Deployability *of* **Python** web applications

Bruno Renié — EuroPython 2013

Deployability, n

The extent to which something is deployable

Disclaimer

Most of this isn't **python-specific**
or even **web-specific**

Oriented at **custom infrastructures**
Some things still apply if you're on PaaS

How easy it is to **install**, **configure** and **operate** your software?

Mostly about **devs** and **ops**
working together

12factor.net

installation
configuration
operation

Installation

Installing postgres

```
sudo apt-get install postgresql
```

Installing a python webapp

```
sudo apt-get install build-essential python-virtualenv
git clone https://deadbeef@github.com/corp/repo
cd repo
virtualenv env
env/bin/pip install -r requirements.txt
# Figure out PYTHONPATH
```

Installing a py

```
sudo apt-get install build
git clone https://deadbeef
cd repo
virtualenv env
env/bin/pip install -r req
# Figure out PYTHONPATH
```

Installing software is
a **solved problem**

Just use packaging

Yep, python packaging

Why python packaging?

Release process
Dependency management
Trivial rollbacks
Easy system packaging

Packaging in 30 seconds

```
# setup.py
from distutils.core import setup
from setuptools import find_packages

with open('requirements.txt') as reqs:
    install_requires = reqs.read().split('\n')

setup(
    name='project',
    version=__import__('project').__version__,
    packages=find_packages(),
    include_package_data=True,
    zip_safe=False,
    install_requires=install_requires,
)

# MANIFEST.in
include requirements.txt
recursive-include project *
```

Private package hosting

Local filesystem

```
python setup.py sdist
pip install --download dist -r requirements.txt
rsync -avz -e ssh dist/ host.corp.com:/srv/pypi

pip install --no-index --find-links=/srv/pypi myproject
```

Network-based, ala pypi.python.org

HTML directory index (apache / nginx / SimpleHTTPServer)

```
pip install --no-index --find-links=http://host myproject
```

System packages

https://github.com/**jordansissel/fpm**

```
fpm -s python -t deb setup.py

awk -F= '{printf "fpm -s python -t deb -v %s %s\n", $3, $1}' \
    requirements.txt | sh
```

Sign, upload to your private repository

https://github.com/**rcrowley/freight**

```
sudo apt-get install python-myproject
sudo apt-get install python-myproject=1.2.3
```

Pin your dependencies

Bad

```
Django
Django>=1.4,<1.5
```

Good

```
Django==1.4.5
```

This is for **end products**, not **libraries**

http://nvie.com/posts/pin-your-packages/

Configuration

Configuring postgres

```
$EDITOR /etc/postgresql/9.2/main/postgresql.conf
service postgresql restart
```

Does your app have a config file?

`settings.py`, `production_settings.py` are **not** config files

Configuration **!=** code

Problems with configuration as code

Incompatible with packaging

Code shouldn't be **tied to environments**
Code shouldn't be **generated** (salt / puppet / fabric)

Environment-specific code

Production-specific code **will** break production.

Define your configuration

What changes between environments?

Database
Secret key
Host / port
Credentials to external services (AWS, Sentry...)

Read configuration from your code

.ini files
yaml
environment variables
...

Code changes

↓

release, deploy app

Infrastructure changes

↓

write config, reload app

Config as environment variables

Pros

Trivial to set with `$PROCESS_MANAGER`
Native to every programming language
De-facto standard (PaaS). Interoperability!

Cons

Shared hosting
Apache

Case study: Django settings

Before

settings_local.py `DATABASES = {'default': {'HOST': '`**`localhost`**`', …}}`

settings_staging.py `DATABASES = {'default': {'HOST': '`**`staging`**`', …}}`

settings_prod.py `DATABASES = {'default': {'HOST': '`**`prod`**`', …}}`

After

settings.py `DATABASES = {'default': dj_database_url.config()}`

env **`DATABASE_URL="postgres://host:5432/db"`**

Config patterns

Sane defaults when possible

```
PORT = int(os.environ.get('PORT', 8000))
```

Use `*_URL` and parsers to reduce the number of variables

```
EMAIL_URL
DATABASE_URL
REDIS_URL
```

Prevent the app from booting if something critical is missing

```
SECRET_KEY = os.environ['SECRET_KEY']
KeyError: 'SECRET_KEY'
```

In development

django-dotenv
virtualenvwrapper postactivate hooks
custom manage.py
envdir
...

Operation

WSGI

Have a WSGI entry point

```
gunicorn myapp.wsgi -b 0.0.0.0:$PORT
```

Stateless processes

Persistence via external services

Database
Caching
Storage
...

Scale out with processes

More traffic? Spawn more processes.

Caveat: backend services
rarely scale horizontally

Maximize dev/prod parity

Same software

If you use postgres in production, use postgres in development

Same versions

PostgreSQL 9.1 and 9.2 do not perform equally

Same people

Developers should know about infrastructure

Continuous integration/deployment

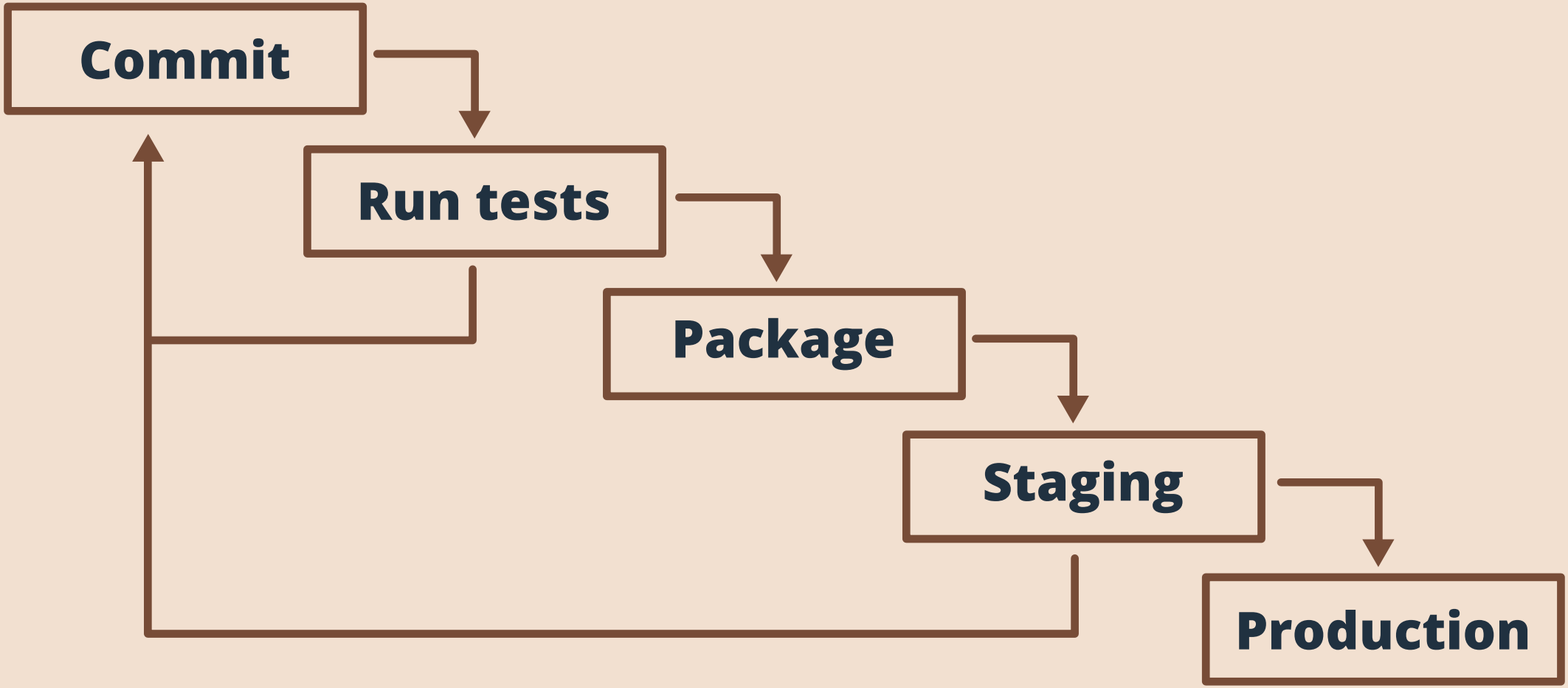
CI **!=** green badge on your github page

CD **!=** always running master in production

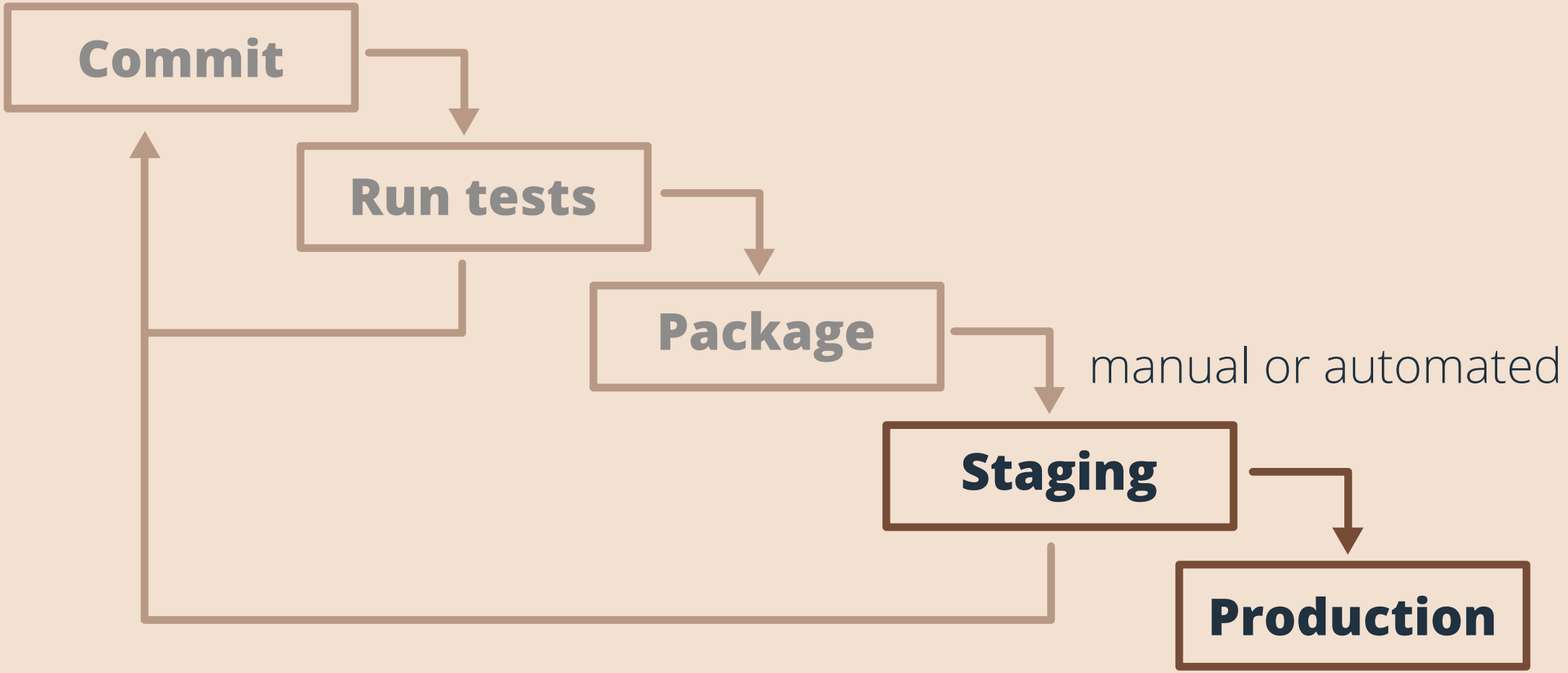
CI Having **shippable** code **tested packaged installable**

CD Deploying it **whenever your want**

Example workflow



Example workflow



Jenkins, **SaltStack**, **IRC bots** are your automation friends

use **packaging** to manage software

clearly define the
configuration contract

automate as much as possible
to minimize **deployment friction**

@brutasse
bruno@renie.fr