Hands-on Session

Build a RESTful WS on top of MongoDB with Python Eve

17th September 2016

Session’s goals

1. Python Eve installation and setup
2. MongoDB basic installation (WDTCS data set)
3. Expanding all to something more real (CRUD, HA, …)

Session’s goals

1. Python Eve installation and setup
2. MongoDB basic installation (WDTCS data set)
3. Expanding all to something more real (CRUD, HA, …)

1. Python Eve installation and setup

- We'll go over the following steps:

  1. Virtualenv setup for our API
  2. Basic setup that we'll use along the session
  3. RESTful WS test out with Postman

1.1 Virtualenv setup for our API

- It is as easy as:

```
Rauls-MacBook-Pro:~ rmarin$ mkdir pyday
Rauls-MacBook-Pro:~ rmarin$ virtualenv pyday
New python executable in /Users/rmarin/pyday/bin/python
Installing setuptools, pip, wheel...done.
Rauls-MacBook-Pro:~ rmarin$ source pyday/bin/activate
(pyday) Rauls-MacBook-Pro:~ rmarin$ pip install Eve

Collecting Eve
Collecting jinja2<3.0,>=2.7.2 (from Eve)
  Using cached Jinja2-2.8-py2.py3-none-any.whl
…
Successfully installed Eve-0.6.4 cerberus-0.9.2 events-0.2.1 flask-0.10.1 flask-
pymongo-0.4.1 itsdangerous-0.24 jinja2-2.8 markupsafe-0.23 pymongo-3.3.0 simplejson-3.8.2
werkzeug-0.11.3
```

1.2 Basic setup

- Let’s create the base line for the next steps:

```
(pyday) Rauls-MacBook-Pro:~ rmarin$ cd pyday/
(pyday) Rauls-MacBook-Pro:pyday rmarin$ mkdir myapi
(pyday) Rauls-MacBook-Pro:pyday rmarin$ cd myapi/
(pyday) Rauls-MacBook-Pro:myapi rmarin$ vi app.py

from eve import Eve
app = Eve()

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

(pyday) Rauls-MacBook-Pro:myapi rmarin$ vi settings.py

DOMAIN = {'people': {}}
```

1.3 RESTful WS test out

- We're ready to go:

```
(pyday) Rauls-MacBook-Pro:myapi rmarin$ python app.py
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

- To test out our API will save us loads of hours and headaches:

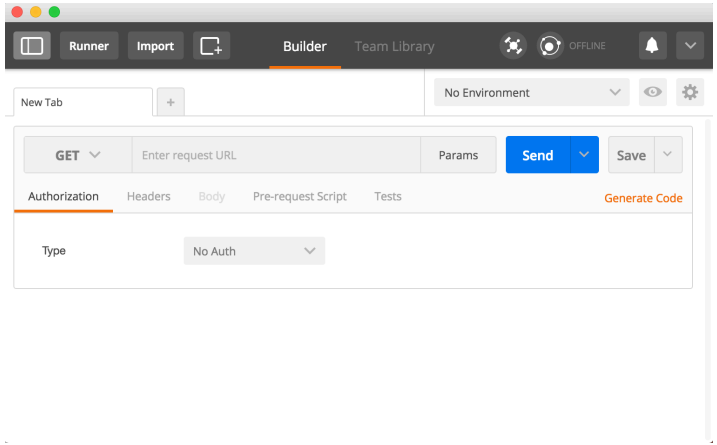
```
(pyday) Rauls-MacBook-Pro:myapi rmarin$  curl -i http://127.0.0.1:5000/
HTTP/1.0 200 OK
Content-Type: application/json
Content-Length: 60
Server: Eve/0.6.4 Werkzeug/0.11.3 Python/2.7.10
Date: Sat, 17 Sep 2016 07:26:54 GMT
```

1.3 RESTful WS test out (II)

- Python Eve (python-eve.org) is designed for human beings:

  let's test our RESTful API as human beings too

- Postman (www.getpostman.com) will help us in this matter:



Session's goals

1. Python Eve installation and setup
2. MongoDB basic installation (WDTCS data set)
3. Expanding all to something more real (CRUD, HA, …)

2. MongoDB basic installation

- We'll go over the following steps:

  1. Spin up a standalone MongoDB instance
  2. Restore the WDTCS data set

2.1 Spin up a MongoDB instance

- Download MongoDB 3.2.9 (.tgz/.zip) Community Server from:

  https://www.mongodb.com/download-center?jmp=nav#community

- Deploy it on your machine (ie. */opt/mongodb*):

```
(pyday) Rauls-MacBook-Pro:mongodb rmarin$ pwd
/opt/mongodb
(pyday) Rauls-MacBook-Pro:mongodb rmarin$ ls
GNU-AGPL-3.0            README                  bin
MPL-2                   THIRD-PARTY-NOTICES
```

2.1 Spin up a MongoDB instance (II)

- Create the folder structure for the MongoDB instance (ie. under the *pyday* folder):

```
(pyday) Rauls-MacBook-Pro:pyday rmarin$ mkdir -p rs/node1/data
(pyday) Rauls-MacBook-Pro:pyday rmarin$ cd rs/node1/
```

2.1 Spin up a MongoDB instance (III)

- Create the following configuration file under the previous folder:

```
(pyday) Rauls-MacBook-Pro:node1 rmarin$ vi mongod.conf
net:
  port: 27017
processManagement:
  fork: "true"
storage:
  dbPath: /Users/rmarin/pyday/rs/node1/data
  engine: wiredTiger
  journal:
    enabled: true
  wiredTiger:
    engineConfig:
      cacheSizeGB: 4
systemLog:
  destination: file
  logAppend: true
  path: /Users/rmarin/pyday/rs/node1/mongod.log
```

2.1 Spin up a MongoDB instance (IV)

- Start the MongoDB instance up:

```
(pyday) Rauls-MacBook-Pro:node1 rmarin$ /opt/mongodb/bin/mongod -f mongod.conf
about to fork child process, waiting until server is ready for connections.
forked process: 25240
child process started successfully, parent exiting
(pyday) Rauls-MacBook-Pro:node1 rmarin$ ls
data            mongod.conf     mongod.log
(pyday) Rauls-MacBook-Pro:node1 rmarin$ ls data
WiredTiger                              WiredTigerLAS.wt
diagnostic.data                         mongod.lock
WiredTiger.lock                         _mdb_catalog.wt
index-1--3316756531886542336.wt         sizeStorer.wt
WiredTiger.turtle                       collection-0--3316756531886542336.wt
index-3--3316756531886542336.wt         storage.bson
WiredTiger.wt                           collection-2--3316756531886542336.wt    journal
```

2.2 Restore the WDTCS data set

- Download the data set from the WDTCS's repo:

```
(pyday) Rauls-MacBook-Pro:node1 rmarin$ wget https://github.com/raulmarinperez/wdtcs/raw/
master/data/dump%202.zip
--2016-09-16 23:34:30--  https://github.com/raulmarinperez/wdtcs/raw/master/data/dump
%202.zip
…
HTTP request sent, awaiting response... 200 OK
Length: 19449335 (19M) [application/octet-stream]
Saving to: 'dump 2.zip'

dump 2.zip             100%[===========================>]  18.55M  7.10MB/s   in 2.6s

2016-09-16 23:34:36 (7.10 MB/s) - 'dump 2.zip' saved [19449335/19449335]
```

2.2 Restore the WDTCS data set (I)

- Unzip the file containing the database dump:

```
(pyday) Rauls-MacBook-Pro:node1 rmarin$ unzip dump\ 2.zip
Archive:  dump 2.zip
   creating: dump/
   creating: dump/WDTCS/
  inflating: dump/WDTCS/containers.bson
  inflating: dump/WDTCS/containers.metadata.json
  inflating: dump/WDTCS/ports.bson
  inflating: dump/WDTCS/ports.metadata.json
  inflating: dump/WDTCS/ships.bson
  inflating: dump/WDTCS/ships.metadata.json
```

2.2 Restore the WDTCS data set (II)

- Restore the dump using the mongorestore tool:

```
(pyday) Rauls-MacBook-Pro:node1 rmarin$ mongorestore
2016-09-16T23:44:46.170+0200    using default 'dump' directory
2016-09-16T23:44:46.171+0200    preparing collections to restore from
…
2016-09-16T23:44:49.174+0200    [##########.............]  WDTCS.containers  70.7MB/164MB
(43.2%)
2016-09-16T23:44:52.174+0200    [#####################...]  WDTCS.containers  149MB/164MB
(90.7%)
2016-09-16T23:44:52.717+0200    [########################]  WDTCS.containers  164MB/164MB
(100.0%)
2016-09-16T23:44:52.717+0200    restoring indexes for collection WDTCS.containers from
metadata

2016-09-16T23:45:01.372+0200    finished restoring WDTCS.containers (911249 documents)
2016-09-16T23:45:01.372+0200    done
```

2.2 Restore the WDTCS data set (III)

- Double check that the data set is available:

```
(pyday) Rauls-MacBook-Pro:node1 rmarin$ mongo
MongoDB shell version v3.3.11
connecting to: 127.0.0.1:27017/test

MongoDB server version: 3.3.11
> show dbs
WDTCS  0.033GB
local  0.000GB
> use WDTCS
switched to db WDTCS
> show collections
containers
ports
ships
```

Session's goals

1. Python Eve installation and setup
2. MongoDB basic installation (WDTCS data set)
3. Expanding all to something more real (CRUD, HA, …)

3. Expanding all to something more real

- We'll go over the following steps:

1. A more real settings.py
2. Address CRUD via the new RESTful API
3. Enable redundancy and data availability

3.1 A more real settings.py

```
MONGO_HOST = 'localhost'
MONGO_PORT = 27017
MONGO_DBNAME = 'WDTCS’

ports_schema = {
  '_id': {'type':'objectid'},
  'Ranking': {'type':'integer'},
  'Name': {'type':'string'},
  'Country': {'type':'string'}
}
ports = {
  'item_title': 'port',
  'schema': ports_schema
}
DOMAIN = {
  'ports': ports
}
```

3.2 Addressing CRUD via the new RESTful API

- List all ports (Read)

  http://127.0.0.1:5000/ports (GET)


```
{ "_updated": "Thu, 01 Jan 1970 00:00:00 GMT",
  "Ranking": 3,
  "Name": "Shenzhen",
  "Country": "China",
  "_links": { "self": { "href": "ports/56fda4e90a162d0f051f2ce7",
                        "title": "port” } },
  "_created": "Thu, 01 Jan 1970 00:00:00 GMT",
  "_id": "56fda4e90a162d0f051f2ce7",
  "_etag": "2ebc7ab98b18a263b7babe46b9de2eab0751f980"
}
```

3.2 Addressing CRUD via the new RESTful API (I)

- List Barcelona's port (Read):

  http://127.0.0.1:5000/ports/56fda4e90a162d0f051f2d33

- List Spanish ports (Read):

  http://127.0.0.1:5000/ports?where={"Country": "Spain"}

  We only consider Algeciras (30), Valencia (31) and Barcelona (77) so far

3.2 Addressing CRUD via the new RESTful API (II)

- Smart and elegant design for data integrity and concurrency control:

http://python-eve.org/features.html#data-integrity-and-concurrency-control

"Consumers are not allowed to edit (PATCH or PUT) or delete (DELETE) a resource unless they provide an up-to-date ETag for the resource they are attempting to edit. This prevents overwriting items with obsolete versions"

3.2 Addressing CRUD via the new RESTful API (III)

- Wouldn't be nice if Barcelona's port's ranking were 1? (Update):

  URL: http://127.0.0.1:5000/ports/56fda4e90a162d0f051f2d33 (POST)
  Content-type: *Body➔Raw➔JSON(application/json)*
  Content: *{ "Ranking": 1 }*

- This query rises an error:

```
{ "_status": "ERR",
  "_error": {
    "message": "The method is not allowed for the requested URL.",
    "code": 405
  } }
```

3.2 Addressing CRUD via the new RESTful API (IV)

- By default Python Eve provides a read-only RESTful API
- Enable writes by adding the following entry in the settings.py file:

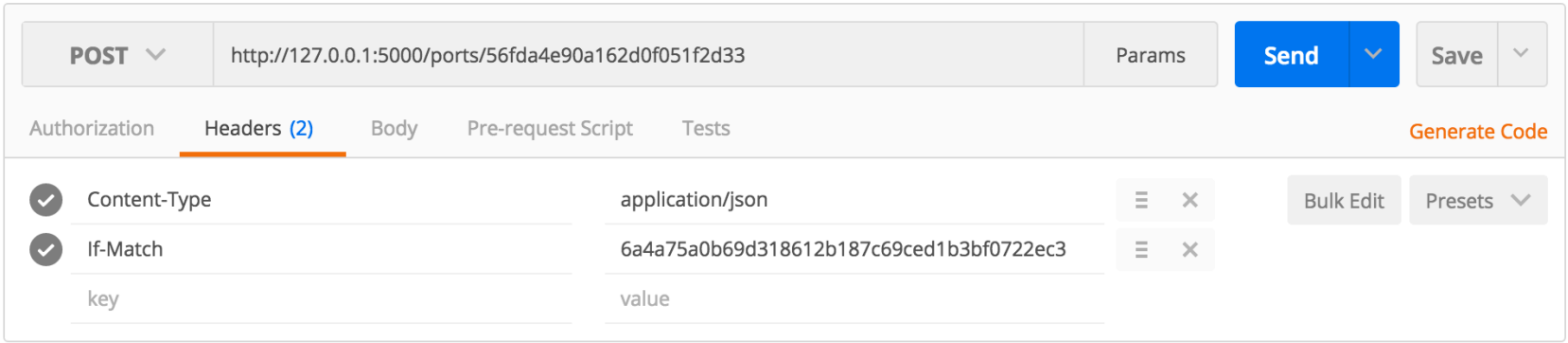
```
RESOURCE_METHODS = ['GET', 'POST', 'DELETE']
ITEM_METHODS = ['GET','PATCH','PUT','DELETE']
```

- Still rises an error:

```
{ "_status": "ERR",
  "_error": {
    "message": "An etag must be provided to edit a document",
    "code": 403
  } }
```

3.2 Addressing CRUD via the new RESTful API (V)

- You need to provide the _etag as a header entry (If-Match):



- Double check that Barcelona's port is now the best port in Spain:

http://127.0.0.1:5000/ports?where={"Country": "Spain"}&sort=[("Ranking", 1)]

3.2 Addressing CRUD via the new RESTful API (VI)

- Hey! I live in Málaga and I want Málaga's port to be in.
- Let's add a new port within the ports collection (Create):

  URL: http://127.0.0.1:5000/ports (POST)
  Content-type: *Body➔Raw➔JSON(application/json)*
  Content: `{ "Ranking": 100, "Name": "Malaga", "Country": "Spain" }`
- Let's do it for my partner in crime living in Madrid too (sounds weird though):

  URL: http://127.0.0.1:5000/ports (POST)
  Content-type: *Body➔Raw➔JSON(application/json)*
  Content: `{ "Ranking": 101, "Name": "Madrid", "Country": "Spain" }`

3.2 Addressing CRUD via the new RESTful API (VII)

- Wasn't a good idea to add "Madrid's port"
- Would be weird to see ships across rivers reaching out "El Manzanares"
- Let's remove the port from the ports collection (Delete):

URL: http://127.0.0.1:5000/ports/57dd0e7ab04cc5656c1ea0cf (DELETE)
Header: `If-Match: 383e5ae9ca3e52ccd2c76b9c4dcd79a7afecb78e`

3.3 Enabling redundancy and data availability

- So far so good! But what happens if MongoDB goes down (stop it)?

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
<title>500 Internal Server Error</title>
<h1>Internal Server Error</h1>
<p>The server encountered an internal error and was unable to complete your request.  Either
the server is overloaded or there is an error in the application.</p>
```

- We need to go a step further ➔ to enable a MongoDB Replica Set
- Amend the following lines to the configuration file:

```
replication:
  oplogSizeMB: 100
  replSetName: wdtcs
```

3.3 Enabling redundancy and data availability (I)

- Let's expand the folders' structure to two additional nodes:

```
(pyday) Rauls-MacBook-Pro:pyday rmarin$ mkdir -p rs/node2/data
(pyday) Rauls-MacBook-Pro:pyday rmarin$ mkdir -p rs/node3/data
```

- Clone node1's configuration file for each new instance and adapt it:

```
net:
  port: 27018 (27019)
storage:
  dbPath: /Users/rmarin/pyday/rs/node2/data (node3)
  …
systemLog:
  path: /Users/rmarin/pyday/rs/node2/mongod.log (node3)
  …
```

3.3 Enabling redundancy and data availability (II)

- Start up all the MongoDB instances
- Connect to node1 and initiate the replica set:

```
(pyday) Rauls-MacBook-Pro:node1 rmarin$ mongo
MongoDB shell version v3.3.11
connecting to: 127.0.0.1:27017/test
MongoDB server version: 3.3.11
> rs.initiate()
{
        "info2" : "no configuration specified. Using a default configuration for the set",
        "me" : "Rauls-MacBook-Pro.local:27017",
        "ok" : 1
}
wdtcs:PRIMARY> rs.add("Rauls-MacBook-Pro.local:27018")
{ "ok" : 1 }
wdtcs:PRIMARY> rs.add("Rauls-MacBook-Pro.local:27019")
{ "ok" : 1 }
```

3.3 Enabling redundancy and data availability (III)

- The RESTful API should work again ➔ list all ports:

  http://127.0.0.1:5000/ports

- Take node1 down and try again ➔ there's still a primary but it's failing
- Our API is still bound to node1 ➔ change the following in settings.py:

```
MONGO_URI = 'mongodb://Rauls-MacBook-Pro.local:27017,Rauls-MacBook-Pro.local:27018,Rauls-MacBook-Pro.local:27019/WDTCS?replicaSet=wdtcs'
#MONGO_HOST = 'localhost'
#MONGO_PORT = 27017
#MONGO_DBNAME = 'WDTCS'
```

3.3 Enabling redundancy and data availability (IV)

- Take node2 down and try again ➔ there's still a secondary but it's failing
- By default pymongo only reads from primaries ➔ update the URI to:

```
MONGO_URI = 'mongodb://Rauls-MacBook-Pro.local:27017,Rauls-MacBook-Pro.local:27018,Rauls-
MacBook-Pro.local:27019/WDTCS?replicaSet=wdtcs&readPreference=primaryPreferred'
```

Questions?

Thank You!

```
{
  name        : "Rubén Terceño",
  title       : "Senior Solutions Architect",
  mail        : "ruben@mongodb.com",
  location    : "Madrid, ES”
}

{
  name        : "Raúl Marín",
  title       : "Senior Consulting Engineer",
  mail        : "raul.marin-perez@mongodb.com",
  location    : ”Málaga, ES”
}
```

mongoDB