Continuous Integration: Put it at the heart of your development

Susan Duncan

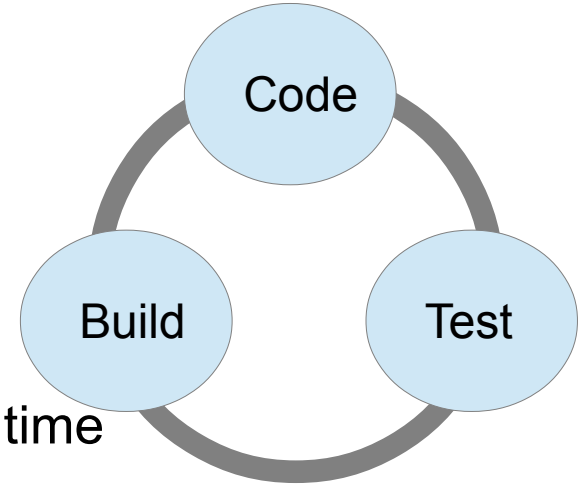
Tools Product Manager, Oracle

Program Agenda

- What is CI?
- What Does It Mean To You?
- Make it Hudson
- Evolving Best Practice
  - For Developers
  - For Build Engineers

What is CI?

- Continuous Integration is a Software Development Process
  - Frequent Integration
    - Code Repository
    - Automatic Build
    - Automated Tests

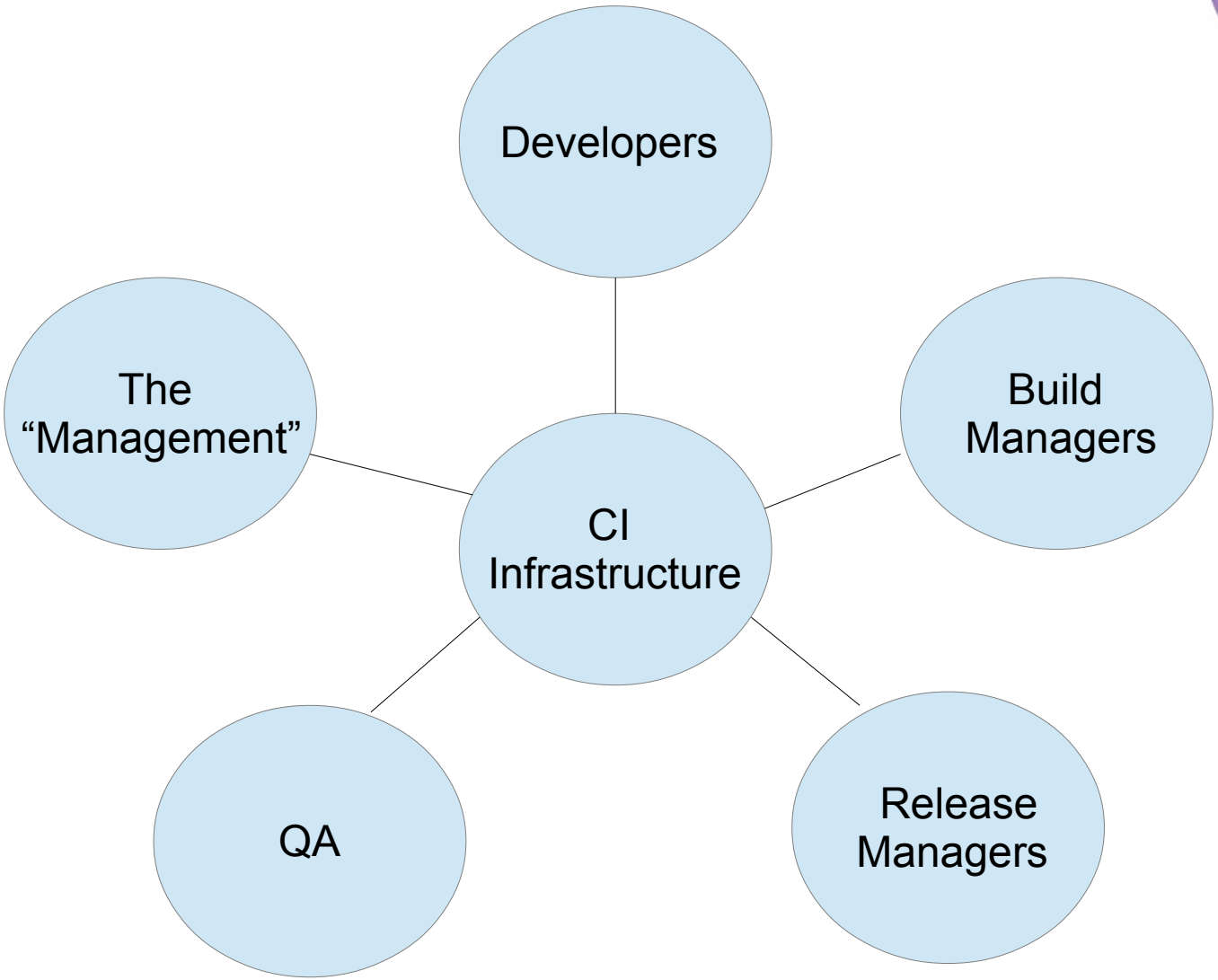


- Discover errors more quickly thereby reducing fix time
- Makes the project available to everyone to install
- Everyone is involved and takes a stake in maintenance

What Does CI Mean to You?

Diff'rent Strokes

Who uses CI?



What do They all Want …?

It's not simply build automation

- Yes it needs to be agile
  - Iterative, incremental, collaborative, rapid, flexible
- We seek improved development process through
  - Better builds, faster
  - Better testing
  - Better visibility
  - Better feedback
  - Better metrics
- Hudson provides the opportunity to grow beyond builds

Introduction to Hudson CI

The Project(s)

- Hybrid Eclipse / Java.net project
  - Eclipse / MIT licensed
- Hudson 2.2.1 (production)
  - Java.net project accessed through hudson-ci.org
- Hudson 3.0 (currently at release candidate)
  - Eclipse Foundation - top level technology project
  - www.eclipse.org/hudson
  - Plug-ins still through java.net

Hudson Has All the Right Bits

So it's a great place to start

- Runs automatically
  - SCM: SVN, Git, CVS, Perforce …
  - Builds: Ant, Maven, scripts …
  - Integrate tests: JUnit, Selenium, Abbott…
  - Deploy: files, App server…
- Feedback via
  - Email, IM, RSS feeds …
- Analysis
  - reporting, graphs: Clover, Sonar, PMD …

Introduction to Hudson

The Basics

- Easy installation and configuration
  - Runs from the WAR out of the box
  - You only need a JDK installed
- Web-based management interface
- Extendable through plug-ins
  - Maintained outside Eclipse
  - Tiered

Hudson 3.0 at Eclipse

What's Changed?

- IP cleanup
  - Core 63MB/109 jars in 2.2.1 → 26MB / 84 jars in 3.0
  - Required extensive core re-write in places
  - Wholesale technology replacement in some cases e.g.
    - Winstone → Jetty
    - JFreeChart → Birt
- Some limited new features
  - Switchable update centers
  - Improved plug-in management
  - Bootstrap configuration

“It is really easy to set up a sever that is very useful but can go away at a moments notice – oh, and present a very large hole in your networks security…”

Gerard Davidson

Long suffering CI user

Getting There – Through Best Practice

Focus First on the Developer

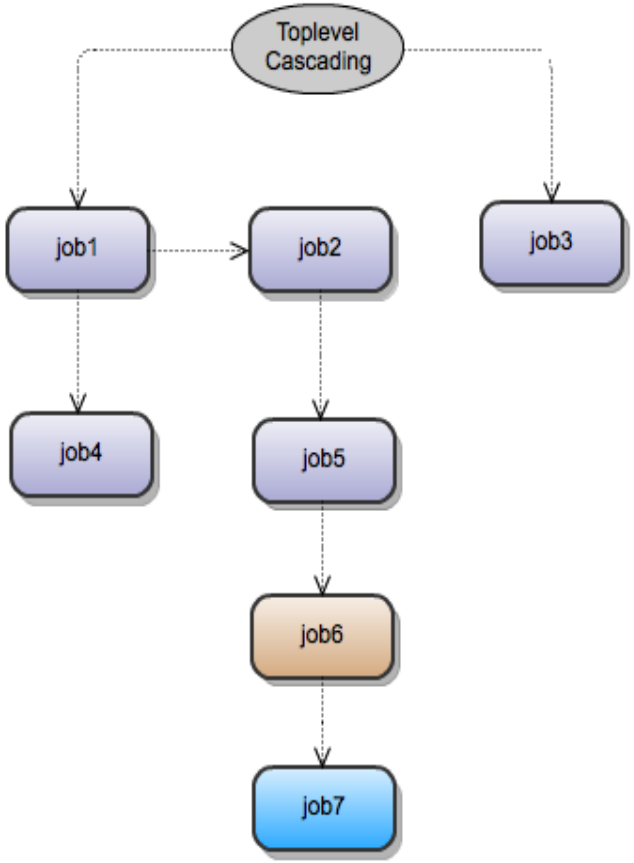
What do they care about?

- Time & convenience
- Make their life better with smaller, faster builds
  - Don’t try and build / test the whole world on every check-in. Make it modular
  - Stick to sanity tests on check-in, full re-build and regression nightly or manually
- Notifications!
  - Tell ‘em when it’s failed – email, jabber, RSS, lava lamps
  - Have the right identity from VCS

Cascading Jobs

Introduced in Hudson 2.2.0

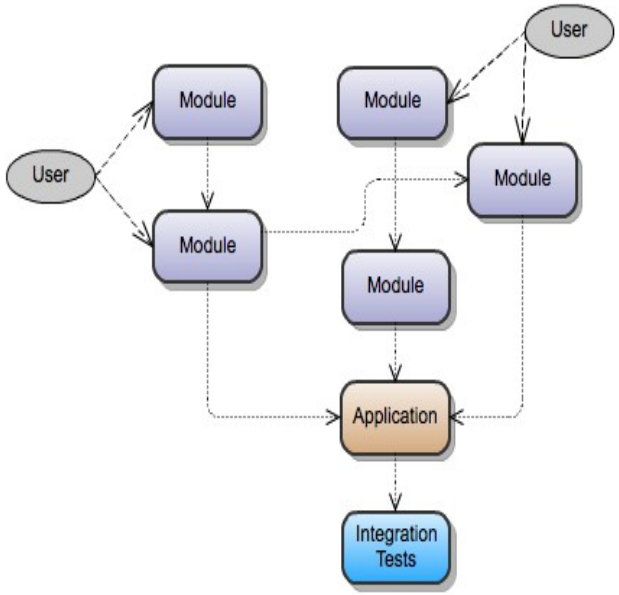
- Provides job template with standardized properties
- Inherited by dependent jobs (and the downstream chain)
- Over writable on a per-job basis
- Inherited unless overridden



Modular Jobs

Don’t do it all at once…

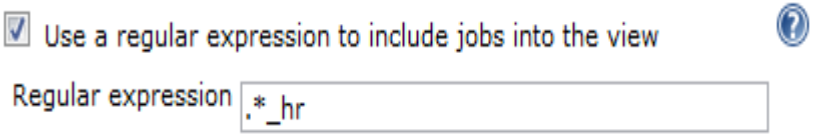
- Job definitions in Hudson are flexible so take advantage
  - Parameterized builds for boilerplate tasks D.R.Y.
  - Upstream and downstream job dependencies
  - Build pipeline plug-in
  - Join plug-in – great for dependency “diamonds”
    - E.g. ensure DB is seeded and App server configured (in parallel) before triggering deployment
- Simpler jobs are better!



Job Management

What's Important

- Put job intelligence into parameterized scripts
  - Put them in source control
  - Can be executed externally from Hudson
  - Hudson jobs become much simpler (which, as we've said, is better!)
- Use smart job naming that can be used with regular expressions to simply / predictably create views
- Just show me the jobs I care about!



Focus on Build Engineering

Promoting Hudson from utility to critical infrastructure

Don’t wake up one day and suddenly discover how much you depended on your CI – plan it!

- You need to manage
  - Metadata
  - Resources
  - Security
  - Infrastructure
  - Upgrades

Managing Your Metadata

Your jobs ARE your CI

- Backup, backup, backup
  - Several plug-ins for this: Backup, thinBackup
  - Move / offsite the backup images
  - Like any backup regimen – test that it works (before you need to)
- Record job history
  - JobConfigHistory plugin – useful when several folks are configuring
- Treat your CI like a production system
  - Test first, then promote into “production”

Resource Management

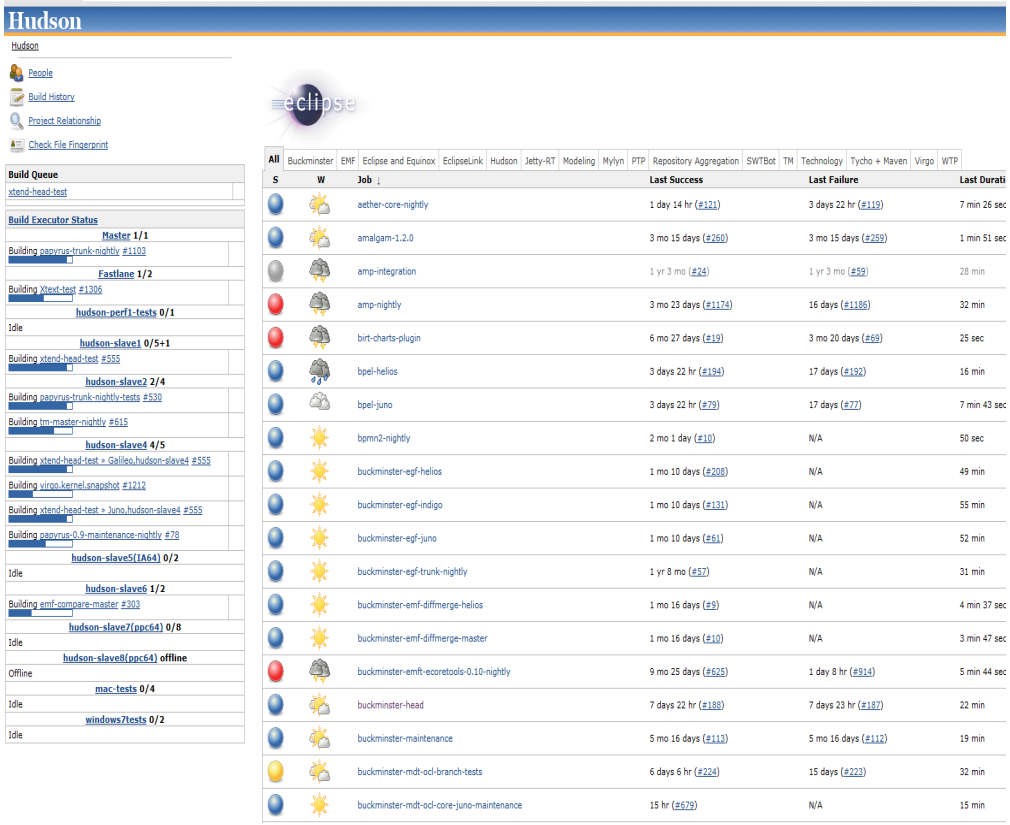
Nothing is infinite

- History
  - Restrict job history depth – improves startup, reduces memory
- Disk space
  - Only fingerprint when necessary
  - Start with one Maven repository per executor to give the best balance between concurrency and space usage. Refine as needed
- Run with sufficient memory allocated to the application server VM

Case Study – hudson.eclipse.org

Lessons Learned

- Keep no. builds restricted
- Promote builds you want to keep
- Restrict debugging info in tests and scripts
- Manage artifacts externally
  - Copy Artifacts plug-in



Security

Hey I trust everybody – don't you?

- Out of the box Hudson is open
  - This may be OK but probably isn't, particularly in larger enterprises and servers accessible via the internet
  - Future direction will be "secure by default"
- So yes, secure your Hudson, please!
- Consider the danger areas
  - Operating system script execution
  - REST API – disable it if you don't need it
  - Credentials passed to scripts
  - Open HTML fields in Job descriptions etc.
  - DOS attacks

Security

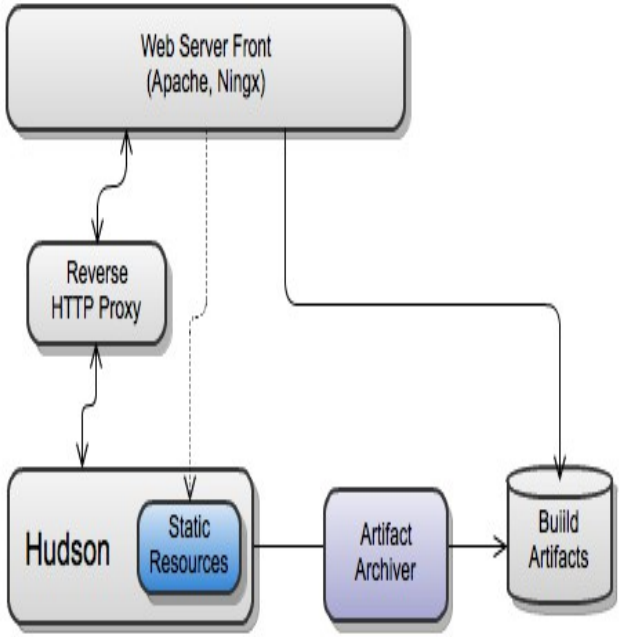
Hey I trust everybody – don't you?

- Out of the box Hudson is open
  - This may be OK but probably isn't, particularly in larger enterprises and servers accessible via the internet
  - Future direction will be "secure by default"
- So yes, secure your Hudson, please!
- Consider the danger areas
  - Operating system script execution
  - REST API – disable it if you don't need it
  - Credentials passed to scripts
  - Open HTML fields in Job descriptions etc.
  - DOS attacks

Typical Practice

Your needs will vary

- Enable security!
  - + switch on “Prevent Cross Site Request Forgery exploits”
- Hand off responsibility for authentication
  - You’re a build engineer not a security officer
  - Use LDAP or container security
- Upgrade from Winstone/Jetty & front with Apache
- Think about resource caching



Infrastructure

K.I.S.S

- Start simple – no need to create a build grid on day-one
- Take advantage of plug-ins, but only load the ones you need and standardize on them.
  - Consider control through a private plug-in site (Hudson 3.0)
- Consolidate on a single primary O/S
  - Only use multiple O/Ss for operating system specific tests and packaging
  - Again, think how long it will take to recover from a disaster

Infrastructure Cont.

What else?

- Keep an eye on free space
  - A filled disk is one of the most common causes of problems
- Maintain identical environments across the farm
  - Ant, JDK etc.
- Consider an infrastructure management methodology / tool
  - VM snapshots, Puppet, Chef etc.
  - You've automated your build, so now automate your infrastructure!

Upgrades

The curse of the easy upgrade…

- Upgrades should not be approached lightly, given that:
  - a) Hudson is critical infrastructure in your org, and
  - b) Every Hudson install is different
- Test it first
  - Set up a parallel instance
  - Test (and possibly upgrade) your plug-in mix
  - Use of cascading jobs and parameterized scripts can really make this simpler

Summary

What we think

- Treat Hudson like any other critical infrastructure.
  - Manage
  - Plan
  - Backup
- Simpler is better for your individual build jobs
  - Hudson is not a automation DSL
  - Externalize complex tasks into scripts
- Use the available plug-ins

Thanks To…

- Winston prakash
- Duncan mills
- Steve Christou
- Henrik Lynggaard Hansen
- Geoff Waymark
- Manfred Moser