INTRODUCTION À FALCON

Le cowboy qui sert des requêtes plus vite que son ombre

Ryan Lahfa, alias Raito Bezarius

4 février 2015

masterancpp@gmail.com

INTRODUCTION

A propos de moi

Ryan Lahfa, mieux connu sur Internet comme *Raito Bezarius* :

A propos de moi

Ryan Lahfa, mieux connu sur Internet comme *Raito Bezarius* :

- Développeur *full-stack*

A propos de moi

Ryan Lahfa, mieux connu sur Internet comme *Raito Bezarius* :

- Développeur *full-stack*
- GitHub : *RaitoBezarius*

A propos de moi

Ryan Lahfa, mieux connu sur Internet comme *Raito Bezarius* :

- Développeur *full-stack*
- GitHub : *RaitoBezarius*
- Twitter : *@Ra1t0_Bezar1us*

« Il semble que la perfection soit atteinte non quand il n'y a plus rien à ajouter, mais quand il n'y a plus rien à retrancher. »

— Antoine de Saint-Exupéry

Falcon est. . .

Falcon est. . .

- Un framework web simple et minimal

Falcon est. . .

- Un framework web simple et minimal
- Il est compatible WSGI (Gunicorn, etc.)

Falcon est. . .

- Un framework web simple et minimal
- Il est compatible WSGI (Gunicorn, etc.)
- Aussi rapide qu'un faucon

Falcon est. . .

- Un framework web simple et minimal
- Il est compatible WSGI (Gunicorn, etc.)
- Aussi rapide qu'un faucon
- Deux dépendances : *Six* et *mimeparse*

Falcon est. . .

- Un framework web simple et minimal
- Il est compatible WSGI (Gunicorn, etc.)
- Aussi rapide qu'un faucon
- Deux dépendances : *Six* et *mimeparse*
- 100% de test coverage

Falcon est. . .

- Un framework web simple et minimal
- Il est compatible WSGI (Gunicorn, etc.)
- Aussi rapide qu'un faucon
- Deux dépendances : ***Six*** et ***mimeparse***
- 100% de test coverage
- Supporte Python 2.6/2.7 et Python 3.3/3.4 (avec en bonus PyPy)

HTTP ET REST, L'ÉPOPÉE DES API

Le HTTP en 9 lignes

```
curl -X ...
```

Le HTTP en 9 lignes

```
curl -X ...

    HTTP/1.1 200 OK
    Connection: close
    Date: Tue, 03 Feb 2015 21:29:44 GMT
    Server: gunicorn/17.5
    content-length: 23
    content-type: application/json
```

Le HTTP en 9 lignes

```
curl -X ...

    HTTP/1.1 200 OK
    Connection: close
    Date: Tue, 03 Feb 2015 21:29:44 GMT
    Server: gunicorn/17.5
    content-length: 23
    content-type: application/json

    {
        "name": "Paris.cpp 1"
    }
```

HTTP : Décomposition des codes de status

Il en existe 5 familles *officiels* :

HTTP : Décomposition des codes de status

Il en existe 5 familles *officiels* :

- 1xx : Informatif.

HTTP : Décomposition des codes de status

Il en existe 5 familles *officiels* :

- 1xx : Informatif.
- 2xx : Rien de mauvais s'est passé.

HTTP : Décomposition des codes de status

Il en existe 5 familles *officiels* :

- 1xx : Informatif.
- 2xx : Rien de mauvais s'est passé.
- 3xx : Redirections.

HTTP : Décomposition des codes de status

Il en existe 5 familles *officiels* :

- 1xx : Informatif.
- 2xx : Rien de mauvais s'est passé.
- 3xx : Redirections.
- 4xx : Problèmes côté client. (mauvaise entrée, etc.)

HTTP : Décomposition des codes de status

Il en existe 5 familles *officiels* :

- 1xx : Informatif.
- 2xx : Rien de mauvais s'est passé.
- 3xx : Redirections.
- 4xx : Problèmes côté client. (mauvaise entrée, etc.)
- 5xx : Problèmes côté serveur. (exceptions non attrapées, etc.)

HTTP : Les headers

Cela fonctionne comme un dictionnaire qui ***ne prend pas en compte la casse*** :

HTTP : Les headers

Cela fonctionne comme un dictionnaire qui ***ne prend pas en compte la casse*** :

```
content-length: 23
content-type: application/json
```

HTTP : Les headers

Cela fonctionne comme un dictionnaire qui ***ne prend pas en compte la casse*** :

```
content-length: 23
content-type: application/json
```

```
h = dict([(key.strip(), value.strip()) for key, value
in (line.split(' :') for line in lines.split('\n'))])
```

HTTP : Les méthodes

On appelle les méthodes, les mots qui servent de « commande » afin de parler avec HTTP, de manière officielle, on a :

HTTP : Les méthodes

On appelle les méthodes, les mots qui servent de « commande » afin de parler avec HTTP, de manière officielle, on a :

- HEAD

HTTP : Les méthodes

On appelle les méthodes, les mots qui servent de « commande » afin de parler avec HTTP, de manière officielle, on a :

- HEAD
- GET

HTTP : Les méthodes

On appelle les méthodes, les mots qui servent de « commande » afin de parler avec HTTP, de manière officielle, on a :

- HEAD
- GET
- POST

HTTP : Les méthodes

On appelle les méthodes, les mots qui servent de « commande » afin de parler avec HTTP, de manière officielle, on a :

- HEAD
- GET
- POST
- PUT

HTTP : Les méthodes

On appelle les méthodes, les mots qui servent de « commande » afin de parler avec HTTP, de manière officielle, on a :

- HEAD
- GET
- POST
- PUT
- DELETE

HTTP : Le corps

Le corps actuellement a des types différents, parfois, c'est du texte, parfois du JSON, parfois du XML. Toutefois, grâce aux headers, et particulièrement ***Content-Type***, on connaît le type du corps et on peut l'analyser.

HTTP : Le corps

```
{
    "name" : "Paris.cpp 1",
    "participants" : [
     {
         "name" : "Raito Bezarius"
        },
        {
         "name" : "Random people 1"
        }
    ]
}
```

HTTP : Requêtes et réponses

Le client (le navigateur) effectue une requête au serveur, et celui-ci crée une réponse. Les deux contiennent des headers, des bodies (corps).

HTTP : Requêtes et réponses

Le client (le navigateur) effectue une requête au serveur, et celui-ci crée une réponse. Les deux contiennent des headers, des bodies (corps).
Protocole ***stateless*** :

- Il ne possède pas d'état.

HTTP : Requêtes et réponses

Le client (le navigateur) effectue une requête au serveur, et celui-ci crée une réponse. Les deux contiennent des headers, des bodies (corps).
Protocole ***stateless*** :

- Il ne possède pas d'état.
- On est obligé de renvoyer tout à chaque fois.

HTTP : Requêtes et réponses

Le client (le navigateur) effectue une requête au serveur, et celui-ci crée une réponse. Les deux contiennent des headers, des bodies (corps).
Protocole ***stateless*** :

- Il ne possède pas d'état.
- On est obligé de renvoyer tout à chaque fois.
- Mais on peut contourner (cookies, etag, etc).

HTTP : Requêtes et réponses

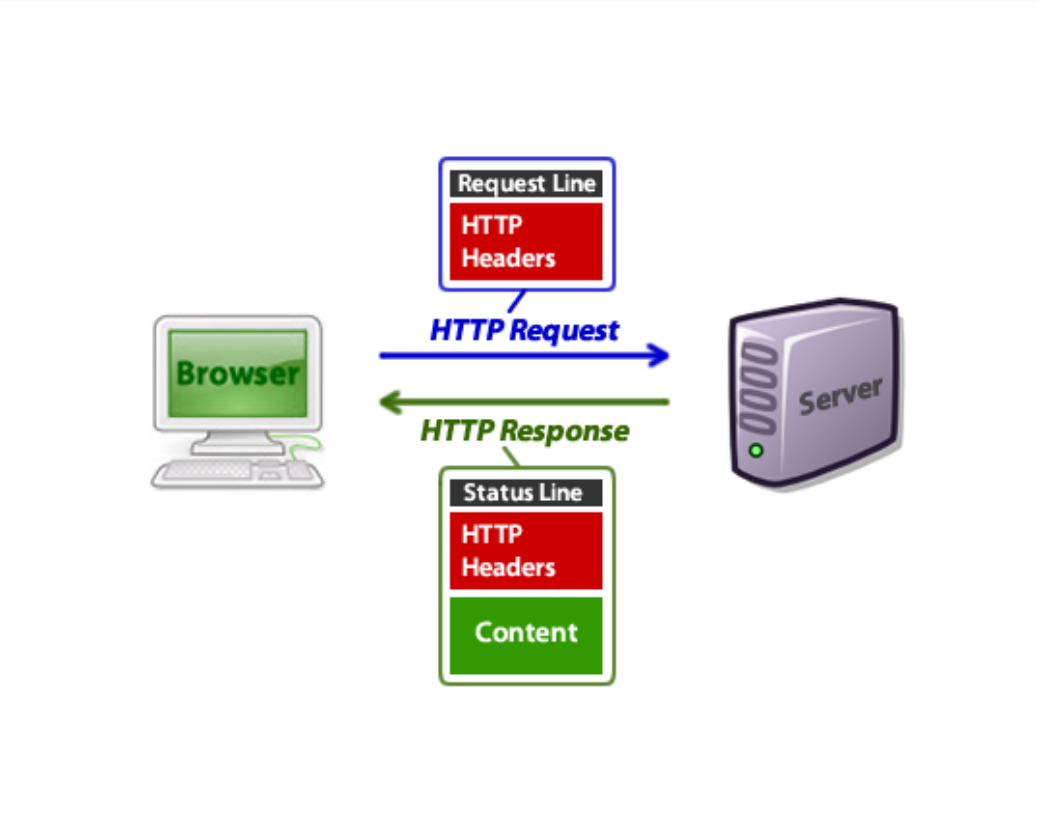

Figure : Diagramme des échanges HTTP (simplifié)

HTTP : Le routage en quelques mots

Quant au routing, qui n'est pas une spécification propre à HTTP, mais plutôt un concept. Il s'agit d'associer à une URI (Uniform Resource Identifier), une ressource.

HTTP : Le routage en quelques mots

Quant au routing, qui n'est pas une spécification propre à HTTP, mais plutôt un concept. Il s'agit d'associer à une URI (Uniform Resource Identifier), une ressource. Exemple :

```
GET /meetups/python/everywhere HTTP/1.1
Host: localhost:8000
```

REST : Un style avant tout

REST est un style d'écriture d'API qui suit le standard HTTP, on le décrit comme :

REST : Un style avant tout

REST est un style d'écriture d'API qui suit le standard HTTP, on le décrit comme :

- HEAD/GET sont des opérations de lecture-seule et non dangereux.

REST : Un style avant tout

REST est un style d'écriture d'API qui suit le standard HTTP, on le décrit comme :

- HEAD/GET sont des opérations de lecture-seule et non dangereux.
- PUT/DELETE modifie une ressource.

REST : Un style avant tout

REST est un style d'écriture d'API qui suit le standard HTTP, on le décrit comme :

- HEAD/GET sont des opérations de lecture-seule et non dangereux.
- PUT/DELETE modifie une ressource.
- POST crée une nouvelle entité par rapport à l'URI.

REST : Un style avant tout

REST est un style d'écriture d'API qui suit le standard HTTP, on le décrit comme :

- HEAD/GET sont des opérations de lecture-seule et non dangereux.
- PUT/DELETE modifie une ressource.
- POST crée une nouvelle entité par rapport à l'URI.
- PATCH modifie une partie de l'entité.

WSGI : Une approche Python du web dev

Le WSGI (Web Server Gateway Interface) est une approche développé par Python afin de faire du dev web, en fait, ça se résume à :

WSGI : Une approche Python du web dev

Le WSGI (Web Server Gateway Interface) est une approche développé par Python afin de faire du dev web, en fait, ça se résume à :

- Headers HTTP + méta-données placés dans un dictionnaire

WSGI : Une approche Python du web dev

Le WSGI (Web Server Gateway Interface) est une approche développé par Python afin de faire du dev web, en fait, ça se résume à :

- Headers HTTP + méta-données placés dans un dictionnaire
- Le contenu du corps est stocké comme un flux (stream)

APERÇU DE FALCON

Hello, World!

```
import falcon

class ParisMeetup(object):
    def on_get(self, request, response):
        language = request.get_param('language')
        if not (language.lower() == "python"):
            raise falcon.exceptions.HTTPNotFound()

        response.body = "Paris.py 6 est en cours!"
        response.status = falcon.HTTP_200


api = falcon.API()
api.add_route('/paris/current_meetup', ParisMeetup())
```

Faire une API

Faire une API

Cela revient à :

```
zip(routes, ressources)
```

Or,

Faire une API

Cela revient à :

```
zip(routes, ressources)
```

Or,

- Une route : Une URI associée à une méthode HTTP

Faire une API

Cela revient à :

```
zip(routes, ressources)
```

Or,

- Une route : Une URI associée à une méthode HTTP
- Une ressource : Un « crafteur » de réponse qui contient la logique de l'application

Un exemple plus complexe

Essayons de fabriquer une API qui permettrait de donner tous les meetups en cours sur un langage de programmation d'un endroit :

Un exemple plus complexe

Essayons de fabriquer une API qui permettrait de donner tous les meetups en cours sur un langage de programmation d'un endroit : Par exemple, à Paris pour Python on a quoi...

Un exemple plus complexe

Essayons de fabriquer une API qui permettrait de donner tous les meetups en cours sur un langage de programmation d'un endroit : Par exemple, à Paris pour Python on a quoi... Faisons un design de l'API d'abord, on écrit les routes pour se donner un ordre d'idée de la forme qu'on aura :

Un exemple plus complexe

Essayons de fabriquer une API qui permettrait de donner tous les meetups en cours sur un langage de programmation d'un endroit : Par exemple, à Paris pour Python on a quoi... Faisons un design de l'API d'abord, on écrit les routes pour se donner un ordre d'idée de la forme qu'on aura :

```
GET /meetups/language/place
DELETE /meetups/language/place
PUT /meetups/language/place
```

Un exemple plus complexe

Essayons de fabriquer une API qui permettrait de donner tous les meetups en cours sur un langage de programmation d'un endroit : Par exemple, à Paris pour Python on a quoi... Faisons un design de l'API d'abord, on écrit les routes pour se donner un ordre d'idée de la forme qu'on aura :

```
GET /meetups/language/place
DELETE /meetups/language/place
PUT /meetups/language/place
```

On aurait pu implémenter POST pour faire un nouveau meetup. Mais c'est laissé en exercice à l'audience !

Un exemple plus complexe

On résume :

Un exemple plus complexe

On résume :

GET Ça sera pour savoir si il existe un meetup d'un langage (language) à un endroit (place). Cette méthode renverra du JSON contenant la liste, ou une erreur 404 si il n'existe ***aucun*** meetup

Un exemple plus complexe

On résume :

GET Ça sera pour savoir si il existe un meetup d'un langage (language) à un endroit (place). Cette méthode renverra du JSON contenant la liste, ou une erreur 404 si il n'existe ***aucun*** meetup

DELETE Ça sera pour supprimer un meetup

Un exemple plus complexe

On résume :

GET Ça sera pour savoir si il existe un meetup d'un langage (language) à un endroit (place). Cette méthode renverra du JSON contenant la liste, ou une erreur 404 si il n'existe ***aucun*** meetup

DELETE Ça sera pour supprimer un meetup

PUT Cela sera utile pour changer des paramètres comme si le meetup se déroule ou les horaires du meetup. Ici, par abus de simplicité, on dira que ça consiste à dire que le meetup se déroule.

Un exemple plus complexe

```
class MeetupResource(object):

    def __init__(self, meetup_db):
        self._db = meetup_db

    def on_get(self, request, response, ...):
        meetups = self._db.get(language, place)
        if meetups is None :
            raise falcon.exceptions.HTTPNotFound()
        response.content_type = 'application/json'
        response.body = json.dumps(meetups)

    ...
```

Un exemple plus complexe

```
class MeetupResource(object):
    ...

    def on_delete(self, request, response, ...):
        self._db.delete(language, place)
        response.status = falcon.HTTP_204
```

Un exemple plus complexe

```
class MeetupResource(object):
    ...

    def on_put(self, request, response, ...):
        if not self._db.exists(language, place):
            raise falcon.exceptions.HTTPNotFound()

        self._db.set_running(language, place)
        if not (language.lower() == 'python'):
            response.status = falcon.HTTP_799
    ...
```

Un exemple plus complexe

```
...

db = MockupDB()
api = falcon.API()

api.add_route('/meetups/{language}/{place}',
                                    MeetupResource(db))
```

Testons !

```
gunicorn meetup:api
```

Testons !

```
gunicorn meetup:api

[5973] [INFO] Starting gunicorn 17.5
[5973] [INFO] Listening at: http://127.0.0.1:8000 (5973)
[5973] [INFO] Using worker: sync
[5978] [INFO] Booting worker with pid: 5978
```

Testons !

```
gunicorn meetup:api

[5973] [INFO] Starting gunicorn 17.5
[5973] [INFO] Listening at: http://127.0.0.1:8000 (5973)
[5973] [INFO] Using worker: sync
[5978] [INFO] Booting worker with pid: 5978
```

Mais. . . ? Comment on teste ? C’est pas du HTML !

TESTER UNE API AVEC FALCON

Le testing manuel

Souvent, on utilise ***curl*** pour se faire ses petits tests.

Le testing manuel

Souvent, on utilise ***curl*** pour se faire ses petits tests.
Je vous propose ***httpie*** :

Le testing manuel

Souvent, on utilise ***curl*** pour se faire ses petits tests.
Je vous propose ***httpie*** :

- C'est basé sur ***python-requests***

Le testing manuel

Souvent, on utilise ***curl*** pour se faire ses petits tests.
Je vous propose ***httpie*** :

- C'est basé sur ***python-requests***
- C'est human-friendly.

Le testing manuel

Souvent, on utilise ***curl*** pour se faire ses petits tests.
Je vous propose ***httpie*** :

- C'est basé sur ***python-requests***
- C'est human-friendly.
- Gestion du HTTP au top.

Le testing manuel

Souvent, on utilise ***curl*** pour se faire ses petits tests.
Je vous propose ***httpie*** :

- C'est basé sur ***python-requests***
- C'est human-friendly.
- Gestion du HTTP au top.

Le testing manuel

Souvent, on utilise ***curl*** pour se faire ses petits tests.
Je vous propose ***httpie*** :

- C'est basé sur ***python-requests***
- C'est human-friendly.
- Gestion du HTTP au top.

```
pip install httpie
```

Alors, on teste ?

```
http get localhost:8000/meetups/c++/paris
```

Alors, on teste ?

```
http get localhost:8000/meetups/c++/paris
```

```
HTTP/1.1 200 OK
Connection: close
Date: Tue, 03 Feb 2015 21:29:44 GMT
Server: gunicorn/17.5
content-length: 23
content-type: application/json

{
    "name": "Paris.cpp 1"
}
```

Alors, on teste ?

```
http get localhost:8000/meetups/php/paris
```

Alors, on teste ?

```
http get localhost:8000/meetups/php/paris
```

```
HTTP/1.1 404 Not Found
Connection: close
Date: Tue, 03 Feb 2015 21:30:44 GMT
Server: gunicorn/17.5
content-length: 0
```

Alors, on teste ?

```
http delete localhost:8000/meetups/c++/paris
```

Alors, on teste ?

```
http delete localhost:8000/meetups/c++/paris
```

```
HTTP/1.1 204 No Content
Connection: close
Date: Tue, 03 Feb 2015 21:32:53 GMT
Server: gunicorn/17.5
```

Et les tests unitaires, dans l'histoire ?

C'est bien de tester manuellement parfois. Mais c'est inefficace à long terme :

Et les tests unitaires, dans l'histoire ?

C'est bien de tester manuellement parfois. Mais c'est inefficace à long terme :

- On doit forcément TOUT tester pour prévenir la régression, alors que des tests automatisés sont plus cools !

Et les tests unitaires, dans l'histoire ?

C'est bien de tester manuellement parfois. Mais c'est inefficace à long terme :

- On doit forcément TOUT tester pour prévenir la régression, alors que des tests automatisés sont plus cools !

Avec Falcon, on peut utiliser un helper de Falcon et carrément utiliser *python-requests*.

Les tests unitaires avec Falcon

Les développeurs Python habitués aux Unit Test utilisent le module *unittest* en créant une classe, dérivant de *unittest.TestCase* qui implémentera des tests automatiques.

Reprenons nos exemples !

```
import unittest
import falcon.testing
import meetup

class MeetupTests(unittest.TestCase):
    def setUp(self):
    def tearDown(self):

    def simulate_request(self, path, **kwargs):

    def test_meetup_existence(self):
    def test_meetup_inexistence(self):
    def test_meetup_deletion(self):
    def test_method_not_allowed(self):
```

Tests Unitaires : Configuration et nettoyage

```
class MeetupTests(unittest.TestCase):

    def setUp(self):
        self.app = meetup.api
        self.mock = falcon.testing.StartResponseMock()
        self.paris_py_path = '/meetups/python/paris'
        self.paris_cpp_path = '/meetups/c++/paris'
        self.simulate_request(self.paris_py_path,
            method='POST', body='name=Paris.py 6')
        self.simulate_request(self.paris_cpp_path,
         method='POST', body='name=Paris.cpp 1')

     ...
```

Tests Unitaires : Configuration et nettoyage

```
class MeetupTests(unittest.TestCase):
    ...

    def tearDown(self):
        self.simulate_request(self.paris_py_path,
            method='DELETE')
        self.simulate_request(self.paris_cpp_path,
            method='DELETE')
```

Tests Unitaires : Simuler une requête

```
class MeetupTests(unittest.TestCase):
    ...

    def simulate_request(self, path, **kwargs):
        env = falcon.testing.create_environ(
            path=path, **kwargs)
        return self.app(env, self.mock)
```

Tests Unitaires : Les tests

```
class MeetupTests(unittest.TestCase):
    ...

    def test_meetup_existence(self):
        self.simulate_request(self.paris_py_path,
          method='GET')
        self.assertEqual(self.mock.status,
            falcon.HTTP_200)

    def test_meetup_inexistence(self):
        self.simulate_request('/meetups/php/paris',
            method='GET')
        self.assertEqual(self.mock.status,
             falcon.HTTP_404)
```

Tests Unitaires : Les tests

```
class MeetupTests(unittest.TestCase):
 ...

    def test_meetup_deletion(self):
        self.simulate_request(self.paris_cpp_path,
            method='DELETE')
        self.assertEqual(self.mock.status,
            falcon.HTTP_204)

    def test_method_not_allowed(self):
        self.simulate_request(self.paris_py_path,
            method='PATCH')
        self.assertEqual(self.mock.status,
            falcon.HTTP_405)
```

Lancer les tests unitaires

En Python, il existe plusieurs moyen de lancer des tests unitaires, le moyen de base est ***unittest*** en exécutant le fichier contenant les tests.

1. BATTERIES INCLUDED

Lancer les tests unitaires

En Python, il existe plusieurs moyen de lancer des tests unitaires, le moyen de base est ***unittest*** en exécutant le fichier contenant les tests. Ici, on utilisera ***nosetests*** qui est une lib plus avancée que ***unittest*** fourni de base[1] avec Python :

1. BATTERIES INCLUDED

Lancer les tests unitaires

En Python, il existe plusieurs moyen de lancer des tests unitaires, le moyen de base est ***unittest*** en exécutant le fichier contenant les tests. Ici, on utilisera ***nosetests*** qui est une lib plus avancée que ***unittest*** fourni de base[1] avec Python :

```
nosetests -v units_tests.py
```

1. BATTERIES INCLUDED

Lancer les tests unitaires

En Python, il existe plusieurs moyen de lancer des tests unitaires, le moyen de base est ***unittest*** en exécutant le fichier contenant les tests. Ici, on utilisera ***nosetests*** qui est une lib plus avancée que ***unittest*** fourni de base[1] avec Python :

```
nosetests -v units_tests.py
```

```
....
----------------------
Ran 4 tests in 0.162s

OK
```

1. BATTERIES INCLUDED

Et les autres libs dans l'histoire ?

Pourquoi Falcon est né ?

En réalité, on peut voir Falcon comme encore un nouveau framework mais pour reprendre les termes de l'auteur :

Pourquoi Falcon est né ?

En réalité, on peut voir Falcon comme encore un nouveau framework mais pour reprendre les termes de l'auteur :

- Les frameworks web Python actuels n'ont pas des ***perfs géniales*** lorsqu'ils sont soumis à des grosses charges.

Pourquoi Falcon est né ?

En réalité, on peut voir Falcon comme encore un nouveau framework mais pour reprendre les termes de l'auteur :

- Les frameworks web Python actuels n'ont pas des ***perfs géniales*** lorsqu'ils sont soumis à des grosses charges.
- La plupart des framework sont embarqués en fait avec des lib de templating, des fonctionnalités shiny qui sont utiles uniquement lorsqu'on fait un site. Elles peuvent augmenter la chance de faille de sécurité, gâchent de la RAM.

Pourquoi Falcon est né ?

En réalité, on peut voir Falcon comme encore un nouveau framework mais pour reprendre les termes de l'auteur :

- Les frameworks web Python actuels n'ont pas des ***perfs géniales*** lorsqu'ils sont soumis à des grosses charges.
- La plupart des framework sont embarqués en fait avec des lib de templating, des fonctionnalités shiny qui sont utiles uniquement lorsqu'on fait un site. Elles peuvent augmenter la chance de faille de sécurité, gâchent de la RAM.

Pourquoi Falcon est né ?

En réalité, on peut voir Falcon comme encore un nouveau framework mais pour reprendre les termes de l'auteur :

- Les frameworks web Python actuels n'ont pas des ***perfs géniales*** lorsqu'ils sont soumis à des grosses charges.
- La plupart des framework sont embarqués en fait avec des lib de templating, des fonctionnalités shiny qui sont utiles uniquement lorsqu'on fait un site. Elles peuvent augmenter la chance de faille de sécurité, gâchent de la RAM.

La ***raison d'être*** de Falcon est de ***résoudre*** ces problèmes.

Mais Falcon n'est pas la solution ultime

Certes, Falcon est vraiment bon pour faire des API. Mais ce n'est pas le meilleur choix partout :

Mais Falcon n'est pas la solution ultime

Certes, Falcon est vraiment bon pour faire des API. Mais ce n'est pas le meilleur choix partout :

DRY Si vous switchez entre votre web app et votre API en changeant de framework systématiquement, ça n'a pas de sens. Il est plus intéressant de prendre un framework *moins spécialisé*.

Mais Falcon n'est pas la solution ultime

Certes, Falcon est vraiment bon pour faire des API. Mais ce n'est pas le meilleur choix partout :

DRY Si vous switchez entre votre web app et votre API en changeant de framework systématiquement, ça n'a pas de sens. Il est plus intéressant de prendre un framework ***moins spécialisé***.

Usine à gaz Falcon est tout sauf une usine à gaz, donc, il faut se préparer à produire des lignes de plus en échange d'une plus grande liberté. Pour le Quick & Dirty, il faudra aller voir Flask/Django, etc.

Mais Falcon n'est pas la solution ultime

Certes, Falcon est vraiment bon pour faire des API. Mais ce n'est pas le meilleur choix partout :

DRY Si vous switchez entre votre web app et votre API en changeant de framework systématiquement, ça n'a pas de sens. Il est plus intéressant de prendre un framework ***moins spécialisé***.

Usine à gaz Falcon est tout sauf une usine à gaz, donc, il faut se préparer à produire des lignes de plus en échange d'une plus grande liberté. Pour le Quick & Dirty, il faudra aller voir Flask/Django, etc.

Maturité Falcon est un projet né en 2013. Il a ***moins*** été testé sur le terrain que ses grands frères dans le domaine.

Un petit benchmark

Table : Comparatif sous Python 2.7.6

Un petit benchmark

Table : Comparatif sous Python 2.7.6

Frameworks	Type	requêtes/s	µs/requêtes	Performance
Falcon (0.1.8)	Simple	24 358	41	8x
Falcon (0.1.8)	Étendu	17 787	56	6x
Bottle (0.11.6)	Simple	13 623	73	4x
Werkzeug (0.9.4)	Simple	5 163	194	2x
Flask (0.10.1)	Simple	3 041	329	1x

CONCLUSION

Résumé

Falcon est un choix intéressant pour une API aujourd'hui :

Résumé

Falcon est un choix intéressant pour une API aujourd'hui :

- On fait de plus en plus de sites avec des frameworks frontend JavaScript qui fetch des données d'une API.

Résumé

Falcon est un choix intéressant pour une API aujourd'hui :

- On fait de plus en plus de sites avec des frameworks frontend JavaScript qui fetch des données d'une API.
- La productivité que permet Falcon.

Résumé

Falcon est un choix intéressant pour une API aujourd'hui :

- On fait de plus en plus de sites avec des frameworks frontend JavaScript qui fetch des données d'une API.
- La productivité que permet Falcon.
- Sa rapidité en général.

A propos des slides

Ces slides ont été faits en LaTeX, vous pouvez retrouver le code source sur GitHub (ainsi que la petite API) dans le repo suivant :

`github.com/RaitoBezarius/falcon-introduction`

En licence WTFPL [2], naturellement.

Le thème beamer utilisé est le mtheme (disponible sur GitHub) fait par Matthias Vogelgesang (matze sur Github).

2. J'avais pas d'idées lorsque j'allais l'open-sourcer

Questions ?

Merci à tous d'avoir assisté à cette présentation !