JENKINS

* It is an continuous integration tool. It is used to test and build software projects continuously making it easier for developers to integrate changes to the project and making it easier for users to obtain a fresh build.
* It also allows you to continuously deliver software projects by providing powerful ways to define your build pipelines and integrating with large number of testing and deployment techniques.

**INSTALL JENKINS**

* To install Jenkins, you have download **Jenkins.war** file from the website.
* You can deploy the Jenkins.war file from cmd line (or) you can deploy it from the tomcat manager console.
* If you are deploying from console. You should have the .war file in the localhost.

**Go to manager page.**

**Mention context path (name)**

**Mention War file path from the local machine.**

**Click deploy.**

* It will be deployed automatically within few minutes.
* If you want to deploy it from cmd line, copy the war file to **webapps** dir.
* After deploying, click on the context path (name), it will take you to jenkins login page.
* It will ask you for password. By default, password will be stored in **.jenkins/secrets/initialAdminPassword** file.
* Copy the password and create an user jenkins jenkins. Now, we have succesfully installed jenkins.
* You can change the default admin password in jenkins dashboard.
* In the Right pane,

**Click manage Jenkins.**

**Click manage users,**

**Select your admin user.**

**Click GEAR icon.**

**Change the admin password.**

* Now, you can login to jenkins dashboard with the new admin password.
* All the configurations and builds of jenkins are stored in **/root/.jenkins/workspace** dir.
* **/root/.jenkins/config.xml** – it will store all the configuration of jenkins.
* To change Jenkins **session timeout**,

**Go to /root/.jenkins/war/war/WEB-INF/web.xml.**

**Add time(min) in session timeout field.**

**INTEGRATE-GIT**

* We use GIT in jenkins to pull code from git repository for build and test purpose. For this we have to integrate git with Jenkins and git should be installed in your Jenkins master server.
* To integrate Git,

**Click manage Jenkins.**

**Click manage plugins.**

**Install GIT-plugin from avaliable list.**

**Add git home dir in global tool configuration.**

**Click, save.**

* To check, whether plugin is installed (or) not,

**Go to projects,**

**Create new free style project,**

* In the source management section,

**If you see GIT option, means plugin is installed succesfully.**

**If you don’t see GIT, means plugin is not installed succesfully.**

* After integrated git with jenkins, you have to add your git repository while you creating a project. And you have to specify your branch.
* After adding your git repo, whenever you build a project, jenkins pulls the code from new commits from that repo and built it.
* There is an option in jenkins called **“POLL SCM”**, where you can set time(**CRON** **Expression**) for when jenkins should pull the code from git repository for the new commits to build.
* It will check the git repo for a specific period of time like we mentioned in poll scm, if there are any new commits it will pull that code and build the job.

**INTEGRATE-MAVEN**

* Maven is used to generate war and jar files. It means it will build the artifacts from the source code.
* We can use **MAVEN** to build in Jenkins (or) we can use **ANT**, which is another build tool.
* To integrate maven with Jenkins, first we have to install maven in server where Jenkins master resides and set home path for maven.
* To integrate Maven,

**Click manage Jenkins.**

**Click manage plugins**

**Install maven-integration-plugin from avaliable list.**

**Add maven tool and home dir in global tool configuration.**

**Click, save.**

* To check, whether plugin is installed (or) not,

**Go to projects,**

**You will see a new project type called “MAVEN PROJECT”**

* If maven-project job is there you have succesfully installed maven. If not, the installation failed.
* While creating a maven-project, you have to specify the git repo and pom.xml file location. Only after this, it will start building.
* After every build jenkins creates a workspace in **/root/.jenkins/workspace** with project title and all the build files will be avaliable there.
* Whatever the process to build, we can see those steps in console output inside the project.

**JOBS**

* There are different types of jobs in Jenkins.

**Freestyle project** = Used to create **ANT** based builds and normal deployment jobs.

**Maven project** = used to create **MAVEN** based builds.

**External job** = Used to build job in an remote machine, outside of Jenkins.

**Multi-configuration job** = Used to run same build in different configurations and envs.

* All the build history of a project shown inside their respective jobs dashboard.
* To create a freestyle project.

**Click new item.**

**Type a project name.**

**Select freestyle peoject.**

**Mention all settings according to your build deployment.**

**Click save.**

* Inside of a job dashboard,there are several options to discuss**.**
* There is an option called **discard old builds**. It means that it will rotate the old builds based on the specification we mentioned.
* If you enable this option, it will ask you

**Days to keep builds** = how many days of builds to keep (no).

**No of builds to keep** = How many builds to keep (no).

* You can disable a specific job by checking the **disable build** option.
* **Quiet period** = Jenkins starts build whenever a new commit occurs in SCM. But, if we set quiet period(sec), it will wait for specific time even a commit happened in SCM, only after quiet period is over, it will start the build. If a build happened in between the quite period, it will reset the quiet period time and it will start count down from the beginning.
* **Retry Count =** If we set Poll SCM, it will check the url for specific time for new commits and again it will check url only after the specified time mentioned in poll SCM. But, if we set retry count, it will check again immediately for specific no of times as we specified after regular checking is completed.
* **Custom Workspace** = Builds will save in default workspace, but if you want to save in different workspace, mentioned the workspace name here.
* **Trigger build remotely** = You can trigger the build remotely by using an url provided in Jenkins. You have to check this option before building remotely.

**JENKINS\_URL/job/job-name/build?token=TOKEN\_NAME**

* Token name is what we give in **AUTHENTICATION TOKEN** in master Jenkins.
* After building job remotely, in the console output it will show remote ip where the job is executed.
* **Build after other projects are build** = If we check this option and entered a job name, this job will run only after that specified job is completed like **downstream** **jobs**.

**Trigger only if build is stable =** This job will run if specified job is stable**.**

**Trigger even if build is unstable** =This job will run even if specified job will is unstable**.**

**Trigger even if build is failed** =This job will run even if specified job is fails.

* If you open the parent job, you can see child job as downstream job.
* If you open child job, you will see parent job as upstream job.
* If you run parent job, child job will run automatically after parent job is completed.
* Based on the settings you selected, child job will run after parent job resides in a specific state (stable, unstable, failed).

**DASHBOARD**

* In the jobs dashboard, you will see all types of jobs created by all users.
* To separate and view jobs based on their type. We have to create another tab (or) view in jobs dashboard like java-build etc.
* Go to jobs dashboard,

**Click + icon,**

**Type tab name,**

**Select view type (list view, my view).**

**List view** = shows items in simpler format.

**My view** = shows all jobs that current user have access to.

* After creating, go to that tab and click create new job, it automatically saves the job in this tab.
* To add existing job to this tab, click **add existing job** and select a job to add to this tab.
* To delete the tab, on the left pane, click **delete this view** to delete a tab.
* You can use **NESTED VIEW PLUGIN**, to create new view like folders and subfolders inside it, you can separate jobs based on the application.
* You have to download the nested view plugin and while creating views, select nested view and type a name to this new view.
* You can create subfolders in this view to separate the jobs. It will show separately inside the tab.

**BUILD-PIPELINE VIEW**

* In Jenkins, we can build jobs queue by triggering settings in parent and child jobs. It will build one after the other.
* With this pipeline plugin, you can see the process of running project one after the other. It will show you all projects in pipeline view.
* First you have to install **BUILD-PIPELINE PLUGIN**.

**In jobs dashboard.**

**Click + icon.**

**Select pipeline view.**

**Type a name to the view.**

* In the pipeline settings,

**Select initial job** = from which job the pipeline will start. It will show you list of jobs present in your Jenkins master.

**No of builds to display** = Select number.

**Refresh frequency** = time in sec to refresh.

* After creating pipeline view, you can see the builds over pipeline view. Click run to start a job, you can see the all the jobs there running sequentially as we created.
* You can see your pipeline view in jobs dashboard by clicking **my views**. After clicking, you will see all your views (pipeline, list etc).

**USER ADMINISTRATION**

* In jenkins, you can create users and restrict users to specific jobs and specific permissions.
* There are several methods for creating users and groups in jenkins.
* **Jenkins-own-db** – You can create only users with this option, it saved in context.xml file.
* **LDAP** = You can use LDAP authentication for jenkins. You can login with your ldap username and password.
* **Unix user and password** = You can login with unix system users and groups where the jenkins master is installed.
* There are several types of permission sets for the users and groups.
* **Anyone can do anything** = means every one will have admin access, if you check this permission.
* **Logged-in-users can do anything** = logged-in-users have admin access for everything.
* **Matrix based security** = To give specific permissions to specific users.
* **Project-based matrix security =** To give specific permissions to specific users for specific projects. After enabling this security, you will get an option in the job dashboard called project based security.
* By this option, you can select which users should have what type of permissions for that specific job.
* **Role-based authorisation strategy** = With this option, you can create a role with specific permissions and you can attach this role users/groups for security concerns.
* All the users/groups who have got this role will get permissions what will role have and you can attach this role to a project. So, whatever the permissions the role have and whoever the users/groups attached to that role will get those permissions to that project.
* You have to install **“ROLE BASED AUTHORIZATION STRATEGY”** plugin.
* After installing that plugin, you will get a new option in manage jenkins dashboard like manage and assign roles.
* You have to create a role with specific permissions and add users/groups to that role to get those permissions.
* Go to manage and assign roles,
* Click Manage roles,

**Create a role.**

**Give specific permissions to that role.**

**Click, Save.**

* Click Assign roles,

**Add an user/group.**

**Select a role to that user/group.**

**Click, Save.**

**NODES**

* We can configure slaves for jenkins master..means that we can build and test code in different environments (test-server), if it succeed we can deploy it in production. For this type of builds we configure slaves in jenkins.
* In **slave**, we have to install **java, maven** and **git** before building the projects. You can add as many slaves as you want.
* To configure slave.

**Go to manage nodes.**

**Click new node.**

**Type a name to node.**

**Select permanent agent** (or) **click copy from existing node** (if you already have a node).

**Executors** = the no of builds thet jenkins may perform in this slave.

**Description**, **remote dir**(where to save all the builds in slave).

**Labels** (to identify slave),

**Usage** = how to use this slave [as much as possible (or) only when build happens].

**Launch** **method** = How to connect to slave (ssh, cmd from master).

If you select SSh, you have to give host ip, host user and password to open ssh connection from master to slave.

Before configuring slave, create an ssh connection between server and client to get rid of ssh errors while connecting for the first time,

**Host key verify** = Controls how jenkins verifies ssh key presented by remote host.

**Known host file verify** = It verifies known\_hosts file for the user, where jenkins is executed under (select this option if you are making an ssh connection with node).

**Avaliability** = keep this host up all time (or) according to schedule (or) up when builds occur and die when no builds occurs.

**Tool locations** = select java and build tools and specify home path for those tools. Click Save.

* Click , launch agent to start configuring the agent. Now, slave will be up and running in few minutes.
* Now, you can launch build projects in slave. It will create workspaces in remote directory where you specified while creating node.
* While creating project. Click on,

**Where this project can be run** (to run this project on slave).

**Type label of your slave.**

**Select all other settings like git, build tools same as normal project.**

**Click save.**

* Now, this build runs in slave and you can view it in remote directory once the build is completed.

**CONFIGURE SMTP IN JENKINS**

* We can configure smtp server in jenkins to get notified when a build action occurs.
* You have to download **Email-ext-plugin** and configure your email server in jenkins.
* Go to Manage jenkins,

**Click Configure system.**

* In extended email configuration,

**Type your smtp server.**

**Click advanced option.**

**Select use smtp authentication.**

**Type your mail, password, smtp port, ssl.**

* In email notification,

**Type your smtp server.**

**Click advanced option,**

**Select use smtp authentication.**

**Type your mail, password, smtp port, ssl.**

* Test the connection by clicking on test button and type your email id. If you get mail, you have configured smtp succesfully.
* While creating the project, at the bottom of the configure page, click **email notification** and type email ID’S to get notified whenever a build failure occurs.
* You can type any number of email in jenkins. It will send notifications to all mails.
* But, this option will send mails only when build failure occurs. It won’t send mails for successful builds.
* To send mail when build is success, there is another option.
* In post build action,

**Click** **editable email notification**.

**Project reciepient list** = type your emails separated with “,” .

**Content type** = select how mail should look (html, plain text etc).

**Default subject** = write down the subject as you like.

* Click advanced settings,

**Click add trigger.**

**Select trigger type (fail, success etc),**

**Select reciepents list (to get emails)**

**Click save.**

* You can add as many triggers as you want.
* Now, it will send mails regarding build success, fails, abort etc to all the reciepents as we mentioned.

**DEPLOY BUILDS TO CONTAINER**

* After the build is completed, the next big thing in the project would be deploy build files to conatainers (tomcat, jboss etc).
* Jenkins will deploy build files to container automatically, after building completes. We have to download **deploy-to-container-plugin** in jenkins.
* You have to create a new user and a new role in tomcat to deploy builds automatically to container through Jenkins and the role is **“manager-script”.**
* Go to **tomcatusers.xml** file**,** add a line in users section.

**<user username=”deploy” password=”deploy” roles=”manager-script”/>**

**While Creating project.**

* In post build action,

**Select Deploy war/ear to container.**

**Type your build files path.**

**Add Credentials (which we created with manager-script role).**

**Add your container Path (url).**

**Add context path.**

**Click, save.**

* It will automatically build and deploy it to your container.
* Username and password for manager-script role can be anything. As for the test purpose I gave “deploy” for both username and password.
* Above is one method to deploy articrafts to container. But, generally In productions everyone uses separate jobs for both building and deploying purposes.
* For this type of deployments, You have to download the **CLONE WORKSPACE SCM PLUGIN.**
* This plugin will clone the build workspace dir to other jobs as we specified. By that cloned workspace, Jenkins will deploy the builds to containers.
* After installing clone plugin.

**Go to build(parent) job,**

* In post build action,

**select archieve for clone workspace SCM,**

**Files to include in workspace** = which files should be include in workspace(\*\*/\*.war).

**Criteria for build to be archieved** = Select most recent successful build to deploy (or) other options to deploy.

**Archieve method** =tar (or) zip**.**

**Click Save.**

* Now, create a freestyle job for deployment.
* In deployment job, you should select which build job should be cloned to deploy to container.
* Go to deployment job,

**In scm, select clone workspace**.

**parent project =** which project to be deployed**.**

**Crieteria for parent build =** Select most recent successful build (or) other options**.**

**Select deploy to container.**

**Select all other settings of container to deploy.**

**Click save.**

* After configuring these settings, it will deploy the builds to container successfully. But, we have to run the build job and deployment job manually which will take lot of time to deploy. We can automate this process by enabling an option in parent (build) job.
* Go to build (parent) job,

**In post build action,**

**select build other projects.**

**Select deployment project to build after this project automatically.**

**Select options to trigger build if parent job is stable (or) even it is unstable.**

**Click, Save.**

* After first job completion, in console output, you can see triggering new job line along with your child job.
* Whenever you build, Jenkins will deploy latest successful build to deploy to container.

**BUILD METRICS**

* We can monitor all the builds in jenkins and about their states (success & failure).
* We have to download **BUILD-METRICS-PLUGIN**, and can see all the builds status with that plugin.
* After installing plugin.
* Go to manage Jenkins.
* you can see build metrics, specify the values like

**Show builds for the last** = select how days, months, years build to show.

* Select filtering,

**Node filtering, job filtering, job filtering, cause filtering** .

* By specifying these, you can see the status of all your builds and the states of your builds like how many build build failures occurred and how many successful builds etc.

AMAZON EC2 PLUGIN

* We can use amazon ec2 instances with Jenkins as slaves with **AMAZON EC2 PLUGIN**.
* It will work just as other local slaves in Jenkins. The advantage with ec2 is Jenkins will create ec2 instances when we start a job and terminates the instances after job completion (or) **idle time out** finishes, it will be an money saving idea for us.
* Jenkins will create instances from an AMI as we specified in Jenkins. So, whenever a new instance is started as slave, it is created by AMI.
* By default, some instance types are not supported by Jenkins. So, Check whether the instance type is supported (or) not before creating cloud in jenkins.

Prequesties:

**IAM User**

**IAM User credentials (access and secret key)**

**Specific instance type**

**AMI**

**Security Group**

**Remote user and remote dir (to store builds)**

**Region**

**Set label, usage and idle timeout.**

* First create an IAM user and give him administrator access. Download the credentials of the IAM user.
* Launch an instance and install all the applications that you want to build (java is mandatory) and create an AMI from that instance.
* Go to Jenkins, Install **AMAZON EC2 PLUGIN**.

**Go to configure system,**

**At the end click add new cloud,**

**Give your aws iam user access and secret access keys.**

**Type your AMI ID.**

**Paste the pem file** and test the connection, if the connection shows success, you have configured correctly.

**Type the security group name that you have used for AMI.**

**Select instance type** (not all types are supported with Jenkins)

**Select region and AZ.**

**Mention Jenkins home path where you want to save the builds.**

**Mention user to connect to ec2 from Jenkins** (with this user it will talk to ec2).

**Set label and set usage.**

**Set Idle timeout** (how much time instance should run. After this timeout completed, instance will be terminated even if there is a job running).

**Select no of executers.**

**Remote ssh port (22).**

**Instance cap** (how many instances can launch from an AMI. If you left this option blank, it will take as infinite).

* After selecting all these options.

**Go to manage nodes.**

**Select your slave.**

**Click configure.**

**Set tools home path as installed in your ec2 instance AMI.**

* while building a job, select this slave to run the job which is identified by a label and it will create an instance and build the job in it.
* After the idle timeout is completed, instance will terminate automatically.
* You can create multiple slaves with multiple AMI’s. Jenkins will bring up nodes according to your settings.
* You have to add another cloud, specify another AMI and mention new label and save.
* While creating job, select ec2 label. Then, it would run on the specific slave as we selected.

**SONARQUBE**

* It is a open source platform to maintain source code quality. It is written in java. But, can analyze 20 different programming languages.
* Default port = **9000**.
* It analyzes architecture & design, unit tests, potential bugs, coding rules etc.
* You can access sonarqube from web browser with your server ip and sonarqube port.
* You should always use sonar as normal user. Root user can’t sonar.
* One of the main issues with sonar is. Your server should have atleast **4** gm ram. If you have less than 4 gb ram, sonar stops automatically.

**INSTALL SONAR**

* Before we install sonarqube, we have to install java and set home path for java.
* After installing java. Download sonarqube zip file from the official site and unzip the file.
* Sonarqube needs a database to store the metrics of analysis...It supports all types of DB available in the market.
* If you want to use default DB provided by sonar, leave all the other settings as default.
* If you want to use non-default database. Create a db called sonar and create user sonar with a password in DB which is installed in your system (mysql in my case).

**CREATE DATABASE sonar CHARACTER SET utf8 COLLATE utf8\_general\_ci;**

**CREATE USER 'sonar' IDENTIFIED BY 'Sonar@1234';**

**GRANT ALL ON sonar.\* TO 'sonar'@'%' IDENTIFIED BY 'Sonar@1234';**

**GRANT ALL ON sonar.\* TO 'sonar'@'localhost' IDENTIFIED BY 'Sonar@1234';**

**FLUSH PRIVILEGES;**

* Go to **/sonarqube/conf/sonar.properties**,
* In the database section,

**Uncomment the jdbc.username and password lines.**

**Add your db username and password.**

**Uncomment jdbc.db.url line, for which db you are using. save the file.**

* Now, all the sonar metrics will be saved in the database that you selected.
* Start the service to test whether sonar is working (or) not…

**/sonarqube/bin/linux-64bit/sonar sh start –** To start sonar.

**/sonarqube/bin/linux-64bit/sonar.sh stop –** To stop sonar.

**/sonarqube/bin/linux-64bit/sonar.sh restart –** To restart sonar.

**/sonarqube/bin/linux-64bit/sonar.sh console –** To see the output.

* Based on the architecture of your server, the start and stop scripts path will change.
* If you have 64bit system, scripts will be stored in linux-64bit dir.
* If you have 32bit system, scripts will be stored in linux-32bit dir.
* Go to browser, type your ip and sonar port(9000), you can see sonar home page.
* If you want to use your ip (or) hostname instead of localhost in the sonarqube url,
* Go to sonar.properties file,

**In web server section.**

**Sonar.web.host = type your server ip (or) hostname.**

* Restart the sonarqube and now you can access sonar with your ip and sonar port in browser.
* By default, username and password to login to sonar dashboard is **admin** for both.
* You can change the password once you logged into sonar dashboard.

**SONAR, JENKINS, MAVEN INTEGRATION**

* You can integrate sonarqube with Jenkins for continuous testing. It will test the code quality right after the build job is finished. It will show errors in the code in sonar GUI.
* First, you have to download sonar-scanner plugin (sonarqube in old versions).
* After installing. Go to Global tool configuration,

**In sonar installations section.**

**Click add sonarscanner.**

**Type a name.**

**Give Sonarqube home path where you installed in your server.**

* Go to configure system,

**In sonarqube servers section.**

**Type name.**

**Sonar server url.**

**Sonar version.**

**Sonar account username and password** if it asks (by default, **admin**).

* We have configured everything with sonar and Jenkins. Now, we have to configure sonar with maven.
* To use sonarqube with maven, you have to install a plugin by making an entry in maven **settings.xml** file.
* Go to settings.xml,
* Under **pluginsGroups** section.
* Below ‘🡪’ mark, remove the **</pluginGroups>** line and paste this.

**<pluginGroup>org.sonarsource.scanner.maven</pluginGroup>**

**</pluginGroups>**

* In **profiles** section,
* Below ‘**🡪’** mark. Remove **</profiles>** line and paste this

**<profile>**

**<id>sonar</id>**

**<activation>**

**<activeByDefault>true</activeByDefault>**

**</activation>**

**<properties>**

**<!-- Optional URL to server. Default value is http://localhost:9000 -->**

**<sonar.host.url>**

**http://192.168.10.32:9000**

**</sonar.host.url>**

**</properties>**

**</profile>**

**</profiles>**

* It will download the plugin from given website above in plugingroup section and it will detect the sonar from the given url.
* After copying this code in settings.xml file, run any maven cmds to see whether we did correct (or) not.
* To test sonar in linux, go to your project pom.xml dir and run **mvn sonar:sonar**….It will test the code and give you the errors in the code in sonar GUI dashboard along with your project name.
* Login to your sonar GUI dashboard with your username and password, there you can see your project name. Inside the project name you can see all your code along with errors if there are any.
* After configuring sonar in Jenkins, while creating a project,
* In build Environment,

Select **Prepare sonarqube scanner environment**.

* In post build actions,

Select **sonar analsys with maven** and save the project.

* Now, after build is completed, it starts sonar analsys and push the code errors and bugs to sonar dashboard.

ANSIBLE

* Ansible is an configuration management and provisioning tool. It is created using python.
* It is an automation tool, which automates all the actions in multiple servers at a time by just writing a script and pushing it to all the server from central server.
* For ex: if you have 100 servers and you want to install an application in all 100 servers. You don't need to login to all servers and do that work, just create an yaml script, run the script in the ansible server and it will do the remaining in all 100 servers.
* Python is mandatory for ansible min **v2.6**.
* To install ansible, first we have to install **epel-release**.
* To Install the ansible….**yum install ansible**…
* To check whether ansible is installed (or) not, type…..**ansible --version**.
* Main configuration file = **/etc/ansible/ansible.cfg**.

**FEAUTURES**

* **Agent less** = No need of creating an agent in client machines like in other CM tools chef, puppet. You just have install ansible in server and we are good to go.
* **SSH** = It uses ssh connections. You don't need to install any extra software to connect to client machines. So, Make sure ssh is working properly in all machines.
* **PUSH** = It uses push based architecture for sending configurations. Write the code in yaml script and execute the script. Ansible will push and executes it in all nodes.

**HOST INVENTORY**

* It contains list of your hosts and host groups.
* It will have all your host ip's. we can also create groups for the specific hosts like web servers are one group, db servers are one group.
* Befor adding hosts to inventory, make sure you have made an successful ssh connection with all the nodes to maintain without intereption.
* If you get an error like this while making ssh connection with node.

**“1 key(s) remain to be installed -- if you are prompted now it is to install the new keys”**

* Just set “**password** **authentication yes”** in /**etc/ssh/sshd\_config** file in **node** machine and restart the node ssh server.
* To add hosts, just write down the ip of hosts.
* To create a group in inventory, write grpname in square bracets and write the hosts ip's down the groupname.

**[host group name]**

**Host ip 1**

**Host ip 2**

**Host ip 3**

* So, whenever you want to make configuration changes just for db server, mention the dbserver grpname while executing the playbook and it will change only that specific grp.
* default location of inventory = **/etc/ansible/hosts**.
* To see how many hosts we configured in inventory = **ansible all –list-hosts**.
* After adding host inventory try to ping whether we configured correctly (or) not = **ansible test –m ping** (test=grp name).
* You can use different paths for inventory. But while executing playbooks you have to mention that path with **–i** option.

**ansible-playbook –I inventory-path file.yml**

**DYNAMIC INVENTORY**

* Normally, we use default ansible static host inventory **(/etc/ansible/hosts)** for nodes.
* If the nodes have static ip, there is no problem. But, if the nodes have dynamic ip, the ip will change after every reboot. Updating the hosts file with new ip’s after every reboot will be a headache.
* For this we use dynamic inventory, We have to write python scripts for dynamic inventory and the scripts will keep the hosts updated.
* In aws, we don’t need to write any python scripts for dynamic inventory. By default, ansible provides you scripts for dynamic inventory in aws.
* **ec2.py** = It is a script written in boto ec2 library. It will query your aws account for running ec2 instances.
* **ec2.ini** = It is the configuration for the ec2.py and it can be used to limit the reach of ansible. You can specify tags, regions etc.
* You can download these scripts from the internet and copy those files to **/etc/ansible**.
* To work with these scripts, you should have installed ansible with **pip**.
* You have to install boto (python interface with aws) to work with ansible and aws.
* Install **python** and **boto** (python interface with aws).

**pip install boto.**

* To link boto with ansible we have to set this in inventory file.
* Create a file called **~/.boto** and configure your credentials like this

**[credentials]**

**aws\_access\_key\_id = AKIAJNLYRAN2L2LFKSXA**

**aws\_secret\_access\_key = HbVeN6YZsuMeVpTDw0VifPLh75jCuEmC7CH0orx1**

* Go to /etc/ansible/

**./ec2.py** **--list** = It will show all your running instances in aws.

**ansible instance-id –m ping** = Ping instances with instance id.

**ansible tag\_Name\_ec2tag –m ping** = To ping instance with tag name.

* Mention the ec2.py with playbook command while executing any playbook.

**ansible-playbook –i ec2.py file.yml.**

* In the playbook, you can specify hosts as **all** (or) you can specify **instance** **tags**.

**MODULES**

* Modules (tasks or library plug-ins) are the ones which actually executed inside the playbooks.
* A playbook contains play, a play contains tasks, a task contains modules and whenever you run the playbook the modules will get executed.

**Ex : apt module, yum module, service module, copy module, fetch module** etc**.**

* If **SElinux** is enabled in your nodes, you must install **libselinux-python** to use file/copy/template related functions in ansible.

**AD-HOC CMDS**

* These are simple one-line cmds in order to perform a quick task. If you want to do a task and you don't want to write a playbook, you can use these ad-hoc commands.
* You have to use the ansible commands with modules which are called ad-hoc commands.

Ex : **ansible all -s -m shell –a “uptime”**  = it will show the uptime of all hosts.

**ansible all –s –m shell –a “tail –f /var/log/access.log”** = To see logs.

**s =** run cmd using sudo**.**

**a =** pass arguments to module**.**

**C =** Check**.**

**m =** modulename**.**

* There are so many ad-hoc commands like these for daily purposes without writing playbooks.
* **ansible all –m setup** = To see nodes info.

**COPY**

* **ansible all –m copy –a “src=/path dest=/path-to-store”** = To copy file from server to hosts.

**FETCH**

* **ansible all –s –m fetch –a “src=/path dest=/path”** = To copy file from hosts to server.

**FILE**

* **ansible all –m file –a “dest=/file-path mode=0644 state=touch”** = To create an empty file.
* **ansible all –m file –a “dest=path file mode=0666 state=directory”** = To create dir.
* **ansible all –m file –a “dest=/file-path state=absent”** = To delete a file (or) dir.
* **ansible all –m file –a “src=/path dest=/path user=root group=root state=link”** = To create symlink.
* **ansible all –s –m file –a “dest=/path mode=0666 user=name group=name” =** To change permission.

**CRON**

* **ansible all –m cron –a “name=’/cron-name’ hour=4 job=/script-2-exe”** = To execute cron job.
* **ansible all –m cron –a “name=’/cron-name’ state=absent”** = To remove cron job**.**

**YUM**

* **ansible all –m yum –a “name=pkg state=installed”** = To install packages.
* **ansible all –m yum –a “name=pkg state=latest”** = To install a pkg. If the pkg is already installed, it will update the pkg to latest version.
* **ansible all –m yum –a “name=pkg-name state=absent”** = To remove a pkg.
* **ansible all –m yum –a “name=’\*’ state=latest” =** To update all pkgs.
* **ansible all –m yum –a “name=url state=present”** = To install pkg from an url.

**SERVICE**

* **ansible all -m -s service –a “name=httpd state=started”** = To start a service.
* **ansible all -m -s service –a “name=httpd state=restarted”** = To restart the service.
* **ansible all -m -s service –a “name=httpd state=stopped”** = To stop the service.

Service modules **= Started, stopped, restarted, reloaded.**

**USER**

* **ansible all –m user “name=user-name password=password group=grp-name”** = To create an user in all hosts. You have to enter crypted password.
* **mkpasswd –method=sha-512 =** To generate crypted password**.**
* **ansible all –m user –a “name=user-name state=absent”** = To delete an user from all servers.

**PLAYBOOKS**

* Playbooks define your workflow. Playbooks are set of instructions that you send to run on hosts machines. If we want to configure something in nodes, we write tasks in playbooks, it actually gets executed in same order in all nodes as we written in playbooks.
* Playbooks are easy to write. These are written in yaml and should end with **.yml** format.
* Playbooks are divided into three sections.
* 1. **Targets (or) hosts =** servers where this playbooks should be executed.
* 2. **variables** (not mandatory) = You can define your variables.
* 3. **Tasks** = tasks to perform on hosts (or) targets.
* 4. Handlers = these are just like tasks, but only run when called bay another task.

EX:

**---**

**- hosts: test**

**sudo: yes**

**tasks:**

**- name: install ftp**

**yum: name=vsftpd state=installed**

**notify:**

**- start ftp**

**handlers:**

**- name: start ftp**

**service: name=vsftpd state=started**

* save and quit the file. Now, run this playbook.....**ansible-playbook file.yml** ..DONE...
* **ansible-playbook file.yml --syntax-check** = To check syntax errors in playbook.
* **ansible-playbook –v file.yml** = To see output of playbook.
* **Ansible-playbook –v file.yml --step** = It will prompt you options (y,n) whether to perform this action (or) not.
* Every lines we write after hosts line should be under it.
* Handlers are similar to tasks, but run only if we set nofity directive.
* You can create lvm with playbooks with **lvg** and **lvol** modules

**ROLES**

* Roles are like predefined files, where we define all our requirements in roles and use those roles in playbooks.
* We can share & reuse these roles in anywhere. Roles are located at **/etc/ansible/roles** dir.
* You can create roles by using **ansible**-**galaxy** command. It will create a directory structure where you can define all your requirements.
* A role directory structure contains of **defaults**, **files**, **vars**, **handlers**, **meta**, **tests**, **templates** and **tasks**.
* **Defaults** = Conatins variables for the role/application.
* **Files =** Contains regular files where you need to copy to nodes**.**
* **Handlers =** Contains targets for notify directives and almost associated with the services.
* **Meta** = consists of atributes such as about the role, author of the role, platform, dependencies etc.
* **Tasks** = Contains all the actions that should be done, when you are using this role like install package, remove package etc.
* **Templates** = These are similar to files but it supports modification (dynamic files) as they are being provisioned to nodes. Modifications are done through jinja2 templating language.

For ex: configuration files.

* **Vars** = variables stores in the vars have higher priority, which are hard to override, whereas variables stores in defaults have low priority.
* Every directory contains its own file called main.yml. You have to write all the data in this file respective to their directories.
* **Ansible-galaxy init apache**= To create a role for apache. If you want to create a role for nginx, type nginx in the place of apache.
* Go to **/etc/ansible/roles/apache/tasks**.
* Edit the **main**.**yml** file and add these following lines to install apache and copy the index.html file to nodes.

**- name: install apache**

**yum: name=httpd state=installed**

**- name: copy index file**

**copy: src=index.html dest=/var/www/html**

**notify:**

**- start apache**

* Go to **/etc/ansible/roles/apache/handlers**.
* Edit the **main**.**yml** file to give targets for notify directories.

**- name: start apache**

**service: name=httpd state=started**

* The Handlers and notify name should be the same.
* To copy files to node machines, the files should be there at **files** directory.
* Copy the index.html file to **/etc/ansible/roles/apache/files** dir.
* Whatever the files you want to copy to nodes, first you should mention them in tasks dir with copy module and those files should be present in the files dir.
* After completing these requirements, create a playbook for apache to install and copy the files. You just have to mention the hosts and role name in the playbook. It will take all the files from that role.

**---**

**- hosts: all**

**Roles:**

**- role-name**

* To copy dynamic content like configuration files(for ex:httpd.conf), we can use template module.
* In tasks dir. Add the following lines above notify line.

**- name: copy dynamic files**

**Template: src=httpd.conf dest=/etc/httpd/conf/httpd.conf**

* It will copy the data which is not there in destination machine not the entire file.
* Templates are based on **jinja2** templating language.
* You can define multiple no of roles in a single playbook. It will perform all the tasks.

**INSTALL EC2 WITH ANSIBLE**

* You can install ec2 instances with ansible by using ec2 module.
* You have to install **pip**, **boto** and configure your aws credentials in ec2.
* **yum install python-pip** = To install pip.
* You can also install pip with url.
* **wget** [**https://bootstrap.pypa.io/get-pip.py**](https://bootstrap.pypa.io/get-pip.py) **=** To download pip**.**
* **python get-pip.py** = To install pip (from downloaded file).
* **pip install boto** (or) **boto3** = to install boto.
* If boto doesn’t work install boto3 with pip.
* To link boto with ansible we have to set this in inventory file.
* Set inventory file = **localhost ansible\_connection=local ansible\_python\_interpreter=python**
* Create a file called **~/.boto** and configure your credentials like this

**[credentials]**

**aws\_access\_key\_id = AKIAJNLYRAN2L2LFKSXA**

**aws\_secret\_access\_key = HbVeN6YZsuMeVpTDw0VifPLh75jCuEmC7CH0orx1**

* Create an ec2 role and define all the variables and tasks and execute it to launch instances.
* You need ami, security group, subnet ID, keypair, region, instance type and other configurations as you want. It will create ec2 instance with the configurations you specified.
* If you are using a playbook to launch ec2 instances, you have to specify variables in playbook separately. You can’t use raw code to launch ec2 instances.
* Download e2.py and ec2.ini files to communicate and manage the nodes.
* Ex: playbook to create ec2

**---**

**- name: Create a new Demo EC2 instance**

**hosts: localhost**

**gather\_facts: False**

**vars:**

**region: ap-south-1**

**instance\_type: t2.micro**

**ami: ami-e60e5a89**

**keypair: docker**

**tasks:**

**- name: Create an ec2 instance**

**ec2:**

**key\_name: "{{ keypair }}"**

**group: RED # security group name**

**instance\_type: "{{ instance\_type}}"**

**image: "{{ ami }}"**

**wait: true**

**region: "{{ region }}"**

**count: 1**

**count\_tag:**

**Name: Demo**

**instance\_tags:**

**Name: Demo**

**vpc\_subnet\_id: subnet-cd1ec9a5**

**assign\_public\_ip: yes**

**register: ec2**

* The above playbook will create one ec2 instance with the configurations we mentioned in playbook.
* You can also use ansible roles to launch instances. The roles are easy to configure.
* Create an ec2 role with ansible galaxy command and configure the variables and tasks and create an playbook with that role to launch ec2 instances.
* Ansible-galaxy init ec2 = To create an ec2 role.
* Go to vars dir and edit the main.yml file, mention all the variables that you need to launch ec2.

**ec2:**

**region: ap-south-1**

**zone: ap-south-1a**

**keypair: keypairname**

**image: ami-id**

**instance\_type: t2.micro**

**group: security group for ec2**

**vpc\_subnet\_id: subnet to launch instance in.**

**public\_ip: yes**

**count: 1**

**instance\_tags:**

**Name: name for ec2**

**volumes:**

**- device\_name: /dev/xvda**

**volume\_type: standard**

**volume\_size: 10**

* The ec2 at the top of the script is nothing but the ec2 module we are using to launch ec2 instances with ansible.
* You can mention infinite number of variables in the vars dir. After mentioning variables ad them to tasks to execute them.
* Go to tasks dir and edit the main.yml file and mention all these variables in that file.

**- name: launch ec2**

**ec2:**

**region: “{{ ec2.region }}”**

**zone: “{{ ec2.zone }}”**

**image: “{{ ec2.image }}”**

**keypair: “{{ ec2.keypair }}”**

**group: “{{ ec2.sg }}”**

**vpc\_subnet\_id: “{{ ec2.vpc\_subnet\_id }}”**

**assign\_public\_ip: “{{ ec2.public\_ip }}”**

**instance\_tags: “{{ ec2.instance\_tags }}”**

**wait: true**

* Create a playbook and specify this ec2 role to launch ec2 instances.

**---**

**- hosts: localhost**

**connection: local**

**roles:**

**- ec2**

* Once you execute this playbook, it will launch ec2 instances with specified configurations.
* In the above example, we wrote the variables and tasks separately and mentioned the variables in tasks to perform actions.
* The values in the bracets represents the values in vars directory. As here in tasks dir, we mentioned the variables along with role (ec2).
* While executing the playbook, ansible goes to that role we specified in playbook, search for tasks dir to execute the actions and pulls the data from vars dir, if we mentioned any variables.

**VAULT**

* Ansible vault can encrypt anything inside an yaml file, with password of your choice.
* It can be ssh keys, ssl certificates, api tokens etc.
* You can’t see (or) edit the file once it is encrypted with vault. You can’t even run the playbook after encrypting, which gives you an error.
* With the help of **ansible-vault** command, you can encrypt and decrypt playbooks.
* The most common encrypted files are variable files which have sensitive data.

Ex: defaults/main.yml, vars/main.yml

* The encrypted files can be distributed (or) can place in version control. The file will be in encrypted mode only even in version control.
* Whenever you are using an vault command, it will prompt you to give password for **encrypt**, **decrypt**, **create** and **edit** actions.
* **ansible-vault encrypt file.yml** = To encrypt an existing file.
* **ansible-vault decrypt file.yml** = To decrypt an file.
* **ansible-vault create file.yml** = To create a new file with encryption.
* **ansble-vault edit file.yml** = To edit an encrypted file.
* **ansible-playbook file.yml --ask-vault-pass** = To run an encrypted playbook. It will ask you vault password to execute the playbook.
* **ansible-vault rekey file.yml** = To change vault password. It will prompt you to type old password and give a new password to vault.

**GIT**

* It is a distributed version control system which uses widely among other version control systems.
* A version control is a software tool which manage the changes to source code over time.
* It keeps track of every change made and provides proper description about what changes have been made in the specific version and when it was changed.
* If a developer wants to use earlier versions of code, you can use it anytime with the help of version control.
* When you add new files and commit them, it will store in local repo. Only after you push, it will go to remote repo.
* **git init** = to initialize repo in local machine.
* **git config --global user.name “name”** = Set a name for your commit transactions.
* **git config --global user.email “email”** = Set a mail for your commit transactions**.**
* **git clone repo-url** = To clone remote repo in local.
* **git status** = To see which files are not commited.
* **git add file** = To add file to staging area (or) index.
* **git rm file** = To remove a file locally**.**
* Commit and push to origin to del the file in remote repo.
* **git add \*=** To add all files which are in your branch at once.
* **git commit –m “commit msg”** = To commit changes to repo.
* **git commit --amend –m “commit msg” =** To change the commit msg of last commit.
* **git push –u origin master** = To push changes to remote repo master branch.

To push to other branch…type branch name in the place of master.

* **git pull origin master** = To pull changes from master remote repo.

To pull from other branch…type branch name in the place of master.

* **git branch** = To see how many branches are there in repo.
* **git branch branch-name** = To create a branch.

After creating a branch in local, push the branch to create in remote repo.

* **git checkout branch-name** = To go to that specified branch.
* **git branch –d branch-name** = To delete a branch.
* **git merge branch-name =** To merge two branches.
* You should be in the destination branch in where you want to merge the src branch. The files in the src branch will be merged to destination branch.
* **git branch –m old new =** To change branch name.
* **git branch --merged** = To see which branches have merged into current branch.
* **git branch --no-merged** = To see branches which have not merged into current branch.
* **git rebase branch =** apply the commits at the top of the history without creating a merge commit.
* **git rebase --abort** = To abort (or) stop the rebasing.
* **git rebase --continue =** To continue the rebase from the last, after you fixing any conflicts.
* **git diff --staged** = Shows files which are in staging area (uncommitted).
* **git diff branch1 branch2** = shows differences b/w two branches.
* **git log** = To see version history with commit id, commit msg, date and time.
* **git checkout commit-ID file** = To revert back a file to specific commit.

If you have commited **multiple** files in one commit…just give commit ID with checkout command to revert the changes in all the files.

* **STASH =** stash takes your current working dir and puts in a stack for later use and gives you back a clean working dir.

For ex : if you are working in a branch, half of the work is completed and you want to go to another branch and you don’t want to commit this work. So we move this files to stash. It will store files in a stack. If you didn’t stashed (or) commited the files and switched to another branch, the files in the old branch will be deleted.

* **git** **stash** = to save files in a stack.
* **git** **stash** **list** = To see stashed files.
* **git stash drop =** To drop files from stack**.**
* **git stash drop stash@{0}** = To remove a specific file from stashing.
* **SQUASH** = merging multiple commits together as one single commit.
* **git rebase –i HEAD~3 =** To merge last 3 commits**.**
* It will show you the configuration file to choose which commit should retain and which commit should be squashed.
* For ex:

**pick commitID commit-msg**

**Pick commitID commit-msg**

**Pick commitID commit-msg.**

* In the above example, I have 3 commits and I want two commits to be squashed.
* To do that, remove **pick** and type **s** before the 2 commits to which you want to squash and leave **pick** before the commit to which you want to retain and Save the file.
* After saving, it asks you to edit the commit history by opening a file and shows you the commit messages.
* Remove the 2 commit messages from the file that you want to squash and leave the one commit, which you want to retain.
* After squashing, if you push changes to remote branch you will get an error. Because, your tip of the branch is now behind. So, push the changes forcefully with **“–f”**.

**MAVEN**

* Maven is a project management and comprehension tool, which is based on project object model(**POM**). It can manage project build, reporting and documentation from a central piece of information. We can build and manage java projects using maven.
* Maven follow few things called **defaults** and **convention** **over** **configuration**.
* The defaults are
* Your **source** code should be in **src/main/java**.
* Your **test** code should be in **src/test/java**.
* **Pom.xml** should be in the root folder.
* You have to run maven commands from where the pom.xml file is located . When you run maven commands, it first looks for pom.xml file. Based on that file it will build the project. If pom.xml is no there in that dir, build will be failed.
* **Pom.xml** = It is an xml file which contains information about the project and configurations used by maven to build the project. It also contains goals and plugins. While executing, maven looks for pom file in current directory and reads the file and gets needed configuration information and executes the goals as defined in the pom file.
* In pom.xml file, you will see all your configurations like **packaging**(jar,war), **group** **ID**, **artifact** **ID**, **name**, **version**, **plugins**, **dependencies etc.**
* You can add any new **dependencies**, **plugins** in pom file if you want for the project.
* **Build life cycle** :
* Maven is based around the central concept of build lifecycle. It is the sequence of the phases(stages), which define the order to execute the goals.
* The phases are:
* **Validate** = validating the information.
* **Compile** = Compiling the source code.
* **Test** = Testing the compiled source code.
* **Packaging** = Create jar/war files as mentioned in pom.xml.
* **Integrated test =** Process and deploy the package into an environment if necessary, where integrate test can run. Test results are stored in **surefire-reports** dir.
* **Verify =** Run any checkups to verify the package is valid and meets quality criteria.
* **Install** = Install the package in local/maven repo.
* **Deploy** = Copy the final package to maven repo.
* Maven follow this life cycle to build a project. If you do maven install directly, it won’t skip the above stages, it will perform all the stages above install stage and finally installs it.
* To install maven, we need java. So, Install java first, set home path for java and download maven from official website, extract and set home path.
* Test wether maven is installed (or) Not…type…**mvn –version.**
* You can use maven commands along with the stage names…**mvn** **install**, **mvn** **package** etc.
* **mvn** **clean** = Cleans(removes) target directory.
* **mvn clean install =** removes target dir and install once again newly.
* You can also deploy artifacts to containers(tomcat) directly with tomcat plugin.
* By configuring the plugin in pom.xml file. After building an artifact, maven will deployed to tomcat.

**MAVEN REPOSITORIES**

* Maven ha 3 types of repositories.
* **Local** :
* By default, maven stores all your dependencies(plugins, jars and other downloaded by maven) in a local folder. In simple words, when you build a maven project, all the dependency files will be stored in **maven** **local** **repository**.
* The default local folder in linux is **~/.m2**. In this directory, you can see all your dependencies installed by maven.
* You can change this default repo dir by adding a line in setting.xml file.
* Go to settings.xml file, add a line below local repository arrow line.

**<localRepository>/path/to/repo</localRepository>**

* **Central :**
* It is a repository which is provided by **maven** **community**. It contains large number of commonly used libraries.
* When maven doesn’t find any dependency in local repository , it searches in central repo using this url **-** [**https://repo1.maven.org/maven2/**](https://repo1.maven.org/maven2/)**.**
* **Remote:**
* If maven doesn’t find dependency in both local and central repo, it gives you an error by stoping the build. To prevent this, maven provides concept of remote repository, which is our own custom repository containing dependencies and other jars.
* We have to mention remote repo url in pom file and it will search the remote repo for dependencies while building the project.
* Maven will search for dependencies from local repo to central, if its there in central, it will download the dependency. If its not there, it will search in remote only if we configured remote repo in pom.xml file.
* Once we configured the remote repo, it will search and download the dependencies to local.

**MAVEN COMPILER PLUGIN**

* It is used to compile the sources of your project. We don’t need to define it in pom file. Maven downloads it whenever it needed.
* We can configure the plugin in pom to define the way to compile our classes.
* Compiler-plugin has two goals.
* **Compile** = compile the class under /src/main/java
* **Test**-**compile** = compile the class under /src/test/java.
* As we said before, there is no need to specify these src directories in pom, maven will take care of it. All we need to do is put the source in these directories.

**NAGIOS**

* It is a tool for actively monitoring services and devices in your server. It can support monitoring up to thousands of services and devices. It is the most used open source monitoring tool.
* It monitors servers and services associated with the servers. If these not working properly it sends an notifications in the form of mail for the person or a group.
* Normally, you need to write code to monitor the server. But, in nagios it uses plug-ins to check the specific servers and it has its own database to store the data. You can use the stored data to see the server status and errors. you can generate a report from this database.

**INSTALL NAGIOS**

* Download epel rpm package from internet, install it and do system update.
* Install Nagios = **yum** **install nagios nagios-plugins-all nagios-plugins-nrpe nrpe**.
* Install http and php for dashboard = **yum install httpd php**\*.
* Add nagios user to apache and nagios groups.

**Usermod –G nagios nagios.**

**Usermod –G nagios apache.**

* Now, create an nagios account to login to nagios web interface.

**htpasswd –b –c /etc/nagios/passwd nagiosadmin nagiosadmin**

* Start the apache server and nagios server.
* Go to browser, type **yourip/nagios**. It will ask for user name and password that we created earlier. Login with those credentials to see nagios dashboard.
* Disable **selinux**, if it shows **”X not running”** error in nagios dashboard.
* To check errors in nagios configuration files =

**/usr/sbin/nagios -v /etc/nagios/nagios.cfg** – If you installed with yum.

**/usr/local/nagios/bin/nagios –v /usr/local/nagios/etc/nagios.cfg** – if you installed with core.

* If you want use different username and password for web interface.
* Go to /**etc/nagios/objects/cgi.cfg** file and add your user name in the fileds where you see the username “nagiosadmin” .
* In /etc/nagios directory, there is a directory called **objects**. In this objects dir, all the supporting config files will be there.

In Nagios.cfg file,

**Status\_update interval** = time(sec) – to update status.dat file.

**Nagios\_user** – the effective user where nagios should run as.

**Nagios\_group** – the effective group that nagios should run as.

**Set log\_notifications** = 1 – it will log the notifications in nagios.log file.

* When we first run nagios, it checks the nagios.cfg file, using this file, nagios will read all the other config files written in this file and monitor based on our configuration.
* **Commands.cfg** – it contains the commands for all the checks. Means that, when we define a check to host, it will monitor host based on these commands as mentioned in commands.cfg file.
* **Templates.cfg** – it contains group of values used to define the hosts (or) services to monitor. You can create your own template and give the values based on your need. By these templates, we don’t need to give check values for every host; we can simply use a template which has specific values.
* **Localhost.cfg** – we need to define the servers to monitor based on our config. We need to give the details of our hosts and template to use to monitor in this file. We need to define hostname, template, alias, host ip.

After defining the host, we have to define the services to the host. You can define those in the same file below few lines. There you have to mention hostname, checks, service description.

**MONITOR REMOTE SERVER**

* You can monitor remote server by using nagios **nrpe** plugin. You have to install nrpe in remote server and add the remote server configurations in nagios server.
* Nrpe plugin will fetch the remote server data to nagios server.
* Install nrpe plugin = **yum install nagios-plugins-all nagios-nrpe nrpe**.
* After installing, edit the **/etc/nagios/nrpe.cfg** file,

**port = 5666.**

**allow from = 127.0.0.1, server ip.**

**don't blame nrpe = 1**.

* If we configured everything correctly, try Start the nrpe service.
* Go to nagios server.
* In **/etc/nagios/objects dir**, Create a new cfg file, define hosts and services. Add a line in **nagios.cfg** for the remote host.
* Check for the errors and restart nagios server to update the web-interface with remote server.

**MONITOR MYSQL SERVER**

* You can monitor mysql databse with nagios tool. You have to download mysql plugin and configure the plugin in nagios and create an user in mysql and give permissions to that user for accessing mysql from nagios.
* Login to mysql server and create a user named nagios with full access. And create user nagios again with anywhere access(%). Type % in the place of localhost.
* Go to **nagios exchange**, download mysql plugin = **check\_mysql\_health** .
* Go to extracted plugin dir, compile the code = **./configure, make** and **make install**.
* You have to install perl modules = **yum install perl-DBD-MySQL.**
* Go to commands.cfg file, paste this script at the end of file to monitor mysql.

**define command{**

**command\_name check\_mysql\_health**

**command-line $USER1$/check\_mysql\_health -H $ARG1$ --port $ARG2$ --username $ARG3$ --password $ARG4$ --mode $ARG5$**

**}**

* Go to **localhost.cfg**, define services for mysql, save the file check for errors and restart.
* **./check\_mysql\_health –H serverip –user mysql-user –password password –mode** **uptime** = To check mysql uptime.

**Define service {**

**Use local-service (template)**

**Host name localhost (or) server-ip**

**Service description Mysql Uptime**

**Check\_command check\_mysql\_health!server-ip!3306!db-user!db-pass!uptime**

**}**

* Now, mysql is available in nagios dashboard.
* If you want to check another service in mysql, copy the defined service and add the service name in the place of uptime.

**NOTIFICATIONS**

* Nagios will send notifications in any specific action occurs based on our configuration.
* There will be default templates for contacts in templates.cfg file to send mails at regular times, specific actuion occurs etc.
* You can use those templates to send notifications.
* All you have to do is define contacts and contactgroups and use those contactgropus in hosts to send notifications by using those default templates.
* Before defining contacts in nagios, make sure you have Installed and configured **postfix** mail server, where the nagios server running to send mails.
* To define contacts.
* Go to /etc/nagios/objects/contacts.cfg.
* Define a contact under contact section.

**define contact{**

**contact\_name name**

**use generic-contact (template to use)**

**alias alias for name**

**email email to send notifications**

**}**

* Create as many contacts as you want.
* Define contactgroup under contactgroups section to add multiple contacts as a single group. So, if we assign notifications this group, all the users in this specific group will get notified when an specific action occurs.

**define contactgroup{**

**contactgroup\_name name of the group**

**alias alias name for the group**

**members contacts to add to this group**

**}**

* Go to localhost.cfg file (or) the file you have defined the hosts and services.
* Add **contact-groups group-name** line in hosts and services sections you want to get notified.

**DOCKER**

* In early days, we use separate systems for every application. Only that application will be running on that particular system. It is hard to maintain all the servers and it is money consuming.
* Later, **virtualization** came into picture, where we can host multiple os and applications in a single system as an independent machine using physical machine’s hardware. We can create n number of vm’s in a single machine, but the performance would be slow.
* Every vm acts as a independent machine with its own cpu, ram, hd taken from physical machine. The problem is, if the application in vm is using only half of the ram from the assigned. The rest is wasted as it can’t be used by any other vm’s.
* But with **docker**, you can avoid this problem. docker will take only as much as it want because it resides on host kernel by sharing kernel hardware. You don’t need to install any other software to use docker. Just you need to install docker.
* Docker is an containerazation platform which is open source. It is a light weighted and fast container which can be used to package, run and distribute applications with in seconds. You can create n number of containers as you want in a single system.
* Docker packs applications and its dependencies, libraries together and gives you as one docker image from which you can launch containers. The container is given by docker hence we call it docker container. No other dependencies are required to run docker container hence it is light weighted.
* By providing a light weighted containers, it enables very efficient utilization of hardware resources (cpu, ram, hd) etc.
* You can ship containers to other platforms and run those containers in there. But with VM’S you can do these actions.

**COMPONENTS**

* **Docker** **image** = Images are building blocks of docker. It takes series of instructions from a text based configuration called **dockerfile** and built the infrastructure according to that file. It can run in any environment where docker is installed.
* **Docker Container** = Containers are runtime (or) execution aspect of docker. Docker borrows the concept of normal shipping containers that ship goods and used it in docker, which is used to ship software. it can create, starts, stops, restarts, destroyed the containers.
* Just like shipping containers, docker doesn’t care of what is inside of a container, it performs all the actions in the same way for all the containers either it is a host os (or) a web server (or) a database server. You can ship and run docker containers in any environment where docker is installed.
* **Docker Registry** = Docker regestries are the distributed component of docker. Docker stores all the build images in the registry.
* There are 2 types of registeries **public** and **private**. Docker inc provides public registry for images called **docker** **hub**. You have to create an account to store and share your images.
* Docker hosts public repositories called docker registry(hub). Where you can find docker images to download and can run in your local machines.
* These public images are ready with os and applications you needed, you just have to download (pull) the image to your local machine.
* You can create your own images and share them to others by uploading them to docker hub.

**INSTALLATION**

* To install docker, first you have to add docker yum repository in your linux machine.

**yum-config-manager --add-repo** [**https://download.docker.com/linux/centos/docker-ce.repo**](https://download.docker.com/linux/centos/docker-ce.repo)

* Install docker **= yum install docker-ce.**
* If you are using aws ec2 instance, the docker yum repo is present by default. So, no need to create yum repo, just give yum install docker. It will install automatically.

**COMMANDS**

* **docker --version (or) -v =** To see docker version.
* **service docker start =** To start docker**.**
* **docker images =** To see if you have any docker images in your server**.**
* **docker search image** = To search if the specific app image is there in docker hub (or) not.
* **docker pull image-name =** To download a docker image to your local system.
* **docker run –it image**  = To create and login to docker container from docker image.

**i – interactive, t = terminal**

* **docker run –itd** **image** = To create container without login into that.

**d – detached mode.**

* We can create multiple containers from an image. Every container differs from each other. Even after deleting the image, you can start and use docker container.
* **exit** = To exit from container. It will stop the container.
* **Ctrl + p + q** = To come to your local system leaving docker in running state. Container will be running.
* **docker attach container-id** = To get back to the running container.
* **docker stop container-id** = To stop running container.
* **docker start container-id =** To start exited container.
* **docker kill container-id =** To kill a running container.
* **docker top container-id** = Shows top running process in that container.
* **docker stats container-id =** Shows container statistics like cpu usage, memory usage & limit.
* **docker rm container-id –** To delete a container
* **docker rmi image** = To delete an docker image.
* **docker rmi –f image** = To delete docker image forcefully.
* **docker images –q =** Shows images with only image id.
* **docker** **ps** = Shows running docker images.
* **docker ps –a** = shows running and exited containers.
* **docker inspect image** = Shows every detail of that image.
* **docker history image** = Shows commands run on that image.
* **docker export container-name > file.tar –** To persist containers to a file.
* **docker import – containername-export:latest** – To import an exported container.
* **docker save image-name > file.tar** – To persist images to a file.
* **docker load < file.tar** – To load the saved tar ball.
* **Export** – didn’t export history and layers.
* **Save** – exports layers and history.
* We can rollback to previous layers if we exported image with **save** cmd. Because, it save all the layers and history. You can’t rollback with export, because it doesn’t save layers & history.
* **docker tag layer-id imagename** – To rollback to specific layer.
* **docker cp file container-id:/destpath** – To copy a file from local to docker running container. It will replace the old file with this new file.

**DOCKER HUB**

* you can push your images to docker hub.

**Commit your docker image with your repo.**

**Login to docker hub from command line.**

**Push the image to hub.**

* **docker commit container-id repo/user** - To commit the image.
* The repository name and user name are important as they should be in order like **reponame/username.**
* After commiting an image check the docker images in your system, you will find your newly commited container as image. You can use this image to launch docker containers.
* **docker login** – To login to hub from cmd line. Enter you username and password.
* **docker push user/repo** – To push commited images to docker repo.
* While commiting image to repo give **tag** after the repository name to identify in hub.
* If you didn’t specified any tag, it wiil take the default tag(**latest**) for all the images.
* **docker commit container-id repo/user:tag** – to commit along with tag.
* **docker push repo/user:tag** – To push along with tag.

**DOCKER PRIVATE REGISTRY**

* We can push our own images to docker hub, which is a public repository. Everyone cann see our images.
* There might be a situation, where wee need to maintain our own private repository for our organizations/teams called **registry**.
* We have to download the registry and can push our local images to that private registry.
* **docker run –d –p 5000:5000 --name registry registry:2** =
* **registry** – It is the container name to host private repository.
* **-d** – To run container detached mode.
* **-p** – To specify port.
* **5000:5000** – mapping the port of docker to localhost.
* **2** – we are tagging the container to differentiate it from host.
* Check the docker images in your system, to know whether registry in downloaded (or) not and check docker ps to verify its running (or) not.
* To push images to private repo, first we have to tag those images with registry.
* **docker tag image-id localhost:5000/image-name** – To tag an image to localhost private registry.
* **docker push localhost:5000/image-name** – To push the tagged image to private registry.
* Now, delete the pushed image from your host system and try to pull it from private registry.
* **docker pull localhost:5000/image-name** – To pull an image from private registry.
* The docker private registry container should be running to do all the actions (pull, push).

**DOCKER VOLUMES**

* Basically, containers are ephermeral, means once a container is removed it is gone and all the data will be gone inside a container.
* With the help of volumes, we can separate the data from container life cycle. Means, even after the container is deleted, the volume will be present, which we can reuse.
* **Data volumes –**

**It is a specially designed directory in containers.**

**It is created when container is created.**

**It won’t delete when container is deleted.**

**Data volumes can be shared across containers too.**

* **docker volume create --name vol-name** – To create a volume.
* **docker run –it –v vol-name:/vol-name image-name** – To create and mount vol to container and create container.
* It will mount the volume in container and launch the container in interactive mode. Write some data in the volume dir and exit from container.
* Once exiting from the container, it will be no longer available, but the volume will be present. You can check by command below.
* **docker volume ls** – To check all the volumes present. Find your volume in the list by your vol name.
* **docker volume inspect vol-name** – To see what we did in the volume. It will show tou the data you added to that volume.
* Create a new container and mount this volume to that container. Afetr entering into container you will see the data, which saved in your volume.
* **docker volume rm vol-name** – To delete a volume.
* Upto now, we have created new volumes and attached it to containers. We can also create new volumes and copy the existing volume dir data to that new volumes.
* **docker run –it –v old-vol:/new-vol image-name** – Creates a new vol and copies the data from old vol to new vol in the container. New volume name can be anything.
* You can share the volumes between multiple containers. Just mount the volume to different containers while creating them. That’s it.
* **docker run –it –v oldvol:/newvol image-name** – It will mount a volume in the container and copies the data from oldvol.
* If you have change the data in oldvol, the newvol will be updated automatically and viceversa.
* **docker run –it –v oldvol:/newvol:ro image-name** – To attach a vol with read only permissions.
* If you stopped a container from running, the volume will be unmounted and once the container starts, again the volume will mount automatically.
* You can also mount a local dir in docker container.
* **docker run –it –v localdirpath:/mountvolname image-name** – To copy and mount a local dir to docker container.
* It will update the data automatically, once the change occurs either in local dir (or) in docker container.

**LINKING CONTAINERS**

* In order to connect multiple containers together and transfer information between them securely, we link containers.
* For example, we have a mysql container and web server container, we have to link these two container to talk to each other.
* **Docker Link flag** – This method is deprecated. No one uses this method now.
* You can also link container using docker **network**. By launching docker containers in same network you can make communication between them.

**Create your own network.**

**Launch containers in your own network. Try to ping.**

* **docker network ls** – To see all networks in docker.
* **docker network create --driver bridge n/w-name** – To create a bridged network with a name.
* Now launch db container and webserver container in this network .
* **docker run –d --network n/w-name --name con-name img-name** – To launch a container in specified network.
* **docker run –d --network n/w-name --name con-name –e MYSQL\_ROOT\_PASSWORD=password img-name –** To launch mysql container.

**DOCKER FILE**

* A docker file is a script which contains commands and instructions, which executes them sequentially in order to create an docker image.
* Here are some dockerfile commands we use.
* **FROM** – The base image to create our new image. It should be on top of the dockerfile.
* **MAINTAINER** – Contains the name of the maintainer of the image.
* **RUN** – Used to execute commands during build process of docker image.
* **ADD** – Used to copy file from host machine to docker container. It have decompression feauture. You can use url to download and copy files.
* **COPY** – same as ADD. But, doesn’t have decompression feauture.
* **ENV** – Define an environment variable.
* **CMD** – Used to executing commands while building a new container from docker image.
* **ENTRY** **POINT** – Defines the default command to execute while container is running.
* **WORKDIR** – Directive for CMD command to execute.
* **USER –** Set the user for container created from the image.
* **VOLUME –** Enable access between local machine and docker.
* We write a sample file to install httpd.

**FROM centos**

**MAINTAINER** [**name@mail.com**](mailto:name@mail.com)

**RUN yum update –y && yum install httpd –y**

**EXPOSE 80**

**ADD /root/index.html /var/www/html**

**ENTRYPOINT [“/usr/sbin/httpd,”-D”,”FOREGROUND”]**

* Execute the above docker file with docker build command.
* **docker build –t imagename:tag dockerfilepath** – To build image from docker file.
* **-t** – Giving tag to that image.
* After building an image run the image to launch container with httpd. Check the browser with your host ip and port you mentioned in dockerfile.
* **docker run –d –p 8080:80 imagename:tag** – To launch container from your custom image.
* **8080:80** is used to bind the port with localhost port.
* **-d** – to run container in detached mode.
* If you want to install nginx with docker file. Just change the **entrypoint** to **[“nginx”, “-g daemon off;”].**
* You can add volumes in docker file **VOLUME /sourcepath /mntpoint**.
* We can deploy war files by using tomcat docker container. We can write dockerfile by using tomcat image to login to manager host.
* Download tomcat docker image.
* Write a dockerfile to copy users file to docker container.

**FROM tomcat:latest**

**MAINTAINER name**

**ADD tomcat-users.xml /usr/local/tomcat/conf/**

**EXPOSE 8080**

* Execute the docker file to build a new image from tomcat image with the additional settings as you specified.
* **docker build –t imagename:tag .** – To create a new image from docker file.
* Run the docker image to start tomcat - **docker run –d –p 8080:8080 imagename:tag**
* It will start your custom tomcat container. You can login to tomcat manager console with your credentials.
* If you want to deploy war file to tomcat container. Just Add copy to dockerfile and build it.

**COPY warfilepath /usr/local/tomcat/webapps/name.war**

* It will deploy war to tomcat. You can search by url with port and app name.
* If you want to use oracle jdk, you have to use **dordoka/tomcat** image, which contains oraclejdk and tomcat. Official oracle jdk image is not available in docker.

**DOCKER COMPOSE:**

* Compose is a tool for defining and running multi-container applications. With compose, you use a yaml for to configure your application services. Then, with the single command you can create and start all the services from the configuration.
* Compose is basically three steps.

**Define your app environment in a dockerfile.**

**Define the services that make up your app in compose.yml file. So, that they can run together in isolated environment.**

**Run docker-compose up and docker starts and run your entire app.**

* To install docker compose, you need python-pip.
* Install pip first and install docker compose.
* **pip install docker-compose** – To install compose.
* **docker-compose --version** – To check version. (if output shows, you have installed compose successfully).
* The yaml file should be named as docker-compose.yml. Otherwise docker compose won’t read the file.
* **docker-compose up** – To run the containers.
* **docker-compose stop** – To stop all the containers.
* **docker-compose ps** – To see all running containers started with compose tool.
* **docker-compose start** – To start all containers which created by compose tool.
* For ex:

**version: ‘3’**

**services:**

**nginx:**

**image: nginx**

**ports:**

**- 8080:80**

**volumes:**

**- “./localdir:/dockerpath”**

**networks:**

**- networkname**

**mysql:**

**image: mysql**

**ports:**

**- 3306:3306**

**environment:**

**MYSQL\_ROOT\_PASSWORD: password**

**networks:**

**- networkname**

**wordpress:**

**image: wordpress**

**ports:**

**- 8008:80**

**environment:**

**WORDPRESS\_DB\_HOST: mysql**

**WORDPRESS\_DB\_PASSWORD: password**

**networks:**

**- networkname**

**networks:**

**networkname**

**driver: bridge**

* In the above example, we have created nginx, mysql and wordpress in the same network.
* We have made an connection between mysql and wordpress by entering wordpress host as mysql.
* Check the connections with localhost and ip whether services are working (or) not.
* Normally we have to write separate dockerfiles for each service and build those images one after the other which wil take more time. But with docker compose, we can up the 3 servcies with one command by writing all of the services in docker-compose file.
* You can use your dockerfile with docker compose. So, it will build from dockerfile and start the services from compose file.

**JENKINS INTEGRATION:**

* Jenkins allows us to use docker host for provisioning build agents, run an build agent and delete agent after build finishes.
* We have to use **docker plugin** to integrate docker with Jenkins.