

DevOps para Infraestrutura de Redes

Marcel R. Faria

Outubro/2017



RNP

MINISTÉRIO DA DEFESA

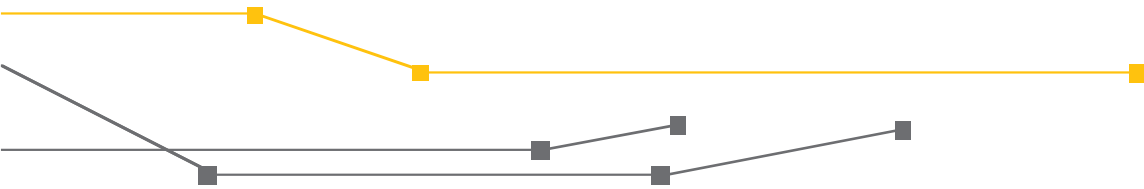
MINISTÉRIO DA CULTURA

MINISTÉRIO DA SAÚDE

MINISTÉRIO DA EDUCAÇÃO

MINISTÉRIO DA CIÊNCIA, TECNOLOGIA, INOVAÇÕES E COMUNICAÇÕES





Sumário

1. Introdução	3
2. O que é DevOps.....	3
3. Infrastructure as Code.....	4
4. Implementando DevOps	5
4.1. Identificar tarefas a serem automatizadas	5
4.2. Escolher ferramentas a serem adotadas.....	5
4.3. Inserir segurança nos pipelines.....	6
4.4. Adotar práticas que facilitem a manutenção.....	6
5. Estudo de Caso: Dyn.....	6
5.1. Motivações	6
5.2. Ferramentas adotadas	7
5.3. Workflow da solução.....	7
5.4. Lições aprendidas e recomendações	9
6. Ferramentas recomendadas	9
7. Considerações Finais	11
8. Agradecimentos	12
9. Referências.....	12



1. Introdução

Nos últimos anos muito tem-se ouvido sobre a tendência de uso DevOps na área de infraestrutura de redes de computadores. DevOps não é um conceito novo, sendo adotado na área de desenvolvimento de software por muitos anos.

Este documento tenta definir o que é DevOps e sua aplicação no escopo de infraestrutura de redes. Em seguida são listadas as melhores práticas para sua adoção, seguida da descrição de um estudo de caso de aplicação do DevOps na empresa Dyn. O documento também lista algumas das ferramentas de software atualmente mais utilizadas em ambientes DevOps, antes das considerações finais.

2. O que é DevOps

O termo DevOps é derivado do inglês, onde temos a junção de “Software **D**evelopment” e “Information Technology **O**perations”, tendo sido introduzido em 2008 na conferência Agile Toronto, por Andrew Shafer e Patrick Debois [1]. Ele define um conjunto de práticas que nasceram na área de desenvolvimento de software. Os objetivos eram a redução do tempo para que novas funcionalidades fossem implementadas em ambiente de produção, diminuição da taxa de falhas das aplicações e aumento da qualidade.

Tende-se a assumir que DevOps seja apenas uma simples automatização dos processos de desenvolvimentos de software, mas na verdade é uma alteração no paradigma de desenvolvimento. Embora pressuponha auto grau de automatização na execução das tarefas do ciclo de vida da aplicação, para que efetivamente funcione é necessário que haja uma integração das equipes de desenvolvimento e operações, de forma a haver uma contínua retroalimentação entre as duas.

O DevOps faz uso de três pipelines (processos automatizados ou semi-automatizados). Primeiro temos o uso de *Continuous Integration* ou CI [2], onde as alterações feitas pelos diferentes elementos do grupo são validadas, testadas e integradas ao repositório compartilhado do código fonte da aplicação. Logo após temos o pipeline *Continuous Delivery*, onde o pipeline CI é estendido até a geração de uma nova release de software. Por último temos o pipeline *Continuous Deployment* onde a nova release é posta em produção.

A figura abaixo resume o conjunto desses pipelines utilizado no DevOps:

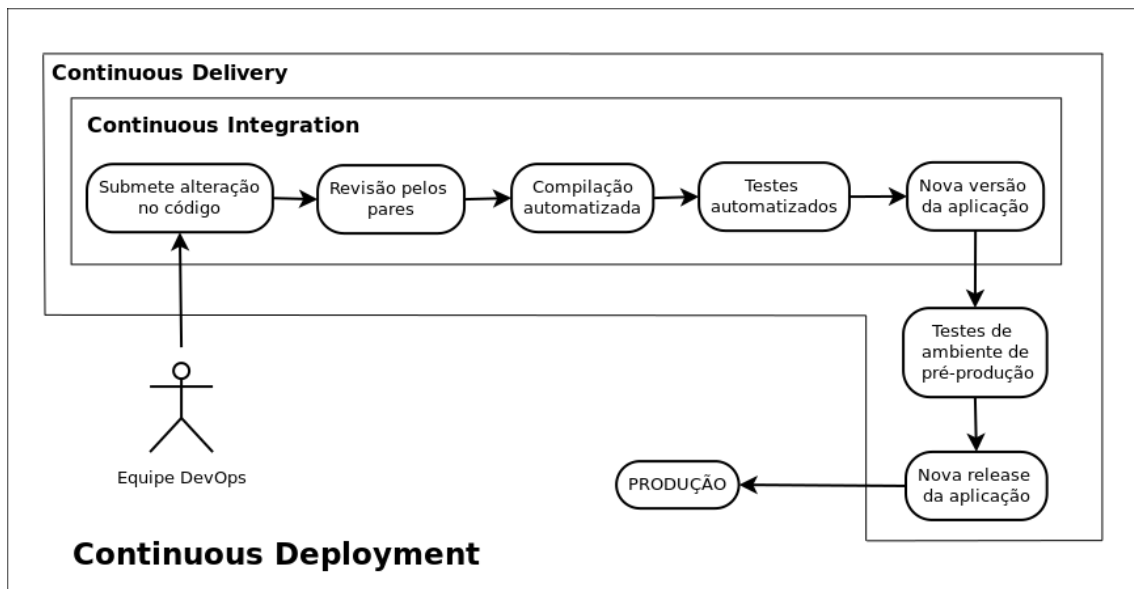


Figura 1: Pipelines utilizados no DevOps.

Embora não apareçam no diagrama acima, todos os resultados dos testes realizados, bem como as informações de desempenho da aplicação em produção, voltam como entradas para equipe DevOps.

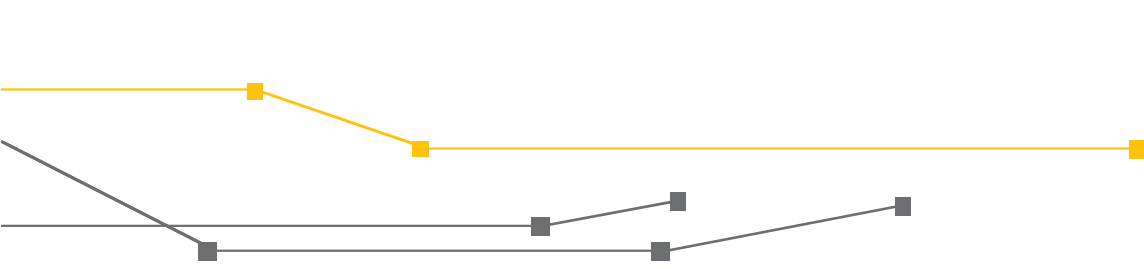
3. Infrastructure as Code

Infrastructure as Code (IaC) é uma abordagem em que a infraestrutura de rede e computadores é definida em código, portando podendo ser tratada como uma aplicação de software [3]. O IaC se torna então um pré-requisito para permitir a aplicação das práticas DevOps no contexto de infraestrutura de redes, seja ela física ou virtual.

O princípio básico da IaC é que o operador não deve acessar um novo equipamento, via *command line interface* ou outra ferramenta interativa, para configurá-lo manualmente com base em documentação. No lugar disto ele descreve em código (scripts, templates, profiles, playbooks) as configurações e ações necessárias para que o equipamento convirja para o estado desejado. [5]

Um conceito interessante trazido pelo IaC é o de configurações *snowflakes*, ou flocos de neve [4]. Assim como cada floco de neve é único, não existindo outro igual, configurações altamente customizadas de um equipamento geralmente não podem ser replicadas aos demais. Tais configurações costumam ser difíceis de manter, e devido a sua especificidade, não passíveis de serem automatizadas.

Principais práticas:

- 
- **Colaboração entre equipes** - O código é elaborado de forma colaborativa pelas equipes, utilizando sistemas de controle de versionamento distribuído (ex.: Git, Mercurial, Visual Studio Team Services, etc). Uma vez que todos os membros das equipes conseguem visualizar o código que está em produção, ou em vias de se tornar uma nova release para produção, podem então contribuir seja codificando ou sugerindo alterações/correções no código.
 - **Processo de testes contínuos** - As configurações são validadas e testadas antes de serem aplicadas, pelo uso de ferramentas de *Continuous Integration*.
 - **Versionamento** – É mantido um controle de versionamento de todas as alterações feitas no código, tornando possível acompanhar problemas devido alterações que não saíram conforme o esperado, e restaurar o equipamento para um estado anterior ao do problema, reaplicando uma versão anterior do código.
 - **Auto documentação** - No lugar de manter as configurações em documentos, o próprio código serve como documentação (templates de configuração, scripts, configurações aplicadas).

4. Implementando DevOps

As recomendações listadas nessa sessão foram retiradas primariamente do livro “Automating Junos Administration” [9], exceto a parte de segurança. São elas:

1. Identificar tarefas a serem automatizadas;
2. Escolher ferramentas a serem adotadas;
3. Inserir segurança nos pipelines;
4. Adotar práticas que facilitem a manutenção.

4.1. Identificar tarefas a serem automatizadas

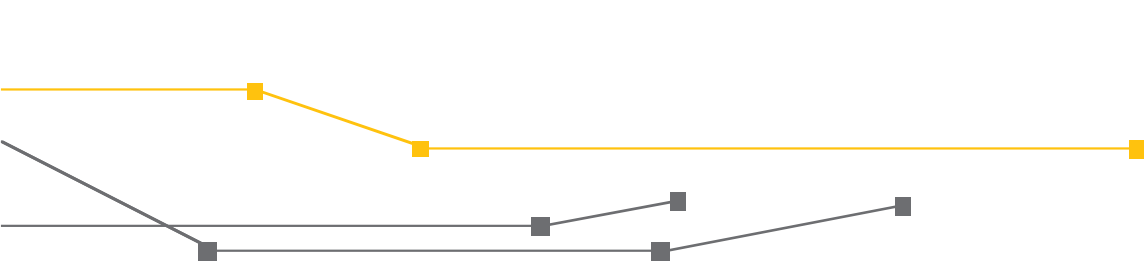
Os seguintes tipos de tarefas podem ser automatizados:

- tarefas repetidas muito frequentemente;
- tarefas muito infrequentes ou complicadas, por serem difíceis de lembrar;
- tarefas que necessitam de muito tempo para serem executadas manualmente;
- tarefas de troubleshooting que possam ser reduzidas a um conjunto de passos fixos;
- implantação de novos de serviços – se não de forma completa, pelo menos parcial;
- provisionamento inicial de novos equipamentos assim que eles são “tirados da caixa” – atualizações necessárias de sistema operacional, e realização de configurações iniciais

4.2. Escolher ferramentas a serem adotadas

Abaixo algumas questões a se considerar na escolha das soluções:

- As ferramentas possuem algum suporte para os equipamentos de rede que irão configurar?
- As ferramentas de automatização serão integradas a outros sistemas, como ferramentas de Continuous Integration/Continuous Delivery?

- 
- As ferramentas são escaláveis para um grande número de equipamentos/operações?
 - As ferramentas obedecem a padrões abertos? O código das ferramentas é aberto (“open source”)?

4.3. Inserir segurança nos pipelines

As recomendações abaixo se baseiam no artigo "10 Tips for Integrating Security into DevOps" [10]:

- Proteger as aplicações CI/CD e ambiente de desenvolvimento.
- Definir os níveis de autorização que cada usuário irá dispor.
- Validar e sanitizar as entradas fornecidas pelos usuários e ferramentas via testes executados de forma automatizada.
- Proteger o ambiente de CI/CD contra injeção de código malicioso. Um bom lugar para esconder estes códigos é nos scripts de testes, porque não costumam serem monitorados e rodam toda vez que ocorre um commit.
- Definir padrões para desenvolvimento de código que previnam abusos.

4.4. Adotar práticas que facilitem a manutenção

As práticas abaixo são recomendadas para facilitar o processo de manutenção da solução:

- Código deve ser bem comentado.
- Adotar sistema de controle de versionamento.
- Remover código não mais utilizado na rede.
- Utilizar sistemas de *bug tracking*.
- Definir metodologia para criação de versões a serem postas em produção.
- Uso de procedimentos de testes automatizados, tanto do código em desenvolvimento quanto do desempenho do aplicativo na rede.
- Definir procedimentos para tratamento de condições excepcionais - a ferramenta deve saber responder quando encontra situações anormais, e em alguns casos, ser capaz de reverter as configurações aplicadas.

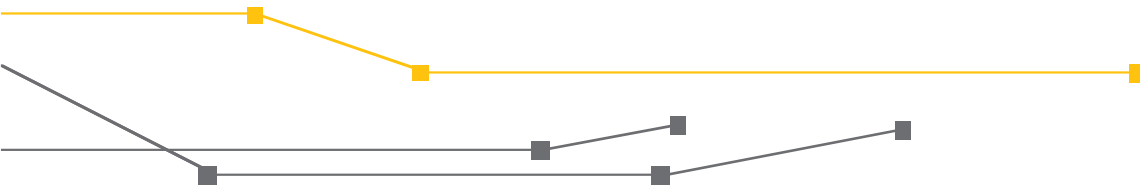
5. Estudo de Caso: Dyn

Um interessante estudo de caso de adoção de DevOps é descrito em uma palestra realizada por Carlos Vicente em 2015, no evento NANOG 63 [6]. A apresentação descreve a solução de automação de configurações de equipamentos de rede implementada na Dyn, empresa especializada em monitoramento e otimização de performance de redes de computadores.

Internamente o projeto teve o codinome de "Kipper". Carlos fez pelo menos duas outras apresentações desse caso [7][8].

5.1. Motivações

Carlos começa a apresentação comentando os problemas enfrentados com o processo anterior, quando as configurações eram manuais:



- Copy-and-pasting:
 - Fácil de se cometer erros;
 - Pode gerar inconsistências;
 - Mais demorado.
- Sem processo formal de aprovação.

Metas esperadas com o projeto de automação eram:

- Facilitar consistência das configurações via uso de templates;
- Minimizar erros - evitar o uso direto de CLI, uso de processo formal de aprovação e testes automatizados.
- Aumentar velocidade na aplicação de grande quantidade de mudanças.

5.2. Ferramentas adotadas

Foram utilizadas as seguintes ferramentas na solução:

- **Ansible (com uso de Netconf - RFC 6241)** - open source, push model, suporte Juniper.
- **GitHub** - versionamento - repositório privado.
- **Jenkins** - ferramenta popular de CI/CD - automatiza a execução de tarefas via crontab e eventos. No caso da Dyn, ele foi integrado com o GitHub, para tomar ações em caso de alterações no código.

5.3. Workflow da solução

Premissas da solução:

- Cada engenheiro possui uma cópia do código em repositório local;
- Haveria um grupo de administradores que aprovariam as alterações.

Workflow da solução:

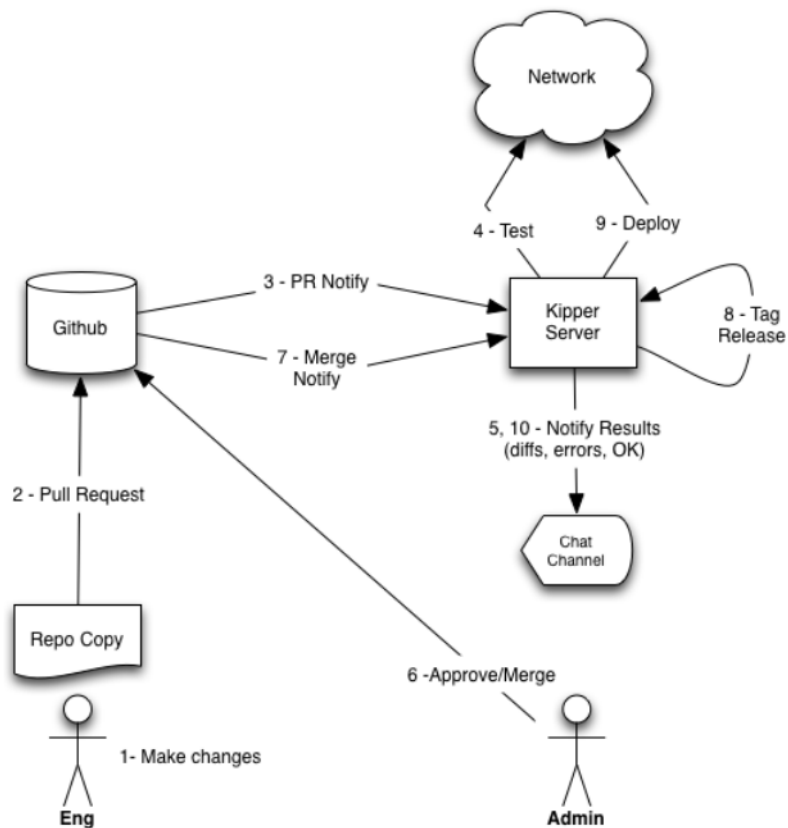
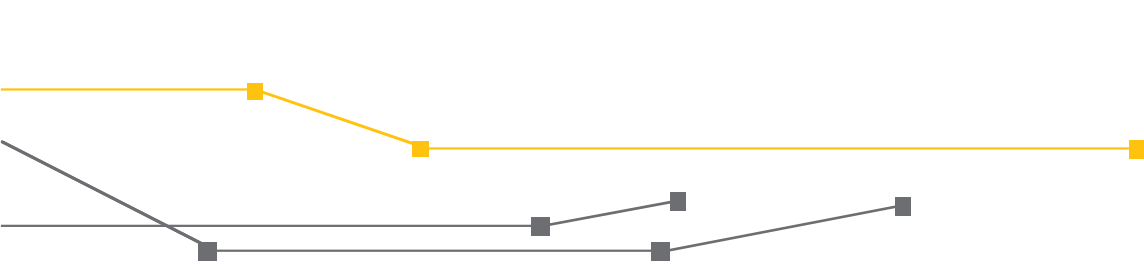


Figura 2: Esquema de automação Dyn Inc. [6]

1. Engenheiro faz uma alteração no repositório local.
2. A alteração submetida ao GitHub ("git push").
3. GitHub gera uma notificação de alteração para o Jenkins (nova "pull request").
4. Jenkins recebe a notificação de alteração:
 - 4.1 Faz download da alteração. ("git fetch")
 - 4.2 Efetua merge da alteração com o código em produção. ("git merge")
 - 4.3 Constrói as configurações com as novas alterações usando os templates. (Ansible)
 - 4.4 Efetua "dry run" na rede - verifica a sintax (aplica as configurações nos equipamentos, efetua "commit check") e o que será alterado no equipamento ("show | compare"). (Ansible)
5. Jenkins coloca os resultados dos comandos em um Gits - feature do GitHub que funciona como uma entrada de um blog sobre o que foi alterado - mantendo-se dessa forma um histórico. O grupo de administradores é notificado.
6. Administrador examina o Gits, e aprova o novo código no GitHub.
7. GitHub gera nova notificação para o Jenkins.

- 
8. Jenkins recebe a notificação de aprovação do código e cria um novo release, e aplica uma nova tag (Ex.: "`git tag -a v2.0 -m 'Release 2.0 lançada'`").
 9. Jenkins utiliza o Ansible para aplicar as novas configurações na rede.
 10. Jenkins retorna logs da aplicação e eventuais erros.

5.4. Lições aprendidas e recomendações

Para empresas querendo fazer uso de automatização foi sugerido como estratégia de implementação:

1. Começar por partes mais simples - contas de usuários, NTP, DNS, SNMP, prefix-list - na direção convergência total.
2. Com a convergência total, os templates se tornam a "regra", e podem ser testados periodicamente via *dry run* ("`show | compare`"), para ver se existe algo fora do padrão. Caso exista, gerar notificações – cabendo então decidir do que fazer: alterar os templates ou restaurar a rede para seu estado anterior.

Desafios a serem enfrentados no processo:

- Mudança cultural e familiaridade com as novas ferramentas.
- Problemas nos *dry runs*, devido a alguém ter bloqueado as configurações para fazer alterações.
- O que fazer quando os administradores que aprovam as alterações não estiverem disponíveis quando houver uma emergência.

6. Ferramentas recomendadas

Com base na bibliografia levantada, foram escolhidas algumas ferramentas utilizadas na implantação de ambientes DevOps.

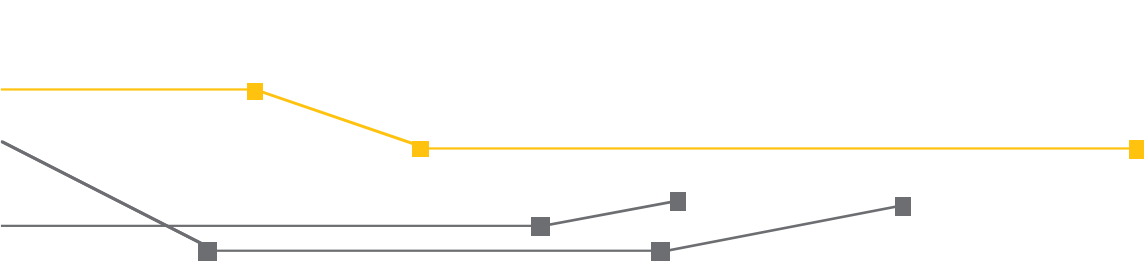


Git é um sistema de versionamento distribuído, criado 2005 pelos desenvolvedores do kernel do Linux, para substituir o sistema anteriormente em uso. [11]

Pode-se considerar o Git como um mini sistema de arquivos, onde todas as alterações no diretório do projeto, e subdiretórios, são controladas. Ele funciona tirando *snapshots* dos arquivos alterados toda vez que uma operação de *commit* é executada.

Os arquivos podem estar em quatro estados:

1. **Unstaged** – são todos os arquivos do diretório de desenvolvimento que foram modificados desde o último commit – esses arquivos necessitam serem marcados como *staged* para posteriormente terem suas alterações armazenados no próximo commit.

- 
2. **Staged** - neste estado os arquivos estão prontos para terem suas alterações armazenadas permanentemente – quaisquer alterações nesses arquivos passam a ser monitoradas;
 3. **Modified** - sempre que um arquivo em *staged* é modificado, ele muda para o estado “modified”, e necessita ser novamente ser marcado para *staged*;
 4. **Committed** - arquivos armazenados permanentemente, ocorre sempre que uma operação de *commit* é executada.

O Git permite a criação de *branches* ou ramos. Quando um desenvolvedor decide adicionar novas funcionalidades ao código, ou corrigir defeitos, ele tem a opção de criar um ramo do código, efetuar as alterações necessárias, e depois fazer a fusão (*merge*) das alterações no ramo principal.

O pacote de software do Git, utilizado do lado do cliente, é composto por um conjunto de programas executados em linha de comando. Embora existam interfaces gráficas, é interessante integrar o Git diretamente no editor de textos do ambiente de desenvolvimento, pois isso facilita sua utilização. Sublime, Atom e Visual Studio Code são algumas das opções de editores com suporte nativo.

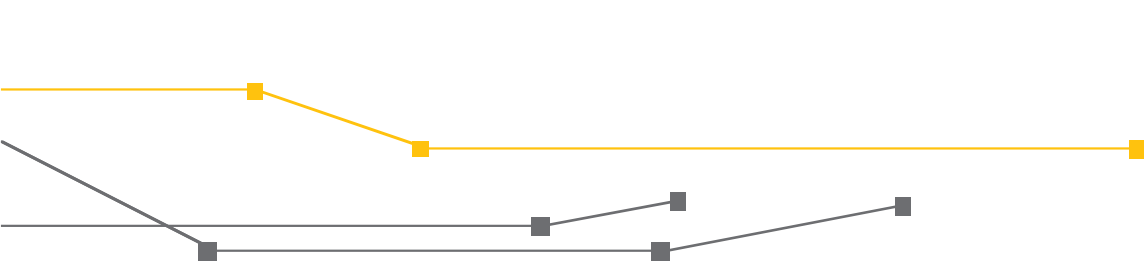
A grande força do Git é quando ele é utilizado de forma a permitir a colaboração de diferentes desenvolvedores remotos. Para isso é feito uso de um servidor centralizado, que pode ser privado ou contratado como serviço de uma empresa de hospedagem. Algumas empresas como a GitHub, Microsoft, GitLab, Bitbucket, fornecem serviços de hospedagem – destes o GitHub costuma ser o mais conhecido, por fornecer hospedagem gratuita para muitos projetos de código aberto.

Ansible

Ansible [12] é um framework de automação de configuração de equipamentos de rede e servidores bastante popular. A companhia foi adquirida em 2015 pela Red Hat. Ele é disponibilizado em duas versões, a “Ansible Core”, que é tanto de código aberto como livre, e o “Ansible Tower”, que é a versão comercial.

O Ansible foi criado em linguagem de programação Python, sendo que o código fonte da versão aberta está disponível no GitHub. Embora não seja necessário conhecimento na linguagem de programação, a plataforma faz uso de arquivos de configuração em formato YAML, e utiliza o formato Jinja2 para expressar templates de configuração, o que torna necessário o conhecimento desses dois formatos.

A plataforma opera na arquitetura cliente/servidor. Em seu modo padrão, servidor Ansible faz o upload do código que necessita executar nas estações clientes, executa o código, e recupera os resultados da execução, só então o código é removido das estações clientes.



No contexto do DevOps, o modo padrão de operação é alterado. O Ansible utiliza o protocolo NETCONF (RFC 6241) para enviar comandos aos equipamentos e receber o resultado da aplicação. Uma sessão SSH é estabelecida entre o servidor e os equipamentos, para garantir a confidencialidade dos dados sendo transmitidos. Alguns módulos do Ansible também permitem o envio de comandos para a CLI dos equipamentos.

SaltStack

Embora a ferramenta seja referenciada como SaltStack [13], este nome é na verdade o da empresa que desenvolve o Salt Open, mais conhecido como Salt. O Salt é um framework de código aberto escrito em Python para configuração de equipamentos, portanto similar ao Ansible. Além de manter a versão de código aberto, a empresa também comercializa uma versão comercial, o SaltStack Enterprise.

Jenkins

Jenkins [14] é um servidor de automação open source, escrito em Java. Ele é normalmente utilizado em ambientes de desenvolvimento de software para automatizar as tarefas de compilação, teste e implantação.

A ferramenta tem um papel importante, no contexto DevOps, por permitir estabelecer os *pipelines* de *Continuous Integration* e *Deployment*. A apresentação de Carlos Vicente [6], descrita na seção anterior deste documento, menciona o uso desta ferramenta.

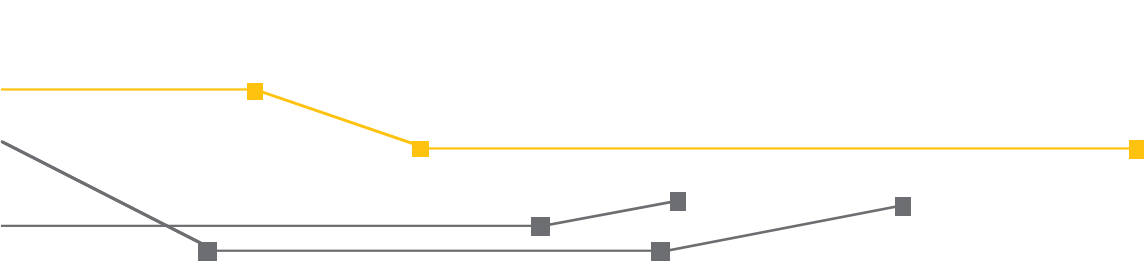
NAPALM

NAPALM (Network Automation and Programmability Abstraction Layer with Multivendor) [15] é uma biblioteca em Python que permite configurar e executar comandos em equipamentos de redes de diversos fabricantes, utilizando uma API comum.

Ela pode ser utilizada de forma independente ou integrada a outros frameworks, como Ansible ou SaltStack, sendo que neste último a integração já é nativa.

7. Considerações Finais

O uso de automatização se tornou inevitável, devido ao crescente número de equipamentos com os quais os provedores Internet têm de lidar. Surge então o desafio de como garantir a



consistência das alterações efetuadas nos equipamentos, considerando que o número destes pode ir de algumas poucas dezenas para casos de milhares, como restaurar a rede para um estado anterior a um erro induzido após a aplicação de uma alteração, e ao mesmo tempo garantir que as equipes trabalhem de forma integrada a fim de agilizar a implementação de mudanças e diminuir a ocorrência erros.

DevOps tem sido adotado para endereçar esses desafios. Mais que um conjunto de aplicações para automatizar o processo de operação de redes, é um novo paradigma, que traz para área de infraestrutura de redes as lições aprendidas na área de desenvolvimento de software.

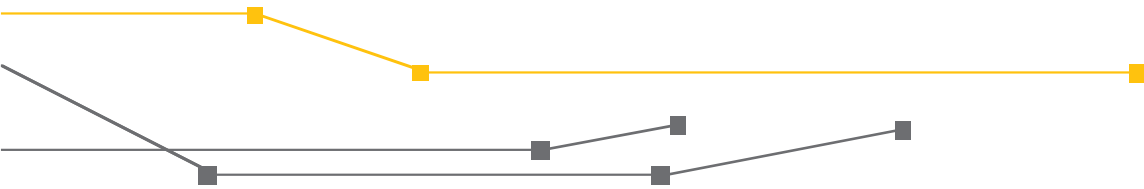
Ele busca integrar as equipes internas, estabelecer workflows bem definidos para operação e contínuo incremento da rede, mas sua adoção requer uma mudança na cultura interna da área técnica da empresa. Independentemente da adoção de suas práticas, as ferramentas que têm sido desenvolvidas para suportar o ambiente DevOps vieram para ficar.

8. Agradecimentos

Agradeço a Vinicius Arcanjo, do NEG de AmLight, pelas indicações e referências repassadas, e a Fabio Okamura, da Gerencia de Engenharia de Redes da RNP, pelo apoio nos testes.

9. Referências

- [1] “DevOps” - <https://en.wikipedia.org/wiki/DevOps>
- [2] “Continuous integration” - <https://martinfowler.com/articles/continuousIntegration.html>
- [3] “Infrastructure as Code” - <https://martinfowler.com/bliki/InfrastructureAsCode.html>
- [4] “What is Infrastructure as Code?” - Sam Guckenheimer
<https://www.visualstudio.com/learn/what-is-infrastructure-as-code/>
- [5] “Infrastructure as code: The engine at the heart of DevOps” -
<https://techbeacon.com/infrastructure-code-engine-heart-devops>
- [6] Apresentação NANOG 63 - “Automatically Build, Test and Deploy Your Network Configurations” - Carlos Vicente, Dyn Inc. -
<https://www.youtube.com/watch?v=OWLTBYgPp0A>
- [7] Apresentação NANOG 67 - “Experiences with network automation at Dyn” - Carlos Vicente, Dyn Inc. - <https://www.youtube.com/watch?v=a4s15nmjDkE>
- [8] Canal do Youtube JuniperNetworks - “Infrastructure as Code with Ansible” -
https://www.youtube.com/watch?v=HXYL3_4_RBI
- [9] “Automating Junos Administration” - Jonathan Looney e Stacy Smith – 2016 - Editora O’Reilly



[10] "10 Tips for Integrating Security into DevOps" -

<https://blog.xebialabs.com/2017/04/20/10-tips-integrating-security-devops/>

[11] "Pro Git" - Scott Chacon and Ben Straub - segunda edição, 2014 - <https://git-scm.com>

[12] Ansible - <https://www.ansible.com>

[13] SaltStack - <https://saltstack.com>

[14] Jenkins - <https://jenkins.io>

[15] Napalm - <https://github.com/napalm-automation/napalm>



MINISTÉRIO DA
DEFESA

MINISTÉRIO DA
CULTURA

MINISTÉRIO DA
SAÚDE

MINISTÉRIO DA
EDUCAÇÃO

MINISTÉRIO DA
**CIÊNCIA, TECNOLOGIA,
INOVAÇÕES E COMUNICAÇÕES**

