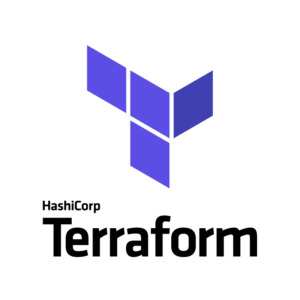
**Automatize a sua infraestrutura com Terraform**

* Blog
* 28 de Dezembro de 2018
* por Cristian Trucco



Se você viu o filme O Homem de Aço vai se lembrar dessa cena em que o general Zod quer ‘terraformar’ o planeta Terra em Krypton. Bom… é exatamente isso que o Terraform faz. Brincadeiras à parte, vamos ver neste post como essa ferramenta funciona e porque tem tanta gente usando e falando sobre ela.

Basicamente, o Terraform transforma o planeta Terra em Krypton. Não aguentei. rs Falando sério agora, o Terraform transforma o nosso ambiente da forma que desejarmos, por isso vem ganhando muito espaço no mundo DevOps: por conta do provisionamento de infraestrutura automatizada. Mas o que muitos ainda não sabem é que ela vai além, já que têm outros recursos bem legais.



**O que é?**

Primeiramente, se você nunca ouviu falar do [Terraform](https://www.terraform.io/) ele é basicamente uma ferramenta para construir, alterar e configurar infraestrutura de maneira segura e eficiente. Ele é **opensource** e foi desenvolvido pela [Hashicorp](https://www.hashicorp.com/), empresa bem famosa por ter vários produtos/ferramentas amplamente usados pela comunidade DevOps, como **Consul**, **Vault**, **Packer** e **Vagrant**.

E se você não conhece nenhuma dessas ferramentas e quiser conhecer, coloca aí nos comentários que faremos um artigo falando delas, que são muito boas para atender determinados cenários.

Mas voltando… o Terraform dá suporte a quase todos os principais Providers de nuvem, citando os mais conhecidos: AWS, Google Cloud, Azure, Digital Ocean etc.

Mas aí você pensa: “a mas a AWS tem o ***CloudFormation*** e o ***Beanstalk***, então porque eu usaria outra ferramenta?” Não precisa responder, eu respondo por você, meu jovem. rs

A vantagem de usar o **Terraform** é que primeiro, como falamos anteriormente, ele é **MultiCloud**, ou seja, conseguimos provisionar vários ambientes em diferentes provedores de nuvem usando uma única ferramenta. Segundo, tando o [CloudFormation](https://aws.amazon.com/cloudformation/) como o [Beanstalk](https://aws.amazon.com/pt/elasticbeanstalk/) são serviços pagos na AWS e só funcionam na AWS, ou seja, se você migrar para uma outra nuvem, basicamente, toda a sua infraestrutura automatizada vai estar “presa” na AWS. Mas digo isso sem criticar os serviços, que são ótimos!

Outro benefício é a vantagem de não precisar aprender uma linguagem nova para criar a infraestrutura no Terraform, a **Hashcorp** criou a [**HCL**](https://github.com/hashicorp/hcl) (HachiCorp Configuration Language). Com isso, a ferramenta se tornou bem amigável e fácil de usar: você olha o código e sabe o que está sendo feito.

O Terraform também aceita o formato **Json**, mas o uso do **HCL** é fortemente recomendado visto que você escreve bem menos **código** usando **HCL** e a chance de você esquecer uma chave na hora de criar o Json é grande. rs

Exemplo de arquivo .**tf**, extensão usada pelo Terraform para provisionamento de uma instância **EC2**.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | # Create a new instance of the latest Ubuntu 14.04 on an  # t2.micro node with an AWS Tag naming it "HelloWorld"  provider "aws" {    region = "us-west-2"  }  data "aws\_ami" "ubuntu" {    most\_recent = true    filter {      name   = "name"      values = ["ubuntu/images/hvm-ssd/ubuntu-trusty-14.04-amd64-server-\*"]    }    filter {      name   = "virtualization-type"      values = ["hvm"]    }    owners = ["099720109477"] # Canonical  }    resource "aws\_instance" "web" {    ami           = "${data.aws\_ami.ubuntu.id}"    instance\_type = "t2.micro"    tags = {      Name = "HelloWorld"    }  } |

Quando falei **código**, deve ter **Sysadmin** que ficou de cabelo em pé… rs É desejável para uma melhor utilização da ferramenta que se conheça um pouco de lógica de programação, isso vai ajudar bastante na hora de usar os recursos avançados do Terraform ou ambientes mais complexos que demandem a criação e a dependência de vários objetos. Mas é bem tranquilo, não se assustem.

Um exemplo: vou subir uma aplicação e preciso criar um load balance com autoscale para fazer o balanceamento das aplicações e preciso criar antes o banco de dados para que a minha aplicação suba sem problemas. Isto é só um exemplo de uma infraestrutura que podemos pressionar de maneira automatizada usando a ferramenta.

**Infrastructure as code (IaC)**

Quando utilizamos qualquer ferramenta ou script que vai provisionar de maneira automatizada nosso ambiente, e vamos declarar/escrever isso em um arquivo, estamos falando de infraestrutura como código [IAC](https://pt.wikipedia.org/wiki/Infraestrutura_como_C%C3%B3digo). Seguindo essa abordagem, ganhamos várias vantagens:

* Documentação da Infraestrutura;
* Versionamento da Infraestrutura;
* Garantia da integridade e estado desejado da infraestrutura (Infraestrutura Imutável);
* Automatização do build da infra no Pipeline;
* Garantia do mesmo resultado todas as vezes que rodar o script, o resultado vai ser sempre o mesmo ([idempotência](https://pt.wikipedia.org/wiki/Idempot%C3%AAncia));
* Redução de erros humanos;
* Não há retrabalho: vc faz apenas uma vez;
* Pode ser usada em conjunto com Ansible,Chef ou Puppet etc.

E a lista só aumenta, mas acho que já te convenci a usar o Terraform. 🙂

Então como já é de costume aqui no Blog da [Concrete](http://www.concrete.com.br), vamos meter a mão na massa também.



**Tome a pílula vermelha e vou te mostrar um mundo novo!**

Preparei um repositório onde podemos brincar à vontade com o Terraform, sem a necessidade de criar uma conta na AWS, Google ou Azure. E o melhor: vamos fazer isso a **0800**! rs

**Coisa legais que conseguimos fazer no Terraform**

Antes uma dica rápida… Atualmente, trabalho em um projeto em que utilizamos o **Gitlab** e trabalhamos com micro-serviços, ou seja, sempre que criamos um serviço novo temos que criar um novo repositório no **Gitlab**. E isso com o tempo fica tedioso, monótono, chato e muito repetitivo. Acho que você já entendeu. rs

E eu nem preciso falar que já temos a ferramenta para resolver isso, né?



O Terraform consegue fazer isso para a gente.

[Neste link da documentação](https://www.terraform.io/docs/providers/gitlab/index.html) tem o que precisamos configurar para fazer o provisionamento de repositórios no Gitlab de maneira automatizada.

Mas depois você olha isso… agora vamos para a **demo**!

**Demo Terraform**

Agora sim vem a parte legal, vamos ver se esse Terraform é bom mesmo.

Toda abordagem da demo que vou usar aqui neste post está nesse repositório [aqui](https://gitlab.com/cristianvitortrucco/terraform-docker).

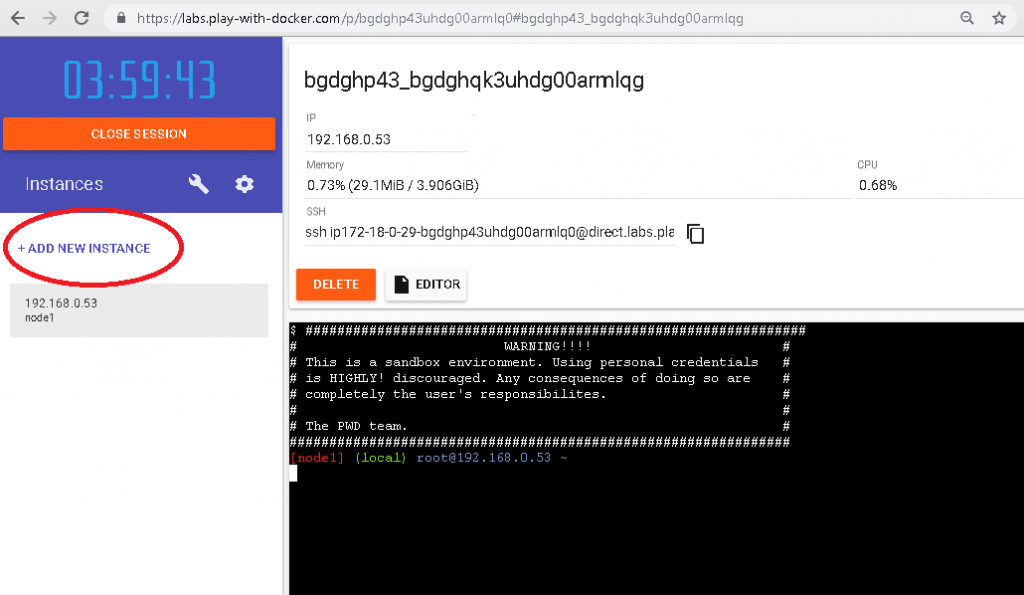
**Play With Docker (PWD)**

Vamos usar o [PWD](http://play-with-docker.com/) para não precisar instalar nada na nossa máquina e poder executar em qualquer lugar desde que tenha acesso à Internet e um browser.

OBS: Esse vai ser o ambiente que vamos usar para o lab, logo é necessário ter uma conta no Github ou Dockerhub para logar no PWD.

Após logar, precisamos lançar uma instância.

Lembrando que o PWD fornece uma janela de **4 horas** para usar o ambiente, após esse período a sessão expira e o conteúdo criado nessa instância é destruído. Mas não vamos demorar nem 5 minutos para subir o nosso ambiente, fica de boa aí.



**Instalando o Terraform no Play With Docker**

Todos os passos a seguir vão ser executados no terminal da instância do PWD.

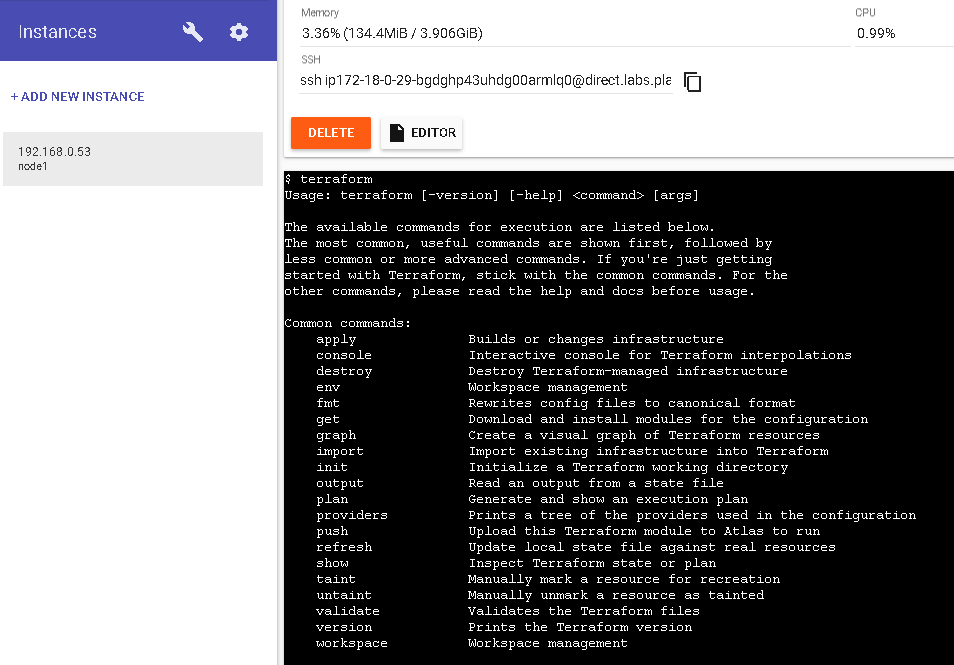
Copie e cole esses comandos no terminal:



|  |  |
| --- | --- |
| 1  2  3  4 | wget https://[releases.hashicorp.com/terraform/0.11.8/terraform\_0.11.8\_linux\_amd64.zip](http://releases.hashicorp.com/terraform/0.11.8/terraform_0.11.8_linux_amd64.zip)  unzip terraform\_0.11.8\_linux\_amd64.zip  mv terraform /bin/  terraform |

Essa sequência de comandos vai baixar o Terraform com o **Wget**, em seguida vamos descompactar o arquivo zip com unzip e mover para o diretório /**bin**. Após isso, o último comando vai checar se o Terraform está instalado.

A saída do comando deve ser igual a dessa da imagem:



**Clonando o Repositório do Lab**

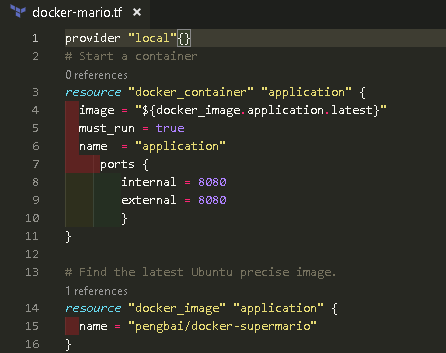
Após o **Terraform** estar devidamente instalado, vamos clonar o repositório onde tem o arquivo da infra que vamos criar usando o Terrraform. Em seguida, vamos entrar na pasta onde se encontra o código do Terraform mais.tf que vamos usar para buildar e fazer o deploy da nossa infra.



|  |  |
| --- | --- |
| 1 | git clone https://[gitlab.com/cristianvitortrucco/terraform-docker](http://gitlab.com/cristianvitortrucco/terraform-docker) |



|  |  |
| --- | --- |
| 1 | cd terraform-docker |



[**Código Terraform**](https://gitlab.com/cristianvitortrucco/terraform-docker/raw/master/main.tf)

**Dica Rápida**

Use o [**Vscode**](https://code.visualstudio.com/) e instale os plugins do Terraform para deixar ainda melhor a experiência de criar sua infraestrutura de forma fácil, rápida e com qualidade, usando esses recursos muito bacanas que o Vscode oferece.

**O que estamos fazendo nesse Lab?**

O que esse código (main.tf) acima está fazendo, exatamente: ele vai subir um container **Docker** usando o provider “local”, ou seja, ele vai usar a máquina onde estamos rodando o Terraform “localhost”. Lembrando que a instância do PWD já tem o Docker instalado, logo se for rodar esse código na máquina se certifique de que o Docker está lá.

Vamos usar a imagem docker (pengbai/docker-supermario) que está declarada no fim do arquivo, mapeando a porta do Host da **8080** para **8080** do container.

**Rodando o Terraform**

O comando a seguir vai inicializar o projeto no Terraform e, automaticamente, baixar os plugins necessários para que o Terraform consiga interagir com API do Docker.

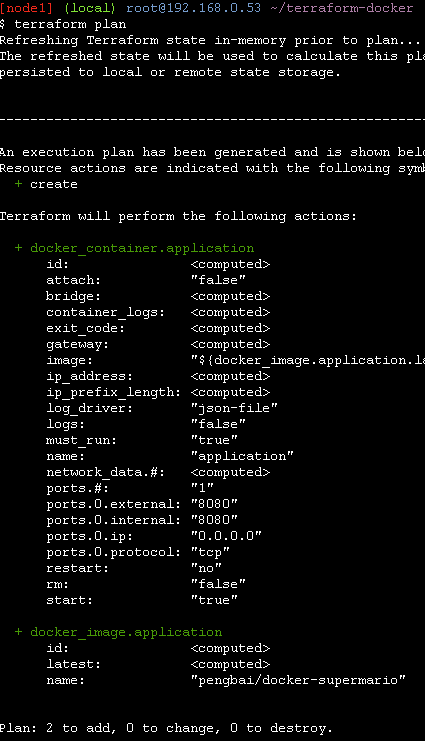


|  |  |
| --- | --- |
| 1 | terraform init |

O comando a seguir “plan” vai verificar o que o Terraform vai fazer. É uma prévia do que o Terraform vai provisionar, então é nesse momento que conferimos se está tudo ok conforme declaramos no código.



|  |  |
| --- | --- |
| 1 | terraform plan |



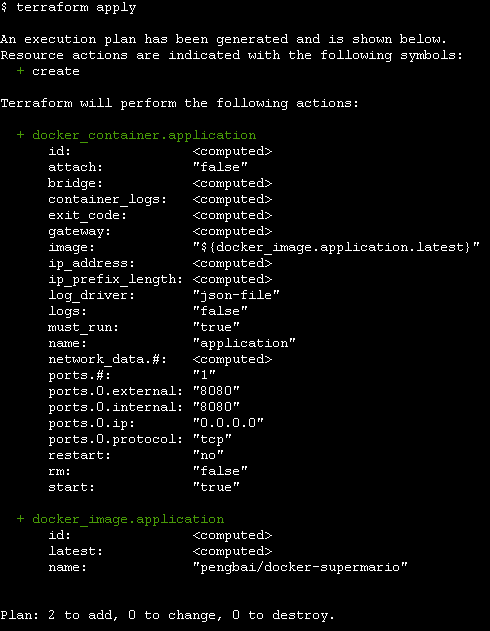
Outra coisa legal é que o Terraform informa que vai criar determinado recurso, por meio do sinal de “+”.

Após verificar a saída, o comando “apply” vai construir a infraestrutura declarada no arquivo **main**.tf.

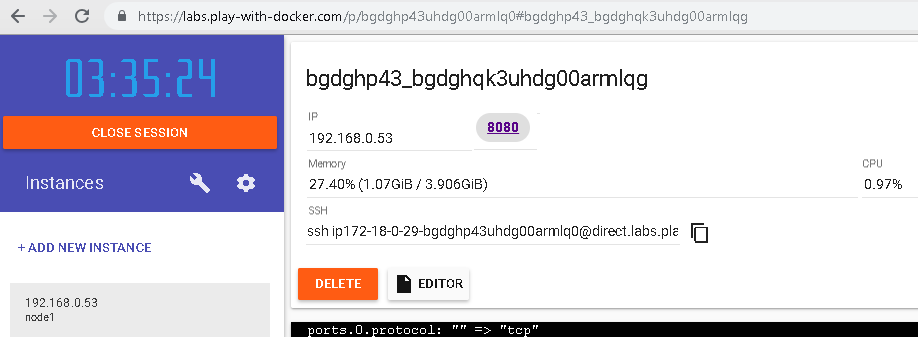


|  |  |
| --- | --- |
| 1 | terraform apply |

Agora é só confirmar com “yes” para realizar o deploy e construir de fato a nossa infra.

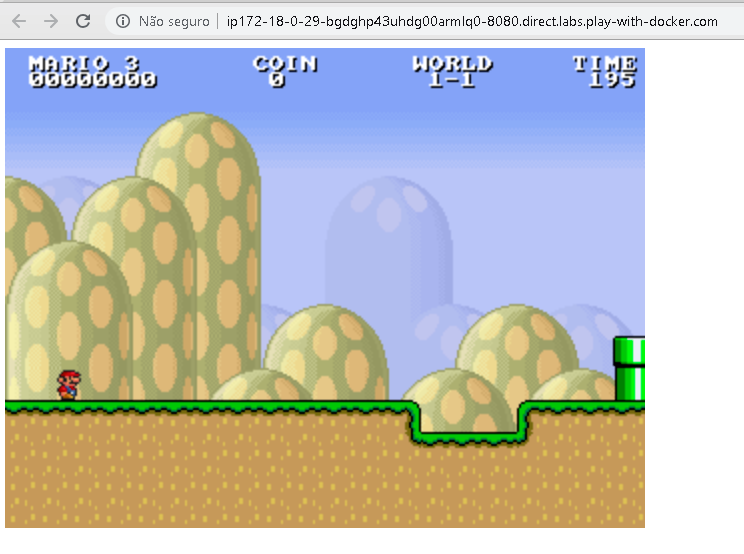


Depois disso, o painel do PWD vai expor a porta **8080**, então vamos clicar e acessar o link para chegar no container (Aplicação). Vale lembrar que essa porta é aquela que declaramos no Terraform.



Agora que você já estudou e aprendeu um conteúdo novo, podemos fazer uma pausa e jogar esse jogo do **Mario** rodando diretamente no container Docker que acabamos de provisionar com o Terraform.





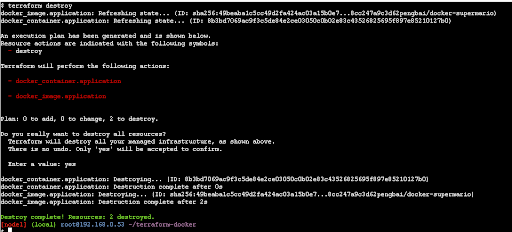
**Destruindo**

Se quiser destruir tudo o que criamos é só rodar outro comando do Terraform. Essa parte eu acho bem legal também. rs



|  |  |
| --- | --- |
| 1 | terraform destroy |

Na saída do comando, confirme com “yes” para que o Terraform destrua a infra declarada no arquivo main.tf. Uma outra coisa legal é que o Terraform mostra por meio de cores e sinais o que ele vai fazer no ambiente, por exemplo: quando vai destruir um recurso, ele o coloca em vermelho junto ao sinal de subtração “-”,  informando que aquele recurso declarado no arquivo vai ser excluído.



**Concluindo**

O legal disso tudo é que não usamos nenhum comando do Docker, tudo foi feito de maneira automatizada usando o Terraform. E isso é só o básico! No site do Terraform, eles fazem um overview bem legal sobre a ferramenta.

A documentação oficial está [aqui](https://learn.hashicorp.com/terraform/#getting-started) e o treinamento Katacoda 0800, [aqui](https://katacoda.com/learn?q=terraform&hPP=12&idx=scenarios&p=0&is_v=1). Este último link apresenta três cenários ensinando como usar o Terraform. Ideal para quem está começando com a ferramenta.

Legal, né?

Curte aí (rs), dê a sua opinião ou deixe uma mensagem dizendo se o conteúdo ajudou em alguma coisa.

Obrigado e até logo!



—

Aqui na Concrete, o capítulo de DevOps é o motor para a agilidade dos times, responsável por pensar estrategicamente na estrutura e arquitetura final, garantindo a qualidade. Justamente por automatizar processos e melhorar o desempenho do time de desenvolvimento, nossos “DevOps” trabalham desde o início da criação dos produtos, implementando processos de integração e entrega contínua, utilizando servidores de CI, repositórios de código, scripts de automatização e relatórios. Quer trabalhar com os melhores? Só vem! [concrete.com.br/vagas](http://concrete.com.br/vagas)