Sobre o Autor

Denilson Bonatti iniciou sua carreira como desenvolvedor web e posteriormente migrou para a área de servidores e infraestrutura. Foi professor do SENAC e SENAI São Paulo nos cursos de redes de computadores, infraestrutura e desenvolvimento de software. Hoje atua como consultor na área de servidores e DevOps, desenvolvedor e professor em cursos de servidores Windows/Linux e outros segmentos em diversas instituições de ensino físicas e online no Brasil e no mundo.

[https://www.linkedin.com/in/denilsonbonatti/]

Prefácio

Hoje tem muito se falado sobre DevOps no mundo da tecnologia da informação e às vezes a grande quantidade de informação sobre o assunto disponível nos mais diferentes tipos de mídia tende a confundir os iniciantes neste mundo. A intenção deste material é tentar esclarecer a verdadeira função do DevOps no desenvolvimento moderno de software e mostrar de forma clara e prática quais problemas essa nova cultura irá resolver nas pequenas, médias e grandes corporações e principalmente mostrar de forma prática a utilidade das ferramentas de DevOps.

Este material é composto por cinco capítulos, onde o primeiro apresenta uma introdução do que é o DevOps e quais problemas no mundo de desenvolvimento de software essa cultura veio solucionar. Em seguida o material irá abranger infraestrutura como código e suas vantagens e sua aplicabilidade no mundo do desenvolvimento de software dentro da cultura DevOps. São apresentados exemplos práticos utilizando as ferramentas Vagrant e Terraform.
O próximo capítulo demonstra qual é a função do Docker em um cenário de aplicações que utilizam a arquitetura de microsserviços. Exemplos práticos de utilização são demonstrados.

Como automatizar a publicação de aplicações e microsserviços dentro de um pipeline de deployment? Esse é o assunto do próximo capítulo. O capítulo aborda o GitLab como ferramenta de CI/CD em um projeto. O termo CI/CD é novo para você? Não se preocupe, iremos falar sobre isso no momento apropriado.

Em um cenário de microsserviços a escalabilidade de uma aplicação é imprescindível, desta forma, esse é o assunto do próximo capítulo onde é utilizado o Kubernetes como cluster de escala de containers.

Para finalizar serão demonstrados exemplos de como realizar testes de carga e testes automatizados em uma aplicação.

Convenções utilizadas neste material

As convenções tipográficas a seguir foram utilizadas neste material:

[itálico]

O texto em itálico entre colchetes delimita URLs.

```
Texto em fonte Courier New
com borda lateral a esquerda
```

O texto utilizando a fonte Courier New com borda lateral à esquerda indica códigos que devem ser utilizados em uma IDE (Ambiente de desenvolvimento integrado) como o Microsoft Visual Code.

```
$ Texto em fonte Courier New
# com borda lateral a direita iniciado com $ ou #
# O caracter $ indica um usuário convencional e o carácter # um superusuário
(root)
```

O texto utilizando a fonte Courier New com borda lateral à direita indica que os comandos indicados devem ser utilizados em um terminal de linha de comando, caso esteja utilizando o Linux ou PowerShell, caso você esteja utilizando o Windows.

O que é um Bootcamp?

Em educação, um bootcamp é um programa de treinamento intensivo e imersivo que é projetado para ensinar um conhecimento específico em um curto período de tempo. Isso é importante ressaltar, como o DevOps abrange uma grande quantidade de ferramentas e técnicas, as ferramentas demonstradas neste material serão aprendidas e discutidas de forma direta a solucionar o problema apresentado. Conceitos mais avançados de cada uma das ferramentas apresentadas não serão compreendidos neste material. É claro que depois de completar este bootcamp você pode e deve buscar mais informações sobre as ferramentas apresentadas.

O termo bootcamp costumava ser associado ao treinamento militar, mas agora é usado na educação e na área de tecnologia para descrever programas de aprendizado intensivo e prático. Bootcamps visam fornecer uma experiência de aprendizado acelerada, geralmente usando uma abordagem que se baseia em problemas do mundo real.

Os bootcamps são uma forma comum de ensinar habilidades técnicas em tecnologia da informação (TI), como programação, desenvolvimento web, ciência de dados e segurança cibernética, entre outras.

Não existe uma maneira "correta" de se ensinar DevOps, e também não existem as melhores ferramentas para cada um dos problemas que serão apresentados. A forma como este bootcamp foi criado e as ferramentas selecionadas são aquelas que eu como professor achei mais adequadas para que o conteúdo seja melhor compreendido. Mas fique tranquilo, as ferramentas utilizadas neste bootcamp são as mais utilizadas atualmente no mercado.

Introdução ao DevOps

Primeiros passos

OK! Mas, o que é o DevOps?

O DevOps tem o principal objetivo de melhorar a capacidade de uma empresa de fornecer aplicações e serviços em alta velocidade, otimizando e melhorando produtos mais rápido do que empresas que seguem procedimentos convencionais de gerenciamento de infraestrutura e desenvolvimento de software. As empresas podem atender melhor aos seus clientes e competir melhor no mercado com essa velocidade. Tudo se resume a organização, comunicação e velocidade de entrega. DevOps não é simplesmente um conjunto de softwares aplicados aos desenvolvimento de aplicação, DevOps é toda uma nova cultura de desenvolvimento.

O termo DevOps vem das palavras "desenvolvimento" (desenvolvimento) e "operações" (operações). O time de operações também é conhecido como o time de infraestrutura, eles são os profissionais responsáveis pelos servidores onde as aplicações são publicadas e disponibilizadas aos usuários finais.
É uma abordagem prática e cultural que visa melhorar a colaboração e a comunicação entre os times de operações de TI e desenvolvimento de software. O DevOps tem como objetivo principal garantir que o desenvolvimento de software e as operações sejam executados de maneira mais integrada e eficiente, o que resultará em entregas de software mais rápidas e confiáveis.

Principais atributos e conceitos do DevOps:

Colaboração: DevOps apóia a colaboração e a comunicação contínua entre os desenvolvedores e as equipes de operações. Uma abordagem mais abrangente ao ciclo de vida do desenvolvimento de software é encorajada por isso, pois elimina as barreiras tradicionais que existem entre essas equipes.

Essa colaboração é importante, imagine que o time de desenvolvimento esteja criando um software que utiliza como principal linguagem o PHP na versão 8.0. A aplicação em desenvolvimento precisa de bibliotecas específicas para ser executada. O banco de dados utilizado pela aplicação é um MySql na versão 5.7. O desenvolvedor está utilizando os softwares em suas versões corretas, mas quando esta aplicação for disponibilizada em um servidor para ser utilizada pelo usuário, as versões e softwares estão corretos no servidor?

É importante que o time de operações esteja ciente do desenvolvimento, pois uma versão errada de software ou uma biblioteca não instalada no servidor pode ocasionar bugs e consequente a aplicação pode ficar off-line.

Automação: O DevOps depende da automação para acelerar os processos e reduzir os erros humanos. A automação de compilação, teste, implantação e gerenciamento de infraestrutura estão todos dentro desse grupo.

Vamos imaginar que uma aplicação que será disponibilizada pela primeira vez necessite ser executada em um cluster de servidores com cinco máquinas virtuais. Cada uma das máquinas virtuais precisa possuir as seguintes configurações: 10GB de RAM, 4 núcleos de processamento, sistema operacional Linux Ubuntu 22 instalado em disco virtual de 10GB.

Os seguintes softwares precisam estar instalados: Apache, interpretador PHP com a biblioteca DBO. Um dos servidores precisa ter uma instância de banco de dados MySql na versão 5.7. Existe uma grande chance de erro humano nesta operação. Eu posso esquecer de instalar o PHP em um dos servidores, ou instalar um software na versão incorreta, ou até mesmo indicar a quantidade de disco e/ou memórias de maneira incorreta caso todo esse processo seja feito de maneira manual, sem falar no tempo que tudo isso irá demorar neste processo. Sabendo disso, a automação é muito importante. Utilizando ferramentas de infraestrutura como código toda essa configuração pode estar disponível na execução de um comando, e como a estrutura estará disponível em forma de código, toda ela poderá ser replicada e atualizada constantemente.

Entrega Contínua: Uma prática fundamental do DevOps é a entrega contínua, que visa fornecer software de forma rápida e consistente. A automação de todo o processo de entrega, desde a integração contínua até a implantação contínua, está incluída nisso.

Até pouco tempo atrás, a disponibilização de uma nova versão ou nova feature de um software levava semanas. Havia um grande planejamento de como toda essa operação seria realizada, incluindo reuniões com os times de desenvolvimento, operações e segurança onde os participantes não estavam cientes de toda a operação, ou até mesmo, quais novas features estariam disponíveis nesta nova versão. Será que houve uma atualização de versão de alguma das bibliotecas utilizadas pelo software? Esse novo software foi testado? Com as práticas e ferramentas de DevOps todo esse processo pode ser automatizado e um deploy de uma nova versão pode ser disponibilizado de maneira automática.

Monitoramento e Feedback: O DevOps acredita que o monitoramento contínuo de aplicativos e infraestrutura é vital para identificar problemas e fornecer feedback às equipes de desenvolvimento e operações.

Black Friday chegando, a aplicação atualmente está consumindo quando de memória e processamento dos servidores em dias regulares de acesso? A infraestrutura atual irá suportar a demanda prevista para os feriados de fim de ano? Como a aplicação está se comportando em um dia de alta demanda, algum time-out? Foram perdidas vendas? São questões importantes e somente com o monitoramento constante da aplicação elas poderão ser respondidas e possíveis problemas poderão ser antecipados.

Infrastructure as Code (IaC): A automação e a replicabilidade são permitidas pelo IaC, que trata a infraestrutura, como redes e servidores, da mesma forma que o código-fonte. Os ambientes DevOps usam muitas ferramentas IaC, como Ansible, Chef e Puppet. Já falamos um pouco sobre IaC e será o primeiro conceito estudado neste bootcamp.

Cultura de Melhoria Contínua: O DevOps ajuda as equipes a aprender e melhorar continuamente, identificando áreas de melhoria e implementando mudanças gradualmente. Monitoramento e Feedback, lembra?

A implementação bem-sucedida do DevOps pode resultar em entregas mais rápidas, melhor qualidade do software, menor taxa de falhas, melhor colaboração entre equipes e maior capacidade de atender às necessidades dos clientes. Na indústria de software, essa abordagem é comumente usada para lidar com problemas de desenvolvimento e operações em um ambiente ágil e dinâmico.

Desenvolvimento moderno de software

Para podermos aplicar melhor o DevOps é importante entender como as aplicações são desenvolvidas nos dias atuais. Se você possuir os cabelos brancos, assim como eu, você deve lembrar que a internet um dia não existiu (ou pelo menos, não esteja presente no nosso dia-a-dia) e as aplicações funcionam localmente e estavam disponíveis em um único computador, ou no melhor dos cenários a aplicação estava disponível nos computadores presentes em uma rede local utilizando computadores chamados de mainframes.
A partir dos anos 90 a arquitetura cliente-servidor mudou o cenário do desenvolvimento. Vamos entender como funciona a arquitetura cliente-servidor.

A arquitetura cliente-servidor é um modelo de design de software que divide as funções de um sistema entre dois tipos de entidades diferentes: clientes e servidores. Sistemas de rede geralmente usam essa arquitetura, que permite que clientes e servidores se comuniquem por meio da troca de requisições e respostas.

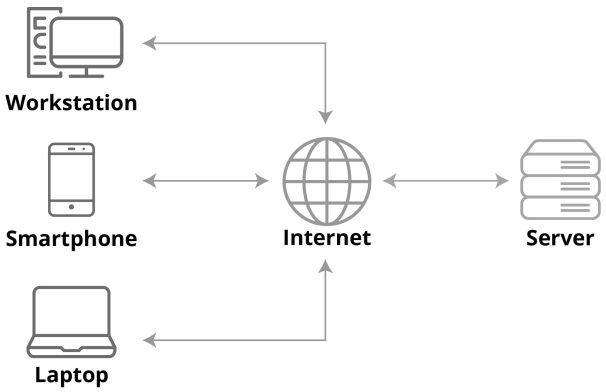

Figura 1 - Representação do modelo cliente-servidor

Vamos a um exemplo prático: Quais são as formas nos dias de hoje de você consultar o seu saldo bancário? Você pode consultar pelo aplicativo no banco no seu dispositivo móvel, pelo navegador de internet no seu computador pessoal, no caixa eletrônico (ATM), por telefone ou diretamente no caixa do banco. Isso quer dizer que o seu saldo bancário não está salvo no seu dispositivo bancário ou no caixa eletrônico (ATM), essa informação deve estar centralizada para que cada um dos dispositivos citados tenha acesso a essa informação.

Isso quer dizer que o seu saldo bancário está salvo em disco em um servidor em algum lugar do mundo, e sempre que um dos dispositivos precisam desta informação é neste ambiente centralizado que a informação é disponibilizada para cada um dos dispositivos requerentes da informação.

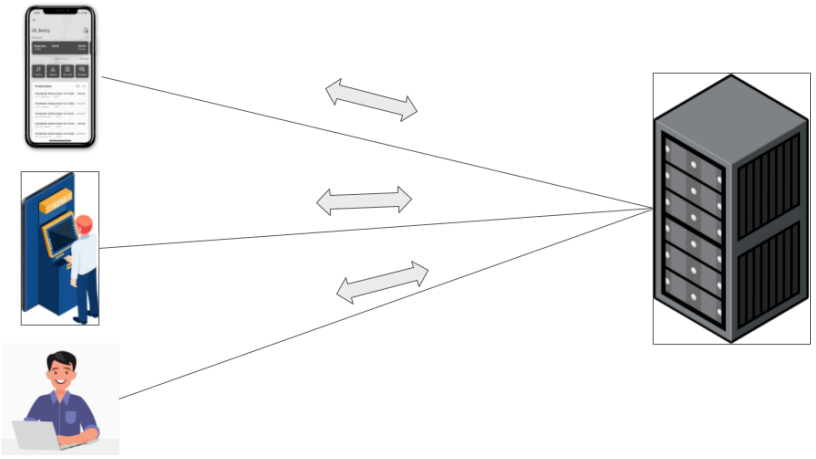

Figura 2 - Servidor, tenho dinheiro para ir ao cinema?

As características principais da arquitetura cliente-servidor:

Os clientes são as entidades que solicitam recursos ou serviços dos servidores. Eles podem ser usuários, dispositivos ou aplicativos. Os clientes iniciam a comunicação informando os servidores.

Os servidores são as entidades que respondem às solicitações dos clientes. Eles fornecem aos clientes os serviços, recursos ou dados que eles solicitam. Os servidores estão sempre atualizados para atender às solicitações dos clientes.

Os clientes e os servidores interagem normalmente através de uma rede usando protocolos comuns, como HTTP para a web ou TCP/IP em redes mais comuns.

Na arquitetura cliente-servidor, os clientes e os servidores são responsáveis pelos seus próprios trabalhos. Os servidores são responsáveis pela lógica comercial, processamento de dados e armazenamento, enquanto os clientes são responsáveis pela apresentação e interação do usuário.

Tanto nos clientes quanto nos servidores existem softwares específicos para cada uma das suas respectivas funções. O software presente nos clientes são chamados de front-end de uma aplicação e o software presente nos servidor são chamados de back-end de uma aplicação.

Também existe um software responsável pela comunicação entre o front-end e o back-end, esse software é chamado de API (Application Programming Interface). Uma API é um

conjunto de padrões que permitem que vários softwares interajam entre si. Essa interface permite que vários componentes do software se comuniquem e interajam com informações.

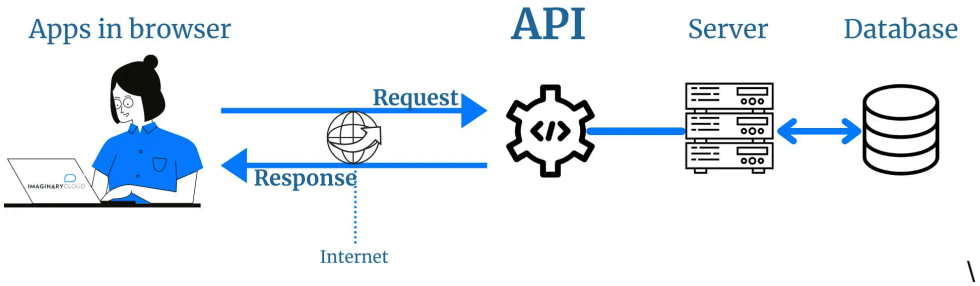

Figura 3 - Uma API no futebol seria os jogadores do meio de campo

Uma API é um conjunto de regras que permitem que um software (ou parte dele) interaja com outro. Essa interação pode incluir solicitações de dados, operações de sistema ou até mesmo a integração de funcionalidades particulares entre dois aplicativos.

Infraestrutura local versus Cloud

OK, eu entendi que o back-end de uma aplicação está disponível em um servidor, agora onde este servidor pode estar?
A resposta é: depende!

Uma empresa pode arcar toda a infraestrutura de servidores e demais equipamentos de rede para hospedar o seu software organizando os servidores em data center local. A principal vantagem de ter um data center local é que a organização tem controle direto sobre todas as facetas do ambiente, incluindo infraestrutura e segurança. Os servidores locais geralmente estão fisicamente próximos aos usuários e às operações da empresa, o que significa que eles podem melhorar o desempenho e a latência. Agora, tudo isso tem um custo elevado, a construção e manutenção de um data center local pode levar a custos iniciais significativos, incluindo hardware, energia, refrigeração e espaço físico.

Um data center local pode enfrentar interrupções de energia, aumentando o risco de tempo de inatividade não planejada, o que pode aumentar os custos de operação caso a empresa opte também em adquirir gerados para suprir este problema.

O termo "cloud" se refere à computação em nuvem, que é um modelo de distribuição de serviços de computação por meio da Internet. Os usuários podem acessar e usar recursos computacionais como servidores, armazenamento, bancos de dados, redes, software e análise por meio da Internet, em vez de depender de servidores locais ou infraestrutura física. A grande vantagem de utilização dos serviços em nuvem oferecidos pela Amazon Web Services (AWS), Google Cloud Platform (GCP), Oracle Cloud entre outros são os serviços gerenciados, permitindo que as equipes de TI se concentrem mais nas estratégias e menos na manutenção de infraestrutura, outra vantagem é o modelo de pagamento por

uso, permitindo que as organizações paguem apenas pelos recursos que realmente utilizam, o que pode ser mais econômico do que investir em infraestrutura local.

Migrar para a nuvem oferece diversas vantagens para organizações em termos de flexibilidade, escalabilidade, eficiência operacional, segurança e colaboração e dependendo da infraestrutura necessária o custo de operação pode ser infinitamente menor do que manter um data center local.

Durante o Bootcamp iremos realizar atividades simulando um data center local e também iremos utilizar serviços na nuvem utilizando as plataformas AWS e GCP.

O que é um Deploy (Deployment)?

Durante o Bootcamp você irá se deparar diversas vezes com essa palavra. Mas você sabe o que é um Deploy? Em desenvolvimento de software, "deploy" (ou implantação) é o processo de colocar uma aplicação ou uma nova versão dela em um ambiente operacional, seja um servidor, um ambiente de teste ou até mesmo a produção, para que os usuários finais possam acessar. Como é uma etapa final do ciclo de vida do desenvolvimento de software, o lançamento é quando o código desenvolvido é colocado em operação e torna-se funcional.

A implantação consiste em várias etapas, como:

Compilação e empacotamento: O código-fonte é compilado e empacotado para que possa ser implantado no ambiente desejado.

Testes: Antes de ser usado em um ambiente de produção, o código geralmente passa por testes severos para garantir que não contenha erros e funcione corretamente. Os testes de sistema, integração e unitários podem fazer parte disso.

Configuração do Ambiente: Para que a aplicação funcione, o ambiente de destino deve ser configurado. A configuração de servidores, bancos de dados e variáveis de ambiente podem fazer parte disso.

Implementação: Os recursos e o código necessários são copiados para o ambiente de destino. É possível fazer isso manualmente ou automaticamente usando ferramentas de automação de deploy.

A implementação pode ser feita de várias maneiras, dependendo da metodologia de desenvolvimento, das práticas da equipe e do tipo de aplicação.

Para garantir que as alterações de software cheguem aos usuários finais sem interromper significativamente os serviços, o deploy bem-sucedido é essencial. As equipes de desenvolvimento podem realizar implantações eficientes e confiáveis com a ajuda de ferramentas de automação, práticas DevOps e estratégias de deploy apropriadas.

Antes das práticas de DevOps era comum problemas acontecerem durante o deploy. Se um problema importante acontecer, a aplicação poderá ficar indisponível podendo gerar prejuízos para a empresa e em muitos casos uma força tarefa é necessária para identificar o problema e restabelecer a aplicação. É por isso que o deploy em uma sexta-feira era evitado (é muito provável que você já tenha visto algum meme sobre isso).

Figura 4 - nada melhor do que um Deploy antes do Happy-hour

O deploy é o coração das práticas de DevOps, adotar práticas DevOps não apenas resolve problemas específicos associados ao deploy, mas também promove uma cultura de melhoria contínua e colaboração, tornando todo o ciclo de vida do desenvolvimento mais eficiente e resiliente.

Quais problemas o DevOps irá solucionar em um Deploy?

O DevOps está intimamente ligado à implantação de aplicações, pois visa integrar o desenvolvimento de software com as operações de TI para otimizar e agilizar o ciclo de vida do desenvolvimento de software e entrega. A fase crucial do ciclo é o deployment, onde o DevOps trabalha para melhorar a cooperação das equipes de desenvolvimento e operações para garantir implantações mais confiáveis e eficientes. Existem várias maneiras pelas quais o DevOps está envolvido no deployment de aplicativos:

A integração contínua (CI): O DevOps é responsável pela implementação de práticas de integração contínua, que incluem a integração rotineira de alterações de código no

repositório compartilhado. Esse método é útil para identificar e resolver problemas de integração no início do processo de desenvolvimento.

Serviço de entrega contínua (CD): A responsabilidade do DevOps vai além da Integração Contínua para a Entrega Contínua, que é quando o software é automaticamente desenvolvido, testado e preparado para uso na produção ou pré-produção.

Automatização: O DevOps tem como objetivo principal automatizar o máximo possível o processo de deployment. A automação da construção, o teste e a implantação do software fazem parte disso.

Pipelines de deployment automatizados que reduzem erros humanos e aceleram o tempo de deployment são geralmente criados usando ferramentas de automação, como Jenkins, GitLab CI, ou GitHub Actions.

A automação também é usada para construir e administrar ambientes de produção, reduzindo as chances de diferenças entre ambientes e garantindo consistência.
A resposta imediata:

Rollback e Reversão: Para resolver problemas durante o deployment, o DevOps deve ter estratégias eficazes de rollback (reverter para uma versão anterior) e roll forward (avançar para uma nova versão).
Isso garante que as instalações possam ser restauradas rapidamente em caso de problema, reduzindo assim o impacto na produção.

Por fim, o DevOps está intimamente ligado ao deployment de aplicações porque trabalha para integrar o desenvolvimento e as operações para otimizar todo o ciclo de vida do software, desde a criação até a entrega e operação. Essa estratégia permite que as equipes realizem implantações mais rápidas, confiáveis e eficientes.

Conhecendo o primeiro problema a ser resolvido

O time de desenvolvimento de software finalizou um website utilizando as linguagens HTML, CSS e JavaScript. O código e demais e demais recursos do website como imagens e sons estão disponíveis em um repositório Git. Um repositório do tipo Git armazena, gerencia e monitora as alterações em um projeto de software. O Git é um sistema de controle de versão distribuído, portanto, cada desenvolvedor que clona um repositório Git tem uma cópia integral de todo o histórico de revisões e pode trabalhar independentemente, mesmo offline.

Os desenvolvedores podem executar uma variedade de tarefas com um repositório Git, incluindo commit (gravar alterações), branch (criar uma nova linha de desenvolvimento), merge (combinar várias linhas de desenvolvimento) e revert (desfazer alterações). A

colaboração eficaz entre desenvolvedores é facilitada pela natureza distribuída do Git, que facilita a fusão de alterações e o gerenciamento de várias versões do código.

Para hospedar repositórios Git de forma remota, plataformas populares como GitHub, GitLab e Bitbucket facilitam o trabalho colaborativo, o gerenciamento de projetos e o rastreamento de problemas. Além disso, essas plataformas oferecem recursos adicionais, como ferramentas de colaboração, integração contínua e controle de acesso.

As ferramentas Git desempenham um papel vital no DevOps, fornecendo uma base sólida para a colaboração eficiente, rastreamento de alterações, automação de processos e garantia de qualidade ao longo do ciclo de vida do desenvolvimento de software.

O repositório do website está disponível no seguinte endereço:

[https://github.com/denilsonbonatti/linux-site-dio]

Este website somente estará disponível dentro de uma infraestrutura local, desta forma, o website será publicado em um servidor local dentro da empresa. Para execução deste website que utiliza as linguagens HTML, CSS e JavaScript é necessário que o servidor possua um software para fornecer a execução do website, para isso precisamos criar um servidor web utilizando softwares como o Apache ou NGINX. Neste exemplo, iremos utilizar o Apache. Logicamente o software Apache deve ser executado em um sistema operacional, neste caso iremos utilizar o Linux Ubuntu 20.04 LTS.

OK! Esta é a infraestrutura básica que precisamos para que o website esteja disponível, neste exemplo, o data center local que será utilizado será o seu computador. Vamos entender um pouco melhor o que será feito.

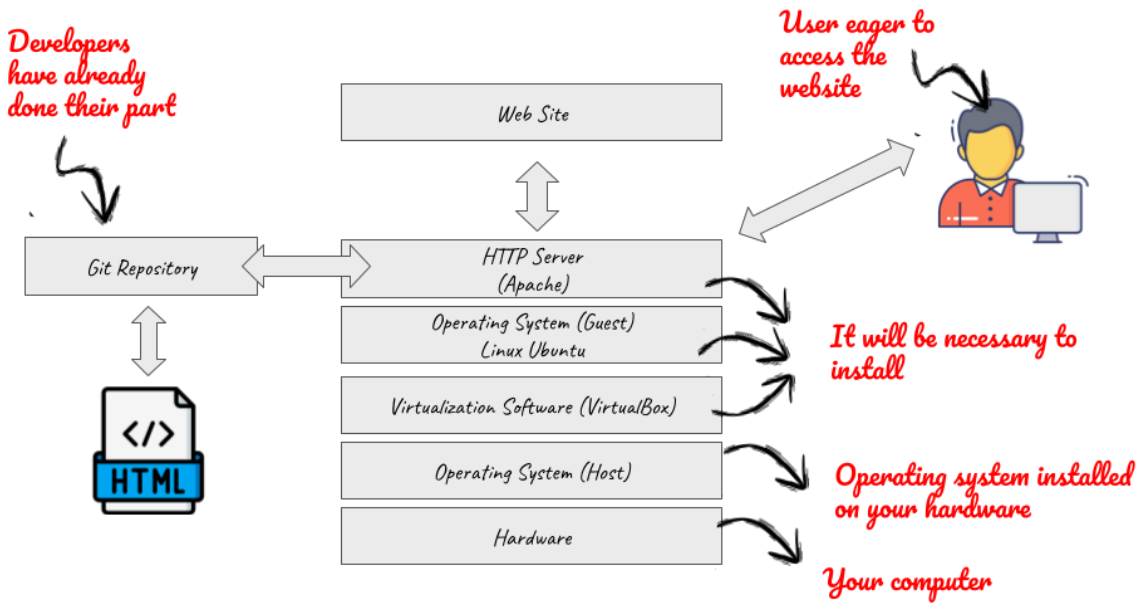

Figura 5 - Pronto! Agora é só fazer!

O Hardware que irá simular um data center local, você já possui, ou pelo menos eu acredito que você tenha acesso a um computador para realizar este bootcamp. Este computador já possui um sistema operacional instalado, neste caso, não importa se é Windows, Linux ou MacOS. O que precisamos é de um software de virtualização, criar uma máquina virtual, instalar o sistema operacional convidado, que será um Linux Ubuntu, instalar o Apache e clonar o repositório, desta forma, teremos o website disponível aos usuários do data center local. Agora como fazer isso utilizando as práticas de DevOps? Meu Deus, o que é uma máquina virtual? Aonde eu encontro esses softwares? Essas e outras perguntas serão respondidas no próximo capítulo. Vejo você lá!

Pulando de cabeça

Infraestrutura como código (IaC)

No capítulo anterior você conheceu qual o problema que temos que resolver utilizando DevOps. É claro que não iremos utilizar todos os recursos do DevOps neste primeiro problema, iremos focar inicialmente em aprender os principais conceitos de infraestrutura como código e como ela poderá nos ajudar a automatizar a publicação de uma página web inicialmente em um servidor local e futuramente em um servidor público utilizando a Amazon Web Services.
Você viu no capítulo anterior que parte do problema já está resolvido, o data center local será o seu computador e o sistema operacional host será o sistema operacional instalado em seu computador. O próximo passo é aprender a compartilhar recursos do seu seu computador com o servidor que iremos utilizar para publicar a página web, pois esse servidor irá utilizar possivelmente um sistema operacional diferente do seu.
Para que possamos criar um novo servidor iremos utilizar recursos de virtualização criando um máquina virtual onde o página web será executada. Todo esse processo de criação da máquina virtual, instalação do software necessários será feita através de código.

A abordagem de Infraestrutura como Código (IaC) emprega código em vez de processos manuais ou interfaces gráficas para gerenciar e provisionar a infraestrutura de TI. Isso significa que a infraestrutura pretendida é implementada e gerenciada por meio de ferramentas de automação, após sua descrição de código. A Infraestrutura como Código é uma prática essencial no contexto de DevOps, permitindo uma automação eficiente e uma gestão mais eficaz da infraestrutura de TI.

O que é uma máquina virtual?

A virtualização é uma tecnologia que permite que recursos de hardware, como computadores, servidores, armazenamento e redes, criem ambientes virtuais ou instâncias virtuais. A utilização dessa abordagem permite que vários sistemas operacionais ou aplicativos funcionem de forma independente e isolada em uma única máquina física conhecida como "host".

Virtualização de máquina é um tipo de virtualização, onde é possível construir máquinas virtuais completas que funcionam como sistemas independentes e têm seu próprio sistema operacional. O VMware, o Microsoft Hyper-V e o VirtualBox são alguns exemplos de hypervisors, que são softwares destinados a gerenciar máquinas virtuais.

A virtualização é fundamental para muitas infraestruturas contemporâneas, pois ajuda a otimizar os recursos, aumentar a flexibilidade e facilitar a administração de ambientes de TI. Neste Bootcamp iremos utilizar o VirtualBox como software de virtualização, pois é um software democrático, ou seja, existem versões do VirtualBox para Windows, Linux e MacOs. Caso você tenha domínio e já utilize outro software de virtualização como VMWare ou Hyper-V você poderá utilizá-los realizando pequenas adequações.

Para baixar o arquivo de instalação do VirtualBox utilize a seguinte URL:

[https://www.virtualbox.org]

Após baixar o arquivo de instalação, execute o arquivo e realize a instalação do VirtualBox.

Introdução ao Vagrant

O Vagrant é uma ferramenta criada pela Hashicorp (mesma empresa proprietária do Terraform), de código aberto projetada para simplificar a configuração e a gestão de ambientes de desenvolvimento e em alguns casos até mesmo ambiente de produção. A ferramenta permite que os desenvolvedores configurem ambientes de desenvolvimento e produção de maneira consistente e replicável, automatizando o processo de criação e provisionamento de máquinas virtuais. Como veremos, com o Vagrant é possível criar uma máquina virtual por código não precisando se preocupar com processos manuais e repetitivos.

Baixe o arquivo de instalação do Vagrant disponível no link abaixo. Existem versões executáveis do arquivo de instalação para os sistemas operacionais Windows e MacOS.

[https://developer.hashicorp.com/vagrant/install]

Para os sistemas operacionais Linux, alguns comandos devem ser executados para configurar o repositório da Hashicorp e realizar a instalação. No Exemplo abaixo temos os comandos para realizar a instalação do Vagrant em um sistema operacional Linux Ubuntu ou Debian. Para as demais distribuições, consulte o link acima.

```
$ wget -O- https://apt.releases.hashicorp.com/gpg | sudo gpg --dearmor -o
/usr/share/keyrings/hashicorp-archive-keyring.gpg

$  echo "deb [signed-by=/usr/share/keyrings/hashicorp-archive-keyring.gpg]
https://apt.releases.hashicorp.com $(lsb_release -cs) main" | sudo tee
/etc/apt/sources.list.d/hashicorp.list
```

```
$ sudo apt update && sudo apt install vagrant
```

Após realizar a instalação do Vagrant, para verificar se a instalação foi realizada com sucesso, utilize o comando abaixo no terminal ou no powershell, caso esteja utilizando o sistema operacional Windows.

```
$ vagrant --version
```

Caso a instalação tenha sido realizada com sucesso a versão atual do Vagrant será exibida.

Para organizar os arquivos que serão criados, crie uma nova pasta. Abra o terminal ou o powershell a partir desta nova pasta.

Certifique-se que a pasta recém criada esteja sendo exibida no prompt de comando. Assim, você terá certeza que os arquivos serão criados dentro desta nova pasta.

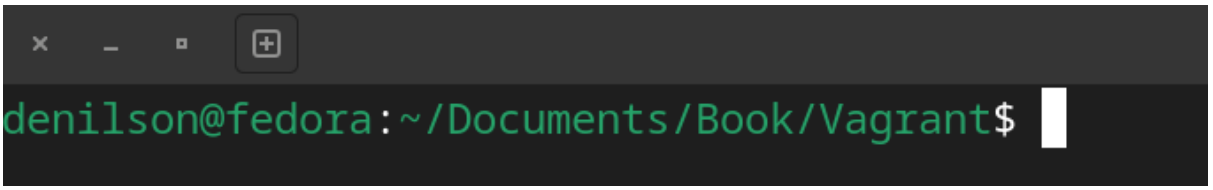

Figura 6 - Opa! Estou no local certo

OK! Com o Vagrant instalado e a pasta dos arquivos de configuração criada, o próximo passo é criar a máquina virtual a partir de um arquivo de configuração. Como iremos trabalhar com IaC nenhuma interação do usuário deverá ser feita no processo de criação da máquina virtual e das demais configurações.

Criando uma máquina virtual

É aconselhável reduzir ou evitar a interação com o usuário quando se trabalha com uma infraestrutura como código (IaC). Para garantir consistência, replicabilidade e previsibilidade nos ambientes de implantação, o processo deve ser totalmente automatizado. A ausência de intervenção manual reduz a possibilidade de erros, acelera o provisionamento e promove a confiabilidade, permitindo a criação de ambientes de maneira eficiente e padronizada. É exatamente isso que iremos fazer criando um arquivo de configuração do provisionamento da máquina virtual.

Vamos criar o arquivo que será utilizado como base para a criação da nova máquina virtual. Digite o abaixo para criar o arquivo inicial de configuração.

```
$ vagrant init
$ ls -l
```

O comando ls -l irá exibir um novo arquivo chamado Vagrantfile. o arquivo de configuração Vagrantfile é usado pelo Vagrant para configurar as configurações e características de uma máquina virtual (VM) específica. Ele é escrito em Ruby e inclui instruções para o Vagrant para criar e provisionar uma máquina virtual e configurar vários componentes do ambiente de desenvolvimento.
O próximo passo é abrir este arquivo para realizar as devidas configurações. Digite o comando abaixo para abrir o Microsoft Vscode.

```
$ code .
```

Observe que o Vscode será aberto. Clique no arquivo Vagrantfile para abri-lo.

A primeira configuração que iremos realizar é a escolha do sistema operacional da nova máquina virtual. Os sistemas operacionais que podem ser utilizados com o Vagrant são armazenados em formato de Boxes. Para consultar os Boxes disponíveis, acesse o site:

[https://app.vangrantup.com]

Os Boxes disponíveis estão organizados por provedores, ou seja, existem boxes para VMware, boxes para Hyper-v, VirtualBox etc.
Como estamos utilizando o VirtualBox como provedor, selecione-o:

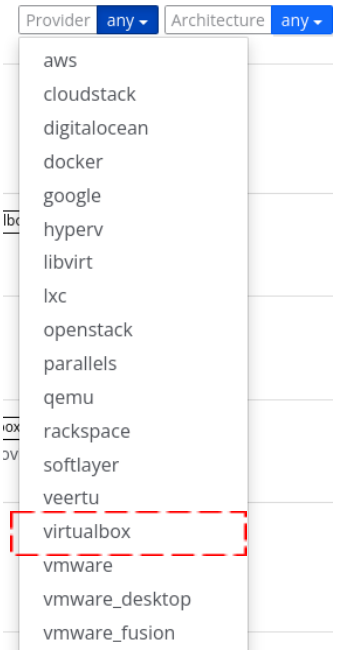

Figura 7 - Achei!!

Na barra de busca, digite o termo "Ubuntu 20".

Enquanto este material é editado o box com a versão oficial do Ubuntu mais recente é a 20.04 LTS.

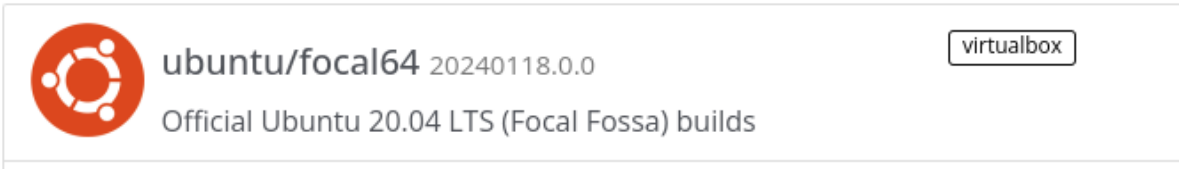

Figura 8 - Box do Linux Ubuntu 22.04 LTS para Virtualbox

Clique sobre o box do ubuntu/focal64.

Observe que um código de como utilizar este box no vagrantfile é exibido.
Selecione o nome do box, conforme indicado na imagem abaixo:

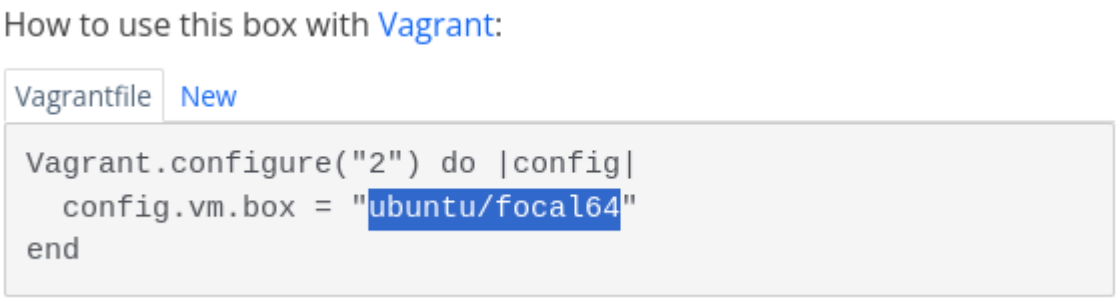

Figura 9 - Selecione o nome do box

Volte ao Vscode e na linha 15 altere a linha de comando conforme indicado a seguir:

```
config.vm.box = "ubuntu/focal64"
```

Esta máquina virtual precisará estar acessível, ou seja, ela precisará estar aberta a requisições externas, desta forma, poderemos acessar a página web que futuramente iremos configurar neste servidor.

Para isso é necessário adicionar uma nova placa de rede em modo bridge. Remova o carácter "#" da linha 40 para que a nova placa de rede em modo bridge seja adicionada a máquina virtual.

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

    # Display the VirtualBox GUI when booting the machine
      vb.gui = false

    # Customize the amount of memory on the VM:
    vb.memory = "1024"
end
```

Somente deixe o caractere de comentário (#) nas linhas indicadas no código acima. Observe que estamos deixando o caractere # somente nos comentários do código. No código foi alterado o valor da propriedade vb.gui para **false**. Desta forma, o modo gráfico de criação da máquina virtual será desabilitado. Estamos fazendo isso para que a máquina virtual seja criada de maneira automática, sem nenhuma interação com o usuário.

O próximo passo é definir o nome da máquina virtual, quantidade de memória e processamento.
No exemplo acima a máquina virtual irá possuir 1GB de memória RAM. Isso é feito pelo parâmetro **vb.memory**. Observe que o valor de 1024 mb foi atribuído ao parâmetro. Caso deseje uma máquina virtual com mais memória RAM altere esse valor para o desejado.
Por padrão a máquina virtual será criada com apenas um núcleo de processamento. No exemplo a seguir iremos adicionar mais dois parâmetros. O parâmetro **vb.cpus** para indicar a quantidade de núcleos de processamento, neste exemplo será alterado para 2. Também será utilizado o parâmetro **vb.name**, onde será indicado o nome da máquina virtual.

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

     # Display the VirtualBox GUI when booting the machine
       vb.gui = false

     # Customize the amount of memory on the VM:
     vb.memory = "1024"
     vb.cpus = "2"
     vb.name = "Ubuntu - Web Server"
end
```

Outro atributo importante é especificar o tempo que o Vagrant irá aguardar para que o boot do novo sistema operacional seja finalizado. Aumentar esse tempo é importante em máquinas hospedeiras mais lentas. Vamos aumentar esse tempo para 1000 segundos. Isso é feito pelo parâmetro **config.vm.boot_timeout**. Ele deve ser adicionado no local indicado no código a seguir:

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

     # Display the VirtualBox GUI when booting the machine
       vb.gui = false

     # Customize the amount of memory on the VM:
     vb.memory = "1024"
     vb.cpus = "2"
     vb.name = "Ubuntu - Web Server"
end

config.vm.boot_timeout = 1000
```

OK! Vamos testar o nosso código antes de inserirmos as demais configurações que serão utilizadas para a instalação do Apache e configuração da página web.

Salve as configurações feitas no arquivo Vagrantfile. Aconselho você a utilizar a opção "Auto Save" disponível no menu "File".

No terminal ou powershell utilize o comando a seguir para criar a nova máquina virtual:

```
$ vagrant up
```

Aguarde a máquina virtual ser criada. Observe que no ambiente gráfico do VirtualBox a máquina virtual já está disponível e observe que ela foi criada com as configurações indicadas no Vagrantfile.

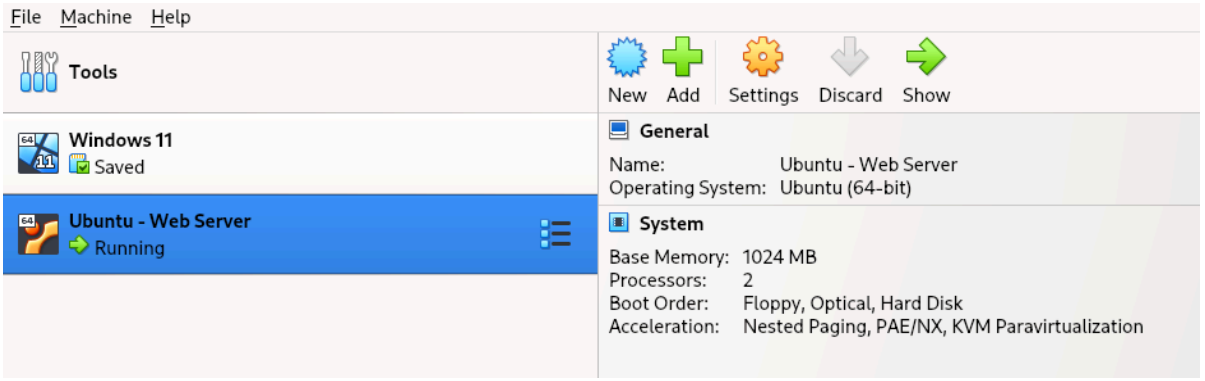

Figura 10 - IaC, eu te amo!

Não iremos realizar o acesso a máquina virtual via ambiente gráfico do virtualbox, mas sim diretamente do terminal ou powershell. Digite o comando a seguir:

```
$ vagrant ssh
```

Observe pelo prompt de comando que agora estamos utilizando o Linux Ubuntu recém instalado.

```
vagrant@ubuntu-focal:~$
```

OK! Teste realizado com sucesso, mas não é esse o resultado final que queremos. Assim que a máquina virtual for criada iremos atualizar o sistema operacional, realizar a instalação do Apache e configurar a página web desejada. Sendo assim, vamos apagar provisoriamente a máquina virtual criada. Digite o comando exit para sair do acesso remoto e o comando vagrant destroy, como indicado a seguir, para excluir a máquina virtual.

```
vagrant@ubuntu-focal:~$ exit
$ vagrant destroy --force
```

Observe que a máquina virtual foi excluída.

Criando um script para automação

Um conjunto de comandos ou instruções escritos em uma linguagem de programação interpretada é chamado de script. O uso de scripts é comumente feito para automatizar tarefas ou realizar operações específicas em sistemas computacionais. Eles não passam por um processo de compilação antes de serem executados, o que os distingue dos programas compilados. Em vez disso, um interpretador de script lê o código fonte e o executa diretamente.

Os Scripts são normalmente utilizados em tarefas repetitivas ou complexas , eles podem executar comandos, configurar o sistema e manipular arquivos, entre outras tarefas. Os scripts permitem que os usuários alterem ou estendam as funcionalidades de acordo com suas necessidades, alterando o comportamento de programas e sistemas. Os scripts podem ser escritos em em Python, JavaScript, Shell Script, Ruby e PowerShell, entre outras linguagens de programação. Neste caso específico iremos executar comandos a partir de shell do Linux, desta forma, iremos utilizar comandos do Linux em um Shell Script.

Abra o Vscode e abra novamente o arquivo Vagrantfile.

Remova o caractere "#" da linha 78 e realize a seguinte alteração no código:

```
config.vm.provision "shell", path: "script.sh"
```

Essa alteração no código irá determinar que o script a ser executado, após o boot do sistema operacional na máquina virtual, estará presente em um arquivo chamado **script.sh**.

Crie um novo arquivo com o nome de script.sh.

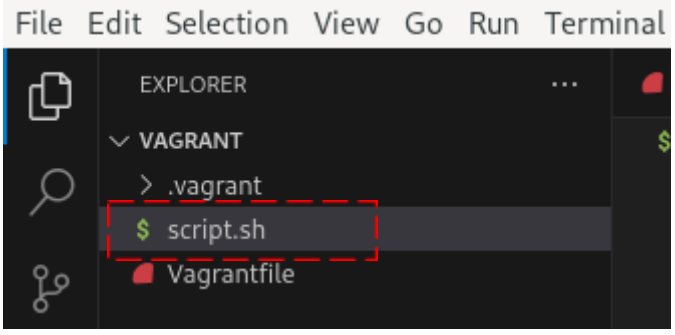

Figura 11 - Um novo e fresquinho arquivo de script

Neste novo arquivo, por boas práticas, deve ser iniciado por **#!/bin/bash**. Isso informa ao sistema que o script deve ser executado usando o interpretador de comandos Bash.
A seguir vamos utilizar os comandos para atualizar o sistema operacional. Vale lembrar que estamos utilizando a distribuição Linux, desta forma, vamos utilizar o comando apt-get. Como a atualização do sistema necessita de privilégios de administrador para tal, vamos utilizar o comando **sudo**. Através do comando sudo, você proporciona privilégios de nível administrativo para usuários comuns. O Vagrant cria um usuário padrão com o nome de vagrant para realizar o acesso remoto e executar o script.

Digite os comandos a seguir no novo arquivo de script.

```
#!/bin/bash

sudo apt-get update
sudo apt-get upgrade -y
```

Observe que a opção -y foi utilizado no comando apt-get upgrade, é importante utilizá-la para que a atualização seja feita automaticamente não pedindo informações ao usuário.

OK! Com o sistema atualizado o próximo passo é instalar o Apache. Atualize o script com o comando a seguir.

```
#!/bin/bash

sudo apt-get update
sudo apt-get upgrade -y

sudo apt-get install apache2 -y
```

Com o Apache instalado, o próximo passo é clonar o repositório do código fonte da página Web. Para isso iremos utilizar o comando git clone. Por padrão, em sistemas Linux as ferramentas Git já estão instaladas e prontas para o uso, desta forma, não há a necessidade de realizar a instalação. Lembre-se que o código fonte da página web está disponível no endereço *[https://github.com/denilsonbonatti/linux-site-dio]*

Abra essa URL. Copie o endereço do repositório.

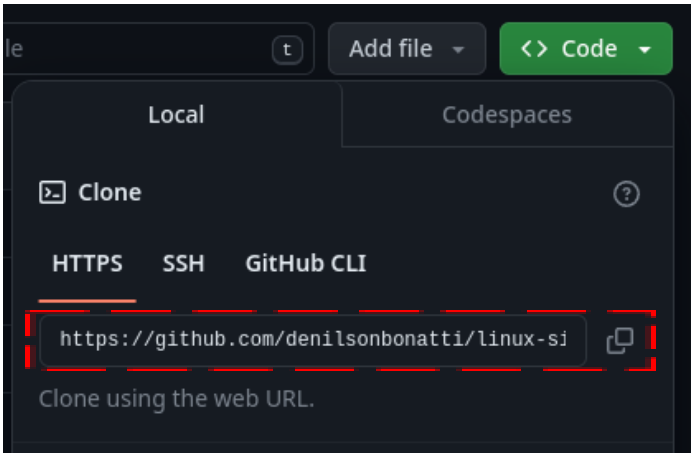

Figura 12 - CTRL+C e CTRL+V é vida!!

Utilize o comando git clone e cole o endereço do repositório copiado, como indicado no código a seguir:

```
#!/bin/bash

sudo apt-get update
sudo apt-get upgrade -y

sudo apt-get install apache2 -y

git clone https://github.com/denilsonbonatti/linux-site-dio.git
```

Ao clonar o repositório uma pasta com o nome **linux-site-dio** será criada. O próximo passo é copiar todos os arquivos presentes na pasta criada para o diretório padrão do Apache. No Linux Ubuntu o diretório padrão do Apache é **/var/www/html.** No script iremos utilizar o comando cp para copiar todos os arquivos e subdiretórios presentes dentro da pasta [nome] para o diretório padrão do Apache. Para que esta cópia seja realizada é necessário permissão de administrador, desta forma, o comando sudo precisará ser utilizado.

Adicione o comando cp, conforme indicado a seguir no arquivo script.sh:

```
#!/bin/bash

sudo apt-get update
sudo apt-get upgrade -y

sudo apt-get install apache2 -y

git clone https://github.com/denilsonbonatti/linux-site-dio.git

sudo cp linux-site-dio/* -R /var/www/html
```

A seguir precisamos saber qual é o endereço ip da placa de rede em modo bridge para que possamos acessar o servidor web através de uma browser de navegação via nossa máquina física, ou seja, acessar a máquina virtual (guest) a partir da nossa máquina host (que é a nossa máquina física). Para saber as definições de ips da placas de rede utilize o comando **ip a**.

Adicione o comando ip a, conforme indicado a seguir:

```
#!/bin/bash

sudo apt-get update
sudo apt-get upgrade -y

sudo apt-get install apache2 -y

git clone https://github.com/denilsonbonatti/linux-site-dio.git

sudo cp linux-site-dio/* -R /var/www/html

ip a
```

O script está finalizado, se você desejar, é possível exibir mensagens ao usuário enquanto o script é executado, assim o usuário saberá exatamente o que está acontecendo enquanto cada comando ou bloco de comandos é executado. Essa mensagem pode ser exibida ao usuário através do comando **echo**. Observe o exemplo utilizado a seguir:

```
#!/bin/bash

echo "Atualizando o sistema…"

sudo apt-get update
sudo apt-get upgrade -y

echo "Instalando o Apache…"

sudo apt-get install apache2 -y

echo "Clonando o repositório…"

git clone https://github.com/denilsonbonatti/linux-site-dio.git

sudo cp linux-site-dio/* -R /var/www/html

echo "Exibindo o IP…"

ip a
```

O próximo passo é criar novamente a máquina virtual através do comando vagrant up.

```
$ vagrant up
```

Observe que a máquina virtual será criada, e nesse processo a execução do script será realizada. Observe que as mensagens ao usuários criadas pelo comando echo são exibidas.

Fique atento à exibição do ip da segunda placa de rede, que normalmente é numerada como **default 3** e tem o nome de **enp0s8.**

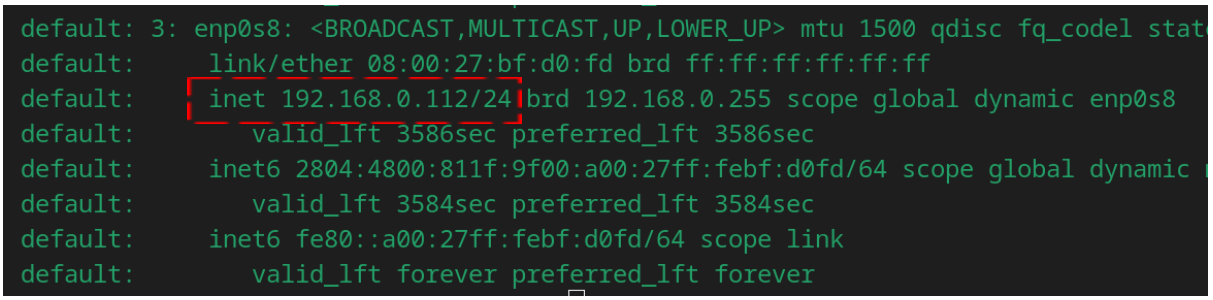

Figura 13 - Achei!!!

É claro que este ip muito provavelmente vai ser diferente do meu, isso vai depender das configurações de ip da sua rede física.

> Copie ou anote este endereço IP. Abra o seu Browser preferido (instalado na sua máquina física) e digite ou cole o endereço ip. Adicione o prefixo http://

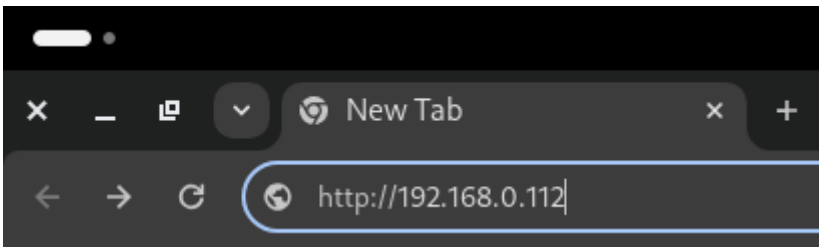

Figura 14 - Eu copiei e colei porque tenho preguiça

Se o script foi criado de maneira correta o website será aberto. Essa é a magia da IaC, em poucos segundos conseguimos publicar um website. Esse código agora pode ser replicado e subir uma máquina virtual exatamente como essa em qualquer lugar do mundo.

Se você precisar parar a máquina virtual sem excluí-la utilize o comando a seguir:

```
$ vagrant halt
```

A máquina virtual será desligada e caso precise reiniciá-la utilize novamente o comando vagrant up.
Como a máquina foi reiniciada é importante saber que provavelmente um novo ip deve ter sido configurado na segunda placa de rede. Para consultar esse novo ip utilize o comando a seguir:

```
$ vagrant ssh -c "ip a"
```

O comando ip a será executado através do ssh e os endereços IPs configurados serão exibidos.

Para suspender a máquina virtual sem desligá-la utilize o comando a seguir:

```
$ vagrant suspend
```

Para utilizar novamente a máquina virtual utilize o comando a seguir:

```
$ vagrant resume
```

Em caso de dúvida e para mais opções, consulte a ajuda do Vagrant:

```
$ vagrant --help
```

Esses são alguns comandos importantes, existem muitos outros que podem ser utilizados para facilitar o processo de criação e replicação de máquinas virtuais dos quais não iremos focar neste bootcamp, mas aconselho uma consulta as formas de criar boxes de máquinas virtuais e como replicá-las. Consulte o link a seguir para mais informações:
[https://developer.hashicorp.com/vagrant/docs/boxes/base]

Isso é importante para que se possa distribuir o box para outras pessoas ou para uso em outros ambientes, desta forma não será necessário compartilhar o código de criação de um determinado ambiente, mas será possível compartilhar o ambiente já criado através do box diretamente.

E SE EU PRECISAR DE MAIS DE UMA MÁQUINA VIRTUAL?

É possível você criar mais de uma máquina virtual utilizando o Vagrant. Isso pode ser feito através de um vetor no Vagrantfile onde iremos especificar as configurações de cada uma das máquinas virtuais. Vamos criar um exemplo onde três novas máquinas virtuais serão criadas.

Para facilitar o processo vamos clonar um repositório existente. Abra a pasta onde a pasta do repositório a ser clonado será criada.
Abra o terminal ou o powershell a partir desta nova pasta.

Certifique-se que a pasta recém criada esteja sendo exibida no prompt de comando. Assim, você terá certeza que os arquivos serão criados dentro desta nova pasta. Iremos utilizar o

comando git para clonar o repositório em nossa máquina física, se você estiver utilizando o sistema operacional Windows, certifique-se que você tenha as ferramentas Git instaladas, caso não tenha realize a instalação a partir do link a seguir:

[https://git-scm.com/downloads]

O próximo passo é abrir a URL do repositório a ser clonado. Abra a URL a seguir:

[https://github.com/denilsonbonatti/vagrant-multi-vm]

Copie o endereço do repositório. No terminal ou powershell digite o comando a seguir:

```
$ git clone https://github.com/denilsonbonatti/vagrant-multi-vm.git
```

Abra o diretório criado e execute o Vscode.

```
$ cd vagrant-multi-vm/
$ code .
```

Neste VagrantFile temos uma variável do tipo vetor com o nome de **machines** onde é possível verificar os nomes da máquina virtuais e suas configurações. Neste exemplo, temos o nome master, node01 e node02. A próxima configuração é a quantidade de memória. Neste exemplo temos 1GB para cada máquina virtual. A Seguir temos a quantidade de núcleos de CPUs de cada máquina virtual. Na configuração de ip é indicado o endereço do host, ou seja, caso a sua rede seja 192.168.0.0 e o endereço ip do host seja indicado **50**, a máquina virtual irá possuir o endereço 192.168.0.**50**. Vale saber que nesta configuração as máquinas virtuais irão possuir apenas uma placa de rede em modo bridge. Essa máquina virtual deverá estar na mesma rede da sua máquina física, como no exemplo acima, 192.168.0.0. Fique atento à entrega dos ips do host para que o ip entregue não entre em conflito com algum ip já utilizado na sua rede local, como outro computador, celular, TV etc.

A seguir é indicado qual o BOX utilizado para cada máquina virtual. Caso você deseje, é possível utilizar um sistema operacional diferente para cada máquina virtual.

```
machines = {
"master" => {"memory" => "1024", "cpu" => "1", "ip" => "50", "image" =>
"ubuntu/focal64"},
"node01" => {"memory" => "1024", "cpu" => "1", "ip" => "51", "image" =>
"ubuntu/focal64"},
"node02" => {"memory" => "1024", "cpu" => "1", "ip" => "52", "image" =>
"ubuntu/focal64"}
}
```

A configuração de ip da sua rede local é definida na seguinte linha de código:

```
machine.vm.network "public_network", ip: "192.168.0.#{conf["ip"]}"
```

> Altere, se necessário, o valor de ip para o endereço da sua rede física da sua casa, escritório ou empresa. Para consultar o endereço ip da sua rede utilize o comando **ip a** no Linux ou MacOS e no Windows utilize o comando **ipconfig**.

Na linha de comando a seguir é indicado qual script será executado por **todas** as máquinas virtuais.

```
machine.vm.provision "shell", path: "update.sh"
```

Nas próximas linhas é indicado qual script será executado exclusivamente pela máquina virtual com o nome de master e pelas demais máquinas virtuais.

```
if "#{name}" == "master"
        machine.vm.provision "shell", path: "master.sh"
else
        machine.vm.provision "shell", path: "worker.sh"
end
```

O próximo passo é criar os arquivos de script que serão executados.

> Crie um novo arquivo com o nome de update.sh. Lembrando que este script será executado por todas as máquinas virtuais.
>
> Digite os comandos a seguir no novo arquivo de script. Observe que iremos atualizar o sistema operacional de todas as máquinas virtuais.

```
#!/bin/bash

sudo apt-get update
sudo apt-get upgrade -y
```

OK! Agora vamos criar o arquivo de script que será executado exclusivamente pela máquina virtual com o nome de **master**.

Crie um arquivo com o nome de **master.sh**. Somente como exemplo, esse script irá criar um novo arquivo com o nome de master.txt. Após a criação das máquinas virtuais iremos verificar se o arquivo foi criado constatando a execução correta do script.

```
#!/bin/bash

touch master.txt
```

O próximo passo é criar o arquivo de script que será executado pelas demais máquinas virtuais.

Crie um arquivo com o nome de **worker.sh**. Somente como exemplo, esse script irá criar um novo arquivo com o nome de worker.txt. Após a criação das máquinas virtuais iremos verificar se o arquivo foi criado constatando a execução correta do script.

```
#!/bin/bash

touch worker.txt
```

Vamos criar as máquinas virtuais. Utilize o comando vagrant up, conforme indicado a seguir:

```
$ vagrant up
```

Após a execução do script, observe que as máquinas virtuais serão criadas:

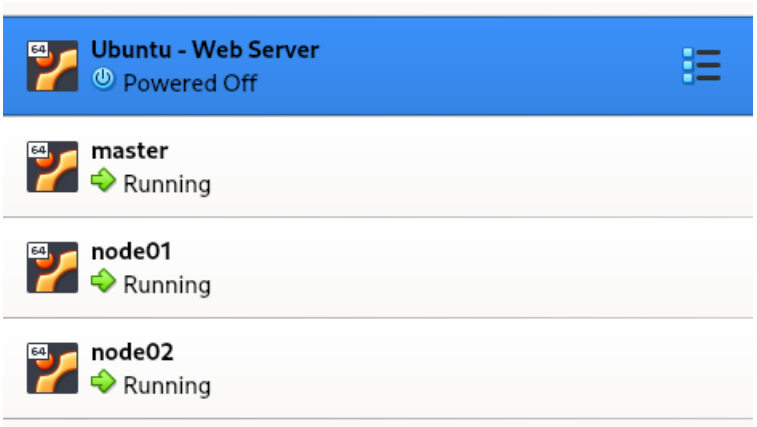

Figura 15 - Três é melhor que uma

Para realizar o acesso remoto a uma máquina específica, o seu nome deve ser indicado no comando vagrant ssh. Por exemplo, vamos realizar o acesso remoto à máquina virtual master e verificar se o arquivo master.txt foi criado.

Digite os comando a seguir:

```
$ vagrant ssh master
vagrant@master:~$ ls -l
```

Observe que o comando master.txt foi criado. Para sair do acesso remoto utilize o comando exit. Se desejar, faça o acesso remoto às demais máquinas e verifique se os scripts foram executados corretamente. Para excluir as máquinas virtuais o comando vagrant destroy --force pode ser utilizado.

OK! Agora que você já conheceu um pouco mais sobre IaC em uma infraestrutura local utilizando o Vagrant, iremos mudar de software, agora iremos utilizar o Terraform.

Enquanto o Terraform é uma ferramenta de automação de infraestrutura que se concentra em nuvem e gerenciamento de recursos distribuídos, o Vagrant é mais focado em ambientes locais e desenvolvedores. Muitas vezes, as duas ferramentas podem trabalhar juntas. Por exemplo, o Vagrant cria ambientes locais de desenvolvimento e o Terraform gerencia a infraestrutura na nuvem para ambientes de produção.