Continuous Integration, Deployment and Delivery Automation in AWS Cloud Infrastructure

K. Sree Poornalinga[1], Dr. P. Rajkumar[2]

[1]P.G Scholar, INFO Institute of Engineering, Coimbatore, Tamil Nadu, India.

[2]Professor, INFO Institute of Engineering, Coimbatore, Tamil Nadu, India.

---***---

Abstract - *Agile and DevOps are emerging best practices in the software engineering world which dictates the prominence of Continuous Integration, Deployment and Delivery as one of the essentials in supporting every single software development team. Automating the integration, deployment and delivery of software development is one of the key solution for attaining productive growth in software industries where by which the ideas of agile and DevOps can be turned into practical solutions. Our research shows that in spite of their massive benefits, the automation of these two practices has not yet been significantly inclined with many organizations where software developers struggles to integrate and deploy the code with main lane and different environments respectively. In this paper, we demonstrated a newly structured cloud Infrastructure called "Continuous Integration, Deployment and Delivery (CIDD) in Cloud Infrastructure" in which the complete automation of the Continuous Integration, Deployment and Delivery are done and experimentally tested. Our analysis shows that the proposed pipeline providing a productive environment for the developing team to automate the build and deploy their code up to the production line and the automation system effectively helps in saving the time and cost by increased software quality and productivity.*

Key Words**: Agile, DevOps, Cloud Infrastructure, Continuous Integration, Continuous Deployment and Continuous Deliver.**

1. INTRODUCTION

Traditional Software Testing and Deployment are being extra challenging in every traditional development practices. Agile and DevOps are emerging best practices which diminishes the weight of traditional software development. Software testing is trickier when it comes to Integration of a build with the main lane projects. Traditional practices (Requirement, Design, Code Build, Testing, and Maintenance) holds Software Testing as one sovereign phase. This type of approach is erroneous for the reason that the earlier detection of error saves more time and funds. For instance, fixing a bug at maintenance stage is ten times more lavish than fixing it in the stage of implementation. Recent practices like agile methodologies has enriched the approach by including testing at every stage of software development. But, still voluminous enterprises habit traditional testing methodology where software testing is typically accompanied later a build and implementation phases. Similarly, in earlier days every companies upholds two different teams for the production of software development viz., Development team and Operational team. However both teams used to work on the identical product, their goal lines are utterly opposite to each other. The aim of Development team is pushing for the feature changes, whereas the goal of Operational team is striving for stability. But today, things have improved, profoundly. A complete transition is made by means of new tools like cloud infrastructure and virtual technologies. This type of cultural change combines both the development and operational teams into one thin fast deployment machine to manage the infrastructure which is called DevOps as in [6]. In summary, agile and DevOps are two emerging practices that provide big transition to the cultural change of every software organization for productive development. By adopting to automated continuous integration, deployment and Delivery practice, the ideas of agile and DevOps can be turned into practical solutions. Continuous Integration (CI) is a crucial section of Agile and DevOps practices as in [1] [8] and [9]. CI is a key practice that frequently integrates a build with mainline shared repository by each developers in a developing team. This type of integration avoids a developer's local copy of code from drifting too extreme as soon as new code is affixed by other developers, avoiding disastrous integration conflicts.

In practice, CI comprises of a centralized server which constantly check-ins all the new source code changes as soon as the developers commit them, reporting any failures during a build as in [3]. CI server compiles, build, and tests every single fresh version of code committed to the main repository, it makes sure that the whole developing team is notified any time the mainline repository holds broken code. In advance, the CI server also deploy the verified application to quality assurance otherwise staging environment, safeguarding the Agile dream of a regular working version of the software product. Development teams take more than a few days for the Deployment process, even in cases of using automated CI to make sure that the code has been completely tested.

Continuous Delivery is an extended version of Agile Development. Continuous Delivery as in [4] takes the idea of Continuous Integration to the next succeeding footstep. As soon as the product is built, after the completion of CI process, delivering it to the subsequent stages is one other difficult jobs. At first the code needs to be delivered to the quality assurance (QA) team for testing and then to the Operational team (the *Ops* in DevOps) designed for transferring it to the production system. The objective of Continuous Delivery is to acquire the new features that the developers are building, out to the customers and users as soon as possible.

However, automating continuous deployment as in [5] is complex, time intense and frightening, also it is not typically precise how to go about it. One key solution to these problem is to automate the process of software build, testing, deployment and delivery to the product reaches the production environment. Doing the automation at the early stages of project development provide immediate worth. This could save time, cost and also helps detecting glitches with deployment timely in the development cycle, where fixing the problem becomes inexpensive too. In this paper, we have created an Infrastructure called "Continuous Integration in Cloud Infrastructure (CICI)" in which the complete automation of the Continuous Integration, Deployment and Delivery are done and experimentally tested. The analysis shows that the proposed infrastructure delivers a complete pipeline based automation of productive environment for the developing team to a build and then deploy their code to the production line. Industries by adopting to such cultural change are expected to deliver productive and fast development.

2. SCOPE OF THE PAPER

This paper compiles by the research started last June, 2015 up to March, 2016. Our research is all about automating the process of continuous integration, deployment and delivery in every software industries for productive their software development environment. The scope of this project is to encourage the corporate world to make benefit of available automation practices and tools.

3. ISSUES

Our research found that 30% of the software companies still uses traditional software development practices, less than 65% of the software organization implements the manual CIDD, and very few core companies (e.g., Microsoft, Google, Facebook, LinkedIn and Netflix) adapted to automated CIDD practice. This visibly confirms that the value of Continuous Integration has not yet significantly recognized by the software engineering world as in [3] [4].

3.1. Problems in the Manual-CI

By performing continuous integration in the manual they would face lots of human errors where the developer struggles to create the clean build. Since the integration is a tedious task, as in [2] the manual integration will take time, which sometimes leads to software failure. As far as a build is not clean, there is no ready-end delivery product which will highly reduce the software productivity and its quality.

3.2. Problems in Automated-CI

The companies who have automated the continuous integration practice also face the following problems: at the outset, one is the lack of knowledge of continuous integration tools and its environment as in [2] [3]. The automating tools and the automated environment is new to the developing team, so developers take time to learn the environment. Next, bigger the team (and the code base), the more often a build get broken. A broken build should be quickly fixed. The longer it takes to fix them, the higher possibility for the project to fail. And in many companies the target of the people is to build a product, but not to fix the broken code. To simply put, product owners do not understand the importance of a "clean build". It's been found that most of the development teams in many organization doesn't take alerts from CIDD seriously. A machine that continuously bounces red signal, that's how it looks to them.

4. PROPOSED INFRASTRUCTURE

The goal of the proposed infrastructure is to make the transformation of the manual Continuous Integration, Deployment and Delivery to automated CIDD and to simplify the Software Development Life Cycle for the software developers. Finally, we tend to enforce this beneficial discipline for each development team in industries by projecting the best in CIDD practice to the software engineering world.

We created an Infrastructure called CIDD that automates the Continuous Integration, Deployment and Delivery practice in an AWS Cloud Infrastructure for the improvement of software quality and productivity, which is as shown in Fig. 1. The proposed infrastructure provide a complete automated environment from build to production for the developing team to build their product in an efficient manner. The automations are done by using selective tools and technologies available in the market. CIDD automates consist of the following five automated stages:

4.1. Source Code Analysis

This stage of the proposed infrastructure works by assisting a code analysis tool which would make the source code less error-prone, more sustainable, more reliable, more readable, more welcoming to new contributors as in [3]. It is a compulsory step for the projects keen to go into the CICI, as the analysis process depend on code metrics extracted by Sonar. The main goal of this phase is to have quantitative measurements of the code quality and analyze the metrics of code to come up by a set of standard measurements.

Sonar helps in achieving these goals in addition to providing tools to instantly evaluate and monitor the standings of any project with respect to provided benchmarks. This also help decision makers in determining the issues, if accepted, offers the major increase in the quality. Furthermore, this tool helps developers in assessing of risk within their current software.

4.2. Continuous Integration Automation

Organizations evidently need to capitalize the build automation and test automation if they need to scale up and deliver features faster and release more often. This speculation can be expensive, but the manual approaches are visibly not scalable as in [1] [9]. As well, automation of build and test have a tendency to crop much sophisticated software quality.

4.2.1. Automated Build

Automating a build have turn out to be a foundation stone for agile development. Each time a developer checks in a code/change, Maven together with Jenkins checks out all the source code, build everything, runs all the unit tests and finally gives instant feedback. This part of automation is known as Continuous Integration (CI).

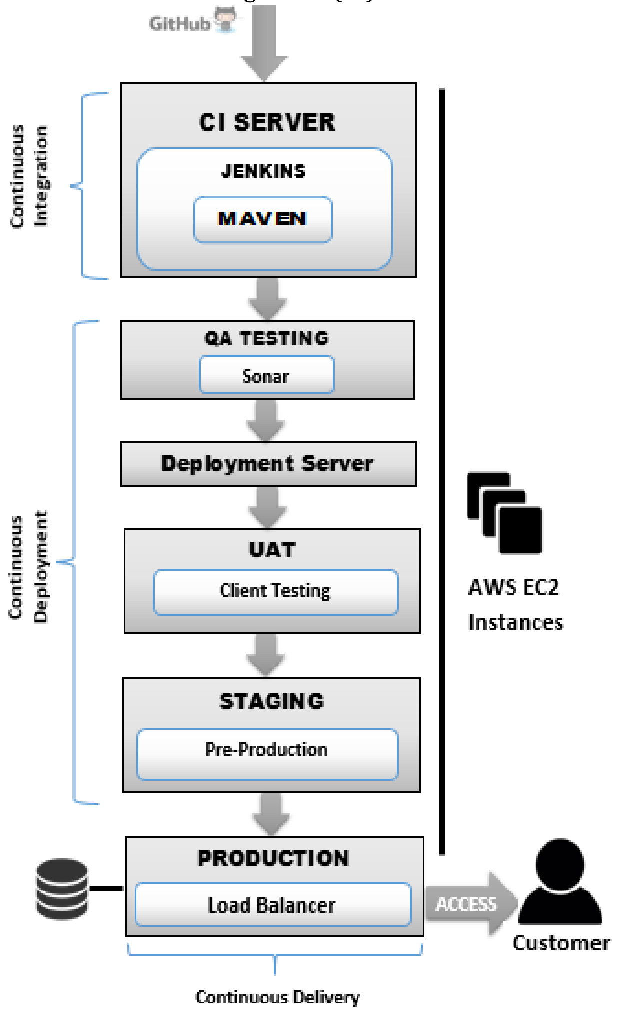


Fig - 1: CIDD in Cloud Infrastructure

4.2.2. Automated Testing

This is the risky part and most of the disclosures are encountered as we move from development in the direction of deployment into production. Test automation using Jenkins has been integrated into the CIDD. Whenever a test is passed, a new version can be built and a series of automated tests can be run against it. Outcomes from the test automation is used to vote to accept or reject contributions, this being a part of the workflow. Doing the integration test earlier in project development helps in revisiting incrementally as the requirements and system evolve.

4.3. Continuous Deployment Automation

Performing continuous deployment by means of Jenkins provide a great free solution for continuous build. However, once setting up a build code, build triggers, notification, data collection, and the subsequent step is to perform the deployments. This is in general performed by setting up a Jenkins's agent on the desired host where the code want to be deployed and by running a shell command along with the help of the Jenkins. A lot of times these kinds of deployments are too simplistic to go further than the testing environment. They take a shell script that replicates a war file which is determined not to work in full life-cycle and call for the target system(s) to be in a specific state which is usually done by manual process. The complete life-cycle deployment worth automatically deploying the application to entire environments from development on the way to production as in [5]. By means of this automation level, it is desirable to study the database fluctuations, configurations and integration points.

4.4. Continuous Delivery Automation

Continuous Integration covers the first principles of Continuous Delivery. Automating the continuous delivery entails that every single successful build will be made obtainable to the production line as in [4] [8]. The choice of whether or not to deploy a build to the production environment is entirely up to the developer.

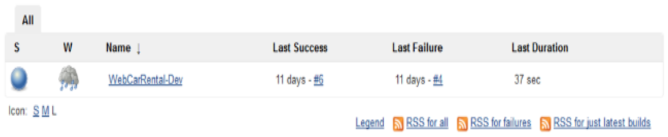
5. INTRODUCTION TO AUTOMATION TOOLS

Appreciatively, implementing the complete automation of Continuous Integration, Deployment and Delivery doesn't require any specialized toolsets outside of the common toolset which is available in the market. Several numbers of tools are available for the automation of these Agile and DevOps practices, where a careful selection of tools based of the business infrastructure is needed for the creation of CICI. A simple demonstration for selecting most popular tools are provided as in [5].

5.1. Jenkins

Jenkins is an open-source Continuous Integration server, like Hudson, Cruise Control, etc. CI server, it's fundamentally an overvalued scheduler in a nutshell, it executes a single build scripts whenever there is a trigger as in [5]. Jenkins has a build pipelines plugin, which was written in recent times by Centrum Systems. This pipeline gives exactly what the organization wants, like a way of breaking the build into smaller loops, and running stages in parallel.

Fig. 2. Jenkins Dashboard



All

S	W	Name ↓	Last Success	Last Failure	Last Duration
●		WebCarRental-Dev	11 days - #6	11 days - #4	37 sec

Icon: S M L

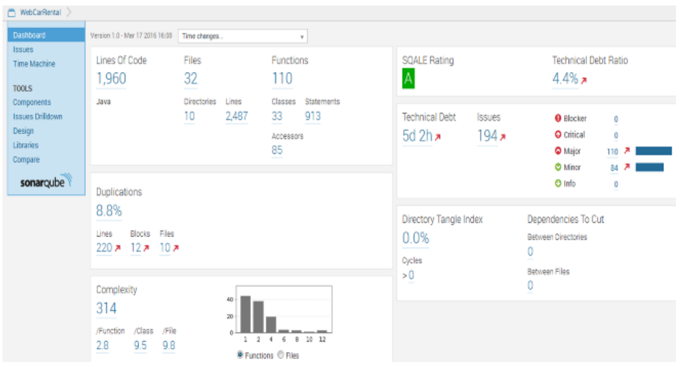
Legend RSS for all RSS for failures RSS for just latest builds

Fig – 2: Jenkins Dashboard

Installing Jenkins is very simple. One of the best things about Jenkins is the way it customs plugins, and how modest it is to get them up and running. The "Manage Jenkins" page has a "Manage Plugins" link on it, which provide the list of all the accessible plugins during Jenkins installation.

5.2. Sonar

Java Sonar is an open source web-based program analysis tool that allows developer to manage code quality. Using statistical code analysis tools, Java Sonar combines metrics self-possessed and reports on the standards of code quality measures such as: Code Duplication, Module Testing, Code Coverage, Coding Standards, Architectural Design, Code Complexity, Comments, and Potential bugs. In spite of the complexity of faced while data collection, Sonar has fairly a simple architecture, which is consist of three parts. First part is the Code analyzers which band together in a Maven plugin, and run on demand. These code analyzers are able to run both in Maven and non-Maven projects. Next is the Database which stores the output reports of the analyzer. They are stored along with the historical data's and project configurations. Sonar works on the bases of fixed rules which is compared for code quality. Sonar dashboard elaborates the software quality information on all of incoming projects combined, as well as on each individual one which is as shown in the Fig - 3.

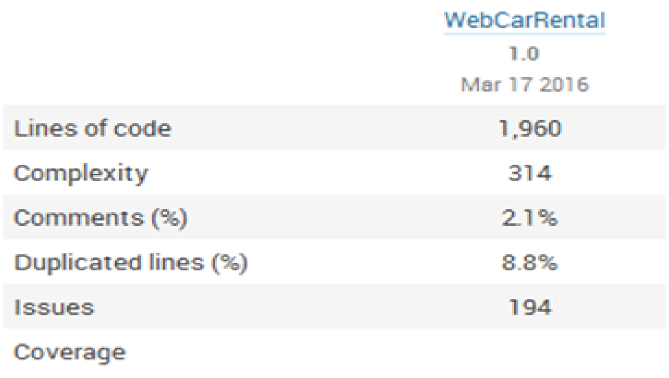


Version 1.0 - Mar 17 2016 16:08

Lines Of Code 1,960 Java
Files 32 Directories 10 Lines 2,487
Functions 110 Classes 33 Statements 913 Accessors 85
SQALE Rating A
Technical Debt Ratio 4.4%
Technical Debt 5d 2h
Issues 194
Blocker 0
Critical 0
Major 110
Minor 84
Info 0
Duplications 8.8% Lines 220 Blocks 12 Files 10
Directory Tangle Index 0.0% Cycles >0
Dependencies To Cut Between Directories 0 Between Files 0
Complexity 314 /Function 2.8 /Class 9.5 /File 9.8

Fig – 3: SONAR Dashboard

The statistics for the well ahead can be attained at the packages and modules level. Also, the several metric and comparison tools are present in this analyzer. Defect hunting is a tool that allows developer to get a better look at what is reported as an issue. And there available a Time Machine that records analytics of the project over time, so that one can able to watch how a project has evolved. Observing at the analytic reports of time machine, it allows us to understand how different approaches affect the project line.



	WebCarRental 1.0 Mar 17 2016
Lines of code	1,960
Complexity	314
Comments (%)	2.1%
Duplicated lines (%)	8.8%
Issues	194
Coverage	

Fig – 4: Code Complexity Metrics

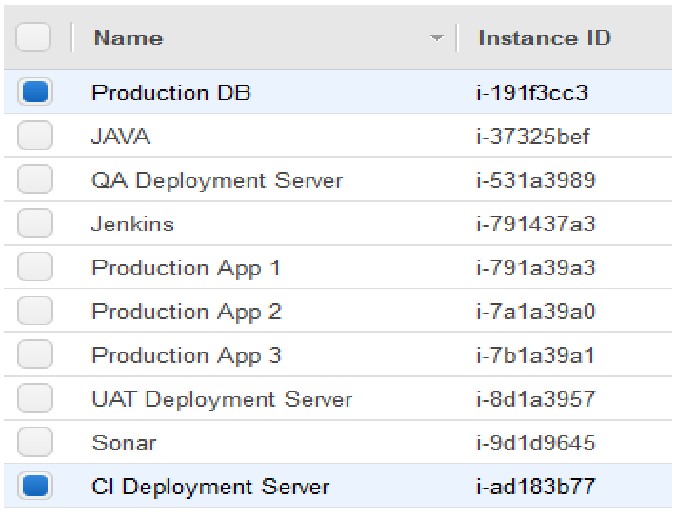
While Java Sonar gives us all of the information that could possibly needed for observing the quality of software code, it claims fairly an unsophisticated architecture. A set of code analyzers, a database, and a web reporting tool that would give us all the information needed to know. Though Java is the primary supported language of Sonar (hence Java Sonar), it also supports the languages viz. C, C#, Flex, PHP, PL/SQL, COBOL, Visual Basic 6, and Python by the help of plugins.

5.3. Maven

Apache Maven is a software project management and command tool. Grounded on the model of a project object model, Maven can cope up with a code build, reporting and documentation from the central piece of information available from code analysis. Maven's main goal line is to allow the developer to figure out the comprehensive state of a development effort in the shortest period of time. In order to achieve its own objective there are several areas of concern that Maven challenges to deal with: a build process becomes stress-free, providing a uniform build system, quality project information, guidelines provided for development best practices, and allows transparent migration to new features.

6. RESULT AND DISCUSSIONS

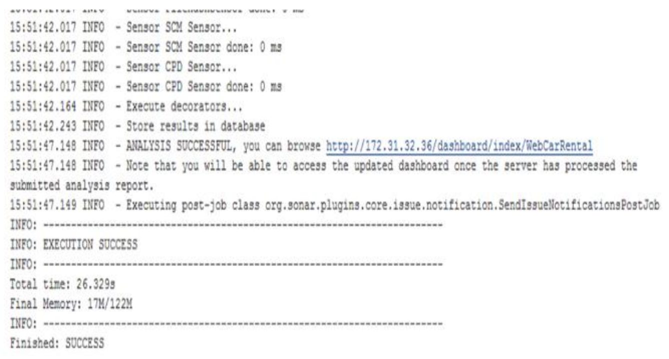
The proposed infrastructure has been tested experimentally with the help of a demo web based Maven project called "Web CAR RENTAL SYSTEM". Our observation shows that the complete automated pipeline worked magically over the testing and deployment process within expected short time. Our report clearly shows that the proposed infrastructure could provide more productive product in development for companies adopting to "CIDD" based Agile and DevOps practices. The demo report of the implemented infrastructure "CIDD" has given more in detail for reader's reference. By observing the performance of the proposed infrastructure, we strongly believe that it would be more productive for companies adopting to CIDD based Agile and DevOps practices.



☐	Name	Instance ID
■	Production DB	i-191f3cc3
☐	JAVA	i-37325bef
☐	QA Deployment Server	i-531a3989
☐	Jenkins	i-791437a3
☐	Production App 1	i-791a39a3
☐	Production App 2	i-7a1a39a0
☐	Production App 3	i-7b1a39a1
☐	UAT Deployment Server	i-8d1a3957
☐	Sonar	i-9d1d9645
■	CI Deployment Server	i-ad183b77

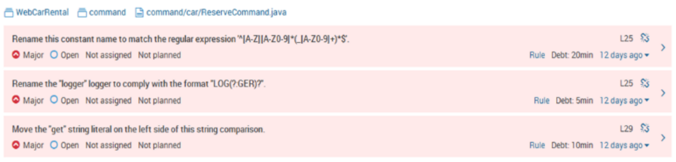
Fig – 5: AWS Cloud Infrastructure – CICI

We have created a cloud infrastructure for the automation of continuous integration, deployment and delivery using the Amazon Web Service (AWS). Jenkins, Sonar, Assembla, Maven and other supporting tools and technologies was used to create the CIDD Automation System. Jenkins is the continuous integration server which is integrated together with maven tool for java support using which the build and test are automated successfully. The console output of successful build and test of web car rental is as shown in the Fig. 5. Maven helps us to clean and install the fresh project many times instantly. This tool also helps us to verify, test and deploy the code at the desired server or environment.



```
15:51:42.017 INFO  - Sensor SCM Sensor...
15:51:42.017 INFO  - Sensor SCM Sensor done: 0 ms
15:51:42.017 INFO  - Sensor CPD Sensor...
15:51:42.017 INFO  - Sensor CPD Sensor done: 0 ms
15:51:42.164 INFO  - Execute decorators...
15:51:42.243 INFO  - Store results in database
15:51:47.148 INFO  - ANALYSIS SUCCESSFUL, you can browse http://172.31.32.36/dashboard/index/WebCarRental
15:51:47.148 INFO  - Note that you will be able to access the updated dashboard once the server has processed the
submitted analysis report.
15:51:47.149 INFO  - Executing post-job class org.sonar.plugins.core.issue.notification.SendIssueNotificationsPostJob
INFO: ------------------------------------------------------------------------
INFO: EXECUTION SUCCESS
INFO: ------------------------------------------------------------------------
Total time: 26.329s
Final Memory: 17M/122M
INFO: ------------------------------------------------------------------------
Finished: SUCCESS
```

Fig – 6: Jenkins - a Build and Test Report



WebCarRental command command/car/ReserveCommand.java

Rename this constant name to match the regular expression '^[A-Z][A-Z0-9]*(_[A-Z0-9]+)*$'. L25
Major Open Not assigned Not planned Rule Debt: 20min 12 days ago

Rename the "logger" logger to comply with the format "LOG(?:GER)?". L25
Major Open Not assigned Not planned Rule Debt: 5min 12 days ago

Move the "get" string literal on the left side of this string comparison. L29
Major Open Not assigned Not planned Rule Debt: 10min 12 days ago

Fig – 7: Automatic Issue Tracker - SONAR

After the successful build the code is then checked into Sonar, the source code analysis tool which follows rules from best practices to identify issues and measure the code quality. Fig. 6 shows the issues in web car rental system found by Sonar. Also, the code quality complexity of the project is as shown in Fig. 4. Continuous integration helps in check in code quality and reports every successful/failure build and issues found in the code that adopts to DevOps culture by means of visible operations.

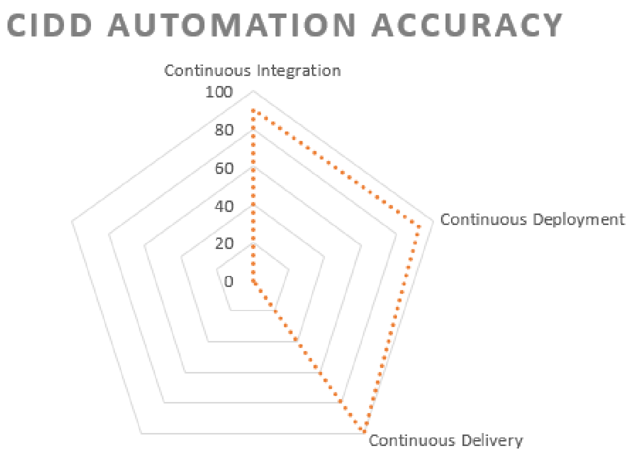


Fig – 8: Accuracy Analysis of CIDD Automation

CIDD deployment server automates the deployment process throughout the software lifecycle and the continuous delivery process is automated which release the final product to the production environment after the complete verification of product functioning at the preproduction staging environment. The graph shown in the Fig. 8 depicts the report analysis of CIDD system of automation accuracy in the integration, deployment and delivery as 90, 94 and 99 percentage respectively.

7. CONCLUSION AND FUTURE WORK

The main focus of our research is to dig out the importance of continuous integration, deployment and delivery in the software engineering world. Automating these practices often considered to be complex and frightening. However, we have examined the configurations involved in building a productive software development environment for development to production. In this paper, we demonstrated a newly structured infrastructure called "Continuous Integration, Deployment and Delivery (CIDD) Automation in Cloud infrastructure" using selective CI tools and amazon cloud services. Also, the proposed infrastructure has been demonstrated using a demo maven project "Web Car Rental System". The experimental evaluation of the proposed infrastructure has been explored the balance between development team and operation team. This prominent cultural change that would provide productive and quality development.

Our research found the importance of adopting agile practice for mobile application development, which would be used widely in the future. Researches are going on exploring difficulties in applying CIDD practice to mobile environment. Therefore, we target to enhance the CIDD Infrastructure for mobile application development in future.

REFERENCE

[1] Alexander Eck, Falk Uebernickel, And Walter Brenner (2014), "Fit For Continuous Integration: How Organizations Assimilate An Agile Practice", Twentieth Americas Conference on Information Systems, Savannah.

[2] Duvall P, Matyas S, and Glover A, "Continuous Integration: Improving Software Quality and Reducing Risk", Addison-Wesley, 2007.

[3] Martin Fowler, Robert. "Continuous Integration", http://www.martinfowler.com/articles/continuousIntegration.html , May 2006.

[4] Jez Humble, David Farley, "Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation", Addison-Wesley, 2010.

[5] Leppanen, M. (2015)"The Highways and Country Roads to Continuous Deployment", Software, IEEE (Volume: 32, Issue: 2), pg. 64 – 72.

[6] Len Bass, Ingo Weber, Liming Zhu, "DevOps: A Software Architect's Perspective", Pearson Edu, 2015.

[7] Meyer, M. (2014), "Continuous Integration and Its Tools", IEEE Software, Vol: 31, Issue: 3, pg. 14 – 16.

[8] Manish Virmani (2015) "Understanding DevOps & Bridging The Gap From Continuous Integration To Continuous Delivery", Innovative Computing Technology (INTECH), IEEE, pg. 78 – 82.

[9] Sean Stolberg (2009) "Enabling Agile Testing Through Continuous Integration" Agile Conference, AGILE'09, IEEE, Pg. 369 – 374.