Apps Factory

Development Operations – Continuous Integration

Motivation

The following proposal intends to explore the Continuous Integration (CI for short) software development philosophy while also highlighting best practices and tooling that can help startup teams effectively use it. The CI mantra focuses on rapid integration and automation: one should integrate code with the existing code repository as frequently as possible to minimize integration conflicts while also being able to create builds automatically based on some triggering mechanism. This fits well with the needs of a startup since such teams often find themselves facing short product development lifecycles. Other benefits include:

- ✓ Reduction in repetitive manual processes – A typical build process requires some forms of code compilation, database integration, integration testing, inspection, deployment, and feedback. By having these processes automated, this can help startups save time and money and allow developers to focus more on development
- ✓ Release deployable software at any point in time – This is possible since with CI, any small change you make to the source code and integrate, will be done on a consistent basis. If there are any problems, the team is notified instantly and the fixes are applied. Teams that do not embrace this practice will wait until immediately after a delivery of a build to integrate and test the software. This can delay a release or prevent fixing certain defects, which will cause new defects while the team is rushing to complete the product
- ✓ Better visibility – Often, startups begin projects with no previous data with the consequence that it prevents them from noticing trends and making effective project decisions. While data can be mined manually from sources such as SQL queries or Git-metadata, the effort can be quite burdensome. A CI system can provide just-in-time information on builds and quality metrics based on tests that were run with each build

Best Practices – Foundations for tooling

1. Commit Often
    - By following a frequent schedule, every developer can reduce the number of merge/integration changes downstream. Early, small

conflicts in an area of the system encourages team members to be actively engaged about the changes they are making which only improve better understanding of the code base in the long run

2. Every commit to the main product branch should be built
    - The system should build commits to the current working version to verify integration is sound. While this can be done manually, automation can help reduce time/effort. And you're in luck, most CI solutions offer automation!
3. Automate, Automate and Automate
    - Many tools support automation of the continuous integration build process to encourage frequency of code submissions. However, automation also encourages teams to build solid test cases that are repeatable to help confirm that the build behaves as the developer expects it too
    - Most CI systems support plugins and scripts, such as packaging, that are utilized after a build finishes. This way, development can focus on making the product while the actual delivery of the product can be made seamless
4. Communicate Results
    - It should be easy for everyone to see the results of the latest build to get an accurate context of the latest state of the build.

CI Tools

Jenkins/Hudson

Both free and open-sourced under the MIT License, Jenkins and Hudson share the same origins and offer similar features[i]. The differences lie merely in the management of the tools:

- Supported by Oracle, Hudson has fewer frequent releases to Jenkins but is more heavily tested to ensure backwards compatibility. Teams in more enterprise environments might be more comfortable with Hudson given the "enterprise-style" approach to release management

- Hudson favors the use of Maven and/or Nexus for its repository and artifact management. This is different to Jenkins which works well with Gradle and other build system/artifact repositories
- Jenkins relies on the developer community to provide support and this can be found on both their GitHub[ii] and JIRA[iii] environments. Hudson provides professional support from SontaType and Oracle
- According to Cloudbees[iv], Jenkins boasts a far larger amount of plug-ins than Hudson. However, for most developer concerns, both CI solutions offer a wide range of support for the following environments/tools:
    - All distributed version control systems such as Git and SVN. Centralized VCS such as Perforce and ClearCase are also supported
    - Similar support for test suits for Ruby/Rails, .Net and Java development

TeamCity

TeamCity offers its CI solution through a license-based service. Free for small teams, the cost is significant for teams that require more than 20 build configurations and/or 3 build agents. Here's a quick explanation of what this means:

- A Build Configuration describes a set of procedures that are used to create a build. This can include many types of builds such as integration builds, release builds, nightly builds, and others. Build Configurations are usually represented by an .xml file (ie POM.xml in Maven) which contain details such as:
    - Username/Password to access a VCS repository
    - Goals – Set of tasks to be executed such as a script that runs a set of test cases
    - Triggering – This includes the automatic build trigger
- A build agent is a piece of software that actually executes the build process as represented by the build configuration. It can be installed on the same computer as the CI server or on a different machine which is

usually the preferred option for server performance reasons. Some of the steps which an agent will usually perform are the following:

- Use the POM.xml to check out the source code and download artifacts of other builds to run the build process if defined in the configuration
- The number of agents basically limit the number of parallel builds and environments in which your build processes are run

Just like with Jenkins/Hudson, TC has an official GitHub plug-in and can support all the common version control systems. Further, it also supports popular build tools such as:

- MSBuild for .Net
- Ant, Maven, and Gradle for Java
- RVM, Bundler, and Ruby SDK for Ruby on Rails

Where TC stands out is their proprietary development environment and code coverage solutions that integrate well with their CI services. While all three CI tools offer support for Eclipse, a popular IDE for Java developers that offers plug-ins for code coverage and testing analytics, TC offers their own proprietary code coverage apps for each development language including:

- JetBrains dotCover for .Net
- JetBrains RubyMine for Ruby on Rails

---

[i] Comparing http://jenkins-ci.org/changelog and http://hudson-

[ii] GitHub Repository for the Jenkins open-sourced project. Pull requests and latest updates can be seen here: https://github.com/jenkinsci/jenkins

[iii] All tracked/submitted issues for Jenkins can be seen here: https://issues.jenkins-ci.org/secure/Dashboard.jspa.

[iv] Jenkins: Safe Investment Whitepaper cloudbees.com/sites/default/files/whitepapers/Jenkins_Safe_Investment_Final.pdf