uWSGI入門

PythonでWebアプリケーションを作成した後サーバー上で稼働させる場合、アプリケーションサーバが必要となります。（開発時は組み込みのサーバーで十分ですが、本番で使用することは推奨されていません。）ここではPythonでよく使われるuWSGIというアプリケーションサーバについて学習しましょう。

Contents[hide]

uWSGIとは

WSGIとは

uWSGIについて学習する前に、WSGIについて学習しましょう。WSGI（Web Server Gateway Interface） とは、PythonのWebアプリケーションとWebサーバー間とのやり取りの規約、プロトコールのことでPEP333で定義されています。

https://www.python.org/dev/peps/pep-3333/

Pythonの大抵のWebフレームワークはこのWSGIという規約に則っています。有名どころとして以下のフレームワークが挙げられます。

Django
Flask
Bottle

uWSGIとは

uWSGIとはPythonでWebサービスを動かすためのアプリケーションサーバ(以降、APサーバーと略することがあります。)の一種です。上記のWSGIに則ったアプリケーションを動作させるアプリケーションサーバをWSGIアプリケーションコンテナやWSGIサーバなどと呼びます。uWSGIは、このWSGIアプリケーションコンテナの一種です。つまり、WSGIに準拠したアプリケーションであれば、上で挙げたDjangoやFlask以外でも動かすことが可能です。補足ですが、WSGIサーバーにはuWSGI以外にGunicornというものもあります。

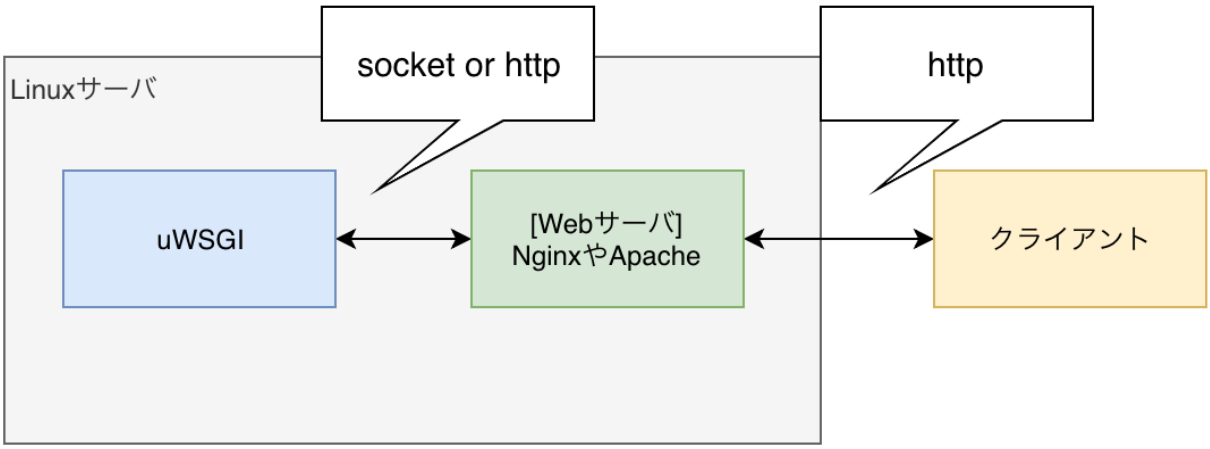
Webサーバとの連携

uWSGIは動作させたアプリケーションとWebサーバーと以下の方法で通信させることができます。

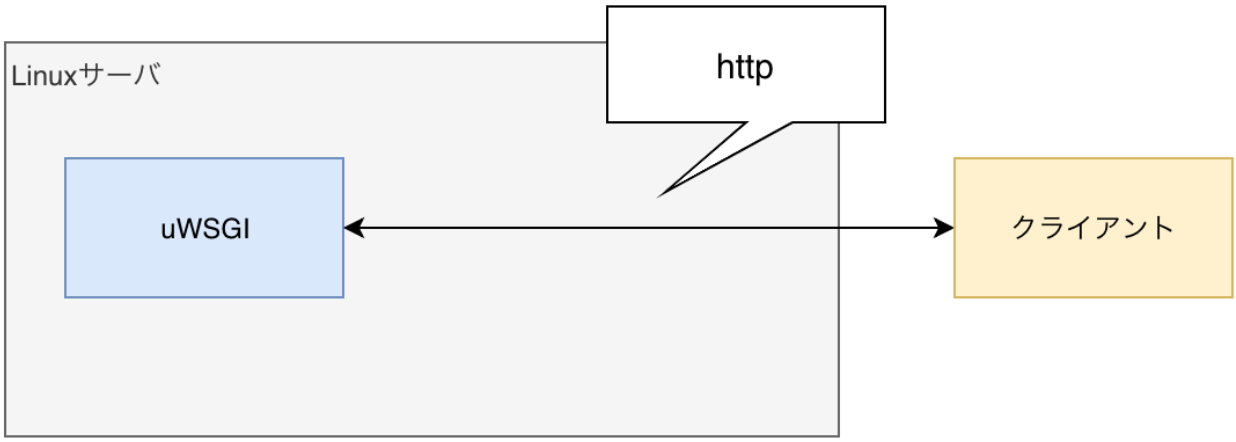
UNIXドメインソケット
HTTP

UNIXドメインソケットは高速でAPサーバーとWebサーバー間の通信を行うことが可能です。ただし、直接リクエストしたりレスポンスを参照することができないため、トラブルが起きた際には調査が難航するかもしれません。

HTTPはソケット通信より速度面で劣りますが、最近のサーバーは低価格なものでもそこそこ性能がありますので内部の通信でHTTPを使用してもそれほど問題にならないと思います。



また、静的ファイルがないAPIの場合はWebサーバーを経由せずに、直接クライアントと通信するのも一つの手でしょう。



uWSGIのインストール

Pythonがインストールされている環境であれば、pipでインストール可能です。

```
pip install uWSGI
```

uWSGIでFlaskを動かしてみる

それでは、ここから実際にWSGIに準拠したFlaskをuWSGIで動かしてみましょう。Flaskについては以下も合わせて参考にしてみてください。

Flask入門

サンプルアプリケーションを作成する

あらかじめFlaskとuWSGIをインストールしておきましょう。

```
pip install uWSGI
pip install flask
```

次にFlaskの簡単なアプリケーションを作成します。run.pyという名前で以下のプログラムを作成しましょう。

```
# run.py
from flask import Flask, jsonify

app = Flask(__name__)


@app.route('/')
def api_sample():
    """
    APIサンプル
    :return:
    """
    result = {"code": "001", "name": "apple"}
    return jsonify(ResultSet=result)


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

/にアクセスすると、jsonを返すだけのプログラムです。 `python run.py` とコマンドを実行するとデバッグ用のサーバーが起動しますので、"http://127.0.0.1:5000/"にアクセスしてください。jsonが返されればOKです。確認が終わればCtrl+c（Windows系の方はctrl+z）でサーバーを中断してください。

uWSGIコマンドで実行

次にuWSGIコマンドで実行してみましょう。以下のコマンドを実行してください。

```
uwsgi --http=0.0.0.0:8080 --wsgi-file=run.py  --callable=app
```

ブラウザやcurlコマンドでhttp://127.0.0.1:8080にアクセスしてください。先程のjsonが出力されればuWSGI経由での実行は成功です。

uWSGIコマンドではオプションにサーバーの設定を指定します。httpオプションはアクセスを許可するホストとポートを、wsgi-fileオプションではwsgiで動作するファイルを指定します。

設定ファイルから実行する

さて、実運用でサーバーを動かす場合、上記で指定したアクセス許可ホスト、ポート番号以外にも、プロセス数、スレッド数、実効ユーザー、実効グループ、リロード方式、ソケット出力先、権限、ログ出力形式、出力先、etc...などなど様々な項目を設定する必要があります。

全てコマンドのオプションで指定することができるのですが、オプションが長すぎて取り回しが悪いですね。このため、大抵のサービス系コマンドと同様、uWSGIコマンドはオプションをひとまとめに記述できる設定ファイルを指定することができます。

設定ファイルで試してみましょう。uwsgi.iniというini形式のファイルを作成します。セクション名は[uwsgi]を指定します。

```
# uwsgi.ini

[uwsgi]

# wsgiファイル
wsgi-file=run.py
callable=app

# アクセス許可ホスト:ポート
http=0.0.0.0:8080
```

設定ファイルが完成したら、以下のコマンドで実行してみましょう。

```
uwsgi uwsgi.ini
```

先程と同様、http://127.0.0.1:8080 にアクセスして出力を確認してみてください。

uWSGIの設定項目

次に、実際にサーバーとして動かす場合の設定値を紹介します。いずれもCentos等のUnix系サーバで動かすことを想定しています。

デーモン化

先程実行した際、コマンドライン上にログが出力され、コマンドラインを終了するとサービスも停止してしまいましたが、以下のオプションでデーモン化することが可能です。

```
daemonize = /var/log/uwsgi.log
log-reopen = true
log-maxsize = 8000000
logfile-chown = on
logfile-chmod = 644
```

log-reopenリロード後にログをリオープンします。log-maxsizeはログの最大サイズでそれを超過するとローテートされますが、Linuxのログローテートを使用してもよいでしょう。

また、以下のように日付をファイル名に指定することもできます。

```
daemonize = /var/log/uWSGI-@(exec://date +%%Y-%%m-%%d).log
```

ディレクトリ

上のサンプルではカレントディレクトリで実行しましたが、chdirでWSGIアプリケーションを配置したディレクトリを設定で指定することができます。

例えば、/opt/apps/current/sample_apiというディレクトリにアプリケーションを配置した場合、以下のように記述することができます。

```
current_release = /opt/apps/current/sample_api
chdir = %(current_release)
wsgi-file=%(current_release)/run.py
```

上のcurrent_releaseは変数となっており、%(current_release)で他の項目で使い回すことができます。

実効ユーザー・グループ

実効ユーザー、グループはuid、gidで指定することができます。

```
uid = uwsgi_user
gid = uwsgi_group
```

プロセス、スレッド

processes、threadsでプロセス数、スレッド数を指定することができます。thunder-lockでリクエストを受けるプロセスを分散することができます。max-requestsで指定した回数リクエストを受けるとリロードします。また、一斉にリロードするとその間サービスが止まりますので、max-requests-deltaでリロードのタイミングにプロセスごとに差を設けます。

```
processes = 4
threads = 2
thunder-lock = true
max-requests = 3000
max-requests-delta = 300
master = True
```

uWSGIの停止、再起動、リロード

デーモン化したuwsgiを停止、リロードするにはpidファイルが必要です。設定に以下を追記します。

```
# pidファイルの位置を指定
pidfile = /var/run/uwsgi/uwsgi.pid
# 前回異常終了した場合、起動時にpidファイルをクリア
vacuum = true
```

停止

```
uwsgi --stop /var/run/uwsgi/uwsgi.pid
```

リロード

```
uwsgi --reload /var/run/uwsgi/uwsgi.pid
```

また、プロセスIDがわかっていればシグナルでも操作可能です。SIGHUPの場合、reloadと同じ動作となります。

```
kill -HUP `cat /var/run/uwsgi/uwsgi.pid`
```

今回はuWSGIについて概要を説明しました。次回はLinuxサーバー上でuWSGIをセットアップする方法について説明する予定です。