**FACULDADE DE TECNOLOGIA DE AMERICANA “MINISTRO RALPH BIASI”**

**Curso Superior de Tecnologia em Segurança da Informação**

# DEPLOYMENT DE APLICAÇÃO PYTHON EM CONTAINERS DOCKER UTILIZANDO O SERVIÇO EC2 DA AWS



| **Elaborador:** | Christian Pereira da Costa de Oliveira Barreto |
| --- | --- |
| **Orientador:** | Henri Alves Godoy |



**FACULDADE DE TECNOLOGIA DE AMERICANA**

**FICHA CATALOGRÁFICA – Biblioteca Fatec Americana - CEETEPS**
**Dados Internacionais de Catalogação-na-fonte**

B261d BARRETO, Christian Pereira da Costa de Oliveira

Deployment de aplicação python em containers docker utilizando o serviço EC2 da AWS. / Christian Pereira da Costa de Oliveira Barreto – Americana, 2021.

38f.

Relatório técnico (Curso Superior de Tecnologia em Segurança da Informação) - - Faculdade de Tecnologia de Americana – Centro Estadual de Educação Tecnológica Paula Souza

Orientador: Prof. Ms. Henry Alves Godoy

1 Segurança em sistemas de informação I. GODOY, Henry Alves II. Centro Estadual de Educação Tecnológica Paula Souza – Faculdade de Tecnologia de Americana

CDU: 681.518.5

Centro Paula Souza

SÃO PAULO GOVERNO DO ESTADO

Secretaria de Desenvolvimento Econômico

---

**CHRISTIAN PEREIRA DA COSTA DE OLIVEIRA BARRETO**

# DEPLOYMENT DE APLICAÇÃO PYTHON EM CONTAINERS DOCKER UTILIZANDO O SERVIÇO EC2 DA AWS

Trabalho de graduação apresentado como exigência para obtenção do título de Tecnólogo em Segurança da Informação pelo CEETEPS/Faculdade de Tecnologia - FATEC/ Americana.
Área de concentração: Segurança da Informação

Americana, 14 de junho de 2021.

**Banca Examinadora:**

_______________________________

Henri Alves de Godoy

Mestre

Fatec Americana

_______________________________

Maxwel Vitorino da Silva

Mestre

Fatec Americana

_______________________________

Wellington Aires da Cruz Pereira

Mestre

Fatec Americana

**FACULDADE DE TECNOLOGIA DE AMERICANA**

# SUMÁRIO

**1 OBJETIVOS DESTE DOCUMENTO ........................................................ 5**
**2 OS DIFERENTES MODELOS DE INFRAESTRUTURA ................................ 7**
2.1 O MODELO TRADICIONAL DE INFRAESTRUTURA .................................... 7
2.2 VIRTUALIZAÇÃO DE SERVIDORES ........................................................ 8
2.3 CONTAINERS ........................................................................... 10
*2.3.1 Containers versus Máquinas Virtuais ........................................ 10*
*2.3.2 Vantagens do Docker ............................................................ 12*
**3 DEMONSTRAÇÂO PRÁTCA ..................................................... 14**
3.1 TECNOLOGIAS UTILIZADAS E PREPARAÇÃO DO AMBIENTE ......................... 15
*3.1.1 Python ............................................................................. 15*
*3.1.2 Docker ............................................................................. 17*
*3.1.3 Amazon Web Services ........................................................... 20*
3.2 EXECUÇÃO ............................................................................ 22
*3.2.1 Código Python .................................................................... 22*
*3.2.2 Dockerfile e Docker Compose .................................................. 24*
*3.2.3 Criação da Instância EC2 ....................................................... 27*
**4 RESULTADOS ........................................................................ 33**
**5 CONCLUSÕES E CONSIDERAÇÕES FINAIS .......................................... 35**

## LISTA DE FIGURAS

Figura 1 – Ambiente sem Virtualização 8
Figura 2 – Representação do funcionamento de um hypervisor 9
Figura 3 – Ambiente utilizando Virtualização 9
Figura 4 – Comparação entre Virtualização e *Containers* 11
Figura 5 – "Na minha máquina funciona" 12
Figura 6 – Comandos para atualizar e verificar a versão do Python 16
Figura 7 – Instalação e utilização do pip 16
Figura 8 – Comandos para instalar o Docker no Ubuntu 20.04 18
Figura 9 – Comandos para instalação do Docker Compose 18
Figura 10 – Comando para instalação do Docker Machine 19
Figura 11 – Comando para instalação da AWS CLI 20
Figura 12 – Painel de Gerenciamento de Credenciais AWS 21
Figura 13 – Chaves de acesso geradas 21
Figura 14 – Configuração das chaves de acesso 22
Figura 15 – Código Python do arquivo index.py 23
Figura 16 – Conteúdo do Dockerfile 25
Figura 17 – Conteúdo do arquivo requirements.txt 26
Figura 18 – Conteúdo do arquivo docker-compose.yml 26
Figura 19 – Estrutura do diretório 27
Figura 20 – Execução do Docker Machine 28
Figura 21 – Grupo de Segurança Docker Machine 28
Figura 22 – Comando para ativar a instância EC2 via Docker Machine 29
Figura 23 – Construir o container a partir do arquivo docker-compose.yml 29
Figura 24 – Editar regras de entrada da instância EC2 30
Figura 25 – Regra de entrada TCP personalizado 31
Figura 26 – Resumo da instância 31
Figura 27 – Mensagem exibida em produção 31

# 1 OBJETIVOS DESTE DOCUMENTO

O modelo de infraestrutura de Tecnologia da Informação (TI) bem como as ferramentas e tecnologias utilizadas sofreram diversas mudanças ao longo do tempo, desde os primeiros computadores utilizados comercialmente, passando por robustos Centros de Processamentos de Dados (*Data Centers*) com equipamentos adquiridos através de investimentos da própria empresa (o modelo chamado de *on premise*), até chegar recentemente à popularização do uso de tecnologias de Nuvem, seja ela Privada — quando a infraestrutura é construída pela própria organização para servir à organização — ou Pública — quando os serviços são contratados de provedores de terceiros.

Essa transição entre o investimento na própria estrutura e a contratação de força computacional de terceiros ainda não se deu por completo, pois muitas empresas já possuíam uma infraestrutura existente e em tais casos não seria um investimento inteligente descomissionar servidores e bancos de dados em perfeito funcionamento apenas para se adaptar as tendências do mercado.

Apesar disso muitas empresas passaram a utilizar os serviços de nuvem em alguns projetos, para hospedar novas aplicações construídas com tecnologias modernas. Essa utilização gradual também se deve ao crescente uso de novas metodologias de desenvolvimento de software, tais como Microsserviços e DevOps, que visam tornar o processo de desenvolvimento de software mais eficiente e orgânico, otimizando a integração entre software e infraestrutura, mudando a forma como aplicações são desenvolvidas, publicadas e gerenciadas.

Diante dessa mudança no modo como aplicações são hospedadas e publicadas, uma das tecnologias que ganhou forte tração nos últimos anos foi o Docker, que busca obter o máximo de aproveitamento dos recursos da máquina na qual está operando, consumindo o mínimo de recursos possível.

O presente relatório tem como objetivo analisar as aplicações e vantagens do uso de *containers* Docker em relação aos métodos tradicionais de *deployment* de

aplicações, através de uma breve análise teórica e a demonstração de seu uso em uma situação real.

Inicialmente serão descritas as arquiteturas dos principais modelos de infraestrutura utilizados atualmente. Em seguida será exposta a teoria sobre o funcionamento dos *containers*, seus recursos, funcionamento, principais aspectos e vantagens em relação ao uso somente de máquinas virtuais convencionais. Por fim será descrito o processo para a publicação de uma aplicação Python que utiliza *containers* Docker no serviço de nuvem pública Amazon Web Services (AWS).

# 2 OS DIFERENTES MODELOS DE INFRAESTRUTURA

A evolução das tecnologias de infraestrutura mudou a forma como as organizações planejam, mantém e gerenciam suas aplicações e operações, tornando imprescindível mudanças que ao longo do tempo pudessem otimizar sua capacidade operacional e computacional. Três das principais grandes etapas que ocorreram nesse cenário foram: servidores físicos próprios, virtualização e recentemente a conteinerização.

A seguir serão tratadas as principais características, aplicações e implicações do uso de cada um desses modelos.

## 2.1 O MODELO TRADICIONAL DE INFRAESTRUTURA

O modelo convencional, também chamado de *on premise*, consiste na utilização de servidores físicos controlados e operados pelas próprias empresas, muitas vezes alocados dentro de suas próprias instalações em grandes salas repletas de servidores e equipamentos, os chamados Data Centers (CISCO, s.d).

Ao manter os próprios servidores as empresas devem considerar todos os custos operacionais, adequações necessárias à legislação, investimento e planejamento para controle de incêndios, segurança patrimonial, controle de acesso, manutenção da rede, backups periódicos, locação física para as instalações, plano de contingência e recuperação de desastres, entre outros aspectos que devem ser levados em conta no planejamento e manutenção de infraestrutura e que demandam grande investimento financeiro (MICROSOFT, s.d).

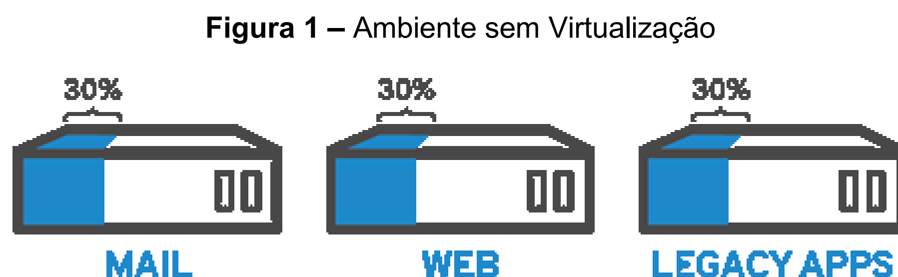
Apesar disso, tal investimento vale cada centavo, pois ao utilizar esse modelo a organização pode seguramente ter controle total sobre as informações processadas e armazenadas. Por se tratar de sua própria infraestrutura torna-se mais fácil garantir

integridade, disponibilidade e confidencialidade de seus dados, sem correr o risco de transmitir informações externamente, além de ser capaz a otimizar e adaptar seu cenário para ter total aderência às necessidades da própria organização.

## 2.2 VIRTUALIZAÇÃO DE SERVIDORES

Ambientes *on premise* modernos em sua grande maioria também se beneficiam da virtualização, através da utilização de *Virtual Machines* (Máquinas Virtuais ou simplesmente VM's) para aproveitar de maneira mais eficiente o *hardware* dos servidores, evitando tanto o desperdício de recursos quanto o desperdício financeiro. A Figura 1 exemplifica um ambiente não virtualizado onde há uma má alocação de recursos para as diferentes aplicações.

**Figura 1 –** Ambiente sem Virtualização




Fonte: Redhat

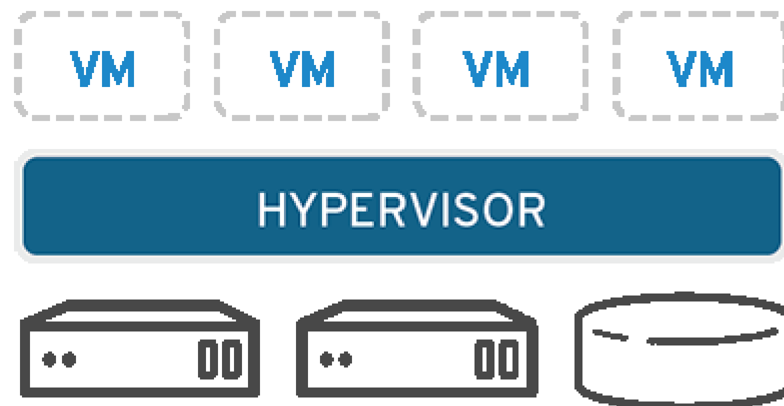
A Oracle nos fornece uma definição bastante simples e clara de virtualização como sendo "a capacidade de executar múltiplas máquinas virtuais em um único hardware". Outra gigante da tecnologia, a RedHat, nos fornece uma definição igualmente simples que ressalta o aspecto de serviços de TI definindo virtualização como "uma tecnologia que permite criar serviços de TI valiosos usando recursos que tradicionalmente estão vinculados a um determinado *hardware*".

A possibilidade de execução de múltiplas máquinas virtuais em um único *hardware* se dá graças a utilização de um Monitor da Máquina Virtual (VMM, na sigla em inglês) que se trata um *software* chamado de *Hypervisor,* responsável pela criação, execução e gerenciamento das VM's (REDHAT).

O *hypervisor* permite que um *hardware* físico — o *host* — ofereça suporte para várias máquinas virtualizadas que consomem seus recursos — *guests* — compartilhando virtualmente memória e processamento (VMWARE).

Normalmente a virtualização é composta pela camada da infraestrutura física (o servidor), o *hypervisor*, que controla e gerencia os recursos consumidos por cada VM e as máquinas virtuais hospedadas rodando individualmente, conforme pode ser visto no esquema da Figura 2.

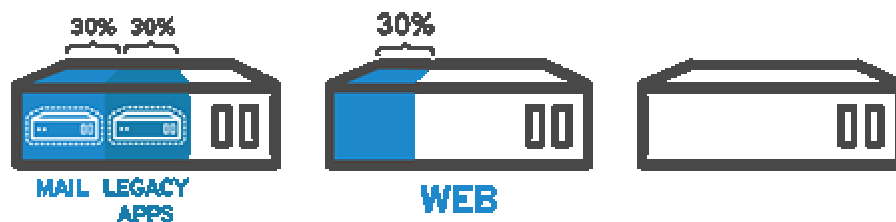
**Figura 2 –** Representação do funcionamento de um hypervisor




Fonte: Redhat

Ao virtualizar além da redução no consumo de recursos computacionais é possível obter um melhor isolamento das aplicações e recursos hospedados em um mesmo *hardware* garantindo que os recursos da máquina física serão distribuídos proporcionalmente para manter as instâncias hospedadas sendo executadas adequadamente. Na Figura 3 temos o exemplo de um ambiente que se aproveita da virtualização para obter um aproveitamento de recursos mais adequado.

**Figura 3 –** Ambiente utilizando Virtualização




Fonte: Redhat

Apesar das vantagens trazidas pela virtualização, outras tecnologias foram desenvolvidas para auxiliar na otimização de recursos, como é o caso dos *containers*, ferramentas similares às máquinas virtuais, mas que consomem ainda menos recursos e possibilitam um melhor aproveitamento dos recursos disponíveis.

## 2.3 CONTAINERS

Modularizar múltiplos ambientes virtuais em um único sistema operacional já era algo existente em distribuições GNU/Linux, como é o caso do *chroot jail*. Entretanto foi o FreeBSD que influenciou o desenvolvimento de *containers* como conhecemos hoje, com o *Jails* nos anos 2000 (REDHAT). Segundo a definição da RedHat o Jails é “uma tecnologia que permite particionar um sistema FreeBSD em vários subsistemas ou celas (por isso o nome *jails*)”.

### 2.3.1 *Containers versus* Máquinas Virtuais

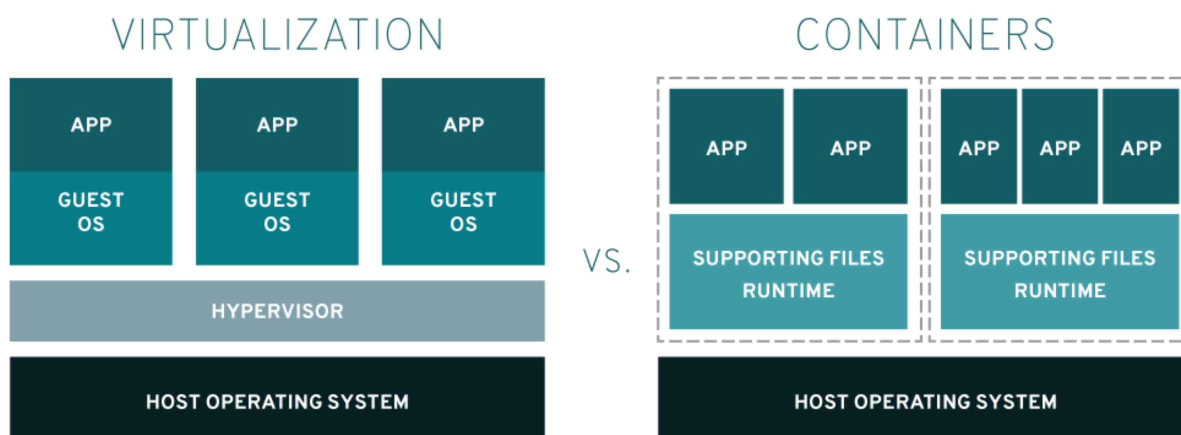
Ao contrário do que se pode crer o objetivo da utilização dos *containers* não é competir ou substituir as máquinas virtuais, trata-se apenas de uma forma de organizar e modularizar as aplicações de modo que elas possam ter um mesmo ambiente padrão isolado independentemente da plataforma (Sistema Operacional ou

Distribuição) na qual estão operando. *Containers* e máquinas virtuais são tecnologias diferentes em suas propostas, motivações e arquiteturas, entretanto ambas as tecnologias são complementares uma à outra.

Conforme citado anteriormente, ao utilizar a virtualização reduz-se o desperdício de recursos, especialmente de *hardware*, possibilitando a execução de diferentes sistemas e aplicações no mesmo servidor.

Já os *containers* "compartilham o mesmo kernel do sistema operacional e isolam os processos da aplicação do restante do sistema" (REDHAT). Na virtualização o *hypervisor* emula um *hardware* físico e executa o sistema operacional, já no caso dos *containers* eles compartilham o mesmo sistema operacional de forma isolada, se comportando como se fossem quaisquer outros processos sendo executados no SO. Essa diferença de arquitetura e funcionamento pode ser vista na comparação da Figura 4.

**Figura 4 –** Comparação entre Virtualização e *Containers*




Fonte: Redhat

De forma simples e resumida a ideia geral dos *containers* é ser "um conjunto de um ou mais processos organizados isoladamente do sistema." (REDHAT) que permita que as aplicações sejam executadas "de forma rápida e confiável de um ambiente de computação para outro" (DOCKER).

Centro Paula Souza | SÃO PAULO GOVERNO DO ESTADO | Secretaria de Desenvolvimento Econômico

### 2.3.2 Vantagens do Docker

Diferentemente das VM's que estão associadas muito mais ao melhor aproveitamento de recursos de um servidor físico e portando mais próximas da eficiência da infraestrutura, em um contexto geral os containers estão mais próximos dos desenvolvedores.

Isso se deve ao fato de que uma das principais contribuições do Docker é acabar com o famoso jargão dos desenvolvedores quando afirmam "Mas na minha máquina funciona", mesmo quando o código desenvolvido carrega diversos problemas de construção, semelhante ao personagem do quadrinho na Figura 5. Dessa forma o Docker elimina a possibilidade dessa desculpa, pois se uma aplicação funciona perfeitamente no *container* rodando na máquina do desenvolvedor e esse mesmo *container* será publicado posteriormente no ambiente produtivo, em caso de problemas o programador não poderá culpar o servidor já que seu código foi concebido no mesmo ambiente padrão e apenas foi movido de um lugar para outro.

**Figura 5 –** "Na minha máquina funciona"

Fonte: Anton on software, 2021

Ao definir um *container* Docker como sendo "um pacote de *software* leve, autônomo e executável que inclui tudo o necessário para executar um aplicativo" a

Docker nos leva a concluir que essa tecnologia é uma forma de padronizar e isolar aplicações de moto leve e otimizado, similar à ideia e utilidade dos *containers* de carga no mundo real.

Portanto a utilização do Docker como solução de infraestrutura garante portabilidade — pois permite que que os ambientes criados possam ser executados em qualquer outra máquina — padronização — a possibilidade de ter um ambiente único e estável para a aplicação livre de erros causados por configurações específicas de um *host* — e leveza — uma vez que compartilha recursos com o *host* e assim é apenas um processo sendo executado.

## 3 DEMONSTRAÇÂO PRÁTICA

O processo comum de *deployment* de aplicações e as tecnologias empregadas nesse processo não precisam e nem devem necessariamente ser totalmente descartadas. O fato de haver uma forma mais rápida de realizar tal processo não torna a forma "mais lenta" ruim. Assim como o fato de existirem carros de Fórmula 1 não torna os carros populares inúteis.

A natureza da tecnologia empregada depende das prioridades e das demandas dos clientes, ou seja, se uma empresa de tecnologia presta serviços para clientes muito grandes ou com uma alta demanda de trabalho, desenvolvimento e gerenciamento de soluções, a velocidade na qual isso é executado torna-se crucial, principalmente em ambientes que utilizam metodologias ágeis e Acordos de Nível de Serviços (*Service Level Agreement* - SLA) com níveis críticos.

Essa demanda e exigência por velocidade é uma das principais razões que torna o Docker uma tecnologia tão eficiente para "equipes DevOps", uma vez que a carga de trabalho de configurações de ambiente e gerenciamento de dependências deixa de ser responsabilidade total do time de infraestrutura propriamente dito e passa a ser compartilhada entre as equipes de infraestrutura e desenvolvimento.

No *deployment* "normal", por exemplo, de uma aplicação *web*, seria necessário definir o ambiente mais adequado (GNU/Linux ou Windows) e qual distribuição tem mais aderência com a aplicação (CentOS, Ubuntu, Debian, RedHat e outros). Uma vez definido isso seria necessário definir qual *Web Server* utilizar, instalá-lo e configurá-lo.

Simplesmente utilizar *containers* não elimina as decisões que devem ser tomadas, entretanto pode ajudar a reduzi-las drasticamente e fazer com que uma aplicação seja publicada para um cliente com horas ou dias de antecedência. Sendo assim vale a pena considerar seu uso levando em consideração o ganho em produtividade em determinados casos.

## 3.1 TECNOLOGIAS UTILIZADAS E PREPARAÇÃO DO AMBIENTE

A seguir serão descritos as tecnologias utilizadas e o processo para fazer o *deployment* de uma simples aplicação Python rodando em *container* Docker e que utiliza os serviços da AWS.

O ambiente de desenvolvimento utilizado foi uma *workstation* Ubuntu 20.04, esse processo também pode ser reproduzido em máquinas Windows ou MacOS, entretanto houve preferência na utilização do GNU/Linux como sistema em função da facilidade para trabalhar com a linha de comando do Docker, no gerenciamento de pacotes do Python e na linha de comando da AWS.

### 3.1.1 Python

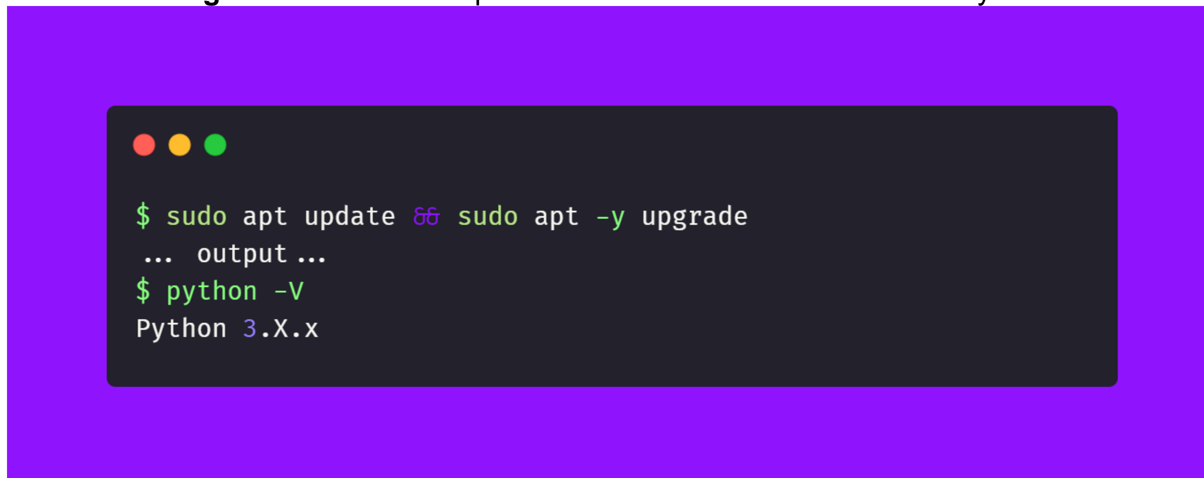
Para desenvolver a parte *web* da demonstração a tecnologia escolhida foi o Python, principalmente pela sua natureza simples, que nos permite alcançar um resultado satisfatório com pouquíssimas linhas de código.

Apesar da simplicidade, o Python possui um poderoso instalador de pacotes chamado “pip”, que nos permitiu instalar o *framework web* utilizado (o Flask) e a Interface de Linha de Comando da AWS (AWS CLI).

Por tratar-se de uma simples aplicação *web* o micro *framework* Flask nos permitirá criar uma rota e exibir uma mensagem na página quando o código for publicado.

Por padrão o Python já vem instalado no Ubuntu 20.04, entretanto por garantia é conveniente executar os comandos exibidos na Figura 6 para atualizar a lista de pacotes e verificar se a versão do Python é a adequada (versão 3 ou maior).

Figura 6 – Comandos para atualizar e verificar a versão do Python

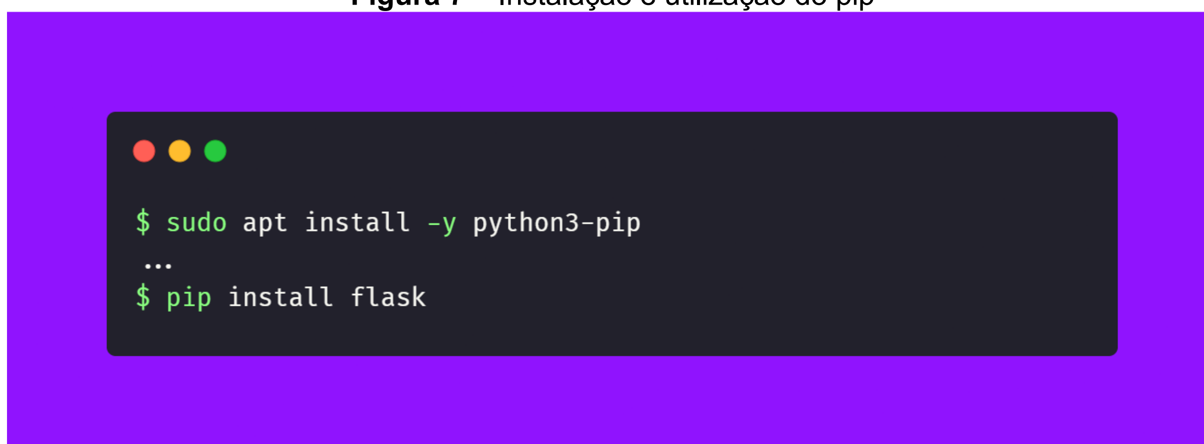


```
$ sudo apt update && sudo apt -y upgrade
... output ...
$ python -V
Python 3.X.x
```

Fonte: Próprio autor

Depois de instalar o Python podemos instalar o pip e em seguida utilizá-lo para instalar o Flask, conforme os comandos exibidos na Figura 7.

**Figura 7 –** Instalação e utilização do pip



```
$ sudo apt install -y python3-pip
...
$ pip install flask
```

Fonte: Próprio autor

Após a execução dos comandos citados a máquina está pronta para o desenvolvimento.

### 3.1.2 Docker

Quando um *container* é configurado e finalizado a aplicação dentro dele pode ser compartilhada e transportada para outros ambientes. Essa aplicação finalizada que foi desenvolvida é chamada de “imagem” e carrega todas as configurações e dependências necessárias para o funcionamento da aplicação. Portanto podemos considerar uma imagem como um projeto isolado rodando dentro de um container.

Os recursos e funcionalidade do Docker podem ser utilizados de forma imperativa, ou seja, digitando comando por comando, mas também podemos utilizar o chamado “Dockerfile”, um documento de texto que permite que as imagens Docker sejam construídas a partir dele, automatizando o processo de construção.

Além do Dockerfile também pode ser utilizado o “Docker Compose” uma ferramenta do Docker que permite automatizar e definir através de um arquivo YAML a configuração da imagem, quais portas devem ser expostas e eventualmente os scripts que devem ser executados.

Na demonstração prática foram utilizados ambos o Dockerfile e o Docker Compose para configurar e executar o container automaticamente rodando a aplicação Flask. Além disso também foi utilizado um binário chamado “Docker Machine” que posteriormente irá nos permitir executar os comandos Docker diretamente na instância da AWS a partir da máquina de desenvolvimento, sem a necessidade de navegar pela vasta (e as vezes confusa) interface da AWS.

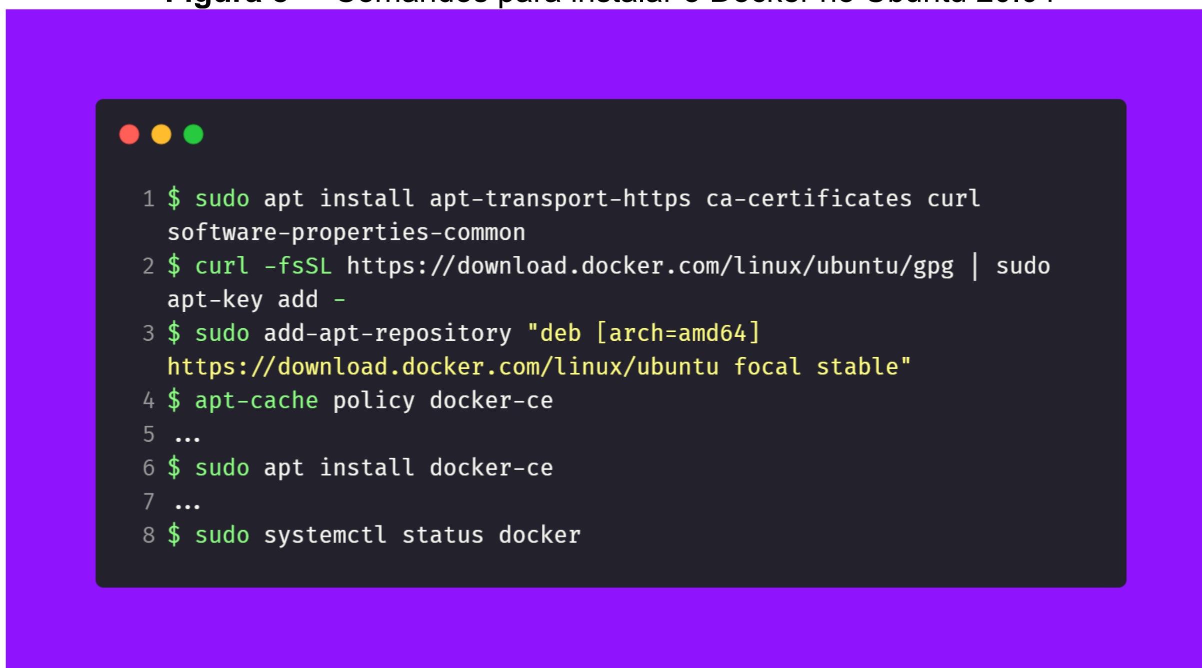
A máquina de desenvolvimento deve ter o Docker instalado para permitir a execução dessas ferramentas. Para instalação no Ubuntu foram executados os comandos exibidos na Figura 8.

Na primeira linha são instalados os pacotes que irão permitir a instalação de outros pacotes via HTTP, isso é necessário pois o Docker será instalado do repositório oficial e não do repositório padrão do gerenciador de pacotes do Ubuntu (o APT).

Na segunda linha são adicionadas as chaves GPG do repositório Docker oficial. Na terceira linha o repositório oficial é adicionado à lista de fontes do APT.

Nas linhas quatro e cinco é feita verificação do candidato de instalação e em seguida a sua instalação. Por fim na linha 8 verificamos o estado da aplicação após ser instalada utilizando o *systemd.*

**Figura 8 –** Comandos para instalar o Docker no Ubuntu 20.04




Fonte: Próprio Autor

Uma vez terminada a instalação do Docker devemos seguir para a instalação do Docker Compose, pois ele não é incluído na instalação base. Os comandos utilizados para instalação são exibidos na Figura 9.

São apenas duas linhas de comando. A primeira faz o *download* da versão mais atual e estável do Docker Compose e a segunda altera as permissões do seu binário.

**Figura 9 –** Comandos para instalação do Docker Compose

```
1 $ sudo curl -L
  "https://github.com/docker/compose/releases/download/1.29.2/docker
  -compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
2 $ sudo chmod +x /usr/local/bin/docker-compose
```

Fonte: Próprio Autor

Por fim devemos instalar o binário Docker Machine que irá facilitar a integração entre Docker e AWS CLI posteriormente. O comando para a instalação é exibido na Figura 10. O comando utilizado simplesmente baixa o binário de seu repositório no Github e o extrai no PATH.

**Figura 10 –** Comando para instalação do Docker Machine

```
1 $ base=https://github.com/docker/machine/releases/download/v0.16.0 && curl
  -L $base/docker-machine-$(uname -s)-$(uname -m)>/usr/local/bin/docker
  machine \   && chmod +x /usr/local/bin/docker-machine
```

Fonte: Próprio autor

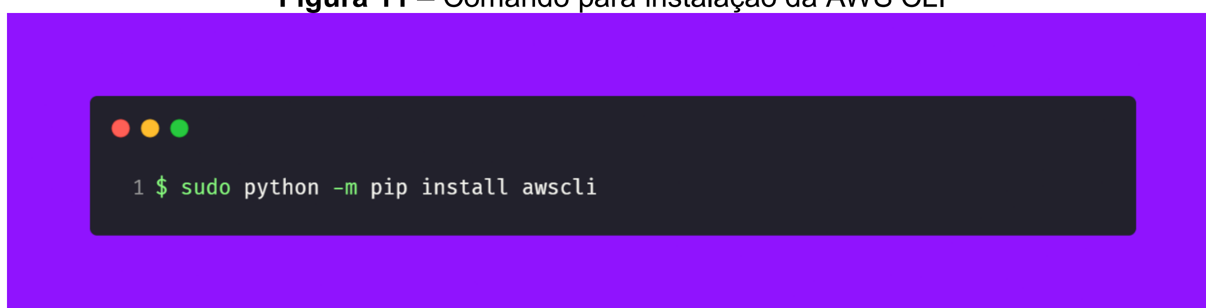
### 3.1.3 Amazon Web Services

Os serviços de computação em nuvem da Amazon possuem uma grande parcela do mercado quando se fala em nuvens públicas. Com 32% do mercado ela ultrapassa seus dois maiores concorrentes Microsoft Azure e Google Cloud que possuem respectivamente 19% e 7% (ParkMyCloud, 2021).

A elaboração da parte prática foi feita com uma conta de estudante da AWS e o serviço utilizado foi o chamado *Amazon Elastic Compute Cloud* (Amazon EC2 ou simplesmente EC2). Segundo a própria documentação o Amazon EC2 "é um serviço Web que disponibiliza capacidade computacional segura e redimensionável na nuvem" (AWS).

No caso do cenário proposto iremos utilizar uma instância EC2 para hospedar o container Docker que será criado. A interação com a instância se dará via linha de comando através da AWS CLI. A instalação da AWS CLI é feita utilizando o gerenciador de pacotes do Python. O comando utilizado para instalação é exibido na Figura 11.

**Figura 11 –** Comando para instalação da AWS CLI

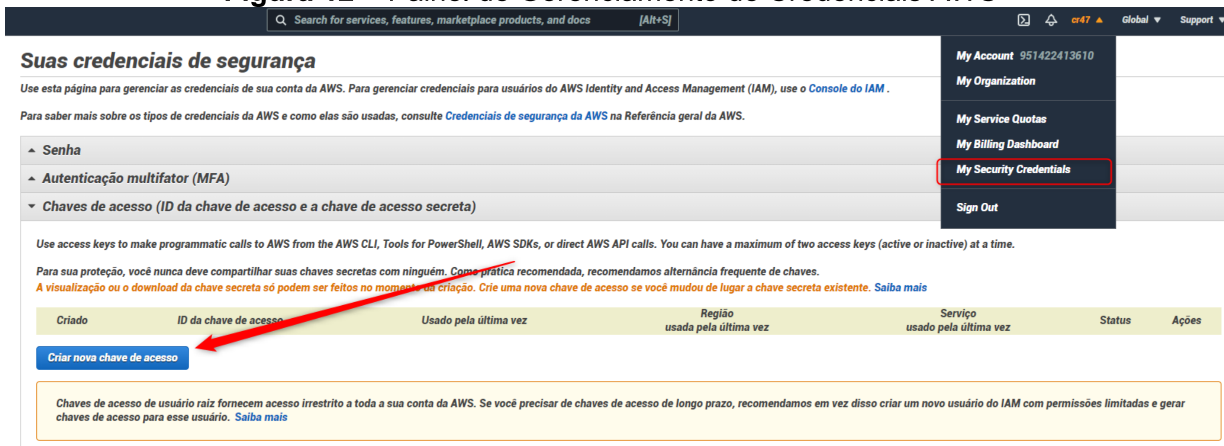



Fonte: Próprio autor

Depois de realizar a instalação da AWS CLI é necessário configurar as credenciais para nos permitir ter acesso à conta através da linha de comando. É preciso ter o ID da chave e a chave de acesso privada, além de definir a região na qual a instância EC2 deve ser criada.

Conforme a Figura 12 para obter essas credenciais deve-se acessar o conta AWS como Usuário *root*, em seguida acessar “*My Security Credentials*”. Depois clicar no botão azul escrito “Criar nova chave de acesso”.

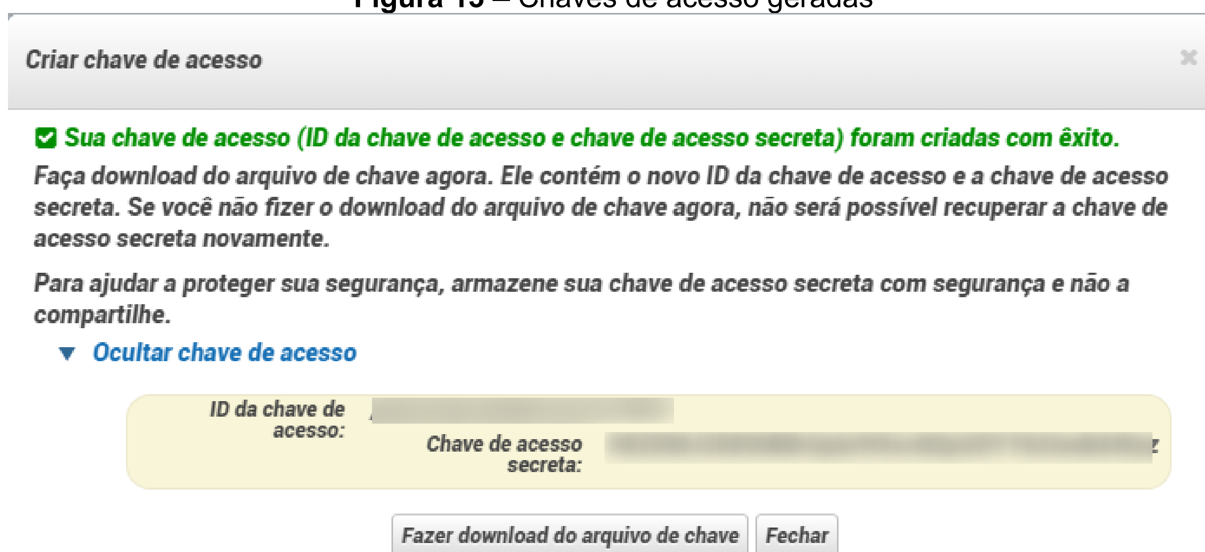
**Figura 12 –** Painel de Gerenciamento de Credenciais AWS




Fonte: Próprio autor

Após a geração os valores serão exibidos, basta copiá-los para utilizar na etapa seguinte.

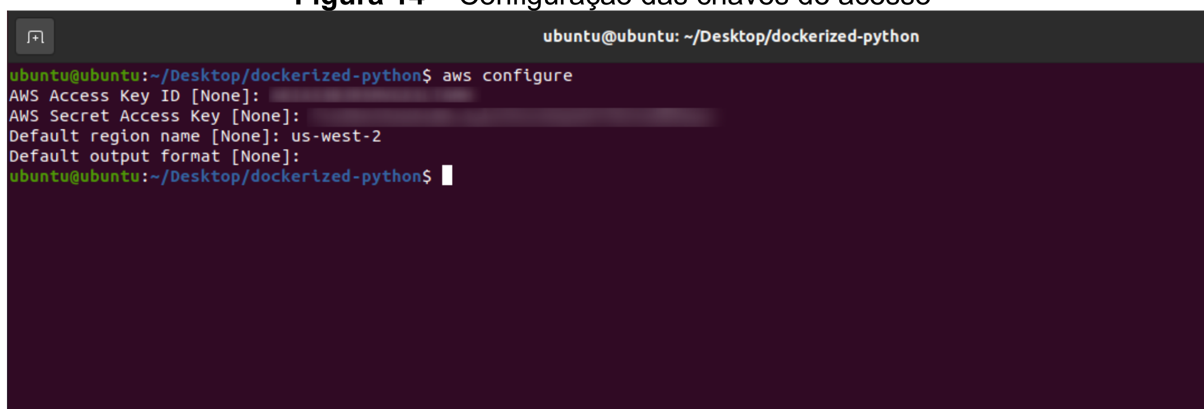
**Figura 13 –** Chaves de acesso geradas




Fonte: Próprio autor

Uma vez em posse da chave, a configuração da AWS CLI pode ser feita na máquina de desenvolvimento. Para isso, conforme a Figura 14, digitamos "aws *configure*" no terminal e adicionamos as chaves que foram geradas bem como a região na qual a instância deve ser criada.

**Figura 14 –** Configuração das chaves de acesso




Fonte: Próprio autor

## 3.2 EXECUÇÃO

Após finalizar a configuração inicial do ambiente de desenvolvimento está tudo pronto para a criação do código Python, dos arquivos que serão usados no *build* do *container* e da instância da AWS.

### 3.2.1 Código Python

O pequeno código desenvolvido em Python é exibido na Figura 15. Na primeira linha é feita a importação do *framework* Flask, instalado anteriormente e na segunda

linha é feita a importação da biblioteca de *sockets*, uma biblioteca padrão do Python. Por se tratar de uma biblioteca padrão ela não precisou ser instalada.

Na linha quatro é feita a criação de uma variável chamada HOST que recebe uma função para obter o endereço IP da máquina na qual o código está rodando.

Na sexta linha o Flask é instanciado e em seguida, na linha 7, é criada uma rota que aponta para o caminho "/", ou seja, a página inicial. Na linha 8 uma função chamada "hello" é definida, essa função retorna as *tags* HTML que exibem a mensagem quando a rota "/" for acessada.

O trecho de código na linha 14 é o que permite o programa ser executado quando, por exemplo, digitamos "python" seguido do nome do arquivo. Nesse caso o arquivo foi nomeado index.py, portando ao digitar no terminal "python index.py" ele irá executar a linha seguinte (que nesse caso específico é o código na linha 15).

Na linha 15 é definido o comportamento do código quando ele for executado através do método "app.run" que recebe o IP da máquina atual obtido através da biblioteca *socket* e a porta na qual ele deve ser exposto.

De maneira resumida o código irá exibir uma mensagem no navegador quando o endereço IP da instância for acessado através de um *browser* na porta 5000.

**Figura 15 –** Código Python do arquivo index.py

Centro Paula Souza — SÃO PAULO GOVERNO DO ESTADO — Secretaria de Desenvolvimento Econômico

```
1 from flask import Flask
2 import socket
3
4 HOST = gethostname()
5
6 app = Flask(__name__)
7 @app.route("/")
8 def hello():
9     return
10     """
11     FATEC AMERICANA\n
12     Essa é uma simples aplicação rodando em Docker
13     """
14     if __name__ == "__main__":
15     app.run(HOST, port=int("5000"))
```

Fonte: Próprio autor

### 3.2.2 Dockerfile e Docker Compose

O Dockerfile é o arquivo base que irá nos ajudar a construir nossa imagem Docker, ele se comporta de forma muito similar a um *script* bash, por exemplo. Entretanto utiliza comandos específicos do Docker.

Conforme pode ser visto na Figura 16, temos na primeira linha do Dockerfile o comando “FROM”, que determina qual imagem base iremos utilizar. Essa imagem base é baixada automaticamente do Docker Hub e estabelece a primeira camada do nosso container, é a partir dessa imagem que iremos customizar a nossa.

O valor passado indica que iremos utilizar o Python versão 3.7 e o complemento “alpine” especifica que é desejado utilizar uma imagem mínima baseada no “Alpine Linux”. As imagens Alpine são significativamente menores do que as imagens padrão

e ao utilizá-las podemos economizar um pouco mais de recursos. Portanto é uma boa prática priorizar o uso de imagens alpine sobre imagens convencionais.

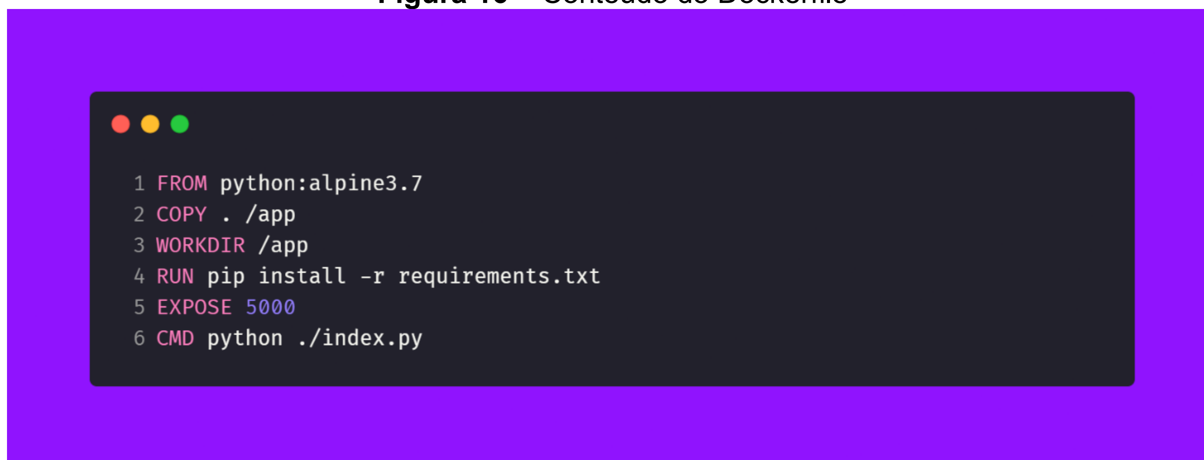
Na linha 2 o comando “COPY” indica que o conteúdo do diretório atual na máquina de desenvolvimento será copiado para o diretório “/app” dentro do container. Em seguida, na linha 3, o comando “WORKDIR” define o diretório “/app” como diretório de trabalho, ou seja, o diretório onde os arquivos e dependências serão baixados e onde os comandos posteriores serão executados.

A seguir, na linha 4, o comando “RUN” define os comandos que o instalador de pacotes do Python deve executar dentro do diretório de trabalho, nesse caso o comando “pip install -r” irá instalar todos os requisitos do Python presentes no arquivo “requirements.txt”, que conforme exposto na Figura 17 possui apenas o Flask.

O arquivo de requisitos é utilizado pelo Docker para saber quais dependências da linguagem utilizada no desenvolvimento da aplicação devem ser instaladas de forma automática durante a construção do *container*, sem a necessidade de ter que instalar esses pacotes posteriormente.

A porta 5000 será exposta com o comando “EXPOSE” na linha 5. E por fim, na linha 6, o comando que executa o arquivo “index.py” é definido através do comando “CMD”.

**Figura 16 –** Conteúdo do Dockerfile




Fonte: Próprio Autor

Centro Paula Souza — SÃO PAULO GOVERNO DO ESTADO — Secretaria de Desenvolvimento Econômico

**Figura 17 –** Conteúdo do arquivo requirements.txt

Fonte: Próprio autor

Conforme citado anteriormente o arquivo do Docker Compose permite fazer um melhor controle de versão e automatizar o *build* do *container*.

Na primeira linha definimos a versão do *container*, caso ele tenha tido uma versão anterior. Na linha 2, são identificados os serviços que o Docker Compose deve configurar, que no cenário que será desenvolvido são apenas os arquivos no diretório “app”. Na linha 4, “build” está configurado para o diretório atual.

Em seguida, na linha 5, “ports” estabelece quais portas deverão ser expostas dentro e fora do *container*, ou seja, o *container* irá expor sua porta 5000 e a aplicação poderá ser acessada na porta 5000 do servidor. Em alguns casos pode ser que o desenvolvedor queira expor uma “porta A” do *container* numa “porta B” do servidor, por exemplo, expor a porta 3000 do *container* na porta 8000 do servidor, que resultaria na atribuição do valor “3000:8000” ao criar o arquivo de configuração.

Na linha 7 “command” tem a mesma função do “CMD” no Dockerfile. A razão pela qual temos que repetir o mesmo comando se deve ao fato de que o arquivo do Docker Compose ignora a última linha do Dockerfile.

**Figura 18 –** Conteúdo do arquivo docker-compose.yml

Centro Paula Souza
SÃO PAULO GOVERNO DO ESTADO
Secretaria de Desenvolvimento Econômico

```
1 version: "2"
2 services:
3     app:
4         build: .
5         ports:
6             - "5000:5000"
7         command: python index.py
```

Fonte: Próprio Autor

A estrutura do diretório depois da criação de todos os arquivos pode ser vista na Figura 19.

**Figura 19 –** Estrutura do diretório

```
ubuntu@ubuntu: ~/Desktop/docker-python
ubuntu@ubuntu:~/Desktop/docker-python$ tree
.
├── docker-compose.yml
├── Dockerfile
├── index.py
└── requirements.txt

0 directories, 4 files
```

Fonte: Próprio autor

### 3.2.3 Criação da Instância EC2

Depois de finalizadas as instalações, configurações do ambiente e configurações de credenciais podemos finalmente criar a instância e publicar a aplicação.

Para criar a instância vamos utilizar o Docker Machine instalado anteriormente, para isso basta um único comando "docker-machine create" que recebe como parâmetro o serviço utilizado no caso "amazonec2" e o nome que a instância criada deve receber, que conforme a Figura 20 foi nomeada "tcc-fatec"

**Figura 20 –** Execução do Docker Machine

```
ubuntu@ubuntu:~/Desktop/dockerized-python$ docker-machine create --driver amazonec2 tcc-fatec
Creating CA: /home/ubuntu/.docker/machine/certs/ca.pem
Creating client certificate: /home/ubuntu/.docker/machine/certs/cert.pem
Running pre-create checks...
Creating machine...
(tcc-fatec) Launching instance...
Waiting for machine to be running, this may take a few minutes...
Detecting operating system of created instance...
Waiting for SSH to be available...
Detecting the provisioner...
Provisioning with ubuntu(systemd)...
Installing Docker...
Copying certs to the local machine directory...
Copying certs to the remote machine...
Setting Docker configuration on the remote daemon...
Checking connection to Docker...
Docker is up and running!
```

Fonte: Próprio autor

Depois de executar o comando podemos consultar os detalhes da instância EC2 no *console* da AWS e verificar que um grupo de segurança chamado Docker Machine foi criado, como é exibido na Figura 21.

**Figura 21 –** Grupo de Segurança Docker Machine

Grupos de segurança (2) Informações

Filtrar grupos de segurança

| Name | ID do grupo de segur... | Nome do grupo de ... | ID da VPC | Descrição |
|---|---|---|---|---|
| – | sg-08493128f231afb7e | docker-machine | vpc-cef363b3 | Docker Machine |
| – | sg-de4dd2d8 | default | vpc-cef363b3 | default VPC security gr... |

Fonte: Próprio autor

A Figura 22 mostra o comando utilizado para acessar a instância recém-criada, basta utilizar o parâmetro "env" passando o nome da instância (tcc-fatec) seguido do comando fornecido na última linha do próprio output. Ao executar esse segundo

Centro Paula Souza
SÃO PAULO GOVERNO DO ESTADO
Secretaria de Desenvolvimento Econômico

comando todas as ações performadas via terminal já não estão sendo mais executadas na máquina local, mas sim no próprio ambiente da AWS.

Isso facilita o processo de construção da aplicação ao mover ela do ambiente de desenvolvimento para o ambiente da Nuvem.

**Figura 22 –** Comando para ativar a instância EC2 via Docker Machine

```
ubuntu@ubuntu:~/Desktop/docker-python$ docker-machine env tcc-fatec
export DOCKER_TLS_VERIFY="1"
export DOCKER_HOST="tcp://3.87.211.98:2376"
export DOCKER_CERT_PATH="/home/ubuntu/.docker/machine/machines/tcc-fatec"
export DOCKER_MACHINE_NAME="tcc-fatec"
# Run this command to configure your shell:
# eval $(docker-machine env tcc-fatec)
ubuntu@ubuntu:~/Desktop/docker-python$ eval $(docker-machine env tcc-fatec)
```

Fonte: Próprio autor

O próximo passo é construir a imagem Docker dentro da instância EC2 a partir do arquivo do Docker Compose. Como pode ser visto na Figura 23, ao executar o Docker Compose a imagem é construída passo a passo segundo as especificações do Dockerfile, como por exemplo a imagem Python Alpine, o WORKDIR e assim por diante.

**Figura 23 –** Construir o container a partir do arquivo docker-compose.yml

```
ubuntu@ubuntu:~/Desktop/docker-python$ docker-compose up -d
Creating network "docker-python_default" with the default driver
Building app
Sending build context to Docker daemon   5.12kB
Step 1/6 : FROM python:alpine3.7
alpine3.7: Pulling from library/python
48ecbb6b270e: Pull complete
692f29ee68fa: Pull complete
6439819450d1: Pull complete
3c7be240f7bf: Pull complete
ca4b349df8ed: Pull complete
Digest: sha256:35f6f83ab08f98c727dbefd53738e3b3174a48b4571ccb1910bae480dcdba847
Status: Downloaded newer image for python:alpine3.7
 ---> 00be2573e9f7
Step 2/6 : COPY . /app
 ---> 0cb14ef0e14e
Step 3/6 : WORKDIR /app
 ---> Running in 1fa0dca51d9c
Removing intermediate container 1fa0dca51d9c
 ---> 18d547ea1afd
Step 4/6 : RUN pip install -r requirements.txt
 ---> Running in 6518110ecf29
Collecting flask (from -r requirements.txt (line 1))
```

Fonte: Próprio autor

Antes de poder acessar a aplicação através do navegador devemos liberar seu endereço IPv4 público no *console* da AWS. Na Figura 24 podemos ver que para expor a aplicação para o mundo exterior devemos editar as regras de entrada nos Grupos de Segurança EC2.

**Figura 24 –** Editar regras de entrada da instância EC2

Fonte: Próprio autor

Conforme exibido na Figura 25, devemos adicionar uma nova regra de entrada que utiliza TCP personalizado, libera a porta 500 e permite acesso de qualquer lugar.

**Figura 25 –** Regra de entrada TCP personalizado

Regras de entrada Informações

| Tipo Informações | Protocolo Informações | Intervalo de portas Informações | Origem Informações | |
|---|---|---|---|---|
| SSH | TCP | 22 | Personalizado | 0.0.0.0/0 |
| TCP personalizado | TCP | 2376 | Personalizado | 0.0.0.0/0 |
| TCP personalizado | TCP | 5000 | Qualquer l... | 0.0.0.0/0 ::/0 |

Fonte: Próprio auto

Ao consultar o resumo da instância, exibido na Figura 25, verificamos que a instância está em execução e que seu endereço IPv4 público é 3.87.211.98.

**Figura 26 –** Resumo da instância

Resumo da instância para i-03a916975f633cf0e (tcc-fatec) Informações
Atualizado há less than a minute

ID de instância
i-03a916975f633cf0e (tcc-fatec)

Endereço IPv4 público
3.87.211.98 | endereço aberto

Estado da instância
Executando

DNS IPv4 público
ec2-3-87-211-98.compute-1.amazonaws.com | endereço aberto

Tipo de instância
t2.micro

Endereços IP elásticos
–

Descoberta do AWS Compute Optimizer
Opte por participar do AWS Compute Optimizer para obter recomendações. | Saiba mais

Função do IAM
–

Fonte: Próprio autor

Para verificar a publicação da aplicação basta acessar o endereço http://3.87.211.98:5000/ no navegador, e conforme pode ser visto na Figura 26, temos como resultado a mensagem exibida pelo código Python.

**Figura 27 –** Mensagem exibida em produção

FATEC AMERICANA Essa é uma simples aplicação rodando em Docker

Fonte: Próprio autor

Portanto ao fim temos uma aplicação web sendo executada em um container Docker que utiliza uma instância da AWS.

# 4 RESULTADOS

Conforme demonstrado na seção prática do presente trabalho, ao utilizar o Docker e as ferramentas que fazem parte de seu ecossistema é possível otimizar o processo de desenvolvimento e *deployment* de aplicações, em especial de aplicações simples que não demandam um forte planejamento e preparação de infraestrutura, como foi o caso do exemplo de código utilizando Flask.

Além do ganho em velocidade de desenvolvimento e implementação trazida pelo Docker, a possibilidade de integração com um ambiente de Nuvem Pública também foi facilitado pelo uso do binário Docker Machine, uma vez que a única mudança no processo seria a troca do *driver* utilizado pelo provedor, dessa forma mesmo que a aplicação fosse hospedada na Google Cloud, Microsoft Azure ou IBM Cloud o processo seria muito similar.

Essa portabilidade entre diferentes provedores de nuvem também beneficia equipes de desenvolvimento trazendo ganho de produtividade, reduzindo significativamente a curva de aprendizado das ferramentas e modo de trabalhar de cada provedor. Ao invés de precisar entender toda a estrutura e a filosofia de um novo ambiente em nuvem, desenvolvedores precisariam apenas entender como um serviço específico de hospedagem funciona.

Isso fica ainda mais evidente em situações nas quais, por exemplo, um determinado *developer* tem maior conhecimento sobre a plataforma de um “Provedor X “e recebe uma nova demanda que requer a utilização do “Provedor Y”. Em um cenário como esse o provedor utilizado seria indiferente, já que a aplicação seria desenvolvida e encapsulada em um *container* Docker, portanto não haveria desperdício de tempo na adaptação para um novo ambiente.

Por fim, demonstrou-se os *containers* delegam aos desenvolvedores parte do poder e capacidade operacional que normalmente é exclusiva dos analistas de infraestrutura, permitindo que um sistema seja construído de ponta a ponta sem a necessidade de interação e validação constante entre os times, dando mais

autonomia aos desenvolvedores e sendo assim potencialmente reduzindo a carga de trabalho dos responsáveis por infraestrutura.

## 5 CONCLUSÕES E CONSIDERAÇÕES FINAIS

A indústria de Tecnologia da Informação é sem sombra de dúvida uma das mais dinâmicas e inovadoras entre as diversas áreas de conhecimento modernas. Essa constante mudança e ascensão de novidades é ao mesmo tempo sua maior força e sua maior fraqueza, principalmente devido ao grande número de especuladores e supostas tecnologias milagrosas que acabam sendo engolidas em um curto espaço de tempo por outras tecnologias que na verdade são o milagre.

Com o Docker não foi diferente, especialmente nos seus primeiros anos de vida e com o crescimento de seu uso impulsionado pelas comunidades técnicas e entusiastas. Entretanto com o tempo o Docker se provou uma tecnologia eficiente e viável, que realmente trouxe melhoria para a indústria e gerou o replanejamento e adaptação diversas de metodologias e arquiteturas.

No ano de 2021 o “Kubernetes”, tecnologia desenvolvida pela Google que permite “gerenciamento de cargas de trabalho e serviços distribuídos em containers” (Kubernetes, 2021) anunciou que deixaria de utilizar o Docker como agente primário de execução, causando uma enorme perda na influência e hegemonia que o Docker possuía até então.

Os *containers* são apenas uma peça de um enorme quebra-cabeça e nem sempre fornecem sozinhos a capacidade de escalabilidade, segurança e gerenciamento necessários para atender os requerimentos de determinados cenários. Portanto em trabalhos futuros sugere-se a abordagem dos orquestradores de containers como uma forma automatizar e escalar o processo de *deployment* .

Pode ser que num futuro relativamente próximo o Docker seja substituído por alguma outra tecnologia de containers, qualquer ferramenta está sujeita a isso e essa é uma tendência natural. Entretanto é inegável que as contribuições dessa ferramenta influenciaram e mudaram o futuro da infraestrutura. Portanto muito além de sua popularidade, base de usuários ou tendências de mercado o Docker sem dúvida deixará um legado para futuras tecnologias.

Centro Paula Souza
SÃO PAULO GOVERNO DO ESTADO
Secretaria de Desenvolvimento Econômico

## REFERÊNCIAS BIBLIOGRÁFICAS


AWS. **Amazon EC2**. Disponível em: https://aws.amazon.com/pt/ec2/?ec2-whats-new.sort-by=item.additionalFields.postDateTime&ec2-whats-new.sort-order=desc. Acesso em: 09 jun. 2021.

CISCO. **What Is a Data Center**. Disponível em: https://www.cisco.com/c/en_uk/solutions/data-center-virtualization/what-is-a-data-center.html#:~:text=In%20the%20world%20of%20enterprise,Email%20and%20file%20sharing&text=Big%20data%2C%20artificial%20intelligence%2C%20and,desktops%2C%20communications%20and%20collaboration%20services. Acesso em: 09 jun. 2021.

DOCKER. **Docker Machine overview**. Disponível em: https://docs.docker.com/machine/. Acesso em: 09 jun. 2021.

DOCKER. **Dockerfile reference**. Disponível em: https://docs.docker.com/engine/reference/builder/. Acesso em: 09 jun. 2021.

DOCKER. **Install Docker Compose**. Disponível em: https://docs.docker.com/compose/install/. Acesso em: 09 jun. 2021.

DOCKER. **Overview of Docker Compose**. Disponível em: https://docs.docker.com/compose/. Acesso em: 09 jun. 2021.

DOCKER. **What is a Container?** Disponível em: https://www.docker.com/resources/what-container#:~:text=A%20Docker%20container%20image%20is,tools%2C%20system%20libraries%20and%20settings.&text=Standard%3A%20Docker%20created%20the%20industry,they%20could%20be%20portable%20anywhere. Acesso em: 09 jun. 2021.

FLASK. **Quickstart**. Disponível em: https://flask.palletsprojects.com/en/1.1.x/quickstart/. Acesso em: 09 jun. 2021.

HAWKINS, Kerry; RESTIVO, Michael. **Data centers: expensive to build, but worth every penny**. JLL, 2021. Disponível em: https://www.us.jll.com/en/views/data-centers-expensive-to-build-but-worth-every-penny. Acesso em: 09 jun. 2021.

KUBERNETES. **Visão Geral**. Disponível em: https://kubernetes.io/pt-br/docs/concepts/overview/. Acesso em: 09 jun. 2021.

MICROSOFT AZURE. **O que é um contêiner?** Disponível em: https://azure.microsoft.com/pt-br/overview/what-is-a-container/. Acesso em: 09 jun. 2021.

Centro Paula Souza | SÃO PAULO GOVERNO DO ESTADO | Secretaria de Desenvolvimento Econômico

REDHAT. **Advantages Of Using Docker**. Disponível em: https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/7.0_release_notes/sect-red_hat_enterprise_linux-7.0_release_notes-linux_containers_with_docker_format-advantages_of_using_docker. Acesso em: 09 jun. 2021.

REDHAT. **Introdução à virtualização**. Disponível em: https://www.redhat.com/pt-br/topics/virtualization. Acesso em: 09 jun. 2021.

REDHAT. **Introdução aos containers Linux**. Disponível em: https://www.redhat.com/pt-br/topics/containers. Acesso em: 09 jun. 2021.

REDHAT. **Linux Containers With Docker Format**. Disponível em: https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/7.0_release_notes/chap-red_hat_enterprise_linux-7.0_release_notes-linux_containers_with_docker_format#sect-Red_Hat_Enterprise_Linux-7.0_Release_Notes-Linux_Containers_with_Docker_Format-Components_of_Docker_Containers. Acesso em: 09 jun. 2021.

REDHAT. **O que é container Linux?** Disponível em: https://www.redhat.com/pt-br/topics/containers/whats-a-linux-container. Acesso em: 09 jun. 2021.

REDHAT. **O que é um hypervisor?** Disponível em: https://www.redhat.com/pt-br/topics/virtualization/what-is-a-hypervisor. Acesso em: 09 jun. 2021.

REDHAT. **O que é Virtualização**. Disponível em: https://www.redhat.com/pt-br/topics/virtualization/what-is-virtualization. Acesso em: 09 jun. 2021.

STALCUP, Katy. **AWS vs Azure vs Google Cloud Market Share 2021**: What the Latest Data Shows. ParkMyCloud, 2021. Disponível em: https://www.parkmycloud.com/blog/aws-vs-azure-vs-google-cloud-market-share/#:~:text=AWS%20has%2032%25%20of%20the,and%20most%20functionality%2Drich.%E2%80%9D. Acesso em: Acesso em: 09 jun. 2021.

TAGLIAFERRI, Lisa. **Como instalar o Python 3 e configurar um ambiente de programação em um servidor Ubuntu 20.04**. Digital Ocean, 2021. Disponível em: https://www.digitalocean.com/community/tutorials/how-to-install-python-3-and-set-up-a-programming-environment-on-an-ubuntu-20-04-server-pt. Acesso em: 09 jun. 2021.