

GITHUB

- Github is a web-based platform used for **version control**.
- it simplifies the process of working with other people and makes it easy to collaborate on projects.
- Team members can work on files and easily merge their changes in with the master branch of the project.

Now if you want to pull your code to Github

git remote add origin url

git push -u origin branch-name

go to Github and check the files that you have pushed.

GITBRANCHES:

A branch represents an independent line of development.

The git branch command lets you create, list, rename, and delete branches.

The default branch name in Git is master.

- | | | |
|----------------------------------|---|-----------------------------|
| • To see current branch | : | git branch |
| • To add new branch | : | git branch branch-name |
| • To switch branches | : | git checkout branch-name |
| • To create and switch at a time | : | git checkout -b branch-name |
| • To rename a branch | : | git branch -m old new |
| • To clone a specific branch | : | git clone -b branch-name |
| repo-URL | | |
| • To delete a branch | : | git branch -d <branch> |

The -d option will delete the branch only if it has already been pushed and merged with the remote branch. Use -D instead if you want to force the branch to be deleted, even if it hasn't been pushed or merged yet. The branch is now deleted locally.

Now all the things you have done is on your local system.

Now we will go to GIT HUB.

PUBLIC REPO & PRIVATE :

In Git-Hub we have two types of repos one is public and another is Private

PUBLIC REPO:Public repositories are accessible to everyone on the internet.

Making a Github repo 'public' will mean that anyone can find it and fork it to get the source files for themselves. This means that anyone would have access to your client's source files.

PRIVATE REPO:

Private repositories are only accessible to you, people you explicitly share access with, and, for organization repositories, certain organization members.

They offer security protections that public repositories don't, such as ensuring that sensitive data like passwords, SSH keys, API keys and other information isn't accidentally expose

MAVEN INTRODUCTION:

- It is an Automation Project management tool developed by Apache software foundation.
- It is based on POM (Project Object Model). (POM.xml) xml: eXtensible Markup Language.
- it is a build tool and manage dependencies.
- It can build any number of projects into desired Output such as .jar, .war and metadata.
- It is mostly used for java-based projects.
- It was initially released on 13 July 2004.
- Maven is written in java.
- Meaning of Maven is Accumulator of Knowledge in Yiddish language.
- Maven helps in getting the right jar file for each project as there may be different version of separate packages.
- For downloading dependencies visit mvnrepository.com.

Dependencies: It refers to the java libraries that are needed for the project

Repositories: Refers to the directories of Packaged jar files.

Build tools:

- C, C ++ : make file
- .Net : Visual studio
- Java : Ant, Maven, Gradle

PROBLEMS WITHOUT MAVEN

- **Adding set of jars in each project:** In case of Struts, Springs we need to add jar files in each project. It must include all the dependencies of jars also.
- **Creating the right project structure:** We must create the right project structure in Servlet, Struts etc. Otherwise it will not be executed.
- **Building and Deploying the Project:** We must build and deploy the Project, so it may work.

WHAT MAVEN DOES?

- It makes project easy to build
- Provides Project Information (log document, cross reference, sources, mailing list, unit test).
- Easy to add New Dependencies.
- Apache Maven helps to manage: Build, Dependencies, Releases, Distribution, Reports.

WHAT IS BUILD TOOL?

- A build tool takes care of everything for building a process. It follows
- Generate Source Code.
- Generate Documentation for Source code.
- Compile Source Code.
- Install the Package code in the local repo. Server repo or Central repo.
- POM refers the XML file that have all information regarding project and configuration details
- Main configuration file is in pom.xml.
- It has description of the Project details regarding version and configuration management.
- The XML file is in the Project home directory.

Pom.xml contains

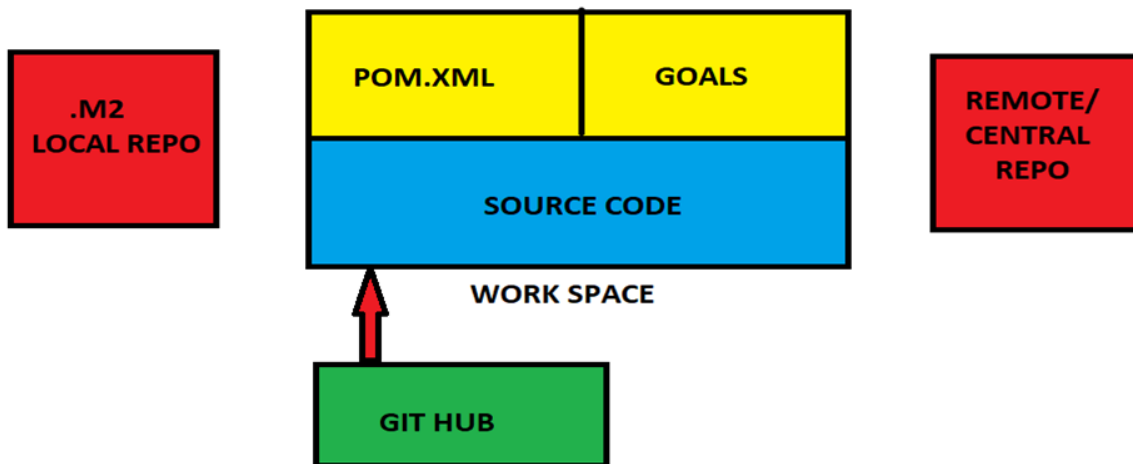
- Metadata.
- Dependencies.
- Kind of Project.
- Kind of Output.
- Description.

One project consists — > One workspace consists — > of only One pom.xml file.

REQUIREMENTS FOR BUILD:

- Source code : Present in work space.
- Compiler : Remote repo — > Local repo — > Workspace.
- Dependencies : Remote repo — > Local repo — > Workspace.

MAVEN ARCHITECTURE:



MAVNE BYILD LIFE CYCLE

- Generate Resource (Dependencies)
- Compile Code
- Unit test
- Package (Build)
- Install (into Local repo and Artifactory)
- Deploy (to Servers)
- Clean (to delete all the Runtime files) — > Optional

Ex: mvn install, mvn clean package

- Build lifecycle consists of sequence build phases & each build phase consist sequence of goals
- Each goal is responsible for Particular task. Goals are nothing but commands.
- When a phase is run, all the goals related to that phase and its plugins are also compiled.

Maven Compile:

Compiles source code of the project.

Maven Test:

Runs tests for the project.

Maven Package:

Creates JAR or WAR file for the project to convert it into a distributable format.

Maven Deploy:

Copies the packaged JAR/ WAR file to the remote repository after compiling, running tests and building the project.

Maven Clean:

Cleans the project and removes all files generated by the previous build.

JENKINS:

JENKINS

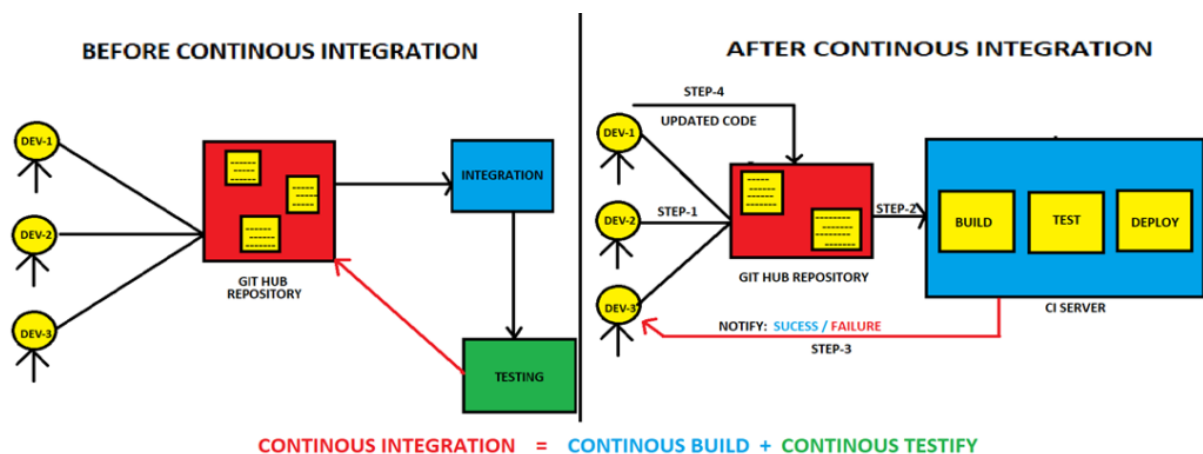
CI = CONTINUOUS INTEGRATION
DELIVERY/DEPLOYMENT

CD = CONTINUOUS

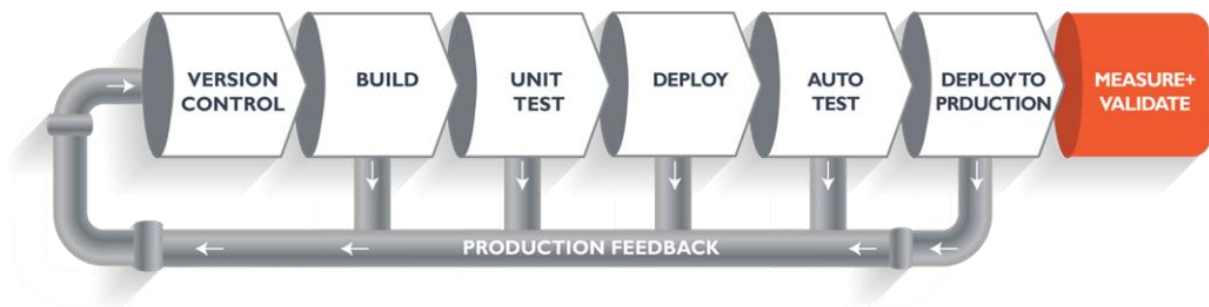
CI/CD means it is not a tool a it is a methodology/Framework which used to develop SDLC.

Pipeline follows First come first serve

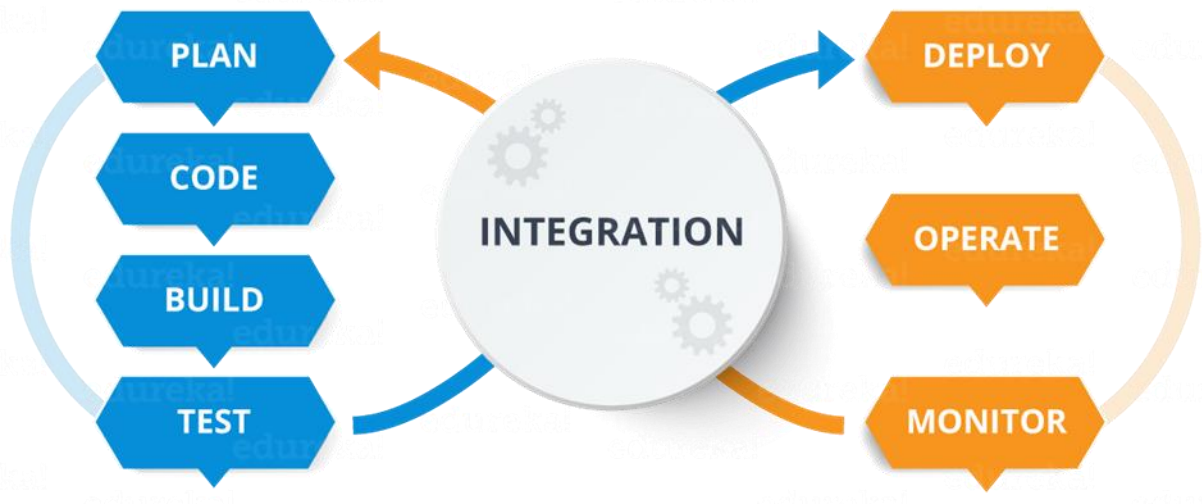
WHY WE USE IT



If we have error in **Code** then it will give feedback and it will be corrected, if we have error in **Build** then it will give feedback and it will be corrected, Pipeline will work like this until it reaches **Deploy**.



edureka!



- Jenkins is an Opensource project written in java that runs on Windows, Linux, Mac-OS.
- It is community supported and Free to use and First choice for Continuous Integration.
-
- Consist of Plugins
- Automates the Entire Software Development Life Cycle (SDLC).
- It was Originally developed by **Sun Microsystem** in **2004** as **HUDSON**.
- Hudson was an enterprise Edition we need to pay for it.
- The project was renamed as Jenkins when Oracle brought the Microsystems.
- It can run on any major platform without Compatibility issue.
- Whenever developers write code, we integrate all the code of all developers at any point of time and we build, test and deliver/deploy to client. This is called as CI/CD.
- Because of this CI, Bugs will be reported fast and get rectified so entire development is fast.

WORKFLOW

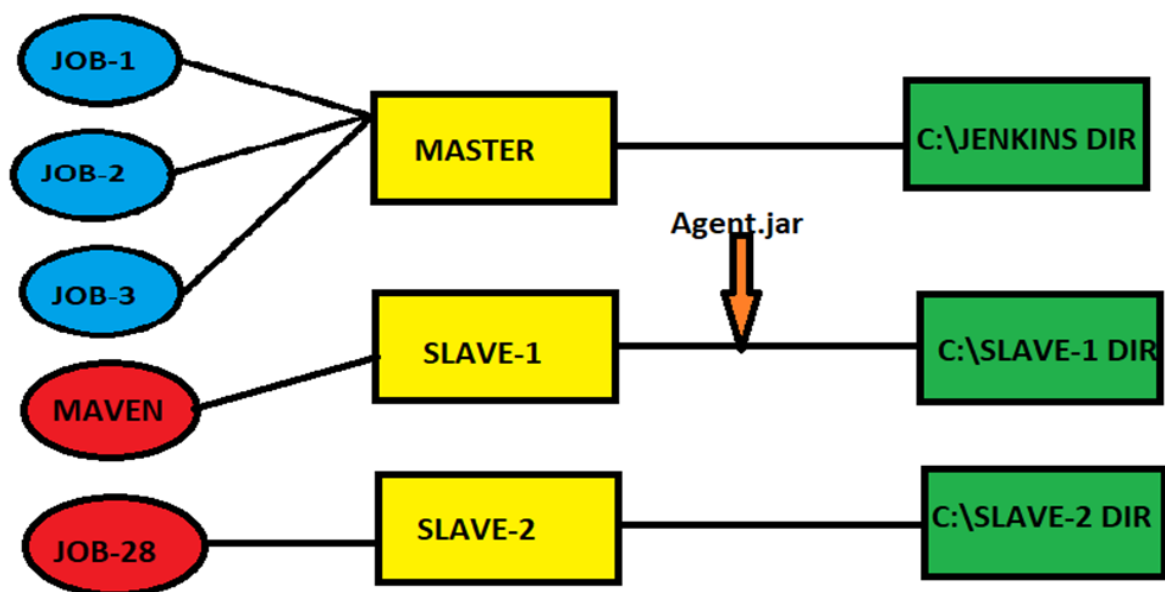
- We can attach Git, Maven, Selenium and Artifactory plugins to the Jenkins.
- Artifactory consists of final code which is ready to use.
- Once Developer put code in GitHub Jenkins pull that code and send to Maven for Build.

- Once Build is done, Jenkins pull that code and send to Selenium for Testing.
- Once Testing is done, Jenkins pull that code and send to Artifactory as per requirement.
- We can also Deploy with Jenkins.

ADVANTAGES

- It has lot of plugins and You can write your own plugin, can use community plugin also.
- It is not a tool it is a framework. i.e., you can do whatever you want all you need is plug-ins.
- Jenkins follows Master-Slave Architecture.
- We can attach slaves (Nodes) to Jenkins's master. It instructs other (Slaves) to do the Job.
- If Slaves are not available Jenkins itself do the job.
- Jenkins also behave as Server Replacement. i.e., it can do schedule job.
- It can create labels. i.e., means who will do that task and assigns the tasks.

MASTER-SLAVE CONCEPT



Jenkins Setup

sudo yum update -y

```
sudo wget -O /etc/yum.repos.d/jenkins.repo https://pkg.jenkins.io/redhat-stable/jenkins.repo
```

```
sudo rpm --import https://pkg.jenkins.io/redhat-stable/jenkins.io.key
```

```
sudo amazon-linux-extras install epel -y : Extra package for enterprise linux
```

```
sudo yum install java-1.8.0-openjdk -y
```

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

```
sudo yum install maven -y
```

```
sudo yum install jenkins -y
```

```
sudo systemctl restart jenkins
```

```
sudo systemctl status jenkins
```

copy the IPV4 and paste it on browser like {ipv4:8080}

```
sudo cat /var/lib/jenkins/secrets/initialAdminPassword
```

```
sudo vim /etc/passwd
```

```
sudo passwd jenkins
```

```
sudo visudo
```

```
sudo vim /etc/ssh/sshd_config
```

```
sudo systemctl restart sshd
```

```
sudo systemctl status sshd
```

```
=====
```

LOGIN TO SLAVE SERVER

```
sudo useradd jenkins
```



```
sudo passwd jenkins
```

```
sudo visudo
```

```
sudo vim /etc/ssh/sshd_config
```

```
sudo systemctl restart sshd
```

```
sudo systemctl status sshd
```

```
=====
```

GO BACK TO MASTER

```
sudo su jenkins
```

```
ssh-keygen
```

```
ssh-copy-id jenkins@localhost
```

```
yes
```

```
exit
```

```
ssh-copy-id jenkins@public IPV4 of slave
```

```
ssh jenkins@public IPV4 of Slave
```

```
=====
```

GO BACK TO SLAVE

```
sudo su – jenkins
```

```
ssh jenkins@public IPV4 of Master
```

```
yes
```

password

logout and restart jenkins

=====

JOBS IN JENKINS

- Name — > freestyle — > ok — > Build — > Add build step — > execute command — > Echo “hello” — > save — > Dashboard — > Select it and build.
- If build is green, it is success and red it will be failed. Check details in console output.
- If you want to copy the job new item — > name: copy project — > copy from: select job want to copy — > Ok — > and you will see the same details of your original job.

Types of Parameters:

Jenkins supports several parameter types. Below is a list of the most common ones, but keep in mind that different plugins may add new parameter types:

String: any combination of characters and numbers

Choice: a pre-defined set of strings from which a user can pick a value

Credentials: a pre-defined Jenkins credential

File: the full path to a file on the filesystem

Multi-line String: same as String, but allows newline characters

Password: similar to the Credentials type, but allows us to pass a plain text parameter specific to the job or pipeline

Run: an absolute URL to a single run of another job

String Parameter:

This parameter is used when we need to pass an parameter as input by default. It can be any combination of characters and numbers.

General — > This Project is Parameterized — > String Parameter

Multi-String Parameter:

This will work as same as String Parameter but the difference is instead of one single line string we can use multiple strings at a time as a Parametres.

General — > This Project is Parameterized — > Multi-String Parameter

File Parameter:

Jenkins provides a File parameter which allows a build to accept a file, to be submitted by the user when scheduling a new build. The file will be placed inside the workspace at the known location after the check-out/update is done so that your build scripts can use this file.

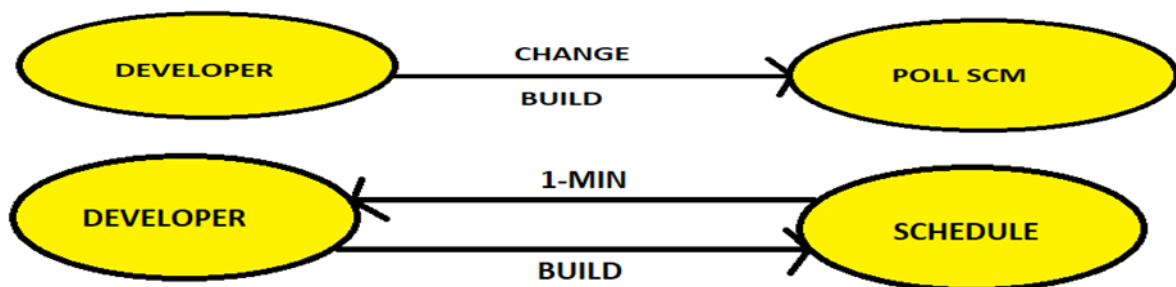
General — > This Project is Parameterized — > File Parameter

SCHEDULE PROJECT

- click on any project — > configure — > build — > triggers — > build periodically — > ***** — >

(1*: minutes, 2*: hours, 3*: days, 4*: month, 5*: week) — > save.

- Can see automatic build after every minute.
- You can manually trigger build as well.
- For schedule jobs it will be built for every one minute continuously.
- For poll SCM it will get build when there will be only change in the file.



What is Jenkins Pipeline?

Jenkins Pipeline is a combination of plugins that supports integration and implementation of continuous delivery pipelines.

A Pipeline is a group of events interlinked with each other in a sequence.

What is a JenkinsFile?

Jenkins pipelines can be defined using a text file called **JenkinsFile**. You can implement pipeline as code using JenkinsFile, and this can be defined by using a domain specific language (DSL). With JenkinsFile, you can write the steps needed for running a Jenkins pipeline.

The benefits of using **JenkinsFile** are:

- You can create pipelines automatically for all branches and execute pull requests with just one **JenkinsFile**.
- You can review your Jenkins code on the pipeline
- You can audit your Jenkins pipeline
- This is the singular source for your pipeline and can be modified by multiple users.

Declarative versus Scripted pipeline syntax:

There are two types of Jenkins pipeline syntax used for defining your JenkinsFile.

1. Declarative
2. Scripted

Declarative:

Declarative pipeline syntax offers an easy way to create pipelines. It contains a predefined hierarchy to create Jenkins pipelines. It gives you the ability to control all aspects of a pipeline execution in a simple, straight-forward manner.

Scripted:

Scripted Jenkins pipeline runs on the Jenkins master with the help of a lightweight executor. It uses very few resources to translate the pipeline into atomic commands. Both declarative and scripted syntax are different from each other and are defined totally differently.

Jenkins Pipeline Concepts

PIPELINE: The pipeline is a set of instructions given in the form of code for continuous delivery and consists of instructions needed for the entire build process. With pipeline, you can build, test, and deliver the application.

NODE: The machine on which Jenkins runs is called a node. A node block is mainly used in scripted pipeline syntax.

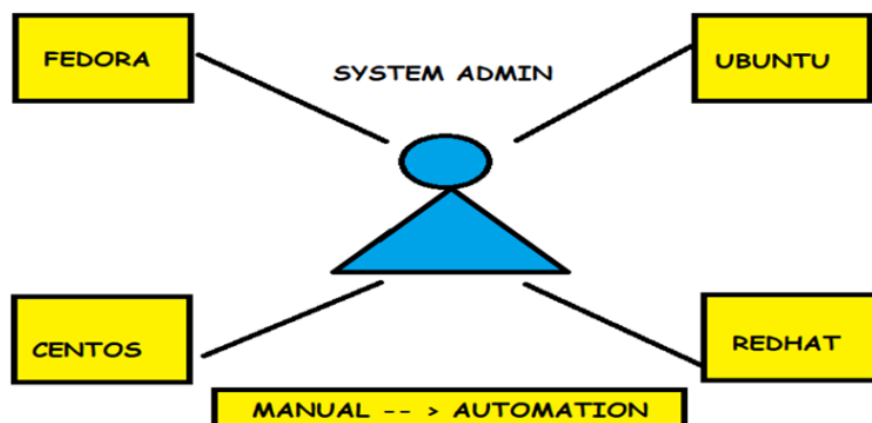
STAGE: A stage block contains a series of steps in a pipeline. That is, the build, test, and deploy processes all come together in a stage. Generally, a stage block is used to visualize the Jenkins pipeline process.

STEP: A step is nothing but a single task that executes a specific process at a defined time. A pipeline involves a series of steps.

ANSIBLE:

- It is a Configuration Management Tool.
- Configuration: Ram, Storage, OS, Software and IP address of device.
- Management: Update, Delete, Add.
- Ansible is simple open-source IT engine which automates application deployment.
- Orchestration, Security and compliance.
- Uses YAML Scripting language which works on KEY-PAIR
- Ansible GUI is called as Ansible Tower. It was just Drag and Drop.
- Used PYTHON for Back end.

HISTORY



- Michael Dehhan developed Ansible and the Ansible project began in Feb 2012.
- Ansible was taken over by Red-hat.
- Ansible is Available for RHEL, Debian, CentOS, Oracle Linux.
- Can use this tool whether your servers are in On-prem or in the Cloud.
- It turns your code into Infrastructure i.e. Your computing environment has some of the same attributes as your application.

If system admin has to install those Linux flavors across all the systems on his company, then he has to do it manually. In manual work there might be some errors so we use here automated tools like

Ansible, Chef, Puppet etc.

ANSIBLE : PUSH CHEF : PULL

PUSH: if we have many servers then it will push the notification for updates in all devices.

PULL: It will go to client server and ask for the notifications for update.

Ansible Important Points

ANSIBLE SERVER: The machine where ansible is installed& from which all task and playbooks will run

MODULE: A command or set of similar Commands meant to be executed on Client side.

TASK: A section that consists of a single Procedure to be completed.

ROLE: A way of organizing the Tasks and Related files to be later called in playbook.

FACT: Info fetched from the client system from the Global variables with the Gather-facts operation.

INVENTORY: File containing Data about the Ansible client servers.

PLAY: Execution of Playbook.

HANDLER: Task which is called only if a notifier is present.

NOTIFIER: Section attributed to a task which calls a handler if the Output is changed.

PLAYBOOKS: It consist code in YAML format, which describes task to be Executed.

HOST: Host or Nodes, which are Automated by Ansible.

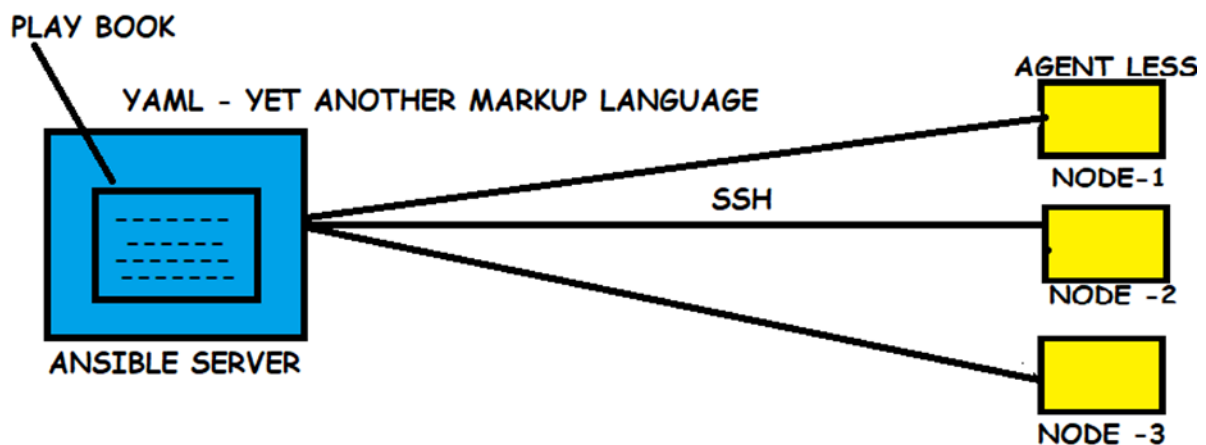
ADVANTAGES

- Very consistent and light weight and no constraints regarding the OS or underlying H.W.
- Secure less due to Agent less Capability and Open SSH Security features.
- Doesn't require any special system admin skills to install and use it (YAML).
- Push mechanism.

DISADVANTAGES

- Ansible does not have any notion of state like other automation tools such as Puppet
- Ansible does not track dependencies and simply executes sequential tasks and stops when tasks finish, fail, or any error comes.
- Ansible has external dependencies to Python modules
- Windows interaction requires some scheming

ANSIBLE WORKFLOW



ANSIBLE INVENTORY HOST PATTREN

- Create 3 EC2 instances in same Availability Zone & Connect through Putty and give sudo su.
- yum update -y
- sudo amazon-linux-extras install ansible2 -y
- yum install git python python-level python-pip openssl -y & check versions.
- vi /etc/ansible/hosts file in Ansible server and [remo] & paste private IP of node-1 & node-2.
- # Vi etc/ansible/ansible.cfg
- Uncommented – > inventory: /etc/ansible/hosts & Sudo-user: root. Save and quit.
- Create user called ansible and set password and add ansible user to sudo file.
- Now do this process on both other nodes too.
- Go to ansible server and install httpd package as ansible user and exit to root.
- Open vi /etc/ssh/sshd_config in **root** in all three servers.
- service sshd restart and login as ansible in all 3 servers .
- Su – ansible & ssh IP of any node it will ask password and enter it then you will be on node-1.
- Create some files on ansible server and it will replicate in node-1

Now again if you want to login in node-1 you need to give password to get rid of that we need to do

- Go to ansible server — > ssh-keygen — > ls -al — > cd .ssh — > ls — > id_ras_pub
- Now we need to copy public key in both the nodes
- ssh-copy-id ansible@private-ipv4 of node-1 and it will ask password enter it.
- Ssh-copy-id ansible@private-ipv4 of node-2 and it will ask password enter it.
- Now go to ansible and ssh ipv4 node-1 it will not ask password now and exit
- ssh ipv4 of node-2 it will also not ask password now.

HOST PATTRENS

- 'all' patterns refer to all the machines in an inventory.
- ansible all-list-hosts

ansible <groupname[remo]> -list-hosts

ansible <groupname> [remo][0] -list-hosts

groupname [0] – picks first machine of group

groupname [1] – picks second machine of group

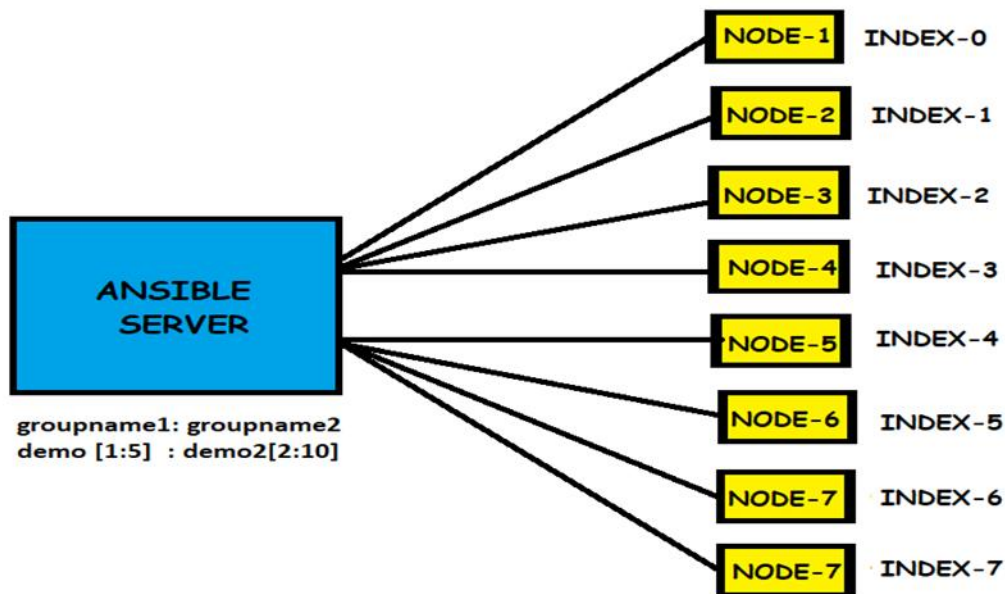
groupname [-1] – picks last machine of group

groupname [1:4] – picks 2,3,4,5 machines in group

groupname [2:5] – picks 3,4,5,6 machines in the group

If we want to push the code from Ansible server to nodes it can be done in 3 ways.

1. Ad-hoc Commands (Simple Linux) Ad-hoc means temporary & it will over-ride commands.
 2. Modules – A Single Command.
 3. Playbooks – More than one module is called Playbook.
- Both module and Playbook is in YAML.



Ad-Hoc Commands

- These commands can be run individually to perform Quick functions.
- Not used for configuration management and deployment, bcz the cmds are one time usage.
- The ansible ad-hoc cmds uses `/usr/bin/ansible/` command line tool to automate single task.

Go to ansible server and switch to ansible server

`ansible remo -a "ls" [remo: Group name, -a: argument, ls: command]`

`ansible remo [0] -a "touch file1"`

`ansible all -a "touch file2"`

`ansible remo -a "sudo yum install httpd -y"`

`ansible remo -ba "yum install httpd -y" (b: become you will become sudo user)`

`ansible remo -ba "yum remove httpd -y"`

ANSIBLE MODULES

- Ansible ships with number of modules (called library modules) that can be executed directly to remote hosts or playbooks.
- Your library of modules can reside on any machine, and there are no servers, daemons or database required.

- The default location for the inventory file is /etc/ansible/hosts
Go to ansible server and switch to ansible server

ansible remo -b -m yum -a "pkg=httpd state=present" (install: present)

ansible remo -b -m yum -a "pkg=httpd state=latest" (update: latest)

ansible remo -b -m yum -a "pkg=httpd state=absent" (uninstall: absent)

ansible remo -b -m service -a "name=httpd state=started" (started: start)

ansible remo -b -m user -a "name=raj" (to check go to that servers and sudo cat /etc/passwd).

ansible remo -b -m copy -a "src=filename dest=/tmp" (to check go to that server and give ls /tmp).

ansible remo -m setup

ansible remo -m setup -a "filter=*ipv4*"

PLAYBOOKS

- Playbooks in ansible are written in YAML language.
- It is human readable & serialization language commonly used for configuration files.
- You can write codes consists of vars, tasks, handlers, files, templates and roles.
- Each playbook is composed of one or more modules in a list.
- Module is a collection of configuration files.
- Playbooks are mainly divided into sections like

TARGET SECTION: Defines host against which playbooks task has to be executed.

VARIABLE SECTION: Defines variables.

TASK SECTION: List of all modules that we need to run in an order.

YAML

For ansible, nearly every YAML file starts with a list

- Each item in the list is a list ok key-value pairs commonly called Dictionary.
- All YAML files have to begin with "—" and end with "..."

- All members of the list line must begin with same indentation level starting with “ — “

For example:

```
— # A list of fruits
```

```
Fruits:
```

```
-mango
```

```
-apple
```

```
-papaya
```

```
-guava
```

```
...
```

- A dictionary is required in a simple key: value form (note: space before value is must)

For example:

```
— # Customer details
```

```
Customer:
```

```
Name: Raham
```

```
Age : 22 y
```

```
Salary: 30,000
```

```
Exp : 1 year
```

- Extension for playbook file is **.yaml**

Go to ansible server and login as ansible and create one playbook

- Vi target.yml

```
—# Target Playbook
```

- hosts: remo — > remo: Groupname

user: ansible — > ansible: You are ansible user now

become: yes — > become: become sudo user — > yes

connection: ssh

gather_facts: yes — > Gives private IP of the nodes — > yes

now save that file and execute the playbook by giving the command: `ansible-playbook target.yml`

Now create one more playbook in ansible server with cmd Vi task.yml

```
--
- hosts: remo
  user: ansible
  become: yes
  connection: ssh
  tasks:
    - name: installing git
      action: yum name=git state=present
```

Now execute the file by command `ansible-playbook task.yml`

VARIABLES

- Ansible uses variables which are defined previously to enable more flexibility in playbooks and roles they can used loop through a set of given values, access various information like the host name of a system and replace certain strings in templates with specific values.
- Write Variable section above tasks so that we define in first and use it later.

Now go to ansible server and create one playbook

```

--- #VARIABLES
- hosts: remo
  user: ansible
  become: yes
  connection: ssh
  vars:
    pkgname: httpd
  tasks:
    - name: installing httpd
      action: yum name='{{pkgname}}' state=present

```

Now save and execute the playbook

HANDLERS

- Handler is same as task but it will run when called by another task. (OR)
- It will run if the task contains a notify directive and also indicates that it changed something.

DRY RUN: Check whether the playbook is formatted correctly or not.

Ansible-playbook handler.yml -check

```

--- # HANDLER
- hosts: remo
  user: ansible
  become: yes
  connection: ssh
  tasks:
    - name: install httpd server on centos
      action: yum name=httpd state=installed
      notify: restart httpd
  handlers:
    - name: restart httpd
      action: service name=httpd state=restarted

```

LOOPS

- Ansible loop includes changing ownership on several files & directories with file module, creating multiple users with user modules and repeating a polling step until result reached.

```

--- # LOOPS
- hosts: remo
  user: ansible
  become: yes
  connection: ssh
  tasks:
    - name: add list of users in my nodes
      user: name='{{item}}' state=present
      with_items:
        - raham
        - mustafa
        - shafi
        - nazeer

```

Now save and execute the file and go to the nodes and check with `cat /etc/passwd`.

ANSIBLE TAGS:

If you have a large playbook, it may be useful to run only specific parts of it instead of running the entire playbook. You can do this with Ansible tags. Using tags to execute or skip selected tasks

```
--
- hosts: remo
  user: ansible
  become: yes
  connection: ssh
  tasks:
    - name: installing git
      action: yum name=git state=present
      tags: install
    - name: uninstalling git
      action: yum name=git state=absent
      tags: uninstall
```

`ansible-playbook abc.yml --skip-tags "uninstall"`

DOCKER

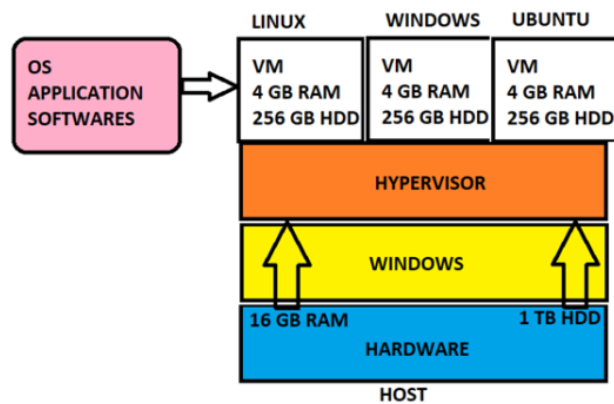
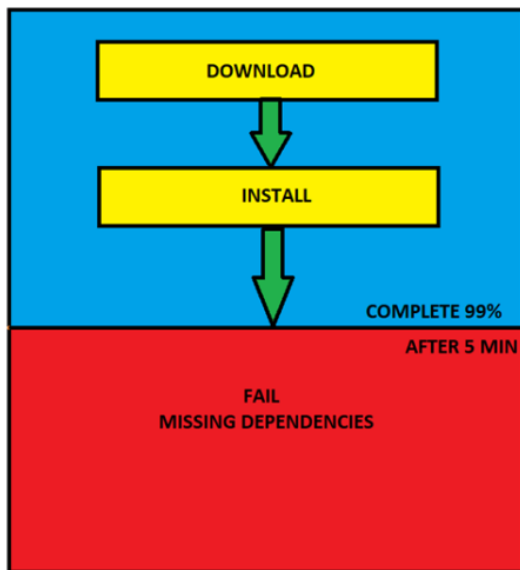
CONTAINER : It is like a Virtual Machine and It does not have any OS.

VIRTUVALIZATION : Process that allows for more efficient utilization of physical computer hardware and is the foundation of cloud computing.

HYPERVISOR : Help to make virtualization and to create a VM

CONTAINARIZATION : Process of packing application along with its dependencies

DOCKER : It is a tool that create this Container. It is advance than Virtualization.

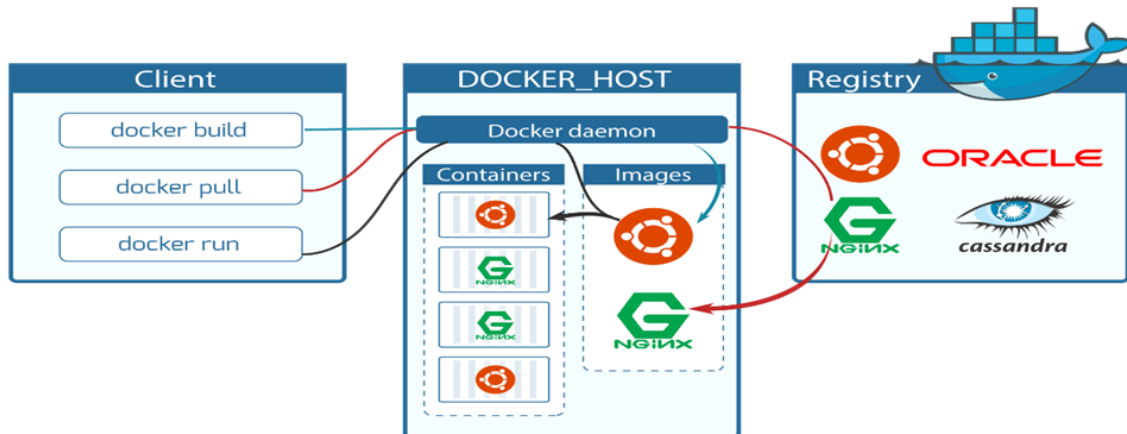


DOCKER

- It is an opensource centralized platform designed to create, deploy and run applications.
- Docker is written on Go language.
- Docker uses container on host O.S to run applications. It allows applications to use same Linux kernel as a system on the host computer, rather than creating a whole virtual O.S.
- We can install Docker on any O.S but docker engine runs natively on Linux distribution.
- Docker performs O.S level Virtualization also known as Containerization.
- Before Docker many user face problems that a particular code is running in the developer's system but not in the user system.
- It was initially release in March 2013, and developed by Solomon hykes and Sebastian pahl.
- Docker is a set of platform-as-a-service that use O.S level Virtualization, where as VM ware uses Hardware level Virtualization.
- Container consists O.S files but its negligible in size compared to original files of that O.S.

ARCHITECTURE

DOCKER COMPONENTS



DOCKER CLIENT: is the primary way that many Docker users interact with Docker. When you use commands such as `docker run`, the client sends these commands to docker daemon, which carries them out. The docker command uses the Docker API.

DOCKER HOST: Docker host is the machine where you installed the docker engine

DOCKER DEAMON: Docker daemon runs on the host operating system. It is responsible for running containers to manage docker services. Docker daemon communicates with other daemons. It offers various Docker objects such as images, containers, networking, and storage.

DOCKER REGISTRY: A Docker registry is a scalable open-source storage and distribution system for docker images.

BASIC DOCKER COMMANDS

To install docker in Linux	: <code>yum install docker -y</code>
To see the docker version	: <code>docker --version</code>
To start the docker service	: <code>service docker start</code>
To check service is start or not	: <code>service docker status</code>
To check the docker information	: <code>docker info</code>

To see all images in local machine	: docker images
To find images in docker hub	: docker search image name
To download image from docker hub to local	: docker pull image name
To download and run image at a time /bin/bash	: docker run -it image name
To give names of a container name /bin/bash	: docker run -it -name raham img-
To start container	: docker start container name
To go inside the container	: docker attach container name
To see all the details inside container	: cat /etc/os-release
To get outside of the container	: exit
To see all containers	: docker ps -a
To see only running containers	: docker ps (ps: process status)
To stop the container	: docker stop container name
To delete container	: docker rm container name

you can create multiple containers by using single image

start vs stop and kill vs remove

TO BUILD IMAGE FROM CONTAINER

- Create a container from our own image.
- Therefore, create a container first.
- docker run -it -name viya ubuntu /bin/bash
- Cd tmp/ and create a file inside it by using touch myfile and exit from container.
- Now create image of this container by using
docker commit container-name update-image.

- Now list images by using docker images
- Now create container for the image using
docker run -it --name remo update-image /bin/bash
- Now go to tmp folder and give ls you will see myfile there.

DOCKER FILE

- It is basically a text file which contains some set of instructions.
- Automation of Docker image creation.
- Always D is capital letter on Docker file.
- And Start Components also be Capital letter.

DOCKER FILE COMPONENTS

FROM: For base image this command must be on top of the file.

RUN: To execute commands, it will create a layer in file.

COPY: Copy files from local system (docker VM) where need to provide Source and Destination.

ADD: It can download files from internet and also, we can extract file at docker image side.

EXPOSE: To expose ports such as 8080 for tomcat and port 80 nginx etc.

WORKDIR: To set working directory for the Container.

ENV: Environment Variables.

FILE CREATION

- Create a file called Docker file and Add instructions in Docker file.
- Build Docker file to create image.
- Run image to Create Container.

Vi Docker file

FROM ubuntu

RUN echo "Hello world!" > /tmp/testfile

- To create image out of Docker file

`docker build -t image-name .` (. = current directory)

`docker ps -a` and `docker images`.

- Now create container from the above image.

`docker run -it --name container-name image-name /bin/bash`

- Cat `/tmp/testfile`.

```
[root@ip-172-31-84-47 ec2-user]# ls
Dockerfile  testfile1
[root@ip-172-31-84-47 ec2-user]# touch test
[root@ip-172-31-84-47 ec2-user]# ls
Dockerfile  test  testfile1
[root@ip-172-31-84-47 ec2-user]# tar -cvf test.tar test
test
[root@ip-172-31-84-47 ec2-user]# ls
Dockerfile  test  testfile1  test.tar
[root@ip-172-31-84-47 ec2-user]# gzip test.tar
[root@ip-172-31-84-47 ec2-user]# ls
Dockerfile  test  testfile1  test.tar.gz
[root@ip-172-31-84-47 ec2-user]# rm -rf test
[root@ip-172-31-84-47 ec2-user]# ls
Dockerfile  testfile1  test.tar.gz
[root@ip-172-31-84-47 ec2-user]# docker build -t newimage .

FROM ubuntu
WORKDIR /tmp
RUN echo "hello world!" > /tmp/testfile
ENV myname raham shaik
COPY testfile1 /tmp
ADD test.tar.gz /tmp
```

Give `docker run -it --name container name image-name /bin/bash`

DOCKER VOLUMES

- When we create a Container then Volume will be created.
- Volume is simply a directory inside our container.
- First, we have to declare the directory Volume and then share Volume.
- Even if we stop the container still, we can access the volume.
- Volume will be created in one Container.
- You can declare directory as a volume only while creating container.
- We can't create volume from existing container.
- You can share one volume across many number of Containers.
- Volume will not be included when you update an image.

- If Container-1 volume is shared to Container-2 the changes made by Container-2 will be also available in the Container-1.
- You can map Volume in two ways
- Container < --- > Container
- Host < --- > Container

CREATING VOLUME FORM DOCKER FILE

- Create a Docker file and write
FROM ubuntu

VOLUME ["/myvolume1"]

- Docker build -t my-image . (Current directory)
- Docker run -it --name container1 myimage /bin/bash
- Now do ls and you will see myvolume-1
- Now share volume with another Container

docker run -it --name container2(new) --privileged=true --volumes-from container1 ubuntu

- Now after creating container2, my volume1 is visible.
- Whatever you do in volume1 in container1 can see in another container.
- touch /myvolume1/samplefile1 and exit from container2.
- docker start container1
- docker attach container1
- ls/volume1 and you will see your samplefile1.

REATING VOLUME BY USING COMMAND

- docker run -it --name container3 -v /volume2 ubuntu /bin/bash
- now do ls and cd volume2
- Now create one file and exit.
- Now create one more container, and share Volume2

Docker run-it --name container4 -- --privileged=true --volumes-from container3 ubuntu

- Now you are inside container and do ls, you can see the Volume2
- Now create one file inside this volume and check in container3, you can see that file.

VOLUMES (HOST – CONTAINER)

- Verify files in /home/ec2-user
- Docker run -it --name hostcont -v /home/ec2-user:/raham --privileged=true ubuntu
- Cd raham [raham is (container-name)]
- Do ls now you can see all files of host machine.
- Touch file1 and exit. Check in ec2-machine you can see that file.

SOME OTHER COMMANDS

- docker volume ls
- docker volume create <volume-name>
- docker volume rm <volume-name>
- docker volume prune (it will remove all unused docker volumes).
- docker volume inspect <volume-name>
- docker container inspect <container-name>

Docker Hub:

Docker Hub is a registry service on the cloud that allows you to download Docker images that are built by other communities. You can also upload your own Docker built images to Docker hub.

It is a full GUI interface that allows us to manage builds, images, swarms, nodes and apps.

We need to have Docker ID to access and control images. If we don't have, create it first.

KUBERNETES:

KUBERNETES

- Kubernetes is an Open-source container management tool which Automates Container Deployment, Container Scaling and Load Balancing.
- It schedules, runs and manages isolated containers which are running on Virtual/Physical/cloud machines.
- It is supported by all the Cloud providers.

ONLINE PLATFORMS for K8s

- Kubernetes playground.
- Kubernetes with K8s
- Kubernetes with Kubernetes classroom.

INSTALLATION TOOLS

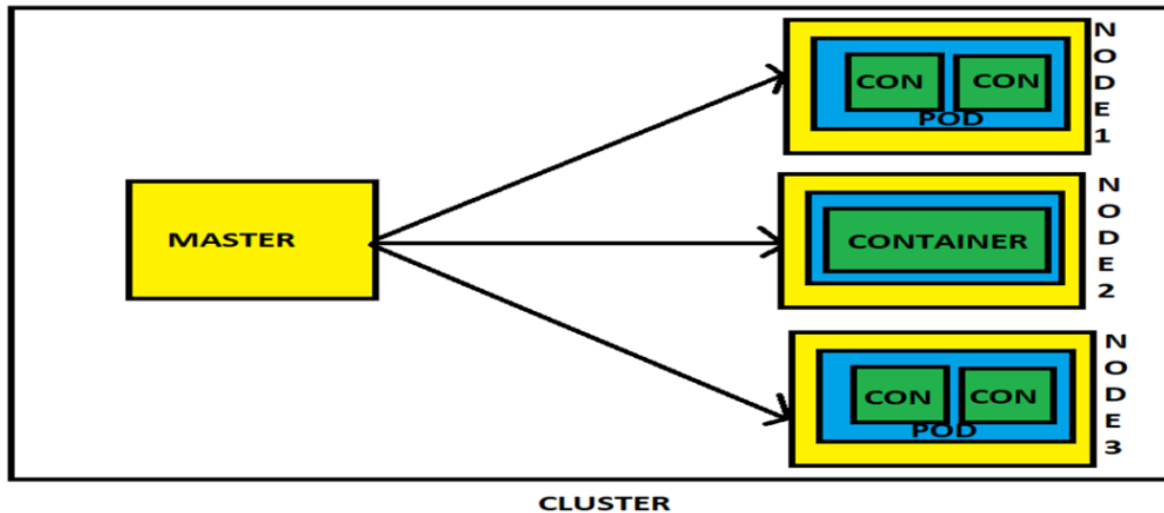
- Mini-Kube
- Kube-adm

CONTAINER SCALEUP PROBLEMS

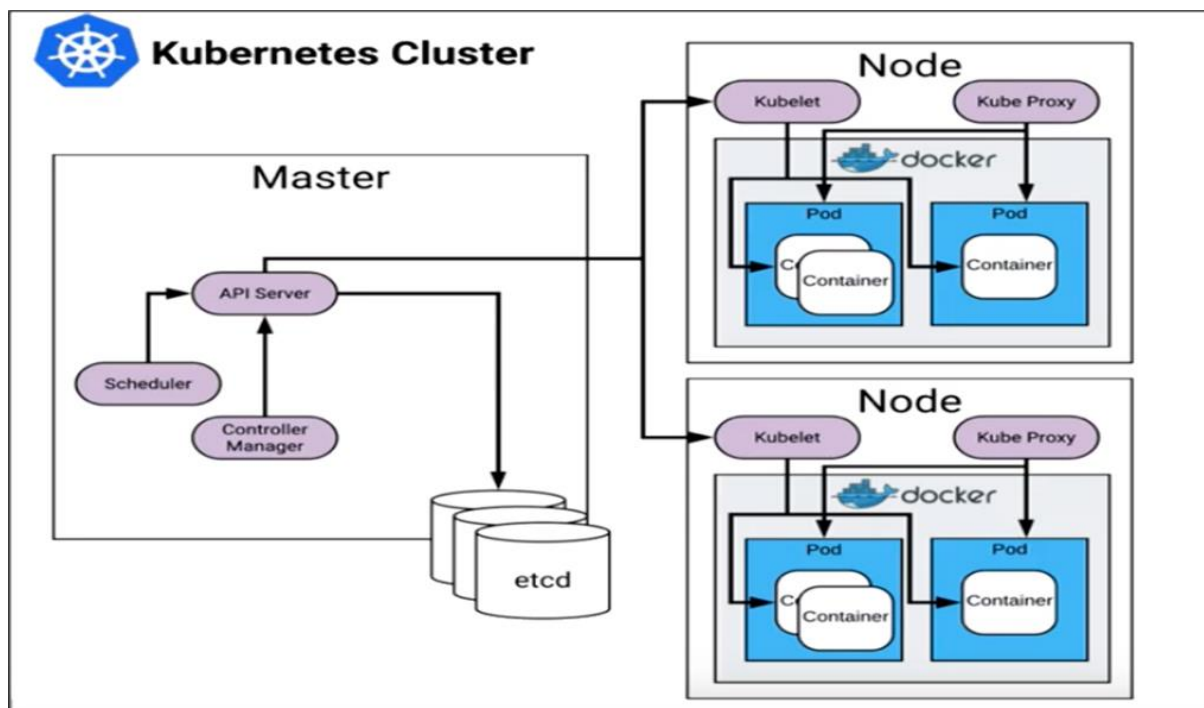
- Containers cannot communicate with each other.
- Auto scaling and Load balancing was not possible.
- Containers had to be managed carefully.

FEATURES

- Orchestration (Clustering of any number of containers running on different networks).
- Auto-scaling (Vertical [Existing] – >most Preferable and Horizontal [New]) and Auto-healing.
- Load balancing.
- Platform Independent (Cloud/Virtual/Physical).
- Fault tolerance (Node/Pod failure).
- Roll back (Going back to previous version).
- Health Monitoring of Containers. If one Container fails it will create another container.
- Batch Execution (One time, Sequential, Parallel).
- Scripts in K8s is called Manifest, which is in form of Json or YAML.



ARCHITECTURE



Master components

- **API Server:** Exposes the API.
- **ETCD Cluster:** Key value stores all cluster data. (Can be run on the same server as a master node or on a dedicated cluster.)
- **Kube-scheduler:** Schedules new pods on worker nodes.
- **Kube-controller-manager:** Runs the controllers.
- **Cloud-controller-manager:** Talks to cloud providers.

Node components

- **Kube-proxy:** Keeps network rules like Addressing IP to Pods.
- **Kubelet:** Agent that ensures containers in a pod are running.
- **Container engine:** Maintains the containers like Docker, Rocket etc.

Pod: A group of one or more containers.

Service: An abstraction that defines a logical set of pods as well as the policy for accessing them.

Volume: An abstraction that lets us persist data. (This is necessary because containers are ephemeral—meaning data is deleted when the container is deleted.)

Namespace: A segment of the cluster dedicated to a certain purpose, for example a certain project or team of developers.

Replica-Set (RS): Ensures that desired amount of pod is what's running.

Deployment: Offers declarative updates for pods an RS.

Stateful-Set: A workload API object that manages stateful applications, such as databases.

Daemon-Set: Ensures that all or some worker nodes run a copy of a pod. This is useful for daemon applications like fluentd.

Job: Creates one or more pods, runs a certain task(s) to completion, then deletes the pod(s).

MINI KUBE:

Minikube creates a single node cluster inside a VM or Cloud Instance. It is good for beginners to learn Kubernetes since you don't have to create a master and worker node to create a cluster and we can practice basic Kubernetes functions and can also install the Kubernetes dashboard on it.

PRE-REQUISTES

- Minimum 2 CPU's or more
- Minimum 2GB of free memory
- Minimum 20GB of free disk space
- Internet connection
- Container or virtual machine manager, such as: Docker, Hyperkit, Hyper-V, KVM, Parallels, Podman, VirtualBox, or VMware Fusion/Workstation

