DyaSaa Tech

DevOps

By

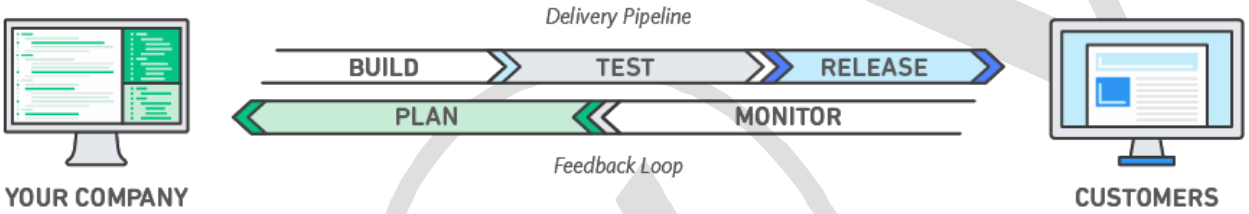
Mr. RG (IT Expert)

Content Index

DevOps

Introduction of Devops

DevOps is the combination of cultural philosophies, practices, and tools that increases an organization's ability to deliver applications and services at high velocity: evolving and improving products at a faster pace than organizations using traditional software development and infrastructure management processes. This speed enables organizations to better serve their customers and compete more effectively in the market.



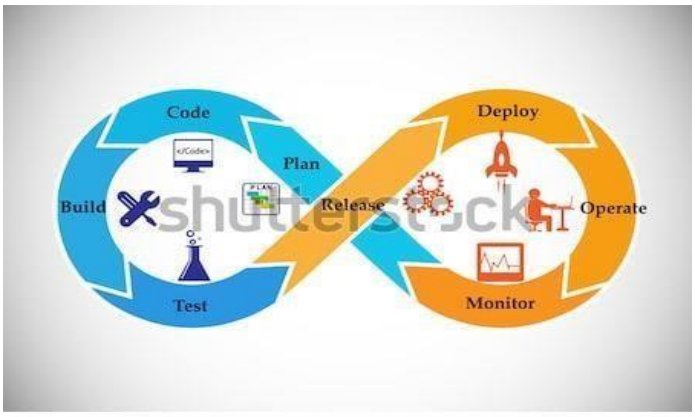
DevOps is the practice of operations and development engineers participating together in the entire service lifecycle, from design through the development process to production support.

Toolchains

As DevOps is intended to be a cross-functional mode of working, those that practice the methodology use different sets of tools—referred to as "toolchains"—rather than a single one.These toolchains are expected to fit into one or more of the following categories, reflective of key aspects of the development and delivery process.

1. **Coding** – code development and review, source code management tools, code merging
2. **Building** – continuous integration tools, build status
3. **Testing** – continuous testing tools that provide feedback on business risks
4. **Packaging** – artifact repository, application pre-deployment staging
5. **Releasing** – change management, release approvals, release automation
6. **Configuring** – infrastructure configuration and management, infrastructure as code tools
7. **Monitoring** – applications performance monitoring, end-user experience

There are different interpretations of these toolchains (e.g. Plan, Create, Verify, Package, Release, Configure, and Monitor.



Benefits of DevOps

Speed:
Move at high velocity so you can innovate for customers faster, adapt to changing markets better, and grow more efficient at driving business results. The DevOps model enables your developers and operations teams to achieve these results. For example, microservices and continuous delivery let teams take ownership of services and then release updates to them quicker.

Rapid Delivery:

Increase the frequency and pace of releases so you can innovate and improve your product faster. The quicker you can release new features and fix bugs, the faster you can respond to your customers' needs and build competitive advantage. Continuous integration and continuous delivery are practices that automate the software release process, from build to deploy.

Reliability:

Ensure the quality of application updates and infrastructure changes so you can reliably deliver at a more rapid pace while maintaining a positive experience for end users. Use practices like continuous integration and continuous delivery to test that each change is functional and safe. Monitoring and logging practices help you stay informed of performance in real-time.

Scale:

Operate and manage your infrastructure and development processes at scale. Automation and consistency help you manage complex or changing systems efficiently and with reduced risk. For example, infrastructure as code helps you manage your development, testing, and production environments in a repeatable and more efficient manner.

Improved Collaboration:
Build more effective teams under a DevOps cultural model, which emphasizes values such as ownership and accountability. Developers and operations teams collaborate closely, share many responsibilities, and combine

their workflows. This reduces inefficiencies and saves time (e.g. reduced handover periods between developers and operations, writing code that takes into account the environment in which it is run).

Security:

Move quickly while retaining control and preserving compliance. You can adopt a DevOps model without sacrificing security by using automated compliance policies, fine-grained controls, and configuration management techniques. For example, using infrastructure as code and policy as code, you can define and then track compliance at scale.

Devops for entire Business

DevOps was invented as a way to make developers and IT operations work closer together, ensuring that the barrier is removed between developers and those who use the results of their development.

Marcus Jewell, VP EMEA at data and storage networking company Brocade describes it as a "cross-disciplinary practice that spans all of the roles involved in delivering applications for an organization".

Karl Hoods, CIO of the charity Save the Children UK defines it as a way of "removing the barriers to deliver code more effectively".

That's code that builds machines, or code that we deploy. It enables us to move an awful lot quicker, so it's about making us more efficient and effective when it comes to development and deployment.

Whereas previously, developers and engineers could only evaluate their own work, the DevOps culture means that they can both instantly evaluate the scope of the entire system and assess the impact of those changes, explains Alex Mathews, EMEA technical manager at security software vendor Positive Technologies.

The general consensus is that organizations that are using software – particularly the development of apps - to innovate in their sectors, should be looking at implementing a DevOps culture.

A company offering personal training that was once manually booked and has since adopted an IT service to allow scheduling by smartphone would not at that point need DevOps. If they were to develop the service to match demand for sessions with personal trainers available near you, then DevOps would become relevant

It enables the business to innovate and test out new ideas quicker than before. Subsequently the business can deliver features and bring on new revenue streams faster, achieving the agility needed to respond immediately to marketplace opportunity, events and trends

Devops for entire IT

The functionality of codeBeamer ALM's DevOps feature set spans the entire lifecycle to help you align teams and

stakeholders, enhance collaboration, automate your CI&CD pipeline, and integrate customer feedback across the process of development. For safety-critical developers, it offers end-to-end traceability and an automatically recorded audit trail to bridge the gap between DevOps and auditors or compliance officers.

The elimination of knowledge transfers between departments

When one DevOps team owns the entire lifecycle, a single team becomes responsible for all aspects. They still rely on the support of other organizational units, but the ownership and work stay within the same team. This results in a better quality of the software and maintenance since the DevOps teams truly understand the problems and why something has been developed a certain way.

Decentralized responsibility [towards the team]

By allowing the team to adapt towards the needs of the products that they maintain, the teams can come up with better ideas for improvement and maintenance due to their previous experience.

Agile teams & processes throughout the entire application lifecycle

When the entire application lifecycle is fully agile, organizations can achieve even quicker time to market for new functionality and products.

Ops is no longer 'just' keeping the lights on

When issues come up, the DevOps team has the ability to listen to the users and implement the needed functionality or fixes.

Reduced cost and risk after the deployment

There is no handover between two teams. The same team that developed the functionality is involved in the go-live, and support during the riskiest moments.

Continuous Organizational automation

By the (Dev)Ops teams who are constantly trying to improve and speed up the most time-consuming processes. This results in faster and more reliable delivery of functionality after every release.

Devops for Developer

Devops is exciting for developers, and can also be scary. It will change what you need to know and the skills you need in order to succeed. Doing devops requires that you learn new tools and embrace deep cultural changes to the way that you think and work. You'll have to adapt to new processes in the shorter term, while also anticipating long-term organizational changes. Adopting devops means you'll learn to work differently than you have before, alongside other developers and sysadmins who are also making this big shift.

A similar transformation is at work for developers. You'll have an easier time adjusting to the changes if you are already familiar with Agile methods, but there is still a learning curve .

While DevOps requires a cultural change for developers, having the right tools right is also essential. Automating the pipeline for continuous integration and continuous delivery (CI/CD) lets developers get new features in front of end users faster, while still assuring high quality.

Devops for Testing:

Traditionally, QA would get a build which is deployed in their designated environment and QA would then commence their functional & regression testing. The build would ideally sit with the QA for a couple of days before the QA sign-off on the build. All these steps change in DevOps.

To achieve such speed and agility, it is important to automate all the testing processes and configure them to run automatically when the deployment is completed in the QA environment. Specialized automation testing tools and continuous integration tools are used to achieve this integration. This also necessitates the building of a mature automation testing framework through which one can quickly script new test cases.

The QA and dev need to sit together and identify the areas affected due to a particular build and execute those related test cases plus a sanity test pass.

Strategy around testing new features need to be formalized and the interim builds can be supplied to QA who would, in turn, create test scripts and run these automation tests on the interim builds till the code becomes stable enough to be deployed onto the Production environment.

Devops for Operations Team

Continuous deployment (CD) lets operations teams automatically release code changes and deploy them to production in short cycles, reliably and repeatedly. The use of containers has changed what's possible when it comes to CD.

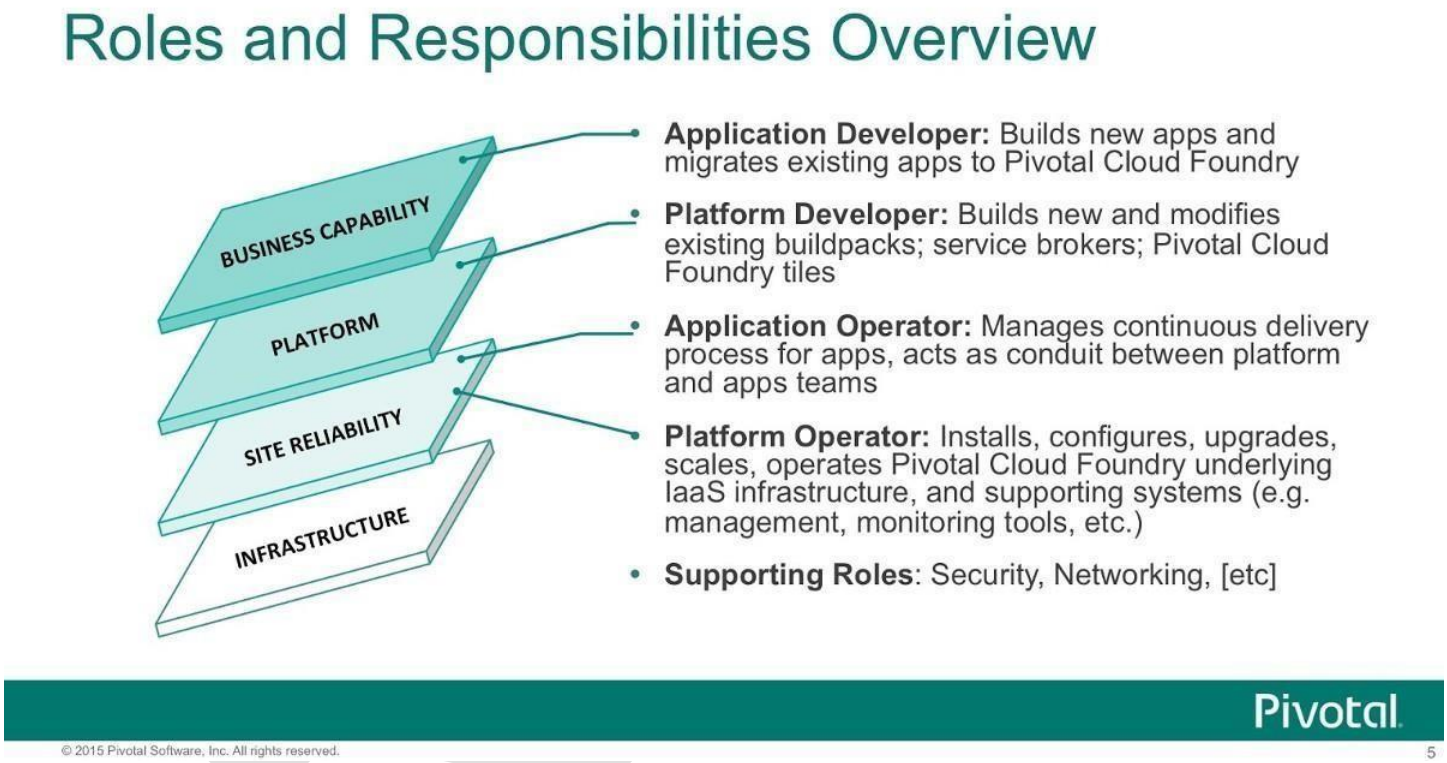
Devops replaces the change control bureaucracy and firefighting that systems administrators are accustomed to with automated continuous deployment and infrastructure as code. The sysadmin becomes an *operations engineer*, using new tools like Puppet and Chef (and Ansible and Salt and Vagrant and Docker) to design and rapidly spin up standardized, preconfigured systems in a public or private cloud. As a result, wait times go down and reliability goes up.

The Operations team has quite a significant role in the DevOps environment. A huge chunk of DevOps involves configuring, managing, provisioning and monitoring servers. Something that's perfect for an individual from operations background.

Learning DevOps enables you to find better solutions for your infrastructure, without worrying about your servers going down or you having to configure new changes manually on all the servers.

Many of the DevOps automation tools like Puppet, Docker, Kubernetes, Jenkins etc are all handled by operations team.

Role of DevOps in Agile Scrum:



Devloper/Engineer—those who not only "code," but gain the operations knowledge needed to support the application in production, with the help of…

Operations—those who work with developers on production needs and help support the application in production. This role may lessen over time as developers become more operations aware, or it may remain a dedicated role.

Product Owner/Product Manager—the "owner" of the application in development that specifies what should be done each iteration, prioritizing the list of "requirements."

Designer—studies how users interact with the software and systematically designs ways to improve that interaction. For user-facing applications this also includes visual design.

These are roles that are not always needed and sometimes be fulfilled partially by shared, but designated to the team staff:

Tester—staff that helps with the effort ensure the software does what was intended and functions properly.

Architect—in large organization, the role or architect is often used to ensure that individual teams are aligning with the larger organization's goals and strategy, while also acting as a consultative enabler to help teams be successful and share knowledge.

Data scientist—if data analysis is core to the application being developed, getting the right analysis and domain skills will be key.

GIT (--fast-version-control)

Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

Git is easy to learn and has a tiny footprint with lightning fast performance. It outclasses SCM tools like Subversion, CVS, Perforce, and ClearCase with features like cheap local branching, convenient staging areas, and multiple workflows.

Version Controlling

This is the process of preserving older and latest versions of the code. All the developers upload their code into the VCS. The VCS's accept the code uploads coming from various developers and create one integrated project out of those uploads. The next time when developers download the code from the VCS they will find the code created by the entire team. Process of uploading into VCS is called "cehckin" and downloading from VCS is called "checkout"

VCS's also preserve the different versions of the code so that the developers can swith between any version based on requirement.VCS's also track who is making what kind of changes VCS's are of two types

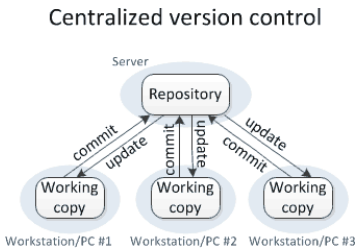
1. Centralised Version Controlling

2. Distributred Version Controlling

Centralised Version Controlling

Here we have a remote server where version controlling happens.All the developers checkin their code into this central server.

Eg: SVN(SubVersion)



Distributred Version Controlling

Here a local repository is installed on every developers machine where version controlling happens at the level of the individual developer.Later from the LR the code is uploaded to the remote repository where version controlling happens at the level of the complete team.

Installation and configuration

Installing git on windows

1. Open https://git-scm.com/downloads
2. Download git for windows--->Install it
3. Open git bash
4. Execute the git commands

Installing git on a Linux(Ubuntu)

1. Update the apt-repository

   sudo apt-get update

2. Install git

   sudo apt-get install -y git

To check the version of git

git –verion

Configuration:

Setting username and email globally for all users

git config --global user.name "sai krishna"

git config --global user.email "sai@gmail.com"

To see the list of all default configuration

git config --global --list

Local Repo:

Git when working on the local machine uses three sections.

1. Working directory (workspace)
2. Stagging Area
3. Local Repository

Working directory

Initially all the files created by the developers are stroed in a folder called working directory and these files are initially called untracked files.

Stagging Area

Files from working directory are moved to an intermediate memory area called stagging area.

The files present here are called stagged files.

Local Repository

All files from stagging area will be moved into the local repository and this is where version controlling happens.These files are called commited files

Basic commands

1. To initilise the working dir to accept git commands. Open git bash

cd path_of_working_dir

git init

The above command will create a hidden folder called .git

where it stores some configuration files for maintianing the local repository

2. To send file from working dir to stagging area

git add filename

To send multiple files into stagging area

git add file1 file2 file3

To send all files and fodlers into the stagging area

git add .

3. To unstage files ie bring files back from stagging to untracked section

git rm --cached filename

(or)

git reset filename

4. To send files from stagging to local repository

git commit -m "Some commit message"

5. To see the status of the untracked and stagging section

git status

6. To see the total commits that are done in LR

git log

To see the commit history in a simplified format

git log --oneline

Git log and git reflog:

git log shows the current HEAD and its ancestry. That is, it prints the commit HEAD points to, then its parent, its parent, and so on. It traverses back through the repo's ancestry, by recursively looking up each commit's parent.

(In practice, some commits have more than one parent. To see a more representative log, use a command like git log --oneline --graph --decorate.)

git reflog doesn't traverse HEAD's ancestry at all. The reflog is an ordered list of the commits that HEAD has pointed to: it's undo history for your repo. The reflog isn't part of the repo itself (it's stored separately to the commits themselves) and isn't included in pushes, fetches or clones; it's purely local.

Aside: understanding the reflog means you can't really lose data from your repo once it's been committed. If you accidentally reset to an older commit, or rebase wrongly, or any other operation that visually "removes" commits, you can use the reflog to see where you were before and *git reset --hard* back to that ref to restore your previous state. Remember, refs imply not just the commit but the entire history behind it.

.gitignore

This is a special file where we can store the private filenames. Any filename that is mentioned in .gitignore will not longer be accessed by git

1. Create few files

   touch file1 file2 file3 file4

2. Check the status of git

   git status

   It will show all the above 4 files as untracked file

3. Create a hidden file .gitignore and enter the above filenames

   cat > .gitignore

   file1

   file2

   file3

   file4

   To come out of cat command press ctrl+d (EOF)

4. Check the status of git

   git status

   It will not longer show the above created four files as untracked

Git Branching

This is a feature of git using which the developers can create seperate functionalities of code on different branches and later merge them with the master branch.This will help in creating the code in an unclutterred way

1. To see the list of all branches

   git branch

   To see all branches (local and remote)

   git branch -a

2. To create a branch

   git branch branchname

3. To move into a branch

   git checkout branchname

4. To create a branch and also move into it

   git checkout -b branchname

5. To merge a brnach with master

   First move to master

   git checkout master

   git merge branchname

6. To delete a branch that is merged

   git branch -d branchname

   This is also called soft delete

7. To delete a branch that is not merged

   git branch -D branchname

   This is also called hard delete

Note: whenever a branch is created whatever is the commit history of the parent branch will be copied to the newly created branch

Note: Irrespective of where a file is created or modified that file belongs to that branch form where it is commited

Working on Git remote repository ("github")

1. Open github.com
2. Signup for a free account
3. Signin into that account

To push the code from local repository to remote repository

1. Signin into github.com
2. Click on + on top right corner
3. click on New repository
4. give some name for repository
5. Select public--->Click on Create repository
6. Go to "Push an exisiting repository from command line"
7. Copy the 1st command and paste in gitbash

   This will create a link between the local repository and the remote repository
8. Copy the second command and paste in gitbash

   Enter username and password of github this process is called checkin

Downloading from remote git repository

This can be done in 3 ways

1 git clone

2 git fetch

3 git pull

git clone

This will download the entire remote repository from git hub into individual developers machine irresepective of whether that code is already present or not

Syntax

git clone remote_repo_url

git fetch

This will work only when there are modifications in the code between the local repository and remote repository

git fetch will download the modified files and place them on a remote branch.We can go to that remote branch check if those changes are accptable or not and if they are acceptable we can merge them with master

1. Create some modifications to any file on github

   Open github.com

   Click on our remote repository

   click on any file that we want to modify

   Click on Edit icon

   Make some changes--->click on commit changes

2. In git bash

   git fetch

3. Check the modified file on master branch

   We will not see any modifications

4. The modifications will be on the remote branch

   Move to the remote branch

   git branch -a

   git checkout -b remotes/origin/master

5. View the modified file

If the changes are accpetable merge with master

git checkout master

git merge remotes/origin/master

git pull:

This will also work only when there are modifications between the local repository and the remote repository. But it will directly merge the modified files with master branch

1. Open github.com

2. Make some changes to a file--->commit changes

3. In git bash

git pull

The modified files can be directly seen on the master branch

Git merge and git rebase

git merge:

In git merge the commits are inserted into the branch that we are merging exactly at the location that they were created. Which means the commits coming from a branch can look like older commits on the master

1. On the master create few commits

touch f1

git add .

git commit -m "a"

touch f2

git add .

git commit -m "b"

2. Create a new branch and create few commits on it

git checkout -b sprint-1

touch f3

git add .

git commit -m "c"

touch f4

git add .

git commit -m "d"

3. Move back to master and create few more commits

git checkout master

touch f5

git add .

git commit -m "e"

touch f6

git add .

git commit -m "f"

4. Merge sprint-1 with master

git merge sprint1

5Delete sprint-1 branch

git branch -d sprint-1

6Check commit history of master

git log –oneline

git rebase:

This is used for performing a fastforward merge ie the commits coming from the branch are added to the top of the master branch.This is helpful when we want the commits coming from a branch to be reflected as the latest working version of code on master.

1. On the master create few commits

```
touch f1
git add .
git commit -m "a"
touch f2
git add .
git commit -m "b"
```

2. Create a new branch and create few commits on it

```
git checkout -b sprint-1
touch f3
git add .
git commit -m "c"
touch f4
git add .
git commit -m "d"
```

3. Move back to master and create few more commits

```
git checkout master
touch f5
git add .
git commit -m "e"
touch f6
git add .
git commit -m "f"
```

4. Rebase sprint-1 with master

git checkout sprint-1

git rebase master

git checkout master

git merge sprint-1

Rearranging the commit history

The commit history can be changed according to our requirment using the git rebase command. The latest commit or the top most commit is genrally called HEAD.We can make this HEAD point to any older commit and make that older commit as the latest commit

Note: The very first commit is called as initial commit and this cannot be rearranged

git rebase -i HEAD~4

The number 4 represents the top 4 comits that we want to rearrange

The above command will open the commit hsitory in vi editor where can can simply change the commit order

Save and Quit Esc :wq Enter

Git Squash

Squash is merging of commits to make multiple commits to make it look like a single commit.

git rebase -i HEAD~4

The above command will open in vi editor

where the complete commit history is shown

Remove the pick word and replace it with squash

Save and quit esc : wq Enter

Check the commit history

git log --oneline

Modifing exisiting commits

Whenever we modify a file or create new files generally we create a new commit.Instead we can put the modifications in the existing commit itself rather than creating a new commit.

This can be done using git **amend** command

1. Create few commits in git LR

   touch f1

   git add .

   git commit -m "a"

   touch f2

   git add .

   git commit -m "b"

   touch f3

   git add .

   git commit -m "c"

2. Check the commit history

   git log --oneline

3. Modify some files or create new files

   cat > f3

   Put some data

   ctrl+d to comeout of cat

4. check status of git

   git status

   We will find a modified file

5. Send it to stagging area

git add .

6. Commit it to local repositry as an exisiting commmit

git commit --amend -m "c"

7. Check the commit history

git log --oneline

We will see that new commit is not created the changes are

pushed into the exisiting "c" commit

Note: git amend actually creates a new commit.

The older "c" commit becomes an orphaned commit which Can not be see using git log

We can use "git reflog" for seeing the entire commit history.ie active and orphaned commits

Git cherry pick

This is used for choosing which commits we want to take into the master branch Generally when we perform "git merge" or "git rebase" all the commits of that branch will come into master branch

Cherry pick will allow us to select only those commits that we require and merge them with master

1. Create few commits on master

touch f1

git add .

git commit -m "a"

touch f2

git add .

git commit -m "b"

2. Create a branch and some commits on it

git checkout -b sprint-1

3. Create some commits here

touch f3

git add .

git commit -m "c

touch f4

git add .

git commit -m "d"

touch f5

git add .

git commit -m "e"

4. Check the commit history on sprint-1 branch

git log --oneline

5. Identify the commits that we want from sprint-1 branch and move back to master branch

git checkout master

git cherry-pick commit_id1 commit_id2

6. Check the commit history on master

git log --oneline

git stash

This is a feature of git which is used for leaving unfinished work and start a new functionality related coding.

Further command nof git should not touch the unfinished files

1. To stash the files present in stagging area

git stash

2. To stash the files present in stagging area and untracked section

git stash -u

3. To stash the files present in stagging area, untracked section and also the .gitignore

git stash -a

4. To see the list of all stashes that we have done

git stash list

5. To unstash the latest stash

git stash pop

6. To unstash any older stash

git stash pop stash{stashno}

Eg: To unstash the second last stash

git stash pop stash{1}

Next case on stash

1. Create a file and send to stagging area

touch f1

git add .

2. Stash the above file

git stash

3. Create a new file and stash it

touch f2

git stash -u

4. Create few new files and place them in .gitignore

Stash .gitignore also

touch f3 f4 f5

cat > .gitignore

f3

f4

f5

Press Ctrl+d to come out of cat command

5. Since f3, f4, f5 are put in .gitignore git status will no longer

show them as untracked files but it will show .gitignore as untracked file.ie further commands of git will send this .gitignore into stagging area and also in local and remote repositories.If we want ot avoid that and .gitignore should not be accessed by git

git stash –a

git squash:

One very nice feature of Git is the ability to rewrite the history of commit.
The principal reason for doing this is that a lot of such history is relevant only for the developer who generated it, so it must be simplified, or made more nice, before submitting it to a shared repository.

Squashing a commit means, from an idiomatic point of view, to move the changes introduced in said commit into its parent so that you end with one commit out of twos.
If you repeat this process multiple times, you can reduce *n* commit to a single one.

Use git rebase -i <after-this-commit> and replace "pick" on the second and subsequent commits with "squash" or "fixup", as described in the manual.
In this example, <after-this-commit> is either the SHA1 hash or the relative location from the HEAD of the current branch from which commits are analyzed for the rebase command. For example, if the user wishes to view 5 commits from the current HEAD in the past the command is git rebase -i HEAD~5

Tagging in Git

Tags are used for placing bookmarks on commits. They are to specify info related to who tagged , when it was tagged and why it was tagged. Generally used for releases. This helps in understanding what are the commits that are related to specific releases

Tags are of two types

1. Light weight tags

2. Annoted tags

Light weight tags

1. To create a light weight tag

   git tag tabname

2. To see the list of all the tags

   git tag

3. To create an annoted tag

   git tag -a tagname -m "some message"

Note: Tags are always created for the latest commit (HEAD)

4. To create tags for an older commit

   git tag -a tagname -m "message" commit_id

5. To delete a localtag

   git tag -d tagname

6. To push all tags into github

   git push --tags

7. To delete tags from the remote github

   git push origin :tagname

git diff

This is used for finding the difference between 2 commits or it can be used for finding the difference between a commit and a file yet to be commited

1. To find diff between 2 commits

   git diff commit1_id commit2_id

2. To find diff between latest commit and a file

git diff HEAD filename

git revert:

Given one or more existing commits, revert the changes that the related patches introduce, and record some new commits that record them. This requires your working tree to be clean (no modifications from the HEAD commit).

Note: *git revert* is used to record some new commits to reverse the effect of some earlier commits (often only a faulty one). If you want to throw away all uncommitted changes in your working directory, you should see git-reset[1], particularly the `--hard` option. If you want to extract specific files as they were in another commit, you should see git-checkout[1], specifically the `git checkout <commit> -- <filename>` syntax. Take care with these alternatives as both will discard uncommitted changes in your working directory.

usage

Git revert <commit-id>

Maven (Build Tool)

Introduction

Maven is a project management and comprehension tool that provides developers a complete build lifecycle 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 lifecycle.

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.

Objective

The primary goal of Maven is to provide developer with the following −

- 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 xml file, pom.xml, referred as Project Object Model (POM), which is the fundamental unit of the entire Maven system. In later chapters, we will explain POM in detail.

Build Lifecycle (Maven stages)

Maven is based around the central concept of a build lifecycle. What this means is that the process for building and distributing a particular artifact (project) is clearly defined.

For the person building a project, this means that it is only necessary to learn a small set of commands to build any Maven project, and the **POM**will ensure they get the results they desired.

There are three built-in build lifecycles: default, clean and site. The default lifecycle handles your project deployment, the clean lifecycle handles project cleaning, while the site lifecycle handles the creation of your project's site documentation.

A Build Lifecycle is Made Up of Phases:

Each of these build lifecycles is defined by a different list of build phases, wherein a build phase represents a stage in the lifecycle.

For example, the default lifecycle comprises of the following phases (for a complete list of the lifecycle phases, refer to the Lifecycle Reference):

Validate:

validate the project is correct and all necessary information is available

Compile:

compile the source code of the project

Test:

Test the compiled source code using a suitable unit testing framework. These tests should not require the code be packaged or deployed

Package:

Take the compiled code and package it in its distributable format, such as a JAR.

Verify:

run any checks on results of integration tests to ensure quality criteria are met

Install:

install the package into the local repository, for use as a dependency in other projects locally

Deploy:

Deploy done in the build environment, copies the final package to the remote repository for sharing with other developers and projects.

These lifecycle phases (plus the other lifecycle phases not shown here) are executed sequentially to complete the **default** lifecycle. Given the lifecycle phases above, this means that when the default lifecycle is used, Maven will first validate the project, then will try to compile the sources, run those against the tests, package the binaries (e.g. jar), run integration tests against that package, verify the integration tests, install the verified package to the local repository, then deploy the installed package to a remote repository.

Note: Based on our goals we use these phases/stages

We want to compile we use **(mvn compile**)

We want generate artifacr we use (**mvn package**)

We want to deploy our application use (**mvn deploy**)

(after configuring related to these phases in pom.xml then only it is run successfully)

These phases are fallowed sequential order (if we make **mvn compile** first run **mvn validate**, **mvn compile**, **mvn test** ofter generate artifact that is **mvn package**)

Maven installation and configuration

Install in windows:

1. Go to this link https://maven.apache.org/download.cgi
2. Go to Files section click on apache-maven-version.zip

Install in Linux:

Open Linux terminal

Type ‘sudo apt-get install –y maven’

Configure:

Set M2_HOME:

1. Go to computer/pc properties
2. Go to Advanced system settings
3. Go to environment variables

4. Select system variables
5. Click on New Variable name=”M2_HOME” and variable value=”C:\Users\nagesh\Downloads\apache-maven-3.5.4-bin\apache-maven-3.5.4”. **note:** copy value before bin

Include maven in path:

1. Go to computer/pc properties
2. Go to Advanced system settings
3. Go to environment variables
4. Select system variables
6. Click on path variable if already exist otherwise click on New Variable name=”path” variable value=”; C:\Users\nagesh\Downloads\apache-maven-3.5.4-bin\apache-maven-3.5.4\bin”

Note: maven is based on java so we install java and set the JAVA_HOME and include java path in path variable

Creating maven from command prompt

Gi to Linux Terminal

mvn archetype:generate

Choose a number or apply filter (format: [groupId:]artifactId, case sensitive contains): 1320:

Note: The type and stracture of the project is depended on the above number. 1324 is default java project 1728 is empty web application.

Choose org.fusesource.scalate.tooling:scalate-archetype-jersey_2.10 version:

1: 1.6.0
2: 1.6.1
Choose a number: 2: 2

Note: the above number indicate the project version
Define value for property 'groupId': itpointgroup
Note: it is project groupid
Define value for property 'artifactId': itpoint
Note: the artifact is created with the above name
Define value for property 'artifactId': itpoint
Note: it is the artifact property name
Define value for property 'version' 1.0-SNAPSHOT: : 1.5

Note: it is the version of the artifact
Define value for property 'package' itpointgroup: : itpoint.com
Define value for property 'package' mygroup: : itpoint.com
Confirm properties configuration:
groupId: mygroup
artifactId: itpoint
version: 1.5
package: itpoint.com
Y: : y

Note: The project is created with artifact name ofter entering “y “

Maven Dependencies

One of the core features of Maven is Dependency Management. Managing dependencies is a difficult task once we've to deal with multi-module projects (consisting of hundreds of modules/sub-projects). Maven provides a high degree of control to manage such scenarios.

Transitive Dependencies Discovery:

It is pretty often a case, when a library, say A, depends upon other library, say B. In case another project C wants to use A, then that project requires to use library B too.

Maven helps to avoid such requirements to discover all the libraries required. Maven does so by reading project files (pom.xml) of dependencies, figure out their dependencies and so on.

We only need to define direct dependency in each project pom. Maven handles the rest automatically.

Note: go to projectdirectory (cd artifactid) then you find pom.xml file in this file all dependencies and plugins are configured based on our project requirement

<dependencies><dependency><groupId>org.scala-lang</groupId><artifactId>scala-library</artifactId><version>${scala-version}</version></dependency><dependency><groupId>org.scala-lang</groupId><artifactId>scala-compiler</artifactId><version>${scala-version}</version></dependency><dependency><groupId>javax.servlet</groupId><artifactId>servlet-api</artifactId><version>${servlet-api-version}</version><scope>provided</scope></dependency><dependency><groupId>org.fusesource.scalate</groupId><artifactId>scalate-page_2.10</artifactId><version>${scalate-

version}</version></dependency><dependency><groupId>com.sun.jersey</groupId><artifactId>jersey-server</artifactId><version>${jersey-version}</version></dependency><dependency><groupId>ch.qos.logback</groupId><artifactId>logback-classic</artifactId><version>${logback-version}</version><scope>runtime</scope><optional>true</optional></dependency>

<!-- testing -->

<dependency><groupId>org.fusesource.scalate</groupId><artifactId>scalate-test_2.10</artifactId><version>${scalate-version}</version><scope>test</scope>

</dependency>

</dependencies>

Note:

By default minium reqiured dependencies are added to run the project and type dependencies are based on project type.

If we required some more libraries to build our project then add all necessary dependencies into these depemdecies section. Ex . our project is connected to database then add mysql dependencies in dependency section.

You can also previous maven project by adding that project property details in dependency section

When we build the project these all dependencies library files downloaded into resources section and add to classpath variable

Maven repositories

Artifact Repositories

A repository in Maven holds build artifacts and dependencies of varying types.

There are exactly two types of repositories: local and remote. The local repository is a directory on the computer where Maven runs. It caches remote downloads and contains temporary build artifacts that you have not yet released.

Remote repositories refer to any other type of repository, accessed by a variety of protocols such as file:// and http://. These repositories might be a truly remote repository set up by a third party to provide

their artifacts for downloading (for example, repo.maven.apache.org and uk.maven.org house Maven's central repository). Other "remote" repositories may be internal repositories set up on a file or HTTP server within your company, used to share private artifacts between development teams and for releases.

Local and remote repositories are structured the same way so that scripts can run on either side, or they can be synced for offline use. The layout of the repositories is completely transparent to the Maven user, however.

Using Repositories

In general, you should not need to do anything with the local repository on a regular basis, except clean it out if you are short on disk space (or erase it completely if you are willing to download everything again).

For the remote repositories, they are used for both downloading and uploading (if you have the permission to do so).

Downloading from a Remote Repository

Downloading in Maven is triggered by a project declaring a dependency that is not present in the local repository (or for a SNAPSHOT, when the remote repository contains one that is newer). By default, Maven will download from the central repository.

To override this, you need to specify a mirror as shown in Using Mirrors for Repositories

You can set this in your settings.xml file to globally use a certain mirror. However, it is common for a project to customise the repository in its pom.xml and that your setting will take precedence. If dependencies are not being found, check that you have not overridden the remote repository.

For more information on dependencies, see Dependency Mechanism.

Using Mirrors for the Central Repository

There are several official Central repositories geographically distributed. You can make changes to your settings.xml file to use one or more mirrors. Instructions for this can be found in the guide Using Mirrors for Repositories.

Building Offline

If you are temporarily disconnected from the internet and you need to build your projects offline, you can use the offline switch on the CLI:

```
1. mvn -o package
```

Many plugins honor the offline setting and do not perform any operations that connect to the internet. Some examples are resolving Javadoc links and link checking the site.

Uploading to a Remote Repository

Using the internal repository is quite simple. Simply make a change to add a repositories element:

While this is possible for any type of remote repository, you must have the permission to do so. To have someone upload to the central Maven repository, see Repository Center.

Internal Repositories

When using Maven, particularly in a corporate environment, connecting to the internet to download dependencies is not acceptable for security, speed or bandwidth reasons. For that reason, it is desirable to set up an internal repository to house a copy of artifacts, and to publish private artifacts to.

Such an internal repository can be downloaded using HTTP or the file system (with a file:// URL), and uploaded to using SCP, FTP, or a file copy.

As far as Maven is concerned, there is nothing special about this repository: it is another **remote repository** that contains artifacts to download to a user's local cache, and is a publish destination for artifact releases.

Additionally, you may want to share the repository server with your generated project sites. For more information on creating and deploying sites, see Creating a Site.

Setting up the Internal Repository

To set up an internal repository just requires that you have a place to put it, and then copy required artifacts there using the same layout as in a remote repository such as repo.maven.apache.org.

It is *not* recommended that you scrape or rsync:// a full copy of central as there is a large amount of data there and doing so will get you banned. You can use a program such as those described on the Repository Management page to run your internal repository's server, download from the internet as required, and then hold the artifacts in your internal repository for faster downloading later.

The other options available are to manually download and vet releases, then copy them to the internal repository, or to have Maven download them for a user, and manually upload the vetted artifacts to the internal repository which is used for releases. This step is the only one available for artifacts where the license forbids their distribution automatically, such as several J2EE JARs provided by Sun. Refer to the Guide to coping with SUN JARs document for more information.

It should be noted that Maven intends to include enhanced support for such features in the future, including click through licenses on downloading, and verification of signatures.

Using the Internal Repository

```
<project>
  ...
  <repositories>
    <repository>
      <id>my-internal-site</id>
      <url>http://myserver/repo</url>
    </repository>
  </repositories>
  ...
</project>
```

If your internal repository requires authentication, the id element can be used in your settings file to specify login information.

Deploying to the Internal Repository

One of the most important reasons to have one or more internal repositories is to be able to publish your own private releases.

To publish to the repository, you will need to have access via one of SCP, SFTP, FTP, WebDAV, or the filesystem. Connectivity is accomplished with the various wagons. Some wagons may need to be added as extension to your build.

Repositories present in our web appliocation:

<repositories><repository><id>repo1.maven</id><name>Maven Central Repo</name><url>http://repo1.maven.org/maven2/</url></repository><repository><id>fusesource.m2</id><name>FuseSource Community Release Repository</name><url>http://repo.fusesource.com/nexus/content/repositories/public</url><snapshots><enabled>false</enabled></snapshots><releases><enabled>true</enabled></releases></repository><repository><id>fusesource.m2-snapshot</id><name>FuseSource Community Snapshot Repository</name><url>http://repo.fusesource.com/nexus/content/repositories/snapshots</url><snapshots><enabled>true</enabled></snapshots><releases><enabled>false</enabled></releases></repository><repository><id>java.net.m2</id><name>java.net Maven 2 Repo</name><url>http://download.java.net/maven/2</url></repository><repository><id>openqa-releases</id><name>OpenQA Releases</name><url>http://archiva.openqa.org/repository/releases</url><releases><enabled>true</enabled></releases><snapshots><enabled>false</enabled></snapshots></repository><repository><id>glassfish-repo-

archive</id><name>Nexus repository collection for Glassfish</name><url>http://maven.glassfish.org/content/groups/glassfish</url><snapshots><updatePolicy>never</updatePolicy></snapshots></repository><repository><id>scala</id><name>Scala Tools</name><url>http://scala-tools.org/repo-releases</url><releases><enabled>true</enabled></releases><snapshots><enabled>false</enabled></snapshots></repository><repository><id>snapshots.scala-tools.org</id><name>Scala-Tools Maven2 Snapshot Repository</name><url>http://scala-tools.org/repo-snapshots</url><snapshots><enabled>true</enabled>

<!--updatePolicy>never</updatePolicy-->

<checksumPolicy>fail</checksumPolicy></snapshots><releases><enabled>true</enabled>

<!--updatePolicy>never</updatePolicy-->

<checksumPolicy>fail</checksumPolicy></releases></repository><repository><id>apache.snapshots</id><name>Apache Development Snapshot Repository</name><url>https://repository.apache.org/content/repositories/snapshots</url><releases><enabled>false</enabled></releases><snapshots><enabled>true</enabled></snapshots></repository></repositorie>

Maven Plugins:

Maven is actually a plugin execution framework where every task is actually done by plugins. Maven Plugins are generally used to −

- create jar file
- create war file
- compile code files
- unit testing of code
- create project documentation
- create project reports

A plugin generally provides a set of goals, which can be executed using the following syntax −

```
mvn [plugin-name]:[goal-name]
```

For example, a Java project can be compiled with the maven-compiler-plugin's compile-goal by running the following command.

```
mvn compiler:compile
```

Plugin Types:

Maven provided the following two types of Plugins

Build plugins

They execute during the build process and should be configured in the <build/> element of pom.xml.

Reporting plugins

They execute during the site generation process and they should be configured in the <reporting/> element of the pom.xml.

Following is the list of few common plugins

clean

Cleans up target after the build. Deletes the target directory.

compiler

Compiles Java source files.

surefire

Runs the JUnit unit tests. Creates test reports.

jar

Builds a JAR file from the current project.

war

Builds a WAR file from the current project.

javadoc

Generates Javadoc for the project.

antrun

Runs a set of ant tasks from any phase mentioned of the build.

List of plugins added to our project:

<plugins><plugin><groupId>net.alchim31.maven</groupId><artifactId>scala-maven-plugin</artifactId><version>${scala-plugin-version}</version><executions><execution><goals><goal>compile</goal><goal>testCompile</goal></goals></execution></executions><configuration><jvmArgs><jvmArg>-Xmx1024m</jvmArg></jvmArgs><args>

<!-- arg>-unchecked</arg -->

<arg>-deprecation</arg></args><scala-version>${scala-version}</scala-version><recompileMode>incremental</recompileMode></configuration></plugin><plugin><artifactId>maven-surefire-plugin</artifactId><version>${surefire-plugin-version}</version><configuration><forkMode>once</forkMode>

<!-- these settings are mandatory to avoid SureFire giving a bogus system property to the web container -->

<useSystemClassLoader>false</useSystemClassLoader><useManifestOnlyJar>false</useManifestOnlyJar><includes><include>**/*Test.*</include></includes></configuration></plugin><plugin><artifactId>maven-war-plugin</artifactId><version>${war-plugin-version}</version><configuration><attachClasses>true</attachClasses></configuration></plugin><plugin><groupId>org.mortbay.jetty</groupId><artifactId>jetty-maven-plugin</artifactId><version>${jetty-plugin-version}</version><configuration><webAppConfig><contextPath>/</contextPath></webAppConfig><systemProperties>

<!-- enable easy JMX connection to JConsole -->

<systemProperty><name>com.sun.management.jmxremote</name><value/></systemProperty><systemProperty><name>scalate.editor</name><value>${scalateEditor}</value></systemProperty><systemProperty><name>scalate.workdir</name><value>${scalate.workdir}</value></systemProperty><systemProperty><name>scalate.package.resources</name><value>${scalate.package.resources}</value></systemProperty><systemProperty><name>scalate.mode</name><value>${scalate.mode}</value></systemProperty></systemProperties><scanIntervalSeconds>0</scanIntervalSeconds></configuration></plugin><plugin><groupId>org.codehaus.mojo</groupId><artifactId>tomcat-maven-plugin</artifactId><version>${tomcat-plugin-version}</version><configuration><path>/</path><systemProperties>

<!-- enable easy JMX connection to JConsole -->

<com.sun.management.jmxremote/>

<!-- Scalate console configuration -->

<scalate.mode>${scalate.mode}</scalate.mode><scalate.editor>${scalateEditor}</scalate.editor><scalate.workdir>${scalate.workdir}</scalate.workdir><scalate.package.resources>${scalate.package.resources}</scalate.package.resources></systemProperties></configuration></plugin><plugin><groupId>org.fusesource.scal

ate</groupId><artifactId>maven-scalate-plugin_2.10</artifactId><version>${scalate-version}</version><executions><execution><goals><goal>precompile</goal></goals></execution></executions></plugin></plugins>

Note: we can control everything by configuring pom.xml file in maven

Integrating maven with Jenkins

1. Download and install the jenkins
2. Download and install the maven
3. Set the Home variable and path variable for maven
4. Run the jenkins (http://localhost:8080)
5. Then goto manage Jenkins
6. Goto global configuration
7. Click on maveninstallation
8. Uncheck the button install automatically
9. Variable Name is for MAVEN_HOME give any name
10. Value is maven directory location before bin folder it is the maven home
11. Click on apply and save
12. Maven is depends java libraries so we can also set the JAVA_HOME
13. Go to manage jenkins and goto global tool configuration
14. Click on AddJdkInstallation button and uncheck install automatically
15. Name is for JAVA_HOME you can give any name
16. Value is java installation directory location before bin folder
17. Click on apply and save
18. Click on New item
19. Give some name for project and select freestyle project click on okey
20. Goto project configuration and select build section
21. Click on Addbuildstep icon and select **Invoke Top level maven Targets** for maven project.

22. Specify goals as **package**

23. Click on apply and save

24. Build the project and the artifact is generated using maven

Note: Before generating we need to down load maven project (must include pom.xml) by using scm tools

Ant (Build Tool)

ANT stands for Another Neat Tool. It is a Java-based build tool from Apache. Before going into the details of Apache Ant, let us first understand why we need a build tool in the first place.

Need for a Build Tool

On an average, a developer spends a substantial amount of time doing mundane tasks like build and deployment that include:

- Compiling the code
- Packaging the binaries
- Deploying the binaries to the test server
- Testing the changes
- Copying the code from one location to another

To automate and simplify the above tasks, Apache Ant is useful. It is an Operating System build and deployment tool that can be executed from the command line.

Features of Apache Ant

- Ant is the most complete Java build and deployment tool available.
- Ant is platform neutral and can handle platform specific properties such as file separators.
- Ant can be used to perform platform specific tasks such as modifying the modified time of a file using 'touch' command.
- Ant scripts are written using plain XML. If you are already familiar with XML, you can learn Ant pretty quickly.
- Ant is good at automating complicated repetitive tasks.
- Ant comes with a big list of predefined tasks
- Ant provides an interface to develop custom tasks.
- Ant can be easily invoked from the command line and it can integrate with free and commercial IDEs.

Configuring ANT:

Apache Ant Ant is distributed under the Apache Software License, a fully-fledged open source license certified by the open source initiative.

The latest Apache Ant version, including its full-source code, class files, and documentation can be found at http://ant.apache.org.

Installing Apache Ant

It is assumed that you have already downloaded and installed Java Development Kit (JDK) on your computer. If not, please follow the instructions here.

- Ensure that the JAVA_HOME environment variable is set to the folder where your JDK is installed.
- Download the binaries from https://ant.apache.org
- Unzip the zip file to a convenient location c:\folder. using Winzip, winRAR, 7-zip or similar tools.
- Create a new environment variable called **ANT_HOME** that points to the Ant installation folder, in this case **c:\apache-ant-1.8.2-bin** folder.
-
- Append the path to the Apache Ant batch file to the PATH environment variable. In our case this would be the **c:\apache-ant-1.8.2-bin\bin** folder.

Verifying Apache Ant Installation

To verify the successful installation of Apache Ant on your computer, type ant on your command prompt.

You should see an output similar to −

```
C:\>ant -version
Apache Ant(TM) version 1.8.2 compiled on December 20 2010
```

If you do not see the above output, then please verify that you have followed the installation steps properly.

Using Build.xml

Typically, Ant's build file, called **build.xml** should reside in the base directory of the project. However there is no restriction on the file name or its location. You are free to use other file names or save the build file in some other location.

For this exercise, create a file called build.xml anywhere in your computer with the following contents in it −

```
<?xml version = "1.0"?>
<project name = "Hello World Project" default = "info">
  <target name = "info">
    <echo>Hello World - Welcome to Apache Ant!</echo>
  </target>
</project>
```

Note that there should be no blank line(s) or whitespace(s) before the xml declaration. If you allow them, the following error message occurs while executing the ant build -

The processing instruction target matching "[xX][mM][lL]" is not allowed.

All build files require the **project** element and at least one **target** element.

The XML element **project** has three attributes −

Attributes & Description:

name

The Name of the project. (Optional)

default

The default target for the build script. A project may contain any number of targets. This attribute specifies which target should be considered as the default. (Mandatory)

Basedir

The base directory (or) the root folder for the project. (Optional)

A target is a collection of tasks that you want to run as one unit. In our example, we have a simple target to provide an informational message to the user.

Targets can have dependencies on other targets. For example, a **deploy** target may have a dependency on the **package** target, the **package** target may have a dependency on the **compile** target and so forth. Dependencies are denoted using the **depends** attribute. For example –

```
<target name = "deploy" depends = "package">
  ....
</target>

<target name = "package" depends = "clean,compile">
  ....
</target>

<target name = "clean" >
  ....
</target>

<target name = "compile" >
  ....
</target>
```

The target element has the following attributes –

Attributes & Description

name

The name of the target (Required)

depends

Comma separated list of all targets that this target depends on. (Optional)

if

Allows the execution of a target based on the trueness of a conditional attribute. (optional)

unless

Adds the target to the dependency list of the specified Extension Point. An Extension Point is similar to a target, but it does not have any tasks. (Optional)

The **echo** task in the above example is a trivial task that prints a message. In our example, it prints the message *Hello World*.

To run the ant build file, open up command prompt and navigate to the folder where the build.xml resides, and type **ant info**. You could also type **ant** instead. Both will work, because **info** is the default target in the build file. You should see the following output −

```
C:\>ant
Buildfile: C:\build.xml

info: [echo] Hello World - Welcome to Apache Ant!

BUILD SUCCESSFUL
Total time: 0 seconds

C:\>
```

Ant build files are written in XML, which does not allow declaring variables as you do in your favorite programming language. However, as you may have imagined, it would be useful if Ant allowed declaring variables such as project name, project source directory, etc.

Ant uses the **property** element which allows you to specify properties. This allows the properties to be changed from one build to another or from one environment to another.

By default, Ant provides the following pre-defined properties that can be used in the build file −

Sr.No.	Properties & Description
1	**ant.file** The full location of the build file.
2	**ant.version** The version of the Apache Ant installation.
3	**Basedir** The basedir of the build, as specified in the **basedir** attribute of the **project**element.
4	**ant.java.version** The version of the JDK that is used by Ant.
5	**ant.project.name** The name of the project, as specified in the **name** atrribute of the **project**element.
6	**ant.project.default-target** The default target of the current project.
7	**ant.project.invoked-targets** Comma separated list of the targets that were invoked in the current project.
8	**ant.core.lib** The full location of the Ant jar file.
9	**ant.home**

	The home directory of Ant installation.
10	**ant.library.dir** The home directory for Ant library files - typically ANT_HOME/lib folder.

Ant also makes the system properties (Example: file.separator) available to the build file.

In addition to the above, the user can define additional properties using the **property**element. The following example shows how to define a property called **sitename** −

```
<?xml version = "1.0"?>

<project name = "Hello World Project" default = "info">

   <property name = "sitename" value = "www.tutorialspoint.com"/>


   <target name = "info">

      <echo>Apache Ant version is ${ant.version} - You are at ${sitename} </echo>

   </target>

</project>
```

Running Ant on the above build file produces the following output −

```
C:\>ant
Buildfile: C:\build.xml

info: [echo] Apache Ant version is Apache Ant(TM) version 1.8.2
      compiled on December 20 2010 - You are at www.tutorialspoint.com

BUILD SUCCESSFUL
Total time: 0 seconds
C:\>
```

Stages in ANT build process:

CLEAN:

In the clean stage ANT will delete the pervious builds.

INIT: in the init stage ANT will recreate the folder structure

COMPILE: in compile stage .java files are converted into .class files

This .class files are also called as Byte code. They can be execute any o/s

Build:

In this stage .class files archive in to a single jar file or war file

Integrating ANT with Jenkins:

I have already setup Jenkins in the server. Now, it is time to install Ant. You can install ant manually and integrate it with Jenkins or you can download and install Jenkins through Jenkins. I am downloading Ant manually and integrating it with Jenkins later.

Download and Setup Ant

```
1 sudo su
2 apt-get install zip
3 # Download apache ant
4 wget http://www-us.apache.org/dist//ant/binaries/apache-ant-1.9.7-bin.zip
5 unzip apache-ant-1.9.7-bin.zip
6 mv apache-ant-1.9.7-bin.zip /opt/ant
7
```

my Ant directory is **/opt/ant** and Ant executable is located at **/opt/ant/bin**. I need to add some lines in **.bashrc** file.

Open your **.bashrc** file and add the following lines.

```
1 export ANT_HOME=/opt/ant
2 export PATH=$PATH:$ANT_HOME/bin/
```

Now, logout and again login to your root shell to use the exported variables.

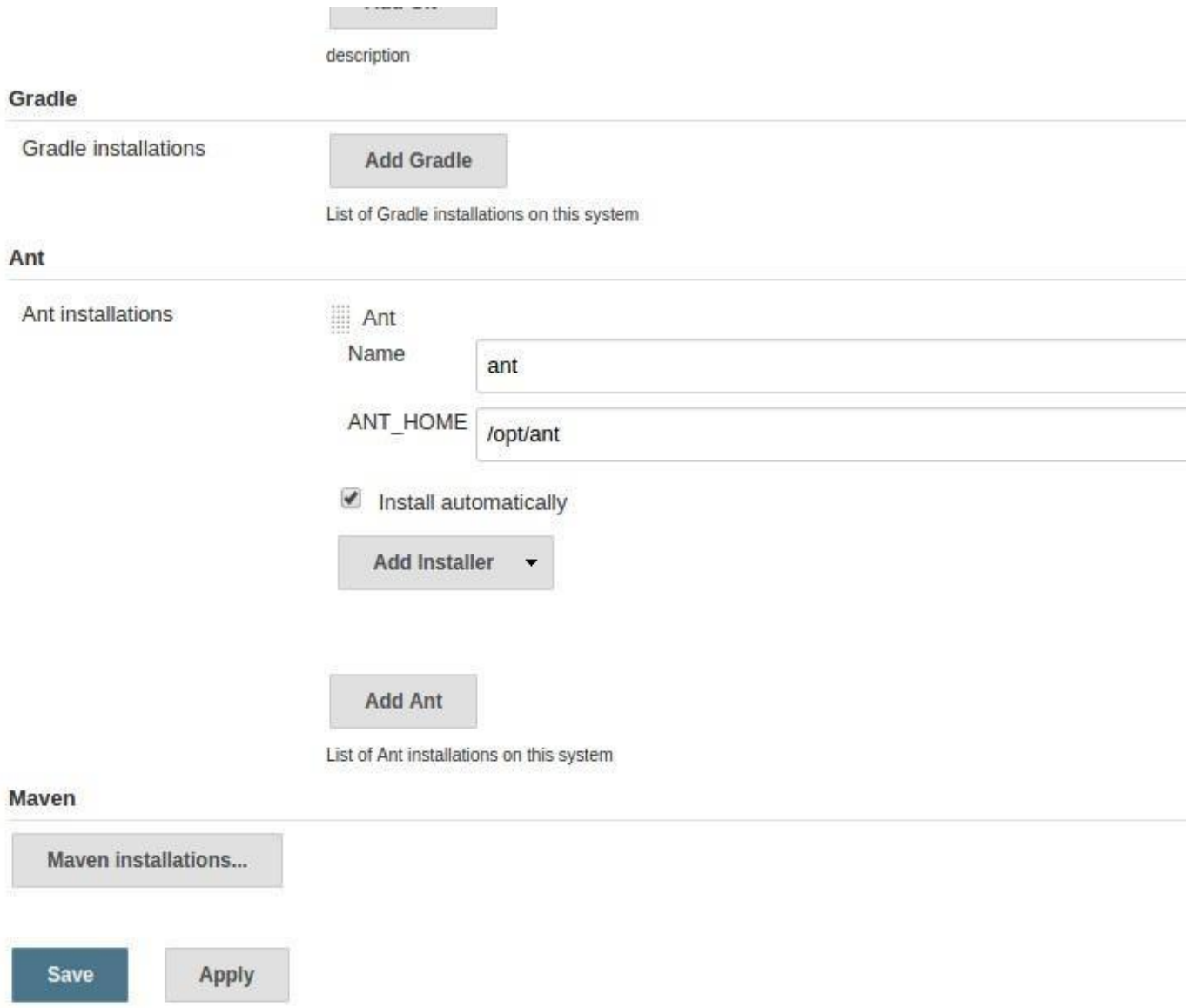
Now, Ant is setup in the server. We need to integrate with Jenkins.

Setup Ant in Jenkins

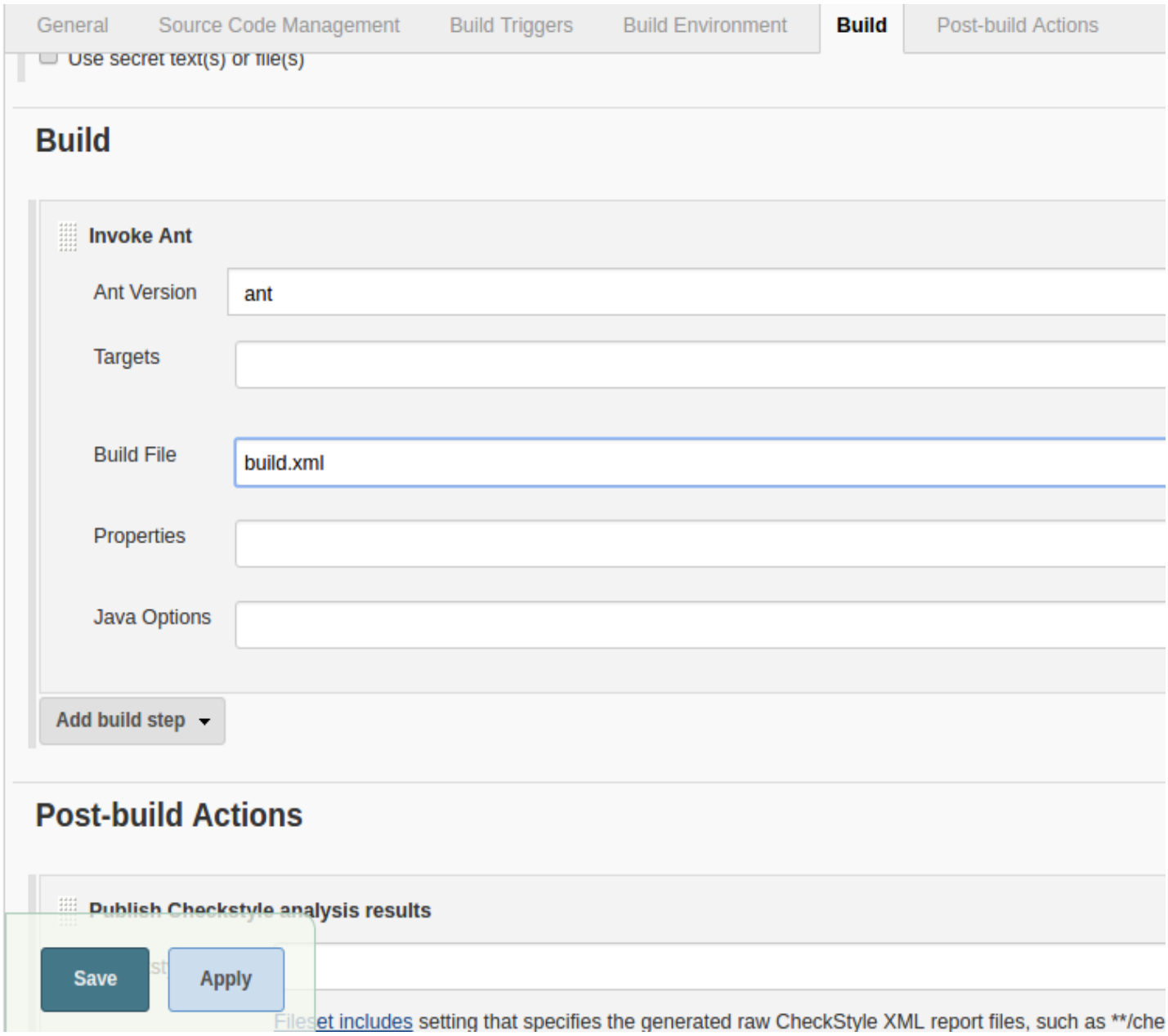
Login to your admin panel in Jenkins. Go to **Manage Jenkins** -> **Global Tool Configuration**

In Global Tool Configuration select **Add Ant**. I have already installed Ant. So I just need to provide **Name** and **ANT_HOME**. My ANT_HOME is /opt/ant.

If you have not Installed Ant, then you can install Ant here. Just select install automatically and choose the appropriate **Add Installer** option whether you want to extract from zip or install from Apache.



Now, Ant is integrated with Jenkins. You can use Ant in the build process. While configuring a job, in the build option select **invoke Ant** and choose the Ant version. In my case, I have given the name ant so there are two options available **default** and **ant**. Select advance option and you can configure extra parameters. I have chosen the build file **build.xml**. So when ever Ant is executed it will read from **build.xml** file.



You can put your **build.xml** file along with git repository. So whenever you want to test some code you can read from the **build.xml** file and define different **build.xml** file for different repository.

SVN

Apache Subversion which is often abbreviated as SVN, is a software versioning and revision control system distributed under an open source license. SVN system that is needed to maintain the current and historical versions of files such as source code, web pages, and documentations.

Prerequisites

Before proceeding with this tutorial, you should have a basic understanding on simple terminologies like programming language, source code, documents, etc. Because using SVN to handle all levels of software projects in your organization, it will be good if you have a working knowledge of software development and software testing processes.

Version Control System

VCS is a software that helps software developers to work together and maintain a complete history of their work.

goals of a Version Control System.

- Allow developers to work simultaneously.
- Do not overwrite each other's changes.
- Maintain history of every version of everything.

Version Control Terminologies

- **Repository:** A repository is the heart of any version control system. It is the central place where developers store all their work. Repository not only stores files but also the history. Repository is accessed over a network, acting as a server and version control tool acting as a client. Clients can connect to the repository, and then they can store/retrieve their changes to/from repository. By storing changes, a client makes these changes available to other people and by retrieving changes, a client takes other people's changes as a working copy.
- **Trunk:** The trunk is a directory where all the main development happens and is usually checked out by developers to work on the project.

- **Tags** : The tags directory is used to store named snapshots of the project. Tag operation allows to give descriptive and memorable names to specific version in the repository.

  For example, LAST_STABLE_CODE_BEFORE_EMAIL_SUPPORT is more memorable than

  Repository UUID: 7ceef8cb-3799-40dd-a067-c216ec2e5247 and

  Revision: 13

- **Branches:** Branch operation is used to create another line of development. It is useful when you want your development process to fork off into two different directions. For example, when you release version 5.0, you might want to create a branch so that development of 6.0 features can be kept separate from 5.0 bug-fixes.
- **Working copy:** Working copy is a snapshot of the repository. The repository is shared by all the teams, but people do not modify it directly. Instead each developer checks out the working copy. The working copy is a private workplace where developers can do their work remaining isolated from the rest of the team.
- **Commit changes:** Commit is a process of storing changes from private workplace to central server. After commit, changes are made available to all the team. Other developers can retrieve these changes by updating their working copy. Commit is an atomic operation. Either the whole commit succeeds or is rolled back. Users never see half finished commit.

SVN Installation and Configuration

Step 1: Install Apache2

Subversion server needs an web or HTTP server… For this setup, we're going to be using Apache2.. On Ubuntu, the commands below will install Apache2…

```
sudo apt update

sudo apt install apache2 apache2-utils
```

After installing Apache2, the commands below can be used to stop, start and enable Apache2 service to always start up with the server boots.

```
sudo systemctl stop apache2.service

sudo systemctl start apache2.service

sudo systemctl enable apache2.service
```

After installing Apache2 continue below to setting up Subversion…

Step 2: Install SVN Packages on Ubuntu

To get Subversion setup and working on Ubuntu, run the commands below to get it including all dependencies:

```
sudo apt-get install subversion libapache2-mod-svn subversion-tools libsvn-dev
```

After installing the above packages, run the commands below to enable Apache2 modules that allows Subversion to function….

```
sudo a2enmod dav

sudo a2enmod dav_svn

sudo service apache2 restart
```

Step 3: Configure Apache2

Now that Subversion packages are installed, run the commands below to edit SVN config file… This should allow you to create an SVN repository for controlling access… Run the commands below to open the file……

```
sudo vim /etc/apache2/mods-enabled/dav_svn.conf
```

Then make the hightlighted changes into the file, then save…

```
# ...

# URL controls how the repository appears to the outside world.

# In this example clients access the repository as http://hostname/svn/

# Note, a literal /svn should NOT exist in your document root.
```

th

```
<Location /svn>

 # Uncomment this to enable the repository

 DAV svn



 # Set this to the path to your repository

 #SVNPath /var/lib/svn

 # Alternatively, use SVNParentPath if you have multiple repositories under

 # under a single directory (/var/lib/svn/repo1, /var/lib/svn/repo2, ...).

 # You need either SVNPath and SVNParentPath, but not both.

 SVNParentPath /var/lib/svn



 # Access control is done at 3 levels: (1) Apache authentication, via

 # any of several methods. A "Basic Auth" section is commented out

 # below. (2) Apache and , also commented out

 # below. (3) mod_authz_svn is a svn-specific authorization module

 # which offers fine-grained read/write access control for paths

 # within a repository. (The first two layers are coarse-grained; you

 # can only enable/disable access to an entire repository.) Note that

 # mod_authz_svn is noticeably slower than the other two layers, so if
```

```
# you don't need the fine-grained control, don't configure it.


# Basic Authentication is repository-wide. It is not secure unless

# you are using https. See the 'htpasswd' command to create and

# manage the password file - and the documentation for the

# 'auth_basic' and 'authn_file' modules, which you will need for this

# (enable them with 'a2enmod').

AuthType Basic

AuthName "Subversion Repository"

AuthUserFile /etc/apache2/dav_svn.passwd


# To enable authorization via mod_authz_svn (enable that module separately):

 #

#AuthzSVNAccessFile /etc/apache2/dav_svn.authz

#


# The following three lines allow anonymous read, but make

# committers authenticate themselves. It requires the 'authz_user'

# module (enable it with 'a2enmod').
```

```
#

  Require valid-user

#

 </Location>
```

When you're done, run the commands below to create a SVN Repository in the **/var/lib/svn** directory….

```
sudo mkdir /var/lib/svn

sudo svnadmin create /var/lib/svn/repository

sudo chown -R www-data:www-data /var/lib/svn

sudo chmod -R 775 /var/lib/svn
```

Step 4: Create SVN User Accounts

Now that your SVN repository is created, run the commands below to create an account name **admin**

```
sudo htpasswd -cm /etc/apache2/dav_svn.passwd admin
```

Type a password and continue to complete the setup….

Repeat the step above to create additional users…

Restart Apache2

```
sudo systemctl restart apache2.service
```

When you're done, open your browser and browse to the reposotory, then logon…

http://server-public-ip/svn/repository

SVN Lifecycle

Create Repository:

The repository is a central place where developers store all their work. Repository not only stores files, but also the history about changes.Which means it maintains a history of the changes made in the files. The 'create' operation is used to create a new repository.Most of the times this operation is done only once. When you create a new repository, your VCS will expect you to say something to identify it, such as where you want it to be created, or what name should be given to the repository.

Checkout

'Checkout' operation is used to create a working copy from the repository. Working copy is a private workplace where developers do their changes, and later on, submit these changes to the repository.

Update

As the name suggests, 'update' operation is used to update working copy. This operation synchronizes the working copy with the repository. As repository is shared by all the teams other developers can commit their

Perform Changes

After the checkout, one can do various operations to perform changes. Edit is the most common operation. One can edit the existing file to add/remove contents from the file.

One can add files/directories. But immediately these files/directories do not become a part of the repository, instead they are added to the pending change-list and become a part of the repository after the commit operation.

Similarly one can delete files/directories. Delete operation immediately deletes file from the working copy, but actual deletion of the file is added to the pending change-list and changes are made to the repository after the commit operation.

'Rename' operation changes the name of the file/directory. 'Move' operation is used to move files/directories from one place to another in a repository tree.

Review Changes

When you check out the working copy or update the working copy, then your working copy is completely synchronized with the repository. But as you do changes to your working copy, it becomes newer than the repository. And it is a good practice to review your changes before the 'commit' operation.

'Status' operation lists the modifications that have been made to the working copy. As we have mentioned before, whenever you do changes in the working copy all these changes become a part of the pending change-list. And the 'status' operation is used to see the pending change-list.

'Status' operation only provides a list of changes but not the details about them. One can use *diff* operation to view the details of the modifications that have been made to the working copy.

Fix Mistakes

Let us suppose one has made changes to his working copy, but now, he wants to throw away these changes. In this situation, 'revert' operation will help.

Revert operation reverts the modifications that have been made to the working copy. It is possible to revert one or more files/directories. Also it is possible to revert the whole working copy. In this case, the 'revert' operation will destroy the pending change-list and will bring the working copy back to its original state.

Resolve Conflicts:

Conflicts can occur at the time of merging. 'Merge' operation automatically handles everything that can be done safely. Everything else is considered as conflict. For example, *"hello.c"* file was modified in branch and deleted in another branch. Such a situation requires a person to make the decision. The 'resolve' operation is used to help the user figure out things and to inform VCS about the ways of handling the conflicts.

Commit Changes

'Commit' operation is used to apply changes from the working copy to the repository. This operation modifies the repository and other developers can see these changes by updating their working copy.

Before commit, one has to add files/directories to the pending change-list. This is the place where changes wait to be committed. With commit, we usually provide a log message to explain why someone made changes. This log message becomes a part of the history of the repository. Commit is an atomic operation, which means either the entire commit succeeds or it is rolled back. Users never see half-finished commit.

Checkout process

Subversion provides the *checkout* command to check out a working copy from a repository. Below command will create a new directory in the current working directory with the name *project_repo*. Don't bother about the repository URL, as most of the time, it is already provided by the subversion administrator with appropriate access.

```
svn checkout http://svn-server-ip/svn/project_repo --username=tom
```

```
Authentication realm: Subversion Repository
Password for 'admin': *****


-----------------------------------------
ATTENTION! Your password for authentication realm:
```

```
Subversion Repository

can only be stored to disk unencrypted! You are advised to configure
your system so that Subversion can store passwords encrypted, if
possible. See the documentation for details.

You can avoid future appearances of this warning by setting the value
of the 'store-plaintext-passwords' option to either 'yes' or 'no' in
'/home/ubuntu/.subversion/servers'.
-----------------------------------------------------------------------
Store password unencrypted (yes/no)? yes
Checked out revision 0.
```

SVN perform changes:

admin checks out the latest version of the repository and starts working on a project. He creates *array.c* file inside the trunk directory.

```
cd project_repo/trunk/

cat array.c
```

The above command will produce the following result.

```
#include <stdio.h>

#define MAX 16


int main(void) {

   int i, n, arr[MAX];

   printf("Enter the total number of elements: ");

   scanf("%d", &n);


   printf("Enter the elements\n");
```

```
  for (i = 0; i < n; ++i) scanf("%d", &arr[i]);

  printf("Array has following elements\n");

  for (i = 0; i < n; ++i) printf("|%d| ", arr[i]);


  printf("\n");

  return 0;

}
```

He wants to test his code before commit.

```
 make array
cc     array.c -o array

 ./array
Enter the total number of elements: 5
Enter the elements
1
2
3
4
5
Array has following elements
|1| |2| |3| |4| |5|
```

He compiled and tested his code and everything is working as expected, now it is time to commit changes.

```
 svn status
?       array.c
?       array
```

Subversion is showing *'?'* in front of filenames because it doesn't know what to do with these files.

Before commit, *admin* needs to add this file to the pending change-list.

```
 svn add array.c
A         array.c
```

Let us check it with the 'status' operation. Subversion shows **A** before *array.c*, it means, the file is successfully added to the pending change-list.

```
snv status
?       array
A       array.c
```

To store *array.c* file to the repository, use the commit command with -m option followed by commit message.

If you omit -m option Subversion will bring up the text editor where you can type a multi-line message.

```
svn commit –m “Intial commit””
Adding         trunk/array.c
Transmitting file data .
Committed revision 2.
```

Now *array.c* file is successfully added to the repository, and the revision number is incremented by one

Note: if you get any error it is not working copy then execute belo command

```
Note: if you got any error linke it is not a working copy the execute command in
svn/trk(sudo svn co --force http://svn-server-ip/svn/repository/ .)
```

SVN review changes:

admin already added *array.c* file to the repository. *Tom* also checks out the latest code and starts working.

```
svn co http://svn.server.com/svn/project_repo --username=tom
```

Above command will produce the following result.

```
A    project_repo/trunk
A    project_repo/trunk/array.c
A    project_repo/branches
A    project_repo/tags
Checked out revision 2.
```

But, he found that someone has already added the code. So he is curious about who did that and he checks the log message to see more details using the following command:

```
svn log
```

Above command will produce the following result.

```
------------------------------------------------------------------------
r2 | jerry | 2013-08-17 20:40:43 +0530 (Sat, 17 Aug 2013) | 1 line

Initial commit
------------------------------------------------------------------------
r1 | jerry | 2013-08-04 23:43:08 +0530 (Sun, 04 Aug 2013) | 1 line

Create trunk, branches, tags directory structure
------------------------------------------------------------------------
```

When *Tom* observes *admin's* code, he immediately notices a bug in that. admin was not checking for array overflow, which could cause serious problems. So Tom decides to fix this problem. After modification, *array.c* will look like this.

```
#include <stdio.h>


#define MAX 16


int main(void)
{
  int i, n, arr[MAX];


  printf("Enter the total number of elements: ");
  scanf("%d", &n);


  /* handle array overflow condition */
  if (n > MAX) {
```

```
    fprintf(stderr, "Number of elements must be less than %d\n", MAX);
    return 1;
  }

  printf("Enter the elements\n");

  for (i = 0; i < n; ++i)
    scanf("%d", &arr[i]);

  printf("Array has following elements\n");
  for (i = 0; i < n; ++i)
    printf("|%d| ", arr[i]);
    printf("\n");

  return 0;
}
```

Tom wants to use the status operation to see the pending change-list.

```
svn status
M       array.c
```

array.c file is modified, that's why Subversion shows **M** letter before file name. Next *Tom*compiles and tests his code and it is working fine. Before committing changes, he wants to double-check it by reviewing the changes that he made.

```
 svn diff
Index: array.c
```

```
===================================================================
--- array.c (revision 2)
+++ array.c (working copy)
@@ -9,6 +9,11 @@
   printf("Enter the total number of elements: ");
   scanf("%d", &n);

+ if (n > MAX) {
+     fprintf(stderr, "Number of elements must be less than %d\n", MAX);
+     return 1;
+ }
+
   printf("Enter the elements\n");

   for (i = 0; i < n; ++i)
```

Tom has added a few lines in the *array.c* file, that's why Subversion shows + sign before new lines. Now he is ready to commit his changes.

```
svn commit -m "Fix array overflow problem"
```

The above command will produce the following result.

```
Sending       trunk/array.c
Transmitting file data .
Committed revision 3.
```

Tom's changes are successfull y committed to the repository.

Svn update process

admin had committed the first version of the code. But he thinks that he should write two functions to accept input and to display array contents. After modification, *array.c* looks as follows.

```
#include <stdio.h>
#define MAX 16

void accept_input(int *arr, int n){
    int i;
    for (i = 0; i < n; ++i)
    scanf("%d", &arr[i]);
```

```
}

void display(int *arr, int n){
    int i;
    for (i = 0; i < n; ++i)
    printf("|%d| ", arr[i]);

    printf("\n");
}

int  main(void){
    int i, n,  arr[MAX];

    printf("Enter the total number of elements: ");
    scanf("%d", &n);

    printf("Enter the elements\n");
    accept_input(arr, n);

    printf("Array has following elements\n");
    display(arr, n);

    return 0;
}
```

admin compiles and tests his code and is ready to commit changes. Before that, he wants to review the changes using the following command.

```
svn diff
```

The above command will produce the following result.

```
Index: array.c
===================================================================
--- array.c    (revision 2)
+++ array.c    (working copy)
@@ -2,6 +2,24 @@

 #define MAX 16

+void accept_input(int *arr, int n)
+{
+    int i;
+
+    for (i = 0; i & n; ++i)
+        scanf("%d", &arr[i]);
+}
+
```

```
+void display(int *arr, int n)
+{
+    int i;
+
+    for (i = 0; i < n; ++i)
+        printf("|%d| ", arr[i]);
+
+    printf("\n");
+}
+
 int main(void)
 {
     int i, n, arr[MAX];
@@ -10,15 +28,10 @@
     scanf("%d", &n);

     printf("Enter the elements\n");
+    accept_input(arr,  n);

-    for (i = 0; i < n; ++i)
-        scanf("%d", &arr[i]);
-
     printf("Array has following elements\n");
-    for (i = 0; i < n; ++i)
-        printf("|%d| ", arr[i]);
-
-    printf("\n");
+    display(arr, n);

     return 0;
 }
```

For the new added lines, Subversion shows + sign before line and for removed line it shows - sign. Now, Jerry tries to commit the changes using the following command:

```
[jerry@CentOS trunk]$ svn commit -m "Add function to accept input and to display array
contents"
```

The above command will produce the following result.

```
Sending        trunk/array.c
svn: Commit failed (details follow):
svn: File or directory 'array.c' is out of date; try
updating svn: resource out of date; try updating
```

Subversion is not allowing to commit *admin's* changes, because *Tom* has already modified the repository and *Jerry's* working copy is out of date. To avoid overwriting each other's changes, Subversion fails this

operation. *admin* must update working copy before committing his changes. So he uses update command as shown below.

```
 svn update
G     array.c
Updated to revision 3.
```

Subversion is showing the letter ***G*** before filename, which means this file has been merged.

```
 svn diff
```

The above command will produce the following result.

```
Index: array.c
===================================================================
--- array.c    (revision 3)
+++  array.c     (working copy)
@@ -2,6 +2,24 @@

 #define MAX 16

+void accept_input(int *arr, int n)
+{
+    int i;
+
+    for (i = 0; i < n; ++i)
+        scanf("%d", &arr[i]);
+}
+
+void display(int *arr, int n)
+{
+    int i;
+
+    for (i = 0; i < n; ++i)
+        printf("|%d| ",arr[i]);
+
+    printf("\n");
+}

+
 int main(void)
 {
     int i, n, arr[MAX];
@@ -15,15 +33,10 @@
     }

     printf("Enter the elements\n");
```

```
+    accept_input(arr, n);

-    for (i = 0; i < n; ++i)
-        scanf("%d", &arr[i]);
-
     printf("Array has following elements\n");
-    for (i = 0; i < n; ++i)
-        printf("|%d| ", arr[i]);
-
-    printf("\n");
+    display(arr, n);

     return 0;
 }
```

Subversion is showing only *admin's* changes, but *array.c* file is merged. If you observe carefully, Subversion is now showing revision number 3. In the previous output, it was showing revision number 2. Just review who made changes in the file and for what purpose.

```
svn log
------------------------------------------------------------------------
r3 | admin | 2013-08-18 20:21:50 +0530 (Sun, 18 Aug 2013) | 1 line

Fix array overflow problem
------------------------------------------------------------------------
r2 | jerry | 2013-08-17 20:40:43 +0530 (Sat, 17 Aug 2013) | 1 line

Initial commit
------------------------------------------------------------------------
r1 | jerry | 2013-08-04 23:43:08 +0530 (Sun, 04 Aug 2013) | 1 line

Create trunk, branches, tags directory structure
------------------------------------------------------------------------
```

Now *admin's* working copy is synchronized with the repository and he can safely commit his changes.

```
svn commit -m "Add function to accept input and to display array contents"
Sending        trunk/array.c
Transmitting file data .
Committed revision 4.
```

SVN Fix mistakes

Suppose *admin* accidently modifies *array.c* file and he is getting compilation errors. Now he wants to throw away the changes. In this situation, 'revert' operation will help. Revert operation will undo any local changes to a file or directory and resolve any conflicted states.

```
svn status
```

Above command will produce the following result.

```
M       array.c
```

Let's try to make array as follows: make array

Above command will produce the following result.

```
cc     array.c -o array
array.c: In function ‘main’:
array.c:26: error: ‘n’ undeclared (first use in this function)
array.c:26: error: (Each undeclared identifier is reported only once
array.c:26: error: for each function it appears in.)
array.c:34: error: ‘arr’ undeclared (first use in this function)
make: *** [array] Error 1
```

admin performs 'revert' operation on *array.c* file.

```
svn revert array.c
Reverted 'array.c'

svn status
```

Now compile the code.

```
make array
cc     array.c -o array
```

After the revert operation, his working copy is back to its original state. Revert operation can revert a single file as well as a complete directory. To revert a directory, use -R option as shown below.

```
pwd
/home/jerry/project_repo

svn revert -R trunk
```

Till now, we have seen how to revert changes, which has been made to the working copy. But what if you want to revert a committed revision! Version Control System tool doesn't allow to delete history from the repository. We can only append history. It will happen even if you delete files from the repository. To undo an old revision, we have to reverse whatever changes were made in the old revision and then commit a new revision. This is called a reverse merge.

Let us suppose Jerry adds a code for linear search operation. After verification he commits his changes.

```
 svn diff
Index: array.c
===================================================================
--- array.c (revision 21)
+++ array.c (working copy)
@@ -2,6 +2,16 @@

 #define MAX 16

+int linear_search(int *arr, int n, int key)
+{
+   int i;
+
+   for (i = 0; i < n; ++i)
+      if (arr[i] == key)
+         return i;
+   return -1;
+}
+
 void bubble_sort(int *arr, int n)
 {
    int i, j, temp, flag =  1;

[jerry@CentOS trunk]$ svn status
?       array
M       array.c

 svn commit -m "Added code for linear search"
Sending        trunk/array.c
Transmitting file data .
Committed revision 22.
```

Jerry is curious about what Tom is doing. So he checks the Subversion log messages.

```
svn log
```

The above command will produce the following result.

```
------------------------------------------------------------------------
r5 | tom | 2013-08-24 17:15:28 +0530 (Sat, 24 Aug 2013) | 1 line

Add binary search operation
------------------------------------------------------------------------
r4 | jerry | 2013-08-18 20:43:25 +0530 (Sun, 18 Aug 2013) | 1 line

Add function to accept input and to display array contents
```

After viewing the log messages, admin realizes that he did a serious mistake. Because Tom already implemented binary search operation, which is better than the linear search; his code is redundant, and now Jerry has to revert his changes to the previous revision. So, first find the current revision of the repository. Currently, the repository is at revision 22 and we have to revert it to the previous revision, i.e. revision 21.

```
svn up
At revision 22.

svn merge -r 22:21 array.c
--- Reverse-merging r22 into 'array.c':
U    array.c

svn commit -m "Reverted to revision 21"
Sending        trunk/array.c
Transmitting file data .
Committed revision 23.
```

SVN Branching

Branch operation creates another line of development. It is useful when someone wants the development process to fork off into two different directions. Let us suppose you have released a product of version 1.0, you might want to create new branch so that development of 2.0 can be kept separate from 1.0 bug fixes.

In this section, we will see how to create, traverse and merge branch. admin is not happy because of the conflict, so he decides to create a new private branch.

```
 ls
branches tags  trunk

 svn copy trunk branches/jerry_branch
A         branches/jerry_branch

 svn status
A  +    branches/jerry_branch

[jerry@CentOS project_repo]$ svn commit -m "Jerry's private branch"
Adding         branches/jerry_branch
Adding         branches/jerry_branch/README

Committed revision 9.
```

Now admin is working in his private branch. He adds sort operation for the array. Jerry's modified code looks like this.

```
 cd branches/jerry_branch/

 cat array.c
```

The above command will produce the following result.

```
#include <stdio.h>

#define MAX 16



void bubble_sort(int *arr, int n)

{

   int i, j, temp, flag =  1;

   for (i = 1; i < n && flag == 1; ++i) {
```

```
    flag = 0;
    for (j = 0; j < n - i; ++j) {
      if (arr[j] > arr[j + 1]) {
        flag = 1;
        temp = arr[j];
        arr[j] = arr[j + 1];
        arr[j + 1] = temp;
      }
    }
  }
}

void accept_input(int *arr, int n)
{
  int i;

  for (i = 0; i < n; ++i)
  scanf("%d", &arr[i]);
}

void display(int *arr, int n)
{
  int i;
```

```
   for (i = 0; i < n; ++i)
   printf("|%d| ", arr[i]);

   printf("\n");
}

int main(void)
{
   int i, n, key, ret, arr[MAX];

   printf("Enter the total number of elements: ");
   scanf("%d", &n);

   /* Error handling for array overflow */
   if (n >MAX) {
      fprintf(stderr, "Number of elements must be less than %d\n", MAX);
      return 1;
   }

   printf("Enter the elements\n");
   accept_input(arr, n);
```

```
   printf("Array has following elements\n");

   display(arr, n);


   printf("Sorted data is\n");

   bubble_sort(arr, n);

   display(arr, n);


   return 0;

}
```

admin compiles and tests his code and is ready to commit his changes.

```
make array
cc     array.c -o array

 ./array
```

The above command will produce the following result.

```
Enter the total number of elements: 5
Enter the elements
10
-4
2
7
9
Array has following elements
|10| |-4| |2| |7| |9|
Sorted data is
|-4| |2| |7| |9| |10|

[jerry@CentOS jerry_branch]$ svn status
?       array
M       array.c
```

```
 svn commit -m "Added sort operation"
Sending        jerry_branch/array.c
Transmitting file data .
Committed revision 10.
```

Meanwhile, over in the trunk, Tom decides to implement search operation. Tom adds code for search operation and his code looks like this.

```
svn diff
```

The above command will produce the following result.

```
Index: array.c
===================================================================
--- array.c (revision 10)
+++ array.c (working copy)
@@ -2,6 +2,27 @@

 #define MAX 16

+int bin_search(int *arr, int n, int key)
+{
+   int low, high, mid;
+
+   low = 0;
+   high = n - 1;
+   mid = low + (high - low) / 2;
+
+   while (low <= high) {
+      if (arr[mid] == key)
+         return mid;
+      if (arr[mid] > key)
+         high = mid - 1;
+      else
+         low = mid + 1;
+      mid = low + (high - low) / 2;
+   }
+
+   return -1;
+}
+
 void accept_input(int *arr, int n)
```

```
 {
   int i;
@@ -22,7 +43,7 @@

 int main(void)
 {
- int i, n, arr[MAX];
+ int i, n, ret, key, arr[MAX];

   printf("Enter the total number of elements: ");
   scanf("%d", &n);
@@ -39,5 +60,16 @@
   printf("Array has following elements\n");
   display(arr, n);

+   printf("Enter the element to be searched: ");
+   scanf("%d", &key);
+
+   ret = bin_search(arr, n, key);
+   if (ret < 0) {
+      fprintf(stderr, "%d element not present in array\n", key);
+      return 1;
+   }
+
+   printf("%d element found at location %d\n", key, ret + 1);
+
   return 0;
 }
```

After reviewing, he commits his changes.

```
[tom@CentOS trunk]$ svn status
?       array
M       array.c

[tom@CentOS trunk]$ svn commit -m "Added search operation"
Sending        trunk/array.c
Transmitting file data .
Committed revision 11.
```

But Tom is curious about what admin has been doing in his private branch.

```
[tom@CentOS trunk]$ cd ../branches/
[tom@CentOS branches]$ svn up
```

```
A    jerry_branch
A    jerry_branch/array.c
A    jerry_branch/README

[tom@CentOS branches]$ svn log
------------------------------------------------------------------------
r9 | jerry | 2013-08-27 21:56:51 +0530 (Tue, 27 Aug 2013) | 1 line

Added sort operation
------------------------------------------------------------------------
```

By viewing the Subversion's log message, Tom found that Jerry implemented 'sort' operation. Tom implemented search operation using binary search algorithm, it always expects data in sorted order. But what if the user provides data in an unsorted order? In that situation, binary search operation will fail. So he decides to take Jerry's code to sort data before search operation. So he asks Subversion to merge code from admin's branch into trunk.

```
[tom@CentOS trunk]$ pwd
/home/tom/project_repo/trunk

 svn merge ../branches/jerry_branch/
--- Merging r9 through r11 into '.':
U    array.c
```

After merging, array.c will look like this.

```
cat array.c
```

The above command will produce the following result.

```
#include <stdio.h>

#define MAX 16



void bubble_sort(int *arr, int n)

{

   int i, j, temp, flag =  1;
```

```
  for (i = 1; i < n && flag == 1; ++i) {
    flag = 0;
    for (j = 0; j < n - i; ++j) {
      if (arr[j] > arr[j + 1]) {
        flag        = 1;
        temp        = arr[j];
        arr[j]      = arr[j + 1];
        arr[j + 1]  = temp;
      }
    }
  }
}

int bin_search(int *arr, int n, int key)
{
  int low, high, mid;

  low = 0;
  high = n - 1;
  mid = low + (high - low) / 2;

  while (low <= high) {
```

```
    if (arr[mid] == key)
      return mid;
    if (arr[mid] > key)
      high = mid - 1;
    else
      low = mid + 1;
    mid = low + (high - low) / 2;
  }
  return -1;
}

void accept_input(int *arr, int n)
{
  int i;

  for (i = 0; i < n; ++i)
    scanf("%d", &arr[i]);
}

void display(int *arr, int n)
{
  int i;
  for (i = 0; i < n; ++i)
```

```
    printf("|%d| ", arr[i]);
  printf("\n");
}

int main(void)
{
  int i, n, ret, key, arr[MAX];

  printf("Enter the total number of elements: ");
  scanf("%d", &n);

  /* Error handling for array overflow */
  if (n > MAX) {
    fprintf(stderr, "Number of elements must be less than %d\n", MAX);
    return 1;
  }

  printf("Enter the elements\n");
  accept_input(arr, n);

  printf("Array has following elements\n");
  display(arr, n);
```

```
   printf("Sorted data is\n");

   bubble_sort(arr, n);

   display(arr, n);


   printf("Enter the element to be searched: ");

   scanf("%d", &key);


   ret = bin_search(arr, n, key);

   if (ret < 0) {

      fprintf(stderr, "%d element not present in array\n", key);

      return 1;

   }


   printf("%d element found at location %d\n", key, ret + 1);


   return 0;

}
```

After compilation and testing, Tom commits his changes to the repository.

```
 make array
cc     array.c -o array

 ./array
Enter the total number of elements: 5
Enter the elements
10
-2
```

```
8
15
3
Array has following elements
|10| |-2| |8| |15| |3|
Sorted data is
|-2| |3| |8| |10| |15|
Enter the element to be searched: -2
-2 element found at location 1

 svn commit -m "Merge changes from Jerry's code"
Sending        trunk
Sending        trunk/array.c
Transmitting file data .
Committed revision 12.
```

Continuous Integration (Jenkins):

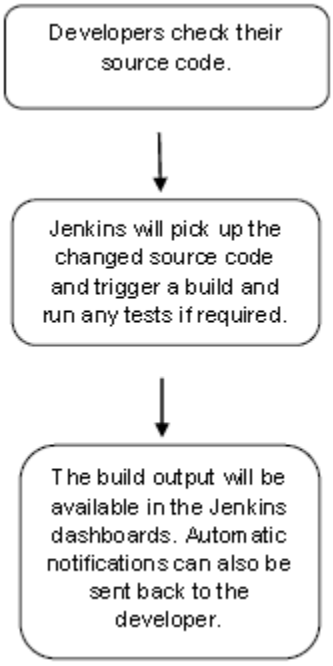
Jenkins is a self-contained, open source automation server which can be used to automate all sorts of tasks related to building, testing, and delivering or deploying software.

Jenkins can be installed through native system packages, Docker, or even run standalone by any machine with a Java Runtime Environment (JRE) installed

Introduction

Why Jenkins?

Jenkins is a software that allows **continuous integration**. Jenkins will be installed on a server where the central build will take place. The following flowchart demonstrates a very simple workflow of how Jenkins works.



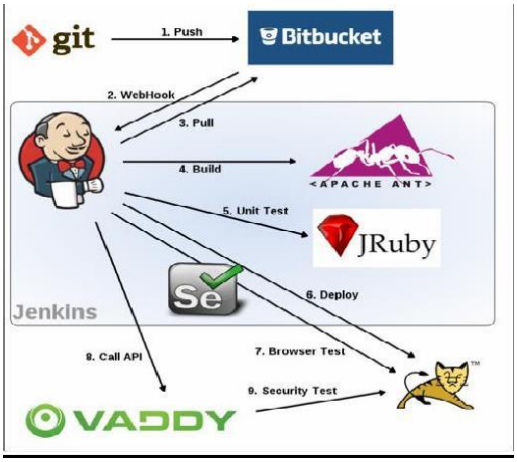
Along with Jenkins, sometimes, one might also see the association of **Hudson**. Hudson is a very popular open-source Java-based continuous integration tool developed by Sun Microsystems which was later acquired by Oracle. After the acquisition of Sun by Oracle, a fork was created from the Hudson source code, which brought about the introduction of Jenkins.

What is 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 lifecycle. Continuous integration requires the developers to have frequent builds. The common practice is that whenever a code commit occurs, a build should be triggered.

System Requirements:

JDK	JDK 1.5 or above
Memory	2 GB RAM (recommended)
Disk Space	No minimum requirement. Note that since all builds will be stored on the Jenkins machines, it has to be ensured that sufficient disk space is available for build storage.
Operating System Version	Jenkins can be installed on Windows, Ubuntu/Debian, Red Hat/Fedora/CentOS, Mac OS X, openSUSE, FReeBSD, OpenBSD, Gentoo.
Java Container	The WAR file can be run in any container that supports Servlet 2.4/JSP 2.0 or later.(An example is Tomcat 5).



CI-CD Stages

1. Stage 1 (Continuous Download)

In this stage Jenkins is integrated with the remote version controlling system(git) in susch a way that when ever developers make changes to the code it will detect those changes and download from the remote repository

2. Stage 2 (Continuous Build)

The code downloaded in the previous stage has to be converted into an artifact.This artifact can be in the form of a jar,war,ear,exe file.This process is called as the build process and Jenkins will perform this step with the help of plugins like ant,maven,ms build etc

3. Stage 3 (Continuous Deployment)

The artifact created in the previous stage has to e deployed into QA servers.The testing servers might be running on application servers like tomcat,jboss weblogic etc.The artifact has to deployed into these servers where testers can access and test the application

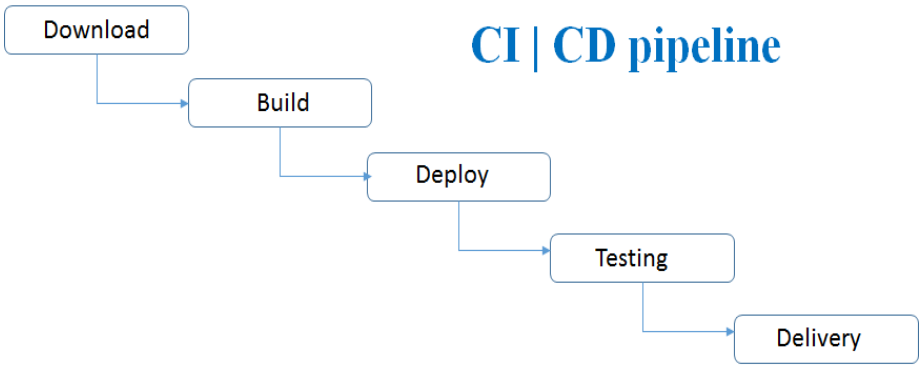
4. Stage 4 (Continuous Testing)

The testers will prepare some automation testing programs using tools like selenium, tosca,codedui etc.These testing programs will be uploaded by the testers into the git version controlling system.Jenkins should download these programs and run them on the application that was deployed into the testing servers.If testing fails jenkins will send automated notifiactions to the team memebers and developers should fix the defects and upload the modified code into the git repository.Jenkins will again trigger the above 4 steps

5. Stage 5 (Continuous Delivery)

If testing passes then jenkins will deploy the artifact into the production servers after taking approvals from the team members.Once it is deployed into the prod servers the enduser/client can start accessing it.

Note: The first four stages are called Continuous Integration and the fifth stage is called continuous delivery



Installing, Integration and Configuration of Jenkins on Windows

1 Download and install jdk 1.8 or jdk 1.9

2 Download jenkins from

https://jenkins.io/download/

Go to LTS and download the version related to windows

3 Install it

4To open jenkins

Launch any browser

localhost:8080

5 Unlock jenkins by entering the admin password

6 Click on Install suggested plugin

7 Create first admin user--->Continue

Setting the path of git in jenkins:

1 Install git from https://git-scm.com/downloads

2 Open c:-->programs files-->git-->bin

Copy the path of git.exe

3Open the dashboard of jenkins

localhost:8080

4 Click on Managejenkins

5 Click on Global tool configurations

6 Go to Git section

7 enter some name for git

Paste the path of git

Stage 1 (Continuous Download)

1 Open the dashboard of jenkins(localhost:8080)

2 Click on New item

3 Enter item name as "Development"

4 Click on Free style project

5 Click on Ok

6 Go to Source code management

7 Select Git

8Enter the github url where developers have uploaded the code

https://github.com/selenium-saikrishna/maven.git

9 Click on Apply--->save

10 Go to the dashboard of jenkins

11 Go to the Development job--->click on Build icon

The above job will download all the code uploaded by the developer into the github repository.

Setting the path of maven in jenkins:

1 Open https://maven.apache.org/download.cgi

2 Go to Files section on this page and download th bin.zip version.

3 Extract it and open it and copy its path

4 Open the dashboard of jenkins

5 Click on manage jenkins

6 Click on Global tool configurations

7 Go to Maven installations

8 Click on Add maven

9 Enter some anme for maven(mymaven)

10 Paste the path of Maven in MAVEN_HOME location

12 Apply--->save

Stage 2 (Continuous Build)

1 Open the dashboard of jenkins

2 Go to the development job--->click on configure

3 Go to Build section

4 Click on add build step

5 Click on Invoke top level maven targets

6 Select maven as mymaven

7 Enter goal as package

8 Click on Apply--->save

9 Go to dashboard of jenkins

10 Go to the development job--->click on Build icon

The above job will now create an artifact from the code that was downloaded in the previous stage.This artifact comes in the format of a war file.

Configuring tomcat7:

1 Open https://tomcat.apache.org/download-70.cgi

2 Go to binary distributions

3 Download the zip version

4 Extract it and open it

5 Since jenkins and tomcat are running on the same machine and both use the 8080 port we should change the port of either jenkins or tomcat

To change poert number for tomcat

Open the tomcat folder

Open conf-->Openserver.xml

Search for "Connector Port" and change it from

8080 to 8899

6Jenkins requires credentials of tomcat to deploy artifacts intotomcat

Open tomcatfolder

Open conf folder

Open tomcat-users.xml file

Add the below statement

<user username="itpoint" password="myitpoint" roles="manager-script"/>

7 To start tomcat

Open bin folder--->Click on startup.bat

8 To access the home page of tomcat

Launch any browser

localhost:8899

Stage -3 (Continuous Deployment)

1 Open the dashboard of jenkins(localhost:8080)

2 Click on Manage jenkins--->Manage plugins

3 Go to Available section

4 Search for "Deploy to Container" plugin--->Install it

5 Go back to the dashboard of jenkins

6 Go to the Development job

7 Click on Configure

8 Go to Post build actions

9 click on Add post build action

10 click on Deploy war/ear to container

Copying artifacts from Development to Testing jobs:

The artifacts created by Development job should be passed to Testing job so that the Testing job can deploy the artifact into the prod environment

1 Open the dashboard of jenkins

2 Click on manage jenkins

3 Click on manage plugins

4 Go to Available Section

5 Search for Copy artifact plugin

6 Install it

7 Open the dashboard of jenkins-->go to Development job

8 Click on Configure

9 Go to Post build action

10 Click on Add post build action

11 Click on archive the artifacts

12 Enter files to be archived as ***.war

13 Apply--->save

14 Go to Dashboard of jenkins--->Go to the Testing job

15 Click on Configure

16 Go to Build section

17 Click on Add build step

18 Click on Copy artifacts from another project

19 Enter project name as "development"

20 Apply--->save

Stage 4 (Continuous Testing)

1 Open the dasboard of jenkins

2 Click on New item--->enter item name as "Testing"

3 Select Free style project--->ok

4 Go to Source Code management

5 Click on git

6 Enter github url where testers have uploaded the selenium testing programs

https://github.com/selenium-saikrishna/FunctionalTesting.git

7 Go to Build section--->click on Add build step

8Click on Execute windows batch command

java -jar testing.jar

9 Click on Apply--->save

10 Go to dashboard of jenkins--->Run the Testing job

This job will download the selenium testing programs from github and run them on the application that was deployed in Stage no 3

Stage -5 (Continuous Delivery)

1 Open the dashboard of jenkins

2 Go to the Testing job--->click on Configure

3 Go to Post build actions

4 Click on Add post build action

5 Click on Deploy war/ear to container

Linking Development job with Testing job:

The Development job should be linked with the Testing job so that once the Development is over it will start the Testing

job.

1 Open the dashboard of jenkins

2 Go to the Development job--->click on Configure

3 Go to Post build actions

4 Click on Post buiild action

5 Click on Build other project

6 enter project name as Testing

7 Apply--->save

Install and Configure Jenkins on Linux

Setup Dev server:

1 Download and install orcaljava8

```
sudo add-apt-repository ppa:webupd8team/java
sudo apt-get update
sudo apt-get install oracle-java8-installer
```

2 Download jenkins.war file

```
wget http://mirrors.jenkins.io/war-stable /latest/jenkins.war
```

3 Install git and maven

```
sudo apt-get install -y git maven
```

4 To start jenkins

```
java -jar jenkins.war
```

Setup QA server and prodserver:

1 Login into the qaserver terminal

2Update the apt repository

3 Install tomcat7

sudo apt-get install -y tomcat7

4 Install tomcat7-admin

sudo apt-get install -y tomcat7-admin

5 Edit tomcat-user.xml file

cd /etc/tomcat7

sudo vim tomcat-user.xml

Go into insert mode by presing i

enter the below statement

<user username="admin" password="admin" roles="manager-script/>

Save and quit(Esc :wq)

6 Restart tomcat7

sudo service tomcat7 restart

Note: *Repeat the same navigations on the prodserver*

Setting the path of jdk, git and maven in Jenkins on Linx

1 Open the dashboard of jenkins

Launch any browser

ipaddress-of-linuxserver-where-jenkins-is-installed:8080

2 Click on manage jenkins

3 Click on Global tool configurations

4 Go to JDK--->click on jdk installations

5 Give some name to java(myjava)

6 Uncheck Install java automatically

7 Enter path of JAVA_HOME

To find this path

Open the linux terminal where jenkins is installed

which java

readlink -f oputput-of above command

Eg:

readlink -f /usr/bin/java

8Copy the path before bin folder and paste in Jenkins

JAVA_HOME location

9 Go to GIT

10 enter some name for git(mygit)

11 To find the path of git executable

Go to the linux terminal where jenkins is installed

which git

12 Copy the output paste in jenkins,Git location

13. readlink -f output-of-above-command

14 Go to MavenSection--->click on maven installations

15 Enter some name for maven(mymaven)

16 Uncheck install maven automatically

17 Enter MAVEN_HOME

To find this path

Go to the linux terminal where jenkins is installed

which mvn

readlink -f output-of-above-command

Eg: readlink -f /usr/bin/mvn

18 Copy the path before bin folder

19 Paste the path in MAVEN_HOME settings

20 click on apply--->save

Setup of Linux servers for Jenkins (Vagrant)

1 Download and install oracle virtual box

https://www.virtualbox.org/wiki/Downloads

2 Download and install vagrant

https://www.vagrantup.com/downloads.html

3 To check the version of vagrant

Open cmd prompt

vagrant --version

4 Copy the vagrantfile into an empty folder

5 Open cmd prompt

6change dir to the folder where vagrantfile is copied

cd path_of_folder_where_vagrantfile_is_copied

7 vagrant up

8 To access these linux servers use putty

Note1: *installation and configuration of Jenkins in above vagrant is same as previous Linux servers since it is also Linux server*

Note2: *ofter installing and configuring of Jenkins when the home page coming the functionalities of jenkins(ci-cd) is same in any operating system.*

Jenkins Setup on AWS

1 Signup for a aws account--->Login into that account

2 Click on Services--->click on EC2--->Click on Instances

3 Click on Launch instances--->Search for ubuntu14-->Select

4 Clcik on Next configuration instance details

5 enter no of instances as 3

6 click on add storage

7 click on add tags

8 click on configure seciurity group

9 click on Add rule-->Select Custom TCP

10 Enter port range as 8080

11 click on Review and Launch--->Launch

12 Click on Create a new keypair-->enter some name for the key pair

13 Click on Download keypair

14 click on Launch instances

15 click on view instances

To connect to these linux servers on the aws cloud

1 Open git bash

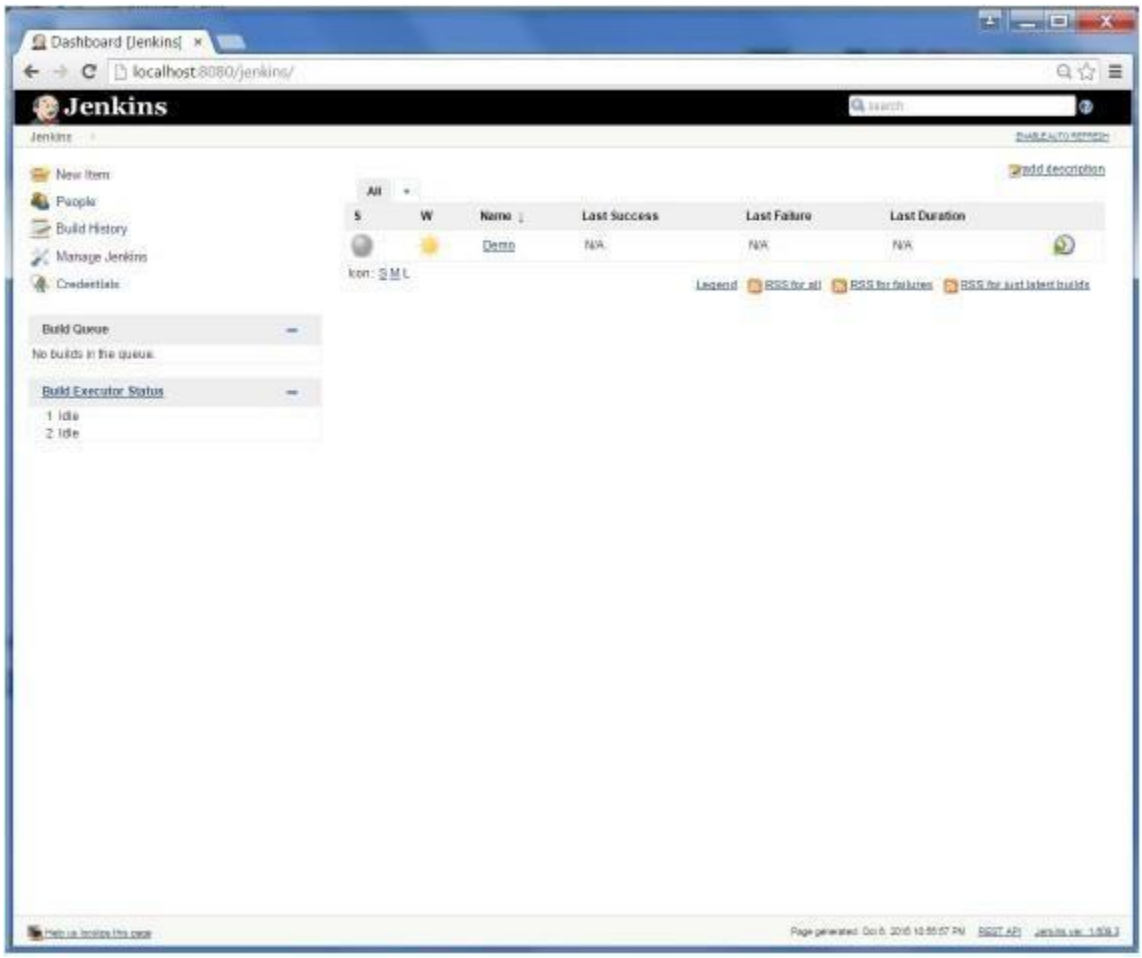
2 Click on the instance that we want to connect-->Connect

3 copy and paste the "ssh" command that aws generates

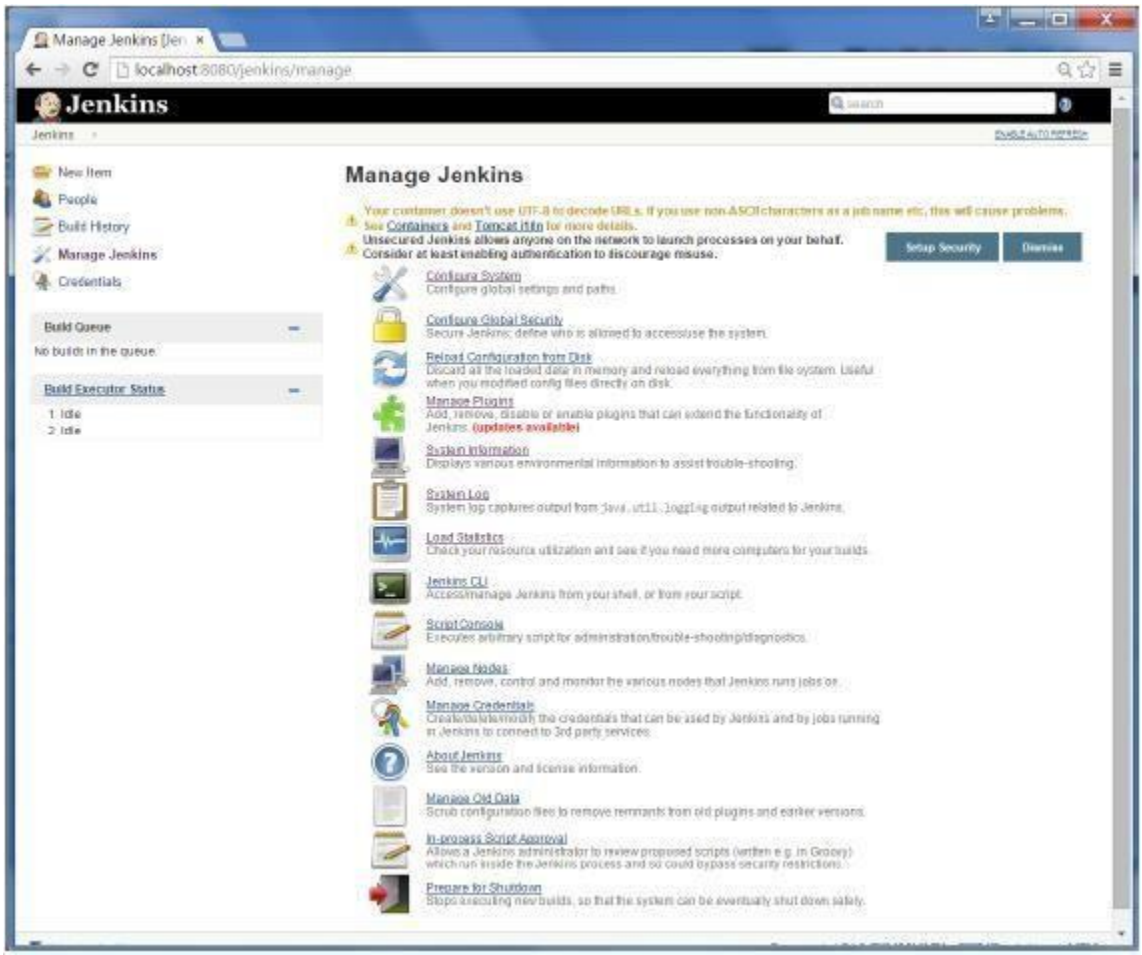
Jenkins Management

To manage Jenkins, click on the ‘Manage Jenkins’ option from the left hand menu side.

So one can get the various configuration options for Jenkins by clicking the ‘Manage Jenkins’ option from the left hand menu side.



You will then be presented with the following screen −



Some of the management options are as follows −

Configure System

and Maven, as well as security options, email servers, and other system-wide configuration details. When plugins are installed. Jenkins will add the required configuration fields dynamically after the plugins are installed.

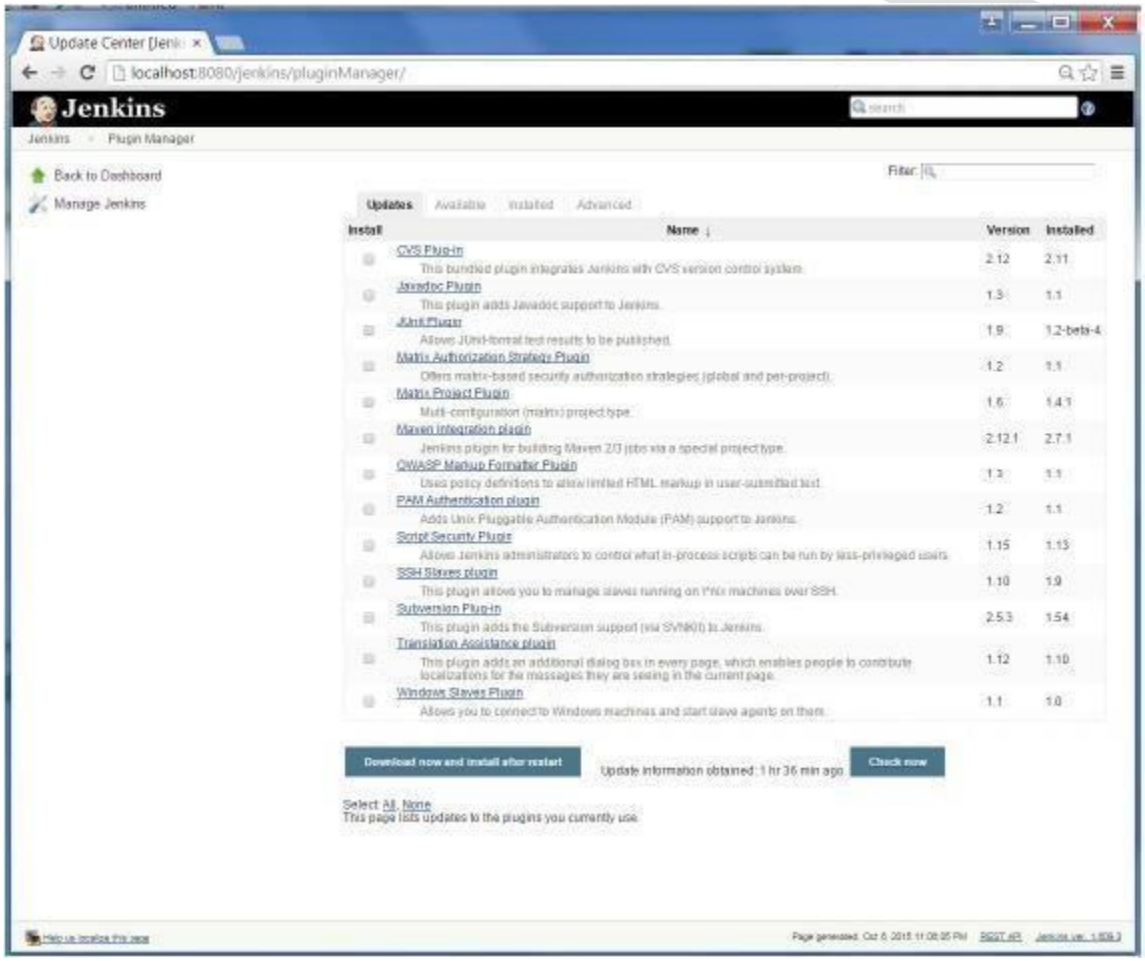
Reload Configuration from Disk

Jenkins stores all its system and build job configuration details as XML files which is stored in the Jenkins home directory. Here also all of the build history is stored. If you are migrating build jobs from one Jenkins

instance to another, or archiving old build jobs, you will need to add or remove the corresponding build job directories to Jenkins's builds directory. You don't need to take Jenkins offline to do this—you can simply use the "Reload Configuration from Disk" option to reload the Jenkins system and build job configurations directly.

Manage Plugin

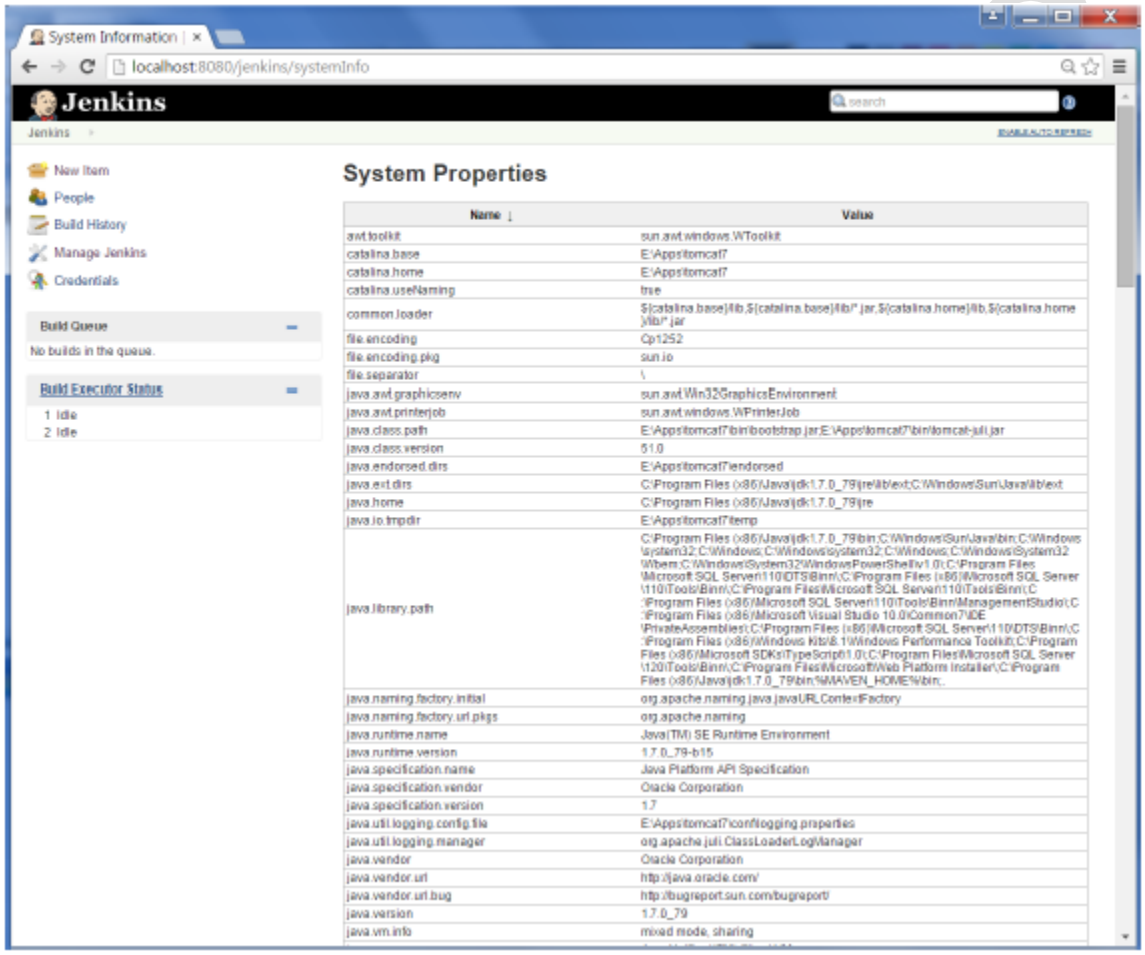
Here one can install a wide variety of third-party plugins right from different Source code management tools such as Git, Mercurial or ClearCase, to code quality and code coverage metrics reporting. Plugins can be installed, updated and removed through the Manage Plugins screen.



System Information

This screen displays a list of all the current Java system properties and system environment variables. Here one can check exactly what version of Java Jenkins is running in, what user it is running under, and so forth.

The following screenshot shows some of the name-value information available in this section.



System Log

The System Log screen is a convenient way to view the Jenkins log files in real time. Again, the main use of this screen is for troubleshooting.

Load Statistics

This pages displays graphical data on how busy the Jenkins instance is in terms of the number of concurrent builds and the length of the build queue which gives an idea of how long your builds need to wait before being

executed. These statistics can give a good idea of whether extra capacity or extra build nodes is required from an infrastructure perspective.

Script Console

This screen lets you run Groovy scripts on the server. It is useful for advanced troubleshooting since it requires a strong knowledge of the internal Jenkins architecture.

Manage nodes

Jenkins is capable of handling parallel and distributed builds. In this screen, you can configure how many builds you want. Jenkins runs simultaneously, and, if you are using distributed builds, set up build nodes. A build node is another machine that Jenkins can use to execute its builds.

Prepare for Shutdown

If there is a need to shut down Jenkins, or the server Jenkins is running on, it is best not to do so when a build is being executed. To shut down Jenkins cleanly, you can use the Prepare for Shutdown link, which prevents any new builds from being started. Eventually, when all of the current builds have finished, one will be ableto shut down Jenkins cleanly.

Manage and Assign Roles

Creating users in Jenkins

1 Open the dashboard of jenkins

2 click on manage jenkins

3 click on manage users

4 clcik on create users

5 enter user credentials

Creating roles and assgning to users

1 Open the dashboard of jenkins

2 click on manage jenkins

3 click on manage plugins

4 click on role based authorization strategy plugin

5 install it

6 go to dashboard-->manage jenkins

7 click on configure global security

8 check enable security checkbox

9 go to authorization section-->click on role based strategy radio button

10 apply-->save

11 go to dashboard of jenkins

12 click on manage jenkins

13 click on manage and assign roles

14 click on mange roles

15 go to global roles and create a role "employee"

16 for this employee in overall give read access and in view section give all access

17 go to project roles-->Give the role as developer and patter as Dev.* (ie developer role can access only those jobs whose name start with Dev)

18 similarly create another role as tester and assign the pattern as "Test.*"

19 give all permissions to developers and tester

20 apply--save

21 click on assign roles

22 go to global roles and add user1 and user2

23 check user1 and user2 as employees

24 go to item roles

25 add user1 and user2

26 check user1 as developer and user2 as tester

27 apply-->save

If we login into jenkins as user1 we can access only the development related jobs and user2 can access only the testing related jobs

Notifications in Jenkins:

whenever a jenkins job is executed we can send notifications to the team memebers.

This can be done in 2 ways

1 Cat light notifications

2 Email notifications

Catlight Notifications

Cat light is a thried party client software which has to be installed on every team members machine.

Cat light can be integrated with various CI tools

Once it is integrated with jenkins we can get desktop notifications

1 Open https://catlight.io/downloads

2 Download catlight for windows

3 Install it

4It will ask details of the jenkinserver and credentials

of jenkins

4 Catlight will display the list of jobs

5 Select those jobs for which we want notifications

6 Run those jobs from jenkins

A popup msg is displayed on the team members machine when the jenkins job is running.It will show the status of the job as running/success/fail.

Email Notifications

1 Open the dashboard of jenkins

2 Go to the job that should send email notifications

3 click on Configure

4 Go to Post build actions

5 click on add post build action

6 click on email notifications

7 enter the emailid's of the team members seperated by space

8 click on apply--->save

9 Go to the dashboard of jenkins

10 Click on manage jenkins

11 Click on configure System

12 Search for Email notifications

Triggering in Jenkins

Scheduling the jenkins job for a specific date and time

1 Open the dashboard of jenkins

2 Go to the job that we want to schedule--->Configure

3 Go to Build Triggers

4 Click on Build Periodically

5 Schedule the date and time

6 Apply--->save

Scheduling the CI-CD to start whenever developer makes changes to the code and pushes into github

1 Open the dashboard of jenkins

2 Go to the job --- >click on Configure

3 Go to Build Triggers

4 Click on POLL SCM

5 In Schedule section * * * * *

6 apply -- >save

7 The developer should make some changes to the code and push in github

Server Maintanemce

The following are some of the basic activities you will carry out, some of which are best practices for Jenkins server maintenance

URL Options

The following commands when appended to the Jenkins instance URL will carry out the relevant actions on the Jenkins instance.

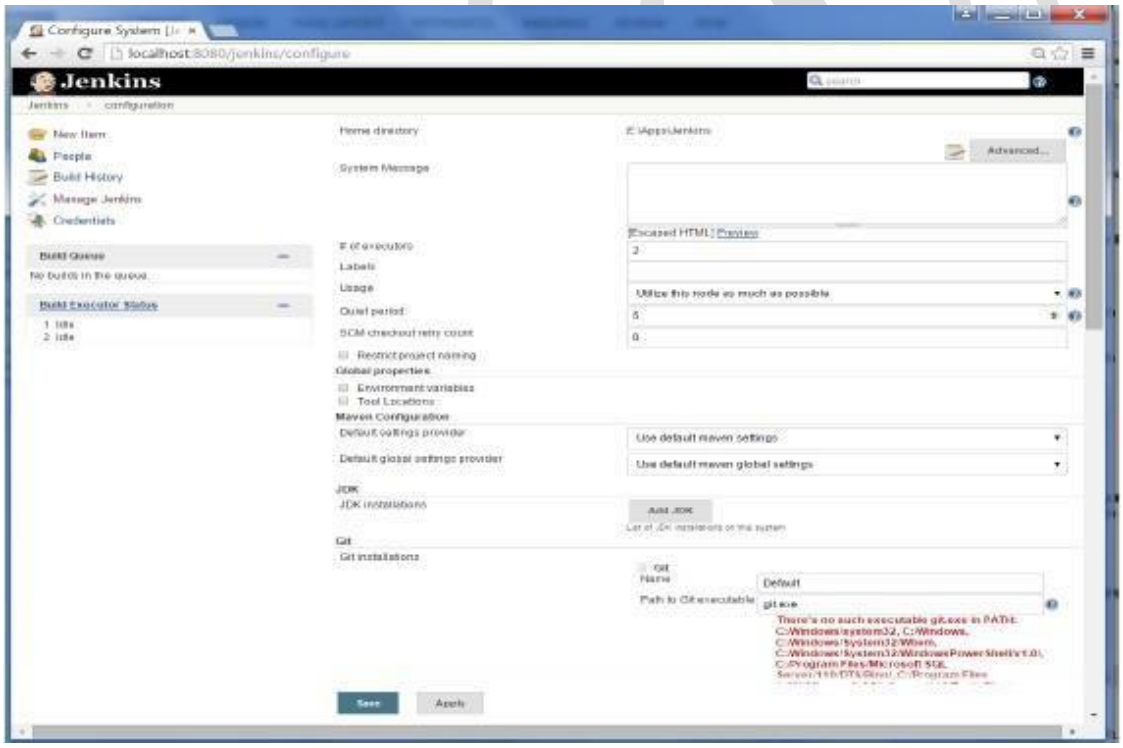
http://localhost:8080/jenkins/exit − shutdown jenkins

http://localhost:8080/jenkins/restart − restart jenkins

http://localhost:8080/jenkins/reload − to reload the configuration

Backup Jenkins Home

The Jenkins Home directory is nothing but the location on your drive where Jenkins stores all information for the jobs, builds etc. The location of your home directory can be seen when you click on Manage Jenkins → Configure system.



Set up Jenkins on the partition that has the most free disk-space – Since Jenkins would be taking source code for the various jobs defined and doing continuous builds, always ensure that Jenkins is setup on a drive that has enough hard disk space. If you hard disk runs out of space, then all builds on the Jenkins instance will start failing.

Another best practice is to write cron jobs or maintenance tasks that can carry out clean-up operations to avoid the disk where Jenkins is setup from becoming full.

Jenkins – Security:

In Jenkins you have the ability to setup users and their relevant permissions on the Jenkins instance. By default you will not want everyone to be able to define jobs or other administrative tasks in Jenkins. So Jenkins has the ability to have a security configuration in place.

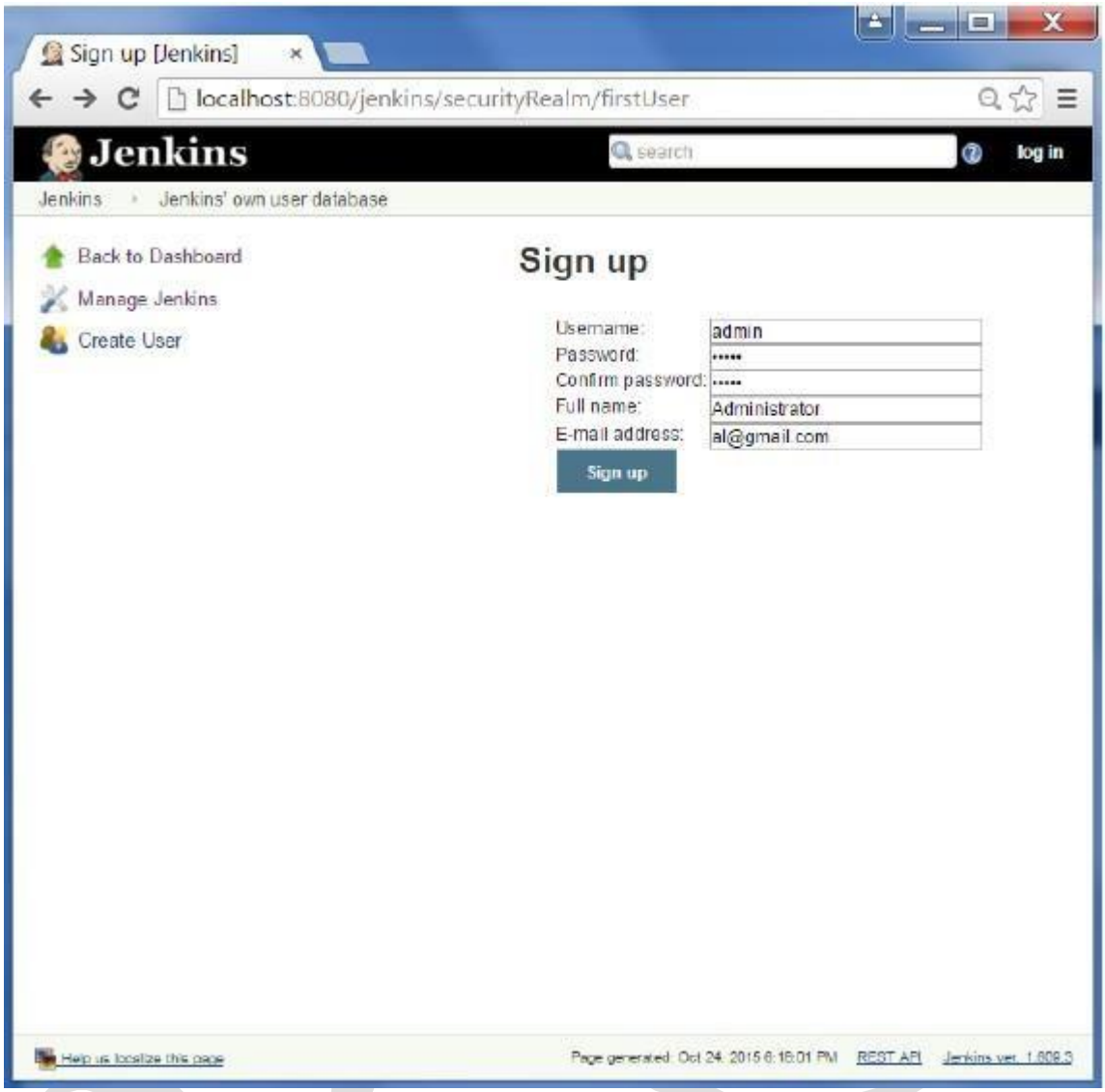
To configure Security in Jenkins, follow the steps given below.

Step 1 − Click on Manage Jenkins and choose the 'Configure Global Security' option.

Step 2 − Click on Enable Security option. As an example, let's assume that we want Jenkins to maintain it's own database of users, so in the Security Realm, choose the option of 'Jenkins' own user database'.

By default you would want a central administrator to define users in the system, hence ensure the 'Allow users to sign up' option is unselected. You can leave the rest as it is for now and click the Save button.

Step 3 − You will be prompted to add your first user. As an example, we are setting up an admin users for the system.

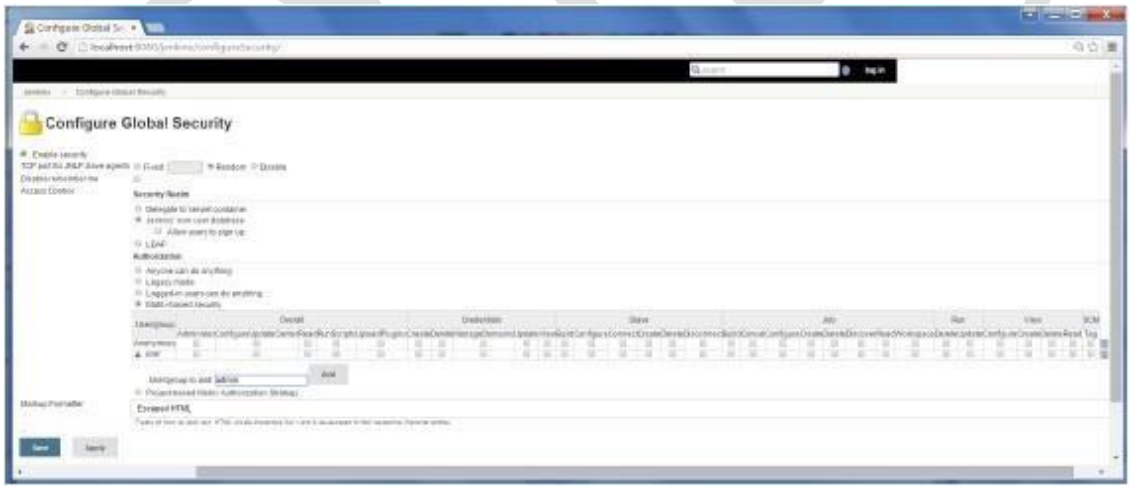


Step 4 − It's now time to setup your users in the system. Now when you go to Manage Jenkins, and scroll down, you will see a 'Manage Users' option. Click this option.

Step 5 − Just like you defined your admin user, start creating other users for the system. As an example, we are just creating another user called 'user'.

Step 6 − Now it's time to setup your authorizations, basically who has access to what. Go to Manage Jenkins → Configure Global Security.

Now in the Authorization section, click on 'Matrix based security'



Step 7 − If you don't see the user in the user group list, enter the user name and add it to the list. Then give the appropriate permissions to the user.

Click on the Save button once you have defined the relevant authorizations.

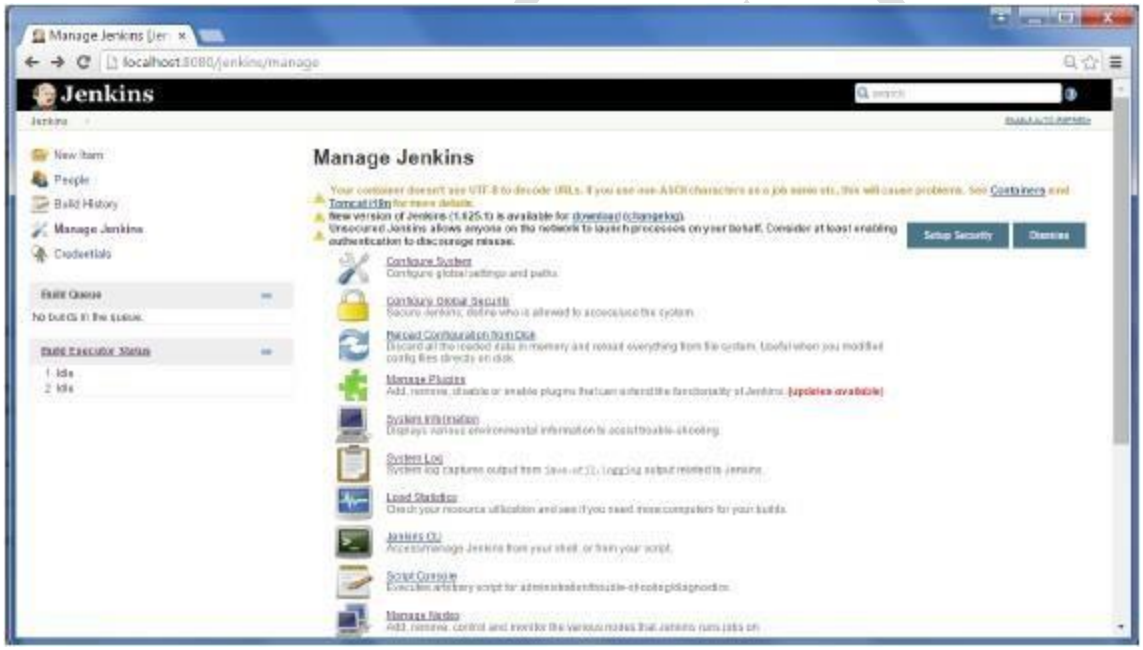
Your Jenkins security is now setup.

Note − For Windows AD authentication, one has to add the Active Directory plugin to Jenkins.

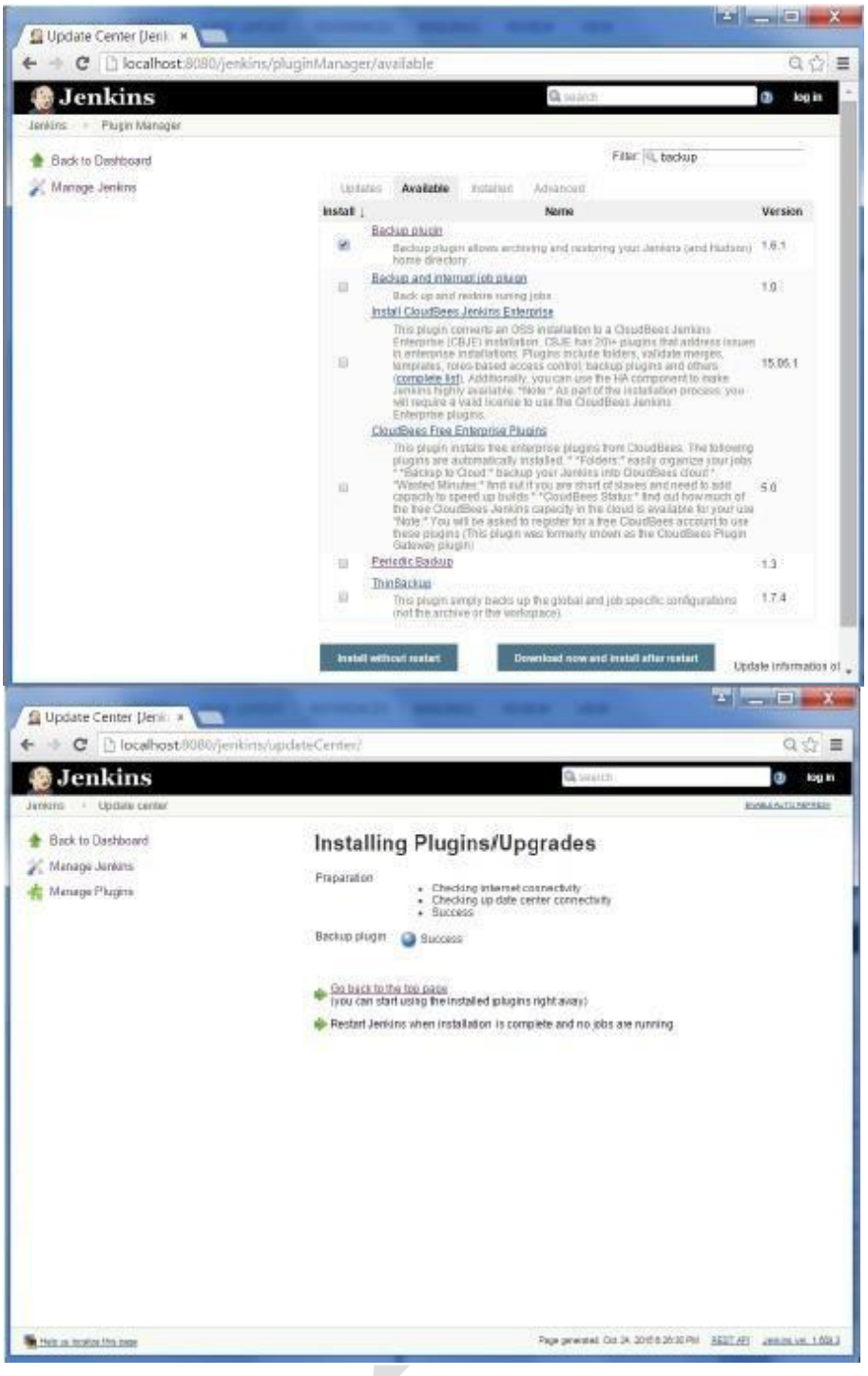
Jenkins – Backup:

Jenkins has a backup plugin which can used to backup critical configuration settings related to Jenkins. Follow the steps given below to have a backup in place.

Step 1 − Click on Manage Jenkins and choose the 'Manage Plugins' option.

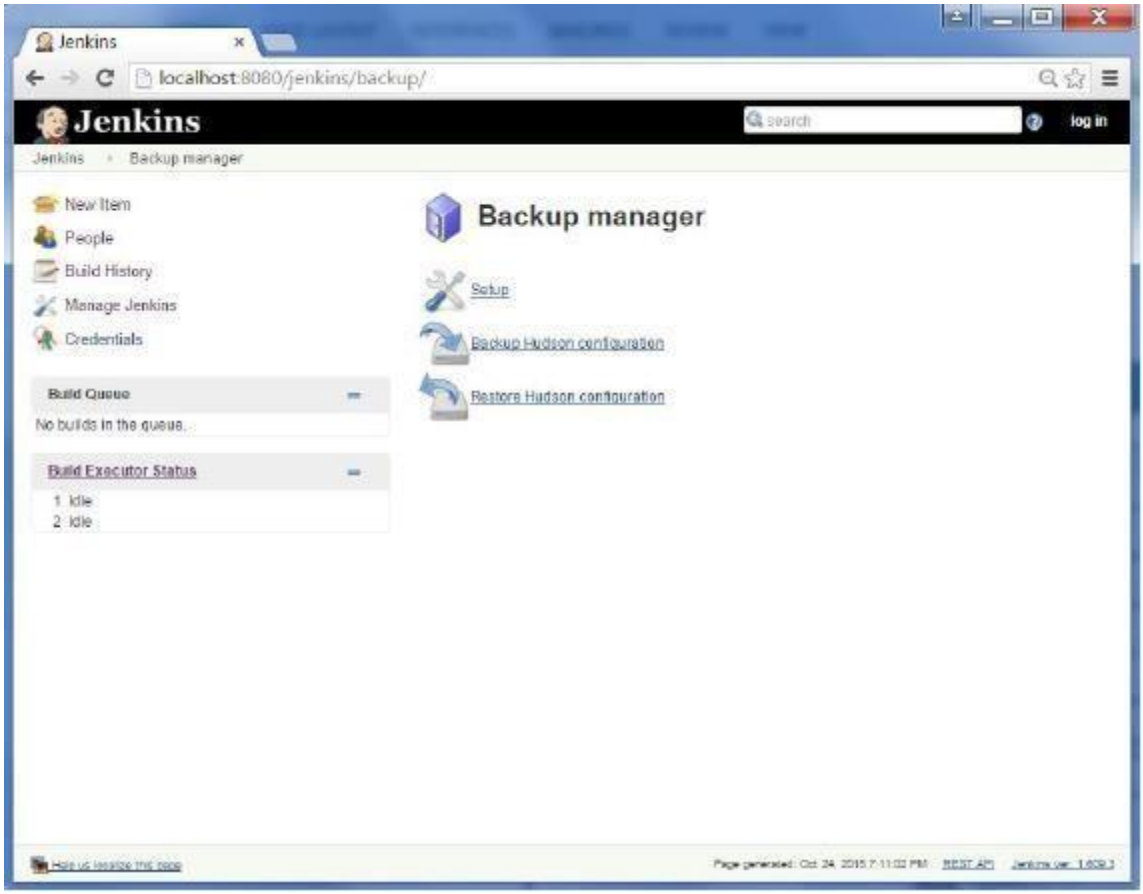


Step 2 − In the available tab, search for 'Backup Plugin'. Click On Install without Restart. Once done, restart the Jenkins instance

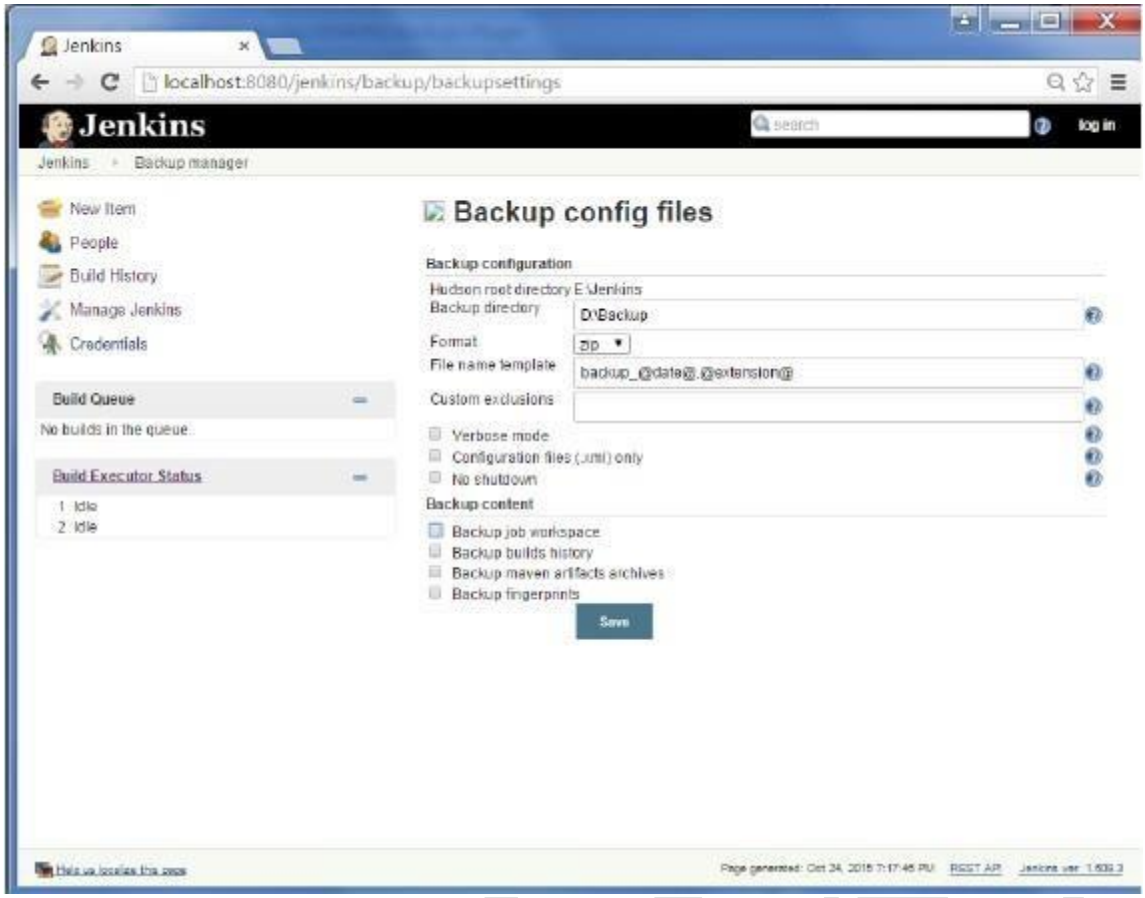


Step 3 − Now when you go to Manage Jenkins, and scroll down you will see 'Backup Manager' as an option. Click on this option.

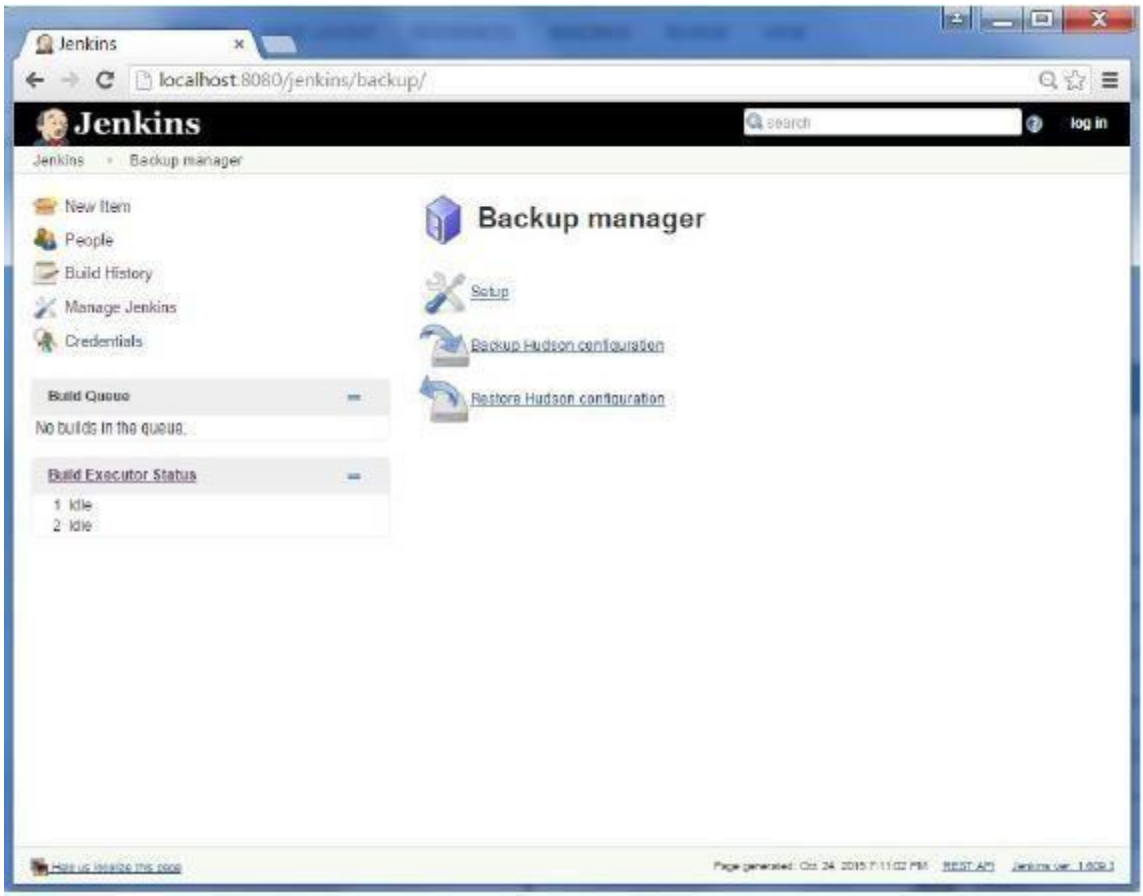
Step 4 − Click on Setup.



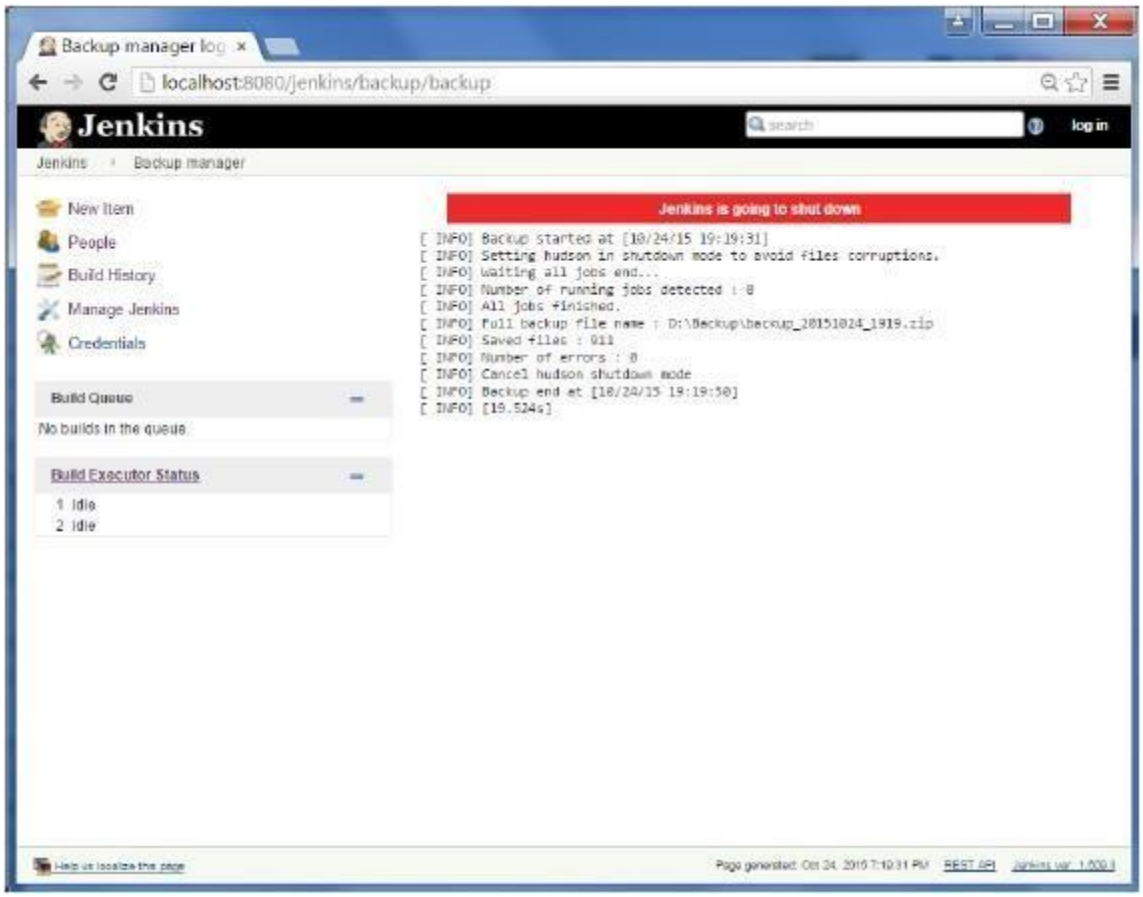
Step 5 − Here, the main field to define is the directory for your backup. Ensure it's on another drive which is different from the drive where your Jenkins instance is setup. Click on the Save button.



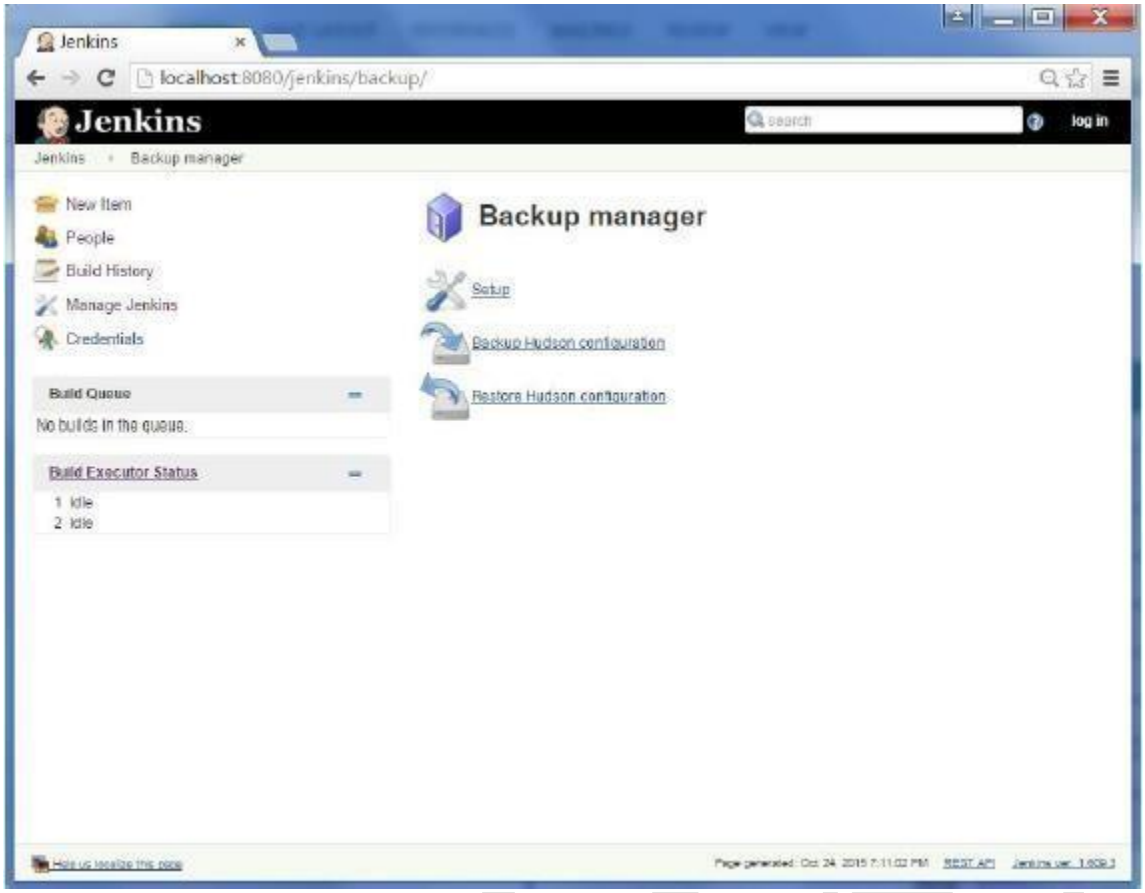
Step 6 − Click on the 'Backup Hudson configuration' from the Backup manager screen to initiate the backup.



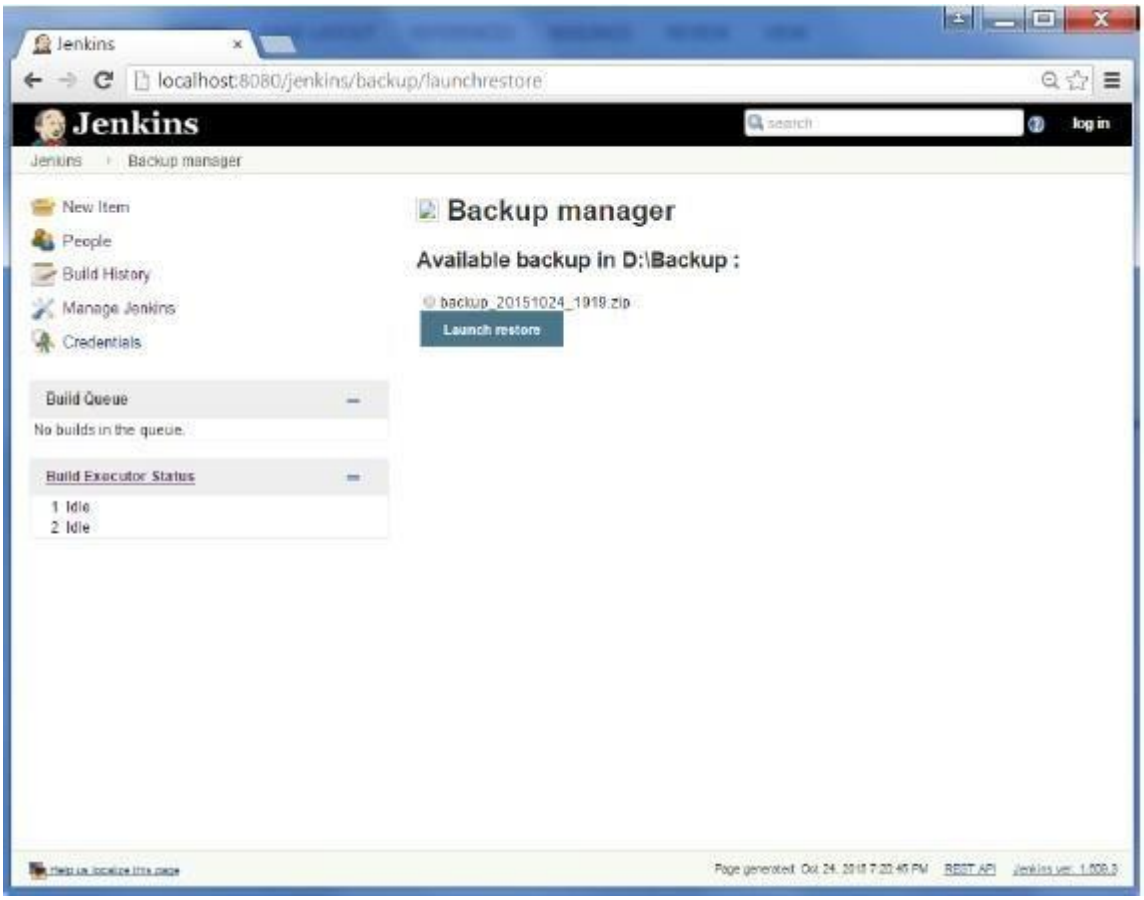
The next screen will show the status of the backup



To recover from a backup, go to the Backup Manager screen, click on Restore Hudson configuration.



The list of backup's will be shown, click on the appropriate one to click on Launch Restore to begin the restoration of the backup.



Master and slave

This single Jenkins server was not enough to meet certain requirements like:

- Sometimes you might need several different environments to test your builds. This cannot be done by a single Jenkins server.
- If larger and heavier projects get built on a regular basis then a single Jenkins server cannot simply handle the entire load.

To address the above stated needs, Jenkins distributed architecture was introduced.

Jenkins Distributed Architecture

Jenkins uses a Master-Slave architecture to manage distributed builds. In this architecture, Master and Slave communicate through TCP/IP protocol.

Jenkins Master

Your main Jenkins server is the Master. The Master's job is to handle:

- Scheduling build jobs.
- Dispatching builds to the slaves for the actual execution.
- Monitor the slaves (possibly taking them online and offline as required).
- Recording and presenting the build results.
- A Master instance of Jenkins can also execute build jobs directly.

Jenkins Slave

A Slave is a Java executable that runs on a remote machine. Following are the characteristics of Jenkins Slaves:

- It hears requests from the Jenkins Master instance.
- Slaves can run on a variety of operating systems.
- The job of a Slave is to do as they are told to, which involves executing build jobs dispatched by the Master.
- You can configure a project to always run on a particular Slave machine, or a particular type of Slave machine, or simply let Jenkins pick the next available Slave.

Now let us look at an example in which Jenkins is used for testing in different environments like: Ubuntu, MAC, Windows etc.

The following functions are performed in master and slave

- Jenkins checks the Git repository at periodic intervals for any changes made in the source code.
- Each builds requires a different testing environment which is not possible for a single Jenkins server. In order to perform testing in different environments Jenkins uses various Slaves as shown in the diagram.
- Jenkins Master requests these Slaves to pe

Configuration and Implementiation:

Configure on slave:

1 Launch one linux machine it is the slave machine
2 install java 1.8
3 download slave.jar file from master machine
Eg. http://master-ip-address:8080/jnlpJars/slave.jar
4 give full permissions to slave.jar file

Eg. Sudo chmod 777 slave.jar
5 create one newdir
Eg. Mkdir workspace
6 give full permissions to workspace
Eg. Sudo chmod 777 workspace
7 setup project based infrastrature that is what packages are rquired to execute project like git maven
8 set the password for user
Eg. Sudo passwd ubuntu
9 make password authendication no to yes in ssh configuration
Sudo vim /etc/ssh/sshd_config
Goto password authendication pattern and make yes to no and save (escape and :wq)
10 restart ssh service (sudo service ssh restart)

Configure on master:

1 open jenkins home page
2 goto manage jenkins
3 goto manage nodes
4 click on create newnode
5 give any name for this node
6 # number of executors is one
7 Give label name for this node
8 Mention the workspace in slave (/home/ubuntu/workspace)
9 Goto launch command method---select launch command from master
10 Write command(ssh ubuntu@ip-of-slave slave.jar)
11 Apply and save
12 Make passwordless authendication from master to slave machine
13 Goto project and goto genderal section
14 Select 'restrict where the project can run' option and selct slave label name
15 Apply and slave
16 This project is distributed to slave machine and run from that machine

BuildPipeline

This is a plugin jenkins which is used to get a better graphical view of the jobs.

1 Open the dashboard of jenkins--->Manage jenkins

2 Click on Manage Plugins--->Go to Available Section

3 Search for "BuildPipeline" plugin-->install it

4 Go to dashboard of jenkins--->Go to the location where all the jobs are listed--->Click on + icon

5 Enter some view name--->Select Build pipeline view

6 Go to Upstream / downstream config and select the initial job as development--->save

Code as Pipeline

This is a programatic way for perfroming CI-CD through jenkins.

We use a file called Jenkinsfile which is created using "groovy".

This jenkinsfile will be part of the code base that is created by the developer.

Whenever the developer makes any modifications to the code

the jenkinsfile will immediately trigger the CI-CD

Pipeline

Jenkins Pipeline (or simply "Pipeline" with a capital "P") is a suite of plugins which supports implementing and integrating *continuous delivery pipelines* into Jenkins.

A *continuous delivery (CD) pipeline* is an automated expression of your process for getting software from version control right through to your users and customers. Every change to your software (committed in source control) goes through a complex process on its way to being released. This process involves building the software in a reliable and repeatable manner, as well as progressing the built software (called a "build") through multiple stages of testing and deployment.

The definition of a Jenkins Pipeline is written into a text file (called a **Jenkinsfile**) which in turn can be committed to a project's source control repository. This is the foundation of "Pipeline-as-code"; treating the CD pipeline a part of the application to be versioned and reviewed like any other code.

Creating a **Jenkinsfile** and committing it to source control provides a number of immediate benefits:

- Automatically creates a Pipeline build process for all branches and pull requests.
- Code review/iteration on the Pipeline (along with the remaining source code).
- Audit trail for the Pipeline.
- Single source of truth for the Pipeline, which can be viewed and edited by multiple members of the project.

While the syntax for defining a Pipeline, either in the web UI or with a Jenkinsfile is the same, it is generally considered best practice to define the Pipeline in a **Jenkinsfile** and check that in to source control.

Declarative versus Scripted Pipeline syntax:

A **Jenkinsfile** can be written using two types of syntax - Declarative and Scripted.

Declarative and Scripted Pipelines are constructed fundamentally differently. Declarative Pipeline is a more recent feature of Jenkins Pipeline which:

- provides richer syntactical features over Scripted Pipeline syntax, and
- is designed to make writing and reading Pipeline code easier.

Many of the individual syntactical components (or "steps") written into a **Jenkinsfile**, however, are common to both Declarative and Scripted Pipeline.

Why Pipeline?

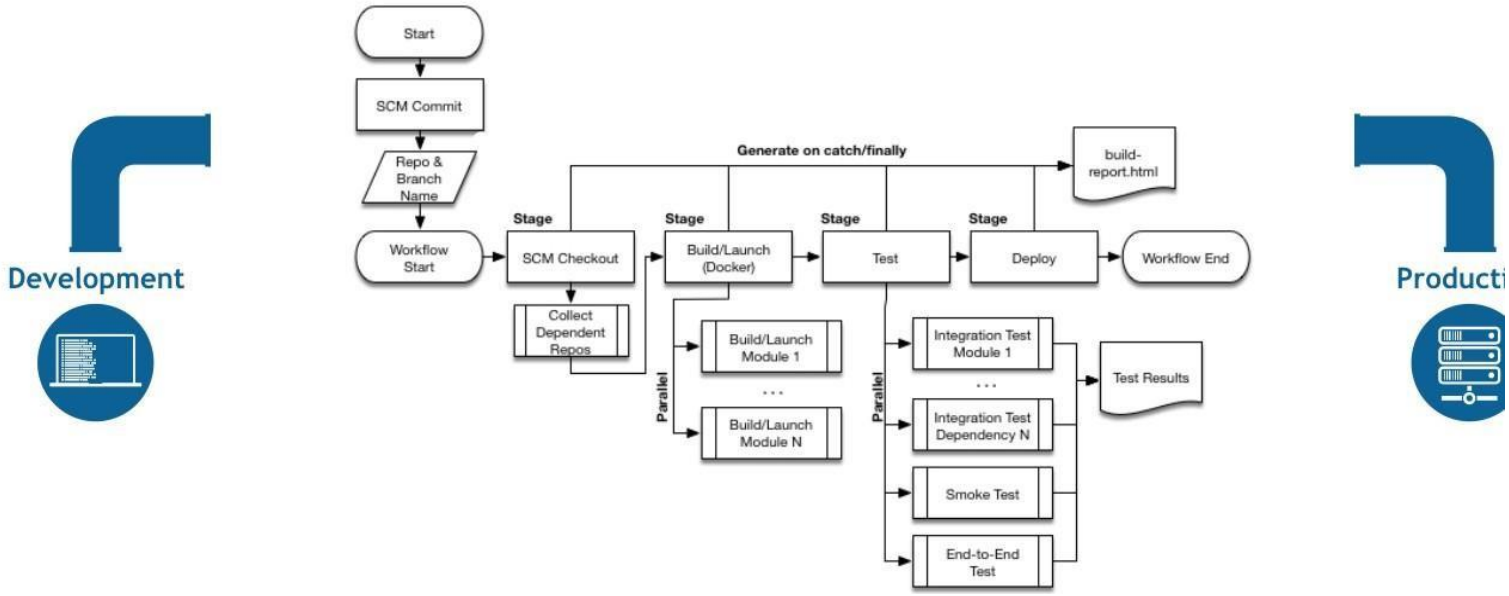
Jenkins is, fundamentally, an automation engine which supports a number of automation patterns. Pipeline adds a powerful set of automation tools onto Jenkins, supporting use cases that span from simple continuous integration to comprehensive CD pipelines. By modeling a series of related tasks, users can take advantage of the many features of Pipeline:

- **Code**: Pipelines are implemented in code and typically checked into source control, giving teams the ability to edit, review, and iterate upon their delivery pipeline.
- **Durable**: Pipelines can survive both planned and unplanned restarts of the Jenkins master.
- **Pausable**: Pipelines can optionally stop and wait for human input or approval before continuing the Pipeline run.
- **Versatile**: Pipelines support complex real-world CD requirements, including the ability to fork/join, loop, and perform work in parallel.
- **Extensible**: The Pipeline plugin supports custom extensions to its DSL and multiple options for integration with other plugins.

While Jenkins has always allowed rudimentary forms of chaining Freestyle Jobs together to perform sequential tasks, Pipeline makes this concept a first-class citizen in Jenkins.

Building on the core Jenkins value of extensibility, Pipeline is also extensible both by users with Pipeline Shared Libraries and by plugin developers.

The flowchart below is an example of one CD scenario easily modeled in Jenkins Pipeline:



Pipeline concepts:

The following concepts are key aspects of Jenkins Pipeline, which tie in closely to Pipeline syntax (see the overview below).

Pipeline
A Pipeline is a user-defined model of a CD pipeline. A Pipeline's code defines your entire build process, which typically includes stages for building an application, testing it and then delivering it.

Also, a **pipeline** block is a key part of Declarative Pipeline syntax.

Node
A node is a machine which is part of the Jenkins environment and is capable of executing a Pipeline.

Also, a node block is a key part of Scripted Pipeline syntax.

Stage
A **stage** block defines a conceptually distinct subset of tasks performed through the entire Pipeline (e.g. "Build", "Test" and "Deploy" stages), which is used by many plugins to visualize or present Jenkins Pipeline status/progress. [6]

Step
A single task. Fundamentally, a step tells Jenkins *what* to do at a particular point in time (or "step" in the process). For example, to execute the shell command **make** use the **sh** step: **sh 'make'**. When a plugin extends the Pipeline DSL, that typically means the plugin has implemented a new *step*.

Pipeline syntax overview

The following Pipeline code skeletons illustrate the fundamental differences between Declarative Pipeline syntax and Scripted Pipeline syntax.

Be aware that both stages and steps (above) are common elements of both Declarative and Scripted Pipeline syntax.

Declarative Pipeline fundamentals

In Declarative Pipeline syntax, the **pipeline** block defines all the work done throughout your entire Pipeline.

Jenkinsfile (Declarative Pipeline)

```
pipeline {
    agent any
    stages {
        stage('Build') {
            steps {
                //
            }
        }
        stage('Test') {
            steps {
                //
            }
        }
        stage('Deploy') {
            steps {
                //
            }
        }
    }
}
```

Execute this Pipeline or any of its stages, on any available agent.

Defines the "Build" stage.

Perform some steps related to the "Build" stage.

Defines the "Test" stage.

Perform some steps related to the "Test" stage.

Defines the "Deploy" stage.

Perform some steps related to the "Deploy" stage.

Scripted Pipeline fundamentals:

In Scripted Pipeline syntax, one or more **node** blocks do the core work throughout the entire Pipeline. Although this is not a mandatory requirement of Scripted Pipeline syntax, confining your Pipeline's work inside of

1. Schedules the steps contained within the block to run by adding an item to the Jenkins queue. As soon as an executor is free on a node, the steps will run.
2. Creates a workspace (a directory specific to that particular Pipeline) where work can be done on files checked out from source control.
   Caution: Depending on your Jenkins configuration, some workspaces may not get automatically cleaned up after a period of inactivity.

Jenkinsfile (Scripted Pipeline)

```
node {
   stage('Build') {
      //
   }
   stage('Test') {
      //
   }
   stage('Deploy') {
      //
   }
}
```

Execute this Pipeline or any of its stages, on any available agent.

Defines the "Build" stage. **stage** blocks are optional in Scripted Pipeline syntax. However, implementing stage blocks in a Scripted Pipeline provides clearer visualization of each `stage's subset of tasks/steps in the Jenkins UI.

Perform some steps related to the "Build" stage.

Defines the "Test" stage.

Perform some steps related to the "Test" stage.

Defines the "Deploy" stage.

Perform some steps related to the "Deploy" stage.

Pipeline example

Here is an example of a **Jenkinsfile** using Declarative Pipeline syntax - its Scripted syntax equivalent can be accessed by clicking the **Toggle Scripted Pipeline** link below:

Jenkinsfile (Declarative Pipeline)

```
pipeline
   {
   agent
   any
   stage
   s {
      stage('Buil
          d') {
         steps {
            sh 'make'
       }
      }
      stage('Te
         st'){
         steps {
            sh 'make check'
            junit 'reports/**/*.xml'
       }
```

```
    }
  }
}
```

pipeline is Declarative Pipeline-specific syntax that defines a "block" containing all content and instructions for executing the entire Pipeline.

agent is Declarative Pipeline-specific syntax that instructs Jenkins to allocate an executor (on a node) and workspace for the entire Pipeline.

stage is a syntax block that describes a stage of this Pipeline. Read more about stage blocks in Declarative Pipeline syntax on the Pipeline syntax page. As mentioned above, stage blocks are optional in Scripted Pipeline syntax.

steps is Declarative Pipeline-specific syntax that describes the steps to be run in this stage.

sh is a Pipeline step (provided by the Pipeline: Nodes and Processes plugin) that executes the given shell command.

junit is another a Pipeline step (provided by the JUnit plugin) for aggregating test reports.

node is Scripted Pipeline-specific syntax that instructs Jenkins to execute this Pipeline (and any stages contained within it), on any available agent/node. This is effectively equivalent to agent in Declarative Pipeline-specific syntax.

CI-CD using Scripted Pipeline:

Node(‘master’)

{

Try

{

Stage(‘continious download’)

{

Git ‘developers-id’

```
}
Stage(‘continious build’)
{
Sh ‘mvn package’
}
Stage(‘continious deployement’)
{
Sh ‘scp artifact-ghenerated-location-on-dev username@qa-ip-address:qa-server-tomcat-webapps-location’
}
Stage(‘continious testing’)
{
Git ‘selenium-developer-git-repo-id’
}
}
Catch(exception E)
{
Send notification
}
Stage(‘continious delivery’)
{
Sh ‘scp artifact-ghenerated-location-on-dev username@prod-server-ip:qa-server-tomcat-webapps-location’
}
}
```

CI-CD using Declarative Pipeline:

```
Pipeline
{
Agent any
{
stages
{
Stage('continious download')
{
Steps
{
Git 'developers-id'
}
}
Stage('continious build')
{
Steps
{
Sh 'mvn package'
}
}
Stage('continious deployement')
{
Steps
{
```

Sh ‘scp artifact-ghenerated-location-on-dev username@qa-ip-address:qa-server-tomcat-webapps-location’

}

}

Stage(‘continious testing’)

{

Steps

{

Git ‘selenium-developer-git-repo-id’

}

}

}

Post

{

Success

{

Stage(‘continious delivery’)

{

Steps

{

Sh ‘scp artifact-ghenerated-location-on-dev username@prod-server-ip:qa-server-tomcat-webapps-location’

}

}

}

Failure

{

Send notification

```
}
}
}
}
```

BlueOcean Plugin

This is a new UI for Jenkins which will become the default interface of Jenkins in the coming months.This is most useful for creating pipeline project

1 Open dashbord of jenkins

2 Manage Jenkins--->manage plugins

3 Go to available section--->Search for BlueOcean --->Click on "Download Now Install after restart" --->restart jenkins

4 Go to dashboard of jenkins--->click on Blue Ocean

5 Click on "New Pipeline"

Now it will ask where the development code is present alogn with the jenkins file.

Since our code is present in git hub

6 Select Github--->Click on Create access token

7 Enter username and passoword of Github account

11 Enter some description for the access token

12 Copy the access token and enter in jenkins

13 Jenkins will list all the projects in github

14 Select the exact project where our code is present

15 Jenkins will automatically read the jenkinsfile and create the ci-cd

Please substitute as needed.

Vagrant (Virtualization Platform)

Set Up a Local Linux Environment with Vagrant

Below is the general process that I've adopted after learning the fundamentals and working through some trial and error. Follow these steps, editing any necessary information specific to your individual needs, and you should be up and running with Vagrant in about 10 minutes or less!

1. Install VirtualBox
2. Install Vagrant
3. Create a local directory for Vagrant
4. Create a **Vagrantfile** in your newly created directory
5. Run **vagrant up** and provisioning your virtual machine

Note: the steps below are outlined as if the sample Vagrant file included in this post is used in its entirety—meaning all file paths and constants referenced in that file will also be referenced throughout the instructions. Please substitute as needed.

1. Install VirtualBox

To grab the latest version of VirtualBox, head on over to the download pageand select the version appropriate for your operating system. Download and install, following the prompts as you would any other program. And while my article focuses mainly on my experiences with this workflow on OS X, the same principles and concepts *should* apply to Windows. Once installed, let's move our attention to Vagrant.

Note: as of writing, VirtualBox 5.0.20 was installed.

2. Install Vagrant

Now that VirtualBox is installed, let's focus on getting Vagrant installed to your machine. Again, head over to their download page and grab the version appropriate for your operating system. Much like above, download and install, following the prompts as you would any other program.

Note: as of writing, Vagrant 1.8.1 was installed.

3. Create a Local Directory for Vagrant

This step comes down to personal preference as to where you want to keep and manage your **Vagrantfile**, as well as offering you some creative options around how and where you want to sync files and folders between your host and guest machines.

Personally, I find that keeping directories for each Vagrant instance within my local Home directory is an easy to remember and easy to manage experience.

For example, I've created a directory titled "new-vm" in my Home directory. This is the exact same value for the **VM_NAME** constant found in the sample **Vagrantfile** in step 4. I then added the **Vagrantfile** specific to this VM within that directory for provisioning.

In short, I executed the following commands in Terminal:

```
cd ~/ && mkdir new-vm
cd new-vm && touch Vagrantfile
```

The first command changes to my Home directory and then creates the "new-vm" directory within Home. The second command changes to the newly created directory and then creates the **Vagrantfile** within that new directory.

4. Update your Vagrantfile

By following along with the instructions from step 3, you should now have a unique directory and **Vagrantfile** for your virtual machine within your Home directory on your Mac. For example, I now have:

~/new-vm/Vagrantfile

Now it's time to review the sample **Vagrantfile** below, and make appropriate edits for your specific needs.

Sample Vagrantfile

The next step is to add some instructions within your **Vagrantfile** for Vagrant to execute during the virtual machine provisioning process. To help expedite this process, and set a handful of the typical default items, below is a sample **Vagrantfile** that will get you set up with a Ubuntu Linux development enviornment with Node.js and Git. This file works through:

1. Setting several constants around Linux OS, directories, users, and networking between host/guest machines
2. Provisioning a virtual machine through VirtualBox, with 1 CPU, 512MB of memory, and 40GB of storage space
3. Installing latest Git, Node 6.x, latest npm, and updating Ubuntu

Sample:

```
# encoding: utf-8
# -*- mode: ruby -*-
# vi: set ft=ruby :
# Box / OS
VAGRANT_BOX = 'ubuntu/trusty64'
# Memorable name for your
VM_NAME = 'new-vm'
```

```
# VM User — 'vagrant' by default
VM_USER = 'vagrant'
# Username on your Mac
MAC_USER = 'John'
# Host folder to sync
HOST_PATH = '/Users/' + MAC_USER + '/' + VM_NAME
# Where to sync to on Guest — 'vagrant' is the default user name
GUEST_PATH = '/home/' + VM_USER + '/' + VM_NAME


# # VM Port — uncomment this to use NAT instead of DHCP
# VM_PORT = 8080
Vagrant.configure(2) do |config|
  # Vagrant box from Hashicorp
  config.vm.box = VAGRANT_BOX

  # Actual machine name
  config.vm.hostname = VM_NAME
  # Set VM name in Virtualbox
  config.vm.provider "virtualbox" do |v|
    v.name = VM_NAME
    v.memory = 2048
  end
#DHCP — comment this out if planning on using NAT instead
  config.vm.network "private_network", type: "dhcp"
  # # Port forwarding — uncomment this to use NAT instead of DHCP
  # config.vm.network "forwarded_port", guest: 80, host: VM_PORT
  # Sync folder
  config.vm.synced_folder HOST_PATH, GUEST_PATH
  # Disable default Vagrant folder, use a unique path per project
  config.vm.synced_folder '.', '/home/'+VM_USER+", disabled: true
  # Install Git, Node.js 6.x.x, Latest npm
  config.vm.provision "shell", inline: <<-SHELL
    apt-get update
    apt-get install -y git
    curl -sL https://deb.nodesource.com/setup_6.x | sudo -E bash -
    apt-get install -y nodejs
    apt-get install -y build-essential
    npm install -g npm
    apt-get update
apt-get upgrade -y
    apt-get autoremove -y
```

```
  SHELL
end
```

Now that you have reviewed this sample **Vagrantfile**, there are two paths you can choose to continue:

A. Fine Tune The Above Sample for YourNeeds

Make any appropriate edits that you may need (such as editing file paths, user name, VM name, etc.) and paste the contents into your local **Vagrantfile**. Once you're done, continue to step 5.

B. Use a Default Vagrantfile Instead

If you dislike the above sample and want to opt for the default **Vagrantfile**provided by Vagrant, you can simply remove the file we had created in step 3 and create a new file that contains the bare minimum to provision a virtual machine with Vagrant:

```
cd ~/new-vm
rm -r Vagrantfile
vagrant init
```

5. "vagrant up" and Provisioning your Virtual Machine

With your **Vagrantfile** in place, it's time to provision your virtual machine!

Open Terminal and enter the following commands:

```
cd ~/new-vm
vagrant up
```

During this process Vagrant will download ubuntu/trusty64 directly from Hashicorp's Atlas and then provision the virtual machine as specified in your **Vagrantfile**. Depending upon your Internet connection and computer

speed, the entire process should take around 3 to 5 minutes to complete. Once the virtual machine is provisioned, you can log into your virtual machine with the following command:

vagrant ssh

This command will take care of connecting you directly to your VM via SSH. It handles the username and password—which, by default, are both *vagrant*/*vagrant*.

The above, "vagrant up" and "vagrant ssh" will more than likely be the two de facto commands you'll use for initializing and accessing your VM. Below are a few more useful commands that will come in handy.

Turning Your Virtual Machine On and Off

Sometimes we need to turn on, turn off, pause, or destroy our Vagrant managed virtual machines, especially after rebooting or testing out new features, etc.

To turn your VM on, navigate to the directory with your Vagrantfile:

vagrant up

To pause your VM, navigate to the directory with your Vagrantfile:

vagrant suspend

To turn your VM off, navigate to the directory with your Vagrantfile:

vagrant halt

To destroy your VM, navigate to the directory with your Vagrantfile:

vagrant destroy

Vagrantfile for Ansible:

```
# -*- mode: ruby -*-
# vi: set ft=ruby :

VAGRANTFILE_API_VERSION = "2"

Vagrant.configure(VAGRANTFILE_API_VERSION) do |config|
config.ssh.insert_key = false
config.vm.provider :virtualbox do |vb|
vb.customize ["modifyvm", :id, "--memory", "256"]
 end

 # control
 config.vm.define "control" do |app|
 app.vm.hostname = "control"
 app.vm.box = "ubuntu/trusty64"
 app.vm.network :private_network, ip: "10.10.10.91"
end
```

```
# Load Balancer.
config.vm.define "lb" do |app|
app.vm.hostname = "lb"
app.vm.box = "ubuntu/trusty64"
app.vm.network :private_network, ip: "10.10.10.92"
end

# Application server 1.
config.vm.define "app1" do |app|
app.vm.hostname = "orc-app1.dev"
app.vm.box = "ubuntu/trusty64"
app.vm.network :private_network, ip: "10.10.10.93"
end

# Application server 2.
config.vm.define "app2" do |app|
app.vm.hostname = "orc-app2.dev"
app.vm.box = "ubuntu/trusty64"
app.vm.network :private_network, ip: "10.10.10.94"
```

```
# Database server.
config.vm.define "dbserver" do |db|
db.vm.hostname = "orc-db.dev"
db.vm.box = "ubuntu/trusty64"
db.vm.network :private_network, ip: "10.10.10.95"
end
end
```

Vagrantfile for Dockerswarm:

```
# -*- mode: ruby -*-
# vi: set ft=ruby :

VAGRANTFILE_API_VERSION = "2"

Vagrant.configure(VAGRANTFILE_API_VERSION) do |config|
config.ssh.insert_key = false
config.vm.provider :virtualbox do |vb|
vb.customize ["modifyvm", :id, "--memory", "1024"]
end
```

```
 # manager
 config.vm.define "Slave1" do |app|
 app.vm.hostname = "Slave"
 app.vm.box = "ubuntu/trusty64"
 app.vm.network :private_network, ip: "10.10.10.24"
 end
end
```

Vagrantfile for Jenkins:

```
# -*- mode: ruby -*-
# vi: set ft=ruby :

VAGRANTFILE_API_VERSION = "2"

Vagrant.configure(VAGRANTFILE_API_VERSION) do |config|
config.ssh.insert_key = false
config.vm.provider :virtualbox do |vb|
vb.customize ["modifyvm", :id, "--memory", "1024"]
 end
```

```
# control
config.vm.define "Jenkins" do |app|
app.vm.hostname = "Jenkins"
app.vm.box = "ubuntu/trusty64"
app.vm.network :private_network, ip: "10.0.0.20"
end


# Load Balancer.
config.vm.define "QA" do |app|
app.vm.hostname = "QA"
app.vm.box = "ubuntu/trusty64"
app.vm.network :private_network, ip: "10.0.0.21"
end

# Application server 1.
config.vm.define "Prod" do |app|
app.vm.hostname = "Prod"
app.vm.box = "ubuntu/trusty64"
app.vm.network :private_network, ip: "10.0.0.22"
end
end
```

Docker (Virtualization Platform)

This is a containerization platform which can be used for creating the development environment, testing environment and production environment..etc. Docker use a concept called containers. This is the next step in virtualization.

Virtualization

This is a process where it is posssible to run multiple OS's on one server. This is done thorugh an applcation called hypervisor.

On the server(baremetal) we first install the Host OS and on top of it we install the hypervisor s/w(Vmware esxi,Citrix xen,Microsoft Hyper-v etc).On the hypevisor we can install the Guest os and on the guest os we install the appllications that we require.

The problem with the above architecture is the applications running on the guest os have to pass through multiple layers in order to access the h/w resources.also it is not possible to aalocate h/w resources dynamically to the VM's depending on their requirement.

Docker Containarization

In docker we have a bare metal on top of which host OS is installed and on the host OS we install a s/w application called "Docker engine".On the docker engine we can run any application as a container.The advantage of this process is these applications have to pass through less number of layers to access the hardware resources

Also docker engine can dynamically allocate h/w resources to individual contianers depending on how many process are running on it

Through docker we achive what is called "process isolation" ie we seperate the application from its dependency on the underlying OS and we create application running only on docker engine.These applications can then be deployed on any OS where docker is running.

Installing docker on windows

Docker can be installed only on Windows 10 Prof 64 bit version or Windows 2016 Server Edition

1 Open https://docs.docker.com/docker-for-windows/install/

2 Download and install docker

3To execute docker commands on windows

Open Power shell

Run the docker commands

Note: docker when installed on windows activates an application called Hyper-v.This application will not allow any other virtualization s/w to run

Installing docker on Linux

1 Open get.docker.com

2Copy the first two commands and paste in a linux terminal

```
curl -fsSL get.docker.com -o get-docker.sh

sh get-docker.sh
```

Docker Images and Containers

An image is a collection of bin/lib that are necessary for an application to run. All the docker images are present in the cloud site of docker called hub.docker.com

A Container is a running instance of the image. It is the indivdual application or process that docker has created in our user space

Docker Components

Dcoker Host*:* This is the machine where docker is installed and all the docker images are downloaded.

Docker Client: This is the terminal of docker where we can fire the docker commands

Docker Deamon: This is a background preocess which takes the commands fired by docker client and sends them to the docker images or contianers or docker registry

Docker Registry*:* This is the cloud site of docker where all the docker images are present. hub.docker.com

We can also create a private registry whcih can be accessed only by our organisation

Important docker commands

Docker Image commands

1 To downlaod an image

docker pull image_name

2To see the list of all images in our docker host

docker images

or

docker image ls

3To search for an image on hub.docker.com

docker search image_name

4To delete an image from our docker host

docker rmi image_name

5To upload a docker image

docker push imagename

6 To create an image from a container

docker commit container_id/container_name new_image_name

7 To create an image from a docker file

docker build -t new_iamge_name path_of_dockerfile

Docker Container commands

8To see the list of running containers

docker container ls

9To see the list of all containers(running and stopped)

docker ps -a

10 To find detailed info about a container

docker inspect container_name/container_id

11 To see the logs generated by a container

docker logs container_name/container_id

12 To start a stopped container

docker start container_name/container_id

13 To stop a running container

docker stop container_name/container_id

14 To restart a container

docker restart container_name/container_id

To restart after 10 seconds

docker restart -t 10 container_name/container_id

15 To remove a stopped container

docker rm container_name/container_id

16 To remove a running container

docker rm -f container_name/container_id

17 To stop all running containers

docker stop $(docker ps -aq)

18 To remove all stopped containers

docker rm $(docker ps -aq)

19 To remove all containers(running and stopped)

docker rm -f $(docker ps -aq)

20 To create a container from an image

docker run image_name

Run command options

-it Used for opening an interactive terminal in the container where we can fire linux commands

-d Runs the container as a deamon ie it runs the container in the background

--name Used for assigning a name to a container

-p Used for port mapping.It will link the container port with the docker host port.Container port is called internal port and docker host port is called external port

Eg: -p 8080:80

Here 8080 will be docker host port and 80 will be container port

-P Used for automatic port mapping ie the internal port of container will be mapped with a port on docker host and this external port will be greater than 30000

-v Used for mounting volumes on a container

--volumes-from Used for creating reusable volumes that can be shared between containers

--network Used for assigning a network to a container

-e Used for assigning environment variables

-rm Used for removing the container on exit

--mem Used for allocating memory to containers

--cpu Used for allocating fixed amount of cpu to containers

21 To see the ports used by a container

docker port container_name/container_id

Docker Networking commands

22 To see the list of docker networks

docker network ls

23 To create a new network

docker network create networkname

24 To find detailed info about a network

docker network inspect network_name/network_id

25 To delete a network

docker network rm network_name/network_id

26 To attach a network to a running container

docker network connect network_name/network_id container_name/container_id

27 To disconnect a network from a running container

docker network disconnect network_name/network_id container_name/container_id

28 To run a particular command in a running container

docker exec -it container_name/container_id command_to_run

Eg: To open bash terminal in an already running contianer

docker exec -it container_name/container_id bash

29 To comeout of a container without exit

ctrl+p,ctrl+q

30 To go into a container which is already runing in background

docker attach container_name/container_id

Docker volume commands

31 To see the list of docker volumes

docker volume ls

32 To create a new volume

docker volume create volume_name

33 To remove a volume

docker volume rm volume_name/volume_id

Creating and using a containers

A container is a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another

Creating containers by using fallow command:

Docker run <options> baseimage

Options:

--name, -it, -d, -p, -P, -e….etc

UseCase1:

Start an nginx container in detached mode and name it webserver. Map the external port of docker host 8080 with internal port 80 of container

1Start nginx as a container

dcoker run -d -p 8080:80 --name webserver nginx

2To access the nginx

Launch any browser

UseCase2:

Start a tomcat container and publish the port numbers,and git it a name mytomcat

1 Start tomcat

docker run --name mytomcat -d -P tomcat

2 To find the ports of tomcat

docker port mytomcat

3 To see the home page of tomcat

Launch any browser

ipaddrsss-of-dockerhost:32768 (external port displayed in the above command)

UseCase 3:

Start an ubuntu container and go into its interactive terminal. Fire some linux commands in it

1 Start ubuntu as a container

docker run --name myubuntu -it ubuntu

2 In the container we can run linux commands

3 To come out of the container

exit

UseCase4:

Start mysql as a container. Go into its interactive terminal and login into the database,Create some sql tables

1 Start mysql as a container

docker run -d --name itpoint-mysql -e MYSQL_ROOT_PASSWORD=itpoint mysql

2 To open interactive terminal in the already runnign container

docker exec -it itpoint-mysql bash

In the container to login into the db

mysql -u root -p

Enter password

3To see the list of default availabel databases

show databases;

4To switch into any of the above databases

use dbname;

5To create some tables in this db

Open

https://justinsomnia.org/2009/04/the-emp-and-dept- tables-for-mysql/

Copy the code for creating emp and depttables

Paste in the mysql container

6 select * from emp;

select * from dept;

Customising Docker images:

We can create our own docker images and save them in hub.docker.com or in our private docker registry

This can be done in two ways

1 Create a container customised according to our requirment and save it as a image(snapshot)

2 Use dockerfile

UseCase5:

Create a ubuntu container and install git in it

Save that container as an image(snpashot)

Start a container frm this new image and we should find git

installed on it

1 Start ubuntu as a contianer

docker run -it --name myubuntu ubuntu

2 In the container

apt-get update

apt-get install -y git

exit

3 Save the container as an image

docker commit containerid/containername new-imagename

Eg. docker commit myubuntu ubuntu1

4Start a container from the above image and we will find

git preinstalled on it

docker run -it ubuntu1

In the container

git --version

Docker Volumes

Since docker containers are for temporary usage, once the container is removed all the data stored in the container will be lost.To preserve the data we can use docker volumes

Docker volumes are of two types

1 Simple docker volumes

2 Docker volume containers

Simple Docker Volumes:

These volumes can be used by only one docker container

UseCase6:

Create a empty folder /itpoint.

Start an ubuntu container and mount /itpoint folder on it

Create some files in /itpoint in the contianer

Stop and remove the container,we should still be able to access the files

1Create a directory

mkdir /itpoint

2Create a ubuntu container and mount /itpoint on it

docker run --name myubuntu -it -v /itpoint ubuntu

In the container

cd itpoint

touch file1 file2 file3

exit

3To find the location where the mounted data is stored

docker inspect myubuntu

In the JSON output that is generated search for "Mounts"

Copy the "Source" path

4Stop and remove the container

docker stop myubuntu

docker rm myubuntu

5To access the files created in the container

cd path-of-source-folder

Docker volume containers:

Share the volume or space from one container to another container as like shared memory

UseCase7:

Create a directory /itpoint and mount it on centos container c1. Create some files in the itpoint folder in the c1 container

Create another 2 container c2 and c3

These two containers should use the voulmes that are used by c1

1 Create a directory

mkdir /itpoint

2 Start centos as a container

docker run -it --name c1 -v /itpoint centos

In the c1 container

cd itpoint

touch file1 file2

Come out of the container without exit(Ctrl+p,Ctrl+q)

3Start another centos container c2 and this should use the volumes used by c1

docker run -it --name c2 --volumes-from c1 centos

In the c2 container

cd itpoint

touch file3 file4

Come out of the container without exit(Ctrl+p,Ctrl+q)

4Start another centos container c3 and this should use the volumes used by c2

docker run -it --name c3 --volumes-from c2 centos

In the c3 container

cd itpoint

touch file5 file6

Come out of the container without exit(Ctrl+p,Ctrl+q)

5Go into any of the three containers and we will see all the files created by all the three contaieners

docker attach c1

cd itpoint

ls

Linking of Docker containers

This can be done in three ways

1 --link option

2 Docker compose

3 Docker networking

--link option:

This is a depricated way of creating contianer architecture

We can link multiple docker containers and passdata between them

UseCase 8:

Start 2 busybox contianers and link them with each other

1 Start a busybox container name it c1

docker run -it --name c1 busybox

Come out of the contianer without exit(ctrl+p,Ctrl+q)

2 Start another busybox container and link with first container

docker run -it --name c2 --link c1:c1alias busybox

In c2 container

ping c1 (It should ping)

Creating Development environment with docker

Use case9:

Create a mysql container and link it with wordpress container.Once the linking is done a developer should be able to create a wordpress website.

1 Start mysql as a container

docker run -d --name itpoint-mysql -e MYSQL_ROOT_PASSWORD=itpoint mysql:5

2 Start a wordpress container and link it with mysql

docker run -d --name itpoint-wordress –p 8080:80 --link itpoint-mysql:mysql worpdress

3 To create a wordpress website

Launch any browser

ipaddress-of-dockerhost:8080

UseCase 10:

Create a setup for implementing ci-cd through jenkins

1 Start jenkins as a container

docker run -d --name jenkinsserver -p 5050:8080 jenkins

To see the home page of jenkins

Launch any browser

ipaddress-of-dockerhost:5050

2 Start tomcat as qa server and link with jenkins

docker run -d --name qaserver –link jenkinsserver:jenkins -p 6060:8080 tomcat

3 Start tomcat as prod server and link with jenkins

docker run -d --name prodserver –link jenkinsserver:jenkins -p 7070:8080 tomcat

to see the homepage of tomcat (qaserver and prodserver)

ipaddress-of-dockerhost:6060

ipaddress-of-dockerhost:7070

UseCase11:

Start a jenkins container and link it with an ubuntu contianer.Make this ubuntu as a slave

1 Start jenkins as a contianer

docker run -d --name jenkinsserver -p 8080:8080 jenkins

2 To see the dashboard of jenkins

Launch any browser

ipaddress-of-dockerhost:8080

3Capture the ipaddress of jenkins container

docker inspect jenkinsserver

4 Start an ubuntu container

docker run --name slave -it --link jenkinsserver:jenkins ubuntu

In the ubuntu slave container

apt-get update

apt-get isntall -y wget

5 Download slave.jar in the slave ubuntu container

wget http://ipaddress-of-jenkinsserver:8080/jnlpJars/slave.jar

Creating QA environment

Usecase12

Start a selenium hub container

Start 2 selenium node containers and link them with the selenium hub container

Run the selenium testing programs

1 Start selenium hub container

```
docker run --name hub -P -d selenium/hub
```

2 Start a node which has firefox installed on it and link with hub container

```
docker run -d --name firefox --link hub:selenium -P -e HUB_HOST=hub selenium/node-firefox-debug
```

3 Start another node which has chrome installed on it and link with hub container

```
docker run -d --name chrome --link hub:selenium -P -e HUB_HOST=hub selenium/node-chrome-debug
```

4 To see the ports of hub,firefox and chrome nodes

```
docker ports hub
docker ports firefox
docker ports chrome
```

5The chrome node and firefox nodes are GUI nodes

We can see them by installing a s/w called vnc viewer

6 Install vnc viewer-->Enter ipaddress-of-dockerhost:port number of firefox or chrome container

7 Enter password as "secret"

Docker Compose

This is a feature of docker where we can create an architecture of multiple contianers according to the requirmenets of the devlopment team, Qa team etc

Docker compose uses yml script for performing these activites

All the containers that we want to start and link are given in the compose file in the form of services. All these services can be started or stopped using one single command

Install docker compose on linux:

1 Open https://docs.docker.com/compose/install/#install- compose

2 Copy the 2 commands and paste in lunux terminal

curl -L https://github.com/docker/compose/releases/download/1.21.0/docker-compose-$(uname -s)-$(uname -m) -o /usr/local/bin/docker-compose

chmod +x /usr/local/bin/docker-compose

3 To check the version of dockercompose

docker-compose --version

Sample.yml file:

```
---
itpoint:
 - trainers:
    - sai:
       - Course: Devops
       - fee: 12000
    - sheshi:
```

```
        - Course: AWS
        - fee: 10000
-     coordinator:
-       shailaja
...
```

UseCase 13:

Create a docker compose file for linking a wordpress contianer with mysql contianer

vim docker-compose.yml

```
---
version: '3'
services:
 itpoint-mysql:
  image: mysql:5
  environment:
   MYSQL_ROOT_PASSWORD: itpoint

 itpoint-wordpress:
  image: wordpress
  ports:
   - 5050:80
...
```

Save and quit(:wq)

To start all the services in the compose file

docker-compose up -d

To stop the service

docker-compose stop

UseCase 14:

Create docker compose file for starting a jenkins contianer and link with two tomcat containers so that we can implement CI-CD

vim docker-compose.yml

```
version: '3'

services:

 jenkinsserver:

 image: jenkins

 ports:

  - 6600:8080

qaserver:

 image: tomcat

 ports:

  - 7700:8080

 links:

  - jenkinsserver
```

```
 prodserver:
  image: tomcat
  ports:
   - 8800:8080
  links:
   - jenkinsserver
```

Save and quit

docker-compose up -d

To see the home page of jenkins

Launch any browser

ipaddress-of-dockerhost:6600

To see the home page of qa and prodserver

Launch any browser

ipaddress-of-docker-host:7700

ipaddress-of-docker-host:8800

UseCase15:

Create a docker compose file for setup of the QA environment.Create seelnium hub container and link it with two node contianers

vim docker-compose.yml

```
version: '3'
```

```
services:
 hub:
  image: selenium/hub
  ports:
   - 4444:4444
 firefox:
  image: selenium/node-firefox-debug
  ports:
   - 5901:5900
  environment:
  HUB_HOST: hub
  links:
   - hub
 chrome:
  image: selenium/node-chrome-debug
  ports:
   - 5902:5900
  environment:
  HUB_HOST: hub
  links:
   - hub
```

Dockerfile

This is a simple text file which is used for creating dockerimages. It uses a syntax called GO which has certain predefined keywords

Important Keywords in Dockerfile

FROM This is generally the first statement in dockerfile and it is used to specify the base image from which we are creating out dockerimages

MAINTAINER: This is the name of the person or the organisation that has created this dockerfile

CMD: This is the default instruction that gets executed when the container is started

ENTRYPOINT: This is used to pass arguments to the CMD instruction and it can also be used to specify the default process that should be initiated when a container starts

RUN: This is used for package management ie installing or uninstalling s/w using apt or yum package managements

ADD: This is used to copy files from the host machine to the docker container or to download some files from a remote url

EXPOSE: This is used for specifing what internalport of the container has to exposed to the outside world

USER: This is used to specify the default user credentials through which we want to login into a container

ENV: This is used to pass environment variables to the container

WORKINGDIR: This is used to specify the directory where the CMD instruction should be executed

VOLUME: Used for attaching volumes to the docker container. This will become the default volume for the container

Create dockerfile from ubuntu base image, specify the maintainer and give the default instruction to execute

vim dockerfile

FROM ubuntu

MAINTAINER itpoint

CMD ["date"]

Save and quit

To create an image from thw above docker file

docker build -t myubuntu .

Note -t reperesents tagname ie some name for the image

that we are creating

. represents current working dir ie it will read the dockerfile present in the current working dir and create an image

Create a dockerfile from ubuntu base image and make the default process as reading /etc/passwd

vim dockerfile

FROM ubuntu

MAINTAINER itpoint

ENTRYPOINT ["/bin/cat"]

CMD ["/etc/passwd"]

Save and quit

To create an image from the above docker file

docker build -t myubuntu .

Create a dockerfile from centos baseimage and install git in it

vim dockerfile

FROM centos

MAINTAINER itpoint

RUN yum -y install git

Save and quit

Build an image from this dockerfile

docker build -t newcentos .

Start a container from the above image

docker run -it --name centos newcentos

In the centos container

git --version

CacheBusting

Whenever we build an image from a dockerfile

Docker will read the cache(memory) to check which instructions we executed previously. It will not execute the commands that are already executed but reads the info from the cache. Only the new instructions will be executed

Example

FROM ubunut

RUN apt-get update

RUN apt-get install -y git

The above docker file updates the apt repository and installs git on it.Now after a period of time if we update the apt repository by adding some more statements like

RUN apt-get install -y maven tree

Only the above new instruction gets executed. This might result in installing maven and tree from a repository that was update long back.To overcome this problem we can use cache busting uisng &&

Create a dockerfile for installing packages and updating the apt repository using cache busting

vim dockerfile

FROM ubuntu

MAINTAINER itpoint

RUN apt-get update && apt-get install -y git maven

Save and quit

Create a dockerfile from jenkins baseimage and make the default user as root and also install git and maven

vim dockerfile

FROM jenkins

MAINTAINER itpoint

USER root

RUN apt-get update && apt-get install -y git maven

Save and quit

Build an image from the above docker file

docker build -t newjenkins .

Start this as a contaier and we will see that root is the default user and git and maven are installed

docker run -d --name jenkins -P newjenkins

To open termina in the jenkins container

docker exec -it jenkins bash

Check the user

whoami

Check if git and maven are present

git --version

mvn --version

Create a dockerfile from ubuntu baseimage and downloa d jenkins.war into it

vim dockerfile

FROM ubuntu

MAINTAINER itpoint

ADD http://mirrors.jenkins.io/war-stable/latest/jenkins.war /root

Save and quit

Build an image from the above docker file

docker buil -t newubuntu .

Start a container

docker run -it newubuntu

cd root

We will find jenkins.war file downloaded here

Create a dcokerfile from jenkins baseimage and make bash as the default process ie it we start this contianer it should automatically go into the terminal of the container

1vim dockerfile

FROM jenkins

MAINTAINER itpoint

ENTRYPOINT ["/bin/bash"]

Save and quit

2Build an image from this file

docker build -t jenkins2 .

3Start a container using the above image

docker run -it jenkins2

We will directly go into the terminal of the container

Working on docker registry

Docker registry is of two types public and private

Public registry:

Public registry is hub.docker.com and this is maintianed by docker community. To upload images in this public registry we should create an account in the hub.docker.com

1 Open hub.docker.com

2 Signup for a free account

3 We will get a dockerid and password

Create a customised centos image and upload into dockerhub

1Start centos as a contianer

docker run -it --name mycentos centos

In the centos container install httpd

yum -y install httpd

exit

2Save the container as an image

docker commit mycentos itpointit/httpd-centos

3Login into docker account

docker login

Enter username and password

4To push the image into docker hub

docker push itpointit/httpd-centos

Note: do you want to push to docker registry the image must be taged with account name/id..(ex. itpoint/httpd-centos)

Private Registry:

This is a local docker registry that is maintained by individual organisations and only the organization team memebers can access it

This local registry can be create by using a docker image called "registry"

1 Start registry image as a container

docker run -d -p 5000:5000 --name localregistry registry

UseCase:

Download a busybox contianer and push into docker local registry

1 docker pull busybox

2Tag that image(ie we should first link it with local registry)

docker tag busybox localhost:5000/busybox

3 Push into localregistry

docker push localhost:5000/busybox

Docker Networking

Docker supports the following 4 networks

2 Hostonly

3 None or Null

4 Overlay

For containers to communincate with each other or with the outside world we can use docker networks

Bridge Network*:* Bridge network is the default network of docker containers that are running on a single docker host

Host network: Host only network is used to create container that can communicate only with the host machine

None or Null Network: None network is used for creating isolated contianers ie these containers cannot communicate with the host machine or with other containers

Overlay network: Overlay network is used when contianers are running in a distributed network ie multiple containers are running on multiple docker hosts and all these containers are communicate with each other(eg docker swarm,docker stack etc)

UseCase:

Create 2 networks itpoint1 and itpoint2

Create 3 busybox containers c1 c2 and c3

c1 and c2 should be on itpoint1 network so they will be ablt to ping to each other

c3 shoudl be on itpoint2 network it should not be able to ping to c1 or c2

Now assign c2 to itpoint2 network ie c2 is present on both itpoint1 and itpoint2 network

Beacuse c2 is present on both the networks it should ping to both c1 nad c3 but c1 and c3 should not be able to communicate with each other

1 Create 2 networks

docker network create itpoint1

docker netwrok create itpoint2

2 Start a busybox container c1 on itpoint1 network

docker run -it --name c1 --network itpoint1 busybox

Come out of c1 container without exit(ctrl+p,Ctrl+q)

3 Identify the ipaddress of c1

docker inspect c1

4 Start another busybox container c2 on itpoint1 network

docker run -it --name c2 --network itpoint1 busybox

In the c2 container

ping ipaddress-of-c1 (it should ping)

Come out of c2 container without exit(ctrl+p,Ctrl+q)

5 Identify the ipaddress of c2

docker inspect c2

6 Start another busybox container c3 on itpoint2 network

docker run -it --name c3 --network itpoint2 busybox

In the c3 container

ping ipaddress-of-c1 (it should not ping)

ping ipaddress-of-c2 (it should not ping)

Come out of c3 container without exit(ctrl+p,Ctrl+q)

7 Connect c2 to itpoint2 network

docker network connect itpoint2 c2

8 Since c2 is running on both itpoint1 and itpoint2 networks it will ping to both c1 and c3

docker attach c2

ping ipaddress-of-c1(it should ping)

ping ipaddress-of-c3(it should ping)

Come out of c2 container without exit(ctrl+p,Ctrl+q)

9 Go into c1 container

docker attach c1

ping ipaddress-of-c3(it should not ping)

Docker Swarm

This is a tool of docker for performing container orchestration.

When we want to run multiple contianers in a distributed network we want a single service to run on all of them we can used docker swarm

Advantages:

Useful in handling loadbalancing

Can be used in failover scenarios

Autoscalling of contianers

Performing rolling update

Performing health checks

Setup of docker swarm

Create 3 vm's through vagrant or on the AWS cloud

Install docker on all of them

If the instances are created on AWS cloud

Open the below ports

TCP port 2377 for cluster management communications

TCP and UDP port 7946 for communication among nodes

UDP port 4789 for overlay network traffic

The main machine where we create the swarm is called as manager and the rest of machines are called workers

To initilize the swarm

1 Go to the manager machine

docker swarm init --advertise-addr ip-address-of-manager

This machine now becomes the manager and it also displays the command that should be used for workers to join swarm

Copy and paste that command in the worker machines

2 To see the list of machines in the swarm

docker node ls

1To create services in docker swarm

docker swarm create imagename

Create an nginx service with 5 replicas and name it webserver

docker service create --name webserver --replicas 5 -p 80:80 nginx

To see the list of serivices runnign on docker swarm

docker service ls

To see the list of nodes on which the webserver service is running

docker service ps webserver

Start mysql as a service in swarm with 3 replicas

docker service create --name mysql --replicas 3 -e MYSQL_ROOT_PASSWORD=itpoint mysql

To see the list of nodes on which this mysql service is running

docker service ps mysql

To see detailed info about any service running in swarm

docker service inspect servciename

Eg

docker service inspect mysql

The above command will show the output in JSON file format

To see the output in simple(pretty)format

docker service inspect mysql --pretty

Scalling of services in swarm

We can increase or decrease the number of replicas on which a service is running based on the business requirement.This can be done in swarm without having any downtime

Scale up the mysql service from 3 replicas to 6 replicas

docker service scale mysql=6

To see the list of nodes on which these 6 mysql replicas are running

docker service ps mysql

Performing rolling updates

Swarm can update any service running without having any downtime. It achives this task by performing the updates in a rolling manner ie it will first update on one replica and once the update is totally done on it it will move to another replica.

Create a service of redis:3 in docker swarm with 3 replicas

Perfrom rolling update to redis:4 and later roll back to redis:3

1 Start redis:3 with 3 replicas

docker service create --name mydb --replicas 3 redis:3

2To see the list of nodes on which this service is running

docker service ps mydb

3 Update redis:3 to redis:4

docker service update --image redis:4 mydb

If we run "docker service ps mydb" we will see that all the containers with redis:3 are shut down and in its place redis:4 containers are running

4To roll back to previous versions

docker service update --rollback mydb

If we run "docker service ps mydb" we will see that all the containers with redis:4 are shut down and in its place redis:3 containers are running

To drain a worker from the swarm

docker node update --availability drain worker1

To reactivate the drained worker into the swarm

For the worker to leave the swarm

Go to the corresponding worker machine

docker swarm leave

For the manager to leave the swarm

docker swarm leave --force

Handling failover scenarios

Start tomcat with 5 replicas in the swarm. Drain a worker from swarm and check if all the 5 replicas of tomcat are running or not

1 Start tomcat as a service

docker service create --name tomcat --replicas 5 -p 8080:8080 tomcat

2Check if all 5 replicas are distributed on the manager and workers

docker service ps tomcat

3 Drain a worker from sswarm

docker node update --availablity drain worker1

4Check the status of worker1..it should be in drained status

docker node ls

5We will still see all the 5 replicas of tomcat running on manager and other workers

docker service ps tomcat

To generate the tokenid to join the swarm as worker

docker swarm join-token worker

To generate the tokenid to join swarm as manager

docker swarm join-token manager

To promote a worker as manager

docker node promote worker1

To demote a manager and make him as worker

docker node demote worker1

Bydefault swarm uses overlay network ie all the services that we start in swarm will be using overlay network and they can communicate with each other.

If we want different services in swarm to run on different overlay networks we can create those overlay networks and assign them to the services in swarm

Sceanrios

Create 2 overlay networks itpoint1 and intalliq2

Start nginx on itpoint1 network

Assign itpoint2 network to already running tomcat

1 Create 2 overlay networks

docker network create --driver overlay itpoint1

docker network create --driver overlay itpoint2

2 Start niginx in swarm on itpoint1 network

docker service create --replicas 3 --network itpoint1 -p 8080:80 nginx

3 Start tomcat in swarm and later assign to intelliq2 network

docker service create --replicas 3 -p 8888:8080 tomcat

docker service update --network-add itpoint2 tomcat

Similarly to remove a network from a service

docker service update --network-rm itpoint2 tomcat

Docker Stack

A stack is a group of interrelated services that share dependencies with each other and they can be scaled or orchestrated together.

A single stack is capable of defining the functionality of a complete application

To deplay a stack in swarm

docker stack deploy -c docker-composefile-name

To see the list of stack

docker stack ls

To see the list of task in a stack

docker stack ps stackname

To remove a stack

docker stack rm stackname

To remove multiple stacks

docker stack rm stack1 stack2

To see the list of servcies in the stack

docker stack services

Kubernetes

This is the container archestration tool similar to docker swarm. By using this kubernets we can handled scenario like load balancing, scaling, failover scenarios, rolling updates ....etc.

Node:

This is machine which is part of kubernets cluster which take care of Container archestration it can be master or slave.

Cluster:

This is the combination of multiple nodes. master machine is where the kubernet service is deployed.

Slave machine take load and perform container archestration

Master has **"kube-api"** server which is responsible for managing nodes

Slave has "**kubelet"** which accepts load from manager

Kubectl:

Kubectl is an application which is used running kubernets commands

Kubeadmin:

Kubeadmin is used for setting master and slave in the kubernets cluster.

Kubernets setup

1. Open https://labs.play-with-k8s.com/

2 Login using github credentials or dockerhub crendentials

3 click on add new instance (create 3 such instances)

One is for master and two is for slave

4 go to node master it is the node1

5 To initialize the kubernets on master

kubeadm init --apiserver-advertise-addr $(hostname -i)

6 To create cluster responsible for managing kubernets

Kubectl apply –n kube-system –f https://cloudweave.works/k8s/net2k85 -versin=$(kubectl version/base 64) \n');

7 copy output of the slave command and paste in slave machine

In kubernets containers are always present in Pod. A single Pod can run single instance of container and multiple instances of container

Pod is an application responsible to maintain your state

Pods are controlled by another application called controller

Controller is of two types

1 replication controller

2 replication set

Responsible for maintaing the kubernets api and controll behaviour of pots and container

Deployment

Deployement is the high level service which is responsible for maintaining the controllers, Pots and container

To see the list of nodes

Kubectl get nodes

Start nginx is deployement in kubernets cluster

Kubectl run webserver –image nginx

To see the Pod's running

Kubectl get pods

To see the pods along with the ip's of Pods

Kubectl get pods –o wide

Start tomcat in kubernets and perform port mapping

Kubernet run –-image tomcat mytom –port:8080 –hostport:9090

Start mysql container in kubernets

Kubernet run msql –-image mysql:5 --env MYSQL_ROOT_PASSWORD=itpoint

To open the interactive terminal

Kubernet exec –it podname bash

To see the logs

Kubectl log –f podname

To delete a Deployement
kubectl delete servicename

Creating pots in kubernets using YAML script

The YAML file contains four top level fields

- apiVersion
- kind
- metadata
- spec

apiVersion :

it is the version of pod object

kind:

it's represent kind of object that getting created. That is pod, controller, service..etc

metadata:

Data about object created with this YAML file is stored here. It is also contains information related to label

This label can be used for filtering and serching for specific pods are deployement

Spec:

Kind	**:**	**version**
Pod	:	v1
Service	:	v1
Replicaset	:	apps/v1
Deployement	:	apps/v1

Create a pod-definitionfile for deploying httpd in kubernets cluster

Vim pod-definitionfile.yml

```
apiVersion: v1
kind: pod
metadata:
  name: myapp-pod
  labels:
    app: myapp
    type: front-end
```

```
spec:
  containers:
        - Name: httpd-containers
          Image: httpd
```

To deploy above pot file

Kubectl create –f poddefinitionfile.yml

To delete the services deployed to above file

Kubectl delete –f p-oddefinition.yml

Create pod-definitionfile which uses postgress image and starts container with name called myapp-postgress

Vim pod-definition.yml

```
apiVersion: v1
kind: pod
metadata:
  name: myapp-postgress
  labels:
    app; helloapp
    tier:  db-tier
spec:
  containers:
        - Name: mypost
```

Image: postgress

Create poddefinitionfile for starting mysqwl as container and container name mydb and pass environment variable and pod name is myapp-mysql

Apiversion: v1

Kind: pod

Metadata:

Names: dbname

Labels:

Spec:

Containers:

- Name: mydb
  Image: mysql
  Env:
  - name: MYSQL_ROOT_PASSWORD
    Value: itpoint

Run:

Kubectl created –f pod-definitionfile.yml

Start jenkins as container with in a pod the name of container should be myjenkins. Map container port 8080 and hostport 5050 and give name of pod as app-jenkins

Vim pod-definitionfile.yml

---

apiVersion: v1

```
kind: pod
metadata:
   name: app-jenkins
    labels:
      ci:cd
spec:
containers:
     -name: myjenkins
      Image: jenkins
      Ports:
        Containerport: 8080
         Hostport: 5050
```

Run:

Kubectl created –f pod-definitionfile.yml

Controllers

These are used for maintaining state of the ports and they can be implemented using replication controller and replicasets

Replication controller:

It is used for maintaining the number of instances in the kubernets cluster

Create Replication controller for tomcat

```
Apiversion: v1
Kind: ReplicationController
Metadata:
  Name: rc-tomcat
Spec:
  Containers:
    Replicas: 3
    Template:
      Metadata:
        Name: app-tomcat
        Labels:
          App: tomcat
      Spec:
      Containers:
        - Name: mytomcat
          Image: tomcat
          Ports:
           Containerport:  8080
           Hostport: 7070
```

Replicatin Set:

This is another form of controller which is similar to replication controller but it has the additional field called sector which is declared in **spec** section of the YAML file.

This helps replicatin controller acuire any pots with the same sectors and adds them to the kubernets cluster. It is also contains replicas section similar to the replication controller which tells how many replicas maintained at any point of time.

The replication set keeps on creating and deleting replicas to match the replica count

Whenever new pot as to be created the replication controller uses it's template which contains the container information

The link between it's replica set and it's pots is maintained by via it's meta data section

Replicaset ansures that any point of time specifies the number of replicas are running in the cluster.

Note:

Replicaset are generally not used in scenarios where update operations have to be done on the application.

Replicationset file:

Replicationset file for launching php redis based image with 3 replicas

Vim replicatinse.yml

```
---
apiVersion: apps/v1
kind: Replicaset
metadata:
  name: frontend
  labels:
    app: gestbook
```

```
    tier: frontend
spec:
  # modify replicas according to your case
  replicas: 3
  selector:
    matchlabels:
    tier: frontend
    template:
    metadata:
      labels:
        tier: frontend
     spec:
     containers:
            - Name:  php-redis
              Image: gcr-io/google_sample/gb_frontend:u3
…
```

Scaling the Replica count to bigger or smaller number

Cubectl scale --replicas=6 -f replicaset.yml

Deployement objects in kubernets

Controller objects like replicaset, replicationcontroller...etc are controlled by the deployement object. The deployement object can also contain a selector field like replicasets where it can capture the pots with the same selector and add them to the kubernets cluster.

It also contain the replica count which always maintain by the deployement by creating or deleting pots.

The main advantage of deployement over controller is deployements are dynamic where controller are statics. That is the application deployed via controllers can not be upgraded to newer version. Where as deployement can be upgraded.

Deployemetfile:

```
---

apiVersion: apps/v1

kind: Deployement

metadata:

  name: frontend

  labels:

    app: nginx

    tier: frontend

spec:

  # modify replicas according to your case

  replicas: 3

  selector:

    matchlabels:
```

```
    app:  nginx
  template:
  metadata:
    labels:
      app: nginx
    spec:
    containers:
        - name: nginx
          image: nginx:1.7.9
          ports:
        - ContainerPort:80
…
```

To create the cluster

Kubectl create -f deployement.yml

To update the existing version of nginx running in the cluster in some another version

Kubectl --record deployement apps/nginx -deployement set image deployement.v1 apps/nginx - deployement nginx = nginx:1.9.1

Configuration Management Tool (Ansible)

The goals of **Software Configuration Management** are generally Configuration, Identification, Configuration idioms and baselines, configuration control, implementing a control change process.
This is usually achieved by setting up a change control board whose primary function is to approve or reject all change request that is sent against any baseline. Configuration status accounting, reporting and recording all the necessary information on the status of the development process.

Software Configuration Management Features

Enforcement: With enforcement feature executing daily ensures that the system is configured to the desired state.

Cooperating Enablement: This feature helps to make the change configuration throughout the infrastructure with one change.

Version Control Friendly: With this feature, the user can take their choice of version for their work.

Enable Change Control Processes: As Software Configuration Management tools are version control and textual friendly we can make changes in code. Changes can be made as a merge request and send for review

Ansible is an open source configuration management and orchestration utility. It can automate the configuration of remote hosts are virtual machines using ansible it is possible to lunches or shutdown multi-tiered architecture due to this reason ansible is used for performing rolling updates with 0% downtime. Instead of creating shell scripts for management remote servers ansible uses **plays.** A play is collection of task that should be performed on remote host a file which contains these plays are called as ansible playbooks.

Ansible uses agent less architecture i.e. ansible pushes its tasks via ssh so it does not require any agent to be installed on the manage hosts.

Note:

Ansible requires python to be installed on all the machines

Controller: This is the centre on which we install ansible

Managed Hosts or managed Nodes: these are the remote machines which will be controlled by ansible.

Disadvantages of Ansible:

It cannot perform installation of the basic o/s.

It cannot monitor the changes done on the remote host.

Configure and Installation of Ansible

1 Create 3-5 linux vm's or aws instances

2 The main machine where ansible is installed is called controller the remaining remote servers that ansible configures are called manged hosts.

3 Establish passwordless ssh between the controller and the managed hosts

4 To install ansible

a) Open terminal in controller

b) Update the apt repository

sudo apt-get update

c) Install software-properties-common

sudo apt-get install -y software-properties-common

d) Add the latest version of ansible ppa to apt repository

sudo apt-add-repository ppa:ansible/ansible

e) Update the apt repository

sudo apt-get update

f) Install ansible

sudo apt-get install -y ansible

5 To check the version of ansible

ansible –version

Note: Install the same version of python in all managed and controller nodes

Ansible store all the remote managed hosts info in a file called as inventory file.

The default location of thsi file is

/etc/ansible/hosts

Open this file and add the ipaddress of all the managed hosts

sudo vim /etc/ansible/hosts

10.10.10.72

10.10.10.73

10.10.10.74

10.10.10.75

Save and quit

Modules in ansible

Ansible performs remote configuration of servers by using in built modules. These have been created using python. Each module is deigns for performing a specific task

Important modules in ansible

command : This is the default module of ansible and it is used to fire linux commands on the remot managed nodes

shell : This is used for running shell scripts and linux commands that involve redirection and piping

ping : This is used to check if a remote server is pingable or not

copy : Used for copying files from the controller to the managed nodes

fetch : Used to copying files from the managed nodes to the controller

file : Used for creating files or directories on the managed nodes

user : Used for usermanagement on the managed hosts ie creating,modifying and deleting users.

apt : Used for package management on the managed nodes ie installing,upgrading uninstalling etc.It works on ubuntu,debain based linux flavours.

yum : similar to apt but it works on redhatlinux,centos,fedoraOEL etc linux flavours.

git : This is used for perfroming git version controlling on the managed nodes

debug : This is used for displaying the output of a module

service : this is used to start,restart or stop services

uri : This is used to check if a rmeote url is reachable or not .

stat : this is used to check the info about files and directories on the managed nodes.

ec2 : This is used for creating ec2 instances on the aws cloud

docker_image : This is used for executing the commands of docker that are related to docker images

docker_container : This is used for container management on the managed nodes

docker_login : Used for loginin into hub.docker.com from the managed nodes.

setup: This is used to capture system facts that is system information.

Command module

Ansible command to see the memory info on all manged nodes

ansible all -m command -a 'free'

Ansible command to create an empty file on all the managed nodes

ansible all -m command -a 'touch file1'

Shell Module

ansible command to execute ls -la on managed nodes and save the output into a file called file1 on on the managed node

ansible all -m shell -a 'ls -la > file1'

Ansible command to install docker on all managed nodes

ansible all -m shell -a 'curl -fsSL https://get.docker.com -o get-docker.sh'

ansible all -m shell -a 'sh get-docker.sh'

Copy Module

ansible command to copy a file into all the managed node

ansible all -m copy -a 'src=myinventory dest=/tmp'

ansible command to copy a file and also change its ownership groupownership and premissions

ansible all -m copy -a 'src=myinventory dest=/tmp owner=Anu group=root mode=700' -b

Note: -b stands for become. It is used for giving higher previlages to execute a command on the managed nodes

File Module

This is used for creating files and directories on the managed nodes

ansible command to create a file in /tmp on all nodes

ansible all -m file -a 'name=/tmp/file state=touch'

Note: state=touch is used for creating files

state=directory is used for creating directories

state=absent is used for deleting files/directories

Ansible command to create a directory in /tmp folder

ansible all -m file -a 'name=/tmp/dir1 state=directory'

File module can also be used for changing the permissions of files and also the ownership and groupownership of files

ansible all -m file -a 'name=/tmp/file1 state=touch owner=Anu group=root mode=700' -b

apt module

This is used for package management on ubuntu based machines

To install tree on all the managed nodes

ansible all -m apt -a 'name=tree state=present' -b

Note: state=present is for install

state=absent is for uninstall

state=latest is for upgrading to latest version

To uninstall git from all managed nodes

ansible all -m apt -a 'name=git state=absent' -b

To install apache2 after updating the apt repository on all managed nodes

ansible all -m apt -a 'name=apache2 state=present update_cache=yes' -b

Service module

This is used for starting stopping and restarting services

Ansible command to stop tomcat7 on all managed nodes

ansible all -m service -a 'name=tomcat7 state=stopped' -b

state= strarted for starting the service

state=stopped for stopping the service

state=restarted for restarting the service

uri Module

This is used to check if a remote url is reachable or not

ansible command to check if google.com is reachable from all managed nodes

ansible all -m uri -a 'url=http://google.com status=200'

status=200 is pass status

status=-1 is failure status

Git module

This is used for git version controlling on the managed nodes

Install git on all managed nodes and download a remote git repository

ansible all -m apt -a 'repo=https://github.com/selenium-saikrishna/maven.git dest=/tmp/mygit'

Setup module

This is used to capture system facts that is system information.

$ ansible all –m setup

To find information about one variable we can give that variable name as on argument

Ext; to find the kernel on remote machine is running

$ ansible 192.168.60.101 –m steup –a ‘filter=facter_kernel’

Fetch module

To fetch a file from manage nodes into the controller machine we can use fetch module.

$ ansible all –m fetch –a ‘src=/home/vagrant/file1 dest=/home/vagrant’

Configure the managed nodes

Configuring the managed nodes in three defferent ways

1 Adhoc commands

2 Playbooks

3 Roles

Adhoc Commands

Syntax of adhoc commands

ansible all/groupname/ipaddress -i /etc/ansible/hosts -m command -a 'arguments'

-i is not manadatory when we use the default inventory file ie /etc/ansible/hosts

-m is not manadatory when we use the deafult module ie command module

Configuing apache2

Install apache2 on one mnaaged node and edit the content of its default index.html file,Restart apache2 and check if this is reachable from all other managed nodes

Install apache2 on one managed node

ansible 10.10.10.72 -m apt -a 'name=apache2 state=present cache_busting=yes' -b

Edit the index.html file

ansible 10.10.10.72 -m copy -a 'content="Hello Itpoint\n" dest=/var/www/html/index.html' -b

Restart the apache2 service

ansible 10.10.10.72 -m service -a 'name=apache2 state=restarted' -b

Check the url response

ansible all -m uri -a 'url=http://10.10.10.72 status=200'

Ansible playbook

The ansible commands that we have execute till now are called has ad-hoc commands and they are useful for only performing single operation and they can work only on a single setup arguments they cannot be used for complex configuration activities.

Ad-hoc commands can execute only one module at time to work on multiple modules we can use playbooks.

Playbooks are created in yamil format

Playbooks are powerful and flexible for performing cm

Using playbooks we can change lengthy and complex at administrative activities in to repeatable routines

Ansible playbooks are combination of plays each play defines set of operation that should be performed on manage nodes these operation are called as tasks and managed nodes called as hosts.

Each task execute specific modules the modules are executed in the order In which we present in the playbooks.

Ansible playbook for isntalling tree on all managed nodes

vim playbook1.yml

```
---
- name: Installing tree
  hosts: all
  tasks:
   - name: tree installation
     apt:
      name: tree
      state: present
   - name: git install
     apt:
      name: git
      state: present
     ...
```

Save and quit

To execute the playbook

ansible-playbook playbook1.yml -b

Ansible playbook for user creation

Vim plavybook2.yml

```
---
- name: User Creation
  hosts: all
  tasks:
   - name: Creating users
     user:
      name: Lakshmi
      password: itpoint
      home: /home/vagrant/Lakshmi
      shell: /bin/bash
...
```

Save and quit

To execute the playbook

ansible-playbook playbook2.yml -b

Ansible playbbok for craeting directory and copy passwd file to that dirctory on all managed nodes

```
---
```

```
- name: Creating dir and copying the passwd file
  hosts: all
  tasks:
   - name: Creating dir
     file:
      name: /tmp/newdir
      state: directory
   - name: Copying passwd file
     copy:
      src: /etc/passwd
      dest: /tmp/newdir/passwd
...
```

Ansible playbbok for installing git and download from github repository

```
---
- name: Install git and download a remote repo
  hosts: all
  tasks:
   - name: Install git
     apt:
      name: git
      state: present
   - name: Download the remote repo
     git:
```

```
    repo: https://github.com/selenium-saikrishna/maven.git
    dest: /tmp/git1
...
```

Create an ansible playbook for going to the manage nodes and fetching the all the users who are using /bin/bash shell

```
---
- name: for capturing user eiht /bin/bash/ shell
  hosts: all
  tasks:
   - name: Capturing info from /etc/password
     shell: grep /bin/bash /etc/passwd > file1
   - name: Fethcing files
     fetch:
      src: /home/vagrant/file1
      dest: /home/vagrant
....
```

Install apche2 on one manage node start apache2 service and check the url response.

```
---
- name: creating users and capturing username and home dir
  hosts: 192.168.60.2
  become: yes
  tasks:
   - name: update apt repo and install apache2
     apt:
```

```
    name: apache2
    state: present
    update_cache: yes
  - name: starting apache service
    service:
     name: apache2
     state: started
  - name: checking url response
    uri:
     url: http://192.168.60.101
     status: 200
...
```

Create a ansible playbook installing apache2 one more mcahine

```
---
- name: installing apache2
  hosts: 192.168.60.1
  tasks:
   - name: installing apache2
     apt:
      name: apache2
      state: present
- name: installing
  hosts: 192.168.60.2
```

```
 tasks:
  - name: installing git
    apt:
     name: git
     state: present
...
```

Note: the playbooks we have created till now are working only on group hosts that is the entire paly is getting executed on that group of hosts. To perform different activity's on different group hosts we can create multiple plays in playbook.

Variables

Variables are classified into 3 types

1) **Global scope**: these variables are defined from the command prompt and they can affect the complete playbook.

2) **Host scope**: these variables are defied with respect the hosts and it can affect all the plays with in the

3) **Play scope**: These are defined at the level of induvial play and they can effect only that particular play.

Global Scope Variables

Passing the global scope variables through playbook command execution by using –extra-vars keyword

Create an ansible playbook which can be used either for installing or uninstalling packages similarly it should also be used either for updating or non-updating the apt repository by using global scope variables

```
---
- name: installing/uninstalling packages
  hosts: all
  tasks:
   - name: installing/uninstalling packages
```

```
  apt:
   name: "{{a}}"
   state: "{{b}}"
   update_chache="{{c}}"
...
```

To run this playbook command prompt for installing maven without updating the repository:

$ ansible-playbook playbook12.yml --extra-vars “a=maven b=present c=no” -b

Similarly we can use the same the playbook for uninstalling git after updating apt repository.

$ansible-playbook playbook12.yml – extra-vars “a=git b=absent c=yes” –b

Create an ansible playbook for creating files or directories on the manage nodes and also for controlling the ownership groupowenership and permissions by passing values through global scope variable

Vim playbook13.yml

```
---
- name: creating files and directories
  hosts: all
  tasks: file creating/ dir creation and controlling ownership
  file:
   name: "{{a}}"
   state: "{{b}}"
   owner: "{{c}}"
   group: "{{d}}"
   mode: "{{e}}"
...
```

Vim playbook14.yml

```
- name: creating files and directories
  hosts: all
  tasks: file creating/ dir creation and controlling ownership
  file:
   name: "{{name}}"
   state: "{{state}}"
   owner: "{{owner}}"
   group: "{{group}}"
   mode: "{{mode}}"
...
```

$ ansible-playbook playbook13.yml - - extra-vars “a=file1 b=touch c=root d=sai e=111” -b

$ ansible-playbook playbook14.yml - - extra-vars “name=dir1 state=directory owner=vagrant group=sai mode=777” -b

using this play book we can create directory and controller the ownership and group ownership

Note: global scope have the highest presidency compare host scope and play scope.

Play scope variables

These variables can effect only that particular play and they should be defined with in the playbook

Create an ansible playbook for installing apache2 on all manage nodes after updating the apt repository

```
---
- name: package installation/uninstallation
```

```
 hosts: all
 vars:
  - a: apache2
  - b: present
  - c: yes
 tasks:
  - name: installing/uninstallation
    apt:
     name: "{{a}}"
     state: "{{b}}"
     update_chache: "{{c}}"
...
```

Note: if we execute above playbook the variable values arr taken from variable section in this playbbok but we overide these variable with global scope variables since blobal scope variable have high priority

Host Scope Variables

These are categorised into 2 types

1 Variables to work on a group of hosts

2 Variables to work on a single host

Variable to work on a group of hosts

1Change dir to the fodler where all playbooksare present

```
 cd path_of_folder_where_playbooks_are_present
```

2 Create a new dir "group_vars"

mkdir group_vars

3Create a file whose name should be similar tothe group name in our inventory file

vim webserver

```
---
a: Radha
b: itpoint
c: /home/vagrant/Radha
...
```

4Change dir back to the fodler where the playbooks are present

cd ..

5Create a playbook for user creation on the webserver group

vim playbook10.yml

```
---
- name: User Creation
  hosts: webserver
  tasks:
   - name: Creating users
     user:
      name: "{{a}}"
      password: "{{b}}"
      home: "{{c}}"
```

...

ansible-playbook playbook10.yml -b

Variables to work on a single host

1Change dir to the folder where all playbooks are present

cd path_of_playbooks_folder

2Create a folder "host_vars"

mkdir host_vars

3Change dir to host_vars folder

cd host_vars

4Create a file whose name is same as an ipaddress of one machine in the inventory file

vim 10.10.10.74

---

a: tree

b: present

c: no

...

5 vim playbook11.yml

```
- name: Installing tree
  hosts: 10.10.10.74
  tasks:
   - name: Install tree
     apt:
      name: "{{a}}"
      state: "{{b}}"
      update_cache: "{{c}}"
...
```

Variable Priorities

1 Global scope

2 Host Scope - single machine (host_vars)

2 Host Scope - Group of machines (group_vars)

4 Play scope varibles

Loops in ansible

Loops can be implemented in ansible in 2 ways

1 with_items

2 with_sequence

with_items is used to pass data to the module and depending on how many sets of data are present with_items will loop those many number of times

with_sequence is used to run a module specific number of times based on a count

ansible playbook to install multiple s/w packages

vim playbook12.yml

```
---
- name: Install s/w
  hosts: 10.10.10.74
  tasks:
   - name: Install s/w packages
     apt:
      name: "{{item}}"
      state: present
      update_cache:
     with_items:
      - tree
      - git
      - apache2
...
```

Playbook for creating multiple users and copying some files into the users home dir

```
---
```

```
- name: Creating user and copying files
  hosts: all
  tasks:
   - name: User Creation
     user:
      name: "{{item.a}}"
      password: "{{item.b}}"
      home: "{{item.c}}"
     with_items:
      - {a: Babu,b: itpoint,c: /home/Babu}
      - {a: Sravani,b: itpoint,c: /home/Sravani}
   - name: Copying files
     copy:
      src: "{{item.a}}"
      dest: "{{item.b}}"
      owner: "{{item.c}}"
      group: "{{item.d}}"
     with_items:
      - {a: /etc/passwd,b: /home/Babu,c: Babu,d: Sravani}
      - {a: /etc/group,b: /home/Sravani,c: Sravani,d: Babu}
...
```

Handlers:

Handlers are just like regular tasks in an Ansible playbook (see Tasks) but are only run if the Task contains a notify directive and also indicates that it changed something. For example, if a config file is changed, then the task referencing the config file templating operation may notify a service restart handler. This means services can be bounced only if they need to be restarted. Handlers can be used for things other than service restarts, but service restarts are the most common usage.

Example:

```
---
    - Name: Handlers example playbook
      Hosts: all
      Tasks:
       -name: install apache2
        Apt;
          Name: apache2
          State: present
          Update_cache: yes
         Notify: restart_apache2
      Handlers:
        -name: restart apache2
          Service:
            Name: apache2
            State: restarted
…
```

Note: Handlers are executed if any changes are done in calling tasks. We can mention Handlers in any order but they are executed in order specified in handler section

Error Handling

Blocks also introduce the ability to handle errors in a way similar to exceptions in most programming languages. Blocks only deal with 'failed' status of a task. A bad task definition, an undefined variable or an unreachable host are not *rescuable* errors.

Block error handling example¶

```
 tasks:
 - name: Handle the error
   block:
     - debug:
         msg: 'I execute normally'
     - name: i force a failure
       command: /bin/false
     - debug:
         msg: 'I never execute, due to the above task failing, :-('
   rescue:
     - debug:
         msg: 'I caught an error, can do stuff here to fix it, :-)'
```

This will 'revert' the failed status of the task for the run and the play will continue as if it had succeeded.

There is also an always section, that will run no matter what the task status is.

Block with always section

```
 - name: Always do X
   block:
     - debug:
         msg: 'I execute normally'
     - name: i force a failure
       command: /bin/false
     - debug:
         msg: 'I never execute :-('
   always:
     - debug:
```

```
      msg: "This always executes, :-)"
```

They can be added all together to do complex error handling.

Block with all sections¶

```
- name: Attempt and graceful roll back demo
  block:
    - debug:
        msg: 'I execute normally'
    - name: i force a failure
      command: /bin/false
    - debug:
        msg: 'I never execute, due to the above task failing, :-('
  rescue:
    - debug:
        msg: 'I caught an error'
    - name: i force a failure in middle of recovery! >:-)
      command: /bin/false
    - debug:
        msg: 'I also never execute :-('
  always:
    - debug:
        msg: "This always executes"
```

The tasks in the block would execute normally, if there is any error the rescue section would get executed with whatever you need to do to recover from the previous error. The always section runs no matter what previous error did or did not occur in the block and rescue sections. It should be noted that the play continues if a rescue section completes successfully as it 'erases' the error status (but not the reporting), this means it won't trigger max_fail_percentage nor any_errors_fatal configurations but will appear in theplaybook statistics.

Another example is how to run handlers after an error occurred :

Block run handlers in error handling¶

```
 tasks:
   - name: Attempt and graceful roll back demo
     block:
```

```
    - debug:
        msg: 'I execute normally'
      notify: run me even after an error
    - command: /bin/false
  rescue:
    - name: make sure all handlers run
      meta: flush_handlers
handlers:
  - name: run me even after an error
    debug:
      msg: 'This handler runs even on error'
```

Ansible Roles

Roles provide a framework for fully independent, or interdependent collections of variables, tasks, files, templates, and modules.

In Ansible, the role is the primary mechanism for breaking a playbook into multiple files. This simplifies writing **complex playbooks**, and it makes them easier to reuse. The breaking of playbook allows you to logically break the playbook into reusable components.

Each role is basically limited to a particular functionality or desired output, with all the necessary steps to provide that result either within that role itself or in other roles listed as dependencies.

Roles are not playbooks. Roles are small functionality which can be independently used but have to be used within playbooks. There is no way to directly execute a role. Roles have no explicit setting for which host the role will apply to.

Top-level playbooks are the bridge holding the hosts from your inventory file to roles that should be applied to those hosts.

Create and Configuring Roles

cd /etc/ansible/roles

ansible-galaxy init apache --offline

Note: role is created with apache after executing above command. Using tree see the role stracture(tree apache)

1 Creating tasks for apache2

a) cd apache/tasks

b) sudo vim main.yml

```
---
- include: install.yml
- include: configure.yml
- include: check_url_response.yml
...
```

Save and quit (:wq)

c) sudo vim install.yml

```
---
- name: Install apache2
  apt:
    name: apache2
    state: present
    update_cache: yes
```

...

Save and quit(:wq)

d) sudo vim configure.yml

```
---
- name: Send index.html file
  copy:
   src: index.html
   dest: /var/www/html/index.html
  notify:
   restart_apache2
...
```

Save and quit(:wq)

e) sudo vim check_url_response.yml

```
---
- name: check_url_reponse
  uri:
   url: http://10.10.10.74
   status: 200
...
```

Save and quit (:wq)

2Creating the static file that should be copied to remote

managed nodes

cd ..

cd files

sudo vim index.html

```
<html>
  <body>
    <h1>Itpoint</h1>
  </body>
</html>
```

Save and quit(:wq)

3Create the handlers

cd ..

cd handlers

sudo vim main.yml

```
---
- name: restart_apache2
  service:
    name: apache2
    state: restarted
```

...

Save and quit(:wq)

4Create a playbook to call this role

cd ..

cd ..

sudo vim configue_apache2.yml

```
---
 - name: Configuring apache2
   hosts: all
   roles:
    - apache
...
```

Save and quit(:wq)

5 Execute the playbook

ansible-playbook configure_apache2.yml -b

Nagios

Continious Monitoring

Continuous monitoring is a process to detect, report, respond all the attacks which occur in its infrastructure. Once the application is deployed into the server, the role of continuous monitoring comes in to play. The entire process is all about taking care of the company's infrastructure and respond appropriately.

Nagio is a free to use open source software tool for continuous monitoring. It helps you to monitor system, network, and infrastructure. It is used for continuous monitoring of systems, applications, service and business process in a DevOps culture.

Nagios runs plugins stored on the same server. It plugin's connects with a host or another server on your network or the Internet. Therefore, in the case of failure Nagios core can alert the technical staff about the issues. So that, your technical team performs the recovery process before outage in the business processes.

Features of Nagios

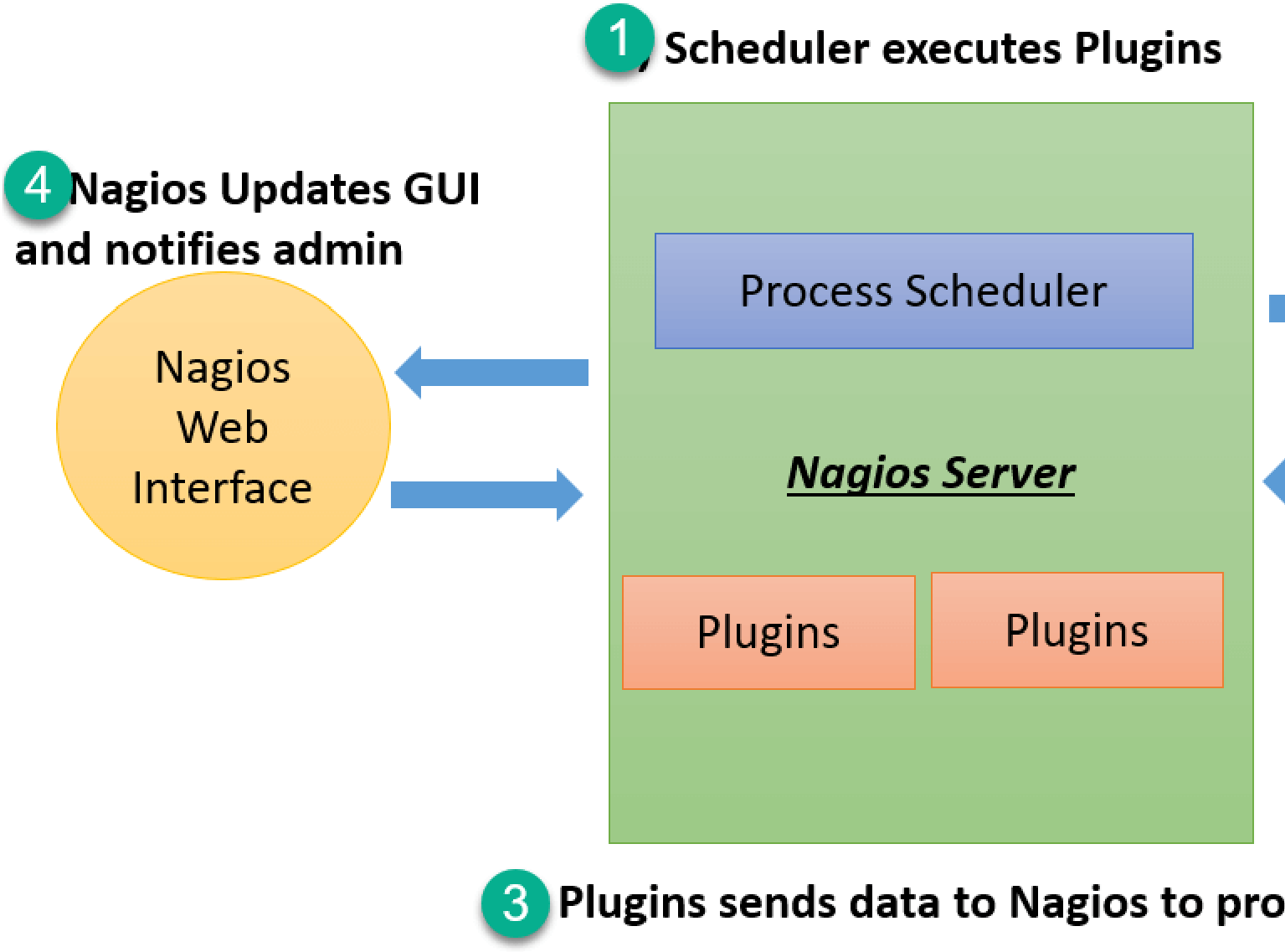
Following are the important features of Nagios:

- Relatively scalable, Manageable, and Secure
- Good log and database system
- Informative and attractive web interfaces
- Automatically send alerts if condition changes
- If the services are running fine, then there is no need to do check that host is an alive
- Helps you to detect network errors or server crashes
- You can troubleshoot the performance issues of the server.
- The issues, if any, can be fixed automatically as they are identified during the monitoring process
- You can monitor the entire business process and IT infrastructure with a single pass
- The product's architecture is easy writing new plugins in the language of your choice
- Nagios allows you to read its configuration from an entire directory which helps you to decide how to define individual files
- Utilizes topology to determine dependencies
- Monitor network services like HTTP, SMTP, HTTP, SNMP, FTP, SSH, POP, etc.
- Helps you to define network host hierarchy using parent hosts

- Ability to define event handlers which runs during service or host events for proactive problem resolution
- Support for implementing redundant monitoring hosts

Nagios Architecture

Nagios is a client-server architecture. Usually, on a network, a Nagios server is running on a host, and plugins are running on all the remote hosts which should be monitored.

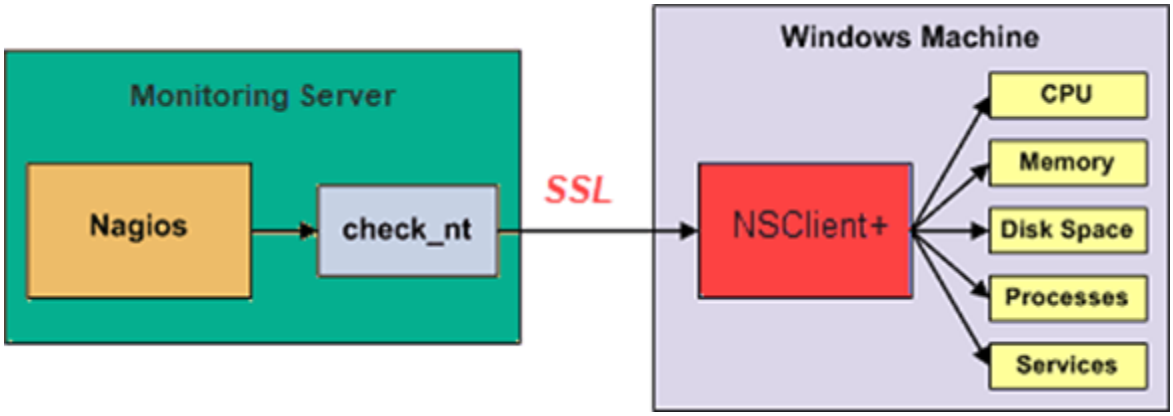


1. The scheduler is a component of server part of Nagios. It sends a signal to execute the plugins at the remote host.
2. The plugin gets the status from the remote host
3. The plugin sends the data to the process scheduler
4. The process scheduler updates the GUI and notifications are sent to admins

Plugins

Nagios plugins provide low-level intelligence on how to monitor anything and everything with Nagios Core. Plugins operate acts as a standalone application, but they are designed to be executed by Nagios Core. It connects to Apache that is controlled by CGI to display the result. Moreover, a database connected to Nagios to keep a log file.

How do plugins work?



Consider the above example-

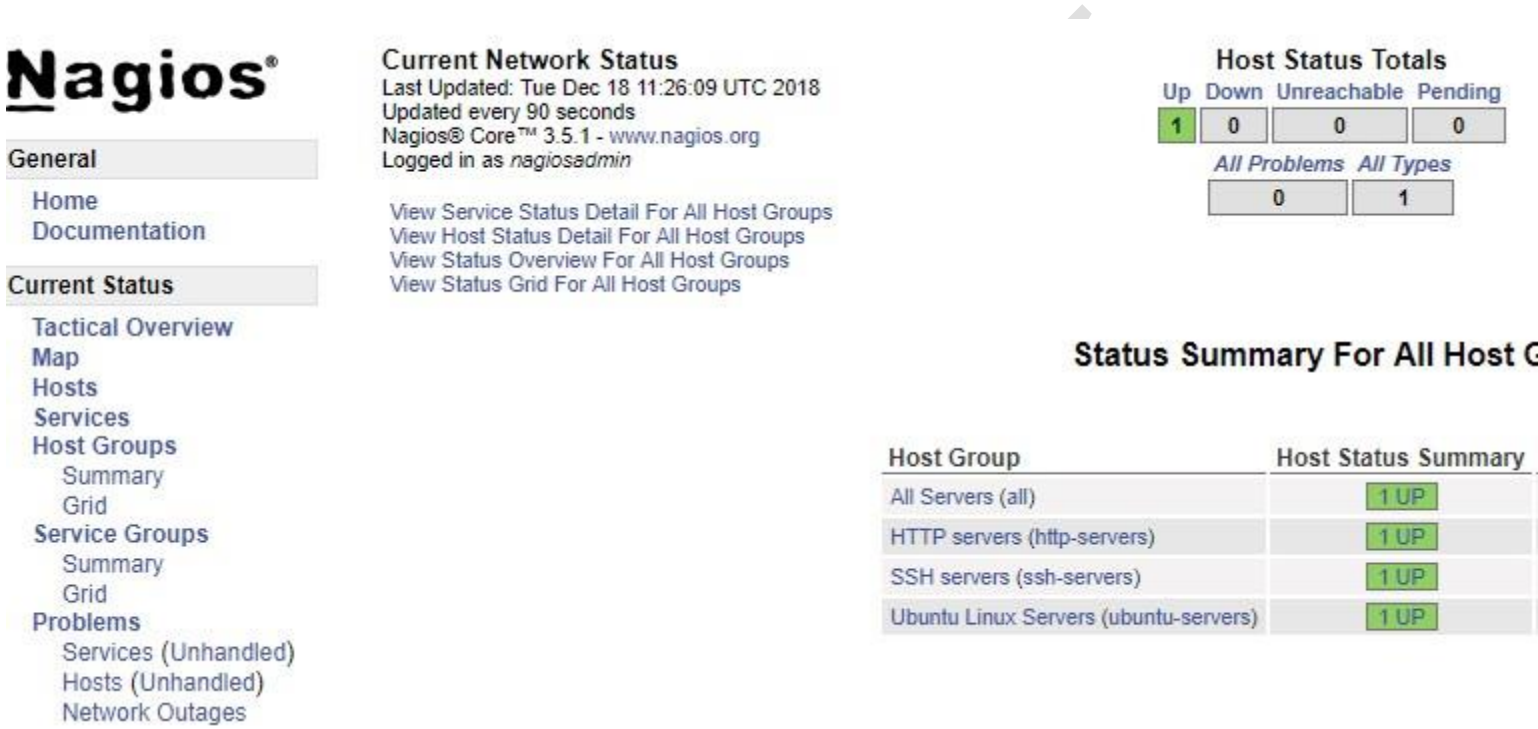
- Check_nt is a plugin to monitor a windows machine which is mostly available in the monitoring server
- NSClinet++ should be installed in every Windows machine that you wants to monitor
- There is an SSL connection between the server and the host which continuously exchange information with each other

Likewise, NRPE(Nagios Remote plug-in Executor) and NSCA plugins are used to monitor Linux and Mac OS X respectively.

GUI

An interface of Nagios is used to display in web pages generated by CGI. It can be buttons to green or red, sound, graph, etc.

When the soft alert is raised many times, a hard alert is raised, then the Nagios server sends a notification to the administrator.



Nagios Installation and Configuration

1 Launch two ec2 instances in aws one name is as nagios-server and another one is remote-server

2 open the concole of nagios server

3 sudo apt-get update

4 sudo apt-get upgrade

5 sudo apt-get install nagios3

Note: to see the home page of nagios (hhtp://nagios-server-public-ip/nagios3)

6 sudo vim /etc/nagios3/nagios.cfg

Search for extenal_commands and changed 0 to 1

7 sudo vim /etc/group

Search for nagios and add www.data at the end of statement

8 sudo chmod g+w /var/lib/nagios3/wr

9 sudo chmod g+w /var/lib/nagios3

10 sudo service apache2 restart

11 sudo service nagios3 restart

Configuring remote servers on nagios for monitoring

1 cd /etc/nagios.conf.d (this is the folder where all remote server configurations are present)

2 create new file (sudo vim remote-server) and write below nagios configuration code

```
define host
    {
 host_name remote-server-dns
 alias remote-server
 address remote-server-ip
  max_check_attempts 3
  check_period 24*7
  check_command check-host-alive
  contracts root
  notification_interval 60
  notification_period 24*7
       }
```

Note: the above code you found (http://www.theurbanpenguin.com/nagios-defining-a-new-host/)

3 To check if the above file is added to the list of configurations or not

Sudo nagios3 –v /etc/nagios3/nagios.cfg | less

4 sudo /etc/init.d/nagios3 restart

Active and passive checks

Nagios perform two kind of checks on remote server Active and passive

Active checks are initiated by nagios server and ther perform fixed frequency as specified in the nagios server

Passive checks are initiated by remote servers and they send notification to the nagios server

Note: chech the remote server status and logs by going hhtp://nagios-server-public-ip/nagios3

Agile scrum

Scrum is an incremental and iterative model which promises more return on investments.

In scrum the complete software is divided into modules each module is developed, tested and delivered to the client with in a time span of 2 to 4 weeks.

This is called as sprint. At the end of the sprint the one module of s/w to client in swapped ready state. Then it is client can start using this module.

This sprint is farther divided into smaller sub sprints as small sprint as 24hours

Continuous monitoring of the client will be present on all the sprints.

Scrum team

A typical scrum team contains the following people

Product owner, business owner - 1

Scrum master – 1

System testers 1 - 2

Developer – 1 - 2

Testers – 1 – 2

Devops admin 1 - 2

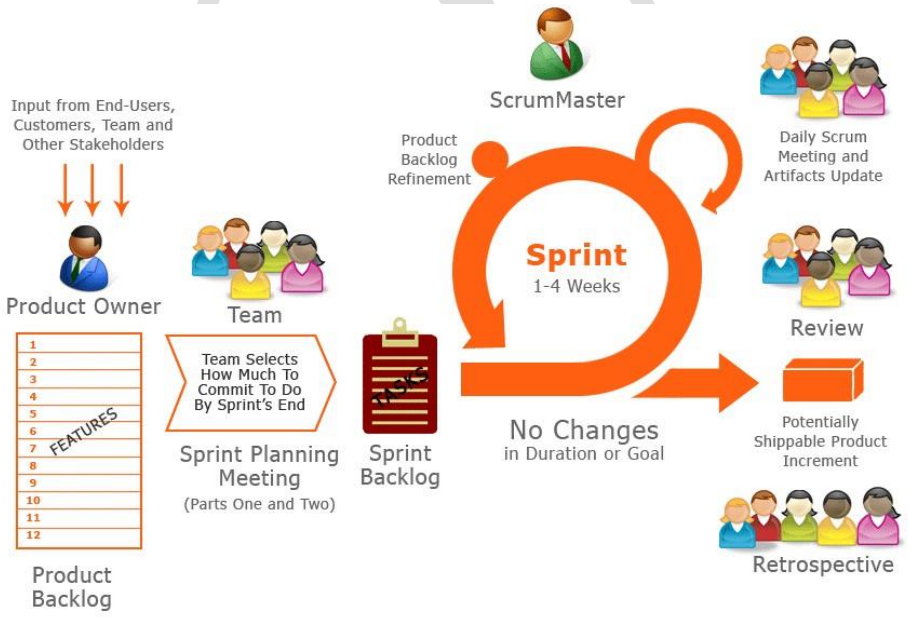
The entire scrum team discusser with product owner to understand his requirements and these requirements are created in the form of "user stories".

The collection of all these user stories is called as product Backlog"

Syntax of user story

As a [role] I want [features]

So, that [benefit to client]



Once the product backlog is finalised the scrum team prioritizes the user stories.

Highest priority user stories will go with first sprint and so on…

This priority can be done in **MoSCoW** principle.

Mo – must have

S – Should have

Co – couldn't have

W – Wouldn't have

Any user story with Mo priority is considered as highest priority and should be delivered in first sprint

User stories with s priority go with second sprint

User stories with Co priority go with 3rd sprint

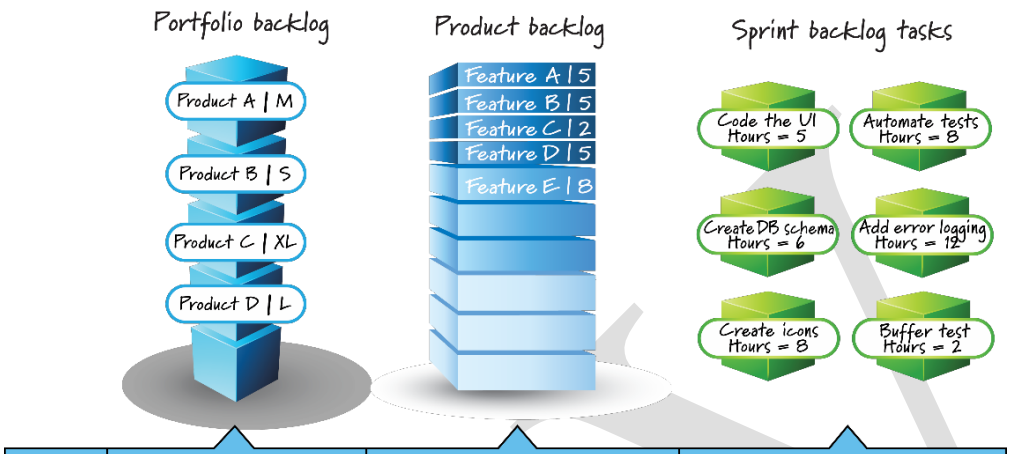
Wouldn't assign to user stories which are eliminated

Agile ceremonies

1) Sprint plan meeting: this is the one day meeting conducted by scrum master and here work allocation is done from all team members.
2) Scrum meeting or standard : this is conducted every day where is scrum team members should discourse with product owner what work done on yesterday and what we work plan today suggestions from the product owner should be implemented
3) Sprint retrospective: this meeting conducted is end of every sprint and here scrum team members will analyse the drawback they faced in previous sprint and suggest solution overcome.

Work estimation for a sprint

Work estimation is done using Fibonacci's numbers
1, 1, 2, 3, 5, 8, 21 …
Each user story assigned a Fibonacci number the smaller Fibonacci number means user story in less complex and it require in amount of time.
Bigger Fibonacci number assigned for a complex number.
Any user story which has Fibonacci number greater than 20 is consider as very complex user story and it is spilt into small user stories.



Item	Portfolio backlog	Product backlog	Sprint backlog tasks
Unit	T-shirt sizes	Story points / ideal days	Ideal hours / effort-hours
When	Portfolio planning	Product backlog grooming	Sprint planning

Scrum Board: it is present in the conference room where scrum meeting are conducted. It contains 3 sessions

1) To do
2) In progress
3) done

Initially all the user stories posted in to do section in the form sticky notes as the sprint progresses it moves into in program and done section this gives transparency to all the scrum team members and they will clearly know how the sprint is progressing.

Thanks and All the best!!!

From

RG..