The road to continuous deployment

Stefano Rivera

Yola

2 October 2015
Pycon ZA

Outline

What world do our apps live in?

Production server architectures

Typical SASS app:

- Redundant front-ends
- Central DB

Production server architectures

Microservices:

- Redundant front-ends
- Redundant services
- Central DBs

Production server architectures

Async:

- Job queues
- Workers
- Central Queues

Production server architectures

Client Load Balancing:

- haproxy / nginx / hardware
- Single IP / hostname for each service / DB

Production server architectures

Internal Service Load Balancing:

- the same, or
- intelligent fancy clients
- service discovery: zookeeper / something

Production server architectures

Development environments:

- Production-like environments (QA)
- Developer's laptops (local dev, maybe one app only)
- Integration environments (all on one box, no LBs)

Production server architectures

Development workflows:

- VCS (git / whatever)
- Build system (?)
- Deployment tool
- ...
- Profit

Outline

What world do our apps live in?

Configuration

Production:

- Probably most complex environment
- Service LBs
- Background workers

Configuration

QA:

- Like production, but different
- No redundancy?

Configuration

Integration environments:

- Everything on one machine
- Name-based vhosting?
- Save the RAM
- Less workers, less background workers

Configuration

Local dev:

- Against prod / QA / int-env?
- sqlite?
- Simple bootstrapping
- Dependencies...

Configuration

Things apps need to know:

- Where are services?
- Where are DBs?
- Secrets
- Other parameters

Configuration

Where should these live:

- Centrally
- In each app

Configuration

Central sucks:

- Adding a setting means commits to 2 repositories
- What uses this setting over here?

Configuration

In each app sucks:

- This setting is the same for every app in production
- I want to put this app in a public github repo

Configuration

Compromise:

- Both
- Which means probably no configuration management system involvement
- per-machine settings

Configuration

yoconfigurator smush

- C: hostname
- C: common
- C: common-ENV
- C: common-ENV-CLUSTER
- C: common-overrides
- A: APP-default
- A: APP-ENV (and cluster)
- C: APP
- C: APP-ENV (and cluster)
- C: APP-overrides

Configuration

Exposing config to apps:

- settings.py - lots of if ladders
- settings.py.ENVIRONMENT - lots of duplication, no shared config
- environment variables - local dev hurts, forks hurt
- config files - actually pretty good
- other things

Configuration

Tooling:

- Config file generator
- Generate against any environment

Outline

Get the code out

Deployment tools

simple git pull

- Actually fairly good, these days
- What about dependencies?
- Configuration?

Deployment tools

git pull + script:

- What happens while this is running?
- What happens if it fails?

Deployment tools

Typical script:

- install OS packages
- manage virtualenv
- build non-python stuff
- build config
- migrate DBs (here be dragons)
- bounce workers

Deployment tools

Unexpected problems:

- PyPI is down
- PyPI changes
- Non-deterministic thing is built
- (esp, minification & filenames)

Deployment tools

Reproducibility:

- Goal: Same generated artifacts on each production server

Deployment tools

Build system:

- Build virtulenvs
- Build non-python
- Build config (?)
- Bundle

Deployment tools

Other benefits:

- Roll-backs
- Centralized build-effort
- Build-time test suites

Deployment tools

Downsides:

- Platform dependencies
- ABI
- test suite length
- Build scripts

Deployment tools

Output:

- git repo
- distro package
- VM
- other: tarball in some kind of archive

Deployment tools

Deployment:

- extract latest build
- generate config?
- run script

Deployment tools

Next goal:

- factor out scripts
- build scripts
- deploy scripts

Deployment tools

How to execute:

- ssh do stuff?
- configuration management system?
- deploy system on box:
  - ssh run command
  - http request

Deployment tools

Deployment automation:

- commit hook triggers build system
- build triggers tests
- build triggers deploy (?)
- deploy triggers validation tests?
- deploy triggers chat notification

Deployment tools

Releases:

- git branches?
- promoted builds?

Outline

Where does it fit in

Configuration management

Does deploys:

- Simple
- Slow (it does everything else, too)

Configuration management

Basic config:

- Standard OS packages and config

Configuration management

Per app:

- external dependencies (rarely change)
- service discovery
- initial deploy (bootstrapping)
- vhosts (?)

Configuration management

Other stuff:

- 3rd party apps that aren't deployed
- DBs
- Caches
- Brokers

Configuration management

Demo envhub

Outline

Continuous deployment means we're going to break things. We need to be able to fix them, quickly

Commit to production latency

Obvious optimizations:

- SSDs
- Cache / mirror PyPI (and npm)
- Cache eggs / wheels
- Cache virtualenvs (and node_modules)
- pre-minify images in git / cache minification
- Get slow tests out of critical path
- Don't build VMs

Outline

Our deployment tool.

Build time:

- Still shell scripts
- (But fairly short boilerplate)
- More and more gulp / grunt
- Spit out tarball, run spade upload

Deploy Workflow:

- receive deploy command
- extract tarball to /srv/APP/versions/VERSION/
- prepare hook
  - generate things that can't be done at build time
- swing /srv/APP/live symlink
- deployed hook
  - bounce things

Hooks:

- Knows about yoconfigurator
- Contains simple templating engine
- Knows about virtualenvs
- Must still be taught about npm
  - But, actually, we keep npm to build-time
- Knows how to drive a Django app
- Knows how to drive upstart
- Knows how to drive tomcat
- Customizable in Python

Outline

What happens during the deploy

zero downtime

ideal:

- web server finishes old requests using old code
- uses new code for new requests

zero downtime

erlang:

- Well, duh
- (But we don't actually deploy any with yodeploy)

zero downtime

openresty:

- :+1:

zero downtime

static HTML Apache:

- :+1:
- But maybe keep old assets around for a while / forever?
- Content-based hashing ftw

zero downtime

Python WSGI:

- mod_wsgi mostly handles this
- some 500s, can be mitigated at LB or client-side retries
- other WSGI containers: Not so much.
  - Alternate between a pair?

zero downtime

Java:

- tomcat7 parallel deployments
- PermGen Space

Outline

What's missing?

What comes next

Better build time:

- Too much class hierarchy
- Not enough implicit config through convention

What comes next

Better orchestration:

- Deploy button
- Integrated graphs

What comes next

Modernize:

- Wheels
- Python 3

Outline

VMs & Containers

Other topics

Log collection

Other topics

Crash collection

Other topics

DB migrations

Other topics

envhub

Other topics

github testing hooks

Thanks!

Questions?

Stefano Rivera
stefanor@debian.org

http://tumbleweed.org.za/

Slides:
Available at https://github.com/stefanor/talks
Built using LaTeX Beamer