Architecture Survey

Name of system	Chapter 9: Continuous Integration (Hudson/Jenkins) *****
Reviewer	Zhiyuan Wu
Date	Oct 30th, 2011

§1 Introduction

1.1 *Purpose and Introduction*

Agile software development is founded on the notion of rapid iterative and incremental development. However, rapid development is often wrought with deviations from specifications. To detect and prevent frequent check-ins from causing these deviations, Continuous Integration software is used. It automatically performs repeated builds and testing. As such, CI software allows developers to become aware of and take action upon errors as soon as new code is submitted.

CI software can be used to facilitate integration and functional testing. It can also used to conduct cross-platform testing. Furthermore, it can also be used to conduct slow-running and data intensive tests. Continuous Integration systems shine in these situations because these forms of testing are often difficult to perform on an individual basis.

In addition to the above standard features, Continuous Integration software sometimes sports many different plugins. For example:

- Deployment mechanisms
- Bug tracking mechanisms

The Wilson book discusses four CI software implementations:

- BuildBot
- CDash
- PonyBuild
- Hudson/Jenkins

This report will go into detail about the architecture of Hudson/Jenkins, the most widely used CI system in industry. I will motivate the general CI system through architectural examples from Hudson/Jenkins.

1.2 *Book Chapter*

Author of software	**Kohsuke Kawaguchi [Hudson/Jenkins]**
Author of book chapter	**Titus Brown · Rosangela Canino-Koning**
Five star rating of book chapter	**Very valuable. Easy to follow. *******

This chapter very clearly explained the difference in architecture between the four CI systems. It also demonstrated how this difference in architecture matched their usage domain. This book chapter explained the form-follows-function nature of Continuous

Integration software architecture design. This design paradigm is characterized by how each of the four software implementations of Continuous Integration systems

1.3 *History of Hudson and Jenkins*

In 2001, Kohsuke Kawaguchi was employed by Sun Microsystems and started work on Hudson. As agile development becomes more prevalent, Hudson became widely adopted in industry.

In January 2010, Oracle purchased Sun. During this time, the Hudson community wished to move the Hudson repository from Java Net, owned by Oracle, to Github, a large open source repository. Oracle blocked this move. This among other reasons, the Hudson community decided to fork out this project and thus started the Jenkins project. An interesting anecdote is that eventually Hudson, Oracle's fork, eventually also moved to Github.

The difference between these two forks is quite striking. The Oracle branch, Hudson, sports very good documentation. Jenkins on the other hand is more feature rich and moves at a faster pace. Jenkins however, has poor documentation and adhoc release intervals.

These two forks entertain a rare comparison between corporate and true open source development cultures.

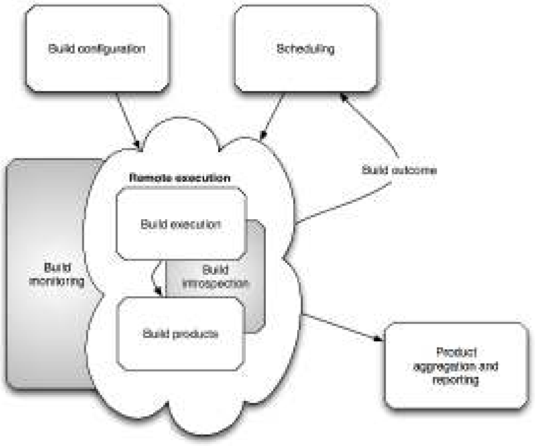
1.4 *Basic metrics*

	Jenkins	*Hudson*
KLOC	**1.3** MLoc	**350** KLoc
Project start-up	**2010** fork from Hudson	**2001**
Number of major releases	**435** (396 since the Jenkins/Hudson Split)	**401** (includes releases before the Jenkins fork)
Number of developers	**524**	**173**
Size of user community or number of installations	**High**, many corporations and software development shops use Jenkins/Hudson for integration testing	
Major stakeholders	**Kohsuke Kawaguchi**	**Oracle**
Use of concurrency	**Simultaneous build jobs** on **multiple slaves**	
Implementation language	**Java**	
Supporting software	**Cross Platform Standalone, requires Java 5 JVM**	

§2 Architecture

2.1.1 *Reference Architecture*

The reference architecture of the general CI system is shown on the right. Boxes are major components. Arrows represent information flow.



All CI systems have a build mechanism that describes and schedule builds. This mechanism forms the core competency. Mostly, they consist of:

- Build configuration
- Scheduling
- Build execution

Independently, they also have various additional systems for:

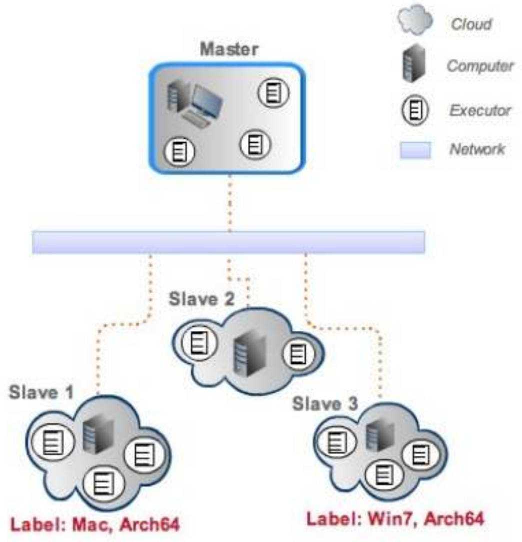
- Build monitoring
- Build reporting

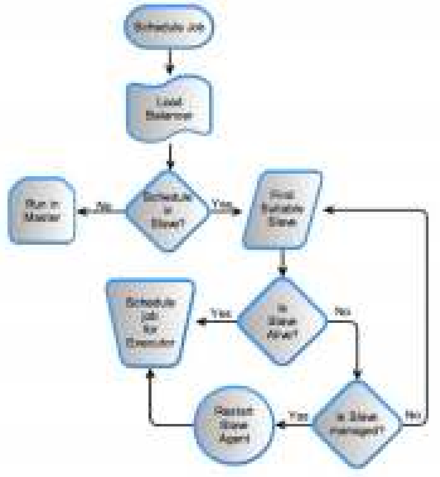
2.1.2 *Distributed Architecture*

In most cases, the build execution, monitoring and report can be distributed to slave nodes. As such, many CI mechanisms use a distributed architecture.

Often, for one software instance, multiple builds may be required. Different builds are required for different operating system support, such as Mac, Windows and Linux. They may be required for different browser support, such as Firefox, IE and Safari.

Much differentiation in CI systems dwells in how distribution control is implemented. For instance, Jenkins and Hudson have a master node that coordinates builds. Section 2.2 describes a high level scenario that demonstrates the master-slave relationship in Jenkins and Hudson.





2.2 *High level scenarios*

Most Continuous Integration systems implement a clear separation of control between the master and slave nodes.

In CDash, control belongs to the individual slavves. The master is merely a centralized reporting server.

Buildbot has a powerful master that produces explicit build scripts for slaves to run. The slaves merely run these scripts as tasks and return the results the master. In this case, control resides in the master.

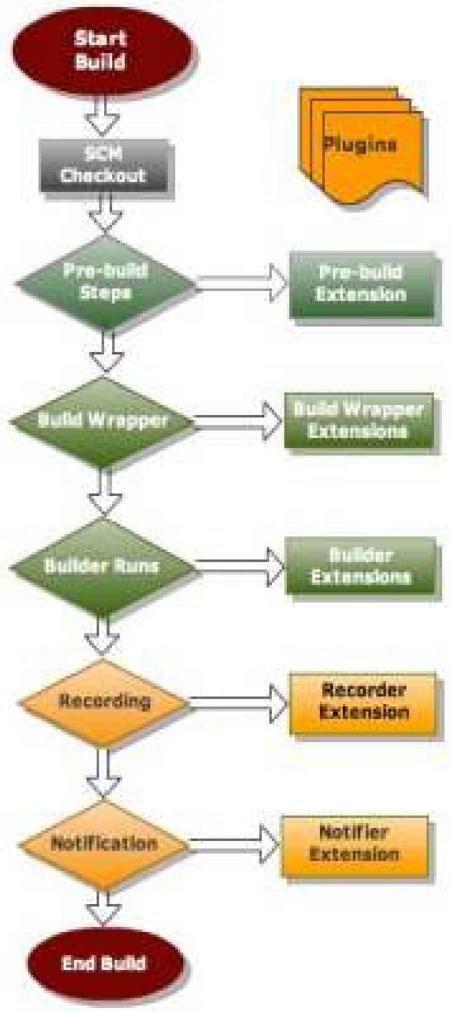
In Hudson and Jenkins, control is distributed. Jobs can be submitted to either the master or a slave. Never the less, the master is capable of querying idle slaves to take part in build jobs. If no slaves are available, the master would do it by itself. The diagram to the left describes the master-slave coordination scenario. This flexible scenario is also true for PonyBuild.

2.3 *Control Flow*

Continuous Integration software implements a pipeline model at the highest level. They always involve:

- Checking code from some code repository
- Building the code
- Running appropriate tests
- Building appropriate build and test metrics

The diagram in the right is the build pipeline for Jenkins and Hudson. Arrows represent information flow. Rectangles represent actions and rhombuses represent decisions.



2.4 *Data structures or algorithms*

An interesting addition to Jenkins and Hudson is its capability to accept plugins. These plugins add capabilities like:

- Multi-language builds (eg: Java, Python, Erlang)
- Multi-platform builds (eg: Mac, Windows, Linux)
- Specific reporting formats
- Additional security layers
- Additional logging mechanisms.

This extensibility has given Jenkins and Hudson a competitive edge over the user specific CDash, feature constrained Buildbot and experimental but unreliable PonyBuild.

§3 Style and Methodology

3.1 *Major evolutionary changes*

CI systems typically have a common architecture. Looking back through the release notes of Hudson, most of the major features have existed since 2002. Each new release merely added bug fixes and feature refinements.

3.2 *Architectural style*
Jenkins and Hudson are written in Java. On a low level, they follow a very OO programming style. At the high level, they follow a pipeline architectural style. This style is illustrated in 2.3 and 2.4.

3.3 *Performance bottlenecks*

The Hudson system is heavily reliant on the master node. The master can easily become a single point of failure. Nevertheless, it is possible to directly query a slave node to process single jobs. However, single slave nodes do not have the capability to delegate tasks the same way the master node does.

3.4 *Real time* [Parts of system critical for fast enough response?]

Since CI systems are essentially processing background builds, latency is not a major issue. As for real time processing, the Master-Slave architecture allows for parallel processing of multiple build jobs. Since builds do not need to communicate with each other or with the master, there is no complicated synchronization mechanism.

3.5 *Notation for architecture*

The notations for architecture are stated within the images shown.

3.6 *Methodology*

For lack of a better term, Hudson and Jenkins both are open source and use an open source paradigm. This methodology involves quick turnover releases. Bugs and fixes are pushed out as soon as they are completed. Features are staged across multiple deliveries. All in all, they use an agile process.

§Appendix: Kruchten's eight context attributes

Stat	**Jenkins (Oracle fork)**	**Hudson (Public fork)**
Size	L = 1MLOC	L = 350KLOC
Criticality	Med = company critical	Med = company critical
Age of System	L=10yr	L=10yr
Rate of Change	Hi	Hi
Business Model	open source	open source
Stable architecture	Lo=stable architecture	Lo=stable architecture
Team distribution	VH=Diff times zones or diff natural languages	VH=Diff times zones or diff natural languages
Governance	Lo=Small intimate team	Lo=Small intimate team