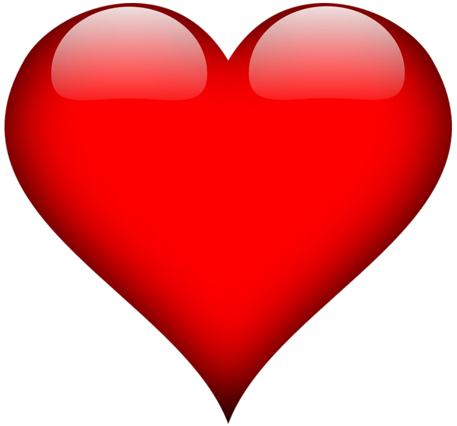
Python Packaging

Niv Mizrahi

VP R&D @ emedgene
@ravinizme
github.com/nivm



We Love Python

Python

The syntax is simple and expressive, it has tons of open source modules and frameworks and a great community.

Python is everything

Almost every aspect of coding:

- Software Development
- [Object, Functional, Aspect] oriented programing
- Web development
- Data science
- Automation
- Deployment
- Devops
- etc.

BUT,

Like everything in the world,
it has its drawbacks

Packaging & Deployment

We're gonna talk about a few:

- Dependency Management is lacking.
- There is no clear way of packaging and deploying your service.

Don’t Diss Python

Let’s Solve a Real World Problem With Python

Let’s Build a Script that counts to 10

Credit: Itai Frenkel @ Forter
softwarearchitectureaddict.com

- Start with something Simple

Credit: Itai Frenkel @ Forter
softwarearchitectureaddict.com

- Start with something Simple
- Client - Server

Credit: Itai Frenkel @ Forter
softwarearchitectureaddict.com

- Start with something Simple
- Client - Server
- Rest API

Credit: Itai Frenkel @ Forter
softwarearchitectureaddict.com

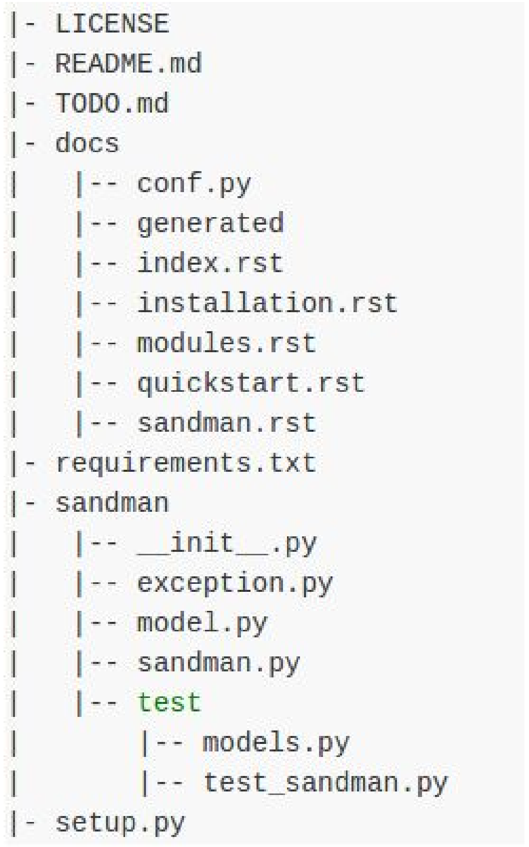
- Start with something Simple
- Client - Server
- Rest API
- MySQL

Credit: Itai Frenkel @ Forter
softwarearchitectureaddict.com

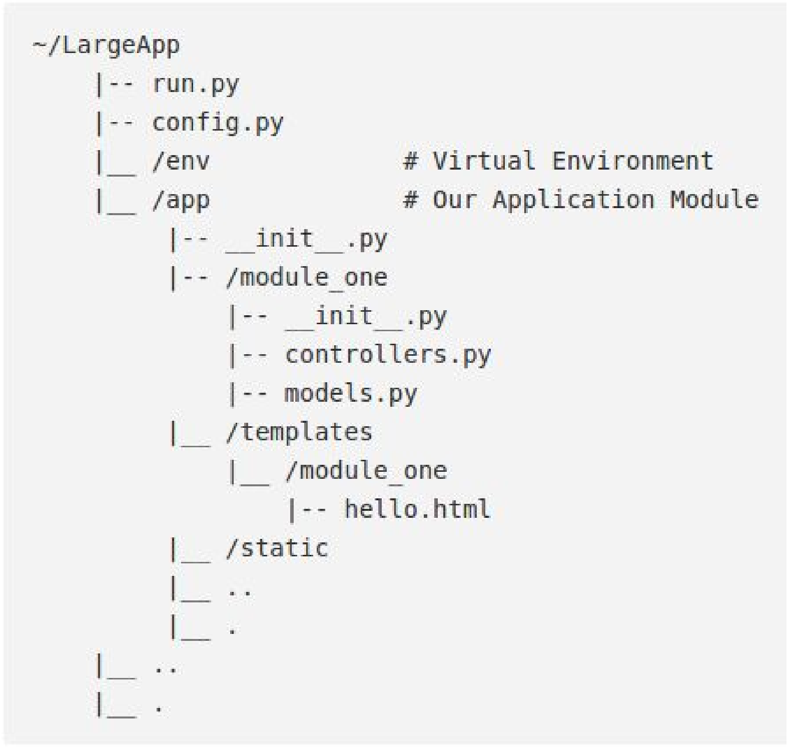
- Start with something Simple
- Client - Server
- Rest API
- MySQL
- KISS

Credit: Itai Frenkel @ Forter
softwarearchitectureaddict.com

What's the structure of the project?



```
|- LICENSE
|- README.md
|- TODO.md
|- docs
|    |-- conf.py
|    |-- generated
|    |-- index.rst
|    |-- installation.rst
|    |-- modules.rst
|    |-- quickstart.rst
|    |-- sandman.rst
|- requirements.txt
|- sandman
|    |-- __init__.py
|    |-- exception.py
|    |-- model.py
|    |-- sandman.py
|    |-- test
|        |-- models.py
|        |-- test_sandman.py
|- setup.py
```



```
~/LargeApp
    |-- run.py
    |-- config.py
    |__ /env             # Virtual Environment
    |__ /app             # Our Application Module
         |-- __init__.py
         |-- /module_one
             |-- __init__.py
             |-- controllers.py
             |-- models.py
         |__ /templates
             |__ /module_one
                 |-- hello.html
         |__ /static
         |__ ..
         |__ .
    |__ ..
    |__ .
```

Add Flask dependency

```
pip install Flask
```

Oops (I didn’t again)

I didn't use a virtualenv

Why didn’t pip warn me ??

Virtualenv & Pip

Let’s Install virtualenv and pip

```
sudo apt-get install python-virtualenv
```

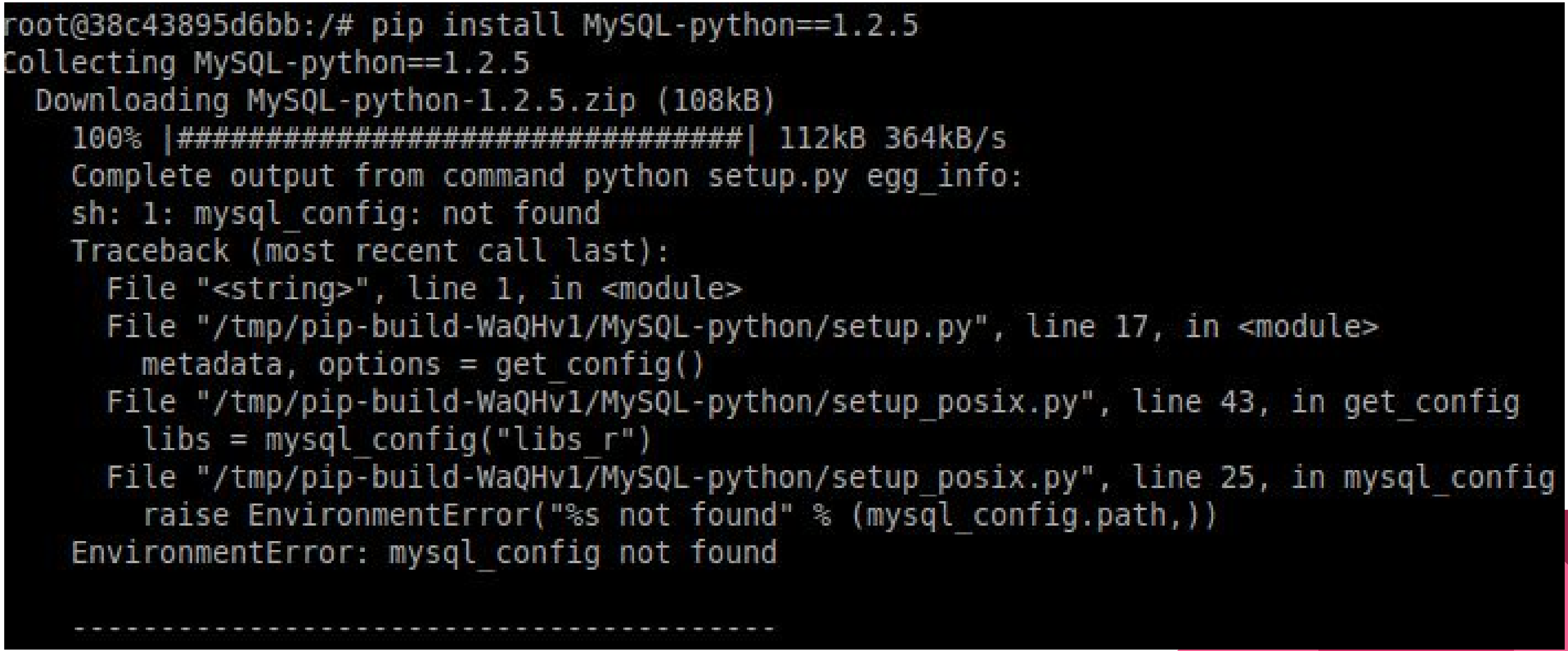
and use our requirements.txt

```
Flask==0.10.1

MySQL-python==1.2.5
```

We are Now Surely Ready

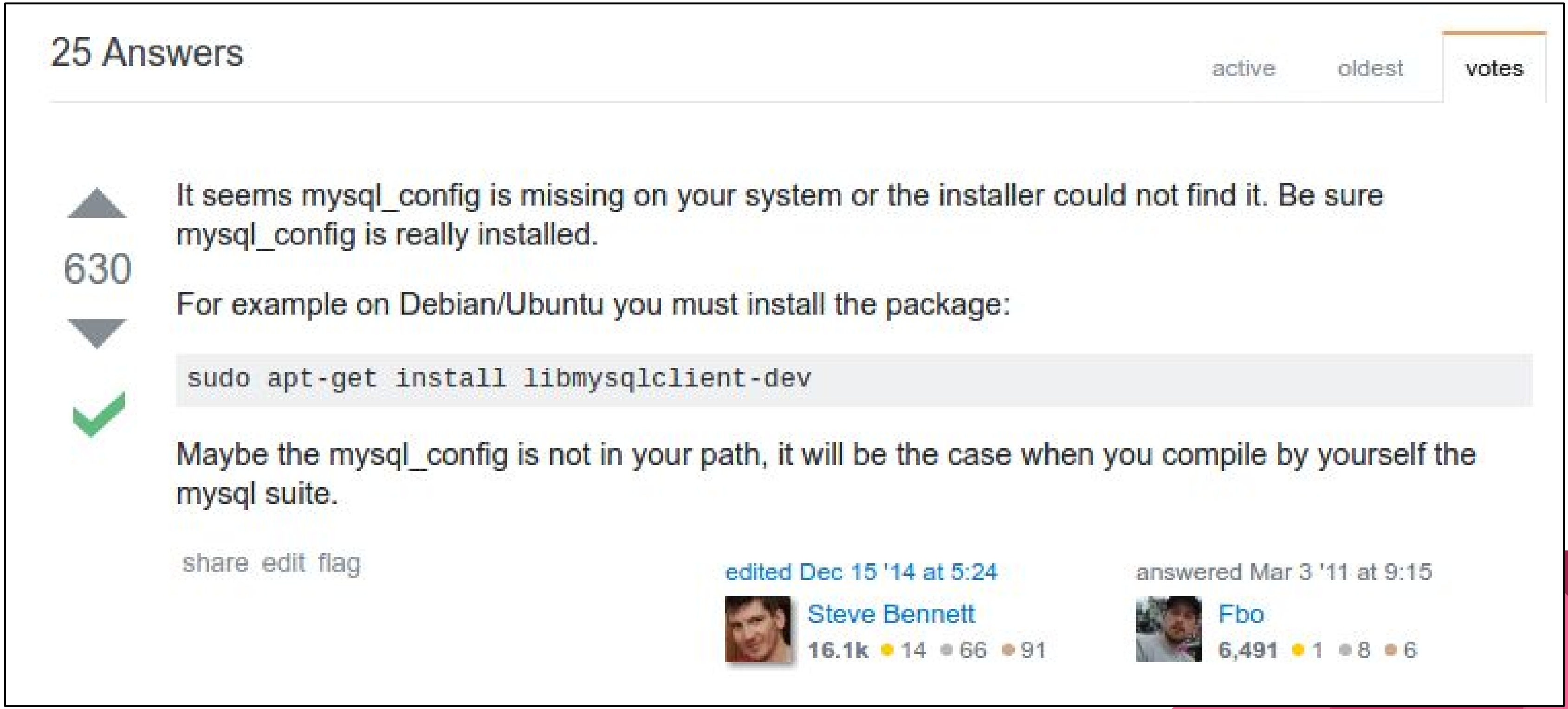
```
pip install -r requirements.txt
```

Daaamn!



```
root@38c43895d6bb:/# pip install MySQL-python==1.2.5
Collecting MySQL-python==1.2.5
  Downloading MySQL-python-1.2.5.zip (108kB)
    100% |################################| 112kB 364kB/s
    Complete output from command python setup.py egg_info:
    sh: 1: mysql_config: not found
    Traceback (most recent call last):
      File "<string>", line 1, in <module>
      File "/tmp/pip-build-WaQHv1/MySQL-python/setup.py", line 17, in <module>
        metadata, options = get_config()
      File "/tmp/pip-build-WaQHv1/MySQL-python/setup_posix.py", line 43, in get_config
        libs = mysql_config("libs_r")
      File "/tmp/pip-build-WaQHv1/MySQL-python/setup_posix.py", line 25, in mysql_config
        raise EnvironmentError("%s not found" % (mysql_config.path,))
    EnvironmentError: mysql_config not found

    ----------------------------------------
```

Seriously ????



25 Answers

active oldest votes

630

It seems mysql_config is missing on your system or the installer could not find it. Be sure mysql_config is really installed.

For example on Debian/Ubuntu you must install the package:

```
sudo apt-get install libmysqlclient-dev
```

Maybe the mysql_config is not in your path, it will be the case when you compile by yourself the mysql suite.

share edit flag

edited Dec 15 '14 at 5:24
Steve Bennett
16.1k ● 14 ● 66 ● 91

answered Mar 3 '11 at 9:15
Fbo
6,491 ● 1 ● 8 ● 6

Conflict Resolution

We want ORM.

```
> bin/pip freeze | grep Flask
Flask==0.8
> bin/pip install Flask-SQLAlchemy==2.0
> bin/pip freeze | grep Flask
Flask==0.10.1
Flask-SQLAlchemy==2.0
```

And the Actual Code

```
from flask import Flask
app = Flask(__name__)

num = {"counter": 0}

@app.route('/')
def count_to_ten():
    num["counter"] += 1
    return str(min(num["counter"], 10))

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

Now,
Version 1.0.0 is DONE,
Let’s package & deploy it

1. git pull & pray

- We have only one service
- I do need redundancy / HA solution
- Need to deploy on multiple machines

I don’t need anything fancy so I’ll just pull the code.

1. git pull & pray

You might be using automation on **git pull & pray:**

Using Fabric, Chef, Puppet, Ansible, SaltStack, etc.

But if you **pull your code,** download and install your dependencies on your target machine you might need to **pray**

1. git pull & pray

OK.
It succeeded on one machine,
But
Failed on another due to pypi timeout.

Bummer!

OK, What did We Learn?

I need to install dependencies once.

Build once, deploy anywhere.

Also Python applications require system dependencies

2. Native Packages (DEB/RPM)

- Why not use Native Packages?
- We use Native Packages every day.
- Most of the open source applications / infrastructures are installed with native packages.

2. Native Packages (DEB/RPM)

- A debian package acts as **single bundled artifact**
- Native packages take care of **system dependencies** for you (libmysqlclient-dev)

2. Native Packages (DEB/RPM)

- Virtualenv + **relocatable**
- I'm using linux so **fpm** looks interesting because **debian packages** take care of system dependencies for you.

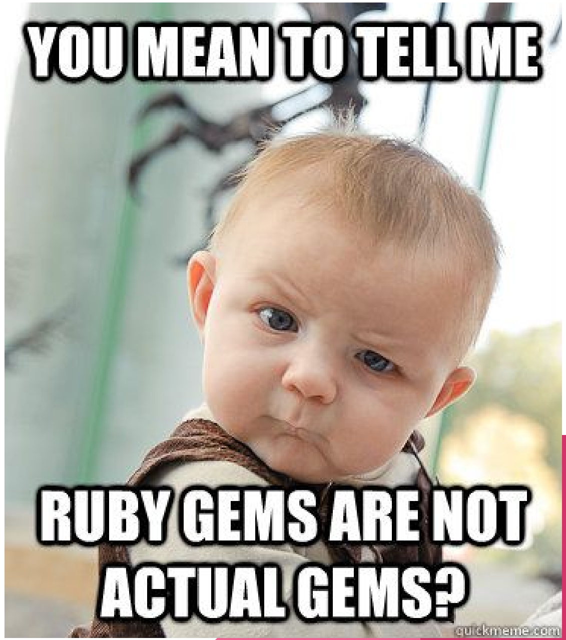
  ```
  fpm -s <source type> -t <target type> [list of sources]...
  ```

2. Native Packages (DEB/RPM)

```
virtualenv --no-site-packages -p $PYTHON

$workdir "$workdir/bin/activate"

(cd $my-package-dir && python setup.py)

"$workdir/bin/deactivate"

virtualenv --relocatable "$workdir"

fpm -s dir -t deb -n "$package" -p "$package.deb" -d <system
dependencies> ...
```

2. Native Packages (DEB/RPM)

Quick note, FPM is in ruby!!!



2. Native Packages (DEB/RPM)

You could use

- Stdeb
- py2deb
- Dh-virtualenv
- Fpm
- etc...

2. Native Packages (DEB/RPM)

- Jenkins to run fpm to build the package
- Python’s wheel cache to avoid re-building dependencies.
- A single bundled artifact (a debian package)

2. Native Packages (DEB/RPM)

But still,

- Our production environment/build machine are **messy** with deb dependencies.

- What if a developer will accidentally removed the “libmysqlclient-dev” ?

  When we will find out ?

2. Native Packages (DEB/RPM)

Only in installation in a new instance.
Meaning the test aren’t checking all aspect of our package management.

OK, What did We Learn?

We need:

- Deployment methods need to be fully reproducible
- Don’t allow deployment to affect the machine state (leaving trace) - Be Green
- Test your packages on a clean instance each time

3. Docker

What is Docker?

- Application + dependencies in one unit.
- Always run the same
- App Isolation

3. Docker

Yes, Docker is a big change for your production environment.

And you might say:

- We already have an automated deployment solution
- We will need a docker registry
- Our developers need to learn docker in production

3. Docker

You could at least start by setting up a Continuous Integration environment using Docker.

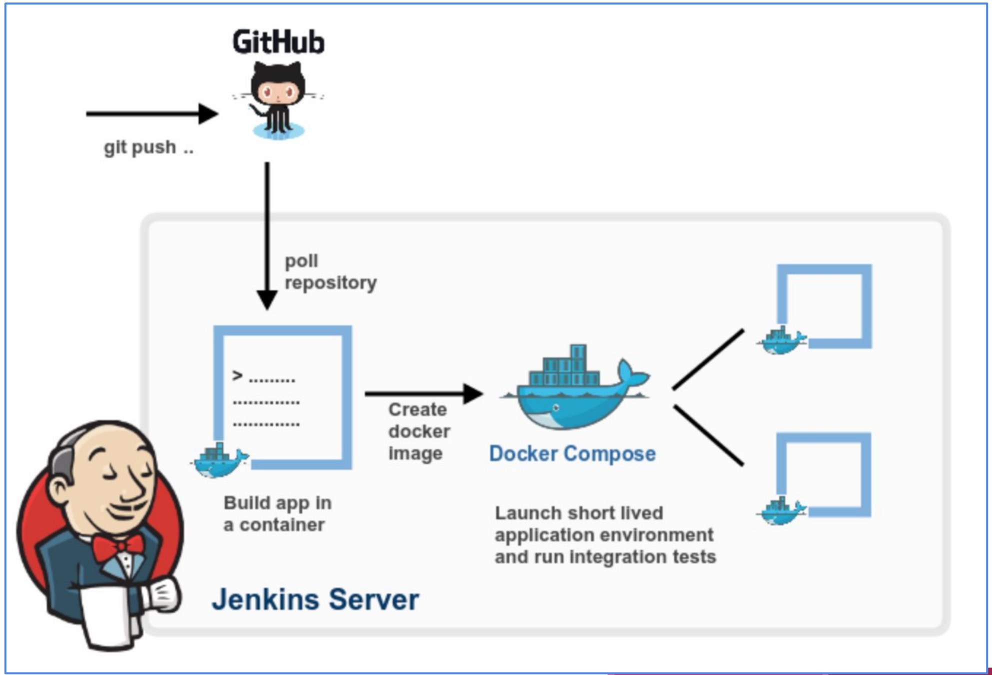
Make your, Test and build task run in an isolated container.

We use JENKINS - Docker Plugin

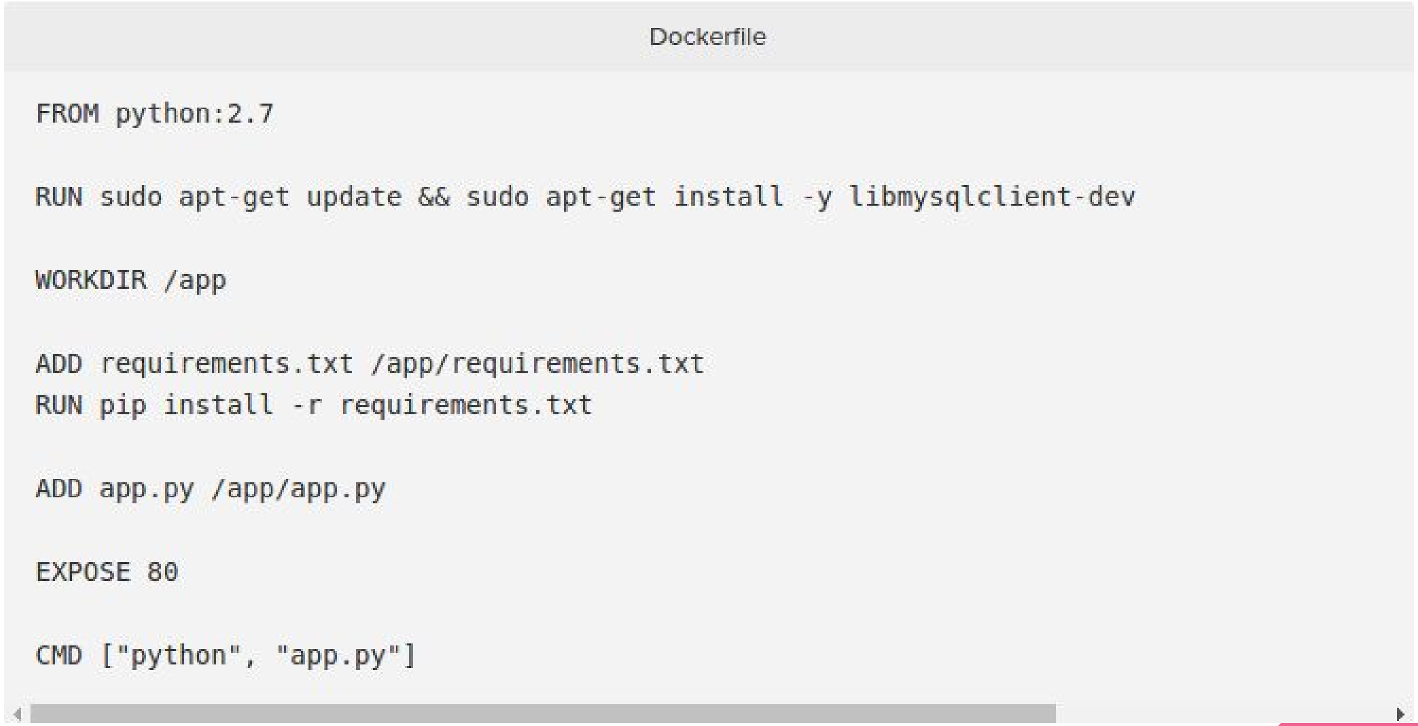
3. Docker

Docker as your

 package management solution



3. Docker



Dockerfile

```
FROM python:2.7

RUN sudo apt-get update && sudo apt-get install -y libmysqlclient-dev

WORKDIR /app

ADD requirements.txt /app/requirements.txt
RUN pip install -r requirements.txt

ADD app.py /app/app.py

EXPOSE 80

CMD ["python", "app.py"]
```

3. Docker

YO DAWG I HEARD YOU LIKE VIRTUALENV
SO I PUT A VIRTUALENV IN YOUR VIRTUALENV
imgflip.com

TL DR - If You Just Woke Up

If you are currently “**git pull & pray**” moving to a **native package solution** might be an easy/effective solution for you. (deb/rpm)

Moving to Docker might take time/risks,
But getting familiar with Docker in your build machine is easy
Grow from there.

Resources

- Why I hate virtualenv and pip - Hacker News
- Packaging-deploying-python - nylas
- Packaging a flask app in a debian package - plankandwhittle
- Things I wish pip learned from npm - Alon Nisser
- Softwarearchitectureaddict.com - Itai Frenkel - Forter
- JENKINS - Docker Plugin

- Thanks for **Shai Cantor**, for helping and putting together the presentation

In Memory of

Udy Brill

Thank You

Q & A