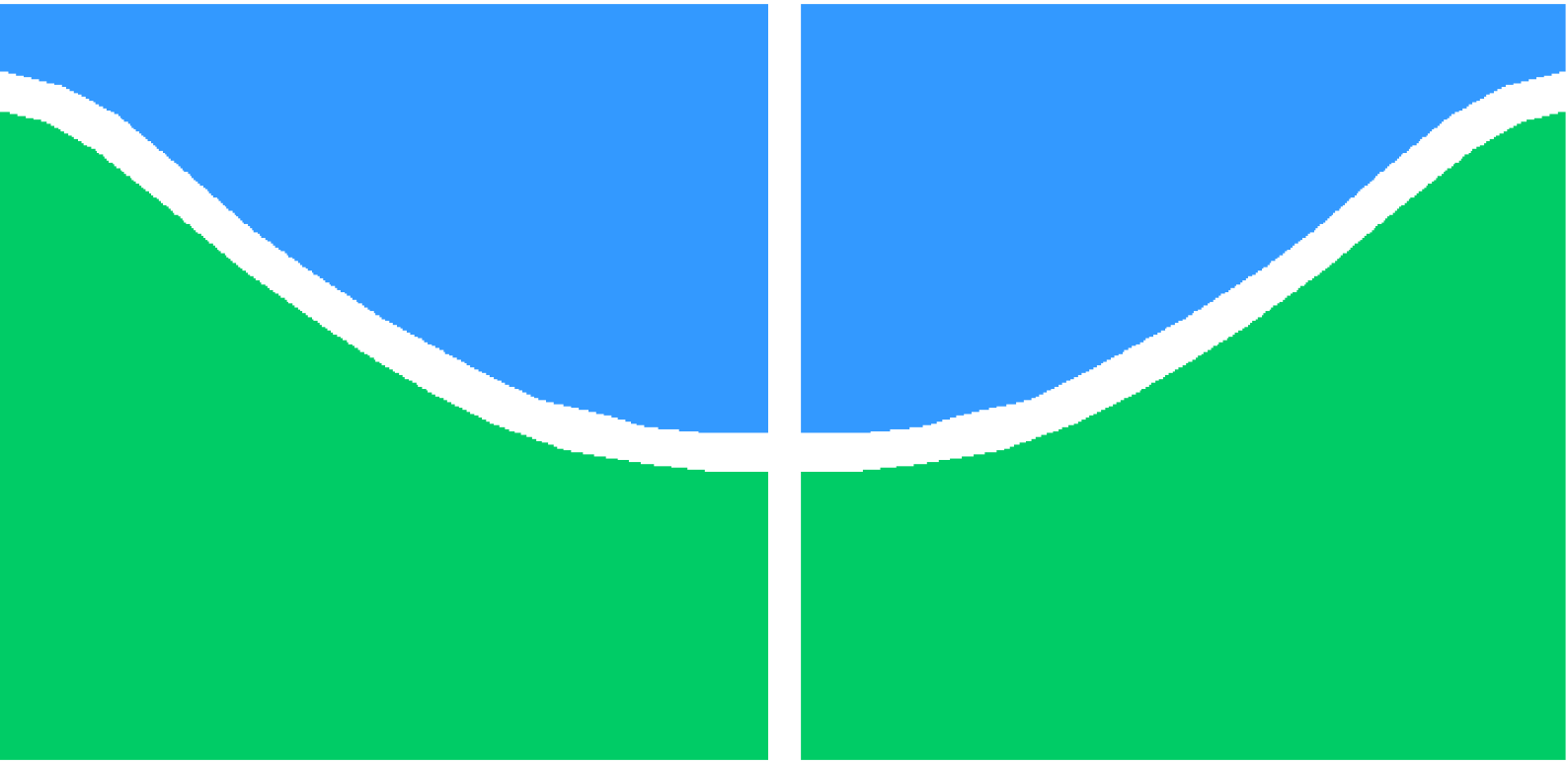

Faculdade do Gama, Universidade de Brasília

TCC-1 -101141

Extração de Configuração de Ambientes para Reprodução Seguindo Padrões de Receita Chef

Autores: Paulo, Lucas

Orientador: Renato Coral

Brasília, DF

2016

Lucas, Paulo

Extração de Configuração de Ambientes para Reprodução Seguindo Padrões de Receita Chef

Trabalho de Conclusão de Curso de Engenharia de Software da Universidade de Brasilia.

Faculdade do Gama, Universidade de Brasília

Orientador: Renato Coral

Brasília, DF
2016

Paulo Tada
Extração de Configuração de Ambientes para Reprodução Seguindo Padrões de Receita Chef / Paulo Tada [et al.]. – 2016.
53 p. : il. (algumas color.) ; 30 cm.

Orientador: Renato Coral

TCC - 1 – Faculdade do Gama, Universidade de Brasília, Brasília, DF, 2016.

1. Devops. 2. Chef. 3. Cupper. I. Severo, Lucas. II. Tada, Paulo. III. Renato Coral. IV. Universidade de Brasília. V. Faculdade UnB Gama. VI. Extração de Configuração de Ambientes para Reprodução Seguindo Padrões de Receita Chef.

Lista de ilustrações

Lista de tabelas

Sumário

Resumo

Atualmente há uma grande procura do mercado pela implementação de DevOps em suas organizações como parte da estratégia de vantagem competitiva. A cultura DevOps traz consigo conceitos e práticas que visam aumentar a quantidade de entrega de *software* em um curto período de tempo com maior qualidade e segurança. A ferramenta Chef, uma das mais populares entre as ferramentas de automação dentro do contexto de DevOps, é um facilitador para a orquestração de infraestrutura, aplicando o conceito de infraestrutura como código. Este trabalho propões a implementação de uma ferramenta para realizar a engenharia reversa deste conceito, construíndo *scrips* no padrão da ferramenta Chef a partir de um ambiente previamente configurado para um objetivo.

Resumo

Currently there is a large market demand for the implementation of DevOps in their organizations as part of the competitive advantage strategy. The DevOps culture brings concepts and practices aimed at increasing the amount of software delivery in a short period of time with higher quality and safety. The Chef tool, one of the most popular automation tools within the context of DevOps is a facilitator for infrastructure orchestration, applying the concept of infrastructure as code. This paper proposes the implementation of a tool to reverse engineering this concept, constructing scrips in the pattern of Chef tool from a preconfigured environment for a goal.

1 Introdução

Neste capítulo serão abordados os itens relacionados ao contexto geral de aplicação e pesquisa deste Trabalho de Conclusão de Curso. As seções estão dispostas em:

- **Contexto**: mostra o contexto em que este trabalho pretende atuar;
- **Objetivos**: mostra o objetivo geral e os específicos;
- **Organização**: mostra a organização dos capítulos deste trabalho.

1.1 Contexto

O mercado de desenvolvimento de software, atualmente, pressiona as organizações a buscarem por modelos no qual ofereçam uma constante entrega de produto com um intervalo cada vez menor de tempo. Como parte deste cenário, tem-se o tradicional problema entre a equipe de desenvolvimento e a equipe de operação (HUMMER et al., 2013).

Enquanto a equipe de desenvolvimento tende a disponibilizar todas novas alterações, solicitadas pelo cliente ou providas pela empresa, para a validação do cliente o mais depressa possível, a equipe de operação tende a manter o sistema estável, o que significa minimizar a implantação de grandes mudanças. A lacuna desse processo é agravada pelos diferentes objetivos de cada equipe (HUTTERMANN, 2012).

A adoção de *DevOps (Development and Operation)* vem sido feita para amenizar esse problema. *DevOps* consiste em uma série de práticas, ferramentas ou mesmo cultura organizacional com o objetivo de diminuir o tempo de entrega e implantação de um software. Juntamente com esse modelo, tem-se os conceitos de automação e infraestrutura como código. Ambos estão fortemente ligados, sendo facilitadores para o desenvolvedor conhecer as regras de implantação de sua aplicação e para o operador documentar e configurar um ambiente para um estado específico definido pelo código de implantação (HUMMER et al., 2013).

Neste contexto, surgiram ferramentas de suporte, como Chef (CHEF, 2008) e Puppet (PUPPET, 2016b), que abstraem os passos de execução de configuração e implantação em forma de código. A chave dessas propostas está em torno da capacidade de *convergence* do ambiente, ou seja, a execução de uma sequência de passos pré-definidos sempre leva a um estado previsível do objeto (HUMMER et al., 2013). Pode-se citar alguns dos principais benefícios (VARSILIEV, 2014):

- **Escalabilidade**: aumenta a quantidade de serviços em uma infraestrutura utilizando *environments* (variáveis do ambientes),*roles* (conjunto de configurações que representa o papel do sistema) e *nodes* (máquina física, virtual, container, etc);
- **Reuso e Replicação**: possibilita configurar a mesma aplicação em um novo *node*, ou seja, conseguir popular as mesmas configurações pela infraestrutura em pouco tempo;
- **Documetação**: um *script*(ou receita) em Chef contém todas as instruções necessárias para configuração do ambiente;

Neste cenário, este trabalho visa uma oportunidade de melhoria com relação a criação dos *scripts* nos padrões da ferramenta de automação Chef, realizando a engenharia reversa das configurações de um ambiente para a criação de *scripts* Chef a fim de facilitar na replicação, documentação e migração da infraestrutura.

1.2 Objetivos

1.2.1 Objetivo Geral

O objetivo deste trabalho é desenvolver uma ferramenta, denominada Cupper, que extrai, de maneira automatizada, informações de uma máquina para realizar a construção de *scrips* de automação de infraestrutura. Os *scripts* serão feitos nos padrões da ferramenta Chef e contém os comandos necessários para configurar e replicar um ambiente.

1.2.2 Objetivos Específicos

Os seguintes passos foram levantados para serem seguindos neste trabalho em conformidade ao proposto no objetivo geral 1.2.1.

1. Identificar as principais e mais relevantes configurações de um sistema alvo;
2. Identificar os componentes estruturais de configuração no padrão da ferramenta Chef;
3. Identificar e definir uma estrutura mínima de configurações Chef;
4. Mapear quais e onde serão armazenados nos *scripts* os diferentes tipos de configurações de um sistema alvo;
5. Utilizar boas práticas da Engenharia de Software (ex.: testes automatizados, modularização e outras) para tornar a ferramenta mais manutenível, extensível e reutilizável;

1.3 Organização do Documento

Este documento está organizado da seguinte maneira:

1. **Fundamentação Teórica**: Caracterização e conceitualização dos objetos de pesquisa;
2. **Metodologia**: Visão da metodologia de pesquisa e implementação do trabalho;
3. **Desenvolvimento**: Visão de implementação e funcionamento da ferramenta proposta;
4. **Resultados Parciais**: Resultados de pesquisas da primeira parte do trabalho.

2 Fundamentação Teórica

Neste capítulo serão apresentados as bases teóricas levantadas para este trabalho. O fluxo de conteúdo está organizado de forma a delimitar o domínio de atuação do trabalho, partindo da visão macro, DevOps, até a visão micro, ferramentas de extração de configuração, que é o foco deste trabalho. As seções estão dispóstas em:

- **DevOps**: as definições e conceitos relacionados a DevOps;
- **Métodos Ágeis e DevOps**: conceitos de métodos ágeits e as suas relações com DevOps;
- **Automação**: o conceito de automação dentro da cultura DevOps e as suas vantagens;
- **Extração de Configuração**: formas de extração das configurações de ambiente;

2.1 DevOps

O termo DevOps tem sido usado com frequência em diversas esferas do desenvolvimento de software da atualidade, mas por ser um conceito recente (2008), muita cunfusão ainda é gerada ao tentar definir e trabalhar com DevOps (BERTRAM, 2015). ”A palavra DevOps vem de duas palavras em inglês, *development* e *operations* (desenvolvimento e operações) e de maneira geral é a cultura, movimento ou conjunto de práticas que incentiva a comunicação, a colaboração e a integração de desenvolvedores de software e outros profissionais de TI. Além das práticas também engloba ferramentas e técnicas que que automatizam o processo de entrega de software e as mudanças de infraestrutura (LOUKIDES, 2012) (ERICH; AMRIT; DANEVA, 2014).

Muitas vezes o termo é confundido com uma nova responsabilidade, ou cargo dentro de uma empresa que desenvolve software, e por mais que seja possível ter profissionais que tenham proficiência nas ferramentas relacionadas a DevOps, o ideal, como dito anteriormente, é ter uma melhor comunicação, colaboração e integração entre os times já existentes. As ferramentas relacionadas à DevOps facilitam esses aspectos, mas o diferencial é a mudança no processo de desenvolvimento para absorver essas melhorias (BERTRAM, 2015).

2.2 Métodos Ágeis e DevOps

A metodologia ágil surgiu como uma resposta às maneiras tradicionais de desenvolvimento de software considerando uma nova abordagem com relação a práticas, organização, documentação e foco no desenvolvimento. (AGILEORG, 2015)

Os métodos ágeis são formas de sustentação da filosofia ágil proposta no manifesto ágil. As duas mais populares são o *Scrum* e o *Extreme Programming*. Nelas são definidas práticas que eram comumente utilizadas em outros contextos, mas foram adaptadas para se adequarem a filosofia ágil (SHORE et al., 2007).

Diferentemente da metodologia de Cascata, por exemplo, o *Scrum* implementa uma abordagem iterativa e incremental, podendo assim desenvolver incrementos de maior valor pro cliente mais cedo, e assim tendo *feedback* para correções mais frequentes. A figura 1 mostra um exemplo dessas Iterações. (SCRUMREFERENCE, 2015)

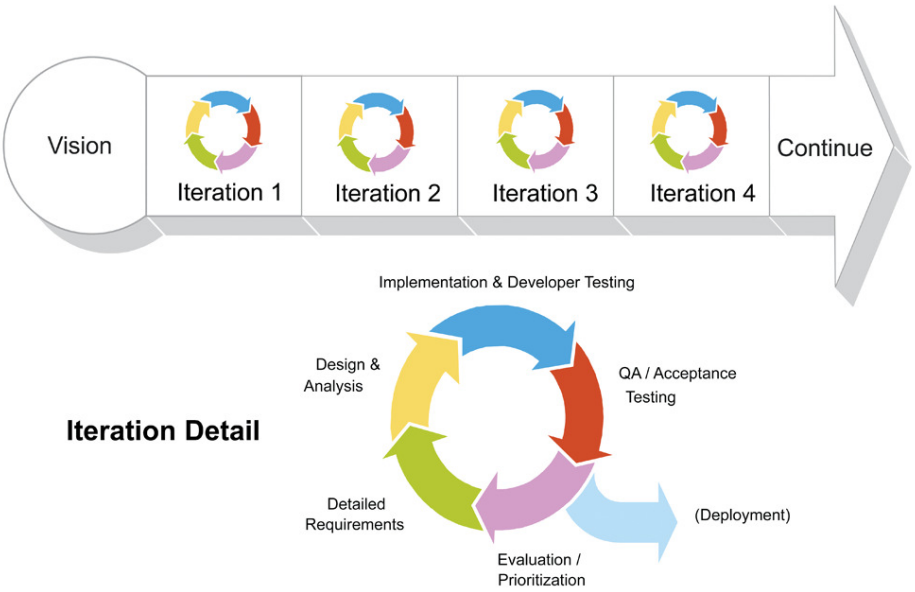

Figura 1: Iterações do Scrum

Com a popularização da metodologia de desenvolvimento Ágil, que tem, dentre outros objetivos, o de entregar com maior frequência, e melhorar a comunicação entre os times, é simples fazer a relação de DevOps com esse tipo de desenvolvimento. DevOps nada mais é do que a implementação de conceitos e mudanças organizacionais e culturais provenientes do pensamento Ágil (AMBLER, 2014).

DevOps tenta alcançar entregas mais frequentes ao preparar um ambiente que facilite, automatize e integre vários dos processos que antes seriam manuais, e mais suscetíveis à falhas e atrasos, o que não é possível sem uma equipe integrada nesse ambiente.

Dessa forma, o conceito de entrega contínua e de integração contínua estão fortemente relacionados à DevOps. (BERTRAM, 2015)

2.2.1 Integração Contínua

Integração contínua é a prática de integrar diversas partes de um software desenvolvido em diversas frentes, de maneira periódica, ou a cada mudança. Foi adotado como parte da *extreme programming* (XP) que sugere integrar partes do software mais de uma vez por dia (FOWLER; FOEMMEL, 2006).

Mesmo que não se adote desenvolvimento orientado a teste (TDD - test driven development), uma funcionalidade só está pronta se estiver com seus testes implementados, levando em consideração metodologias de desenvolvimento Ágil. E dessa forma a integração contínua pode dar retorno com relação aos resultados desses testes a todo momento que ocorrer uma nova integração do software.

2.2.2 Entrega Contínua

A Entrega Contínua é uma prática adotada pelos métodos ágeis que tem o objetivo de preparar um *software* para que ele seja passível de ser posto em produção a qualquer momento (OLAUSSON; EHN, 2016).

A prática de Entrega Contínua é frequentemente confundido com a Integração Contínua. Existe uma relação de dependencia entre as duas práticas para construir uma estrutura que possa sustentar a entrega contínua de um *software*. Olausson e Ehn (2016) resume os dois em:

- **Integração Contínua**: é voltada para estabelecer uma rápida validação da fase de desenvolvimento;
- **Entrega Contínua**: é voltada para estabelecer uma cultura onde pode-se oferecer um recurso ou *feature* para o cliente a qualquer momento.

2.3 Automação

Em DevOps, a automação é um elemento essencial para se alcaçar a maturidade de entregas e *feedback* rápidos. A automação consiste na execução automatica de um processo com o mínimo de intervensão humana. Para a área de TI, a automação ocorre nos processos de controle e administração de sistemas ou softwares (SHARMA; SONI, 2015).

Pode-se citar algumas das vantagens da automação (SHARMA; SONI, 2015):

- Ajuda a reduzir a complexidade de um processo;
- Ajuda a reduzir possibilidade de erros humanos em tarefas repetitivas;

Sharma e Soni (2015) aborda as necessidade de se adotar automação na área de TI (Tecnologia da Informação) e os relaciona com conceitos métodos ágeis (várias implantações em um curto intervalo de tempo), entrega contínua (várias *releases* entregues repidamente), computação em nuvem (tendência do mercado a utilizar infraestrutura em nuvem), etc. Além disso, são citados os benefícios da automação mapeados com as principais preocupações da industria de TI. Algumas delas:

- **Agilidade**: promove pontualidade e agilidade para a TI. Em conjunto com os métodos ágeis resulta em múltiplas implantações em um curto intervalo de tempo, além do rápido *feedback*;
- **Escalabilidade**: a automação ajuda a transformar a infraestrutura em códigos simples, ou seja, a construção, reconstrução e configuração é possível ser feita em poucos minutos. Sendo assim é possível manipular grandes quantidades de ambientes;
- **Precisão de Implantação**: com a utilização de *scritps* é possível realizar rápidas mudanças nas configurações de um ambiente obtendo os resultados esperados.

A automação, em conjunto com a cultura DevOps, consegue suportar rápidas mudanças, entrega contínua, correção de *bugs*. Tudo é feito com a utilização de código que inclui vantagens como testes, versionamento de código e integração de aplicações (SHARMA; SONI, 2015).

2.3.1 Infraestrutura como Código

Segundo Huttermann (2012), em linhas gerais, o que é considerado infraestrutura são os itens como sistema operacional, servidoes, *switches* e *routers*, mas pode, também, ser a combinação de todos os ambientes da empresa e os serviços de suporte (*firewall*, sistema de monitoramento, etc).

Antes dos conceitos de DevOps e dos movimentos ágeis, as configurações de infraestrutura eram automatizadas com *scripts* que geralmente eram difíceis de serem compreendidos por alguém que não fosse o autor (HUTTERMANN, 2012). Recentemente, o termo Infraestrutura como Código veem se popularizando seguindo a mesma lógica que era informalmente aplicada anteriormente, criando *scripts* de automação de configuração para a infraestrutura.

A infraestrutura como código é focada em manipular a configuração da infraestrutura da mesma maneira que os desenvolvedores manipulam os seus códigos: escolhendo a

melhor linguagem e ferramenta para o desenvolvimentos da solução, transformando uma especificação em algo executável que possa ser aplicada em um sistema de forma eficiente e repetível (HUTTERMANN, 2012).

Listing 2.1: "Código em Shell"

```
#!/bin/bash
FILE=/etc/nginx

cat << EOF >
    /etc/nginx/conf.d/mysite.conf
upstream mysite {
  server 127.0.0.1:8081;
}

server {
  listen       *:80;
  server_name 196.164.86.12;

  location /mysite/ {
  }
}
EOF

systemctl restart nginx
```

Listing 2.2: "Código em Chef"

```
template
    "/etc/nginx/conf.d/mysite.conf" do
  action :create
end
service "nginx" do
  action :restart
end
```

A ferramenta Cupper, proposta deste trabalho, é focada em auxiliar uma infraestrutura que utiliza ou pretende utilizar a ferramenta Chef. O Chef consiste em uma ferramenta de automação de configuração de infraestrutura da qual utiliza o conceito de infraestrutura como código para estabelecer os *scripts* de configuração. Os códigos apresentados em 2.1 e 2.2 são exemplos de um *script* escrito em *shell* e outro em *Ruby*, que é interpretada pela ferramenta Chef.

Na seção 2.3.2.1 é descrito a ferramenta Chef e em qual contexto ela se encaixa.

2.3.2 Ferramentas de Automação

As ferramentas de automação, na área de TI, são utilizadas para melhorar a confiabilidade, eficiência e precisão (SHARMA; SONI, 2015). Não se restrige apenas a automação de infraestrutura, mas pode-se também citar *framework* de testes unitários ou testes de comportamento, *build*, qualidade de código. Dentro do contexto de infraestrutura como código, tem-se as ferramentas mais populares Chef e Puppet.

Sharma e Soni (2015) lista as ferramentas de automação que são alternativas ao Chef. Cada qual trata a automação de uma forma diferente apresentando *features* específica que provê desde da automação de configuração da rede até a configuração de ambientes. Dentre elas, destaca-se as mais populares: Chef, Puppet, SaltStack e Ansible. A tabela 2.3.2 mostra a comparação entre as ferramentas.

	Chef	**Puppet**	**SaltStack**	**Ansible**
Linguagem	Ruby (cliente) e Ruby/Erlang (servidor)	Ruby	Python	Python
Arquitetura	Master/Agente	Master/Agente	Master/Agente	Master/Agente
Mecanismo de Push/Pull[1]	*Pull*	*Pull*	*Push*	*Push*
Cloud Integration	Amazon EC2, Windows Azure, HP Cloud, Google Computer Engine, Joyent Cloud, Rackspace, VMWare, IBM Smartcloud e OpenStack	Amazon AWS, VMware e Google Computer Engine	Amazon AWS, Rackspace, SoftLayer, GoGrid, HP Cloud, Google Computer Engine, VMware, Windows Azure e Parallels	Amazon AWS, VMware, OpenStack, CloudStack, Eucalyptus Cloud e KVM.
Empresas que Utilizam	Facebook, Linkedin, Youtube, Splunk, Rackspace, GE Capital, Digital Science e Bloomberg	Twitter, Verizon, VMware, Sony, Symantec, Redhat, Salesforce, Motorola e Paypal.	Lyft	Apple, Juniper, Grainger, WeightWatchers, SaveMartand e NASA

Tabela 1: Comparativo das ferramentas de automação, adaptado de Sharma e Soni (2015)

As vantagens e pontos chaves que o Chef tem em relação ao seus concorrentes (SHARMA; SONI, 2015):

- O Chef é puramente uma *Domain Specific Language (DSL)*. Deursen, Klint e Visser (2000) propões a definição de DSL como uma linguagem de programação executável que contém notações e abstrações específicas para um domínio de problemas. O Chef extende a sua DSL da linguagem Ruby;
- O Chef contém uma consistente documentação, materias de treinamento e atualizações constantes.
- O suporte da comunidade do Chef é grande e responde rapidamente a duvidas e problemas de instalação, configuração e utilização da ferramenta Chef. Além disso, constantemente publicam conferências *online* e materiais multimídia gratuitamente.

A ferramenta Cupper, proposta deste trabalho, é focada para auxiliar especificamente o Chef. Além das vantagens apresentadas, também foi considerado a experiência do grupo com a ferramenta e a utilização dela durante o curso para determinar a sua utilização neste projeto.

2.3.2.1 Chef

Chef é uma ferramenta de gerenciamento e configuração de infraestrutura criada pela comunidade Opscode em 2008 oficialmente lançada em 2009. Seu propósito é tran-

[1] Mecanismo de *network communication*. *Push*: o servidor realiza a requisição. *Pull*: o cliente realiza a requisição por dados.(CHO et al., 2007)

formar uma infraestrutura complexa em código (CHEF, 2008). Sendo assim, a gerência de configuração gira em torno da codificação simplificada e amigável ao invés de comandos manuais de instalação e configuração de aplicações (SHARMA; SONI, 2015).

A estrutura completa da ferramenta Chef contém vários componentes interagindo entre si para prover ao cliente, ou seja, o ambiente alvo, as informações e instruções necessárias para que ele possa executar sua função. Os principais componentes são (CHEF, 2016b):

- *Workstation*: qualquer máquina que possa servir como estação responsável por permitir usuários criar, testar e manter *cookbooks*;
- *Cookbook*: contém as configurações desejadas para o ambiente. Pode ser customizados para um ambiente específico da empresa ou utilizar *cookbooks* disponíveis pela comunidade Chef;
- *Ruby*: linguagem oficial utilizada para a escrita dos *scripts* é o Ruby;
- *Node*: qualquer máquina física, virtual, em núvem, dispositivo de rede, etc, que deva ser gerenciada pelo Chef;
- *Chef Client*: ferramenta instalada em todos os *nodes*. O *Chef Client* é responsável por executar todas as tarefas especificadas em uma *run-list* (conjunto de *cookbooks*). Esse componente é executado pela aplicação CLI *chef-client*;
- *Chef Server*: funciona como um *hub* de informações. Todos os *cookbooks* e as políticas são atualizadas no Chef Server pelos usuários dos *workstations*. O Chef Client acessa o Chef Server para verificar as informações necessárias para sua tarefas e retorna dados para o servidor que serão usados para gerar relatórios;
- *Chef Analytics*: visibilidade em tempo real de dados informativos sobre o servidor como mudanças realizadas, os autores das mudanças e quando ocorreram. Além de detalhes das tarefas executadas nas máquinas *nodes*;
- *Chef Supermarket*: local central onde a comunidade Chef cria e mantém os *cookbooks*. Podem ser customizados de acordo com as necessidades da organização.

Desse componentes, tem-se os três principais: **Chef Server, Node** e **Workstation**. A figura 2 mostra a relação entre esses três componente. Existem os requisitos mínimos para a implementação do Chef (CHEF, 2016b):

- As máquinas *node* devem ter uma plataforma instalada (RedHat, Debian, OpenSUSE, etc);

- A maquina que contenha o Chef Server deve ter os requisitos mínimos de hardware especificada no site oficial do Chef;
- Todas as regras de rede e de *firewall* devem estar configuradas corretamente.

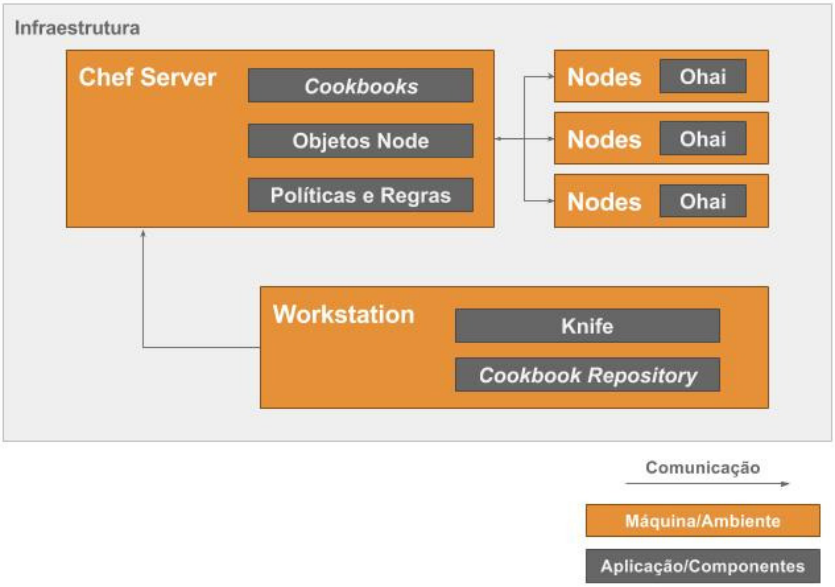

Figura 2: Relação entre os componente do Chef

Chef utiliza os *cookbooks* sendo a unidade básica de configuração do ambiente. Neles são definidos comandos, arquivos, e outros atributos para estabelecer o estado desejado para a instalação e configuração do ambiente alvo (SHARMA; SONI, 2015).

Sharma e Soni (2015) Classifica os *cookbooks* em três categorias:

- **Aplicação**: contém as configurações de acordo com a instalação de uma aplicação. Exemplo PostgreSQL, Apache, Nginx;
- **Biblioteca**: define recursos a serem utilizados por outros *cookbooks*. Não é recomendado utilizado diretamente em um node;
- ***Wrapper***: são *cookbooks* prontos construidos pela comunidade que podem ser alterados para que se adequem ao ambiente que será implantado.

O componente *chef-client* irá executar as receitas (mais detalhes sobre receitas ou *recipes* na seção 4.2) disponíveis nos *cookbooks* indicados pelo *Chef Server* quando necessário, ou seja, se não houver nenhuma mudança quanto as configurações do ambiente o *chef-client* não irá alterar nada, do contrário ele irá aplicar as configurações necessárias (CHEF, 2016b).

2.4 Extração de Configuração

No processo de automação de uma infraestrutura tem-se a extração das informações referente ao sistema. Algumas ferramentas realizam esse processo como o Ohai (CHEF, 2016a) e Facter (PUPPET, 2016a), direcionados para o Chef e Puppet, respectivamente. Essas informações são utilizadas para monitoramento do ambiente controlados pelas ferramentas de automação.

O método de extração consiste na utilização de outras ferramentas de sistema, ou seja, a saída da execução de uma ferramenta contém as informações necessárias a serem extraídas, podendo estar filtrada ou não (CHEF, 2016a). O Ohai, por exemplo, utiliza-se de *plugins* que definem o método de coleta das informações, definindo a ferramenta de sistema a ser utilizada e o método de filtragem.

Como será visto no capítulo **??**, este é o primeiro passo da propósta. As informações extraídas são referentes as configurações do ambiente. O passo seguinte é a criação dos *scripts*. Nas pesquisas realizadas, apenas uma ferramenta, nomeada Blueprint, realiza estes passos.

2.4.1 Blueprint

O Blueprint é uma ferramenta de gerência de configuração que realiza a engenharia reversa do sistema para extrair, em conjunto a outras ferramentas, as informações de pacotes, serviços e fontes de instalação de aplicativos (DEVSTRUCTURE, 2011). O Blueprint foi descontinuado em 2014.

A aplicação é utilizada para criar *scripts* que realizam a instalação dos pacotes e configurações dos serviços que foram extraidos. Há também a opção de realizar a conversão desses *scripts* para receitas Chef e módulos Puppets (funcionam como receitas Chef, mas são específicos para a ferramenta Puppet). Por esse motivo, o Blueprint é considerado como uma ferramenta concorrente ao Cupper, pois a sua funcionalidade é semelhante.

A tablela 2 mostra uma comparação entre o Cupper e Blueprint.

	Cupper	Blueprint
Informações Extraídas	Pacotes, serviços, configurações, rede, plataforma, hardware	Pacotes, serviços, configurações
Saídas	*Cookbook Chef*	*Script Shell, Cookbook Chef, Module Puppet*
Linguagem de Programação	*Ruby*	*Python*
Gerenciador de Pacotes	APT/dpkg, Pacman, RubyGem e PIP	APT, Yum, RubyGems, easy_install, PIP, PECL e NPM.

Tabela 2: Comparativo entre as ferramentas Cupper e Blueprint

É considerado como vantagem do Cupper em relação ao seu concorrente:

- O Cupper é focado na ferramenta Chef;
- O Cupper segue os padrões adotados pelas outras ferramentas do Chef;

3 Metodologia

Neste capítulo serão apresentadas as adaptações das metodologias e técnicas definidas para o desenvolvimento deste trabalho. As seções estão dispostas em:

1. **Levantamento Bibliográfico**: define os métodos utilizados nas pesquisas e levantamento bibliográfico;
2. **Escolha de Tecnologias**: mostra as tecnologias e ferramentas de suporte e dependentes ao desenvolvimento do trabalho;
3. **Análise do Ambiente**: mostra o modo de definição das camadas e os critérios para a seleção da mesma;
4. **Definição dos Recursos Chef**: mostram o modo de escolha dos recurso do Chef a serem utilizados;
5. **Metodologia de Desenvolvimento** que mostra o métodos e práticas da engenharia de software definidos para serem utilizados no desenvolvimento do trabalho;
6. **Coleta e Análise de Resultados** que mostra como será feito a validação da proposta.

3.1 Levantamento Bibliográfico

Para o referenciamento desse trabalho,foram levantados estudos a cerca de DevOps, Chef, Infraestrutura como Código e outros assuntos relevantes com auxílio do Periódico Capes, Google Acadêmico dentre outras fontes. Apesar de serem assuntos recentes, existe muita discussão e pesquisa a respeito deles. Há muitas aplicações desses conceitos, principalmente em empresas de software de renome como Google, Facebook, Apple, em empresas recentes e em start-ups. Dessa forma, existem muitas publicações e palestras nos canais usados por essas empresas e grupos, para divulgar experiências e conhecimentos a respeito desses assuntos e essas fontes também foram utilizadas para referenciar esse trabalho.

3.2 Escolha de Tecnologias

O Cupper, aplicação proposta por este trabalho, tem a intenção de entrar na família de ferramentas relacionadas ao Chef. Portanto é interessante que as tecnologias, linguagem e padrões sejam adotados.

Apesar de não ter sido identificado outra ferramenta similar, com exceção ao Blueprint, para as outras ferramentas de automação de infraestrutura, como o Puppet e Ansible, o motivo da descisão do Cupper ser direcionado para o Chef vem da experiência do grupo com a utilização da ferramenta que por consequência resultou na identificação da oportunidade de melhoria.

O Chef está relacionado ao desenvolvimento do Cupper, mas ele não é uma dependência para a execução do mesmo, e sim uma ferramenta que usa a saída do Cupper, que será explicado no capítulo 4, de Desenvolvimento.

3.2.1 Ferramentas e Serviços de Suporte ao Desenvolvimento

Para o desenvolvimento do trabalho, foram escolhidas as ferramentas e serviços de suporte que serão utilizadas para gerência e controle das atividades do projeto e automação e coleta de dados e teste de desenvolvimento do Cupper. A escolha foi feita considerando a experiência do grupo em relação a utilização da ferramenta no decorrer do curso e a identificação dos padrões utilizados pelas outras ferramentas providas pelo Chef.

3.2.1.1 Git

Git é uma ferramenta de controle de versão gratuita e de código aberto criada em 2005 por Linus Torvalds (CHACON; STRAUB, 2014). O Git foi utilizado para o controle das versões dos códigos para a produção deste documento e para a produção da ferramenta proposta Cupper.

3.2.1.2 Github

O Github é um serviço que provê a utilização dos comando Git em um *browser*, além de disponibilizar um repositório remoto para colaboração de projetos e registro e rastreamento de *issues* (GITHUB, 2016). O Github foi utilizado para armazenamento remoto do repositório deste documento e da ferramenta proposta Cupper, e para o registro e controle de *issues* que são considerados os itens do *Backlog*, como descrito na seção 3.5.2.

3.2.1.3 Travis

O Travis é um serviço de integração contínua integrada com o serviço Github que automatiza a *build* do código e verifica os testes (TRAVIS, 2016). O Travis será utilizado para realizar a *build* automatica da ferramenta Cupper, determinando se a funcionalidade ou *bugfix* será integrada ao código.

3.2.1.4 RubyGems

O RubyGems é um *framework* de empacotamento e instalação de bibliotecas e aplicações construidas com Ruby. As vantagens de se utilizar RubyGem (THOMAS; HUNT, 2001):

- Padronizar formatos de pacotes;
- Centralizar o repositório para distribuição dos pacotes Gem;
- Facilitar a instalação, gerenciamento e manipulação dos pacotes Gem.

A ferramenta Cupper, por ser construída em Ruby, seguirá os padrões adotados pelo RubyGems.

3.2.1.5 RSpec

O RSpec é um *framework* de *Behaviour-Driven Development* (BDD) criado por Steven Baker em 2005 (CHELIMSKY et al., 2010). Com o RSpec são criados testes que descrevem um comportamento esperado do sistema em contexto controlados. O RSpec facilita a escrita de testes simplificando a sintaxe para descrever os cenários e comportamentos.

3.2.2 Ferramentas Dependentes

Foram identificadas as ferramentas dependentes para a construção do Cupper. São ferramentas que estão dentro da família de ferramentas do Chef e são de código aberto disponíveis no repositório oficial do Chef.

3.2.2.1 Ohai

O Ohai é uma ferramenta para detecção de atributos de um *node*. Os atributos são informações que podem ser: detalhe de plataforma, dados de CPU, dados de redes, etc. O Ohai é uma dependência do *chef-client* e é extensível através de *plugins* para incluir mais tipos de dados a serem coletados (CHEF, 2016a).

Para o desenvolvimento do Cupper, a ferramenta Ohai será utilizada para realizar a interface de coleta de dados. Como será apresentado na seção 4.1, o Ohai, sem extensões, pode coletar uma certa quantidade de atributos do ambiente, sendo necessário a criação de *plugins* de acordo com a necessidade de implementação do Cupper.

3.3 Análise do Ambiente

Neste trabalho, é considerado como ambiente uma máquina composta de hardware e sistema operacional. São considerado máquinas físicas e virtuais. Essa seção irá descrever como será a análise e extração de informações do ambiente.

3.3.1 Definição de Camadas

Para decidir até que ponto a aplicação irá analisar as configurações do ambiente é preciso definir os tipos de configuração, as características para cada, e separar esses tipos em camadas onde a aplicação pode atuar. Com essas camadas definidas, identifica-se quais camadas são relevantes e viáveis para o escopo do trabalho.

3.3.2 Definição de Critérios para a Seleção

Após levantar as camadas de configuração do ambiente, é necessário definir em quais camadas e em quais dos seus atributos o Cupper realmente vai atuar. Os critérios vão estar relacionados a dois aspectos importantes: a **relevância** para o projeto e a quantidade de **esforço** e **tempo** para a implementação durante o Trabalho de Conclusão de Curso.

Para classificar um atributo como relevante para análise, ele deve seguir os seguintes critérios:

1. Um atributo é relevante quando ele é necessário para a geração de uma receita *Chef* que atenda aos nossos requisitos.

   Na seção 4.2 e 4.3 definimos os recursos *Chef* que iremos utilizar nas criações de *Cookbooks* e até que ponto avançaremos nas funcionalidades do *Cupper*, e dessa forma, os atributos são relevantes se eles se enquadram nesse patamar, para alcançar esses objetivos.

   Um exemplo básico disso é a arquitetura da CPU, que pode alterar a instalação e qual a versão de pacotes a serem instalados.

2. Um atributo também é relevante se é necessário para criação de *logs* e *debugs*, tanto para uso da ferramenta durante o processo de gerar *Cookbooks* quanto para algum retorno para o usuário, para ajudar a entender possíveis erros ao executar o *Cupper*.

Com relação a dificuldade de implementação e de integração ao *Cupper* o atributo deve seguir os seguinte critérios em ordem de prioridade:

1. O atributo tem maior prioridade de ser selecionado se é um dos dados que o *Ohai* lança por padrão e sem extensões.

2. O atributo tem média prioridade se, não é um dos dados que o *Ohai* lança por padrão e sem extensões, mas existe um comando do sistema que facilita sua extração, e dessa forma facilitando a criação de *plugins* pro *Ohai*.

3. O atributo tem menor prioridade se não é lançado pelo Ohai e nem tem comandos do sistema para recuperá-lo.

3.4 Definição dos Recursos Chef

A ferramenta Chef provê recursos para total automação da infraestrutura. Sendo possível preparar, configurar e integrar a infraestrutura com a flexibilidade de manutenção de scripts (SHARMA; SONI, 2015). Para isso, o Chef tem um estrutura complexa envolvendo diversos componentes para o completo funcionamento e utilização de todos os recursos.

De modo análogo ao levantamento de camadas de ambiente,os recursos de *cookbooks* e do Chef necessários para implementação precisam ser levantados e então selecionados para o escopo do projeto.

3.4.1 Levantamento de Recursos

O principal objeto utilizado para a pesquisa do levantamento dos recursos é a documentação oficial do Chef. A documentação é extensa e completa e com base nela será feito um levantamento para avaliar quais recursos serão necessários para a implementação deste projeto. Além disso, existem referências não oficiais de publicações sobre a ferramenta abordando os mesmos recursos, entretanto, em sua maioria, demonstram a utilização do Chef com outras ferramentas de DevOps.

A organização da documentação oficial é dividida nos principais componentes da arquitetura do Chef além de outros item para conhecimento gerais sobre a ferramenta (CHEF, 2016b). São eles:

- *Getting Started*: visão geral de toda a estrutura do Chef como componentes, recursos, ferramentas auxiliares, etc;
- *The Workstation*: todos os recursos envolvidos na máquina de *workstation* como estrutura, linhas de comando, *kit* de desenvolvimento, etc;
- *The Node*: todos os recursos envolvidos nas máquinas *node* como atributos do *node*, componentes, etc;
- *Cookbook*: explicação de toda a estrutura de um *cookbook* como módulos *recipes, templates, files*, etc;

- *The Chef Server*: todos os recursos envolvidos na máquina *server* como gerenciamento de *nodes*, análise de recursos, *logs*, etc;
- *Chef Compliance*: todos os recursos envolvidos na máquina *compliance* como informaçõe sobre a infraestrutura, auditoria de configurações, etc.

Para o projeto Cupper, os foco principal de estudo dos recursos Chef estão concentrados em duas seções principais na documentação oficial: *The Node* e *Cookbook*. As outras seções são complementares para o entendimento de toda a estrutura e, a princípio, não serão consultadas.

3.4.2 Definição de Critérios de Seleção de Recursos

3.5 Metodologia de Desenvolvimento

Esta seção descreve o método de desenvolvimento da ferramenta Cupper. Os itens estão diretamente relacionados com as metodologias, técnicas e práticas utilizadas na Engenharia de Software, tais como ciclo de desenvolvimento, testes, integração contínua, qualidade de código, etc. Todos os itens utilizados fazem parte do conhecimento adquirido durante o curso e poderão sofrer adaptações de acordo com as necessidades do projeto.

Como será observado nas subseções, as práticas e técnicas são utilizadas nas Metodologias Ágeis. A motivação para a utilização da abordagem ágil é a familiaridade e experiência dos desenvolvedores deste trabalho.

3.5.1 Métodos Base

Segundo Gutierrez et al. (2009) o método *Scrum* representa um trabalho em equipe no qual todos os integrantes e envolvidos trabalham para alcançar o mesmo objetivo, alinhando as mudanças e compartilhando os problemas para que todos caminhem para a mesma direção. O mesmo autor descreve o método *Extreme Programming (XP)* como eficiente, flexível e de baixo risco para equipes pequenas e médias que convivem com constante mudanças.

Ambos os métodos definem pepeis, práticas e modelos de ciclo de vida para projetos de desenvolvimentos ágeis e serão utilizados como base para a definição do método de desenvolvimento deste trabalho.

O *Scrum* não é um método voltado para o desenvolvimento de software. Inicialmente foi construído para gerenciar projetos de uma fábrica de automóveis. Contudo, a sua ideia central é o desenvolvimento de sistemas que tem grandes chances de mudanças durante a sua produção (FADEL; SILVEIRA, 2010).

O *XP* foi formalizado com a seus princípios e as doze práticas chaves. A princípio as práticas não surgiram com o *XP*, mas reunidas de forma a serem independêntes uma das outras (FADEL; SILVEIRA, 2010).

3.5.2 Práticas e Técnicas

Diversos dos conceitos e práticas abordados nessa seção também são abordados na seção 2.1, mas aqui são redefinidas e adaptadas para o escopo do trabalho.

São definidas algumas práticas no método *Scrum* (GUTIERREZ et al., 2009). A lista a seguir mostra as que serão utilizadas, bem como a descrição das adaptações para o trabalho:

- ***Sprint***: ciclos onde são desenvolvidos os itens propóstos. Geralmente são intervalos de 2-4 semanas. Ao final de cada *Sprint* é entregue uma porção executável do *software*. Neste trabalho será utilizado *Sprints* com período de 2 semanas;
- **Planejamento da Sprint**: a cada início de *Sprint* é feito uma reunião na qual são priorizados os itens a serem desenvolvidos durante a *Sprint*. Com o auxílio da ferramenta Github, os itens priorizados para a *Sprint* serão postos como *issues* identificadas com a *label* "priority";
- ***Sprint e Product Backlog***: o *Product Backlog* contém dos os itens a serem desenvolvidos no projeto, sendo uma visão macro de tudo a ser feito. O *Sprint Backlog* contém os itens a serem desenvolvidos na *Sprint* corrente. Com o auxilio da ferramenta Github, ambos os itens dos *backlogs* serão dispostos como *issues*;

O XP traz doze práticas essenciais (GUTIERREZ et al., 2009). A lista a seguir mostra as que serão utilizadas, bem como a descrição das adaptações para o trabalho:

- **Entregas Frequentes**: ao final de cada interação deve haver uma entrega dos itens priorizados. Neste trabalho, as integrações serão as (*Sprints*) e os itens priorizados serão aqueles dispostos no *Sprint Backlog*.
- **Teste**: são divididas em duas partes: teste de aceitação, elaboradas pelo cliente, e testes de unidade, elaboradas pelo programador. Será utilizado apenas os testes de unidade com a ferramenta RSpec.
- ***Refactoring***: consiste em simplificar a estrutura, mudar a organização do código, sem que altere o comportamento (BECK, 2000). Neste trabalho será utilizado conforme a necessidade sendo levado em consideração a importancia, impactos na arquitetura da ferramenta e se é prioritário em relação aos outros itens da *Sprint*;

- **Integração Contínua**: o código deve ser integrado e testado constantemente após o desenvolvimento de novas características. Com o auxílio do serviço Travis CI, os *Pull Requests* e *Branchs* serão monitorados quanto a integridade dos testes. Apenas será integrado os códigos com teste e que não tenham falhado durante a inspeção do serviço de integração contínua;

- **Padrões de Código**: padronização de todo o código para que se apresente de forma familiar a toda a equipe, assim facilitando o entendimento. A ferramenta Rubocop realiza a análise do código com os padrões da comunidade Ruby, este será o padrão adotado neste trabalho. Além disso, todo o repositório e itens desenvolvidos serão feitos em inglês.

- **Programação em Pares**: dois programadores utilizam o mesmo equipamento para o desenvolvimento, sendo assim o código está sempre sendo supervisionado por outro programador.

3.5.3 Controle de Versão

Segundo Pressman (2009), a área da engenharia de software responsável por gerenciar e controlar mudanças em um software é a gerência de configuração de software. Nela são definidas as principais atividades que lidam com a identificação de alterações dos itens de trabalho, estabelecedo uma relação entre eles, definir o gerenciamento das versões de trabalho, controlar e auditar as mudanças impostas.

As principais preocupações da gerência de configuração é (PRESSMAN, 2009):

1. Estabelecer as versões estáveis do sistema ou componente conhecido como *baseline*;

2. Possibilitar desfazer modificações no sistema, por conta de erros, rejeição do usuário, etc;

3. Recuperar informações sobre quem e o que foi alterado no sistema e como essa alteração está ligada com as necessidades do projeto;

Quanto as ferramentas de suporte a gerência de configuração, exitem várias alternativas. Dentre elas, como mapeado na seção 3.2.1.1, tem-se o Git que é responsável pelo controle de versão do sistema. Nela é possível contornar as principais preocupações descritas acima.

Driessen (2010) apresenta um modelo de fluxo de desenvolvimento utilizando a ferramenta Git. Nele são abordados as estratégias de controle de *branchs* e gerenciamento de *release*. Com base nesse modelo, a figura 3 mostra o fluxo básico de desenvolvimento do Cupper.

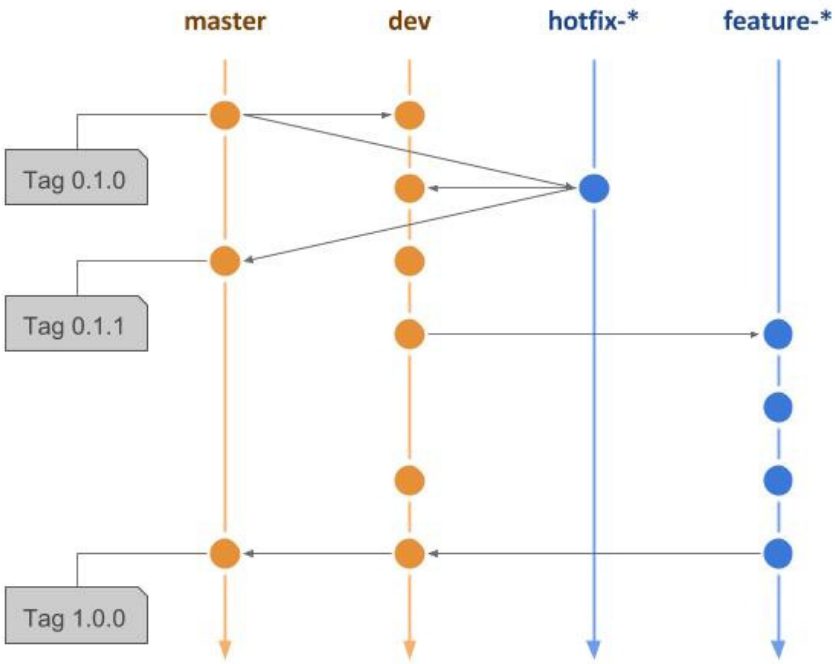

Figura 3: Fluxo de desenvolvimento Cupper

Neste modelo, as principais *branchs* tem o tempo de vida infinito, ou seja, em nenhum momento são removidas do repositório remoto. São elas:

- *master*: reflete o estado de "pronto para produção", ou seja, estabelece uma versão estável do sistema. Todo o trabalho realizado, em algum ponto do desenvolvimento deve ser integrado a esta *branch*;
- *dev*: onde ocorre a integração de todos os componentes, funcionalidades e correções. Contém todos os itens mais recentes desenvolvidos e é considerado a versão instável do sistema, ou seja, pode conter comportamentos indesejados e *bugs* desconhecidos.

Há também as *branchs* de suporte que tem o tempo de vida curto, ou seja, assim que concluidas, devem ser removidas do repositório remoto. São elas:

- *feature*: deve ser ramificada da *branch dev* e integrada a *branch dev* quando finalizada. Contém todos os novos itens desenvolvidos para a nova funcionalidade;
- *hotfix*: deve ser ramificada da *branch master* e integrada a *branch dev* e *master* quando finalizada. São as correções que são realizadas na versão estável do sistema, e geralmente são pontos críticos que devem ser corrigidos imediatamente. Quando finalizadas deve-se criar uma *tag* da versão estável do sistema;

3.6 Coleta e Análise de Resultados

Nesta seção será apresentado o método de coleta e análise dos resultados deste trabalho para validação da proposta. Como disposto nos objetivos (1.2), o resultado da execução do Cupper é um *script* em formato de receita Chef. Tal receita poderá ser utilizada pelo Chef para replicar o ambiente. Sendo assim O foco da coleta e análise de dados está na capacidade de replicar um ambiente utilizando o Cupper e Chef de maneira automatizada.

Divide-se a coleta e análise de dados em quatro etapas:

Etapa 1: Nesta etapa será coletado os dados sobre quais informações o Cupper consegue extrair e quais são as replicações. A coleta será feita por um *checklist* que contenha todos os itens propóstos para implementação da ferramenta. O *checklist* será utilizado para delimitar os dados base que serão coletados do ambiente para a validação. Também é um informativo sobre até qual ponto o projeto conseguiu alcançar.

Etapa 2: Nesta etapa será construído um ambiente que seja possível extraír as configurações sem a utilização do Cupper. Essas informações estarão alinhadas ao *cehcklist* da etapa anterior e serão a base para o comparativo com o ambiente replicado.

Etapa 3: Nesta etapa a ferramenta Cupper irá ser executada no ambiente construído na etapa anterior. As receitas geradas serão usadas pelo Chef em um ambiente limpo que contenha apenas as configurações mínimas para o seu funcionamento, ou seja, um sistema operacional (Debian ou Arch), acesso por interface de rede e o Chef (a seção 2.3.2.1 define o ambiente mínimo para o funcionamento do Chef).

Etapa 4: Nesta etapa é feito a extração sem utilização do Cupper, assim como foi realizado na etapa 2, no ambiente replicado. Então será feito uma comparação das configurações dos dois ambientes.

A conclusão dos dados coletado e analizados será realizado pela porcentagem de item correspondendes dos ambientes como demonstrado na tabela 3.

Camada	**Ambiente (Extração Manual)**	**Ambiente Replicado (Extração Manual)**	**Replicado**
Aplicação	Pacote Nginx Instalado	Pacote Nginx Instalado	X
	Configuração Nginx Aplicada	Configuração Nginx Aplicada	X
	Pacote PostgreSQL Instalado	Pacote PostgreSQL Instalado	X
Serviço	Serviço Nginx Executando	Serviço Nginx Não Executando	
Porcentagem Replicada			75%

Tabela 3: Exemplo de tabela de comparação dos resultados.

A extração manual será feita com a utilização de *scripts bash*. O motivo da utilização desse método é a necessidade de obter os dados de forma imparcial, ou seja, as informações extraídas para a comparação não podem ser realizadas pelo Cupper ou Ohai, visto que estes são parte do processo de extração e configuração do ambiente replicado e não podem ser utilizados para validar a si mesmo.

4 Desenvolvimento

No contexto de DevOps, automação e infraestrutura como código, a ferramenta proposta por este trabalho, Cupper, propõe auxiliar na extração de configuração de um ambiente incluso em uma infraestrutura. As configurações extraídas são transformadas em código em padrões da receita Chef 4. A receita gerada é utilizada para qualquer infraestrutura que suporta a utilização do Chef, como descrito na seção 2.3.2.1.

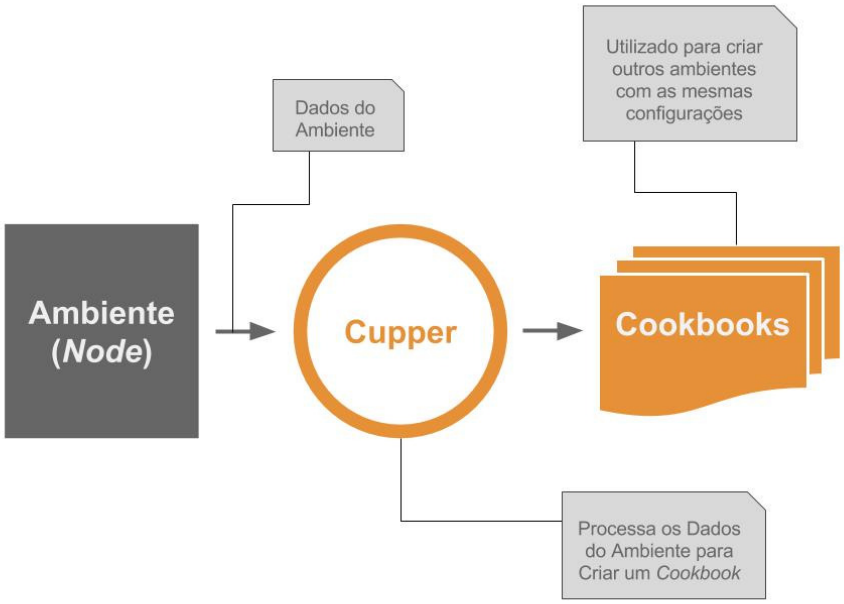

Figura 4: Cupper - Visão geral

Além das dependências de desenvolvimento e de *runtime* por ser uma Gem (*Ruby*, *RubyGems*), o Cupper tem uma dependência muito importante que é o *Ohai* que é levantado na seção de ferramentas 3.2.2.1. É requisito desse projeto também implementar *plugins* para que o *Ohai* lance informações que por padrão não lança.

As subções seguintes, são mostrados o levantamento dos itens necessários para a implementação do Cupper, estando organizados em camadas e seus atributos:

- **Camadas de Ambiente**: levantamento e seleção das camadas de ambiente e seus atributos necessários a serem coletados;
- **Recursos Chef**: levantamento dos recursos providos pelo Chef e quais os itens necessários para a criação de uma receita mínima;

- **Escopo de Implementação**: especificação dos itens a serem implementados da ferramenta, abordando até qual ponto a ferramenta Cupper pretende alcançar no TCC 2.

4.1 Camadas de Ambiente

As camadas apresentam um conjunto de aspectos do sistema e contém as informações necessárias que definem o comportamento específico para o estado desejado daquele ambiente. Os atributos de cada camada são variantes que são consideradas para o *deployment* de uma aplicação. Sendo assim, as camadas são dependentes para o nível de compatibilidade da configuração.

As seguintes camadas foram definidas para representar o estado de configuração do sistema:

- ***Hardware***: definições físicas onde o sistema foi implantado. (Arquitetura, memória, espaço em disco, etc);
- ***Operation System***: definições do sistema operacional implantado. Distribuição, arquitetura, versão, etc;
- ***Application***: definições das aplicações instaladas. (Aplicações instaladas, dependências, etc);
- ***Configuration***: definições das configurações das aplicações. (Especificações de implantação de aplicação, arquivos de configuração);
- ***Service***: definições dos serviços daemon que estão em funcionamento no sistema.
- ***Custom***: definições criadas especificamente para o sistema sem uma forma padrão conhecida.

4.1.1 *Hardware*

A camada de *Hardware* contém as definições físicas do sistema. As informações são referentes as configurações físicas da máquina, como CPU, arquitetura, memória, espaço em disco, particionamento, etc. O informe desses atributos são utilizados para a definição da base do ambiente, ou seja, o sistema operacional e as aplicações podem ter diferentes desempenhos a partir das configurações de hardware e/ou apresentar comportamentos inesperados no sistema. Além das aplicações terem seus requisitos mínimos, algumas são desenhadas para um tipo específico de arquitetura.

A tabela 4 mostra diversas informações de hardware possíveis de serem extraídas do sistema a partir de comandos do sistema operacional e também da ferramenta Ohai,

dependência prevista para a implementação do Cupper (como descrito na seção 3.2). Nem todas essas informações serão úteis para a implementação, mas nessa seção fazemos um levantamento geral antes de fazer uma seleção.

A camada de hardware será uma das camadas a serem analisadas, e a seleção de seus atributos irão seguir os critérios descritos na seção 3.3.2.

A tabela 4 mostra atributos relacionados a CPU, Memória e *Moterboard*, e a tabela 5 mostra atributos relacionados a partições, armazenamento, dispositivos PCI, USB e dispositivos de rede.

A primeira coluna das tabelas mostra o atributo em questão; a segunda mostra a origem do atributo, podendo ser um comando do sistema, ou caminho de um arquivo; a terceira coluna mostra se o atributo é relevante para a aplicação; a quarta mostra se esse atributo pode ser conseguido por meio do *Ohai*; a quinta coluna diz se é viável implementar o acesso a esse atributo e a ultima coluna diz se o atributo foi selecionado para ser usado pelo *Cupper*.

Tabela 4: Atributos Relacionados a CPU, Memória e *Motherboard*

Atributo	Origem	Relevante?	Ohai	Viável?	Selecionado?
Modelo CPU	lscpu	não	sim	sim	não
Modo de Operação CPU	lscpu	sim	sim	sim	sim
Arquitetura CPU	lscpu	sim	sim	sim	sim
Cores	lscpu	sim	sim	sim	sim
Stepping CPU	lscpu	não	sim	sim	não
Frequência CPU	lscpu	sim	sim	sim	sim
Cache CPU	lscpu	sim	sim	sim	sim
Flags CPU	lscpu	sim	sim	sim	sim
Modelo Motherboard	dmidecode	não	não	não	não
Versão BIOS	dmidecode	não	não	não	não
Flags BIOS	dmidecode	não	não	não	não
Slots Memória	dmidecode	não	sim	sim	não
Marca Memória	dmidecode	não	não	não	não
Tamanho Memória	free	sim	sim	sim	sim
Clock Memória	dmidecode	não	não	não	não
Total Swap	free	sim	sim	sim	sim
Livre Swap	free	sim	sim	sim	sim
Total Memória	free	sim	sim	sim	sim
Livre Memória	free	sim	sim	sim	sim
Buffers Memóra	free	sim	sim	sim	sim
Cached Memória	free	não	sim	sim	não
Ativo Memória	free	sim	sim	sim	sim
Inativo Memória	free	sim	sim	sim	sim
Sujo Memória	pmap	sim	sim	sim	sim
Pag. Anon Memória	pmap	não	sim	sim	não

Tabela 5: Atributos relacionados a Partições, Armazenamento, USB, PCI, e Rede

Atributo	Origem	Relevante?	Ohai	Viável	Selecionado?
Nome dos Dispositivos	lsblk	sim	sim	sim	sim
Major/Minor Dispositivos	lsblk	sim	não	sim	sim
Removíveis	lsblk	não	não	não	não
Somente Leitura	lsblk	não	não	não	não
Partições	lsblk	sim	sim	sim	sim
Tamanho Partições/ Discos	lsblk/ df	sim	sim	sim	sim
Tipo Filesystem Dispositivos	lsblk	sim	sim	sim	sim
UUID Dispositivos	lsblk	sim	sim	sim	sim
Mountpoints	mount	sim	sim	sim	sim
Dsipositivos USB	lsusb	não	não	não	não
Dsipositivos PCI	lspci	sim	não	sim	sim
Dispositivos de Rede	lspci/ ifconfig/ dmidecode	sim	sim	sim	não

4.1.2 *Operation System*

A segunda camada de extração de dados é a do Sistema Operacional. Cada sistema tem a sua particularidade com relação aos gerenciamento dos recursos de hardware e administração de processos. Portanto é necessário a construção de uma arquitetura flexível, que contenha os aspectos em comum aos sistemas operacionais, e modular para que seja possível escalar a utilização da ferramenta para mais Sistemas Operacionais.

Inicialmente a ferramenta Cupper irá abordar dois sistemas operacionais, ambos baseados em GNU/Linux: Archlinux e Debian.

A tabela 6 mostra informações a respeito do Kernel e da distribuição analisada. Também mostra algumas configurações gerais do sistema, como configurações de rede.

A primeira coluna das tabelas mostra o atributo em questão; a segunda mostra a origem do atributo, podendo ser um comando do sistema, ou caminho de um arquivo; a terceira coluna mostra se o atributo é relevante para a aplicação; a quarta mostra se esse atributo pode ser conseguido por meio do *Ohai*; a quinta coluna diz se é viável

implementar o acesso a esse atributo e a ultima coluna diz se o atributo foi selecionado para ser usado pelo *Cupper*.

Tabela 6: Atributos relacionados ao Sistema Operacional, Kernel e configs gerais do sistema

Atributo	Origem	Relevante?	Ohai	Viável?	Selecionado?
OS	/etc/distro-release	sim	sim	sim	sim
Kernel	uname	sim	sim	sim	sim
Versão Kernel	uname	sim	sim	sim	sim
Machine Kernel	uname	sim	sim	sim	sim
Módulos Kernel	lsmod	sim	sim	sim	sim
Distribuição	lsb_release	sim	sim	sim	sim
Release Distribuição	lsb_release	sim	sim	sim	sim
Família Distribuição	lsb_release	sim	sim	sim	sim
Geren. de Pacotes	N/A	sim	não	sim	sim
Init System	N/A	sim	não	sim	sim
Módulo de Segurança	N/A	sim	não	sim	sim
Hostname	hostname	sim	sim	sim	sim
Interfaces de Rede	ip	sim	sim	sim	sim
Endereços de Rede	ip	sim	sim	sim	sim
Endereços MAC	ip	sim	sim	sim	sim

Por ser específico de cada distribuição linux, não é possível determinar com um comando qual o gerenciador de pacotes o sistema utiliza. O mesmo acontece para o sistema de inicialização de serviços e para o módulo de segurança utilizados. Dessa forma é necessário implementar a captura dessas informações em um plugin para o *Ohai*.

4.1.3 *Application*

Essa camada trata das aplicações instaladas no sistema. Aplicações podem ter sido instaladas por gerenciadores de pacotes oficiais da distribuição, por gerenciadores não oficiais, por gerenciadores específicos de módulos para linguagens e por fim podem ser instalados manualmente. Nessa camada iremos suportar somente pacotes instalados por gerenciadores oficiais e por gerenciadores específicos de linguagem python e ruby (Pip

e RubyGem). Na camada *Custom* a aplicação irá tentar suportar aplicações instaladas de outras maneiras.

A primeira coluna das tabelas mostra o atributo em questão; a segunda mostra a origem do atributo, podendo ser um comando do sistema, ou caminho de um arquivo; a terceira coluna mostra se o atributo é relevante para a aplicação; a quarta mostra se esse atributo pode ser conseguido por meio do *Ohai*; a quinta coluna diz se é viável implementar o acesso a esse atributo e a ultima coluna diz se o atributo foi selecionado para ser usado pelo *Cupper*.

Tabela 7: Atributos relacionados às Aplicações

Atributo	Origem	Relevante?	Ohai	Viável	Selecionado?
Pacotes	pacman/ dpkg	sim	não	sim	sim
Versões Pacotes	pacman/ dpkg	sim	não	sim	sim
Dependências	pacman/ dpkg	sim	não	sim	sim
Arch	pacman/ dpkg	sim	não	sim	sim
Pacotes Pip	pip	sim	não	sim	sim
Versões Pacotes Pip	pip	sim	não	sim	sim
Pacotes Ruby	gem	sim	não	sim	sim
Versões Pacotes Ruby	gem	sim	não	sim	sim
Pacotes por outro Ger.	N/A	não	não	não	não
Instalação Manual	N/A	não	não	não	não

Nenhum dos atributos dessa camada pode ser extraído pelo *Ohai* por padrão. Dessa maneira toda a extração desses atributos deverá ser implementada num plugin para o *Ohai*.

4.1.4 *Configuration*

A analise dessa camada está relacionada às configurações específicas de cada aplicação, e do sistema. Dados como especificações de implantação das aplicações e arquivos de configuração são analisados nessa camada.

Para os arquivos de configuração iremos considerar o sistema de hierarquia de sistema de arquivos do linux. Alguns arquivos que não seguem o padrão FHS (*Filesystem Hierarchy Standard*, Hierarquia de sistema de arquivos) irão ser tratados na seção **??** de configurações e instalações *Custom*.

Tabela 8: Atributos relacionados às Configurações de Aplicações

Atributo	Origem	Relevante?	Ohai	Viável	Selecionado?
Host Config	/etc	sim	não	sim	sim
Add-on Config	/etc/opt	sim	não	sim	sim
Sgml Config	/etc/sgml	sim	não	sim	sim
X11 Config	/etc/X11	sim	não	sim	sim
Xml Config	/etc/xml	sim	não	sim	sim
User Config	/home/user/ .config	sim	não	sim	sim
User Custom	/home/user/ .some-app	não	não	não	não
Root config	/root/*	não	não	não	não

A primeira coluna das tabelas mostra o atributo em questão; a segunda mostra a origem do atributo, podendo ser um comando do sistema, ou caminho de um arquivo; a terceira coluna mostra se o atributo é relevante para a aplicação; a quarta mostra se esse atributo pode ser conseguido por meio do *Ohai*; a quinta coluna diz se é viável implementar o acesso a esse atributo e a ultima coluna diz se o atributo foi selecionado para ser usado pelo *Cupper*.

O padrão HFS já prevê alguns sub diretórios para configurações padrão abaixo de */etc/* como */etc/opt, /etc/X11* por exemplo, mas outros diretórios de arquivos de configuração são importantes, e seguem praticamente o mesmo padrão. Não serão analisados todos os subdiretórios, só serão analisados os que estiverem sobre padrão similar ao HFS.

4.1.5 *Service*

Essa camada está relacionada ao gerenciamento de serviços do ambiente. O gerenciamento de Serviços depende de qual *Init System* é usado pela distribuição, e em alguns casos, mais de uma abordagem é utilizada por algumas distribuições.

Existe a maneira de implementar a checagem da presença de um *Init System* procurando por presença de diretórios e arquivos, mas essa abordagem não é eficiente. Um exemplo de situação em que isso não funcionaria é a de uma máquina com Archlinux, ou alguma outra distribuição que siga rolling release, e que tenha atualizado seu *Init System* por seguir novos padrões da distribuição. Por não precisar formatar pra atualizar essa nova abordagem, resquícios da abordagem anterior continuariam presentes na máquina. Isso é comum em distribuições rolling release justamente por sua cultura de não formatar

e somente atualizar.

Dessa forma, iremos relacionar os *Init Systems* às distribuições diretamente levando em conta o *Init System* padrão da distribuição. A tabela 9 mostra a relação de *Init System* padrão de cada Distribuição. De qualquer maneira o escopo inicial do projeto prevê implementação somente para *Archlinux* e *Debian*.

Tabela 9: Relação entre Distribuições e *Init Systems*

Upstart	Systemd	SMF	SysV	OpenRC	RC	Launchd
Ubuntu	Fedora15	Solaris	RHEL5	Gentoo	BSDs	OSX
Fedora9	Archlinux	OpenSolaris	Debian			
RHEL6	RHEL7	illuminos	Suse			
Centos	openSUSE12	smartos				
Debian						

Na tabela 10 são listados os atributos relacionados a camada *Service*. A primeira coluna das tabelas mostra o atributo em questão; a segunda mostra a origem do atributo, podendo ser um comando do sistema, ou caminho de um arquivo; a terceira coluna mostra se o atributo é relevante para a aplicação; a quarta mostra se esse atributo pode ser conseguido por meio do *Ohai*; a quinta coluna diz se é viável implementar o acesso a esse atributo e a ultima coluna diz se o atributo foi selecionado para ser usado pelo *Cupper*.

Tabela 10: Atributos relacionados a Serviços e daemon

Atributo	Origem	Relevante?	Ohai	Viável	Selecionado?
Status	systemctl/ service	sim	sim	sim	sim
Unidades Rodando	systemctl/ service	sim	sim	sim	sim
Falhas	systemctl/ service	sim	sim	sim	sim
Unidades Instaladas	systemctl/ service	sim	sim	sim	sim
Unidades Status	systemctl/ service	sim	sim	sim	sim
Relação Unidade Pacote	pacman/ dpkg/ apt-file/ pkgfile	sim	não	sim	sim

4.1.6 *Custom*

O trabalho referente a essa camada permeia algumas das camadas anteriores, mas nos pontos em que podem ter procedimentos que não estejam no padrão estabelecido. Com relação à camada de *Application*, pacotes podem ser instalados por meio de Make-

files ou scripts customizados que não atualizam o histórico de pacotes do gerenciador de pacotes. Da mesma maneira, as configurações desses pacotes podem ser geradas por esses scripts, e colocadas em lugares fora do padrão. Mesmo considerando aplicações instaladas por gerenciadores de pacotes, vários arquivos de configuração podem ser instalados em diretórios fora do padrão HFS. Dessa maneira, além de *plugins* para o *Ohai*, *plugins* para o *Cupper* também devem ser feitos para tratar esses casos especiais.

4.2 Recursos Chef

Como descrito ne seção 2.3.2.1, o Chef é disposto em vários componentes. O foco principal do Cupper é a criação de *cookbooks*, sendo assim os recursos levantados são referente a composição interna dos *cookbooks*.

Os *cookbooks* contém os seguintes componentes: *attributes, recipes, definitions, files, libraries, custom resourses, metadata, resources e providers, templates cookbook versions* (CHEF, 2016b).

Para o funcionamento de um *cookbook*, com uma estrutura mínima, é necessário que tenha ao menos um *recipe default* definido. O exemplo 4.1 mostra essa estrutura mínima necessária para realizar a configuração do *cookbook app* e o exemplo 4.2 mostra uma estrura completa.

Listing 4.1: "Estrutura mínima de um *cookbook*"

```
.
|-- cookbooks
  |-- app
    |-- recipes
      |-- default.rb
```

Listing 4.2: "Estrutura completa de um *cookbook*"

```
.
|-- cookbooks
  |-- app
    |-- README.md
    |-- attributes
    |-- definitions
    |-- files
      |-- default
    |-- libraries
    |-- metadata.rb
    |-- recipes
      |-- default.rb
    |-- resource
    |-- templates
      |-- default
```

As subseções seguintes explicam os componentes dos *cookbooks* e quais serão ge-

rados pelo Cupper.

4.2.1 *Attributes*

Os *attributes* são os detalhes das especificações do *node*. Definem (CHEF, 2016b):

- o estado atual do *node*;
- em qual estado o *node* estava após a última execução do *chef-client*;
- em qual estado o *node* deve alcançar ao final na execução do *chef-client* atual.

Os *attributes* são definidos: no proprio ambiente através do Ohai, nos *cookbooks*, nos *roles* e *environments*.

Cupper: será incluido no escopo

4.2.2 *Recipes*

Os *recipes* são as unidade fundamentais para a execução de um *cookbook*. Definem todas as informações necessárias para configurar o sistema e segue algumas regras (CHEF, 2016b):

- Deve ser incluido em um *cookbook*;
- Deve ser posto em um *run-list* antes de ser usado pelo *chef-client*;
- É executado na mesma ordem disposta na *run-list*;
- Pode ser incluído em outro *recipe*;
- Pode ter dependência de outros *recipe*;
- Pode marcar um *node* para facilitar a criação de agrupamento.

As *recipes* são escritas em Ruby e pode-se utilizar dos recursos providos pela linguagem. Dispoem-se de uma coleção desse recursos que são utilizados para definir as ações da *recipes*. No exemplo 4.3 utiliza-se três recursos: *package, service* e *template*.

Listing 4.3: "Exemplo de *recipe*. Define a instalação e configuração do *app* Nginx"

```
1  package "nginx" do
2    action :install
3  end
4
5  service "nginx" do
6    action :nothing
7  end
8
9  template "/etc/nginx/conf.d/sample.conf" do
10   mode 644
11   action :create
12   notifies :restart, 'service[nginx]'
13 end
```

Cupper:

4.2.3 *Definitions*

As *definitions* são um novo tipo de recurso disponível apartir da versão 12.5 do Chef e é recomentado utilizar o *Custom Resource*(seção 4.2.6) no lugar de *definitions* (CHEF, 2016b).

Os *definitions* são comportamento que podem ser reutilizados por outros *recipes*. São utilizado como recursos padrões de uma receita. No exemplo 4.4 é definido o recurso *host_porter* com os parametros *port* (valor padrão 4000) e *hostname* (valor padrão *nil*) que pode ser utilizado em outro *recipe* com a simples chamada *host_porter*.

Listing 4.4: "Exemplo de *definition*. Adiciona um recurso *host_porter*."

```
1  # Arquivo de definition
2  define :host_porter, :port => 4000, :hostname => nil do
3    params[:hostname] ||= params[:name]
4
5    directory '/etc/#{params[:hostname]}' do
6      recursive true
7    end
8
9    file '/etc/#{params[:hostname]}/#{params[:port]}' do
10     content 'some content'
11   end
12 end
13
14 # Arquivo de recipe
15 host_porter 'www1' do
16   port 4001
17 end
```

Cupper:

4.2.4 *Files*

Os *Files* é um recurso referente a manipulação de arquivos no ambiente (CHEF, 2016b). Pode-se utilizar os seguintes recursos:

- *cookbook_file*: arquivos que são adicionados ao *node* com base nos arquivos presentes no diretório */files* na raíz do *cookbook*;
- *file*: manipula arquivos que estão presentes no *node*;
- *remote_file*: arquivos são adicionado ao *node* a partir de um local remoto;

Cada recurso *file* é utilizado para um propósito, mas os seus comportamentos são similares. O exemplo 4.5 mostra três maneira diferente de inserir o arquivo *eth1-conf* no ambiente.

Listing 4.5: "Exemplo de *file*. Três modos de inserir o arquivo *eth1-conf*."

```
1  # Adiciona o arquivo 'eth1-conf' definido em '/files/eth1-conf'
2  cookbook_file '/etc/network/interfaces.d/eth1-conf'
3
4  # Adiciona o arquivo 'eth1-conf' com o conteudo definido em 'content'
5  file '/etc/network/interfaces.d/eth1-conf' do
6    content 'auto eth1'
7  end
8
9  # Adiciona o arquivo 'eth1-conf' definido no local remoto de 'source'
10 remote_file '/etc/network/interfaces.d/eth1-conf' do
11   source 'http://site.com/eth1-conf'
12 end
```

Cupper: será incluido no escopo os recursos *file* e *cookbook_file*. A leitura de um arquivo de configuração ou de implantação de uma aplicação é replicado para a pasta *cookbooks/app/files/* de mesmo nome. O *cookbook_file* será gerado apartir da coleta de arquivos considerado estáticos, ou seja, o conteúdo não tem variação com relação a nenhum atributo do ambiente, por exemplo o atributo IP do ambiente.

4.2.5 *Libraries*

As *libraries* são formas de inserir código Ruby para extender ou definir uma nova funcionalidade. Semelhante ao que ocorre em *definitions*, entretanto a capacidade de extensão é maior sendo possível extender recursos já existentes (CHEF, 2016b). Por ser construída em Ruby, todos os recursos provídos pela linguagem podem ser utilizadas dentro das *libraries*.

O exemplo 4.6 mostra a definição para extender um *recipe* adicionando um método *client* que pode ser utilizado para facilitar a inserção de dados referente a *ip* e *hostname* de um *node*

Listing 4.6: "Exemplo de *libraries*."

```
1 # Extende o recurso Recipe
2 class Chef
3   class Recipe
4     def client(ipaddr)
5       node[ipaddr][:hostname]
6     end
7   end
8 end
```

Em geral, as *libraries* são usadas para definir novas funções que auxiliam na descrição de rotinas que são mais frequentes durante a construção de um *recipe* como verificação de arquivos ou serviços.

Cupper: não será incluido no escopo, pois não é possível definir as *libraries* necessárias para um contexto desconhecido pelo Cupper.

4.2.6 *Custom Resource*

Adicionado recentemente ao Chef, o *custom resource* é uma forma de criar novos recursos para os *recipes*. É semelhante as *libraries* e as *definitions*, entretanto é direcionado especificamente para a criação de novos recursos (CHEF, 2016b). É uma forma simples de extender o Chef e é implementado dentro de um *cookbook*.

No exemplo 4.7 tem-se a definição de um recurso criado em *cookbooks/app/resources* com o nome *httpd* e uma propriedade *homepage* com um valor padrão vazio. Esse recurso é distribuido por todo o *cookbooks* como uma simples chamada de um recurso Chef como demonstra o código 4.8.

Listing 4.7: "Exemplo de declaração de um *custom resource*."

```
# Recurso para sites com httpd
resource_name :httpd

property :homepage, String, default: ''

load_current_value do
  file = '/var/www/sites/index.html'
  if ::File.exist?(file)
    homepage IO.read(file)
  end
end

action :create do
  package 'httpd'

  service 'httpd' do
    action [:enable, :start]
  end

  file '/var/www/sites/index.html' do
    content homepage
  end
end
```

Listing 4.8: "Exemplo de utilização de um *custom resource*."

```
httpd 'build' do
  homepage '<h1> Hello World <h1>'
  action :create
end
```

Cupper: não será incluido no escopo, pois a definição dos *custom resource* dependem da necessidade de cada organização não sendo possível prevê-las.

4.2.7 *Metadata*

Cupper:

4.2.8 *Resources e Providers*

Cupper:

4.2.9 *Templates*

Cupper:

4.2.10 *Cookbook Version*

Cupper:

4.3 Escopo de Implementação

Nessa seção serão descritas as funcionalidades e a estrutura inicial da gem.

4.3.1 Funcionalidades

Já que o *Cupper* é uma ferramenta de linha de comando, suas funcionalidades estão diretamente relacionadas aos subcomandos e suas opções. Dessa forma os comandos chave da aplicação irão ser listados nessa seção.

1. Mostrar Ajuda: Como diversos outras ferramentas de linha de comando, o *Cupper* deve possuir um subcomando que imprime no terminal os principais subcomandos e algumas de suas opções.

2. Gerar Cookbook: Essa é a funcionalidade mais importante do *Cupper*, e é a motivação principal para implementá-lo e utilizá-lo. A figura 4 mostra o fluxo básico do funcionamento.

3. Gerar Relatório: É possível também utilizar o *Cupper* para gerar um relatório mais completo do que o *Ohai* geraria, considerando os *plugins* da família *Cupper* para o *Ohai*. Esse relatório pode ser utilizado com fins de diagnóstico, ou como uma fase intermediária à criação de *Cookbooks*, para entender o que será finalmente escrito nas *Cookbooks* e também por motivos de *debug*;

4. Gerar Diretório Inicial: Essa funcionalidade é referente ao subcomando que irá gerar a estrutura de diretórios padrão para os arquivos de configuração do *Cupper* e para a inserção de *plugins*;

5. Listar *Plugins* do *Ohai*: Funcionalidade referente ao subcomando que lista os *pligins* do *Ohai* para simples conferência.

6. Listar *Plugins Custom* do *Cupper*:Semelhante à funcionalidade anterior, mas lista *Plugins Custom* do *Cupper*.

4.3.2 Estrutura e módulos

A figura 5 mostra a Estrutura Inicial Planejada para os Módulos do *Cupper*.

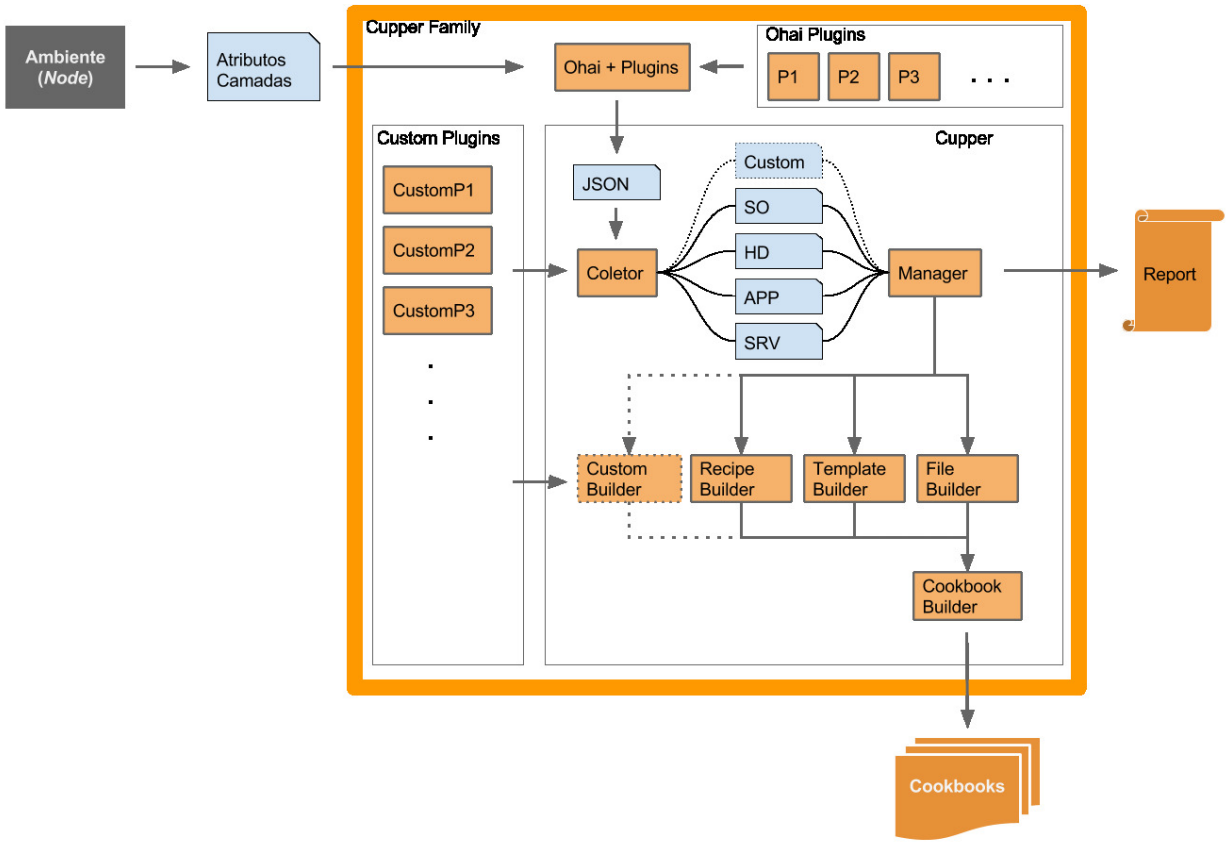

Figura 5: Estrutura Inicial Planejada para os Módulos do Cupper

5 Resultados Parciais

5.1 Cronograma

Esta seção contém o cronograma previsto para a implementação da segunda parte do trabalho (TCC 2). A visão das atividades são referentes ao desenvolvimento do objeto proposto pela primeira parte do trabalho (TCC 1) e pode haver mudanças no decorrer do trabalho.

Tabela 11: Cronograma TCC 1

Atividade	Março	Abril	Maio	Junho	Julho
Escolha do Tema	■				
Levantamento Bibliográfico	■	■			
Estudo Inicial	■	■			
Definição de Objetivos		■	■		
Definições Teóricas		■	■		
Definições de Metodologia		■	■	■	
Descrição do Desenvolvimento		■	■	■	
Desenvolvimento Inicial				■	
Coleta de Dados Parciais				■	■

Tabela 12: Cronograma TCC 2

Atividade	Agosto	Setembro	Outubro	Novembro	Dezembro
Análise Viabilidade Coleta Inicial	■				
Desenvolvimento Funcionalidades Auxiliares	■				
Desenvolvimento da Extração de Atributos Nativos do Ohai		■			
Desenvolvimento da geração de Relatório		■	■		
Desenvolvimento de Plugins Ohai		■			
Desenvolvimento de Geração de Cookbook		■	■		
Coleta de Dados				■	
Analise de Dados				■	
Aplicação de Melhorias					■
Entrega Final					■

Referências

AGILEORG. *Agile Methodology Org.* 2015. Disponível em: <http://agilemethodology.org>. Citado na página 13.

AMBLER, S. W. We need more agile it. San Francisco: UBM, 2014. Citado na página 13.

BECK, K. *Extreme programming explained: embrace change*. [S.l.]: addison-wesley professional, 2000. Citado na página 28.

BERTRAM, A. *Signs you're Doing Devops Wrong*. 2015. Disponível em: <http://www.infoworld.com/article/3011631/devops/7-signs-youre-doing-devops-wrong.html>. Citado 2 vezes nas páginas 12 e 14.

CHACON, S.; STRAUB, B. *Pro git*. [S.l.]: Apress, 2014. Citado na página 23.

CHEF. *Chef - Code Can | Chef*. 2008. Disponível em: <https://www.chef.io/>. Citado 2 vezes nas páginas 9 e 18.

CHEF. *About Ohai*. 2016. Disponível em: <https://docs.chef.io/ohai.html>. Citado 2 vezes nas páginas 20 e 24.

CHEF. *All about Chef*. 2016. Disponível em: <https://docs.chef.io>. Citado 8 vezes nas páginas 18, 19, 26, 42, 43, 44, 45 e 46.

CHELIMSKY, D. et al. *The RSpec book: Behaviour driven development with Rspec, Cucumber, and friends*. [S.l.]: Pragmatic Bookshelf, 2010. Citado na página 24.

CHO, K. C. S. J. et al. Efficient monitoring algorithm for fast news alert. *IEEE Trans. Knowledge and Data Engineering*, p. 950–961, 2007. Citado na página 17.

DEURSEN, A. V.; KLINT, P.; VISSER, J. Domain-specific languages: An annotated bibliography. *Sigplan Notices*, v. 35, n. 6, p. 26–36, 2000. Citado na página 17.

DEVSTRUCTURE. *Blueprint*. 2011. Disponível em: <http://devstructure.com/blueprint/>. Citado na página 20.

DRIESSEN, V. *A successful Git branching model*. 2010. Disponível em: <http://nvie.com/posts/a-successful-git-branching-model/>. Citado na página 29.

ERICH, F.; AMRIT, C.; DANEVA, M. A mapping study on cooperation between information system development and operations. In: *Product-Focused Software Process Improvement*. [S.l.]: Springer, 2014. p. 277–280. Citado na página 12.

FADEL, A. C.; SILVEIRA, H. d. M. Metodologias ágeis no contexto de desenvolvimento de software: Xp, scrum e lean. *Monografia do Curso de Mestrado FT-027-Gestão de Projetos e Qualidade da Faculdade de Tecnologia–UNICAMP*, 2010. Citado 2 vezes nas páginas 27 e 28.

FOWLER, M.; FOEMMEL, M. Continuous integration. *Thought-Works) http://www. thoughtworks. com/Continuous Integration. pdf*, p. 122, 2006. Citado na página 14.

GITHUB. *About GitHub*. 2016. Disponível em: <https://github.com/personal>. Citado na página 23.

GUTIERREZ, C. S. et al. Engenahria de requisitos na metodologia ágil. São Paulo: Universidade Anhembi Morumbi, 2009. Citado 2 vezes nas páginas 27 e 28.

HUMMER, W. et al. Testing idempotence for infrastructure as code. In: *Middleware 2013*. [S.l.]: Springer, 2013. p. 368–388. Citado na página 9.

HUTTERMANN, M. *DevOps for developers*. [S.l.]: Apress, 2012. Citado 3 vezes nas páginas 9, 15 e 16.

LOUKIDES, M. *What is DevOps?* [S.l.]: "O'Reilly Media, Inc.", 2012. Citado na página 12.

OLAUSSON, M.; EHN, J. *Continuous Delivery with Visual Studio ALM 2015*. [S.l.]: Springer, 2016. Citado na página 14.

PRESSMAN, R. S. *Engenharia de software*. [S.l.]: AMGH Editora, 2009. Citado na página 29.

PUPPET. *Facter documentation*. 2016. Disponível em: <https://docs.puppet.com/facter/>. Citado na página 20.

PUPPET. *Puppet - The shortest path to better software*. 2016. Disponível em: <https://www.puppet.com/>. Citado na página 9.

SCRUMREFERENCE. *Scrum Reference*. 2015. Disponível em: <http://scrumreferencecard.com>. Citado na página 13.

SHARMA, R.; SONI, M. *Learning Chef*. [S.l.]: Packt Publishing Ltd, 2015. Citado 8 vezes nas páginas 4, 14, 15, 16, 17, 18, 19 e 26.

SHORE, J. et al. *The art of agile development*. [S.l.]: "O'Reilly Media, Inc.", 2007. Citado na página 13.

THOMAS, D.; HUNT, A. Programming ruby. *Dr. Dobb's Journal*, v. 5, 2001. Citado na página 24.

TRAVIS. *Travis CI*. 2016. Disponível em: <https://travis-ci.com>. Citado na página 23.

VARSILIEV, A. *Cooking Infrastructure by Chef*. [S.l.]: leopard.in.ua, 2014. Citado na página 9.