April 2014

CONTINUOUS INTEGRATION FOR ENTERPRISE SYSTEMS USING HETEROGENEOUS TECHNIQUES

RAKSHA SV

Department of Information Science Engineering, MSRIT, Bangalore, India, raksha.vjkmr@gmail.com

Follow this and additional works at: <https://www.interscience.in/ijcsi>

Part of the [Computer Engineering Commons](#), [Information Security Commons](#), and the [Systems and Communications Commons](#)

Recommended Citation

SV, RAKSHA (2014) "CONTINUOUS INTEGRATION FOR ENTERPRISE SYSTEMS USING HETEROGENEOUS TECHNIQUES," *International Journal of Computer Science and Informatics*: Vol. 3 : Iss. 4 , Article 6.

DOL: 10.47893/IJCSI.2014.1154

Available at: <https://www.interscience.in/ijcsi/vol3/iss4/6>

This Article is brought to you for free and open access by the Interscience Journals at Interscience Research Network. It has been accepted for inclusion in International Journal of Computer Science and Informatics by an authorized editor of Interscience Research Network. For more information, please contact sritampatnaik@gmail.com.

CONTINUOUS INTEGRATION FOR ENTERPRISE SYSTEMS USING HETEROGENEOUS TECHNIQUES

RAKSHA SV

Department of Information Science Engineering, MSRIT, Bangalore, India

E-mail : raksha.vjkmr@gmail.com

Abstract - Continuous integration is an extensive experience in the Enterprise and system architectures, it is the concept of integrating new code into existing code and then utilizing the testing techniques defined by Extreme Programming. This practice yields units of code that are continually tested during development. Using continuous integration, a programmer integrates new code into existing code after the existing code has been relentlessly built. Therefore, the release of the change is done when it is and everyone knows about the new functionality at the time of the release. While this is dynamic, it should not pose problems since the code is released only after thorough testing. This particular idea generates much discussion among the software practitioners, but the result is that it does advocate the thorough building and testing of modules prior to release as its main premise. On this main premise there is no resistance. It also recognizes that the integration of code in the object-oriented paradigm deserves some focus. One of the key requirement of this paper is to develop a common platform to do automated build environment, as the current environment is emulated from decade old build system based on Make and there is no way to track changes made to the system for detecting dependency flaws.

Keywords - Continuous Integration, Mirroring, Virtualization, Computing ,Repository.

I. INTRODUCTION

IT today strives to create an infrastructure that is dynamic, scalable and flexible to satisfy the needs of mission-critical work as well as the development and deployment of new workloads. Business trends are driving a new generation of applications that are diverse in their functions, processing requirements and processing power requirements. Advanced Analytics is fast becoming a staple of business processing trying to take the existing Non-virtualized infrastructure and turn it into a more efficient and highly utilized environment. Without the benefits of virtualization is costing companies an enormous amount of time and money, but still yielding only minimal results. The trend is to use platform-agnostic software. However, that does not always take advantage of the inherent architecture of the platform, such as symmetric multiprocessing systems or workloads that require optimization to meet throughput and processor utilization objectives.

It is currently under a mandate to use resources more efficiently. In the past, IT organizations typically built an infrastructure that tightly coupled application workloads with physical assets, forcing IT administrators to continually perform repetitive manual tasks just to keep the operating environment up and running. In an effort to maintain some control, software is layered alongside the multiple instances of operating systems to enforce security, manage availability, and ensure performance. Then, to further compound the complexity, some applications require unique hardware, software, and skill sets to support specific business requirements.

Fig. 1 : Heterogeneous System Infrastructure

The heterogeneous system infrastructure illustrates the IT dilemma in managing the complexity that business solutions require today. This illustration depicts the platforms necessary to process a single transaction with all the components representing islands of computing. These islands of computing all use different languages and there is little coordination between them.

For business to gain a competitive edge, IT must be able to rapidly deploy servers and server resources to support new multi platform applications. The methodology of sizing the solution, ordering the servers waiting for installation, cabling and testing is changing. Virtualization with a supporting infrastructure can be used to provision virtual servers on existing hardware. Cloning production environment can be very difficult task but virtualization of environments can help.

II. DRAWBACKS OF THE EXISTING SYSTEM

Learning Curve - It is time consuming for any new member to learn the system. All new driver builds will have build issues which have to be resolved manually and no easy way out.

1. Remote Build Machine Storage is generally limited just enough to keep few versions of driver. Increase in team size has impact on storage. Generally there is no Sandbox/Desktop build capability
2. There are no automated tests to run the Junit test cases.
3. No build triggering on atomic commits. No report of code quality/complexity. Ex: Cyclomatic, duplicated code, metrics related to LOC.
4. No component level output bundle /artifact

All these concepts are present as independent entities or some together to achieve the desired competency. The idea of this paper is to put together all the above flaws and bring in the concept of continuous Integration

III. CONTINUOUS INTEGRATION

Continuous Integration is a software development practice where changes are made frequently and are updated to the larger code base on a daily basis. The goal of CI is to provide rapid feedback. The process of frequently integrating one's new or revised code with the current code repository should be done frequently enough that no intermediate window remains between commit and build, so that no errors can arise without being noticed by the developers. Rather than periodically scheduling a build, normal process is to trigger these builds by every commit to a repository.

On all the projects, the difficulty of making changes is directly related to specific design qualities. The most obvious is duplication: when a change we wanted was localized in a single class, it was trivial. When we had to modify similar code over and over, the change was tedious and took a lot longer. Other design qualities also affected our ability to make changes. Simplicity is important. When adding features, we were better off when there was no pre-existing design to handle that feature. Adding code that doesn't exist is easy, fixing someone's preconceptions about a feature first is more costly. The side-bar lists a number of other design qualities that have made our projects easier to maintain and change.

Completely script your build so that it can be run by a headless process such as a CI server and that it doesn't rely on an IDE or similar dependency. You "centralize software assets" by putting everything that is needed to build the software in the version control repository.

Finally, ensure that you can "perform single command builds" without needing to configure this file or that variable, ensuring a smooth integration build.

Adopt CI by determining on where to focus or what to improve by using self check value stream map. Involve and educate the whole team. Management buy in and support to include CI cost/plans in project cost/plan (unit tests, build/deploy tool and scripts)

The advantages of adopting CI:

- Better views into project health
 - Notice trends and then take action earlier
 - Developers have more confidence to refactor
- Stable, working software available sooner and more often
 - Focus is on getting working builds
 - Less escapes, better product for validation testing
 - Early feedback from stakeholders to validate requirements
 - Greater confidence in the product
 - Deployable software is a tangible in short time boxed iterations.

The various continuous Integration engines include Rational team concert build engine, Jenkins, Cruise control and Rational build forge.

IV. TECHNICAL SPECIFICATION OF CI ARCHITECTURE

A brief insight into the working of CI

Fig. 2 : CI Life cycle

Manual build approach is similar to the local build that a developer does before a commit into the repository. The developer goes to the integration machine, checks out the head of the mainline (where his last commit resides) and kicks off the integration build. He monitors the progress, and if the build succeeds he is done with his commit. A continuous integration server acts as a monitor to the repository. Every time a commit against the repository finishes the server automatically checks out the sources onto

the integration machine, initiates a build, and notifies the committer of the result of the build. The committer isn't done until they get the notification - usually an email. Mostly companies do regular builds on a timed schedule, like a nightly build. This is not the same thing as a continuous build and isn't enough for continuous integration. The whole point of continuous integration is to find problems as soon as you can. Nightly builds mean that bugs lie undetected for a whole day before anyone discovers them. The whole point of working with CI is for developing on a known stable base.

V. TECHNIQUES FOR IMPLEMENTING CI

- Jenkins for build
- Mirroring repositories using Subversion

JENKINS – open source CI server Jenkins, is an open source continuous integration tool written in Java. Jenkins provides continuous integration services for software development, primarily in the Java programming language. It is a server-based system running in a servlet container such as Apache Tomcat. It supports tools including CVS, Subversion, Git and Clearcase, and can execute Apache Ant based projects including shell scripts and Windows batch commands. Builds can be triggered by commit in a version control system, scheduling mechanisms like cron jobs, build when other builds have completed, by requesting a specific build URL.

A. Why choose Jenkins

Jenkins improves the Productivity by detecting build breaks sooner, report failing tests more clearly and makes progress more visible. Continuous Integration can be done using Jenkins as it supports convention over configuration: no setup or minimal configuration required to build artifacts, Multi-technology, multi-platform, extensible environment.

1) Implementation

Jenkins provides a user friendly web user interface (UI) where you can access information like build number, last updated files, build artifacts and test reports. You can create continuous integration jobs and configure relative properties, such as SVN source location, schedule, and e-mail notification settings through this web UI. A build process requires a build tool such as Ant, build invoker and some shell scripts. It can be scheduled or done manually using the invoker. Before Jenkins starts to build and execute the regression bucket, it first checks out the latest version of source programs from the SVN repository. The mechanism can be configured either using Update or Revert from the repository, and it will record current version of sources within every Run. Jenkins tool is installed in which a new job can be created to include the projects artifacts which is stored in the job directory. The build script that performs the build (ant, maven, shell script, batch file) is where the real work

happens. The environment variables such as version, build.xml file location has to be set corresponding to Jenkins server. Source code resides in a CVS or SVN repository. The developer checks out the code to the workspace in Jenkins along with the build invoker-build.xml file to Jenkins/jobs/PROJECTNAME/ workspace. Jenkins can poll the revision control system using the same syntax for crontab in linux. If the polling period is shorter, there could be multiple builds for each change. Thereby adjust your polling period to be longer than the amount of time it takes to poll your revision control system, or use a post-commit trigger. You can examine the Polling Log for each build. Optional steps to notify other people/systems with the build result, such as sending e-mails, IMs, updating issue tracker.

Frequently categorize Developer Tests, use a Dedicated Integration Build Machine and Use Continuous Feedback Mechanisms. Continuous integration using Jenkins is easier to monitor and can be refactored depending upon the nature of the project and the team size.

B. Mirroring Repositories Using Subversion.

Mirroring

The primary goal of mirroring is to provide a robust, economic replication solution. Traditional replication solutions often require substantial capital investments in infrastructure, deployment, configuration, software licensing, and planning. The solutions that rely on shared resources are susceptible to single point of failure for that shared resource. Combining continuous integration with mirroring provides an additional level of availability, thus greatly minimizing work flow and user disruption. Configuring the two mirror members in separate data centers offers additional redundancy and protection from catastrophic events. By utilizing logical data replication, mirroring reduces the potential risks associated with physical replication, such as out-of-order updates and carry-forward corruption, which are possible with other replication technologies such as SAN-based replication.

In this paper we explore two ideas of mirroring using the subversion.

- Mirroring the multi-site servers
- Mirroring using a dump

1) Mirroring the multi-site servers

Mirroring is necessary for synchronization and backing up of data. Create a master-slave mirror setup of the desired server which houses the organizations data. The primary/master servers configuration will be managed by the repository administrator using the svnadmin, providing the ability to create subversion repositories and perform several maintenance operations on those repositories.

The architectural representation is given below

Fig. 3 : Mirroring Repository-Flow of Actions.

The primary/master repository is where all the project related artifacts resides. The latest driver releases lie in the head of the repository. Hierarchy of the repository is given by the svnadmin. Here We can treat the master repository as any other file system and recursively copy it to the mirror location. This is not ideal if the repository is in use -you're copying a moving target - so you will have to take down the subversion server while making the mirror. If this downtime is acceptable, netcat, nc, combined with tar is a neat way to recursively copy a directory across a network connection using TCP/IP onto the slave repository. This code synchronization will happen at regular intervals to update the master with all changes made to the slave. The customized repository is used as a sandbox where all the deliverables can be accessed from the slave and modifications can be made right here without affecting the master repository. Build process can be triggered through Jenkins which is associated with Sonar which is a Quality profiling tool used for code quality management such as unit tests, coding rules, duplication's. MySql is used for querying. The organizations infrastructure consists of an Authentication Engine and SMTP server. Authentication engine allows authenticates user to access the build process and SMTP server to notify stakeholders of successful or failed builds.

The main advantage of mirroring is that the master repository remains untouched and refactoring is made by taking advantage of the slave repository, when changes are made unknowingly it does not affect the

primary repository and can be handled at the slave repository level.

2) Mirroring using Dump

Another way to mirror a subversion repository is to combine svnadmin dump with svadmin load.

svnadmin dump PATH_TO_REPOS | svnadmin load PATH_TO_MIRROR

Svnadmin dump is designed to create a portable repository dump. The resulting dumpfile can be loaded into a new subversion repository-even if the new repository is using a different database back-end, or even a different revision of subversion. Svnadmin dump will happily run on a live repository (no need to take the server down). In short, combining svnadmin dump with svnadmin load is probably more powerful than we need if we just want to mirror our repository to a new location. Svnadmin dump - on its own - is the best way to fully backup a repository, since the dumpfile it creates is portable (as described above). If we replicate a repository by piping svnadmin dump to svnadmin load, we lose the dumpfile in the pipeline and do far more work than we need to. svnadmin dump does not dump your repository configuration files and hook scripts. If your backup strategy is based around these commands, you will need separate arrangements for backing up hook scripts and configuration files.

VI. RESULTS

Time/Cost investment => Overall Time/Cost savings and higher product quality

In Continuous Integration executing test is all part of build and build fails if the test fails, XUnit ,JUnit, NUnit – for unit testing purpose. From an efficiency perspective this increase in development time is offset by the by the reduced maintenance costs due to the improvement in quality. For each new deliverable driver received from development team daily, one person hour in configuring testing environment is saved, checking out the latest sources and starting execution. It's a common understanding that for software development; the earlier a defect/error can be detected and fixed, the lower cost it is. E-mail notification allows timely response to errors and defect. Individual tester is now notified when his or her own programs has affected the whole integrity of automation framework. The effort for fixing errors in the regression bucket is now distributed to each tester, and whole team productivity is increased.

VI. CONCLUSION

Continuous Integration is about delivering value predictably .In terms of software quality, continuous integration can help measure cyclomatic complexity, code duplication, dependencies and coding standards so that developers can proactively refactor code before a defect is introduced. If a defect is introduced into a

code base, CI can provide feedback soon after, when defects are less complex and less expensive to fix. Also, when using an effective developer testing regimen, CI provides quick feedback, via regression tests. CI acts as a mirror of your software under development.

Research in pedagogical patterns for implementing pair programming and grading the results of such activities are still emerging. With applying Jenkins to realize the continuous integration approach in Agile process, we are now able to automate and facilitate the process of executing regression testing. The continuous integration approach in Agile gives us better chances to identify programs causing integration errors as early as possible, and secures the product integrity anytime in a regular scheduled time frame in each sprint

REFERENCES

- [1] [http://en.wikipedia.org/wiki/Hudson_\(software\)](http://en.wikipedia.org/wiki/Hudson_(software))
- [2] <https://wiki.jenkins-ci.org/display/JENKINS/>
- [3] http://docs.intersystems.com/cache20121/csp/docbook/DocBook.UI.Page.cls?KEY=GHA_mirror
- [4] <http://svnbook.redbean.com/en/1.7/svn.reposadmin.maint.html#svn.reposadmin.maint.replication.pre-revprop-change>
- [5] <http://martinfowler.com/articles/agileOffshore.html>
- [6] <http://www.redbooks.ibm.com/redpieces/abstracts/sg247921.html?Open>
- [7] <http://martinfowler.com/articles/continuousIntegration.html>