Apostila Puppet

Mantida pela comunidade Puppet-BR
https://github.com/puppet-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 é a autoridade em automação e Puppet no Brasil e é parceira e representante oficial da Companhia Puppet (antiga Puppet Labs) para suporte, licenciamento, treinamento e consultoria. A empresa tem como especialidade soluções na área de automação de infraestrutura e gerenciamento de configuração e possui equipe capacitada e certificada.

Possui os profissionais mais experientes do mercado, como Miguel Di Ciurcio Filho e José Augusto (Guto) Carvalho, primeiros certificados *Puppet Certified Professional* (PCP) pela Puppet na América Latina e também os únicos *Puppet Certified Instructor*.

Saiba mais sobre Puppet e a Instruct com o conteúdo do canal no YouTube e nas redes socias:

- https://youtube.com/instructbr
- https://twitter.com/InstructBR
- https://facebook.com/InstructBR

Caso esteja procurando serviços profissionais relacionados a automação e Puppet, entre em contato:

- Email: http://instruct.com.br/contato/
- Escritório em **São Paulo/SP**: Av. Marquês de São Vicente, 1619 - 18º Andar, Barra Funda. Telefone: **11 3230-6506**
- Escritório em **Brasília/DF**: QS 1, Rua 210, LT 34 E 36, Ed. LED, TORRE 2, SN, Sala 1209, Areal (Águas Claras). Telefone: **61 4042-2250**

2.1 Sobre a Apostila

Esta apostila foi criada pela Instruct em 2014 e cedida para a comunidade Puppet-BR em Jun/2016.

Versões mais recentes e instruções de como contribuir com esta apostila podem ser encontradas em: https://github.com/puppet-br/apostila-puppet

3 Introdução ao Gerenciamento de Configuração

3.1 Os desafios no gerenciamento de infraestrutura e serviços

A quantidade de servidores e serviços computacionais existentes nos mais variados setores de atividades têm crescido constantemente na última década, e não há sinais de que vá diminuir. Administrar, configurar, atualizar e documentar esses crescentes ambientes computacionais torna-se cada vez mais desafiador. E, por isso, garantir que dezenas, ou até mesmo centenas de sistemas estejam configurados corretamente torna-se cada vez mais inviável.

As alterações necessárias na configuração de um grande parque de servidores, sejam para melhorias ou para correção de problemas são maçantes, consomem muito tempo e podem causar mais problemas do que solucioná-los, devido a erros operacionais e muitas vezes também a inviabilidade de revertê-los.

Diante desse cenário, é fundamental utilizar uma ferramenta robusta, que não utilize scripts que apenas executam comandos sequencialmente. É importante também que essa ferramenta certifique-se da corretude da configuração do ambiente, sem se basear em documentação que pode estar desatualizada ou, muitas vezes, incorreta.

3.1.1 O que é gerenciamento de configuração?

Grande parte de problemas de disponibilidade em serviços de missão crítica na área de TI são causados por falhas humanas e gestão de processos. Parte significativa desses problemas poderia ser evitada se fosse feita uma melhor coordenação na gerência de mudanças e de configuração.

Muitas organizações de TI utilizam processos manuais, scripts customizados, imagens de sistema e outras técnicas rudimentares para execução de tarefas repetitivas. Em grandes ambientes ou nos que possuem diversas equipes, esses métodos rudimentares não são escaláveis, são difíceis de manter e podem causar diversos problemas, como: divergências entre a configuração de fato e a documentação, não cumprimento de normas e perda de produtividade e agilidade.

Diante disso, gerenciamento de configuração é o processo de absorção controlada de mudanças, padronizando a aplicação de configurações em infraestrutura de TI de uma forma automatizada e ágil, sendo fundamental para o sucesso de diversos processos, como provisionamento de novos sistemas, aplicação de patches e cumprimento de normas de segurança.

3.2 Questões sobre a cultura estabelecida na administração de sistemas

Diversos setores de Tecnologia da Informação têm uma visão relativamente negativa dos Administradores de Sistemas ("System Administrator" ou simplesmente "SysAdmin") e profissionais que trabalham em atividades correlatas. Esses profissionais geralmente são sobrecarregados e estão sob constante pressão, pois as organizações precisam que suas redes, sistemas e serviços estejam sempre disponíveis.

A visão negativa do SysAdmin tem um certo fundamento, pois existem algumas características nele que são muito comuns.

- Aversão a mudanças

Por ter que garantir que tudo esteja sempre funcionando, o SysAdmin é muito conservador quando analisa qualquer tipo de mudança no ambiente, por menor que seja. Ele sabe que, muitas vezes, alterações a princípio simples e inocentes podem causar sérias consequências. Os mais precavidos não fazem nenhuma alteração antes de se montar um plano para que, em caso de qualquer imprevisto, mesmo que manualmente, as alterações possam ser revertidas.

- Lentidão na entrega de novos serviços

Por estar sempre sob pressão para garantir a disponibilidade e qualidade dos serviços que já estão na rede, novos serviços ou a atualização dos já em funcionamento geralmente não são prioridade. As empresas, muitas vezes por questões de certificação com padrões de qualidade, cobram do SysAdmin uma extensa carga de atividades extras na entrega de um novo serviço, como documentação, treinamento de usuários, operadores e etc.

- Compartilhamento de informações é tabu

Em organizações onde existe mais de um SysAdmin, é muito comum encontrar rachas dentro da equipe do tipo: Fulano cuida das máquinas A, B e C, Ciclano cuida das máquinas X, Y e Z e Beltrano de M, N e O. E nenhum deles se ajuda ou sabe claramente o quê ou como o outro configurou. E, pior ainda, quando Fulano precisa intervir nas máquinas de Ciclano, sobram críticas do tipo "Lógico que deu problema, olha como ele fez!" ou "Onde está a documentação disso?" e, ainda por cima, "A documentação dele está toda errada!".

- Permissões de acesso são vistas como troféu

Nada como ter plenos poderes. Para muitos SysAdmins, infelizmente, conseguir acesso a servidores que anteriormente ele não tinha é uma conquista. Mais por questões sociais do que técnicas. Afinal, muitos encaram isso como a derrota do colega que anteriormente tinha acesso exclusivo a um determinado servidor e agora perdeu esse "privilégio".

As organizações fazem o que podem para solucionar esses problemas, mas não é uma tarefa fácil. Muitos acreditam que o principal método para solucionar essas questões é a criação de uma boa documentação sobre o ambiente. Porém, o próprio SysAdmin geralmente é extremamente cético sobre a eficácia dele documentar suas atividades, configurações e planos.

Das diversas soluções que existem para documentação, talvez a menos pior seja a equipe utilizar uma wiki. Utilizando qualquer wiki, de graça vem o versionamento e mecanismo de busca. Porém, mesmo utilizando a wiki, ao longo do tempo a documentação apresentará diversas limitações.

- Validação da documentação.

Como testar se todos os comandos e arquivos registrados na wiki estão corretos? Provavelmente não houve má fé do SysAdmin quando ele registrou **o como** se chegou ao estado de um servidor com Apache, PHP e PostgreSQL, mas basta que apenas um dos passos seja deixado de lado para que ninguém mais consiga repetir com sucesso a instalação e configuração. O que um gerente deve fazer? Ordenar que outro SysAdmin valide tudo o que o colega fez? Não é nada divertido de se fazer, mas muitas vezes é um mal necessário.

- A documentação ficará desatualizada.

É natural que, ao longo do dia-a-dia da área de TI, emergências surjam e sejam necessárias intervenções imediatas no ambiente. Ajustes como limites de conexões, espaço alocado, etc. A questão é, depois da emergência, alguém se lembrará de atualizar a documentação com as mudanças? Provavelmente não.

Dentro do ambiente de servidores, geralmente as equipes não utilizam nenhum tipo de controle mais rígido no acesso e controle de alterações na configuração. Certamente, quando feitas, essas perguntas são difíceis de serem respondidas:

- Quando foi configurado?
- Quem configurou?
- Mudou por quê?
- Quem mandou?

Uma ferramenta de sistema de chamados combinada com muita disciplina da equipe podem ajudar na resposta dessas perguntas, mas os reflexos dessa burocracia no SysAdmin fazem com que ele se torne menos motivado. Além do mais, alterações incorretas em configurações que estavam funcionando são piores do que deixar de configurar, já que antes funcionava. E, ainda por cima, como garantir que tudo estará configurado corretamente semana que vem?

O resultado disso tudo é que, apenas com o uso desses mecanismos, manter documentação consistente ao longo do tempo requer um grande esforço e disciplina, tornando o sucesso do trabalho do SysAdmin muito improvável.

O ciclo vicioso de necessidade constante de manutenção e documentação leva um imenso gasto de tempo que deveria ser dedicado a atividades mais nobres e estimulantes para o trabalho. E, sem dúvida, quanto mais aplicações, serviços, sistemas, máquinas e etc, maior é o risco de problemas.

3.3 Limitações das soluções comuns de automação

É aceito de praxe pelo mercado que um bom SysAdmin é aquele que faz suas próprias ferramentas e automatiza ao máximo seu trabalho.

Por outro lado, as práticas da maioria dos SysAdmins, até as dos mais experientes, têm certos limites de viabilidade em ambientes mais complexos ou com equipes multidisciplinares. É comum copiar arquivos de configuração de uma máquina "que funciona" para uma máquina nova. Muitas vezes, quando são necessárias alterações em muitas máquinas, fazer SSH em um laço com uma lista de IPs ou nomes de máquinas também é comum.

O que para muitos é magia negra, para o SysAdmin é a solução: shell script. Longos scripts com comandos encadeados e conectados com pipes e saídas filtradas com expressões regulares, que resultam na entrada de outro comando que ordena e depois corta e recorta a saída final.

Shell Script é excelente para atividades repetitivas e cotidianas, mas bom apenas para soluções *ad-hoc* e pontuais. É de difícil leitura, principalmente para quem não é o autor. É comum SysAdmins que descartam todos os scripts de um antecessor na empresa, pois é mais fácil fazer um novo do que tentar entender o que já existe.

Além disso, quando estamos desenvolvendo um Shell Script, nos desprendemos de diversas amarras das linguagens de programação. Mas isso tem um preço. Sejamos honestos, scripts sempre são:

- Protegidos quanto à concorrência? Você está checando se tem acesso exclusivo à máquina ou aos arquivos e diretórios que está manipulando?
- Testáveis? É possível simular a execução de um script?
- Reversíveis? Os comandos dados pelo script podem ser revertidos naturalmente?
- Legíveis? Não existem convenções para estilo.
- Geram bons logs? É possível saber o que está acontecendo, em diferentes níveis de prioridade?
- Portáveis? O mesmo script funciona em todos os sistemas do seu ambiente? (Você tem muita sorte se todos os seus sistemas são idênticos)

Diante de todos esses problemas, é necessário evoluir. Para isso, precisamos quebrar paradigmas.

4 O que é o Puppet

Puppet é uma ferramenta e **plataforma Open Source** para automação e gerenciamento de configuração de servidores e infraestrutura. Foi desenvolvido para ajudar a comunidade SysAdmin na construção e compartilhamento de ferramentas maduras e que evitem a duplicação de esforços na solução de problemas repetidos. O Puppet é distribuído sob a Licença Apache 2.0.

- Possui grande biblioteca com funcionalidades prontas para uso, fornecendo uma plataforma poderosa para simplificar as tarefas executadas por SysAdmins.
- É composto de uma linguagem de configuração declarativa utilizada para expressar a configuração do sistema. O trabalho do SysAdmin é escrito como código na linguagem do Puppet, que é compartilhável tal como qualquer outro código e é possível escrever avançados aplicativos de automação de maneira simples, com apenas algumas linhas de código.
- Utiliza uma arquitetura cliente/servidor para a distribuição da configuração para clientes, que possuem um agente que sempre valida e corrige quaisquer problemas encontrados.
- Extensível via módulos e plugins, permite adicionar funcionalidades sob demanda e compartilhar suas soluções com outros SysAdmins, tornando seu trabalho mais ágil.

Tradicionalmente, a gestão das configurações de um grande conjunto de computadores é feita de práticas imperativas e comandos sequenciais, ou seja, simplesmente executando comandos via SSH em um loop. Essa abordagem simples de executar comandos sequencialmente melhorou ao longo do tempo, mas ainda carrega fundamentais limitações, como vistas no capítulo anterior.

O Puppet tem uma abordagem diferente: cada sistema recebe um catálogo de *resources* (recursos) e relacionamentos, compara com o estado atual do sistema e faz as alterações necessárias para colocar o sistema em conformidade com o catálogo.

Os benefícios dessa metodologia vão além de apenas resolver questões de divergência de configuração de máquinas: modelar sistemas como dados permite ao Puppet:

- Simular mudanças de configuração.
- Acompanhar o histórico de um sistema através de seu ciclo de vida.
- Provar que um código refatorado ainda produz o mesmo estado do sistema.

4.1 Como o Puppet funciona?

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

1. Os clientes (chamados de nó, ou *node*) possuem um agente instalado que permanece em execução e se conecta a um servidor central (chamado tipicamente de *master*) periodicamente (a cada 30 minutos, por padrão).
2. O node solicita a sua configuração, que é copilada e enviada pelo master.
3. Essa configuração "compilada" é chamada de catálogo.
4. O agente aplica o catálogo no node.
5. O resultado da aplicação do catálogo é reportado ao master, 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` ou é disparada via Mcollective. Geralmente, a configuração do ambiente codificada na linguagem do Puppet está armazenada em um sistema de controle de versão, como Subversion ou Git.

As funcionalidades do Puppet são separadas basicamente em três camadas, sendo cada uma responsável por uma parte do sistema como um todo.

4.1.1 Linguagem de configuração

O fato de possuir uma linguagem própria de configuração é um dos maiores diferenciais do Puppet em relação a outras soluções. Isso pode ser ao mesmo tempo um ponto forte ou um ponto fraco, vai depender do que está sendo procurado.

Esse questionamento é natural. Afinal, por que não usar algum formato de arquivo ou dados já existente, como XML ou YAML? Ou ainda, por que não usar Ruby como linguagem, já que o Puppet é desenvolvido em Ruby?

A linguagem declarativa de configuração do Puppet é como se fosse a interface com um humano. XML e YAML, sendo formatos desenvolvidos para intercâmbio de dados e para facilitar o processamento por computadores, não são boas interfaces para humanos. Algumas pessoas podem ter facilidade para ler e escrever nesses formatos, mas há uma razão por que usamos os navegadores ao invés de apenas ler documentos HTML diretamente. Além disso, usar XML ou YAML limitaria qualquer garantia de que a interface fosse declarativa, pois um processo poderia tratar uma configuração XML diferentemente de outro.

4.1.2 Camada de transação

A camada de transação é o motor do Puppet. Nela é realizada a configuração de cada node, seguindo essas etapas:

1. Interpretar e compilar a configuração
2. Enviar ao agente a configuração compilada.
3. Aplicar a configuração no node.
4. Reportar os resultados para o master.

O Puppet analisa a sua configuração e calcula como aplicá-la no agente. Para isso, é criado um grafo que contém todos os resources e suas relações uns com os outros. Isso permite ao Puppet decidir a melhor ordem para aplicação da configuração com base em relacionamentos criados pelo SysAdmin.

Os resources são compilados no que chamamos de catálogo, que é enviado aos nodes e aplicado pelos agentes, que devolvem ao master um relatório sobre o que foi feito. Isso não faz o Puppet ser totalmente transacional, como um tradicional banco de dados onde alterações podem ser revertidas. Porém, é possível modelar sua configuração com um modo "noop" (*no operation*, sem operação), onde é possível testar a execução de sua configuração sem realmente aplicá-la.

Uma das consequências do modo de operação do Puppet é a idempotência, ou seja, as configurações podem ser aplicadas repetidas vezes de maneira segura. O Puppet vai alterar somente o que está em divergência com o declarado.

4.1.3 Camada de abstração de recursos

A *Resource Abstraction Layer (RAL)* do Puppet é o que garante a simplificação das dores de cabeça de um SysAdmin por ter que lidar com diversos sistemas operacionais diferentes. Nomes, argumentos e localização de comandos, formatos de arquivos, controle de serviços e outras tantas diferenças que existem, e muitas vezes não fazem sentido, são abstraídas na linguagem de configuração do Puppet.

Para cada resource, existe um *provider* (provedor). O provider é responsável pelo "como" fazer para gerenciar um resource, por exemplo, a instalação de pacotes. Uma vez que você declara que quer o pacote `sudo` instalado, não será mais necessário se preocupar se a plataforma utiliza `apt` ou `yum`.

5 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 Companhia Puppet (antiga Puppet Labs, empresa que mantém o software). Os pacotes oficiais são extensivamente testatos e extremamente confiáveis.

Existem duas versões do Puppet distribuídas pela Puppet: *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 abaixo:

- https://puppet.com/product/puppet-enterprise-and-open-source-puppet
- https://puppet.com/product/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 Puppet não recomenda a instalação a partir do código fonte. É muito mais confiável e conveniente utilizar pacotes já homologados e testados.

Dica

Turbinando o Vim

Utilize os plugins disponíveis em https://github.com/rodjek/vim-puppet para facilitar a edição de código no Vim.

5.1 Debian e Ubuntu

1. Adicione o repositório da Puppet:

- Debian 8 (Jessie)

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

- Ubuntu 14.04 LTS (Trusty)

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

- Ubuntu 16.04 LTS (Xenial)

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

Você também pode acessar a página http://apt.puppetlabs.com e localizar o pacote adequado para outras versões do Debian ou Ubuntu.

2. Instale o pacote `puppet-agent`:

```
sudo apt-get -y install puppet-agent
```

3. Torne os comandos do pacote `puppet-agent` disponíveis no *path* do sistema:

```
sudo export PATH=/opt/puppetlabs/bin:$PATH
sudo echo "PATH=/opt/puppetlabs/bin:$PATH" >> /etc/bash.bashrc
sudo echo "export PATH" >> /etc/bash.bashrc
```

5.2 CentOS e Red Hat

1. Adicione o repositório da Puppet:

- CentOS/Red Hat 6

```
sudo yum install -y https://yum.puppetlabs.com/puppetlabs-release-pc1-el-6.noarch.rpm
```

- CentOS/Red Hat 7

```
sudo yum install -y http://yum.puppetlabs.com/puppetlabs-release-pc1-el-7.noarch.rpm
```

Você também pode acessar a página http://yum.puppetlabs.com e localizar o pacote adequado de outras versões e distribuições da família Red Hat.

2. Instale o pacote `puppet-agent`:

```
sudo yum -y install puppet-agent
```

3. Torne os comandos do pacote `puppet-agent` disponíveis no *path* do sistema:

```
sudo export PATH=/opt/puppetlabs/bin:$PATH
sudo echo "PATH=/opt/puppetlabs/bin:$PATH" >> /etc/bashrc
sudo echo "export PATH" >> /etc/bashrc
```

4. Obtenha a versão do puppet-agent

```
puppet --version
```

6 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".

6.1 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.

6.1.1 Gerenciando usuários

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

```
sudo puppet resource user

user { 'avahi':
  ensure  => 'present',
  comment => 'Avahi mDNS daemon,,,',
  gid     => '108',
  home    => '/var/run/avahi-daemon',
  shell   => '/bin/false',
  uid     => '105',
}
user { 'backup':
  ensure  => 'present',
  comment => 'backup',
  gid     => '34',
```

```
  home    => '/var/backups',
  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*:

```
sudo puppet resource user root

user { 'root':
  ensure  => 'present',
  comment => 'root',
  gid     => '0',
  home    => '/root',
  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:

```
sudo puppet resource user joe ensure=present home="/home/joe" managehome=true

Notice: /User[joe]/ensure: created
user { 'joe':
  ensure => 'present',
  home   => '/home/joe',
}
```

Verifique se o usuário `joe` foi criado.

```
sudo 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.

6.1.2 Gerenciando serviços

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

```
sudo 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/Ubuntu ou `chkconfig` no CentOS/Red Hat. Além disso, também podemos parar e iniciar serviços.

Parando um serviço que está em execução:

```
sudo puppet resource service iptables ensure=stopped

Notice: /Service[iptables]/ensure: ensure changed 'running' to 'stopped'
service { 'iptables':
  ensure => 'stopped',
}

sudo service iptables status

iptables is stopped
```

Inciando um serviço que estava parado:

```
sudo service saslauthd status

saslauthd is stopped

sudo puppet resource service saslauthd ensure=running

Notice: /Service[saslauthd]/ensure: ensure changed 'stopped' to 'running'
service { 'saslauthd':
  ensure => 'running',
}

sudo service saslauthd status
```

```
iptables (pid  2731) is running...
```

6.1.3 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:

```
sudo rpm -qi aide

package aide is not installed

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

sudo rpm -qi aide
```

- No Debian:

```
sudo 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.

sudo puppet resource package aide ensure=installed

Notice: /Package[aide]/ensure: created
package { 'aide':
  ensure => '0.16~a2.git20130520-3',
}

sudo dpkg -s aide
```

6.1.4 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.

6.1.5 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.

```
sudo 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, forcelocal, gid, groups, home, ia_load_module,
    iterations, key_membership, keys, loginclass, managehome, membership,
    name, password, password_max_age, password_min_age, profile_membership,
    profiles, project, purge_ssh_keys, role_membership, roles, salt, shell,
    system, uid

Providers
---------
    aix, directoryservice, hpuxuseradd, ldap, openbsd, 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.

6.2 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 adicionar 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:

```
sudo 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 '1447253347'
Notice: /Stage[main]/Main/User[joe]/groups: groups changed '' to
['adm', 'bin']
Notice: Applied catalog in 0.07 seconds
```

5. Remova o usuário `joe` com o comando a seguir.

```
sudo puppet resource user joe ensure=absent
```

6. Remova o diretório `/home/joe` com o comando a seguir.

```
sudo puppet resource file /home/joe ensure=absent force=true
```

7. Instale a última versão do `nmap`.

```
sudo puppet resource package nmap ensure=latest
```

8. Crie o grupo de usuário `teste`.

```
sudo puppet resource group teste ensure=present
```

9. Verifique o status do serviço `ssh`.

```
sudo puppet resource service ssh
```

10. Crie o arquivo `/tmp/teste` com o conteúdo "isso é um teste".

```
sudo puppet resource file /tmp/teste.txt ensure=file content='isso eh um teste'
```

11. Execute um comando de `ping` para o Google.com e direcione a saída do comando para um arquivo `/tmp/ping.txt`.

```
sudo puppet resource exec 'ping -c3 google.com > /tmp/ping.txt' \
  path='/bin:/usr/bin'
```

7 Manifests

As declarações de configuração são chamadas de *manifests* (manifestos) e são armazenadas 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.

7.1 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:

```
sudo mkdir /root/manifests
sudo 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:

```
sudo puppet apply /root/manifests/arquivo-1.pp

Notice: /Stage[main]/Main/File[teste]/ensure: \
  defined content as '{md5}14c8346a185a9b0dd3f44c22248340f7'
Notice: Applied catalog in 0.05 seconds
```

```
sudo cat /tmp/teste.txt

Conteudo de teste!
```

```
sudo ls -l /tmp/teste.txt
-rw-r----- 1 root root 19 Nov 11 13:21 /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.
- 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 validate arquivo.pp`.

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

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

Mais exemplos:

```
sudo 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:

```
sudo puppet apply /root/manifests/arquivo-2.pp

Notice: /Stage[main]/Main/File[/tmp/teste1.txt]/ensure: \
  defined content as '{md5}50c32e08ab3f0df064af1a8c98d1b6ce'
Notice: /Stage[main]/Main/File[/tmp/teste2]/ensure: created
Notice: /Stage[main]/Main/File[/tmp/teste3.txt]/ensure: created
Notice: Gerando uma notificação!
Notice: /Stage[main]/Main/Notify[Gerando uma notificação!]/message: \
  defined 'message' as 'Gerando uma notificação!'
Notice: Outra notificação
Notice: /Stage[main]/Main/Notify[Outra notificação]/message: \
  defined 'message' as 'Outra notificação'
Notice: Applied catalog in 0.05 seconds
```

```
sudo ls -la /tmp/teste*

-rw-r--r-- 1 root root    5 Nov 11 13:28 /tmp/teste1.txt
lrwxrwxrwx 1 root root   15 Nov 11 13:28 /tmp/teste3.txt -> /tmp/teste1.txt
-rw-r----- 1 root root   19 Nov 11 13:21 /tmp/teste.txt

/tmp/teste2:
total 8
drwxr-xr-x 2 root root 4096 Nov 11 13:28 .
drwxrwxrwt 8 root root 4096 Nov 11 13:28 ..
```

```
sudo 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`.

```
sudo vim /root/manifests/conflito.pp

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

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

```
sudo puppet apply /root/manifests/conflito.pp

Error: Evaluation Error: Error while evaluating a Resource Statement, \
   Cannot alias File[outroarquivo] to ["/tmp/arquivo.txt"] at \
   /root/manifests/conflito.pp:6; resource ["File", "/tmp/arquivo.txt"] \
   already declared at /root/manifests/conflito.pp:1 at
   /root/manifests/conflito.pp:6:3
```

7.1.1 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`. São eles: `present`, `absent`, `file`, `directory` e `link`. Para saber mais leia a referência do tipo `file`.

```
sudo puppet describe file
```

7.2 Prática: conhecendo os resources

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/sh` e grupo `adm`:

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

3. Crie um grupo chamado `super`:

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

4. Desfaça as alterações anteriores:

```
file{ 'remove_dir':
  path   => '/home/elvis',
  ensure => absent,
  force  => true,
}

user{ 'remove_user':
  name   => elvis,
  ensure => absent,
}

group{ 'remove_group':
  name   => super,
  ensure => absent,
}
```

8 Ordenação

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

Voltemos ao *manifest* `/root/manifests/arquivo-2.pp`, 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?

8.1 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 influenciam 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 de recurso e `File['arquivo']` é uma referência ao *resource* declarado.

8.1.1 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 => "Ola!\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 => "Ola!\n",
}

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

8.1.2 Meta-parâmetros notify e subscribe

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

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

Para um resource `exec`, significa ser executado toda vez que o resource for 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 será gerado um evento `refresh` 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   => '0600',
  source => '/root/manifests/sshd_config',
  notify => Service['sshd'],
}

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

8.1.3 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']
```

Aviso

Meta-parâmetros

Mais informações sobre outros meta-parâmetros podem ser encontradas em: https://docs.puppet.com/puppet/latest/metaparameter.html

8.2 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. Crie um manifest com o código abaixo e execute-o.

```
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 `sudo`, pois sem ele o arquivo 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.

- Copie o arquivo `/etc/sudoers` para `/root/manifests/sudoers`. 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'],
}
```

Ao executar esse manifest, o arquivo `/etc/sudoers` não será atualizado porque há um problema de validação de conteúdo do arquivo de origem `/root/manifests/sudoers`.

Nota

Atributos onlyif e unless do resource exec

Quando o recurso `exec` possuir o atributo `onlyif` ou `unless` declarado, só será executado se o(s) comando(s) informado(s) nestes atributos for(em) executado(s) sem erros. Ou seja, se retornarem o código `0` (zero). Veja mais informações em: http://www.puppetcookbook.com/posts/exec-onlyif.html e https://docs.puppet.com/puppet/latest/types/exec.html

9 Variáveis, fatos e condições

9.1 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 ($).
- Variáveis podem ser usadas para dar nomes em resources e atributos.
- Para interpolar variáveis, a string deve estar entre aspas duplas e a variável deve estar entre chaves: `${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.

> ***Nota***
>
> **Tipos de dados**
>
> As variáveis podem suportar vários tipos de dados, que podem ser: strings, arrays, números, tipo nulo, boleanos e hashes.
>
> Para obter mais informações sobre cada tipo de dados, acesse a página abaixo. https://docs.puppet.com/puppet/latest/lang_data.html

9.2 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

dmi => {
```

```
  bios => {
    release_date => "12/01/2006",
    vendor => "innotek GmbH",
    version => "VirtualBox"
  },
  board => {
    manufacturer => "Oracle Corporation",
    product => "VirtualBox",
    serial_number => "0"
  },
  manufacturer => "innotek GmbH",
  product => {
    name => "VirtualBox",
    serial_number => "0",
    uuid => "95E9353C-948D-4D93-A004-536701C2C673"
  }
}
memory => {
  swap => {
    available => "880.00 MiB",
    available_bytes => 922742784,
    capacity => "0%",
    total => "880.00 MiB",
    total_bytes => 922742784,
    used => "0 bytes",
    used_bytes => 0
  },
  system => {
    available => "670.50 MiB",
    available_bytes => 703074304,
    capacity => "11.04%",
    total => "753.68 MiB",
    total_bytes => 790290432,
    used => "83.18 MiB",
    used_bytes => 87216128
  }
}
os => {
  architecture => "i386",
  family => "Debian",
  hardware => "i686",
  name => "Debian",
  release => {
    full => "8.2",
    major => "8",
    minor => "2"
  },
  selinux => {
    enabled => false
  }
}
system_uptime => {
```

```
  days => 0,
  hours => 2,
  seconds => 8416,
  uptime => "2:20 hours"
}
timezone => BRST
virtual => virtualbox
```

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

Veja que é possível extrair fatos específicos:

```
facter ipaddress
facter ipaddress_eth0
facter mountpoints
facter mountpoints./
facter mountpoints./.available_byte
facter os
facter os.distro
facter os.distro.release
facter os.distro.release.full
```

É possível extrair os fatos em formatos como YAML e JSON.

```
facter --json

facter --yaml
```

9.2.1 Prática: facter

O manifest `facter.pp` a seguir usa algumas das variáveis geradas pelo `facter`:

```
#Obtendo o nome do S.O e a versao do kernel
#Esses dados estao nos fatos: kernel e kernelversion
notify {'kernel':
  message => "O sistema operacional eh ${::kernel} versao ${::kernelversion}."
}

#Obtendo o nome da distro GNU/Linux
#Esse dados estao nos fatos: operatingsystem e operatingsystemrelease
notify {'distro':
  message => "A distribuicao GNU/Linux eh ${::operatingsystem}
    versão ${::operatingsystemrelease}."
}
```

```
#Alguns sysadmins criam o '/home' em particao separada do '/'
#Vamos testar o fato: mountpoints['/home'],
#Se existir essa particao, vamos obter o espaco livre, contido
#no fato mountpoints['/home']['available_bytes']
if $::mountpoints['/home'] {
  $free_space = $::mountpoints['/home']['available_bytes']
}

#Se entrar no elsif, eh porque o '/home' esta na mesma particao do '/'
#Vamos testar o fato: mountpoints['/'],
#E vamos obter o espaco livre, contido
#no fato mountpoints['/']['available_bytes']
elsif $::mountpoints['/'] {
  $free_space = $::mountpoints['/']['available_bytes']
}

#O espaco requerido eh de 2GB ou 2000000 bytes
$space_required = 2000000

#Testando se ha o espaco requerido em '/home'
if $free_space > $space_required {
  notify{ 'info_free_space':
   message => "[OK] Ha espaco livre suficiente em /home. Espaco requerido: \
     ${space_required} bytes, espaco livre: ${free_space} bytes",
  }
}
else {
  notify{ 'info_free_space':
    message => "[ERRO] Espaco insuficiente em /home. Espaco requerido: \
     ${space_required} bytes, espaco livre: ${free_space} bytes"
  }
}
```

E teremos a seguinte saída:

```
puppet apply facter.pp

Notice: Compiled catalog for node1 in environment production in 0.09 seconds \
Notice: O sistema operacional eh Linux versao 4.4.0.
Notice: /Stage[main]/Main/Notify[kernel]/message: defined 'message' as \
  'O sistema operacional eh Linux versao 4.4.0.'
Notice: A distribuicao GNU/Linux eh Ubuntu versão 16.04.
Notice: /Stage[main]/Main/Notify[distro]/message: defined 'message' as \
  'A distribuicao GNU/Linux eh Ubuntu versão 16.04.'
Notice: [ERRO] Espaco insuficiente em /home. Espaco requerido: 200000 bytes, \
  espaco livre: -211103744 bytes
Notice: /Stage[main]/Main/Notify[info_free_space]/message: defined 'message' as \
  '[ERRO] Espaco insuficiente em /home. Espaco requerido: 200000 bytes, espaco \
  livre: -211103744 bytes'
Notice: Applied catalog in 0.04 seconds
```

Nota

 **Sistemas operacionais diferentes**

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

https://docs.puppet.com/facter/latest/core_facts.html
https://docs.puppet.com/puppet/latest/lang_facts_and_builtin_vars.html
https://docs.puppet.com/facter/latest/fact_overview.html
https://docs.puppet.com/facter/latest/custom_facts.html

9.3 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.

9.3.1 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 tipo de definição de configuração. Veja mais um exemplo:

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

9.3.2 Expressões

9.3.2.1 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-insensitive**)

Exemplo do operador `in`:

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

9.3.2.2 Operadores booleanos

- `and`
- `or`
- `!` (negação)

9.3.3 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 pode ser reescrito assim:

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

Exemplo usando uma expressão regular:

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

9.3.4 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 correspondente de `$::operatingsystem`.

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

9.4 Prática: melhor uso de variáveis

1) Reescreva o código do exemplo a seguir 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',
  }
}
```

10 Funções e Lambda

10.1 Funções

As funções são plugins escritos em Ruby, que você pode chamar durante a compilação de um catálogo. Uma chamada para qualquer função é uma expressão que resolve um valor.

A maioria das funções recebe um ou mais valores como argumentos e, em seguida, retorna algum outro valor resultante. O código Ruby na função pode fazer diversas coisas para produzir o valor resultante, tais como: avaliação de modelos, cálculos matemáticos e busca por valores em uma fonte externa.

As funções também podem:

- causar efeitos colaterais que modificam o catálogo;
- avaliar um bloco de código Puppet, possivelmente usando os argumentos da função para modificar esse código ou controlar como ele é executado.

A linguagem Puppet inclui várias funções internas, e muitas estão disponíveis em módulos no Puppet Forge, que será mostrado no capítulo sobre Puppet Forge. Um destes módulos é o `puppetlabs-stdlib` (https://forge.puppet.com/puppetlabs/stdlib). Você também pode escrever funções personalizadas e colocá-las em seus próprios módulos.

10.1.1 Localização

Como qualquer expressão, uma chamada de função pode ser usada em qualquer lugar no qual seja permitido retornar o valor resultante.

As chamadas de função também podem ficar por conta própria, o que fará com que o valor resultante seja ignorado. (Quaisquer efeitos colaterais ainda ocorrerão.)

10.1.2 Sintaxe

A maioria das funções tem nomes curtos, contendo apenas uma palavra. No entanto, a moderna API de função do Puppet também permite nomes de funções qualificadas como: `mymodule::foo`.

As funções devem sempre ser chamadas pelos seus nomes completos. Você não pode encurtar um nome de uma função qualificada.

Há duas maneiras de chamar funções na linguagem Puppet:

- a forma clássica é chamada por prefixo. Exemplo:

```
template("ntp/ntp.conf.erb")
```

- a outra forma é a chamada encadeada. Exemplo:

```
"ntp/ntp.conf.erb".template
```

Essas duas formas de chamada possuem as mesmas capacidades, portanto você pode escolher a que for mais legível. Em geral, os critérios de escolha são:

- Para funções que usam muitos argumentos, são chamadas de prefixo e são mais fáceis de ler.
- Para funções que recebem um argumento normal e um `lambda` (será visto mais adiante), são chamadas encadeadas e são mais fáceis de ler.

- Para uma série de funções onde cada uma leva o resultado da anterior como seu argumento, também são chamadas encadeadas e são mais fáceis de ler.

Vamos conhecer um pouco mais destes dois tipos de chamadas.

10.1.3 Chamadas de função por prefixo

Você pode chamar uma função escrevendo seu nome e fornecendo uma lista de argumentos entre parênteses.

```
file {"/etc/ntp.conf":
  ensure  => file,
  content => template("ntp/ntp.conf.erb"), #chamada de funcao; retorna uma string
}

include apache #chamada de funcao; modifica o catálogo

$binaries = [ "facter",
  "hiera",
  "mco",
  "puppet",
  "puppetserver", ]

#chamada de funcao com lambda; executa um bloco de código algumas vezes
each($binaries) |$binary| {
  file {"/usr/bin/$binary":
    ensure => link,
    target => "/opt/puppetlabs/bin/$binary",
  }
}
```

Nos exemplos acima, `template`, `include` e `each` são todas funções. A função `template` é usada para retornar um valor, a `include` adiciona uma classe ao catálogo, e `each` executa um bloco de código várias vezes com valores diferentes.

A forma geral de uma chamada de função por prefixo é:

```
name(argument, argument, ...) |$parameter, $parameter, ...| { code block }
```

Assim temos:

- O nome completo da função, sem aspas.
- Um parênteses de abertura para a passagem de argumentos (( ). Parênteses são opcionais ao chamar uma função interna como no caso do `include`. Eles são obrigatórios em todos os outros casos.
- Zero ou mais argumentos, todos separados por vírgula. Os argumentos podem ser qualquer expressão que resolve um valor. Veja a documentação de cada função para obter o número de argumentos e seus tipos de dados: https://docs.puppet.com/puppet/latest/function.html.
- Um parênteses de fechamento ( )), caso tenha sido utilizado um parênteses de abertura.
- Opcionalmente, um lambda e um bloco de código, se a função aceitar.

10.1.4 Chamadas de função encadeadas

Você também pode chamar uma função escrevendo seu primeiro argumento, um ponto e o nome da função. Exemplo:

```
file {"/etc/ntp.conf":
  ensure  => file,
  content => "ntp/ntp.conf.erb".template, #chamada de funcao; retorna uma string
}

apache.include #chamada de funcao; modifica um catalogo

$binaries = [ "facter",
  "hiera",
  "mco",
  "puppet",
  "puppetserver", ]

#chamada de funcao com lambda; executa um bloco de código algumas vezes.
$binaries.each |$binary| {
  file {"/usr/bin/$binary":
    ensure => link,
    target => "/opt/puppetlabs/bin/$binary",
  }
}
```

Nos exemplos acima, `template`, `include` e `each` são todas funções e executam o mesmo trabalho explicado na seção anterior.

A forma geral de uma chamada de função encadeada é:

```
argument.name(argument, ...) |$parameter, $parameter, ...| { code block }
```

- O primeiro argumento da função, que pode ser qualquer expressão que resolve um valor.
- Um ponto (`.`).
- O nome completo da função, sem aspas.
- Opcionalmente, os parênteses que contém uma lista de argumentos separados por vírgula, começando com o segundo argumento da função, pois o primeiro argumento já foi citado no começo da chamada.
- Opcionalmente, um lambda e um bloco de código, se a função aceitar.

10.1.5 Comportamento

Uma chamada de função (incluindo o nome, argumentos e lambda) constitui uma expressão. Ela irá retornar um único valor, e pode ser usada em qualquer lugar em que esse valor retornado é aceito.

Uma chamada de função também pode resultar em algum efeito colateral, além de retornar um valor.

Todas as funções são executadas durante a compilação, o que significa que elas só acessam o código e dados disponíveis no Puppet Master. Para fazer alterações em um nó agente, você deve usar um `resource` (https://docs.puppet.com/puppet/latest/lang_resources.html). Para coletar dados de um nó agente use um fato customizado (https://docs.puppet.com/facter/3.5/custom_facts.html).

10.1.6 Funções de instrução embutidas

São um grupo de funções internas que são usadas apenas para causar efeitos colaterais. O Puppet 4 só reconhece as declarações embutidas da própria liguagem. Ele não permite adicionar novas funções de instruções como plugins.

A única diferença real entre as funções de instrução e as outras funções é que você pode omitir os parênteses ao chamar uma função de declaração com pelo menos um argumento (por exemplo, `include apache`).

Funções como a `include` retornam um valor como qualquer outra função, mas sempre retornarão um valor indefinido `undef`.

> ***Aviso***
>
>  **Saiba mais sobre as funções**
>
> Para obter mais informações sobre as funções acesse a página: https://docs.puppet.com/puppet/latest/lang_functions.html

10.2 Prática: Usando as funções

1) Escreva um manifest para montar diversos diretórios remotos compartilhados via NFS em diversos diretórios locais.

```
$storage_base      = "/home/storage/"
$storage_dir       = ["${storage_base}/01", "${storage_base}/02",]
$storage_device_fs = ["192.168.100.13:/home/m2", "192.168.100.13:/home/m3",]

case $::operatingsystem {
  centos,redhat: { $nfsclient = ["nfs-utils","nfs-utils-lib"] }
  debian,ubuntu: { $nfsclient = ["nfs-common"] }
  # fail é uma função
  default: {
    fail("sistema operacional desconhecido")
  }
}

package { $nfsclient:
  ensure => 'latest',
}
```

```
file { $storage_base:
  ensure  => 'directory',
  mode    => '755',
  owner   => root,
  group   => root,
  recurse => true,
}

file { $storage_dir:
  ensure  => 'directory',
  mode    => '755',
  owner   => root,
  group   => root,
  recurse => true,
  require => File[$storage_base],
}

each( $storage_device_fs) | Integer $index, String $value| {
  mount { $storage_dir[$index]:
    device  => $value,
    fstype  => 'nfs',
    ensure  => 'mounted',
    options => 'rw',
    atboot  => true,
    require => File[$storage_dir],
  }
  notice( "Device ${value} mounted in ${storage_dir[$index]}" )
}
```

Aviso

Configurar pontos de montagem via NFS

Para realizar este exercício, será necessário que você configure o NFSv3 num host remoto e compartilhe dois diretórios, com permissão de leitura e escrita para a montagem de diretório remoto. Na Internet você encontra vários tutoriais explicando como fazer isso. Abaixo estão alguns deles.

Ubuntu:
https://www.digitalocean.com/community/tutorials/how-to-set-up-an-nfs-mount-on-ubuntu-16-04

CentOS/Red Hat 7: https://goo.gl/3NqOs2

10.3 Lambdas

São blocos de código Puppet que podem ser passados para funções. Quando uma função recebe um lambda, ela pode fornecer valores para os parâmetros do lambda e avaliar seu código.

Se você usou outras linguagens de programação, você pode pensar em lambdas como simples funções anônimas, que podem ser passadas para outras funções.

10.3.1 Localização

Lambdas só podem ser usados em chamadas de função. Enquanto qualquer função pode aceitar um lambda, apenas algumas funções farão qualquer coisa com eles. Veja na página de interação e loops da linguagem Puppet, a lista de funções que são mais úteis no uso de lambda (https://docs.puppet.com/puppet/latest/lang_iteration.html).

Lambdas não são válidos em nenhum outro lugar na linguagem Puppet, e não podem ser atribuídos a variáveis.

10.3.2 Sintaxe

Lambdas são escritos como uma lista de parâmetros cercados por pipe (|), seguido por um bloco de código arbitrário em Puppet. Eles devem ser utilizados como parte de uma chamada de função. Exemplo:

```
$binaries = ["facter", "hiera", "mco", "puppet", "puppetserver"]

#chamada de funcao com lambda:
$binaries.each |String $binary| {
  file {"/usr/bin/$binary":
    ensure => link,
    target => "/opt/puppetlabs/bin/$binary",
  }
}
```

A forma geral de um lambda é:

```
|Type data optional $variable|
```

- A lista de parâmetros é obrigatória, mas pode estar vazia.
- Isso consiste em: uma barra vertical de abertura (|) e uma lista separada por vírgulas de zero ou mais parâmetros (por exemplo: String $myparam = "default value").
- Cada parâmetro é composto por um tipo de dados opcional, o que restringe os valores que ela permite. O padrão é `any` (qualquer). Também faz parte do parâmetro o nome da variável que o representa, incluindo o prefixo ($). Opcionalmente pode passar o sinal de igual (=).

- Opcionalmente, pode passar outra vírgula e argumentos extras (por exemplo: String $others = ["default one", "default two"]), que consiste em:
    - Um tipo de dados opcional, o que restringe os valores permitidos para argumentos extra (padrão any).
    - Um asterisco (*).
    - O nome da variável para representar o parâmetro, incluindo o prefixo ($).
    - Um sinal de igual opcional (=) e o valor padrão, que pode ser: o valor que corresponde ao tipo de dados especificado ou uma matriz de valores que coincidem com o tipo de dados.
    - Uma vírgula opcional após o último parâmetro.
    - Uma barra vertical fechamento (|).
    - Uma chave de abertura ({).
    - Um bloco de código de Puppet arbitrário.
    - Uma chave de fechamento (}).

10.3.3 Parâmetros e variáveis

Um lambda pode incluir uma lista de parâmetros e as funções podem definir valores para si quando chamam o lambda. Dentro do bloco de código do lambda você pode usar cada parâmetro como uma variável.

Funções passam parâmetros lambda por posição, da mesma forma que passa argumentos em uma chamada de função. Isto significa que a ordem dos parâmetros é importante, mas os seus nomes podem ser qualquer coisa (ao contrário dos parâmetros de classe ou de tipo definido, onde os nomes são a interface principal para os usuários).

Cada função decide quantos parâmetros passaram para um lambda e em que ordem. Consulte a documentação da função para obter os detalhes https://docs.puppet.com/puppet/latest/function.html.

Na lista de parâmetros, cada parâmetro pode ser precedido por um tipo de dados opcional. Se você incluir um, o Puppet verificará o valor do parâmetro no tempo de execução para certificar-se de que tem o tipo de dados certo, e exibirá um erro se o valor for ilegal. Se nenhum tipo de dados for fornecido, o parâmetro aceitará valores de qualquer tipo de dados.

Aviso

Saiba mais sobre os lambdas

Para obter mais informações sobre os lambdas e uso nas funções acesse a página: https://docs.puppet.com/puppet/latest/lang_lambdas.html

10.4 Prática: Usando funções com lambdas

1) Escreva um manifest para criar vários links que apontarão para vários alvos diferentes, sendo um link para cada alvo.

```
$binaries = ['facter', 'hiera', 'mco', 'puppet']

#Função com lambda:
$binaries.each | Integer $index, String $binary| {
  file {"/tmp/${binary}":
    ensure => link,
    target => "/opt/puppetlabs/bin/${binary}",
  }
 notice( "Link $index: nome do link: /tmp/${binary} => \
   alvo: /opt/puppetlabs/bin/${binary}" )
}
```

11 Iteração e Loop

A linguagem Puppet possui recursos de loop e iteração, que podem ajudá-lo a escrever códigos mais sucintos e usar dados de forma mais eficaz.

11.1 Noções básicas

Em Puppet, características de iteração são implementadas como funções que aceitam blocos de código (lambdas), mostrado no capítulo sobre Funções .

Ou seja, você escreve um bloco de código (lambda) que requer algum tipo de informação extra, então passa para uma função que pode fornecer essa informação para avaliar e executar o código, possivelmente várias vezes.

Isso difere de algumas outras linguagens em que as construções em loop são palavras-chave especiais; Em Puppet, eles são apenas funções.

11.1.1 Lista de funções de iteração

As seguintes funções podem aceitar um bloco de código e executá-lo de alguma maneira especial. Consulte a documentação de cada função para obter mais detalhes https://docs.puppet.com/puppet/latest/function.html.

- **each** - Repete um bloco de código qualquer por determinado número de vezes, utilizando um conjunto de valores para fornecer diferentes parâmetros de cada vez.
- **slice** - Repete um bloco de código qualquer por determinado número de vezes, utilizando grupos de valores de uma coleção como parâmetros.
- **filter** - Usa um bloco de código para transformar alguma estrutura de dados através da remoção de elementos não relacionados.
- **map** - Usa um bloco de código para transformar todos os valores em alguma estrutura de dados.
- **reduce** - Usa um bloco de código para criar um novo valor ou estrutura de dados através da combinação de valores a partir de uma estrutura de dados fornecida.
- **with** - Avalia um bloco de código uma vez, isolando-o em seu próprio escopo local. Não itera, mas tem uma semelhança familiar com as funções de iteração.

11.1.2 Sintaxe

Veja o capítulo sobre Funções para conhecer a sintaxe de chamadas de função e para conhecer a sintaxe de blocos de código que pode ser passado às funções.

Em geral, as funções de iteração recebem uma matriz ou um hash como seu principal argumento para, em seguida, iterar sobre seus valores.

11.1.3 Argumentos comuns lambda

As funções `each`, `filter` e `map` podem aceitar um lambda com um ou dois parâmetros. Os valores que passam para um lambda variam, dependendo do número de parâmetros e do tipo de estrutura de dados que você está iterando:

Tipo de dados	Parâmetro Único	Dois parâmetros
Array	<VALUE>	<INDEX>,
Hash	[<KEY>, <VALUE>] (two-element array)	<KEY>, <VALUE>

Por exemplo:

```
['a','b','c'].each |Integer $index, String $value| { notice("${index} = ${value}") }
```

Isso resultará em:

```
Notice: Scope(Class[main]): 0 = a
Notice: Scope(Class[main]): 1 = b
Notice: Scope(Class[main]): 2 = c
```

As funções `slice` e `reduce` lidam com parâmetros de forma diferente. Consulte a documentação oficial para obter detalhes.

11.1.4 Exemplos na declaração de recursos

Uma vez que o foco da linguagem Puppet é declarar recursos, a maioria das pessoas vai querer usar a iteração para declarar muitos recursos semelhantes de uma vez:

```
$binaries = ['facter', 'hiera', 'mco', 'puppet', 'puppetserver']

# function call with lambda:
$binaries.each |String $binary| {
  file {"/usr/bin/${binary}":
    ensure => link,
    target => "/opt/puppetlabs/bin/${binary}",
  }
}
```

Neste exemplo, temos uma matriz de nomes de comandos que queremos usar no caminho e no destino de cada link simbólico. A função `each` faz isso muito fácil e sucinta.

11.1.5 Usando iteração para transformar dados

Você também pode usar a iteração para transformar dados em formulários mais úteis. Por exemplo:

Exemplo 1:

```
$filtered_array = [1,20,3].filter |$value| { $value < 10 }
# retorna [1,3]
```

Exemplo 2:

```
$sum = reduce([1,2,3]) |$result, $value|  { $result + $value }
# retorna 6
```

Exemplo 3:

```
$hash_as_array = ['key1', 'first value',
                  'key2', 'second value',
                  'key3', 'third value']

$real_hash = $hash_as_array.slice(2).reduce( {} ) |Hash $memo, Array $pair| {
  $memo + $pair
}
# retorna {"key1"=>"first value", "key2"=>"second value", "key3"=>"third value"}
```

11.2 Prática: Usando funções de loop e iteração

1) Escreva um manifest, no qual dado um hash retorne todos os valores que contém o trecho "berry"

```
$data = { "orange" => 0, "blueberry" => 1, "raspberry" => 2 }
$filtered_data = $data.filter |$items| { $items[0] =~ /berry$/ }
notice( "Resultado: $filtered_data" )
```

2) Escreva outro manifest, no qual dado um hash retorne todos os valores que contém o trecho "berry" e valor igual a 1.

```
$data = { "orange" => 0, "blueberry" => 1, "raspberry" => 2 }
$filtered_data = $data.filter |$keys, $values| { $keys =~ /berry$/ and $values <= 1 }
notice( "Resultado: $filtered_data" )
```

3) Escreva outro manifest, no qual dado um hash retorne a soma de todos os valores e todas as strings concatenadas.

```
$data = {a => 1, b => 2, c => 3}
$combine = $data.reduce |$memo, $value| {
  $string = "${memo[0]}${value[0]}"
  $number = $memo[1] + $value[1]
  [$string, $number]
}
notice( "Resultado: $combine" )
```

4) Escreva outro manifest, no qual dado um array de números retorne-os organizados em pares.

```
$result = slice([1,2,3,4,5,6], 2)
notice( "Resultado: $result" )
```

5) Escreva outro manifest, no qual dado uma array de caracteres retorne-os organizados em pares.

```
$result = slice('puppet',2)
notice( "Resultado: $result" )
```

12 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***
>
>  **Sobre o exercício**
>
> Faça uma cópia do arquivo `/etc/ssh/sshd_config` 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`.

13 Master / Agent

O Puppet pode ser utilizado com a arquitetura *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, 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.

Aviso

Catálogo e Relatórios

Mais detalhes sobre a compilação do catálogo e envio dos relatórios podem ser encontradas, respectivamente, nas seguintes páginas:
https://docs.puppet.com/puppet/latest/subsystem_catalog_compilation.html
https://docs.puppet.com/puppet/latest/reporting_about.html
https://docs.puppet.com/puppet/latest/report.html
https://docs.puppet.com/puppet/latest/format_report.html

13.1 Resolução de nomes

A configuração de nome e domínio do sistema operacional, além da resolução de nomes, é uma boa prática para o funcionamento do Puppet, devido ao uso de certificados `SSL-Secure Socket Layer` 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.

```
sudo hostname
node1

sudo hostname --domain
domain.com.br

sudo hostname --fqdn
node1.domain.com.br
```

Dica

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

Para resolução de nomes, configure corretamente o arquivo `/etc/resolv.conf` com os parâmetros: `nameserver`, `domain` e `search`. Esses parâmetros devem conter a informação do(s) servidor(es) DNS e do domínio de sua rede.

O arquivo `/etc/hosts` deve possuir pelo menos o nome do próprio host em que o agente está instalado. Neste arquivo deve existir uma entrada que informe o seu IP, FQDN e depois o hostname. Exemplo: `192.168.1.10 node1.domain.com.br node1`.

No Debian/Ubuntu, o `hostname` é cadastrado no arquivo `/etc/hostname`.

No CentOS/Red Hat, o `hostname` é cadastrado na variável `HOSTNAME` do arquivo `/etc/sysconfig/network`.

É uma boa prática que sua rede possua a resolução de nomes configurada via DNS. Neste caso, o hostname e domínio de cada sistema operacional devem resolver corretamente para o seu respectivo IP.

Porém tecnicamente, a resolução de nomes dos agentes via DNS, não é requisito para o funcionamento do Puppet. Em termos de DNS, o único requisito de fato é que o host consiga resolver o nome do servidor Puppet Master. Por padrão, o agente vai usar o FQDN do host como o `CN-Common Name` para indentificá-lo durante a criação do certificado SSL. Entretanto, é possível usar o Puppet em situações que seja necessário que o CN do certificado não possua nenhuma relação com o DNS.

13.2 Segurança e autenticação

As conexões entre agente e servidor Puppet 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 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 operacional no certificado.

13.3 Prática Master / Agent

13.3.1 Instalação do Master

1. O pacote `puppetserver` deverá ser instalado no host que atuará como Master. Certifique-se de que o hostname está correto:

```
sudo hostname --fqdn
master.domain.com.br
```

Assumindo que os passos do capítulo Instalação foram executados anteriormente, instale o PuppetServer com o comando abaixo.

```
sudo puppet resource package puppetserver ensure=latest
```

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

```
sudo tree -F --dirsfirst /etc/puppetlabs/
/etc/puppetlabs/
|-- code/
|   |-- environments/
|   |   |-- production/
|   |       |-- hieradata/
|   |       |-- manifests/
|   |       |-- modules/
|   |       |-- environment.conf
|   |-- modules/
|   |-- hiera.yaml
|-- mcollective/
|   |-- client.cfg
|   |-- data-help.erb
|   |-- discovery-help.erb
|   |-- facts.yaml
|   |-- metadata-help.erb
|   |-- rpc-help.erb
|   |-- server.cfg
|-- puppet/
|   |-- ssl/
|   |-- auth.conf
|   |-- puppet.conf
|-- puppetserver/
|-- |-- conf.d/
|   |   |-- auth.conf
|   |   |-- global.conf
|   |   |-- puppetserver.conf
|   |   |-- web-routes.conf
|   |   |-- webserver.conf
|   |-- services.d/
|   |   |-- ca.cfg
|   |-- logback.xml
|   |-- request-logging.xml
```

- Os arquivos e diretórios de configuração mais importantes são:
  - `auth.conf`: regras de acesso a API REST do Puppet.
  - `code/environments/production/manifests/`: Armazena a configuração que será compilada e servida para os agentes que executam no ambiente de *production* (padrão).
  - `code/environments/production/modules/`: Armazena módulos com classes, arquivos, plugins e mais configurações para serem usadas nos manifests para o ambiente de *production* (padrão).
  - `puppet.conf`: Arquivo de configuração usado pelo Master assim como o Agent.

Dica

 **Sobre os arquivos de configuração**

Nas páginas abaixo você encontra mais detalhes sobre os arquivos de configuração do Puppet:

https://docs.puppet.com/puppet/latest/config_important_settings.html
https://docs.puppet.com/puppet/latest/dirs_confdir.html
https://docs.puppet.com/puppet/latest/config_about_settings.html
https://docs.puppet.com/puppet/latest/config_file_main.html
https://docs.puppet.com/puppet/latest/configuration.html

Nota

 **Sobre os binários do Puppet**

A instalação do Puppet e todos seus componentes fica em `/opt/puppetlabs`.

Os arquivos de configuração ficam em `/etc/puppetlabs`.

2. Configurando o serviço:

Altere as configurações de memória da JVM que é utilizada pelo Puppet Server para adequá-las a quantidade de memória disponível.

No CentOS/Red Hat edite o arquivo `/etc/sysconfig/puppetserver` e no Debian/Ubuntu edite o arquivo `/etc/default/puppetserver`:

```
JAVA_ARGS="-Xms256m -Xmx512m"
```

Reinicie o serviço com o comando abaixo.

```
sudo service puppetserver restart
```

Com esta configuração será alocado 512 MB para uso da JVM usada pelo Puppet Server. Por padrão, são alocados 2 GB de memória.

3. No host PuppetServer, gere um certificado e inicie os serviço com os comandos abaixo.

```
sudo puppet cert generate master.domain.com.br

sudo puppet resource service puppetserver ensure=running enable=true
```

> ***Nota***
>
>  **Configuração de firewall e NTP**
>
> Mantenha a hora corretamente configurada utilizando NTP para evitar problemas na assinatura de certificados.
>
> A porta `8140/TCP` do servidor Puppet Server precisa estar acessível para os demais hosts que possuem o Puppet Agent instalado.

As solicitações de assinatura de certificados no Puppet-Server ficam em: **/etc/puppetlabs/puppet/ssl/ca/requests/**

Os logs do PuppetServer ficam em:

- `/var/log/puppetlabs/puppetserver/puppetserver.log`
- `/var/log/puppetlabs/puppetserver/puppetserver-daemon.log`

13.3.2 Instalação do agente em node1

Assumindo que os passos do capítulo Instalação foram executados anteriormente no host `node1`. O Puppet Agent já está instalado. Configure o Puppet Agent com os passos a seguir.

1. Certifique-se de que o nome e domínio do sistema estejam corretos:

```
sudo hostname --fqdn
node1.domain.com.br
```

2. Em um host em que o agente está instalado, precisamos configurá-lo para que ele saiba quem é o Master.

No arquivo `/etc/puppetlabs/puppet/puppet.conf`, adicione as linhas abaixo:

```
[main]
certname = node1.domain.com.br
server = master.domain.com.br
environment = production

[agent]
report = true
```

> ***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:

```
sudo 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.domain.com.br
Info: Certificate Request fingerprint (SHA256): 6C:7E:E6:3E:EC:A4:15:56: ...
```

4. No servidor Master aparecerá a solicitação de assinatura para o host `node1.domain.com.br`. Assine-a.

- Os comandos abaixo devem ser executados em **master.domain.com.br**.

```
sudo puppet cert list

"node1.domain.com.br" (SHA256) 6C:7E:E6:15:56:49:C3:1E:A5:E4:7F:58:B8: ...
```

```
sudo puppet cert sign node1.domain.com.br

Signed certificate request for node1.domain.com.br
Removing file Puppet::SSL::CertificateRequest node1.domain.com.br at
'/var/lib/puppet/ssl/ca/requests/node1.domain.com.br.pem'
```

Para listar todos os certificados que já foram assinados pelo Puppet Server, use o comando abaixo:

```
sudo puppet cert list -a
```

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

- O comando abaixo deve ser executado em **node1.domain.com.br**.

```
sudo puppet agent -t

Info: Caching certificate for node1.domain.com.br
Info: Caching certificate_revocation_list for ca
Info: Retrieving plugin
Info: Caching catalog for node1.domain.com.br
Info: Applying configuration version '1352824182'
Info: Creating state file /var/lib/puppet/state/state.yaml
Finished catalog run in 0.05 seconds
```

Agora execute os comandos abaixo para iniciar o agente do Puppet como serviço e habilitá-lo para ser executado após o boot do sistema operacional:

```
sudo puppet resource service puppet ensure=running enable=true
```

No Puppet-Agent, os certificados assinados ficam em: **/etc/puppetlabs/puppet/ssl/**

Se precisar refazer a assinatura de certificados do host puppet-agent é só parar o serviço `puppet-agent` com o comando abaixo e depois apagar os arquivos e sub-diretórios que ficam em: **/etc/puppetlabs/puppet/ssl/**.

```
sudo puppet resource service puppet ensure=stop
```

Os logs do puppet-agent ficam em:

- `/var/log/messages` (no Debian/Ubuntu)
- `/var/log/syslog` (no CentOS/Red Hat).
- `/var/log/puppetlabs/puppet`

Dica

Possíveis problemas com certificados SSL

É importante que os horários do Master e dos nodes estejam 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.

Você também pode consultar esta página https://docs.puppet.com/puppet/latest/ssl_regenerate_certificates.html para saber como reconfigurar os certificados no Agente e Master.

http://www.linuxnix.com/puppet-how-to-remove-puppet-client-from-master/

Nota

Recriando certificados para o node

Se por algum motivo, for necessário recriar o certificado do Puppet Agent de um node, execute o seguintes passos:

1. Removendo o certificado do node no Puppet Server.

```
sudo puppet cert clean <name_certificate_hostname>
```

Exemplo:

```
sudo puppet cert clean node1.domain.com.br
```

2. Removendo o certificado do node nele mesmo.

```
sudo puppet resource service puppet ensure=stopped
sudo rm -r /etc/puppetlabs/puppet/ssl
sudo puppet cert list -a
```

Feito isso é só assinar a solicitação do novo certificado no Puppet Server, conforme mostrado neste capítulo. Veja mais detalhes em: https://docs.puppet.com/puppet/latest/ssl_regenerate_certificates.html

Nota

Removendo solicitações indesejadas de assinaturas de certificado

Se no Puppet Master houver solicitações de assinatura de certificados para nodes desconhecidos, basta removê-las executando o comando abaixo no Puppet Server:

```
sudo puppet ca destroy <name_certificate_hostname>
```

Exemplo:

```
sudo puppet ca destroy node4.domain.com.br
```

14 Nodes

O Puppet começa a compilação da configuração de um catálogo pelo arquivo `/etc/puppetlabs/code/environments/production/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).

14.1 Declarando nodes

Sintaxe para se declarar um node:

```
sudo vim /etc/puppetlabs/code/environments/production/manifests/site.pp

node 'node1.domain.com.br' {
  package {'nano':
    ensure => 'present',
  }
}

node 'node2.domain.com.br' {
  package {'vim':
    ensure => 'present',
  }
}
```

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

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.

A documentação oficial está em: https://docs.puppet.com/guides/external_nodes.html

14.2 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**.

É possível casar nomes de nodes usando expressões regulares:

```
Para: www1, www13, www999, a declaracao pode ser:
node /^www\d+$/ {

}

Para: foo.domain.com.br ou bar.domain.com.br, a declaração pode ser:
node /^(foo|bar)\.domain\.com\.br$/ {

}
```

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

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

}
```

14.3 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 {

}
```

14.4 Prática

1. Declare o host **node1.domain.com.br** no arquivo `/etc/puppetlabs/code/environments/production/manifests/site.pp` do master.

2. Declare o pacote `tcpdump` como instalado para **node1.domain.com.br**.

3. Execute o comando `puppet agent -t` no `node1`, certifique-se de que o pacote `tcpdump` foi instalado.

> ***Dica***
>
>  **Simulando a configuração**
>
> Para simularmos as alterações que serão ou não realizadas no host cliente,usamos o comando `puppet agent -t --noop`.

15 PuppetDB e Dashboards Web

Nos capítulos Instalação e Master / Agent, vimos como instalar o puppet-agent e puppetserver. Agora vamos aprender a instalar o PuppetDB e o PuppetBoard, que oferecem uma interface web que permite acompanhar o que acontece em cada ciclo de operação executado nos hosts que possuem o agente instalado.

Os passos de instalação a seguir são executados apenas no host **master.domain.com.br** e é assumido que o `puppet-agent` e `puppetserver` estão instalados.

1. Antes de iniciar a instalação do PuppetDB, instale o PostgreSQL com os comandos abaixo.

- No CentOS/RedHat 6 64 bits:

```
su -
URL_BASE="https://download.postgresql.org/pub/repos/yum/9.5/redhat/rhel-6-x86_64/"
sudo rpm -ivh --force $URL_BASE/pgdg-redhat95-9.5-2.noarch.rpm
yum -y install postgresql95-server postgresql95-contrib
service postgresql-9.5 initdb
chkconfig postgresql-9.5 on
```

- No CentOS/RedHat 7 64 bits:

```
su -
URL_BASE="https://download.postgresql.org/pub/repos/yum/9.5/redhat/rhel-7-x86_64/"
sudo rpm -ivh --force $URL_BASE/pgdg-redhat95-9.5-2.noarch.rpm
yum -y install postgresql95-server postgresql95-contrib
/usr/pgsql-9.5/bin/postgresql95-setup initdb
service postgresql-9.5 start
chkconfig postgresql-9.5 on
```

- No Debian 8:

```
su -
echo "deb http://apt.postgresql.org/pub/repos/apt/ jessie-pgdg main" > \
/etc/apt/sources.list.d/pgdg.list
wget --quiet -O - https://www.postgresql.org/media/keys/ACCC4CF8.asc | apt-key add -
apt-get update
apt-get -y install postgresql-9.5
```

- No Ubuntu Server 14.04:

```
sudo su
echo "deb http://apt.postgresql.org/pub/repos/apt/ trusty-pgdg main" > \
/etc/apt/sources.list.d/pgdg.list
wget --quiet -O - https://www.postgresql.org/media/keys/ACCC4CF8.asc | apt-key add -
apt-get update
apt-get -y install postgresql-9.5
```

- No Ubuntu Server 16.04:

```
sudo su
echo "deb http://apt.postgresql.org/pub/repos/apt/ xenial-pgdg main" > \
/etc/apt/sources.list.d/pgdg.list
wget --quiet -O - https://www.postgresql.org/media/keys/ACCC4CF8.asc | apt-key add -
apt update
apt -y install postgresql-9.5
```

Edite o arquivo `/etc/postgresql/9.5/main/pg_hba.conf` (no Debian/Ubuntu) ou `/var/lib/pgsql/9.5/data/pg_hba.conf` (no CentOS/Red Hat) e configure o arquivo como mostrado abaixo.

Antes:

```
local     all       postgres      peer
local     all       all           peer
host     all      127.0.0.1/32  md5
host     all      ::1/128           md5
```

Depois:

```
local     all       postgres         trust
local     all       all              trust
host    all       127.0.0.1/32     trust
host     all      ::1/128              trust
```

Outro arquivo que precisa ser editado é o `/etc/postgresql/9.5/main/postgresql.conf` (no Debian/Ubuntu) ou `/var/lib/pgsql/9.5/data/postgresql.conf` (no CentOS/Red Hat) conforme mostrado abaixo.

Antes:

```
#listen_addresses = 'localhost'
port = 5432
```

Depois:

```
listen_addresses = 'localhost'
port = 5432
```

Depois de alterar estes arquivos é preciso reiniciar o PostgreSQL para aplicar as configurações realizadas. Faça isso usando o comando abaixo.

No Debian/Ubuntu:

```
sudo service postgresql restart
```

No CentOS/Red Hat:

```
sudo service postgresql-9.5 restart
```

Agora crie o banco de dados e usuário para o puppetdb com a sequencia de comandos abaixo.

```
su postgres
createdb puppetdb
createuser -a -d -E -P puppetdb
exit
```

2. Instale o PuppetDB com o comando abaixo.

```
sudo puppet resource package puppetdb ensure=latest
```

3. No servidor PuppetServer, edite o arquivo `/etc/puppetlabs/puppetdb/conf.d/database.ini` e altere as seguinte linhas:

```
[database]
 classname = org.postgresql.Driver
 subprotocol = postgresql
 subname = //localhost:5432/puppetdb
 username = puppetdb
 password = SENHA-DO-USUARIO-PUPPETDB
```

Agora edite o arquivo `/etc/puppetlabs/puppetdb/conf.d/jetty.ini` e altere os parâmetros a seguir para deixá-los com esses valores.

```
[jetty]
 host = 0.0.0.0
 port = 8080
 ssl-host = 0.0.0.0
 ssl-port = 8081
```

Execute o comando abaixo para gerar os certificados a serem usados pelo PuppetDB e configurar o arquivo `jetty.ini`.

```
sudo puppetdb ssl-setup
```

Reinicie o PuppetDB com o comando abaixo:

```
sudo service puppetdb restart
```

4. Ainda no servidor PuppetServer, instale o pacote `puppetdb-termini` com o comando abaixo.

```
sudo puppet resource package puppetdb-termini ensure=latest
```

Crie o arquivo `/etc/puppetlabs/puppet/puppetdb.conf` e adicione o seguinte conteúdo:

```
[main]
 server_urls = https://master.domain.com.br:8081
```

Edite o arquivo `/etc/puppetlabs/puppet/puppet.conf` e adicione o seguinte conteúdo na seção `[master]`.

```
storeconfigs = true
storeconfigs_backend = puppetdb
reports = puppetdb
reportstore = /var/log/puppetlabs/puppet
```

Crie o arquivo `/etc/puppetlabs/puppet/routes.yaml` com o seguinte conteúdo:

```
---
 master:
   facts:
     terminus: puppetdb
     cache: yaml
```

Atribua as permissões corretas ao arquivo com o comando abaixo:

```
sudo chown -R puppet:puppet `puppet config print confdir`
```

Reinicie o PuppetServer com o comando abaixo:

```
sudo service puppetserver restart
```

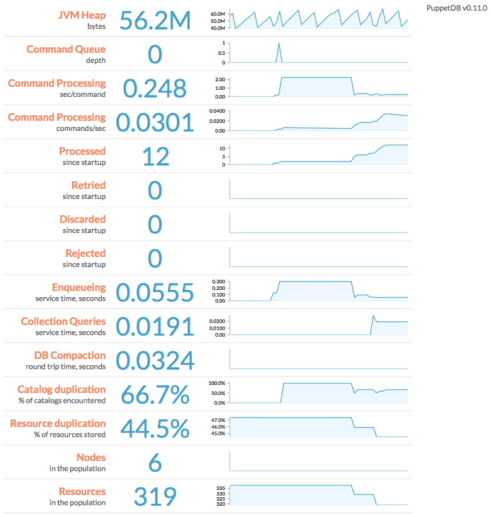
Aviso

Informações sobre o PuppetDB

Mais informações sobre a instalação do PuppetDB podem ser encontradas nas páginas: https://docs.puppet.com/puppetdb/latest/configure.html e https://docs.puppet.com/puppetdb/4.1/connect_puppet_master.html

Os arquivos de log do PuppetDB ficam em: `/var/log/puppetlabs/puppetdb/`

O PuppetDB ficará acessível em: http://master.domain.com.br:8080 ou https://master.domain.com.br:8081



Aviso

Possíveis problemas no acesso ao PuppetDB

Se não conseguir acessar a interface web do PuppetDB, verifique se há algum firewall bloqueando a porta.

No CentOS/Red Hat 6, você pode desabilitar o firewall seguindo estas instruções: http://www.cyberciti.biz/faq/fedora-redhat-centos-5-6-disable-firewall

No CentOS/Red Hat 7, você pode desabilitar o firewall seguindo estas instruções: http://www.liquidweb.com/kb/how-to-stop-and-disable-firewalld-on-centos-7

Você também pode precisar desabilitar o SELinux no CentOS/RedHat. Siga estas instruções: http://www.revsys.com/writings/quicktips/turn-off-selinux.html ou http://aruljohn.com/info/centos-selinux/

15.1 Instalando o PuppetBoard

Os módulos Puppet de instalação do PuppetBoard e dependências podem ser instalados no Puppet-Server usando os comandos abaixo.

```
sudo puppet module install puppetlabs-apache
sudo puppet module install puppetlabs-apt
sudo puppet module install puppet-puppetboard
```

Agora edite o aquivo `/etc/puppetlabs/code/environments/production/manifests/site.pp` e adicione o seguinte conteúdo:

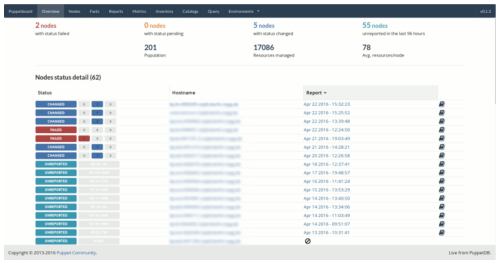
```
node puppetserver.domain.com.br {
  class {'apache':
     default_vhost    => false,
     server_signature => 'Off',
```

```
    server_tokens    => 'Prod',
    trace_enable     => 'Off',
  }

  #Definindo a porta padrao do HTTP
  apache::listen { '80': }

  class { 'apache::mod::ssl':
    ssl_cipher   => 'HIGH:MEDIUM:!aNULL:!MD5:!SSLv3:!SSLv2:!TLSv1:!TLSv1.1',
    ssl_protocol => [ 'all', '-SSLv2', '-SSLv3', '-TLSv1', '-TLSv1.1' ],
  }

  #Configurando o modulo wsgi
  class { 'apache::mod::wsgi':
    wsgi_socket_prefix => '/var/run/wsgi',
  }

  #Configurando o Puppetboard
  class { 'puppetboard':
    manage_git          => 'latest',
    manage_virtualenv   => 'latest',
    reports_count       => 50
  }->
  python::pip { 'Flask':
    virtualenv => '/srv/puppetboard/virtenv-puppetboard',
  }->
  python::pip { 'Flask-WTF':
    virtualenv => '/srv/puppetboard/virtenv-puppetboard',
  }->
  python::pip { 'WTForms':
    virtualenv => '/srv/puppetboard/virtenv-puppetboard',
  }->
  python::pip { 'pypuppetdb':
    virtualenv => '/srv/puppetboard/virtenv-puppetboard',
  }

  #Configurando o Acesso ao Puppetboard via HTTPS
  class { 'puppetboard::apache::vhost':
    vhost_name => 'master.domain.com.br',
    port       => 443,
    ssl        => true,
  }
}
```

Agora execute o comando abaixo.

```
sudo puppet agent -t
```

Ao final da instalação, o PuppetBoard ficará acessível em: https://master.domain.com.br



É possível que você enfrente o problema abaixo na instalação do PuppetBoard no CentOS/Red Hat 7.

Erro:

```
Execution of '/usr/bin/rpm -e python-devel-2.7.5-34.el7.x86_64' returned 1:
 error: Failed dependencies:
python2-devel is needed by (installed) python-virtualenv-1.10.1-2.el7.noarch
Error: /Stage[main]/Python::Install/Package[python-dev]/ensure: change from
 2.7.5-34.el7 to absent failed: Execution of '/usr/bin/rpm -e
 python-devel-2.7.5-34.el7.x86_64' returned 1: error: Failed  dependencies:
python2-devel is needed by (installed) python-virtualenv-1.10.1-2.el7.noarch
```

Solução:

Edite o arquivo `/etc/puppetlabs/code/environments/production/modules/python/manifests/install.pp`. Altere todas as ocorrências de:

```
package { 'python-dev':
ensure => $dev_ensure,
name => $pythondev,
}
```

Para:

```
package { 'python-dev':
ensure => present,
#ensure => $dev_ensure,
name => $pythondev,
}
```

Depois execute:

```
sudo puppet agent -t
```

Mais informações sobre o PuppetBoard podem ser encontradas em: https://forge.puppet.com/puppet/puppetboard

15.2 Configurando os Agentes Puppet

Em cada host que executa o Puppet-Agent, adicione no arquivo `/etc/puppetlabs/puppet/puppet.conf` o seguinte conteúdo:

```
[agent]
 report = true
```

Reinicie o Puppet-Agent com o comando abaixo:

```
sudo service puppet restart
```

16 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.

16.1 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:

```
sudo vim ntp.pp

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

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

Vejamos o resultado ao aplicar esse código:

```
sudo 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.

```
sudo vim 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:

```
sudo 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.

16.1.1 Diretiva include

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

```
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. Mais informações sobre as classes podem ser obtidas na página https://docs.puppet.com/puppet/latest/lang_classes.html

16.2 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`.

```
sudo puppet config print modulepath

/etc/puppetlabs/code/environments/production/modules: \
  /etc/puppetlabs/code/modules:/opt/puppetlabs/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.

16.2.1 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
      - `minha_outra_classe1.pp`: contém uma classe chamada `meu_modulo::um_diretorio::minha_outra_classe1`
      - `minha_outra_classe2.pp`: contém uma classe chamada `meu_modulo::um_diretorio::minha_outra_classe2`
  - `files/`: arquivos estáticos que podem ser baixados pelos agentes
  - `lib/`: plugins e fatos customizados implementados em Ruby
  - `templates/`: contém templates usados no módulo
  - `tests/`: exemplos de como classes e tipos do módulo podem ser chamados

16.3 Prática: criando um módulo

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

```
sudo cd /etc/puppetlabs/code/environments/production/modules
sudo 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.

```
sudo vim /etc/puppetlabs/code/environments/production/modules/ntp/manifests/init.pp

class ntp {
  case $::operatingsystem {
    centos, redhat: { $service_ntp = "ntpd" }
    debian, ubuntu: { $service_ntp = "ntp" }
    default: { fail("sistema operacional desconhecido") }
  }

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

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

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

```
sudo vim /etc/puppetlabs/code/environments/production/manifests/site.pp

node 'node1.domain.com.br' {
  include ntp
}
```

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

```
sudo puppet agent -t
```

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

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

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

Nota

Nome do serviço NTP

No Debian/Ubuntu, o nome do serviço é `ntp`. No CentOS/Red Hat, o nome do serviço é `ntpd`.

16.4 Prática: arquivos de configuração em módulos

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

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

```
sudo cd /etc/puppetlabs/code/environments/production/modules
sudo mkdir -p ntp/files
```

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

```
sudo cp /etc/ntp.conf /etc/puppetlabs/code/environments/production/modules/ntp/files/
```

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

```
class ntp {

  ...

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

}
```

4. Faça qualquer alteração no arquivo `ntp.conf` do módulo (em `/etc/puppetlabs/code/environments/production/modules/ntp/files/ntp.conf`), por exemplo, acrescentando ou removendo um comentário.

5. Aplique a nova configuração no **node1**.

```
sudo puppet agent -t
```

Dica

Servidor de arquivos do Puppet

O Puppet pode servir os 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/`.

Mais informações sobre os módulos podem ser obtidas na página: https://docs.puppet.com/puppet/latest/modules_fundamentals.html

17 Puppet Forge

Ao longo da história da computação, programadores desenvolveram diversas técnicas para evitar retrabalho. Estão disponíveis aos programadores bibliotecas de código que implementam diversas funcionalidades prontas para uso. Além disso, ao desenvolver um software, certamente um programador competente concentra rotinas repetidas ou parecidas em bibliotecas que podem ser reutilizadas no seu projeto.

Infelizmente, no mundo da administração de sistemas, aproveitar soluções de problemas que já foram resolvidos por outro administrador é muito raro. SysAdmins de diferentes organizações estão resolvendo os mesmos problemas todos os dias. Configurando e instalando servidores web, banco de dados, fazendo ajustes de segurança e etc.

Não seria incrível se os SysAdmins pudessem aproveitar o trabalho uns dos outros? Para isso o Puppet Forge foi criado (https://forge.puppet.com)!

Puppet Forge é um repositório de módulos escritos pela comunidade para o Puppet Open Source e Puppet Enterprise. Nele encontramos diversos módulos prontos para uso, e que com pouquíssimas linhas em um manifest podemos poupar horas e horas de trabalho aproveitando módulos úteis desenvolvidos por SysAdmins ao redor do mundo.

17.1 Prática: módulo para sysctl do Puppet Forge

Um dos itens comumente configurados em sistemas operacionais são os parâmetros de kernel, usando o comando `sysctl`.

Poderíamos criar um módulo para que essas configurações fossem gerenciadas via Puppet, mas felizmente alguém já deve ter resolvido esse problema.

1. Faça uma busca por **sysctl** em https://forge.puppet.com
2. Já existem vários módulos para tratar esse problema. Vamos instalar um deles (já testado anteriormente, por isso a escolha):

```
sudo puppet module install trlinkin/sysctl

Notice: Preparing to install into \
  /etc/puppetlabs/code/environments/production/modules ...
Notice: Downloading from https://forgeapi.puppetlabs.com ...
Notice: Installing -- do not interrupt ...
/etc/puppetlabs/code/environments/production/modules
|--- trlinkin-sysctl (v0.0.2)
```

3. Use o módulo, via linha de comando.

```
sudo puppet apply -e "sysctl { 'net.ipv4.ip_forward': value => '1', enable=>true }"
```

4. Declare um valor para um dos nossos nodes no Puppet Master editando o arquivo `/etc/puppetlabs/code/environments/production/manifests/site.pp`.

```
node 'node1.domain.com.br' {
  sysctl { 'net.ipv4.ip_forward':
    value   => '1',
    enable  => true,
  }
}
```

5. Aplique essa regra no **node1**.

```
sudo puppet agent -t

Info: Retrieving plugin
...
Info: Caching catalog for node1.puppet
Info: Applying configuration version '1353001737'
/Stage[main]//Node[node1.puppet]/Sysctl[net.ipv4.ip_forward]/value: \
value changed '0' to '1'
/Stage[main]//Node[node1.puppet]/Sysctl[net.ipv4.ip_forward]/enable: modified \
          running value of 'net.ipv4.ip_forward' from '0' to '1'
Info: FileBucket adding {md5}228c966fd2676164a120f5230fe0b0e1
Finished catalog run in 0.17 seconds
```

17.2 Prática: módulo para autofsck do Puppet Forge

1. Instale o módulo `jhoblitt/autofsck`.

```
sudo puppet module install jhoblitt/autofsck

Notice: Preparing to install into \
 /etc/puppetlabs/code/environments/production/modules ...
Notice: Downloading from https://forgeapi.puppetlabs.com ...
Notice: Installing -- do not interrupt ...
/etc/puppetlabs/code/environments/production/modules
|--| jhoblitt-autofsck (v1.1.0)
|--- puppetlabs-stdlib (v4.12.0)
```

2. Declare o módulo `autofsck` na configuração de **node1**:

```
node 'node1.domain.com.br' {
  include autofsck
}
```

3. Execute o agente em **node1**:

```
sudo puppet agent -t
```

18 Puppet Tagmail

No capítulo PuppetDB e Dashboards Web, vimos como instalar o PuppetDB. Agora vamos aprender a instalar e configurar o módulo **Tagmail** para permitir que os relatórios de cada execução do puppet-agent também sejam enviados por e-mail, além de serem registrados no PuppetDB.

Os passos de instalação a seguir são executados apenas no host **master.domain.com.br** e é assumido que o `puppet-agent` e `puppetserver` estão instalados.

```
puppet module install puppetlabs-tagmail
```

Ainda no servidor PuppetServer, edite o arquivo `/etc/puppetlabs/puppet/puppet.conf` e edite a seguinte linha na seção `[master]`.

Antes:

```
reports = puppetdb
```

Depois:

```
reports = puppetdb,tagmail
```

Crie o arquivo `/etc/puppetlabs/puppet/tagmail.conf` com o seguinte conteúdo:

```
[transport]
reportfrom = reports@example.org
smtpserver = smtp.example.org
smtpport = 25
smtphelo = example.org

[tagmap]
warning,err,alert,emerg,crit: username1@gmail.com,username2@gmail.com
err,emerg,crit: username1@yahoo.com.br
warning,alert: username2@gmail.com
```

Conforme pode ser visto no exemplo acima, na seção `transport` ficam os dados do servidor de email que será utilizado para enviar os relatórios por email. O servidor de email pode ser instalado no mesmo host do PuppetServer ou em outro da sua rede. Nesta apostila não será explicado os detalhes da configuração do servidor de email.

Na seção `tagmap` ficam os dados do email das pessoas que receberão os alertas de acordo com os níveis de log.

Reinicie o PuppetServer com o comando abaixo:

```
service puppetserver restart
```

Aviso

 **Informações sobre o Tagmail**

Mais informações sobre a configuração do módulo Tagmail podem ser encontradas em: https://forge.puppet.com/puppetlabs/tagmail

18.1 Configurando os Agentes Puppet

Em cada node que executa o Puppet-Agent, adicione no arquivo `/etc/puppetlabs/puppet/puppet.conf` o seguinte conteúdo:

```
[agent]
 report = true
 pluginsync = true
```

Reinicie o Puppet-Agent com o comando abaixo:

```
service puppet restart
```

19 Templates

Muitas vezes temos um mesmo serviço ativado em diversos hosts, mas em um conjunto de hosts esse serviço precisa ser configurado de uma maneira e, no restante dos hosts, de outra. Assim, cada conjunto de host 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 hosts, seria necessário atualizar dois arquivos de configuração. Além do cuidado extra de garantir que ambos estivessem 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`:

```
sudo cd /etc/puppetlabs/code/environments/production/modules/
sudo mkdir -p foo/manifests
sudo mkdir -p foo/templates
```

Adicione o conteúdo abaixo ao arquivo `/etc/puppetlabs/code/environments/production/modules/foo/manifests/init.pp`:

```
sudo vim /etc/puppetlabs/code/environments/production/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`.

Adicione o conteúdo abaixo ao arquivo `/etc/puppetlabs/code/environments/production/modu les/foo/templates/foo.conf.erb`:

```
sudo vim /etc/puppetlabs/code/environments/production/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/puppetlabs/code/environments/production/modules/foo/templates/foo.conf.erb`.

Usando o módulo `foo` em um host CentOS/Red Hat:

```
sudo 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

sudo cat /tmp/foo.conf

var1=123456
var2=bar

var3=RedHat
```

Dica

 **Concatenando templates**

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

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

19.1 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 @var1 != "foo" %>
<%= @var1 %> 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 **frutas**:

```
<% @frutas.each do |val| %>
Nome: <%= val %>
<% end %>
```

- Iteração em um array com várias chaves:

Exemplo de como o array é formado:

```
$rotas = [
      {id => '100', name => 'link1',},
      {id => '200', name => 'link2',},
       ]
```

Indexando o conteúdo do array num template:

```
<% rotas.each do |rota| -%>
<%= rota['id'] %> <%= rota['name'] %>
<% end -%>
```

Resultado:

```
100 link1
200 link2
```

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' -%> var3=RedHat <% else -%> var3=Outro
<% end -%>
```

Dica

Mais informações sobre a sintaxe ERB

Para saber mais detalhes sobre a sintaxe ERB, acesse a página abaixo.

https://docs.puppet.com/puppet/latest/lang_template_erb.html

Para saber mais detalhes sobre o uso de linguagens para manipulação de templates no Puppet, acesse a página abaixo.

https://docs.puppet.com/puppet/latest/lang_template.html

19.2 Prática: usando templates

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

```
sudo cd /etc/puppetlabs/code/environments/production/modules/
sudo 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    => '0644',
    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 no arquivo `/etc/motd`.

20 Environments

O Puppet permite você dividir os seus sistemas em diferentes conjuntos de hosts, chamados *environments*.

Cada environment pode servir um conjunto diferente de módulos. Isso é usado geralmente para gerenciar versões de módulos, usando-os em sistemas destinados a testes.

O uso de environments introduz uma série de outras possibilidades, como separar um ambiente em DMZ, dividir tarefas entre administradores de sistemas ou dividir o seu parque por tipo de hardware.

O environment de um node é especificado no arquivo `puppet.conf`. Sempre que um node faz um pedido de configuração ao Puppet Master, o environment do node é utilizado para determinar a qual configuração e quais módulos serão fornecidos.

Por padrão, o agente envia ao Puppet Master um environment chamado *production*.

20.1 Prática: configurando environments

Os comandos abaixo são executados no **master.domain.com.br** e no **node1.domain.com.br**

1. No Puppet Master faça uma cópia do ambiente **production** para **desenv**:

```
sudo cd /etc/puppetlabs/code/environments/
sudo cp -a production desenv
```

2. No arquivo `/etc/puppetlabs/puppet/puppet.conf` do `node1.domain.com.br` acrescente o environment:

```
[main]
environment = desenv
```

Dica

Enviroment em linha de comando

Opcionalmente, podemos chamar o agente passando o environment pela linha de comando: `puppet agent -t --environment desenv`.

3. No Puppet Master, vamos modificar a template do módulo `motd` no environment **desenv**, apenas acrescentando uma linha ao final:

```
sudo cd /etc/puppetlabs/code/environments/desenv/modules/motd/templates
sudo echo "Puppet versão <%= @puppetversion -%>" >> motd.erb
```

4. Execute o agente Puppet no `node1.domain.com.br` (certifique-se de que o `node1.domain.com.br` está declarado no arquivo `/etc/puppetlabs/code/environments/desenv/manifests/site.pp` e possui a classe **motd** declarada):

```
sudo puppet agent -t
```

5. Agora temos dois módulos `motd`, um para cada environment. Uma vez que o módulo no environment **desenv** esteja aprovado, ele pode ser copiado para o environment **production** com o comando a seguir.

```
sudo rsync -a /etc/puppetlabs/code/environments/desenv/modules/motd/ \
 /etc/puppetlabs/code/environments/production/modules/motd/
```

6. Nos nodes que são do environment **production**, a nova versão do módulo **motd** será utilizada a partir de agora.

21 Hiera

O Hiera é uma ferramenta que busca por dados de configuração de seus manifests. Ele foi criado pelo R.I.Piennar (que também criou o extlookup).

O Hiera permite separar os dados do código Puppet. Com ele você pode definir dados para diferentes nodes usando o mesmo código. Ao invés dos dados serem armazenados dentro de um manifest, com o Hiera eles serão armazenados externamente.

O Hiera facilita a configuração de seus nodes de forma que podemos ter configurações com dados default ou então vários níveis de hierarquia. Ou seja, com o Hiera você pode definir dados comuns para a maioria dos seus nodes e sobrescrever dados para nodes específicos.

A partir da versão 4.9 do Puppet, foi lançado a versão 5 do Hiera que irá gradativamente substituir a versão 3, que ainda é bastante ultilizada.

21.1 Hiera Datasources

O Hiera suporta nativamente os datasources YAML e JSON. Outros datasources podem ser suportados com plugin adicionais. Veja os links abaixo:

- https://github.com/crayfishx/hiera-http
- https://github.com/crayfishx/hiera-mysql

21.2 Configurando Hiera

As versões 3 e 5 do Hiera são incompatíveis entre si. O arquivo de configuração é diferente para cada versão. A seguir são mostrados os exemplos de configuração para cada versão.

A versão 4 do Hiera foi lançada na versão 4.8 do Puppet, mas logo foi substituída pela versão 5.

21.2.1 No Hiera 3:

A primeira coisa a se fazer é criar o arquivo `/etc/puppetlabs/code/hiera.yaml` e definir a hierarquia de pesquisa, o backend e o local onde ele deverá procurar os arquivos, veja o exemplo abaixo:

```
---
:hierarchy:
   - "host/%{::trusted.certname}"
   - "host/%{::fqdn}"
   - "host/%{::hostname}"
   - "os/%{::osfamily}"
   - "domain/%{::domain}"
   - "common"
:backends:
   - yaml
:yaml:
   :datadir: '/etc/puppetlabs/code/environments/%{::environment}/hieradata/'
```

Da forma como o arquivo está definido, o Hiera irá procurar pelos dados dentro do diretório `hieradata` de cada environment.

21.2.2 No Hiera 5:

A primeira coisa a se fazer é remover os arquivos `/etc/puppetlabs/code/hiera.yaml` e `/etc/puppetlabs/code/hiera.yaml` e criar o arquivo `/etc/puppetlabs/code/environments/production/hiera.yaml` para definir a hierarquia de pesquisa apenas dentro do environment `production`. Veja o exemplo de configuração abaixo:

```
---
version: 5
defaults:
  datadir: hieradata
  data_hash: yaml_data
hierarchy:
  - name: "Hosts"
    paths:
      - "host/%{::trusted.certname}.yaml"
      - "host/%{::facts.networking.fqdn}.yaml"
      - "host/%{::facts.networking.hostname}.yaml"
  - name: "Dominios"
    paths:
      - "domain/%{::trusted.domain}.yaml"
      - "domain/%{::domain}.yaml"
  - name: "Tipo de S.O"
    path: "os/%{::osfamily}.yaml"
  - name: "Dados comuns"
    path: "common.yaml"
```

Com o Hiera 5, cada environment pode ter um arquivo de configuração `hiera.yaml` diferente especificando uma hierarquia de pesquisa e organização dos dados sem interferir na configuração dos demais environments.

Observe que na configuração do arquivo `hiera.yaml`, tanto na versão 3 como na 5, estamos definindo a seguinte sequencia de pesquisa:

- host/certname
- host/fqdn
- host/hostname
- os/osfamily
- domain/domain
- common

E definimos também que o backend é o YAML e que os arquivos ficarão no diretório `/etc/puppetlabs/code/environments/production/hieradata/`. Se esse diretório não existir, crie-o.

```
sudo mkdir -p /etc/puppetlabs/code/environments/production/hieradata/
```

Dentro desse diretório ficarão os diretórios e arquivos a seguir, por exemplo:

- host/node1.domain.com.br.yaml
- host/node2.yaml

- os/ubuntu.yaml
- os/redhat.yaml
- domain/domain.com.br.yaml
- common.yaml

```
sudo mkdir -p /etc/puppetlabs/code/environments/production/hieradata/os
sudo mkdir -p /etc/puppetlabs/code/environments/production/hieradata/host
sudo mkdir -p /etc/puppetlabs/code/environments/production/hieradata/domain
sudo cd /etc/puppetlabs/code/environments/production/hieradata/
sudo touch os/ubuntu.yaml
sudo touch os/redhat.yaml
sudo touch domain/domain.com.br.yaml
sudo touch host/node2.yaml
sudo touch host/node1.domain.com.br.yaml
sudo touch common.yaml
```

Dentro de cada arquivo YAML, são definidos os valores para as variáveis a serem usadas nos manifests. Essas variáveis podem ter valores diferentes para cada arquivo especificado no exemplo acima. Se houverem variáveis com o mesmo nome e valores diferentes em vários arquivos, o Hiera seguirá a ordem de prioridade da hierarquia dos dados que definimos no arquivo `hiera.yaml`. A seguir está o exemplo do conteúdo de cada arquivo.

Exemplo do conteúdo do arquivo `host/node1.domain.com.br.yaml`:

```
#SSH
ssh_port: '22'
ssh_allow_users: 'puppetbr teste'

#Postfix
smtp_port: '25'
smtp_server: '127.0.0.1'

#Diretorio de conteudos
content_dir:
    - '/home/puppetbr'
    - '/home/puppetbr/content2/'
```

Exemplo do conteúdo do arquivo `host/node2.yaml`:

```
#SSH
ssh_port: '2220'
ssh_allow_users: 'teste'

#Postfix
smtp_port: '587'
```

Exemplo do conteúdo do arquivo `domain/domain.com.br.yaml`:

```
scripts_version: 2.0
```

Exemplo do conteúdo do arquivo `os/ubuntu.yaml`:

```
#Apache
apache_service: apache2
```

Exemplo do conteúdo do arquivo `os/redhat.yaml`:

```
#Apache
apache_service: httpd
```

Exemplo do conteúdo do arquivo `common.yaml`:

```
#Apache
apache_service: apache2

#SSH
ssh_port: '22'
ssh_allow_users: 'puppetbr teste'

#Postfix
smtp_port: '25'
smtp_server: '127.0.0.1'

#Diretorio de conteudos
content_dir:
    - '/home/puppetbr'
    - '/home/puppetbr/content/'
config_package: 'config.tar.bz2'
deploy_scripts: true
scripts_version: 1.0
```

Usando o exemplo dado anteriormente, se queremos obter um valor definido para a variável `apache_service`, o Hiera tentará obter este valor lendo a seguinte sequencia de arquivos e retornará o primeiro valor que encontrar para essa variável.

- host/node1.domain.com.br.yaml
- host/node2.yaml
- os/ubuntu.yaml
- os/redhat.yaml
- domain/domain.com.br.yaml
- common.yaml

Nota

 **Obtendo o certname de um node**

Como já foi visto antes, o certname é definido no arquivo `/etc/puppetlabs/puppet/puppet.conf`. Para ver qual é o certname configurado use o comando: `puppet config print certname`. O certname pode ser diferente do FQDN (Fully Qualified Domain Name).

Depois que o Hiera é configurado, o serviço `puppetserver` precisa ser reiniciado.

```
sudo service puppetserver restart
```

21.3 Comandos e consultas Hiera

Execute o hiera para uma pesquisa seguindo a hierarquia definida.

```
sudo hiera apache_service
```

Execute o hiera especificando parâmetros de busca:

```
sudo hiera apache_service -yaml ubuntu.yaml
```

É bem simples fazer a pesquisa e testar se vai retornar o que você está esperando. O Hiera retornará o valor `nil` quando não encontrar um valor para a variável especificada na busca.

Nota

 **Mais documentação sobre o Hiera**

Mais informações sobre o Hiera podem ser encontradas nestas páginas:

https://docs.puppet.com/hiera/latest/
https://docs.puppet.com/puppet/4.9/hiera_migrate_environments.html
https://docs.puppet.com/puppet/4.9/hiera_config_yaml_5.html
https://docs.puppet.com/hiera/3.3/configuring.html
https://docs.puppet.com/puppet/4.9/hiera_intro.html
https://docs.puppet.com/puppet/4.9/hiera_migrate_v3_yaml.html
https://docs.puppet.com/puppet/4.9/hiera_migrate.html
https://docs.puppet.com/puppet/latest/lookup_quick.html

Você também pode usar o `puppet lookup` para testar a busca do Hiera. Veja o exemplo abaixo.

```
puppet lookup --debug --explain --node node1.domain.com.br \
 --environment production nome_variavel_a_ser_testada
```

O puppet lookup só servirá para os testes se alguma vez o node tiver estabelecido comunicação com o puppet server.

> ***Nota***
>
> **Mais documentação sobre o Puppet Lookup**
>
> Mais informações sobre o `puppet lookup` podem ser encontradas nesta página: https://docs.puppet.com/puppet/latest/man/lookup.html

21.4 Criando um módulo para usar dados vindos do Hiera

Agora que já configuramos o Hiera para localizar os dados, vamos criar um módulo que irá utilizá-los e também definirá os valores padrão para as variáveis, caso não seja possível obtê-los via Hiera.

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

```
sudo cd /etc/puppetlabs/code/environments/production/modules
sudo mkdir -p doc/manifests
sudo mkdir -p doc/templates
```

2. O nosso módulo `doc` terá dois manifests: o `init.pp` (código principal) e o `params.pp` (apenas para declaração de variáveis).

```
sudo vim doc/manifests/init.pp
```

```
class doc(

  #Usando as variaveis definidas no manifest params.pp
  $apache_service  = $doc::params::apache_service,
  $ssh_port        = $doc::params::ssh_port,
  $ssh_allow_users = $doc::params::ssh_allow_users,
  $smtp_port       = $doc::params::smtp_port,
  $smtp_server     = $doc::params::smtp_server,
  $content_dir     = $doc::params::content_dir,
  $config_package  = $doc::params::config_package,
  $deploy_scripts  = $doc::params::deploy_scripts,
  $scripts_version = $doc::params::scripts_version,
  ) inherits doc::params {
```

```
    file { '/tmp/doc.txt':
      ensure  => 'file',
      content => template("doc/documentation.txt.erb"),
      mode    => '0644',
      owner   => 'root',
      group   => 'root',
    }
}
```

```
sudo vim doc/manifests/params.pp
```

```
class doc::params {

  #Variaveis gerais
  $content_dir     = hiera('content_dir', ['/home/puppetbr',
                                           '/home/puppetbr/content/'])
  $config_package  = hiera('config_package', 'config.tar.bz2')
  $deploy_scripts  = hiera('deploy_scripts', true)
  $scripts_version = hiera('scripts_version', '1.0')

  #Apache
  $apache_service = hiera('apache_service', 'apache2')

  #SSH
  $ssh_port        = hiera('ssh_port', '22')
  $ssh_allow_users = hiera('ssh_allow_users', 'puppetbr teste')

  #SMTP
  $smtp_server = hiera('smtp_server', '127.0.0.1')
  $smtp_port   = hiera('smtp_port', '25')
}
```

```
sudo vim doc/templates/documentation.txt.erb
```

```
#Informacoes sobre SSH
SSH_PORT=<%= @ssh_port %>
Usuario que podem acessar o SSH=<%= @ssh_allow_users %>
Distribuição GNU/Linux=<%= @osfamily %>
Hostname=<%= @hostname %>
Qual é o nome do processo do Apache nesta distro? <%= @apache_service %>
#Informacoes sobre o servico de envio de email
SMTP_PORT=<%= @smtp_port %>
SMTP_SERVER=<%= @smtp_server %>
Diretorio de conteudos=<%= @content_dir %>
#Informacoes sobre a atualizacao do Script
PACKAGE=<%= @config_package %>
ENABLE_DEPLOY=<%= @deploy_scripts %>
PACKAGE_VERSION=<%= @scripts_version %>
```

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

```
sudo vim /etc/puppetlabs/code/environments/production/manifests/site.pp
```

```
node 'node1.domain.com.br' {
  include doc
}
```

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

```
sudo puppet agent -t
```

Agora veja o conteúdo do arquivo `/tmp/doc.txt` e observe se o conteúdo está como o esperado.

```
#Informacoes sobre SSH
SSH_PORT=22
Usuario que podem acessar o SSH=puppetbr teste
Distribuição GNU/Linux=Debian
Hostname=node1
Qual é o nome do processo do Apache nesta distro? apache2
#Informacoes sobre o servico de envio de email
SMTP_PORT=25
SMTP_SERVER=127.0.0.1
Diretorio de conteudos=["/home/puppetbr", "/home/puppetbr/content2/"]
#Informacoes sobre a atualizacao do Script
PACKAGE=config.tar.bz2
ENABLE_DEPLOY=true
PACKAGE_VERSION=1.0
```

5. Em **master.domain.com.br** mova o arquivo `node1.domain.com.br.yaml` para `/root/manifests`.

```
sudo cd /etc/puppetlabs/code/environments/production/hieradata/host/
sudo mv node1.domain.com.br.yaml /root/manifests.
```

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

```
sudo puppet agent -t
```

Agora observe o que mudou no conteúdo do arquivo `/tmp/doc.txt`.

22 Recursos Virtuais

A declaração de recurso virtual especifica o estado desejado para um recurso sem necessariamente impor esse estado. Você pode dizer ao Puppet para gerenciar o recurso virtual através da função `realize` em qualquer lugar dos seus manifests que fazem parte do seu módulo.

Embora os recursos virtuais só possam ser declarados uma vez, você pode usar a função `realize` quantas vezes for necessária, mesmo que seja numa mesma classe ou manifest.

22.1 Objetivo

Os recursos virtuais são úteis para:

- Recursos cuja gestão depende de pelo menos uma das múltiplas condições que estão sendo atendidas.
- Sobreposição de conjuntos de recursos que podem ser necessários em várias classes.
- Recursos que só devem ser geridos se forem cumpridas condições entre várias classes.

As características que distinguem os recursos virtuais dos demais são:

- *Searchability* (pesquisa) via coletores de recursos, o que permite que você realize a sobreposição de aglomerados de recursos virtuais.
- *Flatness* (planicidade), de tal forma que você pode declarar um recurso virtual e realizá-lo algumas linhas mais tarde sem ter que encher seus módulos com muitas classes de recurso único.

Um exemplo de uso para recursos virtuais é o gerenciamento de pacotes. Imagine que você tem um módulo no Puppet Master que gerencia a configuração do Apache e que neste módulo existe um manifest que possui o recurso `package` para gerenciar a instalação do pacote `php5.6`. Se houver a necessidade de gerenciar o PHP através de outro módulo e que nele também exista o recurso `package` para gerenciar a instalação do pacote `php5.6`, ao compilar um catálogo, contendo o estado desejado através destes dois módulos, o Puppet irá mostrar um erro de compilação do catálogo e informará que o recurso `package` está duplicado, mesmo que o recurso tenha sido declarado em módulos diferentes.

Para resolver este problema você pode criar um módulo para gerenciar pacotes e neste módulo você pode declarar um recurso virtual para gerenciar o pacote `php5.6`. Nos módulos que gerenciam o Apache e o PHP, você utiliza a função `realize` para realizar o estado desejado do recurso virtual. Veja os detalhes no trecho de código abaixo.

```
#Classe pertecente ao módulo packages
class packages {

  #Declarando um recurso virtual, o pacote nao sera instalado neste ponto.
  @package{ 'php5.6':
    ensure => installed,
  }
}


#Classe pertecente ao módulo apache
class apache {

  #Incluindo as classes do modulo packages
  include packages
  ...
```

```
  #Realizando um recurso virtual, o pacote é instalado neste ponto, \
  # caso nao esteja instalado.
  realize(Package['php5.6'])

  ...
}

#Classe pertecente ao módulo apache
class php {

  #Incluindo as classes do modulo packages
  include packages
  ...

  #Realizando um recurso virtual, o pacote é instalado neste ponto, \
  # caso nao esteja instalado.
  realize(Package['php5.6'])

  ...
}
```

Quando o catálogo for compilado, o Puppet não exibirá erros de compilação porque o recurso `package`, que gerencia o pacote `php5.6`, só foi declarado uma vez como sendo um recurso virtual no módulo `packages`. Os módulos `apache` e `php` não estão declarando o recurso, estão apenas referenciando o recurso virtual para realizar o estado desejado.

22.2 Sintaxe

Para declarar um recurso virtual, você usa o prefixo (`@`) antes do nome do recurso. Exemplo:

```
#Declarando um recurso virtual para um usuario
@user {'deploy':
  uid     => 2004,
  comment => 'Deployment User',
  group   => 'www-data',
  groups  => ["enterprise"],
  tag     => [deploy, web],
}

#Declarando um recurso virtual para um array de pacotes
@package {[
  'build-essential',
  'gcc',
  'gcc-c++',
  'g++',
  'autoconf',
  ]:
    ensure => installed,
}
```

Para realizar um ou mais recursos virtuais, use a função `realize`, que aceita uma ou mais referências de recursos. Exemplo:

```
realize(Package['gcc'],
        Package['autoconf'],
        Package['gcc-c++'],
        Package['g++'],
        Package['build-essential'],
        User['deploy'],
       )
```

Mesmo que a função `realize` referencie várias vezes o mesmo recurso virtual no mesmo manifest, o recurso só será gerenciado apenas uma vez.

Se um recurso virtual estiver declarado em uma classe, ele não poderá ser realizado na mesma, a menos que a classe seja declarada ou referênciada por outra classe ou módulo. Os recursos virtuais que não forem realizados continuarão disponíveis no catálogo, mas eles estarão marcados como inativos. A função `realize` falhará na compilação do catálogo se você tentar realizar um recurso virtual que não foi declarado ou se foi declarado em uma classe ou módulo que em nenhum momento foi referenciado.

Nota

Mais informações sobre recursos virtuais

Para obter mais informações sobre os recursos virtuais, acesse a página abaixo. https://docs.puppet.com/puppet/latest/lang_virtual.html

22.3 Prática: usando recursos virtuais

1. Acesse o servidor Puppet Master. Crie a estrutura básica de um módulo `rvirtual`:

```
sudo cd /etc/puppetlabs/code/environments/production/modules
sudo mkdir -p rvirtual/manifests
sudo mkdir -p rvirtual/templates
```

2. O módulo `rvirtual` terá um manifest: o `init.pp` (código principal). Nele declare os recursos virtuais abaixo.

```
sudo vim rvirtual/manifests/init.pp
```

```
class rvirtual{

   @package {[
     'nfs-utils',
     'nfs-utils-lib',
     'nfs-common',
     ]:
     ensure => installed,
```

```
   }

  @file { '/media/storage':
    ensure  => 'directory',
    mode    => '755',
    owner   => root,
    group   => root,
  }

   @file { '/media/storage/doc.txt':
       ensure  => 'file',
       content => template("rvirtual/doc.txt.erb"),
       mode    => '0644',
       owner   => 'root',
       group   => 'root',
       require => File['/media/storage'],
   }
}
```

3. Informe o conteúdo abaixo no arquivo de template `rvirtual/templates/doc.txt.erb`.

```
#Informacoes sobre o host
Distribuição GNU/Linux=<%= @osfamily %>
Hostname=<%= @hostname %>
```

4. Crie outra estrutura básica para o módulo `mount`:

```
sudo cd /etc/puppetlabs/code/environments/production/modules
sudo mkdir -p mount/manifests
```

5. O módulo `mount` terá um manifest: o `init.pp` (código principal). Nele informe o seguinte conteúdo.

```
sudo vim mount/manifests/init.pp
```

```
class mount{

  include rvirtual

  case $::operatingsystem {
    'ubuntu': {
       realize(Package['nfs-common'])
    }
    'redhat': {
       realize(Package['nfs-utils','nfs-utils-lib'])
    }
    default: {
     fail('[ERRO] S.O NAO suportado.')
    }
  }
```

```
   realize(File['/media/storage/doc.txt'],
           File['/media/storage'], )

   mount { '/media/storage':
    device  => "192.168.100.13:/home/m2",
    fstype  => 'nfs',
    ensure  => 'mounted',
    options => 'rw',
    atboot  => true,
    before  => File['/media/storage/doc.txt'],
  }
}
```

Aviso

Configurar pontos de montagem via NFS

Para realizar este exercício, será necessário que você configure o NFSv3 num host remoto e compartilhe dois diretórios, com permissão de leitura e escrita para a montagem de diretório remoto.

Na Internet você encontra vários tutoriais explicando como fazer isso. Abaixo estão alguns deles.

Ubuntu:
https://www.digitalocean.com/community/tutorials/how-to-set-up-an-nfs-mount-on-ubuntu-16-04

CentOS/Red Hat 7: https://goo.gl/3NqOs2

6. Deixe o conteúdo do arquivo `site.pp` dessa maneira:

```
sudo vim /etc/puppetlabs/code/environments/production/manifests/site.pp
```

```
node 'node1.domain.com.br' {
  include mount
}
```

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

```
sudo puppet agent -t
```

Agora veja se no diretório `/media/storage/` existe o arquivo `doc.txt`.

23 Tags

Os recursos, as classes e instâncias de tipos definidos podem ter várias tags associadas, mas eles recebem algumas tags automaticamente. As tags são úteis para:

- coletar recursos;
- filtrar e analisar relatórios;
- restringir a execução de parte do catálogo.

23.1 Nomes de tag

As tags devem começar com uma letra minúscula, número ou sublinhado, e podem incluir:

- letras minúsculas;
- numeros;
- sublinhados (_);
- dois pontos (:);
- ponto (.);
- Hifens (-);

Os nomes das tags devem corresponder à seguinte expressão regular:

```
\A[a-z0-9_][a-z0-9_:\.\-]*\Z
```

23.2 Atribuição de tags a recursos

Um recurso pode ter várias tags. Há três maneiras de atribuir uma tag a um recurso. Vamos conhecer os detalhes de cada uma delas.

23.2.1 Tag automática

Cada recurso recebe automaticamente as seguintes tags:

- seu tipo de recurso;
- o nome completo da classe e/ou tipo definido em que o recurso foi declarado;
- cada `namespace segment` (https://docs.puppet.com/puppet/latest/lang_namespaces.html) da classe do recurso e/ou tipo definido.

Por exemplo, um recurso `file` na classe `apache::ssl` recebe as tags: `file`, `apache::ssl`, `apache` e `ssl`.

As tags de classe são geralmente as mais úteis, especialmente quando você configura o módulo Puppet Tagmail ou está testando manifests refatorados.

23.2.2 O meta-parâmetro `tag`

Você pode usar o meta-parâmetro `tag` em uma declaração de recurso para adicionar várias tags:

```
apache::vhost {'docs.puppetlabs.com':
  port => 80,
```

```
  tag  => ['us_mirror1', 'us_mirror2'],
}
```

O meta-parâmetro `tag` pode aceitar uma única tag ou um array de tags. Estas tags serão adicionados às tags associadas automaticamente. Além disso, esse meta-parâmetro pode ser utilizado com recursos normais, recursos definidos e classes.

O exemplo acima atribui as tags `us_mirror1` e `us_mirror2` para cada recurso contido na classe `Apache::Vhost['docs.puppetlabs.com']`.

23.2.3 A função `tag`

Você pode usar a função `tag` dentro de uma definição de classe ou tipo definido para atribuir tags para o recipiente envolvente e todos os recursos que ele contém. Exemplo:

```
class role::public_web {
  tag 'us_mirror1', 'us_mirror2'

  apache::vhost {'docs.puppetlabs.com':
    port => 80,
  }
  ssh::allowgroup {'www-data': }
  @@nagios::website {'docs.puppetlabs.com': }
}
```

No exemplo acima, as tags `us_mirror1` e `us_mirror2` são atribuídas a todos os recursos definidos na classe `role::public_web`, bem como a ela mesma.

23.3 Usando as tags

As tags podem ser usadas como um atributo na expressão de busca de um colector de recursos. Isto é útil principalmente para realizar Recursos Virtuais e exportados (https://docs.puppet.com/puppet/latest/lang_exported.html).

Você também pode fazer com que o agente do Puppet aplique o estado desejado em um host usando no catálogo apenas a configuração associada a determinadas tags. Isso é útil ao refatorar módulos e permite que você aplique somente uma única classe em um nó de teste.

As tags de configuração podem ser definidas no arquivo `puppet.conf` (para restringir permanentemente o catálogo) ou na linha de comando (para restringir temporariamente):

```
puppet agent --test --tags apache,us_mirror1
```

As tags configuração devem ser separadas por vírgulas (sem espaços entre as tags).

O módulo Puppet Tagmail pode enviar e-mails para uma lista de pessoas sempre que recursos com determinadas tags forem alteradas.

As tags de recurso também estão disponíveis para serem usadas em manipuladores de relatórios personalizados. Veja mais detalhes nos links abaixo.

https://docs.puppet.com/puppet/latest/lang_tags.html
https://docs.puppet.com/puppet/latest/reporting_about.html
https://docs.puppet.com/puppet/latest/format_report.html

24 Recursos Exportados

Uma declaração de recursos exportados especifica um estado desejado para um recurso, mas não gerencia o recurso no sistema destino. Ela apenas publica o recurso para ser utilizado por outros nós. Qualquer nó (incluindo o nó no qual o recurso é exportado) pode então coletar o recurso exportado e gerenciar uma cópia do mesmo.

24.1 Objetivo

Os recursos exportados permitem que o compilador Puppet compartilhe informações entre nós, combinando informações de catálogos de vários nós. Isso ajuda a gerenciar coisas que dependem do conhecimento dos estados ou atividade de outros nós.

> ***Nota***
>
> 
> **Uso de dados pelos recursos exportados**
>
> Os recursos exportados permitem ao compilador ter acesso à informação, e não podem usar informações que nunca foram enviadas ao compilador, tais como: o conteúdo dos arquivos de um nó.

Os casos de uso mais comuns são: o monitoramento e os backups. Uma classe que gerencia um serviço como o PostgreSQL pode exportar o recurso `nagios_service`, que descreve como monitorar o serviço, incluindo informações como o nome do host e a porta. O servidor Nagios pode então coletar todos os recursos `nagios_service`, e iniciar automaticamente o monitoramento do servidor Postgres.

24.2 Sintaxe

O uso de recursos exportados requer duas etapas: declarar e coletar.

```
class ssh {
  # Declarando a exportação de um recurso:
  @@sshkey { $::hostname:
    type => dsa,
    key  => $::sshdsakey,
  }
  # Coletando os recursos 'sshkey' exportados:
  Sshkey <<| |>>
}
```

No exemplo acima, cada nó com a classe `ssh` irá exportar a sua própria chave de host SSH e depois coletar a chave de host SSH de cada nó (incluindo o seu próprio). Isso fará com que cada nó confie nas conexões SSH de todos os outros nós.

24.2.1 Declarando um recurso exportado

Para declarar um recurso exportado, inclua (`@@`) (um duplo "arroba") antes do tipo de recurso. Exemplo:

```
@@nagios_service { "check_zfs${::hostname}":
  use                 => 'generic-service',
  host_name           => $::fqdn,
  check_command       => 'check_nrpe_1arg!check_zfs',
  service_description => "check_zfs${::hostname}",
  target              => '/etc/nagios3/conf.d/nagios_service.cfg',
  notify              => Service[$nagios::params::nagios_service],
}
```

24.2.2 Coleta de recursos exportados

Para fazer a coleta de recursos exportados você deve usar um coletor de recursos exportados. Exemplo:

```
# Coleta todos os recursos ``nagios_service resources`` exportados
Nagios_service <<| |>>

#Coleta apenas o recurso exportado que contém determinada tag
Concat::Fragment <<| tag == "bacula-storage-dir-${bacula_director}" |>>
```

Veja mais detalhes sobre os coletores de recursos exportados na página: https://docs.puppet.com/puppet/latest/lang_collectors.html#exported-resource-collectors e também no capítulo sobre Coletores de Recursos.

Cada recurso exportado deve ser globalmente exclusivo em cada nó. Se dois recursos forem exportados no mesmo nó com o mesmo título ou mesmo nome/namevar ao tentar coletá-los, a compilação irá falhar.

Para garantir a exclusividade, cada recurso que você exporta deve incluir uma substring exclusiva para o nó que o exporta para seu título e nome/namevar. A maneira mais conveniente é usar fatos como: o hostname ou fqdn.

Os coletores de recursos exportados não coletam recursos normais ou virtuais. Em particular, eles não podem recuperar recursos *não exportados* de outros catálogos de nós.

24.2.3 Recursos exportados com Nagios

O exemplo a seguir mostra tipos nativos de Puppet para gerenciar arquivos de configuração do Nagios. Esses tipos se tornam muito poderosos quando você exporta e os coleta. Por exemplo, você poderia criar uma classe para algo como o Apache que adiciona uma definição de serviço no seu host Nagios, monitorando automaticamente o servidor web:

```
sudo vim /etc/puppetlabs/puppet/modules/nagios/manifests/target/apache.pp
```

```
class nagios::target::apache {
  @@nagios_host { $::fqdn:
    ensure  => present,
    alias   => $::hostname,
    address => $::ipaddress,
    use     => 'generic-host',
  }
```

```
  @@nagios_service { "check_ping_${::hostname}":
    check_command       => 'check_ping!100.0,20%!500.0,60%',
    use                 => 'generic-service',
    host_name           => $::fqdn,
    notification_period => '24x7',
    service_description => "${::hostname}_check_ping"
  }
}
```

```
sudo vim /etc/puppetlabs/puppet/modules/nagios/manifests/monitor.pp
```

```
class nagios::monitor {
  package { [ 'nagios', 'nagios-plugins' ]: ensure => installed, }
  service { 'nagios':
    ensure     => running,
    enable     => true,
    #subscribe => File[$nagios_cfgdir],
    require    => Package['nagios'],
  }
  # Coletando recursos e populando o arquivo /etc/nagios/nagios_*.cfg
  Nagios_host <<||>>
  Nagios_service <<||>>
}
```

Nota

Mais informações sobre recursos exportados

Para obter mais informações sobre os recursos exportados acesse a página abaixo.

https://docs.puppet.com/puppet/latest/lang_exported.html

25 Coletores de Recursos

Coletores de recursos selecionam um grupo de recursos, pesquisando os atributos de cada recurso no catálogo. Esta pesquisa é independente da ordem de avaliação (ou seja, inclui recursos que ainda não foram declarados no momento em que o coletor é escrito). Os coletores percebem recursos virtuais, podendo ser usados em declarações de encadeamento, e podem substituir atributos de recursos.

Os coletores têm uma sintaxe irregular que permite que eles funcionem como uma declaração e um valor.

25.1 Sintaxe

```
#Coleta um único recurso de usuário que possua o título 'luke'
User <| title == 'luke' |>

#Coleta qualquer recurso de usuario da lista que inclui o grupo 'admin'
User <| groups == 'admin' |>

#Cria um relacionamento com os recursos ``package``
Yumrepo['custom_packages'] -> Package <| tag == 'custom' |>
```

A forma geral de um coletor de recursos é:

- O tipo de recurso, começando com a letra maiúscula. Não pode ser uma classe e não há maneira de coletar classes.
- <| - Um sinal de "menor que" (<) e um caractere de pipe (|) .
- Opcionalmente, uma expressão de pesquisa.
- |> - Um caractere de pipe e um sinal de "maior que" (>).

Observe que os coletores de recursos exportados têm uma sintaxe ligeiramente diferente. Veremos isso mais adiante.

25.2 Expressões de pesquisa

Os coletores podem pesquisar os valores dos títulos e atributos de recursos usando uma sintaxe de expressão especial. Um coletor com uma expressão de pesquisa vazio irá corresponder a todos os recursos do tipo de recurso especificado.

Parênteses podem ser usados para melhorar a legibilidade e para modificar a prioridade, bem como agrupamento de and / or . Você pode criar expressões arbitrariamente complexas usando os quatro operadores a seguir:

- == #igual
- != #diferente
- and #e
- or #ou

Os coletores de recursos podem ser usados como declarações independentes, como o operando de uma declaração de encadeamento ou em um bloco de atributo coletor para alteração de atributos de recursos.

Os colectores não podem ser usados nos seguintes contextos:

- como o valor de um atributo de recurso;

- como o argumento de uma função;
- dentro de um array ou hash;
- como o operando de uma expressão diferente de uma declaração de encadeamento.

Os coletores de recursos exportados são idênticos aos coletores, exceto por possuírem os sinais (<) e (>) duplicados. Exemplo:

```
#coleta todos os recursos exportados do tipo ``nagios_service``
Nagios_service <<| |>>
```

Coletores de recursos exportados existem apenas para importar recursos que foram publicados por outros nós. Para usá-los, você precisa ter o armazenamento de catálogo e pesquisa (`storeconfigs`) habilitado. Veja o capítulo de Recursos exportados para mais detalhes. Para habilitar os recursos exportados, siga as instruções de instalação e configuração no capítulo PuppetDB e Dashboards Web.

Nota

Mais informações sobre os coletores

Para obter mais informações sobre os coletores de recursos acesse a página abaixo.

https://docs.puppet.com/puppet/latest/lang_collectors.html

26 Augeas

Muitas vezes precisamos manipular arquivos de configuração e, geralmente, recorremos a soluções simples usando grep, sed, awk ou alguma linguagem de script.

Com isso, tem-se muito trabalho para manipular esses arquivos e o resultado final nunca é flexível ou muito confiável.

O Augeas é uma ferramenta para edição segura de arquivos de configuração, que analisa arquivos de configuração em seus formatos nativos e os transforma em uma árvore. As alterações são feitas manipulando essa árvore e salvando-a de volta ao formato nativo do arquivo de configuração. Usando o Augeas ficamos livres de problemas como tratar linhas em branco ou com comentários.

Para dar suporte a diversos formatos de arquivos de configuração, o Augeas usa o que ele chama de *Lenses* (lentes). Uma lense é um registro de como um arquivo de configuração deve ser suportado pelo Augeas, e atualmente são suportados mais de 100 formatos diferentes.

26.1 Usando o Augeas

O comando `augtool` é um pequeno interpretador de comandos e, através dele, podemos manipular de diversas formas arquivos de configuração.

O pacote `puppet-agent` já traz o `augtool` em `/opt/puppetlabs/puppet/bin`.

Vejamos como o arquivo `/etc/resolv.conf` está configurado:

```
cat /etc/resolv.conf

domain domain.com.br
search domain.com.br
nameserver 8.8.8.8
nameserver 8.8.4.4
```

Como o Augeas o representa:

```
/opt/puppetlabs/puppet/bin/augtool print /files/etc/resolv.conf

/files/etc/resolv.conf
/files/etc/resolv.conf/domain = "domain.com.br"
/files/etc/resolv.conf/search
/files/etc/resolv.conf/search/domain = "domain.com.br"
/files/etc/resolv.conf/nameserver[1] = "8.8.8.8"
/files/etc/resolv.conf/nameserver[2] = "8.8.4.4"
```

O Augeas monta uma estrutura hierárquica da configuração.

Usando um caminho completo da configuração, podemos manipular os arquivos sem recorrer a edição manual.

Vamos trocar o valor da opção `domain` do `resolv.conf` (a opção `-s` diz ao `augtool` para salvar a alteração):

```
sudo /opt/puppetlabs/puppet/bin/augtool -s set \
   /files/etc/resolv.conf/domain outrodominio

Saved 1 file(s)

cat /etc/resolv.conf

domain outrodominio
search domain.com.br
nameserver 8.8.8.8
nameserver 8.8.4.4
```

Em um arquivo `resolv.conf` a opção `nameserver` pode aparecer mais de uma vez, pois podemos configurar vários servidores de nomes em nosso sistema. Devido a isso, o Augeas trata a opção `nameserver` como um vetor, então *nameserver[1]* é `8.8.8.8` e *nameserver[2]* é `8.8.4.4`.

Podemos incluir e remover valores no vetor. Por exemplo, adicionar um terceiro nameserver e depois removê-lo:

```
sudo /opt/puppetlabs/puppet/bin/augtool -s set \
    /files/etc/resolv.conf/nameserver[3] 1.1.1.1

Saved 1 file(s)

cat /etc/resolv.conf

domain outrodominio
search domain.com.br
nameserver 8.8.8.8
nameserver 8.8.4.4
nameserver 1.1.1.1

sudo /opt/puppetlabs/puppet/bin/augtool -s rm \
   /files/etc/resolv.conf/nameserver[3]

rm : /files/etc/resolv.conf/nameserver[3] 1
Saved 1 file(s)

cat /etc/resolv.conf

domain outrodominio
search domain.com.br
nameserver 8.8.8.8
nameserver 8.8.4.4
```

26.1.1 Prática: manipulando o arquivo /etc/hosts

Os comandos abaixo podem ser executados no host **node1.domain.com.br**.

1. Agora vamos utilizar o interpretador de comandos do Augeas, simplesmente executando o `augtool`:

```
sudo /opt/puppetlabs/puppet/bin/augtool
augtool>
```

2. Dentro do interpretador, os comandos `print`, `set`, `rm` funcionam como na linha de comando. Podemos associar o caminho no sistema de arquivos com uma opção de configuração:

augtool> ls /files/etc/resolv.conf

domain = outrodominio search/ = (none) nameserver[1] = 8.8.8.8 nameserver[2] = 8.8.4.4

3. Use o comando `print` no arquivo `/etc/hosts`. Identifique qual é o número do registro do host **node1.domain.com.br**.

```
augtool> print /files/etc/hosts
```

4. De posse do número do registro do host **node1.domain.com.br**, crie um novo alias para o host:

```
augtool> set /files/etc/hosts/NUMERO_DO_HOST/alias[2] node1
augtool> save
Saved 1 file(s)
augtool> quit
```

5. Verifique se **node1** está presente no `/etc/hosts`

26.2 Augeas e Puppet

O Puppet fornece um *resource* para que os poderosos recursos de edição do Augeas possam ser usados nos manifests.

Manipulando o `/etc/resolv.conf`, porém agora com um manifest:

```
augeas {'resolv.conf':
  context => '/files/etc/resolv.conf',
  changes => ['set nameserver[1] 8.8.8.8',
              'set nameserver[2] 8.8.4.4', ],
}
```

Outro exemplo, que garante a configuração correta de `/etc/ssh/sshd_config`:

```
augeas { "sshd_config":
  context => "/files/etc/ssh/sshd_config",
    changes => [
    "set PermitRootLogin no",
    "set RSAAuthentication yes",
    "set PubkeyAuthentication yes",
    "set PasswordAuthentication no",
    "set Port 22221",
  ],
 }
```

Garante que o servidor esteja sempre no runlevel correto:

```
augeas { "runlevel":
  context => "/files/etc/inittab",
  changes => [
    "set id/runlevels 3",
  ],
}
```

27 Puppet no Windows

O suporte a Windows no Puppet vem melhorando a cada nova versão. Mas não é possível hospedar o Puppet Server no Windows, sendo suportado somente o agente.

Praticamente onde é possível criar compatibilidade, os resources do Puppet suportam Windows normalmente. Em alguns casos são necessários certos cuidados devido as diferenças semânticas entre sistemas Unix-like e Windows.

Para obter detalhes sobre a instalação do Puppet Agent no Windows acesse: https://docs.puppet.com/puppet/latest/install_windows.html

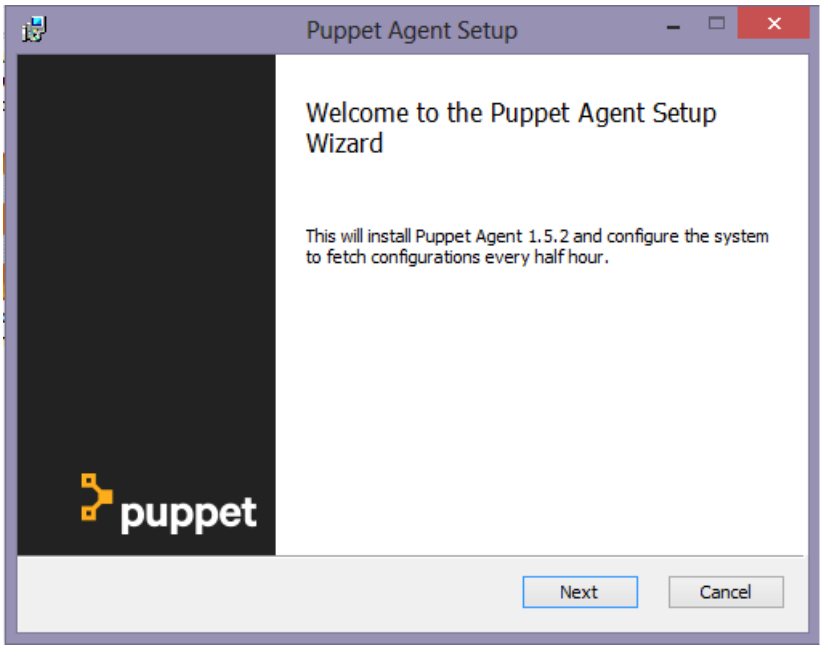
27.1 Prática: Instalação

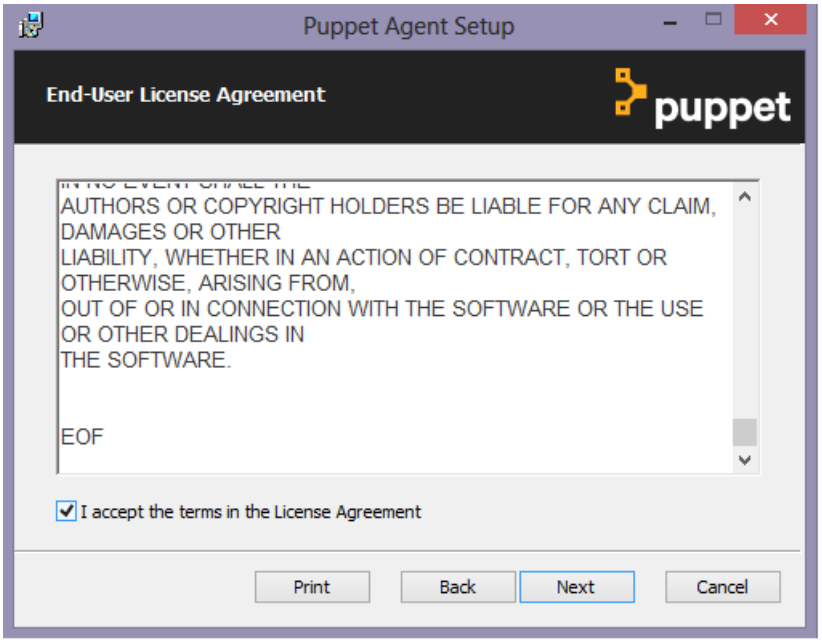
Para a realização dessa prática é necessário você instalar um terceiro host com o Windows 7 ou superior e aqui chamaremos de **win7.domain.com.br**. Também será necessário usar o host **master.domain.com.br**.

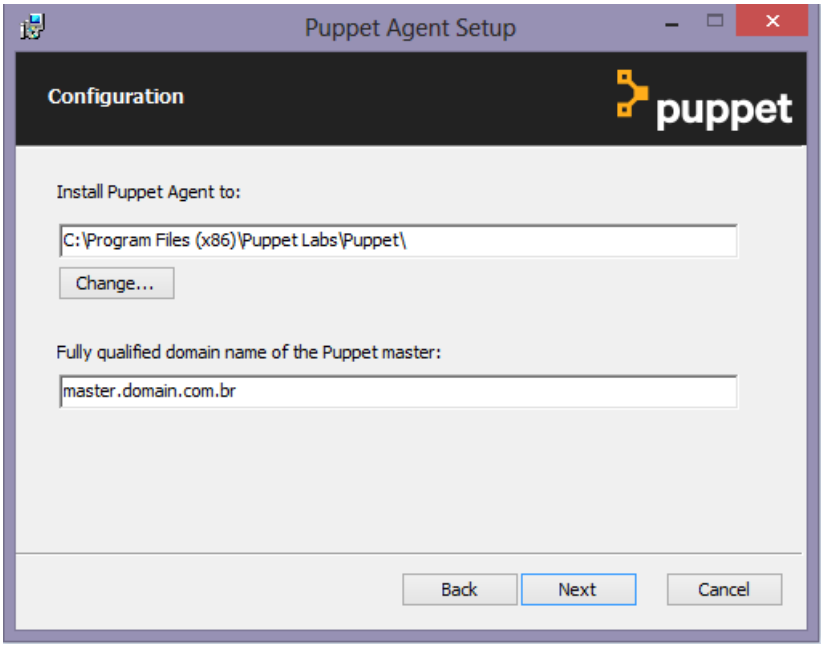
1. Faça login no host **win7.domain.com.br**. Baixe o instalador mais recente do Puppet-Agent para Windows em:

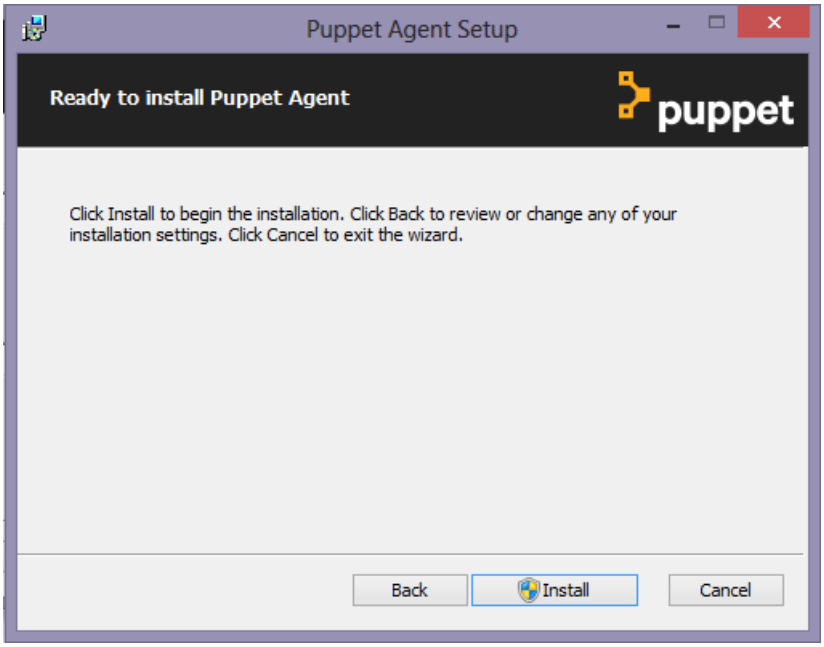
- https://downloads.puppetlabs.com/windows/puppet-agent-x64-latest.msi (Windows 64 bits)
- https://downloads.puppetlabs.com/windows/puppet-agent-x86-latest.msi (Windows 32 bits)

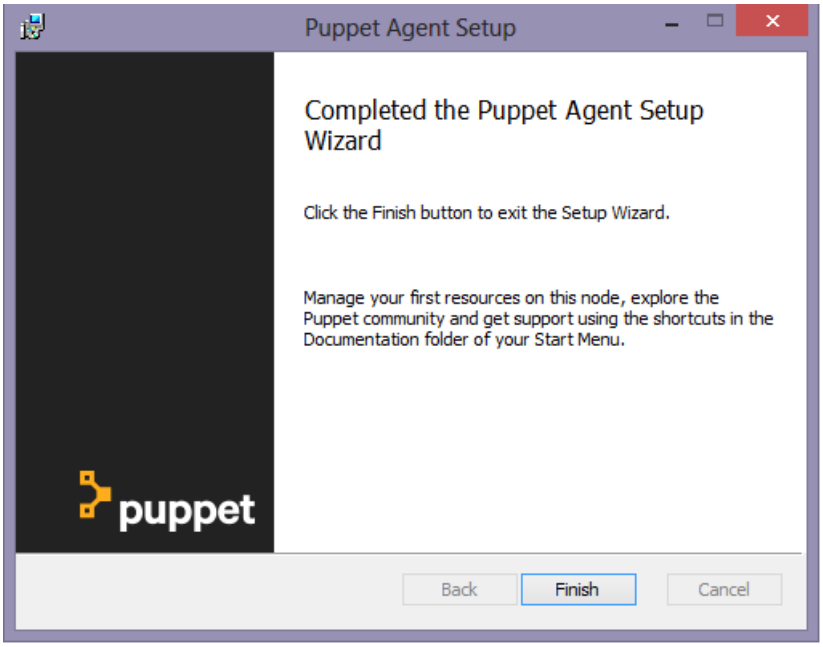
2. Execute o instalador do Puppet-Agent. Aparecerão as telas a seguir. Em uma delas será perguntando qual é o servidor master, preencha com **master.domain.com.br**.











3. Pare o serviço **Puppet**, pois os exercícios serão realizados manualmente.

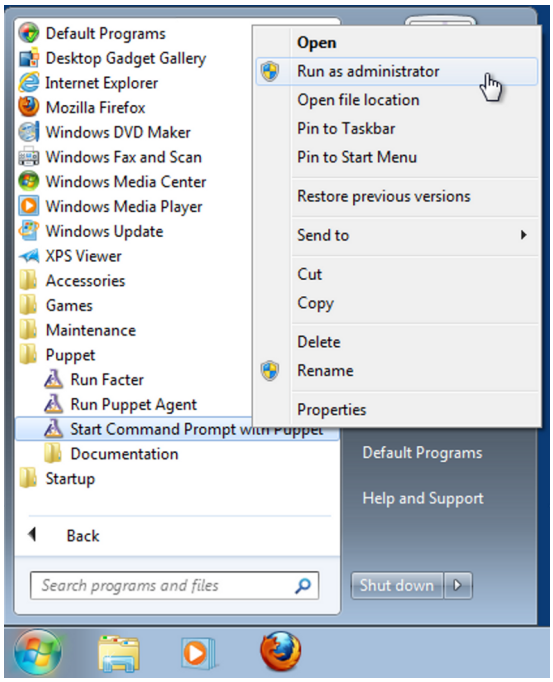
```
sc stop puppet
```

5. Após a instalação, o serviço **Puppet** está em execução e um certificado já foi solicitado ao master. Como de praxe, acesse o master assine o certificado do agente.

```
puppet cert list

  "win7.domain.com.br" (SHA256) E3:6F:64:15:DF:68:A4:21:DA:A3:E2:81:43:3F: ...
```

```
puppet cert sign win7.domain.com.br

Signed certificate request for win7.domain.com.br
Removing file Puppet::SSL::CertificateRequest win7.domain.com.br at \
  '/var/lib/puppet/ssl/ca/requests/win7.domain.com.br.pem'
```

6. Abra um prompt de comando como Administrador no Windows, conforme ilustra a figura abaixo:



7. Execute o agente da mesma maneira que no GNU/Linux.

```
puppet agent -t
```

Nota

Privilégios

No Windows, o Puppet precisa de privilégios elevados para funcionar corretamente, afinal ele precisa configurar o sistema.

O serviço do Puppet é executado com privilégio **LocalSystem**, ou seja, sempre com privilégios elevados.

Quando usar a linha de comando do Windows, é sempre necessário utilizar o Puppet com privilégios elevados.

Dica

Chocolatey

No Windows, você pode usar o chocolatey para facilitar a instalação de pacotes de forma semelhante ao que é feito pelos gerenciadores de pacotes `apt` e `yum` do GNU/Linux. Saiba mais informações nos sites abaixo.

https://chocolatey.org https://puppet.com/blog/chocolatey-using-chocolatey-puppet https://forge.puppet.com/chocolatey/chocolatey

27.2 Prática: resources para Windows

Essa prática será realizada nos hosts **win7.domain.com.br** e **master.domain.com.br**.

1. No host win7.domain.com.br, baixe o pacote http://www.7-zip.org/a/7z1602.exe e copie para c:. Declare o seguinte no arquivo `/etc/puppetlabs/code/environments/production/manifests/site.pp`:

```
node win7.domain.com.br {
  package {'7-Zip 16.02':
    ensure => 'installed',
    source => 'c:\Users\Puppet\Downloads\7z1602.exe',
    install_options => ['/q', { 'INSTALLDIR' => 'C:\Program Files\7-Zip' } ],
  }
}
```

2. Aplique a configuração com o agente (lembre-se de usar um prompt com privilégios elevados).

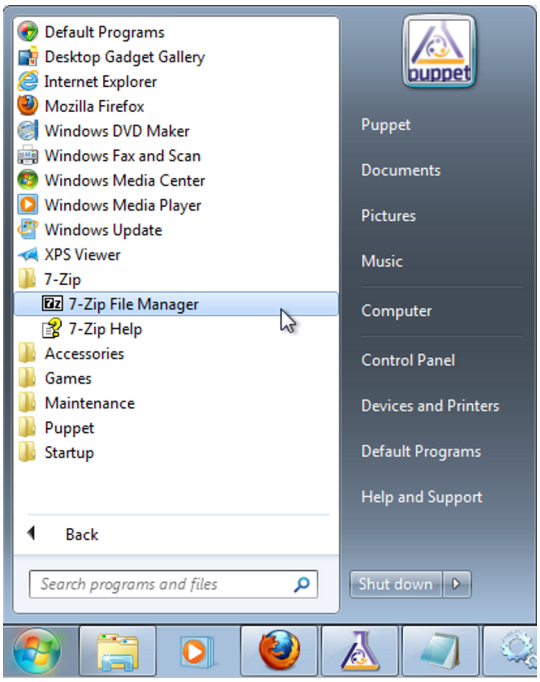
```
puppet agent -t
```

Dica

Título do resource package

O título do resource package precisa ser igual a propriedade *DisplayName* utilizada no registro do Windows para instalação de um pacote MSI. Caso o título seja diferente, o Puppet executará a instalação em todas as execuções.

3. Veja que o 7-Zip foi instalado:



4. Agora vamos configurar um serviço. Acesse o node **master** e adicione o seguinte conteúdo no arquivo `site.pp` para o node **win7.domain.com.br**:

```
service {'Audiosrv':
  ensure => 'stopped',
  enable => false,
}
```

5. Note que o serviço está em execução (terminal com privilégio regular):

```
C:\> sc query audiosrv

SERVICE_NAME: audiosrv
        TYPE               : 20  WIN32_SHARE_PROCESS
        STATE              : 4  RUNNING
                                (STOPPABLE, NOT_PAUSABLE, IGNORES_SHUTDOWN)
        WIN32_EXIT_CODE    : 0  (0x0)
        SERVICE_EXIT_CODE  : 0  (0x0)
        CHECKPOINT         : 0x0
        WAIT_HINT          : 0x0
```

6. Aplique o agente (lembre-se de usar um prompt com privilégios elevados)

```
puppet agent -t
```

7. Veja que o serviço Windows Audio foi parado e desativado.

```
C:\>sc query audiosrv

SERVICE_NAME: audiosrv
        TYPE               : 20  WIN32_SHARE_PROCESS
        STATE              : 1  STOPPED
        WIN32_EXIT_CODE    : 0  (0x0)
        SERVICE_EXIT_CODE  : 0  (0x0)
        CHECKPOINT         : 0x0
        WAIT_HINT          : 0x0
```

Para mais detalhes sobre as diferenças na declaração dos resources no Windows, acesse a página: http://docs.puppetlabs.com/windows/writing.html

27.3 Prática: manipulando o registro

Essa prática é realizada em **win7.domain.com.br** e **master.domain.com.br**.

1. Instale o módulo **puppetlabs-registry** em **master.domain.com.br**:

```
puppet module install puppetlabs/registry

Preparing to install into
   /etc/puppetlabs/code/environments/production/modules ...
Downloading from https://forge.puppetlabs.com ...
Installing -- do not interrupt ...
/etc/puppetlabs/code/environments/production/modules
|-- puppetlabs-registry (v1.1.3)
  |-- puppetlabs-stdlib (v4.12.0)
```

2. Execute o agente no Windows para instalação do módulo **puppetlabs-registry** (lembre-se de abrir o terminal do Puppet como *Administrator*):

```
puppet agent -t
```

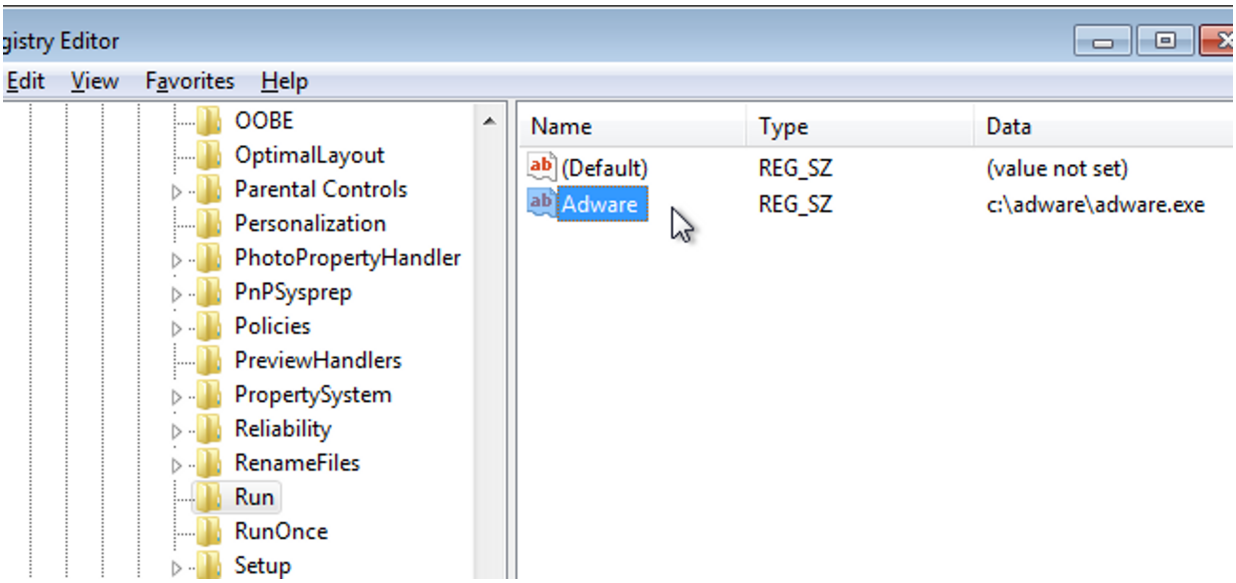
3. Declare uma chave de registro no nosso manifest:

```
node 'win7.domain.com.br' {
  registry::value { 'Adware':
    key   => 'HKLM\Software\Microsoft\Windows\CurrentVersion\Run',
    value => 'Adware',
    data  => 'c:\adware\adware.exe'
  }
}
```

4. Execute o agente no Windows para que a chave no registro seja criada (lembre-se de abrir o terminal do Puppet como *Administrator*):

```
puppet agent -t
```

5. A chave foi criada.



28 Outras Fontes de estudo

Abaixo são listados vários links para você acessar e se aprimorar nos estudos.

Slides:

- https://speakerdeck.com/gutocarvalho/meetup-puppet-br-20160217-origens-devops
- https://speakerdeck.com/gutocarvalho/meetup-puppet-br-20160518-intro-puppet
- https://speakerdeck.com/gutocarvalho/meetup-puppet-br-20160217-introducao-ao-puppet-4
- http://www.slideshare.net/PuppetLabs

Documentação para começar a usar o Puppet:

- http://blog.aeciopires.com/configurando-um-ambiente-puppet/
- http://blog.aeciopires.com/configurando-um-ambiente-puppet-2/
- http://gutocarvalho.net/octopress/2015/08/18/instalando-puppet-4-agent-and-master/
- http://gutocarvalho.net/wordpress/2012/05/23/puppet-documentacao-e-aprendizado/
- http://pt.slideshare.net/GutoCarvalho/oficina-puppet-aprenda-a-gerenciar-configuraes
- http://gutocarvalho.net/dokuwiki/doku.php?id=puppet_serverless
- https://docs.puppetlabs.com/puppet_core_types_cheatsheet.pdf

Documentação para desenvolver manifests, classes e módulos no Puppet:

- https://docs.puppetlabs.com/pe/latest/quick_start.html
- https://docs.puppetlabs.com/guides/style_guide.html
- http://gutocarvalho.net/octopress/2013/04/20/puppet-style-guide/
- https://speakerdeck.com/gutocarvalho/meetup-puppet-br-20160518-desenvolvendo-modulos-e-fatos-puppet
- http://gutocarvalho.net/octopress/2013/04/22/puppet-melhores-praticas/
- http://pt.slideshare.net/GutoCarvalho/trabalhando-com-mdulos-no-puppet
- https://speakerdeck.com/gutocarvalho/meetup-puppet-br-20160518-testes-de-codigo-puppet
- https://docs.puppetlabs.com/guides/module_guides/bgtm.html
- https://docs.puppetlabs.com/puppet/latest/modules_fundamentals.html
- https://docs.puppetlabs.com/pe/latest/puppet_modules_manifests.html
- https://docs.puppetlabs.com/puppet/latest/lang_classes.html

Documentação geral sobre Puppet:

- http://docs.puppetlabs.com/
- http://docs.puppetlabs.com/puppet/latest/
- http://www.example42.com/tutorials/PuppetTutorial/
- http://puppet-br.org/documentacao/

Cursos online e grátis sobre Puppet:

- https://learn.puppetlabs.com/category/self-paced-training
- https://networkfaculty.com/es/courses/coursedetail/41-curso-introduccion-a-puppet
- https://networkfaculty.com/es/courses/coursedetail/42-curso-introduccion-a-la-gestion-de-servicios-con-puppet

Puppet no Windows:

- http://docs.puppetlabs.com/windows/

Livros sobre Puppet:

- https://novatec.com.br/livros/puppet/
- http://www.puppetcookbook.com/
- http://shop.oreilly.com/product/0636920034131.do
- http://shop.oreilly.com/product/0636920038528.do
- https://www.packtpub.com/networking-and-servers/puppet-4-essentials-second-edition
- https://www.packtpub.com/networking-and-servers/mastering-puppet-second-edition
- https://www.packtpub.com/networking-and-servers/extending-puppet-second-edition
- https://www.packtpub.com/networking-and-servers/learning-puppet-windows-server
- https://www.packtpub.com/networking-and-servers/puppet-containerization
- https://www.packtpub.com/all/?search=puppet
- http://search.oreilly.com/?q=puppet&x=0&y=0
- https://puppetlabs.com/resources/books

Puppetboard:

- https://github.com/puppet-community/puppetboard
- http://blog.aeciopires.com/instalando-configurando-puppetdb-puppetexplorer-puppetboard/
- http://domarques.com.br/blog/2014/08/puppet-e-puppetboard-para-automatizacao-de-servidores
- http://www.credativ.de/blog/howto-puppetdb-installation
- https://ask.puppetlabs.com/question/3557/how-to-refresh-puppet-master-cached-catalog/
- https://github.com/puppet-br/pcp

Foreman:

- http://theforeman.org/
- http://theforeman.org/manuals/1.9/quickstart_guide.html#QuickstartGuide
- http://theforeman.org/manuals/1.9/index.html
- http://theforeman.org/media.html

Ferramentas sugeridas para desenvolver no Puppet:

IntelliJ:

- https://www.jetbrains.com/idea

Kate:

- http://kate-editor.org

Vim:

- http://www.openvim.com/tutorial.html
- http://i.imgur.com/1tiqn.png
- https://github.com/rodjek/vim-puppet

Git e Github:

- https://github.com/
- http://rogerdudler.github.io/git-guide/index.pt_BR.html
- http://readwrite.com/2013/09/30/understanding-github-a-journey-for-beginners-part-1
- http://readwrite.com/2013/10/02/github-for-beginners-part-2
- http://www.vogella.com/articles/Git/article.html
- https://git-scm.com/doc
- https://git-scm.com/book/en/v2
- http://stackoverflow.com/questions/6157730/why-cant-i-push-to-this-bare-repository
- https://git-scm.com/book/pt-br/v1/Git-Essencial-Tagging

Vagrant:

- http://blog.rivendel.com.br/2015/09/18/devops-conhecendo-vagrant/
- http://blog.rivendel.com.br/2014/08/20/vagrant-as-vantagens-da-virtualizacao-em-ambiente-local/
- https://speakerdeck.com/gutocarvalho/meetup-puppet-br-20160518-integracao-entre-puppet-e-vagrant
- http://search.oreilly.com/?q=vagrant&x=0&y=0

Para tirar dúvidas com usuários mais experientes, recomendamos fazer parte da comunidade Puppet-BR http://puppet-br.org/comunidade/

29 Histórico de mudanças

- Versão 2.4.5 lançada dia 13/06/2018
- Versão 2.4.4 lançada dia 27/03/2017
- Versão 2.4.3 lançada dia 09/01/2017
- Versão 2.4.2 lançada dia 05/01/2017
- Versão 2.4.1 lançada dia 04/01/2017
- Versão 2.4.0 lançada dia 02/01/2017
- Versão 2.3.0 lançada dia 30/12/2016
- Versão 2.2.0 lançada dia 29/12/2016
- Versão 2.1.0 lançada dia 11/12/2016
- Versão 2.0.2 lançada dia 18/08/2016
- Versão 2.0.1 lançada dia 25/07/2016
- Versão 2.0.0 lançada dia 23/06/2016
- Versão 1.0.0 lançada dia 20/03/2014

29.1 Colaboradores

Aécio Pires:

aeciopires @ gmail.com http://aeciopires.com

Guto Carvalho

gutocarvalho @ gmail.com http://gutocarvalho.net

Miguel Di Ciurcio Filho:

miguel @ instruct.com.br http://instruct.com.br