Case Study of Arnaud Lacour

Who Are You?

I'm Arnaud Lacour. Quality's my thing. Even though I'm originally a developer, I've always looked at it from the angle of high availability, reliability, catastrophic recovery and such. On the performance side of things, I like to keep an eye on regressions introduced as more features get added.

What Do You Use Hudson For?

We use (and abuse?) Hudson for a number of things:

- good old continuous build. We have an "instant build" job, roaming among the slaves pool to build the project whenever a commit is detected. That job uses the poll SCM feature and is configured to poll every minute. We've seen that it sometimes can be too long anyway, I'd need to tweak to do incremental updates of the workspace, the difficulty being that it roams, so you're not guaranteed that an existing workspace will be there.
- continuous (errrr, close enough) unit testing. This job is run with 4 types of JVMs: Sun Java 5, Sun Java 6, OpenJDK 7 and BEA JRockit 5 on Windows.
- daily functional tests. This is based on a test suite using IBM's STAF framework and is such that roaming is not possible so these 6 jobs are tied to dedicated slaves.
- continuous upgrade (the project has an upgrade utility that allows the users to upgrade -or downgrade for that matter- from build X to build Y. There's a job that checks that backward and forward compatibility are not broken by new commits. This Hudson job incrementally updates the workspace revision by revision to allow to pinpoint when a problem is introduced.
- continuous perfomance monitoring. We hooked up SLAMD to Hudson to provide continuous performance assessment. This has saved us quite a few times after what seemed like harmless commits decreased the performance of our project.

Build System

Our build system is written in Ant and we use a single project with little to no external dependencies in a java.net subversion repository.

Distributed Builds

The variety of systems we have is the hard thing to manage for us:

- Solaris
  - 10
    - x86
    - x64
    - sparc
  - 11 (b55 and b69)
    - x64
- Windows
  - XP
    - x86
  - 2000
    - x86
  - 2003
    - x86
    - x64
  - Vista
    - x86
    - x64
- Linux
  - 2.6
    - x86

There are many more possible conbinations but we just don't have the hardware or the energy (or will) to maintain so many boxes. Also, most of the time exhaustive coverage is not required to catch 90% of issues. It takes very little DOE to figure that out and mixing the platforms was done to maximize the odds of finding those 90% of issues. For some of those OSes, we actually have 2 slaves to be able to execute more tests in parallel. We're closing in on 20 nodes and we're going to add 10 more in the near future for automated real-life deployments simulations.

Managing Slaves

Managing all those slaves can be a chore unless it's done right. At least you have to find what works for you. I found that naming the common root to all things hudson /path/to worked fine for me as it works on Unixes and Windows platform (Ant maps /path/to to \path\to which works fine). Then I do the symlink trick on Unixes and I just install hardwired on Windows. Hudson home is always set to /path/to/hudson for example and Java is always /path/to/java /version/latest. this makes managing the tools dependency maintenance really easy on me. And Hudson configuration is down to the bare minimum.

Leveraging The Plugins Power

All in all, if you looked at other CI tools, nothing comes close to Hudson in terms of usability and serviceability. In our previous CI system, we had poor to no reporting facilities and so Hudson comes as a tremendous improvement. For example, here's a snapshot of our performance job.