In Flask we Trust

Igor Davydenko
UA PyCon 2012

Flask is not
a new **Django**

Flask is a micro-framework

- It’s only Werkzeug (WSGI toolkit), Jinja2 (template engine) and bunch of good things on top
- No unnecessary batteries included by default
- The idea of Flask is to build a good foundation for all applications. Everything else is up to you or extensions
- So no more projects. All you need is Flask application

No **ORM**, no **forms**, no **contrib**

- Not every application needs a SQL database
- People have different preferences and requirements
- Flask could not and don't want to apply those differences
- Flask itself just bridges to Werkzeug to implement a proper WSGI application and to Jinja2 to handle templating
- And yeah, most of web applications really need a template engine in some sort

But actually we prepared well

- **Blueprints** as glue for views (*but blueprint is not a reusable app*)
- **Extensions** as real batteries for our application
- And yeah, we have **ORM** (Flask-SQLAlchemy, Flask-Peewee, Flask-MongoEngine and many others)
- We have **forms** (Flask-WTF)
- We have **anything** we need (Flask-Script, Flask-Testing, Flask-Dropbox, Flask-FlatPages, Frozen-Flask, etc)

Application structure

From documentation

```
$ tree -L 1
.
├── app.py
└── requirements.txt
```

From real world

```
$ tree -L 2
.
└── appname/
    ├── blueprintname/
    ├── onemoreblueprint/
    ├── static/
    ├── templates/
    ├── tests/
    ├── __init__.py
    ├── app.py
    ├── manage.py
    ├── models.py
    ├── settings.py
    ├── views.py
    └── utils.py
└── requirements.txt
```

Application source

From documentation

```
$ cat app.py
from flask import Flask

app = Flask(__name__)

@app.route('/')
def hello():
    return 'Hello, world!'

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

From real world

```
$ cat appname/app.py
from flask import Flask
# Import extensions and settings

app = Flask(__name__)
app.config.from_object(settings)

# Setup context processors, template
# filters, before/after requests handlers

# Initialize extensions

# Add lazy views, blueprints, error
# handlers to app

# Import and setup anything which needs
# initialized app instance
```

How to run?

From documentation

```
(env)$ python app.py
 * Running on http://127.0.0.1:5000/
```

From real world

```
(env)$ python manage.py runserver -p 4321
...

(env)$ gunicorn appname.app:app -b 0.0.0.0:5000 -w 4
...

(env)$ cat /etc/uwsgi/sites-available/appname.ini
chdir = /path/to/appname
venv = %(chdir)/env/
pythonpath = /path/to/appname
module = appname.app:app
touch-reload = %(chdir)/appname/app.py
(env)$ sudo service uwsgi full-reload
...
```

From **request**
to **response**

Routing

- Hail to the Werkzeug routing!

```
app = Flask(__name__)
app.add_url_rule('/', index_view, endpoint='index')
app.add_url_rule('/page', page_view, defaults={'pk': 1},
                 endpoint='default_page')
app.add_url_rule('/page/<int:pk>', page_view, endpoint='page')

@app.route('/secret', methods=('GET', 'POST'))
@app.route('/secret/<username>')
def secret(username=None):
    ...
```

- All application URL rules storing in `app.url_map` instance. No more `manage.py show_urls`, just `print(app.url_map)`

URL routes in code

- Just `url_for` it!

```
>>> from flask import url_for
>>> url_for(‘index’)
‘/’
>>> url_for(‘default_page’)
‘/page’
>>> url_for(‘page’, pk=1)
‘/page/1’
>>> url_for(‘secret’, _external=True)
‘http://127.0.0.1:5000/secret’
>>> url_for(‘secret’, username=‘user’, foo=‘bar’)
‘/secret/user?foo=bar’
```

- And in templates too,

```
{{ url_for(“index”) }}

{{ url_for(“secret”, _external=True) }}
```

Request

- View doesn't need a `request` arg!
- There is one request object per request which is read only
- The request object is available through local context
- Request is thread-safe by design
- When you need it, import it!

```
from flask import request

def page_view(pk):
    return 'Page #{0:d} @ {1!r} host'.format(pk, request.host)
```

Response

- There is no `flask.response`
- Can be implicitly created
- Can be replaced by other response objects

Implicitly created response

- Could be a text

```
def index_view():
    return ‘Hello, world!’
```

Implicitly created response

- A tuple

```
from app import app

@app.errorhandler(404)
@app.errorhandler(500)
def error(e):
    code = getattr(e, ‘code’, 500)
    return ‘Error {0:d}’.format(code), code
```

Implicitly created response

- Or rendered template

```
from flask import render_template
from models import Page

def page_view(pk):
    page = Page.query.filter_by(id=pk).first_or_404()
    return render_template(‘page.html’, page=page)
```

Explicitly created response

- Text or template

```
from flask import make_response, render_template

def index_view():
    response = make_response('Hello, world!')
    return response

def page_view(pk):
    output = render_template('page.html', page=pk)
    response = make_response(output)
    return response
```

Explicitly created response

- Tuple with custom headers

```
from flask import make_response
from app import app

@app.errorhandler(404)
def error(e):
    response = make_response(‘Page not found!’, e.code)
    response.headers[‘Content-Type’] = ‘text/plain’
    return response
```

Explicitly created response

- Rendered template with custom headers,

```
from flask import make_response, render_template
from app import app

@app.errorhandler(404)
def error(e):
    output = render_template(‘error.html’, error=e)
    return make_response(
        output, e.code, {‘Content-Language’: ‘ru’}
    )
```

The application and the request **contexts**

All starts with **states**

- Application setup state
- Runtime state
  - Application runtime state
  - Request runtime state

What is about?

```
In [1]: from flask import Flask, current_app, request

In [2]: app = Flask('appname')

In [3]: app
Out[3]: <flask.app.Flask at 0x1073139d0>

In [4]: current_app
Out[4]: <LocalProxy unbound>

In [5]: with app.app_context():
    print(repr(current_app))
   ...:
<flask.app.Flask object at 0x1073139d0>

In [6]: request
Out[6]: <LocalProxy unbound>

In [7]: with app.test_request_context():
   ....:     print(repr(request))
   ....:
<Request 'http://localhost/' [GET]>
```

Flask core

```
class Flask(_PackageBoundObject):
    ...
    def wsgi_app(self, environ, start_response):
        with self.request_context(environ):
            try:
                response = self.full_dispatch_request()
            except Exception, e:
                response = self.make_response(self.handle_exception(e))
            return response(environ, start_response)
```

Hello to **contexts**

- Contexts are stacks
- So you can push to multiple contexts objects
- Request stack and application stack are independent

What depends on contexts?

- **Application** context
  - `flask._app_ctx_stack`
    - `flask.current_app`
- **Request** context
  - `flask._request_ctx_stack`
    - `flask.g`
    - `flask.request`
    - `flask.session`

More?

- Stack objects are **shared**
- There are **context managers** to use
  - `app.app_context`
  - `app.test_request_context`
- Working with shell

```
>>> ctx = app.test_request_context()
>>> ctx.push()
>>> ...
>>> ctx.pop()
```

Applications vs. Blueprints

Blueprint is not an application

- Blueprint is glue for views
- Application is glue for blueprints and views

Blueprint uses data from app

- Blueprint hasn't `app` attribute
- Blueprint doesn't know about application state
- But in most cases blueprint needs to know about application

Trivial example

```
$ cat appname/app.py
from flask import Flask
from .blueprintname import blueprint

app = Flask(__name__)
app.register_blueprint(blueprint, url_prefix='/blueprint')

@app.route('/')
def hello():
    return 'Hello from app!'

$ cat appname/blueprintname/__init__.py
from .blueprint import blueprint

$ cat appname/blueprintname/blueprint.py
from flask import Blueprint

blueprint = Blueprint('blueprintname', 'importname')

@blueprint.route('/')
def hello():
    return 'Hello from blueprint!'
```

Real example

```
$ cat appname/app.py
...
app = Flask(__name__)
db = SQLAlchemy(app)
...
from .blueprintname import blueprint
app.register_blueprint(blueprint, url_prefix='/blueprint')

$ cat appname/models.py
from app import db

class Model(db.Model):
    ...

$ cat appname/blueprintname/blueprint.py
from flask import Blueprint
from appname.models import Model

blueprint = Blueprint('blueprintname', 'importname')

@blueprint.route('/')
def hello():
    # Work with model
    return 'something...'
```

Sharing data with blueprint

```
$ cat appname/app.py
from flask import Flask
from blueprintname import blueprint

class Appname(Flask):
    def register_bluepint(self, blueprint, **kwargs):
        super(Appname, self).register_blueprint(blueprint, **kwargs)
        blueprint.extensions = self.extensions

app = Appname(__name__)
app.register_blueprint(blueprint)

$ cat blueprintname/deferred.py
from .blueprint import blueprint

db = blueprint.extensions[‘sqlalchemy’].db
```

More canonical way

```
$ cat appname/app.py
from flask import Flask
from blueprintname import blueprint

app = Flask(__name__)
app.register_blueprint(blueprint)

$ cat blueprintname/deferred.py
from appname.app import db
```

Factories

- Application can created by factory, e.g. for using different settings
- Blueprint can created by factory for same reasons

Application factory

```
$ cat appname/app.py
from flask import Flask

def create_app(name, settings):
    app = Flask(name)
    app.config.from_pyfile(settings)
    register_blueprints(app.config[‘BLUEPRINTS’])

backend_app = create_app(‘backend’, ‘backend.ini’)
frontend_app = create_app(‘frontend’, ‘frontend.ini’)
```

Blueprint factory

```
$ cat appname/backend_app.py
from blueprintname import create_blueprint
...
app.register_blueprint(create_blueprint(app), url_prefix='/blueprint')

$ cat appname/frontend_app.py
from blueprintname import create_blueprint
...
app.register_blueprint(create_blueprint(app), url_prefix='/blueprint')

$ cat blueprintname/blueprint.py
from flask import Blueprint
from flask.ext.lazyviews import LazyViews

def create_blueprint(app):
    blueprint = Blueprint(__name__)
    views = LazyViews(blueprint)

    if app.name == 'backend':
        blueprint.add_app_template_filter(backend_filter)

    views.add('/url', 'view')
    return blueprint
```

Customizing

- Just inherit `flask.Flask` or `flask.Blueprint`

```
class Appname(Flask):
    def send_static_file(self, filename):
        ...
```

- Apply WSGI middleware to `Flask.wsgi_app` method

```
from werkzeug.wsgi import DispatcherMiddleware

main_app.wsgi_app = DispatcherMiddleware(main_app.wsgi_app, {
    '/backend': backend_app.wsgi_app,
})
```

Extensions

That's what Flask about

- You need some code more than in one Flask app?
- Place it to `flask_extname` module or package
- Implement `Extname` class and provide `init_app` method
- Don't forget to add your extension to `app.extensions` dict
- Volia!

Example. Flask-And-Redis

- Module `flask_redis`, class `Redis`

```
from redis import Redis

class Redis(object):
    def __init__(self, app=None):
        if app:
            self.init_app(app)
        self.app = app

    def init_app(self, app):
        config = self._read_config(app)

        self.connection = redis = Redis(**config)
        app.extensions['redis'] = redis

        self._include_redis_methods(redis)
```

Usage. Singleton

- One Flask application, one Redis connection

```
from flask import Flask
from flask.ext.redis import Redis

app = Flask(‘appname’)
app.config[‘REDIS_URL’] = ‘redis://localhost:6379/0’
redis = Redis(app)

@app.route(‘/counter’)
def counter():
    number = redis.incr(‘counter_key’)
    return ‘This page viewed {:d} time(s)’.format(number)
```

Usage. Advanced

- Initializing without app object (*multiple apps to one extension*)

```
$ cat extensions.py
from flask.ext.redis import Redis

redis = Redis()

$ cat backend_app.py
from flask import Flask
from extensions import redis

app = Flask('backend')
app.config['REDIS_URL'] = 'redis://localhost:6379/0'
redis.init_app(app)

@app.route('/counter')
def counter():
    number = redis.incr('counter_key')
    return 'This page viewed {:d} time(s)'.format(number)
```

So, one more time

- Provide `init_app` method to support multiple applications
- Don't forget about `app.extensions` dict
- Do not assign `self.app = app` in `init_app` method
- Extension should have not-null `self.app` only for singleton pattern

List of **extensions** you **should to** know and **use**

Database, forms, admin

- **SQL ORM**: Flask-SQLAlchemy, Flask-Peewee
- **NoSQL**: Flask-CouchDB, Flask-PyMongo, Flask-And-Redis
- **NoSQL ORM**: Flask-MongoEngine, Flask-MiniMongo
- **Forms**: Flask-WTF
- **Admin**: Flask-Admin, Flask-Dashed, Flask-Peewee

Authentication, REST

- **Base**: Flask-Auth, Flask-BasicAuth, Flask-Login
- **Advanced**: Flask-Security
- **Social auth**: Flask-GoogleAuth, Flask-OAuth, Flask-OpenID, Flask-Social
- **REST**: Flask-REST, Flask-Restless, Flask-Snooze

Management

- **Internationalization**: Flask-Babel
- **Management commands**: Flask-Actions, Flask-Script
- **Assets**: Flask-Assets, Flask-Collect
- **Testing**: flask-fillin, Flask-Testing
- **Debug toolbar**: Flask-DebugToolbar

Other

- **Cache**: Flask-Cache
- **Celery**: Flask-Celery
- **Lazy views**: Flask-LazyViews
- **Dropbox API**: Flask-Dropbox
- **Flat pages**: Flask-FlatPages, Frozen-Flask
- **Mail**: Flask-Mail
- **Uploads**: Flask-Uploads

Debugging, testing and **deployment**

Werkzeug debugger

AssertionError // Werkzeug

127.0.0.1:4391

AssertionError

AssertionError

Traceback (most recent call last)

File "/Users/playpauseandstop/Projects/inflaskwetrust/env/lib/python2.7/site-packages/flask/app.py", line *1701*, in `__call__`

```
return self.wsgi_app(environ, start_response)
```

File "/Users/playpauseandstop/Projects/inflaskwetrust/env/lib/python2.7/site-packages/flask/app.py", line *1689*, in `wsgi_app`

```
response = self.make_response(self.handle_exception(e))
```

File "/Users/playpauseandstop/Projects/inflaskwetrust/env/lib/python2.7/site-packages/flask/app.py", line *1687*, in `wsgi_app`

```
response = self.full_dispatch_request()
```

File "/Users/playpauseandstop/Projects/inflaskwetrust/env/lib/python2.7/site-packages/flask/app.py", line *1360*, in `full_dispatch_request`

```
rv = self.handle_user_exception(e)
```

File "/Users/playpauseandstop/Projects/inflaskwetrust/env/lib/python2.7/site-packages/flask/app.py", line *1358*, in `full_dispatch_request`

```
rv = self.dispatch_request()
```

File "/Users/playpauseandstop/Projects/inflaskwetrust/env/lib/python2.7/site-packages/flask/app.py", line *1344*, in `dispatch_request`

```
return self.view_functions[rule.endpoint](**req.view_args)
```

File "/Users/playpauseandstop/Projects/inflaskwetrust/app.py", line *12*, in `hello`

```
assert False
```

AssertionError

The debugger caught an exception in your WSGI application. You can now look at the traceback which led to the error.

To switch between the interactive traceback and the plaintext one, you can click on the "Traceback" headline. From the text traceback you can also create a paste of it. For code execution mouse-over the frame you want to debug and click on the console icon on the right side.

You can execute arbitrary Python code in the stack frames and there are some extra helpers available for introspection:

- `dump()` shows all variables in the frame
- `dump(obj)` dumps all that's known about the object

Brought to you by DON'T PANIC, your friendly Werkzeug powered traceback interpreter

pdb, ipdb

- Just import pdb (ipdb) in code and set trace

```
def view():
    ...
    import pdb
    pdb.set_trace()
    ...
```

- That's all!
- Works with development server

```
(env)$ python app.py
(env)$ python manage.py runserver
```

- Or gunicorn

```
(env)$ gunicorn app:app -b 0.0.0.0:5000 -t 9000 --debug
```

Debug toolbar

Hello, world!

127.0.0.1:4391

View: 0.11ms

Calls	Total Time (ms)	Per Call (ms)	Cumulative Time (ms)	Per Call (ms)	Function
1	0.021	0.0210	0.08	0.0800	./env/lib/python2.7/site-packages/werkzeug/wrappers.py:620(__init__)
1	0.011	0.0110	0.112	0.1120	./env/lib/python2.7/site-packages/flask/helpers.py:201(make_response)
2	0.01	0.0050	0.016	0.0080	./env/lib/python2.7/site-packages/werkzeug/datastructures.py:1080(set)
1	0.008	0.0080	0.009	0.0090	./env/lib/python2.7/site-packages/werkzeug/wrappers.py:733(_set_status_code)
1	0.007	0.0070	0.02	0.0200	./env/lib/python2.7/site-packages/werkzeug/wrappers.py:764(_set_data)
1	0.007	0.0070	0.089	0.0890	./env/lib/python2.7/site-packages/flask/app.py:1405(make_response)
2	0.006	0.0030	0.023	0.0115	./env/lib/python2.7/site-packages/werkzeug/datastructures.py:1126(__setitem__)
1	0.005	0.0050	0.006	0.0060	./env/lib/python2.7/site-packages/werkzeug/datastructures.py:854(__getitem__)
1	0.005	0.0050	0.007	0.0070	./env/lib/python2.7/site-packages/werkzeug/utils.py:217(get_content_type)
1	0.004	0.0040	0.01	0.0100	./env/lib/python2.7/site-packages/werkzeug/datastructures.py:1025(__contains__)
11	0.004	0.0004	0.004	0.0004	{isinstance}
2	0.003	0.0015	0.003	0.0015	./env/lib/python2.7/site-packages/werkzeug/datastructures.py:1063(_validate_value)
1	0.003	0.0030	0.005	0.0050	./env/lib/python2.7/site-packages/werkzeug/local.py:156(top)
1	0.003	0.0030	0.01	0.0100	./env/lib/python2.7/site-packages/werkzeug/local.py:289(_get_current_object)
3	0.002	0.0007	0.002	0.0007	{method 'lower' of 'str' objects}
1	0.002	0.0020	0.002	0.0020	./env/lib/python2.7/site-packages/werkzeug/local.py:66(__getattr__)
1	0.002	0.0020	0.002	0.0020	{method 'startswith' of 'str' objects}
1	0.001	0.0010	0.001	0.0010	{getattr}
1	0.001	0.0010	0.113	0.1130	./app.py:17(hello)
1	0.001	0.0010	0.001	0.0010	{method 'upper' of 'str' objects}
1	0.001	0.0010	0.001	0.0010	{iter}
2	0.001	0.0005	0.001	0.0005	{len}
1	0.001	0.0010	0.012	0.0120	./env/lib/python2.7/site-packages/werkzeug/local.py:333(__getattr__)
1	0.001	0.0010	0.001	0.0010	{hasattr}
2	0.001	0.0005	0.001	0.0005	{method 'append' of 'list' objects}
1	0.001	0.0010	0.001	0.0010	./env/lib/python2.7/site-packages/werkzeug/datastructures.py:828(__init__)
1	0.001	0.0010	0.006	0.0060	./env/lib/python2.7/site-packages/flask/globals.py:23(_find_app)
1	0.0	0.0000	0.0	0.0000	{thread.get_ident}
1	0.0	0.0000	0.0	0.0000	{method 'disable' of '_lsprof.Profiler' objects}

Hide »

Versions
Flask 0.9

Time
CPU: 0.43ms (0.43ms)

HTTP Headers

Request Vars

Templates
0 rendered

SQLAlchemy
Unavailable

Logging
0 messages

Profiler
View: 0.11ms

Flask-Testing

- Inherit test case class from `flask.ext.testing.TestCase`
- Implement `create_app` method

```
from flask.ext.testing import TestCase
from appname.app import app

class TestSomething(TestCase):
    def create_app(self):
        app.testing = True
        return app
```

- Run tests with unittest2

```
(env)$ python -m unittest discover -fv -s appname/
```

- Or with nosetests

```
(env)$ nosetests -vx -w appname/
```

WebTest

- Setup `app` and wrap it with `TestApp` class
- Don't forget about contexts

```
from unittest import TestCase
from webtest import TestApp
from appname.app import app

class TestSomething(TestCase):
    def setUp(self):
        app.testing = True
        self.client = TestApp(app)
        self._ctx = app.test_request_context()
        self._ctx.push()

    def tearDown(self):
        if self._ctx is not None:
            self._ctx.pop()
```

Application factories & tests

- Yeah, it's good idea to use application factories when you have at least tests
- So `appname.create_app` better than `appname.app`, trust me :)

Deploy to Heroku

- Heroku perfectly fits staging needs
- One dyno, shared database, Redis, Mongo, email support, Sentry for free
- Viva la **gunicorn**!

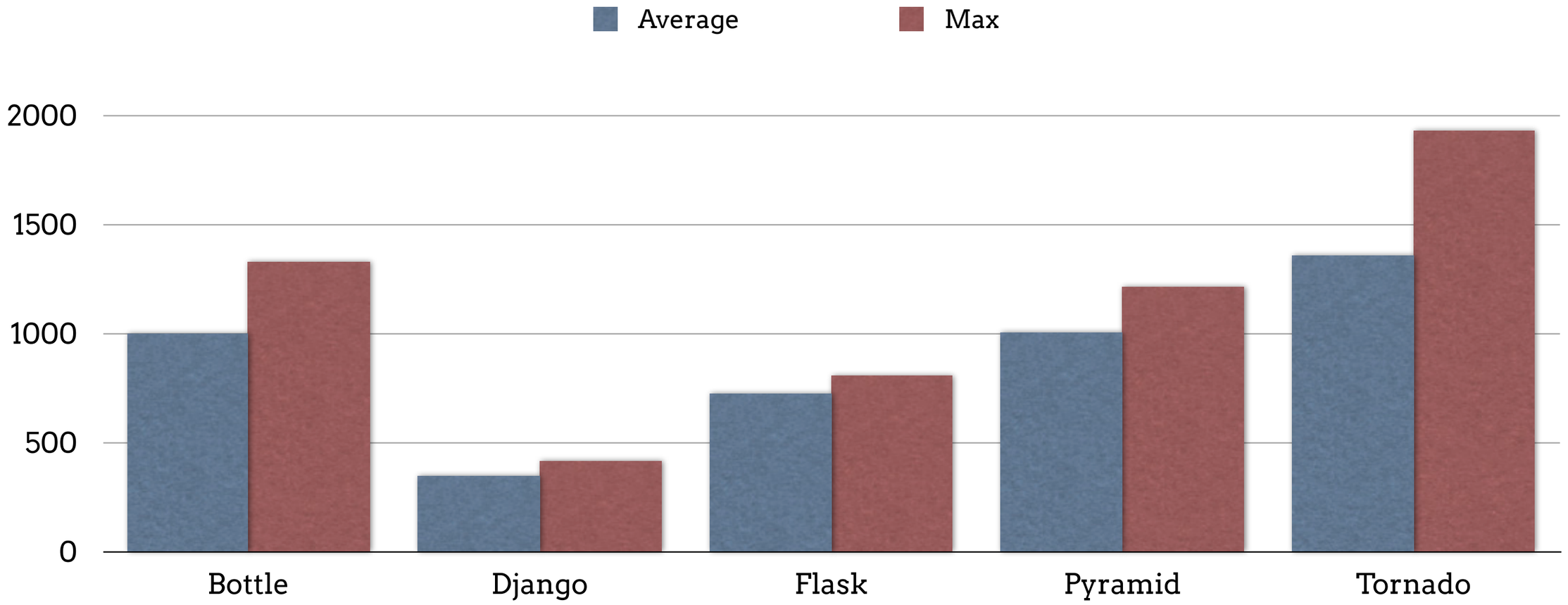
```
$ cat Procfile
web: gunicorn appname.app:app -b 0.0.0.0:$PORT -w 4
```

Deploy anywhere else

- *nginx* + **gunicorn**
- *nginx* + **uwsgi**
- And don't forget that you can wrap your Flask app with **Tornado**, **gevent**, **eventlet**, **greenlet** or any other WSGI container

Funny numbers

Without concurrency



Requests per second

```
$ ab -c 1 -n 1000 URL
```

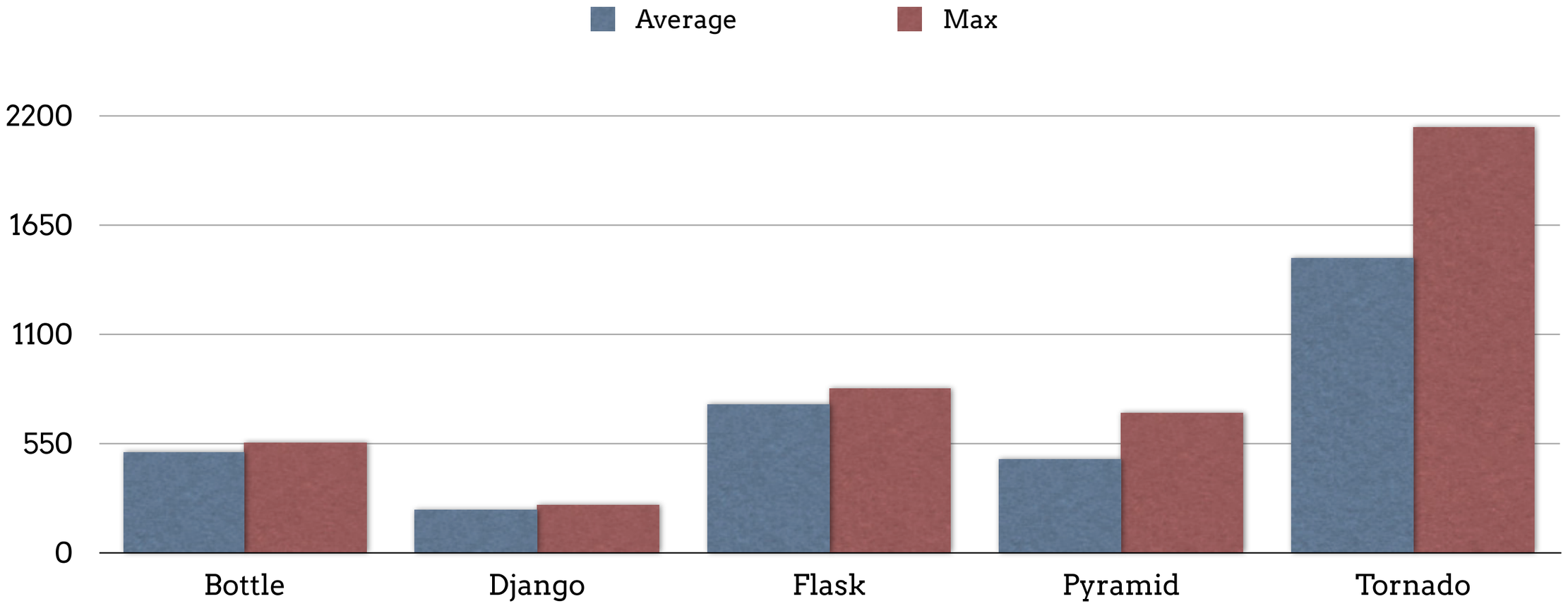
URL	Bottle	Django	Flask	Pyramid	Tornado
/ 13 bytes	1327.99	416.83	806.86	1214.67	1930.96
/environ ~2900 bytes	1018.14	376.16	696.96	986.82	1430.54
/template 191 bytes	654.71	252.96	670.24	814.37	711.49

Time per request

```
$ ab -c 1 -n 1000 URL
```

URL	Bottle	Django	Flask	Pyramid	Tornado
/ 13 bytes	0.748ms	2.360ms	1.248ms	0.826ms	0.521ms
/environ ~2900 bytes	0.963ms	2.672ms	1.425ms	1.007ms	0.715ms
/template 191 bytes	1.523ms	4.177ms	1.475ms	1.189ms	1.399ms

With concurrency



Requests per second

```
$ ab -c 100 -n 1000 URL
```

URL	Bottle	Django	Flask	Pyramid	Tornado
/ 13 bytes	553.02	228.91	826.34	703.82	2143.29
/environ ~2900 bytes	522.16	240.51	723.90	415.20	1557.62
/template 191 bytes	444.37	177.14	693.42	297.47	746.87

Additional notes

- Only Flask and Tornado can guarantee 100% responses on 100 concurrency requests
- Bottle, Django and Pyramid WSGI servers will have 2-10% errors or will shutdown after 1000 requests
- Gunicorn will not help for sure :(

Questions?

I am **Igor Davydenko**
http://igordavydenko.com
http://github.com/playpauseandstop