, incluindo adaptações cinematográficas ou qualquer outra forma na qual a Obra possa ser refeita, transformada ou adaptada, abrangendo qualquer forma reconhecível como derivada da original, com exceção da Obra que constitua uma Obra Coletiva, a qual não será considerada uma Obra Derivada para os propósitos desta Licença. Para evitar dúvidas, quando a Obra for uma Obra musical, performance ou fonograma, a sincronização da Obra em relação cronometrada com uma imagem em movimento (“synching”) será considerada uma Obra Derivada para os propósitos desta Licença.

   b. **"Obra Coletiva"** significa uma coleção de Obras literárias, artísticas ou científicas, tais quais enciclopédias e antologias, ou performances, fonogramas ou transmissões, ou outras Obras ou materiais não indicados na Seção 1(i) abaixo, que em razão da seleção e arranjo do seu conteúdo, constituam criações intelectuais nas quais a Obra é incluída na sua integridade em forma não-modificada, juntamente com uma ou mais contribuições, cada qual constituindo separada e independentemente uma Obra em si própria, que juntas são reunidas em um todo coletivo. A Obra que constituir uma Obra Coletiva não será considerada uma Obra Derivada (como definido acima) para os propósitos desta Licença.

   c. **"Licença Compatível Creative Commons"** significa uma licença que se encontre listada no site http://creativecommons.org/compatiblelicenses, aprovada pela Creative Commons como sendo essencialmente equivalente a esta Licença, incluindo ao menos: (i) termos que possuam a mesma finalidade, significado e efeito dos Elementos da Licença desta Licença e; (ii) permissão explícita para o relicenciamento de Obras Derivadas das Obras tornadas disponíveis sob aquela licença, sob esta Licença, uma licença Creative Commons Unported ou uma licença Creative Commons de outra jurisdição com os mesmos Elementos da Licença desta Licença.

   d. **"Distribuir"** significa colocar à disposição do público o original e cópias da Obra ou Obra Derivada, o que for apropriado, por meio de venda ou qualquer outra forma de transferência de propriedade ou posse.

   e. **"Elementos da Licença"** significam os principais atributos da licença correspondente, conforme escolhidos pelo Licenciante e indicados no título desta licença: Atribuição, Compartilhamento pela mesma licença.

   f. **"Licenciante"** significa a pessoa física ou jurídica que oferece a Obra sob os termos desta Licença.

   g. **"Autor Original"** significa, no caso de uma Obra literária, artística ou científica, o indivíduo ou indivíduos que criaram a Obra ou, se nenhum indivíduo puder ser identificado, a editora.

   h. **"Titular de Direitos Conexos"** significa (i) no caso de uma performance os atores, cantores, músicos, dançarinos, e outras pessoas que atuem, cantem, recitem, declamem, participem

em, interpretem ou façam performances de Obras literárias ou artísticas ou expressões de folclore (ii) no caso de um fonograma, o produtor, sendo este a pessoa ou entidade legal que primeiramente fixar os sons de uma performance ou outros sons; e (iii) no caso de radiodifusão, a empresa de radiodifusão.

i. **"Obra"** significa a Obra literária, artística e/ou científica oferecida sob os termos desta Licença, incluindo, sem limitação, qualquer produção nos domínios literário, artístico e científico, qualquer que seja o modo ou a forma de sua expressão, incluindo a forma digital, tal qual um livro, brochuras e outros escritos; uma conferência, alocução, sermão e outras Obras da mesma natureza; uma Obra dramática ou dramático-musical; uma Obra coreográfica ou pantomima; uma composição musical com ou sem palavras; uma Obra cinematográfica e as expressas por um processo análogo ao da cinematografia; uma Obra de desenho, pintura, arquitetura, escultura, gravura ou litografia; uma Obra fotográfica e as Obras expressas por um processo análogo ao da fotografia; uma Obra de arte aplicada; uma ilustração, mapa, plano, esboço ou Obra tridimensional relativa a geografia, topografia, arquitetura ou ciência; uma performance, transmissão ou fonograma, na medida em que tais Obras/direitos sejam reconhecidos e protegidos pela legislação aplicável; uma compilação de dados, na extensão em que ela seja protegida como uma Obra sujeita ao regime dos direitos autorais; ou uma Obra executada por artistas circenses ou de shows de variedade, conforme ela não for considerada uma Obra literária, artística ou científica.

j. **"Você"** significa a pessoa física ou jurídica exercendo direitos sob esta Licença, que não tenha previamente violado os termos desta Licença com relação à Obra, ou que tenha recebido permissão expressa do Licenciante para exercer direitos sob esta Licença apesar de uma violação prévia.

k. **"Executar Publicamente"** significa fazer a utilização pública da Obra e comunicar ao público a Obra, por qualquer meio ou processo, inclusive por meios com ou sem fio ou performances públicas digitais; disponibilizar ao público Obras de tal forma que membros do público possam acessar essas Obras de um local e em um local escolhido individualmente por eles; Executar a Obra para o público por qualquer meio ou processo e comunicar ao público performances da Obra, inclusive por performance pública digital; transmitir e retransmitir a Obra por quaisquer meios, inclusive sinais, sons ou imagens.

l. **"Reproduzir"** significa fazer cópias da Obra por qualquer meio, inclusive, sem qualquer limitação, por gravação sonora ou visual, e o direito de fixar e Reproduzir fixações da Obra, inclusive o armazenamento de uma performance protegida ou fonograma, em forma digital ou qualquer outro meio eletrônico.

2. **Limitações e exceções ao direito autoral e outros usos livres.** Nada nesta licença deve ser interpretado de modo a reduzir, limitar ou restringir qualquer uso permitido de direitos autorais ou direitos decorrentes de limitações e exceções estabelecidas em conexão com a proteção autoral, sob a legislação autoral ou outras leis aplicáveis.

3. **Concessão da Licença.** O Licenciante concede a Você uma licença de abrangência mundial, sem royalties, não-exclusiva, perpétua (pela duração do direito autoral aplicável), sujeita aos termos e condições desta Licença, para exercer os direitos sobre a Obra definidos abaixo:

   a. Reproduzir a Obra, incorporar a Obra em uma ou mais Obras Coletivas e Reproduzir a Obra quando incorporada em Obras Coletivas;

   b. Criar e Reproduzir Obras Derivadas, desde que qualquer Obra Derivada, inclusive qualquer tradução, em qualquer meio, adote razoáveis medidas para claramente indicar, demarcar ou de qualquer maneira identificar que mudanças foram feitas à Obra original. Uma tradução, por exemplo, poderia assinalar que “A Obra original foi traduzida do Inglês para o Português,” ou uma modificação poderia indicar que “A Obra original foi modificada”;

c. Distribuir e Executar Publicamente a Obra, incluindo as Obras incorporadas em Obras Coletivas; e,

d. Distribuir e Executar Publicamente Obras Derivadas.

e. O Licenciante renuncia ao direito de recolher royalties, seja individualmente ou, na hipótese de o Licenciante ser membro de uma sociedade de gestão coletiva de direitos (por exemplo, ECAD, ASCAP, BMI, SESAC), via essa sociedade, por qualquer exercício Seu sobre os direitos concedidos sob esta Licença.

Os direitos acima podem ser exercidos em todas as mídias e formatos, independente de serem conhecidos agora ou concebidos posteriormente. Os direitos acima incluem o direito de fazer as modificações que forem tecnicamente necessárias para exercer os direitos em outras mídias, meios e formatos. Todos os direitos não concedidos expressamente pelo Licenciante ficam ora reservados.

4. **Restrições.** A licença concedida na Seção 3 acima está expressamente sujeita e limitada pelas seguintes restrições:

a. Você pode Distribuir ou Executar Publicamente a Obra apenas sob os termos desta Licença, e Você deve incluir uma cópia desta Licença ou o Identificador Uniformizado de Recursos (Uniform Resource Identifier) para esta Licença em cada cópia da Obra que Você Distribuir ou Executar Publicamente. Você não poderá oferecer ou impor quaisquer termos sobre a Obra que restrinjam os termos desta Licença ou a habilidade do destinatário exercer os direitos a ele aqui concedidos sob os termos desta Licença. Você não pode sublicenciar a Obra. Você deverá manter intactas todas as informações que se referem a esta Licença e à exclusão de garantias em toda cópia da Obra que Você Distribuir ou Executar Publicamente. Quando Você Distribuir ou Executar Publicamente a Obra, Você não poderá impor qualquer medida tecnológica eficaz à Obra que restrinja a possibilidade do destinatário exercer os direitos concedidos a ele sob os termos desta Licença. Esta Seção 4(a) se aplica à Obra enquanto quando incorporada em uma Obra Coletiva, mas isto não requer que a Obra Coletiva, à parte da Obra em si, esteja sujeita aos termos desta Licença. Se Você criar uma Obra Coletiva, em havendo notificação de qualquer Licenciante, Você deve, na medida do razoável, remover da Obra Coletiva qualquer crédito, conforme estipulado na Seção 4(c), quando solicitado. Se Você criar uma Obra Derivada, em havendo aviso de qualquer Licenciante, Você deve, na medida do possível, retirar da Obra Derivada qualquer crédito conforme estipulado na Seção 4(c), de acordo com o solicitado.

b. Você pode Distribuir ou Executar Publicamente uma Obra Derivada somente sob: (i) os termos desta Licença; (ii) uma versão posterior desta Licença que contenha os mesmos Elementos da Licença; (iii) uma Licença Creative Commons Unported ou uma Licença Creative Commons de outra jurisdição (seja a versão atual ou uma versão posterior), desde que elas contenham os mesmos Elementos da Licença da presente Licença (p. ex., Atribuição-Compartilhamento pela mesma licença 3.0 Unported); (iv) uma Licença Compatível Creative Commons. Se Você licenciar a Obra Derivada sob os termos de uma das licenças a que se refere o item 4(iv), Você deve respeitar os termos daquela licença. Se Você licenciar a Obra Derivada sob os termos de qualquer das licenças mencionadas em 4(i) a 4(iii) (a “Licença Aplicável”), Você deve respeitar os termos da Licença Aplicável e o seguinte: (I) Você deve incluir uma cópia da Licença Aplicável ou o Identificador Uniformizado de Recursos (Uniform Resource Identifier) para aquela Licença em toda cópia de qualquer Obra Derivada que Você Distribuir ou Executar Publicamente; (II) Você não pode oferecer ou impor quaisquer termos sobre a Obra Derivada que restrinjam os termos da Licença Aplicável, ou a habilidade dos destinatários da Obra Derivada exercerem os direitos que lhes são garantidos sob os termos da Licença Aplicável; (III) Você deverá manter intactas todas as informações que se referirem à Licença Aplicável e à exclusão de garantia em toda cópia da Obra tal como incluída na Obra Derivada que Você Distribuir ou Executar Publicamente; (IV) Quando Você Distribuir ou Executar Publicamente uma Obra Derivada, Você não poderá impor

quaisquer medidas tecnológicas eficazes sobre a Obra Derivada que restrinjam a habilidade de uma pessoa que receba a Obra Derivada de Você em exercer os direitos a ela garantidos sob os termos da Licença Aplicável. Esta Seção 4(b) se aplica à Obra Derivada enquanto incorporada a uma Obra Coletiva, mas não exige que a Obra Coletiva, à parte da própria Obra Derivada, esteja sujeita aos termos da Licença Aplicável.

c. Se Você Distribuir ou Executar Publicamente a Obra ou qualquer Obra Derivada ou Obra Coletiva, Você deve, a menos que um pedido relacionado à Seção 4(a) tenha sido feito, manter intactas todas as informações relativas a direitos autorais sobre a Obra e exibir, de forma razoável em relação ao meio ou mídia por Você utilizado: (i) o nome do Autor Original (ou seu pseudônimo, se for o caso), se fornecido, do Titular de Direitos Conexos, se fornecido e quando aplicável, e/ou, ainda, se o Autor Original/Titular de Direitos Conexos e/ou o Licenciante designar outra parte ou partes (p. ex.: um instituto patrocinador, editora, periódico) para atribuição (“Partes de Atribuição”) nas informações relativas aos direitos autorais do Licenciante, termos de serviço ou por outros meios razoáveis, o nome dessa parte ou partes; (ii) o título da Obra, se fornecido; (iii) na medida do razoável, o Identificador Uniformizado de Recursos (URI) que o Licenciante especificar para estar associado à Obra, se houver, exceto se o URI não se referir ao aviso de direitos autorais ou à informação sobre o regime de licenciamento da Obra; e, (iv) em conformidade com a Seção 3(b), no caso de Obra Derivada, crédito identificando o uso da Obra na Obra Derivada (p. ex.: "Tradução Francesa da Obra do Autor Original/Titular de Direitos Conexos", ou "Roteiro baseado na Obra original do Autor Original/Titular de Direitos Conexos"). O crédito exigido por esta Seção 4(c), pode ser implementado de qualquer forma razoável; desde que, entretanto, no caso de uma Obra Derivada ou Obra Coletiva, na indicação de crédito feita aos autores participantes da Obra Derivada ou Obra Coletiva, o crédito apareça como parte dessa indicação, e de modo ao menos tão proeminente quanto os créditos para os outros autores participantes. De modo a evitar dúvidas, Você apenas poderá fazer uso do crédito exigido por esta Seção para o propósito de atribuição na forma estabelecida acima e, ao exercer Seus direitos sob esta Licença, Você não poderá implícita ou explicitamente afirmar ou sugerir qualquer vínculo, patrocínio ou apoio do Autor Original, Titular de Direitos Conexos, Licenciante e/ou Partes de Atribuição, o que for cabível, por Você ou Seu uso da Obra, sem a prévia e expressa autorização do Autor Original, Titular de Direitos Conexos, Licenciante e/ou Partes de Atribuição.

d. Na extensão em que reconhecidos e considerados indisponíveis pela legislação aplicável, direitos morais não são afetados por esta Licença.

5. **Declarações, garantias e exoneração**

EXCETO QUANDO FOR DE OUTRA FORMA MUTUAMENTE ACORDADO PELAS PARTES POR ESCRITO, O LICENCIANTE OFERECE A OBRA “NO ESTADO EM QUE SE ENCONTRA” (AS IS) E NÃO PRESTA QUAISQUER GARANTIAS OU DECLARAÇÕES DE QUALQUER ESPÉCIE RELATIVAS À OBRA, SEJAM ELAS EXPRESSAS OU IMPLÍCITAS, DECORRENTES DA LEI OU QUAISQUER OUTRAS, INCLUINDO, SEM LIMITAÇÃO, QUAISQUER GARANTIAS SOBRE A TITULARIDADE DA OBRA, ADEQUAÇÃO PARA QUAISQUER PROPÓSITOS, NÃO-VIOLAÇÃO DE DIREITOS, OU INEXISTÊNCIA DE QUAISQUER DEFEITOS LATENTES, ACURACIDADE, PRESENÇA OU AUSÊNCIA DE ERROS, SEJAM ELES APARENTES OU OCULTOS. EM JURISDIÇÕES QUE NÃO ACEITEM A EXCLUSÃO DE GARANTIAS IMPLÍCITAS, ESTAS EXCLUSÕES PODEM NÃO SE APLICAR A VOCÊ.

6. **Limitação de responsabilidade.**

EXCETO NA EXTENSÃO EXIGIDA PELA LEI APLICÁVEL, EM NENHUMA CIRCUNSTÂNCIA O LICENCIANTE SERÁ RESPONSÁVEL PARA COM VOCÊ POR QUAISQUER DANOS, ESPECIAIS, INCIDENTAIS, CONSEQÜENCIAIS, PUNITIVOS OU EXEMPLARES, ORIUNDOS DESTA LICENÇA OU DO USO DA OBRA, MESMO QUE O LICENCIANTE TENHA SIDO AVISADO SOBRE A POSSIBILIDADE DE TAIS DANOS.

7. **Terminação**

a. Esta Licença e os direitos aqui concedidos terminarão automaticamente no caso de qualquer violação dos termos desta Licença por Você. Pessoas físicas ou jurídicas que tenham recebido Obras Derivadas ou Obras Coletivas de Você sob esta Licença, entretanto, não terão suas licenças terminadas desde que tais pessoas físicas ou jurídicas permaneçam em total cumprimento com essas licenças. As Seções 1, 2, 5, 6, 7 e 8 subsistirão a qualquer terminação desta Licença.

b. Sujeito aos termos e condições dispostos acima, a licença aqui concedida é perpétua (pela duração do direito autoral aplicável à Obra). Não obstante o disposto acima, o Licenciante reserva-se o direito de difundir a Obra sob termos diferentes de licença ou de cessar a distribuição da Obra a qualquer momento; desde que, no entanto, quaisquer destas ações não sirvam como meio de retratação desta Licença (ou de qualquer outra licença que tenha sido concedida sob os termos desta Licença, ou que deva ser concedida sob os termos desta Licença) e esta Licença continuará válida e eficaz a não ser que seja terminada de acordo com o disposto acima.

8. **Outras disposições**

a. Cada vez que Você Distribuir ou Executar Publicamente a Obra ou uma Obra Coletiva, o Licenciante oferece ao destinatário uma licença da Obra nos mesmos termos e condições que a licença concedida a Você sob esta Licença.

b. Cada vez que Você Distribuir ou Executar Publicamente uma Obra Derivada, o Licenciante oferece ao destinatário uma licença à Obra original nos mesmos termos e condições que foram concedidos a Você sob esta Licença.

c. Se qualquer disposição desta Licença for tida como inválida ou não-executável sob a lei aplicável, isto não afetará a validade ou a possibilidade de execução do restante dos termos desta Licença e, sem a necessidade de qualquer ação adicional das partes deste acordo, tal disposição será reformada na mínima extensão necessária para tal disposição tornar-se válida e executável.

d. Nenhum termo ou disposição desta Licença será considerado renunciado e nenhuma violação será considerada consentida, a não ser que tal renúncia ou consentimento seja feito por escrito e assinado pela parte que será afetada por tal renúncia ou consentimento.

e. Esta Licença representa o acordo integral entre as partes com respeito à Obra aqui licenciada. Não há entendimentos, acordos ou declarações relativas à Obra que não estejam especificados aqui. O Licenciante não será obrigado por nenhuma disposição adicional que possa aparecer em quaisquer comunicações provenientes de Você. Esta Licença não pode ser modificada sem o mútuo acordo, por escrito, entre o Licenciante e Você.

2 Sobre a Instruct

A Instruct tem como especialidade soluções na área de automação e monitoramento de infraestrutura de rede, administração de sistemas e redes e capacitação para certificações. Temos grande experiência na implementação de automação de infraestrutura usando Puppet e possuímos os profissionais mais experientes do mercado, como Miguel Di Ciurcio Filho e José Augusto (Guto) Carvalho, primeiros certificados Puppet Professional (PCP) pela PuppetLabs da América Latina, além de ambos possuírem as certificações RHCE, LPIC-3, dentre outras.

Somos parceiros oficiais da PuppetLabs no Brasil, ministramos os treinamentos oficiais da PuppetLabs e fornecemos suporte e consultoria. Já possuímos clientes que usufruem da automação fornecida pelo Puppet em conjunto com os nossos serviços.

Combinada com a experiência dos profissionais da empresa, além dos serviços e soluções, a Instruct também fornece treinamentos voltados para certificações reconhecidas no mercado, como LPI e CompTIA.

Os exercícios e materiais de estudo são desenvolvidos pela própria empresa, o que demonstra o compromisso com a qualidade e a entrega de uma experiência diferenciada.

Versões mais novas e outras informações sobre essa apostila podem ser encontradas em http://instruct.com.br/apostila-puppet.

3 Instalação

Diversas distribuições empacotam o Puppet, mas as versões empacotadas e a qualidade desses pacotes variam muito, portanto a melhor maneira de instalá-lo é utilizando os pacotes oficiais da PuppetLabs. Os pacotes oficiais são extensivamente testatos e extremamente confiáveis.

Existem duas versões do Puppet distribuídas pela PuppetLabs: *Puppet Open Source* e o *Puppet Enterprise*. O Puppet Enterprise é distribuído gratuitamente para o gerenciamento de até 10 nodes, possui suporte oficial e vem acompanhado de uma versátil interface web para administração.

Para uma comparação mais detalhada sobre as diferenças entre a versão Open Source e a Enterprise, visite as páginas: https://puppetlabs.com/puppet/enterprise-and-open-source e https://puppetlabs.com/puppet/faq .

Aviso

Instalação a partir do código fonte

O Puppet é um projeto grande e complexo que possui muitas dependências, e instalá-lo a partir do código fonte não é recomendado. A própria PuppetLabs não recomenda a instalação a partir do código fonte. É muito mais confiável e conveniente utilizar pacotes já homologados e testados.

3.1 Debian e Ubuntu

1. Adicionando o repositório da PuppetLabs:

• Debian 8.x (Jessie)

```
# cd /tmp
# wget http://apt.puppetlabs.com/puppetlabs-release-pc1-jessie.deb
# dpkg -i  puppetlabs-release-pc1-jessie.deb
# apt-get update
```

• Ubuntu 14.04.x LTS (Trusty)

```
# cd /tmp
# wget http://apt.puppetlabs.com/puppetlabs-release-pc1-trusty.deb
# dpkg -i puppetlabs-release-pc1-trusty.deb
# apt-get update
```

Para instalar o repositório em outras versões do Debian ou Ubuntu, acesse a página http://apt.puppetlabs.com/ e baixe o pacote puppetlabs-release-pc1-SOBRENOME_DISTRO.deb. Por exemplo, o sobrenome do Debian 7 é Wheezy. Logo, o pacote seria puppetlabs-release-pc1-wheezy.deb.

2. Instale o pacote **puppet-agent**.

```
# apt-get -y install puppet-agent
# echo "PATH=/opt/puppetlabs/bin:$PATH" >> /etc/bash.bashrc
# echo "export PATH" >> /etc/bash.bashrc
# export PATH=/opt/puppetlabs/bin:$PATH
```

Dica

 **Turbinando o vim**

Para facilitar a edição de código, caso você utilize o editor vim, ative o plugin que adiciona o suporte a linguagem do Puppet executando os comandos abaixo e não deixe de adicionar a linha **syntax on** no seu `/home/name_user/.vimrc ou /root/.vimrc`.

```
# apt-get -y install vim vim-addon-manager vim-puppet
# vim-addons install puppet
```

3.2 Red Hat e CentOS

1. Adicionando o repositório da PuppetLabs:

   • Red Hat 7.x / CentOS 7.x

```
# yum install -y http://yum.puppetlabs.com/el/7/PC1/x86_64/puppetlabs-release-pc1-0.9.2-1.el7.noarch.rpm
```

Para instalar o repositório em outras versões do Red Hat ou CentOS, acesse a página http://yum.puppetlabs.com/el/ e localize o pacote adequado para a sua distro, para instalar conforme o exemplo mostrado acima.

2. Instale o pacote **puppet-agent**.

```
# yum -y install puppet-agent
# echo "PATH=/opt/puppetlabs/bin:$PATH" >> /etc/bashrc
# echo "export PATH" >> /etc/bashrc
# export PATH=/opt/puppetlabs/bin:$PATH
```

Dica

 **Turbinando o vim**

Para facilitar a edição de código, caso você utilize o editor vim, ative o plugin que adiciona o suporte a linguagem do Puppet executando os comandos abaixo e não deixe de adicionar a linha **syntax on** no seu `/home/name_user/.vimrc ou /root/.vimrc`.

```
# yum -y install vim vim-puppet
```

4 Resource Abstraction Layer (RAL)

O que existe em um sistema operacional? Arquivos, pacotes, processos e serviços em execução, programas, contas de usuários, grupos, etc. Para o Puppet, isso são *resources* (recursos).

Os resources têm certas similaridades entre si. Por exemplo, um arquivo tem um caminho e um dono, todo usuário possui um grupo e um número identificador. Essas características chamamos de *atributos*, e unindo os atributos que sempre estão presentes possibilita a criação de *resource types* (tipos de recursos).

Os atributos mais importantes de um *resource type* geralmente são conceitualmente idênticos em todos os sistemas operacionais, independentemente de como as implementações venham a diferir. A descrição de um resource pode ser separada de como ela é implementada.

Essa combinação de *resources*, *resource types* e atributos formam o *Resource Abstraction Layer* (RAL) do Puppet. O RAL divide resources em tipos (alto nível) e *providers* (provedores, implementações específicas de cada plataforma), e isso nos permite manipular resources (pacotes, usuários, arquivos, etc) de maneira independente de sistema operacional.

Dica

Ordens ao invés de passos

Através do RAL dizemos somente o que queremos e não precisamos nos preocupar no **como** será feito. Portanto, temos que pensar em ordens como "o pacote X deve estar instalado", ou ainda, "o serviço Z deve estar parado e desativado".

4.1 Manipulando resources via RAL

O comando `puppet resource` permite consultar e manipular o sistema operacional via RAL, visualizando a configuração na linguagem do Puppet.

Vamos manipular um pouco o RAL antes de escrevermos código.

4.1.1 Gerenciando usuários

O primeiro argumento que deve ser passado é o *resource type* que será consultado.

```
# puppet resource user
user { 'avahi':
  ensure           => 'present',
  comment          => 'Avahi mDNS daemon,,,',
  gid              => '108',
  home             => '/var/run/avahi-daemon',
  password         => '*',
  password_max_age => '99999',
  password_min_age => '0',
  shell            => '/bin/false',
  uid              => '105',
}
user { 'backup':
  ensure           => 'present',
```

```
  comment          => 'backup',
  gid              => '34',
  home             => '/var/backups',
  password         => '*',
  password_max_age => '99999',
  password_min_age => '0',
  shell            => '/bin/sh',
  uid              => '34',
}
...
```

A saída mostra todos os usuários, com atributos como UID, GID e shell já formatados na linguagem do Puppet que estejam presentes no sistema operacional.

Nós podemos ser mais específicos e consultar apenas um *resource*:

```
# puppet resource user root
user { 'root':
  ensure           => 'present',
  comment          => 'root',
  gid              => '0',
  home             => '/root',
  password         => '$6$.MuEgeA1$mL6sF0FnGUaTQWYBdqzwX3tGT2fgoCYYU...',
  password_max_age => '99999',
  password_min_age => '0',
  shell            => '/bin/bash',
  uid              => '0',
}
```

Esse código gerado pode ser utilizado depois, e é funcional.

É possível passarmos alguns atributos para o `puppet resource`, fazendo com que ele altere o estado de um recurso no sistema.

Tradicionalmente, para criarmos um usuário usamos comandos como `useradd` ou o interativo `adduser`. Ao invés de usar um desses comandos, vamos usar o Puppet:

```
# puppet resource user joe ensure=present home="/home/joe" managehome=true
notice: /User[joe]/ensure: created
user { 'joe':
  ensure => 'present',
  home   => '/home/joe',
}

# id joe
uid=500(joe) gid=500(joe) groups=500(joe)
```

Repare que a linha de comando não necessariamente lê código Puppet. Podemos usar somente argumentos.

4.1.2 Gerenciando serviços

Vamos continuar explorando mais *resources*. Outro *resource type* muito útil é o `service`.

```
# puppet resource service
service { 'acpid':
  ensure => 'running',
  enable => 'true',
}
service { 'auditd':
  ensure => 'running',
  enable => 'true',
}
service { 'crond':
  ensure => 'running',
  enable => 'true',
}
...
```

O comando acima listou todos os serviços da máquina e seus estados. Podemos manipular os serviços via Puppet, ao invés de utilizarmos os tradicionais comandos `update-rc.d` no Debian ou `chkconfig` no Red Hat. Além disso, também podemos parar e iniciar serviços.

Parando um serviço que está em execução:

```
# puppet resource service iptables ensure=stopped
notice: /Service[iptables]/ensure: ensure changed 'running' to 'stopped'
service { 'iptables':
  ensure => 'stopped',
}

# service iptables status
iptables is stopped
```

Inciando um serviço que estava parado:

```
# service saslauthd status
saslauthd is stopped

# puppet resource service saslauthd ensure=running
notice: /Service[saslauthd]/ensure: ensure changed 'stopped' to 'running'
service { 'saslauthd':
  ensure => 'running',
}

# service saslauthd status
iptables (pid  2731) is running...
```

4.1.3 Gerenciando pacotes

Além de usuários e serviços, podemos também manipular a instalação de software via RAL do Puppet.

Com um mesmo comando, podemos fazer a instalação, por exemplo, do `aide`, tanto no Debian quanto no CentOS. Vamos executar `puppet resource package aide ensure=installed` em ambos os sistemas.

- No CentOS:

```
# rpm -qi aide
package aide is not installed

# puppet resource package aide ensure=installed
notice: /Package[aide]/ensure: created
package { 'aide':
  ensure => '0.14-3.el6_2.2',
}

# rpm -qi aide
```

- No Debian:

```
# dpkg -s aide
Package `aide' is not installed and no info is available.
Use dpkg --info (= dpkg-deb --info) to examine archive files,
and dpkg --contents (= dpkg-deb --contents) to list their contents.

# puppet resource package aide ensure=installed
notice: /Package[aide]/ensure: ensure changed 'purged' to 'present'
package { 'aide':
  ensure => '0.15.1-2+squeeze1',
}

# dpkg -s aide
```

4.1.4 Principais Resource Types

O Puppet possui uma série de *resource types* prontos para uso, também chamados de *core resource types*, pois todos são distribuídos por padrão com o Puppet e estão disponíveis em qualquer instalação. Mais *resource types* podem ser adicionados usando módulos.

Os principais são:

- file
- package
- service
- user
- group
- cron
- exec

Podemos dizer também que esses tipos nos fornecem primitivas, com as quais podemos criar soluções de configuração completas e robustas.

4.1.5 Atributos de Resource Types

Até agora vimos atributos básicos dos tipos `user`, `service` e `package`. Porém, esses recursos possuem muito mais atributos do que vimos até agora.

Para sabermos os atributos de um tipo, o próprio comando `puppet` nos fornece documentação completa.

```
# puppet describe -s user

user
====
Manage users.  This type is mostly built to manage system
users, so it is lacking some features useful for managing normal
users.

This resource type uses the prescribed native tools for creating
groups and generally uses POSIX APIs for retrieving information
about them.  It does not directly modify `/etc/passwd` or anything.

**Autorequires:** If Puppet is managing the user's primary group (as
provided in the `gid` attribute), the user resource will autorequire
that group. If Puppet is managing any role accounts corresponding to the
user's roles, the user resource will autorequire those role accounts.


Parameters
----------
    allowdupe, attribute_membership, attributes, auth_membership, auths,
    comment, ensure, expiry, gid, groups, home, ia_load_module,
    key_membership, keys, managehome, membership, name, password,
    password_max_age, password_min_age, profile_membership, profiles,
    project, role_membership, roles, shell, system, uid

Providers
---------
    aix, directoryservice, hpuxuseradd, ldap, pw, user_role_add, useradd,
    windows_adsi
```

Pronto, agora temos uma lista de parâmetros sobre o tipo `user`.

> ***Dica***
>
>  **Documentação completa**
>
> O argumento `-s` mostra uma versão resumida da documentação. Use o comando `puppet describe` sem o `-s` para ter acesso à documentação completa do resource type.

4.2 Prática: Modificando recursos interativamente

Além de podermos manipular recursos em nosso sistema pelo comando puppet resource, ele fornece um parâmetro interessante: `--edit`. Com ele, podemos ter um contato direto com a linguagem do Puppet para manipular recursos, ao invés de usarmos apenas a linha de comando.

Vamos colocar o usuário **joe** aos grupos **adm** e **bin**. Normalmente faríamos isso usando o comando `usermod` ou editando manualmente o arquivo `/etc/group`. Vamos fazer isso no estilo Puppet!

1. Execute o seguinte comando:

```
# puppet resource user joe --edit
```

2. O Puppet abrirá o *vim* com o seguinte código:

```
user { 'joe':
  ensure           => 'present',
  gid              => '1004',
  home             => '/home/joe',
  password         => '!',
  password_max_age => '99999',
  password_min_age => '0',
  shell            => '/bin/bash',
  uid              => '1004',
}
```

3. Vamos acrescentar o seguinte código:

```
user { 'joe':
  ensure           => 'present',
  gid              => '1004',
  groups           => ['bin', 'adm'],  #<-- essa linha é nova!
  home             => '/home/joe',
  password         => '!',
  password_max_age => '99999',
  password_min_age => '0',
  shell            => '/bin/bash',
  uid              => '1004',
}
```

4. Basta sair do `vim`, salvando o arquivo, para que o Puppet aplique a nova configuração. Teremos uma saída parecida com essa:

```
info: Applying configuration version '1348039985'
notice: /Stage[main]//User[joe]/groups: groups changed  to 'adm,bin'
notice: Finished catalog run in 0.30 seconds
```

5 Manifests

As declarações de configuração são chamadas de *manifests* (manifestos) e são salvas em arquivos com a extensão `.pp`.

A principal utilidade da linguagem do Puppet é a declaração de *resources*, representando um estado desejado.

Nos manifests também podemos ter condições, agrupar resources, gerar texto com funções, utilizar e referenciar código em outros manifests e muitas outras coisas. Mas o principal é garantir que resources sejam declarados e gerenciados de maneira correta.

5.1 Declarando resources

Para manipular diversos aspectos de nosso sistema, devemos estar logados como `root`. Para melhor organização, vamos colocar nosso código em `/root/manifests`.

1. Vamos criar nosso primeiro resource, nesse caso, um arquivo:

```
# vim /root/manifests/arquivo-1.pp
file {'teste':
  path    => '/tmp/teste.txt',
  ensure  => present,
  mode    => 0640,
  content => "Conteudo de teste!\n",
}
```

2. Com o manifest criado, é hora de aplicá-lo ao sistema:

```
# puppet apply arquivo-1.pp
notice: /Stage[main]//File[teste]/ensure: created
notice: Finished catalog run in 0.08 seconds

# cat /tmp/teste.txt
Conteudo de teste!

# ls -l /tmp/teste.txt
-rw-r----- 1 root root 19 Sep 19 09:20 /tmp/teste.txt
```

- Temos um *resource type*, nesse caso, `file`, seguido por um par de colchetes que englobam todo o restante das informações sobre o resource.
- Dentro dos colchetes, temos:
  - O *título* do recurso seguido de dois pontos.
  - E uma sequência de `atributo => valor` que serve para descrever como deve ser o recurso. Vários valores devem ser separados por vírgula.

Além disso, algumas regras são fundamentais sobre a sintaxe:

- Esquecer de separar atributos usando a vírgula é um erro muito comum, tenha cuidado. O último par `atributo => valor` pode ser seguido de vírgula também.

- Letras maiúsculas e minúsculas fazem diferença! Na declaração de recursos, `File {'teste':...` significa outra coisa que veremos mais adiante.
- Colocar aspas nos valores faz diferença! Valores e palavras reservadas da linguagem, como `present`, não precisam estar entre aspas, apenas strings. Para o Puppet, tudo é string, mesmo números.
- Aspas simples são para valores literais e o único escape é para a própria aspa (`'`) e a barra invertida (\).
- Aspas duplas servem para interpolar variáveis e podem incluir um `\n` (nova linha).

Dica

Teste antes de executar

O Puppet fornece algumas funcionalidades que nos permitem testar o código antes de executá-lo.

Primeiramente podemos validar se existe algum erro de sintaxe, usando o comando `puppet parser validade arquivo.pp`.

O comando `puppet parser` apenas verifica se o manifest está correto.

Para simularmos as alterações que serão ou não feitas, usamos `puppet apply --noop`.

Mais exemplos:

```
# vim /root/manifests/arquivo-2.pp
file {'/tmp/teste1.txt':
  ensure  => present,
  content => "Ola!\n",
}

file {'/tmp/teste2':
 ensure => directory,
 mode   => 0644,
}

file {'/tmp/teste3.txt':
  ensure => link,
  target => '/tmp/teste1.txt',
}

notify {"Gerando uma notificação!":}

notify {"Outra notificação":}
```

E, finalmente, vamos aplicar:

```
# puppet apply arquivo-2.pp
notice: /Stage[main]//File[/tmp/teste1.txt]/ensure: created
notice: Outra notificação
notice: /Stage[main]//Notify[Outra notificação]/message: defined 'message' as \
```

```
          'Outra notificação'
notice: /Stage[main]//File[/tmp/teste3.txt]/ensure: created
notice: /Stage[main]//File[/tmp/teste2]/ensure: created
notice: Gerando uma notificação!
notice: /Stage[main]//Notify[Gerando uma notificação!]/message: defined 'message' \
          as 'Gerando uma notificação!'
notice: Finished catalog run in 0.03 seconds

# ls -la /tmp/teste*
-rw-r--r-- 1 root root    5 Sep 21 12:08 /tmp/teste1.txt
lrwxrwxrwx 1 root root   15 Sep 21 12:08 /tmp/teste3.txt -> /tmp/teste1.txt

/tmp/teste2:
total 8
drwxr-xr-x  2 root root 4096 Sep 21 12:08 .
drwxrwxrwt 24 root root 4096 Sep 21 12:08 ..

# cat /tmp/teste3.txt
Ola!
```

Repare que deixamos de fora alguns atributos, como `path`, e ainda assim tudo funcionou. Quase todos os *resourse types* possuem algum atributo que assume como valor padrão o título de *resource*. Para o *resource* `file`, é o atributo `path`. Para o recurso `notify`, é `message`. Em muitos outros casos, como `user`, `group`, `package` e outros, é simplesmente o atributo `name`.

No jargão do Puppet, o atributo que recebe como valor padrão o título de um recurso é chamado de `namevar`. Esse valor é sempre utilizado em um atributo que deve ser capaz de dar uma identidade ao recurso, que deve sempre ser único.

Utilizar o valor do título do *resource* é conveniente, mas algumas vezes pode ser desajeitado. Em certas ocasiões é melhor dar um título curto que simbolize e identifique o *resource* e atribuir um valor diretamente ao `namevar` como um atributo. Isso é prático principalmente se o nome de um recurso é longo.

```
notify {'grandenotificacao':
  message => "Essa é uma grande notificação! Ela é tão grande que é
              melhor utilizar um nome pequeno como título do resource.",
}
```

Não é possível declarar o mesmo *resource* mais de uma vez. O Puppet não permite que *resources* com o mesmo título sejam criados e, em vários casos, também não vai permitir que recursos diferentes tenham o mesmo valor de `namevar`.

```
# cat conflito.pp
file {'arquivo':
      path => '/tmp/arquivo.txt',
      ensure => present,
}

file {'outroarquivo':
      path => '/tmp/arquivo.txt',
      ensure => present,
}
```

```
# puppet apply conflito.pp
Cannot alias File[outroarquivo] to ["/tmp/arquivo.txt"] at \
      /root/manifests/conflito.pp:9; resource ["File", "/tmp/arquivo.txt"] \
      already declared at /root/manifests/conflito.pp:4
```

5.1.1 Observações sobre o resource file

Nós declaramos que `/tmp/teste2/` teria permissões 0644, porém, o `ls -lah` mostrou o comum `0755`. Isso acontece porque o Puppet ativa o bit de leitura e acesso de diretórios, pois isso é geralmente o que queremos. A ideia é que se possa gerenciar recursivamente arquivos em diretórios com permissão `0644` sem tornar os arquivos executáveis.

O tipo `file` tem diversos valores para o atributo `ensure`: `present`, `absent`, `file`, `directory` e `link`. Para saber mais, leia a referência do tipo `file`.

5.2 Prática: conhecendo os resources

> ***Dica***
>
>  Para essa atividade, salve o conteúdo de cada exercício em um arquivo `.pp` e aplique-o usando o comando `puppet apply`.

1. Crie uma entrada no arquivo `/etc/hosts`:

```
host { 'teste.puppet':
  ensure       => 'present',
  host_aliases => ['teste'],
  ip           => '192.168.56.99',
}
```

2. Crie um usuário chamado elvis com shell padrão `/bin/false` e grupo `adm`.

```
user {'elvis':
  shell => '/bin/false',
  gid => 'adm',
  home => '/home/elvis',
  managehome => true,
}
```

3. Crie um grupo chamado `super`:

```
group {'super':
  gid => 777,
}
```

6 Ordenação

Agora que entendemos manifests e declaração de recursos, vamos aprender sobre meta-parâmetros e ordenação de recursos.

Voltemos ao *manifest* anterior que criou um arquivo, um diretório e um link simbólico:

```
file {'/tmp/teste1.txt':
  ensure  => present,
  content => "Ola!\n",
}

file {'/tmp/teste2':
  ensure => directory,
  mode   => 0644,
}

file {'/tmp/teste3.txt':
  ensure => link,
  target => '/tmp/teste1.txt',
}

notify {"Gerando uma notificação!":}

notify {"Outra notificação":}
```

Embora as declarações estejam escritas uma após a outra, o Puppet pode aplicá-las em qualquer ordem.

Ao contrário de uma linguagem procedural, a ordem da escrita de recursos em um manifest não implica na mesma ordem lógica para a aplicação.

Alguns recursos dependem de outros recursos. Então, como dizer ao Puppet qual deve vir primeiro?

6.1 Meta-parâmetros e referência a recursos

```
file {'/tmp/teste1.txt':
  ensure  => present,
  content => "Ola!\n",
}

notify {'/tmp/teste1.txt foi sincronizado.':
  require => File['/tmp/teste1.txt'],
}
```

Cada *resource type* tem o seu próprio conjunto de atributos, mas existe outro conjunto de atributos, chamado meta-parâmetros, que pode ser utilizado em qualquer *resource*.

Meta-parâmetros não influenciem no estado final de um *resource*, apenas descrevem como o Puppet deve agir.

Nós temos quatro meta-parâmetros que nos permitem impor ordem aos *resources*: `before`, `require`, `notify` e `subscribe`. Todos eles aceitam um *resource reference* (referência a um recurso), ficando assim: `Tipo['titulo']`.

Nota

Maiúsculo ou minúsculo?

Lembre-se: usamos caixa baixa quando estamos declarando novos resources. Quando queremos referenciar um resource que já existe, usamos letra maiúscula na primeira letra do seu tipo, seguido do título do resource entre colchetes.

`file{'arquivo': }` é uma declaração e `File['arquivo']` é uma referência ao *resource* declarado.

6.1.1 Meta-parâmetros before e require

`before` e `require` criam um simples relacionamento entre resources, onde um resource deve ser sincronizado antes que outro.

`before` é utilizado no *resource* anterior, listando quais *resources* dependem dele.

`require` é usado no resource posterior, listando de quais resources ele depende.

Esses dois meta-parâmetros são apenas duas maneiras diferentes de escrever a mesma relação, dependendo apenas da sua preferência por um ou outro.

```
file {'/tmp/teste1.txt':
  ensure  => present,
  content => "Olah!\n",
  before  => Notify['mensagem'],
}

notify {'mensagem':
  message => 'O arquivo teste1.txt foi criado!',
}
```

No exemplo acima, após `/tmp/teste1.txt` ser criado acontece a notificação. O mesmo comportamento pode ser obtido usando o meta-parâmetro `require`:

```
file {'/tmp/teste1.txt':
  ensure  => present,
  content => "Olah!\n",
}

notify {'mensagem':
  require => File['/tmp/teste1.txt'],
  message => 'O arquivo teste1.txt foi criado!',
}
```

6.1.2 Meta-parâmetros notify e subscribe

Alguns tipos de resources podem ser *refreshed* (refrescados, recarregados), ou seja, devem reagir quando houver mudanças.

Por resource `service`, significa reiniciar ou recarregar após um arquivo de configuração modificado.

Para um resource `exec` significa ser executado toda vez que o resource seja modificado.

> ***Aviso***
>
> **Quando acontece um refresh?**
>
> *Refreshes* acontecem somente durante a aplicação da configuração pelo Puppet e nunca fora dele.
>
> O agente do Puppet não monitora alterações nos arquivos.

Os meta-parâmetros *notify* e *subscribe* estabelecem relações de dependência da mesma maneira que *before* e *require*, mas para relações de refresh.

Não só o *resource* anterior será sincronizado, como após a sincronização um evento `refresh` será gerado, e o *resource* deverá reagir de acordo.

> ***Nota***
>
> **Resources que suportam refresh**
>
> Somente os tipos built-in `exec`, `service` e `mount` podem ser *refreshed*.

No exemplo abaixo, toda vez que o arquivo `/etc/ssh/sshd_config` divergir de `/root/manifests/sshd_config`, ele será sincronizado. Caso isso ocorra, `Service['sshd']` receberá um refresh e fará com que o serviço `sshd` seja recarregado.

```
file { '/etc/ssh/sshd_config':
  ensure => file,
  mode   => 600,
  source => '/root/manifests/sshd_config',
  notify => Service['sshd'],
}

service { 'sshd':
  ensure     => running,
  enable     => true,
  hasrestart => true,
  hasstatus  => true,
}
```

6.1.3 Encadeando relacionamentos

Existe um outro jeito de declarar relacionamentos entre os resources: usando setas de ordenação `->` e notificação ~>. O Puppet chama isso de *channing*.

Essas setas podem apontar para qualquer direção (`<-` funciona também) e você deve pensar nelas como o fluxo do tempo. O resource de onde parte a seta é sincronizado antes que o recurso para qual a seta aponta.

O exemplo abaixo demonstra o mesmo efeito de ordenação, mas de maneira diferente. Para exemplos pequenos as vantagens de se usar setas podem não ser óbvias, mas com muitos *resources* envolvidos elas podem ser bem mais práticas.

```
file {'/tmp/teste1.txt':
  ensure  => present,
  content => "Hi.",
}

notify {'depois':
  message => '/tmp/teste1.txt foi sincronizado.',
}

File['/tmp/teste1.txt'] -> Notify['depois']
```

6.2 Prática: validando o arquivo `/etc/sudoers`

Para essa atividade, salve o conteúdo de cada exercício em um arquivo `.pp` e aplique-o usando o comando `puppet apply`.

1. Certifique-se de que o pacote `sudo` está instalado.

```
package {'sudo':
  ensure => 'installed'
}
```

2. Agora vamos declarar o controle do arquivo `/etc/sudoers` e usar como origem `/root/manifests/sudoers`. O arquivo depende do pacote, pois sem o pacote ele não deve existir.

```
package {'sudo':
  ensure => 'installed'
}

file {'/etc/sudoers':
  ensure => 'file',
  mode => 0440,
  owner => 'root',
  group => 'root',
  source => '/root/manifests/sudoers',
  require => Package['sudo']
}
```

3. Temos uma limitação, pois, caso exista algum erro no arquivo de origem, o arquivo, sempre será copiado para `/etc/sudoers`. Façamos uma verificação antes de o arquivo ser copiado.

    - Edite o arquivo `/root/manifests/sudoers` de forma a deixá-lo inválido antes de aplicar o *manifest* abaixo.

```
package {'sudo':
  ensure => 'installed'
}

file {'/etc/sudoers':
  ensure => 'file',
  mode => 0440,
  owner => 'root',
  group => 'root',
  source => '/root/manifests/sudoers',
  require => [Package['sudo'], Exec['parse_sudoers']],
}

exec {'parse_sudoers':
  command => '/usr/sbin/visudo -c -f /root/manifests/sudoers',
  require => Package['sudo'],
}
```

4. Ainda temos uma limitação. Toda vez que o *manifest* é aplicado, o resource `Exec['parse_sudoers']` é executado. Precisamos de uma condição para que ele só seja executado se necessário.

```
package {'sudo':
  ensure => 'installed'
}

file {'/etc/sudoers':
  ensure => 'file',
  mode => 0440,
  owner => 'root',
  group => 'root',
  source => '/root/manifests/sudoers',
  require => [Package['sudo'], Exec['parse_sudoers']],
}

exec {'parse_sudoers':
  command => '/usr/sbin/visudo -c -f /root/manifests/sudoers',
  unless => '/usr/bin/diff /root/manifests/sudoers /etc/sudoers',
  require => Package['sudo'],
}
```

7 Variáveis, fatos e condições

7.1 Variáveis

A linguagem declarativa do Puppet pode usar variáveis, como no exemplo abaixo:

```
$dns1 = '8.8.8.8'
$dns2 = '8.8.4.4'
$arquivo = '/etc/resolv.conf'
$conteudo = "nameserver ${dns1}\nnameserver ${dns2}"

file {$arquivo:
  ensure => 'present',
  content => $conteudo,
}
```

Mais alguns detalhes sobre variáveis:

- Variáveis começam com o símbolo de cifrão ($)
- Variáveis podem ser usadas para dar nomes em resources e em atributos
- Para interpolar variáveis, a string deve estar entre aspas duplas e a variável deve estar entre chaves: `${var}`
- Variáveis do topo do escopo (algo como global) podem ser acessadas assim: `$::variavel_global`
- Uma variável só pode ter seu valor atribuído uma vez

7.2 Fatos

Antes de gerar a configuração, o Puppet executa o `facter`.

O `facter` é uma ferramenta fundamental do ecossistema do Puppet, que gera uma lista de variáveis chamadas de fatos, que contém diversas informações sobre o sistema operacional.

Exemplo de saída da execução do comando `facter`:

```
# facter
architecture => amd64
augeasversion => 0.10.0
boardmanufacturer => Dell Inc.
boardserialnumber => .C75L6M1.
facterversion => 1.6.14
hardwareisa => unknown
hardwaremodel => x86_64
hostname => inspiron
id => root
interfaces => eth0,lo,vboxnet0
ipaddress => 192.168.56.1
ipaddress_lo => 127.0.0.1
ipaddress_vboxnet0 => 192.168.56.1
is_virtual => false
kernel => Linux
```

```
kernelmajversion => 3.2
kernelrelease => 3.2.0-0.bpo.3-amd64
kernelversion => 3.2.0
lsbdistcodename => squeeze
lsbdistdescription => Debian GNU/Linux 6.0.6 (squeeze)
lsbdistid => Debian
lsbdistrelease => 6.0.6
lsbmajdistrelease => 6
macaddress => 00:26:b9:25:76:ef
macaddress_eth0 => 00:26:b9:25:76:ef
macaddress_vboxnet0 => 0a:00:27:00:00:00
manufacturer => Dell Inc.
memoryfree => 1.70 GB
memorysize => 3.68 GB
memorytotal => 3.68 GB
netmask => 255.255.255.0
netmask_lo => 255.0.0.0
netmask_vboxnet0 => 255.255.255.0
network_lo => 127.0.0.0
network_vboxnet0 => 192.168.56.0
operatingsystem => Debian
operatingsystemrelease => 6.0.6
osfamily => Debian
path => /usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
physicalprocessorcount => 1
processor0 => Intel(R) Core(TM) i3 CPU       M 330  @ 2.13GHz
processor1 => Intel(R) Core(TM) i3 CPU       M 330  @ 2.13GHz
processor2 => Intel(R) Core(TM) i3 CPU       M 330  @ 2.13GHz
processor3 => Intel(R) Core(TM) i3 CPU       M 330  @ 2.13GHz
processorcount => 4
productname => Inspiron 1564
ps => ps -ef
puppetversion => 3.0.1
rubysitedir => /usr/local/lib/site_ruby/1.8
rubyversion => 1.8.7
selinux => false
serialnumber => C75L6M1
sshdsakey => AAAAB3NzaC1kc3MAAACBAJ6Lw5zcJfTkBm6Yp00a8X5XBkYLJtaf ...
sshrsakey => AAAAB3NzaC1yc2EAAAADAQABAAABAQDAErKQ92ShklNso4oBUNH6 ...
swapfree => 0.00 kB
swapsize => 0.00 kB
timezone => BRST
type => Portable
uniqueid => 007f0101
uptime => 34 days
uptime_days => 34
uptime_hours => 826
uptime_seconds => 2973926
virtual => physical
```

Todas essas variáveis estão disponíveis para uso dentro de qualquer manifest e dizemos que estão no escopo de topo (*top scope*).

O exemplo abaixo usa algumas das variáveis geradas pelo `facter`:

```
notify {'kernel':
  message => "O sistema operacional é ${kernel} e versão ${kernelversion}"
}

notify {'distro':
  message => "A distribuição é ${operatingsystem} e versão ${operatingsystemrelease}"
}
```

E teremos a seguinte saída:

```
# puppet apply a.pp
O sistema operacional é Linux e versão 2.6.18
/Stage[main]//Notify[kernel]/message: defined 'message' as 'Nosso sistema operacional \
             é Linux e versão 2.6.18'
A distribuição é CentOS e versão 5.8
/Stage[main]//Notify[distro]/message: defined 'message' as 'A distribuição é CentOS e \
             versão 5.8'
Finished catalog run in 0.05 seconds
```

Nota

Sistemas operacionais diferentes

Alguns fatos podem variar de um sistema operacional para outro. Além disso, é possível estender as variáveis do `facter`.

7.3 Condicionais

A linguagem declarativa do Puppet possui mecanismos de condição que funcionam de maneira parecida em relação às linguagens de programação. Mas existem algumas diferenças.

7.3.1 if

Exemplo de um bloco de condição `if`:

```
if expressao {
  bloco de codigo
}
elsif expressao {
  bloco de codigo
}
else {
  bloco de codigo
}
```

O `else` e o `elsif` são opcionais.

Outro exemplo, usando uma variável do `facter`:

```
if $is_virtual == 'true' {
  notify {'Estamos em uma maquina virtual': }
}
else {
  notify {'Estamos em uma maquina real': }
}
```

Os blocos podem conter qualquer qualquer tipo de definição de configuração, mais alguns exemplos:

```
if $osfamily == 'RedHat' {
  service {'sshd':
    ensure => 'running',
    enable => 'true',
  }
}
elsif $osfamily == 'Debian' {
  service {'ssh':
    ensure => 'running',
    enable => 'true',
  }
}
```

Aviso

 **True e False para o Puppet.**

Quando usamos variáveis que vêm do `facter`, sempre são strings.

Mesmo que seja retornado *false*, por exemplo, no fato $is_virtual, é diferente do tipo booleano `false`.

Portanto, um código como o abaixo sempre cairá no primeiro bloco, pois a variável é uma string.

```
if $is_virtual { ... } else { ... }
```

7.3.2 Expressões

7.3.2.1 Comparação

- == (igualdade, sendo que comparação de strings é **case-insensitive**)
- != (diferente)
- < (menor que)
- > (maior que)
- <= (menor ou igual)
- >= (maior ou igual)

- `=~` (casamento de regex)
- `!~` (não casamento de regex)
- `in` (contém, sendo que comparação de strings é **case-sensitive**)

Exemplo do operador `in`:

```
'bat' in 'batata' # TRUE
'Bat' in 'batata' # FALSE
'bat' in ['bat', 'ate', 'logo'] # TRUE
'bat' in { 'bat' => 'feira', 'ate' => 'fruta'} # TRUE
'bat' in { 'feira' => 'bat', 'fruta' => 'ate' } # FALSE
```

7.3.2.2 Operadores booleanos

- `and`
- `or`
- `!` (negação)

7.3.3 Case

Além do `if`, o Puppet fornece a diretiva `case`.

```
case $operatingsystem {
  centos: { $apache = "httpd" }
  redhat: { $apache = "httpd" }
  debian: { $apache = "apache2" }
  ubuntu: { $apache = "apache2" }
  # fail é uma função
  default: { fail("sistema operacional desconhecido") }
}
package {'apache':
  name   => $apache,
  ensure => 'latest',
}
```

Ao invés de testar uma única condição, o `case` testa a variável em diversos valores. O valor `default` é especial, e é auto-explicativo.

O `case` pode tentar casar com strings, expressões regulares ou uma lista de ambos.

O casamento de strings é *case-insensitive* como o operador de comparação ==.

Expressões regulares devem ser escritas entre barras e são *case sensitive*.

O exemplo anterior, reescrito:

```
case $operatingsystem {
  centos, redhat: { $apache = "httpd" }
  debian, ubuntu: { $apache = "apache2" }
  default: { fail("sistema operacional desconhecido") }
}
```

Exemplo usando uma expressão regular:

```
case $ipaddress_eth0 {
  /^127[\d.]+$/: {
    notify {'erro':
      message => "Configuração estranha!",
    }
  }
}
```

7.3.4 Selectors

Ao invés de escolher a partir de um bloco, um `selector` escolhe seu valor a partir de um grupo de valores. `Selectors` são usados para atribuir valor a variáveis.

```
$apache = $operatingsystem ? {
  centos          => 'httpd',
  redhat          => 'httpd',
  /Ubuntu|Debian/ => 'apache2',
  default         => undef,
}
```

O ponto de interrogação assinala `$operatingsystem` como o pivô do `selector`, e o valor final que é atribuído a `$apache` é determinado pelo valor corresponde de `$operatingsystem`.

Pode parecer um pouco estranho, mas há muitas situações em que é a forma mais concisa de se obter um valor.

7.4 Prática: melhor uso de variáveis

Reescreva o código do exemplo usando uma variável para armazenar o nome do serviço e usando somente um resource `service` no seu código.

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

if $osfamily == 'RedHat' {
  service {'ntpd':
    ensure => 'running',
    enable => 'true',
  }
}
elsif $osfamily == 'Debian' {
  service {'ntp':
    ensure => 'running',
    enable => 'true',
  }
}
```

8 Laboratório

O padrão mais comum na utilização do Puppet é a tríade pacote, arquivo de configuração e serviço.

Codifique um manifest que declare o seguinte:

1. O pacote `openssh-server` deve estar instalado.
2. O arquivo de configuração `/etc/ssh/sshd_config` depende do pacote `openssh-server` e tem como fonte o caminho `/root/manifests/sshd_config`.
    - Dica: faça uma cópia do arquivo para `/root/manifests`.
3. O serviço `sshd` deve ser recarregado quando o arquivo de configuração `sshd_config` for modificado.
4. Seu manifest deve tratar qual é o sistema operacional em relação ao serviço. Se for RedHat/CentOS o serviço será `sshd`, se for Debian/Ubuntu será apenas `ssh`.

9 Master / Agent

O Puppet é geralmente (mas nem sempre) usado como *master/agent*. O ciclo de operação nesses casos é o seguinte:

1. Os clientes (chamados de *node*) possuem um agente instalado que permanece em execução e se conecta ao servidor central (chamado tipicamente de *master*) periodicamente (a cada 30 minutos, por padrão).
2. O node solicita a sua configuração, que é compilada e enviada pelo master.
3. Essa configuração é chamada de catálogo.
4. O agente aplica o catálogo no node.
5. O resultado da aplicação do catálogo é reportado ao master opcionalmente, havendo divergências ou não.

Outra maneira comum de implantação do Puppet é a ausência de um agente em execução nos nodes. A aquisição e aplicação do catálogo é agendada na crontab.

9.1 Resolução de nomes

A configuração de nome e domínio do sistema operacional, além da resolução de nomes, é fundamental para o correto funcionamento do Puppet, devido ao uso de certificados SSL para a autenticação de agentes e o servidor master.

Para verificar a configuração de seu sistema, utilize o comando `hostname`. A saída desse comando nos mostra se o sistema está configurado corretamente.

```
# hostname
node1

# hostname --domain
puppet

# hostname --fqdn
node1.puppet
```

Dica

Configuração de hostname no Red Hat/Cent e Debian/Ubuntu

Para resolução de nomes, configurar corretamente o arquivo `/etc/resolv.conf` com os parâmetros `domain` e `search` com o domínio de sua rede.

O arquivo `/etc/hosts` deve possuir pelo menos o nome da própria máquina, com seu IP, FQDN e depois o hostname: `192.168.1.10 node1.puppet node1`.

No Debian, colocar apenas o hostname no arquivo `/etc/hostname`.

No CentOS, em `/etc/sysconfig/network`, ajuste o valor da variável `HOSTNAME`.

Para um bom funcionamento do Puppet é fundamental que sua rede possua resolução de nomes via DNS configurada. Hostname e domínio de cada sistema operacional devem resolver corretamente para seu respectivo IP, e o IP deve possuir o respectivo reverso.

9.2 Segurança e autenticação

As conexões entre agente e servidor master são realizadas usando o protocolo SSL e, através de certificados, ambos se validam. Assim, o agente sabe que está falando com o servidor correto e o servidor master sabe que está falando com um agente conhecido.

Um servidor master do Puppet é um CA (Certificate Authority) e implementa diversas funcionalidades como gerar, assinar, revogar e remover certificados para os agentes.

Os agentes precisam de um certificado assinado pelo master para receber o catálogo com as configurações.

Quando um agente e master são executados pela primeira vez, um certificado é gerado automaticamente pelo Puppet, usando o FQDN do sistema no certificado.

9.3 Prática Master/Agent

9.3.1 Instalação do master

1. O pacote `puppet-server` deverá estar instalado e certifique-se de que o hostname está correto.

```
# hostname --fqdn
master.puppet

# yum install puppet-server
```

Teremos a seguinte estrutura em `/etc/puppet`:

```
# tree -F --dirsfirst /etc/puppet/
/etc/puppet/
|-- manifests/
|-- modules/
|-- auth.conf
|-- fileserver.conf
`-- puppet.conf
```

- Sendo:
  - `auth.conf`: regras de acesso a API REST do Puppet.
  - `fileserver.conf`: Utilizado para servir arquivos que não estejam em módulos.
  - `manifests/`: Armazena a configuração que será compilada e servida para os agentes.
  - `modules/`: Armazena módulos com classes, arquivos, plugins e mais configurações para serem usadas nos manifests.
  - `puppet.conf`: Principal arquivo de configuração, tanto do master como do agente.

2. Iniciando o serviço:

```
# service puppetmaster start
```

Os logs, por padrão, são enviados para o syslog e estão disponíveis no arquivo `/var/log/messages`:

```
tail /var/log/messages
Nov 13 11:50:57 master puppet-master[2211]: Signed certificate request for ca
Nov 13 11:50:57 master puppet-master[2211]: Rebuilding inventory file
Nov 13 11:50:58 master puppet-master[2211]: master.puppet has a waiting certificate request
Nov 13 11:50:58 master puppet-master[2211]: Signed certificate request for master.puppet
Nov 13 11:50:58 master puppet-master[2211]: Removing file Puppet::SSL::CertificateRequest \
      master.puppet at '/var/lib/puppet/ssl/ca/requests/master.puppet.pem'
Nov 13 11:50:58 master puppet-master[2211]: Removing file Puppet::SSL::CertificateRequest \
      master.puppet at '/var/lib/puppet/ssl/certificate_requests/master.puppet.pem'
Nov 13 11:50:58 master puppet-master[2239]: Starting Puppet master version 3.0.1
Nov 13 11:50:58 master puppet-master[2239]: Reopening log files
```

9.3.2 Instalação do agente em node1

1. Certifique-se de que o nome e domínio do sistema estejam corretos e instale o pacote `puppet` na máquina node1:

```
# hostname --fqdn
node1.puppet

# yum install puppet
```

A estrutura do diretório `/etc/puppet` é a mesma do master, e os logs também são enviados via syslog e estão em `/var/log/messages`.

2. Em uma máquina em que o agente está instalado, precisamos configurá-la para que ele saiba quem é o master.

No arquivo `/etc/puppet/puppet.conf`, adicionar o parâmetro `server` na seção `[main]`.

```
# vim /etc/puppet/puppet.conf
[main]
    logdir = /var/log/puppet
    rundir = /var/run/puppet
    ssldir = $vardir/ssl
    # parâmetro server
    server = master.puppet
```

Nota

Conectividade

Certifique-se de que o servidor master na porta 8140 TCP está acessível para os nodes.

3. Conecte-se ao master e solicite assinatura de certificado:

```
# puppet agent -t
Info: Creating a new SSL key for node1.puppet
Info: Caching certificate for ca
Info: Creating a new SSL certificate request for node1.puppet
Info: Certificate Request fingerprint (SHA256): 6C:7E:E6:3E:EC:A4:15:56: ...
```

4. No servidor master aparecerá a solicitação de assinatura para a máquina node1.puppet. Assine-a

    • O comando abaixo deve ser executado em **master.puppet**.

```
master # puppet cert list
  "node1.puppet" (SHA256) 6C:7E:E6:3E:EC:A4:15:56:49:C3:1E:A5:E4:7F:58:B8: ...

master # puppet cert sign node1.puppet
Signed certificate request for node1.puppet
Removing file Puppet::SSL::CertificateRequest node1.puppet at \
      '/var/lib/puppet/ssl/ca/requests/node1.puppet.pem'
```

5. Execute o agente novamente e estaremos prontos para distribuir a configuração.

    • O comando abaixo deve ser executado em **node1.puppet**.

```
# puppet agent -t
Info: Caching certificate for node1.puppet
Info: Caching certificate_revocation_list for ca
Info: Retrieving plugin
Info: Caching catalog for node1.puppet
Info: Applying configuration version '1352824182'
Info: Creating state file /var/lib/puppet/state/state.yaml
Finished catalog run in 0.05 seconds
```

Dica

Possíveis problemas com certificados SSL

É importante que os horários do master e dos nodes não tenham grandes diferenças e estejam, de preferência, sincronizados. Conexões SSL confiam no relógio e, se estiverem incorretos, então sua conexão pode falhar com um erro indicando que os certificados não são confiáveis. Procure manter os relógios corretamente configurados utilizando NTP.

10 Nodes

O Puppet começa a compilação da configuração de um catálogo pelo arquivo `/etc/puppet/manifests/site.pp`. O `site.pp` é o ponto de entrada do master para identificar a configuração que será enviada a um agente.

Para saber qual configuração deve ser enviada a um agente, precisamos declarar o hostname do agente, utilizando a diretiva `node`. Diretivas `node` casam sempre com o nome do agente. Por padrão, o nome do agente é o valor de `certname` presente no certificado de um agente (por padrão, o FQDN).

10.1 Declarando nodes

Sintaxe para se declarar um node:

```
# /etc/puppet/manifests/site.pp

node 'node1.puppet' {
  package {'nano':
    ensure => 'present',
  }
}

node 'node2.puppet' {
  package {'vim':
    ensure => 'present',
  }
}
```

No exemplo acima, o agente que se identificar como `node1.puppet` receberá a ordem de instalar o pacote `nano`, enquanto `node2.puppet` deverá instalar o pacote `vim`.

É possível modularizar o arquivo `site.pp` usando a diretiva `import`.

```
# /etc/puppet/manifests/site.pp

# Importar os arquivos de /etc/puppet/manifests/nodes/
# Normalmente cada arquivo tem um node
import 'nodes/*.pp'

# Importar vários nodes somente de um arquivo
import 'nodes.pp'
```

Nota

Classificação de nodes

O Puppet fornece um recurso chamado *External Node Classifier* (ENC), que tem a finalidade de delegar o registro de nodes para uma entidade externa, evitando a configuração de longos manifests. Esse recurso será visto mais adiante.

10.2 Nomes

A diretiva *node* casa com agentes por nome. O nome de um node é um identificador único que por padrão é valor de **certname**, ou seja o FQDN.

É possível casar nomes de nodes usando expressões regulares:

```
# www1, www13, www999
node /^www\d+$/ {

}

# foo.dominio.com.br ou bar.dominio.com.br
node /^(foo|bar)\.dominio\.com\.br$/ {

}
```

Também podemos aproveitar uma configuração em comum usando uma lista de nomes na declaração de um node.

```
node 'www1.dominio.com.br', 'www2.dominio.com.br', 'www3.dominio.com.br' {

}
```

10.3 O node default

Caso o Puppet Master não encontre nenhuma declaração de `node` explícita para um agente, em última instância pode-se criar um node simplesmente chamado `default`, que casará apenas para os agentes que não encontraram uma definição de `node`.

```
node default {

}
```

10.4 Herança

É possível utilizar um mecanismo de herança para a declaração de nodes usando a diretiva `inherits` da seguinte maneira:

```
node 'base' {
  package {'nano':
    ensure => 'present',
  }
}

node 'www1.dominio.com.br' inherits 'base' {
  package {'vim':
    ensure => 'present',
  }
}
```

Aviso

 **Cuidados quanto ao uso de herança de nodes**

Apesar de aparentemente atrativo, recomenda-se evitar usar herança em nodes. Em caso de necessidade de refatorar seu código ou criar exceções para máquinas que irão divergir de um node pai, seu código ficará complicado e de difícil entendimento.

10.5 Prática

1. Declare a máquina **node1.puppet** no `site.pp` do master.
2. Declare o pacote `nano` como instalado para **node1.puppet**.
3. Execute `puppet agent -t` no node1, certifique-se de que o `nano` foi instalado.

Dica

 **Simulando a configuração**

Para simularmos as alterações que serão ou não feitas, usamos `puppet agent -t --noop`.

11 Classes e Módulos

Ao criar diversos resources para vários nodes, em algum momento passa a fazer sentido que certos resources que são relacionados estejam juntos, ou resources que sejam utilizados muitas vezes possam ser resumidos.

Muitas vezes, configurações específicas de um node precisam ser aproveitadas em outro e torna-se necessário copiar tudo para o outro node. Quando alterações são necessárias, é preciso realizar diversas modificações.

O Puppet fornece alguns mecanismos chamados *resource collections*, que são a aglutinação de vários *resources* para que possam ser utilizados em conjunto.

11.1 Classes

Deixe seu conhecimento sobre programação orientada a objetos de lado a partir de agora.

Para o Puppet, uma classe é a junção de vários resources sob um nome, uma unidade. É um bloco de código que pode ser ativado ou desativado.

Definindo uma classe:

```
class nomedaclasse {
...
}
```

Dentro do bloco de código da classe podemos colocar qualquer código Puppet, por exemplo:

```
# arquivo ntp.pp
class ntp {
  package { 'ntp':
    ensure => installed,
  }

  service { 'ntpd':
    ensure    => running,
    enable    => true,
    require => Package['ntp'],
  }
}
```

Vejamos o resultado ao aplicar esse código:

```
# puppet apply ntp.pp
Finished catalog run in 0.03 seconds
```

Simplesmente nada aconteceu, pois nós apenas definimos a classe. Para utilizá-la, precisamos declará-la.

```
# arquivo ntp.pp
class ntp {
  package { 'ntp':
    ensure => installed,
  }

  service { 'ntpd':
    ensure    => running,
    enable    => true,
    require => Package['ntp'],
  }
}

# declarando a classe
class {'ntp': }
```

Aplicando-a novamente:

```
# puppet apply --verbose ntp.pp
Info: Applying configuration version '1352909337'
/Stage[main]/Ntp/Package[ntp]/ensure: created
/Stage[main]/Ntp/Service[ntpd]/ensure: ensure changed 'stopped' to 'running'
Finished catalog run in 5.29 seconds
```

Portanto, primeiro definimos uma classe e depois a declaramos.

11.1.1 Diretiva include

Existe um outro método de usar uma classe, nesse caso usando a diretiva `include`.

```
# arquivo ntp.pp
class ntp {
...
}

# declarando a classe ntp usando include
include ntp
```

O resultado será o mesmo.

> ***Nota***
>
> **Declaração de classes sem usar include**
>
> A sintaxe `class {'ntp': }` é utilizada quando usamos classes que recebem parâmetros.

11.2 Módulos

Usando classes puramente não resolve nosso problema de repetição de código. O código da classe ainda está presente nos manifests.

Para solucionar esse problema, o Puppet possui o recurso de carregamento automático de módulos (*module autoloader*).

Primeiramente, devemos conhecer de nosso ambiente onde os módulos devem estar localizados. Para isso, verificamos o valor da opção de configuração `modulepath`.

```
# puppet config print modulepath
/etc/puppet/modules:/usr/share/puppet/modules
```

No Puppet, módulos são a união de um ou vários manifests que podem ser reutilizados. O Puppet carrega automaticamente os manifests dos módulos presentes em `modulepath` e os torna disponíveis.

11.2.1 Estrutura de um módulo

Como já podemos perceber, módulos são nada mais que arquivos e diretórios. Porém, eles precisam estar nos lugares corretos para que o Puppet os encontre.

Vamos olhar mais de perto o que há em cada diretório.

- `meu_modulo/`: diretório onde começa o módulo e dá nome ao mesmo
  - `manifests/`: contém todos os manifests do módulo
    - `init.pp`: contém definição de uma classe que deve ter o mesmo nome do módulo
    - `outra_classe.pp`: contém uma classe chamada meu_modulo::outra_classe
    - `um_diretorio/`: o nome do diretório afeta o nome das classes abaixo
      - `foo.pp`: contém uma classe chamada meu_modulo::um_diretorio::foo
      - `bar.pp`: contém uma classe chamada meu_modulo::um_diretorio::bar
  - `files/`: arquivos estáticos que podem ser baixados pelos agentes
  - `lib/`: plugins e fatos customizados implementados em Ruby
  - `templates/`: contém templates usadas no módulo
  - `tests/`: exemplos de como classes e tipos do módulo podem ser chamados

11.3 Prática: criando um módulo

1. Primeiramente, crie a estrutura básica de um módulo:

```
# cd /etc/puppet/modules
# mkdir -p ntp/manifests
```

2. O nome de nosso módulo é `ntp`. Todo módulo deve possuir um arquivo `init.pp`, e nele deve haver uma classe com o nome do módulo.

```
# vim /etc/puppet/modules/ntp/manifests/init.pp
class ntp {
  package { 'ntp':
    ensure => installed,
  }

  service { 'ntpd':
    ensure    => running,
    enable    => true,
    require => Package['ntp'],
  }
}
```

3. Deixe o código de `site.pp` dessa maneira:

```
# vim /etc/puppet/manifests/site.pp
node 'node1.puppet' {
  include ntp
}
```

4. Em **node1** aplique a configuração:

```
# puppet agent -t
```

5. Aplique a configuração no master também, dessa maneira:

```
# puppet apply -e 'include ntp'
```

Agora temos um módulo para configuração de NTP sempre a disposição!

11.4 Prática: arquivos de configuração em módulos

Além de conter manifests, módulos também podem servir arquivos. Para isso, faça os seguintes passos:

1. Crie um diretório `files` dentro do módulo `ntp`:

```
# pwd
/etc/puppet/modules
# mkdir -p ntp/files
```

2. Como aplicamos o módulo ntp no *master*, ele terá o arquivo `/etc/ntp.conf` disponível. Copie-o:

```
# cp /etc/ntp.conf /etc/puppet/modules/ntp/files/
```

3. Acrescente ao código da classe ntp em `/etc/puppet/modules/ntp/manifests/init.pp` um *resource type* `file`:

```
class ntp {

  ...

  file { 'ntp.conf':
    path     => '/etc/ntp.conf',
    require  => Package['ntp'],
    source   => "puppet:///modules/ntp/ntp.conf",
    notify   => Service['ntpd'],
  }

}
```

4. Faça qualquer alteração no arquivo `ntp.conf` do módulo (em `/etc/puppet/modules/ntp/files/ntp.conf`), por exemplo, acrescentando ou removendo um comentário.
5. Aplique a nova configuração no **node1**.

```
# puppet agent -t
```

Dica

Servidor de arquivos do Puppet

O Puppet pode servir arquivos dos módulos, e funciona da mesma maneira se você está operando de maneira serverless ou master/agente. Todos os arquivos no diretório `files` do módulo ntp estão disponíveis na URL `puppet:///modules/ntp/`.

12 Templates

Muitas vezes temos um mesmo serviço ativado em diversas máquinas, mas em um conjunto de máquinas esse serviço precisa ser configurado de uma maneira e, no restante das máquinas, de outra. Assim, cada conjunto de máquinas precisaria de um arquivo de configuração específico, mesmo que esse arquivo tivesse uma ou duas linhas de diferença.

Então, quando fosse necessário atualizar uma opção de configuração que é comum aos dois conjuntos de máquinas, seria necessário atualizar dois arquivos de configuração. Além do cuidado extra de garantir que ambos estivesses corretos.

O Puppet tem um recurso de templates, em que podemos usar somente um arquivo de dentro dele. Colocamos uma lógica e valores de variáveis que venham do seu código, tornando a nossa configuração mais robusta.

Vamos usar como exemplo um módulo chamado `foo`:

```
# pwd
/etc/puppet/modules

# tree foo
foo
|-- manifests
|   `-- init.pp
`-- templates
    `-- foo.conf.erb
```

Agora o conteúdo do arquivo `init.pp`:

```
# /etc/puppet/modules/foo/manifests/init.pp
class foo {
  $var1 = '123456'
  $var2 = 'bar'
  file {'/tmp/foo.conf':
    ensure => 'file',
    content => template('foo/foo.conf.erb')
  }
}
```

Até aqui nós usávamos o atributo *content* com uma string contendo o que queríamos dentro do arquivo, mas agora estamos usando a função `template()`, que processa o arquivo `foo.conf.erb`.

```
# /etc/puppet/modules/foo/templates/foo.conf.erb
var1=<%= var1 %>
var2=<%= var2 %>
<% if osfamily == 'RedHat' %>
var3=RedHat
<% else %>
var3=Outro
<% end %>
```

Repare que as variáveis do manifest estão disponíveis dentro da template, inclusive as variáveis do facter.

Nota

 **Localização de uma template no sistema de arquivos**

Note que o caminho que deve ser passado para a função `template()` deve conter o nome do módulo, seguido do nome do arquivo de template que usaremos. Portanto, `template('foo/foo.conf.erb')` significa abrir o arquivo `/etc/puppet/modules/foo/templates/foo.conf.erb`.

Usando o módulo `foo` em uma máquina CentOS:

```
# puppet apply -e 'include foo'
/Stage[main]/Foo/File[/tmp/foo.conf]/ensure: defined content as \
          '{md5}8612fd8d198746b72f6ac0b46d631a2c'
Finished catalog run in 0.05 seconds

# cat /tmp/foo.conf
var1=123456
var2=bar

var3=RedHat
```

Dica

 **Concatenando templates**

A função `template()` pode concatenar várias templates de uma vez só, possibilitando configurações sofisticadas.

```
template("foo/foo.conf-1.erb", "foo/foo.conf-2.erb")
```

12.1 Sintaxe ERB

Um arquivo de template no Puppet usa a sintaxe ERB, que é a linguagem padrão de templates do Ruby. Ela é simples e poderosa.

- Comentário:

```
<%# isso será ignorado %>
```

- Extrai o valor de uma variável:

```
<%= qualquer_variavel %>
```

- Condições:

```
<% if var != "foo" %>
<%= var %> is not foo!
<% end %>
```

- Verificar se uma variável existe:

```
<% if boardmanufacturer then %>
  Essa maquina é do fabricante type: <%= boardmanufacturer %>
<% end %>
```

- Iteração em um array chamado **bar**:

```
<% bar.each do |val| %>
   Valor: <%= val %>
<% end %>
```

Dica

Evitando linhas em branco

Repare que no exemplo do arquivo `/tmp/foo.conf` as linhas em que estavam as tags com o `if` e `end` acabaram saindo em branco no arquivo final.

Caso isso seja um problema, existem dois jeitos de resolvermos.

1. Colocar todo o código em apenas uma linha, assim o arquivo final não conterá linhas em branco:

`<% if osfamily == 'RedHat' %>var3=RedHat<% else %>var3=Outro<% end %>`,

2. A outra opção é colocar um hífen no final de cada tag, assim o ERB não retornará uma linha em branco:

`<% if osfamily == '!RedHat' -%>`

12.2 Prática: usando templates

1. Crie a estrutura básica de um módulo chamado `motd`:

```
# pwd
/etc/puppet/modules

# mkdir -p motd/{manifests,templates}
```

2. Defina a classe motd em `motd/manifests/init.pp`, conforme o código abaixo:

```
class motd {
  $admins = ['Joao j@foo.com', 'Edu e@foo.com', 'Bia b@foo.com']
  file {'/etc/motd':
    ensure  => 'file',
    mode    => 644,
    content => template("motd/motd.erb"),
  }
}
```

3. Crie a template em `motd/templates/motd.erb` com o conteúdo abaixo:

```
Bem vindo a <%= fqdn -%> - <%= operatingsystem -%> <%= operatingsystemrelease %>

Kernel: <%= kernel -%> <%= kernelversion %>

Em caso de problemas, falar com:
<% admins.each do |adm| -%>
<%= adm %>
<% end -%>
```

4. Use o módulo no **node1**, execute o agente e confira o resultado:

```
Bem vindo a node1.puppet - CentOS 6.4

Kernel: Linux 2.6.32

Em caso de problemas, falar com:
Joao j@foo.com
Edu e@foo.com
Bia b@foo.com
```

13 Histórico de mudanças

- Versão 1.0 lançada dia 20/03/2014