Infrastructure Design Patterns with Python, Buildbot, and Linux Containers

David Liu
Python Technical Consultant Engineer
Intel Corporation

Overview

• Introduction
• Breaking out of CI: Infrastructure Design patterns with Buildbot framework pieces
• Hooking things up in weird ways: Ports, multi-masters, and pseudo-RPC
• When things don’t fit: Linux Containers, and keeping things movable
• Pulling it all together with Python
• Real-world architectures that have worked
• Summary
Introduction: On Infrastructure

• What does it mean to have infrastructure?
• Is it automation? Is it orchestration? Is it task runners?
• Many options exist depending on what “needs” you have
  • Full on orchestration with Chef, Puppet
  • Dask, IPyParallel, Joblib (These are mostly numerical)
  • Celery, Kafka

• Many of these examples are heavy-handed or square peg/round hole problems

On Infrastructure (con’t)

• Examples
  • Trying to get a distributed task system such as Dask to run a CRON is not exactly the best use case
  • Trying to get Celery to do a map-reduce operation
  • Trying to get puppet to make a task graph

• In essence, every one of these frameworks are meant for vastly different things!
Breaking out of CI: Infrastructure Design patterns with Buildbot

• Buildbot is normally meant for Continuous Integration (CI), but you can construct things out of the elements in weird ways.
• Just like Lego blocks for infrastructure; this differs heavily from things such as Jenkins or TeamCity
• CI Tasks normally encompass interesting pieces: A scheduler, dependencies, a result
• However, these main task components are actually composed of many other primitives that have been assembled together

Breaking out of CI: Infrastructure Design patterns with Buildbot (Con’t)

Examples:

• Resource pools
• Roles and triggers
• Task runners
• Distributed System + communications
• State Logic

• Change triggers
• Schedulers
• Build steps (scripting steps)
• Master/worker system
• Barriers and semaphores
• Dependency tree
Breaking out of CI: Infrastructure Design patterns with Buildbot (Con’t)

• Because Buildbot splits all these items up, one may be able to wire the components up in unusual ways to meet commonly occurring infrastructure patterns

• **Warning:** Before going any farther, I want to reiterate that what I am about to show is *conceptual* and used for proof-of-concepts, and is no replacement for sound orchestration and proper security

• This is considered a very “off use” of Buildbot (and was not intended by the developers), so just be mindful of this

Breaking out of CI: Infrastructure Design patterns with Buildbot (Con’t)

• Infrastructure design patterns are the common tasks/roles, and interconnects that occur in software deployments
  • Using Buildbot is just one way of solving such examples
  • One can utilize this to prototype something and then convert it to enterprise-level deployments

• Examples:
  • CI->Package->Deployment (common use)
  • Enterprise application deployment
  • License Server
  • Linux Session launching/landing on Servers
  • Home Weather Server w/ Machine Learning tasks
Hooking things up in weird ways: Ports, multi-masters, and pseudo-RPC

• Normally, most CI systems do not expose such controls, but because of the flexibility in Buildbot, one may use it quite freely
• The change-port allows for usage of a script or symlinked call to trigger a task-which gives user-level triggers
• By passing in arguments of the script in, one can essentially “RPC” to a worker with a known resource
  • i.e. run some task where the right version of Python/NumPy is
• Buildbot is controlled via the logic of the master.cfg, which is interpreted as majority Python code

Hooking things up in weird ways: Ports, multi-masters, and pseudo-RPC (Con’t)

• In the buildmaster’s configuration, normal change sources look like the following:
  • c['change_source'] = []
    c['change_source'].append(changes.GitPoller( 'git://github.com/buildbot/pyflakes.git', workdir='gitpoller-workdir', branch='master', pollinterval=300))
  • However, you can add a secondary trigger source:
    • c['change_source'].append(changes.PBChangeSource(port=9999, user='myApp', passwd='AppPassword'))
Hooking things up in weird ways: Ports, multi-masters, and pseudo-RPC (Con’t)

- Matched with the “fakechange.py” script in buildbot-contrib, one can initiate and pass arguments (such as X11 info, user info) to a buildmaster
- Utilizes the twisted.internet and twisted.spread capabilities
- Sends change to the scheduler in the Buildbot master.cfg

Example of using the change-port to launch apps
Hooking things up in weird ways: Ports, multi-masters, and pseudo-RPC (Con’t)

• Multi-master gives the ability to chain tasks and resource pools together to grant capabilities such as load balancers to certain tasks
• Don’t hesitate to have one task kick off another subset of Buildbot instances

• Use `util.BuildFactory()` to send commands to workers via `ShellCommand`
• Note that the worker must be privileged to run command and must have resources, so define workers well

```python
display_var = "DISPLAY=:0"
docker = util.BuildFactory()
docker.addStep(steps.ShellCommand(command=["docker", "run", 
"-it", "-v", 
"/:/files",
"-e", display_var, 
"-v", 
"/tmp/.X11-unix:/tmp/.X11-unix", 
"silex/emacs")))

# Runs a retro session for the terminal
session = util.BuildFactory()
session.addStep(steps.ShellCommand(command=["docker", "run", 
"-v", 
"/tmp/.X11-unix:/tmp/.X11-unix", 
"-e", display_var, 
"jess/1995")))
```
When things don’t fit: Linux Containers, and keeping things movable

• What happens when things don’t want to fit together? or you have security concepts to worry about?
• Use Linux Containers to provide additional design flexibility through composition techniques (docker-compose, as an example)
• Use Containers to also cordon off the riskier bits (prevent volume maps, etc.)
• Provide privilege/non-privileged barriers to separate users from full-privileged resources

When things don’t fit: Linux Containers, and keeping things movable (con’t)

• At some point, you may need orchestration to pull off tasks, so just know what responsibilities you want in what technologies
• Depending on how you approach the problem, you might be able to get away with little or no orchestration
• If all else fails, you can somewhat cheat by having the entire Buildbot + logic inside of a container, and use those as building blocks
Pulling it all together with Python

- So with Python at the forefront, you can utilize the Python scripts injected into buildbot itself, or have the master.cfg `unpack` code that it receives
- The scripting capabilities mean that you can use calls in build steps to achieve things in an RPC format on the workers
- Python can call the build masters easily, so scripting it to do your bidding is free-form
  - Mixing this with file opening, web calls and requests, are just some of the advantages of using Python “glue”

Real-world architectures that have worked

- **Company-wide server application deployment**
  - Used Applications set in containers, called by symlinked python scripts calling ports to start program
  - Company used Orchestration to scale up and down the available workers as a “resource pool” depending on server loads

- **License server for a “floating license”**
  - Company only had one license, and software had no ability to gate phone home data or queue
  - Implemented with buildbot worker, and a master that queued/scheduled/gated the users.
Company-wide server application deployment

- Buildmaster handles queue, session details
- Spin up new workers for larger pool
- Update applications via Docker repository on Worker

User ➔ /usr/bin mapped change port script ➔ Buildbot Master ➔ Buildbot Worker ➔ Server with resource & full privilege

Application Screen ➔ User session or Login Node ➔ Request Application Session ➔ Launch Docker App, Mount Volume, Forward X Session (Screen)
Company-wide server application deployment (con’t)
License Server for a “floating license”

- License(s) can be held by master or the attached stock DB
- Either use available pool to block licenses, or via DB or logic
- Can also just hold a lock on the license file instead of application screen

```
/user/bin mapped change port script
```

User session or Login Node

Server with resource & full privilege

Launch Docker App
Mount Volume
Forward X Session (Screen)
Real-world architectures that have worked

• **Compute server Linux session handler**
  • Company used Buildbot master/worker with workers and X11 forwarding to hand sessions to users; queue system via the master’s bone stock scheduler

• **Home Machine Learning server**
  • Used successfully to create a “dashboard” and “compute center” for my home system, which pulls in aggregate data and does ML on large datasets with classifiers

---

 Compute server Linux Session Handler (Tier setup w/ Multimaster)
Machine Learning Server

Summary

• Through a little bit of ingenuity and creative use of components, one may fashion many of the infrastructure design patterns that appear in software and IT

• Being able to rapidly design proof-of-concepts is possible with this method, and can reveal design considerations before making a proper solution

• Remember that the examples shown are not shown with any security or orchestration

• Keeping an open mind and an eye on upcoming technology can widen the available infrastructure patterns one can design

• Experiment with new tools often to see what patterns can be made next
References

• http://buildbot.net
• https://github.com/buildbot
• Repo for examples (To be posted soon):
  • https://github.com/triskadecaepyon/infrastructure_patterns

Q&A?
https://github.com/triskadecaepyon
https://triskadecaepyon.github.io/