Mini Tutorial

O intuito deste tutorial é colocar em prática o conceito de "DevOps" (combinação dos termos "desenvolvimento" e "operações"). A metodologia DevOps descreve abordagens que ajudam a acelerar os processos necessários para levar uma ideia do desenvolvimento à implantação em um ambiente de produção no qual ela seja capaz de gerar valor para o usuário.

Publicaremos uma aplicação web baseada em Python e Flask em uma instância AWS EC2. Para isso, usaremos o WSGI Gunicorn sobre um container Docker.

1. Entendendo cada tecnologia envolvida

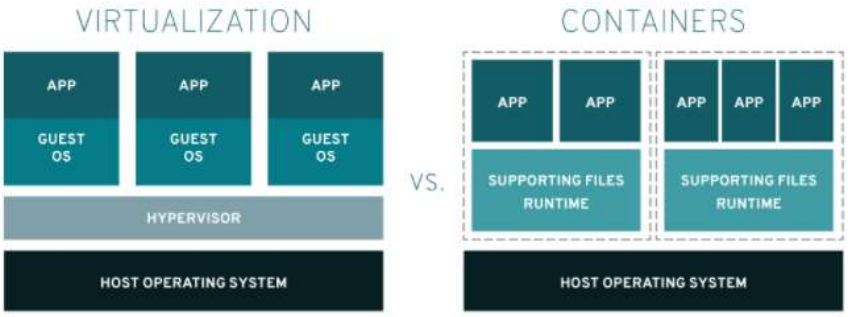
1.1 Docker

A tecnologia Docker usa o kernel do Linux e recursos do *kernel* como *Cgroups* e *namespaces* para segregar processos. Assim, eles podem ser executados de maneira independente. O objetivo dos containers é criar essa independência: a habilidade de executar diversos processos e aplicações separadamente para utilizar melhor a infraestrutura e, ao mesmo tempo, manter a segurança que você teria em sistemas separados.

Analisando as tecnologias de virtualização e de containers pode-se dizer que ambas são distintas, porém complementares.

- Com a virtualização, é possível executar sistemas operacionais (Windows ou Linux) simultaneamente em um único sistema de hardware.

- Os containers compartilham o mesmo kernel do sistema operacional e isolam os processos da aplicação do restante do sistema. Por exemplo, os sistemas ARM Linux executam containers ARM Linux, os sistemas x86 Linux executam containers x86 Linux e os sistemas x86 Windows executam containers x86 Windows. Os containers Linux são extremamente portáteis, mas devem ser compatíveis com o sistema subjacente.



Isso significa que a virtualização usa um hipervisor para emular o hardware, o que permite executar vários sistemas operacionais simultaneamente. Essa não é uma solução tão leve quanto o uso de containers. Quando a capacidade e os recursos são limitados, é necessário usar aplicações leves que possam ser implantadas densamente. Os containers Linux são executados de maneira nativa no sistema operacional, compartilhando-o com todos os outros containers. Assim, as aplicações e os serviços permanecem leves e são executados em paralelo com agilidade.

As ferramentas de container, incluindo o Docker, fornecem um modelo de implantação com base em imagem. Isso facilita o compartilhamento de uma aplicação ou conjunto de serviços, incluindo todas as dependências deles em vários ambientes. O Docker também automatiza a implantação da aplicação (ou de conjuntos de processos que constituem uma aplicação) dentro desse ambiente de container.

Essas ferramentas baseadas nos containers Linux (o que faz com que o Docker seja exclusivo e fácil de usar) oferecem aos usuários acesso sem precedentes a aplicações, além da habilidade de implantar com rapidez e de ter total controle sobre as versões e distribuição.

Deve-se ressaltar que “imagem” e “container” são dois conceitos complementares. Uma imagem é um arquivo inerte, imutável. Já um container é uma instância de uma imagem, ou seja, ele é uma imagem em execução (rodando). Se você iniciar esta imagem, você terá um container em execução. Você pode ter muitos contêineres em execução da mesma imagem.

Imagine um arquivo binário (.exe) armazenado no disco, quando você digita seu nome e pressiona “enter”, ele passar a executar, sendo chamado de “processo”. Pois bem, a imagem está para o arquivo binário assim como o container está para o processo.

Fonte: https://www.redhat.com/pt-br/topics/containers/what-is-docker

1.2 Python

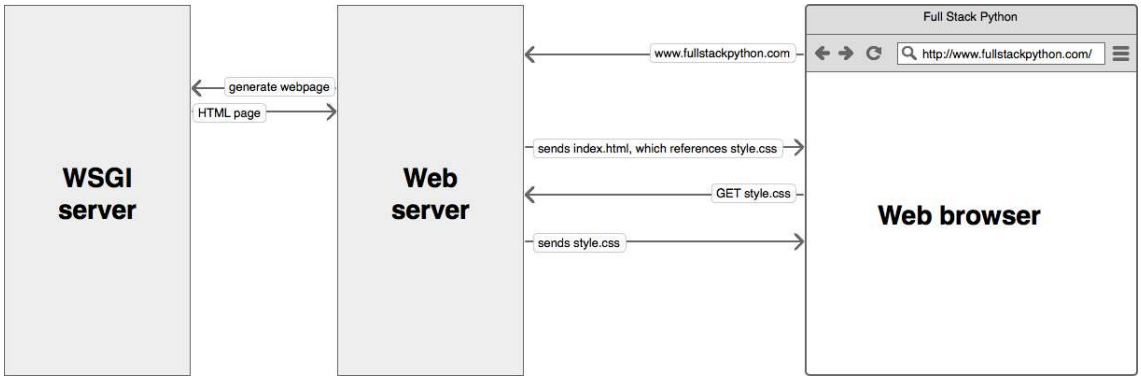
Python é uma linguagem fácil de aprender e poderosa. Ela tem estruturas de dados de alto nível eficientes e uma abordagem simples, mas efetiva de programação orientada a objetos. A elegância de sintaxe e a tipagem dinâmica do Python aliadas com sua natureza interpretativa, o fazem a linguagem ideal para programas e desenvolvimento de aplicações rápidas em diversas áreas e na maioria das plataformas.

O interpretador Python e a extensiva biblioteca padrão estão disponíveis gratuitamente em código ou na forma binária para as maiores plataformas, no endereço eletrônico do Python (https://www.python.org) e pode ser livremente distribuído.

Fonte: https://docs.python.org/pt-br/3/tutorial/index.html

1.3 Gunicorn

Gunicorn 'Green Unicorn' é um servidor Python Web Server Gateway Interface (WSGI) de HTTP para UNIX, sendo amplamente compatível com vários frameworks da web. Além de ser um servidor amplamente compatível com uma série de frameworks, ele é leve e totalmente gerenciável. WSGI é por design uma interface padrão simples para executar código Python.



Há de se ressaltar que neste tutorial não iremos falar sobre *webservers* (ex. Nginx). Entretanto, para uma solução completa recomenda-se que seja adotado um, como o Nginx, para interagir

com o Gunicorn. Apenas para matar a curiosidade, o *webserver* lida melhor com muitos pedidos simultâneos de recursos estáticos (imagens, textos, etc.). As solicitações que precisam ser geradas dinamicamente serão passadas para o servidor de aplicativos, no caso o Gunicorn.

Além disso, o Nginx tem algumas funcionalidades de servidor da web (por exemplo, servir páginas estáticas; manipulação de SSL) que o Gunicorn não possui, enquanto o Gunicorn implementa WSGI (o que o Nginx não).

Fontes: (1) https://gunicorn.org (2) https://en.wikipedia.org/wiki/Gunicorn (3) https://www.fullstackpython.com/wsgi-servers.html

1.4 Flask

Framework é um componente que ajuda a simplificar o desenvolvimento de aplicativos, principalmente web. É como um kit de ferramentas que inclui serviços da web, recursos, APIs e fornece uma base estável para seu aplicativo.

Flask é um pequeno framework web escrito em Python. É classificado como um microframework porque não requer ferramentas ou bibliotecas particulares, mantendo um núcleo simples, porém, extensível.

“Micro” não significa que a sua aplicação web inteira tem que se encaixar em um único arquivo Python, embora certamente pode. Também não quer dizer que o Flask está faltando em termos de funcionalidade. O “micro” no microframework Flask visa manter o núcleo simples, mas extensível. Flask não vai tomar muitas decisões para você, como o banco de dados para usar. Essas decisões que ele faz, como o que motor de templates usar, são fáceis de mudar. Todo o resto é com você, então o Flask que pode ser tudo que você precisa e nada que você não faz.

Por padrão, Flask não inclui uma camada de abstração de banco de dados, uma validação de form ou qualquer outra coisa que já exista em bibliotecas diferentes que pode lidar com isso. Em vez disso, Flask suporta extensões para adicionar essa funcionalidade a sua aplicação como se fosse implantado no Flask. Há várias extensões para proporcionar a integração de banco de dados, validação de formulário, manipulação de upload, várias tecnologias de autenticação abertas, e muito mais. Flask pode ser “micro”, mas está pronto para uso na produção para uma variedade de necessidades.

Apenas uma única observação: utilize o servidor padrão do Flask apenas para testes, nunca em ambiente de produção. A ferramenta não lida muito bem com requisições simultâneas, etc. Por isso precisamos do Gunicorn.

Fontes: (1) https://flask-ptbr.readthedocs.io/en/latest/foreword.html#qual-o-significado-de-micro (2) http://pythonclub.com.br/what-the-flask-pt-1-introducao-ao-desenvolvimento-web-com-python.html

1.5 AWS EC2

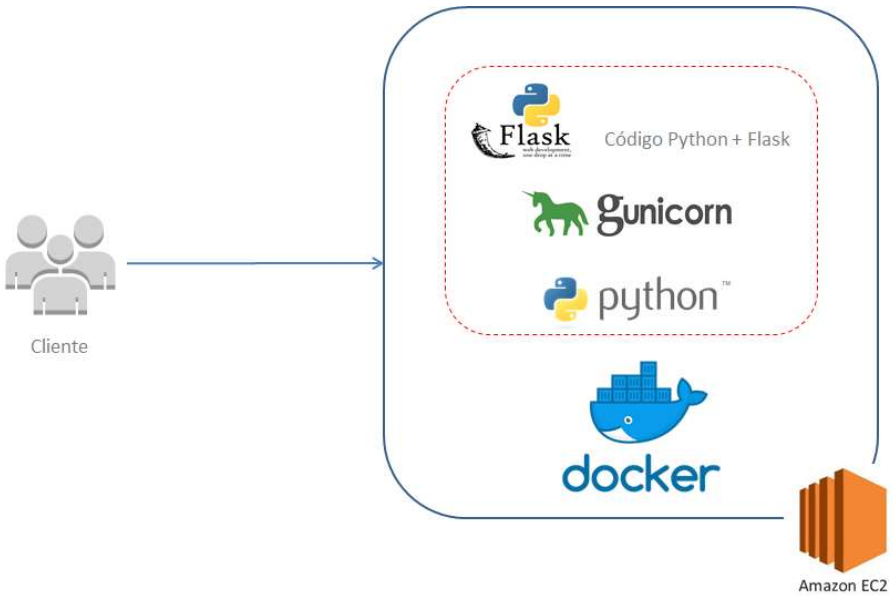
O Amazon Elastic Compute Cloud (EC2) é uma solução que facilita a obtenção de servidores virtuais, também conhecidos como instâncias de computadores na nuvem. É um serviço web que disponibiliza capacidade computacional segura e redimensionável na nuvem. Ele foi projetado para facilitar a computação em nuvem na escala da web para os desenvolvedores. A interface de serviço Web simples do Amazon EC2 permite que você obtenha e configure a capacidade sem muito esforço. Oferece um controle completo de seus recursos computacionais e permite que você trabalhe no ambiente computacional comprovado da Amazon.

O Amazon EC2 oferece a maior e mais abrangente plataforma de computação com a possibilidade de escolha de processador, armazenamento, rede, sistema operacional e modelo de compra. Oferecemos os processadores mais rápidos na nuvem e somos a única nuvem com rede Ethernet de 400 Gbps.

Fontes: (1) https://aws.amazon.com/pt/ec2/?ec2-whats-new.sort-by=item.additionalFields.postDateTime&ec2-whats-new.sort-order=desc (2) https://skyone.solutions/pb/o-que-e-ec2-aws/

2. Entendendo como as tecnologias se relacionam

A arquitetura do sistema é representada pela imagem a seguir:



A instância EC2 na AWS faz parte do nível de serviço IaaS (Infraestrutura como Serviço), ou seja, é a máquina virtual nos servidores da Amazon. É nessa máquina que o Docker é executado, o gerenciador de containers. O Docker executa a imagem (conjunto de arquivos de código fonte e ferramentas), em especial o interpretador Python.

É preciso ressaltar que, não é obrigatório o uso do Docker (ou qualquer ferramenta ligada à containers) para a execução de uma aplicação web. Mas, o Docker oferece uma maneira segura de rodar aplicações isoladas em containers, empacotadas com todas as suas dependências e bibliotecas. Uma aplicação "dockerizada" pode ser transportada e executada em qualquer plataforma sem nenhuma modificação. Em suma, ele reduz tempo de *build* e *deploy.* O ideal é você criar uma imagem, via Docker, no seu ambiente de desenvolvimento e depois transferir para o ambiente de produção. Mas isso é assunto para outro tutorial.

O interpretador Python executa códigos fontes e outras chamadas que o servidor WSGI Gunicorn possa processar. Há de se ressaltar que, embora estejamos usando o servidor Gunicorn, o Flask também possui um servidor web. Entretanto, é preciso dizer que o servidor web incluso no Flask não é tão robusto quanto o Gunicorn, não sendo capaz de lidar muito bem com inúmeras requisições simultâneas. Por isso, sua substituição nesse exemplo.

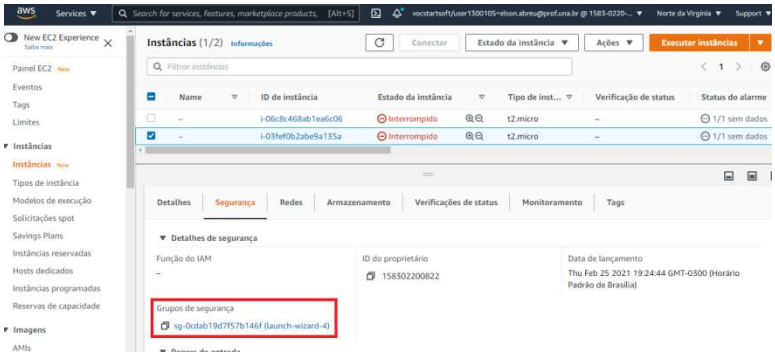
3. Passo a Passo

3.1. *Este passo é opcional, caso você já tenha uma instância EC2.*

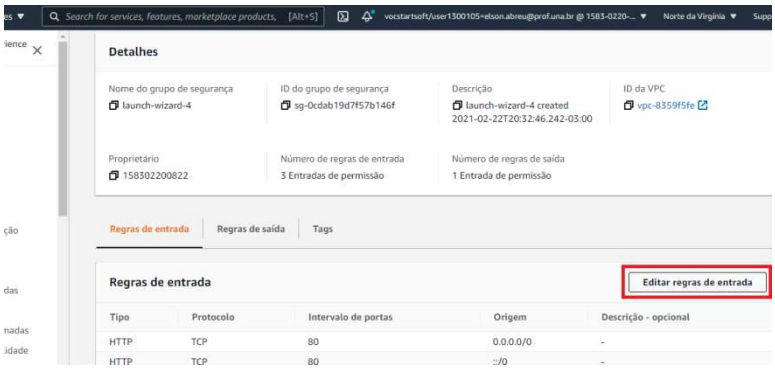
Crie uma instância padrão "***Amazon Linux 2 AMI (HVM), SSD Volume Type - 64 bits(x86)***", com o nome/tag "**vmpython**" e acesse via SSH ou conectar via ec2 console(Sandbox Environment). Lembre-se de liberar a porta 5000 (TCP) quando criar a máquina, caso contrário o teste de conectividade não funcionará. Na atividade anterior aprendemos a criar a máquina, procure resgatar os conceitos.

Caso tenha criado a instância, mas não tenha concedido acesso à porta 5000, faça o seguinte:

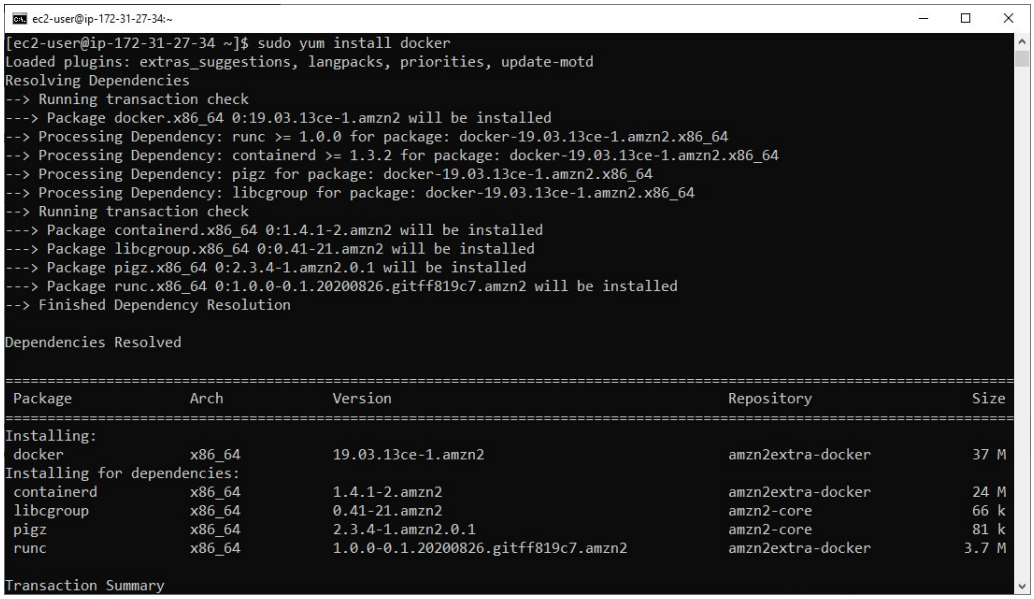
Selecione a instância, clique na aba "Segurança", depois clique no "Grupo de Segurança":



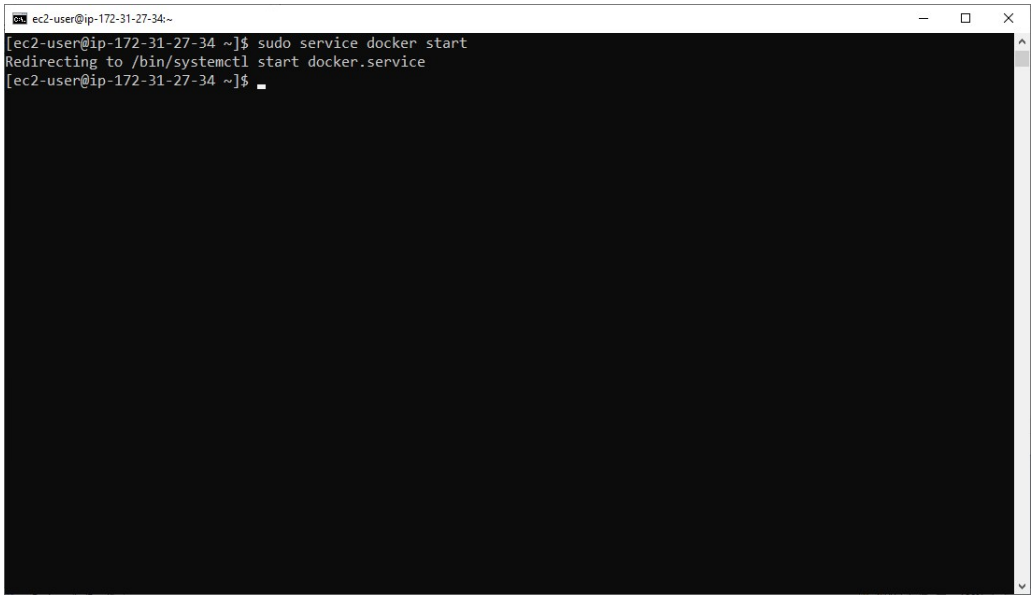
Depois, clique em "Editar regras de entrada" e adicione a porta:



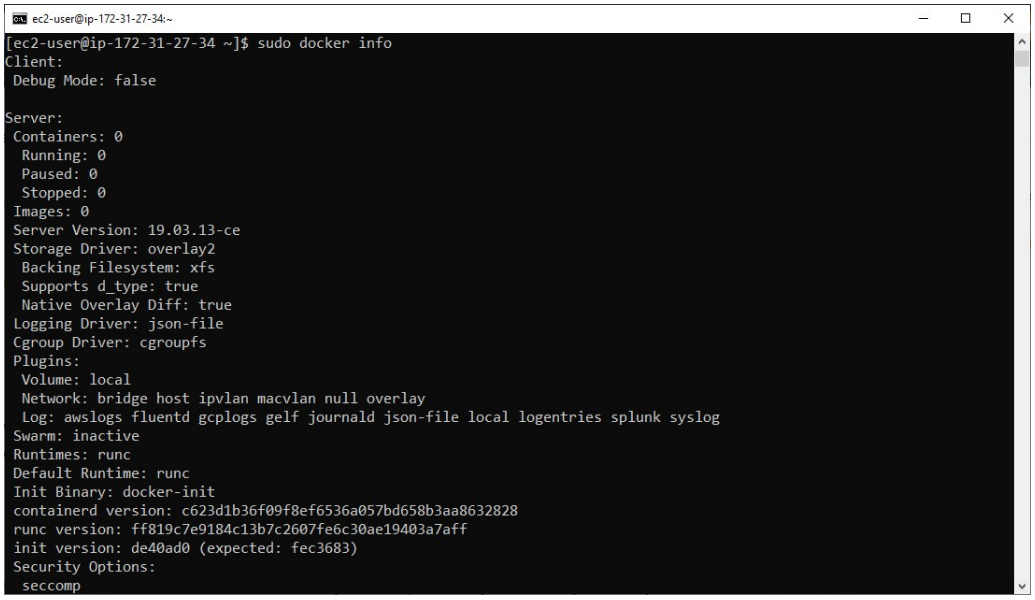
3.2. Conecte a sua instancia e Instale o Docker com "***sudo yum install docker***":



3.3. Dê um start no serviço do Docker com "***sudo service docker start***":



3.4. É importante verificar se o Docker está rodando, observando suas informações com o comando "***sudo docker info***":



3.5. Instale o Flask e o Gunicorn, utilizando o Pip3, com os comandos: **"sudo yum install python3-pip", "*sudo pip3 install flask*" e "*sudo pip3 install gunicorn*".**

3.6. Crie um diretório chamado "projetos", e dentro deste, crie um outro diretório chamado api. **"mkdir projetos", "cd projetos", "mkdir api"** e **"cd api"**

3.7. Dentro do diretório "api", digite o comando **"nano main.py"**. O editor Nano será aberto e o código abaixo deverá ser digitado/colado. Para salvar, pressione **"Ctrl + O"**, depois **"Ctrl+X"** para sair:

```
from flask import Flask

app = Flask(__name__)

@app.route('/api', methods=['GET'])

def handler():

   return 'Esta é a nossa API', 200

if __name__ == "__main__":

    app.run(host="0.0.0.0", port=int("5000"), debug=True)
```

3.8. Crie um arquivo contendo os requisitos do projeto (bibliotecas requeridas) com o comando ***"sudo pip3 freeze > requirements.txt"***

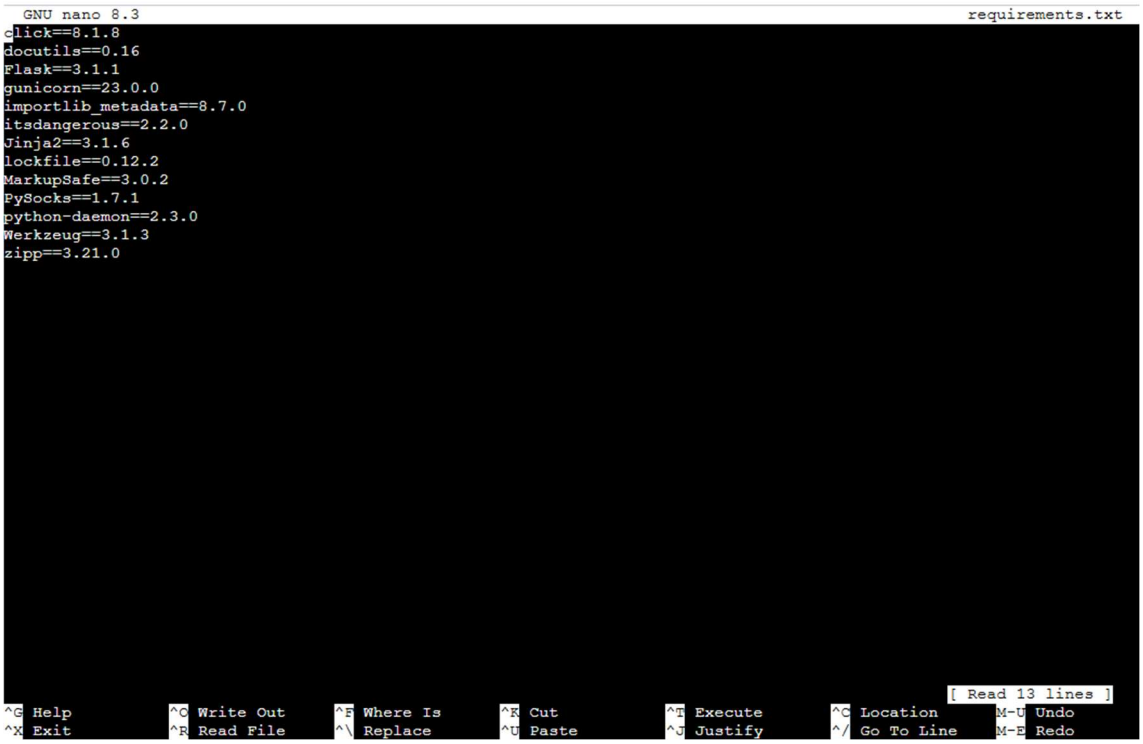
Você pode verificar se o arquivo foi criado com sucesso usando o comando "***nano requirements.txt***", assim verá o conteúdo do mesmo.

Obs.: *O comando "pip3 freeze" lista os pacotes Python instalados no seu ambiente já no formato que o "pip install" consiga entender. Ao associá-lo com o operador ">" a saída é redirecionada para um arquivo de texto.*

Também é possível conferir o conteúdo com o comando ***"cat requirements.txt"***.

Antes de pular para o próximo passo, edite o arquivo requirements.txt através do nano e exclua as linhas diferentes da imagem a seguir pois eles não serão necessários.

O conteúdo do arquivo deverá ficar da seguinte maneira:



Obs.: entre a data de criação deste tutorial e o uso, as versões podem mudar.

3.9. Precisamos criar um arquivo chamado "dockerfile" (sem extensão), para indicar ao Docker o que deverá ser executado quando o container for carregado para execução. Digite ***"nano dockerfile"*** e no editor Nano entre com os comandos abaixo. Pressione **Ctrl+O** e depois **Ctrl+X**.

```
FROM python:3

COPY . /work

WORKDIR /work

EXPOSE 5000

RUN pip3 install --no-cache-dir -r requirements.txt

CMD gunicorn --workers 2 --bind 0.0.0.0:5000 main:app
```

Obs.: *observe que o comando EXPOSE 5000 irá abrir a porta 5000 TCP para receber as conexões HTTP e o Gunicorn será executado na mesma (conforme última linha). Além disso, perceba que o nome da aplicação é "main" e o objeto Flask dentro dela é "app". Por fim, mas não menos importante, a linha "FROM python:3" indica a imagem que será usada a partir do hub Docker.*

3.10. Gere a imagem do container com o seguinte comando:

"sudo docker build --tag my-api ."

Obs.: existe um "ponto" ao final do comando, que indica para utilizar a própria pasta como base. Lembre-se que estamos dentro do diretório da aplicação. Para conferir, dê um comando "pwd". Não deixe de colocá-lo!

3.11. Execute o container ***"sudo docker run --name main -p 5000:5000 my-api":***

```
[ec2-user@ip-10-0-0-130 api]$ sudo docker run --name main -p 5000:5000 my-api
[2021-10-13 00:47:53 +0000] [7] [INFO] Starting gunicorn 20.1.0
[2021-10-13 00:47:53 +0000] [7] [INFO] Listening at: http://0.0.0.0:5000 (7)
[2021-10-13 00:47:53 +0000] [7] [INFO] Using worker: sync
[2021-10-13 00:47:53 +0000] [8] [INFO] Booting worker with pid: 8
[2021-10-13 00:47:53 +0000] [9] [INFO] Booting worker with pid: 9
```

3.12. Se tudo ocorreu conforme previsto, o Gunicorn foi iniciado e a aplicação foi publicada. Para testar se a aplicação está mesmo rodando, abra o browser (firefox, chrome, etc.) do seu computador e digite:

http:// <sua máquina na amazon>:5000/api"

Substitua **<sua máquina na amazon>** pelo nome da sua máquina EC2 na Amazon AWS. No meu caso, o nome é:

"http://ec2-34-228-54-24.compute-1.amazonaws.com:5000/api"

Comandos para start/stop de uma aplicação em Docker

a) Listar os containers:

Todos: ***"sudo docker container ps"***

Em execução: ***"sudo docker container ls -a"***

b) Parar um container em execução:

"sudo docker container stop <container_id>"

Deverá substituir ***<container_id>*** *pelo identificador coletado pelo comando do item "a".*

c) Inicializar um container (com base em uma imagem previamente instalada):

"sudo docker start <container_id>"

Dicas para outras manutenções:

Listar as imagens disponíveis na máquina:

"sudo docker images -a"

Excluir a imagem:

"sudo docker image rmi <IMAGE_ID>"

Substituindo **<IMAGE_ID>** pelo código da imagem.