**ABSTRACT**

Delivery of software solutions at a faster pace is essential in the industry today and to do that, the concept of automation came up and the IT industry has tried to implement the concept of automation to fasten the software production process and to speed up the process of product delivery to customer.

The proposed system does all the operations related to the software development and delivery process in an automated way. DEVOPS model has been incorporated in the system to carry out the task. Initial phase will carry out the manual installation and configuration of all the tools and services required for the proposed system and finally all the manual installations required for the release and deployment are automated using an automation tool called puppet.

The system is built by using tools like Git (Repository), Jfrog, Jenkins, Jira, Maven, Cobertura, Nagios (Monitoring tool) and puppet. These tools cover all the aspects of software development and deployment. The proposed system automatically builds the code, tests the code, bundles the packages, deploys the application and monitors the overall system.

TABLE OF CONTENTS

ABSTRACT i

ORGANISATION OF THE REPORT ii

ACKNOWLEDGEMENT iii

LIST OF FIGURES iv

LIST OF SYMBOLS AND ABBREVIATIONS v

1 INTRODUCTION 1

1.1 INTRODUCTION TO AUTOMATION 1

1.2 PROBLEM STATEMENT 3

1.3 OBJECTIVES OF THE PROPOSED SYSTEM 3

1.4 SCOPE OF THE SYSTEM 4

1.4 CONCEPT OF DEVOPS 4

1.5 DEVOPS CHAIN 5

1.6 SITE RELIABILITY ENGINEERING 6

1.7 INTRODUCTION TO AGILE METHODOLOGY 8

1.8 OVERVIEW OF THE PROJECT 8

1.9 ORGANISATION OF THE THESIS 9

2 LITERATURE SURVEY 10

2.1 EXISTING SYSTEM 10

2.2 LIMITATIONS OF EXISTING SYSTEM 11

2.3 PROPOSED SYSTEM 12

2.4 DRAWBACKS OVERCOME BY PROPOSED SYSTEM 14

2.5 SYSTEM ARCHITECTURE 15

2.5.1 AUTOMATED DEPLOYMENT AND MONITORING 15

2.5.2 DESCRIPTION OF ARCHITECTURE 15

2.5.3 TOOLS USED 18

3SYSTEM REQUIREMENT SPECIFICATIONS

3.1 SYSTEM REQUIREMENT

3.2 HARDWARE REQUIREMENT

3.3 SOFTWARE REQUIREMENT

4 PSEUDOCODE/ALGORITHMS 21

4.1 ALGORITHM TO MONITOR USED SPACE 21

4.2 ALGORITHM TO MONITOR NO OF PROCESSES 22

4.3 ALGORITHM TO MONITOR PUPPET MASTER 22

4.4 ALGORITHM TO MONITOR JENKINS 23

4.5 ALGORITHM TO MONITOR XINETD 24

4.6 ALGORITHM TO AUTOMATE JAVA INSTALLATION 24

5 IMPLEMENTATION / RESULTS AND ANALYSIS 26

5.1 INSTALLATION OF NAGIOS 26

5.1.1 Installation of httpd 27

5.1.2 Installation of nagios 27

5.1.3 Installation of xinetd and nrpe 29

5.2 CREATION OF CUSTOM PLUGINS 31

5.2.1 Plugin to check used space 32

5.2.2 Plugin to check total processes 32

5.2.3 Plugin to check total users 33

5.2.4 Plugin to check artifactory status 34

5.2.5 Plugin to check puppet master status 35

5.3 CONFIGURATION OF MASTER AND SLAVE NAGIOS

SERVERS 37

5.3.1 Definition of command in slave 37

5.3.2 Definition of commands in master 39

5.4 TERMINAL OUTPUT OF HOSTS AND SERVICES 39

5.5 INSTALLATION OF PUPPET 42

5.5.1 Installation of puppet master 42

5.5.2 Installation of puppet slave 43

5.6 PUPPET SCRIPTS FOR AUTOMATION 45

5.6.1 Automated deployment script 46

5.6.2 Automated installation of java 48

5.6.3 Automated jenkins installation 49

5.6.4 Automated cloning of code from git 50

6 CONCLUSION AND FUTURE WORK 53

6.1 CONCLUSION 53

6.2 FUTURE WORK 53

7 REFERENCES 54

LIST OF FIGURES

1.4 Development LifeCycle 9

3.1 AUTOMATED DEPLOYMENT AND MONITORING 15

5.1 Installation of httpd 27

5.2 Installation of nagios 28

5.3 Installation of nagios-2 29

5.4 Installation of XINETD 30

5.5 Installation of XINETD-2 30

5.6 Installation of XINETD-3 30

5.7 List of plugins 31

5.8 Plugin to check space used 32

5.9 Plugin to check total running processes 33

5.10 Plugin to check total active users 34

5.11 Plugin to check artifactory status 35

5.12 Plugin to check puppet master status 36

5.13 Definition of server host type 37

5.14 Defining command in slave 38

5.15 Defining command in master-1 39

5.16 Defining command in master-2 40

5.17 Console output of hosts 41

5.18 Console output of services 41

5.19 Defining puppet master 43

5.20 Defining puppet slave 44

5.21 Execution command of scripts 45

5.22 Automated deployment script 46

5.23 Jfrog artifactory 47

5.24 Tomcat view of deployed war 47

5.25 Automated java installation 48

5.26 Automated java installation-2 49

5.27 Automated jenkins installation 50

5.28 Git clone script 51

5.29 Git clone script 51

5.30 Site.pp file 52

LIST OF SYMBOLS AND ABBREVIATIONS

DevOps Development Operations

SRE Site Reliability Engineering

OPS Operations

HTTPD Hyper Text Transfer Protocol Daemon

XINETD Extended Internet Daemon

NRPE Nagios Remote Plugin Executor

ESXi Enterprise Server Xi

YUM Yellowdog Updater Modified

CI Continuous Integration

CD Continuous Deployment

RPM Redhat Package Manager

DNS Domain Name System

**CHAPTER 1**

**INTRODUCTION**

**1.1 INTRODUCTION TO AUTOMATION**

Today in IT sector there is a very tough competition in every domain. None of the company wants to lag behind with any respect to any domain. Say its time, cost or effort. Each day a new problem comes up for the customer and the need for immediate solutions has posed enormous challenges to the IT professionals who have to come up with good solutions to the customers problems.

There is a range of choice for the customers to choose the right vendor who can solve their problems and fulfil their requirements in the most efficient and cost-effective way.

The customer is in search of a vendor who can deliver good solutions in quick time. Hence the concept of automation came up and the companies have tried implementing the concept of automation to fasten their software production process and to speed up the process of delivery to the customer.

A company which delivers faster and good products is always preferred over any other company. The development stage of a product is just one phase of any project and the processes that follow the development stage are equally important to make sure that the product that we have developed is stable, correct and caters to the needs of the customer in the way it is intended to do so. If the processes like software, building, testing, packaging, deployment, delivery and maintenance are automated then a lot of time can be saved and the process of delivering the product becomes faster. Customers always come up with changes in their requirements and the automated system makes it easier for the IT professionals to make the changes in the system that the customer requires.

Setting up an automated system is a big task and as the system gets a considerable amount of shape the problem gets even bigger when integration of different components needs to be done. Automating the processes of software, building, testing, packaging, deployment, delivery and maintenance are very big aspects and there is a need to follow a systematic approach to achieve the process of automating such a big collection of components. Each processes have multiple tools and multiple services running that all need to be integrated and be in sync to achieve completion of the process and to deliver an effective and efficient system that can be trusted upon to deliver rapid and good service to the customer.

If we have to look at it from the aspect of the resources that are needed to complete any software project. We can say that a lot of human resource is required when it comes to software testing, deployment, delivery and maintenance. The cost and time taken to marshal these human resources in enormous. Whereas in the concept of automation of the software delivery process these resources will get cut down in size and the aspect of human interaction will get reduced. Hence the overhead of marshalling and keeping the human resources occupied can be written off. The machine in itself takes care of the processes that were once possible only with the help of direct human interaction.

Automation saves lots of time in the race to faster delivery of software products. Moreover the concept of automation will help the IT professionals involved to deliver faster and in a more efficient way. Automated testing can help them save time and we can expect faster results of the test which in turn will help the software professionals in proceeding with the next task.

**1.2 PROBLEM STATEMENT**

In traditional approach of software development process all the installations of the tools and services required for the development of the product are done manually. The tools, services, hardware components like switches, routers, etc need to be installed and configured manually. This is a time consuming job and if the number of components are more then it takes a lot of manual intervention and time to complete the task. A lot of dependent software’s and tools need to be installed to carry out the development process. And all these installations and configurations will take a lot of time and hence the delivery of the product will be delayed. Hence a system needs to be developed that automatically installs and configures the required tools and services on a large number of servers.

The bundled code packages are stored in an artifactory and these packages need to be automatically deployed onto the server. To accomplish this process, scripts need to be written to automate all the steps that are involved in the deployment process.

**1.3 OBJECTIVES OF THE PROPOSED SYSTEM**

* Installation and setup of DevOps tools.
* Integration of multiple tools within the DevOps chain.
* Deployment Automation of the DevOps tools.
* Automation of installation and configuration of tools.

**1.4 SCOPE OF THE SYSTEM**

This project aims to maximize the predictability, efficiency, and maintainability of operational processes. DevOps describes techniques for automating repetitive tasks within the software development life-cycle (SDLC), such as software builds, testing, and deployments, allowing these tasks to occur more naturally and frequently throughout the SDLC[1].

DevOps provides continuous integration, continuous deployment, continuous delivery and continuous testing of the software[2].

**1.4.1Continuous Integration**

The advantages of Continuous Integration model are,

* Provide continuous compiling, building, packaging of the software.
* On Each check-in, the system triggers the compilation process, runs the unit test and runs the analysis tools to review the build system.
* Developers can provide tagging for each new version of software build.
* The more stable software with less known issues would be deployed to the users.

**1.4.2Continuous Deployment**

The advantages of Continuous Deployment are,

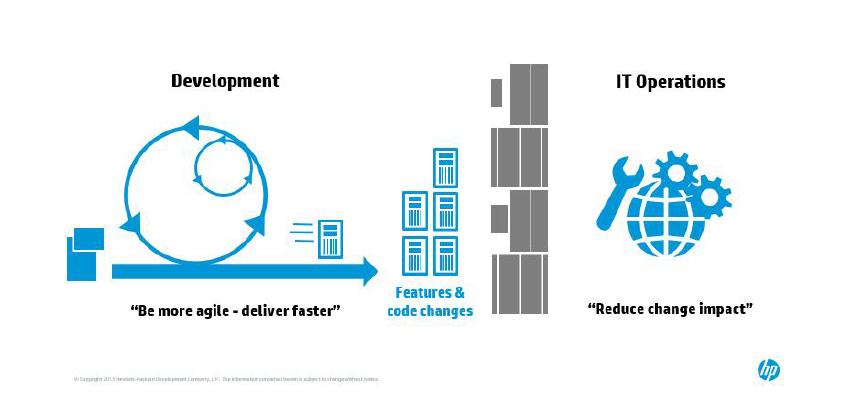
Software Updates are made available more frequently.

* When issues are identified, it will be rectified as soon as possible and new build is deployed after testing.
* Continuous Deployment reduces the cost with the time of software production.

**1.4.3 Continuous Testing**

Continuous Testing enables to achieve the following,

* Testing is executed on every latest software build
* Continuous Testing provides the real-time status of the quality of the software.
* The issues in the software can be identified as early as possible.



1.4Development Life cycle

**1.5 CONCEPT OF DEVOPS**

In order to achieve the enormous task of implementing the automation process of software production, certain set of practices need to be adapted and incorporated that will help to achieve the goal of automation in a more effective and efficient way. Hence the DEVOPS model is incorporated in the proposed system to make sure that there is a pattern or model that helps to get the desired results.

DevOps is a combination of software Development and information technology Operations. It is used to refer to a set of practices that emphasize the collaboration and communication of both software developers and information technology (IT) professionals while automating the process of software delivery and infrastructure changes. It establishes a culture where development, testing, software release can happen rapidly, often and in a reliable and controlled way and both the Developers and the Operations team work together. They have to be in tandem to make sure that the operations go on smoothly and that the IT professionals know how the code works with the help of the developers.

The key advantage of adopting DevOps in the current business trend is business agility. As the rate of change for business accelerates, companies are less able to predict where business is heading. The top strategic imperative becomes responding rapidly via agility and modularity through DevOps and adaptive IT. When it is unknown what the future holds, it’s important to have the infrastructure in place that can respond to business needs on the fly. When you use the DevOps agile methodology, IT works directly with the business users and delivers exactly what they need and nothing more. By focusing on business needs first the projects get finished and move into production faster.

DevOps allows companies to deliver software that is critical to the production, much faster sometimes 10x to 100x faster. For any business that captures advantage through any type of software, this kind of improvement could mean success or failure. In future, when more companies are successfully deploying DevOps, there will no longer be any room for failure to adopt DevOps. While there are other advantages to DevOps, a 10x to 100x speed advantage is the only one that counts.

If DevOps is adopted properly by the organization, IT’s role changes from being a business cost centre to being that of one that leads the innovation for the business. Product and service offerings can be introduced, tested, and rolled out much more quickly to the market place keeping up with the pace of change in the market place. The principles of DevOps allow IT teams to produce and deliver value quickly, test out hypotheses with real users, and roll out actual services and products that bring in revenue to help grow the business and make an impact to the top line.

**1.5 DEVOPS CHAIN**

The devops chain is a collection of tools that are part of devops model because the IT professionals use these tools to do the required tasks.

The developers develop the code as well as work around with these tools in their projects. Hence the devops tools are an essential part of the devops model.

DevOps is a cultural shift and collaboration (between development, operations and testing), there is no single ”DevOps tool”. It is rather a set of tools or a chain called as ”DevOps toolchain”, consisting of multiple tools. Generally, DevOps tools fit into one or more of these categories, which is reflective of key aspects of the software development and delivery process. The following are the types of DevOps tools that can be integrated in the devops chain.

* Code - Code development and code review, version control tools and
* code merging
* Build - Continuous integration tools, shows the build status
* Test - Regular Tests and results determine the performance
* Package - Artifact repository and application pre-deployment staging
* Release - Change management, release approvals, release automation
* Configure - Infrastructure configuration and its proper management, Infrastructure as Code tools.
* Monitor - Applications performance monitoring, end user experience Automated testing saves a lot of time and we can complete the tasks related to the project quickly. The use of monitoring tools helps us to keep track of all the components in the system and we can easily troubleshoot and find out solutions to problems with the help of these monitoring tools. The build tools will automatically generate the pom.xml file and we need not do the build process by ourselves and hence this process too saves a lot of cost and time.

**1.6 SITE RELIABILITY ENGINEERING**

Site reliability engineering (SRE) is a discipline that incorporates the aspects of software engineering and applies that to operations whose goals are to create ultra-scalable and highly reliable software systems. Site Reliability Engineering was created at Google around 2003 by Ben Treynor. A team of google engineers was given the big task to make Google’s sites run smoothly, efficiently and in a more reliable manner. Google’s large-scale systems required the company to come up with new techniques to manage large systems that have never existed before and at the same time introduce new features in a more continuous manner but also incorporating a very high-quality end user experience.

The word site in this aspect refers to a system which is composed of

a collection or a cluster of servers, networking devices, storage devices, set of tools installed on them and the different types of services that are running on the systems to achieve different purposes. A site reliability engineer (SRE) will ideally spend most of his time doing ”ops” related work such as issues, on-call, and manual intervention. Since the software system that an SRE handles is expected to be highly automatic and self-healing, the SRE should also spend their time on development tasks such as new features, scaling or automation.

The ideal SR Engineer is a coder who has operational and systems knowledge. DevOps plays the main role here and it is a superset of SRE. It refers to a set of practices that enables the collaboration of Software developers and Information technology professionals in a joint environment. The main tasks of a Site Reliability Engineer is to create highly scalable and reliable systems. The SR Engineer has the responsibility of automating the integration process of all the various tool and services running in the system and deployment of all the software services into the live environment. In the wake of emergency the SR Engineer will manually intervene and make specific changes that are required in the system to bring back a state of normalcy in the system and to achieve the goal of high availability to keep the customer happy.

Any new upgrade or launch needs to be scrutinized to evaluate the

amount and the type of resources that the proposed system will absorb and what will be the impact on the hardware resources. Hence the Site Reliability Engineers are IT professionals who develop the product as well as carry out the operations related to the product. Because they will understand the system better since they have built it and will be managing the related operations.

**1.7 INTRODUCTION TO AGILE METHODOLOGY**

Agile software development describes a set of principles for software development where requirements and solutions evolve through the collaborative work of different teams. It encourages the adaptive planning, evolutionary development, early delivery, and continuous improvement, and it encourages rapid and flexible response to change. The engineers follow the concepts of continuous integration and continuous deployment to deliver the correct system.

In the agile model the requirements can change at any time and the changes to the system need to be made immediately after due diligence. The changes are made rapidly as soon as the requirements are changed. The system needs to be integrated continuously and the deployment process also takes place continuously. Agile speeds up the process of product delivery. At the end of each cycle of development in this methodology there is a fully functional working module that can be deployed. The use of agile has been extended to this automation project and will be instrumental in the entire process.

**1.8 OVERVIEW OF THE PROJECT**

The project aims at automation of the processes involving software building, testing, deployment and monitoring. It initially involves all the manual installations and setup of multiple DevOps tools that range across different aspects such as build tools, testing tools, configuration tools, deployment tools, packaging tools, provisioning tools and monitoring tools. All the installations are done manually in the beginning.

Then the integration phase comes where multiple tools are integrated within the DevOps chain using a tool called Jenkins. The system requires the integration of a lot of tools across different domains like development, Software building, testing, packaging, deployment, delivery and monitoring tools. The Deployment Automation of the DevOps tools is carried out by using a tool called Puppet. This tool automates the installation of multiple tools and sets up the system automatically within seconds for any live environment. Hypervisor is used for the bare metal provisioning and management of the servers. It does the provisioning of servers (nodes) to the services.

Finally the proposed system will be used to automatically build the code, test the code, package the application, deploy the application and monitor it regularly.

**1.9 ORGANISATION OF THE REPORT**

* Chapter 1 illustrates problem statement, Objective of the project.
* Chapter 2 highlights the literature survey and works related to the existing conventional software development methodology.
* Chapter 3 illustrates the system design and presents the overall architecture of the system with the module descriptions.
* Chapter 4 provides the algorithms and the pseudocode representation of the algorithms used in the proposed system.
* Chapter 5 provides the results and analysis of the system and provides the implementation details of the system.
* Chapter 6 discusses the conclusion of the proposed system and also the future work.

**CHAPTER 2**

**LITERATURE SURVEY**

**2.1 EXISTING SYSTEM**

In traditional approach, a development team develops features and passes them on to a separate operations team may not address the needs of the frequent release scenario. The delays involved in acknowledging, testing and deploying the applications in the traditional manner increase the time to deliver the feature.

CONVENTIONAL SOFTWARE DEVELOPMENT, the existing system uses traditional Manual Installation methods and Deployment methods that create a lag in time, cost and efforts. The use of waterfall model was not very efficient in the software development process since at each stage the modules were fixed and any changes needed took a lot of time and resources and hence was very inefficient in many ways. In a similar manner the process of installations being done in a manual manner results in the same aspect of water fall model with a lot of time being spent on installations, upgrades and changes in the system, all being done manually. This does not help any organisation since the customer is always expecting immediate results and is ready to pay any amount to the organisation who can deliver that kind of results in a sort span of time.

In the existing system all the components need to be man handled and installed. If the downloading is done using source installation method then the system will have to search for the specific packages and download them and then we have to follow the steps and execute the steps of installations. And if this is done in a distributed environment then it gets very difficult since the number of machines is huge and accomplishing the installation on each and every machine is a very tedious and time consuming task. Physically carrying out the process in a big network involves the risk of human error. If a person handling the system performs a wrong step then there is a risk of the whole system crashing. This happens especially in the process of integration of various components in the system. In that case the process of installation for the entire system should be repeated and this will result in wastage of time and resources for the organisation.

The major difference occurs in the testing phase. Testing usually takes a lot of time and resources and it usually delays or extends the project completion date. If there is a system that can achieve the tasks of manual testing then a lot of time can be saved and we can complete the project at a very rapid pace. Thus the manuals operations related to the software delivery process need to shift to a new era of automation that will solve the problem of efficiency and consistency in regards to time, cost and resources.

**2.2 LIMITATIONS OF EXISTING SYSTEM**

* The existing system can work only within a certain scope andit cannot be extended to systems needing rapid change and modification.
* The existing system requires constant human interaction and resources
* In this system the administration of a distributed system is difficult since it will take a lot of time to manually attend each and every component across a distributed system that is very large in size.
* Existing system does not encourage constant change in requirements.
* The management of the system is mainly too time consuming.
* In the existing system addition of new components is an overhead since a lot of configuration changes need to be made.

**2.3 PROPOSED SYSTEM**

The proposed system is used to automate all the manual installations of the tools for the release. Automating the process of software building, deployment, delivery and maintenance are some of the goals of the proposed system. It also aims at automatic testing of the components and code that is used to create the application that will be running in the live environment. Automated monitoring and dash boarding of quality and performance against service level agreements at multiple stages is also one of the objectives of the system. By using advanced monitoring tools, high availability of the services can be achieved. Automate hand-offs/provisions to increase the velocity. It will automate all the manual installations for the release and deployment of the software. It will also automate tests that run fast and have good coverage on code. The system includes various devops tools that serve the purpose of the software production and deployment.

In this DevOps environment, on the other hand, the entire team is responsible for delivering both new features and stability. The combination of a shared code base, continuous integration, test-driven techniques and automated deploys, among other things, exposes problems in application code, infrastructure, or configuration earlier in the process, because the code is not thrown over the wall to operations at the end of coding. Problems tend to be less complex because change sets are smaller. DevOps engineers exploit real-time data into the performance of their systems to quickly understand the impact of application changes. And resolution times are faster because team members do not need to wait for a different team to troubleshoot and fix the problem.

The system requires a lot of installations and configurations of all the tools and services that will be used in the software project. The system will automate all the manual installations for the release and deployment of the software. There will be the usage of various DEVOPS tools like Git (Repository), Jfrog (Binary repository), Jenkins, Jira, Maven, Cobertura, Nagios (Monitoring tool), Zabbix, and Sonarqube. These tools cover the aspects of development, storage, integration, project management, system monitoring and product testing.

As far as automation is concerned integration of all these tools is done manually and then a tool called Puppet serves the purpose of automation of the entire system. Puppet is used to provide all the automation of all the tools and services that are running in the entire system and make them run in an order and correct manner just with the click of a single button.

Since the aim of any company is to make sure that the product or service is stable and available all the time we need to create a mechanism that take care of the concept of high availability. High availability means that the service is available to the customer at all times and to any number of customers.

Due to an overload the service should not go down and should manage the load and cater to all the customer who are requesting for the service.

To achieve high availability nagios is used that will monitor the system at all times. In order to make sure that a service crash is reported properly, custom scripts are written in Nagios to monitor each and every service running on the system. All the alerts are reported to the administrator in the form of alerts to the terminal or mails to the inbox of the administrator. This makes sure that any service crash doesn’t go unnoticed and all the services are tracked at all times.

**2.4 DRAWBACKS OVERCOME BY PROPOSED SYSTEM**

* The proposed system can work in a wide range and it can be extended to systems that need rapid changes and modifications.
* The proposed system requires only very minimal human interaction
* The proposed system enables the easy and efficient administration of a distributed system since there is minimal manual intervention in the system.
* The proposed system can incorporate constant change in requirements.
* The management of the system is easy and time efficient.
* The proposed system can incorporate addition of new components and a lot of configuration changes can be made.

**2.5 TOOLS USED**

**PUPPET**: Puppet is an open source configuration management tool which can run on both Linux and windows based systems. It has its own declarative language to describe system configuration and manages the system based on the script written in the manifests of the puppet configuration file.

Puppet manifests store system resources and their state using Puppet declarative language and contains all the details of all the nodes that are described in the manifest. It uses Client-server paradigm and puppet server controls all the automation processes of all the nodes in the system. The server contains the power to automate all the components in the system. The client responds to all the commands sent by the server. It uses scripts to communicate with the different components in the system.

The scripts define the behaviour of each component. It defines the order or execution that is related to each and every individual component in the system. For example, if it is a script for automating a testing tool, then we have to write the script in such a way that defines from where the code has to be taken, which code has to be taken, what are the test cases related to that particular piece of code and what is the criteria to pass the test. It also define what has to be done with the tested code after successfully completed the test.

There can be multiple puppet servers and the client should be installed on all the components that need to be automated. In our project we will be automating the Source code management system (GIT LABS), the testing tools, the build tool, the integration tool and the project management tool and the hardware provisioning tool (Razor).

**JFROG**: It is a software repository from which software packages can be retrieved and installed on a client machine. It stores the code and all the packages related to all the nodes in the system. It contains the instructions and all the executables of the entire system. It is called as an artifactory of all the executables in the system. All the packages that are built using the build tools and the packaging tools are finally stored in this repository. It serves as the home for all the packages. It then sends the packaged application to the automated deployment server that deploys the application in the live environment.

**NAGIOS**: It is an Open source monitoring tool that monitors all the components in the system. It monitors servers, switches, applications and all the services that are related to the system. It sends an alert whenever there is a problem in the system and it notifies to the manager and it is also updates in the puppet master and in the database that in turn directs the razor master to provision the reserve node for the service. It monitors all the activities of the hosts and services that are configured on it. It also monitors the working condition and status of all the hosts and services that are configured on the Nagios server.

The state of the host is depicted in three types of statuses to the administrator. The three statuses are OK, WARNING AND CRITICAL. In order to run NAGIOS we need to install certain dependencies or services that enable the proper functioning of NAGIOS. They are HTTPD, XINETD and NRPE. Every time we start the monitoring system we need to start the HTTPD,XINETD and the NRPE services to run NAGIOS.

HTTPD is the Apache Hyper Text Transfer Protocol (HTTP) server program. It is designed to be run as a standalone daemon process. When used like this it will create a pool of child processes or threads to handle multiple requests. The daemon answers the request automatically and serves the hypertext and all the documents over the internet using HTTP.

XINETD (extended Internet daemon) is an open-source super-server daemon which runs on many Unix-like systems and manages Internet-based connectivity. It listens for incoming requests over a network and launches the appropriate service for that request. Requests are made using port numbers as identifiers and xinetd usually launches another daemon to handle the request.It can be used to start services with both privileged and non-privileged port numbers.

NRPE stands for Nagios Remote Plugin Executor (NRPE). It is a Nagios agent that allows remote system monitoring using scripts that are hosted on the remote systems. It allows for monitoring of resources such as disk usage, system load or the number of users currently logged in. It allows you to remotely execute Nagios plugins on other Linux/Unix machines. This allows you to monitor remote machine metrics (disk usage, CPU load, etc.). NRPE can also communicate with some of theWindows agent add-ons, so you can execute scripts and check metrics on remote Windows machines as well.

**HYERVISOR**: The hypervisor is used to create virtual machines in our system. The hypervisor presents the guest operating systems with a virtual operating platform and manages the execution of the guest operating systems. Multiple instances of a variety of operating systems may share the virtualized hardware resources. They run directly on the host’s hardware to control the hardware and to manage guest operating systems. Whenever there is a request to create a new machine, the hypervisor takes the request along with details of the node to be created and creates a machine on the basis of the specifications. The specifications include the hardware resources to be allocated for new machine.

VMware ESXi is an enterprise-class, type-1 hypervisor developed by VMware for deploying and serving virtual computers. It is not a software that one installs in an operating system.

**CHAPTER 3**

**SYSTEM REQUIREMENT SPECIFICATION**

**3.1 System Requirements**

While working on a project as our problem statement is all in all what we need to do is we need to automize the whole processes. There are so many companies which are still using their technologies manually . people may get so many new technologies but it is very much necessary to save the time and cost. And most of all backup. Now imagine there isn a worldwide company which is working on various projects and in a team one person is sitting in country A and other fellow is sitting in country B. now sending the code to the other person through mail or other medium is just not feasible all the time. Now we want some solution to this problem. Here we get is the concept of continuous integration and continuous deployment. So here in our project we will be using GIT. Github is a remote repository. Example if admin has assigned some work to the developer , then in this case what we require is that all the codes done by all the developers should be present somewhere that it should be accessible to all the team members irrespective of the work locations. Now imagine there is a worldwide company which is working on various projects and in a team one person is sitting in country A and other fellow is sitting in country B. now sending the code to the other person through mail or other medium is just not feasible all the time. Now we want some solution to this problem.

**3.2Functional Requirements**

Functional requirements specify the key functions to be performed by the proposed system. These describe the functionality or services that the system is expected to provide.

**Functional requirements** are baked into the code that developers deliver (interpreted or compiled).   Events from input devices (network, keyboard, devices) trigger functions to convert input into output -- all functions have the form:   
 [http://accelerateddevelopment.ca/blog/wp-content/uploads/2013/12/Function.png](http://accelerateddevelopment.ca/blog/wp-content/uploads/2013/12/Function.png)

**3.3 Non-functional requirements**

  Non-functional requirements involve everything that surrounds a functional code unit.  Non-functional requirements concern things that involve time, memory, access, and location:

* Availability
* Capacity
* Continuity
* Security

Non-functional requirements are slightly different between desktop applications and services;

**3.3.1Availability**

Availability is about making sure that a service is available when it is supposed to be available. Availability is about a **Configuration Item** (CI) in the environment of the operations centre that specifies how the code is accessed.  Availability is decided independently of the code and is at best part of the **Service Design Package** (SDP) that is delivered to the operations department, at worst it is simply code dumped on the operations personnel.   
  
 Developer's need to be aware of the difficulty of creating the **CI** for the operations personnel.  If a **CI** is manually created then there will always be a potential for an error when the service is installed or updated. The requirement to create a **CI** is a non-functional requirement and the ability to minimize errors is another non-functional requirement.   
  
 Developer's need to be aware of single-points of failure (i.e. services hard-coded to a specific IP) which causes fits in operations that are not running virtual machines (VM) that can have virtual IPs . The requirement to create code that is not reliant on static IPs or specific machines is a non-functional requirement.  Availability is simplified in operations if the code is resilient enough to allow itself to easily move (or be replicated) among servers.

Availability non-functional requirements include:

* Ability to easily make the **CI**
* Automatic installation of **CI** or mechanisms
* Ability to detect and prevent manual errors for a **CI**
* Ability to easily move code between servers

**3.3.2Capacity**

Capacity is about delivering enough functionality when required.  If you ask a web service to supply 1,000 requests a second when that server is only capable of 100 requests a second then some requests will get dropped.  This may look like an availability issue, but

it is caused because you can't handle the capacity requested.

Internet services almost always can't provide enough capacity with a single machine and operations personnel need to be able to run multiple servers with the same software to meet capacity requirements.  The ability to run multiple servers without conflicts is a non-functional requirement. The ability to take a failing node and restart it on another machine

or **VM** is a non-functional requirement.   
Capacity non-functional requirements include:

* Ability to run multiple instances of code easily
* Ability to easily move a running code instance to another server

**3.3.3 Continuity**

Continuity involves being able to be robust against major interruptions to a service, these include power outages, floods or fires in an operational center, or any other disaster that

can disrupt the network or physical machines.

Where availability and capacity often involve redundancy inside a single operation center, continuity involves geographic and network redundancy.  Continuity at best involves having multiple servers that can work in geographically distributed operation centers.  At worst, you need to be able to have a master-slave fail over model with the ability to journal transactions and eventually bring the master back up.

**3.3.4Security**

Security non-functional requirements concern who has access to functions and

preventing the integrity of data from being corrupted.

Where access is concerned, how difficult will it be for operations personnel or help desks to set up security for users?  Developer's build in different levels of access into their applications without considering how difficult it will be for a 3rd party (help desk or operations) to set-up end users.  The ease of setting up security is a non-functional requirement.   
 Data integrity is another non-functional requirement.  Developer's need to consider how their applications will behave if the program encounters corrupted data due to machine or network failures.

**3.4Hardware requirements**

* Processor : Intel Core
* RAM : 4GB
* Hard disk space : 100GB
* Input : Keyboard and Mouse
* Output : LCD Monitor

**3.5 Software Requirements**

* Operating System : Windows 7 and Above
* Browser : Internet Explorer 11, Chrome
* Database : MySQL
* Build : Maven, MS Build
* Integration : Scripting (Puppet, Bash Shell)
* Version Control : GIT
* Project Management : JIRA
* Environment : Eclipse, Intellij
* Analysis : SonarQube

**CHAPTER 4**

**SYSTEM ANALYSIS**

**4.1PROJECT DESCRIPTION:**

Today in IT sector there is a very tough competition with each and everything. None of the company wants to lag behind with any respect. Say its time, cost or effort. Since 2 years back all the companies were working on cloud with tools by installations and configurations. There were 4 big companies who were working on cutting edge technologies of cloud and automation. They say it SRE(Site reliability Engineering).

Our mission is to automate all the manual installations for the release and deployment of the software. We will be using various tools and software like Git, jfrog, Jenkins, jira, maven, git etc. these almost all of them work with each other. In the end we will be using Puppet to automate all the tools. CICD is a process for continuous development, testing, and delivery of new code.

This system automate all the manual installations of the tools for the release. Automate tests that run fast and have good coverage on code. Automate the deployment of the software.

**4.1.2 CONTINUOUS INTEGRATION**

Continuous Integration (CI) is a development practice that requires developers to integrate code into a shared repository several times a day. Each check-in is then verified by an automated build, allowing teams to detect problems early. By integrating regularly, you can detect errors quickly, and locate them more easily. In this project we are going to integrate continuously in a Agile methodology.

Continuous integration in practice, it is a use of a version control tool (CVS, SVN, Git, etc.). An automated build and product release process. Instrumentation of the build process to trigger unit and acceptance tests ”every time any change is published to version control”. In the event of even a single test failing, alerting the team of a ”broken build” so that the team can reach a stable, releasable baseline again soonest.

Optionally, the use of a tool such as a continuous integration server, which automates the process of integration, testing and reporting of test results.

The team responds to a broken build, suggesting that a defect may have been detected. If the team is aware of defects, but tolerates them or continues working on a product that isn’t in a releasable state, the term continuous integration no longer applies, irrespective of tooling.

In software engineering, continuous integration (CI) is the practice of merging all developer working copies to a shared mainline several times a day. Grady Booch first named and proposed CI in his 1991 method, although he did not advocate integrating several times a day. Extreme programming (XP) adopted the concept of CI and did advocate integrating more than once per day - perhaps as many as tens of times per day.

**4.1.3 CONTINUOUS TESTING**

Continuous testing is the process of executing automated tests as part of the software delivery pipeline to obtain immediate feedback on the business risks associated with a software release candidate. Here we are using the methodology continuous testing where we generate automated test cases.

The approach to Continuous Testing can vary and follow diverse pathways to ensure the best user experience is delivered, free of defects.

It’s barely viable to repeat all tests every time a new feature is added, so the Continuous Testing strategy fosters a company-wide cultural change to achieve four capabilities: Test early, test faster, test often and automate.

End-to-end test automation practices are intended to integrate QA into existing fast-paced Dev and Ops processes as a means to create continuity while maintaining faster development cycles.

It’s important to understand the role of automation in Continuous Testing. Automation alone does not enable continuity in testing, but it helps provide a qualitative assessment of risk and practice actionable tasks to mitigate these risks throughout the SDLC.

In essence, automated testing constitutes the detection process for software issues and defect prevention, whereas Continuous Testing addresses the wider challenge of improving the effectiveness of these detection sensors.

**4.1.4 CONTINUOUS DELIVERY**

It is often assumed that if we want to deploy software more frequently,we must accept lower levels of stability and reliability in our systems. In fact,peer-reviewed research shows that this is not the casehigh performance teams consistently deliver services faster and more reliably than their low performing competition. This is true even in highly regulated domains such as financial services and government. This capability provides an incredible competitive advantage for organizations that are willing to invest the effort to pursue it.

**4.1.4.1 Benefits of continuous delivery**

1. Low risk releases: The primary goal of continuous delivery is to make software deployments painless, low-risk events that can be performed at any time, on demand. By applying patterns such as blue-green deployments it is relatively straightforward to achieve

zero-downtime deployments that are undetectable to users.

2. Faster time to market: It’s not uncommon for the integration and test/fix phase of the traditional phased software delivery life cycle to consume weeks or even months. When teams work together to automate the build and deployment, environment provisioning, and

regression testing processes, developers can incorporate integration and regression testing into their daily work and completely remove these phases. We also avoid the large amounts of re-work that plague the phased approach.

3. Higher quality: When developers have automated tools that discover regressions within minutes, teams are freed to focus their effort on user research and higher level testing activities such as exploratory testing, usability testing, and performance and security testing. By building a deployment pipeline, these activities can be performed continuously throughout the delivery process, ensuring quality is built in to products and services from the beginning.

4. Lower costs: Any successful software product or service will evolve significantly over the course of its lifetime. By investing in build, test, deployment and environment automation, we substantially reduce the cost of making and delivering incremental changes to software by eliminating many of the fixed costs associated with the release process. products: Continuous delivery makes it economic to work in small batches. This means we can get feedback from users throughout the delivery life cycle based on working software. Techniques such as A/B testing enable us to take a hypothesis-driven approach to product development whereby we can test ideas with users before building out whole features. This means we can avoid the 2/3 of features we build that deliver zero or negative value to our businesses.

**4.1.5 CONTINUOUS DEPLOYMENT**

Continuous deployment can be thought of as an extension of continuous integration, aiming at minimizing lead time, the time elapsed between development writing one new line of code and this new code being used by live users, in production.

To achieve continuous deployment, the team relies on infrastructure that automates and instruments the various steps leading up to deployment, so that after each integration successfully meeting these release criteria, the live application is updated with new code.

Instrumentation is needed to ensure that any suggestion of lowered quality results in aborting the deployment process, or rolling back the new features, and triggers human intervention.

**4.1.5.1 Expected Benefits**

The main benefits claimed for continuous deployment arise as a result of reducing lead time, with two main effects:

1. Earlier return on investment for each feature after it is developed, which reduces the need for large capital investments

2. Earlier feedback from users on each new feature as it is released to production, which affords techniques such as parallel (or A/B) testing to determine which of two possible implementation is preferred by users.

**4.1.5.2 Potential Costs**

1. Continuous deployment relies on extensive instrumentation to ensure that functionality newly made available to users does not result in incidents, lowering externally perceived quality.

2. For the same reason, continuous deployment relies on infrastructure that allows easily backing out new features when a defect has not been detected by automated tests.

**4.2** **High-Level Design**



Figure 4.2 High Level Design

**Figure** 4.2 illustrates the steps involved to achieve Site Reliability Engineering.

In a deployment environment there will be a considerable amount of traffic between individual tool servers. In order to manage these traffic, It is pretty much necessary for us to implement load balancer.

A load balancer will always take care about the loads (traffic) so we configured all our tools with the load balancer, in our project we use NGINX as our load balancer.

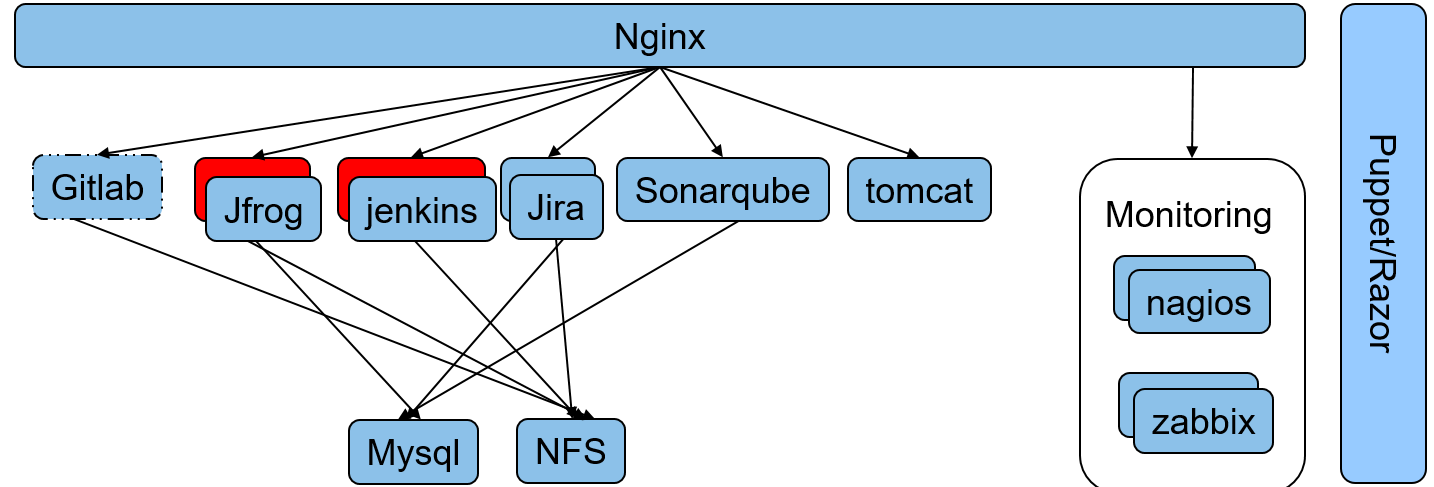


Figure 4.2.1 Load Balancer

Figure 4.2.1 Depicts a typical NGINX Load balancer which is connected on top all tool server.

NGINX (Pronounced as Engine-X) is an open source, lightweight, high-performance web server or proxy server. Nginx used as reverse proxy server for HTTP, HTTPS, SMTP, IMAP, POP3 protocols, on the other hand, it is also used for servers load balancing and HTTP Cache. Nginx accelerates content and application delivery, improves security, and facilitates availability and scalability for the busiest websites on the Internet.

Httpd service can run in two MPS (Multiple process Management)

1. Prefork( It will create a process for every request)
2. Worker (It will create a process for the first request and other request will be processed by the same PID(process id).

**4.3 Feasibility Study**

The feasibility study proposes one or more feasible conceptual solutions to the problem set of the project. The conceptual solutions give an idea of what the new system will look like. This indicates what inputs are needed by the system and what outputs will be produced.

Three things to be done to established feasibility, they are:

* First, it must be checked that the project is technically feasible.
* Second, operational feasibility must be established. It is necessary to consult the system users to see if the proposed solution satisfies user objectives and can be fitted in to current system operation.
* Third, economic feasibility must be checked. The study must determine whether the project’s goal can be achieved within the resource limits allocated to it. It must also determine whether it is worthwhile to proceed with the project at all or whether the benefits obtained from the new system are not worth the cost.
  + 1. **Technical Feasibility**

Technical Feasibility is the study of resource availability that may affect the ability to achieve an acceptable system. Technical Feasibility is the most difficult area to ensure at initial stages. Since the objectives functions and performance cannot be predicted to its fullest, everything seems possible provide, proper assumption are made. It is essential that the process of Technical Feasibility.

**4.3.2 Economic Feasibility**

Automation is highly economically feasible .The organization needed not spend much money for automation of readily available tools. The only thing is to be done is making an environment for the automation with an effective supervision. If we are doing so, we can attain the maximum usability of the corresponding resources .Even after the automation, the organization will not be in condition to invest more. Therefore, the system is economically feasible.

**4.3.3 Operational Feasibility**

An estimate should be made to determine how much effort and care will go into the developing of the system including the training to be given to the user. Usually, people are reluctant to changes that come in their progression. The computer initialization will certainly affected the turn over, transfer and employee job status. Hence an additional effort is to be made to train and educate the users on the new way of the system.

* + 1. **Motivational Feasibility**

An evaluation of the probability that the company is significantly motivated to support the development and implementation of the application with necessary user participation, resources, training etc. The participation and support by the organization during system study was encouraging thus eliminating any resistance in this regard. So from the Behavioural aspect the new system is supposed to have efficient from the company.

**4.3.5 Schedule Feasibility**

The time schedule required for the development of this project is very important since over-runs result is escalated projects costs and also hinders in the development of the other system.

**4.3.6 Feasibility Check**

Automation is a key to success in today’s world. Even a normal showroom vendor also wants to automate his bill statements. But it has not touched these heights till now. Since it is a new technology or not very old.

Companies are still trying their best to come up with best techniques everyday. Puppet is one of the very important invention to automate each and everything. It helps in load balancing also in high availability. As per the development life-cycle of the DevOps system is being carried out with agile methodology. This will deliver faster[3].

**I Objectives of the Proposed System**

The objectives of this proposed system DevOps is depicted as follows,

* To automate the process of build, deploy and testing.
* To increase the rate of production releases from application and business unit stakeholders.
* To effectively automate the test and to continuously integrate the product.
* To improve the system of production and service.
* To improve quality and productivity and thus constantly decrease costs.
* To eliminate the need for massive inspection by building quality into the product in the first place.
* To automate the manual processes like dependency discovery and resolution, system construction, provisioning, update and rollback.

**CHAPTER-5**

**SYSTEM DESIGN**

System design is the process or art of defining the architecture, components, modules, interfaces, and data for a system to satisfy specified requirements. One could see it as the application of systems theory to product development. There is some overlap with the disciplines of systems analysis, systems architecture and systems engineering.

**5.1SYSTEM ARCHITECTURE**

**5.1.1 AUTOMATED DEPLOYMENT AND MONITORING**

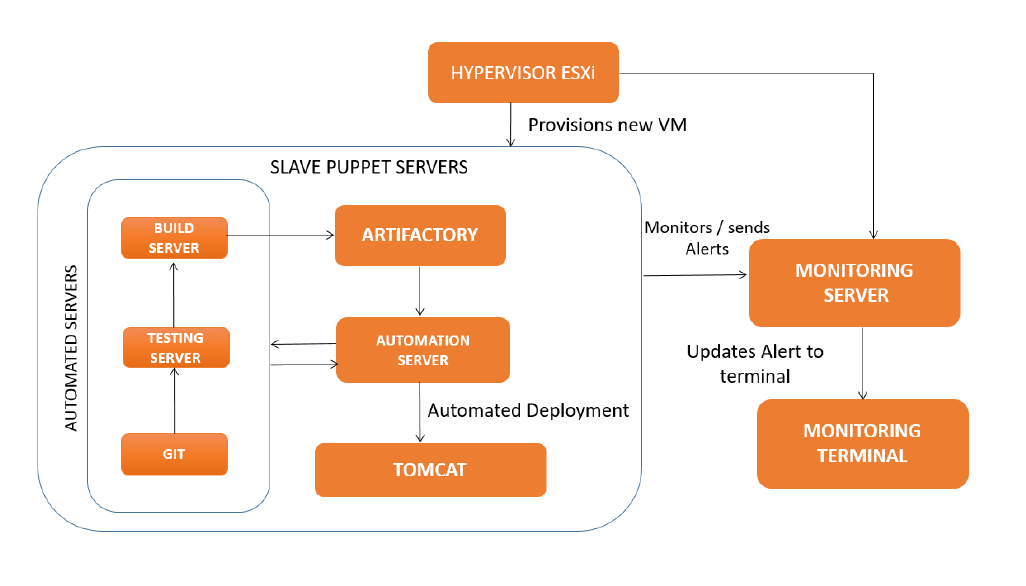


Figure 2.5: AUTOMATED DEPLOYMENT AND MONITORING

**5.1.2 DESCRIPTION OF ARCHITECTURE:**

Figure 2.5 depicts the architecture of the system and particularly concerned with the aspects of system monitoring and automated deployment.

The code is taken from the Git labs server which contains the bulk of the application code and it is tested using various testing tools. It is then built using build tools like Maven and Jenkins and it is forwarded and stored in the artifactory after the build process is completed.

**ARTIFACTORY**: It is a repository that contains all the packages needed to run the application. It contains the war and jar packages that are needed to run the system. Any number of packages can be stored in an artifactory and all the packages can be integrated and deployed as one application. The packages stored in the artifactory are taken and deployed automatically by the automated deployment server.

**AUTOMATION SERVER**: It is an automation server. It uses a tool called puppet for the purpose of automation of the entire system. The packages that are stored in the artifactory are deployed automatically into the live environment using puppet. It uses the puppet scripts to automate all the tools and services that are running in the entire system. This automation server can automate the installation and configuration of any tool or service on any number of hosts. A large number of services can be configured simultaneously on a large number of hosts with zero human intervention.

Once the slave host initiates the pull request from the manifests, all the configuration files of that particular node starts running and automatically calls all the module files that are stored in the master puppet server. These module files contain the script for the installation of all the services. We can split the modules files based on which service is to be installed on which server and accordingly can specify the list of module files under that particular node in the site.pp file.

**MONITORING SERVER**: The monitoring server monitors all the components of the system and keeps track of the system status and uptime and downtime. It monitors the behaviour of the components based on certain rules and an alert is sent to the monitoring terminal wherever an issue arises. The alert is addressed by the SRE (Site Reliability Engineer). A tool called Nagios is used for monitoring the entire system. It is an open source tool that allows the administrator to write custom scripts to monitor specific services.

**MONITORING TERMINAL**: The monitoring terminal is the console from where it is possible to monitor all the hosts, tools and services that are configured in the system. The terminal is part of the Nagios master server and displays the state and run time messages of all the services and tools in the proposed system.

**HYPERVISOR**: The Hypervisor creates a new instance of a machine. The hypervisor is used to create virtual machines in the system. Based on the requirements, a machine is created. It can either be a linux based or a windows based machine. In the proposed system most of the machines are linux based except for one machine. One machine is windows based which is used for installation of testing tools like LeanFT, UFT and monitoring tool AGM.

This was part of the requirements from the testing team. The operations team executive creates these virtual machine using the EXSi hypervisor and provides the platform to install, configure and run all other services.

The beauty of these tools involved in this system is that they can be integrated together and automated to do complex tasks in a short span of time.

The puppet scripts that are written will take care of all the automation operations needed to run the system in an automated way. The scripts that are written for monitoring system will take care of all the components in the system and will send alerts about any changes being made in the system.

**5.2 Data Flow Diagram:**

A data flow diagram (DFD) illustrates how data is processed by a system in terms of inputs and outputs. As its name indicates its focus is on the flow of information, where data comes from, where it goes and how it gets stored.

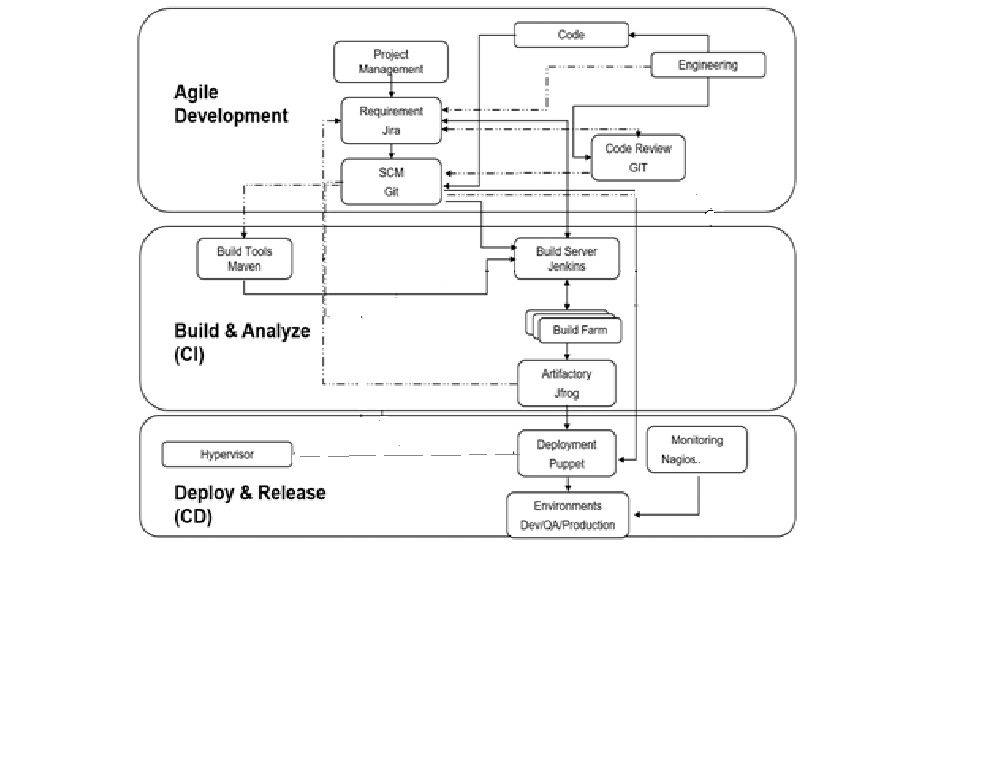
****

Figure 5.2 Data Flow Diagram.

Figure 5.2 will Depicts the Dataflow diagram of our projects.

**5.3USE-CASE DIAGRAM**

The explanation behind use-case diagram is to get the dynamic part of a system. In any case, this definition is unreasonably non particular, making it difficult to delineate the reason. In a matter of seconds when the basic task is done use case layouts are shown to demonstrate the outside point of view. The inspirations driving use case charts can be according to the accompanying:

* Used to amass essentials of a structure.
* Used to get an outside point of view of a structure.
* Recognize external and inside components affecting the system.
* Exhibit the interfacing among the essentials are performing craftsmen.

The Use case diagram of the DevOps office model is given in Figure 5.3

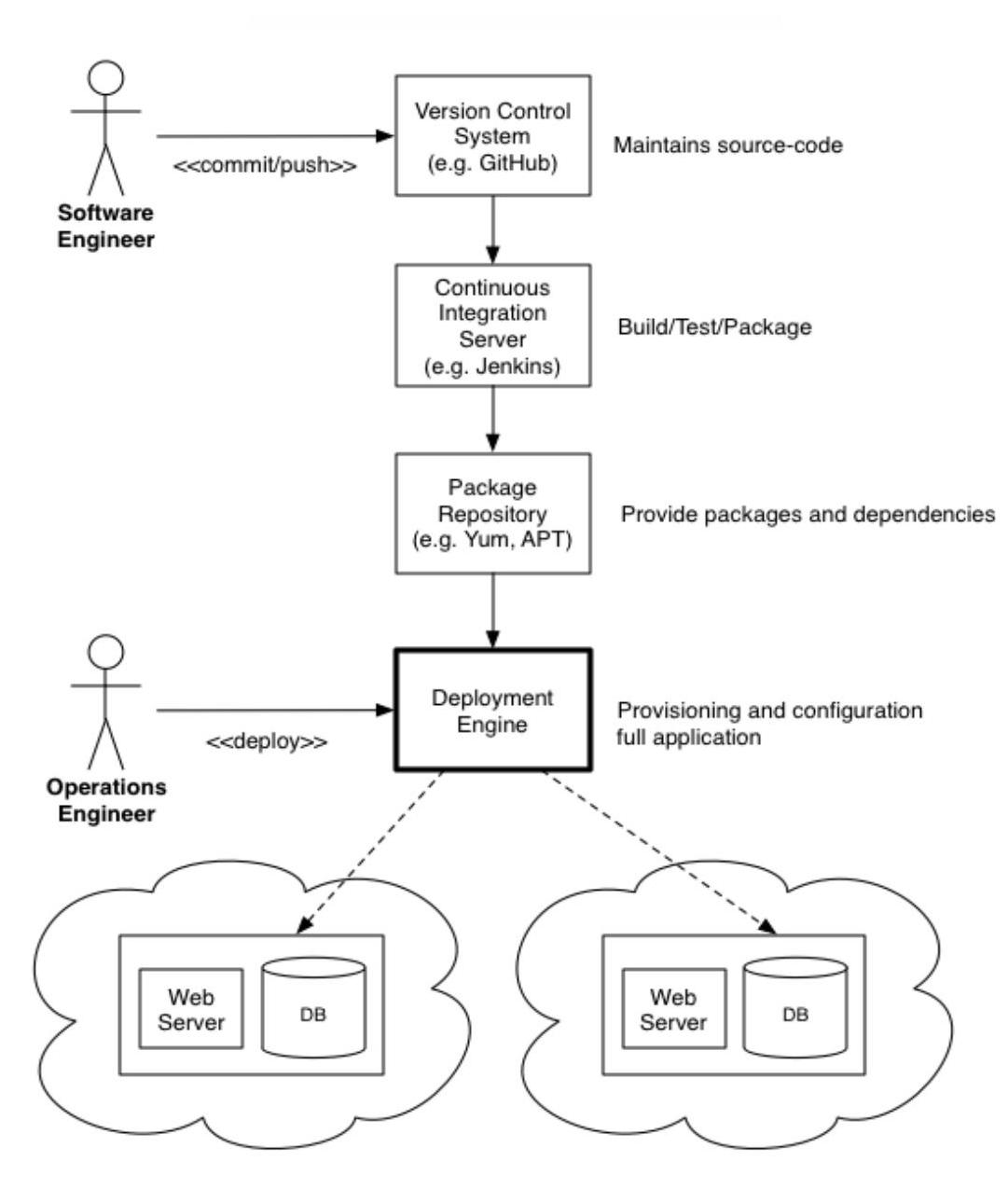


Figure 5.3: Devops Office Model Usecase Diagram

**5.4 SEQUENCE DIAGRAM**

This sections show the sequence diagram for the DevOps Office Model in Figure 4.1 where the sequence flows like the code is being pulled from the Git repository. Then it is build using the maven and pushed to git and the application is being tested with SonarQube and these tools are integrated using the integration server called Jenkins.



Figure 5.4: Sequence Diagram-SRE Integration

**5.5 ALGORITHMS**

The pseudocode for the nagios plugins is written here in this chapter.

**5.5.1 ALGORITHM TO MONITOR USED SPACE**

Input: Command to view memory statistics of host

Ouput: The amount of memory used and status of host

1: GET the memory details

2: SELECT filesystem

3: SELECT value of column5

SET column5 to spaceused

SET limit1 to 75

SET limit2 to 85

4: if spaceused less than limit1 then

5: spaceused used is under limit OK

elseif

6: spaceused greater than limit1 and spaceused less than limit2

7: memory needs to be monitored warning

8: else

9: memory out of bound CRITICAL

10: end if

The above algorithm is used to calculate the amount of memory space used. It takes the input from the command df -h. This command will return the statistics of all the memory space. Then from the over all statistics the file system is selected. From that column 5 which contains the used space is selected. Then it is compared with the limit levels. It returns three statuses. Ok, if the space is within limit. Warning, if the limit is between minimum and maximum limit, Critical, if it has exceeded the maximum limit.

**5.5.2 ALGORITHM TO MONITOR NO OF PROCESSES**

Input: List of active processes

Output: Count of active processes and status

1: GET the process list psaux

2: SELECT wordcount

3: SET wordcount to a

4: PRINT a SET lowerlimit to 500

SET middlelimit to 600

5: if a less than lowerlimit then

6: process count is under limit OK

return 0

elseif

7: a greater than lowerlimit and a less than middlelimit

8: process count needs to be monitored warning

9: return 1

10: else

11: process count out of bound CRITICAL

12: return 2

13: end if

In the above algorithm, the number of running processes are taken as input and displayed. Based on a certain threshold limit, specific warning are returned to the terminal. If the limit is below 500,it is ok. If it is between 500 and 600 then it is in warning stage. If it is above 600, it is in warning stage.

The following pseudocode has the same design because the scripts follow the same template. These services can be monitored in the same way.

**5.5.3 ALGORITHM TO MONITOR PUPPET MASTER**

In the below algorithm, the service query of puppet master is done.

If it returns a value as inactive then the service is down and an exit code 2 is returned to the terminal. If the service returns an active status then the service is running and it returns an exit code 0 to the terminal.

Input: Service query of puppet master

Output: Service status of puppet master

1: GET servicestatus

2: SELECT inactive

3: SET servicestatus to a

4: if a == true then

5: master server is not running CRITICAL

return 2

6: else

7: master is running OK

return 0

8: end if

**5.5.4 ALGORITHM TO MONITOR JENKINS**

In the below algorithm, the service query of jenkins master is done.

If it returns a value as inactive then the service is down and an exit code 2 is returned to the terminal. If the service returns an active status then the service is running and it returns an exit code 0 to the terminal.

Input: Service query of jenkins master

Output: Service status of Jenkins master

1: GET servicestatus

2: SELECT inactive

3: SET servicestatus to a

4: if a == true then

5: master server is not running CRITICAL

return 2

6: else

7: master is running OK

return 0

8: end if

**5.5.5 ALGORITHM TO MONITOR XINETD**

In the below algorithm, the service query of xinetd service is done.

If it returns a value as inactive then the service is down and an exit code 2 is returned to the terminal. If the service returns an active status then the service is running and it returns an exit code 0 to the terminal.

Input: Service query of xinetd service

Output: Service status of xinetd service

1: GET servicestatus

2: SELECT inactive

3: SET servicestatus to a

4: if a == true then

5: master server is not running CRITICAL

return 2

6: else

7: master is running OK

return 0

8: end if

**5.5.6 ALGORITHM TO AUTOMATE JAVA INSTALLATION**

In this algorithm, the java tar file is being extracted and stored in a file. It is then moved to a different directory where all the softwares are stored.

A directory is created to hold the jdk file. The file’s permission is set to root.This means that only the root user can access it. Any user apart from the root user cannot access the file. Then the file is executed using the exec command it is moved into the java path in the bin directory. The java compiler path is set to the javac file. All the files are set to their appropriate file locations from where java will execute effectively. The path set in the .bashrc file will make sure that all the services access java from the correct file locations if there are multiple copies or versions of java. It is possible to have multiple versions of java in the same machine under different users. Hence setting the path in the .bashrc file is very important so that there is no anomaly when others services that require java are executed.

Input: Jdk file

Output: Java installed required host

1: GET jdk f ile

2: SET jdk to f ile

3: SET owner to root

SET pathtoexecute

4: GET f ile

EXECUTE tar f ile

MOVE f ile to =opt

SET :bashrc to pathtoexecute

SET java path to =opt/ file/ java

SET compiler path to =opt/ file/ javac

**CHAPTER 6**

**SYSTEM IMPLEMENTATION**

The Analysis of this Site Reliability Engineering project carries the knowledge about the software tools Nagios and Puppet that have been used for monitoring the system and for automated code deployment process. These tools are used to ensure the implementation of continuous integration and continuous delivery in a effective manner.

This chapter contains the screen shots of the implementation and the results obtained through the implementations.

**6.1 PROJECT MANAGEMENT**

Project management is the phase used to initiate, plan, execute and control the work of a team to achieve the specified goals. This can be done using a project management tool called Jira.

**6.1.1 Jira**

Jira is a proprietary issue tracking product, developed by Atlassian. It provides bug tracking, issue tracking and project management functions. Work process is the development (or move) of an issue through different Statuses amid its lifecycle. Jira as a Project Management tool has 2 modes Kanban and Scrum. Scrum and Kanban are two terms that are often (incorrectly) used interchangeably or thought to be two sides of the same coin. In reality, there are significant differences between these two agile methodologies. Scrum Board, Figure 5.1 a visual representation of the work flow, broken down into manageable chunks called “stories”, with each story moved along the board from the “backlog” (the to-do list), into work-in-progress (WIP) and on to completion. Kanban is also a tool used to organize work for the sake of efficiency.

Like Scrum, Kanban in Figure 6.2 encourages work to be broken down into manageable chunks and uses a Kanban Board (very similar to the Scrum Board) to visualize that work as it progresses through the work flow.

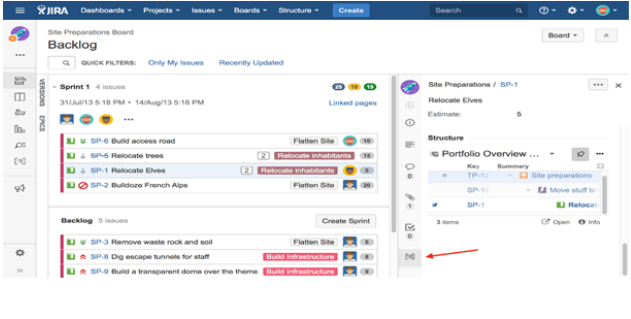


Figure 6.1: Example for Scrum Board

Where Scrum limits the amount of time allowed to accomplish a particular amount of work (by means of sprints), Kanban limits the amount of work allowed in any one condition (only so many tasks can be ongoing, only so many can be on the to-do list).

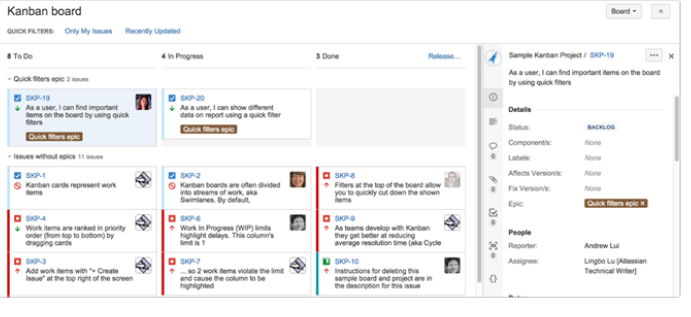


Figure 6.2: Example for Kanban Board

**6.1.2 Features of Jira**

The features of Jira are,

* Issues aroused - Jira lets priorities, assign, track, report and audit the ‘issues’ whatever may be from software bugs and help-desk tickets to project tasks and change requests.
* Reporting and statistics - Customizable reporting allows to monitor the progress of the issues with detailed graphs and charts.
* Workflow on the way - Map the business process with a custom work-flow.
* An extensible platform - Integrate Jira into systems with open API and 100+ free plugins.

**6.1.3 Structure of Jira**

The structure of Jira is shown in Figure 6.3, where it contains the epic, stories/tasks, subtasks.

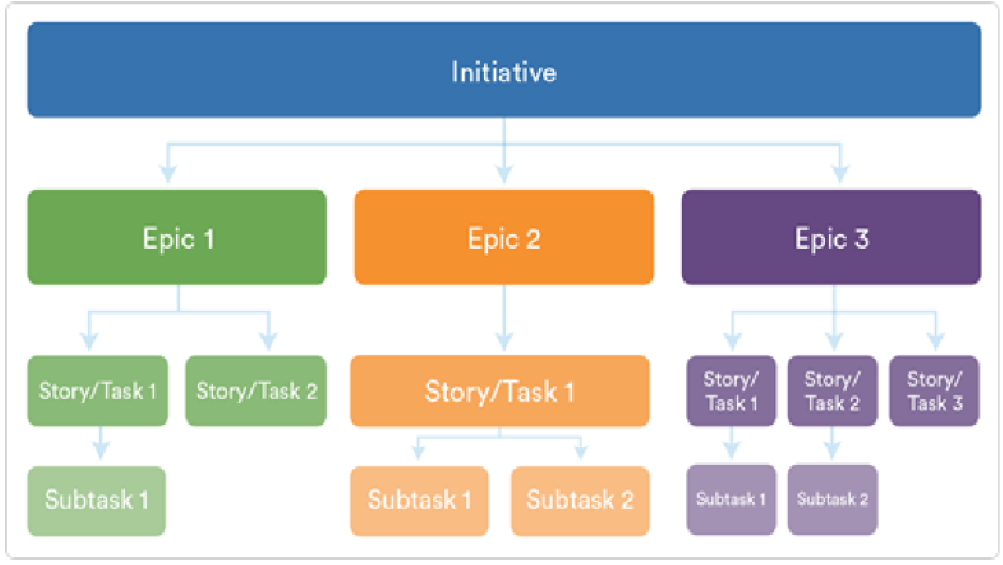


Figure 6.3: Structure of Jira

**6.1.4 Jira Work-flow**

Jira has many work-flows that are inbuilt and can also customize it with own choice. The default one chosen for all the epics is the most simplest one which have 3 steps

* TO DO
* IN PROGRESS
* DONE

Main mission is that how when one person, is a developer and admin has given some work then user will be able to see in login and deadline. And what can be done is just given as a comment in the epic section that coding is started and with the use of JQL it will automatically change the status from “TO DO” to “IN PROGRESS”. And after that the developer will go to Github and develop the code or make the required changes in the Github. And what has to be done is give the keyword used by the main project. Example CICD-1 is the epic then the keyword will be used is CICD-5 done. This is because in configuration part it has been given that Jenkins will take it from Git by checking only the first word of the commit.

**5.1.5 Jira Issue Creations**

The steps to create issues in Jira are as follows,

* Create project by clicking on project menu in menu bar on a page. After clicking the button it will ask for the project name.
* Start making epics (basket of the stories) as shown in Figure 6.4. An epic captures a large body of work. It is essentially a large user story that can be broken down into a number of smaller stories. It may take several sprints to complete an epic.



Figure 6.4: Create Epics

* Create stories and then tasks and sub-tasks as shown in the Figure 6.5. And link it with the required epic.

**Stories**: A story or user story is a software system requirement that is expressed in a few short sentences, ideally using non-technical language. In Jira agile, a story is represented as an issue and individual tasks within the story are represented as sub-tasks.

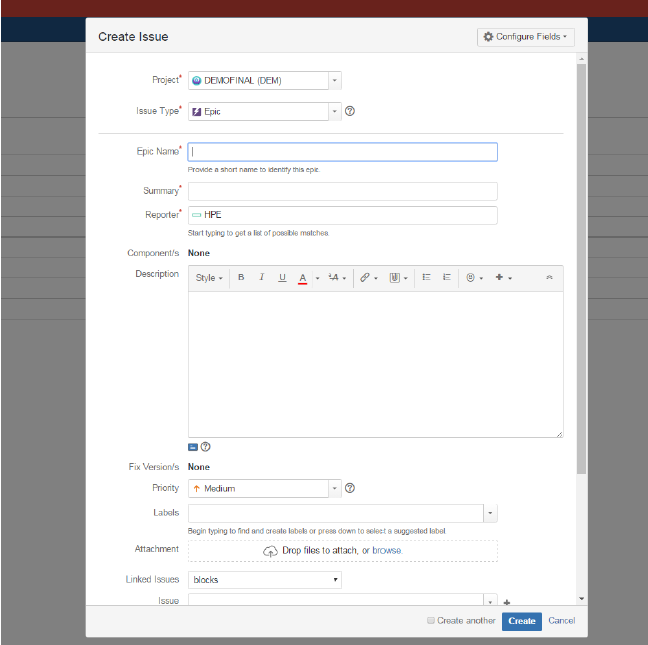


Figure 6.5: Create Stories

* After creating the stories, click on it and assign the task to the users after creating user account in user-management as an administrator can manage users directly in Jira or enable public signup so users can create their own accounts as shown in Figure 6.6.

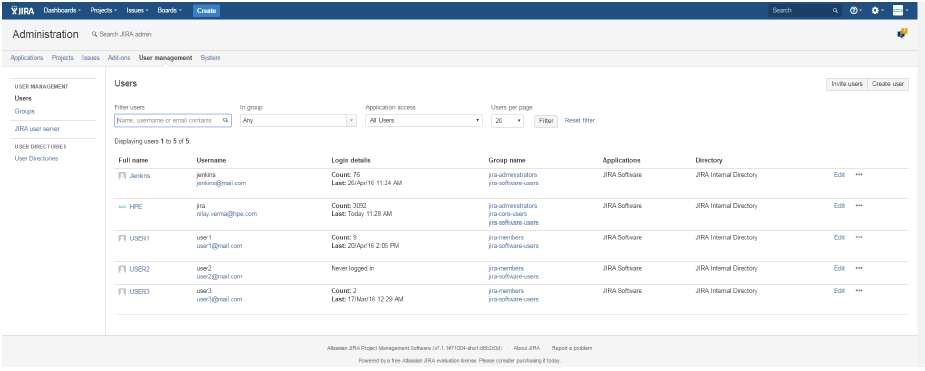


Figure 6.6: User Creation

* Assign the story points. Story point is an arbitrary measure used by scrum teams. This is used to measure the effort required to implement a story. Value of story points may vary according to the teams. E.g., for one team, 1 story point can be 2 weeks and for others it can be 4 weeks.
* Priority is defined as the order in which a issue should be fixed. Higher the priority the sooner the defect should be resolved. Defects that leave the software system unusable are given higher priority over defects that cause a small functionality of the software to fail. Set the priorities and other features.

**5.1.6 Jira Query Language**

The advanced search allows to build structured queries using the Jira Query Language (JQL) to search for issues. Also can specify criteria that cannot be defined in the quick or basic searches (e.g. ORDER BY clause). Note: JQL is not a database query language, even though it uses SQL-like syntax.

A simple query in JQL (also known as a ‘clause’) consists of a field, followed by an operator, followed by one or more values or functions.

Fields- A field in JQL is a word that represents a Jira field (or a custom field that has already been defined in Jira).

Operators- An operator in JQL is one or more symbols or words that compare the value of a field on its left with one or more values (or functions) on its right, such that only true results are retrieved by the clause.

Keywords- A keyword in JQL is a word or phrase that does (or is) any of the following:

* Joins two or more clauses together to form a complex JQL query.
* Alters the logic of one or more clauses.
* Alters the logic of operators.
* Has an explicit definition in a JQL query.
* Performs a specific function that alters the results of a JQL query.

Functions- A function in JQL appears as a word followed by parentheses, which may contain one or more explicit values or Jira fields.

For instance: This query is for changing the status from “To Do” to “In Progress”.

project = “CacheFramework” AND (comment ~‘start’ OR comment ~‘begin’ OR comment ~ ‘begun’) AND (comment ~ ‘code ~’ OR comment ~‘Code ~’) AND status = “ToDo” AND “EpicLink” = CAC-1”

**6.2 SOURCE CODE MANAGER**

A Source Code Manager (SCM) is a software tool used by teams of programmers to manage source code.SCMs are used to track revisions in software. Each revision is given a time-stamp and includes the name of the person is responsible for the change.

Various revisions may be compared, stored and merged with other revisions.

**6.3 GIT**

Git is a site where the transfer of duplicate Git store. It is a Git archive facilitating administration, which offers the greater part of the circulated modification control and source code administration SCM usefulness of Git and in addition including its own elements. Dissimilar to Git, which is entirely a charge line instrument, Github gives an online graphical interface and desktop and also versatile joining. It permits to work together with other individuals on an undertaking. It does that by giving a concentrated area to share the store, an electronic interface to view it and elements like forking, force demands appropriated update control, issues. For configuring the Git in system, need tar or rpm files to install it manually on the very first space.

**6.3.1 Advantages of Git**

Git’s advantages are,

* Free and open source: Git is released under open source license. It is available freely over the internet. As it is an open source, can download its source code and also perform changes according to requirements.
* Fast and small: As most of the operations are performed locally, it gives a huge benefit in terms of speed. Git does not rely on the central server, that is why there is no need to interact with the remote server for every operation. The core part of Git is written in C language, which avoids run time overheads associated with other high-level languages.
* Implicit backup: Data present on any client side mirrors the repository, hence it can be used in the event of a crash or disk corruption.
* No need of powerful hardware: In case of SCM, the central server needs to be powerful enough to serve requests of the entire team. For smaller teams, it is not an issue, but as the team size grows, the hardware limitations of the server can be a performance bottleneck.
* Easier branching: SCM uses cheap copy mechanism, If we create a new branch, it will copy all the codes to the new branch, so it is time-consuming and not efficient. Also, deletion and merging of branches in SCM is complicated and time-consuming.

**6.3.2 Operations in Git**

The main operation in Git.

**Commits**: Commit holds the current state of the repository. A commit is also named by Secure Hash Algorithm1 (SHA1) hash. If a commit has multiple parent commits, then that particular commit has been created by merging two branches.

**Branches**: Branches are used to create another line of development. By default, Git has a master branch. Usually, a branch is created to work on a new feature. Once the feature is completed, it is merged back with the master branch.

**Tags**: Tag assigns a meaningful name with a specific version in the repository. Once a tag is created for a particular commit, even if a new commit is created, it will not be updated. Usually, developers create tags for product releases. The basic operations and their working way is mentioned in Figure 6.7.



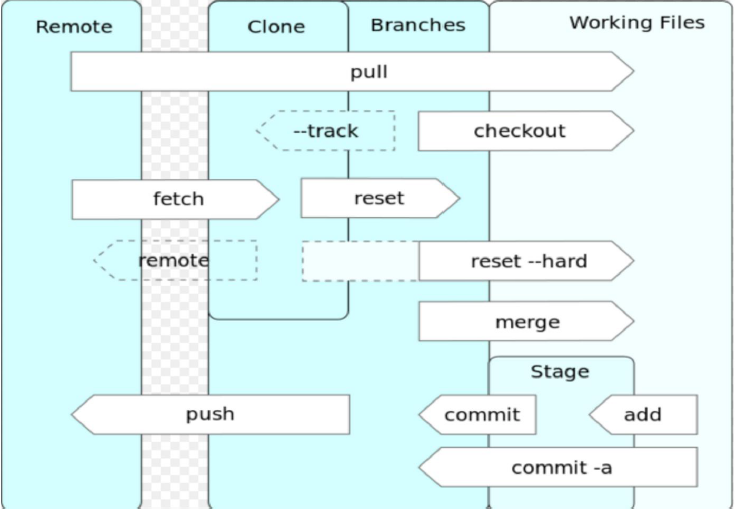


Figure 6.7: Git Operations

* Clone: Clone operation creates the instance of the repository. Clone operation not only checks out the working copy, but it also mirrors the complete repository.
* Pull: Pull operation copies the changes from a remote repository instance to a local one. The pull operation is used for synchronization between two repository instances.
* Push: Push operation copies changes from a local repository instance to a remote one. This is used to store the changes permanently into the Git repository.
* Head: Head is a pointer, which always points to the latest commit in the branch. Whenever a commit is made, head is updated with the latest commit.

**6.3.3 Gitlabs**

Gitlabs is an open-source ‘clone’ of Github. It’s functionality is very similar to that of Github, but because it is open-source it means that we can use it on our servers for free. Gitlabs allows us to create and modify multiple Git.

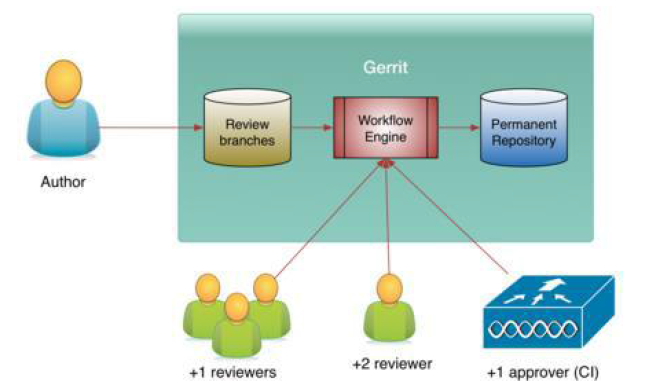


Figure 6.8: Gitlabs Workflow

repositories online. It allows admins to manage each repository, the users, and their groups. It has a simple user interface that is very easy to use.

**6.3.4 Working with Gitlabs**

To work on a Git project locally (from own computer),first it needed to be cloned. To do this, sign in to Gitlabs. From dashboard, click on the project that has to be cloned. To work in the project, copy a link to the Git repository through a Secure Shell (SSH) or a Hyper Text Transfer Protocol.

Secure (HTTPS) and it is shown in the pictorial representation in Figure 6.8.

**6.4 PRODUCT BUILDING**

The building of the application is carried using a tool called Maven. Maven provides developers a complete build life-cycle framework. A build tool takes care of everything for building a process. It does following:

* Generates source code (if auto-generated code is used)
* Generates documentation from source code
* Compiles source code
* Packages compiled code into Java ARchive (JAR) of ZIP file
* Installs the packaged code in local repository, server repository or central repository

**6.5 MAVEN**

Maven is a project management and comprehension tool. Maven provides developers a complete build life-cycle framework. Development team can automate the project’s build infrastructure in almost no time as Maven uses a standard directory layout and a default build life-cycle.

In case of multiple development teams environment, Maven can set-up the way to work as per standards in a very short time. As most of the project setups are simple and reusable, Maven makes life of developer easy while creating reports, checks, build and testing automation setups.

Maven provides developers ways to manage following:

* Builds
* Documentation
* Reporting
* Dependencies
* SCMs
* Releases
* Distribution
* Mailing list

To summarize, Maven simplifies and standardizes the project build process. It handles compilation, distribution, documentation, team collaboration and other tasks seamlessly. Maven increases re-usability and takes care of most of build related tasks.

The detail structure of Maven is explained in Figure 6.9.

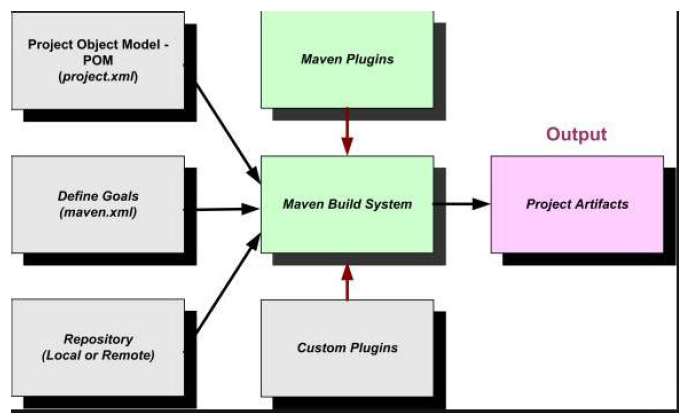


Figure 6.9: Maven Structure

**6.5.1 Maven Objective**

Maven primary goal is to provide developer -

* A comprehensive model for projects which is reusable,maintainable,

and easier to comprehend.

* Plugins or tools that interact with this declarative model. Maven project structure and contents are declared in an eXtensive Mark-up Language (XML) file, pom.xml referred as Project Object Model.

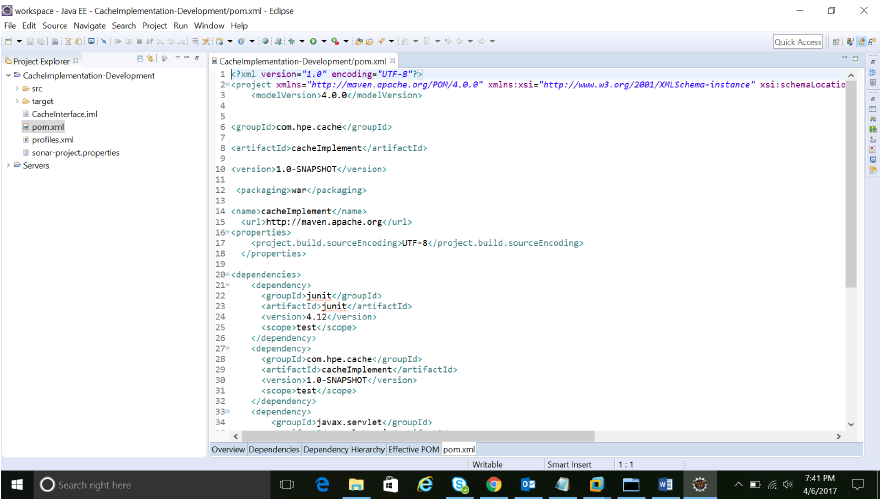


Figure 6.11: Maven pom.xml

(POM), which is the fundamental unit of the entire Maven system. As Shown in Figure 6.11 follow the Maven POM.xml section for more detail.

**6.5.3 Maven pom.xml File**

POM is an acronym for Project Object Model. The pom.xml file contains information of project and configuration information for the maven to build the project such as dependencies, build directory, source directory, test source directory, plugin, goals etc. Maven reads the pom.xml file shown in Figure 6.11, then executes the goal. Before maven 2, it was named as project.xml file. But, since maven 2 (also in maven 3), it is renamed as pom.xml.

**6.5.4 Elements of Maven pom.xml File**

For creating the simple pom.xml file, following elements need to have:

* Project: It is the root element of pom.xml file.
* ModelVersion: It is the sub element of project. It specifies the model version.
* GroupID: It is the sub element of project. It specifies the id for the project group.
* ArtifactID: It is the sub element of project. It specifies the id for the artifact (project). An artifact is something that is either produced or used by a project. Examples of artifacts produced by Maven for a project include: JARs, source and binary distributions, and Web Application aRchive (WARs).
* Version: It is the sub element of project. It specifies the version of the artifact under given group.
* Packaging: Defines packaging type such as jar, war etc.
* Name: Defines name of the maven project.
* URL: Defines Universal Resource Locator (URL) of the project.
* Dependencies: Defines dependencies for this project.
* Dependency: Defines a dependency. It is used inside dependencies.
* Scope: Defines scope for this maven project. It can be compile, provided, run-time, test and system.

**6.5.5 Maven repository**

Maven repository are of two types:

* Local
* Central
* Local Repository: Maven local repository is a folder location on the physical machine. It gets created when the maven command run for the first time. Maven local repository keeps the projects dependencies (library JARs, plugin jars etc.,). When the Maven build runs, Maven automatically downloads all the dependency jars into the local repository. It helps to avoid references to dependencies

stored on remote machine every time a project is build.

* Central Repository: Maven central repository is repository

provided by Maven community. It contains a large number

of commonly used libraries. When Maven does not find any

dependency in local repository, it starts searching in central

repository.

5.6 PRODUCT TESTING

Testing is a process of executing a program or application with the

intent of finding the software bugs. To check the code quality SonarQube is

used.

41

5.6.1 SonarQube

It is an open source tool for continuous inspection of code quality It

is a code quality management platform that allows developer team to manage,

track and eventually improve the quality of source code. It provides fully

automated analysis. It can be integrated with Git, Jenkins. SonarQube is

formally known as sonar. An open source product which is used to gather

several matrices about code quality and put them all in a single dashboard and

provide some tips to help making code better, more sustainable, more reliable

and less bugged.

5.6.2 Why SonarQube?

Code quality analysis helps us to make our code.

\_ Less error prone

\_ More sustainable

\_ More reliable

\_ More readable

SonarQube generate reports on

\_ Duplicate code : It means that the sequence of source code that

occurs more than once that is having duplicate lines of codes.

\_ Coding standard: It is a set of rules or guidelines that is used when

writing the source code for a computer program.

42

\_ Code coverage: Code coverage is a measure that is used to describe

that is how much percentage of our code is accessed by the test

coverage. A program with high code coverage has been more

thoroughly tested and has a lower chance of containing software

bugs than a program with low code coverage. One or more coverage

criteria are used

Basic coverage criteria

– Functional coverage: Has each function in the program been

called?

– Statement coverage : Has each statement in the program been

executed?

– Branch coverage : Has each branch or each control structure

been executed? For example: - if there is an IF statement, will

have to check whether both true and false branches has been

executed or not.

– Condition coverage: Checks for the potential bugs, Comments,

designs and architecture.

5.6.3 SonarQube Structure

It is a simple architecture consists of three components.

A set of source code analyzer (squid-code analyzer) that are grouped in a maven

plugins and are triggered on demand. It is capable to analyze maven projects.

A database to persist the result of the analysis and to keep historical

analysis for time machine. 5 database engines are currently supported Oracle,

MySQL, derby. PostgreSQL, MSSQL server.

43

A web reporting tool to display code quality dashboards on projects.

Enable Hudson plugins on our job. Sonar will locally analyze code and

generate reports from many analyzer. Sonar will push those reports to the sonar

dashboard

5.7 INTEGRATION SERVER

The integration server section contains the details of coninous

integration, Jenkins, Maven plugins, Git integration, Maven integration, Maven

settings and SonarQube integration with Jenkins.

5.7.1 Continuous Integration

Continuous Integration is a development practice that requires

developers to integrate code into a shared repository at regular intervals. This

concept was meant to remove the problem of finding later occurrence of issues

in the build life-cycle. Continuous integration requires the developers to have

frequent builds. The common practice is that whenever a code commit occurs,

a build should be triggered.

5.7.2 Jenkins

Jenkins is one of the open source persistent reconciliation device

written in Java. The undertaking was forked from Hudson after a question

with Oracle. Jenkins gives consistent mix administrations to programming

improvement. It is a server-based framework running in a servlet compartment,

for example, Apache Tomcat and other separate holders too. Along these

lines, It bolsters SCM devices including AccuRev, Cuncurrent Version Systems

(CVS), subversion, Git, Mercurial, Perforce, Clearcase and Rational Team

44

Figure 5.12: Jenkins login page

Concert (RTC) and can execute Apache Ant and Apache Maven based tasks.

Additionally self-assertive shell scripts and Windows clump orders. Jenkins is

free delicate plugins have been discharged for Jenkins that extend its utilization

to ventures written in dialects other than Java. Modules are accessible for

incorporating Jenkins with most form control frameworks and huge databases.

Numerous form apparatuses are upheld by means of their individual modules[4].

The Jenkins URL points to localhost. Once Jenkins is up and

running, one can access Jenkins from the link http://ipaddress:8080. This link

will bring up the Jenkins Login page as shown in Figure 5.12.

5.7.3 Manage Plugins

Plugins are add-ons that allow Jenkins to interact with a variety of

outside software or otherwise extend its innate abilities. Jenkins can be extended

via additional plug-ins with more functionality. Can configure the plug-ins via

the Manage Jenkins Manager Plugins link given in Figure 5.13. To install

plugins in Jenkins select use the Manage Jenkins Manager Plugins link and

search for the plugin and install. Select it from the list and select to install it and

restart Jenkins. Figure 5.14 shows the installation of the plugins for Git.

45

Figure 5.13: Installing Plugins for Tools

Figure 5.14: Integration with tools

5.7.4 Git Integration

The Git integration steps are as follows,

\_ In the Jenkins Dashboard (Home screen), click the Manage Jenkins

option on the left hand side.

\_ In the next screen, click the Manage Plugins option.

\_ In the next screen, click the Available tab. This tab will give a list of

46

Figure 5.15: Fetching Project from Git

plugins which are available for downloading. In the ‘Filter’ tab type

‘Git plugin’.

\_ After Jenkins is restarted, Git will be available as an option whilst

configuring jobs. To verify, click on New Item in the menu options

for Jenkins. Then enter a name for a job, in the following case, the

name entered is ‘Cache’. Select ‘maven project’ as the item type.

Click the Ok button.

\_ As the Figure 5.15 shows, In the next screen, browse to the Source

code Management section, can see ‘Git’ as an option[5].

5.7.5 Maven Integration

Jenkins provides a job type dedicated to Maven 2/3. This job type

integrates Jenkins deeply with Maven 2/3 and provides the following benefits

compared to the more generic free-style software project.

47

\_ Jenkins parses Maven POMs to obtain much of the information

needed to do its work. As a result, the amount of configuration is

drastically reduced.

\_ Jenkins listens to Maven execution and figures out what should be

done when on its own.

\_ Jenkins automatically creates project dependencies between projects

which declare SNAPSHOT dependencies between each other.

5.7.6 Maven Settings

Maven should have have the following settings,

\_ In the Jenkins dashboard (Home screen), click Manage Jenkins from

the left-hand side menu.

\_ Then, click on ‘Configure System’ from the right hand side.

\_ In the Configure system screen, scroll down till the Maven section

and then click on the ‘Add Maven’ button.

\_ Add any name for the setting and the location of the MAVENHOME.

\_ Then, click on the ‘Save’ button at the end of the screen.

As shown in Figure 5.16 the maven build commands has to be given for the

Jenkins to work on it’s own based on the given command[6].

48

Figure 5.16: Build Package

5.7.7 Integrate SonarQube with Jenkins

To integrate jenkins with SonarQube the following steps has to be

followed.

Jenkins > manage Jenkins > manage plugins > available as shown in Figure

5.17.

Figure 5.17: Plugins Page

49

Figure 5.18: Jenkins Plugins for SonarQube

Search SonarQube plugin and click on install without restart, after

that restart Jenkins server as shown in Figure 5.18

5.7.8 SonarQubeWeb interface

For generating service authentication token go to SonarQube web

interface. Administrator > my account > security > Generate tokens >add

any token name in enter token name section >copy and then paste into service

authentication section in Jenkins[7] Adding SonarQube Server: Manage

Jenkins > Configure System > Add SonarQube as shown in Figure 5.19

50

Figure 5.19: Adding SonarQube Server

5.7.9 Sonar Properties

Sonar-project.properties in Git is added as shown in Figure 5.20

Create new file > sonar-project.properties

Figure 5.20: Sonar.properties in Git

Add these following fields in sonar-project.properties

Sonar.projectkey(an unique name of the project)

Sonar.projectName(the project name that needs to be displayed in SonarQube

dashboard)

Sonar.projectVersion(an unique project version) Sonar.sources(src)

51

Figure 5.21: Jenkins Build Console

Sonar.sourceEncoding(which encoding version is used e.g.:- UTF-8)

Sonar.language(programming language that is used).

After the proper integration in Jenkins with SonarQube, Then click

on Build Now. Once the build gets success we should be able to see the analysis

report on SonarQube interface with proper project key, name and version as

shown in Figure 5.21.

5.7.10 SonarQube Result

\_ Technical debt: Also known as design debt or code debt is “a

concept in programming that reflects the extra development work

that arises when code that is easy to implement in the short run is

used instead of applying the best overall solution”.

\_ Duplications: Duplicate code is a computer programming term for

a sequence of source code that occurs more than once, either within

a program or across different programs owned or maintained by the

52

same entity. Duplicate code is generally considered undesirable for

a number of reasons.

Figure 5.22: Test Result

The result page will display the following details as shown in Figure

5.22.

\_ Structure: Structure is the section where the SonarQube checks for

the structure of the Java program and also counts the line and checks

whether those lines are executed.

5.8 CONTINUOUS

**6.1 INSTALLATION OF NAGIOS**

NAGIOS is a monitoring tool that monitors all the activities of the hosts and services that are configured on it. It requires the following three dependent services to perform the monitoring actions:

* HTTPD: It is a software program that runs in the background of a web server and waits for the incoming server requests.
* XINETD: It is an open-source daemon which runs on Linux and Unix systems and manages Internet-based connectivity.
* NRPE: It is used to remotely execute Nagios Plugins on multiple remote machines.

**6.1.1 Installation of httpd**

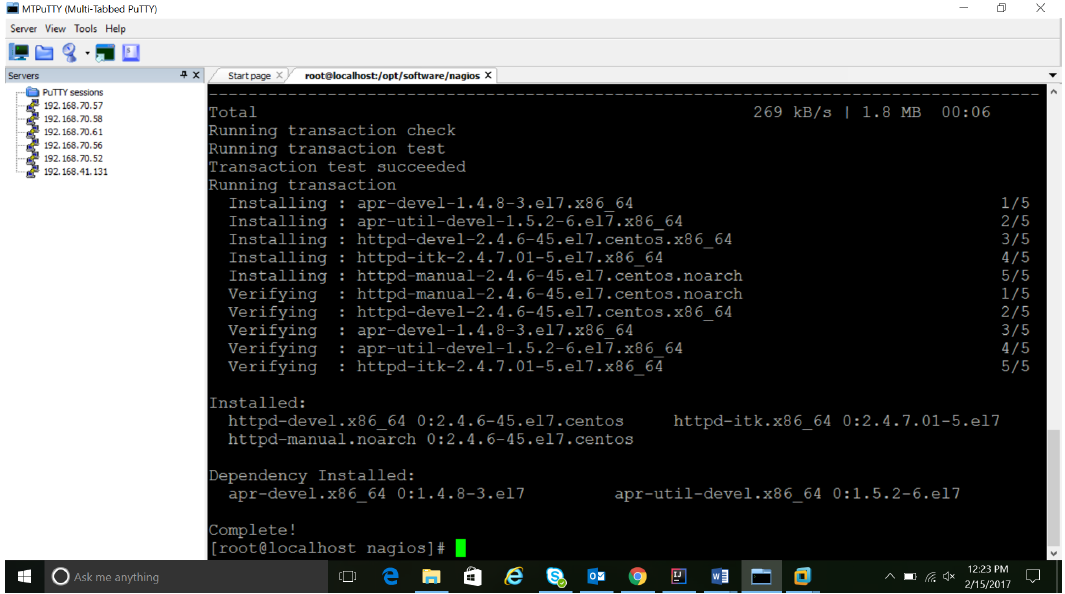


Figure 6.1: Installation of httpd

Httpd is a software program that runs in the background of a web server and waits for the incoming server requests or response. Nagios includes multiple nodes that are configured on it. Each node interacts with the server with an incoming response about the state of the machine or the service running on the server. Figure 5.1 depicts the installation of the httpd packages. The command used is yum install httpd . It will resolve and download all the necessary packages and install them.

**6.1.2 Installation of Nagios**

This command complies all the binaries that are part of the NAGIOS package along with its dependencies

* make all

The following configuration commands that are required to run the NAGIOS monitoring service successfully. These commands are included for installing the dependencies too for running the monitoring service. Now we can run these make commands to install Nagios, init scripts, and sample configuration files:

* make install
* make install-commandmode
* make install-init
* make install-config
* make install-webconf

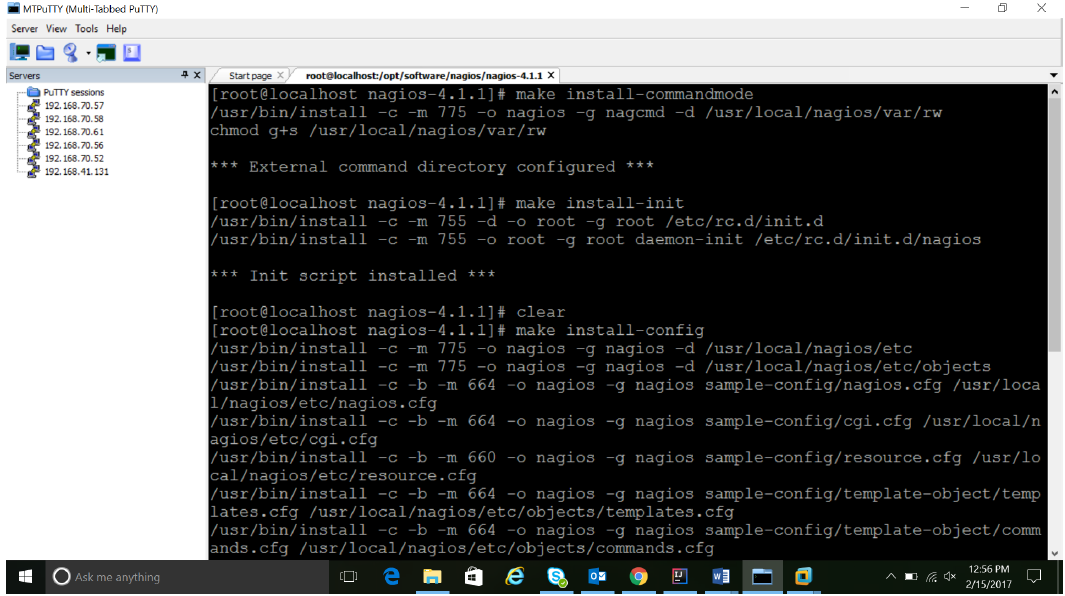


Figure 6.2: Installation of nagios

Figures 6.2 and 6.3shows that the nagios configuration files have been installed and the httpd service has been configured for nagios. Now Nagios plugins can be written and configured for the services. The following command configures the httpd and nagios integration.

* usermod -G nagcmd apache

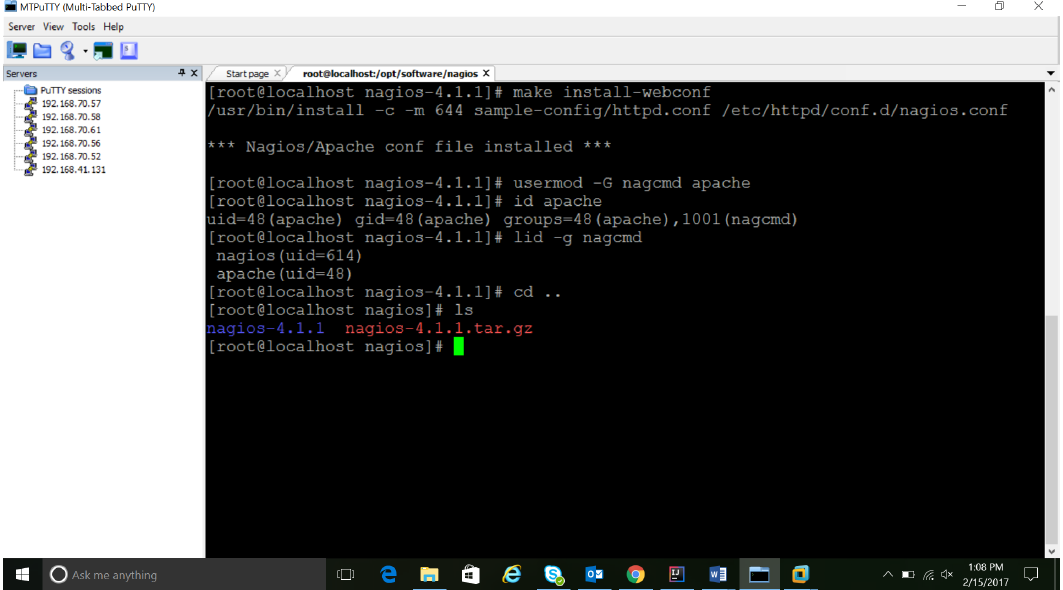


Figure 6.3: Installation of nagios-2

**6.1.3 Installation of xinetd and nrpe**

The following commands are used to build and install NRPE and its related XINETD startup scripts:

* make all
* make install

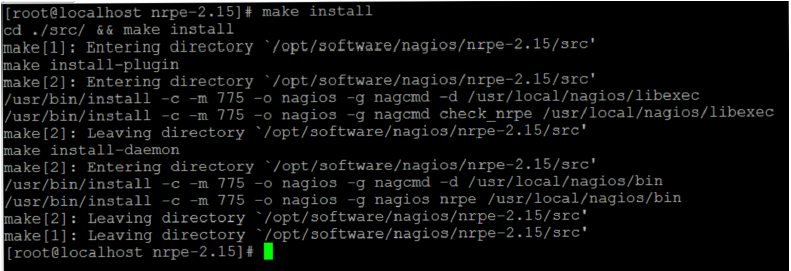


Figure 6.4: Installation of XINETD

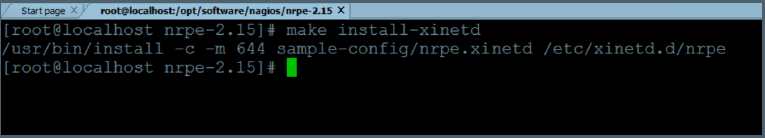


Figure 6.5: Installation of XINETD-2

Figures 6.4, 6.5 and 6.6 show the commands that are used to install the xinetd service from the package. The make install command will install only the xinetd files from the whole network package. Only files related to the daemon process that needs to run everytime the service is started is installed and configured.

* make install-xinetd
* make install-daemon-config



Figure 6.6: Installation of XINETD-3

**6.2 CREATION OF CUSTOM PLUGINS**

Nagios comes in with a set of in-built plugins that can be used to monitor a host or a service based on certain parameters. Nagios also provides the feasibility to write custom scripts that can be used to monitor the system based on parameters that the administrator needs. Plugins are written in the form of bash scripts that are executed by the shell to which they are bonded to. In this case, the normal Bash shell is used for executing the plugins.



Figure 6.7: List of plugins

Figure 6.7 shows a list of plugins that can be used to monitor the System. All default plugins are available in /usr/lib64/nagios/plugins/ directory.

Custom plugins can also be added and saved in this same directory since the NRPE service will be reading the scripts from this directory only.

Nagios consists of four specific exit codes that help the administrator in understanding the state of the system. These codes will be used along with a return message that will help the administrator in understanding the exact problem that is occurring in the system. The exit codes are as follows:

* EXIT 0 : Specifies that everything is ok.
* EXIT 1 : A warning is given indicating that the system needs attention
* EXIT 2 : An alert is given indicating that immediate action needs to be taken.
* EXIT 3 : It specifies an unknown state of the system.

Some of the custom plugins have been specified in this chapter below.

**6.2.1 Plugin to check used space**

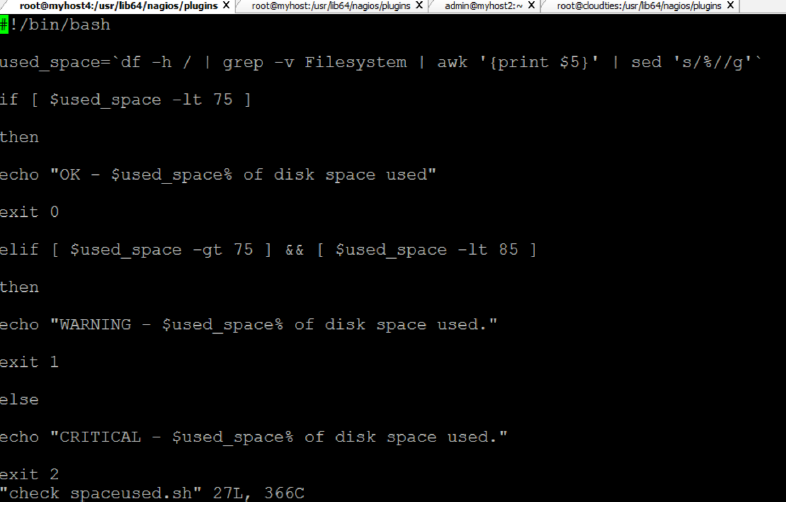
****

Figure 6.8: Plugin to check space used

Figure 6.8 shows the script for the plugin that is used to show whether the used disk space is getting over or not. Different warning levels are set and based on the disk space a warning is sent to the terminal. If the disk space is above the safety limit, then a warning is sent with an exit code 2 that depicts that immediate action needs to be taken on it.

**6.2.2 Plugin to check total processes**

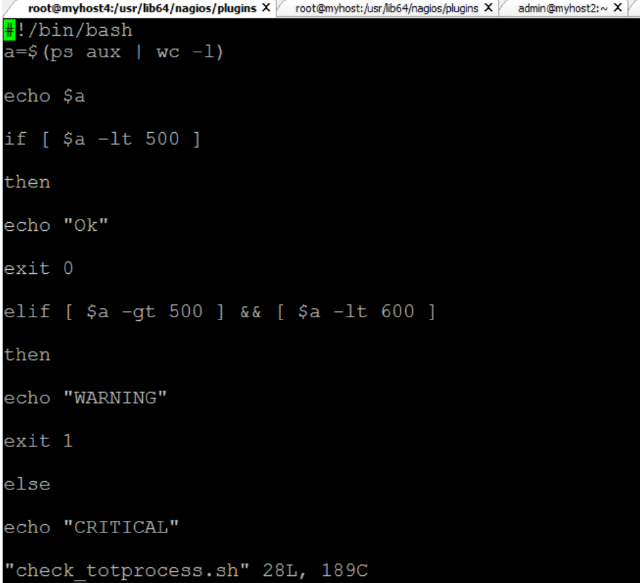


Figure 6.9: Plugin to check total running processes

Figure 6.9 shows the script for the plugin that is used to calculate the total number of processes that are running currently on the host. This allow the administrator to keep track of the processes and make sure that the host is not overloaded with a lot of processes. This script returns the number of processes and also return the exit code based on the threshold level that has been specified by the administrator.

**6.2.3 Plugin to check total users**

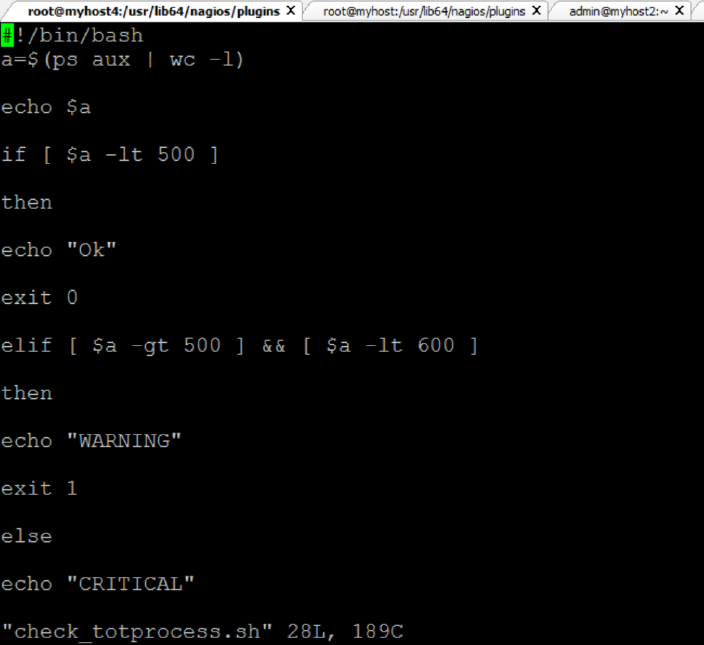


Figure 6.10: Plugin to check total active users

Figure 6.10 shows the script for the plugin that is used to calculate the total number of active users who are currently operating on the particular host. This allow the administrator to keep track of the number of logins and make sure that the host is not overloaded with a lot of users running a lot of processes on a single host at the same time.

This script returns the number of active users and also return the exit code based on the threshold level that has been specified by the administrator.

It returns a code 0 if the number of users are within the allowed limit. It returns an exit code 1 if the number of users are above the safety limit and it will return an exit code 2 if the number of user are above the specified limit.

**6.2.4 Plugin to check artifactory status**



Figure 6.11: Plugin to check artifactory status

Figure 6.11 shows the script for the plugin that is used to determine the status of the service. This allow the administrator to keep track of the service and determine whether the service is running or if it is down. To determine the status the port number of jfrog (8081) should be queried and from there it needs to be determined whether jfrog is listening on that particular port or not. If it is listening on port 8081, then it means that the service is up and running. This plugin can be easily specified because the condition associated with it is very concrete. Jfrog has been specified to run on port 8081 and if there is no service running on port 8081, then it means that Jfrog is down. At any point of time if the service is down, an exit code 2 is returned with an alert message specifying that the service is down.

**6.2.5 Plugin to check puppet master status**



Figure 6.12: Plugin to check puppet master status

Figure 6.12 shows the script for the plugin that is used to determine the status of the service. This allow the administrator to keep track of the service and determine whether the service is running or if it is down. At any point of time if the service is down, an exit code 2 is returned with an alert message specifying that the service is down. This template can be used to write the check plugins for various services like Jenkins, Puppet slave, xinetd, jira, etc.

This template can be used for services that return a service status whenever the service ” service name” status command is executed. Using the grep command any keyword that is unique to a service start or a downtime is used to make sure that system state is captured and returned to the nagios main terminal. The result is returned with the respective exit codes, based on the state of the service.

Similarly, various kinds of scripts can be written to monitor the different states of a host or a service. Plugins can be written to monitor the load of a system, the memory space used, the network status, and status of services like ssh, ntp, ftp, root partitioning and ping.

**6.3 CONFIGURATION OF MASTER AND SLAVE NAGIOS SERVERS**

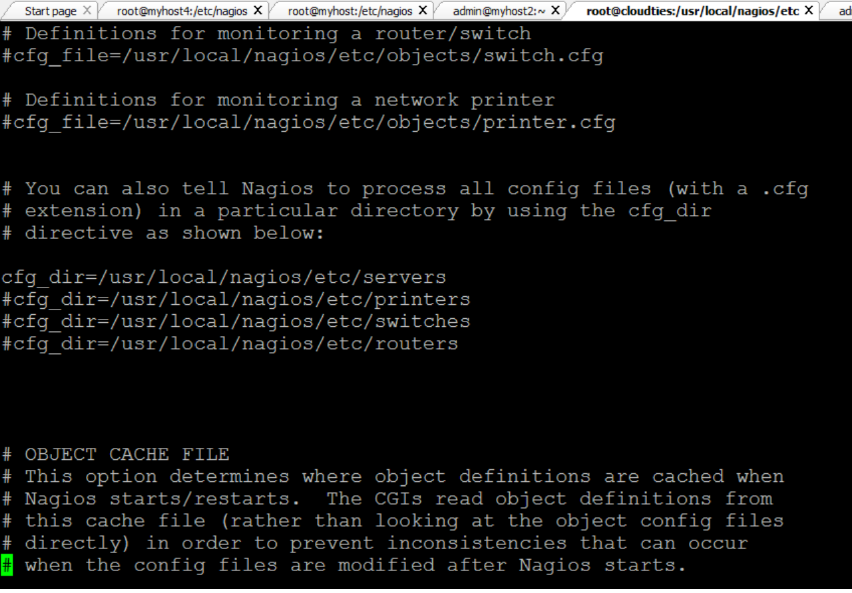


Figure 6.13: Definition of server host type.

The nagios architecture has a master and multiple slaves. The master controls the monitoring service whereas the NRPE service which is installed on the slaves returns the state of the hosts and services back to the master server. The service can be configured to monitor different types of hosts like servers, switches, printers and routers. This can be configured in the done in the nagios.cfg file which contains the path of all the types of hosts.

If any host type is activated then the path related to that particular is created and the hosts can be configured. Here in this configuration the server host type has been uncommented since the scope of the project doesn’t require routers, printers and switches. If printers too need to be configured then the directory related to printer should be configured and the related plugins should be written.

**6.3.1 Definition of command in slave**

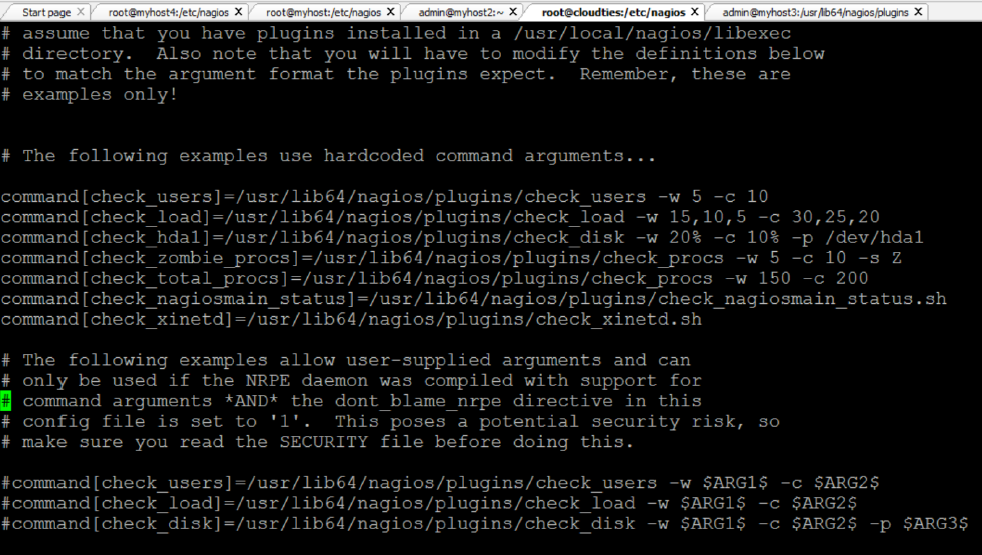


Figure 6.14: Defining command in slave

The command that needs to be executed has to be defined in the slave machine. It is defined in the nrpe.cfg file which is the configuration file for NRPE service. The NRPE service will pick up the command definition and perform the actions specified in the command definition in the master server configuration. Then it will return the state of the host or the service.

Figure 6.14 shows the commands defined in the nrpe.cfg file which serves as the configuration file of NRPE service in the slave servers. All the commands related to that slave are defined in this file.

**6.3.2 Definition of commands in master**

The plugins that have been written need to be executed in the form of a command and hence the commands are defined in the master server in a configuration file called commands.cfg . This file contains all the commands that are executed in the entire network. There can be multiple commands related to multiple hosts and all these commands are stored in this configuration file.

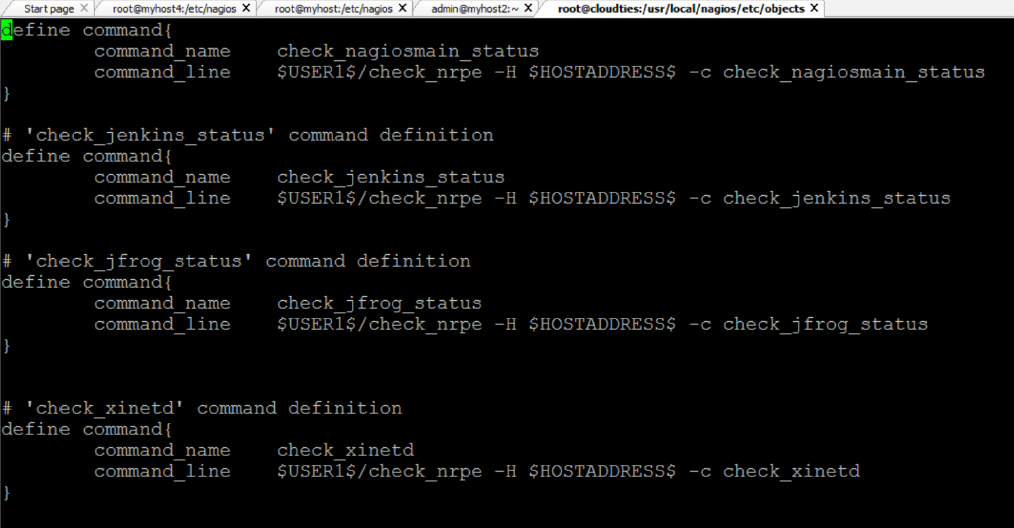


Figure 6.15: Defining command in master-1

The commands defined in Figure 5.15 and Figure 5.16 are picked up by the NRPE service on the slave and based on the path the plugins related to the command are executed. Based on the command execution the result is returned to the terminal of the administrator which in turn is displayed on the administrator console. The configuration file contains commands to monitor the services like puppet master, jenkins, puppet slaves, jfrog, xinetd, nagios master server, jfrog and maven.

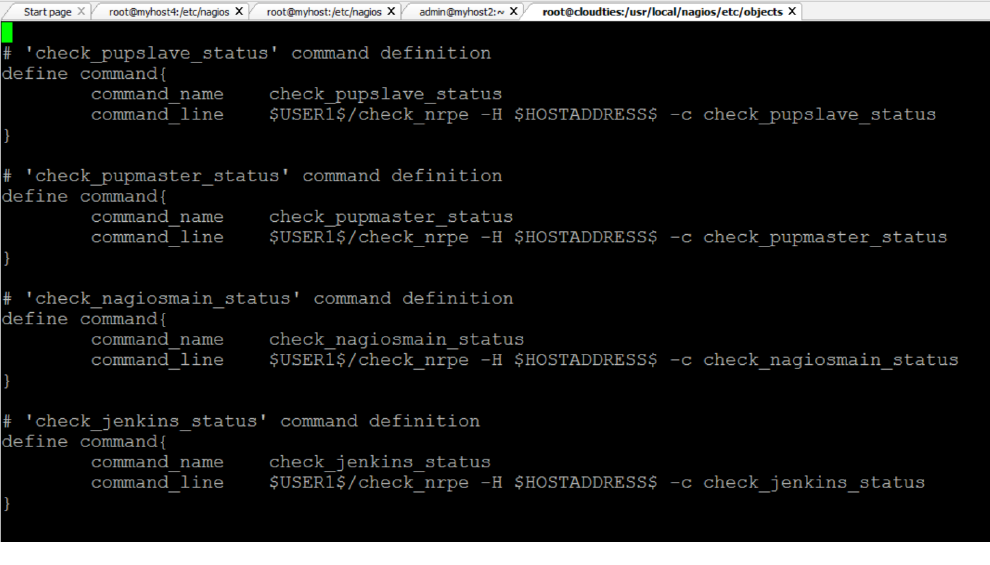


Figure 6.16: Defining command in master-2

**6.4 TERMINAL OUTPUT OF HOSTS AND SERVICES**

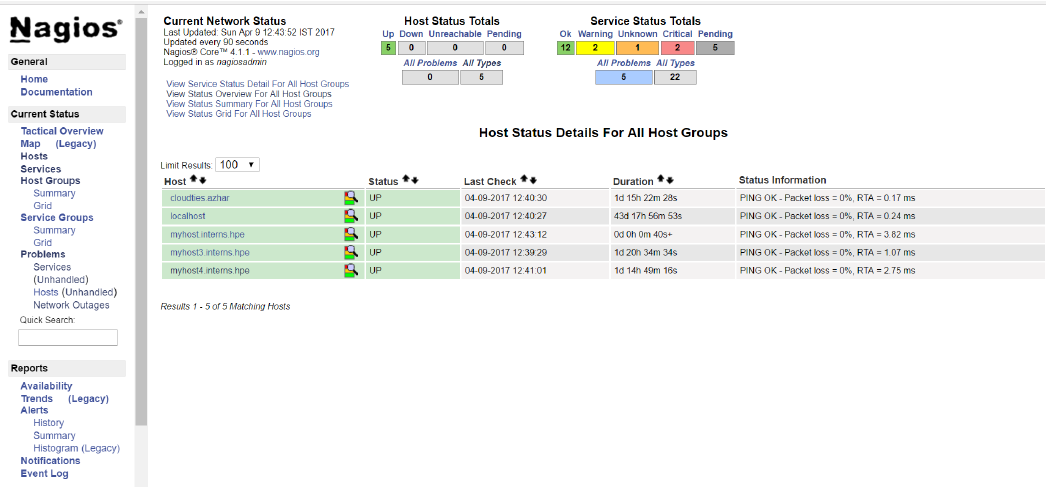


Figure 6.17: Console output of hosts

Figure 6.17 shows the terminal output for the hosts. The console shows the output of servers that have been configured on the network. It shows the states of the servers on the network.

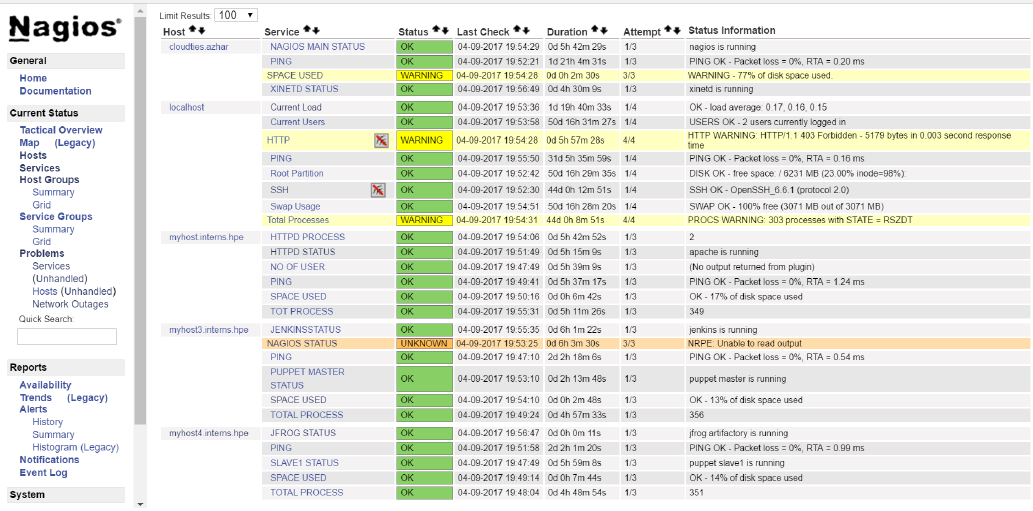


Figure 6.18: Console output of services

Figure 6.18 shows the terminal output for the services. The console shows the output of services that have been configured on the network. It shows the states of the services on the network.

The Figures 6.17 and 6.18 show the outputs of the Nagios administrator console. This is the place where all the services and hosts are monitored and checked for problems. Any kind of defects can be identified by writing appropriate plugins for it. Scheduled checks for services at regular intervals can be carried out notifications for specific services can be switched on and switched off on the basis of priority.

The following section will contain the screen shots of the installation

and the working of puppet master and slave servers.

**6.5 INSTALLATION OF PUPPET**

A prerequisite for the puppet installation is that the server should be able to resolve each others host name. This can be done in two ways:

* Using DNS service on the servers by installing the NAMED service. It provides the dns resolution between two servers.
* By specifying the hostname and corresponding IP address in the hosts file in /etc/hosts directory. This will help the hosts to resolve each others Hostnames.

**6.5.1 Installation of puppet master**

The puppet master can be installed by using the command yum install puppet-server. This will download and install all the dependent packages needed to configure and run the puppet master server. Yum package installer will automatically install all the dependent files and repositories needed for the puppet master service. After the installation is completed, the master server has to be configured to receive all the requests and perform the processes required.

As depicted in figure 5.19, the master server has to identify itself as the master and this can be done by specifying the alternate host names and the master certificate names in the puppet.conf file in /etc/puppet/puppet.conf directory.

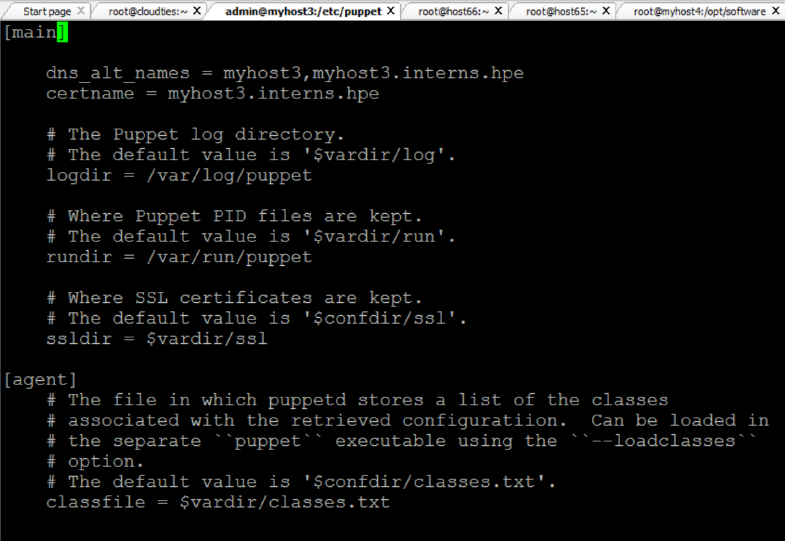


Figure 6.19: Defining puppet master

Now that the master is running, a certificate needs to be created for the master using which it will identify itself as the master to the slaves. To create the master certificate run the below command:

* puppet master –verbose –no- daemonize

This command creates the master certificate and shares it with all the slaves to identify itself as the master.

**6.5.2 Installation of puppet slave**

The puppet slaves can be installed by using the command yum install puppet. This will download and install all the dependent packages needed to configure and run the puppet slave service. After the installation is completed, the slave has to be configured to perform the processes required. As depicted in figure 6.20, the slave server has to identify itself as the slave and this can be done by specifying the server host name in the agent section in the puppet.conf file in /etc/puppet/puppet.conf directory.

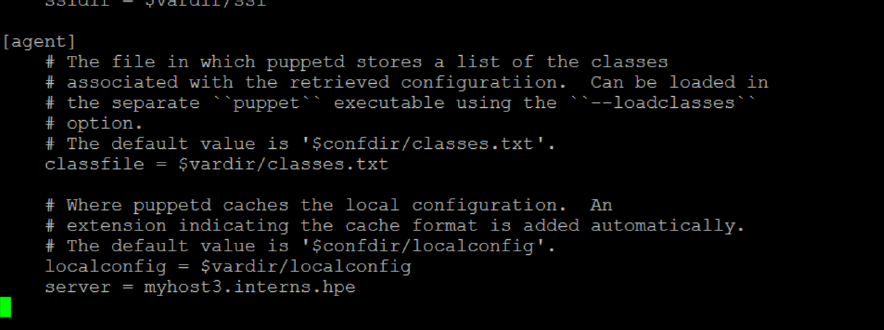


Figure 6.20: Defining puppet slave

After adding the slave configuration the services in the master and slaves are started. This will automatically create the certificates for the slaves.

The certificates are signed by the master using the following commands:

* puppet cert list
* puppet cert sign client.host.local

The final connection is tested using the following command which is also used to initiate the puppet scripts to perform the tasks specified in the manifests. This command is used for initiating all the combined scripts. It triggers the flow of automated tasks through the scripts. The command to execute the scripts is:

* puppet agent -t

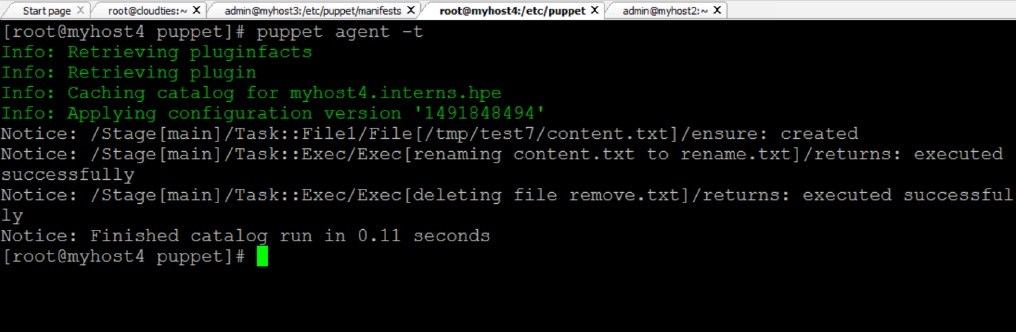


Figure 6.21: Execution command of scripts.

Figure 6.21 depicts the result of a successful test run of the puppet script.

**6.6 PUPPET SCRIPTS FOR AUTOMATION**

Puppet has it sown declarative language and a set of attributes and resource types that helps the administrator to write the script by redirecting the flow of the events that needs to take place. The scripts are written inside a directory called as Modules. The modules directory contains the set of tasks or scripts that need to be executed. All the tasks are defined inside a directory called as Manifests which is recognized by the puppet master as the source of the scripts.

Each module consists of a set of tasks that need to be automatically performed by the tool. Multiple scripts can be written for multiple nodes and it follows a strict directory structure. All the tasks should be included inside the manifests directory.

All the scripts that are written inside the manifests in the modules directory are called into the site.pp file . This file is present in the main manifests directory and is like the junction point for all the scripts written in modules. All the tasks are called in the site.pp file and based on the node defined in it the tasks are performed when a slave initiated the puppet master.

In puppet, the process initialization is always done by the slave.

**6.6.1 Automated deployment script**

Figure 6.22 shows the script for the automated deployment of the war package from the artifactory. The package is deployment using tomcat server.

The script shutdowns the server initially. Then the war package is deployed from the artifactory and moved into the webapps directory of the tomcat server. Then the tomcat server is restart. This method will automatically deploy the war package onto the server.

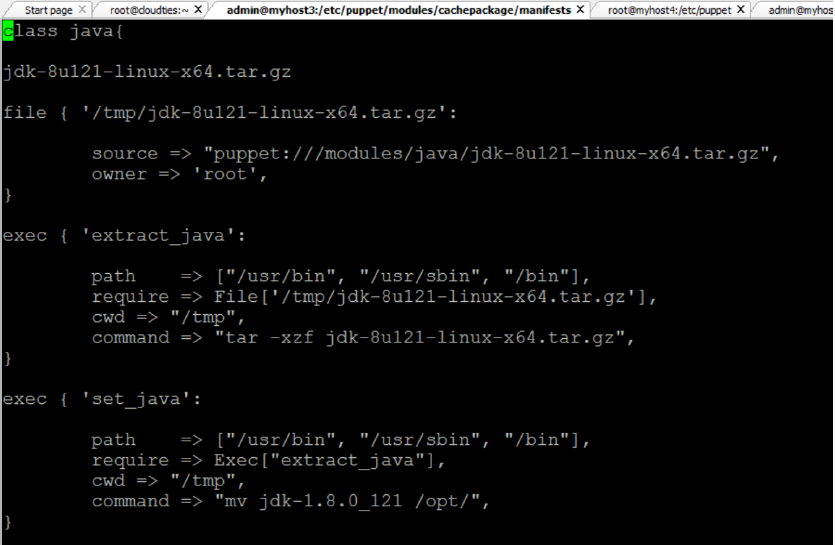


Figure 6.22: Automated deployment script

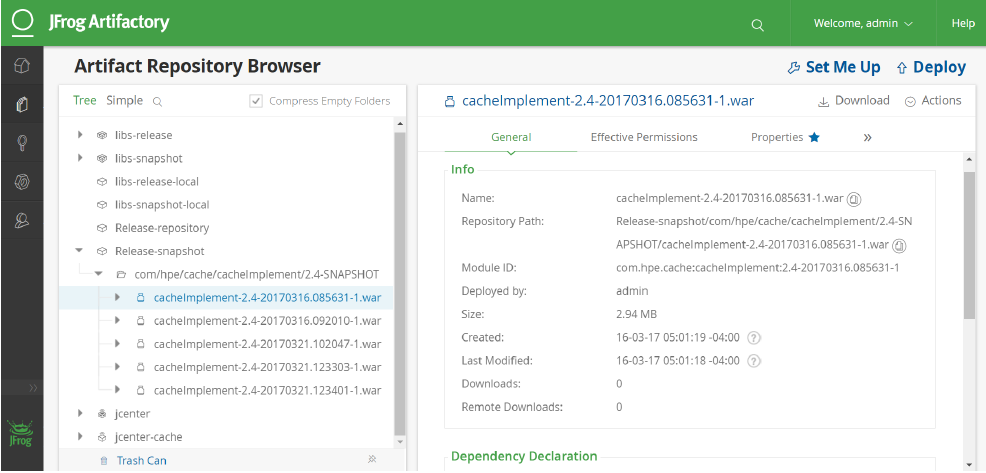


Figure 6.23: Jfrog artifactory

The Figure 6.23 show the path of the war package that needs to be deployed. It also shows the package inside the Release-snapshot directory. This is where all the packages that are ready for deployment are placed. They are deployed from this folder.

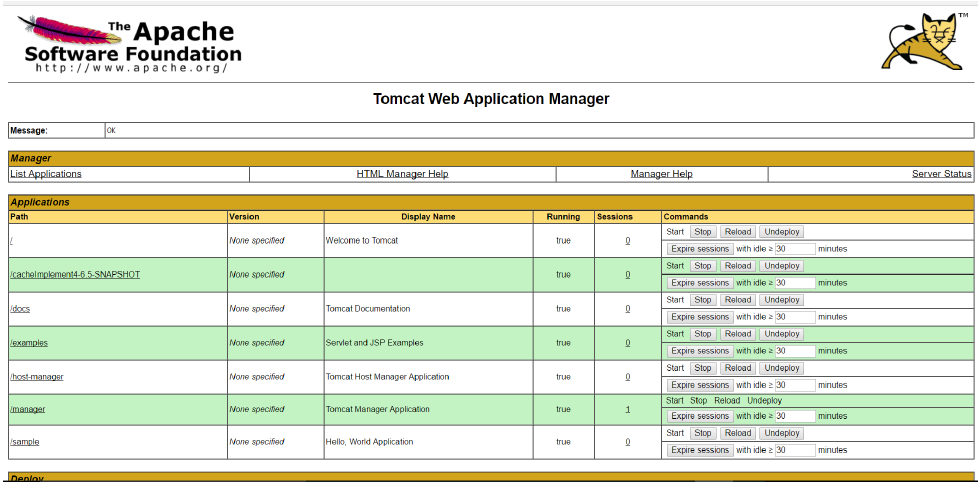


Figure 6.24: Tomcat view of deployed war

Figure 6.24 shows that the cache war package has been deployed on tomcat. The package has been automatically deployed from the artifactory jfrog from the Release-snapshot directory.

**6.6.2 Automated installation of java**

The Figures 6.25 and 6.26 show the script that is used to automatically install java on any node. This script can be used to install java on any new machine. It automatically extracts the file, moves it to the appropriate location, installs it automatically, sets the compiler path to javac and removes the jdk file after installation.

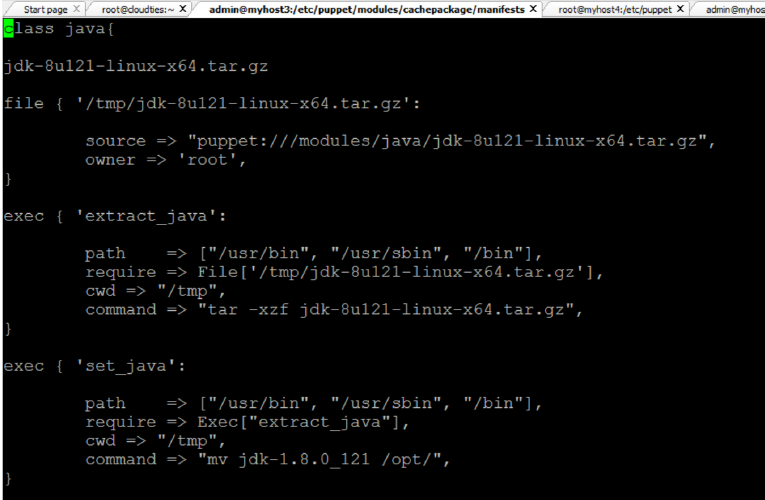


Figure 6.25: Automated java installation

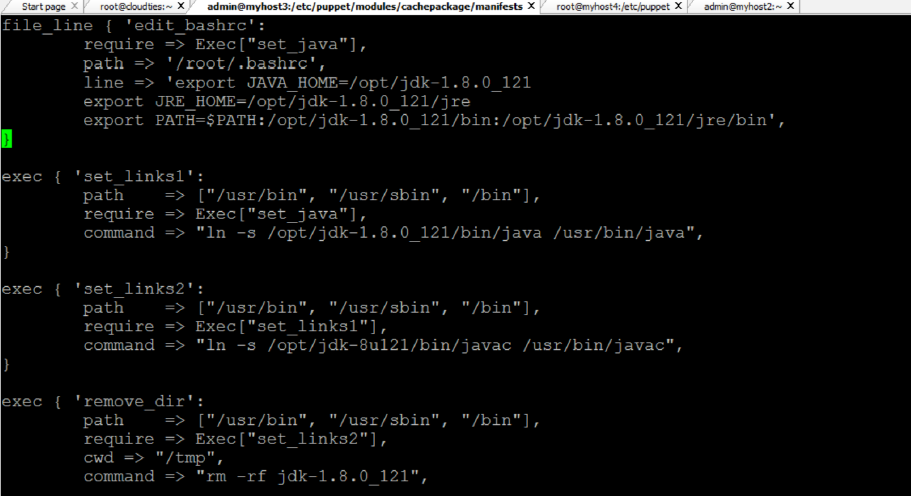


Figure 6.26: Automated java installation-2

**6.6.3 Automated jenkins installation**

Figure 6.27 shows the script that automatically installs jenkins on the nodes. All the nodes that are configured on the site.pp can automatically install jenkins on the nodes. Here rpm installation is carried out and the .rpm file is installed using the command:

* rpm -ivh /tmp/jenkins-1.656-1.1.noarch.rpm

After the service is installed then the service is started automatically.

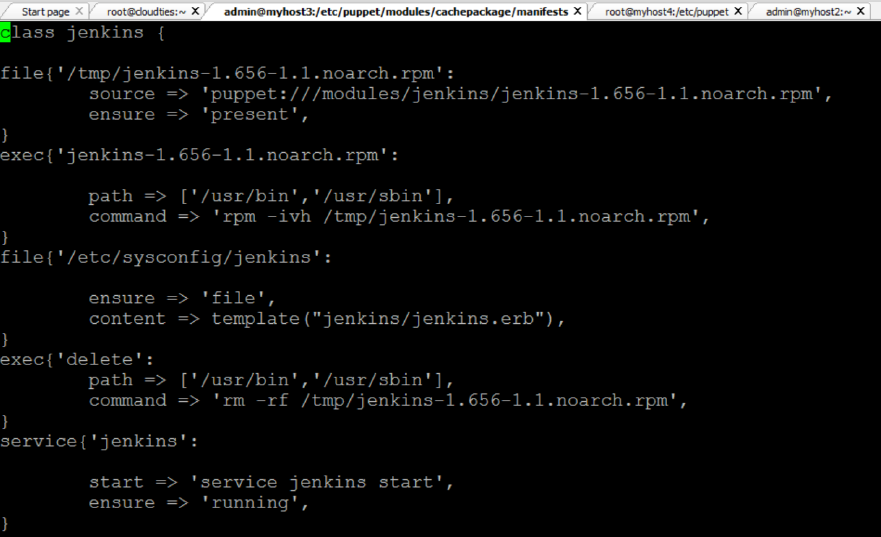


Figure 6.27: Automated jenkins installation

**6.6.4 Automated cloning of code from git.**

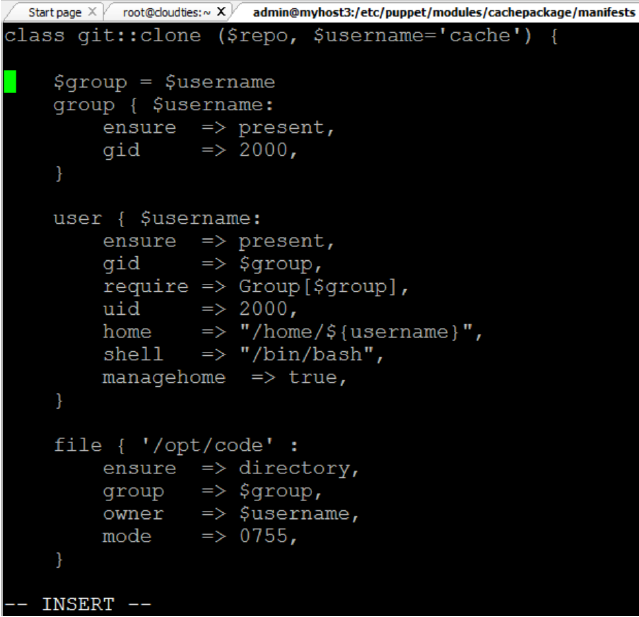


Figure 6.28: Git clone script

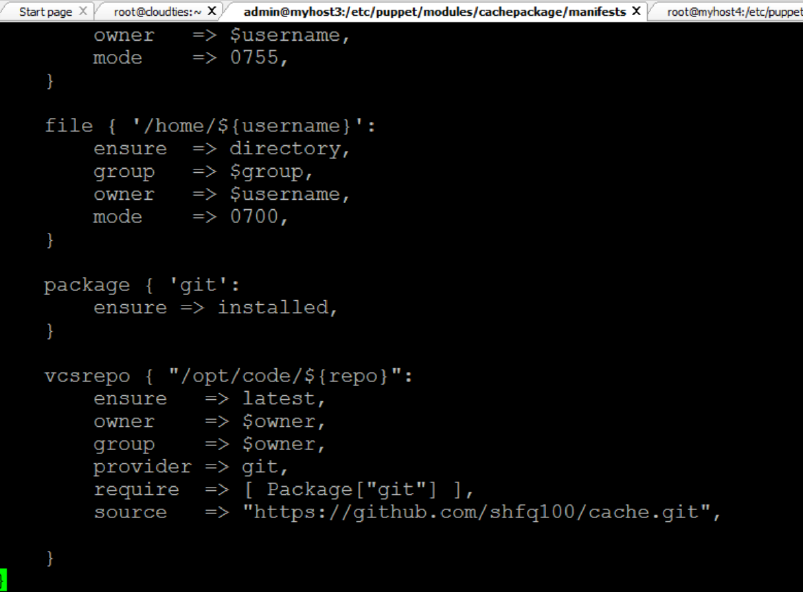


Figure 6.29: Git clone script

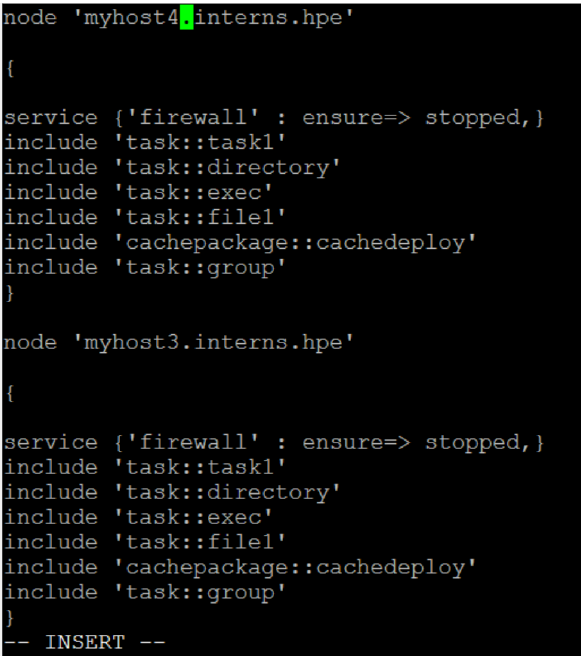


Figure 6.30: Site.pp file

The Figures 6.28 and 6.29 show the script that automatically clones the code from git and stores in the local repository. The vcs repo resource type will automatically deploy the content from the git onto the local machine.

Before the cloning is done this script will create a user called cache and will assign it to a new group. Then a new directory is created where the cloned content will be stored.

Figure 6.30 show the script of the site.pp file. It contains the nodes and the modules that are called on each node. The different modules under the include tag will perform the tasks defined in the scripts in the modules directory. This is the where all the automation tasks start and are performed.

**CHAPTER 7**

**SYSTEM TESTING**

**7.1 INTRODUCTION**

Testing accomplishes a variety of things, but most importantly it measures the quality of the software we are developing. This view presupposes there are defects in the software waiting to be discovered and this view is rarely disproved or even disputed.

Several factors contribute to the importance of making testing a high priority of any software development effort. These include:

* Reducing the cost of developing the program.
* Ensuring that the application behaves exactly as we explain to the user for the vast majority of programs, unpredictability is the least desirable consequences of using an application.
* Reducing the total cost of ownership. By providing software that looks and behaves as shown in the documentation, the customers require fewer hours of training and less support from product experts.
  1. **Different types of Testing:**

**7.2.1 Unit Testing:**

Unit testing focuses verification on the smallest unit of software design, the software component or module. Using the component level design description as a guide, important control paths are tested to uncover errors within the boundary of the module. The unit testing is a white box oriented testing.

First of all the module interface is tested to ensure that the information properly flows into and out of the program until under test. Then the local data structure is tested to ensure the data stored temporarily maintains its integrity during all steps in an execution. Boundary conditions are tested to ensure that the module operates properly at boundaries established to limit or restrict processing. All independent paths through the control structure are exercised to ensure that all statements in a module have been executed at least once. And finally, all errors handling paths are tested. In this project the testing is done according to bottom-up approach. Starting with smallest and lowest level modules and processing one at a time. For each module a driver and corresponding stubs were also written. If any errors found they were corrected immediately and the unit was tested again.

* + 1. **Integration Testing:**

Integration testing is a logical extension of unit testing. In its simplest form, two units that have already been tested are combined into a component and the interface between them is tested. A component, in this sense, refers to an integrated aggregate of more than one unit. The idea is to test combinations of pieces and eventually expand the process to test your modules with those of other groups. Eventually all the modules making up a process are tested together. Any errors discovered when combining units are likely related to the interface between units. This method reduces the number of possibilities to a far simpler level of analysis.

In this software, the bottom-up integration testing approached has been used, starting with the smallest and lowest level modules and proceeding one at a time. For each module the tests were conducted and the results were noted down.

* + 1. **User Testing:**

User Testing is nothing but the test of the software by the users themselves with live data being fed to the system. This helps in building really robust system. User testing in this system has been done extensively ascertain the results.

**7.3 System Testing**

This tends to affirm the end-to-end quality of the entire system. System test is often based on the functional/requirement specification of the system. Non-functional quality attributes, such as reliability, security, and maintainability, are also checked.

**7.4 Acceptance Testing**

This is done when the completed system is handed over from the developers to the Students or users. The purpose of acceptance testing is rather to give confidence that the system is working than to find errors.

7.4.1 Regression Testing:

Any time we modify an implementation within a program, we should also do regression testing. We can do so by retuning existing tests against the modified code to determine whether the changes break anything that worked prior to the change and by writing new tests where necessary. Some strategies and factors considered during this process include the following:

* Test fixed bugs promptly. Watch for side effects of fixes. The bug itself might be fixed but the fix might create other bugs. Regression test is written for each bug fixed. If two or more tests are similar, determine which is less effective and get rid of it. Identify tests that the program consistently passes and archive them. Focus on functional issues, not those related to design. Make changes (small and large) to data and find any resulting corruption. Trace the effects of the changes on program memory.

**7.4.2 Stress Testing:**

Stress testing, which is specialized form of performance testing, is similar to destructive testing in other field of engineering. The goal of stress testing is to crash the application by increasing the processing load past performance degradation until the application begins to fail due to saturation of resources or the occurrence of errors. Stress testing helps to reveal subtle bugs that would otherwise go undetected until the application was deployed. Since such bugs are typically the result of design flaws, stress testing should begin early in the development phase on each area of the application. Fix these subtle bugs at their source instead of fixing symptomatic bugs tat may occur elsewhere in the application if these bugs were ignored.

**7.5 SCREENSHOT:**

**CHAPTER 8**

**CONCLUSION AND FUTURE WORK**

**8.1 CONCLUSION**

The tasks for the successful automated deployment have been completed and the code has been automatically deployed on the server with the help of puppet automation. With the help of puppet the installation of services like jenkins, jfrog, nagios are automated and the installation can be carried out on any server. Nagios successfully monitors the hosts and returns the alerts for all the services running in the network. By following the DevOps module, the site reliability engineering project has been successfully implemented.

**8.2 FUTURE WORK**

The steps involved in the installation and configuration of build tool maven can be automated using puppet script. Till now its the only tool in the proposed system whose installation is yet to be automated. It involves the integration of multiple dependencies and tools to build the code. The definition of paths in maven and installation of plugins in jenkins is a vital part of the future work related to automation. The automated provisioning of the virtual machines by hypervisor can be done using the tool razor. This will lead to the achievement of high availability of the system. The specified tasks upon completion will result in an advanced automated environment.

**REFERENCES**

[1] Hpe company documentations.

[2] https://learn.perl.org/tutorials.

[3] https://en.wikipedia.org/wiki/nagios.

[4] www.puppetresources.com.

[5] https://newrelic.com/devops/benefits-of-devops.

[6] http://www.devopsdigest.com/devops-advantages-1.

[7] www.slashroot.in/puppet-tutorial-introduction-puppet-configuration-management-tool.

[8] https://www.nagios.com/solutions/network-monitoring-tools/.

[9] https://docs.puppet.com/puppet/4.9/architecture.html.