Python microservices on PaaS done right

Michał Bultrowicz
About me

● Work at Intel Technology Poland.
● I do backend services.
● Sadly, mainly in Java.
● I did some C++ security...
● ...and multiplatform distributed automated testing soft.
● I really, really like Python.
● It’s my first time presenting.
Thanks for the help

Izabela Irzyńska
1. Microservices introduction.
2. PaaS introduction.
3. Ingredients of a sane project (with microservices and PaaS).
4. Using Python for that project.
5. Other tools and procedures that you need.
Microservices

- Independant
- Cooperating
- Scale well (e.g. Netflix)
- “Small”
- 12factor.net
- Way to handle big teams
Platform as a Service

- Cloud for applications, not (virtual) machines
- Encapsulates applications
- Eases connecting apps together
- Simplifies deployment
- Helps with logging

http://www.paasify.it/vendors
Microservices on PaaS

- The way to go
- Increase the benefits
- Easy scaling
- Adaptability
- Testable
- Measurable
Not a silver bullet

- Really painful without good automation
- Communication overhead
- Performance overhead
- Risky to start without a monolith

http://martinfowler.com/bliki/MonolithFirst.html
Microservices requirements

1. Twelve factor applications
2. Automated multi-tier testing
3. Continuous delivery pipeline
4. Insight/metrics
5. Proper management
6. Platform versioning
Why use Python for that?

- As many features/libraries as anything else (or more).
- Fast prototyping.
- Easy testing (but static type checking wouldn’t hurt...).
- Good at loose coupling
- Deterministic garbage collection (weakref)
- It’s enjoyable.
- More...
## Sufficient performance

- Don’t trust me! Or anyone! (with benchmarks)
- Falcon + uWSGI vs. Spring Boot + Tomcat

<table>
<thead>
<tr>
<th></th>
<th>Req/s</th>
<th>mean ms/req</th>
<th>failed reqs</th>
<th>50th pct &lt; (ms)</th>
<th>75th pct &lt; (ms)</th>
<th>95th pct &lt; (ms)</th>
<th>99th pct &lt; (ms)</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>Falcon</td>
<td>722</td>
<td>1490</td>
<td>2.8%</td>
<td>59</td>
<td>1038</td>
<td>11782</td>
<td>22376</td>
<td>52193</td>
</tr>
<tr>
<td>Spring</td>
<td>585</td>
<td>5924</td>
<td>0.7%</td>
<td>5421</td>
<td>6484</td>
<td>11293</td>
<td>28092</td>
<td>39639</td>
</tr>
</tbody>
</table>
The app

- Enter Falcon!
- Light!
- Fast!
- No magic!
- ...young...
- I’m not on the team

# app.py

```python
import falcon
import json

class SampleResource:
    @staticmethod
    def on_get(req, resp):
        resp.body = 'Hello world

app = falcon.API()
app.add_route('/', SampleResource())
```

http://falconframework.org/
# app.py

```python
import falcon
import json

class SampleResource:
    @staticmethod
def on_get(req, resp):
        resp.body = 'Hello world\n'

# THE NEW THING
@staticmethod
def on_post(req, resp):
    '''
    Given JSON input returns a JSON with only the keys that start with "A" (case insensitive).
    '''
    if req.content_type != 'application/json':
        raise falcon.HTTPUnsupportedMediaType('Media type needs to be application/json')
    # PYTHON 3
    body_json = json.loads(req.stream.read().decode('utf-8'))
    resp.body = json.dumps({key: value for key, value in body_json.items() if key.lower().startswith('a')})

app = falcon.API()
app.add_route('/', SampleResource())
```
CloudFoundry app

directory structure:

- example_app
  - example_app
    - app.py
  - tests
    - test_app.py
    - requirements.txt
  - service_tests
    - test_service.py
    - requirements.txt
- requirements.txt
- tox.ini
- manifest.yml
- runtime.txt
- .cfignore
manifest.yml

---

applications:
- name: example-app
  command: uwsgi --http :$VCAP_APP_PORT --module example_app:app # etc.
  memory: 128M
  buildpack: python_buildpack
services:
  - redis30-example
  - other-example-app-service
env:
  LOG_LEVEL: "INFO"
  VERSION: "0.0.1"
Continuous delivery

DO IT OR DIE
CD flow

$ git clone --recursive <app_repo>
$ tox
$ bumpversion micro
$ cf push
$ python3 test_e2e.py
$ cf target <production_env>
$ cf push
Unit testing - HTTP
import json
from falcon import testing
from falcon_app.app import app

class SampleTest(testing.TestBase):
    def setUp(self):
        super().setUp()
        self.api = app

    def test_sample_post(self, original_dict, expected_dict):
        response = self.simulate_request('/',
            decode='utf-8',
            method='POST',
            body=json.dumps({'abra': 123, 'kadabra': 4}),
            headers=[('Content-type', 'application/json')]
        )

        self.assertEqual(response,
            json.dumps({'abra': 123})
        )
Unit testing - pub/sub
Service testing

Fake environment

- test clinet
- mountebank
Tox config

- Unit and service test
- Only one Python version.
- No packaging (skipsdist=True)
- Full app analysis (coverage, pylint, etc.)
- Run on dev and CI machines
YOLO SWAGGINS

And the fellowship of the bling
<table>
<thead>
<tr>
<th>Method</th>
<th>Path</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>POST</td>
<td>/pet</td>
<td>Add a new pet to the store</td>
</tr>
<tr>
<td>PUT</td>
<td>/pet</td>
<td>Update an existing pet</td>
</tr>
<tr>
<td>GET</td>
<td>/pet/findByStatus</td>
<td>Finds Pets by status</td>
</tr>
<tr>
<td>GET</td>
<td>/pet/findByTags</td>
<td>Finds Pets by tags</td>
</tr>
<tr>
<td>DELETE</td>
<td>/pet/{petId}</td>
<td>Deletes a pet</td>
</tr>
<tr>
<td>GET</td>
<td>/pet/{petId}</td>
<td>Find pet by ID</td>
</tr>
<tr>
<td>POST</td>
<td>/pet/{petId}</td>
<td>Updates a pet in the store with form data</td>
</tr>
<tr>
<td>POST</td>
<td>/pet/{petId}/uploadImage</td>
<td>uploads an image</td>
</tr>
</tbody>
</table>
E2E/acceptance tests

- Done in staging env
- Run after each commit to master
- ...or nightly
- Only crucial journeys through the system
- Owned by everybody, monitored by selected
Monitoring

● In staging and production.
● State of PaaS resources.
● Periodically runs E2E.
● E.g. Zabbix
Logs and metrics

- All apps log to std out
- Cloud Foundry gathers all logs in a stream
- Logsearch: Cloud-scale ELK
- InfluxDB for real-time metrics
Management tips

● Every app needs an owner
● ...and an additional reviewer
● Review mercilessly
● Nobody is unquestionable
● Architecture visualisation
Platform deployments

- Custom implementation
- E.g. a big manifest binding others together
- Can increase the risk of coupling
● Sam Newman, *Building Microservices*, O'Reilly
● [http://martinfowler.com/bliki/MonolithFirst.html](http://martinfowler.com/bliki/MonolithFirst.html)
● [http://martinfowler.com/articles/microservice-testing/](http://martinfowler.com/articles/microservice-testing/)
● [http://docs.cloudfoundry.org/](http://docs.cloudfoundry.org/)