**CHAPTER 6**

**SYSTEM IMPLEMENTATION**

The Analysis of this Devops office model 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 6.1 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.

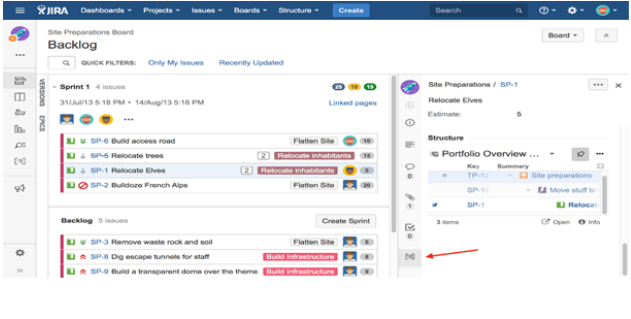


Figure 6.1: Example for Scrum Board

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).

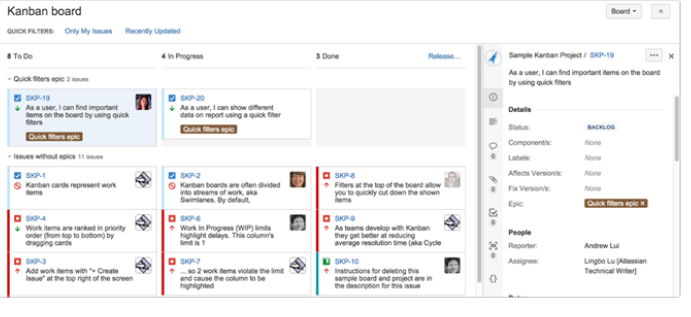


Figure 6.2: Example for Kanban Board

**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.

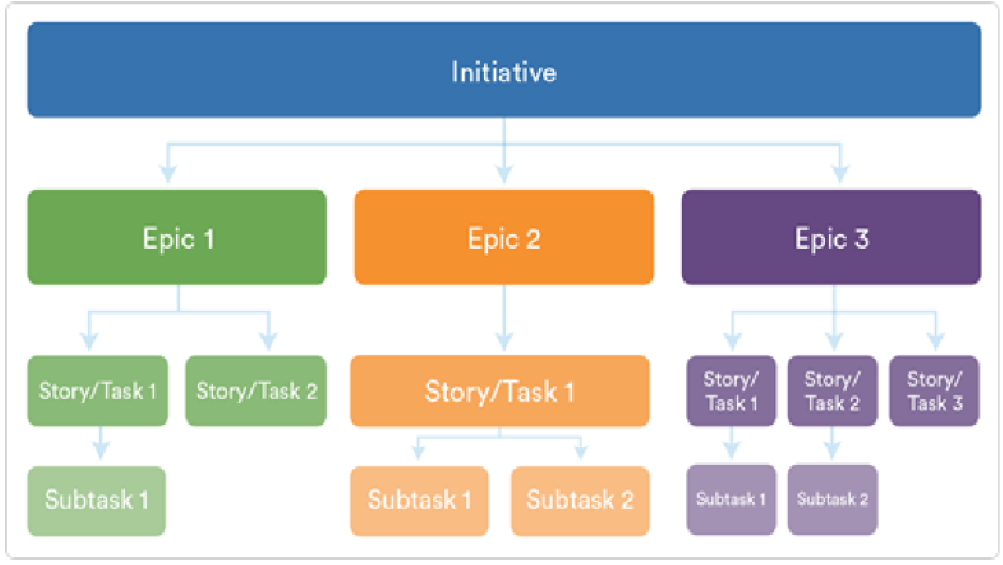


Figure 6.3: Structure of Jira

**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.

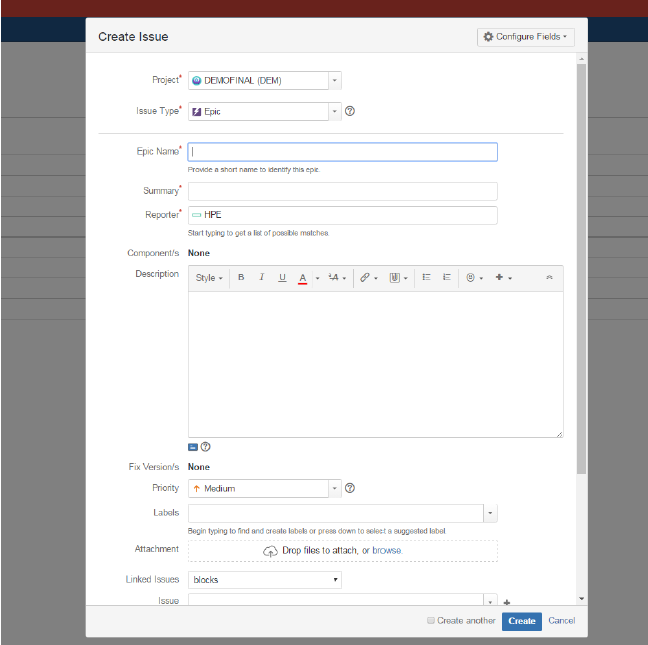


Figure 6.5: Create Stories

* 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.

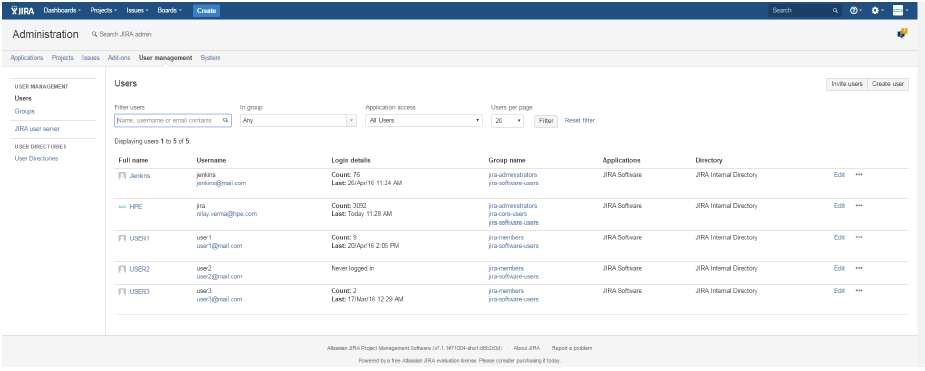


Figure 6.6: User Creation

* 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”.

**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.



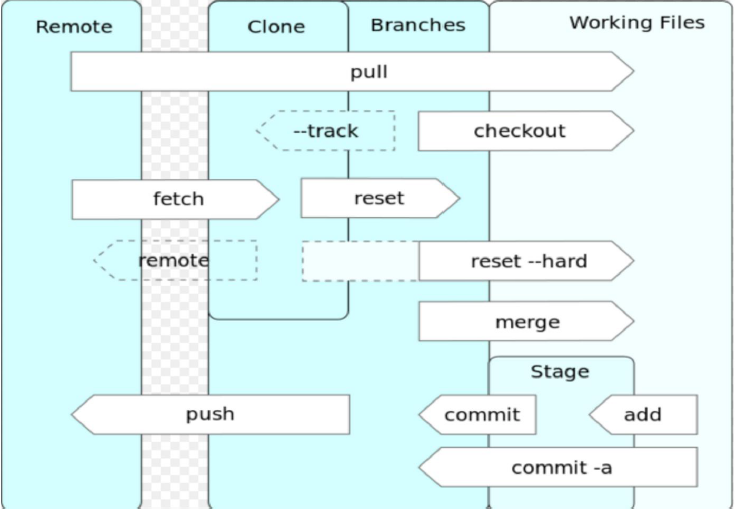


Figure 6.7: Git Operations

* 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.

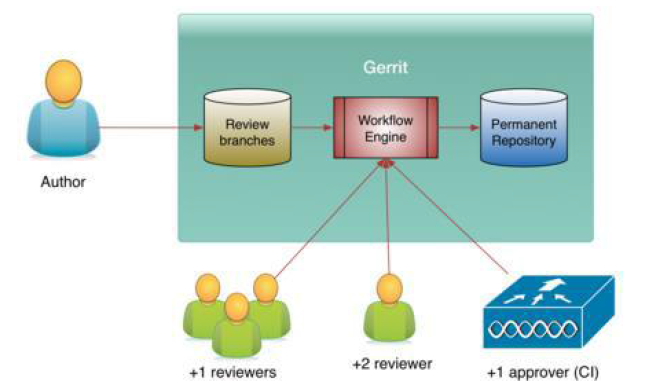


Figure 6.8: Gitlabs Workflow

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.

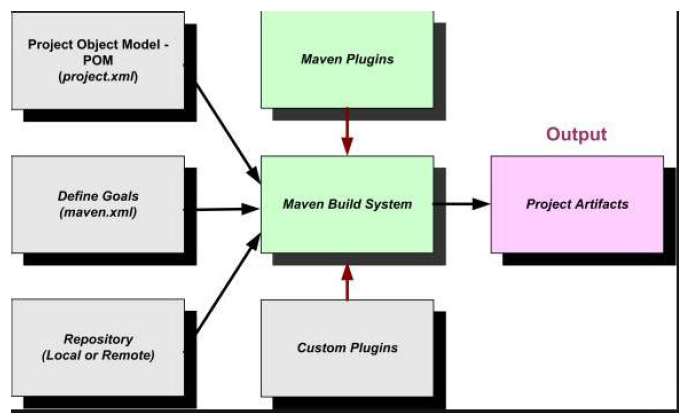


Figure 6.9: Maven Structure

**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.

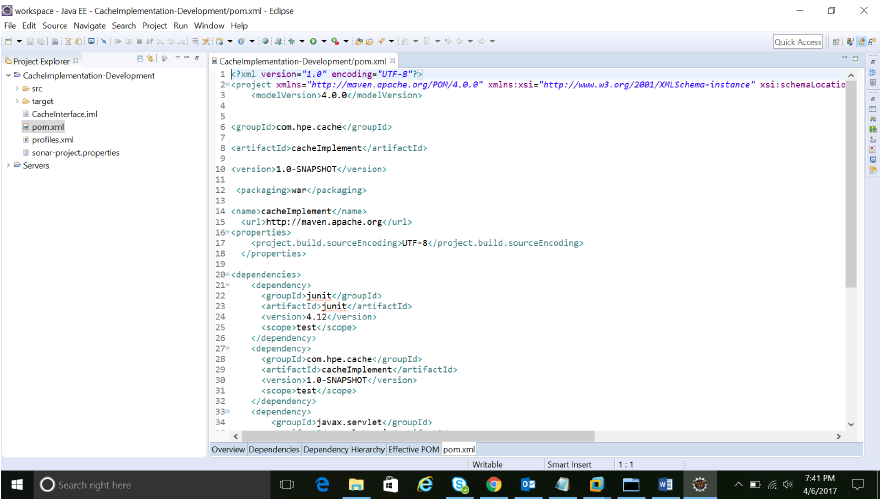


Figure 6.10: Maven pom.xml

(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 6.10, 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.
* Model Version: 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.

**6.6 INTEGRATION SERVER**

The integration server section contains the details of continuous integration, Jenkins, Maven plugins, Git integration, Maven integration, Maven settings and jfrog integration with Jenkins.

**6.6.1 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.

**6.7 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, Concurrent Version Systems (CVS), subversion, Git, Mercurial, Perforce, Clearcase and Rational Team Concert (RTC) and can execute Apache Ant and Apache Maven based tasks.

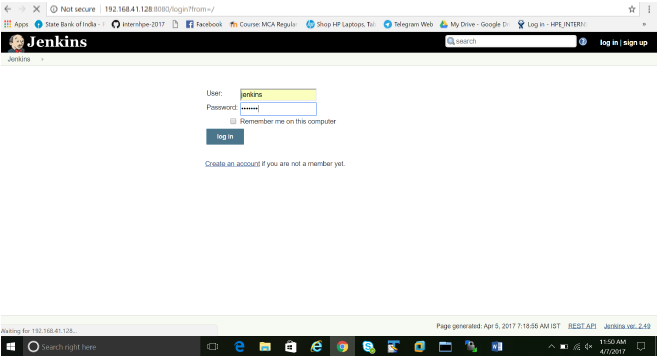


Figure 6.11: Jenkins login page

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 7ialects 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.

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 6.11.

**6.7.1 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 6.12. 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 6.13 shows the installation of the plugins for Git.

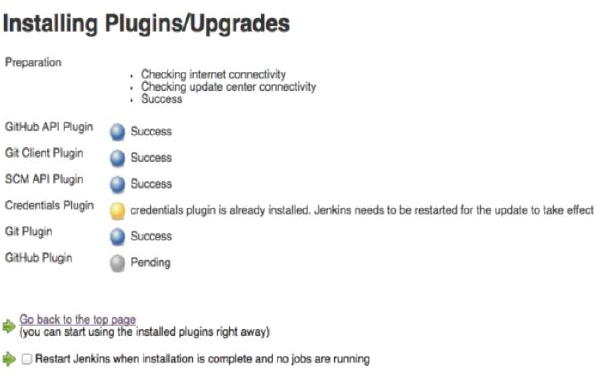


Figure 6.12: Installing Plugins for Tools

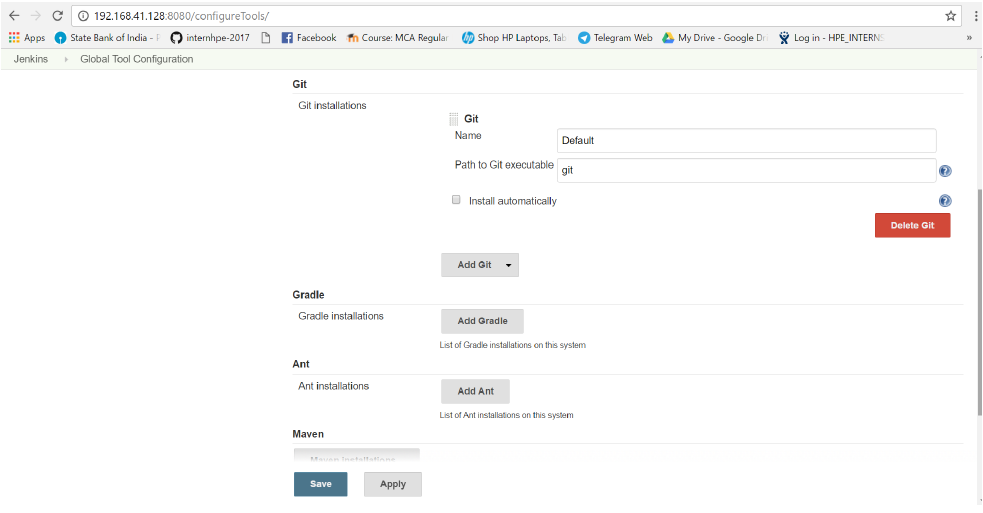


Figure 6.13: Integration with tools

**6.7.2 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 plugins which are available for downloading. In the ‘Filter’ tab type ‘Git plugin’.

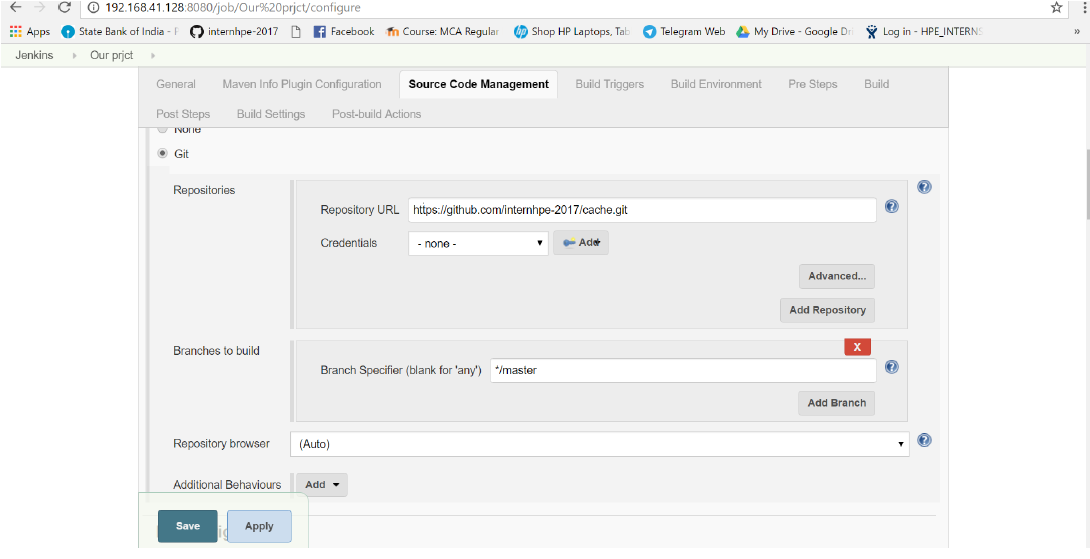


Figure 6.14: Fetching Project from Git

* 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 6.14 shows, In the next screen, browse to the Source code Management section, can see ‘Git’ as an option.

**6.7.3 Maven Integration**

Jenkins provides a job type dedicated to Maven . This job type integrates Jenkins deeply with Maven and provides the following benefits compared to the more generic free-style software project.

* 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.

**6.7.4 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 MAVEN HOME.
* Then, click on the ‘Save’ button at the end of the screen. As shown in Figure 6.15 the maven build commands has to be given for the Jenkins to work on it’s own based on the given command.

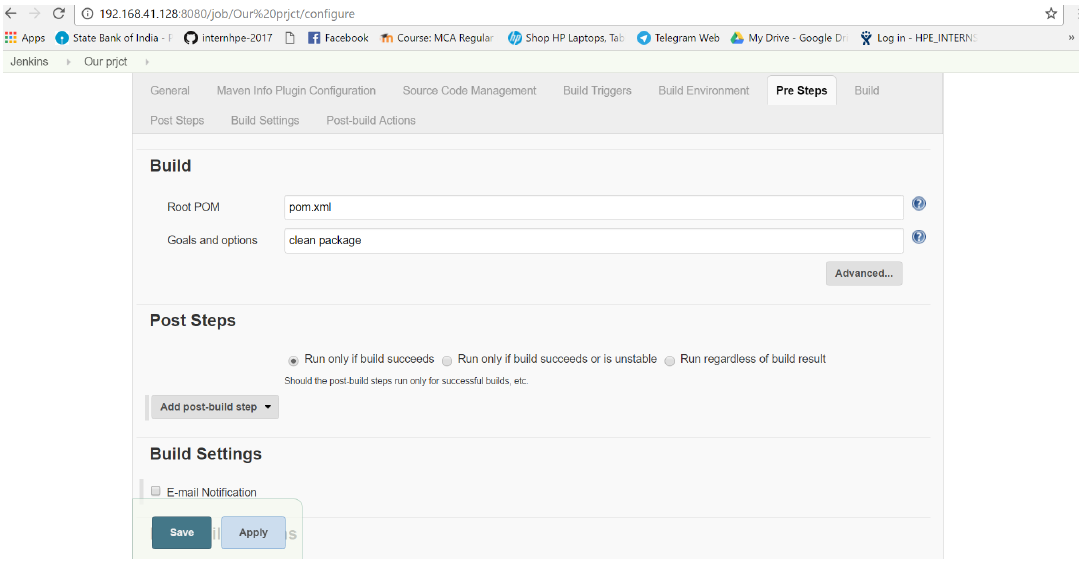


Figure 6.15: Build Package

**6.8 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.

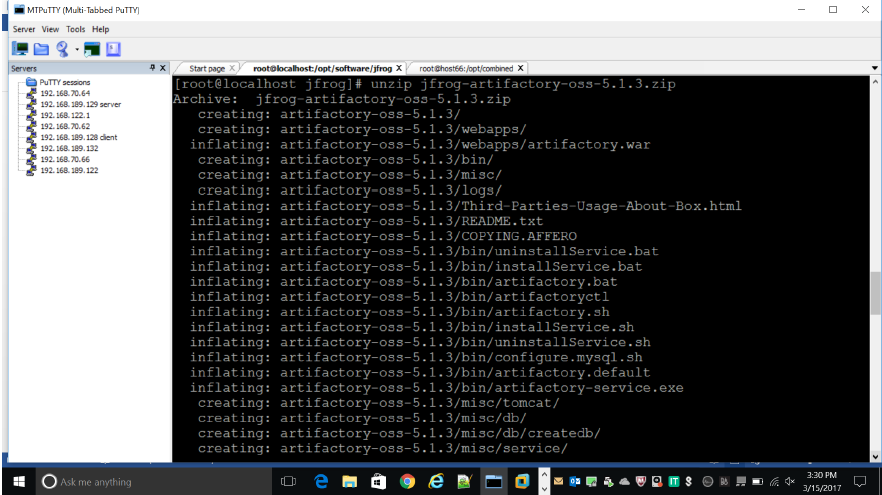
To achieve continuous deployment, the team relies on infrastructure that automates and instruments the various steps leading up to deployment, so that after each integration successfully meeting these release criteria, the live application is updated with new code.

**6.8.1 JFROG ARTIFACTORY**

* As the first, and only, universal Artifact Repository Manager on the market, JFrog Artifactory fully supports jfrog packages created by any language or technology.
* Artifactory is the only enterprise-ready repository manager availabletoday, supporting secure, clustered, High Availability Docker registries.
* Integrating with all major CI/CD and DevOps tools, Artifactory provides an end-to-end, automated and bullet-proof solution for tracking artifacts from development to production.

Prerequisites for JFrog: We need some of the package before we install the JFrog. We have some of the prerequisites for JFrog.

* JAVA (V1.8 and greater)
* PHP (V7.0.5)
* MySQL Remote server / Local Server (5.6.35)

Figure 6.16: Installing jfrog artifactory

* In the figure 6.16 we are unzipping the jfrog artifactory.

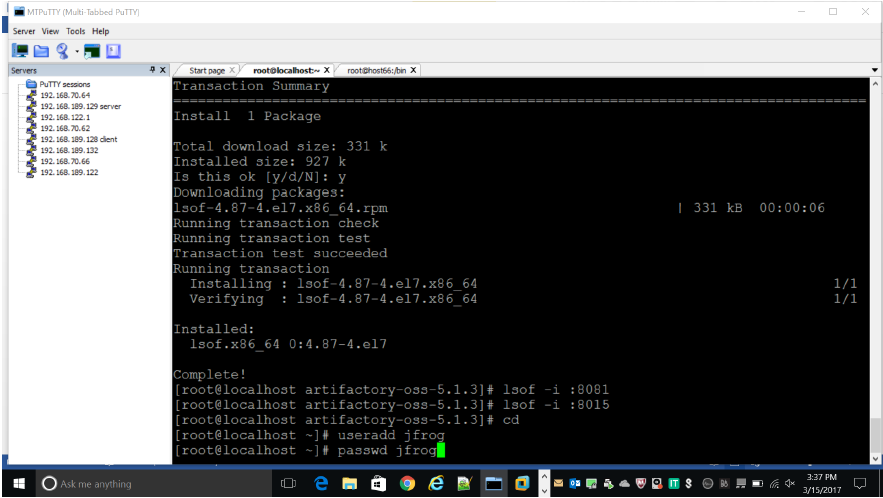


Figure 6.17: Adding jfrog user

* In the figure 6.17jfrog artifactory is installed and jfrog user has been created. Make the jfrog user as a sudo user.

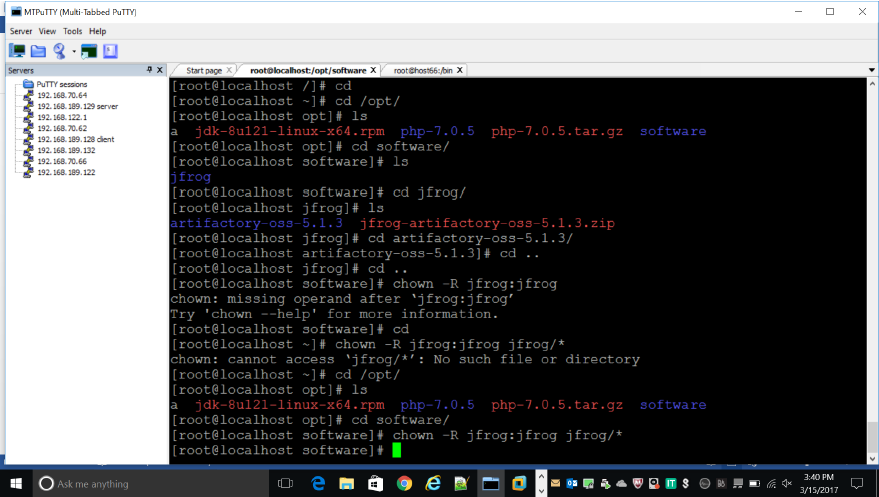


Figure 6.18: Changing ownership to jfrog

* In the figure 6.18 changing the ownership of jfrog directory to jfrog user.

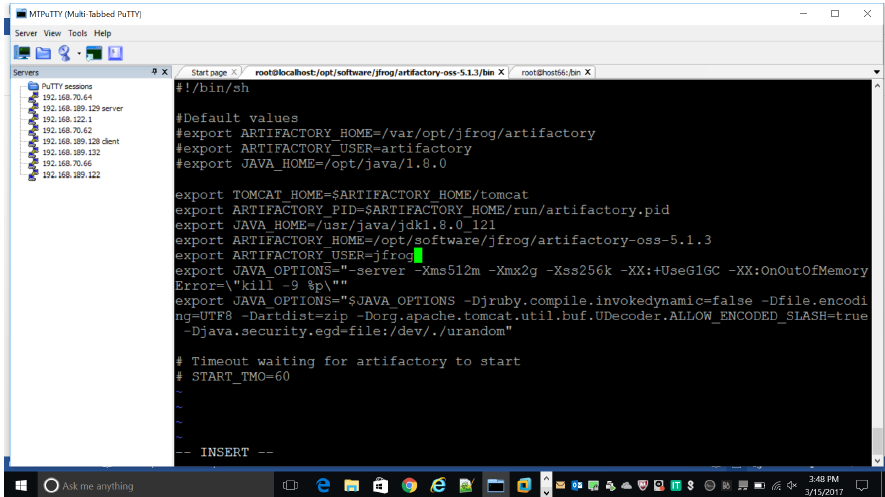


Figure 6.19: Adding variables to the configuration files

In the figure 6.19 adding java, tomcat and jfrog home directories to the jfrog configuration files.

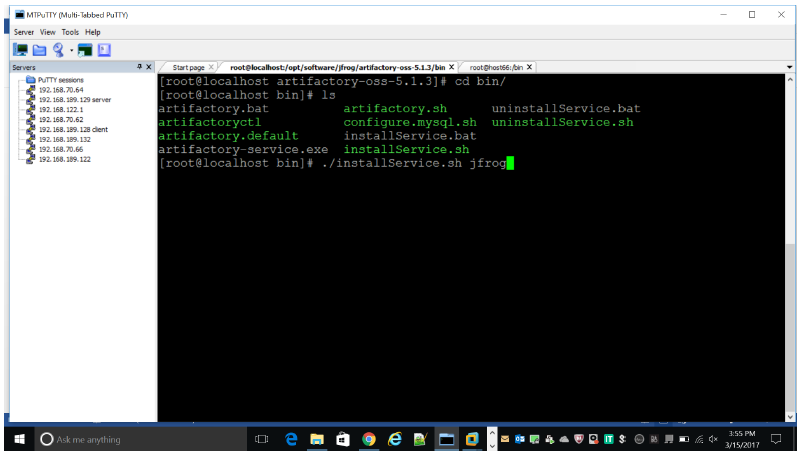


Figure 6.20: Installing jfrog service

* In the figure 6.20 installing the service of jfrog by running the ./installService.sh file.

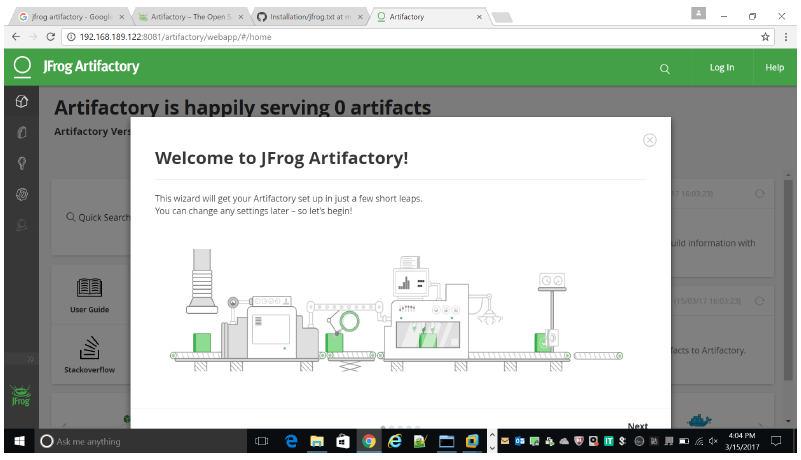


Figure 6.21: Jfrog artifactory interface

* In the figure 6.21 shows the jfrog interface once installed in the machine.

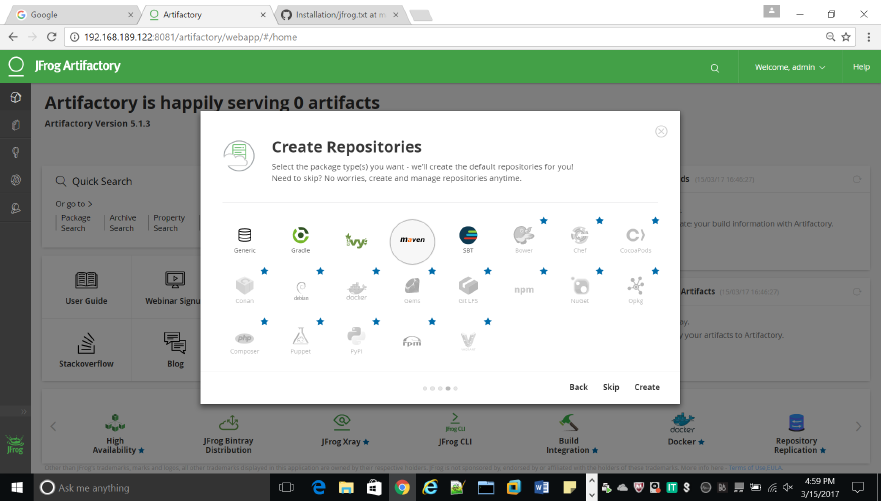


Figure 6.22: creating repository for maven project

* Figure 6.22 creates a maven repository where we can store all artifacts like dependencies, plugins and packages.

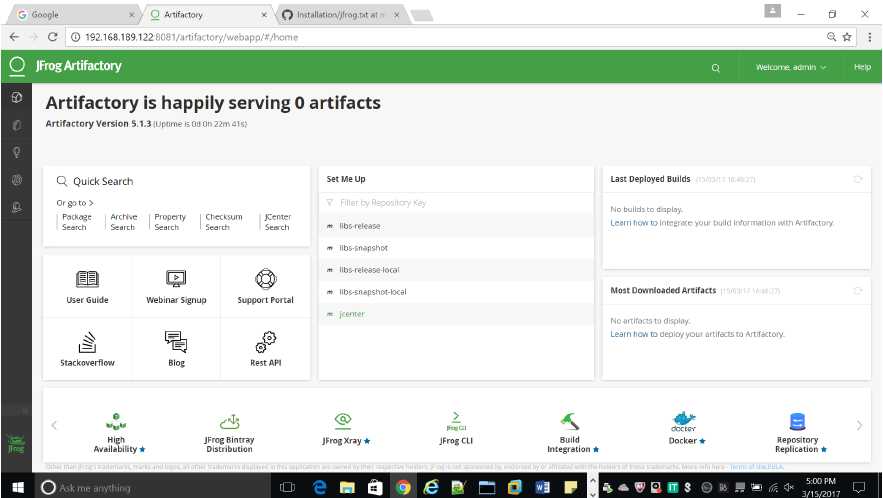


Figure 6.23: Artifactory dashboard

* Figure 6.23 shows the artifactory dashboard once the maven repository has been created.

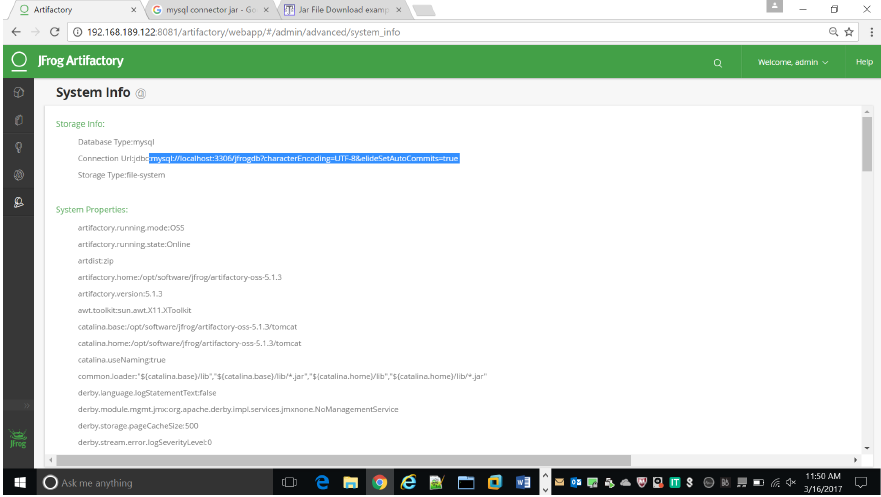


Figure 6.24: Changing database to mysql

* Figure 6.24 changing the default database Derby to Mysql.

**6.8.2 Integrating Jfrog with Jenkins**

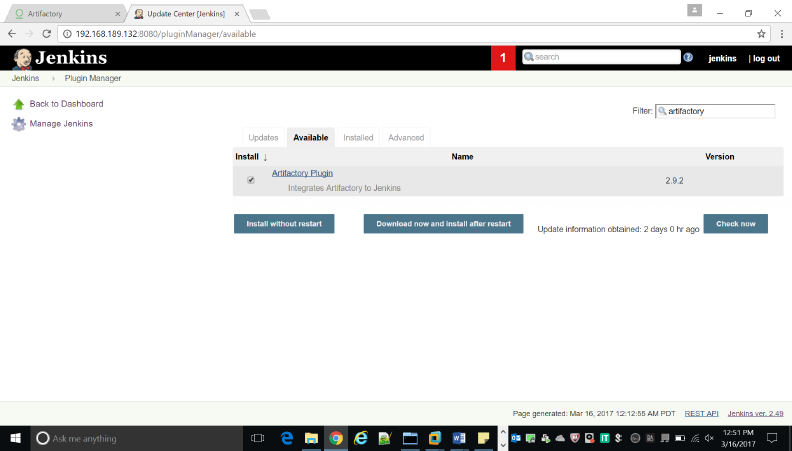
****

Figure 6.25: Installing artifactory plugin

* Figure 6.25 installing the artifactory plugin to integrate the jfrog with jenkins.

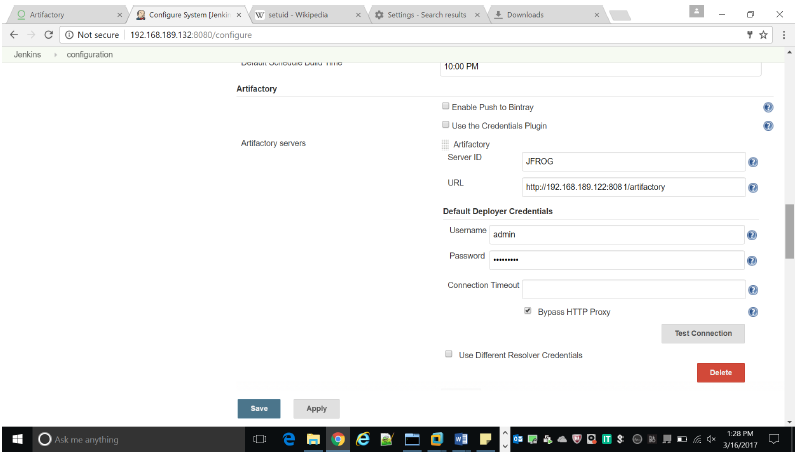


Figure 6.26: Adding artifactory server to the jenkins

* Figure 6.26 adding the jfrog server 192.168.189.122 and adding the user credentials in the jenkins server.

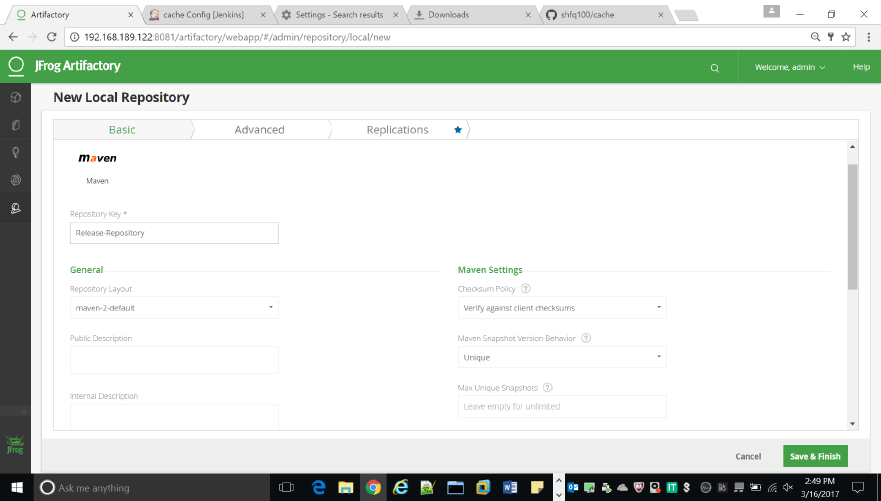


Figure 6.27: Creating local repository

* Figure 6.27 creating the local repository Release repository to resolve the artifacts.

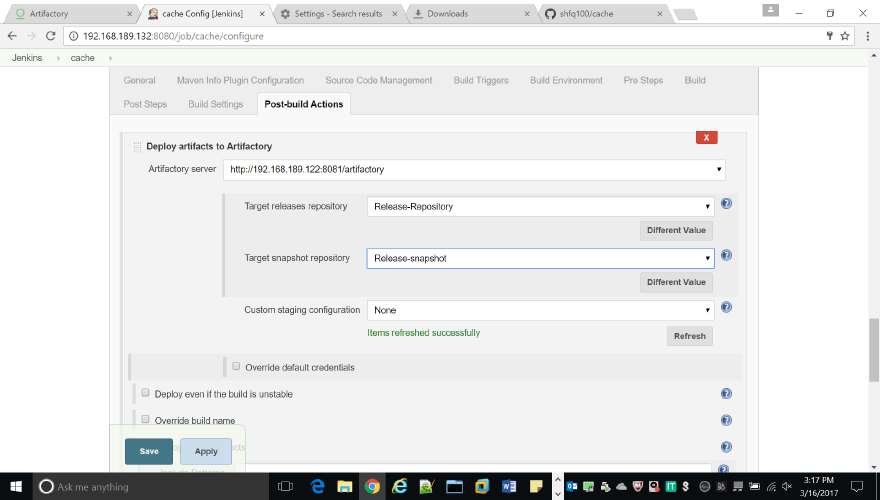


Figure 6.28: Adding target repository to jenkins

* Figure 6.28 chooses target releases and snapshots repository where we deploy all our artifacts like build package.

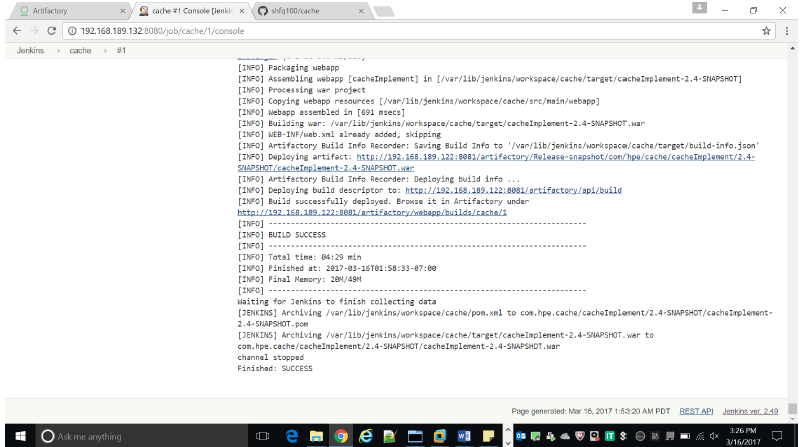


Figure 6.29: Building project to artifactory

* Figure 6.29 building the package and deployed it into the above selected jfrog artifactory.

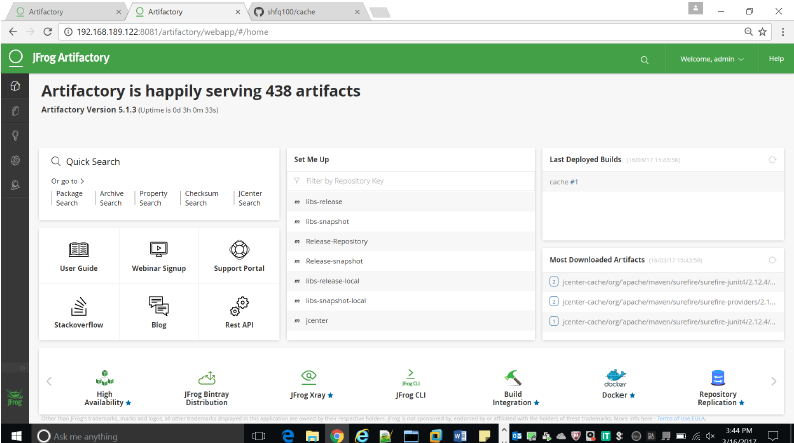


Figure 6.30: Artifacts deployed in the artifactory

* In the figure 6.30 artifacts are deployed and now jfrog is serving many artifacts.

**6.9 Monitoring tools**

Monitoring tools are used to continuously keep track of the status of the system in use, in order to have the earliest warning of failures, defects or problems and to improve them. There are monitoring tools for servers, networks, databases, security, performance, website and internet usage, and applications.

Features or characteristics of monitoring tools are:

* To identify the problems and send an alert message to the administrator (e.g. network administrator);
* To log real-time and historical information;
* To find optimal settings;
* To monitor the number of users on a network;
* To monitor network traffic (either in real time or covering a given length of time of operation with the analysis performed afterwards)

**6.9.1 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.

**6.9.1.1 Installation of httpd**

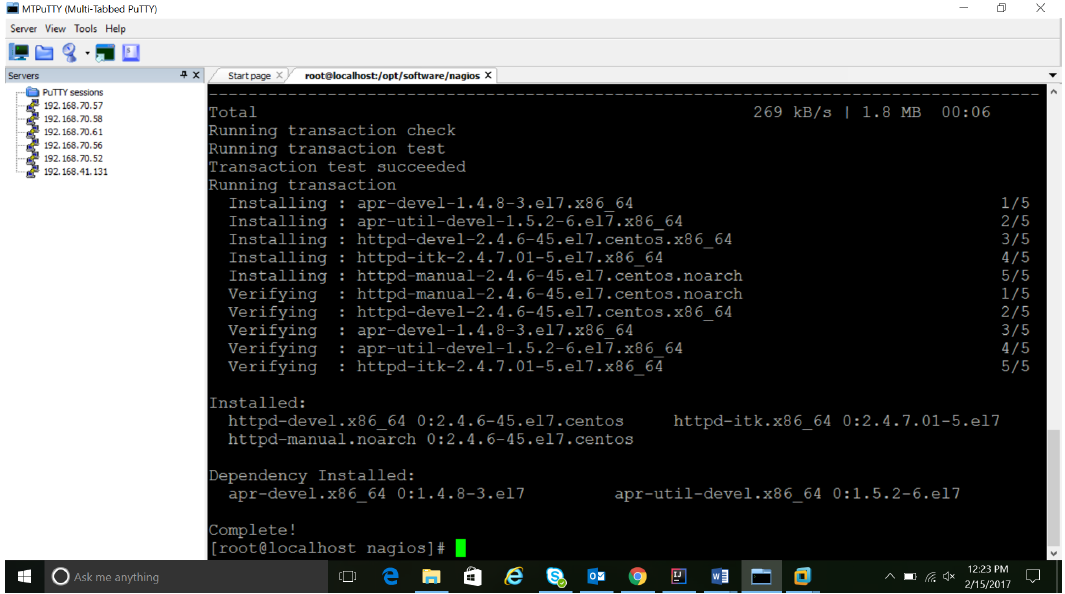


Figure 6.31: Installation of httpd

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.

**6.9.1.2 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

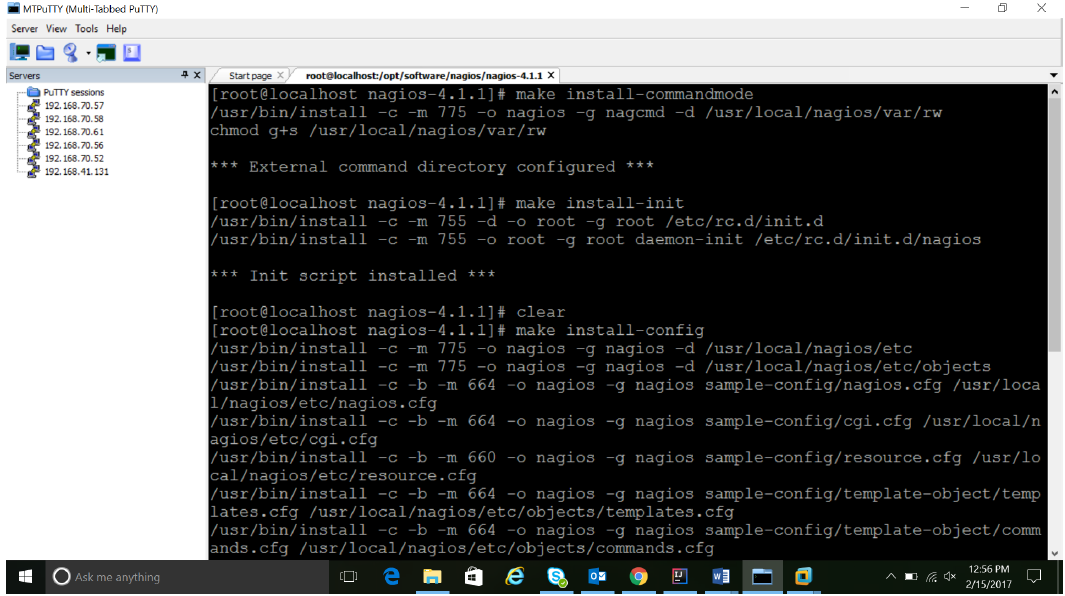


Figure 6.32: Installation of nagios

Figures 6.31 and 6.32 shows 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

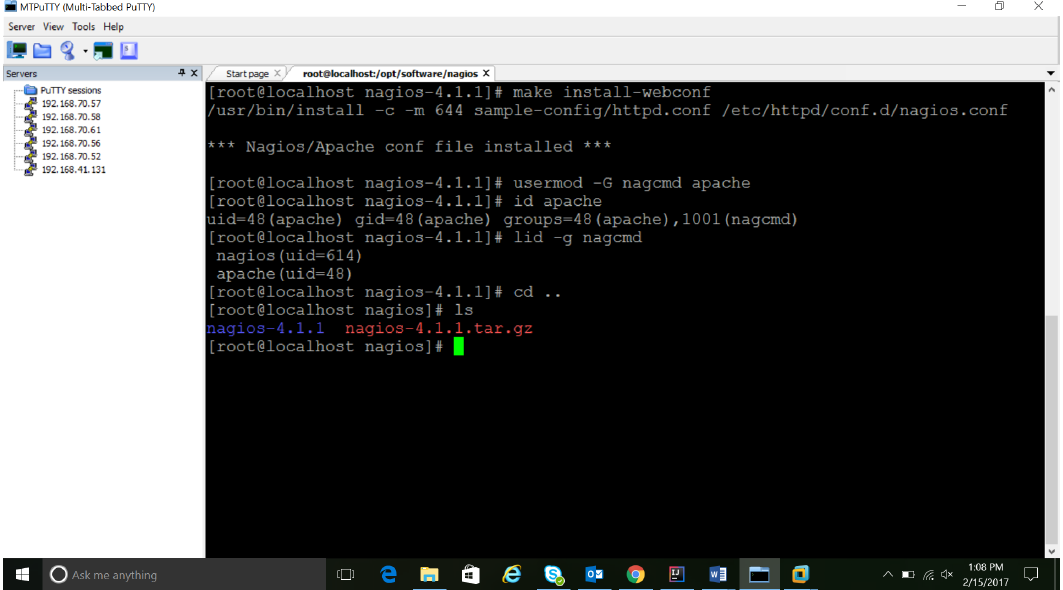


Figure 6.33: Installation of nagios-2

**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

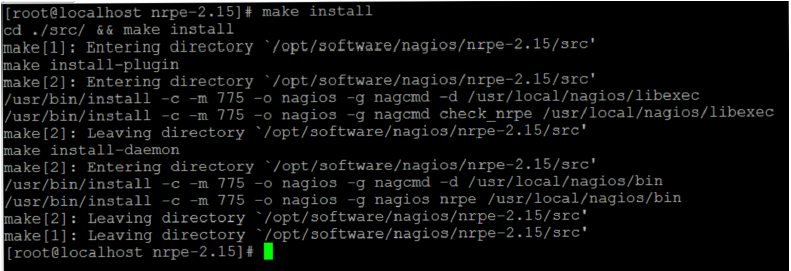


Figure 6.34: Installation of XINETD

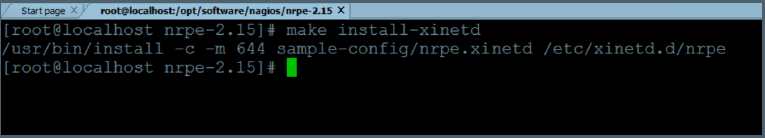


Figure 6.35: Installation of XINETD-2

Figures 6.33, 6.34 and 6.35 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 6.36: Installation of XINETD-3

**6.9.2 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 6.37: List of plugins

Figure 6.37 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.

**6.9.2.1 Plugin to check used space**

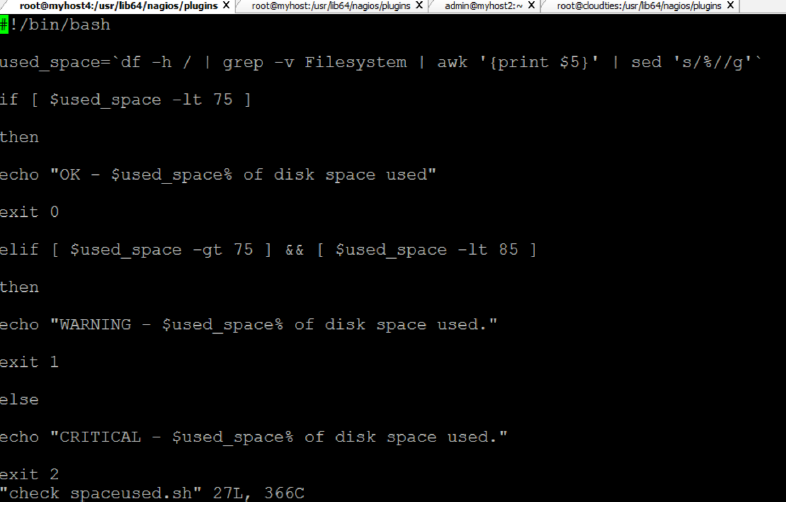
****

Figure 6.38: Plugin to check space used

Figure 6.38 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.

**6.9.2.2 Plugin to check total processes**

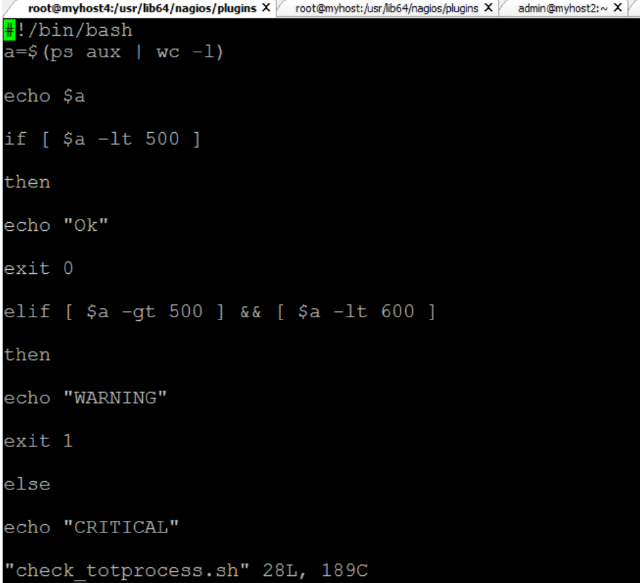


Figure 6.39: Plugin to check total running processes

Figure 6.39 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.

**6.9.2.3 Plugin to check total users**

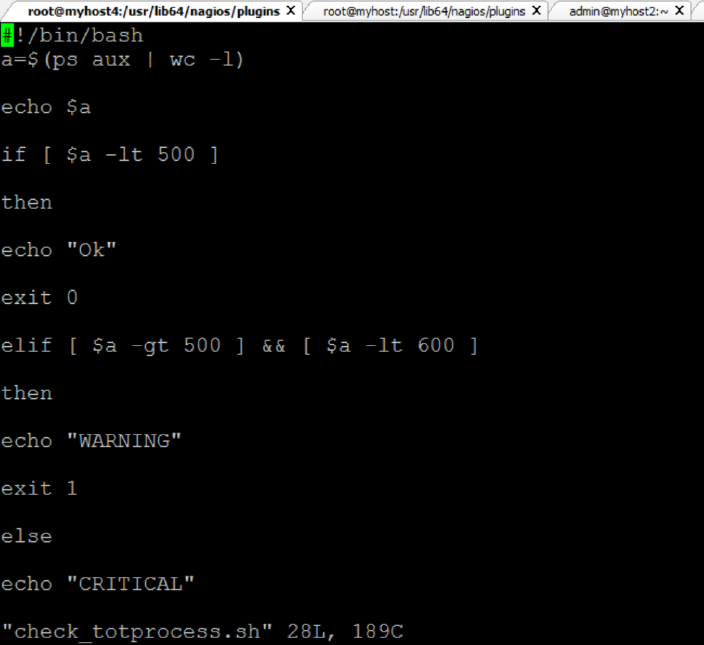


Figure 6.40: Plugin to check total active users

Figure 6.40 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.

**6.9.2.4 Plugin to check artifactory status**



Figure 6.41: Plugin to check artifactory status

Figure 6.41 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.

**6.9.2.5 Plugin to check puppet master status**



Figure 6.42: Plugin to check puppet master status

Figure 6.43 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.

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

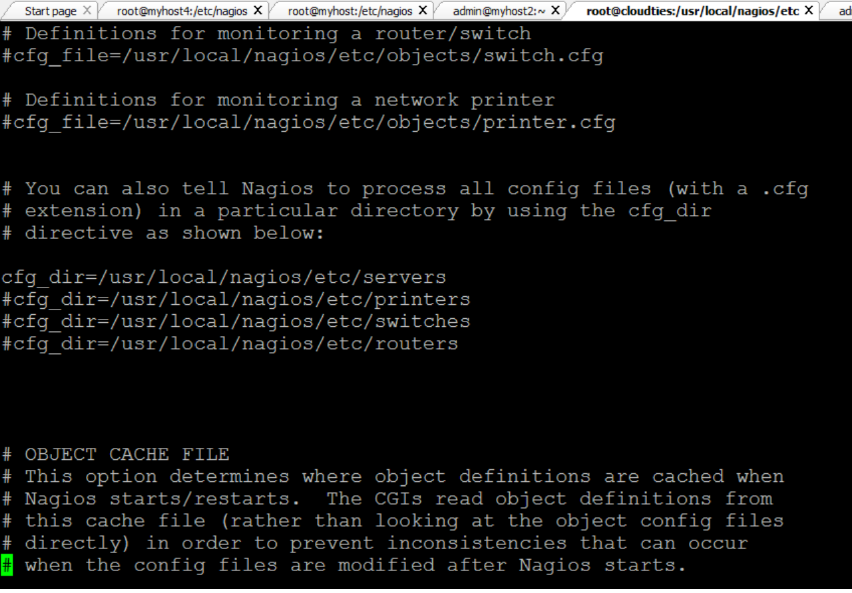


Figure 6.43: Definition of server host type.

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.

**6.9.3.1 Definition of command in slave**

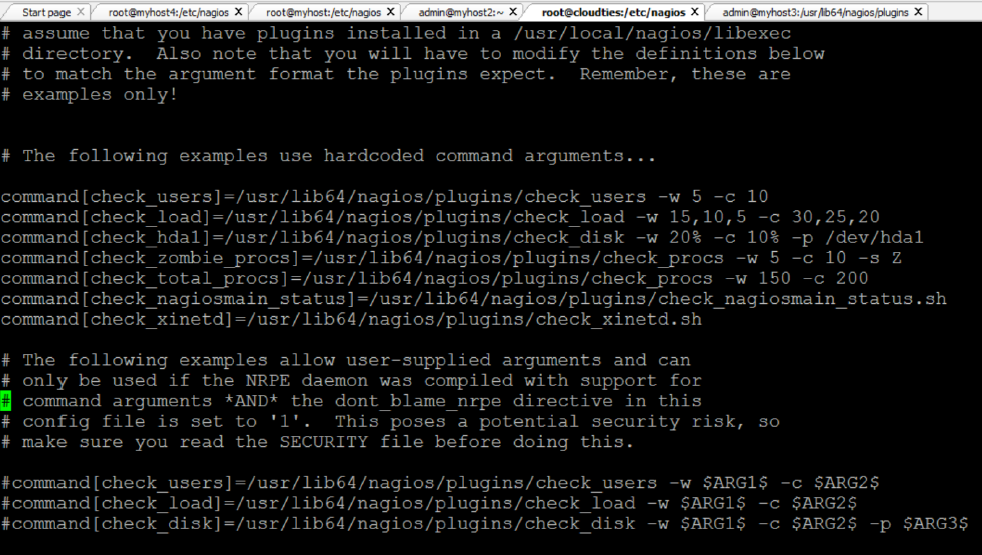


Figure 6.44: Defining command in slave

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 6.44 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.

**6.9.3.2 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.

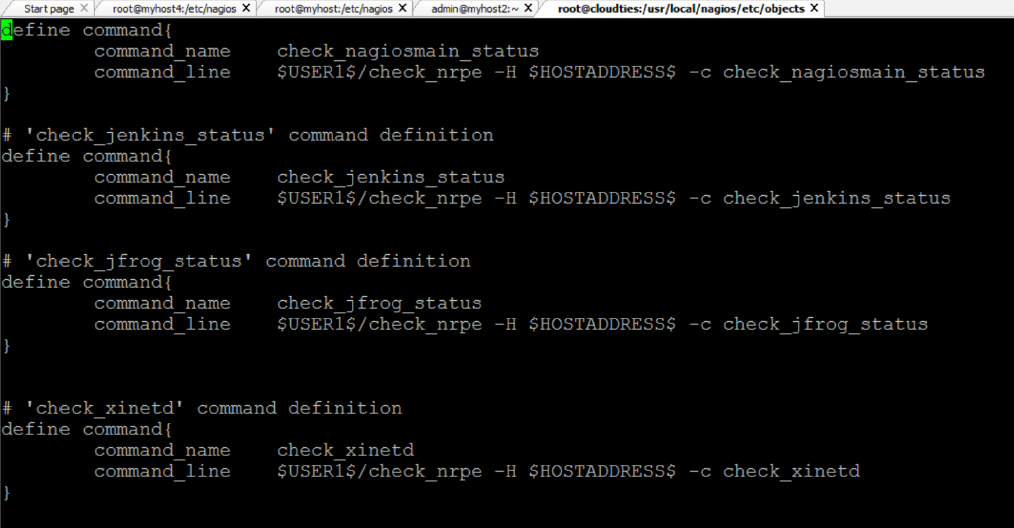


Figure 6.45: Defining command in master-1

The commands defined in Figure 6.45 and Figure 6.46 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.

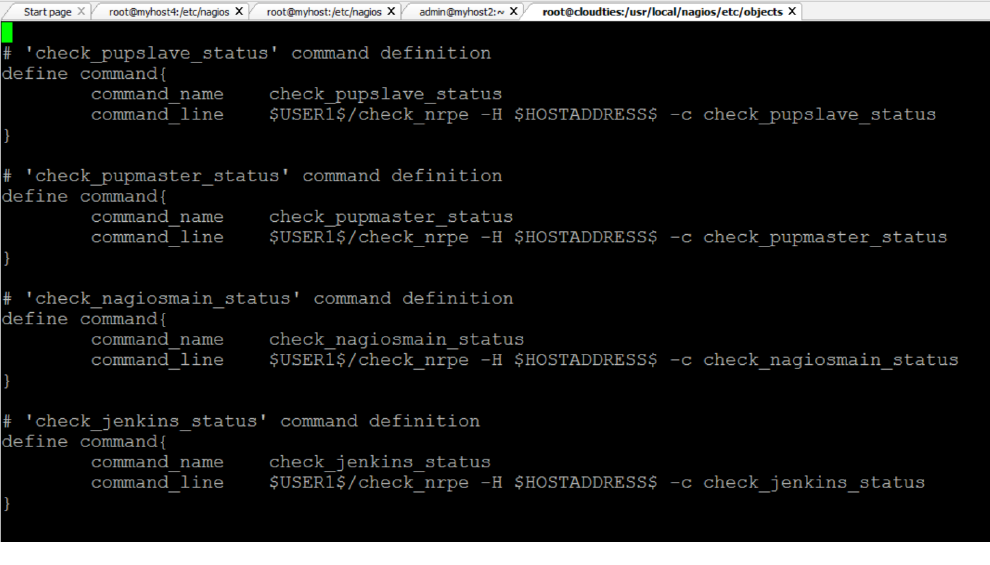


Figure 6.46: Defining command in master-2

**6.9.4 TERMINAL OUTPUT OF HOSTS AND SERVICES**

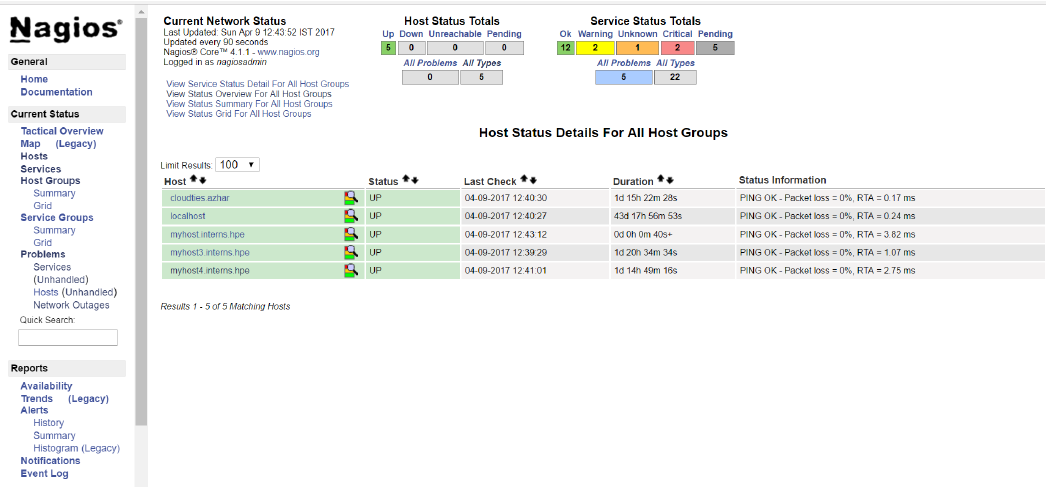


Figure 6.47: Console output of hosts

Figure 6.47 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.

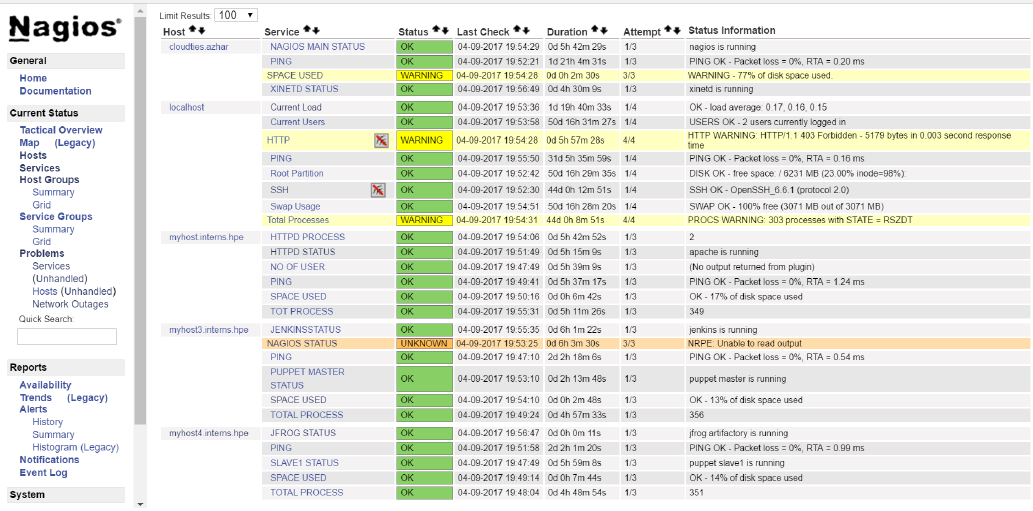


Figure 6.48: Console output of services

Figure 6.48 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 6.47 and 6.48 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.

**6.10 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.

**6.10.1 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 , 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/pup8pet.conf directory.

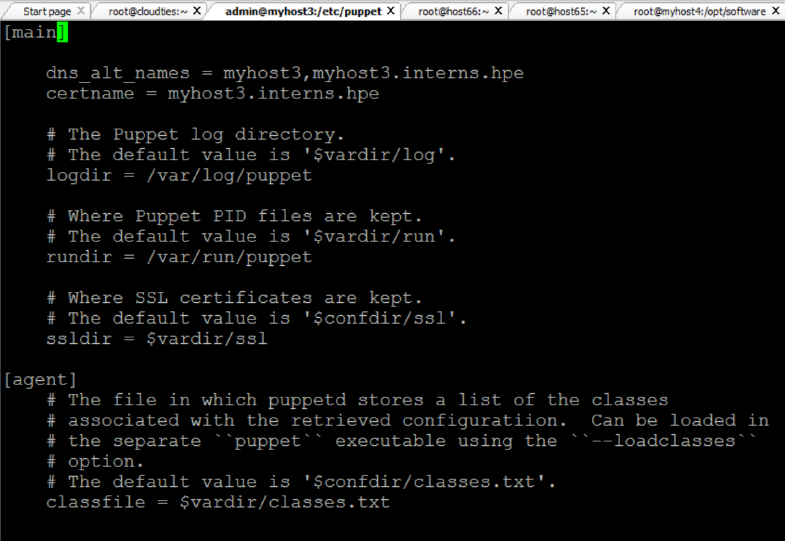


Figure 6.49: Defining puppet master

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.

**6.10.2 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.

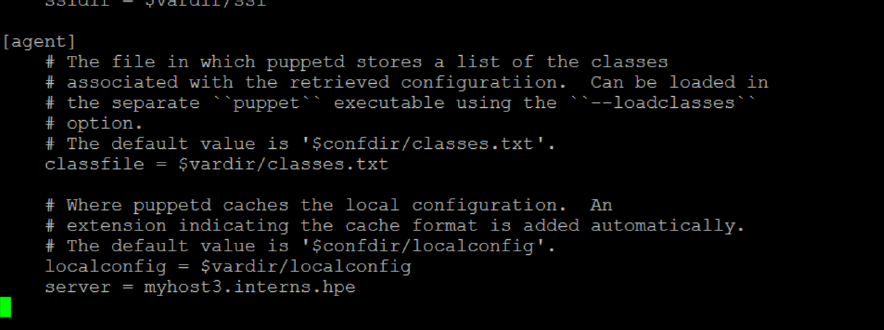


Figure 6.50: Defining puppet slave

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

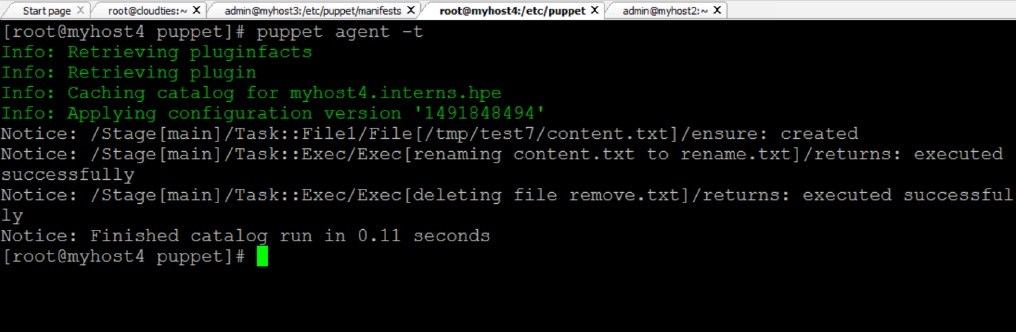


Figure 6.51: Execution command of scripts.

Figure 6.52 depicts the result of a successful test run of the puppet script.

**6.10.3 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.

**6.10.3.1 Automated deployment script**

Figure 6.52 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.

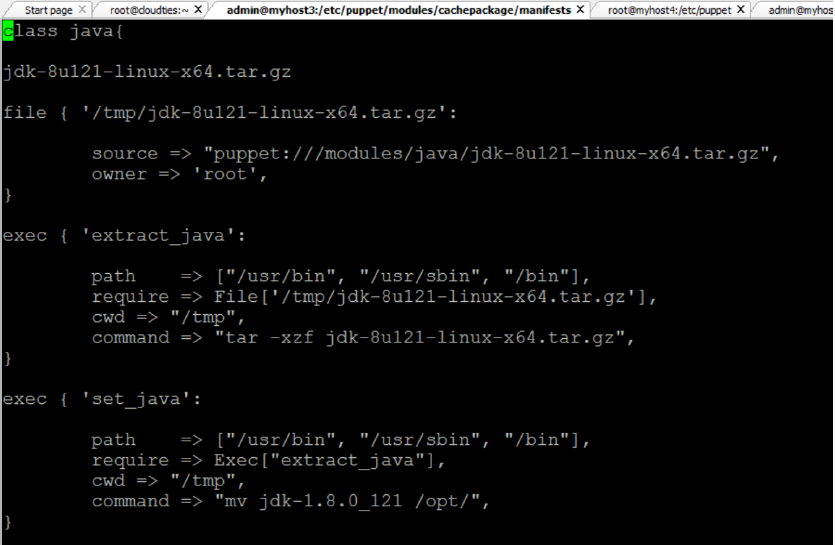


Figure 6.52: Automated deployment script

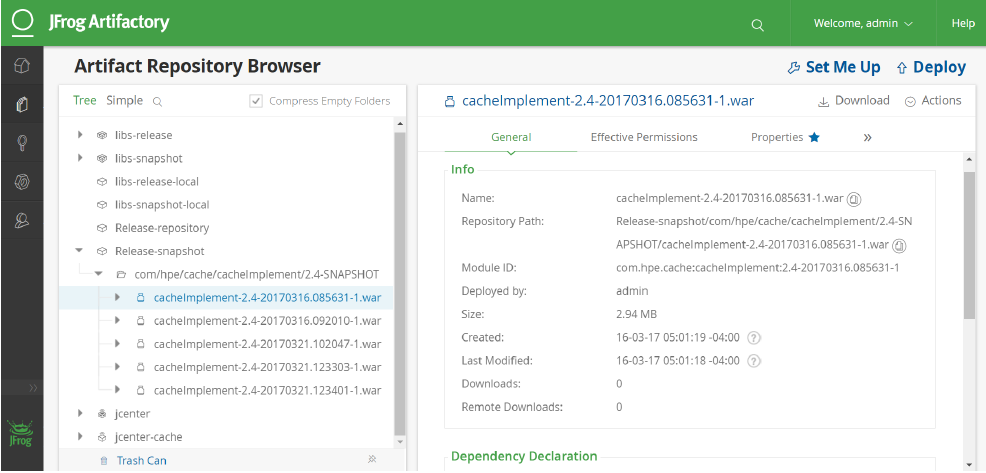


Figure 6.53: Jfrog artifactory

The Figure 6.53 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.

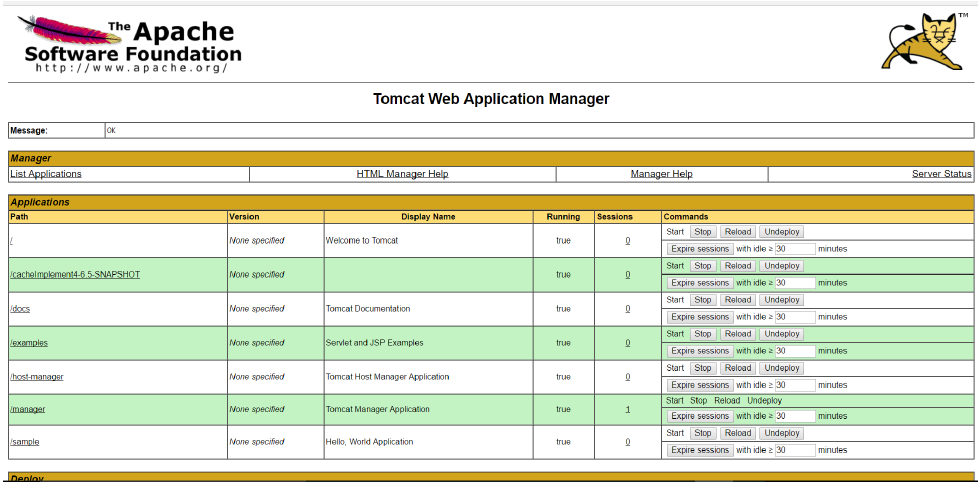


Figure 6.54: Tomcat view of deployed war

Figure 6.54 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.

**6.10.3.2 Automated installation of java**

The Figures 6.55 and 6.56 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.

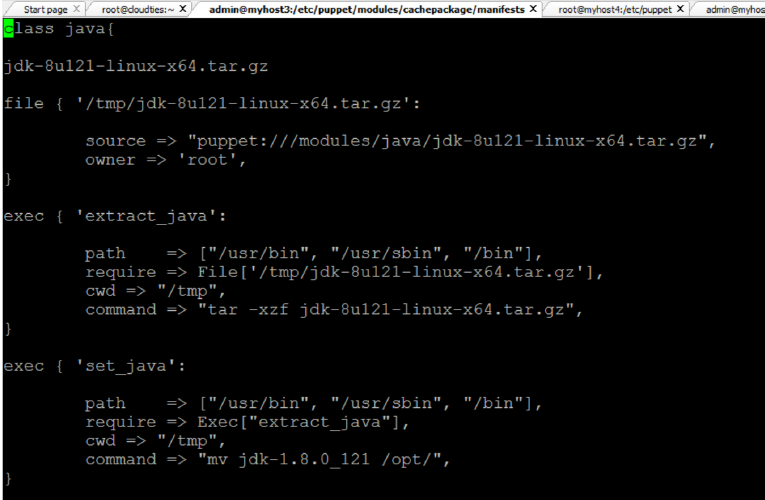


Figure 6.55: Automated java installation

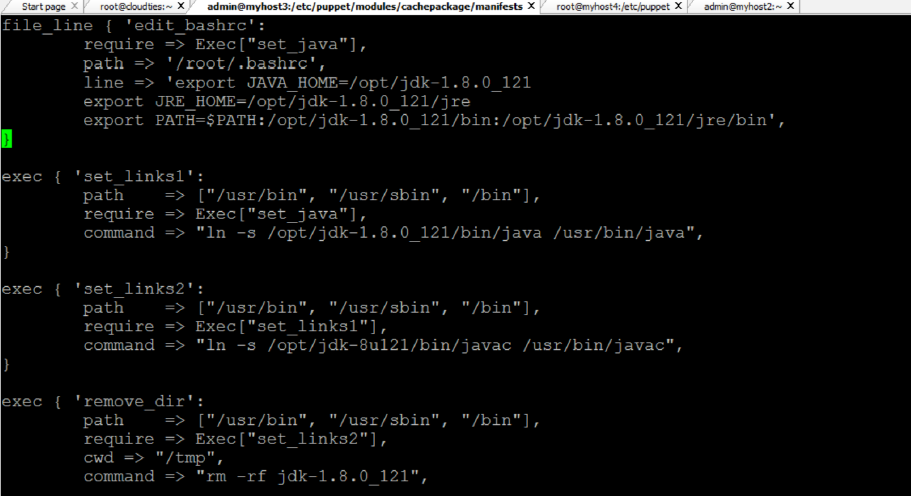


Figure 6.56: Automated java installation-2

**6.10.3.3 Automated jenkins installation**

Figure 6.57 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.

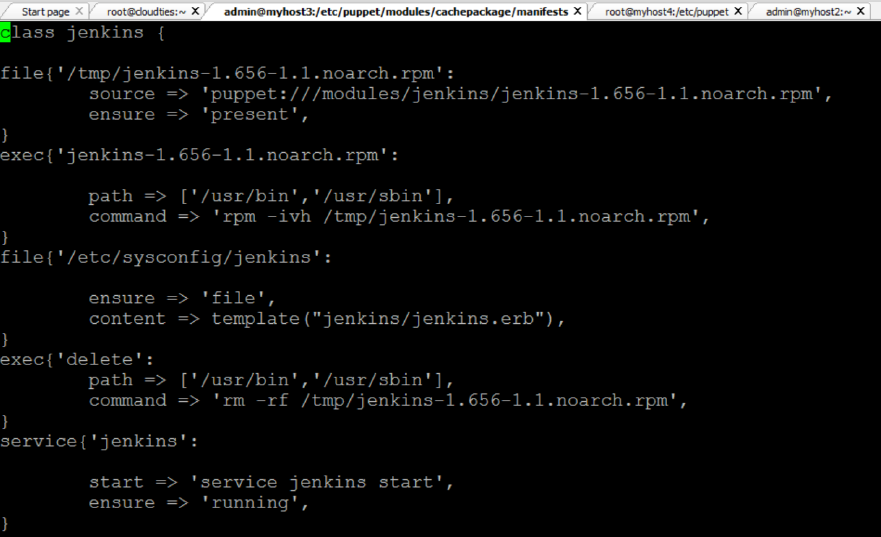


Figure 6.57: Automated jenkins installation

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

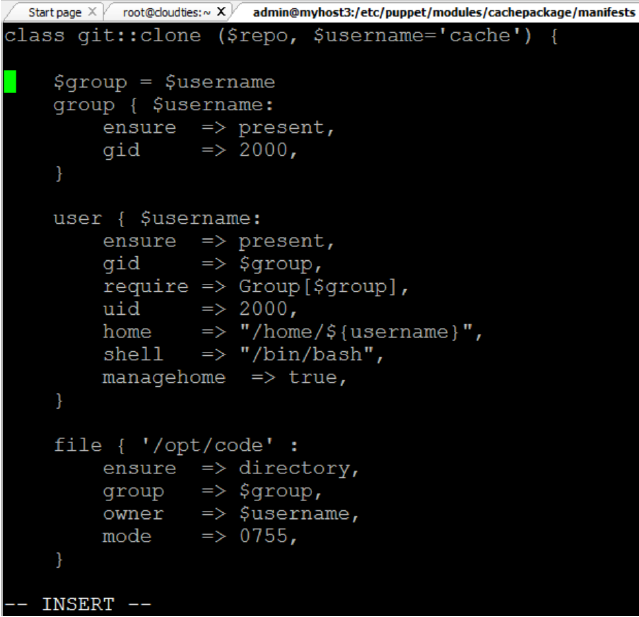


Figure 6.58: Git clone script

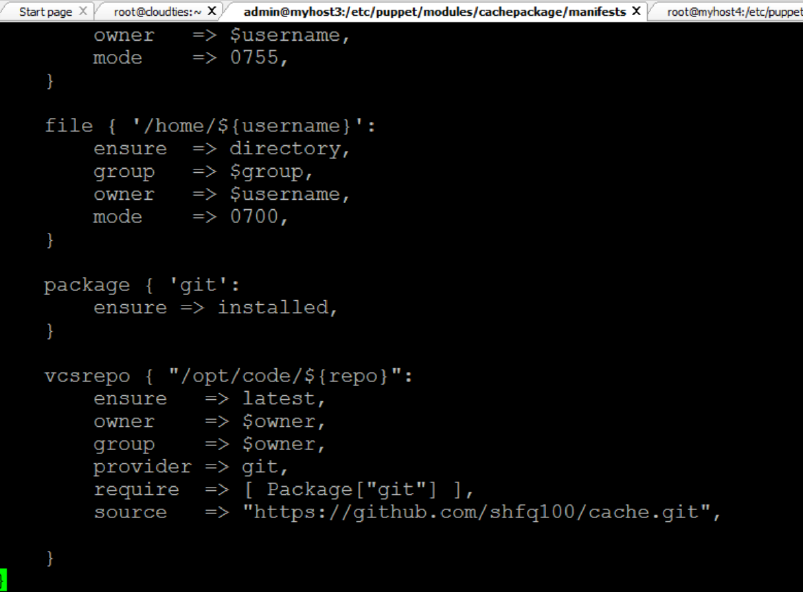


Figure 6.59: Git clone script

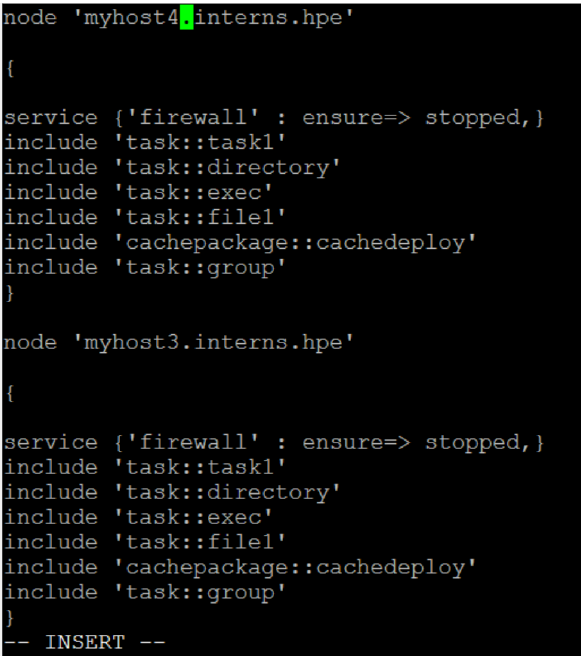


Figure 6.60: Site.pp file

The Figures 6.58 and 6.59 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 6.60 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.