

Continuous Integration

Diego Casella

February 24, 2015



UNIVERSITÀ
DEGLI STUDI
DI PADOVA

- 1 Introduzione
- 2 Cos'è la Continuous Integration
 - Origini della Continuous Integration
- 3 Principi della Continuous Integration
- 4 Software per la Continuous Integration
 - Cruise Control
 - Jenkins
 - Apache Continuum
 - Apache Gump
- 5 Q&A

In questo seminario affronteremo la pratica della Continuous Integration: introdurremo le motivazioni che hanno portato alla sua formalizzazione, i principi sui quali essa si fonda, e come venga applicata nella pratica

Il concetto di *Continuous Integration*, di seguito abbreviata con **CI**, è una pratica nata dall'*extreme programming* e successivamente sviluppata in maniera autonoma, che si applica nei contesti in cui lo sviluppo software avviene attraverso un sistema di versioning. Essa si propone come una tecnica complementare al *Test Driven Development*, dove infatti si suppone che esistano sistemi di test automatici, che i vari programmatori possano eseguire prima di pubblicare le loro modifiche nel server centrale.

Si consideri il seguente scenario:

- una azienda software composta da venti sviluppatori;
- ogni sviluppatore lavora alla propria postazione, clonando il codice dal repository centrale;
- le nuove feature o fix vengono inviate al repository centrale quando sono state realizzate (dopo "x" ore, giorni, settimane?);

Questo scenario é critico per le seguenti ragioni:

- ogni volta che un repository viene clonato, ed uno sviluppatore inizia a lavorare in esso, tale clone si disallinea con lo stato del repository centrale;
- tale differenza si accresce in maniera proporzionale al tempo impiegato dallo sviluppatore per il suo lavoro;

Questo scenario é critico per le seguenti ragioni:

- quando uno sviluppatore vuole pubblicare le proprie modifiche, incorre in un possibile conflitto dovuto ad altri commit realizzati dai suoi colleghi. Tale rischio aumenta con l'aumentare del tempo trascorso fra un proprio commit, e quello successivo;
- si raggiunge un punto per cui il tempo impiegato per integrare le modifiche altrui, eguaglia o addirittura supera il tempo speso per realizzare le proprie.

Questo scenario é critico per le seguenti ragioni: Si viene dunque a configurare una situazione estremamente negativa per l'attività di sviluppo del programmatore e del ciclo di vita software stesso, che viene comunemente denominata *integration/merge hell*.

Ecco quindi che entra in gioco la **CI**, con lo scopo di mitigare notevolmente gli effetti negativi dello sviluppo di un software da parte di piú entità.

Gestione del codice tramite version control

La pratica del **CI** impone che il codice sia sotto versioning, e lo stesso vale anche per eventuali script, file di configurazione e di build che sono necessari per effettuare una build del programma. Inoltre ci si aspetta che, dopo un checkout dal repository, il codice sorgente sia compilabile ed eseguibile senza errori.

Automatizzazione della build

Per *build* si intende il senso più ampio del termine, ovvero:

- rendere automatica l'operazione di compilazione del codice sorgente;
- rendere automatica la generazione della documentazione, pagine web, pacchetti di deploy e diagnostica finale;

il tutto ovviamente senza l'intervento diretto dello sviluppatore.

Rendere la build self-testing

Ogni qual volta viene effettuata una nuova build, devono essere eseguiti tutti i test presenti nel codice. Ciò assicura che le modifiche fatte si comportino come nei test scritti in precedenza, senza generare errori.

Committare nel repository centrale ogni giorno

Committando regolarmente, ogni sviluppatore riduce il rischio di conflitti complessi da risolvere. Inoltre, la risoluzione di questi conflitti "minori" produce l'effetto di tenere costantemente aggiornati ogni membro del gruppo sul lavoro svolto dagli altri.

Ogni commit innesca un processo di build

Ogni volta che si effettua un commit nel **repository centrale** (svn vs. git), il server stesso deve lanciare una build dell'applicazione per poter determinare l'integrità del codice inviatogli. Se l'operazione non è andata a buon termine, l'errore va corretto immediatamente.

Il processo di build deve essere veloce

Dato che ogni commit deve essere compilato dal server, é necessario che questa operazione sia veloce per permettere a tutti gli sviluppatori di inviare il proprio codice senza aspettare che il server termini la build precedente. Questo principio, perché sia efficace, si basa a sua volta sulla premessa che il software sia *modulare* e che dunque sia necessario, di volta in volta, ricompilare solo una piccola parte di esso per poter verificare l'integrità del suo complesso.

É bene inoltre pianificare delle *nightly build*, che effettuino build da zero, verificandone di nuovo l'integrità complessiva ed eseguendo i test presenti.

Eseguire i test in un clone dell'ambiente di produzione

Se l'ambiente su cui vengono eseguiti i test non é assolutamente identico all'ambiente di produzione, c'è il rischio che qualcosa che é stato testato generi dei bug inspiegabili una volta rilasciato in produzione, con conseguenti danni economici e di immagine.

Semplificare l'ottenimento delle ultime versioni dei pacchetti software

Rendere le ultime versioni disponibili ai clienti o ai tester, riduce il rischio di introdurre funzionalità che non corrispondono a quanto il cliente si aspetta. Di conseguenza si può agire prima e più efficacemente per correggere il proprio prodotto.

Tutti possono vedere i risultati delle ultime build

Ciò é necessario per accorgersi il prima possibile se una build non compila, o sono stati generati errori nei test associati, e dunque sistemarli prima che tali modifiche vengano scaricate dagli altri membri del team di sviluppo.

Automatizzare il deploy

Molti software di **CI** forniscono anche la funzionalità di deploy su di un test server al quale tutti gli sviluppatori hanno accesso per poter effettuare altri controlli più affinati con l'ultima build disponibile.

Anche se tutto ciò può sembrare rindondante e macchinoso, i benefici che si ottengono sono molteplici:

- i bug di integrazione vengono scovati molto presto e, dato che le modifiche sono contenute, é piùsemplice risolverli, facendo risparmiare tempo e denaro;
- si elimina il caos pre-release, dove tutti cercano di integrare il risultato di giorni, se non settimane, di lavoro;
- disponibilità costante di una build corrente che può essere utilizzata per scopi di testing, demo o release;
- i costanti check eseguiti sul software spronano gli sviluppatori a creare codice organizzato e modulare;

Ora che conosciamo principi cardine che costituiscono la **CI**, descriveremo brevemente i software più utilizzati nel campo vedendone i principali pregi e difetti. Prima di fare ciò, le caratteristiche che li accomunano sono:

- una interfaccia per configurare il software stesso, solitamente esposta tramite browser web (certi forniscono anche una gui standalone);
- analisi statica del codice, con tanto di raccolta di statistiche sulla qualità del codice presente;
- integrazione con i (D)VCS maggiormente utilizzati come ad esempio git, svn, bazaar, mercurial, perforce ecc...;
- RSS, invio automatico di email in caso di build errata;
- esecuzione di script customizzati;

Cruise Control é uno dei piú vecchi software per **CI**. É stato sviluppato da Kent Beck e Ron Jeffries, coloro che hanno coniato il termine *Extreme Programmig*, definendone i princípi base.

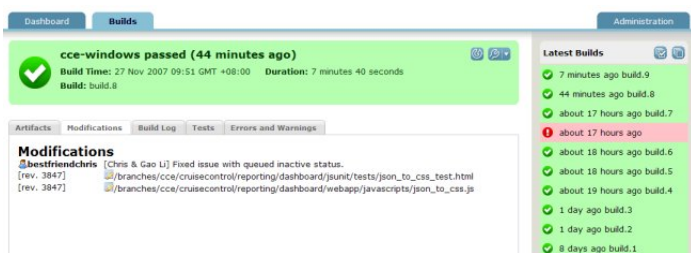
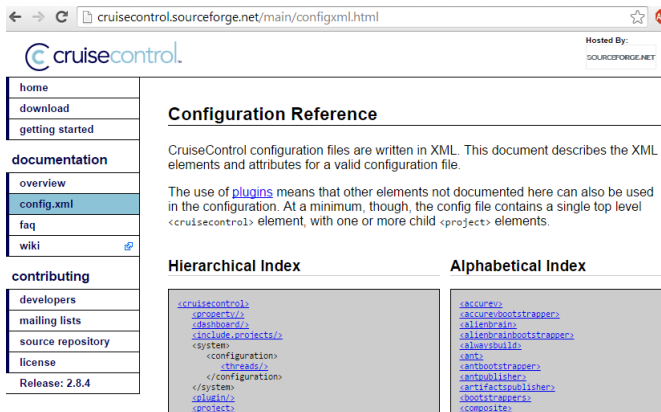


Figure: Cruise Control Web Interface

É stato ideato specificatamente per progetti realizzati in Java, anche se esiste una versione denominata *Cruise Control .Net* che é indirizzata al linguaggio .Net, e una versione *Cruise Control.rb* per linguaggio Ruby . Non presenta una gui per configurarlo: l'intero processo avviene tramite scrittura di un file xml a mano, anche se esistono dei tool di terze parti che facilitano tale setup. Possiede una web UI molto primitiva, e la curva d'apprendimento é molto ripida; tuttavia, una volta installato e configurato opportunamente, si dimostra molto affidabile.



The screenshot shows a web browser window with the address `cruisecontrol.sourceforge.net/main/configxml.html`. The page features a navigation sidebar on the left with links for home, download, getting started, documentation (including overview, config.xml, faq, and wiki), and contributing (including developers, mailing lists, source repository, license, and release 2.8.4). The main content area is titled "Configuration Reference" and explains that CruiseControl configuration files are written in XML. It states that the use of [plugins](#) means that other elements not documented here can also be used in the configuration. At a minimum, the config file contains a single top level `<cruisecontrol>` element, with one or more child `<project>` elements. Below this text are two boxes: "Hierarchical Index" and "Alphabetical Index". The Hierarchical Index shows a tree structure starting with `<cruisecontrol>`, followed by `<property>`, `<dashboard>`, `<include.projects>`, `<system>`, `<configuration>` (containing `<threads>`), `</configuration>`, `</system>`, `<plugin>`, and `<project>`. The Alphabetical Index lists elements in alphabetical order: `<accurev>`, `<accurev.bootstrap>`, `<alienbrain>`, `<alienbrain.bootstrap>`, `<alwaysbuild>`, `<ant>`, `<ant.bootstrap>`, `<ant.publisher>`, `<ant.publisher>`, `<ant.publisher>`, `<bootstrap>`, and `<composite>`.

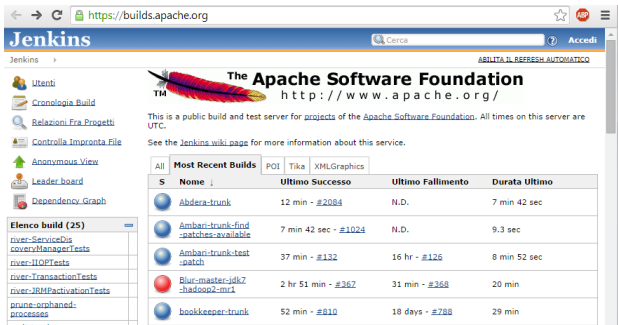
Figure: XML configuration API

Cruise Control é disponibile per qualsiasi tipo di piattaforma, dispone di builders sia per Ant che Maven, dispone di un sistema di notifica tramite email e CCTray, ed é presente un plugin per l'integrazione con Eclipse.

É rilasciato sotto licenza BSD.

Jenkins si é fatto largo negli anni come il software per **CI** nell'ambito di progetti opensource. É nato da un fork del progetto Hudson, sviluppato da Oracle, e si é inseguito sviluppato indipendentemente ed in chiaro, raggiungendo il numero di ben 567 project members (contro i 32 di Hudson).

Jenkins si contraddistingue per una elevata configurabilità e semplicità di utilizzo, unita ad una web interface molto funzionale.



The screenshot shows the Jenkins web interface at <https://builds.apache.org>. The page features the Jenkins logo and a search bar. Below the header, there is a sidebar with navigation links: Utenti, Cronologia Build, Relazioni Fra Progetti, Controlla Impronta File, Anonymous View, Leader board, and Dependency Graph. The main content area displays the Apache Software Foundation logo and a message stating: "This is a public build and test server for [projects](#) of the [Apache Software Foundation](#). All times on this server are UTC. See the [Jenkins wiki page](#) for more information about this service." Below this message is a table titled "Most Recent Builds" with columns: S, Nome, Ultimo Successo, Ultimo Fallimento, and Durata Ultimo. The table lists several builds, including "Abdera-trunk", "Ambari-trunk-find-patches-available", "Ambari-trunk-test-patch", "Blur-master-jdk7-hadoop2-mr1", and "bookkeeper-trunk".

S	Nome	Ultimo Successo	Ultimo Fallimento	Durata Ultimo
1	Abdera-trunk	12 min - #2084	N.D.	7 min 42 sec
2	Ambari-trunk-find-patches-available	7 min 42 sec - #1024	N.D.	9.3 sec
3	Ambari-trunk-test-patch	37 min - #132	16 hr - #126	8 min 52 sec
4	Blur-master-jdk7-hadoop2-mr1	2 hr 51 min - #367	31 min - #368	20 min
5	bookkeeper-trunk	52 min - #810	18 days - #788	29 min

Figure: Jenkins Web Interface

Jenkins é funzionante come Java servlet, e dunque é installabile in qualsiasi macchina che abbia presente una JVM; dispone di builders MSBuild, Ant e Maven, CMake, Grails, Ruby, Scons, Python e molti altri ancora. Integra inoltre sistemi di notifica tramite email, RSS, Google Calendar, bot IRC, XMPP e twitter, ed é integrato attraverso plugin per Eclipse, NetBeans ed IntelliJ IDEA. Inoltre é anche integrato con software per il bug tracking quali Bugzilla, Google Code, JIRA, Redmine ed altri ancora. É licenziato secondo la Creative Commons e MIT License.

Apache Continuum é funzionante, come Jenkins/Hudson, attraverso un Servlet Java. Tuttavia, non supporta Windows builders, e per quanto riguarda Java, é disponibile solo Maven. Prevede l'invio di notifiche tramite email, Jabber e Google Talk, MSN ed IRC. Al momento, non si conoscono integrazioni con gli IDE piú diffusi, né con sistemi di bugtracking.

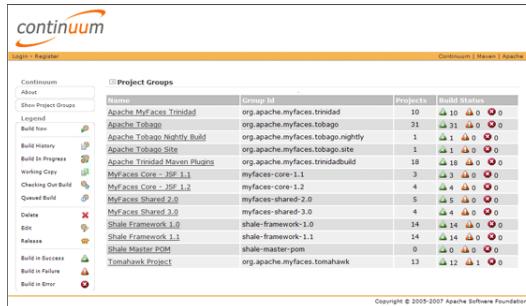


Figure: Apache Continuum Web Interface

Apache Gump é stato il primo sistema di **CI** realizzato da Apache. É scritto interamente in Python, e dunque funziona in qualsiasi macchina dotata di un suo interprete. Gli unici builders che supporta sono Ant e Maven (solo versione 1, la e 3 no) e l'unico mezzo di notifica é tramite email. Non é integrato con IDE o sistemi di bugtracking.



The screenshot shows the Apache Gump web interface in a browser. The address bar displays `vmgump.apache.org/gump/public/`. Navigation links include [Run](#), [Workspace](#), [Log](#), [Issues](#), [Fixes](#), [Pre-reqs](#), [Stats](#), [XRef](#), and [Maven Repository Proxy Log](#). A cartoon bench icon is visible on the right. The 'Run Details' section contains a table with the following data:

Apache Gump(TM) Run GUID	vmgump.apache.org:vmgump:20141013000009
Gump Run (Hex) GUID	B0BA79AE1D0C3321542FB7E4880B9732
Gump Version	2.3
RSS Syndication	RSS 
Atom Syndication	Atom 
RDF Metadata	RDF

The 'Dates/Times' section contains a table with the following data:

@@DATE@@	20141013
Start Date/Time (UTC)	Mon, 13 Oct 2014 00:00:11 (UTC)
End Date/Time (UTC)	Mon, 13 Oct 2014 03:20:36 (UTC)
Timezone	('UTC', 'UTC')
Start Date/Time	Mon, 13 Oct 2014 00:00:11 (UTC)
End Date/Time	Mon, 13 Oct 2014 03:20:36 (UTC)
Elapsed Time	3 hours 20 mins 25 secs

Figure: Apache Gump Web Interface

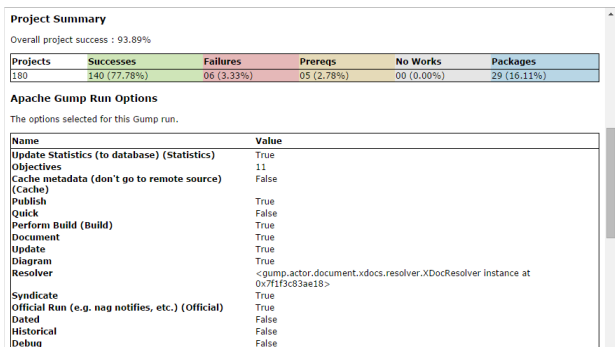


Figure: Apache Gump Web Interface

