CONTINUOUS INTEGRATION

Outcome

Function

Benefit

Who

Scaling Factors

Difficulty

Continuous integration is a software engineering practice focused on continuous quality checks and incremental merging of developer code. These checks encompass:

- Application/system build, i.e. can the application/system be built successfully with the latest version of the code committed by the developers? This is the core practice of continuous integration. Without a successful build, a system cannot be produced and it is impossible to execute any subsequent quality controls.
- Unit tests, i.e. does the code committed pass all the unit tests? This is a very common and strongly recommended check.
- Static code analysis, i.e. does the code committed comply with the team/company/industry standards?
- Test coverage to control the extent of the code base being tested. This is an optional and less common check.
- Functional tests, i.e. does the code committed pass the automated functional tests?

A fundamental principle behind continuous integration is 'continuous'. The quality checks should be run continuously. Ideally, a build – by build we mean a run of the continuous integration scripts and the analysis of the results – should be run as soon as a developer commits code to the source control management (SCM) system.

A build should be rapid in order to give quick feedback to the development team. To keep the builds fast, a common technique is to create different types of continuous integration builds. Very often, teams have 'day' builds and 'night' builds. The day builds are very fast builds – usually less than 10 minutes – to focus on the quick checks: the application/system build and, often, unit testing. They are run several times a day, normally every time a developer commits code to the SCM repository. When the code does not build, the continuous integration system should immediately notify the developer and their first priority should be to fix the build before any more code can be checked in.

The night builds run the more lengthy checks, usually the static code analysis, sometimes the functional tests, and extra features like an automated generation of a technical documentation from the code. They are run once at night, and the teams get the results in the morning.

Benefits

Why is it so important to run the builds daily (several times a day) and have fast builds? The main reason is feedback. If you don't integrate often and quickly, developers can't have confidence that their code works properly with their colleagues' code. Waiting for that feedback would lower their productivity. In addition, the more time a build takes, the higher the probability that other developers commit code to the SCM repository before the results from the build are published.

If a build breaks, i.e. it fails to pass one or more checks, and more code has been committed, then it becomes more difficult to fix the problem(s). Rapid continuous integration provides fast feedback regarding the quality of the code created, which allows the developers to fix quickly and cheaply most of the issues.

Continuous integration also brings a good visibility to the team of the quality of the code produced. This visibility provides greater confidence to the team and management. Since all tests are automated, they are done in a consistent way.

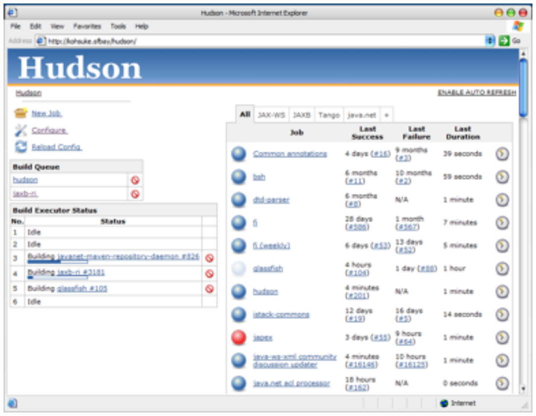
Implementation

Pre-requisites

The main pre-requisite is automation. A fully automated script should be able to build the system under development. Any other check that the team wants to add to the continuous integration (CI) sever should also be automated. All these scripts, automated builds and checks should be first tested on a development machine. Furthermore, the code, the scripts and all the associated data must be under a SCM system.

To build a CI server:

- Define the types of builds needed and their timing.
- Choose a CI tool (e.g. Hudson, Jenkins, CruiseControl, Apache Continuum, TeamCity...) or a CI service (e.g. Tddium, Collabnet...) according to the technology stack and the in-house skills set.
- Integrate build and unit test scripts to the CI tool (depending on the tool chosen).
- Set up the builds and run them (depending on the tool chosen).
- Display the CI results continuously to the team (e.g. screen on a wall presenting the CI dashboard, email reports...).
- Keep an eye on CI results and fix any problem immediately.
- Once the basic continuous integration server is up and running, choose plugins to add extra features to the CI tool. For instance, use of static code analysis tools (e.g. FindBugs, FXCop, PMD...), the measure of the tests coverage (e.g. Emma, Cobertura...), advanced dashboards (Sonar...), automated generation of documentation (Doxygen...).



Build Queue

hudson
jaxb-ri

Build Executor Status

No.	Status
1	Idle
2	Idle
3	Building javanet-maven-repository-daemon #826
4	Building jaxb-ri #3181
5	Building glassfish #105
6	Idle

	Job	Last Success	Last Failure	Last Duration
🔵	Common annotations	4 days (#16)	9 months (#3)	39 seconds
🔵	bsh	6 months (#11)	10 months (#2)	59 seconds
🔵	dtd-parser	6 months (#8)	N/A	1 minute
🔵	fi	28 days (#586)	1 month (#567)	7 minutes
🔵	fi (weekly)	6 days (#53)	13 days (#52)	5 minutes
⚪	glassfish	4 hours (#104)	1 day (#88)	1 hour
🔵	hudson	4 minutes (#201)	N/A	1 minute
🔵	istack-commons	12 days (#19)	16 days (#5)	14 seconds
🔴	japex	3 days (#55)	9 hours (#64)	1 minute
🔵	java-ws-xml community discussion updater	4 minutes (#16146)	10 hours (#16125)	1 minute
🔵	java.net acl processor	18 hours (#162)	N/A	0 seconds

Example of output from a Continuous Integration tool

If you want to learn more, consider reading:
Continuous Integration: Improving Software Quality and Reducing Risk by Paul Duvall, Steve Matyas and Andrew Clover