【PythonのAPサーバー比較！！】NGINX Unit 、uWSGI、Gunicornのベンチマークを比較してみた

2018年4月に正式リリースされた軽量Web/APサーバーNGINX UnitでPythonのAPサーバーを構築し、他のメジャーなAPサーバーとベンチマークを比較してみました。

#WSGI #Nginx #Python

岩田智哉

2018.09.04

3 20 46

はじめに

サーバーレス開発部@大阪の岩田です。 NGINX Unitを使ってDjango REST Frameworkのアプリを動かしてみたので、構築手順やGunicorn、uWSGIとのベンチマークの比較についてご紹介します。

NGINX Unitとは？

NGINX UnitはNGINX社が開発した軽量Web/APサーバーです。 WebサーバーのNGINXはご存知の方も多いと思いますが、実はこんなミドルウェアもリリースされていたりします。 2018年４月に正式リリースされており、2018年9月現在で

- Python
- PHP
- Go
- Perl
- Ruby

といった言語がサポートされています。 JavaScript/Node.jsやJavaについてもcomming soonとなっており、今後サポートされていくようです。

REST API経由で無停止かつ動的に設定を投入できるのが大きな特徴です。

公式ドキュメントでは下記のように紹介されています。

> NGINX Unit is a dynamic web and application server, designed to run applications in multiple languages. Unit is lightweight, polyglot, and dynamically configured via API. The design of the server allows reconfiguration of specific application parameters as needed by the engineering or operations.

今回構築した環境

下記の環境で構築しました。

- EC2インスタンス:m5.large
- OS:AmazonLinux2 20180810
- Python:3.6.2
- Nginx Unit:1.4
- Django:2.1.1
- DjangoRestFramework:3.8.2
- Gunicorn:19.9.0
- uWSGI:2.0.17.1

構築手順

早速環境を構築してみます。

Python3.6のインストール

まずPython3.6のインストールから行います

```
sudo amazon-linux-extras install python3
```

Nginx Unitのインストール

次にNGINX Unitをインストールします。 今回はソースからインストールする方法を選択しました。

まずはビルドに必要なパッケージをインストールしておきます。

```
sudo yum install gcc  git
sudo yum install python3-devel --disablerepo=amzn2-core
```

GitHubからNGINX Unitのソースを取得して、ビルド・インストールします。インストール先は **/usr/local/unit** としました。

```
git clone https://github.com/nginx/unit
cd unit
./configure --prefix=/usr/local/unit
./configure python --config=/usr/bin/python3.6-config
make all
sudo make install
```

インストールしたNGINX Unitをsystemd管理下に置きたいので、下記のファイルを作成します。

/usr/lib/systemd/system/unit.service

```
[Unit]
Description=NGINX Unit
Wants=network-online.target
After=network-online.target

[Service]
Type=forking
PIDFile=/run/unit.pid
EnvironmentFile=-/etc/sysconfig/unit
ExecStart=/usr/local/unit/sbin/unitd $UNITD_OPTIONS
ExecReload=

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

/etc/sysconfig/unit

```
UNITD_OPTIONS="--log /var/log/unit.log --pid /run/unit.pid"
```

ファイルを作成できたらNGINX Unitを起動してみます。

```
sudo systemctl daemon-reload
sudo systemctl start unit
```

ステータスを確認してみます。

```
sudo systemctl status unit
```

```
● unit.service - NGINX Unit
   Loaded: loaded (/usr/lib/systemd/system/unit.service; disabled; vendor preset
   Active: active (running) since 日 2018-09-02 11:35:29 UTC; 1s ago
  Process: 2548 ExecStart=/usr/local/unit/sbin/unitd $UNITD_OPTIONS (code=exited,
 Main PID: 2549 (unitd)
   CGroup: /system.slice/unit.service
           ├─2549 unit: main v1.4 [/usr/local/unit/sbin/unitd --log /var/log/unit
           ├─2551 unit: controller
           └─2552 unit: router

9月 02 11:35:29 ip-172-31-35-222.ap-northeast-1.compute.internal systemd[1]: Sta
9月 02 11:35:29 ip-172-31-35-222.ap-northeast-1.compute.internal unitd[2548]: 20
9月 02 11:35:29 ip-172-31-35-222.ap-northeast-1.compute.internal systemd[1]: Sta
```

OKです

Django REST frameworkのQuickstartアプリ作成

次にNGINX Unitで動かすDjango REST frameworkのアプリを作成します。 venvを使いつつ、**公式のチュートリアル**の手順に従って構築します。

```
cd /home/ec2-user
python3 -m venv django
cd django
source bin/activate
pip install django djangorestframework gunicorn uwsgi
django-admin.py startproject tutorial .
cd tutorial
django-admin.py startapp quickstart
cd ..
python manage.py migrate
```

ここまでできたら、チュートリアルの手順通りにコードを追加し、ユーザー一覧取得のAPIが動くところまで持っていきます。 一通りコードを書き終えたら `python manage.py runserver` でWebサーバーを起動し、APIが動作することを確認しておいて下さい。

NGINX Unitに設定投入

次にNGINX Unit上でDjangoアプリが動くよう、設定を投入します。 まず下記のようなJSONファイルを用意します。 重要なのが `home` というパラメータで、ココでPythonの仮想環境(virtual environment)を指定することができます。

```
{
    "listeners": {
        "*:8000": {
            "application": "django"
        }
    },
    "applications": {
        "django": {
            "type": "python3.6",
            "processes": 2,
            "path": "/home/ec2-user/django/",
            "home": "/home/ec2-user/django/",
            "module": "tutorial.wsgi"
        }
    }
}
```

ファイルが準備できたらcurlコマンドを使い、Unixソケット経由で設定を投入します。

```
sudo curl -XPUT -d @/home/ec2-user/django/unit.json --unix-socket /usr/local/unit
```

すると

```
{
    "error": "Failed to apply new configuration."
}
```

・・・エラーになりました。 ログを見てみます。

`/var/log/unit.log`

```
Current thread 0x00007fd73dcf12c0 (most recent call first):
2018/09/02 06:25:25 [alert] 9048#9048 process 9061 exited on signal 6
2018/09/02 06:25:25 [warn] 9051#9051 failed to start application "django"
2018/09/02 06:25:25 [alert] 9051#9051 failed to apply new conf
2018/09/02 06:25:50 [info] 9065#9065 "django" application started
Fatal Python error: Py_Initialize: Unable to get the locale encoding
ModuleNotFoundError: No module named 'encodings'
```

モジュールが読み込めておらず、仮想環境がうまく認識できていなさそうです。 色々調べた結果、 **user** の指定を追加することでうまく動くようになりました。 設定追加後のJSONは下記の通りです。

```
{
    "listeners": {
        "*:8000": {
            "application": "django"
        }
    },
    "applications": {
        "django": {
            "type": "python3.6",
            "processes": 2,
            "path": "/home/ec2-user/django/",
            "home": "/home/ec2-user/django/",
            "module": "tutorial.wsgi",
            "user": "ec2-user",
            "group": "ec2-user"
        }
    }
}
```

改めてcurlで設定を投入すると・・・

```
{
    "success": "Reconfiguration done."
}
```

今度はOKそうです！！ curlでAPIにアクセスして試してみます。

```
curl http://localhost:8000/users/
```

[{"url":"http://localhost:8000/users/1/","username":"admin","email":"admin@examp

レスポンスが返ってきました。 OKそうですね。

ベンチマーク

せっかくなので、PythonのAPサーバーとしてよく利用されるGunicorn、uWSGIと比較してみました。 NGINX Unitを導入したEC2と同一サブネット上に負荷掛け用のクライアントとしてEC2を追加で構築し、何パターンか設定を変えつつ、abコマンドでベンチマークを行いました。 abコマンドはそれぞれ5回づつ実行し、5回の平均値を結果として採用しています。

各APサーバーの設定

それぞれ利用したベースの設定は下記の通りです。

NGINX Unitの設定

```
{
    "listeners": {
        "*:8000": {
            "application": "django"
        }
    },
    "applications": {
        "django": {
            "type": "python3.6",
            "processes": 1,
            "path": "/home/ec2-user/django/",
            "home": "/home/ec2-user/django/",
            "module": "tutorial.wsgi",
            "user": "ec2-user",
            "group": "ec2-user"
        }
    }
}
```

計測パターンに応じて **processes** を変更しています。

uWSGIの設定

uwsgi.ini

```
[uwsgi]
http-socket = 0.0.0.0:8000
module = tutorial.wsgi:application
processes = 1
threasds = 1
master = 1
max-requests = 100000
```

`processes` 、 `threasds` の値は計測パターンに応じて変更しています。

この設定ファイルを使って、下記コマンドで起動します。

```
uwsgi --ini uwsgi.ini
```

Gunicornの設定

gunicorn.py

```
bind = '0.0.0.0:8000'
max_requests = 100000
workers = 1
threads = 1
```

`workers` 、 `threads` の値は計測パターンに応じて変更しています。

この設定ファイルを使って、下記コマンドで起動します。

```
gunicorn -c gunicorn.py tutorial.wsgi
```

おまけ

おまけで `python manage.py runserver` で起動するDjangoの開発用Webサーバーもベンチマーク対象に入れてみました。

ベンチマーク1(直列の場合)

abコマンド

```
ab -c 1 -n 10000 http://ip-172-31-35-222.ap-northeast-1.compute.internal:8000/us
```

設定

- NGINX Unit: **processes** を1に設定しました。
- uWSGI: **processes** を1に、 **threasds** を1に設定しました。
- Gunicorn: **workers** を1に、 **threads** を1に設定しました。

結果

このような結果になりました。

APサーバー	Time taken for tests	Requests per second	Time per request
uWSGI	30.941	323.196	3.094
Nginx Unit	31.1312	321.222	3.1132
Gunicorn	33.3658	299.71	3.3364
Django	36.6012	273.214	3.66

ベンチマーク2(5並列の場合　その１)

負荷掛けクライアント側の並列度を上げて再実行してみます。

abコマンド

```
ab -c 5 -n 10000 http://ip-172-31-35-222.ap-northeast-1.compute.internal:8000/us
```

設定

- NGINX Unit: **processes** を1に設定しました。
- uWSGI: **processes** を1に、 **threasds** を1に設定しました。
- Gunicorn: **workers** を1に、 **threads** を1に設定しました。

結果

wUSGIを抜いてNginx　Unitがトップになりました

APサーバー	Time taken for tests	Requests per second	Time per request
Nginx Unit	29.9578	333.804	14.979
uWSGI	30.0756	332.498	15.0378
Gunicorn	32.1208	311.33	16.0604
Django	38.3522	260.742	19.1762

ベンチマーク2(5並列の場合　その2)

負荷掛けクライアント側を並列起動するようにしたので、サーバー側も複数プロセス立ち上げるように設定を変更してみます。 m5.largeインスタンスはvCPUが2つなので、APサーバーのプロセスを2つ立ち上げる設定にしています。 また、uWSGIとGunicornのスレッド数の設定ですが、何度か試した感じuWSGI:3、Gunicorn:1あたりが性能が出たので、それぞれこの値を採用しています。

abコマンド

```
ab -c 5 -n 10000 http://ip-172-31-35-222.ap-northeast-1.compute.internal:8000/us
```

設定

- NGINX Unit: **`processes`** を2に設定しました。
- uWSGI: **`processes`** を2に、 **`threasds`** を3に設定しました。
- Gunicorn: **`workers`** を2に、 **`threads`** を1に設定しました。

結果

wUSGIがトップに返り咲きました

APサーバー	Time taken for tests	Requests per second	Time per request
uWSGI	22.7474	439.616	11.3736
Nginx Unit	23.2504	430.1	11.6254
Gunicorn	24.4636	408.778	12.2318
Django	38.3962	260.444	19.198

20並列の場合

負荷掛けクライアント側の並列度をさらに上げて再実行してみます。

abコマンド

```
ab -c 20 -n 10000 http://ip-172-31-35-222.ap-northeast-1.compute.internal:8000/us
```

設定

- NGINX Unit: **processes** を2に設定しました。
- uWSGI: **processes** を2に、 **threasds** を3に設定しました。
- Gunicorn: **workers** を2に、 **threads** を1に設定しました。

結果

5並列の場合とあまり変わりませんでした。

APサーバー	Time taken for tests	Requests per second	Time per request
uWSGI	22.8202	438.218	45.6404
Nginx Unit	23.1924	431.178	46.3848
Gunicorn	24.3002	411.532	48.6002
Django	39.0472	256.1	78.094

まとめ

今回試した条件では、性能面ではuWSGI > NGINX Unit > Gunicornとなりました。 Cで実装されているNGINX Unitが1位になってくれると期待していたので、少し意外な結果でした。

また、参考資料に記載されているのですが、Gunicornに直接httpでリクエストを送ると、レスポンスのヘッダとボディが別パケットになるそうです。 間にNginx等のWebサーバーを噛ませてリバースプロキシするような構成を取ると、もう少しGunicornが伸びそうな気はします。 また時間ができたら検証してみたいと思います。

NGINX Unitは、まだ正式リリースされてから日が浅く、comming soonとなっている機能も多くあります。 まだまだ発展途上なイメージですが、NGINX社が開発していることもあり、今後シェアを伸ばしていく可能性も十分に持ち合わせていると思います。 今後もNGINX Unitの動向を注意深く見守っていきたいと思います。