Instalar o Python 3 e configurar um ambiente de programação em um servidor Linux

Antes de começar a instalação é necessário verificar se o sistema possui algumas bibliotecas que nos ajudarão a compilar o código fonte do Python.

Assim, digite:

```
$ sudo apt install build-essential checkinstall
```

- **$** indica que você deve usar o **usuário comum** para fazer essa operação.
- **sudo** serve para pedir permissões de administrador temporariamente.
- **apt** do inglês, *Advanced Package Tool*, em português, Ferramenta de Empacotamento Avançada; é a ferramenta que nos ajuda na instalação, atualização e desinstalação de programas, entre outras funções.
- **install** é o comando de instalar, indicando ao apt o que fazer.
- **build-essential** é uma biblioteca que reúne diversas aplicações para compilar e instalar outros programas, que inclui, por exemplo, o make, automake, etc.
- **checkinstall** é um programa que facilita e monitora o processo de instalação e desinstalação de programas compilados a partir da fonte (uso do make install).

Vamos instalar também outras bibliotecas de desenvolvimento que nos ajudarão na compilação do código fonte:

```
$ sudo apt install libreadline-gplv2-dev libncursesw5-dev libssl-dev libsqlite3-dev tk-dev libgdbm-dev libc6-dev libbz2-dev libffi-dev zlib1g-dev
```

Passo 1 — Configurando o Python 3

O Ubuntu 20.04 e outras versões do Debian Linux vem com o Python 3 pré-instalado. Para garantir que nossas versões estejam atualizadas, vamos atualizar o sistema com o comando `apt` para funcionar com a **A**dvanced **P**ackaging **T**ool (Ferramenta de Empacotamento Avançada, normalmente referida como “apt” apenas) do Ubuntu:

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

O sinalizador `-y` irá confirmar nossa concordância com todos os itens a serem instalados, mas, dependendo da sua versão do Linux, você pode precisar confirmar prompts adicionais como as atualizações e upgrades do seu sistema.

Assim que o processo for concluído, podemos verificar a versão do Python 3 que está instalada no sistema digitando:

```
1. python3 -V
2.
```

Você receberá um resultado na janela do terminal que irá informar a você o número da versão. Embora esse número possa variar, o resultado será similar a este:

```
Output
Python 3.8.2
```

Para gerenciar os pacotes de software do Python, vamos instalar o **pip**, uma ferramenta que irá instalar e gerenciar pacotes de programação que podemos querer usar em nossos projetos. Você pode aprender mais

sobre os módulos ou pacotes que você pode instalar com o pip lendo "Como Importar Módulos no Python 3."

```
1. sudo apt install -y python3-pip
```

Os pacotes Python podem ser instalados digitando:

```
1. pip3 install package_name
```

Aqui, o `package_name` pode se referir a qualquer pacote ou biblioteca Python, tais como o Django para o desenvolvimento Web ou o NumPy para a computação científica. Então, se você quiser instalar o NumPy, você pode fazer isso com o comando `pip3 install numpy`.

Há mais alguns pacotes e ferramentas de desenvolvimento a serem instalados para garantir que teremos uma configuração robusta para nosso ambiente de programação:

```
1. sudo apt install -y build-essential libssl-dev libffi-dev python3-dev
```

Assim que o Python estiver configurado e o pip e outras ferramentas estiverem instaladas, podemos configurar um ambiente virtual para nossos projetos de desenvolvimento.

Passo 2 — Configurando um Ambiente Virtual

Os ambientes Virtuais permitem que você tenha um espaço isolado no seu servidor para projetos Python, garantindo que cada um dos seus projetos possa ter seus próprios conjuntos de dependências que não irão perturbar nenhum dos seus outros projetos.

Configurar um ambiente de programação fornece um maior controle sobre os projetos Python e sobre como as versões diferentes de pacotes são tratadas. Isso é especialmente importante durante o trabalho com pacotes de terceiros.

É possível configurar quantos ambientes de programação Python você quiser. Basicamente, cada ambiente é um diretório ou pasta no seu servidor que contém alguns scripts dentro dele, com o intuito de fazê-lo funcionar como um ambiente.

Enquanto houver algumas maneiras de alcançar um ambiente de programação no Python, vamos usar o módulo **venv** aqui, que faz parte da biblioteca padrão do Python 3. Vamos instalar o venv digitando:

```
1. sudo apt install -y python3-venv
```

Com ele instalado, estaremos prontos para criar ambientes. Vamos escolher em qual diretório gostaríamos de colocar nossos ambientes de programação Python, ou criar um novo diretório com o `mkdir`, como em:

```
1. mkdir environments
2. cd environments
```

Assim que estiver no diretório onde você quer que os ambientes fiquem, você criará um ambiente executando o seguinte comando:

```
1. python3 -m venv my_env
```

Essencialmente, o `pyvenv` configura um novo diretório que contém alguns itens que podemos ver com o comando `ls`:

```
1. ls my_env
```

```
Output
bin include lib lib64 pyvenv.cfg share
```

Juntos, esses arquivos funcionam para garantir que seus projetos estejam isolados do contexto mais amplo de seu servidor, para que os arquivos do sistema e os arquivos do projeto não se misturem. Esta é uma boa prática para o controle de versão e para garantir que cada um dos seus projetos tenha acesso aos pacotes específicos de que necessita. O, Python Wheels, que é um formato embutido do Python que pode acelerar a produção de seu software, reduzindo o número de vezes que você precisa compilar, estará no diretório `share` do Ubuntu 20.04.

Para usar este ambiente, você precisa ativá-lo, o que pode ser feito digitando-se o seguinte comando que chama o script **activate**:

```
1. source my_env/bin/activate
```

Seu prompt de comando agora estará prefixado com o nome do seu ambiente, neste caso ele se chama my_env. Dependendo da versão do Linux Debian que estiver executando, o seu prefixo pode parecer um pouco diferente, mas o nome do seu ambiente em parênteses deve ser a primeira coisa que verá em sua linha:

```
(myprojectenv)user@host:~/myproject$
```

Esse prefixo nos permite saber que o ambiente my_env está ativo no momento, o que significa que quando criarmos programas aqui, eles somente usarão as configurações e pacotes específicos deste ambiente.

Nota: dentro do ambiente virtual, você pode usar o comando `python` em vez de `python3`, e `pip` em vez de `pip3` se preferir. Se usar o Python 3 em sua máquina fora de um ambiente, será necessário usar os comandos `python3` e `pip3` exclusivamente.

Após seguir esses passos, seu ambiente virtual estará pronto para usar.

Passo 3 — Criando um Programa “Hello, World”

Agora que temos nosso ambiente virtual configurado, vamos criar um programa “Hello, World!” . Isso nos permitirá testar nosso ambiente e nos dará a oportunidade de nos familiarizarmos com o Python.

Para fazer isso, vamos abrir um editor de texto de linha de comando como o nano e criar um novo arquivo:

```
1. nano hello.py
```

Assim que o arquivo de texto abrir na janela do terminal, vamos digitar nosso programa:

```
print("Hello, World!")
```

Saia do nano pressionando as teclas `CTRL` e `X` e, quando solicitado a salvar o arquivo, pressione `y`.

Assim que sair do nano e retornar ao seu shell, vamos executar o programa:

```
1. python hello.py
```

O programa `hello.py` que você acabou de criar deve fazer com que seu terminal produza o seguinte resultado:

```
Output
Hello, World!
```

Para sair do ambiente, digite o comando `deactivate` e você voltará para seu diretório original.

https://dev.to/womakerscode/tutorial-instalando-o-python-310-no-linux-ubuntumintdistros-derivadas-do-debian-pelo-terminal-39ab

https://www.digitalocean.com/community/tutorials/how-to-install-python-3-and-set-up-a-programming-environment-on-an-ubuntu-20-04-server-pt

Configurar um aplicativo Flask

Agora que você está em seu ambiente virtual, podemos instalar o Flask e o Gunicorn e começar a projetar nosso aplicativo:

Instale o Flask e o Gunicorn

Podemos usar a instância local de `pip`para instalar o Flask e o Gunicorn. Digite os seguintes comandos para obter esses dois componentes:

```
pip install gunicorn flask
```

Criar um aplicativo de exemplo

Agora que temos o Flask disponível, podemos criar uma aplicação simples. Flask é um micro-framework. Ele não inclui muitas das ferramentas que estruturas mais completas podem incluir e existe principalmente como um módulo que você pode importar para seus projetos para ajudá-lo a inicializar um aplicativo da web.

Embora seu aplicativo possa ser mais complexo, criaremos nosso aplicativo Flask em um único arquivo, que chamaremos de `myproject.py`:

```
nano ~/myproject/myproject.py
```

Dentro deste arquivo, colocaremos nosso código de aplicação. Basicamente, precisamos importar o frasco e instanciar um objeto Flask. Podemos usar isso para definir as funções que devem ser executadas quando uma rota específica é solicitada. Chamaremos nosso aplicativo Flask no código `application`para replicar os exemplos encontrados na especificação WSGI:

```
from flask import Flask
application = Flask(__name__)

@application.route("/")
def hello():
    return "<h1 style='color:blue'>Hello There!</h1>"

if __name__ == "__main__":
    application.run(host='0.0.0.0')
```

cópia de

Isso basicamente define qual conteúdo apresentar quando o domínio raiz for acessado. Salve e feche o arquivo quando terminar.

Você pode testar seu aplicativo Flask digitando:

```
python myproject.py
```

Visite o nome de domínio ou endereço IP do seu servidor seguido pelo número da porta especificado na saída do terminal (provavelmente `:5000`) em seu navegador da web. Você deve ver algo assim:

Quando terminar, pressione CTRL-C na janela do terminal algumas vezes para parar o servidor de desenvolvimento do Flask.

Criar o ponto de entrada WSGI

Em seguida, criaremos um arquivo que servirá como ponto de entrada para nosso aplicativo. Isso dirá ao nosso servidor Gunicorn como interagir com o aplicativo.

Chamaremos o arquivo `wsgi.py`:

```
nano ~/myproject/wsgi.py
```

O arquivo é incrivelmente simples, podemos simplesmente importar a instância do Flask do nosso aplicativo e executá-lo:

```
from myproject import application

if __name__ == "__main__":
    application.run()
```

cópia de

Salve e feche o arquivo quando terminar.

Testando a capacidade da Gunicorn de servir ao projeto

Antes de prosseguir, devemos verificar se Gunicorn pode corretamente.

Podemos fazer isso simplesmente passando o nome do nosso ponto de entrada. Também especificaremos a interface e a porta a serem vinculadas para que seja iniciada em uma interface disponível publicamente:

```
cd ~/myproject

gunicorn --bind 0.0.0.0:8000 wsgi
```

Se você visitar o nome de domínio ou endereço IP do seu servidor com um `:8000`anexo no final do seu navegador da Web, você deverá ver uma página parecida com esta:

Quando você confirmar que está funcionando corretamente, pressione CTRL-C na janela do terminal.

Agora terminamos nosso ambiente virtual, para que possamos desativá-lo:

```
deactivate
```

Qualquer operação agora será feita no ambiente Python do sistema.

Crie um script de inicialização

A próxima parte que precisamos cuidar é o script Upstart. Criar um script Upstart permitirá que o sistema init do Ubuntu inicie automaticamente o Gunicorn e sirva nosso aplicativo Flask sempre que o servidor inicializar.

Crie um arquivo de script que termine com `.conf`dentro do `/etc/init`diretório para começar:

```
sudo nano /etc/init/myproject.conf
```

Dentro, começaremos com uma descrição simples do propósito do script. Em seguida, definiremos as condições em que esse script será iniciado e interrompido pelo sistema. Os números normais de tempo de execução do sistema são 2, 3, 4 e 5, portanto, diremos a ele para iniciar nosso script quando o sistema atingir um desses níveis de execução. Diremos para parar em qualquer outro nível de execução (como quando o servidor está reiniciando, desligando ou no modo de usuário único):

```
description "Gunicorn application server running myproject"

start on runlevel [2345]

stop on runlevel [!2345]
```

Diremos ao sistema init que ele deve reiniciar o processo se ele falhar. Em seguida, precisamos definir o usuário e o grupo em que o Gunicorn deve ser executado. Nossos arquivos de projeto são todos de propriedade de nossa própria conta de usuário, portanto, nos definiremos como o usuário a ser executado. O servidor Nginx é executado no `www-data`grupo. Precisamos que o Nginx seja capaz de ler e gravar no arquivo de soquete, então daremos a este grupo a propriedade sobre o processo:

```
description "Gunicorn application server running myproject"

start on runlevel [2345]

stop on runlevel [!2345]



respawn

setuid user

setgid www-data
```

Em seguida, precisamos configurar o processo para que ele possa encontrar nossos arquivos corretamente e processá-los. Instalamos todos os nossos componentes Python em um ambiente virtual, portanto, precisamos definir uma variável de ambiente com isso como nosso caminho. Também precisamos mudar para nosso diretório de projetos. Depois, podemos simplesmente chamar o aplicativo Gunicorn com as opções que gostaríamos de usar.

Diremos a ele para iniciar 3 processos de trabalho (ajuste isso conforme necessário). Também diremos a ele para criar e vincular a um arquivo de soquete Unix dentro de nosso diretório de projeto chamado `myproject.sock`. Definiremos um valor de umask `007`para que o arquivo de soquete seja criado dando acesso ao proprietário e ao grupo, enquanto restringe outros acessos. Finalmente, precisamos passar o nome do arquivo do ponto de entrada WSGI:

```
description "Gunicorn application server running myproject"



start on runlevel [2345]

stop on runlevel [!2345]



respawn

setuid user

setgid www-data



env PATH=/home/user/myproject/myprojectenv/bin

chdir /home/user/myproject

exec gunicorn --workers 3 --bind unix:myproject.sock -m 007 wsgi
```

Salve e feche o arquivo quando terminar.

Você pode iniciar o processo imediatamente digitando:

```
sudo start myproject
```

Configurando Nginx para Solicitações de Proxy

Nosso servidor de aplicativos Gunicorn agora deve estar funcionando, aguardando solicitações no arquivo de soquete no diretório do projeto. Precisamos configurar o Nginx para passar solicitações da web para esse soquete fazendo algumas pequenas adições ao seu arquivo de configuração.

Comece criando um novo arquivo de configuração de bloco de servidor no `sites-available`diretório do Nginx. Vamos simplesmente chamar isso `myproject`para nos mantermos alinhados com o restante do guia:

```
sudo nano /etc/nginx/sites-available/myproject
```

Abra um bloco de servidor e diga ao Nginx para escutar na porta padrão 80. Também precisamos dizer a ele para usar este bloco para solicitações de nome de domínio ou endereço IP do nosso servidor:

```
server {

    listen 80;

    server_name server_domain_or_IP;

}
```

A única outra coisa que precisamos adicionar é um bloco de localização que corresponda a todas as solicitações. Dentro deste bloco, incluiremos o `proxy_params`arquivo que especifica alguns parâmetros gerais de proxy que precisam ser definidos. Em seguida, passaremos as solicitações para o soquete que definimos usando a `proxy_pass`diretiva:

```
server {

    listen 80;

    server_name server_domain_or_IP;



    location / {

        include proxy_params;

        proxy_pass http://unix:/home/user/myproject/myproject.sock;

    }

}
```

Na verdade, isso é tudo o que precisamos para servir nosso aplicativo. Salve e feche o arquivo quando terminar.

Para habilitar a configuração do bloco do servidor Nginx que acabamos de criar, vincule o arquivo ao `sites-enabled`diretório:

```
sudo ln -s /etc/nginx/sites-available/myproject /etc/nginx/sites-enabled
```

Com o arquivo nesse diretório, podemos testar erros de sintaxe digitando:

```
sudo nginx -t
```

Se isso retornar sem indicar nenhum problema, podemos reiniciar o processo Nginx para ler nossa nova configuração:

```
sudo service nginx restart
```

Agora você deve conseguir acessar o nome de domínio ou endereço IP do seu servidor no navegador da Web e ver seu aplicativo:

https://www.digitalocean.com/community/tutorials/how-to-serve-flask-applications-with-gunicorn-and-nginx-on-ubuntu-14-04

https://www.digitalocean.com/community/tutorials/how-to-serve-flask-applications-with-uswgi-and-nginx-on-ubuntu-18-04-pt

Como servir os aplicativos Flask com o Gunicorn e o Nginx no Ubuntu 18.04

Introdução

Neste guia, você construirá um aplicativo Python usando o microframework Flask no Ubuntu 18.04. A maior parte deste artigo será sobre como configurar o servidor do aplicativo Gunicorn, como iniciar o aplicativo e configurar o Nginx para atuar como um proxy reverso no front-end.

Pré-requisitos

Antes de iniciar este guia, você deve ter:

- Um servidor com o Ubuntu 18.04 instalado e um usuário não raiz com privilégios sudo. Siga nosso guia de configuração inicial do servidor para orientação.
- O Nginx instalado, seguindo os Passos 1 e 2 de Como Instalar o Nginx no Ubuntu 18.04.
- Um nome de domínio configurado para apontar para o seu servidor. Você pode comprar um no Namecheap ou obter um de graça no Freenom. Você pode aprender como apontar domínios para o DigitalOcean seguindo a relevante documentação para domínios e DNS. Certifique-se de criar os seguintes registros DNS:
  - Um registro com o `your_domain <^>`apontando para o endereço IP público do seu servidor.
  - Um registro A com `www.your_domain` apontando para o endereço IP público do seu servidor.
- Familiarize-se com a especificação do WSGI, que o servidor do Gunicorn usará para se comunicar com seu aplicativo Flask. Esta discussãoaborda mais detalhadamente o WSGI.

Passo 1 — Instalando os componentes dos repositórios do Ubuntu

Nosso primeiro passo será instalar todas as partes que precisamos dos repositórios do Ubuntu. Isso inclui o `pip`, o gerenciador de pacotes Python que irá gerenciar nossos componentes Python. Também vamos obter os arquivos de desenvolvimento do Python necessários para construir alguns dos componentes do Gunicorn.

Primeiramente, vamos atualizar o índice local de pacotes e instalar os pacotes que irão nos permitir construir nosso ambiente Python. Estes incluem o `python3-pip`, junto com alguns outros pacotes e ferramentas de desenvolvimento necessários para um ambiente de programação robusto:

```
1. sudo apt update
2.
3. sudo apt install python3-pip python3-dev build-essential libssl-dev libffi-dev
   python3-setuptools
4.
```

Com esses pacotes instalados, vamos seguir em frente para criar um ambiente virtual para nosso projeto.

Passo 2 — Criando um Ambiente Virtual em Python

Em seguida, vamos configurar um ambiente virtual para isolar nosso aplicativo Flask dos outros arquivos Python no sistema.

Inicie instalando o pacote `python3-venv`, que instalará o módulo `venv`:

```
1. sudo apt install python3-venv
2.
```

Em seguida, vamos fazer um diretório pai para nosso projeto Flask. Acesse o diretório após criá-lo:

```
1. mkdir ~/myproject
2.
3. cd ~/myproject
4.
```

Crie um ambiente virtual para armazenar os requisitos Python do projeto Flask digitando:

```
1. python3.6 -m venv myprojectenv
2.
```

Isso instalará uma cópia local do Python e do `pip` para um diretório chamado `myprojectenv` dentro do diretório do seu projeto.

Antes de instalar aplicativos no ambiente virtual, você precisa ativá-lo. Faça isso digitando:

```
1. source myprojectenv/bin/activate
2.
```

Seu prompt mudará para indicar que você agora está operando no ambiente virtual. Ele se parecerá com isso: `(myprojectenv)user@host:~/myproject$`.

Passo 3 — Configurando um aplicativo Flask

Agora que você está no seu ambiente virtual, instale o Flask e o Gunicorn e comece a projetar seu aplicativo.

Primeiramente, vamos instalar o `wheel` com a instância local do `pip` para garantir que nossos pacotes sejam instalados mesmo se estiverem faltando arquivos wheel:

```
1. pip install wheel
2.
```

Nota: Independentemente da versão do Python que você estiver usando, quando o ambiente virtual for ativado, você deve usar o comando `pip` (não o `pip3`).

Em seguida, vamos instalar o Flask e o Gunicorn:

```
1. pip install gunicorn flask
2.
```

Criando um app de exemplo

Agora que você tem o Flask disponível, você pode criar um aplicativo simples. O Flask é um microframework. Ele não inclui muitas das ferramentas que os frameworks mais completos talvez tenham. Ele existe, principalmente, como um módulo que você pode importar para seus projetos para ajudá-lo na inicialização de um aplicativo Web.

Embora seu aplicativo possa ser mais complexo, vamos criar nosso app Flask em um único arquivo, chamado `myproject.py:<^>`

```
1. nano ~/myproject/myproject.py
2.
```

O código do aplicativo ficará neste arquivo. Ele importará o Flask e instanciará um objeto Flask. Você pode usar isto para definir as funções que devem ser executadas quando uma rota específica for solicitada:

~/myproject/myproject.py

```
from flask import Flask
app = Flask(__name__)

@app.route("/")
def hello():
    return "<h1 style='color:blue'>Hello There!</h1>"

if __name__ == "__main__":
    app.run(host='0.0.0.0')
```

Isso define basicamente qual conteúdo apresentar quando o domínio raiz for acessado. Salve e feche o arquivo quando você terminar.

Se você seguiu o guia de configuração inicial do servidor, você deverá ter um firewall UFW ativado. Para testar o aplicativo, será necessário permitir o acesso à porta `5000`:

```
1. sudo ufw allow 5000
2.
```

Agora é possível testar seu app Flask digitando:

```
1. python myproject.py
2.
```

Você verá um resultado como o seguinte, incluindo um aviso útil lembrando para não usar essa configuração de servidor na produção:

```
Output
* Serving Flask app "myproject" (lazy loading)
 * Environment: production
   WARNING: Do not use the development server in a production environment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
```

Visite o endereço IP do seu servidor seguido de `:5000` no seu navegador Web:

```
http://your_server_ip:5000
```

Você deve ver algo como isto:

Quando terminar, tecle `CTRL-C` na janela do seu terminal para parar o servidor de desenvolvimento Flask.

Criando o ponto de entrada da WSGI

Em seguida, vamos criar um arquivo que servirá como o ponto de entrada para nosso aplicativo. Isso dirá ao nosso servidor do Gunicorn como interagir com o aplicativo.

Vamos chamar o arquivo de `wsgi.py`:

```
1. nano ~/myproject/wsgi.py
2.
```

Neste arquivo, vamos importar a instância Flask do nosso aplicativo e então executá-lo:

~/myproject/wsgi.py

```
from myproject import app

if __name__ == "__main__":
    app.run()
```

Salve e feche o arquivo quando você terminar.

Passo 4 — Configurando o Gunicorn

Seu aplicativo agora está gravado com um ponto de entrada estabelecido. Podemos agora seguir em frente para configurar o Gunicorn.

Antes de continuar, devemos verificar se o Gunicorn pode atender o aplicativo corretamente.

Podemos fazer essa verificação simplesmente passando o nome do nosso ponto de entrada para o Gunicorn. Criamos esse ponto de entrada como o nome do módulo (menos a extensão `.py`) mais o nome do objeto callable dentro do aplicativo. No nosso caso, trata-se do `wsgi:app`.

Também vamos especificar a interface e a porta a vincular, de modo que o aplicativo seja iniciado em uma interface disponível publicamente:

```
1. cd ~/myproject
2.
3. gunicorn --bind 0.0.0.0:5000 wsgi:app
4.
```

Deverá ver um resultado como o seguinte:

```
Output
[2018-07-13 19:35:13 +0000] [28217] [INFO] Starting gunicorn 19.9.0
[2018-07-13 19:35:13 +0000] [28217] [INFO] Listening at: http://0.0.0.0:5000 (28217)
[2018-07-13 19:35:13 +0000] [28217] [INFO] Using worker: sync
[2018-07-13 19:35:13 +0000] [28220] [INFO] Booting worker with pid: 28220
```

Visite o endereço IP do seu servidor com `:5000` anexado ao final no seu navegador Web novamente:

```
http://your_server_ip:5000
```

Você deve ver o resultado do seu aplicativo:

Quando você tiver confirmado que ele está funcionando corretamente, pressione `CTRL-C` na janela do seu terminal.

Acabamos agora o nosso ambiente virtual, para que possamos desativá-lo:

```
1. deactivate
2.
```

Agora, qualquer comando Python voltará a usar o ambiente do sistema Python.

Em seguida, vamos criar o arquivo da unidade de serviço systemd. Criar um arquivo de unidade systemd permitirá que o sistema init do Ubuntu inicie automaticamente o Gunicorn e atenda o aplicativo Flask sempre que o servidor inicializar.

Crie um arquivo de unidade que termine com `.service` dentro do diretório `/etc/systemd/system para começar`:

```
1. sudo nano /etc/systemd/system/myproject.service
2.
```

Ali, vamos começar com a seção `[Unit]`, que é usada para especificar os metadados e dependências. Vamos colocar uma descrição do nosso serviço aqui e dizer ao sistema init para iniciar isso somente após o objetivo da rede ter sido alcançado:

/etc/systemd/system/myproject.service

```
[Unit]
Description=Gunicorn instance to serve myproject
After=network.target
```

Em seguida, vamos abrir a seção `[Service]`. Isso especificará o usuário e o grupo sob o qual que queremos que o processo seja executado. Vamos dar à nossa conta de usuário regular a propriedade sobre o processo, uma vez que ela possui todos os arquivos relevantes. Vamos também dar a propriedade sobre o grupo para o grupo `www-data`, de modo que o Nginx possa se comunicar facilmente com os processos do Gunicorn. Lembre-se de substituir o nome de usuário abaixo pelo seu nome de usuário:

/etc/systemd/system/myproject.service

```
[Unit]
Description=Gunicorn instance to serve myproject
After=network.target

[Service]
User=sammy
Group=www-data
```

Em seguida, vamos mapear o diretório de trabalho e definir a variável de ambiente `PATH` para que o sistema init saiba que os executáveis do processo estão localizados dentro do nosso ambiente virtual. Vamos também especificar o comando para iniciar o serviço. Este comando fará o seguinte:

- Iniciar três processos de trabalho (embora deva ajustar isso conforme necessário)
- Criar e vincular a um arquivo de socket Unix, `myproject<^>.sock`, dentro de nosso diretório de projeto. Vamos definir um valor de umask de `007` para que o arquivo socket seja criado dando acesso ao proprietário e ao grupo, ao mesmo tempo que restringe outros acessos
- Especificar o nome do arquivo de ponto de entrada da WSGI, junto com o objeto callable do Python dentro daquele arquivo (`wsgi:app`)

O systemd exige que seja dado o caminho completo para o executável do Gunicorn, que está instalado dentro do nosso ambiente virtual.

Lembre-se de substituir o nome de usuário e os caminhos do projeto por seus próprios dados:

/etc/systemd/system/myproject.service

```
[Unit]
Description=Gunicorn instance to serve myproject
After=network.target

[Service]
User=sammy
Group=www-data
WorkingDirectory=/home/sammy/myproject
Environment="PATH=/home/sammy/myproject/myprojectenv/bin"
ExecStart=/home/sammy/myproject/myprojectenv/bin/gunicorn --workers 3 --bind
unix:myproject.sock -m 007 wsgi:app
```

Finalmente, vamos adicionar uma seção `[Install]`. Isso dirá ao systemd ao que vincular este serviço se nós o habilitarmos para iniciar na inicialização. Queremos que este serviço comece quando o sistema regular de vários usuários estiver funcionando:

/etc/systemd/system/myproject.service

```
[Unit]
Description=Gunicorn instance to serve myproject
After=network.target

[Service]
User=sammy
Group=www-data
WorkingDirectory=/home/sammy/myproject
Environment="PATH=/home/sammy/myproject/myprojectenv/bin"
ExecStart=/home/sammy/myproject/myprojectenv/bin/gunicorn --workers 3 --bind
unix:myproject.sock -m 007 wsgi:app

[Install]
WantedBy=multi-user.target
```

Com isso, nosso arquivo de serviço systemd está completo. Salve e feche-o agora.

Podemos agora iniciar o serviço Gunicorn que criamos e habilitá-lo para que ele seja iniciado na inicialização:

```
1. sudo systemctl start myproject
2.
3. sudo systemctl enable myproject
4.
```

Vamos verificar o status:

```
1. sudo systemctl status myproject
2.
```

Você deve ver um resultado como este:

```
Output
● myproject.service - Gunicorn instance to serve myproject
   Loaded: loaded (/etc/systemd/system/myproject.service; enabled; vendor preset:
enabled)
   Active: active (running) since Fri 2018-07-13 14:28:39 UTC; 46s ago
 Main PID: 28232 (gunicorn)
    Tasks: 4 (limit: 1153)
   CGroup: /system.slice/myproject.service
           ├─28232 /home/sammy/myproject/myprojectenv/bin/python3.6
/home/sammy/myproject/myprojectenv/bin/gunicorn --workers 3 --bind unix:myproject.sock
-m 007
           ├─28250 /home/sammy/myproject/myprojectenv/bin/python3.6
/home/sammy/myproject/myprojectenv/bin/gunicorn --workers 3 --bind unix:myproject.sock
-m 007
           ├─28251 /home/sammy/myproject/myprojectenv/bin/python3.6
/home/sammy/myproject/myprojectenv/bin/gunicorn --workers 3 --bind unix:myproject.sock
-m 007
           └─28252 /home/sammy/myproject/myprojectenv/bin/python3.6
/home/sammy/myproject/myprojectenv/bin/gunicorn --workers 3 --bind unix:myproject.sock
-m 007
```

Se encontrar erros, certifique-se de resolvê-los antes de continuar com o tutorial.

Passo 5 — Configurando o Nginx para solicitações de proxy

Nosso servidor do aplicativo Gunicorn deve estar funcionando agora, esperando pedidos no arquivo de socket no diretório do projeto. Vamos agora configurar o Nginx para passar pedidos Web para aquele socket, fazendo algumas pequenas adições ao seu arquivo de configuração.

Comece criando um novo arquivo de configuração do bloco do servidor no diretório `sites-available` do Nginx. Vamos chamá-lo de `myproject` para mantê-lo alinhado com o resto do guia:

```
1. sudo nano /etc/nginx/sites-available/myproject
2.
```

Abra um bloco de servidor e diga ao Nginx para escutar na porta padrão `80`. Vamos também dizer a ele para usar este bloco para pedidos para o nome de domínio do nosso servidor:

/etc/nginx/sites-available/myproject

```
server {
    listen 80;
    server_name your_domain www.your_domain;
}
```

Em seguida, vamos adicionar um bloco de localização que corresponda a cada pedido. Dentro deste bloco, vamos incluir o arquivo `proxy_params` que especifica alguns parâmetros gerais de proxy que precisam ser configurados. Vamos então passar os pedidos para o socket que definimos usando a diretriz `proxy_pass`:

/etc/nginx/sites-available/myproject

```
server {
    listen 80;
    server_name your_domain www.your_domain;

    location / {
        include proxy_params;
        proxy_pass http://unix:/home/sammy/myproject/myproject.sock;
    }
}
```

Salve e feche o arquivo quando terminar.

Para habilitar a configuração do bloco do servidor Nginx que acabou de criar, vincule o arquivo ao diretório `sites-enabled`:

```
1. sudo ln -s /etc/nginx/sites-available/myproject /etc/nginx/sites-enabled
2.
```

Com o arquivo naquele diretório, você pode testar quanto a erros de sintaxe:

```
1. sudo nginx -t
2.
```

Se retornar sem indicar quaisquer problemas, reinicie o processo Nginx para ler a nova configuração:

```
sudo systemctl restart nginx
```

Finalmente, vamos ajustar o firewall novamente. Já não precisamos de acesso através da porta `5000`, então podemos remover essa regra. Podemos então conceder acesso total ao servidor Nginx:

```
sudo ufw delete allow 5000

sudo ufw allow 'Nginx Full'
```

Agora, você consegue navegar até o nome de domínio do seu servidor no seu navegador Web:

```
http://your_domain
```

Você deve ver o resultado do seu aplicativo:

Caso encontre quaisquer erros, tente verificar o seguinte:

- `sudo less /var/log/nginx/error.log`: verifica os registros de erros do Nginx.
- `sudo less /var/log/nginx/access.log`: verifica os registros de acesso do Nginx.
- `sudo journalctl -u nginx`: verifica os registros de processo do Nginx.
- `sudo journalctl -u myproject`: verifica os registros do Gunicorn do seu app Flask.

Passo 6 — Protegendo o aplicativo

Para garantir que o tráfego para seu servidor permaneça protegido, vamos obter um certificado SSL para seu domínio. Há várias maneiras de fazer isso, incluindo a obtenção de um certificado gratuito do Let's Encrypt, gerando um certificado autoassinado ou comprando algum de outro provedor e configurando o Nginx para usá-lo, seguindo os Passos 2 a 6 de Como criar um certificado SSL autoassinado para o Nginx no Ubuntu 18.04. Vamos escolher a opção um por questão de conveniência.

Primeiramente, adicione o repositório de software Ubuntu do Certbot:

```
sudo add-apt-repository ppa:certbot/certbot
```

Aperte `ENTER` para aceitar.

Instale o pacote Nginx do Certbot com o `apt`:

```
sudo apt install python-certbot-nginx
```

O Certbot oferece várias maneiras de obter certificados SSL através de plug-ins. O plug-in Nginx cuidará da reconfiguração do Nginx e recarregará a configuração sempre que necessário. Para usar este plug-in, digite o seguinte:

```
1. sudo certbot --nginx -d your_domain -d www.your_domain
2.
```

Esse comando executa o `certbot` com o plug-in `--nginx`, usando `-d` para especificar os nomes para os quais desejamos um certificado válido.

Se essa é a primeira vez que você executa o `certbot`, você será solicitado a informar um endereço de e-mail e concordar com os termos de serviço. Após fazer isso, o `certbot` se comunicará com o servidor da Let's Encrypt, executando posteriormente um desafio para verificar se você controla o domínio para o qual está solicitando um certificado.

Se tudo correr bem, o `certbot` perguntará como você quer definir suas configurações de HTTPS:

```
Output
Please choose whether or not to redirect HTTP traffic to HTTPS, removing HTTP access.
-------------------------------------------------------------------------------
1: No redirect - Make no further changes to the webserver configuration.
2: Redirect - Make all requests redirect to secure HTTPS access. Choose this for
new sites, or if you're confident your site works on HTTPS. You can undo this
change by editing your web server's configuration.
-------------------------------------------------------------------------------
Select the appropriate number [1-2] then [enter] (press 'c' to cancel):
```

Select your choice then hit `ENTER`. A configuração será atualizada e o Nginx recarregará para aplicar as novas configurações. O `certbot` será encerrado com uma mensagem informando que o processo foi concluído e onde os certificados estão armazenados:

```
Output
IMPORTANT NOTES:
 - Congratulations! Your certificate and chain have been saved at:
   /etc/letsencrypt/live/your_domain/fullchain.pem
   Your key file has been saved at:
   /etc/letsencrypt/live/your_domain/privkey.pem
   Your cert will expire on 2018-07-23. To obtain a new or tweaked
   version of this certificate in the future, simply run certbot again
   with the "certonly" option. To non-interactively renew *all* of
   your certificates, run "certbot renew"
 - Your account credentials have been saved in your Certbot
   configuration directory at /etc/letsencrypt. You should make a
   secure backup of this folder now. This configuration directory will
   also contain certificates and private keys obtained by Certbot so
   making regular backups of this folder is ideal.
 - If you like Certbot, please consider supporting our work by:

   Donating to ISRG / Let's Encrypt:   https://letsencrypt.org/donate
   Donating to EFF:                    https://eff.org/donate-le
```

Se seguiu as instruções de instalação do Nginx nos pré-requisitos, a permissão do perfil HTTP redundante não é mais necessária:

```
1. sudo ufw delete allow 'Nginx HTTP'
2.
```

Para verificar a configuração, navegue novamente para seu domínio, usando `https://`:

```
https://your_domain
```

Você deve ver novamente o resultado do seu aplicativo, junto com o indicador de segurança do seu navegador, o qual deve indicar que o site está protegido.

https://www.digitalocean.com/community/tutorials/how-to-serve-flask-applications-with-gunicorn-and-nginx-on-ubuntu-18-04-pt

Instalação PM2

A instalação também pode ser feita em uma linha que você pode usar `npm`ou `yarn`.

```
$ npm install pm2@latest -g
# OR
$ yarn global add pm2
# Done!
# Check if program runs.
$ pm2 -v
vx.x.x
$ Set-ExecutionPolicy -Scope CurrentUser -ExecutionPolicy Unrestricted
```

Após a instalação ser concluída, você pode iniciar qualquer script Python por meio do PM2 usando o comando.`$` **`pm2 start`** `<script name>`

Se você tiver vários trabalhos para executar, também recomendo nomear cada trabalho de forma significativa para que você não se confunda mais tarde. Você pode nomear cada trabalho com **`--name`** `<name>`. Por padrão, o PM2 executará seu script com Python quando você iniciar qualquer `.py`arquivo. No entanto, se você tiver várias versões do Python instaladas em sua máquina, poderá selecionar uma versão específica do Python com **`--interpreter`** `<interpreter name: node, python, python3, python3.x, ...>`.

No geral, o comando para iniciar um script Python seria assim:

```
$ pm2 start job1.py --name job1 --interpreter python3
```

Bem, isso pode ser demais para uma máquina! Portanto, existem 2 maneiras de definir seu script Python para executar periodicamente com PM2: 1. usando restart-delay 2. usando cron

Automatizar o reinício com “reiniciar-atraso”

Você pode definir seu script Python para ser executado periodicamente com **`--restart-delay`** `<xxxx ms>`opção. Dessa forma, depois que seu script Python terminar seu trabalho, ele irá esperar (dormir) por `xxxx`ms até a próxima reinicialização.

Por exemplo, se você deseja que seu script reinicie a cada 10 segundos após cada trabalho, você normalmente pode usar `while`& `time.sleep`como segue em seu Python:

```
while True:
...
   time.sleep(10)
$ pm2 start job1.py --name job1-10s-delay --interpreter python3 --restart-delay 10000
```

Você também pode usar a opção cron para agendar seu script Python com **`--cron`** `<'cron pattern'>`. Também é importante que você desative a opção de reinicialização automática do PM2 `--no-autorestart`para que ele não reinicie automaticamente após a conclusão de um trabalho e siga apenas a expressão cron.

Por exemplo, o comando para reiniciar seu script a cada 1 hora (no minuto 0) seria assim:

```
$ pm2 start .\testPm2.py --cron '0 * * * *' --no-autorestart
```

Vamos encerrar essa parte chata e ver um exemplo do mundo real juntos!

Exemplo do mundo real

Extração de dados COVID-19 globais

Suponha que você deseja monitorar e armazenar os dados dos casos COVID-19 do Worldometer a cada 5 minutos. Você pode fazer isso facilmente usando o **Beautifulsoap4** . Por exemplo, meu script Python (*getCovid19Data.py*) permitirá que você obtenha os dados do caso COVID-19 do Worldometer e armazene os dados no arquivo *CSV* (*world_corona_case.csv*).

```
import requests, datetime

from bs4 import BeautifulSoup #To install: pip3 install beautifulsoup4

url = "https://www.worldometers.info/coronavirus/"

req = requests.get(url)

bsObj = BeautifulSoup(req.text, "html.parser")

data = bsObj.find_all("div",class_ = "maincounter-number")

NumTotalCase = data[0].text.strip().replace(',', '')

NumDeaths = data[1].text.strip().replace(',', '')

NumRecovered = data[2].text.strip().replace(',', '')

TimeNow = datetime.datetime.now()

with open('world_corona_case.csv','a') as fd:

fd.write(f"{TimeNow},{NumTotalCase},{NumDeaths},{NumRecovered};")

print(f"Successfully store COVID-19 data at {TimeNow}")
```

Em vez de usar `while`loop e `time.sleep()` dentro do script ou usar cron regular para reiniciar o script a cada 5 minutos.

Você pode fazer isso usando **PM2** com **retardo de reinicialização** *(5 min = 300000 mseg)* :

```
$ pm2 start getCovid19Data.py --name covid19-5minInt restart-delay 300000
$ pm2 start getCovid19Data.py --name covid19-5minInt --cron '*/5 * * * *' --no-
autorestart
```

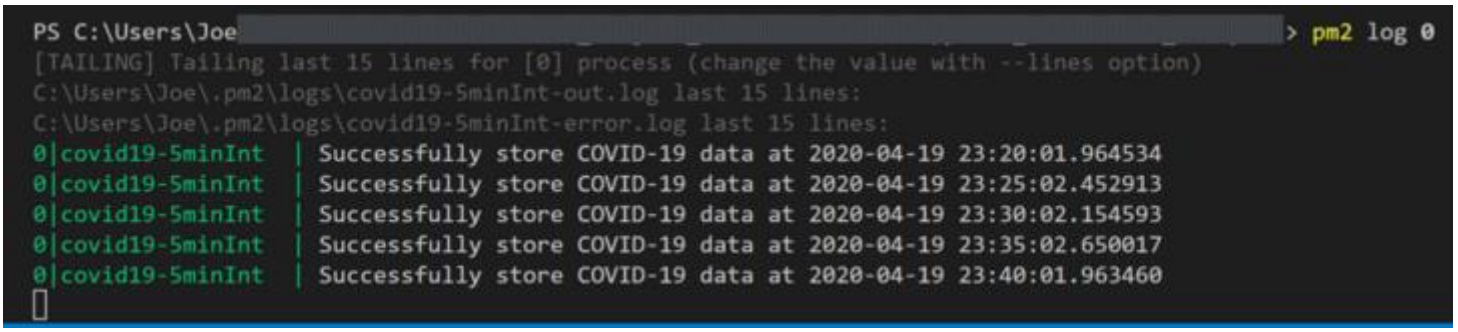
```
$ pm2 l
```

id	name	namespace	version	mode	pid	uptime	↺	status	cpu	mem	user	watching
0	covid19-5minInt	default	N/A	fork	46304	0s	0	online	17.1%	16.2mb	Joe	disabled

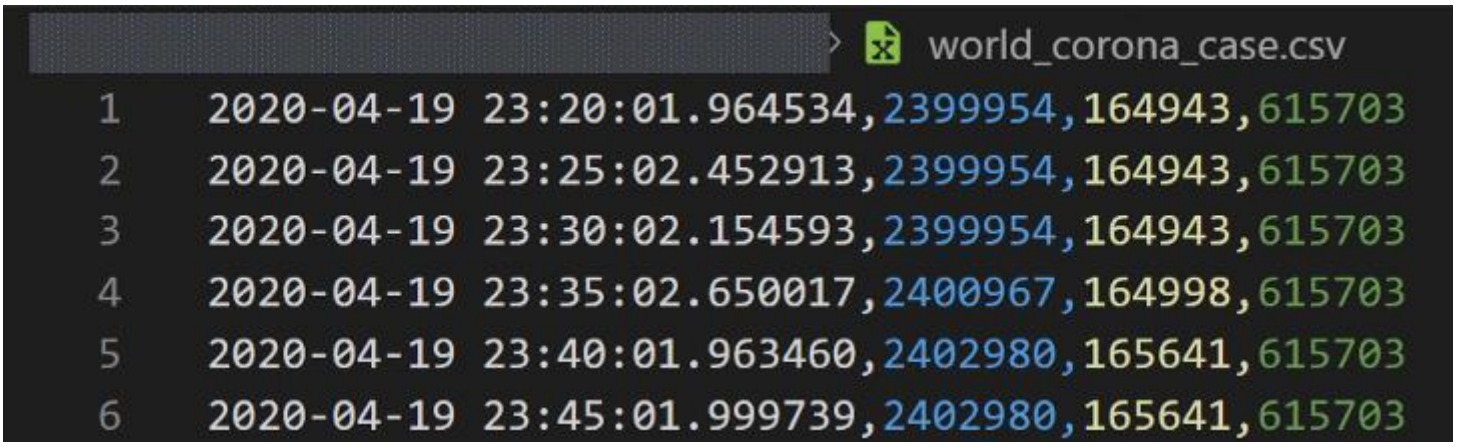
A tabela PM2 mostra uma lista de todos os aplicativos

Você pode usar um comando `pm2 log <id or name>`para ver o log de um trabalho específico. Neste exemplo, você pode usar o seguinte comando:

```
$ pm2 log 0
```



PM2 log for job-id “0”



O resultado CSV do script getCovid19Data.py usando PM2 com cron

Alguns outros comandos PM2 úteis

Quando você tem vários processos em execução com PM2. Você pode querer manipular cada projeto de maneira diferente. Aqui está uma lista de alguns comandos úteis que mais usei.

```
$ pm2 stop <name or id> #stop specific process
$ pm2 restart <name or id> #restart specific process
$ pm2 flush <name or id> #Clear logs of specific process
$ pm2 save #save list of all application
$ pm2 resurrect #brings back previously saved processes
$ pm2 startup #Command for running PM2 on startup
```

https://ichi.pro/pt/automatize-seu-script-python-com-pm2-77181517171557