LAMP

=====

So when it comes to LAMP we are talking about { **Linux, Apache, MySql & PHP** }

LAMP is an **open source Web development platform** that **uses Linux as the operating system**, **Apache as the Web server**, **MySQL as the relational database** management system and **PHP as the object-oriented scripting language**.

Because the platform has four layers, LAMP is sometimes referred to as a LAMP stack. Stacks can be built on different operating systems.

The same setup in windows is called **WAMP**
The same setup in mac is called **MAMP**

Advantages Of LAMP

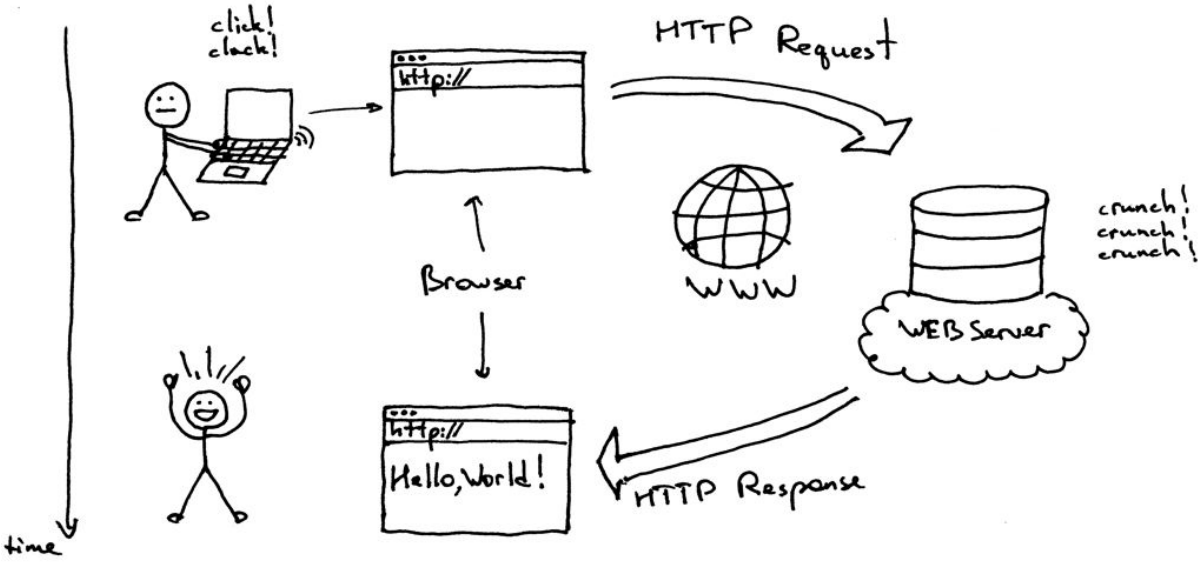
==========

- Open Source
- Easy to code with PHP
- Easy to deploy an application
- Develop locally
- Cheap Hosting
- Easy to build CMS application
    - Wordpress, Drupal, Joomla, Moodle etc

Web Server

==========

A web server is a program which serves web pages to users in response to their requests, which are forwarded by their computers' HTTP clients(Browsers).



All computers that host websites must have web server program.

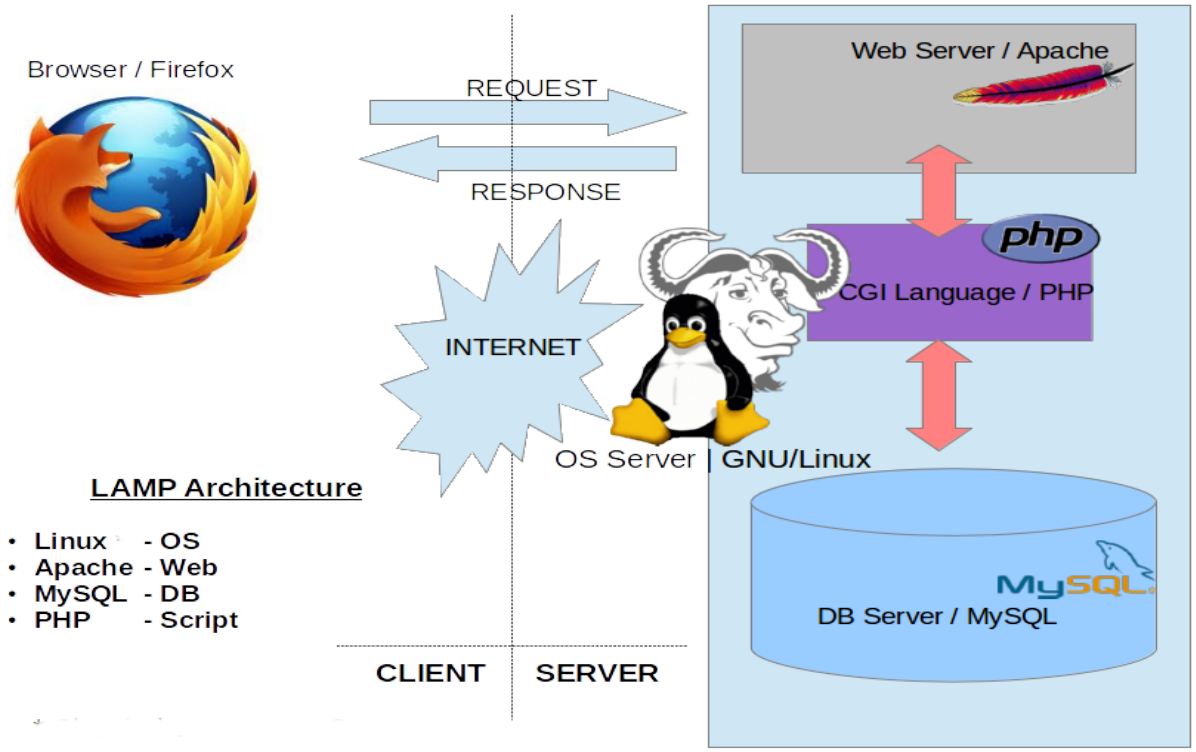
Purpose of Web server

==========

A web server's main purpose is to store web site files and broadcast them over the internet for you site visitors to see. In essence, a web server is simply a powerful computer that stores and transmits data via the internet.

Web servers are the gateway between the average individual and the world wide web.

LAMP Architecture



Linux :: We already talked about linux

Apache

=======

An **open source web server** used mostly for **Unix and Linux platforms**.

It is **fast, secure and reliable**.

Since 1996 Apache has been the most popular web server, presently **apache holds 49.5%** of market share i.e, 49.5% of all the websites followed by **Nginx 34%** and **Microsoft IIS 11%.**

Parameters for Apache (httpd)

==========

Packages	**-**	**httpd & mod_ssl**
Port	**-**	**80 443**
Protocol	**-**	**http https**
Server Root	**-**	**/etc/httpd**
Main config file	**-**	**/etc/httpd/conf/httpd.conf**
Document root	**-**	**/var/www/html {all our web pages }**
Logs	**-**	**/var/log/httpd/error_log**
		/var/log/httpd/access_log

What version of OS you have

==========

 # cat /etc/redhat-release

 # cat /etc/os-release

Checking httpd service is running or not

==========

6.x - Whenever you have problem with httpd {troubleshooting httpd}

 # service httpd status

 # netstat -ntpl | grep 80

 # ps -ef | grep httpd

7.x - Whenever you have problem with httpd {troubleshooting httpd}

 # systemctl status httpd

 # netstat -ntpl | grep 80

 # ps -ef | grep httpd

Now when you are doing server configuration, we need to be very careful while doing server configuration, coz sometimes you may do some typos and you are unaware why the server is not starting, so to confirm your config is correct use following command:

Check configuration file (httpd.conf) is correct or not

==========

 # httpd -t { 6.x and 7.x }

Installing httpd service

==========

rpm -qa | grep httpd

 # sudo yum -y install httpd

Let's **check status** of the web server:

systemctl status httpd

netstat -ntpl | grep 80

ps -ef | grep httpd

Let's **start the web server**:

systemctl start httpd

systemctl status httpd

netstat -ntpl | grep 80

ps -ef | grep httpd

Now we have started the service

If you see **# ls -l /var/www/html**, you have no files in there, let's create a index.html

 # cd /var/www/html

 # vi index.html {put some content}

 # vi /var/www/html/sample.php

```
<?php
echo “Today is ” . date(“Y/m/d”) . <br>”;
echo "Today is " . date("l");
?>
```

http://ip-address/sample.php

{ doesn't show php rendering coz php engine was not there }

 # sudo yum -y install php

http://ip-address/sample.php { php will be rendered }

Apache Virtual Hosting

==========

At the end of file add these lines

<VirtualHost *:81>
DocumentRoot /var/www/html/website2
</VirtualHost>

<VirtualHost *:82>
DocumentRoot /var/www/html/website3
</VirtualHost>

Nginx Server

==========

Nginx is a http server, which is used in many high traffic websites like GitHub, heroku etc.

The **3 imp features that nginx** gives is:
Load balancing, **Caching** and **Reverse proxying**.

In load balancing, say u have a high traffic website which gets 1000 requests per second may be more then that, if u have only one server all the 1000 requests will be severe by the single server where the response time will be decreased.

So we put more servers and distribute the load to all the servers equally.
So the 1000 requests will be distributed to 3 servers like 300 requests for each server assuming we got 3.

Reverse proxy, we are having multiple applications on the server, and u can only run one application on one port, say u running one app on port 80 now u can't run another application on port 80 we got to use another port. { app1.com ==> app1.com:80 }

Now we are running multiple applications we need give diff port right something like { app2.com:81 }

Now I don’t want to specify port 81 in url, I want app2.com, to deal with this problem we can use nginx.

Now nginx intercepts the requests and it will see that the following request is for app2.com now it routes this request to that application which is running on port 81. This is what reverse proxy is.

Forward Proxy & Reverse Proxy
==========

The word "proxy" describes someone or something acting on behalf of someone else.
In the computer realm, we are talking about one server acting on the behalf of another computer.

Forward Proxy
==========
Forward Proxy: Acting on behalf of a requestor (or service consumer)

"forward proxy" retrieves data from another web site on behalf of the original requestee.

In forward proxy we get to see 3 computers:

X = your computer, or "client" computer on the internet
Y = the proxy web site, proxy.example.org
Z = the web site you want to visit, www.example.net
Normally, one would connect directly from X —> Z.
However, in some scenarios, it is better for Y —> Z, on behalf of X, which chains as follows: X —> Y—> Z

Reasons why X would want to use a forward proxy server

==========
X is unable to access Z directly because

a) Someone with administration authority over X's internet connection has decided to block all access to site Z.

Employees at a large company have been wasting too much time on facebook.com, so management wants access blocked during business hours.

Torrent downloads can be blocked by our ISP’s so to view the torrent downloads we can use forward proxy.

b) The administrator of Z has blocked X.

Examples:
The administrator of Z has noticed hacking attempts coming from X, so the administrator has decided to block X's ip address.
Z is a forum web site. X is spamming the forum. Z blocks X.

Reverse Proxy
==========
Reverse Proxy: Acting on behalf of service/content producer

In reverse proxy we get to see 3 computers:

X = your computer, or "client" computer on the internet
Y = the reverse proxy web site, proxy.example.com
Z = the web site you want to visit, www.example.net

Normally, one would connect directly from X --> Z

However, in some scenarios, it is better for the administrator of Z to restrict or disallow direct access, and force visitors to go through Y first. So, as before, we have data being retrieved by Y --> Z on behalf of X, which chains as follows: X --> Y --> Z.

Reasons why Z would want to setup a reverse proxy server
==========

1) Z wants to force all traffic to its web site to pass through Y first.

a) Z has a large web site that millions of people want to see, but a single web server cannot handle all the traffic. So Z sets up many servers, and puts a reverse proxy on the internet that will send users to the server closest to them when they try to visit Z. This is part of how the Content Distribution Network (CDN) concept works.
Examples:
Apple Trailers uses Akamai
Jquery.com hosts its javascript files using CloudFront CDN (sample).

2) The administrator of Z is worried about retaliation for content hosted on the server and does not want to expose the main server directly to the public.

Load Balancing and Reverse Proxying with Nginx
==========

Make sure you have 3 machines setup with you:

M1 IP-ADD: 104.154.22.74
M2 IP-ADD: 104.198.254.34
M3 IP-ADD: 104.198.70.78

Machine1
========

yum -y install httpd
systemctl enable httpd
systemctl start httpd
allow port 80 in firewall
echo “MACHINE01” > /var/www/html/index.html
vi /var/www/html/index.html { MACHINE —> 1 }

Machine2

========

yum -y install httpd
systemctl enable httpd
systemctl start httpd
allow port 80 in firewall
echo “MACHINE02” > /var/www/html/index.html
vi /var/www/html/index.html { MACHINE —> 2 }

Machine3

========

yum -y install nginx
systemctl enable nginx
systemctl start nginx
allow port 80 in firewall
vi /etc/nginx/nginx.conf { delete everything under http section }

I am going to define a group of web servers with directive upstream and going to give this group a name as lbmysite.

Upstream defines a cluster that you can proxy requests to. It's commonly used for defining either a web server cluster for load balancing, or or an app server cluster for routing / load balancing.

nginx.conf

=========

user nginx;
worker_processes auto;
error_log /var/log/nginx/error.log;
pid /run/nginx.pid;

Load dynamic modules. See /usr/share/nginx/README.dynamic.
include /usr/share/nginx/modules/*.conf;

```
events {
        worker_connections 1024;
}

## remove and replace from http && remove old server block { }

http {

        log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
                '$status $body_bytes_sent "$http_referer" '
                '"$http_user_agent" "$http_x_forwarded_for"';

        access_log  /var/log/nginx/access.log  main;

        sendfile        on;
        tcp_nopush              on;
        tcp_nodelay             on;
        keepalive_timeout   65;
        types_hash_max_size 2048;

        include         /etc/nginx/mime.types;
        default_type            application/octet-stream;

        include /etc/nginx/conf.d/*.conf;

upstream lbmysite {
        server 104.197.127.92:90;
        server 35.194.22.247;
        }

server {
        listen          80 default_server;
        listen          [::]:80 default_server;
        location / {
        proxy_pass http://lbmysite;
        }
}
```

}

Database
========

Installation Version 5.6
==========
 # wget http://repo.mysql.com/mysql-community-release-el7-5.noarch.rpm

 # sudo rpm -ivh mysql-community-release-el7-5.noarch.rpm

 # sudo yum -y install mysql-server

Start the mariadb server **# sudo systemctl start mysqld**

Setting password **# sudo mysql_secure_installation**

To login to database **# sudo mysql -u root -p**

 # ps -ef | grep mysql
 # netstat -ntpl | grep 3306

How to connect to Database?

 # mysql -u root -p **{if db is same host}**
 # mysql -u root -p -h <server_ip> **{if db is on diff host**

CMS Application

==========

Download wordpress **# wget https://wordpress.org/latest.tar.gz**

 # sudo tar xvf wordpress.tar -C /var/www/html

 # sudo yum -y install php php-mysql

Login to phpmyadmin, and create user a database and user called wordpress something like that.

Give anything you want as username and password
Copy and paste the code given by wordpress into wp-config.php
Change the ownership of wordpress directory to apache:apache

GIT

Why Version Control ??
==========

Have you ever:

- Made a change to code, realised it was a mistake and wanted to revert back?
- Lost code and didn't have a backup of that code ?
- Had to maintain multiple versions of a product ?
- Wanted to see the difference between two (or more) versions of your code ?
- Wanted to prove that a particular change in code broke application or fixed a application ?
- Wanted to review the history of some code ?
- Wanted to submit a change to someone else's code ?
- Wanted to share your code, or let other people work on your code ?
- Wanted to see how much work is being done, and where, when and by whom ?
- Wanted to experiment with a new feature without interfering with working code ?

In these cases, and no doubt others, a version control system should make your life easier.

Key Points

========

- **Backup**
- **Collaboration**
- **Storing Versions**
- **Restoring Previous Versions**
- **Understanding What Happened**

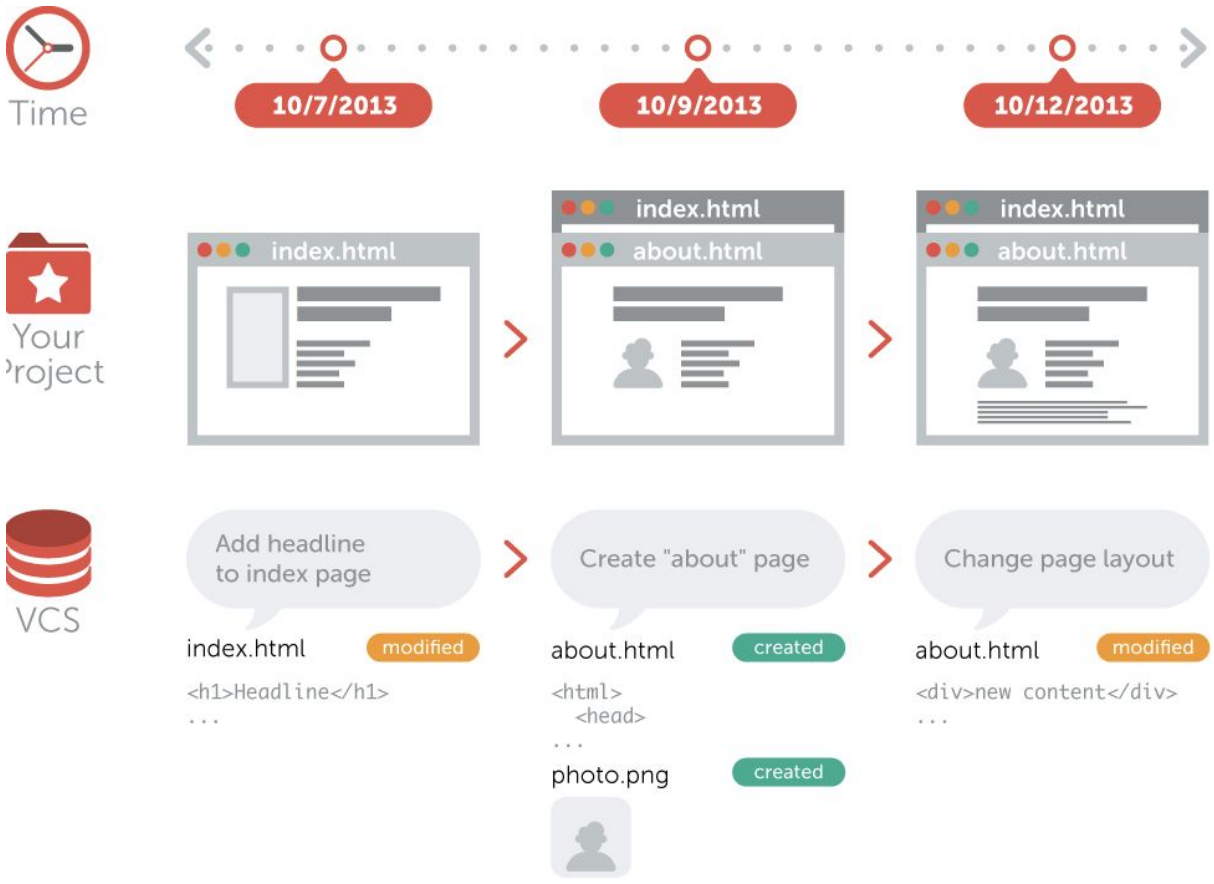
Version Control / Revision control / Source Control is is a software that helps software developers to work together and maintain a complete history of their work.

You can think of a version control system ("VCS") as a kind of "**database**".

It lets you save a snapshot of your complete project at any time you want. When you later take a look at an older snapshot ("version"), your VCS shows you exactly how it differed from the previous one.

A version control system **records the changes** you make to your project's files.

This is what version control is about. It's really as simple as it sounds.



Popular VCS

==========

Types of VCS

==========

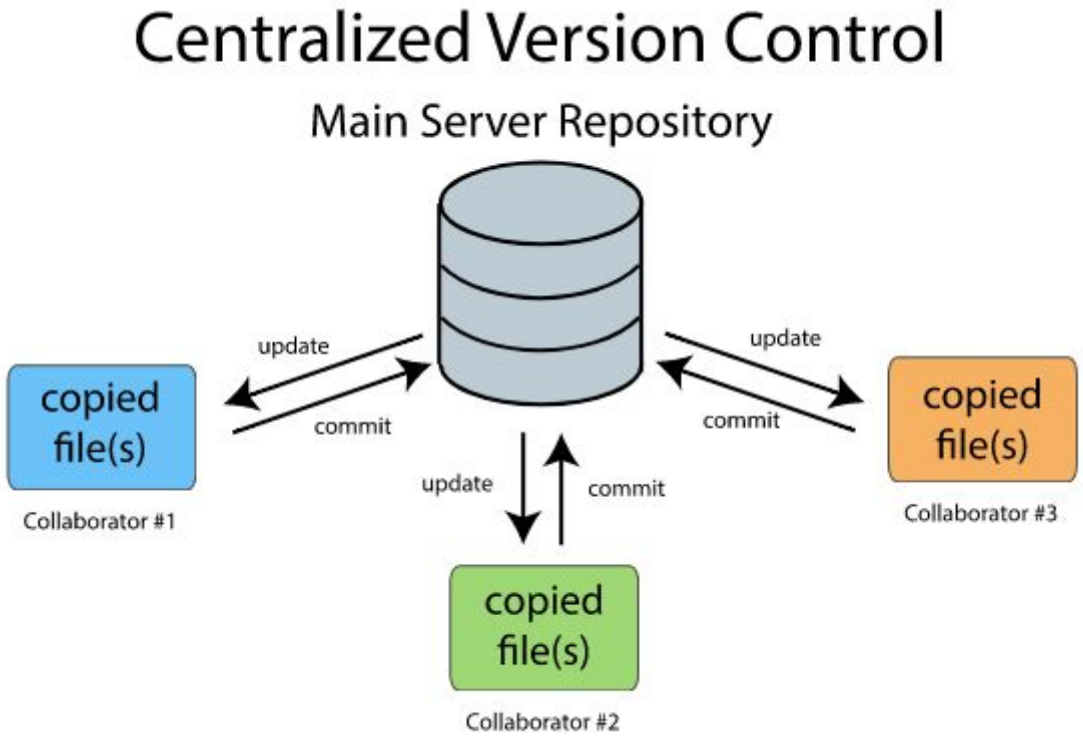
- **Centralized version control system (CVCS)**
- **Ex: CVS, SVN**
- **Distributed version control system (DVCS)**
- **Ex: Git, Mercurial**

Centralized Version Control System (CVCS)

==========

Uses a central server to store all files and enables team collaboration.

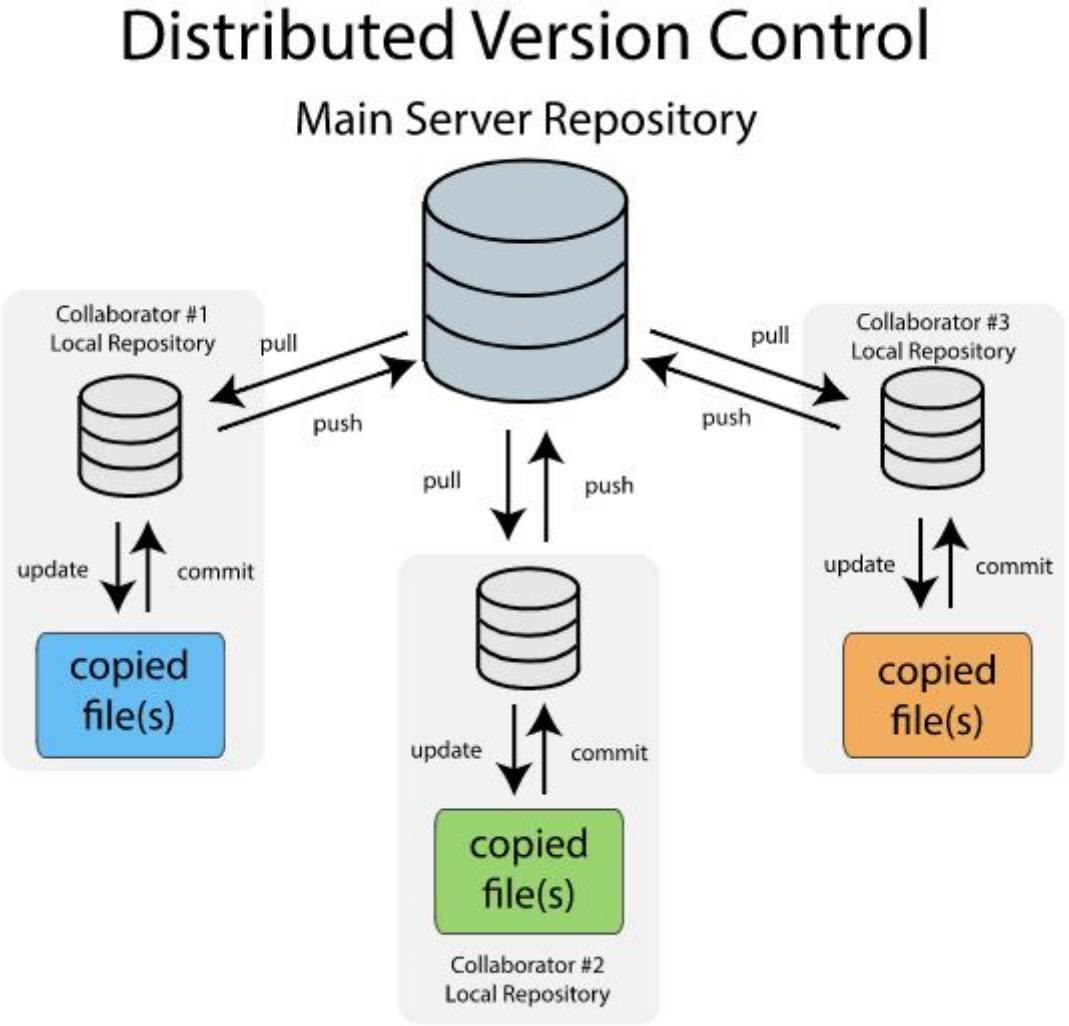
But the major drawback of CVCS is its **single point of failure,** i.e., failure of the central server. Unfortunately, if the central server goes down for an hour, then during that hour, no one can collaborate at all.

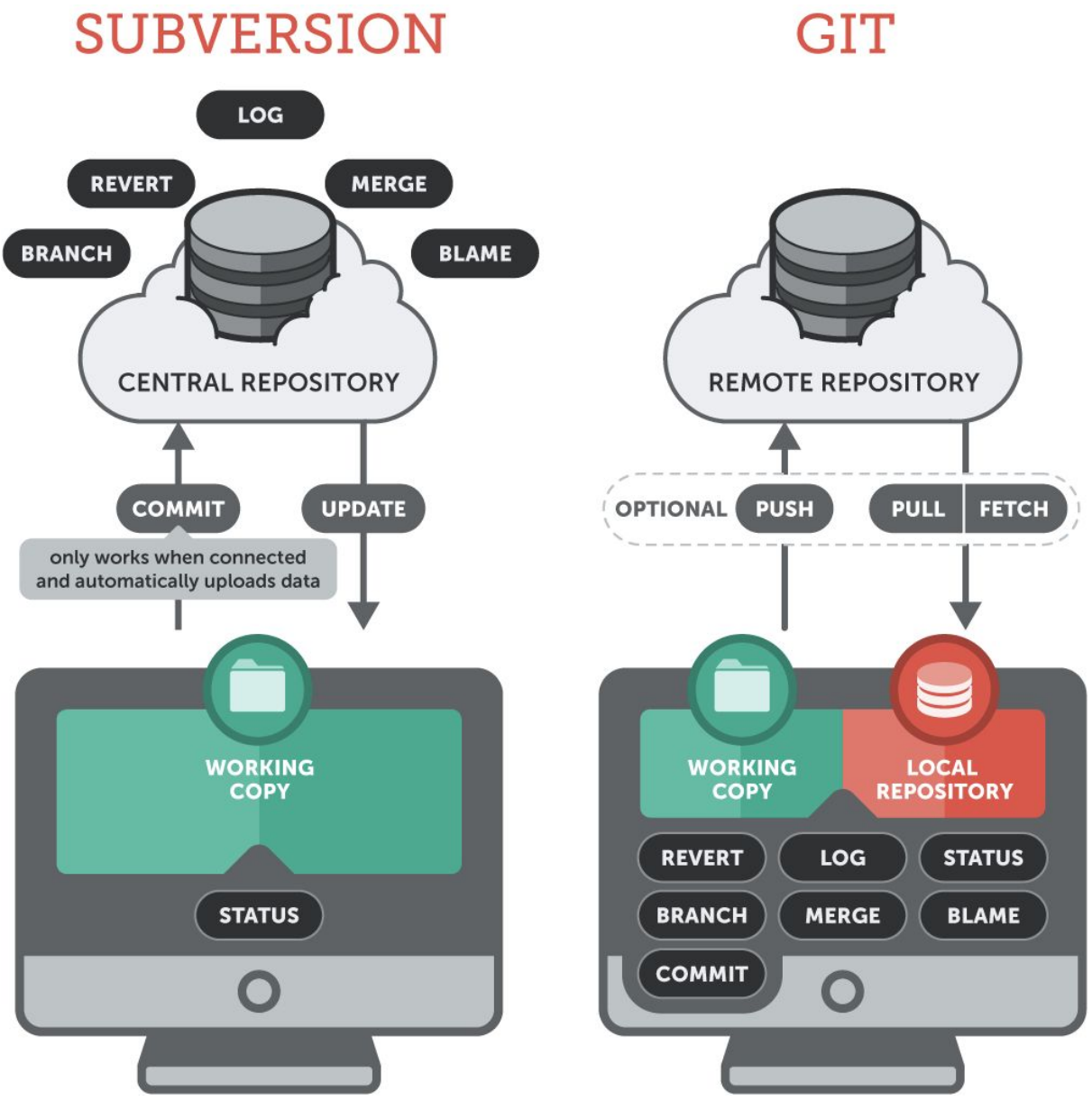


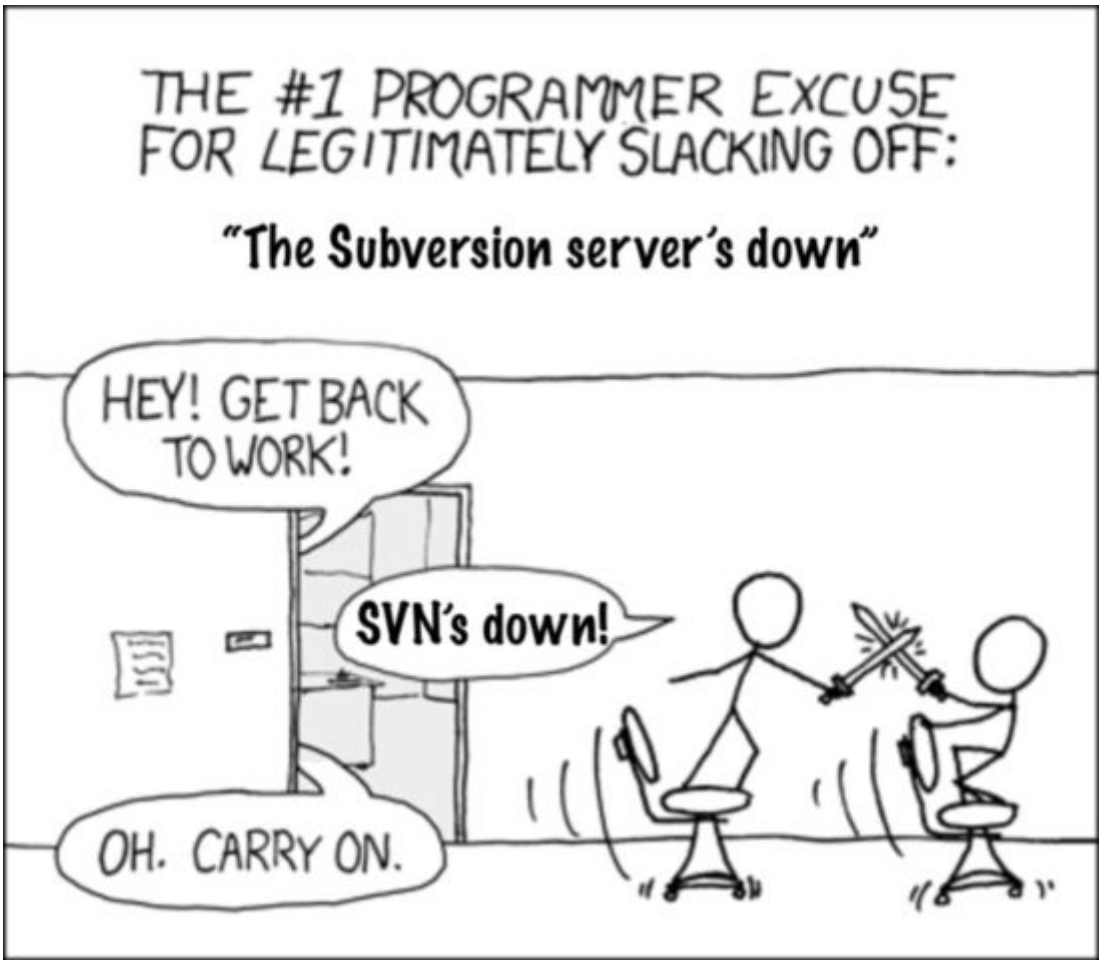
Distributed Version Control System (DVCS)

==========

DVCS does not rely on the central server and that is why you can perform many operations when you are offline. You can commit changes, create branches, view logs, and perform other operations when you are offline. You require network connection only to publish your changes and take the latest changes.




SUBVERSION
LOG
REVERT
MERGE
BRANCH
BLAME
CENTRAL REPOSITORY
COMMIT
UPDATE
only works when connected
and automatically uploads data
WORKING
COPY
STATUS
GIT
REMOTE REPOSITORY
OPTIONAL
PUSH
PULL
FETCH
WORKING
COPY
LOCAL
REPOSITORY
REVERT
LOG
STATUS
BRANCH
MERGE
BLAME
COMMIT



How the Typical VCS works

==========

A typical VCS uses something called **Two tree architecture**, this is what a lot of other VCS use apart from git.

Usually, a VCS works by having two places to store things:

1. **Working Copy**
2. **Repository**

These are our two trees, we call them trees because they represent a file structure.

Working copy [CLIENT] is the place where you make your changes.

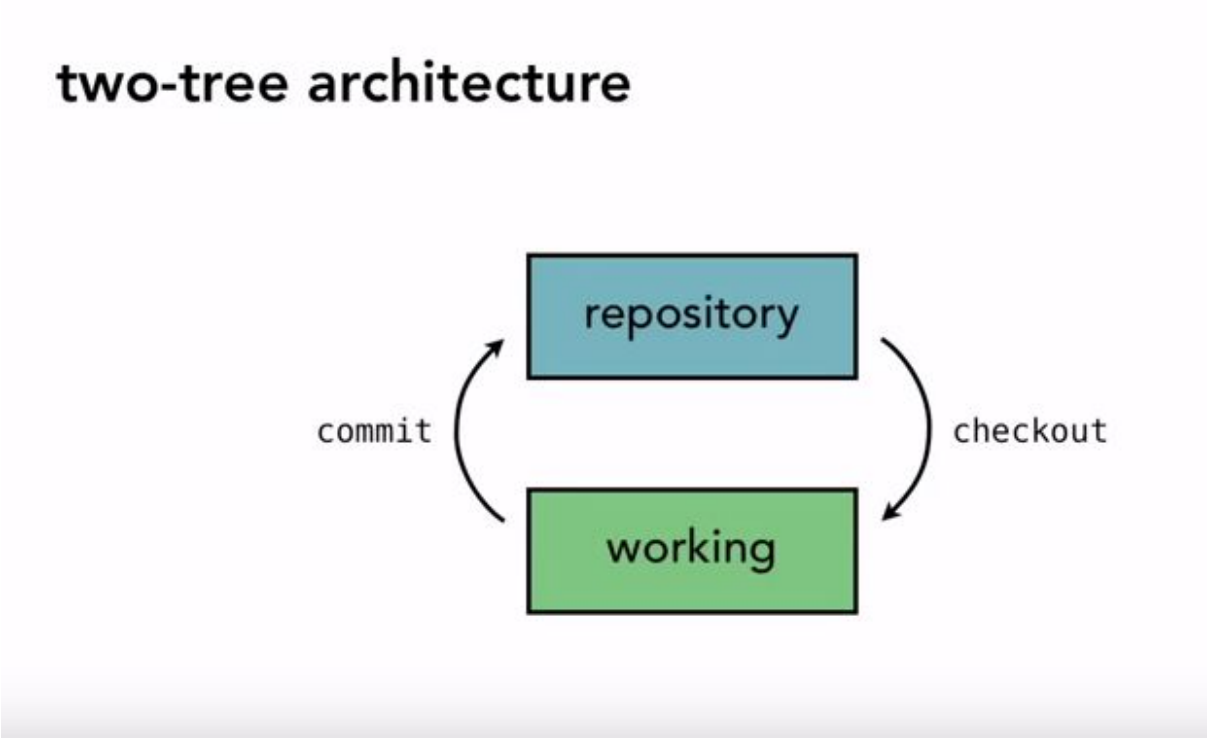
Whenever you edit something, it is saved in working copy and it is a physically stored in a disk.

Repository [SERVER] is the place where all the version of the files or commits, logs etc is stored. It is also saved in a disk and has its own set of files.

You cannot however change or get the files in a repository directly, in able to retrieve a specific file from there, you have to checkout

Checking-out is the process of getting files from repository to your working copy. This is because you can only edit files when it is on your working copy. When you are done editing the file, you will save it back to the repository by committing it, so that it can be used by other developers.

Committing is the process of putting back the files from working copy to repository.



Hence, this architecture is called **2 Tree Architecture.**

Because you have two tree in there **Working Copy** and **Repository**.

The famous VCS with this kind of architecture is Subversion or SVN.

How the Distributed DVCS works

==========

Unusually, a DVCS works by having three places to store things:

1. **Working Copy**
2. **Staging**
3. **Repository**

As Git uses Distributed version control system, So let’s talk about Git which will give you an understanding of DVCS.

Git was initially designed and developed by Linus Torvalds in 2005 for Linux kernel development. Git is an Open Source tool.

History

========

For developing and maintaining Linux Kernel, Linus Torvalds used **BitKeeper** which is also one of the VCS, and is open source till 2004.

So instead of depending on other tools, they developed their own VCS.

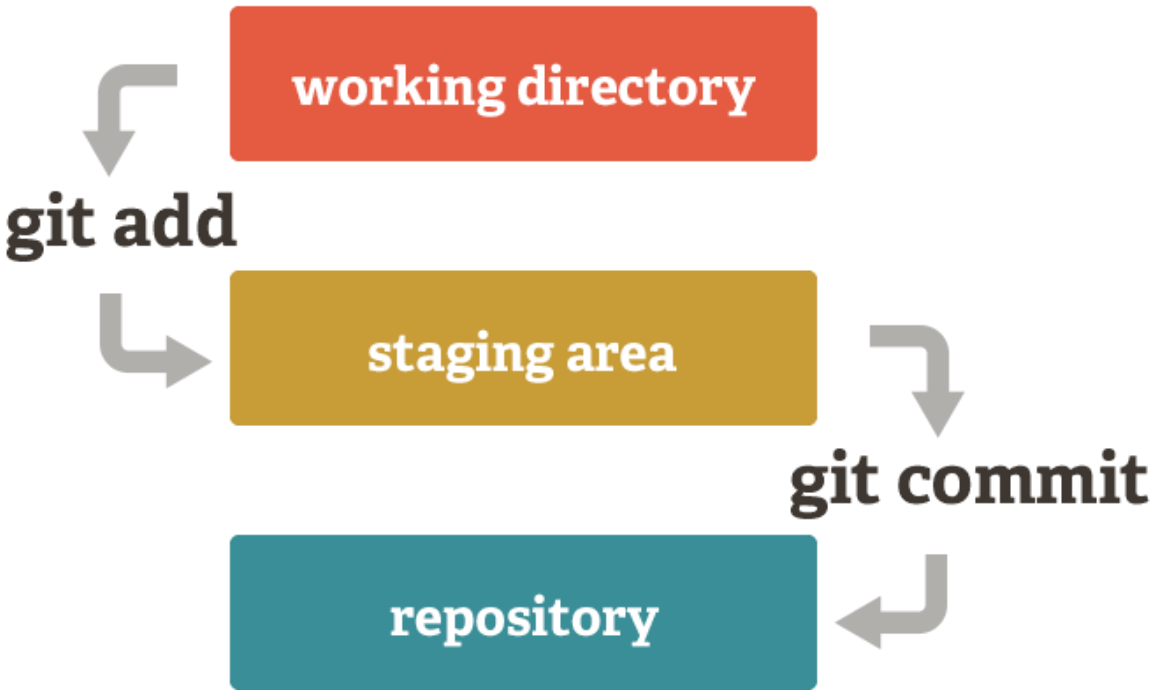
Just see the wiki of Git.

Git Architecture

==========

Git uses three tree architecture.

Well interestingly Git has the **Working Copy** and **Repository** as well but it has added an extra tree **Staging** in between:



As you can see above, there is a new tree called **Staging**.

What this is for ?

This is one of the fundamental difference of Git that sets it apart from other VCS, this **Staging tree** (usually termed as **Staging area**) is a place where you prepare all the things that you are going to commit.

In Git, you don't move things directly from your working copy to the repository, you have to stage them first, one of the main benefits of this is, **to break up your working changes into smaller, self-contained pieces.**

To stage a file is to prepare it for a commit.

Staging allows you finer control over exactly how you want to approach version control.

Advantages Of Git

==========

Git works on most of OS: Linux, Windows, Solaris and MAC.

Installing Git

==========

- Download Git {git website}

To check if git is available or not use:

 # rpm -qa | grep git

 # sudo yum install git

 # git --version

Setting the Configuration

==========

 # git config --global user.name "Ravi Krishna"

 # git config --global user.email "info@gmail.com"

git config --list

NOTE :: The above info is not the authentication information.

What is the need of git config

==========

When we setup git and before adding bunch of files, We need to fill up username & email and it's basically git way of creating an account.

Working with Git

==========

Getting a Git Repository

 # mkdir website

Initialising a repository into directory, to initialize git we use:

 # git init

The purpose of Git is to manage a project, or a set of files, as they change over time. Git stores this information in a data structure called a repository.

git init is only for when you create your own new repository from scratch.

It turns a directory into an empty git repository.

Let's have some configuration set:

git config --global user.name "Ravi"

git config --global user.email "ravi@digital-lync.com"

Taking e-commerce sites as example.

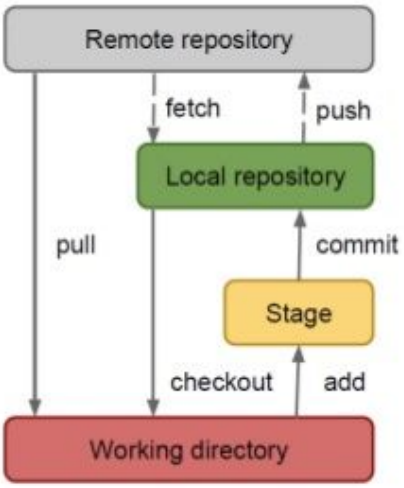
Basic Git Workflow

==========

1. You modify files in working directory
2. You stage files, adding snapshots of them to your staging area.
3. You do a commit, which takes the files as they are in the staging area and stores that snapshot to your git repository.

Understanding of Workflow

- Obtain a repository
  - *git init* or *git clone*
- Make some changes
- Stage your changes
  - *git add*
- Commit changes to the local repository
  - git commit –m "My message"
- Push changes to remote
  - git push remotename remotebranch



mkdir website

git init

git status {Branches will talk later}

vi index.html

{put some tags <html><title><h1><body> just structure}

git status

git add index.html {staged the changes}

git commit -m "Message" {moves file from staging area to local repo}

git status

You can skip the staging area by # git commit -a -m "New Changes"

Commit History - How many Commits have happened ??

==========

To see what commits have been done so far we use a command:

 # git log

It gives commit history basically commit number, author info, date and commit message.

Want to see what happend at this commit, zoom in info we use:

 # git show <commit number>

Let’s understand this **commit number**

This is sha1 value randomly generated number which is 40 character hexadecimal number which will be unique.

Let’s change the title in index.html and go with

git add status commit

git log {gives the latest commit on top and old will get down}

Git diff

========

Let’s see, the diff command gives the difference b/w two commits.

 # git diff xxxxxx..xxxxxx

You can get diff b/w any sha's like sha1..sha20.

Now our log started increasing, like this the changes keep on adding file is one but there are different versions of this file.

git log --since YYYY-MM-DD

git log --author ravi

git log --grep HTML { commit message }

git log --oneline

Git Branching

==========

In a collaborative environment, it is common for several developers to share and work on the same source code.

Some developers will be fixing bugs while others would be implementing new features.

Therefore, there has got to be a manageable way to **maintain different versions** of the same code base.

This is where the branch function comes to the rescue. **Branch allows** each developer **to branch out from the original code base and isolate their work from others.** Another good thing about branch is that **it helps Git to easily merge** the versions later on.

It is a common practice to create a new branch for each task (eg. bug fixing, new features etc.)

Branching means you diverge from the main line(master-working copy of application) of development and continue to do work without messing with that main line.

Basically, you have your master branch and you don't want to mess anything up on that branch.

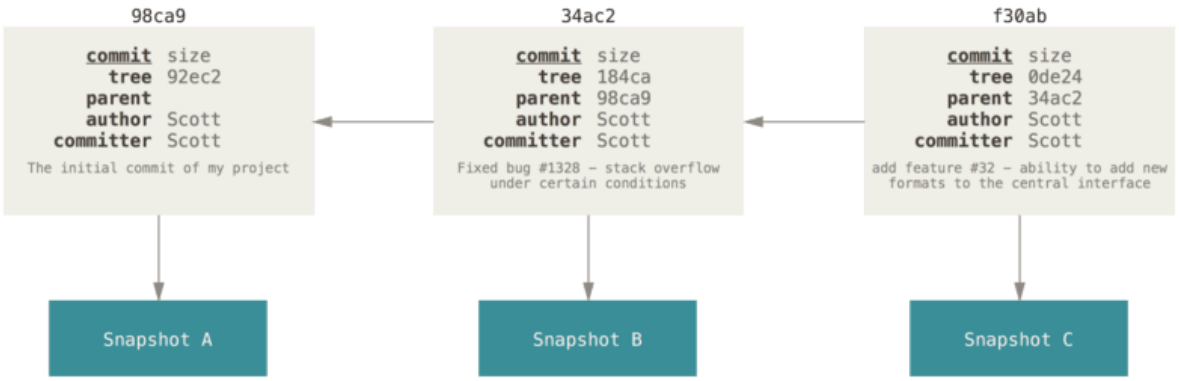
In many VCS tools, **branching** is an **expensive process**, often requiring you to create a new copy of your source code directory, which can take a long time for large projects.

Some people refer to **Git's branching model as its "killer feature"** and it certainly sets Git apart in the VCS community.

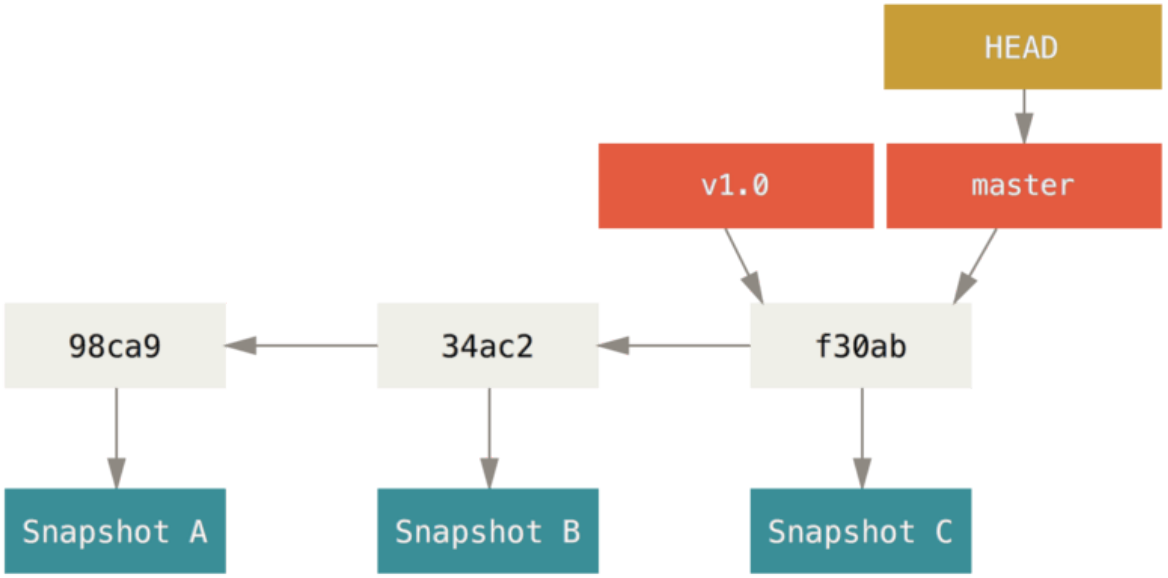
Why is it so special?

The way Git branches is incredibly **lightweight**, making branching operations nearly **instantaneous**, and switching back and forth between branches generally just as fast.

When we make a commits, this is how git stores them.



A branch in Git is simply a lightweight movable pointer to one of these commits. The default branch name in Git is master. As you start making commits, you're given a master branch that points to the last commit you made. Every time you commit, it moves forward automatically.

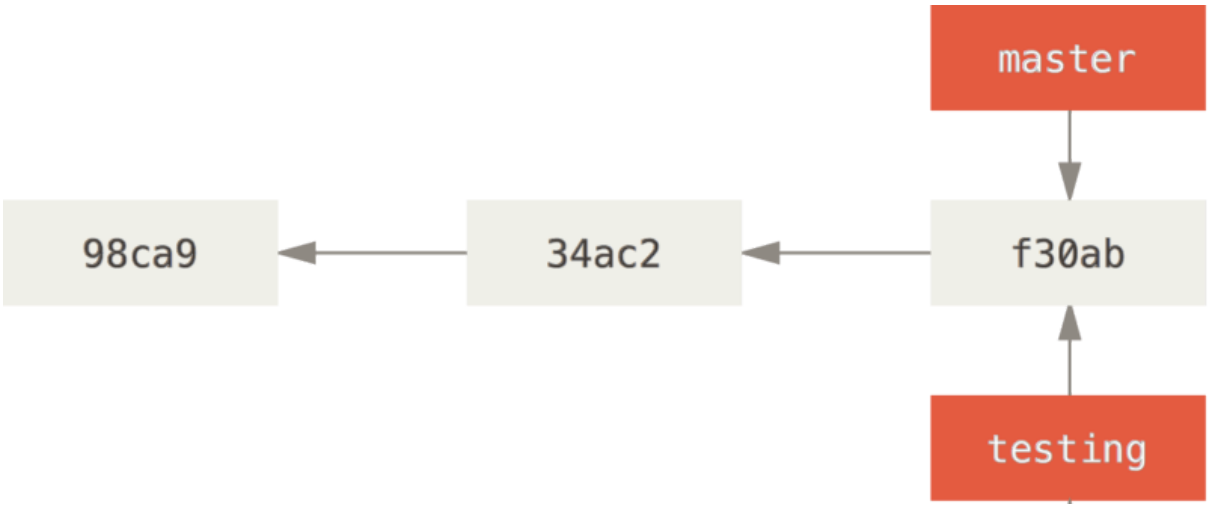


What happens if you create a new branch? Well, doing so creates a new pointer for you to move around.

Let’s say you create a new branch called testing.

 # git branch testing

This creates a new pointer to the same commit you’re currently on.



Two branches pointing into the same series of commits.

How does Git know what branch you're currently on?

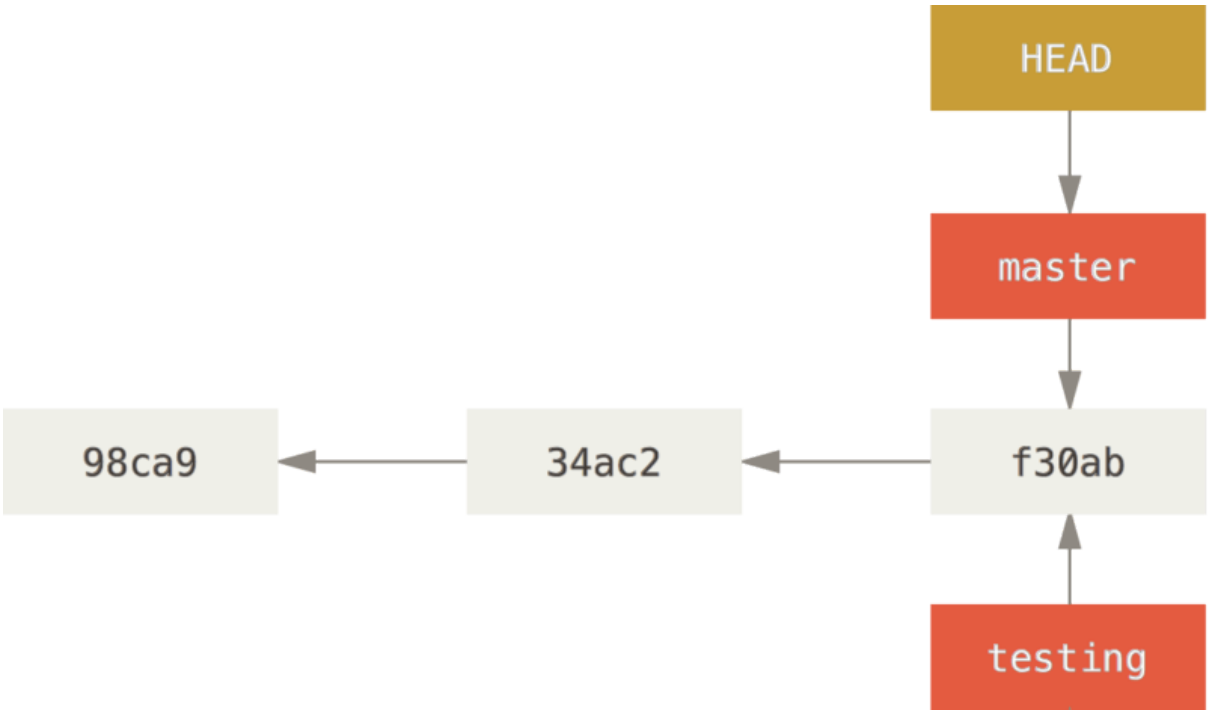
It keeps a **special pointer** called **HEAD**.

HEAD is a pointer to the latest commit id and is **always moving**, not stable.

 # git show HEAD

In Git, this is a pointer to the local branch you're currently on.

In this case, you're still on master. The git branch command only created a new branch — it didn't switch to that branch.



This command shows you **where the branch pointers are pointing:**

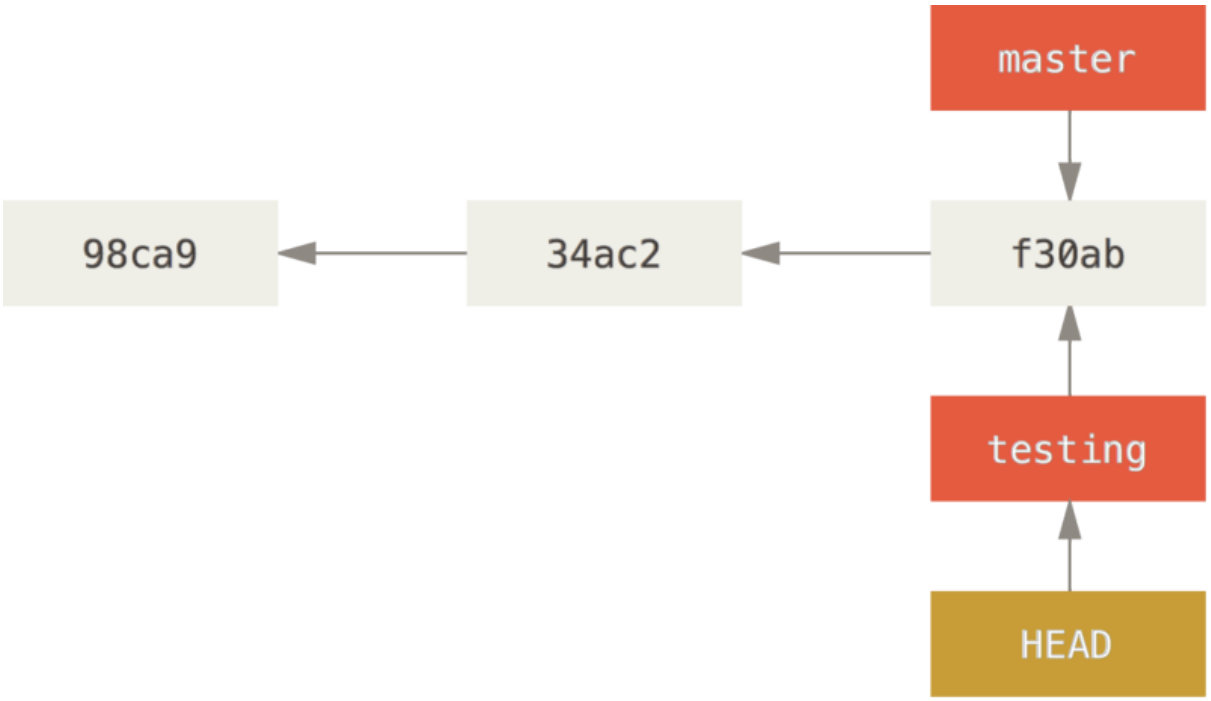
 # git log --oneline --decorate

You can see the "**master**" and "**testing**" branches that are right there next to the f30ab commit.

To **switch to an existing branch**, you run the git checkout command.

git checkout testing

This **moves HEAD** to point to the **testing** branch.

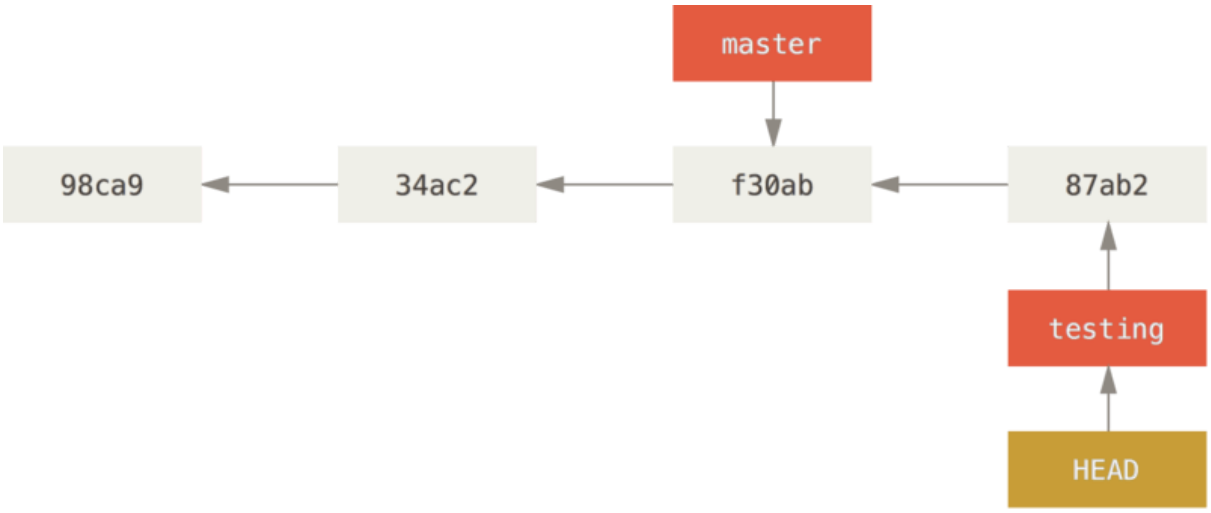


What is the significance of that?

Well, let’s do another commit:

```
# vim test.rb
# git commit -a -m 'made a change'
```

 # git log --oneline --decorate

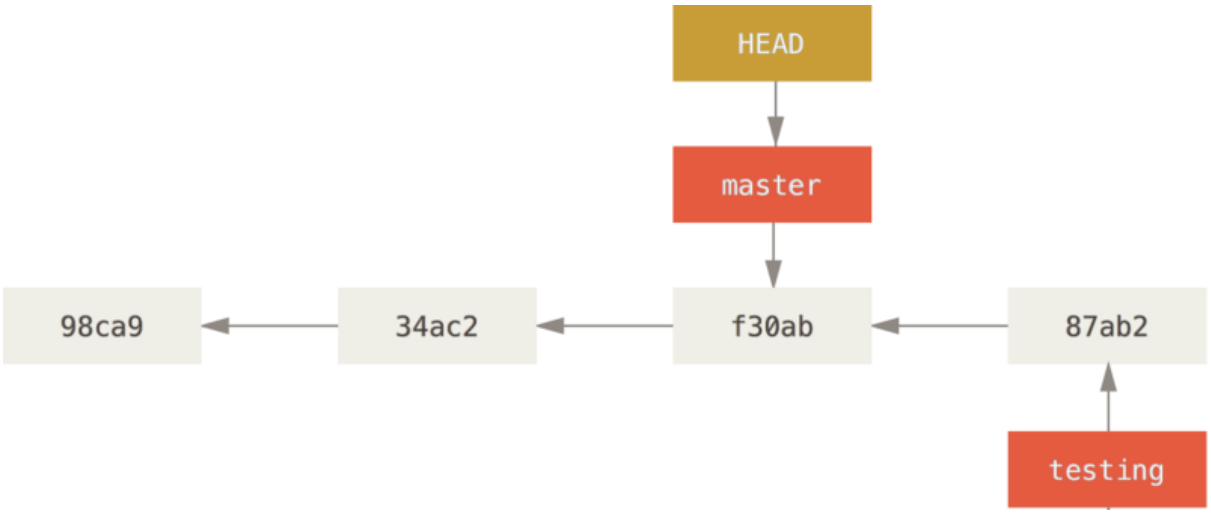


The HEAD branch moves forward when a commit is made.

This is interesting, because now your **testing** branch has **moved forward**, but your master branch still points to the commit you were on when you ran git checkout to switch branches.

Let's switch back to the master branch:

 # git checkout master



HEAD moves when you checkout.

That command(git checkout master) did two things. It **moved** the **HEAD** pointer back to point to the **master branch**, and it **reverted the files** in your working directory **back to the snapshot** that **master points to**.

Let’s make a few changes and commit again:

```
# vim test.rb
# git commit -a -m 'made other changes’
```

Now your project history has diverged.

You created and switched to a branch, did some work on it, and then switched back to your main branch and did other work.

Both of those changes are isolated in separate branches: you can switch back and forth between the branches and merge them together when you’re ready.

And you did all that with simple **branch**, **checkout** and **commit** commands.

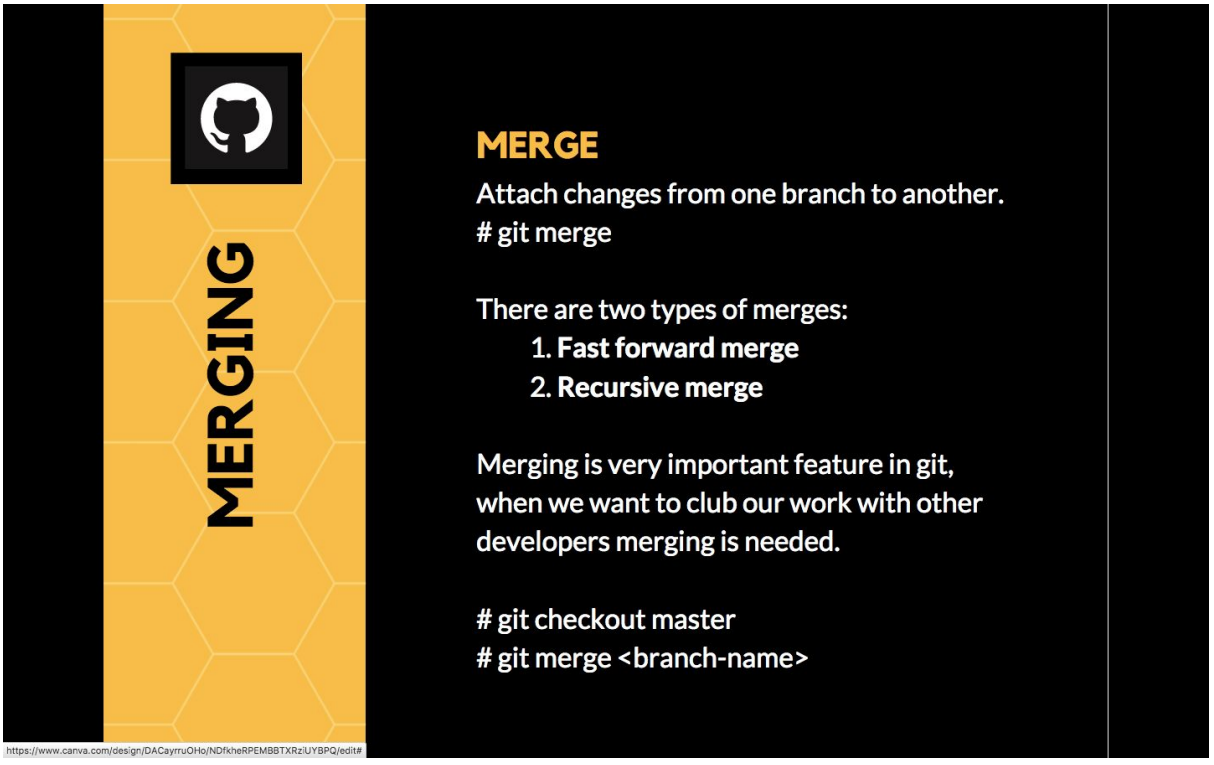
To see all available branches

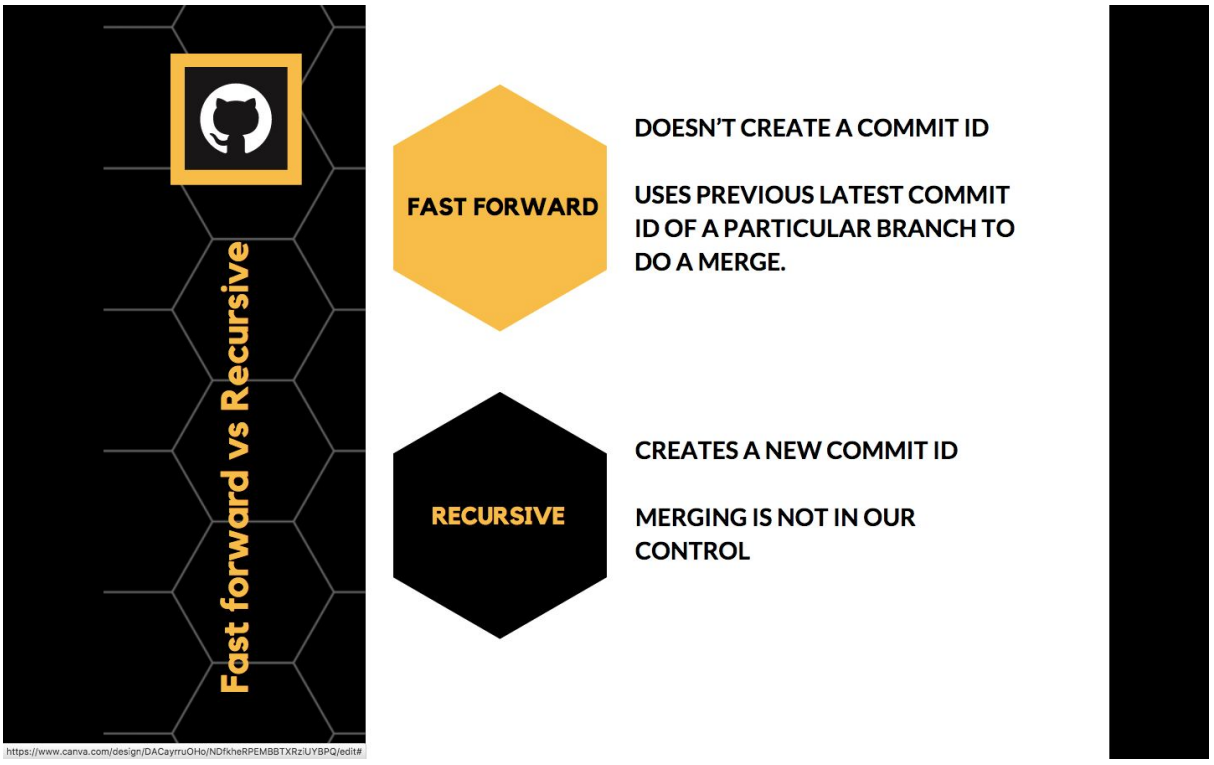
 # git branch -a

 # git reflog {short logs}

Merging

==========





Git Merge Conflict

==========

A merge conflict happens when two branches both modify the same region of a file and are subsequently merged. Git can't know which of the changes to keep, and thus needs human intervention to resolve the conflict.

On **master branch**

vi services.html { Add Two Dummy services like Research & Development }

 # git branch training

 # git branch consulting

git checkout **training**

vi services.html { We provide training }
git add && git commit
git checkout master
git merge training

git checkout **consulting**
vi services.html { We provide Consulting **add in same line**}
git add && git commit
git checkout master
git merge consulting

Automatic merge failed

we do get a merge conflict here, open the **services.html** in **vi** and resolve the conflicts that occurred.

git add .
git commit -m “Conflict resolved”

Git ignore

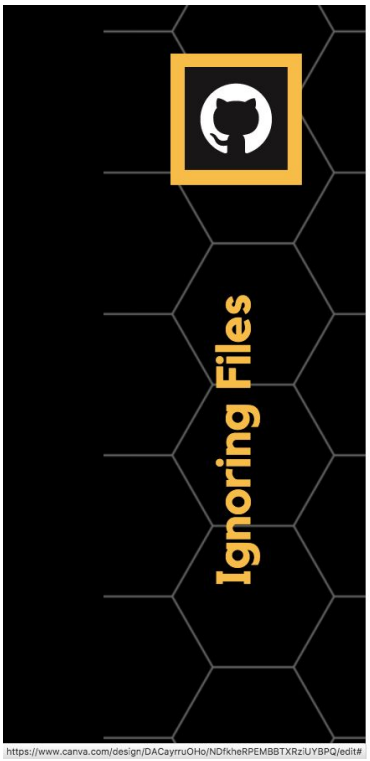
==========

It’s a list of files you want git to ignore in your working directory.

It's usually used to avoid committing transient files from your working directory that aren't useful to other collaborators such as temp files IDE’s create, Compilation files, OS files etc.

A file should be ignored if any of the following is true:

- **The file is not used by your project**
- **The file is not used by anyone else in your team**
- **The file is generated by another process**



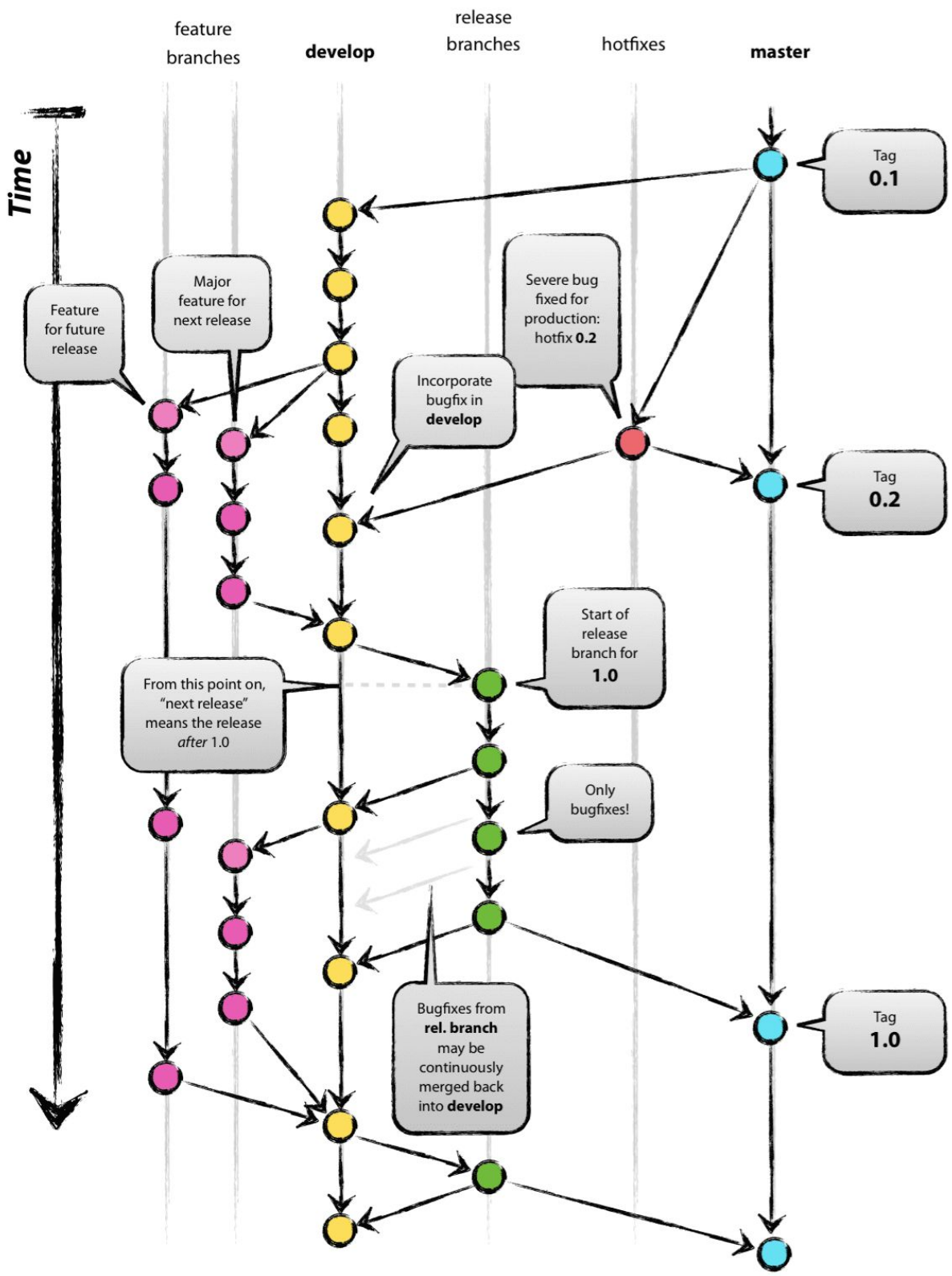
Whenever you want to ignore files we use **.gitignore** file in project directory.

The reason for ignoring files is that you may not want to include some of the files that are not required for project to run.

Example:
db.properties
server.properties

vi .gitignore {add *.properties}

Now all .properties files will not be tracked


feature branches
develop
release branches
hotfixes
master
Time
Tag 0.1
Feature for future release
Major feature for next release
Severe bug fixed for production: hotfix 0.2
Incorporate bugfix in develop
Tag 0.2
Start of release branch for 1.0
From this point on, "next release" means the release after 1.0
Only bugfixes!
Bugfixes from rel. branch may be continuously merged back into develop
Tag 1.0

.gitignore

==========

https://www.gitignore.io/

If you want some files to be ignored by git **create a file .gitignore**

*.bk

*.class

 # Ignore all php files

*.php

 # but not index.php

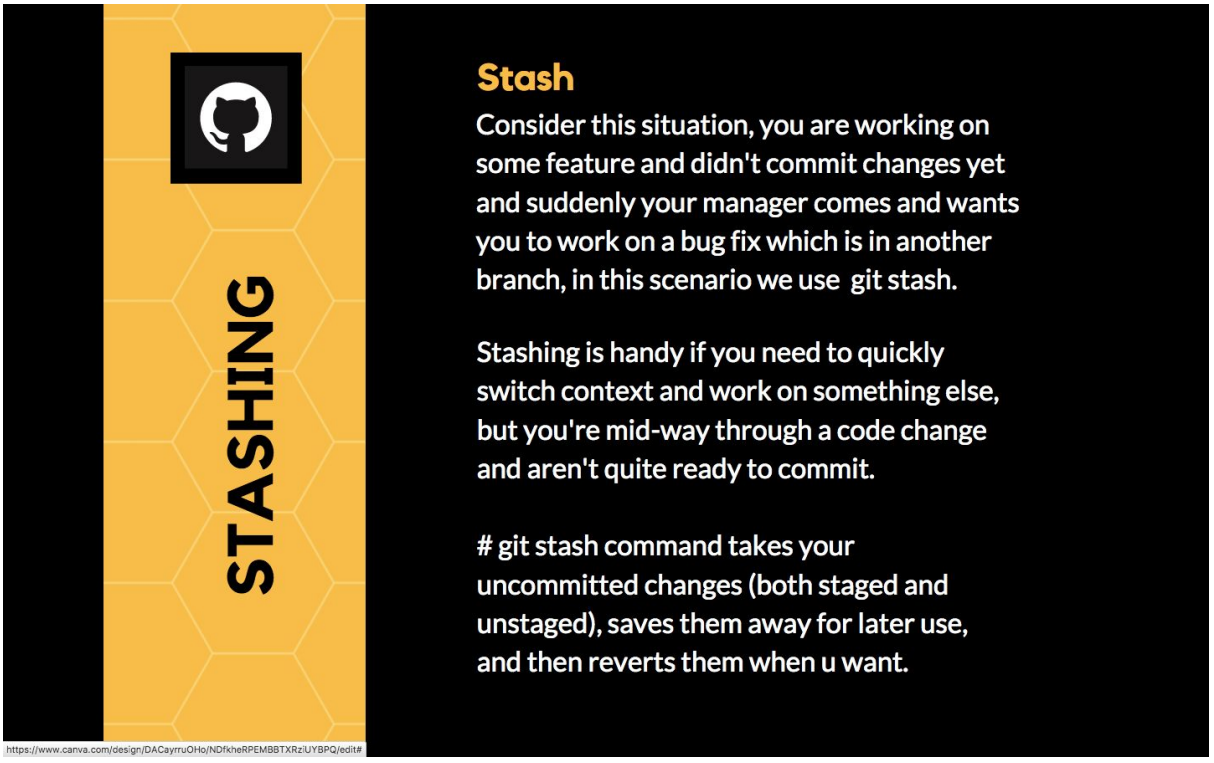
!index.php

 # Ignore all text files that start with aeiou

[aeiou]*.txt

Stashing

==========



 # git stash

 # git stash list

 # git stash apply {apply the top most stashed changes}

 # git stash apply stash@{2} {apply particular stashed changes}

 # git stash show <stash>

 # git stash pop {apply 2nd stash and remove it}

 # git stash pop stash@{2} {pop the stash at 2nd reference}

 # git stash drop stash@{3} {remove the stash}

 # git stash clear {Delete all stash entries}

Tagging

==========

In release management we are working as a team and I'm working on a module and whenever I'm changing some files I'm pushing those files to remote master.

Now I have some 10 files which are perfect working copy, and I don't want this files to be messed up by my other team members, these 10 files they can directly go for release.

But if I keep them in the repository, as my team is working together, there is always a chance that, somebody or other can mess that file, so to avoid these we can do **TAGGING**.

You can tag till a particular commit id, imagine all the files till now are my working copies:

 # git tag 1.0 -m "release 1.0" <commit_id>

 # git show 1.0

 # git push --tags

Goto GitHub and see release click on it, you can download all the files till that commit.

TAGGING helps you in **release management**.

Detached Head

Basically we use checkout for moving from one branch to another branch.

But if i checkout into a commit id, then I go into a state called DETACHED HEAD state.

Say i did **# git checkout <commit-id>**

DETACHED HEAD: is a state where you are not in tree anymore, so we cannot track anymore we are outside the tree. Any change we make in detached head state are not saved.

Now we doesn't have a pointing branch, we are basically in a static commit, if we do
git branch

Reason for checking out into a commit id

To know what happened at that particular commit.

Let's say you have created a file and put some phone no in there, and now person1 changed the file and committed with a message "ph no changed", then again person2 changed the file and committed with same message "ph no changed" later again person3 changed the file and committed with same message "ph no changed".

Now here we can't rely on commit message itself, this is were detached head is needed.

MAVEN

A VCS plays a vital role in any kind of organization, the entire software industry is built around code.

What are we doing with this code ??

- Are we seeing the code when we open the application ?? NO
- Are we seeing the code when we open the app in browser ?? NO

So we are seeing the executable format of the code, that is called build result of the code.

What is build ??

==========

Build is the end result of your source code.

Build tool is nothing but, it takes your source code and converts it into human readable format (executable).

Build

====

The term build may refer to the process by which source code is converted into a stand-alone form that can be run on a computer.

One of the most important steps of a software build is the compilation process, where source code files are converted into executable code.

The process of building software is usually managed by a build tool i.e, maven.

Builds are created when a certain point in development has been reached or the code has been ready for implementation, either for testing or outright release.

Build*: Developers write the code, compile it, compress the code and save it in a compressed folder. This is called Build.*

Maven Objectives

==========

- A comprehensive model for projects which is reusable, maintainable, and easier to comprehend(understand).
- plugins

Convention over configuration

==========

Maven uses *Convention* over *Configuration* which means developers are not required to create build process themselves. Developers do not have to mention each and every configuration detail.

Earlier to maven we had ANT, which was pretty famous before maven.

Disadvantages of ANT

==========

ANT - Ant scripts need to be written for building
[build.xml need to tell src & classes]
ANT - There is no dependency management
ANT - No project structure is defined

Advantages of Maven

==========

No script is required for building [automatically generated - pom.xml]
Dependencies are automatically downloaded
Project structure is generated by maven
Documentation for project can be generated

Maven is called as **project management tool** also, the reason is earlier when we used to create projects and we used to create the directory structure and all by yourself, but now maven will take care of that process.

MAVEN has the ability to create project structure.

Maven can generate documentation for the project.

Whenever i generate a project using maven i will get src and test all by default.

MAVEN FEATURES

==========

Dependency System

==========

Initially in any kind of a build, whenever a dependency is needed, if i'm using ANT i have to download the dependency then keep it in a place where ANT can understood.

If i'm not giving the dependency manually my build will fail due to dependency issues.

Maven handles dependency in a beautiful manner, there is place called **MAVEN CENTRAL**. Maven central is a centralized location where all the dependencies are stored over web/internet.

Plugin Oriented

==========

Maven has so many plugins that i can integrate, i can integrate junit, jmeter, sonarqube, tomcat, cobertura and so many other.

IMPORTANT FILE IN MAVEN

==========

Projects in maven is defined by **POM** (Project Object Model) pom.xml.

Maven lifecycle phases

==========

What is build life cycle?

The sequence of steps which is defined in order to execute the tasks and goals of any maven project is known as build lifecycle in maven.

The following are most common *default* lifecycle phases executed:

- **validate**: validate the project is correct and all necessary information[dependencies] are available and keep it in local repo
- **compile**: compile the source code of the project
- **test**: Execution of unit tests, test the compiled source code using a suitable unit testing framework. These tests should not require the code be packaged or deployed
- **package**: take the compiled code and package it in its distributable format, such as a JAR.
- **verify**: run any checks to verify the package is valid and meets quality criteria, keeps the HelloWorld.jar in .m2 local repo
- **install**: Deploy to local repo [.m2], install the package into the local repository, for use as a dependency in other projects locally
- **deploy**: done in an integration or release environment, copies the final package to the remote repository for sharing with other developers and projects. This will push the libraries from .m2 to remote repo.

There are two other Maven lifecycles of note beyond the *default* list above. They are

- clean: cleans up artifacts created by prior builds
- site: generates site documentation for this project

These lifecycle phases are executed sequentially to complete the default life cycle.

POM { will be in XML format }
====

GAV

Maven uniquely identifies a project using:

- **groupID**: Usually it will be the domain name used in reverse format (going to be given by the project manager).
- **artifactID**: This should be the name of the artifact that is going to be generated
- **Version** : Version of the project, Format {Major}.{Minor}.{Maintenance} and add “-SNAPSHOT” to identify in development

Version can be two things here, a SNAPSHOT and other is RELEASE.

Snapshot - whenever your project is in working condition i mean we are still working on it that would be a snapshot version, you can have multiple snapshots for one single project.

Packaging

=========

Build type is identified by **<packaging>** element, this element will tell how to build the project.

Example packaging types: jar, war etc.

Archetype

=========

Maven archetypes are project templates which can be generated for you by Maven. In other words, when you are starting a new project you can generate a template for that project with Maven.

In Maven a template is called an *archetype*.

Each Maven archetype thus corresponds to a project template that Maven can generate.

Installation

==========

Maven is dependent on java as we are running java applications, so to have maven, we also need to have java in system.

Install java

=========

Java package : java program --- java-1.8.0-openjdk

Java package : java compiler --- java-1.8.0-openjdk-devel

```
# sudo yum -y install java-1.8.0-openjdk
# sudo yum -y install java-1.8.0-openjdk-devel
# java -version        { confirm java version}
```

Install Maven

==========

```
# yum -y install maven
```

Maven repository are of three types

==========

For maven to download the required artifacts of the build and dependencies (jar files) and other plugins which are configured as part of any project, there should be a common place. This common shared area is called as Repository in maven.

Local

=====

The repository which resides in our local machine which are cached from the remote/central repository downloads and ready for the usage.

Remote

=======

This repository as the name suggests resides in the remote server. Remote repository will be used for both downloading and uploading the dependencies and artifacts.

Central

======

This is the repository provided by maven community. This repository contains large set of commonly used/required libraries for any java project. Basically, internet connection is required if developers want to make use of this central repository. But, no configuration is required for accessing this central repository.

How does Maven searches for Dependencies?

==========

Basically, when maven starts executing the build commands, maven starts for searching the dependencies as explained below :

- It scans through the local repositories for all the configured dependencies. If found, then it continues with the further execution. If the configured dependencies are not found in the local repository, then it scans through the central repository.

- If the specified dependencies are found in the central repository, then those dependencies are downloaded to the local repository for the future reference and usage. If not found, then maven starts scanning into the remote repositories.

- If no remote repository has been configured, then maven will throw an exception saying not able to find the dependencies & stops processing. If found, then those dependencies are downloaded to the local repository for the future reference and usage.

Setting up stand alone project

==========

```
# cd ~
# mvn archetype:generate            { generates project structure }
# we get some number like 1085 beside it we have 2xxx, which means maven
 currently supports 2xxx project structures, 1085 is like default project
# press enter
# choose a number :: 6 which means latest, so press enter
# groupId: com.digital.academy    { unique in world, generally domain }
# artifactId: project1                { project name }
# version: press enter                { snapshot - intermediate version }
# package: enter { package name is java package }
# Press: y
```

Now maven has successfully created a project structure for you:

```
# tree -a project1
```

We have pom.xml which contains all the definitions for your project generated, this is the main file of the project.

ls -l ~/.m2/repository

mvn validate { whatever in the pom.xml is correct or not }

Let's make some mistakes and try to fail this phase,

mv pom.xml pom.xml.bk
mvn validate { build failure }
mv pom.xml.bk pom.xml
vi App.java { welcome to Devops }
mvn compile { after changing code we do compilation right }
{ this generates a new structure - # tree -a . with class files}
mvn test { test the application }
mvn package { generates the artifact - jar }

 # java -cp target/xxxx.jar groupid(com.digital.proj1).App

Plugins

=======

We saw maven is only performing phases like validate, compile, test, package, install, deploy but if you remember there is no execution of a jar file,

Can you see any of the phases running jar file, no right ??

Executing jar file is not part not the part of life cycle,
apart from the above phases such as validate, compile, test, package, install, deploy all the other come under <build> under <plugins> </plugins>

These plugins will define, other then regular maven lifecycle phases,

Let's take example, i want to run my jar file,
After </dependencies> in your pom.xml

After </dependencies> in your pom.xml add the following

```
<build>
<plugins>
<plugin>
<groupId>org.codehaus.mojo</groupId>
  <artifactId>exec-maven-plugin</artifactId>
  <version>1.2.1</version>
  <configuration>
    <mainClass>com.digi.App</mainClass>
        <arguments>
            <argument>-jar</argument>
            <argument>target/*.jar</argument>
         </arguments>
</configuration>
</plugin>
</plugins>
</build>
</project>
```

In pom.xml after </dependencies> add <build> <plugins> <plugin>

We need to run

 # mvn exec:java

WEB APP SETUP

Setting up web project

==========

mvn archetype:generate | grep maven-archetype-webapp

type the number you get

tree -a project/

mvn clean package

tree -a project

You can see the **war** generated under target directory

Tomcat Installation [Binaries]

==========

Google tomcat 7 download

goto tomcat downloads page and get the binary tar file for the tomcat 7 by wget

wget <link>

wget http://www-us.apache.org/dist/tomcat/tomcat-7/v7.0.82/bin/apache-tomcat-7.0.82.tar.gz

tar xf apache-tomcat-7.0.82.tar.gz

cd apache-tomcat-7.0.82/bin

./startup.sh

Check for port to be opened in firewall

netstat -ntpl { # sudo yum -y install net-tools }

ps -ef | grep tomcat

Goto http://ip-address/8080 {click cancel and change tomcat-users.xml file}

<role rolename="manager-gui"/>

<user username="tomcat" password="tomcat" roles="manager-gui"/>

<user username="tomcat1" password="tomcat1" roles="manager-script"/>

Change the port number in server.xml

cd apache-tomcat-7.0.81/bin

./shutdown.sh

Copy the generated war file to webapps dir of tomcat

Refresh the tomcat page

Deploy to tomcat maven tomcat plugin

==========

Add Manager-Script Role

==========

Add new role under conf/tomcat-users.xml

vi **conf/tomcat-users.xml**

*<user username="tomcat1" password="tomcat1" roles="**manager-script**"/>*

Add Tomcat 7 Maven Plugin

==========

```
# vi pom.xml
<plugin>
  <groupId>org.apache.tomcat.maven</groupId>
  <artifactId>tomcat7-maven-plugin</artifactId>
  <version>2.2</version>
  <configuration>
        <url>http://localhost:8080/manager/text</url>
        <server>TomcatServer</server>
        <path>/WebApps</path>
  </configuration>
</plugin>
```

Add Maven-Tomcat Authentication

==========

```
# vi ~/.m2/settings.xml
<settings>
  <servers>
        <server>
              <id>TomcatServer</id>
              <username>tomcat1</username>
              <password>tomcat1</password>
        </server>
  </servers>
</settings>
```

 # mvn tomcat7:deploy

 # mvn tomcat7:undeploy

 # mvn tomcat7:redeploy

Profiles

========

So in general, what is the meaning of profile, let's take windows as example for each profile there would be some different settings right.

I mean in same machine we can have different different profiles. Similarly in pom.xml, the project is same, but you can create multiple profiles, for multiple purposes.

So for my project i want to have different different profiles like dev, qa and prod env. Here the requirements are different for each and every env, like in dev env we don't need any of the test to be run.

Let's say there 4 people who have different different req, now i need to create 4 projects instead of 4 projects, within a single project i can have 4 profiles, that's the profile concept.

<profiles> will not be there under <build>, they will be below </dependencies>,
So the pom.xml looks like this with <profiles>

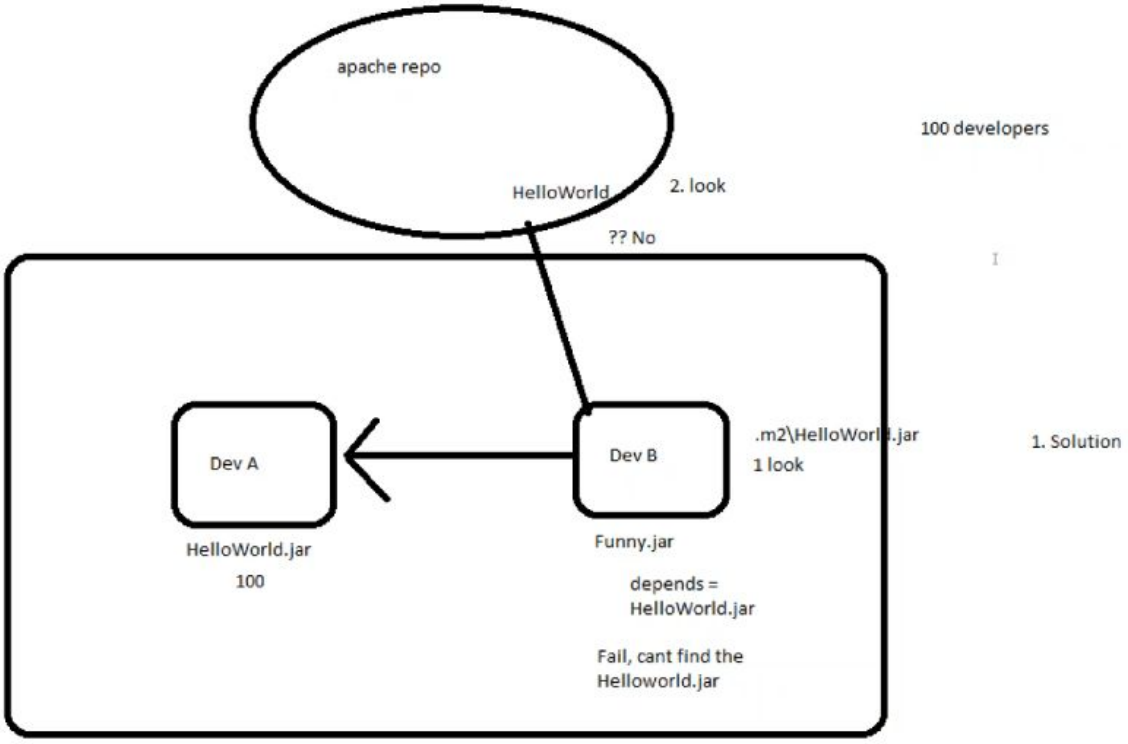
```
</dependencies>
<profiles>
        <profile>
                <id>DEV</id>
                <build>
                    <plugins>
                        <plugin>
```

I have keep the configuration here please go through it

https://github.com/ravi2krishna/Maven-Build-Profiles.git

NEXUS

⇒ Nexus is a Binary Repository Manager



Now the simple solution for this problem is, Dev B should ask Dev A to provide the HelloWorld.jar, so that DevB can keep the Hello.jar in DevB machines .m2 directory,

This works coz the maven will look first in .m2 directory.[Local Repo]
If not then it maven central [Apache Repo]
Now imagine DevA, keeps on changing the Hello.jar code, let's say DevA changed 100 times now DevB should ask DevA 100 times which doesn't make sense.

Tomorrow in your project, you have 100 Developers, now they have to exchange their libraries, now it won't be that easy to exchange the libraries.

It's really cumbersome process, Nobody understands which version is there with whom.

The solution is there, but this is a complex and time taking solution.
This may workout if there are only 2-3 developers, but not more then that.

This is where Binary Repository Concept comes into picture.

Now we will introduce a new server within our organization. This server we call it as Remote Repository.

Just like how apache is maintaining a MavenCentral, similarly we will maintain our own remote repository.

Now DevA instead of sharing his Hello.jar with DevB, DevC etc
He will push it to Our Remote Repository and now DevB, DevC everyone who needs that Hello.jar will pull it from the Remote Repository.

I mean they[DevB, DevC] will add that info in the pom.xml, now pom will take care of downloading it from remote repo.
ow we got totally three kinds of Repo's.

1. Local
2. Public Repo
3. Private Repo

Now even the libraries are secured, coz they are present within your organization.

We got different tools for that Artifactory/Nexus/Archiva.

Sonatype Nexus
==========

Now we are going to set up this within our server.

Install Nexus Server.

Search for Apache Maven Nexus Repository in google,

Snapshot ⇒ Development progress build [Partial Completed Jars]

Release ⇒ Ready to release build [Official proper release]

Installation of Nexus

==========

type download nexus in google
go with version 2.x as 3.x is not yet supported with jenkins
copy the link address for 2.x and do wget
wget http://www.sonatype.org/downloads/nexus-latest-bundle.tar.gz
mkdir -p tools/nexus
tar xvzf nexus.tar.gz -C tools/nexus
Nexus by default starts on the port number 8081 and un & pw is admin & admin123
cd nexus/nexus-2.x/bin
./nexus start { if any problem change ownership to devops to both dirs of nexus}
netstat -ntpl | grep 8081 { be patient it takes some time to start }
http://ip-addr:8081/nexus { allow port 8081 through firewall }
login and give default creds admin & admin123

ERROR: Change the permissions of two nexus directories with logged in username.

http://ip-addr:8081/nexus

Once the nexus is up we will create two repositories, Snapshot & Release.

Search for distribution management tag in google for maven & nexus, and paste it after </dependencies> tag.

If your version contains a string SNAPSHOT, by default it goes to SNAPSHOT repo.
If your version contains only version 1.0, it goes to RELEASE repo.

Now we will, deploy the artifacts to remote repo by deploy phase.

Login to nexus using admin & admin123, now we already we have some default repos, we have something Central, this is Apache Maven Central.

But we will go with our own repo's, using HOSTED repo's.

Add → Hosted Repo → Repo Id: releaseRepo → Repo Name: releaseRepo → Repo Policy: Release → Deployment Policy: Disable Redeploy → Save

Add → Hosted Repo → Repo Id: snapshotRepo → Repo Name: snapshotRepo → Repo Policy: Snapshot → Deployment Policy: Allow Redeploy → Save

Goto the maven project and do # mvn deploy

Search for maven distributionmanagement nexus in google,

Goto pom.xml and update <distributionManagement> below </dependencies>

<distributionManagement>

<repository>
<id>releaseRepo</id>
<name>releaseRepo</name>
<url>http://192.168.56.101:8081/nexus/content/repositories/releaseRepo/</url>
</repository>

<snapshotRepository>
<id>snapshotRepo</id>
<name>snapshotRepo</name>
<url>http://192.168.56.101:8081/nexus/content/repositories/snapshotRepo/</url>
</snapshotRepository>

</distributionManagement>

Goto the maven project and do # mvn deploy again, now we get new error like 401.

Nexus is strictly authenticated, you cannot deploy until and unless you login,

We don't provide usernames and passwords in pom.xml, coz pom.xml files are stored in VCS, which will be shared to other[most] developers as well.

For this maven provides solution in local repo,

```
# cd ~/.m2
# vi settings.xml       {search for maven settings.xml nexus username and pass
}
<settings>
        <servers>
          <server>
            <id>releaseRepo</id>
            <username>admin</username>
            <password>admin123</password>
        </server>
        <servers>
          <server>
            <id>snapshotRepo</id>
            <username>admin</username>
            <password>admin123</password>
        </server>
</settings>
```

Now change the pom.xml <version> to 1.0-SNAPSHOT and redeploy again.

```
# mvn deploy[After one min, again run mvn deploy]
# mvn deploy
```

SONARQUBE

Prerequisites :

1. **Java 1.7 + { recommended 1.8 }**
2. **MySql 5.6+**

Install Java 1.8

==========

 # sudo yum -y install java-1.8.0-openjdk

 # sudo yum -y install java-1.8.0-openjdk-devel

 # java -version ***{ confirm java version}***

Install MySql 5.6+ { for production use }

==========

 # wget http://repo.mysql.com/mysql-community-release-el7-5.noarch.rpm

 # sudo rpm -ivh mysql-community-release-el7-5.noarch.rpm

 # sudo yum -y install mysql-server

 # sudo systemctl start mysqld

 # sudo mysql_secure_installation

 # sudo mysql -u root -p

 # CREATE DATABASE sonar;
 # CREATE USER 'sonar' IDENTIFIED BY 'sonar';
 # GRANT ALL ON sonar.* TO 'sonar'@'localhost' IDENTIFIED BY 'sonar';
 # FLUSH PRIVILEGES;

Sonarqube Installation

==========

Visit https://www.sonarqube.org/downloads/

wget the latest version [LTS]

unzip <file>

cd sonar/conf

vim **sonar.properties**

Changes to make in **sonar.properties**

==========

 # sonar.jdbc.username=sonar

 # sonar.jdbc.password=sonar

 # uncomment sonar.jdbc.url of MySQL 5.6

 # uncomment sonar.web.host=0.0.0.0

 # uncomment sonar.web.port=9000

 # cd sonarqube-6.7/bin

 # cd linux-x86-64

 # ./sonar.sh start

Browse the sonarqube dashboard on
http://ip-addr:9000/

Now browse to one of the maven projects and do mvn sonar:sonar.

cd SampleApp

mvn compile sonar:sonar

git clone https://github.com/wakaleo/game-of-life

cd game-of-life

 # mvn compile sonar:sonar

JENKINS

INTRODUCTION

==========

What is jenkins ??

Jenkins is an application that monitors executions of repeated jobs, such as building a software project.

Now jenkins can do a lot of things in an automated fashion and if a task is repeatable and it can be done in a same way over time, jenkins can do it not only doing it but it can automate the process, means jenkins can notify a team when a build fails, jenkins can do automatic testing(functional & performance) for builds.

Traditionally, development makes software available in a repository, then they give a call to operations/submit a ticket to helpdesk and then operations builds and deploys that software to one or more environments, one this is done, there is usually a QA team which loads and executes performance test on that build and makes it ready for production.

So what jenkins does is a lot of these are repeatable tasks, which can be automated by using the jenkins.

Jenkins has a large number of plugins which helps in this automation process.

Continuous Integration

==========

is a development practise that requires developers to integrate code into a shared repository several times per day (repos in subversion, CVS, mercurial or git). Each check-in is then verified by an automated build, allowing everyone to detect and be notified of problems with the package immediately.

Build Pipeline

==========

is a process by which the software build is broken down in sections:

- **Unit test**
- **Acceptance test**
- **Packaging**
- **Reporting**
- **Deployment**
- **Notification**

The concepts of **Continuous Integration**, **Build Pipeline** and the new "**DevOps**" movement are revolutionizing **how we build, deploy and use software**.

Tools that are effective in automating multiple phases of these processes (like Jenkins) **become more valuable in organizations where resources, time or both are at a premium.**

Installation of Jenkins

==========

- We need **java** to work with jenkins.
    - **# sudo yum -y install java-1.8.0-openjdk**
    - **# sudo yum -y install java-1.8.0-openjdk-devel**
    - **# java -version** ***{ confirm java version}***
- Generally jenkins runs on port 8080, so we need to make sure that there is no other service that is running and listening on port 8080

- Now we need to add jenkins repo, to our repository list, so that we can pull down and install jenkins package.
    - # sudo wget -O /etc/yum.repos.d/jenkins.repo https://pkg.jenkins.io/redhat-stable/jenkins.repo
    - We will run a key so that we can trust this repo and pull down jenkins package
    - # sudo rpm --import https://pkg.jenkins.io/redhat-stable/jenkins.io.key
- Now our key has been imported we can do

    - # sudo yum -y install jenkins
- Now let's enable the service
    - # sudo systemctl enable jenkins
- Now let's enable the service
    - # sudo systemctl start jenkins
- Now if you do **# ip-address:8080** you can see jenkins home

Now if you do # cat /etc/passwd | grep jenkins

jenkins:x:996:994:Jenkins Automation Server:/var/lib/jenkins:/bin/false

Creates a user called jenkins, and keeps this user away from login, this is the default behaviour of jenkins coz we are right now on master and building everything here.

But in real time we have two things here **master** and **build slaves**
Master: where jenkins is installed and administration console is
Build slaves: these are servers that configured to off load jobs so that master is free

Creating Users
==========
Manage Jenkins → Manage Users → Create User Left Side → Fill details
Create Users : tester 1, tester 2, developer 1 and developer 2

Create new jobs
==========
testingJob ⇒ Execute Shell ⇒ echo "Testing Team Jobs Info"

developmentJob ⇒ Execute Shell ⇒ echo "Development Team Jobs Info"

Now login with the tester1 and see the list of jobs.
Now login with the developer1 and see the list of jobs.

As you can see, both the testing and development team jobs are visible across different teams, which would be a security concern, but this is the default behavior of jenkins.

Manage Jenkins → Configure Global Security → Authorization → Logged in users

Now let's see how we can secure the jenkins to make jobs only visible to testing and development teams.

For this we need to install new plugin called **Role-based Authorization Strategy.**

Let's see what are **PLUGINS** first

Plugins: plugins enhances jenkins power and usability.

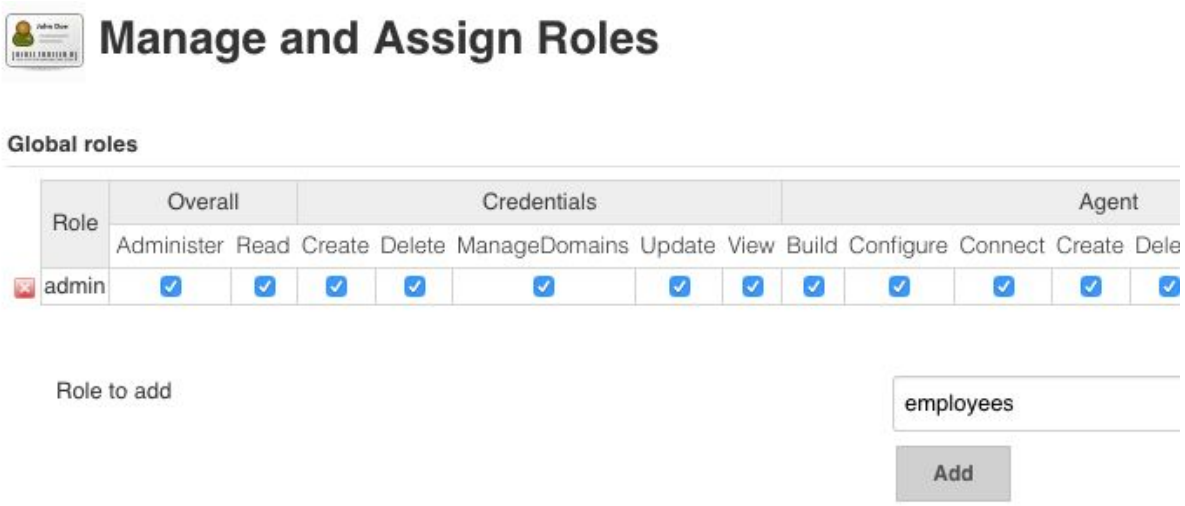
Installing Plugin
==========
Manage Jenkins → Manage Plugins → Available → Search Role-based → Install without restart.

Manage Jenkins → Configure Global Security → Authorization, now we can see the new option **Role-Based Strategy.**

Select the **Role-Based Strategy → Apply → Save**
Now if i login with the **tester** or **developer** user i won't have access, i can only access with the admin user.

Now let's see how we can go and **create some roles** and based on roles we should grant access to users:
Manage Jenkins → Manage and Assign Roles → Manage roles →

Global Roles: check **Role to add,** give something like **employee** and click add



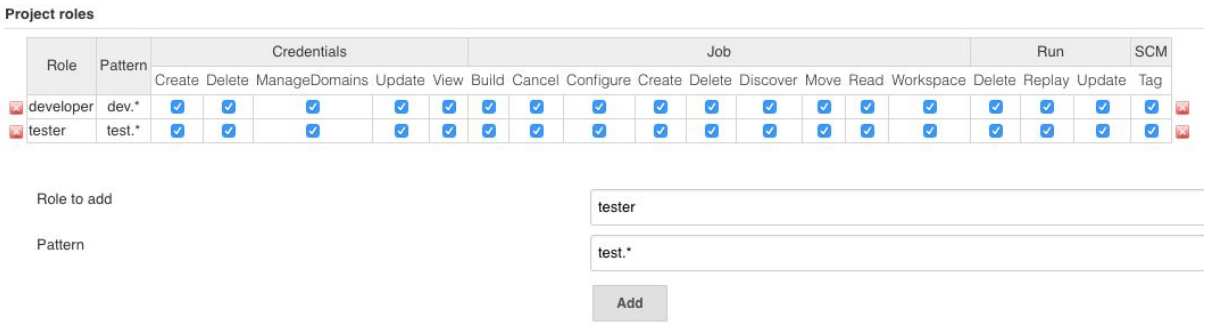
and give overall **read access** and over all **view access**

Project Roles: here we can **create roles specific to a project**,

Role to add: developer && Pattern: dev.* and check everything, now developers will only have access to projects that start with **dev** but nothing else. Click **Apply** and **Save**

Role to add: tester && Pattern: test.* and check everything, now testers will only have access to projects that start with **test** but nothing else. Click **Apply and save**

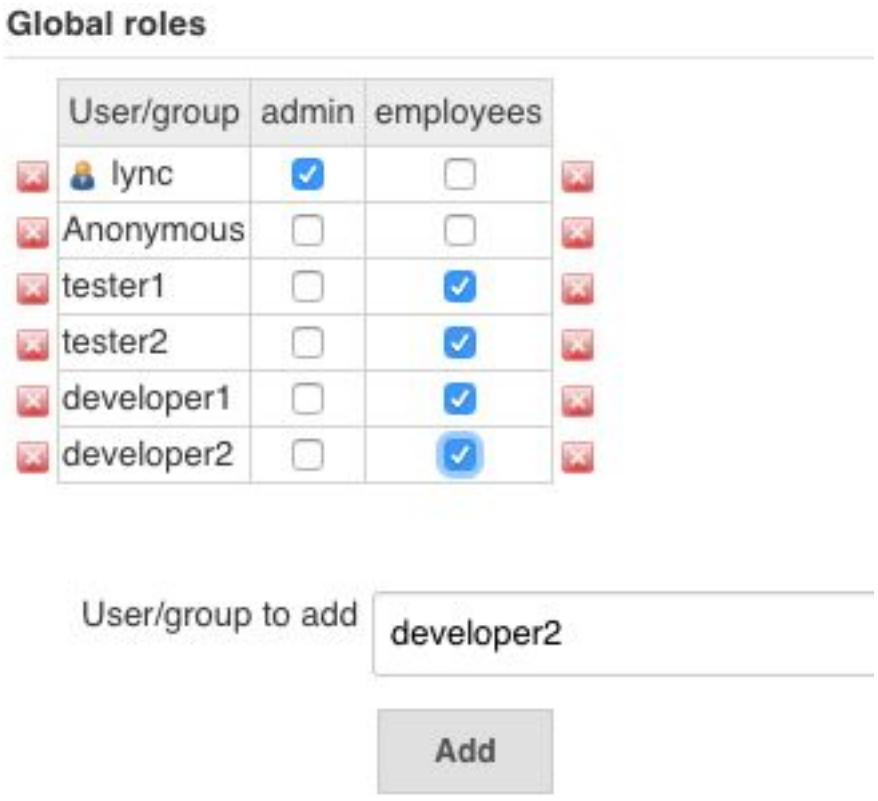
- So we have created an **employee role at global level** and we created two roles **developer and tester at project level**.



Assigning Roles To Users

==========

- **Manage Jenkins → Manage and assign roles → Assign roles → Global Roles → User/group** to add → Add Users tester1, tester2, develoepr1 and developer2 to **Global roles** as **employee → Apply**



Under **Item Roles** **User/group** to add → Add Users tester1, tester2, developer1 and developer2 to **Item roles** → Now add Users tester1 & tester2

to **Tester Roles** and Users developer1 & developer2 to **Developer Roles** → **Apply**

- So we created users, we created roles and we assigned roles

Now if you login with tester users you can only see jobs related to testing, and similarly if you login with developer you can see only development related jobs.

Jenkins [Master - Slave Config]
==========

We know we have user jenkins, who has no shell and this user is the owner of jenkins application but beyond that it's a normal user. So what we are going to do is manage the global credentials.

Now we should make a decision that when we run jobs either locally or remotely we are going to run them with the jenkins user using ssh so that we can control slaves.

So on our system(master) we are going to change jenkins user, so that we can login with jenkins user, let's go and do it

vi /etc/passwd {jenkins change **/bin/false** to **/bin/bash**}
sudo su jenkins

Make sure our jenkins user is a sudoer
usermod -aG google-sudoers jenkins
vi /etc/sudoers or # sudo visudo
Below root add the following

```
# root  ALL=(ALL)     ALL
# jenkins  ALL=(ALL)         NOPASSWD:   ALL
```

Switch to jenkins user

```
# sudo su jenkins
# cd    { make sure you are in jenkins home /var/lib/jenkins}
# ssh-keygen
```

Click on Credentials on homepage of jenkins → Click on Global credentials → Adding some credentials → Kind (select SSH username with private key) → Scope (global) →
Username (jenkins) → Private Key (From jenkins master ~/.ssh) → Ok

Now this sets jenkins user account to be available for SSH key exchange with other servers. This is imp coz we want the ability so that single jenkins user can be in control to run our jobs remotely so that master can off load its jobs to slaves.

Slave
=====
Now create a new centos server(machine) where you will be getting new ip {1.2.3.4}
Create a user jenkins **# useradd jenkins**
usermod -aG google-sudoers jenkins
sudo su jenkins -
cd { make sure you are in jenkins home /home/jenkins}

Now do password less authentication steps on both machines :

vi authorized_keys (add public key of other machine)
chmod 700 ~/.ssh
chmod 600 ~/.ssh/authorized_keys
Now if everything is good then you should be able to login into the slave machine without password.

Doing builds on slaves
==========
So we talked about this a lot, that we don't want to do builds much on master.
We are going to create dumb slave nodes, slaves doesn't need to know anything about implementation they should be able to just run jobs and be controlled by master.

We are going to use this slave nodes in order to off-load the build processing to other machines so that the master server doesn't get CPU, IO or N/W load etc in managing large number of jobs across multiple servers multiple times a day.

So we are going to **create a slave node**:

Now make sure there is key exchange b/w both the machines.

Manage Jenkins (scroll down) → **Manage Nodes** → Master (Now master is always going to be included by default here if we click on master) → Configure → Usage : Only build jobs with label matching nodes(for the most part we want master only to use for controlling jobs we have setup) → Save

Manage Jenkins (scroll down) → Manage Nodes → New Node → Node name : Remote slave 1 → # of executors : 3 (up to 3 concurrent jobs) → Remote root Directory: (/home/jenkins) Labels: remote1 → Usage: (as much as possible) → Launch method: Launch slave via SSH → Host: ip of slave → Credentials: Service acc(Give settings of **Kind**: SSH username with private key **&&** Private key: From jenkins master ~/.ssh) → Availability: Keep online as much as possible → Save

Install java and javac on all slaves.

Then click on node and see the **log**.

Now if i goto jenkins home i can see both master and slave and their executors

Building a job on slave

==========

New job → Under General (Restrict where this project can run)(Label expression: remote1 or expression we gave while creating slave) → Build → Execute Shell → in command give # pwd # uname -a # hostname → Save → Run build.

VAGRANT

Vagrant

=======

Vagrant is a tool for building and managing virtual machine environments.
With an easy-to-use workflow and focus on automation, Vagrant lowers development environment setup time, increases production parity, and makes the "works on my machine" excuse a relic of the past.
Vagrant is suitable for development environment.

Why Vagrant ??

==========

Vagrant provides easy to configure, reproducible, and portable work environments built on top of industry-standard technology and controlled by a single consistent workflow to help maximize the productivity and flexibility of you and your team.

To achieve its magic, Vagrant stands on the shoulders of giants. Machines are provisioned on top of VirtualBox, VMware, AWS, or any other provider. Then, industry-standard provisioning tools such as shell scripts, Chef, or Puppet, can automatically install and configure software on the virtual machine.

For Developers

If you are a **developer**, Vagrant will isolate dependencies and their configuration within a single disposable, consistent environment, without sacrificing any of the tools you are used to working with (editors, browsers, debuggers, etc.). Once you or someone else creates a single Vagrantfile, you just need to vagrant up and everything is installed and configured for you to work. Other members of your team create their development environments from the same configuration, so whether you are working on Linux, Mac OS X, or Windows, all your team members are running code in the same environment, against the same dependencies, all configured the same way. Say goodbye to "works on my machine" bugs.

For Operators

If you are an **operations engineer** or **DevOps engineer**, Vagrant gives you a disposable environment and consistent workflow for developing and testing infrastructure management scripts. You can quickly test things like shell scripts, Chef cookbooks, Puppet modules, and more using local virtualization such as VirtualBox or VMware. Then, with the *same configuration*, you can test these scripts on remote clouds such as AWS or RackSpace with the *same workflow*. Ditch your custom scripts to recycle EC2 instances, stop juggling SSH prompts to various machines, and start using Vagrant to bring sanity to your life.

For Everyone

Vagrant is designed for everyone as the easiest and fastest way to create a virtualized environment!

What is the meaning of setting up environment ??

==========

Let’s say you need web server
Let’s say you need db server
Let’s say you need app server

Let's say you need some machine for R & D purpose quickly, configure that machine quickly and able to use that machine very quickly, now using vagrant we can reduce installation time.

Typically when we setup OS, you may take around 30-45 min to go along with that. But with vagrant you can spin up the development environment very very quickly.

Now how to spin up the environment ??

==========

lets see how it can be done

Goto the official website of vagrant vagrantup.com

We see something like find boxes right, let’s understand some terminology:
So when we are going with Base Installation of Linux, we need couple of things

- You need a CD or ISO image
- You need a physical machine
- You define some CPU & RAM
- Storage
- Network

So whenever we are going with installation we need to go with all these steps always.

So if we are working with cloud we have some images like CentOS, ubuntu etc

- AMI (Amazon Machine Images)
- Virtualization Layer
- CPU {how much CPU i need}
- RAM {how much RAM i need}
- Storage {how much storage i need}
- Network

I need to configure all the above
So in vagrant, we call these pre-installed Images/OS as **BOXES**
Now you can easily download these boxes, and can spin up the virtual machines easily.

As i told to setup the OS it takes around 30-45 min, but using the vagrant it hardly takes 5-10 min depending on internet speed.

Download and Install vagrant
Vagrant supports on following platforms: WIN/MAC/LINUX

But there is another dependency to vagrant, which is the **virtualization layer**.
So in our laptop/desktop, what is the virtualization layer we use:
Oracle Virtual Box

1. Download and install {win - CMD}
 # vagrant --version

Now i don't have any box right, let's download the box:

 # goto vagrantup.com

 # find BOXES {gives list of all boxes}

So once you go to find boxes, it shows what's the virtualization layer is and the diff boxes available.

There is provider, lets understand the terminology

Here **Virtualbox → Provider**

Provider: tool which is giving you the virtual layer

But we have diff providers available which provides virtualization layer.

Getting Started

==========

We will use Vagrant with VirtualBox, since it is free, available on every major platform, and built-in to Vagrant.

But do not forget that Vagrant can work with many other providers.

Providers

========

While Vagrant ships out of the box with support for VirtualBox, Hyper-V, and Docker, Vagrant has the ability to manage other types of machines as well. This is done by using other *providers* with Vagrant.

Before you can use another provider, you must install it. Installation of other providers is done via the Vagrant plugin system.

Once the provider is installed, usage is straightforward and simple, as you would expect with Vagrant.

Your project was always backed with VirtualBox. But Vagrant can work with a wide variety of backend providers, such as VMware, AWS, and more.

Once you have a provider installed, you do not need to make any modifications to your Vagrantfile, just vagrant up with the proper provider and Vagrant will do the rest:

 # vagrant up --provider=vmware_fusion

 # vagrant up --provider=aws

Up and Running

==========

 # vagrant init centos/7

 # vagrant up

After running the above two commands, you will have a fully running virtual machine in VirtualBox running.

You can SSH into this machine with **# vagrant ssh**, and when you are done playing around, you can terminate the virtual machine with **# vagrant destroy**.

Project Setup

==========

The first step in configuring any Vagrant project is to create a **Vagrantfile**. The purpose of the Vagrantfile is:

1. Mark the root directory of your project. Many of the configuration options in Vagrant are relative to this root directory.
2. Describe the kind of machine and resources you need to run your project, as well as what software to install and how you want to access it.

Vagrant has a built-in command for initializing a directory for usage with Vagrant:

 # vagrant init

This will place a Vagrantfile in your current directory. You can take a look at the

Vagrantfile if you want, it is filled with comments and examples. Do not be afraid if it looks intimidating, we will modify it soon enough.

You can also run vagrant init in a pre-existing directory to setup Vagrant for an existing project.

 # vagrant init centos/7

Vagrantfile

=========

The primary function of the **Vagrantfile** is to describe the type of machine required for a project, and how to configure and provision these machines.
Vagrant is meant to run with one Vagrantfile per project, and the Vagrantfile is supposed to be committed to version control.
The syntax of Vagrantfiles is Ruby, but knowledge of the Ruby programming language is not necessary to make modifications to the Vagrantfile, since it is mostly simple variable assignment.

 # vagrant up

==========

In less than a minute, this command will finish and you will have a virtual machine running centos 7. You will not actually *see* anything though, since Vagrant runs the virtual machine without a UI.

 # vagrant ssh

==========

This command will drop you into a full-fledged SSH session. Go ahead and interact with the machine and do whatever you want.

How to work with vagrant

1. **create a project directory**
2. **create Vagrantfile (configuration file - # vagrant init)**
3. **define the image name you want to use**

Let’s go and do these steps

 # mkdir project1

 # cd project1

It's similar to git, whenever you start with git we say # git init

similarly for vagrant we use

 # vagrant init

Open Vagrantfile

→ It's a ruby configuration file

→ I'll remove unwanted things like commented section, just keep

```
Vagrant.configure("2") do |config|
    config.vm.box = "base"
end
```

I want centos-7 box so let's search for it in vagrantup.com

Change the vagrantfile **# vi Vagrantfile**

```
Vagrant.configure("2") do |config|
    config.vm.box = "centos/7"
end
```

 # vagrant up

 # vagrant up {this command does the following}

1. It will create a VM with name "default"
2. It downloads the centos image to project dir
3. Start the VM (by defualt it will be 1 CPU, 512MB RAM)
4. Creates NAT n/w
5. Setup SSH port forwarding
6. Installs SSH keys
7. Maps the storage
8. Execute provision script

 # vagrant up {execute the command, this brings up the machine}

 # vagrant ssh {logs you into the machine}

 # vagrant halt {brings down the machine}

 # vagrant status {status}

 # vagrant port {port forwarding}

Network
=======

In order to access the Vagrant environment created, Vagrant exposes some high-level networking options for things such as forwarded ports, connecting to a public network, or creating a private network.

Port Forwarding
==========
Port forwarding allows you to specify ports on the guest machine to share via a port on the host machine. This allows you to access a port on your own machine, but actually have all the network traffic forwarded to a specific port on the guest machine.

Let us setup a forwarded port so we can access Apache in our guest. Doing so is a simple edit to the Vagrantfile, which now looks like this:

```
Vagrant.configure("2") do |config|
  config.vm.box = "hashicorp/precise64"
  config.vm.provision :shell, path: "bootstrap.sh"
  config.vm.network :forwarded_port, guest: 80, host: 4567
end
```

Run a # vagrant reload or # vagrant up. Once the machine is running again, load http://127.0.0.1:4567 in your browser. You should see a web page that is being

served from the virtual machine that was automatically setup by Vagrant.

Vagrant also has other forms of networking, allowing you to assign a static IP address to the guest machine, or to bridge the guest machine onto an existing network.

By default vagrant uses **NAT** network provided by your provider aka Virtual Box. But let say you want to use bridge network to get connected to router, just uncomment the “public_network”, and when you do **# vagrant up**, it asks for the network to connect.

 # config.vm.network "public_network"

 # config.vm.network :public_network, bridge: "en0: Wi-Fi (AirPort)"

Configure Hostname

==========

 # config.vm.hostname = “centos”

Changing RAM

==========

```
# Customize the amount of memory on the VM:
config.vm.provider "virtualbox" do |vb|
        vb.memory = "1024"
end
```

Changing CPU

==========

```
# Customize the number of CPU’s on the VM:
config.vm.provider "virtualbox" do |vb|
        vb.cpus = "2"
end
```

Provisioning

==========

Provisioners in Vagrant allow you to automatically install software, alter configurations, and more on the machine as part of the vagrant up process.

Of course, if you want to just use vagrant ssh and install the software by hand, that works. But by using the provisioning systems built-in to Vagrant, it automates the process so that it is repeatable. Most importantly, it requires no human interaction.

Vagrant gives you multiple options for provisioning the machine, from simple shell scripts to more complex, industry-standard configuration management systems.

Provisioning happens at certain points during the lifetime of your Vagrant environment:

- On the first vagrant up that creates the environment, provisioning is run. If the environment was already created and the up is just resuming a machine or booting it up, they will not run unless the --provision flag is explicitly provided.
- When vagrant provision is used on a running environment.
- When vagrant reload --provision is called. The --provision flag must be present to force provisioning.

You can also bring up your environment and explicitly *not* run provisioners by specifying --no-provision.

If we want to install web server in our server**.** We could just SSH in and install a webserver and be on our way, but then every person who used Vagrant would have to do the same thing. Instead, Vagrant has built-in support for *automated provisioning*. Using this feature, Vagrant will automatically install software when you vagrant up so that the guest machine can be repeatedly created and ready-to-use.

We will just setup Apache for our basic project, and we will do so using a shell script. Create the following shell script and save it as bootstrap.sh in the same directory as your Vagrantfile.

bootstrap.sh

==========

```
yum -y install git
yum -y install httpd
systemctl start httpd
systemctl enable httpd
git clone https://github.com/devopsguy9/food.git /var/www/html/
systemctl restart httpd
```

Next, we configure Vagrant to run this shell script when setting up our machine. We do this by editing the Vagrantfile, which should now look like this:

```
Vagrant.configure("2") do |config|
        config.vm.box = "centos/7"
        config.vm.provision :shell, path: "bootstrap.sh"
    end
```

The "provision" line is new, and tells Vagrant to use the shell provisioner to setup the machine, with the bootstrap.sh file. The file path is relative to the location of the project root (where the Vagrantfile is).

After everything is configured, just run vagrant up to create your machine and Vagrant will automatically provision it. You should see the output from the shell script appear in your terminal. If the guest machine is already running from a previous step, run vagrant reload --provision, which will quickly restart your virtual machine.

After Vagrant completes running, the web server will be up and running. You can see the website from your own browser.
This works because in shell script above we installed Apache and setup the website.

Load Balancing

==========

In this project we are going to work with three different machines,

In our previous Load balance example we took one nginx and two apache servers.

But in this example we will be using three nginx servers:

Vagrantfile

==========

```
Vagrant.configure("2") do |config|

config.vm.define "lb1" do |lb1|
        lb1.vm.box = "ubuntu/trusty32"
        lb1.vm.network "private_network", ip: "192.168.45.10"
  lb1.vm.network :forwarded_port, host: 7777, guest: 80
        lb1.vm.provision "shell", path: "provision-nginx.sh"
        end

config.vm.define "web1" do |web1|
        web1.vm.box = "ubuntu/trusty32"
        web1.vm.network "private_network", ip: "192.168.45.11"
  web1.vm.network :forwarded_port, host: 8888, guest: 80
        web1.vm.provision "shell", path: "provision-web1.sh"
end

config.vm.define "web2" do |web2|
        web2.vm.box = "ubuntu/trusty32"
        web2.vm.network "private_network", ip: "192.168.45.12"
  web2.vm.network :forwarded_port, host: 9999, guest: 80
        web2.vm.provision "shell", path: "provision-web2.sh"
end

end
```

provision-nginx.sh

==========

```
#!/bin/bash

echo "Starting Provision: Load balancer"
sudo apt-get -y install nginx
sudo service nginx stop
sudo rm -rf /etc/nginx/sites-enabled/default
sudo touch /etc/nginx/sites-enabled/default

echo "upstream testapp {
        server 192.168.45.11;
        server 192.168.45.12;
}

server {
        listen 80 default_server;
        listen [::]:80 default_server ipv6only=on;
        server_name  localhost;
        root            /usr/share/nginx/html;
        #index index.html index.htm;

        location / {
        proxy_pass http://testapp;
}
        }" >> /etc/nginx/sites-enabled/default
sudo service nginx start
echo "MACHINE: LOAD BALANCER" >> /usr/share/nginx/html/index.html
echo "Provision LB1 complete"
```

provision-web1.sh

==========

```
echo "Starting Provision on A"
sudo apt-get install -y nginx
echo "<h1>MACHINE: A</h1>" >> /usr/share/nginx/html/index.html
echo "Provision A complete"
```

provision-web2.sh

==========

```
echo "Starting Provision on B"
sudo apt-get install -y nginx
echo "<h1>MACHINE: B</h1>" >> /usr/share/nginx/html/index.html
echo "Provision B complete"
```

```
# vagrant status
# vagrant global-status
# vagrant port lb1
# vagrant port web1
```

Running Provisioners

==========

Provisioners are run in three cases:

```
# vagrant up
# vagrant provision
# vagrant reload --provision
```

DOCKER

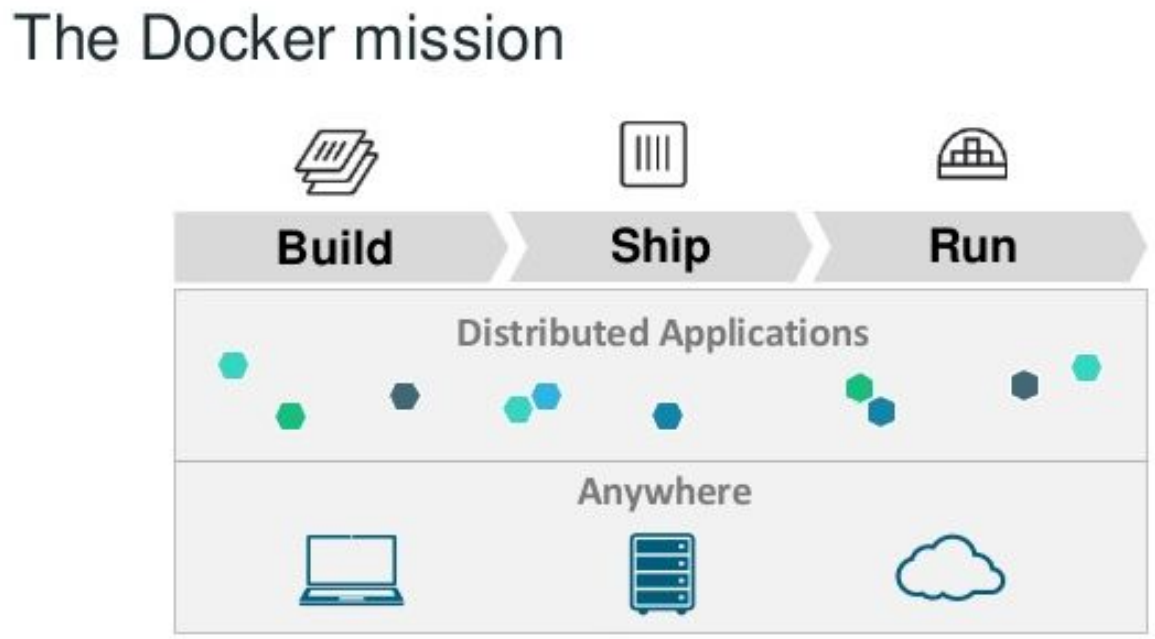
Docker

=======

Docker is a container management service.

The keywords of Docker are **develop, ship** and **run** anywhere.

The whole idea of Docker is for developers to **easily develop applications**, **ship them into containers** which can then be **deployed anywhere**.



Release of Docker was in March 2013 and since then, it has become the buzzword for modern world development.

Features of Docker

==========

- Docker has the ability to **reduce the size of development** by providing a **smaller footprint of the operating system via containers**.
- With containers, it becomes easier for teams across different units, such as development, QA and Operations to work seamlessly across applications.
- You can **deploy Docker containers anywhere**, on any physical and virtual machines and even on the cloud.
- Since Docker containers are pretty lightweight, they are easily scalable.

Why Virtualization ??

==========

- Hardware Utilization
- To reduce no of physical servers
- Reduce Cost
- More different OS

Your whole design of virtualization, is to target the Applications.

We are focusing more on Hardware, Virtualization and OS.
But no one is focusing on Application side,

On Application side we need two fundamental characteristics:

Data Isolation & Data Protection

Let say we are having 3 VM's and minimum requirements for this system are:

1 CPU & 1 GB RAM now like this i need 3 CPU and 3 GB RAM

If the same things is needed for like 1000+ machines, it becomes more cumbersome.

So **Docker** took advantage of this, by using **CONTAINERIZATION**.

Using Docker we can build up entire application with OS.

Sometimes it happens like this application works fine on Linux OS, but doesn't work on Unix and Win, these kind of problems can be avoided by using Docker

Using docker we can create entire application with OS itslef(OS dependent files).

Docker uses special file system,
Layered File system[COW - Copy On Write]

Box → VM's
AMI → Instances
Images → Containers

Three important things to check in docker:
Docker Container, Docker Images & Docker Registry.

Docker Container: is a running instance of an OS image.
Run time object

Installation
==========

 # get.docker.com
sudo usermod -aG docker <user-name>
Docker comes in two components SERVER & CLIENT
systemctl start docker

sudo docker <options>

First thing we need to have is images, from those images will create containers.

rpm -qa | grep docker
sudo systemctl status docker

Let’s see do we have any images

sudo docker images

So where do i get images from ??

==========

hub.docker.com {search for nexus, jenkins, tomcat and centos}

docker pull centos {this is how we get image from internet}

sudo docker images

{ unique image id and size is also very less coz its limited OS }

 # docker info

docker images

I want to see how many containers are running ??

sudo docker ps

open two sessions of the same instance {we can do things simultaneously}

s1 - sudo docker ps {shows running containers}

s2 - sudo docker run -it centos /bin/bash

{now we are inside container}

it - terminal interactive

s1 - sudo docker ps {shows running containers}

This container is created from centos image

If we use container it will take at least 2 min but. here its 3 seconds

s1 - top {so many tasks}

s2 - top {literally 2 process}

s2 - ps -ef {same 2 process}
s2 - cat /etc/hosts {my hostname is container_id}
s1 - sudo docker ps
s1 - sudo docker ps
s2 - exit {bash is finished - container is gone}
s1 - sudo docker ps

docker images

docker run -it { attached mode runs in **foreground** }

docker run -dt { detached mode runs in **background** }
docker exec -it <container-id> bash

naming container
==========
s2 - # sudo docker run --rm -ti --name "web-server01" docker.io/centos /bin/bash
s1 - sudo docker ps
s2 - exit

setting hostname
==========
s2 - sudo docker run --rm -ti --name "web-server1" --hostname "web-server" docker.io/centos /bin/bash
s2 - cat /etc/hostname
s2 - hostname
s2 - exit

Container lifetime and Persistent data

==========

Containers are usually immutable and **ephemeral**, **just fancy buzzwords for unchanging and temporary or disposable**, but the idea here is that we can just throw away the container and create a new one from an image right!!!!.

Containers are Ephemeral and once a container is removed, it is gone.

What about scenarios where you want the applications running inside the container to write to some files/data and then ensure that the data is still present. For e.g. let's say that you are running an application that is generating data and it creates files or writes to a database and so on. Now, even if the container is removed and in the future you launch another container, you would like that data to still be there.

In other words, the fundamental thing that we are trying to get over here is to separate out the container lifecycle from the data. Ideally we want to keep these separate so that the data generated is not destroyed or tied to the container lifecycle and can thus be reused. This is done via Volumes, which we shall see via several examples.

So we are not talking about actual limitation of containers, but more of design goal or best practise, this is the idea of immutable infrastructure

So docker has two solutions to this problem known as **Volumes** and **Bind mounts**.

Working with volumes

==========

There are three main use cases for Docker data volumes:

1. To keep data around when a container is removed
2. To share data between the host filesystem and the Docker container

By **Docker Volumes**, we are essentially going to look at how to **manage data within your Docker containers**.

Few points about volumes
==========

- A data volume is a specially designed directory in the container.
- It is initialized when the container is created. By default, it is not deleted when the container is stopped. It is not even garbage collected when there is no container referencing the volume.

By **Docker Volumes**, we are essentially going to look at how to **manage data within your Docker containers**.

Few points about volumes
==========

- A data volume is a specially designed directory in the container.
- It is initialized when the container is created. By default, it is not deleted when the container is stopped. It is not even garbage collected when there is no container referencing the volume.

MySQL Volumes Example
==========

goto hub.docker.com
search for mysql and goto → Details → Click on latest Dockerfile → Scroll down and you can see **VOLUME /var/lib/mysql,** this is the default location of MySQL Databases.

This **mysql image** is programmed in a way to tell docker, when we start a new container from it, it actually **creates a new volume location** and **assign it to** this directory **/var/lib/mysql,** in the container,

Which means any files we put in the container will outlive the container, until we manually delete the volume.

Volumes need manual deletion, you can't clean them up just by removing the container that's an extra setup with volumes, the whole point of volume command is to say that this data is particularly important at least much more important than container itself.

Note: You might wanna do
 # docker volume prune
to cleanup unused volumes and make it easier to see what you're doing.

docker volume ls
docker pull mysql
docker image inspect mysql { **scroll down to see Mounts** }

Let's run a container from it:

docker container run -d --name mysql -e MYSQL_ALLOW_EMPTY_PASSWORD=True mysql
docker container ls
docker volume ls
docker container inspect mysql { you can see **Volumes /var/lib/mysql** }

And if you go up in the output, you can see **Mounts,** and this is actually the running container
so actually the **container actually thinks it's getting data or writing data is from /var/lib/mysql**,

But in this case, we can see the **data is actually living** in **Source** above line of /var/lib/mysql on the host.

So let's do:
docker volume ls
docker volume inspect 213b30c

If you are doing this on a linux machine, You can actually navigate to the volume **Source** location { **/var/lib/docker** } and can see the data, i.e some databases.

And if i do just hit an up arrow and create a multiple mysql container:

```
# docker container run -d --name mysql2 -e
MYSQL_ALLOW_EMPTY_PASSWORD=True mysql
# docker volume ls

# docker container run -d --name mysql3 -e
MYSQL_ALLOW_EMPTY_PASSWORD=True mysql
# docker volume ls

# docker container run -d --name mysql4 -e
MYSQL_ALLOW_EMPTY_PASSWORD=True mysql
# docker volume ls
```

We can see **two volumes** but we can see the problem right ??
There is **no easy way to tell which volume belongs to which container**.

```
# docker container stop mysql
# docker container stop mysql2
# docker container stop mysql mysql2

# docker container ls
# docker volume ls
 # docker container rm -f mysql mysql2 mysql3

# docker volume ls
```

{ my volumes are still there, my data is still safe, so we solved one prob }

```
# docker container run -d --name mysql-default -e
MYSQL_ALLOW_EMPTY_PASSWORD=True -v /var/lib/mysql mysql
```

[same like the volume command in the Dockerfile did, so we really don't need to do that here → This is how mysql image is built actually]

Check volume info in two containers:

docker container inspect mysql2

docker container inspect mysql-default

So how we make little more user friendly ??

That is where **named volumes** come in, the ability for us to specify names for docker volumes.

Named volume [i can put a name in front of it with **: that is known as named vol**]

docker container run -d --name mysql -e MYSQL_ALLOW_EMPTY_PASSWORD=True **-v mysql-db:/var/lib/mysql** mysql

docker volume ls

{ you can see my new container is using a new volume and it's using a friendly name }

docker volume inspect **mysql-db {** this is easier to use here**}**

And if i removed my container:

docker container rm -f mysql { -f coz it's still running, if not stop & remove }

And if i run another container with some other name and same volume:

docker container run -d --name **mysql3** -e MYSQL_ALLOW_EMPTY_PASSWORD=True **-v mysql-db:/var/lib/mysql** mysql

docker volume ls { **only mysql-db is there** }

You can see that we haven't created a new volume, but still using the same mysql-db volume from earlier.

docker container inspect mysql3

{ and we can see Volumes, changed the **Source** location to be a little friendlier as well }

More on Mysql - Volumes

==========

Running mysql container with password:

docker run --name some-mysql -e MYSQL_ROOT_PASSWORD=mypassword -d mysql

The following command line will give you a bash shell inside your mysql container:

docker exec -it some-mysql bash

mysql -u root -p

The MySQL Server log is available through Docker's container log:

docker logs some-mysql

Removing volumes along with container:

docker container rm -f -v some-mysql

Creating new mysql container **with password and volume**

docker container run -d --name mysql4 -e MYSQL_ROOT_PASSWORD=mypassword **-v mysql-db:/var/lib/mysql** mysql

docker exec -it mysql4 bash

mysql -u root -p { **create a database like ravi** and quit from db }
docker container stop mysql4
docker volume ls { database is still intact }

Creating another container with volume **mysql-db**

docker container run -d --name mysql5 -v mysql-db:/var/lib/mysql mysql
docker volume ls
docker exec -it mysql5 bash

mysql -u root -p
show databases;

Persistent Data: Bind Mounting
==========

Look at the same process of mounting a volume but this time **we will mount an existing host folder in the Docker container**. This is an interesting concept and is **very useful** if you are looking **to do some development where you regularly modify a file** in a folder outside and expect the container to take note or even sharing the volumes across different containers.

Bind mounts are actually cool, this helps how to use docker for local development.

So really a bind mount is just a mapping of the host files or directories into a container file or directory.
In background, it's just having two locations pointing to the same physical location(file) on the disk.

Full path rather than just a name like volumes, the way actually docker can tell the difference between named volume and bind mount, is that **bind mounts starts with a forward slash /** { root }

Now where it really comes to shine is with development and running services inside your container, that are accessing the files you are using on your host which you are changing. So let's do that with nginx:

Using multiple volumes on Single Container

==========

Saving both logs and website data

docker container run -v **~/nginx-logs:/var/log/nginx** -v ~/website:/usr/share/nginx/html nginx

Nginx

======

sudo docker run -d -P --name web-server nginx { **-P random ports** }

Nginx

=====

docker container run -d --name nginx1 -p 8080:80 nginx

docker container run -d --name nginx2 -p 80:80 nginx

Check the site in host machine http://ip-address:8080

The site which we are seeing is default nginx html file, actually coming from default nginx image.

Nginx with Mountpoints example

==========

docker container run -d --name nginx -p 80:80 -v ~/website:/usr/share/nginx/html nginx

Nginx Example (Volumes)

==========

mkdir nginxlogs

sudo docker run -d -v ~/nginxlogs:/var/log/nginx -p 80:80 nginx

I do this because every time i don’t want to go into the container and check the logs. Now i have all the logs in host machine itself rather than container.

Wordpress example - Backup
==========

Creating DB
docker run -d --name=**wp-mysql** -e MYSQL_ROOT_PASSWORD=mypassword -v ~/mysql-data:/var/lib/mysql mysql
docker exec -it wp-mysql bash
mysql -u root -p
 # create database wordpress;

Creating WP
docker run --name my-wordpress --link wp-mysql:mysql -p 8080:80 -d -v ~/wp-data:/var/www/html wordpress

Docker Custom Images
==========
To work with custom images go through the github url which is self explanatory:
https://github.com/ravi2krishna/Node-Js-Sample-App.git

NAGIOS

Nagios is a monitoring tool.
Now let's say you have 1000 systems in your infrastructure, so daily it's a tedious task to go and understand what is the status of these 1000 devices.

For monitoring all these servers we use NAGIOS.

What we monitor ??
==========
1. Health {device is up/down}
2. Performance {RAM & CPU utilization}
3. Capacity {Watch HDD capacity}

Parameters to monitor
==========
CPU
RAM
Storage
Network etc

NAGIOS SERVER SETUP
==========

NAGIOS CORE
==========
Goto nagios.org → Downloads → Nagios Core

On Host Machine
mkdir nagios-software
cd nagios-software

copy #link of nagios-core.tar.gz

wget <nagios-core-link>

extract the tar

sudo yum install httpd php php-cli gcc glibc glibc-common gd gd-devel net-snmp openssl-devel wget unzip -y

sudo useradd nagios
sudo groupadd nagcmd
sudo usermod -a -G nagcmd nagios
sudo usermod -a -G nagcmd apache
After extracting the tar file go to the directory
cd nagios-4.2.0
./configure
make all
sudo make install
sudo make install-init
sudo make install-config
sudo make install-commandmode
sudo make install-webconf
sudo cp -R contrib/eventhandlers/ /usr/local/nagios/libexec/
sudo chown -R nagios:nagios /usr/local/nagios/libexec/eventhandlers

Creating nagiosadmin user account

sudo htpasswd -c /usr/local/nagios/etc/htpasswd.users nagiosadmin

Checking for syntax errors and to see if everything is working fine

sudo /usr/local/nagios/bin/nagios -v /usr/local/nagios/etc/nagios.cfg
sudo /etc/init.d/nagios start
sudo systemctl start httpd

NAGIOS PLUGINS
==========

On nagios.org → Downloads → Nagios Core Plugin

wget <link-nagios-plugins>
extract the tar
cd nagios-plugin-x
sudo ./configure
make
sudo make install

NRPE PLUGIN
==========

To get monitor a system we are going to install NRPE plugin,
NRPE - Nagios Remote Plugin Executor

Goto nagios.org → Nagios core plugin → Find more plugins → General Addons → NRPE → Copy download URL

wget <link-nrpe>
cd nrpe
./configure

sudo make all
sudo make install
ls -l /usr/local/nagios/libexec/check_nrpe {installed successfully}
sudo service nagios restart

sudo systemctl restart httpd

To view Nagios server Dashboard : # ip-add/nagios

NAGIOS CLIENT/AGENT SETUP

==========

Use another linux machine either on cloud or vm

sudo useradd nagios
sudo yum install -y wget php gcc glibc glibc-common gd gd-devel make net-snmp unzip openssl-devel net-tools xinetd

NAGIOS PLUGINS

==========

On nagios.org → Downloads → Nagios Core Plugin

wget <link-nagios-plugins>
extract the tar
cd nagios-plugin-x
sudo ./configure
make all
sudo make install

NRPE PLUGIN

==========

To get monitor a system we are going to install NRPE plugin,
NRPE - Nagios Remote Plugin Executor

Goto nagios.org → Nagios core plugin → Find more plugins → General Addons → NRPE → Copy download URL

```
# wget <link-nrpe>
# cd nrpe
# ./configure
# make

# sudo make install


# sudo mkdir -p /usr/local/nagios/etc
# cd nrpe {dir - make sure you are in nrpe directory }
# sudo cp sample-config/nrpe.cfg /usr/local/nagios/etc
# cd sample-config
# sudo vi sample-config/nrpe.xinetd    { cd sample-config}
service nrpe
{
      flags           = REUSE
      port            = 5666
      socket_type     = stream
      wait            = no
      user            = nagios
      group           = nagios
      server          = /usr/local/nagios/bin/nrpe
      server_args     = -c /usr/local/nagios/etc/nrpe.cfg --inetd
      log_on_failure  += USERID
      disable         = no
     only_from       = 127.0.0.1 <ip-add-server>
}


# sudo cp sample-config/nrpe.xinetd /etc/xinetd.d/nrpe

# sudo vi /etc/services
  Add →  nrpe           5666/tcp              # NRPE service
```

chown -R nagios:nagios /usr/local/nagios
sudo service xinetd start
netstat -ntpl

Goto **Server machine** and do :

/usr/local/nagios/libexec/check_nrpe -H <ip-client> {if u get version its success }

Configuring Agent

==========

In Server machine

cd /usr/local/nagios/etc
sudo touch hosts.cfg
sudo touch services.cfg
sudo vi /usr/local/nagios/etc/nagios.cfg [goto OBJECT CONFIGURATION FILE(S)]
Add the following lines below templates.cfg

cfg_file=/usr/local/nagios/etc/hosts.cfg
cfg_file=/usr/local/nagios/etc/services.cfg

sudo vi /usr/local/nagios/etc/hosts.cfg

define host{
use generic_host ; Inherit default values from a template
host_name c1 ; The name we're giving to this server
alias CentOS 7 ; A longer name for the server
address 192.168.44.11; IP address of Remote Linux host
max_check_attempts 5;

}

sudo vi /usr/local/nagios/etc/services.cfg

define service{
use generic-service
host_name c1
service_description CPU Load
check_command check_nrpe!check_load
}

define service{
use generic-service
host_name c1
service_description Total Processes
check_command check_nrpe!check_total_procs
}
# sudo vi /usr/local/nagios/etc/objects/commands.cfg

Add the following to end of the file

# Command to use NRPE to check remote host systems
define command{
command_name check_nrpe
command_line $USER1$/check_nrpe -H $HOSTADDRESS$ -c $ARG1$
}

And check for any syntax errors

sudo /usr/local/nagios/bin/nagios -v /usr/local/nagios/etc/nagios.cfg

You can see Checked hosts

sudo /etc/init.d/nagios restart
sudo systemctl restart httpd

Assignments

==========

- Create a brand new git repository with name **hrms**
- Under hrms repo create a new html page with name home.html with some sample content.
- Create two branches with name **employers** and **candidates**
- In **employers** branch create three html pages emp1.html, emp2.html & emp3.html
- In **candidates** branch create three html pages can1.html, can2.html & can3.html
- I want all the files that are in both **employers** and **candidates** to be present in master branch.
- Now create a new branch with any name you want from **candidates** branch and in the new branch that is created i want you to perform some commits and merge them to **candidates** branch while doing so i want to see two types of merging strategies.
- Implement a scenario where a merge conflict occurs and fix the conflict.
- Fork the following repo https://github.com/ravi2krishna/CalculatorJavaApp.git
- In the Calculator application i want to ignore all the intermediate files like .class, .jar, etc only changes in src and pom should go to remote repo. [Hint : ignore]
- Tag the above application with 1.0 after the ignore is complete and push it to your forked repository.
- Generate artifact for the forked calculator and execute the artifact using the normal approach and also with maven plugin.

- Install and configure Nexus with two repos:
    - snps
    - rles
- Push the Snapshot version to snps with the latest code
- For rles, go with the new code in CalculatorJavaApp/src/main/java/com/ravi/cal/RaviCalculator/Calculator.java

  In line 40 paste this code:

  ```
  public long mulFucn(long first, long second){

      return first*second;
  }
  ```

  commit the changes and generate the new artficat and push this new artifact to rles repository in nexus.
- Generate a new maven archetype to go with web application and generate the artifact and deploy the artifact to Nexus and also to tomcat with plugin.
- Perform sonar analysis for the forked Calculator application

CHEF

When it comes to learning chef

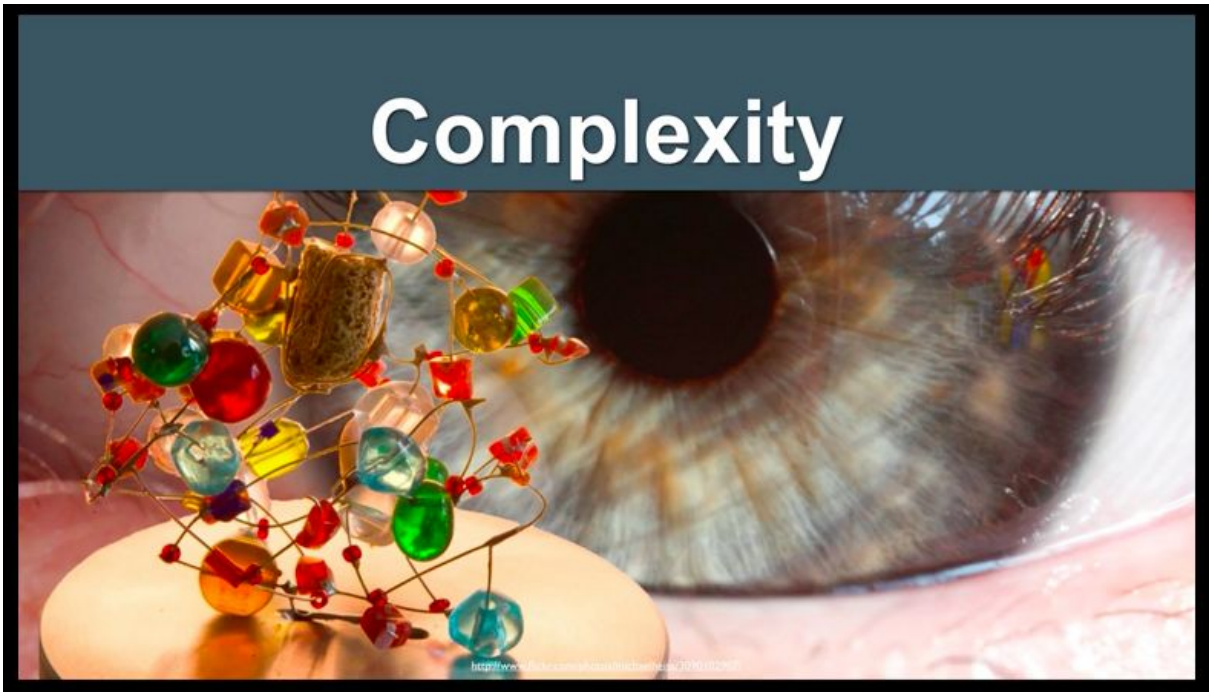
- You bring your business and problems
- You know about your infrastructure, you knew the challenges you face within your infrastructure and you know how your infrastructure works
- Chef will provide a framework to solve those problems

The Best way to learn chef is to use chef.

In chef terminology is very important

COMPLEXITY

==========

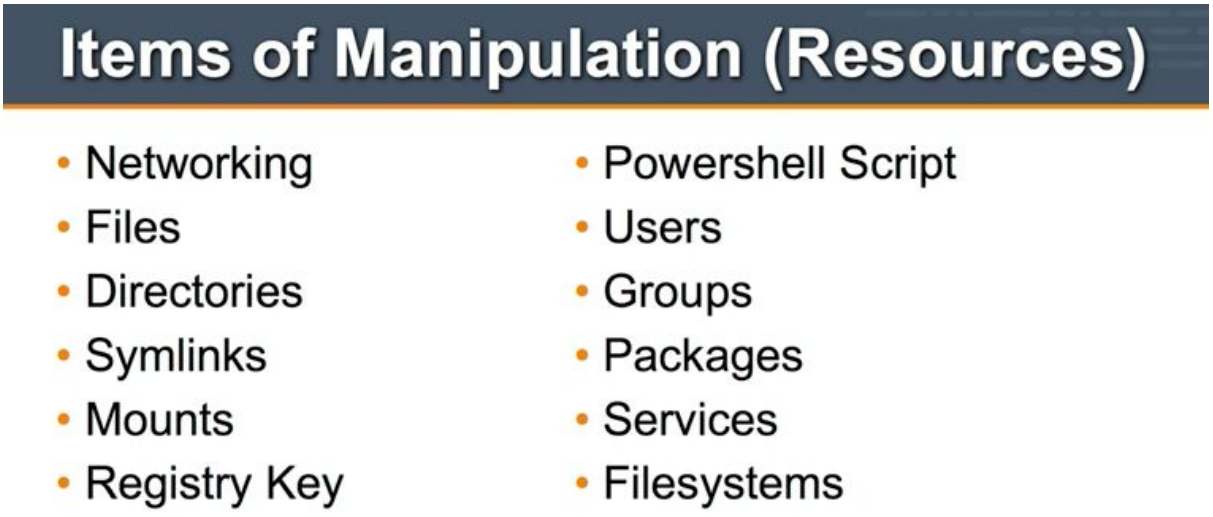


System administrators will have a **lot of complexity** to manage.
This complexity can be from many **items** (**Resources**) across the infrastructure.
We call these items as **resources**.

Resources

=========

The **resources** in your infrastructure can be **files**, **directories**, **users** that you need to manage, packages that should be installed, services that should be running and list goes on.



Let's look at typical application

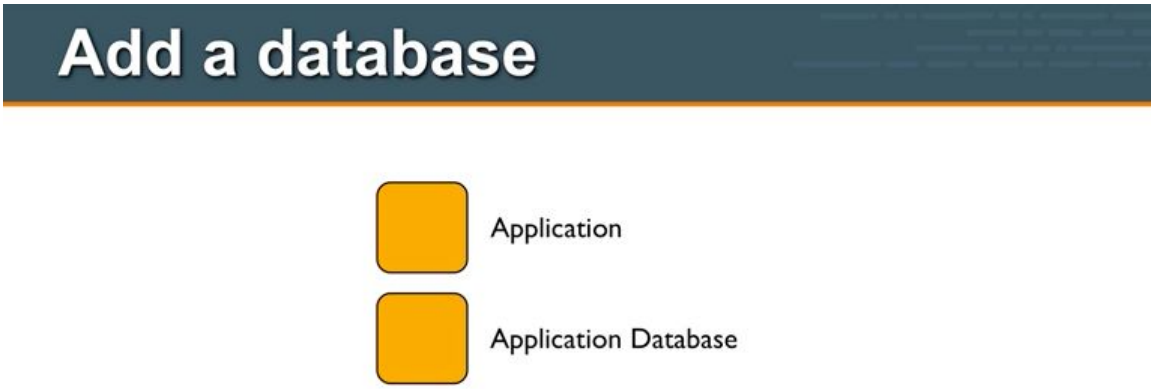
==========

You usually start configuring and installing that **application** on a single server(node).

To set up this node we may need to install packages, manage their configurations, installing a database, installing web server, installing application server and lots of things to make this application up and running.

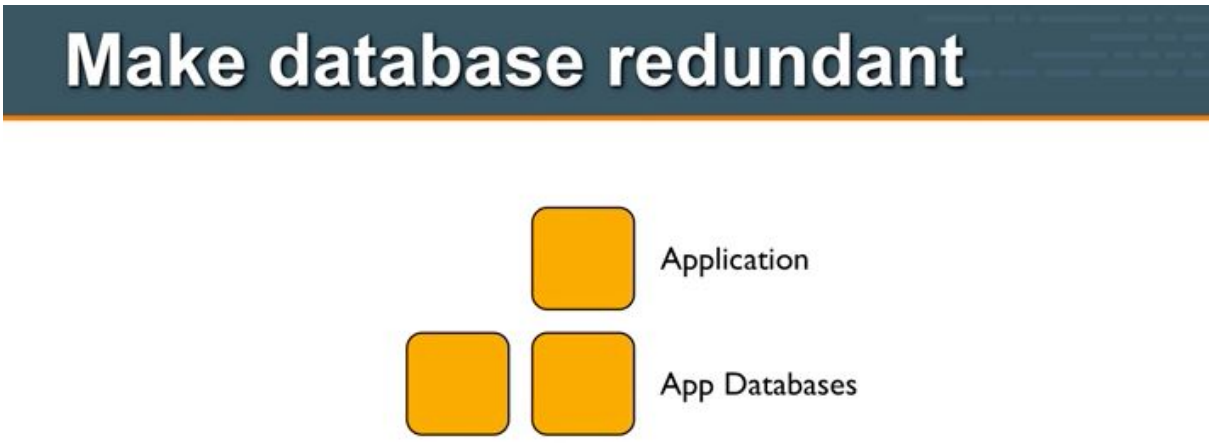
Now overtime you wanna take this application and make it **available to the public/client**.

So you are going to have a **database server**, so maybe now we are going to have multiple environments like staging, dev, qa or even production i.e multi tier environment.

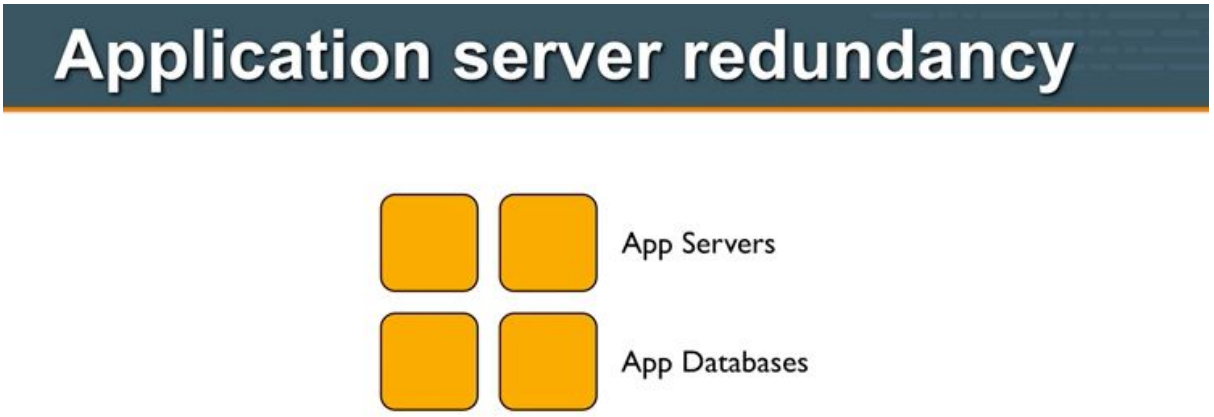


We have one server that handles all the Application requests and a separate server for Database.

Of course once we have a database server, **we want to make sure we don't lose data**, so we decide to **make database server redundant**. We want to **prevent failure and loss of the data.**



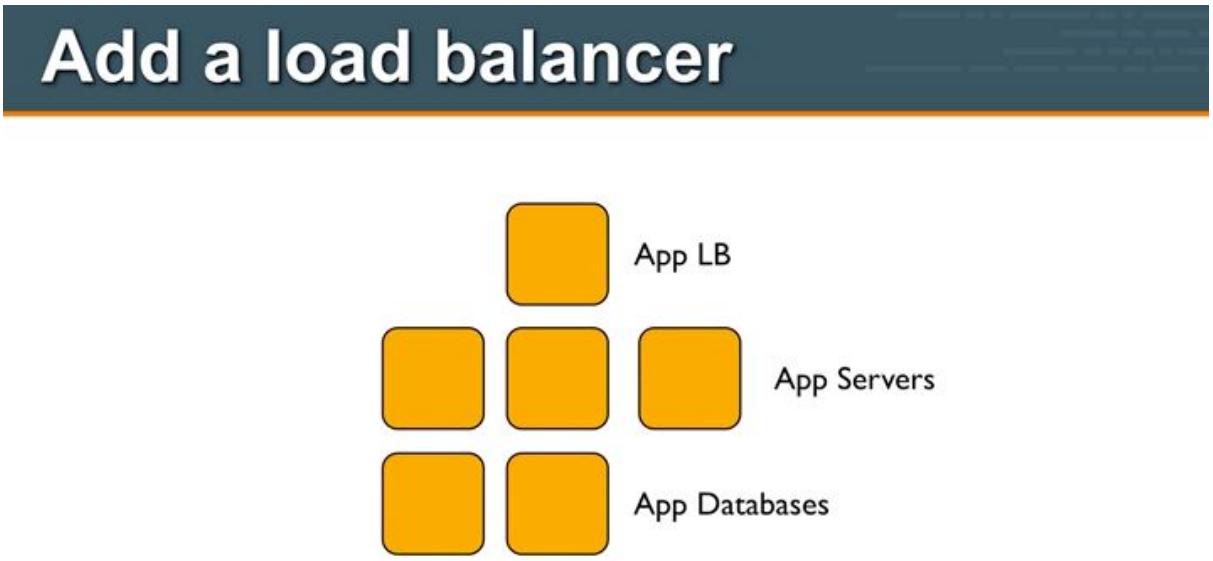
We may as well make the Application server redundant as well.



So now we are up to 4 Servers.

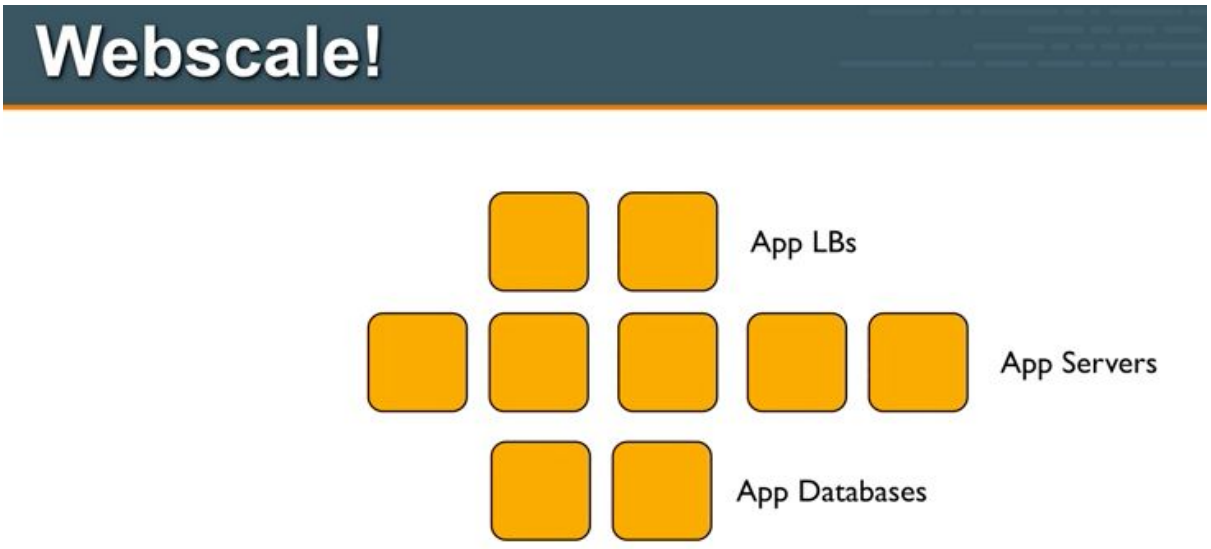
========

But of course as time goes on, load increases on our application, and we need to scale out the no of application servers that we have and in order to do that we need to put a load balancer in front of our app servers, so that requests can be evenly distributed.



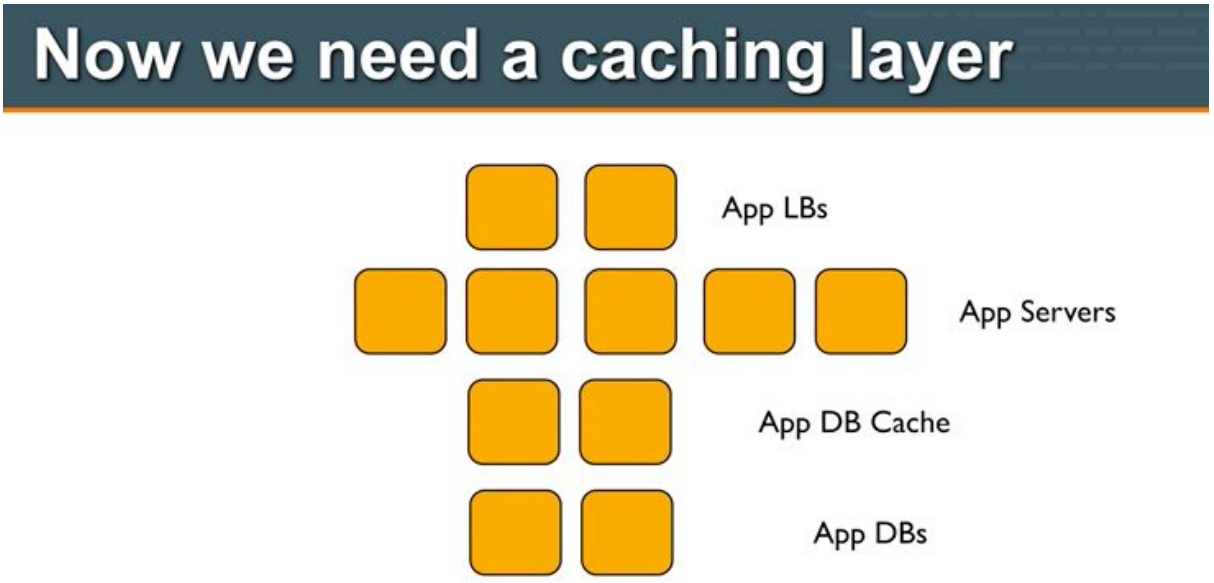
==========

Eventually we gonna reach web scale, this application has grown and grown and getting bigger and bigger and we are now at **web scale**. Look at all these servers now we have to manage.



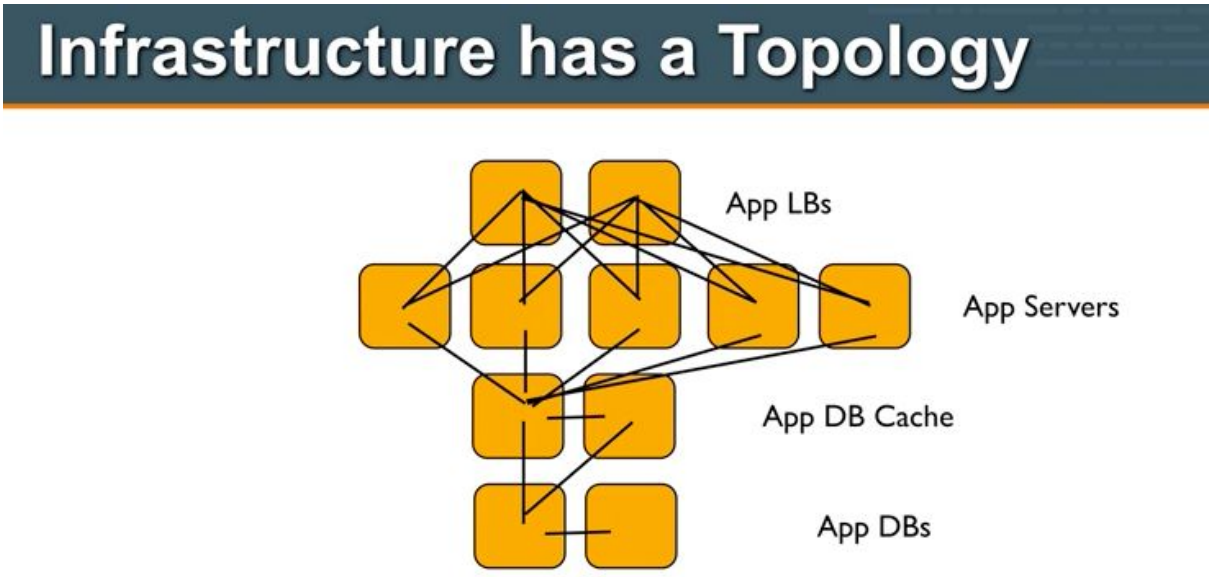
========

Now we ran into another problem, our database just isn't keeping up with the demands from our users, so we decided to add a caching layer.
Now i got even more servers and more complexity to infrastructure.



==========

As if this no of servers are not complex enough, and of course each infrastructure has its own topology. There are connections between the load balancer and Application servers.



The load balancers need to know which application servers to talk to, which application server they are sending requests to.

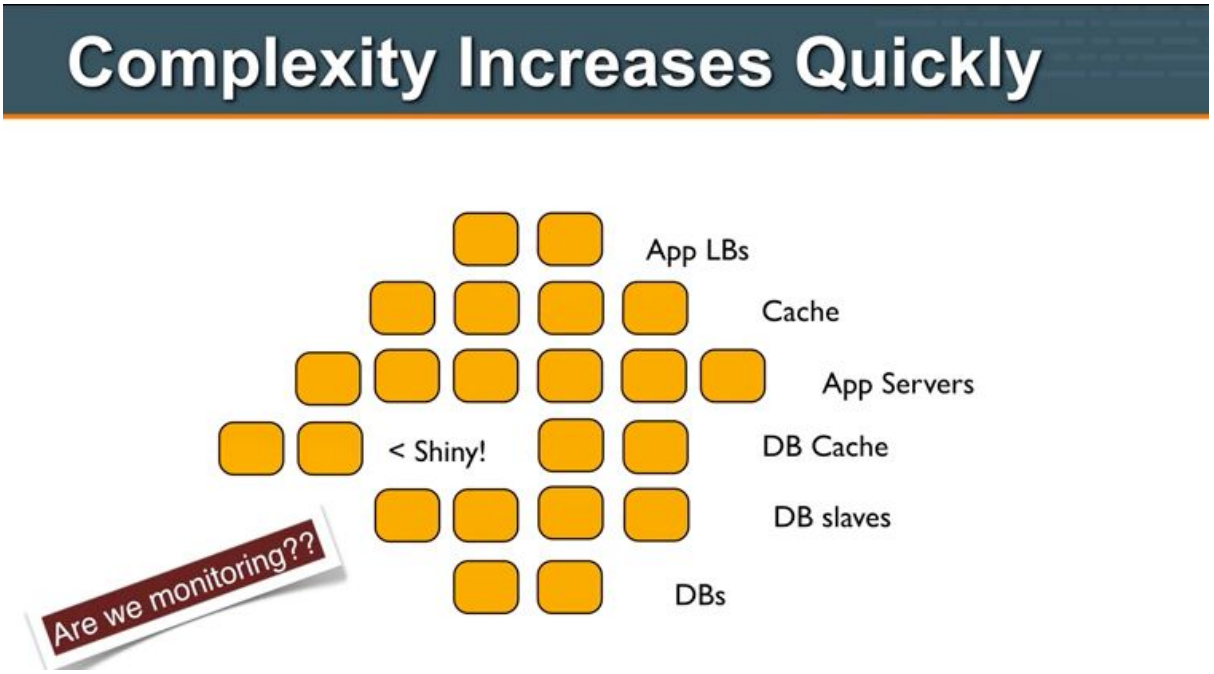
The Application server in turn needs to know about the database server.

The Database cache servers need to know what are the backend database servers which they need to do caching.

All of this just adds up more and more complexity to your infrastructure.

=========

And your complexity is only going to increase and its increasing quickly.



==========

So chef solves this problem for you.

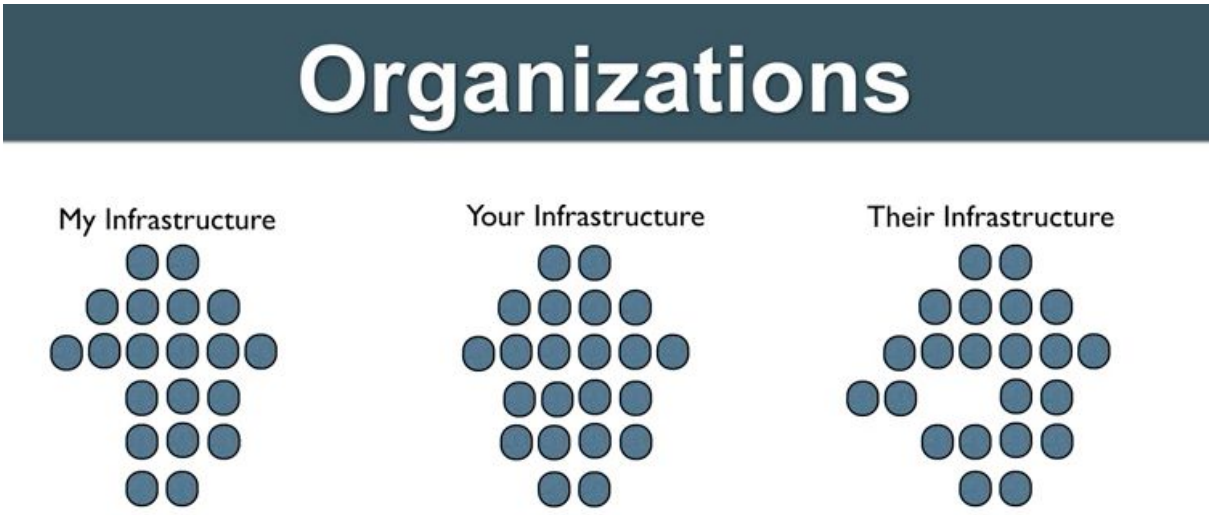
========

So how can we manage complexity with chef ??

Chef gives us a no of items/tools with which we can manage this complexity, we are going to look at this

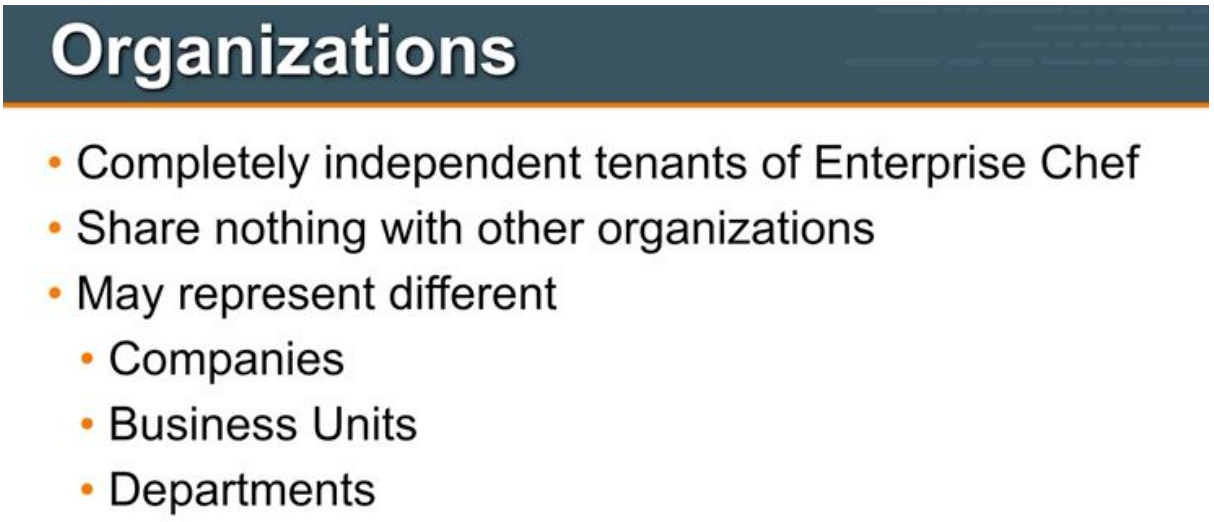
We are going to look at organizations, and how organizations can help you manage complexity.

So let’s start with top level, let’s see Organizations:



If you think about organizations, let’s take digital lync itself it has its own infrastructure and TCS has its own infrastructure etc etc

=====

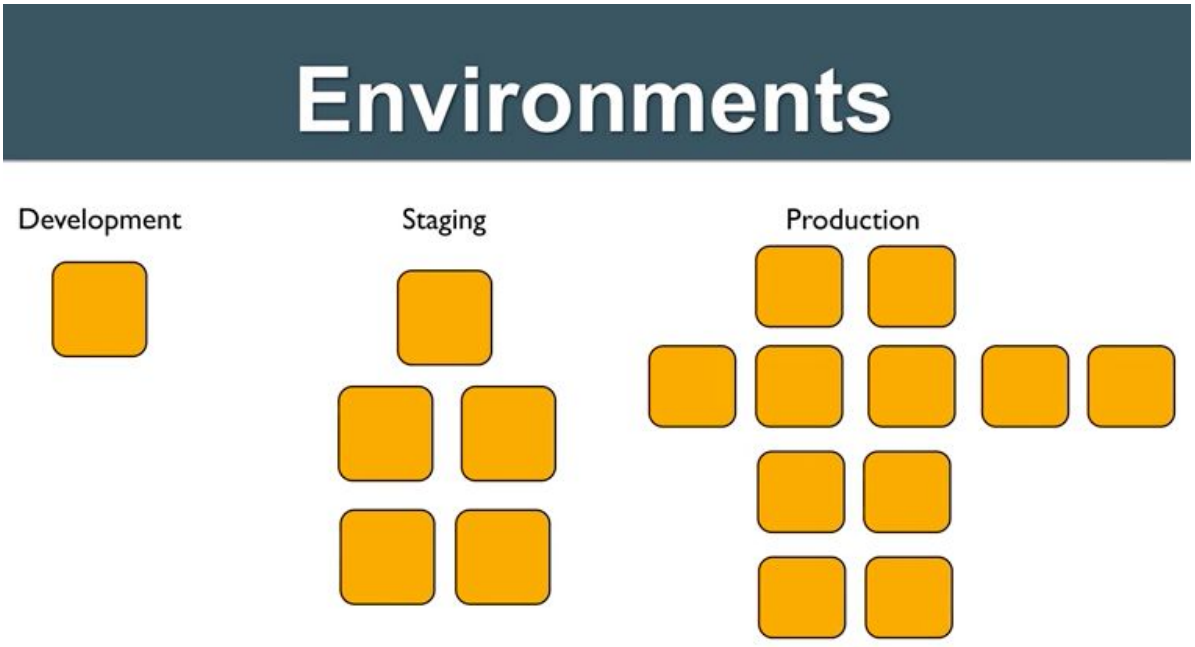


Organizations are independent tenants on enterprise chef.

Nothing is shared across the organizations, your organizations may represent different companies take Tata group as example.

=======

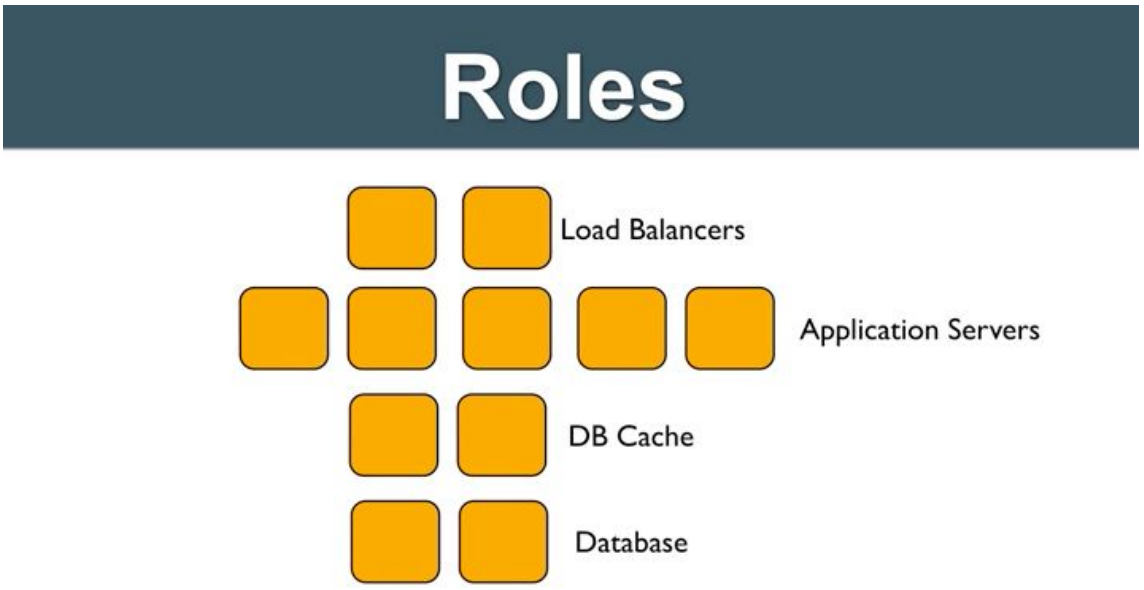
The next layer down is environments.



========

Next comes Roles

Roles is a way of identifying or classifying different types of servers that you have within your infrastructure.



Roles

- Roles represent the types of servers in your infrastructure
  - Load Balancer
  - Application Server
  - Database Cache
  - Database
  - Monitoring

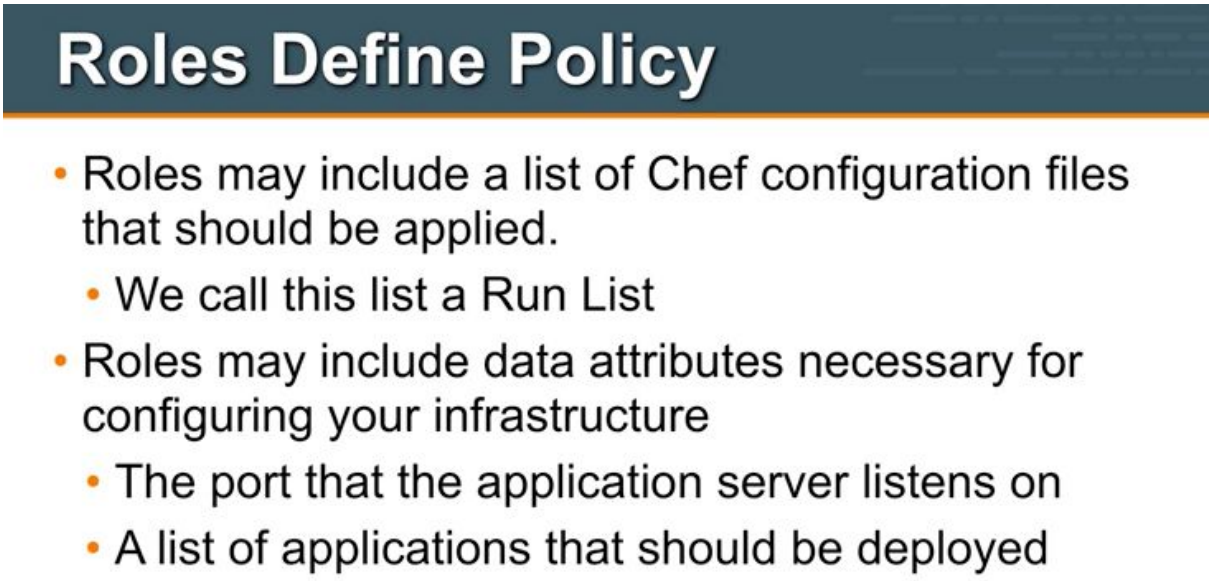
Within your infrastructure you have multiple instances of servers that are in each one of these roles, you may have multiple application servers, multiple database servers etc

You will specify this as roles in chef.

==========

Roles allow you to define policy, they may include list of chef configuration files that should be applied to the servers or nodes that are within that role.

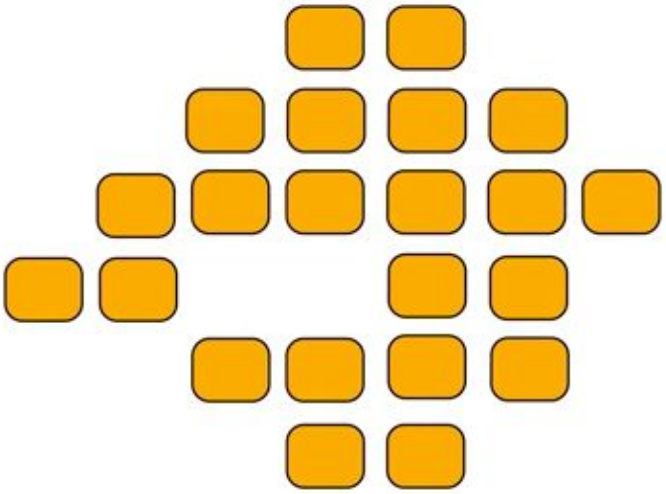
We call this list of chef configuration files that should be applied we call this a Run list.



==========

Stepping down from roles then we look at nodes

Nodes



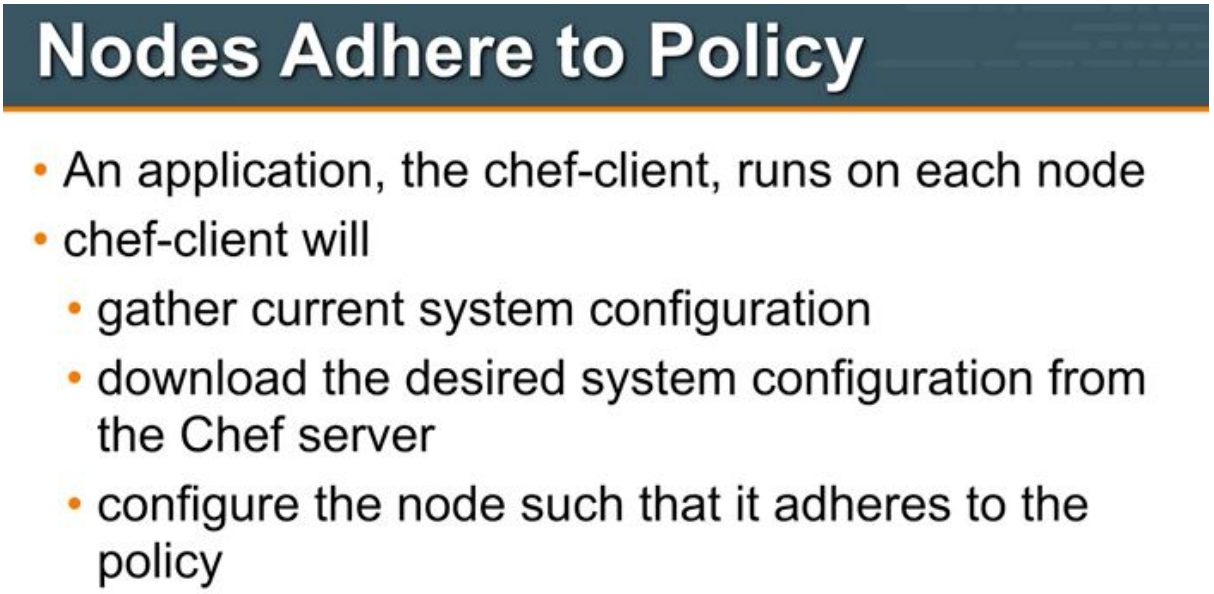
Nodes

- Nodes represent the servers in your infrastructure
- Nodes may represent physical servers or virtual servers
- Nodes may represent hardware that you own or may represent compute instances in a public or private cloud

Belong to Environment : Server can be either in dev or testing or staging or production

Roles : Each node may have zero or more roles, so a node maybe a database server or an application server or it can have both roles the same node can have both database and the application servers.

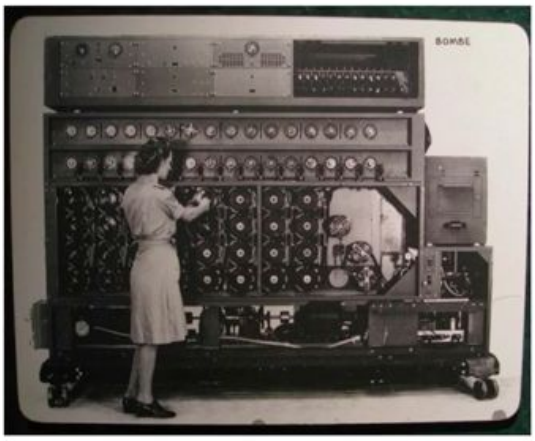
==========



Chef client will do all of the heavy lifting, it will make updates, it will configure the node, such that it adheres to the policy that is specified on the chef server.

========

Chef is Infrastructure as Code

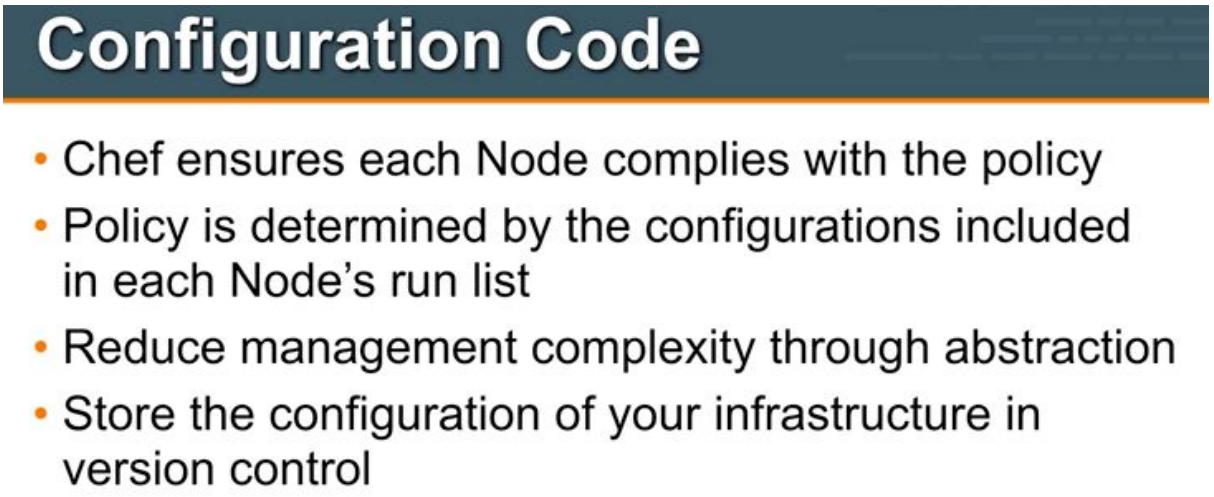


http://www.flickr.com/photos/louisb/4555295187/

- Programmatically provision and configure components
- Treat like any other code base
- Reconstruct business from code repository, data backup, and bare metal resources.

By capturing your infrastructure as code, you can reconstruct all of your business applications from three things a code repository, a backup of your data and the bare metal resources or compute resources that are required to run your applications. This puts you in a really really nice state and it’s a great way to manage your complexity of your infrastructure, with these three simple things i can rebuild my applications.

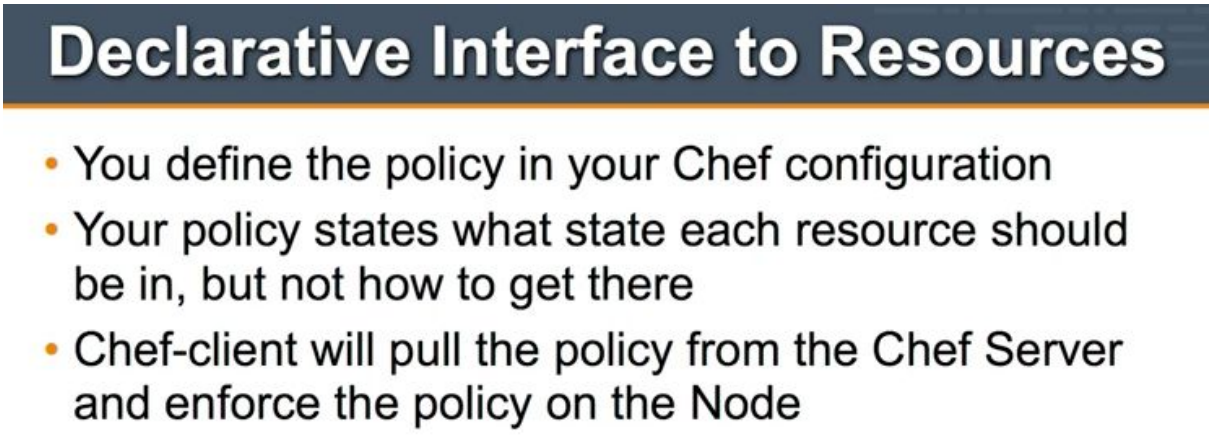
==========



Store the configuration of infrastructure in version control:
So gone are the days when you have a server that was hand crafted lovingly by a system administrator and that system administrator is the only person in your organization that knows all of the knobs, dials, tweaks, tricks, packages etc that have been placed onto that server, now we can take the knowledge of that system administrator has and move it into a framework that can be stored in a Source code repository.

Framework that allows for abstraction so that you can build up bits and pieces and transform the way you manage your infrastructure.

==========



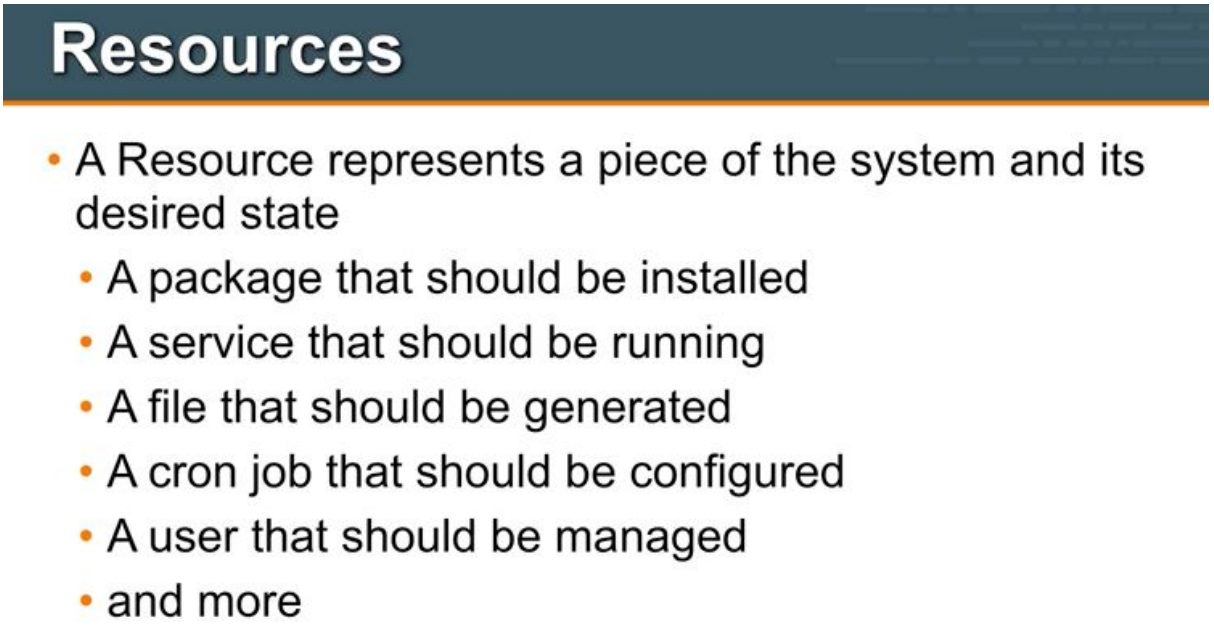
With chef you can define policies that your infrastructure should follow,

Your policy states what each resource should be in, but not how to get there
For example we will in our chef configuration file, we will say that a package should be installed but we will not need to specify how to install that particular package, chef is smart enough to sort that out.

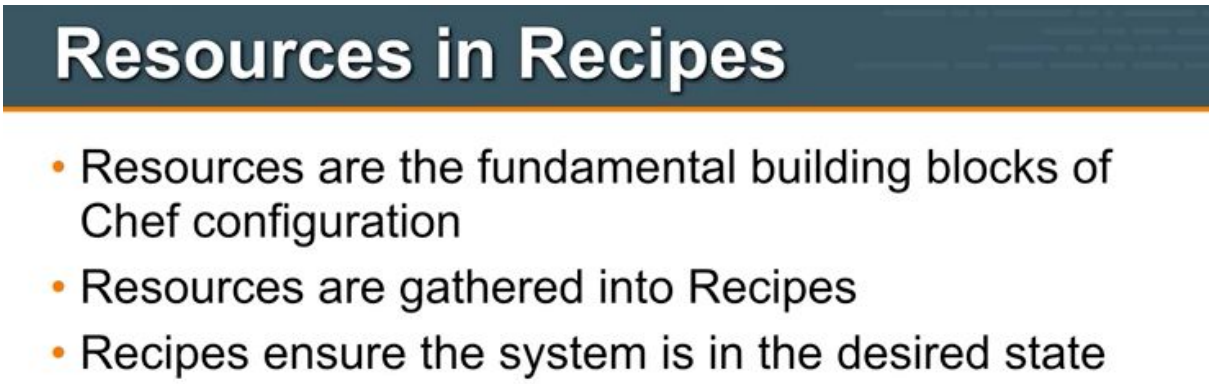
So for example if you want to install a package on a debian based system you may use apt package manager to install that package, # apt-get install <pkg-name> and If you are running on a redhat based distribution such as centos then apt will not work here we would rather use yum package manager # yum install <pkg-name>

Chef abstracts all of that away from you so that you can write configuration files that will work across the various platforms.

========

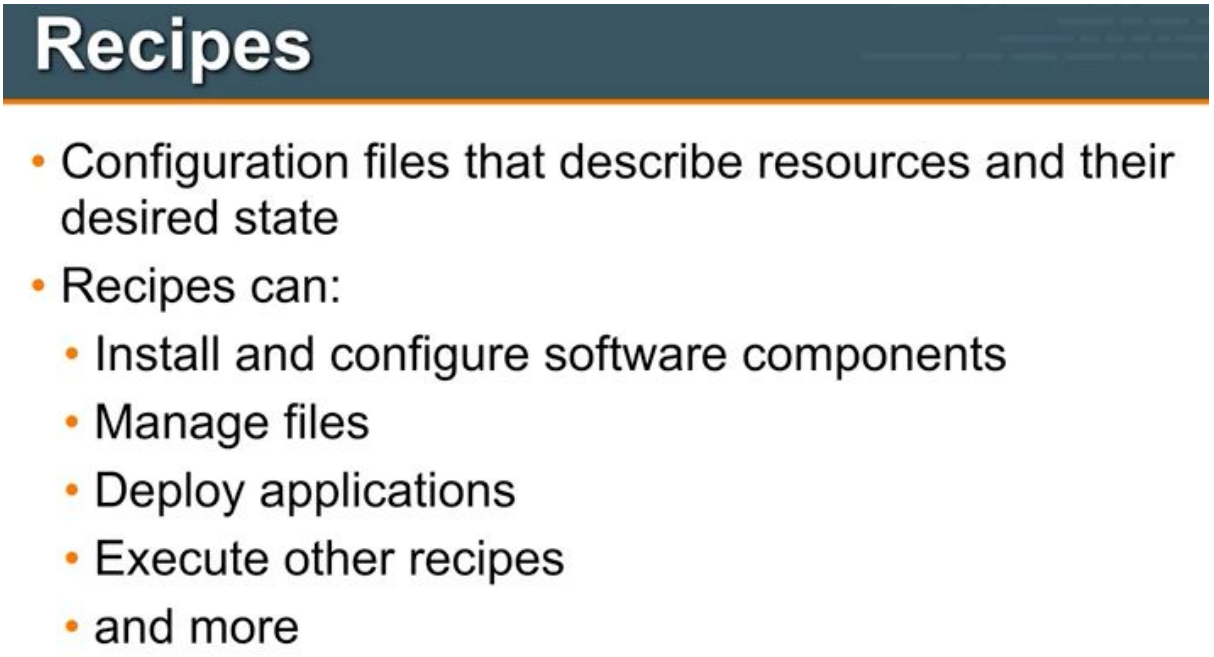


========



You will take resources and gather them together into recipes.

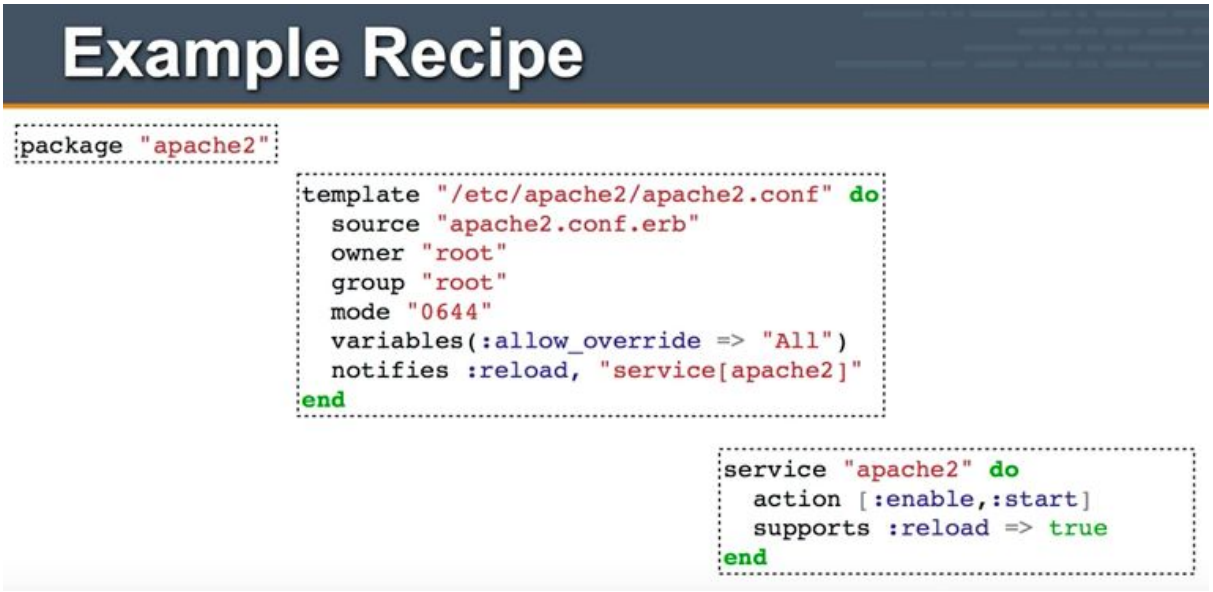
=========



Recipes are the real work forces within chef.

So resources are the building blocks and will take those building blocks and we would gather them together into recipes that would help bring our systems in line with policy.

Let’s see some example recipe code



I will walk through you what happens when the chef client encounters this recipe code

So **Chef-client** is an application that runs on the **nodes** within your infrastructure. It will gather it's policy, where its run list from the chef server and it will inspect current system configuration and it will then execute through this recipes or walk through these recipes and ensure the node is in desired state or the node complies with policy.

The first resource this recipe includes is a package resource and this package is named apache2, when the chef client sees this, it knows that the package named apache2 should be installed on this particular server/node, if it is not installed the chef-client will go ahead and installs that for you.

Again we are not telling the chef-client, how to install apache2, it's smart enough to figure that out on its own.

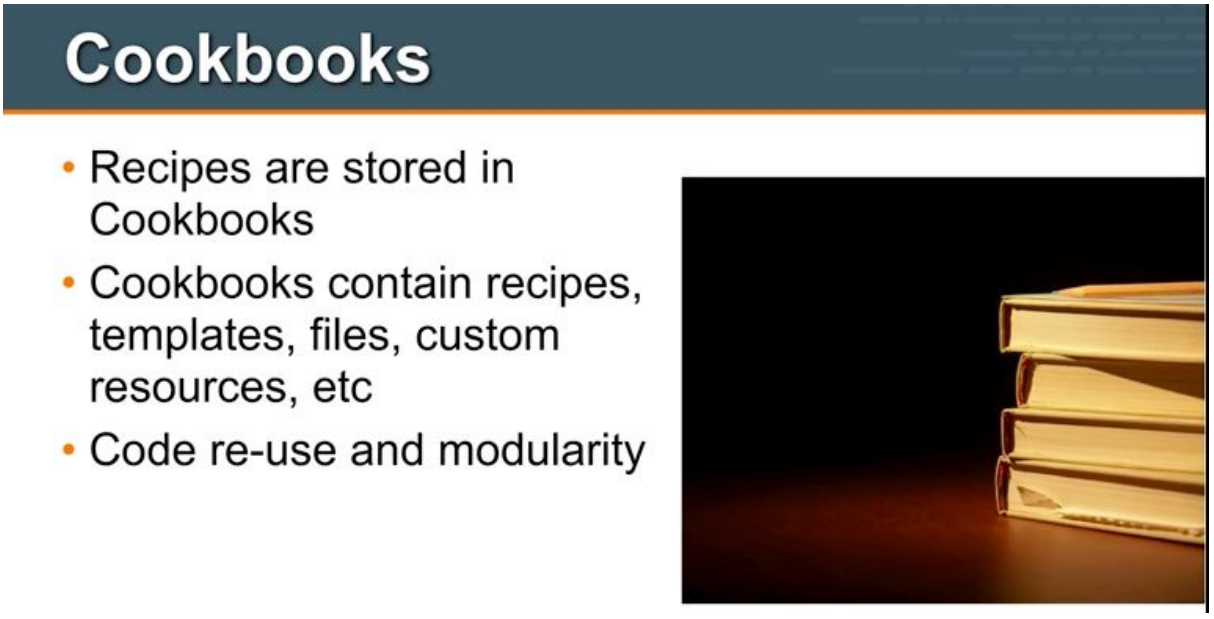
So our policy states that the package apache2 should be installed, if this is the case already, then the chef-client will move on to the next resource in our recipe.

If that package is not yet installed, chef-client will take care to install it and then move on to next resource in our recipe.

With chef we are going to take this recipes and package them up into **cookbooks**.

=========

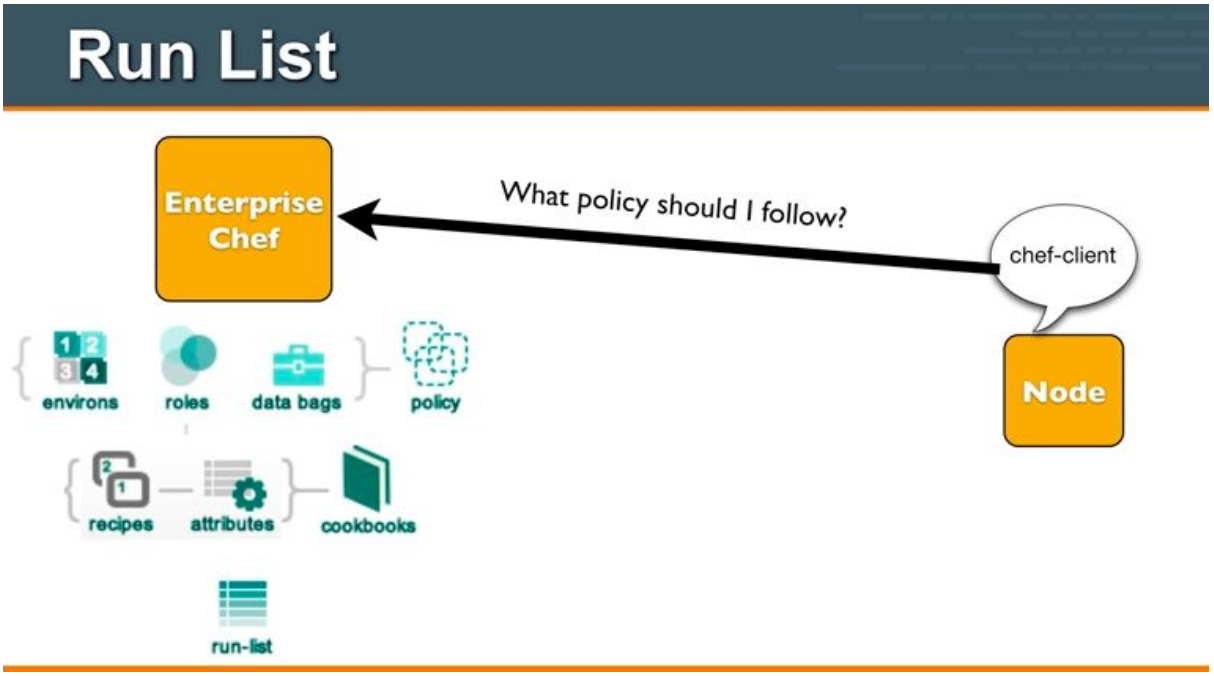
So a **cookbook** is a container that we will use to describe, configuration data and configuration policies about our infrastructure.
A cookbook may certainly contain recipes, but it can also include templates, files etc



So the recipe we just looked at had **template resource** in it, the template resource itself had a source file apache2.conf.erb, that source file is stored as part of the same apache2 cookbook.
This cookbooks allow code reuse and modularity.

========

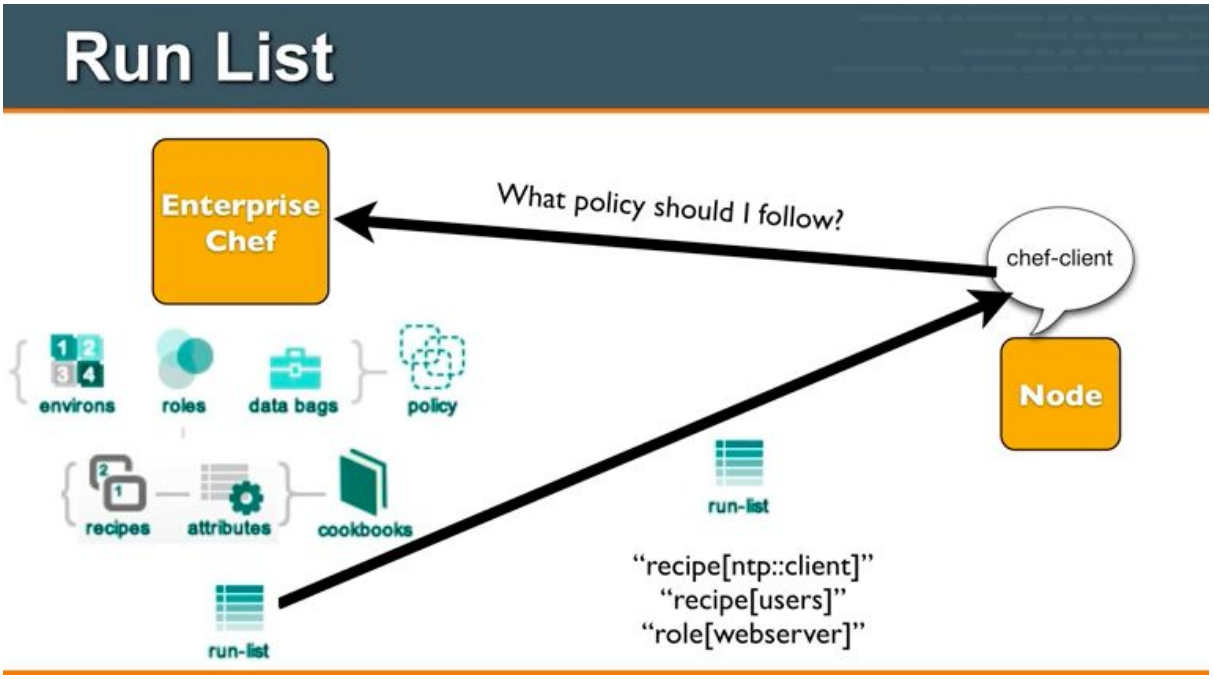
Let's look what happens at very very high level, when the chef-client runs on the node.



So node is a server in our infrastructure, on the node we have an application called chef-client, this is typically configured to execute on a regular interval maybe every 15-30 min using cron job.

When chef-client executes it will ask the enterprise-chef(chef-server), what policy should i follow, all of our policy is described on the chef-server, our policy includes things like our environments, our roles and our cookbooks.

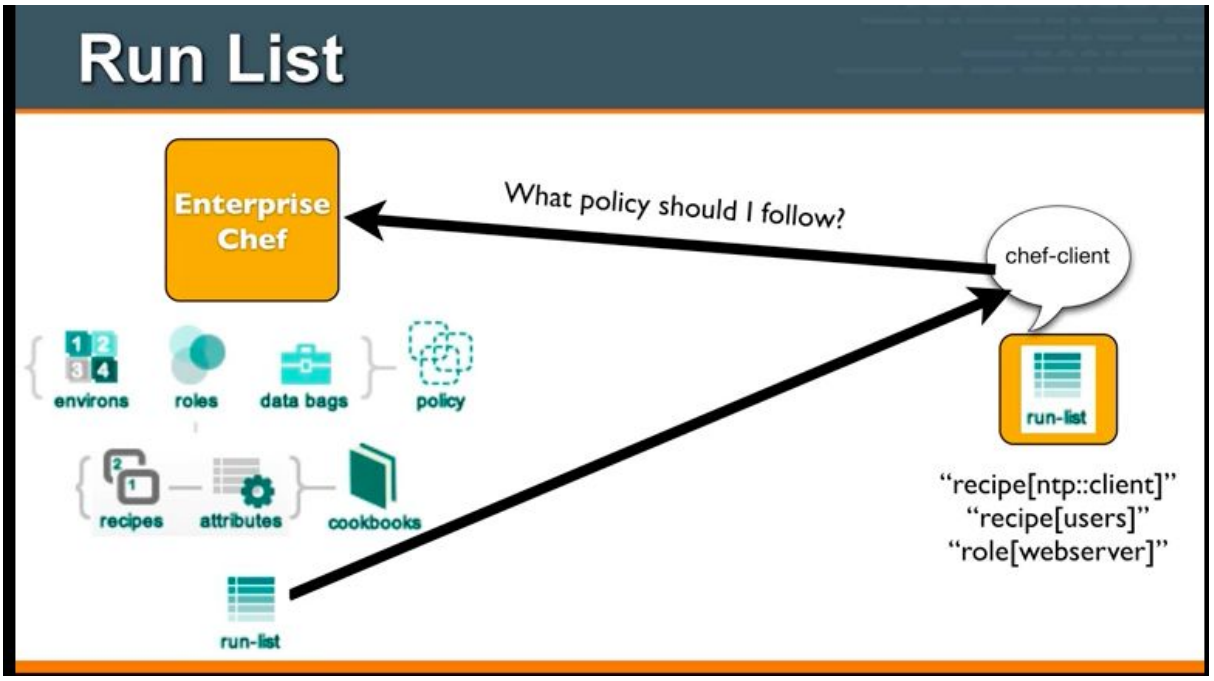
So the joining of a node, to a set of policies, we call that as a **run-list.**



So the chef-client will download all of the necessary components, that make up the runlist and will move those down to the node, so you'll see here the run-list includes the recipe ntp-client, recipe to manage users, and the role to make this node a web server.

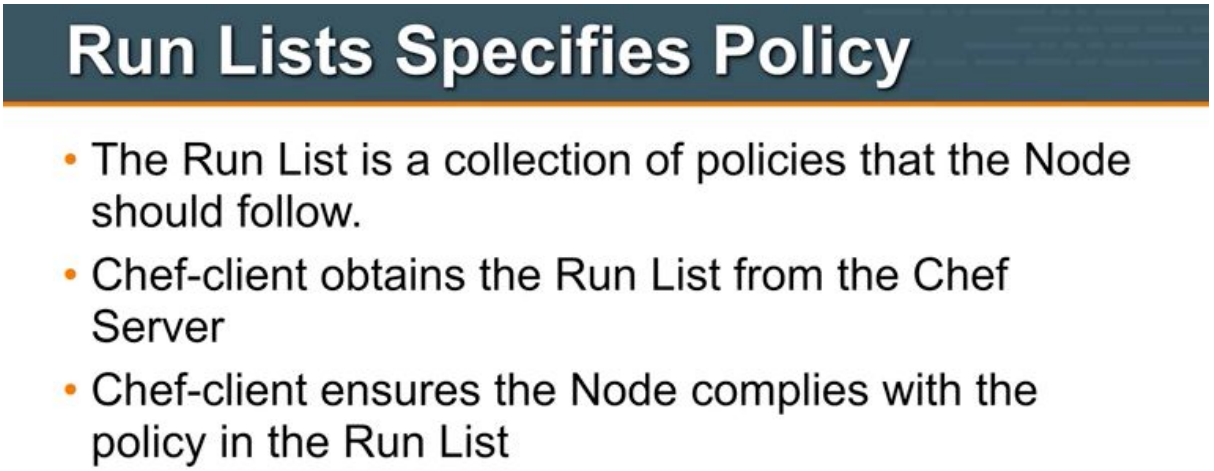
==========

Once the run-list has been downloaded to the node,the chef-clients job is to look at each of the recipes within that run-list and ensure that policy is enforced on that particular node, so it brings the node inline with policy.



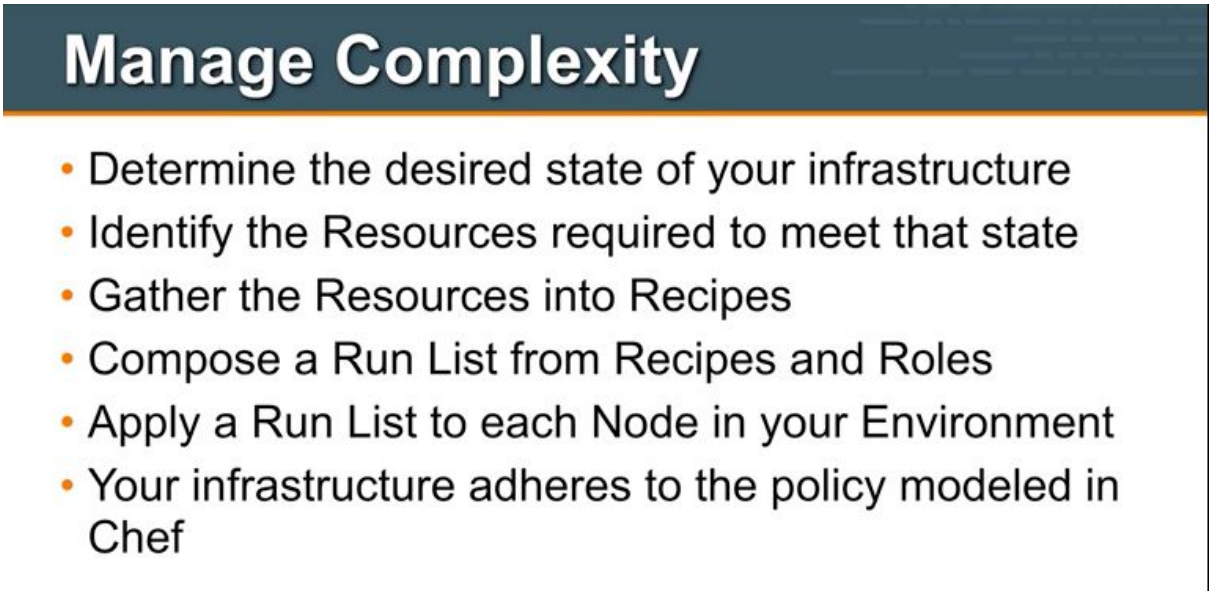
=========

So run-list is, how we specify policy for each node within our infrastructure.
The run-list is a collection of policies that node should follow.
Chef-client will obtain this run-list from the chef server.
Then chef-client ensures the node complies with the policy in the run-list.



=====

So with Chef this is how you manage complexity:



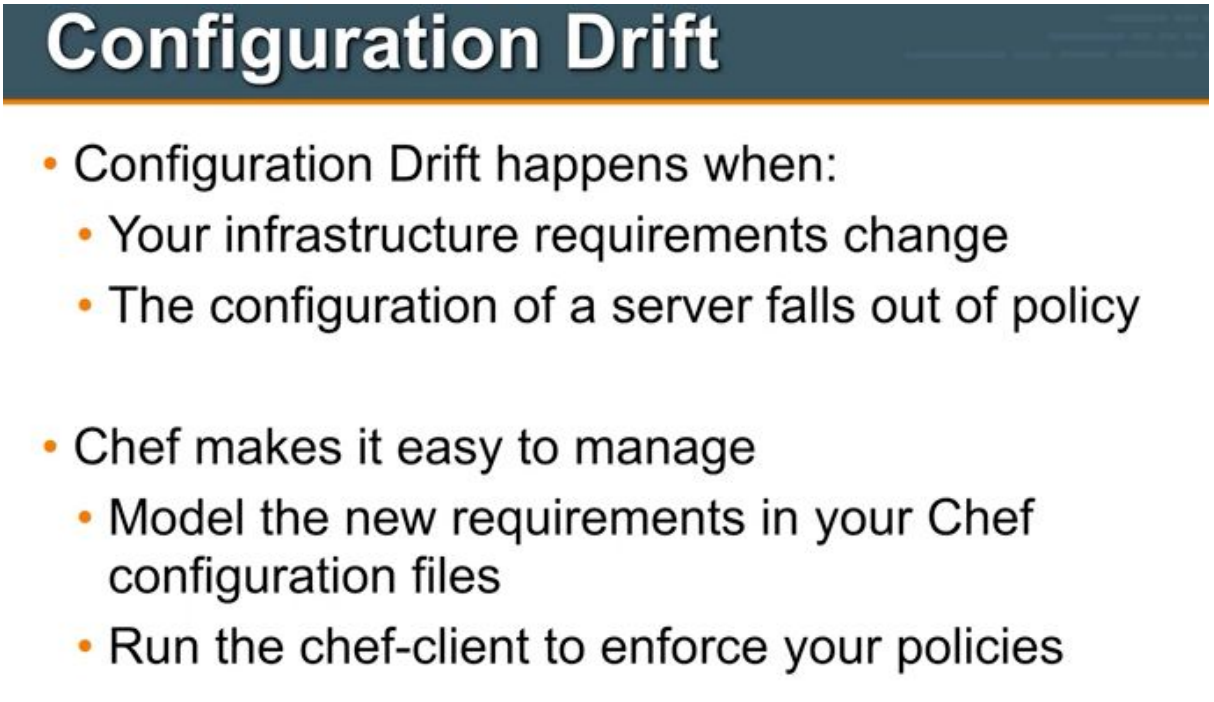
The first thing you will do, is to determine the desired state of your infrastructure. One we done that, you will identify the resources required to meet that state, what resources are required to meet that state, what resources are required users, services, packages etc.
You gather up each of these resources into recipe.
Again the run-list is the thing that gets applied to the node, you apply a run-list to each node within your environment and as the chef client runs on those nodes, your infrastructure will adhere to the policy modeled in the chef.

This is a great way to launch new servers to add the capacity to your infrastructure, this is how you add new capacity to an existing infrastructure.

Adding new capacity to an existing infrastructure is not, the only challenge that we face, the other challenge we face is **Configuration Drift.**

=======

Configuration drift happens when your infrastructure requirements change.
No infrastructure has static list of requirements, the requirements are certainly going to change over time.



Additionally a server within your infrastructure may fall out of line with policy, perhaps a system administrator logged into a server and changed the port number that an application was listening to, maybe that shouldn't have happened it's outside the policy, how do we address that, well chef makes it very easy to manage this, we are going to model the new requirements that we have in chef configuration files and then re-run the chef-client, so when chef-client runs it will enforce that each node within your infrastructure is following the current and accurate policy as stored on chef server, so overtime you can manage change across the infrastructure and you can enforce this policies across the infrastructure.

In open source we can manage up to 25 nodes freely.

On **workstation** side we use chef development kit (**chefdk**)
Chef-server we can work directly on cloud and we use **chef-manage**
On **nodes** we use **chef-client**

So we totally need three systems to workout.

Workstation	**Chef-Server**	**Node**
MACH1	MACH2	MACH3

Install Chef-Server
==========
Let's get the Chef-server from https://downloads.chef.io/chef-server

mkdir chef-sw
wget <link-of-rhel-7-distro-rpm>
sudo rpm -ivh chef-server.rpm

Once we installed it, we need to configure it by using command:

 # sudo chef-server-ctl reconfigure

Above command will set up all of the required components, RabbitMQ, PostgreSQL DB, SSL Certificates etc.

Once it is done successfully, it means server has been set successfully.

status of chef server we can use: **# sudo chef-server-ctl status** which is going to give all the services which are running in the background.

Now let's access the chef-server by ip add https://192.168.33.10/

Now **let's setup the web interface** to **manage chef-server**, to **get the GUI** we need to install another package which is **chef-manage**.

 # sudo chef-server-ctl install chef-manage

This installation of chef-manage will glue the chef-manage to chef-core means it will integrate chef-manage with your chef-core.

 # sudo chef-manage-ctl reconfigure

Next accept the license agreement and go on with further steps **say q for quit and yes**.
Once the installation is successful use the ip of machine to see the web interface:

https://ip_address_machine

Create admin user and organization
==========

Creating User
==========

We need to create an admin user. This user will have access to make changes to the infrastructure components in the organization we will be creating.
Below command will **generate the RSA private key** automatically and should be saved to a safe location.

 # sudo chef-server-ctl user-create --help

 # chef-server-ctl user-create <USER_NAME> <FIRST_NAME> <LAST_NAME> <EMAIL> 'PASSWORD' -f PATH_FILE_NAME

 # sudo chef-server-ctl user-create admin admin admin admin admin@digital.com password -f /etc/chef/admin.pem

pem file is your private key which will authenticate you while logging into the server and the corresponding public key will be stored in the server. We should not share this pem file coz now anyone with this pem file can login into the server.

Now you have created the user successfully.

Creating Organization

==========

It is the time for us to create an organization to hold the chef configurations.

 # sudo chef-server-ctl org-create --help

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

 # sudo chef-server-ctl org-create dl "Digital-lync-academy" --association_user admin -f /etc/chef/dl-validator.pem

That's it this is how we will be creating organization name.

Let's access the web interface now.

https://ip_address_machine

Once we logged in we can see **Nodes**, but we didn't setup any node so will see this in later part.

We can see the **organization**, on the top right corner.

Setting Up Workstation

==========

Setting up workstation is very important coz even if you don't know how to set up chef-server it's okay coz in real time chef-server is already set-up but work station we need to set up yourself.

Steps involved

==========

- Create a new centos machine for workstation
- Login to the workstation machine
- **Setup hostname and add all three in /etc/hosts**
- Download chef-dk on workstation (laptop/desktop/vm)
    - https://downloads.chef.io/chefdk
    - # **wget <link-of-chef-dk>**
    - **# sudo rpm -ivh chefdk-1.3.43-1.el7.x86_64.rpm**
    - **# sudo chef-client -v**
- Download chef-dk on workstation (laptop/desktop/vm)
- Install the chef-dk through rpm
- **# cd ~** { workstation }
- Generate Chef-Repo using "**chef generate repo**" command.
    - **# chef generate repo chef-repo**
    - This command places the basic chef repo structure into a directory called "**chef-repo**" in your home directory.
- In chef server we created two pem files one is user and another is organization, now we need to **bring this two pem files into the workstation**
- Now we need to create a directory **.chef** in chef-repo where we put RSA keys,
    - **# cd ~/chef-repo; # mkdir .chef; cd .chef;**
- Copy the RSA keys to the workstation **~/chef-repo/.chef**
    - **Make the handshake b/w chef-server & workstation**
    - **# scp /etc/chef/*.pem vagrant@workstation:~/chef-repo/.chef/**
        - **Run the above in chef-server**

- Knife is a command line interface for between a local chef-repo and the Chef server. To make the knife to work with your chef environment, we need to configure it by **creating knife.rb** in the "**~/chef-repo/.chef/**" directory.
    - Goto Web interface of chef-server → Click **Administration** → Select **Organization (dl)** → Click on **settings** wheel(extreme right) → Click on the **Generate Knife Config**
    - Copy the file contents and paste it in **~/chef-repo/.chef/knife.rb**
- **Testing knife:** test the configuration by running **knife client** list command. Make sure you are in **~/chef-repo/** directory.
    - **# cd ~/chef-repo/**
    - **# knife client list**
    - **# knife ssl check**
    - To resolve this issue, we need to fetch the Chef server's SSL certificate on our workstation
    - **# knife ssl fetch**
        - The above command will add the Chef server's certificate file to trusted certificate directory. (**# tree .chef**)
- Once the SSL certificate has been fetched, run the previous command to test the knife configuration.

 # knife client list { output then verification completed successfully **}**

Setting Up Node / Bootstrapping new node with knife
==========

Bootstrapping a node is a process of installing chef-client on a target machine so that it can run as a chef-client node and communicate with the chef server.

From the workstation, you can bootstrap the node with elevated user privileges.

 # setup new machine
 # Setup hostname and add all three in /etc/hosts
 # Install wget
Make **handshake between workstation to node1** using vagrant user in both machines

 # knife node list {nothing is seen **run this command in workstation**}

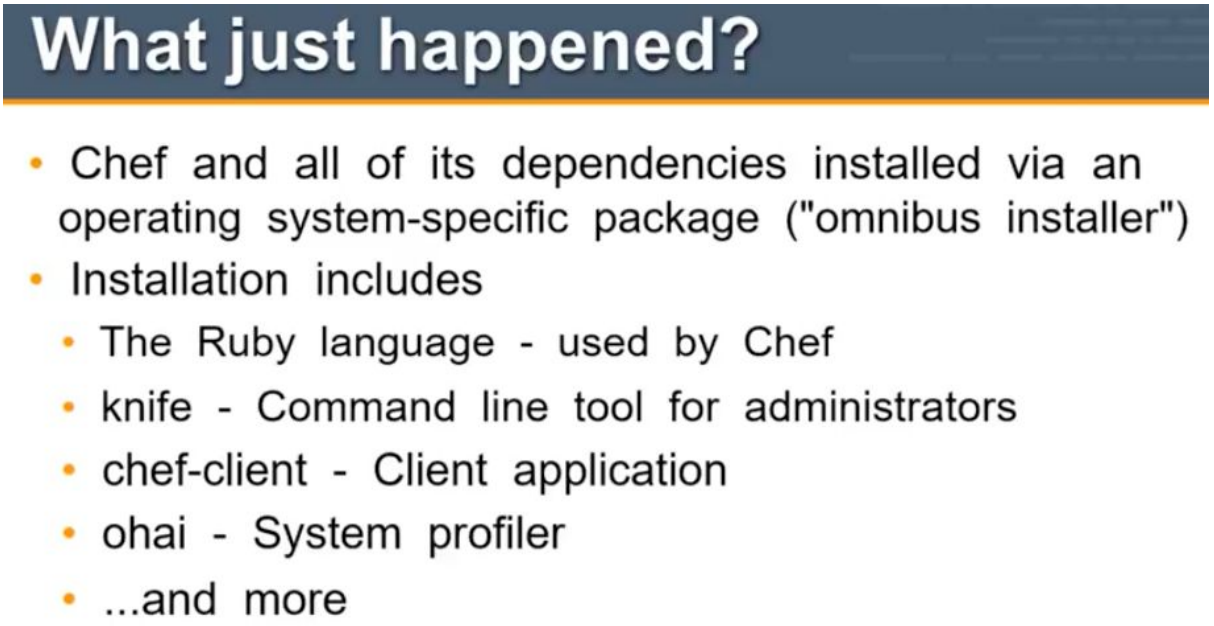
Show the web interface of chef-server, there are no nodes as of now.

knife bootstrap --help {shows options}

 # knife bootstrap 192.168.33.92 -x vagrant -P vagrant -N node1 --sudo

This command runs the following

- It runs ohai process and collects the node info
- Uploads the info to chef server
- Registers the node with chef server
- Runs any cookbooks/recipes if defined



Show the web interface of chef-server, there is node1.

knife node show node1{ Get the node details }

Select the **node1** and select **Attributes,** it will show all the info about our node.
So how did it it get that info it runs a process called **ohai** in nodes.

 # login to node1
 # ohai | more { u can see same info as attributes }

Now if you login to chef server, in Nodes section you can see the machine which is configured.
It indicates the node is successfully bootstrapped with chef server.

Now there is no connection directly with the node, we will never login to node directly, until and unless you want to troubleshoot something.

We completed installation part now all the three machines are ready with us, we configured them and made communication between them, now we need to write **recipes and cookbooks**.

Let's write first recipe
==========
[Recipes are written in Workstation under cookbooks dir]

 # chef generate cookbook <cook-book>

Everything will do it on workstation.

1. **Create cookbook on workstation**
2. **Then will upload this cookbook on server**
3. **Then will associate the cookbook with node**
4. **Then will execute that cookbook on that particular node**

 # chef generate cookbook file-test [makes sure under cookbooks dir]

 # sudo vim cookbooks/file-test/metadata.rb (change Maintainer etc)

Let's start creating our resources :

 # vim cookbooks/file-test/recipes/default.rb

So here we will be defining resources in ruby language, now it's not necessary that you should be master in ruby lang to understand this, coz structure is very easy to understand.

default.rb

========

```
file '/tmp/sample.txt' do
        content 'This file is created with CHEF'
        owner 'root'
        group 'root'
        mode '644'
        #action :create
end
```

Every resource has got a default action, the action here is to create.

 # knife cookbook test file-test

Now we created the cookbook but we need to upload this cookbook to server.

 # knife cookbook list

{when i execute this command, it fetches list of cookbooks from server}

Now if we check the same in web interface [Policy are cookbooks], i don't have file-test cookbook, so will upload the cookbook.

 # knife cookbook upload file-test

Uploaded successfully, let's confirm it.

 # knife cookbook list

Now let's check the same in web interface

Now we need to associate this cookbook with a node.

First let's see what nodes are available

 # knife node list

More info about node

 # knife node show <node-name>

 # knife node show node1

Our focus will be on run list:

 # knife node show node1

There is nothing in **Run List**

Goto web interface → Select node1 → Click on Edit run list { nothing }, let's add one

Adding node to run list

knife node run_list add <node-name> <cookbook-name>

 # knife node run_list add node1 file-test

Again i'll go with show, previously we had empty run list

 # knife node show node1 { we can see file-test in run-list }

Now let's move to node machine

Run the command **# sudo chef-client**

 # sudo chef-client

1. Runs the ohai process
2. It will upload the latest host info to server
3. It will get the list of runlist and its dependencies
4. Downloads the cookbooks as mentioned in run_list
5. Execute the run_list

RESULT : **you will get resource with desired state**

If we run the command for the second time # sudo chef-client

Nothing is updated coz the resource is already in its desired state.

Let's do one thing, forcefully damage the system (someone made changes to sample file content)

 # vi /tmp/sample {made changes which corrupted the system}

Now let's run the **# sudo chef-client** again

Now you can see that the file was not in desired state, bring it back to desired state.

Now if you open and see the file content # vim /tmp/sample.txt it's back to original all the old tampered data is gone away.

That's the desired state, if you define the desired state, it will make sure that it has the desired state.

Let's tamper the permissions **# chmod 777 sample.txt**

Now if i run **# sudo chef-client** then the file permissions will be back to normal desired state.

Now that's what chef is trying to do, whatever you define, it will try to control and make sure it stays that way, but if it's already there in desired state then it will not make any changes and this is called IDEMPOTENCY.

That means if the state is already achieved then it will not disturb the state. Now imagine that you have 10000 systems, so you don't have to worry about any of the system. You will write the recipe and will execute them that's it.

Now let's say you want to edit some content in sample.txt then you will be doing that on recipe, then it will automatically reflect on your node machines.

This is the basic resource we worked on, but in the upcoming session we would see some more recipes to work.

will install web server that is httpd, and will make sure that httpd package is up and running then will create one index.html and will see everything is working fine as expected. Let's do it one by one.

So we will create cookbook in **workstation (under cookbooks directory)**

 # chef generate cookbook webtest

 # vi webtest/metadata.rb (change maintainer and maintainer email)

Now lets update the recipes

 # vi webtest/recipes/default.rb

On the command line we generally do

```
# yum -y install httpd
# systemctl start httpd
# systemctl enable httpd
# echo "WELCOME" > /var/www/html/index.html
```

Now let's login to node machine and disable firewall and set selinux to permissive

Now we need to install the package, here in chef we have a **resource** with name package:

```
# package 'httpd' do
        action :install  {by default}
# end
```

By default the action is install, but we want more information like what we can do more with package, so chef has excellent documentation, lets go and check the document and see what more we can do with the package.

 # docs.chef.io/resource_package.html

We can install the **package** like this,

```
package 'httpd' do
        action :install
end
```

Or simply you can say

```
package 'httpd'
```

coz the default action is install

Now if you want to remove the package you can figure it out from document (actions)

```
package 'httpd' do
        action :remove
end
```

Now i need to start the **service**, so i will use the second resource module,
Now let's go to document section again and see resource_service
https://docs.chef.io/resources.html { **select service**}

```
service 'httpd' do
        action  [:start, :enable] now i can specify two actions also
end
```

Or

```
service 'httpd' do
        action  :start
```

```
end

service 'httpd' do
        action  :enable
end
```

Now the third thing is we need to go with **index.html,** so i'll use the same resource **file**

```
file '/var/www/html/index.html' do
        content "<h1>Welcome to APACHE - By CHEF</h1>"
        owner 'root'
        group 'root'
        mode '644'
end
```

Now need to save the changes then test it and upload it.

 # knife cookbook test webtest { good no syntax error }

 # knife cookbook upload webtest { we can check this in web interface }

knife node show node-name {show what's already attached to this node}

 # knife node show node1

knife node run_list add <node-name> <cookbook-name>

 # knife node run_list add node1 webtest

Now if i say show node then i should see two recipes

 # knife node show node1

Let's go and confirm it on server side as well

go to **Nodes** section → Select Node → Click Settings (gear) → Edit run list

We can see two run_lists

Now we have done the association, now our task is to execute the chef-client on the node, that will install httpd service and starts the service and it will set index page.

Login back to node machine

 # sudo chef-client