Practicing Continuous Delivery using Hudson

Winston Prakash
Oracle Corporation
Development Lifecycle

Typical turn around time is 6 months to 1 year

Sprint cycle is typically 2 weeks to 4 weeks
Typical Sprint Cycle

Sprint 1
- Plan
- Develop
- Test
- Stage

Sprint 2
- Plan
- Develop
- Test
- Stage

Sprint 3
- Plan
- Develop
- Test
- Stage

Reasonably working product

Product Backlog → Sprint Backlog → Sprint → Working increment of the software

24 h
30 days
Further Thought Process

- There are some greedy people out there
- They can’t wait until the end of the sprint cycle to get a working product
- They want a working product on every commit

Thus the **Continuous Delivery** concept was born
What is Continuous Delivery?

- A set of practices and principles aimed at building, testing and releasing software faster and frequently.
- Produce a deployable-to-production build regularly, probably on each commit.
- Every build is a potential release.
Commit to deploy

In a Test Driven Development build pipeline, Continuous Integration is the first step and the end result is the Continuous Delivery.

While Continuous Delivery promotes the concept of keeping your product in a deliverable state on each commit, Continuous Deployment takes it further. On each commit, the deliverable can be deployed to a production environment.
Typical Hudson CI Server Usage

Hudson is mostly tuned to focus on development teams.
Build Pipeline

- Commit
- Build and Test
  - Unit tests
  - Static code coverage
  - Packaging
  - Integration tests
  - UI tests
  - Performance tests
  - Regression tests
  - Deployment tests (install, uninstall etc.)
  - Manual exploratory tests
  - Regulatory, compliance checks
  - Clearance from UAT
- Stage and Deploy
Setting up Hudson to do Continuous Delivery

Identifying the relevant plugins and configuring the jobs to participate in the pipeline is critical.
Setting up CI Environment

- A Centralized SCM repositories (Git, SVN, CVS etc)
- Dedicated build servers
- Continuous Integration software (Hudson)
- Unit testing framework (jUnit, nUnit etc)
- Build tool (Maven, Gradle, Ivy, Ant etc)
- Deployment environment (Application Servers)
- Build Dashboard (Hudson UI)
- Communication tool (E-mail, twitter, IRC etc)
- Deployment Tool
Effective practicing of CI

- Maintain a Single Source Repository.
- Automate the Build
- Make Your Build Self-Testing
- Everyone Commits To the Mainline Every Day
- Every Commit Should Build the Mainline on an Integration Machine
- Keep the Build Fast
- Test in a Clone of the Production Environment
- Make it Easy for Anyone to Get the Latest Executable
- Everyone can see what's happening
- Automate Deployment
Choosing Hudson Plugin for Effective practicing of CI

http://wiki.eclipse.org/Hudson-ci#Hudson_Plugins

Maintain a Single SCM

This principle encourages the project team to use SCM to maintain their source code. Hudson supports the following versions:

- Git
- CNS
- SVN
- Perforce
- Clearcase
- Mercurial

99% of Hudson users use one of the following:

- Git
- CNS
- SVN
- Perforce
- Clearcase
- Mercurial

Hudson supports ~20 additional SCM versions.

Automate the Build

Automating the build using a single change of a CI build. Hudson supports various Frameworks via Plugins. 99% of Hudson users use one of the following:

- Ant
- maven
- gradle
- MSBuild
- Nant
- Rake

Hudson supports ~40 additional build tool versions.

Make your build self-testing

CI build is not about catching errors, but more quickly and efficiently. Hudson supports various Code Coverage Tools via Plugins. 99% of Hudson users use one of the following:

- JUnit
- nUnit
- Selenium
- CppUnit
- TestNG
- xUnit

Hudson supports ~10 additional Unit Test with the following signatures.

Make your build self-testing (Code Coverage)

Self testing is best achieved if there is uniform code coverage. Hudson supports various Code Coverage Tools via Plugins. 99% of Hudson users use one of the following:

- Clover
- Cobertura
- Emma
- Sonar
- NCover

Hudson supports ~2 additional Code Coverage which are used by less than 1% of the users.
Buildable Units

Important guideline of CI is to build fast and give back feedback quickly. To achieve this

- Rather than building the entire source in one single job, divide the project sources into buildable chunks. Each chunk of software must be able to build independent of each other.

- The dependent chunks must be built separately and stored in an artifact repository manager for other software chunks to use them as dependencies.

- Each of the software chunks is a buildable unit and is built by a single Hudson job.
Setting up upstream-downstream builds

Build Triggers

- Build after other projects are built

Set up a trigger so that when some other projects finish, it is convenient for running an extensive test after a build.

This configuration is the opposite view of the "Build on change the other automatically.

Projects names: sherwood_checkout, sherwood_checkout

Multiple projects can be specified like 'abc, def'
Speeding up CI Builds

• The first stage would do the compilation and localized unit tests. The unit tests may be created with out any real time database or server connections to keep it fast. (Mockito, Powermock)

• In the second stage, the extended builds run different suits of tests, may be with real time server and database connections.

Staged Builds
Build pipeline plugin
Cascading jobs

Job Configurations

- Project name: sherwood_checkout_backend
- Cascading Project: sherwood_checkout_top_level
- Description: None, sherwood_checkout, sherwood_checkout_rest, sherwood_checkout_test_harness, sherwood_checkout_webapp
- Discard Old Build: unchecked
- This build is passed: unchecked

E-mail Notification

- Recipients: dev_sherwood_checkout@sherwood_library.org, qa_sherwood_checkout@sherwood_library.org

Whitespace-separated list of recipient addresses. May reference build parameters like $YOURMAIL$. E-mail will be sent when a build fails, becomes unstable or returns to stable.

- Send e-mail for every unstable build: checked
- Send separate e-mails to individuals who broke the build: unchecked
In a CI build, the unit tests should never fail. During the initial stage of the project, the integration test may be in flux.
Monitor Quality Metrics Trend

Code quality measurement is important in Continuous delivery. Improves the confidence of the product being deliverable state.
Automated Upload
(FTP)

As part of Build pipeline, often there may be requirements to copy configuration files or database schemas, test scripts, properties files, install scripts etc., which are part of a build to another machine to facilitate additional test run.
Automated Execution (SSH)

Execute commands on that remote machine to ready the machine for automatic deployment.

- **SSH site**: test-server.mycompany.com
- **Pre build script**:
  - `/usr/local/tomcat6/bin/shutdown.sh`
  - `rm /usr/local/tomcat6/wepapp/test-app.war`
  - `rm -rf /usr/local/tomcat6/wepapp/test-app`
- **Post build script**: `/usr/local/tomcat6/bin/startup.sh`
Join plugin triggers a job after all the downstream jobs are completed in parallel. This allows a pipeline to branch out to perform many steps in parallel, and then run another job after all the parallel jobs are finished.
Automated Promotion

Though promoted build plugin provides opportunity to promote every build, typically the promotion process is done for a pipeline.
Part 1: Practicing effective CI using Hudson

Part 2: Hudson plugin development

http://www.amazon.com/Hudson-Continuous-Integration-Practice-Burns/dp/0071804285