

Auto QA

Building an Automated Quality Assurance Infrastructure Using Open-Source Python Software

Aaron Maxwell
QA Infrastructure, SnapLogic, Inc.
amax@snaplogic.org
amax@redsymbol.net

This talk archived at <http://redsymbol.net/talks/>

Testing Automation

What is an “Automated QA System”?

*A set of policies, behaviors, and supporting software systems to **constantly** test **everything**.*

Uses unit and integration tests, plus other tools, as building blocks

(Similar but different from Continuous Integration)

Prerequisites

Version Control

(for *all* code)

Unit and Integration Tests

(for *nearly all* code)

Management responsibility. Consistent usage by everyone, and done well

Auto QA Ingredients: VC

- Will have one or more “mainline” branches
 - For different products and releases
 - Merges automatically trigger full QA battery
- Developers use QA system on their private branches
 - Run at their request
 - Can limit to specific tests or tools

Auto QA: Reporting

Reporting: Making the results of what the QA system determined usefully available and accessible.

Your QA System is ONLY as good as its reporting of results.

If you don't get this done well... none of the rest matters.

Under appreciated

And critically, critically important.

Auto QA: Build and Installation

Depending on the nature of the system or product...

- Build product installer from source
- Execute installer in a clean environment
- All the above is automated. No manual steps

Auto QA: Actual tests

- Discovery, organization and running of tests
- Code coverage measurements
- Static source analysis tools
- Performance measurements and profiling

Auto QA: Generality

- A good QA system is:
 - **language agnostic**
 - **version control agnostic**
 - **bug-tracking system agnostic**
- Test on multiple platforms
- Framework is extensible

FOSS CI Frameworks

(looking only at select Python frameworks)

- `svnchecker` (<http://svnchecker.tigris.org/>) - framework for Subversion pre/post-commit hooks
- `Bitten` (<http://bitten.edgewall.org>) – built upon Trac
- `Pybots` (<http://pybots.org>) - optimized for Python projects
- `BuildBot` (<http://buildbot.net/>) - CI/autoQA framework

BuildBot is probably the best **general purpose** Python-based, open-source framework available now.

BuildBot

- Created by Brian Warner; current primary maintainer is Dustin Mitchell
- An open-source, Python-based framework for building automated testing systems
- Flexible, language-independent, and easily extensible
- Supports many version control systems and testing policies
- In active development. See <http://buildbot.net/>

BuildBot Architecture

Architectural elements...

- The **master** for coordination, dispatch, reporting
- Many **build slaves** of different platforms that can run tests as directed
- **Changes** objects, encapsulating events that potentially require testing
- **Schedulers** which the master delegates decisions about when and where to order tests
- **Pollers** and **Triggers** signaling the base change events

A Simple BuildBot

- SVN repository, single mainline branch
- Single build platform
- Change events detected by periodic polling
- Product under test is Python based
- Tests run using Twisted's trial

A Simple BuildBot

Some imports omitted...

```
from buildbot.process import factory
svnurl="svn+ssh://svnhost/simplerepo"
```

**# The slaves will run the tests, after
any prerequisites. Here there is a
single Linux slave.**

```
c['slaves'] = [BuildSlave('linuxbot', 'linuxbotpw')]
```

**# Changes will be discovered by polling
the Subversion repository.**

```
c['change_source'] = SVNPoller(svnurl)
```

A Simple BuildBot

The factory defines the build process.

```
fac = factory.BuildFactory()
fac.addStep(SVN(svnurl))
fac.addStep(Compile(command=['python',
                             'setup.py',
                             'build'])
```

Trial is Twisted's test finder/runner/reporter

```
fac.addStep(Trial(testpath="."))
```

A Simple BuildBot

The builder is a Buildbot installation
that implements the factory.

```
main_builder = {'name' : 'main-b',  
                'slavename' : 'linuxbot',  
                'builddir' : 'main',  
                'factory' : fac}
```

```
c['builders'] = [main_builder]
```

The scheduler decides WHEN a QA run happens.
It kicks off the builders.

```
main_scheduler = Scheduler(name='main-s',  
                           branch=None,  
                           treeStableTimer=60,  
                           builderNames=['main-b'])
```

```
c['schedulers'] = [main_scheduler]
```

A More Complex BuildBot

- Still using SVN for VC; project source in Python
- Test under two platforms – Linux and Windows XP
- Measure code coverage using figleaf
- Static source analysis using Pylint

A More Complex BuildBot

Now we have two slaves.

```
c['slaves'] = [BuildSlave('linuxbot', 'linuxbotpw'),  
              BuildSlave('winbot', 'winbotpw')]
```

Let's define a “full-on” factory that does

more work than the regular one

```
full_fac = factory.BuildFactory()  
full_fac.addStep(SVN(svnurl))  
full_fac.addStep(Compile(command=['python',  
                                  'setup.py', 'build'])  
full_fac.addStep(CreateListings())  
full_fac.addStep(Figleaf(tests='all_tests.txt',  
                       sources='all_sources.txt'))  
full_fac.addStep(Pylint(command=['pylint', 'src']))
```

A More Complex BuildBot

```
# We need to specify a builder for it
full_builder_linux = {'name' : 'full-b-linux',
                      'slavename' : 'linuxbot',
                      'builddir' : 'full',
                      'factory' : full_fac}
full_builder_win = {'name' : 'full-b-win',
                    'slavename' : 'winbot',
                    'builddir' : 'full',
                    'factory' : full_fac}
c['builders'] = [main_builder,
                 full_builder_linux,
                 full_builder_win]
```

A More Complex BuildBot

```
# Rather than triggering off of commits, we
# just want this one to run every night. at 2 AM
full_scheduler = Nightly(name='full-s',
                          builderNames=['full-b-linux', 'full-b-win'],
                          hour=2)
# Now we have two schedulers – the regular one,
# and the nightly.
c['schedulers'] = [main_scheduler, full_scheduler]
```

Extending BuildBot

```
from buildbot.steps.shell import ShellCommand
class Figleaf(ShellCommand):
    def __init__(self, test_file, sources):
        self.test_file = test_file
        self.sources = sources
        ShellCommand.__init__(self)
    def start(self):
        self.setCommand(['figleaf',
                        '--files-list',
                        self.sources,
                        '--output-dir',
                        './coverage_report/',
                        self.test_file])
        ShellCommand.start(self)
```

Extending BuildBot

```
from buildbot.steps.shell import ShellCommand
class CreateListings(ShellCommand):
    def start(self):
        if 'nt' == os.name:
            create_listing = 'create_listing.bat'
        else:
            create_listing = 'create_listing.sh'
        self.setCommand(create_listing)
        ShellCommand.start(self)
```

Thank You!

... to the developers of BuildBot (<http://buildbot.net/>) for creating an outstanding Python CI framework

... to the developers of all the other testing tools we love to use. Too many to enumerate()

... to SnapLogic (<http://snaplogic.org/>), for sponsoring my trip here, and being a great bunch of **scary smart** engineers to work with

... and **YOU** for listening!

Aaron Maxwell * amax@snaplogic.org * amax@redsymbol.net

This talk archived at <http://redsymbol.net/talks/>