



**UNIVERSIDADE COMUNITÁRIA DA REGIÃO DE CHAPECÓ
(UNOCHAPECÓ)**

**ÁREA DE CIÊNCIAS EXATAS E AMBIENTAIS
CURSO DE CIÊNCIA DA COMPUTAÇÃO**

**AMBIENTES PADRONIZADOS DE DESENVOLVIMENTO E
MONITORAMENTO DE APLICAÇÕES
BASEADO NA CULTURA DEVOPS**

ALEFE VARIANI

CHAPECÓ, NOVEMBRO DE 2015

**UNIVERSIDADE COMUNITÁRIA DA REGIÃO DE CHAPECÓ
(UNOCHAPECÓ)
ÁREA DE CIÊNCIAS EXATAS E AMBIENTAIS
CURSO DE CIÊNCIA DA COMPUTAÇÃO**

**AMBIENTES PADRONIZADOS DE DESENVOLVIMENTO E
MONITORAMENTO DE APLICAÇÕES
BASEADO NA CULTURA DEVOPS**

**Relatório do Trabalho de Conclusão de
Curso submetido à Universidade
Comunitária da Região de Chapecó
para obtenção do título de bacharelado
no curso de Ciência da Computação.**

ALEFE VARIANI

Orientador: Prof. Cezar Júnior de Souza

CHAPECÓ, NOVEMBRO DE 2015

**AMBIENTES PADRONIZADOS DE DESENVOLVIMENTO E
MONITORAMENTO DE APLICAÇÕES
BASEADO NA CULTURA DEVOPS**

ALEFE VARIANI

**ESTE RELATÓRIO, DO TRABALHO DE CONCLUSÃO DE CURSO,
FOI JULGADO ADEQUADO PARA OBTENÇÃO DO TÍTULO DE:**

BACHAREL EM CIÊNCIA DA COMPUTAÇÃO

Prof. Cezar Júnior de Souza

Orientador

BANCA EXAMINADORA:

Prof. Jorge Antônio Di Domênico

Unochapecó

Prof. Radamés Pereira

Unochapecó

Prof. Sandro Silva de Oliveira

Supervisor de TCC

Prof. Viviane Duarte

Coordenadora de Curso

Chapecó , 25 de Novembro de 2015

Dedico...

Especialmente aos meus pais, Elidio Variani e Anadilce Sgarbossa Variani, por me apoiarem desde o início da graduação até agora, me mantendo sempre de cabeça erguida e respeitando todos.

Agradeço...

A minha família por me ajudarem nos momentos difíceis.

Ao professor Cezar Júnior de Souza, por aceitar o convite de ser meu orientador e pelo apoio da melhor forma para realização do trabalho.

Aos professores Jorge Antônio Di Domênico e Radamés Pereira por aceitarem fazer parte da banca avaliadora deste trabalho e a todos que de alguma maneira contribuíram para o trabalho.

LISTA DE ILUSTRAÇÕES

Figura 1: Representação do sistema operacional como interface entre usuários e recursos.....	19
Figura 2: Percentual de utilização dos SO mais usados na atualidade.....	20
Figura 3: Linguagens mais populares da atualidade.....	21
Figura 4: Representação controle de versão centralizado.....	23
Figura 5: Representação controle de versão distribuído.....	23
Figura 6: Representação do funcionamento de uma equipe.....	33
Figura 7: Representação dos dois personagens do DevOps.....	39
Figura 8: Fatores necessários para a cultura DevOps.....	39
Figura 9: Metodologias e pensamentos formados em 2009.....	42
Figura 10: Ilustração das edições passadas e próximas do DevOpsDay.....	43
Figura 11: Representação do local onde o DevOps atua.....	47
Figura 12: Ilustração do papel do líder da equipe com a cultura DevOps.....	49
Figura 13: Logo Vagrant.....	51
Figura 14: Logo Docker.....	54
Figura 15: Comandos fornecidos pelo Docker.....	55
Figura 16: Logo Puppet.....	56
Figura 17: Logo Chef.....	57
Figura 18: Logo Composer.....	59
Figura 19: Representação do arquivo composer.json.....	60
Figura 20: Logo Bundler.....	61
Figura 21: Representação do arquivo Gemfile.....	61
Figura 22: Logo Maven.....	62
Figura 23: Logo New Relic.....	63
Figura 24: Ilustração da interface da ferramenta New Relic.....	64
Figura 25: Logo Nagios.....	65
Figura 26: Representação da interface de administração do Nagios.....	66
Figura 27: Dados questão 1.....	72
Figura 28: Dados questão 4.....	73

Figura 29: Dados questão 11.....	77
Figura 30: Dados questão 12.....	78
Figura 31: Dados questão 15.....	79
Figura 32: Dados questão 16.....	80
Figura 33: Dados questão 17.....	81
Figura 34: Tela de cadastro New Relic.....	88
Figura 35: Lista de APM New Relic.....	89
Figura 36: Lista de agent oferecidos.....	89
Figura 37: Tela inicial PHP agent.....	90
Figura 38: Chave de licença New Relic.....	90
Figura 39: Opções de ambientes New Relic.....	91
Figura 40: Visão inicial PHP Application New Relic.....	92
Figura 41: Exemplo hosts New Relic.....	93
Figura 42: Exemplo endereços hosts New Relic.....	94
Figura 43: Tela monitoring overview New Relic.....	94
Figura 44: Tela 2 monitoring overview New Relic.....	95
Figura 45: Tela monitoring service maps New Relic.....	95
Figura 46: Tela monitoring transactions New Relic.....	96
Figura 47: Tela monitoring throughput New Relic.....	96
Figura 48: Tela monitoring databases New Relic.....	97
Figura 49: Tela 2 monitoring databases New Relic.....	97
Figura 50: Tela InfoHab New Relic.....	98
Figura 51: Tela Infohab transações Web.....	99
Figura 52: Tela consultas mais demoradas InfoHab.....	99
Figura 53: Service Maps InfoHab.....	100
Figura 54: Tela transactions InfoHab.....	100
Figura 55: Tela Databases InfoHab.....	101
Figura 56: Tela Databases operações mais feitas InfoHab.....	101
Figura 57: Tela Databases sqls das consultas InfoHab.....	102
Figura 58: Tela Error InfoHab.....	103
Figura 59: Tela Alerts InfoHab.....	103
Figura 60: Tela Browser InfoHab.....	104

Figura 61: Tela login cadastro Docker.....	108
Figura 62: Tela inicial Docker.....	108
Figura 63: Tela de criação repositório Docker.....	109
Figura 64: Tela repositório teste Docker.....	109
Tabela 1: Apresentação dos principais padrões que suportam o DevOps.....	45
Tabela 2: Principais comandos do Vagrant.....	53
Tabela 3: Principais componentes do Puppet.....	57
Tabela 4: Principais componentes do Chef.....	58
Tabela 5: Diferenças do ambientes CRS e DTI da Unochapecó.....	105
Tabela 6: Versões dos recursos da máquina criada com o Docker.....	105
Quadro 1: Comando pegar chave New Relic.....	91
Quadro 2: Comando adicionar repositório New Relic.....	91
Quadro 3: Comando instalação New Relic.....	91
Quadro 4: Comando adicionar chave licença New Relic.....	92
Quadro 5: Comando reiniciar New Relic.....	92
Quadro 6: Comando reiniciar apache.....	92
Quadro 7: Comando abrir arquivo configuração New Relic.....	92
Quadro 8: Comando para atualizar pacotes.....	106
Quadro 9: Comando para instalação Docker.....	106
Quadro 10: Comando configurar Docker para inicialização.....	106
Quadro 11: Comando para reiniciar serviços.....	106
Quadro 12: Comando para verificar se Docker está rodando.....	106
Quadro 13: Comando para baixar repositório crs-uno-server.....	107
Quadro 14: Comando lista container Docker.....	107
Quadro 15: Comando lista images Docker.....	107
Quadro 16: Comando para rodar e acessar shell container crs-uno-server.....	107
Quadro 17: Comando criar pasta e arquivo Dockerfile.....	110
Quadro 18: Exemplo arquivo Dockerfile.....	110
Quadro 19: Executar arquivo Dockerfile.....	110
Quadro 20: Comando listar último container.....	111

Quadro 21: Comando fazer commit container Docker.....	111
Quadro 22: Comando enviar imagem Docker.....	111

LISTA DE SIGLAS

APM	<i>Application Performance Monitoring</i>
CPU	<i>Central processing unit</i>
CRS	Centro de Residencia em <i>Software</i>
DBA	<i>Database administrator</i>
DNS	<i>Domain Name System</i>
DTI	Diretoria de Tecnologia da Informação
ERP	<i>Enterprise Resource Planning</i>
HTTP	<i>Hypertext Transfer Protocol</i>
LTS	<i>Long Term Support</i>
NS	<i>Rapid Application Development</i>
NTP	<i>Network Time Protocol</i>
OMSA	<i>Dell OpenManage Server Administrator</i>
PHP	<i>Hypertext Preprocessor</i>
RAD	<i>Rapid Application Development</i>
RAM	<i>Random Access Memory</i>
SDK	<i>Software Development Kit</i>
SLA	<i>Service Level Agreement</i>
SO	Sistema Operacional
SSH	<i>Secure Shell</i>
SSL	<i>Secure Socket Layer</i>
TDD	<i>Test Driven Development</i>
TI	Tecnologia da Informação
TOC	<i>Theory of Constraints</i>
IBM	<i>International Business Machines</i>
IDE	Ambientes de desenvolvimento integrado

RESUMO

Com o avanço da tecnologia em *software*, aplicativos e outros meios digitais que buscam ajudar usuários em processos ou tarefas tem se tornado parte fundamental no dia a dia. Sistemas quando param de funcionar ou ficam fora do ar, horas de trabalho são perdidas por pessoas ou empresas que as utilizam, devido a isso é importante investir na qualidade e estabilidade do *software*.

Com a chegada dos métodos ágeis isso melhorou bastante, permitiu que *software* funcionais sejam entregues mais rápido para os clientes, mas isso acabou criando divisões de responsabilidades, criando uma distância de quem cria ou desenvolve novos produtos, de quem mantêm e monitora os produtos para que continuem funcionando de maneira estável.

DevOps é uma cultura que visa diminuir esta distância entre a equipe de operações e a equipe de desenvolvimento, este trabalho tem como objetivo levantar os conceitos e ferramenta do DevOps, analisando as melhorias que podem trazer ao ambiente de desenvolvimento de *software* e aplicar em um ambiente real de desenvolvimento.

Palavras-Chaves: Qualidade e estabilidade do *software*, Métodos ágeis, Equipe de operações, Equipe de desenvolvimento e DevOps.

ABSTRACT

With the advancement in software technology, applications and other digital seeking help users in processes or tasks has become an essential part in everyday life. When systems stop working or get off the air, working hours are lost by people or companies that use them, because it is important to invest in the quality and stability of the software.

With the arrival of agile methods that improved, allowed functional software to be delivered faster to customers, but that ended up creating divisions of responsibilities, creating a distance who creates or develops new products, who maintain and monitor the products to which continue to function stably.

DevOps is a culture that aims to reduce this distance between the operations team and the development team, this work aims to raise the concepts and DevOps tool, analyzing the improvements it can bring to software development environment and apply in an environment real development.

Keywords: Software quality and stability, Agile, Operations team, Development team and DevOps.

SUMÁRIO

LISTA DE ILUSTRAÇÕES.....	VI
LISTA DE SIGLAS.....	VII
RESUMO.....	VIII
ABSTRACT.....	IX
1. INTRODUÇÃO.....	8
1.1 Contextualização.....	8
1.2 Delimitação do Problema.....	10
1.3 Justificativa.....	12
1.4 Objetivos.....	14
1.4.1 Objetivo Geral.....	14
1.4.2 Objetivos Específicos.....	14
1.5 Procedimentos Metodológicos.....	15
1.6 Estrutura do Trabalho.....	16
2. APRESENTAÇÃO DOS DIFERENTES AMBIENTES DE DESENVOLVIMENTO DE SOFTWARE.....	17
2.1 Ambiente de Desenvolvimento.....	18
2.2 Ambiente de Produção.....	24
2.3 Ambiente de Operações.....	25
2.4 Considerações Finais.....	28
3. MONITORAMENTO DE APLICAÇÕES.....	28
3.1 Considerações Finais.....	32
4. PROBLEMAS ENTRE EQUIPES DE DESENVOLVIMENTO E EQUIPES DE OPERAÇÕES.....	33
4.1 Equipes de Desenvolvimento.....	34
4.2 Equipes de Operações.....	35
4.3 Problemas entre as Equipes.....	36
4.4 Considerações Finais.....	38

5. DEVOPS.....	38
5.1 Introdução.....	38
5.2 Como surgiu.....	41
5.3 Conceitos.....	44
5.4 Ferramentas.....	50
5.4.1 Gerenciamento do Ambiente de Desenvolvimento.....	51
5.4.2 Gerenciamento de Configurações.....	56
5.4.3 Gerenciamento das Configurações da Aplicação.....	59
5.4.4 Monitoramento da Aplicação.....	63
5.5 Considerações Finais.....	67
6. LEVANTAMENTO E ANÁLISE DE DADOS SOBRE DEVOPS NO CRS/DTI UNOCHAPECÓ.....	68
6.1 Aplicação do questionário entre CRS e DTI da Unochapecó.....	69
6.2 Aplicação do questionário para empresas de desenvolvimento da região do Oeste de Santa Catarina sobre DevOps.....	71
6.3 Identificação dos problemas enfrentados entre CRS e DTI da Unochapecó.....	83
6.4 Considerações Finais.....	84
7. APLICAÇÃO DO DEVOPS NO CRS/DTI DA UNOCHAPECÓ.....	85
7.1 Aplicação dos conceitos para CRS e DTI da Unochapecó sobre o DevOps.....	85
7.2 Aplicação da ferramenta de Monitoramento <i>New Relic</i> no CRS.....	87
7.3 Aplicação da ferramenta <i>Docker</i> para padronização dos ambientes.....	105
7.4 Considerações Finais.....	111
8. LEVANTAMENTO DOS RESULTADOS CONQUISTADOS.....	112
8.1 Considerações Finais.....	114
9. CONCLUSÕES.....	114
9.1 Considerações Finais.....	114
9.2 Trabalhos Futuros.....	115
REFERÊNCIAS BIBLIOGRÁFICAS.....	116

1. INTRODUÇÃO

1.1 Contextualização

Atualmente com avanço da tecnologia em *software*, aplicativos, programas de computadores, que visam ajudar os usuários em algum processo ou tarefa da sua vida, tem se tornado parte fundamental do dia a dia de muitas pessoas e empresas que as utilizam. Quando um desses sistemas para de funcionar ou fica fora do ar por algum motivo, além dos usuários ficarem insatisfeitos por não conseguirem realizar suas tarefas, empresas responsáveis pelo produto acabam perdendo dinheiro, negócios e clientes. Por esse motivo é importante investir na qualidade e estabilidade do *software*, desde do momento que se inicia a desenvolver até o momento que está disponível para os usuários (SATO, 2013).

O desenvolvimento de *software* pode ser definido por um ciclo de vida, passando pelo levantamento e análise de requisitos, onde é compreendido o problema, levantado as necessidades, validado os requisitos que serão feitos e definido o investimento parcial do projeto de *software*, depois a definição e descrição do projeto, onde será definido pelos envolvidos no desenvolvimento do *software*, quais recursos ou ferramentas serão usadas e quantas etapas serão necessárias para cumprir os requisitos. Implementação, onde são codificados os problemas em forma de código, para tornar o *software* executável. Testes, momento onde é verificado e validado o produto, para analisar se cumpre o que era esperado. Implantação e Manutenção, a implantação é a disponibilização do *software* ao cliente, sendo feita a instalação no ambiente de produção, já a manutenção é um processo para que caso ocorra erros no *software* a equipe de desenvolvimento ofereça suporte para a manutenção do produto (ENGHOLM, 2010).

Estes processos citados acima evoluíram com o surgimento de Metodologias de desenvolvimento de software¹. Dentre as metodologias, há Métodos Ágeis de desenvolvimento de *software* que surgiram no final da década de 90, propondo uma nova abordagem para organizar tais atividades onde acontecem entregas e interações com prazos mais curtos. Abrindo espaço pra as equipes decidirem junto como o cliente a próxima etapa a ser desenvolvida

¹ Maneira de utilizar um conjunto de métodos (sequência de passos para realizar uma tarefa) para atingir um objetivo, visando à qualidade e produtividade dos projetos (REZENDE, 2002).

(SATO, 2013).

Às metodologias Ágeis tornaram a construção de *software* mais eficiente, mesmo com equipes pequenas. Mas a partir do momento em que o cliente decide que o *software* está pronto para ser implantado ou mandado pra produção e ser usado realmente, diversas outras preocupações se tornam relevantes: suporte, monitoramento, segurança, disponibilidade, desempenho, usabilidade, dentre outras. E neste momento as equipes responsáveis precisam estar preparadas para resolver falhas ou problemas, mantendo o *software* rodando de forma estável (SATO, 2013).

Devido à isso, empresas de TI (Tecnologia da Informação), dividem as responsabilidades entre equipe de desenvolvimento (responsável por criar novos produtos e aplicações e prestar manutenção nos produtos já desenvolvidos) e equipe de operações (responsável por monitorar estes produtos e aplicações mantendo-os em produção), isso acaba causando conflitos entre as equipes, pois enquanto a equipe de desenvolvimento é incentivada a introduzir mudanças, a equipe de operações preza pela estabilidade. Trazendo assim ambientes de desenvolvimento diferentes do ambiente de produção e o monitoramento do *software* se torna complicado (SATO, 2013).

Diante destes fatores e problemas encontrados no desenvolvimento de *software* apareceu a cultura DevOps, que surgiu a pouco tempo e tem como objetivo acabar com divisões de áreas mantendo uma relação saudável, com colaboração e compartilhamento, buscando sempre automação de qualquer processo envolvido (CARVALHO, 2013).

1.2 Delimitação do Problema

Vivemos em uma sociedade hiperconectada, com *smartphones*, *tablets* e outros dispositivos móveis fazendo parte do nosso dia a dia. A velocidade das transformações é alucinante e aumenta constantemente. Nesta nova sociedade digital as altas demandas para um curto espaço de tempo, fazem com que empresas líderes não tenham sua sobrevivência garantida. Com a crescente digitalização da sociedade, todos setores de negócio passam a ser afetados em maior ou menor grau pela tecnologia digital (TAURION, 2013).

Tendo em vista isto, setores de TI não estão tendo boa cotação como inovadores, os atuais processos de desenvolvimento e gestão do ciclo de vida das aplicações, foi baseado no conceito de sistema estáticos e as modificações eram acumuladas e embutidas para novas versões do *software* de no mínimo um ano entre estas mudanças. Este modelo funcionou nos últimos anos, mas começa a dar claros sinais de esgotamento. O mercado demanda mudanças urgentes e mostraram que o fator essencial é proporcionar uma boa experiência aos usuários. E esta busca por melhores experiências deve ser contínuas. Mantendo uma evolução em intervalos cada vez mais curtos. Atualizações não são mais anuais, mas sim mensais, semanais ou mesmo diárias (TAURION, 2013).

Para alcançar estas mudanças as equipes de desenvolvedores adotaram os métodos ágeis para desenvolvimento. Mas esta aceleração no processo de desenvolvimento não alcançou todo valor da aplicação, que vai do desenvolvimento ao usuário final (TAURION, 2013). Com o surgimento dos conceitos do DevOps, a cultura propõe auxiliar a solucionar alguns problemas, como cita Sato (2013).

- Área de desenvolvimento e infraestrutura trabalhando separadamente. Esta divisão é bem clara em departamentos de TI, onde existe a equipe de desenvolvimento (dev), responsável pela criação de novos produtos e aplicações, incluir novas funcionalidades e correções de problemas(*bugs*), e a equipe de operações (infra) responsável por manter segurança e estabilidade em aplicações em produção. Está distância entre dev e infra dificulta a comunicação e entendimento de ambas as partes, onde causam conflitos na hora de implantar novas funcionalidades da aplicação. Diminuir ou acabar com está distância é um dos principais objetivos da cultura/movimento do DevOps;
- Ambientes de desenvolvimento diferentes do ambiente de produção: Inúmeras

empresas sofrem com ambientes de desenvolvimento e produção diferentes, onde causam problemas na hora de fazer a liberação da nova versão do *software* (*deploy*);

- Mal monitoramento da aplicação (desempenho, performance, consultas e erros não tratados). Aplicações possuem pontos críticos que são difíceis de serem identificados, como o usuário usa a aplicação, desempenho das transações com o banco de dados e também a verificação quando alguma disponibilidade da aplicação para de funcionar, entre outros;

Segundo Sete (2015), o status da indústria de *software* hoje é que os projetos vão atrasar, quando entregues (se entregues), não vão se comportar bem no ambiente de produção, que apresentarão problemas de performance e não corresponderão às expectativas dos patrocinadores, isto por que existe uma grande quantidade de fatores envolvidos no desenvolvimento de *software*, além dos problemas citados acima, que muitas vezes não são esperados pelas equipes envolvidas.

Sabe-se que para reagir rapidamente às necessidades dos clientes e atingir uma velocidade mais rápida para o mercado é uma prioridade. Esse tipo de agilidade requer um ciclo contínuo de liberação e ajuste, e ficar a par disso depende do quanto os departamentos de desenvolvimento e operações podem colaborar. Alcançar a colaboração necessária nem sempre é fácil. Mas é devido à necessidade e aos esforços para obter melhorias nos processos de desenvolvimento de *software* que o movimento DevOps surgiu (RACKSPACE, 2015).

1.3 Justificativa

Entregar um *software* em produção se torna cada dia um processo mais difícil, ciclos longos para entregas e para testes, divisão entre equipes de desenvolvimento e operações, aplicações sem monitoramento e ambientes de desenvolvimento e produção diferentes, são fatores para este problema. Mesmo equipes ágeis produzindo *software* entregáveis ao final de cada iteração, ainda sofrem para chegar em produção quando encontram estas barreiras (SATO, 2013).

Embora tenham surgido várias metodologias ágeis de desenvolvimento de *software*, e muitas empresas aderiram à uma metodologia ágil, desenvolvendo produtos mais rapidamente, as empresas ainda encontram problemas de monitoramento de suas aplicações, dificuldade em manter o ambiente de desenvolvimento idêntico ao ambiente de produção e realizar *deploy* de maneira fácil e com menos chances de erros no ambiente de produção (SATO, 2013).

Sendo assim, o DevOps é um movimento cultural e profissional que tenta quebrar esta barreira, mostrando que desenvolvedores e engenheiros de sistema têm muito o que aprender uns com os outros (SATO, 2013). Segundo Sete (2015) este movimento se propõem a resolver alguns itens, como:

- Medos de mudanças, quando uma versão de *software* finalmente é entregue, equipes de operações acreditam que *software* e ambientes são coisas frágeis e vulneráveis, que qualquer mínima interferência é suficiente para quebrar a aplicação e tirá-la do ar. Criando assim um processo caro e burocrático para gestão de mudanças e então um longo e doloroso processo deve ser percorrido para que seja possível atualizar versões das aplicações ou mesmo corrigir erros;
- Liberações de versões (*Deployments*) arriscadas, desenvolvedores na maioria de vezes não sabem como os códigos se comportam no ambiente de produção, se está funcionando como o esperado, se vai suportar o volume de acesso. Estas respostas dos pontos citados demoram para chegar ao desenvolvedor ou às vezes nem chegam, acontecendo assim uma nova liberação de versão para o ambiente de produção, aguardando ou observando se ele vai continuar de pé;
- “Funciona na minha máquina” expressão bastante comum, são problemas que se manifestam apenas no ambiente de produção. Estes problemas normalmente são identificados pela equipe de operações, quando os próprios usuários descobrem e

reportam os problemas. Após análise, o problema é informado aos desenvolvedores, que verificam e acabam dizendo “na minha máquina funciona”. Entretanto não são analisadas as diferenças entre o ambiente de desenvolvimento do ambiente de produção e os vários fatores importantes diante disso, SO (sistema operacional), versões e arquivos de configuração, são alguns deles;

- Silos², geralmente nas empresas de TI as equipes são divididas por desenvolvedores e operações ou outras equipes que existam. Isso causa desperdício, podendo levar à uma cultura onde problemas são transferidos entre as equipes. É comum que estes silos não estejam localizados fisicamente dentro do mesmo escritório, na mesma cidade ou ainda no mesmo país, tornado uma mentalidade entre os times de desconfiança, suspeita, criando um ambiente não agradável entre às áreas;
- Foco e Visão, existe claramente um déficit de visão compartilhada entre os responsáveis por construir e operar aplicações. Enquanto as equipes de desenvolvedores se convencem que os códigos estão ótimos, as equipes de operações se isolam, cuidando dos ambientes e combatendo tudo e todos que podem de alguma forma colocar em risco sua estabilidade;

Kevin Behr, autor e diretor de área tecnológica da *Praxis Flow*, explica "Você não pode comprar DevOps numa caixa" (RACKSPACE), Peter Drucker representa bem o maior desafio de uma iniciativa de DevOps. “Infelizmente, não se pode fazer download da cultura. Mudança de comportamento e foco nas pessoas é o que vai mudar a cultura das organizações”. O grande ponto do DevOps é o modelo mental que preconiza que todos estão do mesmo lado, tentando a mesma coisa: entregar continuamente *software* de valor, capaz de transformar a vida das pessoas (SETE, 2015).

2 Silos – são considerados os vários departamentos que a empresa de TI pode ter, onde cada um mantém suas funcionalidades, sem haver interações umas com as outras.

1.4 Objetivos

1.4.1 Objetivo Geral

Levantar ferramentas e práticas que utilizam os conceitos do DevOps, analisar às melhorias que o DevOps pode trazer para o ambiente de desenvolvimento de *software* e aplicá-la no Centro de Residencia em *Software*, para verificar se a abordagem realmente traz benefícios aos envolvidos no processo de desenvolvimento de *software*.

1.4.2 Objetivos Específicos

- Conhecer os processos do desenvolvimento de *software*;
- Entender conceitos da metodologia que se aplicam ao ambiente DevOps;
- Melhorar o ambiente de desenvolvimento e monitoramento da aplicação através da cultura do DevOps;
- Aplicar uma proposta em um ambiente desenvolvimento real (CRS e DTI da Unochapecó), utilizando conceitos e práticas do DevOps;
- Analisar o comportamento, vantagens e desvantagens que a proposta traz ao ambiente de desenvolvimento;

1.5 Procedimentos Metodológicos

Este projeto tem o objetivo de aplicar conceitos e práticas que a cultura DevOps propõem para ambientes de desenvolvimento de um *software* no Centro de Residência em *Software* (CRS) – Unochapecó, projeto que oferece estágios aos estudantes dos cursos de Ciência da Computação e de Sistemas de Informação para adquirir experiências de ordem prática, associadas aos conhecimentos teóricos, com enfoque nos ciclos de desenvolvimento de *software*.

Para isto, faz-se necessária pesquisa de levantamento de dados, através de uma questionário sobre o atual funcionamento entre os profissionais de desenvolvimento e operações envolvidos no processo de construção do *software* entre o CRS e a DTI da Unochapecó. Para análise e interpretação de quais os problemas que os envolvidos enfrentam com mais frequência, onde dificultam ou travam suas tarefas, visando automatizar ou melhorar qual for o processo através da elaboração de etapas, previamente definida das necessidade das equipes. Os procedimentos utilizados no projeto são por pesquisa bibliográfica, a partir de livros, artigos e materiais disponibilizados na *Internet*.

Sendo assim, utilizando conceitos impostos pelo movimento cultural do DevOps por meio de mudança de cultura entre os envolvidos ou por ferramentas que auxiliam na automação de processos difíceis de manter ou de serem feitos. Buscando relatar os resultados conquistados, analisando os ganhos, para posteriormente propor novas etapas.

1.6 Estrutura do Trabalho

O trabalho constitui-se de 9 capítulos, sendo que nos 5 primeiros capítulos são de pesquisas e entendimentos dos assuntos envolvidos e a partir do sexto capítulo inicia-se a aplicação do trabalho em um ambiente real. Então, no primeiro capítulo **Introdução**, é apresentado o contexto geral do trabalho, as delimitações do problema, a justificativa onde apresenta o por que do trabalho, em seguida os objetivos e os procedimentos metodológicos.

No segundo capítulo **Apresentação dos diferentes ambientes de desenvolvimento de software** levanta-se os ambientes envolvidos, que são o ambiente de desenvolvimento, ambiente de produção (onde o usuário tem acesso) e ambiente de operações (onde o pessoal do suporte acessa fazendo *build*, *deploy*, backups e outros), em seguida, no terceiro capítulo **Monitoramento de Aplicações**, apresenta-se como é importante ter um bom monitoramento da aplicação. No quarto capítulo **Problemas entre equipes de desenvolvimento e equipes de operações**, é abordado sobre as equipes e seus principais problemas que existem entre elas.

No capítulo 5 **DevOps**, é estudado a abordagem sobre o movimento cultural DevOps, mostrando os conceitos, os pilares principais do DevOps sendo eles, cultura, automação, medição/avaliação e compartilhamento. Além disso, foi destacado diversas ferramentas utilizadas para alcançar alguns objetivos impostos pelo DevOps.

Em seguida, inicia-se a parte de aplicação do trabalho, no capítulo 6 **Levantamento e análise de dados sobre DevOps no CRS/DTI Unochapecó**, onde é levantado a situação atual entre o CRS e DTI da Unochapecó como está funcionando e quais são as principais dificuldades e problemas enfrentados que podem ser melhorados e também um pesquisa sobre os atual entendimento sobre DevOps para empresas de desenvolvimento da região do Oeste de Santa Catarina. No sétimo capítulo **Aplicação do DevOps no CRS/DTI da Unochapecó**, é mostrado as etapas de aplicações feitas no CRS e DTI da Unochapecó e os capítulos 8 (**levantamento dos resultados conquistados**) e 9 (**conclusões**), são para o levantamento dos dados conquistados, conclusões finais e trabalhos futuros.

2. APRESENTAÇÃO DOS DIFERENTES AMBIENTES DE DESENVOLVIMENTO DE *SOFTWARE*

Quando se começa a desenvolver um *software* é preciso a criação de um cenário que contenha os componentes e ferramentas necessários para a compilação ou execução do *software*. Um ambiente reúne todos estes itens e conjuntos, visando melhorar os processos de desenvolvimento.

Definir um ambiente é importante, pois possibilita a definição do que está envolvido na administração do ambiente. De modo que se refere ao conjunto de processos, arquivos e variáveis do ambiente que compõe uma instalação que roda uma aplicação (MARINESCU, 2004).

Cada vez mais engenheiros de *software* têm sido cobrados para realmente fazerem engenharia do produto de *software*: planejar, acompanhar, executar e controlar. Cresce, então, a necessidade de ferramentas para apoiar estas tarefas... Neste contexto, é crescente a demanda por Ambientes de Desenvolvimento de *Software*, que buscam combinar técnicas, métodos e ferramentas para apoiar o Engenheiro de *Software* na construção de produtos de *software*, abrangendo todas as atividades inerentes ao processo, tais como planejamento, gerência, desenvolvimento e controle da qualidade. (MIAN et al, 2001, p. 2).

Nesta dimensão, o controle se torna importante. Um processo eficaz deve considerar as relações entre as atividades, os artefatos produzidos no desenvolvimento, as ferramentas e os procedimentos necessários, a habilidade, o treinamento e a motivação do pessoal envolvido. Neste contexto, os seguintes conceitos se tornam importantes, segundo Falbo (1998):

- **Atividade:** É uma tarefa ou trabalho a ser realizado, requerendo um ou mais recursos para executá-la. Uma atividade pode consumir ou produzir artefatos. Num processo real, as atividades são divididas em sub atividades, que, por sua vez, podem também ser subdivididas. Dependendo da complexidade, uma atividade pode ser subdividida em vários níveis;
- **Artefatos:** São produtos de *software*. Podem ser produzidos ou consumidos por uma atividade. São exemplos de artefatos: documentos de especificação de requisitos, manuais de qualidade, atas de revisão, diagramas de classes e código fonte;

- **Recursos:** São as pessoas da organização, as ferramentas de *software*, equipamentos ou quaisquer outros recursos necessários à execução de uma atividade;
- **Processo:** Um processo de desenvolvimento de *software* é um conjunto organizado de atividades, métodos, ferramentas, procedimentos e técnicas, com o fim de desenvolver e manter um *software*;
- **Projeto:** Consiste em um desenvolvimento de *software*, que emprega vários processo de *software*, onde englobando um conjunto de atividades necessárias, artefatos consumidos e produzidos e recursos utilizados;

No desenvolvimento de um *software* podem ser citados três ambientes: ambiente de desenvolvimento, ambiente de produção e ambiente de operações, no qual será tratado nos itens 2.1, 2.2 e 2.3 a seguir.

2.1 Ambiente de Desenvolvimento

Em todo o processo de produção do *software*, dentre vários que existem, o desenvolvimento é fundamental, nele é envolvido a equipe de desenvolvimento que necessita de um ambiente que lhe permita desenvolver novos produtos ou funcionalidades e dar manutenção para possíveis problemas que ocorram no *software*. Equipes de desenvolvimento buscam criar coisas novas, usar novas bibliotecas e ferramentas, para ter um ambiente ideal para criar o *software*.

Um ambiente de programação é uma coleção de ferramentas usadas no desenvolvimento de *software*. Essa coleção pode consistir em apenas um sistema de arquivos, um editor de textos, um ligador e um compilador, ou pode incluir uma grande coleção de ferramentas integradas, cada uma acessada por meio de uma interface de usuário uniforme, no último caso, o desenvolvimento e a manutenção de *software* é enormemente melhorada. Logo, as características de uma linguagem de programação não são a única medida da capacidade de desenvolver um sistema, segundo SEBESTA (2011).

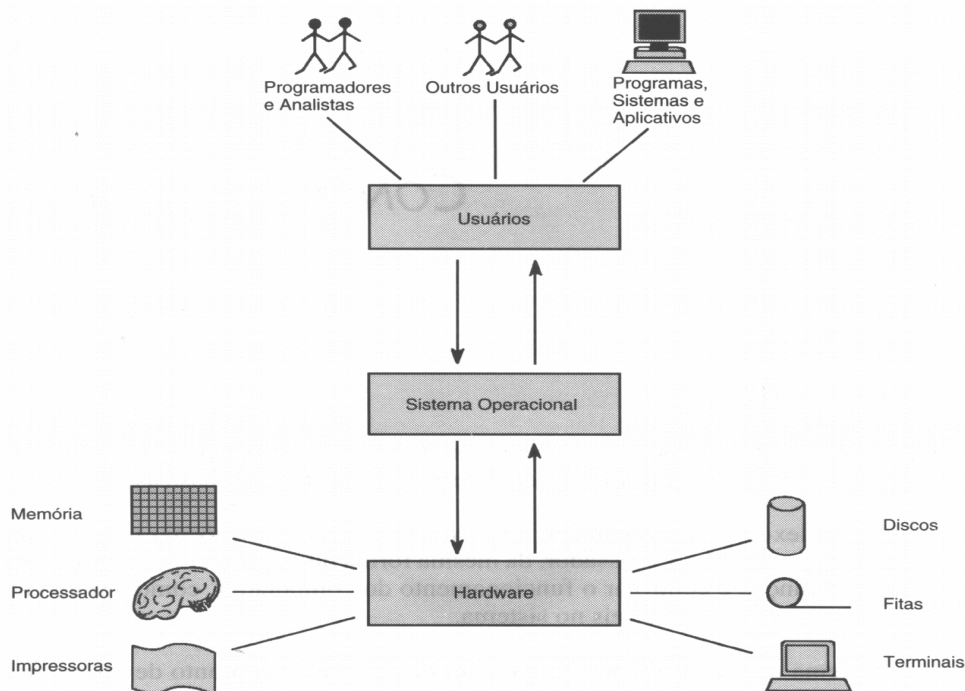
Essas ferramentas foram evoluindo gradativamente, conforme a necessidade de automação foi sendo reconhecida, isto ajudou a aumentar a produtividade da programação por meio de redução das chances de erros (GHEZZI, 1991).

Um ambiente de desenvolvimento pode-se dividir em certas etapas:

- **Sistemas Operacionais:** Um sistema operacional tem a tarefa de controlar todos os recursos do computador e fornece uma base sobre a qual os programas e aplicativos podem ser escritos (TANENBAUM, 2006). É uma camada de *software* que opera entre o *hardware* e os programas aplicativos voltados ao usuário final.

A figura 1 mostra onde o sistema operacional atua:

Figura 1: Representação do sistema operacional como interface entre usuários e recursos.



Fonte: Zanini (2000).

Atualmente os SO (Sistemas Operacionais) mais usados são:

Windows: criado pela *Microsoft* por Paul Allen e Bill Gates, sua primeira versão foi anunciada em 1983, mas demorou um pouco para ser desenvolvido. Em 20 de novembro de 1985, dois anos após o anúncio inicial, a *Microsoft* começa a vender o *Windows*, nome escolhido pois descreve melhor as caixas ou "janelas" de computação que são fundamentais para o sistema. (MICROSOFT, 2015), atualmente é o sistema operacional mais usado no mundo, com porcentagem de 91.22% (NETMARKETSHARE, 2015).

OS X: criado, fabricado e vendido pela *Apple* em 2001, é baseado no *kernel Unix* ³. Possui recursos básicos do *Unix* acessíveis por meio de linha de comando e com uma interface amigável. Hoje já está na versão 10.x.

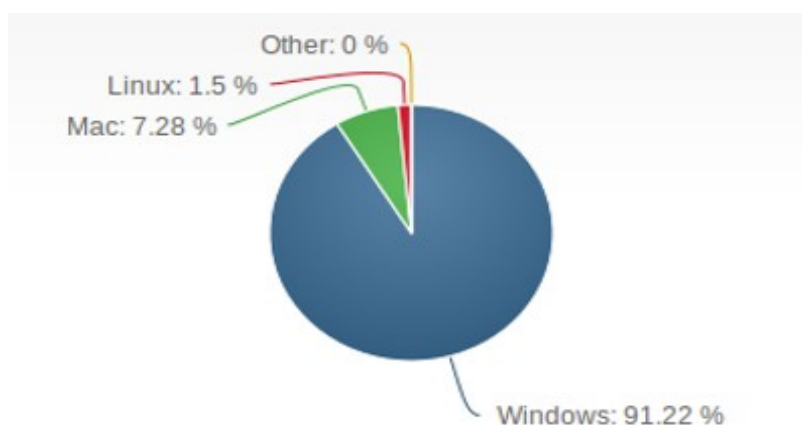
³ *kernel Unix* – *Kernel* é o núcleo do sistema operacional, *Unix* originalmente desenvolvido por Ken Thompson em 1965 é sistema operacional multifuncional (VIVAOLINUX, 2015).

Linux: criado por Linus Torvalds em 1991, é um *software* livre ⁴ e *Open source* ⁵. Este sistema operacional possui diversas distribuições, uma distribuição nada mais é que um *kernel* acrescido de programas escolhidos a dedo pela equipe que a desenvolve.

Cada distribuição possui suas particularidades, tais como forma de se instalar um pacote (ou *software*), interface de instalação do sistema operacional em si, interface gráfica, suporte a *hardware*. Exemplos de distribuições *Linux* são: Ubuntu, Debian, Fedora, CentOS e outros (VIVAOLINUX, 2015).

A figura a seguir é relatado o percentual de utilização dos SO, o que é mais usado na atualidade.

Figura 2: Percentual de utilização dos SO mais usados na atualidade.



Fonte: Netmarketshare (2015).

- Linguagens de Programação: Da mesma forma que as nossas linguagens naturais, as linguagens de programação facilitam a expressão e a comunicação de ideias entre pessoas. Elas também permitem a comunicação de ideias entre computadores e pessoas, mas possuem em domínio de expressão mais reduzido do que as linguagens naturais (TUCKER e NOONAN, 2009).

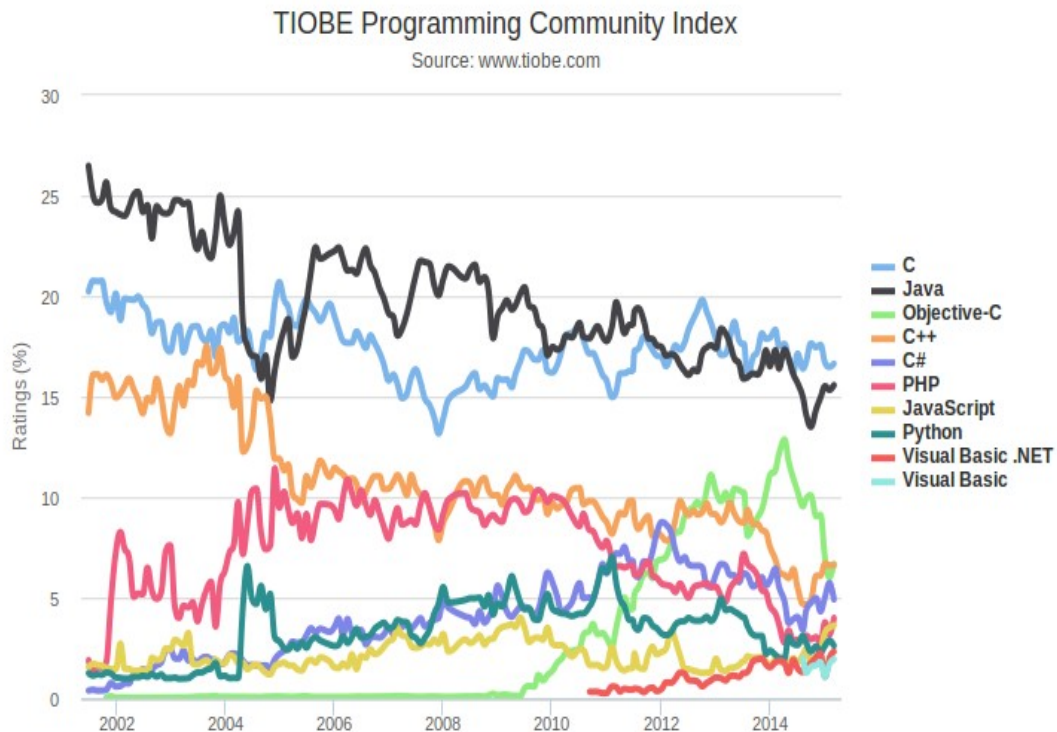
Muitas linguagens de programação surgem até hoje, isso torna a escolha da linguagem ainda mais difícil no desenvolvimento do *software*, isso interfere diretamente nas possíveis ferramentas do ambiente de desenvolvimento, deve-se fazer uma análise detalhada para as linguagens e verificar qual melhor se encaixa nas necessidades do *software*.

Na figura a seguir é mostrado as linguagens mais populares do momento:

⁴ *Software* livre – entende-se aquele *software* que respeita a liberdade e senso de comunidade dos usuários, permitindo os usuários possuem a liberdade de executar, copiar, distribuir, estudar, mudar e melhorar o *software* (VIVAOLINUX, 2015).

⁵ *Open source* – Código aberto, deve permitir distribuição livre e licença, executar, copiar, mudar e melhorar o código, possui conceito derivado e semelhante do *Software* livre (VIVAOLINUX, 2015).

Figura 3: Linguagens mais populares da atualidade.



Fonte: Tiobe (2015).

Linguagens como C, C++ e Java continuam sendo as mais populares e utilizadas, tanto no meio acadêmico como no comercial, mas nos últimos anos tiveram uma pequena queda, já linguagens que vem crescendo e merecem destaque são Objective-C, PHP, JavaScript e Python, ondem permitem criar desde aplicações *Web* até aplicações *Mobile*. Estas linguagens de programação podem ser utilizadas por inúmeras ferramentas, as quais serão citadas no próximo tópico das etapas da definição do Ambiente de desenvolvimento.

Ambientes de desenvolvimento integrado (IDE): Segundo Sebesta (2000), IDE pode ser identificado como um ambiente de desenvolvimento integrado, que reúne características e ferramentas que dão apoio ao desenvolvimento de *software*, com o objetivo de agilizar o processo de desenvolvimento. Geralmente, IDE apresenta a técnica RAD (do inglês *Rapid Application Development*), que consiste em permitir que os desenvolvedores tenham um aproveitamento maior, desenvolvendo códigos com mais rapidez e facilidade. É integrado porque envolve pelo menos, editor, compilador e depurador.

Dentre várias IDEs, podem ser destacadas:

- **Netbeans:** Criado por um grupo de alunos da Tchecoslováquia, atual República Tcheca, em 1996 (NETBEANS, 2015). É um ambiente de desenvolvimento

gratuito e de código aberto, se destaca pelo amplo suporte às mais diversas linguagens de programação. A IDE é executada em muitas plataformas, como *Windows*, *Linux*, *Solaris* e *MacOS* (ORLANDO, 2015);

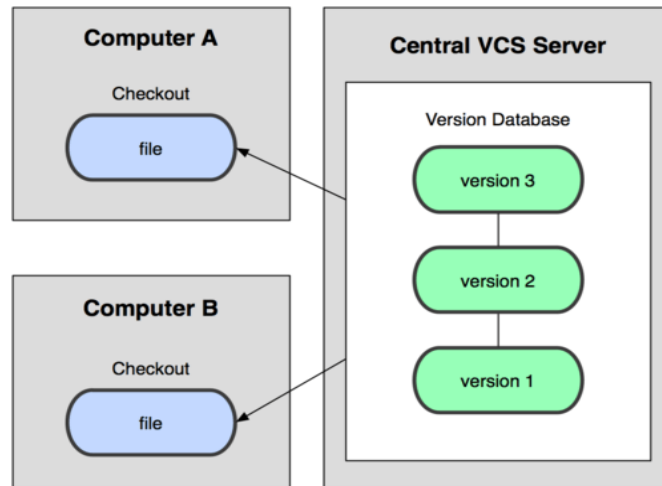
- **Eclipse:** Criado pela IBM (*International Business Machines*) em 2001 (ECLIPSE, 2015), projetado inicialmente para desenvolvimento em Java, hoje o *software* permite que programe em quase todas as linguagens de programação. Além disso, é muito usado para desenvolvimento de aplicativos para Android, pois faz parte do kit de desenvolvimento de *software* (SDK) recomendado pelo Google aos desenvolvedores do sistema operacional (ORLANDO, 2015);

Ambas também são compatíveis com ferramentas de Controles de versão, que serão mostrados mais a diante. Existem IDE para os mais diversos tipos de projetos, deve-se analisar qual é o mais compatível com o projeto em execução.

- **Controle de versão:** O controle de versão é um sistema que registra as mudanças feitas em um arquivo ou um conjunto de arquivos ao longo do tempo, de forma que possa recuperar versões específicas (GIT-SCM, 2015). Permitindo manter históricos das alterações feitas por cada usuário.

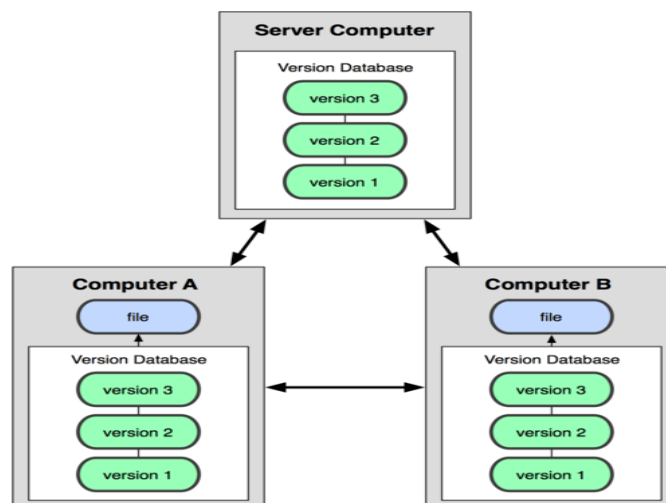
Se destacam o *Git* e o *Subversion*, onde possuem conceitos diferentes, enquanto o *Subversion* é um sistema de controle de versão centralizado, que consiste em um ou mais cliente e um servidor central (SANCHES, 2015), já o *Git* é um sistema de controle distribuído, não possui nenhum servidor centralizado (IBM, 2015). Nas figuras 4 e 5 a seguir mostram como funcionam o controle de versão distribuído e controle de versão centralizado.

Figura 4: Representação controle de versão centralizado.



Fonte: GIT-SCM (2015).

Figura 5: Representação controle de versão distribuído.



Fonte: GIT-SCM (2015).

Neste tópico sobre Ambientes de Desenvolvimento pode-se notar a grande quantidade de ferramentas e configurações que um ambiente pode ter para um desenvolvedor, isso torna a escolha sobre estes determinados artefatos ainda mais difícil, tornando a complexidade de configuração alta. No próximo tópico será apresentado o ambiente de produção, o qual os usuários utilizam.

2.2 Ambiente de Produção

Ao contrário do que muitos processos de desenvolvimento sugerem, o ciclo de vida do *software* só deveria começar quando usuários começam a usá-lo. Pois nenhum *software* entrega valor antes de entrar em produção (SATO, 2013).

A implantação de um novo sistema não está restrita à apenas a tarefa de instalação do *software*, mas também envolve uma série de outras atividades, dentre elas: treinamento, comunicação, verificação e validação do sistema em ambiente de produção. Para implantar um sistema ou componentes em produção deve-se planejar quais as tarefas serão envolvidas em termos de instalação do novo *software*, além de detalhar o *hardware* e o ambiente da instalação. A implantação normalmente envolve preparo de ambiente, instalação, testes e entrega da solução (ENGHOLM, 2010).

Devido a isso, ambientes de produção mantêm sempre um grau de complexidade alto, existem pontos em que aparecem várias dificuldades que podem se tornar em problemas futuros, como:

- Gerenciamento de dependências, conforme o desenvolvimento de uma aplicação vai progredindo e o sistema cresce, é comum também aumentar pacotes e bibliotecas usadas no projeto. Este é um problema comum que diversas comunidades de desenvolvimento têm: na comunidade PHP (*hypertext preprocessor*), o *Composer* é utilizado para baixar pacotes necessários; na comunidade Ruby, o *Bundler* gerencia dependências entre *gems* (bibliotecas sendo um conjunto de arquivos Ruby reusáveis); na comunidade Java, o Maven e o *Ivy* gerenciam *jars* (arquivos compactados de um conjunto de classes Java) e suas dependências. Não manter as dependências do projeto atualizado no ambiente de produção, pode-se tornar em uma dor de cabeça (SATO, 2013);
- Arquivos de configurações, são arquivos onde são guardadas as informações, que podem ser facilmente alteradas sem a necessidade de recompilação da aplicação, ou seja, basta alterar esse arquivo e reiniciar a aplicação que as alterações serão aplicadas. Existem alguns tipos de configurações de máquinas (configurações que se aplicam em toda a máquina), configurações de aplicação (configurações para uma aplicação específica) e configurações de segurança (configurações de hierarquia de grupos e permissões associadas aos níveis de política de segurança)

(AÉCE, 2011). Embora a maioria destes arquivos podem ser mantidos em Controles de versão, é necessário atenção;

- Versões diferentes, aplicações podem ser divididas em várias etapas, estas etapas são definidas por versões (beta, *release*, LTS (*Long Term Support*), final, 1.x, 2.x e assim por diante), no ambiente de produção saber qual versão está sendo usada se torna difícil, pois depende dos *deploy* (implantação/liberações de funcionalidade) feitos pela equipe de operações;

O Ambiente de produção se resume aos processos de execução do produto, no qual os usuários terão acesso diretamente, onde eles podem utilizar o produto de *software* desenvolvido. Mas para isso o Ambiente de produção precisa de um Ambiente de operações que monitora o produto, verificando possíveis falhas ou indisponibilidades que existem no produto. No próximo tópico será falado sobre isso, o último processo antes de os usuários terem acesso ao produto.

2.3 Ambiente de Operações

Como no ambiente de desenvolvimento onde é envolvida a equipe de desenvolvimento (programadores), no Ambiente de operações é envolvida a equipe de infraestrutura ou *sysadmin*. Essa equipe é responsável por manter o ambiente de produção funcionando (CARVALHO, 2013).

Os envolvidos nesta área, sendo eles da infraestrutura ou *sysadmin*, tem papéis importantes, como cita CARVALHO (2013):

- Rodar as aplicações;
- Monitorar o funcionamento e performance;
- Avaliar e propor melhorias;
- Reduzir riscos;
- Preocupação com segurança e estabilidade;
- Fazer *rollbacks* e *Backups*;

Estes profissionais possuem grande importância para as empresas de TI, buscando constantemente melhorias para o ambiente de produção e desenvolvimento, avaliando e monitorando o ambiente para que todos os processos não sofram interrupções que cause atraso. É a equipe de infraestrutura que geralmente interage com o cliente e protege o valor do negócio

(CARVALHO, 2013).

Diante de várias responsabilidades, como foi citado logo acima, o Ambiente de operações envolve alguns processos como:

- **Instalação/Configuração de Servidores:** Um servidor é uma máquina que pode rodar vários serviços e armazenar dados, este serviço pode ser entendido como uma requisição do cliente pelo aplicativo. Existem vários tipos de servidores, como servidores *web*, servidores de arquivos, servidores de banco de dados (MORIMOTO, 2010).

Servidores *web* são a espinha dorsal da Internet, são eles que hospedam todas as páginas, incluindo os mecanismos de busca, servindo como base para todo tipo de aplicativo via *web* (MORIMOTO, 2010). Eles são acessados geralmente pelos navegadores (*browser*) onde recebem requisições HTTP⁶ (*Hypertext Transfer Protocol*), permitindo perguntas ou respostas que os usuários desejam.

Servidores de banco de dados são basicamente o local cuja a finalidade geral é armazenamento de informações de registros, que permite que os usuários busquem e atualizem essas informações quando solicitados, as informações podem ser qualquer coisa que seja do interesse do usuário (DATE, 2004).

O termo provisionamento é usado por equipes de operações para se referir às etapas de preparação iniciais de configuração de um novo recurso. Em servidores estes processos pode variar de empresa para empresa dependendo da infraestrutura e da divisão de responsabilidade entre as equipes, sendo que o processo de provisionamento só acaba quando o servidor e a aplicação estiverem rodando e acessíveis na rede (SATO, 2013). Caso uma empresa possui infraestrutura própria, porém não tem servidores sobrando para uso, as etapas necessárias para colocar uma nova aplicação no ar, como cita SATO, que seria:

- 1) **Compra de *hardware*:** processo envolve diversas justificações e aprovações pois investir dinheiro em *hardware* influi na contabilidade e no planejamento financeiro da empresa.
- 2) **Instalação física do *hardware*:** essa etapa envolve montar o novo servidor no *data center*, assim como instalação de cabos de força, de rede etc.
- 3) **Instalação e configuração do sistema operacional:** Depois do servidor ligado, é preciso instalar um sistema operacional e configurar os itens básicos de *hardware* como: interfaces de rede, armazenamento, autenticação e autorização de usuários.

6 HTTP – implementado para permitir uma transferência rápida e eficiente de documentos na Internet, baseado no paradigma pedido e resposta (*request e response*) (OPENBASE, 2015).

4) Instalação e configuração de serviços comuns: Alguns servidores precisam de configuração de serviços de infraestrutura como: NS (*Rapid Application Development*), NTP (*Network Time Protocol*), SSH (*Secure Shell*), coleta e rotação de *logs*, *backups*, *firewall*, impressão etc.

5) Instalação e configuração da aplicação: é preciso instalar e configurar a aplicação.

O item 5 está ligado ao *Deploy* que será abordado na etapa a seguir, pois cada aplicação pode efetuar *deploy* várias vezes, enquanto o provisionamento de novos servidores acontece com menos frequência.

- **Build e Deploy da aplicação:** *Build* é o processo de compilação, teste e empacotamento da aplicação, as etapas do processo de *build* também podem incluir gerenciamento de dependências, rodar ferramentas de análise estática do código, cobertura de código, geração de documentação, dentre outros. A principal função é gerar um ou mais artefatos, com versões específicas, que se tornam potenciais candidatos para o *deploy* em produção que será falado mais a frente (SATO, 2013). O *build* necessita de um controle de versão, que foi citado no tópico 2.1, e um gerenciador de dependências para que a aplicação possua todos os pacotes e extensões necessárias.

Falando um pouco mais sobre os processos que o *Build* faz, segundo Sato (2013):

- **Resolução de dependências:** Responsável por todas as dependências do projeto e baixar as bibliotecas e *frameworks* necessários para compilar e rodar a aplicação.

- **Compilação:** A aplicação precisa ser compilada para execução verificando erros na hora de rodar a aplicação.

- **Testes automatizados:** Os módulos que são definidos para testes automatizados devem ser executados e gerar relatórios de quais testes passaram e quais falharam.

- **Empacotamento de artefatos:** preparação dos conjuntos a serem empacotados, onde contêm bibliotecas, arquivos estáticos e de configurações necessárias para rodar uma aplicação.

Depois de apresentado um pouco sobre *Build* da aplicação, outro momento importante para aplicação é implantação da aplicação ou lançamento de novas versões, onde é o papel do *Deploy*.

Constantemente de tempos em tempos a equipe de desenvolvimento empacota o *software* para ir para produção, escreve a documentação explicando como configurar o sistema, como realizar a instalação em produção e repassa a responsabilidade para a equipe de operações. Este processo de levar código do desenvolvimento e teste para produção é chamado de *Deploy*, que seria “implantar” (SATO, 2013).

2.4 Considerações Finais

Neste tópico sobre Ambientes abordou-se um pouco sobre ambientes de desenvolvimento, onde levantou-se os recursos, ferramentas e equipes de desenvolvimento, logo depois foi abordado sobre o ambiente de produção, no qual o usuário ou cliente tem acesso e por fim foi descrito sobre ambientes de operações, onde apareceram as principais responsabilidades da equipe de operações.

Pode-se notar a grande quantidade de recursos, ferramentas, responsabilidades das equipes envolvidas para a construção de um *software* ou aplicação, no próximo capítulo, monitoramento de aplicações, será mostrado como manter um acompanhamento da aplicação de maneira mais detalhada.

3. MONITORAMENTO DE APLICAÇÕES

A partir do momento que uma aplicação está em produção, conquista-se uma parte importante do ciclo de vida de qualquer *software*, daqui pra frente é preciso se responsabilizar pela estabilidade do sistema em produção, garantindo que o *software* continue funcionando sem problemas, verificando que o sistema não fique fora do ar. Muitas vezes esperar por reclamações dos usuários não é uma boa opção. Estas responsabilidades são geralmente encarregadas para as equipes de operações, citadas no item 2.3 (SATO, 2013).

Devido à isso, enquanto a aplicação fica inativa por apenas alguns minutos, isso potencialmente pode custar dinheiro. O conceito de Monitorar é observar, analisar, mantendo o acompanhamento de como a aplicação está se comportando e ficando atento aos possíveis sinais de que algo não está normal. Em tecnologia da informação, “não está normal” pode indicar indisponibilidade de um ou mais partes de um sistema ou mesmo uma lentidão ou diminuição na qualidade de serviços percebida pelo cliente (4LINUX, 2015).

Existem vários pontos que o sistema de monitoramento engloba que estão recebendo bastante atenção pela comunidade e ferramentas de monitoramento, que podem ser destacadas segundo Sato (2013):

- Notificações – Um bom monitoramento deve alertá-lo quando algo estiver errado, permitindo que se investigue e arrume o que estiver causando problemas o mais rápido possível;

- Agregação de *logs* – quanto mais servidores possuir, mais difícil é investigar e depurar problemas. *Logs* pode-se entender por informações/mensagem geralmente de erros. A agregação de *logs* é fazer a coleta, filtrar, analisar entradas de *log* de diversas fontes diferentes;
- Métricas – entende-se por medir aplicações, processos, negócios e até pessoas o máximo possível, o DevOps incentiva isso, pois tendo um problema, sabendo onde e quando o problema ocorreu, torna-se mais fácil resolvê-lo;
- Visualizações – quanto mais dados e métricas forem coletados, maior é o risco de sofrer uma sobrecarga de informações. Utiliza-se ferramenta para poder sintetizar a informação pra uma forma entendível para os envolvidos;
- Informações em tempo de execução – para saber como o usuário está utilizando a aplicação é necessário coletar informações em tempo de execução e entender como a aplicação está se comportando na máquina do usuário, tem grande importância. Informações como desempenho de consultas no banco de dados, erros não tratados e disponibilidade são as analisadas;
- Monitoramento de disponibilidade – outro tipo de monitoramento é saber quando a sua aplicação está no ar ou algum recurso da aplicação está indisponível para a utilização dos usuários;
- Sistemas analíticos – outro ponto vem se tornando importante, todos esses dados podem ser consumidos por um sistema analítico, para tentar encontrar correlações, tendências ou descobrir algo que está oculto na aplicação. Atualmente é necessário inúmeras tomadas de decisões, ter um sistema inteligente em tempo real facilita para ‘sobreviver’ no mercado;

Com isso departamentos de TI sofrem uma tremenda dor de cabeça, quando acontecem interrupções nos processos-chave de negócios e quando surgem reclamações constantes dos usuários sobre tempo de resposta lento ou usabilidade ruim das aplicações. Esta situação tem levado a uma crescente adoção de ferramentas focadas em monitorar como os usuários estão utilizando as aplicações corporativas (WORLD, 2010).

Como resposta, hoje existe um grande investimento por parte da indústria em soluções voltadas a monitoramento de aplicações da corporação, com foco em garantir o desempenho adequado dos sistemas. Medindo a experiências dos usuários, as empresas conseguem enxergar como uma aplicação se comporta na máquina do usuário, permitindo assim as equipes de

profissionais de operações atuarem de forma proativa para solucionar problemas (WORLD, 2010).

As atuais opções de ferramentas de monitoramento são muito variadas (onde serão citadas no tópico 5.4.4), como ferramentas de monitoramento para identificar o que acontece nas máquinas dos clientes, além de tabular métricas como tempo de resposta, erros e outras informações de sessões. Segundo Aberdeen, boas escolhas das empresas pode trazer benefícios significativos. De acordo com a pesquisa, as empresas podem antecipar em 53% de problemas nas aplicações antes de receber uma reclamação e percebem uma melhoria de 48% no tempo de correções de falhas no desempenho. Além disso, ainda é citado que as empresas melhoraram a visibilidade sobre transações críticas para os negócios em 42% e reduziram o número total de reclamações dos usuários em 15% (WORLD, 2010).

Para Jeffrey Hill, analista de pesquisas da Aberdeen, as empresas com visibilidade sobre experiência do usuário coletam dados valiosos, que podem ser usados futuramente para novas aplicações mais fáceis de usar e com menos recursos para funcionar. “As métricas tradicionais não conseguem avaliar complexidade e usabilidade das ferramentas usadas na empresa”, afirma o pesquisador (WORLD, 2010).

A Universidade de Northwestern, localizada na região de Chicago (Estados Unidos), implementou ferramentas de monitoramento de usuários, visando detectar desempenho e disponibilidade de suas aplicações *web*, com objetivo de criar uma visualização única sobre os serviços em tempo real, que abrangesse todo o campus. A ideia foi manter não só os profissionais de TI, mas todos os usuários informados sobre potenciais problemas de desempenho. Os dados coletados são utilizados pela universidade para determinar se há necessidades de redimensionamento de recursos ou atualizações nas aplicações que apresentam algum problema e definição de orçamento, como avalia Dana Nielsen, diretora da área de monitoramento de sistemas da Universidade Northwestern (WORLD, 2010).

Há sempre um conjunto de métricas padrão que são universalmente monitorados (uso do disco, uso de memória, carga, pings, etc). No artigo “10 *Things We Forgot to Monitor*” (10 coisas que esquecemos de Monitorar) escrito por Jehiah para a empresa Bitly, Inc, é criado uma lista de 10 coisa que são esquecidas de monitorar e ocasionam transtornos (JEHIAH, 2014).

1. Taxa de *Fork* (derivação/cópia autorizada de um repositório), normalmente, se espera uma taxa de 1 à 10/ s de *fork* em uma caixa de produção com o tráfego constante.
2. Controle de fluxo de pacotes, configurações de rede não feito da maneira correta,

pode ocasionar perda de pacotes ou derrubar o tráfego da rede temporariamente.

3. Troca de Entrada/Saída de taxa.
4. Notificação da inicialização de servidores, saber quando os servidores reiniciaram é importante para se comunicar com o provisionamento de novos servidores, e ajuda a capturar alteração de estado, mesmo se os serviços de lidar com o fracasso graciosamente sem alertar.
5. NTP Relógio *offset*, sincronização de tempo entre as máquinas.
6. Resoluções DNS (*Domain Name System*), verificar disponibilidade e quantidade de consultas, além de manter domínios *online*.
7. Vencimento SSL (*Secure Socket Layer*), canal criptografado entre um servidor *web* e um navegador (*browser*) deve garantir que todos os dados transmitidos sejam sigilosos e seguros, isso tem validade e pode trazer problemas, mesmo sendo raras as ocasiões.
8. Dell OMSA (*Open Manage Server Administrator*), ambiente gerenciado com o *hardware* da Dell, é importante para monitorar as saídas de OMSA. Isso alerta para status, falha de discos e questões de RAM (*Random Access Memory*).
9. Limites de Conexão, devido ao auto uso de *cache* com limites de conexão não se sabe o quão perto chega com dimensões de nível de aplicação.
10. *Status* do balanceamento de carga, examinar onde está sendo usado mais recursos.

Segundo Josephsen (2013), em conceito, sistemas de monitoramento são simples: sendo um sistema extra ou uma coleção de sistemas, cujo trabalho é assistir outros sistemas para ver se acontece algum problema, embora pareça simples, sistemas de monitoramento têm se tornado caros e em *software* complexos.

Um sistema de monitoramento pode ser o melhor amigo quando aplicado e implementado de maneira correta, permitindo notificar administradores de falhas antes que elas se transformem em crises, ajuda arquitetos a destrinchar padrões, correspondente a questões de interoperabilidade crônicas, e da aos engenheiros de planejamento capacidade detalhadas de informações. Um bom sistema de monitoramento pode ajudar a manter o acordo de nível de serviço (SLA)⁷ e até mesmo tomar medidas para resolver os problemas sem precisar acordar ninguém no meio da noite, poupar dinheiro, trazer estabilidade para ambientes complexos

⁷ Acordo de nível de serviço – do inglês *Service Level Agreement* (SLA) é um documento que define níveis de serviços acordados entre o cliente e o provedor de serviços, deve ser escrito de forma clara e concisa onde ambas as partes entendam, para definir quais serviços devem ser oferecidos e a disponibilidade necessária (FERNANDES, 2014).

tornando todos felizes (JOSEPHSEN, 2013).

No entanto um sistema mal monitorado pode causar vários estragos, se tornando em um fardo para todos os envolvidos, nesse contexto grandes corporações empregam especialistas de monitoramento em tempo integral e compram *software* sem se importar pelo preço, sabendo da importância de fazer um monitoramento direito na primeira vez. Mas para empresas pequenas e universidade de médio porte também podem existir ambientes complexos ou até mais complexo do que as empresas grandes, mas obviamente não têm o luxo de adquirir ferramentas de alto preço e conhecimentos especializados sobre o assunto de monitoramento. Assim como cita o autor Josephsen (2013), depois de ter passado os últimos 13 anos na construção e manutenção de sistemas de monitoramento, ele garante que um bom monitoramento pode ser feito de maneira gratuita, com algumas ferramentas de código aberto, com aplicação e imaginação dos envolvidos (JOSEPHSEN, 2013).

3.1 Considerações Finais

Nesse tópico sobre monitoramento de aplicações foi descrito a grande importância que um monitoramento pode ter pra um sistema ou aplicação, onde mostrou-se os pontos que estão recebendo bastante atenção pela comunidade e diversas ferramentas de monitoramento, dos benefícios que um bom monitoramento pode trazer e dos malefícios que um mal monitoramento do sistema traz, apresentando dados coletados da pesquisa feita por Aberdeen, onde foi mostrado que empresas podem antecipar problemas, melhorar a visibilidade dos negócios e reduzindo reclamações dos usuários.

No tópico 5.4 sobre ferramentas que o DevOps pode usar, será mostrado uma grande quantidade de ferramentas, das mais variadas para o monitoramento, no tópico a seguir será apresentado sobre outro assunto importante, quais são os problemas entre equipes de desenvolvimento e equipes de operações e como a abordagem de DevOps pode auxiliar na solução destes problemas.

4. PROBLEMAS ENTRE EQUIPES DE DESENVOLVIMENTO E EQUIPES DE OPERAÇÕES

Antes de falarmos sobre problemas e conflitos encontrados sobre as equipes, precisamos entender o que são equipes, qual seu real objetivo. Equipe consiste em um grupo de pessoas que compreende seus objetivos e está engajada em alcançá-los, de forma compartilhada. A comunicação entre os membros é verdadeira, opiniões divergentes são estimuladas, tornando um ambiente de trabalho onde profissionais assumam riscos com confiança, proporcionando que habilidades complementares dos membros possibilitam alcançar resultados e os objetivos compartilhados determinando seu propósito e direção. O grupo constantemente investe em seu próprio crescimento (MOSCOVICI, 2004).

Como é ilustrado na figura a seguir.

Figura 6: Representação do funcionamento de uma equipe.



Fonte: RH (2015).

4.1 Equipes de Desenvolvimento

Para que o *software* ou aplicação seja utilizado futuramente é preciso que este produto seja desenvolvido por pessoas capacitadas, para isso, existem as equipes de desenvolvimento, onde trabalham em um ambiente que envolve os mais variados fatores. O(a) desenvolvedor(a) que pertence a equipe de desenvolvimento busca sempre lógica e criatividade, passando a boa parte do tempo codificando soluções, focando o trabalho nos requisitos que o analista mapeou junto ao cliente (CARVALHO, 2013).

Como aborda Carvalho (2013), os(as) desenvolvedores(as) estão constantemente criando e aprimorando suas aplicações, com isto novas versões são criadas e precisam ser disponibilizadas, assim os clientes poderão usufruir dos recursos solicitados. Em outras palavras desenvolvedores(as) se preocupam em aumentar o valor do negócio.

A equipe de desenvolvimento tem muitas responsabilidades que precisam de atenção, como cita Leite (2000):

- *Design do Software*, atividade do desenvolvimento na qual o *software* deve ser concebido e especificado do ponto de vista do usuário e não do desenvolvedor, mantendo o foco na visão externa do *software*, onde será percebida pelo usuário;
- Prototipação, uma outra forma de concretizar a concepção de um *software* é a através de um protótipo, esta responsabilidade pode ser aplicada em vários momentos, antes de iniciar o desenvolvimento ou no andamento da construção do *software*, sendo ele junto ao cliente ou não;
- Programação, esta é a principal responsabilidade da equipe de desenvolvimento, consiste na atividade de construção de um programa que implementa uma determinada solução para um problema algorítmico. É também chamada de implementação, construção ou codificação do *software*. A codificação enfatiza a que o *software* deve ser construído utilizando uma linguagem de programação. Além da codificação, a programação deve envolver ainda a sua tradução num código de máquina executável. Esta responsabilidade pode envolver várias outras atividades, como aplicação de uma metodologia para a equipe, gestão dos fontes (manter controle dos códigos através de controle de versões, já citados), plano e casos de testes, automação do *Build* e distribuição automática, dentre outros;

- Avaliação ou Verificação, visa assegurar algumas das principais qualidades do *software*. Dentre as atividades de avaliação se destaca correções, validações e usabilidade do *software*. Consiste em testar o *software* para identificar possíveis erros;

O comportamento e rendimento de equipes de desenvolvimento tem reflexo direto com a motivação, equipes geralmente desmotivadas não produzem o que poderiam produzir, muitas vezes por mau gerenciamento do projeto ou mesmo por projetos que não atraem o interesse da equipe.

Sendo assim, equipes de desenvolvimento motivadas produzem mais e tornam proativa ou auto gerenciáveis. A valorização da equipe é outro fator importante, o que mostra a pesquisa feita pela SkillSoft, onde afirma que 28% dos profissionais de TI estariam insatisfeitos com o atual trabalho. Mais além, 75% deles trocariam de emprego na primeira oportunidade (JAGGS, 2006). Outro relatório técnico recente cita ainda que 41% dos profissionais espalhados pelo mundo sentem-se desmotivados no emprego (FRANÇA, 2015).

A seguir será apresentada outra equipe que pertence ao processo de desenvolvimento de *software*, a equipe de operações.

4.2 Equipes de Operações

Após o desenvolvimento do *software*, feita pela equipe de Desenvolvimento, é a vez da equipe de operações colocar o *software* acessível para os usuários ou clientes usarem. Para isso existe a equipe de operações, onde os profissionais capacitados têm a missão de manter os sistemas funcionando, são eles que fazem os *deploys* e os *rollbacks* das aplicações dos desenvolvedores, é responsabilidade deles manter o ambiente de produção intacto (CARVALHO, 2013).

Estes profissionais tem os deveres de rodar as aplicações, monitorar o funcionamento, a performance, avaliar e propor melhorias de forma a manter as aplicações com cuidado para estarem funcionando perfeitamente, rodando de forma rápida e estável, além disto, mudanças devem ser planejadas com cautela, tentando minimizar os riscos envolvidos. Se preocupando com segurança estabilidade e principalmente com o acordo de nível de serviço (SLA) de cada produto sob sua responsabilidade, está preocupação é fundamental para o negócio. Pode-se

dizer que as equipes de operações se preocupam em proteger o valor do negócio (CARVALHO, 2013).

Além dos itens citados, existem outros de importância como:

- Gestão de Ambientes de teste, gerenciamento do ambiente de teste do *software*, para que em teoria não apareça erros no ambiente de produção;
- Gestão de erros e incidentes, gerenciamento dos erros e incidentes ocorridos na aplicação, resolver de forma rápida e segura os problemas;
- *Feedback* Contínuo, manter as equipes envolvidas atualizadas sobre o que está acontecendo nos processos, no ambiente de produção e outros;

Sendo assim equipes de Operações tem todos os papéis relacionados à manutenção do produto, geralmente são também chamados de administradores de sistema (*sysadmins*). No tópico a seguir apresenta-se sobre os problemas que existem entre as equipes desenvolvimento e operações.

4.3 Problemas entre as Equipes

Hoje entregar *software* em ambientes de produção é um processo que tem se tornado cada vez mais complicado nas empresas de Tecnologia da Informação (TI), ciclos longos de testes e divisões entre as equipes de desenvolvimento e equipes de operações são alguns dos fatores que contribuem para o aumento da complexidade das entregas do *software* (DEVMEDIA, 2015).

Porém, a pressão sobre a TI para entregar mais inovação, com intervalos de tempo cada vez menores, para o negócio, vem mudando esse cenário. Esta pressão tem gerado tensão entre as equipes, onde equipes encarregadas de mudar o negócio (desenvolvimento) e aquelas responsáveis por mantê-lo funcionando (operações), enquanto equipes de desenvolvimento querem colocar suas funcionalidades ou aplicações em funcionamento o mais rápido possível, equipes de operações querem ter certeza de que as aplicações estão estáveis o suficiente para entrar em produção sem gerar incidentes (DEVMEDIA, 2015).

Este obstáculo cultural normalmente é encontrado entre equipes de desenvolvimento e operações tradicionais, devido as equipes trabalharem em silos, limitando a comunicação até os momentos de *release* do *software* (DUVALL, 2013).

Nos últimos anos esse conflito foi latente no mundo de TI, algumas empresas ainda mantêm regras rígidas, onde só permitem *deploy* uma vez por semana ou em casos mais rígidos apenas uma vez por mês, tudo isto pensando em proteger o negócio (CARVALHO, 2013).

Alguns fatores que geram estes conflitos entre as equipes, segundo Carvalho (2013):

- Surgimento de metodologia para desenvolvimento ágil para equipes de Desenvolvimento;
- Demora em fazer *Deploys* para a produção;
- Falta de *FeedBack* aos desenvolvedores sobre suas implementações;
- Ambiente dos desenvolvedores diferente do ambiente de produção;
- Equipe de operações com culturas arcaicas de administração;

A partir do momento que algum incidente acontece, ouve-se muito a expressão “Funciona na minha máquina” é uma situação bastante comum, são problemas que se manifestam apenas no ambiente de produção, como já foi comentado. Para isso é preciso entender que equipes de desenvolvimento e equipes de operações trabalham separadas, cada um com suas responsabilidades, sendo que ambos não estão dispostos a mudar a cultura, criando assim pessoas que não conseguem estabelecer uma forma sadia e eficiente de comunicação, e com isso, não existe trabalho colaborativo entre estas duas áreas (CARVALHO, 2013).

Sendo assim, cresce a necessidade de mudanças, equipes precisam ser multidisciplinares, onde todos os membros da equipe são responsáveis pelo processo de entrega. Qualquer pessoa na equipe pode modificar qualquer parte do sistema de *software*. O antipadrão correspondente são as equipes isoladas, que no desenvolvimento, teste e operações têm os próprios *scripts* e processos e não fazem parte da mesma equipe (DUVALL, 2013).

4.4 Considerações Finais

Neste tópico levantou-se os problemas que existem entre equipes de desenvolvimento e equipes de operações, onde apresentou-se um pouco sobre equipes de desenvolvimento, que trabalham com lógica e criatividade, passando a maior parte do tempo codificando soluções, focando o trabalho nos requisitos que o analista mapeou junto ao cliente e transformado está solução em código computacional. Logo após, foi abordado sobre equipes de operações, que tem como responsabilidade monitorar a aplicação e manter o ambiente de produção intacto.

Por fim, foi abordado sobre os reais problemas e conflitos encontrados entre as equipes, e a necessidade de unir estas equipes, tornando-as multidisciplinares e fazendo com que todos sejam responsáveis pelo processo de entrega do *software*. Este é o real objetivo que o DevOps se propõe a resolver, que será falado no capítulo a seguir.

5. DEVOPS

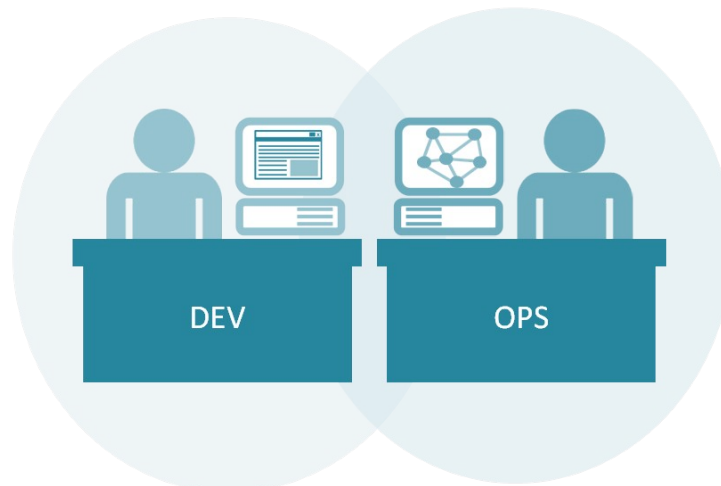
5.1 Introdução

O termo DevOps está em constante movimento e evolução, contudo, comenta-se sobre a temática, ferramentas e ideias que cercam o tema. O DevOps surgiu devido a necessidade de melhorar a agilidade das entregas no setor de TI. O movimento/cultura DevOps foca em aperfeiçoamento da comunicação, colaboração e integração entre desenvolvedores de *software* e administradores da infraestrutura de TI. Buscando modificar a antiga dinâmica, onde desenvolvedores e gerentes de infra eram vistos como ilhas isoladas livres de intercomunicação e cooperativismo (RELIC, 2015).

Sendo assim, pode-se conceituar o DevOps como um movimento cultural e profissional, que tem como objetivo fazer com que desenvolvedores e engenheiros trabalhem juntos, com foco em automação no maior número de processos possíveis no desenvolvimento de *software*. (SATO, 2013).

A figura 7 ilustra os dois personagens do DevOps.

Figura 7: Representação dos dois personagens do DevOps.

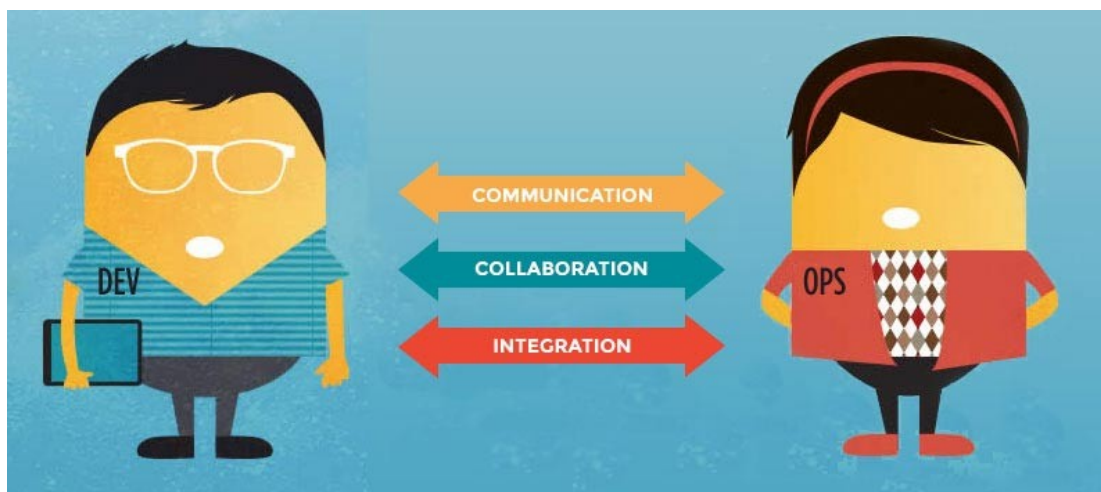


Fonte: Carvalho (2013).

Dentre vários significados do DevOps o mais óbvio deles é como o próprio termo já diz, significa a união de **Desenvolvedores** (devs) e **Operadores** (ops) de Sistemas (também conhecidos como *SysAdmins*) (CUKIER, 2010).

A figura a seguir mostra uns dos fatores necessários entre Dev e Ops: *Communication*(comunicação), *Collaboration* (colaboração) e *Integration* (integração).

Figura 8: Fatores necessários para a cultura DevOps.



Fonte: Relic (2015).

A adoção do movimento DevOps está sendo impulsionada por fatores como:

- Desenvolvimento por meio de metodologias ágeis;
- Aumento da velocidade das mudanças;
- Virtualização da infraestrutura;

Empresas inovadoras, como Facebook e Google entregam múltiplas versões dos sistemas por dia, com um elevado grau de qualidade. Enquanto empresas tradicionais não possuem uma infraestrutura ágil adequada para fazer o processo de forma contínua e utilizam métodos manuais como, por exemplo, *deploy* uma vez por semana e com isso não dão vazão às demandas (DEVMEDIA, 2015).

Conforme as empresas crescem, começam a surgir a necessidade de especialização nas áreas de desenvolvimento e *sysadmins*, e problemas começam a aparecer quando criam-se silos e a empresa fica dividida entre aqueles que criam o *software* e aqueles que mantêm tudo funcionando em produção. Essa divisão pode ser muito nociva para a empresa, uma vez que os profissionais ao invés de colaborarem para o sucesso e evolução da empresa, ficam em busca de quem foi o culpado, onde pouco importa para o negócio (CUKIER, 2010).

Devido a estes fatores, o DevOps prioriza a colaboração e a integração contínua, além da busca por ferramentas que promovam uma certa vantagem para a automação da configuração da infraestrutura para o desenvolvimento, isso tem trazido muitos benefícios para as empresas que produzem soluções de TI (RELIC, 2015):

- Melhoria na frequência de *deployments*, disponíveis para o mercado em menor tempo;
- Taxa de falhas muito baixa;
- Tempo de espera de projetos mais curto;
- Tempo de recuperação mais rápido;

Após observar as principais características deste movimento, analisa-se em como aplicar isto em um ambiente real, para entendimento, são analisados dois ambientes: ambiente de *startup* e ambiente corporativo em empresas, verificando em qual o movimento se encaixa melhor, segundo Carvalho (2013):

- Ambiente *startup*, a cultura DevOps combina muito com *startups*, nestes locais normalmente já se trabalha com metodologias ágeis, inclusive neste nicho que começaram a se discutir infraestrutura ágil – a precursora do movimento DevOps, portanto estas pessoas absorvem com mais facilidade os conceitos da cultura e compreendem os preceitos de colaboração e *feedback*. Em *startup* normalmente não existe divisões de departamentos, todos trabalham juntos, isso evita amarras e vícios da corporação, tornando assim um facilitador, afinal não existem barreiras para se comunicar;

- Ambiente Corporativo, diferentes das *startups* em ambientes corporativos existe burocracia e o uso vicioso de métodos ultrapassados, portanto não bastará o estímulo da alta hierarquia para que equipes de infra⁸ e devel⁹ comecem a vivenciar a cultura DevOps. A implantação neste ambiente não é impossível, mas será necessário grande motivação entre os envolvidos e flexibilidades entre as partes, o processo pode ser lento mas mudanças fantásticas podem ocorrer;

Para melhor entendimento sobre para que foi desenvolvido o DevOps, no próximo tópico será abordado sobre como surgiu o DevOps, onde será apresentado onde ele surgiu, como surgiu e como é mantido e divulgado para o mundo.

5.2 Como surgiu

O movimento DevOps não começou em apenas um lugar, existem muitos lugares que dão pistas sobre as origens do termo, por volta de 2008 começa-se a utilizar o termo infraestrutura ágil¹⁰ em algumas listas de discussão com foco em desenvolvimento ágil, e na mesma época durante evento *Agile*¹¹ 2008, surgem discussões sobre o tema metodologia ágil para a administração de infraestrutura (operações), inspirada no modelo ágil de desenvolvimento (CARVALHO, 2013).

O *Agile* abriu caminho para o DevOps, desviando as direções das metodologias de desenvolvimento de *software* modelo cascata e indo em direção a um ciclo contínuo de desenvolvimento, no entanto, o *Agile* não incluía o lado da operação, o qual continuava utilizando o modelo cascata (RELIC, 2015).

Em 2009 uma tempestade de metodologias e pensamentos se formou ainda mais, como é ilustrado na figura 9, *Agile*, *Theory of Constraints* (TOC), *Systems Thinking & Dynamics*, *Lean* e outras metodologias para a gestão dos serviços de TI se juntaram e criaram uma grande quantidade de seminários, conferências e palestras ao redor do mundo que começavam a ganhar força e serem cada vez mais difundidas a partir de então (RELIC, 2015). Após uma lista de

8 infra – representa a equipe de operações ou o profissional que trabalha na equipe (*sysadmins*).

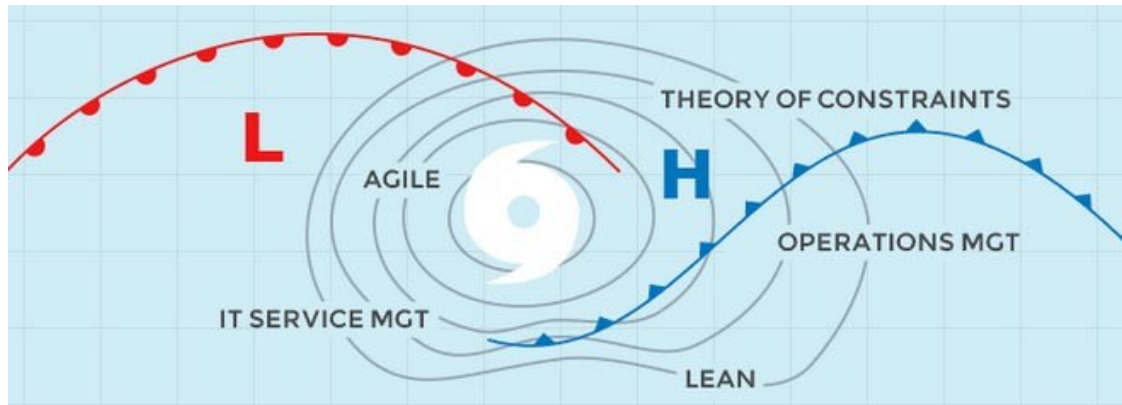
9 devel – representa a equipe de desenvolvimento ou o profissional que trabalha na equipe (desenvolvedor).

10 Infraestrutura ágil – é automatizar processos que aconteçam nas equipes de infraestrutura, sendo ela através de ferramentas e mudanças da cultura e a forma de trabalhar na infraestrutura (CARVALHO, 2013).

11 *Agile* – manifesto para descobrir melhores maneiras de se desenvolver *software*, valorizando a entrega do *software* em um curto espaço de tempo e incentivando a colaboração com o cliente.

discussão europeia com nome *agile-sysadmin* começar a abordar o tema com propriedade, isso ajudou a iniciar a ligação entre *developers* e *sysadmins* (CARVALHO, 2013).

Figura 9: Metodologias e pensamentos formados em 2009.



Fonte: Relic (2015).

Foi então que o termo DevOps foi criado durante a conferência *Velocity* da O'Reilly em 2009, onde John Allspaw (Etsy.com) e Paul Hammond (*Typekit*) apresentaram o trabalho “10+ Deploys Per Day: Dev and Ops Cooperation at Flickr”, com o objetivo de unir desenvolvedores(Dev) e administradores da infra de TI (Ops) de forma a promover a integração contínua até a entrega, como o próprio nome já diz: DevOps (CARVALHO, 2013).

Um dos participantes foi um entusiasta do assunto e sempre se mantinha ativo sobre o tema, era Patrick Debois, que após ter visto a palestra citada acima ficou muito animado, tendo a grande ideia de criar um encontro chamado DevOpsDay, que teve seu primeiro encontro em Ghent – Bélgica no final de 2009, o encontro durou 2 dias, e foi onde o assunto começou a ser conhecido pelo mundo. Novos eventos foram levados para diversos lugares do mundo, onde podem ser vistos na figura 10 as próximas edições e as edições passadas, seguidos por Patrick Debois, Gildas Le Nadan, Andrew Clay Shafer, Kris Buytaert, Jez Humble, Lindsay Holmwood, John Willis, Chris Read, Julian Simpson, R.I. Piennar (mcollective) e muitos outros (CARVALHO, 2013).

Figura 10: Ilustração das edições passadas e próximas do DevOpsDay.

Upcoming Past

2015	2009	2010	2011	2012	2013	2014	2015
Denver - April 2015	Ghent 2009	Sydney 2010	Boston 2011	Austin 2012	New Zealand 2013	Nairobi 2014	Ljubljana 2015
New York - April 2015		Mountain View 2010	Mountain View 2011	Tokyo 2012	London 2013	Ljubljana 2014	Paris 2015
Austin - May 2015		2010	2011	Delhi 2012	Paris 2013	Austin 2014	Paris 2015
Toronto - May 2015		Hamburg 2010	Melbourne 2011	Mountain View 2012	Austin 2013	Pittsburgh 2014	
Washington, DC - June 2015		Sao Paulo 2010	Bangalore 2011	2012	Berlin 2013	Amsterdam 2014	
Amsterdam - June 2015			Göteborg 2011	Rome 2012	Amsterdam 2013	Silicon Valley 2014	
Singapore - 2015			Manila 2011	New York 2012	Silicon Valley 2013	2014	
Minneapolis - July 2015					Downunder 2013	Minneapolis 2014	
Melbourne - July 2015					Bangalore 2013	Brisbane 2014	
Pittsburgh - August 2015					London Autumn 2013	Boston 2014	
Chicago - August 2015					2013	Toronto 2014	
Perth - October 2015					Barcelona 2013	New York 2014	
Silicon Valley - Nov 2015					Vancouver 2013	Warsaw 2014	
					Portland 2013	Chicago 2014	
					New York 2013	Berlin 2014	
					Atlanta 2013	Belgium 2014	
					Tel Aviv 2013	Helsinki 2014	
					Tokyo 2013	Vancouver 2014	

The [DevOps Calendar](#) lists DevOps conferences & events, both devopsdays and other.

Fonte: Devopsdays (2015).

É importante dizer que as pessoas que foram citadas a cima, foram responsáveis por disseminar a cultura DevOps pelo mundo, levando o evento para diversos países, com isso, direta ou indiretamente se tornaram referências para a revolução no mundo da TI (CARVALHO, 2013).

Inicialmente a cultura DevOps se mostrou muito presente no ambiente das *startups*, porém, algum tempo depois começou a fazer parte do mundo corporativo, apesar de terem organizado o *DevOpsDays* em diversos países, não foi estabelecido um manifesto para o assunto, criando a possibilidade de várias interpretações acerca deste termo (CARVALHO, 2013). Para os benefícios serem alcançados é necessário um entendimento dos conceitos que a cultura DevOps envolve e que são implantados em ambientes reais, o qual será abordado no tópico a seguir.

5.3 Conceitos

Como descrito, o DevOps tem o objetivo de trazer os conceitos e boas práticas aprendidas pelos Engenheiros de *Software* Ágeis para o mundo dos *sysadmins*. Não só isso, DevOps também procura clarear para os desenvolvedores as preocupações (justas) e práticas dos *sysadmins*. Faz parte do trabalho de devs e ops estarem alinhados e colaborarem um com o outro (CUKIER, 2010).

O obstáculo cultural normalmente encontrado, é que equipes de operações e desenvolvimento trabalham separadamente e sem comunicação ou sendo ela limitada. Pensando nisso surge a necessidade de estabelecimento de equipes multidisciplinares e a ampliação dos conjuntos de qualificações dos membros da equipe de entrega, são formas de aumentar a colaboração e transpor as barreiras tradicionais que impedem que os *softwares* sejam entregues continuamente (DUVALL, 2015).

Como Duvall (2015) aborda, para equipes serem bem sucedidas no DevOps é preciso:

- Crescimento das equipes multidisciplinares, consiste em especialistas em todo o ciclo de fornecimento de *software*, como engenheiros de operações, administradores de banco de dado (DBAs), testadores e analistas. Tornando assim todos os membros da equipe responsáveis pelo processo de entrega, em vez de tratar cada disciplina como uma organização de serviço centralizada separada, a equipe de entrega se torna o construtor organizacional chave, evitando que entregas do *software* sejam afetadas pelo impedimento de tempo inerentes de quando as equipes se comunicam através da organização;
- Todos fazem tudo, após um período, torna-se possível perceber que os conjuntos de qualificação frequentemente mudam para engenheiros/analistas, fazendo deles mais completos, começando a realizar mais do que apenas uma parte do processo de entrega de *software*. Cada membro tem o conhecimento e a capacidade dos processos da empresa. Quando os membros das equipes se tornam multiquificados os silos são desmanchados, a comunicação é melhorada e os gargalos são removidos das organizações e do *software* do passado;
- Projetos pilotos, as tentativas de mudar uma grande organização de uma só vez quase sempre tem a garantia de falharem, preferencial começar com um pequeno projeto piloto que forneça algo de valor de negócios para a produção em um

período de tempo relativamente curto (não mais de 90 dias), fazendo com que essa nova equipe seja multidisciplinar e todos os membros estejam completamente dedicados ao projeto, alcançados o sucesso comprovado desse projeto piloto é possível escalas maiores na organização, fazendo que os membros do processo piloto colaborem com o compartilhamento de conhecimento;

- Responsabilidades centralizadas, as funções centralizadas são aquelas de desenvolver serviços usados pelo restante da equipe de entrega de *software* ou aquelas que realizam o monitoramento do sistema, com equipes centralizadas estas responsabilidades tornam-se em autoatendimento;
- Sistemas de autoatendimento, uma das principais características do DevOps é que um membro da equipe nunca deve precisar de outro membro fora de sua equipe multidisciplinar para executar uma atividade como parte do processo de entrega;
- Os principais padrões que suportam o trabalho do DevOps podem ser apresentados na tabela 1.

Tabela 1: Apresentação dos principais padrões que suportam o DevOps.

Padrão	Descrição
Grandes painéis visíveis	As equipes em toda a organização obtêm informações em tempo real sobre o estado do sistema de software, incluindo status de desenvolvimento, métricas do cliente e disponibilidade.
Colocação	As equipes estão fisicamente próximas para aprimorar a comunicação.
Integração contínua	O software é desenvolvido (ambientes, aplicativos, etc.) com cada mudança.
Equipes multidisciplinares	Equipes de entrega de software compostas por especialistas de várias disciplinas, incluindo programadores, testadores, analistas e operações.
Especialistas com várias qualificações	Reduz os silos de especialistas ao expandir os conjuntos de qualificações nas equipes multidisciplinares.
Implementações com script	A implementação do software nos ambientes é completamente definida por scripts para que possa ser executada com um único comando.
Ambientes com script	A criação de ambientes é completamente definida por scripts para que possa ser executada com um único comando.
Releases de autoatendimento ("Você desenvolve, você executa.")	Qualquer pessoa autorizada na equipe pode e executa implementações para a produção.
Parar a linha	Todos podem e devem parar o sistema de integração contínua quando necessário.
Todos os itens orientados por teste	Escreva testes automatizados para tudo: aplicativos, infraestrutura, tudo. Isso deve incluir escrita da unidade, a aceitação, o carregamento e os testes de desempenho.
Indicação de versão de todos os itens	Indicação versão de todos os artefatos: infraestrutura, configuração, código do aplicativo e dados.

Fonte: Duvall (2015).

Os segredos para o DevOps ser efetivo é a quebra dos silos e a criação de equipes multidisciplinares que podem implementar o *software* na produção. Organizações que mantêm

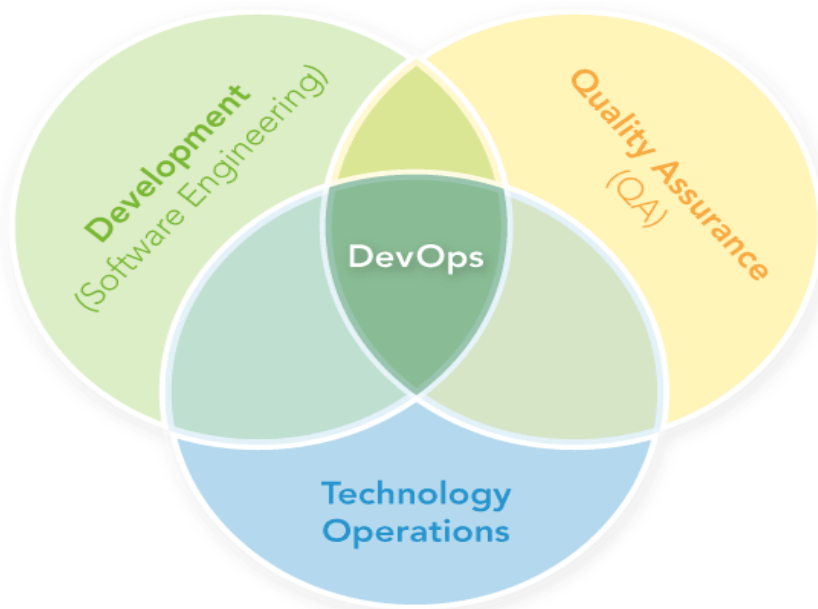
longos ciclos de intermediação e entrega devem evoluir ou se retirar dos negócios. Grandes organizações, o desenvolvimento pode ser demorado e requer uma mudança na cultura organizacional, ponto que é um dos pilares do DevOps (DUVALL, 2015). Sendo assim como relata Sete (2015), a cultura/movimento DevOps se mantém em quatro pilares principais, conhecidos pelas siglas C.A.M.S, são eles:

- **Cultura**, as equipes precisam ter colaboração, manter uma relação saudável entre as áreas, mudança de comportamento, flexibilidade, buscando sempre ter em mente que pessoas e processos veem em primeiro lugar. Se a cultura de integração e colaboração entre as equipes de desenvolvimento e operação não existir, qualquer tentativa de automação será em vão, por este motivo torna-se o maior desafio para qualquer iniciativa de DevOps. Se refletirmos, DevOps não tem nada de novo em relação ao que foi dito há quase 10 anos, quando surgiu o Manifesto Ágil. O manifesto propunha que o cliente e o time trabalhassem juntos, numa relação de confiança e cumplicidade. Já o DevOps propõe que esse “trabalhar junto” se estenda para **todos os envolvidos**. Não só quem faz o produto, mas também quem mantém (*sysadmins*), quem vende (comercial), quem divulga (*marketing*), quem atende o cliente (suporte técnico). Todos devem ter conversas, respeito e principalmente, trabalharem juntos. Por isso, vestir a camisa de DevOps exige grandes mudanças culturais (CUKIER, 2015);
- **Automação**, após a etapa da cultura ser vencida, ferramentas entram em cena para automatizar o maior número de processos, sendo eles: automação para liberação de versão, automação de *build*, de provisionamento de ambientes para testes, monitoramento de disponibilidade e performance de ambientes, análise de comportamentos dos usuários, controle, gerencia e outros. Automação de processos é fundamental para que haja boa sinergia entre desenvolvimento e operação. Automação é um componente de DevOps e não o DevOps propriamente dito, ferramentas não são o suficiente para que todo esse movimento cultural aconteça. Para que se mude o paradigma é preciso mudança de atitudes e comportamentos, que consigam juntar pessoas em torno de um real propósito;
- **Medição/Avaliação**, deve-se medir tudo que possível, performance, processos e interações e até mesmo pessoas. Sem medir, não se pode melhorar nem aperfeiçoar os processos;

- **Compartilhamento**, ter uma boa comunicação entre as equipes, incentivar as pessoas a se comunicarem e compartilharem ideias e problemas é um ponto crucial numa iniciativa do DevOps. Histórias de sucesso atraem novos talentos para o movimento e criam um excelente canal de *feedback*, que fomentam um processo de melhoria contínua;

A figura 11 ilustra onde o DevOps atua, entre os *Development* (desenvolvimento), *Technology Operations* (Operações) e *Quality Assurance* (Garantia de qualidade).

Figura 11: Representação do local onde o DevOps atua.



Fonte: Carvalho (2013).

Segundo Carvalho (2013) existe uma relação de características técnicas que um ambiente DevOps deve ter/possuir/oferecer/permitir, são elas:

- Infraestrutura como código, virtualização de ambientes, uma máquina física poderia se tornar um parque com dezenas de máquinas virtuais administradas na nuvem;
- Orquestração de servidores, são ferramentas que permitem executar comandos e controlar servidores em tempo real;
- Gerência de configurações, controlam estados do sistema, ajudam a centralizar todas as configurações e facilitam a administração e criação de novos ambientes;
- Provisionamento dinâmico de ambientes, etapas de preparações iniciais de

configuração de ambientes;

- Controle de versões compartilhado entre infra e devel;
- Ambiente de desenvolvimento, teste e produção (no mínimo);
- O ambiente de devel deve possibilitar TDD (*Test Driven Development*);
- Infra deve participar dos projetos desde o início, significa participar das reuniões técnicas, quanto mais problemas foram resolvidos durante o projeto (com ajuda da infra) menos problemas serão expostos aos clientes;
- Infra deve participar das reuniões de devel, importante para a infra observar quais são as metas da empresa a longo prazo, analisando onde o devel quer chegar, permitindo se programar melhor.
- Devel deve participar das reuniões de infra, o devel deve entender e ter ciência da realidade da infra, seus processos, qualidades e limitações, possibilitando que o devel ajude a solucionar problemas e a forma que administram o ambiente;
- Ambiente de entrega contínua, o devel precisa adotar alguma metodologia de entrega ou desenvolvimento contínuo e a infra entender esse processo para criar um ambiente com as ferramentas certas;
- Os desenvolvedores devem conseguir fazer o *deploy* sem interferência da infra, a infra precisa ceder para evoluir, permitindo que o devel consiga gerar e controlar o código, validando a aplicação e entregando uma nova versão sem encarar um processo burocrático de mudança;
- Monitoramento eficaz com processamento adequado dos eventos e métricas;
- Capacidade de resposta rápida a incidentes e problemas;
- *Backup e restore* confiável;

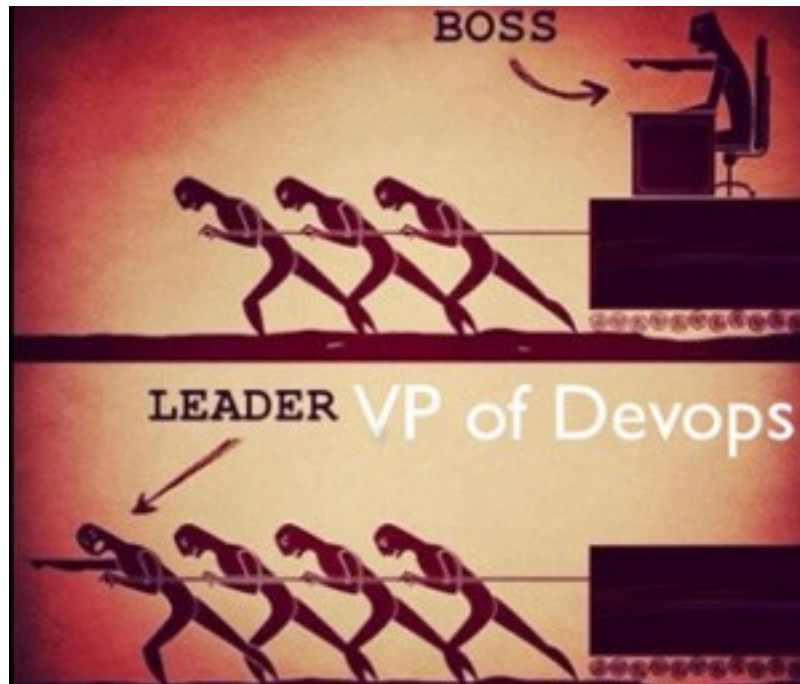
Mas para alcançar todas estas características técnicas as equipes precisam estar motivadas para a cultura funcionar, a equipe precisa observar e exercitar os seguintes valores (CARVALHO, 2013):

- Confiar no trabalho das equipes;
- Respeito pessoal e profissional por todos da equipe;
- Sinceridade/Honestidade sobre eventos e incidentes ocorridos (não esconda nada da sua equipe);
- Entendimento de que o problema ou solução é responsabilidade de todos;
- Entendimento de que os resultados são o reflexo do trabalho de toda a equipe;

- Comunicação efetiva e dinâmica;
- Postura construtiva sempre;
- Espírito de colaboração;

A figura a seguir mostra o papel do líder na equipe com a cultura DevOps.

Figura 12: Ilustração do papel do líder da equipe com a cultura DevOps.



Fonte: Debois (2014).

Alcançando todas essas técnicas, e tendo observado e exercitado os valores citados acima, apareceram ganhos em adotar a cultura DevOps (CARVALHO, 2013):

- Ganhos para a infra: infraestrutura como código (equipe para de administrar e passa a desenvolver a infra), infra mais eficiente e rápida usando métodos ágeis, mais organizada, comunicação melhor, infra fazendo mais em menos tempo com menos gente, ambientes de gerência de configuração, orquestração e provisionamento implantados, *deploys* da infra (novos ambientes) mais rápidos e seguros tornando a entrega mais rápida, ambiente padronizado e sob controle e o *feedback* rápido em todas as atividades de infra;
- Ganhos para o devel: ambiente mais adequado para trabalhar (desenvolvimento/teste/produção), passa a contar com ambiente de desenvolvimento contínuo e testes automatizados, *deploys* de apps (novas versões) mais rápidos e seguros tornando a entrega e o *feedback* simples em

todas as fases de desenvolvimento;

- Ganhos mútuos para Infra/Devel: fim da divisão infra e devel, infra participa dos projetos e acompanha tudo de perto, infra melhora o planejamento do ambiente de produção e monitora a aplicação de forma mais eficaz, devel passa a entender melhor a infra, resultando em um produto melhor e equipes trabalhando em conjunto, aumentando o valor do negócio;
- Ganhos para a empresa: melhor comunicação entre devel e infra (diminuição de conflitos), soluções rodando com maior estabilidade e desempenho, entregas mais rápidas, menor tempo de paradas, diminuição de incidentes/custos/riscos e aumento do valor de negócio;

O DevOps é um movimento em constante construção e definição, seguir ou aplicar a cultura DevOps em uma empresa parte da mudança de cultura, mas além disso, outro fator é a automação, e para alcançar a automação de processos é necessário ferramentas, no qual será abordado no tópico a seguir, Mudança de comportamento e foco nas pessoas é o que vai mudar a cultura das organizações explica Peter Drucker.

5.4 Ferramentas

Um dos principais princípios do DevOps é investir em automação. A automação permite executar tarefas ou processos mais rapidamente e diminuir a possibilidade de erros humanos. Um processo automatizado é mais confiável e pode ser analisado com mais facilidade. Com o avanço da cultura DevOps e o aumento da colaboração entre administradores de sistema e desenvolvedores diversas ferramentas das mais variadas tem evoluído para tentar automatizar processos existentes no desenvolvimento de um *software* (SATO, 2013).

O DevOps tem uma forte ligação com várias ferramentas para automatização, o primeiro passo é avaliar a grande quantidade de ferramentas disponíveis, sendo elas gratuitas ou pagas (FLUX7, 2014).

A seguir serão citadas e exploradas algumas das ferramentas mais utilizadas e indicadas pela comunidade para o gerenciamento de ambiente de desenvolvimento, gerenciamento da configuração de máquinas ou servidores, gerenciamento das configurações da aplicação e do monitoramento da aplicação. Que permitem alcançar os conceitos e princípios do DevOps.

5.4.1 Gerenciamento do Ambiente de Desenvolvimento

Vagrant

Ferramenta que permite a construção de ambientes virtualizados de desenvolvimento completos, com um fluxo de trabalho fácil e simples de usar e com foco na automação. Muito usado atualmente, o *Vagrant* permite reduzir o tempo de instalação de um ambiente, visando aumentar a semelhança dos ambientes de desenvolvimento e produção. *Vagrant* é um projeto *open source* que surgiu em 2012 (VAGRANT, 2015).

A figura 13 ilustra o logo do *Vagrant*.

Figura 13: Logo *Vagrant*.



Fonte: Vagrant (2015).

Desenvolvedores *web* usam ambientes virtuais todo dia em suas aplicações *web*, a virtualização é a ferramenta preferida para facilitar a implantação e o gerenciamento de infraestrutura. O *Vagrant* busca usar esses mesmos princípios por meio do fornecimento de máquinas virtuais fáceis de configurar, leves, reproduzíveis e portáteis, direcionadas para ambientes de desenvolvimento (VAGRANT, 2015).

Alguns benefícios de usar o *Vagrant* é (VAGRANT, 2015):

- Para desenvolvedores individuais, cada projeto depende de suas próprias bibliotecas, sistemas de fila de mensagens, bancos de dados, *frameworks* e mais, cada um com suas próprias versões, além das dependências. Rodar tudo isso em uma máquina pessoal e desligar tudo no final do dia ou quando estiver trabalhando em outros projetos também é inviável. O *Vagrant* fornece ferramentas para construir ambientes de desenvolvimento únicos para cada projeto de uma vez, e depois facilmente derrubá-los e reconstruí-los, economizando tempo e frustração;
- Para equipes, atualmente os membros da equipe idealmente não mantêm ambientes de desenvolvimento idênticos: mesmas dependências, mesmas versões, mesmas configurações e etc. Devido a evolução constante de bibliotecas e versões de programas, tudo isso está destinadas a causarem problemas reais em

algum momento futuro, com *Vagrant* equipes tem a possibilidade de garantir um ambiente virtual de desenvolvimento consistente e portátil que seja fácil e rápido de criar;

- Para empresas, para manutenção em uma aplicação web grande, uma das partes mais difíceis é acrescentar novos recursos. *Cache*, servidores de banco de dados e outros pontos de infraestrutura significam uma série de instalações e um monte de outras configurações. O *Vagrant* fornece ferramentas para construir um ambiente de desenvolvimento uma vez e depois distribuí-lo facilmente para os novos membros da sua equipe de desenvolvimento, desta forma novos desenvolvedores podem começar a trabalhar rapidamente economizando tempo e dinheiro;

Para entender melhor como o *Vagrant* funciona, é importante conhecer os termos mais comuns usados, segundo Heidi (2014).

Box: Uma *box* (caixa) é basicamente um pacote que representa um sistema operacional instalado (e alguns pacotes básicos), para um provedor específico. O *Vagrant* irá replicar essa imagem para uma máquina virtual. Definido o projeto e ter escolhido qual *box* será a base do ambiente de desenvolvimento. Inicialmente quando ser rodado o *Vagrant* com uma *box* nova, ele irá fazer o *download* da *box* e importá-la para o sistema.

Host e Guest: A máquina/sistema operacional *Host* é onde o *Vagrant* está instalado. A máquina *Guest*, como pode-se deduzir, é a máquina virtual inicializada pela *Host*.

Provedores: o provedor é o *software* de virtualização responsável por criar as máquinas virtuais que o *Vagrant* administra. VirtualBox é o provedor padrão do *Vagrant*, mas pode ser usado VMWare, KVM, dentre outros. Para usar outros provedores, normalmente é preciso instalar um *plugin*.

Plugins: Um *plugin* pode adicionar funcionalidades extras ao *Vagrant*, como suporte a outros provedores, por exemplo.

Provisionadores: Um provisionador (*provisioner*) irá automatizar o processo de instalação e configuração do ambiente, instalando pacotes e executando tarefas em geral.

Vagrantfile: O Vagrantfile é o arquivo de configuração do *Vagrant*, que conterá todas as definições da máquina virtual. Normalmente ele fica localizado na raiz do projeto.

Diretórios sincronizados: Os diretórios sincronizados (*synced folders*) são usados para compartilhar conteúdo entre a máquina *Host* e a máquina *Guest*. As modificações são refletidas em tempo real, porque o conteúdo do diretório é compartilhado entre as máquinas.

Requerimentos: Para provisionar sua máquina virtual, o *Vagrant* precisa de um *software* de virtualização, como VirtualBox ou VMWare. O VirtualBox é o padrão, por ser gratuito e *open source*.

A tabela 2 é uma referência rápida para os principais comandos que o *Vagrant* oferece:

Tabela 2: Principais comandos do *Vagrant*.

comando	descrição	uso comum
up	Inicializa a máquina virtual e roda o(s) provisionador(es)	Quando a máquina não está rodando ainda
reload	Reinicia a máquina virtual	Quando você faz modificações no Vagrantfile
provision	Roda apenas o(s) provisionador(es)	Quando você faz modificações no provisionamento
init	Cria um novo Vagrantfile baseado em uma box	Quando você quer gerar um Vagrantfile
halt	Desliga a máquina virtual	Quando você quer desligar a VM
destroy	Destroi a máquina virtual	Quando você quer começar do zero
suspend	Suspende a execução da máquina virtual	Quando você quer salvar o estado da VM
resume	Retoma a execução da máquina virtual	Quando você quer retomar a execução de uma máquina virtual que foi suspensa anteriormente
ssh	Faz login via SSH (não pede senha ou login)	Quando você quer fazer modificações manuais ou debugar

Fonte: Heidi (2014).

O *Vagrant* não trabalha apenas com máquinas virtuais, alguns provedores específicos utilizam outros métodos para criação do ambiente, como no caso do *Docker*, que será abordado na sequência, que utiliza *containers* (VAGRANT, 2015).

Docker

Parecido com o *Vagrant*, com objetivo de padronizar ambientes, *Docker* é uma plataforma aberta para desenvolvedores e administradores de sistema, que ajuda na criação e execução de aplicações distribuídas. Permite que aplicativos sejam montados rapidamente a partir de componentes (DOCKER, 2015).

Figura 14 ilustrativa do logo do projeto *Docker*:

Figura 14: Logo Docker.



Fonte: Docker (2015).

Atualmente é um dos grandes desafios trazer sua aplicação juntamente com suas dependências e fazê-las funcionar sem tropeços em outro lugar ou em um servidor de produção remoto. *Docker* busca resolver isso, oferecendo um conjunto completo de ferramentas de alto nível para transportar tudo que constitui uma aplicação entre sistemas e máquinas. Este projeto teve o código aberto pela dotCloud em Março de 2013, consiste de várias partes principais e elementos as quais são todas construídas em cima de funcionalidades já existentes, bibliotecas e *frameworks* oferecidos pelo *kernel* do *Linux*, elementos que serão citados a seguir, segundo Tezer (2014).

Contêineres Docker: Todo o processo de portar aplicações usando *docker* depende, exclusivamente, do envio de contêineres. Contêineres são diretórios que podem ser empacotados como qualquer outro, e então, compartilhados e executados entre várias máquinas e plataformas, é os contêineres que contêm tudo que constitui sua aplicação. A única dependência é ter o *Docker* instalado e ajustado.

Imagens Docker: As imagens *docker* constituem a base para os contêineres *docker*, similares às imagens de disco padrão de sistema operacional, que são utilizadas para executar aplicações em servidores e computadores de mesa. Permitindo a portabilidade com uma base sólida, consistente e confiável com tudo de necessário para executar a aplicação.

Quanto mais camadas são adicionadas em cima da base, novas imagens podem ser formadas aplicando-se estas alterações. Este processo pode ser feito manualmente trabalhando com o *docker* CLI, para criação de um novo contêiner ou podem ser especificadas dentro de um *Dockerfile* para construção de imagem automatizada.

Dockerfiles: são *scripts* contendo uma série sucessiva de instruções, orientações e comandos que devem ser executados para formar uma nova imagem *docker*. Cada comando executado é uma nova camada para o produto final. Usados para substituir o processo de se fazer tudo manualmente e repetidamente.

A figura 15 mostra os comandos de utilização oferecidos pelo *Docker*:

Figura 15: Comandos fornecidos pelo Docker.

```
A self-sufficient runtime for linux containers.

Commands:

attach    Attach to a running container
build     Build a container from a Dockerfile
commit    Create a new image from a container's changes
diff      Inspect changes on a container's filesystem
export    Stream the contents of a container as a tar archive
history   Show the history of an image
images    List images
import     Create a new filesystem image from the contents of a tarball
info      Display system-wide information
insert    Insert a file in an image
inspect   Return low-level information on a container
kill      Kill a running container
login     Register or Login to the Docker registry server
logs      Fetch the logs of a container
port      Lookup the public-facing port which is NAT-ed to PRIVATE_PORT
ps        List containers
pull      Pull an image or a repository from the Docker registry server
push      Push an image or a repository to the Docker registry server
restart   Restart a running container
rm        Remove a container
rmi       Remove an image
run       Run a command in a new container
search    Search for an image in the Docker index
start     Start a stopped container
stop      Stop a running container
tag       Tag an image into a repository
version   Show the Docker version information
wait      Block until a container stops, then print its exit code

you@tutorial1:~$
```

Fonte: Docker (2015).

Como mostrou-se anteriormente existem ótimas ferramentas para utilização na padronização de ambientes de desenvolvedores e operações, evitando atritos entre as áreas e buscando resolver um dos problemas centrais, que é ambientes de desenvolvimento diferentes dos ambientes de produção, no qual foram citadas duas ferramentas o *Vagrant* e *Docker*. No tópico a seguir será abordado sobre como gerenciar as configurações de máquinas ou servidores, através de duas ferramentas oferecidas.

5.4.2 Gerenciamento de Configurações

Puppet

Ferramenta de código aberto para gerenciamento de configuração. Utilizado para gerenciar milhares de máquinas físicas e virtuais. A ideia é ter a configuração centralizada e sendo distribuída pra várias máquinas ou servidores na rede (CARVALHO, 2013).

A figura a seguir ilustra o logo.

Figura 16: Logo Puppet.



Fonte: <http://puppetlabs.com/>.

O *Puppet* é usado desde 2005 por muitas organizações, incluindo Google, Twitter, Oracle e Rackspace para o gerenciamento da infraestrutura (DUVALL, 2012).

Cada comando declarado em uma linguagem entende-se por uma diretiva, no *Puppet* essas diretivas são chamadas de recursos. Alguns exemplos de recursos que podem ser declarados no *Puppet* são: pacotes, arquivos, usuários, grupos, serviços, trechos de *script* executável, dentre outros. O arquivo que é declarado esses conjuntos de diretivas é chamado de manifesto. Com a utilização do *puppet* a principal ação é a criação e edição desse manifesto (SATO, 2013).

O funcionamento do *puppet* usa o conceito de servidor principal, que centraliza a configuração entre nós e agrupa-os com base em tipo. Permitindo implementar mudanças de infraestrutura em vários nós simultaneamente.

Na tabela 3 são listados os principais componentes do Puppet:

Tabela 3: Principais componentes do *Puppet*.

Componente	Descrição
Agente	Um processo daemon executando em um nó que coleta informações sobre o nó e envia essas informações ao Puppet principal.
Catalog	Compilação de fatos que especificam como configurar o nó.
Facts	Dados sobre um nó, enviados pelo nó para o Puppet master.
Manifest	Descreve recursos e as dependências entre eles.
Módulo	Manifestos relacionados a grupos (em um diretório). Por exemplo, um module pode definir como um banco de dados como o MySQL é instalado, configurado e executado.
Nó	Um host gerenciado pelo Puppet principal. Os nós são definidos como classes, mas contêm o nome do host ou nome completo do domínio.
Puppet master	O servidor que gerencia todos os nós do Puppet.
Recursos	Por exemplo, o pacote, arquivo ou serviço.

Fonte: Duvall (2012).

Por utiliza um modelo declarativo com gerenciamento de dependência explícito ele tende a ser uma das primeiras considerações de ferramenta pelos engenheiros com experiências de administração de sistemas, que buscam criar um *script* dos seus ambientes (DUVALL, 2012). A seguir será explorada o Chef, outra ferramenta popular para automação de infraestrutura de *software*.

Chef

Disponível desde 2009 e influenciado pelo *puppet* e pelo CFEngine. O Chef oferece suporte para várias plataformas e é descrito como mais fácil de usar, especialmente por desenvolvedores de Ruby, pois todos os *scripts* são escritos em Ruby (DUVALL, 2012).

Ilustração do logo figura 17.

Figura 17: Logo Chef.



Fonte: Chef (2015).

O Chef permite automatizar a forma como se constrói, implanta e gerencia a infraestrutura, tornando versionável e testável. Todos os servidores, máquinas virtuais ou dispositivo que se gerencie é mantido com o Chef instalado, considerando assim como nós, que

mantêm as etapas e dados de configuração da aplicação. Isso acaba facilitando a consulta de verificar se algum nó está desatualizado (CHEF, 2015).

Existem três componentes centrais que interagem entre si, *Chef server*, *nodes* e a *Chef workstation*: O Chef executa *cookbooks*, que consistem em *recipes* que realizam etapas(ações) automatizadas em nós, exemplo instalar e configurar *software* ou adicionar arquivos. O servidor Chef contém dados de configuração para gerenciar vários nós, estes arquivos e recursos de configuração são armazenados nele, que são puxados pelos nós quando solicitados. O Chef também oferece uma interface de linha de comando, chamada *Knife*. Uma estação de trabalho Chef é uma instância com um repositório Chef local e Knife instalados nela (DUVALL, 2012).

Na tabela 4 são descritos os principais componentes do Chef:

Tabela 4: Principais componentes do Chef.

Componente	Descrição
Attributes	Descrevem os dados do nó, como um endereço IP e o nome de host.
Chef Client	Realiza trabalho em nome de um nó. Um único Chef client pode executar receitas para vários nós.
Chef Solo	Permite que você execute os livros de receita Chef na ausência de um servidor Chef.
Cookbooks	Contém todos os recursos necessários para automatizar sua infraestrutura e podem ser compartilhados com outros usuários do Chef. Os cookbooks normalmente consistem em várias receitas.
Data bags	Contém dados globalmente disponíveis usados por nós e funções.
Knife	Usado por administradores do sistema para carregar alterações de configuração para o Servidor Chef. O Knife é usado para comunicação entre os nós via SSH.
Management console	A interface da web do servidor Chef para gerenciar nós, funções, livros de receitas, pacotes de dados e clientes de API.
Nó	Hosts que executam o cliente Chef. Os principais recursos de um nó, do ponto de vista do Chef, são seus atributos e lista de execução. Os nós são o componente ao qual receitas e funções são aplicadas.
Ohai	Detecta dados sobre seu sistema operacional. Pode ser usado de maneira independente, mas seu objetivo principal é fornecer dados do nó ao Chef.
Recipe	A configuração fundamental no Chef. Recipes contém coleções de recursos que são executados na ordem definida para configurar os nós.
Repository (Chef repository)	O lugar em que são hospedados os cookbooks, roles, configuration files e outros artefatos para gerenciar sistemas com o Chef.
Resource	Uma abstração de várias plataformas de algo que você está configurando em um nó. Por exemplo, usuários e pacotes podem ser configurados de maneira diferente em plataformas de sistemas operacionais diferentes. O Chef elimina a complexidade de fazer isso para o usuário.
Role	Um mecanismo de agrupar recursos similares em nós similares.
Server (Chef server)	Repositório centralizado da configuração do seu servidor.

Fonte: Duvall (2012).

A comunidade do Chef vem crescendo rapidamente, enquanto é desenvolvido livros de receitas para que outros usem. Por ser uma ferramenta que estende a linguagem *Ruby* para fornecer um modelo de aplicar configuração a muitos nós de uma só vez. O Chef usa um modelo interpretativo sem gerenciamento de dependência explícito, assim, pessoas com pouca experiência em desenvolvimento tendem a sofrer mais na realização dos *scripts* dos ambientes (DUVALL, 2012).

Observa-se na última década que várias ferramentas comerciais e de *software* livre

surgiram para dar suporte à automação da infraestrutura, dentre as de código livre Bcfg2, CFEngine, Chef e *Puppet*. E abordamos sobre as ferramentas mais populares Chef e *Puppet*, segundo Duvall (2012). No tópico a seguir serão destacadas ferramentas que auxiliam no gerenciamento de bibliotecas, pacotes da aplicação, que tem grande importância para desenvolvedores hoje em dia.

5.4.3 Gerenciamento das Configurações da Aplicação

Outro ponto importante e que evoluiu muito é o gerenciamento das configurações da aplicação. Muitas linguagens já possuem este recurso, exemplo o PHP com o *Composer*, Ruby com o *Bundler* e Java como o Maven e *Ivy*. Estas ferramentas facilitam o desenvolvimento da aplicação, evitando tempo perdido para a configuração de uma biblioteca que será utilizada. Mantendo a gerencias das mesmas de maneira mais fácil. A seguir será descrito um pouco delas:

Composer

Na comunidade PHP, o *Composer* é um gerenciador de dependências da aplicação, permite manter e incluir novos pacotes/bibliotecas necessárias facilmente na aplicação. O *Composer* requer a versão do 5.3.2 ou maior do PHP e o git para funcionar (COMPOSER, 2015).

Logo do *Composer* na figura 18.

Figura 18: Logo Composer.



Fonte: Composer (2015).

Para utilização é necessário obter o *Composer*, disponível no *site*¹², e manter este arquivo *composer.phar* no projeto, este arquivo é um arquivo binário, que pode ser executado por linha de comando e ter um arquivo *composer.json* onde é declarado as dependências do projeto (COMPOSER, 2015).

Na figura a seguir é mostrado o formato do arquivo *composer.json* e como são declaradas as dependências.

Figura 19: Representação do arquivo *composer.json*.

```
{
  "require": {
    "monolog/monolog": "1.2.*"
  }
}
```

Fonte: *Composer* (2015).

No exemplo a cima foi informado a dependência do *monolog* para o projeto, que é disponível no repositório do *composer*, que é o Packagist¹³, no packagist são encontradas todas as dependências que o *composer* pode fazer (CARDOSO, 2014). Um dos principais comandos no *composer* são, segundo Composer (2015):

- `php composer.phar install`, comando de instalação, que lê o arquivo *composer.json* do diretório atual e resolve as dependências as instalando em um diretório */vendor*;
- `php composer.phar update`, comando utilizado para obter novas versões das dependências;
- `php composer.phar self-update`, comando utilizado para obter a última versão do *composer*;

¹² Link para download – <https://getcomposer.org/installer>.

¹³ Link de acesso – <https://packagist.org/>.

Bundler

Na comunidade *Ruby*, o *Bundler* gerencia dependências entre as *gems* (bibliotecas de conjuntos de arquivos *Ruby* reusáveis), oferecendo um ambiente consistente para projetos *Ruby* através do rastreamento e instalação das *gems* exatas ou versões que são necessários (BUNDLER, 2015).

Figura 20 mostra o logo do *Bundler*.

Figura 20: Logo Bundler.



Fonte: Bundler (2015).

Da mesma forma que o *composer* utiliza o arquivo *composer.json* para declarar as dependências, o *Bundler* utiliza o arquivo *Gemfile*, como mostra a figura 21.

Figura 21: Representação do arquivo *Gemfile*.

```
source 'https://rubygems.org'

gem 'rails', '4.1.0.rc2'
gem 'rack-cache'
gem 'nokogiri', '~> 1.6.1'
```

Fonte: Bundler (2015).

Os principais comandos do *Bundler* são, segundo Bundler (2015):

- *bundle install*, o *Bundler* irá se conectar ao *rubygems.org*, e encontrará uma lista de todas as *gems* necessários que atendam aos requisitos especificados. E instalara as *gems* que foram definidas no arquivo *Gemfile*;
- *bundle update*, utilizado para atualizar todas as dependências ou uma em específico;

Maven

Na comunidade Java, o Maven e o *Ivy* se destacam para gerenciar *jars* (arquivos compactados de um conjunto de classes Java) e suas dependências.

A figura a seguir representando o logo do Maven.

Figura 22: Logo Maven.



Fonte: Maven (2015).

Maven é uma ferramenta que pode ser usado para a construção e gerenciamento de qualquer projeto baseado na linguagem Java, tornando trabalho dos desenvolvedores Java mais fácil. O objetivo principal é permitir que um desenvolvedor tenha a compreensão do estado completo, com um esforço de desenvolvimento no menor período de tempo. Segundo Maven (2015), para atingir o que foi visto até aqui, o Maven lida com:

- Tornar o processo de compilação fácil;
- Fornecer um sistema de construção uniforme;
- Fornecer informações sobre o projeto de qualidade;
- Fornecendo orientações para melhor desenvolvimento de práticas;
- Permitindo a migração transparente para novas funcionalidades;

Como descrito, gerenciar bibliotecas ou pacotes de terceiros tem grande importância no momento, permitindo que desenvolvedores se preocupem com o desenvolvimento e não com dificuldades em utilizar uma biblioteca ou pacote. No próximo tópico será abordado outro assunto importante para o acompanhamento do funcionamento da aplicação, item de grande importância para o DevOps, o monitoramento da aplicação.

5.4.4 Monitoramento da Aplicação

New Relic

New Relic é uma ferramenta de análise da aplicação, ajuda pessoas que constroem *software* a entender o que os históricos de dados estão tentando dizer-lhes, através da coleta, armazenamento e análise dos mesmos (*RELIC*, 2015).

Logo da *New Relic* na figura 23.

Figura 23: Logo *New Relic*.



Fonte: *Relic* (2015).

Trata-se de uma ferramenta Multiplataforma (PHP, Java, .NET, *Ruby* e *Python*) capaz de monitorar não só a aplicação, mas também os servidores que a hospedam. O principal propósito do *New Relic* é auxiliar no diagnóstico de possíveis problemas no código da aplicação, facilitando a identificação de gargalos que prejudiquem sua performance (LOCALWEB, 2014).

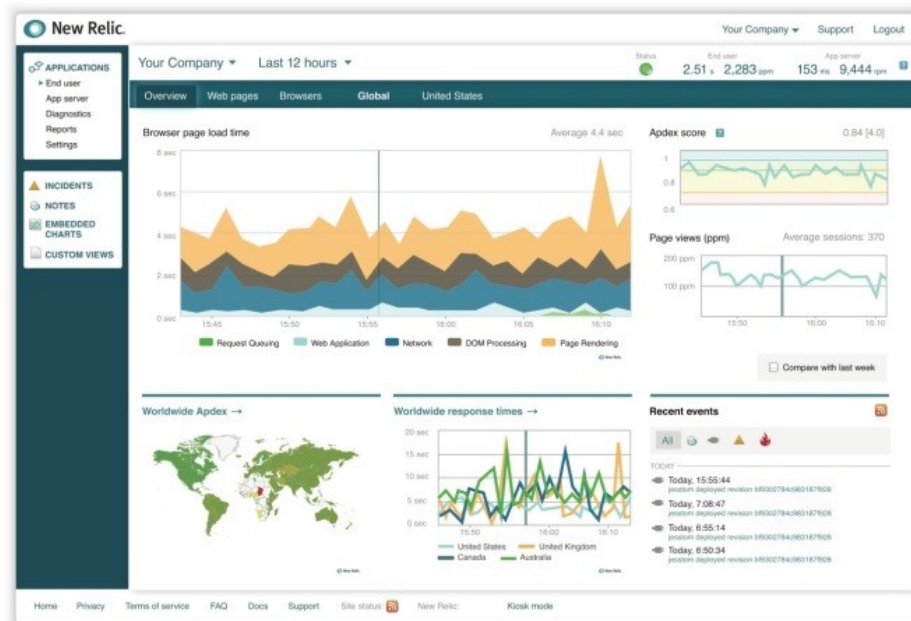
A partir de gráficos de desempenho de fácil compreensão, é possível acompanhar resumos dos tempos de resposta da aplicação, transações com o banco de dados, conexões com serviços externos, visualizar relatórios de disponibilidade e de erros recorrentes, configurar alertas, entre outros (LOCALWEB, 2014).

O *New Relic* oferece vários produtos, como podem ser citados:

- *New Relic APM (Application Performance Monitoring);*
- *New Relic MOBILE;*
- *New Relic INSIGHTS;*
- *New Relic SERVERS;*
- *New Relic BROWSER;*
- *New Relic PLUGINS;*
- *New Relic SYSTHETICS;*

A Figura 24 ilustra a interface do *New Relic*.

Figura 24: Ilustração da interface da ferramenta *New Relic*.



Fonte: Ramos (2014).

Como foi destacado, *New Relic* torna-se uma ótima opção, contendo versões gratuitas ou pagas e contendo um alto número de usuários/clientes. A ferramenta que será descrito a seguir é o Nagios, que apesar de ser uma ferramenta antiga e possuir uma interface gráfica não muito amigável, ele ainda é uma das opções mais populares por ter uma implementação robusta e um ecossistema com grande quantidade de *plugins* escritos e mantidos pela comunidade, afirma Sato (2013).

Nagios

Nagios é um sistema de monitoramento que permite organizações identificar e resolver problemas de infraestrutura de TI, antes que eles afetem os processos de negócios. Lançado em 1999, hoje inclui milhares de projetos desenvolvidos pela comunidade Nagios em todo o mundo. O Nagios permite alertar as pessoas do problema, caso alguma falha aconteça, permitindo que os processos de correção sejam feitos antes que afetem os clientes ou usuários (NAGIOS, 2015).

Figura 25 ilustra o logo do *software* Nagios.

Figura 25: Logo Nagios.



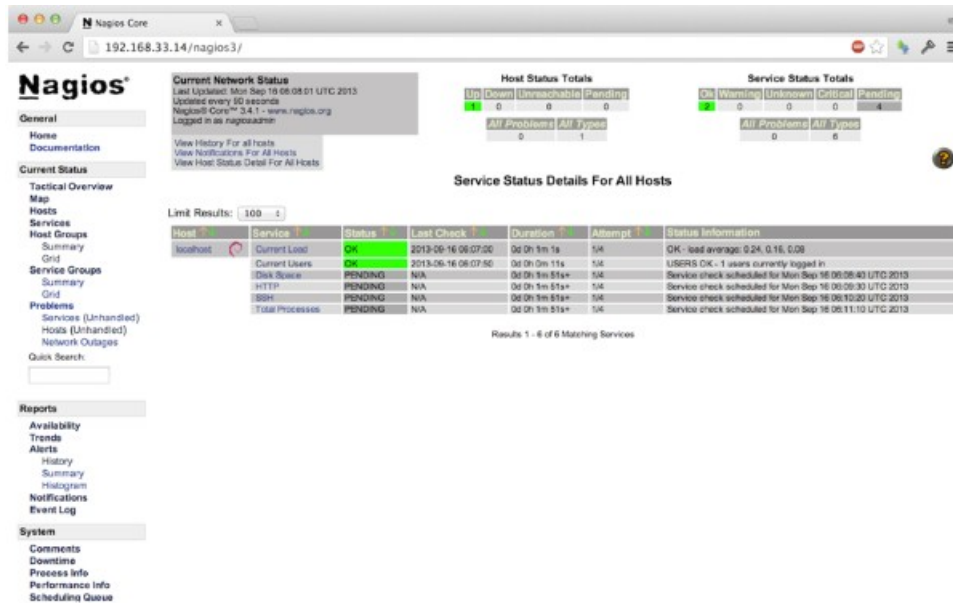
Fonte: Nagios (2015).

Pelo Nagios existem verificações de padrões que são, segundo Sato (2013):

- Carga atual: a carga é uma média móvel do tamanho da fila de processos esperando para executar. Conforme a CPU (*Central processing unit*) começa a ser sobrecarregada, os processos demoram mais pra serem executados fazendo o sistema rodar mais devagar. Esta verificação envia um alerta quando a carga atual ultrapassa um certo limite;
- Usuários atuais: verifica quantos usuários estão logados no sistema;
- Espaço de disco: servidor sem espaço de disco pode ocasionar vários problemas, um deles é a geração de *logs* em sistemas de produção. Para evitar este problema é importante monitorar quanto espaço livre ainda resta, notificando quando o espaço livre atinge um certo limite;
- HTTP: verifica se o servidor está aceitando conexões HTTP, esta verificação enviará alertas caso não consiga abrir uma conexão HTTP na porta 80;
- SSH: verifica se o servidor está aceitando conexões SSH na porta 22 e envia alertas quando encontra algum problema;
- Número total de processos: saber avaliar quando um servidor está sobrecarregado através do número total de processos, torna-se de grande importância, pois com muitos processos executando ao mesmo tempo o SO vai gastar mais tempo gerenciando qual pode rodar na CPU, não sobrando tempo suficiente para que nenhum deles rode efetivamente;

Na Figura a seguir é mostrado a interface de administração do Nagios.

Figura 26: Representação da interface de administração do Nagios.



Fonte: Sato (2013).

Apesar de ser uma ferramenta de monitoração antiga é ainda uma das mais utilizadas por equipes de operações. Com a grande quantidade de ferramentas disponíveis no momento, deve-se analisar as necessidades, custos e facilidade de utilização que a empresa oferece, garantindo que as aplicações sejam monitoradas da melhor forma.

5.5 Considerações Finais

Neste tópico foi destacado diversas ferramentas utilizadas para alcançar alguns objetivos impostos pelo DevOps, como manter ambientes de desenvolvimento e operações iguais através das ferramentas de gerenciamento de ambientes, onde foram citadas o *Vagrant* e o *Docker*, logo após, foi falado sobre ferramentas de gerencia de configuração, para manter configurações únicas permitindo o compartilhamento entre máquinas, servidores ou dispositivos, no qual foram citados o *Puppet* e o *Chef*.

Logo em seguida, levantou-se outra questão importante, que é a gerencia das configurações da aplicação, ferramentas como *Composer*, *Bundler* e *Maven* que são utilizadas para instalação, gerencia e atualização de pacotes ou bibliotecas extras. Por fim foram citadas as ferramentas de monitoramento da aplicação *New Relic* e *Nagios*, que permite ver e analisar como sua aplicação está funcionando.

Estas ferramentas são de extrema importância para que a cultura DevOps seja alcançada, trazendo uma grande quantidade de processos automatizados, mas não é só de ferramentas que o DevOps precisa, são necessários pessoas motivadas e de ambas as partes, tendo em vista que as equipes devem ter a flexibilidade de ceder, para que resultados sejam alcançados. A seguir será apresentado o levantamento e análise dos dados coletados entre CRS e DTI da Unochapecó e também das empresas de desenvolvimento da região do Oeste de Santa Catarina para iniciar a aplicação dos conceitos e ferramentas do DevOps.

6. LEVANTAMENTO E ANÁLISE DE DADOS SOBRE DEVOPS NO CRS/DTI UNOCHAPECÓ

O CRS (Centro de Residência em *Software*) iniciou em 2010 e está localizado na Unochapecó (Universidade Comunitária da Região de Chapecó) bloco G, este projeto tem como objetivo fornecer aos estudantes dos cursos de Ciência da Computação e Sistemas de Informação através da experiência práticas da construção de *software*, criando um diferencial profissional e capacitando para o mercado de trabalho. Os conceitos mais explorados são engenharia de *Software*, linguagens de programação e banco de dados (CRS, 2015).

O DTI (Diretoria de Tecnologia da Informação) está localizada também na Unochapecó bloco B, e esta divisão é responsável pelo desenvolvimento de soluções em sistemas para a instituição.

Para levantamento do problemas encontrados entre a equipe de desenvolvimento do CRS e a equipe de operações da DTI da Unochapecó foi elaborado e aplicado um questionário para estas equipes, onde pode ser visto como são os atuais processos e como é a comunicação entre as partes.

O questionário é composto de 19 questões descritivas e de múltiplas escolhas e além de ter sido aplicado no CRS e no setor de DTI da Unochapecó foi disponibilizado e compartilhado para empresas de desenvolvimento de *software* através do formulários oferecidos pelo Google¹⁴. O formulário está disponível neste *link*¹⁵. No questionário buscou-se identificar como é o atual funcionamento e comunicação das partes no desenvolvimento de um *software*, perguntas como: como é feito o monitoramento de aplicações, os ambientes de desenvolvimento e operações são iguais, como é a comunicação entre as equipes e também, qual é o conhecimento sobre DevOps.

14 <https://www.google.com/forms/about/>

15 <http://goo.gl/forms/5mdIRBOq1p>

6.1 Aplicação do questionário entre CRS e DTI da Unochapecó

Há aplicação do questionário aconteceu para as duas equipes, equipe de desenvolvimento (CRS) e equipe de operações (DTI Unochapecó), onde notou-se algumas diferenças e incertezas de ambas as partes. As questões aplicadas foram:

1. Já ouvi-o falar sobre DevOps? Sim ou não?

Resposta: Não.

2. Se sim, qual é o entendimento sobre o assunto?

Resposta: O entendimento é muito pouco

3. As equipes de desenvolvimento trabalham no mesmo espaço que às equipes de operações? Sim ou não?

Resposta: Não

4. Os ambientes de produção e desenvolvimento são iguais (versões de linguagens, SO, banco de dados entre outros)? Sim ou não, se não quais as principais diferenças?

Resposta: Não, as principais diferenças são em SO (Sistemas operacionais) e pacotes.

5. Atualmente, qual é o principal problema enfrentado nos processos de desenvolvimento de *software*?

Resposta: O principal problema é em fazer *Deploy* (liberação de versão) para produção e a instalação de dependências de pacotes.

6. Como acontece a comunicação entre as equipes envolvidas quando ocorre um problema?

Resposta: E-mail e *Help Desk* (atendimento as solicitações).

7. Existem processos repetitivos? Às equipes avaliam e pesquisam sobre ferramentas que podem agilizar os processos e torná-los menos repetitivos?

Resposta: Sim e Não.

8. Dentre os processos burocráticos que existem, entre equipes de desenvolvimento e equipe de operações, quais são os mais demorados?

Resposta: Compra de *software*, pois precisa de liberações da instituição para fazer.

9. Qual é o tempo médio para o atendimento de solicitações entre às equipes de desenvolvimento e operações?

Resposta: Dependendo o caso, geralmente de 1 à 2 dias.

10. Quais ferramentas são utilizadas hoje pra desenvolver e manter o(s) *software*?

R: SVN, Nagios, PHP, Doctrine, Zend *Framework*, PostgreSQL e Composer.

11. É usado alguma ferramenta para liberação de versões dos *software*?
Sim ou não? Se sim quais?

Respostas: SVN (Controle de versão).

12. É usado alguma ferramenta para monitorar as aplicações?
Sim ou não? Se sim quais?

Resposta: Não.

13. Como é feito o monitoramento das aplicações?

Resposta: O monitoramento de aplicações não é realizado.

14. Às equipes de operações e de desenvolvimento participam de quais etapas do desenvolvimento de um sistema?

Resposta: Depende do sistema todos participam, sistemas semelhantes não.

15. Às equipes de desenvolvimento e operações tomam decisões em conjunto referente a um determinado *software*?

Resposta: Depende do caso, geralmente sistemas novos.

16. Às equipes de desenvolvimento e operações utilizam alguma metodologia de desenvolvimento? Se sim, qual?

Resposta: Sim, SCRUM no desenvolvimento, mas a equipe de operação não participa.

17. Com que frequência é feita liberação de versão do(s) sistema(s)/*software*?

- A cada dia;
- De 3 em 3 dias;
- A cada semana;
- A cada mês;
- A cada etapa;
- Outro;

Resposta: A cada semana.

18. Quando são encontrados erros, eles são resolvidos e atualizados imediatamente ou são acumulados para próxima entrega?

Resposta: Dependendo a gravidade é resolvido imediatamente.

19. Analisando às equipes e os profissionais envolvidos, eles tem motivação e facilidade para mudanças em processos e projetos que possa trazer melhorias? Sim ou não, por quê?

Resposta: Acredito que não, existe muita burocracia envolvida para se criar processos novos, isso dificulta bastante as mudanças e acaba desmotivando as pessoas a buscarem soluções melhores.

Analisando as respostas alcançados através do questionário aplicado pode-se perceber que existem claramente uma grande distância entre as equipes, resultando em problema já citados anteriormente, no tópico 6.3 será levantado e analisado com mais detalhes os problemas encontrados entre o CRS e DTI da Unochapecó. A seguir serão mostrados os dados do questionário das empresas da região do oeste de Santa Catarina, trazendo também um pouco das suas realidades e entendimentos sobre o assunto.

6.2 Aplicação do questionário para empresas de desenvolvimento da região do Oeste de Santa Catarina sobre DevOps

Em busca de um melhor entendimento e conhecimento sobre DevOps na região do oeste de Santa Catarina foi desenvolvido o questionário onde foi distribuído para mais de 25 empresas de desenvolvimento de *software*, a divulgação aconteceu por meios digitais, através de e-mail, redes sociais e pessoalmente com pessoas de contatos próximos.

Com 11 respostas obtidas na pesquisa, pode-se identificar pouco conhecimento das empresas entrevistadas sobre DevOps, e apresentaram vários problemas que a cultura DevOps poderia auxiliar a resolver, como: distância entre equipe de Desenvolvimento e equipe de Operação, intervalos grandes entre entregas ou liberações de *software*, diferenças entre ambiente de desenvolvimento para o ambiente de produção e o mau monitoramento dos *software*/aplicações. As 18 questões aplicadas foram:

1. Já ouvi-o falar sobre DevOps? Sim ou não?

Das 11 resposta 5 disseram que nunca ouviram falar sobre DevOps e os outros 6 já ouviram falar e tem um certo entendimento sobre o assunto, como mostra a figura 27.

Figura 27: Dados questão 1.



Fonte: Autor.

2. Se sim, qual é o entendimento sobre o assunto?

E para os que disseram sim, mostraram que tem entendimento sobre DevOps, apresentando as seguintes respostas:

- Atividades integradas entre a equipe de desenvolvimento e de analistas de suporte;
- Uma sinergia entre os desenvolvedores e equipe de suporte no sentido de entregar artefatos de *software* com grande valor agregado em curto espaço de tempo ao usuário final;
- Pouco, apenas ouvi algumas coisas;
- Essencialmente o DevOps tem como objetivo diminuir a distância entre ambientes desenvolvimento e produção, existem várias razões porque isso é importante, mas as principais na minha opinião é justamente a parte de automatização de processos para que o ciclo de liberação de versões se torne mais eficiente e rápido, seguindo geralmente metodologias ágeis. Outra questão é redução do impacto entre cada ambiente produção/desenvolvimento, com incompatibilidade de versões, problemas específicos de plataforma e por aí vai;
- Muito pouco;
- É a metodologia (com profissionais específicos ou somente ferramentas de automação) que facilitam a integração e testes do *software* em desenvolvimento para o seu *deploy*;

3. As equipes de desenvolvimento trabalham no mesmo espaço que às equipes de operações? Sim ou não?

6 afirmaram que às equipes de desenvolvimento e equipes de operações trabalham no mesmo espaço e 5 responderam que as equipes não trabalham no mesmo espaço de trabalho.

4. Os ambientes de produção e desenvolvimento são iguais (versões de linguagens, SO, banco de dados entre outros)? Sim ou não, se não quais as principais diferenças?

Sobre as diferenças entre os ambientes de produção e desenvolvimento 7 afirmaram que os ambientes são iguais e 4 que não e que as principais diferenças são entre versões de sistemas operacionais e pacotes.

Figura 28: Dados questão 4.



Fonte: Autor.

5. Atualmente, qual é o principal problema enfrentado nos processos de desenvolvimento de *software*?

Perguntados dos principais problemas enfrentados nos processos de desenvolvimento de *software*, foram apresentadas as seguintes respostas:

- Padronização e qualidade de desenvolvimento de *software*;
- Levantamento de requisitos, pois as empresas tem dificuldade de expor e entender o que elas realmente precisam em seu sistema. Já trabalhamos com empresas sem estrutura alguma de TI e empresas até com gestores de projetos, focados só em acompanhar a criação do nosso *software*. Ambas tiveram retrabalho devido não saberem o que queriam;
- O problema é falta de preocupação com escala antes de sequer começar a desenvolver a aplicação. Hoje em nossa empresa existe falta de padrões de ambientes/plataformas e

também dependência de plataforma (sistema operacional);

- Entendo que a sincronização dos ambientes de desenvolvimento e produção, de forma que seja transparente ao usuário final do *software*;
- Falta de análise;
- Falta de equipe qualificada, os mesmos profissionais intercalam no desenvolvimento e implantação;
- Burocracia e demora em testes e *deploy*;
- Gestão de erros na liberação de controle de versões;
- Automação de testes para envio para produção;
- Atendimento a cronograma;
- *Deploy* para a versão de produção, instalação de dependências de pacotes;

Dentre as respostas pode ser destacado a falta de padronizações em processos, dificuldades em fazer *Deploy* para produção, devido também pelas diferenças entre os ambientes e dependências de pacotes, fazendo que os cronogramas fiquem sempre apertados e deixando os clientes insatisfeitos.

6. Como acontece a comunicação entre as equipes envolvidas quando ocorre um problema?

Perguntado sobre como acontece a comunicação entre às equipes quando o ocorre um problema, as respostas foram:

- Gerenciador de Tarefas;
- Verbalmente, por e-mail ou com uma ferramenta de controle de trabalho;
- Se ocorre em ambiente de produção se torna uma urgência, então é resolvido com reuniões e ligação (contato direto) para resolução. Em ambiente de teste ocorre de forma menos urgente, onde pode ter um analista como intermediário para resolução do problema e determinação de ações;
- Via abertura de chamado/tarefa;
- E-mail, *HelpDesk*;
- Através de metodologia própria de desenvolvimento;
- E-mail;
- De forma direta;

- O Gestor de projetos distribui as tarefas e coordena as equipes. Ele entende a situação e momento atual de cada uma delas, interpretando e tomando decisões. Caso ocorram problemas estes são apontados e discutidos junto com a direção da empresa;
- De forma verbal;
- O suporte analisa incidentes através do Zabbix (monitoramento) ou em chamado do próprio cliente, tenta resolver e se não conseguir, aciona o desenvolvimento;

Das formas apresentadas as principais foram através de e-mail, *help desk*, ferramentas de gerenciamento de tarefas, chamadas para equipe de desenvolvimento e também verbalmente por meio do diálogo entre os profissionais, o que ajuda muito para conclusão precisas sobre as tomadas de decisões.

7. Existem processos repetitivos? Às equipes avaliam e pesquisam sobre ferramentas que podem agilizar os processos e torná-los menos repetitivos?

Sobre os processos repetitivos e se é avaliado formas de agilizar os mesmos:

- Utilizamos diversos tipos de *frameworks* e algumas metodologias para agilizar o desenvolvimento. Sempre estamos buscando novidades neste sentido. Mas mesmo assim acabamos encontrando no dia a dia alguns processos repetitivos;
- Existem. Avaliam, mas não colocam em prática;
- Existem vários processos repetitivos;
- Existem processos repetitivos sim, as equipes não analisam tanto quanto deveriam. Hoje isso está mudando, muita coisa está sendo automatizada, mas ainda tem um bom caminho pela frente para engajar mais na cultura do DevOps;
- Existem. Sim existe avaliação;
- Sim e não;
- Temos processos repetitivos nos testes de *deploy* das aplicações. Possíveis soluções para automação são testadas ocasionalmente para avaliar se podem auxiliar no processo;
- Muito pouco;
- Existem. Na medida do possível, sim;

A maioria afirmou que, existem sim processos repetitivos, mas que geralmente não são avaliados e colocados em prática pra melhorar os processos.

8. Dentre os processos burocráticos que existem, entre equipes de desenvolvimento e equipe de operações, quais são os mais demorados?

Os processos burocráticos citados foram:

- Envio para produção (*deploy*);
- Compra de *software*;
- Sincronização dos ambientes;
- Aprovação do cliente é um item que impacta bastante no tempo do projeto;
- Geração de Manuais;
- Correções de *bugs* e atendimento de novas necessidades;
- Backup de aplicativos antes de uma atualização e processo de atualização em si. Como na maioria dos casos não são aplicativos compilados, existem vários arquivos necessários para validação no momento do *deploy*;
- Preparar ambiente, configurações, binários e dependendo a necessidade, escalar um servidor novo para ambiente de produção e configurá-lo;
- O entendimento do que o cliente espera;
- A parte de documentação dos processos;
- Não existe;

9. Qual é o tempo médio para o atendimento de solicitações entre às equipes de desenvolvimento e operações?

E o tempo médio para os atendimentos das solicitações:

- Variável depende de equipe para equipe em média 2 horas;
- 3 dias;
- 2 dias;
- 1 dia;
- 5 dias;
- 15 dias;
- 1 dia;
- 1 à 2 dias;

10. Quais ferramentas são utilizadas hoje pra desenvolver e manter o(s) *software*?

As ferramentas utilizadas para desenvolver e manter o(s) *software*:

- C# Web / Visual Studio; Git; Scripts bat - para cópia de binários para os servidores;
- PHP, MySQL, Workbench, Dreamweaver, Netbeans, GitHub, Vagrant;
- Oracle Forms&Reports e Oracle PLSQL;
- NetBeans; pgAdmin III;
- PHP, Doctrine, Zend Framework, SVN, PostgreSQL e Composer, Nagios;
- Ferramentas de desenvolvimento; Ferramentas de banco de dados; Ferramentas para gestão de servidores e hospedagem; Ferramentas de gerenciamento de projetos;
- Redmine e sistema interno de chamados;
- Controle de Versão, Zend Framework 2;
- Text Mate; Chrome; Nginx; Ruby on Rails; Postgres;
- IDE's com suporte a controle de versão;

11. É usado alguma ferramenta para liberação de versões dos *software*?

Sim ou não? Se sim quais?

Figura 29: Dados questão 11.



Fonte: Autor.

E as ferramentas citadas foram:

- Script, não é automático, mas é feito por bat;
- Subversion e ERP próprio;
- Git e Github;
- Git;
- SVN;

12. É usado alguma ferramenta para monitorar as aplicações?

Sim ou não? Se sim quais?

Figura 30: Dados questão 12.



Fonte: Autor

As ferramentas utilizadas pra monitorar as aplicações:

- Próprio (manualmente);
- *New Relic*;
- *Software do Linux, Uptime*;
- Zabbix;
- Interna;

Monitoramento é um dos itens que não levam muita atenção e que muitos não usam uma ferramenta específica, muitas vezes o monitoramento da aplicação é feito manualmente.

13. Como é feito o monitoramento das aplicações?

O monitoramento das aplicações são feitas através de:

- Acompanhamos CPU, disco, memória, link;
- Cada *software* em produção roda como serviço no *windows* e são monitorados através do Zabbix. Por enquanto o máximo de informações que conseguimos saber é disponibilidade ou não do serviço que é verificada a cada 30 segundos;
- O monitoramento de aplicações não é realizado;
- Relatórios mensais feito por DBA terceiro especializado;
- Programa que verifica a cada minuto se as aplicações encontram-se ativas;
- Usamos vários fornecedores de infraestrutura e usamos as ferramentas / notificações disponibilizadas por ele (quando existe);
- Não sei afirmar;
- Manualmente;

- Através do *software New Relic*. Também temos uma aplicação para controlar alguns pontos dos vários servidores de aplicações de clientes;
- Utilizando de aplicativos internos;

14. Às equipes de operações e de desenvolvimento participam de quais etapas do desenvolvimento de um sistema?

Respostas:

- Desenvolvimento acompanha todo o projeto. Operações participam de reuniões específicas;
- Todas posterior a análise;
- Sim;
- Todas abaixo do Gestor de Projetos;
- Sim;
- Desenvolvimento atualmente faz apenas liberação por etapas/módulos. Como o *software* depende de APIs externas, o próprio desenvolvimento já homologa as mesmas. O time de suporte e operações coloca o *software* em produção e monitora;
- Dependendo do sistema, todos participam;
- Todos;
- Todas;
- Ambas;

15. Às equipes de desenvolvimento e operações tomam decisões em conjunto referente a um determinado *software*?

10 afirmaram que sim e 1 afirmou que não.

Figura 31: Dados questão 15.



Fonte: Autor.

16. Às equipes de desenvolvimento e operações utilizam alguma metodologia de desenvolvimento? Se sim, qual?

Figura 32: Dados questão 16.



Fonte: Autor.

E as metodologias foram:

- Scrum;
- Scrum;
- Scrum;
- Scrum adaptado;
- Ágil;
- Um padrão próprio;
- Scrum na de desenvolvimento, mas na de operações não participa;
- Pegamos o melhor de várias metodologias e formamos a nossa. Que está em constante evolução;

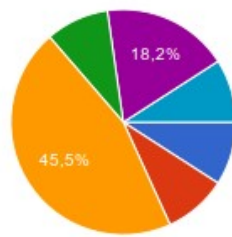
Pode notar que muitos utilizam a metodologia ágil Scrum, mas apenas entre os desenvolvedores, sendo que a equipe de operações não participa.

17. Com que frequência é feita liberação de versão do(s) sistema(s)/software?

- A cada dia;
- De 3 em 3 dias;
- A cada semana;
- A cada mês;
- A cada etapa;
- Outro;

Como mostra a figura 33:

Figura 33: Dados questão 17.



Fonte: Autor.

As liberações de *software* quase 50% das respostas foram que acontecem a cada semana, o que representa que a cada *deploy* muitas coisas são entregues e abre possibilidades de erros serem repostados, pela falta de validações.

18. Quando são encontrados erros, eles são resolvidos e atualizados imediatamente ou são acumulados para próxima entrega?

Respostas:

- Geralmente é feito imediatamente a não ser que seja realmente não tenha impacto significativo;
- Resolvidos imediatamente;
- Depende a gravidade é resolvido imediatamente;
- Imediatamente;
- Depende da urgência;
- Depende do projeto e nível do erro, se for um erro crítico é ajustado na mesma hora, mas se forem melhorias são acumuladas e feitas em até um ou dois dias;
- Se for grave é corrigido, caso contrário fica para a próxima entrega;
- Dependendo dos problemas;
- Imediato;
- São resolvidos;
- Resolvido o mais breve possível;

19. Analisando às equipes e os profissionais envolvidos, eles tem motivação e facilidade para mudanças em processos e projetos que possa trazer melhorias? Sim ou não, por quê?

Respostas:

- Não. Há um padrão de desenvolvimento ligado a um ERP (*Enterprise Resource Planning*), ambos próprios, que dificulta mudanças ou alterações;
- Sim. Embora ainda falte um pouco de atitude e maturidade da equipe para ser mais pró-ativa e crítica;
- Sim, é uma equipe nova e motivada;
- Sim, nossa equipe aceita muito bem mudanças de metodologia e novas formas de trabalho e tecnologias. Estão em constante aprendizado e evolução junto com a empresa, acompanhando a tecnologia;
- Não. A empresa como um todo, tem muita dificuldade em mudança de processo;
- Sim e não, para algumas coisas sim e para algumas coisas não;
- Sim, existe a motivação, a dificuldade maior é sempre criar uma rotina (cultura) para os novos processos;
- Sim, trabalhando motivados;
- Sim;
- Acredito que não, existe muito burocracia envolvida para se criar processos novos, isso dificulta bastante às mudanças e desmotiva às pessoas a buscarem soluções melhores;
- Sim, a equipe é bastante comprometida e capacitada;

Com o questionário percebe-se que muitas das empresas que participaram da pesquisa ainda mantêm culturas antigas, onde o desenvolvedor apenas desenvolve o que foi mapeado diante dos analistas ou profissionais de operações, o que acaba dificultando justamente a comunicação de maneira clara e entendível de ambas as partes.

A comunicação, liberações de *software* em intervalos de tempos grandes, a distância entre profissionais do desenvolvimento e profissionais de operações acabam acarretando em problemas que hoje já não são mais permitidos, causando perda de credibilidade e clientes, em um mercado aquecido como *mobile*, *web* entre outros, ficar parado com processos burocráticos, acaba-se ficando para trás. No tópico a seguir será abordado os problemas identificados

especificamente entre o Centro de Residência em *Software* (CRS) e TI da Unochapecó.

6.3 Identificação dos problemas enfrentados entre CRS e DTI da Unochapecó

Após a aplicação do questionário de pesquisa para o CRS e DTI da Unochapecó, onde pode-se ver o atual funcionamento e relação entre os Devs e Ops, visando analisar o que pode ser feito e melhorado, foi levantado as principais dificuldades vistas que serão apresentadas a seguir.

Dificuldades identificadas depois do questionário, aplicado na DTI e CRS.

- Contato só existe quando abrem-se chamados, a comunicação entre operações e desenvolvimento acontece na maioria das vezes por meios eletrônicos, através de e-mail e *help desk*, isso acaba criando distância entre as equipes;
- Equipe de Operações (ops) desconhece metodologias e ferramentas que a equipe de Desenvolvimento (devs) utilizam, a equipe de operações não tem conhecimento dos processos, metodologias e ferramentas que os desenvolvedores do CRS utilizam;
- Equipe de Desenvolvimento (devs) desconhece também o papel da equipe de Operações (ops), da mesma forma que o item acima, os desenvolvedores não tem entendimento dos papéis que a equipe de operação faz;
- Os dois fatores acima causam diferenças entre os ambientes, diferenças entre ambientes podem causar vários problemas, os quais já foram citados no capítulo 2 Apresentação dos diferentes ambientes de desenvolvimento de *software*, mas as causas das diferenças entre o ambiente de desenvolvimento e operações é devido a falta de entendimento entre as equipes;
- Em poucos casos o Devs e Ops decidem os projetos juntos, outro item identificado é a falta dos Devs e Ops participarem nas decisões juntos adiantes dos projetos existentes;
- Monitoramento da aplicação não é feito, as aplicações desenvolvidas não são monitoradas, isso trás, dificuldade em analisar performance, disponibilidade de funcionalidades e além de deixar clientes insatisfeitos;

6.4 Considerações Finais

Nos tópicos citados a cima pode-se, através do questionário realizado para empresas da região do Oeste de Santa Catarina, incluindo o CRS e a DTI da Unochapecó, ver como é o atual funcionamento do CRS com relação a DTI da Unochapecó para o desenvolvimento e manutenção dos sistemas de *software*, além das indagações diante do entendimento sobre DevOps e a motivação das equipes envolvidas.

A pesquisa feita para as empresas de TI da região do Oeste de Santa Catarina, mostrou-se importante pois, o conhecimento, entendimento e aplicação dos conceitos impostos pelo DevOps, ainda não são tão explorados e poucas empresas apresentaram um relacionamento direto e claro entre a equipe de operações e a equipe de desenvolvimento, o que mostra a existência de problemas identificados também no CRS e DTI da Unochapecó.

No capítulo a seguir serão abordadas as etapas de aplicação em um ambiente real (CRS e DTI da Unochapecó), sendo que os itens mais explorados são: apresentação dos conceitos e propostas do DevOps, monitoramento de aplicações e padronização dos ambientes.

7. APLICAÇÃO DO DEVOPS NO CRS/DTI DA UNOCHAPECÓ

Após a identificação dos problemas através do questionário aplicado, inicia-se a parte de aplicação do DevOps no CRS e DTI da Unochapecó. No primeiro tópico é abordado sobre a aplicação dos conceitos do DevOps, como a **cultura** (colaboração, comportamento, flexibilidade), **automação**, (ferramentas, automatizar processos), **medição/avaliação**, (medir performance, processos e pessoas), **compartilhamento** (comunicação, incentivação, compartilhamento de ideias) apresentados para a equipe de operações e desenvolvimento.

No segundo tópico é a utilização de uma ferramenta de monitoramento, a ferramenta utilizada foi *New Relic*, uma ferramenta sem custo e com possibilidade de compras de serviços, onde pode-se monitorar várias aplicações, neste caso foi feito o monitoramento da aplicação InfoHab¹⁶, sistema desenvolvido pelo CRS.

No terceiro tópico é utilizado um ferramenta para padronização dos ambientes, a ferramenta *Docker* é utilizado para manter os ambientes iguais, evitando diferenças de versões e facilitando para liberações de versões (*deploys*).

7.1 Aplicação dos conceitos para CRS e DTI da Unochapecó sobre o DevOps

Os conceitos do DevOps não são difíceis de serem entendidos, a falta de motivação e estrutura organizacional pode muitas vezes tornar-se um ponto decisivo para aplicação da cultura, no CRS e DTI por estarem em locais físicos diferentes e em um meio corporativo, onde a TI não é o foco da instituição, são fatores que dificultam os processos de melhorias.

A aplicação dos conceitos sobre DevOps para o CRS e DTI aconteceu da seguinte forma, inicialmente aconteceram 2 encontros entre os responsáveis pelas áreas de desenvolvimento, operações e acadêmico, o primeiro encontro foi entre o acadêmico e o responsável pela equipe de operações da DTI da Unochapecó, para aplicação do questionário e o segundo foi entre o acadêmico, responsável de operações e de desenvolvimento, para análise do que poderia ser feito em relação as diferenças do ambientes.

Depois disso foram distribuídos materiais para todos os envolvidos, equipe de desenvolvimento e a equipe de operações, estes materiais eram compostos por 3 documentos, o

¹⁶ <http://infohab.org.br>

primeiro era constituído do assunto DevOps, onde é apresentados os itens:

- Introdução sobre DevOps, movimento cultural que foca no aperfeiçoamento da comunicação, colaboração e integração, fazendo com que desenvolvedores e engenheiros de TI trabalhem juntos;
- Como surgiu o DevOps, pro volta de 2009 através da infraestrutura ágil, onde Patrick Debois criador do DevOpsDay foi uma das pessoas que mais auxiliaram e auxiliam na disseminação do DevOps pelo mundo;
- Conceitos/pilares do DevOps, conhecidos pelas siglas C.A.M.S, são eles cultura, automação, medição/avaliação e compartilhamento, citados no tópico 5.3;
- Ferramentas que possam auxiliar, ferramentas para padronização de ambientes (*vagrant* e *docker*), ferramentas para monitoramento de aplicações (*new relic*), ferramentas para gerenciamento de configurações (*chef* e *puppet*) e para gerenciamento de pacotes (*composer*);
- E referências e links úteis para leitura de DevOps, dentre as principais foram: *Blog* do Patrick Debois¹⁷, O que é DevOps afinal? Do Guto Carvalho¹⁸, DevOps Brasil¹⁹ dentre outros;

O segundo documento era o tutorial sobre a instalação e utilização da ferramenta de monitoramento de aplicações, que foi aplicada no CRS o *New Relic*, este tutorial contém um sequência de passos a serem executados, que vai desde o cadastro no *site* do *New Relic* até a configurações de várias aplicações para monitoramento e um breve descrição das funcionalidades oferecidas pelo *New Relic*, este documento pode ser usado e distribuído para qualquer um que deseje trabalhar com o *New Relic*, isso será abordado no tópico 7.2 Aplicação da ferramenta de Monitoramento *New Relic* no CRS.

No terceiro e último documento é o tutorial sobre a instalação do *Docker*, ferramenta que permite criar máquinas com configurações de aplicação e serem distribuídas de maneira fácil e rápida, tornando os ambientes de desenvolvimento e produção iguais e padronizados, evitando diferenças de versões, com o tutorial é possível instalar o *Docker*, baixar o repositório criado para o CRS, criar um conta no *Hub Docker*, onde permite a criação de novos repositórios e por fim, como criar um máquina e mandar-lha para o repositório, isso será apresentado no tópico 7.3. Da mesma forma que o tutorial do *New Relic*, este também pode ser distribuído.

¹⁷ <http://www.jedi.be/blog/>

¹⁸ <http://gutocarvalho.net/octopress/2013/03/16/o-que-e-um-devops-afinal>

¹⁹ <http://devops-br.org/>

Feito isso, foi marcado um encontro no CRS para todos os envolvidos, nem todos puderam se fazer presente, devido ao horário de trabalho ou compromissos, neste encontro foram apresentados os principais conceitos do DevOps, sem se estender muito e tornando o encontro em uma conversa e troca de ideias entre as pessoas, foram abordados os temas: motivação para mudanças, automatização de processos, monitoramento, ferramentas disponíveis, na qual foi citada o *composer*, ferramenta que já é usada no CRS para aplicações PHP, onde permite instalação de pacotes e dependências extras. Abordou-se também das ferramentas aplicadas o *New Relic* e *Docker*, mas principalmente a mudança de cultura para se adequar ao que o mercado necessita.

No próximo tópico será apresentando as etapas de aplicação da ferramenta *New Relic*, onde são levantados: informação sobre a ferramenta, instalação e configuração para fazer o monitoramento com *New Relic* e por fim falando de como foi utilizado no CRS e da DTI na Unochapecó.

7.2 Aplicação da ferramenta de Monitoramento *New Relic* no CRS

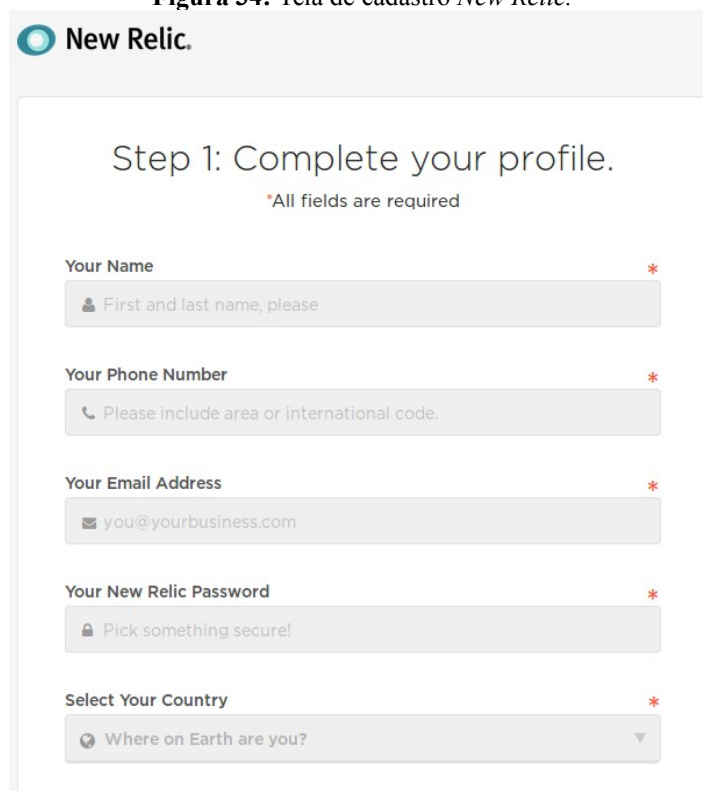
New Relic foi fundada em 2008 para fornecer um gerenciamento/monitoramento avançado de desempenho de aplicações para empresas de qualquer tamanho, hoje possui, segundo *Relic* (2015):

- 250 mil usuários;
- 4 mil aplicações monitoradas por dia;
- 1 bilhão de aplicações *mobile* monitoradas;
- 690 bilhões de métricas por dia;

Esta ferramenta permite monitorar aplicativos oferecendo vários tipos de produtos, dentre eles para: *mobile*, *browser*, *servers*, *apm* (*application performance monitoring*) dentre outros. Para utilização da ferramentas é necessário seguir as seguintes etapas, este tutorial foi aplicado em um ambiente *Linux* (Ubuntu) para o monitoramento de aplicações PHP:

1 – Cadastro no Site²⁰;

Figura 34: Tela de cadastro *New Relic*.



The screenshot shows the New Relic sign-up interface. At the top, the New Relic logo is visible. Below it, the heading "Step 1: Complete your profile." is displayed, followed by the note "*All fields are required". The form consists of five input fields, each with a red asterisk indicating it is required:

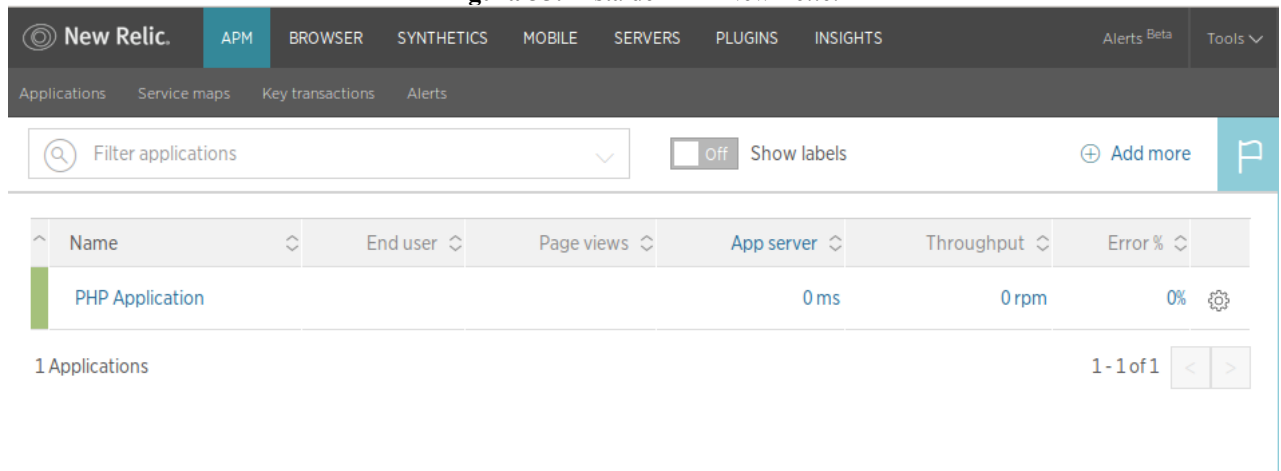
- Your Name:** A text input field with a person icon and placeholder text "First and last name, please".
- Your Phone Number:** A text input field with a phone icon and placeholder text "Please include area or international code".
- Your Email Address:** A text input field with an email icon and placeholder text "you@yourbusiness.com".
- Your New Relic Password:** A text input field with a lock icon and placeholder text "Pick something secure!".
- Select Your Country:** A dropdown menu with a globe icon and placeholder text "Where on Earth are you?".

Fonte: Autor.

Neste cadastro é solicitado as informações pessoais, empresa em que está atuando, quantidade de servidores, pessoas da empresa e por fim é perguntado quais plataformas gostaria de monitorar, dentre as opções: PHP, Java, Ruby, Dotnet, NodeJs, Android e IOS, neste caso foi escolhido PHP.

2 – Após o cadastro é necessário a instalação do “agent” PHP, escolhendo a opção *Add more*, mostrado na figura 35;

²⁰ <https://newrelic.com/signup>

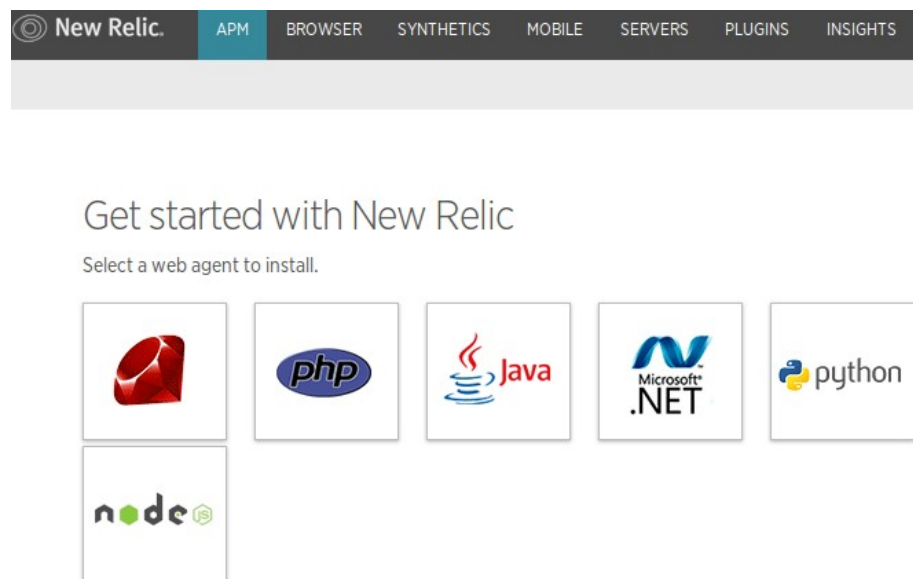
Figura 35: Lista de APM *New Relic*.


Name	End user	Page views	App server	Throughput	Error %
PHP Application			0 ms	0 rpm	0%

1 Applications 1 - 1 of 1

Fonte: Autor.

Neste momento será apresentado as opções dos “*agent*” disponíveis para fazer o monitoramento das aplicações, para este caso foi escolhido a opção PHP.

Figura 36: Lista de *agent* oferecidos.

Fonte: Autor.

3 – Escolhendo a opção PHP é apresenta as etapas para instalação do *agent* PHP;

Figura 37: Tela inicial PHP *agent*.

Install the **PHP** agent

Before you begin

You will need:

1. Administrator access to the computer on which you will install.
2. Ability to configure any firewalls or proxies to allow the agent to report data to New Relic.

Fonte: Autor.

3.1 – *Get your license key* (pegar chave de licença);

Figura 38: Chave de licença *New Relic*.

Your license key is:

-----c9-----16-----66f-----71-49-9b0-----d372-----074-----

Fonte: Autor.

Percebe-se que foi gerado uma chave, que representa a licença para utilização. Esta chave é utilizada para a instalação e configurações do “*agent*”, link de ajuda²¹. Na figura 39 são apresentados os comandos para instalação do *agent New Relic* que é um componente para a configuração com o PHP, acesse com o usuário Administrador.

3.2 – *Install the PHP agent* (instalar *agent* PHP);

3.2.1 – *Select your environment* (selecione o ambiente);

²¹ <https://docs.newrelic.com/docs/agents/php-agent/installation/php-agent-installation-ubuntu-debian#repo>

Figura 39: Opções de ambientes *New Relic*.


☐ RPM-based (Redhat, Centos, etc.)
☒ Debian-based (Debian, Ubuntu, etc.)
☐ All others (using tarballs)

Fonte: Autor.

3.2.2 – *Debian-based* (Debian, Ubuntu, etc.) (base debian) *Get the key* (pegar chave);

Quadro 1: Comando pegar chave *New Relic*.

```
wget -O - https://download.newrelic.com/5.12.0/debian/newrelic.gpg | sudo apt-key add -
```

Fonte: Autor.

3.2.3 – *Add our repository* (adicionar repositório);

Quadro 2: Comando adicionar repositório *New Relic*.

```
sudo sh -c 'echo "deb http://apt.newrelic.com/debian/ newrelic non-free"
> /etc/apt/sources.list.d/newrelic.list'
```

Fonte: Autor.

3.2.4 – *Install de package and configure it* (instalação e configuração);

Quadro 3: Comando instalação *New Relic*.

```
sudo apt-get update
sudo apt-get install newrelic-php5
sudo newrelic-install install
```

Fonte: Autor.

Nestes comando serão solicitados as chaves de licença do usuário e o nome da aplicação inicial para o monitoramento, seu nome vem por padrão “*PHP Application*”.

3.3 – *Add your PHP license key* (adicionar chave de licença no arquivo newrelic.ini ou php.ini);

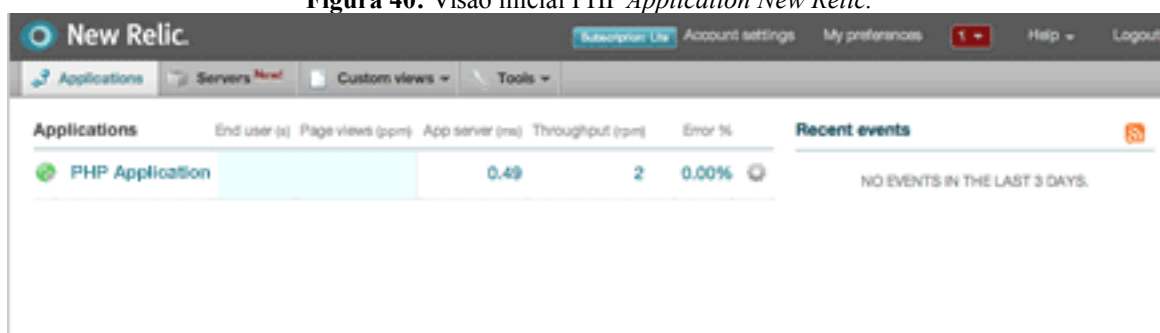
Quadro 4: Comando adicionar chave licença *New Relic*.

```
newrelic.license="609c9d016a4566f5471649c9b0c4dd372a85074e"
```

Fonte: Autor.

3.4 – *Stop and Start your PHP services* (parar e iniciar serviços PHP);

Figura 40: Visão inicial *PHP Application New Relic*.



Fonte: Autor.

4 – *Restart os web server*, no Ubuntu com os comandos;

Quadro 5: Comando reiniciar New Relic.

```
sudo service newrelic-daemon restart
```

Fonte: Autor.

Quadro 6: Comando reiniciar apache.

```
sudo service apache2 restart
```

Fonte: Autor.

5 – O quinto passo é aguardar 5 minutos para que o *New Relic* comece a receber dados. Com as instalações e configurações feitas corretamente (mais a frente será mostrado a configuração com várias aplicações).

6 – Informar a chave do usuário, neste passo é necessário informar a chave em um arquivo;

Quadro 7: Comando abrir arquivo configuração *New Relic*.

```
sudo nano /etc/newrelic/nrsysmond.cfg
```

Fonte: Autor.

- Alterar na linha onde é encontrado `license_key=CHAVE_USUARIO`, passando a chave de licença;

7 – Com os passos seguidos acima o monitoramento das aplicações com *New Relic* está quase pronto, basta informar quais são as aplicações que serão monitoradas, existe algumas maneiras de fazer isso, através do `.htaccess` da aplicação, pelo próprio PHP da aplicação ou pela configuração dos *hosts*, que será o exemplo mostrado na figura 41. No arquivo de configurações dos *hosts* do Apache, localizado em `/etc/apache2/sites-available/000-default.conf` foram criados dois *hosts*.

Figura 41: Exemplo *hosts* New Relic.

```

31 <VirtualHost sga:80>
32     ServerName sga.dev
33     DocumentRoot /var/www/html/sga/public
34     SetEnv APPLICATION_ENV "development"
35     SetEnv PROJECT_ROOT "/var/www/html/sga/public"
36     <IfModule php5_module>
37         php_value newrelic.appname "SGA"
38     </IfModule>
39     <Directory /var/www/html/sga/public>
40         DirectoryIndex index.php
41         AllowOverride All
42         Order allow,deny
43         Allow from all
44     </Directory>
45 </VirtualHost>
46
47 <VirtualHost agm:80>
48     ServerName agm.dev
49     DocumentRoot /var/www/html/agm/public
50     SetEnv APPLICATION_ENV "development"
51     SetEnv PROJECT_ROOT "/var/www/html/agm/public"
52     <IfModule php5_module>
53         php_value newrelic.appname "AGM"
54     </IfModule>
55     <Directory /var/www/html/agm/public>
56         DirectoryIndex index.php
57         AllowOverride All
58         Order allow,deny
59         Allow from all
60     </Directory>
61 </VirtualHost>
62

```

Fonte: Autor.

Percebe-se que a única diferença é o IfModule, onde é informado o nome da aplicação definida pelo usuário. E no arquivo `/etc/hosts` ficará como na figura 42, onde é informado um endereço para os ips para as aplicações sga e agm.

Figura 42: Exemplo endereços *hosts* *New Relic*.

1	127.0.0.1	localhost
2	127.0.1.1	alefe
3	127.0.1.1	sga
4	127.0.1.2	agm
5		

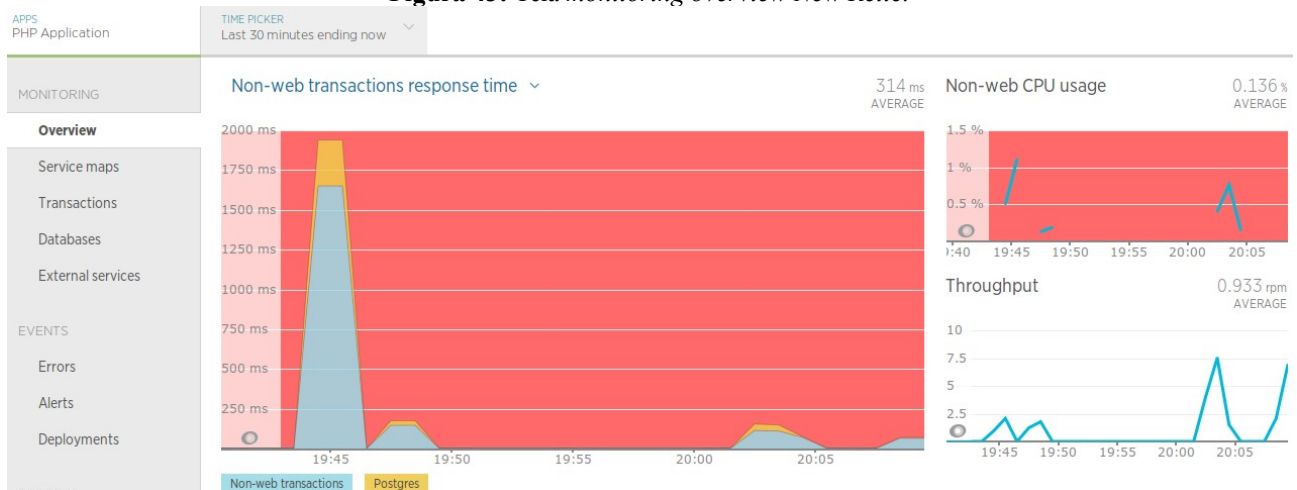
Fonte: Autor.

Por fim, basta reiniciar o apache, e analisar os dados coletados no navegador das aplicações monitoradas. A ferramenta *New Relic* oferece alguns recursos que auxiliam no monitoramento e acompanhamento da aplicação, dentre eles:

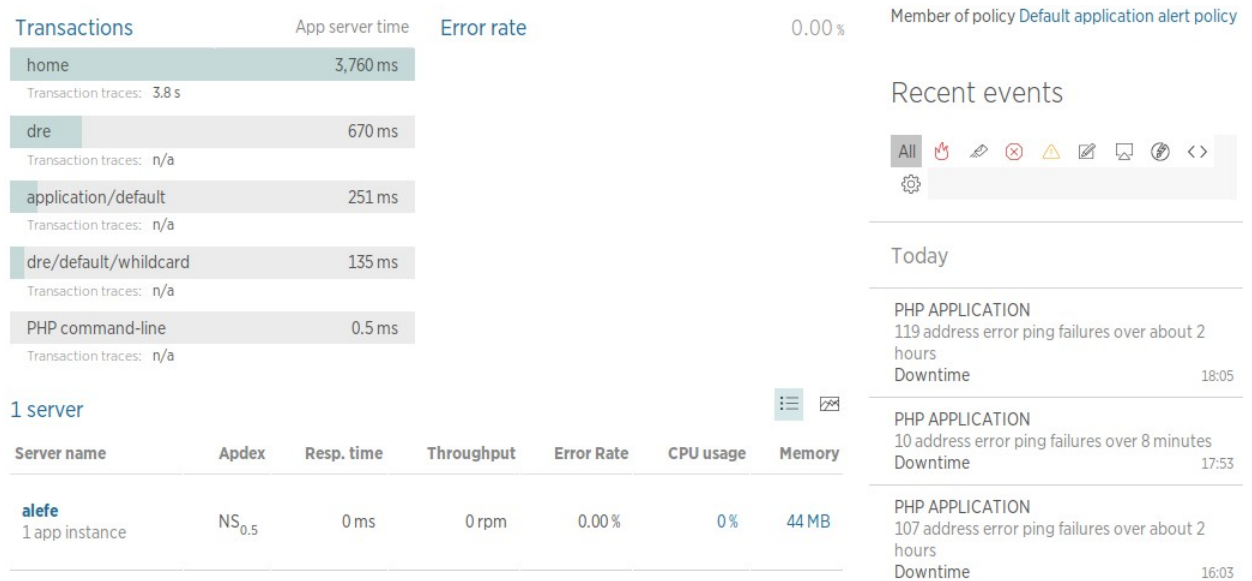
Monitoramento:

Monitoring → **Overview**, na figura 43 é mostrado um visão geral da aplicação, como informações tempo de respostas de transações na *Web*, taxa de transferências e porcentagens de erros.

Figura 43: Tela *monitoring overview* *New Relic*.

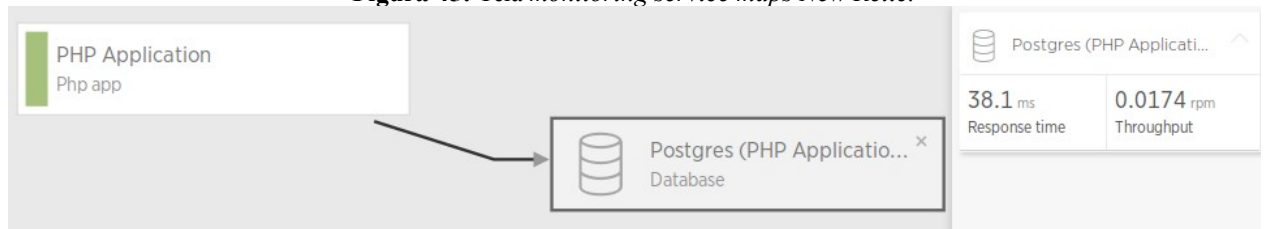


Fonte: Autor.

Figura 44: Tela *monitoring overview* New Relic.

Fonte: Autor.

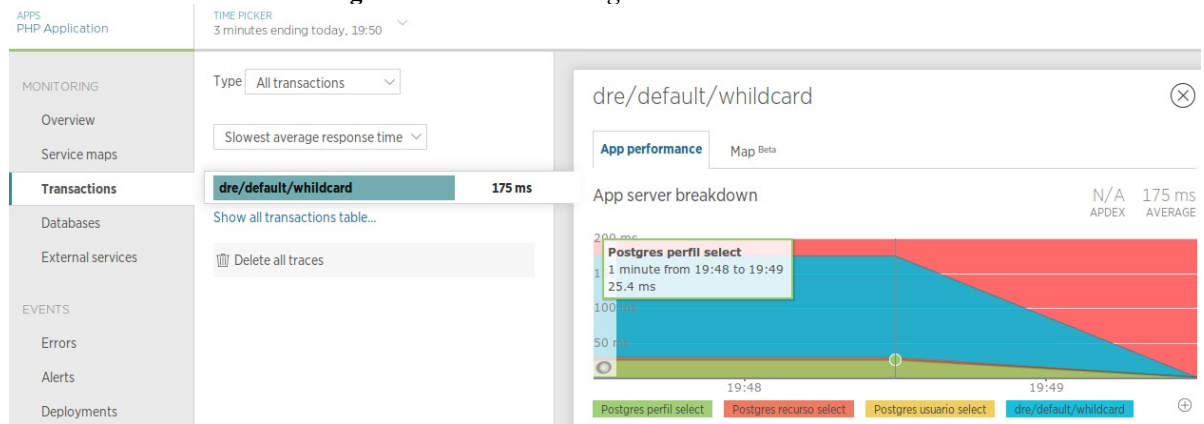
Monitoring → **Service Maps**, onde permite ver quais bases de dados a aplicação mantém conexão.

Figura 45: Tela *monitoring service maps* New Relic.

Fonte: Autor.

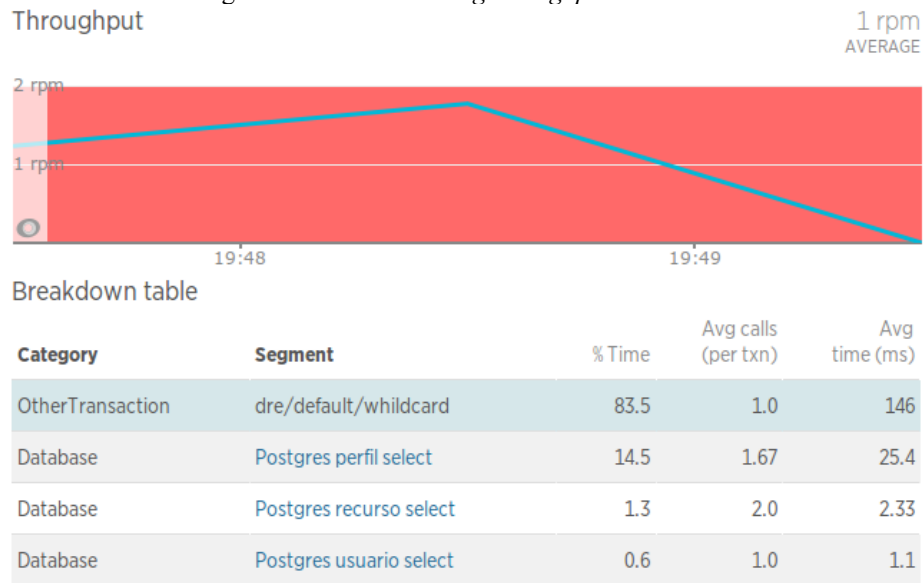
Monitoring → **Transactions**, esta tela mostra as transições com mais detalhes, apresentando a sua performance e as categorias de cada transação, podem ser transições *web* e ou transições *database*.

Figura 46: Tela monitoring transactions New Relic.



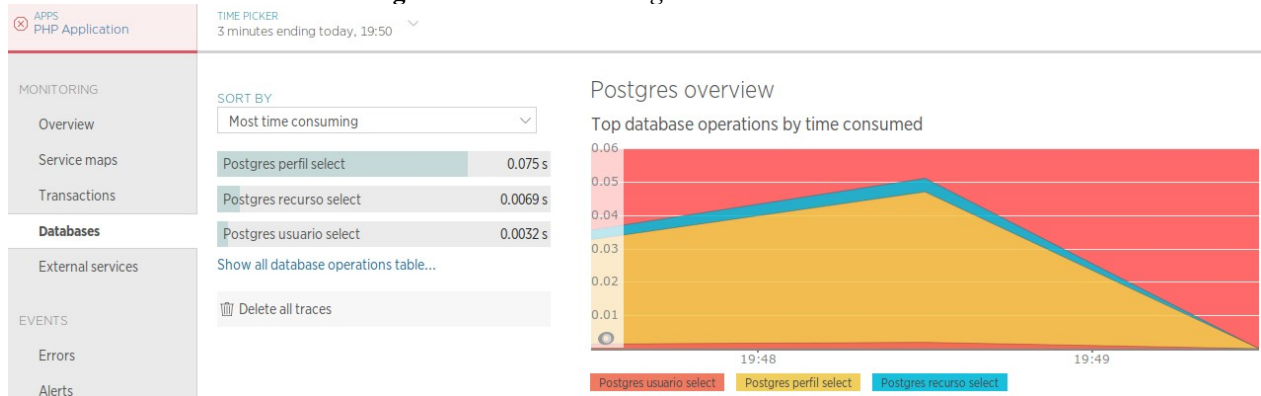
Fonte: Autor.

Figura 47: Tela monitoring throughput New Relic.

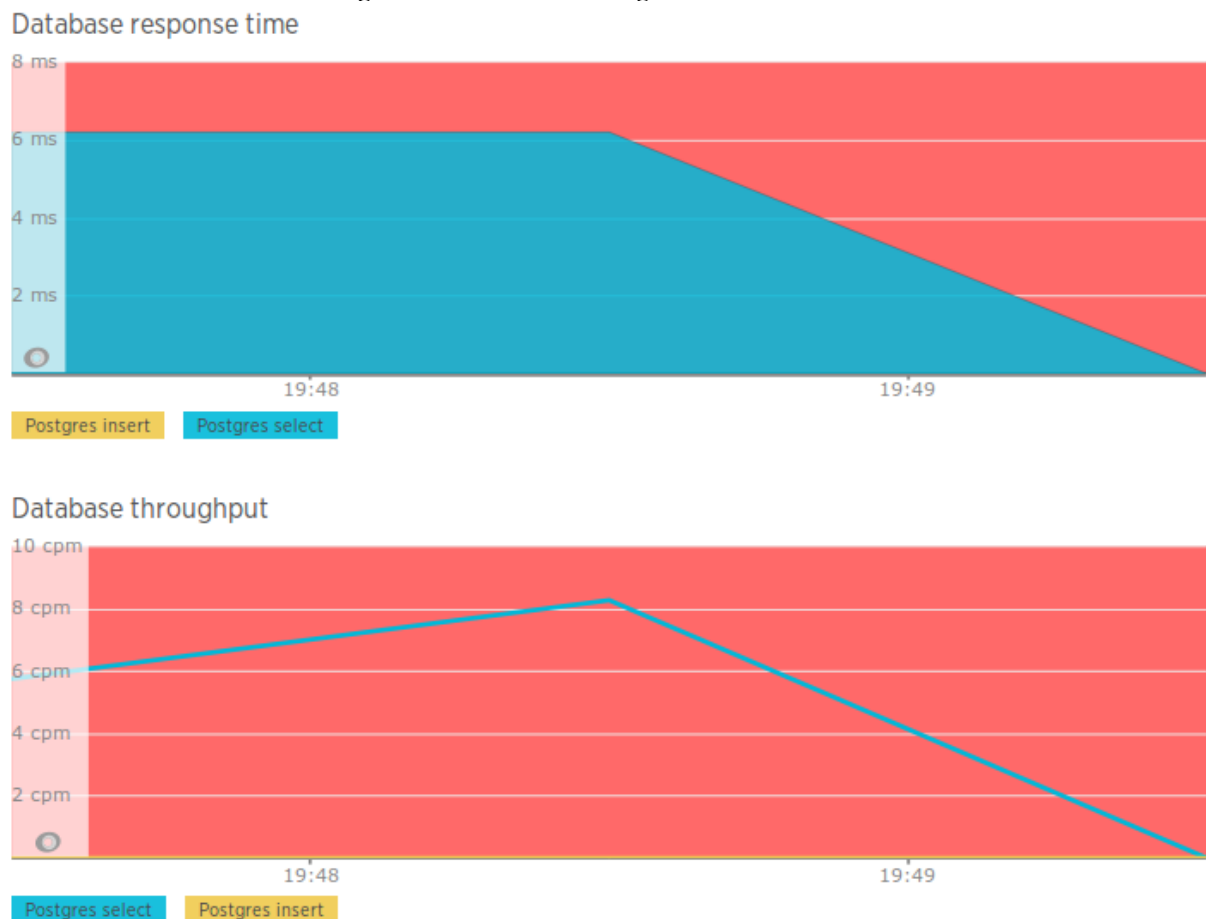


Fonte: Autor.

Monitoring → **Databases**, na tela de *Databases* são apresentados as consultas que foram feitas e o tempo que foi necessário para executar.

Figura 48: Tela *monitoring databases New Relic*.

Fonte: Autor.

Figura 49: Tela 2 *monitoring databases New Relic*.

Fonte: Autor.

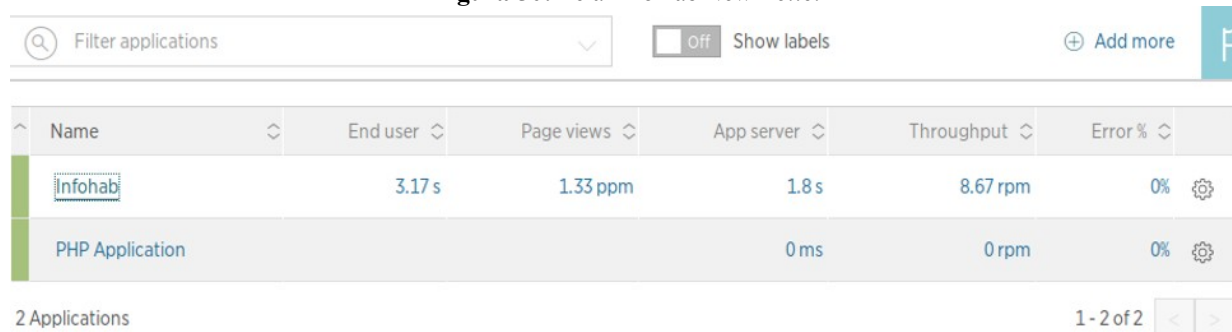
O *New Relic* oferece ainda telas de erros da aplicação, configuração de Alertas para casos com mais relevâncias, além de disponibilizar uma bom quantidade de relatórios, que visão levantar informações como: escalabilidade, transações das bases de dados e disponibilidade.

A utilização do *New Relic* foi testado para um sistema desenvolvidos e mantido atualmente pelo CRS e a DTI da Unochapecó, este sistema é o InfoHab (centro de referência e informação em habitação) e foi desenvolvido a partir do PHP (linguagem de programação), *Oracle* (banco de dados), SVN (controle de versão) e o *framework* PHP *Zend Framework 1*²²

InfoHab é um centro de referência e informação em habitação, a proposta começou como uma biblioteca digital, voltado para a divulgação da produção científica, mas hoje oferece produtos e serviços que servem de apoio a todas as etapas processo/ciclo de geração do conhecimento, desde a discussão entre pesquisadores, divulgação de chamadas de trabalhos, realização de eventos, gestão do processo de avaliação de eventos e periódicos até a divulgação dos resultados de pesquisa na biblioteca que servirão de insumo para novas pesquisas (INFOHAB, 2015).

O sistema InfoHab tem como características realizar muitas consultas no banco de dados e consultas que acabam demorando devido a quantidade de dados ou pelos filtros feitos, com *New Relic* pode-se identificar os gargalos da aplicação, com a configuração do *New Relic*, como mostra na figura 50:

Figura 50: Tela InfoHab *New Relic*.



Name	End user	Page views	App server	Throughput	Error %
Infohab	3.17 s	1.33 ppm	1.8 s	8.67 rpm	0%
PHP Application	0 ms		0 ms	0 rpm	0%

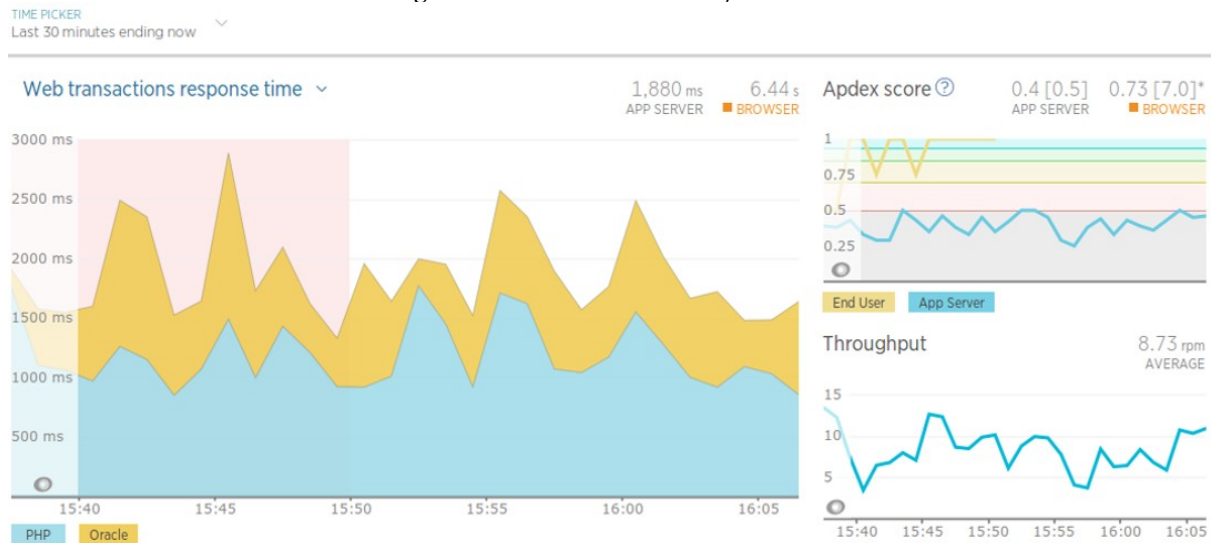
2 Applications 1 - 2 of 2

Fonte: Autor.

Acessando a aplicação InfoHab, pode-se ver as transações *web* dos últimos 30 minutos, as consultas mais demoras, erros e taxa de transferência, como é apresentados nas figuras 51 e 52.

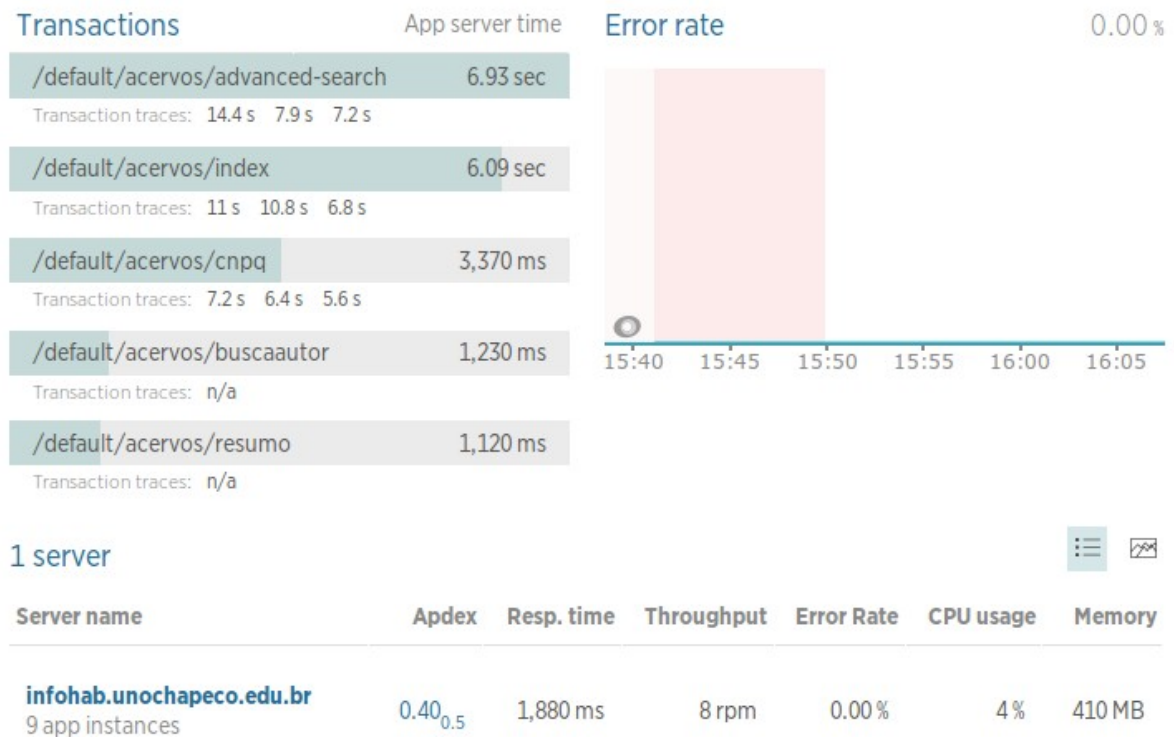
²² Conjunto de classes com objetivo de reutilização para criação de *software*

Figura 51: Tela Infohab transações Web.



Fonte: Autor.

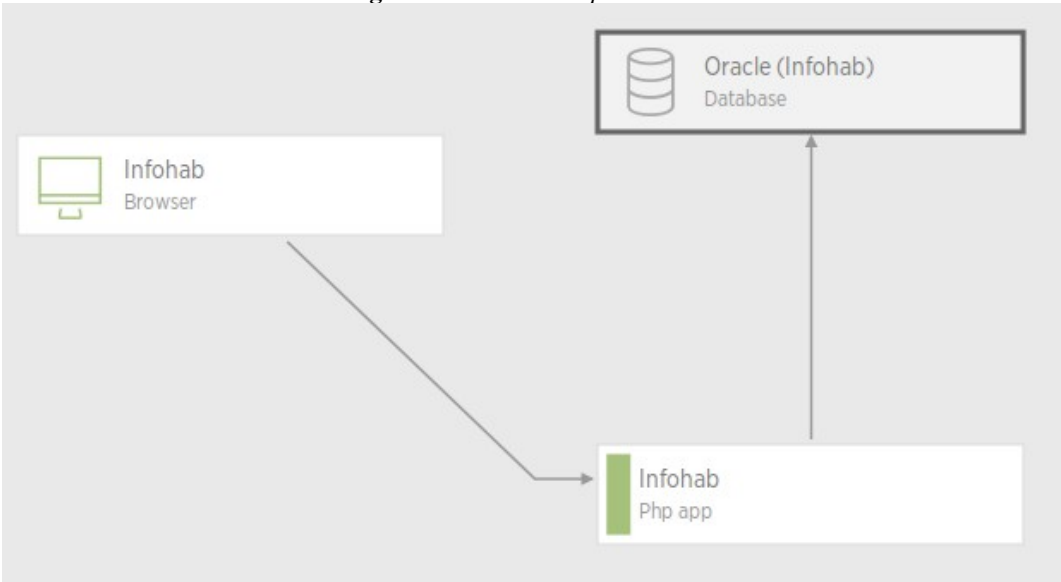
Figura 52: Tela consultas mais demoradas InfoHab.



Fonte: Autor.

Com a tela de *Service Maps* é visualizado as bases que possui conexão, neste caso com o *Oracle* e o *Browser* por onde é acessada.

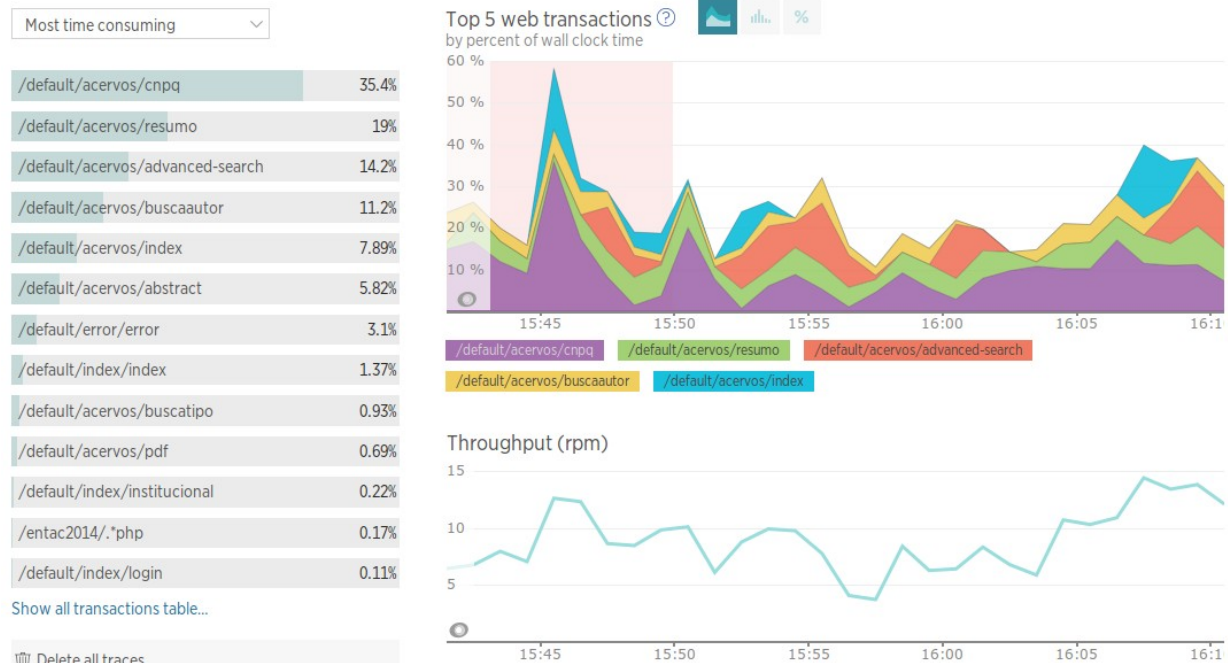
Figura 53: *Service Maps* InfoHab.



Fonte: Autor.

Na tela de *Transactions* pode-se ver qual consulta teve mais consumo de processamento e qual demorou mais, informação importante que podem ser passadas para desenvolvedores buscarem melhores soluções.

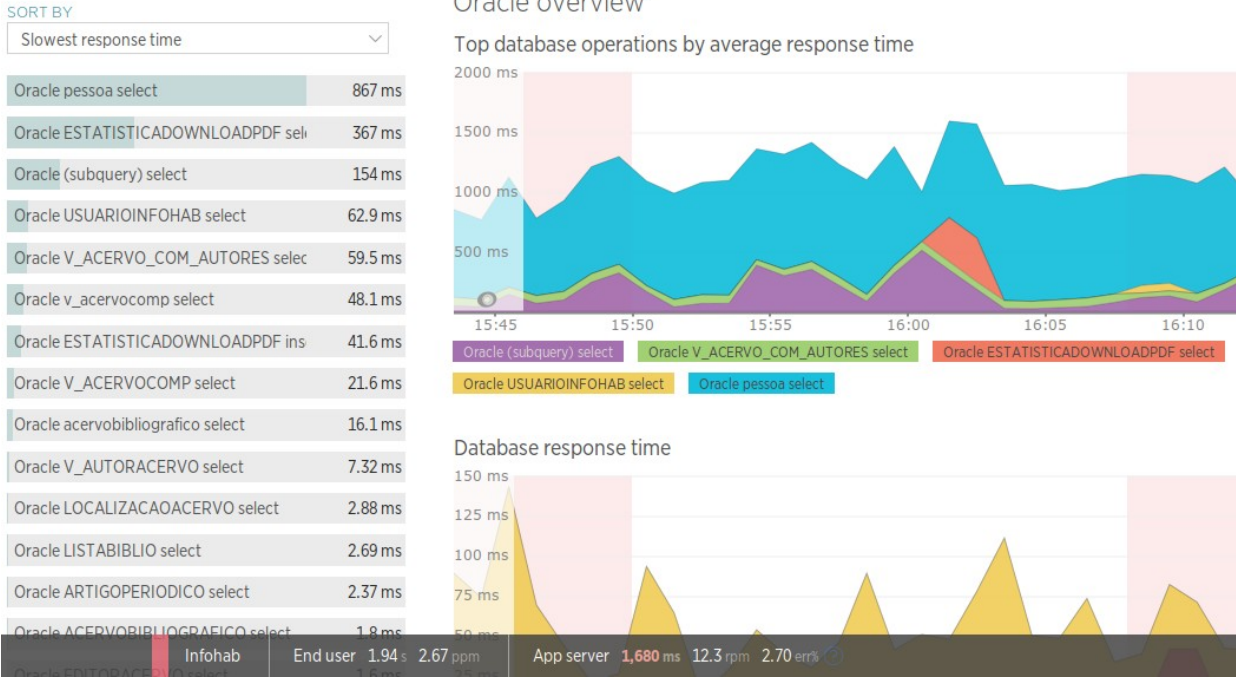
Figura 54: Tela *transactions* InfoHab.



Fonte: Autor.

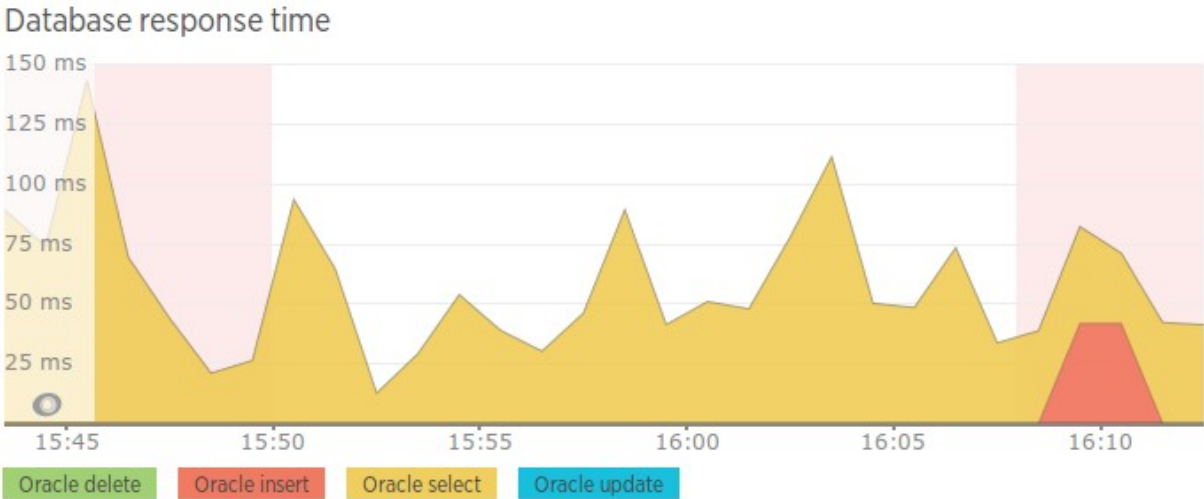
Na tela de *Databases* são mostrados com mais detalhes as consultas, qual operação é mais feita (*insert*, *select*, *delete* ou *update*) e também mostrando as *queries* das consultas com a quantidade de registros de cada uma, como é mostrado nas figuras 55, 56 e 57.

Figura 55: Tela *Databases* InfoHab.



Fonte: Autor.

Figura 56: Tela *Databases* operações mais feitas InfoHab.



Fonte: Autor.

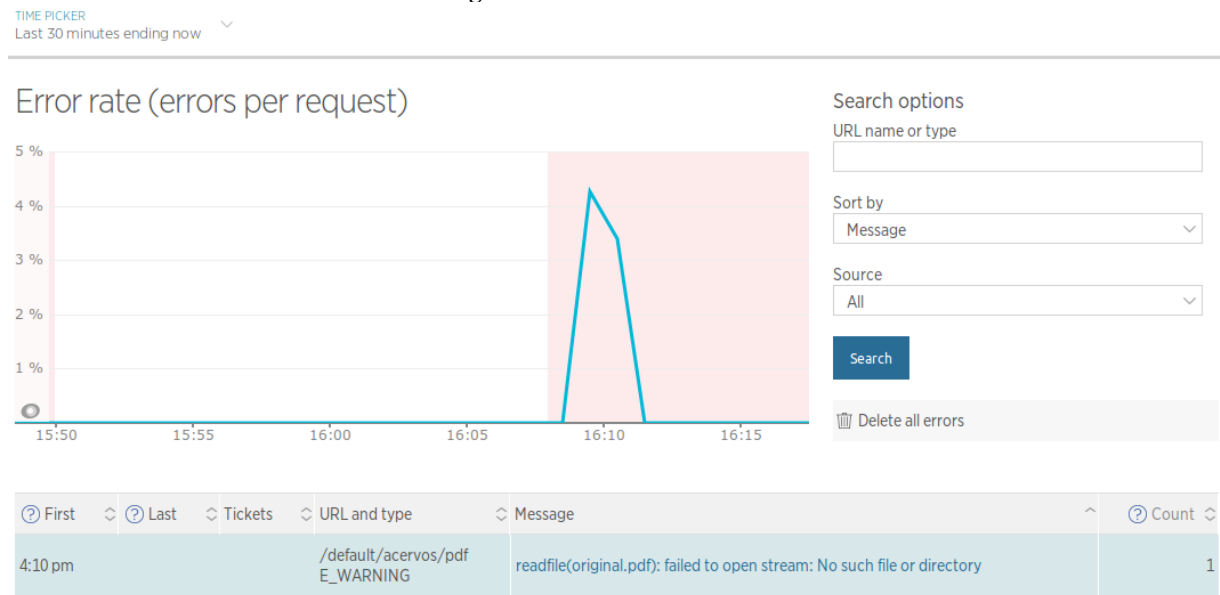
Figura 57: Tela *Databases* sqls das consultas InfoHab.

Slow queries	Response time	Sample count
select NOME, COD_LATTES AS CODIGO_LATTES from INFOHAB?.pessoa where REGEXP_LIKE(nome,?, ?) and REGEXP_LIKE(nome,?, ?) union all select lattes.NOME, lattes.CODIGO_LATTES from INFOHAB_LEGADO?.AUTOR aut JOIN INFOHAB_LEGADO?.REVERSOLATTES lattes on lattes.codigo = aut.cd_autor where REGEXP_LIK...	3.03 s	43
SELECT COUNT(?) AS ? FROM (select acervobibliografico.CODIGO_BIBLIO,acervobibliografico.TITULO_BIBLIOGRAFIA,acervobibliografico.CD_TIPO,acervobibliografico.DATA_PUBLIC_BIBLIOGRAFIA, acervobibliografico.desc_fisica, acervobibliografico.NOTA_ESPECIAL, acervobibliografico.ativo, cidade.cd_local, cidade.nome_local, congresso.nome_co...	1.34 s	3
	1.25 s	3
select NOME, COD_LATTES AS CODIGO_LATTES from INFOHAB?.pessoa where REGEXP_LIKE(nome,?, ?) and REGEXP_LIKE(nome,?, ?) and REGEXP_LIKE(nome,?, ?) union all select lattes.NOME, lattes.CODIGO_LATTES from INFOHAB_LEGADO?.AUTOR aut JOIN INFOHAB_LEGADO?.REVERSOLATTES lattes on lattes.codigo = aut.cd_autor ...	968 ms	29
select NOME, COD_LATTES AS CODIGO_LATTES from INFOHAB?.pessoa where REGEXP_LIKE(nome,?, ?) and REGEXP_LIKE(nome,?, ?) and REGEXP_LIKE(nome,?, ?) and REGEXP_LIKE(nome,?, ?) and REGEXP_LIKE(nome,?, ?) union all select lattes.NOME, lattes.CODIGO_LATTES from INFOHAB_LEGADO?.AUTOR aut JOIN INFOHAB_LEGADO?.REVERSOL...	865 ms	9

Fonte: Autor.

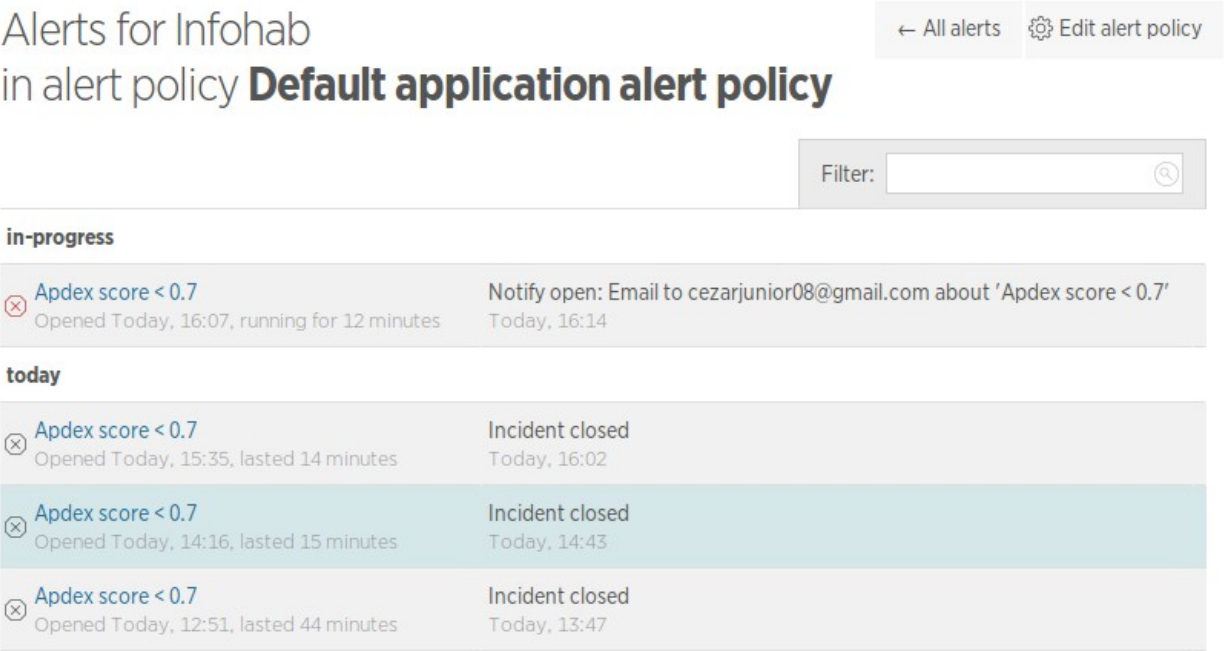
New Relic também oferece um tela de erros encontrados que foram lançados pelo sistema e uma tela de notificações, onde erros críticos da aplicação podem ser informados por e-mail automáticos.

Figura 58: Tela *Error* InfoHab.



Fonte: Autor.

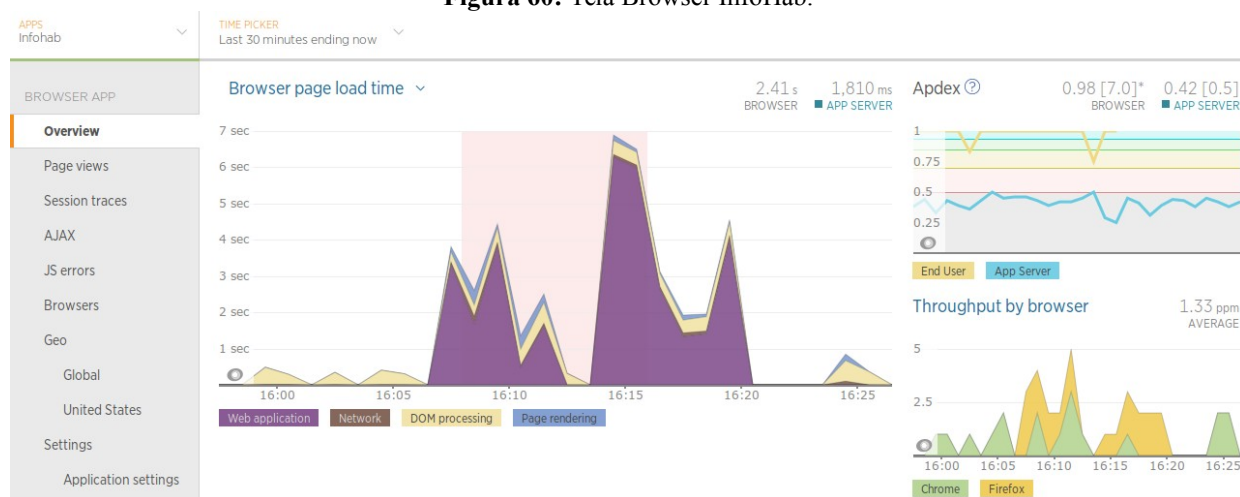
Figura 59: Tela *Alerts* InfoHab.



Fonte: Autor.

Outro recurso útil do *New Relic* é o *Browser*, onde pode-se ver o tempo de carregamento das páginas, erros de JS (*javascript*), *browser* com mais utilização e dentre outros. Como mostra a figura 60.

Figura 60: Tela Browser InfoHab.



Fonte: Autor.

Percebe-se que o *New Relic* oferece vários recursos úteis para os desenvolvedores, que dificilmente seriam percebidos manualmente, com um monitoramento da aplicação pode-se melhorar pontos críticos, antes que possam ser percebidos pelo usuário, isso acaba sendo muito importante hoje pois, melhora a satisfação dos usuários, diminuem naturalmente as reclamações, tornando assim a aplicação mais estável.

No próximo tópico será abordado outro ponto importante, que é a padronização do ambientes utilizados, evitando diferenças de versões e dores de cabeça para as liberações de *software*, no CRS e DTI existe essas diferenças nos ambientes, como foi citados no tópico 6.3 onde foram apresentados os principais problemas enfrentados entre o CRS e DTI da Unochapecó, para isso foi utilizado a ferramenta *Docker*.

7.3 Aplicação da ferramenta *Docker* para padronização dos ambientes

O CRS e DTI da Unochapecó possuem dois ambientes: o ambiente de desenvolvimento que está localizado no CRS e o ambiente de produção que está localizado no DTI. Para identificar especificamente as diferenças do ambientes foi solicitado um relatório a equipe de operações das versões de linguagem de programação, banco de dados, sistema operacionais, configurações de aplicação e bibliotecas ou pacotes utilizados, com este relatório e os detalhes apresentados pode-se visualizar claramente as diferenças como mostra a tabela a seguir.

Tabela 5: Diferenças do ambientes CRS e DTI da Unochapecó.

Recursos/Ambientes	Ambiente de Produção (DTI)	Ambiente de Desenvolvimento (CRS)
Sistema Operacional	CentOS <i>release</i> 5.8 (Final) 64bits	CentOS <i>release</i> 5.5 (Final) 32bits
PHP	5.4.17	5.4.48
Httpd Apache	2.2.3-83	2.2.3-91
PostgreSQL	8.1.23	8.4.4
<i>Mysql</i>	5.5.25	5.5.38
<i>Oracle client</i>	11.2	11.2
OCI8	1.4.9	1.4.8

Fonte: Autor.

Com a ferramenta *Docker* foi criado uma máquina para os dois ambientes, baseado no ambiente de produção do DTI com as seguintes configurações.

Tabela 6: Versões dos recursos da máquina criada com o Docker.

Recurso/Ambiente	Ambiente para CRS e DTI criado com <i>Docker</i>
Sistema Operacional	CentOS <i>release</i> 5.8 (Final) 64bits
PHP	5.4.45
Httpd Apache	2.2.3-83
PostgreSQL	8.1.23
<i>Mysql</i>	5.0
<i>Oracle client</i>	11.2
OCI8	1.4.9

Fonte: Autor.

O objetivo desta máquina é que ela seja utilizada no ambiente de desenvolvimento e no ambiente de produção para que não exista diferença entre os ambientes. A seguir será

apresentado como é feita a instalação e configurações do *Docker*, utilização e construção de uma *container*.

Docker é uma ferramenta de *open source* (código aberto) sem nenhum custo, que permite um conjunto completo do ambiente, desde configurações de máquinas, aplicações e programas necessários, podendo ser distribuído e virtualizado de maneira fácil e rápida para outras máquinas. Para utilização da ferramenta foi criado as seguintes etapas, este tutorial foi aplicado em um ambiente *Linux* (Ubuntu) para o padronização dos ambientes do CRS e DTI Unochapecó:

1 – Atualização pacotes;

Quadro 8: Comando para atualizar pacotes.

```
sudo apt-get update
```

Fonte: Autor.

2 – Instalação do *Docker*;

Quadro 9: Comando para instalação *Docker*.

```
sudo apt-get install docker.io
```

Fonte: Autor.

3 – Configuração *Docker* para inicialização;

Quadro 10: Comando configurar *Docker* para inicialização.

```
sudo systemctl enable docker
```

Fonte: Autor.

4 – Reiniciar os serviços *NetworkManager* e *Docker*;

Quadro 11: Comando para reiniciar serviços.

```
sudo restart network-manager  
sudo restart docker
```

Fonte: Autor.

5 – Verificar se o *Docker* está rodando;

Quadro 12: Comando para verificar se *Docker* está rodando.

```
sudo docker run hello-world
```

Fonte: Autor.

6 – Depois de tudo instalado, o que falta é o *container* onde contém a máquina com as configurações e programas necessários para o ambiente de desenvolvimento. O *Docker* mantém um local pra armazenar repositórios²³. E para este caso foi criado um com o nome de *crs-uno-server*²⁴.

7 – Para baixar o repositório *crs-uno-server*;

Quadro 13: Comando para baixar repositório *crs-uno-server*.

```
docker pull alefevariani/crs-uno-server
```

Fonte: Autor.

8 – Listando os *container*;

Quadro 14: Comando lista *container Docker*.

```
docker ps
```

Fonte: Autor.

9 – Listar *images Docker* existentes na máquina;

Quadro 15: Comando lista *images Docker*.

```
docker images
```

Fonte: Autor.

10 – Rodando e acessando o *shell* do *container*, informando o ID *image*;

```
docker run -it ID IMAGE bash
```

Quadro 16: Comando para rodar e acessar *shell container* *crs-uno-server*.

```
docker run -it CONTAINER ID IMAGE bash
```

Fonte: Autor.

²³ <https://hub.docker.com>

²⁴ <https://hub.docker.com/r/alefevariani/crs-uno-server/>

Criando um repositório no *Docker*.

11 – Para a criação de um repositório é necessário criar uma conta no *site*²⁵

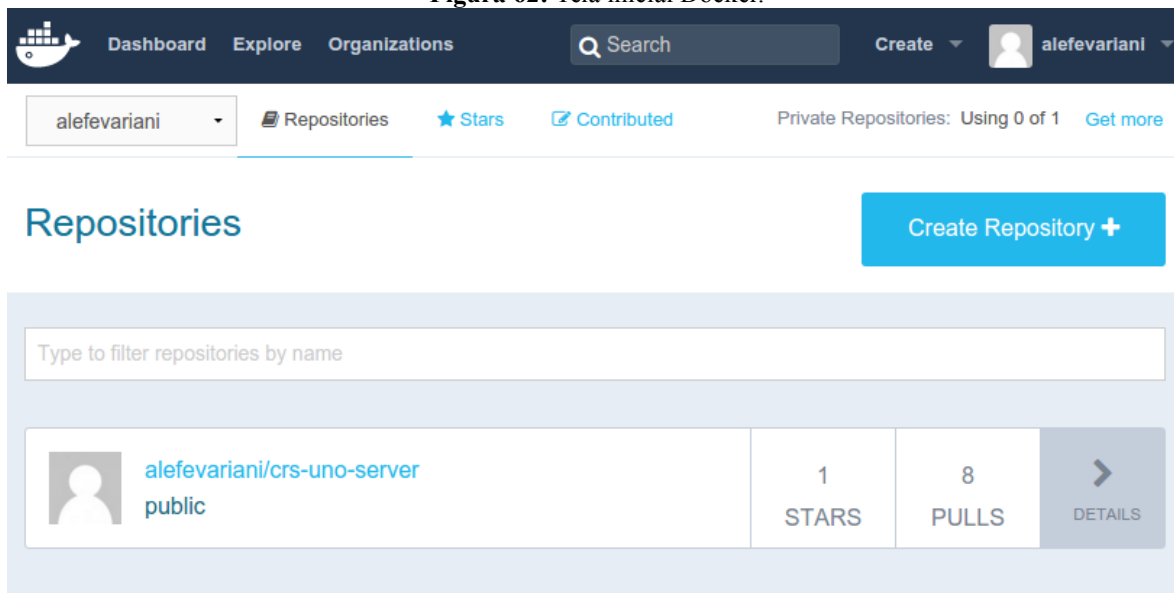
Figura 61: Tela login cadastro Docker.



Fonte: Autor.

12 – Após o login efetuado, será apresentado uma tela semelhante a figura 62, onde terá a opção *Create Repository* (criar repositório);

Figura 62: Tela inicial Docker.



Fonte: Autor.

²⁵ <https://hub.docker.com/>

13 – Selecionada a opção *Create Repository* (criar repositório), a tela será a figura 63, que possui o nome, descrições e a visibilidade que pode ser privada (*private*) ou pública (*public*);

Figura 63: Tela de criação repositório Docker.

The form for creating a Docker repository includes the following elements:

- A dropdown menu showing the username **alefevariani**.
- A text input field for the **Repository Name**.
- A text area for the **Short Description (100 Characters)**.
- A text area for the **Full Description**.
- A **Visibility** dropdown menu currently set to **public**.
- A blue **Create** button.

Fonte: Autor.

14 – Criado o repositório, neste caso a tela será semelhante a figura 64;

Figura 64: Tela repositório teste Docker.

The repository page displays the following information:

- Repository Name:** alefevariani/teste ☆
- Status:** Last pushed: never
- Tabs:** Repo Info, Tags, Collaborators, Webhooks, Settings
- Short Description:** A text box with the placeholder "Short description is empty for this repo." and an edit icon.
- Full Description:** A text box with the placeholder "Full description is empty for this repo." and an edit icon.
- Docker Pull Command:** docker pull alefevariani/teste
- Owner:** alefevariani (with a profile icon)

Fonte: Autor.

Nesta figura já é apresentado o comando “docker pull alefevariani/teste” para baixar o *container image* criado, mas como está vazio é preciso criar a máquina com os itens a seguir:

15 – Via terminal *linux*, criar uma pasta com o arquivo *Dockerfile*;

Quadro 17: Comando criar pasta e arquivo Dockerfile.

```
mkdir teste
cd teste/
nano Dockerfile
```

Fonte: Autor.

Adicionar conteúdo no *Dockerfile* onde neste arquivo ficam as configurações da máquina, mais informações para criação do arquivo *Dockerfile* no link²⁶ e também para os repositórios oficiais oferecidos, segue link²⁷;

Quadro 18: Exemplo arquivo Dockerfile.

```
# INSTALAR SO UBUNTU, ÚLTIMA VERSÃO
FROM ubuntu:latest

# INSTALAR APACHE 2 E PHP
RUN apt-get -y install apache2 php5

#LIBERAÇÃO DE PORTAS 80 E 443
EXPOSE 80 443
```

Fonte: Autor.

16 – Para executar o arquivo *Dockerfile* e verificar se os comando estão todos corretos:

Quadro 19: Executar arquivo Dockerfile.

```
docker build -t alefevariani/teste .
```

Fonte: Autor.

No local de alefevariani/teste, deve-se passar nomeSeuUsuario/nomeRepositorio.

17 – Para montar o *container* para o repositório no *Hub Docker*, são os seguintes passos, semelhante a um controle de versão;

²⁶ <https://docs.docker.com/reference/builder/>

²⁷ <https://hub.docker.com/explore/>

Listar último *container*.

Quadro 20: Comando listar último container.

```
docker ps -l
```

Fonte: Autor.

Fazer *commit* (*confirmação*) da imagem criada, informando o CONTAINER ID.

Quadro 21: Comando fazer commit container Docker.

```
docker commit CONTAINER ID alefevariani/teste
```

Fonte: Autor.

Enviar a imagem da máquina para o repositório, permitindo depois fazer *pull* por outras máquinas.

Quadro 22: Comando enviar imagem Docker.

```
docker push alefevariani/teste
```

Fonte: Autor.

Feito isso, pode-se utilizar este *container* para várias máquinas, no caso, para o servidor de desenvolvimento e produção, evitando diferenças de versões em sistemas operacional, linguagem de programação, banco de dados e pacotes extras.

7.4 Considerações Finais

Neste tópico foi abordado sobre a aplicação do projeto no CRS e DTI da Unochapecó, onde foram aplicados os seguintes itens: aplicação dos conceitos do DevOps no CRS e DTI no tópico 7.1, utilização do *New Relic* para o monitoramento das aplicações no tópico 7.2 e por fim a utilização da ferramenta *Docker* para a padronização dos ambientes no tópico 7.3.

A partir dos conceitos e ferramentas aplicadas no CRS e DTI pode-se perceber melhorias, no próximo capítulo sobre os levantamentos dos resultados conquistados, será apresentados estas melhorias e se os objetivos inicial impostos do projeto foram alcançados.

8. LEVANTAMENTO DOS RESULTADOS CONQUISTADOS

Através dos itens aplicados no CRS e DTI da Unochapecó no último capítulo e comparando com os objetivos apresentados, onde eram:

1. Conhecer os processos do desenvolvimento de *software*;
2. Entender conceitos da metodologia que se aplicam ao ambiente DevOps;
3. Melhorar o ambiente de desenvolvimento e monitoramento da aplicação através da cultura do DevOps;
4. Aplicar uma proposta em um ambiente desenvolvimento real (CRS e DTI da Unochapecó), utilizando conceitos e práticas do DevOps;
5. Analisar o comportamento, vantagens e desvantagens que a proposta traz ao ambiente de desenvolvimento;

Os objetivos foram divididos da seguinte forma, os objetivos 1, 2 e 3 são responsáveis pela parte da pesquisa científica e entendimentos dos temas abordados e nos objetivos 4 e 5 são responsáveis pela aplicação dos temas do projeto.

Nos objetivos 1 e 2, pode-se conhecer os processos do desenvolvimento de *software* e entender os conceitos da metodologia que se aplica aos ambientes DevOps pelos capítulos 2 (apresentação dos diferentes ambientes de desenvolvimento de *software*) e o capítulo 4 (problemas entre equipes de desenvolvimento e equipes de operações), onde foram abordados sobre os processos, ambientes e as equipes existentes no desenvolvimento de *software*.

No objetivo número 3, são apresentados nos capítulos 3 (monitoramento de aplicações) e 5 (DevOps), mostrando a importância destes fatores e como melhorar o ambiente de desenvolvimento e o monitoramento da aplicação através dos conceitos e práticas do DevOps.

Nos objetivos 4 e 5 são abordados nos capítulos 6 Levantamento e análise de dados sobre DevOps no CRS/DTI Unochapecó e a região do oeste de Santa Catarina, 7 Aplicação do DevOps no CRS/DTI da Unochapecó e no capítulo 8 Levantamento dos dados conquistados, a proposta de aplicação foi definida através dos resultados alcançados no questionário aplicado entre o CRS e o DTI da Unochapecó, onde os itens da proposta foram: apresentação dos conceitos do DevOps, monitoramento de aplicações e padronização dos ambientes.

A análise dos comportamentos são avaliados entre as vantagens e desvantagens que foram levados ao CRS e DTI da Unochapecó:

Desvantagens e dificuldades:

- Tempo gasto para aprender e utilizar novas ferramentas, ferramentas são muito úteis e auxiliam em diversas atividades, mas passar a utilizar uma nova ferramenta é necessário um certo tempo para adaptação e muitas vezes por existir prazos apertados se tornam inviável;
- Tempo para assimilar os novos conceitos, DevOps é muito mais mudança de cultura do que apenas utilização de ferramentas, mudanças na cultura passa por acabar com vícios e responsabilidades centralizadas criadas;
- Processos burocráticos da Instituição, instalações e compras de *software* na Instituição levam tempo e onde viu-se falta de liberdade para criação de máquinas por exemplo;

Vantagens da aplicação:

- Conhecimento de novos conceitos e ferramentas do DevOps, conhecer novos conceitos e ferramentas é muito importante no mercado atual e se torna um diferencial principalmente para pessoas que trabalham com TI;
- Melhor acompanhamento das aplicações através da ferramenta de monitoramento *New Relic*, com o monitoramento da aplicação pode-se conseguir dados em tempo real que dificilmente seriam capturados manualmente, *New Relic* oferece diversos recursos, o que ajuda na estabilidade e satisfação dos clientes;
- Padronização dos ambientes de desenvolvimento e produção através da ferramenta *Docker*, esta ferramenta é muito usada atualmente, a facilidade e rapidez para a utilização são uns dos pontos positivos, permitindo criar um máquina com *software* e configurações específicas de cada projeto, podendo manter também a aplicação;
- Diminuição das diferenças entre os ambientes de desenvolvimento e produção, tornar ambientes de desenvolvimento e produção iguais traz resultados como liberação de versões (*deploy*) sem complicações;
- Diminuição da distância entre equipe de desenvolvedores (CRS) e equipe de operações (DTI da Unochapecó), isso permitiu que desenvolvedores entendessem um pouco mais sobre operações e que operações entendessem um pouco mais de desenvolvedores;

8.1 Considerações Finais

Neste tópico pode-se ver a apresentação dos objetivos impostos pelo projeto e o levantamento dos pontos positivos alcançados para o CRS e DTI da Unochapecó, onde foram citados as vantagens e as desvantagens/dificuldades da aplicação do trabalho, no capítulo a seguir são abordados as considerações finais no trabalho e as sugestões para trabalhos futuros, já que é um tema pouco difundido na região.

9. CONCLUSÕES

9.1 Considerações Finais

DevOps é um tema atual e é utilizados por várias empresas de grande porte como: *Microsoft*, *HBO Latin America*, *Flickr*, *Praxis Flow* dentre outras, segundo Rackspace (2015), ainda, uma pesquisa aponta que 73% das empresas Brasileiras possuem a presença da cultura DevOps, a pesquisa global conduzida pela Vanson Bourne, mostra ainda ganhos na frequência de lançamentos de aplicativos e aumento da receita em companhias que adotam metodologia, segundo Cavalcanti (2015).

Com este trabalho pode-se alcançar os objetivos propostos, mesmo devido as dificuldades encontradas. Percebe-se que existe ainda um bom caminho para a quebra de barreiras e utilização do DevOps seja feita. O DevOps para meios corporativos, como o CRS e DTI da Unochapecó, mostram que é necessário muita motivação diante dos processos burocráticos existentes enfrentados.

Neste contexto, a proposta de aplicar os conceitos e ferramentas em um ambiente real de desenvolvimento foi conquistada, onde 3 itens importantíssimos foram aplicados, que são: apresentação dos conceitos da cultura DevOps, o monitoramento da aplicação e a padronização dos ambientes de desenvolvimento de *software*.

Com isso, o trabalho pode melhorar a relação entre o CRS e a DTI da Unochapecó e também ajudar a divulgar o DevOps pela região do Oeste de Santa Catarina através do questionário de pesquisa disponibilizado.

9.2 Trabalhos Futuros

Para trabalhos futuros sugere-se aplicar novos itens no CRS e DTI da Unochapecó ou empresas que desconheçam o DevOps:

- Automatizar e facilitar processo de *Deploy* da aplicação;
- Gerenciamento de *logs*;
- Ferramentas para centralização das configurações (*Chef* e *Puppet*);
- Análise entre a aplicação do DevOps para meios corporativos e *startups*;

REFERÊNCIAS BIBLIOGRÁFICAS

4LINUX. **O que é Monitoramento?** Disponível em: <<http://www.4linux.com.br/o-que-e-monitoramento>>. Acesso em: 15 abr. 2015.

AÉCE, Juliano. **Usando Arquivos de Configuração.** 2011. Disponível em: <<https://msdn.microsoft.com/pt-br/library/gg537280.aspx>>. Acesso em: 22 abr. 2015.

BUNDLER. **Bundler's Purpose and Rationale.** Disponível em: <<http://bundler.io/rationale.html>>. Acesso em: 11 maio 2015.

BUNDLER. **What is Bundler?** Disponível em: <<http://bundler.io/>>. Acesso em: 11 maio 2015.

CAVALCANTI, Vitor. **Estudo aponta presença de DevOps em 73% das empresas brasileiras.** 2015. Disponível em: <<http://itforum365.com.br/noticias/detalhe/115149/estudo-aponta-presenca-de-devops-em-73-das-empresas-brasileiras>>. Acesso em: 04 nov. 2015.

CARDOSO, Andre. **Composer para iniciantes.** 2014. Disponível em: <<http://tableless.com.br/composer-para-iniciantes/>>. Acesso em: 11 maio 2015.

CARVALHO, Guto. **O que é DevOps afinal?** Disponível em: <<http://gutocarvalho.net/octopress/2013/03/16/o-que-e-um-devops-afinal/>>. Acesso em: 16 mar. 2015.

CHEF. **Infrastructure as Code.** Disponível em: <www.opscode.com/chef>. Acesso em: 10 maio 2015.

COMPOSER. **Command-line interface / Commands.** Disponível em: <<https://getcomposer.org/doc/03-cli.md>>. Acesso em: 11 maio 2015.

COMPOSER. **Introduction.** Disponível em: <<https://getcomposer.org/doc/00-intro.md>>. Acesso em: 11 maio 2015.

CRS. **A Residência.** Disponível em: <crs.unochapeco.edu.br/Zend/crs/Aresidencia/index>. Acesso em: 29 out. 2015.

CUKIER, Daniel. **DevOpsDays – Agilidade em todos os níveis.** 2010. Disponível em: <<http://www.agileandart.com/2010/06/29/devopsdays-agilidade-em-todos-os-niveis/>>. Acesso em: 02 maio 2015.

DATE, J. C. **Introdução a sistemas de bancos de dados.** 9.ed. Elsevier, 2004. 865p.

DEBOIS, Patrick. **5 years of Devops - Devops in the enterprise - Serena Webinar.** 2014. Disponível em: <<http://pt.slideshare.net/jedi4ever/5-years-of-devops-devops-in-the-enterprise-serena-webinar>>. Acesso em: 29 maio 2015.

DEVMEDIA. **DevOps: alinhando operação e desenvolvimento.** Disponível em: <www.devmedia.com.br/devops-alinhando-operacao-e-desenvolvimento/31281>. Acesso em: 27 abr. 2015.

DEVOPSDAYS. **Devopsdays Events.** Disponível em: <<http://www.devopsdays.org/events/>>. Acesso em: 28 maio 2015.

DOCKER. **The best way to understand Docker is to try it!** Disponível em: <<http://www.docker.com/tryit/>>. Acesso em: 29 maio 2015.

DOCKER. **What is Docker?** Disponível em: <<https://www.docker.com/whatisdocker/>>. Acesso em: 09 maio 2015.

DUVALL, Paul. **Agile DevOps: Automação de infraestrutura.** 2012. Disponível em: <<http://www.ibm.com/developerworks/br/library/a-devops2/>>. Acesso em: 10 maio 2015.

DUVALL, Paul. **Agile DevOps: Quebrando os silos.** 2013. Disponível em: <<http://www.ibm.com/developerworks/br/library/a-devops9/#ibm-pcon>>. Acesso em: 27 abr. 2015.

ECLIPSE.ORG. **About the Eclipse Foundation.** Disponível em: <<https://eclipse.org/org/>>. Acesso em: 30 mar. 2015.

ENGHOLM, Hélio Jr. **Engenharia de Software na Prática.** Novatec, 2010. 440p.

FALBO, R. A., **Integração de Conhecimento em um Ambiente de Desenvolvimento de Software, Tese de Doutorado,** COPPE/UFRJ, 1998.

FRANÇA, César C.. **Artigo Engenharia de Software 16 - Motivação em Integrantes de Equipes de Desenvolvimento de Software.** Disponível em: <<http://www.devmedia.com.br/artigo-engenharia-de-software-16-motivacao-em-integrantes-de-equipes-de-desenvolvimento-de-software/14188>>. Acesso em: 27 abr. 2015.

FERNANDES, Ingrid. **Gerenciamento de serviços e Acordo de Nível de Serviço ANS.** 2014. Disponível em: <<http://www.hdibrasil.com.br/publicacoes/artigos-e-noticias/item/gerenciamento-de-servicos-e-acordo-de-nivel-de-servico-ans>>. Acesso em: 30 abr. 2015.

FLUX7. **DevOps Glossary - Monitoring Tools/Services.** 2014. Disponível em: <<http://blog.flux7.com/blogs/glossary/devops-glossary-monitoring-toolsservices>>. Acesso em: 06 maio 2015.

GIT-SCM. **1.1 Primeiros passos - Sobre Controle de Versão.** Disponível em: <<http://git-scm.com/book/pt-br/v1/Primeiros-passos-Sobre-Controle-de-Vers%C3%A3o>>. Acesso em: 31 mar. 2015.

GHEZZI, Carlo. **Conceitos de Linguagens de Programação.** Rio de Janeiro, Campus, 2002. 306p.

HEIDI, Erika. **Vagrant Cookbook**: Guia prático para Vagrant e seus principais Provisioners. 2014. Disponível em: <<https://leanpub.com/vagrantcookbook-ptbr>>. Acesso em: 07 maio 2015.

IBM, DeveloperWorks. **Git para Usuários do Subversion, Parte 1: Introdução**. Disponível em: <<http://www.ibm.com/developerworks/br/linux/library/l-git-subversion-1/>>. Acesso em: 31 mar. 2015.

INFOHAB. **Institucional**: Apresentação sobre o Infohab. Disponível em: <<http://infohab.org.br/index/institucional>>. Acesso em: 31 out. 2015.

JEHIAH. **10 Things We Forgot to Monitor**. 2014. Disponível em: <<http://word.bitly.com/post/74839060954/ten-things-to-monitor>>. Acesso em: 22 abr. 2015.

JOSEPHSEN, David. **NAGIOS**: Building Enterprise-Grade Monitoring Infrastructure for Systems and Networks. 2. ed. New Jersey: Pearson Education, 2013. 304 p.

LEITE, Jair C. **O Processo de Desenvolvimento de Software**. 2000. Disponível em: <<https://www.dimap.ufrn.br/~jair/ES/c2.html>>. Acesso em: 27 abr. 2015.

LOCALWEB. **New Relic**. 2014. Disponível em: <http://wiki.locaweb.com.br/pt-br/Categoria:New_Rellic>. Acesso em: 11 maio 2015.

MARINESCU, Floyd. Padrões de projeto EJB - Padrões avançados, processos e idiomas, prefácio. Bookman Companhia, 2004. 220p.

MAVEN. **Introduction**. Disponível em: <<https://maven.apache.org/what-is-maven.html>>. Acesso em: 11 maio 2015.

MIAN, P. G.; NATALI, A. C.; FALBO, R. A. **Ambientes de Desenvolvimento de Software e o Projeto ADS**. 2001.7 f. Disponível em: <<http://www.inf.ufes.br/~falbo/download/pub/RevistaCT072001.pdf>> Acesso em: 28 mar. 2015.

MICROSOFT. **Uma história do Windows**. Disponível em: <<http://windows.microsoft.com/pt-br/windows/history#T1=era0>>. Acesso em: 28 mar. 2015.

MOSCOVICI, Fela. **Equipes Dão Certo**: A Multiplicação do Talento Humano. 9. ed. Rio de Janeiro: José Olympio Editora, 2004. 240 p.

MORIMOTO, Carlos E. **Servidores Linux - Guia Prático**. 2.ed. Sulina, 2010. 736p.

NAGIOS. **About Nagios**. Disponível em: <<http://www.nagios.org/about>>. Acesso em: 12 maio 2015.

NETBEANS. **A Brief History CUKIER, Daniel. DevOpsDays – Agilidade em todos os níveis**. 2010. Disponível em: <<http://www.agileandart.com/2010/06/29/devopsdays-agilidade-em-todos-os-niveis/>>. Acesso em: 02 maio 2015. of NetBeans. Disponível em: <<https://netbeans.org/about/history.html>>. Acesso em: 30 mar. 2015.

NETMARKETSHARE. **Desktop Operating System Market Share**. Disponível em: <<http://www.netmarketshare.com/operating-system-market-share.aspx?qprid=8&qpcustommd=0>>.

Acesso em: 01 abr. 2015.

ORLANDO, Felipe. **Qual IDE devo utilizar para começar a programar?**. Disponível em: <<http://canaltech.com.br/dica/software/Qual-IDE-devo-utilizar-para-comecar-a-programar/>>.

Acesso em: 30 mar. 2015.

OPENBASE. **O protocolo HTTP**. Disponível em: <<http://www.openbase.com.br/openbdoc/html-pb/opusweb/Ow1p2.htm#T5>>. Acesso em: 28 maio 2015.

RACKSPACE. **A postura do DevOps: A percepção do mundo real dos líderes tecnológicos**. Disponível em: <<http://www.rackspace.com/pt/devops/the-devops-mindset/>>. Acesso em: 16 mar. 2015.

RAMOS, Thiago Santiago. **Medindo o desempenho de aplicações Web – Parte 2**. 2014. Disponível em: <<http://blog.octo.com/pt-br/medindo-o-desempenho-de-aplicacoes-web-parte-2/>>. Acesso em: 29 maio 2015.

RELIC, New. **About New Relic Application Performance Management & Monitoring | New Relic**. Disponível em: <<http://newrelic.com/about>>. Acesso em: 11 maio 2015.

RELIC, New. **O que é DevOps? Metodologia, benefícios e ferramentas**. Disponível em: <<http://newrelic.com/devops/what-is-devops>>. Acesso em: 01 maio 2015.

RELIC, New. **NewRelicCompanyFactSheet**. Disponível em: <<https://try.newrelic.com/rs/newrelic/images/NewRelicCompanyFactSheet.pdf>>. Acesso em: 27 out. 2015.

REZENDE, Denis Alcides. **Engenharia de Software e Sistemas de Informação**. 2. ed. Rio de Janeiro, Brasport, 2002.

RH, Ponto. **Como trabalhar bem em equipe?** Disponível em: <<http://www.pontorh.com.br/como-trabalhar-bem-equipe/>>. Acesso em: 27 abr. 2015.

SANCHES, Camila. **As diferenças entre Git e SVN**. Revista Java Magazine 116, Devmedia. Disponível em: <<http://www.devmedia.com.br/as-diferencas-entre-git-e-svn-revista-java-magazine-116/28079>>. Acesso em: 31 mar. 2015.

SATO, Danilo. **DevOps na prática: entrega de software confiável e automatizada**. São Paulo: Casa do Código, 2013. 248 p.

SEBESTA, Robert W. **Conceitos de Linguagens de Programação**. 9.ed. Bookman Companhia, 2011. 762p.

SEBESTA, Robert W. **Conceitos de linguagem de programação**. 4.ed. Rio de Janeiro: Alta Books, 2000.

SETE, Márcio. **Afinal de contas, que trem é esse chamado DevOps?** 2015. Disponível em: <<http://especificacoes.com/afinal-de-contas-que-trem-e-esse-chamado-devops/>>. Acesso em: 12 abr. 2015.

TANENBAUM, Andrew S; WOODHULL, Albert S. **Sistemas Operacionais: Projeto e Implementação**. 3. ed. Bookman Companhia, 2006. 990p.

TAURION, Cezar. **Como o DevOps pode solucionar os problemas de desenvolvimento nesse mundo tão tecnológico**. 2013. Disponível em: <<http://imasters.com.br/desenvolvimento/como-o-devops-pode-solucionar-os-problemas-de-desenvolvimento-nesse-mundo-tao-tecnologico/>>. Acesso em: 12 abr. 2015.

TEZER, O.s. **Como Instalar e Utilizar o Docker: Primeiros passos**. 2014. Disponível em: <<https://www.digitalocean.com/community/tutorials/como-instalar-e-utilizar-o-docker-primeiros-passos-pt>>. Acesso em: 10 maio 2015.

TIOBE. **TIOBE Index for March 2015**. Disponível em: <<http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html>>. Acesso em: 30 mar. 2015

TUCKER, Allen; NOONAN, Robert. **Linguagens de Programação**. 2. ed. Porto Alegre: Grupo A Educação, 2009. 611 p.

VAGRANT. **About Vagrant**. Disponível em: <<https://www.vagrantup.com/about.html>>. Acesso em: 08 maio 2015.

VAGRANT, Friends Of. **Por que o Vagrant?** Disponível em: <<http://friendsofvagrant.github.io/v1/docs/getting-started/why.html>>. Acesso em: 08 maio 2015.

VIVAOLINUX. **O que é GNU/Linux**. Disponível em: <<http://www.vivaolinux.com.br/linux/>>. Acesso em: 22 abr. 2015.

WORLD, Network. **Monitoramento de aplicações vira aliado dos departamentos de TI**. 2010. Disponível em: <<http://computerworld.com.br/tecnologia/2010/09/27/monitoramento-de-aplicacoes-vira-aliado-dos-departamentos-de-ti>>. Acesso em: 22 abr. 2015.

ZANINI, Alessandro. **SISTEMAS OPERACIONAIS I**. 2000. Disponível em: <<http://www.ebah.com.br/content/ABAAABIZ0AH/sistemas-operacionais>>. Acesso em: 02 abr. 2015.