

---

**525**

**Infraestrutura Ágil com  
Práticas DevOps**



# Sumário

<b>1</b>	<b>Cultura DevOps</b>	<b>2</b>
	A História do DevOps . . . . .	2
	Infraestrutura Como Código . . . . .	5
	Conhecendo as Ferramentas . . . . .	6
<b>2</b>	<b>Desenvolvimento de ambientes virtuais com HashiCorp Vagrant</b>	<b>13</b>
	Conceitos de Virtualização . . . . .	13
	Definição de Máquinas Virtuais no Vagrant . . . . .	16
	Variáveis, Loops e Condições no Vagrant . . . . .	19
	Provisionadores . . . . .	23
<b>3</b>	<b>Versionamento de Código com GIT</b>	<b>27</b>
	Versionamento de Código Local . . . . .	27
	Comandos essenciais do GIT . . . . .	31
	Trabalhando com Branches em projetos . . . . .	33
	Ignorando arquivos no versionamento . . . . .	36
<b>4</b>	<b>Repositório Remoto com Gogs</b>	<b>39</b>
	Introdução e Instalação do Gogs . . . . .	39
	Utilizando Repositórios Remotos . . . . .	45
<b>5</b>	<b>Automação para ambientes ágeis com Ansible</b>	<b>55</b>
	Características do Ansible . . . . .	55
	Usando o Ansible Ad-hoc . . . . .	58
	Introdução a Linguagem YAML . . . . .	60
	Execuções em Playbooks . . . . .	62
<b>6</b>	<b>Construindo a Infraestrutura como Código</b>	<b>67</b>
	Ansible Galaxy . . . . .	67
	Construindo a Role puppet-agent . . . . .	69
	Construindo a Role nginx . . . . .	71
	Construindo a Role Rundeck . . . . .	73
	Construindo a Role Docker . . . . .	75
<b>7</b>	<b>Gerência de Configuração com Puppet</b>	<b>78</b>
	Estrutura do Puppet . . . . .	78
	Usando o Puppet em modo Autônomo e escrevendo Manifestos . . . . .	81
	PuppetServer . . . . .	85
	Módulos e Puppet Forge . . . . .	88
<b>8</b>	<b>Construindo pipelines de Infraestrutura com Rundeck</b>	<b>98</b>
	Características do Rundeck . . . . .	98
	Registrando e Verificando Nodes . . . . .	109
	Construindo Jobs . . . . .	121
	Integrações com SCM . . . . .	130

<b>9</b>	<b>Containers com Docker</b>	<b>154</b>
	Conceitos de Containers . . . . .	154
	Comandos essenciais do Docker . . . . .	156
	Docker Volumes e Network . . . . .	165
	Docker Compose . . . . .	174
<b>10</b>	<b>Teste de Infraestrutura com InSpec</b>	<b>187</b>
	Características do InSpec . . . . .	187
	Escrevendo Testes . . . . .	191
	Integrações . . . . .	205
<b>11</b>	<b>Gerenciando Logs com Elastic Stack</b>	<b>216</b>
	Características do Elastic Stack . . . . .	216
	Configuração do Elastic Stack . . . . .	219
	Beats . . . . .	223
	Construindo Dashboards com Kibana . . . . .	229
<b>12</b>	<b>Implementando ChatOps com Rocket.Chat</b>	<b>257</b>
	Conceitos e Características ChatOps . . . . .	257
	Instalando e Configurando o Rocket.Chat . . . . .	258
	Interações . . . . .	271
	Implantando e Utilizando ChatBots . . . . .	302

# Capítulo 1

## Cultura DevOps

### A História do DevOps

#### A Cultura

**DevOps (Development + Operations)** é um processo de desenvolvimento e entrega de software que enfatiza a comunicação entre os profissionais de desenvolvimento e operações, promovendo quebra de silos e paradigmas interdepartamentais tratando as equipes como único time para entregar juntos com excelência.

O termo DevOps tem tomado uma grande importância no mundo todo. Empresas como Amazon, Netflix, Facebook, Walmart, Spotify, entre tantas outras, estão investindo em funcionários, ferramentas e ambientes que sigam as práticas DevOps para melhorar a entrega e a qualidade de seus produtos.

Objetivos de um ambiente com práticas DevOps: \* Melhorar a frequência dos deploys; \* Automatizar processos; \* Diminuir a ocorrência de erros em novas versões; \* Curtos períodos de tempo para mudanças e melhorias; \* Recuperação rápida em caso de falhas no ambiente; \* Padronização nos processos de configuração e servidores;

#### Manifesto Ágil

A utilização de métodos ágeis para desenvolvimento como o Lean, Scrum, Extreme Programming (XP), entre outras já é feita há algumas décadas, todas com o intuito de auxiliar o desenvolvedor a entregar seus projetos o mais rápido possível e ter uma fácil resposta a mudanças. Porém, somente em 2001 que um grupo de 17 desenvolvedores se reuniram em um resort de ski em Utah para realmente discutir quais os pontos positivos em diversas metodologias que utilizavam.

A partir dessa discussão, surgiu o **Manifesto para Desenvolvimento Ágil de Softwares**, que ficou popularmente conhecido como **Manifesto Ágil**. O Manifesto Ágil afirma que melhores resultados no desenvolvimento de software podem ser obtidos através da valorização de: \* Indivíduos e interações, mais que processos e ferramentas; \* Software em funcionamento, mais que documentação abrangente; \* Negociação com o cliente, mais que negociação de contratos; \* Responder a mudanças, mais que seguir um plano.

#### Agile Conference

O termo DevOps surgiu em 2009, porém a ideia por trás já existia anos antes. Entre 2001 e 2008 diversas discussões começaram sobre como agilizar as entregas, sendo uma delas a palestra **Infraestrutura Ágil** de **Andrew Schafer** durante a Agile Conference de 2008, onde Andrew conheceu Patrick Debois.

Essa foi a primeira vez que houve uma discussão sobre o assunto. Após a palestra ambos discutiram diversos assuntos relacionados, e posteriormente iriam criar o grupo **Agile System Administrator** na plataforma Google Docs para difundir o assunto e convidar mais pessoas para colaborar.



Figura 1.1: Manifesto Ágil



Figura 1.2: Agile Conference



Velocity

Figura 1.3: Velocity Conference

## Velocity Conference

Em 2009 na Velocity Conference da O'Reilly, John Allspaw e Paul Hammond apresentaram a famosa palestra chamada “**10+ Deploys per Day: Dev and Ops Cooperation at Flickr**”. Durante a palestra foram discutidos assuntos como: \* Interação entre os desenvolvedores e a equipe de operações; \* Como conseguir um aumento dos deploys com ferramentas e mudanças culturais;

Patrick Debois comentou no twitter que lamentava não estar presente na palestra recebendo do Flickr a seguinte resposta: “*Porque não organizar sua própria conferência de Velocity na Bélgica?*”

## DevOpsDays



Figura 1.4: DevOpsDays

Motivado pela palestra dos engenheiros da Flickr, Patrick Debois decidiu criar sua própria conferência na Bélgica que ficou conhecida como DevOpsDays e foi um sucesso. Para lembrar o dia, foi criada a tag #DevOps no Twitter, que posteriormente seria usado como o nome da cultura.

## Atualidade

Desde então, cada vez mais pessoas estão se tornando adeptas da cultura DevOps. Alguns dos acontecimentos mais recentes que favoreceram isso foram o surgimento de diversas tecnologias de apoio a essa ideologia. Phoenix Project - O Projeto Fênix é uma ficção que conta como um gerente de TI salvou o departamento de uma companhia com a cultura DevOps.

## Infraestrutura Como Código

**Infraestrutura como código (Infrastructure as a code)** é uma forma de construir e gerenciar uma infraestrutura da mesma maneira que ocorre no desenvolvimento de software. É também um processo de gerenciamento e provisionamento de recursos de infraestruturas, onde todas as atividades e passos para sua criação e configuração são registrados em scripts,



Figura 1.5: Infra-as-a-Code

utilizados por ferramentas de automação (como Ansible e Puppet) e passam pelo mesmo processo de versionamento que um software.

Com esse processo torna-se mais fácil a gestão de ambientes, pois garante que novos serviços possuam as mesmas configurações e versões de pacotes. Outro ponto positivo é que o profissional de operação evita se preocupar em configurar cada servidor, e passa a se preocupar em codificar o ambiente.

Objetivos de uma **Infraestrutura como Código**: \* Aproveitar o tempo com melhorias ao invés de tarefas rotineiras e repetitivas; \* Recuperar rapidamente o ambiente em caso de falhas ou perda total; \* Possibilitar mudanças na infraestrutura de forma automatizada; \* Garantir ambientes idênticos, baseados na mesma configuração; \* Testar as mudanças antes que entrem em produção;

## Desenvolvimento Ágil

**Desenvolvimento Ágil** de software é um conjunto de metodologias de desenvolvimento de software. Alguns princípios da metodologia ágil são: \* Garantir a satisfação do cliente entregando rapidamente e continuamente; \* Rápida adaptação a mudanças; \* Simplicidade; \* O Design do software deve prezar pela excelência técnica; \* Cooperação constante entre pessoas que entendem do negócio e desenvolvedores;

## Ambientes transitórios

A Virtualização e a Computação em Nuvem (Cloud Computing) também disponibilizam a possibilidade de construir ambientes transitórios. **Ambientes transitórios** são ambientes de curta duração que são terminados com uma certa frequência

Tipo de Ambiente	Descrição
<b>Ambientes baseados em Scripts</b>	São completamente baseados em scripts, atribuídos em versões e testados.
<b>Ambientes Self-Service</b>	Qualquer pessoa autorizada pode ativar um novo ambiente.
<b>Término Automático</b>	Os ambientes são automaticamente finalizados, de acordo com a política determinada na empresa.

## Conhecendo as Ferramentas

Em nosso laboratório utilizaremos diversas ferramentas para práticas DevOps:

- Vagrant
- Ansible
- Gogs
- Rundeck
- Docker
- Puppet
- Elastic Stack
- Rocket Chat

## Topologia

### Vagrant

7

Inicialmente vamos realizar a construção do nosso ambiente como código. Através desse código será feito o provisionamento de todas as máquinas virtuais usando o Vagrant, um construtor de ambientes virtuais desenvolvido pela HashiCorp.

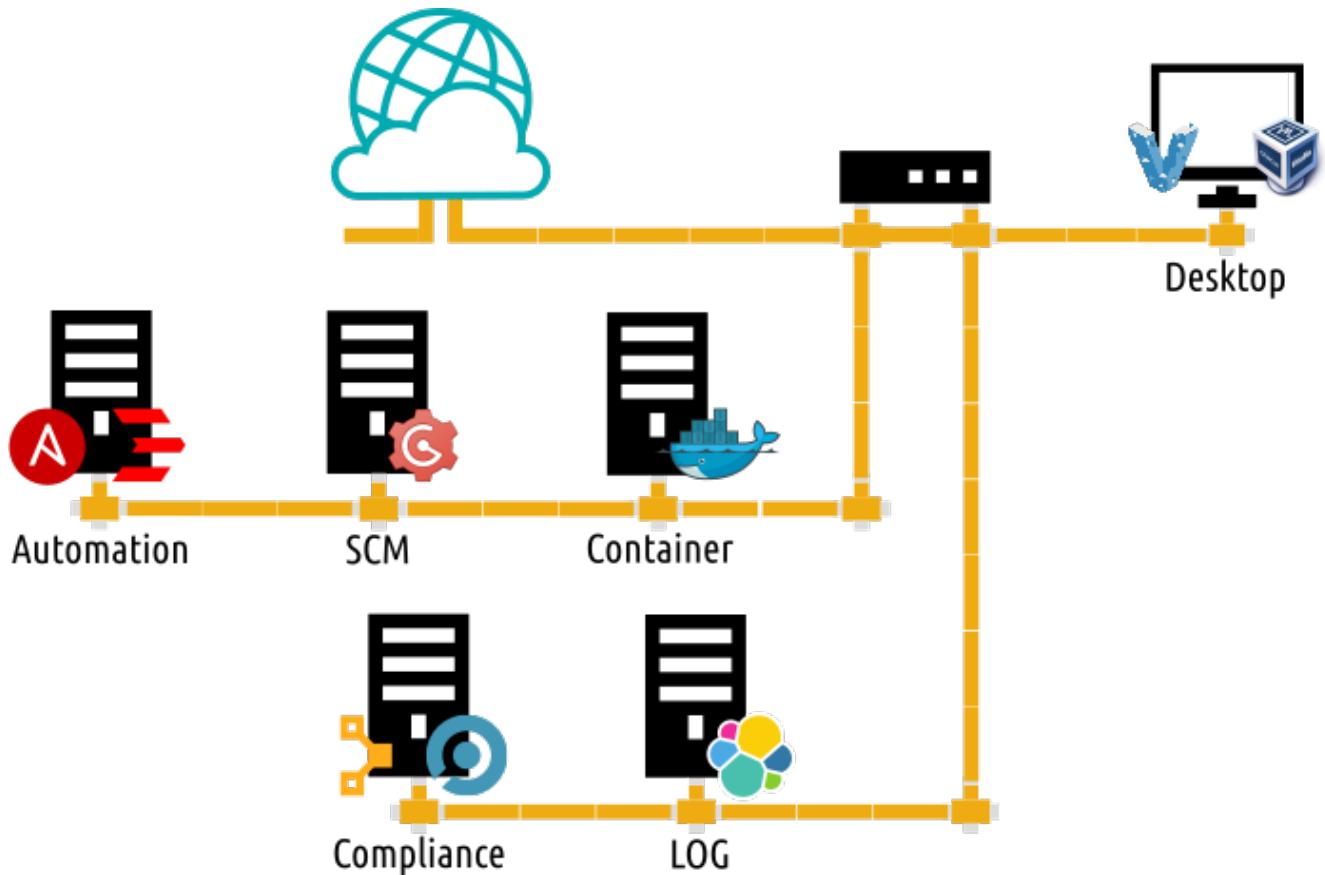


Figura 1.6: Topologia



HashiCorp  
**Vagrant**

Figura 1.7: Vagrant



ANSIBLE

Figura 1.8: Ansible

## Gogs



Figura 1.9: Gogs

O Próximo passo será criar uma maneira de versionarmos o código de nossa infraestrutura. Para esse versionamento, será usado a ferramenta Gogs (Go Git Service), uma aplicação escrita em GoLang para gerenciamento de repositórios GIT, permitindo assim a colaboração entre pessoas nos projetos.

## Rundeck



Figura 1.10: Rundeck

Utilizaremos o Rundeck para automatização da nossa infraestrutura e deploy, uma plataforma Open-Source de gerenciamento de operações que conecta processos e ferramentas. Através do Rundeck iremos interligar as ferramentas a fim de entregar nosso ambiente provisionado de maneira automática.

## Puppet

Para manter esse ambiente será necessário a utilização de uma ferramenta de gerenciamento de configurações. Neste caso, vamos usar o Puppet com o código sendo versionado e validado da mesma maneira que faremos com o Ansible.



Figura 1.11: Puppet

## Docker



Figura 1.12: Docker

Através do Docker podemos provisionar aplicações em containers para sustentação da nossa infraestrutura. Atualmente o Docker é a ferramenta de containers mais utilizada no mercado.

## Elastic Stack

Quando se tem um parque muito grande de máquinas é necessário o uso de uma ferramenta para centralizar o armazenamento e visualização de logs. Utilizaremos a solução Elastic Stack (Elasticsearch + Logstash + Kibana) dessa forma será permitido concentrar os logs de todas as máquinas em um único local para análise.

## Rocket.Chat

Utilizaremos o RocketChat como ferramenta de colaboração. Iremos integrar para que as ações realizadas no Gogs e no Rundeck sejam refletidas no Chat de Operações. Também utilizaremos o Hubot para disparar jobs do Rundeck através da metodologia ChatOps.

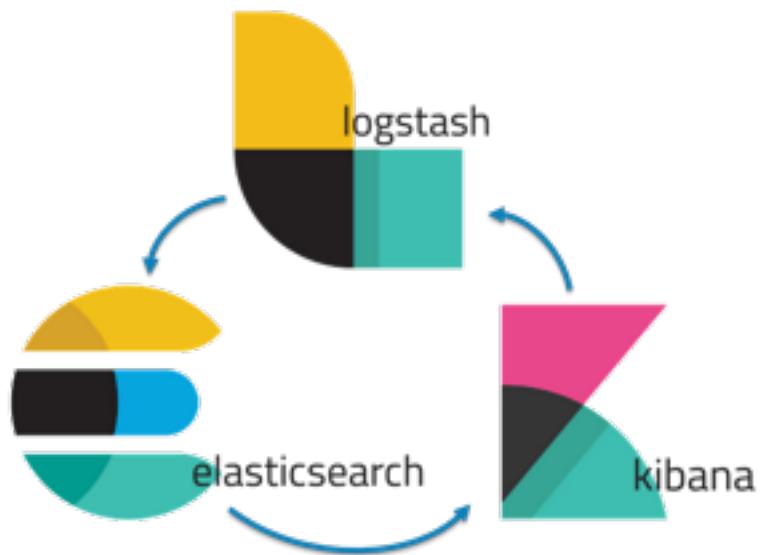


Figura 1.13: ElasticStack



Figura 1.14: Rocketchat

## Capítulo 2

# Desenvolvimento de ambientes virtuais com HashiCorp Vagrant

### Conceitos de Virtualização

#### O que é Virtualização

Virtualização é o processo de criar uma versão virtual de algum recurso, por exemplo uma rede de servidores, sistemas operacionais, dispositivos de armazenamento, etc...

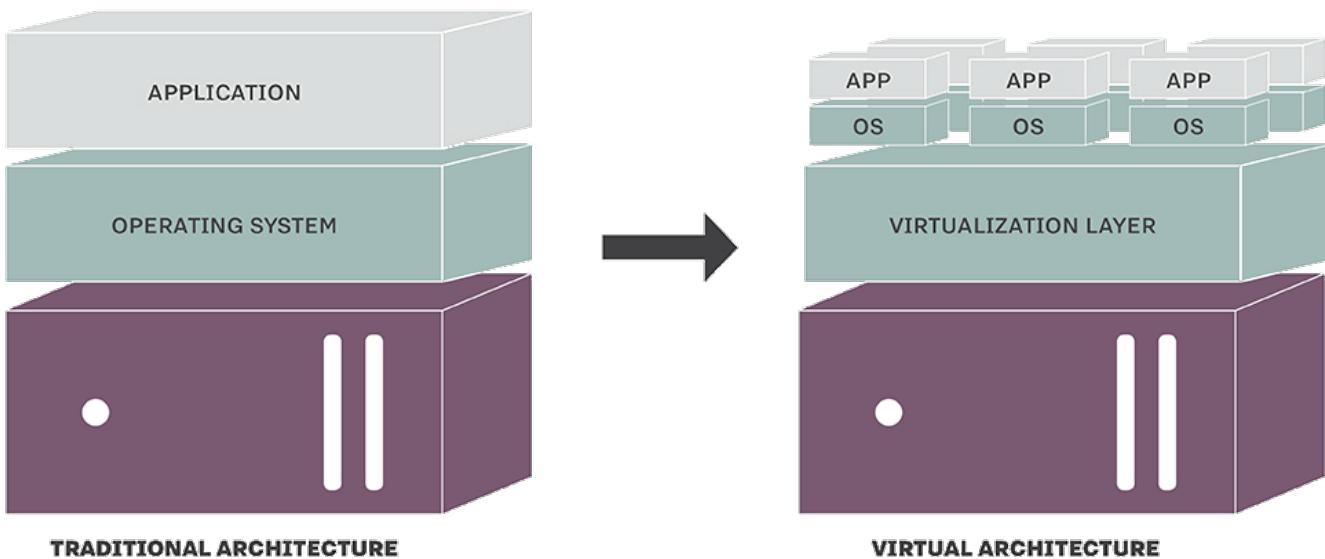


Figura 2.1: Virtualization

### Benefícios da virtualização

#### Gerenciamento de recurso

Agora podemos gerenciar o quanto de recurso computacional cada serviço irá consumir, considerando que cada máquina virtual manterá um serviço rodando.

## **Segurança**

Máquinas virtuais proporcionam maior segurança devido a isolação entre elas, sua conversa com a máquina hospedeira é apenas para operação do hardware virtual

## **Automação**

Atualmente existem diversas ferramentas que permitem criar infraestruturas virtualizadas em questão de segundos

## **O que é o Vagrant**



Figura 2.2: Vagrant

Vagrant é uma ferramenta de automação escrita em Ruby e mantida pela HashiCorp, a mesma empresa responsável por outras ferramentas como o Terraform, Vault e o Nomad. Através de um script é possível subir uma infraestrutura inteira e até mesmo configurar suas máquinas.

O Vagrant possui uma linguagem própria para os scripts baseada em Ruby e permite descrever todos os aspectos de uma máquina virtual, como hostname, ip, cpu, memoria, disco, sistema operacional, quantidades de máquinas que serão criadas, scripts que devem ser executados para o provisionamento, entre outros aspectos. Toda máquina criada pelo Vagrant é feita através de uma **Vagrantbox**, que é uma imagem base com todos os pacotes que uma máquina virtual deve possuir quando for criada.

## **Provedor x Provisionador**

O Vagrant trabalha com dois papéis distintos:

## Provedor

O provedor (**provider**) é o responsável pela criação da instância dos ambientes, geralmente uma máquina virtual, podendo também ser uma instância em alguma Cloud.

## Provisionador

O provisionador (**provisioner**) é o responsável pelas tarefas a serem executadas de forma automatizada, como por exemplo a instalação de pacotes e configuração do sistema.

## Fluxo de Criação de uma máquina virtual:

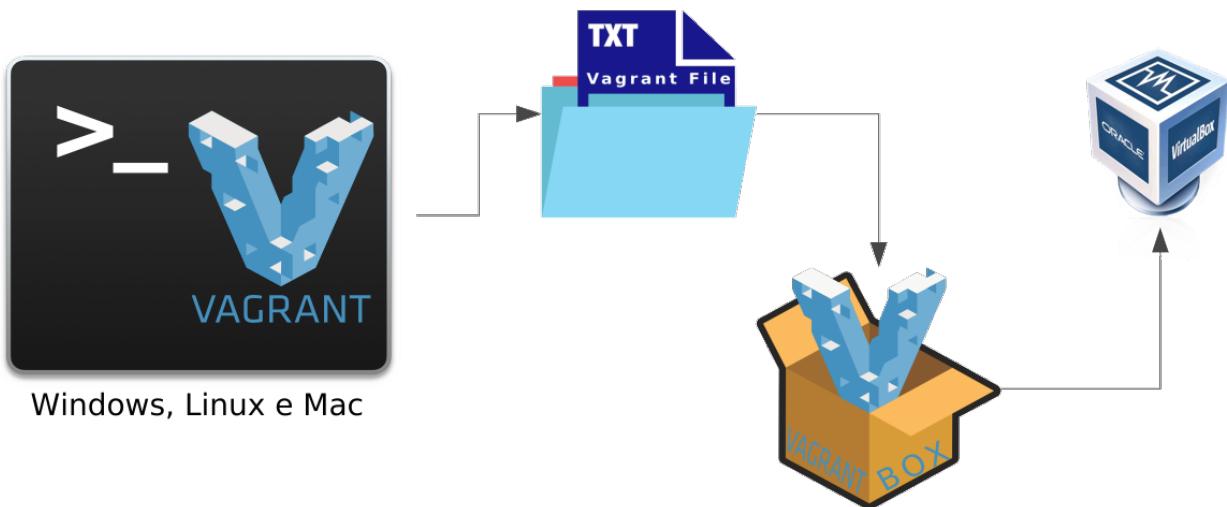


Figura 2.3: Fluxo Vagrant

A construção de uma máquina virtual pelo Vagrant se inicia com o download de uma imagem base através da Vagrant Cloud, após o download é iniciado o processo de criação e parametrização da máquina no virtualizador (hypervisor) escolhido.

Por padrão o Vagrant possui suporte a diversos providers como Virtualbox, Docker, Hyper-V, podendo ser expansível através de plugins a outras soluções como VMWare e AWS.

É importante que seja verificado se a imagem em questão fornece suporte ao Hypervisor escolhido. Após a criação da máquina no Hypervisor, o processo de execução do provisionador é executado.

## Instalação do Vagrant

O Processo de instalação do Vagrant é bem simples, basta baixar o instalador correspondente ao seu sistema operacional, instalar com seu gerenciador de pacotes e então validar a Instalação

Baixe o Instalador do Vagrant:

```
1 wget https://releases.hashicorp.com/vagrant/2.2.6/vagrant_2.2.6_x86_64.deb
```

Instale o Vagrant através do DPKG

```
1 sudo dpkg -i vagrant_2.2.6_x86_64.deb
```

Valide se o processo de instalação ocorreu com sucesso :

```
1 vagrant --version
```

## Definição de Máquinas Virtuais no Vagrant

### Entendendo o Vagrantfile

Vagrantfile é o nome do arquivo que contém instruções do que o Vagrant deve fazer para criar uma máquina virtual. O arquivo que deverá conter todas as instruções de criação de sua instância, como a quantidade de memória, cpu, rede, box, entre outras opções.

O Vagrantfile é baseado em Ruby, e por isso segue as mesmas regras de sintaxe dessa linguagem de programação.

```
1 Vagrant.configure("2") do |config|
2   config.vm.box = "centos/7"
3   config.vm.provider "virtualbox" do |vb|
4     vb.memory = "1024"
5     vb.cpus = "2"
6   end
7 end
```

No exemplo acima estamos definindo uma infraestrutura de apenas uma máquina com a box do CentOS 7, a máquina terá 1024 MB de memória e 2 vCPUs.

*É importante que o arquivo tenha o nome de **Vagrantfile** (case-sensitive) ou o comando do vagrant não irá localiza-lo.*

Toda estrutura de um Vagrantfile começa informando a versão que irá utilizar do configfile e termina com a palavra end.

**Vagrant.configure("2") do |config| (...) end**

O próximo passo é informar qual a imagem que iremos utilizar nessa instância, nesse caso usaremos a imagem chamada **centos/7**

**config.vm.box = "centos/7"**

Por último, criamos uma estrutura onde informamos qual é o provedor de nossa instância e suas configurações como cpu, memória, hostname, ip, etc...

**config.vm.provider "virtualbox" do |vb|**

### Criando uma instância

Vamos criar uma instância com 1GiB de memória e 2 vCPU's, igual ao exemplo anterior. O Vagrant permite que você gere o arquivo utilizando o parâmetro **init**, porém também é possível criar o arquivo manualmente, desde que o nome seja respeitado.

Gere o arquivo Vagrantfile com o mínimo de configuração:

```
1 vagrant init -m centos/7
```

*Ao executar o comando, um arquivo Vagrantfile é criado com diversas opções já preenchidas, porém comentadas. Para quem está trabalhando pela primeira vez com vagrant, talvez seja a melhor opção, caso contrário use a opção -m ou -minimal para remover todas essas linhas e deixar somente o necessário.*

Edite o arquivo para ficar igual ao exemplo:

```
1 vim Vagrantfile
```

```
1 Vagrant.configure("2") do |config|
2   config.vm.box = "centos/7"
3   config.vm.provider "virtualbox" do |vb|
4     vb.memory = "1024"
5     vb.cpus = "2"
6   end
7 end
```

Conseguimos verificar se a syntax do arquivo editado está correta:

```
1 vagrant validate
```

Construa seu ambiente de acordo com o que se encontra no Vagrantfile:

```
1 vagrant up
```

Para validar se a criação da instância ocorreu com sucesso, podemos acessa-la através do parametro **ssh** e em seguida desconectar da mesma

Acessando sua instância via ssh

```
1 vagrant ssh
2 exit
```

O comando **vagrant ssh** efetua a conexão com a máquina descrita no Vagrantfile, também é possível especificar o nome ou id da máquina após o parâmetro **ssh** para conectar-se a uma máquina específica.

Podemos verificar as máquinas provenientes de um Vagrantfile utilizando a opção status

```
1 vagrant status
```

Também é possível listar todas as máquinas criadas através do **vagrant** sem a necessidade de estar no diretório do Vagrantfile através da opção **global-status**

```
1 vagrant global-status
```

Para desligar a instância, utilizamos a opção **halt**

```
1 vagrant halt
```

O comando **vagrant halt** efetua o desligamento da máquinas descritas no Vagrantfile, também é possível especificar o nome ou id da máquina após o parâmetro **halt** para desligar uma máquina específica.

Para deletar uma instância utilize a opção **destroy**

```
1 vagrant destroy
```

**ATENÇÃO:** A opção **destroy** é irreversível, tudo relacionado aquele ambiente que você criou será excluído, com exceção do conteúdo que está dentro do diretório do seu Vagrantfile

## Criando Múltiplas Instâncias

Quando trabalhamos com Vagrant, não criamos um arquivo por instância e sim descrevemos um ambiente todo dentro dele, assim iremos adicionar múltiplas instâncias no nosso Vagrantfile

Crie uma pasta chamada infraagil para armazenar os arquivos e acesse a pasta

```
1 mkdir infraagil  
2 cd infraagil
```

Crie o arquivo Vagrantfile para definir duas instâncias.

```
1 vim Vagrantfile
```

```
1 Vagrant.configure("2") do |config|  
2   config.vm.define "server1" do |server1|  
3     server1.vm.box = "centos/7"  
4   end  
5   config.vm.define "server2" do |server2|  
6     server2.vm.box = "ubuntu/bionic64"  
7   end  
8 end
```

Construa o ambiente de acordo com o Vagrantfile:

```
1 vagrant up
```

Liste as máquinas:

```
1 vagrant status
```

Podemos conectar as instâncias passando como parametro do comando vagrant ssh o nome que foi definido no campo *config.vm.define*:

Conecte-se a máquina server1 e verifique a release do sistema operacional.

```
1 vagrant ssh server1  
2 cat /etc/*-release
```

Desconecte da máquina server1 e repita o procedimento para a máquina server2:

```
1 exit  
2 vagrant ssh server2  
3 cat /etc/*-release
```

Desconecte da máquina server2 e destrua o ambiente

```
1 exit  
2 vagrant destroy
```

## Configurando interfaces de Rede

É possível definir um endereço IP e configurações de interfaces de rede para cada instância no Vagrantfile através do parâmetro **vm.network**

Vamos editar o Vagrantfile para incluir as linhas de configuração de rede em cada instância

```
1 vim Vagrantfile
```

```
1 Vagrant.configure("2") do |config|
2   config.vm.define "server1" do |server1|
3     server1.vm.box = "centos/7"
4     server1.vm.network "private_network", ip: "10.5.25.10"
5   end
6   config.vm.define "server2" do |server2|
7     server2.vm.box = "ubuntu/bionic64"
8     server2.vm.network "private_network", ip: "10.5.25.20"
9   end
10 end
```

Através do parâmetro **vm.network** também é possível configurar encaminhamento de portas, redes privadas e públicas, IP estático ou DHCP entre várias outras funcionalidades

Construa o ambiente de acordo com o Vagrantfile:

```
1 vagrant up
```

Execute o comando nas máquinas e verifique o endereço de rede

```
1 vagrant ssh server1 -c "ip -c a show eth1"
2 vagrant ssh server2 -c "ip -c a show enp0s8"
```

A opção **-c** no comando `vagrant ssh` faz com que seja executado diretamente e o retorno seja exibido na tela

Destrua o ambiente

```
1 vagrant destroy -f
```

A opção **-f** no comando `vagrant destroy` força a destruição do ambiente sem a solicitação de confirmação do usuário

## Variáveis, Loops e Condições no Vagrant

### Trabalhando com Variáveis

Podemos utilizar variáveis no nosso Vagrantfile para parametrizar nosso ambiente, isso faz com que o código possa ser reaproveitado para outros ambientes, sendo possível modificar apenas algumas linhas e ter uma infraestrutura completamente nova criada.

Vamos editar nosso Vagrantfile e adicionar algumas variáveis:

```
1 vim Vagrantfile
```

Para utilização da variável, basta efetuar a declaração no topo do Vagrantfile e inseri-la no código.

```

1 NETWORK = "10.5.25."
2 BOX_SRV1 = "centos/7"
3 BOX_SRV2 = "ubuntu/bionic64"
4
5 Vagrant.configure("2") do |config|
6   config.vm.define "server1" do |server1|
7     server1.vm.box = BOX_SRV1
8     server1.vm.network "private_network", ip: NETWORK+"10"
9   end
10  config.vm.define "server2" do |server2|
11    server2.vm.box = BOX_SRV2
12    server2.vm.network "private_network", ip: NETWORK+"20"
13  end
14 end

```

No caso do IP de rede, utilizamos a concatenação da variável adicionando a variável o sinal de soma seguido da terminação do IP de host entre aspas.

**Exemplo:** NETWORK+"20" = "10.5.25.20", uma vez que NETWORK = 10.5.25.

Construa novamente o ambiente:

```
1 vagrant up
```

Execute o comando nas máquinas para validar as configurações

```

1 vagrant ssh server1 -c "cat /etc/*release"
2 vagrant ssh server1 -c "ip -c a show eth1"
3 vagrant ssh server2 -c "cat /etc/*release"
4 vagrant ssh server2 -c "ip -c a show enp0s8"

```

Destrua o ambiente

```
1 vagrant destroy -f
```

## Trabalhando com Laços de Repetição

Quando trabalhamos com Vagrant, não criamos um arquivo por instância, e sim descrevemos um ambiente todo dentro dele. Para esse fim, utilizamos laços de repetição e especificaremos todas as configurações para o provisionamento do nosso ambiente.

Para definir uma estrutura de laços de repetição podemos definir as informações e configurações de máquinas no topo do nosso arquivo Vagrantfile em uma variável que irá conter um dicionário que iremos chamar de **machines**

```

1 machines = {
2   "compliance" => { "image" => "ubuntu/bionic64"}, 
3   "container"   => { "image" => "centos/7"}, 
4   "scm"         => { "image" => "debian/buster64"}, 
5   "log"          => { "image" => "ubuntu/bionic64"}, 
6   "automation"  => { "image" => "centos/7"} 
7 }

```

Para acessar as variáveis em um array, é necessário que ela esteja no formato **#{VARIÁVEL}**, onde a hash identifica que trata-se de um array e as chaves delimitam o campo da variável.

```

1 Vagrant.configure("2") do |config|
2   machines.each do |name,conf|
3     config.vm.define "#{name}" do |machine|
4       machine.vm.box = "#{conf["image"]}"

```

```

5     machine.vm.hostname = "#{name}.4labs.example"
6   end
7 end
8 end

```

A definição das máquinas é feito na linha **machines.each do |name,conf|**, onde é realizada a leitura do bloco machines, sendo o primeiro campo informado como nome e os demais campos são formados com o array chave e valor, sendo chamado como **#{conf[valor]}**.

O laço de repetição ocorre quando acessamos o método each, passando o conteúdo de cada iteração para a variável **machine**. Basicamente, a cada repetição um item de sua lista será salvo dentro da variável machine, e então um trecho de código será repetido. No nosso caso, todas as instruções de configuração de servidor ficarão dentro do laço, assim iremos criar nossas instâncias baseadas nesse conteúdo.

Por se tratar de um array de chave e valor, o conteúdo da variável machine para termos acesso ao valor, precisamos informar uma chave, no caso informamos image como nossa chave para que então tenhamos acesso ao valor “**centos**”, “**ubuntu**”, etc...

O hostname de cada máquina também é configurado através do parametro **machine.vm.hostname**, onde o ele concatena a chave **name** com **.4labs.example** para configurar o hostname.

Vamos editar nosso Vagrantfile e adicionar o nosso bloco de machines e as declarações das variáveis:

```
1 vim Vagrantfile
```

```

1 # -*- mode: ruby -*-
2 # vi: set ft=ruby :
3
4 machines = {
5     "compliance" => {"memory" => "2048", "cpu" => "2", "ip" => "20", "image" => "ubuntu/
6         bionic64"}, 
7     "container"    => {"memory" => "2048", "cpu" => "1", "ip" => "30", "image" => "centos/7"
8 }, 
9     "scm"          => {"memory" => "256", "cpu" => "1", "ip" => "40", "image" => "debian/
10        buster64"}, 
11    "log"           => {"memory" => "1024", "cpu" => "1", "ip" => "50", "image" => "ubuntu/
12        bionic64"}, 
13    "automation"   => {"memory" => "3072", "cpu" => "2", "ip" => "10", "image" => "centos/7"
14 }
15 Vagrant.configure("2") do |config|
16   config.vm.box_check_update = false
17   machines.each do |name, conf|
18     config.vm.define "#{name}" do |machine|
19       machine.vm.box = "#{conf["image"]}"
20       machine.vm.hostname = "#{name}.4labs.example"
21       machine.vm.network "private_network", ip: "10.5.25.#{conf["ip"]}"
22       machine.vm.provider "virtualbox" do |vb|
23         vb.name = "#{name}"
24         vb.memory = conf["memory"]
25         vb.cpus = conf["cpu"]
26         vb.customize ["modifyvm", :id, "--groups", "/525-InfraAgil"]
27       end
28     end
29   end
30 end

```

Construa novamente o ambiente:

```
1 vagrant up
```

Verifique os status das máquinas

```
1 vagrant status
```

Destrua o ambiente

```
1 vagrant destroy -f
```

## Trabalhando com Condições

Como o Vagrantfile é escrito em ruby, também é possível que sejam configuradas condições utilizando um bloco de **IF** para realizar um teste e executar ações de acordo com o resultado do teste.

Vamos adicionar uma condição no loop do nosso Vagrantfile, quando a máquina a ser criada for a automation, iremos adicionar mais um disco de 20GiB, primeiramente efetuando um teste para verificar se um disco existe e em seguida criando o disco.

```
1 if name == "automation" and not File.file?('iscsi525.vdi')
2   vb.customize ['createhd', '--filename', 'iscsi525.vdi', '--size', 20 * 1024]
3   vb.customize ['storagectl', :id, '--name', 'SATA Controller', '--add', 'sata']
4   vb.customize ['storageattach', :id, '--storagectl', 'SATA Controller', '--port', 1, '--device', 0,
5     '--type', 'hdd', '--medium', 'iscsi525.vdi']
6 end
```

A declaração **IF** verificar se o arquivo iscsi525.vdi existe, se o mesmo não existir o arquivo é criado, em seguida é adicionada uma controladora SATA a máquina e o disco é anexado a mesma. Este tipo de declaração é necessária, uma vez que se o ambiente for provisionado diversas vezes o arquivo poderia ser reescrito apagando seu conteúdo.

Vamos editar nosso Vagrantfile e adicionar o nosso bloco de condições:

```
1 vim Vagrantfile
```

```
1 # -*- mode: ruby -*-
2 # vi: set ft=ruby :
3
4 machines = {
5   "compliance" => {"memory" => "2048", "cpu" => "2", "ip" => "20", "image" => "ubuntu/bionic64"},
6   "container" => {"memory" => "2048", "cpu" => "1", "ip" => "30", "image" => "centos/7"},
7   "scm" => {"memory" => "256", "cpu" => "1", "ip" => "40", "image" => "debian/buster64"},
8   "log" => {"memory" => "1024", "cpu" => "1", "ip" => "50", "image" => "ubuntu/bionic64"},
9   "automation" => {"memory" => "3072", "cpu" => "2", "ip" => "10", "image" => "centos/7"}
10 }
11 Vagrant.configure("2") do |config|
12   config.vm.check_update = false
13   machines.each do |name, conf|
14     config.vm.define "#{name}" do |machine|
15       machine.vm.box = "#{conf["image"]}"
16       machine.vm.hostname = "#{name}.4labs.example"
17       machine.vm.network "private_network", ip: "10.5.25.#{conf["ip"]}"
18       machine.vm.provider "virtualbox" do |vb|
19         vb.name = "#{name}"
20         vb.memory = conf["memory"]
21         vb.cpus = conf["cpu"]
22         vb.customize ["modifyvm", :id, "--groups", "/525-InfraAgil"]
23         if name == "automation" and not File.file?('iscsi525.vdi')
24           vb.customize ['createhd', '--filename', 'iscsi525.vdi', '--size', 20 * 1024]
25           vb.customize ['storagectl', :id, '--name', 'SATA Controller', '--add', 'sata']
```

```
26     vb.customize ['storageattach', :id, '--storagectl', 'SATA Controller', '--port', 1, '--
27     device', 0, '--type', 'hdd', '--medium', 'iscsi525.vdi']
28   end
29 end
30 end
31 end
```

Construa novamente o ambiente:

```
1 vagrant up
```

Verifique se o arquivo iscsi525.vdi foi criado e anexado a máquina automation

```
1 file iscsi525.vdi
2 vagrant ssh automation -c "lsblk"
```

*Deverá ser exibido um disco sdb com o tamanho de 20G*

```
1 NAME      MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
2 sda        8:0    0 40G  0 disk 
3 sda1       8:1    0 40G  0 part /
4 sdb        8:16   0 20G  0 disk
```

Destrua o ambiente

```
1 vagrant destroy -f
```

## Provisionadores

### O que são Provisionadores

Provisionadores no Vagrant possibilitam a instalação automática de software, alteração de configuração e arquivos na máquina como parte do processo do vagrant up

### Tipos de Provisionadores

Existem vários tipos de provisionadores para o Vagrant, dentre eles podemos citar: \* File \* Shell \* Ansible \* CFEngine \* Chef \* Puppet \* Docker \* Salt

### Utilizando Provisionadores

Para a utilização dos provisionadores, efetuamos a chamada do mesmo através do parâmetro **vm.provision**, seguido do nome do provisionador

```
1 Vagrant.configure("2") do |config|
2   config.vm.box = "centos/7"
3   config.vm.provision "shell",
4     (...),
5   end
6 end
```

Para utilização do provisionador **file**, podemos utilizar como parâmetros a origem e o destino dos arquivos:

```

1 Vagrant.configure("2") do |config|
2   config.vm.box = "centos/7"
3   config.vm.provision "file",
4     source: "~/files/hosts",
5     destination: "/etc/hosts"
6   end
7 end

```

Através do provisionador `file`, devemos informar a origem na máquina hospedeira e o destino na máquina virtual criada pelo vagrant  
Para utilização do provisionador `shell`, podemos informar qual será o comando executado através do parâmetro **inline**

```

1 Vagrant.configure("2") do |config|
2   config.vm.box = "centos/7"
3   config.vm.provision "shell",
4     inline: "mkdir -p /srv/app"
5   end
6 end

```

Através do parâmetro `inline`, podemos adicionar os comandos diretamente no `Vagrantfile`, como por exemplo executar a criação da estrutura de diretórios `/srv/app`

Também podemos declarar um script no topo do `Vagrantfile` e referenciar via **inline**:

```

1 $script = <<-SCRIPT
2 mkdir -p /srv/app
3 touch /srv/app/file1
4 SCRIPT
5
6 Vagrant.configure("2") do |config|
7   config.vm.box = "centos/7"
8   config.vm.provision "shell", inline: $script
9 end
10 end

```

A definição do script no topo do `Vagrantfile` faz com que a manutenção e edição do arquivo seja feita de forma rápida, deixando o arquivo mais organizado. Como em um script em shell, as tarefas são executadas linha a linha

Outra maneira de chamar um script é através do parâmetro **path**, isto fará com que seja executado o script que se encontra na máquina local

```

1 Vagrant.configure("2") do |config|
2   config.vm.box = "centos/7"
3   config.vm.provision "shell",
4     path: "script.sh"
5   end
6 end

```

O parâmetro **path** aceita também o script através de uma URL

```

1 Vagrant.configure("2") do |config|
2   config.vm.box = "centos/7"
3   config.vm.provision "shell",
4     path: "https://www.4labs.example/provision.sh"
5   end
6 end

```

Podemos também descrever nossa automação utilizando Ansible e Puppet que são assuntos deste curso, para tal precisamos primeiramente conhecer a ferramenta. Mais informações sobre estes módulos podem ser encontrados em Vagrant Provisioning

## Construindo o Ambiente

Vamos começar a construir nosso ambiente. Nas próximas aulas iremos utilizar 5 máquinas (compliance, container, scm, log e automation). Para isso, iremos alterar o Vagrantfile para descrever todo o nosso ambiente, como CPU, Memória, Imagem, Hostname, Rede dentre outros.

Para facilitar a criação destes arquivos iremos clonar o repositório do curso no github da 4linux

Acesse a home do usuário, faça o clone do repositório, acesse o diretório e inicie o ambiente

```
1 cd ~
2 git clone https://github.com/4linux/4525.git
3 cd 4525
4 vagrant up
```

Quando existem multiplas instâncias do Vagrant iniciadas, será necessário informar o nome ou id da máquina para que seja possível acessa-las via ssh. Para isto executaremos o comando vagrant ssh seguido do identificador (nome ou id) da máquina.

Verifique a conexão com as máquinas

```
1 vagrant ssh <MAQUINA>
```

## Configuração do ambiente

Estrutura de Arquivos

```
1 4525/
2   cybertux.png
3   files
4     gogs.service
5     key
6     key.pub
7     ntp.conf
8     rundeck-rocketchat-notifier-0.6.jar
9   README.md
10  script.sh
11  Vagrantfile
```

No arquivo Vagrantfile foram criadas diversas variáveis para descrever os aspectos de cada uma dessas instâncias (rede, imagem, cpu, memória e hostname).

```
1 # -*- mode: ruby -*-
2 # vi: set ft=ruby :
3
4 machines = {
5   "automation" => {"memory" => "1536", "cpu" => "2", "ip" => "10", "image" => "centos/7"},
6   "compliance" => {"memory" => "1024", "cpu" => "2", "ip" => "20", "image" => "ubuntu/bionic64"},
7   "container"  => {"memory" => "1536", "cpu" => "1", "ip" => "30", "image" => "centos/7"},
8   "scm"        => {"memory" => "256",  "cpu" => "1", "ip" => "40", "image" => "debian/buster64"},
9   "log"        => {"memory" => "2048", "cpu" => "1", "ip" => "50", "image" => "ubuntu/bionic64"}
10 }
11
12 Vagrant.configure("2") do |config|
13
14   config.vm.box_check_update = false
15   config.vm.boot_timeout = 600
16   machines.each do |name, conf|
17     config.vm.define "#{name}" do |machine|
18       machine.vm.box = "#{conf["image"]}"
19       machine.vm.hostname = "#{name}.4labs.example"
20       machine.vm.network "private_network", ip: "10.5.25.#{conf["ip"]}"
```

```

21     machine.vm.provider "virtualbox" do |vb|
22       vb.name = "#{name}"
23       vb.memory = conf["memory"]
24       vb.cpus = conf["cpu"]
25       vb.customize ["modifyvm", :id, "--groups", "/525-InfraAgil"]
26   end
27   if "#{conf["image"]}" == "ubuntu/bionic64" or "#{conf["image"]}" == "debian/buster64"
28     machine.vm.provision "shell", inline: "apt-get update ; apt-get install python -y;
29     hostnamectl set-hostname #{name}.4labs.example"
30   end
31 end
32 config.vm.provision "shell", path: "script.sh"
33 end

```

O arquivo **script.sh** será responsável pela garantia das chaves de acesso, por popular o arquivo hosts e pela criação de um swapfile de 1G

```

1 # Garantindo as chaves
2 KEY_PATH='/vagrant/files'
3 mkdir -p /root/.ssh
4 cp $KEY_PATH/key /root/.ssh/id_rsa
5 cp $KEY_PATH/key.pub /root/.ssh/id_rsa.pub
6 cp $KEY_PATH/key.pub /root/.ssh/authorized_keys
7 chmod 400 /root/.ssh/id_rsa*
8 cat /root/.ssh/id_rsa.pub >> /home/vagrant/.ssh/authorized_keys
9
10 # Garantindo os hosts
11 HOSTS=$(head -n7 /etc/hosts)
12 echo -e "$HOSTS" > /etc/hosts
13 echo '10.5.25.10 automation.4labs.example' >> /etc/hosts
14 echo '10.5.25.20 compliance.4labs.example' >> /etc/hosts
15 echo '10.5.25.30 container.4labs.example chat.4labs.example' >> /etc/hosts
16 echo '10.5.25.40 scm.4labs.example' >> /etc/hosts
17 echo '10.5.25.50 log.4labs.example' >> /etc/hosts
18
19 # Criando arquivo de SWAP
20 fallocate -l 1G /swapfile
21 chmod 600 /swapfile
22 mkswap /swapfile
23 echo -e "/swapfile swap swap defaults 0 0" >> /etc/fstab
24 swapon -a

```

Na pasta **files**, existem arquivos de parametrização que iremos utilizar nas próximas aulas e também um par de chaves SSH para a utilização nos laboratórios.

# Capítulo 3

## Versionamento de Código com GIT

### Versionamento de Código Local

#### O que é Versionamento de Código

Versionamento de código é o processo de se criar versões de códigos de acordo com uma linha do tempo para o andamento dos desenvolvimento de melhorias, correções ou novas funcionalidades.

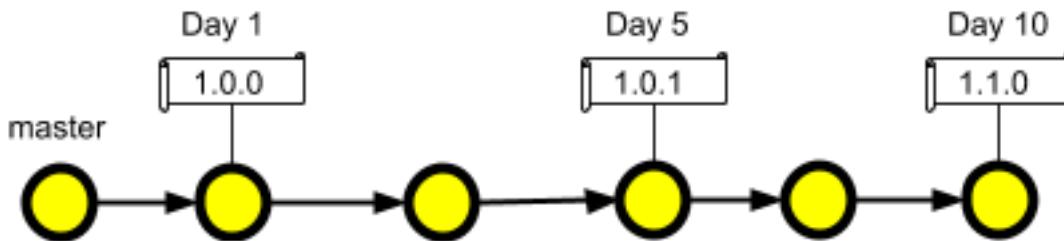


Figura 3.1: Timeline

Para o versionamento de código são utilizados sistemas chamados de **VCS** (Version Control System), que registram as mudanças de um conjunto de arquivos baseado em uma tratativa de dados, e o **SCM** (Source Code Management), que são softwares que têm a finalidade de gerenciar as diferentes versões no desenvolvimento de um código. Alguns exemplos de softwares de controle de versão Open-Source são: \* CVS \* Mercurial \* GIT \* SVN

#### O que é o GIT

O GIT é um VCS amplamente utilizado nos dias de hoje para o processo de desenvolvimento de software e versionamento de documentos. Foi projetado e desenvolvido por *Linus Torvalds* para o desenvolvimento do Kernel Linux após o Bitkeeper, VCS proprietário da época, remover o acesso gratuito. Hoje se tornou a maior ferramenta OpenSource de versionamento do mercado.

O GIT trabalha com dados como se fossem grupos de **snapshots** de um pequeno sistema de arquivo.

#### Entendendo o GIT

Toda vez que um usuário realiza um commit ou salva o estado atual do projeto, o GIT tira uma imagem do estado de todos os arquivos e grava uma referência do estado (**snapshot**). Se o arquivo não possui nenhuma alteração, o GIT não grava os arquivos novamente, ele simplesmente cria um link para o arquivo anterior. O GIT pensa como se os arquivos e suas alterações fossem um fluxo de snapshots.



Figura 3.2: GIT

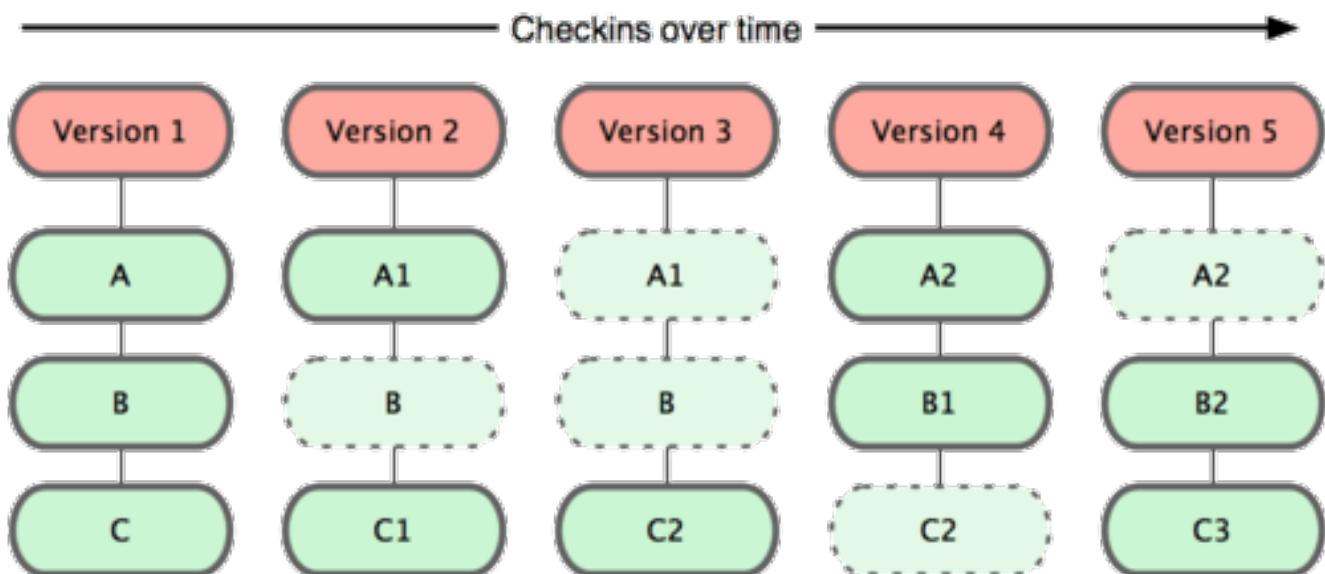


Figura 3.3: snapshot

É possível comparar o sistema do GIT com um backup FULL, já que cada snapshot é uma imagem completa e não apenas a diferença entre versões.

## Outros VCS's

Outros sistemas costumam armazenar dados, como mudanças em uma versão inicial de cada arquivo.

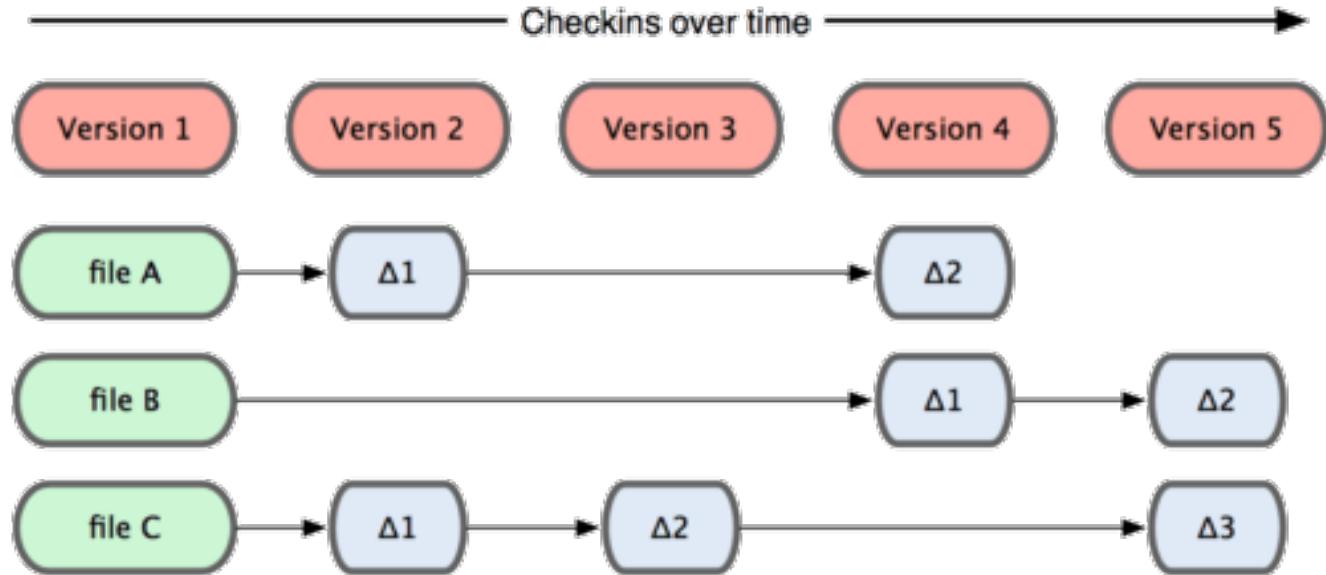


Figura 3.4: Outros VCS

Como vimos, o GIT, em relação a outros VCS, faz uma tratativa um tanto diferenciada dos dados. Em suma, os outros VCS armazenam uma índice de mudanças para cada arquivo baseado no tempo, considerando que cada versão do arquivo se torna um apontamento para suas diferenças de conteúdo.

Para um entendimento mais simples, podemos comparar essa forma de tratativa de dados com um *backup incremental*, devido a seu tratamento nas diferenças do arquivo e não no arquivo completo.

## Os Três Estados

Diretório de Trabalho, área de preparação e o diretório do GIT.

Essa é uma parte das mais importantes para entender o funcionamento do GIT. O GIT faz com que seus arquivos sempre esteja em um dos três estados fundamentais:

- \* Modificado (Modified): Trata-se de um arquivo que sofreu mudanças mas que ainda não está pronto para que as mudanças sejam aplicadas;
- \* Preparado (Staged): Após realizar as mudanças a segunda etapa é deixar o arquivo pronto para ser versionado. Com isso, é feita uma marcação no arquivo que define as mudanças como sendo a sua versão mais recente e preparando o arquivo para o próximo commit;
- \* Consolidado (Committed): Após o preparo dos dados modificados é realizada a sua consolidação, dessa forma os dados estarão armazenados na base de dados do Git de forma segura e em formato de snapshot.

Isso nos traz para as três seções do GIT:

- \* Diretório do GIT (Git Directory, Repository): Local onde o GIT faz referência e armazena os metadados e o banco de objetos de seu projeto. É a parte copiada quando você clona um repositório.
- \* Diretório de Trabalho (Working Directory): Representa as modificações de uma das muitas versões de um projeto. Os arquivos são adquiridos da base de dados do diretório *.git*, dessa forma é possível fazer que fiquem alocados em disco e possam ser manipulados para gerar uma nova versão.
- \* Área de Preparação (Staging area): É um referência de um arquivo ou objeto do diretório de trabalho que contém as informações de modificações que estão prontas para o próximo commit. É bastante conhecido como índice(index), mas está se tornando padrão chamá-lo apenas de "Staging Area"

O workflow básico do GIT pode ser descrito assim:

1. Você modifica arquivos no seu diretório de trabalho;
2. Você seleciona os arquivos, adicionando snapshots deles a sua área de preparação;
3. Você faz um commit, que leva os arquivos como eles estão na sua área de preparação e os armazena permanentemente no seu diretório GIT.

# Local Operations

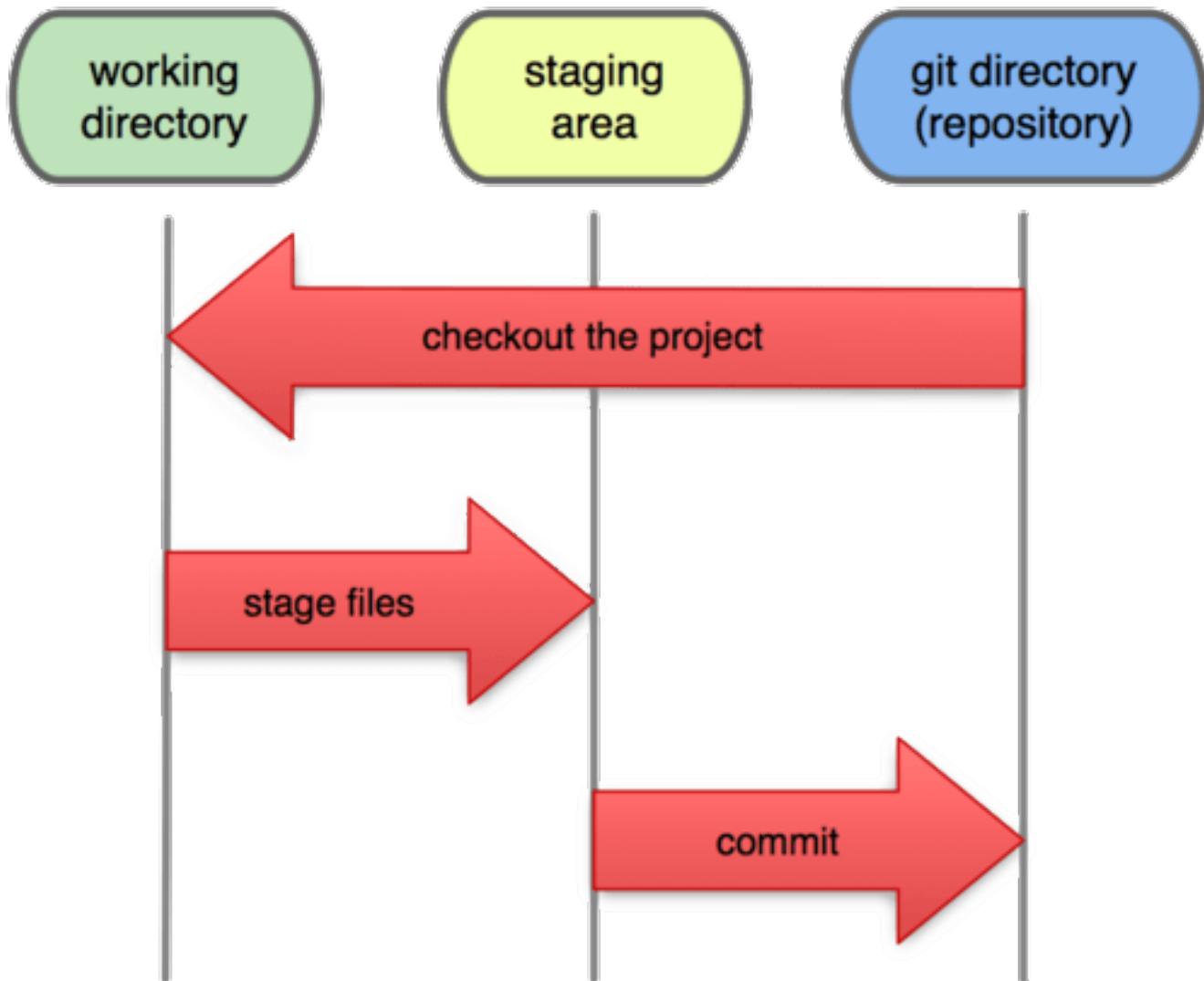


Figura 3.5: Working Directory

## Instalação do GIT

Instalação do GIT em distribuições baseadas em Debian/Ubuntu

```
1 sudo apt-get update  
2 sudo apt-get install git
```

Instalação do GIT em distribuições baseadas em Red Hat

```
1 sudo yum install git
```

Verifique a instalação do GIT através do comando:

```
1 git --version
```

## Comandos essenciais do GIT

### Iniciando um repositório.

Os comandos essenciais serão digitados na máquina automation. Execute os comandos abaixo para preparar a infraestrutura do curso.

```
1 cd 4525  
2 vagrant up  
3 vagrant ssh automation  
4 sudo yum install git -y
```

Qualquer repositório pode se tornar um repositório do GIT, para isto basta executar o comando git init

Crie um novo diretório e accesse o mesmo

```
1 mkdir projeto  
2 cd projeto
```

Inicie o repositório

```
1 git init
```

Ao digitar esse comando, será criado um diretório oculto chamado **.git** no diretório atual. Esse diretório é a base de dados do GIT, e constitui o repositório em si. Nele serão guardadas as informações de **commits, tags e branches**, ou seja, todas as informações referentes ao repositório.

Os comandos que aprenderemos o git irão manipular o conteúdo desta pasta, por isso não precisamos mexer diretamente nela.

Algumas informações acompanharão esse processo, tais como: e-mail, nome, data, hora e mensagem de identificação. Algumas destas informações são geradas automaticamente, outras deveos informar.

Configure o nome do proprietário

```
1 git config --global user.name "Seu Nome"
```

Configure o e-mail do proprietário

```
1 git config --global user.email "seuemail@seudominio.com.br"
```

Com o comando **git config**, salvamos os dados em variáveis de configuração do GIT, ao enviar os dados para um servidor estas informações também serão enviadas.

O parâmetro **-global** diz ao GIT que salvaremos esses valores no arquivo **~/.gitconfig**, essas configurações serão utilizadas para todos os outros projetos daquele usuário. Existe outro parâmetro (**-system**) que salva as configurações em **/etc/gitconfig** sendo utilizado para projetos de todos os usuários.

A ausência desses parâmetros gravará as informações no **.git/config**, sendo assim, válidas apenas para aquele projeto.

Outras configurações podem ser realizadas. Vamos configurar o editor padrão do GIT, que aparecerá sempre que salvarmos uma nova versão do projeto

Instale o editor vim e altere as configurações do editor padrão do GIT

```
1 sudo yum install vim -y
2 git config --global core.editor vim
```

É possível listar todas as configurações do GIT configuradas através do comando:

```
1 git config --list
```

Agora que temos nossas informações já configuradas, vamos iniciar o versionamento do diretório.

Crie um arquivo qualquer

```
1 touch arquivo1
```

Adicione o arquivo para a área de staging

```
1 git add arquivo1
```

O comando **git add** adiciona o arquivo em uma área chamada **staging**

Visualize o estado atual do repositório

```
1 git status
```

O comando **git status** mostra o estado do repositório no momento atual

Grave a nova versão do repositório:

```
1 git commit -m "Meu primeiro commit"
```

O comando **git commit** salva o estado dos arquivos que estão na área de stage. O parâmetro **-m** atribui uma mensagem ao commit, servindo para identificar facilmente o que foi realizado na alteração.

Para modificar nosso commit, iremos criar outros arquivos

```
1 touch arquivo{2..4}
```

Adicione alguns arquivos a área de staging

```
1 git add arquivo2 arquivo3
```

Faça o commit atual do repositório

```
1 git commit -m "Commit Incompleto"
```

Ao executar o commit, o GIT informará que os arquivos (arquivo2 e arquivo3) passaram pelo commit sendo adicionado ao repositório local  
Adicione o arquivo4 a área de staging

```
1 git add arquivo4
```

Para adicionar o arquivo ao ultimo commit, utilize o parametro **-amend**

```
1 git commit --amend
```

O comando **amend** deve ser utilizado quando for necessário adicionar novos arquivos a um commit já realizado  
Com isso, é possível checar os registros de commits realizados utilizando o seguinte comando:

```
1 git log
```

Possibilitando a visualização de informações do commit como seu ID, Autor, Data, e Comentário.

## Clonando um repositório

Através do GIT podemos clonar um repositório que já existe na web:

Acesse a home do usuário e clone um repositório

```
1 cd ~  
2 git clone https://github.com/leachim6/hello-world.git
```

Também é possível definir o nome do diretório criado através de um **git clone**

```
1 git clone https://github.com/leachim6/hello-world.git ola-mundo
```

Verifique que existem duas pastas com nomes diferentes, mas com o mesmo conteúdo

```
1 ls hello-world  
2 ls ola-mundo
```

## Trabalhando com Branches em projetos

### O que são branches

Realizando uma associação com árvores, branches(ramos) são linhas que divergem do tronco principal de desenvolvimento.  
Em novos branches, realizamos o desenvolvimento de componentes que serão combinados, posteriormente, com o projeto principal

Criar uma branch significa: > "...divergir da linha principal de desenvolvimento e continuar a trabalhar sem bagunçar essa linha principal." fonte: Ramificação e Branching no GIT

Normalmente em ambientes de integração contínua existem três branches para os projetos

- **Master** -> Versão atual, pronta para produção.
- **Homolog** -> Versão de homologação, código sendo testado
- **Develop** -> Branch onde são feitos os commits

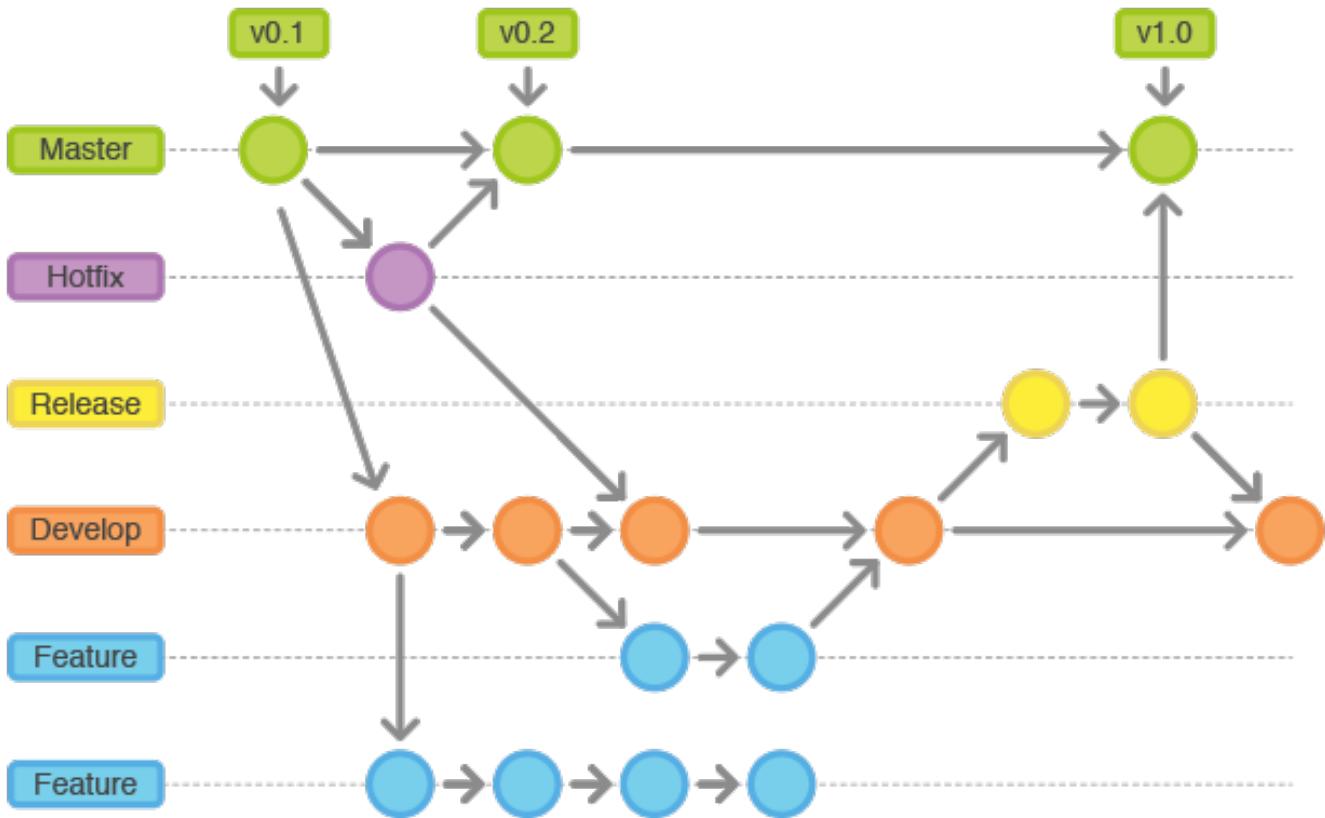


Figura 3.6: Branches

## Criando, Listando e Trocando de Branches

Tudo no GIT é feito através de branches. A branch principal é a master, automaticamente criada sempre que um projeto git é clonado ou iniciado a partir de um diretório existente.

Acesse o diretório e liste as branches disponíveis

```
1 cd projeto  
2 git branch
```

Crie uma nova branch e liste as branches

```
1 git branch development  
2 git branch
```

O comando `git branch` é utilizado para criação de uma nova branch, é possível notar que a branch foi criada porém não foi feita a mudança para a nova branch

## Trocando de branches

```
1 git checkout development  
2 git checkout master
```

## Criando e trocando de branch

```
1 git checkout -b homolog
```

O comando `git checkout -b` faz a criação de uma nova branch e automaticamente acessa a branch criada.

Não existem limites para essas ramificações. A utilização de branches é útil tanto para o processo de desenvolvimento alinhado a ambientes, quanto no desenvolvimento de novas features sem modificar a branch atual.

Além de criar e modificar as branches, podemos excluí-las

Primeiramente, troque para a branch master

```
1 git checkout master
```

Excluir uma branch

```
1 git branch -d development
```

É necessário que o `merge` da branch tenha sido realizado antes da remoção

Forçando a remoção de uma branch

```
1 git branch -D homolog
```

Com a opção `-D` a branch é removida mesmo que o `merge` não tenha sido realizado

## Merge

**Merge** é o processo realizado quando terminamos de desenvolver uma feature e precisamos que nosso código saia de uma branch para a outra, como por exemplo de desenvolvimento para homologação.

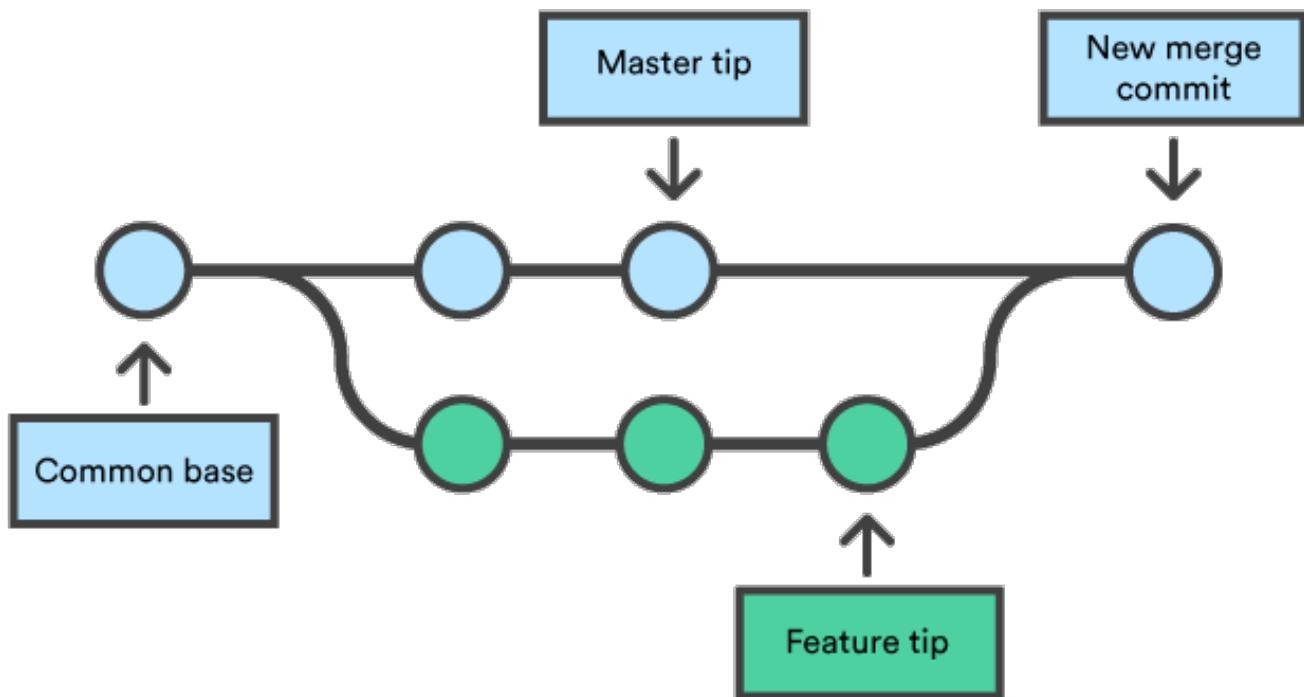


Figura 3.7: Merge

Crie uma branch para simular um merge

```
1 git checkout -b develop
```

Sempre que criamos uma nova branch, ela é iniciada como uma cópia da branch atual.

Crie um novo arquivo e adicione a um commit

```
1 touch arquivo6  
2 git add arquivo6  
3 git commit -m "commit arquivo6"
```

Tudo que for feito nessa nova branch e passar por um commit produzirá uma diferença em relação a branch copiada.

Troque para a branch master e efetue o merge da develop para master

```
1 git checkout master  
2 git merge develop
```

O comando **git merge** traz os commits a frente e os aplica na branch atual.

## Ignorando arquivos no versionamento

### Porquê ignorar arquivos?

Normalmente no processo de desenvolvimento de software existem alguns arquivos que não gostaríamos que fossem versionados como por exemplo, arquivos compilados, arquivos com senha ou arquivos temporários que são criados por editores.

### .gitignore

Com o GIT o controle destes arquivos fica muito mais fácil, podemos criar um arquivo chamado **.gitignore** e passar algumas opções e assim conseguiremos dizer quais arquivos por projeto ou usuário não devem ser publicados.

A maneira mais simples e fácil de gerenciar as exclusões é criar um arquivo **.gitignore** no diretório raiz do projeto.

Os arquivos que escolheremos para ignorar serão ignorados em todos os diretórios do projeto.

**ATENÇÃO:** É importante lembrar que o arquivo é um arquivo oculto, por isto seu nome começa com o caractere ponto (.)

O Arquivo **.gitignore** consegue compreender alguns tipos de declarações, tais como: \* Arquivos específicos -> **arquivo1.txt** \* Curingas -> \*\*\_\*.txt \_ Diretórios -> \_/files/keys/ \* **Expressões Regulares (Regex)** -> \_arquivo{1..5}.txt\*\*

O **gitignore** especifica em estado untracked que o Git deve ignorar. Qualquer arquivo que se apresente em estado tracked continuará como tal, precisando ser removido manualmente.

Sintaxe do arquivo **gitignore**:

- “!” — Deve ser utilizado para negar um padrão, tornando-o valido novamente. Não valendo para diretórios.
- “**foo/**” — Caso o padrão termine em “/” o git irá considerar somente o diretório foo.
- “**foo**” — Nesse caso será tratado como um padrão a ser verificado no level mais alto do diretório onde o **.gitignore** se encontra.
- “\*\*/” — Casa qualquer diretório. Por exemplo: se especificado **\*/foo** será válido para **foo/** e **\*/foo/bar**, sendo válido para qualquer diretório “bar” antecedido por “foo”.
- “**/\*\***” — Casa todo o conteúdo dentro do diretório.
- “**a/\*\*/b**” — Casa qualquer conteúdo entre os diretório existentes entre a e b.

## Utilizando o .gitignore

Agora iremos criar nosso arquivo **.gitignore** com alguns padrões

Crie o arquivo .gitignore com o conteúdo config.conf

```
1 echo "config.conf" > .gitignore
```

Neste exemplo, qualquer arquivo com o nome de config.conf em qualquer diretório será ignorado no commit

Adicione o arquivo ao repositório e efetue o commit

```
1 git add .gitignore
2 git commit -m "Adicionado gitignore"
```

Crie alguns arquivos e adicione todos os arquivos ao commit

```
1 touch config.conf conf.txt
2 git add --all
```

Verifique que o arquivo config.conf foi ignorado do commit

```
1 git status
```

Efetue o commit

```
1 git commit -m "Testando o gitignore"
```

Vamos editar nosso arquivo gitignore para adicionar outros exemplos

```
1 vim .gitignore
```

```
1 #Ignorar os arquivos específicos
2 config.conf
3
4 #Ignorar arquivos por coringa
5 */
6 *.swap
7
8 #Ignorar Diretórios
9 files/secret
```

Após a edição do arquivo precisamos efetuar o commit do mesmo

```
1 git add .gitignore
2 git commit -m "Atualizando o gitignore"
```

É importante que o arquivo gitignore seja enviado antes dos arquivos, uma vez que o arquivo gitignore válido é o que já faz parte do commit

Vamos criar os arquivos e pastas que iremos excluir do commit

```
1 mkdir -p files/secret
2 touch file~ file.swap arquivoX files/secret{1..5}
```

Adicione os arquivos ao repositório e verifique os status

```
1 git add .
2 git status
3 git commit -m "Atualizando o repositório"
```

Note que apenas o **arquivoX** foi enviado para o commit, uma vez que é o único que não se enquadra no conteúdo do `.gitignore`, é possível verificar quais arquivos serão enviados através do comando `git status`

## Capítulo 4

# Repositório Remoto com Gogs

### Introdução e Instalação do Gogs

#### Gerenciamento de repositório com GIT

Existem diversos serviços para hospedagem de repositórios GIT. Utilizando estes serviços podemos compartilhar os recursos com diversos times que irão contribuir ou desenvolver nestes repositórios.

Os sistemas de gerenciamento GIT mais conhecidos atualmente são:



Figura 4.1: Re却itorios GIT

#### O que é o Gogs

Gogs (GO Git Service) é um software multiplataforma baseado em GIT e escrito em GoLang.

Seu principal ganho em relação a plataformas como Gitlab é a baixa utilização de recursos, o que faz com que seja possível a instalação do Gogs em dispositivos como um Raspberry PI\*

\_ \*Raspberry PI é um microcomputador de pequeno porte com menos de 1GiB de memória RAM\_

#### Funcionalidades do Gogs

- Criar repositórios GIT e gerenciar direitos de acesso;
- Trabalhar de maneira colaborativa
- Autenticação de dois fatores e conexão LDAP
- Gerencia de acesso por branch a repositórios



Figura 4.2: Gogs

- Ferramenta de relatório de bug
- Criar repositórios espelhos

## Instalando o Gogs

Iremos instalar o gogs na máquina virtual **scm**, a qual é um Debian com apenas 256MiB de memória RAM  
Efetue o login na máquina **scm**

```
1 vagrant ssh scm
```

Troque para o usuário root e crie o usuário gogs com sua home

```
1 sudo su -  
2 useradd git -m
```

Primeiramente precisaremos instalar o banco de dados, utilizaremos o sqlite3.

Instale as dependencias e os binários que irão nos auxiliar

```
1 apt update  
2 apt install libsqlite3-0 vim git sqlite3 curl -y
```

Efetue o download do Gogs, extraia o pacote para o diretório **/opt** e modifique o dono da pasta gogs

```
1 curl https://dl.gogs.io/0.11.91/gogs_0.11.91_linux_amd64.tar.gz -o gogs.tar.gz  
2 tar -xf gogs.tar.gz -C /opt  
3 chown git:git -R /opt/gogs
```

Link para a última versão [https://gogs.io/docs/installation/install\\_from\\_binary](https://gogs.io/docs/installation/install_from_binary)

Vamos registrar gogs como um serviço gerenciável pelo systemd

```
1 cp /vagrant/files/gogs.service /lib/systemd/system/gogs.service
```

```

1 [Unit]
2 Description=Gogs
3 After=syslog.target
4 After=network.target
5 After=mariadb.service mysqld.service postgresql.service memcached.service redis.service
6
7 [Service]
8 User=git
9 Group=git
10 Type=simple
11 WorkingDirectory=/opt/gogs
12 ExecStart=/opt/gogs/gogs web
13 Restart=always
14 Environment=USER=git HOME=/home/git
15
16 [Install]
17 WantedBy=multi-user.target

```

Devido a necessidade da abertura da porta 80, temos que permitir que o abertura dessa porta seja feito por um usuário non-root, ou seja, o usuário git, na chamada do serviço do Gogs

Dessa forma, é preciso liberar esse capability com o seguinte comando:

```
1 setcap cap_net_bind_service+ep /opt/gogs/gogs
```

Neste caso a flag `cap_net_bind_service` é o capability de manipulação de portas privilegiadas, ou seja, menores que 1024. Já o `+ep` são os atribuitos para adicionar o capability no arquivo do serviço(/opt/gogs/gogs). Para mais informações [man capabilities](#)

Após a criação do arquivo de serviço do gogs, é necessário recarregar o daemon do systemctl, habilitar e iniciar o gogs.

```

1 systemctl daemon-reload
2 systemctl enable gogs
3 systemctl start gogs

```

Acesse o endereço <http://10.5.25.40:3000> pelo navegador.

## Database Settings

Gogs requires MySQL, PostgreSQL, SQLite3, MSSQL or TiDB.

**Database Type \***

**Path \***

The file path of SQLite3 database.  
Please use absolute path when you start as service.

Altere o campo Database Type para SQLite3

Em Application General Settings, altere os campos conforme a imagem

## Application General Settings

<b>Application Name *</b>	<input type="text" value="Gogs"/>
Put your organization name here huge and loud!	
<b>Repository Root Path *</b>	<input type="text" value="/home/git/gogs-repositories"/>
All Git remote repositories will be saved to this directory.	
<b>Run User *</b>	<input type="text" value="git"/>
The user must have access to Repository Root Path and run Gogs.	
<b>Domain *</b>	<input type="text" value="scm.4labs.example"/>
This affects SSH clone URLs.	
<b>SSH Port *</b>	<input type="text" value="22"/>
Port number which your SSH server is using, leave it empty to disable SSH feature.	
<input type="checkbox"/> <b>Use Builtin SSH Server</b>	
<b>HTTP Port *</b>	<input type="text" value="80"/>
Port number which application will listen on.	
<b>Application URL *</b>	<input type="text" value="http://scm.4labs.example/"/>
This affects HTTP/HTTPS clone URL and somewhere in email.	
<b>Log Path *</b>	<input type="text" value="/opt/gogs/log"/>
Directory to write log files to.	
<input type="checkbox"/> <b>Enable Console Mode</b>	

Figura 4.3: Application General Settings

## Optional Settings

▶ Email Service Settings

▶ Server and Other Services Settings

▼ Admin Account Settings

You do not have to create an admin account right now, user whoever ID=1 will be the admin.

Username

Password

Confirm Password

Admin Email

Em Optional Settings, crie a conta admin para o gogs.

Clique em **Install Gogs**

Após efetuar a instalação do gogs, é necessário reiniciar o serviço uma vez que a porta da aplicação foi modificada.

```
1 systemctl restart gogs
```

Acesse pelo navegador web o endereço <http://scm.4labs.example>, clique em sign in e efetue login com o usuário **analista** senha **devops@4linux**



Home

Explore

Help

Register

**Sign In**

Username or email \*

Password \*

Remember Me

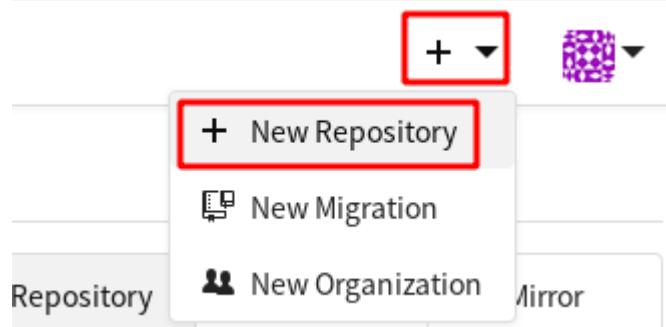
[Forgot password?](#)

[Need an account? Sign up now.](#)

Figura 4.4: Gogs Login

## Gerenciando Repositórios

Para criar um novo repositório, podemos clicar no simbolo + ao lado esquerdo da foto do usuário e em seguida clicar em +

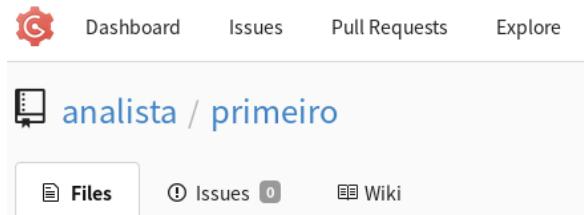


### New Repository

Para criar um repositório, basta preencher os campos com o nome e descrição do repositório e clicar em **Create Repository**

The screenshot shows the 'New Repository' creation form. It includes fields for 'Owner' (set to 'analista'), 'Repository Name' (set to 'primeiro'), 'Visibility' (unchecked 'Private'), 'Description' (set to 'Meu primeiro repositório com Gogs'), and sections for '.gitignore', 'License', and 'Readme'. At the bottom, there is a checkbox for initializing the repository with files and templates, which is unchecked. A large red arrow points to the green 'Create Repository' button at the very bottom of the form.

Será exibida a tela com os dados do repositório, nesta tela é possível verificar informações de como criar o repositório por



## Create a new repository on the command line

```
touch README.md  
git init  
git add README.md  
git commit -m "first commit"  
git remote add origin http://scm.4labs.example/analista/  
git push -u origin master
```

## Push an existing repository from the command line

```
git remote add origin http://scm.4labs.example/analista/  
git push -u origin master
```

uma linha de comando, ou como enviar um repositório já existente para o gogs

## Utilizando Repositórios Remotos

### Gerenciando Repositórios via CLI

É possível gerenciar os repositórios remotos diretamente via interface de linha de comando (**CLI**), para isto, iremos utilizar os comandos do git para adicionar um repositório remoto ao nosso repositório local, editar o mesmo e em seguida efetuar o commit do código.

Conecte-se na máquina scm

```
1 vagrant ssh scm
```

Antes de gerenciar os repositórios, precisamos configurar os parâmetros globais do GIT.

Configure o nome e email do usuário.

```
1 git config --global user.name "Analista DevOps"  
2 git config --global user.email "analista@4labs.example"
```

É possível realizar essa mesma modificação diretamente pelo editor de texto com o comando **git config --global -e**

Com isso, vamos criar um diretório e executar o comando *git init* para inicializar nosso repositório local.

```
1 mkdir primeiro
2 cd primeiro/
3 git init
```

Agora vamos criar alguns arquivos

```
1 touch arquivo{1..5}.txt
2 echo "# Meu primeiro repositório" > README.md
```

Adicione os arquivos para o commit e envie para o repositório local

```
1 git add --all
2 git status
3 git commit -m "Meu primeiro commit no gogs"
```

Após a criação do commit, podemos associar nosso repositório local ao repositório criado no Gogs.

Acesse a página do gogs no endereço <http://scm.4labs.example> e efetue o login com o usuário **analista** e senha



Sign In

Username or email \*

Password \*

Remember Me

[Sign In](#) [Forgot password?](#)

Need an account? [Sign up now.](#)

devops@4linux

Repository Organization Mirror

My Repositories 1

 [primeiro](#) 0 ★

Collaborative Repositories

Clique no repositório criado anteriormente para acessá-lo

The screenshot shows a GitHub repository interface. At the top, there's a navigation bar with 'analista / primeiro'. Below it, tabs for 'Files', 'Issues (0)', and 'Wiki' are visible. A 'Quick Guide' section is present with the heading 'Clone this repository'. It provides three cloning methods: 'HTTP' (which is highlighted with a red box), 'SSH', and the full URL 'http://scm.4labs.example/analista/primeiro.git'.

Iremos copiar sua URL, clique em **HTTP** e copie o endereço do repositório

Dessa forma, vamos voltar ao terminal da *máquina scm* e associar essa URL ao nosso repositório local

```
1 git remote add origin http://scm.4labs.example/analista/primeiro.git
```

Neste caso, estamos adicionando um nome ao nosso repositório remoto, **origin**, e apontando esse nome para a sua respectiva URL. > É comum que o nome repositório seja **origin**, mas isso não é uma regra, logo, é possível adicionar qualquer nome de sua preferência.

Podemos, então, listar os repositórios remotos da seguinte forma

```
1 git remote -v
```

Agora podemos enviar as alterações para o servidor gogs, faremos isto através do comando **git push**.

Execute o **push** e informe o usuário **analista** e senha **devops@4linux**

```
1 git push origin master
2
3 Username for 'http://scm.4labs.example': analista
4 Password for 'http://analista@scm.4labs.example': devops@4linux
```

O comando **git push** envia todos os commits de uma branch específica que estão a frente do repositório remoto

## Gerenciando repositórios remotos via web

The screenshot shows a repository named 'analista / primeiro'. The top navigation bar includes 'Files' (selected), 'Issues 0', and 'Pull Requests'. Below this, a summary box says 'Meu primeiro repositório com Gogs' with '1 Commits'. A green button labeled 'New file' is visible. The main area lists files: 'README.md', 'arquivo1.txt', 'arquivo2.txt', 'arquivo3.txt', 'arquivo4.txt', and 'arquivo5.txt'. A preview of 'README.md' shows the text 'Meu primeiro repo'.

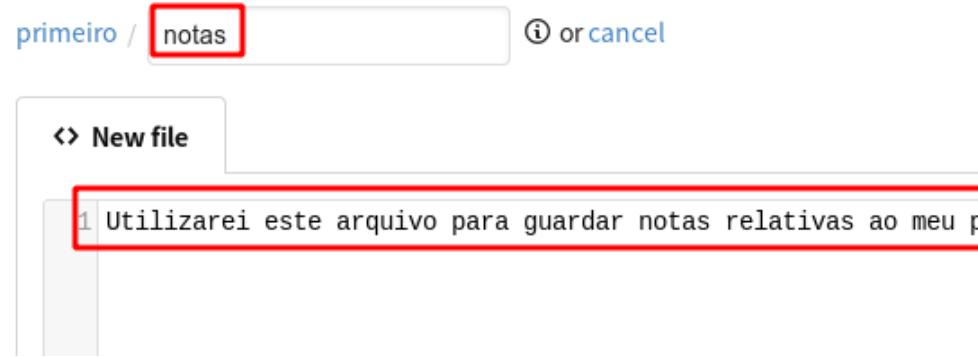
Atualize a página do repositório no gogs para visualizar os arquivos disponíveis no servidor.

Na página do repositório temos acesso a todas as informações do repositório, como por exemplo: \* ID do Commit \* Usuário que realizou o Commit \* Arquivos do Re却itório \* Quantidade de Branches e Commits

Também é possível enviar e criar arquivos diretamente pela interface web através dos botões **New file** e **Upload file**. Clique

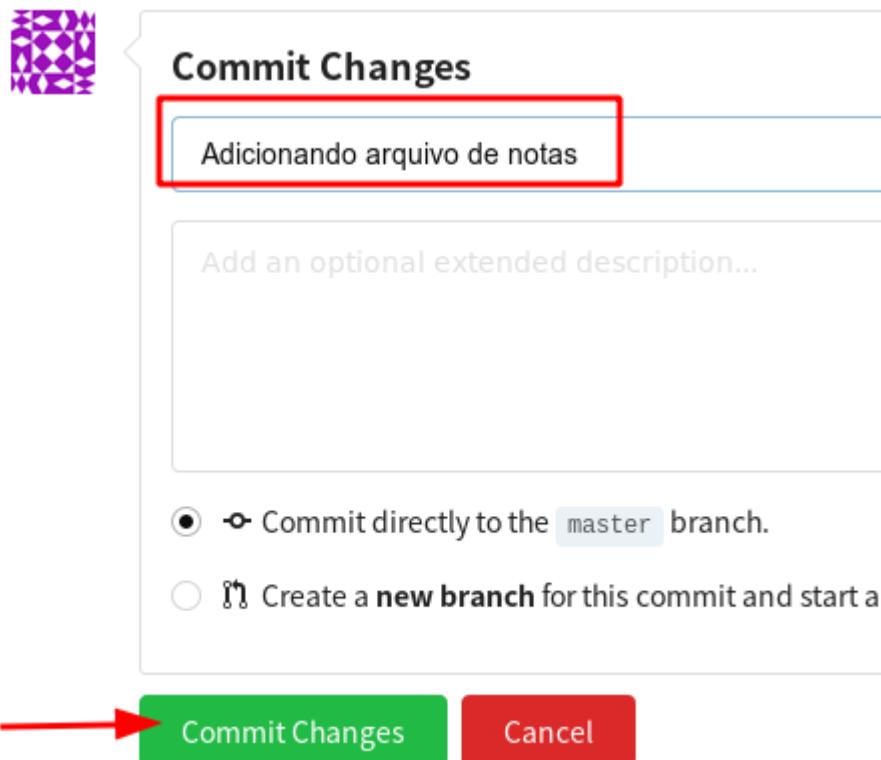
The screenshot shows the same repository interface. A red arrow points to the 'New file' button in the top right corner of the header bar. The rest of the interface is identical to the previous screenshot, showing files and a preview of the README content.

em **New File** para criarmos um novo arquivo.



Utilize o editor web para criar o arquivo **notas**

Na parte inferior podemos adicionar uma mensagem de commit e enviar as alterações diretamente para nosso repositório



Clicando no nome do repositório **primeiro** podemos voltar para a home do repositório e verificar o arquivo adicionado e a

Meu primeiro repositório com Gogs

2 Commits 1 Branches 0 Releases

Branch: master primeiro

New file Upload file HTTP SSH http://scm.4labs.example/a

File	Commit	Message	Time
Analista DevOps	50fe4a4b30	Adicionando arquivo de notas	24 sec
README.md	652655f9c3	Meu primeiro commit no gogs	14 min
arquivo1.txt	652655f9c3	Meu primeiro commit no gogs	14 min
arquivo2.txt	652655f9c3	Meu primeiro commit no gogs	14 min
arquivo3.txt	652655f9c3	Meu primeiro commit no gogs	14 min
arquivo4.txt	652655f9c3	Meu primeiro commit no gogs	14 min
arquivo5.txt	652655f9c3	Meu primeiro commit no gogs	14 min
notas	50fe4a4b30	Adicionando arquivo de notas	24 sec

mensagem de commit.

Para atualizar o seu repositório local, execute o comando `git pull` e verifique os arquivos recebidos

```
1 git pull origin master
```

```
1 Updating 652655f..50fe4a4
2 Fast-forward
3  notas | 1 +
4  1 file changed, 1 insertion(+)
5  create mode 100644 notas
```

## SSH Keys

SSH Keys são chaves utilizadas para poder acessar e manipular um repositório utilizando o serviço ssh.

Neste caso, o SSH Keys são associadas ao usuário, sendo chaves pessoais, tornando desnecessária a utilização de senhas ao realizar um `git clone`, `git push`, ou `git pull` de um repositório privado.

Para adicionar uma **SSH Keys** ao usuário *analista* precisamos acessar a página do gogs e efetuar o login com um usuário dono

Sign In

Username or email \* analista

Password \* devops@4linux

Remember Me

[Sign In](#) [Forgot password?](#)

Need an account? [Sign up now.](#)

do projeto

Repository Organization Mirror

My Repositories 1

**primeiro** 0 ★

## Collaborative Repositories

Clique no repositório criando anteriormente:

Vamos agora alterar o repositório para **Privado** no menu **Settings, Options**, marcando a opção **This repository is Private** e

**analista / primeiro**

Files Issues 0 Pull Requests 0 Wiki

Settings Options Collaboration Branches Webhooks Git Hooks Deploy Keys

Basic Settings

Repository Name \* primeiro

Description

Meu primeiro repositório com Gogs

Description of repository. Maximum 512 characters length. Available characters: 478

Official Site

Visibility  This repository is **Private**

[Update Settings](#)

clicando em **Update Settings**

Com isso, podemos agora acessar o menu do usuário analistas e ir em **Your Settings** e no menu a esquerda clicar em **SSH**

The screenshot shows the GitHub user settings interface. On the left, a sidebar lists various settings options: Profile, Avatar, Password, Email Addresses, **SSH Keys** (which is selected and highlighted with a red box), Security, Repositories, Organizations, Applications, and Delete Account. On the right, the main content area is titled "Manage SSH Keys". It contains a message about the importance of SSH keys for repository access. Below this, there's a note about GitHub's guide for creating SSH keys and solving common issues. At the top right of the main area, it says "SIGNED IN AS ANALISTA". To the right of the main content, there's a vertical sidebar with links: Your Profile, **Your Settings** (which is also highlighted with a red box), Help, Admin Panel, and Sign Out.

## Keys

Já temos uma chave pública em cada máquina, iremos utilizar a mesma e copiar o seu conteúdo para o gogs

```
1 vagrant ssh scm
2 cat /vagrant/files/key.pub
```

```
1 ssh-rsa AAAAB3NzaC1yc2EAAAQABAAQCXAbLFK7Vcdk4fjiCDqIoL9/rLC0iYaGxBPXCgEOBDRpK2eFRsgLgzIgNG+
kXUbayJht1YYoQzkLahCUDrA+
go0XVGwKg6GuLM0sLIqph0vX53238np8HMs0t5hZqBSVM55Se1YDq6F0jMW9pWWBCwNE3ESbla4Dsp0Y1Jh3cNFjUB1xFe+
YEqiN/y+KC6hG8nQA8SmQ2TPf3DewLqbpe9jZQclxT/Z86wAyexFjrTl9cSQTNE0lpmiwKttM3/0V+
h2kuAUIImrw0nABqJeQvezMAIdyZpKZKf2WYppz3ukf0vLNw5dJBY1gx8T0cas791hQsWDd3Jt4G0MsYi7
```

Agora vamos clicar em **Add Key** e, em seguida, de um nome a sua chave e cole o conteúdo da chave pública no campo **content**,

## Manage SSH Keys

This is a list of SSH keys associated with your account. As these keys allow anyone using them to access to your repositories, it is highly important that you make sure you recognize them.

**Don't know how?** Check out GitHub's guide to [create your own SSH keys](#) or solve [common problems](#) you may encounter using SSH.

### Add SSH Key

#### Key Name

personal\_key

#### Content

```
ssh-rsa
AAAAB3NzaC1yc2EAAAQABAAQCAubLFK7Vcdk4fjiCDqloL9/rLCOiYaGxBPXCgB
K2eFRsgLgzIgNG+kXUbayJhtlYYoOzkLahCUDrA+go0XVGwKg6GuLM0sLlqphOvX53238
0t5hZqBSVM55SelYDq6FOjMW9pWWBCwNE3ESbla4Dsp0Yljh3cNFjUBIxFe+YEgiN
/y+KC6hG8nQA8SmQ2TPf3DewLqbpe9jZQclxT/Z86wAyexFjrTl9cSQTNEOpmiwKttM3
/OV+h2kuAUllmrw0nAbqjeQvezMAldyZpKZKf2WYppz3ukfOvLNw5dJBY1gx8TOocas791
3Jt4GOMsYi7
```

Add Key

em seguida clique em **Add Key**

## Manage SSH Keys

This is a list of SSH keys associated with your account. As these keys allow anyone using them to access to your repositories, it is highly important that you make sure you recognize them.



personal\_key

SHA256:Ut150L/kVED+Xvf2O84i+J30KEGaPtz

Added on Jan 11, 2020 — ⓘ No recent activity

E após clicar em **Add Key** a chave será armazenada, conforme abaixo:

analista / primeiro

Files Issues 0 Pull Requests 0

Meu primeiro repositório com Gogs

2 Commits

Branch: master primeiro

Analista DevOps 50fe4a4b30 Adicionando arquivo d

Volte a home do repositório e copie o campo **ssh** para efetuar o clone do repositório.

A chave SSH está sob poder do usuário root, vamos alterar o usuário na máquina **scm** para o root, acessar o diretório **/tmp** e testar o clone através do terminal

```
1 sudo su -  
2 cd /tmp/  
3 git clone ssh://gogs@scm.4labs.example/analista/primeiro.git  
4 ls -l /tmp/primeiro
```

As vezes pode ser necessário reescrever o arquivo **.ssh/authorized\_keys** para ajustar ID de chaves invalidos. Com isso, é necessário seguir o caminho **Admin Panel -> Dashboard -> Operations Rewrite '.ssh/authorized\_keys' -> Run**

## Capítulo 5

# Automação para ambientes ágeis com Ansible

### Características do Ansible

O que é o Ansible?



Figura 5.1: Ansible

Ansible é uma ferramenta de automação criada para gerenciar múltiplas máquinas de uma única vez, através de scripts escritos em formato **YAML**. Sua popularização atual se deve muito a sua documentação, considerada por muitos uma das mais detalhadas, além da curva de aprendizado ser amigável, permitindo que qualquer pessoa possa escrever suas próprias **playbooks** com certa facilidade.

## Características

- Arquitetura Master to Nodes;
- Linguagem de Configuração Simples;
- Não necessita do uso de agents;
- Adaptável a qualquer situação;
- Simples de realizar manutenção.

## Arquitetura

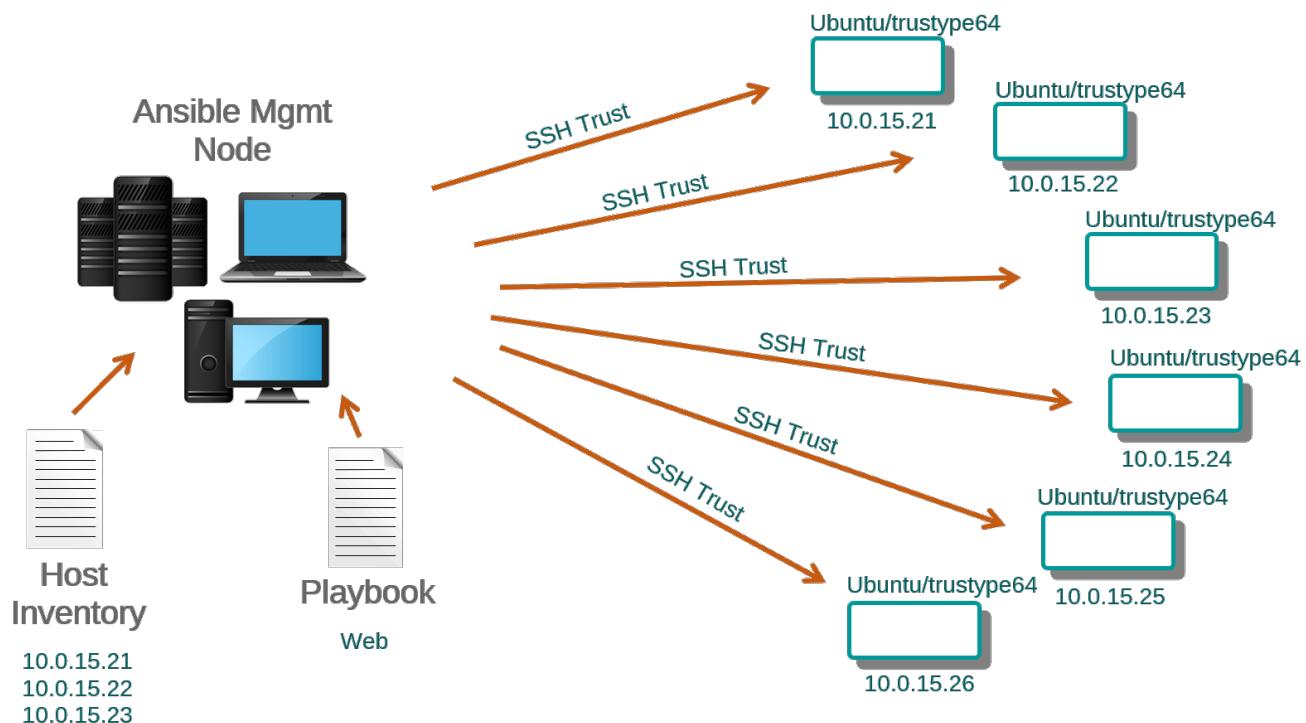


Figura 5.2: Arquitetura

O Ansible possui uma arquitetura **MASTER TO NODES**, sendo necessário apenas que os nós possuam o **Python** instalado e uma conexão SSH com a máquina que deverá responder ao comando remoto, podendo ser realizada via usuário/senha ou até mesmo troca de chaves.

## Instalação do Ansible

Executaremos o ansible na máquina automation. Acesse a máquina automation via ssh e instale o repositório *epel-release* e o *vim*.

```
1 vagrant ssh automation
2 sudo yum install epel-release vim -y
```

O pacote *epel-release* é responsável pela instalação dos pacotes **Extra Packages for Enterprise Linux** da RedHat

Instale o ansible

```
1 sudo yum install ansible -y
```

Para verificar a instalação do ansible:

```
1 ansible --version
```

## Inventário

O arquivo de inventário do Ansible é onde armazenamos a lista de servidores. Por padrão ele fica localizado em `/etc/ansible/hosts`

Dentro do arquivo podemos definir um host ou múltiplos, depois separá-los por grupos com o uso de “[ ]”

Edite o arquivo hosts:

```
1 sudo vim /etc/ansible/hosts
```

```
1 [automation]
2 automation.4labs.example ansible_python_interpreter=/usr/bin/python
3
4 [compliance]
5 compliance.4labs.example ansible_python_interpreter=/usr/bin/python3
6
7 [container]
8 container.4labs.example ansible_python_interpreter=/usr/bin/python
9
10 [scm]
11 scm.4labs.example ansible_python_interpreter=/usr/bin/python
12
13 [log]
14 log.4labs.example ansible_python_interpreter=/usr/bin/python3
```

No arquivo de inventário podemos especificar uma máquina colocando o hostname, endereço IP ou até mesmo um grupo utilizando chaves.

Além de especificar as máquinas em si, é possível registrar variáveis de ambiente para cada máquina, de forma independente. Para tal, basta adicionar no final do nome das máquinas o a declaração da variável de ambiente conforme o exemplo abaixo:

```
1 [ubuntu]
2 srv01.4labs.example ansible_python_interpreter=/usr/bin/python
3
4 [rhel]
5 srv02.4labs.example ansible_python_interpreter=/usr/bin/python3
6 srv03.4labs.example ansible_python_interpreter=/usr/bin/python3
```

## Configuração do ansible

No arquivo `ansible.cfg` podemos definir os parâmetros de configuração do ansible

Edite o arquivo de configuração do ansible

```
1 sudo vim /etc/ansible/ansible.cfg
```

Edite os valores:

```
1 host_key_checking = False
2 private_key_file = /root/.ssh/id_rsa
3 roles_path      = /etc/ansible/roles
```

Estamos definindo qual será a chave de acesso privada a ser utilizada nas conexões SSH. Definimos também que não será feita a checkagem do fingerprint

Teste a conexão com as máquinas listadas no inventário

```
1 sudo ansible all -m ping
```

O comando acima faz com que o programa (**ansible**) execute o módulo (**-m**) **ping** em todos os hosts (**all**)

O Módulo ping do ansible executa uma conexão SSH e retorna caso a conexão tenha sido realizada com sucesso ou não.

## Usando o Ansible Ad-hoc

### Módulos

Módulos são utilizados para descrever as atividades que o ansible executará na máquina de destino. Alguns desses módulos são

Instalação	Arquivos	Execução	Download
package	file	command	get_url
yum	unarchive	shell	git
apt	copy	raw	unarchive
pacman	template		fetch

Ansible possui milhares de módulos para as mais variáveis atividades, desde gerenciar os pacotes que serão instalados em uma máquina virtual, até o gerenciamento de conteineres. Alguns dos módulos mais utilizados no Ansible são:

- **apt**: Gerencia a instalação de pacotes, utilizando o APTITUDE (Debian/Ubuntu);
- **yum**: Gerencia a instalação de pacotes, utilizando o YUM (RedHat/CentOS/Fedora);
- **package**: Gerencia a instalação de pacotes, utilizando o APTITUDE/YUM;
- **command**: Executa um comando em node remoto;
- **shell**: Executa um shell script dentro da máquina, após realizar as transferências;
- **service**: Gerencia serviços em máquinas remotas;
- **copy**: Copia arquivos na máquina local para um node remoto;
- **git**: Gerencia repositórios do git;
- **unarchive**: Descompacta arquivos na máquina remota;
- **mount**: Gerencia os dispositivos montados na máquina;
- **template**: Gerencia templates no ansible.

A Documentação de todos os módulos disponíveis do Ansible e seus parâmetros podem ser visualizados no website Ansible Docs

Também podemos utilizar o comando **ansible-doc** para visualizar a documentação.

Conecte-se na máquina automation e em seguida verifique a documentação:

```
1 vagrant ssh automation
2 ansible-doc --help
```

Liste os módulos

```
1 ansible-doc -l
```

O parâmetro **-l** lista todos os módulos disponíveis

Para visualizar a documentação de um módulo basta usar o comando

```
1 ansible-doc <MODULO>
2 ansible-doc -s <MODULO>
```

O parâmetro **-s** mostra um pequeno fragmento da documentação

## Comandos

Para executar um comando no ansible precisamos informar qual o módulo será utilizado, quais os parâmetros se houverem e quais máquinas do inventário receberão o comando.

```
1 ansible -m <MODULO> -a <PARAMETROS> <HOSTS>
```

Ou

```
1 ansible <HOSTS> -m <MODULO> -a <PARAMETROS>
```

Por padrão o Ansible lê o arquivo de inventário localizado em **/etc/ansible/hosts**, porém é possível que seja informado outro arquivo de inventário passando o parâmetro **-i** seguido do arquivo de inventário a ser utilizado.

**Exemplo:** ansible srv01 -m ping -i /files/inventario

Vamos realizar algumas tarefas rotineiras utilizando o ansible

Primeiramente visualize o *help* do ansible

```
1 ansible --help
```

Lendo o conteúdo de um arquivo no servidor compliance

```
1 sudo ansible compliance -m shell -a "cat /etc/hosts"
```

Transferindo um arquivo para o servidor log

```
1 sudo ansible log -m copy -a "src=/etc/hosts dest=/tmp"
```

Instalando um aplicativo no servidor compliance

```
1 sudo ansible compliance -m apt -a "name=cmatrix state=present update_cache=yes"
```

Reiniciando um serviço no servidor automation

```
1 sudo ansible automation -m service -a "name=crond state=restarted enabled=yes"
```

## Facts

Facts são informações coletadas pelo Ansible quando o mesmo é executado em uma máquina remota.

Através dele podemos coletar diversas informações importantes tais como armazenamento, configurações de redes, configurações do sistema, versão do sistema operacional dentre outras.

Para visualizar os dados trazidos pelo **facts**, podemos utilizar o módulo **setup**

Verifique os facts do servidor log

```
1 sudo ansible log -m setup
```

Colete os facts de todos os servidores

```
1 sudo ansible all -m setup
```

Filtrando uma seleção de facts

```
1 sudo ansible all -m setup -a "filter=ansible_distribution"
```

## Introdução a Linguagem YAML

### O que é YAML



Figura 5.3: YAML

O **YAML** (YAML Ain't Markup Language) é um formato de serialização de dados legíveis por humanos para todas as linguagens de programação.

É amplamente utilizadas ambientes DevOps devido a sua facilidade de interpretação.

O YAML utiliza de uma notação simples de se ler e escrever - o Ansible utiliza esse formato nos arquivos de configuração das *playbooks* e *roles*.

Exemplo de arquivo em YAML:

```
1 # Funcionário da Dexter
2
3
4 - name: João Devops da Silva
5   job: Analista DevOps
6   skills:
7     - Administração de Sistema GNU/Linux
8     - Automação com Ansible
9     - Gerencia de Configuração com Puppet
10    - Testes de Infraestrutura com Inspec
11    - Versionamento com Git e Gogs
```

## Características

- A estrutura do documento é composta por identação com espaços em branco, não é permitido o uso de caracteres de tabulação (**TAB**) para a identação.
- Comentários começam com o caractere cerquilha (#) e continuam até o final da linha
- Os valores simples não levam aspas, mas podem ser incluidas aspas (") ou aspas simples ('')
- Listas são encabeçadas por um traço (-) nos títulos e com um membro em cada linha
- Vetores associativos são representados por dois pontos (:) seguido por um espaço na forma de “**chave: valor**”
- O arquivo pode ser salvo com a extensão **.yaml** ou **.yml**
- Os arquivos **YAML** podem opcionalmente começar com três traços (—) e terminar com três pontos (...)

## Preparando o VIM

Podemos utilizar alguns recursos no VIM que nos ajudarão a escrever nossos arquivos no formato YAML, para isso iremos adicionar alguns parâmetros no arquivo vimrc

Na máquina automation, edite o arquivo vimrc e adicione o conteúdo no final do arquivo

```
1 sudo vim /etc/vimrc +$
```

```
1 "Recursos para o YAML
2 set cursorline
3 set cursorcolumn
4 set number
5 set paste
```

- **vimrc** -> Arquivo de parametrização do vim
- **cursorline** -> Exibe uma linha onde o cursor estiver posicionado
- **cursorcolumn** -> Exibe uma coluna onde o cursor estiver posicionado
- **number** -> Exibe o numero das linhas

## Verificando a Sintaxe

Existem alguns recursos na internet onde podemos digitar/colar o conteúdo do nosso arquivo YAML para que o mesmo seja validado. Isso nos ajuda inicialmente a corrigir pequenos erros em nosso YAML

## YAML Lint

Paste in your YAML and click "Go" - we'll tell you if it's valid or not. Optimized for Ruby.

YAML Lint

```
1 ---
2 # Funcionário da Dexter
3
4 - name: João Devops da Silva
5   job: Analista DevOps
6   skills:
7     - Administração de Sistema GNU/Linux
8     - Automação com Ansible
9     - Gerencia de Configuração com Puppet
10    - Testes de Infraestrutura com Inspec
11    - Versionamento com Git e Gogs|
```

```

1  ---
2  # Funcionário da Dexter
3
4  - name: João Devops da Silva
5    job: Analista DevOps
6    skills:
7      - Administração de Sistema GNU/Linux
8      - Automação com Ansible
9      - Gerencia de Configuração com Puppet
10     - Testes de Infraestrutura com Inspec
11     - Versionamento com Git e Gogs

```

YAML Validator

## Execuções em Playbooks

### O que são playbooks

**Playbooks** são arquivos do Ansible em que você descreve as tarefas a serem executadas nos servidores. Em um nível mais básico, as Playbooks podem ser usadas para gerenciar as configurações e realizar deploys nas máquinas remotas.

### Escrevendo uma Playbook

Antes de escrever nossa primeira Playbook, iremos criar um diretório para organizar nossas playbooks e exemplos de playbooks.

Troque para o usuário root.

```
1 sudo su -
```

Crie a estrutura de diretórios e acesse a pasta exemplos

```
1 mkdir -p /etc/ansible/{playbooks,exemplos}
2 cd /etc/ansible/exemplos
```

Nossa primeira playbook será responsável por instalar e transferir o arquivo **resolv.conf** para todas as máquinas.

Crie o arquivo resolv.conf

```
1 cat > resolv.conf <<EOF
2 # Atualizado via ansible
3 nameserver 8.8.8.8
4 nameserver 8.8.4.4
5 search 4labs.example
6 domain 4labs.example
7 EOF
```

Vamos agora criar a nossa playbook

```
1 vim primeiro.yml
```

```
1 # Minha primeira Playbook
2
3 - name: Minha primeira playbook
4   hosts: all
5   tasks:
6
7   - name: Atualizar o resolv.conf
8     copy:
9       src: resolv.conf
10      dest: /etc/resolv.conf
```

Execute a playbook

```
1 ansible-playbook primeiro.yml
```

Utilize o módulo shell para verificar o arquivo modificado

```
1 ansible all -m shell -a "cat /etc/resolv.conf"
```

## Trabalhando com Variáveis

Através de variáveis podemos permitir que nossas playbooks se comportem de forma dinâmica. Ao se criar uma variável você está permitindo que o usuário gerencie qual será a informação daquele campo, permitindo melhor reusabilidade da playbook

A Maneira mais simples de declarar variáveis é através de uma seção **vars** dentro da nossa playbook com os nomes e valores das variáveis

```
1 vars:
2 - server_name: servidor01
3 - username: analistadevops
4 - conf_file: /opt/app/app.conf
```

Vamos criar agora uma playbook para criação de usuários utilizando variáveis.

```
1 vim users.yml
```

```
1 # Playbook para criar usuários
2 - name: Criacao de usuarios nos servidores
3   hosts: all
4   vars:
5     - username: sysadmin
6   tasks:
7
8   - name: Criando usuarios nos servidores
9     user:
10       name: "{{ username }}"
11       shell: /bin/bash
```

Execute a playbook

```
1 ansible-playbook users.yml
```

Utilize o módulo shell para verificar o usuário sysadmin

```
1 ansible all -m shell -a "getent passwd sysadmin"
```

A grande vantagem de se utilizar variáveis é que podemos declará-las de forma externa a playbook

```
1 ansible-playbook users.yml -e username=developer
```

Utilize o módulo shell para verificar o usuário developer

```
1 ansible all -m shell -a "getent passwd developer"
```

Outra maneira de utilizar variáveis é através de um arquivo externo sendo declarado na seção **vars\_files** playbook

```
1 vars_files:  
2 - config.yml
```

config.yml

```
1 server_name: servidor01  
2 username: analistadevops  
3 conf_file: /opt/app/app.conf
```

Vamos criar o arquivo variaveis.yml

```
1 echo "username: analista" > variaveis.yml
```

Edita a playbook users e modifique o campo de variáveis

```
1 vim users.yml
```

```
1 # Playbook para criar usuarios  
2  
3 - name: Criacao de usuarios nos servidores  
4   hosts: all  
5   vars_files:  
6     - variaveis.yml  
7   tasks:  
8  
9     - name: Criando os usuarios nos servidores  
10       user:  
11         name: "{{ username }}"  
12         shell: /bin/bash
```

Execute a playbook

```
1 ansible-playbook users.yml
```

Utilize o módulo shell para verificar o usuário analista

```
1 ansible all -m shell -a "getent passwd analista"
```

## Laços de Repetição

**Laços de Repetição** são estruturas criadas para que possamos executar uma mesma ação repetidas vezes, utilizando o mesmo trecho de código.

O Ansible permite que criemos laços de repetição utilizando uma variável que receberá uma lista ou uma lista de **loop**, permitindo assim executar a mesma ação de acordo com a quantidade de itens que ela possui.

Vamos criar agora uma playbook para instalação de vários programas.

Crie a playbook loop

```
1 vim loop.yml
```

```
1 # Playbook para Loop
2
3 - name: Instalando uma lista de pacotes na Automation
4   hosts: automation
5   vars:
6     - packages:
7       - wget
8       - vim
9       - tree
10  tasks:
11
12    - name: Instalando pacotes
13      yum:
14        name: "{{ packages }}"
```

Execute a playbook

```
1 ansible-playbook loop.yml
```

Verifique se os pacotes foram instalados

```
1 rpm -qa | egrep "wget|vim|tree"
```

Vamos criar agora uma segunda playbook utilizando loops em variáveis.

Crie a playbook loop2

```
1 vim loop2.yml
```

```
1 - name: Instalando uma lista de pacotes nos servidores Ubuntu
2   hosts: compliance log
3   tasks:
4
5     - name: Instalando pacotes
6       apt:
7         name: "{{ item }}"
8       loop:
9         - sl
10        - cmatrix
11        - ncdt
```

**loop:** Cria uma variável temporária chamada **item**, onde serão armazenados todos os valores que você passou para dentro do loop. Esta variável só existe enquanto estiver sendo executado aquele módulo, após a conclusão os valores serão perdidos. Caso necessite de uma lista

*reutilizável, use variáveis no lugar. Em versões mais antigas do Ansible(anterior a 2.5) o parâmetro a ser utilizado era o **with\_items** ao invés do **loop**, porém o mesmo foi descontinuado.*

Execute a playbook

```
1 ansible-playbook loop2.yml
```

Verifique se os pacotes foram instalados

```
1 ansible compliance -m shell -a "dpkg -l | egrep 'sl|cmatrix|ncdu'"  
2 ansible log -m shell -a "dpkg -l | egrep 'sl|cmatrix|ncdu'"
```

## Condicionais

Existem casos em que precisamos que um módulo só execute se uma determinada condição seja verdadeira, como por exemplo instalar o epel-release apenas se o sistema operacional for um CentOS, para isso utilizamos **Condicionais**.

O Ansible utiliza o parâmetro **when** para efetuar este teste.

Vamos criar agora uma playbook para instalação de programas caso o sistema operacional seja Ubuntu

```
1 vim when.yml
```

```
1 - name: Instalando uma lista de pacotes nos servidores Ubuntu  
2 hosts: all  
3 tasks:  
4   - name: Instalando pacotes  
5     apt:  
6       name: "{{ item }}"  
7     loop:  
8       - cmatrix  
9       - sl  
10      - ncdu  
11    when: ansible_distribution == "Ubuntu"
```

Execute a playbook

```
1 ansible-playbook when.yml
```

*Note que ao executar a playbook é feita uma varredura no inventário e os hosts que não trazem a condicional tratada no when como verdadeira não executam a playbook e apresentam como skipping (pulando) no terminal*

## Capítulo 6

# Construindo a Infraestrutura como Código

## Ansible Galaxy

### O que é uma Role?

Existem momentos em que nossa playbook pode ficar muito grande e complexa, dificultando a manutenção futura. Para facilitar este processo, o Ansible possui o sistema de roles.

Uma role nada mais é do que um diretório contendo um conjunto de arquivos, variáveis e playbooks que deverão ser aplicados em seu ambiente.

### Estrutura de uma Role

Uma role é formada pelos seguintes diretórios:

- **default** -> Diretório onde será informado os valores padrões das variáveis de sua role;
- **files** -> Diretório onde são armazenados arquivos que deverão ser copiados para dentro do servidor;
- **handlers** -> Tarefas que deverão ser executadas caso exista uma diretiva (*notify*) que indica alguma mudança;
- **tasks** -> É igual a uma playbook, deverá conter uma lista de módulos e ordem em que cada um deverá executar e como executar;
- **templates** -> Template de arquivos que deverão ser construídos no servidor;
- **vars** -> Diretório onde deve conter todas as variáveis internas de sua playbook

## Ansible Galaxy



Figura 6.1: Ansible Galaxy

Ansible Galaxy é um site de hospedagem de roles do Ansible, permitindo que você compartilhe suas roles com outros usuários ou utilize as roles de outros usuários.

No site existem roles das mais variadas atividades, desde a instalação do Prometheus até gerenciamento do Bind.

## Usando Roles do Ansible Galaxy

Para trabalhar com roles usamos o comando **ansible-galaxy** que permite criar, listar, baixar e remover roles.

Parâmetros do **ansible-galaxy**

- **search** -> Procura por role no ansible galaxy;
- **list** -> Lista playbooks que estejam instaladas;
- **install** -> Instala uma role;
- **install -f** -> Atualiza a role para a ultima versão;
- **remove** -> Remove uma role;
- **init** -> Inicializa um diretório com a estrutura de uma playbook.

Primeiramente vamos conectar a máquina automation e acessar o diretório de roles

```
1 vagrant ssh automation
2 cd /etc/ansible/roles
```

Procure na base do Ansible Galaxy as roles disponíveis para o **puppet**

```
1 ansible-galaxy search puppet
```

Vamos instalar a role `geerlingguy.puppet`

```
1 sudo ansible-galaxy install geerlingguy.puppet
```

Crie uma playbook chamada `compliance.yml` dentro do diretório de playbooks para que possamos utilizar a role que instalamos

```
1 sudo vim /etc/ansible/playbooks/compliance.yml
```

```
1 - name: Aplicando as roles de Compliance
2   hosts: compliance
3   roles:
4     - geerlingguy.puppet
```

Execute a playbook

```
1 sudo ansible-playbook /etc/ansible/playbooks/compliance.yml
```

## Criando Roles

Para criar uma role utilizamos o parametro **init** do `ansible-galaxy`. Com ele conseguimos criar a estrutura de uma role no diretório que estivermos.

Vamos criar uma role de teste.

Acesse o diretório `/etc/ansible/roles`

```
1 cd /etc/ansible/roles
```

Crie a role teste:

```
1 sudo ansible-galaxy init teste
```

Verifique a estrutura de diretórios da role:

```
1 tree /etc/ansible/roles/teste
```

## Construindo a Role puppet-agent

### Entendendo a Estrutura

O Puppet Agent será responsável pela comunicação da estação com o servidor Puppet.

Iremos separar nossa instalação em blocos devido aos diferentes sistemas operacionais utilizados: Debian, Ubuntu e CentOS.

### Separando em Blocos

Para facilitar a visualização das Playbooks, iremos separá-las em blocos de acordo com sua distribuição.

**Blocos** são agrupamentos lógicos de tarefas e são muito utilizados quando queremos aplicar uma diretiva **when**.

Utilizaremos a condicional `when` em cada bloco para filtrar o que deve ser executado em cada distribuição.

### Entendendo a Role do puppet Agent

É importante saber que devemos criar as estruturas condicionais utilizando o `when` para cada bloco conforme o exemplo a seguir:

```
1 - name: Download e instalacao do Puppet Agent - Ubuntu
2   block:
3     - get_url:
4       url: https://apt.puppetlabs.com/puppet6-release-bionic.deb
5       dest: /tmp/puppet6.deb
6     - apt:
7       deb: /tmp/puppet6.deb
8     - apt:
9       name: puppet-agent
10      state: present
11      update_cache: yes
12    when: ansible_distribution == 'Ubuntu'
13
14 - name: Download e instalacao do Puppet Agent - Debian
15   block:
16     - get_url:
17       url: https://apt.puppetlabs.com/puppet6-release-buster.deb
18       dest: /tmp/puppet6.deb
19     - apt:
20       deb: /tmp/puppet6.deb
21     - apt:
22       name: puppet-agent
23       state: present
24       update_cache: yes
25     when: ansible_distribution == 'Debian'
26
27 - name: Download e instalacao do puppet Agent - RedHat
28   block:
29     - yum:
```

```

30     name: 'https://yum.puppet.com/puppet6-release-el-7.noarch.rpm'
31     state: present
32   - yum:
33     name: puppet-agent
34     state: present
35   when: ansible_distribution == 'CentOS'

```

O parâmetro **when** está alinhado com o bloco, isso significa que esta condicional serve para qualquer task dentro do bloco declarado

Para verificar o *ansible\_distribution* de cada máquina podemos executar o comando

```
1 sudo ansible all -m setup -a "filter=ansible_distribution"
```

Precisaremos editar o arquivo de configuração do puppet para que todas as máquinas conheçam o endereço do servidor

```

1 - name: Adicionando configurações do puppet
2   lineinfile:
3     path: '/etc/puppetlabs/puppet/puppet.conf'
4     line: "{{ item }}"
5   loop:
6     - '[main]'
7     - 'server = compliance.4labs.example'
8     - 'runinterval = 1h'

```

Precisaremos também habilitar e executar o puppet agent nos servidores

```

1 - name: Executando o Puppet Agent nos servidores
2   systemd:
3     name: puppet
4     state: restarted
5     enabled: yes

```

## Criando a Role Puppet Agent

Primeiramente iremos criar a estrutura da nossa role utilizando o Ansible Galaxy e então prosseguir com a criação da mesma.

```

1 cd /etc/ansible/roles
2 sudo ansible-galaxy init puppet-agent

```

Crie a task para instalação do puppet-agent

```
1 sudo vim /etc/ansible/roles/puppet-agent/tasks/main.yml
```

```

1 ---
2 # tasks file for puppet-agent
3 - name: Download e instalacao do Puppet Agent - Ubuntu
4   block:
5     - get_url:
6       url: https://apt.puppetlabs.com/puppet6-release-bionic.deb
7       dest: /tmp/puppet6.deb
8     - apt:
9       deb: /tmp/puppet6.deb
10    - apt:
11      name: puppet-agent
12      state: present
13      update_cache: yes
14    when: ansible_distribution == 'Ubuntu'
15

```

```

16 - name: Download e instalacao do Puppet Agent - Debian
17   block:
18     - get_url:
19       url: https://apt.puppetlabs.com/puppet6-release-buster.deb
20       dest: /tmp/puppet6.deb
21     - apt:
22       deb: /tmp/puppet6.deb
23     - apt:
24       name: puppet-agent
25       state: present
26       update_cache: yes
27     when: ansible_distribution == 'Debian'
28
29 - name: Download e instalacao do puppet Agent - RedHat
30   block:
31     - yum:
32       name: 'https://yum.puppet.com/puppet6-release-el-7.noarch.rpm'
33       state: present
34     - yum:
35       name: puppet-agent
36       state: present
37     when: ansible_distribution == 'CentOS'
38
39 - name: Adicionando o binario do Puppet ao PATH
40   block:
41     - file:
42       path: /root/.bashrc
43       state: touch
44     - lineinfile:
45       path: /root/.bashrc
46       line: 'PATH=/sbin:/bin:/usr/sbin:/usr/bin:/opt/puppetlabs/bin:/opt/puppetlabs/bin'
47
48 - name: Adicionando configurações do puppet
49   lineinfile:
50     path: '/etc/puppetlabs/puppet/puppet.conf'
51     line: "{{ item }}"
52   loop:
53     - '[main]'
54     - 'server = compliance.4labs.example'
55     - 'runinterval = 1h'
56
57 - name: Executando o Puppet Agent nos servidores
58   systemd:
59     name: puppet
60     state: restarted
61     enabled: yes

```

Agora precisamos criar nossa playbook para executar a role puppet-agent

```
1 sudo vim /etc/ansible/playbooks/puppet-agent.yml
```

```

1 - name: Instalando e configurando o Puppet Agent
2   hosts: all
3   roles:
4     - puppet-agent

```

Por enquanto não iremos executar a role, uma vez que precisaremos antes parametrizar o servidor.

## Construindo a Role nginx

### O que é o NGINX

O NGINX é um servidor web conhecido por sua alta performance, estabilidade, simples configuração e baixo consumo de recurso.

## Estrutura

O NGINX possui alguns diretórios importantes para seu funcionamento:

- **sites-available** -> Diretório onde ficam localizados todos os arquivos de configuração dos Virtual Hosts disponíveis para serem habilitados.
- **sites-enabled** -> Diretório onde ficam os Virtual Hosts habilitados, normalmente são utilizados links simbólicos para arquivos localizados no diretório sites-available.

O NGINX utiliza arquivos de Virtual Host para configurar o destino de uma requisição que o mesmo receba.

```
1 server {  
2     listen 80;  
3     root /var/www/html;  
4     index index.html index.htm;  
5     location / {  
6         try_files $uri $uri/ = 404;  
7     }  
8 }
```

## Criando a Role NGINX

Primeiramente iremos criar a estrutura da nossa role utilizando o Ansible Galaxy e então prosseguir com a criação da mesma.

```
1 cd /etc/ansible/roles  
2 sudo ansible-galaxy init nginx
```

Crie o arquivo index.html

```
1 sudo vim /etc/ansible/roles/nginx/files/index.html
```

```
1 <!DOCTYPE html>  
2 <html>  
3     <meta charset="UTF-8">  
4     <body>  
5         <h1> Bem vindo a 4Labs</h1>  
6         <p>Esta é uma página de teste da 4Labs</p>  
7     </body>  
8 </html>
```

Edite a task principal da nossa role

```
1 sudo vim /etc/ansible/roles/nginx/tasks/main.yml
```

```
1 ---  
2 # tasks file for nginx  
3 - name: Instalando o NGINX  
4   package:  
5     name: nginx  
6     state: present  
7  
8 - name: Removendo website padrao  
9   file:  
10    path: "{{ item }}"  
11    state: absent  
12  loop:  
13    - /var/www/html/index.nginx-debian.html
```

```
14     - /var/www/html/index.html
15     - /var/www/html/index.htm
16
17 - name: Copiando website 4labs
18   copy:
19     src: index.html
20     dest: /var/www/html/index.html
21
22 - name: Habilitando e reiniciando o serviço do NGINX
23   service:
24     name: nginx
25     state: restarted
26     enabled: yes
```

Crie a playbook para aplicar a role nginx

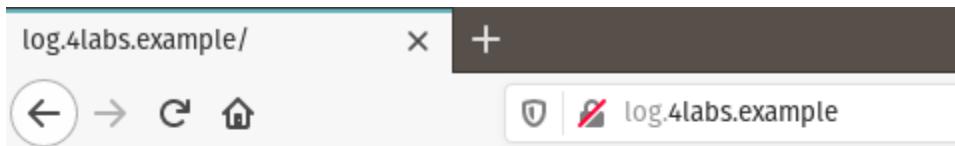
```
1 sudo vim /etc/ansible/playbooks/log.yml
```

```
1 - name: Aplicando as roles de log
2   hosts: log
3   roles:
4     - nginx
```

Execute a playbook

```
1 sudo ansible-playbook /etc/ansible/playbooks/log.yml
```

Acesse pelo navegador o endereço <http://log.4labs.example> e verifique se o website está disponível



## Bem vindo a 4Labs

Esta é uma página de teste da 4Labs

Figura 6.2: NGINX

## Construindo a Role Rundeck

### O que é o Rundeck

O Rundeck é um software de código aberto para automatização de processos e rotinas de operação amplamente utilizado em ambientes DevOps de Entrega Contínua.

### Estrutura

O Rundeck possui alguns arquivos importantes para seu funcionamento e parametrização.

- **framework.properties** -> Arquivo de configuração utilizado pelo core do Serviço do Rundeck onde são configurados o hostname dentre outras configurações
- **rundeck-config.properties** -> Arquivo onde é configurado o webapp do Rundeck, define o nível padrão de log, origem de dados e customização da interface gráfica.

Iremos modificar alguns campos no **framework.properties** para que o nosso Rundeck responda pelo nome e porta corretos, por padrão o Rundeck responde na porta 4440, iremos manter esta configuração

### **framework.properties**

```
1 framework.server.hostname = automation.4labs.example
2 framework.server.name = 4labs
3 framework.server.port = 4440
4 framework.server.url = http://automation.4labs.example:4440
```

Temos também que modificar um campo chamado **grails.serverURL** no arquivo **rundeck-config.properties**.

```
1 grails.serverURL=http://automation.4labs.example:4440
```

## Criando a Role Rundeck

Primeiramente iremos criar a estrutura da nossa Role utilizando o Ansible Galaxy

```
1 cd /etc/ansible/roles
2 sudo ansible-galaxy init rundeck
```

Edite a task principal da role Rundeck

```
1 sudo vim /etc/ansible/roles/rundeck/tasks/main.yml
```

```
1 ---
2 # tasks file for rundeck
3 - name: Instalação das Dependências e Repositório Rundeck
4   yum:
5     name: "{{ item }}"
6     state: present
7   loop:
8     - vim
9     - java-1.8.0
10    - https://repo.rundeck.org/latest.rpm
11
12 - name: Instalação do Rundeck
13   yum:
14     name: rundeck
15     state: present
16
17 - name: Configurando Rundeck
18   lineinfile:
19     path: "{{ item.path }}"
20     regexp: "{{ item.regexp }}"
21     line: "{{ item.line }}"
22   loop:
23     - {path: '/etc/rundeck/framework.properties', regexp: '^framework.server.name', line: 'framework.server.name = 4labs'}
24     - {path: '/etc/rundeck/framework.properties', regexp: '^framework.server.hostname', line: 'framework.server.hostname = automation.4labs.example'}
25     - {path: '/etc/rundeck/framework.properties', regexp: '^framework.server.port', line: 'framework.server.port = 4440'}
```

```

26   - {path: '/etc/rundeck/framework.properties', regexp: '^framework.server.url', line: 'framework.
27     server.url = http://automation.4labs.example:4440'}
28   - {path: '/etc/rundeck/rundeck-config.properties', regexp: '^grails.server', line: 'grails.
29     serverURL=http://automation.4labs.example:4440'}
30
31 - name: Iniciando e Habilitando o serviço do rundeck
32   systemd:
33     name: rundeckd
34     state: started
35     enabled: yes

```

Crie a playbook para aplicar a role Rundeck

```
1 sudo vim /etc/ansible/playbooks/rundeck.yml
```

```

1 - name: Instalação do Rundeck
2   hosts: automation
3   roles:
4     - rundeck

```

Execute a playbook

```
1 sudo ansible-playbook /etc/ansible/playbooks/rundeck.yml
```

Após o término da execução da playbook, acesse pelo navegador o endereço <http://automation.4labs.example:4440>



Figura 6.3: Rundeck

## Construindo a Role Docker

### O que é o Docker

Docker é uma plataforma Open Source escrita em Go que facilita a criação e Administração de Ambientes Isolados

### Estrutura

A Solução Docker será composta por alguns pacotes:

- **docker-ce** -> Docker Community Edition, é o daemon do Docker;
- **docker-ce-cli** -> Docker Community Edition Command Line Interface;
- **containerd.io** -> É um runtime de contêineres com ênfase em simplicidade, robustez e portabilidade;
- **Docker Machine Bash Completion** -> São os recursos de auto-completar do Docker

## Criando a Role Docker

Primeiramente iremos criar a estrutura da nossa role utilizando o Ansible Galaxy e então prosseguir com a criação da mesma.

```
1 cd /etc/ansible/roles
2 sudo ansible-galaxy init docker
```

Edite a task principal da Role Docker

```
1 sudo vim /etc/ansible/roles/docker/tasks/main.yml
```

```
1 ---
2 # tasks file for docker
3 - name: Instalação das Dependências
4   yum:
5     name: "{{ item }}"
6     state: present
7   loop:
8     - epel-release
9     - yum-utils
10    - device-mapper-persistent-data
11    - curl
12    - lvm2
13    - vim
14    - bash-completion
15
16 - name: Adicionar repositório do Docker-Engine
17   get_url:
18     url: https://download.docker.com/linux/centos/docker-ce.repo
19     dest: /etc/yum.repos.d/docker-ce.repo
20
21 - name: Instalar o Docker-Engine
22   yum:
23     name: "{{ item }}"
24     state: present
25   loop:
26     - docker-ce
27     - docker-ce-cli
28     - containerd.io
29
30 - name: Instalar o Docker Machine bash completion
31   get_url:
32     url: https://raw.githubusercontent.com/docker/machine/v0.16.0/contrib/completion/bash/docker-
33     machine.bash
34     dest: /etc/bash_completion.d/docker-machine
35
36 - name: Habilitando e iniciando o serviço do Docker-ce
37   systemd:
38     name: docker
39     state: started
40     enabled: yes
41
42 - name: Instalando o docker-compose
43   get_url:
44     url: https://github.com/docker/compose/releases/download/1.24.0/docker-compose-Linux-x86_64
45     dest: /usr/local/bin/docker-compose
46     mode: 0755
47 - name: Criando link simbólico docker-compose
```

```
48   file:
49     src: /usr/local/bin/docker-compose
50     dest: /usr/bin/docker-compose
51     state: link
```

Crie a playbook para aplicar a role Docker

```
1 sudo vim /etc/ansible/playbooks/container.yml
```

```
1 - name: Instalação do Docker
2   hosts: container
3   roles:
4     - docker
```

Execute a playbook

```
1 sudo ansible-playbook /etc/ansible/playbooks/container.yml
```

Verifique se a instalação foi efetuada com sucesso

```
1 sudo ansible container -m shell -a "docker --version"
2 sudo ansible container -m shell -a "docker system info"
```

## Capítulo 7

# Gerência de Configuração com Puppet

### Estrutura do Puppet

#### O que é o Puppet?



Figura 7.1: Puppet

O Puppet é uma ferramenta de gerenciamento de configurações de sistemas Unix-like e Windows. Através de uma linguagem declarativa é possível descrever todo o ambiente.

O Puppet foi desenvolvido como uma solução para o problema de manter e gerenciar grandes parques de máquinas. Antes da sua criação era um trabalho muito complexo manter a configuração e até mesmo análise de erros, visto que cada máquina em seu ambiente estava com uma configuração diferente.

Atualmente a ferramenta é muito utilizada no mercado, apesar de possuir concorrentes como o Ansible, Chef, Saltstack, entre outros. Em alguns ambientes é possível encontrar o Puppet como mais uma ferramenta de gerenciamento de configurações, muito dos casos com o Ansible. Obviamente nesse caso o Ansible é mais utilizado como uma forma de deploy de aplicações, enquanto o Puppet serve para manter o ambiente.

Um dúvida que surge bastante é com relação a diferença do Puppet para o Ansible, ou até mesmo se é realmente necessário aprender os dois. A maior diferença, além do fato do Ansible ser agentless, é que a ferramenta não é totalmente idempotente, ou seja, a maioria dos commandos do Ansible irão executar diversas vezes, mesmo que na primeira vez já tenha sido alterado.

## Características do Puppet

- **Agent to Master** -> O puppet possui um servidor Master, responsável por manter e gerenciar as configurações de todas as máquinas de seu ambiente
- **Linguagem Declarativa** -> Possui uma linguagem para desenvolvimento de módulos muito semelhante ao C, que permite descrever todo o ambiente de forma fácil;
- **Ambientes Segregados** -> Com um único Puppet Server é possível manter diversos ambientes, permitindo que seja fácil mover uma máquina de um ambiente para outro;
- **Comunicação Segura** -> Toda a comunicação entre os agentes e o servidor é feita através de SSL, garantindo a segurança dos dados que são trafegados entre as máquinas.

## Vantagens do Puppet

- **Padronização** -> Sua infraestrutura agora estará com a mesma configuração, não importa o que aconteça. Caso uma mudança seja realizada na máquina, o agente replicará a configuração descrita novamente;
- **Automação** -> Com o puppet, conseguimos automatizar a aplicação de configuração em diversas máquinas ao mesmo tempo;
- **Unificação de Configuração** -> Diferente do Ansible, o Puppet possui todos os módulos funcionando para qualquer distribuição e/ou sistema operacional.

## Desvantagens do Puppet

- **Linguagem Declarativa** -> Comparado ao Ansible, o Puppet possui uma curva de aprendizado bem maior;
- **Validação da Execução** -> Caso não seja informado uma ordem de execução o Puppet executará um módulo e então passará para o próximo sem esperar seu resultado;
- **Necessita de Agent** -> Diferente do Ansible, que executa seus scripts via SSH e só precisa do Python, o Puppet precisa de um agent para aplicar as configurações.

## Puppet Autônomo x Puppet Server

- **Puppet Autônomo** -> É o modelo onde instalamos os módulos do Puppet em cada máquina e então deixamos que o serviço do agent prossiga executando os passos dentro do arquivo. Funcionaria de certa forma que nem o Ansible, porém nesse caso o Autônomo só executa localmente;
- **Puppet Server** -> Possui um serviço central cuja responsabilidade é validar o certificado de cada agent e então repassar os módulos requisitados. Nesse modelo você não precisa garantir que o modulo tenha sido enviado para o servidor, somente é necessário garantir que o agent esteja sendo executado

Diferente do Ansible, a palavra **módulo** se refere a um conjunto de recursos com o objetivo de configurar algo na máquina hospedeira. Seria o equivalente as roles que vimos anteriormente

## Estrutura do Puppet Server

Um Ambiente que se utiliza o Puppet Master(server) é bem simples, o agent irá requisitar os módulos que deverão ser executados na máquina hospedeira, informando em qual ambiente ele se encontra, seu hostname e seu certificado.

Caso os dados sejam considerados válidos o master irá conferir se dentro do seu catálogo existem os módulos para esse agent. Caso existam, eles serão compilados e então enviados para o agent, que passará a aplicar cada uma dessas configurações.

Ao término, será definido um status para o servidor e então enviado um relatório da execução para o servidor master.

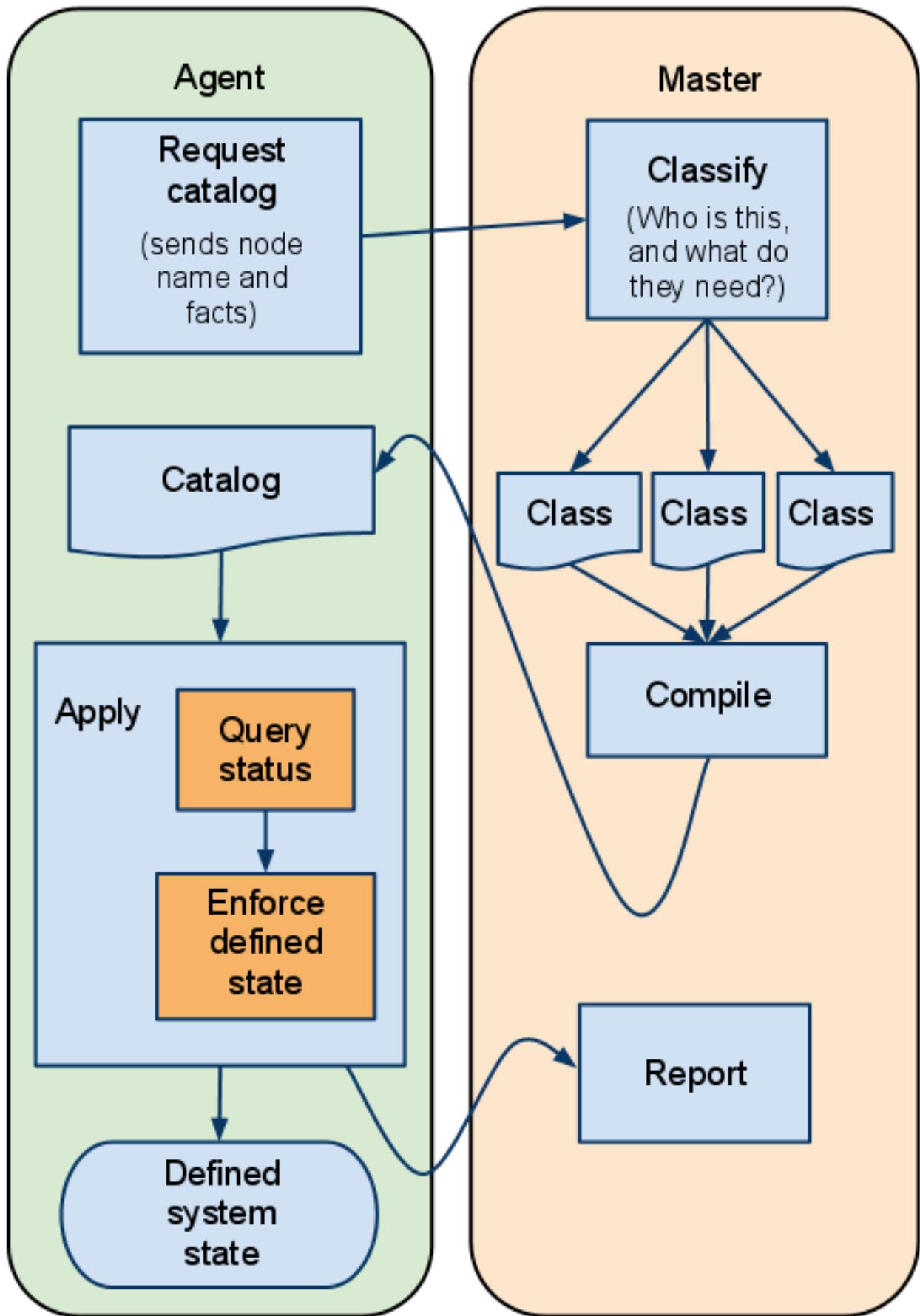


Figura 7.2: Arquitetura Puppet Server  
81

## Recursos

Recursos é como são chamadas as ações que você pode executar numa máquina de destino, sendo desde instalar um pacote, alterar um arquivo, executar um comando, entre outros. Recursos abstraem a necessidade de saber o sistema operacional da máquina de destino.

Package	Service	User
✓ gems;	✓ systemd;	✓ useradd;
✓ yum;	✓ service;	✓ ldapadd;
✓ apt;	✓ launchd;	✓ adduser;
✓ deb;	✓ upstart;	✓ netinfo;
✓ rpm;		

Figura 7.3: Resources

Assim como o Ansible, o Puppet também possui milhares de recursos disponíveis para o uso, e uma documentação para descrever o que fazem e os parâmetros de cada um.

## Usando o Puppet em modo Autônomo e escrevendo Manifestos.

### Instalando o puppet

Para instalar o Puppet utilizaremos a playbook do Ansible clonada através do ansible galaxy.

Acesse a máquina automation e execute a playbook de compliance.

```
1 vagrant ssh automation
2 sudo ansible-playbook /etc/ansible/playbooks/compliance.yml
3 exit
```

Acesse a máquina automation altere para o usuário root e verifique se o puppet foi instalado corretamente

```
1 vagrant ssh compliance
2 sudo su -
3 puppet --version
```

### Modo Autônomo

Primeiramente iremos utilizar o puppet em modo autônomo, para isto devemos entender os recursos disponíveis como **service**, **user**, **group**, **exec**, **package**, **cron** e **file**.

- **service**: Responsável por gerenciar os serviços do sistema;
- **user** -> Responsável por gerenciar as contas de usuários do sistema;
- **mount** -> Responsável por gerenciar os sistemas de arquivos e o arquivo fstab;
- **group** -> Responsável por gerenciar os grupos de usuários do sistema;
- **exec** -> Execute comandos no servidor remoto, contanto que o comando em questão não seja uma shell.
- **package** -> Instalação e remoção de pacotes do servidor, não importando a distribuição;

- **cron** -> Instalação e gerenciamento de tarefas agendadas;
- **file** -> Gerenciar todos os tipos de arquivos do sistema, desde texto a até diretórios.

Liste os usuários na máquina através do recurso **user**

```
1 puppet resource user
```

Para listar um usuário em específico basta passar o nome do usuário como parâmetro

```
1 puppet resource user root
2 puppet resource user linus
```

Para adicionar o usuário através do puppet podemos utilizar o parâmetro *ensure=present*

```
1 puppet resource user linus ensure=present
2 puppet resource user linus
```

Para remover o usuário através do puppet podemos utilizar o parâmetro *ensure=absent*

```
1 puppet resource user linus ensure=absent
2 puppet resource user linus
```

Podemos passar outros parâmetros para garantir que o usuário exista e definir sua pasta home

```
1 puppet resource user linus ensure=present home="/srv/linus"
```

Para gerenciar os serviços podemos utilizar o recurso **service**

```
1 puppet resource service
```

Para listar um serviço em específico, podemos passá-lo como parâmetro.

```
1 puppet resource service cron
2 puppet resource service ntp
```

Para interromper um serviço podemos utilizar o parâmetro *ensure=stopped*

```
1 puppet resource service cron ensure=stopped
2 ps -aux | grep cron
```

Para iniciar o serviço podemos utilizar o parâmetro *ensure=running*

```
1 puppet resource service cron ensure=running
2 ps -aux | grep cron
```

Para gerenciar um pacote podemos utilizar o recurso **package**

```
1 puppet resource package
```

Para listar um pacote em específico podemos passar o nome como parâmetro

```
1 puppet resource package cron
2 puppet resource package ntpdate
3 dpkg -l | egrep "cron|ntpdate"
```

Para instalar um pacote podemos utilizar o parâmetro *ensure=present*

```
1 puppet resource package ntpdate ensure=present
2 dpkg -l | egrep "cron|ntpdate"
```

Para desinstalar um pacote podemos utilizar o parâmetro *ensure=absent*

```
1 puppet resource package ntpdate ensure=absent
2 dpkg -l | egrep "cron|ntpdate"
```

## Factor

O puppet utiliza o factor para recolher informações como detalhes de hardware, configuração de rede dentre outros.

Ele retorna diversos dados em variáveis que podem ser utilizadas posteriormente na construção dos manifestos.

Verifique se o factor está instalado

```
1 factor --version
```

Execute o comando factor para listar todas as informações do sistema

```
1 factor
```

Podemos listar algumas configurações passando-as como parâmetros do comando factor

```
1 factor hostname
2 factor interfaces
```

É possível verificar informações mais a fundo como por exemplo o IP de uma determinada interface

```
1 factor ipaddress_enp0s3
2 factor ipaddress_enp0s8
```

*Os parâmetros possíveis para o filtro do factor podem ser encontrados na Documentação*

## Manifests

Quando precisamos de ações mais complexas do que executar um único comando utilizamos Manifests.

Manifests são basicamente uma coleção de declarações de recursos utilizando a extensão .pp

*Por padrão a pasta de manifests é localizada em /etc/puppetlabs/code/environments/production/manifests é utilizada pelo puppet master(server), posteriormente veremos sobre este assunto.*

Vamos criar nosso primeiro manifest para adicionar um usuário

Crie um diretório para armazenarmos os manifests e em seguida crie o nosso primeiro manifest

```
1 mkdir -p /opt/puppet/
2 vim /opt/puppet/users.pp
```

```

1 user { 'linus':
2     ensure  => 'present',
3     password => '$1$R8ziIJZe$sALXhzd2uEp6NmAfPA3/F/',
4     home    => '/srv/linus'
5 }
6 user { 'analista':
7     ensure  => 'present',
8     password => '$1$R8ziIJZe$sALXhzd2uEp6NmAfPA3/F/',
9     home    => '/srv/analista'
10 }

```

Para criar a senha criptografada podemos utilizar o comando **openssl passwd -1**

Para executar um manifest localmente utilizaremos o comando puppet apply

```
1 puppet apply /opt/puppet/users.pp
```

Verifique se os usuários foram criados com sucesso

```

1 puppet resource user linus
2 puppet resource user analista

```

Efetue o logout e efetue login com o usuário analista e a senha 4linux

```

1 exit
2 su analista

```

Efetue o logout e efetue login com o usuário linus e a senha 4linux

```

1 exit
2 su linus

```

Efetue o logout e retorne para o usuário root

```

1 exit
2 sudo su -

```

Podemos utilizar relacionamentos e ordenações para dizer que um recurso deve ser executado antes ou após outro recurso.

Para isto podemos utilizar os parâmetros **require**, **before**, **notify** e **subscribe**, ou então utilizar **setas de encadeamento**.

- **require** -> Aplica um recurso após o recurso alvo;
- **before** -> Aplica um recurso antes do recurso alvo;
- **notify** -> Aplica um recurso antes do recurso alvo, o alvo é atualizado se o recurso notify for modificado;
- **subscribe** -> Aplica um recurso depois do recurso alvo, o recurso subscribe é atualizado se o recurso alvo for modificado

Para exemplificar o uso dos parâmetros de relacionamento e ordenação iremos instalar um webserver e garantir que o serviço esteja habilitado e executando.

Crie o manifest nginx.pp

```
1 vim /opt/puppet/nginx.pp
```

```

1 package { 'nginx':
2     ensure => installed,
3     before => Service['nginx']
4 }
5 service { 'nginx':
6     ensure => running,
7     enable => true,
8     require => Package['nginx']
9 }

```

*Configuramos para que o serviço só seja habilitado após a instalação do mesmo e que a instalação seja efetuada antes de habilitar o serviço*

Execute o manifest **nginx.pp** e verifique se o serviço foi habilitado e está sendo executado

```

1 puppet apply /opt/puppet/nginx.pp
2 puppet resource service nginx
3 curl compliance.4labs.example

```

Vamos criar um novo manifest utilizando as setas de encaminhamento

```
1 vim /opt/puppet/ntp.pp
```

```

1 package { 'ntp':
2     ensure => installed
3 }
4 -> file { '/etc/ntp.conf':
5     ensure => file,
6     mode   => '0600',
7     source => '/vagrant/files/ntp.conf'
8 }
9 ~> service { 'ntp':
10    ensure => running,
11    enable => true
12 }

```

A seta de ordenação é um hífen e sinal maior que **->**. Aplica o recurso da esquerda ante recurso da direita.

A seta de notify é um til e um sinal de maior que **~>**. Aplica o recurso da esquerda antes, se o recurso modificar algo, o recurso da direita é aplicado.

**Package['ntp'] -> File['/etc/ntp.conf'] ~> Service['ntp']** > Execute o Block Package, então execute bloco File e após sua modificação execute o bloco Service

Execute o manifesto e verifique se o serviço do ntp foi habilitado e está sendo executado

```

1 puppet apply /opt/puppet/ntp.pp
2 puppet resource service ntp
3 head /etc/ntp.conf

```

## PuppetServer

### O que é o PuppetServer

O PuppetServer, também conhecido como **Puppet Master** é responsável por entregar ao **Puppet Agent** o catálogo que deve ser aplicado.

É importante lembrar que a arquitetura do puppet é **Agent to Master**, sendo assim quem efetua a solicitação é o Puppet Agent

## Configurando o Puppetserver

Para garantir o endereço do puppetserver, precisamos adicionar uma entrada no arquivo puppet.conf

Edite o arquivo puppet.conf e adicione o FQDN do Puppet Server

```
1 vim /etc/puppetlabs/puppet/puppet.conf
```

```
1 [main]
2 server = compliance.4labs.example
```

Por padrão são alocados 2g de memória RAM para executar o puppetserver, iremos modificar para que seja utilizado no máximo 512m exclusivo da JVM (Java Virtual Machine) que será utilizada pelo puppetserver

```
1 sed -i s/2g/512m/g /etc/default/puppetserver
```

Execute o serviço do puppetserver e verifique se o mesmo está habilitado

```
1 puppet resource service puppetserver ensure=running enable=true
2 ss -ntpl | grep 8140
```

Precisamos agora habilitar o puppetagent que será responsável por aplicar os manifestos e se comunicar com o puppetserver

Execute o puppet agent

```
1 puppet agent -t
```

O parametro **-t** (*-test*) faz com que o puppet server seja executada uma única vez em primeiro plano

Execute o serviço do agent e garanta que o mesmo está habilitado

```
1 puppet resource service puppet ensure=running enable=true
2 systemctl status puppet
```

## Certificados

O Puppetserver utiliza certificados para verificar a identidade dos nodes (nós). Estes certificados são gerados pelo Certificate Authority(CA) Service do Puppet Master.

Quando um nó conecta ao Puppet Master pela primeira vez, ele solicita um certificado, o Puppet Master examina a requisição e cria o certificado para o nó.

Iremos agora conectar a máquina automation ao puppetserver para que seja gerado um certificado.

Conecte-se em um novo terminal a máquina automation e execute o puppet agent

```
1 ansible-playbook -l automation /etc/ansible/playbooks/puppet-agent.yml
2 vagrant ssh automation
3 sudo su -
4 puppet agent -t --server compliance.4labs.example
```

Após a execução será exibido uma mensagem informando que o certificado foi gerado porém ainda não foi assinado

Na máquina compliance, podemos listar os certificados que não foram assinados ainda

```
1 puppetserver ca list
```

Para assinar um certificado, basta executar o comando sign --certname seguido do nome da máquina que solicitou o certificado.

```
1 puppetserver ca sign --certname automation.4labs.example
```

Para listar todos os certificados podemos utilizar o parâmetro *-all*

```
1 puppetserver ca list --all
```

Na máquina **automation** e execute novamente o puppet agent

```
1 puppet agent -t --server compliance.4labs.example
```

Caso seja necessário revogar o certificado podemos utilizar o parâmetro clean

Execute o comando na máquina **compliance**

```
1 puppetserver ca clean --certname automation.4labs.example
```

Na máquina **automation** e execute novamente o puppet agent

```
1 puppet agent -t --server compliance.4labs.example
```

Será exibido um erro, informando que o certificado é desconhecido, para conectar novamente no puppetserver é necessário que seja removida a pasta com os certificados na máquina **automation** para que seja feita uma nova requisição de certificado

```
1 rm -rf /etc/puppetlabs/puppet/ssl
```

## Autosign

É possível criar um arquivo de autosign com o nome das máquinas que terão seus certificados assinados automaticamente.

Crie o arquivo **autosign.conf** na máquina **compliance**

```
1 vim /etc/puppetlabs/puppet/autosign.conf
```

```
1 automation.4labs.example
```

Na máquina **automation** e execute novamente o puppet agent

```
1 puppet agent -t --server compliance.4labs.example
```

*Devido a existencia do arquivo **autosign.conf** com o registro do nome do servidor **automation.4labs.example** o certificado foi assinado automaticamente*

Liste todos os certificados na máquina **compliance** e verifique que o certificado da **automation** foi assinado automaticamente.

```
1 puppetserver ca list --all
```

Iremos editar o nosso arquivo do autosign para que ele assine automaticamente o certificado para todas as máquinas do domínio 4labs.example

```
1 echo '*.*.4labs.example' > /etc/puppetlabs/puppet/autosign.conf
```

Para o conteúdo do arquivo autosign.conf podemos passar o nome de cada servidor ou então utilizar uma regex como \*.4labs.example para dizer que todas as máquinas de um determinado domínio terão seus certificados autoassinados

Na máquina **automation**, execute a playbook do puppet agent apenas para a máquina automation

```
1 sudo ansible-playbook -l automation /etc/ansible/playbooks/puppet-agent.yml
```

## Módulos e Puppet Forge

### O que são módulos

Módulos são conjuntos de configurações e manifestos utilizados para parametrizar nosso ambiente.

Através dos módulos podemos agrupar as configurações em um único local, sendo possível reutilizar e otimizar nosso código.

### Criação de Módulos

Para a criação de módulos precisamos do Puppet Development Kit (PDK)

Na máquina **compliance** instale o puppet development kit

```
1 puppet resource package pdk ensure=present
```

Acesse o diretório dos módulos

```
1 cd /etc/puppetlabs/code/environments/production/modules
```

Vamos criar nosso primeiro módulo para instalação do inspec.

Execute o comando para criação do módulo através do pdk

```
1 pdk new module puppet-inspec
```

```
1 [Q 1/1] Y
2 [Q 1/4] analistadevops
3 [Q 2/4] Analista DevOps
4 [Q 3/4] <ENTER>
5 [Q 4/4] * Debian
6 Metadata will be generated based on this information, continue? (Y/n) Y
```

Vamos criar o arquivo **init.pp** na pasta manifests do módulo inspec e adicionar as instruções de download e instalação

```
1 vim inspec/manifests/init.pp
```

```

1 class inspec {
2
3     file {'/opt/inspec.deb':
4         path      => '/opt/inspec.deb',
5         source    => 'https://packages.chef.io/files/stable/inspec/4.18.39/ubuntu/18.04/inspec_4
6         .18.39-1_amd64.deb',
7         before    => Package['inspec']
8     }
9
10    package {'inspec':
11        ensure   => installed,
12        provider => dpkg,
13        source   => '/opt/inspec.deb',
14        require   => File['/opt/inspec.deb']
15    }

```

O arquivo `init.pp` é o primeiro arquivo a ser procurado em um módulo do puppet.

Para que o módulo seja aplicado, devemos criar o arquivo `site.pp` com as definições dos nós.

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

```

1 node 'compliance' {
2     include inspec
3 }
4
5 node 'default' {
6 }

```

O arquivo `site.pp` é onde definimos quais módulos devem ser aplicados em cada nó que possua o puppet-agent configurado e sendo executado para garantir a configuração da infraestrutura como código.

O Arquivo segue o padrão:

```

1 node <NOME DA MAQUINA> {
2     include <MODULO>
3     include <MODULO>
4 }

```

Podemos declarar também um nó com o nome **default**, isto faz com que as configurações do bloco default sejam aplicadas a todos os servidores que se conectarem ao Puppet Master e não possuírem definições. >**ATENÇÃO:** O Bloco default deve ser o ultimo bloco, o puppet lê o arquivo de cima para baixo, uma vez que um nó deu **match** com o nome, é feita a compilação do catalogo e a aplicação das configurações.

Caso Necessário podemos utilizar o comando `puppet parser validate` para verificar a sintaxe do nosso código

```
1 puppet parser validate /etc/puppetlabs/code/environments/production/modules/inspec/manifests/init.pp
```

*Caso não seja exibido nenhum retorno, indica que o arquivo está com sua sintaxe correta*

A configuração será aplicada de tempos em tempos conforme configurada no arquivo `/etc/puppetlabs/puppet.conf`, por padrão a configuração é executada a cada 30 minutos.

```

1 [main]
2 server = compliance.4labs.example
3 runinterval = 30m

```

O parametro **runinterval** pode ser modificado para minutos, segundos, horas, dias ou anos. Caso não seja especificado a unidade, o puppet assumirá que o valor inserido é em segundos

Valor	Tempo
runinterval = 10	10 Segundos
runinterval = 30s	30 Segundos
runinterval = 45m	45 Minutos
runinterval = 6h	6 Horas
runinterval = 2d	2 Dias
runinterval = 5y	5 Anos

Para forçar que a configuração seja aplicada na máquina **compliance** podemos utilizar o comando puppet agent

```
1 puppet agent -t
```

Após a aplicação do catalogo, verifique se o inspec foi instalado corretamente

```
1 inspec --version
```

## Puppet Forge

O Puppet Forge é o repositório online para módulos e recursos do puppet, através dele podemos pesquisar e utilizar códigos feitos por outros usuários ou então enviar um código e compartilhar com a comunidade.

Para acessar o Puppet Forge basta abrir o navegador web e acessar o link <https://forge.puppet.com>

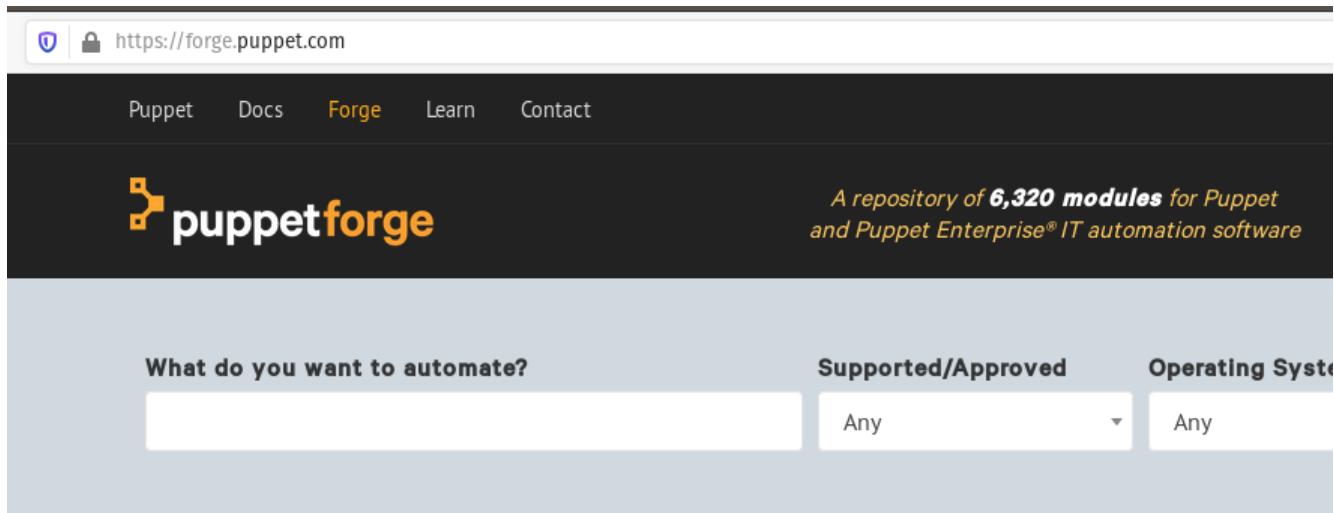


Figura 7.4: Puppet Forge

Podemos utilizar o campo de pesquisa para buscar por algum módulo, iremos procurar pelo módulo **elastic\_stack**

Após selecionar o módulo podemos agora clicar em Manual Installation e copiar o comando para a instalação do módulo

Também é possível utilizar a CLI (Command Line Interface) para procurar por um módulo.

```
1 puppet module search elastic_stack
```

Instale o módulo do elastic\_stack

```
1 puppet module install elastic-elastic_stack --version 7.0.0
```

What do you want to automate? Supported/Approved Operating System With Tasks

elastic Any Any Any

Found 50 modules matching '*elastic*'

Filter by Puppet version: Puppet/Puppet Enterprise Version Sort by: Relevancy

The screenshot shows the Puppet Forge search interface. A red box highlights the search term 'elastic' in the search bar. Below the search bar, there are dropdown menus for 'Supported/Approved', 'Operating System', and 'With Tasks', all set to 'Any'. The search results page displays 50 modules matching the query. One module, 'elastic\_stack' by 'elastic', is highlighted with a red box. This module's details are shown: it was updated 3 months ago, has 8,526,071 total downloads, and a quality score of 5.0. The module description states: 'Helpers for installing and configuring components of the Elastic Stack.' It is version 7.0.0.

Figura 7.5: Puppet Forge

The screenshot shows the detailed page for the 'elastic/elastic\_stack' module. At the top, there is a logo for 'elastic', the name 'elastic/elastic\_stack' with a note 'by: Elastic', and a 'PDK' button. Below this, a description states: 'Helpers for installing and configuring components of the Elastic Stack.' There are links for 'Project URL' and 'RSS Feed'. A section titled 'Latest version is compatible with:' lists compatibility: 'Puppet Enterprise 2019.2.x, 2019.1.x, 2019.0.x, 2018.1.x, 2017.3.x, 2017.2.x, 2016.4.x', 'Puppet >= 4.10.0 < 7.0.0', and 'RedHat, Ubuntu, Debian, SLES, OpenSuSE'. Below this, a section titled 'Start using this module:' shows 'new! Bolt' and 'r10k or Code Manager' options. A red arrow points to the 'Manual installation' section, which contains the command: 'puppet module install elastic-elastic\_stack --version 7.0.' At the bottom right, there is a 'Report issues' link.

Figura 7.6: Puppet Forge

## Escrevendo Módulos

O módulo **elastic-elastic\_stack** é responsável pela instalação dos repositórios do Elastic, iremos agora criar os módulos para instalação dos produtos **elasticsearch**, **logstash**, **kibana** e **filebeat** que serão utilizados posteriormente para gerenciarmos os logs em nossa infraestrutura como código.

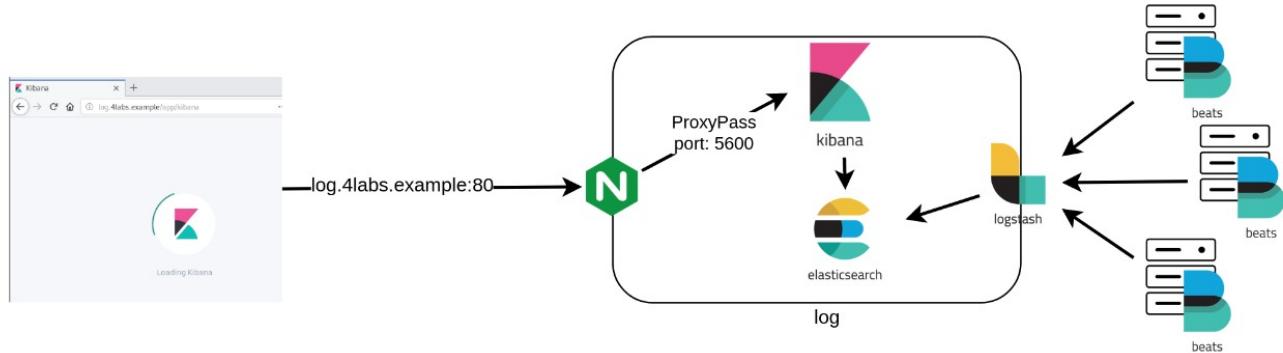


Figura 7.7: Elastic Stack

Iremos criar um manifest para a instalação do Java8 que é uma das dependências necessárias para a instalação do ELK, iremos criar todos os manifests dentro da pasta do módulo do ELK.

```
1 cd /etc/puppetlabs/code/environments/production/modules
2 vim elastic_stack/manifests/java8.pp
```

```
1 class elastic_stack::java8 {
2
3   package {'openjdk-8-jre-headless':
4     ensure  => installed,
5   }
6 }
```

Agora iremos criar o manifest e os arquivos de configuração necessários pra instalação e funcionamento do elasticsearch

```
1 mkdir elastic_stack/files
2 vim elastic_stack/files/elasticsearch.yml
```

```
1 path.data: /var/lib/elasticsearch
2 path.logs: /var/log/elasticsearch
3 network.host: localhost
4 http.port: 9200
```

Crie o manifest para instalação do elasticsearch

```
1 vim elastic_stack/manifests/elasticsearch.pp
```

```
1 class elastic_stack::elasticsearch {
2
3   package {'elasticsearch':
```

```

4   ensure  => installed,
5   before   => Service['elasticsearch']
6 }
7
8 file { 'elasticsearch.yml':
9   path      => '/etc/elasticsearch/elasticsearch.yml',
10  source    => 'puppet:///modules/elastic_stack/elasticsearch.yml',
11  require   => Package['elasticsearch'],
12 }
13
14 file_line {'JVM-elasticsearch-XMS':
15   path  => '/etc/elasticsearch/jvm.options',
16   line   => '-Xms512m',
17   match  => '-Xms1g',
18   notify  => Service['elasticsearch']
19 }
20
21 file_line {'JVM-elasticsearch-XMX':
22   path  => '/etc/elasticsearch/jvm.options',
23   line   => '-Xmx512m',
24   match  => '-Xmx1g',
25   notify  => Service['elasticsearch']
26 }
27
28 service {'elasticsearch':
29   ensure  => running,
30   enable   => true,
31   require   => File['elasticsearch.yml']
32 }
33 }

```

Crie o arquivo kibana.yml que será responsável pela parametrização da aplicação

```
1 vim elastic_stack/files/kibana.yml
```

```

1 server.port: 5601
2 server.host: "localhost"
3 elasticsearch.hosts: ["http://localhost:9200"]

```

Crie o manifest para instalação do Kibana

```
1 vim elastic_stack/manifests/kibana.pp
```

```

1
2 class elastic_stack::kibana {
3
4   package {'kibana':
5     ensure  => installed,
6     before   => Service['kibana']
7   }
8
9   file { 'kibana.yml':
10    path      => '/etc/kibana/kibana.yml',
11    source    => 'puppet:///modules/elastic_stack/kibana.yml',
12    require   => Package['kibana'],
13    notify   => Service['kibana']
14  }
15
16   service {'kibana':
17     ensure  => running,
18     enable   => true,
19     require   => File['kibana.yml']
20   }
21 }

```

Vamos instalar e configurar um NGINX como proxy reverso para o kibana

```
1 vim elastic_stack/files/default
```

```
1 server {
2     listen 80;
3
4     server_name log.4labs.example;
5     location / {
6         proxy_pass http://localhost:5601;
7     }
8 }
```

Crie o manifest para instalação do nginx

```
1 vim elastic_stack/manifests/nginx.pp
```

```
1 class elastic_stack::nginx {
2
3     package {'nginx':
4         ensure => installed,
5         before => Service['nginx']
6     }
7
8     file { 'virtualhost':
9         path      => '/etc/nginx/sites-available/default',
10        ensure   => present,
11        owner    => 'root',
12        group   => 'root',
13        mode     => '0644',
14        source   => 'puppet:///modules/elastic_stack/default',
15        require  => Package['nginx'],
16        notify   => Service['nginx']
17     }
18
19
20     service {'nginx':
21         ensure   => running,
22         enable   => true,
23         require  => File['virtualhost']
24     }
25 }
```

Para o logstash iremos criar alguns arquivos de configurações que serão utilizados pela aplicação.

```
1 vim elastic_stack/files/filebeat-input.conf
```

```
1 input {
2     beats {
3         port => 5443
4         type => syslog
5     }
6 }
```

Crie o arquivo output-elasticsearch.conf

```
1 vim elastic_stack/files/output-elasticsearch.conf
```

```

1 output {
2   elasticsearch { hosts => ["localhost:9200"]
3     hosts => "localhost:9200"
4     manage_template => false
5     index => "%{@metadata}[beat]-%{+YYYY.MM.dd}"
6   }
7 }
```

Crie o arquivo syslog-filter.conf

```
1 vim elastic_stack/files/syslog-filter.conf
```

```

1 filter {
2   if [type] == "syslog" {
3     grok {
4       match => { "message" => "%{SYSLOGTIMESTAMP:syslog_timestamp} %{SYSLOGHOST:syslog_hostname} %{DATA:syslog_program}(?:\[%{POSINT:syslog_pid}\]): %{GREEDYDATA:syslog_message}" }
5       add_field => [ "received_at", "%{@timestamp}" ]
6       add_field => [ "received_from", "%{host}" ]
7     }
8     date {
9       match => [ "syslog_timestamp", "MMM d HH:mm:ss", "MMM dd HH:mm:ss" ]
10    }
11  }
12 }
```

Crie o manifest para instalação do Logstash

```
1 vim elastic_stack/manifests/logstash.pp
```

```

1 class elastic_stack::logstash {
2
3   package {'logstash':
4     ensure  => installed,
5     before  => Service['logstash']
6   }
7
8   file { 'filebeat-input.conf':
9     path      => '/etc/logstash/conf.d/filebeat-input.conf',
10    source    => 'puppet:///modules/elastic_stack/filebeat-input.conf',
11    require   => Package['logstash'],
12  }
13
14   file { 'syslog-filter.conf':
15     path      => '/etc/logstash/conf.d/syslog-filter.conf',
16    source    => 'puppet:///modules/elastic_stack/syslog-filter.conf',
17    require   => Package['logstash'],
18  }
19
20   file { 'output-elasticsearch.conf':
21     path      => '/etc/logstash/conf.d/output-elasticsearch.conf',
22    source    => 'puppet:///modules/elastic_stack/output-elasticsearch.conf',
23    require   => Package['logstash'],
24  }
25
26   file_line {'JVM-logstash-XMS':
27     path      => '/etc/logstash/jvm.options',
28     line     => '-Xms512m',
29     match    => '-Xms1g',
30     notify   => Service['logstash']
31 }
```

```

32   file_line {'JVM-logstash-XMX':
33     path  => '/etc/logstash/jvm.options',
34     line   => '-Xmx512m',
35     match  => '-Xmx1g',
36     notify  => Service['logstash']
37   }
38
39   service {'logstash':
40     ensure  => running,
41     enable   => true,
42     require  => [ File['output-elasticsearch.conf'], File['filebeat-input.conf'] ]
43   }
44 }
45 }
```

Crie o arquivo filebeat.yml que será responsável pela parametrização da aplicação

```
1 vim elastic_stack/files/filebeat.yml
```

```

1 filebeat.inputs:
2 - type: log
3   enabled: true
4   paths:
5     - /var/log/*.log
6 filebeat.config.modules:
7   path: ${path.config}/modules.d/*.yml
8   reload.enabled: false
9 output.logstash:
10   hosts: ["log.4labs.example:5443"]
11 processors:
12   - add_host_metadata: ~
13   - add_cloud_metadata: ~
```

Crie o manifest para instalação do Filebeat

```
1 vim elastic_stack/manifests/filebeat.pp
```

```

1 class elastic_stack::filebeat {
2
3   package {'filebeat':
4     ensure  => installed,
5     before  => Service['filebeat']
6   }
7
8   file { 'filebeat.yml':
9     path    => '/etc/filebeat/filebeat.yml',
10    source  => 'puppet:///modules/elastic_stack/filebeat.yml',
11    require  => Package['filebeat'],
12    notify  => Service['filebeat']
13  }
14
15   service {'filebeat':
16     ensure  => running,
17     enable   => true,
18     require  => File['filebeat.yml']
19   }
20 }
```

Após escrever todos os manifests precisamos editar o site.pp para que sejam aplicados os manifests nos servidores corretos.

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

```

1 node 'log' {
2     include elastic_stack::java8
3     include elastic_stack::repo
4     include elastic_stack::nginx
5     include elastic_stack::elasticsearch
6     include elastic_stack::kibana
7     include elastic_stack::logstash
8 }
9
10 node 'compliance' {
11     include inspec
12     include elastic_stack::repo
13     include elastic_stack::filebeat
14 }
15
16
17 node 'default' {
18     include elastic_stack::repo
19     include elastic_stack::filebeat
20 }

```

## Executando a instalação do Puppet Agent

Após a criação de todos os arquivos necessários, precisamos instalar o puppet agent que será responsável por aplicar todos os manifests em nossa infraestrutura

Acesse a máquina **automation** e execute a playbook do puppet-agent

```

1 vagrant ssh automation
2 sudo ansible-playbook /etc/ansible/playbooks/puppet-agent.yml

```

Na máquina **compliance** verifique se o agent foi executado corretamente nas máquinas listando os certificados gerados e assinados pelo puppet

```

1 vagrant ssh compliance
2 sudo su -
3 puppetserver ca list --all

```

## Capítulo 8

# Construindo pipelines de Infraestrutura com Rundeck

### Características do Rundeck

O que é o Rundeck?



Figura 8.1: Rundeck

Rundeck é um gerenciador e orquestrador de tarefas e rotinas Open-Source orientada a plugins.

A ferramenta é muito utilizada pela comunidade DevOps e também em Data Centers, sendo possível conectá-la a ferramentas de ChatOps para que seja informado o estado de execuções de jobs.

### Características do Rundeck

- Execução de comandos de forma distribuída
- Workflow (incluindo condicionais, tratamento de erros e estratégias multiplas de workflows)
- Sistema de Execução Plugável (SSH e WinRM por padrão, Powershell via plugins)
- Modelo de Recursos Conectáveis (Coleta de detalhes de infraestrutura de sistemas externos)
- Execuções de tarefas Planejadas
- Painel Self-Service (Interface Gráfica via WEB)
- Armazenamento seguro para senhas e chaves

- Controle de acesso baseado em políticas (RBAC) com suporte para LDAP, Active Directory e SSO
- Ferramentas de edição de Políticas de Controle de Acesso
- Logs de Histórico e Auditoria
- Utilize qualquer linguagem de script

## Primeiros Passos

Abra o navegador e acesse o endereço <http://automation.4labs.example:4440> e efetue o login com usuário e senha admin.

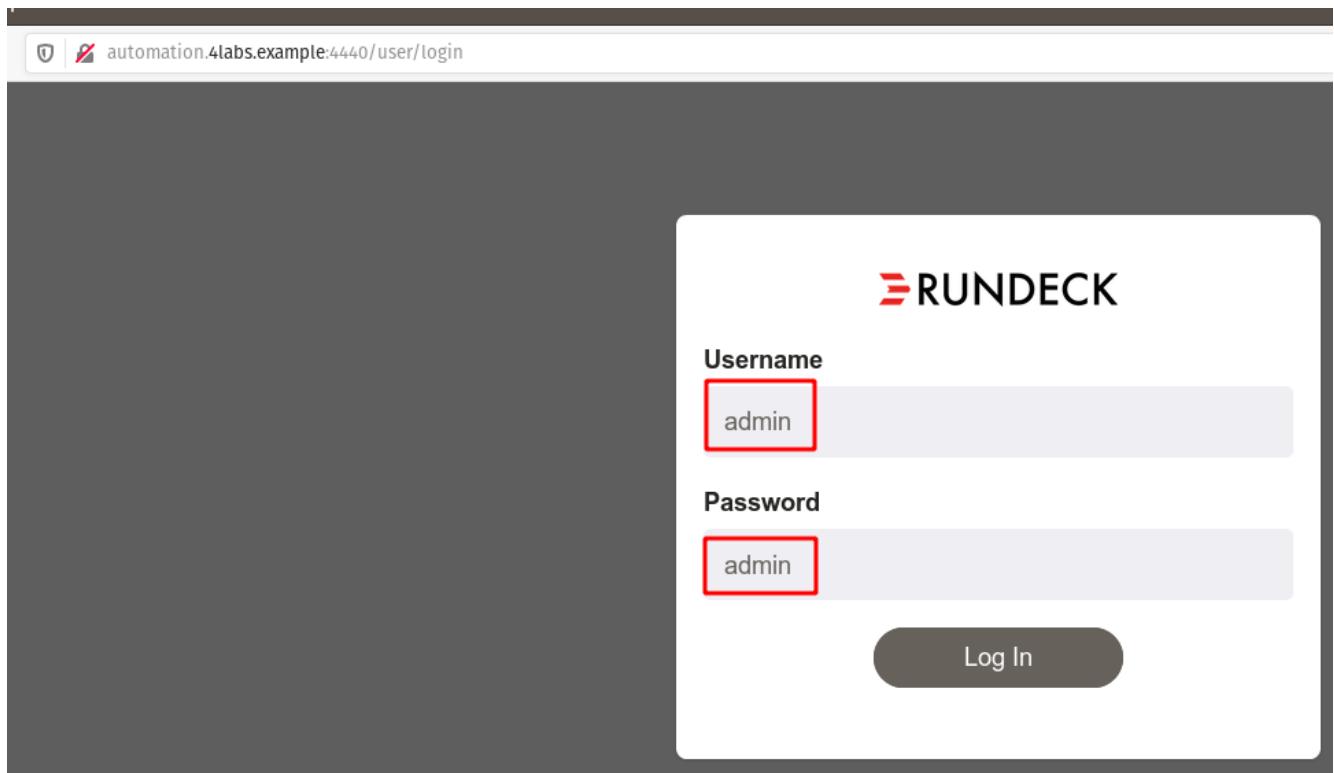


Figura 8.2: Rundeck Login

Vamos criar nosso primeiro projeto, para isto clique em **New Project +**

Preencha o nome do projeto, a descrição e clique em **Create**

Após a criação do projeto podemos adicionar tarefas (jobs) clicando em Jobs e em seguida em Create a New Job

Vamos chamar nosso primeiro job de **verify-hosts** e escrever uma descrição para o mesmo

Na aba **workflow** podemos adicionar um passo a ser executado

Clique em **Add a Step** e em seguida em **Command**

Preencha com o comando **cat /etc/hosts**, um nome para o passo a ser executado e em seguida clique em **Save**

Clique em **Create** para finalizar a criação do nosso Job

Será exibida a tela com o job criado, seleciona **Log Output** em Follow Execution e clique em **Run Job Now**

Após executar o job, será exibida uma tela com a atividade e o resultado da execução

Em **DASHBOARD** podemos verificar os status geral das execuções das jobs do nosso projeto

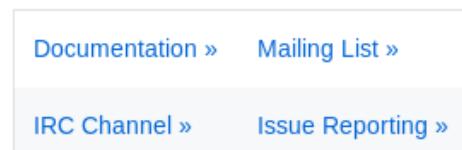
Em **ACTIVITY** podemos verificar as atividades de execução das tarefas, clique sobre a atividade para visualizar informações da mesma.

# Welcome to Rundeck 3.1.0-20190731



Get your [free Rundeck stickers](#).

For help see the links below, or click "help" in the top part of the page.



To get started, create a new project.



You can see this message again by clicking the version number in the page footer.

Figura 8.3: New Project

Serão exibidos alguns dados como percentual de acerto, falha e tempo de duração da execução, caso queira executar novamente, clique em **Run Again** e em seguida em **Run Job Now**

The screenshot shows the Rundeck interface with the title 'Primeiro-Projeto'. A job named 'verify-hosts' is listed, showing a success rate of 100% (1/1 complete), 0 failed steps, and 0 incomplete steps. The job was run by the user 'you' at 5:17 pm, with a duration of 0.00:01. There is a 'Run Again' button with a red arrow pointing to it. Below the job details, there are sections for 'Stats' and 'Activity', showing 1 execution and a 100% success rate with an average duration of 4s.

## Create a new Project

Details   Execution History   Clean   Execution Mode   User Invitations

---

**Project Name**

Primeiro-Projeto

**Label**

**Description**

Meu primeiro projeto com Rundeck

**Cancel** **Create** 

Figura 8.4: New Project

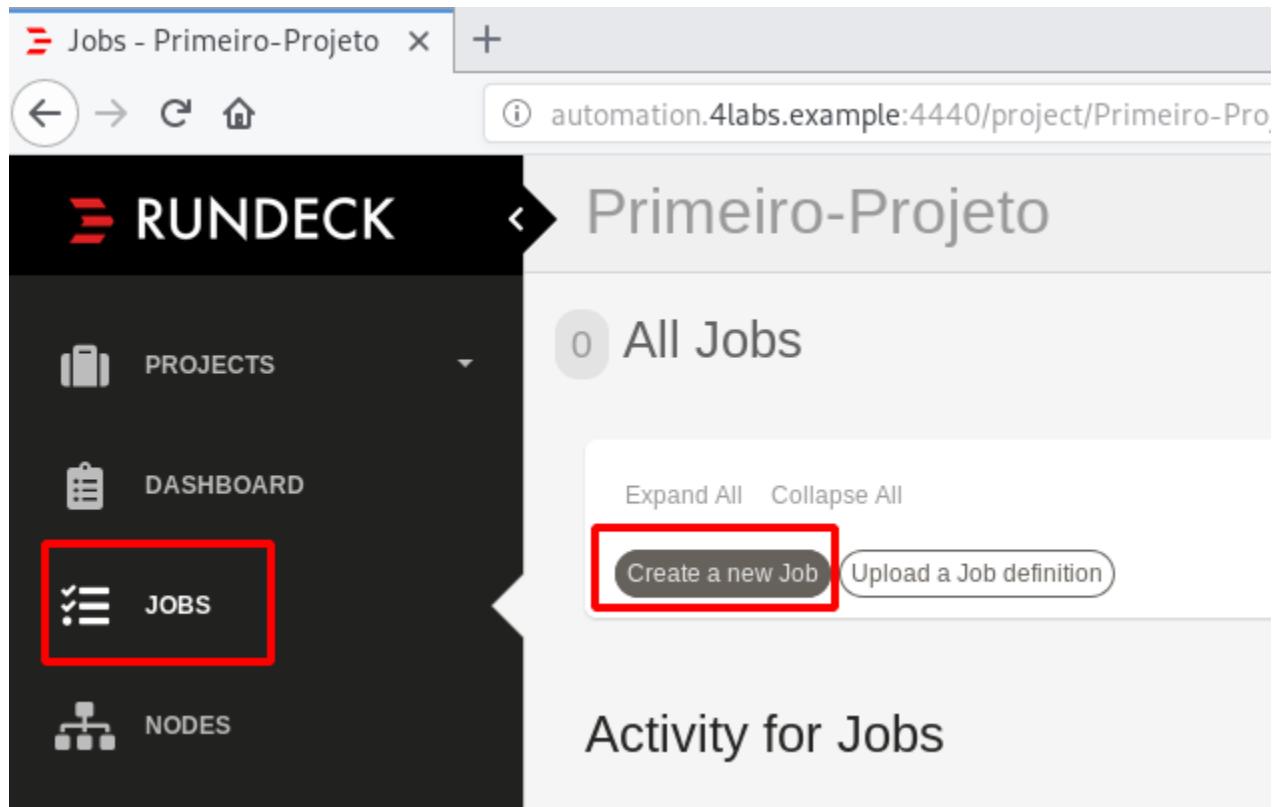


Figura 8.5: New Job

Create New Job: verify-hosts

Details   Workflow   Nodes   Schedule   Notifications   Other

**Job Name**  Group is a / separated path

**Description**   
1 Projeto para verificação dos hosts da máquina.

The first line of the description will be shown in plain text, the rest will be rendered with Markdown. [More...](#)

Figura 8.6: New Job

# ▶ Primeiro-Projeto

Create New Job: verify-hosts

Workflow

Details Workflow Nodes Schedule Notifications Other

**Options**

No Options

---

**Workflow** If a step fails:

Stop at the failed step.  Run remaining steps before failing.

Strategy: Node First

Execute all steps on a node before proceeding to the next node.

Explain >

---

Global Log Filters

Figura 8.7: New Job

No Workflow steps

## Add a Step

Click on a Step type to add.

Search step  ? Search

Node Steps Workflow Steps

**Runs once for each node in the workflow.**

- Command - Execute a remote command (Selected)
- Script - Execute an inline script
- Script file or URL - Execute a local script file or a script from a URL
- Job Reference - Execute another Job for each Node

Figura 8.8: New Job

1.

<b>Command</b>	cat /etc/hosts
<b>Step Label</b>	Lendo o conteúdo do arquivo Hosts
<span>Cancel</span>	<span>Save</span> <span style="color: red; font-size: 2em;">→</span>

No Workflow steps

Figura 8.9: New Job

**Workflow** If a step fails:

Stop at the failed step.  Run remaining steps before failing.

**Strategy:** Node First

Execute all steps on a node before proceeding to the next node.

Explain >

---

**Global Log Filters** (+ add)

---

( Undo Redo Revert All Changes)

1.

cat /etc/hosts  
Lendo o conteúdo do arquivo Hosts

(gear, X, trash, info)

+ Add a step

(Cancel) (Create)

Figura 8.10: New Job

## Primeiro-Projeto

verify-hosts

Projeto para verificação dos hosts da máquina.

bd0fd041-ba02-4699-8234-65ff67937216

Action ▾ Definition ▾

Follow execution Log Output ▾

Run Job Now ▾

Stats Activity

0 EXECUTIONS - AVG DURATION

Figura 8.11: New Job

The screenshot shows the Rundeck interface with a project titled "Primeiro-Projeto". A job named "verify-hosts" has been run successfully. The log output shows the contents of the "/etc/hosts" file, listing various local and public IP addresses mapping to hostnames like "automation.4labs.example" and "localhost".

```

17:17:21 [x] 1. Lendo o conteúdo do arquivo Hosts 127.0.0.1      automation.4labs.example      automation
17:17:21                               127.0.0.1      localhost      localhost.localdomain      localhost4
17:17:21                               localhost4.localdomain4
17:17:21                               ::1          localhost      localhost.localdomain      localhost6
17:17:21                               localhost6.localdomain6
17:17:21                               10.5.25.10    automation.4labs.example
17:17:21                               10.5.25.20    compliance.4labs.example
17:17:21                               10.5.25.30    container.4labs.example
17:17:21                               10.5.25.40    scm.4labs.example
17:17:21                               10.5.25.50    log.4labs.example

```

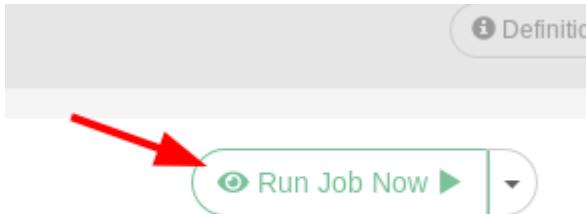
Figura 8.12: New Job

The screenshot shows the Rundeck dashboard for the "Primeiro-Projeto" project. The "DASHBOARD" tab is selected. It displays a summary of recent executions, showing 1 execution in the last day, completed successfully by user "admin" for the job "verify-hosts".

Figura 8.13: DASHBOARD

The screenshot shows the Rundeck activity log for the "Primeiro-Projeto" project. The "ACTIVITY" tab is selected. It lists the execution of the "verify-hosts" job on 08/19/2019 at 5:17 PM, which was successful ("1 ok") and took "a few seconds". A red arrow points to this specific execution entry.

Figura 8.14: Activity



Clicando novamente em **Activity** agora veremos duas execuções do nosso job **verify-hosts**

Date	Status	Duration	Run By	Job Name	ID
08/19/2019 5:23 PM	1 ok	a few seconds	by admin	verify-hosts	#2
08/19/2019 5:17 PM	1 ok	a few seconds	by admin	verify-hosts	#1

Figura 8.15: Job

## Criando Usuários

A criação de usuários é feita via CLI, para isto iremos acessar a máquina automation e editar o arquivo **realm.properties**

```
1 vagrant ssh automation
2 sudo vim /etc/rundeck/realm.properties +$
```

```
1 analista:devops@4linux,user,admin,architect,deploy,build
```

Reinic peace o serviço **rundeckd** para aplicar as alterações

```
1 sudo systemctl restart rundeckd
```

Acesse novamente o rundeck pelo navegador em <http://automation.4labs.example:4440> e efetue o login com o usuário **analista** e a senha **devops@4linux**

Para listar os usuários clique na engrenagem no canto superior direito da tela e em seguida em User Summary

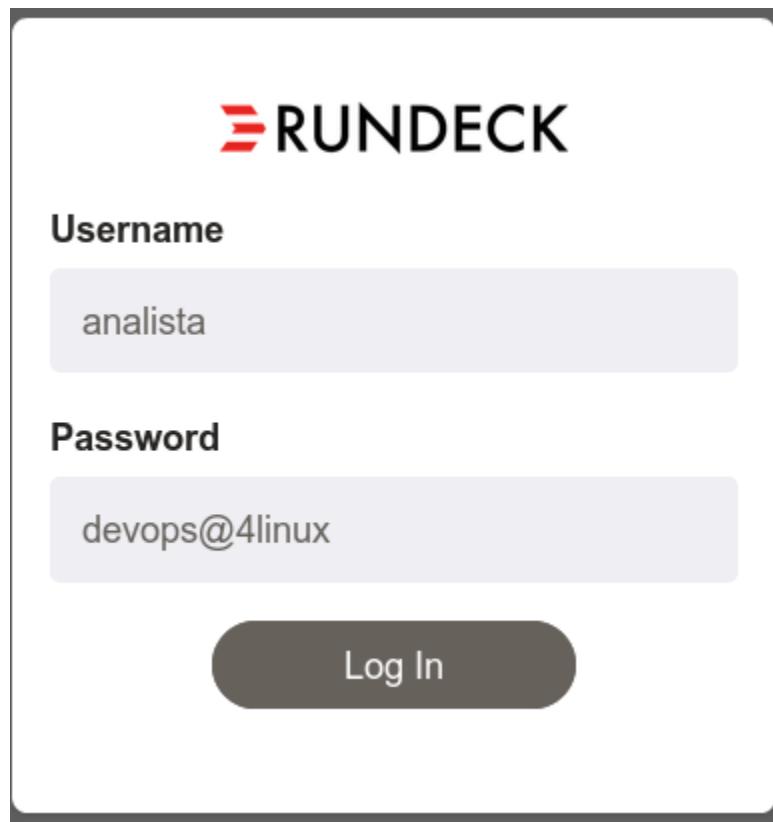


Figura 8.16: Login

A screenshot of the Rundeck application interface. On the left is a navigation sidebar with the number "2" at the top. It contains the following items: System (with a red arrow pointing to it), Key Storage, System Configuration, Access Control, System Report, Log Storage, Plugins (with a red arrow pointing to it), Password Utility, User Summary (highlighted with a red arrow and bolded), and Execution Mode...  
  
The main content area shows a table titled "List of Rundeck users." with the following data:

LOGIN	EMAIL	FIRST NAME	LAST NAME	CREATED	UPDATED
admin	NOT SET	NOT SET	NOT SET	11/06/2019 06:15 PM	11/06/2019 06:15 PM
analista	NOT SET	NOT SET	NOT SET	11/12/2019 09:07 PM	11/12/2019 09:07 PM

109

## Registrando e Verificando Nodes

### O que é um node

Node (Nó) é um recurso que pode ser uma máquina física, virtual ou até mesmo um host acessível pela rede.

No Rundeck podemos adicionar outras máquinas como nodes de um projeto para que elas possam executar determinadas ações.

### Registrando Nodes

Primeiramente iremos criar um projeto para anossa infraestrutura ágil, acesse o endereço <http://automation.4labs.example:4440> e efetue o login como usuário **analista** e senha **devops@4linux**

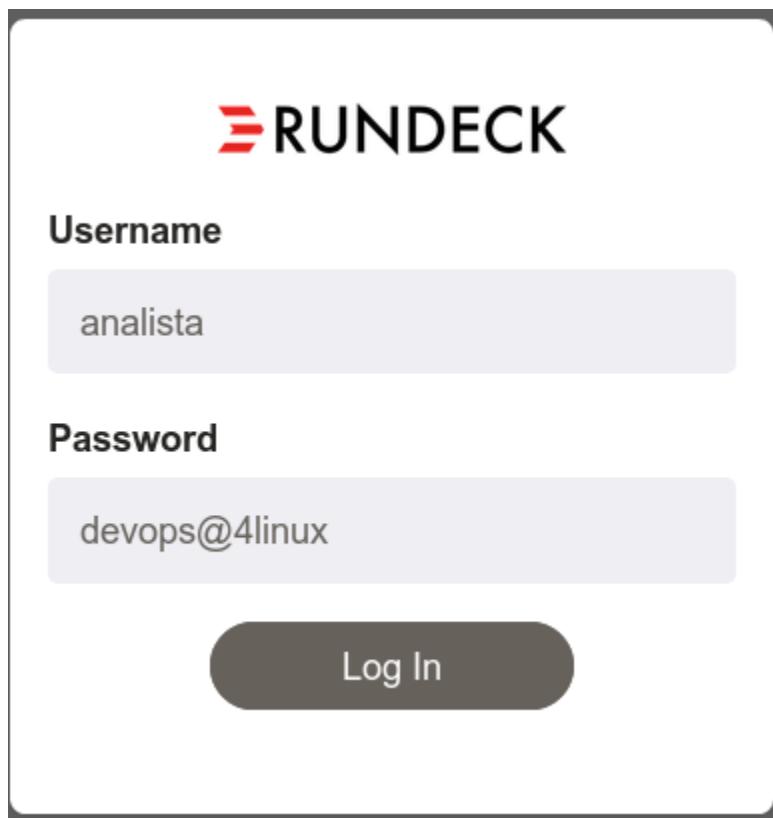
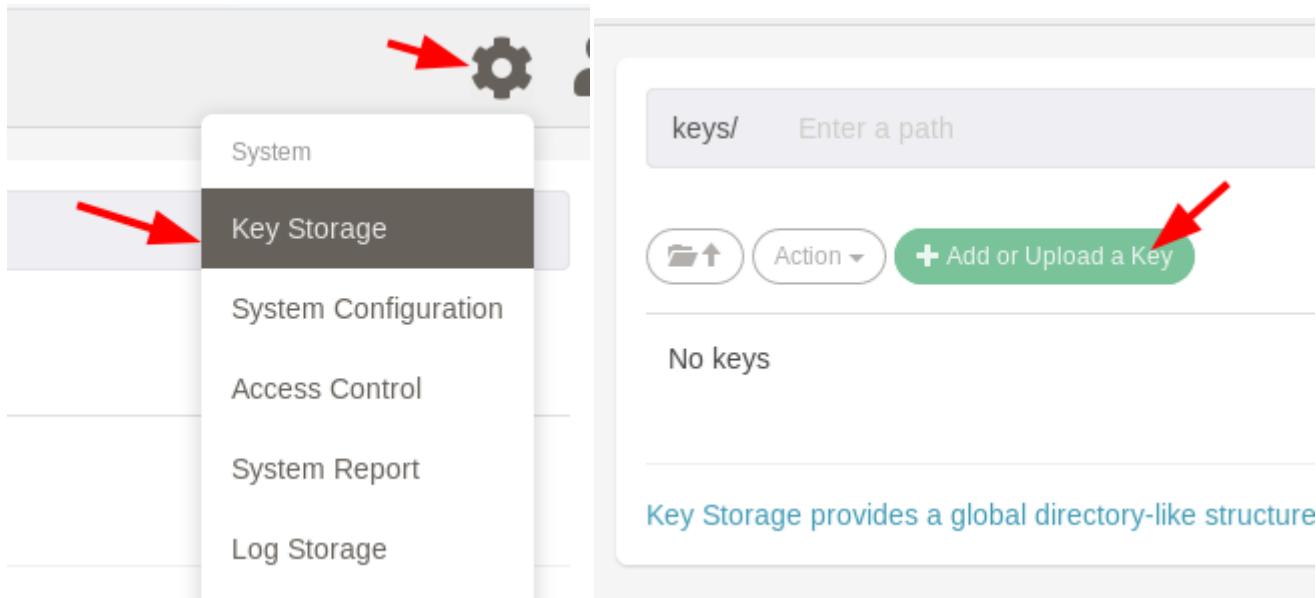


Figura 8.17: Login

Precisaremos adicionar as chaves de acesso ao nosso sistema, para isto clique na engrenagem no canto superior direito e em seguida em **key storage** e **Add or Upload a Key**



Iremos copiar a chave privada de acesso utilizada pelo ansible e adiciona-la ao Rundeck.

Acesse a máquina automation e copie o conteúdo da chave id\_rsa

```
1 vagrant ssh automation
2 sudo cat /root/.ssh/id_rsa
```

Altere a opção de tipo da chave para **Private Key**, cole o conteúdo da chave id\_rsa no campo ao lado da opção **Enter text** e coloque o nome da chave de **devops** e clique em **save**

Como resultado a chave estará criada, porém não será possível visualizar mais seu conteúdo, apenas sobrescreve-la

Clique em **Projects** e em seguida em **New Project**

Vamos chamar nosso projeto de **InfraAgil**, preencher o label com **525** e adicionar uma descrição

Em **Default Node Executor** altere **SSH Key Storage Path** para **keys/devops** e clique em **Create**

Clique agora em **Projects** e selecione **525** para acessar o projeto.

Em seguida clique em **Project Settings** e em **Edit Nodes**

Clique em **Add a new Node Source +** para adicionar um novo recurso

Vamos adicionar nosso node a partir de um arquivo, clique na opção **File**

Selecione o formato como **resourceyaml**, digite o caminho do arquivo **/var/lib/rundeck/projects/525/nodes.yml**, selecione as opções **Generate**, **Require File Exists** e **Writable** e clique em **Save**

Na próxima tela será necessário clicar em **Save** novamente para confirmar a criação do arquivo de configuração

Iremos remover o node local clicando em **delete** e em seguida clicaremos em **Save**

## Overwrite a Key

X

Key Type:

Private Key

Private Keys and Passwords are not available to download once they are stored. Public keys can be downloaded.

Enter text

```
-----BEGIN RSA PRIVATE KEY-----  
MIQwEAAQIBAAQDf...  
-----END RSA PRIVATE KEY-----
```

Storage path:

keys/ Enter the directory name

Name:

devops

Do not overwrite a file with the same name.

You can reference this stored Key using the storage path:

[keys/devops](#)

Cancel

Save

Figura 8.18: Key

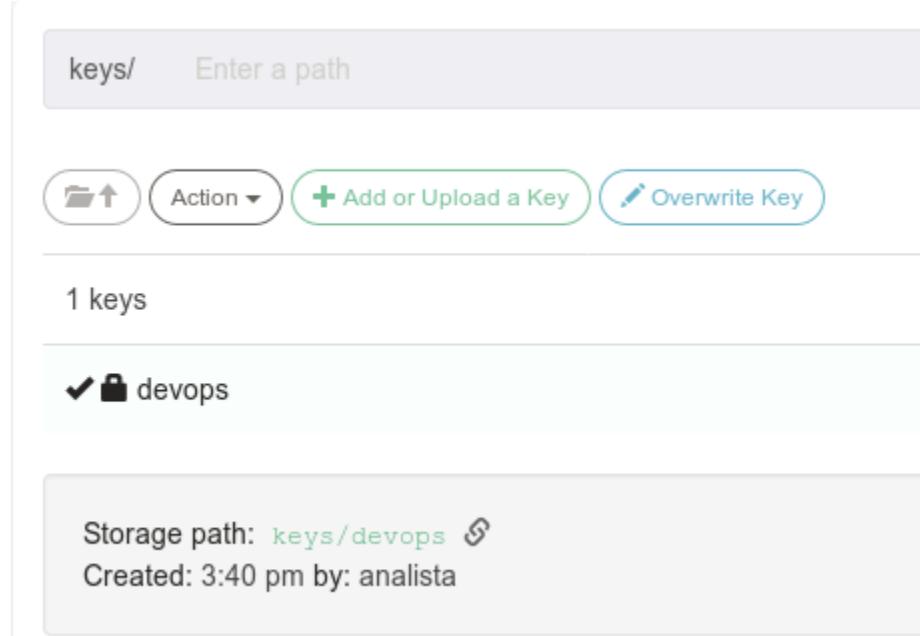


Figura 8.19: Key

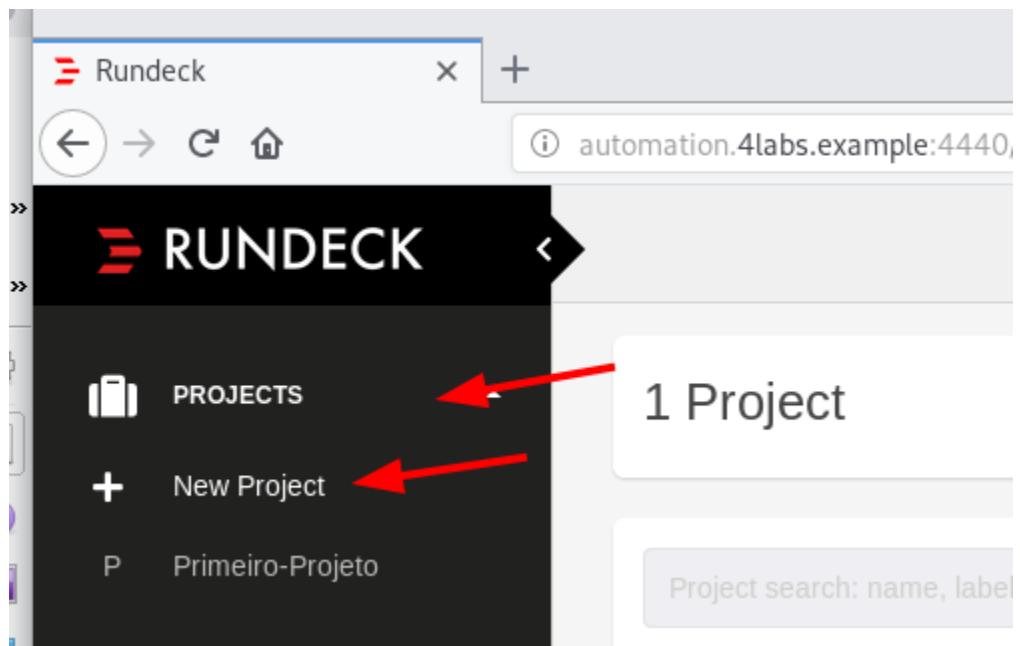


Figura 8.20: New Project

Create a new Project

Details   Execution History Clean   Execution

---

**Project Name**

InfraAgil

**Label**

525

**Description**

Infraestrutura Ágil com Práticas DevOps

**Buttons:** Cancel, Create

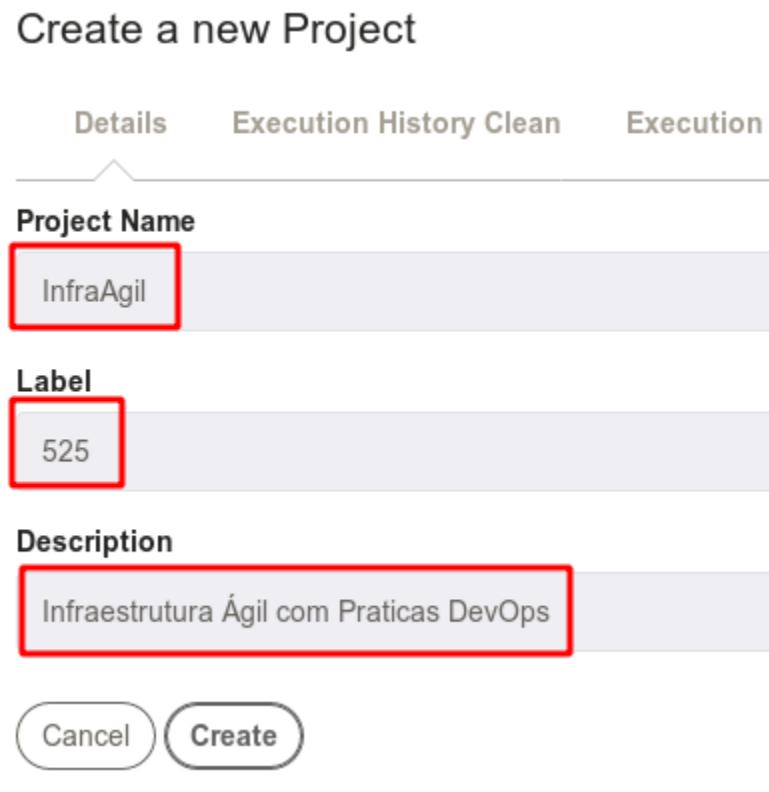


Figura 8.21: New Project

Details   Execution History Clean   Execution Mode   User Interface   **Default Node Executor**   Default File Copier

---

The Node Executor is responsible for executing commands and scripts on remote nodes.

▼ ♦ SSH   Executes a command on a remote node via SSH.

**SSH Key File path** /var/lib/rundeck/.ssh/id\_rsa  
File Path to the SSH Key to use

**SSH Key Storage Path** keys/devops   **Select...** 

Path to the SSH Key to use within Rundeck Storage. E.g. "keys/path/key1.pem"

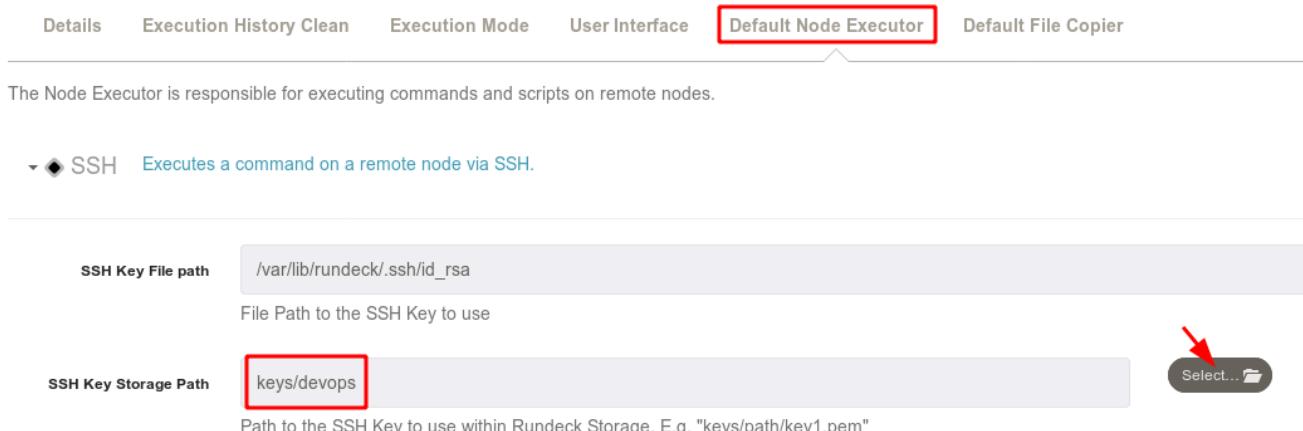


Figura 8.22: New Project

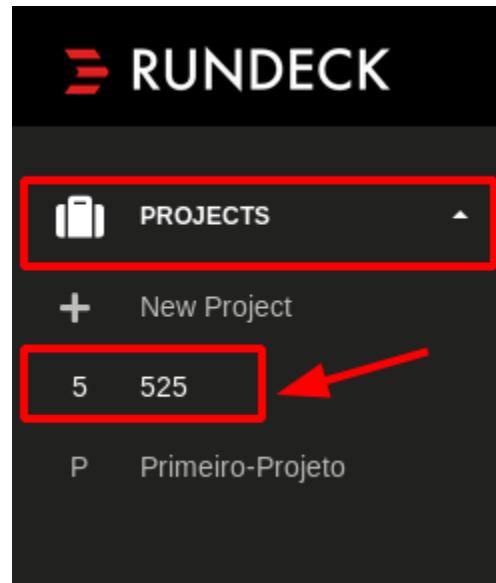


Figura 8.23: Project

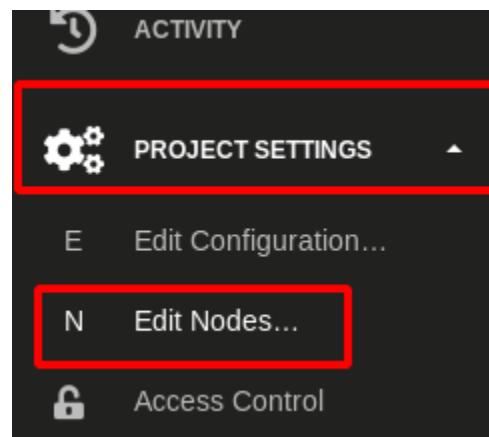


Figura 8.24: Project Settings

## 525 | Edit Nodes

 Edit

 Sources

 Enhancers

 Conf

Node Sources for the project. Sources are loaded in the defined order and values to substitute the project name.)

1.  Local Provides the local node as the single resource

 Edit

Add a new Node Source 

Figura 8.25: New Node

### Add a new Node Source

#### AWS S3 remote model source

Obtain nodes information from a file located in a S3 bucket

#### Ansible Resource Model Source

Imports nodes from Ansible's inventory.

#### Directory

Scans a directory and loads all resource document files

#### File

Reads a file containing node definitions in a supported format

Figura 8.26: New Node

2. File Reads a file containing node definitions in a supported format

**Format** resourceyaml **resourceyaml**

Format of the file. If unspecified, the format will be determined by the file extension.

**File Path** /var/lib/rundeck/projects/525/nodes.yml

Path of the file

**Generate**  
Automatically generate the file if it is missing? Also creates missing directories.

**Include Server Node**  
Automatically include the server node in the generated file?

**Require File Exists**  
Require that the file exists

**Writable**  
Allow this file to be editable.

**Save** **Cancel**

Figura 8.27: New Node

Edit    Sources    Enhancers    Configuration

---

Node Sources for the project. Sources are loaded in the defined order, with late use \${project.name} inside configuration values to substitute the project name.)

---

1. Local Provides the local node as the single resource  
[Edit](#)

---

2. File Reads a file containing node definitions in a supported format  
Format: `resourceyaml`  
File Path: `/var/lib/rundeck/projects/525/nodes.yml`  
Generate: `yes`  
Require File Exists: `yes`  
Writeable: `yes`  
[Edit](#)  Modified

---

[Add a new Node Source +](#)

[Revert](#) [Save](#)  Changes have not been saved.

Figura 8.28: New Node

Node Sources for the project. Sources are loaded in the defined order, with later sources overriding earlier sources. (You can use \${project.name} inside configuration values to substitute the project name.)

1. Local Provides the local node as the single resource

Node Sources for the project. Sources are loaded in the defined order, with later sources overriding earlier sources. (You can use \${project.name} inside configuration values to substitute the project name.)

You have unsaved changes: Sources

1. File Reads a file containing node definitions in a supported format

Format: `resourceyaml`

File Path: `/var/lib/rundeck/projects/525/nodes.yml`

Generate: `yes`

Require File Exists: `yes`

Writeable: `yes`

Add a new Node Source

Changes have not been saved.

Clique em **Edit** e em seguida em **Modify** para adicionar as configurações dos nodes

Edit

Sources

Enhancers

Configuration

2. File `/var/lib/rundeck/projects/525/nodes.yml`

Format: `text/yaml`

Modify

Figura 8.29: New Node

Adicione os dados para conexão dos nodes e clique em **Save**

```
1 automation:
2   nodename: automation
3   hostname: automation.4labs.example
4   description: Automation Node
5   username: root
6   ssh-key-storage-path: keys/devops
7
8 compliance:
9   nodename: compliance
10  hostname: compliance.4labs.example
11  description: Compliance Node
12  username: root
13  ssh-key-storage-path: keys/devops
14
15 container:
16   nodename: container
17   hostname: container.4labs.example
18   description: Container Node
19   username: root
20   ssh-key-storage-path: keys/devops
21
22 scm:
23   nodename: scm
24   hostname: scm.4labs.example
25   description: SCM Node
26   username: root
27   ssh-key-storage-path: keys/devops
28
29 log:
30   nodename: log
31   hostname: log.4labs.example
32   description: LOG Node
33   username: root
34   ssh-key-storage-path: keys/devops
```

## Verificando os Nodes

Para visualizar os nodes registrados, clique em **Nodes** e em seguida em **All Nodes**

The screenshot shows the Jenkins dashboard. On the left, there are two main navigation items: 'JOBS' and 'NODES'. The 'NODES' item is highlighted with a red box and a red arrow pointing to it. In the center, there's a 'TAGS' section and a 'FILTERS' section. The 'FILTERS' section has a dropdown menu set to 'All Nodes 5'. Below these sections, there are three tables: 'NODE', 'TAGS', and 'USER @ HOST'. The 'NODE' table lists five nodes: 'automation' (Automation Node), 'compliance' (Compliance Node), 'container' (Container Node), 'log' (LOG Node), and 'scm' (SCM Node). The 'TAGS' table shows no tags assigned to the nodes. The 'USER @ HOST' table shows the user 'root' with hostnames: 'automation.4labs.example', 'compliance.4labs.example', 'container.4labs.example', 'log.4labs.example', and 'scm.4labs.example'.

NODE	TAGS	USER @ HOST
> automation (Automation Node)		root @ automation.4labs.example
> compliance (Compliance Node)		root @ compliance.4labs.example
> container (Container Node)		root @ container.4labs.example
> log (LOG Node)		root @ log.4labs.example
> scm (SCM Node)		root @ scm.4labs.example

Para testar uma execução, podemos clicar em **Commands**, digitar o filtro .\* clicar em search para selecionar todos os nodes e executar um comando como por exemplo **cat /etc/hosts** clicando em **Run on 5 Nodes**

A execução atualizará a tela com o resultado dos comandos

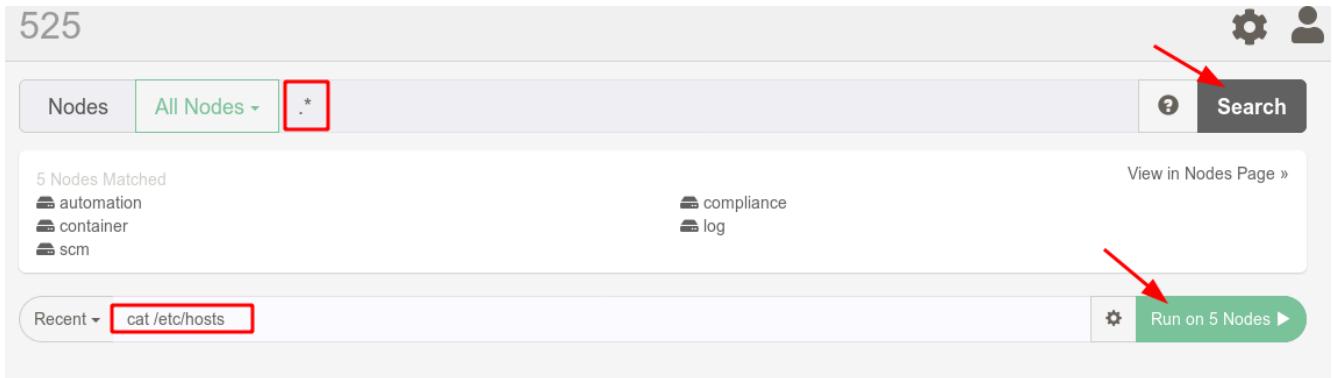


Figura 8.30: Run Command

The screenshot shows the execution results for the command 'cat /etc/hosts'. It lists the output from five nodes. A red box highlights the command 'cat /etc/hosts' in the recent history. To the right, there are buttons for 'Save as a Job' and a status message '#3 Succeeded' with a red box around it. The log output shows the contents of the /etc/hosts file for each node, including IPv4 and IPv6 entries for automation, compliance, container, scm, and log services.

```

16:17:07 127.0.0.1      automation.4labs.example      automation
16:17:07 127.0.0.1      localhost localhost.localdomain localhost4 localhost4.localdomain4
16:17:07 ::1            localhost localhost.localdomain localhost6 localhost6.localdomain6
16:17:07 10.5.25.10     automation.4labs.example
16:17:07 10.5.25.20     compliance.4labs.example
16:17:07 10.5.25.30     container.4labs.example chat.4labs.example
16:17:07 10.5.25.40     scm.4labs.example
16:17:07 10.5.25.50     log.4labs.example
16:17:11 127.0.0.1      localhost
16:17:11 # The following lines are desirable for IPv6 capable hosts
16:17:11 ::1      ip6-localhost   ip6-loopback
16:17:11 fe00::0 ip6-localnet
16:17:11 ff00::0 ip6-mcastprefix

```

Figura 8.31: Command Execution

## Consruindo Jobs

### O que são Jobs?

Jobs (Trabalhos/Tarefas) são atividades criadas no rundeck com o fim de encapsular um processo.

Um Job é uma configuração representando opções de entrada, os passos de um processo, uma expressão de filtro que seleciona os nós onde as tarefas serão executadas e os parâmetros que especificam como os passos serão executados.

O Rundeck permite que você organize, execute e observe o progresso a medida que um Job está sendo executado.

### Características de um Job

Primeiramente iremos acessar nosso projeto 525 para nossa Infraestrutura Ágil, acesse o endereço <http://automation.4labs.example:4440> e efetue o login com o usuário **analista** e senha **devops@4linux**

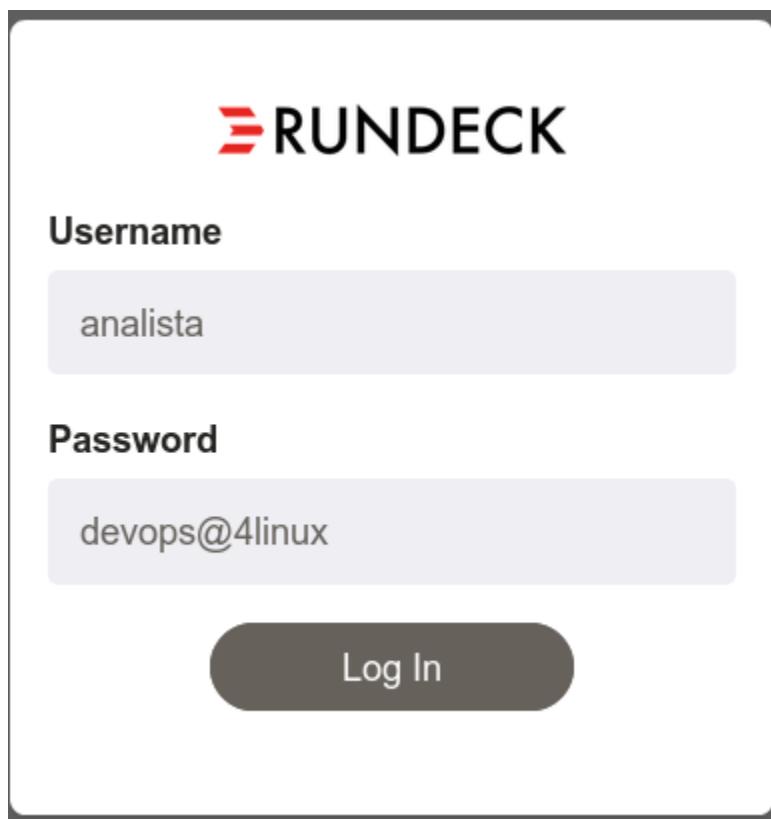


Figura 8.32: Login

Para acessar o projeto podemos clicar no menu **PROJECTS** e em seguida em **525**, ou ir ao dashboard e clicar diretamente no projeto

Para visualizar as tarefas, podemos clicar em **Jobs**, será exibido um dashboard com todos os jobs do projeto bem como as atividades do mesmo. Clique em **Create a New Job**

Os Jobs tem suas definições distribuidas em 6 menus.

**Details** - Menu onde é possível dar um nome, grupo e descrição a um job.

- **Job Name** - Nome a ser dado ao Job;
- **Group** - Campo utilizado para criar agrupamentos de Jobs;
- **Description** - Descrição do Job.

**Workflow** - Menu onde são definidas as tarefas a serem executadas dentro de um job e suas opções.

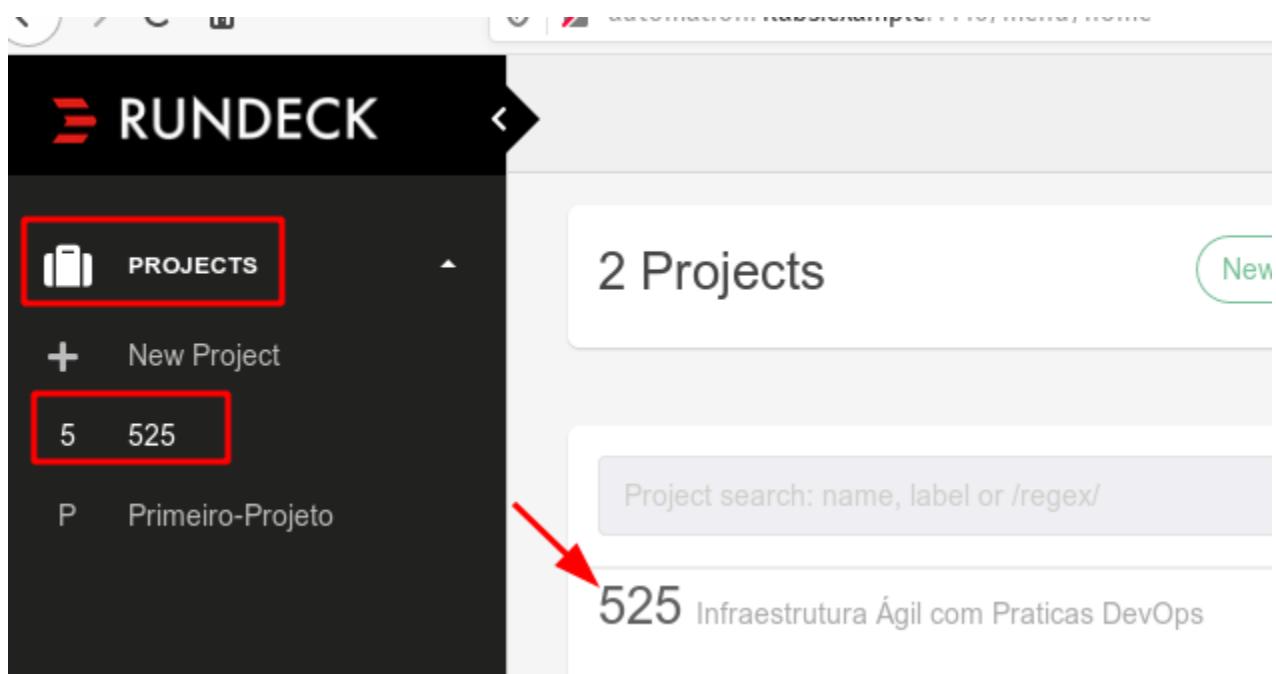


Figura 8.33: Project

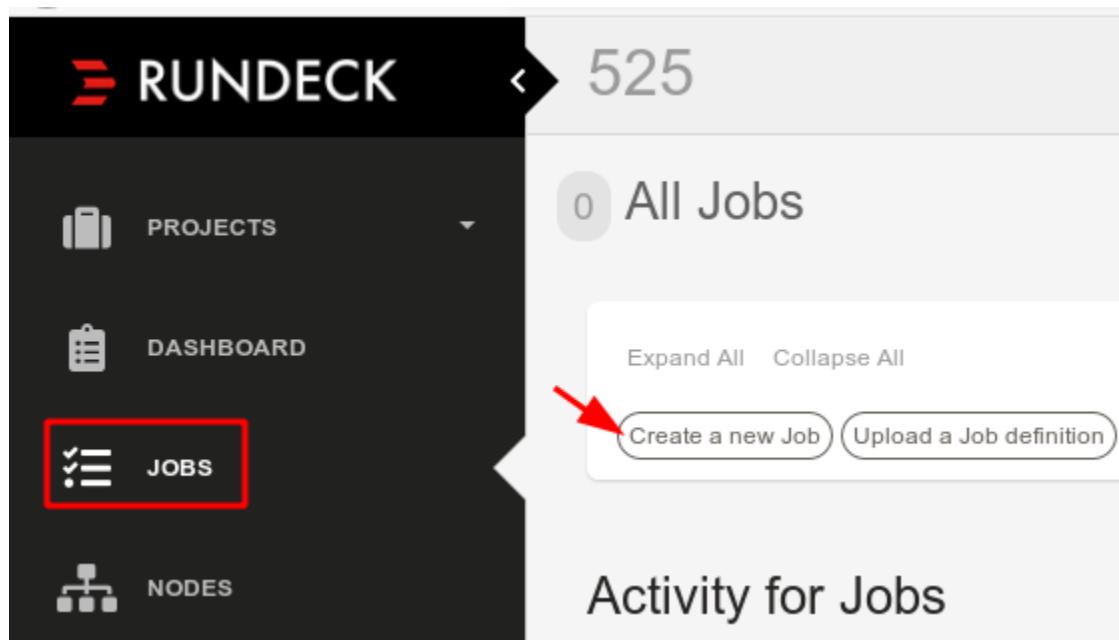


Figura 8.34: Jobs

## Create New Job

The screenshot shows a user interface for creating a new job. At the top, there is a navigation bar with tabs: Details, Workflow, Nodes, Schedule, Notifications, and Other. The 'Details' tab is highlighted with a red border. Below the tabs, there is a large input field labeled 'Job Name' where the text 'Job Name' is typed. The background of the interface is white, and the overall design is clean and modern.

Figura 8.35: Jobs

- **Options** - Neste campo podemos definir perguntas a serem feitas ao usuário no momento da execução do job, podemos utilizar por exemplo para alterar alguma variável que será lida pelo script ou até mesmo definir que o job só será executado se o usuário digitar uma determinada palavra;
- **Workflow** - Definição de como o job será executado e qual será sua estratégia de execução;
- **Global Log Filters** - Definições de filtragem de log, podemos definir por exemplo que determinadas palavras serão realçadas no log;
- **Workflow Steps** - São as ações propriamente ditas que serão executadas durante nosso Job.

**Nodes** - Definições de quais nodes executarão as tarefas do workflow do nosso job.

- **Nodes** - Definição de quais nodes executarão as Tarefas;
- **Show Excluded Nodes** - Exibe os nós na execução do job, mesmo se não houver nenhuma tarefa atrelada ao mesmo.

**Schedule** - Configurações de Agendamento do Job

- **Schedule to run repeatedly** - Define se o Job será executado outras vezes e como será a repetição;
- **Enable Scheduling** - Habilita o Job para ser agendado;
- **Enable Execution** - Habilita a execução do Job.

**Notifications** - Envio de notificações do Job \* **Send Notification** - Define se o job enviará notificações e em quais casos; \* Casos Possíveis: \* Sucesso \* Falha \* Início \* Tempo de Duração Superior \* Job falhou e executou novamente

**Other** - Outras configurações

- **Log Level** - Define o nível de log do Job;
- **Multiple Executions** - Habilita o job para ser executado mais de uma vez simultaneamente;
- **Timeout** - Define o tempo máximo de execução do job;
- **Retry / Retry Delay** - Define a quantidade máxima de tentativas e o tempo entre tentativas;
- **Log Output Limit / Log Limit Action** - Tratativas do tamanho máximo de linhas do log;
- **Default Tab** - Define a aba padrão a ser exibida quando executar um job.
- **UUID** - Definição Manual de Identificação única do Job

## Criando Jobs

Vamos criar alguns jobs para nossa infraestrutura. Clique em Details e defina o nome do job para **Verificando Hosts** e em seguida adicione uma descrição

Em **Workflow** vamos adicionar um passo de **comando** para executar a leitura dos hosts

Details	Workflow	Nodes	Schedule	Notifications	Other
<p><b>Job Name</b> Verificando Hosts</p> <hr/> <p><b>Description</b></p> <div style="border: 1px solid #ccc; padding: 5px; width: fit-content;"> <span style="border: 1px solid #ccc; padding: 2px 5px;">Edit</span>          1 Job para verificação do /etc/hosts         </div>					

The first line of the description will be shown in plain text, t

Figura 8.36: Job

**Search step** Enter a step filter string.

---

Node Steps
Workflow Steps

**Runs once for each node in the workflow.**

Command - Execute a remote command

Script - Execute an inline script

1.

```
> cat /etc/hosts
Lendo o conteúdo do arquivo Hosts
```

+ Add a step

Em **Nodes** vamos marcar a opção Dispatch to Nodes, adicionar o filtro .\* e clicar em **Search**

Altere tambpém as opções de falha para continuar executando antes de falhar um passo

Em **Schedule** vamos programar o job para ser executado todo dia as 0h e ajustar o timezone para **America/Sao\_Paulo**

## Nodes

Dispatch to Nodes  Execute locally

Choose whether the Job will run on filtered nodes or only on the local node.

### Node Filter

### Exclude Filter

Figura 8.37: Job

### If a node fails

- Fail the step without running on any remaining nodes.  
 Continue running on any remaining nodes before failing the step.

Figura 8.38: Job

Details Workflow Nodes Schedule Notifications Other

Schedule to run repeatedly?

No  Yes

Simple Crontab

00 : 00

Every Day

Every Month

Time Zone

America/Sao\_Paulo

A valid Time Zone, either an abbreviation such as "PST", a full name

Figura 8.39: Job

Em **Other** vamos alterar a **Default Tab** para **Log Output** e clicar em **Create**

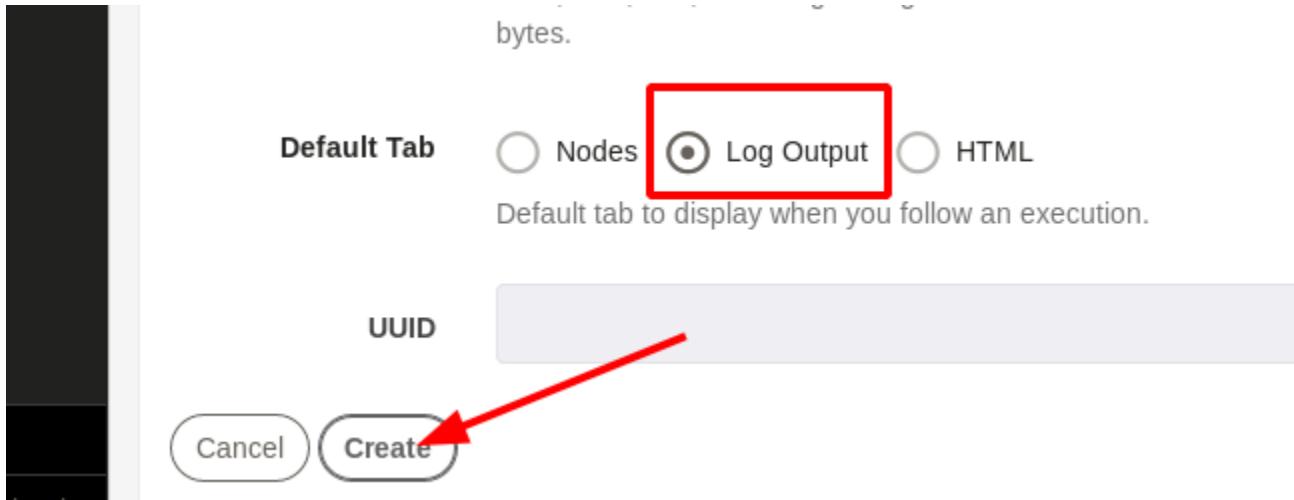


Figura 8.40: Job

Caso seja necessário a execução instantânea do Job basta clicar em **Run Job Now**

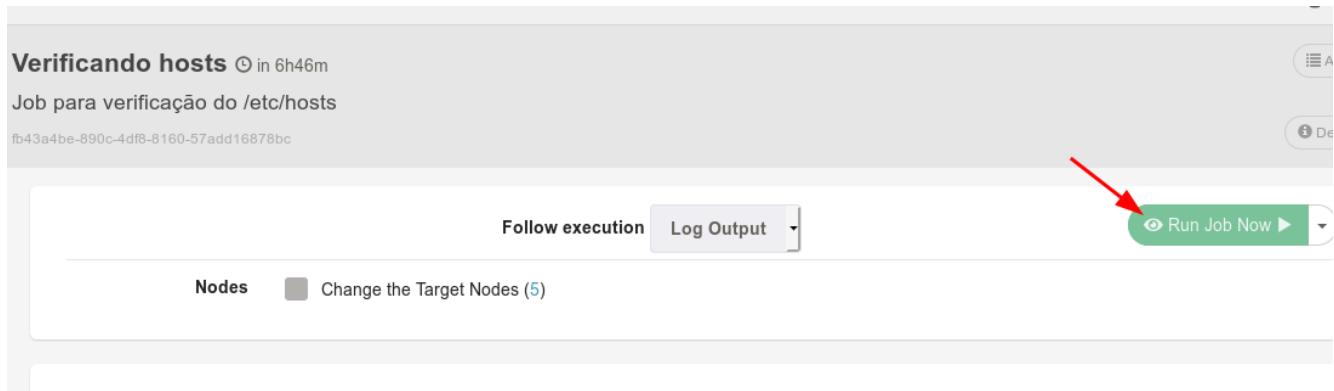


Figura 8.41: Job

Após a execução será exibida a tela com o log output do Job executado

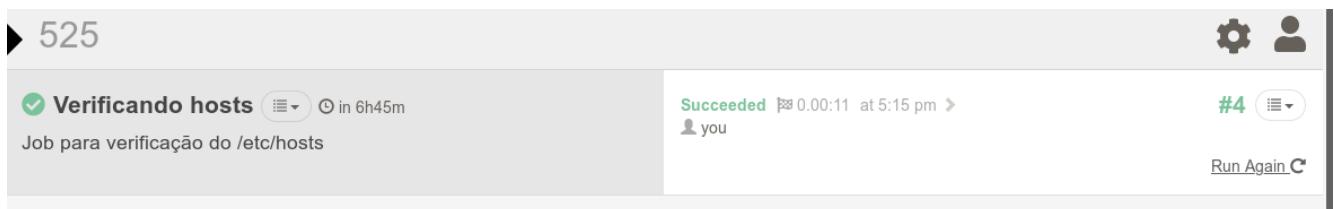


Figura 8.42: Job

Para visualizar o histórico de Jobs clique em **ACTIVITY**, nesta tela é possível escolher qual Job queremos visualizar.

Clicando em **JOBS** também podemos visualizar as atividades dos Jobs bem como em quanto tempo será executado cada job agendado

Podemos também criar um novo Job clicando **Job Actions** e em seguida em **New Job**

Vamos criar um job simples para verificar se o DNS da Google está acessível

Em workflow adicione um passo para executar o comando `ping -c4 8.8.8.8` e clique em Save

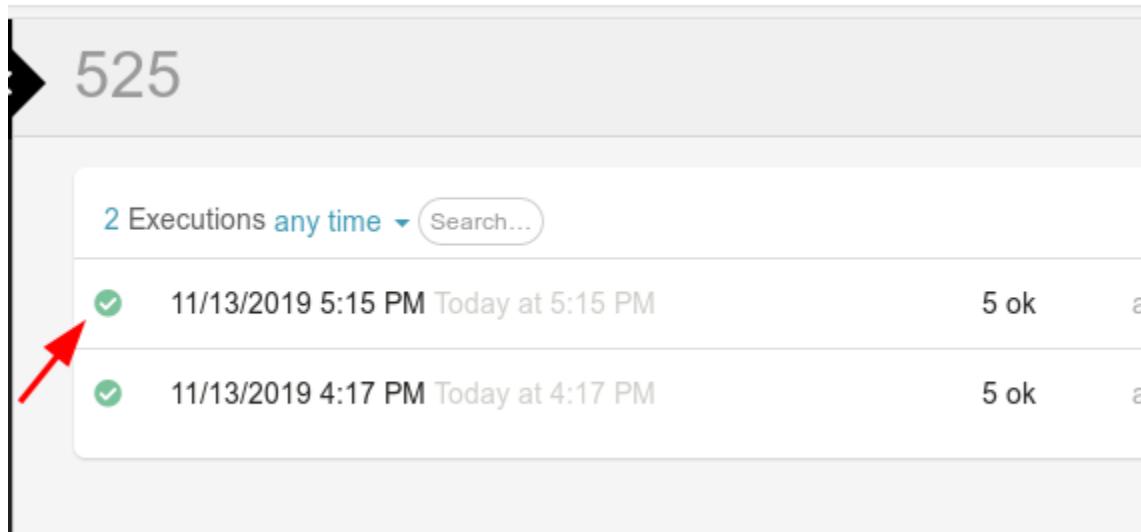


Figura 8.43: Job

A screenshot of a detailed job history view. At the top left is a large number '525'. The main title is 'Verificando hosts' with a status of 'Succeeded' and a duration of '0.00:11 at 5:15 pm'. The job description is 'Job para verificação do /etc/hosts'. On the right, there are buttons for '#4' and 'Run /'. Below this, there is a 'Log Output' button. The results section shows the following data:

Category	Status	Count	Start Time	Duration
100% 5/5 COMPLETE	COMPLETE	5		
0 FAILED	FAILED	0		
0 INCOMPLETE	INCOMPLETE	0		
0 NOT STARTED	NOT STARTED	0		

Node details:

Node	Status	Start time	Duration
> automation	All Steps OK		0.00:01
> compliance	All Steps OK		0.00:04
> container	All Steps OK		0.00:01
> log	All Steps OK		0.00:04
> scm	All Steps OK		0.00:01

Figura 8.44: Job

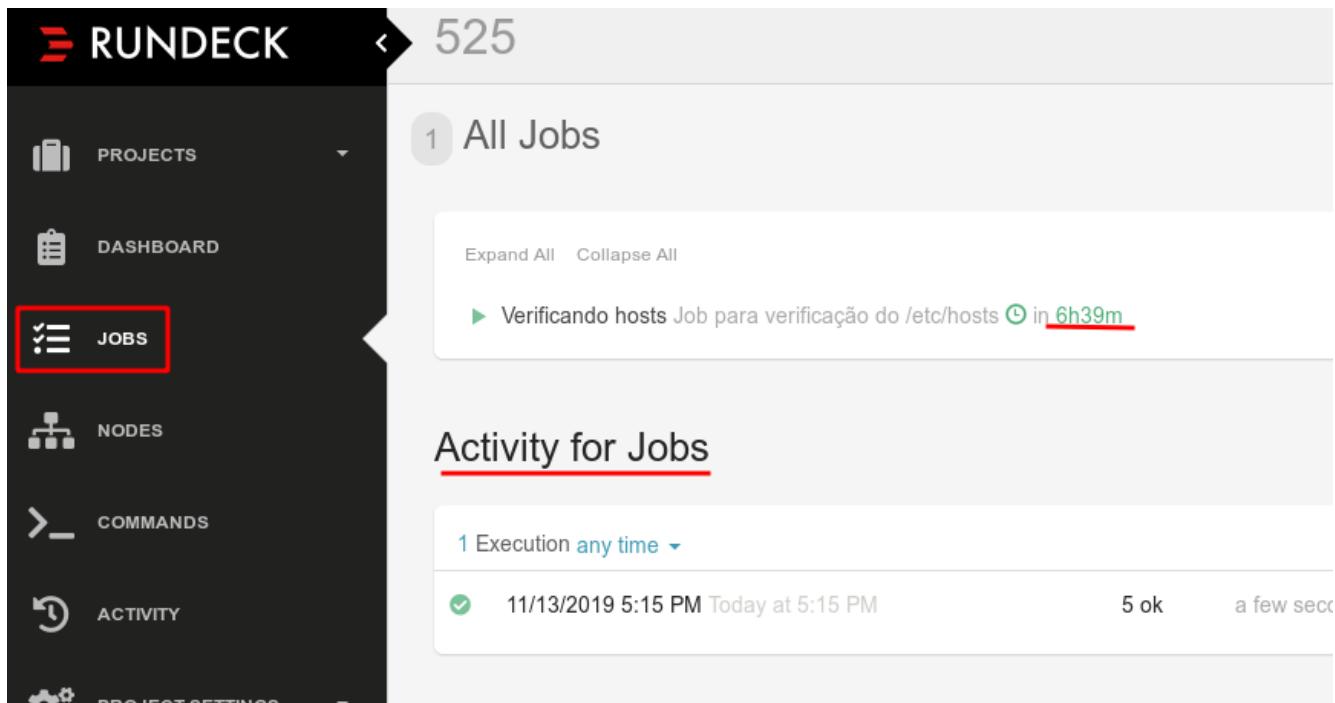


Figura 8.45: Job

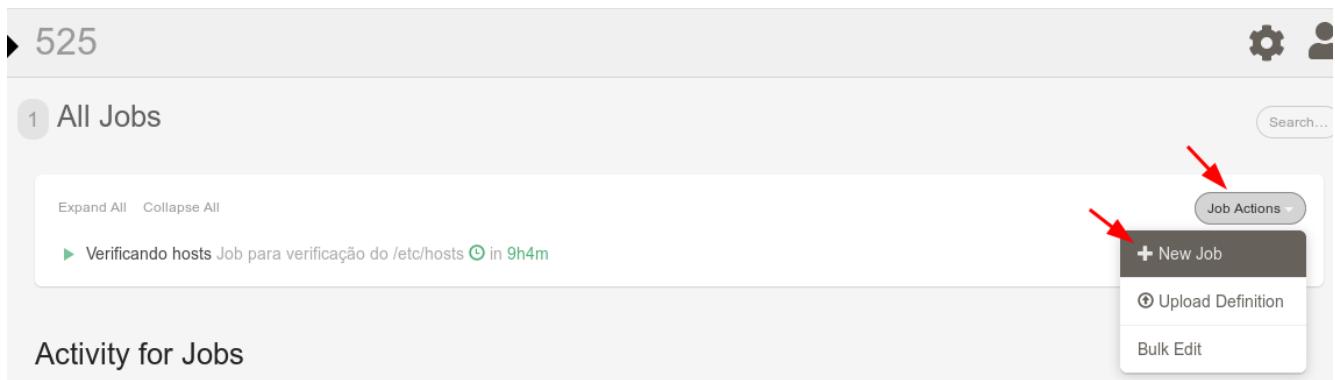


Figura 8.46: Job

Details   Workflow   Nodes   Schedule   Notifications   Other

---

**Job Name** Verificar o DNS-Google Group

**Description** Edit

1 Job para verificar se o dns da google está acessível pelos servidores

The first line of the description will be shown in plain text, the rest will be rendered with the rest of the job description.

Cancel Create

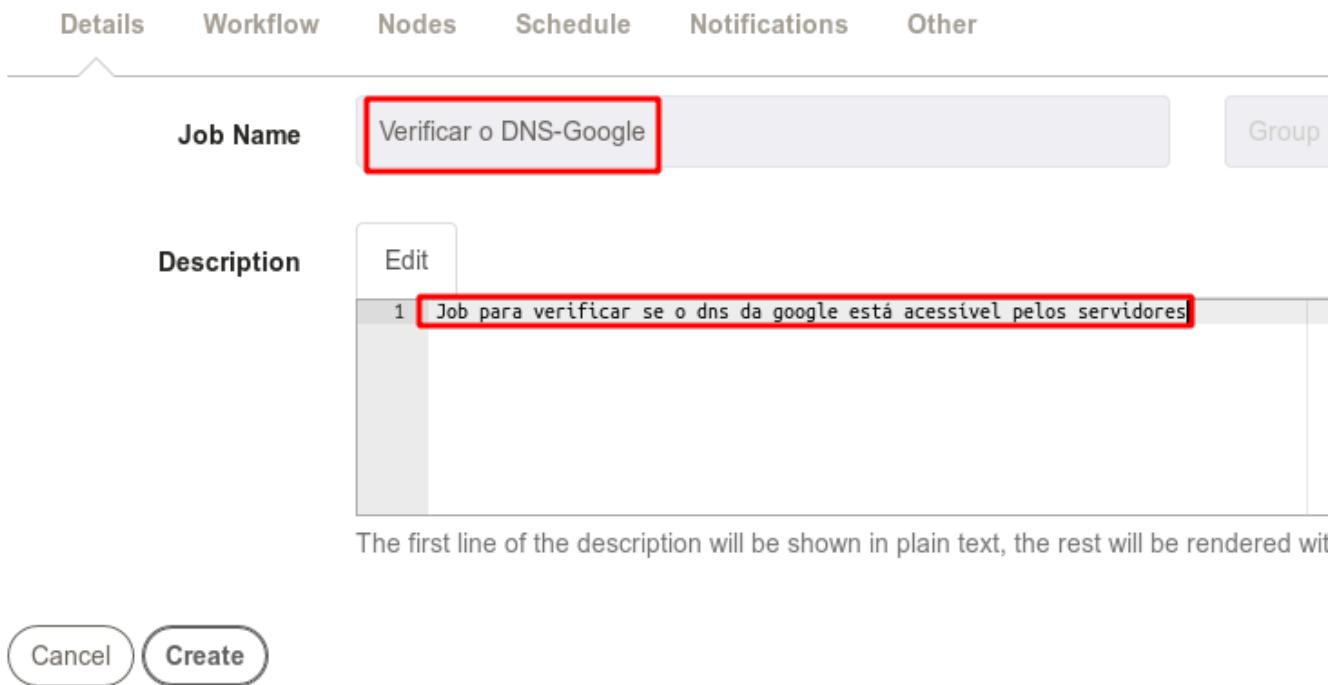


Figura 8.47: Job

---

**Command** ping -c4 8.8.8.8

---

**Step Label** Teste de ping ao DNS Primário da Google

Cancel Save

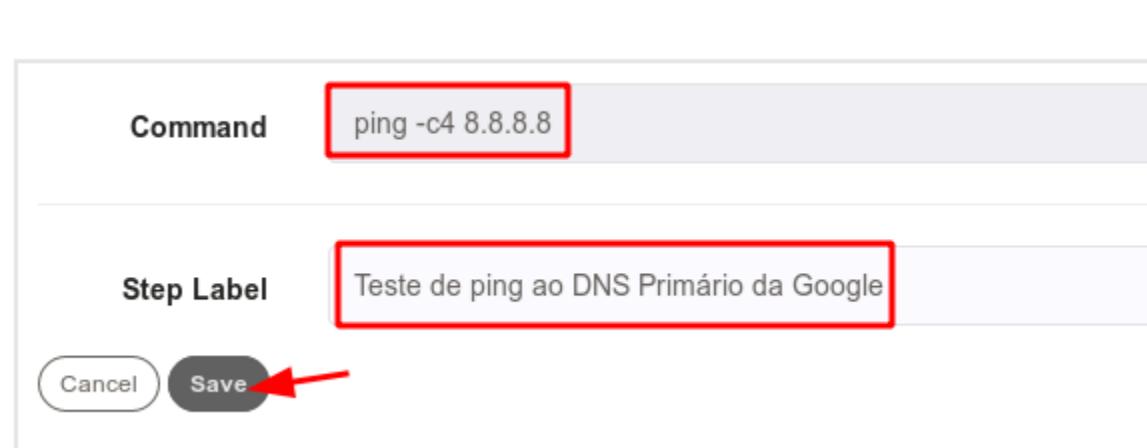


Figura 8.48: Job

Adicione um segundo passo para efetuar uma pesquisa de nome ao DNS

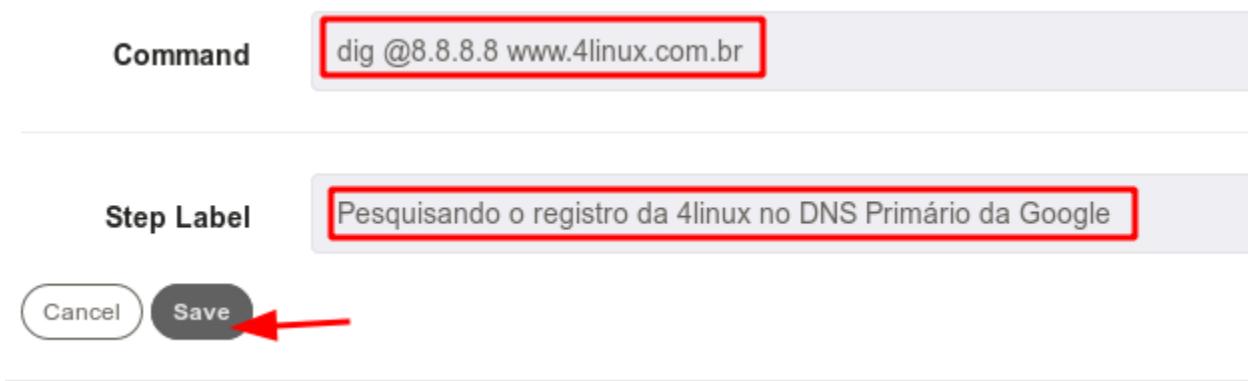


Figura 8.49: Job

Repita os passos para o DNS Secundário da Google

Em Nodes selecione **dispatch to nodes** e aplique o filtro .\*

Marque para continuar a execução em caso de falha e clique em **Create**

Para executar clique em **Run Job Now**

Acompanhe a execução e verifique que alguns passos irão falhar, clique em expandir e verifique quais foram os problemas.

Verificamos que o **dig** não está instalado nas máquinas **automation**, **container** e **scm**, para corrigir este problema podemos ir no menu Commands aplicar o filtro para **automation**, **container** e executar o comando **yum clean all && yum install bind-utils -q -y** para instalação do dig.

Para a máquina **scm** executaremos o comando **apt-get update && apt-get install dnsutils -y**

Volte ao menu de Jobs e clique para executar novamente o job **Verificar DNS**

Verifique que todos os passos foram concluídos com sucesso, para visualizar cada passo é possível clicar para expandir

## Integrações com SCM

### Criando um Repositório no Gogs

Acesse o Gogs pelo endereço <http://scm.4labs.example> clique em **Sing In** e efetue o login com o usuário **analista** e senha **devops@4linux**

No canto superior direito, clique em + e em seguida em + **New Repository**

Preencha o nome do repositório como **rundeck**, marque o repositório como **privado**, digite uma descrição e clique em **Create Repository**

### Configurando a Integração

Iremos copiar a chave pública e adiciona-la ao Gogs para que seja possível o acesso ao repositório, para isto copie a chave da máquina **automation**

```
1 vagrant ssh automation
2 sudo cat /root/.ssh/id_rsa.pub
```

No canto superior direito, clique na foto do usuário **analista** e em seguida em **Your Settings**

Clique em **SSH Keys** e em seguida em **Add Key**, cole a chave copiada e clique em **Add Key** no final da página.

1.  
 ping -c4 8.8.8.8  
Teste de ping ao DNS Primário da Google
2.  
 dig @8.8.8.8 www.4linux.com.br  
Pesquisando o registro da 4linux no DNS Primário da Google
3.  
 ping -c4 8.8.4.4  
Teste de ping ao DNS Secundário da Google
4.  
 dig @8.8.4.4 www.4linux.com.br  
Pesquisando o registro da 4linux no DNS Secundário da Google

Figura 8.50: Job

Nodes  Dispatch to Nodes  Execute locally

Choose whether the Job will run on filtered nodes or only on the local node.

**Node Filter**   ? Search

**Exclude Filter**  ? Search

This is a secondary filter that excludes nodes from the Node Filter results

Figura 8.51: Job

**Rank Order**  Ascending  Descending

**If a node fails**  Fail the step without running on any remaining nodes.  Continue running on any remaining nodes before failing the step.

**If node set empty**  Fail the job.  Continue execution.

**Node selection**  Target nodes are selected by default  The user has to explicitly select target nodes

**Orchestrator**

This can be used to control the order and timing in which nodes are processed.

**Create** 

Figura 8.52: Job

**Verificar DNS Google**

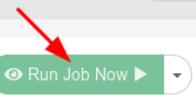
Job para verificar se o dns da google esta acessível pelos servidores.

02a7eb65-a528-4502-a775-9d86351ee399

Action ▾

Definition ▾

Follow execution Nodes ▾

Run Job Now ▾ 

Nodes Change the Target Nodes (5)

Figura 8.53: Job

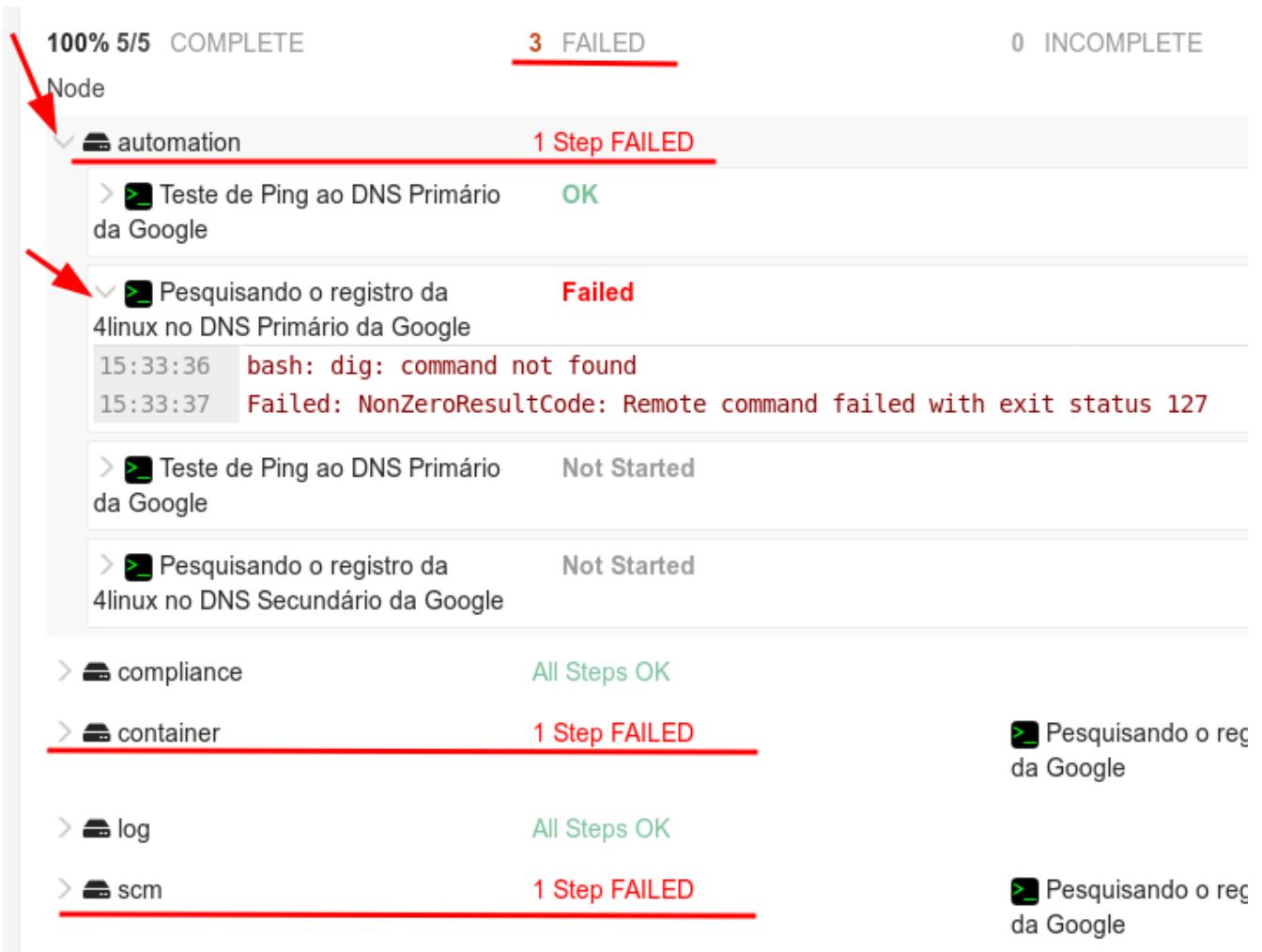


Figura 8.54: Job

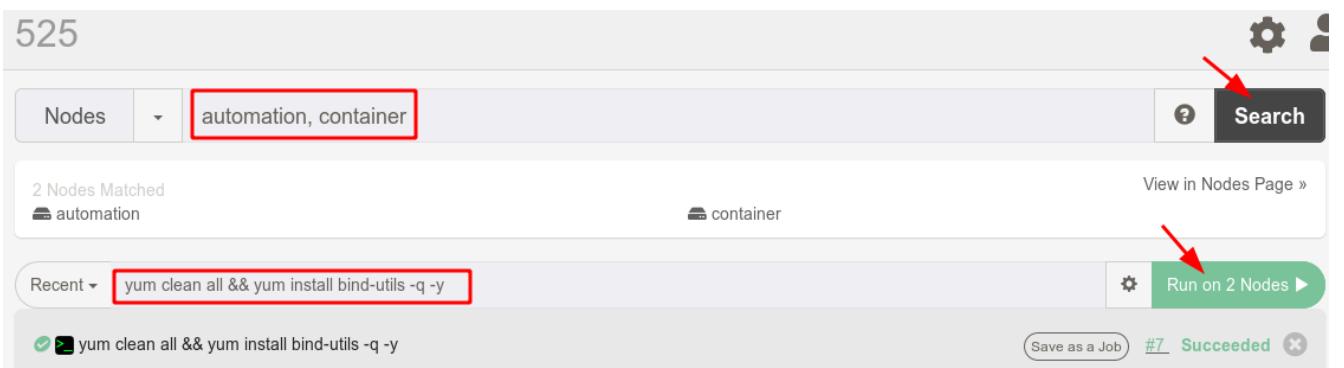


Figura 8.55: Job



Figura 8.56: Job

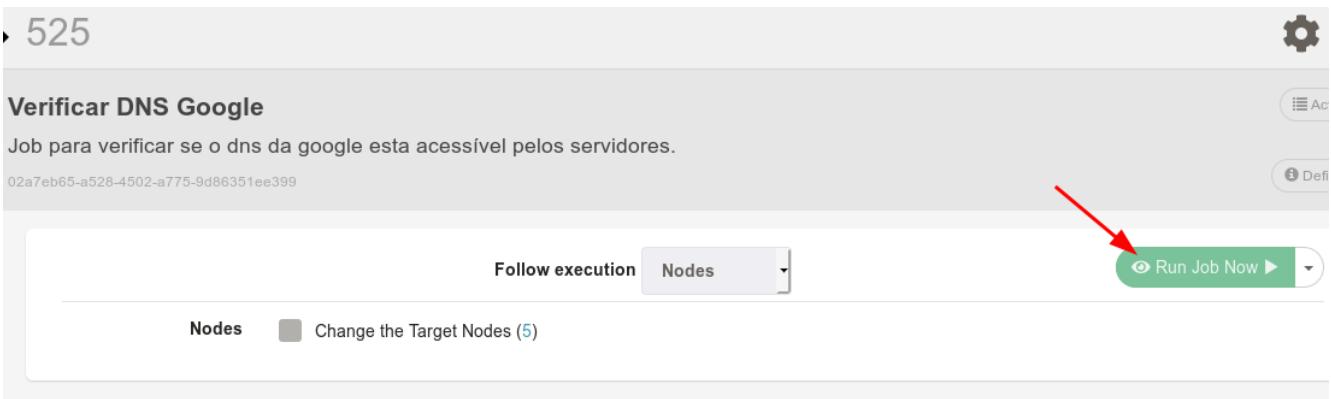


Figura 8.57: Job

Uma vez adicionada a chave, podemos configurar a integração com o rundeck

Clique em **Dashboard** e em seguida em **rundeck**

Selecione **SSH** e copie o endereço do repositório

Acesse o Rundeck pelo endereço <http://automation.4labs.example:4440> e efetue o login como **analista** e senha **devops@4linux**

Selecione o projeto **525**

Clique em **Project Settings** e em **Setup SCM**

Aqui teremos as opções de Exportar e Importar definições dos Jobs para repositórios externos do GIT, vamos clicar em **SETUP** em Git Export

Altere o Committer Name para **Analista DevOps** e Email para **analista@4labs.example**, marque o checkbox para criar a branch se ela não existir.

Em **Git Repository** selecione **true** em **Synchronize Automatically** e cole o endereço do repositório **SSH** em **Git URL**

Em **Autentication** selecione **no** em **SSH Strict Host Key Checking**, adicione a chave **keys/devops** em **SSH Key Storage Path** e clique em **Setup**

Será exibida a tela com as configurações do plugin informando que a configuração foi concluída.

## Sincronizando os Jobs

Clique em **Jobs** e em seguida em **Job Actions, Commit Changes to SCM** no Job **Verificar DNS Google**

Adicione uma mensagem de commit e clique em **commit**

The screenshot shows a Gogs interface for a job named "Verificar DNS Google". The job status is "Succeeded" with a duration of 0.00:50 at 4:09 pm. The job was run by "you". The job details show a 100% complete rate with 5/5 steps. A red arrow points to the "automation" node, which contains five sub-steps all marked as "OK".

Node	Step Details	Start time	Duration
automation	All Steps OK		0:00:11
> automation	Teste de Ping ao DNS Primário da Google OK	4:08:44 pm	0:00:04
> automation	Pesquisando o registro da 4linux no DNS Primário da Google OK	4:08:49 pm	0:00:00
> automation	Teste de Ping ao DNS Primário da Google OK	4:08:49 pm	0:00:03
> automation	Pesquisando o registro da 4linux no DNS Secundário da Google OK	4:08:53 pm	0:00:01
> compliance	All Steps OK		0:00:11
> container	All Steps OK		0:00:10
> log	All Steps OK		0:00:11
> scm	All Steps OK		0:00:08

Figura 8.58: Job

The screenshot shows the Gogs "Sign In" page. The "Username or email" field contains "analista" and the "Password" field contains "devops@4linux", both of which are highlighted with a red box. There is also a "Remember Me" checkbox and a "Sign In" button.

Figura 8.59: Gogs Login

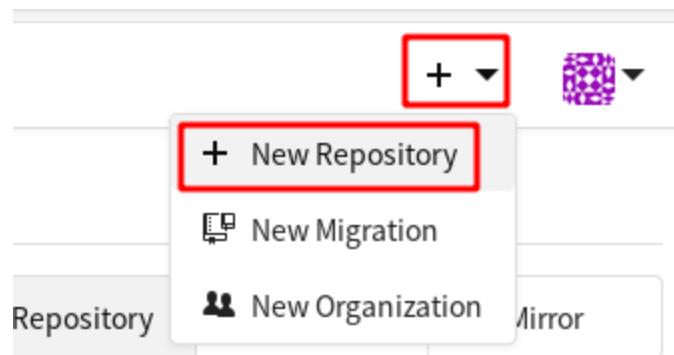


Figura 8.60: New Repository

### New Repository

Owner \* analista

Repository Name \* **primeiro**

A good repository name is usually composed of short, memorable and unique keywords.

Visibility  This repository is **Private**

Description **Meu primeiro repositório com Gogs**

Description of repository. Maximum 512 characters length.  
Available characters: 478

.gitignore Select .gitignore templates

License Select a license file

Readme **Default**

Initialize this repository with selected files and template

**Create Repository** **Cancel**

Figura 8.61: New Repository

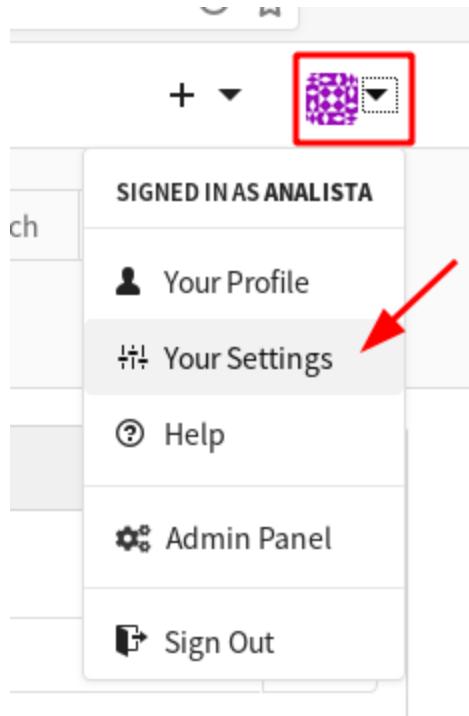


Figura 8.62: Gogs Settings

<b>Settings</b>	
Profile	
Avatar	
Password	
Email Addresses	
<b>SSH Keys</b>	(highlighted)
Security	
Repositories	
Organizations	
Applications	
Delete Account	

### Manage SSH Keys

[Add Key](#)

This is a list of SSH keys associated with your account. As these keys allow anyone using them to gain access to your repositories, it is highly important that you make sure you recognize them.

**Don't know how?** Check out GitHub's guide to [create your own SSH keys](#) or solve [common problems](#) you might encounter using SSH.

**Add SSH Key**

**Key Name**

**Content**

```
ssh-rsa
AAAAB3NzaC1yc2EAAAQABAAQCXaubLFK7Vcdk4fjiCDqloL9/rLCOiYaGxBPXCgEOBDR
pk2eFRsgfLgzlgNG+kXUbaylhtlYYoQzkLahCUdRA+go0XVGwKg6GuLMOsLlqphOvX53238np8H
Ms0t5hZqBSVM55Se1YDq6FOjMW9pWWBCwNE3Esbla4Dsp0Yljh3cNFjUBIxFe+YEgiN
/y+KC6hG8nQA8SmQ2TPf3DewLqbpe5jZQclXT/Z86wAyexFjrTl9c5QTNEOlpmiwKttM3
/OV+h2kuAUllmrw0nABqjeQvezMAldyZpKZKf2WYppz3ukfOvLNw5djBY1gx8TOocas791hQsWD
d3Jt4GOMsYi7 caiodelgado@serenade
```

[Add Key](#)

Figura 8.63: SSH Key

New SSH key 'devops' has been added successfully!

## Manage SSH Keys

Add Key

This is a list of SSH keys associated with your account. As these keys allow anyone using them to gain access to your repositories, it is highly important that you make sure you recognize them.

	Key	Name	SHA256	Added on	Activity	Action
○		devops	SHA256:Ut150L/kVED+Xvf2O84i+J30KEGaPtszSJ5SNIHZdbs	Added on Nov 14, 2019	— <small>① No recent activity</small>	<a href="#">Delete</a>

**Don't know how?** Check out GitHub's guide to [create your own SSH keys](#) or solve [common problems](#) you might encounter using SSH.

Figura 8.64: SSH Key

Os Jobs que ainda não foram enviados para o repositório estarão com uma indicação (!) informando que foram

2 All Jobs

Expand All Collapse All

▶ Verificando hosts Job para verificação do /etc/hosts in 11h27m

▶ Verificar DNS Google Job para verificar se o dns da google esta acessível

modificados (seu estado atual não está no repositório)

Podemos também ir diretamente na tela de jobs na opção **Job Actions** e em seguida em **Commit Changes to Git**

The screenshot shows a GitHub repository page for 'analista / primeiro'. At the top, there's a navigation bar with links for Dashboard, Issues, Pull Requests, and Explore. On the right side of the header are buttons for Unwatch (with a count of 1), Star (with a count of 0), and Settings. Below the header, the repository name 'analista / primeiro' is displayed, along with a 'Quick Guide' section. This section includes a 'Clone this repository' link with instructions and a 'Help' link. It also shows cloning options via HTTP (selected) and SSH, with the URL 'http://scm.4labs.example/analista/primeiro.git'. Below this is a 'Create a new repository on the command line' section containing a code block with git commands. Further down is a 'Push an existing repository from the command line' section with another code block.

Figura 8.65: Repository

The screenshot shows a GitHub repository page for 'analista / rundeck'. The layout is similar to Figura 8.65, with a header, a 'Quick Guide' section, and sections for cloning and pushing. A red arrow points to the 'SSH' button in the cloning section, which is highlighted with a red box. The URL shown is 'ssh://gogs@scm.4labs.example:2222/analista/rundeck.git'.

Figura 8.66: Repository

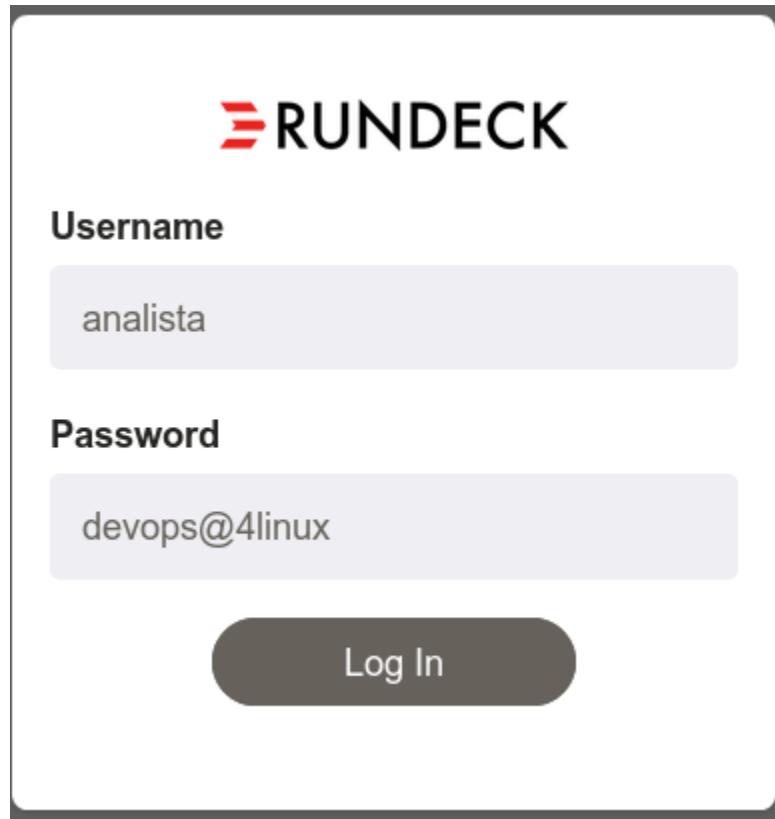


Figura 8.67: Rundeck Login

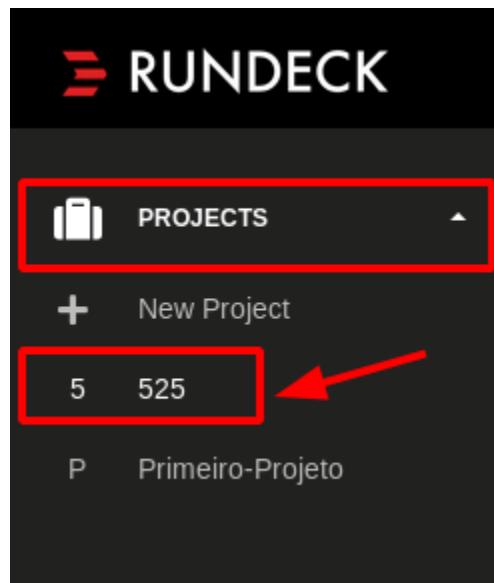


Figura 8.68: Project

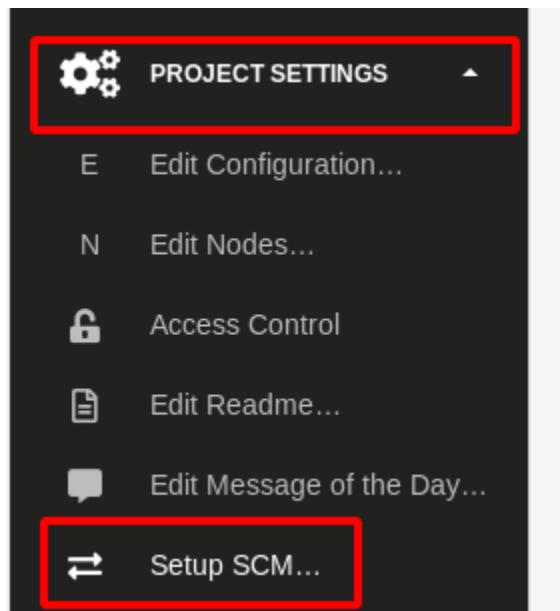


Figura 8.69: Setup SCM

A screenshot of the 'All Jobs' page. At the top left, it says '2 All Jobs'. On the right, there's a search bar and a 'Job Actions' dropdown menu. The menu items are: 'Modified' (highlighted with a red arrow), 'New Job', 'Upload Definition', 'Bulk Edit', 'SCM Export Actions', 'Commit Changes to Git' (highlighted with a red arrow), and 'Toggle SCM OFF'. Below the menu, there's a section titled 'Activity for Jobs' with a dropdown for '3 Executions any time' and a timestamp '11/14/2019 4:09 PM Yesterday at 4:09'.

## Setup SCM

Enable or configure SCM integration.

### SCM Export

SCM Export Plugins can export Job definitions to an external SCM Repo

#### Git Export

Export Jobs to a Git Repository [More...](#)

 Setup



### SCM Import

SCM Import Plugins can import Job definitions from an external SCM Repo

#### Git Import

Import Jobs from a Git Repository [More...](#)

 Setup

Figura 8.70: Setup SCM

## Setup SCM Plugin: Git Export

Project		InfraAgil
Committer Name	<input type="text" value="Analista DevOps"/>	
Name of committer/author of changes. <a href="#">More...</a>		
Committer Email	<input type="text" value="analista@4labs.example"/>	
Email of committer/author of changes. <a href="#">More...</a>		
Export UUID Behavior	<input type="text" value="preserve"/>	
How to handle UUIDs for exported Job source files <a href="#">More...</a>		
<input checked="" type="checkbox"/> Create Branch if it doesn't exist		
If the entered branch doesn't exist on remote repo, create a new one. If false, it will fail if the branch doesn't exist		

Figura 8.71: Setup SCM

Git Repository	
Synchronize Automatically	<input type="text" value="true"/>
Automatically pull remote changes on automatic fetch. If false, you can perform it manually	
Base branch on	<input type="text" value="master"/>
Create the new branch based on the existent branch	
Base Directory	<input type="text" value="/var/lib/rundeck/projects/InfraAgil/scm"/>
Directory for checkout	
Git URL	<input type="text" value="ssh://gogs@scm.4labs.example:2222/analista/rundeck.git"/>
Checkout url. <a href="#">More...</a>	
Branch	<input type="text" value="master"/>
Checkout branch	
Fetch Automatically	<input type="text" value="true"/>
Automatically fetch remote changes for local comparison. If false, you can perform it manually	

Figura 8.72: Setup SCM

Authentication ▾

SSH: Strict Host Key Checking	<input type="text" value="no"/>
Use strict host key checking. <a href="#">More...</a>	
SSH Key Storage Path	<input type="text" value="keys/devops"/>
Path can include variable references <a href="#">More...</a>	
Password Storage Path	<input type="text"/>
Password to authenticate remotely (e.g. for SSH or HTTPS URLs). <a href="#">More...</a>	

**Setup** 

**Select...** 

Figura 8.73: Setup SCM

### Git Export: Commit Changes

Commit changes to local git repo.

Select changes to export.

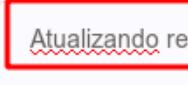
Verificando hosts - [View Diff...](#)

Verificando hosts-fb43a4be-02a7e...

Verificar DNS Google - [View Diff...](#)

Verificar DNS Google-02a7e...

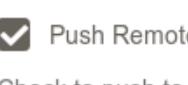
Select All • Select None

Commit Message 

Enter a commit message

Tag 

Enter a tag name

Push Remote 

Check to push to

SCM Plugin Setup Complete

## Setup SCM

Enable or configure SCM integration.

### SCM Export

SCM Export Plugins can export Job definitions to an external SCM Repo



#### Git Export Enabled

Export Jobs to a Git Repository [More...](#)

Committer Name: [Analista DevOps](#)

Committer Email: [analista@4labs.example](mailto:analista@4labs.example)

Export UUID Behavior: [preserve](#)

Synchronize Automatically: [true](#)

Create Branch if it doesn't exist: [Yes](#)

Base branch on: [master](#)

File Path Template: \${job.group}\${job.name}-\${job.id}.\${config.format}

Base Directory: </var/lib/rundeck/projects/InfraAgil/scm>

Git URL: <ssh://gogs@scm.4labs.example:2222/analista/rundeck.git>

Branch: [master](#)

SSH: Strict Host Key Checking: [no](#)

SSH Key Storage Path: [keys/devops](#)

Format: [xml](#)

Fetch Automatically: [false](#)

[Disable](#)

Figura 8.74: Setup SCM

The screenshot shows the RunDeck web interface. The left sidebar has buttons for 'PROJECTS', 'DASHBOARD', **JOBS** (which is selected and highlighted with a red box), 'NODES', 'COMMANDS', 'ACTIVITY', and 'PROJECT SETTINGS'. The main area shows 'All Jobs' with two listed: 'Verificando hosts' and 'Verificar DNS Google'. Below this is the 'Activity for Jobs' section, which lists three executions: one successful at 11/14/2019 4:09 PM and two failed at 11/14/2019 3:34 PM. On the right, a context menu for the 'Verificar DNS Google' job is open, with a red box highlighting the 'Actions' dropdown. A red arrow points to the 'Commit Changes to Git' option in this menu.

Figura 8.75: Sync Jobs

Uma mensagem exibida na tela de jobs informa que o push do repositório foi realizado com sucesso.

No **Gogs** atualize a pagina do repositório e verifique que o conteúdo dos Jobs foi adicionado/atualizado

Vamos alterar o Job e adicionar alguns dados. Clique em **Jobs** e em **Verificar DNS Google**

Clique em **Action** e em **Edit this Job**

Em **Workflow** adicione um passo para verificar o site da 4linux com o comando `curl -I www.4linux.com.br` e clique em **Save** para o passo e **Save** novamente para gravar o Job.

Para visualizar as mudanças clique em **Action** e em **Diff Changes**

Será exibida uma tela com as modificações realizadas no código, clique em **Commit Changes to Git** para atualizar o gogs com o arquivo mais recente

Adiciona um comentário e realize o **commit**

No **Gogs** atualize a pagina do repositório e verifique que o conteúdo dos Jobs foi adicionado/atualizado

## Git Export: Commit Changes to Git

Commit changes to local git repo.

Select changes to export.

-  Verificar DNS Google - [View Diff...](#)  
 Verificar DNS Google-02a7eb65-a528-4502-a775-9d86351ee399.xml

Commit  
Message

Commit inicial do Job DNS

Enter a commit message. Committing to branch: 'master'

Tag

Enter a tag name to include, will be pushed with the branch.

- Push Remotely?

Check to push to the remote.

Cancel

Commit



Figura 8.76: Sync Jobs

The screenshot shows a Rundeck interface with the title '525' at the top. Below it, a section titled '2 All Jobs' is visible. A prominent blue message box displays the text 'Remote push result: OK. (Commit: e7b1f1e6f5865b02d9b8704a8d3b02a46968423f)'. Below this message, there are buttons for 'Expand All' and 'Collapse All'. Two job entries are listed under the heading 'Sync Jobs': 'Verificando hosts' (Job para verificação do /etc/hosts) and 'Verificar DNS Google' (Job para verificar se o dns da google esta acessível pelos servidores). Both jobs are marked as completed.

Figura 8.77: Sync Jobs

The screenshot shows a GitHub repository page for 'analista / rundeck'. The repository has 1 commit, 1 branch, and 0 releases. The 'Files' tab is selected. A dropdown menu shows the branch is 'master'. The repository description is 'Repositório de Jobs do Rundeck'. Below the repository info, a table lists three sync jobs: 'Analista DevOps' (commit e7b1f1e6f5, status 'Atualizando repositório'), 'Verificando hosts-fb43a4be...' (commit e7b1f1e6f5, status 'Atualizando repositório'), and 'Verificar DNS Google-02a7e...' (commit e7b1f1e6f5, status 'Atualizando repositório'). All jobs were updated '1 minute ago'.

Analista DevOps	e7b1f1e6f5	Atualizando repositório	1 minute ago
Verificando hosts-fb43a4be...	e7b1f1e6f5	Atualizando repositório	1 minute ago
Verificar DNS Google-02a7e...	e7b1f1e6f5	Atualizando repositório	1 minute ago

Figura 8.78: Sync Jobs

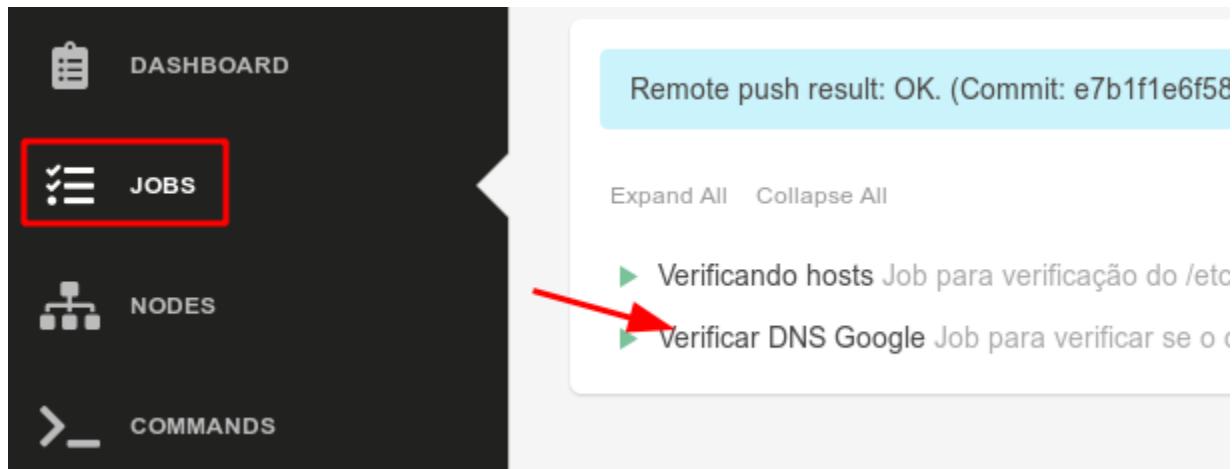


Figura 8.79: Sync Jobs



Figura 8.80: Sync Jobs



Figura 8.81: Sync Jobs

The screenshot shows a Jenkins job named "Verificar DNS Google" which has been modified. The job description is "Job para verificar se o dns da google esta acessível pelos servidores." and its ID is 02a7eb65-a528-4502-a775-9d86351ee399. The "Nodes" tab is selected under the "Follow execution" dropdown. The job has 2 executions, a success rate of 50%, and a duration of 50s. The "Stats" and "Activity" tabs are also visible. A red arrow points to the "Action" dropdown menu at the top right, which contains options like "Edit this Job...", "Duplicate this Job...", "Delete this Job", "Disable Execution", "Download Job definition in XML", "Download Job definition in YAML", "SCM Export Plugin", "Commit Changes to Git", and "Diff Changes". The "Diff Changes" option is highlighted with a red arrow.

Figura 8.82: Sync Jobs

## SCM Export: View Changes

### Verificar DNS Google

Job para verificar se o dns da google esta acessível pelos servidores.

 Modified  Verificar DNS Google-02a7eb65-a528-4502-a775-9d86351ee399.xml

### Current Commit

#### Atualizando repositório

— Analista DevOps<analista@4labs.example> 9m ago in e7b1f1e6f5865b02d9b8704a8d3b02a46968423f ➔

 Download Diff

```
1 00 -36,6 +36,10 00
2      <description>Pesquisando o registro da 4linux no DNS Secundário da Google </description>
3      <exec>dig @8.8.4.4 www.4linux.com.br</exec>
4      </command>
5 +     <command>
6 +         <description>Verificando Website da 4Linux</description>
7 +         <exec>curl -I www.4linux.com.br</exec>
8 +     </command>
9      </sequence>
10     <uuid>02a7eb65-a528-4502-a775-9d86351ee399</uuid>
11   </job>
12
```

Commit Changes to Git

Figura 8.83: Sync Jobs

## Git Export: Commit Changes to Git

Commit changes to local git repo.

Select changes to export.



! Verificar DNS Google - [View Diff...](#)

 Verificar DNS Google-02a7eb65-a528-4502-a775-9d86351ee399.xml

Commit  
Message

Adicionando - Verificar Website

Enter a commit message. Committing to branch: `master`

Tag

Enter a tag name to include, will be pushed with the branch.

Push Remotely?

Check to push to the remote.

[Cancel](#)

[Commit](#)



Figura 8.84: Sync Jobs

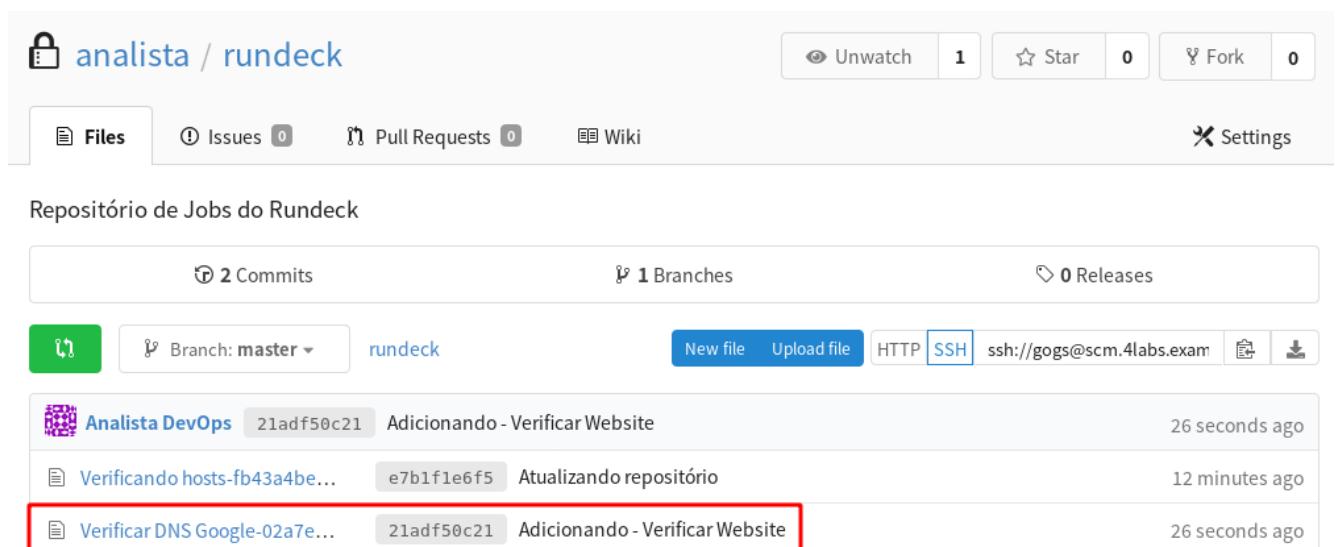


Figura 8.85: Sync Jobs

# Capítulo 9

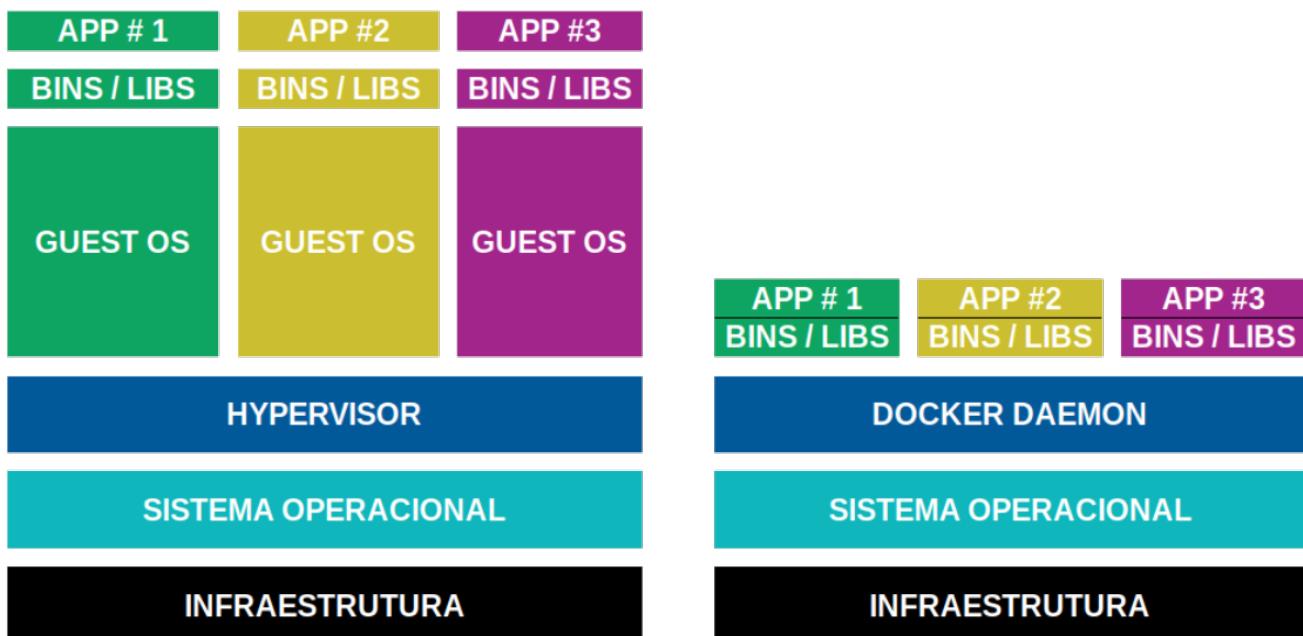
## Containers com Docker

### Conceitos de Containers

#### Introdução ao Docker

Docker é uma plataforma Open Source escrita em Go (linguagem de programação de alto desempenho desenvolvida dentro da Google) que facilita a criação e a administração de ambientes isolados.

Descreve-se como uma plataforma aberta para desenvolvedores e administradores de sistemas para construir, compartilhar e executar aplicações distribuídas. Ele permite a execução das aplicações em container.



Virtual Machines (VMs)

Containers

Figura 9.1: VM vs Docker

**Container** é um processo em área restrita que executa um aplicativo e suas dependências no sistema operacional hospedeiro.

A aplicação dentro de um container é um único processo em execução na máquina, permitindo ao servidor hospedeiro executar vários containers de forma independente. Este modo de trabalho permite remover conflitos entre dependências e

simplificar a implantação de toda a instalação e configuração de forma instantânea, comparado com o modelo tradicional de virtualização.

## Plataformas de Execução

Para a execução do Docker é necessário que o sistema hospedeiro possua um Kernel Linux.

O Docker Engine dispõe de duas Plataformas para execução: \* **Docker Desktop** -> Executado em sistemas hospedeiros Apple macOS e Microsoft Windows, é instalada uma máquina virtual com um kernel linux e a Docker Engine.

- **Docker Server** -> Executado em diferentes distribuições de sistemas operacionais Linux, tais como CentOS, RHEL, Debian, Ubuntu, Fedora, etc.

Os procedimentos de instalação do Docker Engine encontram-se na Documentação Oficial

## Arquitetura

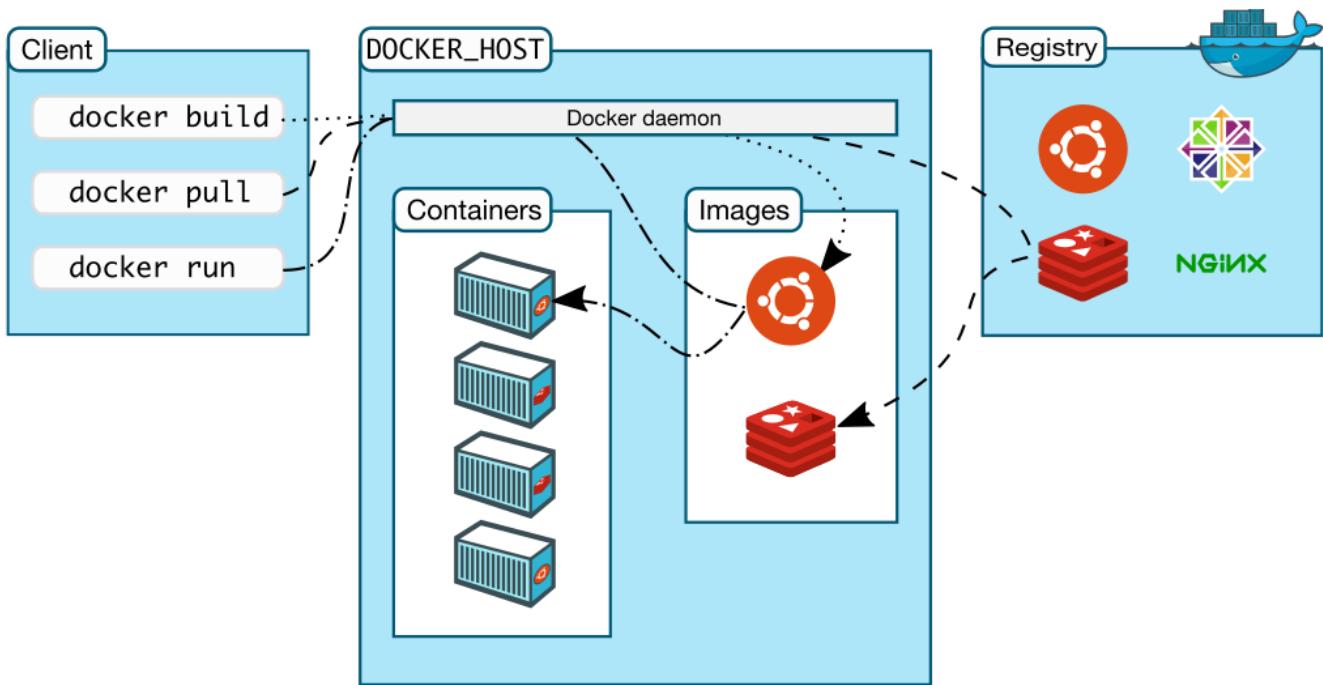


Figura 9.2: Arquitetura

O Docker utiliza uma arquitetura *client-server*. O Docker *client* se comunica com o Docker *daemon*, o qual realiza as tarefas de **build**, **execução** e **distribuição** dos containers.

O Docker *client* e *daemon* podem ser executados em um mesmo host ou em sistemas diferentes.

- **Docker Client** - é o binário (*docker*) para comunicação com o Docker *Daemon*
- **Docker Daemon** - é o processo (*dockerd*) que escuta as solicitações via API e gerencia os objetos do Docker tais como imagens, containers, redes e volumes.
- **Docker Registry** - é o local onde são armazenadas as imagens docker.

## Funcionamento

### Namespaces

Os *Namespaces* fornecem isolamento para os containers, limitando seu acesso ao recursos do sistema e a outros namespaces. Isto significa, por exemplo, que um usuário *root* dentro de um container é diferente de um usuário *root* da máquina hospedeira.

Com o isolamento, os sistemas em execução nos containers tem suas próprias árvores de processo, sistemas de arquivos, conexões de rede e muito mais.

## Cgroups

Os containers trabalham com *cgroups* (Control Groups) que fazem isolamento dos recursos físicos da máquina. Em geral os cgroups podem ser utilizados para controlar estes recursos tais como limites e reserva de CPU, limites e reserva de memória, dispositivos, etc.

# Comandos essenciais do Docker

## Comandos Nativos

Antes de executar os comandos do docker, vamos conectar na máquina container e alterar o usuário para o root

```
1 vagrant ssh container  
2 sudo su -
```

Para visualizar informações do ambiente, podemos utilizar o comando **docker system info** o qual exibirá informações do Docker como versão, quantidade de containers em execução, storage drivers, entre outros

```
1 docker system info  
2 docker info
```

*Os comandos listados acima são equivalentes*

Para listar containers, imagens, redes e volumes no docker, utilizamos o comando **docker <comando> ls**

```
1 docker container ls  
2 docker image ls  
3 docker network ls  
4 docker volume ls
```

- **docker container ls** - lista os containers
- **docker image ls** - lista as imagens
- **docker network ls** - lista as redes
- **docker volume ls** - lista os volumes

Para pesquisar por uma imagem, utilizamos o comando **docker search**

```
1 docker search debian
```

Para efetuar o download da imagem utilizamos o comando **docker image pull**

```
1 docker image pull debian
```

Para executar um contâiner, utilizamos o comando **docker container run**

```
1 docker container run [OPTIONS] IMAGE [COMMAND] [ARG...]  
2 docker container run -dit --name container1 --hostname devops debian
```

**Descrição do comando:** \* **docker container run (...)** **debian** - Executa um container, sendo o ultimo parâmetro o nome da imagem a ser utilizada(informação obrigatória) \* **-dit** - Executa um container como processo (**d** = Detached), habilitando a

interação com o container (**i** = Interactive) e disponibiliza um pseudo-TTY(**t** = TTY) \* **-name** - Define o nome do container \* **-hostname** - Define o hostname do container

Agora que temos nosso primeiro contêiner em execução, podemos listar os containers(**docker container ls**) e conectar ao mesmo através do comando **docker container attach**

```
1 docker container ls
2 docker container attach container1
```

*Note que ao se conectar ao container a PS1 será modificada para root@devops:/#*

Execute alguns comandos no container

```
1 ip -c a
2 hostname
3 cat /etc/hosts
4 uname -r
5 exit
```

Liste novamente os containers

```
1 docker container ls -a
```

*Note que agora o container está parado, isto aconteceu pois o processo principal do container recebeu um return code diferente de 0*

Inicie novamente o container e conecte-se ao mesmo

```
1 docker container start container1
2 docker container attach container1
```

O comando **docker container start** inicia um container parado, o comando **docker container stop** para um container que esteja em execução

Utilize a sequencia de teclas **<CTRL> + <P> + <Q>** para se desconectar do container sem que ele seja parado. Este comando é chamado de *Read escape sequence*.

```
1 <CTRL> + <P> + <Q>
2 docker container ls
```

*Note que agora o container ainda está em execução.*

Para verificar os logs do container utilizamos o comando **docker container logs**

```
1 docker container logs container1
```

Pare e remova o container, apóis isto verifique os containers existentes

```
1 docker container stop container1
2 docker container rm container1
3 docker container ls -a
```

*Podemos utilizar o parâmetro **-f** no comando **docker container rm** para que o container seja removido mesmo que esteja sendo executado*

Execute um novo container

```
1 docker container run -dit --name c1 --hostname server debian
2 docker container ls
```

Crie um arquivo de teste na pasta atual para enviar ao container c1

```
1 echo "Arquivo de teste" > /tmp/arquivo
2 docker container cp /tmp/arquivo c1:/tmp
```

O comando **docker container cp** copia um arquivo da máquina host para o container ou vice-versa.

Verifique se o arquivo existe dentro do container através do comando **exec**

```
1 docker container exec c1 ls -l /tmp
2 docker container exec c1 cat /tmp/arquivo
```

O comando **docker container exec** executa um comando no container e envia o retorno na saída padrão(STDOUT) da máquina, caso o container não tenha sido iniciado com a opção **-i** o retorno não será mostrado no STDOUT

## Dockerhub

O **Dockerhub** é o maior repositório de imagens de containers (*registry*) do mundo. Através dele os usuários obtêm acesso a repositórios públicos para armazenar e compartilhar imagens, ou podem escolher um plano de assinatura para repositórios privados.

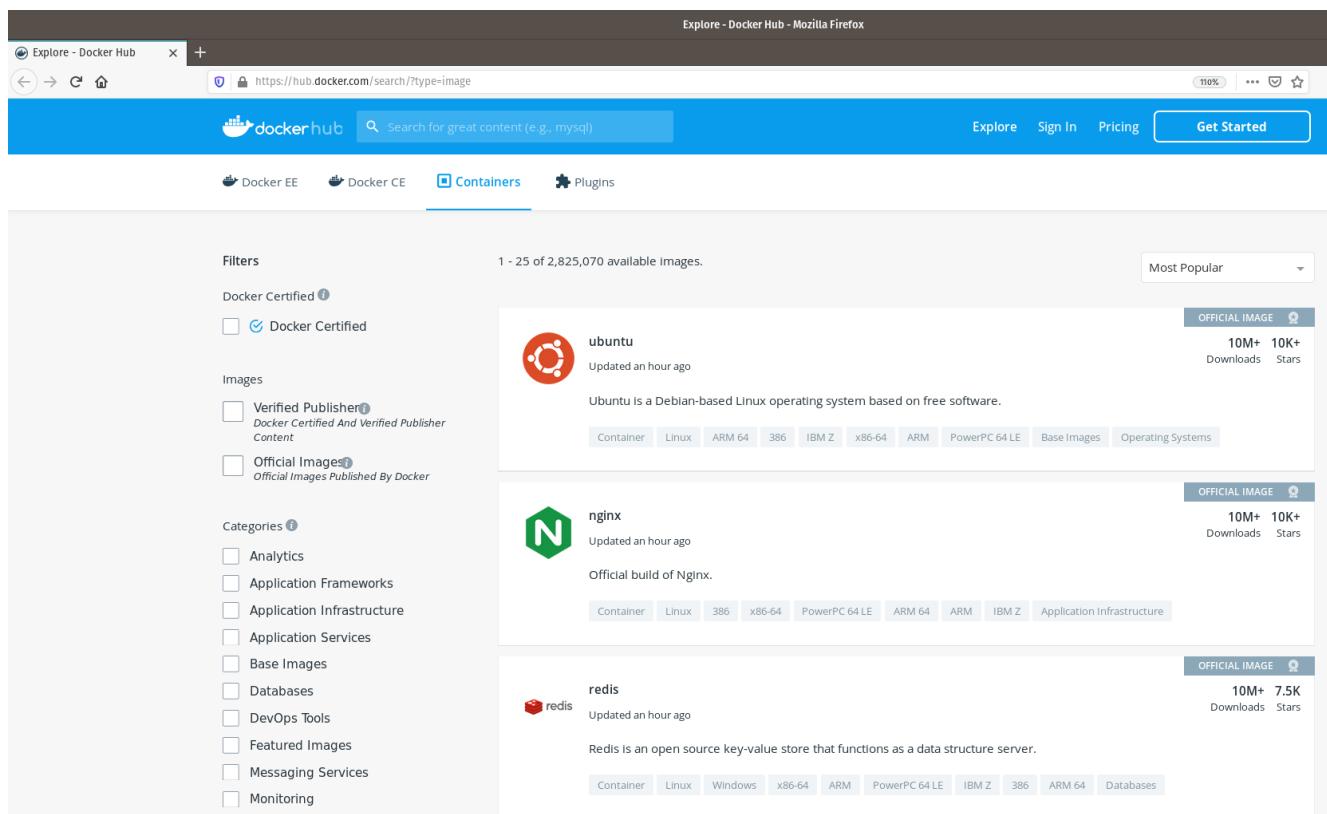


Figura 9.3: Dockerhub

Serviços Fornecidos pelo **Dockerhub**: \* Hospedagem de Imagens Docker; \* Autenticação de usuário \* Automatização do processo de construção de imagens através de triggers (webhooks) \* Integração com o Github e Bitbucket

OBS: **Dockerhub** pode servir como fonte de consulta para utilizar as imagens da melhor maneira, já que geralmente os projetos criam um documento orientando sobre informações de tags, volumes, environments, entre outras opções validas da imagem.

## Docker Images

Uma imagem Docker é um pacote executável que inclui tudo o que é necessário para executar um aplicativo, incluindo o código, bibliotecas, variáveis de ambiente e arquivos de configuração.

As imagens do Docker possuem camadas intermediárias que aumentam a capacidade de reutilização, diminuem o uso do disco e aceleram a construção do docker, permitindo que cada etapa seja armazenada em cache. Essas camadas intermediárias não são mostradas por padrão.

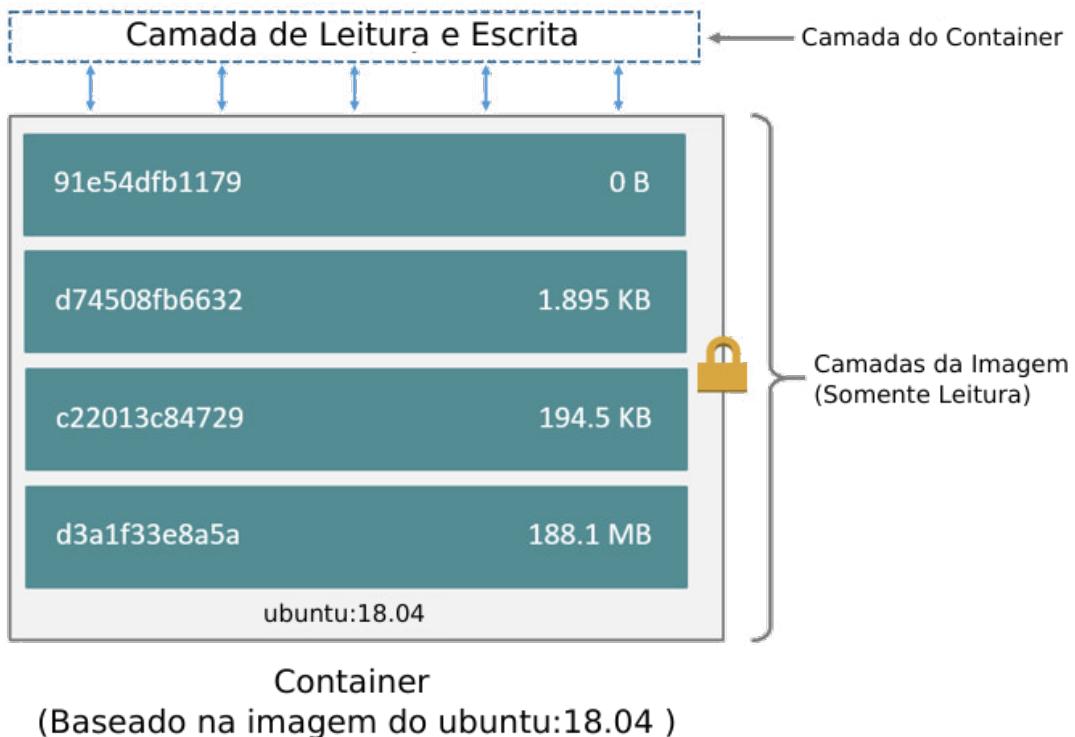


Figura 9.4: Image Layers

As imagens do Docker possuem várias camadas somente leitura e não podem ser modificadas. Você pode criar novas imagens a partir de imagens existentes, mas não pode modificar essas imagens existentes.

A principal diferença entre um container e uma imagem é a camada gravável superior. Todas as gravações no container que adicionam novos dados ou modificam dados existentes são armazenados nessa camada gravável. Quando o container é excluído, a camada gravável também é excluída. A imagem subjacente permanece inalterada.

Como cada container tem sua própria camada gravável e todas as alterações são armazenadas nessa camada de container, vários containers podem compartilhar o acesso a uma mesma imagem subjacente e ainda sim ter seu próprio estado de dados.

## Gerenciar Imagens no Docker

Liste as imagens e verifique o histórico de comandos utilizados para sua construção

```
1 docker image ls
2 docker image history debian
```

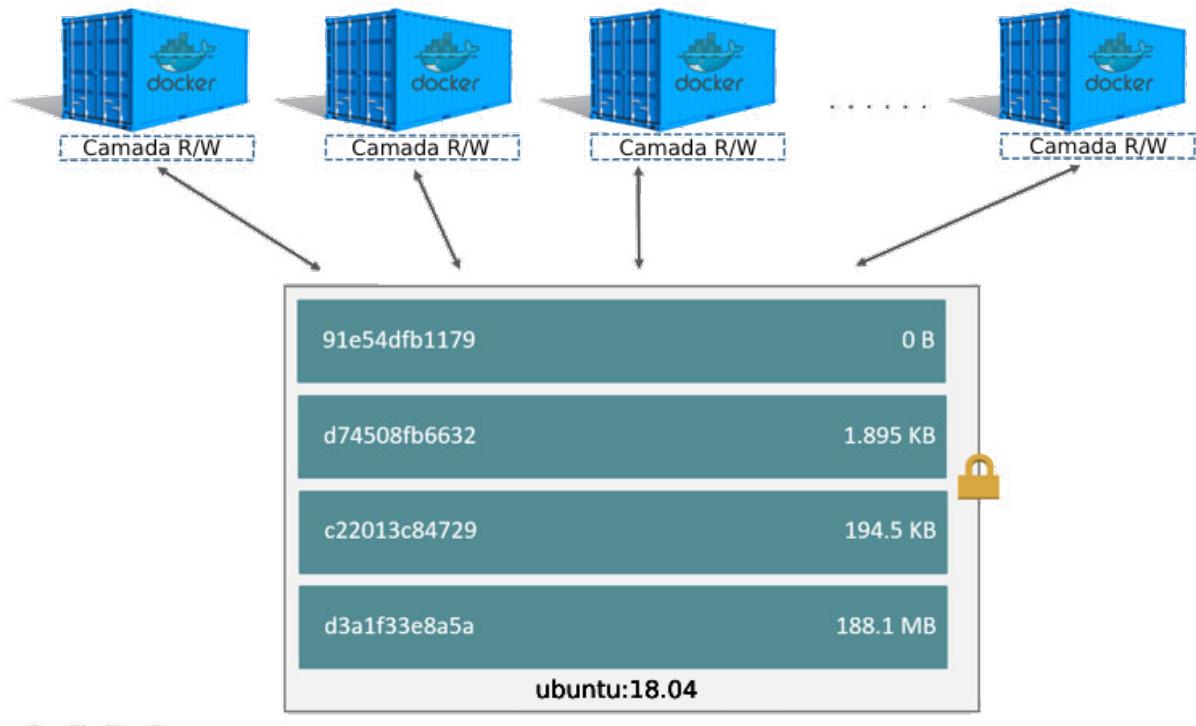


Figura 9.5: Image Layers

O comando **docker image history** mostra as camadas que compõem uma imagem

Para inspecionar uma imagem utilizamos o comando **docker image inspect**

```
1 docker image inspect debian
```

O comando **docker image inspect** exibe informações detalhadas de uma imagem

Vamos criar uma nova imagem a partir de um container existente, para isso vamos criar um container e instalar alguns pacotes

```
1 docker container run -dit --name servidor-debian debian
2 docker container exec servidor-debian apt-get update
3 docker container exec servidor-debian apt-get install apache2 -y
```

Para criar uma nova imagem a partir das alterações feitas em um container podemos utilizar o parâmetro **commit**

```
1 docker container commit servidor-debian servidor-web
2 docker image ls
```

O comando **docker container commit <container> <imagem>** cria uma imagem a partir de alterações realizadas em um container, este procedimento não é o recomendado para este fim, mas a frente veremos outras soluções

Para salvar a imagem podemos utilizar o parâmetro **save**

```
1 docker image save servidor-web -o imagem-servidor-web.tar
2 du -sh imagem-servidor-web.tar
```

Remova o container e a imagem servidor web

```
1 docker container rm -f servidor-debian
2 docker image rm servidor-web
3 docker image ls
```

Para carregar uma imagem salva a partir de um arquivo, podemos utilizar o parâmetro **load**

```
1 docker image load -i imagem-servidor.tar
2 docker image ls
```

Para testar o funcionamento da imagem, podemos criar um container utilizando a mesma

```
1 docker container run -dit --name webserver servidor-web
2 docker container ls
```

Remova todos os containers

```
1 docker container rm -f $(docker container ls -aq)
2 docker container ls
```

No comando acima, estamos passando através de um subshell o comando **docker container ls -aq** que lista todos os containers por **id**, sendo assim, será feita a remoção de todos os containers

**ATENÇÃO** - É necessário tomar muito cuidado ao executar o comando de remoção de todos os containers, já que essa é um remoção forçada e indiscriminada que consequentemente pode ocasionar perda de informações relevantes.

## Dockerfile

### O que é o Dockerfile

O Docker pode criar imagens automaticamente, lendo as instruções de um **Dockerfile**, que é um documento de texto que contém as instruções para a criação de uma imagem docker através do comando **docker image build**.

### Sintaxe

Parâmetro	Valor
<b>FROM</b>	Distribuição:Versão
<b>COPY</b>	Arquivo_Local Caminho_Absoluto_no_Container
<b>ADD</b>	Arquivo_Remoto Caminho_Absoluto_no_Container
<b>RUN</b>	Comando
<b>EXPOSE</b>	Porta do serviço
<b>CMD</b>	Comando executado ao iniciar o Container

O arquivo de Dockerfile não é case-sensitive, no entanto por convenção utilizamos os parâmetros em maiúsculo para que sua leitura seja mais agradável e de fácil compreensão. O nome do arquivo deve se chamar **Dockerfile** apenas com a letra inicial **D** em maiúsculo.

O Docker executará as instruções do Dockerfile em ordem (Top-down) e deverá sempre iniciar com a instrução **FROM**, as linhas que começam com **#** são tratadas como comentário a menos que a linha seja uma diretiva de analisador válida, o caractere **#** em qualquer outro lugar em uma linha é tratado como um argumento.

### Definições

- **FROM** - Inicializa um novo estágio de compilação e define a imagem de base para instruções subsequentes;

- **COPY** - Copia arquivos ou diretórios de origem local adicionando-os a imagem do container;
- **ADD** - Similar ao parâmetro **COPY** porém possibilita que a origem seja uma URL bem como a alteração de permissionamento ao adicionar os arquivos a imagem do container;
- **RUN** - Executa os comandos em uma nova camada na parte superior a imagem atual, é uma boa prática combinar diversos comandos em um unico **RUN** utilizando de ; e && para a combinação, assim criando apenas uma camada;
- **EXPOSE** - Informa ao docker a porta na qual o container estará escutando enquanto estiver sendo executado, é possível especificar portas TCP e UDP, caso não seja declarado o tipo de porta, o padrão (TCP) é assumido.
- **CMD** - Só pode existir uma unica instrução deste tipo em um arquivo, o propósito desta instrução é prover os padrões para a execução do container, podendo ser um executável ou até mesmo opções para o executável definido na instrução **ENTRYPOINT**
- **ENTRYPOINT** - Possibilita configurar o container para rodar como um executável, o comando **docker run <image>** inicializará o container em seu entrypoint somado ao **CMD** se existente.

## Criando o Primeiro Dockerfile

Vamos criar um diretório para armazenar os dockerfiles e criar nosso primeiro Dockerfile

```
1 mkdir -p /root/dockerfiles/echo-container
2 cd /root/dockerfiles/echo-container
3 vim Dockerfile
```

```
1 FROM      alpine
2 ENTRYPOINT ["echo"]
3 CMD       ["--help"]
```

Para criar a imagem a partir do Dockerfile, utilizamos o comando **docker image build**

```
1 docker image build -t echo-container .
2 docker image ls
```

A opção **-t** significa TAG, após o nome da imagem:tag informamos qual o diretório em que o Dockerfile se encontra, utilizamos o ponto (.) para dizer que o Dockerfile está no diretório atual (**PWD**)

Execute o container com a imagem criada

```
1 docker container run --rm -it echo-container
```

A opção **-rm** informa que o container será apagado após cumprir seu papel

Execute o container com a imagem criada alterando seu CMD

```
1 docker container run --rm -it echo-container Container DevOps
```

Ao passar um parâmetro após o nome da imagem estamos alterando o CMD do container, anteriormente **echo -help** para **echo Container DevOps**

## Dockerfile - Servidor WEB

Vamos criar um diretório para armazenar o dockerfile para nosso servidor WEB e criar o Dockerfile

```
1 mkdir -p /root/dockerfiles/webserver
2 cd /root/dockerfiles/webserver
3 vim Dockerfile
```

```
1 FROM      ubuntu
2 RUN       apt-get update; \
3          apt-get install wget git apache2 -y -qq
4 EXPOSE    80
5 CMD       ["apachectl", "-D", "FOREGROUND"]
```

Crie a imagem

```
1 docker image build -t webserver .
2 docker image ls
```

## Enviando a imagem para o Dockerhub

### Criando uma conta no Dockerhub

Acesse o endereço <https://hub.docker.com> e clique em **Sign Up** para criar uma conta, preencha com os seus dados e clique em

The screenshot shows the Docker Identification sign-up page. At the top right is the Docker logo. On the left is a large, semi-transparent 3D cube icon. The main title is "Docker Identification". Below it, a message says "In order to get you started, let us get you a Docker ID. Already have an account? [Sign In](#)". There are three input fields: "Enter a Docker ID", "Password" with an eye icon, and "Email". Below these are three checkboxes: "I agree to Docker's [Terms of Service](#).", "I agree to Docker's [Privacy Policy](#) and [Data Processing Terms](#).", and "(Optional) I would like to receive email updates from Docker, including its various services and products.". A reCAPTCHA box is present with the text "I'm not a robot" and a checkbox. At the bottom is a large blue "Continue" button.

### Continue

Confirme o cadastro em seu e-mail e logue no **Docker Hub** através do link **Sign In**

Para enviar uma imagem para o dockerhub é necessário que sejam feitos 3 passos:

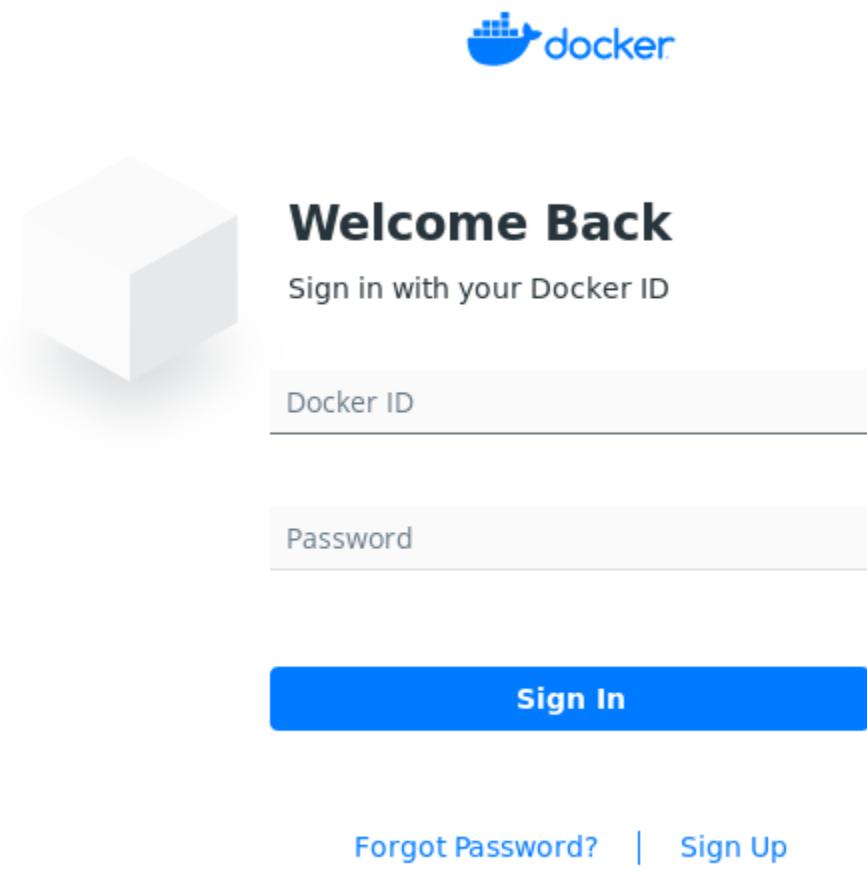


Figura 9.6: login

1. docker login
2. docker image tag
3. docker image push

Vamos efetuar o login com nossa conta no dockerhub

```
1 docker login -u <usuario_dockerhub>
```

Verifique se um arquivo de autorização foi criado

```
1 cat ~/.docker/config.json
```

Agora precisamos criar a tag da imagem, que deve seguir o padrão **usuario/imagem:versao**

```
1 docker image tag echo-container <usuario_dockerhub>/echo-container:latest
2 docker image tag webserver <usuario_dockerhub>/webserver
```

Caso não seja informada uma versão, o docker entende que a versão trata-se da **latest**

Agora podemos enviar as imagens para o dockerhub

```
1 docker image push <usuario_dockerhub>/echo-container
2 docker image push <usuario_dockerhub>/webserver
```

Ao finalizar, lembre-se de efetuar o logout em sua conta do dockerhub

```
1 docker logout
```

As imagens enviadas podem ser visualizadas na sua página do dockerhub e todos os usuários podem efetuar o download da mesma com o comando **docker image pull usuario/imagem:versao** desde que o repositório seja público

## Docker Volumes e Network

### Docker Volumes

Volume é um diretório especialmente designado, seja em um ou mais containers que compartilham o sistema de arquivos UnionFS

Os volumes são projetados para manter os dados, independentemente do ciclo de vida do container. O Docker nunca exclui um volume automaticamente quando você remove um container.

Existem 3 tipos de Volumes:

- **host** - Reside no sistema de arquivos do host do docker e pode ser acessado dentro de um container;
- **nomeado** - Volume gerenciado e criado pelo Docker, na criação do volume é dado um nome para o mesmo.
- **anônimo** - Volume gerenciado e criado pelo Docker, na criação do volume não é informado o nome para o mesmo e o Docker se encarrega de nomeá-lo com uma hash de 64 caracteres.

Existem alguns tipos de montagem para os volumes. O volume do tipo **host** tem sua montagem realizada como um **bind mount** e existe ainda um volume do tipo **tmpfs mount** o qual reside na memória do sistema (volátil)

Existem algumas vantagens ao se utilizar volumes do Docker ao invés de **bind mounts**

- Volumes são mais fáceis de efetuar o backup ou migração que bind mounts;
- É possível efetuar a gerência de volumes utilizando o Docker CLI ou o Docker API;
- Volumes podem ser compartilhados de maneira mais segura entre múltiplos containers;

 **caiodelgadonew** [Edit profile](#)

Community User São Paulo Joined March 9, 2019

[Repositories](#) [Starred](#) [Contributed](#)

Displaying 3 of 3 repositories

 [caiodelgadonew/webserver](#) 1 Download  
By [caiodelgadonew](#) • Updated a few seconds ago  
[Container](#)

 [caiodelgadonew/echo-container](#) 1 Download  
By [caiodelgadonew](#) • Updated 2 minutes ago  
[Container](#)

Figura 9.7: Repositório

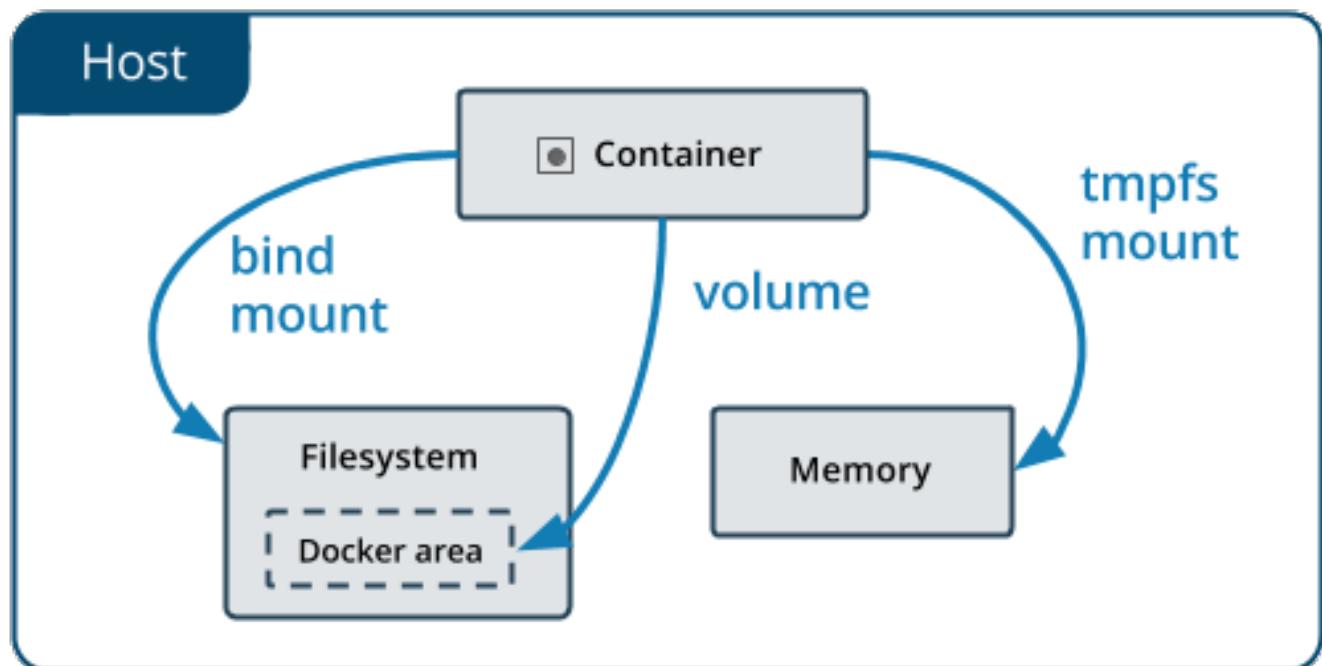


Figura 9.8: Mounts

- Drivers de Volumes podem habilitar o armazenamento em hosts remotos ou provedores de cloud, criptografar o conteúdo ou adicionar novas funcionalidades;
- Novos volumes podem ter seu conteúdo pré-populados por um container.

## Gerenciar Volumes

Para utilizar volumes no docker utilizamos a opção **-v** ou **--volume** para indicar qual volume deve ser montado no container.

Primeiramente vamos montar um volume do tipo host através de um **bind mount**

```
1 docker container run -dit --name servidor -v /srv:/srv ubuntu
2 docker container exec servidor df -Th
```

O parâmetro **-v** seguido de um diretório em caminho absoluto ou relativo separado por : (dois pontos) a um caminho absoluto faz com que o diretório **/srv** da máquina hospedeira seja montado dentro do container gerado.

Vamos efetuar a cópia de alguns arquivos para o volume, verificando o conteúdo da pasta antes e após a execução

```
1 docker container exec servidor ls -lR /srv
2 cp -r /root/dockerfiles /srv
3 docker container exec servidor ls -lR /srv
```

*Note que os arquivos copiados para a pasta /srv do host estão sendo exibidos também na pasta /srv do container*

Vamos remover o container e cria-lo novamente utilizando um volume anônimo

```
1 docker container rm -f servidor
2 docker container run -dit --name servidor -v /volume ubuntu
```

Podemos utilizar o comando **docker container inspect** para verificar o local e nome do volume criado

```
1 docker container inspect servidor | grep volume
```

*Por padrão, os volumes criados e gerenciados pelo docker se localizam no diretório /var/lib/docker/volumes*

Podemos também visualizar informações do volume através do comando **docker volume inspect**

```
1 docker volume ls
2 docker volume inspect <hash>
```

Com o comando **docker volume create** é possível criar volumes *nomeados* que serão gerenciados da mesma forma que o volume anônimo.

```
1 docker volume create <nome_volume>
```

Visualize o volume criado no container

```
1 docker container exec servidor df -Th
2 docker container exec servidor ls -lR /volume
```

Copie os arquivos no host hospedeiro e verifique o conteúdo no container

```
1 cp -r /root/dockerfiles /var/lib/docker/volumes/<hash>/_data
2 docker container exec servidor ls -lR /volume
```

Remova o container servidor

```
1 docker container rm -f servidor
```

Para remover um volume no docker podemos utilizar a opção **rm**

```
1 docker volume ls
2 docker volume rm <hash>
3 docker volume rm $(docker volume ls -q)
```

O comando **docker volume ls -q** lista os volumes por id > **ATENÇÃO:** a remoção do volume através do comando **docker volume rm** faz com que o volume seja excluído, não sendo possível a recuperação dos dados.

## Docker Networks

Uma das razões para que o container seja tão poderoso é a possibilidade de conectá-lo uns aos outros ou até mesmo a outros servidores físicos ou virtuais através de redes.

Existem alguns tipos de **Drivers de Rede**, trataremos as redes **Bridge** e **Host** neste curso.

**Bridge** - É a rede padrão do Docker, caso o container seja criado sem especificar um driver de rede a rede do tipo bridge é criada. Esta rede é criada utilizando um driver de rede de ponte que instancia uma interface de rede no linux chamada de **dockerX** sendo X um numero. Por padrão a rede bridge conta com um serviço interno de DNS, ou seja, os containers respondem internamente através de seu nome. Também é possível conectar e desconectar containers on-the-fly.

**Host** - É a rede onde o IP da máquina é compartilhado com o container, nesta rede não é possível existir dois containers utilizando a mesma porta de rede.

## Administrando Redes

Por padrão quando um container é criado ele não publica nenhuma de suas portas para o mundo externo. Para disponibilizar uma porta para serviços fora do ambiente Docker ou para containers que não estejam conectados a mesma rede, utilizamos a flag **-publish** ou **-p**. Isso cria uma regra de firewall que mapeia uma porta de container para uma porta do host.

Quando executamos um container, podemos definir a publicação de uma ou mais portas através da opção **-p hostaddress:hostport:containerport**. Para expor as portas em qualquer endereço utilizamos o hostaddress **0.0.0.0** ou apenas subtraímos esse valor.

Vamos subir nosso container servidor-web expondo a porta para a nossa rede

```
1 docker container run -dit --name webserver -p 80:80 webserver
```

Podemos verificar a porta em uso por um container através do comando **docker container port**

```
1 docker container ls
2 docker container port webserver
```

Podemos acessar a aplicação no navegador pelo endereço <http://container.4labs.example>

Remova o container webserver

```
1 docker container rm -f webserver
```

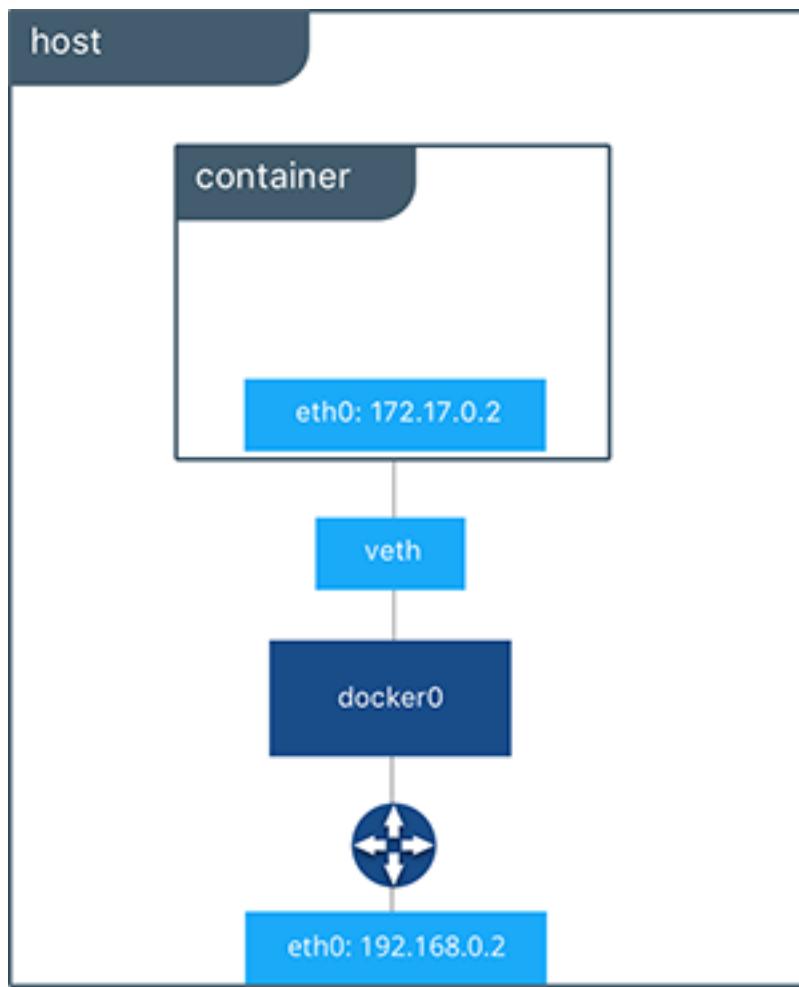


Figura 9.9: Rede Bridge

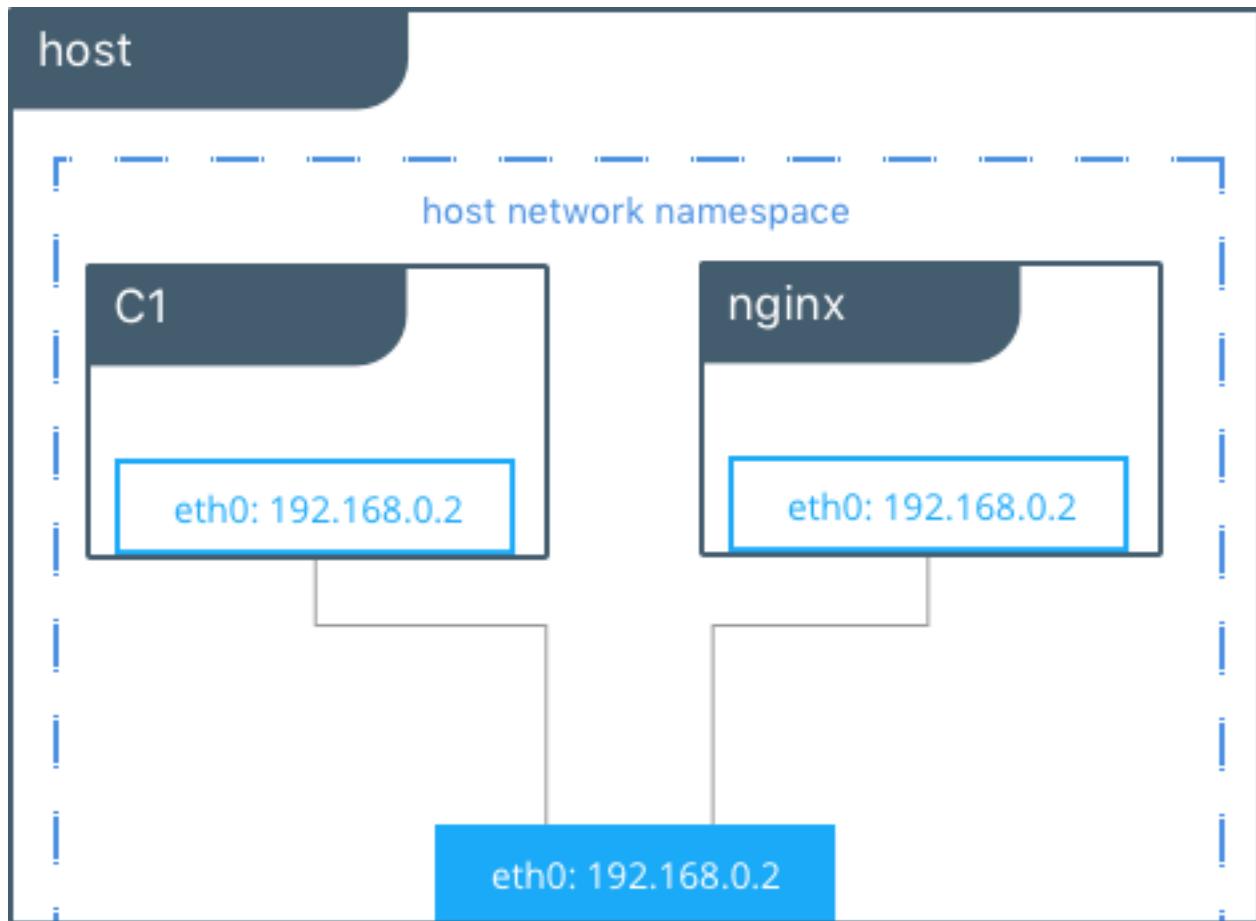


Figura 9.10: Rede Host

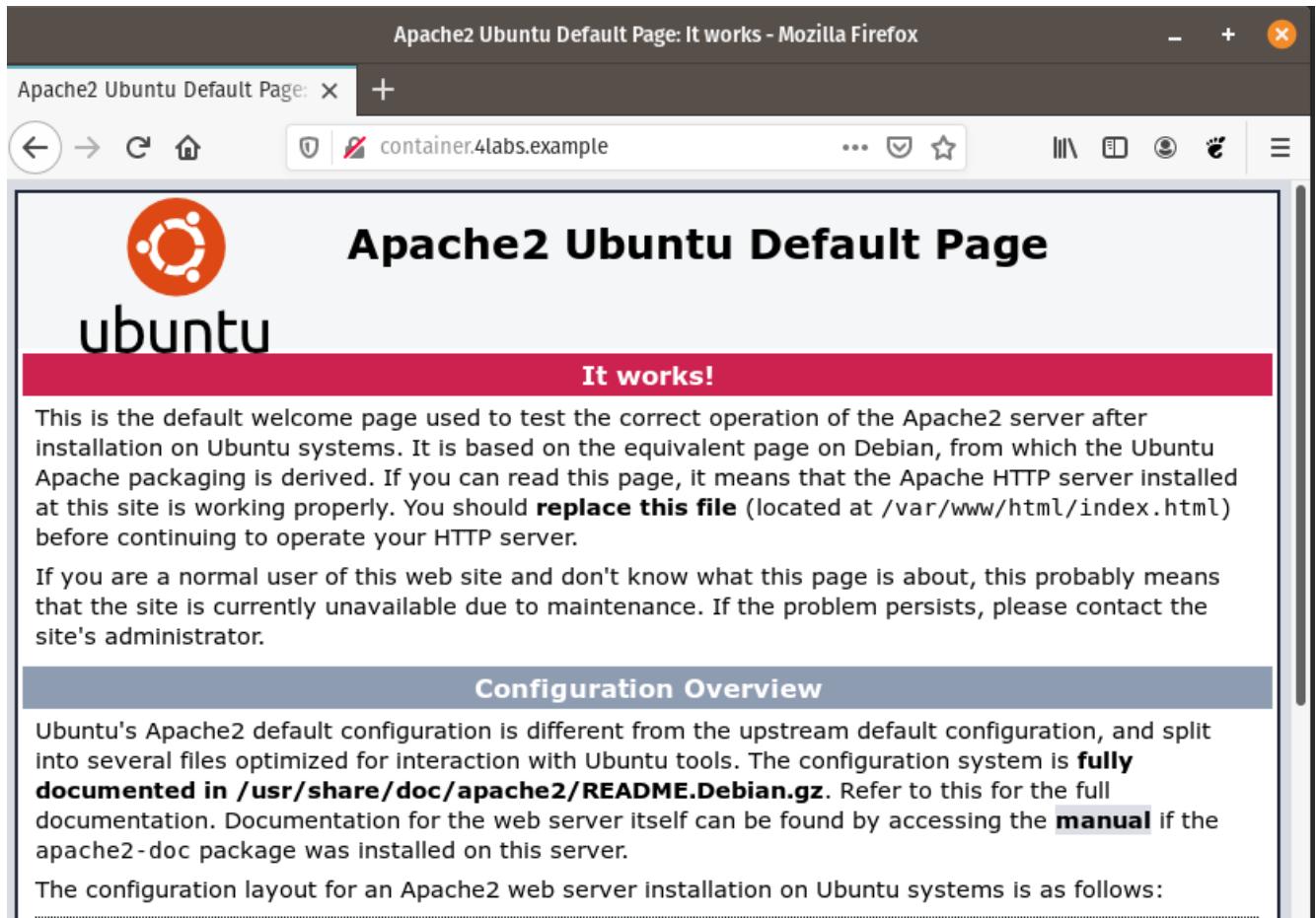


Figura 9.11: webserver

## Executando o Container com Rede Bridge

Para executar um container com uma rede bridge podemos passar a flag **--network bridge** no comando docker run

```
1 docker container run -dit --name webserver --network bridge -p 80:80 webserver
```

Verifique o serviço na porta 80 do host

```
1 ss -ntpl | grep 80
```

O processo que é executado na porta através da rede bridge é o *docker-proxy*

Remova o container webserver

```
1 docker container rm -f webserver
```

## Executando o Container com Rede Host

Para executar um container com a rede host podemos passar a flag **--network host** no comando docker run

```
1 docker container run -dit --name webserver --network host webserver
```

Note que ao utilizar o drive de rede host não é necessário publicar a porta uma vez que não existe uma ponte de conexão

Verifique o serviço na porta 80 do host

```
1 ss -ntpl | grep 80
```

O processo que é executado na porta através da rede host é o próprio processo interno do container, neste caso o *apache2*. Mesmo com a utilização da rede host ainda ocorre o isolamento de namespace a nível de processos e usuários

Remova o container webserver

```
1 docker container rm -f webserver
```

## Conectando Containers

Configure dois containers para que seja possível efetuar testes e modificações na rede.

```
1 docker container run -dit --name c1 -h servidor debian
2 docker container run -dit --name c2 -h servidor debian
```

Verifique o endereço IP dos containers

```
1 docker container exec c1 ip -c a show eth0
2 docker container exec c2 ip -c a show eth0
```

Faça um teste de conectividade entre os containers através do comando ping

```
1 docker container exec c1 ping -c4 172.17.0.3
2 docker container exec c2 ping -c4 172.17.0.2
```

Remova os containers c1 e c2

```
1 docker container rm -f c1 c2
```

Para criar uma nova rede, podemos utilizar o comando **docker network create**

```
1 docker network create --driver bridge --subnet 172.19.0.0/16 4labs-lan
2 docker network ls
```

Com o comando acima criamos a rede 4labs-lan especificando o endereçamento de rede

Para detalhar a rede podemos utilizar o parâmetro **inspect**

```
1 docker network inspect 4labs-lan
```

Crie o container c1 e c2 adicionando-os a rede 4labs-lan

```
1 docker container run -dit --name c1 -h servidor --network 4labs-lan debian
2 docker container run -dit --name c2 -h client --network 4labs-lan debian
```

Faça um teste de conectividade através do comando ping

```
1 docker container exec c1 ping -c4 c2
2 docker container exec c2 ping -c4 c1
```

Remova os containers c1 e c2

```
1 docker container rm -f c1 c2
```

Também é possível criar o container especificando seu endereço IP

```
1 docker container run -dit --name c1 -h servidor --network 4labs-lan --ip 172.19.0.111 debian
2 docker container run -dit --name c2 -h client --network 4labs-lan --ip 172.19.0.112 debian
```

A opção **-ip** só pode ser utilizada em uma rede definida pelo usuário

Faça um teste de conectividade através do comando ping

```
1 docker container exec c1 ping -c4 c2
2 docker container exec c2 ping -c4 c1
```

Para desconectar uma rede de um container podemos utilizar o parâmetro **disconnect**

```
1 docker network disconnect 4labs-lan c2
2 docker container exec c1 ping -c4 c2
```

Para conectar a rede novamente utilizamos o parâmetro **connect** podendo informar o endereço IP através do parâmetro **-ip**

```
1 docker network connect --ip 172.19.0.222 4labs-lan c2
2 docker container exec c1 ping -c4 c2
```

Remova os containers c1 e c2 e a rede 4labs-lan

```
1 docker container rm -f c1 c2
2 docker network rm 4labs-lan
```

## Docker Compose

### O que é o Compose

Compose é uma ferramenta para definir e executar aplicações multi-containers em Docker. Com um simples comando é possível iniciar todos os serviços através de sua configuração.

O Compose funciona em todos os ambientes: produção, homologação, desenvolvimento, teste e fluxos de integração contínua.

### Etapas de um Compose

Utilizar o Compose é um processo de três etapas:

1. Definir seu ambiente com um Dockerfile para que ele possa ser reproduzido em qualquer lugar.
2. Definir os serviços que serão executados em um arquivo docker-compose.yml
3. Executar o comando docker-compose up

### docker-compose.yml

O arquivo **docker-compose.yml** define a estrutura e como serão executados os containers.

É importante notar que as versões do arquivo docker-compose podem não ser compatíveis com determinados parâmetros e até mesmo versões diferentes da Docker Engine.

Podemos visualizar as referências do arquivo compose diretamente na documentação oficial do Docker na seção Compose File Reference

O docker-compose.yml é apresentado da seguinte maneira:

```
1 version: '3'
2
3 volumes:
4   logvolume01: {}
5
6 networks:
7   app-net:
8
9 services:
10   web:
11     build: .
12     ports:
13       - 5000:5000
14     volumes:
15       - logvolume01:/var/log
16     network:
17       - app_net
18
19   redis:
20     image: redis
21     network:
22       - app_net
```

Obrigatóriamente o docker-compose.yml deve começar indicando qual a versão do compose a ser utilizada, em seguida podemos definir os serviços, volumes e redes não importando a ordem. Cada container a ser executado é uma nova seção do grupo **services**.

## Criando Docker-Compose

Crie um diretório para armazenarmos nosso compose

```
1 mkdir /root/compose  
2 cd /root/compose
```

Copie o Dockerfile do nosso webserver para a pasta compose e crie o arquivo docker-compose.yml

```
1 cp /root/dockerfiles/webserver/Dockerfile .  
2 vim docker-compose.yml
```

```
1 version: '3'  
2  
3 services:  
4   webserver:  
5     build: .  
6     hostname: webserver  
7     ports:  
8       - 80:80  
9     restart: always
```

### Parâmetros do docker-compose.yml

- **version** - define a versão do arquivo compose
- **services** - define a seção de serviços
- **build** - define o local do Dockerfile
- **hostname** - define o hostname do container
- **ports** - define quais portas serão publicadas
- **restart** - define a política de restart

Execute a criação do ambiente

```
1 docker-compose up -d
```

O comando `docker-compose up` com o parâmetro `-d` inicia o compose de forma **detached**

**IMPORTANTE:** por padrão o comando `docker-compose` procura por um arquivo `docker-compose.yml` no diretório em que o comando foi executado, é possível informar o arquivo compose a ser utilizado através da opção `-f`

Podemos acessar a aplicação no navegador pelo endereço `http://container.4labs.example`

Para listar os containers criados pelo compose podemos utilizar o parametro **ps**

```
1 docker-compose ps
```

Caso seja necessário parar o container criado com o compose podemos utilizar o parâmetro **stop**

```
1 docker-compose stop  
2 docker-compose ps
```

Para iniciar novamente o compose podemos utilizar o parâmetro **start**

```
1 docker-compose start  
2 docker-compose ps
```

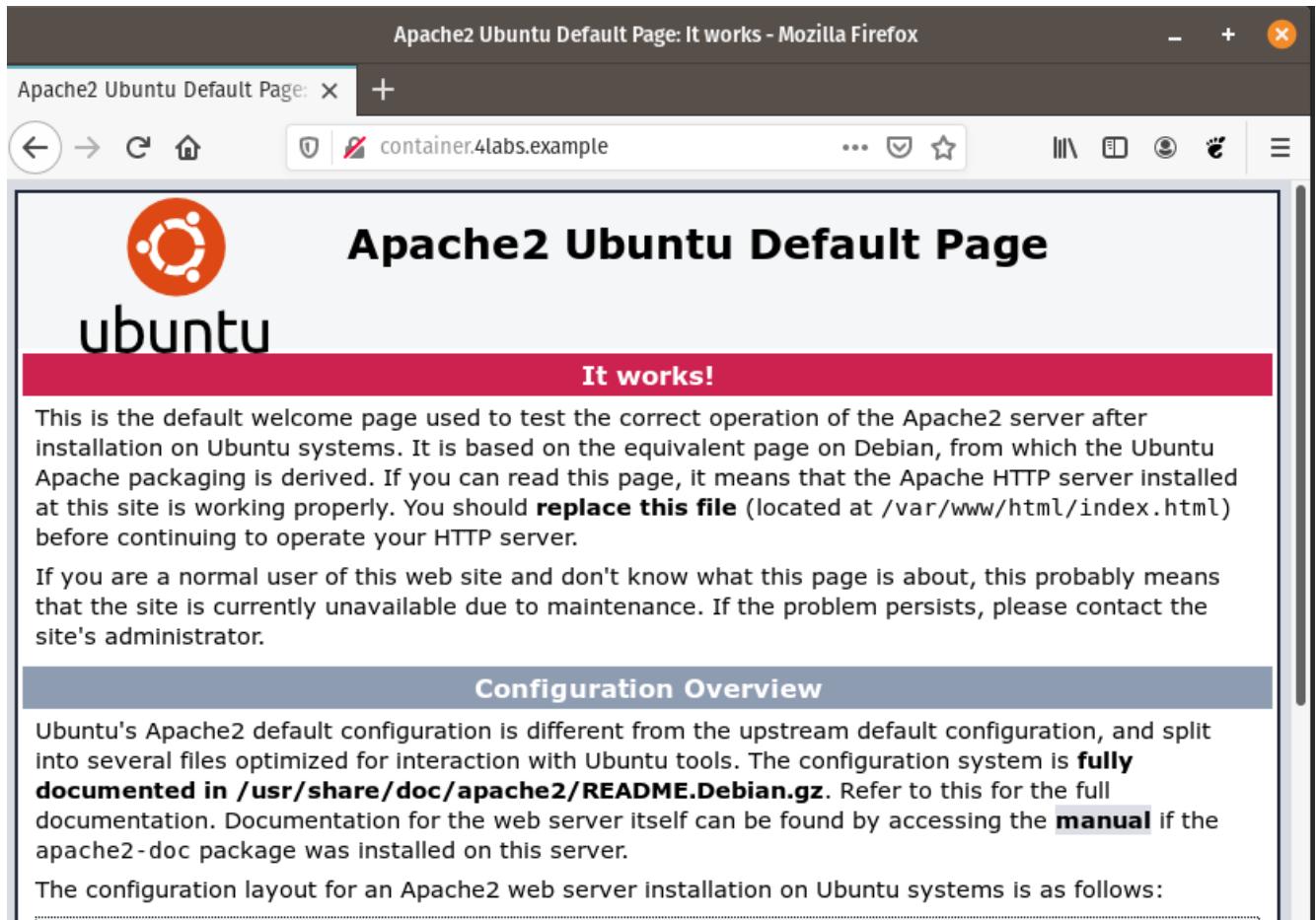


Figura 9.12: webserver

Para destruir o ambiente criado através do docker-compose utilizamos o parâmetro **down**

```
1 docker-compose down
2 docker-compose ps
```

Vamos agora criar uma página web para ser exibida em nosso container através do docker-compose, crie a pasta html e o arquivo index.html

```
1 mkdir html
2 echo "<h1> Website 4Labs </h1>" > html/index.html
```

Edite o compose e adicione o mapeamento de volume:

```
1 vim docker-compose.yml
```

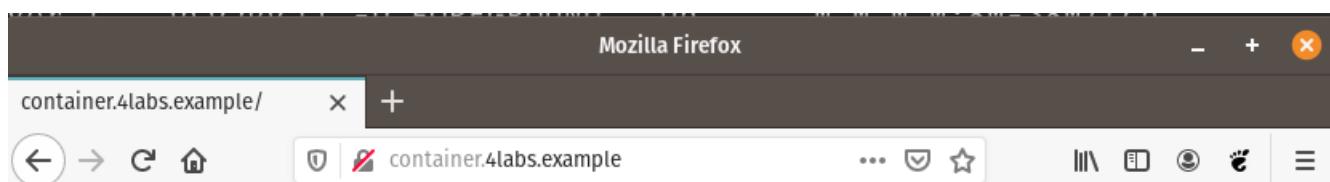
```
1 version: '3'
2
3 services:
4   webserver:
5     build: .
6     hostname: webserver
7     ports:
8       - 80:80
9     restart: always
10    volumes:
11      - $PWD/html:/var/www/html
```

O parâmetro **\$PWD** referencia o Parent Working Directory (diretório atual)

Através deste mapeamento de volumes estamos substituindo o conteúdo do caminho /var/www/html por /root/compose/. Execute a criação do ambiente

```
1 docker-compose up -d
```

Acesse a aplicação no navegador pelo endereço <http://container.4labs.example>



**Website 4Labs**

Figura 9.13: webserver

Para destruir o ambiente criado

```
1 docker-compose down
```

## Compose Multi-containers

A principal funcionalidade de um compose é subir serviços multi-containers como, por exemplo, um serviço de wordpressm onde será executado um container para a aplicação e um segundo container para o banco de dados. Isto é possível com a definição de múltiplos segmentos dentro de um docker-compose

Crie o arquivo wordpress-compose.yml

```
1 vim wordpress-compose.yml
```

```
1 version: '3'
2
3 volumes:
4   mysql_db:
5
6 services:
7   wordpress:
8     image: wordpress
9     restart: always
10    ports:
11      - 80:80
12    environment:
13      WORDPRESS_DB_HOST: db
14      WORDPRESS_DB_USER: wpuser
15      WORDPRESS_DB_PASSWORD: devops@4linux
16      WORDPRESS_DB_NAME: wordpress
17   db:
18     image: mysql:5.7
19     restart: always
20     volumes:
21       - mysql_db:/var/lib/mysql
22     environment:
23       MYSQL_DATABASE: wordpress
24       MYSQL_USER: wpuser
25       MYSQL_PASSWORD: devops@4linux
26       MYSQL_RANDOM_ROOT_PASSWORD: '1'
```

Execute a criação do ambiente

```
1 docker-compose -f wordpress-compose.yml up -d
2 docker-compose -f wordpress-compose.yml ps
```

O parâmetro **-f** especifica o arquivo **.yml** a ser utilizado com o compose

Acesse o endereço <http://container.4labs.example> e configure o wordpress

Vamos alterar o tema do Wordpress, para isto clique em **Aparência** e em seguida em **Temas**

Selecione um tema e clique em **Ativar**

Para visualizar o tema clique em **Wordpress - 4Labs**

Podemos verificar os logs do compose

```
1 docker-compose -f wordpress-compose.yml logs
```



Figura 9.14: wordpress

## Bem-vindo (a)

Bem-vindo (a) à famosa instalação do WordPress em cinco minutos! Basta preencher as informações abaixo e você estará a poucos passos de usar a plataforma de publicação mais extensível e poderosa do mundo.

### Informação necessária

Forneça as seguintes informações. Não se preocupe, você pode alterar estas configurações mais tarde.

**Título do site**

Wordpress - 4Labs

**Nome de usuário**

analista

Nomes de usuário podem ter somente caracteres alfanuméricos, espaços, sublinhados, hífens, pontos e o símbolo @.

**Senha**

devops@4linux

Forte

 Esconder

**Importante:** Você precisará dessa senha para entrar. Guarde-a em um local seguro.

**O seu e-mail**

analista@4labs.example|

Verifique o seu endereço de e-mail antes de prosseguir.

**Visibilidade nos mecanismos de busca**

Evitar que mecanismos de busca indexem este site

Cabe aos mecanismos de busca atender esta solicitação.

**Instalar WordPress**



Figura 9.15: wordpress

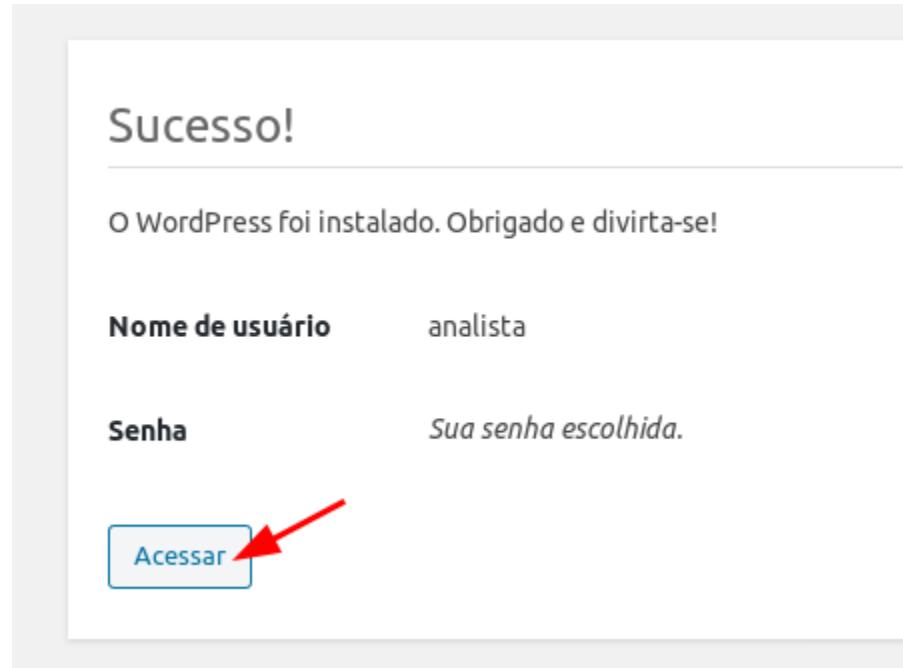


Figura 9.16: wordpress



Figura 9.17: wordpress

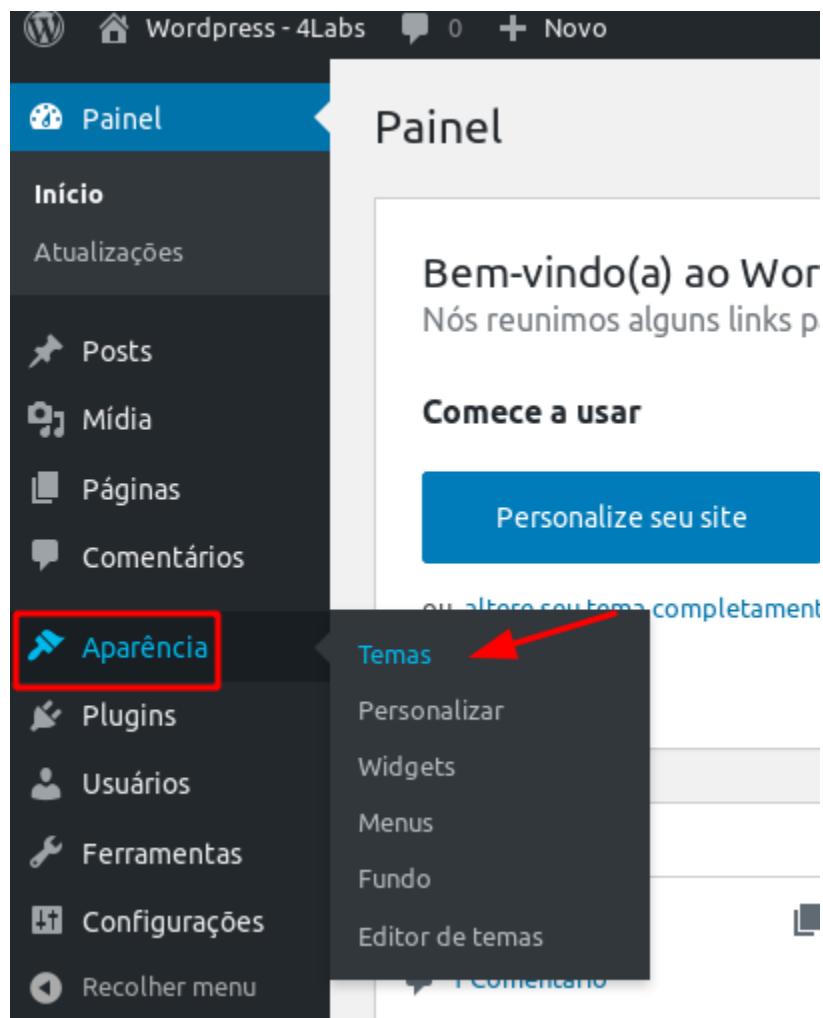


Figura 9.18: wordpress

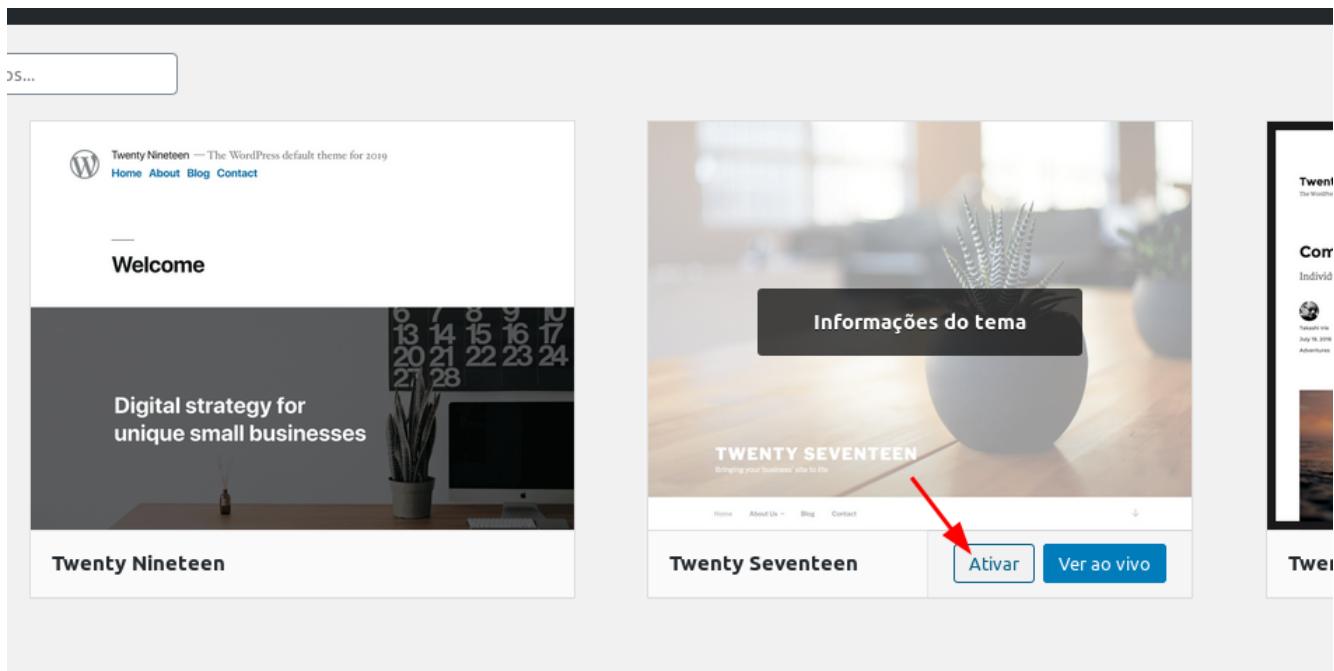


Figura 9.19: wordpress

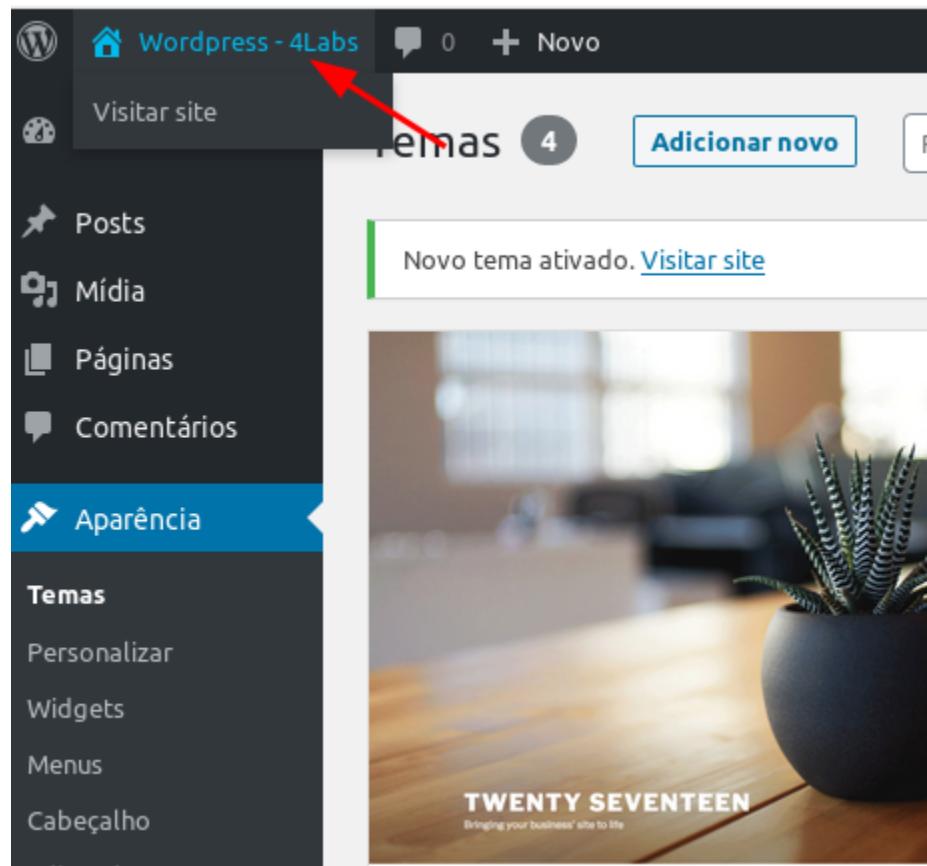


Figura 9.20: wordpress

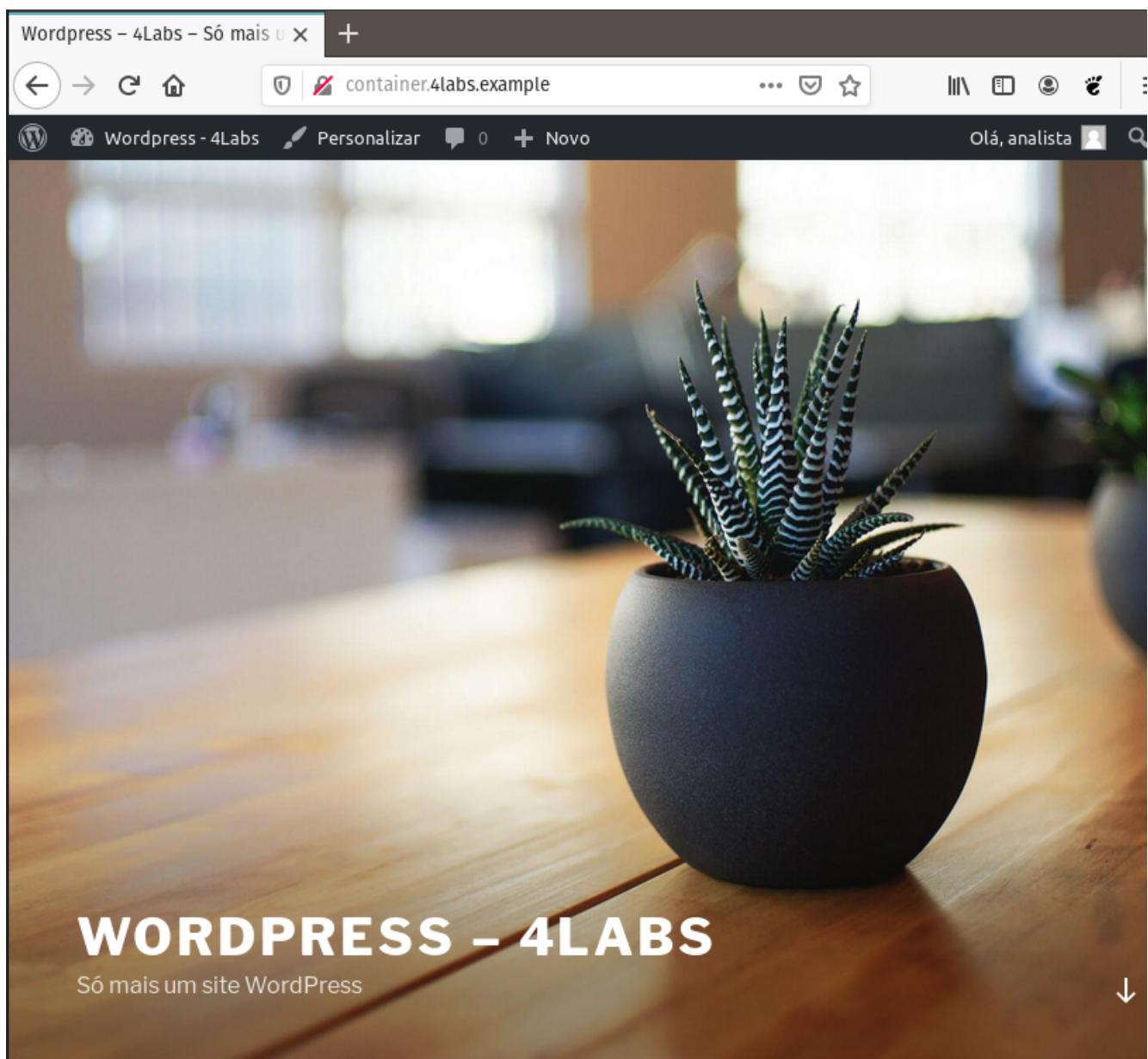


Figura 9.21: wordpress

Para visualizar os logs em tempo real podemos utilizar o parâmetro **-f**

```
1 docker-compose -f wordpress-compose.yml logs -f
```

Pressione <CTRL> + <C> para sair

Destrua o ambiente

```
1 docker-compose -f wordpress-compose.yml down
```

Note que ao destruir o compose os volumes continuam persistentes no disco, sendo possível recriar a aplicação com o mesmo conteúdo

Tente acessar o website e verifique que o mesmo está indisponível

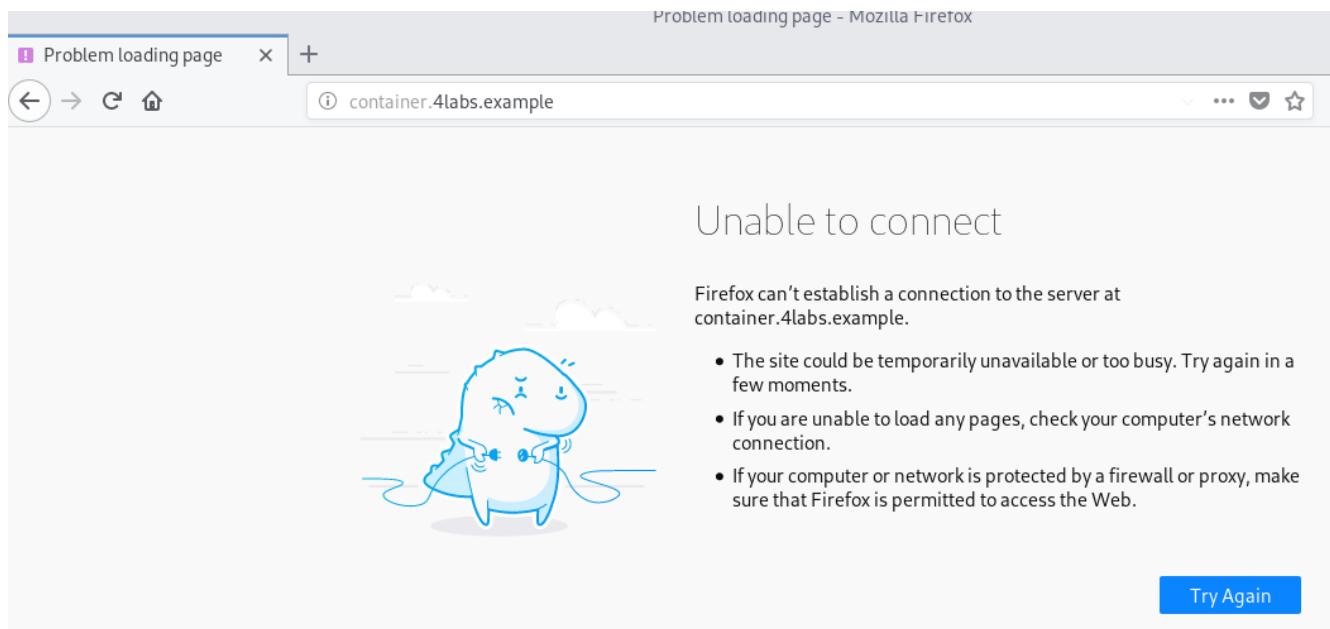


Figura 9.22: wordpress

Execute a criação do ambiente novamente

```
1 docker-compose -f wordpress-compose.yml up -d
```

Acesse o endereço <http://container.4labs.example> e verifique que os dados persistiram na aplicação

Destrua o ambiente e remova o volume criado

```
1 docker-compose -f wordpress-compose.yml down --volumes
```

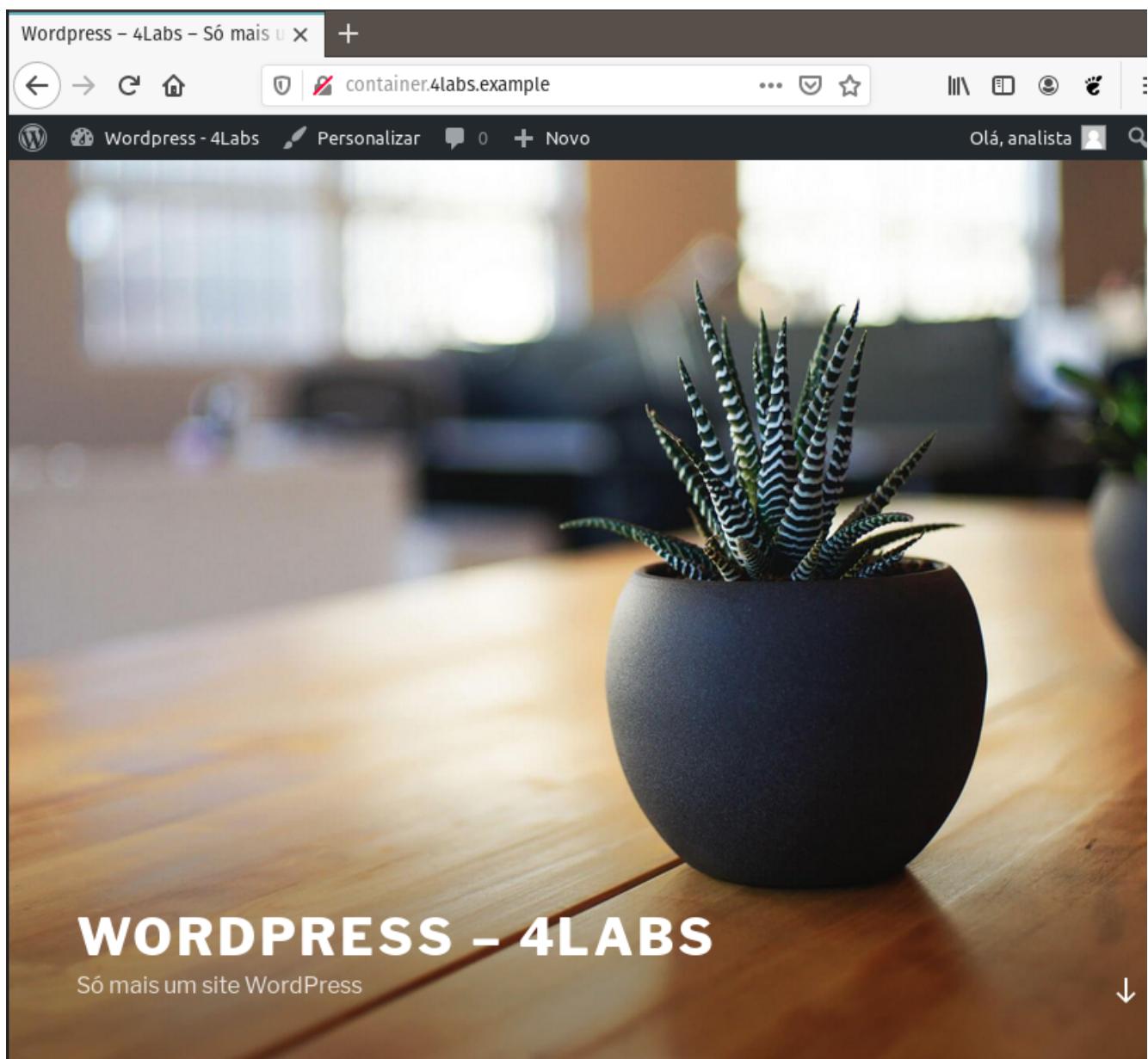


Figura 9.23: wordpress

## Capítulo 10

# Teste de Infraestrutura com InSpec

### Características do InSpec

#### O que é o InSpec



Figura 10.1: InSpec

InSpec é um framework gratuito e Open Source para teste e auditoria de aplicações e infraestrutura.

Podemos dizer que o InSpec é uma ferramenta de Compliance-as-a-Code

### Características do InSpec

- Não necessita de um agent instalado
- Utiliza como base a linguagem Ruby
- Possui mais de 400 recursos sendo boa parte para tratamento das Clouds mais utilizadas no mercado (AWS, GCP, Azure)
- Documentação extremamente detalhada
- Possui um shell interativo
- Funciona em Linux, macOS e Windows

### InSpec Shell

O InSpec Shell é um shell interativo utilizado para executar rapidamente controles e testes do InSpec sem precisar escrever um arquivo.

Podemos utilizar também o InSpec Shell para verificar os possíveis comandos aceitos pelo InSpec e conectar remotamente via SSH transmitindo o InSpec Shell ao computador remoto.

Acesse a máquina compliance e troque para o usuário root

```
1 vagrant ssh compliance
2 sudo su -
```

Acesse o InSpec Shell

```
1 inspec shell
```

A primeira vez que o `inspec shell` for executado por um usuário é mostrado os termos de uso solicitando o aceite, digite `yes` para aceitar a os termos de licenciamento do `inspec`

Note que ao executar o `inspec shell` nossa PS1 será modificada para `inspec>` informando que estamos dentro do shell do `inspec`

Visualize a ajuda através do comando `help`

```
1 help
```

Utilize o comando `help resources` para verificar os recursos disponíveis.

```
1 help resources
```

Para exibir a ajuda para um recurso utilize o comando `help`

```
1 help file
```

Para utilizar um determinado recurso precisamos definir o conteúdo do mesmo através do comando `<recurso>('conteúdo')`  
Crie um recurso com o arquivo `/etc/hosts`

```
1 file('/etc/hosts')
```

Para visualizar as sub-rotinas de um recurso adicionamos o parâmetro `.methods`

```
1 file('/etc/hosts').methods
```

É possível navegar entre os métodos utilizando as setas direcionais, para sair do comando digite `q`

Vamos utilizar o recurso `file` com `methods` para verificar se um arquivo existe e qual seu conteúdo

```
1 file('/etc/hosts').exist?
2 file('/etc/hosts').content
3 file('/etc/hosts').mode
```

Utilizando o recurso `Package`, podemos verificar informações de pacotes existentes

```
1 package('git').installed?
2 package('git').version
3 package('git').info
```

## Matchers

Para trabalhar com o InSpec, utilizamos comparadores (matchers)

Liste os comparadores com o comando **help matchers**

```
1 help matchers
```

Matcher	Uso
be	Comparação de valores numéricos
eq	Igualdade entre dois valores
cmp	Igual ao eq porém menos restritivo
include	Verifica se o valor está incluso na lista
match	Verifica um texto utilizando uma RegEX

*Informações e Exemplos de comparadores podem ser encontrados diretamente na documentação oficial do InSpec em matchers*

## Controls

O InSpec utiliza de **controls** para expressar expectativas com relação ao estado de um recurso, como por exemplo: propriedades, condições, comparadores e resultados esperados.

Vamos descrever um controle para verificar se o arquivo hosts existe.

```
1 describe file('/etc/hosts') do
2   it { should exist }
3 end
```

É possível executar uma série de controls em um único recurso

```
1 describe file('/etc/hosts') do
2   it { should exist }
3   it { should be_file }
4   it { should be_owned_by 'root'}
5   its('mode') { should eq '0644'}
6 end
```

O teste `its('mode') { should eq '0644'}` falhou pois utilizamos o `eq`, o qual realiza um teste literal do conteúdo, podemos utilizar o comparador `cmp` o qual tentará encaixar o valor atual ao tipo que estamos comparando. Neste caso estamos utilizando um **OCTAL** e comparando a um **DECIMAL**, o `cmp` irá igualar as unidades e efetuar a comparação.

```
1 describe file('/etc/hosts') do
2   it { should exist }
3   it { should be_file }
4   it { should be_owned_by 'root'}
5   its('mode') { should cmp '0644'}
6 end
```

Vamos escrever um teste para verificar se o pacote git está instalado

```
1 describe package('git') do
2   it { should be_installed }
3   its('version') { should eq '1:2.17.1-1ubuntu0.4'}
4 end
```

Para sair do Inspec Shell, utilize o comando **exit**

```
1 exit
```

## Executando o InSpec Shell em hosts remotos

Podemos executar o InSpec Shell em um host remoto, isto faz com que os comandos sejam transferidos via SSH, não sendo necessário que o destino possua o InSpec instalado.

Podemos acessar utilizando usuário e senha ou chaves ssh

```
1 inspec shell -t ssh://<usuario>@<host> --password <senha>
2 inspec shell -t ssh://<usuario>@<host> -i <private_key>
```

Vamos conectar a máquina container para efetuar alguns testes

```
1 inspec shell -t ssh://container.4labs.example -i /root/.ssh/id_rsa
```

Verifique informações sobre o Docker CE

```
1 package('docker-ce').installed?
2 package('docker-ce').version
```

Vamos escrever um teste para verificar se o Docker CE está instalado e sua versão é superior a 19

```
1 describe package('docker-ce') do
2   it { should be_installed }
3   its('version') { should cmp > '19' }
4 end
```

Diferente do comparador **eq** o comparador **cmp** pode receber sinais de maior, menor ou igual. No caso do nosso teste a versão retornada foi a **19.03.4-3.el7**, o **cmp** consegue compreender que o valor **19.03...** é maior que **19**

Podemos também escrever um teste para verificar se o serviço do docker está habilitado, instalado e em execução

Execute o teste para verificar o serviço do docker

```
1 describe service('docker') do
2   it { should be_installed }
3   it { should be_enabled }
4   it { should be_running }
5 end
```

Após os testes podemos fechar nosso InSpec Shell Remoto

```
1 exit
```

# Escrevendo Testes

## Estrutura de um Profile

Profiles são utilizados pelo InSpec para organizar múltiplos controles em um artefato reutilizável, o qual pode ser descrito e versionado

Ao criarmos um profile, uma série de arquivos e diretórios são criados

```
1 profile/
2   README.md
3   controls
4     example.rb
5   inspec.yml
6   libraries
```

- **README.md** - Arquivo em Markdown para explicações e comentários sobre o profile
- **controls** - Diretório onde devem ser inseridos os testes
- **inspec.yml** - Arquivo de metadados
- **libraries** - Diretório para armazenar as bibliotecas necessárias para um profile

## Criando Profiles

Vamos criar um diretório para armazenar nossos profiles

```
1 mkdir /root/inspec-profiles
2 cd /root/inspec-profiles
```

Para criar a estrutura do nosso profile podemos utilizar o comando **inspec init profile <profile\_name>**

```
1 inspec init profile compliance
```

Será exibida uma mensagem sobre a inicialização do profile e arquivos que foram criados

```
1                     InSpec Code Generator
2
3 Creating new profile at /root/inspec-profiles/compliance
4   • Creating file README.md
5   • Creating file inspec.yml
6   • Creating directory controls
7   • Creating file controls/example.rb
8   • Creating directory libraries
```

Edite o arquivo **README.md** e adicione algumas informações sobre o profile

```
1 vim compliance/README.md
```

```
1 # InSpec Profile - Compliance
2 Este profile tem como objetivo verificar se as aplicações necessárias para um servidor de compliance
   estão instaladas e seus serviços em execução
3
4 ## Controles
5 Neste arquivo são efetuados testes com:
6
7 ### **Pacotes**:
8 - InSpec
9 - Puppet Server
```

```
10
11 ### **Serviços**
12
13 - Puppet Server
```

Atualize o arquivo de metadados

```
1 vim compliance/inspec.yml
```

```
1 name: compliance
2 title: InSpec Profile - Compliance
3 maintainer: Analista DevOps
4 copyright: 4labs
5 copyright_email: analista@4labs.example
6 license: Apache-2.0
7 summary: An InSpec Compliance Profile
8 version: 0.1.0
9 supports:
10   os-family: linux
```

Os controles serão descritos no diretório **controls**. Por padrão todo profile criado possui um arquivo **example.rb** contendo exemplos da aplicação de controles em um profile.

Vamos visualizar e entender o que está acontecendo no arquivo de exemplo

```
1 cat compliance/controls/example.rb
```

```
1 # copyright: 2018, The Authors
2
3 title "sample section"
4
5 # you can also use plain tests
6 describe file("/tmp") do
7   it { should be_directory }
8 end
9
10 # you add controls here
11 control "tmp-1.0" do
12   impact 0.7
13   title "Create /tmp directory"
14   desc "An optional description..."
15   describe file("/tmp") do
16     it { should be_directory }
17   end
18 end
```

No arquivo de exemplo é informado o título (**title**) e os testes. Primeiramente é realizado um teste para verificar se o diretório **/tmp** existe. Em seguida é realizado um controle chamado de **tmp-01** o qual tem um índice de criticidade **0.7** (**impact**) uma descrição e o teste para verificar se o diretório **/tmp** existe

## Impact

O parâmetro **impact** (string ou valor numérico) em um controle descreve qual a criticidade do teste que está sendo realizado. Estes valores são baseados no CVSS 3.1 (**Common Vulnerability Scoring System**) o qual é um framework internacional para pontuação comum de vulnerabilidades.

Os valores possíveis são: \* **0.0 até < 0.01** ou **none** - Sem impacto, Informações \* **0.01 até < 0.4** ou **low** - Baixo impacto \* **0.4 até < 0.7** ou **medium** - Médio Impacto \* **0.7 até < 0.9** ou **high** - Alto Impacto \* **0.9 até 1.0** ou **critical** - Impacto Crítico

## Executando Profiles

Vamos executar o teste do exemplo na nossa máquina local para visualizar o resultado

```
1 inspec exec /root/inspec-profiles/compliance

1 Profile: InSpec Profile - Compliance (compliance)
2 Version: 0.1.0
3 Target: local://
4
5     tmp-1.0: Create /tmp directory
6         File /tmp should be directory
7
8     File /tmp
9         should be directory
10
11 Profile Summary: 1 successful control, 0 control failures, 0 controls skipped
12 Test Summary: 2 successful, 0 failures, 0 skipped
```

Para executar o profile em uma máquina remota podemos primeiramente detectar as configurações da máquina a fim de verificar se a mesma está acessível através do comando **inspec detect**

```
1 inspec detect -t ssh://automation.4labs.example -i /root/.ssh/id_rsa
```

```
1
2
3     Platform Details
4
5     Name:      centos
6     Families:  redhat, linux, unix, os
7     Release:   7.6.1810
8     Arch:      x86_64
```

Execute o teste

```
1 inspec exec /root/inspec-profiles/compliance -t ssh://automation.4labs.example -i /root/.ssh/id_rsa

1 Profile: InSpec Profile - Compliance (compliance)
2 Version: 0.1.0
3 Target: ssh://root@automation.4labs.example:22
4
5     tmp-1.0: Create /tmp directory
6         File /tmp should be directory
7
8     File /tmp
9         should be directory
10
11 Profile Summary: 1 successful control, 0 control failures, 0 controls skipped
12 Test Summary: 2 successful, 0 failures, 0 skipped
```

## Supermarket

O InSpec dispõe de um repositório de profiles chamado **supermarket**. Através dele podemos executar testes on-the-fly. Visualize os parâmetros do supermarket e liste os profiles disponíveis

```
1 inspec supermarket help
2 inspec supermarket profiles
```

Para exibir informações de um profile podemos executar o comando `inspec supermarket info <profile>`

```
1 inspec supermarket info dev-sec/linux-baseline
```

```
1 name:    linux-baseline
2 owner:   dev-sec
3 url:     https://github.com/dev-sec/linux-baseline
4
5 description: Linux compliance profile, used for Security + DevOps. More information is available
               at http://dev-sec.io
```

Podemos executar os testes utilizando profiles do supermarket de duas maneiras

```
1 inspec supermarket exec dev-sec/linux-baseline
2 inspec exec supermarket://dev-sec/linux-baseline
```

A diferença principal é que ao utilizar o `inspec exec supermarket://profile` o STDOUT é exibido com cores, facilitando a leitura

## Definindo Requerimentos

Antes de escrever os testes precisamos de uma estratégia para definir quais são os testes importantes a serem feitos em nossa Infraestrutura.

Alguns dados para se levar em conta ao efetuar uma análise de requerimento dos testes podem ser:

- **Estrutura** - Quais programas devem ser testados ?
- **Dados** - Existe algum arquivo que seu conteúdo deve ser persistente ?
- **Funcionalidade** - Existe alguma porta para ser testada ? Acesso a Internet ?
- **Plataforma** - É preciso verificar qual o Sistema Operacional ?
- **Segurança** - Hardening, CVS, LGPD, etc..

## Escrevendo Testes

### Requerimentos - Compliance

Para a máquina compliance iremos verificar:

- **Pacotes Instalados**
  - InSpec
  - Puppet Server
- **Serviços em Execução**
  - Puppet Server
- **Arquivos de Configuração**
  - Puppet Server
    - \* Autosign.conf

### Testes - Compliance

Remova o arquivo de exemplo

```
1 rm /root/inspec-profiles/compliance/controls/example.rb
```

Crie o arquivo **compliance.rb** e adicione os testes

```
1 vim /root/inspec-profiles/compliance/controls/compliance.rb
```

```

1 title 'Testes de Compliance'
2
3 control 'inspec' do
4   impact 'critical'
5   title 'Verificando a instalacao do Inspec'
6   desc 'testes de infraestrutura da 4labs'
7   describe package('inspec') do
8     it { should be_installed }
9     its('version') { should cmp > '4' }
10    end
11   describe file('/root/inspec-profiles/') do
12     it { should be_directory }
13   end
14 end
15
16 control 'puppetserver' do
17   impact 'critical'
18   title 'Verifica a instalacao do Puppet Server'
19   desc ' Verifica se o Puppet Server está instalado, se o servico esta sendo executado e se o
        arquivo autosign tem o conteúdo correto'
20   describe package('puppetserver') do
21     it { should be_installed }
22     its('version') { should cmp >= '6' }
23   end
24   describe file('/etc/puppetlabs/puppet/autosign.conf') do
25     it { should be_file }
26     its('content') { should include '*.4labs.example' }
27   end
28   describe service('puppetserver') do
29     it { should be_installed }
30     it { should be_enabled }
31     it { should be_running }
32   end
33 end

```

Após escrever o teste podemos utilizar o comando **inspec check** para verificar se existe algum erro no nosso arquivo.

```
1 inspec check /root/inspec-profiles/compliance
```

```

1 Location : /root/inspec-profiles/compliance
2 Profile : compliance
3 Controls : 2
4 Timestamp : 2019-11-16T13:11:52+00:00
5 Valid : true
6
7 No errors or warnings

```

## Requerimentos - Container

Para a máquina container iremos verificar: \* **Pacotes Instalados** \* docker-ce \* docker-ce-cli \* containerd.io \* **Serviços em Execução** \* docker \* **Imagem do Docker** \* webserver

## Testes - Container

Crie o profile Container e remova o arquivo de exemplo

```

1 inspec init profile /root/inspec-profiles/container
2 rm /root/inspec-profiles/container/controls/example.rb

```

Crie o arquivo **container.rb** e adicione os testes

```
1 vim /root/inspec-profiles/container/controls/container.rb
```

```
1 title 'Testes de Docker'
2
3 control 'Docker' do
4   impact 'critical'
5   title 'Verificando a instalacao do Docker'
6   desc 'testes de infraestrutura da 4labs'
7   describe package('docker-ce') do
8     it { should be_installed }
9     its('version') { should cmp > '19' }
10  end
11  describe package('docker-ce-cli') do
12    it { should be_installed }
13    its('version') { should cmp > '19' }
14  end
15  describe package('containererd.io') do
16    it { should be_installed }
17    its('version') { should cmp > '1.2' }
18  end
19  describe service('docker') do
20    it { should be_installed }
21    it { should be_enabled }
22    it { should be_running }
23  end
24 end
25 control 'docker_image' do
26   impact 'medium'
27   title 'Verifica imagens do docker'
28   desc ' Verifica se as imagens necessárias do docker estao presentes'
29   describe docker_image('webserver') do
30     it { should exist }
31     its('repo') { should eq 'webserver' }
32     its('tag') { should eq 'latest' }
33   end
34 end
```

Verifique se o arquivo possui algum erro

```
1 inspec check /root/inspec-profiles/container
```

## Requerimentos - SCM

Para a máquina iremos verificar:  
\* **Serviços em Execução** \* gogs  
\* **Arquivos de configuração** \* app.ini  
\* **Portas** \* 80 \* 2222  
\* **http** \* http://scm.4labs.example  
\* status: 200

## Testes - SCM

Crie o profile SCM e remova o arquivo de exemplo

```
1 inspec init profile /root/inspec-profiles/scm
2 rm /root/inspec-profiles/scm/controls/example.rb
```

Crie o arquivo .rb e adicione os testes

```
1 vim /root/inspec-profiles/scm/controls/scm.rb
```

```

1 title 'Testes de SCM'
2
3 control 'Gogs' do
4   impact 'critical'
5   title 'Verificando a instalacao do Gogs'
6   desc 'testes de infraestrutura da 4labs'
7   describe file('/opt/gogs') do
8     it { should be_directory}
9   end
10  describe service('gogs') do
11    it { should be_installed }
12    it { should be_enabled }
13    it { should be_running }
14  end
15  describe file('/opt/gogs/custom/conf/app.ini') do
16    it { should be_file }
17    its('content') { should include 'scm.4labs.example' }
18  end
19  describe port('80') do
20    it { should be_listening }
21    its('processes') {should include 'gogs'}
22  end
23  describe port('2222') do
24    it { should be_listening }
25    its('processes') {should include 'gogs'}
26  end
27  describe http('http://scm.4labs.example') do
28    its('status') { should cmp 200}
29  end
30 end

```

Verifique se o arquivo possui algum erro

```
1 inspec check /root/inspec-profiles/scm
```

## Requerimentos - LOG

Para a máquina LOG iremos verificar: \* **Pacotes Instalados** \* openjdk-8-jre-headless \* elasticsearch \* kibana \* logstash \* nginx \* **Serviços em Execução** \* elasticsearch \* kibana \* logstash \* nginx \* **Arquivos de Configuração** \* elasticsearch.yml \* kibana.yml \* output-elasticsearch.conf \* syslog-filter.conf \* default (nginx) \* **HTTP** \* http://log.4labs.example \* status: 302 - redirecionamento \* **Portas** \* 80 \* 5443 \* 5601 \* 9200 \* 9300 \* 9600

## Testes - LOG

Crie o profile e remova o arquivo de exemplo

```
1 inspec init profile /root/inspec-profiles/log
2 rm /root/inspec-profiles/log/controls/example.rb
```

Crie o arquivo .rb e adicione os testes

```
1 vim /root/inspec-profiles/log/controls/log.rb
```

```

1 title 'Testes de LOG'
2
3 control 'nginx' do
4   impact 'medium'
5   title 'Verificando a instalacao do NGINX e configuração'
6   desc 'testes de infraestrutura da 4labs'
7   describe package('nginx') do

```

```

8   it { should be_installed}
9   its('version') { should cmp > '1.1' }
10 end
11 describe file('/etc/nginx/sites-enabled/default') do
12   it { should be_file}
13   it { should be_owned_by 'root' }
14   its('mode') { should cmp '0644'}
15   its('link_path') { should eq '/etc/nginx/sites-available/default'}
16   its('content') { should include 'listen 80'}
17   its('content') { should include 'server_name log.4labs.example'}
18   its('content') { should include 'proxy_pass http://localhost:5601'}
19 end
20 describe service('nginx') do
21   it { should be_installed }
22   it { should be_enabled }
23   it { should be_running }
24 end
25 describe port('80') do
26   it { should be_listening }
27   its('processes') {should include 'nginx'}
28 end
29 describe http('http://log.4labs.example') do
30   its('status') { should cmp 302}
31 end
32 end
33
34 control 'ELKB' do
35   impact 'critical'
36   title 'Verificando a instalacao do Elastic Stack'
37   desc 'testes de infraestrutura da 4labs'
38   describe package('openjdk-8-jre-headless') do
39     it { should be_installed}
40     its('version') { should cmp > '8' }
41   end
42   describe package('elasticsearch') do
43     it { should be_installed}
44     its('version') { should cmp > '7' }
45   end
46   describe package('kibana') do
47     it { should be_installed}
48     its('version') { should cmp > '7' }
49   end
50   describe package('logstash') do
51     it { should be_installed}
52     its('version') { should cmp > '1:7' }
53   end
54   describe service('elasticsearch') do
55     it { should be_installed }
56     it { should be_enabled }
57     it { should be_running }
58   end
59   describe service('kibana') do
60     it { should be_installed }
61     it { should be_enabled }
62     it { should be_running }
63   end
64   describe service('logstash') do
65     it { should be_installed }
66     it { should be_enabled }
67     it { should be_running }
68   end
69 end
70
71 control 'elkb-files' do
72   impact 'critical'
73   title 'Verificando arquivos de configuracao do Elastic Stack'
74   desc 'testes de infraestrutura da 4labs'
75   describe file('/etc/elasticsearch/elasticsearch.yml') do
76     it { should be_file}
77     its('content') { should include 'http.port: 9200'}
78   end

```

```

79  describe file('/etc/kibana/kibana.yml') do
80    it { should be_file }
81    its('content') { should include 'server.port: 5601' }
82  end
83  describe file('/etc/logstash/conf.d/filebeat-input.conf') do
84    it { should be_file }
85    its('content') { should include 'port => 5443' }
86    its('content') { should include 'type => syslog' }
87  end
88  describe file('/etc/logstash/conf.d/syslog-filter.conf') do
89    it { should be_file }
90    its('content') { should include 'if [type] == "syslog"' }
91  end
92  describe file('/etc/logstash/conf.d/output-elasticsearch.conf') do
93    it { should be_file }
94    its('content') { should include 'hosts => "localhost:9200"' }
95  end
96 end
97
98 control 'elkb-ports' do
99   impact 'critical'
100  title 'Verificando portas do Elastic Stack'
101  desc 'testes de infraestrutura da 4labs'
102  describe port('5443') do
103    it { should be_listening }
104    its('processes') {should include 'java'}
105  end
106  describe port('5601') do
107    it { should be_listening }
108    its('processes') {should include 'node'}
109  end
110  describe port('9200') do
111    it { should be_listening }
112    its('processes') {should include 'java'}
113  end
114  describe port('9300') do
115    it { should be_listening }
116    its('processes') {should include 'java'}
117  end
118  describe port('9600') do
119    it { should be_listening }
120    its('processes') {should include 'java'}
121  end
122 end

```

Verifique se o arquivo possui algum erro

```
1 inspec check /root/inspec-profiles/log
```

## Requerimentos - Container

Para a máquina container iremos verificar:

- \* **Pacotes Instalados**
- \* ansible
- \* java-1.8.0-openjdk
- \* rundeck
- \* **Serviços em Execução**
- \* rundeckd
- \* **Arquivos de Configuração**
- \* ansible/hosts
- \* ansible.cfg
- \* framework.properties
- \* rundeck-config.properties
- \* **HTTP**
- \* http://automation.4labs.example:4440
- \* status: 302 - redirecionamento
- \* **Portas**
- \* 4440
- \* 3260

## Testes - Container

Crie o profile Automation e remova o arquivo de exemplo

```
1 inspec init profile /root/inspec-profiles/automation
2 rm /root/inspec-profiles/automation/controls/example.rb
```

Crie o arquivo **automation.rb** e adicione os testes

```
1 vim /root/inspec-profiles/automation/controls/automation.rb
```

```
1 title 'Testes de Automation'
2
3 control 'ansible' do
4   impact 'critical'
5   title 'Verificando a instalacao do Ansible e arquivos de configuracao'
6   desc 'testes de infraestrutura da 4labs'
7   describe package('ansible') do
8     it { should be_installed }
9     its('version') { should cmp > '2.8' }
10  end
11  describe file('/etc/ansible/ansible.cfg') do
12    it { should be_file }
13    it { should be_owned_by 'root' }
14    its('mode') { should cmp '0644' }
15    its('content') { should include 'host_key_checking = False' }
16    its('content') { should include 'private_key_file = /root/.ssh/id_rsa' }
17  end
18  describe file('/etc/ansible/hosts') do
19    it { should be_file }
20    it { should be_owned_by 'root' }
21    its('mode') { should cmp '0644' }
22    its('content') { should include 'automation.4labs.example' }
23    its('content') { should include 'compliance.4labs.example' }
24    its('content') { should include 'container.4labs.example' }
25    its('content') { should include 'log.4labs.example' }
26    its('content') { should include 'scm.4labs.example' }
27  end
28 end
29
30 control 'rundeck' do
31   impact 'critical'
32   title 'Verificando a instalacao do rundeck e arquivos de configuracao'
33   desc 'testes de infraestrutura da 4labs'
34   describe package('rundeck') do
35     it { should be_installed }
36     its('version') { should cmp > '3.1' }
37   end
38   describe package('java-1.8.0-openjdk') do
39     it { should be_installed }
40     its('version') { should cmp > '1.8' }
41   end
42   describe file('/etc/rundeck/framework.properties') do
43     it { should be_file }
44     it { should be_owned_by 'rundeck' }
45     its('mode') { should cmp '0640' }
46     its('content') { should include 'framework.server.name = 4labs' }
47     its('content') { should include 'framework.server.hostname = automation.4labs.example' }
48     its('content') { should include 'framework.server.port = 4440' }
49     its('content') { should include 'framework.server.url = http://automation.4labs.example:4440' }
50   end
51   describe file('/etc/rundeck/rundeck-config.properties') do
52     it { should be_file }
53     it { should be_owned_by 'rundeck' }
54     its('mode') { should cmp '0640' }
55     its('content') { should include 'grails.serverURL=http://automation.4labs.example:4440' }
56   end
57   describe service('rundeckd') do
58     it { should be_installed }
59     it { should be_enabled }
60     it { should be_running }
61   end
62   describe port('4440') do
63     it { should be_listening }
64     its('processes') { should include 'java' }
65   end
66   describe http('http://automation.4labs.example:4440') do
```

```
67     its('status') { should cmp 302}
68   end
69 end
```

Verifique se o arquivo possui algum erro

```
1 inspec check /root/inspec-profiles/automation
```

## Includes

Podemos utilizar **Includes** para chamar profiles dentro de outros profiles, com isso podemos criar um profile base para todos os sistemas e inclui-lo aos testes.

### my-app-profile

```
control 'myapp-1'
control 'myapp-2'
control 'myapp-3'

include_controls 'my-baseline'
```

### my-baseline

```
control 'baseline-1'
control 'baseline-2'
```

Figura 10.2: Include

É possível incluir um profile e pular etapas através do parâmetro **skip\_control**

Podemos também dizer quais etapas serão utilizadas através do parâmetro **require\_controls**

Podemos então criar um arquivo de base com os testes comuns a todos os servidores e adicionar este include a cada profile

## Requerimentos - Base

Para a máquina container iremos verificar: \* **Pacotes Instalados** \* puppet-agent \* filebeat \* **Serviços em Execução** \* puppet-agent \* filebeat \* **Arquivos de Configuração** \* puppet.conf \* filebeat.yml \* **Interface** \* IPv4

## Testes - Base

Crie o profile base e remova o controle de exemplo

```
1 inspec init profile /root/inspec-profiles/base
2 rm /root/inspec-profiles/base/controls/example.rb
```

```
my-app-profile
control 'myapp-1'
control 'myapp-2'
control 'myapp-3'

include_controls 'my-baseline' do
  skip_control 'baseline-2'
end
```

### **my-baseline**

```
control 'baseline-1'
control 'baseline-2'
```

```
my-app-profile
control 'myapp-1'
control 'myapp-2'
control 'myapp-3'

require_controls 'my-baseline' do
  control 'baseline-2'
  control 'baseline-4'
end
```

### **my-baseline**

```
control 'baseline-1'
control 'baseline-2'
control 'baseline-3'
control 'baseline-4'
control 'baseline-5'
```

Figura 10.3: Include

Crie o arquivo **base.rb** e adicione os testes

```
1 vim /root/inspec-profiles/base/controls/base.rb
```

```
1 title 'Testes de Base'
2
3 control 'puppetagent' do
4   impact 'critical'
5   title 'Verificando o puppetagent'
6   desc 'testes de infraestrutura da 4labs'
7   describe package('puppet-agent') do
8     it { should be_installed }
9     its('version') { should cmp > '6' }
10    end
11   describe file('/etc/puppetlabs/puppet/puppet.conf') do
12     it { should be_file}
13     it { should be_owned_by 'root' }
14     its('mode') { should cmp '0644'}
15     its('content') { should include 'server = compliance.4labs.example' }
16   end
17   describe service('puppet') do
18     it { should be_installed }
19     it { should be_enabled }
20     it { should be_running }
21   end
22 end
23
24 control 'filebeat' do
25   impact 'critical'
26   title 'Verificando o Filebeat'
27   desc 'testes de infraestrutura da 4labs'
28   describe package('filebeat') do
29     it { should be_installed }
30     its('version') { should cmp > '7' }
31   end
32   describe file('/etc/filebeat/filebeat.yml') do
33     it { should be_file}
34     it { should be_owned_by 'root' }
35     its('mode') { should cmp '0600'}
36     its('content') { should include 'hosts: ["log.4labs.example:5443"]' }
37   end
38   describe service('filebeat') do
39     it { should be_installed }
40     it { should be_enabled }
41     it { should be_running }
42   end
43 end
44
45 control 'interface' do
46   impact 'critical'
47   title 'Verificando a interface de rede'
48   desc 'testes de infraestrutura da 4labs'
49   IP = [
50     '10.5.25.10',
51     '10.5.25.20',
52     '10.5.25.30',
53     '10.5.25.40',
54     '10.5.25.50',
55   ]
56   describe interface('lo') do
57     its('ipv4_addresses') { should include '127.0.0.1' }
58   end
59   if os.name == 'ubuntu'
60     describe interface('enp0s8') do
61       it { should be_up }
62       its('speed') { should eq 1000 }
63       its('ipv4_addresses') { should be_in IP }
64     end
```

```

65   elsif os.family == 'redhat' or os.name == 'debian'
66     describe interface('eth1') do
67       it { should be_up }
68       its('speed') { should eq 1000 }
69       its('ipv4_addresses') { should be_in IP }
70     end
71   end
72 end

```

Verifique se o arquivo possui algum erro

```
1 inspec check /root/inspec-profiles/base
```

## Adicionando o Include

Vamos agora adicionar o include aos profiles **automation**, **compliance**, **container**, **scm** e **log**

```

1 cd /root/inspec-profiles
2
3 for profile in automation compliance container scm log
4 do
5   echo "include_controls 'base'" > $profile/controls/base_inc.rb
6 done

```

Para a máquina log não executaremos o controle filebeat, uma vez que ela não utiliza o mesmo para captura de logs, vamos editar o arquivo para remover este include

```
1 vim log/controls/base_inc.rb
```

```

1 include_controls 'base' do
2   skip_control 'filebeat'
3 end

```

Quando adicionamos dependencias, é necessário que elas sejam informadas no arquivo **inspec.yml**

```

1 for profile in automation compliance container scm log
2 do
3 echo -e "
4 depends:
5   - name: base
6     path: /root/inspec-profiles/base" >> $profile/inspec.yml
7 done

```

Quando um profile é executado, é feita a leitura do arquivo **inspec.yml** para que seja feito o cache das dependências localmente e gerar um arquivo **inspec.lock**.

Se adicionarmos ou atualizarmos alguma dependência no **inspec.yml** é necessário recriar o arquivo **inspec.lock**

Podemos fazer isto através do comando **inspec vendor <profile> --overwrite**

```

1 for profile in automation compliance container scm log
2 do
3   inspec vendor /root/inspec-profiles/$profile --overwrite
4 done

```

## Testando os Profiles

Execute o teste nos profiles e visualize o estado

```
1 inspec exec /root/inspec-profiles/automation -t ssh://automation.4labs.example -i /root/.ssh/id_rsa
2 inspec exec /root/inspec-profiles/compliance -t ssh://compliance.4labs.example -i /root/.ssh/id_rsa
3 inspec exec /root/inspec-profiles/container -t ssh://container.4labs.example -i /root/.ssh/id_rsa
4 inspec exec /root/inspec-profiles/scm -t ssh://scm.4labs.example -i /root/.ssh/id_rsa
```

Ao executar o profile log é possível verificar que o control filebeat não foi executado conforme programado

```
1 inspec exec /root/inspec-profiles/log -t ssh://log.4labs.example -i /root/.ssh/id_rsa
```

## Integrações

### Versionando o Código com o Gogs

Primeiramente iremos criar nosso repositório no gogs, acesse o endereço <http://scm.4labs.example> e efetue o login com o usuário **analista** e senha **devops@4linux**

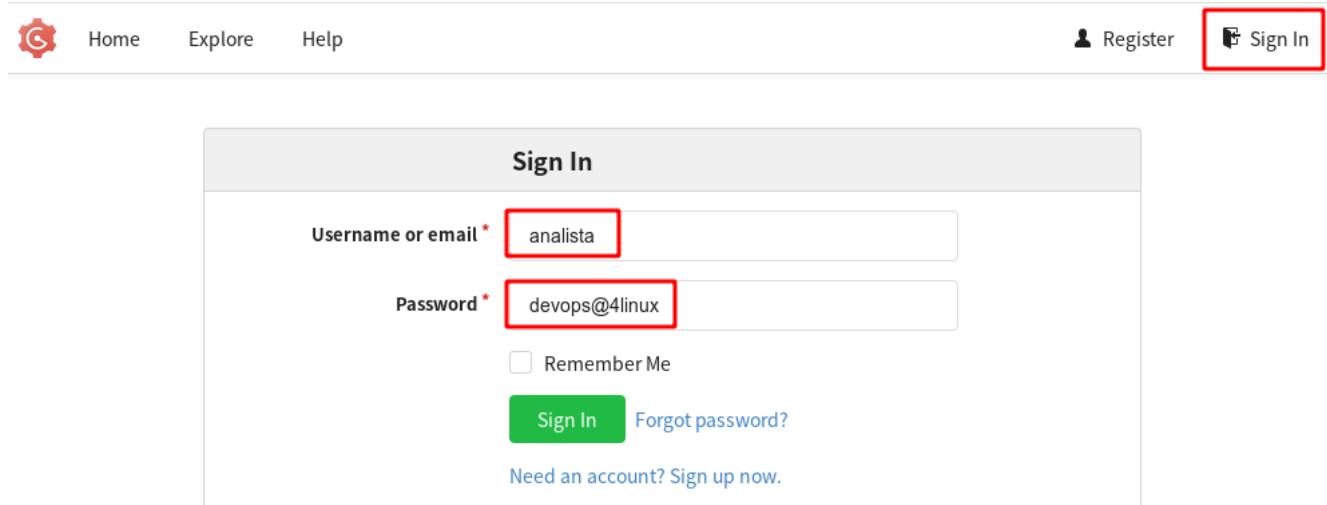


Figura 10.5: Gogs Login

No canto superior direito, clique em + e em seguida em + New Repository

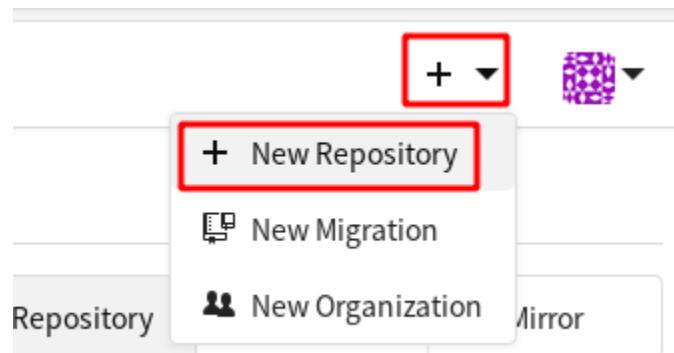


Figura 10.6: New Repository

Preencha o nome do repositório como **inspec**, marque o repositório como **privado**, digite uma descrição e clique em **Create Repository**

New Repository

Owner \* analista

Repository Name \* primeiro

A good repository name is usually composed of short, memorable and unique keywords.

Visibility  This repository is Private

Description Meu primeiro repositório com Gogs

Description of repository. Maximum 512 characters length.  
Available characters: 478

.gitignore Select .gitignore templates

License Select a license file

Readme Default

Initialize this repository with selected files and template

Create Repository Cancel

Figura 10.7: New Repository

Acesse o diretório **/root/inspec-profiles** na máquina compliance

```
1 cd /root/inspec-profiles
```

Configure as variaveis de ambiente e efetue o commit

```
1 git config --global user.name 'Analista DevOps'
2 git config --global user.email 'analista@4labs.example'
3 git init
4 git add .
5 git commit -m "Commit inicial"
```

Copie o endereço SSH do repositório no gogs

Adicione o repositório remoto

The screenshot shows a GitHub repository page for 'analista / primeiro'. At the top, there's a navigation bar with links for Dashboard, Issues, Pull Requests, and Explore. To the right of the navigation are buttons for creating a new repository or issue, and a dropdown menu. Below the navigation, the repository name 'analista / primeiro' is displayed, along with 'Unwatch' (1), 'Star' (0), and a settings icon. A tab bar below the repository name includes 'Files' (selected), 'Issues' (0), 'Wiki', and 'Settings'. A 'Quick Guide' section provides instructions for cloning the repository, with options for 'HTTP' (selected) and 'SSH'. It also shows the URL 'http://scm.4labs.example/analista/primeiro.git'. Below this, a 'Create a new repository on the command line' section contains a code block with the following commands:

```
touch README.md  
git init  
git add README.md  
git commit -m "first commit"  
git remote add origin http://scm.4labs.example/analista/primeiro.git  
git push -u origin master
```

Further down, a 'Push an existing repository from the command line' section contains a code block with the following commands:

```
git remote add origin http://scm.4labs.example/analista/primeiro.git  
git push -u origin master
```

Figura 10.8: Repository

```
1 git remote add origin ssh://gogs@scm.4labs.example:2222/analista/inspec.git
2 git push origin master
```

Atualize a página do repositório no gogs e verifique o código enviado

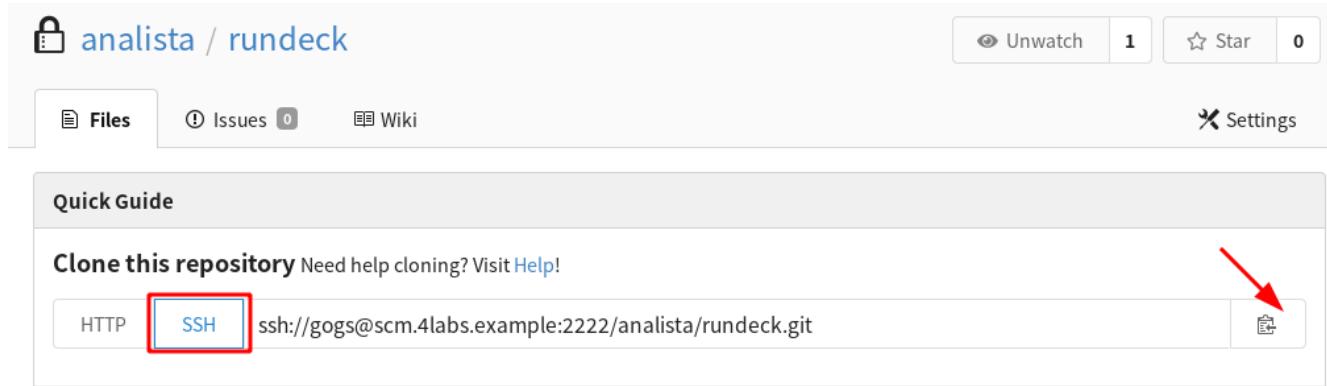


Figura 10.9: Repository

### Automatizando os Testes com Rundeck

Acesse o rundeck através do endereço <http://automation.4labs.example:4440> e efetue o login

Acesse o Rundeck pelo endereço <http://automation.4labs.example:4440> e efetue o login como **analista** e senha **devops@4linux**

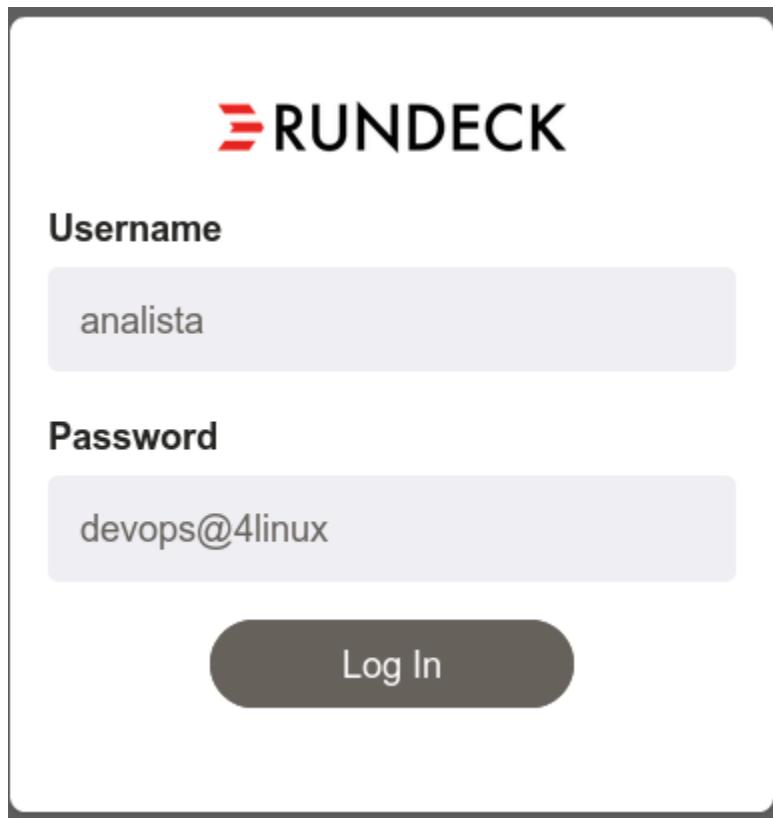


Figura 10.10: Rundeck Login

Selecione o projeto **525**

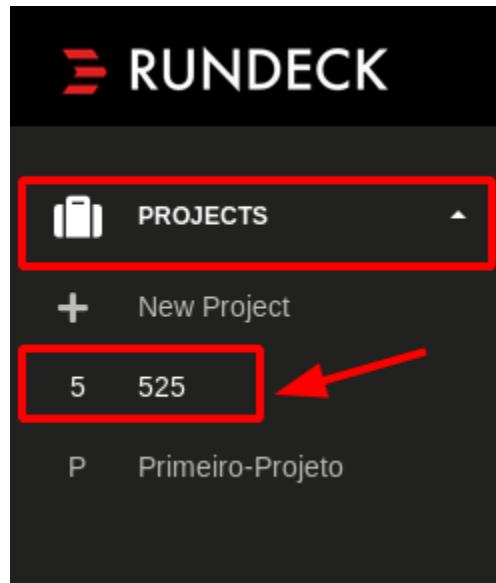


Figura 10.11: Project

Crie um novo job

Adicione o nome **Testes de Infraestrutura** e adicione um comentário

Em workflow marque a opção **Run remaining steps before failing**.

Ainda em workflow adicione os passos:

```
1 inspec exec /root/inspec-profiles/automation -t ssh://automation.4labs.example -i /root/.ssh/id_rsa
2 inspec exec /root/inspec-profiles/compliance -t ssh://compliance.4labs.example -i /root/.ssh/id_rsa
3 inspec exec /root/inspec-profiles/container -t ssh://container.4labs.example -i /root/.ssh/id_rsa
4 inspec exec /root/inspec-profiles/scm -t ssh://scm.4labs.example -i /root/.ssh/id_rsa
5 inspec exec /root/inspec-profiles/log -t ssh://log.4labs.example -i /root/.ssh/id_rsa
```

Em nodes selecionaremos apenas o nó compliance para execução dos comandos

Clique em Create

Execute o novo job clicando em **Run Job Now**

Acompanhe a execução

Agora que o Job foi configurado e testado podemos atualizar nosso repositório de Jobs indo no menu **JOBs** e clicando em **Job Actions** e em seguida em **Commit Changes to Git**

Marque o checkbox, adicione uma mensagem de commit e clique em **commit**

Verifique o commit no repositório do gogs

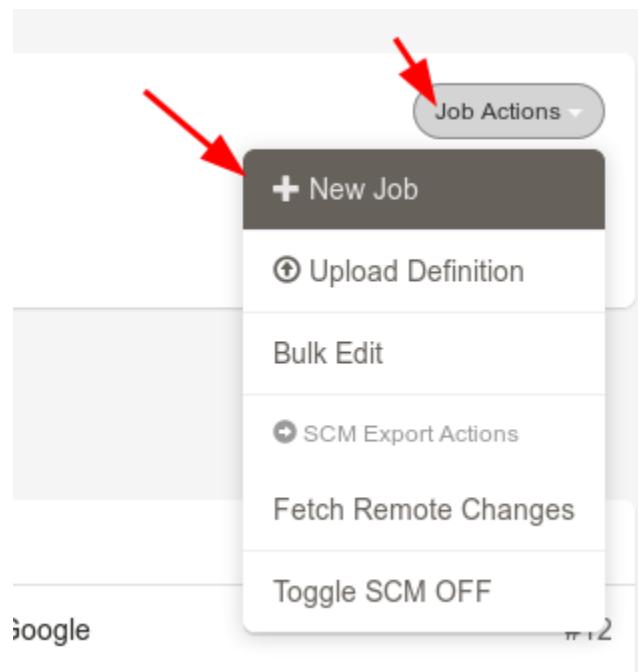


Figura 10.12: Job

## Create New Job: Testes de Infraestrutura

Details   Workflow   Nodes   Schedule   Notifications   Other

---

Job Name	Testes de Infraestrutura
Description	<p>Edit</p> <p>1 Execução dos testes de infraestrutura com InSpec</p>

---

The first line of the description will be shown in plain text, the

Figura 10.13: Job

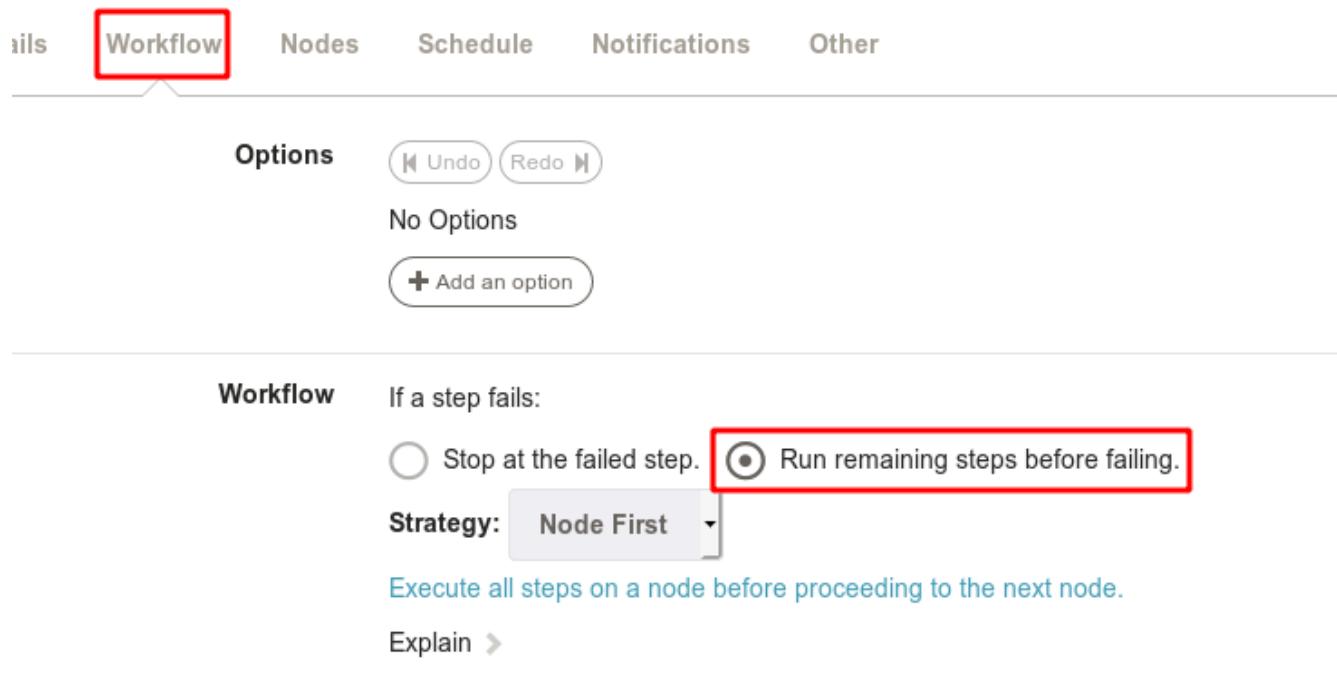


Figura 10.14: Job

1. ➔ inspec exec /root/inspec-profiles/automation -t ssh://automation.4labs.example -i /root/.ssh/id\_rsa  
testes de automação
2. ➔ inspec exec /root/inspec-profiles/compliance -t ssh://compliance.4labs.example -i /root/.ssh/id\_rsa  
testes de compliance
3. ➔ inspec exec /root/inspec-profiles/container -t ssh://container.4labs.example -i /root/.ssh/id\_rsa  
testes de container
4. ➔ inspec exec /root/inspec-profiles/scm -t ssh://scm.4labs.example -i /root/.ssh/id\_rsa  
testes de scm
5. ➔ inspec exec /root/inspec-profiles/log -t ssh://log.4labs.example -i /root/.ssh/id\_rsa  
testes de log

Figura 10.15: Job

Create New Job: Testes de Infraestrutura

Details Workflow **Nodes** Schedule Notifications Other

**Nodes**  Dispatch to Nodes  Execute locally  
Choose whether the Job will run on filtered nodes or only on the local node.

**Node Filter**

**Exclude Filter**

**Show Excluded Nodes**  Yes  No  
If true, the excluded nodes will be indicated when running the Job. Otherwise they will not be shown at all.

**Matched Nodes** 1 Node Matched

Figura 10.16: Job

**Rank Order**  Ascending  Descending

**If a node fails**  Fail the step without running on any remaining node  
 Continue running on any remaining nodes before fail

**If node set empty**  Fail the job.  
 Continue execution.

**Node selection**  Target nodes are selected by default  
 The user has to explicitly select target nodes

**Orchestrator**

This can be used to control the order and timing in which

Figura 10.17: Job

525

## Testes de Infraestrutura Created

Execução dos testes de infraestrutura com InSpec

a04bac9d-32e5-4a26-8a13-fdd6e671778d

Action ▾

Definition

Follow execution

Nodes

Run Job Now ▶

Nodes

Change the Target Nodes (1)

Figura 10.18: Job

## Testes de Infraestrutura Created

Execução dos testes de infraestrutura com InSpec

Succeeded 0.00:45 at 3:11 pm ▶  
you

Log Output »

100% 1/1 COMPLETE

0 FAILED

0 INCOMPLETE

0 NOT STARTED

Start time

Duration

Node	compliance	All Steps OK	3:10:39 pm	0.00:10
> testes de automação	OK		3:10:49 pm	0.00:01
> testes de compliance	OK		3:10:59 pm	0.00:07
> testes de container	OK		3:11:07 pm	0.00:06
> testes de scm	OK		3:11:14 pm	0.00:09
> testes de log	OK			

Figura 10.19: Job

## All Jobs

Expand All Collapse All

▶ Created Testes de Infraestrutura Execução dos testes de infraestrutura com InSpec

▶ Verificando hosts Job para verificação do /etc/hosts 0 in 8h48m

▶ Verificar DNS Google Job para verificar se o dns da google esta acessivel pelos servidores.

Modified Job Actions

+ New Job

Upload Definition

Bulk Edit

SCM Export Actions

Commit Changes to Git

## Activity for Jobs

Figura 10.20: Job

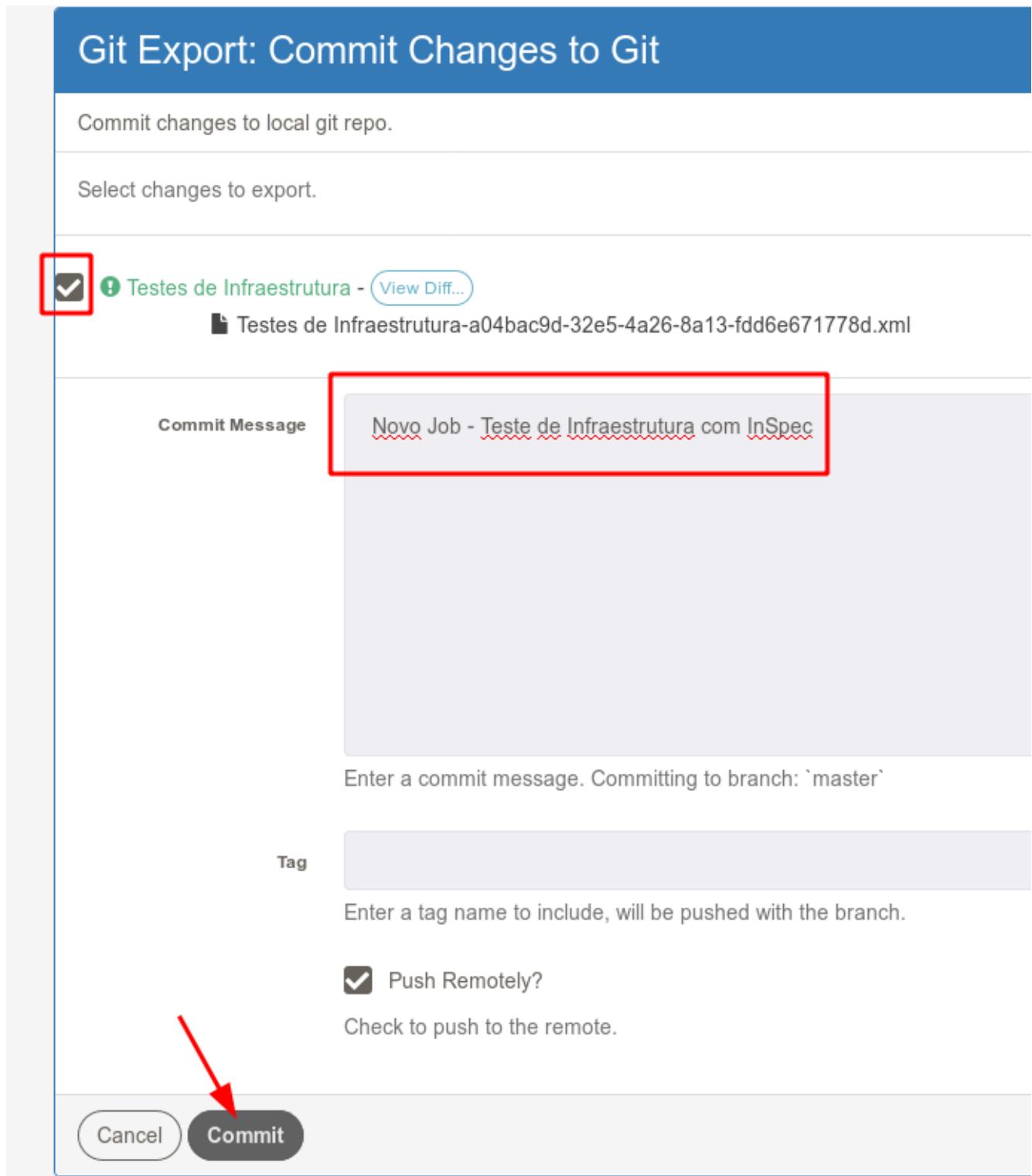


Figura 10.21: Job

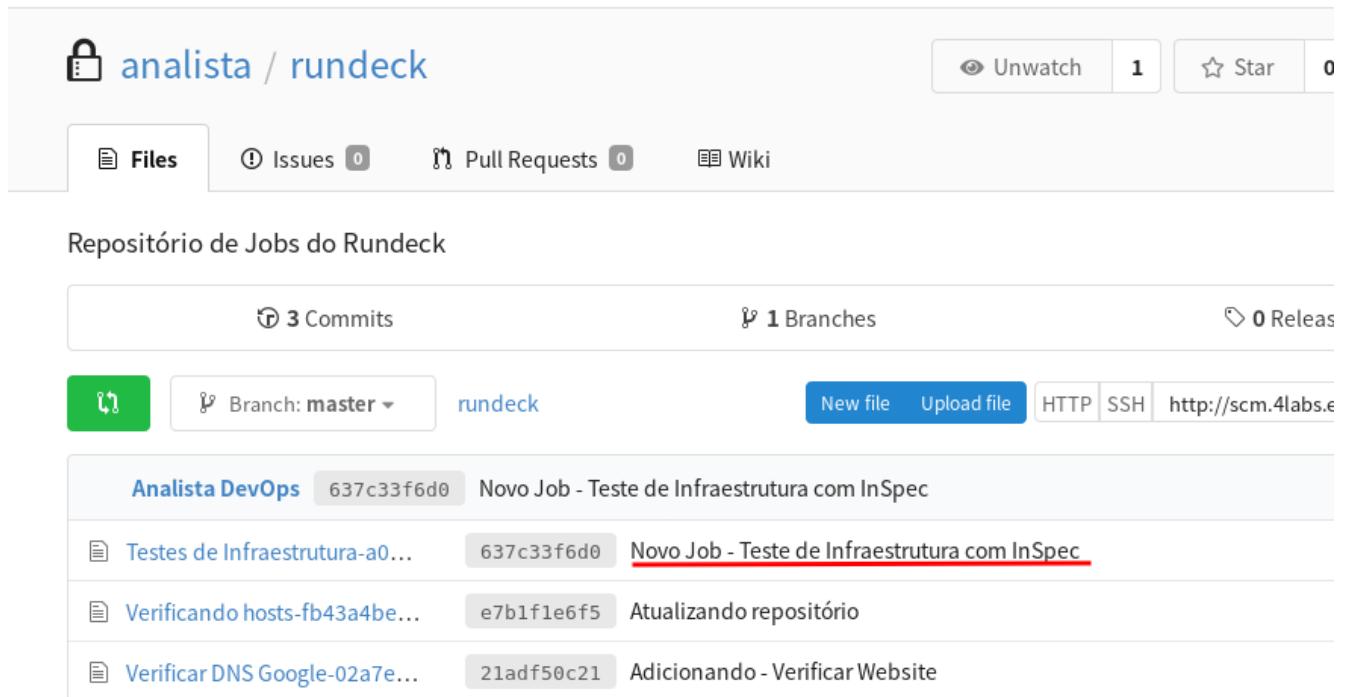


Figura 10.22: Repository

## Capítulo 11

# Gerenciando Logs com Elastic Stack

### Características do Elastic Stack

#### Componentes e Características

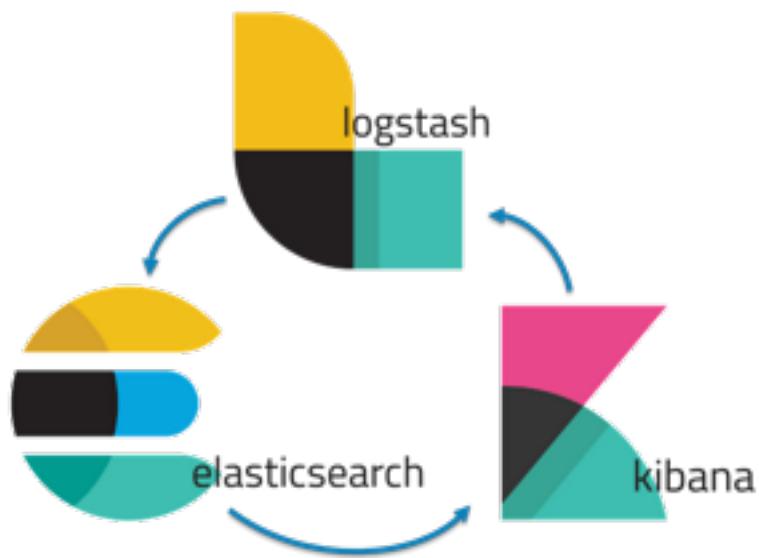


Figura 11.1: Elastic Stack

**Elastic Stack**, anteriormente conhecido como **elk stack** é um conjunto de três projetos Open Source: **Elasticsearch**, **Logstash** e **Kibana**.

Em 2015 foi introduzido o **Beats** e a solução passou a ser conhecida como **ELKB**.

Hoje em dia quando falamos sobre **ELKB**, **ELK Stack** ou **Elastic Stack** estamos nos referindo a mesma solução.

*É possível encontrar referencias ao **ELKB** em outros anacrônimos como o **ELKB**, **ELBK**, **LKEB**, etc..*

### Elasticsearch

Elasticsearch é um motor de pesquisa por texto completo e analítico muito utilizado por habitar a funcionalidade de pesquisa em diferentes aplicações e é o coração do Elastic Stack.

Podemos utilizar o elasticsearch por exemplo em um blog, no qual desejamos que os usuários possam pesquisar por vários tipos de dados que poderiam ser posts, produtos, categorias, etc...



Figura 11.2: Elasticsearch

Através do Elasticsearch podemos implementar uma pesquisa por texto completo levando em conta outros fatores como tamanho de posts ou avaliações.

Tecnicamente o Elasticsearch pode fazer tudo que gostaríamos de ter em uma poderosa ferramenta de pesquisa e não ficar limitado a apenas pesquisas de texto completo. É possível escrever consultas para dados estruturados e utilizá-los para criar gráficos utilizando o Elasticsearch como plataforma de análise

## Kibana

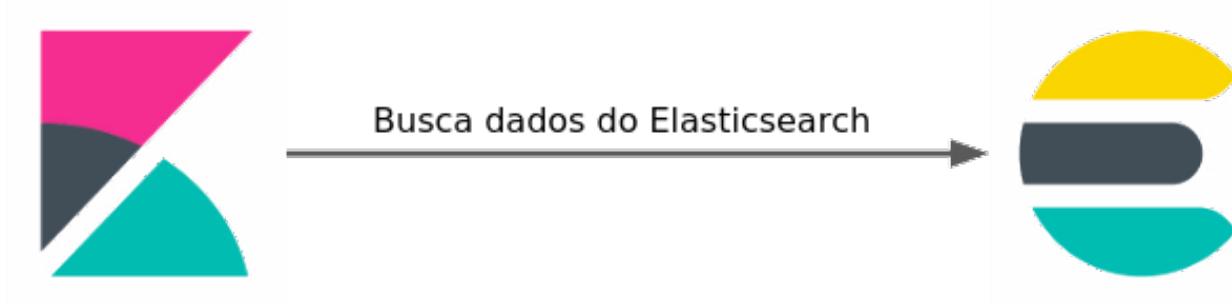


Figura 11.3: Kibana - Elasticsearch

Kibana é uma plataforma de analíticos e visualização que possibilita visualizar os dados do Elasticsearch e analisa-los. Podemos dizer que o Kibana é um Dashboard do Elasticsearch onde podemos criar visualizações e também gráficos.

O Kibana fornece uma interface para construir pesquisas e configurar a exibição de resultados através dos dados do Elasticsearch e envia solicitações para o Elasticsearch através de uma REST API.

*Existe uma website da própria Elastic com um DEMO online do dashboard do kibana com exemplos de dashboards.*

## Logstash

O Logstash é utilizado para processar os logs de aplicações e enviar-los para o Elasticsearch.

Os dados coletados pelo logstash são gerenciados como eventos que podem ser de qualquer tipo, como por exemplo, entradas de arquivos, pedidos de e-commerce, clientes, mensagens de chat etc...

O logstash desempenha seu papel através de 3 estágios:

- **Entrada** - Como o logstash recebe os dados, pode ser um endpoint HTTP, uma base de dados relacional ou até mesmo um arquivo.
- **Filtro** - Como o logstash processa os eventos recebidos na etapa de Entrada, aqui podemos analisar arquivos no formato CSV, XML ou até mesmo JSON. Também é possível fazer o enriquecimento dos dados.
- **Saida** - Para onde são enviados os dados processados, podem ser uma instância do Elasticsearch, uma base de dados, um arquivo etc...

## Beats

Beats é uma coleção de exportadores de dados, que são agentes com algum propósito em particular.

Existem diversos tipos de exportadores de dados e todos são chamados de beats: \* Packetbeat \* Metricbeat \* Winlogbeat \* Auditbeat \* Filebeat \* Heartbeat

É possível instalar os beats por servidor ou requerimento. Os beats enviam os dados para o Elasticsearch e/ou Logstash

Em nosso laboratório iremos utilizar o Filebeat que é basicamente um beat para coletar arquivos de logs e enviar suas entradas para o Elasticsearch e/ou Logstash

O Filebeat é muito útil para coleta de logs como logs de acesso ou erro e possui diversos módulos para leitura de logs tais como nginx, Apache web server, Mysql etc...

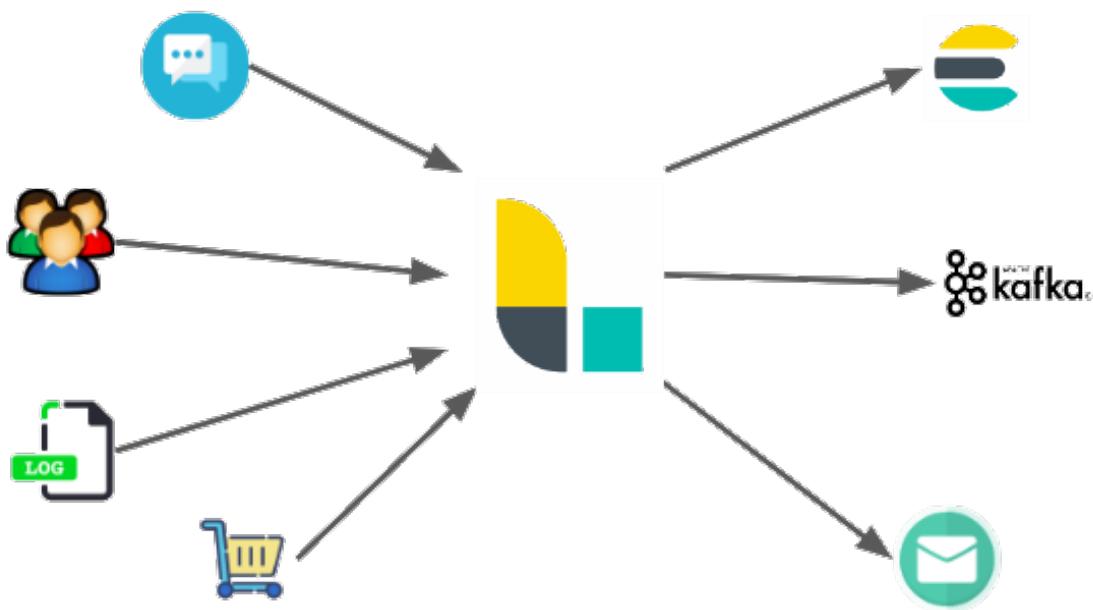


Figura 11.4: Logstash

Os beats não estão limitados aos citados como exemplo, existem inumeros tipos de beats diferentes desenvolvidos pela comunidade, cada um com sua particularidade porém todos com o mesmo propósito.

## Elastic Stack

O Elastic Stack é a solução por completa com a integração de todas as ferramentas.

## Configuração do Elastic Stack

Neste módulo iremos navegar nos arquivos de configuração do nosso Elastic Stack, uma vez que o mesmo já foi pré-configurado através do puppet

### Configuração do Elasticsearch

Para o Elasticsearch fizemos algumas alterações no arquivo **elasticsearch.yml** localizado no diretório **/etc/elasticsearch**

```

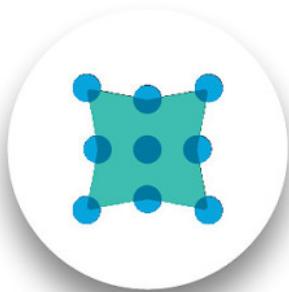
1 path.data: /var/lib/elasticsearch
2 path.logs: /var/log/elasticsearch
3 network.host: localhost
4 http.port: 9200

```

O arquivo **elasticsearch.yml** é responsável por todas as configurações da aplicação Elasticsearch

É possível definir através dos campos: \* **path.data** - Caminho para armazenamento dos dados do elasticsearch; \* **path.logs** - Caminho para armazenamento dos logs do elasticsearch; \* **network.host** - Endereço de rede no qual o elasticsearch estará escutando. Em nossa aplicação utilizamos o localhost, caso fossemos utilizar um cluster de elasticsearch deveríamos alterar o campo para o endereço IP; \* **http.port** - Porta padrão na qual o elasticsearch estará escutando e aguardando conexões (por padrão a porta 9200 é utilizada).

## The Beats family



**Packetbeat**

Network data



**Metricbeat**

Metrics



**Winlogbeat**

Windows Event Logs



**Auditbeat**

Audit data



**Filebeat**

Log files



**Heartbeat**

Uptime monitoring

Figura 11.5: Beats

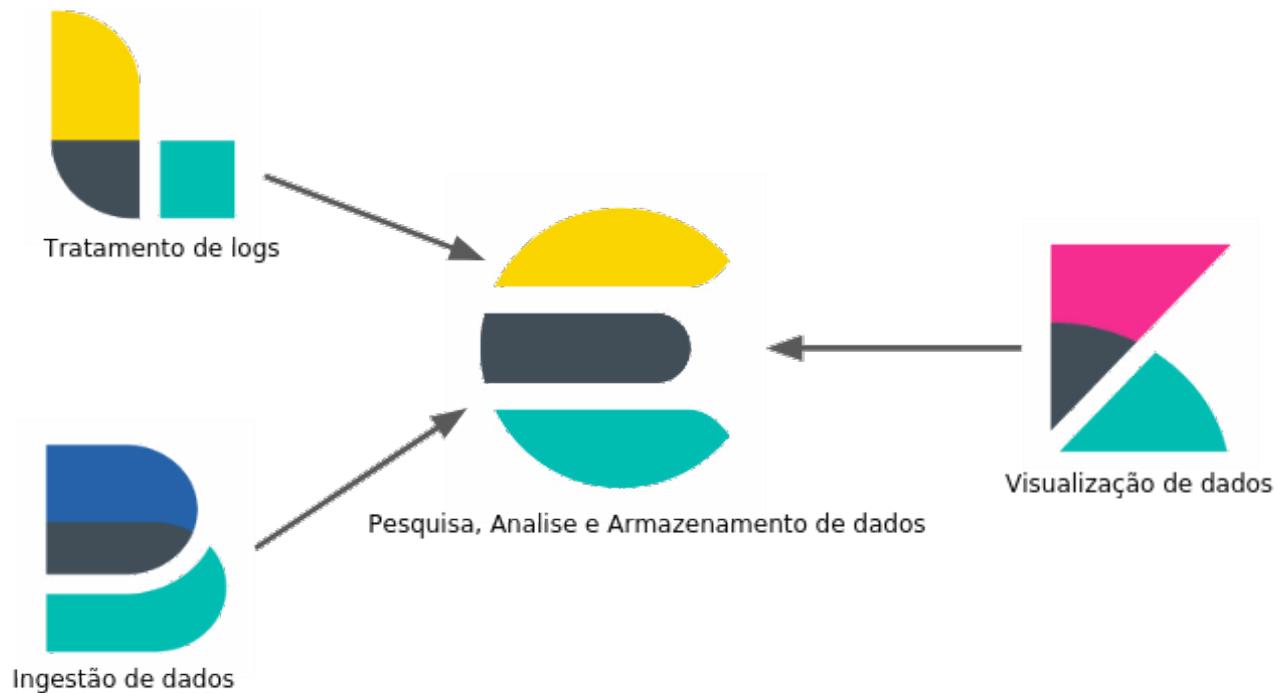


Figura 11.6: Elastic Stack

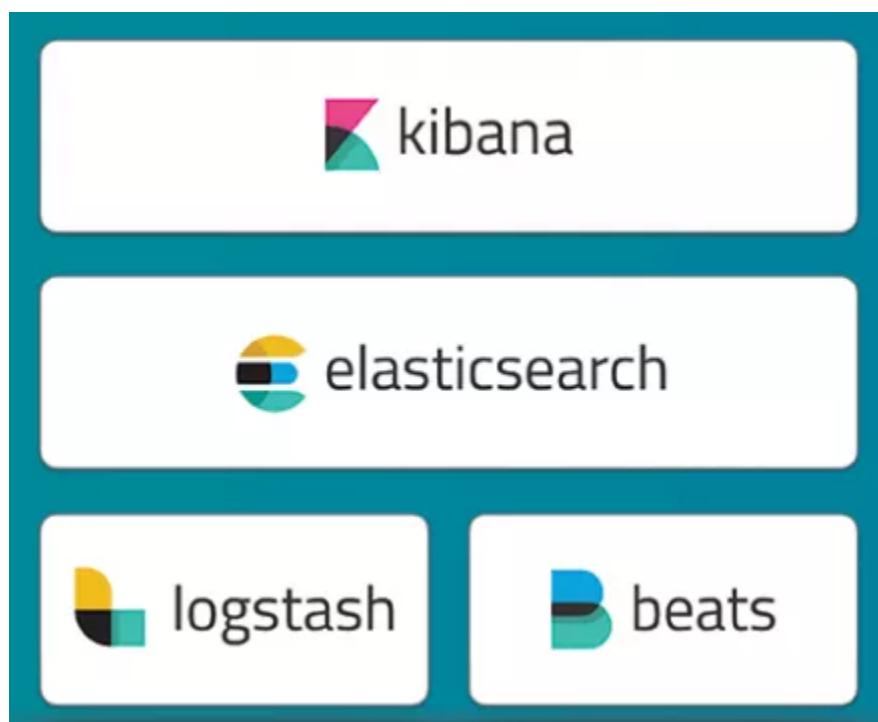


Figura 11.7: Elastic Stack

## Configuração do Kibana

Para o Kibana, fizemos alterações no arquivo **kibana.yml** localizado em **/etc/kibana**

```
1 server.port: 5601
2 server.host: "localhost"
3 elasticsearch.hosts: ["http://localhost:9200"]
```

O arquivo **kibana.yml** é responsável por todas as configurações da aplicação Kibana.

É possível definir através dos campos:

- **server.port** - Porta padrão na qual o kibana estará escutando e publicando sua interface web;
- **server.host** - Endereço de rede no qual o kibana estará escutando. Como em nossa aplicação utilizamos o **nginx** para efetuar o redirecionamento através de proxy reverso, utilizamos a configuração como localhost, caso fossemos publicar sem o nginx, poderíamos alterar para que ele ouça todas as conexões alterando o campo para **0.0.0.0**;
- **elasticsearch.hosts** - Endereços dos servidores elasticsearch no qual o kibana se conectará para geração das dashboards e índices.

## Configuração do Logstash

Para o logstash, não efetuamos alterações no arquivo **logstash.yml** uma vez que o mesmo aponta para o diretório **/etc/logstash/conf.d** para leitura dos arquivos de configuração.

Neste diretório criamos 3 arquivos para as etapas da pipeline de logs:

- filebeat-input.conf
- output-elasticsearch.conf
- syslog-filter.conf

### Input Stage

O arquivo **filebeat-input.conf** é responsável por informar qual a porta no qual o logstash estará escutando dados enviados pelo filebeat no Input Stage.

```
1 input {
2   beats {
3     port => 5443
4     type => syslog
5   }
6 }
```

As tags:

- **input** - Informa qual o estágio a ser trabalhado por meio da configuração;
- **beats** - Define que a entrada será de um módulo de beats;
- **port** - Define a porta de escuta para o serviço;
- **type** - Define o tipo do evento que será recebido.

*Outras opções podem ser vistas na documentação oficial*

### Filter Stage

O arquivo **syslog-filter.conf** é responsável por informar quais filtros serão aplicados no arquivo recebido pelo filebeat no Filter Stage

```
1 filter {
2   if [type] == "syslog" {
3     grok {
```

```

4     match => { "message" => "%{SYSLOGTIMESTAMP:syslog_timestamp} %{SYSLOGHOST:syslog_hostname} %{DATA:syslog_program}(?:\[%{POSINT:syslog_pid}\])?: %{GREEDYDATA:syslog_message}" }
5     add_field => [ "received_at", "%{@timestamp}" ]
6     add_field => [ "received_from", "%{host}" ]
7   }
8   date {
9     match => [ "syslog_timestamp", "MMM d HH:mm:ss", "MMM dd HH:mm:ss" ]
10  }
11 }
12 }
```

- **filter** - define que o arquivo trata um estágio de filtragem de dados
- **if [type]==“syslog”** - verifica se o tipo da entrada recebida pelo input stage é do tipo syslog e aplica as configurações
- **grok** - polugin para analistar e estruturar texto. Atualmente o grok é a melhor maneira de analisar e organizar dados de log não estruturados em algo estruturado e consultável. Possuí mais de 120 padrões incorporados ao logstash. Mais informações podem ser encontradas na documentação oficial
- **match** - é o parâmetro do plugin grok no qual são combinados padrões de texto em algo que possa ser comparado aos logs. Utilizamos o match no padrão **%{SYNTAX:SEMANTIC}** onde:
  - **SYNTAX** - nome do parâmetro no qual corresponderá com o texto;
  - \* Exemplos:
    - **3,44** é correspondido pelo padrão **NUMBER**;
    - **123.234.12.1** é correspondido pelo padrão **IP**;
  - **SEMANTIC** - é o identificador dado a fração do texto que foi correspondido
  - Exemplos:
    - \* **3,44** poderia ser a duração de um evento em segundos, poderíamos chamar de **duration**
    - \* **123.234.12.1** pode ser o identificador de um **client** efetuando um request.
  - Exemplo de filtro: **%{NUMBER:durantion} %{IP:client}**

*Outras opções podem ser encontradas na documentação oficial*

## Output Stage

O arquivo **output-elasticsearch.conf** é o responsável por informar no Output Stage o endpoint do elasticsearch o qual receberá os dados capturados e filtrados (no Filter Stage).

```

1 output {
2   elasticsearch { hosts => ["localhost:9200"]
3     hosts => "localhost:9200"
4     manage_template => false
5     index => "%{[@metadata][beat]}-%{+YYYY.MM.dd}"
6   }
7 }
```

- **output** - define que o arquivo trata um estágio de saída de dados;
- **elasticsearch** - definir quais hosts/portas do elasticsearch receberão os dados, caso não deseje que os dados sejam enviados para o elasticsearch, podemos utilizar no lugar de **elasticsearch** o parâmetro **file** o qual gravará os dados de evento no disco, **graphite** que enviará os dados para o graphite (uma ferramenta popular para armazenamento e geração de gráficos de métricas) ou **statsd** (um serviço que escuta por estatísticas como contadores e timers via UDP);
- **hosts** - configura o host da instância remota do elasticsearch;
- **manage\_template** - parâmetro para aplicar um template ao elasticstash durante o inicio do logstash;
- **index** - Formato no qual serão indexados os dados enviados ao elasticsearch

*Outras opções podem ser vistas na documentação oficial*

## Beats

Beats são ferramentas leves desenvolvidas para exportar todos os tipos de dados para o Elasticsearch e/ou Logstash.

Através de Beats é possível exportar dados de servidores, containers e até mesmo funções.



Figura 11.8: Beats

O principal ganho na utilização de beats conectados diretamente ao Logstash ou Elasticsearch é a utilização de um recurso chamado **Back-Pressure Sensitive Protocol**, onde o Filebeat e o Logstash/Elasticsearch conversam entre si para evitar o travamento e a perda de informações ao receber os logs.

### Back-Pressure Sensitive Protocol

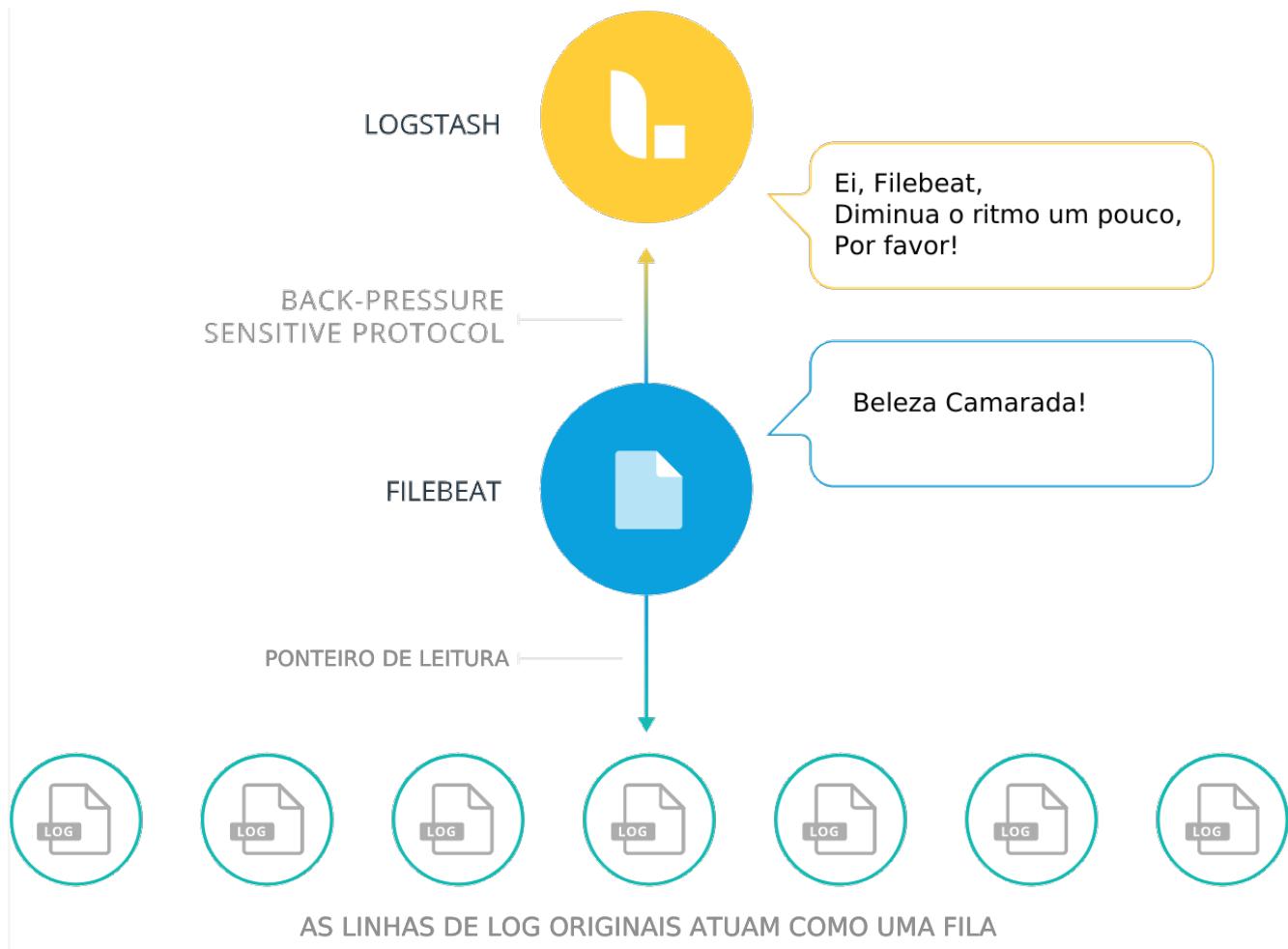


Figura 11.9: Back-Pressure Sensitive Protocol

O **Filebeat** possui um ponteiro de leitura que opera sob uma velocidade constante, quando o **Logstash** estiver ocupado tratando os arquivos ele informa ao Filebeat para que o mesmo diminua sua taxa de leitura para que os dados não sejam perdidos, uma vez descongestionado o Filebeat continua em seu ritmo original de leitura/envio de logs.

### Beats Elastic Co.

Beats Desenvolvidos pela Elastic Co.

- **Filebeat** - Beat para leitura de arquivos de log, com módulos internos para programas como auditd, Apache, Nginx, System, MySQL, etc...
- **Metricbeat** - Beat para monitoramento a nível de sistema como nível de uso de CPU, memória, sistemas de arquivos, I/O, redes, processos, etc...
- **Packetbeat** - Beat para monitoramento de protocolos de redes e aplicações;
- **Winlogbeat** - Beat para leitura de logs de eventos do Windows;
- **Auditbeat** - Beat para monitoramento de atividade de usuário e processos. Coleta os mesmos dados que o Auditd;
- **Heartbeat** - Beat para monitoramento de Uptime de serviços e hosts.

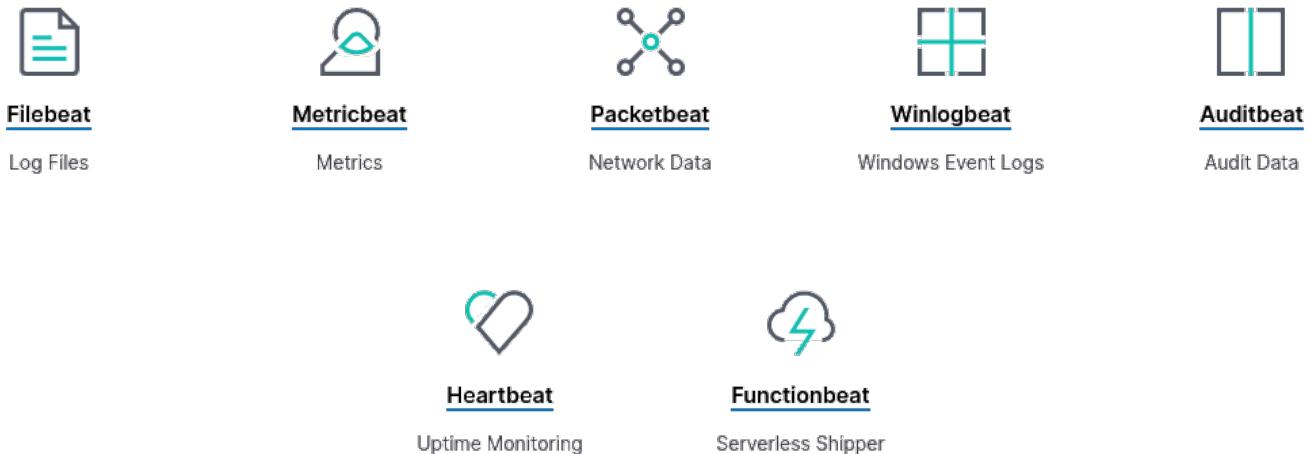


Figura 11.10: Beats Family

- **Functionbeat** - Beat para monitoramento de infraestrutura de cloud, seu deploy é feito como função em frameworks serverless, como por exemplo AWS Lambda.

## Community Beats

Beats não são desenvolvidos somente pela Elastic, uma vez que são Open Source, toda a comunidade contribui no desenvolvimento de novos beats chamados de **Community Beats**

Atualmente existem mais de 90 Community Beats os quais atendem diversas áreas não cobertas pelos Beats tradicionais da Elastic Co.

*A lista completa de community beats pode ser vista na documentação oficial*

## Filebeat

O Filebeat foi desenvolvido para atender aqueles usuários que queriam apenas executar um **tail** em um arquivo de log de maneira simples, agregando o mesmo a possibilidade de pesquisa e centralizando o mesmo.

Iremos utilizar o Filebeat para capturar logs de sistema de nossos servidores da nossa infraestrutura e visualizar no Kibana.

## Metricbeat

O Metricbeat foi desenvolvido para atender os usuários que queriam monitorar servidores em nível de sistema, coletando dados como: Uso de CPU, Memória, Sistema de Arquivos, I/O de Disco, I/O e estatísticas de redes.

Iremos utilizar o Metricbeat para visualizar as métricas do nosso servidor de logs

## Instalando e Configurando o Metricbeat

Acesse a máquina log e troque para o usuário root

```
1 vagrant ssh log
2 sudo su -
```

Atualize a lista de repositórios e instale o pacote metricbeat

```
1 apt update
2 apt install metricbeat -y
```

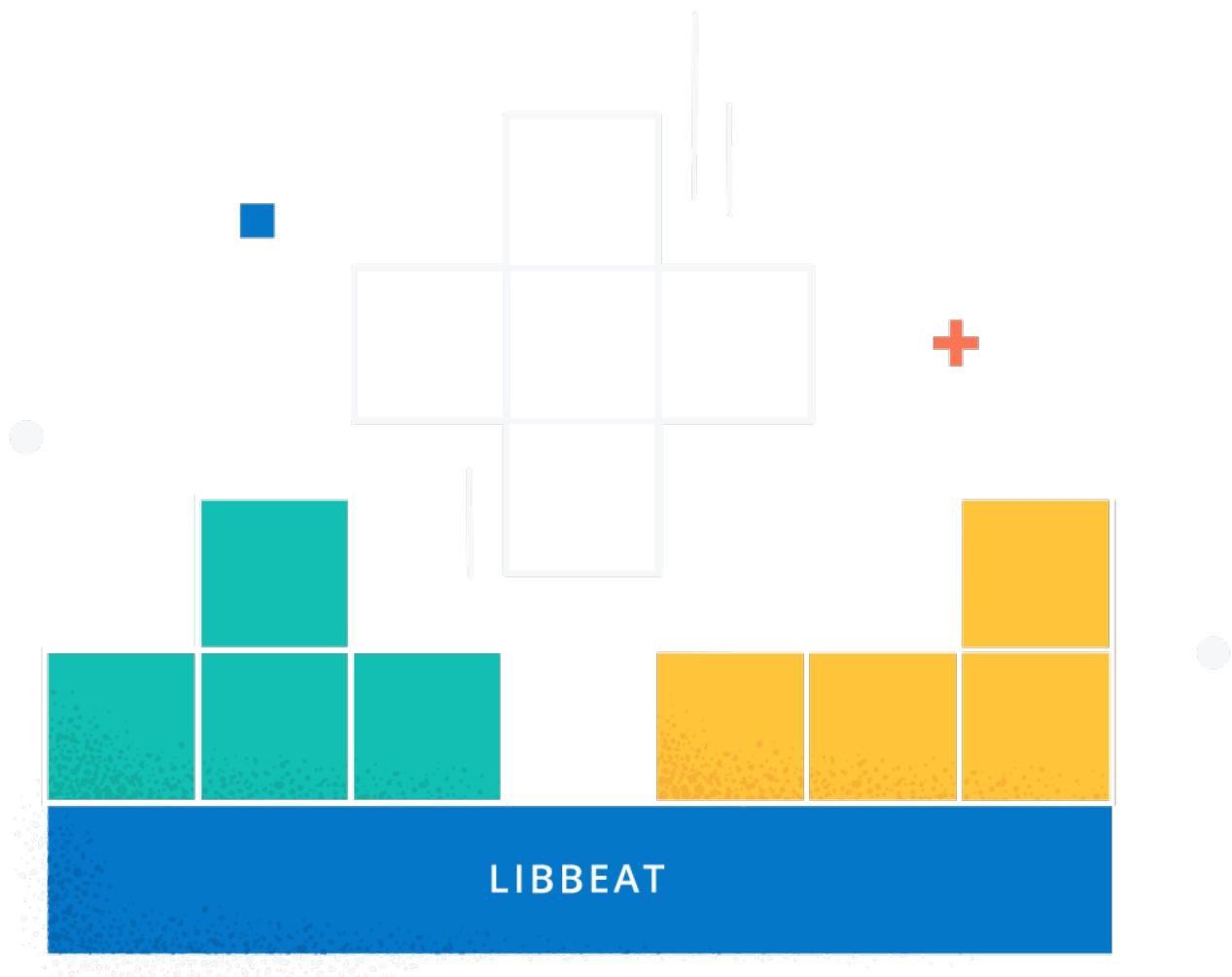


Figura 11.11: libbeat



Figura 11.12: Filebeat



Figura 11.13: Metricbeat

Limpe o conteúdo do arquivo **metricbeats.yml** e adicione as configurações

```
1 > /etc/metricbeat/metricbeat.yml
2 vim /etc/metricbeat/metricbeat.yml
```

```
1 metricbeat.config.modules:
2   path: ${path.config}/modules.d/*.yml
3   reload.enabled: false
4 output.logstash:
5   hosts: ["log.4labs.example:5443"]
6 processors:
7   - add_host_metadata: ~
8   - add_cloud_metadata: ~
```

Dentro da pasta **modules.d** existem mais de 40 modelos de módulos do metricbeat. Podemos utilizar qualquer um deles para nossas máquinas, por padrão apenas o modulo **system** vem habilitado, o qual monitora diversos dados relacionados ao sistema

Execute a configuração inicial do ambiente

```
1 metricbeat setup --dashboards
```

O comando **metricbeat setup -dashboards** faz a configuração inicial do ambiente bem como a configuração dos dashboards do kibana

Habilite e inicie o serviço do metricbeats

```
1 systemctl enable metricbeat
2 systemctl start metricbeat
```

Verifique se as métricas estão sendo exportadas

```
1 curl http://localhost:9200/_cat/indices/metricbeat*?v
2 curl -XGET 'http://localhost:9200/metricbeat-*/_search?pretty'
```

Os comandos acima fazem a listagem dos índices de métricas e a coleta de eventos no formato raw

## Construindo Dashboards com Kibana

### Kibana

Kibana é uma ferramenta que nos permite visualizar os dados do Elasticsearch e navegar no Elastic Stack. Através dele podemos rastrear dados, gerar gráficos e dashboards para entender a maneira que as requisições e dados caminham sobre nossas aplicações.

### Primeiros passos com Kibana

Acesse a webpage do kibana através do endereço <http://log.4labs.example>

Após o sistema carregar, será exibida a tela inicial do kibana

### Index Patterns

Primeiramente precisamos configurar os padrões de índices, clique no ícone para expandir o menu

Clique em **Management**

Clique em **Index Patterns**



# kibana

Figura 11.14: kibana

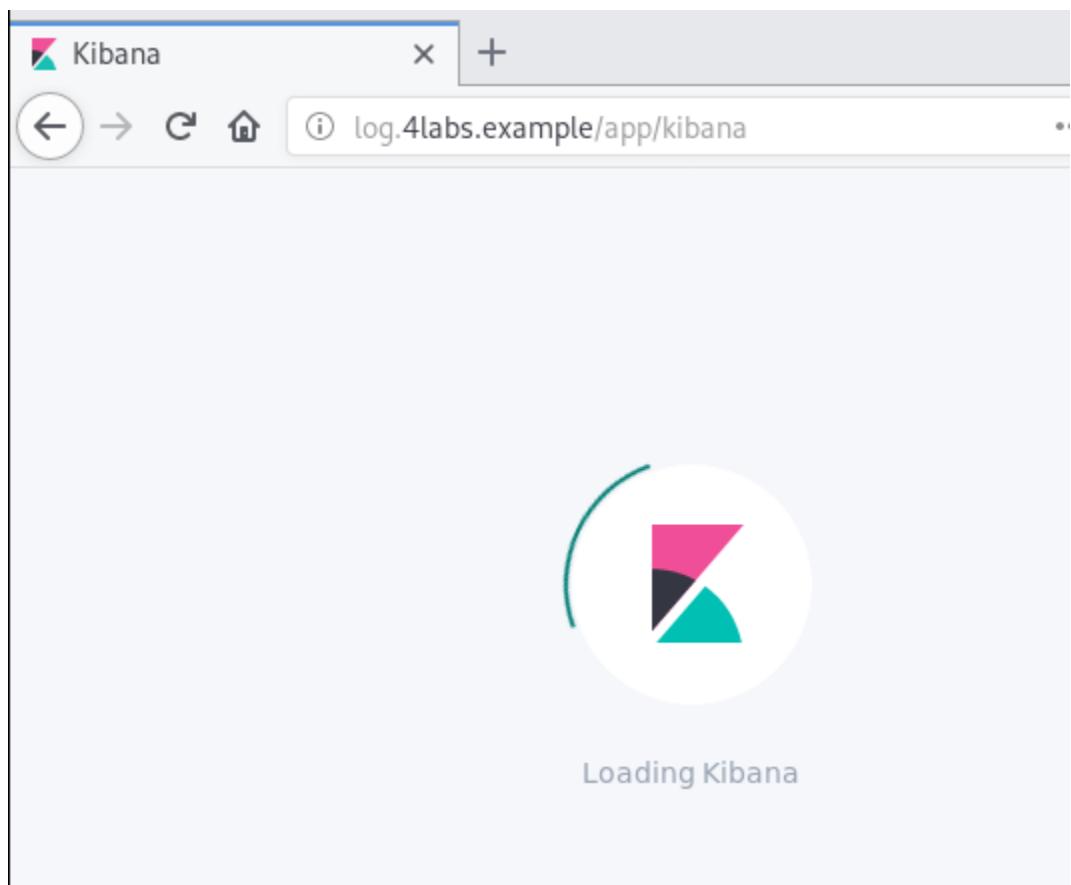


Figura 11.15: kibana

The screenshot shows the Kibana interface with the following sections:

- Add Data to Kibana:**
  - APM:** Automatically collects in-depth performance metrics and errors from inside your applications. Includes a "Add APM" button.
  - Logging:** Ingest logs from popular data sources and easily visualize in preconfigured dashboards. Includes a "Add log data" button.
  - Metrics:** Collect metrics from the operating system and services running on your servers. Includes a "Add metric data" button.
  - SIEM:** Centralize security events for interactive investigation in ready-to-go visualizations. Includes a "Add security events" button.

Below these are three data input options:

  - Add sample data:** Load a data set and a Kibana dashboard.
  - Upload data from log file:** Import a CSV, NDJSON, or log file.
  - Use Elasticsearch data:** Connect to your Elasticsearch index.
- Visualize and Explore Data:**
  - APM:** Automatically collect in-depth performance metrics and errors from inside your applications.
  - Dashboard:** Display and share a collection of visualizations.
  - Canvas:** Showcase your data in a pixel-perfect way.
  - Discover:** Interactively explore your data by querying and filtering raw documents.
- Manage and Administer the Elastic Stack:**
  - Console:** Skip cURL and use this JSON interface to work with your data directly.
  - Monitoring:** Track the real-time health and performance of your Elastic Stack.
  - Index Patterns:** Manage the index patterns that help retrieve your data from Elasticsearch.
  - Rollups:** Summarize and store historical data in a smaller index for future analysis.

Figura 11.16: kibana

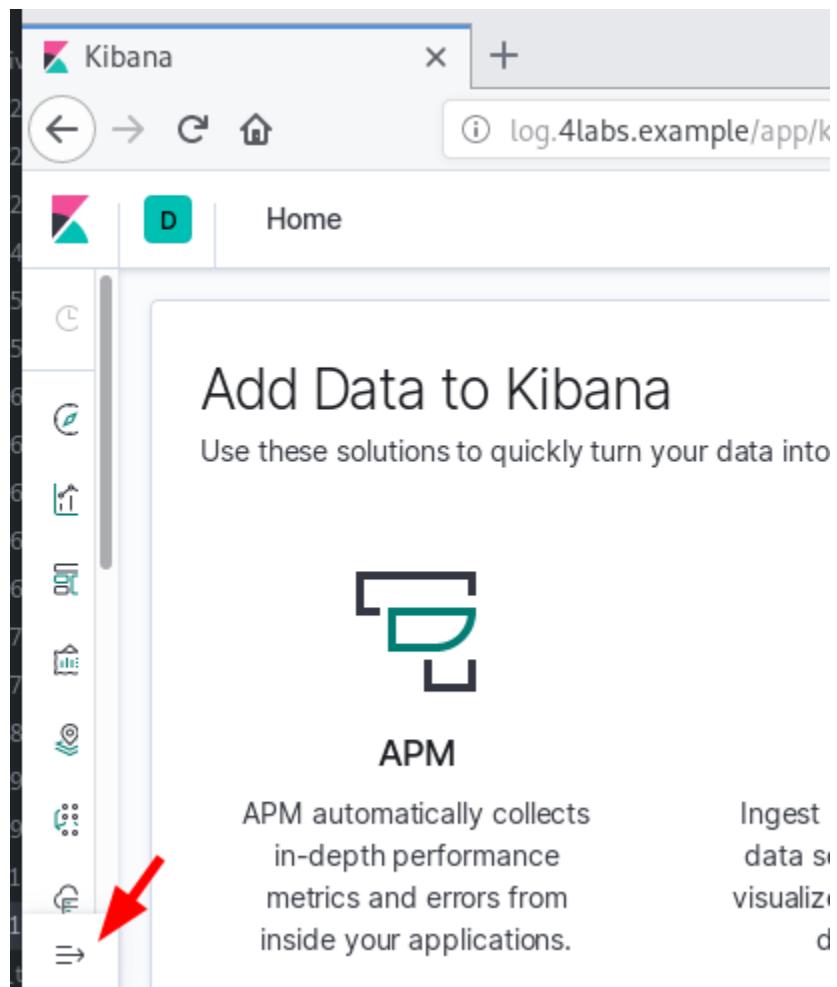


Figura 11.17: kibana

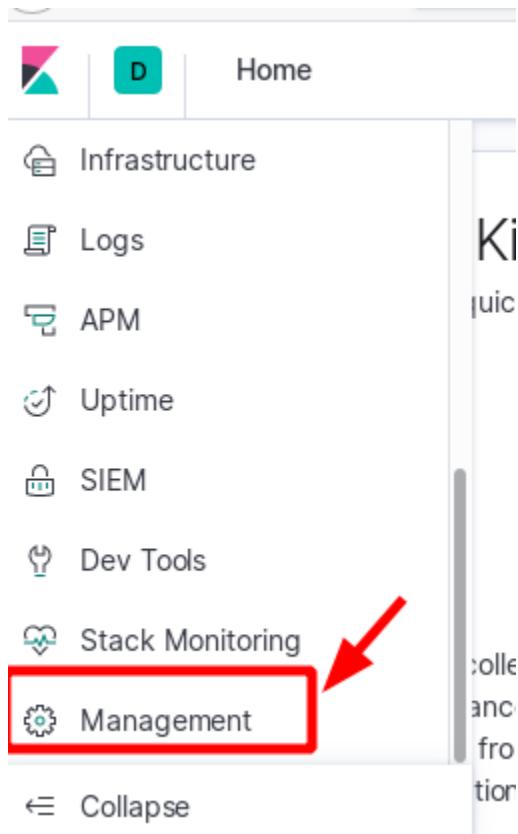


Figura 11.18: kibana

Clique em **Create Index Patterns**

Preencha o Pattern com o valor **metricbeat\*** e clique em **Next Step**

Selecione no dropdown o parâmetro **@timestamp** e clique em **Create Index Pattern**

Será exibida uma tela com os valores listados pelo índice **metricbeat\***

Repita os passos para adicionar um novo índice para o **filebeat\***

Selecione no dropdown o parâmetro **@timestamp** e clique em **Create Index Pattern**

Será exibida uma tela com os valores listados pelo índice **filebeat\***

## Visualizando os Dados

Para visualizar os dados no Kibana, basta clicar no ícone **Discovery** e será exibida uma tela com os Index Patterns e seus dados.

Através desta tela podemos listar os dados por Index Pattern ou até mesmo efetuar seleções e filtros

Vamos selecionar o Index Pattern **filebeat\*** para visualizar as métricas disponíveis

Podemos filtrar a pesquisa por data/tempo no menu de filtro por datas

Para adicionar um filtro clique em cima do campo no menu de campos disponíveis e em seguida clique na lupa com o zoom +

O filtro será adicionado e os campos correspondentes nos logs ficarão realçados

Também é possível adicionar filtros no campo **Filters** e clicar em refresh, um filtro composto será montado e suas correspondências serão realçadas.

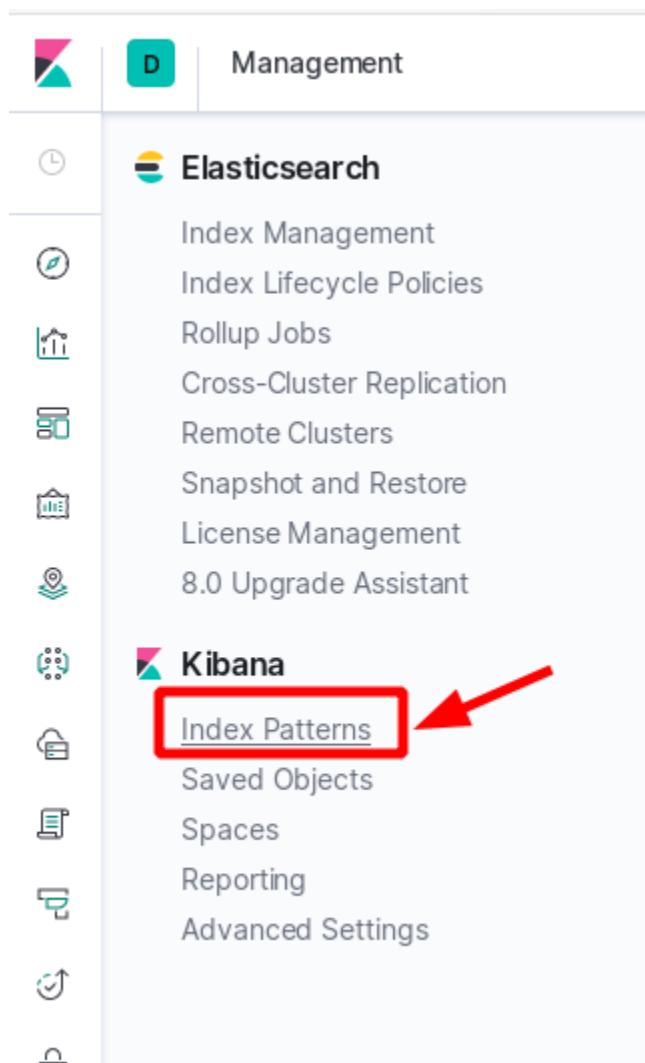


Figura 11.19: kibana

# Create index pattern

Kibana uses index patterns to retrieve data from Elasticsearch indices for things like visualizations.

Include system indices

## Step 1 of 2: Define index pattern

### Index pattern

metricbeat\*

You can use a \* as a wildcard in your index pattern.

You can't use spaces or the characters \, /, ?, ", <, >, |.

 [Next step](#)

✓ **Success!** Your index pattern matches **1 index**.

**metricbeat-2019.11.16**

Rows per page: 10 ▾

Figura 11.20: kibana

## Create index pattern

Kibana uses index patterns to retrieve data from Elasticsearch indices for things like visualizations.

X Include system indices

### Step 2 of 2: Configure settings

You've defined **metricbeat\*** as your index pattern. Now you can specify some settings before we create it.

Time Filter field name Refresh

@timestamp	<input type="button" value="▼"/>
------------	----------------------------------

The Time Filter will use this field to filter your data by time.

You can choose not to have a time field, but you will not be able to narrow down your data by a time range.

[Show advanced options](#)

[Back](#)

[Create index pattern](#)

Figura 11.21: kibana

## ★ metricbeat\*



Time Filter field name: @timestamp Default

This page lists every field in the **metricbeat\*** index and the field's associated core type as recorded by Elasticsearch. To change a field type, use the Elasticsearch [Mapping API](#).

Fields (232)	Scripted fields (0)	Source filters (0)			
Filter		All field types ▾			
Name	Type	Format	Search...	Aggre...	Excluded
@timestamp	date		•	•	
@version	string		•		
@version.keyword	string		•	•	
_id	string		•	•	
_index	string		•	•	

Figura 11.22: kibana

### Step 1 of 2: Define index pattern

#### Index pattern

filebeat\*

You can use a \* as a wildcard in your index pattern.  
You can't use spaces or the characters \, /, ?, ", <, >, |.

> [Next step](#)

✓ Success! Your index pattern matches 11 indices.

Figura 11.23: kibana

## Step 2 of 2: Configure settings

You've defined **filebeat\*** as your index pattern. Now you can specify some settings before we create it.

Time Filter field name Refresh

 ▼

The Time Filter will use this field to filter your data by time.  
You can choose not to have a time field, but you will not be able to  
narrow down your data by a time range.

> Show advanced options

< Back

Create index pattern

Figura 11.24: kibana

## filebeat\*



Time Filter field name: @timestamp

This page lists every field in the **filebeat\*** index and the field's associated core type as recorded by Elasticsearch. To change a field type, use the Elasticsearch [Mapping API](#).

Fields (65)

Scripted fields (0)

Source filters (0)

Filter

All field types ▾

Name	Type	Format	Search...	Aggre...	Excluded
@timestamp ⓘ	date		●	●	✎
@version	string		●		✎
@version.keyword	string		●	●	✎
_id	string		●	●	✎
_index	string		●	●	✎
_score	number				✎

Figura 11.25: kibana

## filebeat\*



Time Filter field name: @timestamp

This page lists every field in the **filebeat\*** index and the field's associated core type as recorded by Elasticsearch. To change a field type, use the Elasticsearch [Mapping API](#).

Fields (65)

Scripted fields (0)

Source filters (0)

Filter

All field types ▾

Name	Type	Format	Search...	Aggre...	Excluded
@timestamp ⓘ	date		●	●	✎
@version	string		●		✎
@version.keyword	string		●	●	✎
_id	string		●	●	✎
_index	string		●	●	✎
_score	number				✎

Figura 11.26: kibana

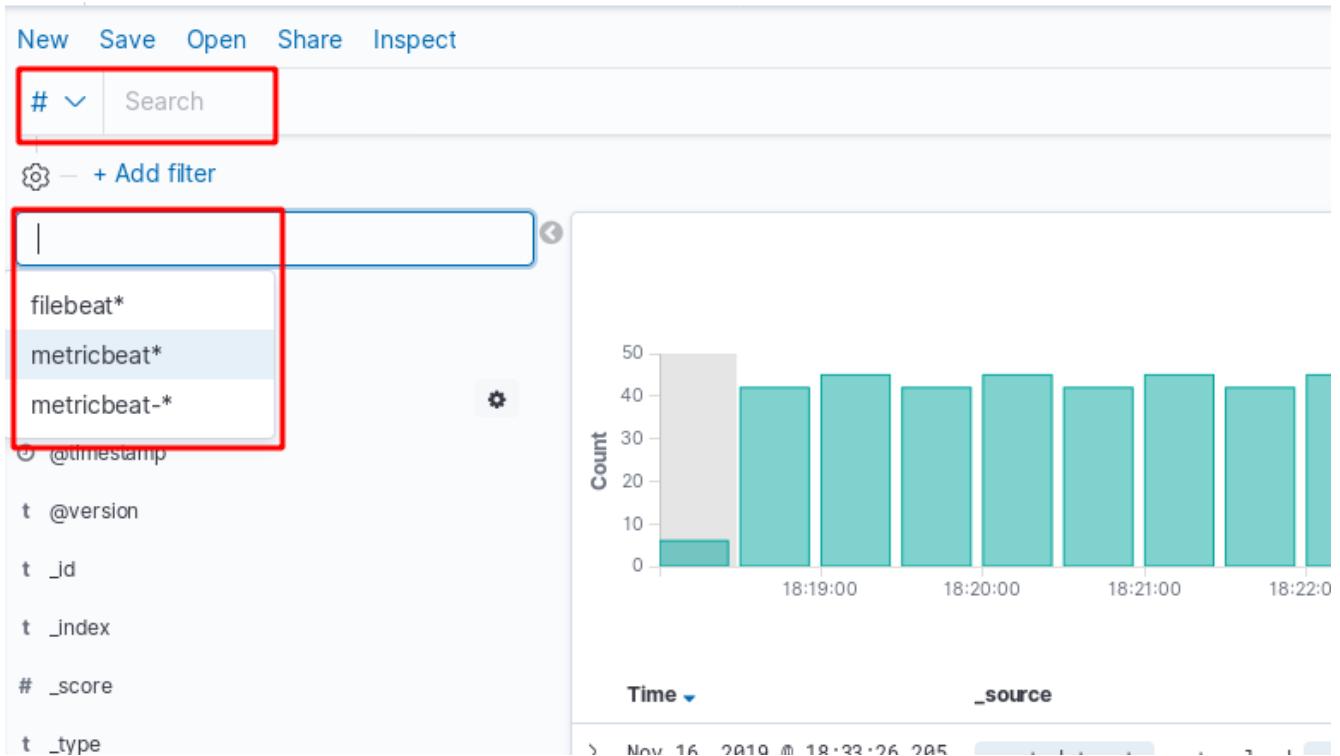


Figura 11.27: kibana

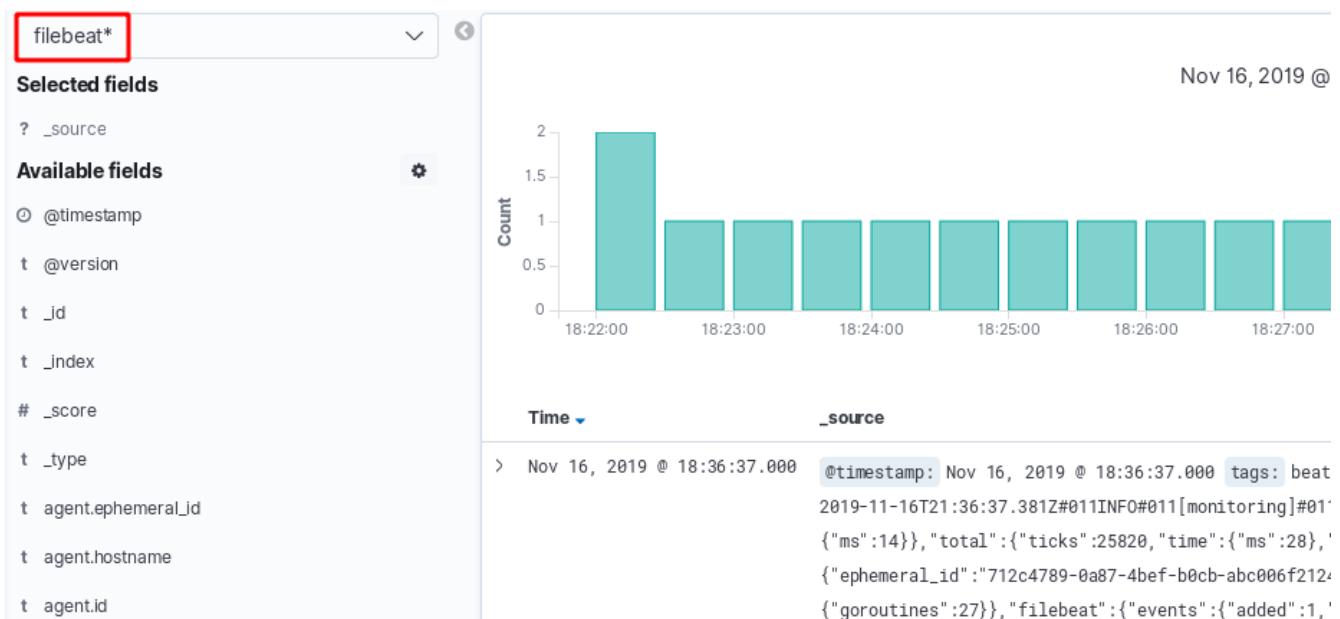


Figura 11.28: kibana

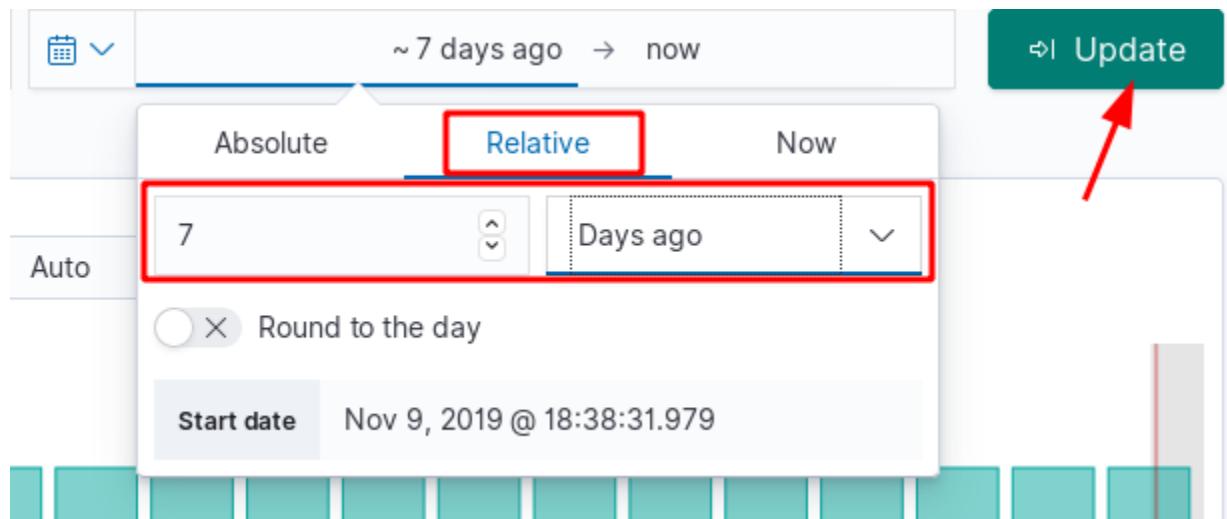


Figura 11.29: kibana

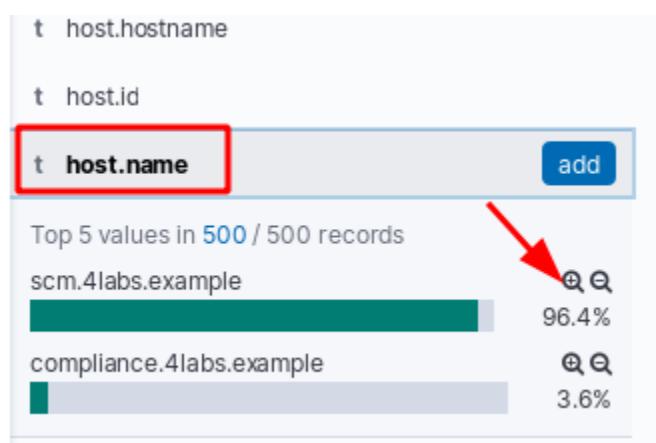


Figura 11.30: kibana

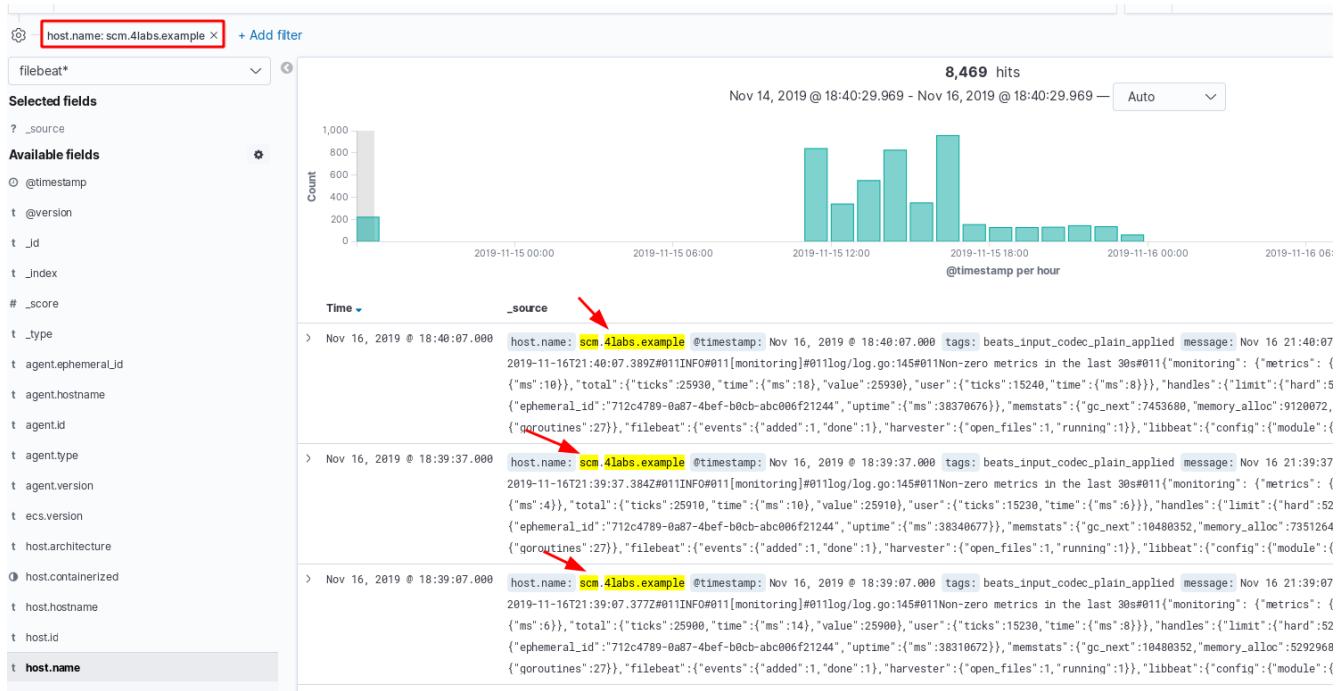


Figura 11.31: kibana

Explore as possibilidades, mas lembre-se de limpar os filtros anteriores ou sua pesquisa pode não retornar nenhum resultado

## Construindo Dashboards

Um dos recursos mais interessantes do Kibana é a possibilidade de criar dashboards com as informações que sejam importantes para o usuário final, podemos criar gráficos de acordo com a nossa necessidade. Para criar um dashboard, clique em expandir o menu

Clique em dashboard

Será exibida uma tela com diversos dashboards disponíveis e um campo de pesquisa

Vamos criar um novo dashboard clicando no botão **Create new dashboard**

Clique em **Add** para adicionar uma visualização

Adicione alguns campos a visualização, basta clicar no painel que o mesmo é adicionado instantâneamente. Ao terminar clique no **X** para ir a visualização

Clique em **Save** para salvar o dashboard e dê um nome ao mesmo

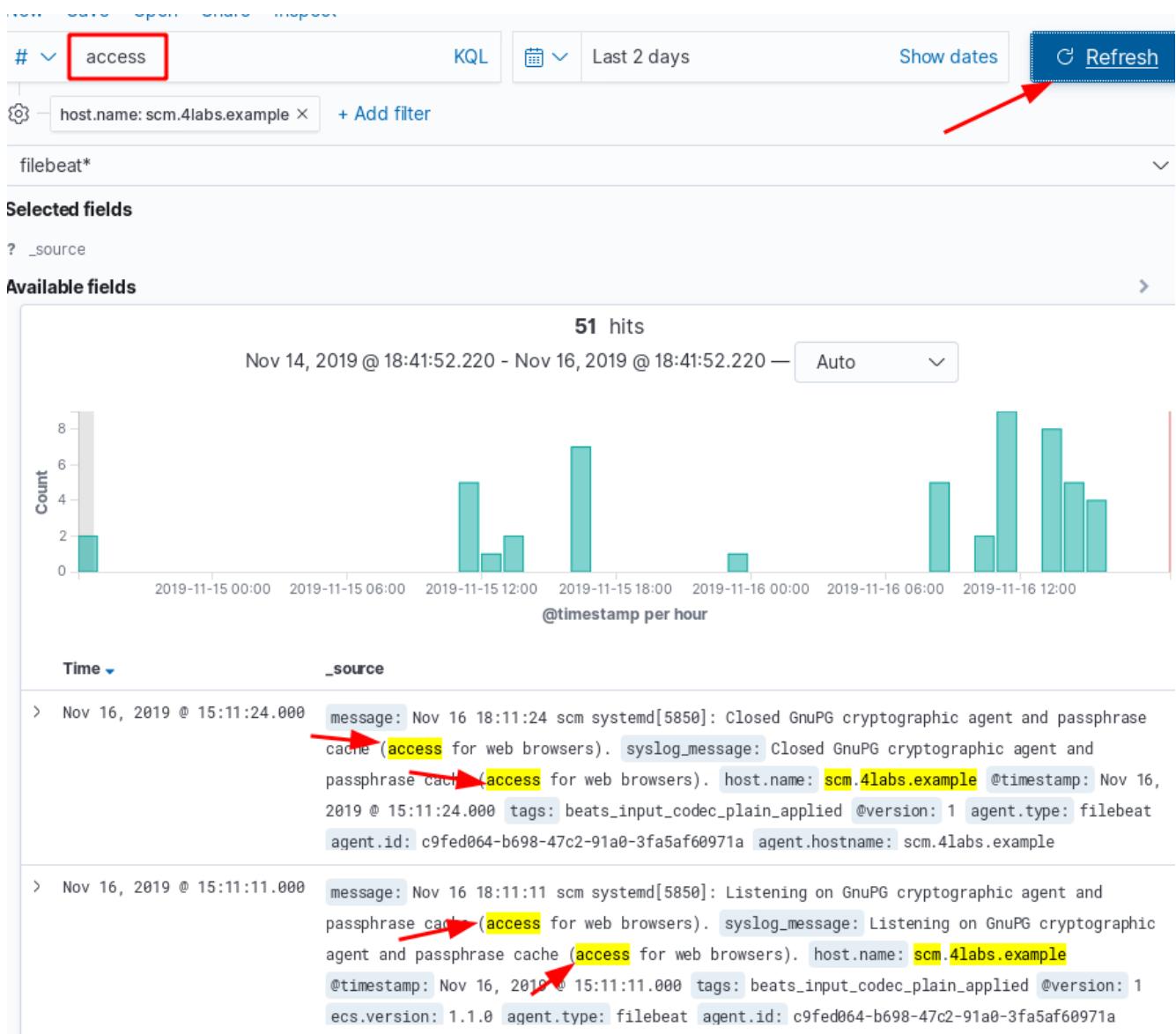


Figura 11.32: kibana

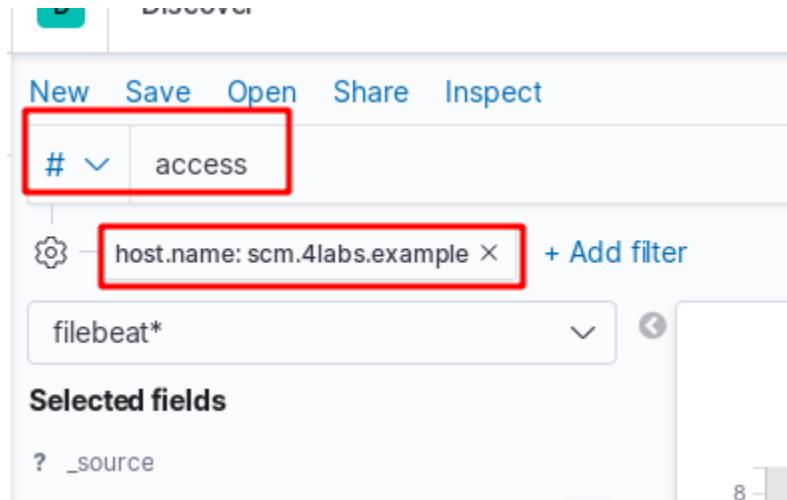


Figura 11.33: kibana

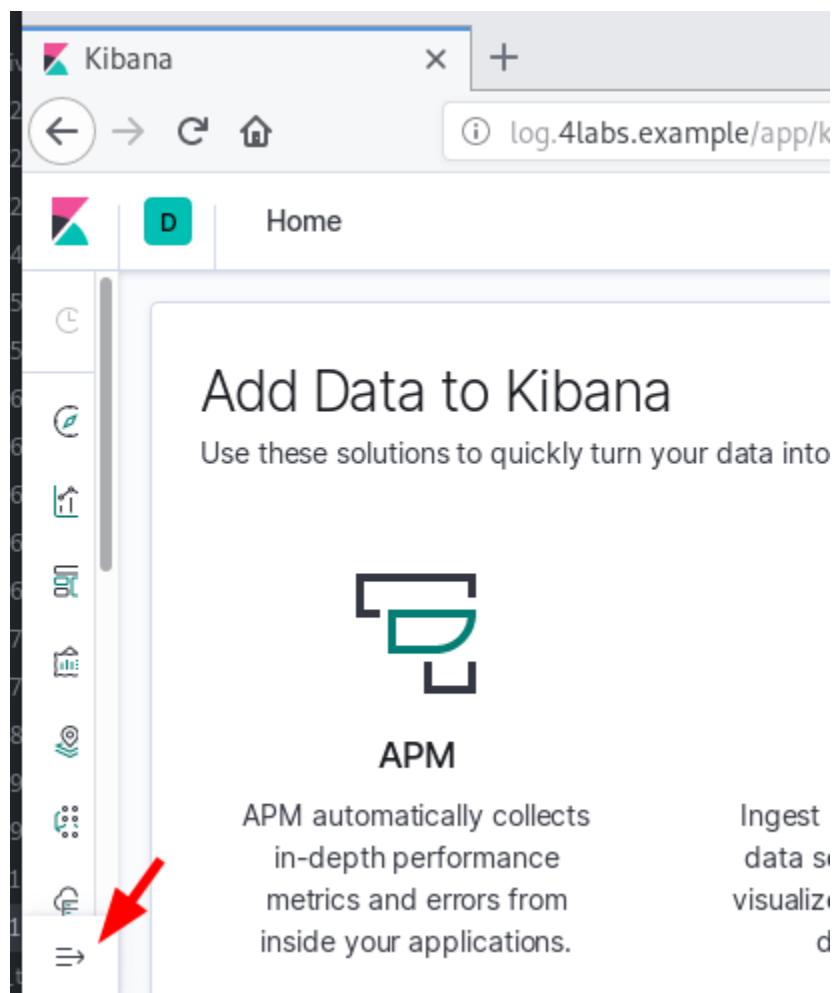


Figura 11.34: kibana

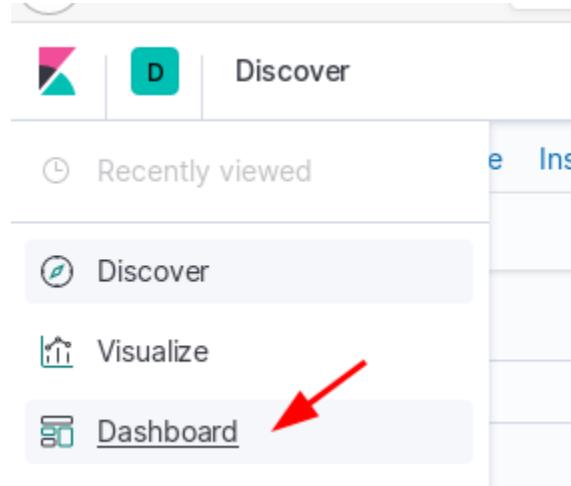


Figura 11.35: kibana

## Dashboards

Search...

<input type="checkbox"/> Title	Description	Actions
<a href="#">[Metricbeat AWS] EBS Overview</a>	[Metricbeat AWS] Overview of EBS Metrics	
<a href="#">[Metricbeat AWS] EC2 Overview</a>	Overview of AWS EC2 Metrics	
<a href="#">[Metricbeat AWS] ELB Overview</a>	Overview of AWS ELB Metrics	
<a href="#">[Metricbeat AWS] Overview</a>	Overview of AWS Metrics	
<a href="#">[Metricbeat AWS] RDS Overview</a>	Overview of AWS RDS Metrics	
<a href="#">[Metricbeat AWS] S3 Overview</a>	Overview of AWS S3 Metrics	
<a href="#">[Metricbeat AWS] SQS Overview</a>	Overview of AWS SQS Metrics	
<a href="#">[Metricbeat Aerospike] Database Overview</a>	This Aerospike dashboard visualizes the most important metrics for Aerospike namespaces.	
<a href="#">[Metricbeat Apache] Overview ECS</a>	Overview of Apache server status	
<a href="#">[Metricbeat Ceph] Cluster Overview</a>	This Ceph dashboard that shows the most important cluster metrics.	

Rows per page: 10 ▾

< 1 2 3 4 5 >

Figura 11.36: Dashboard

The screenshot shows the Grafana Dashboards page. At the top, there is a search bar labeled "Search...". Below it is a table with three columns: "Title", "Description", and "Actions". There are two entries in the table:

Title	Description	Actions
[Metricbeat AWS] EBS Overview	[Metricbeat AWS] Overview of EBS Metrics	
[Metricbeat AWS] EC2 Overview	Overview of AWS EC2 Metrics	

In the top right corner of the table area, there is a blue button with a white plus sign and the text "Create new dashboard". A red arrow points from the text "Figura 11.37: Dashboard" to this button.

Figura 11.37: Dashboard

The screenshot shows the "Editing New Dashboard" interface. At the top, there is a header with a logo, a title "Dashboard / Editing New Dashboard", and a save/cancel button. Below the header is a toolbar with several icons: a clock, a magnifying glass, a gear, and a bar chart. To the right of the toolbar is a menu bar with "Save", "Cancel", "Add", "Options", and "Share". The "Add" button is highlighted with a red arrow. Below the toolbar is a search bar with a dropdown menu showing "# <down arrow>" and a "Search" input field. At the bottom of the toolbar is a " + Add filter" button.

Figura 11.38: Dashboard

## Add panels

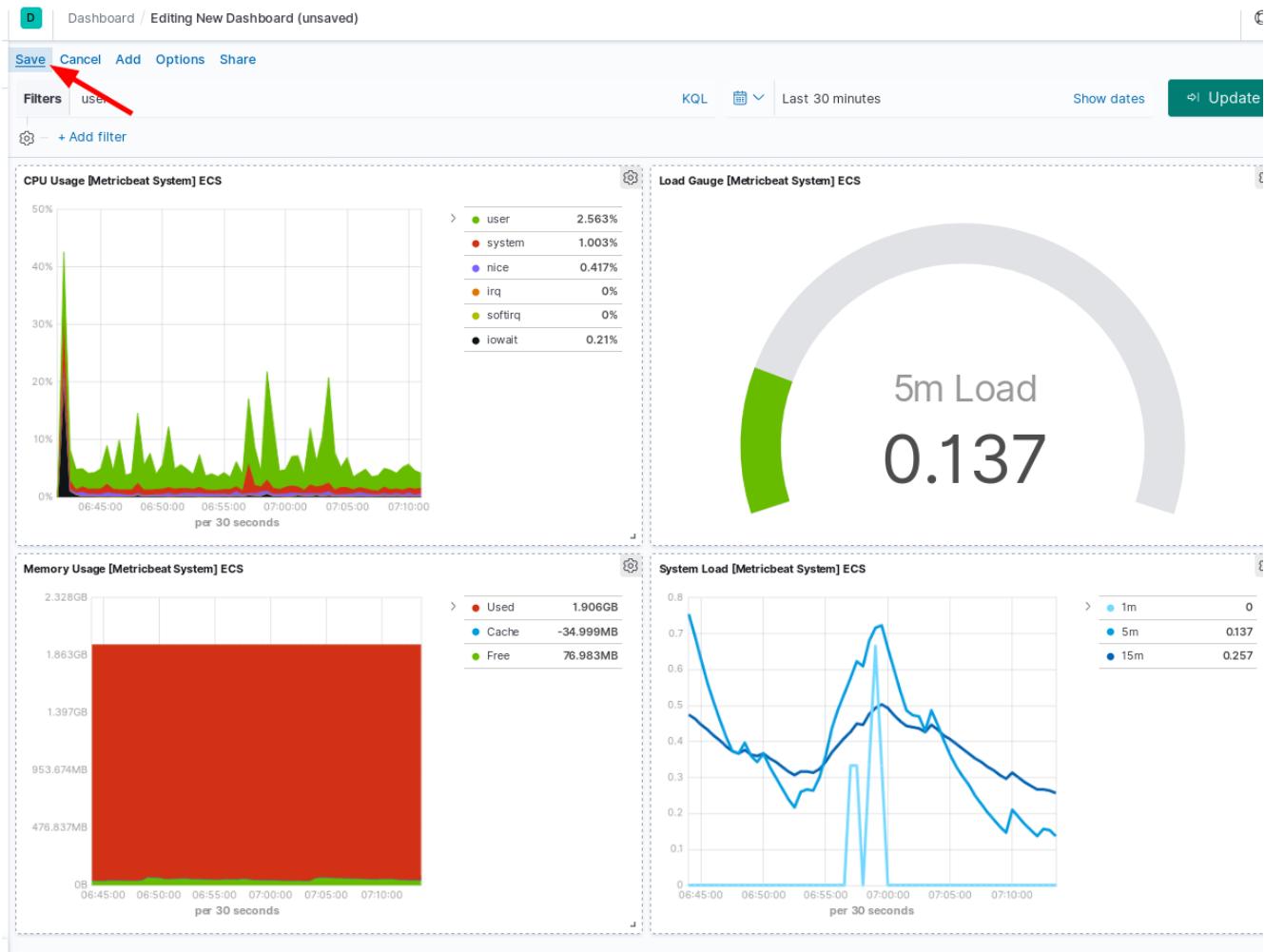
Q Metricbeat System ECS

Sort ▾ Types 3 ▾

- System Load [Metricbeat System] ECS
- System Navigation [Metricbeat System] ECS
- Packetloss [Metricbeat System] ECS
- Tip [Metricbeat System] ECS
- System [Metricbeat Golang] ECS
- Memory Usage [Metricbeat System] ECS
- CPU Usage [Metricbeat System] ECS
- Load Gauge [Metricbeat System] ECS
- Disk Usage [Metricbeat System] ECS
- Swap usage [Metricbeat System] ECS

Rows per page: 10 ▾ < 1 2 3 4 >

Figura 11.39: Dashboard



## Save dashboard

### Title

Metricbeat - log.4labs.example

### Description

Meu Primeiro Dashboard

Store time with dashboard

This changes the time filter to the currently selected time each time this dashboard is loaded.

[Cancel](#)

[Confirm Save](#)

Podemos também selecionar um dashboard pronto para utilizar, basta filtrar e clicar no nome do dashboard

The screenshot shows the Kibana interface. On the left, there's a sidebar with various icons: a magnifying glass, a document, a house, a gear, a cloud, and a gear. A red arrow points to the 'house' icon. Next to it is a green button with a white 'D'. To the right of these is a 'Dashboards' tab. Below the sidebar is a search bar containing the text 'Host Overview'. A red box highlights this search term. Below the search bar is a 'Title' field with the placeholder '[Metricbeat System] Host overview ECS'. At the bottom right of the sidebar, there's a 'Rows per page: 10' dropdown. A red arrow points to the 'Host Overview' search result.

Figura 11.40: Dashboard

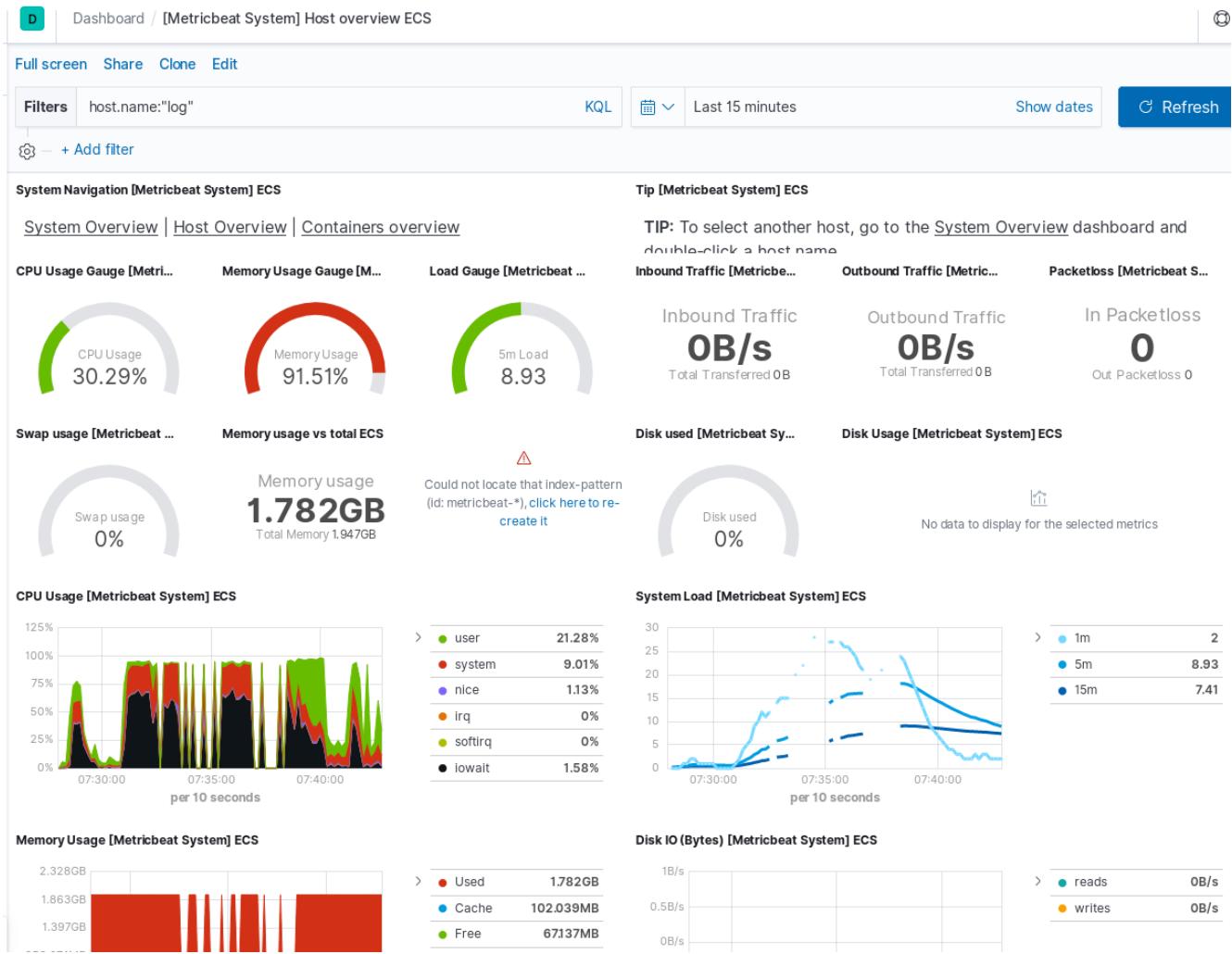


Figura 11.41: Dashboard

Podemos criar visualizações através do menu **Visualize**

The screenshot shows the Grafana dashboard. At the top left is the Grafana logo. Next to it is a blue button with a white 'D' and the word 'Visualize'. Below this is a sidebar with several icons: a clock, a magnifying glass, a house (highlighted with a red arrow), a gear, and a document. The main area has a title 'Visualizations' and a search bar labeled 'Search...'. Below the search bar are three columns: 'Title' (with an empty input field), 'Type' (with a dropdown menu), and 'Actions' (with a 'Create new visualization' button highlighted with a red arrow). A red arrow also points to the house icon in the sidebar.

Figura 11.42: Dashboard

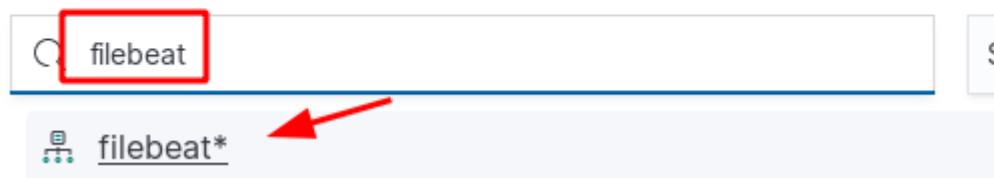
Crie uma visualização do tipo **Tag Cloud** e selecione a fonte **filebeat\***

The screenshot shows the 'New Visualization' dialog. At the top left is a 'Filter' search bar. On the right, there is a section titled 'Tag Cloud' with the description: 'A group of words, sized according to their importance'. Below this are four rows of visualization icons:

- Row 1: Area, Controls, Coordinate Map, Data Table
- Row 2: Gauge, Goal, Heat Map, Horizontal Bar
- Row 3: Line, Maps, Markdown, Metric
- Row 4: Pie, Region Map, TSVB, Tag Cloud (highlighted with a red arrow)

A large pink diagonal shape is visible on the right side of the dialog.

## New Tag Cloud / Choose a source



Configure o Metrics para **Tag Size / Count** e adicione um novo **Bucket Tags**

Metrics

Tag size

Aggregation Count

Custom label

Advanced

Buckets

Add

ADD BUCKET

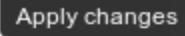
Tags

Figura 11.43: Dashboard

Edite os campos **Aggregation**, **Field**, **Order**, **Size** e clique em **Apply Changes**

filebeat\*

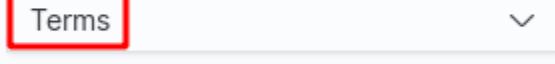
Data Options  

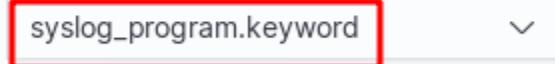
Custom label 

> Advanced

**Buckets**

Tags

Aggregation Terms help 

Field 

Order by Metric: Count 

Order Ascending  Size  10 

Group other values in separate

Figura 11.44: Dashboard

Será exibido um gráfico de cloud com as ocorrências em ordem de tamanho, quanto maior mais ocorrências.

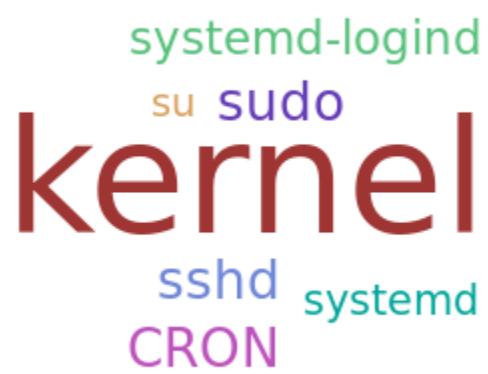


Figura 11.45: Dashboard

## Capítulo 12

# Implementando ChatOps com Rocket.Chat

### Conceitos e Características ChatOps

#### O que é ChatOps?

**ChatOps** é a junção das tarefas de Automação e Colaboração. É delegar responsabilidade de tarefas e ações para um robô interno que também faz parte da organização e está escutando as conversas.

**ChatOps** não é somente tecnologia, é também uma metodologia e uma mudança cultural, assim como DevOps

**ChatOps = Chat + DevOps**

#### História do ChatOps

O conceito de ChatOps não é algo novo, é basicamente a utilização de ferramentas de colaboração para executar tarefas através de alguma aplicação.

O termo **ChatOps** foi inventado pelo **GitHub**, sendo dado o crédito pelo uso do mesmo na conferência Rubyfuza 2013 e rapidamente se espalhou por todas as comunidades na internet sobre DevOps, SRE e operações modernas de TI. Porém a abordagem já era utilizada desde a fundação da GitHub em 2008.

#### Como funciona?

O funcionamento do ChatOps é simples, utilizando a aplicação de Chat, adicionamos bots para executar ações em nossa infraestrutura através de comandos enviados pelas equipes DevOps

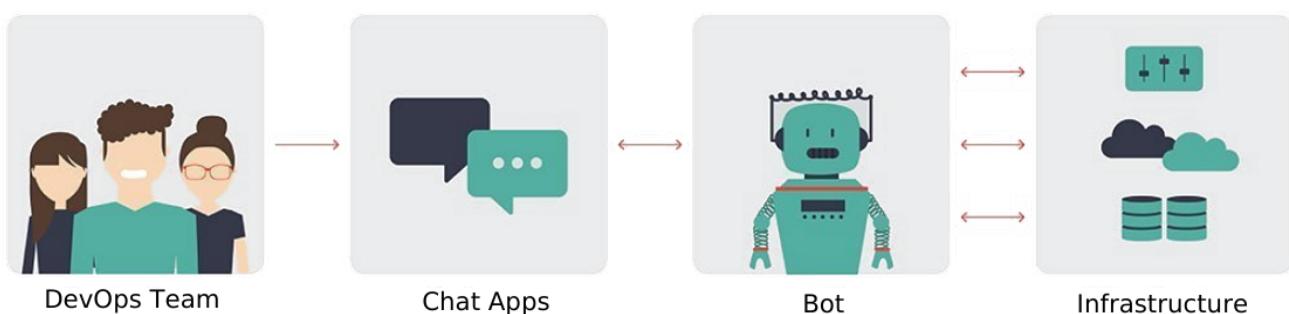


Figura 12.1: ChatOps

## Benefícios do ChatOps

A utilização de ChatOps nos ajuda principalmente nos quesitos:

- **Team Scaling** - O time pode crescer facilmente sem a preocupação da comunicação e explicação minuciosa de processos e atividades que serão executadas pois o histórico de problemas e alterações está atrelado ao Chat.
- **Comandos Resumidos** - Comandos complexos podem ser resumidos através de frases simples como:
  - faça o deploy da aplicação X
  - me mostre as issues do repositório Y
  - quem efetuou o commit #1a2b3c ?
- **Contexto Compartilhado** - Como todas as ações são compartilhadas através de canais, a equipe tem o contexto equalizado de todas as situações
- **Inline History** - Todo o histórico de atividades é registrado diretamente no chat, sendo facilmente acompanhado por qualquer usuário
- **Rápida resposta a incidentes** - Em um ambiente mais maduro com relação a ChatOps podem ser implementadas ações para resposta e solução de incidentes de maneira ágil.

## Estrutura ChatOps

Para nossa estrutura ChatOps utilizaremos a ferramenta de colaboração **Rocket.Chat** efetuando a conexão com o Gogs e o Rundeck através de webhook e o **Hubot** como nosso ChatBot para disparo de Jobs

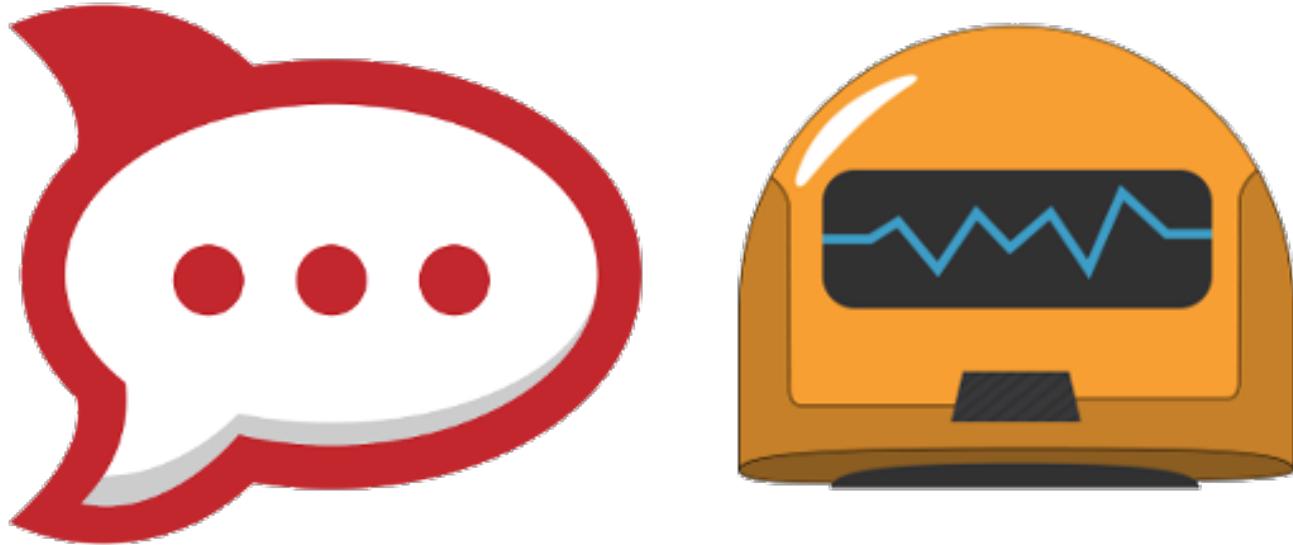


Figura 12.2: Rocket.Chat e Hubot

## Instalando e Configurando o Rocket.Chat

### Arquitetura do Rocket.Chat

Iremos subir uma estrutura utilizando docker-compose para provisionar nosso servidor de chat.

Nossa configuração será realizada através de 4 containers:

- **traefik** - Proxy reverso para o container do rocketchat;
- **rocketchat** - Aplicação Rocket.Chat executada na porta 3000;
- **mongo** - Banco de Dados NoSQL;
- **mongo-init-replica** - container para iniciar o replica-set no banco criado, este container tem o propósito único de inicializar o replica-set e em seguida o mesmo é removido.

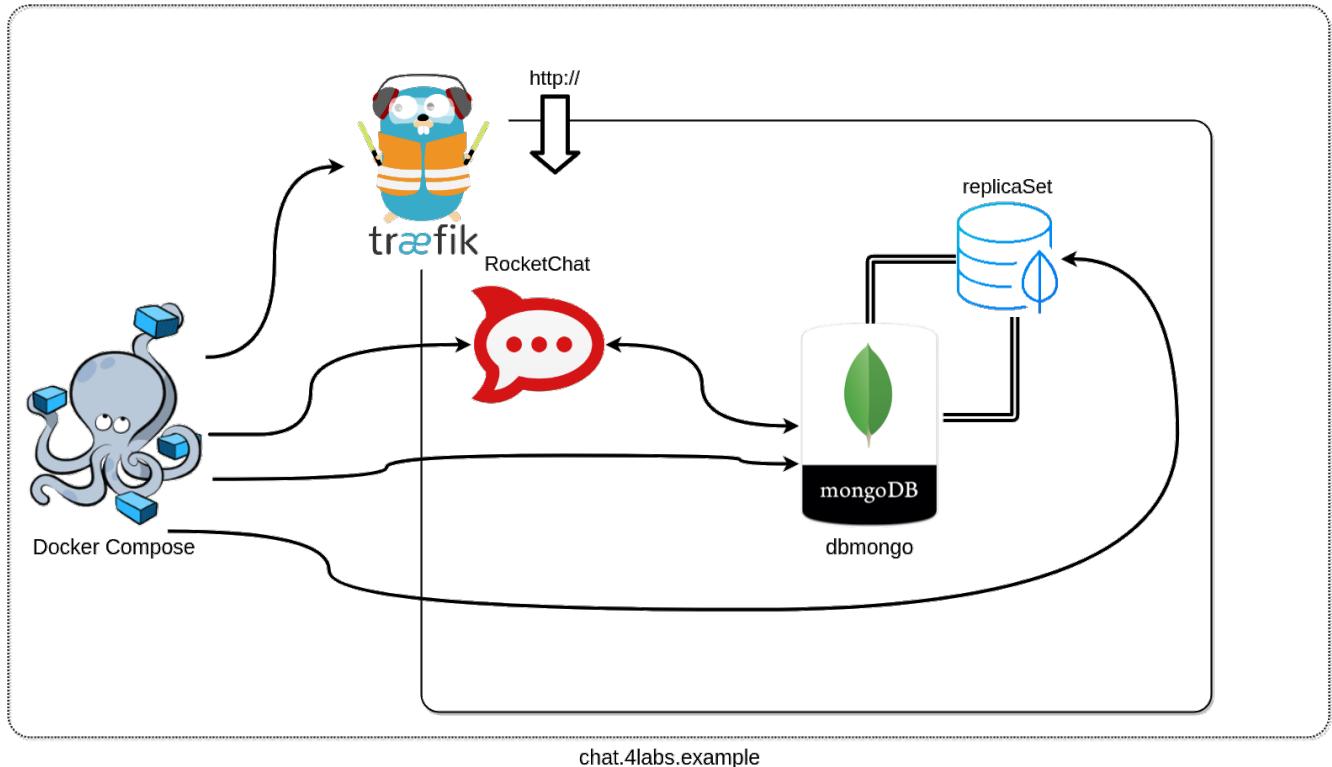


Figura 12.3: Arquitetura

## Traefik

O Traefik é um roteador Open-Source de borda para clouds desenvolvido na linguagem Go.

O traefik é muito utilizado em containers por ser extremamente leve e de fácil configuração, através dele é possível visualizar métricas e utilizar recursos de auto-discovery em Kubernetes, Docker Swarm, Mesos, Rancher, Marathon, etc...

Uma das grandes funcionalidades do traefik é a implementação do lets encrypt, sendo assim o mesmo se encarrega de solicitar o certificado, registrar e efetuar as conexões seguras com as aplicações via SSL bem como a renovação automática do certificado lets encrypt.

## Rocket.Chat

O Rocket.Chat é nossa aplicação de ChatOps propriamente dita. Através dela efetuaremos as configurações e integrações para nossa infraestrutura.

O Rocket.Chat é o maior projeto Open Source do Brasil, sua empresa tem sede em Porto Alegre e o mesmo possui atualmente mais de 500 contribuidores, 5 mil forks e 24 mil estrelas.

Foi idealizado para atender a demanda do mercado com relação a uma ferramenta Open Source de ChatOps para atender diversas necessidades como Omni Channel, Live Chat, etc....

Atualmente empresas como 4Linux, Departamento de Defesa dos Estados Unidos, Policia Federal do Brasil, Caixa Económica Federal, Funai, Banco do Brasil, Samsung, Havan, Agencia Espacial Brasileira, DataPrev, Owncloud, Câmara dos Deputados e BNDES utilizam como ferramenta para ChatOps.

## MongoDB

O MongoDB é um software de banco de dados NoSQL orientado a documentos e Open Source.

MongoDB é um dos bancos NoSQL mais utilizados em todo o mercado, tem código Open Source e multi-plataforma e oferece alta-disponibilidade utilizando replicsets.

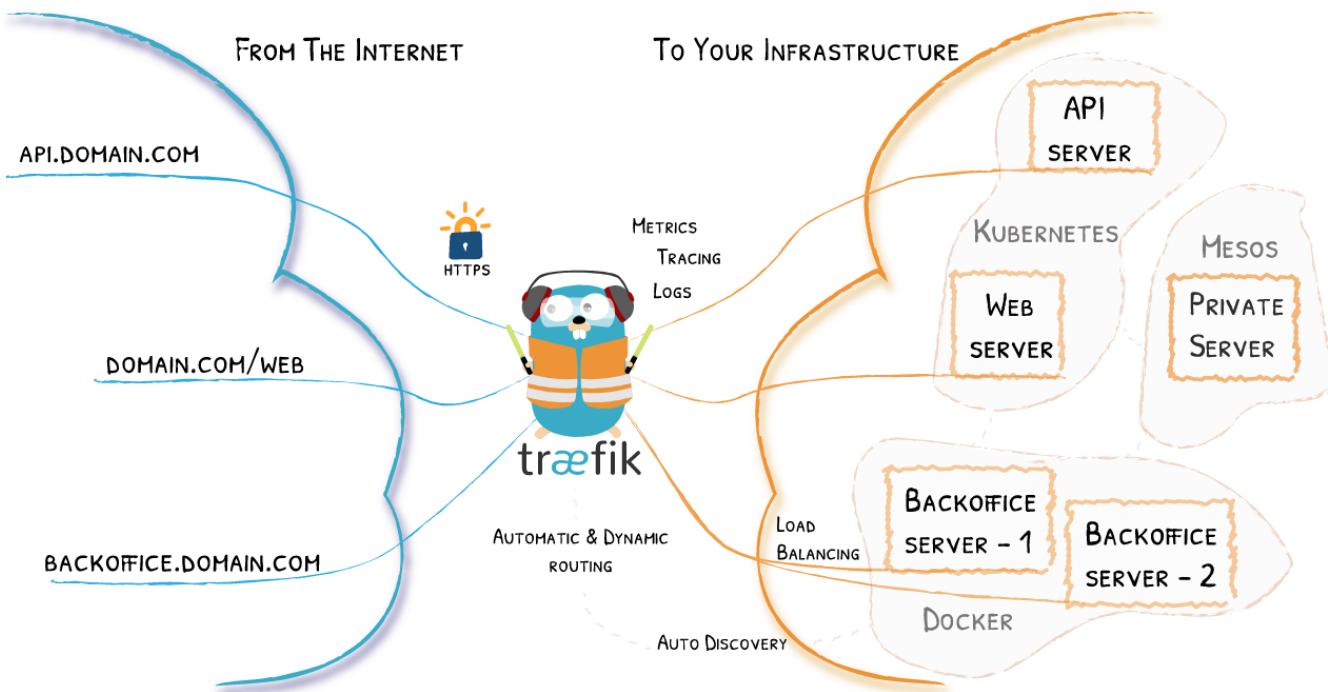


Figura 12.4: Traefik



Figura 12.5: Rocket.Chat



Figura 12.6: MongoDB

## Instalando o Rocket.Chat

Iremos executar o Rocket.Chat através do docker-compose na máquina container.

Conecte-se a máquina container através do vagrant ssh e troque para o usuário root

```
1 vagrant ssh container
2 sudo su -
```

Crie o diretório para o compose do Rocket.Chat e acesse a mesma

```
1 mkdir /root/rocketchat
2 cd /root/rocketchat
```

Crie o arquivo docker-compose.yml e adicione o conteúdo

```
1 vim docker-compose.yml
```

```
1 version: '3'
2
3 services:
4   rocketchat:
5     image: rocketchat/rocket.chat:latest
6     restart: unless-stopped
7     environment:
8       - PORT=3000
9       - ROOT_URL=http://localhost:3000
10      - MONGO_URL=mongodb://mongo:27017/rocketchat
11      - MONGO_OPLOG_URL=mongodb://mongo:27017/local
12     depends_on:
13       - mongo
14     labels:
15       - "traefik.backend=rocketchat"
16       - "traefik.frontend.rule=Host: chat.4labs.example"
17
18 mongo:
19   image: mongo:4.0
20   restart: unless-stopped
21   volumes:
22     - $PWD/db:/data/db
23   command: mongod --smallfiles --oplogSize 128 --replicaSet rs0 --storageEngine=mmapv1
24   labels:
25     - "traefik.enable=false"
26
27 mongo-init-replica:
28   image: mongo:4.0
29   command: >
30     bash -c
31     "for i in `seq 1 30`; do
32       mongo mongo/rocketchat --eval \
33         rs.initiate({
34           _id: 'rs0',
35           members: [ { _id: 0, host: 'localhost:27017' } ]})\" &&
36       s=$$? && break || s=$$?;
37       echo \"Tried $$i times. Waiting 5 secs...\";
38       sleep 5;
39     done; (exit $$s)"
40   depends_on:
41     - mongo
42
43 traefik:
44   image: traefik:1.7
45   restart: unless-stopped
46   command: traefik /
```

```

47     --docker /
48     --api /
49     --docker.domain=4labs.example /
50     --entrypoints="Name:http Address::80" /
51     --entryPoints="Name:https Address::443" /
52     --defaultentrypoints="http,https"
53   ports:
54     - 80:80
55     - 443:443
56     - 8080:8080
57   volumes:
58     - /var/run/docker.sock:/var/run/docker.sock

```

Inicie a criação do ambiente e acompanhe através dos logs

```

1 docker-compose up -d
2 docker-compose logs -f

```

Nosso servidor estará disponível quando for exibida a mensagem **SERVER RUNNING** nos logs

```

1 rocketchat_1      | System    startup
2 rocketchat_1      | +-----+
3 rocketchat_1      | |           SERVER RUNNING
4 rocketchat_1      | +-----+
5 rocketchat_1      |
6 rocketchat_1      | |   Rocket.Chat Version: 2.2.0
7 rocketchat_1      | |   NodeJS Version: 8.15.1 - x64
8 rocketchat_1      | |   MongoDB Version: 4.0.13
9 rocketchat_1      | |   MongoDB Engine: mmapv1
10 rocketchat_1     | |   Platform: linux
11 rocketchat_1     | |   Process Port: 3000
12 rocketchat_1     | |   Site URL: http://localhost:3000
13 rocketchat_1     | |   ReplicaSet OpLog: Enabled
14 rocketchat_1     | |   Commit Hash: 9410bbc53c
15 rocketchat_1     | |   Commit Branch: HEAD
16 rocketchat_1     |
17 rocketchat_1     | +-----+

```

Podemos visualizar através do endereço <http://chat.4labs.example:8080> o dashboard do traefik e verificar que a rota para nosso rocketchat foi criado

## Configurando o Rocket.Chat

Vamos acessar o endereço <http://chat.4labs.example> e configurar nosso Rocket.Chat

Crie o usuário administrador **analista** com a senha **devops@4linux**

Preencha os dados da organização e clique em continue

Preencha os dados do Servidor e clique em Continue.

Nesta tela selecione a opção **Keep Standalone** uma vez que nosso e-mail não é um endereço válido. Você pode utilizar a opção register to access para receber o mailing da Rocket.Chat e o acesso a chaves para funcionalidades extras.

Terminada a configuração clique em **Go to your workspace** para acessar o Rocket.Chat

Na tela inicial do RocketChat vamos clicar no menu e em seguida em Administration

Na tela inicial de configurações podemos ver dados importantes e parametrizar nosso Rocket.Chat

Em **General** altere o **Site URL** para <http://chat.4labs.example> e clique em **Save Changes**

The screenshot shows the Traefik dashboard interface. At the top, there's a header with the Traefik logo, a search bar containing 'chat.4labs.example:8080/dashboard/', and navigation icons. Below the header, the page title is 'PROVIDERS HEALTH' and the version is 'V1.7.14 / MAROILLES DOCUMENTATION'. A search bar with the placeholder 'Filter by name or id ...' is present. The main content is divided into two sections: 'FRONTENDS' and 'BACKENDS'. The 'FRONTENDS' section contains a card for 'frontend-Host-chat-4labs-example-0' with a 'Route Rule' (Host: chat.4labs.example), 'Entry Points' (http, https), and a 'Backend' (backend-rocketchat). The 'BACKENDS' section contains cards for 'backend-rocketchat' (Main, Server: http://172.22.0.4:3000, Weight: 1) and 'backend-traefik-rocketchat' (Main, Server: http://172.22.0.4:3000, Weight: 1). Both the 'FRONTENDS' and 'BACKENDS' sections are highlighted with a red border.

Figura 12.7: Traefik Dashboard

The screenshot shows the Rocket.Chat setup wizard. The browser title is 'Rocket.Chat' and the URL is 'chat.4labs.example/home'. The page has a header with the Rocket.Chat logo and a 'SETUP WIZARD' button. The main content is titled 'Setup Wizard' with the sub-section 'STEP 1 Admin Info'. It includes fields for 'NAME' (placeholder: Type your name), 'USERNAME' (placeholder: Type your username), 'ORGANIZATION EMAIL' (placeholder: Type your email), and 'PASSWORD' (placeholder: Type your password). On the left, a vertical navigation bar shows steps: 1 Admin Info (highlighted with a blue circle), 2 Organization Info, 3 Server Info, and 4 Register Server.

Figura 12.8: Configuração Rocketchat

STEP 1

## Admin Info

Name

Analista DevOps



Username

analista

@

Organization Email

analista@4labs.example



Password

devops@4linux



Continue

Figura 12.9: Configuração Rocketchat

STEP 2

## Organization Info

Organization Type

Community ▾

Organization Name

4Labs

Industry

Technology Services ▾

Size

11-50 people ▾

Country

Brazil ▾

Website

<http://4labs.example>

Continue



Figura 12.10: Configuração Rocketchat

STEP 3

## Server Info

Site Name

4Labs - Rocket.Chat

Language

Default

Server Type

Private Team

Back

Continue

Figura 12.11: Configuração Rocketchat

STEP 4

## Register Server

Use the preconfigured gateways and proxies provided by Rocket.Chat Technologies Corp.

Register to access

Mobile push notifications gateway

Livechat omnichannel proxy

OAuth proxy for social network

Apps Marketplace

Newsletter, offers and product updates

**●** Keep standalone, you'll need to

- Create accounts with service providers
- Update the preconfigured settings
- Recompile the mobile apps with your own certificates

Back

Continue



Figura 12.12: Configuração Rocketchat

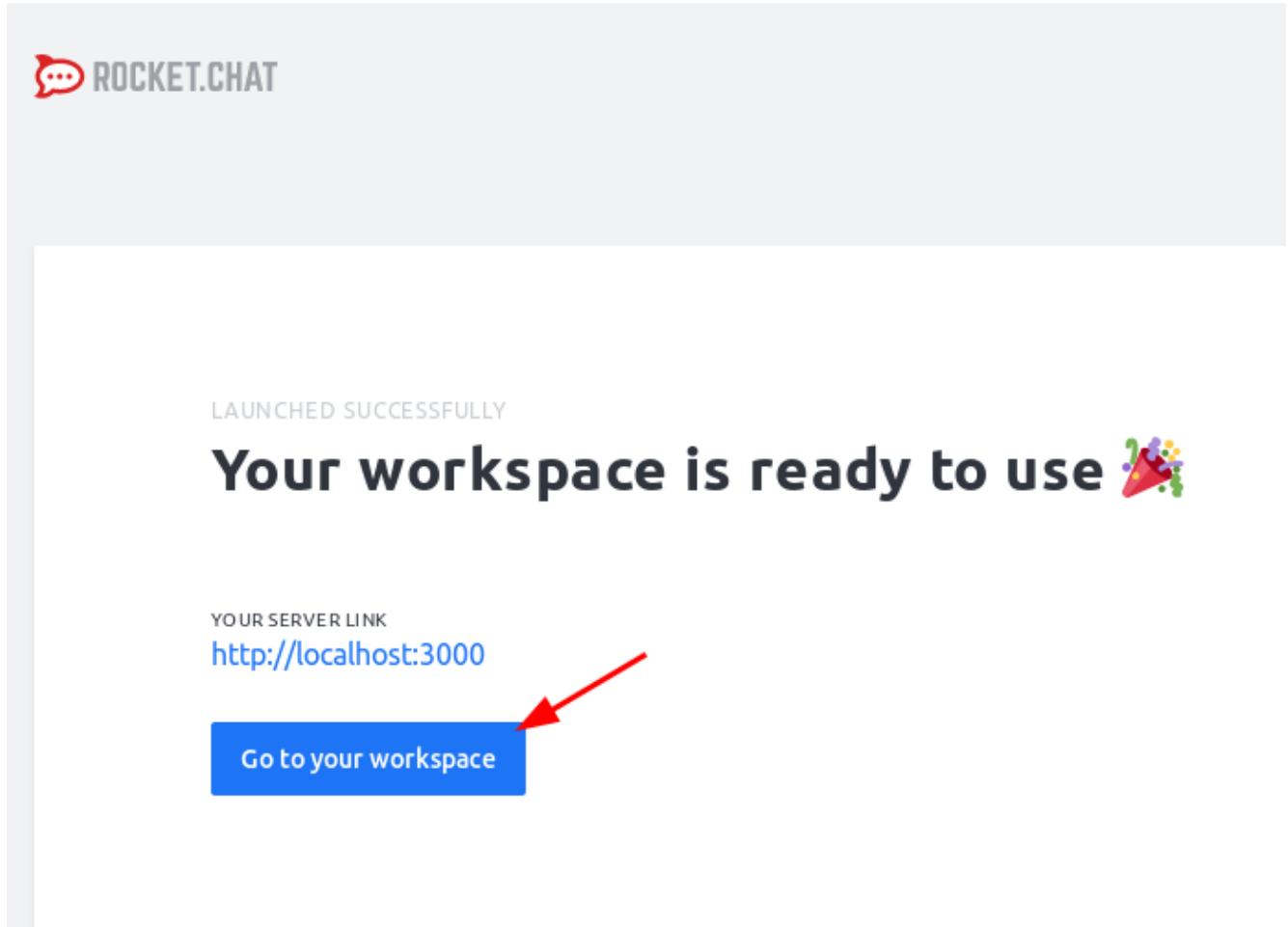


Figura 12.13: Configuração Rocketchat

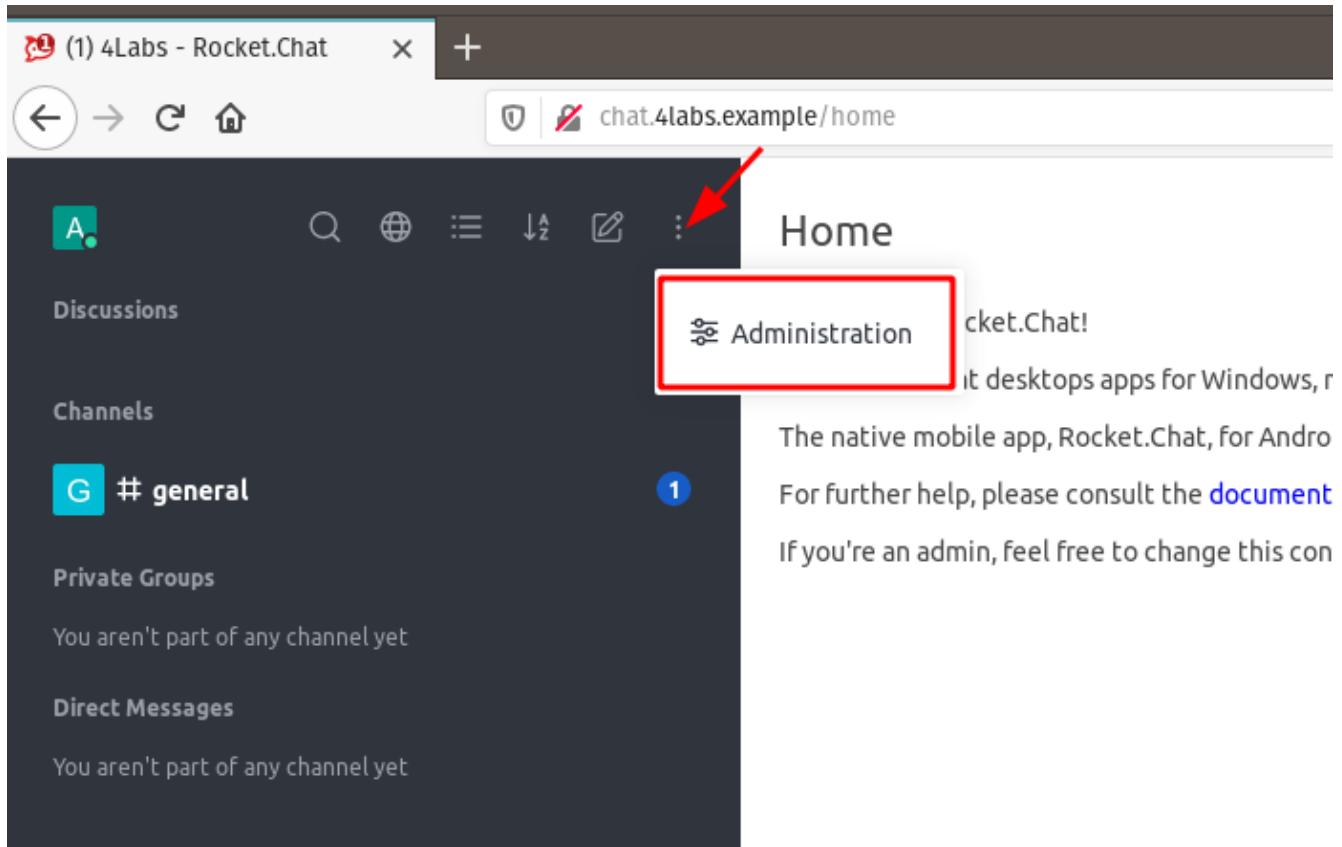


Figura 12.14: Configuração Rocketchat

A screenshot of the Rocket.Chat Administration panel. On the left is a sidebar with links: Info, Import, Rooms, Users, Connectivity Services, Custom Sounds, Custom Emoji, Federation Dashboard, Integrations, OAuth Apps, Custom User Status, Mailer, Permissions, View Logs, Apps, Marketplace, and Settings. The "Info" link is selected. The main content area is titled "Info" and shows "Rocket.Chat" details. It includes a table with rows for Version (2.2.0), Apps Engine Version (1.7.0), Database Migration (166), and Database Migration Date (November 16, 2019 7:44 PM). Below this is another table with rows for Installed at (November 16, 2019 7:44 PM), Uptime (40 seconds), Deployment ID (gYMMGfGv3T92Dv42P), PID (1), Running Instances (1), and OpLog (Enabled). At the bottom, there's a "Commit" section with a table showing Hash (9410bbc53cdcede4fa32b416718da71a15097dd01), Date (Sun Oct 27 17:49:02 2019 -0300), Branch (HEAD), Tag (2.2.0), Author (Rodrigo Nascimento), and Subject (Merge pull request #15681 from RocketChat/release-2.2.0).

Figura 12.15: Configuração Rocketchat

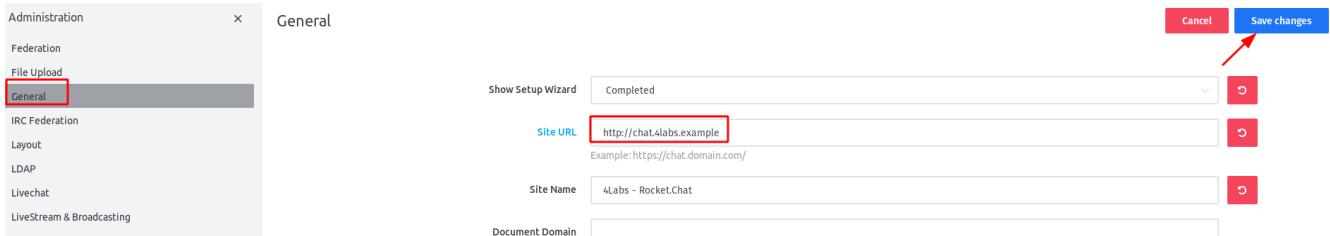


Figura 12.16: Novo Usuário

Administration

Users

SHOWING 2 RESULTS

Name	Username	Email	Roles	Status
A Analista...	analista	analista@4l...	admin	online
Rocket....	rocket.cat		bot	online

**+**

Figura 12.17: Novo Usuário

## Criando Usuários

Para criar um novo usuário clique em **Users** e em seguida no ícone +

Preencha os campos, adicione o usuário a role **user** e clique em **Save**

Um novo usuário será exibido na tela de usuários

Feche a tela de administração

Abra o canal **#general** e verifique que o diretor foi adicionado ao mesmo.

## Criando Canais

Para criar uma nova sala clique no ícone **Create New** e em seguida em **Channel**

Preencha os dados do Canal e clique em **Create**

O novo canal será criado com o membro criador e os membros adicionados

## Integrações

### O que é um Webhook?

**Webhook** é uma forma de recebimento de informações quando um evento acontece. Podemos dizer que o webhook é a forma de receber informações entre dois sistemas de uma forma passiva. Serão utilizados como gatilho para disparo de funcionalidades, tarefas ou ações após um sistema comunicar-se com um segundo sistema.

Também é conhecido como **gancho-web** ou **retorno de chamada HTTP**

### Criando o Webhook Gogs

Os projetos no Gogs podem ser integrados a qualquer outro sistema que aceite webhooks, com este recursos podemos enviar uma notificação via **HTTP POST** caso alguma mudança aconteça no Gogs.

Os webhooks são acionados por gatilhos (**triggers**). No *gogs* existem os seguintes triggers disponíveis: \* Create -> Criação de Branch ou TAG \* Delete -> Remoção de Branch ou TAG \* Fork -> Fork de Repositório \* Push -> Push do Repositório \* Issues -> Issues abertas, fechadas, reabertas, editadas, atribuídas, etc.. \* Pull Request -> Abertura, fechamento, reabertura, etc... \* Issue Comment -> Criação, edição ou remoção de comentário em issues \* Release -> Release publicada no repositório

Dessa forma, vamos integra-lo com o RocketChat:

Primeiramente vamos acessar o RocketChat no endereço <http://chat.4labs.example>, efetuar o login

Crie um novo canal público com o nome de Gogs, somente leitura e adicione o usuário **rocket.cat**

Clique para editar o canal e adicione um Announcement e uma Descrição

Clique para editar os usuários do canal e adicione o **rocket.cat** como moderador para que ele possa postar no canal

Agora que criamos nosso canal e adicionamos o **rocket.cat** como moderador vamos criar nosso recebimento de webhook, para isto acesse o menu de Administração.

Vamos definir nosso webhook no menu **integrations** através do botão **New Integration**

#### Escolha Incoming Webhook

As opções tratam o sentido do fluxo do webhook:

**Incoming Webhook** - Webhooks recebidos pelo Rocket.Chat **Outgoing Webhook** - Webhooks enviados pelo Rocket.Chat

Preencha os dados e clique em **Save Changes**

Copie a URL do Webhook gerado que será adicionada ao gogs

Acesse o gogs no endereço <http://scm.4labs.example> e efetue o login com o usuário **analista** e senha **devops@4linux**

Name

Username

Email

Verified

Status Message

Password

Require password change

Roles  
 user

Add Role

Join default channels

Send welcome email

Figura 12.18: Novo Usuário

Users				
<input type="text"/> Search				
SHOWING 3 RESULTS				
Name	Username	Email	Roles	Status
A Analista DevOps	analista	analista@4labs.example	admin	online
D Diretor DevOps	diretor	diretor@4labs.example	user	offline
Rocket.Cat	rocket.cat		bot	online

Figura 12.19: Novo Usuário

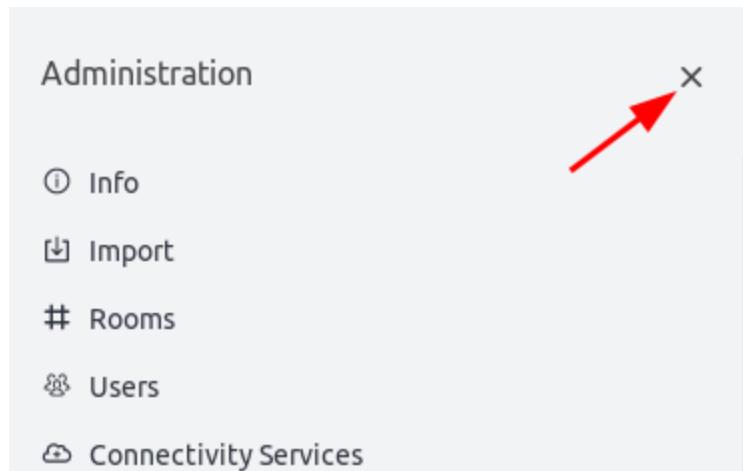


Figura 12.20: Rocketchat

The screenshot shows the RocketChat interface. On the left, there's a sidebar with sections for Discussions, Channels, and Private Groups. The Channels section highlights the '# general' channel, which has a red border around its name. A blue notification badge with the number '1' is visible above the channel name. On the right, the main chat area shows two messages at the bottom:

- A analista 7:47 PM Has joined the channel.
- D diretor 8:08 PM Has joined the channel.

Both messages are also highlighted with a red border.

Figura 12.21: Rocketchat

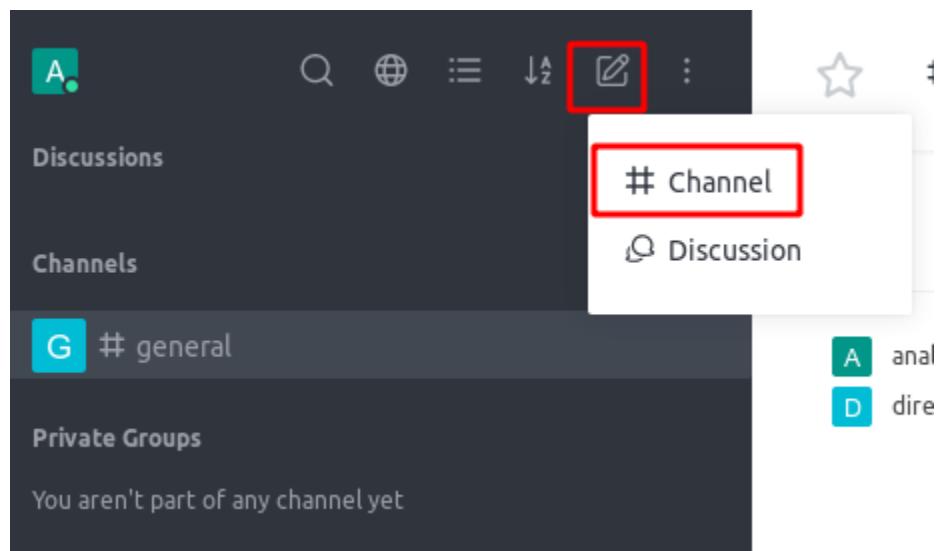


Figura 12.22: Novo Canal

## Create a New Channel

Channels are where your team communicate

Public Channel  
Everyone can access this channel

Read Only Channel  
All users in the channel can write new messages

Broadcast Channel  
Only authorized users can write new messages, b

Channel Name

# devops

Invite Users

@  Please enter usernames

**Create**

Figura 12.23: Novo Canal

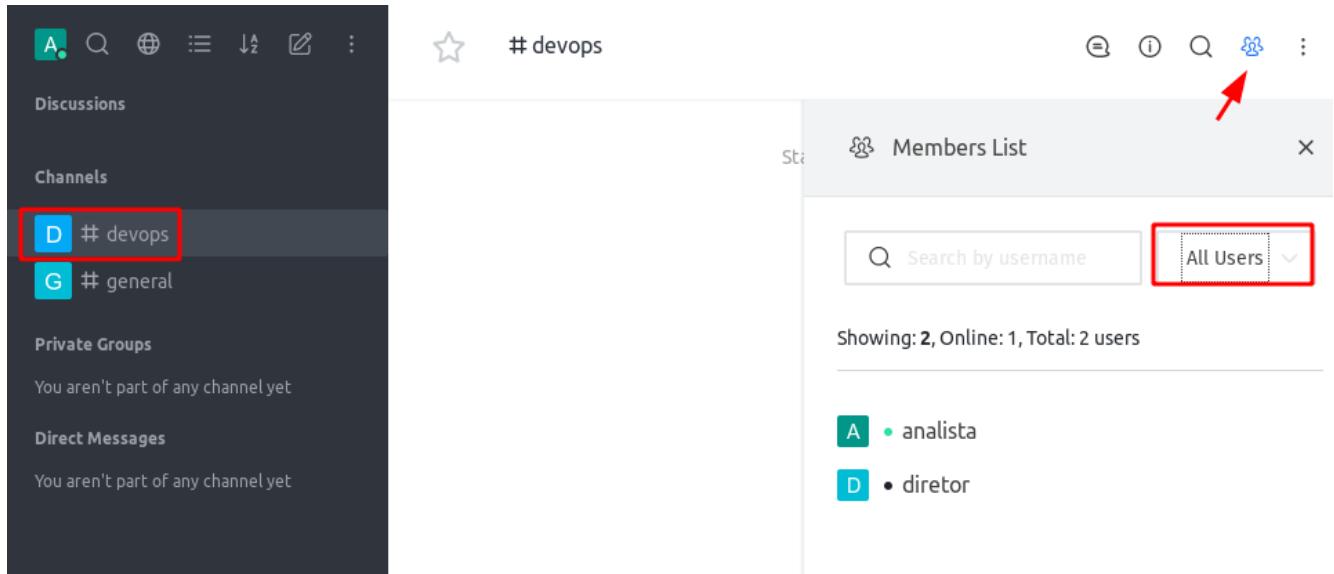


Figura 12.24: Novo Canal

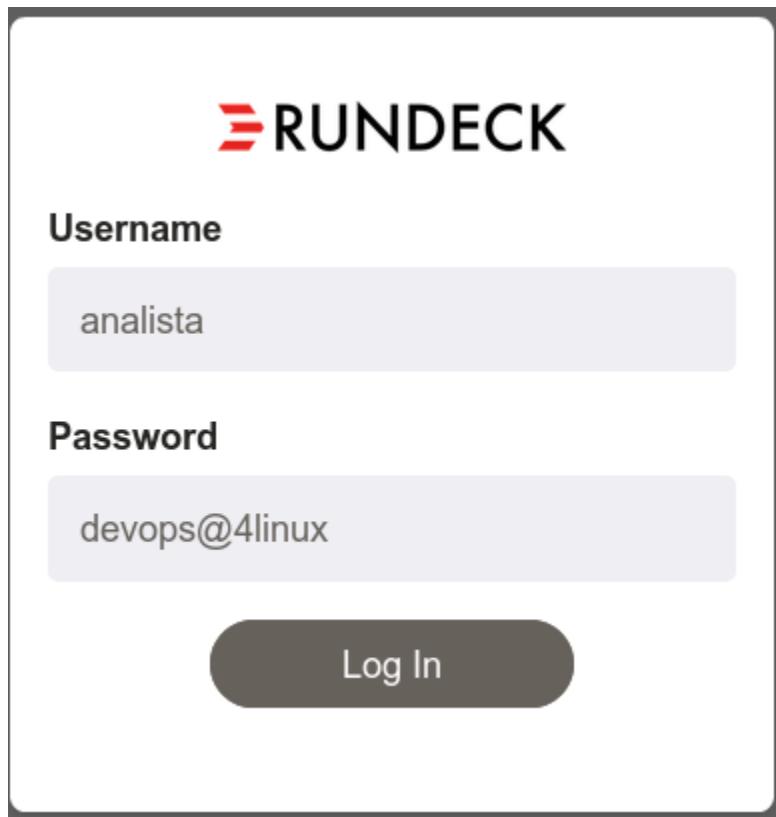


Figura 12.25: Login

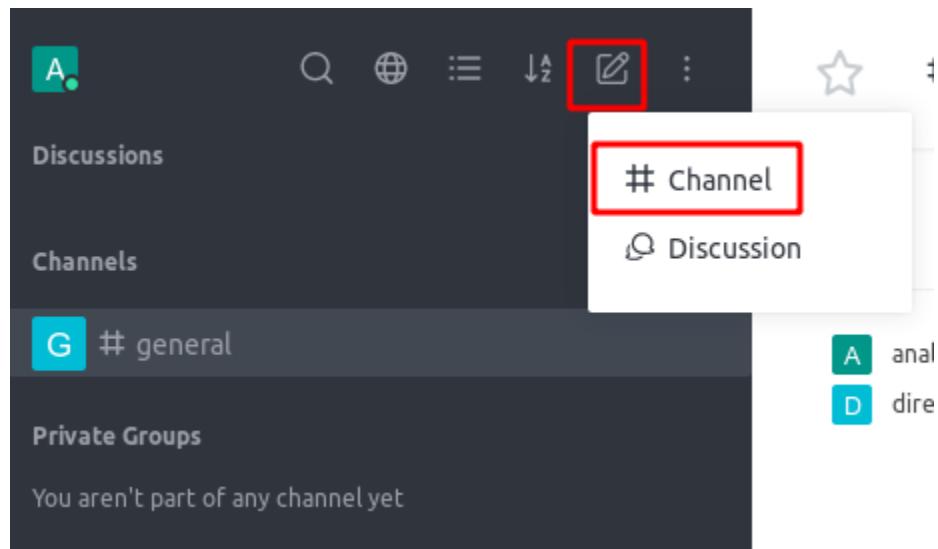


Figura 12.26: Novo Canal

Acesse o repositório **inspec**

Clique em **Settings**, **Webhooks**, **Add Webhook** e selecione **Slack**

O *Webhook do Rocketchat* possui a mesma estrutura do *Webhook do Slack*, se não houvesse uma opção ideal, deveríamos criar um webhook do tipo *Gogs* e trata-lo através de *Scripts*.

Cole a URL copiada do *Rocket.Chat*, preencha o restante dos dados e clique em **Add Webhook**

Repita o processo e adicione uma nova webhook, desta vez escolhendo a opção **Let me choose what i need**, **Issues**, **Issue Comment** e alterando a cor para **warning**

Será exibida a tela com os webhooks criados, podemos então testar o webhook clicando no link e então selecionar **Test Delivery**, faça o procedimento para o primeiro webhook

New webhook has been added.

## Webhooks

Add Webhook

Webhooks are much like basic HTTP POST event triggers. Whenever something occurs in Gogs, we will handle the notification to the target host you specify. Learn more in this [Webhooks Guide](#).

- <http://chat.4labs.example/hooks/WNK5gBvot2EX8cQQ9/NhJrzNgRkP8fAdkF4cZYLYzGc9jf7WtCaadDjXryf4D2Jia>
- <http://chat.4labs.example/hooks/WNK5gBvot2EX8cQQ9/NhJrzNgRkP8fAdkF4cZYLYzGc9jf7WtCaadDjXryf4D2Jia>

**Test Delivery**

## Create a New Channel

Channels are where your team communicate

Public Channel

Everyone can access this channel

Read Only Channel

Only authorized users can write new messages

Broadcast Channel

Only authorized users can write new messages, but the channel is broadcasted to everyone

Channel Name

# gogs

Invite Users

@  rocket.cat ×

Please enter usernames...

Create

Figura 12.27: Novo Canal

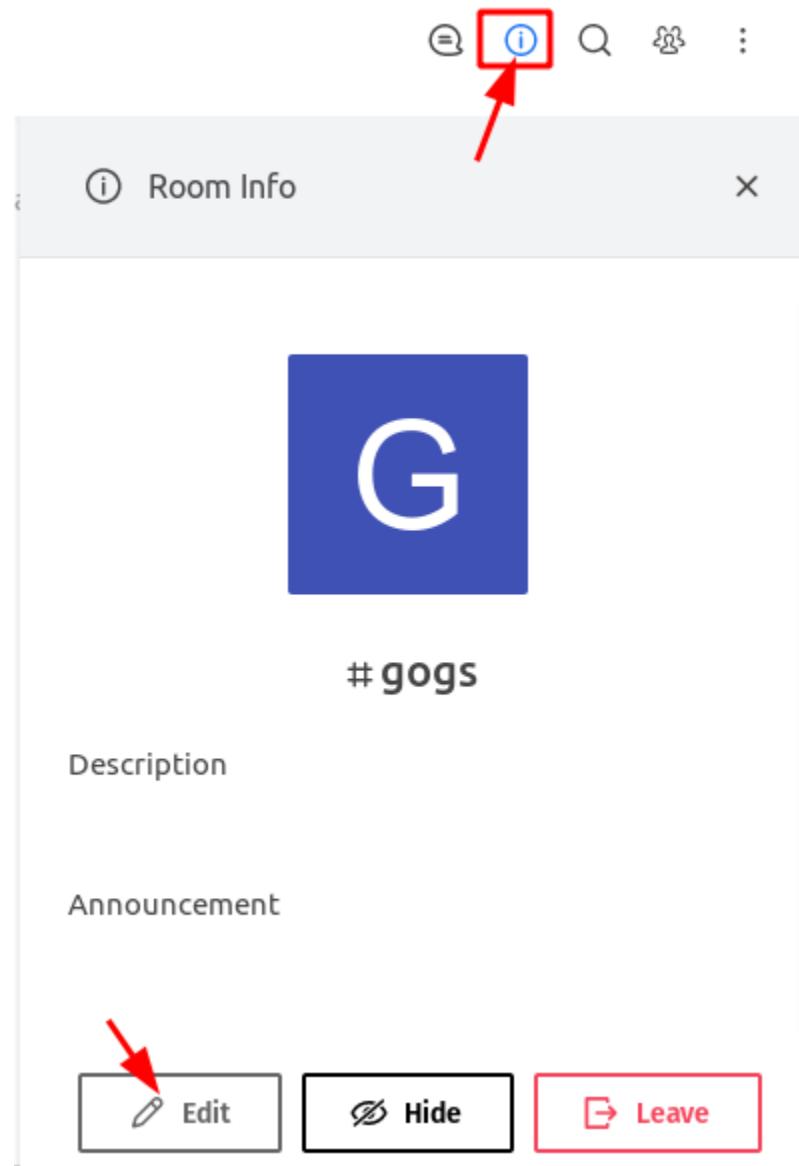


Figura 12.28: Novo Canal

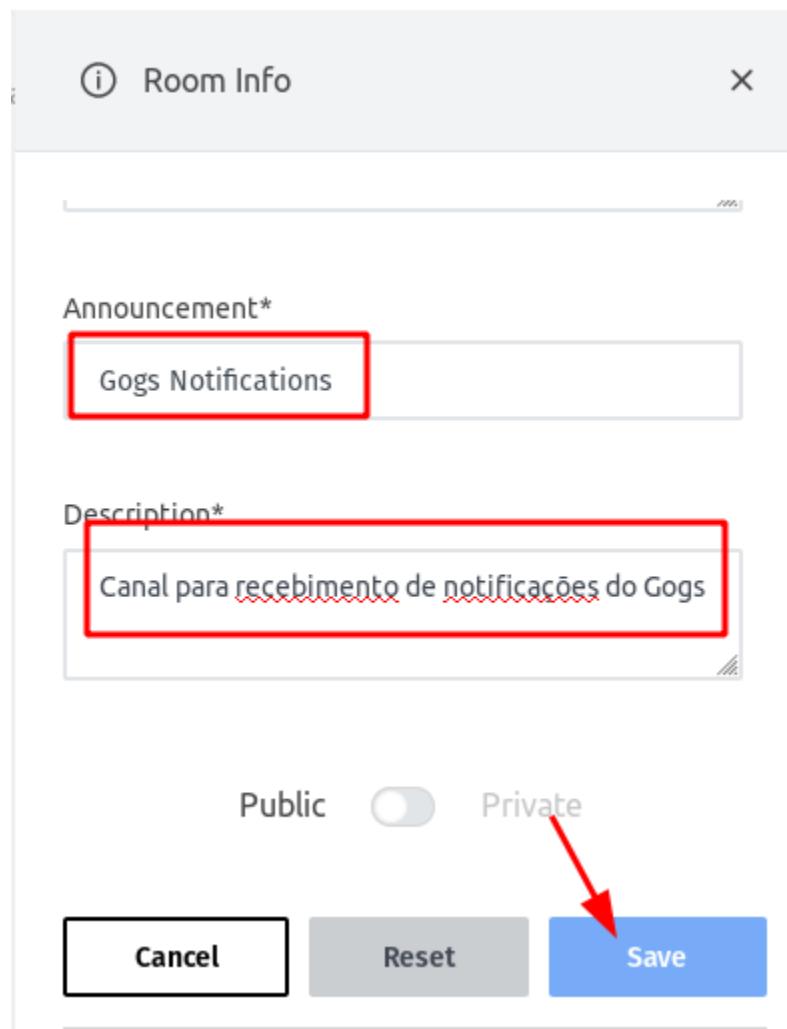


Figura 12.29: Novo Canal

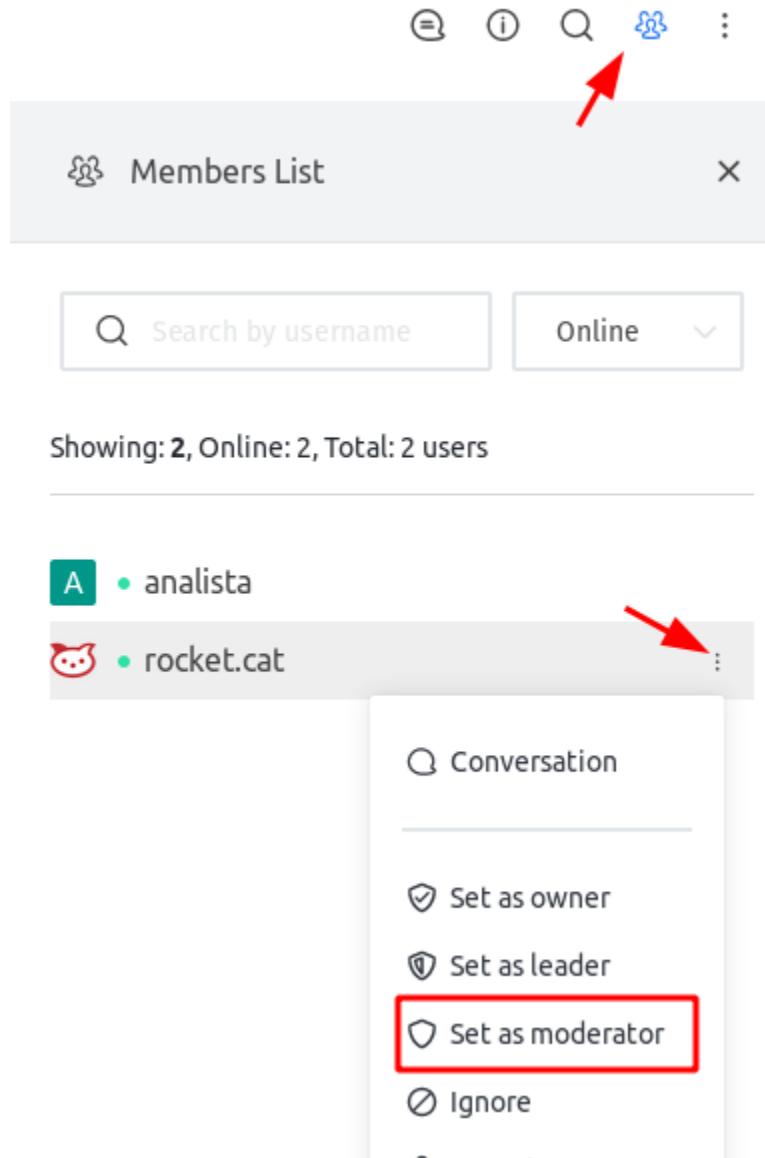


Figura 12.30: Novo Canal

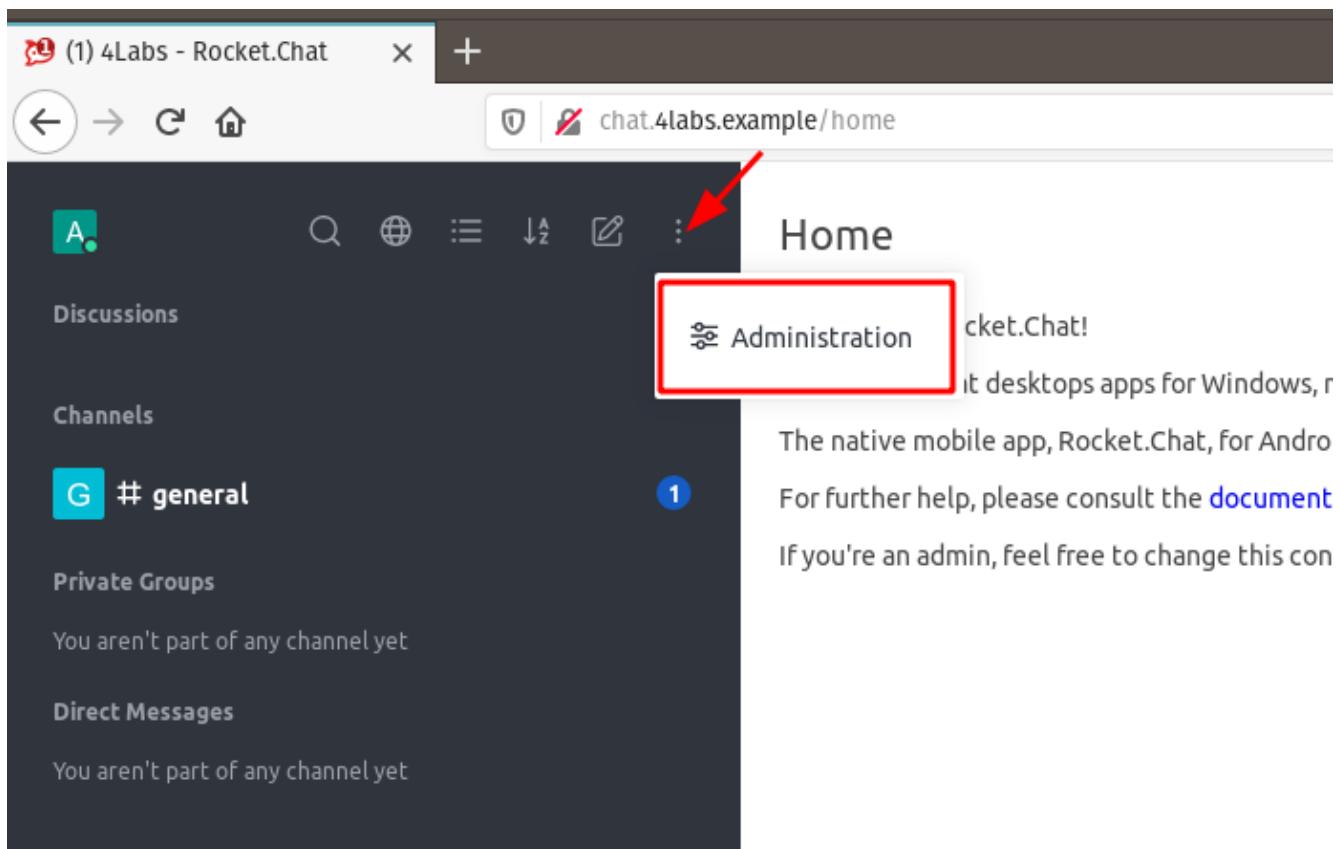


Figura 12.31: Administração Rocketchat

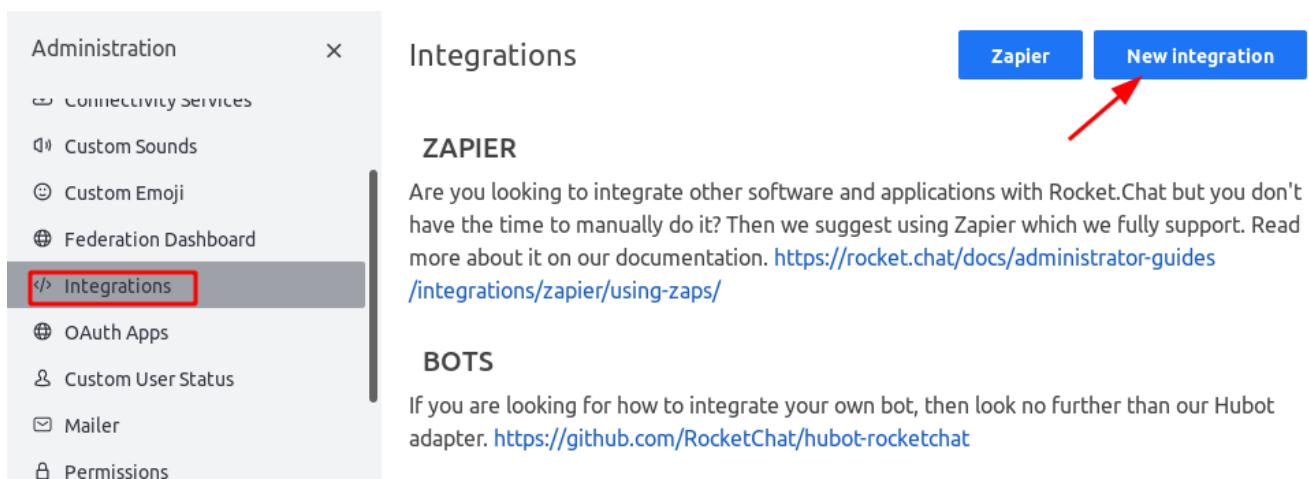


Figura 12.32: New Integration

## New Integration

< Back to integrations

The screenshot shows the 'New Integration' interface. At the top, there's a back navigation link. Below it, two main integration types are listed: 'Incoming WebHook' and 'Outgoing WebHook'. The 'Incoming WebHook' section is highlighted with a red border. It contains an icon of a right-pointing arrow, the text 'Incoming WebHook', and the sub-instruction 'Send data into Rocket.Chat in real-time.' The 'Outgoing WebHook' section has its own icon of a left-pointing arrow, the text 'Outgoing WebHook', and the sub-instruction 'Get data out of Rocket.Chat in real-time.'

Figura 12.33: Webhook

Após a execução será exibida uma tela com a informação do delivery do webhook

Verifique também no canal **#gogs** do RocketChat se o webhook foi ativado com sucesso

Navegue até a página do projeto **inspec** no gogs e clique em **New File**

Crie um novo arquivo com alguns dados e efetue o Commit

Vamos criar também uma issue, clique em **Issues** e em seguida em **New Issue**

Preencha com alguns dados e clique em **Create Issue**

Visualize os chats no canal gogs com as informações

*É possível navegar entre os commits e projetos clicando nos itens que estão em destaque*

## Criando o Webhook Rundeck

Agora vamos criar nosso webhook para as atividades do Rundeck, crie um canal no Rocket.Chat com as mesmas configurações do canal **#gogs**

Clique para editar o canal e adicione um Announcement e uma Descrição

Clique para editar os usuários do canal e adicione o **rocket.cat** como moderador para que ele possa postar no canal

Agora que criamos nosso canal e adicionamos o usuário rocket.cat vamos criar nosso recebimento de webhook, para isto acesse o menu de administração

Vamos definir nosso webhook no menu **integrations** através do botão **New Integration**

Escolha **Incoming Webhook**

Preencha os dados e clique em **Save Changes**

Copie a URL do Webhook gerado que será adicionado ao Rundeck

Agora é necessário instalar o plugin de webhook do rocketchat para o rundeck. Conecte-se máquina automation e troque para o usuário root

```
1 vagrant ssh automation
2 sudo su -
```

## Incoming WebHook Integration

Delete

Save changes

Enabled

True

False

Name (optional)

Gogs

You should name it to easily manage your integrations.

Post to Channel

#gogs

Messages that are sent to the Incoming WebHook will be posted here.

Start with @ for user or # for channel. Eg: @john or #general

Post as

rocket.cat

Choose the username that this integration will post as.

The user must already exist.

Alias (optional)

Optional

Choose the alias that will appear before the username in messages.

Avatar URL (optional)

<http://scm.4labs.example/img/favicon.png>

You can override the avatar used to post from this

Figura 12.34: Webhook

Webhook URL

<http://chat.4labs.example/hooks/WNK5gBvot2EX8cQQ9/NhJrzNgRkP8fAdkF4cZYLyZGc9jf7WtCaadDjXryf4D2Jia>

Send your JSON payloads to this URL.

COPY TO CLIPBOARD

Token

WNK5gBvot2EX8cQQ9/NhJrzNgRkP8fAdkF4cZYLyZGc9jf7WtCaadDjXryf4D2Jia

COPY TO CI IPBOARD

Figura 12.35: Webhook

The screenshot shows the Gogs 'Sign In' page. A red box highlights the 'Sign In' button in the top right corner. The input fields for 'Username or email' (containing 'analista') and 'Password' (containing 'devops@4linux') are also highlighted with red boxes. Below the inputs is a 'Remember Me' checkbox, which is unchecked. At the bottom left is a 'Sign In' button in a green box, and at the bottom right is a 'Forgot password?' link.

Figura 12.36: Login Gogs

The screenshot shows the 'My Repositories' page with three entries: 'rundeck', 'inpec', and 'primeiro'. Each entry has a lock icon and a star rating of '0'. A red arrow points to the 'inpec' repository.

Figura 12.37: Repository

The screenshot shows the repository settings for 'analista / inspec'. The sidebar on the left has a 'Webhooks' button highlighted with a red box. The main area shows the 'Webhooks' section with a description: 'Webhooks are much like basic HTTP POST event triggers. Whenever something occurs in Gogs, handle the notification to the target host you specify. Learn more in this [Webhooks Guide](#)'. A red arrow points to the 'Gogs' option in the dropdown menu, which also includes 'Slack', 'Discord', and 'Dingtalk'. The top right corner has 'Unwatch' (1), 'Star' (0), 'Fork' (0), and a 'Settings' button highlighted with a red box.

Figura 12.38: Webhook

**Add Webhook**

Add [Slack](#) integration to your repository.

**Payload URL \***

at.4labs.example/hooks/WNK5gBvot2EX8cQQ9/NhJrzNgRkP8fAdkF4cZYZLyZGc9jf7WtCaadDjXryf4D2Jia

**Channel \***

#gogs

**Username**

Gogs

**Icon URL**

http://scm.4labs.example/img/favicon.ico

**Color**

good

**When should this webhook be triggered?**

Just the push event  
 I need **everything**  
 Let me choose what I need

Active

Details regarding the event which triggered the hook will be delivered as well.

**Add Webhook**

Figura 12.39: Webhook

**Payload URL \***

<at.4labs.example/hooks/WNK5gBvot2EX8cQQ9/NhJrzNgRkP8fAdkF4cXYZLyZGc9jf7WtCaadDjXyf4>

**Channel \***

#gogs

**Username**

Gogs

**Icon URL**

<http://scm.4labs.example/img/favicon.ico>

**Color**

warning

**When should this webhook be triggered?**

Just the push event

I need **everything**

Let me choose what I need

Create  
Branch or tag created

Fork  
Repository forked

Issues  
Issue opened, closed, reopened, edited, assigned, unassigned, label updated, label cleared, milestone, or demilestone.

Issue Comment  
Issue comment created, edited, or deleted.

Delete  
Branch or tag deleted

Push  
Git push to a repository

Pull Request  
Pull request opened, closed, reopened, edited, assigned, unassigned, label updated, label cleared, milestone, demilestone, or synchronized.

Release  
Release published in a repository.

Active

Details regarding the event which triggered the hook will be delivered as well.

**Add Webhook**

Figura 12.40: Webhook  
287

The screenshot shows a "Recent Deliveries" section with a single item. A green checkmark icon is followed by the commit hash "b441e9a3-9eaf-439f-8elf-bbf3cdc45e6". To the right of the hash is the timestamp "2019-11-16 23:38:54 GMT". In the top right corner of the delivery card, there is a blue button labeled "Test Delivery".

Figura 12.41: Webhook

The screenshot shows a GitHub notifications page for the "# gogs" repository. At the top, there is a star icon and the repository name "# gogs". Below the repository name, a dark blue header bar contains the text "Gogs Notifications". Underneath the header, the message content is displayed. It starts with "Start of conversation" and the date "November 16, 2019". The message itself is from a user named "Gogs" (@rocket.cat) at 8:40 PM, stating "[inspec:master] 1 new commit pushed by analista" and providing a link to the commit: "7b3c7ae: Commit inicial - Analista DevOps".

Figura 12.42: Webhook

The screenshot shows a GitHub repository page for "analista / inspec". The repository name is displayed prominently at the top left. On the right side of the header, there is a "Unwatch" button. Below the header, there are navigation links for "Files", "Issues 0", "Pull Requests 0", and "Wiki". The main content area displays statistics: "1 Commits" and "1 Branches". A red arrow points to the "New file" button, which is located in a row of buttons along the bottom of the commit list. The commit details show a user named "Analista DevOps" (represented by a purple and white icon) with the commit hash "7b3c7ae377" and the message "Commit inicial".

Figura 12.43: Webhook

The screenshot shows a GitHub interface for creating a new file. At the top, there are navigation links: Files, Issues (0), Pull Requests (0), and Wiki. Below these, a search bar contains the text "inspec teste-webhook" with a placeholder "(i) or cancel". A red box highlights the file name "teste-webhook". The main area shows a "New file" dialog with the text "Arquivo criado unicamente para teste do Webhook". Below this, a "Commit Changes" dialog is open. It contains a text input field with "Webhook Test" (also highlighted by a red box). There is a large text area for an optional extended description. At the bottom, there are two radio buttons: one selected for "Commit directly to the master branch" and one for "Create a new branch for this commit and start a pull request". A red arrow points from the bottom left towards the green "Commit Changes" button. The "Commit Changes" button is green with white text, while the "Cancel" button is red with white text.

Figura 12.44: Webhook  
289

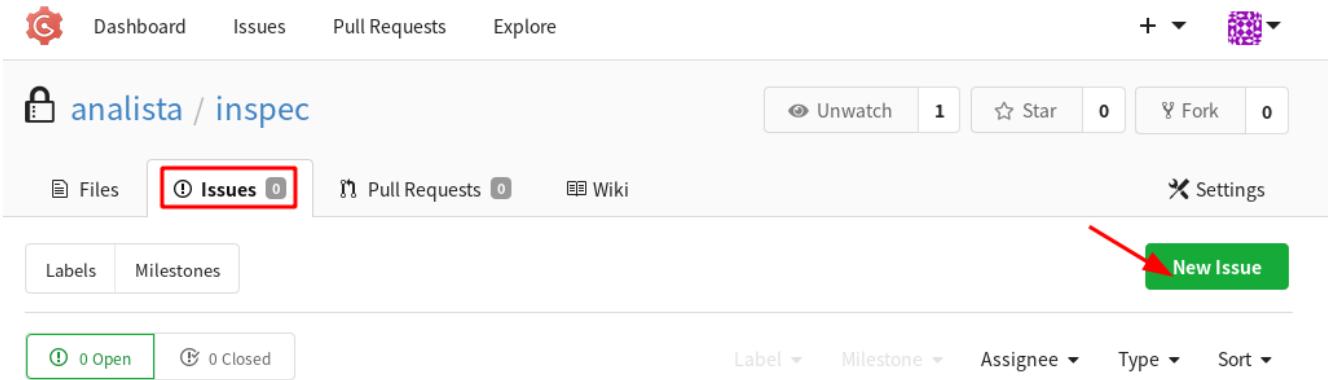


Figura 12.45: Webhook

Acesse o diretório de plugins

```
1 cd /var/lib/rundeck/libext/
```

Copie o arquivo do plugin para a pasta de plugins e altere o dono do arquivo para rundeck

```
1 cp /vagrant/files/rundeck-rocketchat-notifier-0.6.jar .
2 chown rundeck. rundeck-rocketchat-notifier-0.6.jar
```

Reincie o serviço do rundeck

```
1 systemctl restart rundeckd
```

Após a instalação de um plugin normal o sistema levar um tempo para ficar disponível, isto ocorre porque o rundeck faz a descompactação e instalação do plugin bem como a leitura de novos arquivos.

O sistema estará disponível quando a porta **4440** estiver ouvindo conexões

```
1 ss -ntpl | grep 4440
```

Acesse o rundeck em <http://automation.4labs.example:4440> e efetue o login

Verifique se o plugin foi instalado corretamente através do menu **Configuration, Plugins, Installed Plugins**

Abra o projeto 525

Vamos editar o job **Verificando Hosts** para adicionar o webhook

Em **Notifications** habilite a notificação e preencha com a **URL do Webhook** e o canal **#rundeck** para as opções **On Success**, **On Failure**, **On Start**, **Average Duration Exceeded** e **On Retractable Failure** e clique em **Save**

*Em ambientes de produção o ideal é selecionar apenas as informações importantes para ativação do webhook, uma vez que todas as ações geram alarmes, na ocorrência de um alarme de erro o mesmo pode passar despercebido devido a quantidade de mensagens*

Teste o webhook clicando em **Run Job Now**

Verifique na sala **rundeck** no Rocket.Chat o webhook ativado e sua saída

*Na imagem temos exemplos de inicio de job (cor amarela), falha (cor vermelha) e sucesso (cor verde). Esse código de cores nos auxilia na visualização e entendimento dos status do Job*

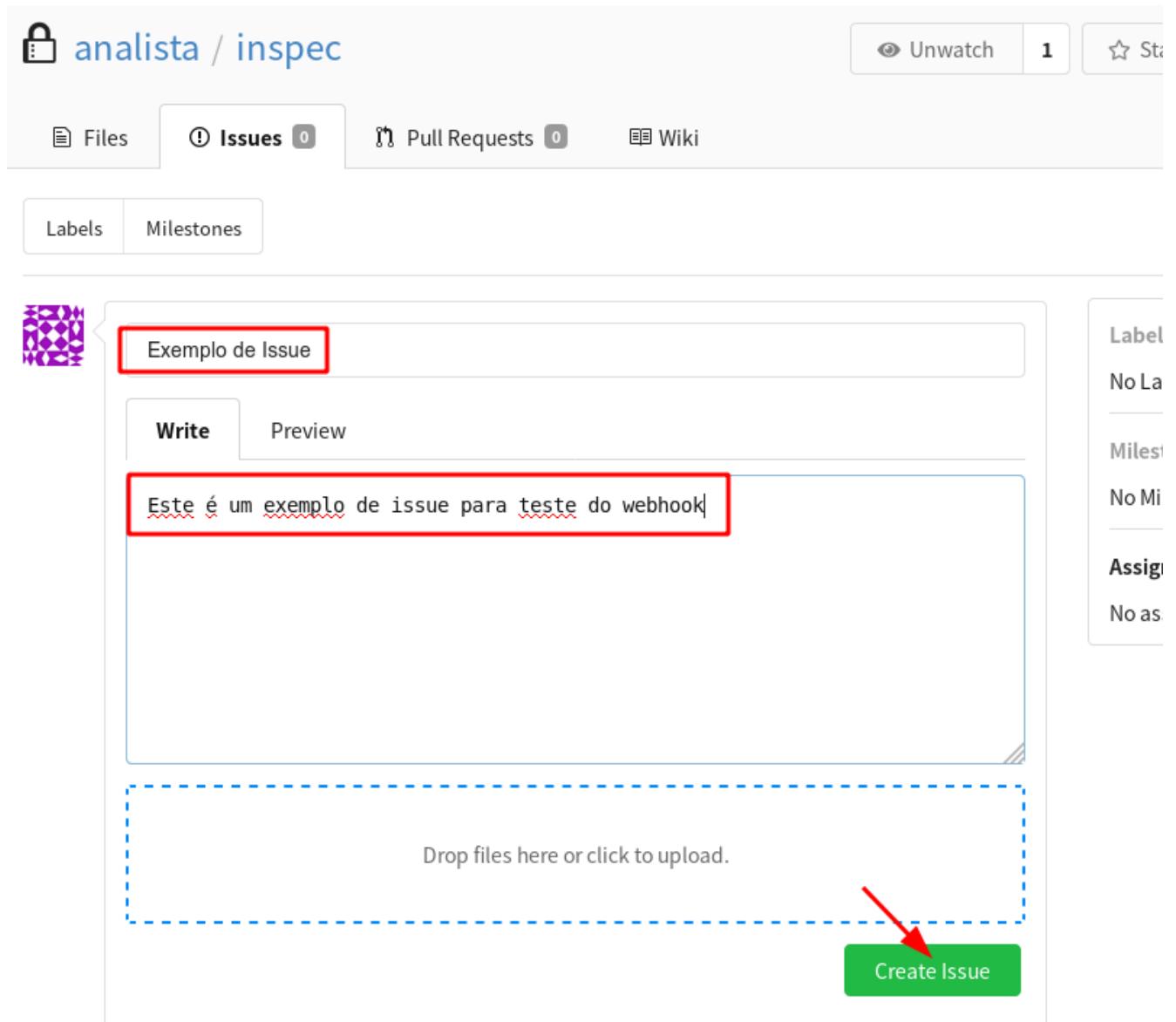


Figura 12.46: Webhook

 Gogs @rocket.cat Moderator Bot 8:43 PM  
[inspec:master] 1 new commit pushed by analista  
| 4766a13: Webhook Test - Analista DevOps

 Gogs @rocket.cat Moderator Bot 8:46 PM  
[analista/inspec] New issue created by analista  
<<http://scm.4labs.example/analista/inspec/issues/2>|#2 Exemplo de Issue> ▾  
Este é um exemplo de issue para teste do webhook

Figura 12.47: Webhook

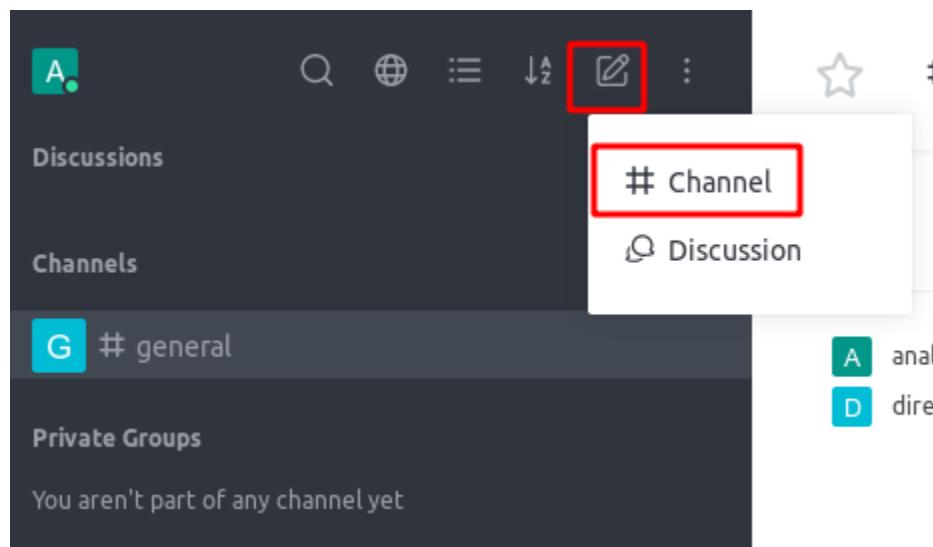


Figura 12.48: Novo Canal

## Create a New Channel

Channels are where your team communicate

Public Channel

Everyone can access this channel

Read Only Channel

Only authorized users can write new messages

Broadcast Channel

Only authorized users can write new messages, but the other users'

Channel Name

# rundeck

Invite Users

@  rocket.cat ×

Please enter usernames...

**Create**

Figura 12.49: Novo Canal

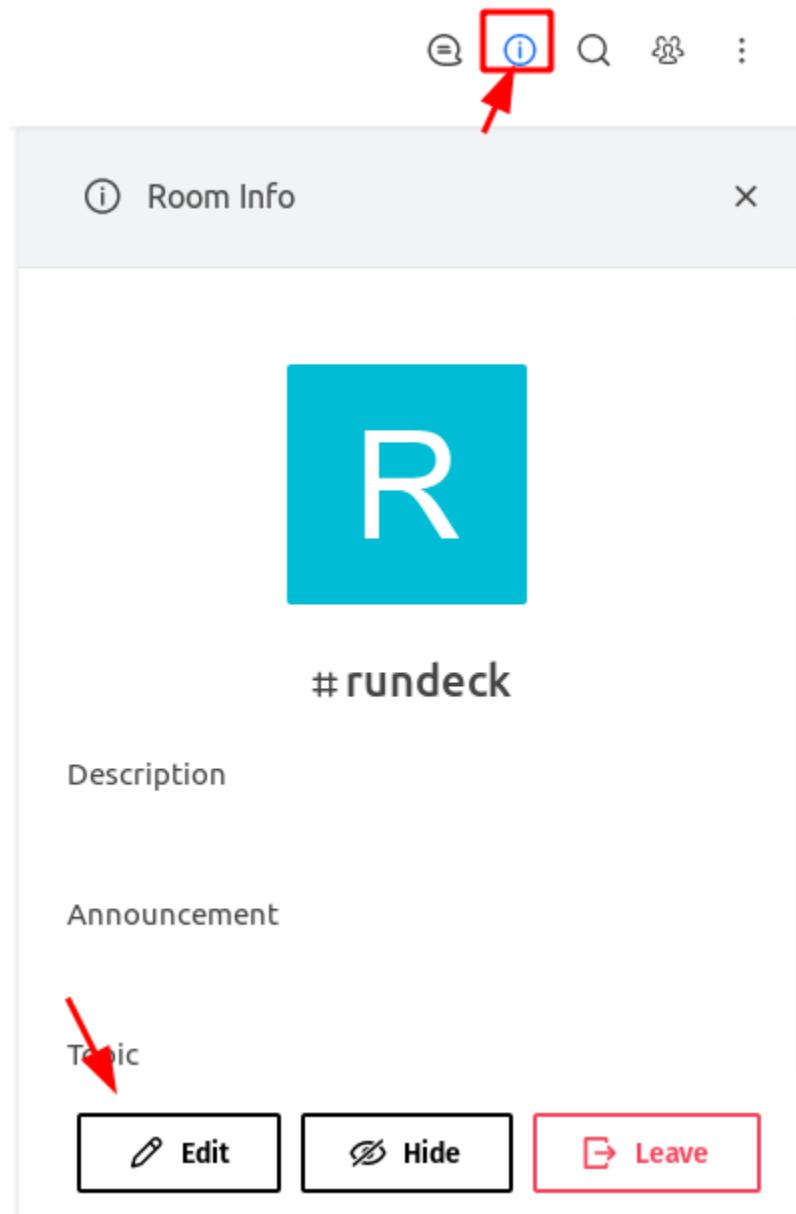


Figura 12.50: Novo Canal

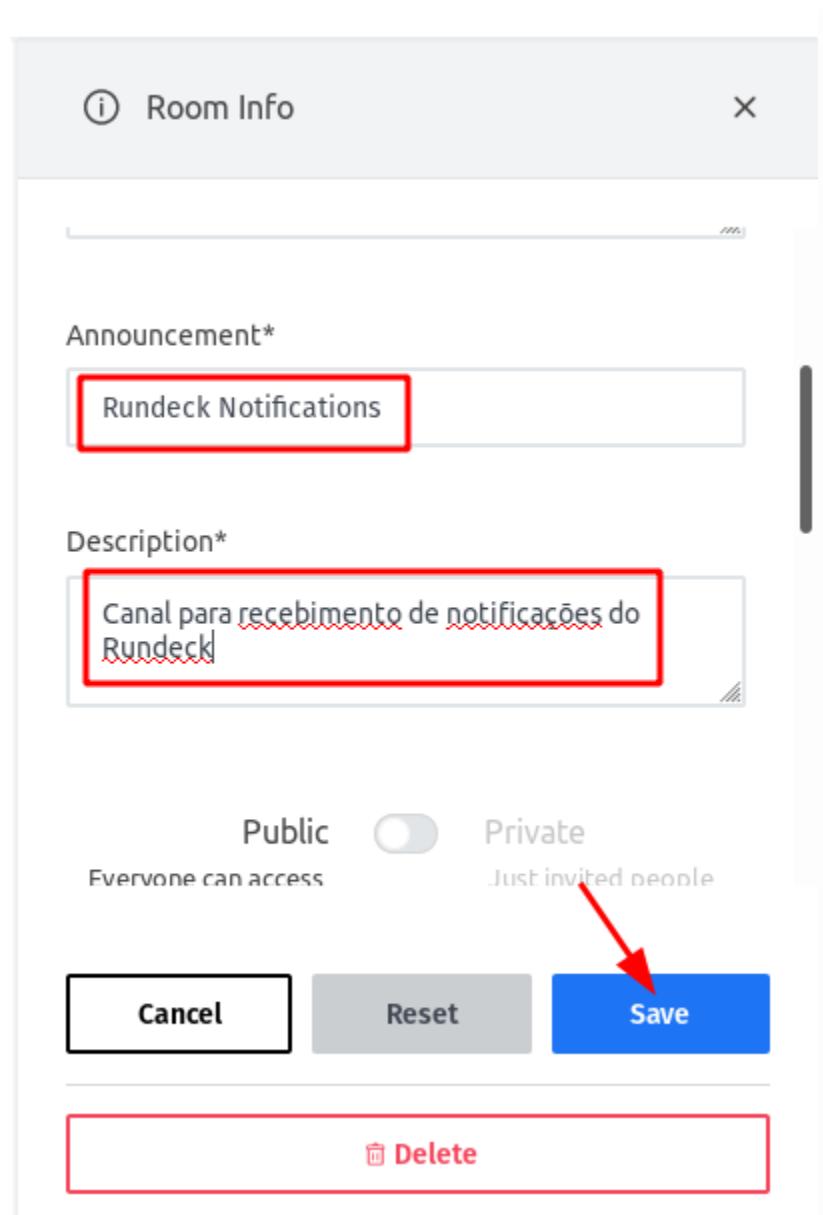


Figura 12.51: Novo Canal

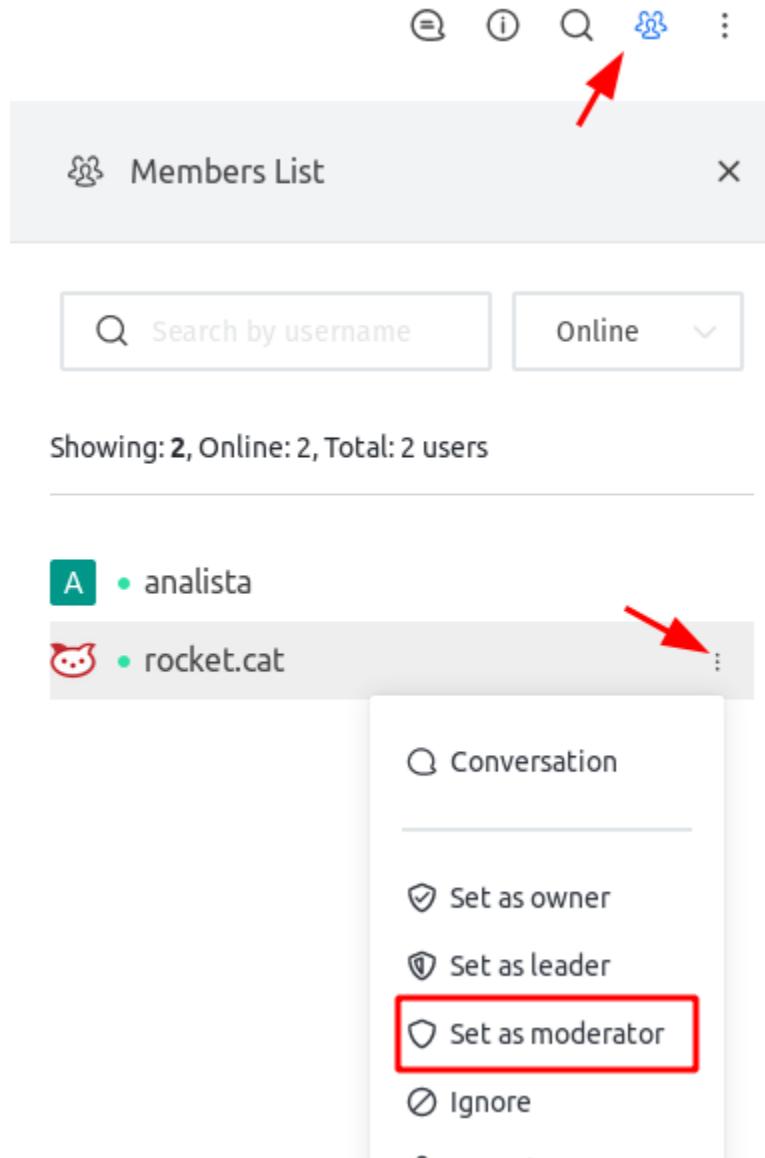


Figura 12.52: Novo Canal

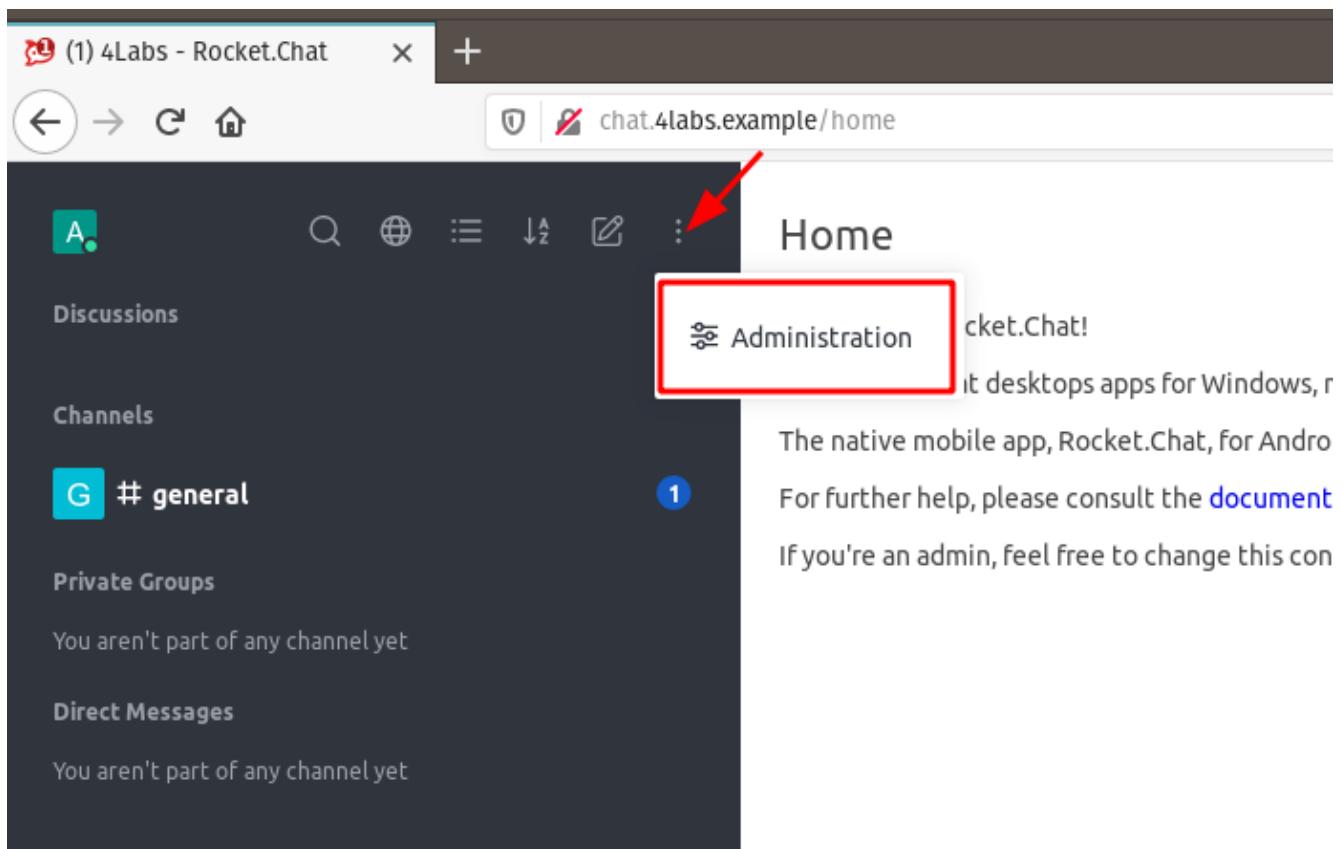


Figura 12.53: Administração Rocketchat

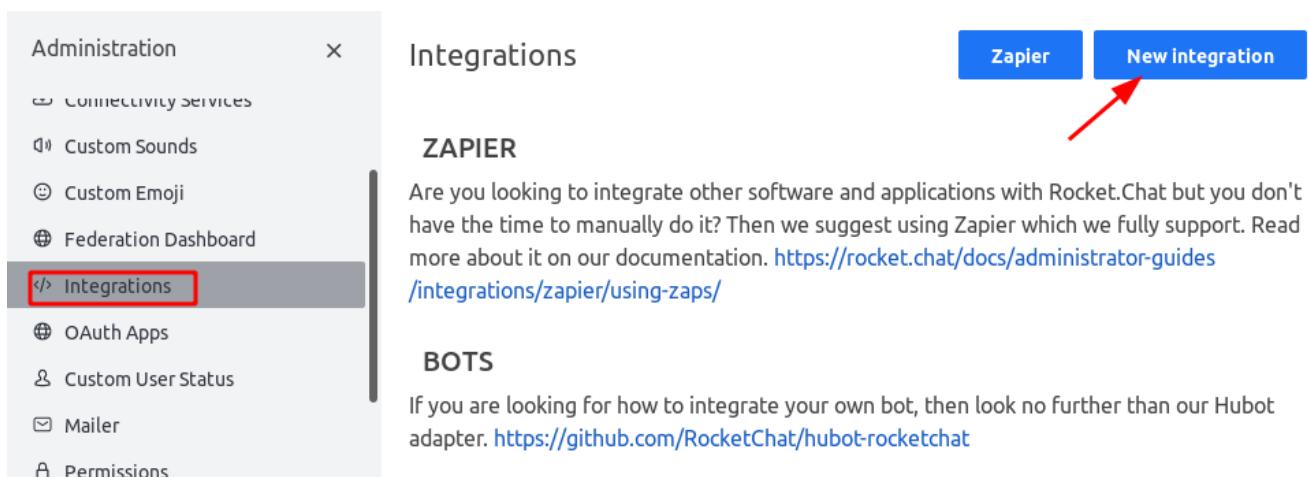


Figura 12.54: New Integration

## New Integration

< Back to integrations

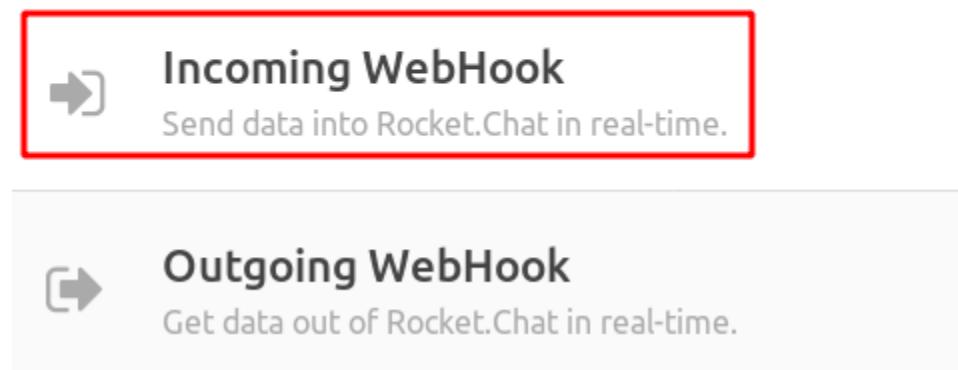


Figura 12.55: Webhook

tion

Enabled  True  False

Name (optional)   
You should name it to easily manage your integrations.

Post to Channel   
Messages that are sent to the Incoming WebHook will be posted here.  
Start with @ for user or # for channel. Eg: @john or #general

Post as   
Choose the username that this integration will post as.  
The user must already exist.

Alias (optional)   
Choose the alias that will appear before the username in messages.

Avatar URL (optional)   
You can override the avatar used to post from this integration.  
Should be a URL of an image.

Emoji (optional)

Figura 12.56: Webhook

Webhook URL <http://chat.4labs.example/hooks/aPcPzBn8GASDeco3F/7xQN6AsnCK4cChADqEEMnRajYnYgPoMdF4ydkfk8pGkx4JbY>

Send your JSON payloads to this URL.

COPY TO CLIPBOARD

Token [aPcPzBn8GASDeco3F/7xQN6AsnCK4cChADqEEMnRajYnYgPoMdF4ydkfk8pGkx4JbY](#)

COPY TO CLIPBOARD

Figura 12.57: Webhook

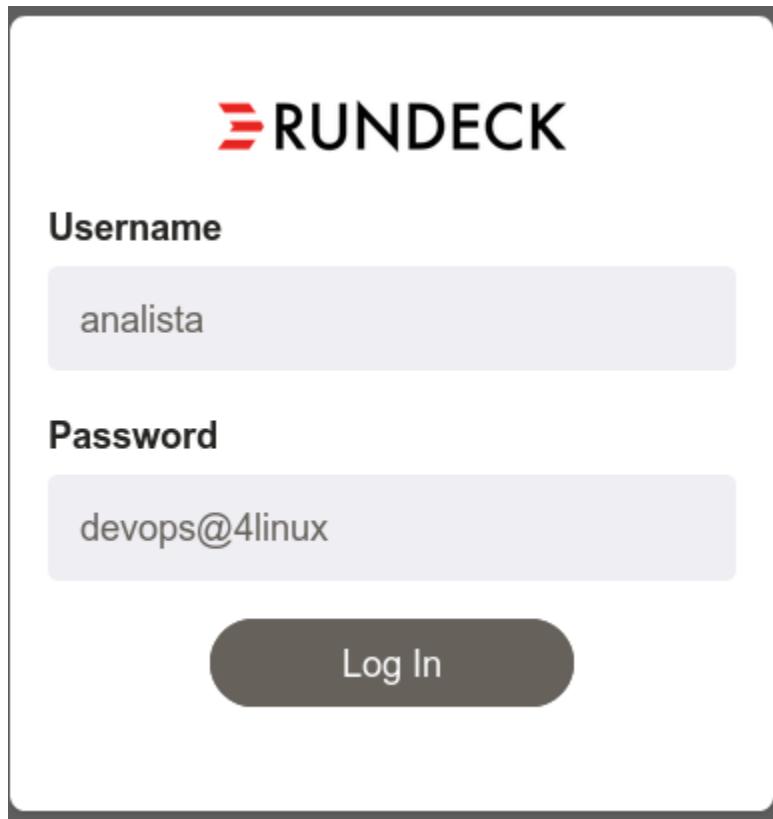


Figura 12.58: Rundeck Login

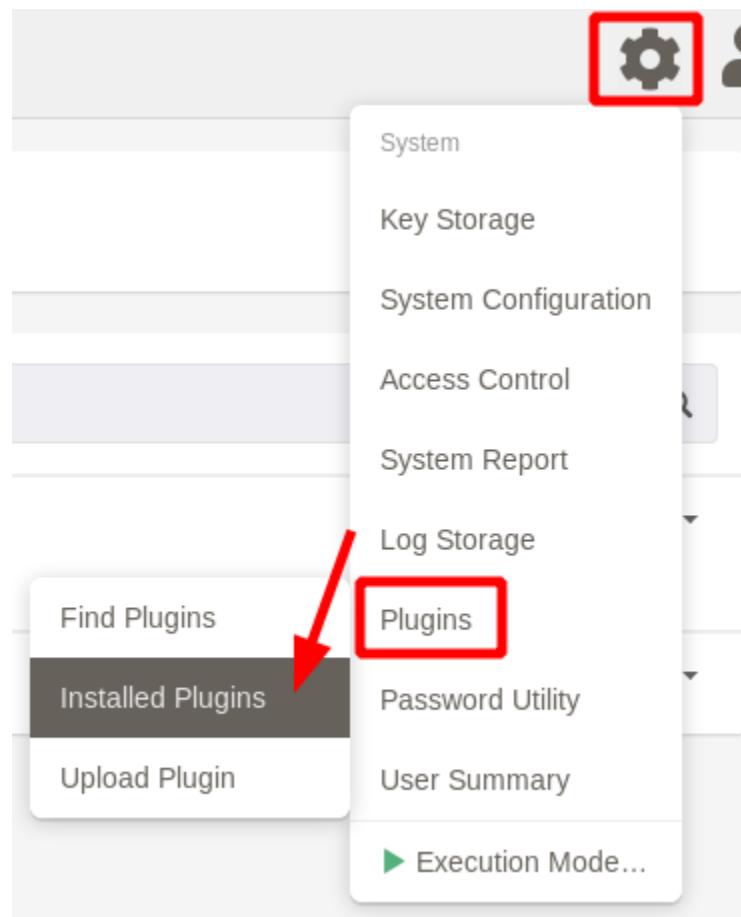


Figura 12.59: Rundeck Plugin

## Installed Plugins

View All

rocketchat

0.6

### Rocket.Chat Notification

Uninstall

Sends a Rundeck Notification to Rocket.Chat

Notification



Figura 12.60: Rundeck Plugin

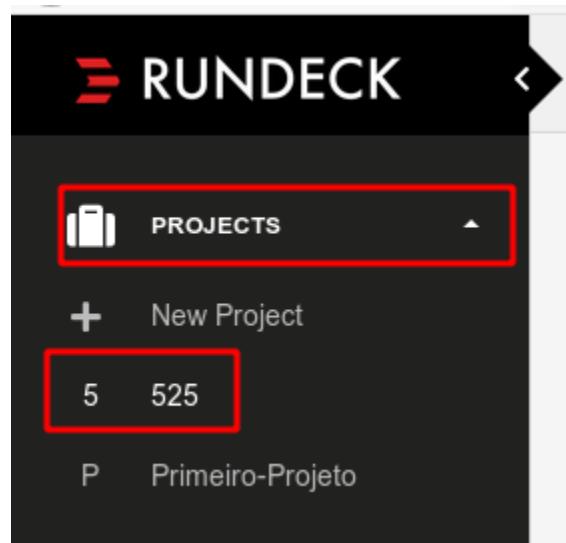


Figura 12.61: Rundeck Plugin



Figura 12.62: Rundeck Plugin

Details Workflow Nodes Schedule Notifications Other

No  Yes

On Success  Send Email  
 Webhook

Rocket.Chat Notification Sends a Rundeck Notification to Rocket.Chat

Rocket.Chat WebHook URL

URL of the Incoming WebHook/Integration that is configured in Rocket.Chat

Channel

The Rocket.Chat channel to send notification messages to.

Template

Message template.

Message on abort

Send a message when a job is aborted.

Figura 12.63: Rundeck Plugin

525

**Verificando hosts** Modified in 1h11m

Job para verificação do /etc/hosts

fb43a4be-890c-4df8-8160-57add16878bc

Action Definition

Follow execution Log Output

Nodes  Change the Target Nodes (5)

Run Job Now ▶

Figura 12.64: Rundeck Plugin

525

**Verificando hosts** Modified in 1h11m

Job para verificação do /etc/hosts

fb43a4be-890c-4df8-8160-57add16878bc

Action Definition

Follow execution Log Output

Nodes  Change the Target Nodes (5)

Run Job Now ▶

Figura 12.65: Rundeck Plugin

# Implantando e Utilizando ChatBots

## ChatBots

ChatBots são programas de computador que tentam simular um ser humano através de uma “Inteligência” e responder perguntas ou executar diversas ações

Os ChatBots mais conhecidos e utilizados no mercado são: **Hubot**, **LITA** e **Err**

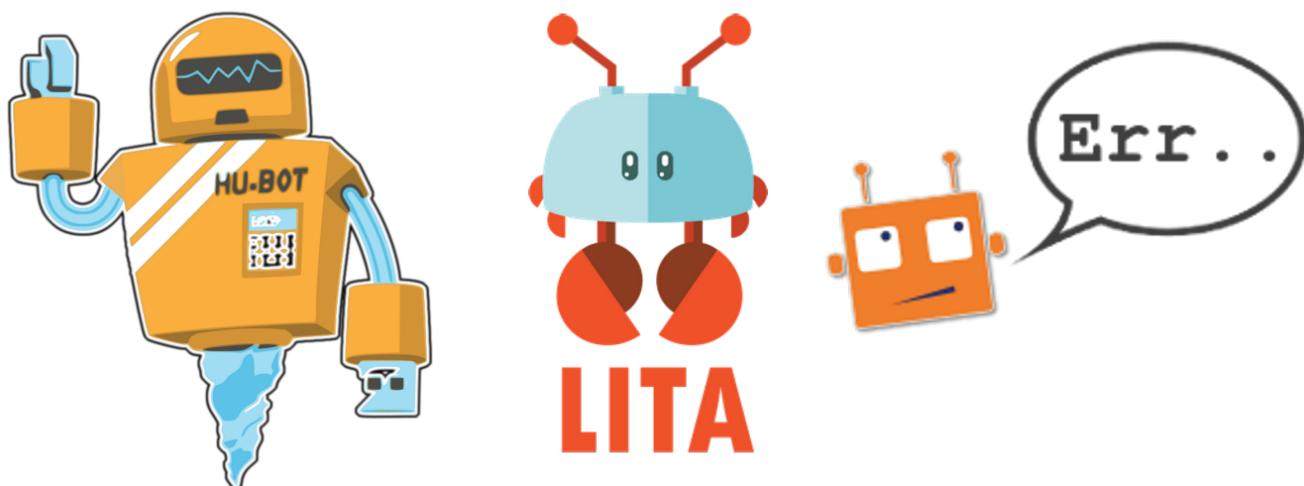


Figura 12.66: Bots

- **Hubot** - ChatBot free e Open Source desenvolvido pelo Github em CoffeeScript / NodeJS
- **LITA** - ChatBot free e Open Source desenvolvido Ruby
- **Err** - ChatBot free e Open Source desenvolvido em Python

Iremos utilizar o Hubot por se tratar do ChatBot mais utilizado no mercado e possuir numeros scripts da comunidade para utilização.

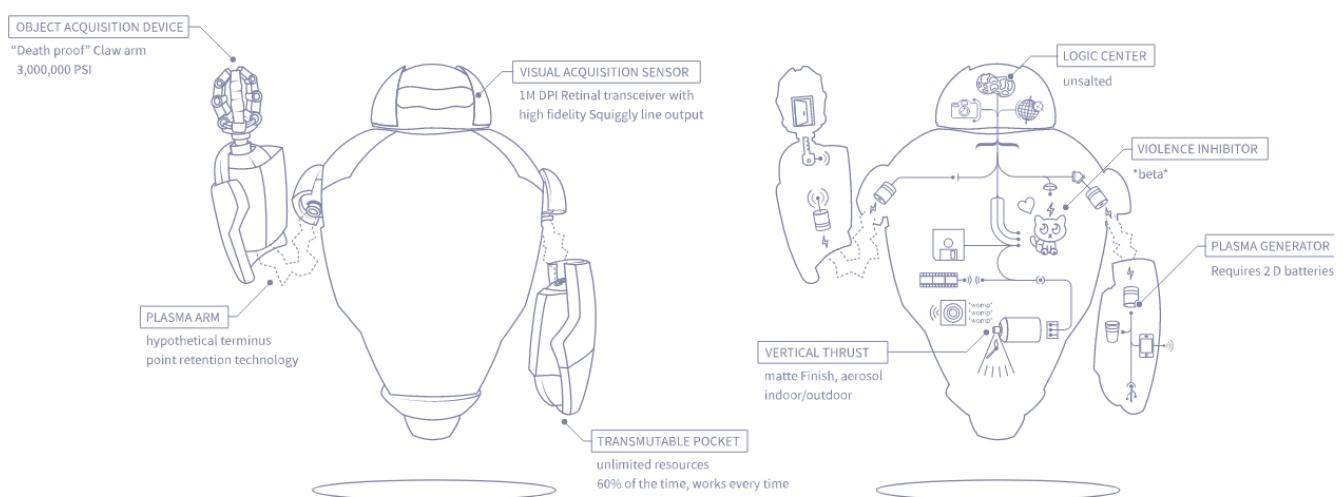
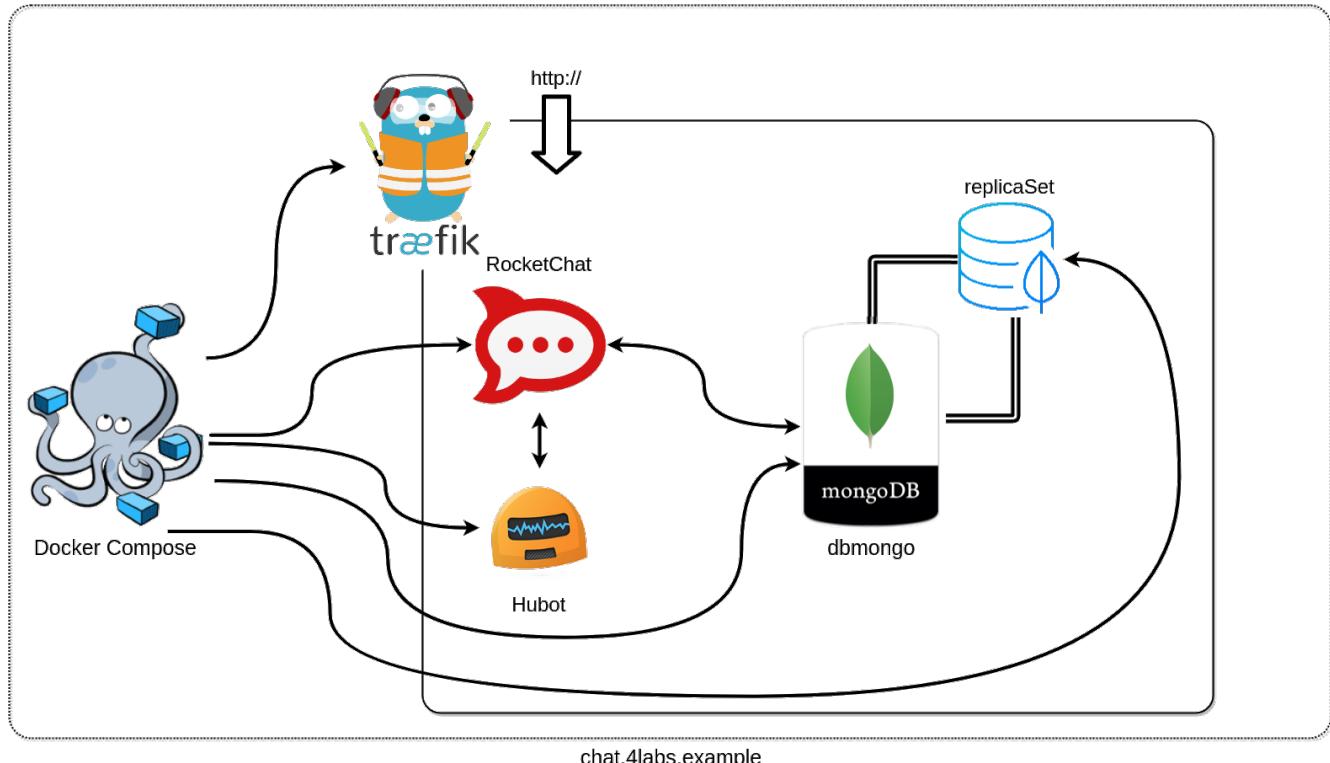


Figura 12.67: Hubot

O hubot aceita scripts na linguagem CoffeeScript, é possível que o usuário crie suas próprias interações e construa a

inteligência do Bot.

## Arquitetura do Rocket.Chat + Hubot



chat.4labs.example

Figura 12.68: Arquitetura

Iremos adicionar um contêiner do Hubot para que o mesmo faça a conexão e mantenha o usuário bot online e respondendo comandos.

### Instalação do Hubot

É necessário atender alguns pré requisitos antes de efetuar a instalação do ChatBot, para isto vamos acessar o nosso Rocket.chat, efetuar o login e acessar o menu de Administração.

Criaremos agora o Cyber Tux, nosso ChatBot que realizará as tarefas através do Rocket.Chat

É interessante que seja criado um personagem para o nosso ChatBot, Isso faz com que o ambiente se torne mais agradável.

Também é possível criar interações extra-operacionais para o ChatBot, como por exemplo **cantar (em texto) parabéns** aos aniversariantes do dia. Atitudes como esta criam ótimos ambientes de descontração.

No menu do usuário adicione um novo usuário que será impersonado pelo nosso ChatBot

Preencha os campos, adicione um avatar, adicione a role bot e clique em **Save**

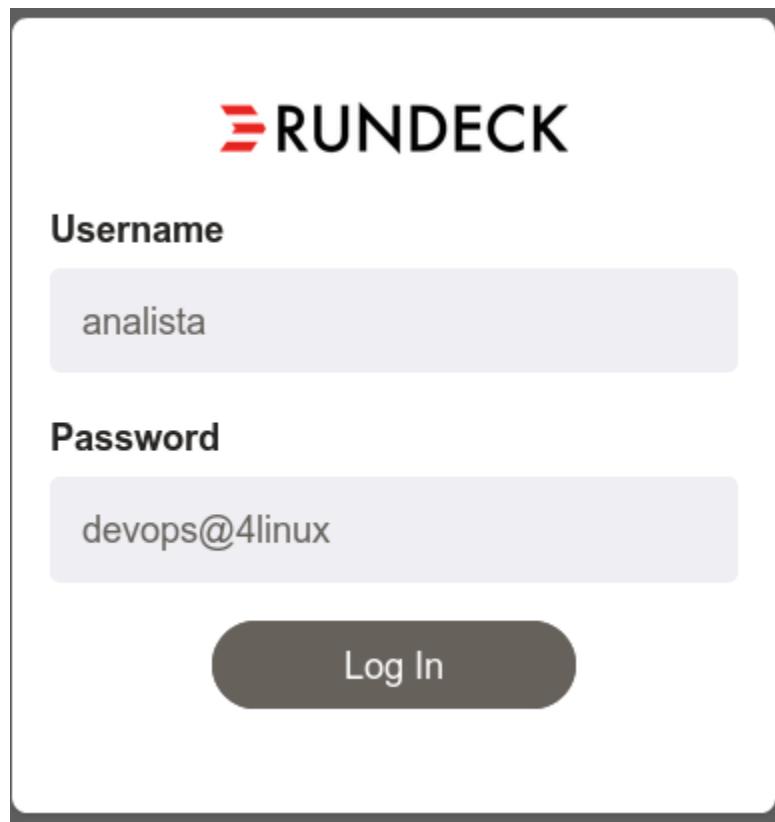


Figura 12.69: Login

Profile Picture

The user profile section includes:

- Profile Picture:** A circular icon of a robot named Cyber Tux.
- Avatar Options:** Buttons for 'U' (Upload), a cloud (Use URL for avatar), and a link icon. A red arrow points to the 'Use URL for avatar' button.
- Name:** Input field containing 'Cyber Tux'.
- Username:** Input field containing '@ cybertux'.
- Email:** Input field containing 'bot@4labs.example'.
- Verified:** A green radio button labeled 'Verified'.
- >Password:** Input field containing 'devops@4linux'.
- Require password change:** A toggle switch set to 'on'.
- Roles:** A single role assigned: 'bot'.
- Add Role:** A 'Select a Role' dropdown and a 'Add Role' button.
- Additional Options:** Two checkboxes: 'Join default channels' and 'Send welcome email'.

Red boxes highlight the 'Use URL for avatar' button, the 'devops@4linux' password field, the 'Require password change' toggle, the 'bot' role, the 'bot@4labs.example' email field, and the 'verified' status.

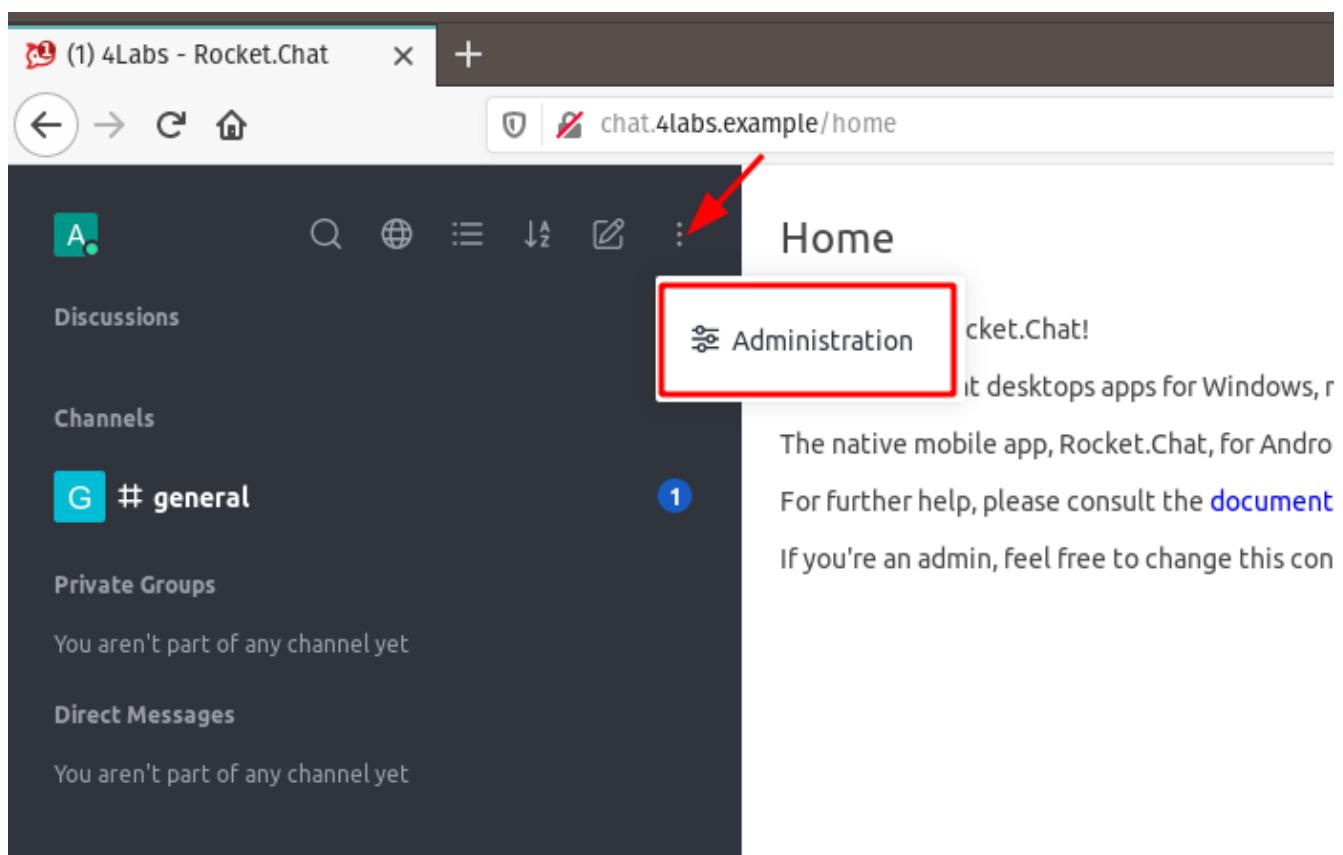


Figura 12.70: Configuração Rocketchat



Figura 12.71: Cyber Tux

Administration

- Info
- Import
- # Rooms
- Users**
- Connectivity Services
- Custom Sounds
- Custom Emoji
- Federation Dashboard
- Integrations

Users

Search

SHOWING 2 RESULTS

Name	Username	Email	Roles	Status
A Analist...	analista	analista@4l...	admin	online
Rocket....	rocket.cat		bot	online

A red arrow points to the '+' icon in the top right corner of the user list.

Figura 12.72: Novo Usuário

Existe uma imagem chamada **cybertux.png** na raiz do repositório do curso **4525**, caso deseje é possível utilizar qualquer outra imagem de avatar clicando em **upload**

Na tela inicial vamos criar o canal **#chatops** e adicionar o usuário **cybertux**

## Create a New Channel

Channels are where your team communicate

Public Channel

Everyone can access this channel

Read Only Channel

All users in the channel can write new messages

Broadcast Channel

Only authorized users can write new messages, b

Channel Name

# devops

Invite Users

@ D diretor x

Please enter usernames:

Create

Figura 12.73: Novo Canal

Vamos editar o canal e adicionar um Announcement e Topic.

Iremos executar o hubot através do docker-compose do Rocket.Chat na máquina container.

Acesse a máquina container troque para o usuário root e acesse a pasta do compose

```
1 vagrant ssh container
2 sudo su -
3 cd /root/rocketchat
```

Adicione o conteúdo do hubot ao docker-compose

```
1 vim docker-compose.yml
```

## Create a New Channel

Channels are where your team communicate

Private Channel

Just invited people can access this channel.

Read Only Channel

All users in the channel can write new messages

Broadcast Channel

Only authorized users can write new messages, but the other users will be able to read them.

Channel Name

chatops

Invite Users

@



cybertux

x Please enter usernames...



Figura 12.74: Novo Canal

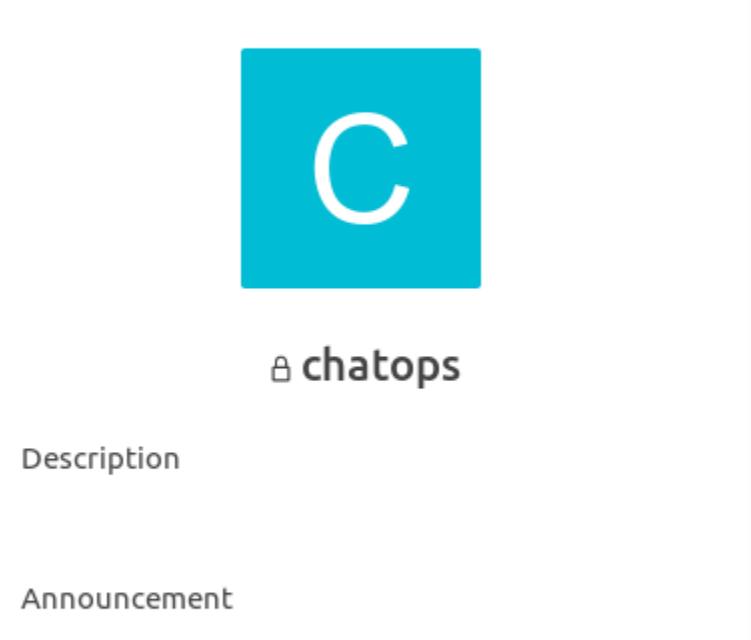
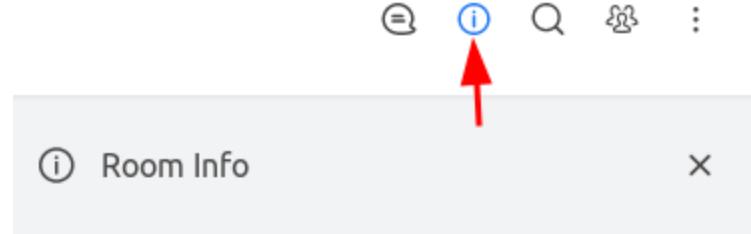


Figura 12.75: Novo Canal

Room Info

Name  
chatops

Topic\*  
ChatBot @cybertux

Announcement\*  
ChatOps Channel

**Cancel** **Reset** **Save**

The image shows a 'Room Info' dialog box. At the top left is an info icon and the title 'Room Info'. At the top right is a close button ('X'). Below the title are three input fields: 'Name' containing 'chatops', 'Topic\*' containing 'ChatBot @cybertux' (which is highlighted with a red border), and 'Announcement\*' containing 'ChatOps Channel'. At the bottom are three buttons: 'Cancel' (white background), 'Reset' (light gray background), and 'Save' (blue background). A red arrow points from the text 'Novo Canal' in the caption below to the 'Save' button.

Figura 12.76: Novo Canal

```

1 version: '3'
2
3 services:
4   rocketchat:
5     image: rocketchat/rocket.chat:latest
6     restart: unless-stopped
7     environment:
8       - PORT=3000
9       - ROOT_URL=http://localhost:3000
10      - MONGO_URL=mongodb://mongo:27017/rocketchat
11      - MONGO_OPLOG_URL=mongodb://mongo:27017/local
12     depends_on:
13       - mongo
14     labels:
15       - "traefik.backend=rocketchat"
16       - "traefik.frontend.rule=Host: chat.4labs.example"
17
18 mongo:
19   image: mongo:4.0
20   restart: unless-stopped
21   volumes:
22     - $PWD/db:/data/db
23   command: mongod --smallfiles --oplogSize 128 --replicaSet rs0 --storageEngine=mmapv1
24   labels:
25     - "traefik.enable=false"
26
27 mongo-init-replica:
28   image: mongo:4.0
29   command: >
30     bash -c
31     "for i in `seq 1 30`; do
32       mongo mongo/rocketchat --eval \
33         rs.initiate({
34           _id: 'rs0',
35           members: [ { _id: 0, host: 'localhost:27017' } ]})\" &&
36       s=$$? && break || s=$$?;
37       echo \"Tried $$i times. Waiting 5 secs...\";
38       sleep 5;
39     done; (exit $$s)"
40   depends_on:
41     - mongo
42
43 hubot:
44   image: rocketchat/hubot-rocketchat:latest
45   hostname: hubot
46   restart: unless-stopped
47   extra_hosts:
48     - "automation.4labs.example:10.5.25.10"
49   environment:
50     - ROCKETCHAT_URL=rocketchat:3000
51     - ROCKETCHAT_ROOM=general
52     - ROCKETCHAT_USER=cybertux
53     - ROCKETCHAT_PASSWORD=devops@4linux
54     - ROCKETCHAT_AUTH=password
55     - BOT_NAME=chatops
56     - LISTEN_ON_ALL_PUBLIC=true
57     - EXTERNAL_SCRIPTS=hubot-help,hubot-seen,hubot-links,hubot-diagnostics,hubot-rundeck
58   depends_on:
59     - rocketchat
60   labels:
61     - "traefik.enable=false"
62
63 traefik:
64   image: traefik:1.7
65   restart: unless-stopped
66   command: traefik /
67     --docker /
68     --api /
69     --docker.domain=4labs.example /
70     --entrypoints="Name:http Address::80" /

```

```

71      --entryPoints="Name:https Address::443" /
72      --defaultentrypoints="http,https"
73  ports:
74    - 80:80
75    - 443:443
76    - 8080:8080
77  volumes:
78    - /var/run/docker.sock:/var/run/docker.sock

```

Execute o compose e acompanhe os logs

```

1 docker-compose up -d
2 docker-compose logs -f

```

Após a execução o usuário **@cybertux** ficará online, sendo impersonado pelo container hubot.

Name	Username	Email	Roles	Status
A Analista DevOps	analista	analista@4labs.example	admin	online
 cybertux	cybertux	bot@4labs.example	bot	online
D Diretor DevOps	diretor	diretor@4labs.example	user	offline
 Rocket.Cat	rocket.cat		bot	online

Figura 12.77: Cyber Tux

## Interagindo com o Hubot

Acesse a sala **#chatops** e faça interações com o bot através do comando **@cybertux help**

Verifique no terminal da máquina container que a mensagem foi recebida e interpretada pelo cérebro do hubot

Pecisamos agora integrar nosso bot com o Rundeck acesse o endereço <http://automation.4labs.example:4440> e efetue o login

Iremos criar uma **API TOKEN** para que nosso bot utilize e consiga ter acesso ilimitado ao Rundeck, clique no menu do usuário e em seguida em **Profile**

Clique no ícone + em **API TOKEN**

Clique em **Generate New Token**

Clique em **Show Token** e copie o mesmo, iremos utilizá-lo no Rocket.Chat

No Rocket.Chat o primeiro passo necessário é adicionar o alias do rundeck através do comando

```

1 @<BOTNAME> rundeck add alias <ALIAS> <ADDRESS> <TOKEN>

```

Agora podemos executar alguns comandos para o hubot interagir com o rundeck.

Liste os projetos do rundeck

```

1 @cybertux rundeck projects automation

```

Liste os Jobs do projeto de InfraAgil

```

1 @cybertux rundeck 'InfraAgil' jobs automation

```

 analista Admin Owner 11:17 PM  
 cyberTux help

---

 cyberTux 11:17 PM  
 cyberTux adapter - Reply with the adapter  
 cyberTux echo <text> - Reply back with <text>  
 cyberTux help - Displays all of the help commands that this bot knows about.  
 cyberTUX help <query> - Displays all help commands that match <query>.  
 cyberTUX links clear - Clears all tracked links  
 cyberTUX links list [<number>] - Returns the last 5 (or <number> if specified) links shared  
 cyberTUX ping - Reply with pong  
 cyberTUX rundeck add alias [alias name] [url] [authToken] - sets the alias for a given url and authentication token  
 cyberTUX rundeck clear alias [alias name] - removed the given alias  
 cyberTUX rundeck projects [alias] - Gets a list of the projects for the given server alias  
 cyberTUX rundeck show aliases - shows the aliases for the list of rundeck instances  
 cyberTUX rundeck status '[job]' '[project]' [alias] - Shows the current status for the latest execution of the given job  
 cyberTUX rundeck trigger '[job]' '[project]' [alias] [args] - Triggers the given job for the given project  
 cyberTUX seen <user> - show when and where user was last seen  
 cyberTUX seen in last 24h - list users seen in last 24 hours  
 cyberTUX time - Reply with current time  
 hubut rundeck jobs '[project]' [alias] - Gets a list of all the jobs in the given project for the given server alias  
 rundeck show status of (.\*) (?::in|for) (.\*) (?::in|for) (.\*)

Figura 12.78: Cyber Tux

```

[Sun Nov 17 2019 02:17:39 GMT+0000 (UTC)] INFO Message received with ID id
[Sun Nov 17 2019 02:17:39 GMT+0000 (UTC)] INFO Message receive callback id KFFBehmCwKBTTc2F ts Sun No
[Sun Nov 17 2019 02:17:39 GMT+0000 (UTC)] INFO [Incoming] analista: @cyberTux help
[Sun Nov 17 2019 02:17:39 GMT+0000 (UTC)] INFO Checking to see if method: getRoomNameById exists
[Sun Nov 17 2019 02:17:39 GMT+0000 (UTC)] INFO Looking up Room Name for: hjsDghqjkNDn6gZJn
[Sun Nov 17 2019 02:17:39 GMT+0000 (UTC)] INFO setting roomName: chatops
[Sun Nov 17 2019 02:17:39 GMT+0000 (UTC)] INFO Message sent to hubot brain.
[Sun Nov 17 2019 02:17:39 GMT+0000 (UTC)] INFO Sending Message To Room: chatops
[Sun Nov 17 2019 02:17:39 GMT+0000 (UTC)] INFO Looking up Room ID for: chatops
[Sun Nov 17 2019 02:17:39 GMT+0000 (UTC)] INFO Message received with ID id
  
```

Figura 12.79: Cyber Tux

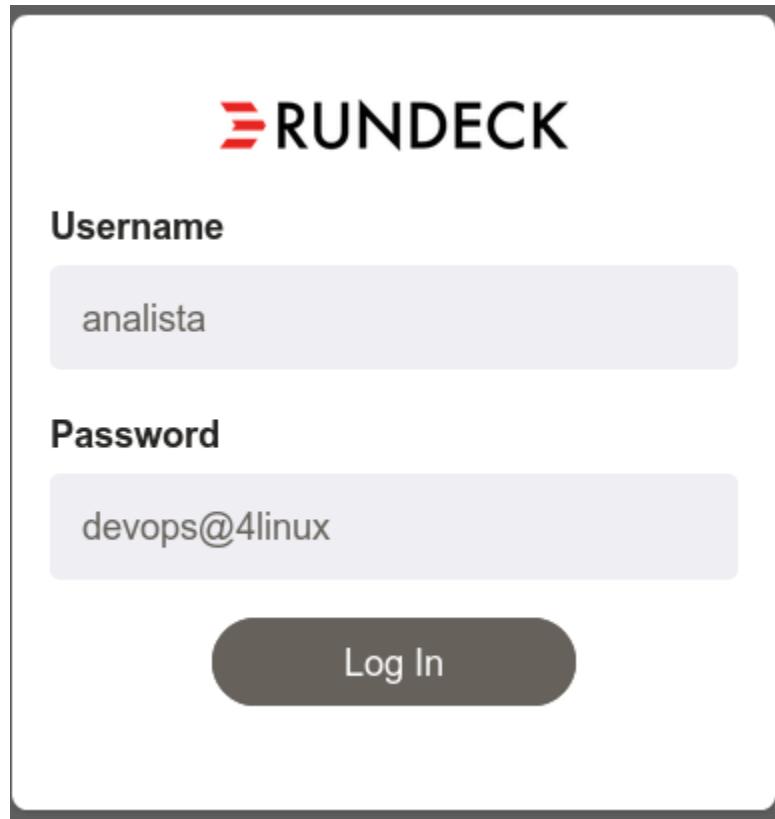


Figura 12.80: Rundeck Login

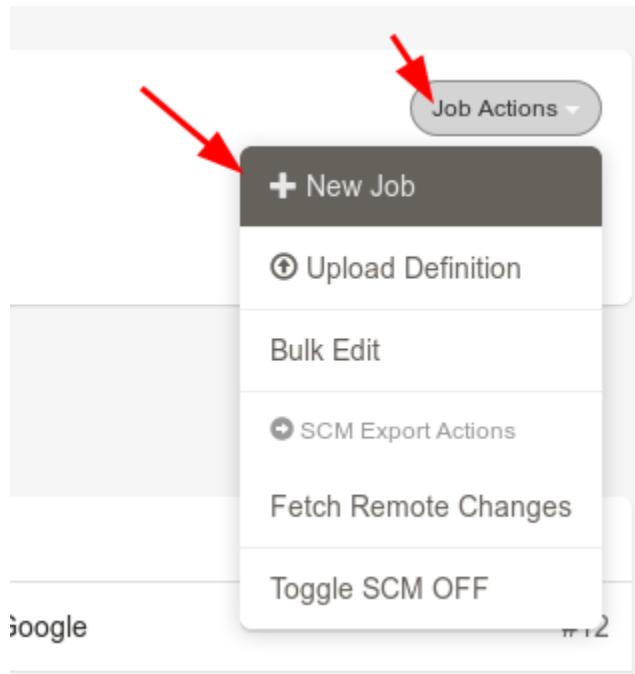


Figura 12.81: API Token

## Create New Job: Testes de Infraestrutura

Details   Workflow   Nodes   Schedule   Notifications   Other

---

Job Name	Testes de Infraestrutura
Description	<p>Edit</p> <p>1 Execução dos testes de infraestrutura com InSpec</p>

---

The first line of the description will be shown in plain text, the

Figura 12.82: API Token

1. ➔ inspec exec /root/inspec-profiles/automation -t ssh://automation.4labs.example -i /root/.ssh/id\_rsa  
testes de automação
2. ➔ inspec exec /root/inspec-profiles/compliance -t ssh://compliance.4labs.example -i /root/.ssh/id\_rsa  
testes de compliance
3. ➔ inspec exec /root/inspec-profiles/container -t ssh://container.4labs.example -i /root/.ssh/id\_rsa  
testes de container
4. ➔ inspec exec /root/inspec-profiles/scm -t ssh://scm.4labs.example -i /root/.ssh/id\_rsa  
testes de scm
5. ➔ inspec exec /root/inspec-profiles/log -t ssh://log.4labs.example -i /root/.ssh/id\_rsa  
testes de log

Figura 12.83: API Token

Create New Job: Testes de Infraestrutura

Details Workflow **Nodes** Schedule Notifications Other

Dispatch to Nodes  Execute locally

Choose whether the Job will run on filtered nodes or only on the local node.

Node Filter

Exclude Filter

Show Excluded Nodes  Yes  No

If true, the excluded nodes will be indicated when running the Job. Otherwise they will not be shown at all.

Matched Nodes

Figura 12.84: API Token

**Rank Order**  Ascending  Descending

**If a node fails**  Fail the step without running on any remaining node  Continue running on any remaining nodes before failing

**If node set empty**  Fail the job.  Continue execution.

**Node selection**  Target nodes are selected by default  The user has to explicitly select target nodes

**Orchestrator**

This can be used to control the order and timing in which

Figura 12.85: API Token

A analista Admin Owner 11:25 PM  
 cybertux rundeck add alias automation <http://automation.4labs.example:4440> XtOgUP5ezOeXzjeza80uiEUjZwYNRVTlIe

cybertux 11:25 PM  
 The rundeck system alias automation for <http://automation.4labs.example:4440> has been added to the brain

Figura 12.86: Cyber Tux

**A** analista Admin Owner 11:33 PM  
**cybertux** rundeck projects automation

 **cybertux** 11:33 PM  
InfraAgil - Infraestrutura Ágil com Práticas DevOps  
Primeiro-Projeto - Meu primeiro projeto com Rundeck

Figura 12.87: Cyber Tux

**A** analista Admin Owner 11:34 PM  
**cybertux** rundeck 'InfraAgil' jobs automation

 **cybertux** 11:34 PM  
Testes de Infraestrutura - Execução dos testes de infraestrutura com InSpec  
Verificando hosts - Job para verificação do /etc/hosts  
Verificar DNS Google - Job para verificar se o dns da google está acessível pelos servidores.

Figura 12.88: Cyber Tux

Execute o Job Verificando Hosts

```
1 @cybertux rundeck trigger 'Verificando hosts' 'InfraAgil' automation
```

**A** analista Admin Owner 11:50 PM  
**cybertux** rundeck trigger 'Verificando hosts' 'InfraAgil' automation

 **cybertux** 11:50 PM  
Successfully triggered a run for the job: Verificando hosts

Figura 12.89: Cyber Tux

#rundeck

Rundeck Notifications

Start of conversation

Job Name	Project	Status	Execution ID
/Verificando hosts	InfraAgil	Started	#17
/Verificando hosts	InfraAgil	Succeeded	#17

Verifique no canal **#rundeck** a publicação da execução do Job

Verifique no rundeck que o Job foi executado

The screenshot shows the Rundeck Activity page with a header '525' and a search bar. Below is a table of recent executions:

Date	Status	Duration	By	Project	ID
11/16/2019 11:51 PM	5 ok	a few seconds	by analista	Verificando hosts	#17
11/16/2019 10:50 PM	1 failed	a few seconds	by analista	Verificando hosts	#16
11/16/2019 10:49 PM	5 ok	a few seconds	by analista	Verificando hosts	#15
11/16/2019 3:11 PM	1 ok	a minute	by analista	Testes de Infraestrutura	#14

Figura 12.90: Rundeck Activity

É possível verificar nos logs todas as chamadas de API que o hubot faz de acordo com a solicitação

```
[Sun Nov 17 2019 02:50:46 GMT+0000 (UTC)] INFO Message received with ID id
[Sun Nov 17 2019 02:50:46 GMT+0000 (UTC)] INFO Message receive callback id nXxZA87yc289oorCo ts Sun Nov 17 2019 02:50:46 GMT+0000 (UTC)
[Sun Nov 17 2019 02:50:46 GMT+0000 (UTC)] INFO [Incoming] analista: @cybertux rundeck trigger 'InfraAgil' jobs automation
[Sun Nov 17 2019 02:50:46 GMT+0000 (UTC)] INFO setting roomName: chatops
[Sun Nov 17 2019 02:50:46 GMT+0000 (UTC)] INFO Message sent to hubot brain.
http://automation.4labs.example:4440/api/12/project/InfraAgil/jobs
[Sun Nov 17 2019 02:50:46 GMT+0000 (UTC)] INFO Sending Message To Room: chatops
[Sun Nov 17 2019 02:50:46 GMT+0000 (UTC)] INFO Sending Message To Room: chatops
[Sun Nov 17 2019 02:50:46 GMT+0000 (UTC)] INFO Sending Message To Room: chatops
[Sun Nov 17 2019 02:50:46 GMT+0000 (UTC)] INFO Message received with ID id
[Sun Nov 17 2019 02:50:46 GMT+0000 (UTC)] INFO Message received with ID id
[Sun Nov 17 2019 02:50:51 GMT+0000 (UTC)] INFO Message receive callback id mnvfbM6nfYZy7axcH ts Sun Nov 17 2019 02:50:51 GMT+0000 (UTC)
[Sun Nov 17 2019 02:50:51 GMT+0000 (UTC)] INFO [Incoming] analista: @cybertux rundeck trigger 'Verificando hosts' 'InfraAgil' automation
[Sun Nov 17 2019 02:50:51 GMT+0000 (UTC)] INFO setting roomName: chatops
[Sun Nov 17 2019 02:50:51 GMT+0000 (UTC)] INFO Message sent to hubot brain.
http://automation.4labs.example:4440/api/12/project/InfraAgil/jobs
http://automation.4labs.example:4440/api/12/job/fb43a4be-890c-4df8-8160-57add16878bc/run?argString=
[Sun Nov 17 2019 02:50:52 GMT+0000 (UTC)] INFO Sending Message To Room: chatops
[Sun Nov 17 2019 02:50:52 GMT+0000 (UTC)] INFO Message received with ID id
[Sun Nov 17 2019 02:50:52 GMT+0000 (UTC)] INFO Message received with ID id
[Sun Nov 17 2019 02:50:52 GMT+0000 (UTC)] INFO Message receive callback id DvMyaMdmcocRrhN4 ts Sun Nov 17 2019 02:50:52 GMT+0000 (UTC)
[Sun Nov 17 2019 02:50:52 GMT+0000 (UTC)] INFO [Incoming] rocket.cat:
[Sun Nov 17 2019 02:50:52 GMT+0000 (UTC)] INFO Looking up Room Name for: FkAvdjwiJM25QQfgv
[Sun Nov 17 2019 02:50:52 GMT+0000 (UTC)] INFO setting roomName: rundeck
[Sun Nov 17 2019 02:50:52 GMT+0000 (UTC)] INFO Message sent to hubot brain.
[Sun Nov 17 2019 02:51:06 GMT+0000 (UTC)] INFO Message received with ID id
[Sun Nov 17 2019 02:51:06 GMT+0000 (UTC)] INFO Message receive callback id 4MCXyzGwXM6Ks9F4d ts Sun Nov 17 2019 02:51:06 GMT+0000 (UTC)
[Sun Nov 17 2019 02:51:06 GMT+0000 (UTC)] INFO [Incoming] rocket.cat:
[Sun Nov 17 2019 02:51:06 GMT+0000 (UTC)] INFO setting roomName: rundeck
[Sun Nov 17 2019 02:51:06 GMT+0000 (UTC)] INFO Message sent to hubot brain.
```

Figura 12.91: Container Logs