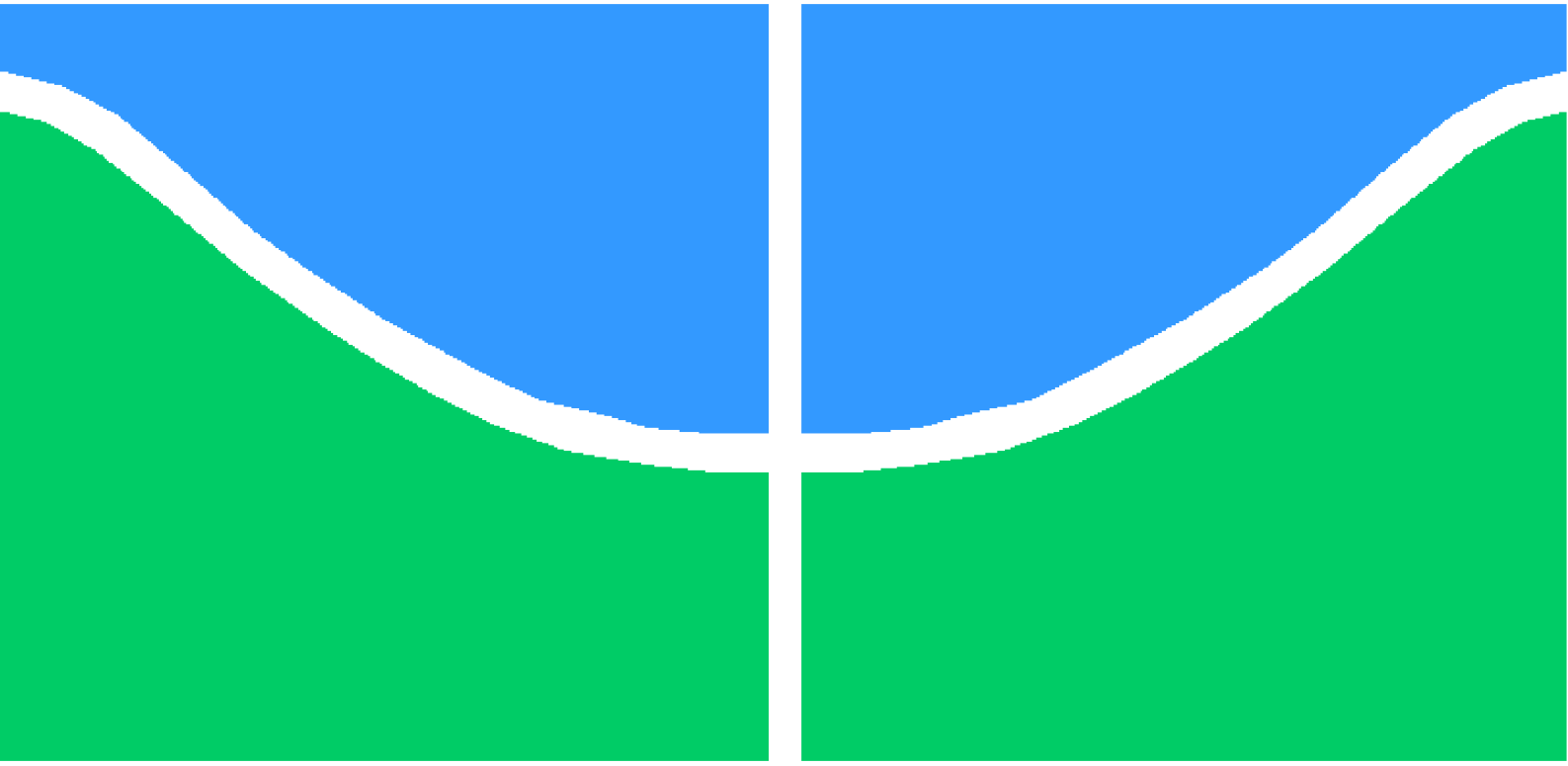

Universidade de Brasília - UnB
Faculdade UnB Gama - FGA
Engenharia de Software

Implantação automatizada de aplicações web em ambientes Debian GNU/Linux

Autor: Thiago de Souza Fonseca Ribeiro
Orientador: Prof. Dr. Paulo Roberto Miranda Meirelles

Brasília, DF
2016

Thiago de Souza Fonseca Ribeiro

Implantação automatizada de aplicações web em ambientes Debian GNU/Linux

Monografia submetida ao curso de graduação em (Engenharia de Software) da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em (Engenharia de Software).

Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Orientador: Prof. Dr. Paulo Roberto Miranda Meirelles

Coorientador: Dr. Antonio Soares de Azevedo Terceiro

Brasília, DF

2016

Thiago de Souza Fonseca Ribeiro
Implantação automatizada de aplicações web em ambientes Debian GNU/Linux/ Thiago de Souza Fonseca Ribeiro. – Brasília, DF, 2016-
71 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Dr. Paulo Roberto Miranda Meirelles

Trabalho de Conclusão de Curso – Universidade de Brasília - UnB
Faculdade UnB Gama - FGA , 2016.

1. Implantação de Software. 2. Debian. I. Prof. Dr. Paulo Roberto Miranda Meirelles. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. Implantação automatizada de aplicações web em ambientes Debian GNU/Linux

CDU 02:141:005.6

Thiago de Souza Fonseca Ribeiro

Implantação automatizada de aplicações web em ambientes Debian GNU/Linux

Monografia submetida ao curso de graduação em (Engenharia de Software) da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em (Engenharia de Software).

Trabalho aprovado. Brasília, DF, 2 de Agosto de 2016:

Prof. Dr. Paulo Roberto Miranda Meirelles
Orientador

Dr. Fabio Macedo Mendes
Convidado 1

Msc. Renato Coral Sampaio
Convidado 2

Brasília, DF
2016

Agradecimentos

Agradeço primeiramente a Deus, pela minha vida e por alimentar a minha fé, por me dar forças para continuar e sabedoria para tomar as decisões certas.

Agradeço aos meus professores da Universidade de brasília, da Faculdade UnB Gama, por todo conhecimento compartilhado e pela importância de cada um na minha graduação, em especial, gostaria de fazer um agradecimento a alguns professores, pela sua importância, não só na minha formação profissional, como também na formação do meu caráter e suas contribuições na minha vida.

Agradeço ao meu orientador, Prof. Paulo Roberto Miranda Meirelles, por toda confiança que foi creditada em mim nesses últimos anos, pela sua paciência e dedicação na minha formação de engenharia de software e por sua dedicação ao curso de engenharia de software , por todos os ensinamentos, que são muito importantes para minha vida profissional e pessoal, obrigado pelo apoio e amizade, pelas portas que foram abertas e pela orientação neste trabalho.

Agradeço ao meu professor, Prof. Hilmer Rodrigues Neri, também pela confiança que foi creditada em mim, por ser o meu primeiro professor a acreditar no meu potencial, por todos os ensinamentos e sua dedicação ao curso de engenharia de software, agradeço pois muito do que sei tem a sua contribuição, tudo que conquistei até hoje foi graças a primeira porta que você abriu, agradeço pelos ensinamentos e por toda a dedicação e pelo apoio durante a minha graduação, obrigado pela amizade e por sempre confiar em mim aonde estiver.

Agradeço ao meu professor, Prof. Edson Alves da Costa Junior, por todos os ensinamentos, não só em disciplinas como na vida, tive a sorte de ter aula com um grande mestre, no qual nunca esquecerei cada ensinamento e cada momento de aprendizado. Obrigado pelo apoio e por ser sempre disponível a ajudar, por ser um excelente mestre, pela sua dedicação no curso de engenharia de software e por sua companhia nos cafés do obelisco.

Agradeço ao Antônio Terceiro, por todos os ensinamentos e toda confiança depositada em mim, pelas portas que foram abertas e por compartilhar seu conhecimento comigo, por toda paciência e dedicação que teve ao me ensinar muito do que sei, por ser um grande engenheiro de software, e um exemplo de profissional que quero seguir, que um dia eu pretendo ser ao menos um pouco do que você é.

Agradeço à todos os membros do Laboratório Avançado de Produção, Pequisa e Inovação em Software - LAPPIS pelas experiências compartilhadas, o crescimento con-

junto e o trabalho em equipe. Sei que dificilmente encontrarei uma equipe tão brilhante como essa, por isso agradeço pelos momentos nesses três anos de LAPPIS, foram momentos de muita alegria, desafios e aprendizado, espero um dia poder trabalhar com vocês novamente.

Agradeço a minha família, principalmente aos minha mãe Rosilene e meu pai Haroldo, por a toda a sua dedicação e amor, pela confiança e pelo apoio incondicional em todos os momentos da minha vida, por serem minha maior motivação e meu espelho como ser humano, por toda admiração e todo investimento. A minha conquista também é a conquista de vocês, e agradeço por serem excelentes pais e grandes profissionais em suas áreas de atuação, isso me faz querer ser o melhor, assim como vocês.

Agradeço imensamente meus amigos e engenheiros de software Athos Ribeiro e Lucas Kanashiro, por me ajudarem na revisão deste trabalho. A colaboração de vocês foi importantíssima, além de me darem a oportunidade de aprender com vocês, mais uma vez.

Agradeço também aos meus familiares e amigos que me apoiaram e me ajudaram a chegar até aqui, principalmente tios, amigos, primos, avós, e minha namorada Camila, por sempre acreditarem em mim e por sempre me motivarem a não desistir nunca.

"Deus não coloca um peso nos ombros de um homem se souber que ele não pode carregá-lo. Muhammad Ali

Resumo

A implantação de aplicações web de grande escala apresenta vários desafios, com isso, a implantação automatizada de software vem se tornando uma necessidade, principalmente pelos desafios de instalação e configuração das ferramentas web mais modernas, podendo facilitar a instalação e configuração de aplicações, tanto para desenvolvedores como para usuários. Este trabalho trata da implantação de software com seus procedimentos e ferramentas, com o foco na automação da instalação de aplicações web em ambientes Debian GNU/Linux. Para isso, foi feito uma colaboração da construção da ferramenta Shak, que propõe a instalação de aplicações web com apenas uma instrução, o que possibilita a instalação e configuração de ferramentas web com pacotes disponíveis nos servidores oficiais do Debian, utilizando protocolos seguros como HTTPS e implantação de aplicações web utilizando hospedagem virtual.

Palavras-chaves: Implantação de software, Debian.

Abstract

The deployment of large-scale web applications present several challenges, with it, the automated deployment of software is becoming a necessity, mainly for installation challenges and configuration of more modern web tools and facilitate the installation and configuration of applications, for boths developers and users. This work is about software deployment with procedures and tools, with focus on automation of web applications instalation on Debian GNU/Linux environments, for this, Shak tool had been improved, which offers a Web application installation with only one instruction, which allows the installation and configuration of web tools with available packages in official debian servers, using secure protocols, like HTTPS and web application deployment using virtual hosts.

Key-words: Software deployment, Debian.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

ACME	Automatic Certificate Management Environment
CSS	Cascading Style Sheets
DevOps	Amálgama de Desenvolvimento e Operações
DNS	Amálgama de Desenvolvimento e Operações
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
IMAP	Internet Message Access Protocol
IP	Internet Protocol
IRC	Internet Relay Chat
OMG	Object Management Group
POP3	Post Office Protocol
RFC	Request for Comments
SHAK	Self Hosting Applications Kit
SMTP	Simple Mail Transfer Protocol
SSL	Secure Socket Layer
SQL	Linguagem de consulta estruturada
TCP	Transmission Control Protocol
TLS	Transport Layer Security

Sumário

1 Introdução

A Internet emergiu no contexto da Guerra Fria na década de 60, em um projeto do exército norte-americano. Os motivos de sua criação eram: criar um sistema de informação e comunicação em rede, que sobrevivesse a um ataque nuclear e dinamizar a troca de informações entre os centros de produção científica. Os militares pensaram que um único centro de computação centralizando toda informação era mais vulnerável a um ataque nuclear do que vários pontos conectados em rede, pois assim, a informação estaria espalhada por inúmeros centros computacionais pelo país (GILES, 2010).

Desde a sua origem a Internet foi projetada para ser uma rede descentralizada e flexível (CASTELLS, 2003). A maioria dos protocolos utilizados na rede não dependem de pontos centrais para funcionar. Considerando que no passado a maioria dos softwares eram como uma ilha isolada, hoje os softwares estão mais complexos pois se caracterizam por um conjunto de componentes, que são disponibilizados através da web ou conectado através da Internet, como por exemplo: as redes sociais e mercados de compras online (SPINELLIS, 2012a).

Essa grande sofisticação implica numa alta complexidade de desenvolvimento e manutenção, que faz com que a implantação de software e a manutenção de serviços exija conhecimento técnico especializado. Para as organizações isto implica num maior custo para manter sistemas e aplicações. Para os indivíduos isto quase sempre impossibilita a manutenção de serviços próprios, e os leva à procura por serviços gratuitos (TERCEIRO, 2016).

Existe atualmente uma tendência de centralização da Internet. Vaz (2006) mostra através de uma série de dados empíricos que a Internet está centralizada. Isso implica que os serviços que são utilizados pelos usuários, na maioria das vezes, são serviços que são providos por grandes empresas. Um exemplo disso é que os usuários que utilizam contas de e-mail preferem usar algum dos grandes provedores de e-mail, como Gmail[1].

Isso também acontece com aplicações responsáveis por armazenar dados dos usuários, com armazenamento de documentos na nuvem. Alguns exemplos são documentos como: fotos, vídeos e arquivos. Eles são geralmente armazenados em ferramentas conhecidas, como Google Drive [2] ou Dropbox [3].

Estes serviços não custam nada ao usuário, mas quando o usuário aceita os termos de uso, é possível que algumas empresas utilizem das informações pessoais dos usuários

[1] http://gmail.com/
[2] https://www.google.com/intl/pt-BR/drive/
[3] https://www.dropbox.com/pt_BR/

para usos comerciais. Essa excessiva centralização traz problemas para os usuários que utilizam deste tipo de ferramentas, tais como: riscos a privacidade, subordinação dos usuários aos provedores e pontos centrais de falhas (TERCEIRO, 2016).

Para solucionar este problema, seria necessário a existência de uma alternativa à centralização da Internet, através de uma solução que elimina barreira técnica, e para que usuários interessados possam ter servidores próprios com aplicações web, sem a necessidade de conhecimento técnico especializado.

A dificuldade de prover essa alternativa é que os sistemas voltado para a Internet consistem em muitas partes que são complicadas de gerenciar. Alguns requisitos como: disponibilidade e desempenho são indispensáveis, além de ser necessário gerenciar as aplicações em seus devidos servidores, com o gerenciamento de banco de dados e rotinas de recuperação de dados, além de administrar uma infinidade de bibliotecas de terceiros e serviços online (SPINELLIS, 2012b).

Além disso, existe o impeditivo dos usuários terem que hospedar seus próprios serviços. Porém, ultimamente este fator vem perdendo relevância em função de dois fenômenos. O primeiro deles é o barateamento do acesso a servidores virtuais privados, decorrente dos avanços da computação em nuvem. Por outro lado, a disponibilidade de servidores físicos de dimensão reduzida e baixo consumo de energia, em conjunto com o barateamento de conexões de banda larga à Internet (TERCEIRO, 2016).

Esses impeditivos atingem de fato o usuário final, e a decisão acaba sendo de utilizar os serviços gratuitos das grandes empresas. Porém, cada vez mais caminhamos para uma mídia que atenda às necessidades do indivíduo. O usuário será o responsável por aquilo que deseja consumir na rede, essa mídia denomina-se You-Media *(U-Media)*. São inerentes colaboração à U-Media: contribuição e comunidades, participação e customização, além da descentralização dos serviços (TERRA, 2006).

Este trabalho relata a evolução da ferramenta Shak, que é uma ferramenta para implantação automatizada de aplicações web em sistema Debian GNU/Linux, que tem como objetivo fornecer uma alternativa à centralização da Internet. Essa alternativa é dada através de uma solução que elimina barreira técnica, para que usuários interessados possam ter servidores próprios com aplicações web, sem a necessidade de conhecimento técnico especializado.

O problema que envolve este trabalho é:

Como implantar aplicações web em sistema Debian GNU/Linux de forma automatizada e segura?

1.1 Objetivo

O objetivo deste trabalho consiste na contribuição em uma ferramenta que possa automatizar instalação e configuração de aplicações web em sistemas Debian GNU/Linux, a partir de pacotes que sejam distribuídos oficialmente pelo Debian. Mostrando também os aspectos mais importantes que devem ser tratados durante todo o processo de configuração e instalação de um software, facilitando assim, que tanto usuários e desenvolvedores possam implantar aplicações com apenas uma instrução.

As contribuições deste trabalho são:

Contribuições Tecnológicas

1. **CT1** Implantação de sistemas Debian GNU/LINUX,
   a) Implantação de aplicações web com certificados digitais autoassinados.
   b) Implantação de aplicações web com suporte a múltiplas instâncias no mesmo servidor. hospedagem virtual.
   c) Implantação de automatizada de aplicações web utilizando pacotes distribuídos oficialmente pelo Debian.

Contribuições Científicas

1. **CC1** Gestão de configuração de software,
   a) Estudo teórico sobre implantação automatizada de software.

1.2 Organização do trabalho

O trabalho está organizado na seguinte forma: O Capítulo 2 traz o referencial teórico necessário para apoio o desenvolvimento da solução, como a gerência de configuração de software, o processo de implantação de software e métodos e técnicas para implantação automatizada de software. O Capítulo 3 fala sobre a definição e preparação dos estudos, que envolve os trabalhos relacionados, envolve também como será construída a solução e também como a solução será validada. Já no Capítulo 4 contém os resultados alcançados, e por fim as considerações finais no Capítulo 5.

2 Gestão de configuração de software

De acordo com IEEE (2012), gestão de configuração de software *(GCS)* é uma área da engenharia de software cujo objetivo é:

- Identificar e documentar as características funcionais de um produto.
- Controlar quaisquer alterações a estas características.
- Registrar e relatar cada mudança do seu estágio de implantação
- Apoiar a auditoria dos produtos, resultados, serviços ou componentes para verificar a conformidade com requisitos

Gestão de Configuração de Software é uma das capacidades fundamentais que devem estar em qualquer projeto de desenvolvimento de software. Ela é tradicionalmente dividida em três atividades, e são elas: configuração, identificação e controle. Isto é encarado como uma ferramenta de gestão que pode ajudar na orientação de cada projeto, pois ajuda a manter a integridade do produto e manter o qualidade do projeto sob controle (BENDIX; KOJO; MAGNUSSON, 2011).

Ela também é essencial para a engenharia de software, pois estabelece e protege a integridade de um produto ao longo da sua vida útil, percorrendo através dos processos de desenvolvimento, teste e entrega do produto, bem como durante a sua instalação, operação, manutenção e eventual evolução (IEEE, 2012).

Gestão de configuração de software também fornece a infraestrutura que é a base para qualquer tipo de projeto de software. Isto facilita a coordenação e comunicação entre os vários participantes numa equipe de desenvolvimento de software. Dentro da gestão de configuração de software existem várias áreas de atuação (BENDIX; KOJO; MAGNUSSON, 2011). Este trabalho aborda especificamente a implantação de software.

Este capítulo possui a Seção 2.1, que fala sobre implantação de software, além da Seção 2.2 que fala sobre *DevOps*, uma nova abordagem que traz um conceito de implantação automatizada de software junto aos métodos ágeis de desenvolvimento de software.

Depois, a Seção 2.3 que fala sobre ferramentas e técnicas que buscam facilitar e resolver os problemas da implantação automatizada de software.

E por fim, a Seção 2.4, que fala sobre quais são os procedimentos que devem ser levados em consideração durante uma implantação de aplicações voltada para a web.

2.1 Implantação de Software

Software só oferece valor para os clientes quando ele está implantado e em produção, e assim, possa proporcionar as funcionalidades necessárias ao usuário final. Por isso, a importância da fase de implantação de software, pois é nessa fase em que a equipe de desenvolvimento disponibilizará o software ou uma atualização com novas funcionalidades ao usuário.

Sendo assim, a implantação de software é um conjunto de atividades cruciais para todos os fornecedores de software. Isto começa desde um pedido de um novo requisito de software, até todas as medidas necessárias para que essa nova versão fique disponível para o cliente (MANTYLA; VANHANEN, 2011). A implantação de software é composta por várias atividades que são essenciais para disponibilizar um produto a alguém, como por exemplo: instalação de dependências, arquivos de configuração e instalação da própria aplicação.

Uma grande aliada das equipes de desenvolvimento é a implantação automatizada de software, que se refere à prática de implantar o software para os usuários finais automaticamente, evitando qualquer tipo de execução de esforço manual. Por isso, a prática de implantação automatizada facilita na rápida entrega de software, tanto para implantações de aplicações em servidores na nuvem, como implantação de software para usuários em seus computadores (RAHMAN et al., 2015).

As aplicações de software não são mais sistemas isolados, são cada vez mais o resultado da integração de coleções de componentes. Isso pode tornar a implantação de software um processo complicado, e nem sempre existir a garantia de que cada componente será implantado corretamente. Portanto, a equipe de desenvolvimento deve encontrar uma maneira de lidar com uma maior incerteza no ambiente nos quais seus sistemas vão operar, garantindo que a implantação do software seja feita da forma correta, evitando qualquer acontecimento de erros e imprevistos (CARZANIGA et al., 1998).

2.1.1 Processo de Implantação de Software

A implantação de um software é o processo que vai desde a aquisição desse software até a sua execução (LEITE, 2014). O processo de implantação consiste em diversas atividades que devem ser executadas, desde o planejamento da implantação até a disponibilidade para uso.

A *OMG* é uma organização internacional sem fins lucrativos, que aprova padrões abertos para tecnologias. Eles definem uma especificação de implantação e configuração de aplicações distribuídas baseadas em componentes (OMG, 2006). A *OMG* também diz que o processo de implantação é um processo que se inicia desde a aquisição de um componente, até o momento em que esse componente está em plena execução pronto pra

uso.

Segundo OMG (2006) os principais termos definidos dentro de uma implantação de software são:

- **Implantador:** Pessoa ou equipe responsável pelo processo de implantação de um sistema.
- **Ambiente alvo:** São o servidor ou conjunto de servidores em que os componentes são implantados.
- **Nó:** É um recurso computacional onde se implanta um componente, por exemplo uma máquina virtual dentro do servidor que vai hospedar um serviço de banco de dados. Os nós fazem parte do ambiente alvo.
- **Pacote:** Artefato executável que contém o código binário do componente . É por meio de um pacote que um serviço pode ser instalado e executado dentro de um sistema operacional, e que são característicos dependendo da distribuição do sistema operacional, por exemplo: .deb para Debian e .rpm para Redhat, ou pacotes independentes de sistema operacional como por exemplo pacotes jar.

Além disso, é definido as fases que compõem o processo de implantação e de acordo com OMG (2006) as fases são:

- **Planejamento:** O planejamento da implantação é uma fase para identificar os componentes necessários na implantação e como cada um será distribuído entre os nós do ambiente alvo.
- **Preparação:** São os procedimentos necessários para preparar o ambiente alvo para que um determinado componente possa ser executado, isso envolve configuração do sistema operacional, instalação e configuração de dependências necessárias (por exemplo um servidor web como Apache ou Nginx).
- **Instalação:** O implantador transfere o componente para a infraestrutura alvo, é quando instalamos um dado componente em um servidor, um exemplo disso é a instalação de uma aplicação via pacotes (.deb ou .rpm).
- **Configuração:** Edição de arquivos de configuração para alterar determinado comportamento de uma aplicação, o implantador aplica configurações específicas à aplicação a partir de arquivos de configuração.
- **Inicialização:** É quando a aplicação é iniciada e entra em execução, pronta para receber chamadas de seus clientes.

Os termos e fases formam uma estrutura básica que um processo de implantação de software deve conter. Cada fase é necessária para que se possa ter uma implantação consistente, ou seja, as atividades dentro de uma implantação de software estarão organizadas, evitando assim possíveis erros durante a implantação. Essa organização também pode ajudar os implantadores a diagnosticar eventuais problemas.

Dependendo do tamanho da aplicação as fases podem se tornar tarefas complicadas, por isso, existe a necessidade de automatizar o processo de implantação. Isso diminui a possibilidade de erros, comparado a uma tarefa manual, e consequentemente torna o trabalho mais ágil, eliminando a necessidade de manuais de instalação. O objetivo de um processo de implantação automatizado é proporcionar um processo de implantação reprodutível, confiável e fácil de ser executado (HUMBLE; FARLEY, 2010).

Para que a implantação automatizada de software seja possível, é necessário o uso de ferramentas que possam automatizar todo o processo de implantação. Porém, é necessário que o implantador compreenda cada fase do processo para que possa ser feito um bom planejamento e execução da implantação.

2.2 Desenvolvimento e Operações (DevOps)

Como resultado ao longo dos anos, as equipes de desenvolvimento de software são capazes de entregar software a um ritmo muito mais rápido, principalmente com a adoção de métodos ágeis de desenvolvimento de software (VIRMANI, 2015). As equipes desenvolvem de forma acelerada, enquanto a equipe de operações consegue implantar software de forma sequencial. Sendo assim, *DevOps* nasceu a partir dessa necessidade, de implantar software no mesmo ritmo em que as equipes são capazes de entregar novas funcionalidades.

DevOps pode ser visto como um conjunto de práticas e princípios para a entrega de software, com foco na velocidade de entrega e na automação (VIRMANI, 2015). Antes o time de desenvolvimento terminavam suas funcionalidades e seus testes e entregavam a uma equipe de implantação, e a equipe de implantação precisava lidar com os problemas sem a ajuda do time de desenvolvimento. Por isso *DevOps* tenta unir esses dois mundos, fazendo com que as duas equipes compartilhem atividades e responsabilidades (SPINELLIS, 2012b).

A comunidade *DevOps* defende a comunicação entre a equipe de operações e a equipe de desenvolvimento, como um meio de assegurar que o os desenvolvedores entendam os problemas associados com as operações. Um dos principais benefícios disso, é a capacidade de quantificar os problemas dos dois mundos, para que possa levar a melhoria do desenvolvimento do produto (HTTERMANN, 2012).

Em Virmani (2015) são abordados as práticas *DevOps*, que podem ser aplicadas diversas vezes dentro de um ciclo de desenvolvimento de software. As praticas *DevOps* podem ser resumidas em:

- **Planejamento Contínuo:** O planejamento da equipe de operações deve ser sempre contínuo e evoluído junto ao planejamento da equipe de desenvolvimento, com tarefas sendo priorizadas o tempo todo e sempre alinhadas de acordo com as decisões tomadas em conjunto com a equipe de desenvolvimento.
- **Integração Contínua:** Compartilhar as alterações feitas com toda equipe evitando que as novas modificações fiquem apenas nas mãos do time de desenvolvimento.
- **Implantação Contínua:** O coração do *DevOps*, recomenda-se automatizar todo o processo de implantação, removendo qualquer tipo de etapas manuais com auxílio de uma ferramenta que possa automatizar a instalação, sendo assim a implantação de cada nova versão de software passa a ser feita de forma mais rápida e eficiente.
- **Testes Contínuos:** Automatiza também os testes da sua aplicação.
- **Monitoração Contínua:** Monitorar as novas alterações que foram implantadas a fim de aumentar a capacidade de reagir a quaisquer surpresas em tempo hábil.

Os benefícios do uso do *DevOps* são vários, como por exemplo: economia de tempo, já que implantação de software passa a ser um processo natural e automatizado, gerando economia de custos, e aumentando a eficiência do desenvolvimento de software como um todo (VIRMANI, 2015).

Nesse trabalho, foi utilizada a prática *DevOps* de implantação contínua, automatizando todo o processo de implantação de aplicações web.

Para entender melhor a prática de implantação automatizada, que é o coração do *DevOps*, na Subseção 2.2.1 será abordado a infraestrutura como código, que é uma prática bastante difundida nas comunidades *DevOps*.

2.2.1 Infraestrutura como código

Infraestrutura como código é a prática de especificar configurações de sistema de computação através de código, com o intuito de automatizar a implantação de um software e o seu gerenciamento de configurações. Por exemplo, um sistema com diferentes configurações de hardware e diferentes requisitos de software podem ser especificados e implantados automaticamente sem intervenção humana (SHARMA; FRAGKOULIS; SPINELLIS, 2016).

Tal procedimento pode ser ainda personalizado, conforme as necessidades da implantação. Sendo assim, a implantação automatizada não só é mais rápida do que o processo manual, mas também é mais confiável e reproduzível.

Para aplicar a infraestrutura como código em um projeto, é necessário migrar a infraestrutura do projeto em código, essencialmente como um código que o desenvolvedor lida no seu trabalho diário (SPINELLIS, 2012a).

Dado que um *script* de configuração realmente contém código, é mais natural para o desenvolvedor tratá-lo como tal, utilizando suas habilidades de codificação, além de poder utilizar ferramentas que auxiliem a prática do *DevOps* e de colaboração, entre a equipe de desenvolvimento e a equipe de infraestrutura (SPINELLIS, 2012a).

Com a infraestrutura sendo tratada como código, é possível utilizar um sistema de controle de versão que permita outras pessoas colaborem e trabalhem em equipe. Com a ajuda de um sistema de controle de versão, por exemplo, é possível ter uma documentação completa do estado do seu sistema, através do histórico de alterações, para que seja possível trazer o código da infraestrutura para um estado desejado a partir de qualquer outro estado.

Para realizar essa prática, existem ferramentas como Chef [1] e Puppet [2], que fornecem aos desenvolvedores várias abstrações para expressar etapas de automação, de forma independente. Esses recursos podem ser executados várias vezes, obtendo-se sempre o mesmo resultado, trazendo uma capacidade de atingir um determinado estado desejado. Isso permite múltiplas iterações, com implantações frequentes de infraestruturas complexas (HUMMER et al., 2013).

2.3 Métodos e ferramentas para implantação automatizada de software

Existem várias soluções técnicas para melhorar a implantação de software. No entanto, apesar da sua importância para as empresas de software, as atividades de implantação de software têm recebido pouca atenção (MANTYLA; VANHANEN, 2011). Recentemente um número de novas tecnologias começaram a emergir para resolver o problema de implantação. As características típicas oferecidas por estas tecnologias incluem sistemas para automatizar a implantação a partir de configurações, pacotes, gerenciamento de rede e instalação de recursos, com propósito de entrega das atualizações de forma automática (CARZANIGA et al., 1998).

Para a implantação automatizada de software, é possível utilizar linguagens de

[1] https://www.chef.io/chef/
[2] https://puppet.com/

script de propósito geral como: Python, Ruby e Shellscript. Também existem ferramentas gerais voltados para o processo de implantação, como sistemas de *middleware* especializados em determinados tipos de artefatos implantáveis (LEITE, 2014).

Um processo de implantação automatizado depende bastante da integração de diferentes papéis numa organização. Como foi dito na Seção 2.2, é importante a integração entre desenvolvedores e operadores, uma vez que o desenvolvimento desses *scripts* ou utilização das ferramentas precisam de participação de ambos os perfis.

Tais ferramentas, permitem-nos controlar e automatizar a configuração de todos os elementos que compõem um sistema. São eles: software a serem instalados, usuários do sistema, serviços em execução, arquivos de configuração, tarefas agendadas, configuração de rede, armazenamento de arquivos, monitoramento e segurança. A sua função principal é automatizar a instalação e configuração de um sistema.

Com essas ferramentas, é possível escrever algumas regras que expressam como um software deve ser configurado, e assim, a ferramenta configurará o sistema conforme as especificações. Essas ferramentas funcionam de forma declarativa, especificando a instalação de um software através de regras (SPINELLIS, 2012b).

Por exemplo, é possível especificar regras para a configuração do sistema a partir de uma infraestrutura básica. No Chef, essa estrutura é conhecida como livro de receitas, já em Puppet, essa estrutura é conhecida como manifesto. Esses recursos são basicamente arquivos, no qual o implantador define os passos que serão executados.

Cada passo da configuração desejada pode depender de outros passos, ou seja, para instalar uma aplicação pode ser necessário instalar previamente um compilador ou interpretador da linguagem. Um outro exemplo é que para executar uma aplicação web é necessário a instalação prévia de um servidor web. Para lidar com as dependências essas ferramentas descrevem o estado em que a aplicação deve estar, ou seja, todos os pacotes, arquivos de configuração diretórios e usuários, sendo possível definir a dependência entre as tarefas a serem executadas.

Com essas ferramentas a implantação de um software dentro de uma infraestrutura não precisa de esforço manual. Com elas, é possível automatizar toda a implantação de um software, desde a preparação do ambiente até a inicialização da aplicação. Sendo assim, reduzir o tempo gasto na implantação sem qualquer esforço adicional.

Além das ferramentas de automação, é importante também entender o empacotamento de programas. Tanto Chef como Puppet utilizam a instalação de pacotes como recurso para poder instalar softwares, ou seja, é possível buscar os softwares desejados a partir dos pacotes disponíveis numa distribuição GNU/Linux.

As distribuições GNU/Linux que optam por disponibilizar pacotes, mantém uma infraestrutura de servidores como fonte de distribuição de programas (ARAÚJO, 2011).

Tais servidores são chamados de repositórios, e esses pacotes são gerenciados com softwares conhecidos como sistemas gerenciadores de pacotes, que possuem a responsabilidade de buscar os softwares a serem instalados que estão disponíveis nos respectivos repositórios.

Existem pacotes que dependem do sistema operacional, como por exemplo, pacotes .deb para a distribuição Debian e pacotes .rpm para a distribuição Fedora, e pacotes independentes de sistema operacional como por exemplo, pacotes jar da linguagem java. Neste trabalho foi utilizado apenas de pacotes Debian.

2.3.1 Pacotes Debian

O formato utilizado pelo Debian GNU/Linux e seus derivados é o formato de pacotes binários conhecido como .deb. Um pacote .deb é composto de arquivos executáveis, bibliotecas e documentação associada a um programa ou a um conjunto de programas, além de todos os dados e procedimentos necessários para instalar, configurar e remover aplicativos de um sistema (ARAÚJO, 2011). A estrutura dos pacotes .deb e seus requisitos para que sejam distribuídos oficialmente pelo Debian, estão especificados no Manual de Políticas do Debian (DEBIAN, 2016).

Ao instalar um pacote .deb, são feitos todos os procedimentos necessários para a instalação de uma aplicação, porém, existem ainda, casos em que o usuário precise fazer configurações adicionais. Um exemplo é quando o usuário precisa configurar um banco de dados ou fazer uma configuração específica de uma aplicação a partir de arquivos de configuração.

Essas tarefas não são de responsabilidade dos pacotes, e sim do usuário que deseja utilizar esse software, já que o pacote não poderia prever qual é a estrutura e a configuração que o usuário pretende utilizar. Logo, a execução desses passos são feitas de acordo com a preferência do usuário, podendo ser feita manualmente ou a partir de uma implantação automatizada que utilize alguma ferramenta para isso. Um exemplo desse tipo de atividade, que não é de responsabilidade de um pacote, é a configuração de hospedagem virtual, que será visto na Subseção 2.3.2.

2.3.2 Múltiplas Instâncias de Aplicações Web com Hospedagem Virtual

Na implantação de software, é necessário escolher um ambiente alvo, onde a aplicação será instalada. Esse ambiente deve possuir um nó, que é um recurso computacional onde se implanta um componente.

Porém, pode existir a necessidade de implantar várias aplicações no mesmo servidor, e no caso de aplicações web, isso pode ser um problema. O problema é que as aplicações estarão em um único servidor, assim também em um único endereço de *IP*, e

isso pode gerar conflitos de nomes, visto que nesse caso, todos os serviços precisariam ter o mesmo nome, e isso seria inviável para várias aplicações no mesmo servidor.

Uma forma de solucionar esse problema é o uso de hospedagem virtual, que é um termo que se refere à prática de executar mais de uma aplicação web numa única máquina (APACHE, 2016). Com a hospedagem virtual, é possível hospedar mais de um nome de domínio no mesmo computador, o que possibilita ter vários nomes de domínio em execução, num único endereço de *IP*. Isso possibilita que, num mesmo servidor, tenha uma aplicação com o endereço www.blog.com e uma outra aplicação com o endereço www.nuvem.com.

O fato de que eles estão em execução no mesmo servidor físico não é aparente para o usuário final. Essa solução é importante para economizar recursos, já que não será necessário possuir um servidor dedicado apenas a uma aplicação.

Alguns softwares possuem suporte para a implementação de hospedagem virtual, como por exemplo o Apache e o Nginx. Além disso, existem vários outros recursos que essas ferramentas possuem, como por exemplo o recurso de proxy. Porém, para este trabalho será necessário apenas o uso de hospedagem virtual.

2.4 Segurança na Implantação de Aplicações Web

Um pacote cumpre bem suas responsabilidades dentro de uma implantação de software, porém, outras atividades também são importantes e fogem da responsabilidade de um pacote. Outro fator, é que um pacote também não pode prever quais são os procedimentos de segurança que devem ser tomados numa implantação, como por exemplo, a criação de certificados digitais.

Esses procedimentos estão relacionados a segurança na implantação de aplicações web utilizando pacotes Debian, no qual trata o uso de protocolos que utilizam criptografia para a segurança de dados.

Existem algumas categorias predominantes e prejudiciais de ataques na Internet. São elas: ataques via malware, recusa de serviços, analisador de pacotes, disfarce na fonte e modificação e exclusão de mensagem. Para proteger desses ataques é importante o uso de comunicação segura, protegendo as aplicações (KUROSE et al., 2010)

Para isso, Kurose et al. (2010) ainda identifica as propriedades desejáveis da comunicação segura, são elas:

- **Confidencialidade:** Diz respeito a somente o remetente e o destinatário pretendido poderem entender o conteúdo da mensagem transmitida.

- **Autenticação do ponto final:** Diz respeito a tanto o remetente como o destinatário precisar confirmar a identidade da outra parte envolvida na comunicação.
- **Integridade da Mensagem:** Diz respeito a assegurar que o conteúdo da mensagem não seja modificado.
- **Segurança Operacional:** Diz respeito a segurança da rede, no qual os atacantes podem tentar adquirir os segredos da rede e lançar ataques DoS, por isso é necessário uma boa configuração de firewall e sistemas de detecção de invasão.

No contexto de implantação de aplicações web, foi tomado como objetivo a segurança dos dados transmitidos. Com isso, aplicando criptografia nos protocolos da camada de aplicação, que serão utilizados neste trabalho, sendo eles os protocolos Hypertext Transfer Protocol *(HTTP)* e Simple Mail Transfer Protocol *(SMTP)*, assim adiciona-se uma camada de segurança nas aplicações que serão implantadas.

2.4.1 Protocolos da Camada de Aplicação e Camada de Segurança

As aplicações web utilizam protocolos da camada de aplicação (KUROSE et al., 2010). A camada de aplicação é onde residem as aplicações de rede e seus protocolos, incluindo os protocolos *HTTP* e *SMTP*, que são protocolos importantes neste trabalho, já que aplicações web utilizam o *HTTP* para transferência de arquivos e o *SMTP* para transferência de mensagens de correio eletrônico.

Os protocolos *HTTP* e *SMTP* estão especificados em *RFCs*, a especificação do protocolo *HTTP* está disponível na *RFC* 2616 [3], já o protocolo *SMTP* está especificado na *RFC* 5321 [4].

A aplicações que serão utilizadas neste trabalho, são aplicações web empacotadas no Debian. Aplicações web costumam ser aplicações cliente-servidor, que permitem aos usuários obterem documentos de servidores web a partir de requisições. Esses documentos são padronizados, como por exemplo o *HTML*, para que os navegadores possam interpretar e passar a informação ao usuário (KUROSE et al., 2010).

O *HTTP* é implementado em dois programas, um cliente e um servidor, conversando um com outro a partir de troca de mensagens. O papel do *HTTP* é definir a estrutura dessas mensagens e o modo como o servidor e o cliente as trocam. O principal conteúdo das mensagens trocadas entre clientes e servidores são os objetos. Um objeto é basicamente um arquivo, podendo ser um vídeo, uma imagem ou um arquivo *HTML* (KUROSE et al., 2010).

[3] https://www.ietf.org/rfc/rfc2616.txt
[4] https://www.ietf.org/rfc/rfc5321.txt

Já o *SMTP*, é o protocolo responsável por transferir mensagens de servidores de correio remetentes para servidores de correio destinatários, também transferindo arquivos, de um servidor de correio para o outro. Uma diferença entre o *HTTP* e o *SMTP* é que o protocolo *HTTP* é um protocolo de recuperação de informações, enquanto o protocolo *SMTP* é um protocolo de envio de informações (KUROSE et al., 2010).

Isso implica que, para um usuário recuperar as informações de e-mail, é necessário o uso de outros componentes. Já que isso não é possível via *SMTP*, para resolver isto existem os protocolos de acesso ao correio, entre eles o Post Office Protocol *(POP3)* e Internet Message Access Protocol *(IMAP)*.

Para adicionar uma camada de segurança, com sigilo e integridade de dados, é necessário o uso de outro protocolo, o protocolo Secure Socket Layer *(SSL)*, para complementar os protocolos *HTTP* e *SMTP*. O protocolo *SSL* é utilizado por basicamente todos os sites conhecidos, como Google, Amazon e eBay. Seu uso é para oferecer segurança em transações, pois ele cria um canal criptografado entre um servidor e um cliente, garantido que todos os dados transmitidos sejam sigilosos e seguros (KUROSE et al., 2010).

Para poder utilizar o *SSL*, é necessário que o servidor possua um certificado digital, que possa responder algumas questões sobre a identidade do seu servidor. Com as devidas configurações, tanto *HTTP* como *SMTP* estarão sobre uma camada adicional de segurança, que utiliza o protocolo *SSL*. Essa camada adicional permite que os dados sejam transmitidos por meio de uma conexão criptografada e que se verifique a autenticidade do servidor e do cliente por meio de certificados digitais.

2.4.2 Assinaturas e Certificados Digitais

As assinaturas e certificados digitais servem para agregar confiança e segurança nas trocas de dados pela Internet, principalmente quando se trata de troca de informações sigilosas que precisam de uma confiabilidade maior. O que viabiliza a assinatura digital é o emprego de criptografia assimétrica ou criptografia de chaves públicas.

Logo, a assinatura digital deve ser verificável e não falsificável. Uma aplicação importante de assinaturas digitais é a certificação de chaves públicas, que certifica que uma chave pública pertence a uma entidade específica (KUROSE et al., 2010), e a certificação de chaves públicas é utilizada no protocolo *SSL*, visto anteriormente.

Uma maneira de se obter o certificado é através de uma entidade conhecida como entidade certificadora, que é responsável por validar e emitir certificados, e verifica se uma entidade é realmente quem ela diz ser. Após essa validação, a entidade certificadora cria um certificado que vincula a chave pública da entidade a essa verificação, assim o certificado passa a ter a chave pública e a identificação de que é realmente o proprietário daquela chave pública.

Porém, essas entidades certificadoras cobram por esse serviço, e os certificados também possuem validade. Uma outra maneira de se obter certificados é a partir dos certificados autoassinados, que são os certificados gerados por conta própria através de ferramentas, um exemplo de ferramenta é a ferramenta OpenSSL, que permite requisitar, assinar, gerar, exportar e converter certificados digitais.

Um problema em utilizar certificados autoassinados é que os clientes, como navegadores, aceitam apenas os certificados das entidades certificadores em quem eles confiam. Por exemplo, um navegador não vai conhecer o certificado que foi autoassinado por alguém que ele não conhece, e logo vai avisar ao usuário de que ele não está numa conexão segura, por mais que esteja sobre uma conexão *HTTPS*.

Por isso, manter certificados autoassinados pode gerar esse desconforto, porém é uma saída, principalmente para fins de testes, e melhor do que não utilizar nenhum tipo de segurança.

Entretanto, em 2014, a Universidade de Michigan e a fundação Mozilla, entre outras instituições e empresas, criaram uma nova entidade certificadora, que se chama Let's Encrypt, que fornece certificados digitais de forma gratuita, e que são reconhecidos por todos os navegadores. Let's Encrypt utiliza o protocolo *ACME* (BARNES et al., 2014), que faz o trabalho de automatizar a validação da identidade pela autoridade certificadora, facilitando a configuração de aplicações web com *HTTPS*.

Este trabalho aborda apenas os certificados autoassinados, porém, o suporte a certificados que utilizam Let's Encrypt foi referenciado nos trabalhos futuros.

3 Definição e Preparação dos Estudos

Este capítulo aborda o planejamento para a execução do projeto, contendo os procedimentos e técnicas utilizados, a fim de explicar como o desenvolvimento foi realizado para atingir os seus objetivos, servindo como base para a sua reprodução em trabalhos futuros.

O trabalho é fundamentado na proposta de uma solução, utilizando conceitos e práticas da engenharia de software, com a definição de um objetivo, a definição de um planejamento, e a execução das atividades planejadas, e posteriormente a validação dos resultados. A principal contribuição deste trabalho está em ajudar a responder a seguinte questão:

QP: *Como implantar aplicações web em sistema Debian GNU/Linux de forma automatizada e segura?*

Dado objetivo e a questão problema, este trabalho consiste em contribuir com a proposta de uma solução de gerência de configuração de software, que permita implantar aplicações web em sistemas Debian GNU/Linux de forma automatizada e segura.

Além disso, a solução proposta deverá passar por uma validação dos resultados, baseada na observação da execução da solução em exemplos de uso. Cada exemplo conterá uma aplicação real que deverá ser implantada com sucesso. Essa validação tem seus procedimentos definidos na Seção 3.3.2 e as definições dos exemplos de uso estão na Seção 3.3.3.

Inicialmente, foi realizada uma pesquisa dos trabalhos relacionados para compreender quais eram as soluções na engenharia de software que buscam a automação da implantação de aplicações web, a pesquisa tem como objetivo gerar conhecimentos para aplicação prática, relacionada a aplicações que auxiliam na implantação automatizada de software.

Os trabalhos encontrados serviram de insumo para auxiliar na construção do trabalho, a partir da análise dos resultados que foram encontrados.

3.1 Trabalhos Relacionados

Para compreender o que acontece na área relacionada com a implantação automatizada de aplicações, foi feito uma busca de alguns trabalhos relacionados recentes. Leite (2014) em seu mestrado, desenvolveu um sistema de middleware chamado CHOReOS

Enactment Engine, que possibilita a implantação distribuída e automatizada de composições de serviços web numa estrutura virtualizada, no qual opera no modelo computacional conhecido como plataforma como serviço. Seu foco é na automação da implantação aliado com a gerência de recursos de hardware.

Existem ferramentas disponíveis no mercado que também trazem a proposta de automação de instalação de aplicações, BitNami [1] é uma biblioteca de aplicativos populares e ambientes de desenvolvimento, que podem ser instalados com apenas um clique, através de uma interface web amigável. Ela automatiza todo o processo de compilar e configurar os aplicativos, e todas as suas dependências (bibliotecas de terceiros, linguagem de programação, bases de dados) para que o usuário comum não se preocupe com questões técnicas (BITNAMI, 2016).

Sandstom.io [2] é uma plataforma de código aberto para servidores pessoais. Sandstorm permite instalar facilmente as aplicações em que o Sandstorm suporta (SANDSTORM.IO, 2016). Sandstorm traz aplicações prontas para uso, em que o usuário escolhe a aplicação desejada por meio de uma interface web, e logo após isso, a aplicação já está disponível para uso.

JuJu [3] é uma ferramenta que permite implementar, configurar, gerenciar, e manter serviços de forma rápida e eficiente, automatizando a instalação e configuração de aplicações na nuvem (JUJU, 2016). É uma ferramenta mantida pela Canonical [4].

O estudo dessas aplicações serve como base para entender como funcionam as aplicações já existentes, e como elas são feitas, para que seja possível identificar as ferramentas que são utilizadas para solucionar problemas semelhantes ao problema deste trabalho.

A ferramenta JuJu (2016) utiliza uma abstração conhecida como charm, que é basicamente um arquivo com trechos de código. Um charm contém toda a lógica de que é preciso para implementar e integrar uma aplicação, contendo todo o processo de download de ferramentas, instalação e configuração de aplicações. É possível utilizar provisionadores como Chef, Puppet ou Docker.

A interação do usuário com a ferramenta JuJu é através de uma interface web, no qual os usuários selecionam os charms disponíveis e os interligam. Por exemplo, é possível adicionar o charm do MySQL e ligar ao charm de uma aplicação web, como por exemplo a MediaWiki [5]. Existem vários charms já criados pela comunidade, para que os usuários possam reaproveitar os charms já existentes.

Já Bitnami (2016) traz as aplicações prontas para uso. Assim, o usuário interage

[1] https://bitnami.com/
[2] https://sandstorm.io/
[3] https://jujucharms.com/
[4] http://www.canonical.com/
[5] https://www.mediawiki.org/wiki/MediaWiki

com uma interface web, no formato clique para instalar. Porém, o usuário só tem a disposição as aplicações disponibilizadas pelo Bitnami. Isso também é a forma em que o Sandstorm trabalha, com a diferença de que o Sandstorm é software livre, já Bitnami é um software proprietário, o que dificulta a análise da arquitetura que é utilizada por eles.

Por último, o trabalho feito por Leite (2014) é voltado para serviços web de grande escala, com foco em implantação de aplicações na nuvem. Também é possível escalar infraestrutura, utilizando o middleware construído em seu trabalho, podendo gerenciar ambientes de computação em nuvem como Amazon EC2 e Openstack. A sua aplicação utiliza o Chef como seu agente de configuração.

Com esse levantamento de trabalhos recentes, as soluções encontradas não resolvem a questão problema deste trabalho. A grande diferença é que as ferramentas encontradas não utilizam os pacotes disponibilizados pelo Debian. Alguns motivos que justificam a escolha do sistema Debian GNU/Linux são: suporte de segurança com um time específico para essas questões, integração de seus pacotes e ciclo de desenvolvimento com período de estabilização.

De toda forma, as ferramentas encontradas buscam abstrair todo o processo de instalação e configuração das aplicações, para que o usuário não tenha dificuldades na instalação de aplicações. As ferramentas também trazem uma interface web para que o usuário não precise executar tarefas via terminal.

Em resumo, por um lado, as ferramentas encontradas também utilizam do conceito de infraestrutura como código, como visto no Capítulo 2. Para isso, utilizam os provisionadores Chef ou Puppet, tornando mais simples a construção de códigos para a instalação e configuração de aplicações. Por outro lado, observa-se que o problema tratado neste trabalho não está totalmente resolvido. Assim, a proposta de solução de implantação automatizada de software em sistemas Debian, é válida, já que se diferencia das demais ferramentas encontradas.

3.2 Proposta da solução

Para a proposta da solução, foram definidas as ferramentas de apoio ao desenvolvimento da solução. Posteriormente, a escolha da ferramenta que foi utilizada para solução do problema.

3.2.1 Ferramentas de apoio

As ferramentas de apoio ao desenvolvimento de software são importantes para a organização do trabalho, auxiliando nas atividades típicas dentro de um projeto de engenharia de software. Algumas ferramentas são importantes, pois tornam mais fácil a

execução de algumas atividades. Os critérios para escolha das ferramentas são:

- **Gerenciador de repositórios de código:** Uma ferramenta que possa gerenciar diversas versões do desenvolvimento do código fonte, utilizando o sistema de controle de versão git, e que seja de preferência um software livre ou que não cobre o serviço de hospedagem. A ferramenta escolhida foi o Gitlab [6], por ser um software livre.
- **Ferramenta para documentação do projeto:** Uma ferramenta que possa documentar o projeto, o Gitlab já possui uma wiki disponível para documentar cada projeto.
- **Ferramenta de gerenciamento de tarefas:** Uma ferramenta que possa gerenciar as tarefas que serão executadas, que estão em execução ou que vão ser executadas durante o desenvolvimento do projeto. O gerenciamento de tarefas também pode ser feito no Gitlab, onde é possível criar atividades para serem feitas, e criar marcos com datas de início e fim das atividades.
- **Ferramenta de comunicação:** A ferramenta de comunicação será importante para tirar dúvidas rápidas em relação a dificuldades e desafios do trabalho, as comunidades de software livre costumam usar listas de e-mail e canais no *IRC* para a comunicação de seus desenvolvedores. Logo, toda comunicação, é feita pelos canais de *IRC* das ferramentas e suas respectivas listas de e-mail.

3.2.2 Ferramenta Shak

A solução proposta neste trabalho será baseada na evolução da ferramenta Shak (Self Hosting Applications Kit). Ela tem o objetivo de facilitar ao máximo que usuários sem conhecimento técnico possam ter os seus próprios serviços de Internet, garantindo a sua privacidade e segurança.

Durante o Google Summer of Code 2015 [7], o autor deste trabalho realizou várias atividades relacionadas à evolução da ferramenta Shak. Tornando a ferramenta, uma solução para implantação automatizada de aplicações web em sistemas Debian, justificando assim a sua escolha.

De acordo com Terceiro (2016), esta plataforma está concebida de acordo com os seguintes princípios:

Base: Pacotes Debian

A plataforma Shak utiliza aplicações que são distribuídas oficialmente pelo sistema de pacotes do Debian. Essas aplicações utilizam os pacotes do Debian, que fornecem

[6] http://gitlab.com/
[7] https://www.google-melange.com/gsoc/project/details/google/gsoc2015/thiagovsk/5757334940811264

atualizações consistentes de correção e de segurança, e são utilizados por uma grande quantidade de usuários que reportam problemas a serem resolvidos.

Nova abstração: Aplicação

Para resolver as questões que não podem ser resolvidas a nível de pacotes, o Shak introduz uma nova abstração: a aplicação. Uma aplicação geralmente contém mais de um pacote, inclusive necessita de configurações em vários pacotes.

Uma vez que o usuário seleciona uma determinada aplicação para ser instalada, os pacotes necessários são instalados e as configurações necessárias são feitas de forma automática, fornecendo de fato um instalador de um clique para aplicações suportadas.

Diferencial para outras soluções existentes

Existem outras soluções disponíveis para instalação de aplicações por um clique, como Bitnami e Sandstorm, mas o Shak se diferencia delas nas seguintes características:

- O Shak reutiliza o trabalho dos mantenedores Debian, que fornecem pacotes de alta qualidade.
- O Shak reutiliza a infraestrutura do Debian, que é utilizada por várias outras distribuições GNU/Linux baseadas no Debian, como por exemplo: Ubuntu [8] e Linux Mint [9].

A arquitetura simplificada do Shak, na Figura 1, mostra quatro componentes importantes. Os dois primeiros são a interface web e a interface por linha de comando, essas duas interfaces são responsáveis por coletar os dados do usuário, como por exemplo, a aplicação que se deseja instalar.

Os dados vindos dessas interfaces são gerenciados pela aplicação Shak, e mantidos na base de dados chamada repositório. O repositório do Shak contém as aplicações que foram adicionadas, e suas informações.

No Shak, uma aplicação possui uma abstração que está exemplificada na Figura 2, onde uma aplicação é uma classe Ruby, e cada aplicação possui um livro de receitas Chef associado. Além disso, cada livro de receitas pode definir quais são as entradas que a aplicação necessita, como por exemplo, o endereço da aplicação. Essas entradas são informadas pela aplicação web, ou pela interface de linha de comando. As entradas também serão processadas pela aplicação Shak que guarda essas informações no seu repositório de aplicações.

Em resumo, a estrutura do Shak utiliza:

[8] http://www.ubuntu.com/
[9] https://www.linuxmint.com/

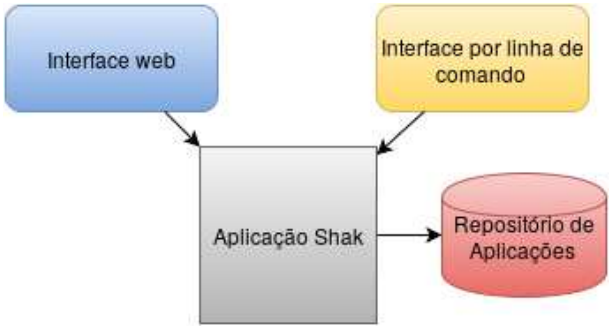

Figura 1 – Arquitetura simplificada da ferramenta Shak

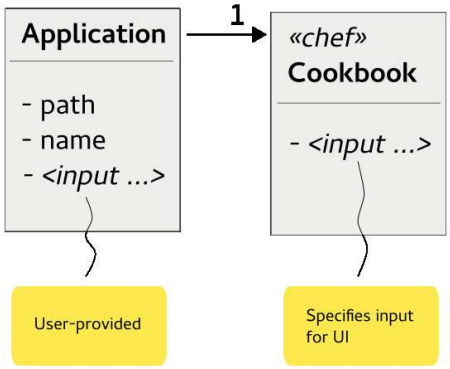

Figura 2 – Modelo de dados do Shak

- **Livro de Receitas Chef:** Uso de livros de receitas para poder organizar a instalação de cada componente, contendo um livro de receitas para cada aplicação que for instalada.
- **Código Ruby:** Arquitetura desenvolvida na linguagem Ruby, com programação orientado a objetos.
- **Servidor web:** O servidor web é o Nginx, que é um servidor *HTTP* de alto desempenho (NGINX, 2016).
- **Pacotes Debian:** As bibliotecas utilizadas estão incluídos na distribuição oficial do Debian.
- **Gems:** Uso de gems para a aplicação, uma gem nada mais é do que uma biblioteca Ruby, que provê um formato padrão para a distribuição de programas Ruby (RUBYGEMS, 2016).

- **Código Shellscript:** Os códigos ShellScript são utilizados principalmente para configurações de ambiente de desenvolvimento.

Para adicionar uma aplicação que esteja disponível nos repositórios oficiais do Debian na ferramenta Shak, é necessário construir o seu respectivo livro de receitas, utilizando a ferramenta Chef.

Para adicionar uma receita Chef à ferramenta Shak, existe um recurso no Shak que automatiza todo esse processo, no qual cria uma estrutura padrão de livro de receitas Chef. Para usar esse recurso, é necessário executar o comando *rake cookbook* no terminal, além de informar o nome do livro de receitas desejado. O resultado disso é a criação de arquivos e pastas, que formam a estrutura de um livro de receitas. Um exemplo disso é o livro de receitas da aplicação MoinMoin, como na Figura 3.

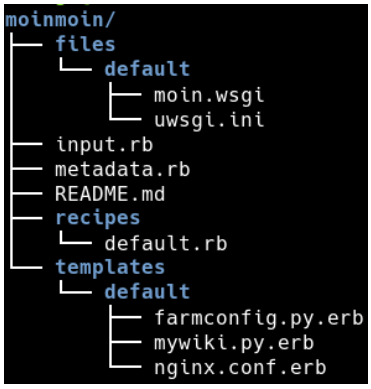

Figura 3 – Estrutura do livro de receitas no Shak.

As receitas ficam no diretório *recipes*, o Shak sempre executa a receita que está no arquivo *default.rb*, mas isso não impede a criação de outras receitas em outros arquivos, desde que sejam incluídas nesse arquivo.

Os arquivos de configuração ficam nos diretórios *files* e *templates*, a diferença entre eles é que no diretório *files* ficam os arquivos que não são alterados, ou seja, esses arquivos não vão precisar de alterações durante a implantação.

Já no diretório *templates*, ficam os arquivos de configuração que sofrem alterações durante a implantação. Além disso, os arquivos template possuem uma característica, eles são arquivos no formato .erb *(Embedded Ruby)*, que permite que o implantador adicione código Ruby no arquivo de texto. Isso permite configurações mais robustas, durante a implantação da aplicação.

Também existem os arquivos que estão na raiz do livro de receitas. O *input.rb* é o arquivo em que é possível declarar os atributos das aplicações. Também é possível declarar quais atributos são obrigatórios e quais são únicos.

O código 3.1 possui um exemplo de arquivo *input.rb*.

Código 3.1 – Exemplo de código no arquivo input.rb

```
text :hostname do
  title 'Website domain'
  mandatory
end

text :path do
  title 'Path'
  default '/'
end

unique :hostname, :path
```

Por fim, os arquivos *metadata.rb* e *README.md*, o primeiro possui informações sobre o livro de receitas, como o nome, o mantenedor do livro de receitas, e sua licença. Já o segundo, possui instruções para utilizar a receita.

Com a estrutura criada, foi necessário construir as receitas, para automatizar a implantação das aplicações. A atividade de criação de receitas e suporte das aplicações foi uma das atividades deste trabalho, sendo assim, adicionando novas aplicações web à ferramenta Shak.

3.2.3 Gerência de Ambientes de Desenvolvimento

Uma das necessidades do desenvolvimento do projeto é ter um ambiente de desenvolvimento flexível, que possa ser rapidamente construído e destruído. Onde seja possível testar as implantações feitas pela ferramenta Shak e verificar os erros da implantação, caso aconteçam.

Para isso, foi necessário utilizar alguma ferramenta que automatize o processo de construir ambientes de desenvolvimento. Construir ambientes manualmente pode ser um processo longo e demorado, além disso, é possível que ocorra erros por desatenção ao executar vários passos manuais. A ferramenta escolhida serviu para auxiliar na criação das máquinas virtuais, que eram os nós alvo das implantações.

A ferramenta que foi escolhida para auxiliar na gerência de um ambiente de desenvolvimento é a ferramenta Vagrant [10]. Com o Vagrant é possível gerenciar a criação de máquinas virtuais para os ambientes de desenvolvimento do projeto.

O ambiente de desenvolvimento escolhido, foi um ambiente com o sistema Debian na sua versão sid 64 bits(que é a versão que contém os pacotes mais atuais, conhecida como versão instável), apesar de a versão ser a versão instável, isso foi importante para

[10] https://www.vagrantup.com/

que tenha disponível as versões mais novas dos pacotes das ferramentas escolhidas. Além disso, é preciso que a ferramenta Shak suporte primeiramente a versão instável do Debian, para que assim fosse incluída no próximo lançamento estável do Debian.

3.3 Planejamento das Atividades

As atividades planejadas para a evolução da ferramenta Shak são baseadas nas aplicações que foram escolhidas na Seção 3.3.3. Essas foram as aplicações que tiveram sua implantação automatizadas pela ferramenta.

No trabalho, foi feito uma fase exploratória, no qual o planejamento foi suportar a instalação do Owncloud e Wordpress de forma automatizada. A escolha das duas ferramentas foi feita pelo autor e pelo co-orientador deste trabalho, levando em consideração a popularidade das duas ferramentas nas comunidades de software livre.

Essa fase exploratória foi necessária para evoluir a ferramenta , visto que a ferramenta Shak só suportava aplicações estáticas, ou seja, apenas aplicações que contenham código *HTML*, *CSS* e *Javascript*.

A fase exploratória foi importante para apoiar na definição das fases e procedimentos para implantação automatizada. Além de que, a partir dela, foram definidas características importantes que devem ser levadas em consideração para a escolha das próximas aplicações.

As atividades levantadas foram:

1. Suporte a instalação automatizada do Wordpress.
2. Suporte a instalação automatizada do Owncloud.
3. Forçar as aplicações Wordpress e Owncloud a utilizar o protocolo HTTPS.
4. Suporte a instalação automatizada do servidor de e-mail.
5. Suporte a instalação automatizada do MoinMoin.
6. Suporte a instalação automatizada do Roundcube.
7. Suporte a instalação automatizada do Noosfero.

Ao verificar que tanto a ferramenta Owncloud como a ferramenta Wordpress precisam de configurações de servidor de e-mail para algumas funcionalidades, também foi adicionado a atividade de criação de uma receita para a automatizar a configuração de servidor de e-mail. Como aspecto de segurança, ficou definido que tanto a aplicação Owncloud como Wordpress deveriam usar sempre HTTPS por padrão, isso envolve também

uma estratégia de como serão gerenciados os certificados de segurança para aplicar o protocolo HTTPS.

Após a fase exploratória, outras aplicações foram escolhidas, são elas o MoinMoin, Roundcube e Noosfero, de acordo com as características desejadas, definidas na Seção 3.3.2

Dentro da implantação automatizada de software Debian GNU/Linux, existem várias configurações possíveis. Neste trabalho, existem algumas características que foram levadas em consideração, são elas:

1. Segurança na implantação de aplicações web, como por exemplo utilizar protocolos como *HTTPS*.
2. Configuração de múltiplas instâncias de aplicações, utilizando hospedagem virtual.

Por fim, foi necessário definir procedimentos para a execução da implantação automatizada das aplicações. Considerando o processo de implantação de software visto no Capítulo 2 e a referência dos trabalhos relacionados. Foi necessário definir as fases e os procedimentos para implantação automatizada, definindo as etapas da implantação.

3.3.1 Fases e Procedimentos para implantação

Primeiramente, era necessária a definição das fases e os procedimentos para a implantação automatizada, de acordo com as fases que compõem o processo de implantação de aplicações, e de acordo com a Subseção 2.1.1.

Neste trabalho, foi adicionado na fase de configuração, a atividade de configuração de múltiplas instâncias, que é a fase responsável por habilitar a implantação de múltiplas instâncias de uma aplicação. Isso possibilita a configuração de várias instâncias da mesma aplicação sem duplicação de recursos. A sequência das fases estão na Figura 4:

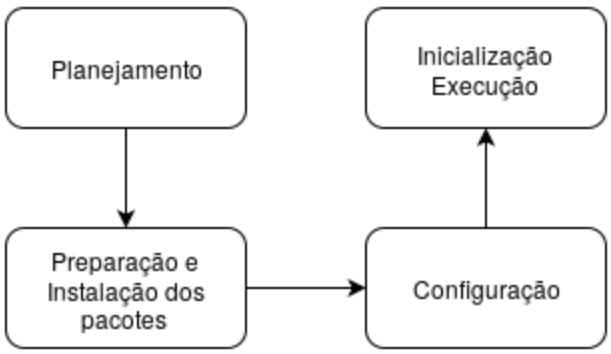

Figura 4 – Sequência de fases para implantação automatizada de aplicações

1. **Planejamento**: Identificar os componentes mínimos necessários na implantação da aplicação.

2. **Preparação e Instalação de Pacotes**: Preparar o ambiente alvo, isso envolve configuração do sistema operacional e instalação de dependências necessárias.

3. **Configuração**: Editar os arquivos de configuração necessários, tanto da aplicação como de suas dependências. Além disso, configurar o suporte de múltiplas instâncias da aplicação.

4. **Inicialização/Execução**: Testar as receitas construídas, utilizando a ferramenta Shak.

Também foram definidas as caraterísticas de segurança que as aplicações devem ter na sua implantação, foi definido que as aplicações sempre usem protocolos seguros, como o *HTTPS*. Para as aplicações web é necessário utilizar o protocolo HTTPS, para aplicações de e-mail é necessário utilizar os protocolos *SMTP* e *IMAPS*.

3.3.2 Validação da solução

Para validar a evolução da ferramenta, foram feitos exemplos de uso, com as aplicações definidas, ou seja, aplicações que tenham todo o seu processo de instalação e configuração automatizado, a fim de refinar e evoluir a ferramenta, conforme problemas forem surgindo. A escolha desses exemplos de uso devem ser feitas a partir de aplicações reais e conhecidas na comunidade de software livre.

Depois da fase exploratória, observam-se outras características importantes, e que devem ser levados em consideração para a escolha das aplicações, além da popularidade, utilizado anteriormente. As seguintes características foram levadas em consideração para a escolha das aplicações:

Aplicações empacotadas no Debian: Como o intuito do trabalho é realizar implantações múltiplas a partir de um pacote único, tais aplicações devem estar empacotadas e disponíveis para instalação nos servidores do Debian. Caso a aplicação não esteja disponível no Debian, é necessário empacotá-la e distribuir nos servidores oficiais do Debian.

Servidor web compatível: As ferramentas escolhidas possuem, no mínimo, o servidor web compatível. Por exemplo, todas as aplicações devem possuir suporte ao servidor Nginx ou Apache.

Aplicações com comunidades ativas: É importante que os softwares escolhidos possuam comunidades ativas, isso pode ajudar na resolução de possíveis problemas.

Logo, aplicações abandonadas pela sua comunidade foram evitadas, e aplicações com comunidade de desenvolvedores e usuários ativas foram priorizadas. Por utilizar a distribuição instável, é possível que sejam descobertos erros na integração dessas ferramentas com o Debian. Tais erros e melhorias são reportados para o Debian, ou até solucionados, e devolvidos aos mantenedores dos pacotes.

Documentação do software: A documentação do software também foi levada em consideração, principalmente a documentação da instalação e configuração dentro da ferramenta. As ferramentas que não possuem documentação de instalação e configuração foram evitadas.

Aplicações com suporte a federação: Aplicações federadas permitem utilização de métodos de autenticação única para cada instância da aplicação, mantendo a compatibilidade entre elas, um exemplo é a aplicação Owncloud, que permite compartilhar seus arquivos na nuvem independente do Owncloud que estiver usando, isto é possível utilizando apenas um identificador único, chamado ID Federated Cloud, assim possibilitando que os arquivos do usuário sejam compartilhados entre suas instâncias do Owncloud na nuvem.

A partir dessas características definidas, também era necessário encontrar os exemplos de uso utilizados para a execução da solução.

3.3.3 Exemplos de uso: busca dos pacotes das aplicações

Para encontrar as aplicações que possam se encaixar dentro desses parâmetros foi necessário buscar por alguns exemplos de uso. Para a escolha das aplicações que foram utilizadas como exemplos de uso, foi necessário fazer uma busca nas aplicações web que possuem suporte a configuração de múltiplas instâncias, essa busca levou em consideração também, a documentação para realizar tal configuração. Foram levantados algumas aplicações web da seguinte forma:

```
apt-cache search web | wc -l
```

O resultado obtido com pacotes que contenham a palavra web recebe o resultado de 3470 pacotes de diversas aplicações ou módulos de aplicações, como Wordpress; Owncloud; Drupal; Mailman e Chromium.

Foram escolhidas as aplicações Wordpress, Owncloud, MoinMoin e Roundcube, por conter as características definidas anteriormente. Além disso, também foi escolhida a aplicação Noosfero, que é uma aplicação que ainda não está no Debian, porém neste trabalho será feito um esforço inicial para que isso seja possível.

Para realizar a implantação das aplicações, foi importante que as instalações e configurações fossem executadas num ambiente limpo. Os testes foram criados em máquinas virtuais com a configuração conhecida como mínima, que contém instalado apenas as aplicações necessárias para o funcionamento do sistema operacional.

Após a execução da implantação, o testador verificou o perfeito funcionamento de algumas funcionalidades básicas das aplicações escolhidas, e principalmente verificar a implantação de várias instâncias da mesma aplicação no mesmo servidor destino, observando o perfeito funcionamento de todas as instâncias implantadas, para assim, validar a implantação de múltiplas instâncias, conforme apresentado nos resultados do próximo capítulo.

4 Resultados obtidos

Este capítulo relata os resultados encontrados, a partir da proposta de definição e preparação dos estudos, no Capítulo 3. Também dito no Capítulo 3, as aplicações escolhidas foram Owncloud; Wordpress; Moinmoin; Roundcube; e Noosfero. Além da aplicação de serviço de e-mail.

Cada fase da implantação será tratada como uma subseção na descrição de cada aplicação que foi escolhida para exemplo de uso. Todas as atividades foram organizadas, via *issues*, no repositório do próprio projeto Shak [1].

4.1 Wordpress

Wordpress (2016) é uma plataforma semântica de vanguarda para publicação pessoal, com foco na estética, nos padrões web e na usabilidade, ao mesmo tempo é um software livre. Além disso, Wordpress é um dos maiores softwares de publicação de conteúdo .

Primeiramente, foram seguidos os procedimentos definidos na Seção 3.2. Para isso, era necessário definir as fases e procedimentos da implantação automatizada, seguindo as fases que compõem o processo proposto descrito na Seção 3.3.1.

4.1.1 Planejamento

Nesta fase foi preciso definir quais são as dependências mínimas para o funcionamento da aplicação, tais como: banco de dados; pacotes pré-instalados; e aplicações pré-configuradas. Para o Wordpress foram escolhidas:

- **Pacote Wordpress para o Debian**
- **Pacote Nginx para o Debian**
- **Pacote MySQL para o Debian**
- **Arquivos de configuração:** Criação dos arquivos de configuração necessários para configurar o Wordpress, o servidor web Nginx e o banco de dados MySQL.

A instalação e execução desses recursos foram feitas nas fases seguintes.

[1] https://gitlab.com/shak/shak/issues

4.1.2 Preparação e Instalação de Pacotes

Nesta fase foram definidos os procedimentos necessários para preparar o ambiente alvo, para que o Wordpress possa ser executado. Isso envolveu a configuração do sistema operacional; instalação e configuração de dependências necessárias; e a transferência do componente para o servidor onde ele será executado.

Para esse procedimento, foi necessário instalar os pacotes *php5-fpm*; o pacote do banco de dados MySQL; e o pacote *nginx*. Além disso, habilitar o serviço do *php5-fpm*.

Para executar essas instalações, foi preciso criar uma receita no livro de receitas do Wordpress. Foi necessário criar o próprio livro de receitas no Shak. Para isso, existe o comando *rake cookbook* que cria a estrutura básica de um livro de receitas que deve ser usado pelo Chef.

Assim, com a estrutura gerada pelo rake, é possível declarar as dependências do Wordpress dentro do arquivo *default.rb*, dentro do diretório *recipes*. Nesse arquivo, são declarados: os pacotes que devem ser instalados; os arquivos de configuração; os serviços que precisam executar; os diretórios; e permissões de usuários.

O código 4.1 é um exemplo de como declarar pacotes e serviços dentro de uma receita Chef.

Código 4.1 – Exemplo de criação de serviço do mysql com o chef

```
package "mysql-server"
service "mysql" do
  action :start
end
```

4.1.3 Configuração

O primeiro arquivo de configuração é o *config.php*. Esse arquivo contém a configuração onde ficam as informações de banco de dados, como nome do banco de dados; login do usuário do banco de dados; senha; o endereço do banco de dados. Além disso, cada aplicação deve possuir um diretório onde ficam os arquivos estáticos, como: temas; galeria de mídias; e plugins de cada instância do Wordpress.

Esse arquivo deve ficar no diretório */etc/wordpress/* e seu nome deve conter a seguinte estrutura: *config-nomehost.php*, onde o nomehost deve ser o endereço desejado, como por exemplo config-fga.unb.br.php. O segundo arquivo é o *database.sql*, que é um pequeno script *SQL* que cria o banco de dados do Wordpress e dá os privilégios ao usuário desejado. Por fim, o arquivo de configuração do Nginx, com a configuração do servidor web.

Na receita chef, esses arquivos de configuração serão criados dentro do diretório *template*, com o conteúdo dos arquivos de configuração. A ação que cria um arquivo na receita chef é declarada dentro do arquivo *default.rb*, de forma semelhante como foi feito com a declaração dos pacotes.

O código 4.2 é um exemplo de como gerenciar templates numa receita Chef.

Código 4.2 – Exemplo de criação de templates com o chef

```
template "Create␣autoconfig.php␣in␣/etc/wordpress" do
  source "autoconfig.php.erb"
  path "/etc/wordpress/autoconfig.php"
  owner "www-data"
  group "www-data"
end
```

4.1.4 Múltiplas Instâncias

O Wordpress já suporta a funcionalidade de múltiplas instâncias nativamente. Para configurá-lo, foi necessário que alguns diretórios do Wordpress fossem isolados, sendo um para cada aplicação. Cada instância precisa ter seus diretórios isolados.

Outro fator importante é que, cada instância tenha seu banco de dados. Para solucionar esse problema, o Shak possui um recurso interessante, onde cada aplicação possui um atributo *id*, que é um atributo único para cada instância executada pelo Shak. Com isso, foi possível criar diretórios personalizados com o id de cada aplicação, e também bancos de dados específicos e arquivos *config.php* específicos.

Por fim, era necessário um arquivo de configuração Nginx para cada aplicação, como visto no Capítulo 2. Todo arquivo de configuração do Nginx possui um bloco *server*, e cada bloco *server* equivale a uma hospedagem virtual. Por isso, as aplicações também se tornam independentes, podendo ser acessadas pelo mesmo servidor, mas com endereços diferentes.

4.1.5 Inicialização

Após a construção da receita do Wordpress, foi necessário testar a receita construída. Para isso, o Shak precisa de duas informações importantes. A primeira, é a aplicação que será instalada, e segunda, o endereço destino. Como a implantação foi um ambiente de desenvolvimento, não é preciso configurar um ip ou configurar um *DNS*. Foi preciso adicionar o endereço desejado no arquivo */etc/hosts*. Assim, mesmo que esteja na sua máquina local, era possível acessar um endereço mais familiar em seu navegador.

Um exemplo da execução da instalação via Shak é:

Código 4.3 – Exemplo de execução de instalação do wordpress com shak

```
shak install wordpress hostname=wordpress.dev
```

O Shak processa os dados informados, registrando os dados em seu repositório, e inicia a instalação da aplicação, executando sua receita. Dessa forma, o Chef inicia: o processo de instalação dos pacotes; criação dos arquivos de configuração; incia os serviços desejados.

Ao fim do procedimento, a aplicação estará pronta para uso no endereço escolhido, é possível também verificar o estado atual da aplicação.

Para listar todas as aplicações que estão no repositório do Shak e verificar suas informações é preciso executar:

Código 4.4 – Listagem de aplicações instaladas pelo shak

```
shak list
```

Assim, é possível verificar o endereço em que a aplicação está disponível, além do estado atual da aplicação.

Como dito na Seção 3.3.2, a validação da instalação foi feita a partir da verificação do funcionamento das funcionalidades básicas da aplicação, além de testar múltiplas instalações no mesmo servidor e verificar se todas as aplicações instaladas eram acessíveis através do protocolo *HTTPS*.

4.2 Owncloud

Owncloud (2016) é uma ferramenta para compartilhamento de arquivos, é um software livre em que é possível compartilhar um ou mais arquivos e diretórios do seu computador na nuvem, e sincronizá-los com o seu servidor Owncloud. Esses arquivos são imediatamente sincronizados com o servidor e disponibilizados para outros dispositivos que utilizam o ambiente de trabalho Owncloud ou aplicativo Android ou aplicativo IOS.

Owncloud é uma ferramenta que se assemelha a aplicações conhecidas, como Dropbox [2] e Google Drive [3], porém, por ser software livre, é possível instalar Owncloud em servidores que o usuário preferir. Em alguns casos, usuários criam nuvens privadas com owncloud, em pequenos servidores configurados em casa.

Para a ferramenta Owncloud, foram seguidos os mesmos passos feitos na aplicação Wordpress na Seção 4.1.

[2] https://www.dropbox.com/pt_BR/
[3] https://www.google.com/intl/pt-BR/drive/

4.2.1 Planejamento

As dependências para o funcionamento do Owncloud são:

- **Pacote Owncloud para o Debian**
- **Pacote Nginx para o Debian**
- **Pacote Postgresql para o Debian**
- **Arquivos de configuração:** Criação dos arquivos de configuração necessários para configurar o Owncloud, o servidor web Nginx e o banco de dados Postgresql.

Diferentemente da aplicação Wordpress, na aplicação Owncloud o banco de dados escolhido foi o postgresql. Na documentação do Wordpress é recomendado o uso do banco de dados MySQL, porém no Owncloud fica a escolha do desenvolvedor. A escolha do Postgresql teve como motivação, a possibilidade de trabalhar com dois bancos de dados diferentes, aumentando o suporte do Shak. Assim, quando existirem aplicações que suportam apenas o Postgresql, o Shak já terá uma estrutura pronta.

4.2.2 Preparação e Instalação de Pacotes

Para esse procedimento, foi necessário instalar os pacotes *php5-fpm*; o banco de dados Postgresql; e o pacote Nginx. Além disso, habilitar o serviço do php5-fpm. Também foi necessário instalar o pacote php5-pgsql, que é um módulo para conexões de banco de dados Postgresql, necessário para o funcionamento de aplicações na linguagem PHP com o banco de dados Postgresql.

Para executar essas instalações, foi preciso criar uma receita no livro de receitas do Owncloud, da mesma forma que em Wordpress.

Com a estrutura inicial, foi possível declarar as dependências do Owncloud dentro do arquivo *default.rb*, que se encontra dentro do diretório *recipes*.

4.2.3 Configuração

O primeiro arquivo de configuração é o *autoconfig.php*, esse arquivo contém informações importantes, como: nome do banco de dados; login do usuário do banco de dados; senha do usuário do banco de dados; o endereço do banco de dados; o diretório onde irá ficar os arquivos de configuração do Owncloud; e o login e senha do administrador.

Além disso, foi necessário criar o diretório de conteúdos públicos do Owncloud, que por padrão devem ficar em */etc/owncloud*. O segundo arquivo é o *postgresql-conf.sql*, que é um pequeno script *SQL* que cria o banco de dados do Owncloud, e configura os

privilégios ao usuário desejado. Por fim, a configuração do servidor web, feitos através de templates, da mesma forma que a aplicação Wordpress.

4.2.4 Múltiplas Instâncias

Diferentemente do Wordpress, o Owncloud não suporta nativamente múltiplas instâncias. Porém isso não foi um impeditivo na execução do trabalho, com uma busca, encontrou-se uma discussão no repositório oficial do Owncloud relacionado a implementação dessa funcionalidade.

Foi discutido uma proposta [4] de solução que permitiria a configuração de múltiplas instâncias. O resultado desta discussão foi que, os desenvolvedores do Owncloud não acharam relevante a funcionalidade, mas caso o desenvolvedor ache necessário, ele poderia fazer essa alteração diretamente no código fonte.

Porém, manter isso no Shak não seria uma boa solução. Uma solução para esse problema seria, enviar uma contribuição ao pacote do Owncloud no Debian, onde assim a contribuição feita poderia ser facilmente utilizada por mais pessoas que queiram essa funcionalidade, bastando utilizar a versão disponibilizada nos servidores do Debian.

Com isso, foi feito a contribuição que adicionaria a funcionalidade de múltiplas instâncias para o Owncloud, e enviado ao mantenedor do pacote do Owncloud no Debian. Na discussão [5], a contribuição foi bem vista pelo mantenedor do pacote.

Os procedimentos feitos para suportar múltiplas instâncias no Owncloud são bem semelhantes aos do Wordpress, criando um diretório de dados para cada aplicação; um arquivo de configuração para cada aplicação; e um banco de dados para cada aplicação. Seguindo a mesma abordagem do Wordpress, também foi criado uma hospedagem virtual para cada instância do Owncloud.

4.2.5 Inicialização

A inicialização da aplicação se dá da mesma forma que na aplicação Wordpress.

Para executar a instalação do Owncloud basta:

Código 4.5 – Exemplo de execução de instalação do Owncloud com shak

```
shak install owncloud hostname=owncloud.dev
```

A validação da instalação foi feita conforme dito na Seção 3.3.2, da mesma forma que em Wordpress.

[4] https://github.com/owncloud/core/pull/16424
[5] https://bugs.debian.org/cgi-bin/bugreport.cgi?bug=789726

4.3 Servidor de e-mail

Tanto Owncloud como Wordpress, possuem funcionalidades que utilizam serviços de e-mail. Assim, a construção de um servidor de e-mail possui bastante relevância, para que qualquer aplicação que precisar de um servidor de e-mail, possa utilizar o servidor de e-mail que o Shak possa fornecer.

Para adicionar o suporte a servidor de e-mail, também foram seguidos os mesmos passos feitos no Wordpress na Seção 4.1 e no Owncloud na Seção4.5.2.

4.3.1 Planejamento

Primeiramente, foi definido que o protocolo escolhido fosse o protocolo *IMAP*, já que o protocolo *IMAP* é um protocolo online, ou seja, ele se conecta ao servidor e realiza o download das mensagens, depois ainda mantém a conexão para que as alterações e mensagens novas sejam atualizadas em tempo real. Diferentemente do protocolo *POP3* que é um protocolo offline, onde após o download das mensagens encerra a conexão. Outra vantagem do *IMAP* em relação ao *POP3*, que é que o *IMAP* mantém uma cópia de mensagens no servidor, ideal para quem precisa acessar os e-mails de mais de um local ou a partir de mais de um dispositivo.

O servidor *IMAP* escolhido foi o Dovecot, que é um servidor de e-mail *IMAP*, além disso, Dovecot é um software livre simples de configurar, requer nenhuma administração especial, e ainda usa muito pouca memória (DOVECOT, 2016).

O agente de transferência de e-mails escolhido foi o Postfix, que é um software livre para envio e entrega de e-mails. A sua escolha foi feita pela facilidade de integração com o Dovecot. Postfix é o software responsável pelo método de entrega de e-mail utilizando *SMTP*, como protocolo de transferência de e-mails.

Por fim, a aplicação que servirá como anti-spam será o Apache SpamAssassin, que é uma plataforma anti-spam que dá aos administradores de servidor de e-mail, um filtro para classificar e-mails e bloquear os e-mails que julgarem como spam (SPAMASSASIN, 2016).

Em resumo, o servidor de e-mail é composto de:

- **Pacote Dovecot para o Debian**
- **Pacote Postfix para o Debian**
- **Pacote spamassasin para o Debian**
- **Arquivos de configuração** Criação dos arquivos de configuração necessários para configurar o Spamassasin, Dovecot e Postfix.

4.3.2 Preparação e Instalação de Pacotes

Nesta fase foi necessário separar as dependências dos pacotes para cada ferramenta escolhida, para compor o servidor de e-mail. Primeiramente, para o Postfix, são necessários os pacotes *postfix* e *bsd-mailx*, além de habilitar o serviço do Postfix. Para o Dovecot, foi necessário instalar o pacote *dovecot-imadp* além de habilitar o serviço do Dovecot. Para o Spamassassin, eram necessários os pacotes *spamassassin* e o pacote *spamc*, além de habilitar o serviço do Spamassassin.

Na receita do servidor de e-mail, cada aplicação possui a sua receita específica, e o arquivo *default.rb*, que contêm a receita do servidor de e-mail.

Esse arquivo contém apenas a chamada das receitas das aplicações, como no código 4.6:

Código 4.6 – Exemplo da receita de e-mail composta pelas receitas das outras aplicações

```
include_recipe 'email::dovecot'
include_recipe 'email::postfix'
include_recipe 'email::spamassasin'
```

A função de *include_recipe* do Chef permite que, aplicações que são compostas por outras aplicações menores, possam ter suas receitas separadas em diferentes arquivos, sendo uma receita para cada aplicação menor, que compõe o todo. Com isso, foi possível incluir esses arquivos na receita que será executada. No caso do Shak, a receita que sempre será executada é a receita que está no arquivo *default.rb* no diretório *recipe*.

4.3.3 Configuração

Primeiramente, para a configuração do Dovecot, foram utilizados 5 arquivos de configuração, o primeiro é o *10-mail.conf* que é um arquivo de configuração para indicar onde é o local no qual os e-mails estão. O segundo arquivo, é o *10-ssl.conf* que é o arquivo em que possui os caminhos do certificado *SSL* e da chave *SSL*.

O terceiro arquivo é o arquivo *11-postfix-auth.conf*, arquivo responsável por indicar o caminho do arquivo de autenticação do Postfix. O quarto arquivo é o arquivo *20-imap.conf*, onde é indicado o tamanho máximo de conexões por ip permitidas no servidor. Por fim, o arquivo *20-disable-imap-non-ssl.conf*, que permitirá que o imap utilize sempre uma conexão segura utilizando o protocolo *IMAPS* (*IMAP* + *SSL*), assim bloqueando qualquer conexão segura ou tentativa de conexão insegura.

Para o posfix, são apenas dois arquivos de configuração, o *main.cf* e o *master.cf*. O *main.cf* é onde se configuram os parâmetros mínimos de configuração, esse arquivo também contém as configurações que filtram os e-mails com Spamassassin e algumas configurações do *SMTP*.

No *master.cf*, é definido como um programa cliente se conecta a um serviço, e qual programa é executado quando um serviço é solicitado. Neste caso, foi necessário indicar que o servidor de e-mail utilize *SSL*, com autenticação segura, apontando o caminho dos arquivos de chave e certificados.

Por fim, a configuração do spamassasin. São necessários dois arquivos de configuração. O arquivo *spamassassin*, que é o arquivo de configurações padrão para o spamassasin e o *local.cf* que são as configurações locais. No Spamassassin, foi preciso habilitar o serviço do spamassasin e configurar o caminho dos arquivo de log.

Já o arquivo *local.cf*, possui o parâmetro required_score. Esse parâmetro possui um nível de 0 a 10, em que são classificados os e-mails como spam. Por padrão, esse valor é cinco. Porém, se o usuário quiser aumentar o filtro, é necessário aumentar para valores como seis ou sete, até chegar em um nível de confiança em que o spamassasin cuide dos casos falsos-positivos, ou seja, e-mails que não são spam, porém foram interpretados como spam.

4.3.4 Inicialização

A inicialização da aplicação se dá da mesma forma que nas aplicações Wordpress e Owncloud.

Para executar a instalação do servidor de e-mail basta:

Código 4.7 – Exemplo de execução de instalação do servidor de e-mail com shak

```
Shak install e-mail hostname=owncloud.dev
```

Para testar o funcionamento, foram feitos dois procedimentos. O primeiro, era utilizar um cliente de e-mail para testes. O cliente de e-mail escolhido foi o mutt, utilizando suas funções básicas de enviar e receber e-mails. O segundo passo, foi realizar conexões com o servidor de e-mail via telnet.

Para isso foi necessário executar os seguintes comandos:

Código 4.8 – Exemplo de teste de conexão telnet no servidor imap

```
telnet localhost imap
telnet localhost imaps
```

Pela configuração do servidor de e-mail, não foi possível abrir uma conexão telnet com o parâmetro imap, porém com o parâmetro imaps foi possível, isso serviu para testar que o servidor de e-mail impediu conexões que não utilizam os protocolos criptografados.

4.4 Moinmoin

(MOINMOIN, 2016) é uma ferramenta de construção de wikis, com uma grande comunidade de usuários, podendo criar páginas web facilmente editáveis. Moinmoin é software livre, alguns exemplos de wiki que utilizam o Moinmoin são: Wiki do Debian, Wiki do Python e Wiki do Apache, contendo várias documentações relacionadas aos seus softwares.

Para a implantação automatizada do MoinMon, foram seguidos os mesmos passos feitos em Wordpress 4.1 e Owncloud 4.5.2.

4.4.1 Planejamento

As dependências necessárias para a aplicação Moinmoin são:

- **Pacote python-moinmoin para o Debian**
- **Pacote Nginx para o Debian**
- **Pacote uwsgi para o Debian**
- **Arquivos de configuração:** Criação dos arquivos de configuração necessários para configurar o Uwsgi, o servidor web Nginx e o servidor web uwsgi.

Diferentemente das outras aplicações web, Moinmoin não utiliza banco de dados.

4.4.2 Preparação e Instalação de Pacotes

Para esse procedimento, foi necessário instalar os pacotes *uwsgi*, *uwsgi-plugin-python*, e o pacote do Moinmoin, chamado *python-moinmoin*. Para executar essas instalações, foi preciso criar uma receita no livro de receitas do Moinmoin, bastando executar o comando *rake cookbook* que cria a estrutura básica de um livro de receitas que deve ser usado pelo Chef.

Assim, com a estrutura inicial, foi possível declarar as dependências do Moinmoin dentro do arquivo *default.rb* dentro do diretório *recipes*.

4.4.3 Configuração

O primeiro arquivo de configuração é o *moin.wsgi*, arquivo onde é necessário informar, por exemplo, o caminho dos arquivos de configuração do Moinmoin. No caso de um pacote Debian, esse caminho por padrão é */etc/moin*. Também foi necessário editar o arquivo *uwsgi.ini*, que é o arquivo de configuração do uwsgi para a sua aplicação. Nele,

foi possível adicionar informações importantes, como por exemplo: caminho do arquivo de log do uwsgi; caminho do socket; e a quantidade máxima de requisições.

O terceiro arquivo de configuração é o *mywiki.py*, onde foi necessário informar informações básicas da wiki, como o nome da wiki e seu diretório de dados. O último arquivo de configuração é o *farmconfig.py*, que é o responsável pelos links da barra de navegação e responsável por configurar múltiplas instâncias.

Além disso, foi necessário criar o diretório de conteúdos do Moinmoin, que por escolha devem ficar em /var/lib/moin. Após isso, para que o Nginx funcione corretamente com o uwsgi, foi necessário indicar o socket do uwsgi, que foi indicado no uwsgi.ini.

4.4.4 Múltiplas Instâncias

Moinmoin possui nativamente a funcionalidade que permite a criação de múltiplas instâncias de wiki no mesmo servidor. Para isso, foi necessário editar um arquivo, que é chamado de *farmconfig.py*, com todas as wikis, e com seus respectivos identificadores e domínios. Assim, o Moinmoin consegue gerenciar as wikis separadamente.

Uma outra etapa, era fazer com que o Shak entenda que existem outras wikis já configuradas. A cada nova instância o arquivo *farmconfig.py* deve ser atualizado com as novas informações de wiki, sem remover as informações das outras wikis anteriormente inseridas. Para isso, foi adicionado numa lista de wikis, para que a cada vez que surja uma wiki nova, apenas as informações novas são inseridas no arquivo *farmconfig.py*.

4.4.5 Inicialização

A inicialização da aplicação se dá da mesma forma que nas aplicações Wordpress e Owncloud.

Para executar a instalação do MoinMoin basta:

Código 4.9 – Exemplo de execução de instalação do owncloud com shak

```
shak install moinmoin hostname=moinmoin.dev
```

A validação da instalação foi feita conforme dito na Seção 3.3.2, da mesma forma que em Wordpress e Owncloud.

4.5 Outras Aplicações

Até o fim deste trabalho, as aplicações Noosfero e Roundcube não foram finalizadas. Porém, é importante reportar o estado atual do que foi feito até o momento, com essas duas aplicações.

4.5.1 Noosfero

Noosfero (2016) é uma plataforma web para redes sociais e de economia social e solidária. Ela possui funcionalidades como blog; e-Porfolios; CMS; RSS; discussão temática; agenda de eventos; e inteligência coletiva para a economia solidária no mesmo sistema.

O estado atual da aplicação Noosfero no Shak encontra-se na fase do planejamento. No planejamento, já era de conhecimento do autor que o Noosfero não estava nos repositórios oficiais do Debian, mas que seria possível estar, desde que todas as dependências do Noosfero também estivessem no Debian.

O Noosfero já possui um repositório [6] com suas dependências empacotadas. Porém, algumas delas não se encontram nos repositórios oficiais do Debian, dificultando assim, que o pacote do Noosfero fique disponível nos repositórios oficiais do Debian.

Um exemplo é a biblioteca *ruby-selenium-webdriver*, que está no repositório do Noosfero, porém, ela não está disponível no repositório oficial do Debian. Outro problema que foi encontrado, é que algumas dependências do Noosfero estão em versões diferentes das dependências que estão no repositório Debian.

Para resolver esse problema, seria necessário empacotar todas as dependências do Noosfero que ainda não estão no Debian. Além disso, sincronizar as versões que já estão no Debian, com as versões que são utilizadas pelo Noosfero.

Na tabela 1 em anexo, encontra-se uma lista das dependências que estão no repositório do Noosfero, com suas versões, e as dependências que estão no Debian Jessie.

Como visto na tabela 1, grande parte das dependências do Noosfero encontram-se apenas nas versões *testing* e *unstable* no Debian. Para a continuar com a implantação automatizada, seria necessário discutir alguma solução com a comunidade do software.

Como sugestão, uma das soluções viáveis seria sincronizar as versões das dependências do Noosfero, com as versões que são disponibilizadas pelo Debian *testing*. Com isso, o Noosfero passa a ter boa parte de suas dependências suportadas pelo Debian. Além disso, também seria necessário empacotar as bibliotecas do Noosfero que o Debian não possui.

Porém, essa decisão fica a cargo da comunidade da aplicação, e com o Noosfero disponível nos repositórios oficiais do Debian, o Shak poderá dar suporte ao Noosfero, em trabalhos futuros.

4.5.2 Roundcube

Roundcube (2016) é uma solução de webmail gratuita, de código aberto. Possui uma interface funcional e personalizável. Com ele, é possível ler seus e-mails a partir de

[6] http://download.noosfero.org/debian/jessie/

um cliente web.

Para a ferramenta Roundcube, foram seguidos os mesmos passos feitos em 4.1 e 4.5.2.

As dependências identificadas foram:

- **Pacote Roundcube para o Debian**
- **Pacote Nginx para o Debian**
- **Pacote Postgresql para o Debian**
- **Arquivos de configuração:** Criação dos arquivos de configuração necessários para configurar o Roundcube, o servidor web Nginx e o banco de dados Postgresql.

Para facilitar a configuração de aplicações PHP, foi feito uma receita genérica que instala dependências básicas para qualquer aplicação PHP que for usada pelo Shak. Essa receita instala e habilita o PHP5, além de instalar o Nginx. A partir disto, as aplicações Owncloud e Wordpress também passaram a usar esta receita genérica, removendo duplicações de suas receitas.

Por fim, também foi necessário instalar o pacote roundcube-pgsql que é um módulo necessário para que Roundcube funcione com o banco de dados PostgreSQL.

Para executar essas instalações, é preciso criar uma receita no livro de receitas do Roundcube com o comando *rake cookbook*, da mesma forma com as outras aplicações.

O estado atual da aplicação Roundcube no Shak, encontra-se na fase de configuração. A pendência é a configuração do servidor de e-mail, que será utilizado pelo Roundcube. Por ser uma configuração pessoal do usuário, ou seja, ele deve inserir o seu serviço de e-mail preferido, essa configuração ainda não foi automatizada.

Como sugestão de trabalhos futuros, é necessário evoluir a receita do Roundcube para que possa receber os dados do serviço de e-mail, fornecidos pelo usuário durante sua instalação. Com isso, ao finalizar a instalação a aplicação já estará disponível ao usuário, com a conta de e-mail que foi informada.

Atualmente, a receita do Roduncube já automatiza a instalação da aplicação web. O trabalho atual está disponível no repositório oficial do Shak [7], na branch roundcube.

4.6 Uso de Protocolos Seguros

Na implantação das aplicações web, foram feitos dois procedimentos de segurança, o primeiro foi forçar as aplicações web a sempre utilizarem o protocolo *HTTPS* e a segunda

[7] https://gitlab.com/shak/shak/branches

foi forçar o servidor de e-mail a não permitir a conexão via protocolos sem criptografia. Para que isso fosse possível, foi necessário gerar um certificado *SSL*. Certificados *SSL* são necessários para que um determinado serviço opere com suporte a conexão segura por meio de criptografia. É de conhecimento do autor que a melhor forma é obter um certificado assinado por uma entidade certificadora registrada, porém inicialmente foram gerados apenas certificados autoassinados.

Para adicionar esse novo suporte ao Shak, foi necessário criar uma receita Chef, para que possa gerenciar os certificados *SSL*. Optou-se por utilizar a ferramenta Openssl para geração das chaves e certificados, Openssl é uma ferramenta de implementação do Transport Layer Security *(TLS)* e Secure Sockets Layer *(SSL)*, além de ser uma biblioteca de propósito geral de criptografia (OPENSSL, 2016).

Esse novo componente no Shak é composto do pacote openssl, além da criação de certificados autoassinados utilizando o Openssl. Os certificados são gerados para cada aplicação. O caminho dos arquivos dos certificados que são gerados foi */etc/ssl/certs/hostname.pem*, onde hostname é o endereço do servidor, e o caminho onde as chaves são geradas é */etc/ssl/private/hostname.key*.

Com as chaves geradas, bastou configurar as aplicações indicando os caminhos dos certificados e das chaves. Para forçar as aplicações web a utilizarem o *HTTP*, foi necessário fazer uma configuração específica no servidor web Nginx.

O código 4.10, mostra como foi feito.

Código 4.10 – Exemplo de arquivo de configuração do Nginx para aplicações web no shak

```
server {
  server_name  <%= @hostname %>;
  rewrite  ^https://\$server\_name\$request\_uri? permanent;
}

server {
  server_name  <%= @hostname %>;
  listen 443 ssl;
  ssl_certificate  /etc/ssl/certs/<%= @hostname %>.pem;
  ssl_certificate_key  /etc/ssl/private/<%= @hostname %>.key;
  access_log  /var/log/nginx/<%= @hostname %>.access.log;
  error_log  /var/log/nginx/<%= @hostname %>.error.log;
  include  /var/lib/Shak/etc/nginx/<%= @hostname %>/*.conf;
}
```

Com isso, todas as requisições foram forçadas a utilizar a porta 443 que é a porta *TCP* padrão para sites que utilizam *HTTP* e *SSL*. Assim, as aplicações utilizarão um

protocolo mais seguro, utilizando criptografia dos dados , utilizando o certificado autoassinado que foi gerado para o aplicação.

4.7 Protótipo da aplicação web

Por fim, outra contribuição neste trabalho foi a proposta de um protótipo para a interface web da aplicação. Como um dos objetivos do Shak é possuir uma interface web para que os usuários não precisem utilizar um terminal para realizar as ações, também foi feito um protótipo funcional, para ser aplicado no shak.

O protótipo foi feito na ferramenta Pingendo, que é um software livre utilizado para construir protótipos funcionais de aplicações web. Além de ter um protótipo funcional, a ferramenta Pingendo também gera o código *HTML* e *CSS*, que pode ser aproveitado para a construção do layout do Shak. A sugestão inicial foi feita utilizando *HTML* e *CSS* utilizando bootstrap 3, que é um framework para criar aplicações responsivas na web.

O protótipo foi pensado em três áreas para o usuário, a primeira delas é na Figura 5 e na Figura 6, onde o usuário pode ver as aplicações disponíveis, além da aplicação que possui a maior quantidade de instalações.

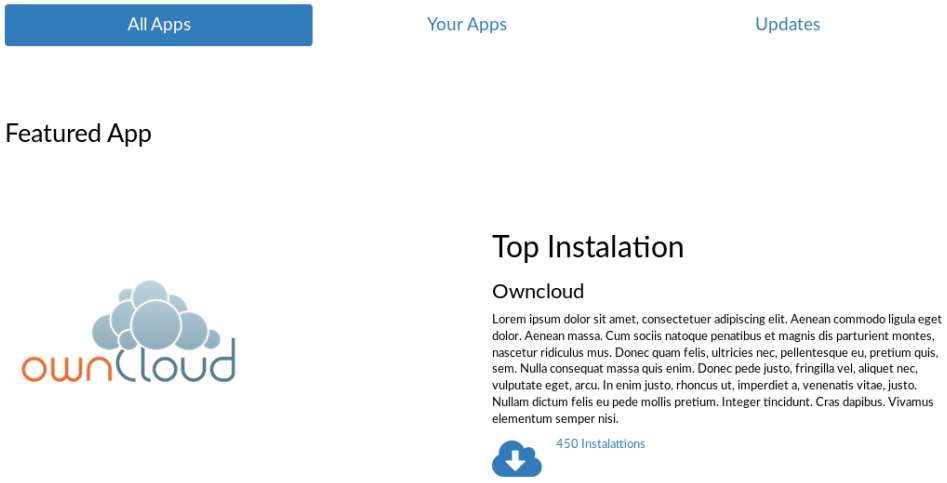

Figura 5 – Aplicação mais instalada pelo Shak

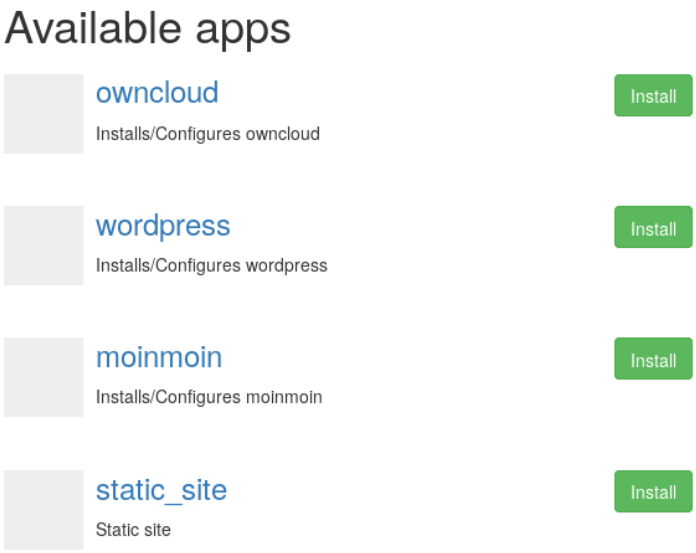

Figura 6 – aplicações disponíveis para instalação

Ao selecionar uma aplicação para instalar, o usuário deverá informar as mesmas informações que são necessárias na instalação via linha de comando. Na Figura 7 o usuário deve informar o endereço, o caminho da aplicação e confirmar.

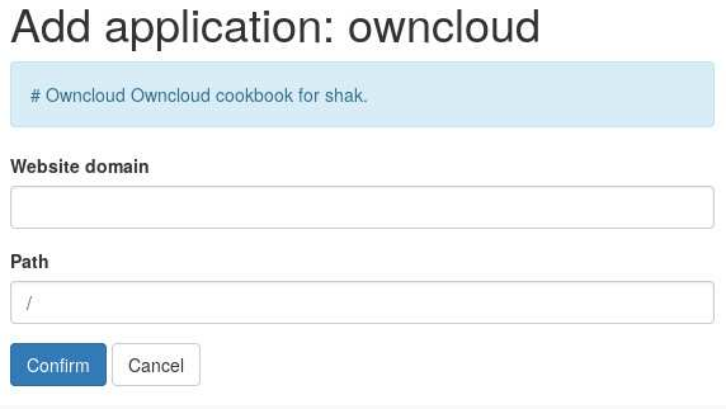

Figura 7 – Informações necessárias para implantar uma aplicação

Na Figura 8, é onde o usuário pode ver suas aplicações instaladas e editar suas informações.

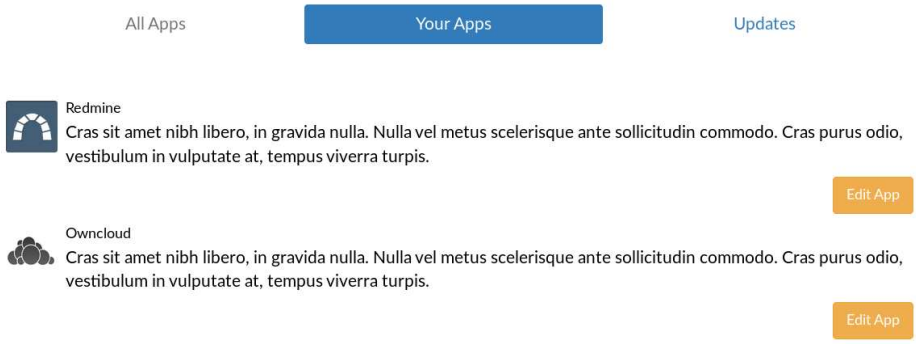

Figura 8 – Aplicações já instaladas pelo Shak

Por fim, na Figura 9, é onde o usuário pode atualizar suas aplicações.

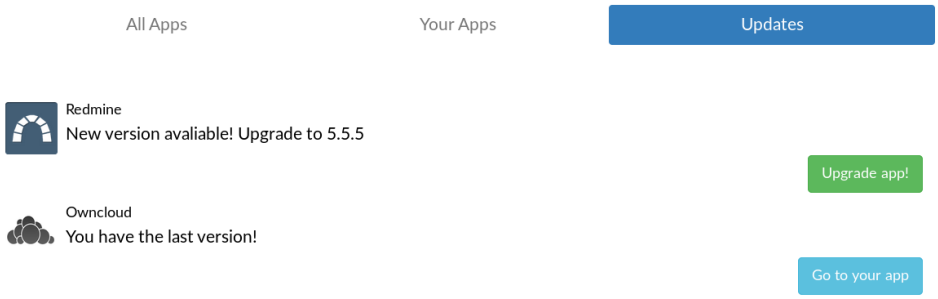

Figura 9 – Aplicações com atualizações disponíveis

5 Conclusão

A gerência de configuração de software é uma importante área da engenharia de software, visto que o software precisa estar implantado e disponível para uso, para que ofereça valor ao usuário final. Uma atividade importante nesse cenário é a implantação automatizada de software, que facilita a instalação de ferramentas, principalmente para usuários que não possuam conhecimento técnico para realizar instalações complexas.

Observou-se que outros trabalhos já utilizam os conceitos de implantação automatizada, porém nenhuma delas voltados a aplicações em sistemas Debian GNU/Linux, que possibilita algumas vantagens como suporte à segurança.

Também, foi visto que apenas a instalação dos pacotes não são suficientes, existem outras atividades a serem feitas e nem sempre o usuário detém o conhecimento necessário para configurar as aplicações. Com isso, dado o objetivo deste trabalho e as motivações, foi feito a evolução da ferramenta Shak, para que usuários possam instalar aplicações com "apenas um clique" num sistema Debian GNU/Linux.

No decorrer deste trabalho, alguns resultados foram alcançados. Primeiramente, estabeleceu-se o conhecimento sobre implantação automatizada de software e tecnologias que permitam tal implementação. Além disso, também foi discutido o conceito de *DevOps*,

Em relação a evolução da ferramenta, foram realizadas melhorias na ferramenta Shak provendo quatro novas aplicações, sendo elas: Owncloud, Wordpress, MoinMoin e servidor de e-mail, além de iniciar a implantação das aplicações Noosfero e Roundcube. Isso com todas as aplicações com suporte a múltiplas instâncias, e utilizando protocolos seguros.

Por fim, todos esses resultados são importantes para o objetivo deste trabalho, que vão desde como implantar aplicações web de forma automatizada em sistemas Debian/GNU Linux, até compreender como é o processo de implantação de um software. Além da evolução da ferramenta Shak, e até os cuidados a se tomar ao implantar uma aplicação.

5.1 Trabalhos Futuros

Algumas melhorias na ferramenta Shak não puderam ser implementadas no contexto deste trabalho, destaca-se o suporte a servidores autoassinados utilizando Let's encrypt, que trariam ganhos para os usuários finais, visto que os certificados autoassinados que utilizam a ferramenta Openssl não são reconhecidos automaticamente pelos navegadores.

Outra melhoria seria o suporte a criação de máquinas virtuais de forma automatizada, visto que é necessário que o usuário instale a ferramenta no servidor destino. Uma sugestão seria integrar com ferramentas que disponibilizam servidores na nuvem, como a DigitalOcean [1].

Também como trabalhos futuros, finalizar as aplicações Noosfero e Roundcube, além de suportar novas aplicações no Shak.

[1] www.digitalocean.com/

Referências

APACHE. *Apache Virtual Hosting Documentation*. [S.l.], 2016. Disponível em: <https://httpd.apache.org/docs/2.2/en/vhosts/>. Citado na página 24.

ARAÚJO, T. C. *AppRecommender: um recomendador de aplicativos GNU/Linux*. Tese (Doutorado) — Universidade de Sao Paulo, 2011. Citado 2 vezes nas páginas 23 e 24.

BARNES, R. et al. *Automatic certificate management environment (acme)*. [S.l.]: Tech. rep. IETF, 03/2016. url: https://datatracker. ietf. org/doc/draftietf-acme-acme, 2014. Citado na página 28.

BENDIX, L.; KOJO, T.; MAGNUSSON, J. Software configuration management issues with industrial opensourcing. In: *2011 IEEE Sixth International Conference on Global Software Engineering Workshop*. [S.l.: s.n.], 2011. p. 85–89. ISSN 2329-6305. Citado na página 17.

BITNAMI. *What is Bitnami*. [S.l.], 2016. Disponível em: <https://bitnami.com/learn_more>. Citado na página 30.

CARZANIGA, A. et al. A characterization framework for software deployment technologies. Department of Computer Science,University of Colorado,Boulder,CO, 1998. Citado 2 vezes nas páginas 18 e 22.

CASTELLS, M. *A Galáxia Internet: reflexões sobre a Internet, negócios e a sociedade*. [S.l.]: Zahar, 2003. Citado na página 14.

DEBIAN. *Debian Policy Manual*. [S.l.], 2016. Disponível em: <https://www.debian.org/doc/debian-policy/>. Citado na página 24.

DOVECOT. *Dovecot, Secure IMAP server*. [S.l.], 2016. Disponível em: <http://www.dovecot.org/>. Citado na página 48.

GILES, D. *Psychology of the media*. [S.l.]: Palgrave Macmillan, 2010. Citado na página 14.

HTTERMANN, M. *DevOps for developers*. [S.l.]: Apress, 2012. Citado na página 20.

HUMBLE, J.; FARLEY, D. Continuous delivery: Reliable software releases through build, test, and deployment automation. In: ADDISON-WESLEY (Ed.). [S.l.: s.n.], 2010. Citado na página 20.

HUMMER, W. et al. Testing idempotence for infrastructure as code. In: _____. *Middleware 2013: ACM/IFIP/USENIX 14th International Middleware Conference, Beijing, China, December 9-13, 2013, Proceedings*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013. p. 368–388. ISBN 978-3-642-45065-5. Disponível em: <http://dx.doi.org/10.1007/978-3-642-45065-5_19>. Citado na página 22.

IEEE. Ieee standard for configuration management in systems and software engineering - redline. *IEEE Std 828-2012 (Revision of IEEE Std 828-2005) - Redline*, p. 1–126, March 2012. Citado na página 17.

JUJU. *What is juju?* [S.l.], 2016. Disponível em: <https://jujucharms.com/docs/stable/about-juju>. Citado na página 30.

KUROSE, J. F. et al. *Redes de computadores ea Internet: Uma abordagem top-down.* [S.l.]: Pearson, 2010. Citado 3 vezes nas páginas 25, 26 e 27.

LEITE, L. A. F. Implantação automatizada de composições de serviços web de grande escala. In: . USP, São Paulo: [s.n.], 2014. Citado 4 vezes nas páginas 18, 22, 29 e 31.

MANTYLA, M.; VANHANEN, J. Software deployment activities and challenges - a case study of four software product companies. In: *Software Maintenance and Reengineering (CSMR), 2011 15th European Conference on.* [S.l.: s.n.], 2011. p. 131–140. ISSN 1534-5351. Citado 2 vezes nas páginas 18 e 22.

MOINMOIN. *User Manual.* [S.l.], 2016. Disponível em: <https://moinmo.in/>. Citado na página 51.

NGINX. *Nginx Documentation.* [S.l.], 2016. Disponível em: <https://www.nginx.com/resources/wiki/>. Citado na página 34.

NOOSFERO. *About Noosfero.* [S.l.], 2016. Disponível em: <http://noosfero.org/bin/view/Site/About>. Citado na página 53.

OMG. Deployment and configuration of component-based distributed applications specification. 2006. Disponível em: <http://www.omg.org/spec/DEPL/4.0/>. Citado 2 vezes nas páginas 18 e 19.

OPENSSL. *OpenSSL Documentation.* [S.l.], 2016. Disponível em: <https://www.openssl.org/>. Citado na página 55.

OWNCLOUD. *User Manual.* [S.l.], 2016. Disponível em: <https://doc.owncloud.org/server/9.0/user_manual/>. Citado na página 45.

RAHMAN, A. et al. Synthesizing continuous deployment practices used in software development. In: *Agile Conference (AGILE), 2015.* [S.l.: s.n.], 2015. p. 1–10. Citado na página 18.

ROUNDCUBE. *About the Roundcube webmail project.* [S.l.], 2016. Disponível em: <https://roundcube.net/about/>. Citado na página 53.

RUBYGEMS. *What is a gem ?* [S.l.], 2016. Disponível em: <http://guides.rubygems.org/what-is-a-gem/>. Citado na página 34.

SANDSTORM.IO. *What is Sandstorm?* [S.l.], 2016. Disponível em: <https://docs.sandstorm.io/en/latest/>. Citado na página 30.

SHARMA, T.; FRAGKOULIS, M.; SPINELLIS, D. Does your configuration code smell? In: *Proceedings of the 13th International Conference on Mining Software Repositories.* New York, NY, USA: ACM, 2016. (MSR '16), p. 189–200. ISBN 978-1-4503-4186-8. Disponível em: <http://doi.acm.org/10.1145/2901739.2901761>. Citado na página 21.

SPAMASSASIN, A. *Welcome to SpamAssasin.* [S.l.], 2016. Disponível em: <http://spamassassin.apache.org/>. Citado na página 48.

SPINELLIS, D. Don't install software by hand. *IEEE Software*, IEEE Computer Society, Los Alamitos, CA, USA, v. 29, n. 4, p. 86–87, 2012. ISSN 0740-7459. Citado 2 vezes nas páginas 14 e 22.

SPINELLIS, D. Don't install software by hand. *Software, IEEE*, v. 29, n. 4, p. 86–87, July 2012. ISSN 0740-7459. Citado 3 vezes nas páginas 15, 20 e 23.

TERCEIRO, A. Shak: uma proposta para descentralização dos serviços de internet. Artigo não publicado, em anexo, 2016. Citado 3 vezes nas páginas 14, 15 e 32.

TERRA, C. F. *Comunicação corporativa digital: o futuro das relações públicas na rede.* Tese (Doutorado) — Universidade de São Paulo, 2006. Citado na página 15.

VAZ, P. As esperanças democráticas e a evolução da internet. *Revista FAMECOS: midia, cultura e tecnologia*, v. 1, n. 24, 2006. ISSN 1980-3729. Disponível em: <http://200.144.189.42/ojs/index.php/famecos/article/view/391/320>. Citado na página 14.

VIRMANI, M. Understanding devops bridging the gap from continuous integration to continuous delivery. In: *Innovative Computing Technology (INTECH), 2015 Fifth International Conference on*. [S.l.: s.n.], 2015. p. 78–82. Citado 2 vezes nas páginas 20 e 21.

WORDPRESS. *Documentação Wordpress*. [S.l.], 2016. Disponível em: <http://codex.wordpress.org/pt-br:P%C3%A1gina_Inicial>. Citado na página 42.

Anexos

ANEXO A – Ferramenta Shak

A.1 Uma proposta para descentralização dos serviços de internet

Desde a sua origem, a internet foi projetada para ser uma rede descentralizada. A grande maioria dos protocolos utilizados na rede não dependem de pontos centrais para funcionar. De fato, é de conhecimento geral que, por exemplo, usuários com contas de e-mails em provedores diferentes podem trocar mensagens entre si sem qualquer problema.

O mesmo vale para outras aplicações, em especial para aquelas que utilização a web: um blog, rede social ou site de notícias pode estar hospedado em qualquer provedor, e não existe nenhuma razão técnica pela qual usuários que utilizam outros provedores de acesso à rede não possam acessá-los.

Apesar disso, existe atualmente uma tendência de centralização da internet. A imensa maioria dos usuários individuais utilizam contas de e-mail em algum poucos grandes provedores. O mesmo acontece com utilitários como armazenamento de documentos/arquivos/fotos, agenda, etc.

A.1.1 Consequências da centralização da internet

Essa excessiva centralização traz os seguintes problemas para a sociedade. Entre eles, podemos citar:

Riscos à privacidade : A quantidade de informações pessoais fornecidas a serviços online centralizados por uma quantidade imensa de pessoas faz com que estes serviços representem um risco muito grande à privacidade destas pessoas. Diversos destes provedores mencionam explicitamente em seus termos de serviço que informações pessoais serão usadas para fins de marketing como exibição de propaganda, e não há forma de saber se são feitos outros usos dessas informações. Mesmo com a premissa de que os provedores de serviços são confiáveis, eventuais problemas de segurança em seus sistemas podem deixar muitas informações pessoais de muitos usuários à mercê de atacantes mal-intencionados (crackers, criminosos, que são popular e incorretamente denominados de hackers). Se essas informações estivessem dispersas em diversos provedores de serviços, uma eventual invasão afetaria uma quantidade muito menor de pessoas.

Subordinação dos usuários aos provedores : Os grandes provedores de serviços acumulam uma quantidade excessiva de poder sobre a sociedade. Uma vez que a imensa

maioria dos usuários da internet usa os seus serviços, estas empresas têm a capacidade de unilateralmente definir o que é e o que não é possível com base puramente em seus interesses comerciais.

Pontos centrais de falha : Provedores centralizados também se tornam pontos centrais de falha, e uma eventual indisponibilidade de seus serviços podem consequências econômicas gravíssimas. Na medida em que mais e mais pessoas e negócios dependem do seu bom andamento de suas atividades, os riscos à economia mundial se tornam cada vez maiores.

Mesmo que toda a sociedade tivesse plena consciência destes problemas causados pela centralização dos serviços digitais, a maioria dos usuários ainda continuaria utilizando-os, devido a um fator que acaba sendo decisivo na adoção de provedores de serviço: a conveniência. Qualquer solução para esta questão precisa ser no mínimo tão conveniente quanto as alternativas centralizadas existentes na atualidade.

Este projeto visa fornecer uma alternativa à centralização da internet através de uma solução que elimina barreira técnica para que usuários interessados possam ter servidores próprios com aplicações sem a necessidade de conhecimento técnico especializado. Esta solução está também norteada na maior conveniência possível, sem sacrifício da segurança e proteção à privacidade.

A.1.2 Causas da centralização da internet

Nos últimos anos a demanda por soluções na internet vem crescendo de forma impressionante. Por um lado, o acesso à internet aumenta mais e mais, tanto em função do aumento da disponibilidade de conexões de banda larga, quanto a explosão na utilização de dispositivos móveis.

Em paralelo, o tecnologia para desenvolvimento de soluções na internet avança a passos largos, tanto no que diz respeito a possibilidades, quanto no que diz respeito à complexidade necessária para tornar novas funcionalidades possíveis. Serviços de alta sofisticação implicam numa alta complexidade de desenvolvimento e manutenção.

Esta complexidade faz com que a implantação e manutenção de serviços exija conhecimento técnico especializado. Para organizações, isto implica num maior custo para manter sistemas e aplicações. Para indivíduos, isto quase sempre impossibilita a manutenção de serviços próprios e leva à procura por serviços "gratuitos". Estes serviços podem ser gratuitos do ponto de vista estritamente financeiro e imediato, mas os termos com os quais os usuários concordam, muitas vezes inadvertidamente, ao usar estes serviços, permitem aos provedores fazerem uso de informações pessoais dos usuários que podem vir a ter um alto custo tanto para o próprio indivíduo quanto para a sociedade em geral.

A.1.3 Software livre como uma alternativa para a descentralização

O conceito de Software Livre foi cunhado na década de 1980. Um Software Livre fornece aos seus usuários 4 liberdades fundamentais:

1. Executar o software para qualquer finalidade;
2. Estudar o funcionamento do software e adaptá-lo às necessidades do usuário;
3. Redistribuir cópias do software;
4. Distribuir cópias de versões modificadas do software.

Desde lá, o software livre se tornou parte importante da infraestrutura da internet. Mais da metade dos servidores da internet utilizam servidores HTTP livres. Os dos principais navegadores web (Firefox e Google Chrome) são softwares livres, assim como é livre o sistema operacional Android, utilizado na maioria esmagadora dos smartphones disponíveis no mercado.

Sistemas operacionais livres, como o Debian, permitem que usuários com conhecimento técnico para tal possam hospedar os seus próprios serviços de internet, utilizando exatamente a mesma tecnologia que é empregada nos maiores sites profissionais. Todo tipo de serviço pode ser hospedado com sistemas operacionais livres, mas esta realidade ainda está longe do usuário final.

O usuários final tem a opção de ou contratar um técnico qualificado para fazê-lo, no entanto esta é uma barreira que é muito mais difícil de transpor, por não ser conveniente em comparação com a alternativa: simplesmente utilizar serviços "gratuitos"de grandes players da internet, em troca de acesso às suas informações pessoais, cedidas de forma consciente ou não, e com todas as consequências negativas já mencionadas anteriormente.

Um outro impeditivo para que usuários possam hospedar os seus próprios serviços de internet é o custo de manutenção do servidor em si, e de sua conectividade. Nos últimos anos, no entanto, este fator vem perdendo relevância em função de dois fenômenos. Por um lado, o barateamento do acesso a servidores virtuais privados decorrente os avanços da "computação em nuvem". Por outro lado, a disponibilidade de servidores físicos de dimensão reduzida e baixo consumo de energia, em conjunto com o barateamento de conexões de banda larga à internet.

As inovações na computação em nuvem (especialmente em virtualização e armazenamento) fazem com que atualmente seja possível alugar servidores virtuais privados (VPS, da singla em inglês "virtual private server") em provedores comerciais por valores tão baixos quanto 5 dólares por mês. Mesmo que o valor não pareça ser tão baixo, a depender do serviço um único VPS pode servir uma quantidade razoável de usuários, possibilitando que um grupo de usuários possam dividir este custo.

Por outro lado, servidores virtuais em data centers comerciais também deixam uma pula atrás da orelha dos usuários mais preocupados com a sua privacidade. A segurança e a integridade dos dados hospedados em um VPS estão sujeitos à confiabilidade do provedor: sendo o dono do hardware onde o VPS está hospedado, o provedor teria em tese acesso total a tudo que está hospedado nele mesmo num VPS individual.

Em situações onde nem um VPS oferece as garantias de privacidade necessárias, usuários têm ainda a alternativa de hospedar os serviços em hardware próprio em casa, utilizando a conectividade oferecida por uma conexão banda larga. Existem hoje no mercado uma grande quantidade de servidores de dimensão reduzida (e.g. do tamanho de um modem ADSL, ou menores), que possuem um consumo de energia mínimo. Projetos como Freedombox apostam em servidores físicos e privados hospedados em domicílios, em conjunto com a utilização de criptografia, como uma saída para privacidade de fato nas comunicações online.

A.2 A Proposta

Este projeto visa a desenvolver uma plataforma chamada shak cujo objetivo é facilitar ao máximo que usuários sem conhecimento técnico possam ter os seus próprios serviços de internet, garantindo sua privacidade e segurança. Esta plataforma está concebida de acordo com os seguintes princípios:

A.2.1 Base: pacotes Debian

A plataforma shak é baseada no sistema de pacotes do Debian, uma das distribuições GNU/Linux mais consolidadas e com maior comunidade internacional.

Os pacotes do Debian fornecem atualizações consistentes de correção e de segurança, e são utilizados por uma grande quantidade de usuários que reportam problemas a serem resolvidos. Porém, para garantirem uma flexibilidade e suportar diferentes casos de uso, normalmente os pacotes não realizam a configuração completa do ambiente, deixando os detalhes finais para o administrador do sistema.

Além disso, normalmente aplicações são compostas de diversos pacotes, e requerem configurações que dizem respeitos a mais de um deles e que por consequência não poderiam ser feitas automaticamente de uma forma sustentável em nenhum dos pacotes.

A.2.2 Nova abstração: aplicação

Para resolver as questões que não podem ser resolvidas no nível de pacotes, o shak introduz uma nova abstração: a aplicação. Uma aplicação geralmente contém mais de

um pacote, e é responsável por toda a configuração, inclusive configurações em múltiplos pacotes.

Concretamente, uma vez que o usuário seleciona uma determinada aplicação para ser instalada, os pacotes necessários são instalados e as configurações necessárias são feitas de forma automática, fornecendo de fato um "instalador de um clique"para aplicações suportadas.

Todas as configurações são feitas de forma consistente utilizando um framework comum, buscando o maior nível de segurança possível, e assumindo premissas que não poderiam ser assumidas no nível da abstração pacote.

A.2.3 Diferencial

para outras soluções existentes

Existem outras soluções disponíveis para "instaladores de um clique"no mercado, mas o shak se diferencia delas nos seguintes aspectos:

1. O shak pode ser implantando em um VPS na nuvem ou em servidores físicos próprios, a critério do usuário.

2. O shak reutiliza o trabalho dos mantenedores Debian, que fornecem pacotes de alta qualidade.

3. Por outro lado, apenas software que esteja empacotado no Debian estará disponível, mas isso pode ser visto de forma positiva também: o shak fornecerá um incentivo para que ainda mais software esteja disponível no Debian, o que é um benefício coletivo para o ecossistema do software livre, e para os seus usuários.

O shak reutiliza a infraestrutura do projeto Debian. O Debian possui espelhos do seu repositório espalhados por todo o mundo, fornecendo links para downloads mais rápidos aonde quer que o usuário esteja. Atualizações de defeitos e correções de segurança realizadas no Debian estarão automaticamente disponíveis para usuários do shak de forma transparente. Além disso, o próprio shak estará disponível como um pacote dentro do próprio repositório oficial do Debian.

ANEXO B – Dependências do Noosfero

Tabela 1 – Dependências do Noosfero nos repositórios Debian e Noosfero

Repositório do Noosfero Debian Jessie	Repositório oficial do Debian Jessie
bundler (1.10.6)	bundler (1.7.4-1)
Noosfero Apache (1.5)	inexistente
Noosfero Chat (1.5)	Inexistente
Rails (4.2.5)	Rails (4.1.8-1+deb8u2)
ruby-actionmailer (4.2.5)	ruby-actionmailer (4.1.8)
ruby-actionpack-page-caching (1.0.2)	Apenas na versão testing/unstable (1.2.0)
ruby-actionpack (4.2.5)	ruby-actionpack (4.1.8)
ruby-actionview (4.2.5)	ruby-actionview (4.1.8)
ruby-activejob (4.2.5)	Apenas na versão testing/unstable (4.2.6)
ruby-activemodel (4.2.5)	ruby-activemodel (4.1.8)
ruby-activerecord-deprecated-finders (1.0.4)	ruby-activerecord-deprecated-finders (1.0.3)
ruby-activerecord-session-store (0.1.1)	Apenas na versão testing/unstable (1.0.0)
ruby-activerecord (4.2.4)	ruby-activerecord (4.1.8)
ruby-activesupport (4.2.5.1)	ruby-activesupport (4.1.8)
ruby-acts-as-taggable-on (3.5.0)	ruby-acts-as-taggable-on (2.4.1)
ruby-api-pagination (4.2.0)	Apenas na versão testing/unstable (4.2.0)
ruby-arel (6.0.3)	ruby-arel (5.0.1)
ruby-axiom-types (0.1.1)	ruby-axiom-types (0.1.1)
ruby-binding-of-caller (0.7.2)	Apenas na versão testing/unstable (0.7.2)
ruby-byebug (5.0.0)	Apenas na versão testing/unstable (5.0.0)
ruby-chronic (0.10.2)	ruby-chronic (0.10.2)
ruby-coffee-rails (4.1.0)	ruby-coffee-rails (4.0.1)
ruby-columnize (0.9.0)	ruby-columnize (0.8.9)
ruby-cucumber-rails (1.4.2)	Apenas na versão testing/unstable (1.4.2)
ruby-dalli (2.7.4)	Apenas na versão testing/unstable (2.7.4)
ruby-debug-inspector (0.0.2)	Apenas na versão testing/unstable (0.0.2)
ruby-delayed-job-active-record (4.0.3)	Apenas na versão testing/unstable (4.0.3)
ruby-delayed-job (4.0.6)	Apenas na versão testing/unstable (4.0.6)
ruby-doorkeeper (2.2.1)	Apenas na versão testing/unstable (3.1.0)
ruby-eita-jrails (0.10.0)	Inexistente no Debian
ruby-fast-gettext (0.9.2)	ruby-fast-gettext (0.9.0)
ruby-globalid (0.3.6)	Apenas na versão testing/unstable (0.3.6)

ruby-grape-entity (0.4.8)	Apenas na versão testing/unstable (0.5.1)
ruby-grape-logging (1.1.2)	Apenas na versão testing/unstable (1.3.0)
ruby-grape (0.12.0)	Apenas na versão testing/unstable (0.16.0)
ruby-hashie (3.4.2)	ruby-hashie (2.0.5)
ruby-i18n (0.7.0)	ruby-i18n (0.6.9)
ruby-ice-nine (0.11.1)	ruby-ice-nine (0.11.0)
ruby-launchy-shim (2.3.0)	Apenas na versão testing/unstable (2.3.0)
ruby-liquid (3.0.4)	ruby-liquid (2.6.1)
ruby-loofah (2.0.3)	Apenas na versão testing/unstable (2.0.3)
ruby-minitest-reporters (1.0.19)	Apenas na versão testing/unstable (1.0.19)
ruby-molinillo (0.2.3)	Apenas na versão testing/unstable (0.5.0)
ruby-oauth2 (1.0.0)	ruby-oauth2 (0.9.3)
ruby-omniauth-google-oauth2 (0.2.6)	ruby-omniauth-google-oauth2 (0.2.4)
ruby-omniauth-oauth2 (1.3.1)	ruby-omniauth-oauth2 (1.1.2)
ruby-omniauth (1.3.1)	ruby-omniauth (1.2.1)
ruby-pothoven-attachment-fu (3.2.16)	Inexistente no Debian
ruby-progressbar (1.4.2)	ruby-progressbar (0.21.0)
ruby-protected-attributes (1.1.3)	ruby-protected-attributes (1.0.8)
ruby-rack-accept (0.4.5)	ruby-rack-accept (0.4.5)
ruby-rack-contrib (1.3.0)	Apenas na versão testing/unstable (1.3.0)
ruby-rack-mount (0.8.3)	Apenas na versão testing/unstable (0.8.3)
ruby-rack (1.6.4)	ruby-rack (1.8.3)
ruby-rails-deprecated-sanitizer (1.0.3)	Apenas na versão testing/unstable (1.0.3)
ruby-rails-dom-testing (1.0.6)	Apenas na versão testing/unstable (1.0.6)
ruby-rails-html-sanitizer (1.0.2)	Apenas na versão testing/unstable (1.0.3)
ruby-rails-observers (0.1.2)	ruby-rails-observers (0.1.1)
ruby-railties (4.2.4)	ruby-railties (4.2.6)
ruby-rakismet (1.5.2)	Inexistente no Debian
ruby-rspec-rails (3.3.3)	Apenas na versão testing/unstable (2.14.2)
ruby-selenium-webdriver (2.50.0)	Inexistente no Debian
ruby-spy (0.4.2)	Apenas na versão testing/unstable (0.4.3)
ruby-thread-order (1.1.0)	Apenas na versão testing/unstable (1.1.0)
ruby-thread-safe (0.3.5)	ruby-thread-safe (0.3.1)
ruby-turbolinks (2.5.3)	ruby-turbolinks (2.2.2)
ruby-web-console (2.2.1)	Apenas na versão testing/unstable (2.2.1)
ruby-whenever (0.9.4)	Apenas na versão testing/unstable (0.9.4)
ruby-will-paginate (3.0.7)	ruby-will-paginate (3.0.5)
ruby-xpath (2.0.0)	ruby-xpath (2.0.0)