Python & DevOps

Your own heroku

jib.li

@sp4ke

- Freelance Full Stack Developer / DevOps
- CTO & CoFounder at Jib.li
- Python Enthusiast

Summary

1. Environment & Stack
2. Agile deployment with uWSGI

Our needs:

- Web Application
- Deep linking with social networks
- Agile development
- Community and available packages

Our needs:

- Web Application
- Deep linking with social networks
- Agile development
- Community and available packages

= Django + MongoDB + Github

Our needs:

- Web Application
- Deep linking with social networks
- Agile development
- Community and available packages

= Django + MongoDB + Github

gevent-socketio, zmq, celery, AWS boto ...

Local environment : The essentials

- Virtualenv
- PIP

Local environment : The essentials

- Virtualenv
- PIP
- But they have RVM !

Local environment : The essentials

- Virtualenv
- PIP
- But they have RVM !
- Pythonbrew: utahta/pythonbrew.git

- Pythonbrew: utahta/pythonbrew.git
  - Compile system independent pythons

```
$ pythonbrew install 2.7.3
$ pythonbrew use 2.7.3
$ pythonbrew list && which python
# pythonbrew pythons
  Python-2.7.3 (*)
/home/spike/.pythonbrew/pythons/Python-2.7.3/bin/python
```

- Pythonbrew: utahta/pythonbrew.git
  - Easy management of virtualenvs

```
$ pythonbrew venv create jibli
$ pythonbrew venv use jibli && which python && which pip

# Using `jibli` environment
# To leave an environment, simply run `deactivate`
/home/spike/.pythonbrew/venvs/Python-2.7.3/jibli/bin/python
/home/spike/.pythonbrew/venvs/Python-2.7.3/jibli/bin/pip
```

Environment bootstrap:

```
git clone jibli/project && cd project
pythonbrew create venv jibli && pythonbrew activate jibli
pip install -r dependencies.txt
```

Environment bootstrap:

```
git clone jibli/project && cd project
pythonbrew create venv jibli && pythonbrew activate jibli
pip install -r dependencies.txt --download-cache=CACHE
```

pip & dependencies.txt:

pip & dependencies.txt:

```
package-foo
package-bar==4.2

git+https://github.com/user/repo
```

pip & dependencies.txt:

```
package-foo
package-bar==4.2

git+https://github.com/user/repo
git+https://github.com/user/repo#egg=mon-package
```

pip & dependencies.txt:

```
package-foo
package-bar==4.2

git+https://github.com/user/repo
git+https://github.com/user/repo#egg=mon-package
git+https://github.com/user/repo@branch
```

pip & dependencies.txt:

```
package-foo
package-bar==4.2

git+https://github.com/user/repo
git+https://github.com/user/repo#egg=mon-package
git+https://github.com/user/repo@branch
```

- Quick dependencies update

```
pip freeze > dependencies.txt
```

MongoDB

- NoSQL, Schemaless, Document Oriented
- BSON data format
- Advantage: Python Dict -> JSON
- Good Python API pymongo

MongoDB

- MongoDB Javascript Console

```
$ mongo jibli
MongoDB shell version: 2.0.6
connecting to: jibli
> db.users.find( {'profil.age': 10} );
```

MongoDB

- MongoDB Javascript Console

```
$ mongo jibli
MongoDB shell version: 2.0.6
connecting to: jibli
> db.users.find( {'profil.age': 10} );
```

- PyMongo equivalent

```
u = pymongo.Connection(host='localhost', port=27017)['jibli']['users']
u.find( {'profil.age': 10} )
```

Local development

- Git branch feature
- Unit test
- Implement
- Test on local server (./manage.py runserver)
- Commit and merge on master branch

Agile deployment

- Many features require a production like environment:
  - OAuth Authentication and Social Networks
  - Async tasks, Celery (notifications, crons ...)
  - Push Notifications
  - Hard to clone a production environment in local

- Development Scenario
  - Start a new feature

```
$ git checkout -b feature
```

- Development Scenario
  - Start a new feature

    ```
    $ git checkout -b feature
    ```
  - Implement and push on dev server

    ```
    $ fab push
    ```

- Development Scenario
  - Start a new feature

    ```
    $ git checkout -b feature
    ```

  - Implement and push on dev server

    ```
    $ fab push
    ```

  - My branch is UP on feature.dev.com

- Development Scenario
  - Start a new feature

    ```
    $ git checkout -b feature
    ```

  - Implement and push on dev server

    ```
    $ fab push
    ```

  - My branch is UP on feature.dev.com
  - Remote access to the deployed app from local shell with a production environment (ie. restart, upgrade, ipython, mongo shell ...)

- Development Scenario
  - Start a new feature

    ```
    $ git checkout -b feature
    ```

  - Implement and push on dev server

    ```
    $ fab push
    ```

  - My branch is UP on feature.dev.com
  - Remote access to the deployed app from local shell with a production environment (ie. restart, upgrade, ipython, mongo shell ...)
  - Once satisfied merge on master and push on prod

Solution

- Nginx
- Github
- Fabric

+

- uWSGI

Solution

- Nginx
- Github
- Fabric

+

- uWSGI

= your heroku like solution

uWSGI

- Create developement stacks
- Host application clusters

uWSGI

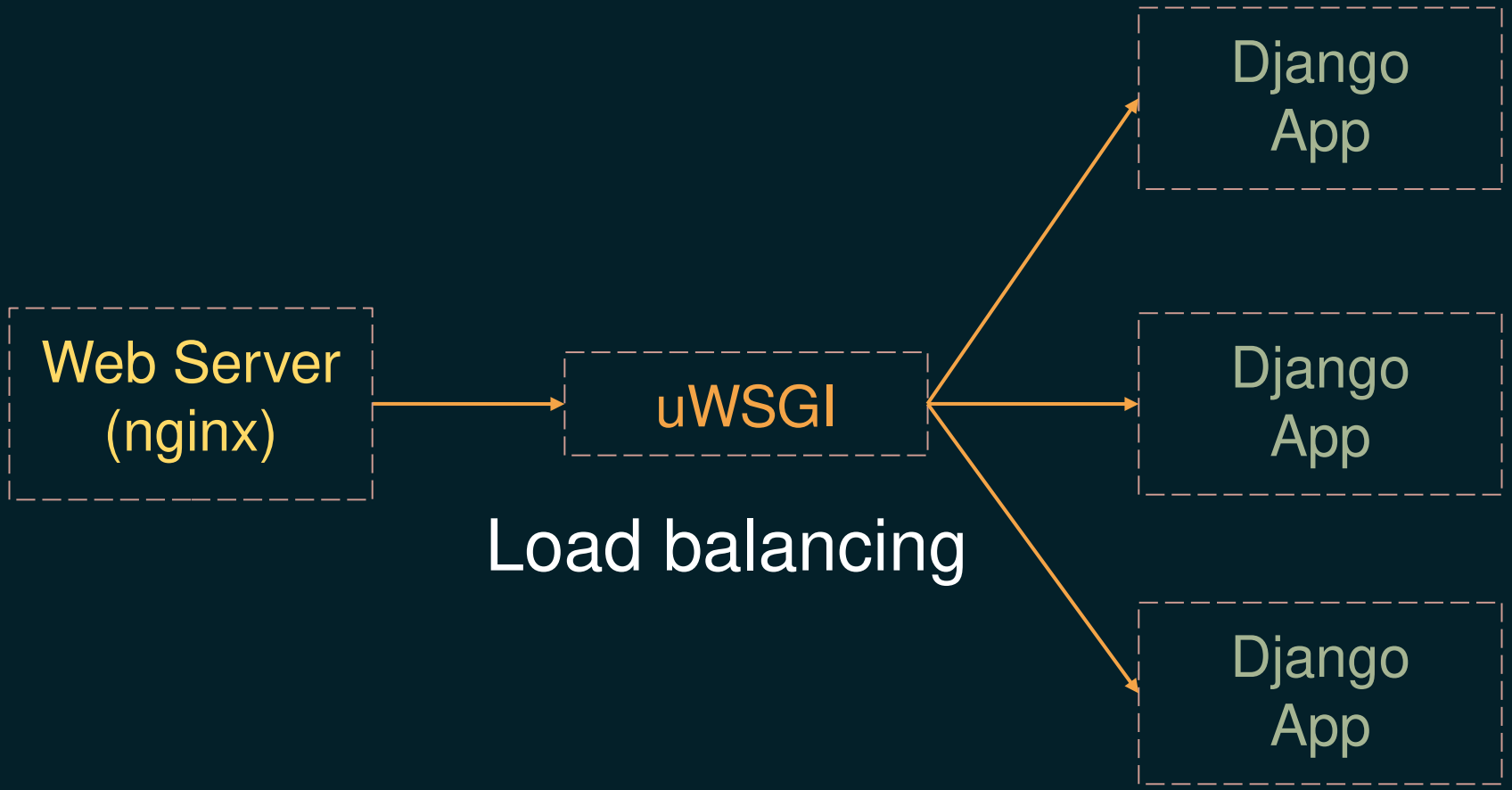
- how ?

WSGI Interface

also: FastCGI,CGI, PHP, Rack, ...

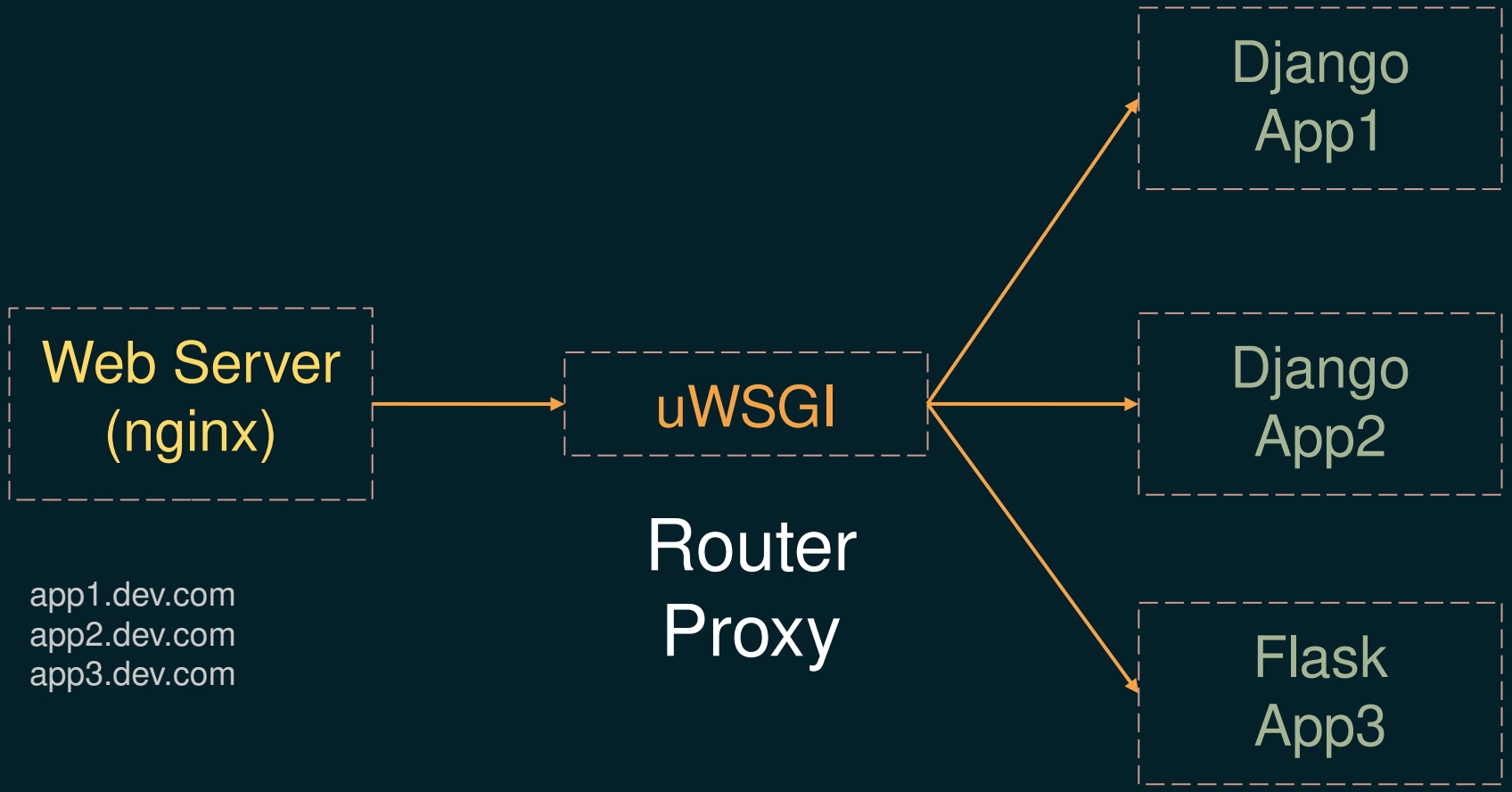
uWSGI

- how ?


Web Server
(nginx)
uWSGI
Load balancing
Django
App
Django
App
Django
App

uWSGI

- how ?


Django
App1
Web Server
(nginx)
uWSGI
Django
App2
Router
Proxy
app1.dev.com
app2.dev.com
app3.dev.com
Flask
App3

2. Agile deployment with uWSGI

- how ?

uWSGI Emperor

- Event based dynamic handling of applications (Vassals)

Default:

- Scan for config files in directories (.ini, .xml, .yml, .json ...)
- dir:// & glob:// for conf files monitoring
- Much more plugins available (mongodb, ampq, ldap ...)

glob:// plugin

```
uwsgi --emperor /opt/apps/*/*.ini
```

glob:// plugin

```
uwsgi --emperor "/opt/apps/*/*.ini"
```

One does not simply use glob patterns !

glob:// plugin

```
uwsgi --emperor "/opt/apps/*/*.ini"
```

Example:

- New file "/opt/apps/appn/uwsgi.ini"
  - Spawn vassal
- File modified
  - Restart vassal
- File removed
  - Kill vassal
- Emperor dies
  - All vassals die with him

2. Agile deployment with uWSGI

- Create a conf file for each deployed app ?

/opt/apps/app1/uwsgi.ini
/opt/apps/app2/uwsgi.ini
/opt/apps/appn/uwsgi.ini

2. Agile deployment with uWSGI

- Create a conf file for each deployed app ?

  /opt/apps/app1/uwsgi.ini
  /opt/apps/app2/uwsgi.ini
  /opt/apps/appn/uwsgi.ini

- ln -s
  - Use template conf files

  /opt/apps/template

ln -s /opt/apps/template /opt/apps/app1/app1.ini

Template conf file (Django App)

```
[uwsgi]
djangoproject = %d/app/
home = %d/virt
pythonpath = %d/
env = DJANGO_SETTINGS_MODULE=app.settings
chdir = %(djangoproject)
module = uwsgi_app
socket = 127.0.0.1:0

master = true
processes = 1
idle = 300

subscribe-to = 127.0.0.1:9999:%n.dev.com
logto = %d/log/uwsgi.log
```

Template conf file

```
[uwsgi]
djangoproject = %d/app/
```

- Use variables like here djangoproject
- Magic variables :
  - %d - Absolute path to configuration file
  - %n - Name of configuration file without extension

Template conf file

```
[uwsgi]
djangoproject = %d/app/
home = %d/virt
pythonpath = %d/
env = DJANGO_SETTINGS_MODULE=app.settings
```

- Define your app's virtualenv

Template conf file

```
[uwsgi]
djangoproject = %d/app/
home = %d/virt
pythonpath = %d/
env = DJANGO_SETTINGS_MODULE=app.settings
```

- Define your app's virtualenv
- Python search paths (You can repeat this one)

Template conf file

```
[uwsgi]
djangoproject = %d/app/
home = %d/virt
pythonpath = %d/
env = DJANGO_SETTINGS_MODULE=app.settings
```

- Define your app's virtualenv
- Python search paths (You can repeat this one)
- Custom environment variables

Template conf file (Django App)

```
[uwsgi]
chdir = %(djangoproject)
module = uwsgi_app
```

- Which module to run when starting application
  - django.core.handlers.wsgi:WSGIHandler()

- Best spot to run your custom scripts and setup environment before launching application
  - ie. compile static, zMQ sockets, syncdb ...

Up until now we can:

- Git push origin feature
- Clone feature in remote /opt/apps/feature
- Prepare dirs structure
- Create venv & install dependencies
- Symllink to the uWSGI template file
- uWSGI emperor launches feature app

We still need to:

- access our feature using subdomains
  - `feature.dev.com`

FastRouter

- Proxy/Load Balancing/Router
- Speaks uWSGI protocol
- Unlimited setup possibilities
- Key/Value store

FastRouter

- Proxy/Load Balancing/Router
- Speaks uWSGI protocol
- Unlimited setup possibilities
- Key/Value store

Example:

```
uwsgi --fastrouter  /tmp/fastrouter.socket  \ --
fastrouter-subscription-server 127.0.0.1:9999
```

Always use unix sockets instead of localhost tcp

Nginx

```
server {
        listen                  80;
        server_name             dev.com *.dev.com;
        location / {
                include /etc/nginx/uwsgi_params;
                uwsgi_param UWSGI_FASTROUTER_KEY $host;
                uwsgi_pass unix: /tmp/fastrouter.socket;
        }
}
```

Template conf file (Django App)

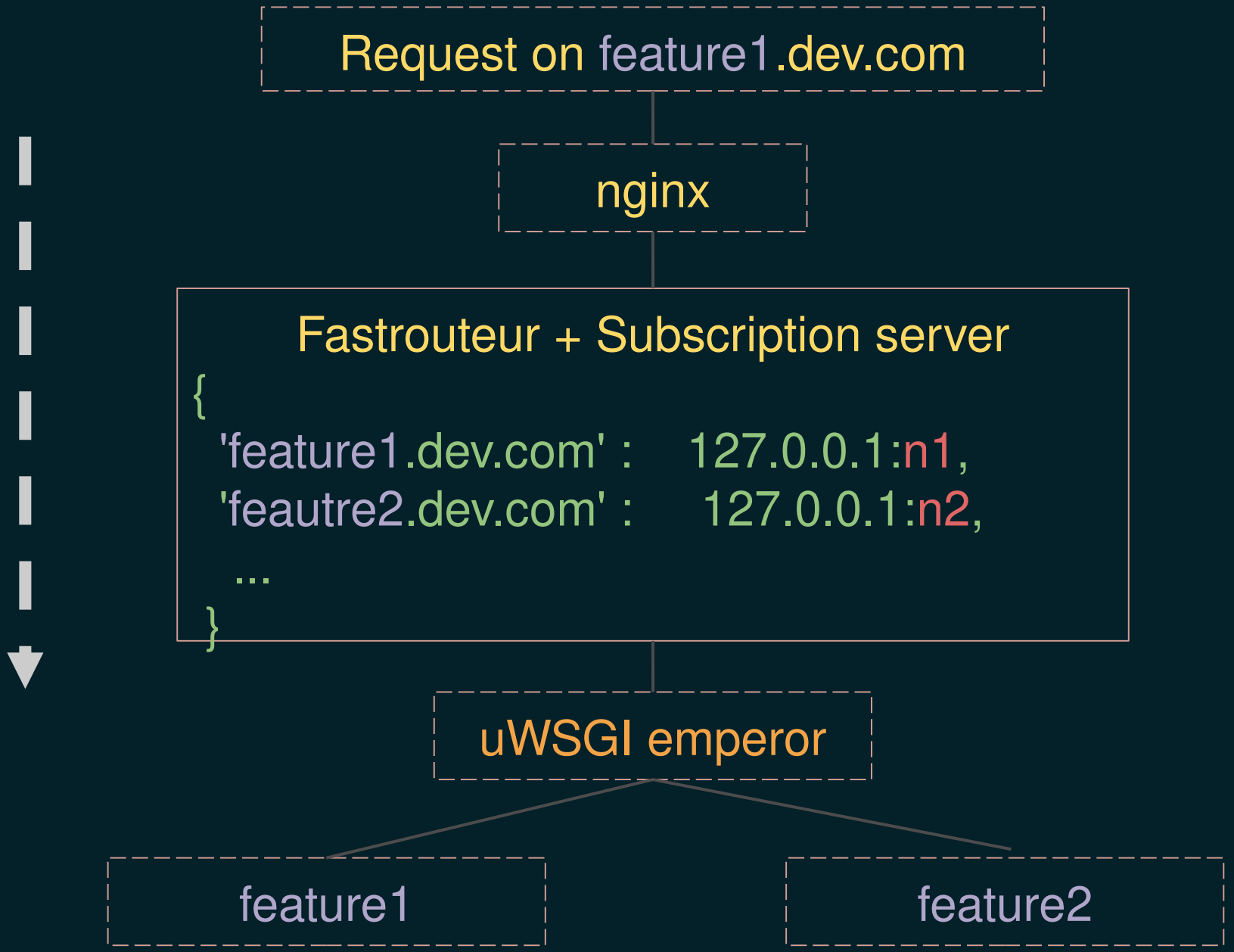
```
[uwsgi]
...
socket = 127.0.0.1:0
subscribe-to = 127.0.0.1:9999:%n.dev.com
```

Template conf file (Django App)

```
[uwsgi]
...
socket = 127.0.0.1:0
subscribe-to = 127.0.0.1:9999:%n.dev.com
```

/opt/apps/feature1/feature1.ini

2. Agile deployment with uWSGI



HTOP

Deploying feature1

```
├── supervisord
│   ├── /uwsgi --fastrouter ... --emperor /opt/apps/*/*.ini
│   │   ├── /usr/local/bin/uwsgi #fastrouter
│   │   └── /usr/local/bin/uwsgi #master
│   │       ├──uwsgi --ini /opt/apps/feature1/feature1.ini
```

HTOP

Deploying feature2

```
├── supervisord
│   ├── /uwsgi --fastrouter ... --emperor /opt/apps/*/*.ini
│   │   ├── /usr/local/bin/uwsgi #fastrouter
│   │   └── /usr/local/bin/uwsgi #master
│   │       ├──uwsgi --ini /opt/apps/feature1/feature1.ini
│   │       └──uwsgi --ini /opt/apps/feature2/frature2.ini
```

End

twitter: @sp4ke

email: spike@jib.li

slides: sp4ke.com/pythondevops