TEJOYASHA IT SOLUTIONS PVT LTD.

DevOps

[DevOps is the future]

Sunil Gadgil

11/26/2016

This document is suitable for beginners, who want to start their career in Information Technology and has some of little back-ground of programming. This document will give you information about DevOps, and its tool sets

Table of Contents

Chapter 1: Introduction to DevOps:

Important Terminologies/Definitions

Node: It is a computer/machine on which some data will processed to get desired output.

Client (Hardware) : It is a computer, on which a user will sit and do day to day work, example Desktop/Laptop.

Client (Software) : It is a piece of code which runs on Desktop/Laptops, e.g. Web Browser

Server (Hardware): It is a computer, which has more hardware capacity , designed to work 24x7x365 days.

Server: (Software): It is a piece of software which typically run on top of Server type hardware. e.g Web Server.

Operating System (OS) : It is a software which will interact and control hardware, like Keyboard, Screen.

Shell : It is a piece of software, which will help a user to interact with Operating System. e.g. bash, c shell

Device Driver : It is a piece of software which help Operating System interact with hardware.

Software: It is a piece of code, which will help to perform specific task. e.g. word processor to create documents.

Computer Hardware: It is a combination of CPU or Processor, RAM, Storage, Keyboard, Screen, Network.

Language: It is a set of instruction which will be understood by a human (High Level Language - HLL) or computer (Low Level Language - LLL)

High Level Language (HLL) : Language which can be easily read/written by a human (e.g. C, Java)

Low Level Language (LLL) : Language which need little more efforts to read/write by human (e.g. assembly language)

Compiler: A software which will help to convert high level language "non running" program to low language program.

Interpreter : A software which will convert high level language "running program" to low level language.

Programmer/Developer: It is a person who will right a meaning full code to produce desired result.

System Administrator/Operation : It is a person, who will help to install, configure and manage computer hardware/software.

Environments: It is a set/combination of hardware, software, configuration to produce certain results. e.g. Development Environment, Test Environment, Production Environment etc.

Code Testing: This task is to identify if the code is performing desired function. In various ways testing is done e.g. manual testing, automated testing.

File: In this we can store, data, code or images. e.g. oracle.db, program.java or image.jpg

Directory/Folder: It will be having collection of files.

Cloud: It is a combination of services like IaaS, PaaS or SaaS, through which you will able to get the environment like dev, test, or production.

Public Cloud: This service can be available to any person in the world.

Amazon AWS, Microsoft Azure, Google and other will provide this service.

Private Cloud: This service is only available to certain category of people, e.g. company employee

A company can host this service in own data center with software like..
Openstack, CloudStack etc.

IaaS: Infrastructure as a Service, this is delivered by a Cloud Service Provider, it will give basic components like Virtual Machine, Network, Storage and associated services

PaaS: Platform as a Service, this is delivered via Cloud Service Provider, it will give OS, Application and its configuration,

SaaS: Software as Service, this is your company product using which you company will generate revenue like gmail, facebook etc

What is DevOps?

It is a culture, movement or practice that emphasizes the collaboration and communication of both software development and other information-technology (IT) professionals while automating the process of software delivery and infrastructure changes.

It aims at establishing a culture and environment where building, testing, and releasing software can happened rapidly, frequently and more reliably

Environmental Definitions:

Production Environment : In this all servers are engaged to support customer activities in real time. The ideal aim is to make sure that using this servers the desired service need to be up and running 24x7x365 days. Example: Banking Web Site

Staging Environment : In this all the servers are treated as pre-production server, which are use to do tests as of real time servers. e.g. bombarding dummy users and test the capacity of server.

Testing Environment : In this all types of tests are performed, manual test, automated tests, black box test, hacking, cracking, software stability and much more.

Quality Assurance Environment : In this all types of quality checks are performed like User Interface (UI) look and feel, flow of the steps and so on

Development Environment : In this all types of development activities and small tests are performed, this can be even setup on small computer or laptop.

Development Cycle:

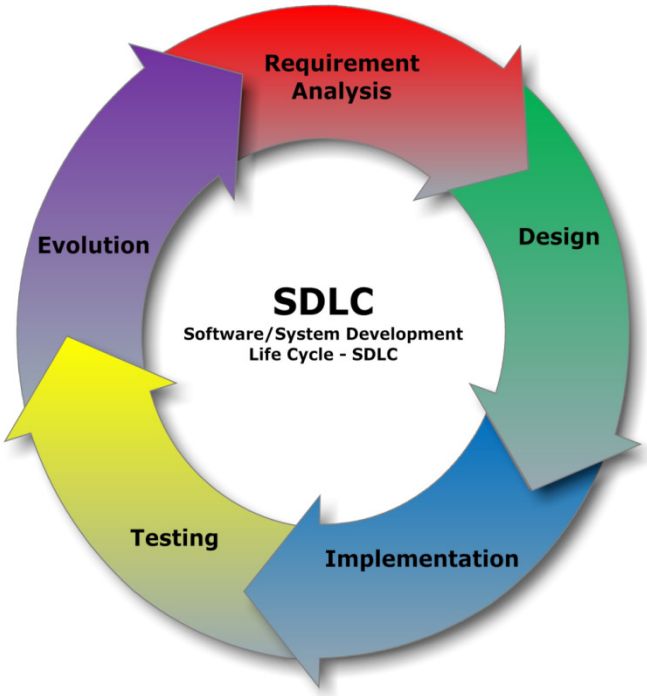


Diagram : 1

Operation Cycle:

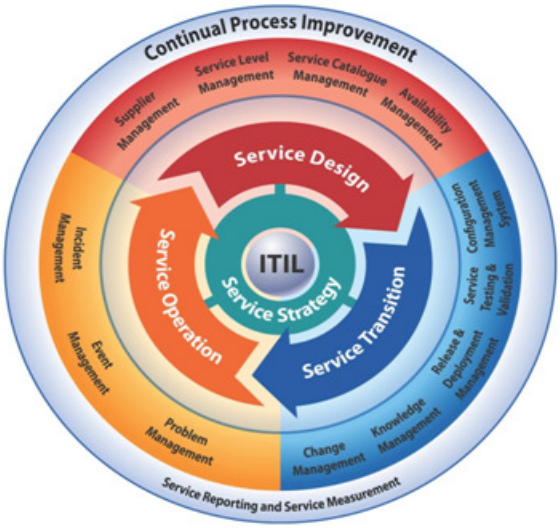


Diagram : 2

DevOps Cycle :

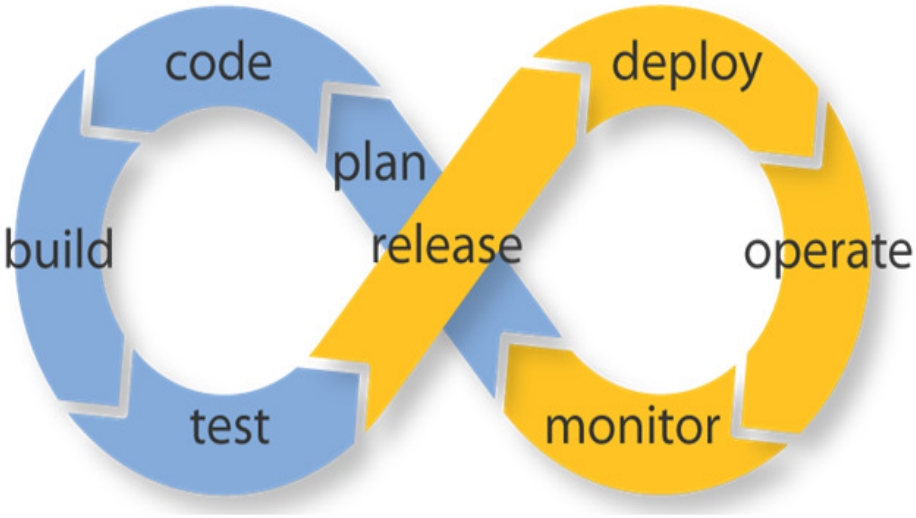


Diagram : 3

DevOps Tools Examples

Tools Category	Software/Tools Name
Source Code Management (SCM)	git, svn, perforce
Continues Integration (CI)	Jenkins, Hudson, Bamboo
Build	make, maven
Artifact Storage	Artifactory, Nexus
Monitoring	Nagios, Zenoss, Zabbix

Table: 1

Continuous Integration (CI) is a development practice that requires developers to **integrate** code into a shared repository several times a day. Each check-in is then verified by an automated build, allowing teams to detect problems early.

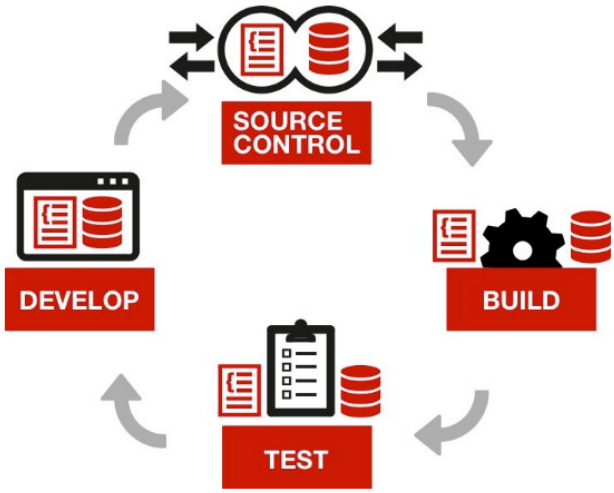


Diagram : 4

Continuous Delivery (**CD**) is a software engineering approach in which teams produce software in short cycles, ensuring that the software can be reliably released at any time. It aims at building, testing, and releasing software faster and more frequently. The approach helps reduce the cost, time, and risk of delivering changes by allowing for more incremental updates to applications in production. A straightforward and repeatable deployment process is important for continuous delivery.

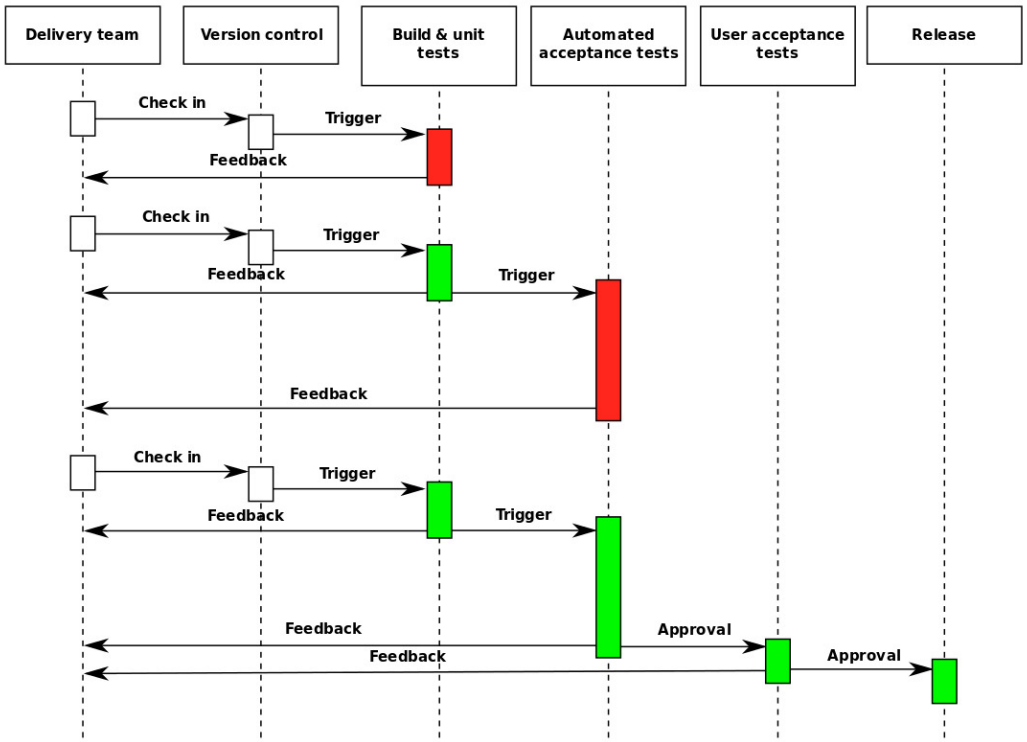


Diagram : 5

Continues Deployment

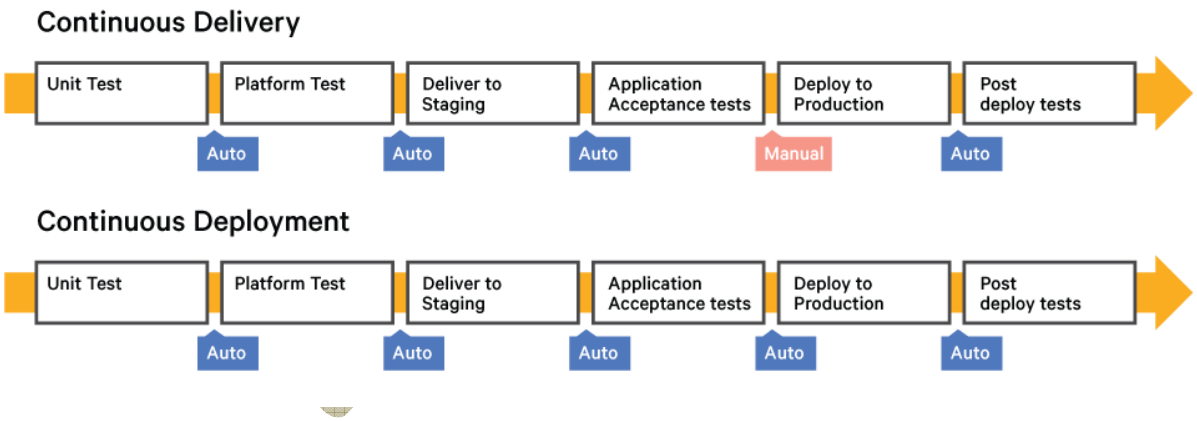


Diagram : 6

Chapter 2: Basics of Linux OS

Introduction

The **Linux** open source operating system, or **Linux OS**, is a freely distributable, cross-platform operating system based on Unix that can be installed on PCs, laptops, net-books, mobile and tablet devices, video game consoles, servers, supercomputers and more.

Linux comes in different flavors like Redhat, Ubunut, SuSE and so on. We are going to use CentOS which is one of the distribution of RedHat (CentOS, RHEL, Fedora).

It comes in two different categories 32-bit, 64-bit. In real time 64-bit is used, for our work 32-bit OS is also sufficient. We can download the same from

64-bit
http://ftp.iitm.ac.in/centos/7/isos/x86_64/
http://mirror.fibergrid.in/centos/7/isos/x86_64/
http://centos.excellmedia.net/7/isos/x86_64/
http://mirror.nbrc.ac.in/centos/7/isos/x86_64/

32-bit

CentOS 7

http://mirror.centos.org/altarch/7/isos/i386/

CentOS 6

http://mirror.nbrc.ac.in/centos/6.8/isos/i386/
http://ftp.iitm.ac.in/centos/6.8/isos/i386/
http://mirror.fibergrid.in/centos/6.8/isos/i386/
http://centos.excellmedia.net/6.8/isos/i386/

Make sure you have laptop with good resources like, minimum recommended configuration, dual core, 4 GB RAM, 100 GB HDD. We will use virtualization concept, i.e. we may use Oracle Virtual Box or VMWare work station as per your choice. We will install base OS, then Virtual Layer and on top of that we will install various OS flavours and applications (See. Diagram 7)

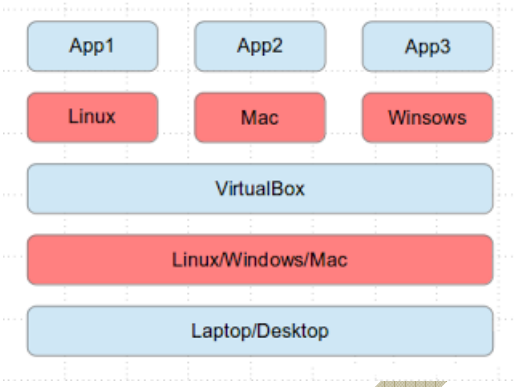
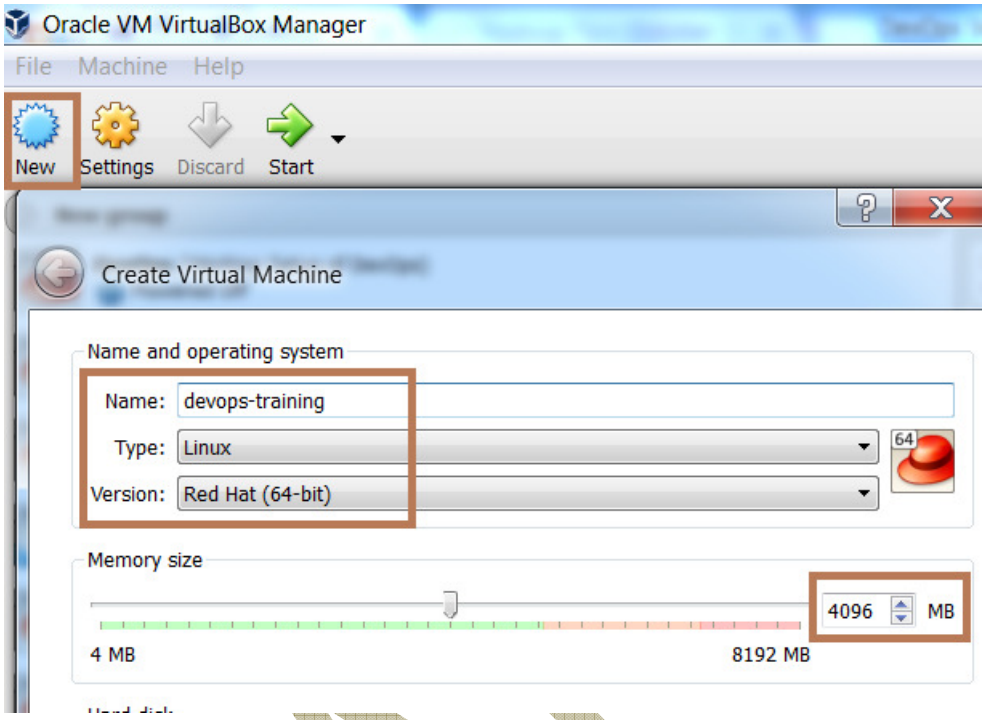


Diagram : 7

Linux Installation

Installation is the easiest part of the Operating System, below will be the steps involved for installation.

Select the Base OS type (Linux/Windows)
We will go with Windows 7
Select the virtualization layer (Oracle Virtual Box/VMWare Workstation)
We will go with Oracle Virtual Box
https://www.virtualbox.org
Define new system under Virtual Box
Name : devops-training
Type: Linux
RedHat: 64-bit



Select the disk size of appropriate size

We are selecting 50 GB

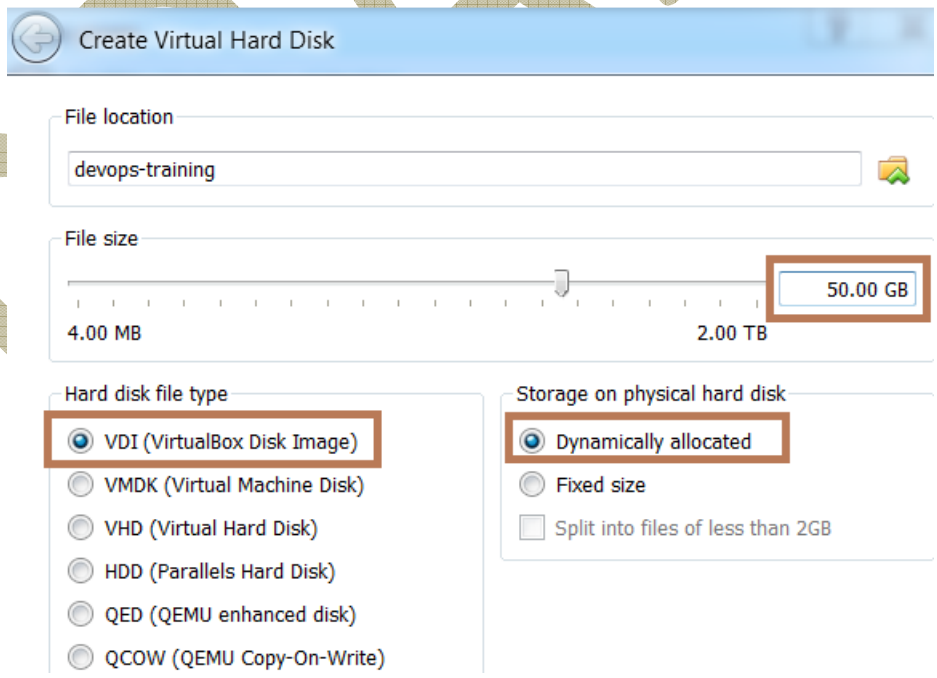


Diagram: 8

Go to settings, then Storage and provide the media in CD ROM Slot

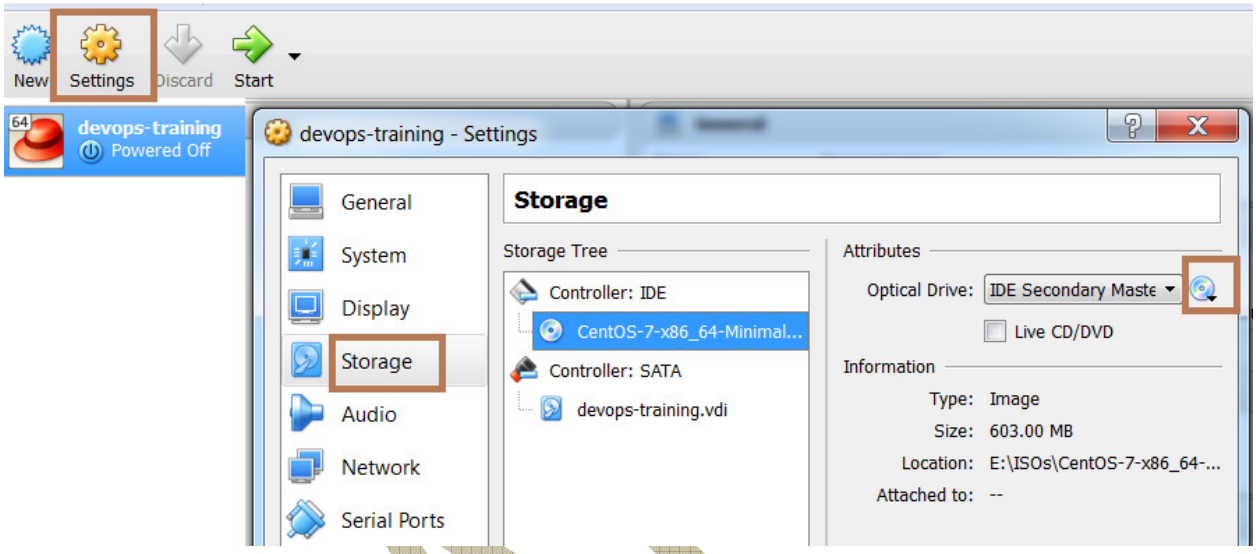


Diagram: 9

Click on start

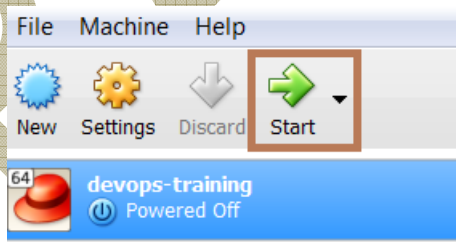


Diagram: 10

7. Select "Install Option" and proceed further.

CentOS 6.x (32-bit) CentOS 7.x (64-bit)

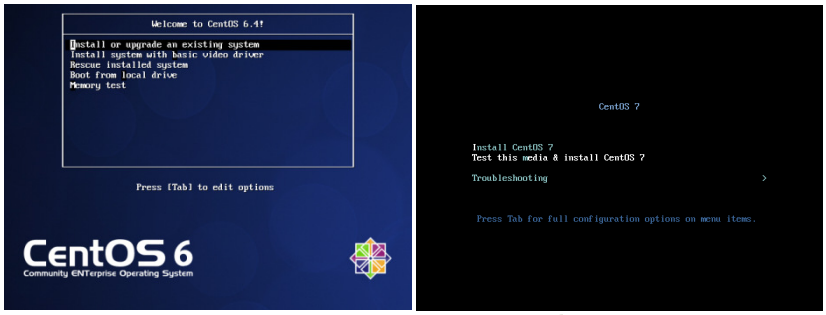


Diagram: 11

As we are using CentOS 7 , we will follow below steps

a. Select default language and keyboard type

b. Select appropriate Time Zone

c. Select destination

d. Set the network , the click on begin installation

for detailed, information , go to below link

http://linoxide.com/how-tos/centos-7-step-by-step-screenshots/

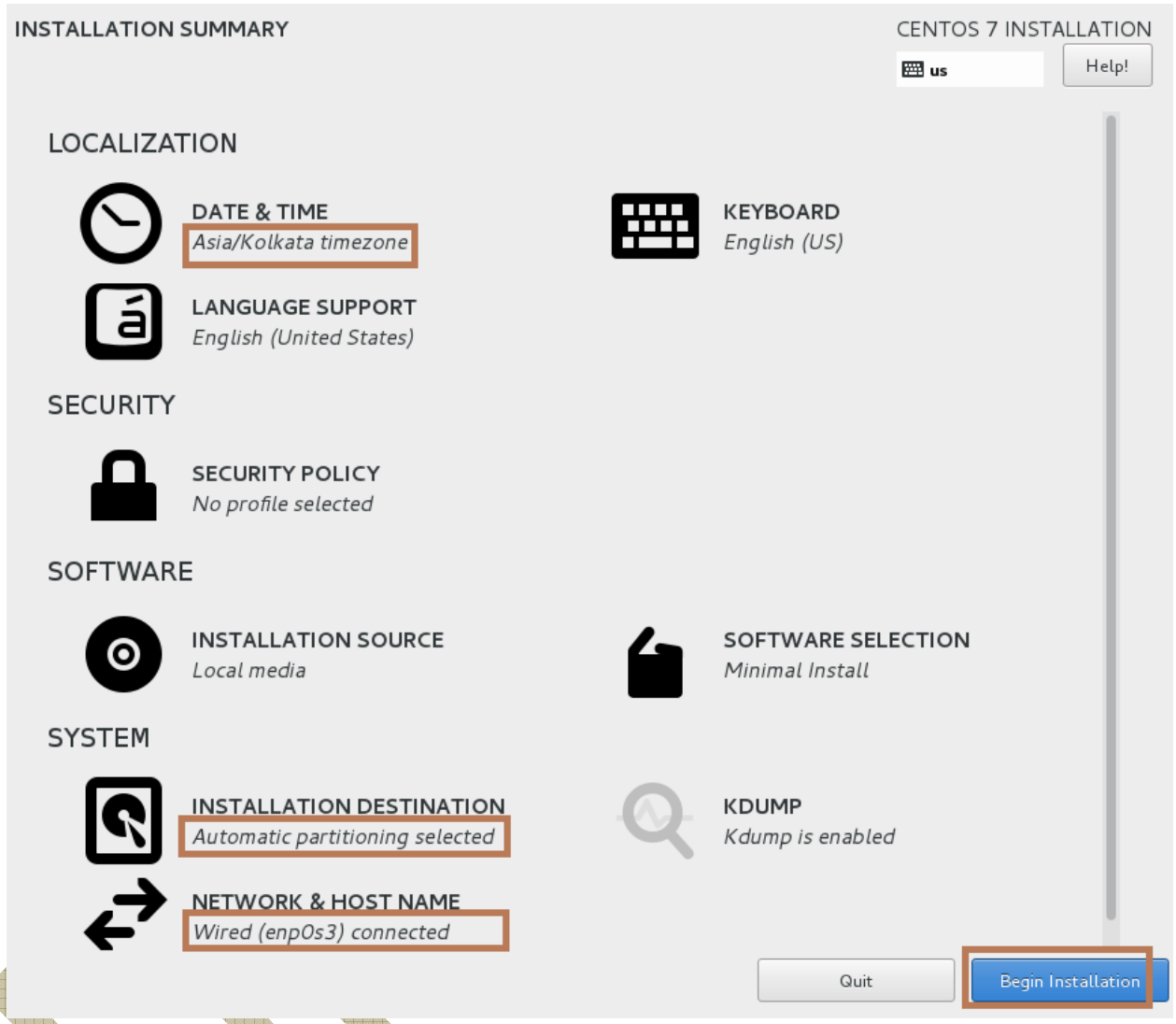


Diagram : 12

Once you begun with installation, set the root user password (Administrator user password) and create a user with name "devops" and set the appropriate password, also make sure you select "Make this user administrator"

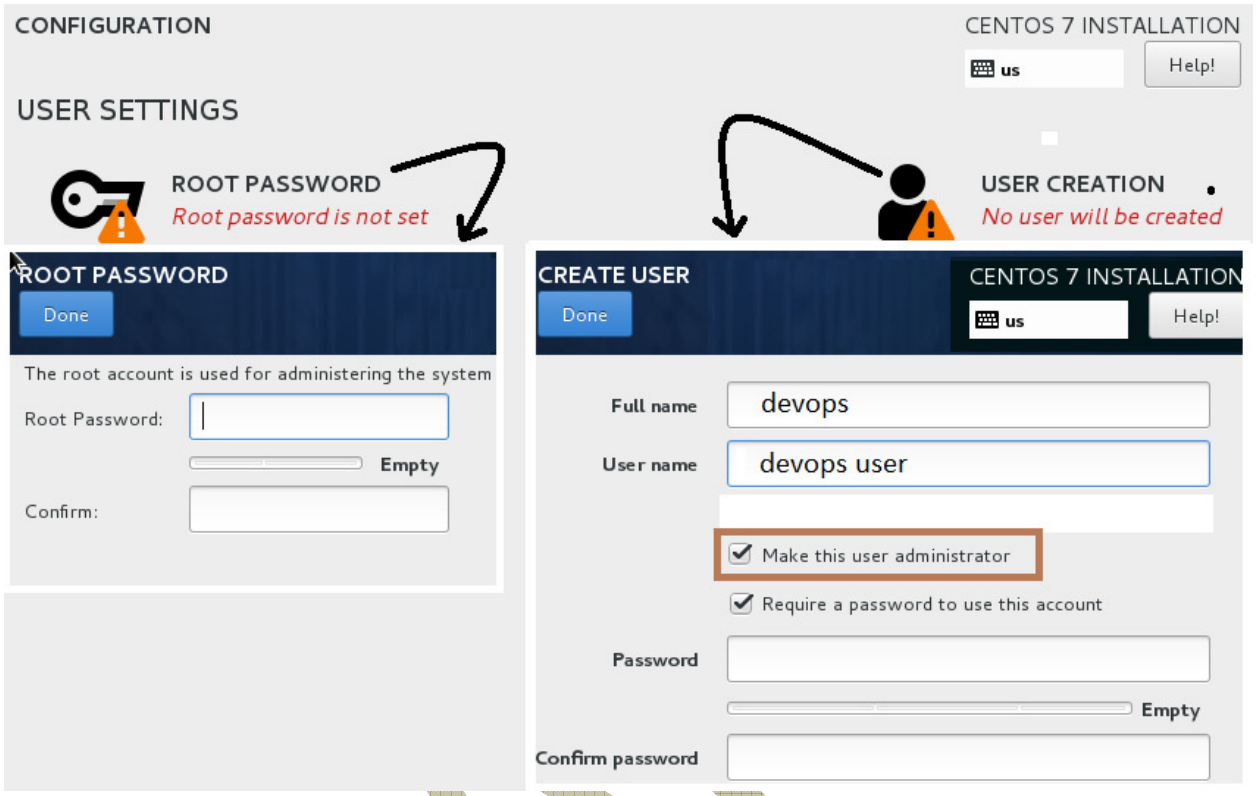


Diagram: 13

Linux Commands

Linux command has a specific syntax

command [options] arguments

command : It is a specific instruction given to the computer
options: It is a small variation expected while command is getting executed
arguments: The items on which the command need to operates called arguments.

You will able to get more details about command using "man" command. It is a very good documentation provided by the system.

Using man pages, you will able to get syntax information, short description, summary , detailed description with some examples and references. It is highly recommend to use man pages for any information needed for operation.

GUI Mode: Graphical User Interface

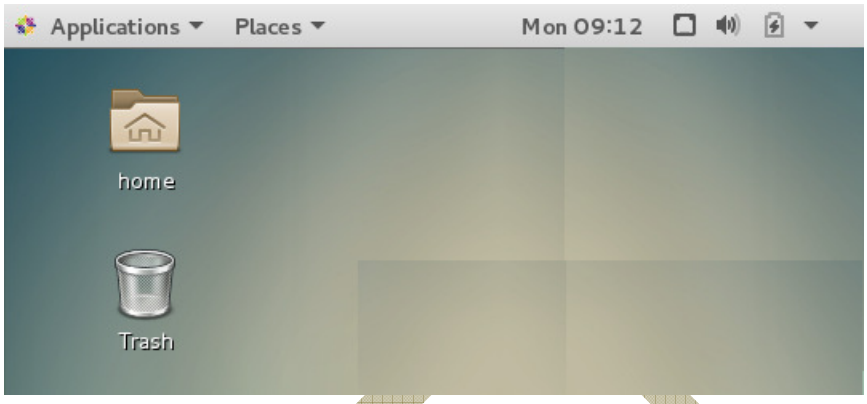


Diagram: 14

GUI mode is used on Desktop, generally this is not available on server side, it is not that you cannot have GUI mode on server, but it is not recommended to have the GUI as no one is expected to sit on the server to work on with GUI. GUI will take some RAM, Memory and processing power, so it is switched off on the server side. But on system like Fedora/Ubuntu flavors you will able to get many fonts and theme as per your choice, so try GUI to see the power and GUI features of Linux. For beginners we recommend GUI mode, to get some good familiarity before they start going into expert mode of operation that is firing command on terminal interface.

Terminal Usage: Text User Interface

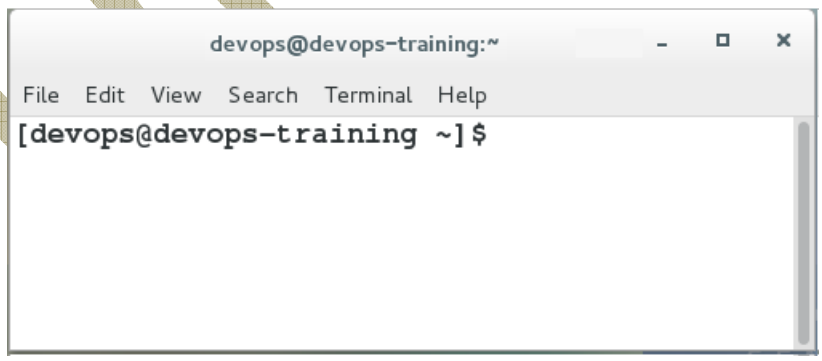


Diagram: 15

On the server side, this is the most practical and useful way to interact with Linux system. It is not only efficient, but gives a user full power and control over operating system, which may not be possible on GUI.
A administrator will be login to server via program like SSH (Linux) or Putty (MS Windows). User can use username/password, or certificate based authentication and other security aspects of the organization.

Once user logged in via command line, he/she may get the command prompt like

[devops@devops-training ~] $

the above pattern is called command prompt, let us see what information its giving. Here the user name is devops, system/host/node name is devops-training, the ~ indicates the user is in his home directory and $ indicates user is normal user.

[root@devops-training ~] #

Here the user name is "root" (System Administrator), system/host/node name is devops-training, the ~ indicates the user is in hist home directory and # indicates the user is normal user.

Basic Command Table:

Command Name	Short Description
cat <filename>	Use to see the text file contents OR to concatenate files
echo "message"	Use to print messages on screen, specifically in shell scripting
Date	Use to get/system system date
Pwd	Use to see the current/present working directory path
cd "/path/dirname"	Use to navigate between different directories
mkdir "/path/dirname"	Use to create directory
rm "/path/filename	Use to remove files or directories
More	Use as a pager, break the output with 24 lines on screen
Less	Similar to above, and have some more features for

navigation

Note: We are going through each command in details in our class room sessions.
Table II

User Management :
When we are working with a system, we need to have a user, we different types of users, system users (e.g. root) , application user (e.g. apache) and normal user (e.g. suresh) .

As a root user, we can create, modify and remove the user from system. Normally you always work with normal user, if you need root level access/privileges, you will use sudo command.

A user is also associated with group. A group is a collection or set of users. we have similar category of groups as that of user type.

Below are the important command and files associated with user management

Command Name	Description	Example
Useradd	Create New User	useradd devops
Usermod	Modify Existing User	usermod -G wheel devops
Userdel	Delete Existing User	userdel suresh
Groupadd	Create New Group	groupadd employee
Groupmod	Modify Existing Group	groupmod -g 2000 jenkins
Groupdel	Delete Existing Group	groupdel sample
Passwd	Change User Password	passwd devops
Id	Give user information	id devops
Chage	Change user password age info	su - root
Sudo	Get super user privileges for short time	sudo cat /etc/shadow
Su	Switch User	su - user01
File Names	Description	
/etc/passwd	Use to store user information	
/etc/shadow	User to store user password and age info	
/etc/group	User to store group information	

Package Management

Package is a bundle of files associated with a given application. Example we have to install web server (httpd) , to install this we need lots of files, programs, images, documents. This will come into bundled form.

For RedHat Products, rpm is the package manager. we will use this to installed, update or remove packages from the operating system.

package naming convention:

pakageName-packageVersion-OSRelease-archictureType

httpd-1.2.3-456-el7-x86_64.rpm

packageName = httpd
packageVersion=1.2.3-456
OSRelase=el7
Arch=x86-64 (64-bit)

Important RPM commands

Package Query

This is use to query more information about packages which are already installed on the system. This is very useful while doing system update/maintenance.

rpm -q -q - query (this is query option and also need some sub options)

rpm -qa -query all installed package names

e.g. rpm -qa | more
rpm -qa | grep http

rpm -qi -give detailed information about already installed package

e.g. rpm -qi httpd

rpm -ql -list all files from already installed package

e.g. rpm -ql httpd

-qf -show the package with associated with this file

e.g. rpm -qf /etc/passwd

Package Installation

This is used to get additional package installation on your system, here we assume that you have already downloaded the necessary packages and the dependencies has been taken care.

rpm -i -install a given package

e.g. rpm -i httpd-v1

rpm -iv -install a given package with more info on screen

e.g. rpm -iv httpd-v1

rpm -ivh -install a given package with more info and progress bars with "#"

e.g. rpm -ivh httpd-v1

Package Upgrade

This is use to upgrade the version of your package to the next level. This is become handy when you want to update from verson1 to version2.

rpm -uvh -upgrade a give package with more info and progress bar

e.g. rpm -Uvh httpd-v2

Package Remove

Some time you do not need some packages as those are no longer needed in service. This option become handy and it will remove all the mentioned packages, you can also erase multiple package from the same command line.

rpm -e -remove a package from a given system

e.g. rpm -e httpd

some time you have to forcefully remove the package without bother about dependencies you can do that with option --nodeps (no dependencies check)

rpm --nodeps -e httpd

Warning : The --nodeps options may break your system dependencies, so be careful.

vi editor

vi editor has 3 modes of operation, Edit mode, Interactive mode and command mode.

1. Edit Mode

i	insert,
a	append,
R	replace

2. Interactive Mode

#yy	copy (yank)
#p	paste
#dd	delete line
#dw	delete word
#x	delete char
u	undo
/<word>	find

3. Command Mode

:w	write or save
:q	quit without save
:wq	write and quit

Find and replace

:line_start,line_end<substitute>/what/which/<options>

:5,10s/editor/EDITOR/

More find and replace examples

Repace first occurance

:1,$s/editor/LINUX/

Replace first column

:1,$s/^editor/LINUX/

Replace last column

:1,$s/editor$/LINUX/

Replace all words

:1,$s/editor/LINUX/g

:5,10d : delete lines from 5 to 10
:%d : delete all lines from the file

Chapter 3: Shell Scripting:

What is Shell

Shell is a small program which will help to user to interact with Operating System. Shell scripting is a script/program written to control Operating System to do various tasks like log file analysis, taking system backup and much more.

Types of shell

Below are some examples of various shells available

example :

/bin/sh	bourne shell
/bin/bash	bourne advance shell
/bin/csh	c shell
/bin/tcsh	c advance shell

and so on

Understanding she-bang

Combination of two characters "#!" is called she-bang, at starting of every script, it is recommended that your first line need to start with she-bang

example :

firstscript.sh
#!/bin/bash

Understanding comments

Comments are the extra information give by program creator, it will give you the hint what is use of the commands, or login inside the script.

Comments start with hash character "#", you can start writing comments from second line onwards.

Note: When shell script is executing, the lines starting with # are ignored.

example :

firstscript.sh
```
#!/bin/bash
###################
# Purpose: To write first shell script
# Owner: info@tejoyasha.com
# Version :1.0
# #################
# Clear the screen
clear
# Print message on screen
echo "!! Welcome to Shell Scripting !!"
# End of script
```

Start a shell script

You need to give executable permission to run the script e.g.

```
chmod 755  firstscript.sh
chmod +x firstscript.sh
```

You can start shell with different method

```
/bin/bash firstscript.sh
/home/<username>/scripts/firstscript.sh
cd /home/<username>/scripts; ./firstscript.sh
```

Understanding Variable

Variable is a name/label given to a memory location, which will hold some value in it.
e.g. VERSION=9
here VERSION is variable name and 9 its value

Types of Variables

We have different types of variable like

- System Variable: Already defined when you installed OS
  - PATH, SHELL
- User Defined Variables : You are creating system
  - NAME, AGE
- Shell Variables : Shell will use this variable for its own purpose
  - $$, $@

Variable Names

You should only use letters, underscores and numbers to defined variable
valid names

COMPAYNAME=tejoyasha
COMPANY_NAME=accenture
_COMPANYNAME=infosys
COMPANYNAME1=ibm

set/unset variables

When you are on command prompt, you can set the variable using command

export OSNAME=CentOS

You can also unset the variable with below command

unset OSNAME

Using quotes with variables

What is use of quotes

Using quotes around variables you can do lot of string manipulations
Lets assume you define variable COMMAND="/bin/ls"

What is use of single quote

echo '$COMMAND'
It will print the string as it is or exactly

What is use of double quote

echo "$COMMAND"
It will evaluate the variable name and get its value to work

What is use of back-tick quote

echo `$COMMAND`
It will evaluate the variable name and try to execute it as command

if statement

You can use the if statement to check the condition and get the true or false decision

Syntax:

```
if [ condition ]; then
    #if the condition is true execute all below commands
    command[1]
    command[2]
```

```
        ....
        ....
        command[N]
fi
```

Syntax:

```
if [ condition ]; then
        #if the condition is true execute all below commands
        command[1]
        command[2]
else
        #if the condition is false execute all below commands
        command[3]
        command[4]
fi
```

Syntax:

```
if [ condition ]; then
        #if the condition is true execute below command
        command[1]
elif [ condition ]
        #if the condition is true execute below command
        command[2]
elif [ condition ]
        #if the condition is true execute below command
        command[3]
else
        #if the condition is false execute below command
        command[4]
fi
```

Test Conditions

Test conditions are use to perform decisions in your shell scripting, there are below types

File Test

String Test

Numeric Test

Logical Test

File Test:

File tests are used to get more information about file or directories.
some important options are

Option	Description
-f	Test a file
-d	Test a directory
-r	Test a user can read the given file
-w	Test a user can write in given file
-x	Test a user has executable permission on the given file
-s	Test the size of file, if size is greater than zero, the result is true
-k	Test the given file as sticky bit permission

Note: These are only most frequently used options.

example :

```
if [ -f "/etc/passwd" ]; then
        echo "Yes, the /etc/passwd file is available"
fi
```

String Test:

String tests are performed the comparison between two string and identify they are matching or not.

Below are the operators available

Option	Description
==	To check if two strings are matching (if match, result is true)
!=	To check if two strings are not matching, (if doesn't match, result is true)

example:

```
USERNAME="suresh"
if [ "$USERNAME" != "root" ]; then
        echo "Yes, string username suresh is not matching with string root"
fi
```

Number Test

These are used to perform the number comparison, we have various symbols like =, >, < , >=, <= and more. In shell scripting we need to use below symbols for numeric comparison.

Symbol	Description
-eq	If numbers are equal result is true , otherwise false
-ne	If numbers are not equal result is true, otherwise false
-gt	If first number is greater than other result is true, otherwise false
-lt	If first number is less than other result is true, otherwise false
-ge	If first number is greater than or equals to other result is true, otherwise false
-le	If first number is less than or equals to other result is true, otherwise false

example:

```
MARKS=58
if [ "$MARKS" -gt 65 ]; then
        "Student Result : First Class"
elif [ "$MARKS" -gt 55 ]
        "Student Result : Second Class"
elif [ "$MARKS" -gt 45 ]
        "Student Result : Third Class"
else [ "$MARKS" -gt 35 ]
        "Student Result:  Pass Class"
else
        "Student Result: Failed"
fi
```

When you run the script the output will be :

Student Result: Second Class

Logical Tests

It is used to combine two or more expressions, condition and create one result.

Symbol	Description
-a	Use to AND the expression
-o	Use to OR the expression
!	Use to NOT the expression

example:

```
MARKS=60
if [ "$MARKS" > 55  -a "$MARKS" -le 65" ]; then
        echo "Student Result:  Student is first class"
fi
```

The above statement says, if the students (MARKS are greater than 55) AND (MARKS are less than or equal to 65) the result is first class, here we used logical AND operator.

File redirectors

We have 3 different categories, standard input, standard output and standard error

Name	Description	Numeric Representation
STDIN	Standard Input (input from keyboard)	0
STDOUT	Standard Output (output to screen)	1
STDERR	Standard Error (output to screen)	2

example :

```
cat /etc/passwd >  /tmp/dummy.txt
```

by default cat /etc/passwd command will show the output on screen, the use of ">" will transfer the screen output to the /tmp/dummy.txt file

">" or ">>" are a redirector symbol, they have special meanings.

">>" indicates that, if the /tmp/dummy.txt file already exists, then append to the file
if the /tmp/dummy.txt file doesn't exists, then create it.

the same above example can be written in different ways also.

```
cat /etc/passwd > /tmp/dummy.txt    OR
cat /etc/passwd 1> /tmp/dummy.txt
```

both are one and the same but in second case we are explicitly mentioned the redirection of STDOUT to a file.

You can redirect the output and error to different files.

```
./myscript.sh  1>output.txt 2>error.txt
```

If the program executed successfully without any error, it will redirect the result to output.txt, if it has some problem/errors it will redirect the same to error.txt

Exit Status

When any program executes either its successful or it will fail, when it successful, it will set a special shell variable called "$?" to 0, otherwise it will set the variable to any value greater than 0.

example :

```
id root
# result of id command will be stored under "$?" special shell variable.
if  [ "$?" -eq 0 ]; then
        echo "The id command ran successfully"
else
        echo "The id command failed"
fi
```

Case Statement

We learn about if/elif nested statement, to simplify more easy way of writing, case statement was created. It is more clean way of execution if if/elif.

example:

```
MYOS="CentOS"
case $MYOS in
        AIX)
```

```
                echo "You have chosen $MYOS"
                echo "Use smitty command"
                ;;
        CentOS)
                echo "You have chosen $MYOS"
                echo "Use yum command"
                ;;
        Ubuntu)
                echo "You have chosen $MYOS"
                echo "Use apt-get command"
                ;;
        *)
                echo "You have chosen $MYOS"
                echo "Please select correct OS"
                ;;
esac
```

Loops

Loops are use to iterate over the same set of commands for multiple times. It will add more power and flexibility to the operation.

for loop

Below is the syntax of for loop, you can get the number of iteration specified and you are ready to go and loop over again and again.

```
syntax :
for VAR in LIST
do
        command[1]
        command[N]
done

example 1:

for COUNT in {1..10}
do
        echo "The count value is $COUNT"
done
```

the above program COUNT is the variable, in which we are passing value from 1 to 10 using number generator {1..10}, the commands between do and done will print the value of each listed items.

example 2:

```
for NAME in suresh, ali, ramesh,  harprit, kiran, john
do
        echo "The user name is $NAME"
done
```

in above example, NAME is a variable under which a name from list of user will be taken and loop over do and done in one by one fashion.

while loop

The limitation of for loop is you do not have control on the loop as when to conditionally exit out of the loop, while loop will give that opportunity.

syntax :

```
initialize condition
while [ condition ]
do
        command[1]
        command[N]
done
```

example :

```
ANS="Y"
while [ "$ANS" == "Y" ]
do
        FRIEND=""
        echo "Please enter your friend name"
        read  FRIEND

        echo "Your friend name is $FRIEND"
```

```
        echo "Do you want to continue (Y/N)"
        read ANS
done
```

In the above example, we initialize a condition about ANS is equal to Y, when it comes to the while condition, it confirms that ANS is matching with string "Y" , then only it will enter in the loop, (here while works on true logic, that means it will only execute the loop when the condition is true)
Then it will ask for friend name, and get the value via keyboard into FRIEND variable using "read" key-word, it will print the value of the FRIEND, then it will ask to enter your choice to continue with Y/N.
if you give Y, it will loop over again, if you choose N, it will come out of the loop.

```
example 2:
COUNT=1
while [ "$COUNT" -le 10 ]; then
do
        echo "The count value is $COUNT"
        COUNT=`expr $COUNT + 1`
done
```

The above example will print a count value from 1 to 10, note you cannot add numbers directly in shell script, so we are using other command "expr" to do it.

Until loop

Until loop is said to the negative logic or false logic loop and it is exact opposite of while loop.

```
syntax:
initialize condition
until [ condition ]
do
        command[1]
        command[N]
done
```

```
example 1:
COUNT=1
until [ "$COUNT" -gt 10 ]; then
do
        echo "The count value is $COUNT"
```

```
        COUNT=`expr $COUNT + 1`
done
```

The above example will print a count value from 1 to 10, note you cannot add numbers directly in shell script, so we are using other command "expr" to do it.

Note: in while loop we used "-le" logic, where as in until we use opposite logic "-gt"

Chapter 4: Git (Version Control)

Introduction

By far, the most widely used modern version control system in the world today is Git. Git is a mature, actively maintained open source project originally developed in 2005 by Linus Torvalds, the famous creator of the Linux operating system kernel. A staggering number of software projects rely on Git for version control, including commercial projects as well as open source. Developers who have worked with Git are well represented in the pool of available software development talent and it works well on a wide range of operating systems and IDEs (Integrated Development Environments).

Having a distributed architecture, Git is an example of a DVCS (hence Distributed Version Control System). Rather than have only one single place for the full version history of the software as is common in once-popular version control systems like CVS or Subversion (also known as SVN), in Git, every developer's working copy of the code is also a repository that can contain the full history of all changes.

Why to use version control ?

Keeps records of changes
Allows developers to work simultaneously.
Does not allow overwriting each other’s changes.
Backup

Advantages of Git

Free and open source
Simplicity
Small and faster
Security
No need of powerful hardware
Easer branching
Implicate backup

Git is platform independent tool :

It is supported on
Linux
Windows
Mac

Installation of Git on CentOS

Use yum command to install the package (-y is use to give a default answer as yes)

```
$ sudo yum  install git  -y
```

if you are not able to get the package, probably the epel repo is not enable. please ref. Annexure I for more details on EPEL.

Chapter 5: Git Background

In other version control management system, we have client and server relation, that means, always the server need to be available for revision control, and always there are overheads over the network for each commit. This is not always the case with git.

What is repository?

It is a place at which you will find a collection of files, folders, programs, images, music, documents and much more.

Local Repository : Files stored locally in systematic manner.

Remote Repository: Files stored on the server/different system in systematic manner.

Initialization :

You need to initialize a local repository, then you can sync it up with remote repository. To initialize the repo follow below command.

```
$ git init
```

It will create a .git directory, under which it will create the repository to track files locally. You are not suppose to go in this directory it is for internal git use only.

Global settings:

To work further you need to set couple of global variables

```
git config --global user.name "Your Name"
git config --global user.email "your@email.com"
```

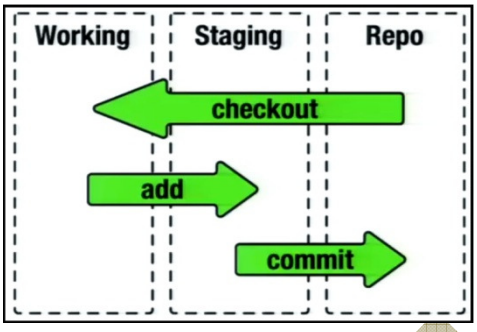
Note that these setting has nothing to do with authentication, your authentication details will be set further when you setup your local repository with remote repository. The above variable just indicates that who as a user you are connecting to repository and what name and email it indicates if some one try to understand it.

The 3 stages:

Once you initialize the git, it will create a local repository for you. This local repository has 3 different areas.

1. Working Area
2. Staging Area
3. Repo Area

The working area is your directory under which you are creating your files, programs for tracking purpose. The staging area is an intermediate area in which you are pushing files which you feel are now read to ship to the repo. The Repo Area is a final destination of your local repository. All the files which are committed in this area are have a version number associated with it for tracking purpose.



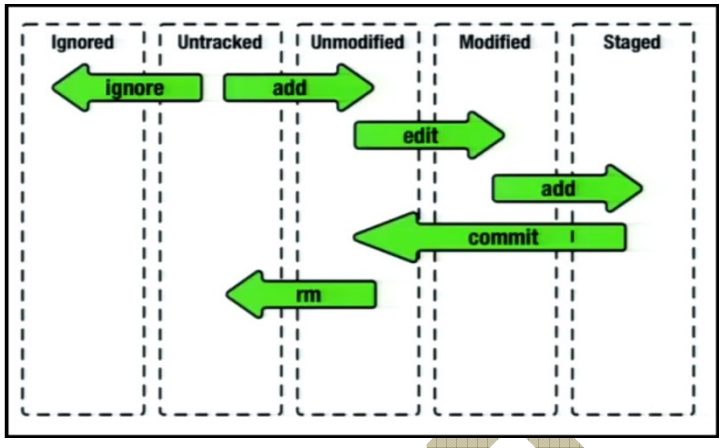
Let's assume you have create your first file under repository and want to keep a track of it. You may use below command to do so

```
$   vi daily-backup.sh
#!/bin/bash
##############################
#  Purpose: To take backup of a  /etc directory
# Owner : info@tejoyasha.com
# Version: 1.0
##############################

:wq



$ git status
                # Untracked File
                # daily-backup.sh
nothing added to commit but untracked files present

$ git  add daily-backup.sh

$ git status
                #Initial Commit
                #new file: daily-backup.sh

        $ git commit   -m "First version of backup script"  daily-backup.sh
[master (root-commit) 58fe1b9] First version of backup script
 1 files changed, 10 insertions(+), 0 deletions(-)
 create mode 100644 daily-backup.sh

$ git status
```

```
# On branch master
nothing to commit (working directory clean)

        $git log
commit 58fe1b955fd3dc8dd3fd92554fb397211d6d661d
Author: Sunil Gadgil <info@tejoyasha.com>
Date:   Sat Dec 3 16:19:22 2016 +0530

First version of backup script


$ git show
commit 58fe1b955fd3dc8dd3fd92554fb397211d6d661d
Author: Sunil Gadgil <info@tejoyasha.com>
Date:   Sat Dec 3 16:19:22 2016 +0530

   First version of backup script

diff --git a/daily-backup.sh b/daily-backup.sh
new file mode 100644
index 0000000..f154449
--- /dev/null
+++ b/daily-backup.sh
@@ -0,0 +1,10 @@
+#!/bin/bash
+##############################
+#  Purpose: To take backup of a  /etc directory
+# Owner : info@tejoyasha.com
+# Version: 1.0
+##############################
```

The 5 stages

Now we will see more command in this stage



Many a times we do not want to track some files, so git had made a provision to ignore these files. for that you need to create a file name ".gitignore" , then mentioned the name of the files that you want to exclude them from git tracking.

```
$vi   .gitignore
*.bak
*.log
```

Once you add the file you may want to remove the file totally from the revision control. Once you removed the file its no longer being tracked.

```
$git rm  test-file.sh
        rm 'test-file.sh'

$git log

commit 846ad938ce4bebdb3c90231527b93af9d67871d1
Author: Sunil Gadgil <info@tejoyasha.com>
Date:   Sat Dec 3 17:49:26 2016 +0530

removing test-file.sh
```

Git Options	Description
git init	To initialized the git repository
git config	To set the global configuration
git add <filename>	To add new files into tracking system
git commit -m <filename>	To confirm the changes and mark them as final
git rm <filename>	To remove a file from git tracking system

git status	To get the status details
git diff	To see the difference between different commits
git log	To get the tracking logs
git show	To get the differences with previous commits
git branch	To do various operations with branches
git merge	To merge the necessary branches

Installation of gitBucket server (Open Source)

Download the necessary gitbucket.war file from internet.

Copy the file into the appropriate directory, in this we are going to create in our home directory

start the git bucket service, by default will start at port 8080, use default credentials root/root

open the browser http://serverip:port/gitbucket

$ java -jar gitbucket.war

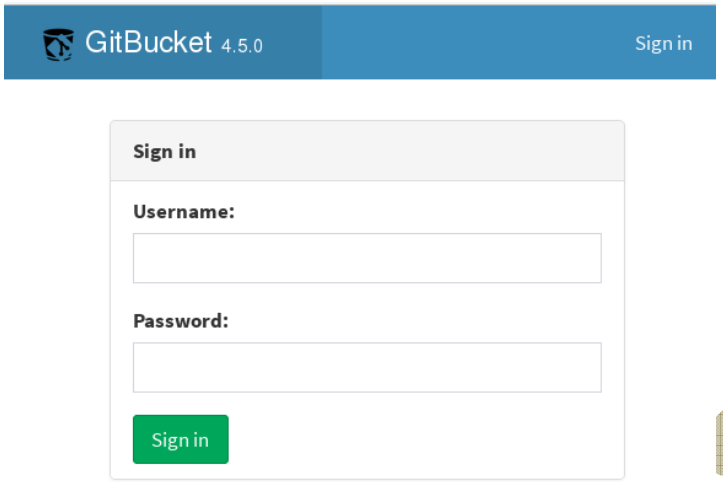
--port=[NUMBER]

--prefix=[CONTEXTPATH]

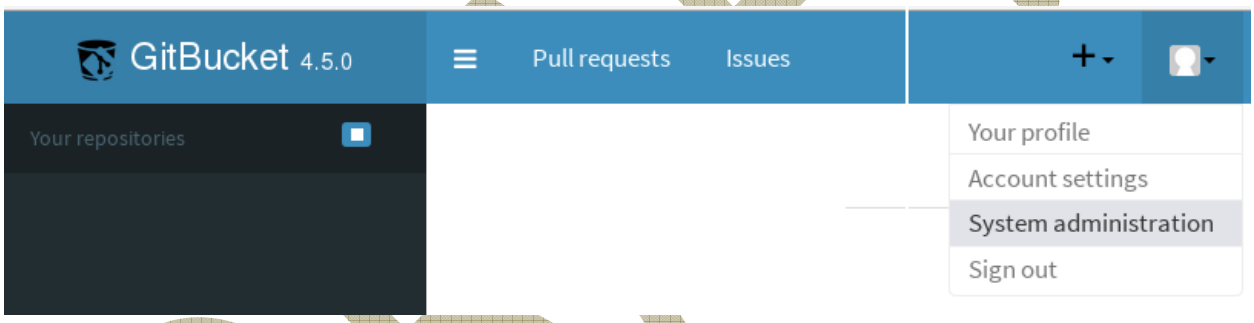
--host=[HOSTNAME]

--gitbucket.home=[DATA_DIR]

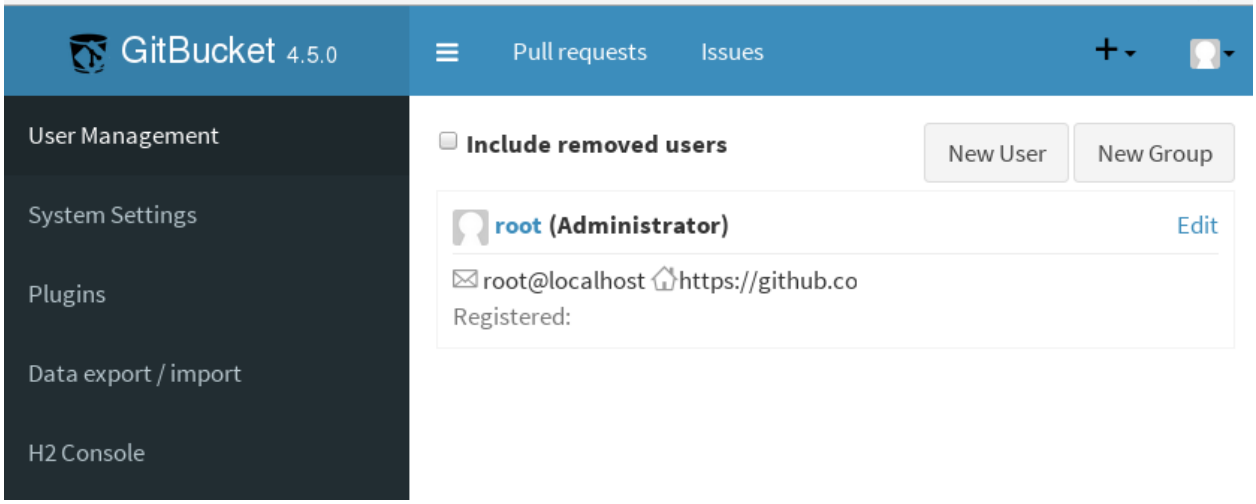
Note: It needs java to run the application, by default it may not be available. use yum command to install the same.



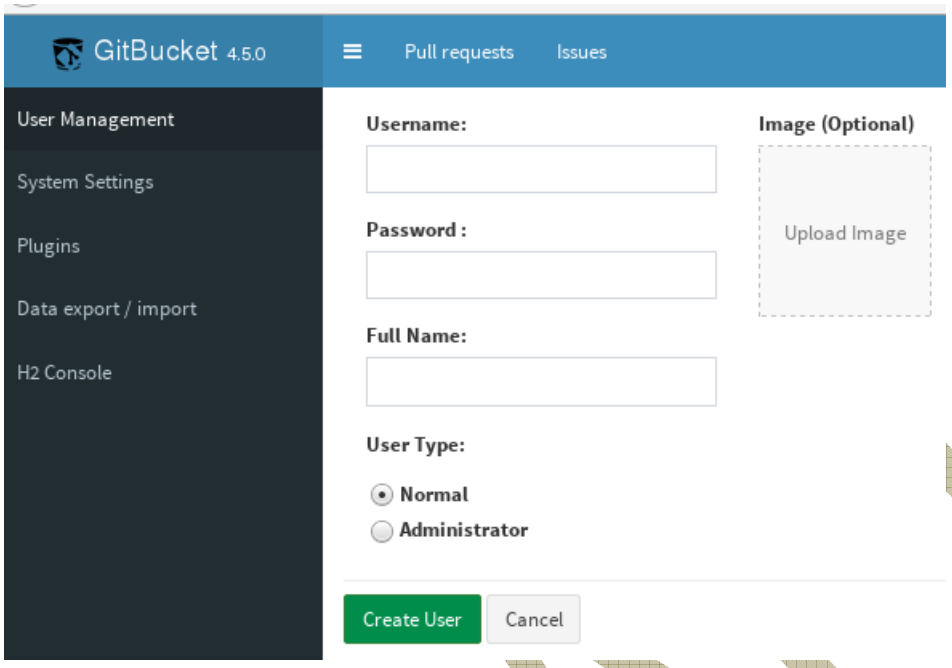
Now login to the server and create users and repositories



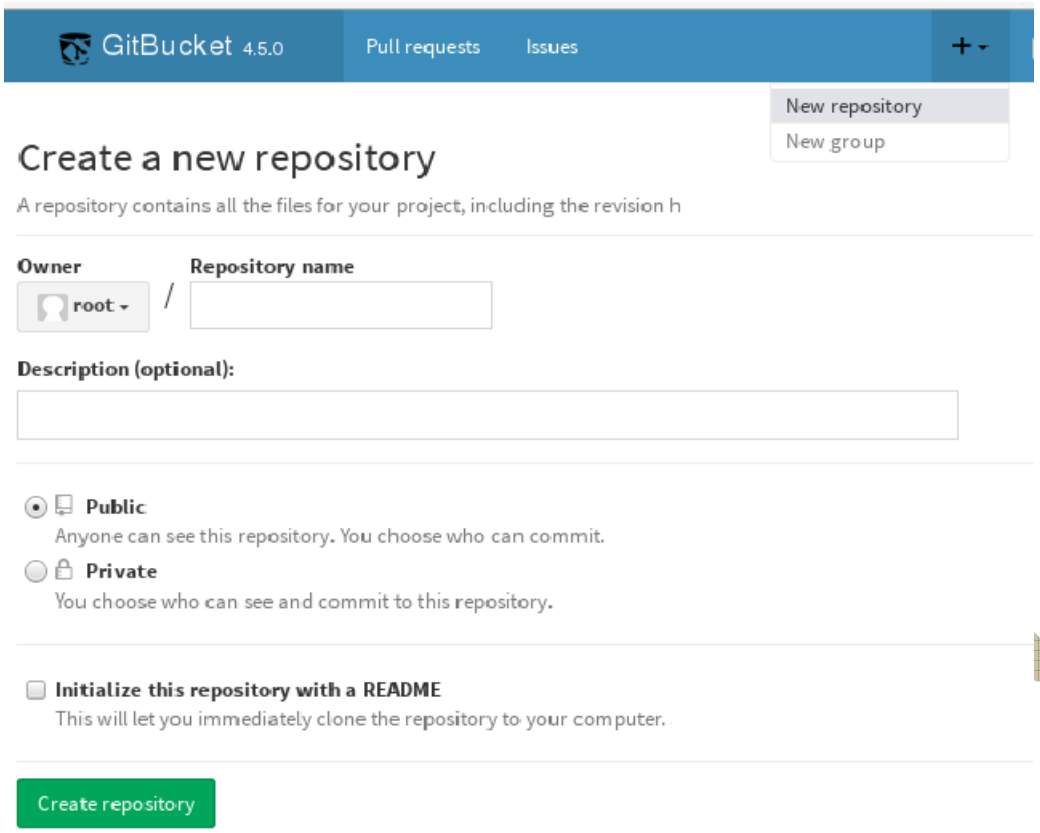
Click "New User" button



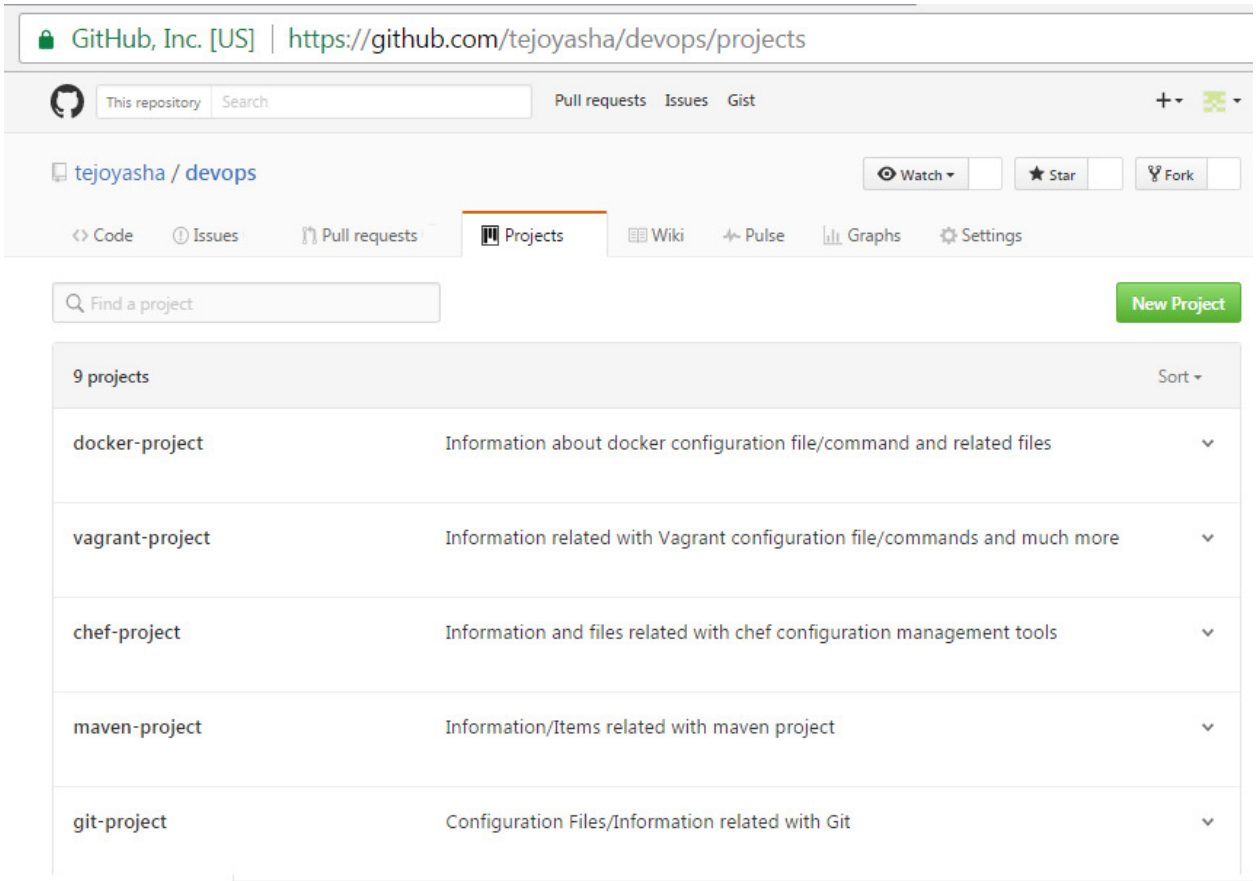
Fill in the form and create the user



Now login with the new user and create the "New repository"



GitHub (Free & Paid Service for Repository)


GitHub, Inc. [US] | https://github.com/tejoyasha/devops/projects
This repository
Search
Pull requests
Issues
Gist
tejoyasha / devops
Watch
Star
Fork
Code
Issues
Pull requests
Projects
Wiki
Pulse
Graphs
Settings
Find a project
New Project
9 projects
Sort
docker-project
Information about docker configuration file/command and related files
vagrant-project
Information related with Vagrant configuration file/commands and much more
chef-project
Information and files related with chef configuration management tools
maven-project
Information/Items related with maven project
git-project
Configuration Files/Information related with Git

Chapter 5: Vagrant

It is a mechanism through which we can easily deploy the development environment. It is use to create VM and quickly configure them according to your need.

The main configuration file is the Vagrant file in your project directory, which will give the necessary configuration details about the system you are going to spin up. The advantage with vagrant is that you are reducing your installation and deployment time during real world operation.

It is supports MS Windows, Linux, Debain and Mac OS. With some supported virtualization layer like Virtualbox, VMWare, AWS, Azure and other cloud platform.

You can download the vagrant as per your choice of OS.

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

To work with vagrant you need to down load the image boxes from net to your local machine.

You can get the images from vagrant cloud.

Vagrant Installation Steps

Step 1

Download and install Vagrant within minutes on Linux, Windows or Mac OS. No complicated setup process, just a simple to use OS-standard installer.

Step 2

Create a single file for your project to describe the type of machine you want, the software that needs to be installed, and the way you want to access the machine. Store this file with your project code.

Step 3

Run a single command — "vagrant up" — and sit back as Vagrant puts together your complete development environment. Say goodbye to the "works on my machine" excuse as Vagrant creates identical development environments for everyone on your team.

Vagrant Command List

You can run command vagrant list-commands and you will able to get the list of all commands supported by vagrant, this list his very handy to know what is inside and what you can quickly do with the vagrant and its command with come options.

```
box             manages boxes: installation, removal, etc.
cap             checks and executes capability
connect         connect to a remotely shared Vagrant environment
destroy         stops and deletes all traces of the vagrant machine
docker-exec     attach to an already-running docker container
docker-logs     outputs the logs from the Docker container
docker-run      run a one-off command in the context of a container
global-status   outputs status Vagrant environments for this user
halt            stops the vagrant machine
help            shows the help for a subcommand
init            initializes a new Vagrant environment by creating a Vagrantfile
login           log in to HashiCorp's Atlas
package         packages a running vagrant environment into a box
plugin          manages plugins: install, uninstall, update, etc.
port            displays information about guest port mappings
powershell      connects to machine via powershell remoting
provider        show provider for this environment
provision       provisions the vagrant machine
push            deploys code in this environment to a configured destination
rdp             connects to machine via RDP
reload          restarts vagrant machine, loads new Vagrantfile configuration
resume          resume a suspended vagrant machine
rsync           syncs rsync synced folders to remote machine
rsync-auto      syncs rsync synced folders automatically when files change
share           share your Vagrant environment with anyone in the world
snapshot        manages snapshots: saving, restoring, etc.
ssh             connects to machine via SSH
ssh-config      outputs OpenSSH valid configuration to connect to the machine
status          outputs status of the vagrant machine
suspend         suspends the machine
up              starts and provisions the vagrant environment
version         prints current and latest Vagrant version
```

Create your project directory, go into it and initialized the vagrant environment

vagrant init: command is use to get the vagrant sample file, which you can tweak later as per your convenience

mkdir Candycrush
cd Candycrush

vagrant init

It will create the necessary configuration file Vagrantfile in the project directory, open the configuration file and update the same if necessary as per your need.

vagrant up : This command will read our configuration file from project directory and spin up a virtual machine.

```
C:\Users\devops\Candycrush>vagrant up
Bringing machine 'default' up with 'virtualbox' provider...
==> default: Importing base box 'ubuntu/trusty32'...
==> default: Matching MAC address for NAT networking...
==> default: Checking if box 'ubuntu/trusty32' is up to date...
==> default: Clearing any previously set forwarded ports...
==> default: Clearing any previously set network interfaces...
==> default: Preparing network interfaces based on configuration...
    default: Adapter 1: nat
==> default: Forwarding ports...
    default: 22 (guest) => 2222 (host) (adapter 1)
==> default: Booting VM...
==> default: Waiting for machine to boot. This may take a few minutes...
    default: SSH address: 127.0.0.1:2222
    default: SSH username: vagrant
    default: SSH auth method: private key
    default: Guest Additions Version: 4.3.36
    default: VirtualBox Version: 5.0
==> default: Mounting shared folders...
    default: /vagrant => C:\Users\devops\Candycrush
```

vagrant box : is command used to see the current available boxes in your repository.

```
>vagrant box --help
Usage: vagrant box <subcommand> [<args>]

Available subcommands:
     add
     list
     outdated
     remove
     repackage
     update

For help on any individual subcommand run `vagrant box <subcommand> -h`

>vagrant box list
box-cutter/centos67-i386 (virtualbox, 2.0.18)
ubuntu/trusty32          (virtualbox, 20161130.0.0)
```

vagrant status : will help you to get the current status of your vagrant boxes

```
>vagrant status
Current machine states:

default                   running (virtualbox)
```

vagrant suspend : if you want to hibernate your VM for some reason, you can use this command, without halting system temporary you can suspend this system and save the machine state.

```
>vagrant suspend
==> default: Saving VM state and suspending execution...
```

vagrant resume: to bring the system out of hibernate mode, you can make use of this command, it will quickly bring back the system for use.

```
>vagrant resume
==> default: Resuming suspended VM...
==> default: Booting VM...
==> default: Waiting for machine to boot. This may take a few minutes...
    default: SSH address: 127.0.0.1:2222
    default: SSH username: vagrant
    default: SSH auth method: private key
==> default: Machine booted and ready!
==> default: Machine already provisioned. Run `vagrant provision` or use the `--
provision`
==> default: flag to force provisioning. Provisioners marked to run always will
still run.
```

vagrant halt : is command to stop your vagrant VM, it is useful if you are doing some maintenance and you need to bring down the system.

```
>vagrant halt
==> default: Attempting graceful shutdown of VM...
```

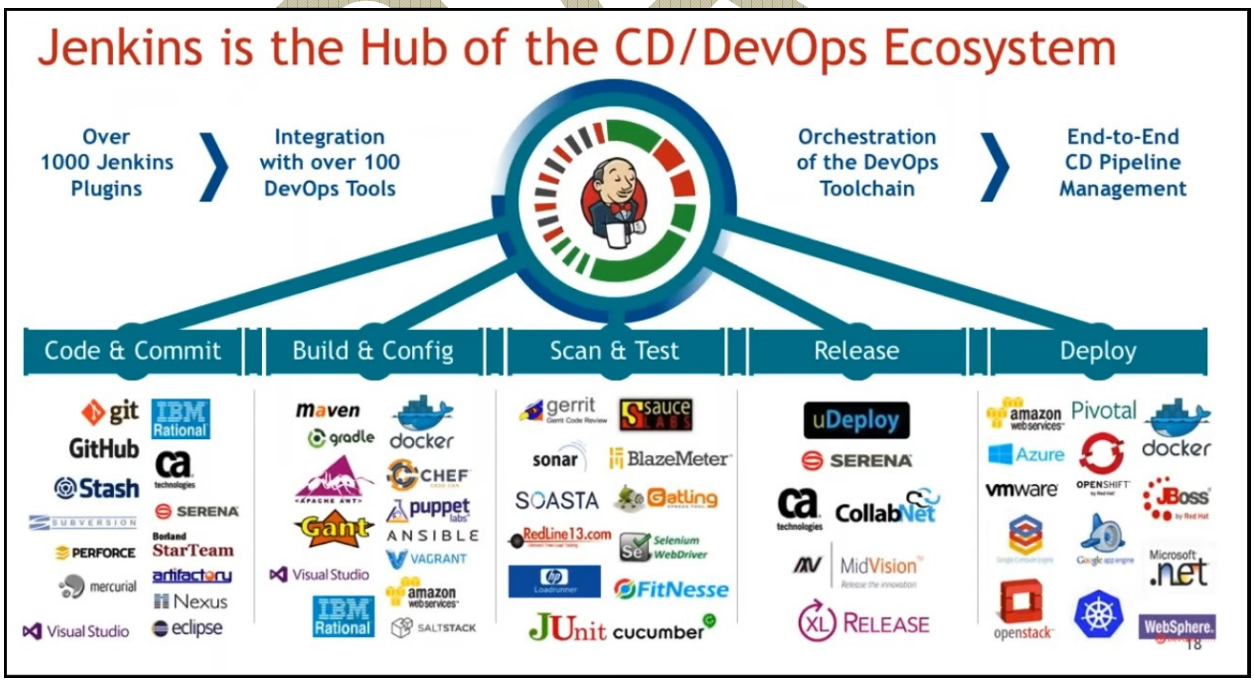
vagrant ssh : command is use to login your vagrant running VM via network, you can jump in your system with easy to do some update/patching or troubleshooting

```
>vagrant ssh
Welcome to Ubuntu 14.04.5 LTS (GNU/Linux 3.13.0-105-generic i686)

  System load:  0.47                Processes:           79
  Usage of /:   3.3% of 39.34GB     Users logged in:     0
  Memory usage: 15%                 IP address for eth0: 10.0.2.15
  Swap usage:   0%

  0 packages can be updated.
  0 updates are security updates.

vagrant@vagrant-ubuntu-trusty-32:~$ _
```

vagrant vm command prompt

Chapter 6: Jenkins

Jenkins is an open source automation server written in Java. Jenkins is a Continuous Integration server. Basically Continuous Integration is the practice of running your tests on a non-developer machine automatically every time someone pushes new code into the source repository.



Installation of Jenkins

You can download the jenkins latest version from https://jenkins.io/download/ , it is a cross platform tool, you will able to run the tool on linux, windows, mac OS.

Installation of Jenkins is very easy, download the latest war file, put it in some folder and bring the jenkins instance up.

This tools has a prior dependency of Java, so make sure you have java installed on your system. if it's not install the same with below command

```
$ sudo yum -y install java
$ mkdir -p ~/tools/jenkins
$ cd  ~/tools/jenkins
$ wget   http://mirrors.jenkins.io/war-stable/latest/jenkins.war
$ java -jar jenkins.war  [--httpPort=8080]
```

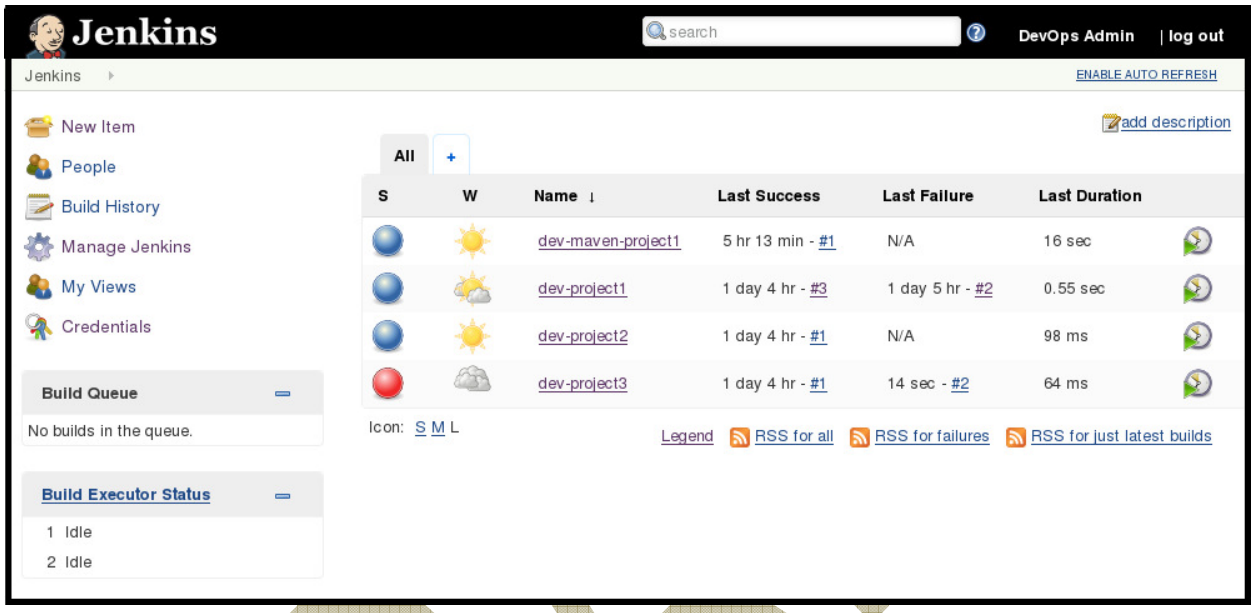
By default, jenkins will be starting at the port 8080, but you can change the port any time by passing parameter value to --httpPort

```
[devops@devops-training jenkins]$ java -jar jenkins.war
Running from: /home/devops/Desktop/tools/jenkins/jenkins.war
webroot: $user.home/.jenkins
Dec 20, 2016 5:22:56 PM Main deleteWinstoneTempContents
WARNING: Failed to delete the temporary Winstone file /tmp/winstone/jenkins.war
Dec 20, 2016 5:22:56 PM org.eclipse.jetty.util.log.JavaUtilLog info
INFO: Logging initialized @2591ms
Dec 20, 2016 5:22:56 PM winstone.Logger logInternal
INFO: Beginning extraction from war file
Dec 20, 2016 5:22:56 PM org.eclipse.jetty.util.log.JavaUtilLog warn
WARNING: Empty contextPath
Dec 20, 2016 5:22:56 PM org.eclipse.jetty.util.log.JavaUtilLog info
INFO: jetty-9.2.z-SNAPSHOT
Dec 20, 2016 5:22:59 PM org.eclipse.jetty.util.log.JavaUtilLog info
INFO: NO JSP Support for /, did not find org.eclipse.jetty.jsp.JettyJspServlet
Jenkins home directory: /home/devops/.jenkins found at: $user.home/.jenkins
Dec 20, 2016 5:23:01 PM org.eclipse.jetty.util.log.JavaUtilLog info
INFO: Started w.@791f145a{/,file:/home/devops/.jenkins/war/,AVAILABLE}{/home/dev
ops/.jenkins/war}
Dec 20, 2016 5:23:01 PM org.eclipse.jetty.util.log.JavaUtilLog info
INFO: Started ServerConnector@740cae06{HTTP/1.1}{0.0.0.0:8080}
Dec 20, 2016 5:23:01 PM org.eclipse.jetty.util.log.JavaUtilLog info
INFO: Started @7832ms
```

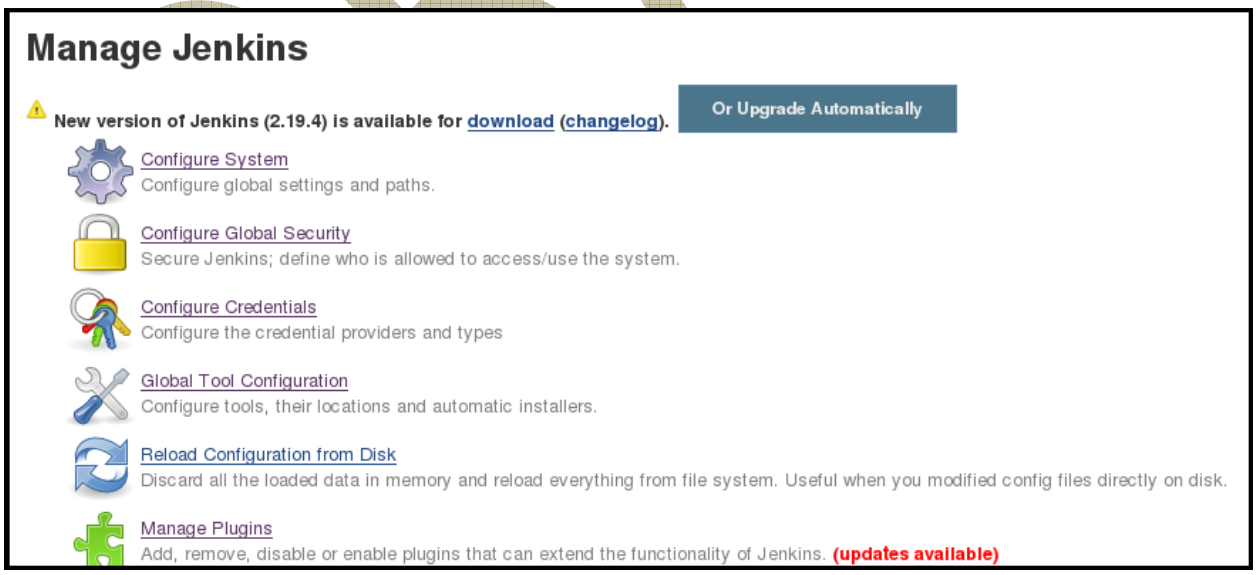
Jenkins Dashboard

The dashboard is every effective way to represent the build/job status, it has various legends, using that we will able to understand the job information. we can create new job by going inside the new item tab. If you are admin user, you will able to see the people and "Manage

Jenkins" links, inside the link you can perform many activities, like installation of plugins, creating users, Credentials configuration, Security configuration, Jenkins Logs for troubleshooting.

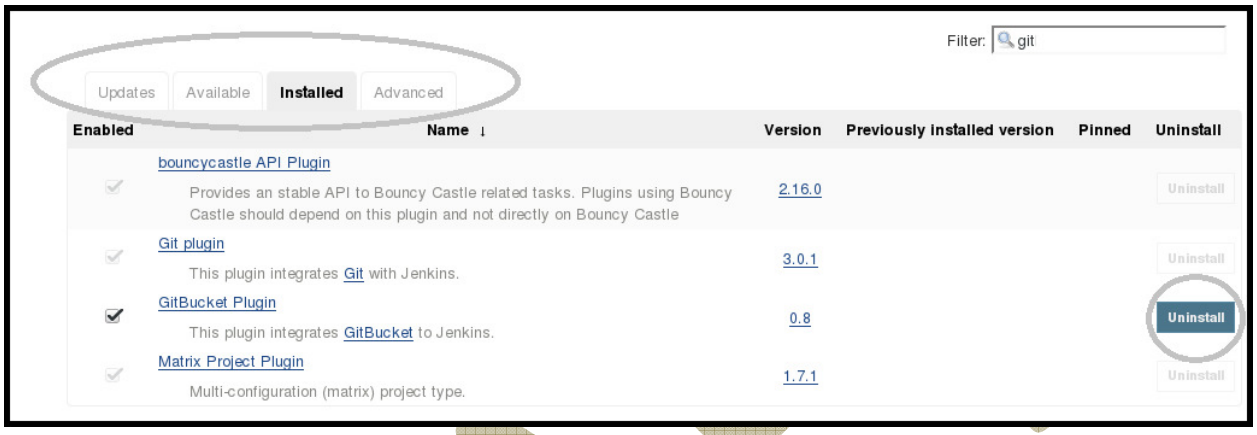


Managing Jenkins



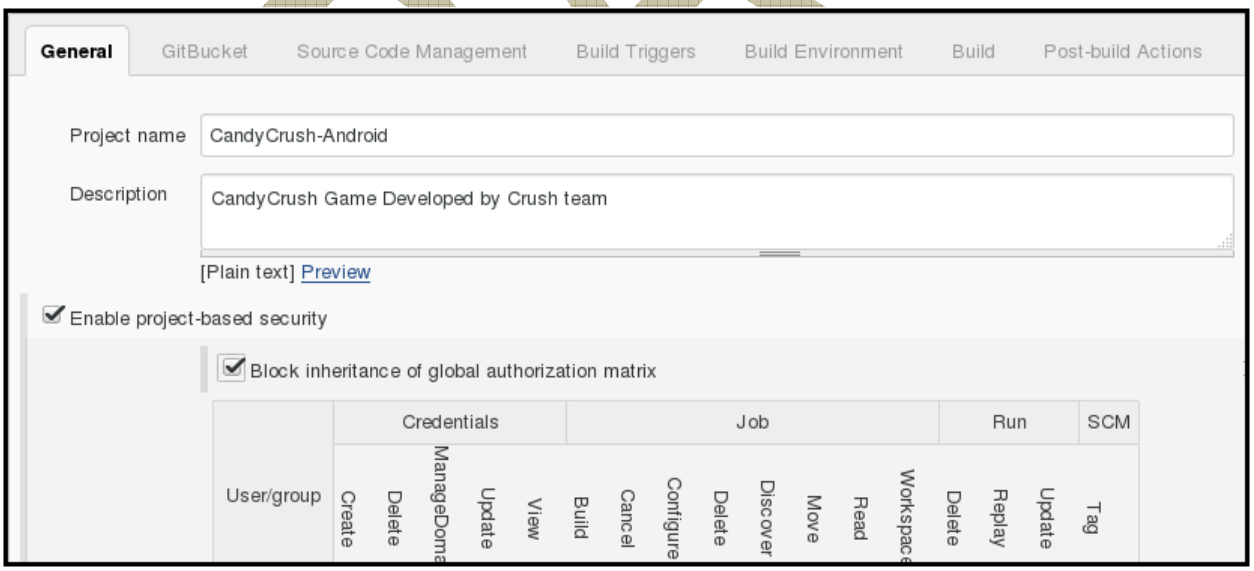
Jenkins Plugins

Jenkins supports 1000s of plugins with various tools, you just go to the plugin management section and search the plugin for download, installation or maintenance. Its the easiest way to do the plugin management. You can have various different categories like Updated, Available, Installed and Advanced.



Build Job/Item

Build has various phases, through which we need to do appropriate configuration, to make the build job successful.



Jenkins jobs are broken in various layers

General Section

Source Code Management Section

Build Triggers Section
Build Environment Section
Post Build Section

Using combination of all these section, we need to setup a job.

Chapter 7: MariaDB/MySQL Database

MariaDB is a community-developed fork of the MySQL relational database management system intended to remain free under the GNU GPL. It is notable for being led by the original developers of MySQL, who forked it due to concerns over its acquisition by Oracle.

What is a Database?

Before I can answer what MySQL means, I have to explain what a computer "database" means.

Essentially, where computers are concerned, a database is just a collection of data. Specialized (or "specialized" in US English) database software, like MySQL, are just programs that lets you store and retrieve that data as efficiently as possible.

A little analogy may help make it clearer why we use specialized database software. Think about the documents stored on your computer. If you were to save all your documents using a (brain-dead) file naming scheme like "1.doc", "2.doc", "3.doc", ... "9,999,999.doc" (etc), you will eventually face a problem of finding the right file if you're looking for a specific document. For example, if you're looking for a business proposal you made some time ago to XYZ Company, which file should you open? One way is to sequentially check every single file, starting from "1.doc", till you get the right data. But this is obviously a highly inefficient method of getting the right file. And it's primarily the result of an inefficient method of storing your data (ie, saving your files) in the first place.

Now, this is of course a ridiculous example. I mean, no one I know saves files with names like these, and even if so, there are many search software that can help you locate the correct file without your having to manually open every single one in sequence. But it serves to make the point that once you have a lot of data, if you don't have a good system of organizing it, finding the correct piece of data is a very time consuming operation. And it becomes more time consuming as the amount of data grows.

A database program is a type of computer software that is designed to handle lots of data, but to store them in such a way that finding (and thus retrieving) any snippet of data is more efficient than it would have been if you simply dumped them willy nilly all over the place. With such a database software, if you (say) keep a list of customers and their shipping addresses, entering and retrieving information about your one millionth customer will not take much longer (if at all) than entering and retrieving information about your 1st customer.

What is SQL? What is MySQL? What is PostgreSQL?

Many computer programs, including web-based programs like blogs, photo galleries and content management systems need to store and retrieve data. For example, blog software need to store the posts (ie, articles) you write, and retrieve them when a visitor goes to your site. Similarly, photo galleries store information about their pictures (for example, for sites that allow users to rate the photos, the numerical rating for each picture is stored in a database). Instead of reinventing the wheel and implementing their own system of storing and retrieving data, these software simply use the specialized database programs I mentioned earlier.

To make it easy for other programs to access data through them, many database software support a computer language called "SQL" (often pronounced as "sequel"). SQL was specially designed for such a purpose. Programs that want the database software to handle the low-level work of managing data simply use that language to send it instructions.

There are many databases that support the use of SQL to access their data, among them MySQL and PostgreSQL. In other words, MySQL is just the brand of one database software, one of many. The same goes for PostgreSQL. These two databases are very popular among programs that run on websites (probably because they are free), which is why you often see one or both of them being advertised in the feature lists of web hosts, as well as being listed as one of the "system requirements" for certain web software (like blogs and content management systems).

MariaDB installation

To install it you need to configure yum repo under /etc/yum.repos.d/MariaDB.repo

An example MariaDB.repo file for CentOS 7 is:

```
[mariadb]
name = MariaDB
baseurl = http://yum.mariadb.org/10.1/centos7-amd64
gpgkey=https://yum.mariadb.org/RPM-GPG-KEY-MariaDB
gpgcheck=1
```

Use below command, it will install the necessary packages and dependencies

$ sudo yum -y install mariadb-server,

Start the service by below command

$ sudo systemctl start mariadb.service

Once you start the service you will need to connect to database via a client with below command

$ mysql -u root -p

Note: The root is mysql root user and its nothing to do with Linux OS root user. By default you will not have the password to the database, just press enter and you will get connected to the database and receive a sql prompt

```
$ mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 174
Server version: 5.6.30 MySQL Community Server (GPL)

Copyright (c) 2000, 2016, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

Below are some typical commands

```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| test               |
|  prod-xxx-xxx      |
+--------------------+
 5 rows in set (0.04 sec)
```

below is the list of commands you can give with

use <dbname>;

show tables;

desc tables <tblname>;

create database <dbname>;

quit

Database backup and recovery

Database Backup:
backup: # mysqldump -u root -p[root_password] [database_name] > dumpfilename.sql

Database Recovery:
restore:# mysql -u root -p[root_password] [database_name] < dumpfilename.sql

Database Password reset/recover:

$ sudo /etc/init.d/mysql stop

$ sudo mysqld --skip-grant-tables &

$ mysql -u root mysql

$ UPDATE user SET Password=PASSWORD('YOURNEWPASSWORD') WHERE User='root'; FLUSH PRIVILEGES; exit;

Chapter 8: Web Server

Apache is a freely available **Web server** that is distributed under an "open source" license. Version 2.0 runs on most UNIX-based operating systems (such as Linux, Solaris, Digital UNIX, and AIX), on other UNIX/POSIX-derived systems (such as Rhapsody, BeOS, and BS2000/OSD), on AmigaOS, and on Windows 2000.

The Apache HTTP Server Project is an effort to develop and maintain an open-source HTTP server for modern operating systems including UNIX and Windows. The goal of this project is to provide a secure, efficient and extensible server that provides HTTP services in sync with the current HTTP standards.

The Apache HTTP Server ("httpd") was launched in 1995 and it has been the most popular web server on the Internet since April 1996. It has celebrated its 20th birthday as a project in February 2015.

Apache Directory Structure:

Under /etc/httpd web root directory, we have multiple directories available, in which conf directory is important for main configuration file, conf.d is for supportive application configuration, under conf.modules.d you will have all modules related configuration files, with the logs you will able to do debugging and they are found or linked under/var/log/messages.

```
$ tree httpd
httpd
├── conf
│   ├── httpd.conf
│   └── magic
├── conf.d
│   ├── autoindex.conf
│   ├── fcgid.conf
│   ├── userdir.conf
│   └── welcome.conf
├── conf.modules.d
│   ├── 00-base.conf
│   └── 00-dav.conf
├── logs -> ../../var/log/httpd
├── modules -> ../../usr/lib64/httpd/modules
├── run -> /run/httpd
├── sites-available
│   └── dummy.mycompany.com.conf
└── sites-enabled
    └── dummy.mycompany.com.conf -> /etc/httpd/sites-available/dummy.mycompany.com.conf
```

The web site related configuration files can be found under site-available directory, you can selectively enable the sites by creating symbolic link from sites-enabled to sites-available folder

Important Configuration Options under virtual host:

```
<VirtualHost *:80>
 ServerName www.mycompany.com
 DocumentRoot /var/www/www.mycompany.com
 CustomLog /var/log/httpd/www.mycompany.com -access.log combined
 ErrorLog /var/log/httpd/www.mycompany.com-error.log
</VirtualHost>
```

Apache Installation Steps:

The Apache web server is currently the most popular web server in the world, which makes it a great default choice for hosting a website.

We can install Apache easily using CentOS's package manager, yum. A package manager allows us to install most software pain-free from a repository maintained by CentOS. For our purposes, we can get started by typing these commands:

$ sudo yum install httpd

Since we are using a sudo command, these operations get executed with root privileges. It will ask you for your regular user's password to verify your intentions.

Afterwards, your web server is installed.

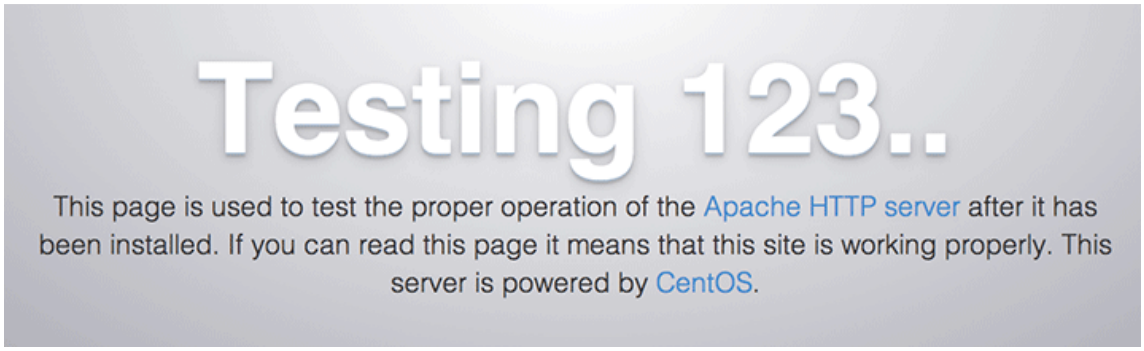
Once it installs, you can start Apache on your VPS:

sudo systemctl start httpd.service

You can do a spot check right away to verify that everything went as planned by visiting your server's public IP address in your web browser (see the note under the next heading to find out what your public IP address is if you do not have this information already):

http://your_server_IP_address/

You will see the default CentOS 7 Apache web page, which is there for informational and testing purposes. It should look something like this:



If you see this page, then your web server is now correctly installed. The last thing you will want to do is enable Apache to start on boot. Use the following command to do so:

sudo systemctl enable httpd.service

Chapter 9: Chef [Configuration Management Engine]

Chef is a powerful automation platform that transforms infrastructure into code. Whether you're operating in the cloud, on-premises, or in a hybrid environment, Chef automates how infrastructure is configured, deployed, and managed across your network, no matter its size.

The workstation is the location from which users interact with Chef. On the workstation users author and test cookbooks using tools such as Test Kitchen and interact with the Chef server using the knife and chef command line tools.

Nodes are the machines—physical, virtual, cloud, and so on—that are under management by Chef. The chef-client is installed on each node and is what performs the automation on that machine.

Use the Chef server as your foundation to create and manage flexible, dynamic infrastructure whether you manage 50 or 500,000 nodes, across multiple datacenters, public and private clouds, and in heterogeneous environments.

The Chef server acts as a hub for configuration data. The Chef server stores cookbooks, the policies that are applied to nodes, and metadata that describes each registered node that is being managed by the chef-client. Nodes use the chef-client to ask the Chef server for configuration details, such as recipes, templates, and file distributions. The chef-client then does as much of the configuration work as possible on the nodes themselves (and not on the

Chef server). This scalable approach distributes the configuration effort throughout the organization.

Chef Components:

Workstation: are configured to allow users to author, test, and maintain cookbooks. Cookbooks are uploaded to the Chef server from the workstation. Some cookbooks are custom to the organization and others are based on community cookbooks available from the Chef Supermarket

Cookbooks: Ruby is the programming language that is the authoring syntax for cookbooks. Most recipes are simple patterns (blocks that define properties and values that map to specific configuration items like packages, files, services, templates, and users). The full power of Ruby is available for when you need a programming language.

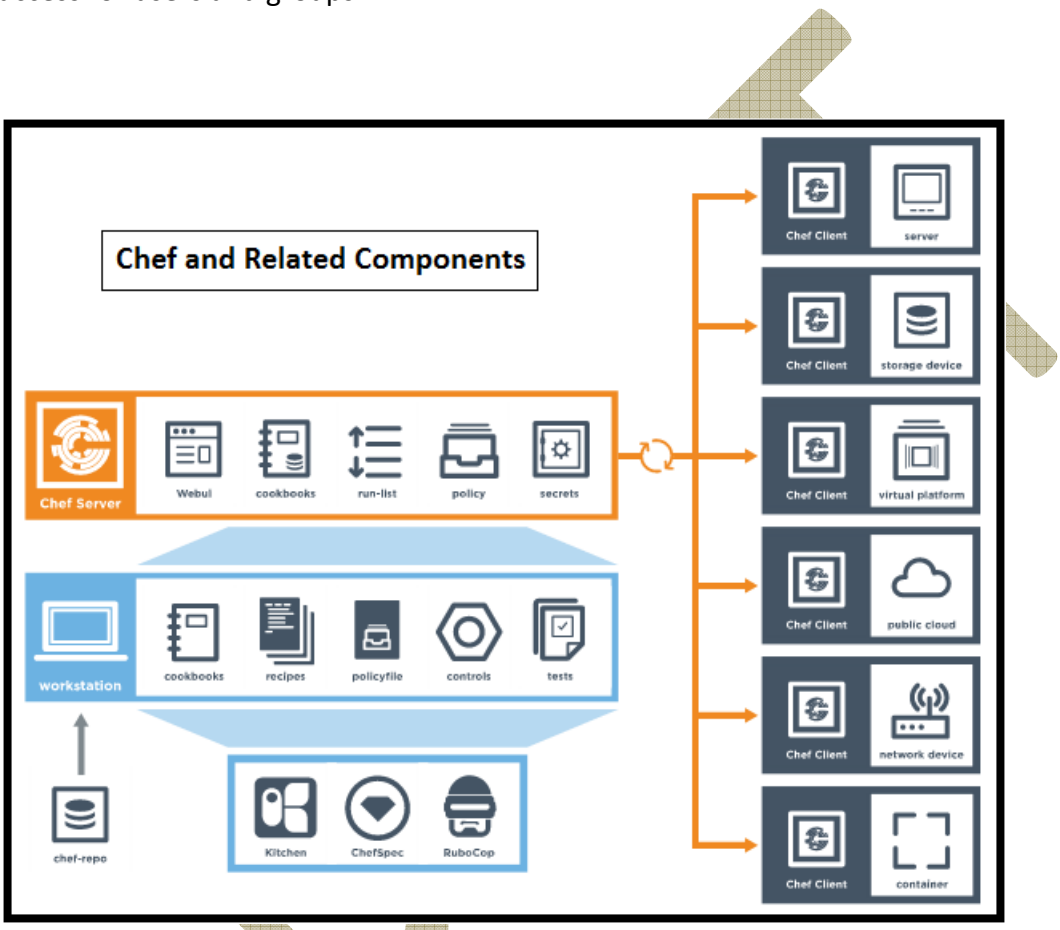
Node: A node is any machine—physical, virtual, cloud, network device, etc.—that is under management by Chef.

Chef-Client: A chef-client is installed on every node that is under management by Chef. The chef-client performs all of the configuration tasks that are specified by the run-list and will pull down any required configuration data from the Chef server as it is needed during the chef-client run.

Chef-Server: The Chef server acts as a hub of information. Cookbooks and policy settings are uploaded to the Chef server by users from workstations. (Policy settings may also be maintained from the Chef server itself, via the Chef management console web user interface.)

The chef-client accesses the Chef server from the node on which it's installed to get configuration data, performs searches of historical chef-client run data, and then pulls down the necessary configuration data. After the chef-client run is finished, the chef-client uploads updated run data to the Chef server.

Chef management console is the user interface for the Chef server. It is used to manage data bags, attributes, run-lists, roles, environments, and cookbooks, and also to configure role-based access for users and groups.



Chef-Supermarket : Chef Supermarket is the location in which community cookbooks are shared and managed. Cookbooks that are part of the Chef Supermarket may be used by any Chef user. How community cookbooks are used varies from organization to organization.

Chef Work Station Details:

A workstation is a computer running the Chef Development Kit (ChefDK) that is used to author cookbooks, interact with the Chef server, and interact with nodes.

The workstation is the location from which most users do most of their work, including:

Developing and testing cookbooks and recipes
Testing Chef code
Keeping the chef-repo synchronized with version source control
Configuring organizational policy, including defining roles and environments, and ensuring that critical data is stored in data bags
Interacting with nodes, as (or when) required, such as performing a bootstrap operation

The Chef Development Kit tooling encourages integration and unit testing, and defines workflow around cookbook authoring and policy, but it's important to note that you know best about how your infrastructure should be put together. Therefore, Chef makes as few decisions on its own as possible. When a decision must be made tools uses a reasonable default setting that can be easily changed. While Chef encourages the use of the tooling packaged in the Chef DK, none of these tools should be seen as a requirement or pre-requisite for being successful using Chef.

Cookbooks Details

A cookbook is the fundamental unit of configuration and policy distribution. A cookbook defines a scenario and contains everything that is required to support that scenario:

Recipes that specify the resources to use and the order in which they are to be applied

Attribute values
File distributions
Templates
Extensions to Chef, such as custom resources and libraries

The chef-client uses Ruby as its reference language for creating cookbooks and defining recipes, with an extended DSL for specific resources. A reasonable set of resources are available to the chef-client, enough to support many of the most common infrastructure automation scenarios; however, this DSL can also be extended when additional resources and capabilities are required.

Ohai Tool:

Ohai is a tool that is used to detect attributes on a node, and then provide these attributes to the chef-client at the start of every chef-client run. Ohai is required by the chef-client and must be present on a node. (Ohai is installed on a node as part of the chef-client install process.)

The types of attributes Ohai collects include (but are not limited to):

Platform details

Network usage

Memory usage

CPU data

Kernel data

Host names

Fully qualified domain names

Virtualization data

Cloud provider metadata

Other configuration details

Attributes that are collected by Ohai are automatic level attributes, in that these attributes are used by the chef-client to ensure that these attributes remain unchanged after the chef-client is done configuring the node.

Chef Policies:
Policy maps business and operational requirements, process, and workflow to settings and objects stored on the Chef server:

Roles define server types, such as “web server” or “database server”

Environments define process, such as “dev”, “staging”, or “production”

Certain types of data—passwords, user account data, and other sensitive items—can be placed in data bags, which are located in a secure sub-area on the Chef server that can only be accessed by nodes that authenticate to the Chef server with the correct SSL certificates

The cookbooks (and cookbook versions) in which organization-specific configuration policies are maintained

Chef Resource:
A resource is a statement of configuration policy that:

Describes the desired state for a configuration item

Declares the steps needed to bring that item to the desired state

Specifies a resource type—such as package, template, or service

Lists additional details (also known as resource properties), as necessary

Are grouped into recipes, which describe working configurations

Where a resource represents a piece of the system (and its desired state), a provider defines the steps that are needed to bring that piece of the system from its current state into the desired state.

Resource Functionalities:
Below are some resource functionalities:

Actions	install, restart, create and more
Properties	ignore_failure, retries, supports and more
Attributes	not_if, only_if
Arguments	:user, :group
Guard Interpreter	:bash, :perl, :default and more
Notifications	:before, :delayed, and more
Notifies	trigger to other resources

Chef Server Installation:

Prerequisites:
The Chef server has the following prerequisites:

An x86_64 compatible system architecture; Red Hat Enterprise Linux and CentOS may require updates prior to installation

A resolvable hostname that is specified using a FQDN or an IP address

A connection to Network Time Protocol (NTP) to prevent clock drift

A local mail transfer agent that allows the Chef server to send email notifications

Using cron and the /etc/cron.d directory for periodic maintenance tasks

Disabling the Apache Qpid daemon on CentOS and Red Hat systems

Optional. A local user account under which services will run, a local user account for PostgreSQL, and a group account under which services will run

Standalone Server Installation

The standalone installation of Chef server creates a working installation on a single server. This installation is also useful when you are installing Chef server in a virtual machine, for proof-of-concept deployments, or as a part of a development or testing loop.

Steps to install Chef server:

Download the package from https://downloads.chef.io/chef-server/.

Upload the package to the machine that will run the Chef server, and then record its location on the file system. The rest of these steps assume this location is in the /tmp directory.

As a root user, install the Chef server package on the server, using the name of the package provided by Chef. For Red Hat and CentOS:

rpm -Uvh /tmp/chef-server-core-<version>.rpm

After a few minutes, the Chef server will be installed.

Run the following to start all of the services:

chef-server-ctl reconfigure

Because the Chef server is composed of many different services that work together to create a functioning system, this step may take a few minutes to complete.

Run the following command to create an administrator:

chef-server-ctl user-create USER_NAME FIRST_NAME LAST_NAME EMAIL 'PASSWORD' --filename FILE_NAME

An RSA private key is generated automatically. This is the user's private key and should be saved to a safe location. The --filename option will save the RSA private key to the specified absolute path.

Run the following command to create an organization:

chef-server-ctl org-create short_name *'full_organization_name'* --association_user user_name --filename ORGANIZATION-validator.pem

The --association_user option will associate the user_name with the adminssecurity group on the Chef server.

An RSA private key is generated automatically. This is the chef-validator key and should be saved to a safe location. The --filename option will save the RSA private key to the specified absolute path.

Enable additional features of the Chef server! The packages may be downloaded directly as part of the installation process or they may be first downloaded to a local directory, and then installed.

Additional Components:

chef manage:

Use Chef management console to manage data bags, attributes, run-lists, roles, environments, and cookbooks from a web user interface.

On the Chef server, run:

chef-server-ctl install chef-manage

then:

chef-server-ctl reconfigure

and then:

chef-manage-ctl reconfigure

chef push jobs, chef reporting and others are additional components, please install it if you required them on your system.

Chef Workstation Setup

Download the package chefdk from https://downloads.chef.io/chefdk and install it

$sudo rpm -ivh chefdk-<version>.<arch>.rpm

$ mkdir -p chef-repo/.chef

$ vi chef-repo/.chef/knife.rb

```
current_dir = File.dirname(__FILE__)
log_level                :info
log_location             STDOUT
node_name                'user'
```

```
client_key              "#{current_dir}/user.pem"
validation_client_name  'organization-validator'
validation_key          "#{current_dir}/organization-validator.pem"
chef_server_url         'https://chef-server/organizations/organization'
cache_type              'BasicFile'
cache_options           ( :path => "#{ENV['HOME']}/.chef/checksums" )
cookbook_path           ["#{current_dir}/../cookbooks"]
```

Download the user.pem and organization-validator.pem file in your workstation under chef-repo/.chef directory.

$ cd chef-repo

$mkdir cookbooks

$ knife ssl fetch

$ knife ssl check

The final thing should look likes

tree -a ./chef-repo

```
$ tree -a ../chef-repo
../chef-repo
└── .chef
    ├── knife.rb
    ├── organizaton-validator.pem
    ├── trusted_certs
    │   └── servername.crt
    └── user.pem

2 directories, 4 files
```

Workstation validation:

$ cd chef-repo

$ knife client list # command
organization-validator # expected output

Workstation Commands:

cookbook options:

```
$ knife cookbook --help
** COOKBOOK COMMANDS **
knife cookbook bulk delete REGEX (options)
knife cookbook create COOKBOOK (options)
knife cookbook delete COOKBOOK VERSION (options)
knife cookbook download COOKBOOK [VERSION] (options)
knife cookbook list (options)
knife cookbook metadata COOKBOOK (options)
knife cookbook metadata from FILE (options)
knife cookbook show COOKBOOK [VERSION] [PART] [FILENAME] (options)
knife cookbook test [COOKBOOKS...] (options)
knife cookbook upload [COOKBOOKS...] (options)
```

create your first cookbook

previous version

$ knife cookbook create myfirstcookbook

OR with latest version

$ chef generate cookbook myfirstcookbook

Upload your cookbooks using upload option

$ knife cookbook upload myfirstcookbook

Get the current cookbook list from the server with list option

$ knife cookbook list

Node Setup:

You need to bootstrap the node to get the node under chef management.

Install a bare OS , give the node preferred IP address e.g. 1.2.3.4, create user "username" with password "password" and give the user necessary sudo permissions. Assuming your node name is node-web01. Then execute below command to bootstrap the node

$ knife bootstrap 1.2.3.4 -x username -p password --sudo

other bootstrap options as below

```
** BOOTSTRAP COMMANDS **
knife bootstrap [SSH_USER@]FQDN (options)
knife bootstrap windows ssh FQDN (options)
knife bootstrap windows winrm FQDN (options)
```

once the node is bootstrapped, attached the created cookbook under its run list.

$ knife node run_list add node-web01 myfirstcookbook

other node related options are as below

```
** NODE COMMANDS **
knife node bulk delete REGEX (options)
knife node create NODE (options)
knife node delete NODE (options)
knife node edit NODE (options)
knife node environment set NODE ENVIRONMENT
knife node from file FILE (options)
knife node list (options)
knife node run_list add [NODE] [ENTRY[,ENTRY]] (options)
knife node run_list remove [NODE] [ENTRY[,ENTRY]] (options)
knife node run_list set NODE ENTRIES (options)
knife node show NODE (options)
knife node status [<node> <node> ...]
```

Chef Structure:

Organization
Environment
Roles
Nodes
Resources
Recipes
Cookbooks

Organization

Completely independent tenants of Enterprise Chef
Share nothing with other organization
May represent different
Companies
Business Units
Departments

Environment

Development
Testing
Staging
Production
etc

Roles

It represent the type of server in your infrastructure
Load Balancer
Application Server
Database Cache
Database
Monitoring

Nodes

It represent the servers in your infrastructure
could be a physical or virtual server
my represent hardware that you own or compute instance in public or private cloud
could also be network hardware like switches, routers, etc.

Resources

A resource represent a piece of the system and its desired state

A package that should be installed
A service that should be running
A file that should be generated
A cron job that should be configured
A user that should be managed and much more

Recipes

Configuration file that describe resources and their desired state
Install and configure software components
Manages files
Deploy Applications
Execute other recipes
and more

Cookbooks

It is a collection of many items
Recipes are stored in cookbooks
cookbooks contains recipes, templates, files, custom resources and etc
best code re-use and modularity

Data Bags

A data bag is a container for items that represent information about your infrastructure that is not tied with single node

e.g.
users
group
application release information

Information to the data bags are imported from a file with .json format.

```
mkdir -p data_bags/users
vi  data_bags/users/devopsadmin.json

{
```

```
        "id": "devopsadmin",
        "comment": "DevOps Admin User",
        "uid": 2000,
        "gid": 0,
        "home": "/home/devopsadmin",
        "shell": "/bin/bash"
}
```

$ knife data_bag create devopsadmin

$ knife data_bag from file users devopsadmin.json

You can find more options with data bags as below

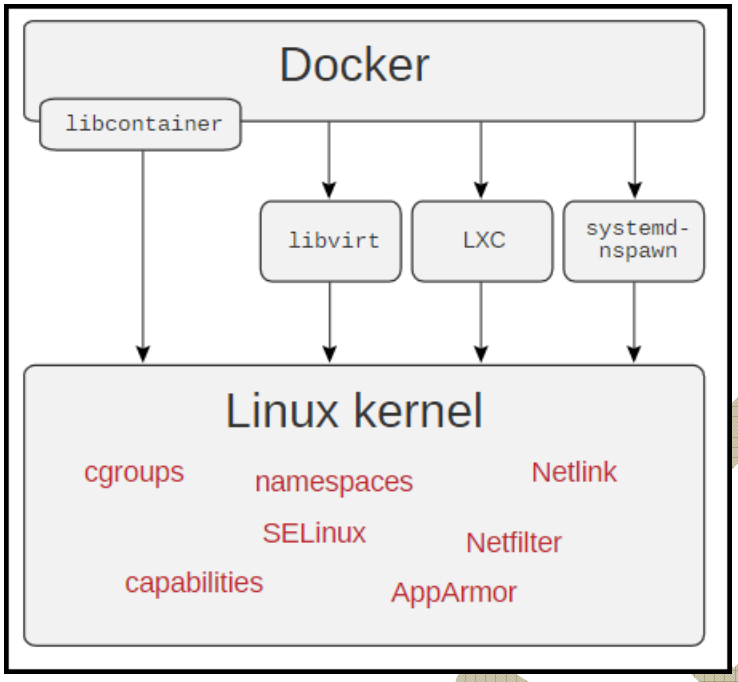
```
** DATA BAG COMMANDS **
knife data bag create BAG [ITEM] (options)
knife data bag delete BAG [ITEM] (options)
knife data bag edit BAG ITEM (options)
knife data bag from file BAG FILE|FOLDER [FILE|FOLDER..] (options)
knife data bag list (options)
knife data bag show BAG [ITEM] (options)
```

Chapter 10: Docker

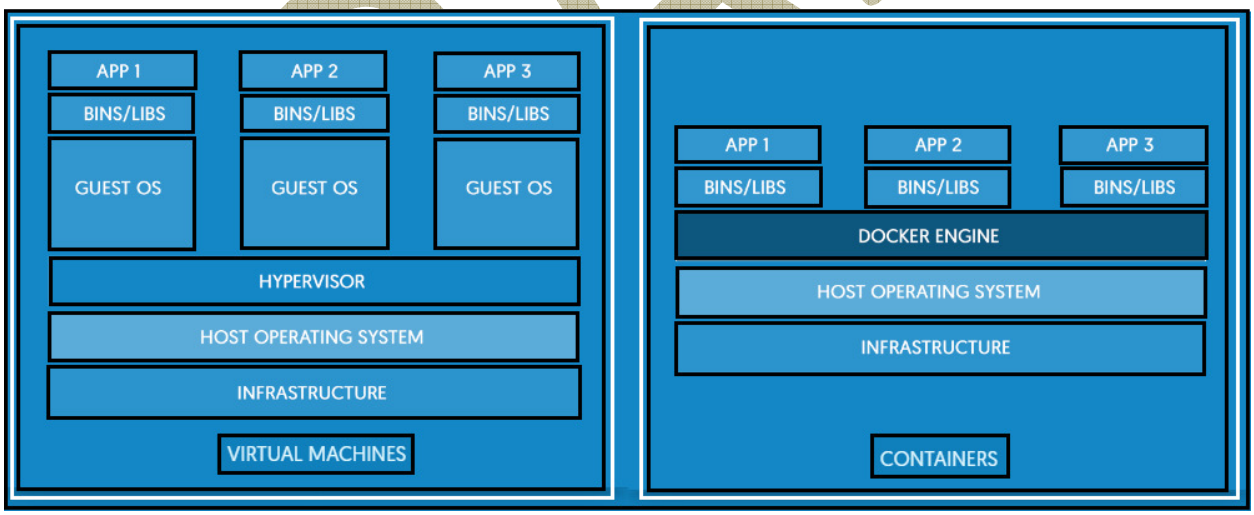
Docker is an open-source project that automates the deployment of applications inside software containers.

Docker containers wrap up a piece of software in a complete file system that contains everything it needs to run: code, runtime, system tools, system libraries – anything you can install on a server. This guarantees that it will always run the same, regardless of the environment it is running in.

Docker provides an additional layer of abstraction and automation of operating-system-level virtualization on Linux. Docker uses the resource isolation features of the Linux kernel such as cgroups and kernel namespaces, and a union-capable file system such as OverlayFS and others to allow independent "containers" to run within a single Linux instance, avoiding the overhead of starting and maintaining virtual machines.



If we compare Docker vs traditional Virtualization Machine, it will be appear to be like



Below are couple reference from Docker e-book publication

Containers are not VMs

Docker is one of the most successful open source projects in recent history, and organizations of all sizes are developing plans around how to containerize their applications. The first step in this journey is, of course, to understand what containers are, and what are their key benefits.

A natural response when first working with Docker containers is to try and frame them in terms of virtual machines. Oftentimes we hear people describe Docker containers as “lightweight VMs”. This is completely understandable, and many people have done the exact same thing when they first started working with Docker.

It’s easy to connect those dots as both technologies share some characteristics. Both are designed to provide an isolated environment in which to run an application. Additionally, in both cases that environment is represented as a binary artifact that can be moved between hosts. There may be other similarities, but these are the two biggest.

The key is that the underlying architecture is fundamentally different between the containers and virtual machines. The analogy we use here at Docker is comparing houses (virtual machines) to apartments (Docker containers).

Houses (the VMs) are fully self-contained and offer protection from unwanted guests. They also each possess their own infrastructure –plumbing, heating, electrical, etc. Furthermore, in the vast majority of cases houses are all going to have at a minimum a bedroom, living area, bathroom, and kitchen. It’s incredibly difficult to ever find a “studio house” – even if one buys the smallest house they can find, they may end up buying more than they need because

that’s just how houses are built.

Apartments (Docker containers) also offer protection from unwanted guests, but they are built around shared infrastructure. The apartment building (the server running the Docker daemon,

otherwise known as a Docker host) offers shared plumbing, heating, electrical, etc. to each apartment. Additionally apartments are offered in several different sizes – from studio to multi-bedroom penthouse. You're only renting exactly what you need.

Docker containers share the underlying resources of the Docker host. Furthermore, developers build a Docker image that includes exactly what they need to run their application: starting with the basics and adding in only what is needed by the application. Virtual machines are built in the opposite direction. They start with a full operating system and, depending on the application, developers may or may not be able to strip out unwanted components.

For a lot of people these concepts are easily grasped. However, even when someone understands the architectural differences between Docker containers and virtual machines, they will often still try and adapt their current thoughts and processes around VMs to containers.

> "How do I backup a container?"
> "What's my patch management strategy for my running containers?"
> "Where does the application server run?"

To many the light bulb moment comes when they realize that Docker is not a virtualization technology, it's an application delivery technology. In a VM-centered world, the unit of abstraction is a monolithic VM that stores not only application code, but often the stateful data.

A VM takes everything that used to sit on a physical server and just packs it into a single binary so it can be moved around. But it is still the same thing. With Docker containers the abstraction is the application; or more accurately a service that helps to make up the application. In a micro-services architecture, many small services (each represented as a single Docker container) comprise an application.

Applications are now able to be deconstructed into much smaller components which fundamentally changes the way they are initially developed, and then managed in production.

So, how does a sysadmin backup a Docker container? They don't.

The application data doesn't live in the container, it lives in a Docker volume that is shared between 1-N containers as defined by the application architecture. Sysadmins backup the data volume, and forget about the container. Optimally Docker containers are completely stateless and immutable.

Certainly patches will still be part of the sysadmin's world, but they aren't applied to running Docker containers. In reality if someone patched a running container, and then spun up new containers based on an unpatched image, serious chaos could ensue. Instead admins update their existing Docker image, stop their running containers, and start up new ones. Because a container can be spun up in a fraction off a second, these updates are done in

exponentially more quickly than they are with virtual machines.

Application servers translates into a service run inside of a Docker container. Certainly there may be cases where microservices-based applications need to connect to a non-containerized service, but for the most part standalone servers where application code is executed give way to one or more containers that provide the same functionality with much less overhead (and much better horizontal scaling).

Containers and VMs Together

So if containers are not VMs, a logical question is:

Can VMs and Docker containers coexist??

The answer is a resounding "**yes**."

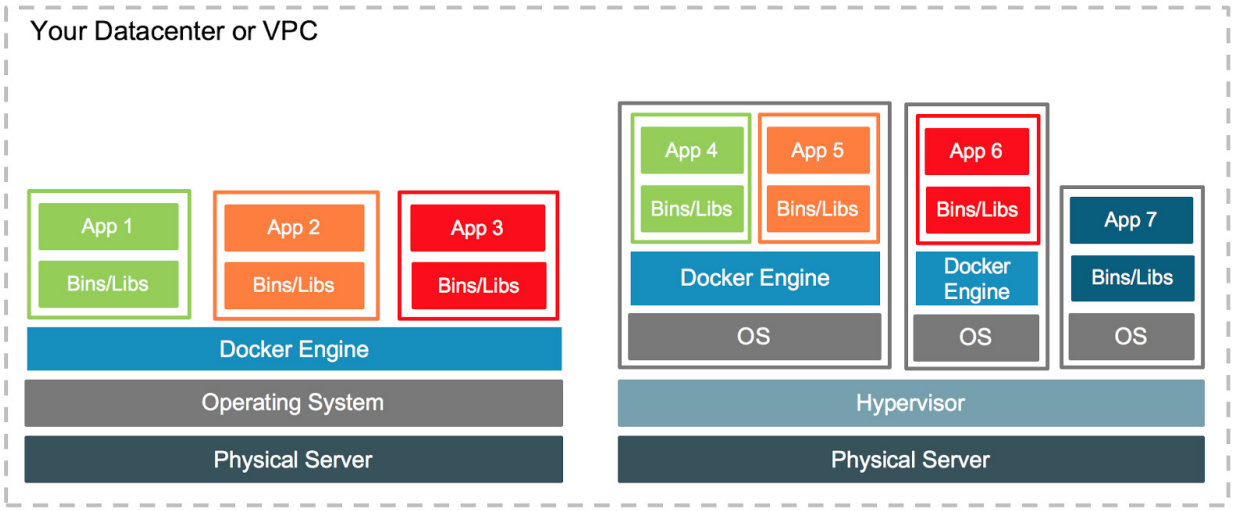
At the most basic level VMs (in all their forms) are a great place for Docker hosts to run. Whether it's a vSphere VM or a Hyper-V VM or an AWS EC2 instance, all of them will serve equally well as a Docker host. Depending on what you need to do, a VM might be the best

place to land those containers. But the great thing about Docker is that, it doesn't matter where you run containers – and it's totally up to you.

Another question that is often asked relates to whether or not Docker container-based services can interact with VM-based services. Again, the answer is absolutely yes. Running your application in a set of Docker containers doesn't preclude it from talking to the services running in a VM. For instance, your application may need to interact with a database that resides in a virtual machine. Provided that the right networking is in place, your application can interact with that database seamlessly.

Another area where there can be synergy between VMs and Docker containers is in the area of capacity optimization. VMs gained early popularity because the enabled higher levels of server

utilization. That's still true today. A virtualization host, for instance, can host VMs that may house Docker hosts, but may also host any number of traditional monolithic VMs. By mixing and matching Docker hosts with "traditional" VMs, sysadmins can be assured they are getting the maximum utilization out of their physical hardware.



Docker embraces running Docker hosts on a wide variety of virtualization and cloud platforms. Docker Cloud and Docker Datacenter can easily manage Docker hosts regardless of where

they run. And with Docker Machine you can provision new Docker hosts onto a wide variety of platforms including VMware vSphere, Microsoft Hyper-V, Azure, and AWS. One of the most powerful things about Docker is the flexibility it affords IT organizations.

The decision of where to run your applications can be based 100% on what's right for your business. You're not locked into any single infrastructure, you can pick and choose and mix and match in whatever manner makes sense for you organization. Docker hosts on vSphere? Great. Azure? Sure. Physical servers? Absolutely. With Docker containers you get a this great combination of agility, portability, and control

Physical or Virtual?

Virtual machines make great Docker hosts, but often companies wonder if containers would be better served running on bare metal physical servers.

And when they pose this question to Docker experts, the conversation goes something like this:

Docker Expert: It's not a question of "either / or" – that's the beauty of Docker. That choice is based solely on what's right for your application and business goals – physical or virtual,

cloud or on premise. Mix and match as your application and business needs dictate (and change).

User: But, surely you have a recommendation.

Docker Expert: I'm going to give you the two word answer that nobody likes: "It depends."

User: You're right, I don't like that answer.

Docker Expert: I kind of figured you wouldn't, but it really is the right answer.

There are tough questions in the world of tech, and the answer "It depends" can often be a way of avoiding them. But in the case of where to run your containerized applications it really is the best answer because no two applications are exactly the same, and no two companies have exactly the same business needs.

Any IT decision is based on a myriad of variables: Performance, scalability, reliability, security, existing systems, current skillsets, and cost (to name just a few). When someone sets out to decide how to deploy a Docker-based application in production all of these things need to be considered. Docker delivers on the promise of allowing you to deploy your applications seamlessly regardless of the underlying infrastructure.

Bare metal or VM. Datacenter or public cloud. Heck, deploy your application on bare metal in your data center and on VMs across multiple cloud providers if that's what is needed by your application or business.

The key here is that you're not locked into any one option. You can easily move your application from one infrastructure to another. There is essentially zero friction. But that freedom also makes the process of deciding where to run those applications seem more difficult than it really is. The answer is going to be influenced what you're doing today, and what you might need to do in the future.

And, while there is no easy answer to this question, there are a number of things to consider when it comes time to make your decision. The list here is probably far from complete, but hopefully it's enough to start a conversation and get the gears turning

Latency: Applications with a low tolerance for latency are going to do better on physical. This something we see quite a bit in financial services (trading applications are prime example).

Capacity: VMs made their bones by optimizing system load. If your containerized application doesn't consume all the capacity on a physical box, virtualization still offers a benefit here.

Mixed Workloads: Physical servers will run a single instance of an operating system. So, you if you wish to mix Windows and Linux containers on the same host, you'll need to use virtualization

Disaster Recovery: Again, like capacity optimizations, one of the great benefits of VMs are advanced capabilities around site recovery and high availability. While these capabilities may exist with physical hosts, there are a wider array of options with virtualization.

Existing Investments and Automation Frameworks: A lot of the organizations have already built a comprehensive set of tools around things like infrastructure provisioning. Leveraging this existing investment and expertise makes a lot of sense when introducing new elements.

Multitenancy: Some customers have workloads that can't share kernels. In this case VMs provide an extra layer of isolation compared to running containers on bare metal.

Resource Pools / Quotas: Many virtualization solutions have a broad feature set to control how virtual machines use resources. Docker provides the concept of resource constraints, but for bare metal you're kind of on your own.

Automation/APIs: Very few people in an organization typically have the ability to provision bare metal from an API. If the goal is automation you'll want an API, and that will likely rule out bare metal.

Licensing Costs: Running directly on bare metal can reduce costs as you won't need to purchase hypervisor licenses. And, of course, you may not even need to pay anything for the OS that hosts your containers.

In the end, there is something really powerful about being able to make a decision on where to run your application solely based on the technical merits of the platform AND being able to easily adjust that decision if new information comes to light.

In the end the question shouldn't be "bare metal OR virtual" – the question is which infrastructure makes the most sense for my application needs and business goals. So mix and match to create the right answer today, and know with Docker you can quickly and easily respond to any changes in the future

Docker Download:
Windows: https://download.docker.com/win/stable/InstallDocker.msi

Linux: yum -y install docker

Docker Images:
https://hub.docker.com/

docker commands

Through the command line you can work with the docker environment, example as below

```
$ docker --help
Usage: docker [OPTIONS] COMMAND [arg...]
       docker [ --help | -v | --version ]

A self-sufficient runtime for containers.

Options:

Commands:
    attach    Attach to a running container
    build     Build an image from a Dockerfile
    commit    Create a new image from a container's changes
    cp        Copy files/folders between a container and the local filesystem
    create    Create a new container
    diff      Inspect changes on a container's filesystem
    events    Get real time events from the server
    exec      Run a command in a running container
    export    Export a container's filesystem as a tar archive
    history   Show the history of an image
    images    List images
    import    Import the contents from a tarball to create a filesystem image
```

docker version

Docker version will help you to get the current Docker application version information

docker info

It will provide deeper insight about Docker, running systems, plugins, storage and much more

```
$ docker --version
Docker version 1.12.5, build 7392c3b


$ docker info
Containers: 1
 Running: 0
 Paused: 0
 Stopped: 1
Images: 1
Server Version: 1.12.5
Storage Driver: aufs
 Root Dir: /mnt/sda1/var/lib/docker/aufs
 Backing Filesystem: extfs
 Dirs: 3
 Dirperm1 Supported: true
Logging Driver: json-file
Cgroup Driver: cgroupfs
```

Docker Images

To run a docker instance you need a docker image, to find out what images you have use the image command

```
$ docker images
REPOSITORY    TAG       IMAGE ID        CREATED          SIZE
hello-world   latest    c54a2cc56cbb    6 months ago     1.848 kB
```

currently its showing only one image is available to use.

docker run

It will help you to start an instance/container from the Docker images. If the image is not available, it will download the image from the default Docker hub location. Lets assume you want to start ubuntu container and go to shell prompt, use below command :

--rm : it will remove the container once you are exited out

-t : give the pseudo terminal

-i: interactive session

```
$ docker run --rm -ti ubuntu /bin/bash
Unable to find image 'ubuntu:latest' locally
latest: Pulling from library/ubuntu
b3e1c725a85f: Pull complete
4daad8bdde31: Pull complete
63fe8c0068a8: Pull complete
4a70713c436f: Pull complete
bd842a2105a8: Pull complete
Digest: sha256:7a64bc9c8843b0a8c8b8a7e4715b7615e4e1b0d8ca3c7e7a76ec8250899c397a
Status: Downloaded newer image for ubuntu:latest
root@bc4764f1f26c:/#
```

Once you run the command you will directly login to the container, as shown in above image, you are in the container with id bc46x...

docker ps

You want to see the list of running containers, you can use docker ps command

```
$ docker ps
CONTAINER ID   IMAGE    COMMAND       CREATED         STATUS         PORTS    NAMES
bc4764f1f26c   ubuntu   "/bin/bash"   11 minutes      Up 10 minutes           trusting_col
```

Dockerfile

Now you want to create your own images or layered structure with docker, Dockerfile is the first thing you need to work with. In this file you will define all the container characteristics, application, environment variables, command and so on

cat Dockerfile

```
FROM ubuntu:latest
MAINTAINER Sunil Gadgil <info@mycompany.com>
USER root
ENV AP /data/app
ENV SCPATH /etc/supervisor/conf.d
RUN apt-get -y update
# The daemons
RUN apt-get -y install supervisor
RUN mkdir -p /var/log/supervisor
# Supervisor Configuration
ADD ./supervisord/conf.d/* $SCPATH/
# Application Code
ADD *.js* $AP/
WORKDIR $AP
RUN npm install
CMD ["supervisord", "-n"]
```