How to build a RESTFUL Server using python – flask

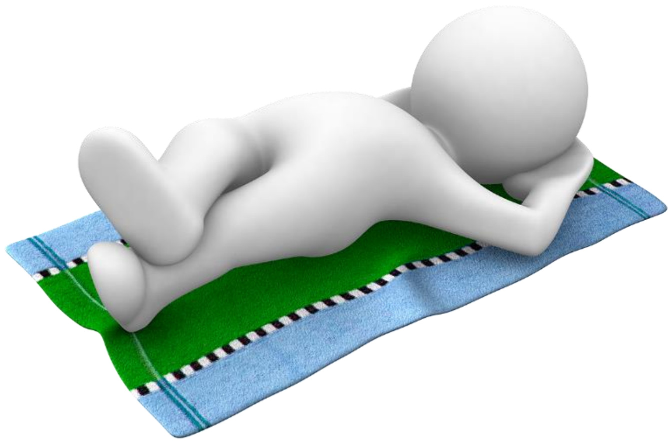
Klaas Brant

KBCE b.v.

Session code: E04

Monday 2nd October 2017 16:30 hrs

Cross Platform [Server on Linux/Cloud]



How to build a RESTFUL Server
using python – flask

Klaas Brant – KBCE b.v.

KBCE
Klaas Brant Consulting & Education

P.O. Box 200, 5520 AE Eersel, The Netherlands
Tel.:(+31) 497-530190 E-mail: kbrant@kbce.com

Disclaimer / Trademarks / Abstract / Recognition

Abstract: The RESTFUL interface has become an important interface between many applications and your corporate data. It has become much more than a simple CRUD operation on tables. In this session, we will review what is REST, what problems (e.g. security) we need to deal with and how to solve this. And we will look into building a production ready, scalable rest server using python flask.

This presentation contains some pictures from Michał Karzyński's presentation at EuroPython 2016 – He inspired our design!

Agenda

- What is REST?
- Let's design it
- Demo
- Security
- Deployment

Before we begin

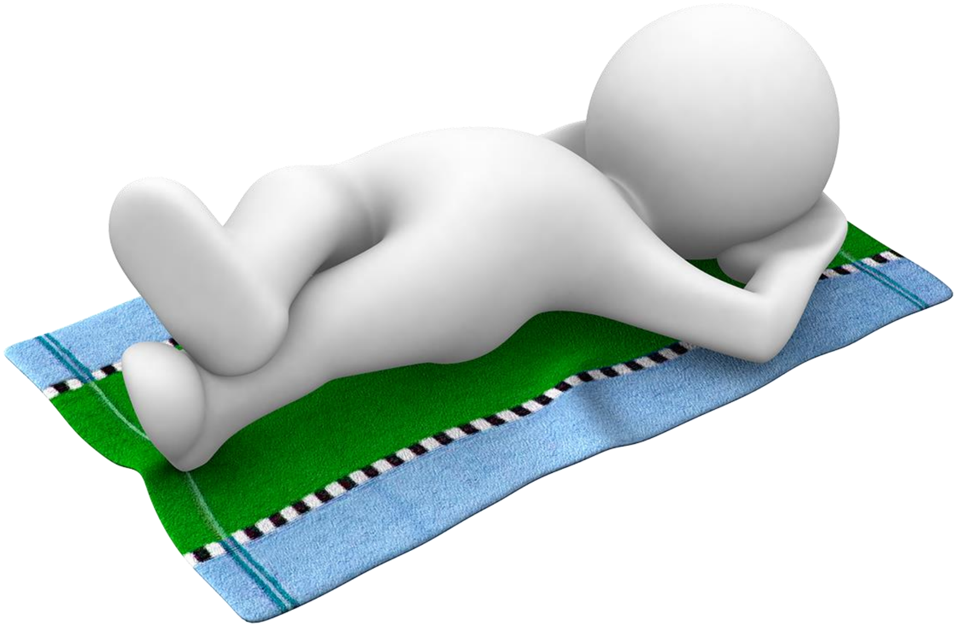
- This presentation applies to ANY data source
  - Including Db2 for LUW and Db2 for z/OS and many others
- Presentation covers basics, there are so much more
  - e.g. testing, optimize ORM performance, Unicode handling…
- Always make sure you comply with the audit rules for your data
  - You are exposing data to the world: Build a (very) secure interface!
    - Use secure SSL connections
    - Make 100% sure all debug options are off in production

I need REST interface. It should be scalable with minimal cost. You are not allowed to add to the complexity of the host (z/OS). Keep CPU cycles on the host to minimum. And yes, I need it quickly and make it state of the art (swagger)

Swagger?

- Swagger is a specification and complete framework implementation for **describing, producing, consuming, and visualizing** RESTful web services
- Has become a de facto standard to document REST
- See https://swagger.io

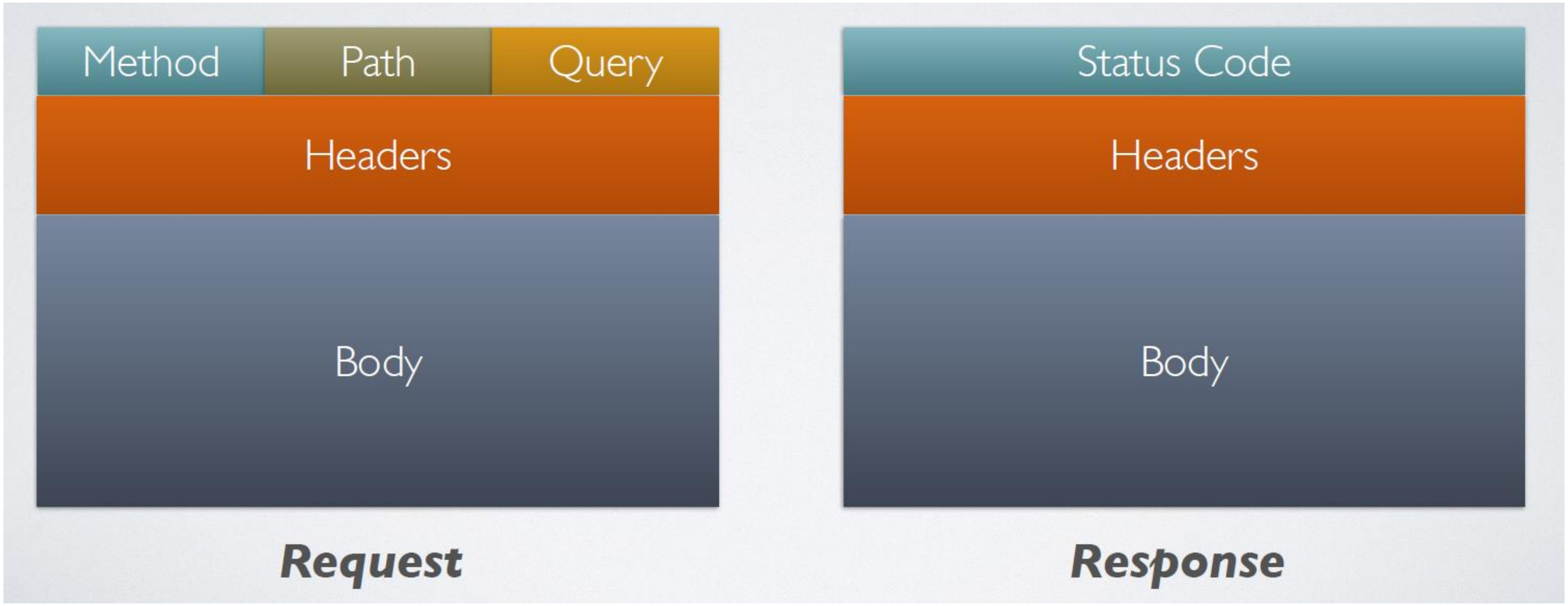
What is REST?



What is REST

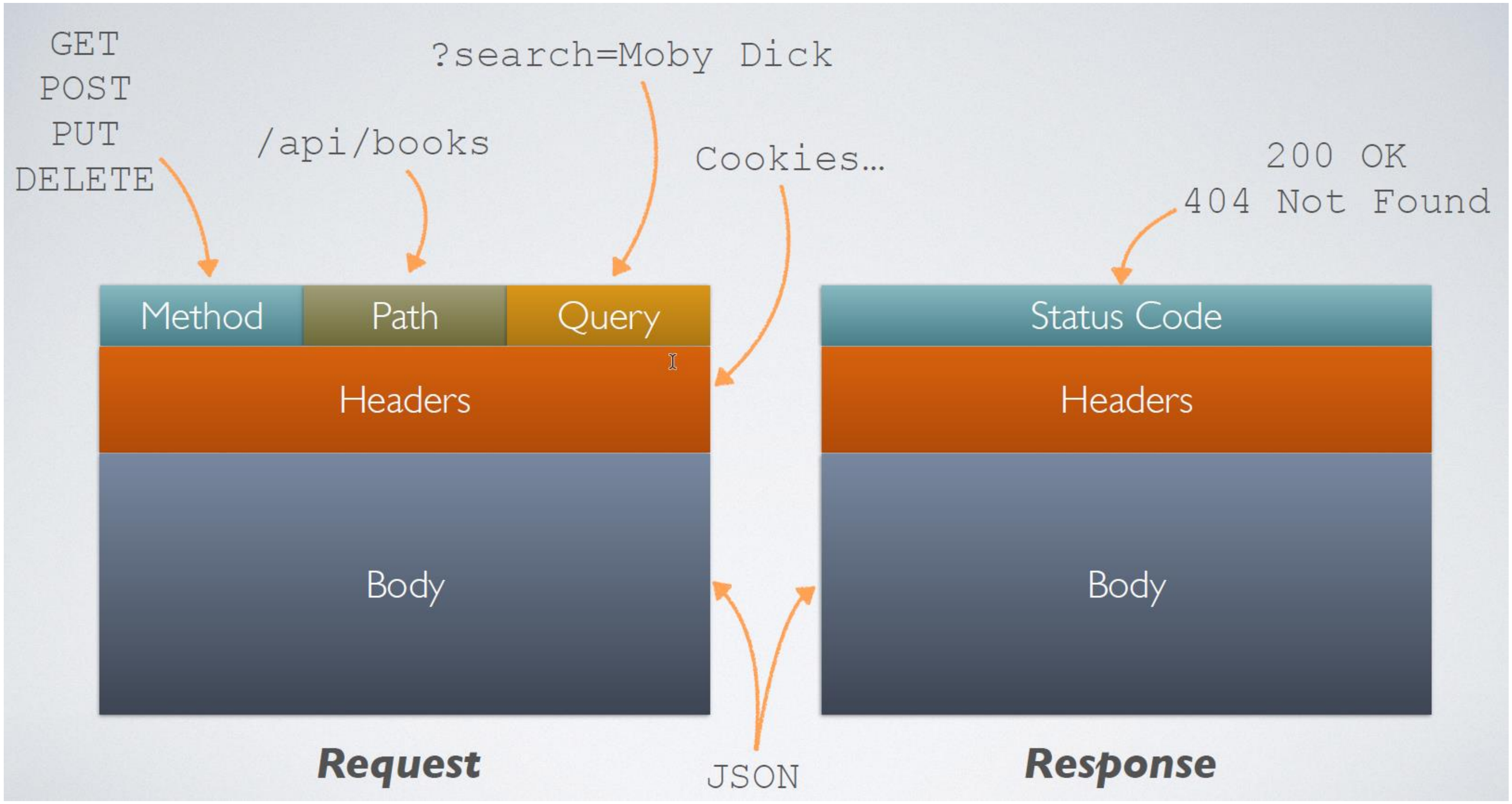
- Stands For:
  - **<u>RE</u>**presentational **<u>S</u>**tate **<u>T</u>**ransfer
- Architecture based on **HTTP** - a web standard
  - Request and Response
  - Stateless
  - Cacheable
- REST is not a standard! Really? Yes!
  - Simplification of SOAP
  - Using **JSON**, not XML, for data exchange
  - Very popular in distributed systems (mobile apps!)
- REST can lightweight, easy maintainable, and very scalable

What is http



- Method is often called **VERB**
- Path is often called url, uri or **ENDPOINT**
- Body is often called **PAYLOAD**

A http example



REST Convention

	GET	PUT	POST	DELETE
api/sample/employee	list all		new	
api/sample/employee/<#>	list single	update		delete

Correct: DELETE / URI: http://myhost/agent/007
Delete agent with id 007

Incorrect: GET http://myhost/agent/007?action=delete

Using REST

- An application sends a request using a **URI**
  - Often called "endpoint" in REST
  - Endpoint same for all actions against the same resource
  - URI has optional parameters to identify a single object (key)

- The **VERB** indicates the "what"
  - GET = retrieve data
  - PUT = update data
  - POST = create data
  - DELETE = delete data

VERB: DELETE / URI: http://myhost/agent/007
Delete agent with id 007

NOT: GET http://myhost/agent/007?action=delete

Host response

- Responses come back as an HTTP Status Code + optional data

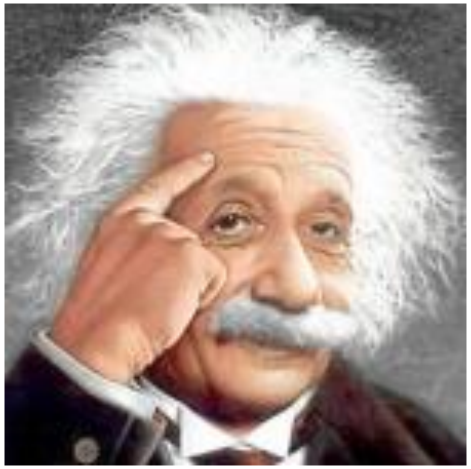
- HTTP Status Codes
  - 1XX – Request Received and I’m processing it
  - 2XX – Request Received and processed it successfully!
  - 3XX – APP must do something else to complete the request
  - 4XX – APP made an error
  - 5XX – HOST cannot handle the request

- Be careful: REST is NOT a standard!
  - Different implementations might use different codes to express the same thing…
  - Lots of discussion about the 400 series.
  - Make sure you document things!

Let's design it!

python

Flask
web development,
one drop at a time



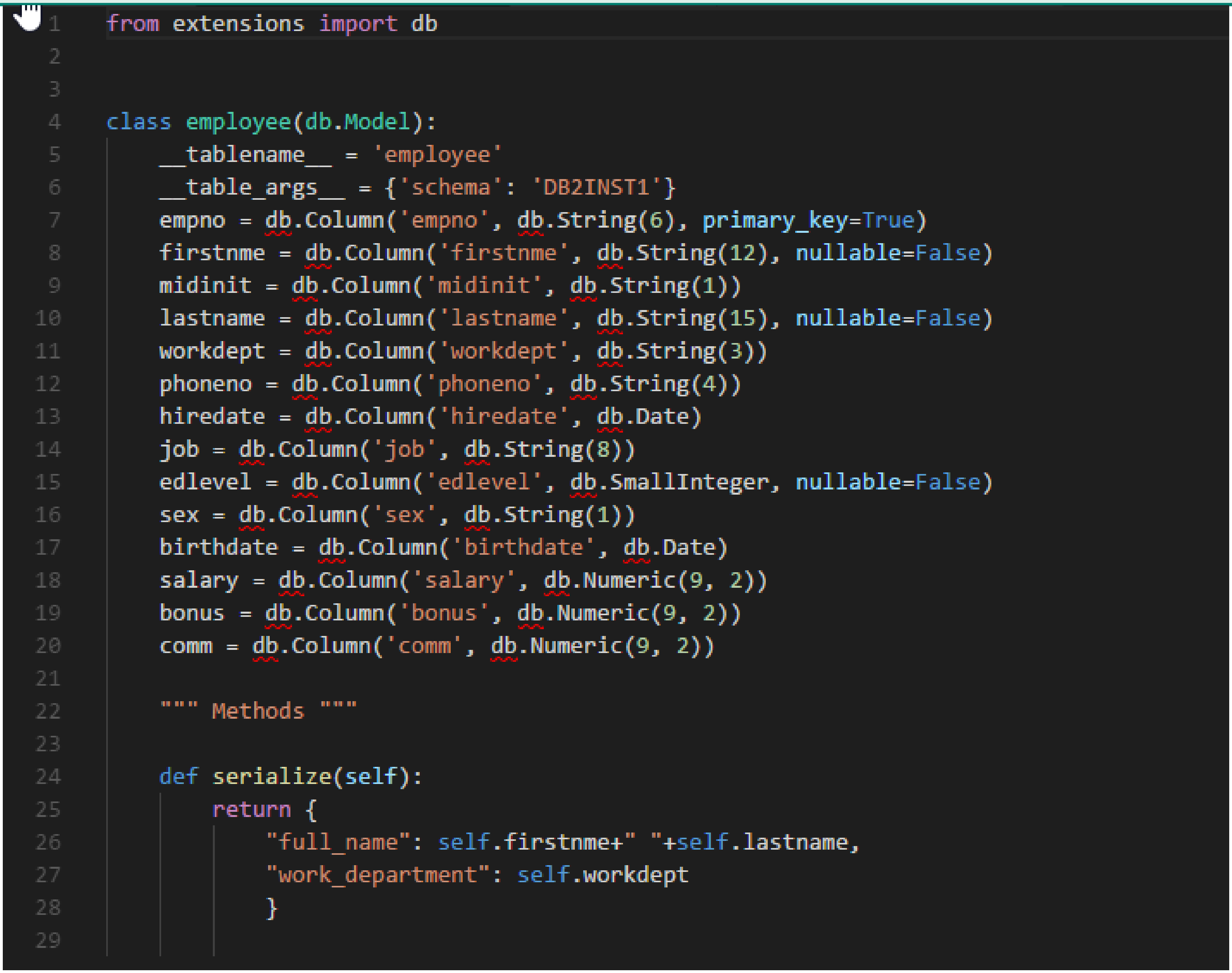
Choices to be made…

- Why Python? The #3 language in the world!
  - Simple, fast, low overhead, scales very well
  - Excellent libraries!
  - Very straight forward with our implementation (Docker)
  - Easy to test and debug

- Why Flask?
  - Three choices: Django, Flask and Bottle
  - Flask is low overhead and very well designed micro framework

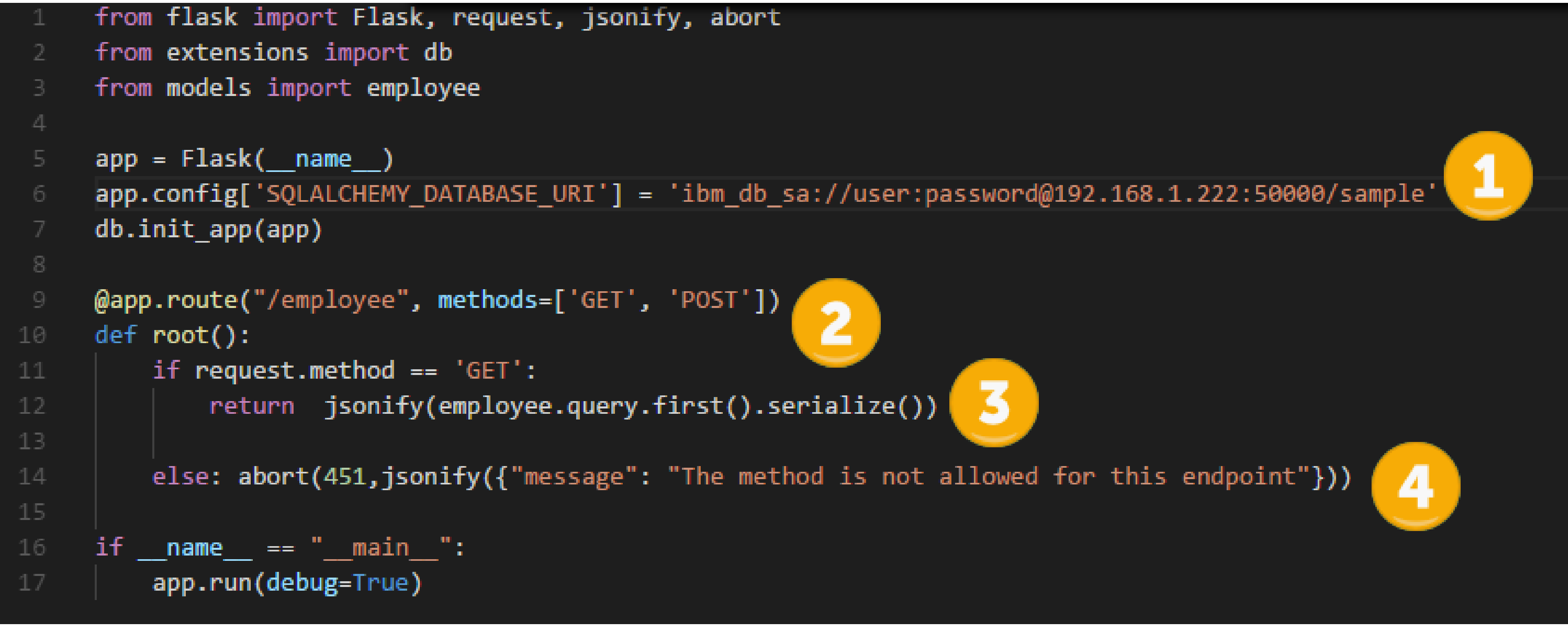
- To ORM or not to ORM, is that a question?
  - Yes, for the sake of DRY
  - #1 ORM (and only one that supports Db2) is SQLAlchemy
    - But… more later…

Database model



```python
from extensions import db


class employee(db.Model):
    __tablename__ = 'employee'
    __table_args__ = {'schema': 'DB2INST1'}
    empno = db.Column('empno', db.String(6), primary_key=True)
    firstnme = db.Column('firstnme', db.String(12), nullable=False)
    midinit = db.Column('midinit', db.String(1))
    lastname = db.Column('lastname', db.String(15), nullable=False)
    workdept = db.Column('workdept', db.String(3))
    phoneno = db.Column('phoneno', db.String(4))
    hiredate = db.Column('hiredate', db.Date)
    job = db.Column('job', db.String(8))
    edlevel = db.Column('edlevel', db.SmallInteger, nullable=False)
    sex = db.Column('sex', db.String(1))
    birthdate = db.Column('birthdate', db.Date)
    salary = db.Column('salary', db.Numeric(9, 2))
    bonus = db.Column('bonus', db.Numeric(9, 2))
    comm = db.Column('comm', db.Numeric(9, 2))

    """ Methods """

    def serialize(self):
        return {
            "full_name": self.firstnme+" "+self.lastname,
            "work_department": self.workdept
            }

```

Code



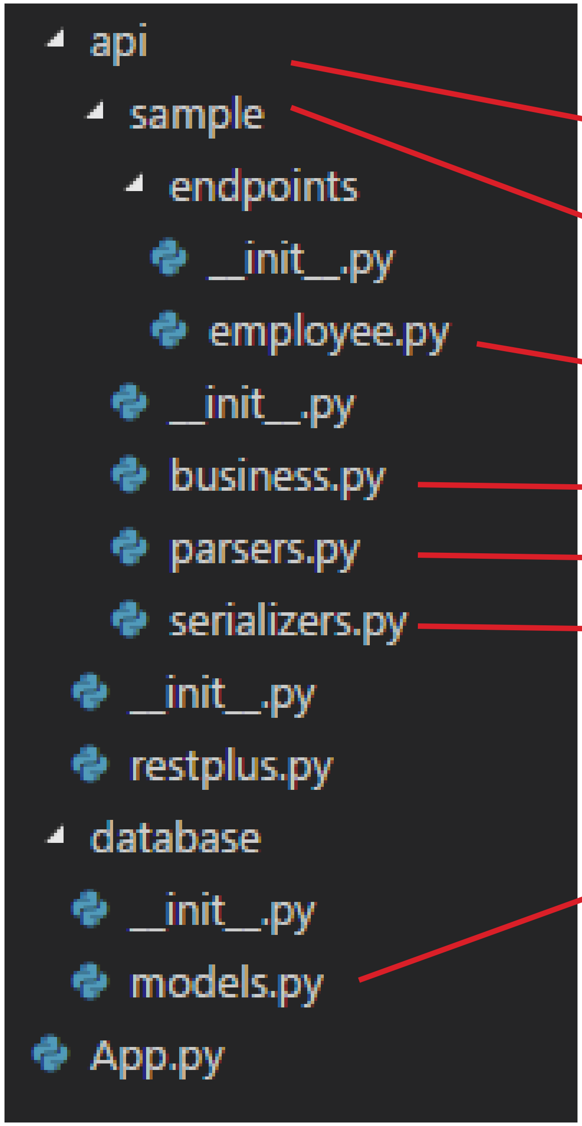
```
from flask import Flask, request, jsonify, abort
from extensions import db
from models import employee

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'ibm_db_sa://user:password@192.168.1.222:50000/sample'
db.init_app(app)

@app.route("/employee", methods=['GET', 'POST'])
def root():
    if request.method == 'GET':
        return  jsonify(employee.query.first().serialize())

    else: abort(451,jsonify({"message": "The method is not allowed for this endpoint"}))

if __name__ == "__main__":
    app.run(debug=True)
```

1. URI for Alchemy – ibm_db_sa is interface by IBM
2. The VERBs allowed and checking
3. The Payload using jsonify
4. Error handling…

Flask-RESTPlus to the rescue

- Define and document endpoints
- Validate input
- Format output (as JSON)
- Intercepts Python / Database exceptions
  - Output is correct JSON response
- Very simple
- Generate interactive Swagger documentation
  - Instant testing facilities
- Generate Swagger JSON
  - Used by generator tools for boilerplate client

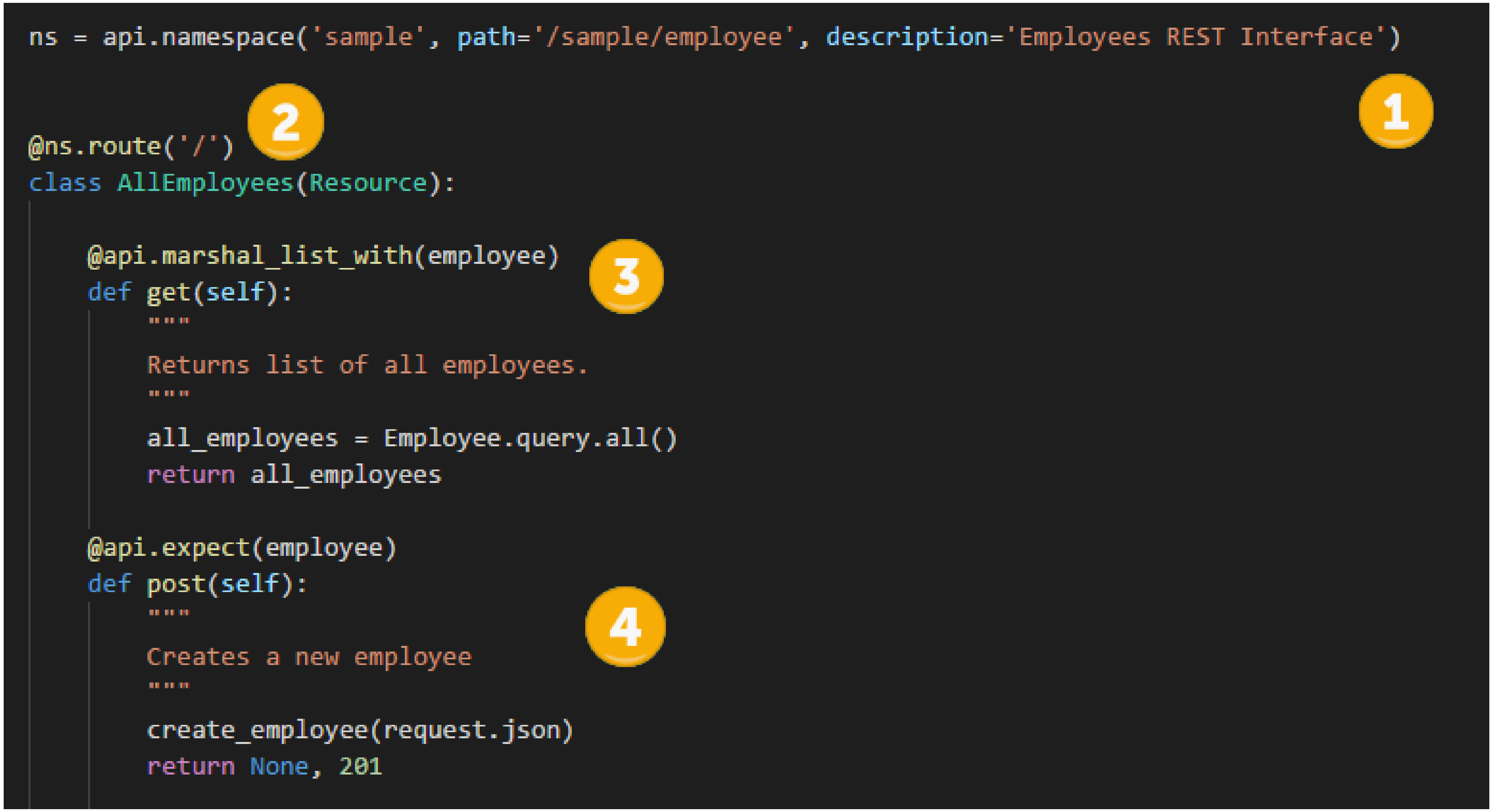
How we setup the application



```
api
  sample
    endpoints
      __init__.py
      employee.py
    __init__.py
    business.py
    parsers.py
    serializers.py
  __init__.py
  restplus.py
database
  __init__.py
  models.py
App.py
```

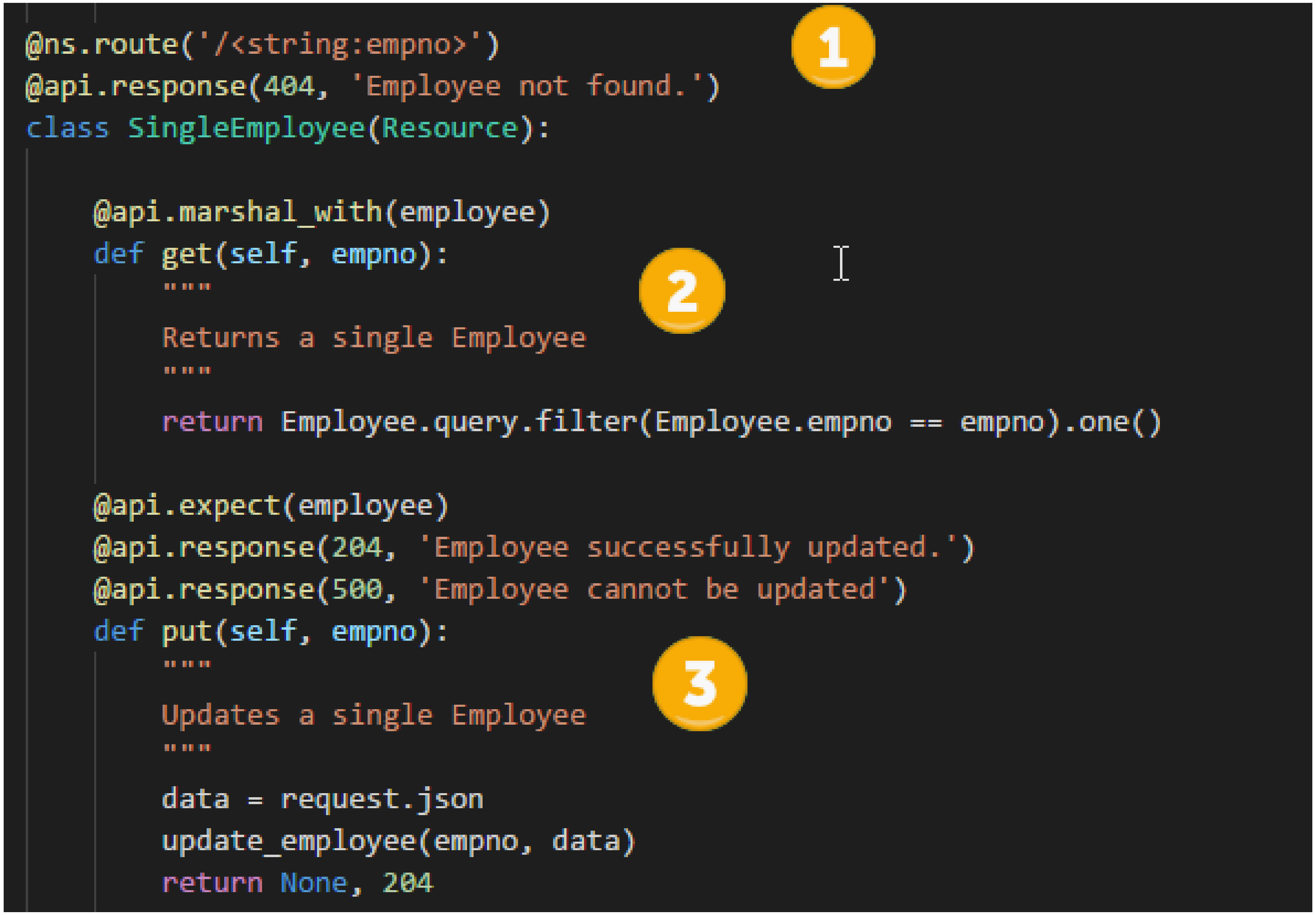
1. Main directory
2. Namespace directory
   - Endpoints
   - Business logic
   - Parsers
   - Serializers
3. Database directory

Endpoint /



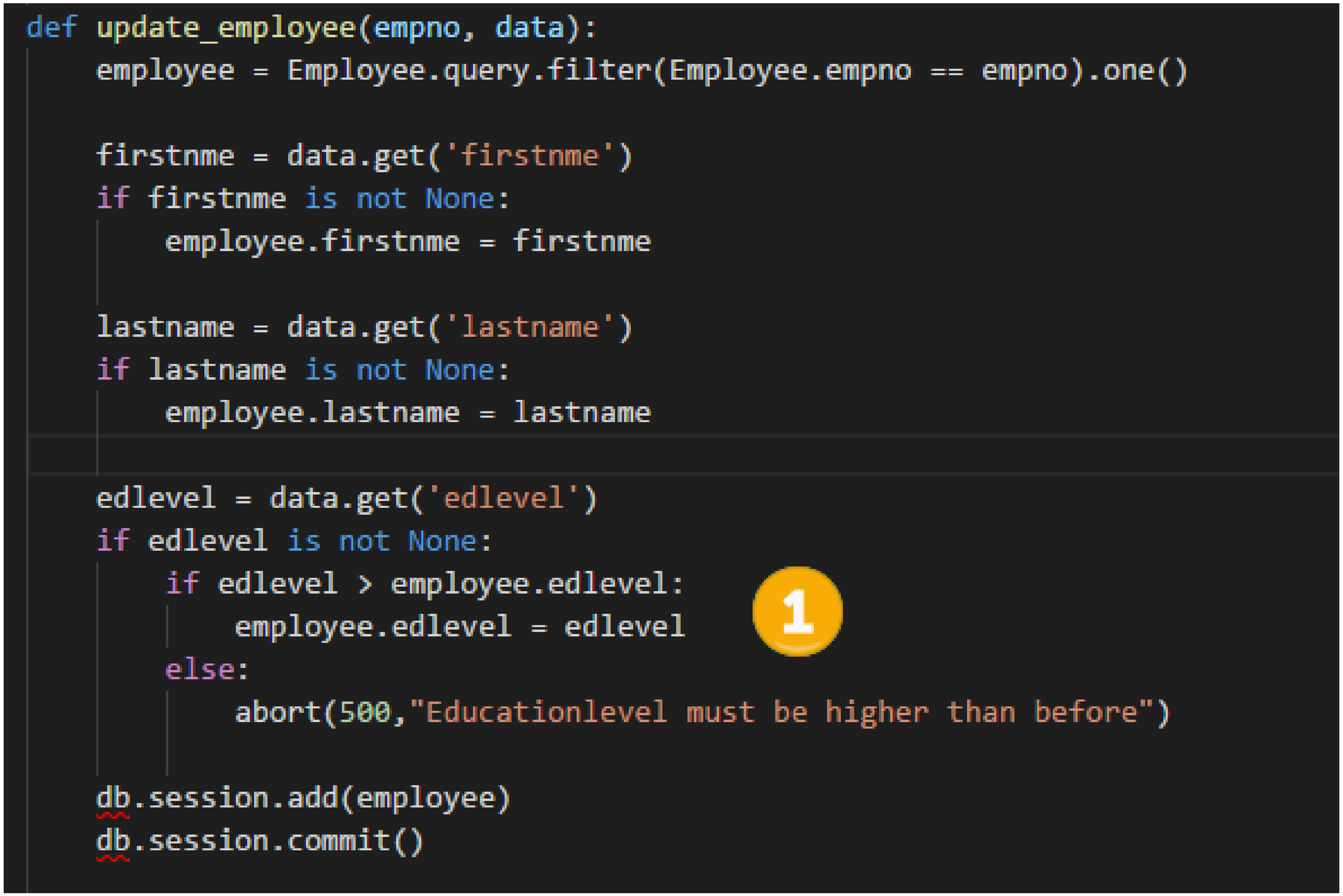
```
ns = api.namespace('sample', path='/sample/employee', description='Employees REST Interface')


@ns.route('/')
class AllEmployees(Resource):

    @api.marshal_list_with(employee)
    def get(self):
        """
        Returns list of all employees.
        """
        all_employees = Employee.query.all()
        return all_employees

    @api.expect(employee)
    def post(self):
        """
        Creates a new employee
        """
        create_employee(request.json)
        return None, 201
```

Endpoint /<empno>



```
@ns.route('/<string:empno>')
@api.response(404, 'Employee not found.')
class SingleEmployee(Resource):

    @api.marshal_with(employee)
    def get(self, empno):
        """
        Returns a single Employee
        """
        return Employee.query.filter(Employee.empno == empno).one()

    @api.expect(employee)
    @api.response(204, 'Employee successfully updated.')
    @api.response(500, 'Employee cannot be updated')
    def put(self, empno):
        """
        Updates a single Employee
        """
        data = request.json
        update_employee(empno, data)
        return None, 204
```

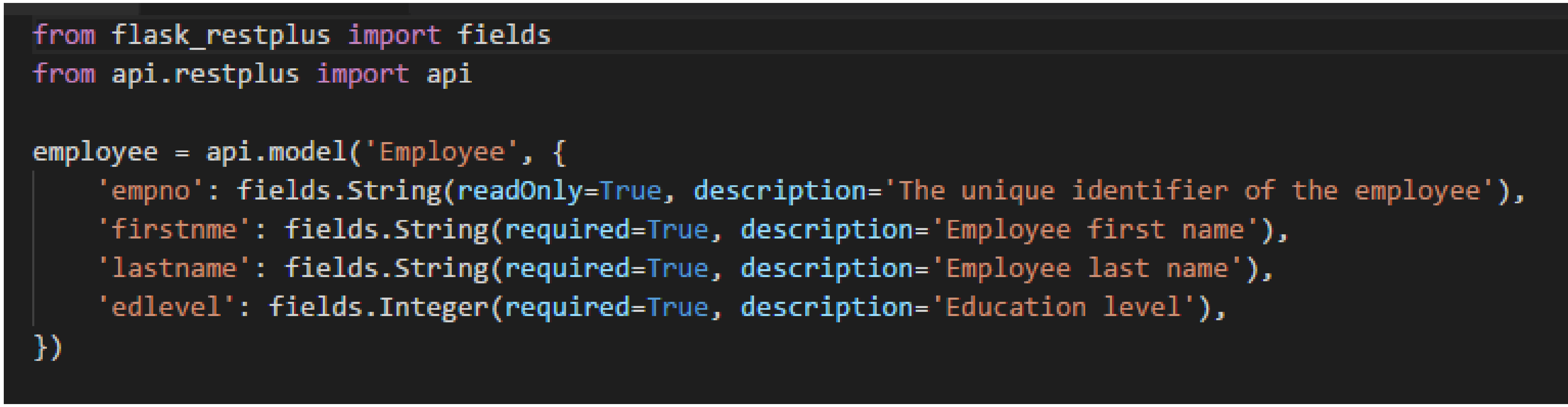
A business processor



```
def update_employee(empno, data):
    employee = Employee.query.filter(Employee.empno == empno).one()

    firstnme = data.get('firstnme')
    if firstnme is not None:
        employee.firstnme = firstnme

    lastname = data.get('lastname')
    if lastname is not None:
        employee.lastname = lastname

    edlevel = data.get('edlevel')
    if edlevel is not None:
        if edlevel > employee.edlevel:
            employee.edlevel = edlevel
        else:
            abort(500,"Educationlevel must be higher than before")

    db.session.add(employee)
    db.session.commit()
```

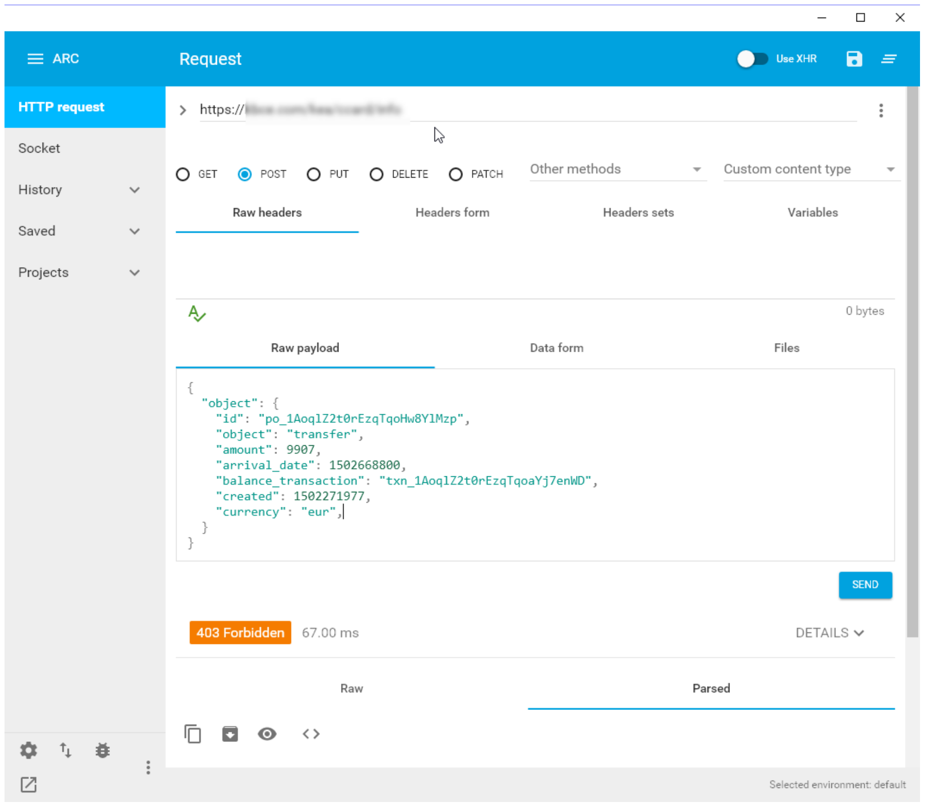
1

The serializer


```
from flask_restplus import fields
from api.restplus import api

employee = api.model('Employee', {
    'empno': fields.String(readOnly=True, description='The unique identifier of the employee'),
    'firstnme': fields.String(required=True, description='Employee first name'),
    'lastname': fields.String(required=True, description='Employee last name'),
    'edlevel': fields.Integer(required=True, description='Education level'),
})
```

- Used on input and output side
- Can validate
- Describe for Swagger

Tools we use for testing (**free** chrome apps)



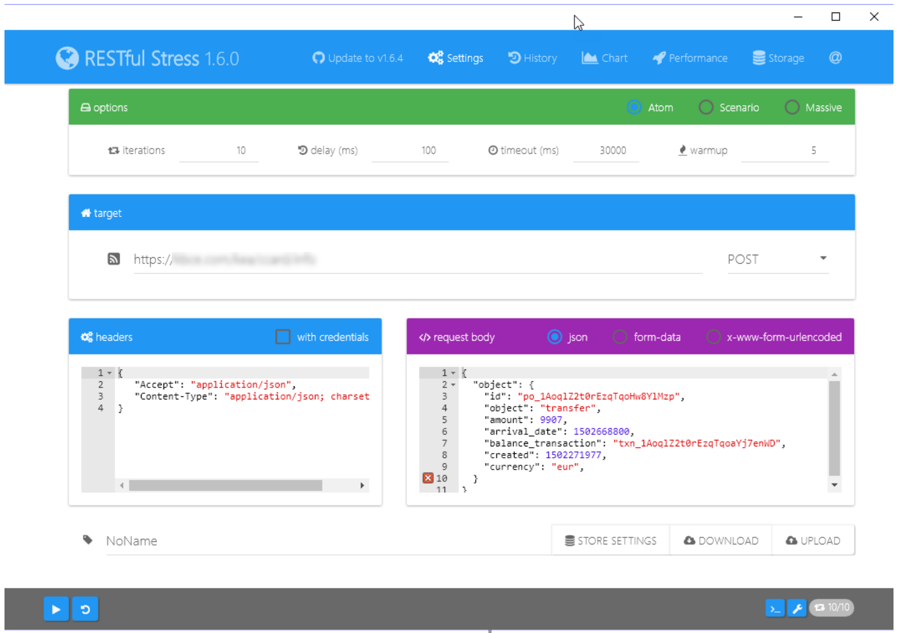
https://[redacted]

GET [✓ POST] PUT DELETE PATCH Other methods Custom content type

0 bytes

```
{
  "object": {
    "id": "po_1AoqlZ2t0rEzqTqoHw8YlMzp",
    "object": "transfer",
    "amount": 9907,
    "arrival_date": 1502668800,
    "balance_transaction": "txn_1AoqlZ2t0rEzqTqoaYj7enWD",
    "created": 1502271977,
    "currency": "eur",
  }
}
```

403 Forbidden 67.00 ms



[✓ Atom] Scenario Massive

iterations 10 delay (ms) 100 timeout (ms) 30000 warmup 5

https://[redacted] POST

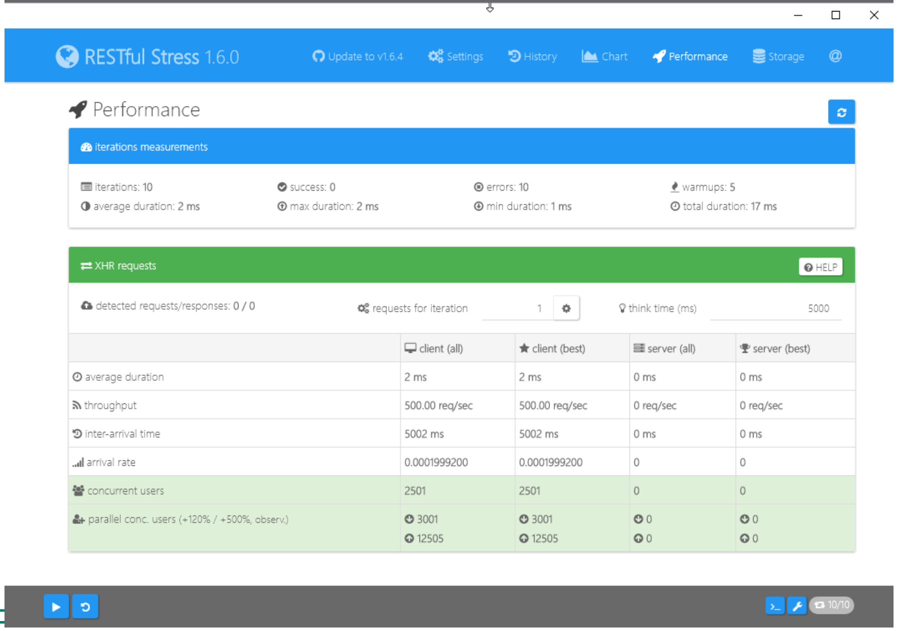
headers — with credentials

```
{
  "Accept": "application/json",
  "Content-Type": "application/json; charset
}
```

request body — [✓ json] form-data x-www-form-urlencoded

```
{
  "object": {
    "id": "po_1AoqlZ2t0rEzqTqoHw8YlMzp",
    "object": "transfer",
    "amount": 9907,
    "arrival_date": 1502668800,
    "balance_transaction": "txn_1AoqlZ2t0rEzqTqoaYj7enWD",
    "created": 1502271977,
    "currency": "eur",
  }
}
```

NoName



Performance

iterations measurements

iterations: 10 success: 0 errors: 10 warmups: 5
average duration: 2 ms max duration: 2 ms min duration: 1 ms total duration: 17 ms

XHR requests

detected requests/responses: 0 / 0 requests for iteration 1 think time (ms) 5000

	client (all)	client (best)	server (all)	server (best)
average duration	2 ms	2 ms	0 ms	0 ms
throughput	500.00 req/sec	500.00 req/sec	0 req/sec	0 req/sec
inter-arrival time	5002 ms	5002 ms	0 ms	0 ms
arrival rate	0.0001999200	0.0001999200	0	0
concurrent users	2501	2501	0	0
parallel conc. users (+120% / +500%, observ.)	3001 12505	3001 12505	0 0	0 0

ARC

and

RESTful Stress

Time for some action

Showstopper

ibm_db_sa
DOES NOT SUPPORT Db2 for z/OS!

- Write own extension to ibm_dbi module called DBPlus
- Business models now contain SQL ☺
- Upside:
  - our module is faster than SQLAlchemy
  - Better pooling of host threats
- Looking into making it open source

Security

Authentication / Authorization / Session

- All traffic is done on SSL (TLS 1.2) communication
- Mobile app sends data (account # / pin and more)
- z/OS stored proc authenticates
  - Generate UUID
  - Create session Scratch Pad Area (SPA) - must be on host!
  - Wrap UUID in JSON Web Token (JWT) using changing password
  - Send to client
- Almost every endpoint needs the JWT back to retrieve SPA
- Money transaction are protected by bank identifier
- Mobile app can clear session / expired sessions SPA is deleted

IS JWT SAFE?

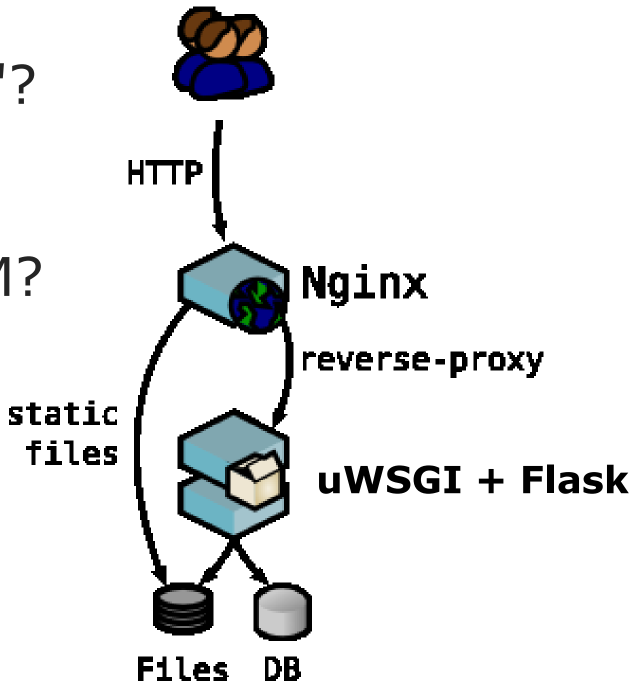
Deployment

uWSGI

docker

Problems threading

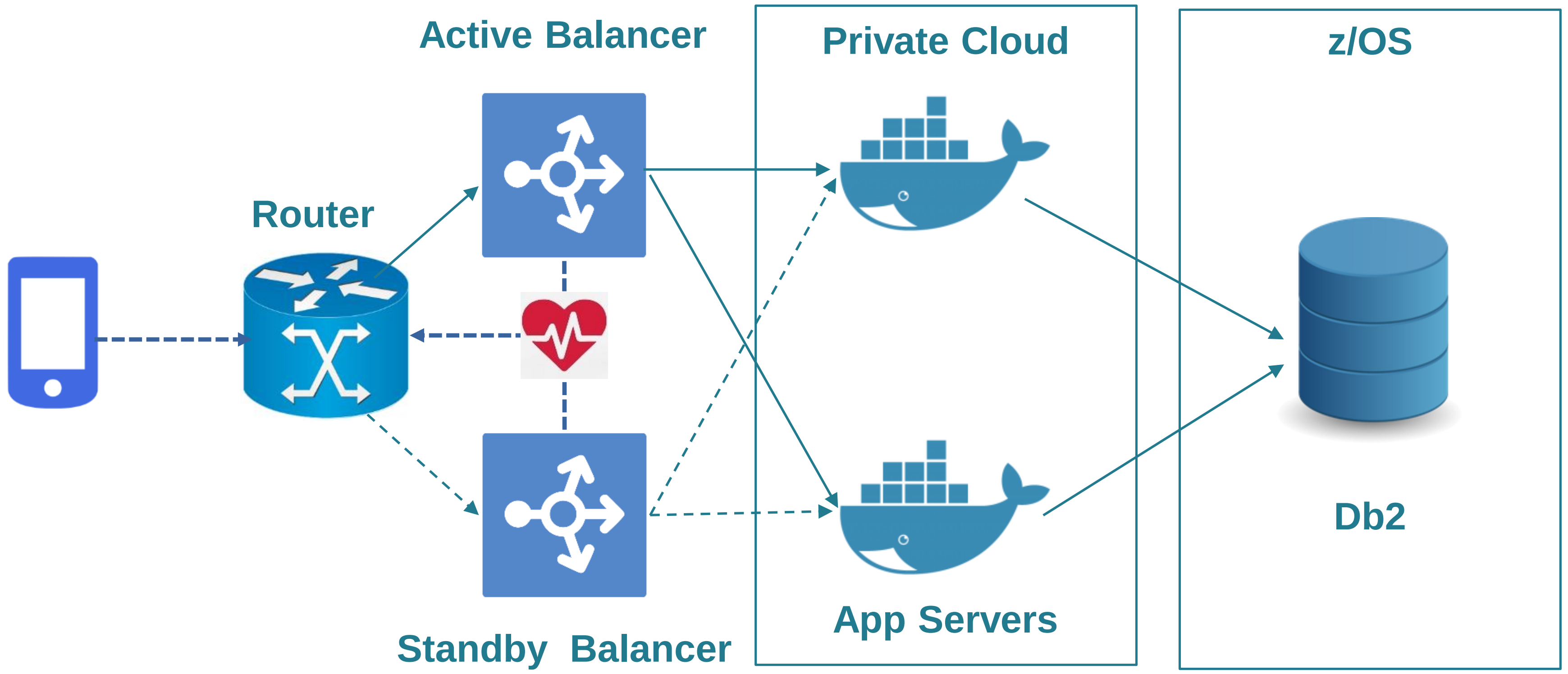
- Flask is single threaded
- Need an interface to multiply
  - Many to pick from, most popular uWSGI en Gunicorn
  - uWSGI is fast and scales well! n process, each has n threads
- Why not use a "webserver"?
- uWSGI limits? C10K? C10M?



Why Docker

- Resource measurement / setting limits
- Implement new version very quickly
- Fully tested contained environment
- Auto restart failed application
- Very user friendly, low resource, easy to learn / understand

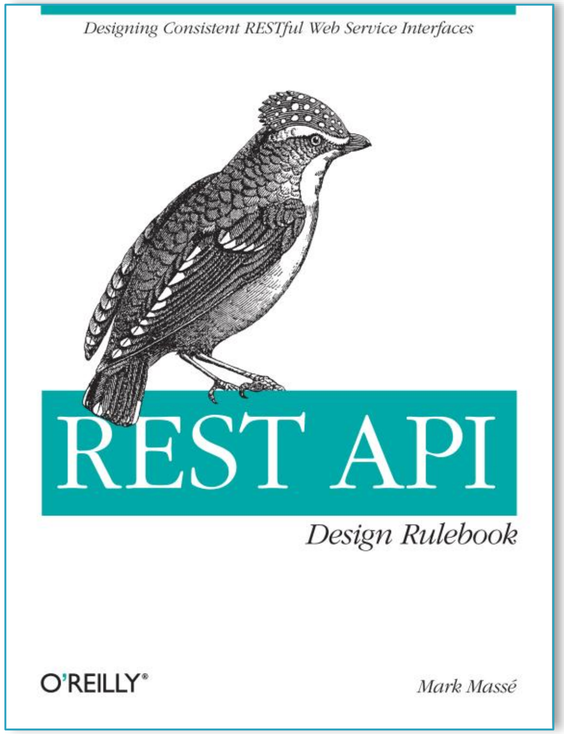
Load Balancer+



- **Router(s) / Load Balancers all hardware**
  - **No C10K / C10M problem**
  - **Allow up to 8 app servers**
  - **Auto failover using heart beat monitoring**

Good reference materials

- lynda.com has some good python training including restful!
- REST API Design Rulebook O'Reilly
- http://www.restapitutorial.com/
  - REST API Best Practices pdf!
- Michał Karzyński
  - http://michal.karzynski.pl/blog/2016/06/19/building-beautiful-restful-apis-using-flask-swagger-ui-flask-restplus/
  - http://michal.karzynski.pl/blog/2016/08/22/europython-2016-presentation/



QUESTIONS ?

Goodbye for now!

KBCE

Klaas Brant Consulting & Education

info@kbce.com

www.kbce.com

Klaas Brant
KBCE b.v.
info@kbce.com

Session code: **E04**

Please fill out your session evaluation before leaving!