Flasync Await

David Bordeynik
Software Architect
@ Nvidia

EuroPython 2020


WHO AM I?
makeameme.org

Setting up expectations

- This session IS a mind opener on how to provide added value with minimal effort.
- This session IS NOT about saying X is a bad technology and Y is a good technology.
- Assuming knowledge in web development and REST APIs in particular.

Motivation

```python
from flask import Flask

app = Flask(__name__)


@app.route("/")
def hello():
    return {"hello": "world"}


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

3-4x

```python
from sanic import Sanic
from sanic.response import json

app = Sanic(__name__)


@app.route("/")
async def hello(_):
    return json({"hello": "world"})


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

Motivation - cont.

```
Concurrency Level:      100
Time taken for tests:   13.247 seconds
Complete requests:      10000
Failed requests:        0
Total transferred:      1630000 bytes
HTML transferred:       180000 bytes
Requests per second:    754.88 [#/sec] (mean)
Time per request:       132.471 [ms] (mean)
Time per request:       1.325 [ms] (mean, across all
Transfer rate:          120.16 [Kbytes/sec] received
```

3-4x

```
Concurrency Level:      100
Time taken for tests:   4.000 seconds
Complete requests:      10000
Failed requests:        0
Total transferred:      1070000 bytes
HTML transferred:       170000 bytes
Requests per second:    2499.90 [#/sec] (mean)
Time per request:       40.002 [ms] (mean)
Time per request:       0.400 [ms] (mean, across all
Transfer rate:          261.22 [Kbytes/sec] received
```

Notes on the motivation experiment

- simplejson is installed as an optional dependency for flask.
- ab is used for benchmarking.

Flask

A micro web framework that revolutionized how web is developed with python.

Asyncio

Library to write concurrent IO-bound* code using the async/await syntax.

* Example for IO-bound: http requests ; example for CPU-bound: compression.

Asyncio - cont.

Why asyncio? what's wrong with thread / process per request?

Currently, we consume more HTTP based services than ever.

=> We easily reach 10k connections concurrently on a single server (AKA c10k problem).

=> cooperative tasks that can better utilize a CPU can save a lot of $$$.

Sanic

Python 3.6+ web server & web framework that's written to go fast using the async/await syntax.

Introducing pyaday

- CRUD for python packages metadata used by content curators.
- Get your daily random python package metadata for fun and profit.

Introducing pyaday - cont.

```
> http http://127.0.0.1:5000/rand
HTTP/1.0 200 OK
Content-Length: 97
Content-Type: application/json
Date: Thu, 09 Jul 2020 11:52:28 GMT
Server: Werkzeug/1.0.1 Python/3.8.2

{
    "name": "poetry",
    "short_desc": "Python dependency management and packaging made easy."
}
```

Introducing pyaday - cont.

```python
from flask import Flask

from pyaday.api.packages import packages_bp
from pyaday.api.rand import rand_bp

app = Flask(__name__)
app.register_blueprint(blueprint=packages_bp,
                       url_prefix="/packages")
app.register_blueprint(blueprint=rand_bp, url_prefix="/rand")

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

Introducing pyaday - cont.

```python
import json

from flask import Blueprint, Response, request
from werkzeug.datastructures import Headers

packages_bp = Blueprint(name="packages_bp", import_name=__name__)

python_packages = [
    {
        "name": "flask",
        "short_desc": "A simple framework for building complex web applications.",
    },
    {
        "name": "sanic",
        "short_desc": " A web server and web framework that's written to go fast. Build fast. Run fast.",
    },
    {
        "name": "poetry",
        "short_desc": "Python dependency management and packaging made easy.",
    },
]
```

Introducing pyaday - cont.

```python
@packages_bp.route("", methods=["POST"])
def create_package():
    req_data = request.json
    python_packages.append(
        {"name": req_data.get("name"), "short_desc": req_data.get("short_desc")}
    )
    return Response(
        response=None,
        status=201,
        headers={"Location": f"/packages/{req_data.get('name')}"},
    )
```

Introducing pyaday - cont.

```python
@packages_bp.route("/<package_name>", methods=["GET"])
def read_package(package_name):
    for python_package in python_packages:
        if python_package["name"] == package_name:
            return jsonify(python_package)
    return Response(
        response=json.dumps({"title": "Could not find a package"}),
        status=404,
        content_type="application/problem+json",
    )
```

Introducing pyaday - cont.

```python
@packages_bp.route("/<package_name>", methods=["PUT"])
def update_package(package_name):
    req_data = request.json
    for python_package in python_packages:
        if python_package["name"] == package_name:
            python_package["short_desc"] = req_data.get("short_desc")
            return Response(response=None, status=204)
    return Response(
        response={"title": "Could not find a package"},
        status=404,
        content_type="application/problem+json",
    )
```

Introducing pyaday - cont.

```python
@packages_bp.route("/<package_name>", methods=["DELETE"])
def delete_package(package_name):
    for idx, python_package in enumerate(python_packages):
        if python_package["name"] == package_name:
            del python_packages[idx]
            return Response(response=None, status=204)
    return Response(
        response=json.dumps({"title": "Could not find a package"}),
        status=404,
        content_type="application/problem+json",
    )
```

Introducing pyaday - cont.

```python
import random

from flask import Blueprint, jsonify

from pyaday.api.packages import python_packages

rand_bp = Blueprint(name="rand_bp",
                    import_name=__name__)


@rand_bp.route("", methods=["GET"])
def read_random_package():
    return jsonify(random.choice(python_packages))
```

Why convert?

Better bang for the buck for a large scale expensive cloud deployment or a limited in resources on premises deployment.

=> Meaning - it will save you $$$

In addition, we'll try to show the migration is not difficult and the flask knowledge is not wasted.

Let the conversion begin!

Prerequisite:
A project that can benefit from conversion written in python3.6-3.8 (I used 3.8.3).

```
$ poetry init #not mandatory, my preference
$ poetry add sanic
```

* Flask v1.1.2 & Sanic v20.3.0 were used, so syntax may vary on different versions.

App constructor

```python
from flask import Flask

app = Flask(__name__)
```

```python
from sanic import Sanic

app = Sanic(__name__)
```

Route

```python
from flask import request

@app.route("/")
def hello():
```

```python
@app.route("/")
async def hello(request):
```

- On Flask request object is globally imported ; on Sanic it is the first arg.
- On Sanic, the route is a coroutine (a function that uses the async keyword).

JSON response

Happy path:

```
return {"hello": "world"}
```

```
from sanic.response import json

return json({"hello": "world"})
```

```
from flask import jsonify

return jsonify({"hello": "world"})
```

JSON response - cont.

Error handling (according to RFC7807):

```
return Response(
    response=json.dumps({"title": "Could not find a package"}),
    status=404,
    content_type="application/problem+json",
)
```

* There are other Flask options:

```
response = jsonify(title="...")

response.status =...

return response
```

```
return response.json(
    body={"title": "Could not find a package"},
    status=404,
    content_type="application/problem+json",
)
```

Auto reload for development

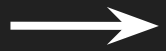
```python
if __name__ == "__main__":
    app.run(debug=True)
```

Same for Flask & Sanic*

* There are other options as well:
  - Flask, from terminal:
    FLASK_ENV=development FLASK_APP=main_flask.py flask run
  - Sanic: app.run(auto_reload=true)

Blueprint

```python
import random

from flask import Blueprint, jsonify

from pyaday.api.packages import python_packages

rand_bp = Blueprint(name="rand_bp",
                    import_name=__name__)


@rand_bp.route("", methods=["GET"])
def read_random_package():
    return jsonify(random.choice(python_packages))
```



```python
import random

from sanic import Blueprint, response

from pyadayasync.api.packages import python_packages

rand_bp = Blueprint(name="rand_bp")


@rand_bp.route("", methods=["GET"])
async def read_random_package(_):
    return response.json(random.choice(python_packages))
```

* Used for sub-routing => contains all the exposed methods of a certain route.
* Sanic does not require import_name.

Blueprint - cont.

```
app.register_blueprint(blueprint=rand_bp, url_prefix="/rand")
```

```
app.blueprint(blueprint=rand_bp, url_prefix="/rand")
```

* register_blueprint can work as well in Sanic, but it is marked as deprecated.

Post conversion diff

```
from flask import Blueprint, Response, request

packages_bp = Blueprint(name="packages_bp", import_name=__name__)
```

```
from sanic import Blueprint, response

packages_bp = Blueprint(name="packages_bp")
```

Post conversion diff - cont.

```
@packages_bp.route("", methods=["POST"])
def create_package():
    req_data = request.json
    python_packages.append(
        {"name": req_data.get("name"), "short_desc": req_data.get("short_desc")}
    )
    return Response(
        response=None,
        status=201,
        headers={"Location": f"/packages/{req_data.get('name')}"}
    )
```

```
@packages_bp.route("", methods=["POST"])
async def create_package(request):
    req_data = request.json
    python_packages.append(
        {"name": req_data.get("name"), "short_desc": req_data.get("short_desc")}
    )
    return response.empty(
        status=201,
        headers={"Location": f"/packages/{req_data.get('name')}"}
    )
```

Post conversion diff - cont.

```
@packages_bp.route("/<package_name>", methods=["GET"])
def read_package(package_name):
    for python_package in python_packages:
        if python_package["name"] == package_name:
            return jsonify(python_package)
    return Response(
        response=json.dumps({"title": "Could not find a package"}),
        status=404,
        content_type="application/problem+json",
    )
```

```
@packages_bp.route("/<package_name>", methods=["GET"])
async def read_package(_, package_name):
    for python_package in python_packages:
        if python_package["name"] == package_name:
            return response.json(body=python_package)
    return response.json(
        body={"title": "Could not find a package"},
        status=404,
        content_type="application/problem+json",
    )
```

Post conversion diff - cont.

```
@packages_bp.route("/<package_name>", methods=["PUT"])
def update_package(package_name):
    req_data = request.json
    for python_package in python_packages:
        if python_package["name"] == package_name:
            python_package["short_desc"] = req_data.get("short_desc")
            return Response(response=None, status=204)
    return Response(
        response=json.dumps({"title": "Could not find a package"}),
        status=404,
        content_type="application/problem+json",
    )
```

```
@packages_bp.route("/<package_name>", methods=["PUT"])
async def update_package(request, package_name):
    req_data = request.json
    for python_package in python_packages:
        if python_package["name"] == package_name:
            python_package["short_desc"] = req_data.get("short_desc")
            return response.empty(status=204)
    return response.json(
        body={"title": "Could not find a package"},
        status=404,
        content_type="application/problem+json",
    )
```

Post conversion diff - cont.

```
@packages_bp.route("/<package_name>", methods=["DELETE"])
def delete_package(package_name):
    for idx, python_package in enumerate(python_packages):
        if python_package["name"] == package_name:
            del python_packages[idx]
            return Response(response=None, status=204)
    return Response(
        response=json.dumps({"title": "Could not find a package"}),
        status=404,
        content_type="application/problem+json",
    )
```

```
@packages_bp.route("/<package_name>", methods=["DELETE"])
async def delete_package(_, package_name):
    for idx, python_package in enumerate(python_packages):
        if python_package["name"] == package_name:
            del python_packages[idx]
            return response.empty(status=204)
    return response.json(
        body={"title": "Could not find a package"},
        status=404,
        content_type="application/problem+json",
    )
```

Testing

```python
with app.test_client() as test_client:
    response = test_client.get("rand")
    assert response.status_code == 200
    assert response.headers["content-type"] == "application/json"
```

```python
test_client = app.test_client
_, response = test_client.get("rand")
assert response.status == 200
assert response.headers["content-type"] == "application/json"
```

* Test client: Flask through a method and a context manager ; Sanic - through an attribute.
* Calling routes: Flask - returns `response` ; Sanic - returns `request` & `response`.
* Check response status: Flask - `status_code` ; Sanic - `status`.

Testing Diff

test_packages.py (/Users/db/PycharmProjects/flasync-await/pyaday/api)

```python
from main_flask import app


def test_packages_crud():
    with app.test_client() as test_client:
        response = test_client.post(
            "/packages",
            json={
                "name": "dynaconf",
                "short_desc": "The dynamic configurator for your Python Project
            },
        )
        assert response.status_code == 201
        response = test_client.get("/packages/dynaconf")
        assert response.status_code == 200
        assert (
            response.json["short_desc"]
            == "The dynamic configurator for your Python Projects"
        )
        response = test_client.put(
            "/packages/dynaconf",
            json={"short_desc": "The dynamic configurator for your Python Proje
        )
        assert response.status_code == 204
        response = test_client.get("/packages/dynaconf")
        assert response.status_code == 200
        assert (
            response.json["short_desc"]
            == "The dynamic configurator for your Python Project"
        )
        response = test_client.delete("/packages/dynaconf")
        assert response.status_code == 204
        response = test_client.get("/packages/dynaconf")
        assert response.status_code == 404
```

test_packages.py (/Users/db/PycharmProjects/flasync-await/pyadayasync/api)

```python
from main_sanic import app


def test_packages_crud():
    test_client = app.test_client
    _, response = test_client.post(
        "/packages",
        json={
            "name": "dynaconf",
            "short_desc": "The dynamic configurator for your Python Projects",
        },
    )
    assert response.status == 201
    _, response = test_client.get("/packages/dynaconf")
    assert response.status == 200
    assert (
        response.json["short_desc"]
        == "The dynamic configurator for your Python Projects"
    )
    _, response = test_client.put(
        "/packages/dynaconf",
        json={"short_desc": "The dynamic configurator for your Python Project"}
    )
    assert response.status == 204
    _, response = test_client.get("/packages/dynaconf")
    assert response.status == 200
    assert (
        response.json["short_desc"]
        == "The dynamic configurator for your Python Project"
    )
    _, response = test_client.delete("/packages/dynaconf")
    assert response.status == 204
    _, response = test_client.get("/packages/dynaconf")
    assert response.status == 404
```

Testing Diff - cont.

Sanic tests can also be async (pytest-sanic package is a requirement for this):

```
test_client = app.test_client
_, response = test_client.get("rand")
assert response.status == 200
assert response.headers["content-type"] == "application/json"
```

```
@pytest.fixture
def test_client(loop, sanic_client):
    return loop.run_until_complete(sanic_client(app))


async def test_rand_async(test_client):
    response = await test_client.get("/rand")
    assert response.status == 200
    assert response.headers["content-type"] == "application/json"
```

Testing Diff - cont.

* There is only one return value - response, similar to Flask.

* Need to “await” every server call as opposed to Flask.

Deployment

```
> gunicorn -w 4 main_flask:app -b 127.0.0.1:5000
[2020-07-09 15:11:07 +0300] [98065] [INFO] Starting gunicorn 20.0.4
[2020-07-09 15:11:07 +0300] [98065] [INFO] Listening at: http://127.0.0.1:5000 (98065)
[2020-07-09 15:11:07 +0300] [98065] [INFO] Using worker: sync
```

```
> gunicorn -w 4 -k uvicorn.workers.UvicornWorker main_sanic:app -b 127.0.0.1:8000
[2020-07-09 15:11:53 +0300] [98087] [INFO] Starting gunicorn 20.0.4
[2020-07-09 15:11:53 +0300] [98087] [INFO] Listening at: http://127.0.0.1:8000 (98087)
[2020-07-09 15:11:53 +0300] [98087] [INFO] Using worker: uvicorn.workers.UvicornWorker
```

Deployment - cont.

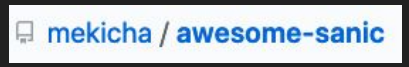
```
Server Software:        gunicorn/20.0.4
Server Hostname:        127.0.0.1
Server Port:            5000

Document Path:          /rand
Document Length:        Variable

Concurrency Level:      100
Time taken for tests:   9.816 seconds
Complete requests:      10000
Failed requests:        0
Total transferred:      2491241 bytes
HTML transferred:       967854 bytes
Requests per second:    1018.78 [#/sec] (mean)
Time per request:       98.156 [ms] (mean)
Time per request:       0.982 [ms] (mean, across all
Transfer rate:          247.85 [Kbytes/sec] received
```

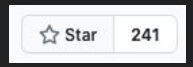
```
Server Software:        uvicorn
Server Hostname:        127.0.0.1
Server Port:            8000

Document Path:          /rand
Document Length:        Variable

Concurrency Level:      100
Time taken for tests:   1.552 seconds
Complete requests:      10000
Failed requests:        0
Total transferred:      2210797 bytes
HTML transferred:       957432 bytes
Requests per second:    6445.09 [#/sec] (mean)
Time per request:       15.516 [ms] (mean)
Time per request:       0.155 [ms] (mean, across all
Transfer rate:          1391.48 [Kbytes/sec] received
```

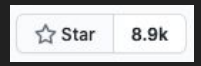
5-6x for GET /rand route

Not always a fairytale

- A cognitive bourdain: for a performant (and an effective) async code the event loop must never be blocked:
  - IO should be await(ed)
  - CPU should run elsewhere (loop.run_in_executor(...))
- Sanic's ecosystem is not as rich as Flask's ecosystem. It is noticeable on Github, on the number of available tutorials and on 3rd party integrations (like okta, auth0 or swagger-codegen).







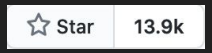
Not always a fairytale - cont.

- Need to use 3rd party libraries that do not block IO:
  - psycopg2 -> asyncpg / aiopg*
  - requests -> httpx / aiohttp
  - redis -> aioredis / asyncio-redis

  ...

* That's why a DB wasn't used for the converted application - to make the comparison simple.

The async web framework landscape

- Sanic was chosen for this talk because:
  - It is popular on Github 

  - The API it exposes is very similar to the API exposed by Flask. When the API is not the same, it seems like a reasonable evolution that's made possible because there isn't a lot of backward compatibility needed.
  - It is backed by a community run organization.
  - 90s flashback :) 
- Quart is also a Flask like async web framework.
- Fastapi is a hybrid web framework (sync and an async) with dependency injection as a guiding principle.

Summary

- When a Flask app that mostly performs IO becomes resource hungry, it is worthwhile to convert it to Sanic in reasonable effort.
- After converting, the code must be IO & CPU aware in order to not block the event loop.

@DavidBordeynik