Pipeline Orchestration for Test Automation using Extended Buildbot Architecture

Sushant G.Gaikwad
Department of Computer Science and engineering,
Walchand College of Engineering,
Sangli, India.

M.A.Shah
Department of Computer Science and engineering,
Walchand College of Engineering
Sangli, India.

ABSTRACT

Whenever developers do any changes into code base of software, they want to see effect of changes quickly, Continuous Integration can solve this problem. In recent software development where agile methodology is followed turnaround time for software is less, this leads to availability of less time to develop, build and test processes. Test Orchestration methodology can be used to reduce testing efforts and time. Test Orchestration in broad sense is a tests execution step by step in automated fashion, where different type of tests will executes like Junit tests, Integration tests, Sniff tests, Acceptance tests and so on. Here tests are selected dynamically based on developer's check-ins. For Junit tests, we have extended Buildbot architecture which is a master/slave architecture, with a single central server and multiple build slaves. The objective of this paper is to develop distributed architecture for test orchestration. All higher level tests like functional tests, sniff tests and performance tests will execute in pipeline, finally result of all tests will send to developer through email.

Keywords

Continuous Integration (CI), Agile Methodology, Test Orchestration, Dynamic test Selection, Deployment Pipeline (DP), Pipeline Test Orchestration, Business works (BW).

1. INTRODUCTION

Software is useful to customer only if it is deployed in production and provide necessary functionalities in right time. In continuous Integration every check-in needs to be verified, to integrate new checked-in code into entire code base. This integration requires build to be performed and exhaustive testing on new build. Continuous delivery also requires frequent testing of different types. We want to reduce manual efforts required for testing and test engineers to focus on key areas. Also to detect problems early in development we need to speed up testing. Benefit of CI is to reduce time lag between development and testing.

A long standing problem for software development teams has been to maintain the stability of an application while integrating the changes made by multiple developers. Later the integration is delayed the more potential there is a risk and failure in the integration process. Continuous Integration (CI) tries to mitigate this risk by frequently integrating small changes into an evolving code base [3].

In Continuous Integration members of a team integrate their work as much as possible. Each integration is verified by a build and series of different types of tests. Many teams find that this approach leads to significantly reduced integration problems and allows a team to develop cohesive software more rapidly [7].

Continuous integration requires that for every commit the whole application will built and a set of automated tests will run against it. If the build or test process fails, the development team fixes the problem immediately. The goal of continuous integration is that the software is in a working state all the time [3]. To make this process faster and efficient Test Orchestration is helpful. Test Orchestration involves selecting platform and configuration information on which tests will executes, carefully selecting particular tests to execute in a particular order according to build and finally analysis of tests reports so that developers can identify errors if any.

Continuous Integration (CI) systems are systems that build and test software automatically and regularly. CI systems can simplify and automate the execution of many otherwise tedious tasks like detection of infrequently failing tests, the regular production of up-to-date release products and so forth [2].

CI is often a first step towards a continuous deployment framework wherein software updates can be deployed quickly to live systems after testing [5].

A deployment pipeline is an automated implementation of an application's build, deploy, test, and release process. The deployment pipeline has its foundation in the process of continuous integration.

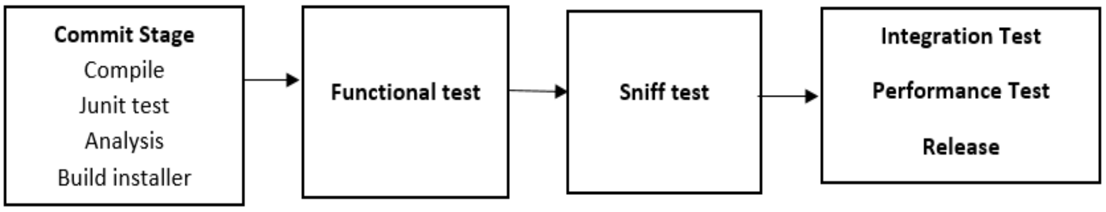

Fig. 1 Deployment Pipeline

The deployment pipeline has three advantages, it makes processes visible and transparent to everyone. It provides early feedback so that problems are identified, and hence resolved, as soon as possible [3]. Finally, teams can have any version of their software for any environment.

Continuous Integration has become a mainstream technique for software development. Hardly any Thought Works projects goes without it [7]. Krishnan develops an economic model to optimize the delivery cycle of delivering good quality software [8]. Lahtela presented the challenges in the release of software [9]. Most organizations are in the process of achieving continuous delivery and it becomes universal standard and CD advocates the creation of maximally automated deployment pipelines. According to “DZone community” survey more than 65% organizations have implemented continuous Delivery [11].

1.1 Overview of problem

A lot of work already done in the field of software testing automation. Many tools like Jenkins, Cruise Control are available in market to automate testing procedure. But the problem with these tools is that none of them provide end-to-end support for build management and automation tasks [1]. Organizations have been orchestrating pipelines with existing Jenkins plugins for several years. With time, organizations want to move beyond simple pipeline and chart complex flows to map to their specific delivery process [10].

The goal of this research is to develop architecture which provides end to end support for automation of testing process in pipeline fashion, which allow developers to trigger testing quickly and supports on demand, triggered or scheduled testing.

2. DESIGN OF EXTENDENT BUILDBOT ARCHITECTURE

2.1 Delivery Pipeline Approach

Software build process consists of following activities

1. Download latest updates of code and related configuration files from svn.
2. Build with latest code (compilation).
3. Setup test beds. Configure test machines with all prerequisite tools and test suites.
4. Executes test suites and store results.
5. Result are aggregated and stored on server.

For above activities organizations already using many tools. Pipeline orchestration requires coupling between number of tools and technologies, and provide more efficiency in work. So pipeline orchestration framework requires to achieve complex and diverse external coordination and needs to

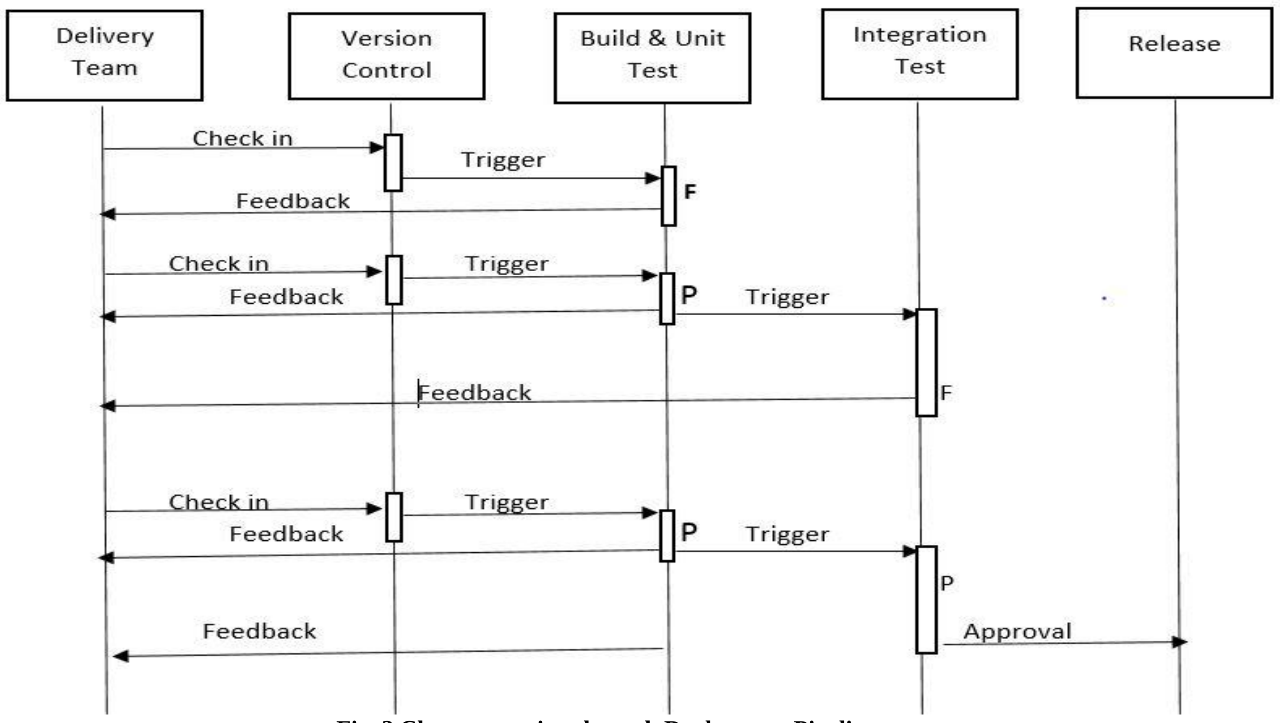

Fig. 2 Changes moving through Deployment Pipeline

The process starts with the developers committing changes into their version control system. At this point, the continuous integration management system triggers a new instance of a pipeline. The first (commit) stage of the pipeline compiles the code, runs unit tests, and performs code analysis. If the unit tests all pass and the code is up to scratch, next level of tests starts execution. We provide a facility to store results of test execution and make them easily accessible both to the users and to the later stages in pipeline. In case of failures process will be stopped at failure point and feedback will be given to respective developer. CI servers will execute these test jobs in parallel. Once the tests are successful the notification will be given to developers.

2.2 Buildbot Architecture

Buildbot uses a master/slave architecture, with a single central server
and multiple build slaves. Master is responsible for managing remote executions.
Configuration specifies the command to be executed on each remote system [2]. Scheduling and build requests are not only coordinated through the master but directed entirely by the master. Buildbot

maintains a constant connection with each build slave, and manages and coordinates job execution between them. Coordinate remote machines with constant connection is complex [2].

Many modern CI tools follows Buildbot architecture but fail to manage complex external coordination required with different technologies and tools. So there is a need to develop a pipeline orchestration framework that fills gap left open by CI tools and mange build slaves.

In purposed architecture, Orchestrator will acts as a Master which directs and controls builds. Orchestrator is the process that schedules builds on different slaves. We call it as a test beds, which have configured to execute a particular type of tests. This is a loosely coupled architecture where different technologies are interact with each other through orchestrator.

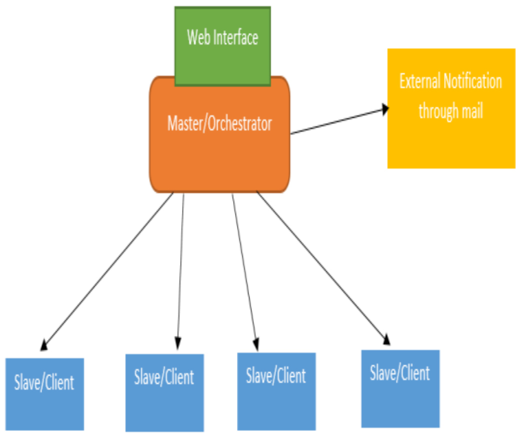

Fig. 3 Buildbot Architecture [2]

2.3 Extended Buildbot Architecture

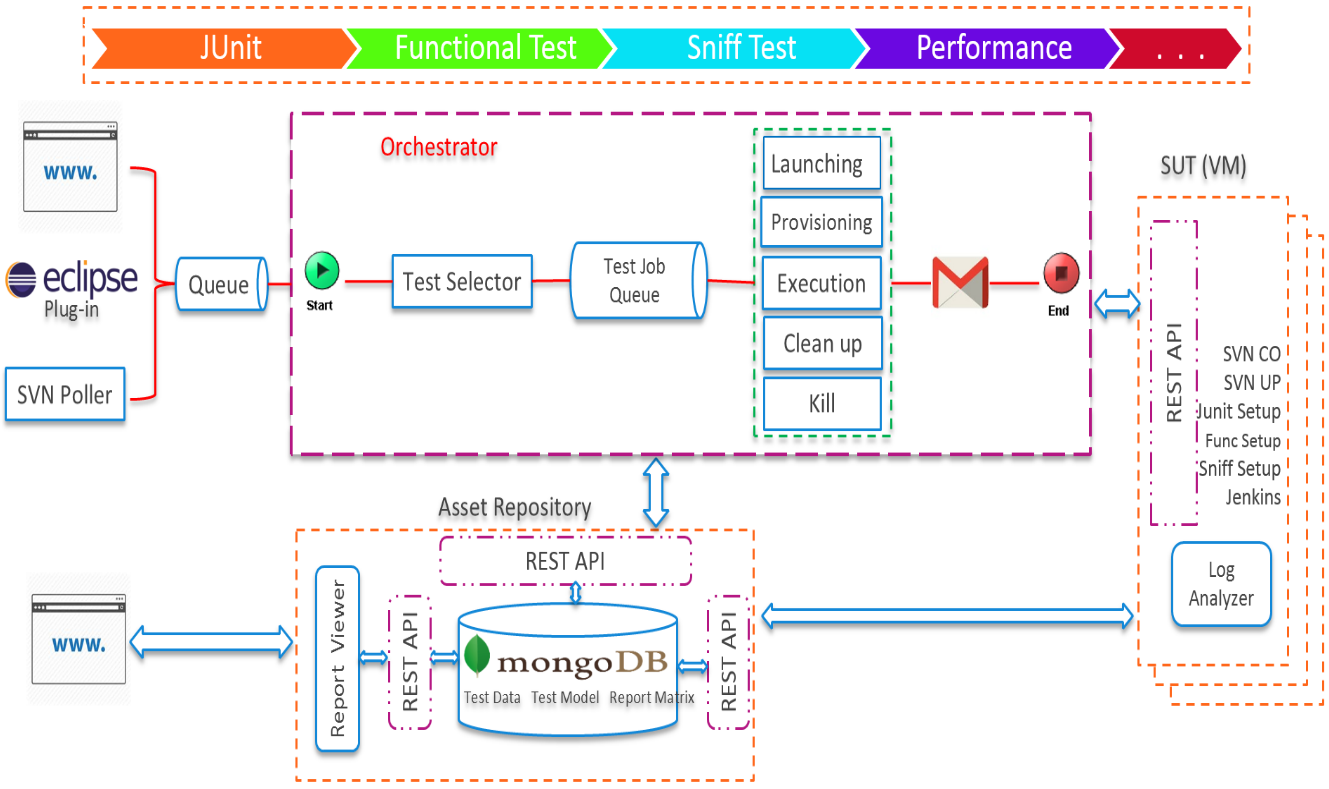

Fig. 4 Extended Buildbot Architecture

So orchestrator will acts as coordinator. For communicating among different components we have used REST architecture. Testing can be triggered either through UI, Eclipse plug-in or svn poller which is a java application. SVN Poller is the Java application that polls from SVN after a fixed time period and comes up with a log containing revision number, user name, code path, and comments then SVN Poller will parse the log. Then poller create a unique id and put it in the JMS/EMS queue and all information in asset repository.

We have different machines and different queues for each type of tests. We keep track of idle and occupied machines.

To execute particular type of test like Junit, if Junit VM is idle then id is retrieved from queue, and puts the id in the test job queue. Based on id orchestrator queries asset repository to get all details for that id. Once the request is put in the test job queue it is then picked and the execution is started.

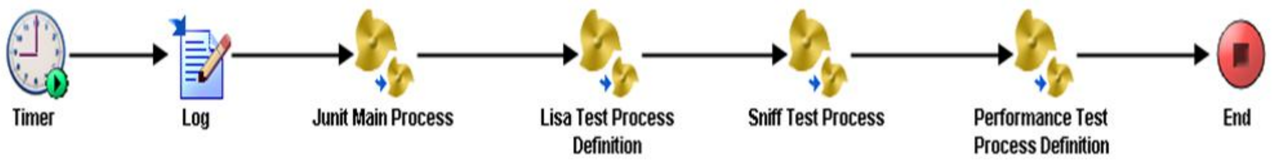
Timer
Log
Junit Main Process
Lisa Test Process Definition
Sniff Test Process
Performance Test Process Definition
End

Fig.5 BW Pipeline process Setup

Following steps has been performed for test execution

- Firstly the product code is updated.
- Jenkins job configuration and Build is done according to data which is fetched form asset repository.
- Jenkins job is triggered to execute the test cases on the new build.
- Once the execution is successfully finished the Log analyzer is triggered and it analyses the test result logs.
- The analyzed logs are then pushed to the asset repository and are maintained for future reference. The report summary mail is then sent to respective developer along with the link to detailed report. Once developer clicks on the link detailed report is shown in the browser.

For each type of test we have designed separate orchestrating process and that are connected to each other in sequence as shown in figure. After completing Junit tests, it passes id to Lisa test process (kind of Functional test) and so on. For designing orchestrating processes of different types, we have used Tibco's Business work where we can design different activity with minimum coding and designing is also very simple. We can use java to design and define flow of processes. Each process has to bring other components together to execute test as shown in Lisa Junit test BW setup. Each BW process may contains number of different processes

3. ANALYSIS OF APPROCH

We have implemented end to end pipeline for Tibco's Active Matrix Service grid product in Tibco Software Pune. For every commit testing is automatically triggered and parsed reports with detailed report link is send to developer. Parsed reports gives exact statistics like number of passed and failed test cases and causes. From which developer can easily detects whether code has properly integrated or has introduced any bugs.

We have taken developers and test engineers feedback in which we have got following information

- No need to configure any environment as it is already done.
- All tests executes one after another without manual intervention, it saves lot of time of developers and test engineers.
- No need to analyse complicated reports as report analyser gives all required information.
- Different types of tests executes parallel and independent.
- Thus all resources has been used optimally and efficiently.

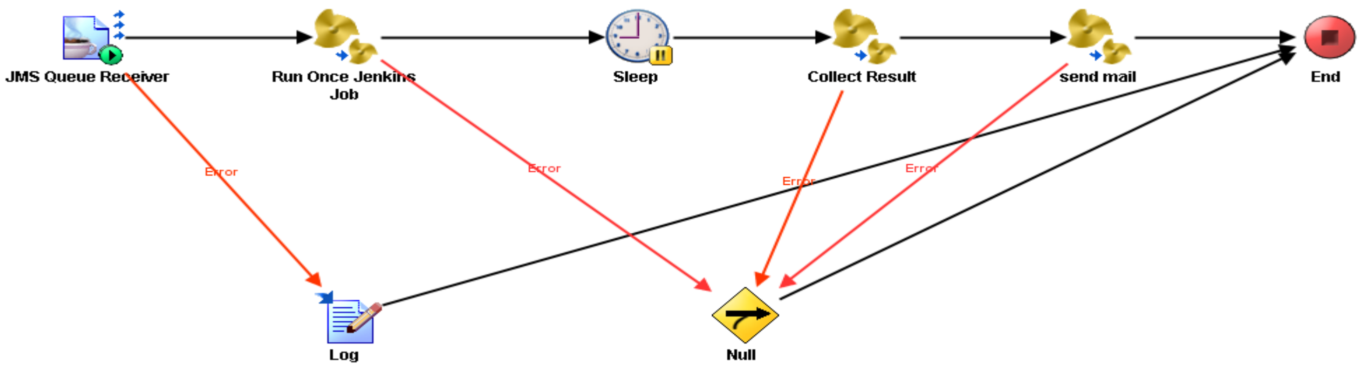

Fig. 6 Functional Test Setup

4. CONCLUSION

We have designed pipeline test orchestration framework that fills the gap left open by many CI tools, it can be integrated with any supporting technology and leverages the testing process. The purposed solution scales well for any number of test types also optimally uses available resources. In future we will improve the log analysis approach and the GUI. Also purposed model can be extended to use cloud where based on requirements environment can be created dynamically and tests will be executed on it, finally environment will be deleted after tests execution.

5. REFERENCES

[1] P. P. Bhanu Prakash Gopularam, Yogeesha C B, "Highly scalable model for test execution in cloud environment," 18th Annual International Conference on Advanced Computing and communication, 2012.

[2] The architecture of open source applications," osabook.org/en/integration.html, August 2014.

[3] J. Humble and D. Farley, "Continuous delivery: reliable software releases through build, test, and deployment automation," Addison-Wesley.

[4] "http://continuousdelivery.com/," August 2014.

[5] http://in.wikipedia.org/wiki/continuousintegration," August 2014

[6] http://www.ibm.com/developerworks/rational/library/contiuous-integration-agiledevelopment/," August 2014.

[7] http://martinfowler.com/articles/continuousintegration.htm" #building a feature with continuous integration, August 2014.

[8] M. Krishnan, "Software release management: a business

[9] perspective" center for Advances studies on collaborative research, June 1994. M. J. A Lahtela, "Challenges and problems in release management process: A case study," IEEE 2nd International Conference on Software Engineering and Service Science (ICSESS), 2011.

[10] Continuous Delivery with Jenkins – Deliver Software more quickly with Jenkins Workflow by "ClodBees Enterprise".

[11] http://www.cloudsidekick.com/blog/pipelineorchestration-frameworks-part-two.html.