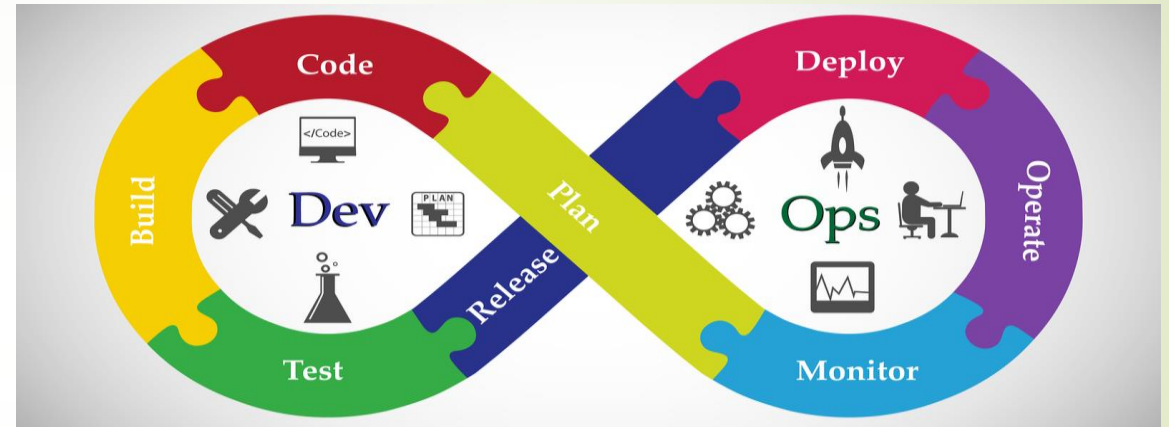


# DevOps

## ■ Course Content

- 1→Devops-Introduction
- 2→GIT
- 3→Maven
- 4→Jenkins
- 5→Chef
- 6→Ansible
- 7→Docker
- 8→Nagios
- 9→Kubernetes
- 10→Terraform
- 11→Helm
- 12-DevSecOps-Introduction



*Written by- Ashok Anupam*

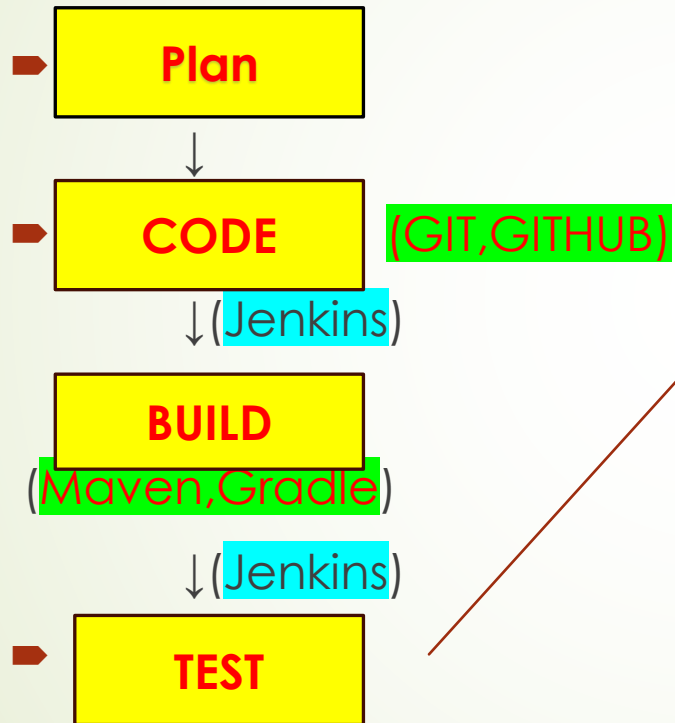
# DevOps-Introduction

- DevOps is a methodology or Practice where a single team performs the task of entire Software Development Lifecycle(SDLC).
- Main objective of DevOps is Implementing automation at every stage.
- Stages include:-
  - Version control(performed by GIT).
  - Continuous Integration(performed by Jenkins).
  - Continuous Deployment(performed by Docker).
  - Continuous Delivery(performed by Kubernetes,Ansible,Chef)
  - Continuous Monitoring(performed by Nagios)

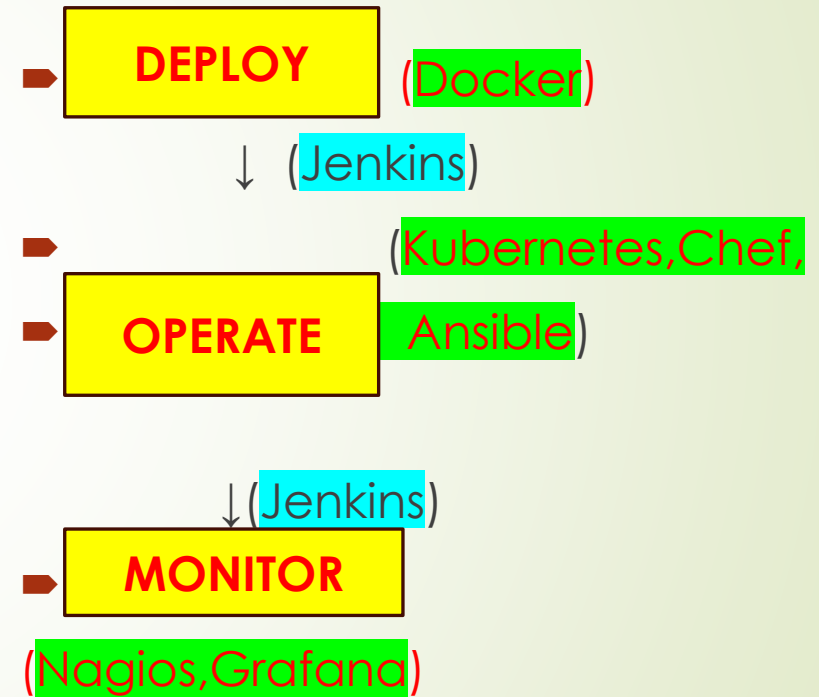
# Why The Need For DevOps

- To overcome the short-comings of waterfall model and Agile model, DevOps was introduced.
- Waterfall model took lot of time to complete project, almost 5-10 years, also requirements once decided, could not be rolled back.
- In Agile model ,work happens fast ,it's done in sprints but here also gap between Development and operations Team called Silos which results in hostile relationships between departments, which negatively impacts efficiency and harms the bottom line.
- In DevOps ,both Development and operations Team, work together and give better results.

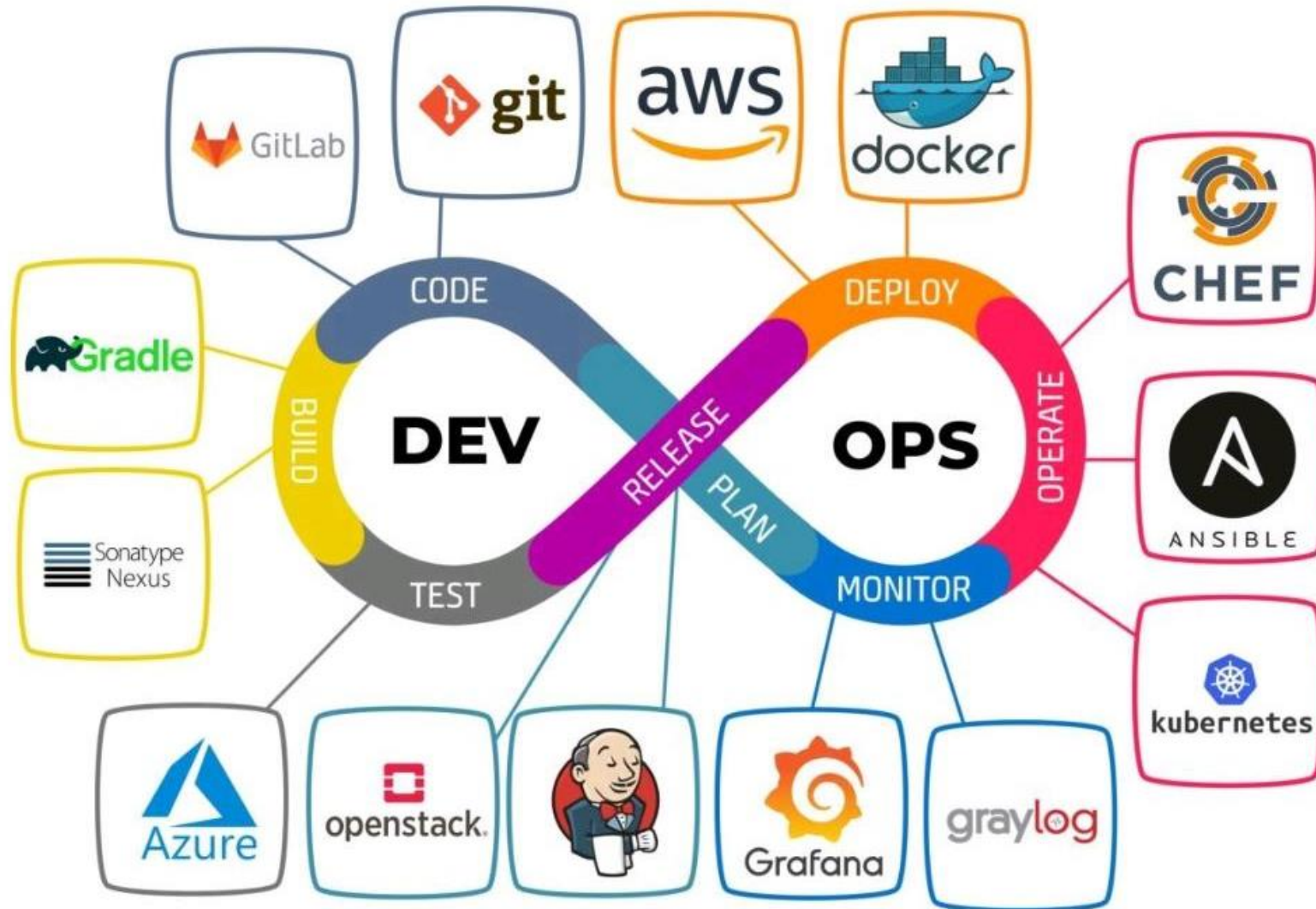
## Dev



## Ops



Plan→Code→Build→Test→Release→Deploy→Operate→Monitor



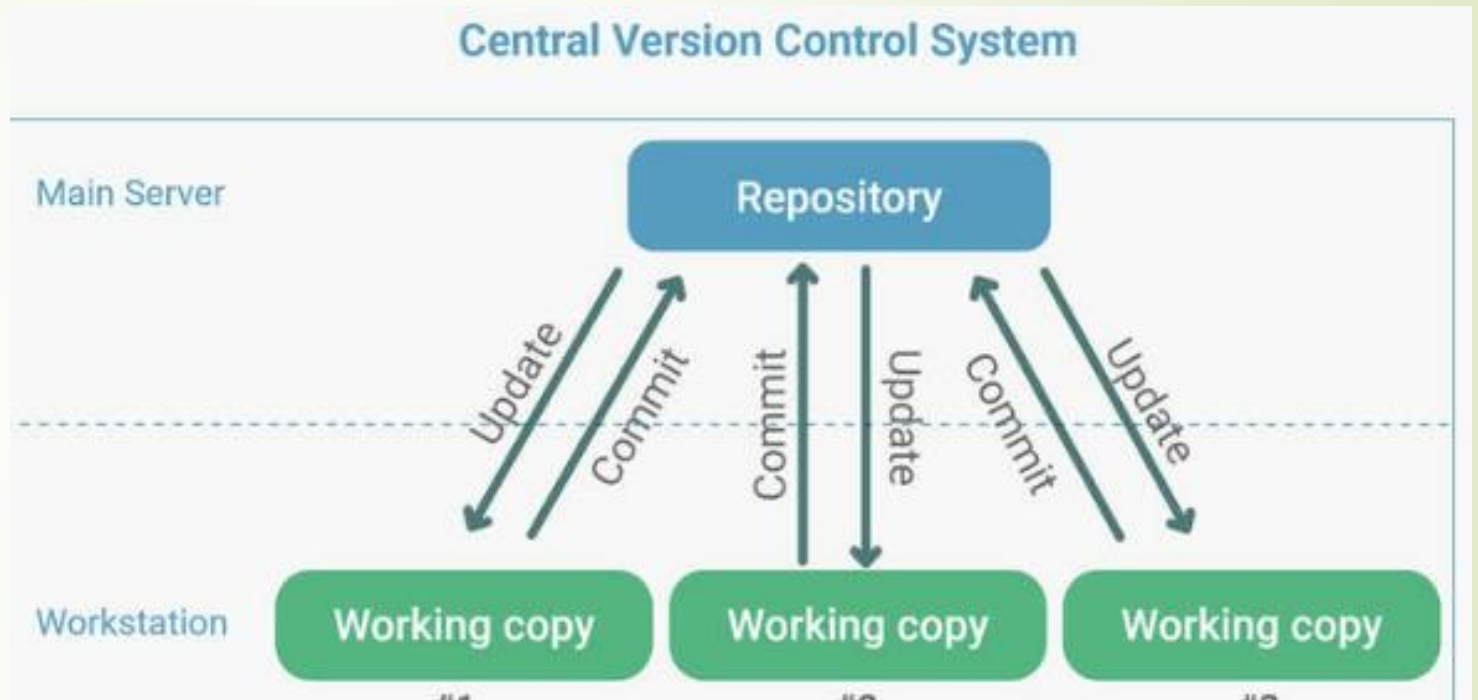
***GIT - A Distributed Version Control  
System***





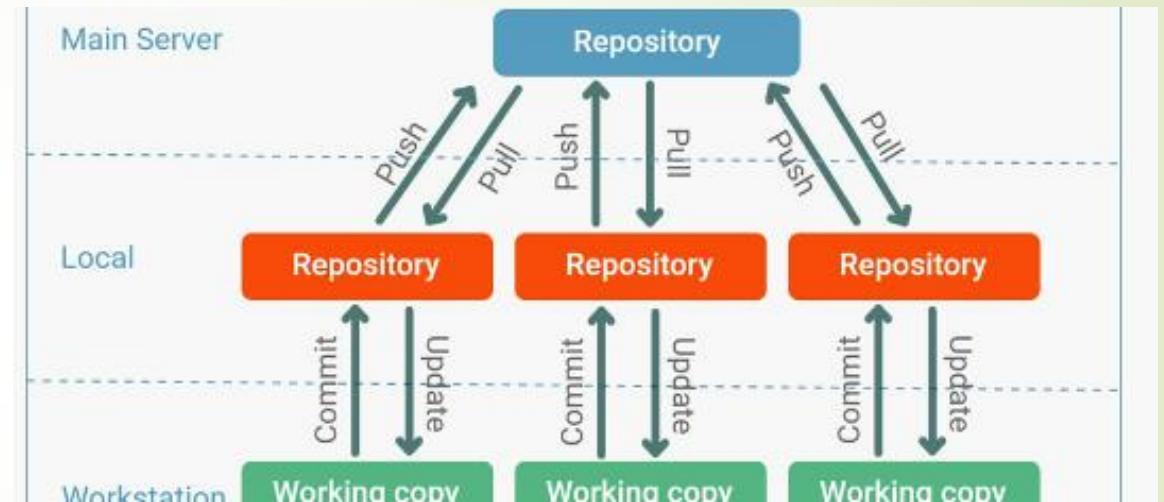
# Central Version Control system

- First Developer writes code and uploads it to Central repository.
- Second Developer will pull the code ,make changes and will commit and upload code.
- Similarly rest Developers will do.
- Commands are easy to learn.
- Single Copy of code is maintained here.
- Difficult to work in Branches.
- Merge Conflicts can occur.
- Complete Dependency on Internet, if Internet slow ,it will impact working.
- If Central Repository goes down, everything goes down.
- Examples are SVN Tool.



# Distributed Version Control System-GIT

- Here, each Developer will maintain a local copy of the code in his machine.
- First Developer will write code, first commit it to local Repository and then push to Central Repository.
- Second Developer will pull the entire code in his local repository first, Commit changes in local repository and then push to Central repository.
- Similarly rest Developers will do.
- Hence each Developer will have two copy of codes one at Local and other at Central Repository.
- Here, working in Branches is easy.
- Merge conflict don't occur, if occur, can be resolved easily.
- Helpful for parallel Development.
- Difficult commands compared to Central Version Control system.
- Works fast as Local copy maintained.
- Example is GIT





# ***GIT-In-Action***

- Git has three stages of workflow
- 1. Working area 2. Staging area 3. Local Repository.
- We send data or code from working area to staging area by add command and staging area to Local repository by commit command and finally send data/code from Local repo to central repo by push command.
- Update Linux operating system in working area (Mumbai Ec2-user)
- `# yum update -y`
- `# yum install git -y`
- `# which git`
- `User /bin /git`
- `# git --version`
- `2.40.1`
- `# git config --global user.name "Ashok"`
- `# git config --global user.email ashok.anupam465@gmail.com`
- `# git config --list` (this command shows the all configured list)

# ***Git-In-Action***

- Now work inside the Mumbai machine, create Directory and make file inside localrepo
- [Ec2-user] # **mkdir mumbaigit**
- [Ec2-user] # **cd mumbaigit**
- [Mumbaigit]# **git init** ( init command turn Dir into local Repo)
- [Mumbaigit]# **cat >Mumbai1** (write inside the [Dir] local repo by cat > command)
- # **cat Mumbai 1** (to check the data/code what has been written in repo)
- Put and write some code/data inside the file mumbai1, and come out by Ctrl+d
- I Love India
- # **git status**
- Untracked files: **Mumbai1** (it's in red color means not added yet staging aria)
- [mumbaigit]# **git add .** (Add command to add created file to staging aria)
- # **git status**
- New file: **Mumbai1** (it's in green color means added staging area )

# GIT-In-Action

- Now commit data from staging area to Local repo
- [Mumbaigit]# **git commit -m "first commit from Mumbai"** (m=message)
- **# git status**
- **# git log** (to check what commits have been done, when and who did?)
- You will see commit Id like 12345678KD458F4IW3E4 .Author, Mail id, Date, Time, message: first commit from Mumbai
- [Mumbaigit] **# git show** (show command the content of commit ID)
- first commit from Mumbai
- + I Love India
- If we run the git commit command again it will show nothing to commit, working tree clean means data has been committed.
- If want to send this code to my central repository,
- I have to connect local repo to central repo first, for this action I have to create a new repository (any name) and paste the URL of git repo and execute command as given below
- [Mumbaigit] **# git remote add origin <https://github.com/ashok1012/centralgit.git>**
- Now local repo has been connected to central repo, for pushing data to central repo execute this command
- [Mumbaigit] **# git push -u origin master (push command local repo to central repo)**
- It will ask for username and password of your git hub account, after filling this and you can see all committed data/code inside central repo but now tokens being asked.

# ***GIT-In-Action***

- Now create a machine in Singapore region and connect to git hub.
- **[ec2-user] # yum update -y**
- **# yum install git -y**
- **# git config --global username**
- **# git config --global user.name "Alok"**
- **# git config --global user.email alok.anupam26@gmail.com User.name=Alok User.email=[alok.anupam26@gmail.com](mailto:alok.anupam26@gmail.com)**
- **[Ec2-user] # mkdir singaporegit**
- **# cd singaporegit**
- **[singaporgit] # git init** (Initialized empty git repository in Home/ec2- user/singaporegit/.git/)
- **[singaporgit]# ls -a**
- **. . . .git**
- **[singaporgit] # git remote add origin <https://github.com/ashok1012/centralgit.git>**
- Now local repo has been connected to central repo, for Pulling data to central repo, execute this command
- **[singaporgit]# git pull -u origin master** (you can execute without -u as well)
- Now you can see it has pulled all data/code from remote directory central repo, all details and commits has been done by other Mumbai machine.

# ***GIT-In-Action***

- [singaporgit]# cat >mumbai1 (> used to write and overwrite code inside mumbai1) I love Bengaluru too  
Ctrl+D
- If you want to add lines or something on this code inside the file use command
- # cat >>file
- # git status
- Modified: mumbai1
- # git add .
- # git status
- Modified: mumbai1
- # git commit -m "first commit from singapore"
- # git log
- Now it will show all messages commits ids and steps done by both Mumbai and Singapore machines
- # git show 12345678KD458F4IW3E4
- I Love India Old commit
- I Love Bengaluru new commit
- Push data/code into central git from local repo
- # git push -u origin master (you can use -f instead of -u for force push)
- Now Enter username and password of git hub account, after that you will see all new and old commits updates in central git, click mumbai1 file you will get code "I Love Bengaluru"



# GIT-IGNORE

- -This command is used to ignore some specific file which we don't want to add & commit.
- [mumbaigi1]# **vi .gitignore**
- \* CSS \* used to ignore particular file
- \* java Esc+:wq
- # **git add .gitignore**
- # **git commit -m "ignore file format"** can use single comma as well
- # **git status**
- Nothing to commit, working tree is clean, now create some files in different formats by using touch command
- # **touch file1.txt file4.java file3.css file5.java file2.txt**
- # **ls**
- # **git status** File1.txt only showing 2 untracked files **rest three have been ignored** File2.txt
- # **git add .**
- # **git status** Now both files have been added and showing us in green color after status command File1.txt File2.txt
- # **git commit -m "IGNORE JAVA CSS FILES"**
- # **git log 12345678KD458F4IW3E4** (HEAD -> master) So many commit ids are showing 12345678KD458F4IW3E4 (HEAD -> master) 12345678KD458F4IW3E4 (HEAD -> master)
- # **git show 12345678KD458F4IW3E4** IGNORE JAVA CSS FILES

# GIT-IGNORE

- `# touch ashok.java`
- `# git status`
- Nothing to commit, working tree is clean, now create some files in different formats by using touch command
- `# touch Alok.txt`
- `# git status`
- Alok.txt (Again it showed text file, not java file means ignored)
- If I want to see latest commit, last 2 commits, last-n commits and all commits in one line.
- `# git log -1`
- `# git log -2`
- `# git log --oneline`
- 12345678KD458F4IW3E4 (HEAD -> master) message "1" So many commits are showing in one column
- 12345678KD458F4IW3E4 message "2"
- 12345678KD458F4IW3E4 message "3"
- If I want to find specific commit, Action and file use grep command with specific name rest will be ignored. [mumbaigi1]
- `# git log --grep "ignore"`
- ignore=ignore css and java files

# ***GIT BRANCHES:***

- Each task has one separate branches, after done with code other branches merge with master.
- This concept is useful for parallel development. Master branch is default branch
- We make branches, one for little features and other one for longer running features.
- It keeps the main master branch free from error.
- Files created in workspace will be visible in any of the branch workspace, until you commit,
- once you commit then those files belong to that particular branch
- How to create Branches:
- [ec2-user] # **cd mumbaigit**
- **[mumbaigit] # git log --oneline**
- **# git branch**
- **\*master**
- **# git branch branch1**
- **# git branch**
- **\*master**
- **Branch1**
- **# git checkout branch1** (switch to branch branch1)
- **Master**
- **\*Branch1**
- **# git branch -d** (to delete any branch)

# GIT BRANCHES:

- Branches Working process:
- # **git checkout branch1**
- # **cat >branch1file** (create branch1file and write anything inside by >)
- Something is better than Nothing
- # **ls**
- mumbai1
- # **git checkout master**
- # **ls**
- mumbai1 branch1
- branch1file and code is showing inside master branch because it hasn't committed with any branch yet.
- # **git commit -m "branch1 first commit"**
- # **git log --oneline**
- Branch1 first commit
- # **git checkout master**
- # **git log --oneline**
- branch1file & code will not show inside master branch because that file has been committed with Branch1.
- How to Merge Branches: we use pulling mechanism, we can't merge branches of different repositories
- # **git checkout master**
- # **git merge branchA** (to verify the merge)
- Executed checkout command before merge command means, you wanted to merge any branch with master branch
- # **git log --oneleine**



# ***GIT BRANCHES:***

- Now you can see All commits of both branches which have been merged together
- **# ls**
- Now you can see All files of both branches which have been merged together.
- **# git push origin master** (to push central repo lit git hub)
- Enter username & password you can see merged data in central repository on git hub.



# GIT CONFLICT:

- When same files having different content in different branches, if you do merge ,conflict can occur. (Resolve conflict then add and commit)
- # **Cat >anupfile**
- hello anup <ctrl+d >
- # **git add .**
- # **git commit -m "commit before conflict"**
- # **git checkout branch1** (switch to branch1 )
- # **Cat >anupfile**
- create same file but write different code inside
- hello Ashok <ctrl+d>
- # **git commit -m "commit from branch1"**
- # **git checkout master** (switch to Master)
- # **git merge branch1** Merge failed: fix conflict, then commit result
- # **vi anupfile** (update inside anupfile)

# GIT CONFLICT:

```
<<<<<<HEAD delete HEAD
```

```
Hello anup
```

```
===== delete =====
```

```
Hello Ashok
```

```
>>>>>> branch1 Esc+:wq
```

- We can change data according to ourself which we exactly needed before conflict do changes in file git will understand the change and execute data accordingly.
- # **git status**
- # **git add .**
- # **git commit -m "Resolve conflict"**
- # **git log --oneline 12h3a8g90 (HEAD □ master) Resolve conflict**

# GIT BRANCH STASH

- If our code is in progress and suddenly need changes through client escalation, we have to keep aside current code and have to work on new features for some hours.
- we can't commit your parallel code, so we need some temporary storage to store partial changes and later on commit it.
- To stash an item only applied for modifies files not new files.
- # **git checkout master**
- # **cat >anupamfile1**
- # **git commit -m "anupamfile1 commit"**
- # **vi anupamfile1**
- Boss asks to do some other work
- # **git stash**
- # **Cat anupamfile1** (anupamfile1 empty, data stashed ,now you can do new work)
- # **Git stash list**
- Stash (0) : WIP on master 1372ee7 .anupamfile1
- # **vi anupamfile1** My super anupam code-2 Esc+:wq
- # **cat anupamfile1** My super anupamfil2 code-2

# GIT BRANCH STASH

- `# git stash`
- `# git stash list`
- `Stash (0)`
- `Stash (1)`
- `# cat anupamfile1`
- `(anupamfile1 empty, data/code has been stashed)` Now going to do old pending work
- `# git stash apply stash@{1}`
- `# cat anupamfile1`
- `My super anupam code-2`
- `# git add.`
- `# git commit -m "anupamfile1 commit done"`
- `# git stash apply stash@{0}`
- Auto merging anupamfile1; CONFLICT:Merge conflict in

# GIT BRANCH STASH

- # Vi anupamfile1
- (update inside anupamfile1)
- <<<<< update stream My super anupam-1 final code would be my super anupam-2
- ===== delete =====
- My super anupamcode-2
- >>>> stashed changes Esc+:wq
- # git add .
- # git commit -m "anupamfile1 commit done2"
- # git status (empty)
- # git log --oneline anupamfile commit anupamfile xommit done anupamfile commit done2
- # git stash list
- Stash (0)
- Stash (1) (still available in stash list delete it by # git stash clear, recheck by stash list command)



# GIT RESET:

- It is a powerful command that is used to undo local changes to the state of a git repository.
- It used to undo the add . command.
- # `cat >testfile`
- Hello Ashok
- # `git add .`
- # `git status`
- New file:testfile (now realized did mistake in data, wanted to change)
- • • To reset from staging aria
- # `git reset testfile`
- # `git reset .` (removed data from staging aria)
- # `git status`
- `testfile`
- # `git add .`
- # `git status testfile`
- • • To reset from staging area
- # `git reset -hard`
- # `git status`
- One branch master nothing to commit: working tree clean

# GIT REVERT:

- Revert command helps you to undo the existing commit, it doesn't delete any data, instead git creates a new commit with the included previous files reverted to the previous state.
- So, history moves forward while the state of your file moves backward.
- # **cat >checkfile**
- I LOVE MY INDIA
- # **git add.**
- # **git commit -m "checkfile1 commit" after commit I realized did wrong commit**
- # **git log --oneline**
- Now you can see so many commits copy previous commit id just before the mistake and paste on revert command
- # **git revert 12h3a8g90**
- Wrong commit undo state moves to backward also write a message in this commit "please ignore previous commit"
- How to remove untracked files
- # **git clean -n dry run**
- # **git clean -f forcefully**

# Git Tags

- Tag operation allows giving meaningful name to a specific version in the repository.
- To Apply Tag
- # **git tag -a -m "message" commit-id**
- # **git tag -a Ashok -m "love you India" 12h3a8g90g6k**
- To see tag # **git tag**
- # **git show tag-name** to see particular commit content by using Tag
- To delete a tag
- # **git tag -d tag-name**

# Git Hub Clone:

- go to existing repo in git Hub copy the URL of central repo and paste with run command of Linux machine.
- # **git clone**
- [ec2-user] # **git clone** <https://github.com/ashok1012/centralgit.git>
- It creates a local repo automatically inside Linux machine with the same name of git hub account.
- # **ls Mumbai1**
- **git Anupamfile centralgit node1git**
- Both repositories can be connected together easily by master branch



*Maven- A project Management tool*



# Maven-Introduction

- Maven is an automation and project management tool developed by apache software foundation.
- It is based on POM (project object model).
- Maven can build any number of projects into desired output such as .jar, .war, metadata.
- Mostly used for java-based project.
- It was initially released on 13th July 2004.
- Maven is written in java language.
- Meaning of maven is 'accumulator of knowledge'.
- Maven helps to get the right .jar file for each project as there may be different version of separate packages.
- To download dependencies, it is no more needed to visit the official website of each software. It could now be easily done by visiting 'mvnrepository.com'
- **Dependencies:** it refers to the java libraries that are needed for the project.
- **Repositories:** it refers to the directories of packaged .jar files.
- **Build tools:** C, C++: make file .Net: visual studio Java: Ant, Maven, Gradle

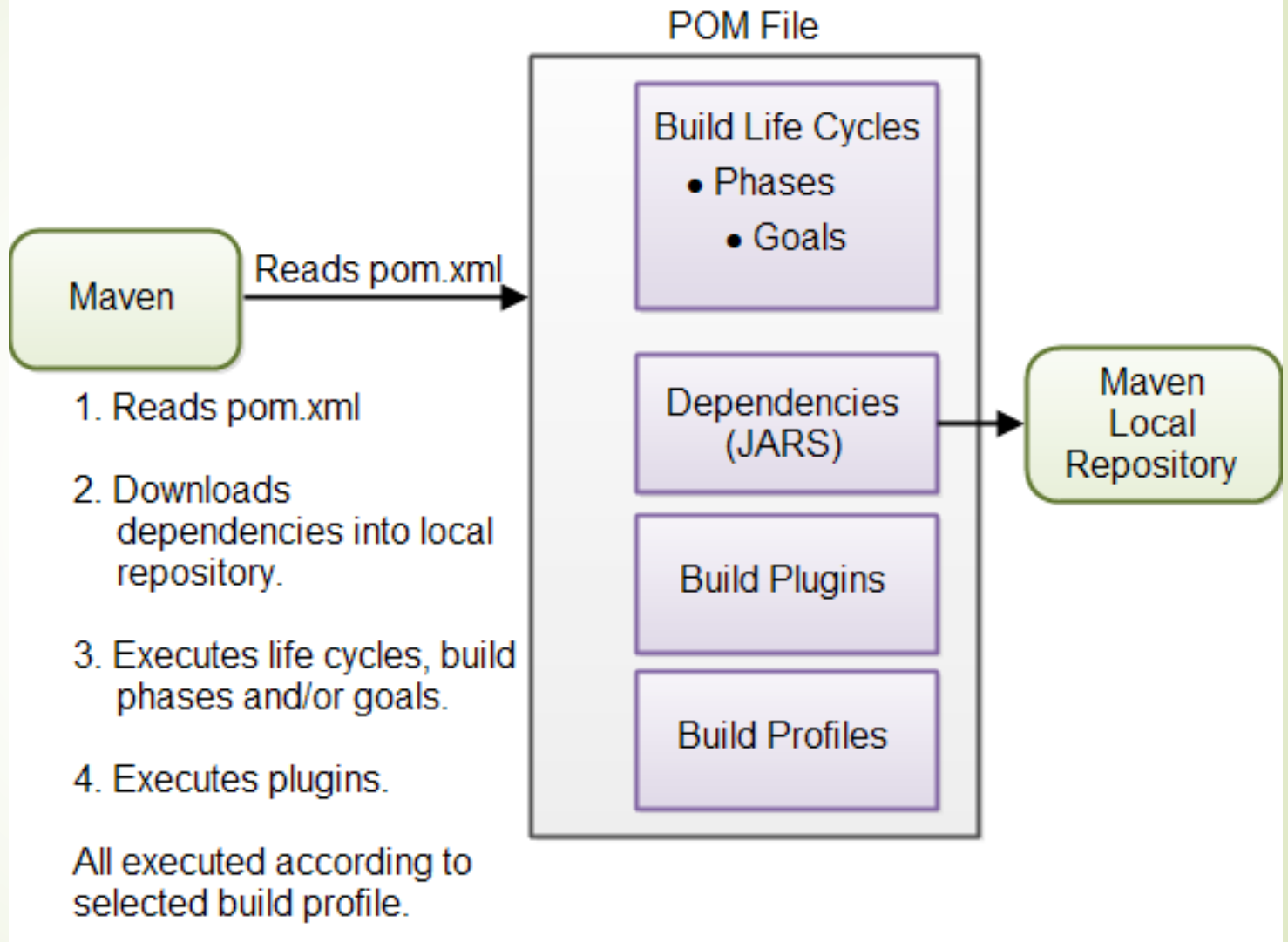
# Problems without Maven:

- 1. Adding set of jar in each project:
  - in case if struts, spring we need to add jar files in each project.
  - It must include all the dependencies of jars also.
- 2. Creating the right project structure: we must create the right project structure in servlet, struts etc, otherwise it will not be executed.
- 3. Building and deploying the project: we must have to build and deploy the project so that it may work.
- **What Maven does?**
  - 1. It makes a project easy to build.
  - 2. It provides project information (for e.g: log documents, crossreference sources, mailing list, dependencies list, unit testing)
  - 3. Easy to add new dependencies, therefore apache maven helps to manage
    - a. Build
    - b. Dependencies
    - c. Reports
    - d. Release
    - e. Distribution

# What is Build Tool:

- A build tool takes care of everything for building a process.
- It does following:
  - Generate source code.
  - Generate documentation from source code.
  - Compiles source code.
  - Install the package code in local repo., server repo. Or central repo
- **POM (project object model):**
  - POM refers to the .xml file that have all the information regarding project and configuration details.
  - Main configuration file in pom.xml.
  - It has the description of the project details regarding the version and configuration management of the project.
  - The .xml file is in the project home directory.
  - Pom.xml contains: • Metadata • Dependencies • Kind of project • Kind of output (.jar, .war) • Description One project – one workspace – one pom.xml file

# Maven- Architecture



# Requirement for Build

- Source code (present in workspace)
- Compiler (remote repo – local repo – work space)
- Dependencies (remote repo – local repo – work space)
- **Local Repository**: local repo. refers to the machine of the developer where all the project material is saved.
- **Remote Repository**: it refers to the repo. present on a webserver which is used when maven needs to download dependencies. This repo. works same as the central repo. whenever anything is needed from remote repo. it is first downloaded to the local repo. and then it is used.
- **Central Repository**: central repo. refers to the maven community that comes into action when there is a need of dependencies and those dependencies are not found in the local repo.

# Maven Build Life-Cycle

- Goals:
- 1. Generate resources (dependencies)
- 2. Compile code
- 3. Unit test
- 4. Package (build 5. Install (into local repo. and artifactory)
- 6. Deploy (to server)
- 7. Clean (delete all run time files)
- e.g- mvn install
- mvn clean package
- 1 to 6 – default and sequence order
- 7 – not default and it won't allow sequence
- Build life cycle consists of a sequence of build phases and each build phase consists of a sequence of goals.
- Each goal is responsible for a particular task.
- When a phase is run all the goals related to that phase and its plugins are also compiled. e.g- mvn install
- mvn clean package



# ANT vs MAVEN

- Ant doesn't have formal conventions, so we need to provide information of the project structure in build.xml file.
- ➤ Ant is procedural, you need to provide info about what to do and when to do through code.
- ➤ There is no lifecycle in ant.
- ➤ It is a tool box.
- ➤ It is mainly a build tool.
- ➤ It is less preferred than maven.

- Maven have a convention to place source code, compiled code etc. so we don't need to provide information about the project structure in pom.xml file.
- ➤ Maven is declarative, everything you define in the pom.xml file.
- ➤ There is a life cycle in maven.
  - It is a framework.
- ➤ It is mainly a project management tool.
- ➤ It is more preferred

# *Jenkins*


Jenkins-An Integration Tool






# ***JENKINS-Introduction***

- Jenkins is an open-source project written in Java that runs on windows, macOS and other unix like os. It is free, community supported and might be your first-choice tool for CI.
- Jenkins can automate the entire s/w development life cycle.
- Jenkins was originally developed by sun microsystem in 2004 under the name hadson.
- The project was later named Jenkins when oracle bought microsystem.
- It can run on any major platform without any compatibility issues.
- Whenever developer write code, we integrate all that code of all developers at that point of time and we build test and deliver/ deploy to the client. This process is called as CI/CD.
- Jenkins helps us to achieve this.
- Because of CI now bugs will be reported fast and get rectified fast. So that entire software development happens fast.



## *Workflow of Jenkins:*

- We can attach git, maven, selenium and artifactory plugins to Jenkins.
- Once developers put codes in github Jenkins pull that code and send to maven for build.
- Once build is done, then Jenkins will pull that code and send to artifactory as per requirement and so on.
- We can also deploy with Jenkins.



## *Advantages of Jenkins:*

- It has lots of plugins available.
- You can write your own plugins.
- You can use community plugins.
- Jenkins is not just a tool. It is a framework i.e you can do whatever you want. All you need is plugins.
- We can attach slaves (nodes) to jenkins master. It instructs other (slaves) to do job. If slaves are not available, Jenkins itself does the job.
- ➤ Jenkins also behave as cron server replacement i.e can do scheduled task.
- ➤ It can create labels.



## *CI/CD Project:*

- Go to google chrome – search ‘git download’ – **download 2.44.0** for windows – click to download. Open the download file preamble
- – c:\programfiles\Git
- – select components
- – select start menu folder
- – choosing default editor vim
- – let git decide
- – git from the command line and also from the 3rd party s/w
- – use the open SSL library
- – check as-is, commit Unix-style line endings
- – use minty
- – choose default behavior
- – git credentials manager core
- – enable file system caching
- - enable experimental built-in file system monitor
- – install.





## *Download and Installation of JDK21:*

- Go to google chrome – search ‘java development kit download’ – **download java SE development kit 21.0.2 for windows x64.**
- Run and follow the steps to install.
- Now go to C drive – program files – java jdk21 – select path and copy it
- Search ‘edit system environment variables’ in laptop – go to use variables – new
- Variable name- JAVA\_HOME
- Variable value- paste the path here
- Now go to system variable – new
- Variable name – JAVA\_HOME
- Variable value- paste the path here
- Now go inside program files – bin – copy the path
- Again, go to ‘edit system variables’ – system variable – path – new – paste path
- Now verify in command prompt
- C:\users\home\echo% JAVA\_HOME%
- o/p- c:\ Program Files\Java\jdk-21

# *Maven download and install:*

- Go to google chrome – search maven.apache.org – [download Apache Maven 3.9.6– binary zip archive](#)
- Extract files – c:\Devtools
- Go to c:\ - Devtools – apache-maven – copy the path
- Now search ‘edit system environmental variables’ – system variable – new
- Variable name – M2\_HOME
- Variable value - paste the path here
- Now go inside apache maven folder – bin – copy path
- Now again go to ‘environmental variable’ – system variable – path – new – paste path
- Now open command prompt
- C:\users\home > mvn -version
- C:\users\home > echo % M2\_HOME%
- Now restart the laptop

# *Jenkins download and install:*

- Go to google chrome – Jenkins.io – download – select LTS – windows – download
- Open download file – run and install
- After installation it automatically open as local host: 8080
- Unlock the page by using password
- Now install suggested plugins
- Ask for username and password
- Username – admin
- Password – admin123
- Email address- ashok.anupam465@gmail.com
- Save and continue
- Start using Jenkins
- Plugins: plugins are small libraries that add new abilities to Jenkins and can provide integration points to other tools.
- Go to google chrome – localhost:8080 – login



## Manage Jenkins

- Go to google chrome – localhost:8080 – login
- Go to manage Jenkins on left side of Jenkins dashboard – manage plugins – available – select maven integration and green balls – install without restart
- Go to new item – maven project
- Now go to manage Jenkins – global tool configuration
- Go to add JDK
- Uncheck the install automatically option
- NAME – JAVA
- JAVA\_HOME – c:\programfiles\java\jdk
- Now go to maven
- Name – MAVEN
- MAVEN\_HOME – c:\devtools\apache-maven

# *Maven Project (by maven):*

- Go to <https://github.com/technicalguftgu/time-tracker>
- Click on time tracker repo
- 'fork' to copy this repo
- Sign-in into your github account
- Click on time-tracker repo
- Clone
- Go to c drive
- Git clone <url of time-tracker repo>
- Cd time-tracker
- C:\time-tracker > maven clean package
- **Maven Project (by Jenkins):**
- Now go to Jenkins – new item – entername -> mymavenproject
- Then select maven project – ok
- Source code management – git – repo url
- Build option – root POM – pom.xml
- Goals and options – clean package – save
- Go to Jenkins home page – click on mymavenproject – build now



## **Scheduled Project: and Source Code Polling (Poll SCM):**

- Click on any project – configure – build triggers – build periodically - \* \* \* \* \* - save
- Can see automatic builds after every 1 min.
- You can manually trigger build as well.
- **Source Code Polling (Poll SCM):**
- Now go to Jenkins home page – go to mymavenproject – configure
- Now go to build trigger
- Enable poll SCM
- Schedule \* \* \* \* \* - save
- Now go to github account – do some changes in README.md – commit changes.
- You can see after 1 min, it builds automatically.





# *User Management:*

- go to Jenkins homepage – manage Jenkins – manage user
- create two users – Ashok and Anupam
- now login as Ashok
- {by default you have all the permissions}
- Login as ‘admin’ again
- Go to manage Jenkins – manage plugins – search ‘role-based authorization strategy’ – install without restart.
- Go to Jenkins homepage – manage Jenkins – configure global security – select role-based strategy – save.
- Login as ‘Ashok’ – access denied
- Now attach permission
- Go to Jenkins – manage Jenkins – manage and assign role – manage roles
- Role to add – employee
- Go to item project – add developer and tester [pattern – dev\* test\*]
- Then assign roles
- User/group to add Ashok and Anupam.

## ***How to install Jenkins on Ubuntu:***

- Login into AWS account and create one ubuntu instance.
- ➤ Access it through putty and login as 'ubuntu'.
- ➤ Use given commands
- # sudo apt-get update
- # sudo apt-cache search openjdk
- #sudo apt-get install openjdk-8-jdk
- #java -version
- #sudo vi /etc/apt/source.list
- Paste at the bottom
- deb https://pkg.jenkins.io/debian-stablebinary/
- #sudo apt-get update
- # wget -q -o ---
- # sudo apt-cache search Jenkins
- # sudo apt-cache Madison Jenkins
- # sudo apt-get install Jenkins -y
- # sudo service Jenkins status -q
- Copy public ip from AWS and paste in chrome url – public:8080
- Go to instance #sudo cat /var/lib;initialpassword



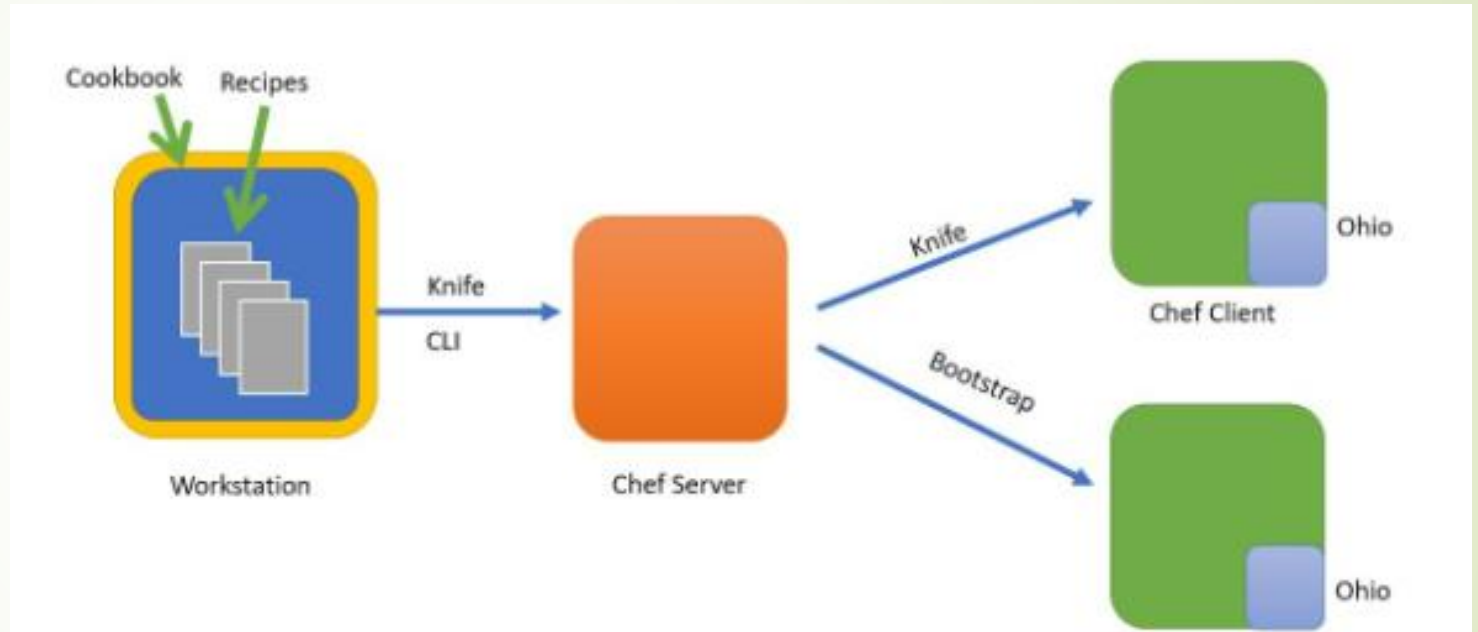
*Chef-A pull based Configuration  
Management Tool*

# CHEF-Introduction

- It is a pull -based configuration management tool unlike Ansible which is push based in nature.
- It turns our code into Infrastructure hence, its Infrastructure As Code[IAC].
- Chef has following components.
- (a)-Workstation-They are personal machine or virtual servers , where configuration codes are created, Codes are called Recipes and collection of recipes is called cookbook.
- Workstation communicate with Chef Server through Knife.
- (b)Chef-Server-It is stored between Workstation and Node.
- All Cookbooks are stored here. Servers may be stored locally or remote.
- (c) Nodes-They are systems which require configuration.
- OHAI fetches current state of node.
- Node communicate with Chef Server with help of Chef-Client

# CHEF- Architecture

- **Workstation**- Where we write our code called Recipe.
- **Chef-Server**- Where we upload our Code.
- **Node**-Where we apply our code.
- **Knife**-Command Line Interface to communicate between Workstation and Chef-Server and between Chef-server to Chef-Client.
- **Ohai**-It contains Meta data about Node
- **Node** communicate with **Chef-server** through **Chef Client**.



# CHEF-Installation

- Download & install Chef and create Cookbook, Recipes
- ❖ Wget <chef download link> [[Install Chef Workstation for Linux](#)]
- ❖ mkdir cookbooks
- ❖ cd cookbooks/
- ❖ chef generate cookbook Ashok-cookbook
- ❖ cd Ashok-cookbook
- ❖ yum install tree -y
- ❖ tree
- ❖ chef generate recipe Ashok-recipe
- ❖ cd ..
- vi Ashok-cookbook/recipes/Ashok-recipe.rb
- file '/myfile' do
- content 'Welcome Ashok Anupam'
- action :create
- end
- enter+esc+:wq



# CHEF

- Chef exec ruby -c Ashok-cookbook/recipes/Ashok-recipe.rb (check the syntax)
- ❖ Syntax OK
- ❖ (run the chef client)
- ❖ Chef-client -zr "recipe[Ashok-cookbook::Ashok-recipe]"
- ❖ Cat /myfile(xyz) (also try ls /) (to check inside the file)
- Apache server:
- [cookbooks]#chef generate cookbook apache-cookbook
- #cd apache-cookbook
- #chef generate recipe apache-recipe

# CHEF

- `#cd ..`
- `#vi Apache-cookbook/recipes/apache-recipe.rb`

```
package 'httpd' do
```

- `action :install`
- `end`

- `file '/var/www/html/index.html' do`
- `content 'Welcome Ashok Anupam'`
- `action :create`
- `end`

- `service 'httpd' do`
- `action [:enable, :start]`
- `End`

- `#Chef-client -zr "recipe[Apache-cookbook::Apache-recipe]"`

- **Now ping public IP address to see content on apache website,First enable All Traffic in EC2-Instance else will not work.**

# ATTRIBUTES

- What is this: **Attributes is a key value pair which represent a specific detail about node.**
- **Who uses? Chef client**
- **Why uses? To determine**
  - current state of node
  - what was the state of the node at the end of previous chef client run?
  - What should be the state of the node at the end of current chef client will run?
- **Types:**

	<b>Priority</b>
1. Default	1 st maximum
2. Force-default	2 nd more
3. Normal	3 rd may be
4. Override	4 th less
5. Force override	5 th very less
6. Automatic	6th minimum



# ATTRIBUTES

- Who defines Attributes?
- Ans: (Node, Cookbooks, Roles, Environment)
- **\*\* (attribute defined by Ohai have highest priority)**
- # sudo su
- # ohai
- # ohai ipaddress
- # ohai memory/total
- # ohai cpu/0/mhz
- # ls
- # cd cookbooks
- # cd Apache-cookbook
- # Chef generate recipe recipe10
- # cd ..

# ATTRIBUTES

- **# vi apache-cookbook/recipes/recipe10.rb**
- **[+ Enter then ]**
- file '/basicinfo' do
- content "This is to get Attributes
- HOSTNAME: #{node['hostname']}
- IPADDRESS: #{node['ipaddress']}
- CPU: #{node['cpu']['0']['mhz']}
- MEMORY: #{node['memory']['total']}
- owner 'root'
- group 'root'
- action :create
- End
- **# chef exec ruby -c apache-cookbook/recipes/recipe10.rb**
- **# chef-client -zr "recipe[apache-cookbook::recipe10]" (call the client)**
- **SEE OUTPUT ATTRIBUTES**

# ATTRIBUTES

- Insert Linux commands [cookbooks]
- # **vi ABC-cookbook/recipes/ABC-recipe.rb**
- **i+ Enter then**
- #
- # Cookbook:: ABC-cookbook
- # Recipe:: ABC-recipe
- #
- # Copyright:: 2024, The Authors, All Rights Reserved.
- execute "run a script" do
- command <<-EOH
- mkdir /ashokdir
- touch /ashokdir/anupamfile
- EOH
- End
- [ABC-cookbook] ls /
- ashokdir



# ATTRIBUTES

- Create user
- `# vi ABC-cookbook/recipes/ABC-recipe.rb`
- `user "ashok" do`
- `action :create`
- `End`
- Enter+Esc+:wq
- (now run the recipe, call chef-client)
- `# chef-client -zr "recipe[ABC-cookbook::ABC-recipe]"`
- Create group
- `# vi ABC-cookbook/recipes/ABC-recipe.rb`
- `group "devopsgroup" do`
- `action :create`
- `members 'ashok'`
- `append true`
- `End`
- Enter+Esc+:wq (now check the recipe)
- `# Chef exec ruby -c ABC-cookbook/recipes/ABC-recipe.rb`
- `# syntax OK (now run the recipe)`
- `# chef-client -zr "recipe[ABC-cookbook::abc-recipe]"`
- `# cat /etc/group (also try ls /) (to check the group)`

# RUNLIST

- To run the recipe in a sequence order , we mention that in a run list.
- With this process we can run multiple recipes but the condition is, they must be from one cookbook.
- `***(chef client calling default recipes from Ashok-cookbook & Apache-cookbook together) ***`
- `[cookbooks]# chef-client -zr "recipe[Ashok-cookbook::default],recipe[Apache-cookbook::default]"`
- Include Recipe: To call recipes/recipe from another recipe with in same cookbook.
- To run multiple recipes from same cookbook. We can run any numbers of recipes with include command but all must be from same cookbook. Here including recipes with default recipe in Ashok-Cookbook.
- `[cookbooks]# vi Ashok-cookbook/recipes/default.rb`
- `Include_recipe "ABC-cookbook::ABC-recipe"`
- `Include_recipe "ABC-cookbook::XYZ-recipe"`
- `Include_recipe "ABC-cookbook::PQR-recipe"`
- `Esc+:wq` (call the chef client)
- `#chef-client -zr "recipe[Ashok-cookbook::default]"`



# BOOTSTRAP

- Connect workstation to chef server to node using chef-repo, bootstrap
- Chef server is going to be mediator between the code and cookbooks.
- Bootstrapping is Attaching a node to chef server, while using Bootstrap command, both workstation and node should be in same AZ.
- Two actions will be done while bootstrapping
  - 1. adding node to chef server
  - 2. installing chef package.
- Chef-repo It would be the main directory inside it you have to run any commands, it is also having cookbooks).
- Create chef manage account by “manage.chef.io” and download starter kit.
- Go to download and extract file chef-repo,
- After extracting we got more files inside chef-repo are (.chef ,cookbooks ,gitignore, README.md, roles)
- For sending chef-repo file to Linux machine we use the software called WinSCP. Drag Chef-repo from left window and drop

# BOOTSTRAP

- **# Sudo su**
- **# ls**
- **chef-repo chef-workstation-20.7.96-1.e17.x86\_64.rpm cookbooks nodes**
- **# cd chef-repo**
- **# ls -a**
- **. . . .chef cookbooks .gitignore README.md roles**
- **# cd .chef/**
- **[/chef]# ls/**
- **# config.rb      kritikaji.pem (organization name is kritikaji)**
- **# cat config.rb**
- **Inside config.rb you will get chef\_server\_url**
- **<https://api.chef.io/organizations/kritikaji>**
- **# knife ssl check** (to check workstation is connected with chef server ?)
- **Create Linux machine (Node1) same AZ of workstation with new security group and new key pair name node1-key, save Private IP for further knife bootstrap commands.(SSH & HTTPs)**

# BOOTSTRAP

- Attach Advance details
- `[ #!/bin/bash`
- `Sudo su`
- `Yum update -y]`
- With the help of WinSCP please transfer downloaded node1-key.pem to Chef-repo for bootstrap command
- Now go to chef workstation and execute Bootstrap command to attach node1 to chef-server.
- `[chef-repo] # knife bootstrap 172.31.44.20 --ssh-user ec2-user --sudo -i node1.pem -N Node1`
- (Y for YES/NO) Node has been connected to server and node package has installed
- `[chef-repo] # knife node list`
- node1 → (showing result node1 means node1 has been connected to server)



# Moving and delete cookbooks in chef-repo to avoid cookbooks confusion

- `# mv cookbooks/apache-cookbook chef-repo/cookbooks`
- `# mv cookbooks/Ashok-cookbook chef-repo/cookbooks`
- `# ls` [didn't get any cookbook, all empty]
- `# cd chef-repo`
- `# ls` [get (cookbooks node1-key.pem README.md roles)]
- `apache-cookbook ,chefignore starter, Ashok-cookbook` (got inside the chef-repo-cookbook)
- It means both cookbooks have been moved to Chef-repo Cookbooks from
- Upload apache-cookbook to chef-server:
- `[Chef-repo] # knife cookbook upload apache-cookbook`
- `# knife cookbook list` (confirm uploading?)
- `apache-cookbook`
- Now we will attach the recipe on node1 which we would like to run on node1, by this command
- `[chef-repo]# knife Node1:node set node1 "recipe[apache-cookbook::apache-recipe] run_list: recipe[apachecookbook::apache-recipe] [chef-repo]"`
- `# knife node show node1` (get so many info including recipes in run\_list)



# NODE-SERVER

- Now access the Node1
- **# sudo su**
- **# chef-client**
- **\*\*This chef-client implement the code (inside the recipe) on server Automatically\*\*** Now back to workstation and edit the recipe:
- **[Chef-repo]# vi cookbooks/apache-cookbook/recipes/apache-recipe.rb**
- **"Update recipe"**
- Upload apache-cookbook to chef-server
- **[chef-repo] # knife cookbook upload apache-cookbook**
- Now go to the node1 and call chef client
- **# chef-client**
- You can see all updated content, also you can ping node1's public IP and see change.

# How can we automate this process:

- Go to node1
- `[ec2-user] # vi /etc/crontab`
- `* * * * * root chef-client`
- Esc+:wq \*/n (HR DAY MONTH YEAR WEEK)
- With the help of this command automation will start no need to call the chef client again=2 Chef-client command execute periodically according to “\*/n \* \* \* \* crontab method”
- Now see full automation:
- Create one more linux machine Node2 \*(we also can use existing key of node1 for node2 creation)
- Attach Advance details
- `[ #!/bin/bash`
- `Sudo su`
- `Yum update -y`
- `echo “* * * * *root chef-client”>> etc/crontab]`
- Now back to workstation and run Bootstrap command
- `[chef-repo]# knife bootstrap 172.31.10.120 --ssh-user ec2-user --sudo -i node-2key.pem -N node2`
- (Y for YES/NO) Node has been connected to server and node package has been installed
- Now Attach the Recipe to node2 run\_list
- `[chef-repo]# knife node run_list set node2 “recipe[apache-cookbook::apache-recipe]”`
- then for check ping the IP of node2 and see webpage.



# *How to Delete everything from chef-server:*

- To see list of client present inside chef-server, To delete clients
- **[chef-repo] #knife client list**
- **# knife client delete clientname -y**
- To see cookbook list, To delete cookbook
- **[chef-repo] # knife cookbook list**
- **#knife cookbook delete cookbookName -y**
- To see Role list To delete Role
- **[chef-repo] # knife role list**
- **#knife cookbook delete roleName -y**
- To see Node list, To delete Node
- **[chef-repo] # knife node list**
- **#knife cookbook delete nodeName -y**

# How to create *ROLE*:

- `[chef-repo]# ls`
- `.chef roles`
- `# cd roles/ [roles]`
- `#ls`
- `#starter.rb [roles]`
- `# vi Engineer.rb` \*(this is the command to create role name Engineer)
- `Name "Engineer"`
- `Description "webserver role"`
- `run_list "recipe[apache-cookbook::apache-recipe]"` ESC+:wq
- Now back to chef-repo
- `# cd ..` and upload the role on chef server [chef-repo]
- `# knife role from file roles/Engineer.rb`
- If you want to see the created role
- `# knife role list`
- o/p: Engineer Now create 4 instances (1,2,3,4) by one IMA on same availability zone as of workstation with new security group sg-1 with SSH +HTTP.

# How to create *ROLE*:

- Attach Advance details
- `[ #!/bin/bash`
- `Sudo su`
- `Yum update -y`
- `echo"* * * * *root chef-client">> etc/crontab]`
- Now Bootstraps the nodes 1,2,3,4 one by one
- `#[chef-repo]# knife bootstrap 172.31.10.121 --ssh-user ec2-user --sudo -i node-1key.pem -N node1`
- `#[chef-repo]# knife bootstrap 172.31.10.122 --ssh-user ec2-user --sudo -i node-1key.pem -N node2`
- `#[chef-repo]# knife bootstrap 172.31.10.123 --ssh-user ec2-user --sudo -i node-1key.pem -N node3`
- `#[chef-repo]# knife bootstrap 172.31.10.124 --ssh-user ec2-user --sudo -i node-1key.pem -N node4`

## How to create *ROLE*:

- Now connect these nodes to roles one by one.
- **# knife node run-list set node1 "role[Engineer]"**
- Node1:Run\_list:role[Engineer] (similarly for rest 3 nodes)
- **# knife node run-list set node2 "role[Engineer]"**
- **# knife node run-list set node3 "role[Engineer]"**
- **# knife node run-list set node4 "role[Engineer]"**
- **UPLOAD** cookbook to server
- **# knife cookbook upload apache-cookbook**
- Now we can check public IP of any node on webserver, every node will behave like server cause, now cookbook has been uploaded despite of uploading different recipes, all recipes have uploaded together inside role by cookbok.



# How to create *ROLE*:

- Now we are doing changes in recipe
- **# vi cookbooks/apache-cookbook/recipes/apache-recipe.rb**
- Content change to "I Love India"
- ESC+:wq
- Now see if Boss need changes, said do work on another recipe (recipe10)
- **#cat cookbooks/apache-cookbook/recipes/recipe10.rb**
- Paste code update recipe and go to the role in workstation
- **# vi roles/Engineer.rb vi Engineer.rb**
- **Name "Engineer"**
- **Description "webserver role"**
- **run\_list "recipe[apache-cookbook::apache-recipe]" update apache-recipe to recipe10 in role**
- **run\_list "recipe[apache-cookbook::recipe10]"**
- ESC+:wq \*for update in recipe we can create user and file by these commands below
- **#user" Ashok" #file "Anupamfile"**
- now upload role to server
- **[chef-repo] # knife role from file roles/Engineer.rb**

# How to create *ROLE*:

Again, go to the workstation

- **# vi roles/Engineer.rb**
- **Name “Engineer”**
- **Description “webserver role” (change last line only apache-cookbook in role)**
- **run\_list “recipe[apache-cookbook]”**
- **ESC+:wq now upload role to server**
- **[chef-repo] # knife role from file roles/Engineer.rb**
- **Do not mention any recipe just upload only cookbook for all recipes, will update automatically on serve**
- **# knife cookbook upload apache-cookbook**
- **Now we are adding 2 cookbooks in roles**

# How to create *ROLE*:

- `vi roles/Engineer.rb` Name
- `"Engineer"` Description `"webserver role"`
- `run_list "recipe[apache-cookbook]","recipe[Ashok-cookbook]"`
- `esc+:wq`
- now upload role to server
- `[chef-repo] # knife role from file roles/Engineer.rb`
- Do not forget to upload Ashok-cookbook on server otherwise role will not perform properly
- `# knife cookbook upload Ashok-cookbook`
- Boss need changes again but this time in Ashok-recipe
- `[Chef-repo]# vi cookbooks/apache-cookbook/recipes/Ashok-recipe.rb`
- `%W (httpd mariadb-server unzip git vim) .each do |p| Package p do`
- `Action :install end`
- `end esc+:wq # knife cookbook upload Ashok-cookbook`
- Go to inside any node and search git by using command
- `# which git` after 1 minute execute again same command and you will see output `/bin/git` it means working properly.

*Ansible- A push-based  
Configuration management Tool*





# ***Ansible-Introduction***

- **Ansible is an opensource IT configuration management, deployment and orchestration tool.**
- It aims provide large productivity gains to a wide variety of automation challenges.
- Ansible History:
- Michael Dehaan developed ansible and the ansible project began in February 2012.
- Redhat acquired the ansible tool in 2015
- Ansible is available for RHEL, Debian, cent OS and oracle Linux.
- You can use this tool whether your servers are in on-premises or in cloud.
- **It turns your code into infrastructure i.e. your computing environment has some of the same attributes as your application.**

# Ansible-Advantages& Disadvantages

## ➤ Advantages:

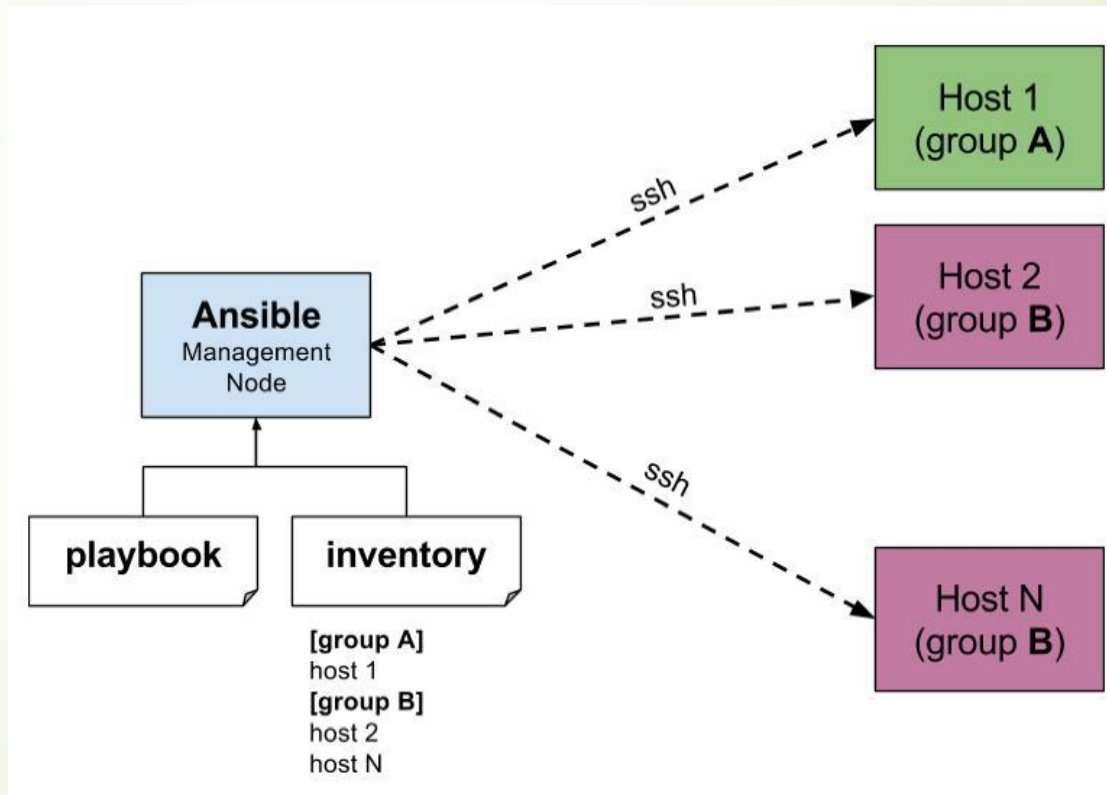
- Ansible is free to use by everyone.
- Ansible is very consistent and light weight and no constraints regarding the OS or underlying hardware are present.
- **It is very secure due to its agentless capabilities and open SSH security features.**
- Ansible doesn't need any special system administrator skills to install and use it.
- **It is push mechanism.**

## ➤ Disadvantages:

- Insufficient user interface, though ansible tower is GUI, but it is still in development stage.
- **Cannot achieve full automation by ansible.**
- New to the market, therefore limited support and document is available.



# Ansible-Architecture





# Terms used in Ansible:

- **Ansible Server:** the machine where ansible is installed and from which all tasks and Playbooks will be run.
- **Module:** a module is a command or set of similar commands meant to be executed on the client side.
- **Task:** a task is section that consist of a single procedure to be completed.
- **Role:** a way of organizing tasks and related files to be later called playbook.
- **Fact:** information fetched from the client form the global variables with the gather facts operation.
- **Inventory:** file containing data about the ansible client servers. g. Play: execution of playbook.
- **Handler:** task which is called only if notifier is present.
- **Notifier:** section attributed to a task which calls a handler if the output is changed.
- **Playbooks:** it consists code in YAML format which describes tasks to be executed.
- **Host:** nodes which are automated by ansible

# Ansible-Installation

- Go to AWS account- create 3 EC2 instances in same AZ.
- Take access of all machines via putty.
- Now go inside ansible server and download ansible package
- # `wget https://dl.fedoraproject.org/pub/epel/epel-release-latest-9.noarch.rpm`
- Now do `#ls`
- # `yum install epel-release-latest-9.noarch.rpm`
- `#yum update -y`
- Now we have to install all the packages one by one
- `sudo yum install ansible`
- `sudo yum install python`
- `sudo yum install git`
- Let's rename Master server and Node 1 and Node2,their names
- `[ec2-user@ip172-31-91-98] $ sudo hostnamectl set-hostname ansible-master`
- `[ec2-user@ip172-31-80-251] $ sudo hostnamectl set-hostname ansible-Node1`
- `[ec2-user@ip172-31-82-233] $ sudo hostnamectl set-hostname ansible- Node2`
- `[ec2-user@ip172-31-91-98] $ hostname`
- `ansible-master ansible- Node1 ansible- Node2`

# Ansible->Node Connection

- Now go to host file- inside ansible server and paste private ip of node1 and node2
- `# vi /etc/ansible/hosts`
- Now this host file is only working after updating ansible.cfg file
- `# vi /etc/ansible/ansible.cfg`
- `[defaults] inventory = /etc/ansible/hosts`
- `sudo-user = root`
- Now create one user, in all the three instances
- `# adduser ansible`
- Now set password for this user
- `# passwd ansible`
- Now switch as ansible user
- `# su – ansible`
- This ansible user don't have sudo priviledges right now.If you want to give sudo priviledge to ansible user
- `# visudo`

# Ansible->Node Connection

- Now go inside this file
- `Root ALL= (ALL) ALL`
- `(ansible ALL= (ALL) NOPASSWD: ALL)`
- `:wq!`
- Now do this thing in other nodes also.
- Now go to ansible server and try to install httpd package as an ansible user.
- `# sudo you install httpd -y`
- Now establish connection between server and node, go to ansible server
- `$ ssh 172.31.41.240`
- o/p- permission denied now we have to do some changes in sshd-config file, go to ansible server

# Ansible->Node Connection

- `# vi /etc/ssh/sshd-config`
- Do some changes and saved the file.
- Do this work in node1 and node2 also.
- Now verify in ansible server
- `# su -ansible`
- `# ssh 172.31.41.240`
- Now it asks for passwd, enter the password after that you will be inside node1.
- Now go to ansible server and create keys.
- Run this command as ansible user.
- `# ssh-keygen`
- `# ls -a`
- `o/p- .ssh`
- `# cd .ssh/`
- `ls`
- `o/p- id_rsa id_rsa_pub`
- now I need to copy public key in both the nodes. (ansible@private\_ip of node)
- `# ssh-copy-id ansible(username)@172.31.41.240`





# ***Ansible->Node Connection***

- Ask for password
- `# ssh-copy-id ansible@172.31.41.228`
- –Ask for password
- Now verify, go to ansible server
- `# cd ..`
- `# ssh 172.31.41.240`
- Now you will enter into node1.



# Host Patterns:

- `# vi /etc/ansible/hosts`
- “all” pattern refers to all the machines in an inventory
- Ansible all -list-host
- Ansible<group name> --list-hosts
- Ansible<group name>[0] -list-hosts
- Groupname[0]- picks first machine of the group
- Groupname[1]- picks second machine of the group
- Groupname[-1]- picks last machine of the group
- Groupname[0:1]- picks first two machines of the group
- Groupname[2:5]- picks 3,4,5 and 6 machines of the group
- Group separated by colon (:) can be used to use hosts from multiple groups. Groupname1:groupname2

# Ad-hoc Commands:

- Ad-hoc commands are commands which can be run individually to perform quick functions.
- These ad-hoc commands are not used for configuration management and deployment, because these commands are of one-time usage
- The ansible ad-hoc commands uses the /usr/bin/ansible command line tool to automate a single task.
- Go to ansible server
- `$ ansible demo -a "ls"`
- `$ ansible demo[0] -a "touch filez"`
- `$ ansible all -a "touchfile4"`
- `$ ansible demo -a "ls-al"`
- `$ ansible demo -a "sudo yum install httpd -y" or`
- `$ ansible demo -ba "yum install httpd -y"`
- `$ ansible demo -ba "yum remove httpd -y"`

# Ansible Modules:

- Ansible ships with a number of modules (called module library) that can be executed directly on remote hosts or through “playbooks”
- Your library of modules can reside on any machine and there are no servers, daemons or databases required.
- Q. where ansible modules are stored?
- The default location of the inventory file is `/etc/ansible/hosts`.
- `$ ansible demo -b -m yum -a “pkg=httpd state=present”`
- `$ ansible demo -b -m yum -a “pkg=httpd state=latest”`
- `$ ansible demo -b -m yum -a “pkg=httpd state=absent”`
- `$ ansible demo -b -m service -a “name=httpd state=started”`
- `$ ansible demo -b -m user -a “name=raj”`
- `$ ansible demo -b -m copy -a “src=file4 dest=/tmp”`
- `$ ansible demo -m setup`
- `$ ansible demo -m setup -a “filter= *ipv4*”`



# Playbook

- ▶ Playbooks in ansible are written in YAML format
- ▶ It is human readable data serialization language and is commonly used for configuration files.
- ▶ Playbook is like a file where you write codes consists of variables, tasks, handlers, files, templates and roles.
- ▶ Each playbook is composed of one or more 'modules' in a list. Module is a collections of configuration files.
- ▶ Playbooks are divided into many sectors like
  - ▶ a. Target section: defines the host against which playbooks task has to be executed.
  - ▶ b. Variable: define variables
  - ▶ c. Task section: list of modules that we need to run in an order. YAML (Yet Another Markup Language):
- ▶ For ansible nearly every YAML files starts with a list.
- ▶ Each item in the list is a list of key-value pairs commonly called as a directory.



# Playbook

- All YAML files have to begins with "---" and ends with "...".
- All members of a list lines must begin with same indentation level starting with "-".
- For e.g:
- --- # a list of fruits
- -Fruits:
- Mango
- Strawberry
- Banana
- Grapes
- Apple
- ... A directory is represented in a sample key : value form
- --- # details of customer
- -Customer: Name: Ashok
- Job: ASE
- Skills: devops
- Exp: 12 year





# Playbook

- ▶ Extension for playbook files is .yaml.
- ▶ Note: there should be space between : and value.
- ▶ Go to ansible server.
- ▶ Now create one playbook.
- ▶ # vi target.yaml
- ▶ --- # target playbook
- ▶ -hosts: demo
- ▶ -user: ansible
- ▶ -become: yes
- ▶ -connection: ssh
- ▶ -gather\_facts: yes
- ▶ Esc- :wq!
- ▶ \$ ansible-playbook target.yaml

# Variables

- Ansible uses variables which are defined previously to enable more flexibility in playbooks and roles.
- They can be used to 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.
- Put variable section above tasks so that we define it first and use it later. Now go to ansible server and create one playbook.
- `$ vi vars.yml`
- `--- # my variable playbook`
- `-host: demo`
- `- user: ansible`
- `-become: yes`
- `-connection: ssh`
- `Vars:`
- `Pkgname: httpd`
- `Tasks:`
- `-name: install httpd server`
- `-action: yum name= "{{pkgname}}" state=install`
- `Esc - :wq!`
- `Now execute playbook $ ansible-playbook vars.yml`



# ***Handlers Section:***

- A handler is exactly the same as a task, but it will run when called by another task.
- Or Handlers are just like regular tasks in an ansible playbook, but are only 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.
- Anible-playbook handlers.yml –check



# Handlers Section:

- Go to ansible server
- `$ vi handler.yml`
- `---# handlers playbook`
- `-hosts: demo`
- `-user: ansible`
- `-become: yes`
- `-connection:ssh`
- Tasks:
- Name: install httpd server
- Action: `yum name=httpd state=installed`
- Notify: restart HTTPD
- Handlers:
- Name: restart HTTPD
- Action: `service name=httpd state=restarted`
- Esc- :wq!
- Now execute this playbook `$ ansible-playbook handlers.yml`

# Loops

- Sometimes you want to repeat a task multiple time. In computer programming this is called as loops.
- Common ansible loops include changing ownership on several files and/or directories with the file module, creating multiple users with the user module and repeating a polling step until certain result is reached.
- Now go to ansible server.
- `$ vi loops.yml`
- `---# my loops playbook`
- `-host: demo`
- `-user: ansible`
- `-become: yes`
- `-connection:ssh`
- Tasks: `-name:`
- `add a list of users`
- `-user: name= "{{item}}" state=present`
- With items:
- `-Ashok`
- `-Anupam`
- `-Kritika`
- `-Sunny`
- `-Summy`
- `Esc- :wq!`
- `$ansible-playbook loops.yml` To verify go inside node1 `$cat /etc/passwd`



# Conditions

- Whenever we have different different scenarios we put conditions to the scenario.
- We put conditions in ansible by “when” statement.
- ---# condition playbook
- -hosts: demo
- -user: ansible
- -become: yes
- -connection:ssh
- Tasks:
- -name: install apache on Debian
- Command: apt-get -y install apache2
- When: ansible\_os\_family == “Debian”
- -name: install apache for redhat
- Command: yum -y install httpd
- When: ansible\_os\_family == “Redhat”





# Vault

- Ansible allows keeping sensitive data such as passwords or key in encrypted files, rather than a plaintext in your playbooks.
- Creating a new encrypted playbook:
- `$ ansible-vault create vault.yml`
- Edit the encrypted playbook:
- `$ ansible-vault edit vault.yml`
- To change the password:
- `$ ansible-vault rekey vault.yml`
- To encrypt an existing playbook:
- `$ ansible-vault encrypt target.yml`
- To decrypt an encrypted playbook:
- `$ ansible-vault decrypt target.yml`

# Roles

- We can use two techniques for reusing a set of tasks :
- includes and role.
- Roles are good for organizing tasks and encapsulating data needed to accomplish those tasks.
- We can organize playbook into a directory structure called roles.
- Adding more and more functionality to the playbooks will make it difficult to maintain in a single file.
- Ansible Roles:
  - **a. Default:**
    - it stores the data about role/ application. Default variables e.g: if you want to run to port 80 or 8080 then variables need to define in this path.
  - **b. Files:** it contains files need to be transferred to the remote VM (static files).
  - **c. Handles:** they are triggers or task. We can segregate all the handlers required in playbook.
  - **d. Meta:** this directory contains files that establish roles dependencies. E.g: author name, supported platform, dependencies if any.
  - **e. Templates:**
  - **f. Tasks:** it contains all the tasks that is normally in the playbook. E.g: installing packages and copies files etc.
  - **g. Vars:** variables for the role can be specified in this directory used in your configuration files. Both vars and default stores variables

# Roles

- `$ mkdir -p playbooks/roles/webserver/tasks`
- `$ tree -p playbook/roles/webserver/handler`
- `$ cd playbook/`
- `$ tree`
- `$ touch roles/webserver/tasks/main.yml`
- `$ touch master.yml`
- `$ vi roles/webserver/tasks/main.yml`
- **Inside main.yml**
- `-name: install apache`
- `-yum: pkg=httpd state=latest`
- `Esc- :wq!`
- `$ vi master.yml`
- `-host: all`
- `-user: ansible`
- `-become: yes`
- `-connection:ssh`
- `Roles:`
- `-webserver`
- `$ ansible-playbook master.yml`

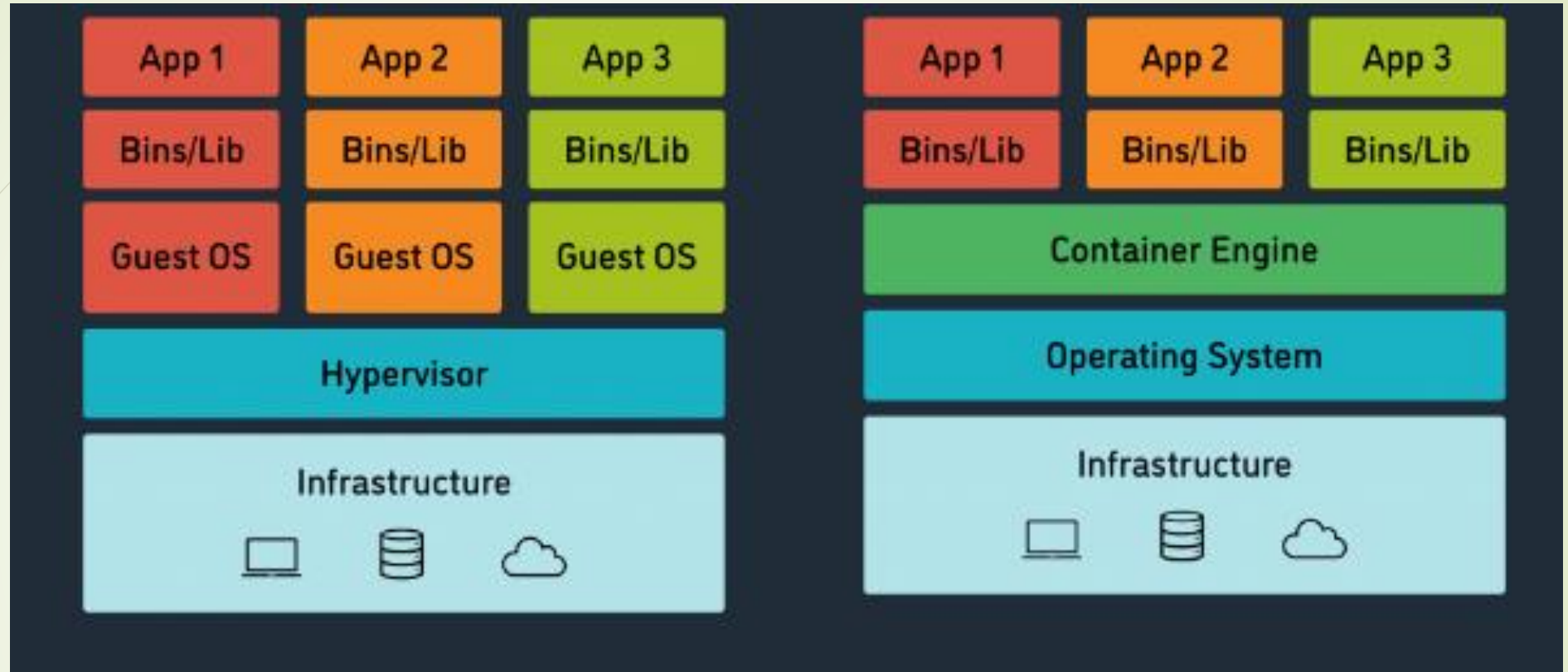


***Docker- A Containerization Tool***



# Docker-Introduction

- Docker is an opensource centralized platform designed to create, deploy and run applications.
- **Docker uses container on the host OS to run applications.** It allows applications to use same Linux kernel as a system on the host computer rather than creating a whole virtual OS.
- We can install docker on any OS but docker engine runs natively on Linux distributions.
- Docker written in “GO” programming language.
- Docker is a tool that performs OS level virtualization also known as Containerization.
- Before docker many users face the problem that a particular code is running in the developer's system but not in the user's system.
- Docker was first released in march 2013. It is developed by Solomon Hykes and Sebastian Pahl.
- Docker is a set of Platform-as-a-Service that uses OS level virtualization whereas VMWare uses hardware level of virtualization.




**Docker uses container on the host OS to run applications**

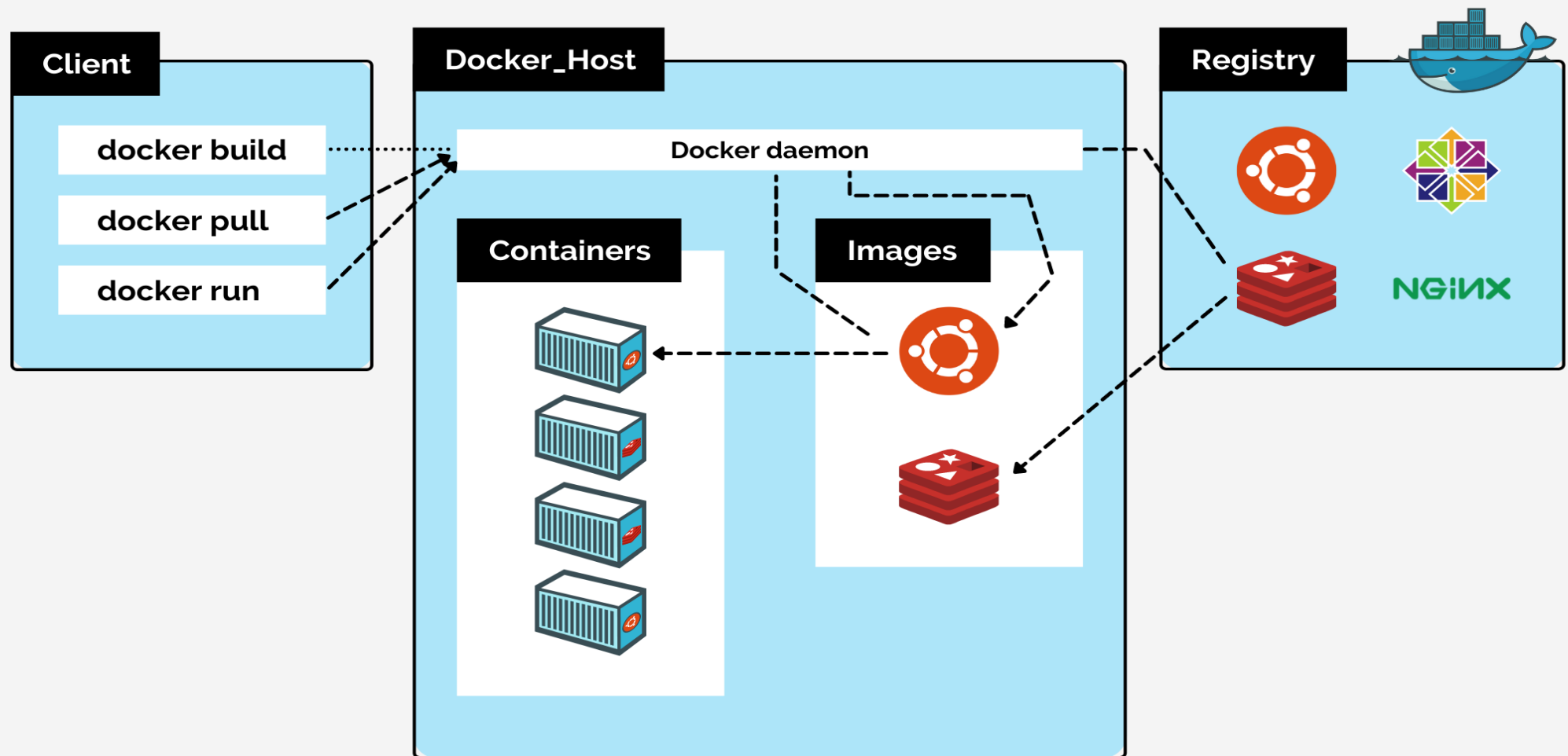
Here OS Level Virtualisation, whereas VMWARE has Hardware level virtualisation.





# Advantages of Docker:

- No pre-allocation of RAM.
  - CI efficiency: - docker enables you to build a container image and use that same image across every step of the deployment process.
  - Less cost.
  - It is light in weight.
  - It can run on physical h/w / virtual h/w or on cloud.
  - You can reuse this image.
  - It takes very less time to create Containers
- 



WHIZLABS

# Docker Architecture

# Components of Docker:

## ➤ A. Docker Daemon:

- Docker daemon runs on host O.S
- It is responsible for running containers to manages docker services.
- Docker daemon can communicate with other daemons.

## ➤ B. Docker Client:

- Docker users can interact with docker through a client.
- Docker client uses commands and REST API to communicate with the docker daemon.
- When a client runs any server command on the docker client terminal, the client terminal sends these docker commands to the docker daemon.
- It is possible for docker client to communicate with more than one daemon.

## ➤ C. Docker Host:

- Docker host is used to provide an environment to execute and run applications.
- It contains the docker daemon, images, containers, networks and storages

# Components of Docker:

## ➤ D. Docker Hub/ Registry:

- Docker registry manages and stores the docker image.
- There are two types of registries in the docker:
  - a. Public Registry: it is also called as docker hub.
  - b. Private Registry: it is used to share image with in the enterprise.

## ➤ E. Docker Image:

- Docker images are the read only binary templates used to create docker containers.
- Or Single file with all the dependencies and configuration required to run a program.
- Ways to Create an Image:
  - a. Take image from the docker hub.
  - b. Create image from docker file.
  - c. Create image from existing docker containers.



# Components of Docker:

- **F. Docker Containers**
- Containers hold the entire packages that is needed to run the application.
- Or In other words, we can say that the image is a template and the container is a copy of that template.
- Container is like a virtual machine.
- Images becomes container when they run on docker engine.

# Basic Docker Commands:

- To see all images present in your local repo: `# docker images`
- To find out images in docker hub: `#docker search image_name`
- To download image from dockerhub to local machine `# docker pull image_name:`
- To check service start or not (status) : `# docker service status`
- To start: `#docker service start`
- To stop: `# docker service stop`
- To start container : `#docker start container_name`
- To go inside container `# docker attach container_name`
- To see all containers: `# docker ps -a`
- To see running containers: `# docker ps`
- To stop container: `# docker stop container_name`
- To delete a container: `# docker rm container_name`



# Create container from our own Image:

- Login into AWS account and start your EC2 instance,
- access it from putty.
- Now we have to create container from our own image.
- Therefore, create one container first:
- `#docker run -it --name container_name image_name /bin/bash`
- `#cd tmp/`
- Now create one file inside this tmp directory
- `# touch myfile`
- Now if you want to see the difference between the basic image and the changes on it
- `# docker diff container_name image_name`



## *Now create image of this container*

- `# docker commit newcontainer_name image_name`
- `# docker images`

## ➤ *Now create container from this image*

- `# docker run -it --name newcontainer_name image_name /bin/bash`
- `# ls`
- `# cd tmp`
- `# ls (you will get all of your files)`

# Dockerfile

- Dockerfile is basically a text file. It contains some set of instructions. Automation of docker image creation.
- Dockerfile components:
- **FROM**: for base image, this command must be on the top of the dockerfile.
- **RUN**: to execute commands, it will create a layer in image
- **MAINTAINER**: author/ owner/ description
- **COPY**: copy files from local system (docker vm) we need to provide source, destination (we can't download) file from internet and any remote repo.)
- **ADD**: similar to copy but it provides a feature to download files from internet, also extract file at docker image side.
- **EXPOSE**: to expose ports such as port 8080 for tomcat , port 80 for nginx etc.CMD: execute commands but during container creation.
- **ENTRYPOINT**: similar to CMD but has higher priority over CMD, first commands will be executed by ENTRYPOINT only.
- **ENV**: environment variables

# Dockerfile

- ➤ Create a file named Dockerfile
- ➤ Add instructions in Dockerfile
- ➤ Build dockerfile to create image
- ➤ Run image to create container
- # vi Dockerfile
- FROM ubuntu
- RUN echo "Ashok Anupam" > /tmp/testfile
- To create image out of Dockerfile
- # docker build -t myimg
- # docker ps -a
- # docker image



# Dockerfile

- Now create container from the above image
- `#docker run -it --name mycon myimg /bin/bash`
- `#cat /tmp/testfile`
- `#vi dockerfile`
- `FROM ubuntu`
- `WORKDIR /tmp`
- `RUN echo "thank you" > /tmp/testfile`
- `ENV myname Ashok`
- `COPY testfile1 /tmp`
- `ADD test.tar.gz /tmp`

# Docker Volume:

- Volume is simply a directory inside our container.
- ➤ Finally, we have to declare this directory as a volume and then share volume.
- ➤ Even if we stop the container still, we can access volume.
- ➤ Volume will be created in one container.
- ➤ We can declare a directory as a volume only while creating container.
- ➤ We can't create volume from existing container.
- ➤ We can share one volume across any number of containers.
- ➤ Volume will not be included when We update an image.
- ➤ We can map volume in two ways:
  - a. Container to container
  - b. Host to container
- **Benefits of Volume**
  - ➤ Decoupling container from storage.
  - ➤ Share volume among different containers.
  - ➤ Attach volume to containers.
  - ➤ On deleting container volume doesn't delete.



# Creating Volume from Dockerfile:

- Create a Dockerfile and write
- FROM ubuntu
- VOLUME "myvolume"
- Then create image from this Dockerfile
- #docker build -t myimage
- Now create a container from this image and run
- # docker run -it --name container1 myimage /bin/bash
- Now do ls, you can see myvolume.
- Now share volume with another container
- Container to container
- # docker run -it --name container2 (new) --privileged=true --volumesfrom container1 ubuntu /bin/bash

# Creating Volume from Dockerfile

- Now after creating container2, myvolume is visible. Whatever you do in one volume, can see from other volume.
- `#touch /myvolume/samplefile`
- `#docker start container1`
- `# docker attach container1`
- `#ls/myvolume`
- You can see sample file here then exit.
- **Now create volume by using command:**
- `#docker run -it --name container3 -v /volume2 ubuntu /bin/bash`
- `# ls`
- `#cd /volume2`
- Now create one file cont3file and exit
- Now create one more container and share volume2
- `#docker run -it --name container4 --privileged=true --volumefrom container3 ubuntu /bin/bash`
- Now you re inside container do ls you can see volume2 Now create one file inside this volume and then check in container3 you can see that file

# Volumes (Host to Container)

- Verify files in /home/ec2-user
- `#docker run -it --name hostcontainer -v /home/ec2-user:/container --privileged=true ubuntu /bin/bash`
- `#cd /container`
- Do ls, now you can see all files of host machine.
- `#touch containerfile` (in container) and exit
- Now check in EC2 machine you can see this above file



## *Some other commands:*

- `#docker volume`
- `ls`
- `#docker volume create`
- `#docker volume rm`
- `#docker volume prune` (it removes all unused docker volume)
- `#docker volume inspect`
- `#docker container inspect`

# Docker Port Expose:

- Login into AWS account, create one linux instance .
- Now go to putty -> login as -> ec2-user
- #sudo su
- # yum update -y
- # yum install docker -y
- # service docker start
- # docker run -td --name techserver -p 80:80 ubuntu
- # docker ps
- # docker port techserver
- o/p- 80/tcp - 0.0.0.0/80
- # docker exec -it techserver /bin/bash
- # apt-get update # apt-get install apache2 -y
- # cd /var/www/html
- # echo "write some msg" > index.html
- #service apache2 start
- # docker run -td --name myjenkins -p 8080:8080 jenkins

# Difference between docker attach and docker exec:

- Docker 'exec' creates a new process in the container's environment
- while docker 'attach' just connect the standard input/output of the main process inside the container to corresponding standard input/output error of current terminal.
- Docker 'exec' is specifically for running new things in an already started container be it a shell or some other process.
- What is the difference between docker expose and publish:
- Basically you have three options:
- 1. Neither specify expose nor -p Docker
- 2. Only specify expose
- 3. Specify expose and -p
- 1. If you specify neither expose nor -p, the service in the container will only be accessible from inside the container itself.
- 2. If you expose a port, the service in the container is not accessible from outside docker but from inside other docker containers so this is good for inter-container communication.
- 3. If you expose and -p a port, the service in the container is accessible from anywhere even outside docker.
- If you do -p but do not expose docker does an implicit expose.
- This is because if a port is open to the public, it is automatically also open to the other docker containers. Hence -p includes expose



# How to push docker image in docker hub:

- Go to AWS account – select Amazon linux
- Now go to putty – login as – ec2-user
- `#sudo su`
- `#yum update -y`
- `#yum install docker -y`
- `#service docker start`
- `#docker run -it ubuntu /bin/bash`
- Now create some files inside container, now create image of this container
- `#docker commit container1 image1`
- Now create account in [hub.docker.com](https://hub.docker.com)
- Now go to EC2 instance
- `#docker login`
- Enter your username and password Now give tag to your image
- `#docker tag image1 dockerid/newimage`
- `#docker push dockerid/newimage`
- Now you can see this image in docker hub account
- Now create one instance in another region and pull image from hub
- `#docker pull dockerid/newimage`
- `#docker run -it --name mycon dockerid/newimage /bin/bash`

# NAGIOS

Nagios-A Monitoring Tool




# Nagios-Introduction

- Nagios is an open-source software for continuous monitoring of systems, networks and infrastructure.
- It runs plugins stored on a server which is connected with a host or another server on your network or the internet.
- In case of any failure Nagios alerts about the issues so that the technical team can perform recovery process immediately.
- History of Nagios:
  - In the year 1999, Ethan Galstad developed it as a part of netsaint distribution.
  - 2002, ethan renames the project to “Nagios” because of trademark issues with the name “netsaint”.
  - 2009, Nagios releases its first commercial version, Nagios XI.
  - In 2012, Nagios again renamed as Nagios core.
  - It uses port number 5666, 5667 and 5668 to monitor its client.



# *Why Nagios?*

- Detect all types of network or server issues.
- Helps to find the root cause of the problem which allow you to get the permanent solution to the problem.
- Reduce downtime.
- Active monitoring of entire infrastructure.
- Allow you to monitor and troubleshoot server performance issues.
- Automatically fix problems.



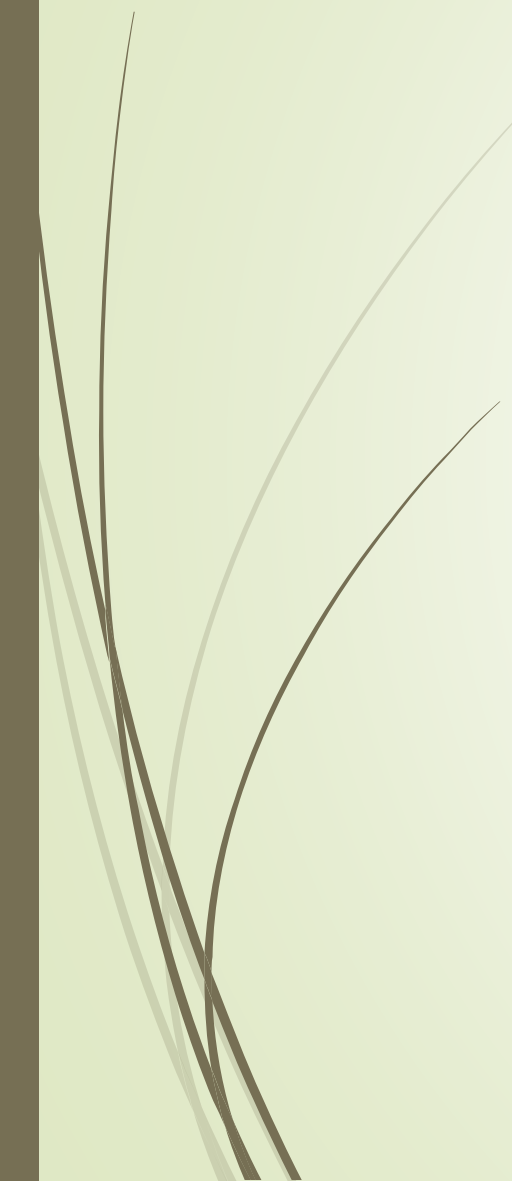
## *Features of Nagios:*



- Oldest and latest.
- Good log and database system.
- Informative and attractive web interface.
- Automatically send alerts if condition changes.
- Helps you to detect network errors or server crashes.
- You can monitor the entire business process and IT infrastructure with a single pass.
- Monitor network services like http, smtp, snmp, ftp, ssh, pop, DNS, LDAP, IPMI etc.


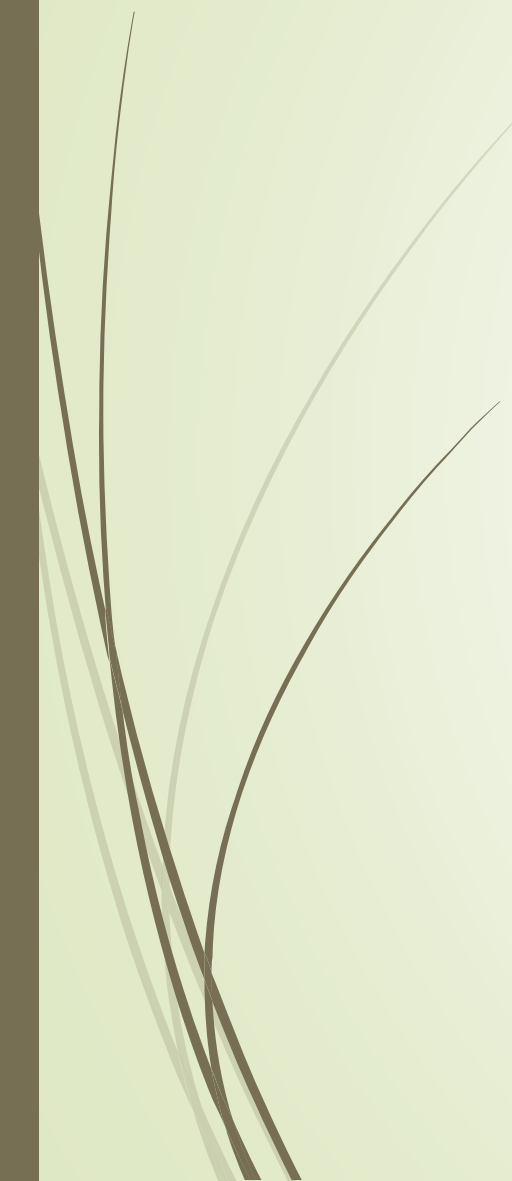


## *Phases of Continuous Monitoring:*

- 1. Define: develop a monitoring strategy
  - 2. Establish: how frequently you are going to monitor it.
  - 3. Implement
  - 4. Analyze data and report finding
  - 5. Respond
  - 6. Review and update
- 

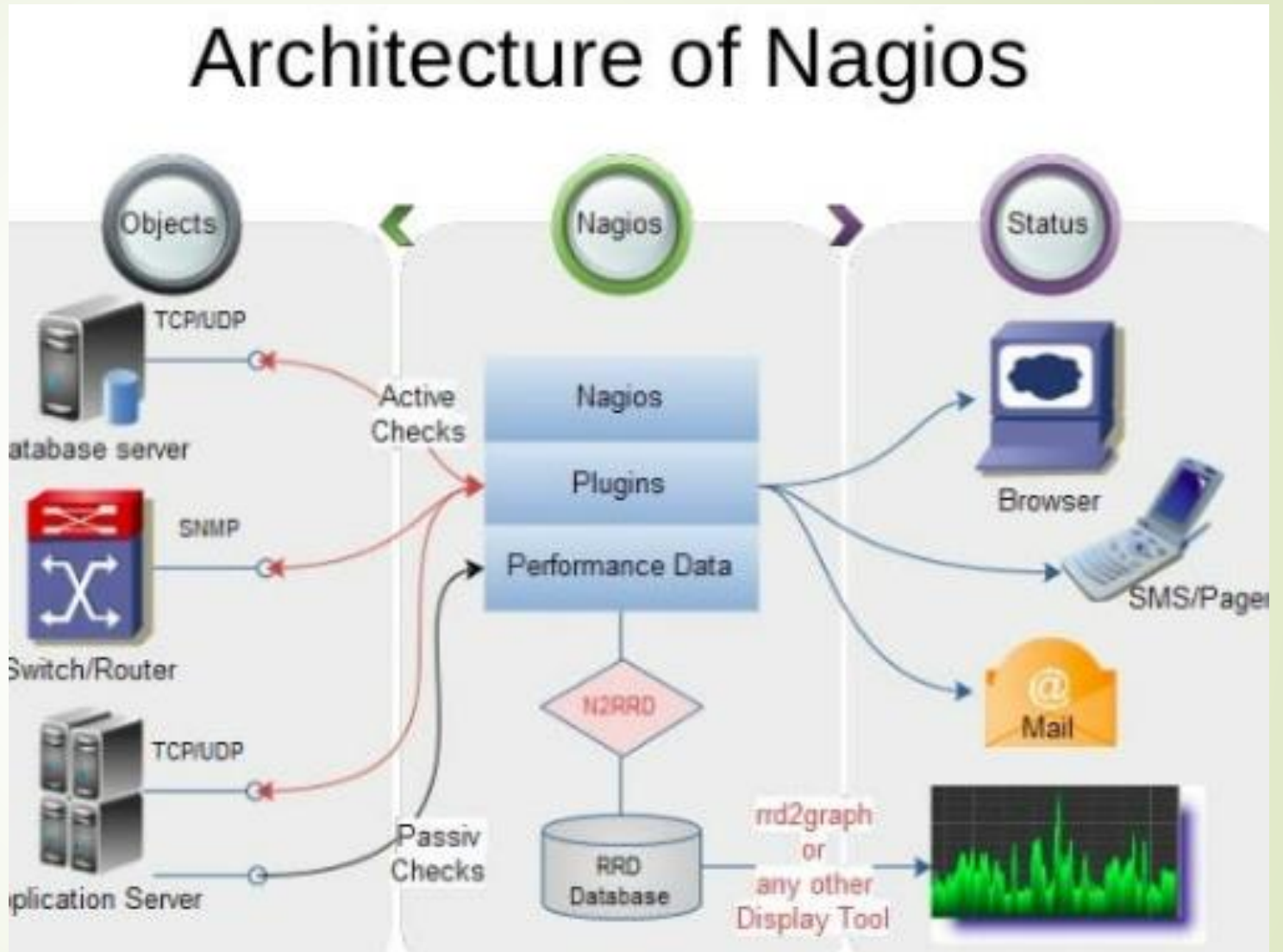


# *Nagios Architecture:*

- 
- 
- Nagios is a client-server architecture. Usually on a network, a Nagios server is running on a host and plugins are running on all the remote host which should you monitor.

# How does Nagios works?

- Mention all details in configuration files.
- Daemon read those details what data to be collected.
- Daemon use NRPE plugins to collect data from nodes and store in its own database.
- Finally shows everything in dashboard.



## Pre-requisites and Config Files

- **Pre-requisites:**
- • Httpd (browser)
- • Php (dashboard)
- • Gcc and gd (compiler)
- • Makefile (to build)
- • Perl (script)
- Main configuration file: /usr/local/Nagios/etc.Nagios.cfg
- All monitoring things called as service.
- For e.g: 5 servers – 4 checks each
- Then you have to monitor  $5 \times 4 = 20$  services
- Dashboard overview:
- In dashboard you can see
- Host: down, unreachable, up, recovery, none
- Service: warning, unknown, critical, recovery, pending

## Installation of Nagios on Linux:

- To start Nagios core installation, you must have your EC2 instance up and run and have already configured SSH access to the instance.
- **Step-1**: install pre-requisites software on your EC2 machine prior to Nagios installation like apache, php, gcc compiler and gd development libraries.
- `# sudo su`
- `# yum install httpd php`
- `# yum install gcc glibc glibc-common`
- `# yum install gd gd-devel`
- **Step-2**: create account information you need to setup a Nagios user, run the following commands,
- `# adduser -m Nagios`
- `# passwd Nagios`
- Now it will ask to enter new password give '12345' as password.
- `# groupadd Nagioscmd`
- `# usermod -a -G Nagioscmd Nagios`
- `# usermod -a -G Nagioscmd apache`

# Installation of Nagios on Linux:

- **Step-3:** download Nagios core and the plugins. Create a directory for storing the downloads.
- `# mkdir ~/downloads`
- `# cd ~/downloads`
- Download the source code tarballs of both Nagios and the Nagios plugins.
- `# wget http://prdownloads.sourceforge.net/sourceforge/Nagios/Nagios-4.0.8.tar.gz`
- `# wget http://Nagios-plugins.org/download/Nagios-plugins-2.0.3.tar.gz`
- **Step-4:** complete and install Nagios extract the Nagios source code tarball
- `# tar zxvf Nagios-4.0.8.tar.gz`
- `# cd Nagios-4.0.8`
- Run the configuration script with the name of the group which you have create in above step.
- `# ./configure --with-command-grouo=Nagioscmd`
- Compile the Nagios source code
- `# make all`
- Install binaries, init script, sample config files and set permissions on the external command directly.
- `# make install`
- `# make install-init`
- `# make install-config`
- `# make install-commandmode`



# Installation of Nagios on Linux:

- **Step-5:** configure the web interface
- # make install-webconf
- **Step-6:** create a 'Nagiosadmin' account for login into Nagios web interface, set password as well.
- # htpasswd -c /usr/local/Nagios/etc/htpasswd.users Nagiosadmin
- Asking for a password, set a new pwd
- # service httpd restart
- **Step-7:** compile and install the Nagios plugins. Extract the Nagios plugins source code tarball.
- # cd ~/downloads
- # tar zxvf Nagios-plugins-2.0.3.tar.gz
- # cd Nagios-plugins-2.0.3
- Compile and install the plugins
- # ../configure --with-Nagios-user=Nagios --with-Nagios-group=Nagios
- # make
- # make install



# Installation of Nagios on Linux:

- **Step-8:** start Nagios. Add Nagios to the list of system services and have it automatically start when the system boots.
- # chkconfig --add Nagios
- # chkconfig Nagios on
- Verify the sample Nagios configuration files
- # /usr/local/Nagios/bin/Nagios -v
- # usr/local/Nagios/etc/Nagios.cfg
- If there are no errors, start Nagios
- # service Nagios start
- # service httpd restart
- **Step-9:** copy public ip of EC2 instance and paste in google chrome, in given way
- For e.g 20.1.1.1/Nagios/Nagios
- Ask for username- Nagiosadmin
- Password-



**KUBERNETES- K8s**

*A container management tool*

# KUBERNETES-Introduction

- 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.
- All top cloud providers support Kubernetes.
- History:
  - Google developed an internal system called 'borg' (later named as omega) to deploy and manage thousands google application and services on their cluster.
  - In 2014, google introduced Kubernetes as an open-source platform written in Golang, and later donated to CNCF (cloud Native Computing Foundation).
- Online platform for K8s:
  - i. Kubernetes playground
  - ii. Play with K8s
  - iii. Play with Kubernetes classroom

# KUBERNETES-Intro

- Cloud based K8s services:
  - i. GKE: Google Kubernetes Services
  - ii. AKS: Azure Kubernetes services
  - iii. Amazon EKS: Amazon Elastic Kubernetes Services
- Kubernetes installation tool:
  - i. Minicube
  - ii. Kubeadm
- Problems with scaling up the containers:
  - ➤ Containers cannot communicate with each other.
  - ➤ Autoscaling and load balancing was not possible.
  - ➤ Containers had to be managed carefully



## Features of Kubernetes:

- Orchestration (clustering of any number of containers running on different networks)
- Autoscaling (supports both horizontal and vertical scaling)
- Auto-healing
- Load balancing
- Platform independent (cloud/ virtual/ physical)
- Fault tolerance (node/ POD failure)
- Rollback (going back to previous version)
- Health monitoring of containers
- Batch execution (one time, sequential, parallel)

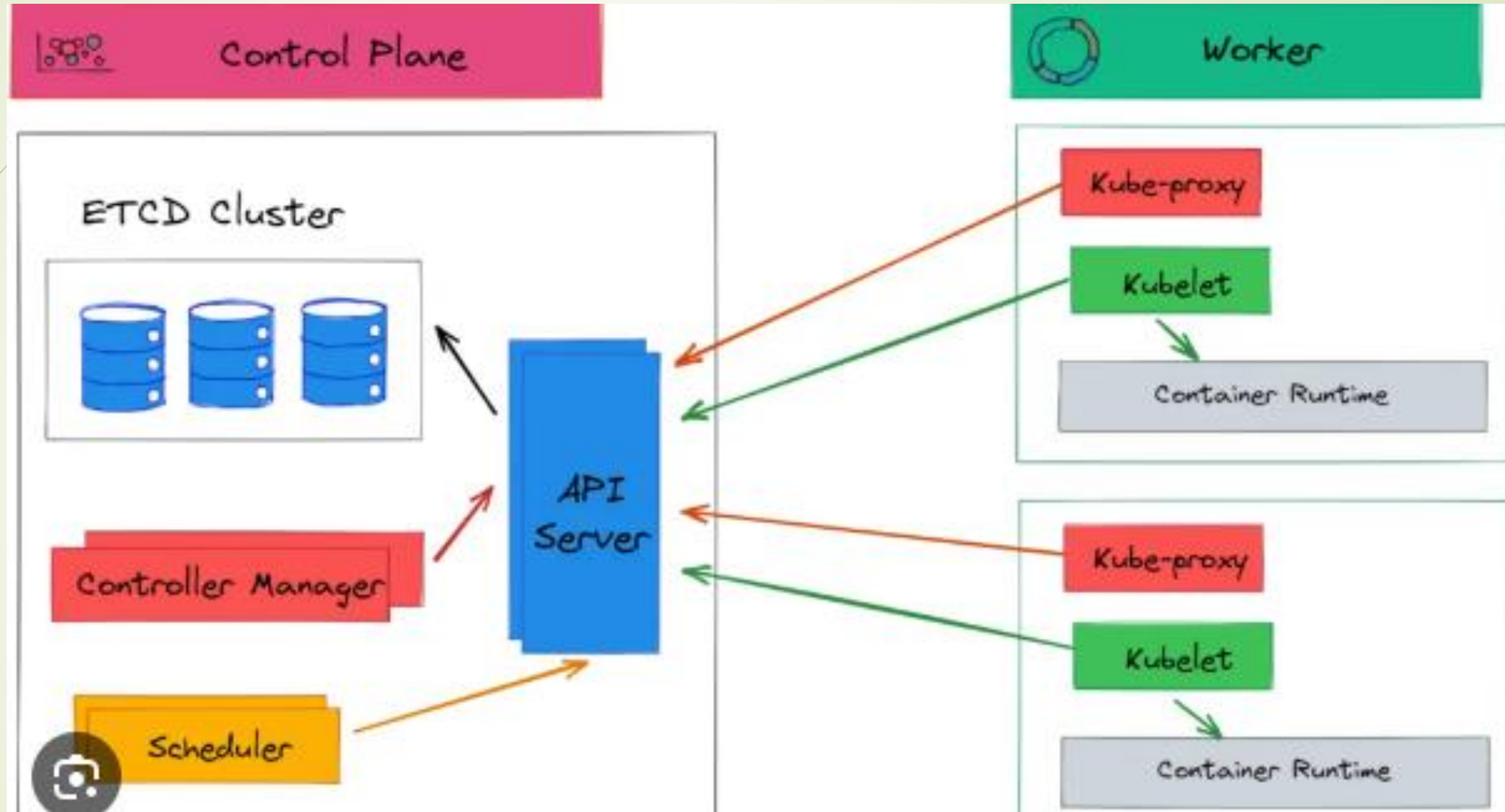


FEATURES	KUBERNETES	DOCKER SWARM
Installation and cluster configuration	Complicated and time consuming	Fast and easy
Supports	K8s can work with almost all container types like rocket, docker, containerD	Work with docker only
GUI	GUI available	GUI not available
Data volumes	Only shared with containers in same POD	Can be shared with any other containers
Update and rollback	Process scheduling to maintain services while updating	Progressive updates and service health monitoring throughout the update
Autoscaling	Support vertical and horizontal autoscaling	Not support autoscaling
Logging and monitoring	Inbuilt tool present for monitoring	Used 3 <sup>rd</sup> party tools like splunk

## Kubernetes and Docker-Swarm Comparison

**Kubernetes-Better option**





K8'S

# KUBERNETES- ARCHITECTURE

# Working with Kubernetes

- We create manifest (.yaml/json).
- Apply this to cluster (to master) to bring desired state.
- POD runs on node which is controlled by master.
- **Role of Master Node:**
- Kubernetes cluster contains containers running or bare metal/ vm instances. Cloud instances/ all mix.
- Kubernetes designates one or more of these as master and all others as workers.
- The master is now going to run set of k8s processes. These processes will resume smooth functioning of cluster. These processes are called "control plane".
- Can be multi-master for high availability.
- Master runs control plane to run cluster smoothly.
- **Components of Control Plane (master):**
- 1. Kube-API server
- 2. Etcd
- 3. Kube-scheduler
- 4. Controller manager

# Components of Control Plane (master):

- 1. Kube-API server (for all communications):
  - ➤ This api-server interacts directly with user (i.e we apply .yaml or json manifest to kube-apiserver).
  - ➤ This kube-apiserver is meant to scale automatically as per load.
  - ➤ Kube api-server is front-end of control-plane.
- 2. Etcd:
  - ➤ It stores metadata and status of cluster.
  - ➤ Etcd is consistent and high available store (key-value store).
  - ➤ Source of truth for cluster state (info about state of cluster).
  - Etcd has following features: -
    - a. Fully replicated: the entire state is available on every node in the cluster.
    - b. Secure: implements automatic TLS with optional client-certificate authentication.
    - c. Fast: benchmarked at 10,000 writes per second.

# Components of Control Plane (master):

## 3. Kube scheduler:

- When users make request for the creation and management of PODs, kube scheduler is going to take action on these requests.
- Handles POD creation and management.
- Kube scheduler match/ assign any node to create and run PODs.
- A scheduler watches for newly created PODs that have no node assigned. For every POD that the scheduler discovers the scheduler becomes responsible for finding best node for that POD to run on.
- Scheduler gets the information for hardware configuration from configuration file and schedules the PODs on nodes accordingly

## 4. Controller Manager:

- Make sure that actual state of cluster matches the desired state.
- Two possible choices for controller manager:
  - a. If k8s on cloud, then it will be cloud-controller-manager.
  - b. If k8s on non-cloud then it will be kube-controller-manager



## Components on master that runs controller:

- a. **Node controller:** for checking the cloud providers to determine if a node has been detected in the cloud after it stops responding.
- b. **Route controller:** responsible for setting up network, routes on your cloud.
- c. **Service controller:** responsible for load balancers on your cloud against services of type load balancers.
- d. **Volume controller:** for creating, attaching and mounting volumes and interacting with the cloud provider to orchestrate volume.



# Nodes (kublet and container engine.)

- Node is going to run 3 important pieces of software/ process.
- **1. Kublet**
- **2. Container engine**
- **3. Kubeproxy**
- 1. Kublet:
  - Agent running on the node.
  - Listens to Kubernetes master. (e.g-POD creation request)
  - Use port 10255.
  - Send success/fail reports to master.
- 2. Container Engine:
  - Works with kublet.
  - Pulling images.
  - Start/ stop containers.
  - Exposing containers on ports specified in manifest.
- 3. Kubeproxy:
  - Assign IP to each POD.
  - It is required to assign IP address to PODs (dynamic).
  - Kube-proxy runs on each node and this make sure that each POD will get its own unique IP address. These 3 components collectively called NODE.





## POD:

- ➤ Smallest unit in Kubernetes.
- ➤ POD is a group of one or more containers that are deployed together on the same host.
- ➤ A cluster is a group of nodes.
- ➤ A cluster has at least one worker node and master node.
- ➤ In Kubernetes the control unit is the POD, not containers.
- ➤ It consists of one or more tightly coupled containers.
- ➤ POD runs on node which is control by master.
- ➤ Kubernetes only knows about PODs (does not know about individual container).
- ➤ Cannot start containers without a POD.
- ➤ One POD usually contains one container.
- **Multi container PODs:**
- ➤ Share access to memory space.
- ➤ Connect to each other using local host <container port>.
- ➤ Share access to the same volume.
- ➤ Containers within POD are deployed in an all-or-nothing manner.
- ➤ Entire POD is hosted on the same node (scheduler will decide about which node).



# POD

- **POD limitations:**
  - No auto healing or auto scaling.
  - POD crashes.
- **Higher level Kubernetes objects:**
  - a. Replication set: auto scaling and auto healing
  - b. Deployment: versioning and rollback
  - c. Service:
  - d. Static (non-ephemeral) IP and networking.
  - e. Volume: non-ephemeral storage.
- Important:
  - Kubectl- single cloud
  - Kubeadm- on premise
  - Kubefed- federated

# Kubernetes Objects

- Anything which has shape and size is called Object.
- In Kubernetes, Work is called Objects e.g. in Manifest file we write, this is our desired state, we want this. It's called Object.
- Object can be Container creation, Open port etc.
- Kubernetes uses object to represent state of the cluster, Which Containerized applications are running and on which node.
- Policies around how these applications behave such as restart policies, upgrades and fault tolerance.
- Once object is created, Kubernetes system ensures Object exists and maintains.
- Every K8s objects includes two nested that govern object config.
- 1-The Spec- Describes the desired state.
- 2-The Status-Actual state of object ,supplied and updated by K8s System

# Basic K8s Objects

- 1-Pod
- 2-Replicaset
- 3-Deployment
- 4-Service
- 5-Volumes
- 6-Configmaps
- 7-Secrets
- 8-Namespace
- 9-Jobs
- 10-Daemon Sets

# Relationship Between Objects

- Pod manages Containers.
- Replicaset manages Pods.
- Deployment Help Pod to Rollback to previous state.
- Services Expose Pod to outside world(expose to Internet)
- Configmaps and Secrets help to configure Pods.
- K8s objects are created by us, and we push them to K8s API through Kubectl.
- Kubectl is a command line Tool which supports several different ways to create and manage K8s Objects.
- Commands can be written in either Declarative or Imperative method.
- Declarative method used in Prod projects stating what trying to achieve without telling how to.
- Imperative method used in Dev Projects explicitly telling how to accomplish.

# POD-Features

- When a POD is created, its scheduled to run in our cluster.
- POD remains in that Node until
- A-Process is terminated.
- B-POD object is deleted.,
- C-POD is evicted for lack of resources.,
- D-Node fails.
- If Node dies, PODS scheduled to that node are scheduled for deletion after timeout period.
- A given POD is not rescheduled to a new node , instead replaced with a new node even with same name but different UID.
- Volume in a pod will exist until Pod exists , if that POD is deleted, Volume also destroyed.
- A controller can create multiple Pods , manage ,handle replication, rollout provide self healing capabilities.



# K8s Configurations

## ➤ All In One Single Node Installation:-

With All In One, both Master and Worker are installed on single node. Useful for Learning and Development not to be used for Production.

**Minikube** is one such example.

## Single Node etcd,Single Master and multi-worker installation:-

In this we have single master node which runs on single etcd instance ,multiple workers are connected to Master node.

## Single node etcd,Multi Master and multi worker- Installation:-

**24\*7 Availability**,here we have multiple masters for HA(high availability)mode but we have a single node etcd instance. Multiple workers are connected to Master.

# ====To create a pod=====

- Start Minikube
- \$ apt install conntrack
- \$ minikube start --vm-driver=none
- \$ minikube status

vi pod.yml

- **kind: Pod** ----->Object of type POD
- **apiVersion: v1**
- **metadata:**
- **name: testpod** ----->name of the pod
- **spec:** ----->what is desired is written under spec
- **containers:**
- **- name: c00**
- **image: ubuntu**
- **command: ["/bin/bash", "-c", "while true; do echo Hello-Ashok; sleep 5 ; done"]**
- **- name: c01**
- **image: httpd**
- **ports:**
- **- containerPort: 80**

## K8s Commands To Create Pod

- `Kubectl apply -f pod.yml`
- o/p --- pod/testpod created
- `Kubectl get pods`
- o/p testpod 1/1 running
- `Kubectl describe pod testpod`
- Above to see where running
- `Kubectl logs -f testpod`
- `Kubectl get pods -o wide` -To See details
- If required to delete pod
- `Kubectl delete -f pod1.yml`
- For Multi-container,
- `Kubectl logs -f testpod3--`→error
- `Kubectl logs -f testpod3 -c cool`
- `Kubectl exec testpod3 -it -c c01 -- /bin/bash`

# Labels, Selectors, Replication controller, Replica Set and Node Controller

- Labels are the mechanism to organise K8's objects.
- A label is key-value pair without pre-defined meaning that can be attached to objects.
- So we are free to choose labels to refer environment- Dev or Prod or test.
- Multiple labels can be attached to single objects.
- Vi pods.yml

kind:pod

apiversion:V1

Metadata:

name:delhiprod

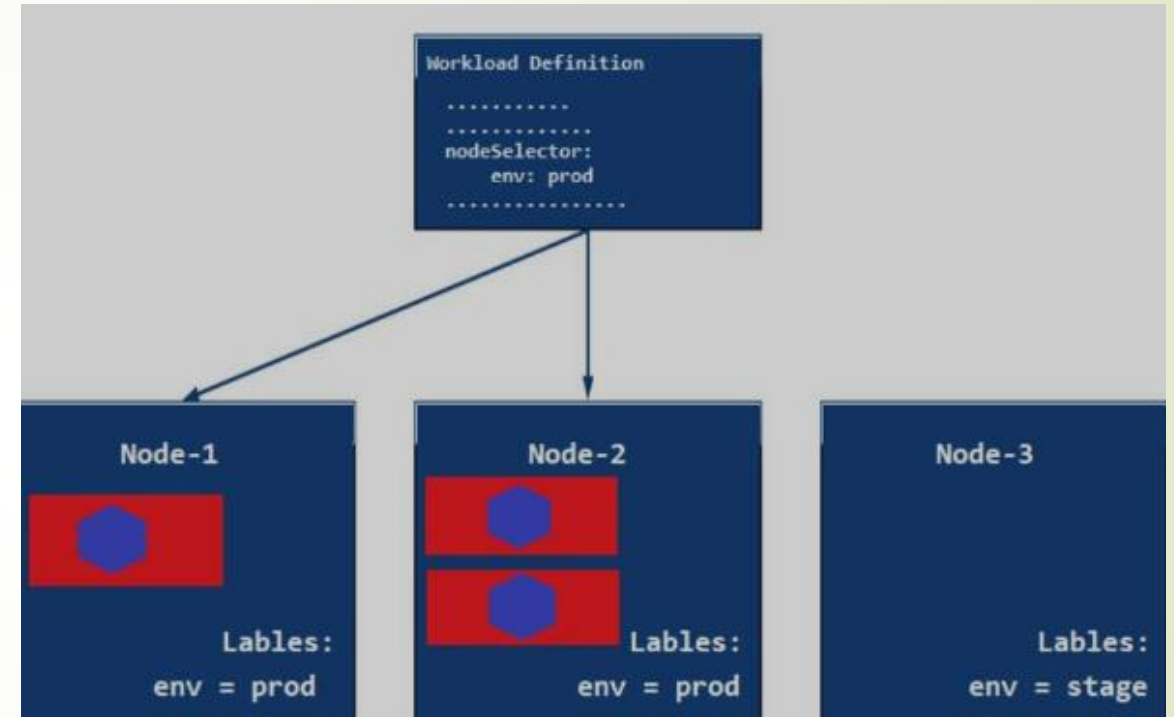
labels:

env: development

class: pods

Name: Ashok

company: CBA



# K8s Label Commands

- `kubectl get pods - - show -labels`
- Now,if we want to add label to existing pod.
- `kubectl label pods delhiprod myname=Ashok`
- Now,list pods matching a label
- `kubectl get pods -l env=development`
- Now list pods where label is not development
- `kubectl get pods -l env!= development`
- Now ,if we want to delete using labels
- `kubectl delete pod -l env!= development`
- `Kubectl get pods`

# Selectors

- Unlike name/UIDs ,Labels do not provide uniqueness as in general,we can expect many objects to carry same label.
- Once labels are attached to an object, we may need filters to narrow down and these are called label-selectors.
- Api currently supports two types of selectors:-
- Equality based and Set-based
- Equality based= C= ,!=
- The Label selector is made up of multiple requirements which are comma separated.
- Set-based(in,not in,exists)
- Key→env in(Dev,prod)
- Key→env not in(team1,team2)
- K8s also supports set based selectors
- Match multiple values
- `Kubectl get pods -l 'env in(development,testing)`
- `Kubectl get pods -l class=pods,myname=Ashok`



# Node-Selector

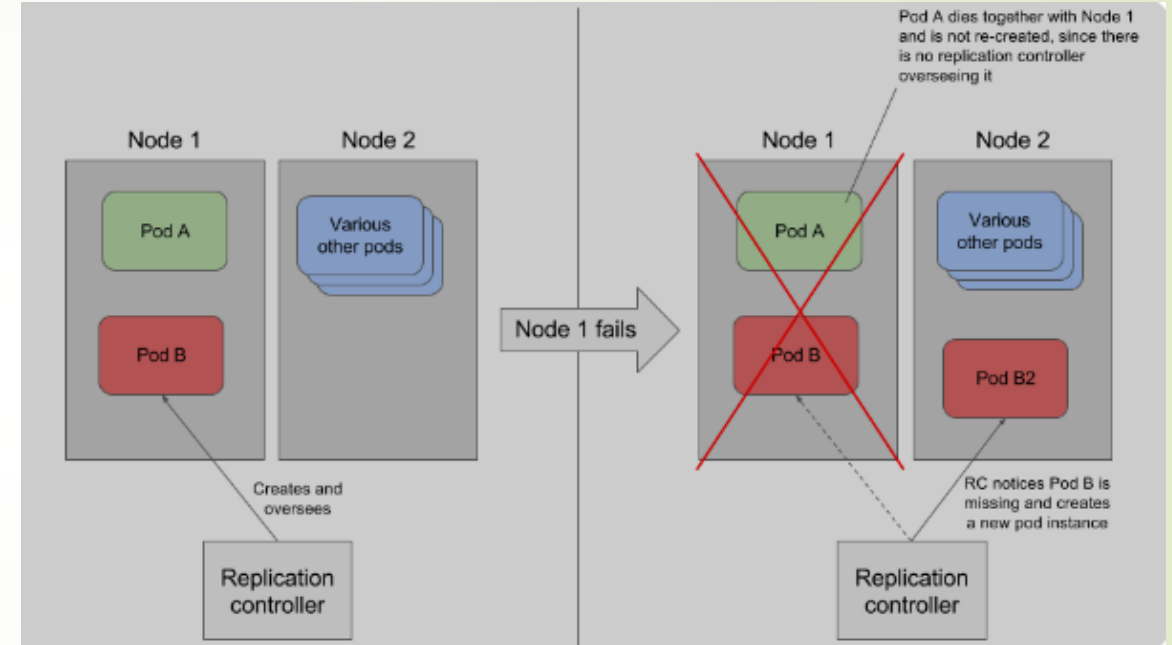
- If we want pod should be created on specific node e. g t2.medium,specifically mark hardware =t2.medium.
- If all are t2.medium,wherever master finds resources are more available, it creates there.
- One use case for selecting labels is to contain set of nodes onto which pod can schedule,we can tell POD to run on a particular node.
- Generally, such constraints are unnecessary as scheduler will automatically do reasonable placement but sometimes, we may need.
- We can use labels to tag nodes.
- Once nodes are tagged ,we can use label-selectors to specify Pods to run only on specific nodes.
- First, we give label to nodes.
- Then we use node selector for Pod configuration.

# Scaling and Replication(HA,24\*7)

- Kubernetes is designed to orchestrate multiple containers and Replication.
- Need for multiple containers/replication helps us with these
- **Reliability**→ By having multiple versions of an application, we can prevent problems ,if one or more fails.
- **Load-Balancing** → Having multiple versions of containers enables to easily send traffic to different instances to prevent over loading in single instances.
- **Scaling**→ when load becomes too much for the number of existing instances, Kubernetes enables to easily scale up application adding instances as needed.
- **Rolling updates**→ updates to a service by replacing pods one by one.
- Replicas 2→ 2 ditto pods created
- Replicas 3→3 ditto pods created

# Replication Controller

- A Replication controller is an object which enables us to create multiple pods to make sure that desired number of pods always exist.
- A pod created by Replication controller will be automatically replaced by a new pod in case pods get crashed, failed or terminated.
- RC is recommended if we want to make sure atleast 1 pod always exist even after system reboots.
- We can run the RC with 1 replica and RC will make sure the POD is always running.



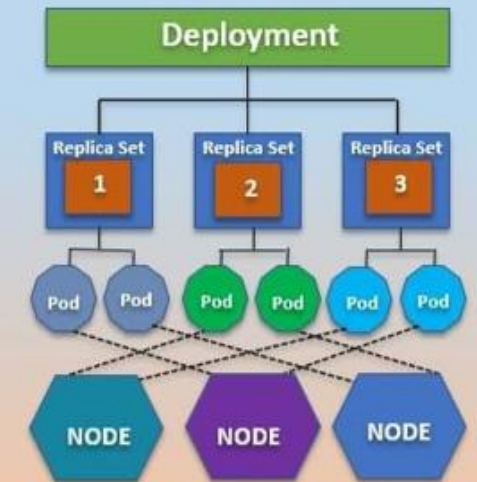
# Replication Controller

- **kind:Replication Controller** -----→defines to create object of type RC
- **apiversion: v1**
- **metadata:**
  - **name: myreplicas** -----→Name of replica
- **spec:** -----→What we desire
  - **replicas:3** -----→no of desired pods
  - selector:** -----→ RC gets POD created on that node
    - myname: Ashok** -----→must match labels
  - template:** -----→defines template to launch POD
    - metadata:**
      - name: testpod**
      - labels: Ashok**
    - spec:**
      - containers:**
        - name: C00**
        - image: ubuntu**
        - command: ["/bin/bash", "-c", "while true; do echo Hello-Ashok; sleep 5 ; done"]**

## Deployment and Roll-Out and Rollback

- How to come back to previous state?
- A deployment object gives capability and control over how and when a new POD created can be rolled out, updated or Rolled back to previous state.
- Deployment is upper-level object than RS and PODS.
- When using Deployment object, we first define state of the app then K8s cluster schedules that app on specific node.
- K8s then monitors, if node hosting an instance goes down or POD is deleted then Deployment replaces providing self healing mechanism

## Kubernetes Deployment





# Use Cases of Deployment

- Create a Deployment to roll out a replica set.
- Declare a new state of pod → By updating the Pod template spec of deployment a new replica set is created and the deployment manages moving the pods from old replica set to new one at a controlled rate. Each new Replica set updates the revision of deployment.
- Rollback to an earlier Deployment revision
- Scale up the Deployment to facilitate more load.
- Pause the Deployment to apply multiple fixes to its Pod template spec and then resume
- Clean up older Replica set that's not required any more.
- We can roll back to a specific version- by specifying it with - -to-revision
- `Kubectrl rollout undo deploy/mydeployment --to-revision=2`
- The name of replicaset is always formatted as [Deployment-name]-[Random string]





## *Inspection of Deployment in Cluster*

- When we inspect Deployment in our cluster and give below command
- `kubectl get deploy`
- Following Fields will appear
- **Name** → List the name of Deployment in namespaces
- **Ready** → Display how many replicas of application are available to your users.
- **Up-To-Date** → Display the number of replicas that have been updated to achieve the desired state.
- **Available** → Displays how many replicas of application are available to users
- **Age** → Display the amount of time application was running.

# Deployment-Example

- kind: Deployment
- apiVersion: apps/v1
- metadata:
  - name: mydeployments
- spec:
  - replicas: 1
  - selector: # tells the controller which pods to watch/belong to
  - matchLabels:
    - name: deployment
  - template:
    - metadata:
      - name: testpod1
    - labels:
      - name: deployment
    - spec:
      - containers:
        - - name: c00
        - image: httpd
        - ports:
          - - containerPort: 80



## Deployment Commands

- To Check Deployment was created or not.
- `kubectl get deploy`
- To check how deployment creates RS and PODS
- `kubectl describe deploy mydeployments`
- `kubectl get rs`
- To Scale up or down
- `kubectl scale --replicas=1 deploy mydeployments`
- To scale up or down
- `kubectl scale --replicas=1 deploy mydeployment`
- To check what is running inside container
- `Kubectl logs -F <podname>`
- `Kubectl rollout status deployment mydeployments`
- `Kubectl rollout history deployment mydeployments`
- `Kubectl rollout undo deploy mydeployments`

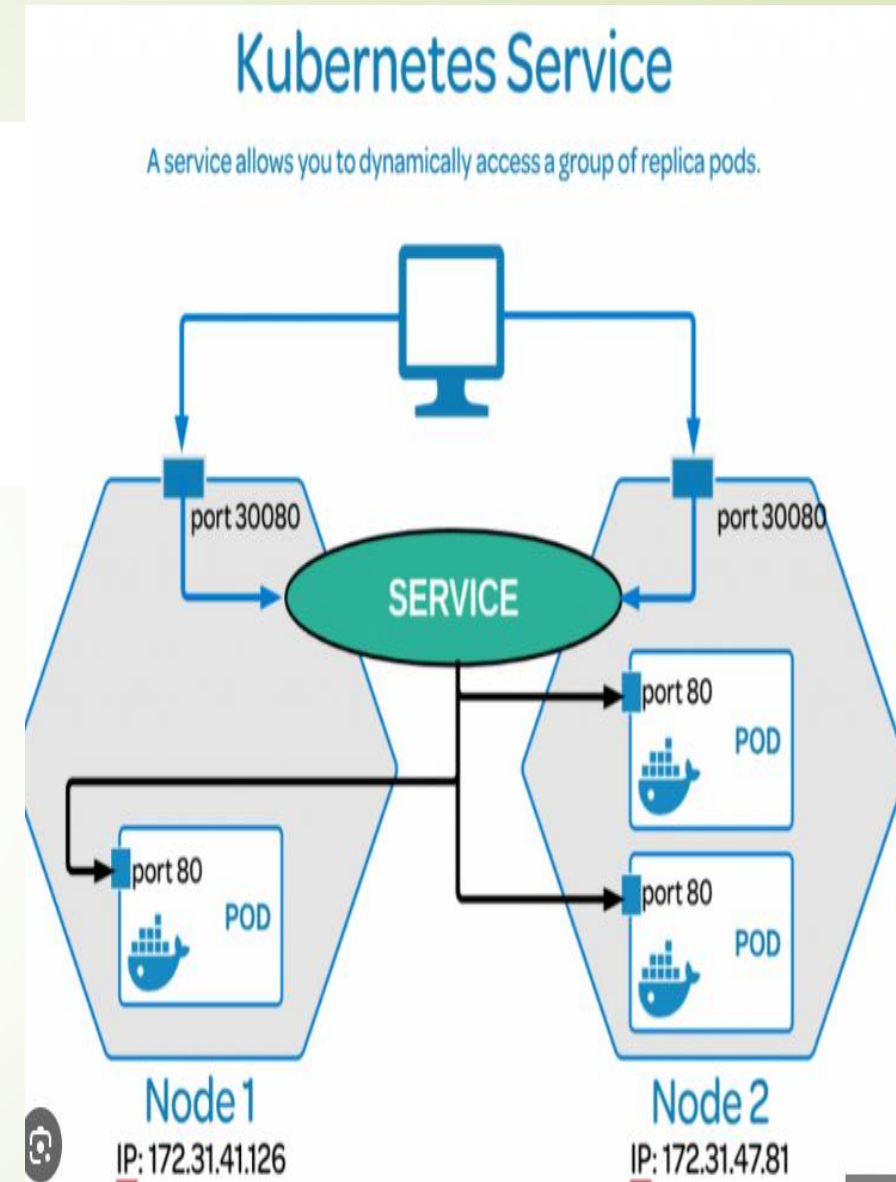
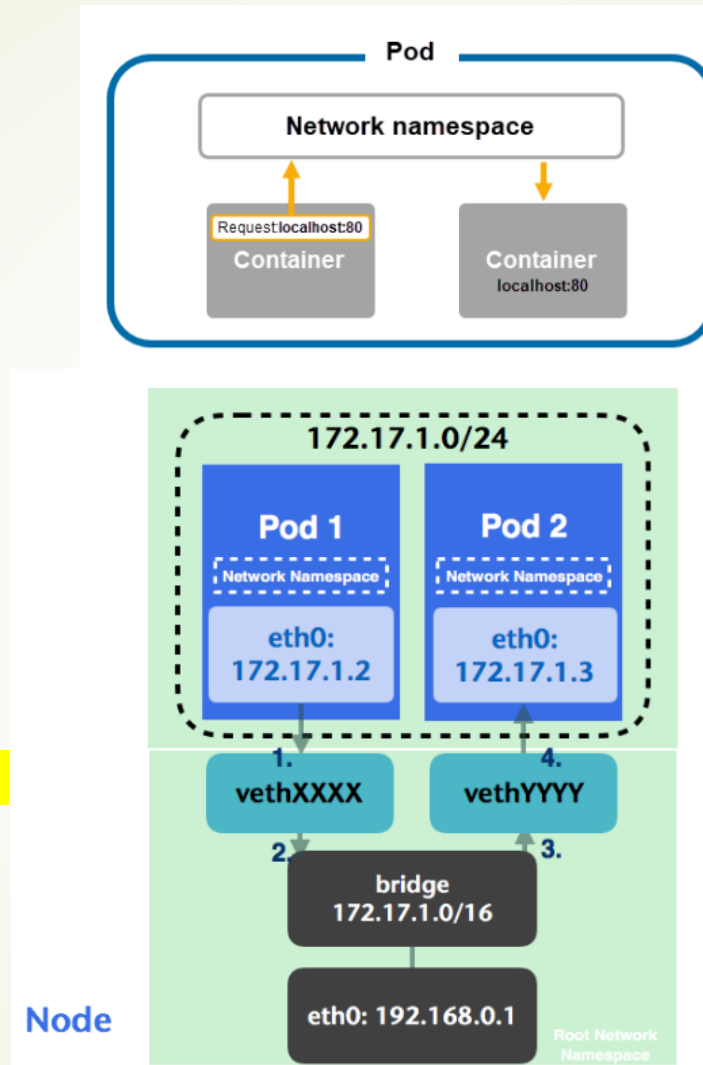


# *Failed Deployments*

- Deployment may get stuck trying to deploy its newest replicaset without completing.
- 1→Insufficient Quota
- 2→Readiness probe failures
- 3→Image pull errors
- 4→Insufficient permissions
- 5→Limit Ranges
- 6→Application run time misconfiguration

# K8s Networking, Services, Node port

- Kubernetes Networking addresses four major concerns.
- 1→Container to Container within a pod use networking to communicate through loopback(localhost).
- 2→Cluster Networking provides communication between different pods
- 3→The service object lets to expose app running on pod to be reachable from outside world.
- 4→We can also use services to publish services only for consumption inside cluster.



# K8s Networking, Services

- 1-Container to Container communication happens through localhost-within the container.
- Pod To Pod Communication on same worker node happens through POD IP.
- By default , POD IP will not be accessible from outside world.
- Object Services applied above RS to provide Virtual IP.
- Each POD gets its own IP Address.
- However, in deployment, the set of PODS running at one moment could be different from set of PODS running moment later, similarly RS creates new POD to maintain minimum value 1,so PODS can be different at different moment.
- This results in problem, if some set of PODS(call them 'Backend') provide functionality to other PODS(call them 'frontend'), inside cluster, how frontend pods keep a track of backend pods, through which IP ,they can contact.



# K8s Networking, Services

- When using RS ,PODS are terminated and created during scaling or replication operations.
- Similarly ,while using Deployments, while updating image version, PODS are terminated, and new PODS take place of other PODS.
- So, **PODS are very dynamic in nature** they come and go and get created on nay node ,so difficult to access them as IP changes every time.
- **Service objects are logical bridge between pods and end users which provides virtual IP(VIP)**
- **The VIP is not Actual IP connected to a N/W but its purpose is to forward traffic to one or more pods.**
- Kube-proxy keeps mapping between VIP and POD up-to-date with queries to API servers to learn about new services in the cluster.
- Although each POD has unique IP but they are not exposed to outside world.
- **Services help to expose the VIP mapped to the ports and allows application to receive traffic.**

# K8s Networking, Services

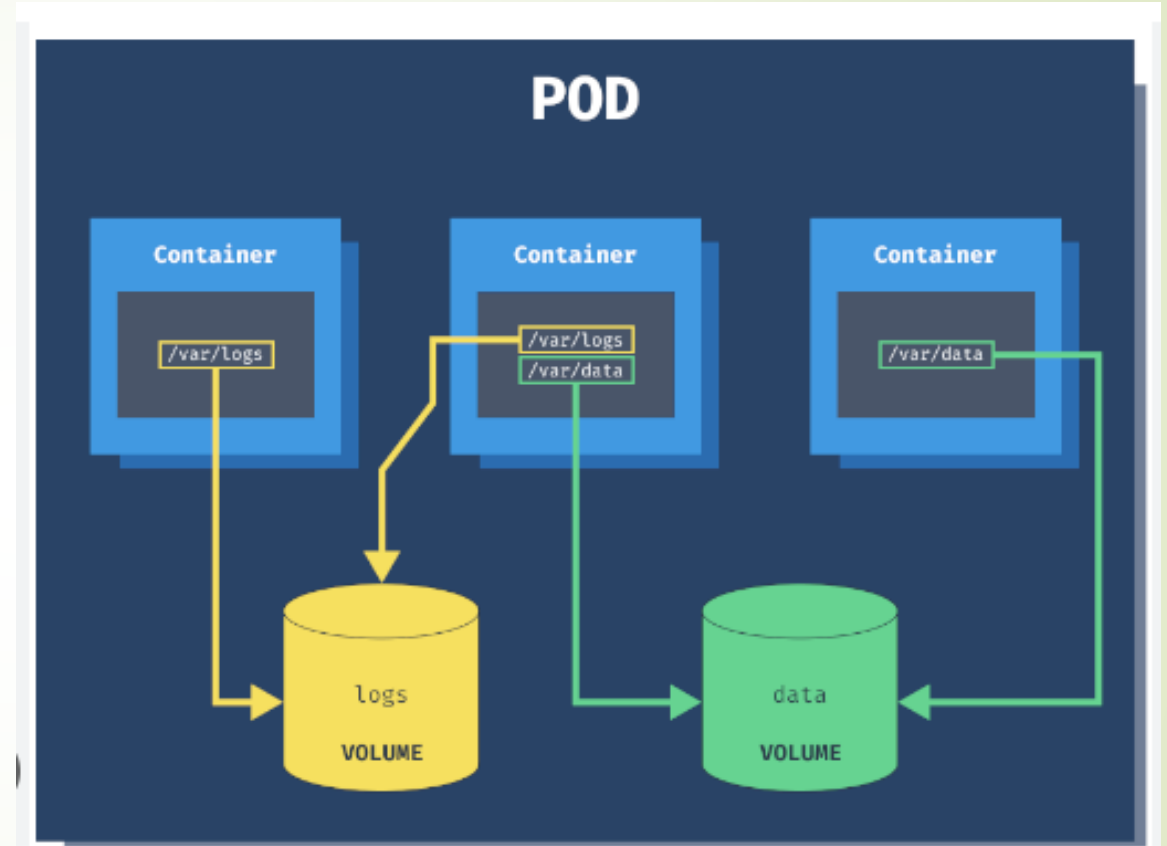
- Labels are used to select which are the PODS to be put under a service.
- Creating a service will create an end point to access the PODS/applications in it.
- Services can be exposed in many ways by specifying type in service spec.
- 1→Cluster IP(default)
- 2→Nodeport
- 3→LoadBalancer created by cloud providers they will route external traffic to every node(e.g. ELB),by default services can run between ports(30,000-32767),the set of pods target determined by selector.
- Cluster-IP exposes VIP only reachable from within the cluster.
- Mainly used to communicate between the components of micro-services
- Nodeport is applied above Cluster-IP and and Load-Balancer service type is on top.
- Nodeport makes a service accessible from outside cluster
- Exposes services on the same port of each selected Node in cluster using NAT.Node gets mapped to VIP,if AWS EC2 has public DNS,Nodeport between(30,000-32,767) will be given.

# K8s Services-Lab

- `kind: Service` # Defines to create Service type Object
- `apiVersion: v1`
- `metadata:`
- `name: demoservice`
- `spec:`
- `ports:`
- `- port: 80` # Containers port exposed
- `targetPort: 80` # Pods port
- `selector:`
- `name: deployment` # Apply this service to any pods which has  
the specific label
- `type: ClusterIP` # Specifies the service type i.e ClusterIP or  
`NodePort`
- `$ kubectl get svc`

# Volumes

- Containers are short lived in nature.
- All data stored inside a container is deleted ,if container crashes. however, Kubernetes system will restart it , which means no old data.
- To overcome this,K8s uses volume. A volume is a directory backed by storage medium and its content is determined by volume type.
- In K8s volume is attached to a pod and shared among the containers of that POD.
- The Volume has the same life span as of POD and it outlives that of a container ,this allows the data to be preserved in case of container restarts.
- A Volume type decides the property of a directory like size , content etc e.g.
- Node local type such as emptydir and host path.
- Filesharing types like nfs
- Cloud provider specific types like AWS EBS,Azure etc.
- Distributed file system types e.g. glusterfs or cephfs.
- Special purpose types like secret,gitrepo



# volume labs

- apiVersion: v1
- kind: Pod
- metadata:
- name: myvolemptydir
- spec:
- containers:
- - name: c1
- image: centos
- command: ["/bin/bash", "-c", "sleep 15000"]
- volumeMounts:                      # Mount definition inside the container
- - name: xchange
- mountPath: "/tmp/xchange"
- - name: c2
- image: centos
- command: ["/bin/bash", "-c", "sleep 10000"]
- volumeMounts:
- - name: xchange
- mountPath: "/tmp/data"
- volumes:
- - name: xchange
- emptyDir: {}



# *EmptyDir*

- We use this when we want to share content among different containers of same POD and not to host ,machine.
- An EmptyDir is first created when a POD is assigned to a node and exists as long as POD is running on that node.
- AS name suggests, its empty initially.
- Containers in the POD can all read and write the same files in the emptyDir volume ,although that volume can be mounted at the same and different path in each containers.
- When the POD is removed from a node for any reason , data mounted on EmptyDir is also removed forever.
- Container crashes does not remove POD ,hence data in emptyDir is safe across container crashes.



# HostPath

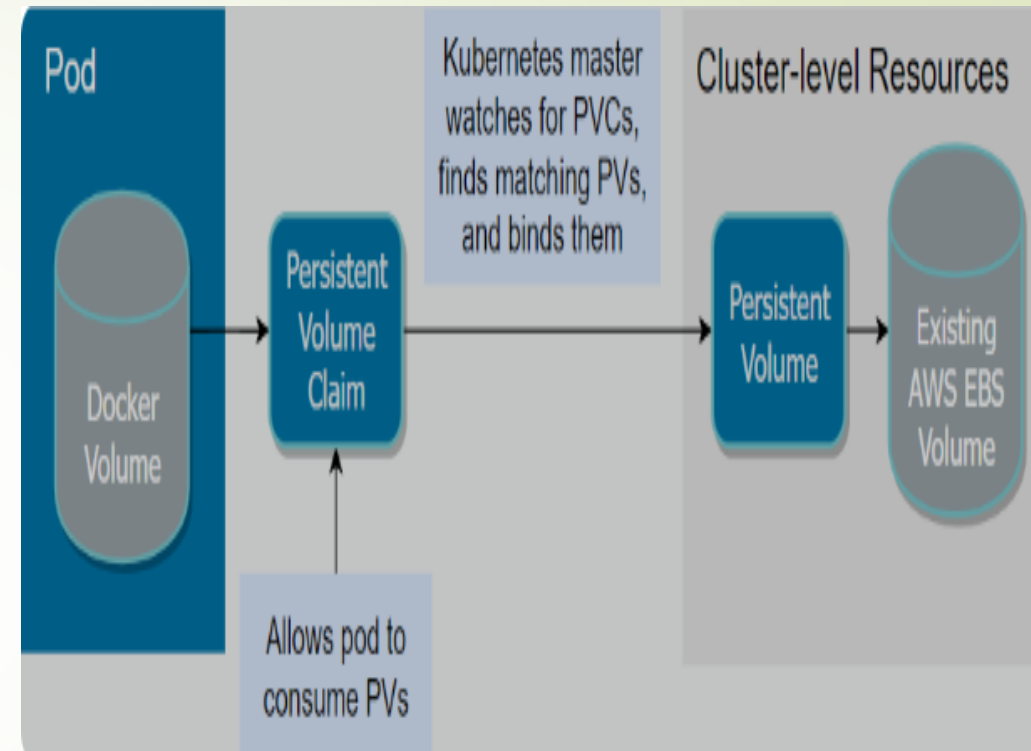
- We use this when we want to access the content of POD/Container from host machine.
- A hostpath volume mounts a file or directory from host nodes filesystem into your POD.
- Create a volume and map it to hostpath
- **apiVersion: v1**
- **kind: Pod**
- **metadata:**
- **name: myvolhostpath**
- **spec:**
- **containers:**
- **- image: centos**
- **name: testc**
- **command: ["/bin/bash", "-c", "sleep 15000"]**
- **volumeMounts:**
- **- mountPath: /tmp/hostpath**
- **name: testvolume**
- **volumes:**
- **- name: testvolume**
- **hostPath:**
- **path: /tmp/data**

# Persistent Volume and Liveness Probe

- In IT Sector , Storage is managed by Storage admin not by users.
- In containerized world, K8s resolves the problem of handling Volume types with help of Persistent Volume(PV) subsystem.
- PV is not backed by locally attached storage on a worker node but by Network attached storage system such as EBS or EFS or distributed file system like Ceph.
- K8s provides API for users and administrators to manage and consume resources.
- To manage the volume, it uses the Persistent volume API resource type.
- And to consume it, uses the persistent –volume-claim API resource type.
- Normally, life of a volume exists till life of POD, if POD dies, RS, Deployment automatically creates a new POD in any node and volume associated with old POD , goes with it.
- Solution is connect the two nodes with EBS and mount the path of POD to EBS to get volume shared even if POD dies.

# Persistent Volume Claim

- In order to use a PV ,we need to claim it first using Persistent Volume claim.
- The PVC request a PV with desired specification(size,access modes,speed etc) and once a suitable persistent volume is found,it is bound to persistent volume claim.
- After a successful bound to a POD , we can mount it as a volume.
- Once a user finishes its work , the attached Persistent Volume can be released, the underlying PV can be reclaimed and recycled for future usage.
- AWS EBS Volume mounts EBS Volume into our POD unlike emptyDir contents of EBS Volume are preserved and volume is merely unmounted,Instance storage is non-persistent.
- There are a few restrictions.
- 1→Same region and Availability Zone.
- 2→EBS only Supports a single EC2 instance mounting a volume.





# ***PERSISTENT VOLUME***

- **apiVersion: v1**
- **kind: PersistentVolume**
- **metadata:**
- **name: myebsvol**
- **spec:**
- **capacity:**
- **storage: 1Gi**
- **accessModes:**
- **- ReadWriteOnce**
- **persistentVolumeReclaimPolicy: Recycle**
- **awsElasticBlockStore:**
- **volumeID:       # YAHAN APNI EBS VOLUME ID DAALO**
- **fsType: ext4**



# ***PERSISTENT VOLUME***

- **apiVersion: v1**
- **kind: PersistentVolumeClaim**
- **metadata:**
  - **name: myebsvolclaim**
- **spec:**
  - **accessModes:**
    - **- ReadWriteOnce**
  - **resources:**
    - **requests:**
      - **storage: 1Gi**

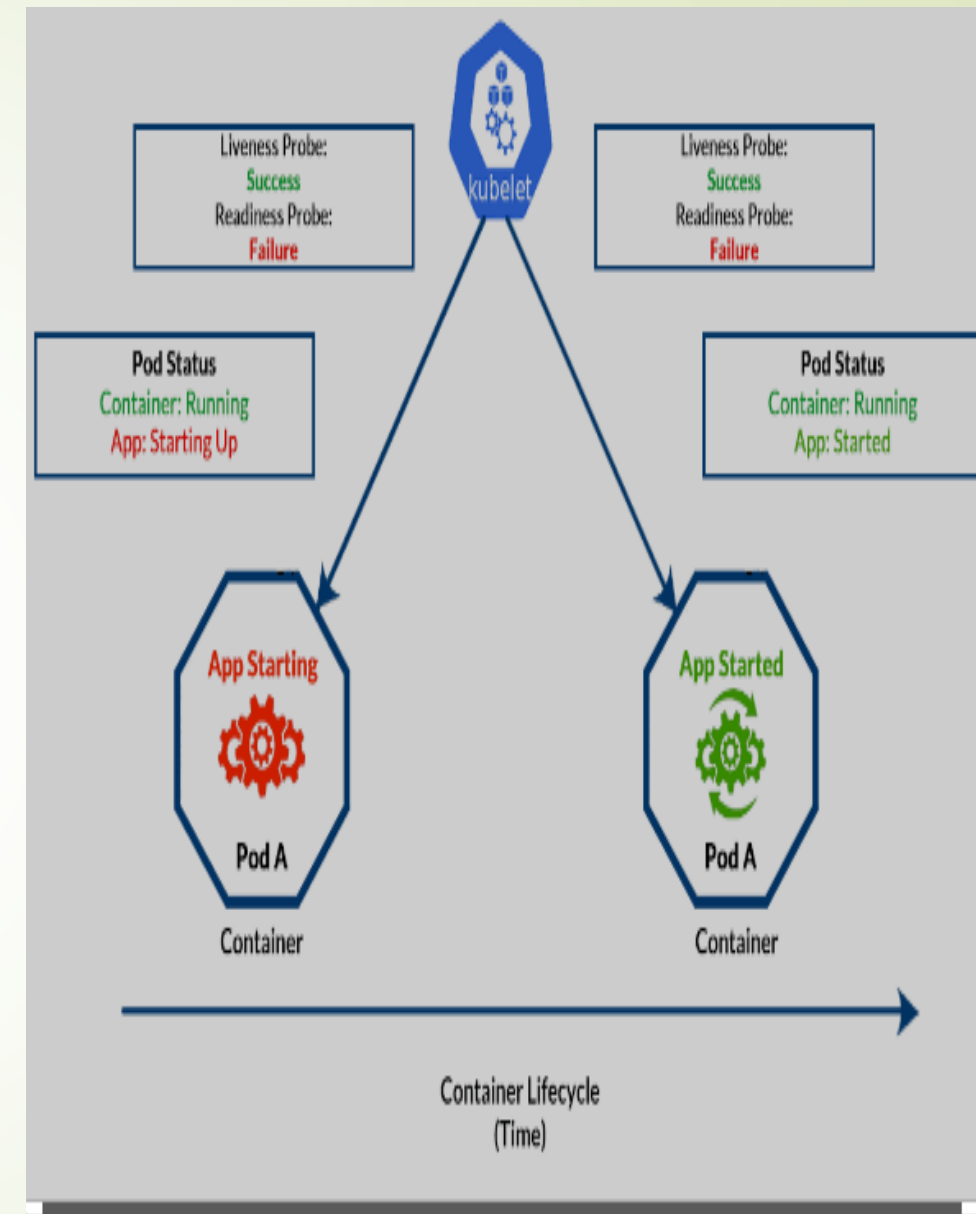
# PERSISTENT VOLUME

- `apiVersion: apps/v1`
- `kind: Deployment`
- `metadata:`
  - `name: pvdeploy`
- `spec:`
  - `replicas: 1`
  - `selector:` # tells the controller which pods to watch/belong to
    - `matchLabels:`
      - `app: mypv`
  - `template:`
    - `metadata:`
      - `labels:`
        - `app: mypv`
    - `spec:`
      - `containers:`
        - `- name: shell`
        - `image: centos`
        - `command: ["bin/bash", "-c", "sleep 10000"]`
        - `volumeMounts:`
          - `- name: mypd`
          - `mountPath: "/tmp/persistent"`
      - `volumes:`
        - `- name: mypd`
        - `persistentVolumeClaim:`
          - `claimName: myebsvolclaim`



# Health check and Liveness Probe

- A Pod is considered ready when all the containers are ready.
- In order to verify readiness of containers in POD to send traffic K8s provides a range of health check mechanism.
- Health checks are carried out by kubelet to determine when to re-create a container for liveness probe and used by Services and Deployment to determine if PODS can send/receive traffic.
- Liveness probe could catch a deadlock when application running but unable to make a progress, restarting that app could help to make application more available.
- One use of Probes(readiness) is to control which PODS are used as backends for services when POD is not ready, its removed from Service Load balancers.
- Commands for Health check return 0, kubelet considers containers alive in this case. For non-zero value, it kills and re-creates it.

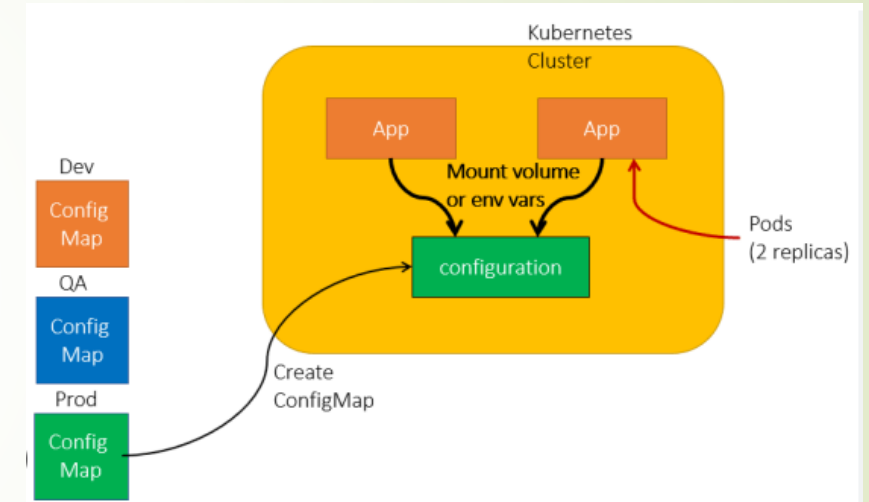


# HEALTHCHECK/LIVENESSPROBE

- `apiVersion: v1`
- `kind: Pod`
- `metadata:`
  - `labels:`
    - `test: liveness`
  - `name: mylivenessprobe`
- `spec:`
  - `containers:`
    - `- name: liveness`
    - `image: ubuntu`
    - `args:`
      - `- /bin/sh`
      - `- -c`
      - `- touch /tmp/healthy; sleep 1000`
  - `livenessProbe:` `# define the health check`
    - `exec:`
      - `command:` `# command to run periodically`
        - `- cat`
        - `- /tmp/healthy`
    - `initialDelaySeconds: 5` `# Wait for the specified time before it runs the first probe`
    - `periodSeconds: 5` `# Run the above command every 5 sec`
    - `timeoutSeconds: 30`

# Configmap and Secrets

- While performing app deployments on K8s cluster we may need to check the app configuration file depending on the environment like Dev, QA, Stage or prod.
- Changing the application configuration means we need to change source code, commit change, create a new image and then go through complete deployment process.
- Here, these configuration files should be de-coupled from image content in order to keep containerised application portable.
- This is where K8s configmaps comes handy , it allows us to handle configuration files much more efficiently.
- Configmaps are used to store and share non-sensitive, unencrypted information use secrets otherwise.
- Configmaps can be used to store fine-grained information like individual properties or entire configuration file.
- Configmapfiles are not good replacement for properties file.
- Configmap files can accessed in following ways
  - 1→As environment variable
  - 2→As Volume in pod
- `kubectl configmap<mapname>- - from-file=<fileto read>abc.cnf`



# Configmap

- apiVersion: v1
- kind: Pod
- metadata:
- name: myvolconfig
- spec:
- containers:
- - name: c1
- image: centos
- command: ["/bin/bash", "-c", "while true; do echo Hi Ashok; sleep 5 ; done"]
- volumeMounts:
- - name: testconfigmap
- mountPath: "/tmp/config" # the config files will be mounted as ReadOnly by default here
- volumes:
- - name: testconfigmap
- configMap:
- name: mymap # this should match the config map name created in the first step
- items:
- - key: sample.conf
- path: sample.conf

# Configmap

- ▶ `apiVersion: v1`
- ▶ `kind: Pod`
- ▶ `metadata:`
  - ▶ `name: myenvconfig`
- ▶ `spec:`
  - ▶ `containers:`
    - ▶ `- name: c1`
    - ▶ `image: centos`
    - ▶ `command: ["/bin/bash", "-c", "while true; do echo Technical-Guftgu; sleep 5 ; done"]`
    - ▶ `env:`
      - ▶ `- name: MYENV      # env name in which value of the key is stored`
      - ▶ `valueFrom:`
        - ▶ `configMapKeyRef:`
          - ▶ `name: mymap      # name of the config created`
          - ▶ `key: sample.conf`

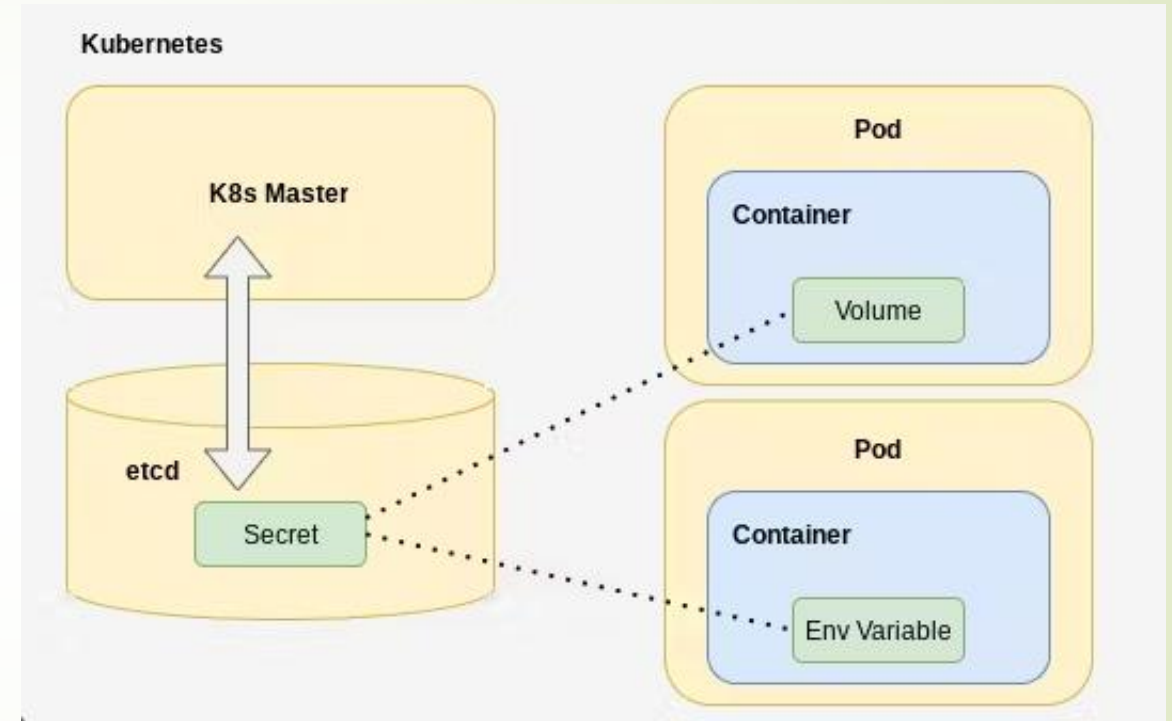
# Secrets-Lab

- `echo "root" > username.txt; echo "password" > password.txt`
- `kubectl create secret generic mysecret --from-file=username.txt --from-file=password.txt`
- `apiVersion: v1`
- `kind: Pod`
- `metadata:`
  - `name: myvolsecret`
- `spec:`
  - `containers:`
    - `- name: c1`
    - `image: centos`
    - `command: ["/bin/bash", "-c", "while true; do echo Hi Ashok; sleep 5 ; done"]`
    - `volumeMounts:`
      - `- name: testsecret`
      - `mountPath: "/tmp/mysecrets" # the secret files will be mounted as ReadOnly by default here`
  - `volumes:`
    - `- name: testsecret`
    - `secret:`
      - `secretName: mysecret`



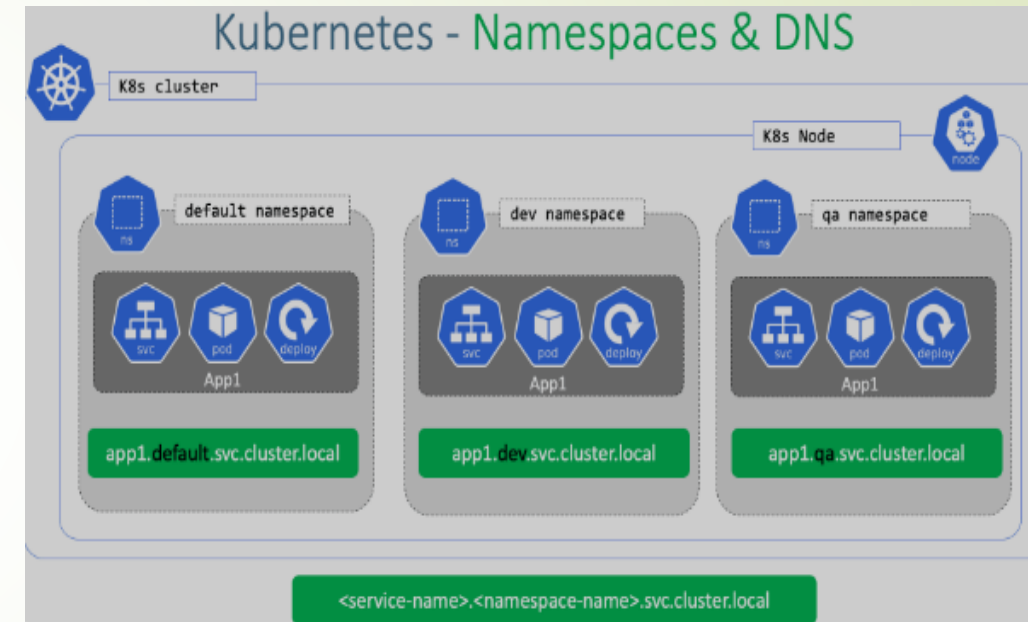
# Secrets

- We don't want sensitive information like database password or an API key kept around in clear text.
- We can access them via volume or environment variable from a container running in pod
- Secret data is stored on nodes in tmpfs volumes(tmpfs)is a file system which keeps file in virtual memory.
- The API server stores secrets as plaintext in etcd.
- Secrets can be created from:-
  - 1—from text file
  - 2→from a yaml file.



# Namespaces, Limits and Requests

- We can name our objects but if many are using same object then it would be difficult to manage.
- A namespace is a group of related elements that have a unique name or identifier.
- Each project has a namespace inside which pods are created , If project B has namespace , project A can't see project B without namespace.
- If no namespace is defined for a cluster then default namespace is given i.e. default namespace.
- If we give command `kubectl get pods` and no pod is present then it gives output –"No resource found in default namespace"
- It's area or folder of my project where I m working.
- We can define Resource quota for namespace,how much CPU, RAM,Storage is defined for them,so that resource wastage doesn't happen.
- It provides a mechanism to attach Authorization and policy to sub section of the cluster.
- We can use Resource Quota on specifying how much resource each namespace can use.



# Namespaces, Limits and Requests

- Most K8s resources(Pods, Services, Replication-controller) are in same namespace and low level resources such as nodes and persistent volume are not in any namespace.
- Namespaces are intended for use in environments with many users spread across multiple teams or projects. For clusters with a few to ten ,number of users, it's not required.
- If we have shared K8s clusters for DEV and Prod use cases , Dev Team would like to maintain space in cluster where they can view a list of PODS, Services, Deployments with no restriction on modification to enable Agile Development.
- For Prod Team, we can enforce strict procedure on who can or cannot manipulate set of PODS,Services and Deployment.
- `$ kubectl config set-context $(kubectl config current-context) --namespace=dev`
- `$ kubectl config view | grep namespace:`

# Manage Compute Resources For Containers

- A POD in K8s will run with no limit on CPU and memory
- We can optionally specify how much CPU and Memory(RAM) each container needs
- Scheduler decides about which nodes to place pods , only if node has enough CPU resources available to satisfy POD CPU requests.
- CPU is specified in terms of cores and memory in unit of bytes
- Two types of constraints can be set for each resource type-
  - 1→Request
  - 2→Limits
- A Request is that amount of a resource which system will guarantee for the container and K8s will use value to decide on which node to place pods
- A Limit is max amount of resources, K8s will allow container to use.
- In the case Request is not set for a container, it defaults to Limits.
- If Limits is not set, then it defaults =0
- CPU Values are specified in 'millicpu' and memory in MB

# Namespaces Lab

- `apiVersion: v1`
- `kind: Namespace`
- `metadata:`
  - `name: dev`
  - `labels:`
    - `name: dev`

- `=====to create a pod=====`
- `vi pod.yml`

- `kind: Pod`
- `apiVersion: v1`
- `metadata:`
  - `name: testpod`
- `spec:`
  - `containers:`
    - `- name: c00`
    - `image: ubuntu`
    - `command: ["/bin/bash", "-c", "while true; do echo Hi Ashok; sleep 5 ; done"]`
  - `restartPolicy: Never`

# Namespaces-Resources Lab

- ▶ apiVersion: v1
- ▶ kind: Pod
- ▶ metadata:
  - ▶ name: resources
- ▶ spec:
  - ▶ containers:
    - ▶ - name: resource
    - ▶ image: centos
    - ▶ command: ["/bin/bash", "-c", "while true; do echo Hi Ashok; sleep 5 ; done"]
    - ▶ resources:
      - ▶ requests:
        - ▶ memory: "64Mi"
        - ▶ cpu: "100m"
      - ▶ limits:
        - ▶ memory: "128Mi"
        - ▶ cpu: "200m"



# Resource Quota

- A K8s cluster can be divided into namespaces, if a container is created in a namespace that has a default CPU, Container does not specify its own CPU Limit, then container is assigned default CPU limit.
- Namespaces can be assigned Resource quota objects, this limits amount of usage allowed to objects in that namespace.
- 1 → Compute
- 2 → Memory (RAM)
- 3 → Storage
- Here are two restrictions that a resource quota imposes on a Namespace
- 1 → Every Container that runs in the name space must have its own CPU Limit.
- 2 → The total amount of CPU used by all containers in the namespace must not exceed a specified limit.

# RESOURCEQUOTA

- apiVersion: v1
- kind: ResourceQuota
- metadata:
  - name: myquota
- spec:
  - hard:
    - limits.cpu: "400m"
    - limits.memory: "400Mi"
    - requests.cpu: "200m"
    - requests.memory: "200Mi"



# RESOURCEQUOTA

- apiVersion: v1
- kind: LimitRange
- metadata:
  - name: cpu-limit-range
- spec:
  - limits:
    - - default:
      - cpu: 1
  - defaultRequest:
    - cpu: 0.5
  - type: Container



# RESOURCEQUOTA

- apiVersion: v1
- kind: Pod
- metadata:
  - name: default-cpu-demo-2
- spec:
  - containers:
    - - name: default-cpu-demo-2-ctr
    - image: nginx
    - resources:
      - limits:
        - cpu: "1"

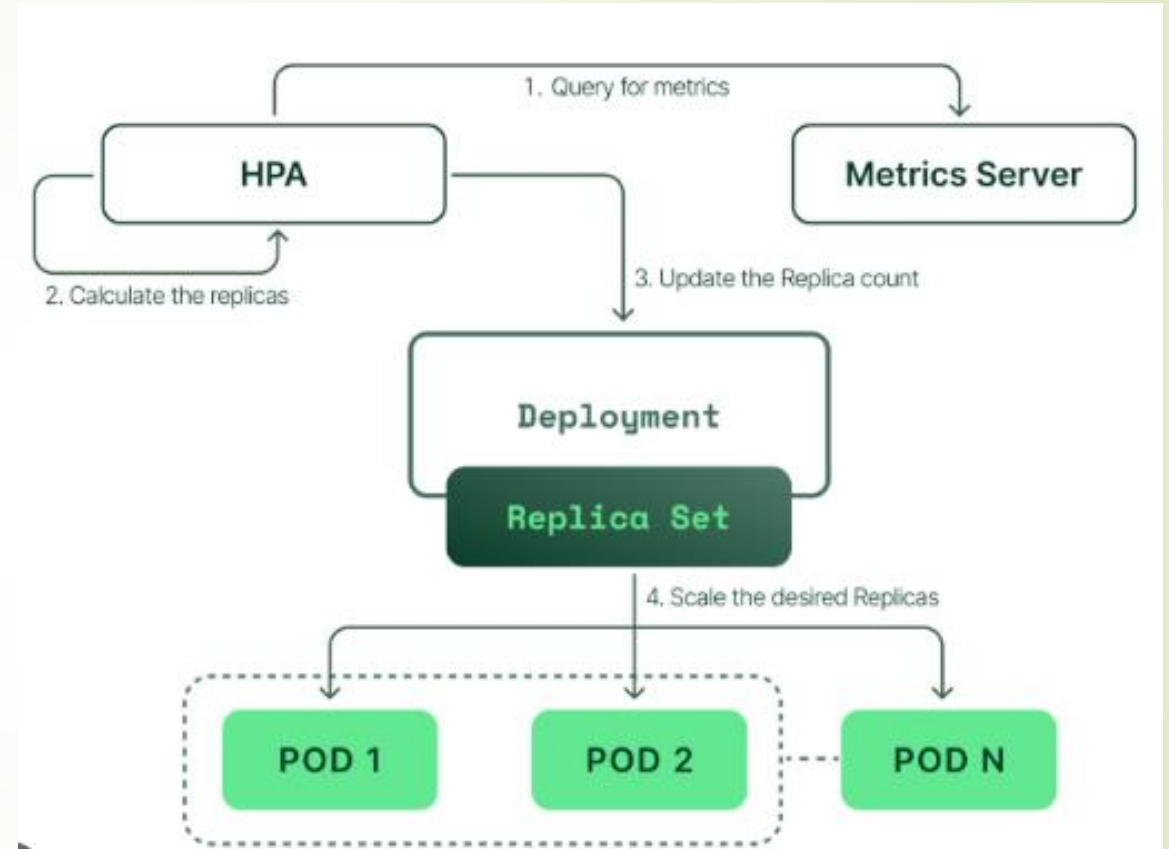


# RESOURCEQUOTA

- apiVersion: v1
- kind: Pod
- metadata:
  - name: default-cpu-demo-3
- spec:
  - containers:
    - - name: default-cpu-demo-3-ctr
    - image: nginx
    - resources:
      - requests:
        - cpu: "0.75"

# Horizontal POD Scaling

- Kubernetes has the capability to automatically scale PODS based on observed CPU Utilization which is horizontal POD scaling.
- Scaling can be done only for scalable objects like controller , Replicaset or deployment.
- HPA is implemented as a K8S API resources and a controller.
- The Controller periodically adjust the number of replicas in a replication controller or deployment to match the observed average CPU utilisation to target specified by user





# Horizontal POD Scaling

- For per POD resource metrics (like CPU), the controller fetches the metrics from resource metrics API for each POD targeted by HPA.
- Then if target utilization value is specified by user, the controller calculates the utilization value as % of equivalent resource request on each container in each POD
- `$ kubectl autoscale deployment mydeploy --cpu-percent=20 --min=1 --max=10`
- Cool down period to wait before another downscale operation can be performed is controlled by `--horizontal-pod-autoscaler-downstabilisation` flag
- Default value 5 minutes

# Jobs ,Init& Pod Life Cycle

- Jobs are object for specific purpose.
- We have replicaset ,Deployment ,Daemonsets ,Statefullsets ,they all share one common property they ensure ,their pods are always running ,if it fails , the controller restarts ,it recreates it to another node to make sure the application the pods is hosting keeps running.
- If we don't want pod to get recreated once purpose done , JOB object ensures it doesn't get created.
- Use Cases:
  - 1→Taking backup of a DB
  - 2→Running batch processes.
  - 3→Run a task at a scheduled interval
  - 4→Log Rotation
- Job does not get deleted on its own , we have to delete it.

# Jobs-Lab

- **apiVersion: batch/v1**
- **kind: Job**
- **metadata:**
  - **name: testjob**
- **spec:**
  - **template:**
    - **metadata:**
      - **name: testjob**
    - **spec:**
      - **containers:**
        - **- name: counter**
        - **image: centos:7**
        - **command: ["bin/bash", "-c", "echo Hi Ashok; sleep 5"]**
      - **restartPolicy: Never**

# Jobs-Lab

- `apiVersion: batch/v1`
- `kind: Job`
- `metadata:`
  - `name: testjob`
- `spec:`
  - `parallelism: 5` `# Runs for pods in parallel`
  - `activeDeadlineSeconds: 10` `# Timesout after 30 sec`
  - `template:`
    - `metadata:`
      - `name: testjob`
    - `spec:`
      - `containers:`
        - `- name: counter`
        - `image: centos:7`
        - `command: ["bin/bash", "-c", "echo Technical-Guftgu; sleep 20"]`
      - `restartPolicy: Never`



# *The Cron-Job Pattern*

- `apiVersion: batch/v1beta1`
- `kind: CronJob`
- `metadata:`
  - `name: Ashok`
- `spec:`
  - `schedule: "* * * * *`
  - `jobTemplate:`
    - `spec:`
      - `template:`
        - `spec:`
          - `containers:`
            - `- image: ubuntu`
            - `name: Ashok`
            - `command: ["/bin/bash", "-c", "echo Hi Ashok; sleep 5"]`
          - `restartPolicy: Never`



# *Init Container*

- Initialising Container.
- Init Containers are specialised containers that run before app containers in pod
- Init containers are always run to completion.
- If a pod's init container fails ,Kubernetes repeatedly restarts the pod until the init container succeeds.
- Init container does not support readiness probe.
- **Use-Cases**
  - 1→Seeding a database
  - 2→Delaying app launch until dependencies available
  - 3→clone a git repository into a volume
  - 4→Generate configuration file dynamically



# Pod Lifecycle

➤ Pending→Running→Succeeded→Failed→Completed→Unknown

➤ **Pending**→

- (A)-The Pod has been accepted by the k8's system but its not running till master stores info in etcd and asks it to allocate node and download image,e.g ubuntu
- (B)-One or more image is still downloading.
- (C)-If Pod cannot be scheduled because of resource constraints.

➤ **Running**→

- (A)-The Pod has been bound to node.
- (B)-All of the containers have been created.
- (C)-At least one container is still running or is in the process of starting or restarting

➤ **Succeeded**→

- (A)-All Containers have terminated in success and will not be started.

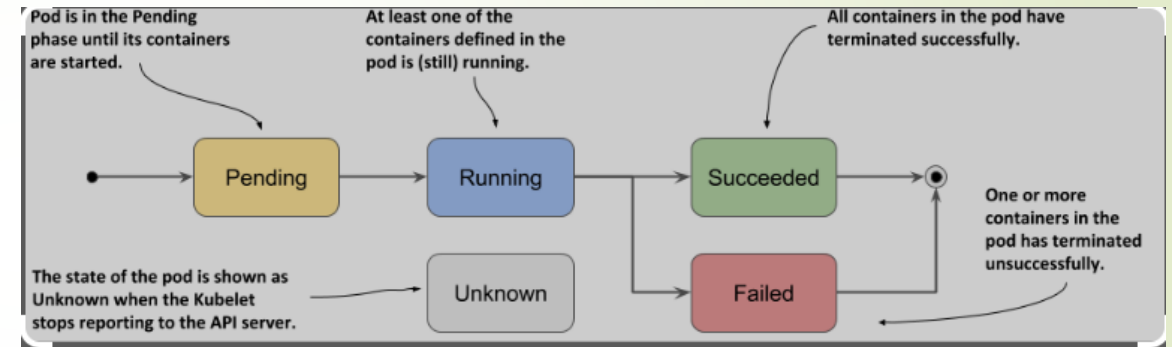
# Pod Lifecycle

- **Failed**→ All containers have terminated or atleast one container has terminated in failure.

(B)-The container either exited with non-zero status or was terminated by system.

**Unknown**→ (A)State of POD cannot be obtained possibly due to error in Network or communicating host in the pod.

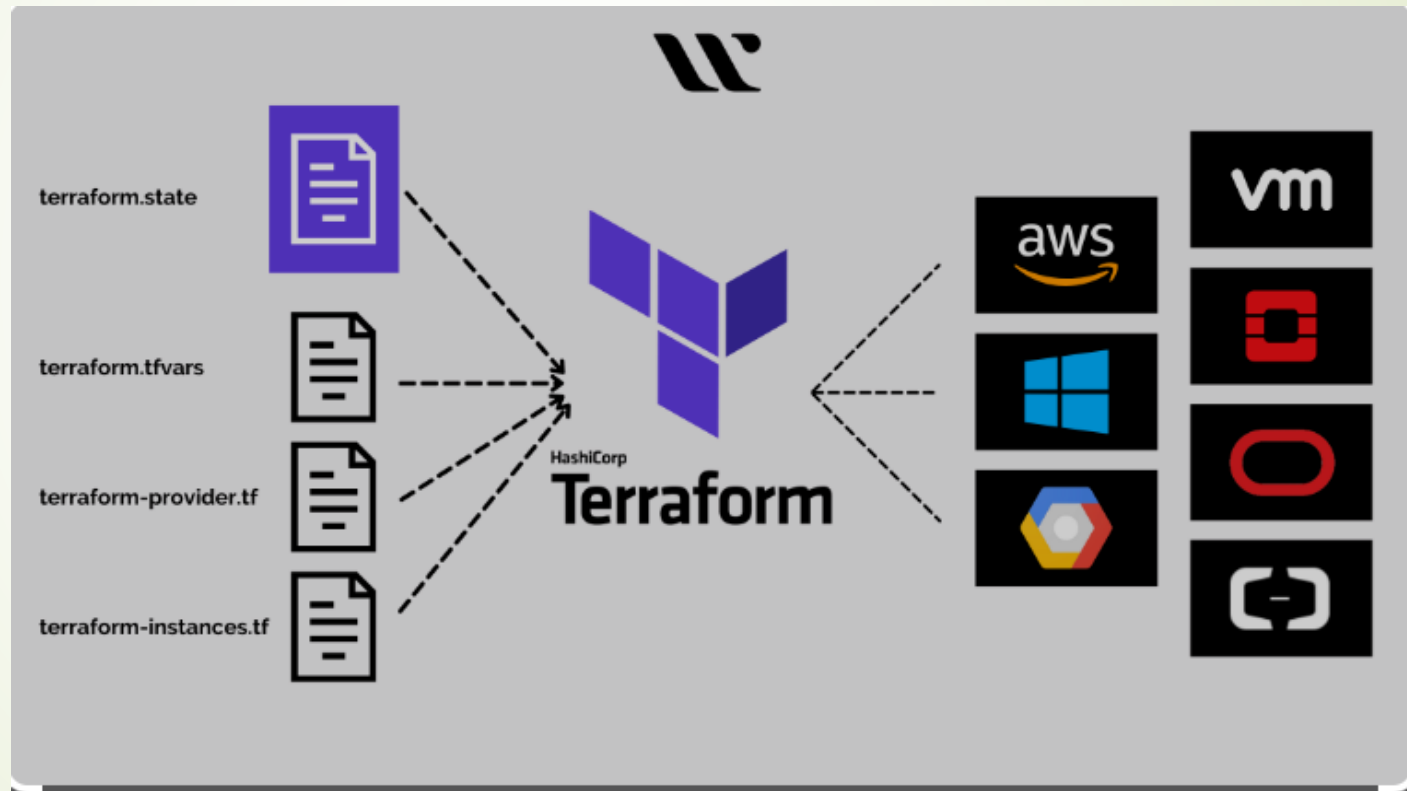
**Completed**→ The Pod has run to completion as there is nothing to keep it running e.g. Completed jobs.



# POD-Conditions

- The Pod has Pod Status which is an array of- Pod Conditions through which Pod has passed or has not passed
- Using “kubectl describe pod Podname”, we can get pod conditions.
- **These are possible types:-**
- **PodScheduled:-** The Pod has been scheduled to a node.
- **Ready:-** The Pod is able to serve request and will be added to load balancing pools of all matching Services.
- **Initialized:-** All Init Containers have started successfully
- **Unscheduled:-** The Scheduler cannot schedule the Pod right now e.g. due to lack of resources or other constraints.
- **Container-Ready:-** All Containers in POD are ready

# Terraform



# Terraform

- Terraform is an IAC tool for Cloud Infrastructure Automation.
- Infrastructure refers to Servers, Storage, Router, Switches, Firewall.
- In Typical VM;EC2 instance, VPC, Subnet all are Infrastructure.
- In 2011,AWS introduced CloudFormation, which is an automation technique which can make 100 EC2 Instances in minutes. But it will not work in Azure, Google Cloud.
- Similarly, For Azure we have ARM Template, for GCP-Deployment Manager
- In 2014,Terraform was introduced.
- Terraform means Extra Terrestrial, It makes planet like earth.
- In 2017,Microsoft Azure tied up with Terraform.
- Nobody wants to be vendor specific for Automation, so Terraform,to do automation on any cloud.

# Terraform

- Infrastructure As A code(IAC) means managing and provisioning of Infrastructure through code instead of manual process.
- There are two ways to approach IAC.
- **Imperative Approach**:-It defines specific commands needed to achieve the desired configuration and those commands needed to be executed in specific order.
- For example ;To create EC2 instance in AWS, we have to create VPC ,then subnet then IGW then RT else EC2 will not be created ,this is imperative Approach.
- **Declarative Approach**:-It defines the desired state of system, including what resources you need and any properties, they should have and IAC Tool will configure that for you.
- It is important to understand difference between Configuration Management Tool and IAC Tool.
- Ansible, Chef, Puppet are configuration management tool which means they are primarily designed to install and manage software on existing servers.
- Terraform and CloudFormation are IAC Tool to provision servers, once created, Terraform makes API Call to Ansible to install Apps





# Terraform

- **Benefits:-**
- It can manage almost all Cloud providers and can manage infrastructure on all clouds.
- Terraform has simple language called HashiCorp Language.
- Easy to integrate with Configuration Management Tools.
- It is easily extensible with Pluggins.
- It is free.
- Terraform keeps track of real infrastructure in state filr.
- We can reuse code

# Terraform-Commands and Execution

## ► Providers(AWS/AZURE/GCP/IBM/Oracle)→

(1) A Provider is responsible for understanding API interactions and exposing resources.

(2) If an API is available , we can create a provider.

(3) A provider uses a plugin.

(4) In order to make provider available on Terraform we need Terraform INIT command ,this downloads any plugin , we need for our providers.

(5) Below are different Commands for Terraform

**TF INIT**

**TF VALIDATE**

**TF PLAN**

**TF APPLY**

**TF DESTROY**

# Terraform-Commands and Execution

## ➤ Terraform INIT→

- A Terraform INIT command is used to initialize a working directory containing Terraform Configuration Files.
- It is safe to run this command multiple times.
- This command will never delete our existing Configuration or state.
- During INIT ,the root configuration directory is consulted for backend configuration and chosen backend is initialized- using given configuration settings.

## ➤ Terraform VALIDATE→

- It validates configuration files.
- It Verifies whether configuration file is syntactically valid and internally consistent.
- It is useful in general verification of reusable modules including correctness of attribute name and value type.
- It does not access remote services such as remote state ,provider's API



# Terraform-Commands and Execution

- **Terraform PLAN→**

- It is used to create an execution plan.

- Terraform performs a refresh unless explicitly disabled and then determines what actions are necessary to achieve the desired state specified in the configuration files.

- **Terraform APPLY→**

- It is used to apply the changes required to reach the desired state of the configuration or pre-determined set of actions generated by Terraform plan execution plan.

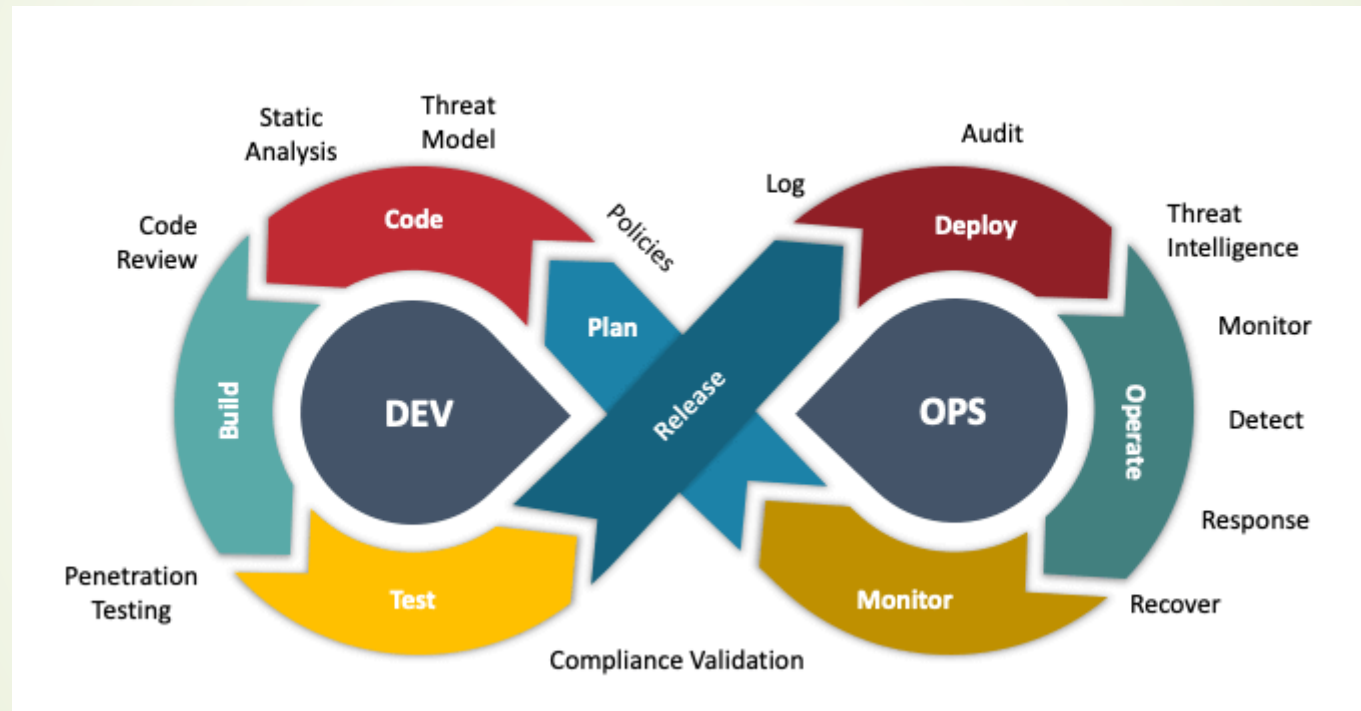
- **Terraform DESTROY→**

- Terraform Destroy is used to destroy the Terraform infrastructure

# HashiCorp Syntax

- HCL syntax is basic and should be readable like other scripting languages.
- It has three parts
- **(1)-Blocks-**
  - ```
resource "aws_instance" "example" {
```
  - ```
  ami = "abc123"
```
  - ```
  network_interface {
```
  - ```
    # ...
```
  - ```
  }
```
  - ```
}
```
  - **A block has a type (resource in this example). Each block type defines how many labels must follow the type keyword.**
  - **The resource block type expects two labels, which are aws\_instance and example in the example above.**
  - **After the block type keyword and any labels, the block body is delimited by the { and } characters. Within the block body, further arguments and blocks may be nested, creating a hierarchy of blocks and their associated arguments.**
- **(2)Arguments-**
  - Arguments are abstraction that enable IT admins to assign values to descriptive names which can represent or compute values.
  - They can be simple such as string or numeric values or more complicated such as arithmetic or logical expressions.
  - Key-value pair `image_id = "abc123"`
- **(3)Expressions-**
  - Expressions either represent or compute values.
  - They can be simple such as string or numeric values or more complicated such as arithmetic or logical expressions.

# DevSecOps-Introduction







# DevSecOps

- DevSecOps is made up of three key concepts: Development, Security and operations.
- Security has become bottleneck.
- DevSecOps fixes this problem.
- DevSecOps approach integrates security practices into the entire SDLC, right from the planning and coding stages to testing, deployment and ongoing operations.
- One of the core principle is shifting security to left.
- This will enable developers to identify and address potential security Vulnerabilities as they write code, this way we can catch security issues early reducing the risk in final product.