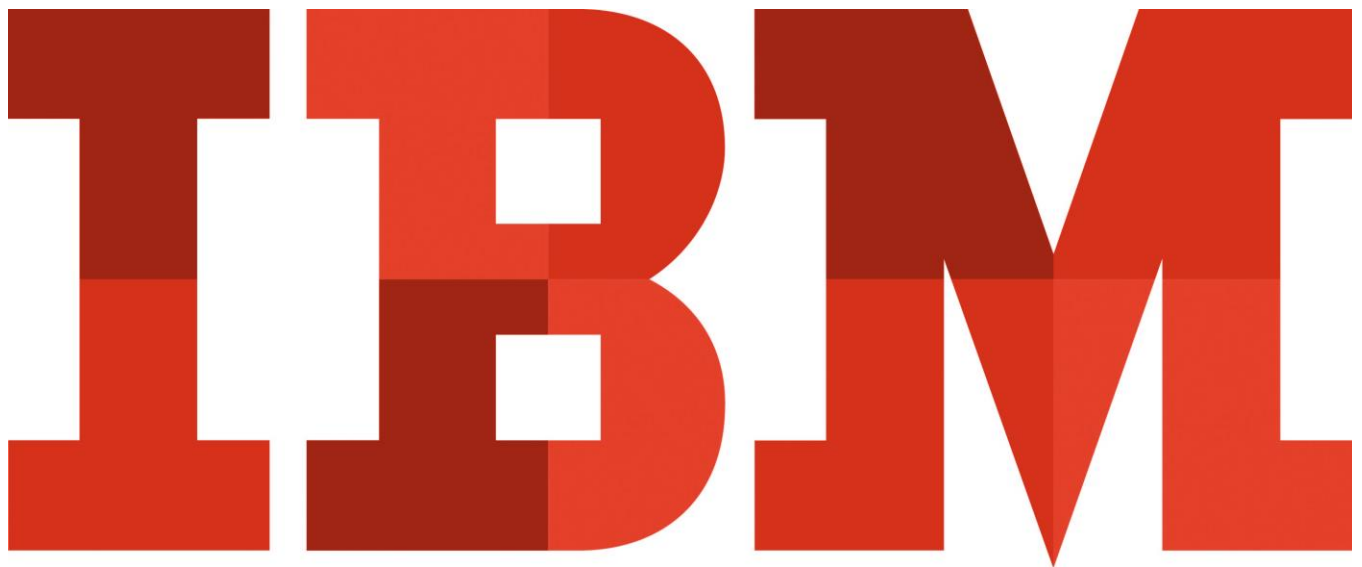


# Python and Cloudant on IBM Bluemix

A guide for hackathons

Lennart Frantzell [alf@us.ibm.com](mailto:alf@us.ibm.com)

V1. 2/9/2016



A digital copy of this lab and code snippets can be found at:  
<http://ibm.biz/Bdsw8e>

## Quick Introduction

---

When I attend Hackathons I am often asked if IBM Bluemix supports Python and how you access Cloudant from Python on the IBM Bluemix cloud. This little guide aims to provide a brief introduction to accessing Cloudant from Python on Bluemix. The assumption is that the person using this lab will have some knowledge of Python. Enjoy.



# Let's get started and set up our app and environment in Bluemix

## Pre-requisites

This app has been tested on the latest versions of Windows and the Mac.

1. On Windows we start by installing Python 3 <https://www.python.org/downloads/> Python comes as default on the Mac but we may want to check that we have the latest version.
  2. We then install the Python install utility pip <https://pip.pypa.io/en/stable/installing/>
  3. We then do a **pip install flask** and **pip install couchdb**
    - a. **Please note** that on the Mac we need to do `sudo pip install` in both cases
  4. Finally we install CouchDB for Windows or the Mac <http://couchdb.apache.org/> On the Mac we need to allow for installing apps from unidentified developers [https://support.apple.com/kb/PH25088?locale=en\\_US](https://support.apple.com/kb/PH25088?locale=en_US)
- These are all the pre-requisites.

Bluemix comes with two Python build packs that are maintained by the open source community

1. The first is the Python buildpack <https://console.ng.bluemix.net/catalog/starters/python/> The original version is found on GitHub <https://github.com/cloudfoundry/python-buildpack>, and it is based on the Heroku Buildpack.

More about the Python buildpack on Bluemix at this URL:

[https://console.ng.bluemix.net/docs/runtimes/python/index.html#python\\_runtime](https://console.ng.bluemix.net/docs/runtimes/python/index.html#python_runtime)

2. The popular micro framework <http://flask.pocoo.org/> is also hosted on Bluemix, as a Boilerplate. <https://console.ng.bluemix.net/catalog/starters/python-flask/>

View all

### Create a Cloud Foundry App

Python Flask

A simple Python Flask application that will get you up and running quickly.

Community

View Docs

VERSION 0.0  
TYPE Boilerplate  
REGION US South

App name: Enter a unique name

Host name: Enter a unique name

Domain: mybluemix.net

Selected Plan:

Python

Default

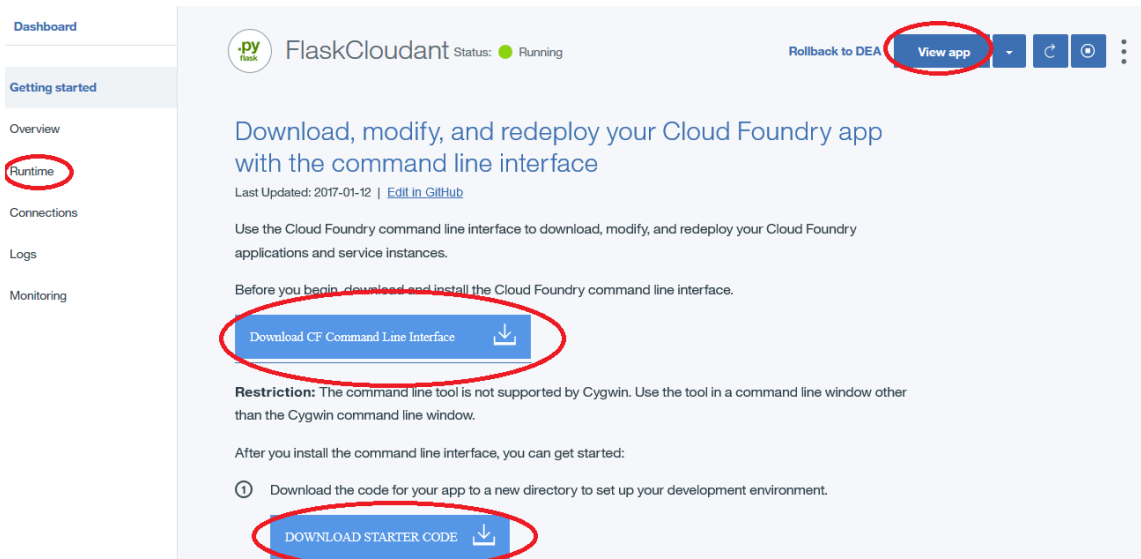
.py  
Python

Need Help?  
Contact Bluemix Sales  
[console.ng.bluemix.net/catalog/](https://console.ng.bluemix.net/catalog/)

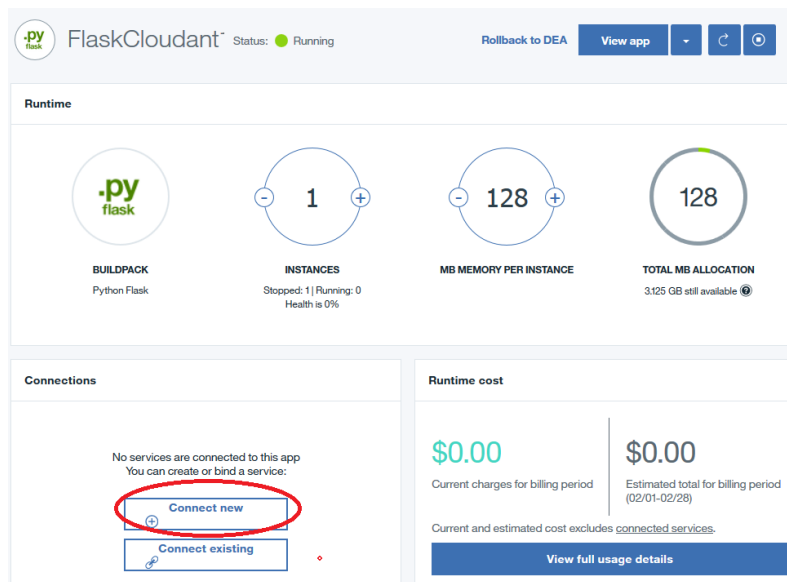
Estimate Monthly Cost  
[Cost Calculator](#)

Create

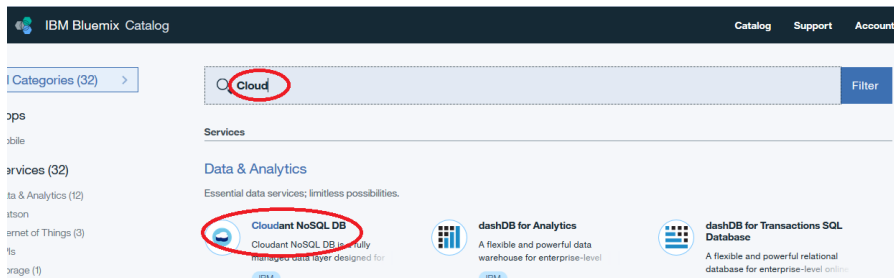
In this workshop we will use the Flask buildpack. We will start by entering a unique app name in the Bluemix window, and then click on the **Create** button, as shown in the screen capture above.



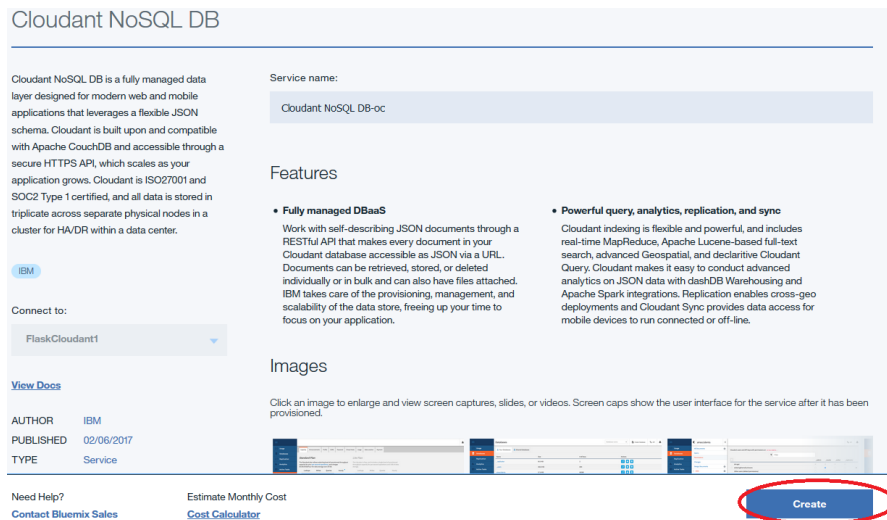
1. When the buildpack has been launched we come to the **Getting Started Window**, see the window above, where we can see the actual app, by clicking the **View App** Button
2. We will then click on the **Download CF Command Line Interface** button, circled in red above, to download and install the Cloud Foundry command line interface which we will use to interact with our app from the command line on our laptop. Let's do so now.
3. And we will then click on the **Download Starter Code** button to download the client code that we can use to begin to build our app. Let's do so now as well. We open the zip-file and place it anywhere on our disk. We will return to the Starter code momentarily.



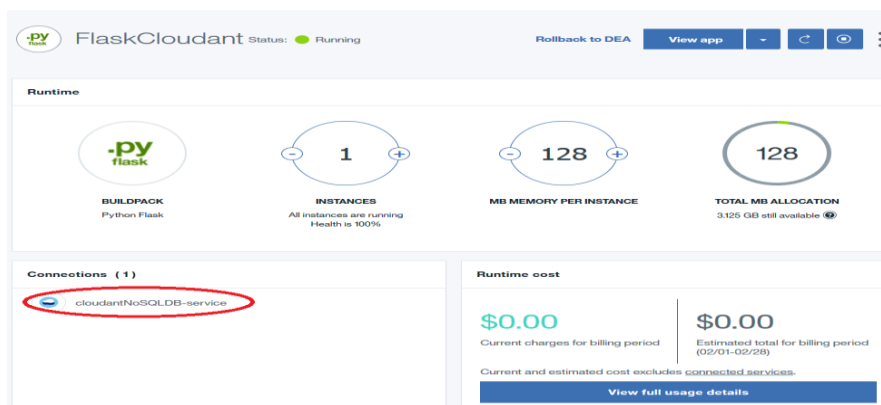
4. In the console window above we see the Connections Pane with a **Connect new** circled in red. Let's click on this link to install the Cloudant NoSQL database.



When we click on the link, the Bluemix catalog pops up and we enter **Cloudant**. We see the **Cloudant NoSQL database**. We accept the default service name and click on the create button in the lower right-hand corner.










Now when we look at the console window we will see the Cloudant NoSQL database tied to our Python Flask build-pack. As shown in the screen capture below.



By clicking on the Cloudant URL in the red circle above we come to the Cloudant console. We will do this later in this lab.

And now with Flask and Cloudant installed and wired together, let's look at the Started Code that we just downloaded to our laptop drive.

 static	File folder
 CHANGELOG.md	MD File
 manifest.yml	YML File
 Procfile	File
 README.md	MD File
 requirements.txt	Text Document
 server.py	PY File

The key files on our drive are the server.py file which consists of the Flask code, and the manifest.yml file which is the package file we use when we deploy the files that we downloaded back to Bluemix by doing a cf push command. Also in the static directory is the index.html file we will modify for our app.

We will now open a console window on our laptop and navigate to the directory where we have installed the Getting Started code. We enter the following command **cf apps**

We will see the name of the app we created is running in Bluemix: Your app will have a different name from mine.

FlaskCloudant	started	1/1	128M	1G	FlaskCloudant.mybluemix.net
---------------	---------	-----	------	----	-----------------------------

We will then enter the command **cf env <my app name>** where we replace **<my app name>** with our actual app name. When we do so we will see a screenful of text where the most important is

```
CAP_SERVICES": {
  "cloudantNoSQLDB": [
    {
      "credentials": {
        "host": "c3eb2101-67bd-4213-bab4-.....bluemix.cloudant.com",
        "password": "3d031f00fbec27750562d30596e203c0fa6765595f6c960.....",
        "port": 443,
        "url": "https://c3eb2101-67bd-4213-bab4-2023... bab4-2023b3d7d97e-bluemix
```

The only line we need to save is the very long string after "url": that starts with "https://" and ends with bluemix. We will save this string, which consists of the concatenated username, password and host variables. We will need this string to access the Cloudant service.

We are now ready to begin to work with our app and with Cloudant. To do so we will use the Python Flask micro framework that we installed earlier.

## A brief tour of Flask



Flask <http://flask.pocoo.org/> is a popular BSD-licensed micro framework for Python based on Werkzeug, <http://werkzeug.pocoo.org/>, and the Jinja 2 template engine. <http://jinja.pocoo.org/docs/2.9/>

The salient portions of the out-of-the-box version of Flask look as below:

A good and simple getting started guide with Flask can be found on this link: <http://flask.pocoo.org/docs/0.12/quickstart/>

```
import os
from flask import Flask, jsonify

app = Flask(__name__)

@app.route('/')
def Welcome():
    return app.send_static_file('index.html')

@app.route('/myapp')
def WelcomeToMyapp():
    return 'Welcome again to my app running on Bluemix!'

@app.route('/api/people')
def GetPeople():
    list = [
        {'name': 'John', 'age': 28},
        {'name': 'Bill', 'val': 26}
    ]
    return jsonify(results=list)

@app.route('/api/people/<name>')
def SayHello(name):
    message = {
        'message': 'Hello ' + name
    }
    return jsonify(results=message)

port = os.getenv('PORT', '5000')
if __name__ == "__main__":
    app.run(host='0.0.0.0', port=int(port))
```

The code consists of a number of routes, or URLs if we will, that handle requests for service.

At the bottom of the file is the invocation of the built-in server with a port number that you can modify.

To simplify our work we will leave all the Flask code in place and simply replace the code in index.html in the Static folder in the Getting Started Code that we downloaded and unzipped earlier. And then we will just add a couple of simple routes in the welcome.py file to handle the requests of the app we are going to write.

## Interacting with Cloudfant with the console

Cloudfant comes with a client console which we mentioned earlier. Let's click on it and we will come to a Launch button, which we will click on.

[← Data & Analytics](#)

## cloudantNoSQLDB-service

[Manage](#)[Service Credentials](#)[Plan](#)[Connections](#)

## Cloudant NoSQL DB

[LAUNCH](#)

Once we have clicked on the Launch button we will come to the Cloudant Console where work with databases and data. When we create a Bluemix app with data we usually start by loading it into our database from the Cloudant Console in the picture below.

**Databases**

Database name

Create Database [API](#)

[Your Databases](#)

Name	Size	# of Docs	Actions
<a href="#">_replicator</a>	51.7 KB	1	
<a href="#">_users</a>	42.6 KB	0	
<a href="#">botdb</a>	58.2 KB	18	
<a href="#">katarina</a>	1.7 KB	1	
<a href="#">teaterhotellet</a>	4.2 KB	3	

## Interacting with Cloudant with the CouchDB API

To read and write to and from the Cloudant database from Python we make use of the CouchDB Python package for working with CouchDB from Python code. <https://pythonhosted.org/CouchDB/>. The reason we can do this is because Cloudant follows the CouchDB standard.

### We'll start by connecting our Flask program to the CouchDB API

Since we will write some new python code in the welcome.py file I have included the updated file with all the new code in my github repository: <http://ibm.biz/Bdswg2>

In start of our welcome.py file in Flask, where we currently have the following code

Python and Cloudant on IBM Bluemix

```
import os
from flask import Flask, jsonify
```

We add the following imports

```
import couchdb, couchdb.mapping
from couchdb import Database, Server, Session
from couchdb.mapping import Document, TextField, IntegerField, DateTimeField
```

We also need to put the following line in the requirements.txt file in the Starter Code

```
Couchdb
```

We then need to do one more thing.

## We'll then create a database with the CouchDB API

Before we begin it is important to point out that we have placed a common function in a separate file called mymethods.py.

The file looks like below. Please note that the ("**https://c3eb210...1-bluemix.cloudant.com**") has been abbreviated.

This is the url we get back when we do **cv env** as we discussed on page 5.

It is this URL we need to connect to the Cloudant NoSQL database.

```
import os
from flask import Flask, jsonify
import json
import requests
from flask import Flask, render_template, request, url_for
import couchdb, couchdb.mapping
from couchdb import Database, Server, Session
from couchdb.mapping import Document, TextField, IntegerField, DateTimeField
from datetime import datetime
import platform

def selectdb(srv):
    if (srv == "local"):
        couch = couchdb.Server("http://127.0.0.1:5984/")
    elif (srv == "remote"):
        couch = couchdb.Server("https://c3eb2101-67b.....dac174e0@c4213-bab4-2023b3d7d97e-bluemix.cloudant.com")
    else:
        return "Database entry error"
    return couch
```

This file, without the full URL, can be found in my GitHub repository: <http://ibm.biz/BdswgX>

Before we enter our HTML code into the index.html file that we downloaded with the Starter Pack, let us copy the original index.html file to index.original.html or a similar name.

We will then create a brand new index.html file.

We now begin by entering the following HTML code in the new index.html file in the static folder in the starter kit. You will recognize a typical HTML Form that sends its action to the **/createdb** route in the welcome.py file.

```
<table>
<caption><h2>Create Database</h2></caption>
<form action="/createdb" method="post">
```



```

<tr><td>Server: <td><select name='srv'>
  <option value="remote">Remote Server</option>
  <option value="local">Local Server</option>
</select> </tr>

<tr><td> Database : <td><input type="text" name="db" size="35"></tr>
<tr><td> <input type="submit" value="Submit"></tr>
</table>
</form>

```

When we press the submit button we will invoke the **/createdb** route in the welcome.py file, as shown below.

```

#-----
@app.route('/createdb', methods = ['POST', 'GET'])
def createdb():
    db = request.form['db']
    srv = request.form['srv']
    print ("Database " + db)

    couch=selectdb(srv)

    print ("Creating database " + db)
    rc = couch.create(db)

    return "created database " + db

```

The key line above is the following: **rc = couch.create(db)**

## We'll then write data to our Cloudant database

In the **Flask index.html** file:

```

<form action="/dbinsert" method="post">
<table>
<caption><h1>Insert data in database</h1></caption>
<tr><td>Server: (local), (remote): <td> <input type="text" name="server" maxlength="40" size="40"></tr>
<tr><td> Database :<td> <input type="text" name="database" size="40"></tr>
<tr><td> Name :<td> <input type="text" name="name" size="40"></tr>
<tr><td> Age :<td> <input type="text" name="age" maxlength="40" size="40"></tr>
<tr><td> Health :<td> <input type="text" name="health" maxlength="40" size="40"></tr>
<tr><td> <input type="submit" value="Submit"></tr>
</table>
</form>

```

We use a Form to gather the data and post it to the Flask route in the welcome.py file.

```

@app.route('/dbinsert', methods=['GET','POST'])
def dbinsert():
    srv = request.form['srv']
    database = request.form['database']
    id = request.form['id']

```

```

name    = request.form['name']
age     = request.form['age']
health  = request.form['health']

couch=selectdb(srv)

db = couch[database]
doc = ({ 'name': name,'age':age,'health': health})
db.save(doc)

return " database " + database + " name: " + name + " age " + age + " health: " + health

```

## We'll then read data from our Cloudant database and display it

```

<form action="/classprint" method="post">
<table>
<caption><h2>Fetch data</h2></caption>
<tr><td>Server: <td><select name='srv'>
<option value="remote">Remote Server</option>
<option value="local">Local Server</option>
</select> </tr>

<tr><td>Database :<td> <input type="text" name="db" size="40"></tr>
<tr><td><input type="submit" value="Submit"></tr>
</table>
</form>

```

In the Index.html file in the static folder of our Code Starter we place the following code:

```

<form action="/classprint" method="post">
<table>
<caption><h2>Fetch data</h2></caption>
<tr><td>Server: <td><select name='srv'>
<option value="remote">Remote Server</option>
<option value="local">Local Server</option>
</select> </tr>

<tr><td>Database :<td> <input type="text" name="db" size="40"></tr>
<tr><td><input type="submit" value="Submit"></tr>
</table>
</form>

```

And in the **classprint** route in the welcome.py file we handle the request,

```

@app.route('/classprint', methods = ['POST'])
def classprint():
    srv    = request.form['srv']
    db     = request.form['db']
    print ("Classprint DB : "+ db)

    couch=selectdb(srv)
    db = couch[db]

    record = "<h1>Records</h1><table><hr>"

```

```

for id in db:
    #print (id)
    doc = db[id]
    rid = doc['_id']
    name = doc['name']
    age = doc['age']
    health = doc['health']
    rec = "<tr><td>Id: "+rid+"<td>Name: <td>"+name + "<td> Age: "+str(age)+"<td> Health: "+health+"</tr>"
    record=record+rec
record=record+"</table>"
print ("Classprint")
return record

```

Here is the result from the CouchDB database:

Id: 16c1a151eb972492d50a1723e9008de1 Name: Kristina Petterson Age: 42 Health: No health issues

Id: 16c1a151eb972492d50a1723e900a117 Name: Jules Sylvain Age: 62 Health: High blood preassure

You can obviously change the data you write and read from Cloudant.

## Wrapping up

---

We have created a Python Flask app that reads and writes to the Cloudant NOSQL database on Bluemix. We have also set up a local Couch Database on our laptop that is a mirror of the Cloudant database on Bluemix.

We are now ready to modify this app for our own purpose to compete in the next hackathon.

We can for example add Watson Services to our App and store data in Cloudant.

## Appendix, installing CouchDB on our laptop

---

We have installed Cloudant and Flask in the Bluemix Cloud. We have also downloaded the Starter Pack to our laptop.

We can also, if we want to, install CouchDB on our local drive. <http://couchdb.apache.org/>. It is really easy and it would give us a full-fledged local environment to experiment with.

We can use the Bluemix-based and laptop-based versions of CouchDB interchangeably. And then when we have our app ready to go, Simply switch over to Cloudant-only.

The code in this lab as a switch that allows you to simultaneously run with Cloudant on Bluemix and Couchbase on your laptop during the development phase. The choice is yours.

## Resources

- Deploying a Python Web Application to IBM Bluemix:  
<https://www.ibm.com/blogs/bluemix/2016/03/deploying-python-app-to-bluemix/>
- Create a scalable and fault-tolerant REST endpoint using Flask and Python  
<https://www.ibm.com/developerworks/cloud/library/cl-scalable-fault-tolerant-rest-endpoint-flask->

python-bluemix-trs/

- <https://notroot.wordpress.com/2010/11/04/playing-with-python-and-couchdb/>
- Deploying a Python Web Application to IBM Bluemix:  
<https://www.ibm.com/blogs/bluemix/2016/03/deploying-python-app-to-bluemix/>
- Simple Hello World Python App using Flask  
<https://www.ibm.com/blogs/bluemix/2015/03/simple-hello-world-python-app-using-flask/>