Python é uma linguagem de programação excelente, fácil de aprender e poderosa, e com frameworks como Django e Flask, podemos usá-lo para criar aplicações web completas. Uma vez que criamos uma aplicação web usando bibliotecas e frameworks como Flask, precisamos hospedá-la em um servidor e expô-la ao público. Este tutorial discute como hospedar sua aplicação web em um servidor rodando Nginx usando Gunicorn. Antes de começarmos com Flask e Gunicorn, certifique-se de atender aos seguintes requisitos:

- Um servidor com acesso SSH
- Servidor web Nginx rodando no servidor (instalação coberta)
- Pitão
- Você é um usuário sudo.

# Configurando o Servidor

Vamos agora começar a configurar o ambiente do servidor que usaremos para hospedar nosso servidor. Este tutorial usa o Ubuntu Server. Comece atualizando os repositórios e instalando Python3 e Pip.

**sudo apt-get update**
**sudo apt-get upgrade -y**
**sudo apt-get install python3 python3-pip -y**

Em seguida, precisamos criar um diretório para armazenar o projeto.

**sudo mkdir / var / www / application**
**cd / var / www / application**

Altere a propriedade e as permissões do diretório:

**sudo chown -R www-data:www-data / var / www / application /**

Em seguida, instale os pacotes usando o apt (flask e Gunicorn)

**sudo apt-get install python3-flask python3-gunicorn**

Vamos agora prosseguir para inicializar um aplicativo de frasco. Comece criando o main.py — contém o aplicativo — e o wsgi.py, que fará com que o aplicativo seja executado.

**sudo touch main.py wsgi.py**

Edite o arquivo main.py e configure seu aplicativo Flask e todas as rotas. Como este tutorial não é um guia do Flask, vamos configurar uma rota básica e uma mensagem de olá mundo.

```
from flask import Flask

app = Flask ( __name__ )
@ app.route ( "/" )
def home ( ) :
   return "<h1>Nginx & Gunicorn</h1>"
```

Edite o arquivo wsgi.py e adicione o código para importar app e execute como:

```
from main import app
if __name__ == "__main__" :
   app.run ( debug =True )
```

Finalmente, teste se ele está funcionando chamando flask como:

```
$ flask run
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

**Tente acessar o aplicativo em** http://localhost:5000 .

# Configurando o Gunicorn

Uma vez que nosso aplicativo esteja rodando com sucesso, podemos usar o Gunicorn para testar o aplicativo usando os comandos:

```
$ sudo gunicorn --workers 5 wsgi:app
```

```
[ 2021 -03- 19 09: 19 : 34 +0000 ] [ 14047 ] [ INFO ] Iniciando gunicorn 20.0.4
[ 2021 -03- 19 09: 19 : 34 +0000 ] [ 14047 ] [ INFO ]Ouvindo em: http: // 127.0.0.1: 8000 ( 14047 )
[ 2021 -03- 19 09: 19 : 34 +0000 ] [ 14047 ] [ INFO ] Usando worker: sync
[ 2021 -03- 19 09: 19 : 34 +0000 ] [ 14049 ] [ INFO ]Inicializando o trabalhador com pid: 14049
[ 2021 -03- 19 09: 19 : 34 +0000 ] [ 14050 ] [ INFO ] Inicializando o trabalhador com pid: 14050
[ 2021 -03- 19 09: 19 : 34 +0000 ] [ 14051 ] [ INFO ] Inicializando o trabalhador com pid: 14051
[ 2021 -03- 19 09: 19 : 34 +0000 ] [ 14052 ] [ INFO ] Inicializando trabalhador com pid: 14052
[ 2021 -03- 19 09: 19 : 35 +0000 ] [ 14053 ] [ INFO ] Inicializando trabalhador com pid: 14053
```

Os comandos acima executam o aplicativo do frasco usando o Gunicorn usando o número especificado de trabalhadores. Em seguida, chamamos o arquivo wsgi:app, que é o arquivo e a instância do aplicativo a ser executado.

Depois de executar o aplicativo usando o Gunicorn, pressione CTRL + C para parar o servidor e configurar o Nginx.

Use os comandos abaixo para instalar e executar o Nginx.

```
sudo apt-get install nginx -y
sudo systemctl start nginx
sudo systemctl enable nginx
```

A próxima etapa é editar a configuração do Nginx no diretório habilitado para sites e adicionar o bloco do servidor. Considere a seguinte configuração. Altere o aplicativo para o nome do seu projeto.

```
sudo nano /etc/nginx/sites-available/application.conf
server {
        listen 80;
        server_name application;

        access_log /var/log/nginx/application.access.log;
        error_log /var/log/nginx/appliation.error.log;

        location / {
                include proxy_params;
                proxy_pass http://unix:/var/www/application/application.sock;
        }
}
```

Prossiga para criar um link para o diretório habilitado para site para habilitar o site.

**sudo ln -s /etc/nginx/sites-available/application.conf /etc/nginx/sites-enabled/**

Agora reinicie o serviço Nginx como:

**sudo systemctl restart nginx**

Em seguida, precisamos criar um arquivo de unidade systemd para servir o aplicativo.

**sudo nano /etc/systemd/system/application.service**
**[Unit]**
**Description=application.service - A Flask application run with Gunicorn.**
**After=network.target**

**[Service]**
**User=www-data**
**Group=www-data**
**WorkingDirectory=/var/www/application/**
**ExecStart=/usr/bin/gunicorn --workers 3 --bind unix:/var/www/application.sock wsgi:app**

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

Por fim, salve o arquivo de unidade, ative e recarregue o daemon.

**sudo systemctl restart application.service**
**sudo systemctl daemon-reload**

Agora você pode acessar o aplicativo usando o endereço IP ou o endereço especificado no arquivo de configuração do nginx. Pode ser necessário adicioná-lo ao arquivo do host.

**curl http://application**
**HTTP/1.1 200 OK**
**Server: Werkzeug/0.16.1 Python/3.8.5)**
**Date: Fri, 19 Mar 2021 10:00:39 GMT**
**Content-Type: text/html**
**Content-Length: 25**
**Last-Modified: Fri, 19 Mar 2021 09:22:47 GMT**

**https://linuxhint.com/use-nginx-with-flask/**

# 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 **Beautifulsoup4** . 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 |

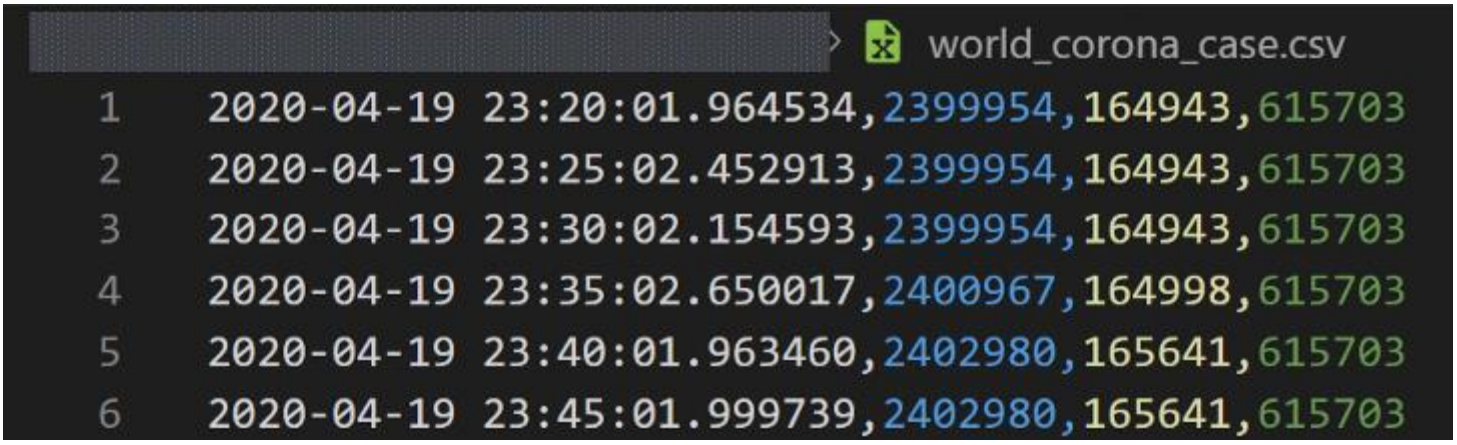
A tabela PM2 mostra uma lista de todos os aplicativos

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
```

```
PS C:\Users\Joe                                                    > pm2 log 0
[TAILING] Tailing last 15 lines for [0] process (change the value with --lines option)
C:\Users\Joe\.pm2\logs\covid19-5minInt-out.log last 15 lines:
C:\Users\Joe\.pm2\logs\covid19-5minInt-error.log last 15 lines:
0|covid19-5minInt  | Successfully store COVID-19 data at 2020-04-19 23:20:01.964534
0|covid19-5minInt  | Successfully store COVID-19 data at 2020-04-19 23:25:02.452913
0|covid19-5minInt  | Successfully store COVID-19 data at 2020-04-19 23:30:02.154593
0|covid19-5minInt  | Successfully store COVID-19 data at 2020-04-19 23:35:02.650017
0|covid19-5minInt  | Successfully store COVID-19 data at 2020-04-19 23:40:01.963460
```

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